Miro—Middleware for Mobile Robot Applications

Hans Utz, Stefan Sablatnög, Stefan Enderle, and Gerhard Kraetzschmar

***Abstract*—Developing software for mobile robot applications is a tedious and error-prone task. Modern mobile robot systems are distributed systems, and their designs exhibit large heterogeneity in terms of hardware, operating systems, communications protocols, and programming languages. Vendor-provided programming environments have not kept pace with recent developments in software technology. Also, standardized modules for certain robot functionalities are beginning to emerge. Furthermore, the seamless integration of mobile robot applications into enterprise information processing systems is mostly an open problem. We suggest the construction and use of object-oriented robot middleware to make the development of mobile robot applications easier and faster, and to foster portability and maintainability of robot software. With Miro, we present such a middleware, which meets the aforementioned requirements and has been ported to three different mobile platforms with little effort. Miro also provides generic abstract services like localization or behavior engines, which can be applied on different robot platforms with virtually no modifications.**

***Index Terms*—CORBA, distributed systems, middleware, mobile robots, multirobot applications, object orientation, robot control architectures.**

I. Introduction

Developing software for mobile robot applications is a tedious and error-prone task. Modern mobile robots are usually composed of heterogeneous hardware components, which are connected using different networking technologies and communication protocols with widely differing bandwidths. A large number of different methods for processing sensor information and controlling actuators, for performing computational vision and cognitive tasks like planning, navigation, and user interaction, must be integrated into a well-engineered piece of software. All these issues contribute to the enormous complexity of the mobile robot software development task. Vendor-provided programming environments have not kept pace with recent developments in software technology. Mobile robot software are often custom made, closed systems, which makes it difficult to integrate them in enterprise information processing frameworks. Furthermore, standardized modules for certain robot functionalities are beginning to emerge.

We suggest the construction and use of object-oriented robot middleware to make the development of mobile robot applications easier and faster and to foster portability and maintainability of robot software. High-quality robot software and an improved software development process will be key factors in enhancing both the research state of the art as well as the likelihood of deploying working applications in industrial and consumer markets.

Manuscript received December 11, 2001. This paper was recommended for publication by Associate Editor G. Menga and Editor N. Viswanadham upon evaluation of the reviewers' comments. This paper was presented at Telematics Applications and automation in Robotics (IFAC), Weingarten, Germany, 2001, and at the RoboCup Symposium, Seattle, WA, 2001.

H. Utz and G. Kraetzschmar are with the University of Ulm, Neuroinformatics, 89069 Ulm, Germany (e-mail: hutz@neuro.informatik.uni-ulm.de; gkk@neuro.informatik.uni-ulm.de).

S. Sablatnög is with Temic Sprachverarbeitung GmbH, 89077 Ulm, Germany (e-mail: ssablatn@neuro.informatik.uni-ulm.de).

S. Enderle is with **<AUTHOR: AFFILIATION? ADDRESS?>** (e-mail: steve@neuro.informatik.uni-ulm.de).

Publisher Item Identifier 10.1109/TRA.2002.802930.

With Miro, we present such a middleware, which meets the aforementioned requirements and has been been ported to three different mobile platforms with little effort. Miro also provides generic abstract services like engines for mapping, localization, and behavior generation, which can be applied on different robot platforms with virtually no modifications.

II. Design Goals for Robotics Middleware

In mainstream software development projects, middleware systems are mainly applied in order to reduce development time and cost. This is achieved by providing well-structured frameworks for often-needed data structures, precoded and well-tested code for particular functionalities, and standardized communication protocols and synchronization mechanisms. Using these facilities leads to improved software structures, less error-prone implementations, and better overall software quality. By writing code based on middleware, it is easier for programmers to meet given standards and more likely for the code to be reused later on. Furthermore, middleware that is available on a wide range of hardware platforms and operating systems makes it easier to scale up systems, to port them to another platform, and to maintain large applications. If we would like to exploit all these advantages in mobile robotics (and, maybe, factory automation), what design goals and requirements does that imply for robotics middleware?

