Journal of Marine Science and Technology, Vol. 28, No. 5, pp. 422-432 (2020)
DOI: 10.6119/JMST.202010_28(5).0012

MOTION AND CONTROL SIMULATION FOR UNDERWATER ROBOTS

Shyh-Kuang Ueng and Chieh-Shih Chou

Key words: remotely operated vehicles, hydrodynamic motion simulation, auto-piloting, underwater robots.

ABSTRACT

Remotely Operated Vehicles (ROVs) are widely used in underwater explorations and constructions. When navigating under the sea, the motions and stabilities of ROVs are influenced by various forces. Operating ROVs to capture images, grasp samples, avoid obstacles, and examine man-made facilities is challenging. This paper proposes an innovative simulator to train users to handle ROVs. Conventional simulators only can mimic undeformable ROVs. Their educational capabilities are limited. The proposed simulator is developed based on advance mathematics models and is able to emulate a ROV equipped with movable robot arms. Therefore, it can perform more realistic and meaningful simulations to enhance the training courses. We deduce a voxel-based method to calculate the mass properties of the ROV. Thus, the hydrodynamic effects caused by its robot arms can be calculated in real time. A numerical algorithm is also invented to coordinate the propellers to produce required forces and moments. Hence, the ROV can attain specified velocities, positions, and orientations under the influences of loads, sea currents, and moving robot arms. Furthermore, an artificial intelligence engine is integrated into the system to perform auto-piloting, auto-balancing, and command interpretation such that the simulator is more user friendly. Besides, the system is augmented with an interactive user interface and a graphics engine to display simulation progressions and information to increase its usability.

I. INTRODUCTION

A Remotely Operated Vehicle (ROV) is an unmanned underwater vehicle equipped with propellers, cameras, light sources, sensors, and robot arms. ROVs can navigate and perform designated tasks in deep water where human divers are unable to reach. The applications of ROVs include undersea constructions, mining, investigation, and monitoring (Manley, 2008). In recent decades, numerous ROVs have been built and widely used for exploring underwater resources.

However, piloting a ROV to conduct a mission is challenging. When moving below the sea surface, the motions and stability of a ROV are affected by various external forces. The operator has to precisely manipulate the propellers to counter these influences such that the ROV can move steadily. Furthermore, obstacles and man-made facilities may reside in the working place. The operator must drive the vehicle with care to avoid colliding with these objects. In addition to overcoming these hazards, the operator may have to control the robot arms to grasp samples, tune the lights to illuminate the surroundings, and use the cameras to capture feedback images at the same time. Obviously, handling a ROV is a multi-tasking process in nature and requires sophisticated skills. People should have undertaken intensive training courses before operating ROVs (Christ and Wemli, 2007).

Training users to pilot ROVs by using real facilities is very expensive and dangerous. Instead, people employ simulators to fulfill this job. Compared with physical machines, simulators are more convenient and flexible. The trainers can easily modify the simulation settings to create new subjects for the students. More importantly, since all exercises are carried out in virtual worlds, the learners are able to repeatedly practice each skill with lower costs and less risk. Thus, they will possess the capabilities to handle ROVs in a short time (Agba, 1995). Besides serving as teaching equipment, simulators have also been used to study the characteristics of ROVs, verify control algorithms, and rehearse mission plans to uncover errors and prevent dangers.

1. Related Work

Many simulators had been constructed to train people to handle ROVs. Some of them were designed by integrating hardware and software to improve simulation fidelities (Christensen et al., 2009; Xu et al., 2016). Nonetheless, the operational and building costs of these systems are high, and their flexibilities are limited, compared with pure virtual simulators. Other researchers developed ROV simulators which could generate hydrodynamic motions and display simulation progressions in real time (Fabekovic et al., 2007). The successes of these software rely on the underlying kinetic models and graphical display methods, which were not well explained in the work. A useful simulator should be able to emulate

Paper submitted 01/09/20; revised 05/05/20; accepted 07/07/20. Corresponding Author: Shyh-Kuang Ueng(Email: skueng@mail.ntou.edu.tw)
Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung, Taiwan

different kinds of ROVs. In (Kim, 2014), Kim presented a configurable simulator, which allows users to tune the embedded components to reflect the characteristics of the target ROV. However, implementation details are absent from his paper.

Some researchers were interested at developing simulators to evaluate ROV controlling algorithms. In the work of (Lee et al., 2009), Lee et al. constructed a simulation program to study ROV control methods. In the experiments given in (Miskovic et al., 2006), a specialized micro-ROV is used for comparing auto-piloting methods. Tehrani et al. built a micro-ROV, which was used as a platform for testing autonomous underwater vehicles (Tehrani et al., 2010). In the work of (Hsu et al., 2000), Hsu et al. constructed a small ROV to verify their auto-position systems. In (De Souza and Maruyama, 2007), researchers used simulation programs to analyze controlling algorithms for positioning ROVs under the influences of sea currents, tension of tethers, and other hydrodynamic forces. These simulators aim to reveal the properties of ROVs or to test controlling schemes. They lack visualization capabilities and user interfaces. Hence, they are unsuitable for job-training.

Modern ROVs are usually equipped with movable robot arms. Their stabilities and motions are influenced by the movements of the robot arms. In (Featherstone, 1984; Featherstone, 2014), fast algorithms had been invented to compute the dynamics and mass inertia of robots fixed at the bases. In these two researches, a robot is regarded as an articulated body composing of joints and links. Each joint possesses a degree of freedom, influenced by the external forces acting on the links and the internal forces propagating from other joints. At each time step, the accelerations of the joints are computed from the base outward to the end joint. Then, the inertias are updated in a backward manner to simulate motions of the robots.

2. Research Motivation and Summary

All the aforementioned simulators lack the ability to emulate ROVs equipped with moving robot arms. In some of these systems, their ROVs have no movable part at all. Hence, the authors did not have to worry about the effects produced by robot arms. In other simulators, the embedded ROVs do contain robot arms. As the robot arms maneuver, the gravity and buoyancy centers migrate to new positions. The resulting buoyance forces will produce torques and rotate the ROVs. However, these researchers ignored this problem and took no action to attain the ROVs' stabilities.

The proposed simulator is developed based on advance hydrodynamics models and is able to emulate a ROV equipped with movable robot arms. It can perform more realistic and meaningful simulations to enhance the training courses. A good simulator should be able to calculate all acting forces in real time. In this work, we develop a voxel-based strategy to precisely compute the tensor of inertia and the gravity and buoyancy centers on the fly such that the hydrodynamic forces created by the moving robot arms can be efficiently computed.

