FAST PROTOTYPING OF AN AUTONOMOUS UNDERWATER VEHICLE WITH THE CUBESYSTEM

Diana Albu, Andras Birk, Petar Dobrev, Farah Gammoh, Andrei Giurgiu, Sergiu-Cristian Mihut, Bogdan Minzu, Razvan Pascanu, Sören Schwertfeger, Alexandru Stan, Stefan Videv [1,2]

Abstract: This paper describes the Jacobs autonomous underwater vehicle (Jacobs-AUV). The vehicle has been largely developed using the CubeSystem, a collection of hard- and software components for fast robot prototyping. The CubeSystem has previously been used for the development of various mobile robots, but never for an underwater system. As the development of the Jacobs-AUV shows, a significant amount of component reuse across significantly different application domains is possible for autonomous robots.

Keywords: Autonomous Underwater Robotics, Fast Prototyping

1. INTRODUCTION

Underwater robotics poses substantial scientific challenges, especially with regard to autonomy in this field [Yongkuan (1992)]. Underwater exploration is of tremendous interest from the industrial as well as from the scientific perspective, let it be in shallow coastal waters or in the deep sea. Examples include the exploration of seismic and volcanic activities [Ura et al. (2001)], environmental monitoring [Uliana et al. (1997); Craik (1986)], archaeological research [Coleman et al. (2000)] and of course the exploration of natural resources [Nebrija et al. (1976)]. An interesting question is to which extent the development of an autonomous underwater vehicle (AUV) can be facilitated by robotics prototyping tools.

Many projects assume, that a robots hard- and software are two rather distinct parts, that can be easily brought together by the usage of the right type of abstractions and interfaces [Mallet et al. (2002); Bruyninckx (2001)]. The so-called CubeSystem in contrast tries to offer a component collection for fast prototyping of complete robots, i.e., the hardware and the software side [Birk (2004)]. The center of the CubeSystem is the so-called RoboCube controller hardware [Birk et al.

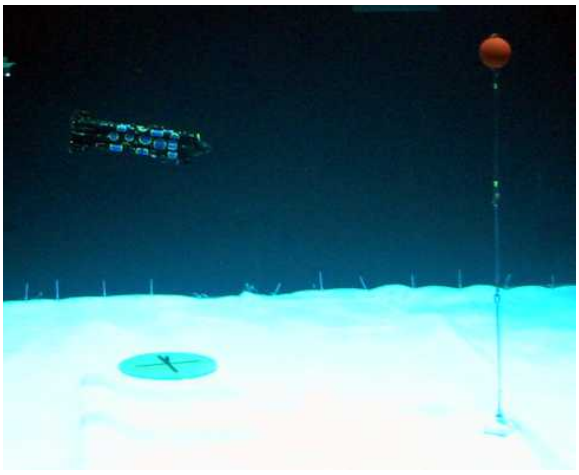


Fig. 1. The AUV searching for a midwater target in form of a submerged orange buoy.

[1] We are all Jacobs, Robotics, EECS, Jacobs University Bremen (previously International University Bremen), D-28725 Bremen, Germany. a.birk@iu-bremen.de

[2] This work is partially supported by *ATLAS ELEKTRONIK*.

(1998)]. On the software side, the CubeSystem features a special operating system, the CubeOS [Kenn (2000)], and libraries for common robotics tasks [Birk et al. (2002)], supporting teleoperation as well as autonomy [Birk and Kenn (2003)].

The CubeSystem has been used in various robotics applications, ranging from educational activities [Asada et al. (2000)]over basic research [Birk and Wiernik (2000); Birk et al. (2002)] to industrial applications [Birk and Kenn (2002, 2001)]. It has been noted before that the CubeSystem indeed supports component re-use for the development of autonomous intelligent systems [Kenn and Birk (2005)]. Here it is shown that this is also feasible across substantially different application domains.

2. THE AUV HARDWARE

The Jacobs-AUV (Fig. 1) is intended for basic research and robotics education purposes. Some of the most basic hardware parts, namely the hull, the motors and the batteries, are based on parts from a so-called Seafox ROV that was provided by ATLAS ELEKTRONIK. The complete development of the electronics, the sensors, the on-board computer, the actuators, and software was solely done by the Jacobs team. The AUV uses two computation units, which is a common approach for CubeSystem applications [Birk (2004)]. The basic hardware control is done by a RoboCube. Higher level AI software is running on an embedded PC. The robot can either operate autonomously or be controlled via a wireless link which is relayed by an antenna-buoy.

