RoboDaemon - A device independent, network-oriented, modular mobile robot controller

Gregory Dudek and Robert Sim

Centre for Intelligent Machines, McGill University
3480 University St, Montréal, Québec, Canada H3A 2A7
{dudek,simra}@cim.mcgill.ca

***Abstract*— We discuss a software environment for multi-robot, multi-platform mobile robot control and simulation. Like others, we have observed that mobile robotics research is greatly facilitated by the availability of a suitable simulator for both vehicle kinematics as well as sensing, and have created an environment that permits this while allowing a large measure of device independence. By using a multi-processor internet-based architecture, our platform permits multiple users to use a variety of programming interfaces (visual, script-based or various application programming interfaces (API's)) to rapidly prototype methods to control multiple heterogeneous robots both in simulation and in real-world settings. We present an overview of our architecture and discuss its future directions.**

I. Introduction

In this paper, we describe a software system and associated architecture for controlling groups of mobile robots. Robotics, and particularly mobile robotics, is a problem domain that entails an exceptional requirement for the integration of multiple software and hardware technologies. In order to construct a truly effective mobile robot, several extant problems in computational perception, artificial intelligence and discrete control must be solved simultaneously. As the mechanical systems for mobile robotics research have matured it has become increasingly feasible to implement robotics sensing and navigation techniques in a device-independent manner. Further, the use of a standardized interface between robot control software and, concomitantly, higher-level modules greatly facilitates research on the integration of and interactions between different components that constitute a mobile robot system. We will describe a mobile robot control architecture that facilitates the use and combination of different software subsystems and different hardware platforms, providing a uniform and simple, but versatile interface to the user.

In this paper we focus on a particular software package called *RoboDaemon* that serves as a nexus for both robots and other software subsystems. By virtue of its role as an interface between components, this software package plays a critical role in defining an entire architecture (the McGill Mobile Robotics Architecture, or MMRA) for mobile robot control. In this paper, however, we confine our attention to the role of this single package itself since it provides somewhat better focus. It should also be noted that the role of connecting different devices and subsystems has also been approached recently in the context of robot operating systems and special purpose programming languages. While there are common themes to our own in such work, this work is particularly well suited to experimentation and rapid prototyping as opposed to the development of high performance applications.

Fig. 1. A core problem: how to control multiple, heterogeneous robots through a uniform interface.

One of our objectives has been to construct an infrastructure that would allow researchers to combine software and hardware modules for different mobile robotic subproblems. This has been necessary if only to permit the investigation of specific research problems that are our primary objective. In many ways, this mirrors the issues being faced by the community at large: to develop systems that integrate results not only from multiple individual researchers, but from multiple teams and institutions, each

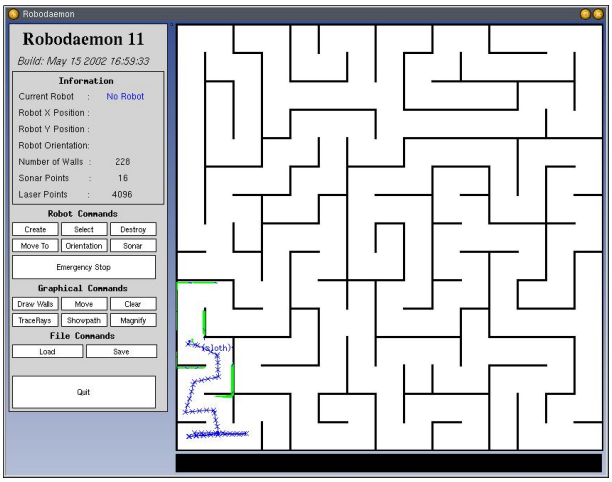

Fig. 2. The *RoboDaemon* graphical user interface

of whom may employ different robotic hardware or who may favour different programming environments. Such an objective involves motivating different individuals and groups to adopt components of one another's systems in the first place; that task motivates this work, but it is outside the scope of this discussion. Equally important, it presupposes that a mechanism and infrastructure exists that permits subsystems to be combined with limited effort, assuming there exists an underlying motive. (In fact, adding functionality to an existing subsystem is usually attractive to researchers if the infrastructure facilitates it.)