Object-Oriented Design: The middleware should be designed using the object-oriented paradigm thoroughly. Object-oriented concepts like information hiding, name spaces to prevent naming ambiguities when using multiple mutually independent libraries, exception handling to separate normal program control flow from error recovery, abstraction, type polymorphism, and inheritance can all significantly contribute to improve design and implementation of mobile robot software when applied deliberately and consistently.

Open Architecture Approach: Considering the immense spectrum of different hardware and software components used in mobile robot systems, adopting an open architecture approach, including availability of all source code, seems indispensable. Only if applications programmers can access, modify, and fine tune all components of the software environment, the integration of a heterogeneous set of hardware and software components can be successful. Also, integration of new hardware developments as well as new computational methods developed by ongoing research is usually much easier and faster, if all source code is available for inspection, reuse, and debugging purposes.

Hardware and Operating System Abstraction: The object-oriented paradigm permits a clean abstraction of sensor and actuator subsystems, as well as low-level system services like communications, and provides a uniform interface design. A suitable level of abstraction from low-level system details, both in terms of hardware as well as software, may be a key element in attaining a better understanding of how to integrate a significant number of different hardware devices and computational methods.

Multiplatform Support and Interoperability: The middleware should be available on a wide range of hardware platforms and common operating systems. The use of different programming languages for implementing different modules should be supported without requiring much extra overhead cost for integration.

Communication Support and Interoperability: The middleware should provide a carefully designed set of interfaces for communication between objects and communications transparency, i.e., the programmer should not need to worry much about where objects are actually allocated and which communications protocol s/he needs to apply. Single robot and multirobot applications should be equally well

supported, and integration into enterprise software frameworks should be smooth.

Client/Server Systems Design: We suggest adopting a client/server view, at least for larger modules of a robot control architecture. The modules, implemented as objects, provide services to other modules (objects) through a set of interfaces. Most modules of an application fill both the client and the server role, albeit to different subsystems, using the services of sensor objects as clients and providing services to actuator objects as servers.

Flexible Integration of User Interfaces: A commonly underestimated problem in mobile robot research is the development and use of suitable user interfaces for various tasks. The task-oriented user interface for end users interacting with the robot requires careful design in order to avoid compromising the robot's autonomy and integrity. People operating and supervising the robots need interfaces for monitoring its operational state, while programmers may need even further internal state information for debugging purposes. These user interfaces might be activated only occasionally or under special circumstances. Nevertheless, their effect on the runtime performance should be limited, and the middleware should support the programmer in meeting this goal.

Software Design Patterns: The middleware should encourage the use of design patterns by providing a class framework for common, well-understood functionalities. Examples from the robotics domain are patterns for mapping, self-localization, behavior generation, and path planning.

III. Miro—Middleware for Mobile Robots

The desire for robotics middleware in our group grew after developing software for five different mobile robot platforms, with different operating systems, different programming languages, different development environments, and with a number of collaborators, for several years. We realized that for ensuring further progress in mobile robot research, it was time to blend the different existing software architectures into a generalized framework of middleware for autonomous mobile robots. With the Miro software framework, we consequently tried to address the design challenges discussed above, and to lay the foundation for implementing robust and reliable robot control architectures.

Miro is designed and implemented by rigorously applying object-oriented design and implementation techniques. By adhering to the common object request broker architecture (CORBA) standard, interprocess and cross-platform interoperability for distributed robot control architectures are achieved. All core Miro functionality, like routines for processing sensor data and control actuators, is entirely implemented in C++ and allows for high runtime efficiency. Miro builds itself upon widely used, industrial-strength middleware packages, which are open source and available on a wide range of hardware and operating system platforms.

A. Miro Architecture

Miro is structured into three architectural layers, which are interwoven with two major layers of the CORBA middleware underlying the Miro design (see Fig. 1).

1) The *Miro Device Layer* provides object-oriented interface abstractions for all sensory and actuatory facilities of a robot. This is the platform-dependent part of Miro.
2) The *Miro Service Layer* provides active service abstractions for sensors and actuators via CORBA interface definition language (IDL) descriptions and implements these services as network-transparent objects in a platform-independent manner. The programmer uses standard CORBA object protocols to interface to any device, either on a local or a remote robot. Also, event-based communication services based on the CORBA notification services are available.
3) The *Miro Class Framework* provides a number of often-used functional modules for mobile robot control, like modules for mapping, self localization, behavior generation, path planning, logging, and visualization facilities.