When the moving robot arms cause the ROV to rotate, we rely on the thrusts of the propellers to keep the ROV in balance. In the proposed simulator, we design a numerical algorithm to coordinate the propellers to generate the desired moments to counter the rotations. Hence, the posture and position of the ROV can be maintained. In addition, this propeller control algorithm is also utilized to create forces and torques to attain specified velocities, positions, and orientations for the ROV in auto-piloting and auto-balancing simulations.

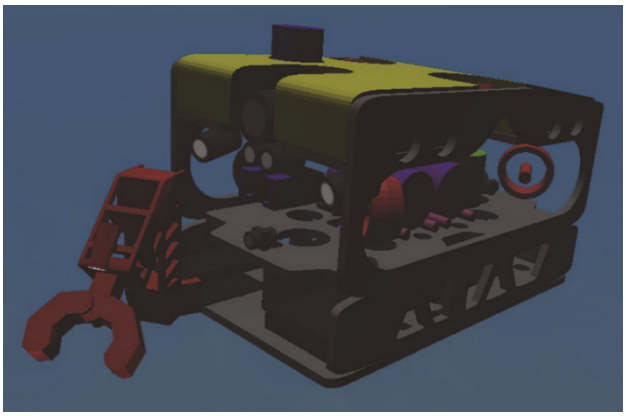


Fig. 1. The virtual ROV resembles a physical machine.

In a ROV simulator, the actuator is responsible for converting high-level commands into low-level actions. Intrinsic algorithms for implementing the actuator had not been well explained in the previous work. In this article, the method of implementing the actuator is also presented. Lacking decent visual effects and user-friendly interfaces is a common problem in conventional ROV simulators, since most of these systems were developed for verifying controlling algorithms and studying ROV characteristics. To remedy this problem, our simulator integrates information processing subroutines, interactive user interfaces, and graphics display modules to enhance its user interface and visual effects.

II. SYSTEM ARCHITECTURE

The proposed ROV simulator is developed based on game programming paradigm (Gregory, 2017) to improve system integration and efficiency. It contains a physics engine (Bourg and Bywalec, 2013), an Artificial Intelligence (AI) engine (Schwab, 2009), a graphics engine, and a Human Computer Interaction (HCI) interface. These computational procedures are used to operate a virtual ROV, which serves as the simulation platform.

1. The Virtual ROV Platform

The virtual ROV resembles a modern underwater robot. It is composed of a structure frame, a float, five cameras, six propellers, two light sources, a robot arm, and several waterproof boxes, which enclose electronic and electrical devices. The image of the virtual ROV is shown in Fig. 1.

The main camera is mounted in the front face of the frame.

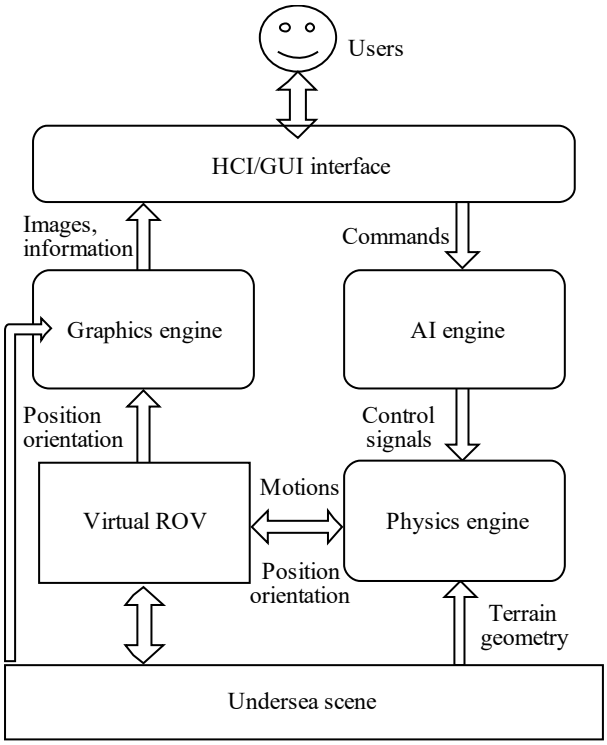


Fig. 2. System architecture of the ROV simulator.

Three extra cameras are located at other sides of the frame to monitor the surroundings. Another camera is attached to the robot arm to catch feedback images for operating the robot arm. The robot arm contains three joins, two links, and one grasper. In total, it possesses four degrees of freedom. Four horizontal propellers are mounted at the four bottom corners of the frame, and two vertical propellers are attached to the left and right sides of the frame. The propeller blades can rotate clockwise and counter-clockwise at varying angular speeds to generate different thrusts. Compared with those virtual ROVs employed in conventional simulators, our machine is more realistic and advanced. It is an effective facility for training users to handle modern ROVs.

2. Architecture of the Simulator

The architecture of the simulator is illustrated in Fig. 2. The HCI allows users to drive the ROV, control the robot arm, turn on/off the lights, and manipulate the cameras by using the keyboard, mouse, and joysticks. The HCI also provides a graphical user interface (GUI), which is composed of several menu systems, widgets, and canvas for function selection, command input, image display, and data visualization.

The AI engine is responsible for converting users' commands and high-level control signals into physical actions for the propellers, cameras, robot arm, and light sources. The AI engine also supports auto-piloting and auto-balancing functionalities such that the users can focus on operating the robot arm without worrying the influences of the sea currents and rotational torques.

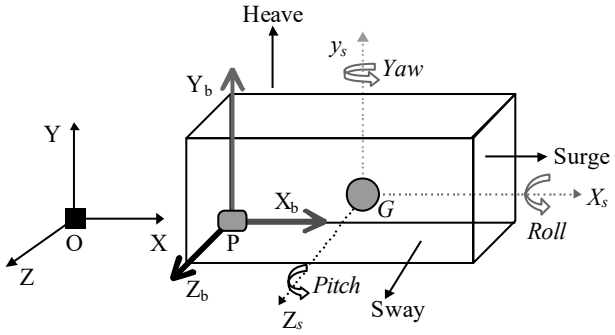


Fig. 3. The 3 coordinate systems and 6 DoF motions. The box represents the ROV.

The physics engine is used to compute mass properties, forces, and moments during the simulation process. It is also responsible for calculating accelerations and velocities and modifying the position and orientation of the ROV. The graphics engine is employed to render the scene and depict essential information during the simulations. The scene are displayed from the view angles of the main camera, the side cameras, the robot arm camera, and a bystander. The information revealed by the graphics engine include the position, orientation, speed, and trajectory of the ROV. In auto-piloting and auto-balancing simulations, this module is utilized to illustrate the progressions of these processes as well as the variations of the ROV's position, orientation, and velocity.

