Stonefish: An Advanced Open-Source Simulation Tool Designed for Marine Robotics, With a ROS Interface*

Patryk Cieślak[1]

***Abstract*— The marine robotics community is lacking a high quality simulator for doing scientific research, especially when it comes to testing control and vision algorithms in realistic underwater intervention tasks. All of the solutions used today are either outdated or try to combine different software tools, which often results in bad performance, stability issues and lack of important features. This paper presents a new software tool, focused on, but not limited to, simulation of intervention autonomous underwater vehicles (I-AUV). It delivers advanced hydrodynamics based on actual geometry, simulation of underwater sensors and actuators, as well as realistic rendering of underwater environment and ocean surface. It consists of a library written in C++ and a Robot Operating System (ROS) package.**

I. INTRODUCTION

Simulation tools are essential in robotics, due to the many different aspects of robot design that have to be integrated and tested. They are even more important when access to the testing facilities and the robot itself is limited, which is often the case in marine robotics and especially underwater robotics. Multiple open-source solutions were created during the realisation of different projects in this field but there is no ultimate modern tool. Moreover, most of the developed solutions were either partial or abandoned after the projects were finished. One of the first and most commonly known, which is still used although not developed anymore, is the UWSim [14]. It enabled simulation of underwater inspection and intervention missions with one or more robots. One of its main advantages is that it delivers a ROS (Robot Operating System [15]) package which allows for easy integration with ROS-based software architectures, commonly used in robots developed by scientific community. This allows for replacing the real robot with a simulator, without the need for any code modifications. However, in practice, UWSim is mostly used as a visualisation tool and for simulating underwater sensors. It lacks accurate simulation of dynamics and hydrodynamics of vehicles and it does not support simulation of manipulator dynamics (only kinematics). In practice, users of UWSim often simulate the physics with their own code. The rendering it delivers is still the best available, although outdated. A more modern approach is to use the ROS standard simulator Gazebo [12]. It is lacking native support for hydrodynamics and its poor rendering quality does not allow for convincing simulation of underwater vision. It can be used either with a custom dynamics code, like UWSim, or combined, e.g., with the UUV Simulator plugin [13] which adds simple hydrodynamics to the Gazebo physics engine. However, this solution is lacking the simulation of manipulator hydrodynamics and buoyancy. Gazebo was used in a recent project called DexROV [1], coupled with a set of plugins. Apart from visual tools, prepared for integration with ROS, there is also the Marine Systems Simulator [11] which is a Matlab/Simulink package. However, lack of visualisation, and no direct interface with ROS makes it unpopular in the community.

This paper presents a new simulation tool that was designed to fulfil the needs of researchers requiring accurate simulation of robot dynamics, vision and underwater sensors/actuators, i.e., ones working with control, perception and navigation algorithms. What makes it unique is the simulation of complete dynamics/hydrodynamics of vehicle-manipulator systems with contact and force sensing. It also sports a modern rendering pipeline to deliver better rendering of underwater scenes. In Section II the general architecture of the simulation tool is described, including its integration with external software and ROS. Section III presents all important and unique features of the software. In Section IV author describes the technical details of the simulation of robot hydrodynamics. Section V provides insight into the rendering pipeline, responsible for generating realistic images of underwater scenes. Section VI gives an overview of building a simulator using the *Stonefish* library. Finally, Section VII concludes the presentation and discusses further developments planned by the author.

II. ARCHITECTURE

The presented simulation tool is composed of two parts: a C++ library and a ROS package, as shown in Fig. 1.

The first part, the C++ library, constitutes the simulation framework, which is a set of classes that wrap around the physics library (Bullet Physics [7]), adding an abstraction layer which greatly simplifies creation of simulation scenarios and adds features related to robotics, especially marine robotics, e.g., buoyancy and hydrodynamics. The choice of the physics library to build upon was made based on a survey [2]. The *Stonefish* library also contains classes representing different types of virtual sensors and actuators used in surface, underwater and even flying robots. To deliver a lightweight, high performance solution, the rendering pipeline is custom and closely coupled with the simulation, i.e., no external graphics engine was used, all is implemented directly in OpenGL. At the top level, the

*Patryk Cieślak has received funding from the European Community H2020 Programme under the Marie Skłodowska-Curie grant agreement no. 750063 and under EU Marine Robots project grant agreement no. 731103.

