UW MORSE

The UnderWater Modular Open Robot Simulation Engine

Eirik Hexeberg Henriksen
Department of Petroleum Technology and Applied Geophysics
Norwegian University of Science and Technology
Trondheim, Norway
eirik.henriksen@ntnu.no

Ingrid Schjølberg
Department of Marine Technology
Norwegian University of Science and Technology
Trondheim, Norway
ingrid.schjolberg@ntnu.no

Tor Berge Gjersvik
Department of Petroleum Technology and Applied Geophysics
Norwegian University of Science and Technology
Trondheim, Norway
tor.b.gjersvik@ntnu.no

***Abstract*—This paper presents an open-source simulation environment for underwater vehicles and robots. The simulation environment allows the user to simulate underwater robotic vehicles with realistic dynamic behavior in a 3-dimensional virtual environment. The environment is highly configurable, and offers a set of modules for simulating different types of vehicles in a number of underwater scenarios. The simulator can be used for control system development, path planning, risk management and testing in a safe virtual environment. The possibility for virtual testing will lower the cost and reduce time of operations. The simulation environment is an expansion of MORSE - Modular Open Robots Simulation Engine. The modular nature of MORSE allows the user to easily configure the simulations, making new environments and robots, as well as adding sensor and control interfaces. This expansion includes modules for hydrodynamic simulation, thrusters and underwater sensors. These modules enables the user to make a virtual replica of a specific underwater robot system. Such replica may be used as a Software-in-the-loop system for testing and verification of control systems and algorithms. The simulation environment allows interaction with a large set of robotic middlewares.**

I. Introduction

Underwater robots are complex systems mechanically, electronically as well as in software. The robots needs to be able to withstand extreme conditions imposed by the ocean environment, the electronics need to be reliable and at the same time able to drive powerful thrusters and collect data from on-board sensors. The software needs to be able to control the vehicle, and make use of sensor data to understand the state of the vehicle. Being able to simulate underwater robots motions, interactions with the environment and sensor output in a realistic fashion enables researchers and developers to develop guidance, navigation and control (GNC) software for underwater robots without the need for advanced hardware to test their innovations. This is expected to lower the cost, making it faster and easier to test new software and making it possible to study how robots interact with the environment without the risk of breaking equipment.

Underwater robotic simulators are not easily available today. This paper will present the work of extending an open source robotic simulator made for simulation of ground and aerial robots, MORSE to be used for underwater robots. Hydrodynamical effects and modules for simulating underwater sensors have been implemented. Including underwater sensors in the simulation makes it possible to test signal processing, and filter algorithms. The MORSE framework has modules for modifying sensor data, such as adding noise or sensor bias. These are effects that need to be taken into account for realistic testing of GNC algorithms.

The work is implemented both as MORSE modules, and as modifications to the Blender Game Engine used by the MORSE simulator. Together with the existing MORSE software this represent a complete open-source suite for simulation of underwater robots.

II. Software Overview

The simulation of underwater robotic vehicles still remains a niche problem, and functionality for simulating hydrodynamic forces and underwater sensors are typically not included in regular robotic simulators. The develpment of a suitable simulator started by defining a set of requirements. These requirements where used when evaluating the capabilities of existing simulators, and later as a guide for development. These requirements are:

- Underwater physics
- Underwater sensors
- Collision dynamics
- Real-Time simulation
- Modularity - Easy to configure and reconfigure
- Software In the Loop capabilities
- Open to any middleware

[1] presents a survey of existing simulation software, and their suitability for simulation of Autonomous Underwater Vehicles (AUV). The paper treats the three most promising simulators in more detail. These three are UWSim [2], MORSE [3] and Gazebo [4]. Neither of these did however fulfill all the defined requirements. UWSim was especially developed to simulate underwater robots, but at the time this development started it was not capable of concurrently simulating hydrodynamic forces and collisions between objects. This functionality has later been added as an add-on module. The process of configuring new simulation scenarios, with new

978-1-5090-5716-0/16/$31.00 ©2016 IEEE

robots or sensors was done by writing an XML-file, and the documentation is deficient. MORSE and Gazebo are both general purpose robotic simulators, and does not come with any underwater features. The documentation is however very adequate, including tutorials on how to set up simulations. The interface for configuring new simulations is quite good. The MORSE simulator seemed especially easy both to configure and to interface with new modules and sensors. It also provides connections to a large number of robotic middlewares, and an interface for adding even more. Most importantly MORSE provides an interface to the MOOS middleware that was designed for underwater robots and that is used by the marine autonomy software MOOS-IvP [5] [6]. MORSE additionally provides interfaces for ROS [7], YARP [8], as well as others. UWSim provides an interface to ROS, and Gazebo currently provide interfaces to Player [9], YARP as well as ROS. It was decided to extend the MORSE simulator to include hydrodynamic effects and underwater sensors, as it seemed to match all the other requirements.