III. MASS PROPERTY COMPUTATION

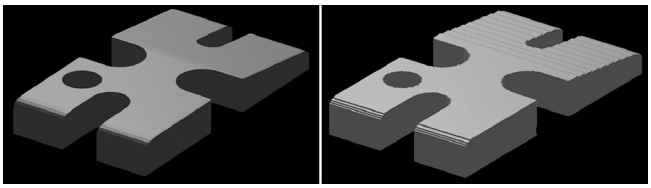
Since the ROV contains a movable robot arm and other irregular-shaped components, to compute its mass properties in real time is not easy. Thus, we develop an innovative algorithm to accomplish this goal. In this section, we introduce the coordinate systems adopted in the proposed simulator at first. Then, the details of the algorithm are formulated.

1. The Coordinate Systems

The proposed simulator uses three coordinate systems to specify the underwater scene, model the ROV, and solve kinetics and kinematics equations. Similar approaches were adopted in the researches of (Ueng et al., 2008; Ueng, 2013).

These three coordinate systems are depicted in Fig. 3. The world coordinate system is used to specify the undersea scene. Its X and Z axes span the horizontal plane while its Y axis points vertically to the sky. The center of the motion coordinate system is located at the gravity center, **G**. Its X_s axis points to the bow of the ROV while its Z_s axis is directed to the starboard. The origin of the body coordinate system resides on the lower right corner of the rear face of the bounding box (BBox). Its three axes are parallel to the faces of the BBox. The body and the motion coordinate systems share the same axes.

The body coordinate system is utilized to model the ROV. The gravity and buoyance enters are designated in this space.



(a) Polygonal model (b) Voxel model

Fig. 4. Voxelization of the float.

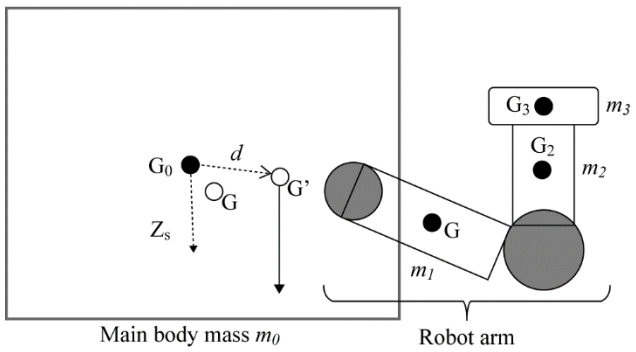


Fig. 5. The four super-voxels, which constitute the main body and the links of the robot arm.

Positions of fixed components in the ROV are invariant in the body coordinate system when the ROV moves. Motions of the ROV are specified in the motion coordinate system. Surge, heave, and sway are translate motions along the X_s, Y_s, and Z_s axes. Roll, yaw, and pitch are rotations about these axes. Forces and moments acting on the ROV are calculated in this space. Accelerations, velocities, and motions are also computed there. The resultant motions are transformed to the world space to update the position and gesture of the ROV.

2. Voxelization and Mass Property Computation

Before initiating the simulation procedure, some mass properties of the ROV have to be calculated. These data include the gravity center G, the buoyancy center B, the mass and tensor of inertia of the ROV. The volume of the ROV is essential for computing the buoyance force and is also computed at this stage.

In this work, we propose a voxel-based method to estimate these mass properties. At first, all the parts of the ROV are decomposed into equal-sized cubes. In literatures, these cubes are called *voxels* (Huang et al., 1998). Then the mass of each voxel is determined according to the constituting material. An example of the voxelization process is shown in Fig. 4. The original float and its voxelization results are displayed in the left and right images respectively.

After the voxelization process, the mass and volume of the ROV are obtained by integrating the masses and volumes of the voxels. The gravity and buoyancy centers are computed by using the following equations:

$$G = \frac{\sum m_i \overline{r_i}}{\sum m_i}, \quad B = \frac{\sum v_i \overline{r_i}}{\sum v_i}. \tag{1}$$

Where m_i, $\boldsymbol{r_i}$, and v_i are the mass, position, and volume of the i-th voxel.

Then the tensor of inertia $\boldsymbol{I}$ is computed by using the formulas listed below:

$$\begin{aligned} &I_{xx} = \sum \left(y_i^2 + z_i^2\right) m_i,\ I_{yy} = \sum \left(x_i^2 + z_i^2\right) m_i, \\ &I_{zz} = \sum \left(x_i^2 + y_i^2\right) m_i, \\ &I_{xy} = \sum x_i y_i m_i, I_{xz} = \sum x_i z_i m_i, I_{yz} = \sum y_i z_i m_i. \\ &\overline{\overline{I}} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}. \end{aligned} \tag{2}$$

Where x_i, y_i, and z_i are the coordinates of the i-th voxel in the motion coordinate system. In this work, the tensor of inertia is computed about **G** and the axes of the motion coordinate system.

3. Super-voxels and Run-time Mass Property Updating

As the robot arm moves, **G** and **B** change. Their positions and the tensor of inertia have to be recomputed. If we use Equations (1) and (2) to update these parameters, it would be too slow for a real-time simulation. Instead, we adopt the parallel-axis and parallel-plane theorems given in (Hibbeler, 2017) to speed up the computations.

We group all the voxels to create four *super-voxels*. The first super-voxel contains the voxels of all non-movable parts. The second, third, and fourth super-voxels are composed of those voxels forming the upper link, lower link, and grasper of the robot arm. Then, the mass properties and volumes of these four super-voxels are computed based on Equations (1) and (2). In this article, the masses of these super-voxels are denoted as M_0, M_1, M_2, and M_3, their volumes are represented by V_0, V_1, V_2, and V_3, their buoyancy centers are designated as $\mathbf{B_0}$, $\mathbf{B_1}$, $\mathbf{B_2}$, and $\mathbf{B_3}$, and their mass centers are specified as $\mathbf{G_0}$, $\mathbf{G_1}$, $\mathbf{G_2}$, and $\mathbf{G_3}$, as shown in Fig. 5.

