A Survey of AUV and Robot Simulators for Multi-Vehicle Operations

Daniel Cook*, Andrew Vardy† and Ron Lewis*
*Marine Environmental Research Laboratory for Intelligent Vehicles
†Department of Computer Science / Department of Electrical and Computer Engineering
Memorial University of Newfoundland
St. John's, Canada
{dac456,av,ron}@mun.ca

***Abstract*—This paper presents a survey of a selection of currently available simulation software for robots and unmanned vehicles. In particular, the simulators selected are reviewed for their suitability for the simulation of Autonomous Underwater Vehicles (AUVs), as well as their suitability for the simulation of multi-vehicle operations. The criteria for selection are based on the following features: sufficient physical fidelity to allow modelling of manipulators and end effectors; a programmatic interface, via scripting or middleware; modelling of optical and/or acoustic sensors; adequate documentation; previous use in academic research. A subset of the selected simulators are reviewed in greater detail; these are UWSim, MORSE, and Gazebo. This subset of simulators allow virtual sensors to be simulated, such as GPS, sonar, and multibeam sonar making them suitable for the design and simulation of navigation and mission planning algorithms. We conclude that simulation for underwater vehicles remains a niche problem, but with some additional effort researchers wishing to simulate such vehicles may do so, basing their work on existing software.**

I. Introduction

In the increasingly complex domain of robotics and unmanned vehicles, simulation is an important research and development tool. In particular to unmanned vehicles, testing newly devised control algorithms on a production vehicle may prove to be costly and potentially dangerous should the algorithm fail, or the vehicle not behave as expected or predicted. In the case of underwater vehicles, failed experiments in vehicle control may result in an unrecoverable vehicle, and as a consequence, considerable financial losses.

Simulation presents an opportunity to evaluate algorithms and control schemes in a virtual environment, avoiding altogether the risk involved in real-world experimentation. More exotic and untested algorithms (such as in multi-vehicle operations) may be entirely unfeasible to test in reality, without first verifying their correctness in a simulation.

A rather wide variety of simulation software has been developed with various aims; the most common are general purpose simulators that are intended to be flexible enough to simulate any environment and any type of robot or vehicle. Few however are used or tested with AUV simulation, and fewer still cite underwater vehicles as their primary purpose.

Surveys of vehicle and robot simulators have been conducted previously. A survey by Craighead et al. [1] of unmanned vehicle simulators included a wide range of vehicle types, and evaluated against a small set of criteria. Work by Matsebe et al. [2] reviewed simulators that were designed specifically for the simulation of AUVs, however many of the simulators reviewed are internal academic projects unavailable for public use.

This paper will review and evaluate a number of presently available simulators for unmanned vehicles and robots, and in particular, determine their suitability for simulating underwater vehicles. A simulator's capacity to simulate multi-vehicle operations will also be considered.

The remainder of this paper is organized as follows: Section II will provide a background on various topics that are common to the discussion of all simulation software. Section III will establish the criteria against which the simulators are evaluated. Section IV will provide a detailed description and review of three simulators deemed to be the best candidates found for our own research. Section V will provide a brief description and review of other potential candidates, that are not currently considered for our own research. Section VI will briefly discuss the use of game engines in the domain of robot and vehicle simulation. Section VII will offer a general discussion of the simulators and software presented in this paper. Section VIII will summarize the findings of this paper.

II. Prerequisites

In a general sense, a computer simulation is the use of one or more algorithms to model a real-world system or phenomenon. It does not necessarily have to provide a real-time visual output or allow user interaction while the simulation is performed. These sorts of simulations (offline and without interaction) are often applied to complex mathematical models, such as in physics or meteorology. In the case of robotics simulation, it is more helpful to view the simulation in real-time using either a 2D or 3D visual representation. Computing power found in modern computers allows high-fidelity physics to drive the simulation in real-time.

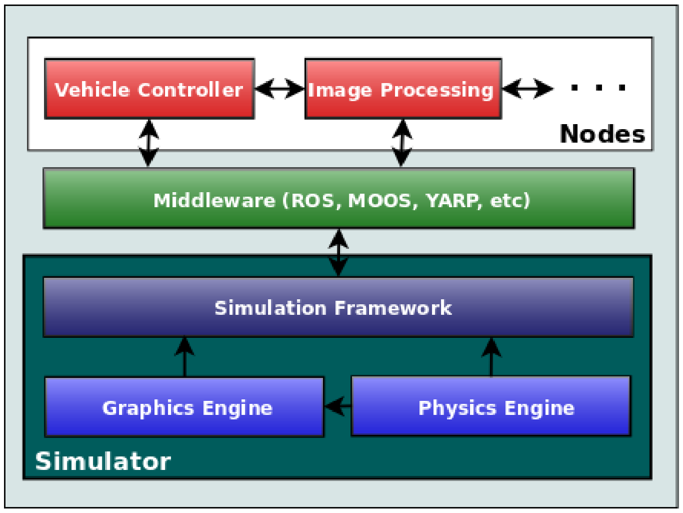


Fig. 1: Typical software architecture of a simulator.