A. *Software Structure*

The simulation software presented in this paper is fundamentally an extension of three existing open-source projects. These projects are Blender, a 3D creation software and a game engine for realtime 3D simulation [10]. Blender Game Engine (BGE) uses the Bullet physics engine to handle the motion and collision between objects [11]. The last software is MORSE - Modular Open Robots Simulation Engine, which provides a framework for using BGE as a robot simulation tool [3]. In the rest of the paper the combination of these three open-source software projects will be referred to as MORSE.

MORSE offers an interface for interacting with the BGE simulation. This is done by defining six main types of software interfaces, or classes of components. These are:

- Robots: The base component that carries sensors and actuators. This component do typically define the physical properties of the simulated robot.
- Sensors: Collects data from the simulated 3d World using an interface to BGE
- Actuators: Produce actions on the robot that carries it. Typically causes motion by setting forces or velocities on the robot by using an interface to BGE.
- Modifiers: The simulated sensors often produce ”perfect data” as the virtual measurements are very accurate and not prone to any kind of noise. The modifiers are modules that are especially designed to alter data coming from the virtual sensors, e.g. to add noise or biases to the measurements, or transform measurements to different reference frames.
- Middlewares: This class of MORSE component serves as an adapter between the sensor- and actuator-components and the middleware software that is used to process the simulated sensor data and control the virtual robot.
- Scenes: The modeled environment where the robot will reside during the simulation

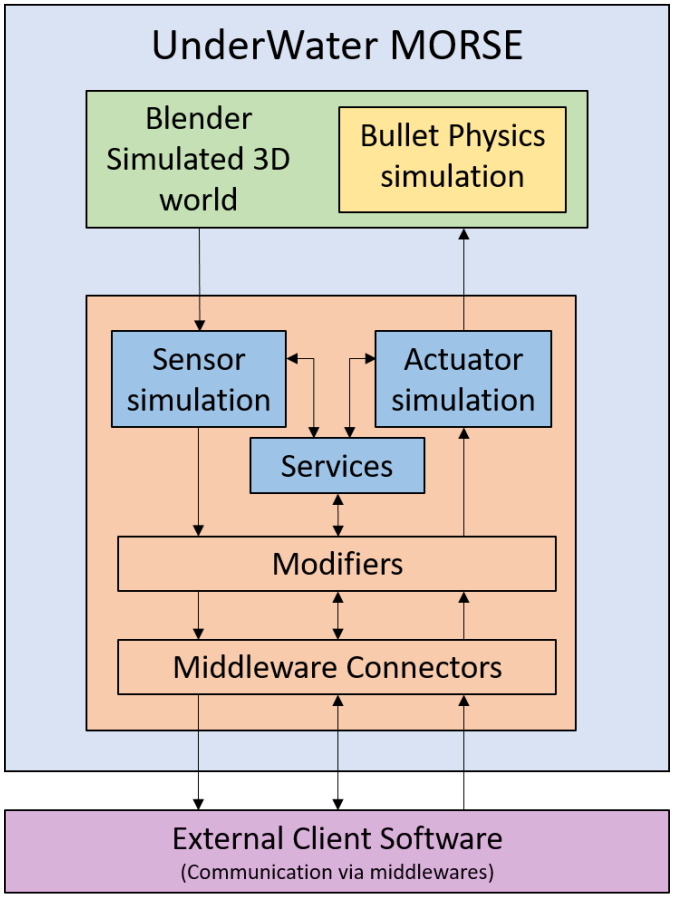

Fig. 1. Software information flow diagram. The sensor output datastream is generated by modifying information from the Blender 3D world. The actuator input datastream is used to simulate actuator responses and these are applied in the Blender 3D world. Communication with sensor and actuator can also be done in a request/reply fashion using services.

