2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 7-12, 2012. Vilamoura, Algarve, Portugal

An Open Source Tool for Simulation and Supervision of Underwater Intervention Missions

Mario Prats, Javier Pérez, J. Javier Fernández and Pedro J. Sanz

***Abstract*— This paper presents UWSim: a new software tool for visualization and simulation of underwater robotic missions. The software visualizes an underwater virtual scenario that can be configured using standard modeling software. Controllable underwater vehicles, surface vessels and robotic manipulators, as well as simulated sensors, can be added to the scene and accessed externally through network interfaces. This allows to easily integrate the simulation and visualization tool with existing control architectures, thus allowing hardware-in-the-loop simulations (HIL). UWSim has been successfully used for simulating the logics of underwater intervention missions and for reproducing real missions from the captured logs. The software is offered as open source, thus filling a gap in the underwater robotics community, where commercial simulators oriented to ROV pilot training predominate.**

I. INTRODUCTION

Experimentation with underwater robots is normally very difficult due to the high number of resources required. A water tank –high enough for the systems to be tested– is normally needed, which requires significant space and maintenance. Another possibility is to access to open environments such as lakes or the sea, but this normally involves high costs and requires special logistics. In addition, the nature of the underwater environment makes it very difficult for researchers (operating in the surface) to observe the evolution of the running system. As a consequence, experimental validation of these systems is highly laborious.

In order to facilitate the development of underwater robots, it is of utmost importance to develop suitable simulators that allow to (i) develop and test the systems before they are deployed, and (ii) supervise a real underwater task where the developers do not have a direct view of the system.

Several virtual simulators for autonomous underwater vehicles have been developed in previous projects, see [1] for a review. Unfortunately, most of these simulators are either obsolete, or are specific to a given project and cannot be easily adapted to other different projects. In many cases, the software design makes it difficult to interface these systems with existing control architectures. In addition, access to the source code and collaborative development possibilities are normally very limited. Furthermore, support for underwater manipulators is not common in existing projects. There are also commercial simulators such as VortexSim [2], ROVsim [3], ROVolution [4] or DeepWorks [5]. However, these are normally addressed to ROV pilot training, are closed, and thus, difficult to integrate with custom control architectures.

M. Prats, J. Pérez, J.J. Fernández and P.J. Sanz are with Computer Science and Engineering Department, University of Jaume-I, 12071 Castellón, Spain mprats@uji.es

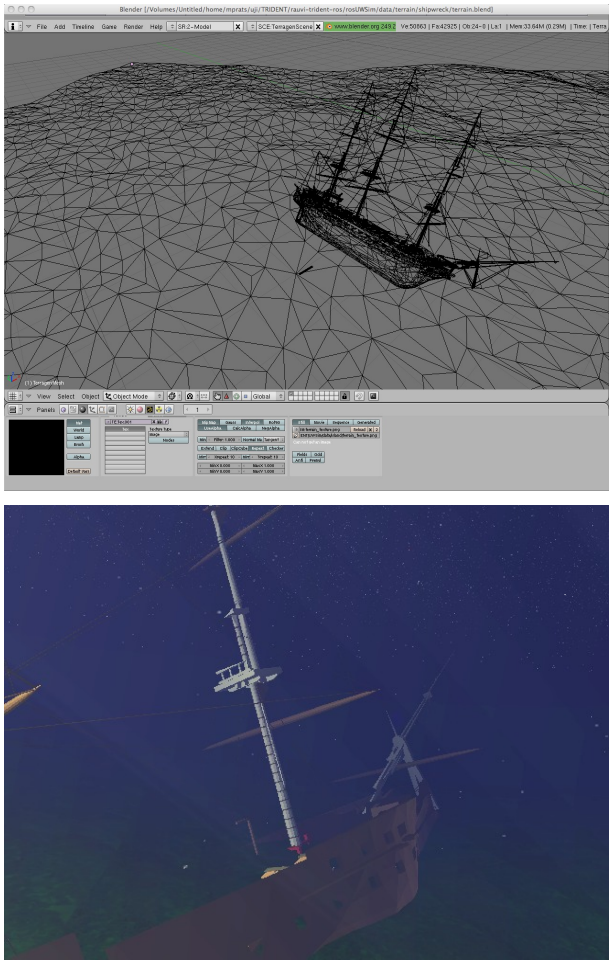

Fig. 1. (Top) A shipwreck scenario being modeled in Blender. (Bottom) The same scenario loaded in UWSim.

