A dynamic simulator for underwater vehicle-manipulators

Olivier Kermorgant

A dynamic simulator for underwater vehicle-manipulators

Olivier Kermorgant[1]

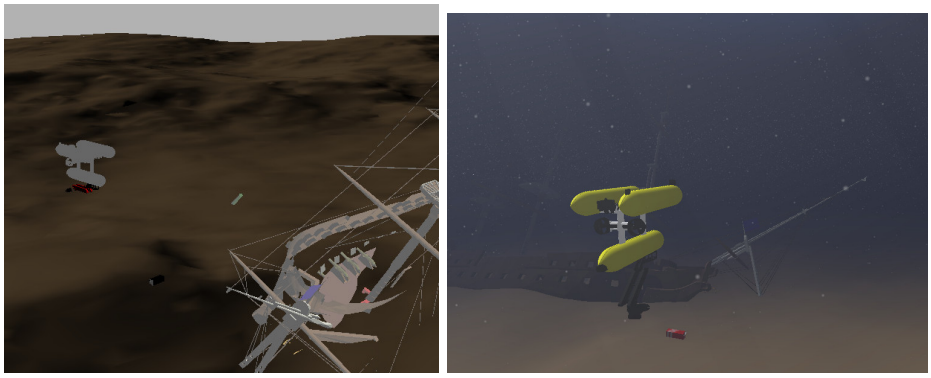
ICube Laboratory, Université de Strasbourg, France
kermorgant@unistra.fr

Abstract. In this paper we present a dynamic simulator for intervention autonomous underwater vehicles. Prototyping and testing of such robots is often tedious and costly, and realistic simulation can greatly help validating several aspects of the project. In order to benefit from existing software, the presented system is integrated with ROS, through the Gazebo dynamic simulator, and the underwater image rendering UWSim. The whole approach allows realistic rendering of dynamic multi-robot simulation, with contact physics, buoyancy, hydrodynamic damping and low-level PID control. This paper details the modeling choices that are done and exposes how to build its own AUV model. Integration with other ROS programs is exposed, and a simulation shows an example of behavior during a black box recovery mission.

1 Introduction

A strong trend in underwater robotics is the use of autonomous underwater vehicles instead of the classical Remotely Operated Vehicles and maned submersibles. Risk and cost are highly reduced, as it is never easy to deploy a team on a surface vessel or operators in a submersible. However, experimentation with AUV's is very difficult, because of the environment and the nature of the vehicle. Small experiments can be carried in water tanks, for example low-level control and basic prototyping, but this already requires space and resources. Higher-level experiments like navigation, waypoint following, seabed mapping or sensor-based control are designed to be carried in open environments, which involves high costs, human resources and are highly time consuming. Furthermore, in most autonomous underwater experiment the researchers do not have full knowledge of what is happening underwater, which is another difficulty during early development stages.

For these reasons, simulators have been developed in order to help prototyping AUV control and design. They allow having a full real-time access to all data during an experiment, and thus make it possible to greatly improve the AUV before going into the real experiments. A survey of AUV simulator has been carried in 2008 [6]. Since then, the robotic community got used to the ROS framework [2] which acquired its own AUV simulator, called UWSim [8]. This simulator has been used extensively in the Trident project, and renders realistic images



(a) Gazebo view. (b) UWSim view

Fig. 1. Rendering comparison between Gazebo (a) and UWSim (b). osgOcean allows UWSim to render automatically water color, visibility, floating particles and other underwater characteristics.

through OpenSceneGraph[1] (OSG) and osgOcean[2]. OSG is an open source 3D graphics application, while osgOcean was developped to render realistic underwater images in OSG. Fig. 1 gives an example of comparison between classical and underwater-oriented rendering. Embedded cameras, basic sonar and other AUV sensors are supported, and multiple underwater or surface vehicles can be present in the same simulation, which allows carrying experiments that would be very difficult in the real environment.

A drawback still present in the current state of the art is the absence of dynamic AUV simulation. Indeed, UWSim is only a kinematic simulator, with an external dynamic module that has to be coded in Matlab and only handles single-body vehicles. It is thus limited to kinematic control in the case of intervention AUV's (I-AUV), that carry a robotic arm. Several works have been using UWSim to show advanced whole-body control schemes [1, 4], but the simulations lack realism. On the other hand, the ROS community is used to the Gazebo simulator [5], that was designed mostly for ground robots. This simulator handles dynamics, contact physics and is very versatile through its plugin-based design. A good example of Gazebo extension is the recent Hector Quadrotor package [7] that proposes quadrotor UAV simulation. However, as seen in Fig. 1a the rendering is of course far less satisfactory for underwater environments.