There are several components in each class supplied with the simulator, and it is easy to design new ones to fill the needs for a developer or researcher. MORSE also provides an easy method for configuring and combining the components into a simulation - the Builder script. A diagram showing the information flow between modules can be seen in Figure 1.

The two main reasons MORSE in its original form is not suitable to simulate underwater robots is lack of hydrodynamic effects like underwater inertia and drag effects, and the lack of underwater sensor simulation modules. The presented work adds these capabilities to the MORSE simulator, enabling simulation of Underwater Robots using the MORSE framework.

III. Underwater Robot Dynamics

When a rigid body is moving in a high density fluid like water, the fluid-body interaction forces constitutes a large portion of the motion properties for the rigid body. When compared to an object moving in air, a body moving in water will be influence by an added mass effect, as well as higher drag force. These effects are very important to describe the motion of rigid bodies moving underwater, and need to be simulated for a realistic dynamic behavior. The

current available open-source physics engines do however not incorporate these effects.

Game physics engines is software that is made to simulate movement, and collisions between bodies in real time. Morse, and Blender Game Engine uses the Bullet physics engine [11]. The Bullet engine, and most other open source engines are focused on bodies moving in air. The inertial forces due to fluid body interaction is small in air, and is neglected. However in water these effects need to be taken into account to get realistic motions of the body.

A body that is accelerated in any fluid will cause the fluid to gain kinetic energy. This gain in energy will be at the expense of the work done on the submerged body. It is common to model this effect as an added mass, causing newtons second law to be rewritten as:

$$F = m \cdot a \rightarrow F = (m + m_{added}) \cdot a \tag{1}$$

Where the force

$$F_{added} = m_{added} \cdot a \tag{2}$$

is the force needed to accelerate the fluid surrounding the body. The size of this force is only dependent on the shape and size of the body. This causes the added mass to anisotropic, unlike the real mass. The exception is a sphere that is symmetric about all axis. The added mass may also introduce a coupling between translational and rotational accelerations.

The added mass effects can be neglected for bodies moving in air, due to the low amount of energy that is taken up by the air during acceleration. The effect is however very important for describing dynamics underwater. The added inertia due to surrounding water is often in the same magnitude of the inertia due to the body-mass [12]. Bodies moving under water will also experience a buoyancy force as well as coupled drag forces. These effects are currently not implemented in the Bullet physics engine.

A common way to simulate rigid bodies moving in, or under water is to use Fossen's Robot-Like Vectorial Model for Marine Craft [13]. The model captures the most important forces acting on a rigid body in water, and is expressed in a vectorial form for making it easy to use and implement. This model can be found in Equation 3.

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}_{\Theta}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{3a}$$

$$\begin{aligned} \boldsymbol{M}_{RB}\dot{\boldsymbol{\nu}} + \boldsymbol{C}^{*}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} + \\ \boldsymbol{M}_{A}\dot{\boldsymbol{\nu}}_r + \boldsymbol{C}^{*}_{A}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \\ \boldsymbol{D}_{L}\boldsymbol{\nu}_r + \boldsymbol{D}_{NL}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \boldsymbol{g}(\eta) = \boldsymbol{\tau}_{ctrl} + \boldsymbol{\tau}_{ext} \end{aligned} \tag{3b}$$

Where $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$ are the state vectors denoting 6-DOF position and velocity respectively. In this equation the position is given in the NED-frame (North-East-Down). Subscript r on the velocity vector indicates the relative velocity with respect to water. $\boldsymbol{M}$ is a mass matrix, $\boldsymbol{C}$ a Coriolis and centripetal matrix and $\boldsymbol{D}$ is denoting a damping matrix. $\boldsymbol{g}$ indicates restoring forces and moments. Subscripts RB and A indicates rigid body forces, and added mass forces respectively, while L/NL indicates linear/nonlinear damping. Control and external forces are denoted by $\boldsymbol{\tau}$. $\boldsymbol{J}$ is the 6-DOF rotation matrix and is defined in Appendix A. To be able simulate a body moving under water we need to know the parameters used in this model. A method for determining these parameters for underwater robots can be found in [14].

Fossen's Robot-Like Vectorial Model for Marine Craft does however not take into account the physical extent of the body, and the forces it will experience when it collides with other bodies. To get a realistic simulation of underwater bodies with collision dynamics we need to implement the Fossen model in the Bullet physics engine. Previous work from Weißman and Pinkall [15] describes a similar implementation that is used to simulate objects in water.