At the run time, we apply the following formula to update the global gravity and buoyancy centers when the robot arm moves:

$$G = \frac{\sum M_i G_i}{\sum M_i}, B = \frac{\sum B_i V_i}{\sum V_i} \tag{3}$$

After **G** has been revised, the tensor of inertia of each super-voxel is computed by using the parallel axis theorem and the parallel plane theorem (Hibbeler, 2017) as follows:

$$\begin{aligned}
&\vec{d}_G = \begin{bmatrix} d_x & d_y & d_z \end{bmatrix}^T = G - G_i. \\
&I_{xx}^{new} = I_{xx}^{old} + M_i(d_y^2 + d_z^2), \\
&I_{yy}^{new} = I_{yy}^{old} + M_i(d_x^2 + d_z^2), \\
&I_{zz}^{new} = I_{zz}^{old} + M_i(d_x^2 + d_y^2), \\
&I_{xy}^{new} = I_{xy}^{old} + M_i d_x d_y, \\
&I_{xz}^{new} = I_{xz}^{old} + M_i d_x d_z, \\
&I_{yz}^{new} = I_{yz}^{old} + M_i d_y d_z.
\end{aligned} \tag{4}$$

Where $\boldsymbol{d_G}$ is the displacement vector from the global gravity center to the gravity center of the i-th super-voxel, as shown in Fig. 5. As the tensors of inertia of the four super-voxels have been updated, the tensor of inertia of the whole ROV is computed by:

$$\bar{\bar{I}} = \bar{\bar{I}}_0 + \bar{\bar{I}}_1 + \bar{\bar{I}}_2 + \bar{\bar{I}}_3. \tag{5}$$

Where $\boldsymbol{I_i}$ is the tensor of inertia of the i-th super-voxel.

IV. THE PHYSICS ENGINE

The physics engine is responsible for computing forces, moments, accelerations, velocities, and motions at each time step. The kinematics models employed for accomplishing these duties are presented in this section, followed by the methods for estimating drag forces and resistant moments.

1. The Kinematics Models

The six DoF motions are illustrated in Fig. 3. All the motions are specified in the motion coordinate system. We use the Newton's second law to calculate the linear accelerations and velocities and apply the Euler equations to compute the angular accelerations and velocities (Nahon, 1996).

The forces acting on the ROV include the buoyancy force $\boldsymbol{F_B}$, the drag force $\boldsymbol{F_d}$, the propellers' thrusts $\boldsymbol{T}$, the gravity force (weight of the ROV) $\boldsymbol{W}$, and the force of the sea current $\boldsymbol{F_c}$. These forces are summed up to produce the net force $\boldsymbol{F}$. In term, $\boldsymbol{F}$ is used to calculate the acceleration $\boldsymbol{a}$. In the following step, the linear velocity $\boldsymbol{v}$ is modified and used to update the position of the ROV:

$$\begin{aligned}
&\vec{F} = \vec{F}_B + \vec{F}_d + \vec{T} + \vec{W} + \vec{F}_c, \\
&\vec{a} = \vec{F} / M, \\
&\vec{v}(t+h) = \vec{v}(t) + h\vec{a}, \\
&\vec{d} = h\vec{v}(t+h).
\end{aligned} \tag{6}$$

Where $\boldsymbol{d}$ represents the linear motions (surge, heave, and sway) and M is the mass of the ROV.

Once the linear motions have been computed, the moments produced by the buoyancy force, propeller thrusts, and drag force are estimated. The sum of these moments is used to solve the angular acceleration. Then, the angular velocity is calculated and the Euler angles of the ROV is renewed:

$$\begin{aligned}
&\vec{N} = \vec{N}_B + \vec{N}_T + \vec{N}_d, \\
&\vec{\alpha} = (\bar{\bar{I}})^{-1}(\vec{N} - \vec{\omega}(t) \times (\bar{\bar{I}}\vec{\omega}(t))), \\
&\vec{\omega}(t+h) = \vec{\omega}(t) + h\vec{\alpha}, \\
&\vec{\theta}(t+h) = \vec{\theta}(t) + h\vec{\omega}(t+h).
\end{aligned} \tag{7}$$

The term $\boldsymbol{N}$ is the net moment. $\boldsymbol{N_B}$, $\boldsymbol{N_T}$, and $\boldsymbol{N_d}$ are the moments produced by the buoyancy force, propeller thrusts, and drag force. Vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ are the angular acceleration and velocity. The three Euler angles are represented by vector $\boldsymbol{\theta}$. In each step, the orientation of the ROV is changed by using $\boldsymbol{\theta}$.

2. Drag Forces and Moments

As the ROV moves under the sea, the surrounding water produces drag forces to slow it down. The combined drag force is determined by using an empirical equation given in (Bourg and Bywalec, 2013):

$$F_d = \tfrac{1}{2} C_d \rho u^2 A. \tag{8}$$

Where C_d, ρ, u, and A are the drag coefficient, water density, speed of the ROV, and projection area of the ROV in the direction of motion. The acting direction of $\boldsymbol{F_d}$ is opposite to the direction of motion.

As the ROV rotates, the water produces a reaction torque to wear down its rotation. We adopt the empirical formula, proposed by (Bourg and Bywalec, 2013), to estimate the reaction torque:

$$N_d = \tfrac{1}{2} C_r \rho \omega^2 A. \tag{9}$$

Where C_r and ω are the reaction moment coefficient and the angular speed of the ROV. $\boldsymbol{N_d}$ acts in the opposite direction of the angular velocity.

V. THE AI ENGINE

The AI engine is used to translate high-level control commands into physical actions. It also supports auto-piloting and auto-balancing. In this section, we present the propeller vector concept for modelling the propulsions of the propellers at first. Then, based on this concept, a numerical method is deduced to coordinate the propellers to produce desired forces and moments. Following this numerical procedure, the command translation and auto-piloting and balancing methods are formulated.

1. The Concept of Propeller Vector

In this work, we use the *propeller vector* $\boldsymbol{f_i}$ to represent the thrust of the i-th propeller when it rotates counterclockwise at

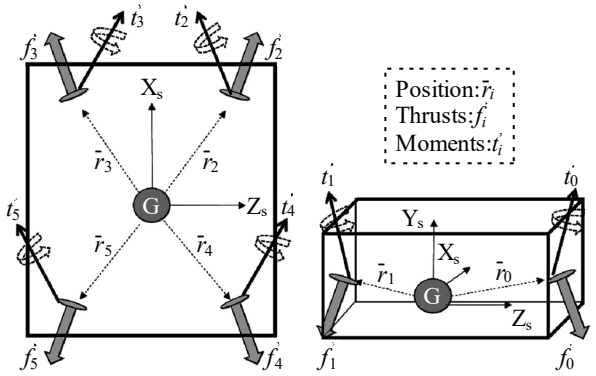


Fig. 6. Positions, thrusts, and moments of the six propellers.

the standard angular speed. The propeller vector $\boldsymbol{f_i}$ is a constant and never changes. The standard moment $\boldsymbol{\tau_i}$ created by the i-th propeller is

$$\vec{\tau}_i = \vec{r}_i \times \vec{f}_i. \tag{10}$$

Where $\boldsymbol{r_i}$ is the positional vector of the i-th propeller. The positions, thrusts, and moments of the six propellers in the ROV are shown in Fig. 6, where the torques are denoted as $\boldsymbol{t_i}$.

