Efficient Continuous System Integration and Validation for Deep-Sea Robotics Applications

Tobias Fromm, Christian A. Mueller,
Max Pfingsthorn, and Andreas Birk
Robotics Group,
Computer Science & Electrical Engineering,
Jacobs University Bremen, Germany

Paolo Di Lillo

Interuniversity Center of Integrated Systems
for the Marine Environment (ISME),
University of Cassino and Southern Lazio, Italy

***Abstract*— Deep-sea operations of remotely-operated vehicles (ROV) need robust testing and deployment strategies beyond the traditional pre-deployment validation on real hardware. Seamless integration of simulated components into the validation pipeline allows for rapid development of components and validation under controlled conditions. We describe the benefits arising from such a continuous integration and validation approach as well as an example setup in the EU project *DexROV*.**

I. Introduction

Marine robotics have been exploring the sea in increasing depth, being beneficial for many other fields of science and engineering. However, deep-sea operations demand for specialized testing and deployment strategies because the effort and costs for failure are magnitudes higher than in ground robotics, for example, when a remotely-operated vehicle (ROV) malfunctions in deep sea and cannot be retrieved anymore.

Hence, the traditional approach of running pre-deployment validation entirely on the real hardware must be questioned in this field because similar conditions cannot be generated artificially. In the following, we will explain a versatile integration and validation architecture which allows for pre-deployment testing using simulated and real system components besides each other in a seamless way. We propose pre-deployment simulation as the key factor for testing single components as well as the entire system in an efficient way.

As a deployment example we provide the EU project *DexROV* (Effective Dexterous ROV Operations in Presence of Communication Latencies, [1]).

II. System Architecture

DexROV features a full-fledged ROV system, deployed from a vessel in the Mediterranean and, amongst others, equipped with a stereo perception system and a manipulator. Within the project, several real-life field trials have been scheduled and partly performed already.

However, the different workgroups do not have access to the hardware prior to the annual trials despite the need to test their developments thoroughly. For this reason, we assigned high priority to establish a simulation base before implementing any business logic in order to allow developers to work independently.

The research leading to the presented results has received funding from the European Union's Horizon 2020 program within the project "Effective Dexterous ROV Operations in Presence of Communication Latencies" (DexROV).

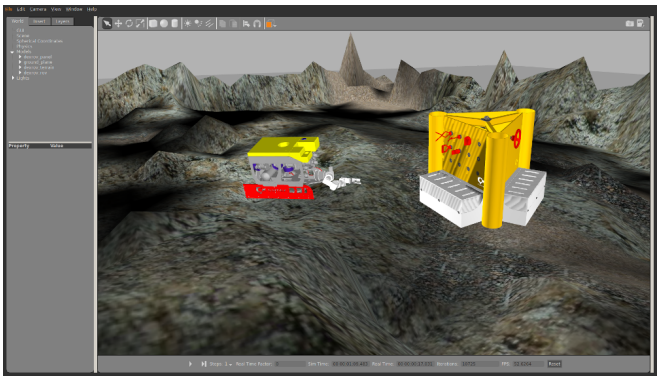


Fig. 1. Example scene containing ROV, testing panel and terrain

A. Simulation Framework

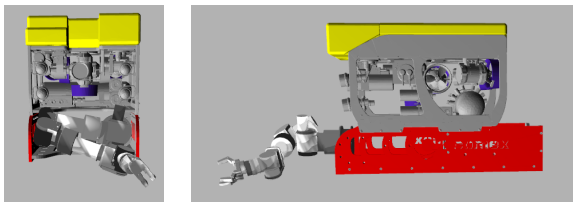
Since many different components need to be validated in simulation in a real robotics scenario, we need to rely on an established simulation framework. As such, we use Gazebo [2] which already offers most of the necessary capabilities, including basic marine systems simulation support in the shape of a buoyancy plugin. Additionally, simulated sensor data can easily be obtained by integrating various simulated sensors into the robot models, e.g. RGB and depth cameras.

