PROCEEDINGS

55. IWK

Internationales Wissenschaftliches Kolloquium
International Scientific Colloquium

13 - 17 September 2010

Crossing Borders within the ABC

Automation,

Biomedical Engineering and

Computer Science

Faculty of
Computer Science and Automation

www.tu-ilmenau.de

TECHNISCHE UNIVERSITÄT
ILMENAU

Impressum
Published by

Publisher:	Rector of the Ilmenau University of Technology Univ.-Prof. Dr. rer. nat. habil. Dr. h. c. Prof. h. c. Peter Scharff
Editor:	Marketing Department (Phone: +49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de) Faculty of Computer Science and Automation (Phone: +49 3677 69-2860) Univ.-Prof. Dr.-Ing. habil. Jens Haueisen
Editorial Deadline:	20. August 2010
Implementation:	Ilmenau University of Technology Felix Böckelmann Philipp Schmidt

USB-Flash-Version.

Publishing House:	Verlag ISLE, Betriebsstätte des ISLE e.V. Werner-von-Siemens-Str. 16 98693 Ilmenau
Production:	CDA Datenträger Albrechts GmbH, 98529 Suhl/Albrechts
Order trough:	Marketing Department (+49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de)

ISBN: 978-3-938843-53-6 (USB-Flash Version)

Online-Version:

Publisher:	Universitätsbibliothek Ilmenau ilmedia Postfach 10 05 65 98684 Ilmenau

Home / Index:
http://www.db-thueringen.de/servlets/DocumentServlet?id=16739

A GENERIC GUIDANCE SYSTEM FOR UNDERWATER VEHICLES

Helge Renkewitz, Torsten Pfuetzenreuter

Fraunhofer Application Center System Technology
Am Vogelherd 50
98693 Ilmenau, Germany
Email: helge.renkewitz@iosb-ast.fraunhofer.de
Email: torsten.pfuetzenreuter@iosb-ast.fraunhofer.de

ABSTRACT

Today, it is not exceptional for an institution to own several different autonomous underwater vehicles. Despite the nice effect of having multiple platforms available, this can easily become a challenge for the user and even more for a system designer. Each vehicle is usually equipped with a native control system and hardware specific modules. The Fraunhofer Application Center System Technology in Ilmenau (Germany) currently possesses three underwater vehicles (both autonomous underwater vehicles (AUV) and remotely operated vehicles (ROV)); another novel vessel is being developed. All are equipped with individual guidance systems. Thus, modifying and creating new software, planning missions and evaluating them has to be done in very different ways. This fact shows the necessity to develop a new software framework for underwater vehicles. It is called ConSys (short for Control System) and offers several features that will be described in this paper.

***Index Terms*—** Autonomous Underwater Vehicles (AUV), Remotely Operated Vehicles (ROV)

1. INTRODUCTION

The Fraunhofer Application Center owns three underwater vehicles. Each vehicle has its individual control system that is very different from the others with respect to software module creation or modification, mission planning and evaluation. An ongoing project dealing with the construction of a new AUV (up to 6000m) for deep sea missions will provide us with another vehicle in the near future.

Operating those autonomous systems from different vendors is a challenge for the crews. Beside different mission preparation tasks like battery charging, sensor and actuator checks and communication setup, the mission planning procedures vary entirely from vehicle to vehicle. Typically, expert knowledge is required to plan and monitor the mission as well as to evaluate the collected data. While the data post processing and evaluation is an automatable job, planning and monitoring are usually time consuming and often error sources.