The purpose of this paper is to present an integration between Gazebo and UWSim, in order to achieve both dynamic and visually realistic I-AUV simulation. In this work we focus on the low-level dynamics, as higher-level control schemes or navigation tasks can already be implemented through the numerous ROS packages. The proposed simulator, called free-floating Gazebo [3], aims to

[1] http://www.openscenegraph.org
[2] http://code.google.com/p/osgocean
[3] https://github.com/freefloating-gazebo

be flexible, letting each user specifying the characteristics of his own (I-)AUV. A typical scenario would be to use CFD software or water tank experiments to estimate the hydrodynamics coefficients of the vehicle. This parameters can then be used in the proposed simulator in order to allow real-time simulation of navigation or sensor-based missions, which leads to a good compromise between computation cost and realism compared to pure kinematics simulators.

The paper is organized as follows. In Section 2 the simulated dynamics are described together with their implementation in the robot model. Section 3 presents the low-level PID control that is included in the simulator, allowing to control the I-AUV in position, velocity or effort depending on the controller design. The integration with other programs such as high-level control is then exposed in Section 4. Finally, an experiment is detailed in Section 5, showing the recovery of a black box.

2 Description and simulation of a submerged object

The simulation is carried through two separate Gazebo plugins:

- A world plugin that simulates the overall buoyancy direction and water surface, together with the water current. The current may be varying but is assumed to be the same in the whole environment.
- A model plugin that handles low-level control through thruster effort (vehicle body) or joint torque (embedded arm).

We first describe the overall simulated dynamics before going into implementation details.

2.1 Dynamic model

Gazebo performs dynamic simulation and takes as inputs the forces and torques that acts on a body. Through plugins, it is possible to add any kind of effort and thus to consider an hydrodynamic model. In this section we describe the dynamic model of a robot link and the approximations that are carried in the implementation.

We denote $\mathbf{p}$ the pose of a robot link expressed in the world frame, and $\mathbf{v}$ the velocity screw expressed in the local frame. In this case, the general form of the state-space dynamic model of the link yields [3]:

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{G}(\mathbf{v})\mathbf{v} + \mathbf{d}(\mathbf{v}) + \boldsymbol{\tau}_g + \boldsymbol{\tau}_u + \boldsymbol{\tau}_e \tag{1}$$

$$\dot{\mathbf{p}} = \mathbf{J}\mathbf{v} \tag{2}$$

where the parameters are defined as:

- $\mathbf{M}$ is the inertia matrix. It is the sum of the rigid body inertia matrix $\mathbf{M}_d$ and the added inertia matrix $\mathbf{M}_a$ due to the fluid. Besides of the mass that can be measured directly, the values of $\mathbf{M}_d$ and $\mathbf{M}_a$ are usually determined empirically. The error on $\mathbf{M}_a$ can be quite large and we will not consider this matrix in the model.

- $\mathbf{G}(\mathbf{v})$ represents the Coriolis and centrifugal forces. It is also the combination of the corresponding dynamic $\mathbf{G}_d$ and hydrodynamic $\mathbf{G}_a$ matrices, that are computed from the inertia matrices. For the same reason as $\mathbf{M}_a$, the matrix $\mathbf{G}_a$ is considered null.
- $\mathbf{d}(\mathbf{v})$ is the hydrodynamic damping force, or parasitic drag. Its values are also usually poorly known, but this effect cannot be ignored in the underwater case.
- $\boldsymbol{\tau}_g$ is the combination of the gravity and the buoyancy forces and induced torques. In most vehicles, the center of gravity and the center of buoyancy are vertically aligned in order to produce a keel effect that stabilizes the roll of the vehicle.
- $\boldsymbol{\tau}_u$ represents the forces and torques that are added by the user. In our case, we consider additional forces through several thrusters that can be placed on the AUV body. No other kind of motors (for example fins) are considered. For the links related to the robotic arm, the applied torque and forces correspond to the joint efforts.
- $\boldsymbol{\tau}_e$ regroups the other forces and torques applied to the robot link. This corresponds to the interactions with other robot links or through contact points with other objects, which are modeled by the Gazebo simulator.
- $\mathbf{J}$ is the transformation matrix between the world frame and the link frame.

