ConSys - a New Software Framework for Underwater Vehicles

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

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

***Abstract*— Today a large number of autonomous underwater vehicles (AUVs) are evaluated or operated all over the world. Most of them use their own control systems created by the vehicle's manufacturer or scientists of different research areas. The Fraunhofer Application Center System Technology currently owns three underwater vehicles (both AUVs and remotely operated vehicles, ROVs), a forth is under development. All these vehicles have their individual control systems that are very different with respect to changing / creating software modules, mission planning and evaluation. These differences are one reason to develop a new software framework for underwater vehicles called ConSys.**

I. Introduction

Operating a number of autonomous systems from different vendors is a challenge for their 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 very time consuming and important error sources.

We at Fraunhofer Application Center own three underwater vehicles. All these vehicles have their individual control systems that are very different with respect to software module creation or modification, mission planning and evaluation. An ongoing project dealing with the construction of a deep diving AUV (up to 6000m) gives the opportunity to develop an adaptable software framework for underwater vehicles called ConSys with the following characteristics:

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

The three vehicles available 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.

The *Seewolf* is the big brother 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)). In the 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 refitted with a computer module and navigation sensor to run 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, the ideas of the graphical user interface and some information about the adaptation of ConSys on the different vehicles owned by the Fraunhofer Application Center.

[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].

978-1-4244-5222-4/10/$26.00 ©2010 IEEE

(a) Seebaer (b) Seewolf (CAD model) (c) ExAUV (CAD model)

Fig. 1. Underwater vehicles

II. Framework Structure

Applying the framework and its rules during software development leads to a clean 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 group communication system [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, bindings for a large number of programming languages are available.

In ConSys a communication group is set up for each vehicle, the data transfer between groups is handles by Spread. For the software modules the GCS 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. 2(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 (figure 2(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.

Developing a new software module, like a sensor driver or a high-level control application, is facilitated by a number of design rules and code fragments. Listing 1 shows the C++ code required to implement a navigation data subscriber. Basically, one has to define the module name, the messages the module receives or publishes and to fill this into the code skeleton. The base class ConSys::AppBase contains the code for communication setup, logging, configuration file handling as well as for message sending and reception.

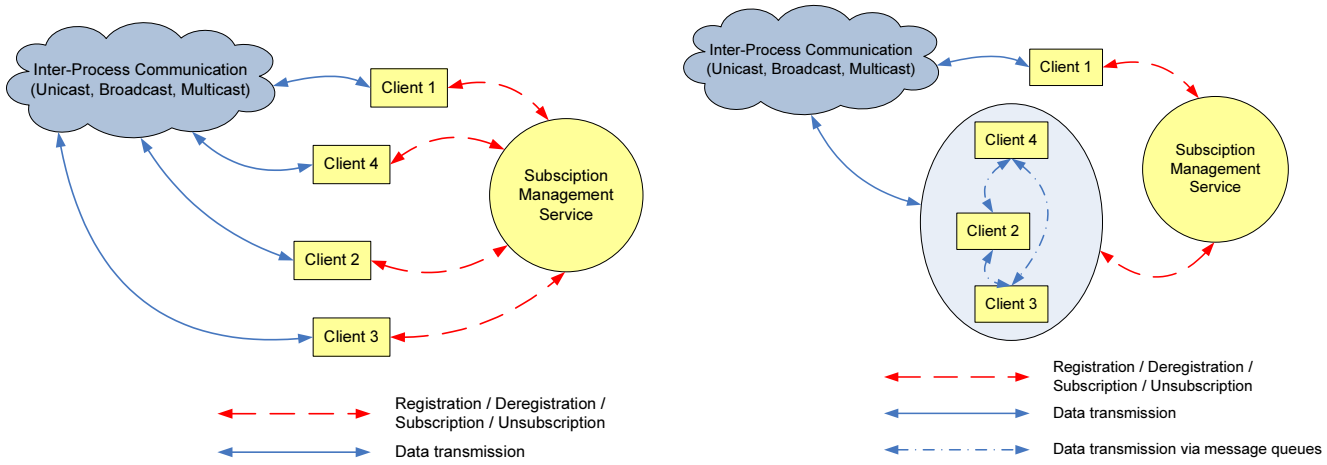
Listing 1
NAVIGATION DATA SUBSCRIBER MODULE

```
#include <consys/AppBase.h>
#include <interfaces/NAV_Data.h>

CONSYS_APP_CLASS(NavSubscriber)
{
public:
  CONSYS_APP_CONCSTRUCTOR(NavSubscriber)
  {
  }
  virtual void OnHandleMsg(const
                 ConSys::ConSysMsg& msg)
  {
    NAV::Data navData;
    if (fromConSysMsg(msg, navData)) {
      printf("Received NAV_Data message\n");
      return;
    }
    ConSys::AppBase::OnHandleMsg(msg);
  }
  virtual void OnSetup()
  {
    SubscribeMessage("NAV_Data");
  }
  virtual void OnRunLoop()
  {
  }
  virtual void OnCleanup()
  {
  }
};

CONSYS_MAIN("NavSubscriber", NavSubscriber)
```