Many of the simulators studied in this paper can be thought of as having three parts; a *rendering engine*, a *physics engine*, and a *simulation framework*. A rendering engine is software (usually a library) that provides the user with high-level rendering functions, such as mesh drawing, scene graph, camera, etc. It is essentially a framework for drawing 2D or 3D images to the screen, typically in real-time. Examples of open-source rendering engines are Ogre3D [3] and Irrlicht [4]. A physics engine is similar, in that it provides the user with a high-level framework. Rather than rendering, it simulates physical interactions of defined objects in the scene. When combined with a rendering engine, this physical simulation is used to update the visual representation of an object in the rendering engine, frame-by-frame. Examples of open-source physics engines include Bullet [5] and Open Dynamics Engine (ODE) [6]. In this paper, a simulation framework is considered to be the software that binds both a rendering engine and physics engine together into usable software that also provides facilities pertinent to robotics simulation such as sensors and interfaces to *robotics middleware*. Figure 1 depicts a block diagram of the typical architecture of a simulator.

Robotics middleware is typically treated as a high-level software layer that couples low-level control subsystems of a robot. The Robot Operating System (ROS) [7] for example allows the creation of *nodes*, each of which may be a controller for a particular device. ROS allows these nodes to communicate through a common message-passing interface often without prior knowledge of each other. In the case of simulation, treating the simulator as a node separate from a control node offers several advantages. Control algorithms can be written separately from the simulation software avoiding modification of the simulator itself. Different control algorithms under consideration may be written into separate nodes, allowing them to be swapped and interchanged with ease. Additionally, robots and vehicles that use middleware on-board may also interface with the simulator allowing hardware-in-the-loop simulation.

III. Evaluation Criteria

A set of criteria is established to facilitate the comparison and evaluation of simulation software.

- *Physical fidelity* - Here, the physical fidelity of a simulation will refer to the capacity for the simulator to simulate robotic arms and end effectors, and use these effectors to manipulate the surrounding environment in some way, whether by pushing objects, picking up objects, grasping objects, etc. This manipulation must be physically accurate through simulated forces and collision.
- *Programmatic interface* - A programmatic interface refers to any mechanism whereby the simulation software may be controlled by an external program or script without the need to modify the core source code of the simulation software. An example may be a simulator that allows communication from a ROS node or allows scripting by way of a Python (or other language) interpreter.
- *Sensor modelling* - Sensor modelling refers to the capacity of the simulation software to simulate optical or acoustic sensors. Typically by using virtual cameras attached to the vehicle and processing the image to simulate, for example, sonar imagery.
- *Adequate documentation* - Adequacy of documentation is an admittedly subjective criterion, but for the purposes of this survey documentation will be considered adequate if it is up-to-date with the latest release of the software, and covers all major components of the software.
- *Previous use in research* - If a simulator has been used in previous research it should be favoured more highly than those that have not, since we expect that the software present in previous research is more rigorously tested.

It should be noted that *physical fidelity* as described here differs from the definition established in previous work by Craighead, et al [1]. In their survey of unmanned vehicle simulators, the authors propose *physical fidelity* and *functional fidelity* as separate evaluation criteria. The authors describe functional fidelity as the degree to which physical forces acting on the vehicle are simulated. This is partly analogous to our definition of *physical* fidelity, though we also consider the fidelity of robotic arms and end effectors in the simulated environment.

Visual fidelity, while considered, is not prioritized in our evaluation. Much of the visual fidelity observed in any simulator is dependent on the assets (such as textures and models) that are packaged with the simulator; a simulator that uses high-resolution textures and models by default may be perceived to have higher visual fidelity. However, in simulators that are packaged with less ideal textures and models, the assets can be replaced should the user desire. To this end, the rendering capabilities of most simulators are the same once user-replaceable assets are ignored. Where simulators offer built-in effects such as reflection and refraction on a water surface it will be noted.

The ability to extend a simulator either through plugins or modifying core source code is also considered but not prioritized in our evaluation. The ability to modify particular software for individual requirements is desirable, however the the focus of the analysis should be on the features provided in the default build of each simulator.