RoboDaemon, a core component of our infrastructure and control system, provides a variety of such services, namely the abstraction of robotic and sensing programming interfaces away from the low-level devices, the ability to develop robotic applications in a multi-user setting, and the ability to operate in full simulation when hardware resources are scarce, slow or expensive to operate. The objectives of our effort have many commonalities with other general-purpose robot control systems such as the *Task Control Architecture* [1], [2] (TCA) and the related systems TCX and IPC, the *Reactive Action Package* [3], [4] (RAP), the RAVE architecture [5], or even the subsumption architecture [6], [7]

A. Objectives

The objectives of our archtecture are to provide robotics researchers with an abstract programming layer that provides access to a variety of robotic and sensing platforms. Included among the goals of the project are:

1) **Hardware independence.** We have implemented robot drivers for a variety of vendors (including Nomadics, RWI and Cyberworks), providing users with an abtraction layer, allowing for vendor-independent application development (Figure 1).
2) **Multi-sensor platform.** Our architecture facilitates multiple sensing capabilities on each instantiated robot, including sonar, laser, infra-red sensing and video.
3) **Multi-user platform.** Our architecture employs a client-server model to allow multiple users to collaborate in the control of a single, or multiple, robots, working alone or together in the real world, or in one or more simulated worlds.
4) **Multi-interface platform.** We provide both graphical and command-line interfaces for user commands and feedback (Figure 2). Furthermore, we have implemented network-based clients for a variety of popular programming languages.
5) **Fully simulated environment.** When hardware resources are scarce or cumbersome to operate, or where time is of the essence, the user may opt to use simulators for any of the supported hardware platforms. Simulation includes modelling of robot kinematics (including noise models), sensor properties and the environment in which they operate. The user has the ability to instantiate simulators in parallel and real-time with real robots, or to operate in instantaneous mode, enabling the rapid evaluation of the outcome of user algorithms.
6) **Extensible environment.** Our architecture provides interfaces for extending functionality and behavior. A plug-in interface provides access to low-level functionality, enabling, for example, the addition of new graphical interfaces, collision-avoidance modes, or random map-generation features, while a module interface provides the ability to access external computational services, such as out-sourcing map or image data processing.

B. Outline

In the remainder of this paper we discuss related work, and the problem of promoting the integration of robotics solutions. We continue with an exposition of the architecture, provide an illustrative example, and discuss some of the realized benefits and challenges. Finally, we close with a discussion of open problems, shortcomings and findings from our work.

II. Related Work

Several research groups have considered the issues of building systems to control mobile robots. While our work is directed primarily at rapid prototyping and the support of diverse hardware and software systems and interfaces, related work on architectures for mobile robots has served as an important precursor.

Several authors have considered task planning as it applies to robotic systems. While the range of contributions is too long to enumerate, systems that combine planning

with exception management are particularly noteworthy. PRS (Procedural Reasoning System) [8] is a schema-based system that is particularly suited to goal and subgoal management and dealing with the sequencing and interaction between subgoals. PRS interacts with a "lower level" egocenteric representational framework called LPS - Local Perceptual Space which is core to the Saphira robot control package. The Saphira system [9], [8] uses LPS and shares several objectives and features with *RoboDaemon*. Saphira is also based on a client-server model of robot control whereby the robot provides a set of basic functions that can be used to interact with it. Like the system described here, it allows for extensibility on the part of end-users, although as far as we know the process is substantially more intensive than either the "plug-in" or scripting facilities we seek to provide.

The execution support language deals with task and goal management and provides rich mechanisms for dealing with failures and exceptional events [10], [11]. IPEM and similar systems also show their key strength in the domain of planning, plan monitoring and exception handling [12]. In our work we place very little emphasis on planning *per se* and assume it is handled by either a higher level supervisory system or (in the case of reactive planning) by systems below the level of abstraction of what we describe here.

ROGUE [13] is an architecture built on a real robot which provides algorithms for the integration of high-level planning, low-level robotic execution, and learning. Somewhat closer to this work is the Task Description Langauge (TDL) and its sibling Task Control Architecture (TCA) [14], [15]; architectures based on language extensions to C++. These provide mechanisms to facilitate planning, exception handling and inter-task communication and synchronization via TCA. Like the present work, TCA uses a centralised process control mechanism. Again, the emphasis in that work is more specifically on real time control and on planning than in the present work where a premium is put on ease of use, user interface and flexibility.

Several powerful commercial systems have also been developed, such as ControlShell by Real-Time Innovations. However, as observed in [16] their closed nature makes them difficult to integrate into an research environment except in special circumstances.