III. 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. 3). This layer contains the usually needed elements for application programming:

- the communication class basing 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) Classical modules (b) Shared modules

Fig. 2. ConSys modules communication structure

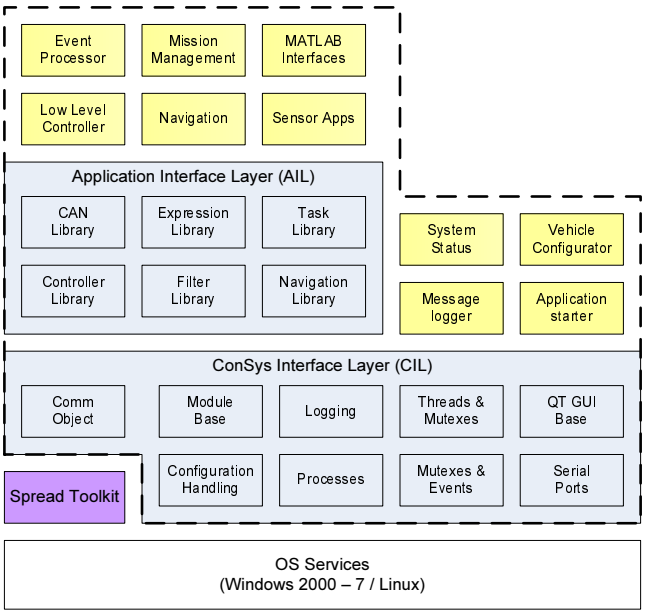


Fig. 3. ConSys framework overview

- 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). One remarkable tool is the Vehicle Configurator that merges a configuration template for ConSys modules with a vehicle configuration to create the final ConSys configuration file.

Application dealing with sensors and actuators or controlling the vehicles flight path 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 replan it if necessary [10].

IV. 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)

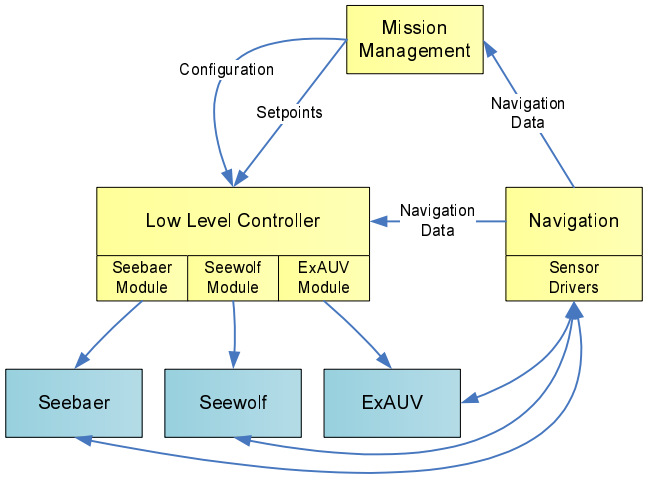


Fig. 4. Simplified message flows for vehicle abstraction

are controlled automatically and the operator can concentrate on his primary tasks, for instance to manipulate objects 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 sends the speed values during operation.