[1]Patryk Cieślak is with the Computer Vision and Robotics Research Institute, Scientific and Technological Park of the University of Girona, CIRS lab, 17003 Girona, Spain patryk.cieslak@udg.edu

978-1-7281-1450-7/19/$31.00 ©2019 IEEE

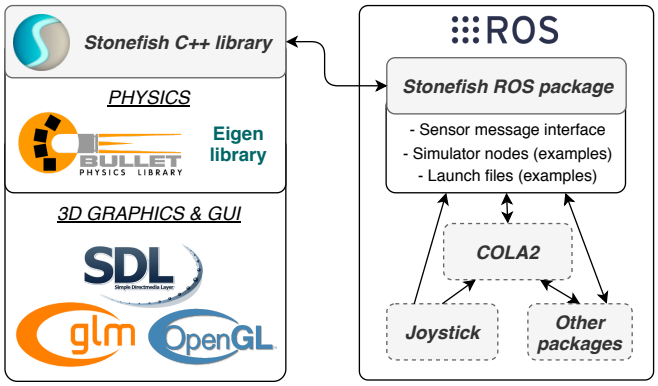

Fig. 1. General architecture of the *Stonefish* simulation tool.

Stonefish library implements classes which allow for creating standalone simulators, thus it can be used on any platform. Moreover, if the graphical part of the simulation is not needed (not only visualisation but also virtual cameras) or the platform that is used does not meet GPU requirements, one can use a console based simulation.

The second part, the ROS package, contains a ROS message interface, which allows for convenient publishing of virtual sensors' measurements in a standard way, as well as examples of simulators and launch files needed to run them. These examples are based on GIRONA500 AUV [16] which is the main research platform of the author. Thus, they need open-source COLA2 architecture, developed by Iqua Robotics, to be installed and run. However, it is not necessary to use COLA2 in general, users can combine the simulator with whatever architecture they normally use on-board their robots, to naturally substitute the real robots with the simulation.

III. FEATURES

The *Stonefish* library implements multiple standard and unique features, not limited to marine robotics. An overview of the most important components is presented in Fig. 2. The simulation scenario is a combination of a model of the environment and one or more models of robots. The environment can be configured to include ocean simulation but it is not required. There is also an atmosphere model implemented, with a realistic dynamic sky. It is possible to create static as well as dynamic objects in the environment. Apart from standard solids, the library implements a heightfield terrain, i.e., terrain mesh based on a 2D height map, and allows for loading geometry from mesh files. Each robot can be a single rigid-body or a rigid-body tree. Links of the robot are connected with joints (revolute, linear, fixed), which can be propelled by different actuators. Some actuators are attached to the links directly. Sensors work in the same way. The following subsections give more details about most important components of the simulation framework.

A. Dynamics and Collision

The software can accurately simulate complete dynamics of vehicle-manipulator systems, using the Featherstone