In this paper we present an open source underwater simulation tool that has been designed with the following main goals in mind:

- To be easily integrable with existing control architectures. Control algorithms are external to the simulator that in many cases only works as a visualization of the output computed by external programs. The software is mainly addressed to researchers and developers on underwater robotics, although special scenarios for ROV pilot training could be also implemented.
- To be general, modular and easily extendible. New robots can be easily included with XML description files. Support for widgets is also provided, allowing to

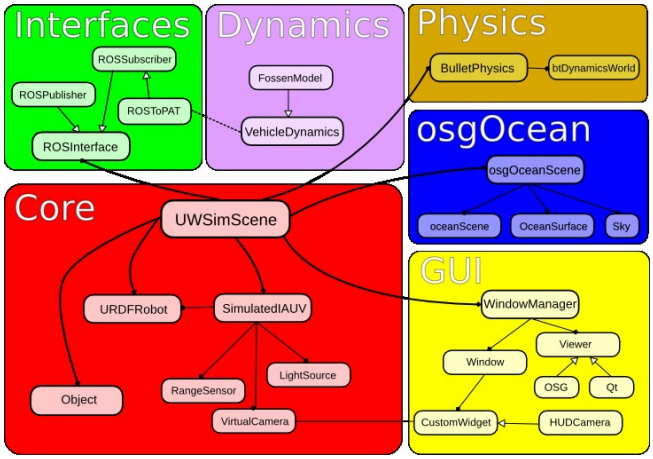

Fig. 2. A diagram of the main parts that compose UWSim.

show useful information on top of the scene.

- To include support for underwater manipulators, thus allowing simulating underwater intervention missions. Kinematic chains can be created and controlled.
- To be visually realistic, and to allow the user configuration of important parameters such as water color, visibility, floating particles, etc.

This paper is organized as follows: Section II briefly describes the software design aspects. Section III presents an overview of the main features of UWSim. Three different application examples are described in Section IV and, finally, Sections V and VI describe future work and conclude this paper.

II. SOFTWARE DESIGN

The simulator has been implemented in C++ and makes use of the OpenSceneGraph (OSG) [6] and osgOcean [7] libraries. OSG is an open source 3D graphics application programming interface used by application developers in fields such as visual simulation, computer games, virtual reality, scientific visualization and modeling. The toolkit is written in standard C++ using OpenGL and runs on a variety of operating systems including Microsoft Windows, Mac OS X, Linux, IRIX, Solaris, FreeBSD and recently also Android. On the other hand, osgOcean is another open source project that implements realistic underwater rendering using OSG. osgOcean was developed as part of an EU funded research initiative called the VENUS project [8].

UWSim uses the above mentioned libraries and adds further functionality for easily adding underwater robots to the scene, simulate sensors, and do the interface with external control programs through the *Robot Operating System* (ROS). Figure 2 shows the main components and classes of UWSim in its current version. Basically, there is a *Core* module in charge of loading the main scene and its simulated robots; an *Intefaces* module that provides communication with external architectures; a *Dynamics* module that implements underwater vehicle dynamics; a *Physics* module that manages the contacts between objects in the scene; the *osgOcean*, in charge of rendering the ocean surface and special effects, and the *GUI* module, that provides support for visualization and windowing toolkits.

III. MAIN CHARACTERISTICS

The main features of UWSim are described in the following sections.

A. Configurable environment

The 3D geometry to be loaded in UWSim can be easily configured with third-party modeling software as Blender, 3D Studio Max, etc. Figure 1 shows a screenshot of a shipwreck scenario being modeled with the open source modeling application Blender. The basic scene can be freely modeled, including materials and textures. The resulting scene will have the possibility to be loaded in the simulator as long as it is exported to any of the formats that OSG can read (.ive, .3ds, .wrl, etc.).