By creating an AUV of our own we were given the opportunity to develop an adaptable software framework for underwater vehicles called ConSys with the following characteristics:

- *Framework structure and communication:* supports for modular control systems with simple and powerful inter process communication mechanisms,
- *Abstraction layer:* complete abstraction layer for all needed interfaces to the underlying operating system, sensor and actor buses,
- *Graphical user interface:* easy to use, extensible, task-oriented application for mission planning and evaluation,
- *Vehicle independent control system:* support for development of a vehicle independent control system for AUVs and ROVs, including autonomous and teleoperated manipulation capabilities.
- *Messaging data structures:* based upon the framework infrastructure a number of domain-specific data structures are defined to distribute all required information between the AUV's software modules. For other use cases only new data structures are needed to create a very different control system.[1]

All three available vehicles to test and validate the software framework are of different size, weight as well as propulsion and steering configuration. The smallest vehicle called *Seebaer* is derived from a mine-disposal vehicle and equipped with a pan-tilt-zoom color camera (length: 1,30 m, weight: 40 kg, Fig. 1(a). It is powered by four stern thrusters and a vertical thruster located at the center of gravity and controllable via a fiber-optic link.

[1]The idea of domain-specific data structures is similar to the CORBA standard, but without the abstraction overhead used in the standardized architecture [1], [2].

(a) Seebaer

(b) Seewolf

(c) ExAuv

Fig. 1. Underwater vehicles

The *Seewolf* is a bigger version of the Seebaer since it shares the propulsion concept (4 stern thrusters, one vertical thruster, length: 2,00 m, weight: 120 kg, Fig. 1(b). Last year it was equipped with two lateral thrusters located near bow and stern to allow transverse movements, which are necessary for inspection of underwater structures like ship hulls, piers or sheet piles. Both Seebaer and Seewolf are currently modified with a computer module and navigation sensor in order to use them as autonomous vehicles in addition to the remote control feature.

The newest vehicle was designed by students to have an easy adaptable test vehicle for shallow waters up to 100 m and is called *ExAUV* (length: 0,84 m, weight: 37 kg, Fig. 1(c). With an ROV-like shape and six thrusters to control five degrees of freedom (DOFs) it is a mobile and versatile system that can operate autonomously as well as remotely controlled. The selected construction allows the modification of thruster positions, the integration of additional sensors or mounting an underwater manipulator.

This paper will focus on the structure of the software framework developed in C++, details on the implementation including problems typically arising during adaptation of new sensor and actor hardware, the ideas of the graphical user interface and the application of ConSys on the different vehicles owned by the Fraunhofer Application Center.

2. FRAMEWORK STRUCTURE

Applying the framework and its rules during software development leads to a clean, modular software design where functional units are individual software modules. This eases the design and test of the units, but requires more effort for data transfer (inter-process communication between modules) and associated synchronization overhead. In that it is similar to other software frameworks like MOOS or ROS [3], [4]. The following statements focus on the most important differences to these frameworks.

The communication structure builds on top of the Spread Toolkit, a high performance messaging service [5]. In Spread, one or more computers form a communication group. It allows dynamical join, leave, subscribe and unsubscribe operations from distributed applications. Messages can be served in unreliable, reliable or safe manner (more types applicable), bindings for a large number of programming languages are available.