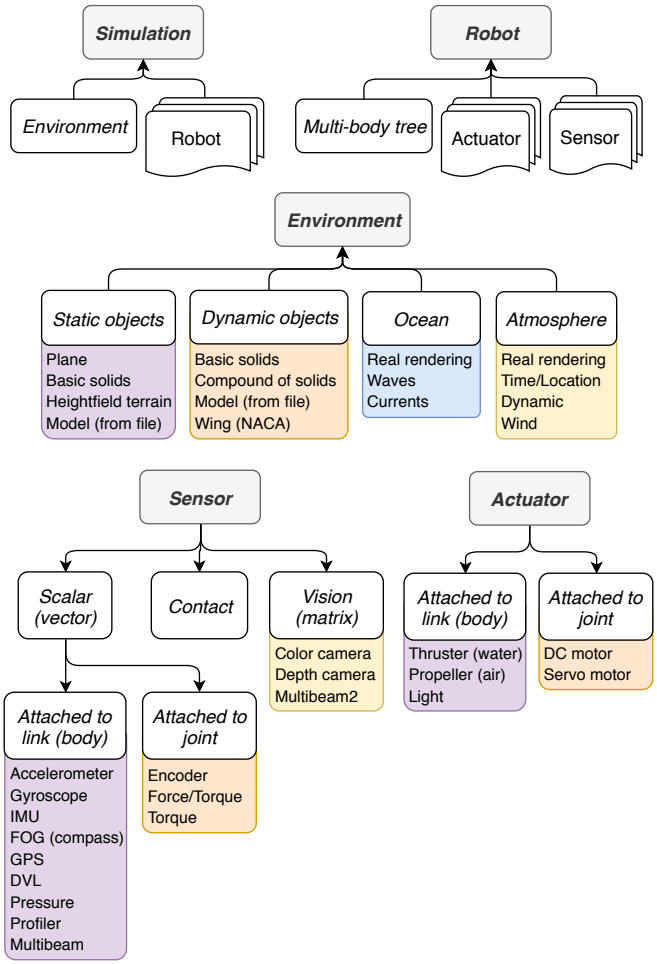

Fig. 2. Features implemented in the library.

multi-body algorithm [9]. Each link of the robot can be a simple solid, a polyhedron loaded from a mesh file, a wing profile (NACA 4-digit system) or a composition of all of them. It is possible to define geometry used for physics and rendering separately. To calculate the physical properties, like mass and inertia, a material is assigned to each body. Materials are created before the geometry and include such data as density and restitution. Moreover, to accurately simulate friction of sliding bodies, interaction between each pair of materials is defined with static and dynamic friction coefficients. All dynamic and static bodies can collide and the character of the collisions can be stiff or compliant. In case of a compliant contact it is possible to define stiffness and damping coefficients. Additionally, it is possible to track the contact paths and forces between selected bodies.

B. Ocean and Atmosphere

The most unique features of the simulation tool are ones connected with the medium in which robots work. The *Stonefish* library implements ocean and atmosphere models, including fluid dynamics and realistic rendering. Both elements play key roles in the simulation of robots: the former one is necessary to achieve high quality estimation of the

motion of bodies and the latter one to produce realistic images from virtual cameras, including the effects of light attenuation and scattering, which limit the practical use of vision sensors.

The ocean model includes geometrical waves, underwater currents, computation of geometry-based hydrodynamics and buoyancy (see Section IV), as well as realistic surface and underwater rendering (see Section V). Waves are generated using FFT based algorithms [18]. The velocity of water in the ocean volume can be modified by defining velocity fields: uniform, currents, jets, map based, and combining them.

The atmosphere model includes winds, computation of geometry-based drag (see Section IV), as well as physically correct sky rendering and lighting (see Section V). The velocity of air in the atmosphere can be modified by defining velocity fields like in the case of the ocean model.

C. Actuators

The software is prepared for simulating actuators which are attached to the multi-body joints or links. Joint actuators are rotary or linear drives that are commonly used in robots. A basic servomotor is implemented to serve the purpose of moving joints of a virtual manipulator. There is also a linear model of a geared DC motor, to present how to extend the capabilities of the library by subclassing general actuator classes. In terms of link actuators, the library implements models of an underwater thruster and a fixed propeller (for a quadcopter). At the time of writing the fins are simulated using the geometry-based hydrodynamics and the airplane wings are not supported, due to the lack of lift calculation. The solution to the latter will be to implement lift models based on wing profile characteristics. This is not the main focus of the software thought. It should be noted that lights are also considered link actuators in the presented software.

D. Sensors

All commonly used robot sensors can be simulated with *Stonefish*. Sensors are primarily divided based on the data they deliver and the way they are implemented, into *Scalar* and *Vision*. The *Scalar* sensors are ones outputting a single number or a vector of numbers, while the *Vision* sensors output a matrix of data and they only work in graphical simulation mode. Like in the case of the actuators, the *Scalar* sensors can also be attached either to a joint or to a link of a robot, depending on the type of the sensor. Moreover, all sensors provide a way to define their range as well as noise characteristics. The available models of sensors should cover most of the needs but in case a specific sensor model has to be implemented subclassing standard sensor classes is the way to go. What is currently implemented can be checked in Fig. 2. Finally, the library implements a special kind of sensor called *Contact*, which enables monitoring of contact path and contact forces between selected pair of rigid bodies. With the actuators and sensors currently implemented the author was able to simulate complete underwater non-destructive testing experiment, presented in Fig. 3, which was later performed with the real system [6].