In addition to the basic 3D structure, additional elements can be dynamically added, modified and removed from the main program. OSG represents the virtual scenario with a scene graph, where the nodes can be easily accessed and managed. This includes not only geometry nodes, but also cameras, light sources, etc.

The complete scene can be described by the user with an XML file, where the main tags that can be used are:

- The `<oceanState>` block, that allows the configuration of ocean parameters as wind direction and speed (affects to the amount of waves), underwater color, visibility and attenuation factors.
- The `<simParams>` block, that lets the user disable visualization effects, set the window resolution, and set a world frame, given as an offset with respect to the default one.
- The `<camera>` block sets the main camera parameters. The main camera is the one that observes the scene and renders to the main window. The user can set the camera motion mode (free camera vs. *look at* camera), and other parameters such as the field of view, aspect ratio and clipping planes. It is also possible to set the camera parameters from the intrinsic calibration matrix.
- With the `<vehicle>` block, underwater robots can be included. The user has to specify a robot description file in *Unified Robot Description Format* (URDF), default joint values in case the robot contains joints, the pose of the robot in the scene, and vehicle sensors if needed (will be detailed later).
- The `<object>` block allows including other 3D models to the scene from existing file resources in any of the 3D formats supported by OSG.
- Finally, the `<rosInterfaces>` tags allows attaching ROS interfaces to certain objects. These provide sensor information to external software (pose of objects, images from virtual cameras, joint values, etc.), and also receive external references used for updating the pose of objects and robots in the scene.

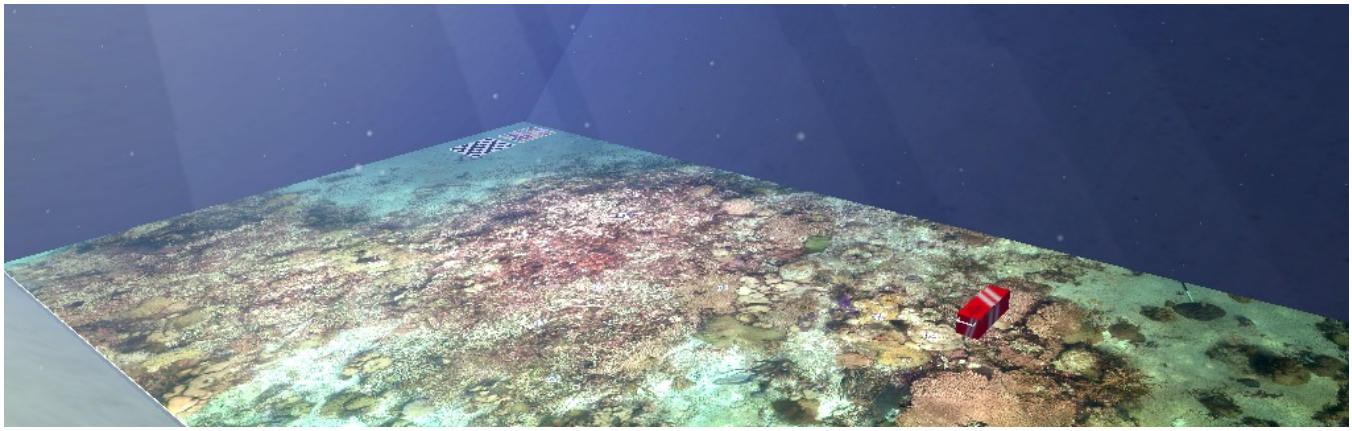

Fig. 3. A virtual visualization of the CIRS water tank (University of Girona). A printed posted is placed on the bottom and loaded in multi-resolution mode (seafloor texture provided by Pam Reid, Univ of Miami and Nuno Gracias, University of Girona).

B. Multiple robots support

A default vehicle in UWSim is composed of a 3D model (created by the user in third party modeling software) that can be positioned in the scene by setting 6 degrees of freedom (DOF). Support for kinematic chains (needed for manipulators) is also included. The robots are described with an XML file according to the URDF format, that can include kinematic, dynamic and visual information. As an example, two different underwater manipulators of 4 and 7 DOF are included in by default. These are the Lightweight ARM5E (see ref. [9]) and a Mitsubishi PA10 Arm. Different robots can be loaded and managed simultaneously in the scene by including multiple `<vehicle>` tags.

