Proceedings of the 2004 IEEE
International Conference on Robotics & Automation
New Orleans, LA • April 2004

An Architecture for Indoor Navigation

Wenfeng Li
Dept. of Logistic Engineering,
Wuhan University of Technology,
Wuhan 430063, China

Henrik I Christensen and Anders Orebäck
Centre for Autonomous Systems
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

Dingfang Chen
CAD Laboratory, Institute of Computing Technology,
Chinese Academia Sinica,
Beijing, 100080, China

***Abstract*—This paper is concerned with the design and implementation of a control architecture for a mobile robot that is to navigate in dynamic unknown indoor environments. It is based on the framework of Open Robot Control Software @ KTH, which is discussed and evaluated in this paper. As a hybrid architecture, it is decomposed into several basic components which can be classified as either deliberative or reactive. Each component can concurrently execute and communicate with another using unified communication interfaces. Scalability and portability and reusability are the goals of the design.**

Keywords-component; framework; architecture; indoor navigation

I. INTRODUCTION

A fundamental problem in mobile robotics is the design of a suitable architecture for integration of the various components in a system. The problem of architectures for mobile robotics is by no measure a new problem. It has been widely studied for more than 3 decades, and prominent architectures include NASREM [1], TCA [2], Behavior-Based [3], [4], Dervish [5], CLARAty [13] etc. A good overview of the wide variety of architectures available can be found in [6]. Today it is widely recognized that the most suitable architecture for a mobile robot is the hybrid deliberative approach [3], where deliberative and reactive systems are combined into a unified framework. Such an architecture allows rapid reaction to unexpected situations (the reactive layer) and at the same time goal directed behavior and use of prior knowledge, such as geometric maps, through the deliberative layer situated on top of the reactive layer. While there is an agreement on the control paradigm to be used in a mobile robot architecture, there is a lack of standard software packages for integration of systems. A few popular choices include Saphira [7], Mobility by iRobot, TeamBots [8], TCA [2], and Berra [9]. Unfortunately most of these systems have a number of problems, as for example reported in [10] in terms of:

- Portability to different hardware platforms
- Easy distribution for use of multiple computers
- Object oriented design
- Support for different control models

To address this problem an open source framework has been developed as part of the EU Open Robot Control Systems (OROCOS) initiative. Primary design objectives have been

- A set of standardized communication patterns
- Use of existing middleware technology (CORBA)
- Component based design to facilitate easy integration
- Support for multiple control paradigms
- Hardware abstraction.

One of the objectives of this paper is to design a hybrid architecture for a mobile robot to navigate in unknown indoor environments. At the same time the system is used to evaluate the characteristics of the framework of Open Robot Control Software (OROCOS) at KTH [11].

In this paper we start by introducing the OROCOS @ KTH framework and go on to describe an example system for indoor navigation, and finally we draw a number of conclusions as to the adequacy of the framework for flexible development of mobile robot systems and the features of the navigation architecture.

II. THE OROCOS FRAMEWORK

OROCOS@KTH is an application independent framework for soft-real-time mobile robots, as a part of the European project: Open Robot Control Software. The philosophy of the design is: 1) Component based. That is, details of low level control and communication, etc. are abstracted and encapsulated into modules. Developers only need to design application oriented components with these modules. 2) Code sharing. It is not bound to specific hardware in a specific lab. And the code is open under GPL. Users should be able to migrate to the system with limited effort. 3) Support for distributed computing. Components can concurrently run and communicate in a distributed and heterogeneous environment. Figure 1 is the overview of the toolkit.