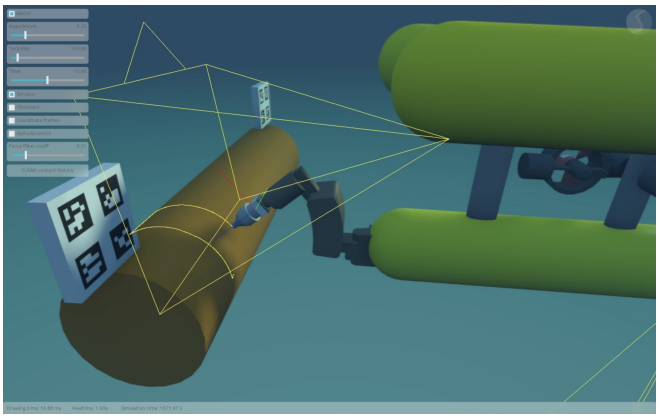

Fig. 3. Simulated NDT testing performed by GIRONA500 I-AUV.

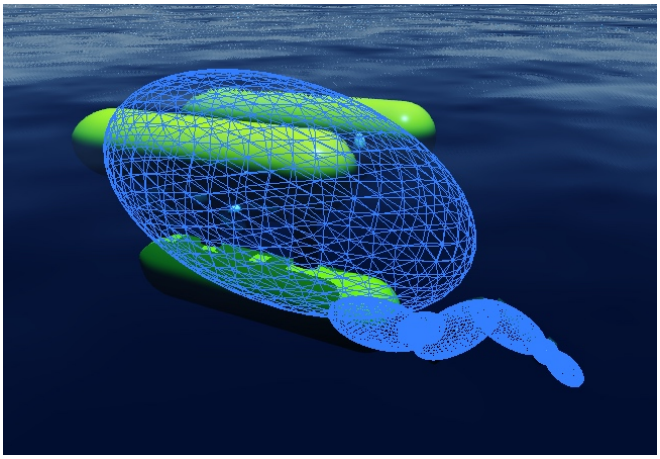

Fig. 4. Automatically computed, best fitting ellipsoids, for the estimation of added mass and shape coefficients.

E. Input Support and Simple GUI

To facilitate the use of the software to build stand-alone simulators, the library implements support for keyboard and joystick input as well as a simple immediate-mode graphical user interface (GUI). It allows for putting buttons, sliders, checkboxes and even simple plots in the main application window. It can also be useful to change some parameters of the simulation that are not necessarily connected with the robot architecture, like time of day or turbidity of water. There is a standard GUI implemented in the base classes, which can be overwritten by the user. Moreover, the GUI also includes a console which displays messages, warnings and errors, generated during the initialisation of the simulation world, which is useful for debugging.

IV. GEOMETRY-BASED HYDRODYNAMICS

The most unique feature of the *Stonefish* library is the computation of buoyancy and hydrodynamics based on actual geometry. Due to the fact that true computational fluid dynamics (CFD) cannot be simulated in realtime, the method used in this software is an approximation and extension of the Fossen model [10], accurately reproducing motion of underwater objects in a qualitative sense. Each rigid body is

represented by a polyhedron. For standard solids, the mesh used for force computation is generated automatically. When loading geometry from a file, the user is fully in control of the quality of the simulation, because the computations are based on the mesh loaded. Therefore it is important to avoid large faces and keep them all at a similar size. In the future, automatic mesh refinement algorithms may be implemented. Computation of buoyancy, linear drag and quadratic drag is performed by iterating through all N faces of a mesh. For each face i a force component and a torque component is computed. The sum of all of the components gives the net force and torque acting on the centre of gravity of the body. Thus, the buoyancy calculation is given by