C. Simulated sensors

Four different sensors can be simulated in the current version of UWSim. By default, any vehicle object includes localization sensors that provide its 6 DOF pose (x, y, z, roll, pitch, yaw) in the scene. Gaussian noise can be added, for instance, to simulate uncertainties in the sensor readings. The underwater manipulators also include simulated position sensors in the joints that provide the joint angles.

Virtual cameras can be added to vehicles, thus providing virtual images of the environment that can be used for developing vision algorithms. These cameras can be initialized from intrinsic parameters, thus allowing the definition of virtual cameras with the same visual properties of their real counterparts.

Finally, virtual sensors that measure distances to obstacles along pre-defined directions are also included, thus simulating range sensors. Several of them can be added to vehicles by specifying the desired poses (relative to the vehicle origin frame), and the range limits.

D. Network interfaces

All the different robots sensors and actuators can be interfaced with external software through the network.

UWSim includes an interface for its integration with the Robot Operating System (ROS) [10], that is a set of libraries and tools that assist software developers create robotic applications. ROS is a distributed system where different nodes can run on different computers and mainly communicate through 'topics' via publishing/subscribing mechanisms. The ROS interface of UWSim allows running the simulator as another ROS node that can communicate with the rest of the architecture with the standard ROS communication facilities. This allows to seamlessly validate control methods developed in ROS either on UWSim or on the real robots, as long as they provide the same interface.

Through the ROS interfaces, it is possible to access/update any vehicle position or velocity, to move arm joints, and to access the data generated by virtual sensors. Interfacing with Matlab is also possible through the `ipc_bridge` ROS package, as will be described in one of the use cases at the end of the paper.

E. Contact physics

Simulation of contacts is supported by the physics engine Bullet [11], wrapped in *osgBullet* for its use with OSG. This allows detecting collisions and forces and automatically updating the scene accordingly. The different bodies collision shapes can be automatically generated from the 3D models. In addition, it is possible to set the position and attitude of collision shapes automatically from the scene graph, which is necessary, for instance, for updating the collision shapes transforms of kinematic chains like manipulators.

F. Dynamics

UWSim allows the dynamic simulation of rigid body motion, by using a state-space dynamic model in terms of state variables representing body linear and angular velocities and positions. This representation is suitable for control and simulation issues. It takes as inputs the forces and torques that act on the body (or other related variables) and their current state vector value. After an integration process, the output is a future estimation of the state vector, that is used for updating the vehicle poses in the scene.

The state-space dynamic model is based on two well-known relationships: i) the non-linear 6 DOF rigid body

motion equations, and ii) the jacobian matrix that maps body velocities into an inertial frame. In a matrix compact form, and adopting the SNAME notation [12], these equations can be writen as:

$$\mathbf{M}_{RB}\dot{\nu} + \mathbf{C}_{RB}(\nu)\nu = \tau_{RB} \quad (1)$$

$$\dot{\eta} = \mathbf{J}(\eta)\nu \quad (2)$$

where $\mathbf{M}_{RB}$ is the rigid body inertia matrix, $\nu = (u, v, w, p, q, r)^T$ denotes the linear and angular velocity vector with coordinates in the body-fixed frame, $\mathbf{C}_{RB}$ is the Coriolis and centripetal terms matrix, $\tau_{RB} = (X, Y, Z, K, M, N)^T$ is used to describe the forces and moments acting on the body in the body-fixed frame, $\eta = (x, y, z, \phi, \theta, \psi)^T$ denotes the body position and orientation vector with coordinates in the earth-fixed frame.