The Bullet physics engine uses a sequential impulse constraint solver to simulate rigid body motions and collisions [11]. This solver works in the global (world) frame to be able to solve collisions and force interactions between several objects. In order to include 6-DOF damping forces, buoyancy, thruster-forces etc. we translate these into the world frame and apply them to the rigid body in Bullet. The forces that are added in each simulation step are integrated to a velocity impulse, or velocity change for the given time step. The 6-DOF velocity impulse is a function of the forces and moments and the inertia:

$$\Delta\boldsymbol{\nu}_{world} = \boldsymbol{M}^{-1}_{world} \sum \boldsymbol{\tau}_{i,world} \tag{4}$$

Where $\boldsymbol{\nu}_{world}$ is the 6-DOF velocity vector containing both linear and rotational velocities, $\boldsymbol{M}_{world}$ is the combined rigid-body and added mass matrix and $\boldsymbol{\tau}_{i,world}$ denotes all forces and moments experienced by the body. In this equations all quantities are evaluated in the world frame. This means that the inertia matrix and the forces and moments needs to be rotated into the global reference frame. The 6-DOF velocity impulse is then used as input to Bullets solver algorithm.

The MORSE simulator framework is used to configure and interact with the simulation in BGE and Bullet. The configuration is done by using a MORSE actuator module. The module takes advantage of the MORSE interface for easy configuration of the parameters needed simulating fluid effects on the robot.

IV. Simulated Underwater Sensors

An essential part of robotics research is making robots capable of doing automated tasks. To facilitate automated execution of tasks and decision making the robot needs to sense the world. The MORSE simulator already has a comprehensive library containing sensor simulation modules that is typically used for ground and aerial robots. Some of these can be used for underwater simulations as well, like cameras, IMUs (accelerometer, gyro and magnetometer) and possibly laser scanners. However, underwater robots often need different types of sensors. The following sections will present some underwater sensors that is implemented as part of the UW-MORSE project as MORSE simulation modules.

A. Acoustic Positioning Sensors

GPS is not available under water due to the high attenuation of radio signals in water. One of the most applied alternatives that is underwater acoustic positioning systems. Hydro-acoustic positioning is based on the basic triangulation principles, and works by measuring the time it takes for a sound to propagate to a specific point and back. This is then

used to calculate the position [16]. There are two dominating methods for calculating the position of submersibles, namely Ultra Short Base Line (USBL) or Long Base Line (LBL) systems [17].

The USBL system uses two main devices to measure the position of the ROV. This is an *interrogator* and a *transponder*. The interrogator is a combined transmitter and receiver device that sends out an acoustic signal on one frequency and receives a reply on another frequency. This device is typically mounted on a ship, while the transponder is mounted on the underwater robot. The transponder sends out a response signal when a signal is received from the interrogator ([16]). See figure 2 for the system set up. The distance between the robot and the interrogator is calculated by measuring the time of flight of the signal through the water. The bearing angle of the returning signal is calculated using the phase shift between several hydrophones on the interrogator element. Combining the distance and angle makes it possible to calculate the position of the ROV.

An LBL system consist of one or more interrogators, and a set of either transponders or *pingers*. A pinger is a device that continuously sends out an acoustic pulse on a particular frequency ([16]). In this system the interrogator is typically mounted at the ROV, and the interrogator uses the response from the other acoustic beacons to calculate the distance to each device. The devices are typically mounted on the seafloor at known positions. The robot with the interrogator then use trilateration to calculate its own position relative to the beacons. See figure 3 showing the system set up.

The MORSE sensor modules developed to simulate acoustic positioning sensors returns a set of distances between an interrogator and one or more predefined acoustic beacons. There are two types of setups corresponding to the two different types of systems. The LBL sensor module return distances from the sensor to static points, representing static acoustic beacons. The USBL sensor module on the other hand works the opposite way by defining an array of hydrophones mounted on the robot, and one or more static acoustic beacons. This sensor returns distances between the robot-mounted hydrophones and the beacons. It is the up to the navigation system developer to calculate the position.