$$\begin{aligned} \bar{b}_i &= \begin{cases} -\hat{n}_i h_i S_i & h_i > 0 \\ 0, & h_i \leq 0 \end{cases} \\ \bar{B} &= \rho g \sum_{i=1}^{N} \bar{b}_i, \quad \bar{\tau}_b = \rho g \sum_{i=1}^{N} \bar{r}_i \times \bar{b}_i, \end{aligned} \tag{1}$$

where $\hat{n}_i$ is a versor normal to face i (pointing outwards), h_i is the water depth at the centre of face i (this allows for geometrical waves), S_i is the surface area of face i, ρ constitutes the density of fluid, g is the gravitational acceleration and $\bar{r}_i$ is a vector from the centre of gravity to the centre of face i. The accuracy of buoyancy computation is only limited by the discretisation of the geometry. The same technique is used to compute the linear and quadratic drag:

$$\begin{aligned} \bar{l}_i &= \begin{cases} \bar{v}_i e^{-0.5|\bar{v}_i|^2} S_i, & \bar{v}_i \cdot \hat{n}_i < 0 \\ 0, & \bar{v}_i \cdot \hat{n}_i \geq 0 \end{cases} \\ \bar{q}_i &= \begin{cases} \bar{v}_i |\bar{v}_i| S_i, & \bar{v}_i \cdot \hat{n}_i < 0 \\ 0, & \bar{v}_i \cdot \hat{n}_i \geq 0 \end{cases} \\ \bar{D}_l &= \frac{1}{2}\rho \bar{C}_l \cdot \sum_{i=1}^{N} \bar{l}_i, \quad \bar{\tau}_l = \frac{1}{2}\rho \bar{C}_l \cdot \sum_{i=1}^{N} \bar{r}_i \times \bar{l}_i \\ \bar{D}_q &= \frac{1}{2}\rho \bar{C}_q \cdot \sum_{i=1}^{N} \bar{q}_i, \quad \bar{\tau}_q = \frac{1}{2}\rho \bar{C}_q \cdot \sum_{i=1}^{N} \bar{r}_i \times \bar{q}_i, \end{aligned} \tag{2}$$

where $\bar{v}_i$ is a component of fluid velocity perpendicular to face i, at the centre of this face, while $\bar{C}_l$ and $\bar{C}_q$ are vectors of shape coefficients for the linear and quadratic drag respectively (three coefficients - one for each axis). Thanks to the sampling of velocity at each face of the mesh it is possible to simulate effects of currents, water jets etc., acting only on a part of the robot. The accuracy of this computation is limited by the fact that fluid velocity around the body follows Navier-Stokes equations, which are not possible to solve in realtime in a general case, and the unknown shape coefficients. The latter ones are approximated by finding an ellipsoid or cylinder best fitting the geometry. This idea is also used for approximating the added mass of the bodies, see Fig. 4. All of the presented calculations can be easily parallelised to significantly improve the performance and allow for more submerged bodies and higher sampling rates, which is a future goal of the author.

V. REALISTIC RENDERING

An important part of the software is the realistic rendering of simulated scenarios. It is crucial for accurate simulation of images generated by cameras, which are then used to test developed vision algorithms. This is especially important for researchers working with underwater robots, for whom the more similar the images are to reality the better the algorithms can be tuned for real applications. Realistic rendering of underwater scenes as well delivers an estimation of the range at which cameras can be used, depending on the type of water and amount of light available.