Figure 1 shows the simulation environment with an example scene. This contains the functional ROV model along with a model of a testing panel which features different valves and levers, specifically generated to test the manipulation capabilities of the ROV, and an example terrain (see Section II-C).

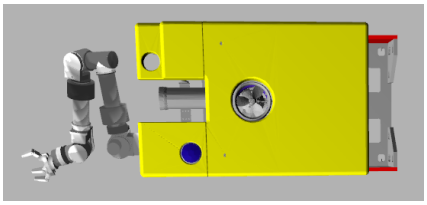
B. ROV Simulation

In order to simulate kinematics and dynamics of the ROV, a suitable model was integrated into the simulation environment, containing:

- CAD models of ROV, payload skid, arm and hand, see Fig. 2
- physical properties for all models
- several controllers for low- and high-level navigation.



(a) front view (b) side view



(c) top view

Fig. 2. ROV model

The DexROV system needs two controllers for effectively operating the arm and the vehicle. Each controller is designed as two nested control loops, in which the high-level controller (HLC) is kinematic and takes as input the desired system pose and outputs a reference system velocity, and the low-level controller (LLC) computes the reference torque for the motors. The architecture embedding the different controllers is shown in Fig. 3.

The integration of these controllers within the simulation environment has been designed by implementing two Gazebo plugins for the low-level controllers and two Robot Operating System (ROS) [3] nodes for the high-level controllers.

For each low-level controller, two different Gazebo plugins have been developed, one purely kinematic and one dynamic. Both of them exhibit the same interfaces in terms of ROS topics, so they are completely interchangeable. The kinematic ones just apply instantaneously the reference velocity to the models in the simulation, without any kind of dynamics. This is one of the main advantages of the usage of a simulator, because it is very useful for validating the correctness of the high level controllers. The dynamic low-level controllers are standard PID controllers that take as input the reference velocity and computes the desired forces and torques to be applied to the models in the simulation.

The developed plugins take as parameters the PID gains that have to be properly tuned with respect to the realistic dynamic parameters of the vehicle and the arm models. This kind of controller, jointly with the accurate physics simulation, represents a good starting point for the actual controllers design and tuning.

The high-level controllers take their inputs (vehicle current and desired pose, joint position and end-effector desired pose) and publish their outputs (vehicle and joint velocities) following the algorithms described in [4] and [5].

The motion control of the arm has been developed resorting to an inverse kinematics algorithm that allows to perform multiple tasks simultaneously, defining a priority among them. Given a task hierarchy composed by n tasks

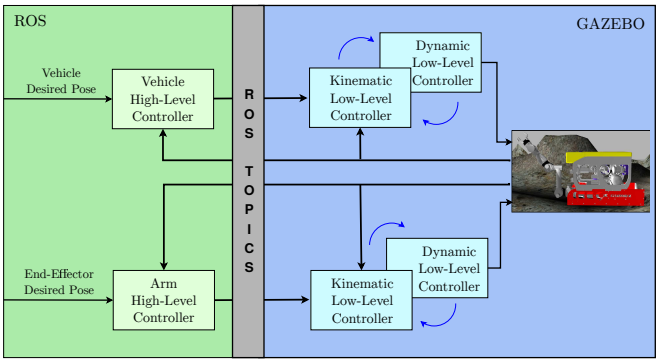


Fig. 3. Controller architecture

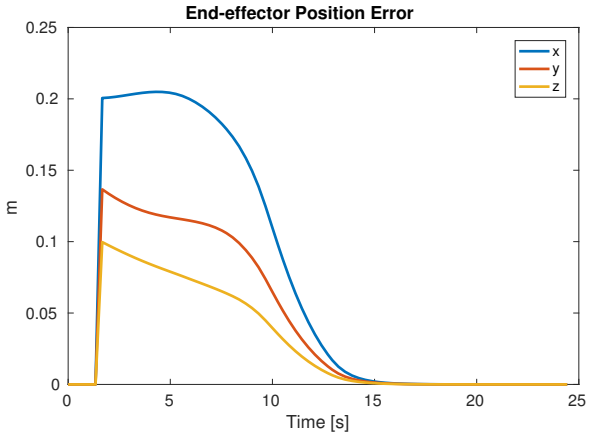


Fig. 4. End-effector position error on x (blue), y (red) and z (yellow) axis over time, given a constant waypoint