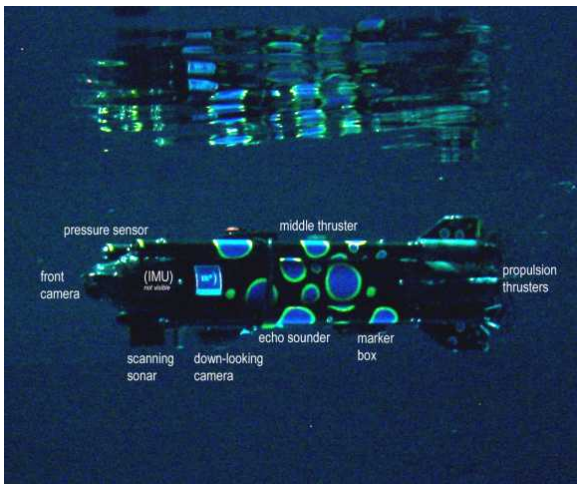


Fig. 2. The main sensors of the AUV.

The submarine is composed of a main hull, four long tubes and a nose which contains the pressure sensor, the gyro, a high-resolution scanning sonar head and the front camera. The Cube System, the high level controller (PC), the wireless access point and the DC/ DC converters are placed inside the middle section, while the echo sounder, the marker disposal system and a bottom camera are outside of the middle section. The vertical middle thruster is mounted in the middle section in an open ended cylinder perpendicular to the vehicle body. Four battery tubes are attached to the middle section, each containing a battery and a propulsion motor with electronics for the motor control and guarded propellers. The four thrusters allow a kind of differential drive steering in a horizontal, respectively vertical plane. To maximize re-use of software from land robots, the AUV mainly uses differential driving in the horizontal plane and adjusts its depth as desired with the middle thruster.

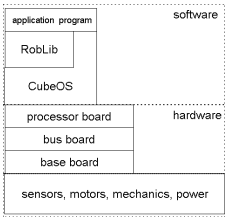
The submarine is trimmed slightly positively buoyant, i.e. it comes up to the surface in case of some error. Thus the middle thruster has to be used to force the vehicle under water. The maximum power of the four propulsion motors is 350 W of which up to 280 W are usable due to PWM duty cycle limitations.

2.1 Low level controller

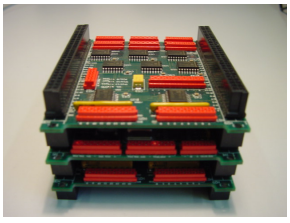
As mentioned before, the CubeSystem is a collection of hardware- and software-components for fast robot prototyping. The main goal of the CubeSystem project is to provide an open source collection of generic building blocks that can be freely combined into an application.

The most basic parts of the CubeSystem are

- *RoboCube*: a special embedded controller, based on the MC68332 processor
- *CubeOS*: an operating system
- *RobLib*: a library with common functions for robotics



(a) The structure of a CubeSystem application.



(b) A processor-, bus- and I/O-board stacked together.

Fig. 3. The CubeSystem.

The general CubeOS and RobLib are collections of software components that can be customized and combined to a particular set of libraries suited for a particular application. The CubeOS has for example POSIX constructs to write realtime code. The RobLib provides for example generic functions for controlling motors via Pulse-Width-Modulation (PWM). The CubeSystem also allows

quite some flexibility to adopt the hardware side to the actual application. The RoboCube or short Cube features some standard electronic components like the processor-, bus-, and I/O-board that are combined with a more application specific base-board. The boards are equipped with a special stacking connector that allows to put several boards on top of each other. The compact form factor of the boards and the stacking leads to a cubic shape of the controller (Fig. 3), hence the name Robo*Cube*. The organization of the Cube architecture can be thought of in a tree-like manner. At the root is the minimal Cube, namely a processor-board which can be expanded by a bus-board. This new branch adds new functionalities like, e.g., UARTs and I^2C controllers that allow to expand to further branches, for example in form of certain sensors.

So, a CubeSystem application is a concrete instance of a collection of components, for example in form of a mobile soccer robot. The task for the Jacobs AUV team was hence to design a suited baseboard and low level software based on RobLib functions. The particular challenge is the number of motors, namely five, in contrast to the usual number of two on land-based robots. Furthermore, locomotion is harder as it is in 6D.