The work has in part been sponsored by the EU OROCOS project and the Swedish Foundation for Strategic Research

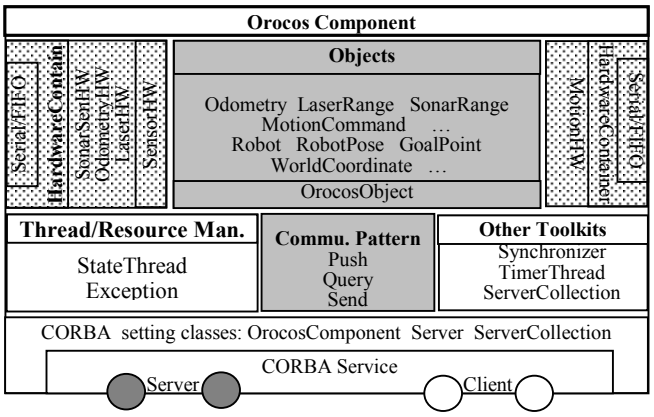

Fig. 1 Framework of OROCOS@KTH for hardware in-/dependent components. It is based on CORBA and can be used in distributed multi-agents. And specific hardware information is encapsulated in a runtime library.

- All components in an application are built on CORBA (the Common Object Request Broker Architecture) so that they can work in heterogeneous and distributed environments. The CORBA specific code is encapsulated in a set of base-class (i.e. OrocosComponent). All components inherit from this class.
- Its core includes communication patterns. Communication patterns are a set of common modes of data and event transfer utilizing CORBA. There are three patterns designed. 1) *Push*. It asynchronously sends an object to other components, using a one-way communication to push an object to other components that subscribe. 2) *Query*. It is a method of retrieving an object from a component (pull). 3) *Send*. It is a one-way communication to send an object to the server side.
- The objects transmitted via the communication patterns are composed from CORBA valuetypes. A valuetype, or an object, is a data structure which parameterizes the communication pattern templates. It should be predefined from the predefined base-class OrocosObject. A communication pattern, when bounded with a predefined valuetype object, becomes an interface, which can be a client or a server. Consequently, a component must inherit from communication patterns and define relevant valuetypes. With the support of CORBA broker, it decouples other components and becomes independent units.
- To be independent on hardware such as robotic actuators, sonar-sensors and laser-sensors, a set of classes is developed to provide hardware abstraction as shown in shaded areas in Fig. 1. Special hardware information can be compiled into a runtime library and activated at runtime.
- To coordinate communication and control resources, a group of tools have also been developed: StateThread (a basic state-machine for use in a component) and Synchronizer(used for synchronizing data from two or more components), etc.

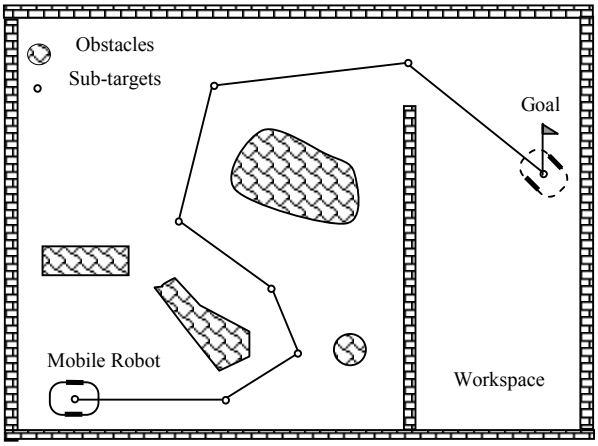

Fig. 2 Scenario of indoor navigation

Based on the framework of OROCOS@KTH, an application dependent architecture and its components are interoperable and portable across specific platforms and hardware.

OROCOS@KTH is currently using the ACE/TAO implementation of CORBA. It is programmed in C++. It utilizes the CORBA Naming Service for mapping names to object references but also has support for using the Trading Service. It has its own component model, but the plan is to use the CORBA Component Model in the future. Configuration, including the wiring of components, is made in XML files.