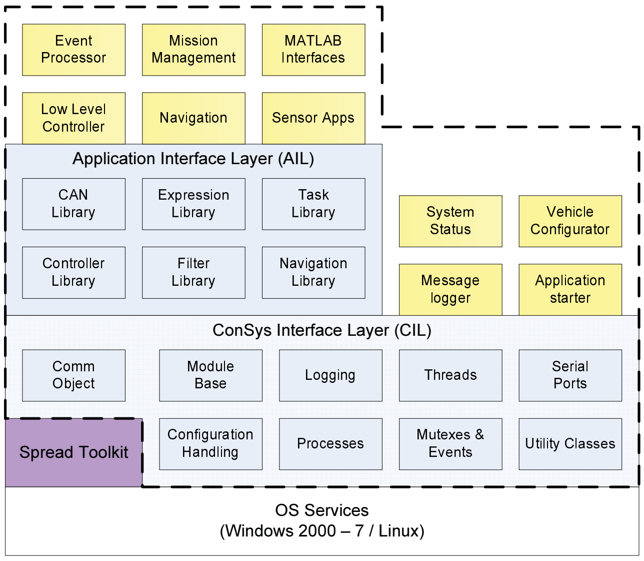
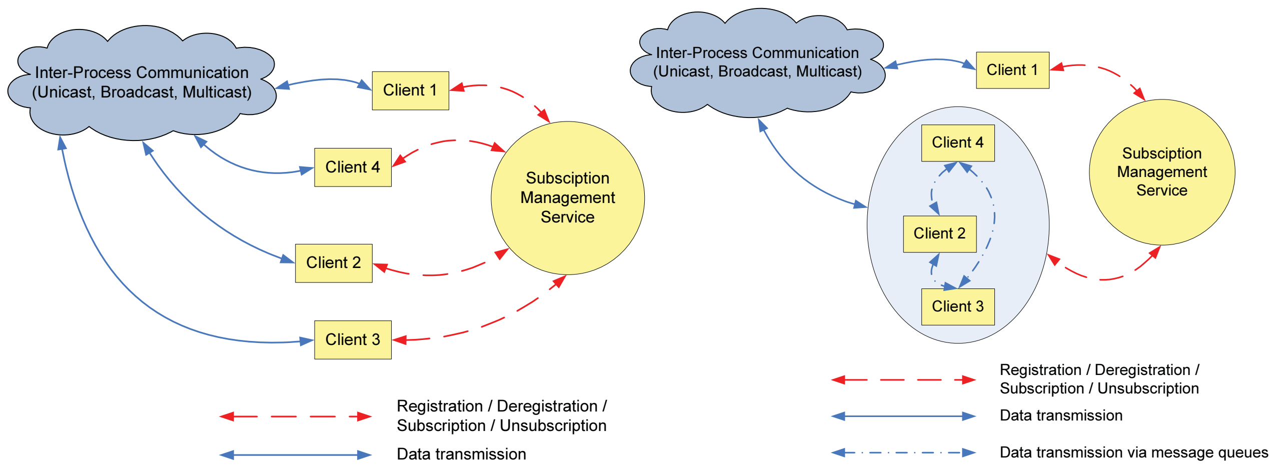


Fig. 2. ConSys framework overview

For the ConSys software modules the messaging system and its configuration is transparent - changes to message delivery settings or network configurations are handled in the toolkit. Modules need to connect to a Spread daemon (locally or centralized in the network) and join a group in order to receive ConSys messages (Fig. 3(a).

ConSys has the capability to create so called 'shared modules'. These are started as one process and consist of all functional units configured as shared modules (Fig. 3(b). The data transmission between these units occurs with very low latencies while the communication with other modules remains unchanged. Initially, this concept will be used for the navigation module linking the information of different sensors to form the vehicle's position and attitude estimation.


Inter-Process Communication (Unicast, Broadcast, Multicast)
Client 1
Client 4
Client 2
Client 3
Subsciption Management Service
Registration / Deregistration / Subscription / Unsubscription
Data transmission
Inter-Process Communication (Unicast, Broadcast, Multicast)
Client 1
Client 4
Client 2
Client 3
Subsciption Management Service
Registration / Deregistration / Subscription / Unsubscription
Data transmission
Data transmission via message queues

(a) Classical modules (b) Shared modules

Fig. 3. ConSys modules communication structure

3. ABSTRACTION LAYER

The ConSys framework introduces an operating system abstraction layer called ConSys Interface Layer (CIL) to form a platform-independent base for the software modules (Fig. 2). This layer contains the elemental elements for application programming:

- a communication class based on Spread Toolkit,
- a serial port class necessary for sensor interfacing,
- multi-thread support classes (threads, mutexes, events)
- logging facilities to save log messages from the modules,
- a module base class that incorporates the communication, configuration and message handling procedures and
- several other utility classes for easier module programming.

On top of the CIL a number of commonly used applications are implemented, for instance an application starter (starts and monitors all other modules), a system status observer (checks CPU load, memory consumption, etc.) or an universal message logger (logs messages sent via the communication channel).