The *Stonefish* library is not using any external graphics engine but implements a lightweight, custom solution, closely coupled with the simulation. Most of the open-source rendering engines are designed for game developers and need to cover a wide range of applications, which results in bulky and heavy software, with lots of configuration and unnecessary code burden. Moreover, the rendering quality often does not live up to the current standards and the algorithms used are not physically based. They also do not provide functions needed for generating realistic underwater images. The rendering pipeline of *Stonefish* is using the new OpenGL API (application programming interface), solely based on shaders, with no fixed rendering functions. Shaders enable endless possibilities for utilising physically based algorithms and thus simulating all effects encountered in unusual environments, like the underwater. All lighting calculations are performed with HDR (high dynamic range) buffers, later tone-mapped to LDR (low dynamic range) results, which allows for covering a wide range of light intensities, occurring especially in real world outdoor scenes. The main light sources in the environment are sun and sky. *Stonefish* simulates both of them with physical correctness, based on [4], [3]. The simulated effects include multiple scattering and absorption of sun light in a multilayered model of Earth's atmosphere. The complexity of the problem can be appreciated by realising that the illuminance and colour at each point on Earth's surface depends on light reaching it from every direction. The heavy computations are done at the start of the simulation and stored in multidimensional textures. The position of the sun can be changed dynamically, e.g., according to the date/time and geographic coordinates. All this gives natural sky rendering and lighting of objects. The pipeline also supports local lights and shadows. Another complex element to render is the ocean surface. The shaders used here are implemented following [8], [5]. The result of all of the mentioned algorithms can be appreciated in Fig. 5.

The most important, as the simulation is directed towards marine robotic researchers, is the rendering of underwater environment. The simulated effects of underwater lighting include absorption, scattering and air-light. Other simulators commonly implement just a simple fog effect when rendering underwater scenes. The intensity of light absorption depends heavily on light wavelength and water type (algae), as shown in [17]. This work was used to obtain absorption coefficients for three basic colour components - red, green and blue.

Fig. 5. Realistic rendering of sky and ocean surface.

This results in realistic colour fading and image darkening. It is easiest to see with colours related to long wavelengths - the red elements are quickly faded to dark blue. Figure 6 shows an example of this effect on a poster used for vehicle localisation. The scattering of light results from turbidity and it affects images with haze (fog) and blur. Figure 7 presents an example of images generated with different turbidities.

VI. BUILDING A SIMULATOR

The *Stonefish* library contains all components needed to build a stand-alone simulator. This kind of simulator can run on any platform supported by the SDL library and the OpenGL library (version $\geq$ 3.2). In case of the console mode simulation the OpenGL support is not necessary. There are two base classes that can be overwritten when building a simulator: a) a class representing the application (GUI and input handling) and b) a class representing the simulation (physics, rendering). If no modification of GUI and no special input handling are needed, only class b) has to be subclassed. In class b) there are basically 3 methods that have to be overwritten: b1) constructor (e.g., initialisation of data fields, creating ROS publishers and subscribers), b2) scenario definition (creating and setting up all elements of the simulation) and b3) post simulation tick callback (called after a simulation step is completed, used for, e.g., publishing sensor outputs, updating actuators). In class a) it is possible to overwrite handling of keyboard, mouse and joystick input as well as display and operation of the immediate mode GUI.

Building a simulation scenario (overwriting method *b2*) is based on writing intuitive code, hiding the complex interaction with the Bullet Physics library. User has to first define physical and graphical materials and enable ocean simulation if needed. Then objects in the environment are created - either static (flat plane, terrain etc.) or dynamic, moving when in collision. Next, the user defines meshes used for robot links, creates sensors and actuators. Lastly, a robot or multiple robots are composed of the created components, following concepts used in the URDF (Unified Robot Description Format) files.