$\boldsymbol{\sigma_i}(\mathbf{q})$ with $i = 1 \dots n$, the reference system velocity $\dot{\mathbf{q}}$ that fulfills all the tasks simultaneously can be computed as:

$$\dot{\mathbf{q}} = \mathbf{J_1^\dagger K_1 \tilde{\sigma}_1} + \mathbf{N_1 J_2^\dagger K_2 \tilde{\sigma}_2} + \cdots + \mathbf{N_{1,n-1} J_n^\dagger K_n \tilde{\sigma}_n}$$

where $\mathbf{J_i^\dagger}$ is the Moore-Penrose pseudoinverse of the Jacobian matrix of the i-th task, $\mathbf{K_i}$ is a definite-positive matrix of gains, $\tilde{\boldsymbol{\sigma}}_i$ is the i-th task error defined as $\tilde{\boldsymbol{\sigma}}_i = \boldsymbol{\sigma}_{i,d} - \boldsymbol{\sigma}_i$ and $\mathbf{N_{1,i}}$ is the null space of the augmented Jacobian:

$$\mathbf{J_{1,i}} = \begin{bmatrix} \mathbf{J_1} \\ \mathbf{J_2} \\ \vdots \\ \mathbf{J_i} \end{bmatrix}$$

The projection of the solutions of the lower priority tasks into the null space of the higher priority ones deletes the velocity components that would conflict with them. As a representative example, in Fig. 4 and Fig. 5 the results of a simple task hierarchy composed by the end-effector position and orientation tasks are shown, in which a target set point for the arm configuration has been given. The same approach will be used to add more tasks, such as obstacle avoidance and mechanical joint limits, in order to achieve the needed operations for the DexROV system in a safe and effective manner.

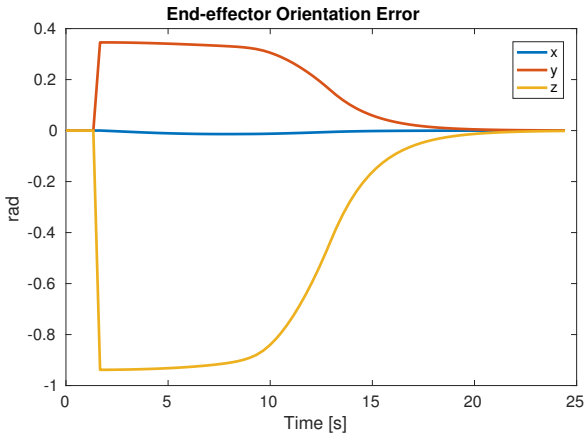


Fig. 5. End-effector orientation error on x (blue), y (red) and z (yellow) axis over time, given a costant waypoint

The vehicle high-level controller exibits an interface for receiving a list of desired waypoints via ROS topic. Each waypoint is reached by resorting to two PI controllers for the position and the orientation, that in this case is only a heading controller, since the ROV is not fully actuated on the rotational axis. Regarding the position control, the control input $\mathbf{u}$ is computed as:

$$\mathbf{u} = \mathbf{K_p e} + \mathbf{K_i} \int_{\mathbf{t_i}}^{\mathbf{t}} \mathbf{e}(\tau)\mathbf{d}\tau$$

where $\mathbf{K_p}$ is the proportional gain, $\mathbf{K_i}$ is the integral gain and $\mathbf{e} = \mathbf{p_d} - \mathbf{p}$ is the position error. Similarly, the yaw controller is designed as:

$$r = K_{p,\psi}\tilde{\psi} + K_{i,\psi} \int_{t_i}^{t} \tilde{\psi}(\tau)d\tau$$

where $K_{p,\psi}$ is the proportional gain, $K_{i,\psi}$ is the integral gain and $\tilde{\psi} = \psi_d - \psi$ is the heading error.