Depending on the level of realism needed, the previous model can be complemented with other dynamic models like those corresponding to the marine environment (hydrodynamics, waves, wind, underwater currents, etc.), to the actuators (thrusters and control surfaces as rudders or fins), and to the sensors (sonar, DVL, etc.). In [13], models suited for dynamic simulation and control of the hydrodynamic effects and disturbance forces generated by waves, wind and currents are detailed. A compact representation of these models can be expressed as:

$$(\mathbf{M}_{RB} + \mathbf{M}_A)\dot{\nu} + \mathbf{C}_{RB}(\nu)\nu + \mathbf{C}_A(\nu_\mathbf{r})\nu_\mathbf{r} + \mathbf{D}(\nu_\mathbf{r})\nu_\mathbf{r} + \mathbf{g}(\eta) = \tau_E + \tau \quad (3)$$

$$\nu_r = \nu - \nu_c \quad (4)$$

being $\mathbf{M}_A$ the added inertia matrix due to the acceleration experimented by the fluid surrounding the body when it is moving, $\mathbf{C}_A(\nu_\mathbf{r})$ the added Coriolis and centripetal terms matrix due also to the fluid, $\mathbf{D}(\nu_\mathbf{r})$ the hydrodynamic damping matrix (radiation-induced potential and viscous), $\mathbf{g}(\eta)$ the gravitational and buoyant forces and torques vector, τ_E used to describe the environmental (waves and wind) forces and moments acting on the vehicle, τ the propulsion forces and moments, and $\nu_c = (u_c, v_c, w_c, 0, 0, 0)^T$ a vector of irrotational body-fixed current velocities. Notice that this model representation is based on the state variables (ν, ν_c, η).

The model above is implemented in a separate module, currently writen in Matlab. The output variables (position and attitude of vehicles) are published on the ROS network and captured by the UWSim core for updating the visualization. In the near future we plan to extend the model to consider open kinematic chains, e.g. for I-AUV simulation [14]. In this case, due to the complexity and characteristics of the system, numeric algorithms like Recursive Newton-Euler are normally used [15].

G. Widgets

Customizable widgets can be added to the main window. Widgets are small windows that can be placed inside the main display in order to show specific data to the user.

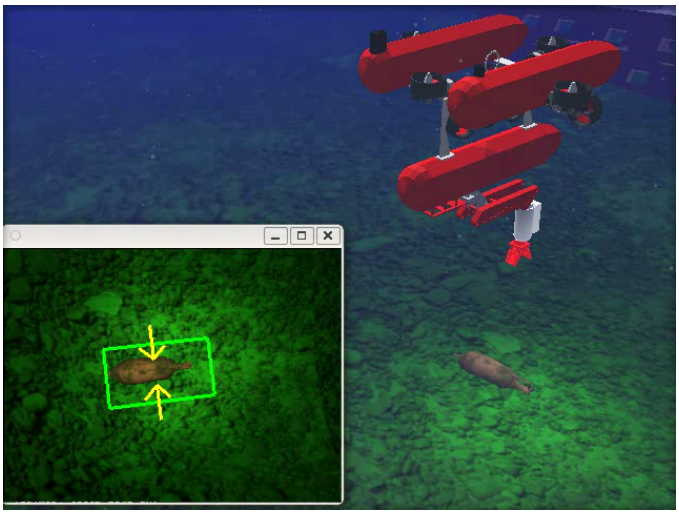

Fig. 4. A screenshot of the simulator, used in this case for testing a tracking algorithm on virtual images. A video of a complete mission simulation can be found accompanying this paper, and also at `http://goo.gl/lmzjw`

An abstract interface for the creation of custom widgets is provided. It allows to create specialized classes for displaying useful data depending on the specific application. For instance, the virtual camera view is displayed on a widget, which position and size can be modified during execution by the user.

H. Multi-resolution terrain

Terrain models can be loaded as standard objects, even though they can be composed of complex meshes with multi-resolution textures generated externally from bathymetry and imagery. OpenSceneGraph has built-in support for database paging that is used for loading different Level-of-Detail (LOD) models from disk in realtime. Other projects like *VirtualPlanetBuilder*, are built on this feature and provide facilities for loading large terrains and textures that are common in underwater robotics. As an example, Figure 3 shows a large image mosaic (with a resolution 9921×20126 pixels) loaded in multi-resolution in UWSim.

IV. EXAMPLES OF APPLICATION

UWSim can be used in many different ways depending on the specific application. These are three possible use cases:

A. Validate perception and control algorithms

As UWSim supports virtual cameras, it is possible to apply vision-based algorithms on virtual images simulating real conditions. Figure 4 shows a scenario where a I-AUV is tracking an amphora found on the seabed. In this example UWSim just renders the scenario and publishes on the network the image captured from the virtual camera. An external tracking algorithm, totally independent of UWSim, receives the image as if it came from a real camera, and performs tracking using its own libraries (e.g. OpenCV).