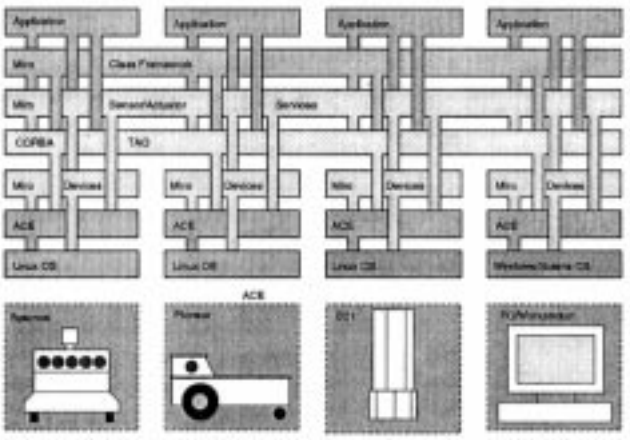

Fig. 1. Miro architecture. The ACE and Miro device layers provide abstract interfaces for specific platforms. The TAO CORBA and Miro service layers achieve platform-independence, network, and location transparency. The Miro class framework provides classes for generic robot control functionality.

Miro Device Layer: It provides classes that wrap the message or package-based communications links to the low-level controller boards (serial link, can bus, etc.) into ordinary method calls, for invocation by the service implementations.

Miro Service Layer: Sensors and actuators can be naturally modeled as objects, which can be controlled and queried by their respective methods. Thereby, mobile robot software can be viewed as aggregations of sensory, actuatory, and cognitive objects, which can trade information and services in an agent-like manner.

In the Miro Service Layer, object-oriented interfaces for all hardware components are provided. Wrapper classes for all sensors and actuators hide away all the peculiarities of sensor/actuator-specific interface functions. The class hierarchy in this layer specifies useful abstractions and generalizations for certain subsets of sensors and actuators. An example is a `RangeSensor` class, defining interface functionality common to sensors like infrareds, sonars, and laser range finders. Similar abstractions are available for the behavioral aspect. Classes like `SynchroMotion` or `DifferentialMotion` define generalized interfaces for the respective kind of kinematics and inherit this functionality to their specializations, e.g., `SparrowMotion`, `PioneerMotion`, or `B21Motion`, as illustrated in Fig. 2.

To overcome location and programming language dependencies, all sensor and actuator services export their interfaces as network transparent CORBA objects, which can be addressed from any language and platform for which language bindings and CORBA implementations exist. This enables seamless integration of high-level robot control subsystems, like Lisp-based planners or Java-based user interfaces.

While a traditional-method call is the most convenient way of object interaction, the classical synchronous query interface does not scale well in large mobile robot applications. This is especially true for sensory devices. Sensors usually produce measurements either on demand (bumper) or at some fixed or maximum rate (LRF, IR)<**AUTHOR: PLEASE DEFINE "LRF" AND "IR"**>. In order to avoid missing a sensor reading, the consumer is required to either continuously poll the sensor or to wait for the next reading. For example, in a well-behaved

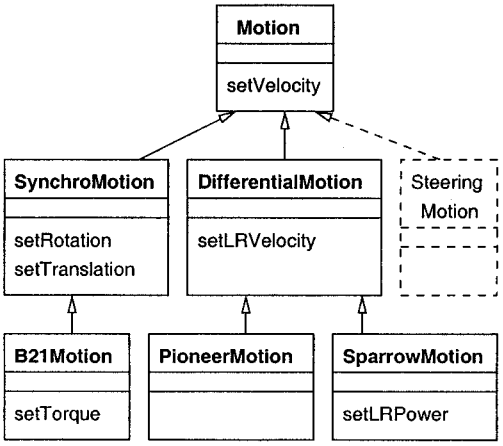

Fig. 2. Motion-interface inheritance for different mobile bases.

mobile robot, a bump sensor should hardly ever be pressed. Nevertheless, for safety reasons, no bump situation should be missed. What we are interested in is not the actual sensor reading, but a particular event, like the bumper being pressed. Therefore, the Miro Service Layer provides an abstraction of sensors and actuators as active services with event-based communication.