There are several other toolkits available for construction of robot systems. TCA/Xavier is an indoor navigation system built on TCA. It is a layered control architecture and the interprocess communication is C/Socket based [2, 14]. The Mobility system has been developed by iRobot Co. It is an object-oriented and CORBA-based control architecture to support code reuse, transportability and extensibility. But it is proprietary. Saphira is the standard software environment delivered with robots from ActiveMedia. It is C++/shared memory and parts of it are in open source. Its shared core, named Local Perceptual Space (LPS), allow for easy composition of "fuzzy" behavior coordination, but the shared memory might be a bottle neck for communication [7]. TeamBots is a pure Java-based open source toolkit for multi-agent and mobile research.

III. ARCHITECTURE OF THE APPLICATION

A. Scenario

Navigation in dynamic and unknown surroundings is a widely studied problem. One traditional scenario is "go from place A to place B without bumping into an obstacle".

One way to avoid obstacles is to move the robot around the obstacles with keeping track of them. To do this its sensory system must estimate the tangent direction of the obstacle. This is often not realistic. Another question is the planning of trajectories for realistic robots in real environments. In [12], an algorithm of free space detection based on a range scanner, is

presented to provide an experimental approach to online sensor-based navigation in unknown indoor environment. In order to detect free space, the area in the front of the robot is divided into a number of regions, along both the radial and angular directions of the range sensor. The robot can move from one free block (point) to another adjacent block. Hence, the robot in the paper keeps off obstacles with constant interval or moves through free space to the target point, rather than moving around obstacles. This algorithm is simple and efficient, with robot moving from point to point, and has reachability to the goal. But information is not fully utilized, the trajectory might be coarse and the robot might perform excessive turns.

In the present scenario, as in Fig. 2, the navigation might be composed of tasks: point-to-point navigation, door detection, door traversal and obstacle handling. Using these tasks it is possible to perform most any-place to any-place navigation missions in an indoor environment. The detection of places is achieved using a laser scanner, while obstacles might be detected by sonar or laser. The entire system is tested on a Pioneer-II system, but the same system can also be executed on a Nomadic Super Scout.

B. General strategy

According to above scenario, the robot should have a capability to deliberatively reason and detect a "*door*" with the perceptual input from sensors. Meanwhile, it should reactively steer clear of obstacles on the way to sub-goals. Therefore, it is implemented as a hybrid architecture. The general layout of this simple evalution system is shown in Fig. 3.

C. Specific design

As shown in Fig. 4, this application is decomposed into 10 components with the support of OROCOS@KTH. The components can be classified into four types: 1) Deliberative components: *Userinterface* and *Pathplanning*. 2) Monitor component: *Obstacledetecting*. 3) Reactive components: *Orienting*, *Obstaleavoiding* and *Goto*. 4) Hardware abstractions: *RobotBaseServer* and *LaserServer*. Each component and its communication are briefly discussed in the following section.

IV. COMPONENTS

The components that compose this application system are shown in Fig. 4. The communication patterns are shown as directed arrows. For example, Component "*Userinterface*" sends *GoalPoint* object to component "*Pathplanning*" and queries its state with object *LogMessage*.

A. Userinterface

Through this component the user can enter missions which through the *Pathplanning* are converted into *Goalpoints*. The status of the *Pathplanning* can at any time be requested through the use of the Query pattern, in which case a *LogMessage* object is returned.

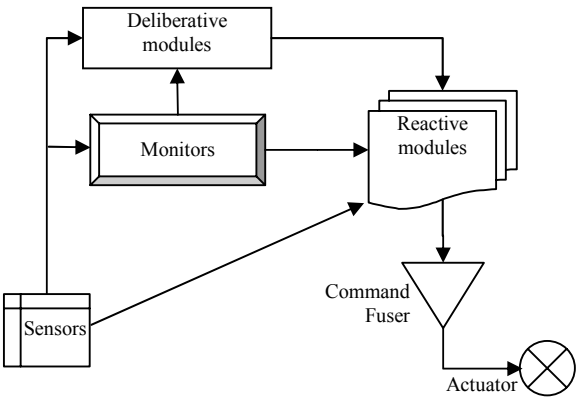

Fig. 3 A hybrid architecture based on components and hardware abstract

B. Path planning

Path planning is one of the important functions to realize indoor navigation. Here, it is the commander and coordinator of a mission. When it receives the coordinates of a target (*Goalpoint* object) from the *Userinterface* component, it actuates Orienting component by sending the object of a *WorldCoord*. It also actuates the component of obstacle detection by subscribing to data *ObstRead*. In parallel it actuates the component of *DLEdetecting* by subscribing to *DLEdata* if necessary to get a sub-target. Finally, it sends a *WorldCoord* object to the component *Goto*.