III. Promoting Integration

In many cases integrating the results of disparate researchers depends critically on having this objective in mind from the outset of a project life cycle. While it is possible to retrofit existing subsystems to work together, it is usually infeasible due to limits on available time and energy.

In addition, several projects have, understandably, focussed on the examination of what can be accomplished using a uniform consistent formalism which makes all the components of the systems adhere to a single methodology and communications mechanism. Systems based on the subsumption architecture, for example, were originally composed *in principle* of components that can *all* be described as simple finite-state automata [6], [7] and which communicate with one another using a specific class of abstraction mechanism[1]. While such an approach serves to exemplify the potential of a given approach, and even to simplify design, it can frustrate progress[2]. Further, since researchers are governed by their own peculiar tastes, it discourages widespread adoption.

In this work, we focus on factors that make the system readily acceptable to potential researchers working on particular subsystems. Three most important factors that have contributed to this acceptability are:

1) making basic functionality easy to learn and accomplish,
2) providing intuitive and informative, yet unobtrusive, feedback on system behaviour, and
3) providing readily available facilities that would be difficult to replicate otherwise.

a) Ease of acquisition: Persuading researchers to adopt a framework for system integration involves an undeniable component of salesmanship. While being easy to learn is not, in fact, a critical attribute for long-term users or system functionality, it has proven very important in attracting users initially (especially since they are often on tight schedules, even with their initial implementation efforts).

b) Informative feedback: A common difficulty in working with abstracted interfaces is the lack of informative diagnostic information when failures occur "under the hood". Users are also often frustrated by black-box interfaces that fail to produce an expected result. Finally, textual output can be useful to diagnose some failures but, in a robotics context and particularly in simulation, visual output can be far more compelling and informative.

c) Facilities: The third aspect that encourages adoption is the provision of facilities and features that would be cumbersome to replicate otherwise. In the case of robot controllers, essential features include a high level of control abstraction (so that the developer can ignore the underlying actuation problems in moving the robot), robust simulation and sensor modelling, and rich visualization tools for observing sensor output and obtaining

[1]Several highly capable systems have been constructed using this architecture and it is possible to relax the standard design constraints if necessary.

[2]For example, the widespread successful adoption of the C programming language is partly attributed to its frighteningly lax enforcement of stylistic and syntactic rules, for example with respect to type checking.

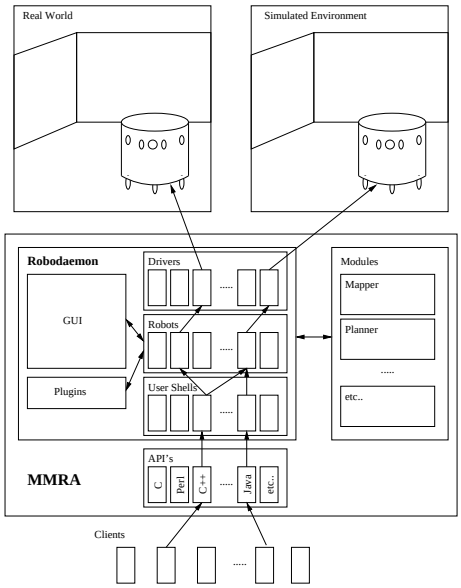

Fig. 3. *RoboDaemon* and the MMRA architecture.

feedback on the robot's state. It should be noted that this goal is often in competition with the ease of use criteria and it is important to keep this tension in mind when adding new features.

In addition to the attributes noted above, there are several additional features that we believe are important for a widely acceptable common infrastructure.

IV. System Architecture

Figure 3 provides an overview of the *RoboDaemon* package in the context of the MMRA architecture. *RoboDaemon*'s core role is to receive client connections and manage a set of instantiated robots. The robot instances may interface to real robots through device drivers and wireless connections, or may instead communicate with simulated counterparts. *RoboDaemon* also provides visualization tools and a set of plugins that define high-level robot behaviours. Finally, *RoboDaemon* provides a module interface that facilitates inter-process communication with subsystems that perform CPU-intensive processing tasks. In the remainder of this section we will discuss the various components of the architecture.

A. Simulation

Hardware simulation has become a major factor in many domains. Despite this, many robotic systems have only limited simulation abilities and only a few can *realistically* simulate the performance of their sensors. In many ways, a poor simulation is worse than none at all as it leads to false expectations. The advantages of an effective simulation are threefold: demand on scarce or high-maintenance resources is reduced; extensive sets of experiments can be readily developed, conducted and exhaustively examined; and experiments can be repeated when anomalies are detected.