2. The Propeller Coordinate Method

Assuming that a force $\boldsymbol{f}$ has to be generated, the following equation can be used to calculate the required thrust of each propeller:

$$\vec{f} = \sum_{i=0}^{5} c_i \vec{f}_i. \tag{11}$$

Where c_i is the scaling factor of $\boldsymbol{f_i}$ and has to be solved such that the target force can be produced. In this work, c_i is called the *propeller coefficient* of the i-th propeller. If c_i equals 1, the propeller operates in the standard mode. Otherwise, the angular speed of the propeller is adjusted and the output thrust is changed.

Since forces are 3D vectors, Equation (11) represents a 3×6 linear system. It contains three conditions and six unknowns and is thus under-constraint. In order to form a well-defined linear system, we force the propellers to generate a moment $\boldsymbol{\tau}$:

$$\vec{\tau} = \sum_{i=0}^{5} c_i \vec{\tau}_i = \sum_{i=0}^{5} c_i (\vec{r}_i \times \vec{f}_i). \tag{12}$$

Moments are 3D vectors. Therefore, Equation (12) is a 3×6 linear system too. By combining Equations (11) and (12), we form a 6×6 linear system, which can be solved to produce the target force $\boldsymbol{f}$. However, $\boldsymbol{\tau}$ has to be set to 0 since no torque is needed.

If the propellers are commanded to generate a moment $\boldsymbol{\tau}$, the force of Equation (11) is set to 0 and the moment of Equation (12) is set to $\boldsymbol{\tau}$. Then, by solving Equations (11) and (12) together, we can compute the propeller coefficients for producing $\boldsymbol{\tau}$.

3. The PID Controller

In underwater vehicles, Proportional Integral Derivative (PID) controllers (Schwab, 2009) are widely used for auto-piloting and auto-balancing. In a PID controller, the user specifies a target value and then the controller continuously measures the state of the system to calculate the error term *e(t)*, where *t* is the time variable. The error term represents the difference between the target value and the measured one. Then, *e(t)*, the integration of *e(t)*, and the derivative of *e(t)* are combined to generate the control signal *u(t)*:

$$u(t) = K_p e(t) + K_i \int_0^t e(s)\,ds + K_d \frac{de(t)}{dt}. \tag{13}$$

Where K_p, K_i, and K_d are the gains of the proportional, integral, and derivative terms. They are given by the users.

In the proposed simulator, the AI engine uses a PID controller to generate control signals to accomplish auto-piloting and auto-balancing. The error term *e(t)* in Equation (13) represents the differences between the target and the observed positions, orientations, or velocities. The control signal *u(t)* is utilized to produce fixations for these differences. It can be a displacement, an angle, or a velocity, depending on the usage of the PID controller. The meanings of *u(t)* and details of the controlling methods are to be formulated in the following subsections.

4. Auto-piloting

In an auto-piloting process, the operator sets a target status and asks the ROV to reach this goal. At the following time steps, the AI engine measures the state of the ROV to calculate the error *e(t)* and sends *e(t)* to the PID controller to produce the control signal *u(t)*. Then, the AI engine uses *u(t)* to compute the required thrusts and moments for reaching the target status.

If the target is a position, the control signal *u(t)* represents the displacement vector. The thrust force $\boldsymbol{f}$ for achieving the target position can be derived as follows:

$$\begin{aligned} \vec{v}_1 &= u(t) / \Delta t, \\ \vec{a} &= (\vec{v}_1 - \vec{v}_0) / \Delta t, \\ \vec{f} &= M\vec{a}. \end{aligned} \tag{14}$$

Where $\boldsymbol{v_1}$, $\boldsymbol{v_0}$, $\boldsymbol{a}$ and M are the new linear velocity, current linear velocity, acceleration, and mass of the ROV. The variable Δt represents the remaining time to reach the target. The rationale behind this method can be explained as follows: We compute the new velocity by dividing *u(t)* with Δt. Then, the acceleration is calculated by using the current velocity $\boldsymbol{v_0}$ and the new velocity $\boldsymbol{v_1}$. By using Newton's 2nd law, we estimate the required force $\boldsymbol{f}$.

Table 1: Propeller vectors of the six propellers.

propellers	positions	propeller vector
0	(0.375,-0.085,-0.2)	(0.866,0.0,-0.5)
1	(0.00, 0.160,-0.21)	(0.0,-0.966, 0.258)
2	(0.40, -0.085,-0.2)	(0.866, 0.0, 0.5)
3	(0.375,-0.085, 0.2)	(0.866,0.0, 0.5)
4	(0.0, 0.16, 0.21)	(0.0,-0.966,-0.258)
5	(-0.40,-0.085, 0.2)	(0.866,0.0, -0.5)

Table 2: Propeller coefficients of motions.

Motions	Propeller coefficients {c0, c1, c2, c3, c4, c5}
Surge	{0, 0, 1, 1, 1, 1}
Heave	{1, 1, 0, 0, 0, 0}
Sway	{0, 0, -1, 1, 1, -1}
Roll	{1,-1,0.259,-0.259,-0.259, 0.259}
Yaw	{0, 0, 1, -1, 1, -1}

At the following step, the AI engine substitutes $\boldsymbol{f}$ into Equation (11) and sets the moment $\boldsymbol{\tau}$ of Equation (12) to 0 to create a linear system. Finally, the propeller coefficients are solved and used to adjust the thrusts of the propellers for generating $\boldsymbol{f}$.

If the target value is an orientation, *u(t)* represents the differences of the Euler angles. By using a similar approach, we deduce the required moment $\boldsymbol{\tau}$ as follows.

$$\begin{aligned} \vec{\omega}_1 &= u(t)/\Delta t, \\ \vec{\alpha} &= (\vec{\omega}_1 - \vec{\omega}_0)/\Delta t, \\ \vec{\tau} &= \bar{\bar{I}}\vec{\alpha}. \end{aligned} \tag{15}$$

Where $\boldsymbol{\omega_0}$, $\boldsymbol{\omega_1}$, and $\boldsymbol{\alpha}$ are the current angular velocity, new angular velocity, and angular acceleration of the ROV. Then $\boldsymbol{\tau}$ is substituted into Equation (12) and the force of Equation (11) is set to 0 to compute the propeller coefficients for producing $\boldsymbol{\tau}$.