Basic support for multirobot control comes naturally with a distributed robot software development environment as provided by the Miro Services Layer. Small groups of robots can address each other by exchanging the object references if their respective sensor/actuator configurations are known. This approach is facilitated by the availability of naming service functionality, providing a separate namespace for each robot. Also sharing of sensor data can be achieved, via filtered event-processing frameworks based upon the notification service. We are currently evaluating such a framework in our Sparrows robot team.

For larger multirobot systems, additional middleware support for robot interaction has to be provided, for example, in order to enable spontaneous cooperation between robots that initially do not know of each other. Frameworks for the flexible description of system properties of an individual robot are not yet part of Miro, but will be subject to further research.

Miro Class Framework. An essential step toward easier and faster development of mobile robot software is reuse of code for commonly used tasks. As mobile robots research has seen a constant flow of new methods and research results in the past, this has seemed to make little sense so far. However, a number of recently developed methods and techniques seem to emerge as *de facto* standard solutions, e.g., grid-based probabilistic egocentric and allocentric mapping, self localization based on segment matching [1] or particle filters (a.k.a. Monte Carlo localization, [2]), behavior-based robot control [3], [4], or various path planning methods. The Miro Class Framework provides classes that implement commonly used techniques for mobile robot control, such that they can be applied uniformly on the different platforms supported. Available functionality includes a behavior engine, which permits dynamic activation, enabling and disabling of sets of behaviors and arbitrators, and sample-based pose estimation based on particle filters. Providing generic mapping and path-planning functionality is a work in progress.

Aside from functionality used for applications, the Miro Class Framework includes functionality useful for development and experimental evaluation. A good example is Miro's generic logging facility for experimental data acquisition. It exploits the event-triggered communications model, much like a specialized CORBA Telecom Log Service [5]. Its main focus is on efficiency and the ability to

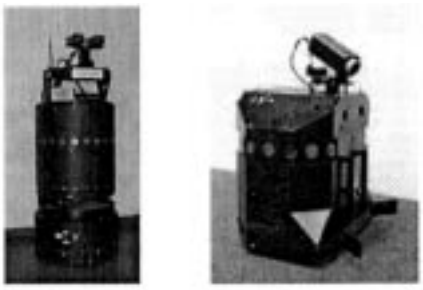

Fig. 3. B21 and Sparrow-99 robots.

replay logged data streams. It logs the notification messages using the methods provided by CORBA for the marshalling of data for remote method calls. That way, every data package that can be passed via the event channel can be logged for later use. A log player reads the logged data from a file and sends it back through the event channel in a timely manner. The receiver cannot distinguish between logged data and data generated in real time by sensors and actuators. The visualization tool for the RoboCup soccer team exploits this Miro feature, and can visualize both live data streams during games as well as game logs during postgame analysis without any modifications. The only difference is that the logplayer permits slow motion. Likewise, it is easily possible to perform offline testing, evaluation, or learning by intercepting and logging an arbitrary online event stream and feeding the logged data back to the module under evaluation or learning offline.

B. Implementation of Miro

Miro makes extensive use of multiplatform libraries for easy portability. The adaptive communications environment (ACE) provides object-oriented abstraction layers for many operating systems and communications primitives. The TAO **<AUTHOR: PLEASE DEFINE "TAO">** package is an implementation of the CORBA based on ACE [6]. The Miro implementation exploits the CORBA Notification Service [7] for providing the event-based communication functionality. For the construction of graphical user interfaces (GUIs), like monitoring and visualization tools, we use Qt, a GUI application framework, which is available for various Unix derivates including Linux, as well as Windows.

Presently, Miro is implemented for three mobile robot platforms, which are equipped with different sensors, actuators, and computational equipment. They are used in different scenarios, ranging from an office delivery to highly dynamic soccer games [8]. These platforms are as follows.