Fig. 4. Simulated camera viewpoint.

There are three aspects to simulation in our architecture: robot modelling, sensor modelling, and environment modelling.

- **Robot models:** We have endeavoured to produce accurate kinematic models of our robots. This includes not only addressing the motion properties of the robot, but also idiosyncracies such as quantization error (for example, the differential drive on the Nomadics Scout exhibits significant quantization error when rotations are induced), and appropriate noise models for odometry error.
- **Sensor models:** We employ the sonar simulator described in [17], [18] which takes into account attenuation, beam width, multi-path reflections and material reflectivity. We have also developed a laser simulator and modeled noise properties as a function of distance, as well as accurate models of blind spots and the pose of the real sensor on our real robots. Finally, through our plugin and module interfaces, we have developed camera simulators that render the robot's view point using OpenGL or PovRay (Figure 4).
- **Environment models:** Part and parcel with sensor modelling is the problem of representing the environment with sufficient accuracy. Furthermore, different problem domains require different levels of realism. Our core environment facility employs planar polygons to represent walls and other obstacles. The reflectivity of these polygons can be controlled individually, and texture maps can be applied for visual sensors. Furthermore, we have additional tools for converting large sets of real sonar and laser data into line segment models for producing realistic simulations of planar environments.

B. Run-time Modes

RoboDaemon has a variety of run-time modes to suit the needs of developers:

- **Real-time vs instantaneous events:** While the robot simulators are event driven, it is possible to turn off the clock, allowing events to be processed without delay. The primary benefit of instantaneous mode is that long simulations can be sped up, allowing for faster debugging and completion times.
- **Synchronous vs asynchronous control:** *RoboDaemon* allows the user to define the level of interaction they will have with the robots. For many applications, stop and shoot mode is sufficient, not to mention straightforward to debug, whereas more intensive applications call for continuous velocity and acceleration control. A variety of safeguards are taken to ensure that a robot in asynchronous mode avoids collisions. These modes also apply equally to real and simulated agents.

C. User Interface

As we have mentioned, *RoboDaemon* supports both command-line and graphical user interaction. The server also facilitates remote interaction using telnet or one of the client API's. The graphical interface supports robot creation, point and click navigation, map editing and user-defined buttons for higher-level commands. Visualization is a key component of the debugging process and the path of the robot (both real, if known, and odometry-based), sensor measurements, and user-defined notes can all be plotted visually. Furthermore, the user is not restricted to using the built-in X11 client, but can turn any command-line connection to the server, local or remote, into a GUI event stream for contructing custom GUI's.

The command-line interface provides a slightly richer set of primitives to work with than the GUI. In addition to common navigation commands, the user can specify local variables and macros, execute shell escapes, read a script of commands from file, and examine and set various internal parameters controlling sensing and individual robots. The overall design is object oriented, such that each robot can be configured individually, and each connected user can set their own shell preferences.

D. Network-Based Client Support

As mentioned, the system implements a client-server model, allowing multiple clients to connect, possibly from remote locations. Our current set of client API's include C, C++, Perl, and Java. It is also possible to simply connect and supply commands in ASCII format. The versatility of this approach is obvious– multiple users can gain access to the robotic hardware, in the language of their preference.

E. Robot Drivers

Our multi-vendor implementation currently supports Nomadics Scouts and Nomad 200 robots, RWI B-12, and Cyberworks robots. We have implemented simulators for all of these models, as well as a generic cylindrical robot simulator (on which the Nomadics and RWI simulators are based), and an ever-popular Point-Robot Simulator, which is convenient for studying theoretical problems without the burden of additional safety constraints. Among our current projects is the development of a driver and simulator for Sony Aibo robots.

F. Embedded Plugins and External Modules

One of the core goals of this project is the facilitation of subsystem integration. This goal is often circumvented by incompatible programming interfaces and data types. We provide two venues for users to extend the functionality of the architecture, often with minimal revision to the imported system.

a) Plugins: Plugins provide a method for extending core *RoboDaemon* functionality. The plugin interface is an API that gives the programmer access to robot, gui and shell internals. Examples of ideal plugins are renderers for camera simulation, new GUI extensions, and path planners. A key feature of the plugin interface is that new plugins can be created and loaded dynamically without recompilation of the core system.

b) Modules: Modules provide an extremely powerful method for system integration. The module interface allows users to import commands from other running processes by interfacing with that process' command line interface.