Because of the usual modeling uncertainties and the low velocity of the vehicle, the added inertia and Coriolis forces are not taken into account. All additional efforts related to the hydrodynamic effects are considered into the damping force $\mathbf{d}(\mathbf{v})$, which is a quadratic function of the velocity between the robot link and the water velocity:

$$\mathbf{d}(\mathbf{v}) = -\mathbf{k}^\top |\mathbf{v} - \mathbf{J}^{-1}\mathbf{v}_c|(\mathbf{v} - \mathbf{J}^{-1}\mathbf{v}_c) \tag{3}$$

where $\mathbf{k}$ expresses the damping coefficients and $\mathbf{v}_c = (\boldsymbol{v}_c, 0)$ is the water velocity expressed in the world frame and has null angular values.

The same modeling is used for non-actuated objects that may be present in the simulation. A free-floating buoy or an object lying on the seabed are also subject to buoyancy and hydrodynamic drag, and a robot that interacts with such objects will have to cope with the corresponding force. We now detail the implementation of the given model.

2.2 Implementation

World plugin for hydrodynamics The water velocity $\mathbf{v}_c$ and the buoyancy direction are taken into account in a world plugin, The Gazebo simulator subscribes to a ROS topic that gives the intensity and direction of the water current. It is thus possible to have the current vary if needed. The buoyancy direction is of course opposed to the gravity, and becomes null as the vehicle goes above the surface. A Gazebo world file is given and handles this plugin.

We use additional tags in the URDF [4] to handle buoyancy and damping related to a given link or object. This allows giving the position of the center of buoyancy with regards to the link frame, together with the compensation coefficient that indicates the percentage of gravity that is compensated. In most I-AUV this coefficient is slightly greater than 1, meaning that without external effort the vehicle tends to come to the surface. Here is an example of these additional tags:

```
<buoyancy>
     <compensation>1.01</compensation>
     <origin xyz= ".5 0 .2"/>
     <limit radius=".5"/>
     <damping xyz="40 80 80"/>
</buoyancy>
```

The `<limit>` tag indicates where the buoyancy force begins to decrease when approaching the water surface. In I-AUV simulation we are mostly interested in controlling the vehicle when it is entirely underwater, therefore no complex computation are carried to handle the surface approach: the buoyancy force applied to a given link simply decreases to zero when the link is near the surface.

Model plugin for thruster control Thrusters are also defined in the URDF. For each thruster the maximum effort is given, together with the mapping between the thrust direction and the body frame:

```
<link>base_link</link>
<bodyCommandTopic>body_command</bodyCommandTopic>
<jointCommandTopic>joint_command</jointCommandTopic>
<!-- for each thruster, give the map to XYZ+RPY and the maximum effort -->
<thruster>
    <map>-1 0 0 0 0 ${-body_width/2}</map>
    <effort>30</effort>
</thruster>
```

It is thus possible to simulate thruster saturation and to handle it in the control law. For now all thrusters are assumed to be fixed on the same link, which is defined as `base_link` in the given example. This tag is read by the model plugin to allow low-level control. In the given example, we see that the Gazebo simulator will listen to thruster forces on the `body_command` topic, and to joint efforts on the `joint_command` topic. These efforts can be generated from a high-level force controller, or more usually from the low-level controller that is included in the simulator. We now detail the implementation of this controller.

[4] Universal Robot Description File

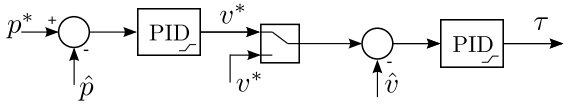


Fig. 2. Cascaded PID, here in position control. The position PID outputs a velocity setpoint that is then followed with the velocity PID.

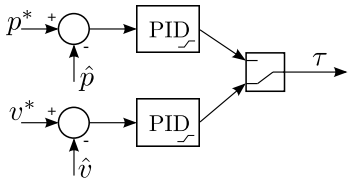


Fig. 3. Parallel PID, here in velocity control. Both the position and the velocity PID's output the desired effort.