The AUV is just like any mobile robot a typical embedded system with an according development environment. The structure of the development environment is shown in figure 4. The CubeSystem is connected via a serial connection to the so-called access-host. This connection is a simple RS-232 cable. The AUV CubeSystem is programmed for efficiency reasons only in C. The generation of executables, i.e., compilation and linking, is done on a so-called compile-host. In doing so, the GNU tool chain with an according cross-compiler is used. The generic CubeOS and Roblib, as well as customized and pre-compiled instances, are also located on the compile-host.

2.2 High level controller

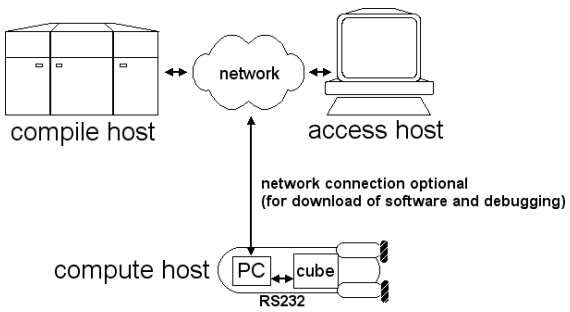


Fig. 4. Different hosts used for the AUV.

The RoboCube with its MC68332 micro-controller is not suited for any high-level computation that are required for fully autonomous control of an underwater vehicle. Like in many other Jacobs robots an additional PC is used as a so-called compute host, which is also dubbed "cognition" unit. For the submarine, a Sumicom S625F car-computer is used as compute host. This PC is a fanless computer with a Celeron M 370 at 1.5 GHz, it weighs about 1.5 kg and it is very compact.

2.3 The Sensor Equipment

2.3.1. The Heading Sensor A MTi IMU from Xsens is used. It is a low-cost miniature inertial measurement unit with an integrated 3D compass. It has an embedded processor capable of calculating the roll, pitch and yaw in real time, as well as outputting calibrated 3D linear acceleration, rate of turn (gyro) and (earth) magnetic field data.

2.3.2. Scanning Sonar The submarine has a hight resolution scanning sonar which enables it to scan the surrounding for obstacles and their distance by emmitting soundsignals of 550 kHz and measuring the time it takes until the sounds echo comes back. The sonar covers up to 360 degrees since it is being turned by a motor in steps of one degree or greater and it has an opening angle of 1.5°. The range of the sonar is selectable to up to 80 meters. The sensor can also scan vertically with a coverage of ± 20°. The horizontal scan rate is about 60° per second at distances of up to 10 m.

2.3.3. Pressure sensor The pressure sensor measures the depth of the submarine below the water surface. It has an operating pressure range from 0 to 31 bar thus being operational to depth of 300 meters. It can be directly interfaced to the A/D converters of the RoboCube.

2.3.4. Echo sounder The echo sounder measures the distance from the submarine to the ground by emmitting sound pulses with a frequency of 500 kHz and a width of 100 μs. The echo of this sound signal is being received and the travel time measured is then used to calculate the depth. The beam width is ± 3°, the accuracy ± 5 cm and the depth range 0.5 to 9.8 meters.

2.3.5. USB cameras Two Creative NX Ultra USB cameras are used in the vehicle. They support a resolution of up to 640 x 480 pixel. Those USB 1.1 devices have a wide-angle lens which enables a field of view of 78°. Both cameras are inserted in a waterproof protective lid of transparent plastic.

2.4 Power

The submarine has four tubular packs of nickel-cadmium rechargeable batteries with 29 volts that can supply the AUV for more than two hours. The battery power is connected to different DC/DC converters. One converter is used to deliver the 12V needed for the PC, the access point and the Sonar, another 12V converter powers the cube and the motor relays while a third DC/DC converter with an output voltage of 5V is connected to a USB hub and the gyro. An external switch is used to cut off the power for all systems. A pushbutton is connected to the binary input of the Cube. If this button is pressed all motors are stopped.