Fig. 6 and Fig.7 show the position and heading error relative to a list of three desired waypoints.

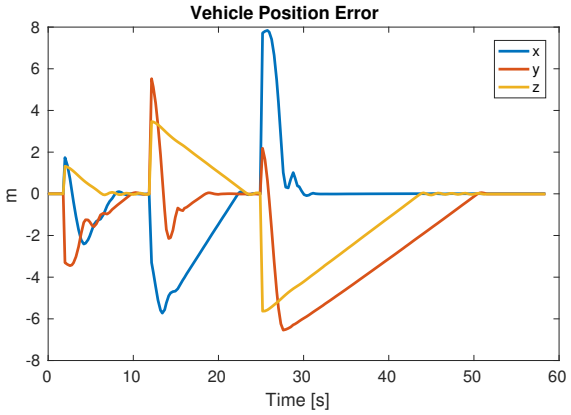


Fig. 6. Vehicle position error on x (blue), y (red) and z (yellow) axis over time, given a sequence of three waypoints

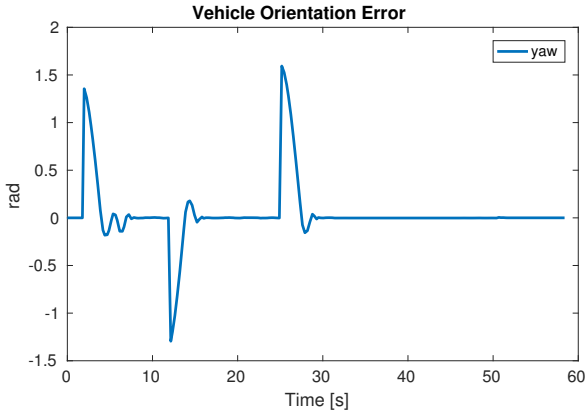


Fig. 7. Vehicle heading error over time, given a sequence of three waypoints

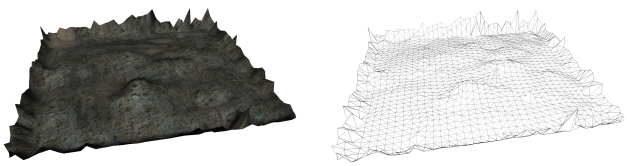
A ROS node for the teleoperation of the system has been also developed, useful for debugging all the modules that compose the system. This node allows to teleoperate both the vehicle and the end-effector of the arm, by reading the commands from a standard usb joystick and writing them on the topics of the high-level controllers.

C. Scene Modeling

Like in real underwater settings, we model the simulated scene in a user-friendly way to allow for the operator to quickly perceive the current vehicle view, occupied areas and recognized objects. In order to obtain colored point clouds, we use a simulated depth camera beneath a simulated RGB camera to produce the results of Fig. 8(c) & (e).

In the real underwater case, the simulated cameras are replaced by a custom-made stereo camera system which delivers monocular images as well as 3D point clouds. An occupancy map [6] representation is used for environment modeling (Fig. 8(d)). The scene model is completed with recognized objects like the panel in Fig. 8(f), using different texture and shape-based recognition methods.

Populating the simulated world with an underwater terrain allows for a more realistic view and handling of the ROV in the target scenario. We produced a terrain mockup from a point cloud recorded in underwater trials using a recently developed surface reconstruction method [7]. The resulting surface is shown in Figure 9. This model is visibly satisfactory, but coarse enough to create no visible processing load which would slow down the simulation.