B. 360° Scanning Sonar and Echosounder Altimeter

A Sonar and an echosounder are both acoustic range sensors. They both send out an acoustic pulse and use the returned echo for range measurements. The echosounder is typically used as an altitude sensor, measuring the height of the underwater robot over the seabed. A sonar is typically used as a range sensor in the horizontal plane. The sonar is typically set up to scan either a sector or the full 360 °circle. The echosounder simply uses the first echo that is received to calculate the distance to the seafloor. A Sonar on the other hand, maps all echoes that is received within a configured range, and returns a vector of echo intensities. Each index in the return-vector corresponds to a range bin, and contains the echo intensity for that bin. This mapping from acoustic beam, to intensity bins is illustrated in figure 4.

The sonar simulation module presented here is based on the Tritech Micron Scanning sonar, and the echosounder

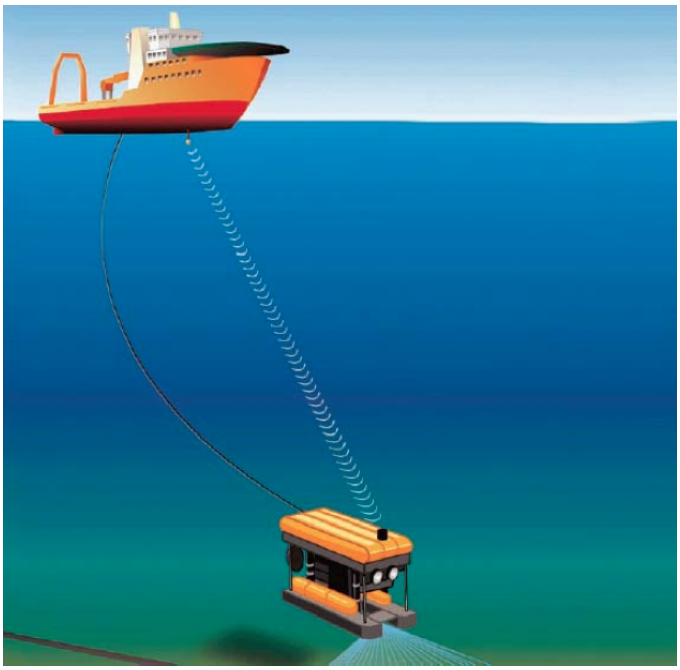

Fig. 2. Ultra Short Baseline acoustic positioning system. An Array of acoustic hydrophones is mounted on the ship, using the phase shift to decide the angle to the transponder mounted on the underwater robot. The range is calculated using the time of flight of the acoustic pulse. Courtesy of Kongsberg Maritime

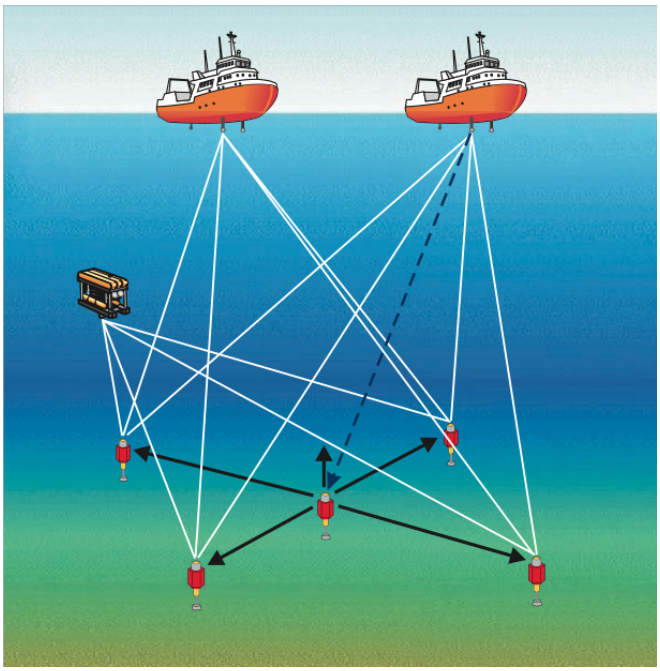

Fig. 3. Long Baseline acoustic positioning system. A set of ranges between the interrogator on the robot and the transponders. The ranges are used to find the relative position of the robot using trilateration. Courtesy of Kongsberg Maritime