IV. Simulator Review

Simulators covered in this section were selected for their suitability for our own research. At a minimum, they all provide compatibility with ROS and are all open-source software. They are all also built on modern rendering and physics engines, which themselves are actively maintained and have large support communities. Finally, these simulators provide some mechanism for defining a custom scene (underwater in our case), and defining multiple agents for the simulation. Table I provides an overview comparison of each simulator discussed in this section.

A. UWSim

UWSim [8] is a project of the IRS Lab at *Universitat Jaume I* in Castellón, Spain. It is an open-source project designed specifically for underwater vehicles; it is built upon the OpenSceneGraph rendering engine and the Bullet physics engine.

In terms of physical fidelity, the Bullet physics library underlying UWSim provides accurate collision detection between convex meshes and shapes, as well as rigid body dynamics. Robotic arms attached to a vehicle can also interact physically with rigid bodies; they may conceivably be used to push, scoop up, or grab objects [9] [10].

UWSim provides an interface to external software through ROS. By creating one or more ROS nodes, topics established by UWSim can be used to simulate vehicle control or to receive sensor data captured by a vehicle. For example, sonar data is broadcast as an image message to ROS, which may be captured by a node and processed with OpenCV and used for vehicle control. Multiple virtual cameras may be attached to a vehicle to simulate stereo vision for which ROS provides support using `stereo_image_proc`. It is also possible for UWSim to interface with Matlab using the `ipc_bridge` ROS package.

A variety of sensors and devices may be simulated in UWSim including multibeam, contact sensor, force sensor, pressure sensor, Global Positioning System (GPS), Inertial Measurement Unit (IMU), and sonar. The majority of these devices are broadcast over the ROS interface using standard ROS message types (some however, such as the pressure sensor, use a custom message type). Sonar is simulated using a *range camera*, which essentially captures the depth buffer of the renderer, and passes it as a message to ROS.

Documentation for UWSim is sparse, though that does not necessarily imply that the software is difficult to use. Users with previous experience using ROS software will likely be able to rely on inspecting the source code or ROS messages of UWSim. The source code for several example ROS nodes is available as well. The UWSim wiki page contains articles on installing UWSim, as well as on how to configure and create simulation scenes.

UWSim has been used in academic research previously (two examples of which are [11] and [12]), indicating that it has been vetted for suitability.

Overall, UWSim is an excellent fit for the simulation of underwater vehicles. It is very similar in functionality and features to other simulators reviewed later in this paper; most allow an environment to be rendered, and robots or vehicles to be defined with or without actuators and end effectors. UWSim is set apart however, by its default use of osgOcean [13], a plugin for OpenSceneGraph that allows highly realistic visual simulation of an underwater environment and the water surface (waves, reflection/refraction, etc). It *supports* multiple AUVs, though creating a simulation that actually contains many vehicles and/or objects may be tedious; simulation scenes are manually defined in an XML file where environment settings, vehicles, objects, and ROS interfaces are defined. If many vehicles or objects are required this may become unwieldy to manage, especially considering that ROS interfaces and topics must also be defined manually for each vehicle. UWSim also does not provide a convenient way to extend the software (i.e. to add new sensor types); any modifications of this kind must be written into the core source code.

B. MORSE

MORSE [14] is a general purpose academic robot simulator, originally developed at and supported by *Laboratoire d'Analyse et d'Architecture des Systèmes* at the *University of Toulouse*, France. It is built upon on the Blender Game Engine and the Bullet physics engine.

Like UWSim, MORSE's physical fidelity is largely bound by the fidelity possible with the Bullet engine. MORSE provides actuator components which "*Produce actions upon the associated robots or components*". This means actuator components may either affect vehicle properties such as linear/angular velocity or the position of an attached robotic arm. Attached arms and grippers behave similarly to those in UWSim in that the attached components may interact physically with the world around them, though fine grasping of objects is generally not possible [15].

MORSE boasts support for four robotics middlewares; these are ROS, Yet Another Robot Platform (YARP)[16], Pocolibs[17], and MOOS[18]. It also supports a socket-based protocol, allowing MORSE to be integrated with unsupported middleware or tools.

A relatively large number of sensor components are shipped with MORSE including: accelerometer, battery sensor, collision sensor, depth camera, GPS, odometry sensor, laser scanner, etc. In the latest release of MORSE (as of this writing), the *depth camera* component generates a 3D point cloud from the perspective of the camera, in contrast to the 2D depth buffer captured by UWSim's *range camera*. A *generic camera* component is also offered which could conceivably be used to return a depth buffer in the same manner as UWSim, if necessary. MORSE also provides a convenient facility for

	Primary Criteria					**Secondary Criteria**	
	Physical Fidelity	Programmatic Interface	Sensor Modelling	Adequate Documentation	Previous Use in Research	Visual Fidelity	Extensibility
UWSim	High	Single	Yes	Low	Yes	High	Low‡
MORSE	High	Multiple	Yes†	High	Yes	Medium	High
Gazebo	High	Single	Yes†	High	Yes	Medium	High
†Extensible. ‡Extensible by modifying core source code.							