3 Low-level PID control

In this section we describe the PID controller of the AUV simulator. Because of the numerous uncertainties on the dynamic model, it is very tedious to design a AUV advanced force controller. The usual approach is to have a high-level controller that generates position or velocity setpoints for the body and the arm, and a low-level PID controller that ensures the given setpoints are followed.

General design A independent ROS node handles the global PID controller, which is implemented as a set of parallel or cascaded controllers depending on the user choice. The two configurations are detailed in Fig. 2 (cascaded) and Fig. 3 (parallel). p^* and v^* correspond to the desired position and velocity of a joint value or a body linear or angular direction. $\hat{p}$ and $\hat{v}$ represents the estimation of the position and of the velocity, coming from the sensors or from an estimation algorithm. The velocity PID is the same in both configuration, but the position PID outputs the velocity setpoint in cascaded mode, and the effort in parallel mode. Dynamic Reconfigure[5] allows real-time gain tuning, and a configuration file is used to store the tuned gains. A ROS service is used to switch between position and velocity control, which are defined independently for the body and the arm.

Anti-windup and body thrusters mapping The maximum velocities and efforts that are given in the URDF are used for anti-windup inner loops in each PID. In particular, the maximum thruster efforts are mapped to the body directions in order to get the actual controllable directions and their associated maximum effort. No PID are instanciated for non-controllable directions (typically roll, but sometimes also pitch and sway). The position setpoint of the AUV

[5] http://wiki.ros.org/dynamic_reconfigure

is given in the world frame, while the velocity setpoint are given in the vehicle frame. The several PID outputs represent the desired wrench $\boldsymbol{\tau}_u^*$ to apply to the AUV body. This desired wrench is then mapped to the thruster efforts $\boldsymbol{\tau}_t^*$ in order to apply the saturation, and mapped back to the actual wrench $\boldsymbol{\tau}_u$ to be applied in Gazebo:

$$\boldsymbol{\tau}_u = \mathbf{T}.\boldsymbol{\tau}_t = \mathbf{T}.\text{sat}(\boldsymbol{\tau}_t^*) = \mathbf{T}.\text{sat}(\mathbf{T}^+\boldsymbol{\tau}_u^*) \tag{4}$$

where $\mathbf{T}$ is the mapping matrix between the body directions and the thrusters and $\mathbf{T}^+$ is its Moore-Penrose pseudo-inverse. Another possible strategy in case of thruster saturation is to scale all thruster efforts instead of only applying the saturation:

$$\boldsymbol{\tau}_u = \mathbf{T}.\min_{i=1}^{n_t} \frac{\tau_i^{\max}}{\tau_i^*}\boldsymbol{\tau}_t^* \tag{5}$$

where n_t is the number of thrusters and $\tau_i^{\max}$ is the maximum effort of thruster i. The real-time load of the thrusters is available for logging or high-level control, as we will see in Section 5. We now detail the integration of the simulator with Gazebo, UWSim and higher-level control nodes.

4 Integration

In this section we detail the interactions between the I-AUV simulation in Gazebo, the PID controller, the UWSim visualization and other potential higher-level controllers. First the file integration is mentioned as it is necessary to synchronize the Gazebo and UWSim description files. We then expose and detail an example of graph of nodes.

4.1 File synchronization

Gazebo and UWSim both have their own approach to describe the robot model, additional moving objects and the world terrain. As UWSim is only a kinematic simulator, the URDF used to describe the robots are usually simpler that the Gazebo ones, that contains all the inertial and collision-related data. On the opposite, the scene file, used to describe the whole world setup in UWSim, contains all information about the considered simulation while the same data is separated in several files when using Gazebo. In this section we briefly expose the synchronization script that helps building lesser-information files from higher ones.