Applications dealing with sensors and actuators or controlling the vehicles route require additional features that are located at the Application Interface Layer (AIL):

- The *navigation library* contains coordinate transformations, a serial port driver class and several sea water computation algorithms derived from [6] and [7]. Position estimation classes are available to compute the vehicle's location and attitude based on navigation sensor inputs [8].
- The *filter* and *controller libraries* are needed for input data filtering and vehicle control and consist of state-of-the-art algorithms.
- The *CAN library* aggregates the Controller Area Network (CAN) drivers for different hardware interfaces and CANopen device profiles for motor controllers (DSP-402) and general I/O modules (DSP-401) [9].
- The *expression library* is needed for evaluation of logical expressions and primarily used by the event processor application. The event processor is configured by scripts to check statements like $((\text{Depth} < 1.0)\ \&\&\ \text{WifiAllowed})$ and to react depending on the evaluation result with the execution of a shell script or the publication of a message.
- The *task library* contains a number of mission tasks that can be combined to form a complex mission. The mission management application uses this library to execute a mission and to re-plan it if necessary [10].

All these libraries are continuously extended with algorithms, drivers and mission tasks.

4. GRAPHICAL USER INTERFACE

Complex underwater systems like AUVs and ROVs are usually equipped with a wide range of sensors that produce a huge amount of data. This data can be automatically processed and evaluated but the user finally has to make decisions based on this information. In case of controlling a ROV, this can lead to a high cognitive load for an operator during a mission and makes it even more important to provide a graphical interface which is ergonomically designed [11].

As a matter of fact, controlling such a system with all its sensors and actuators requires an experienced user. But it's not always guaranteed to provide sufficient knowledge to the actual user of the system so it is very important to design the Graphical User Interface (GUI) according to well-proven guidelines (based on the DIN EN ISO 9421-110 [12]). By following these guidelines it can be guaranteed that a new user will not be confused by the system's operating mode or by the windows design and functionality.

Based on these thoughts the aspired GUI should have the following properties:

- highly customizable for each user
- adjustable at runtime
- load and save different layouts
- provide shortcuts for experienced users
- help system for new users

4.1. Current prototype

The GUI will be implemented in Nokias QT framework [13]. This software framework was chosen because it allows porting the application onto different operating systems. Developing the GUI is done by using Microsoft® Visual Studio® but the platform for the actual application will be a Linux system.

A graphical interface should provide a better view on the data that is processed in the ConSys core. Hence, an interface between the core application and libraries was necessary. Once the data is correctly delivered to the GUI, showing this information is only design and filter process.

Right now, the current prototype is able to receive data from the ConSys framework and to show this data in a QT application. Those applications are designed as widgets which will register automatically to the corresponding ConSys application (when they are started) and receive its data. Given this approach it is clear that all sensors will have an own sensor widget that can be placed anywhere in the mainwindow. The QT widget that allows this design is called Dockwidget. These widgets can be docked to a main widget, to other sensor widgets or even moved out of the actual window to become a flowing widget. This will be very useful if one or more sensors have to be observed very intensely.

As mentioned before, each user might have own preferences in how to design the GUI. Therefore, it is necessary to save one or more certain GUI layouts and load it later as they are required. This functionality is not only helpful for the individual desires of different users but also for defining pre-assembled layouts for specific vehicles. Imagine a set of layouts for each vehicle that can be loaded at runtime by just choosing the corresponding menu item.

One of the most important widgets in the GUI will be the mission planning tool. As mentioned before, processing and evaluating the sensor data can be automated but planning the mission for the vehicle will still be done by the operator. Thus, carefully designing this application is the most important part in the GUI implementation and will be described in the next section.