TABLE I: An overview comparison of simulators covered in Section IV.

creating custom components (including sensors and actuators), should functionality be required that is not included by default with the package.

Documentation for MORSE [19] appears thorough. It offers several tutorials aimed at multiple levels of proficiency, as well as documentation on the provided robots and components, integrating MORSE into external software, extending MORSE, building simulation environments, and advanced usage (such as multi-agent operations).

MORSE has been used in previous academic research including [20] and [21].

MORSE is functionally similar to UWSim, however it offers several advantages. Creating a simulation with MORSE is much more straight-forward using a Python API known as *Builder* [22]. Furthermore, a workspace for the simulation can be created with a single command (`morse create <simulation name>`). While builder scripts are still 'manual' in the same sense as UWSim XML, they are much more convenient for creating multiple vehicles or objects (since they may be generated in a loop rather than defined individually). MORSE's integration with the Blender Game Engine also allows various robot/vehicle properties to be modified from within Blender itself. As a general purpose simulator, it does not offer a default environment for underwater vehicles as in the case of UWSim. It does ship with a submarine model and two underwater environments however (see figure 2 for an example), which are accessible through the Builder API; custom underwater environments and vehicles may be created relatively trivially [23].

MORSE's disadvantage from a research standpoint is its reliance on Blender and the Blender Game Engine. Those who are unfamiliar with Blender and its interface may find themselves with a learning curve that could be avoided by choosing a different simulator. In particular to underwater simulation, creating an underwater environment more detailed than the one packaged with MORSE may require learning a great deal about using Blender before ever doing work that pertains directly to simulation. Creating a water simulation that contains the visual fidelity found in UWSim and osgOcean requires a knowledge of both programming for the Blender Game Engine, as well as shader programming. For multi-agent simulation, MORSE's documentation states that "MORSE is able to handle dozen of robots in a single environment as long as cameras are not simulated (because of bandwidth limitation)" [15]. It is reasonable to assume that this bandwidth limitation would be found in most simulators due to the computational requirements of render-to-texture functionality used in simulated cameras.

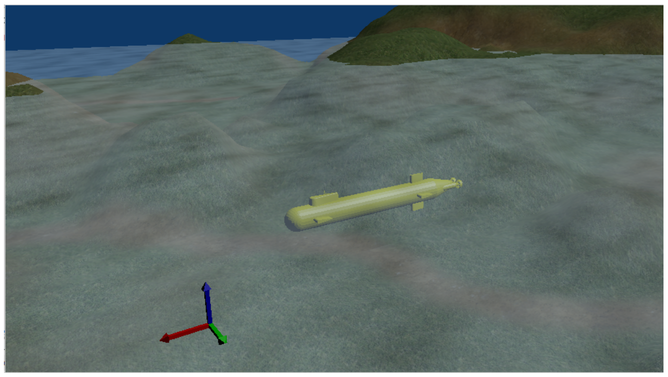


Fig. 2: MORSE with a water scene and submarine robot.

C. Gazebo

Gazebo [24] is a general purpose robot simulator originally developed at the *University of Southern California* and currently stewarded by the *Open Source Robotics Foundation*. It is built upon the Ogre3D rendering engine and has support for several physics engines including ODE, Bullet, Simbody, and DART. It is compiled against ODE by default.

The physical fidelity provided by Gazebo will be largely dependent on the physics engine the simulator is compiled against. The supported engines should be similar in many cases [25], but may differ in suitability for specific cases; ODE and Bullet perform well in simulating cluttered environments, while DART and Simbody may be more accurate in simulating humanoid robots [26].

Gazebo has historically been tightly integrated with ROS and the same holds true today in the form of the `gazebo_ros_pkgs` package set which provide ROS wrappers around stand-alone Gazebo. Third-party middleware wrappers for Gazebo are possible however, such as the YARP plugin [27].