The starting point is the UWSim launchfile that contains the UWSim scene file, that describes the whole setup. This allows retrieving the list of robots URDF files and moving objects, their starting position and the terrain 3D mesh that is used. The URDF files may already exist or need to be generated from existing xacro files, that are used in Gazebo. All starting positions are found in the UWSim scene file, which allows generating a Gazebo model spawner: this ensures that the same objects will be present at the same position in Gazebo and

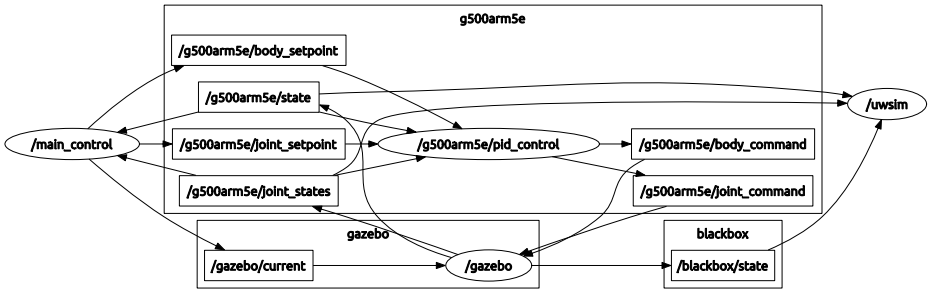


Fig. 4. Auto-generated graph of nodes (round-shaped) and topics (rectangular).

UWSim. Finally, for now UWSim and Gazebo do not use the same convention for file pathes. The pathes are thus adapted on the fly when generating or updating missing files. With such synchronization, an experiment can first be carried only in UWSim with kinematic control for simplicity and debugging, then with the dynamic simulator and low-level PID loops to have a realistic behavior. We now detail the interactions between the several ROS nodes in a typical simulation.

4.2 Node interactions

This section details the interactions between the simulation node, integrated through plugins in Gazebo, and the other nodes of a typical I-AUV simulation. The graph of nodes is represented in Fig. 4. We can see all moving objects have their own namespace. Here `/g500arm5e` is the robot namespace and regroups all nodes and topics related to its low-level control, while `/blackbox` is related to a moving black box. Moving objects are considered as non-actuated robots, thus have their own namespaces. The `gazebo` node subscribes to the `/gazebo/current` topic, which allows defining and modifying the water current.

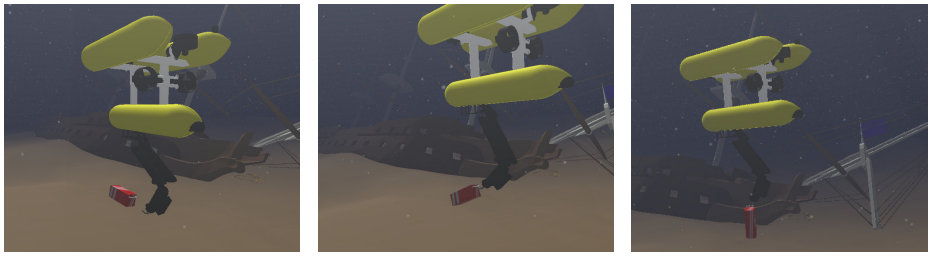
The main controller is called `/main_control`. It reads body and joint positions from the corresponding topics. In this simulation these values are the true ones and come directly from Gazebo. Water current, as well as body and joint position setpoints are generated. These setpoints are handled by the `pid_control` node, which generates the joint effort `joint_command` and the body wrench `body_command`. These topics are read by Gazebo in order to apply the corresponding forces and torques to the vehicle and its arm.

As we can see on the graph, UWSim only receives data in order to visualize the current pose of the I-AUV and of the black box. Still, an embedded camera is simulated in UWSim and the generated images could be used in the main controller to perform visual servoing as shown in [4].

We now expose the simulation results.

5 Experiments

In this section we show an example of behavior of a black box recovering mission.



(a) End of approach. The hook is lying on the seabed and the arm cannot follow its setpoint.

(b) Beginning of lifting. One of the vertical thruster is saturating.

(c) Recovery complete.

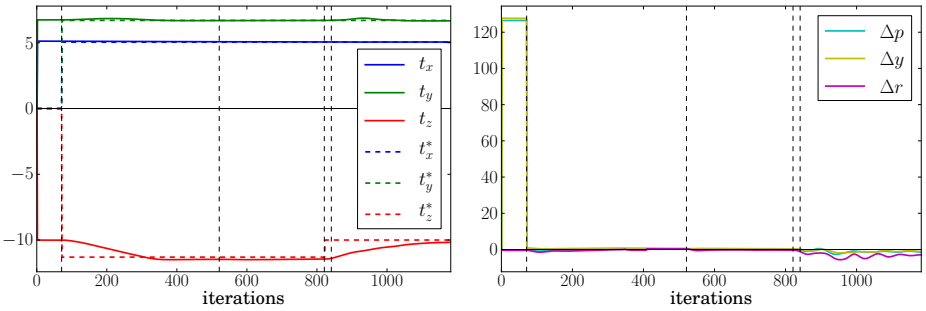
Fig. 5. UWSim view of the simulation.