C. DLEdetecting

This component is designed for door-like-exit detection and reasoning. A door-like-exit is an exit, which is characteristically like a door in 2D:

- This is a local opening with a width of at least 800mm.
- The free space beyond the opening must be at least 1000mm.

This component employs a specific Door-like-exit detection algorithm to detect sub-targets from the data of online laser sensor. The raw data used in this component are subscribed as the object of *LaserRead* from the component *LaserServer*. These sub-targets are abstracted into *DLEdata* and communicated to *Pathplanning* with Push.

D. Obstacledetecting

This component is a monitor to detect if there is any obstacle on the way. It also informs the *Goto* component if there is a condition that drives the robot in a dead end. It communicates with *Pathplanning* and *Goto* using a Push method and the objects are of type *ObstRead*. It subscribes to the readings from the sonar ring (object of *SonarRead*) from the component *RobotBaseServer*.

E. Goto

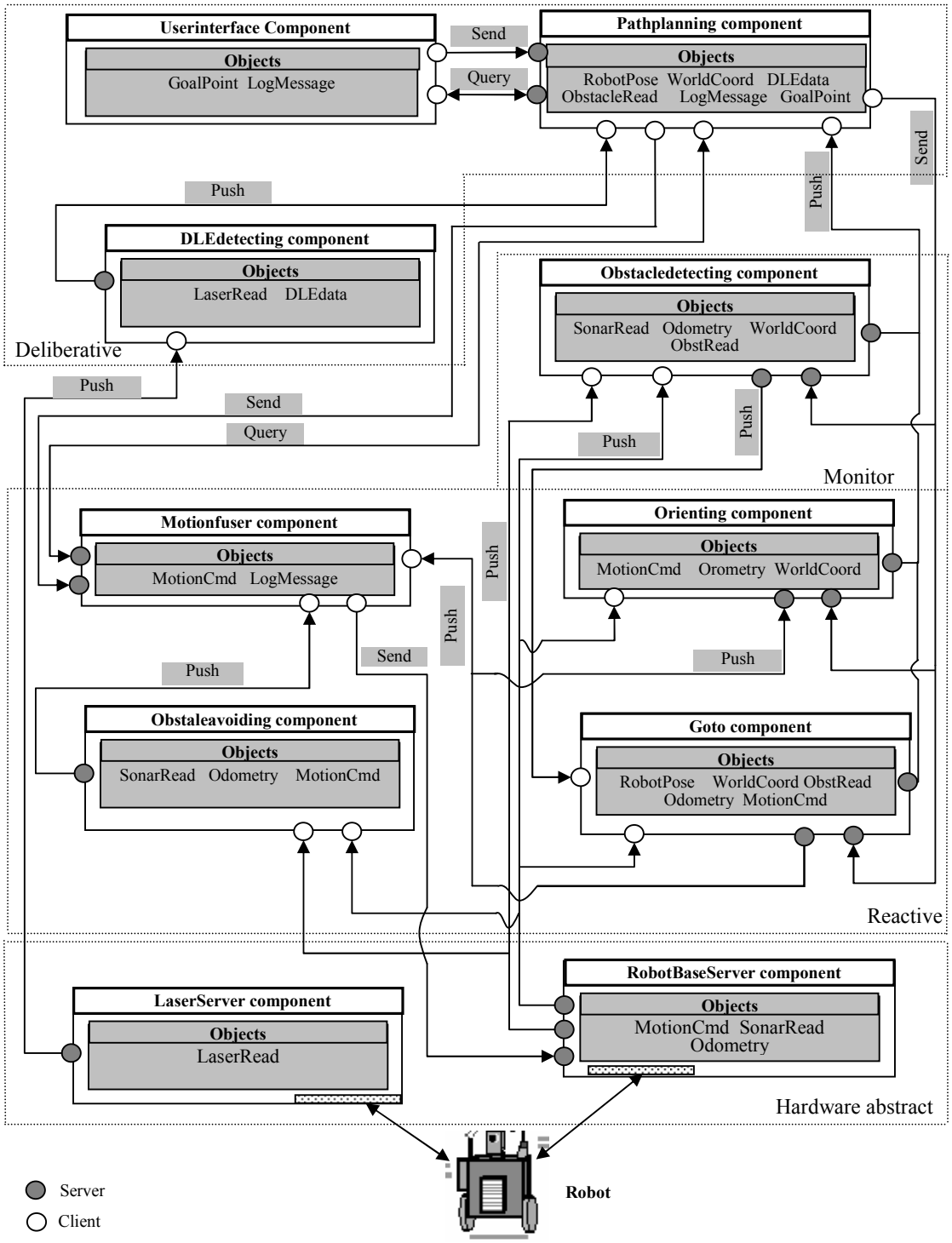

Fig. 4 Architecture of the application and the communication among the components

A *Goto* function is a necessary part for navigation. It generates motion commands and sends them to *Motionfuser* with Push and object of *MotionCmd*. It decides reachability and motion gaits. To be convergent, there always is an attractive force to a sub-/target. To guarantee smooth movement, it keeps moving in the same direction under the monitor of *Obstacledetecting*. It subscribes to the readings from the sensor *Odometry* (Object of *Odometry*) from the component *RobotBaseServer* to detect if the goal is reached.

F. Orienting

This is another generator of motion commands, rotating the robot to a certain degree. The purpose of this component is to simplify the structure and implementation of the component *Goto*. In addition, it also enhances the modular level of the architecture and the reusability of the components *Goto*, *Obstacleavoiding* and *Motionfuser*. Here, it uses the Push pattern to send turn commands (object of *MotionCmd*) to *Motionfuser*. It subscribes to readings from sensor *Odometry* (Object of *Odometry*) from the component *RobotBaseServer* to detect if the goal has been reached.