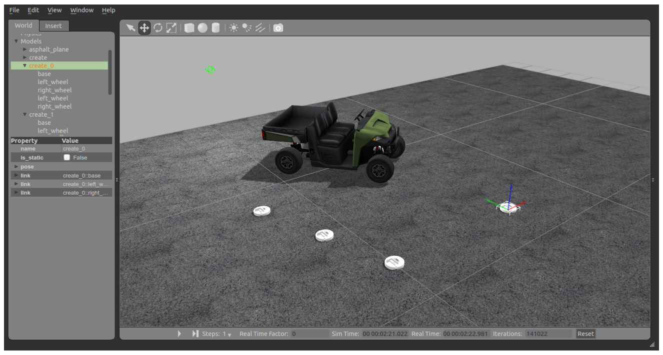
A variety of sensors are readily available for Gazebo including camera, multicamera (stereo camera), laser scanner, IMU, contact sensor, etc. An API is also provided to allow creating new sensors as plugins for Gazebo.

Documentation for Gazebo is similar to that of MORSE and provides a large number of tutorials at three different skill levels. Tutorials also indicate which versions of Gazebo they are compatible with.

Gazebo has been featured extensively in previous academic research (see [28] [29] [30] for examples), and has also been used to run the Virtual Robotics Challenge (a component of the DARPA Robotics Challenge).

Gazebo shares many of the same functionalities of both UWSim and MORSE; a mechanism is provided for defining environments and vehicles/robots, and like MORSE these environments are not underwater by default. Gazebo does provide a graphical world editor which can be used to place objects and robots from either the local system or an online database (see figure 3). The editor also allows placing primitive shapes, as well as lights. A unique feature of Gazebo (as of version 3.0) that may prove useful in the domain of underwater simulation is its support for Digital Elevation Model (DEM) data [31]. Using digital elevation data, one may import a patch of ocean floor into Gazebo conducting simulated experiments in the same geographical location and terrain in which real experiments may take place in the future.

While Gazebo provides a suitable environment for land-based robots and vehicles, it is more difficult to adapt it to an underwater environment. There is no built-in way to create a water plane or underwater visual effects as there is in UWSim or MORSE. It may be possible to create a plugin for this functionality, but it has not been attempted as of this writing to our knowledge. In terms of multi-vehicle operations, it has been suggested that Gazebo is too computationally demanding to simulate many vehicles in real-time and has an upper limit of ten robots [32].



Property	Value
name	create_0
is_static	False
pose	
link	create_0::base
link	create_0::left_w...
link	create_0::right_...

Steps: 1 Real Time Factor: 0 Sim Time: 00 00:02:21.022 Real Time: 00 00:02:22.981 Iterations: 141022

Fig. 3: Gazebo with several instances of the iRobot Create, as well as an off-road vehicle.

V. Other Simulators

Several simulators exist beyond those described in the previous section, but for various reasons have not been considered for use in our own research are therefore not reviewed here in great detail.

A. SwarmSimX

SwarmSimX (SSX) [33] is a swarm simulation platform developed by Johannes Lächele at the *Max Planck Institute for Biological Cybernetics* in Tübingen, Baden-Württemberg, Germany. It is built upon the Ogre3D rendering engine and the PhysX physics engine, though due to the modular architecture of the simulator it should be possible to replace these underlying engines without affecting the high-level simulation framework. SSX is intended to simulate dozens of robots, making it ideal for simulating multi-vehicle operations. There is no evidence to suggest it has been previously used in the simulation of underwater vehicles, but like most simulators, it should be achievable by writing additional plugins or modifying the default simulation environment. SSX also provides a plugin that allows support for ROS.

Despite being used in previous academic research [34], we have been unable to see SSX run correctly on the workstation we tested it on (Ubuntu 13.04, Intel Xeon E31335 x4, 16GB RAM, nVidia Quadro 2000); the software itself will start, but the renderer will stop responding. While the simulation framework is open-source, the renderer portion is not, making any bugs or incompatibilities in this portion of the software difficult to fix.

B. V-REP

V-REP [35] [36] is a general purpose robot simulator developed by Coppelia Robotics. Unlike other simulators reviewed, it is dual-licensed under both a commercial licence as well as a free educational license. The source code is also dual-licensed, under either a commercial licence or GNU GPL.

V-REP uses a custom rendering engine and provides support for three physics engines (Bullet, ODE, and Vortex) which are switchable at run-time. It allows simulations to be programmed using plugins, embedded scripts or ROS nodes; an API for C++ and Lua is provided also. Its default sensor support is somewhat limited, listing proximity sensors and vision sensors [37], however it is possible to extend sensor support via plugins.

There has been no known application of V-REP to the simulation of AUVs, though the simulator does make multi-agent operations trivial. The simulator does support underwater robots in general however, and simulating an underwater vehicle should be possible by creating an appropriate environment and possibly extending V-REP with plugins.