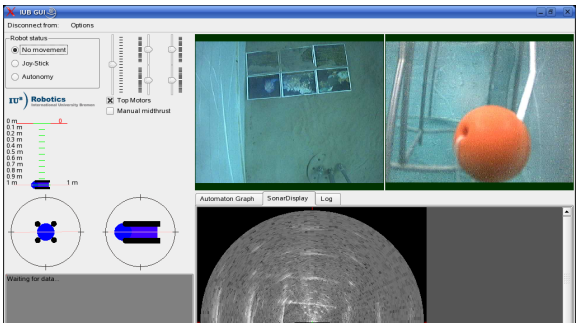


Fig. 5. The GUI showing the front and down camera as well as the depth, roll and pitch and the scanning sonar display.

3. THE AUV SOFTWARE

The software for the AUV is structured like in other CubeSystem robots in two parts [Birk and Kenn (2003)]. The basic low level control is done on the RoboCube. It uses the CubeOS and the RobLib to generate the proper PWM signals, to decode the encoder signals from the motors and to access the analog pressure and echo sound sensors.

The higher level AI software is running on the Linux compute host. It collects all sensor data from the cube, as well as from the sensors directly attached to the PCs interfaces. It uses the scanning sonar to do obstacle avoidance and localize other objects in the basin. The cameras are used to find targets in the water or at the bottom. The software reuses quite some functions and libraries developed for other Jacobs Robotics projects, especially the rescue robot.

The AI software is embedded in a robot-server, which is a multi-threaded program written in C++. All system-wide constants like port numbers, resolutions, etc., are read at startup from a configuration file. A client GUI running on a PC or a laptop is connecting to this server in order to manually drive the robot to a start position using a gamepad, to start the autonomy and to observe the submarine's status during the mission if wireless connection is still available (Fig. 5).

The NIST RCS framework [Albus (1 April 1997)] is used to handle communication between the robot-server and the operator GUI. This framework allows data to be transferred between processes running on the same or different machines using Neutral Message Language (NML) memory buffers.

3.1 Finite State Automaton

During a mission autonomous underwater vehicles have to perform different tasks. In order to allow fast and reliable mission planning we use a finite state automaton which is defined in a human readable way as a text file. For debugging and during a run this automaton is shown as a graph within the GUI, indicating the current state and the last transition.

This finite state automaton consists of states, transitions (start with a $), conditions and actions (start with a #). At a new iteration all actions of a state are executed in the order they were specified. Actions can actively manipulate the submarine by, for example, changing the desired speeds of the thrusters, by opening the marker box or by starting a timer. Only after all actions have been called the new motor speeds are applied, allowing for nice implementation of behavior-based robotics. After that the conditions of the transitions are checked. New actions and conditions have to be implemented in C++. The first transition whose condition is true is then being used to change the state of the automaton. If no condition is true the state of the automaton is not changed. A small example automaton is shown here:

```
START_STATE
{
  $ lets_go : trueCond => GO_TO_MEDIUM_DEPTH
  # StartMissionTiming(100)
}
GO_TO_MEDIUM_DEPTH
{
  $ missionEnd : isMissionTimeOver => MISSION_ENDING_STATE
  $ reachedMediumDepth : reachedMediumDepth => DRIVE_STATE
  # goToMediumDepth()
  # StartTimer(driving_over,15)
 }
[...]
}
```

The automaton is executed as a speed of 10 Hz. This means that the actions as well as the conditions can block the execution only for a very short time. Actions and conditions have a unique string identifier that corresponds to the actual implementation of those action or condition in the code.

The visualization that is being generated using the graphviz library is shown in figure 6.

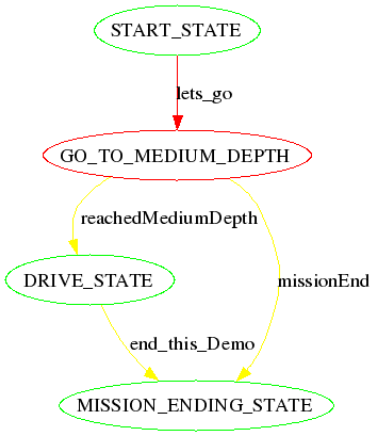


Fig. 6. The automaton controlling the AUV. The current state and the last transition are colored red.