The output of the tracker could be used for feeding a control algorithm in order to stabilize the vehicle, perform grasping (see Fig. 5), etc. These controllers could be connected to

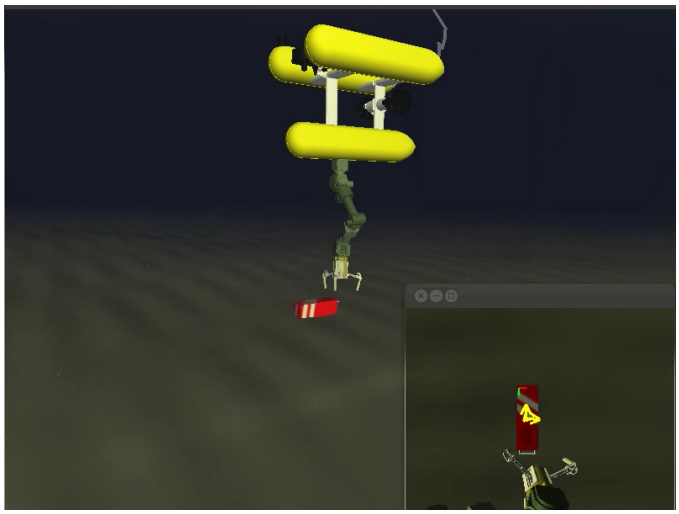

Fig. 5. Arm control and manipulation actions can also be simulated.

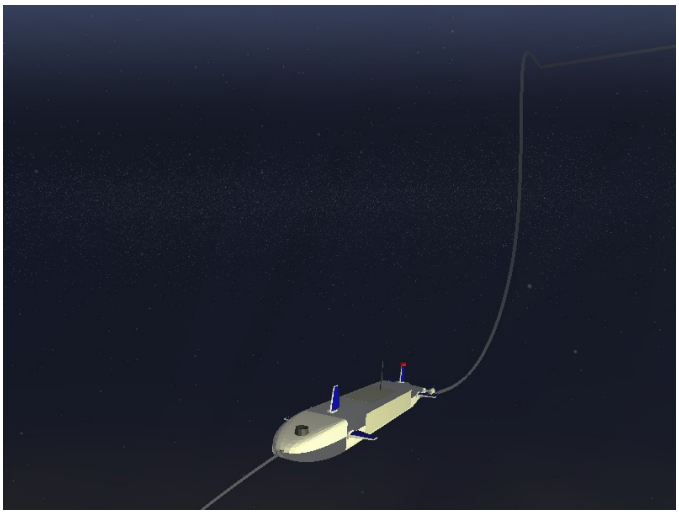

Fig. 7. The Phoenix dynamics being tested in UWSim.

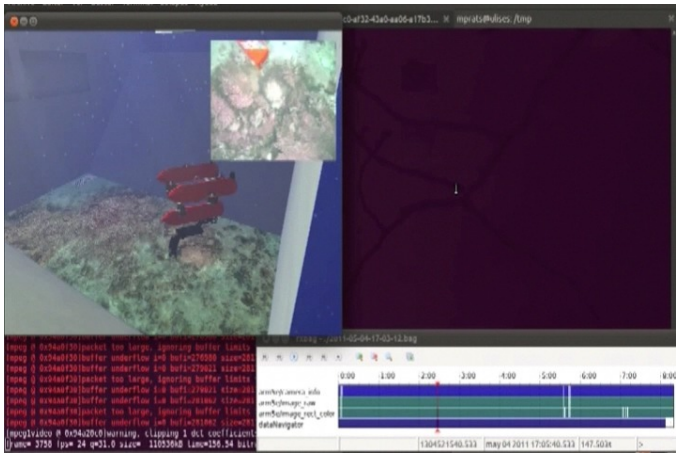

Fig. 6. A survey being reproduced in UWSim, from the dataset captured during the real survey in ROS bag format. The complete sequence can be found at `http://goo.gl/HEkrg`

a real robot or back to UWSim, thus closing the loop through an external architecture. The video accompanying this paper shows a complete survey & intervention mission simulated in this way. Vehicle and arm controllers developed in an external control architecture send their outputs to UWSim that updates the virtual vehicle and arm accordingly, and sends back virtual sensor feedback to the real controllers.