(a) with texture (b) without texture

Fig. 9. Simulated terrain generated using [7]

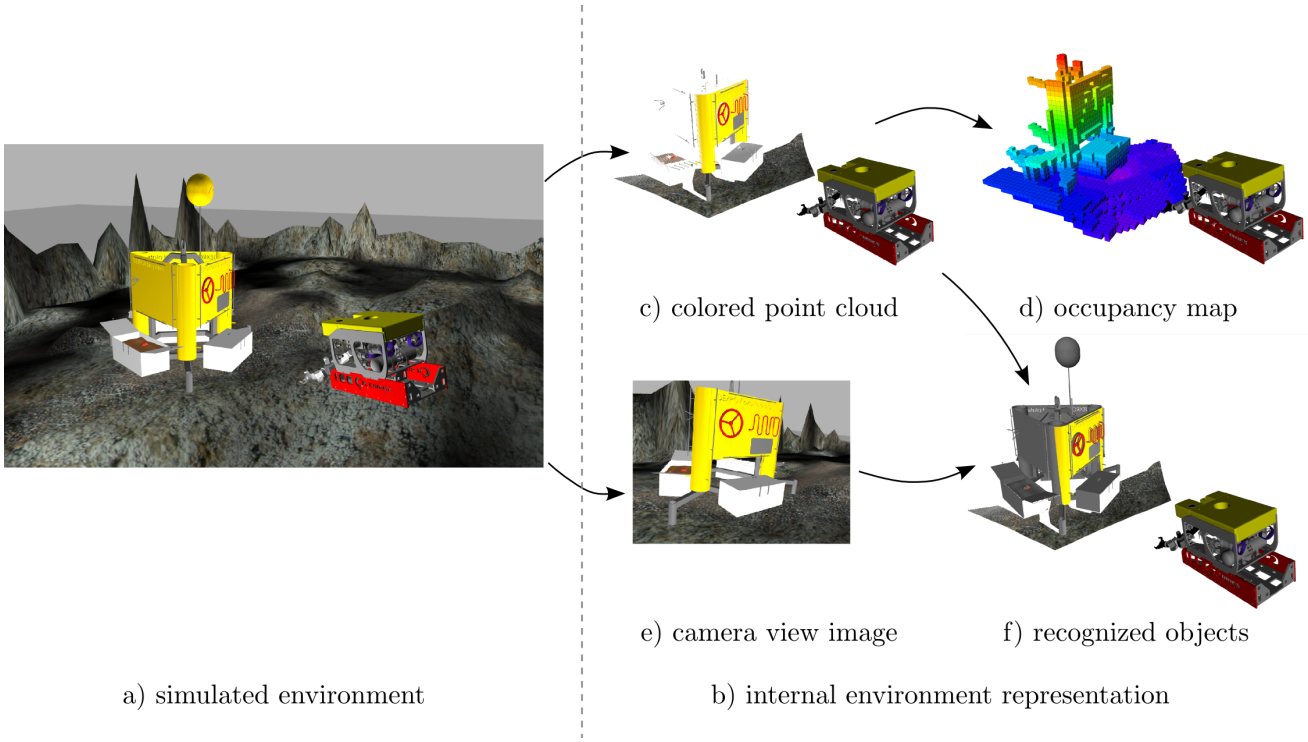


Fig. 8. Scene Modeling Overview – find a video showing our ROV modeling a simulated scene on `https://youtu.be/pcYAgYt65Bc`

D. Packaging, Virtualization and Networking

As a tool for packaging, virtualization and easy deployment, we use Docker [8]. Since usually a robotic system consists of at least two machines (the onboard computer and some operator station), Docker facilitates to emulate the separation of process contexts and data transfer which occurs in a real distributed system. The important advantage of packaging each individual system into a Docker container is that the user can replace real machines by emulated ones on demand.

In the case of DexROV, four containers have been defined:

- **Simulation:** ROV physics and sensor/actuator simulation which can be replaced by the real ROV
- **Onboard:** ROV onboard low-level processing (e.g. stereo camera data fusion, actuator control)
- **Vessel:** autonomous capabilities running on the support vessel (e.g. motion planning, scene modeling)
- **Onshore:** high-level user interface in a command center which receives data over satellite link.

In addition to logical separation, inter-machine networking plays an important role which, especially in case of open-sea marine robot deployment, usually features delayed and degraded network connections. We take this into account prior to deployment using a network simulator [9] based on *netem* [10] which allows for dynamic changes of the following parameters:

- available bandwidth
- delay (with variance)
- package loss percentage
- package corruptness percentage.