G. Obstacleavoiding

This component generates a behavior of stay clear of obstacles on the way. It subscribes to *SonarRead* from *RobotBaseServer* to detect if there is an obstacle in front of the robot. Given an obstacle, for example, in front of one of the left sonar sensor of the robot, it generates a motion command to move away from the obstacle along the direction of the sonar sensor. Each command has an associated weight, which is correlated with the distance and the direction of the obstacle. All commands are fused as a weighted vector sum and the result is sent as an object of type *MotionCmd* to *Motionfuser* using a Push.

H. Motionfuser

This component is the only one to send motion commands to the component *RobotBaseServer*, which controls the motor of the robot finally. It also subscribes to and receives all motion commands generated from other components such as *Goto*/ *Orienting* and *Obstacleavoiding*, and fuses them into one final command. Experiments show that it is the busiest component in this application and could be a bottleneck of the system. This is partially because all control commands have to flow together and be fused into one to control the actuator. The more the components which generate control commands, the busier this component. Another reason is this component has to synchronize all commands. This synchronization greatly increases the working time of this component. Therefore, its running state must be monitored so that further navigation can be implemented only when it is idle and accessible. This monitoring function is implemented with the Query pattern and *LogMessage* object sent to *Pathplanning*.

I. RobotBaseServer and LaserServer

LaserServer is a component abstraction on top of the laser sensor. *RobotBaseServer* encapsulates the interfaces to the actuator, the odometry and the sonar ring. All components can only communicate with hardware through these two components.