Experimental setup The considered I-AUV is the Girona 500 [9], equipped with a 3-degrees of freedom robot arm mounted with a hook on the end-effector. The AUV body has 5 thrusters, allowing to control all degrees of freedom except for the roll. A keel effect is induced by the relative position of the center of gravity and the center of buoyancy such that the roll is stable. The AUV is slightly lighter than water, thus the two vertical thrusters are always used to stabilize the depth.

Simulation scenario As the goal is to show the dynamic behavior, position control is in open loop and the PID gains are not well tuned. The scenario includes several steps that illustrate difficulties of underwater interventions:

1. At first only joint control is performed to set the arm in an idle position.
2. At iteration 70, body position control is used to approach the black box. At the end of the approach, around iteration 350, the hook will touch the seabed and thus the arm will not be able anymore to follow its setpoint.
3. At iteration 520, the joint setpoint is changed to prepare the recovery
4. At iteration 820, the body position setpoint is changed to go up and recover the box, which is entirely carried by the AUV after iteration 840. Because of the weight of the box, some joints and some thrusters will be saturated at this time.

Fig. 5 depicts the I-AUV in various steps. At the end of the approach (Fig. 5a), the hook slightly hits the seabed, which will appear on the joint measurements. Fig. 5b shows the beginning of the recovery, which corresponds to some oscillations and to the saturation of a vertical thruster. In Fig. 5c the I-AUV has recovered the black box and is going up. Because of the weight of the black box, some joints are saturated.



(a) Body position (plain) and setpoint (dotted). Surge (blue), sway (green) and heave (red).

(b) Body orientation error in degree. Roll (magenta), pitch (cyan) and yaw (yellow).

Fig. 6. 3D behavior of the I-AUV body. The heave (red) follows its setpoint after iteration 70, while the orientation error (b) is null even if small oscillations appear when lifting the black box after iteration 840.

Body position control The overall behavior of the body control is shown in Fig. 6. The vertical dotted lines correspond to the key-iterations that are mentioned above. The measurement rate is 30 Hz. The low-level PID controllers ensure the body position and orientation setpoints are followed after iteration 70 (beginning of body position control). Interaction with the black box is clearly visible from iteration 840. As soon as the box is entirely recovered, small oscillations appear on the orientation error (Fig. 6b). This is due to the fact that the box behaves like a pendulum when carried by the recovering hook, and induces some perturbations on the vehicle position control.

Thruster load The described body behaviors also appear in Fig. 7, that represents the thruster load in percent. When gived a new position setpoint at iteration 70, we see one of the vertical thruster (cyan) is saturated. This explains the slow reaching of the heave setpoint (red) in Fig. 6a. From iteration 350 the hook is lying on the seabed. The load is thus reduced on the vertical thruster. It is saturated again after iteration 820, when the new position setpoint makes the I-AUV go up.

The oscillations of the orientation error in Fig. 6b are also visible in Fig. 7 after iteration 820. The thruster load oscillations are more important than the orientation error, which means the PID controllers of the thrusters manage to greatly reduce the perturbation induced by the carrying of the black box.

Joint behavior The behavior of the 3-degrees of freedom arm is represented in Fig. 8. First the setpoints are followed but when the hook comes in contact with the seabed at iteration 350, the arm is stuck between the ground and the I-AUV body and it becomes impossible to follow the position setpoint. The setpoint

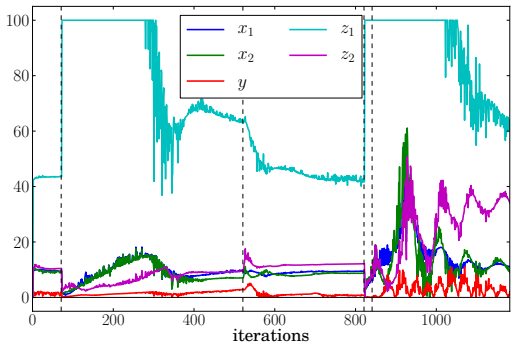


Fig. 7. Thruster load in percent. Surge/yaw (blue and green), sway (red) and heave/pitch (cyan and magenta).