These can be set according to the limitations in the application scenario. In DexROV, for the *onboard-vessel* connection, we use settings of 20 MBit/s and 100 ms mean delay with 10 ms variance. For the satellite connection between *vessel* and *onshore*, a bandwidth of 1 Mbit/s, a 150 ms mean delay with 75 ms variance, a package loss of 5% and a package corruption of 0.1% are applied which reflect the properties of the physical connections between the respective stations.

III. Efficient Continuous System Integration

Not only in deep-sea offshore operations, but especially there due to harsh conditions and limited data transmission capabilities, efficient integration and testing is crucial during the development of complex systems.

We identify the following advantages of our modularized, simulation-based approach against conventional monolithic testing on the complete system:

A. Distributed Deployment

Different people and workgroups can work collaboratively on the whole system, individually replacing specific simulated components with real ones as per their needs.

This is relevant in integration settings where each group is to deploy their own components in the processing pipeline. Preferrably, simulated components are used at first to eliminate noise and other issues which typically accrue in the respective real components. The latter can then gradually replace the former.

B. Interface/Pipeline Testing

Before even involving real hardware, interfaces and processing pipelines can already be tested thoroughly using simulated data.

Especially when hardware interfaces still have to be designed, implemented and tested, developers can use simulation to optimize their processing pipelines and workgroup-internal as well as external interfaces.

C. Regression/Degradation Testing

Specific constraints (bandwidth, delays, processing power, environmental conditions, sensor noise) can be taken care of individually, without influencing each other.

Since for certain components and conditions only a selection of the mentioned constraints may be occuring, particular constraint profiles can be defined in order to test each component.

D. Parallelized Testing

Developers can test their components in parallel and, in case repetitive testing is necessary, even on parallel instances of the whole system, on cheap desktops instead of onboard hardware.

Especially when only one instance of hardware is available, simulated components may replace this. Using packaging and virtualization tools makes it easy to deploy the whole system multiple times in parallel, also increasing the overall computing power and thus reducing testing times.

E. Fault Recovery/Safety Testing

Instead of taking the risk to lose a robot in deep sea, many cases and margins can be validated already in the simulator before deployment under real conditions.

Expensive hardware generally poses limitations onto borderline testing which may damage the hardware. Since simulated components can be restored with no costs and efforts, hardware may come into play only in final overall-system verification steps.

IV. Deployment Scenario

Within the DexROV project, several field trials have already taken place where we successfully applied the explained concept in various hybrid configurations. For example, we deployed an ROV in the old harbor of Marseille where limited view, spatial constraints as well as software components were still under development, did not allow for full ROV operation. For instance, the scene modeling part was replaced by the respective simulated components as shown in Fig. 8.

As a result, the field trial was successfully conducted considering that components could consume simulated scene modeling data in order to function properly in the DexROV framework. Furthermore, we were able to test parts of the hardware, concretely the thrusters and sonars, while the confined space in the harbor did not allow for safe displacement and the dirty water rendered stereo perception impossible. Nevertheless, all components could be tested fully integrated in the processing pipeline with the constrained ones replaced by their simulated counterparts.

Another important aspect of this trial was the evaluation of the communication bandwidth between vessel and onshore control center. The user interface at the control center consumes live scene data from the ROV. Therefore ensuring a reliable communication via the satellite link is crucial, since a reliable supply of real-time scene data has to be guaranteed for safe operation. Using the simulated scene modeling components we could generate artificial scene data with different dimensioning and parameters like occupancy map resolution.

As a result, we were capable to evaluate the required communication setup including bandwidth constraints for the produced scene data under real satellite communication conditions.

V. Related Work and Further Uses

Within the DexROV project, the pipelines and procedures described in this paper are used for multiple purposes.

Amongst others, we use the system together with a stereo camera to perform object recognition like shown in Fig. 8(f). This can be achieved only using an accurately calibrated camera system, hence we employ a calibration procedure [11] in air, prior to starting real-component testing, and exploit its results in the integrated underwater setting.