If the target value is a velocity, then the following method is used to compute the required force:

$$\begin{aligned} \vec{a} &= u(t)/\Delta t, \\ \vec{f} &= m\vec{a}. \end{aligned} \tag{16}$$

Then, by substituting $\boldsymbol{f}$ into Equation (11) and by setting the moment of Equation (12) to 0, the AI engine computes the propeller coefficients for producing the required thrust. If the target value is an angular velocity, the required moment is computed by using a similar approach:

$$\begin{aligned} \vec{\alpha} &= u(t)/t, \\ \vec{\tau} &= \bar{\bar{I}}\vec{\alpha}. \end{aligned} \tag{17}$$

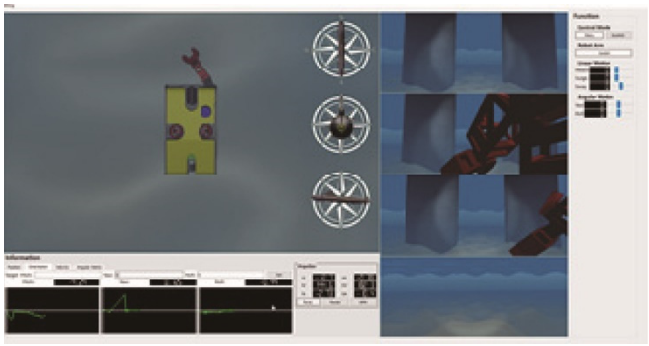


Fig. 7. The multi-window GUI of the simulator.

5. Auto-Balancing

As the robot arm moves, the gravity and buoyancy centers migrate to new positions and may not be located in the same vertical line. Under such a condition, the buoyancy force $\boldsymbol{f_B}$ creates a moment to rotate the ROV. This moment can be computed by

$$\begin{aligned} \overline{GB} &= B - G, \\ \vec{\tau}_B &= \overline{GB} \times \vec{f}_B. \end{aligned} \tag{18}$$

To counter this effect, the AI engine sets the moment term of Equation (12) to $-\boldsymbol{\tau_B}$ and the force the force term of Equation (11) to 0 to form a linear system. Then the propeller coefficients are solved to tune the propellers to generate the counter moment which keeps the ROV in balance.

VI. IMPLEMENTATION AND EXPERIMENTS

We implemented the ROV simulator by using C-language and OpenGL libraries (Shreiner et al., 2013). The embedded hardware is a desktop PC equipped with a graphics card and several input devices, including a keyboard, a mouse, and two joysticks. Multiple experiments had been performed and some of the results are presented in this section.

1. The Virtual ROV and Propeller Vectors

The structure of the virtual ROV has been described in Section II, and an image of the ROV was shown in Fig. 1. The dimension of the ROV is 1.15×0.675×0.7 m^3. We use our voxelization program to discretize the ROV and compute its mass properties. Based on our calculation, it weighs about 114.46 kg and possesses a volume of 0.133 m^3. Thus, it is positive buoyant, i.e. it will float on the sea surface if all the propellers are shut down and must rely on the propellers' thrusts to dive into the sea.

The positions and propeller vectors of the propellers are depicted in Table 1. The propeller coefficients for performing surge, heave, sway, roll, and yaw are pre-computed by using Equations (11) and (12) and presented in Table 2. As the user operates a joystick, the motion of the joystick is mapped into

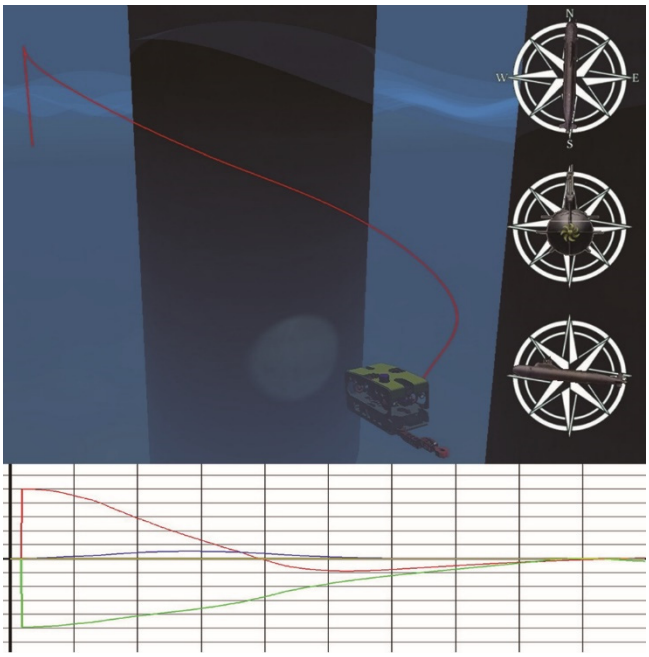


Fig. 8. Top: trajectory of the ROV, bottom: errors of the auto-piloting test.

correspondent propeller coefficients to drive the propellers. Thus, we don't have to compute them during the simulations. Since the propellers cannot produce a moment in the direction of the Z_s axis, the ROV cannot perform pitch directly. It has to move the robot arm forward to incline its body.

2. The GUI

Parts of the GUI is shown in Fig. 7, including the main window, four sub-windows, and some widgets. The main window shows the scene viewed by a bystander or the front camera. The four sub-windows display the images captured by the three side cameras and the robot arm camera. The widgets below the main canvas depict the states of the ROV, including its attitude, orientation, position, and velocities.

The users can drive the ROV and manipulate the cameras, light sources, and robot arm by using the joysticks, keyboard, and mouse. The HCI also supplies menus for the users to input control commands and to specify target positions, orientations, and velocities in auto-piloting and auto-balancing simulations.

3. Auto-piloting Simulations

Two tests had been performed to verify the auto-piloting functionality. In the first test, the user commands the ROV to reach a designated position while the ROV is moving toward another destination. At each of the following steps, the AI engine first computes the control signal and propeller coefficients. Then, the physics engine estimates the net force and moment and generates the resultant motions. Subsequently, the position and orientation of the ROV are modified. At the end of the step, the scene and key parameters are displayed on the screen by the graphics engine. The above process is automatically repeated until the ROV reaches the target place.

Table 3. PID coefficients for positioning.

	K_p	K_i	K_d
x	2.00	1.00	0.00
y	6.00	3.00	1.00
z	2.00	1.00	0.00

Table 4. PID coefficients for linear-velocity-keeping.

	K_p	K_i	K_d
surge	2.00	1.00	0.00
heave	5.00	4.00	0.00
sway	2.00	1.00	0.00

Table 5. PID coefficients for angular-velocity-keeping.