We illustrate the interface by example. We have a software package, called Rhys, that processes range images in order to fit geometric models[19]. Rhys operates through a command line interface. The module interface allows the *RoboDaemon* user to run an instance of Rhys and import all of the commands available in Rhys to *RoboDaemon*. For example, if Rhys has a command called `'fit'`, one can call the command `'rhys.fit'` in *RoboDaemon*. Map and robot data is shared with the module using simple file formats, and it is possible for a "*RoboDaemon*-aware" application to send explicit commands back to *RoboDaemon* for execution. The module interface requires only that the subsystem can operate through a command-line interface. The key point to be noted is that modules can be developed independently of the MMRA architecture, in whichever programming environment the developer prefers (in fact, Rhys was developed long before the current version of *RoboDaemon* was written).

V. Validation

Our experience with *RoboDaemon* has suggested it meets several of its key design goals. In particular, we

Fig. 5. Cooperative localization setup. (Figure courtesy Yiannis Rekleitis)

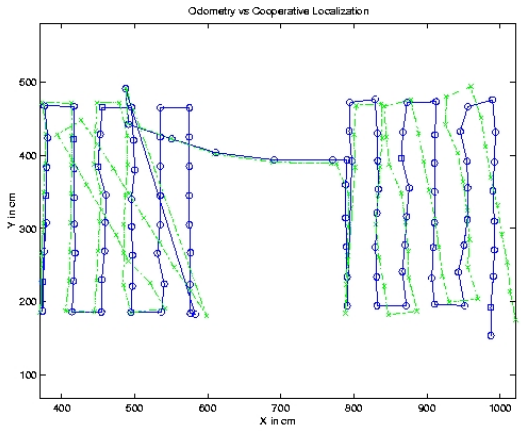

Fig. 6. Odometric (x) vs Tracker-corrected (o) trajectories of the robot.

Fig. 7. Robot viewpoint during exploration.

have observed that unfamiliar users can acquire a working knowledge and implement control algorithms rapidly.

Specifically, we have found that students in an introductory robotics course having no prior robot programming experience are able to learn the interface and design and test sensor-based robotics algorithms in limited time (for example wall following, obstance avoidance, etc.). Typically, students are able to produce working prototypes within a week (concurrent with other course work) assuming they already understand the algorithms to be used. The quality and utility of the simulation is further validated when the resulting algorithms were tested on the real robots in our laboratory with a roughly 85% success rate for algorithms that worked well in simulation (and failures due primarily to incorrectly set parameters).

RoboDaemon also meets the need for enabling subsystem integration. For the purposes of illustration, we present here an example of how the platform has been used to integrate research from various sources in our laboratory. The results of these experiments have been reported elsewhere [20].

Our recent work on contructing visual maps requires that a collection of training images be obtained with ground-truth position information as to where the images were collected[21]. Related research in our lab entails using multiple robots to perform "cooperative localization" whereby a laser range finder mounted on an observing robot is used to locate the pose of a second robot with a mounted target[20]. Furthermore, the cooperative localization method employs Rhys, the line segment fitting software described above.

The scenario we are faced with is one in which three processes (mapper, localizer, and line fitter) must interact to control two robots with distinctly different sensor configurations (one with a laser, the other with a camera) (Figure 5). *RoboDaemon* acts as the intermediary, servicing sensing requests and motion commands, enabling images to be added to the mapper and employing Rhys to outsource data processing. The localizer interfaces with *RoboDaemon* with the C API over a TCP/IP socket, while the mapper collects images through *RoboDaemon* and obtains the ground truth pose estimates with the Perl API. Rhys can operate in the background as a module, although in our example it was connected to directly by the localizer. Figure 6 illustrates the error-prone odometry-based trajectory of the robot versus the "ground truth" obtained from the laser setup. One of the images collected for the visual mapper is depicted in Figure 7.

VI. Discussion

We have presented an architecture for robotic application development in a heterogeneous linguistic, vendor and user environment. Our architecture provides a rich set of facilities and features that encourage the integration of subsystems, and allows for both real-time control and maintenance of a collection of robots as well as full simulation of their kinematics and sensor properties. We

believe that a core development and control environment such as the one presented here is essential to the succesful development of complete, functional robotic systems. Our future work involves the ongoing development of this platform, including increased integration with other robotic platforms and with other researchers' navigation, localization and map-building solutions.