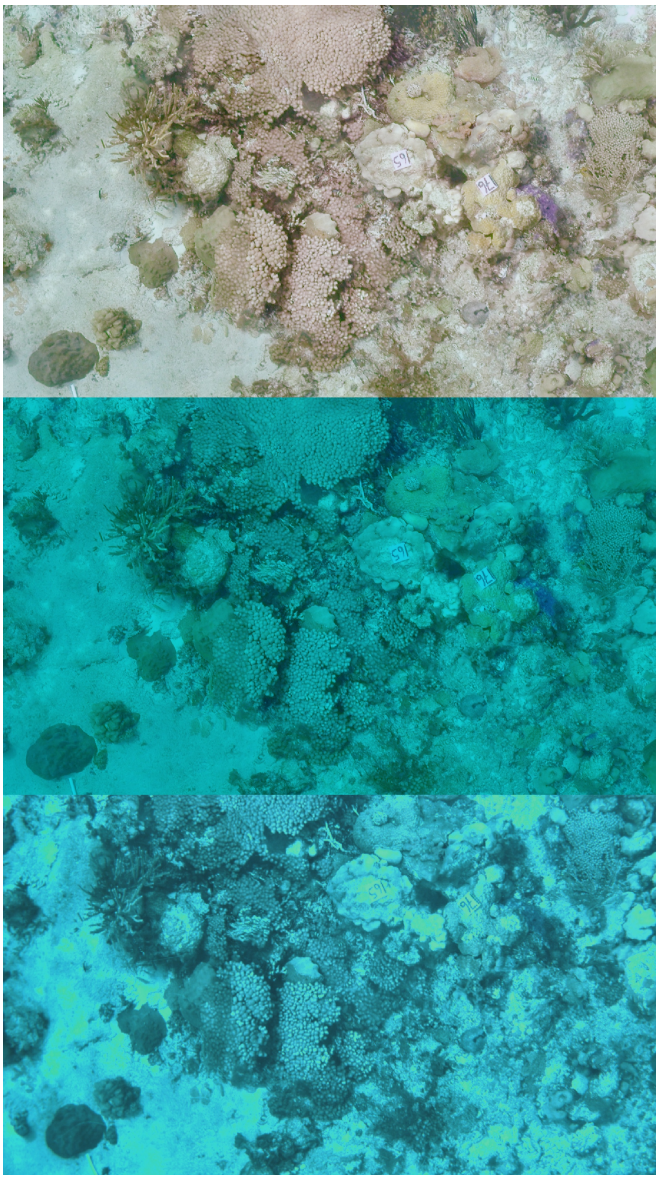

Fig. 6. Simulated map-based navigation using a large poster. From the top: simulated image without water, simulated image with water, image from a real camera captured in our test tank.

VII. CONCLUSIONS

In this paper a new open-source simulation tool called *Stonefish* is presented. It is composed of a C++ library and a ROS package to facilitate easy integration with ROS based robots. The library can run on any platform to ensure independence from the control architecture used by the users in their robots, i.e., the ROS package is just an optional part of the tool. *Stonefish* delivers realistic simulation of physics and high quality rendering, which can satisfy the needs of researchers working both in control of robots and vision algorithms. The tool implements unique features related to marine robotics, like geometry-based hydrodynamics, reproducing real effects in a qualitative sense, or advanced rendering of underwater scenes, including light absorption and scattering, really limiting the usable range of cameras.

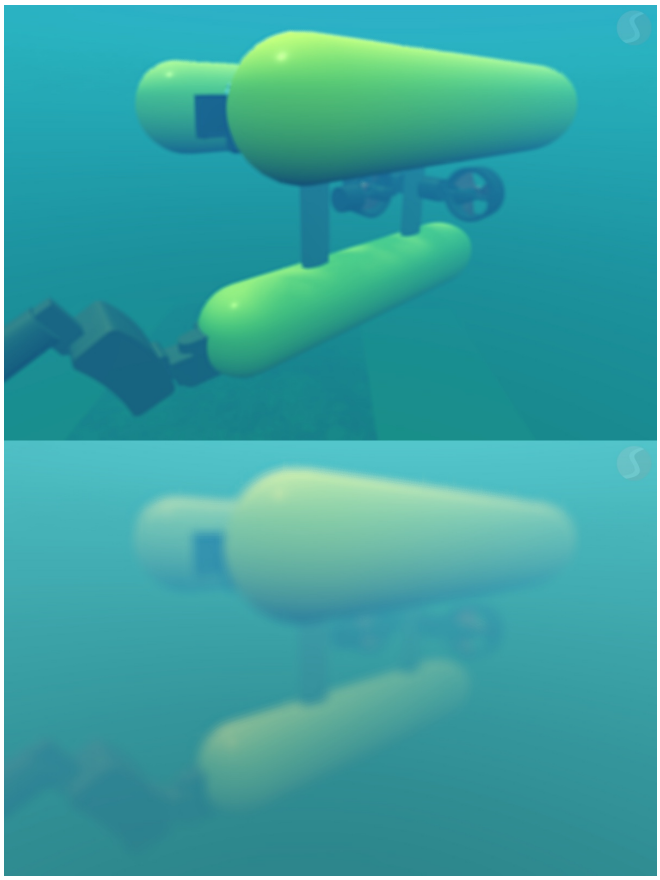

Fig. 7. Simulated effect of light scattering and air-light for different turbidity values. From the top: low turbidity, high turbidity.