	K_p	K_i	K_d
roll	1.00	0.00	0.00
yaw	2.00	2.00	0.00
pitch	2.00	0.01	0.00

Table 6. PID coefficients for auto-orientation.

	K_p	K_i	K_d
roll angle	0.050	0.001	0.100
yaw angle	0.050	0.000	0.000
pitch angle	0.050	0.001	0.100

The trajectory of the ROV is shown in the top image of Fig. 8. The trajectory is shaded in red color. The bottom image of Fig. 8 reveals the errors at each step. The red, green, and blue curves represent the errors of the x-, y-, and z-coordinates respectively. As the process converges, the steady state error is less than 10 cm.

In the second test, the ROV is ordered to change its orientation. The AI engine coordinates the propellers to produce the required moments. The physics engine computes the resultant rotations to adjust the Euler angles of the ROV. The graphics engine reveals the simulation in real time. The initial and final orientations of the ROV are shown in the left and right images in Fig. 9. The errors of the roll (yellow curve), yaw (purple curve), and pitch (green curve) angles are drawn in the low part of this figure. When the process converges, the steady state error is less than 1 degree.

4. The Gains of the PID Controller

In the aforementioned tests, K_p, K_i, and K_d gains of the PID controller are acquired by try-and-error procedures. The PID gains for auto-positioning, linear-velocity-keeping, angular-velocity-keeping, and auto-orientation are presented in Tables 3, 4, 5, and 6.

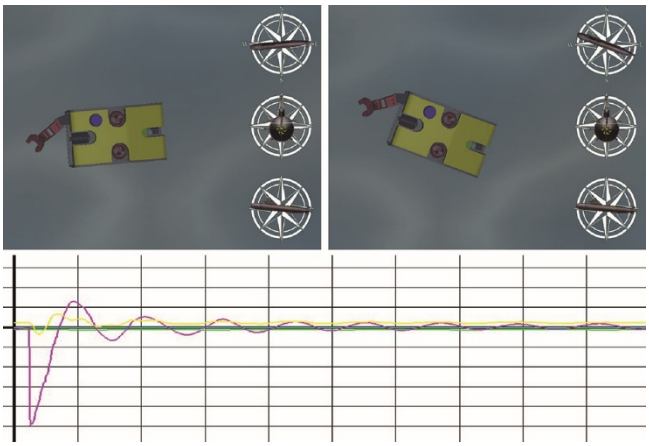


Fig. 9. Auto-orientation test, top: initial and final orientations of the ROV, bottom: errors of roll (yellow), yaw (purple), and pitch (green) angles .

As shown in Tables 3, 4, and 5, the proportional error has a significant role in velocity keeping and auto-positioning. The derivative error is useful only for maintaining the y (vertical) coordinate. In the auto-orientation experiments, we found that the PID controller is very sensitive to the variations of gains and these correspondent gains are relatively small, compared with those for velocity-keeping and auto-positioning as shown in Table 6. The derivative error becomes the most influential factor for reaching target roll and pitch angles. The integral error is decisive for achieving specified yaw angles.

5. Manual Driving Simulation

In another two tests, the user drives the ROV to form specialized trajectories by using the joysticks. These tests verify the integration of the HCI, AI engine, physics engine, and the graphics engine. Two snapshots of the tests are shown in Figure 10.

In the left image, the ROV completes a turning circle. To do so, the user uses the joysticks to control the ROV to move in constant angular and linear velocities. The trajectory of the ROV resembles a circle in a plane. The right image shows another test result. The user drives the ROV to perform heave, surge, and yaw at the same time. The resultant path forms a helix curve in the 3D space.

6. Auto-balancing Simulations

We carried out another test to verify the auto-balancing capability of the AI engine. At first, we turn-off the auto-balancing function and manipulate the robot arm to grasp samples. As the robot arm moves, the gravity and buoyancy centers of the ROV change. The buoyancy force produces a torque and causes the ROV to rotate until these two centers are aligned in a vertical line. One snapshot of the progression is shown in the top image of Fig. 11. The gravity and buoyancy centers are rendered in yellow and cyan colors respectively. As the

Fig. 10. Turning circle and helix trajectory of the ROV driven by the user.

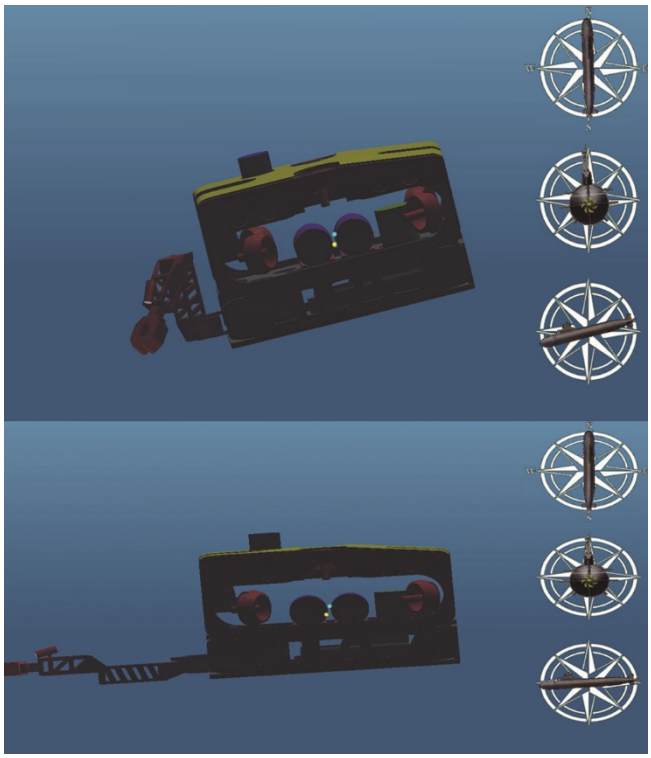


Fig. 11. Snapshots of auto-balancing test, top image: auto-balancing is turned off, bottom image: auto-balancing is turned on.

image shows, the weight of the robot arm makes the ROV pitch forward.

Then, we turn-on the auto-balancing function. The AI engine coordinates the propellers to produce desired torques to maintain the stability of the ROV while the robot arm moves. One snapshot is shown in the bottom image of Fig. 11. In this image, the gravity and buoyancy centers do not reside in the same vertical line and the ROV should pitch forward. However, the torques produced by the propellers keep the ROV in balance.