3.2 Submarine Control

An example for a basic behavior is obstacle avoidance. The data from the seeking sonar head are used for this purpose. Unlike laser range finders the sonar returns a value for every distance which is sampled in discrete steps. That is why a minimum echo intensity for obstacles has to be defined such that the closest sample above this value determines the closest obstacle in this direction. The closest obstacle distances from the left, the right and the front of the vehicle are then extracted and used to change the motor speeds in order to avoid collisions. A problem is the relatively slow speed of the seeking sonar which takes six seconds for a complete 360 degrees scan. Since the submarine has a decent speed the update rate of the sonar makes the obstacle avoidance relatively unreliable.

The AUV tries to work as much as possible in a horizontal plane to exploit 2D differential steering, which can serve as basis for standard mapping, path planning, exploration, and so on. The extension to 3D can be done by so-to-say switching between different levels. In doing so, the depth is controlled by a standard PID controller using the middle-thruster and the echo sounder.

3.3 Vision processing

An open source Palantir server running on the robot PC is used to serve the images from the webcams so that different application can access them - in this case the images are used by the vision system on the submarine itself as well as the GUI interface on the operator station. The image processing is done in two steps. The first step is always a segmentation of the picture. This implies deleting all pixels that are considered to not belong to the searched object. In our case this was decided based on the color of the object by defining color ranges that should be accepted.

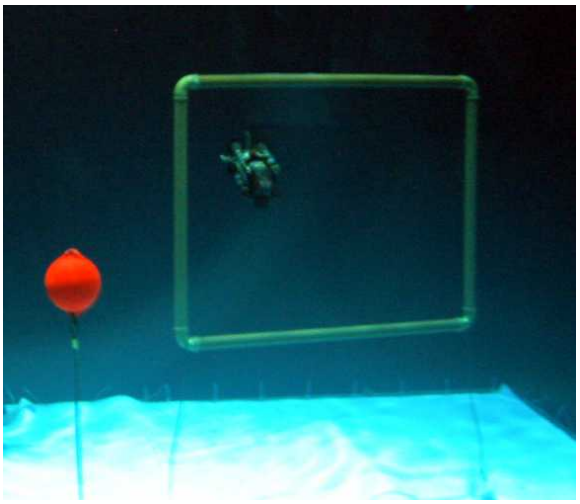


Fig. 7. The AUV autonomously approaching a gate at SAUC-E.

After the segmentation a second step is either hough transform or detecting the largest blob of color. Both algorithms have their advantages and disadvantages, therefore the user has to make the decision when designing the automaton. The hough transform is slower, but performs better with noisy data. On the other hand detecting the largest blob of color is fast and usefull for situations when the object is not completely in the field of view of the camera used. Hough transform returns the lines on which most of the points are, while the other algorithm returns the bounding box sourounding the largest connected blob of color as well as the weight center and the total number of points found. The actions specified in the finite state automaton define which strategy is used.

3.4 The Facilitation of Re-Use with the CubeSystem

The AUV was developed within only three months. A major part of this effort was done by undergraduate students as part of free time activities within the Jacobs robotics club. The only reason why this is possible is that the CubeSystem supports an effortless re-use of components from different systems. The AUV is to a large extent identical with an autonomous rescue robot developed within the robotics group of Jacobs University Bremen [Birk et al. (2006); Birk and Carpin (2006)]. This holds with respect to hardware as well as software.

The amount of re-use can be illustrated as follows. For the software, a detailed analysis of the relevant code reveals that 91.3% are identical. Quantifying this for the electronics side is more difficult. A rough indicator in form of the ratio of the PCB-area of identical versus different components that are not off the shelf can be used. With this indicator, it can be seen that 79% of the electronic components are indeed re-used.

4. CONCLUSION

The Jacobs-AUV is presented here. It is based on the CubeSystem, a collection of hardware and software components for fast robot prototyping. The fast and successful development of the AUV, which was to quite some extent done by undergraduates as free time activity in a robotics club, shows that component re-use is even possible across robots developed for very different application domains.

REFERENCES

James S. Albus. The nist real-time control system (rcs): an approach to intelligent systems research. *Journal of Experimental & Theoretical Artificial Intelligence*, 9:157–174(18), 1 April 1997.

Minoru Asada, Raffaello D'Andrea, Andreas Birk, Hiroaki Kitano, and Manuela Veloso. Robotics in Edutainment. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*. IEEE Presse, 2000.

Andreas Birk. Fast Robot Prototyping with the CubeSystem. In *Proceedings of the International Conference on Robotics and Automation, ICRA'2004*. IEEE Press, 2004.