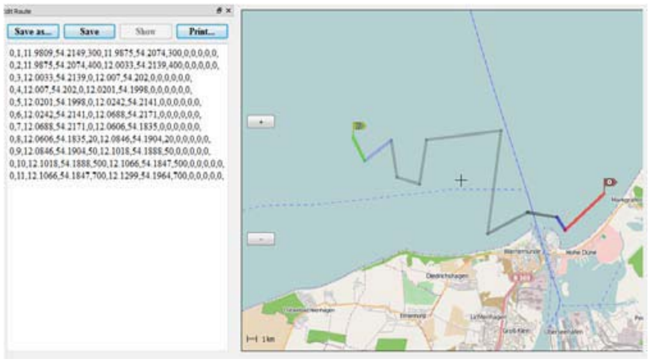
4.2. Mission Planning Tool

The map interface for the current mission planning tool is based on a LGPL QT widget called *QMapControl* [14] but was modified for the first approach. Its native functionality can be described as follows:

- it is compatible with many map providers
- custom objects can be drawn into the map
- new objects can be added to any coordinate
- the navigation is customizable
- map tiles can be stored persistently

At first, it should be possible to add routes or missions to these maps. Therefore, new functionality was integrated that allows the user to draw new routes or missions into the map. These routes can be displayed as visual overlays to the map (see figure 4) or as textual information into an additional route editor.

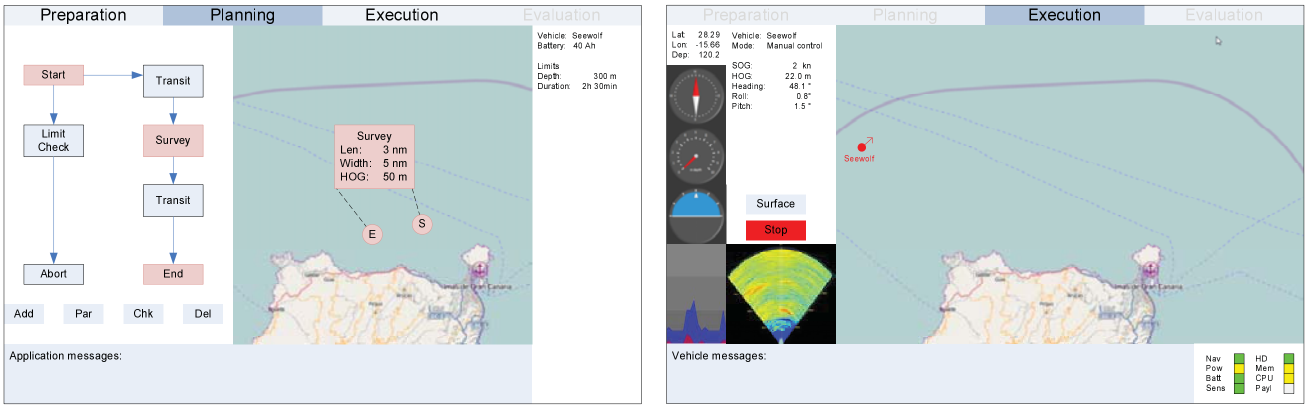
This editor shows each part of the mission with its corresponding properties like the type, the ID and the start/end position of the mission segment. Obviously, as the route and the mission is getting more complex, the mission plan and its corresponding data grows rapidly.



```
0,1,11.9809,54.2149,300,11.9875,54.2074,300,0,0,0,0,0,
0,2,11.9875,54.2074,400,12.0033,54.2139,400,0,0,0,0,0,
0,3,12.0033,54.2139,0,12.007,54.202,0,0,0,0,0,0,
0,4,12.007,54.202,0,12.0201,54.1998,0,0,0,0,0,0,
0,5,12.0201,54.1998,0,12.0242,54.2141,0,0,0,0,0,0,
0,6,12.0242,54.2141,0,12.0688,54.2171,0,0,0,0,0,0,
0,7,12.0688,54.2171,0,12.0606,54.1835,0,0,0,0,0,0,
0,8,12.0606,54.1835,20,12.0846,54.1904,20,0,0,0,0,0,
0,9,12.0846,54.1904,50,12.1018,54.1888,50,0,0,0,0,0,
0,10,12.1018,54.1888,500,12.1066,54.1847,500,0,0,0,0,0,
0,11,12.1066,54.1847,700,12.1299,54.1964,700,0,0,0,0,0,
```

Fig. 4. A mission plan with encoded depth information

Another disadvantage of a two-dimensional planning tool for underwater vehicles is that the information about depths has to be displayed somehow. In this prototype, depth information is stored in the mission plan as well - so experienced users can see this information in the editor. For unexperienced users the corresponding tracks or arcs are painted in the map with different colors. Figure 4 shows the encoded depth information.