C. USARSim

USARSim is the Urban Search and Rescue Simulator [38]. It is based on the Unreal Engine [39], and is the platform used for the Rescue Simulation League of the RoboCup. As of this writing, USARSim appears to be transitioning from a platform based on Unreal Tournament 2004, to the more modern Unreal Development Kit (UDK). As a result, there is little current documentation available for USARSim itself, though UDK provides its own documentation for the engine and tools. Being based on a high-end game engine, a great deal of visual and physical fidelity should be possible.

D. SubSim

SubSim is a project of the *Robotics and Automation Lab* at *The University of Western Australia*. Aside from UWSim, it is the only known simulator designed specifically for underwater vehicles. The project does not appear to be actively developed any longer, though the simulator software is still available. The majority of the documentation is no longer active. It provides a plugin system, but no other methods of extension or programming (such as ROS) are available. It also only supports one vehicle and does not appear to support robotic arms or end effectors.

E. NetLogo

NetLogo is a multi-agent (swarm) simulator written by *Uri Wilensky* and developed at *The Center for Connected Learning at Northwestern University*. NetLogo includes both a 2D and 3D version, though the 3D application is labelled as experimental. Unlike most simulators covered in this paper, NetLogo does not provide a high level of physical fidelity; it is however well suited to prototyping multi-agent algorithms, and it has been previously used to simulate the agents in the CoCoRo project at the University of Gratz, Austria [32]. Depending on individual requirements, NetLogo may be useful in initial prototyping of an algorithm which could later be ported to another simulator offering higher visual and physical fidelity.

VI. Game Engines for Simulation

A common alternative to utilizing a dedicated simulator for vehicle and robot simulation, is to re-purpose an available game engine for simulation and research [40]. Simulations based on a game design toolkit are often marketed as "serious games" and several existing game engine developers now make a point of advertising this capability of their products. However, in general any available game engine could conceivably be used for the purpose of simulation.

A. Unity

Unity is a dual-licensed game engine by Unity Technologies [41]. A free version with a reduced feature set, as well as a fully-featured pro version are available. Unity also offers specific licensing options for those wishing to use Unity for serious games. For the purposes of 'internal' research use, it should be possible to accomplish any task in the free or pro version of the engine. While support for robotics middleware does not yet exist, it may be theoretically possible to provide support in the pro version through a plugin. Unity also provides support for scripting in C#, JavaScript, and Boo, which may be used in place of an external control node using robotics middleware.

B. Leadwerks

Leadwerks is a game engine by Leadwerks Software [42]. Unlike Unity, it provides no free version, but two commercial licences (Indie and Standard). Leadwerks is not currently marketed for serious games, but like Unity it should be possible to accomplish many simulation tasks within the standard release of the engine. There is currently no known support for robotics middleware, however Leadwerks provides a Lua scripting interface that may be used instead for vehicle simulation and control.

C. Unreal Engine

Unreal Engine [39] is a game engine by Epic Games [43]. The current iteration, Unreal Engine 4, is available for a monthly licence fee. The previous version (Unreal Development Kit) which is based on Unreal Engine 3 is still available for free and is the platform on which USARSim is built. While there is no known support for robotics middleware, a license for Unreal Engine provides the user with full source code access with which support for a middleware package such as ROS could potentially be added. Unreal also provides a scripting environment which could be used to control agents without the need to modify existing source code.

VII. Discussion

Previous sections describe a set of simulators, as well as a brief selection of game engines that may be adapted for simulation and research. The simulators covered in Section IV were selected in part due to their open-source licensing. While open-source software is often desirable in research (due to inherently low cost and source code availability), they are also exposed to some of the issues that often befall open-source projects. In particular, open-source software tends to be at a higher risk of becoming abandoned or unsupported. This is seen in some simulators listed in Section V; projects that began relatively recently are no longer actively supported or developed. Projects developed exclusively at academic institutions may become neglected as the priorities of researchers and developers shift to other areas.

Release of binary packages for some simulators also tends to fall behind in some cases. UWSim for example, which is highly dependent on ROS, has yet to provide a binary package for the latest release of ROS as of this writing (ROS Indigo); this requires building UWSim from source and adapting now out-of-date building instructions for the new release. Binary packages of MORSE are available only for Debian and Ubuntu Linux and packages for MORSE 1.2 (the latest as of this writing) only became available with Ubuntu 14.04.