Andreas Birk and Stefano Carpin. Rescue Robotics - a crucial milestone on the road to autonomous systems. *Advanced Robotics Journal*, 20(5), 2006.

Andreas Birk and Holger Kenn. An Industrial Application of Behavior-Oriented Robotics. In *Proceedings of the International Conference on Robotics and Automation, ICRA'2001*. IEEE Press, 2001.

Andreas Birk and Holger Kenn. A Control Architecture for a Rescue Robot ensuring Safe Semi-Autonomous Operation. In Gal Kaminka, Pedro U. Lima, and Raul Rojas, editors, *RoboCup-02: Robot Soccer World Cup VI*, volume 2752 of *LNAI*, pages 254–262. Springer, 2003.

Andreas Birk and Holger Kenn. RoboGuard, a Teleoperated Mobile Security Robot. *Control Engineering Practice*, 10(11):1259–1264, 2002.

Andreas Birk and Julie Wiernik. An N-Player Prisoner's Dilemma in a Robotic Ecosystem. In *8th International Symposium on Intelligent Robotic Systems, SIRS'00*. 2000.

Andreas Birk, Holger Kenn, and Thomas Walle. RoboCube: an "universal" "special-purpose" Hardware for the RoboCup small robots league. In *4th International Symposium on Distributed Autonomous Robotic Systems*. Springer, 1998.

Andreas Birk, Holger Kenn, and Luc Steels. Programming with Behavior Processes. *International Journal of Robotics and Autonomous Systems*, 39:115–127, 2002.

Andreas Birk, Stefan Markov, Ivan Delchev, and Kaustubh Pathak. Autonomous Rescue Operations on the IUB Rugbot. In *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR)*. IEEE Press, 2006.

H. Bruyninckx. Open robot control software: the OROCOS project. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 2523 –2528. IEEE Computer Society Press, 2001.

D.F. Coleman, J.B. Newman, and R.D. Ballard. Design and implementation of advanced underwater imaging systems for deep sea marine archaeological surveys. In *OCEANS 2000 MTS/IEEE Conference and Exhibition*, volume 1, pages 661–665 vol.1, 2000.

W. Craik. Monitoring in the Great Barrier Reef Marine Park. In *OCEANS*, volume 18, pages 785–790, 1986.

Holger Kenn. *CubeOS, The Manual*. Vrije Universiteit Brussel, AI-Laboratory, 2000.

Holger Kenn and Andreas Birk. From Games to Applications: Component reuse in Rescue Robots. In Daniele Nardi, Martin Riedmiller, and Claude Sammut, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence (LNAI)*, page p.669ff. Springer, 2005.

A. Mallet, S. Fleury, and H. Bruyninckx. A specification of generic robotics software components: future evolutions of GenoM in the Orocos context. In *International Conference on Intelligent Robotics and Systems*. IEEE, 2002.

E. Nebrija, C. Young, R. Meyer, and J. Moore. Electrical Prospecting Methods Applied to Shallow-Water Mineral Exploration. In *OCEANS*, volume 8, pages 56–66, 1976.

M. Uliana, F. Andreucci, and B. Papalia. The navigation system of an autonomous underwater vehicle for Antarctic exploration. In *OCEANS '97. MTS/IEEE Conference Proceedings*, volume 1, pages 403–408 vol.1, 1997.

T. Ura, T. Obara, S. Takagawa, and T. Gamo. Exploration of Teisi Knoll by autonomous underwater vehicle "R-One robot". In *OCEANS, 2001. MTS/IEEE Conference and Exhibition*, volume 1, pages 456–461 vol.1, 2001.

L. Yongkuan. AUV's trends over the world in the future decade. In *Autonomous Underwater Vehicle Technology, 1992. AUV '92., Proceedings of the 1992 Symposium on*, pages 116–127, 1992.

DOI: 10.3182/20070903-3-FR-2921.00016 (dysfunctional)

Albu, D., A. Birk, P. Dobrev, F. Gammoh, A. Giurgiu, S-C. Mihut, B. Minzu, R. Pascanu, S. Schwertfeger, A. Stan, et al., "Fast Prototyping of an Autonomous Underwater Vehicle (AUV) with the CubeSystem", 6th International Symposium on Intelligent Autonomous Vehicles (IAV 2007): IFAC, 2007.