V. EVALUATION

The system was evaluated using experiments on Nomadic robot simulator, a Nomadic Scout - Louie and a PeopleBot - Goofy (Fig. 5). Both robots have a SICK scanner, which has a 180 degree field of view, with 361 steps of resolution, an odometry sensor and a set of sonar sensors. The systems both use differential drive configuration. The operation system is Linux RH7.3, with TAO1.3 and ACE5.3. OROCOS@KTH has also been installed.

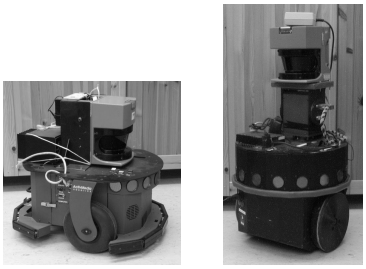

Fig. 5 Robot Louie (Left) and Robot Goofy (right)

The experimental scenario is the living room at the Centre for Autonomous Systems (CAS). The tested robots are commanded to reach the goal under three cases: in the same room, in the corridor, and in another room within the corridor. The general scenario is shown in Fig. 2. Below a number of issues related to use of the systems is outlined.

A. Portability

The system was used on three different platforms with little modification. For porting to new platform the hardware abstraction modules must be rewritten for the new system.

B. Behaviors and efficiency

Because of the introduction of door-like-exits, Robots can move almost straight to sub-targets in static environments with small path perturbations. They can move in high speed and their trajectories are balanced and smooth. The time that component *DLEdetecting* spends door-like-exits is only averagely 0.040s per detection. The small system does not offer enough complexity to challenge the system in terms of efficiency. At present the access to sensor data defines the overall speed of the system.

C. Coding

As all components are built on the toolkit OROCOS@KTH, their programming is rather easy. Due to the extensive set of functionalities available in the OROCOS library, the coding of each component is compact. The resulting footprint is however rather large. The binary of a component is typically 1.5MB on disk. For example, the component Pathplanning has a total of 551 lines of C++ code, and the binary is 1.59MB. the MotionFuser is 334 lines and 1.54MB. A big problem, as in all multi-process application programming, is debugging, which can be extremely difficult. Another issue is the learning curve. To write components one needs to know c++, CORBA, … so ther is an investment to get started.

D. Communication and transparency

All communication is performed with the three communication patterns. This is an efficient and natural model of component interaction. The experiments also demonstrated that the implementation is robust. There is no failure caused by communications themselves. Access to components/objects

through the CORBA Name Service [15] is efficient and provides for transparency in the system.

E. Resource management and monitors

For this particular application design, the only point of resource synchronization was the *Motionfuser*, which needs to be monitored. This monitor is implemented with Query pattern and *LogMessage* object. However, query is not very efficient method for monitoring the working state of a resource. Another issue is the capability of synchronizing components, which might cause system to clog at component Motionfuser. For implementation of more advanced systems it might be useful to consider introduction of efficient mechanisms for inter-process synchronization.

VI. SUMMARY

A hybrid architecture for mobile robots to navigate in dynamic unknown indoor environments was presented in this paper. The main focus of the paper was not the system behavior, but the evaluation of methods for system design. Consequently, the system uses 1) incrementally design using component technology. It is decomposed into components with the support of the common and open framework of OROCOS@KTH. Each component has a unified communication interface and can be concurrently executed. Hence, it is scalable, portable and reusable. 2) a hybrid architecture. It take advantage of both deliberative and reactive modules. 3) It utilizes an application specific component for detection of door-like-exit. This algorithm is encapsulated into component *DLEdetecting* and utilizes 2D features of a door in combination with the range data from laser sensor. Cooperating with the *Goto* function and *Obstacleavoiding*, the algorithm and the corresponding *path-planning* allow robots to move robustly, efficiently and smoothly.