The future works on the library will focus on further development of underwater rendering algorithms and improving the performance of the physics computations. In terms of graphics, author wants to include suspended, moving particles, volumetric light effects, god-rays and caustics. In terms of physics, the simulator needs parallelisation of geometry-based hydrodynamics, through the use of OpenCL, and an option to use a simplified model following the Fossen equations. Moreover, robot definition will be simplified by implementing a URDF file parser.

ACKNOWLEDGEMENT

This work was part of a project titled "Force/position control system to enable compliant manipulation from a floating I-AUV", which received funding from the European Community H2020 Programme, under the Marie Sklodowska-Curie grant agreement no. 750063.

The work is continued in a project titled "EU Marine Robots", which received funding from the European Community H2020 Programme, grant agreement no. 731103.

REFERENCES

[1] A Birk, T Doernbach, C Mueller, T Luczynski, A G Chavez, D Koehntopp, A Kupcsik, S Calinon, A K Tanwani, G Antonelli, P D Lillo, E Simetti, G Casalino, G Indiveri, L Ostuni, A Turetta, A Caffaz, P Weiss, T Gobert, B Chemisky, J Gancet, T Siedel, S Govindaraj, X Martinez, and P Letier. Dexterous Underwater Manipulation from Onshore Locations: Streamlining Efficiencies for Remotely Operated Underwater Vehicles. *IEEE Robotics and Automation Magazine*, 2018.

[2] Adrian Boeing and Thomas Bräunl. *Evaluation of real-time physics simulation systems*. ACM, New York, USA, December 2007.

[3] Eric Bruneton. A Qualitative and Quantitative Evaluation of 8 Clear Sky Models. *IEEE Transactions on Visualization and Computer Graphics*, 23(12):2641–2655, December 2016.

[4] Eric Bruneton and Fabrice Neyret. Precomputed atmospheric scattering. *Computer Graphics Forum*, 27(4):1079–1086, January 2008.

[5] Eric Bruneton, Fabrice Neyret, and Nicolas Holzschuch. Real-time Realistic Ocean Lighting using Seamless Transitions from Geometry to BRDF. *Computer Graphics Forum*, 29(2):487–496, June 2010.

[6] Patryk Cieslak and Pere Ridao. Adaptive admittance control in task-priority framework for contact force control in autonomous underwater floating manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, October 2018.

[7] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15, New York, NY, USA, 2015. ACM.

[8] Jonathan Dupuy and Eric Bruneton. Real-time animation and rendering of ocean whitecaps. In *SIGGRAPH Asia 2012 Technical Briefs*, pages 1–3, New York, New York, USA, 2012. ACM Press.

[9] Roy Featherstone. Rigid Body Dynamics Algorithms. Springer US, Boston, MA, 2008.

[10] T Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, May 2011.

[11] T I Fossen and T Perez. Marine Systems Simulator (MSS), 2004. https://github.com/cybergalactic/MSS.

[12] N Koenig and A Howard. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154. IEEE, September 2004.

[13] Musa Morena Marcusso Manhães, Sebastian A. Scherer, Martin Voss, Luiz Ricardo Douat, and Thomas Rauschenbach. UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation. In *OCEANS 2016 MTS/IEEE Monterey*. IEEE, September 2016.

[14] M Prats, J Perez, J J Fernandez, and P J Sanz. An open source tool for simulation and supervision of underwater intervention missions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2577–2582. IEEE, October 2012.

[15] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[16] David Ribas, Pere Ridao, Lluís Magí, Narcis Palomeras, and Marc Carreras. The Girona 500, a multipurpose autonomous underwater vehicle. pages 1–5, April 2011.

[17] Michael G Solonenko and Curtis D Mobley. Inherent optical properties of Jerlov water types. *Applied Optics*, 54(17):5392–10, 2015.

[18] Jerry Tessendorf. Simulating ocean water. In *ACM SIGGRAPH Course Notes*. ACM Press, 2001.