1) A B21 robot (see Fig. 3, left picture), which is equipped with bumpers, IRs, sonars, a laser range finder, and a vision system. It features a synchro drive mobile base and is controlled by two onboard PCs.
2) Pioneer-1 robots, which are differential-drive robots equipped with sonar sensors only, and are controlled either via a laptop mounted on top of a robot or by a host PC via a serial radio link.
3) Sparrow-99 robots [9], which are custom-built robots developed in our lab at the University of Ulm (see Fig. 3, right picture). Sensors include sonars, IRs, and a camera. The mobile base is a differential drive system. The robot also has a pan-tilt unit and, for its special purpose, a kicker. It is controlled via an onboard embedded PC.

The Miro implementation includes a number of clients, which are provided for testing and evaluating the Miro service functionalities. For each service, the following clients are provided.

1) *Sample client.* This client is held as simple as possible in order to show the user how the respective service functionality is used.
2) *Test client.* This client is used for debugging. It calls each service method provided and checks the results.
3) *Monitoring client.* This client implements a visual interface to one or more services. It is useful during normal robot execution for monitoring data.

IV. Experiences and Evaluation of Miro

Miro is successfully used in several projects striving for autonomous mobile robot control, where the robots are required to perform tasks like neurosymbolic mapping of indoor environments [10], hybrid multirepresentation world modeling [11], [12], autonomous self localization based on the Monte Carlo Localization method (two variants, one based on distance sensor data and one based on visual features [13]), hierarchical path planning [14], and reactive execution. Below, we summarize our experiences and discuss how well (or not) Miro achieves the design goals, how it compares to other mobile robot software frameworks, and provide some performance results.

Achievement of Design Goals Some of the design goals formulated in Section II are obtained for free by using CORBA as communications middleware. Type safe and network-transparent interfaces come with the strongly typed IDL of CORBA. Programming language independence is ensured by the CORBA standard, and therefore a solved problem in the context of Miro development. We have validated this by developing a Java-based browser frontend for teleoperating our B21 robot over the Internet.

The consistent and generalized service interfaces allow for multirobot platform development. This is demonstrated by the sample `wander`, `avoid`, and `wallfollow` behaviors included in the examples collection of the Miro source tree. The abstract `RangeSensor` interface permits for a generic avoid behavior, based on the selected sensor(s) for all three platforms. The programmer may want to tune the safety distance and maximum speed due to the different sensor characteristics.

Operating system independence is currently not an active topic within Miro development, since all our robots and work stations run under Linux. However, as a proof of concept, we ported the client side to Solaris within one day.

Related Work But how does Miro perform in comparison to other robot control architectures? Currently, there are mostly manufacturer-provided packages available.

Saphira [15], [16] is the software development environment delivered with the Pioneer mobile robot family. Its core is a C library for accessing the controller board. By using the Saphira library, the programmer implicitly imports elements of a particular robot-control architecture, including mechanisms like state reflection (a kind of implicit communication between client programs and server), data structures like a local perceptual space (LPS) and a global perceptual space (GPS), a fuzzy control-based behavior specification language, and Colbert, a language for reactive control [16]. Saphira lacks location transparency, and when integrating hardware (vision systems, manipulators) from other manufacturers, the programmer must carefully synchronize the communication of client programs with add-on hardware with the basic Saphira control loop.

Mobility is a distributed, object-oriented software development framework for the B21r/B14r family of robots by RWI, Jaffrey, NH [17]. It scales well with respect to user interaction, client libraries for visualization, and remote access of sensory information. Client libraries are available in C++ as well as in Java. On the other hand, Mobility lacks conceptual support for robot control. As a manufacturer-provided package, it only is available for their mobile platforms. Since even our old B21 robot is not supported by Mobility, it could only be evaluated by investigating its documentation and header files.

TABLE I
Results of Basic Response Test

response time in μ-sec	min	mean	max	var
raw Response	4789	12519	20722	1878
	14613	20807	27164	4543
poll Response	5459	13476	21257	6208
	10176	15660	22634	5321
notify Response	8372	14115	21661	5060
	8511	15091	21917	6350

Miro provides network-transparent interfaces, and also offers a framework for behavior-based robot control. However, it does not enforce the usage of a singular arbitration paradigm, and therefore does not provide higher-level idioms for robot control but concentrates on the flexibility and configurability of the control infrastructure.