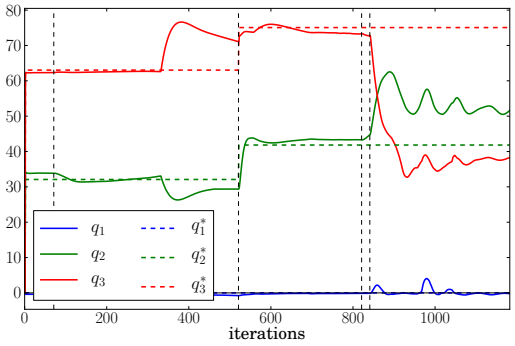


Fig. 8. Joint positions and setpoints. Slew (blue), shoulder (green) and elbow (red). The hook touching the seabed is clearly visible at iteration 350, as the arm cannot ensure its position setpoint. Oscillations appear after iteration 820 because of effort limits.

changes at iteration 520, making it possible to be followed again. Once the arm begins to lift the black box at iteration 820, the joint effort limits clearly appear as the shoulder and elbow cannot cope with the weight of the black box.

Discussion The whole experiment demonstrates the dynamic simulation, together with contact physics between the hook and both the seabed and the black box. Dynamic coupling is clearly visible when lifting the box, as its pendulum-like oscillation are passed on the whole I-AUV. The PID controllers of the thrusters manage stabilizing the body position, but the arm still oscillates and cannot follow its position setpoint due to the limits on joint efforts. The overall simulation shows that dynamic simulation of a I-AUV raises realistic difficulties that are possible to try and solve in the simulated environment, before having to perform real experiments.

6 Conclusion

We have presented a dynamic simulator for intervention autonomous underwater vehicles. Through the ROS framework, it was possible to use both the dynamic simulation capabilities of Gazebo and the realistic underwater rendering of UWSim. The considered modeling is focused on the main effects due to hydrodynamic forces, that are drag and buoyancy. A possible improvement is to take into account the added inertia and Coriolis, but these effects are difficult to quantify precisely. That is why all uncertainties are here considered in the damping coefficients $\mathbf{d}(\mathbf{v})$. Additional URDF tags allows defining the buoyancy and damping parameters of each robot link or other moving object, such as the black box that is considered in the example. The presented experiment was designed to show that dynamic simulation can be used to reproduce realistic difficulties such as thruster or joint effort limit, collision and uncertainties on the setpoint following. As in real I-AUV, this can be due to non-optimal tuning of the low-level controllers, or on the uncertainties on the hydrodynamic parameters. The proposed simulation framework can thus be used to validate new algorithms for sensor-based control or state estimation. In future works we will focus on underwater sensors in order to perform realistic sensor-based missions.

References

1. Casalino, G., Zereik, E., Simetti, E., Torelli, S., Sperinde, A., Turetta, A.: Agility for underwater floating manipulation: Task & subsystem priority based control strategy. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems. pp. 1772–1779. Vilamoura, Portugal (September 2012)
2. Cousins, S.: Welcome to ros topics. IEEE Robotics & Automation Magazine 17(1), 13–14 (2010)
3. Fossen, T.I.: Guidance and control of ocean vehicles, vol. 199. Wiley New York (1994)
4. Kermorgant, O., Pétillot, Y., Dunnigan, M.: A global control scheme for free-floating vehicle-manipulators. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems. Tokyo, Japan (November 2013)
5. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IROS. vol. 3, pp. 2149–2154. IEEE (2004)
6. Matsebe, O., Kumile, C.: A review of virtual simulators for autonomous underwater vehicles (auvs). Killaloe, Ireland (2008)
7. Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., von Stryk, O.: Comprehensive simulation of quadrotor uavs using ros and gazebo. In: Simulation, Modeling, and Programming for Autonomous Robots, pp. 400–411. Springer (2012)
8. Prats, M., Pérez, J., Fernández, J.J., Sanz, P.J.: An open source tool for simulation and supervision of underwater intervention missions. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems. pp. 2577–2582. Vilamoura, Portugal (October 2012)
9. Ribas, D., Ridao, P., Magí, L., Palomeras, N., Carreras, M.: The girona 500, a multipurpose autonomous underwater vehicle. In: IEEE OCEANS. pp. 1–5. Santander, Spain (June 2011)