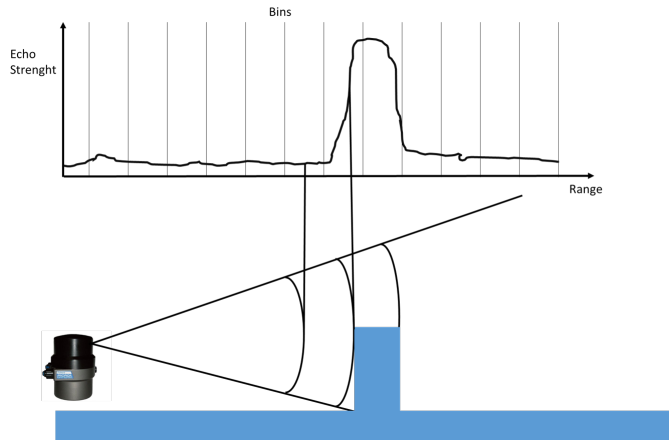

Fig. 4. The lower part shows an acoustic beam being emitted from a sonar head. When the beam hits an object an echo is returned and mapped to a range bin (in the upper part of the figure)

simulation module is based on the Micron Echosounder. These are both small sensors that suit smaller underwater vehicles. The Micron sonar sends out an acoustic beam which is 3° wide in the horizontal plane, and 30° high in the vertical plane. The sonar head is mechanically moved to do 360° scanning. It is also capable of performing scans over smaller sectors.

The simulation module for both the Sonar and the echosounder use raycasting, a method commonly used in computer graphics to find the intersection between a surface and a ray [18]. The method is already implemented in BGE. The echosounder module emits a single ray pointing straight down, and returns the distance to the first object hit by the ray.

The angular spread of the acoustic beam emitted from the sonar is simulated by using an array of rays. The array spreads 3 °in the horizontal plane, and 30 °in the vertical plane, effectively simulating the angular spread of the acoustic ray. The principle can be seen in figure 5. The number of rays that is used to simulate the beam can be configured by the user. This will be a trade-off between realism and computational cost. The simulated echo intensity is estimated by taking the dot product between the surface normal at the point where the ray hits and the unity vector with the same direction as the ray. This is done for all the rays in the array for each scan location. The calculated echo intensity is then put into the vector at the bin corresponding to the distance between the sensor and the surface.

C. Pressure Depth Sensor

A pressure sensor capable of measuring the hydrostatic pressure under water is a very simple, but useful sensor. The hydrostatic pressure can be used to calculate the depth of the underwater robot below the sea surface. The simulation module of this sensor is made by using the position of the sensor on the z-axis to calculate the hydrostatic pressure acording to:

$$p = \rho g h \tag{5}$$

where ρ is the volumetric mass density, g is the gravitational acceleration and h is the distance between the sea surface and the sensor. By default the sea surface is placed in $z = 0$, but this can be changed in the configuration of the senor module.

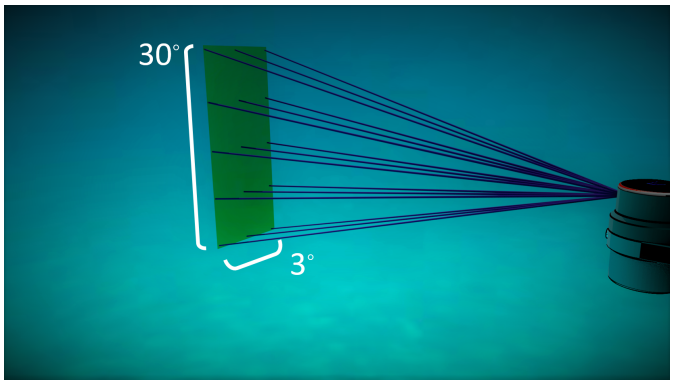

Fig. 5. Ray array used to simulate the angular spread of the acoustic sonar beam. The sonar head (to the left) emits an array of rays with a 3°horizontal spread and a 30°vertical spread. The array simulate one acoustic pulse.

The sensor can also be configured to give depth directly instead of pressure.