7. Discussion

The auto-piloting and auto-balancing experiments are inspired by the work of (Miskovic et al., 2006), (Lee et al., 2009), (Tehrani et al., 2010), (Hsu et al., 2000), and (De Souza and Maruyama, 2007). In these researches, their simulators support only fundamental graphic display functions. Hence, no

visualization of the simulation progressions was presented. They rely on numerical data to verify the results. On the other hand, the proposed simulator shows the virtual scenes as well as essential numerical data during the simulation processes. Users can gain more knowledge about the controlling mechanisms from the displayed results.

These tests also reveal the accuracies of our control methods. The results show that our control algorithms are comparative to those methods presented in the related work. If the target is a position, the final error is less than 10 cm. For orientation control, the errors of our simulator are less than 1 degree. In the auto-balancing simulation, the errors are within 3 degrees. These errors are shown in the bottom images of Fig. 8 and 9.

Our experiments do not include a thorough accuracy analysis because of lacking ground-truth data. It is hard to get these information from ROV vendors or the internet. Furthermore, the accuracy of each simulation is greatly influenced by the environment. To measure simulation errors in all potential working places is too expensive and impractical. Thus, error analysis is partially studied in this work.

In this research, we do not adopt Featherstone's methods to develop our simulator. The reasons can be explained as follows: His algorithms are dedicated for emulating robots operating on land. The buoyancy force produced by the air is negligible. Thus, the effects of buoyancy forces are not considered in his models. On the other hand, ROVs navigate under the sea. The buoyancy forces, generated by the sea water, significantly influence the stabilities and motions of these machines. Hence, buoyancy forces cannot be ignored in ROV simulations.

Secondly, the base of a land-based robot is usually heavy enough to keep the robot in balance as its arms move. No extra force is required to maintain the stability of the robot. However, a ROV loses its balance when its robot arm moves. It relies on the propeller thrusts to retain its stability. Featherstone's models do not concern propeller thrusts and are not suitable for simulating ROVs. Functionally speaking, ROVs are underwater robots. Nonetheless, their dynamics are quite different from those of the land-based robots. We must derive new physics models when developing our simulator.

VII. CONCLUSIONS

In this work, we present a ROV simulator for teaching people to handle ROVs. This simulator is capable of simulating a ROV equipped with movable parts. We developed a voxel-based method to renew mass properties of the ROV on the fly such that the simulation can be carried out in real time. We invent the concept of propeller vector to model the thrust of each propeller. Then we extend this model to deduce a numerical method for controlling the propellers to produce desired forces and moments.

In this work, we also developed an AI engine, which interfaces the HCI and the physics engine by translating high-level user commands and control signals into physical actions for the propellers. The AI engine also contains a PID controller such that auto-piloting and auto-balancing can be achieved. We developed the simulator by using game programming paradigm to ensure system integration, efficient computational speed, and decent visual effects.

The proposed ROV is similar to a flight simulator. Its usage is to emulate the behavior of a ROV and to offer a practicing platform for users. The embedded virtual ROV is composed of all parts of a modern ROV. Especially, it contains a moving robot arm. It resembles a real underwater vehicle better than those presented in the related work. Thus, the proposed simulator possesses higher fidelity in simulating ROVs.

REFERENCES

Agba, E. I. (1995). SeaMaster: an ROV-manipulator system simulator. IEEE computer graphics and applications, 15.1: 24-31.

Bourg, D. M. and B. Bywalec (2013). Physics for game developers. O'Reilly Media Inc.

Christ, R. D. and R. L. Wemli (2007). The ROV Manual: A User Guide for Remotely Operated Vehicles, Elsevier.

Christensen, L., P. Kampmann, M. Hildebrandt, J. Albiez and F. Kirchner (2009). Hardware ROV simulation facility for the evaluation of novel underwater manipulation techniques. Proceedings of IEEE OCEANS-EUROPE, 1-8.

De Souza E. C. and N. Maruyama (2007). Intelligent UUVs: some issues on ROV dynamic positioning. IEEE Transactions on Aerospace and Electronic Systems, 43(1): 214-226.

Fabekovic, Z., Z. Eskinja and Z. Vukic (2007). Micro rov simulator. Proceedings of the 49th International Symposium ELMAR focused on Mobile Multimedia, 97-101.

Featherstone, R. (1984). Robot dynamics algorithms.

Featherstone, R. (2014). Rigid body dynamics algorithms. Springer.

Gregory, J. (2017). Game engine architecture, AK Peters/CRC Press.

Hibbeler, R. C. (2017). Engineering mechanics, dynamics, 14 ed., Pearson Canada.

Hsu, L., R. R. Costa, F. Lizarralde and J. P. V. S. Da Cunha (2000). Dynamic positioning of remotely operated underwater vehicles. IEEE Robotics & Automation Magazine, 7(3): 21-31.

Huang, J., R. Yagel, V. Filippov and Y. Kurzion (1998). An accurate method for voxelizing polygon meshes. Proceedings of IEEE Symposium on Volume Visualization, 119-126.

Kim, T. W. (2014). Auto-configurable ROV simulator. Proceedings of IEEE OCEANS, 1-6.

Lee, S. K., K. H. Sohn, S. W. Byun and J. Y. Kim (2009). Modeling and controller design of manta-type unmanned underwater test vehicle. Journal of Mechanical Science and Technology, 23: 987-990.

Manley, J. E. (2008). Unmanned surface vehicles, 15 years of development. proceedings of OCEANS 2008.

Miskovic, N., Z. Vukic, M. Barisic and B. Tovornik (2006). Autotuning autopilots for micro-ROVs. Proceedings of the 14th IEEE Mediterranean Conference on Control and Automation, 1-6.

Nahon, M. (1996). A simplified dynamics model for autonomous underwater vehicles. Proceedings of IEEE Symposium on Autonomous Underwater Vehicle Technology, 373-379.

Schwab, B. (2009). AI game engine programming, Nelson Education.

Shreiner, D., G. Sellers, J. Kessenich, and B. Licea-Kane (2013). OpenGL programming guide: The Official guide to learning OpenGL, version 4.3. Addison-Wesley.

Tehrani, N. H., M. Heidari, Y. Zakeri and J. Ghaisari (2010). Development, depth control and stability analysis of an underwater remotely operated vehicle (ROV). Proceedings of IEEE ICCA 2010, 814-819.

Ueng, S. K., D. Lin, and C. H. Liu (2008). A ship motion simulation system. Virtual reality, 12(1), 65-76.

Ueng, S. K. (2013). Physical models for simulating ship stability and hydrostatic motions. Journal of Marine Science and Technology, 21(6), 674-685.

Xu, G., K. Liu, Y. Zhao, S. Li and X. Wang (2016). Research on the modeling and simulation technology of underwater vehicle. Proceedings of IEEE OCEANS-Shanghai, 1-6.