In [18], Alami *et al.* document the task of integrating a demonstration from scratch on a newly obtained mobile platform within 40 days (not including porting their tools from Solaris to Linux). They utilize their LAAS **<AUTHOR: PLEASE DEFINE "LAAS">** architecture [19] for an office-navigation scenario. Even though most of the tools' functionalities are located on a higher control level as currently provided within Miro, this work documents nicely the potential of generalization and code reuse within the mobile-robotics domain.

Experiments: A question often raised within the robotics domain is the overhead introduced by features such as location transparency and event triggered communication. High performance and real-time conformance are not really an issue for many commercially available mobile robot platforms, due to the limited capabilities of their hardware. The Pioneer-1 controller board, as well as the B21 (pre-rFlex) motor controller are both attached to a PC via a slow serial link (9600 resp. 38 400 Baud). Furthermore, they can report their status to the PC only in 100–ms intervals. The odometry resolution is in each case about 1 cm. Therefore, a basic response test, the time between issuing a motor command and the reflection of the robots movement within an odometry reading, is heavily dominated by the latencies of the low-level controller and can hardly reflect the performance of the actual software architectures. Nevertheless, with ever-growing hardware capabilities, data throughput and predictable response times will become an important issue also in the field of autonomous mobile robots.

Of the robots in our lab, the Sparrow-99 controller board seemed most suited for performance measurements. It is attached to the PC via a 1 MHz CAN-Bus and, with currently used firmware, capable of reporting odometry updates at 100 Hz.

The basic response test was designed as follows. A motion command was issued on a still standing robot and the time was measured till the first odometry reading was received, which indicated that the robot had moved. Afterward the robot was halted for about one second, before the next iteration of the test. We used three implementations of a basic response test. The first was calling the Miro device layer directly. The second was invoking the methods via the CORBA method interface, actively waiting for the next odometry measurement. The third one was setting the velocity via a CORBA method call and evaluating the odometry messages pushed to it via the event-triggered communications interface. The CORBA-enabled tests were run on the same machine as the MotionService as separate tasks. Table I shows two runs of each test with 100 iterations. The runs of the different implementations were interleaved to compensate for the decreasing battery voltage.

Looking at the performance stats of the CORBA implementation used within Miro [20], one has to expect that even on this platform

the basic response time is still heavily dominated by the 100–Hz update cycle of the odometry reports. Doing a random wait before issuing the motor command, this latency becomes randomized to half an update cycle and should therefore contribute 5 ms to the averaged latency. Nevertheless, the remaining jitter, mostly coming from the nondeterministic latencies of the motors themselves, is much too high to make the existing latency of the CORBA overhead measurable. Only an increase of the variance could be measured, but note that the variance itself is also heavily biased by the 10–ms cycle of the odometry events.

V. Conclusions

We propose the adoption of object-oriented middleware for robots to improve the software development process. We designed and implemented Miro, which meets many of the requirements identified during problem analysis. Miro builds upon standard and widely used middleware packages like ACE, TAO CORBA, and Qt, which significantly eases integration into enterprise information-processing frameworks. The Miro implementation is stable and robust and has been applied in several development projects. Our overall experiences so far are very encouraging, supporting our view that the development of object-oriented middleware for robots is a necessary and beneficial step to improve our software development practices.

Miro is available as open source at http://smart.informatik.uni-ulm.de/Miro/.

References

[1] J.-S. Gutmann and C. Schlegel, "Amos: Comparison of scan matching approaches for self-localization in indoor environments," in *Proc. 1st Eurornicro Workshop Advanced Mobile Robots*, 1996, p. <AUTHOR: LOCATION? PAGE RANGE?>.

[2] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo localization: Efficient position estimation for mobile robots," in *Proc. Nat. Conf. Artificial Intelligence (AAAI)*, 1999, p. <AUTHOR: LOCATION? PAGE RANGE?>.

[3] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Automat.*, vol. RA-2, p. <AUTHOR: PAGE RANGE?>, Mar. 1986.

[4] A. Saffiotti, K. Konolige, and E. H. Ruspini, "A multivalued logic approach to integrating planning and control," *Artif. Intell.*, p. <AUTHOR: PAGE RANGE?>, Feb. 1995.

[5] "Telecom Log Service Specification," Object Management Group, <AUTHOR: LOCATION?>, 2000.