B. Supervision/Playback of a survey

It is also possible to use UWSim for mission playback. For instance, the navigation data acquired during a survey mission with a real robot can be logged and then reproduced in UWSim in order to analyze the vehicle trajectory. If bathymetry and images of the seafloor are gathered during the survey, it would be possible to build a textured terrain from them and visualize it in UWSim. Therefore, it is possible to obtain a virtual visualization of a real mission.

Figure 6 illustrates this concept. It shows a 3D visualization of a real survey task carried out with an I-AUV at the CIRS water tank (University of Girona), during the TRIDENT EU Project 1st Annual Review. The robot autonomously performed a survey of the floor. At the same time, odometry and imaging data was published on the network (via ROS topics). This data was recorded with ROS bagging tools and later used as input to UWSim. Both UWSim and the real robot could have been running simultaneously, thus allowing to visualize the vehicle survey in real time. As UWSim also supports the visualization of external video (through a widget), it is possible to include the real video stream on top of the virtual scenario.

C. Simulation of a vehicle dynamics

In this example, UWSim is used in order to simulate the dynamic behaviour of the NPS AUV II (PHOENIX) vehicle. The dynamic model of this vehicle was established in [16], including the rigid body dynamics, the added mass inertia, added Coriolis and centripetal terms, damping and buoyancy hydrodynamic effects.

To demonstrate the modularity and flexibility of UWSim, Matlab was used for the dynamic simulation of the vehicle, by using the GNC toolbox [17], where the Phoenix dynamic model is already implemented. The test simulated the Phoenix behaviour from its inital state $\mathbf{x}_i = (0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T$, with an inital step input of $n = 500$ rpm in the thrusters and an initial step input of $\delta_r = 6$ degrees in the rudder. The Euler's method was used in the integration algorithm.

To communicate Matlab with UWSim, the `ipc-bridge` ROS stack was used as mentionned in the previous sections. A `nav_msgs/Odometry` message, containing the position and orientation of the vehicle with respect to a fixed frame, was continuously computed from the dynamic equations and published on a topic from the Matlab side. UWSim was listening to these messages on the same topic, and updated the virtual scene accordingly. The vehicle trajectory was also plotted inside UWSim for visualization purposes. Figure 7

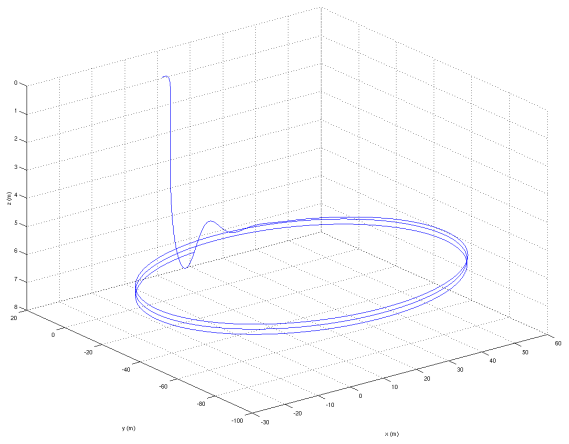

Fig. 8. The complete trajectory followed by the Phoenix vehicle on the dynamics simulation.

shows a part of the robot trajectory being reproduced and plotted in UWSim, whereas Figure 8 shows a Matlab plot with the complete 3D path followed by the vehicle.

This example shows how existing algorithms developed with external software like Matlab, using specific libraries or toolboxes, can be easily interfaced and validated with UWSim. It is worth mentionning that the inputs to this simulation were only the thrusters velocity and the rudder angle. It is part of our future work to introduce also disturbance forces and torques as those created by collisions (detected by the physics engine), or by underwater currents.

