Robot Programming Environments

**Bruce MacDonald, David Yuen, Sylvia Wong, Evan Woo,
Rowan Gronlund, Toby Collett, Félix-Étienne Trépanier, Geoff Biggs**

Department of Electrical and Electronic Engineering, University of Auckland
Private Bag 92019, Auckland, New Zealand

Email: b.macdonald at auckland.ac.nz

Abstract: Robot programming systems have not kept pace with developments in general purpose programming environments. Robot systems have special demands related to their complex interactions in real environments, and their complex sensors and actuators. We focus on robot programming in a laboratory research environment where robots, operating systems, hardware platforms, programming languages, and researchers all differ and change. Robot programming systems must provide appropriate human–robot programming interaction, programming languages and tools, and distributed infrastructures. We describe our work in developing a robot programming system, including our three layer distributed CORBA based, service broker application infrastructure for distributed robot programming, and our associated graphical and simulation tool which provides a virtual environment with rich graphics capabilities.

Keywords: Robot programming, human–robot interaction, robot infrastructure

1 INTRODUCTION

Robot programming systems have not kept pace with developments in general purpose programming environments and robot researchers face difficulties developing large software systems. Much of the software infrastructure is proprietary, much is necessarily targeted at specific hardware, robot software development kits may be limiting, and there is a lack of open standards to promote collaboration, code reuse and integration.

Robot systems have special demands related to the complex interactions robots have in real environments, and the complex sensors and actuators that robots use, including:

- Up to thousands of devices for input, output and storage, which far exceed human programmers' familiar senses and effectors, compared to the few devices in a desktop or server.
- Simultaneous and unrelated activity on many inputs and outputs.
- Real time requirements, as the automation system must operate in the real environment.
- Unexpected real world conditions.
- Wide variations in hardware and interfaces, as opposed to the highly commoditized desktop.

Programmers of robot arms and other complex articulated automatic devices, must also deal with non–intuitive geometry. Programmers of mobile robots must deal with widely varying and unpredictable conditions as the robot moves through its environment. The paper will discuss our work in providing a robot programming environment.

2 ROBOT PROGRAMMING

We view robot programming systems as having three important conceptual components that are of interest to their designers (Fig. 1):

- The programming component, including designs for programming language/s, libraries and application programming interfaces (APIs), which enable a programmer to describe desired robot behaviour.
- The underlying infrastructure including designs for architectures that support and execute robot behaviour descriptions, especially in distributed environments.
- The design of interactive systems that allow the human programmer to interact with the programming component, to create, modify and examine programs and system resources, both statically and during execution. The human programmer may also interact with

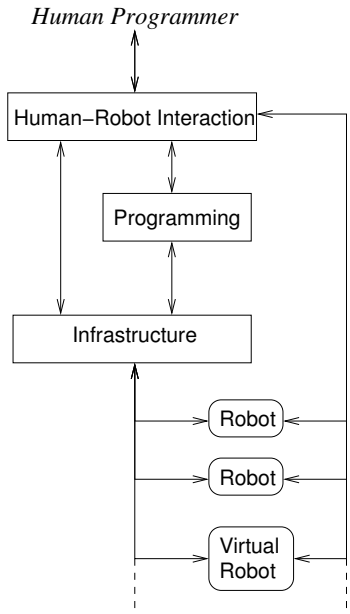

Figure 1: Parts of a robot programming system.

the infrastructure component, to examine, monitor and configure resources, and directly with robots as they perform tasks.

There are other components that are not of particular concern to designers of robot programming systems, such as the robots themselves, operating systems, compilers, robot hardware drivers and so on. A few aspects, such as real time operating system performance, will be of concern. The following sections describe the three main components.

2.1 Programming

The programming activity is to describe desired robot behaviour, and must be supported by a programming system. Robot programming systems can be distinguished by both their aim and by their method of programming. The aim may be programming at the system level or at the task level. System programming is typically undertaken by developers before releasing the software, and often includes low or medium level tasks. Task level programming is the allocation of tasks by the consumer and is at a higher level, sometimes with little overt programming, perhaps viewed as configuration. For example, the developer might program the ability to perform a navigation or guiding task into the robot using system level programming. Consumers might use the task–based interface to program the robot to guide people through a museum.

There are two main programming methods, manual and automatic. Manual programming involves text–based or graphical systems. Automatic programming includes programming by demonstration (PBD), currently popular for training specific tasks, particularly in industry. Considerable effort is aimed at improving PBD systems. Rather than simply playing back the demonstration, the robot may pick out important movements (such as grasps), and plan its own path between these points, or it may execute the key steps in a different order [1, 2].