This paper has compared several simulators that are themselves composed of smaller software packages for rendering and physics such as Ogre3D and Bullet. The option of building a simulator customized for one's needs using these libraries has not been considered. The primary differences found between simulators in Section IV are generally not found in rendering or physics capability, but in the framework and simulation-specific software built around these engines and libraries. These underlying libraries tend to be well documented and supported and it is conceivable that researchers with specific needs may wish to develop their own simulation environment using these libraries. The practicality of writing a custom simulator would be dependent on the specific needs of the user. Discussion on the effort such an endeavour would require is beyond the scope of this paper.

None of the simulators described in previous sections include simulation of hydrodynamic forces or ocean currents acting on the vehicle; any simulation is underwater only in a visual regard. It is up to the user to include an approximation of underwater vehicle physics in a control script, or to extend the simulator itself.

In general, the suitability of any simulation software will be dependent on individual needs and requirements and any qualitative analysis of software is inherently subjective in this regard.

VIII. Conclusion

We conclude based on the simulators evaluated in this paper that there are many options for the simulation of robots and autonomous vehicles *in general*, however the application of many of these simulators to autonomous underwater vehicles may require additional effort on the part of the researcher. The simulators reviewed in Section IV each provide their own strengths and weaknesses, but each may require additional effort for use depending on individual requirements. Given the thin line separating technologies used for simulation and those used for games, it may be desirable to use existing game development toolkits for simulation. With the availability of open-source engines for rendering and physics, it may also be more appropriate to build a simulator customized for individual requirements. In particular to multi-vehicle simulation, no simulators have been covered that are designed to be well-suited to multiple *underwater* vehicles. While several simulators claim support for multi-agent simulation, none consider the additional requirements of underwater vehicles. Typical AUV simulation requires the use of virtual cameras to simulate acoustic instruments such as sonar. If not properly managed, the overhead involved in render-to-texture for multiple virtual cameras per vehicle may bring the performance of a simulator below real-time.

Future development in AUV simulation should seek to add higher visual fidelity to simulation, and to increase physical fidelity both in terms of end effector-world interaction as well as in terms of simulated hydrodynamic forces and ocean currents acting on a vehicle.

Ultimately there are a number of options in simulators that may be used as a base for underwater vehicle simulation, but there is still more work required in development of these simulators to make them robust enough to conduct accurate experiments and simulations with AUVs.

IX. Acknowledgements

The work presented in this paper was funded by the Atlantic Canada Opportunities Agency, through an Atlantic Innovation Fund grant, under the Responsive AUV Localization and Mapping project (REALM), and Research and Development Corporation of Newfoundland and Labrador.

References

[1] J. Craighead, et al. ”A survey of commercial & open source unmanned vehicle simulators.” *Robotics and Automation, 2007 IEEE International Conference on.* IEEE, 2007.

[2] O. Matsebe, C. M. Kumile, and N.S. Tlale. ”A review of virtual simulators for autonomous underwater vehicles (auvs).” *NGCUV, Killaloe, Ireland* (2008).

[3] ”OGRE - Open Source 3D Graphics Engine”. Internet: http://www.ogre3d.org [August 12, 2014]

[4] ”Irrlicht engine - A free open source 3D engine”. Internet: http://irrlicht.sourceforge.net [August 12, 2014]

[5] ”Real-Time Physics Simulation”. Internet: http://bulletphysics.org [August 12, 2014]

[6] ”OpenDE homepage”. Internet: http://opende.sourceforge.net [August 12, 2014]

[7] M. Quigley, et al. ”ROS: an open-source Robot Operating System.” *ICRA workshop on open source software.* Vol. 3. No. 3.2. 2009.

[8] M. Prats, et al. ”An open source tool for simulation and supervision of underwater intervention missions.” *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on.* IEEE, 2012.

[9] ”Grasp execution within UWSim”. *YouTube*. Available: https://www.youtube.com/watch?v=3lSvb6Tgrbs [July 30, 2014]

[10] ”UWSim physics with Bullet”. *YouTube*. Available: https://www.youtube.com/watch?v=Zj6M0g6E3QE [July 30, 2014]

[11] P. Kormushev, and D. G. Caldwell. ”Towards improved AUV control through learning of periodic signals.” *Proc. MTS/IEEE Intl Conf. OCEANS*. 2013.

[12] A. Carrera, et al. ”Towards valve turning with an AUV using Learning by Demonstration.” *OCEANS-Bergen*, 2013 MTS/IEEE. IEEE, 2013.

[13] ”osgocean - An ocean rendering nodekit for OpenSceneGraph”. Internet: https://code.google.com/p/osgocean/ [August 12, 2014]

[14] G. Echeverria, et al. ”Modular open robots simulation engine: Morse.” *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011.