In order to subsequently manipulate the valves and levers on the testing panel, candidates for them can be discovered first [12] and then the respective components are identified [13], [14] using for instance methods of our previous work. Manipulation then eventually takes place using the ROV controllers as described in Section II-B.

As a high-level addition to the whole perception and manipulation pipeline, manipulation strategy planning can be performed for pickable objects found in submarine settings, like archaeological artifacts or biological samples [15].

VI. Conclusion

In the EU project *DexROV*, we established a continuous integration and validation approach which can be transferred to projects which put similar emphasis on hardware-agnostic testing and distributed development. At the same time, the ratio of simulated components can be adjusted as per available resources and criticality of the mission.

Not only for deep-sea, but also other robotic operations which involve larger groups working with limited hardware under rough conditions, we propose to generally adopt a versatile simulation-based validation scheme based on the conditions described in this paper.

References

[1] J. Gancet, P. Weiss, G. Antonelli, M. Pfingsthorn, S. Calinon, A. Turetta, C. Walen, D. Urbina, S. Govindaraj, P. Letier, X. Martinez, J. Salini, B. Chemisky, G. Indiveri, G. Casalino, P. di Lillo, E. Simetti, D. de Palma, A. Birk, T. Fromm, C. Mueller, A. Tanwani, I. Havoutis, A. Caffaz, and L. Guilpain, "Dexterous Undersea Interventions with Far Distance Onshore Supervision: the DexROV Project," in *IFAC Conference on Control Applications in Marine Systems*, 2016.

[2] N. Koenig and A. Howard, “Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator,” in *International Conference on Intelligent Robots and Systems*, 2004.
[3] A. Koubaa, *Robot Operating System (ROS): The Complete Reference (Volume 1)*, 1st ed. Springer, 2016.
[4] D. De Palma and G. Indiveri, “Underwater vehicle guidance control design within the DexROV project: preliminary results,” in *IFAC Symposium on Intelligent Autonomous Vehicles*, June 2016.
[5] S. Moe, G. Antonelli, A. Teel, K. Pettersen, and J. Schrimpf, “Set-based Tasks within the Singularity-robust Multiple Task-priority Inverse Kinematics Framework: General Formulation, Stability Analysis and Experimental Results,” *Frontiers in Robotics and AI*, vol. 3, p. 16, 2016.
[6] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: An efficient probabilistic 3D mapping framework based on octrees,” *Autonomous Robots*, 2013.
[7] T. Fromm, C. Mueller, and A. Birk, “Unsupervised Watertight Mesh Generation From Noisy Free-Form RGBD Object Models Using Growing Neural Gas,” Tech. Rep., 2016, https://arxiv.org/abs/1603.00663.
[8] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” *Linux Journal*, vol. 2014, no. 239, Mar. 2014.
[9] M. Pfingsthorn, *Docker Networking Simulation*, Jacobs University, 2016, https://github.com/maxpfingsthorn/mini-network-simulator.
[10] “Netem,” The Linux Foundation. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem
[11] T. Luczynski, M. Pfingsthorn, and A. Birk, “The Pinax-model for accurate and efficient refraction correction of underwater cameras in flat-pane housings,” *Ocean Engineering*, vol. 133, pp. 9–22, 2017.
[12] C. A. Mueller and A. Birk, “Hierarchical Graph-Based Discovery of Non-Primitive-Shaped Objects in Unstructured Environments,” in *International Conference on Robotics and Automation*, 2016.
[13] C. A. Mueller, K. Pathak, and A. Birk, “Object Shape Categorization in RGBD Images using Hierarchical Graph Constellation Models based on Unsupervisedly Learned Shape Parts described by a Set of Shape Specificity Levels,” in *International Conference on Intelligent Robots and Systems*, 2014.
[14] H. Bülow and A. Birk, “Spectral 6-DOF Registration of Noisy 3D Range Data with Partial Overlap,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 4, pp. 954–969, 2013.
[15] T. Fromm and A. Birk, “Physics-Based Damage-Aware Manipulation Strategy Planning Using Scene Dynamics Anticipation,” in *International Conference on Intelligent Robots and Systems*, 2016.