V. FUTURE WORK

UWSim is still in development, and new features are expected to be continuously added. In particular, the following items are planned to be implemented in near future versions:

- Interactive Markers. ROS interactive markers allow to create and interact with geometry in the scene, and to seamlessly report this user interaction to external modules. They are, therefore, a very useful tool for creating 3D user interfaces.
- Qt frontend, to be added as an optional module wrapping UWSim. This would allow to create user interfaces based on buttons and dialogs using the Qt library.
- Support for benchmarking is being actively developed, and will allow quantitative comparison of different control/vision algorithms working on the same scenario.
- New simulated contact sensors like force and touch will be implemented and used for simulating manipulation algorithms. Support for typical navigation sensors like DVL (Doppler Velocity Log), IMU (Inertial Measurement Unit), AHRS (Attitude Heading Reference System), etc. is also planned.

VI. CONCLUSIONS

This paper has presented a new software tool for 3D visualization and simulation of underwater missions. The 3D scene is highly configurable by the user, that first sets the basic geometry using third-party modeling software, and then can include multiple simulated underwater vehicles and manipulators. UWSim can be used mainly as validation of external control programs that can be easily attached to the virtual world. Furthermore, simulation of sensors such as virtual cameras and range sensors is also possible. The interface with external software is implemented through ROS interfaces, thus allowing it to be used in a distributed manner. Vehicle dynamics can be simulated and interfacing with Matlab is possible. This software fills a gap in the underwater robotics community and opens new possibilities to researchers on underwater control, vision and manipulation. The simulator is publicly available as open source at `http://www.irs.uji.es/uwsim`, and at the ROS wiki: `http://ros.org/wiki/UWSim`.

VII. ACKNOWLEDGEMENTS

This research was partly supported by Spanish Ministry of Research and Innovation DPI2011-27977-C03 (TRITON Project), by the European Commission Seventh Framework Programme FP7/2007-2013 under Grant agreement 248497 (TRIDENT Project), by Foundation Caixa Castelló-Bancaixa PI.1B2011-17, and by Generalitat Valenciana ACOMP/2012/252. The authors are very grateful to the ViCOROB group at University of Girona that provided us with vehicle 3D models and photo-mosaics.

REFERENCES

[1] O. Matsebe and C.M. Kumile. A review of virtual simulators for autonomous underwater vehicles (auvs). In *NGCUV 2008*, Lakeside Hotel, Killaloe, Ireland, 2008.
[2] CMLabs. Vortexsim. http://www.vxsim.com/en/simulators.
[3] Marine Simulation. Rovsim. http://marinesimulation.com.
[4] GRL. Rovolution. http://www.generalrobotics.co.uk.
[5] Fugro General Robotics Ltd. Deepworks. http://www.fugrogrl.com/software/.
[6] OpenSceneGraph. http://www.openscenegraph.org.
[7] osgOcean. http://code.google.com/p/osgocean.
[8] F. Alcala et al. Venus (virtual exploration of underwater sites) two years of interdisciplinary collaboration. In *14th International Conference on Virtual Systems and Multimedia*, Limassol, Cyprus, October, 2008.
[9] M. Prats, J. C. García, J. J. Fernández, R. Marín, and P. J. Sanz. Towards specification, planning and sensor-based control of autonomous underwater intervention. In *18th IFAC World Congress*, pages 10361–10366, Milano, Italy, Aug. 2011.
[10] S. Cousins. Welcome to ROS topics. *IEEE Robotics & Automation Magazine*, 17(1):13–14, 2010.
[11] Bullet physics library. http://bulletphysics.org/.
[12] Nomenclature for treating the motion of a submerged body through a fluid. Technical report, Society of Naval Architects and Marine Engineers, 1950.
[13] T.I. Fossen. *Guidance and control of ocean vehicles*. Wiley, 1994.
[14] G. Antonelli. *Underwater robots: motion and force control of vehicle-manipulator systems*. Springer tracts in advanced robotics. Springer, 2006.
[15] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul. On-line computational scheme for mechanical manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102(2):69–76, 1980.
[16] A.J. Healey and D. Lienard. Multivariable sliding mode control for autonomous diving and steering of unmanned underwater vehicles. *IEEE Journal of Oceanic Engineering*, 18(3):327–339, jul 1993.
[17] Mss: Marine systems simulator. http://www.marinecontrol.org.