The *controller setpoints message* consists of:

- *controller setpoints* for controlled DOFs: depth, forward speed, lateral speed, course and pitch,
- *rotational speed* for thrusters that are not controlled in the same sequence as described above.

A low level controller application is responsible for driving the actuators (Fig. 4). 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. Using this control scheme an easy adaptation to new vehicles seems possible even if they differ in propulsion and steering configuration.

V. Graphical User Interface

Underwater vehicles like AUVs and ROVs usually are 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 teleoperating a vehicle, 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.

A. Current prototype

This section describes the current prototype of our GUI as it is in a development process. It does not show the final application; moreover a recent snapshot.

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 are processed in the ConSys core. Building the GUI is supported by a base class for all QT based applications that handles the data transmission from an to one or more ConSys communication groups. Right now, the current prototype is able to receive data from the ConSys framework and to show these data in a QT application.

Next change to the prototype is to transform it into a container application. In this container, a number of graphical elements (widgets in QT speech) can be arranged to create the desired user interface. Each widget will be able to register for messages it needs to display information and to publish messages if it takes an user action as input.

Given this approach it is clear that all sensors will have an own sensor widget that can be placed anywhere in the main window. 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 different GUI layouts and load them 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. This functionality is yet implemented and ready to use.

Currently, rather common functionality is also integrated into the existing prototype. This includes intuitive and useful functions like sending the whole GUI into a fullscreen mode or the utilization of well known shortcuts (for example Ctrl+S to save). This guarantees that normal users can operate with the GUI in a way that they are used to.

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

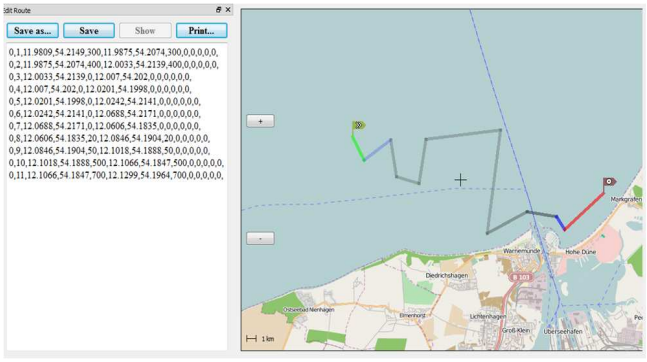


Fig. 5. A mission plan with encoded depth information

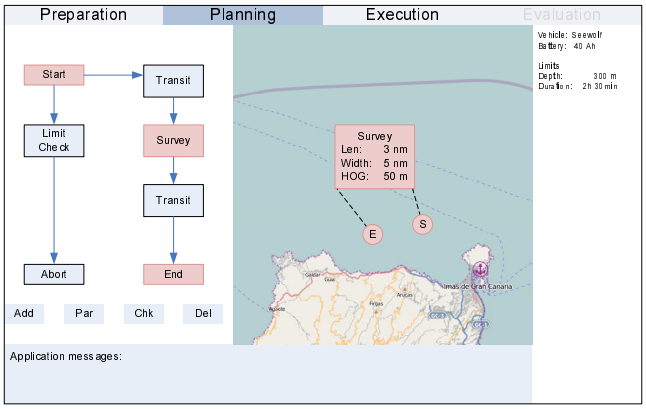


Fig. 6. Mission planning: start position, survey and end position plannend by user, other elements are inserted automatically

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.

B. Mission Planning Tool

The map interface for the current mission planning tool is based on a 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 (fig. 5) 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.

Another disadvantage of a two-dimensional planning tool for underwater vehicles is that the information about depths has to be displayed somehow. This is not a trivial matter because it should be very intuitive for the user. In this prototype, depth information is stored in the mission plan as well - so experienced users can see this information in the editor. But what about inexperienced users? The solution was to encode this information into colors and to paint the corresponding tracks or arcs in the map with different colors.