[15] ”What is MORSE?”. Internet: http://www.openrobots.org/morse/doc/latest/what_is_morse.html [July 30, 2014]

[16] Metta, Giorgio, Paul Fitzpatrick, and Lorenzo Natale. ”YARP: yet another robot platform.” *International Journal on Advanced Robotics Systems* 3.1 (2006): 43-48.

[17] ”Pocolibs”. Internet: http://pocolibs.openrobots.org, [July 29, 2013]

[18] Oxford Mobile Robotics Group. ”MOOS”. Internet: http://www.robots.ox.ac.uk/ mobile/MOOS/wiki/pmwiki.php/Main/HomePage, [July 29, 2014].

[19] ”The MORSE Simulator Documentation”. Internet: http://www.openrobots.org/morse/doc/stable/morse.html, [July 29, 2014]

[20] M. Karg, M. Sachenbacher, and A. Kirsch. ”Towards expectation-based failure recognition for human robot interaction.” *Proceedings of 22nd International Workshop on Principles of Diagnosis (DX-2011).* Vol. 15. 2011.

[21] L. Kunze, K. K. Doreswamy, and N. Hawes. ”Using Qualitative Spatial Relations for Indirect Object Search.” *IEEE International Conference on Robotics and Automation (ICRA)*. 2014.

[22] ”Builder Overview - The MORSE Simulator Documentation”. Internet: http://www.openrobots.org/morse/doc/latest/user/builder_overview.html [August 12, 2014]

[23] ”A Journey to a New Simulation”. Internet: http://www.openrobots.org/morse/doc/latest/user/advanced_tutorials/a_journey_to_a_new_simulation.html [August 12, 2014]

[24] N. Koenig, and A. Howard. ”Design and use paradigms for gazebo, an open-source multi-robot simulator.” *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on.* Vol. 3. IEEE, 2004.

[25] ”Gazebo Multi-Physics Engine Support”. *Vimeo*. Available: http://vimeo.com/84443645 [July 30, 2014]

[26] ”Gazebo supports 4 physics engines”. Internet: http://gazebosim.org/blog/gazebo-supports-4-physics-engines.html, [July 30, 2014]

[27] ”Gazebo plugin to interface GAZEBO with YARP”. Internet: https://github.com/robotology/gazebo-yarp-plugins, [July 30, 2014]

[28] D. J. Naffin, M. Akar, and G. S. Sukhatme. ”Lateral and longitudinal stability for decentralized formation control.” *Distributed Autonomous Robotic Systems 6*. Springer Japan, 2007. 443-452.

[29] I. Chen, et al. ”A simulation environment for OpenRTM-aist.” *System Integration, 2009. SII 2009. IEEE/SICE International Symposium on.* IEEE, 2009.

[30] I. Y. H. Chen, B. MacDonald, and B. Wunsche. ”Mixed reality simulation for mobile robots.” *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009.

[31] ”Digital Elevation Models”. Internet: http://gazebosim.org/tutorials?tut=dem&cat=build_world [August 14, 2014]

[32] M. Read, et al. ”Profiling Underwater Swarm Robotic Shoaling Performance using Simulation.” *Autonomous Robotic Systems (TAROS)* (2013).

[33] J. Lächele, et al. ”SwarmSimX: Real-time simulation environment for multi-robot systems.” *Simulation, Modeling, and Programming for Autonomous Robots.* Springer Berlin Heidelberg, 2012. 375-387.

[34] A. Franchi, P. Stegagno, and G. Oriolo. ”Decentralized Multi-Robot Target Encirclement in 3D Space.” *arXiv preprint arXiv:1307.7170* (2013).

[35] Coppelia Robotics. ”Coppelia Robotics v-rep: Create. Compose. Simulate. Any Robot”. Internet: http://www.coppeliarobotics.com/ [July 31, 2014]

[36] Coppelia Robotics. *PDF Document*. Available: www.coppeliarobotics.com/v-repOverviewPresentation.pdf [July 31, 2014]

[37] Coppelia Robotics. Internet: http://www.coppeliarobotics.com/features.html [July 31, 2014]

[38] S. Carpin, et al. ”USARSim: a robot simulator for research and education.” *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007.

[39] ”Unreal Engine Technology”. Internet: http://www.unrealengine.com [August 12, 2014]

[40] M. Lewis, and J. Jacobson. ”Game engines.” *Communications of the ACM* 45.1 (2002): 27.

[41] ”Unity - Game Engine”. Internet: http://unity3d.com/ [August 12, 2014]

[42] ”Leadwerks Game Engine”. Internet: http://www.leadwerks.com/ [August 12, 2014]

[43] ”Epic Games”. Internet: http://epicgames.com/ [August 12, 2014]