Vehicle: Seewolf
Battery: 40 Ah

Limits
Depth: 300 m
Duration: 2h 30min

Survey
Len: 3 nm
Width: 5 nm
HOG: 50 m

Application messages:

Lat: 28.29
Lon: -15.66
Dep: 120.2

Vehicle: Seewolf
Mode: Manual control

SOG: 2 kn
HOG: 22.0 m
Heading: 48.1 °
Roll: 0.8°
Pitch: 1.5 °

Vehicle messages:

Nav	🟩	HD	🟩
Pow	🟨	Mem	🟨
Batt	🟩	CPU	🟨
Sens	🟩	Payl	⬜

(a) Mission planning: start position, survey and end position plannend by user, other elements are inserted automatically

(b) Mission execution: in teleoperated mode the most important vehicle data are shown beside the forward looking sonar and the sea map

Fig. 5. GUI design ideas

In this example you can see that the mission's operating depth at the beginning is different from the depth at the end. Taking a closer look at the editor will give the additional information that at the end of the mission the vehicle descends below the surface, then ascends to the surface and descends again at the end of the mission.

Some ideas of how to improve the User Interface are illustrated in figures 5(a) and 5(b). These conceptual pictures show the aspired type of GUI that will be finally used in our framework.

Here, the typical mission stages are shown at the top of the windows. You can see different tabs for preparation, planning and execution. This can also be extended to other stages like observation and evaluation. Each of these tabs has a certain layout and will be clearly distinguishable from each other.

Figure 5(a) shows the automated mission planning. As mentioned above, the current tool produces a huge amount of points for complex missions. An operator would need much time to set up such a complex mission. The new idea is simple: the user just inserts the start and end position (or a point of interest (POI)) and defines the type of mission (like exploration, inspection, etc.). All the other steps will be done by the application. Given the type of mission, environmental parameters (sea maps, sensor data, vehicle dynamics etc.) the tool will automatically generate a route with subtasks. In this figure, the subtasks include starting and ending the mission, transitting the vehicle and survey special areas.

The information displayed on the screen is just the one the user entered (it can be seen at the right side of 5(a)). Reducing the information avoids the information overload of the operator and allows inexperienced users to work with the system. But it has to be guaranteed that there will be a mode available, where the operator can check the automatically generated data in order to change values manually.

As the planning is done, a user can switch to the execution tab shown in figure 5(b). Here, the current vehicle(s) are shown on the map and the most important sensor data can be displayed at the side of the window. These sensor widgets can be arranged as the operator desires. A small status control is shown at the bottom right corner of the screen. With an information display like that crucial information like battery charge status, communication link and error messages can be displayed (encoded into colors).

Special configuration files for different vehicles allow the creation of special layouts for every vehicle. Infact, none of the vehicles have identical sensor equipments. Configuring the GUI according the vehicle could be done once - with all the sensors provided - and then be saved. As the system is started, the configuration file is parsed and as soon as a certain vehicle is defined its corresponding GUI layout is automatically loaded. Registering the sensors and actuators and giving the operator the ability to work as the system is ready.

5. VEHICLE INDEPENDENT CONTROL SYSTEM

The different vehicles that will be used for evaluation of the ConSys framework have different propulsion configurations. A common data structure is defined for the controller configuration as well as for the controller set points. The controller configuration message contains the information needed to set up the low level controller of the vehicle in question:

- The *control mode* defines what DOFs are controlled by an automatic control algorithm (in autonomous mode all functions will be controlled and in pure teleoperated mode no one). This feature enables the assisted teleoperated mode where some degrees (e.g. course, pitch, depth) are controlled automatically and the operator can concentrate on his primary tasks, for instance to manipulate object with an robotic arm.