Figure 5 shows the encoded depth information. 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.

This current snapshot clearly has a lot of potential for improving the tool and add more functionality. Some of the functions are working but not the best solution, yet. One example is the color encoding for the depths. In a future version this could be replaced with a depth diagram or even more in the future with a 3D widget for planning the depth route. But this can only be done when reliable depth information (sea maps, scans, etc.) are available.

Some ideas of how to improve the user interface are illustrated in fig. 6 and 7. 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 screen. 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 customizable layout and will be clearly distinguishable from each other.

Figure 6 shows the automated mission planning. Compared to the current prototype the new approach will speed up the mission planning process by supporting the operator with an automatic mission completion. The user just inserts the start and end positions, defines the type of the mission (like survey, inspection, etc.) and marks the desired mission area (or route, e.g. for a pipeline inspection). The missing connections between start and end positions and the mission area are created by inserting transits (or descent and surface maneuvers) as well as the safety check task depending on the parameters of the current vehicle and the amount of energy available for the mission.

The information displayed on the map is just the one the user entered. Reducing the information avoids the information overload of the operator and allows inexperienced users to work with the system. A feature that will mostly be used by experienced users is to show or change the automatically generated mission tasks also on the map.

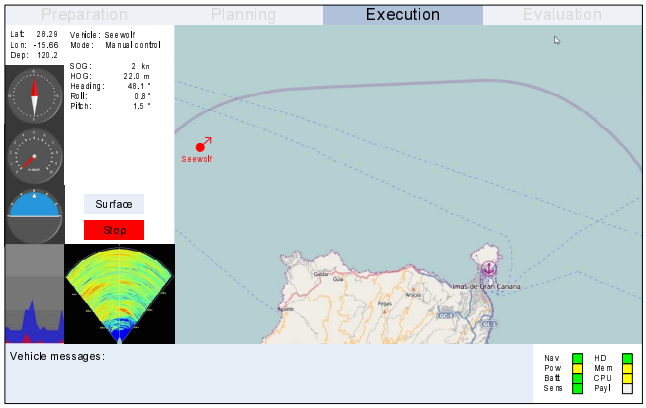


Fig. 7. Mission execution: in teleoperated mode the most important vehicle data are shown beside the forward looking sonar and the sea map

As the planning is done, a user can switch to the execution tab (fig. 7). Here, the selected 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 or sized as the operator desires. A small status control is located at the bottom right corner of the screen that displays crucial information like battery charge status, communication link and error messages encoded into colors.

Special configuration files for different vehicles would allow creating special layouts for every vehicle. As a matter of fact, 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 desired vehicle is selected, its corresponding GUI layout is automatically loaded.

VI. Summary

This paper presented a new software framework for developing the advanced and autonomous functions of underwater vehicles called ConSys that eases the creation of new software modules. ConSys will be applied to the ROVs and AUVs currently owned by the author's institution to collect experiences and to further enhance the framework.

The framework is not limited to underwater vehicles since its base layer only encapsulates the essential functions and classes of a modular system with advanced messaging support. Even though some of the application layer libraries are orientated on maritime sensors and their data processing, most of the software code of the framework and its applications are directly portable to a new robot vehicle domain.

A very important part of a software system for unmanned vehicles is the graphical user interface. Here an intuitive, customizable design and support functions for both unexperienced and skilled users are the key functions for consumer acceptance.

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.

References

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

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

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

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

[5] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, and J. Stanton, "The Spread toolkit: Architecture and performance," Computer Science Department, Johns Hopkins University, Tech. Rep. CNDS-2004-1, 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," The Hydrographic Society, Tech. Rep. Special Publication No. 34, 1993.

[8] T. Pfuetzenreuter, T. Rauschenbach, and J. 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, Sep. 2006, pp. 151–154.

[9] O. Pfeiffer, A. Ayre, and C. Keydel, *Embedded Networking with CAN and CANopen*. Copperhill Media Corporation, 2008.

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

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

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

[13] "Qt reference documentation," http://qt.nokia.com/developer.

[14] K. 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".