V. Applications Scenarios in Underwater Robotics Research

A simulator capable of simulating realistic behavior of underwater vehicles has several applications in underwater robotic research and development. In a development cycle the simulator can be used to test and verify guidance, control and navigation algorithms and systems. The simulator is capable of simulating a large range of sensors, and the modifier system allows the user to add noise and biases to the measurements to test the robustness of the algorithm that is implemented. The MORSE framework allows the use of a large range of robotic middlewares for communication between the control system and the robot. This makes it possible to use the exact same guidance, navigation or control software during simulator testing as for the real robot. This is often called Software in The Loop (SIL) testing. The UW-MORSE simulation software offers the possibility to do SIL-testing of control software in a realistic and safe environment. Simulation testing can be done without the risk of damaging hardware, and even as a test to evaluate what kind of sensors or robots to use for a specific task. This will potentially lower both the hardware cost, and lower the development time. Figure 6 shows a scenario where control algorithms for automated inspection of subsea equipment are tested.

A realistic simulation of an underwater operation will enable easier, better and more efficient planning. Using the simulator as a planning tool will be especially efficient for non-routine operations. The pilots operating the underwater robots and other equipment involved in the mission can also use the simulator to practise on specific parts of the mission to be better prepared.

UW-Morse is also capable of visualizing data that is taken during ocean or pool testing of underwater robots. Many middlewares has the possibility to replay log files from such missions. The simulator can then be configured to visualise the the data in a 3D-world. A typical scenario would be to visualise the performance of a navigation filter. This can be done by using several robots in the simulation and make one follow the

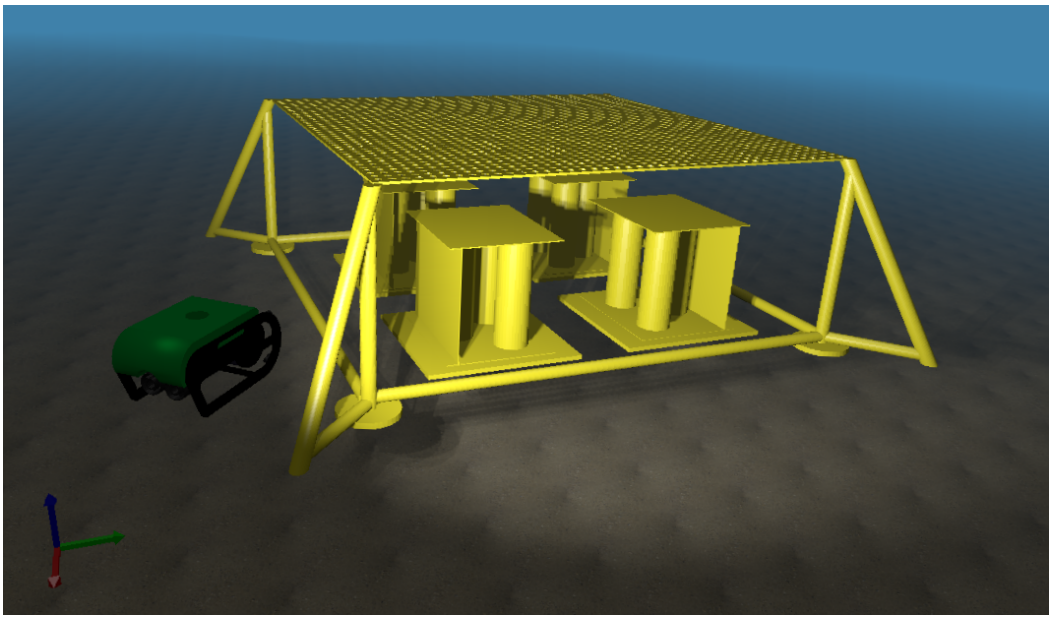

Fig. 6. A screenshot from a simulation where an underwater robot is used for inspection of subsea equipment for petroleum extraction.

sensor data, while another one is following the output from the navigation filter.

Additional uses for realistic underwater simulations is virtual prototyping of underwater equipment. Being able to put virtual prototypes - 3D-models in an underwater scenario where one can test the interaction with other systems, like other underwater robots or other underwater equipment might speed up the development and can increase the chance of success.

VI. Conclusion

A simulator environment has been developed in order to offer researchers and students the possibility to develop, test and verify control algorithms for underwater vehicles, and has a whole range of possible usage scenarios. This ranges from visualization, virtual rapid prototyping, testing designs and verification of underwater equipment, as well as underwater vehicle pilot training. The simulation environment provides a fast, easy and most importantly an affordable alternative to pool and ocean testing. This makes development of underwater robotic solutions faster and less expensive.