One of main objectives of the work has been to evaluate OROCOS@KTH. It examines the characteristics of the toolkit in terms of construction of scalable, portable, and easy to develop autonomous mobile architecture. A particular design is presented. Through use of several platform it is demonstrated that a level of portability has been achieved. In addition the system has demonstrated adequate efficiency for navigation and obstacle avoidance. The experience in term of programming is that the toolkit allows for effacing coding, though the learning curve might be challenging. In addition the footprint is relatively large which primarily is due to the use of the ACE/CORBA. Finally the handling of critical resource is not as detailed as one might want in real-time control systems.

ACKNOWLEDGMENT

This work has in part been sponsored by the EU OROCOS project, the China Scholar Council and the Swedish Foundation for the Strategic Research through its Centre for Autonomous Systems. The funding is gratefully acknowledged. Meanwhile, the experiments were carried out at the Center of Autonomous Systems (CAS), Royal Institutes of Technology, Sweden. The authors would also like to thank the anonymous reviewers for the useful insights and suggestions.

REFERENCES

[1] J. Albus, H. McCain, and R. Lumia, "Nasa/nbs standard reference model for telerobot control system architecture (nasrem)," Tech. Rep. NBS Technical Note 1235, Robot Systems Division, National Bureau of Standards, Gathersburgh, VA, 1987.

[2] R. G. Simmons, "Structured control for autonomous robots," IEEE Transactions on Robotics and Automation, vol. 10, no. 1, pp. 34–43, 1994.

[3] R. C. Arkin, "Integrating behavioral, perceptual, and world knowledge in reactive navigation," Robotics and Autonomous Systems, vol. 6, pp. 105–122, 1990.

[4] R. A. Brooks, "A robust layered control system for a mobile robot," IEEE Journal of Robotics and Automation, vol. RA - 2, pp. 14 – 23, March 1986.

[5] I. Nourbakhsh, R. Powers, and S. Birchfield, "Dervish: An office navigation robot," AI Magazine, vol. 16, no. 2, pp. 53–60, 1995.

[6] R. C. Arkin, Behaviour Based Robotics. Intelligent Robots and Autonomous Agents, Boston, MA: MIT Press, 1998. ISBN 0-262-01165-4.

[7] K. Konolige and K. Myers, "The saphira architecture for autonomous mobile robots," in Artificial Intelligence and Mobile Robots (P. B. D. Kortenkamp and R. Murphy, eds.), ch. 9, pp. 211–242, Menlo Park, CA: AAAI Press/The MIT Press, 1998. ISBN: 0-262-611376.

[8] T. Balch, Behavioral Diversity in Learning Robot Teams. PhD thesis, College of Computing, Georgia Tech, Atlanta, Ga. – USA, December 1998.

[9] M. Lindström, A. Orebäck, and H. Christensen, "Berra: A research architecture for service robots," in Intl. Conf. on Robotics and Automation (Khatib, ed.), vol. 4, (San Francisco), pp. 3278–3283, IEEE, May 2000.

[10] A. Orebäck and H. I. Christensen, "Evaluation of architectures for mobile robotics," Autonomous Robots, vol. 14, pp. 33–50, Jan 2003.

[11] OROCOS@kth. http://cogvis.nada.kth.se/orocos/

[12] T. Tsubouchi, T. Yamaguchi, S. Yuta, "Online sensor-based behaviour decision and navigation of a mobile robot in unknown indoor environment," the 8th International Symposium on Experimental Robotics (ISER '02), Sant'Angelo d'Ischia, Italy, July 2002.

[13] I. Nesnas, R. Volpe, T. Estin, etc., "Toward Developing Reusable Software Components for Robotic Applications," Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Maui Hawaii, Oct. 2001.

[14] R. Simmons, J. Fernandez, R. Goodwin, S. Koenig, and J. O'Sullivan, "Xavier: An Autonomous Mobile Robot on the Web." Beyond Webcams: An Introduction to Online Robots, K. Goldberg and R. Siegwart (Eds.), MIT Press, 2002

[15] http://www.cs.wustl.edu/~schmidt/TAO.html