- The remaining thrusters get the demanded rotational speed.[2] For these actuators the controlling software module sends the *thruster identifiers* (names) in the same sequence as it send the speed values during operation.

The controller set points message consists of:

- *controller setpoints* for controlled DOFs: depth, forward speed, lateral speed, course and pitch,
- *rotational speed* for thrusters that are not controlled.

A low level controller application is responsible for driving the actuators. Therefore it transforms the setpoints into control actions (if automatic control mode is active for one or more DOFs) or directly sends the desired rotational speed to the thrusters. The low level controller contains control modules for the different vehicles.

6. CONCLUSION

The software introduced in this paper could help developers and operators of ROVs and AUVs to ease their work. Using one software framework on different underwater vehicles is unique and will provide the opportunity to rapidly implement new components for an existing system. A common GUI which is designed according to Human Factors should increase the performance of the operators.

Acknowledgment

The authors would like to thank the Fraunhofer Gesellschaft in Munich, Germany for funding this research work via its MAVO program. It is part of the MAVO project TIETeK of several Fraunhofer institutes to develop a deep diving autonomous underwater vehicle. A specific feature of this vehicle will be the pressure neutral design of most of the components reducing the vehicle's weight, size and costs.

7. REFERENCES

[1] Michi Henning and Steve Vinoski, *Advanced CORBA programming with C++*, Addison-Wesley, Reading, Mass., 1999.

[2] Michi Henning, "The rise and fall of corba," *Queue*, vol. 4, no. 5, pp. 28–34, 2006.

[3] Paul Newman, "MOOS-a mission oriented operating suite," Tech. Rep. OE2003-07, MIT Department of Ocean Engineering, 2003.

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

[5] Yair Amir, Claudiu Danilov, Michal Miskin-Amir, John Schultz, and Jonathan Stanton, "The Spread toolkit: Architecture and performance," Tech. Rep. CNDS-2004-1, Computer Science Department, Johns Hopkins University, 2004.

[6] N. P. Fofonoff and R. C. Millard, "Algorithms for computation of fundamental properties of seawater," *UNESCO Technical Papers in Marine Science, 44*, 1983.

[7] J. M. Pike and F. L. Beiboer, "A comparison between algorithms for the speed of sound in seawater," Tech. Rep. Special Publication No. 34, The Hydrographic Society, 1993.

[8] Torsten Pfuetzenreuter, Thomas Rauschenbach, and Juergen Wernstedt, "Multisensor fusion for navigation of underwater vehicles," in *Proceedings of the 2006 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2006)*, Heidelberg, Germany, Sept. 2006, pp. 151–154.

[9] Olaf Pfeiffer, Andrew Ayre, and Christian Keydel, *Embedded Networking with CAN and CANopen*, Copperhill Media Corporation, 2008.

[10] Torsten Pfuetzenreuter, "Advanced mission management for long-range autonomous underwater vehicles," in *OCEANS 2003. Proceedings*, 22-26 Sept. 2003, vol. 2, pp. 928–933.

[11] Markus Dahm, *Grundlagen der Mensch-Computer-Interaktion*, Pearson Studium, München [u.a.], 2006.

[12] "DIN 9421-110: Ergonomics of human-system interaction - Part 110: Dialogue principles," 4 2006.

[13] "Qt 4.6: Qt reference documentation," http://qt.nokia.com/doc/4.6/index.html.

[14] Kai Winter, "QMapControl - a Qtopia widget for map applications on mobile devices," diploma thesis (in german), Design Computer Science Media Department, University of Applied Sciences Wiesbaden, 2008, Original title: "QMapControl ein Qtopia-Widget für Kartenanwendungen auf mobilen Geräten".

[2] If the vehicle has control fins, the desired angles will be transmitted.