The development of the simulation environment is still in progress, and there is room for developing both more and better modules for simulating sensors, actuators and robots. A very exciting area of future work is to develop robotic manipulator arms that can interact with the environment, using the bullet physics engine API for making links between rigid bodies.

VII. Acknowledgements

The work presented in this paper is founded by the Research Council of Norway, Statoil and FMC Technologies through the project Next Generation Subsea Inspection, Maintenance and Repair, 234108/E30. The project is associated with CoE AMOS, 223254.

References

[1] D. Cook, A. Vardy, and R. Lewis, "A survey of auv and robot simulators for multi-vehicle operations," in *2014 IEEE/OES Autonomous Underwater Vehicles (AUV)*, Oct 2014, pp. 1–8.

[2] M. Prats, J. Prez, J. J. Fernndez, and P. J. Sanz, "An open source tool for simulation and supervision of underwater intervention missions," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 2577–2582.

[3] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular openrobots simulation engine: Morse," in *Proceedings of the IEEE ICRA*, 2011.

[4] Open Source Robotics Foundation. (2016) Gazebo - robot simulation made easy. [Online]. Available: http://gazebosim.org/

[5] P. M. Newman, "MOOS - mission orientated operating suite," Cambridge, MA: Department of Ocean Engineering, MIT, Tech. Rep. OE2003-07, 2003.

[6] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard, "Nested autonomy for unmanned marine vehicles with moos-ivp," *Journal of Field Robotics*, vol. 27, no. 6, pp. 834–875, 2010.

[7] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[8] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: Yet another robot platform," *International Journal on Advanced Robotics Systems*, 2006.

[9] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *In Proceedings of the 11th International Conference on Advanced Robotics*, 2003, pp. 317–323.

[10] Blender Foundation. (2016) Blender. [Online]. Available: http://www.blender.org

[11] E. Coumans, "Bullet physics simulation," in *ACM SIGGRAPH 2015 Courses*, ser. SIGGRAPH '15. ACM, 2015.

[12] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd, 2011.

[13] T. I. Fossen, "Nonlinear modeling and control of underwater vehicles," Ph.D. dissertation, Department of Engineering Cybernetics, Norwegian University of Science and Technology. Trondheim, Norway, 1991.

[14] O. A. Eidsvik and I. Schjølberg, "Determination of hydrodynamic parameters for remotely operated vehicles," in *Proceedings of the ASME 2016 35th International Conference on Ocean, Offshore and Arctic Engineering*. ASME, 2016.

[15] S. Weißmann and U. Pinkall, "Underwater rigid body dynamics," *Proceedings of ACM SIGGRAPH 2012*, vol. 31, no. 4, pp. 104:1–104:7, Jul. 2012.

[16] R. Christ and R. Wernli, *The rov manual - a user guide for observation-class remotely operated vehicles*. Elsevier, 2007.

[17] M. Ludvigsen, "An rov toolbox for optical and acoustical seabed investigations," Ph.D. dissertation, Department of Marine technology, Norwegian University of Science and Technology. Trondheim, Norway, 2010.

[18] S. D. Roth, "Ray casting for modeling solids," *Computer Graphics and Image Processing*, vol. 18, no. 2, pp. 109 – 144, 1982.

APPENDIX

A. Rotation and Rigid Body matrices

1) Rotation matrix:

$$\boldsymbol{J_\Theta(\eta)} = \begin{bmatrix} \boldsymbol{R_b^n(\Theta_{nb})} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \boldsymbol{T_\Theta(\Theta_{nb})} \end{bmatrix} \tag{6}$$

The rotation matrix for the linear velocity $\boldsymbol{R_b^n(\Theta_{nb})}$, and the rotation matrix for the angular velocities $\boldsymbol{T_\Theta(\Theta_{nb})}$ are given by:

$$\boldsymbol{R_b^n(\Theta_{nb})} = \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\theta \\ s\psi c\theta & c\psi c\theta + s\phi s\theta s\psi & -c\psi s\theta + s\theta s\psi c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{7}$$

$$\boldsymbol{T_\Theta(\Theta_{nb})} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & \frac{s\phi}{c\theta} & \frac{c\phi}{c\theta} \end{bmatrix} \tag{8}$$

where $c(\cdot) = cos(\cdot), s(\cdot) = sin(\cdot)$ and $t(\cdot) = tan(\cdot)$.