VII. REFERENCES

[1] R. Simmons, "Structured control for autonomous robots," *IEEE Transactions on IEEE Transactions on Robotics and Automation*, vol. 10, February 1994.

[2] R. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan, "A layered architecture for office delivery robots," in *Proceedings of First International Conference on Autonomous Agents* (W. L. Johnson, ed.), (Marina del Rey, CA), pp. 245–252, ACM Press, New York, NY, February 1997.

[3] R. J. Firby, *Adaptive Execution in Complex Dynamic Domains*. New Haven, CT: Yale University Technical Report YALEU/CSD/RR 672, January 1989.

[4] R. J. Firby, "The rap lanauge manual," Animate Agent Project Working Note AAP-6, University of Chicago, March 1995.

[5] K. Dixon, J. Dolan, W. Huang, C. Paredis, and P. Khosla, "Rave: A real and virtual environment for multiple mobile robot systems," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'99)*, vol. 3, pp. 1360–1367, October 1999.

[6] R. A. Brooks, "Intelligence without representation," *Artificial Intelligence*, vol. 47, pp. 139–1159, 1991.

[7] R. Brooks, "A layered intelligent control system for a mobile robot," in *Robotics Research 3* (Faugeras and Giralt, eds.), pp. 365–372, MIT Press, 1986.

[8] K. Konolige and K. Myers, "The saphira architecture for autonomous mobile robots," in *AI and Mobile Robots* (D. K. R. P. B. R. Murphy, ed.), Cambridge, MA: MIT Press, 1998.

[9] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti, "The saphira architecture: A design for autonomy," *Journal of experimental & theoretical artificial intelligence*, vol. 9, pp. 215–235, 1997.

[10] B. Pell, E. Gat, R. Keesing, N. Muscettola, and B. Smith, "Plan execution for autonomous spacecraft," in *Proc. International Joint Conf on Artificial Intelligence*, AAAI Press, August 1997.

[11] R. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack, "Experiences with an architecture for intelligent, reactive agents," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, march 1997.

[12] J. A. Ambros-Ingerson and S. Steel, "Integrating planning, execution and monitoring," in *Proc. Seventh National Conference on Artificial Intelligence*, AAAI Press, 1988.

[13] K. Z. Haigh and M. M. Veloso, "Interleaving planning and robot execution for asynchronous user requests," *Autonomous Robots*, vol. 5, pp. 79–95, march 1998.

[14] R. Simmons, T. Smith, M. B. Dias, D. Goldberg, D. Hershberger, A. Stentz, and R. Zlot, "A layered architecture for coordination of mobile robots," *Multi-Robot Systems: From Swarms to Intelligent Automata*, p. Kluwer, 2002.

[15] E. Coste-Maniere and R. Simmons, "Architecture, the backbone of robotic systems," in *Proceedings of the IEEE Conference on Robotics and Automation*, (San Francisco, CA), IEEE press, April 2000.

[16] L. Petersson, D. Austin, and H. Christensen, "Dca: A distributed control architecture for robotics," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2361–2368, IEEE Press, October 2001.

[17] D. Wilkes, G. Dudek, M. Jenkin, and E. Milios, "Modelling sonar range sensors," in *Advances in Machine Vision: Strategies and Applications* (C. Archibald and E. Petriu, eds.), Singapore: World Scientific Press, 1992.

[18] G. Dudek, "Shape classification and scale-space texture," *1992 Association for Research in Vision and Opthamology Program Book*, May 1992.

[19] P. MacKenzie and G. Dudek, "Precise positioning using model-based maps," in *Proceedings of the International Conference on Robotics and Automation*, (San Diego, CA), IEEE Press, 1994.

[20] I. Rekleitis, R. Sim, G. Dudek, and E. Milios, "Collaborative exploration for the construction of visual maps," in *IEEE/RSJ/ International Conference on Intelligent Robots and Systems*, vol. 3, (Maui, Hawaii, USA), pp. 1269–1274, IEEE/RSJ, IEEE,ISBN 0-7803-6614-X, October 2001. http://www.cim.mcgill.ca/˜yiannis/Publications/iros01.pdf.

[21] R. Sim and G. Dudek, "Learning and evaluating visual features for pose estimation," in *Proceedings of the Seventh IEEE International Conference on Computer Vision(ICCV)*, (Kerkyra, Greece), IEEE Press, sept 1999.