[6] D. C. Schmidt, A. Gokhale, T. Harrison, and G. Parulkar, "A high-performance endsystem architecture for realtime CORBA," *IEEE Commun. Mag.*, vol. 14, p. <AUTHOR: PAGE RANGE?>, Feb. 1997.

[7] T. H. Harrison, D. L. Levine, and D. C. Schmidt, "The design and performance of a real-time CORBA event service," in *Proc. OOPSLA'97*, Atlanta, GA, Oct. 1997, p. <AUTHOR: PAGE RANGE?>.

[8] S. Enderle, H. Utz, S. Sablatnög, S. Simon, G. Kraetzschmar, and G. Palm, "Miro: Middleware for autonomous mobile robots," in *Proc. 1st IFAC Conf. Telematics Applications in Automation and Robotics (TA200I)*, Webgarten, Germany, July 2001, p. <AUTHOR: PAGE RANGE?>.

[9] S. Sablatnög, S. Enderle, M. Dettinger, T. Boß, M. Livani, M. Dietz, J. Giebel, U. Meis, H. Folkerts, A. Neubeck, P. Schaeffer, M. Rittar, H. Hraxmeier, D. Maschke, G. Kraetzschmar, J. Kaiser, and G. Palm, "The Ulm Sparrows 99," in *Proc. RoboCup-99: Robot Soccer World Cup 111*, Lecture Notes in Computer Science, Berlin, Germany, 1999, p. <AUTHOR: PAGE RANGE?>.

[10] S. Enderle, "Probabilistic spatial representations for mapping and self-localization in autonomous mobile robots," <AUTHOR: Ph.D. DISSERTATION?, Comput. Sci. Dept., Univ. Ulm , Ulm, Germany, 2000.

[11] G. K. Kraetzschmar, S. Sablatnög, S. Enderle, and G. Palm, "Application of neurosymbolic integration for environment modeling in mobile robots," in *Hybrid Neural Systems*, S. Wermter and R. Sun, Eds. Berlin, Germany: Springer-Verlag, 2000, vol. 1778, Lecture Notes in Computer Science.

[12] G. K. Kraetzschmar, S. Sablatnög, S. Enderle, H. Utz, S. Simon, and G. Palm, "Integration of multiple representations and navigation concepts on autonomous mobile robots," in *Proc. SOAVE 2000*, Ilmenau, Germany, Oct. 2000, p. <AUTHOR: PAGE RANGE?>.

[13] S. Enderle, H. Folkerts, M. Ritter, S. Sablatnög, G. Kraetzschmar, and G. Palm, "Vision-based robot localization using sporadic features," in *Workshop Robot Vision 2001*, Auckland, New Zealand, Feb. 2001, p. <AUTHOR: PAGE RANGE?>.

[14] H. Utz, "Quo Vadis? Robuste Hierarchische Navigation für Autonome Mobile Roboter," Diplomarbeit (in German), Univ. Ulm, Ulm, Germany, 2000.

[15] *Pioneer I Software Manual*, RWI, Jaffrey, NH, 1996.

[16] K. Konolige, "Colbert: A language for rective control in saphira," in *Proc. German Conf. Artificial Intellgence*, Freiburg, Germany, 1997, p. <AUTHOR: PAGE RANGE?>.

[17] *Mobility 1.1, Robot Integmtion Software, User's Guide*, Real World Interface, Jaffrey, NH, 1999.

[18] R. Alami, R. Chatila, S. Fleury, M. Herrb, F. Lngrand, M. Khatib, B. Morisset, P. Moutarlier, and T. Simeon, "Around the lab in 40 days," in *Proc. 2000 IEEE Int. Conf. Robotics Automation (ICRA '2000)*, San Francisco, CA, Apr. 24–28, 2000, pp. 88–94.

[19] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," *Int. J. Robot. Res.*, p. <AUTHOR: PAGE RANGE? VOLUME? MONTH?>, 1997.

[20] C. O'Ryan, F. Kuhns, D. C. Schmidt, and J. Parsons, "Applying patterns us to develop a pluggable protocols framework for orb middleware," in *Design Patterns in Communications*, L. Rising, Ed. Cambridge, U.K.: Cambridge Univ. Press, 2000.