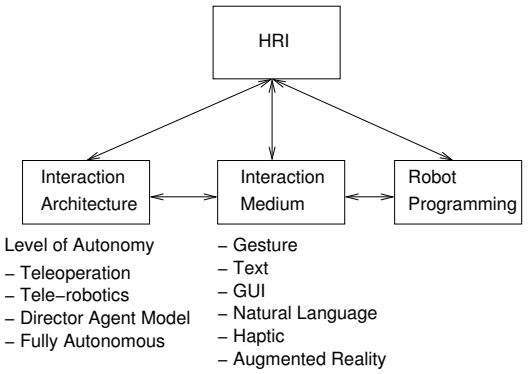

Figure 2: Aspects of human–robot interaction.

Text-based programming is common in industry, where simple robot languages are used, typically provided by the robot developer. A BASIC-like syntax is often used, with built in robot commands. Recent work in text–based systems has diverged from these robot–specific languages to develop more general purpose higher level programming languages suitable for any robot. Typically, this involves extending existing languages such as C++ [3], Java [4, 5], and Haskell [6, 7].

Graphical (or icon based) languages are a hybrid automatic/manual method; manual because the user must show the actions to perform, automatic since code is generated from the results of the user interaction. The user has greater control over the robot's actions compared to a fully automatic system. Graphical languages require less technical skill than a text-based system, sacrificing versatility for usability, and are more useful for task level programming. Flowcharts [8], [9, a popular graphical system for children] or graphs [10] are used.

2.2 Human–robot interaction (HRI)

Here the design is about how humans interact with robots, during all the activities associated with programming. Fig. 2 distinguishes issues within human–robot programming design: architecture, medium of interaction, and programming related. The architecture varies from the direct control model through the director–agent model, where the human gives instruction and supervision, to fully autonomous mode. There is considerable interest in how humans and robots interact. However, unlike work in the HRI areas of direct control (teleoperation/telerobotics), social interaction and "end-user programming" there has been little work on HRI at the system development level. The

current research needs integrating with program development stages, allowing more effective development with more interaction between developer and robot.

The three main levels of HRI can be distinguished by the purpose of interaction. The lowest level is robot development, which corresponds to the system level programming activities of the previous section, where the work is on developing the robot system itself. The next level of interaction is that of customizing the robot tasks, or task level programming, for example giving a list of commands to a vacuuming robot or an assembly robot. The final level is social interaction, for example in a tour guide robot.

Our programming environment focuses on robot prototype development, where interface development has been largely neglected; most researchers use traditional application development and programming tools with standard keyboard, screen, and mouse interfaces.

2.3 Infrastructure

The infrastructure supports and executes robot behaviour descriptions. There is much work on robot programming systems, but little coherence and focus on the underlying architecture. Themes include client/server tools, specific methods for producing robot software, for example graphically, reusable software architecture, portable APIs, hierarchies of classes with attention to their interfaces [11, 12], fault tolerance, component interaction via a blackboard [12], mission programming aimed at end users [13], layer architectures, real–time object–oriented automation, component based systems for real time programming [14]. Many propose simulation and virtual reality to support programming.

Some authors describe distributed robot programming systems, including architectures for: the separation of cognitive from reactive components [15], planning [16], components and patterns [17], distributed development with three levels of workflow: user, system and execution [18], hybrid models for integrating decision and reactive levels [19], a layered, object–oriented client/server architecture for sensorimotor systems [20]. Orocos [21] focuses on object–oriented components and patterns. CORBA inspires the distributed communications and is used in some work. Miro [22] is an object–oriented layered client/server robot middleware system based on ACE [23] and the associated real–time CORBA ORB, TAO. MCA2 [24] focuses on reusable modules all with the same standardised interface. Claraty [25] provides an architecture with functional and decision layers, and a set of functional components for JPL's mobile robot systems.

3 OBJECTIVES

We focus on robot programming in a laboratory research environment; we want to avoid time consuming practices such as porting software to special robot environments.

3.1 Programming component

3.1.1 Developer tools must provide editors enhanced with code–aware presentation, browsing, useful application components, help systems, debuggers, and inspectors. Simulations of robot behaviour are essential, enabling programmers to visualise and evaluate robot programs in repeatable, safe environments. Good quality models and simulations of environments are essential, enabling programmers to reason about how the robot behaves in an environment, and to test that reasoning.

3.1.2 Standards and protocols, components and reuse Robot programming should follow common standards and protocols to encourage the creation of reusable components, including not only object–oriented concepts but also standards for robot intercommunication. The programming environment should be open and extendable to allow experimentation with programming abstractions (eg object–oriented, functional, reactive and behaviour-based). The standards and common components should be provided in open source form, to enhance openness and extendability, to allow importing of others' work, and encourage other groups to contribute.

3.1.3 Sensors and actuators Good I/O is essential, including sensor and actuator models for reasoning about errors, asynchrony, uncertainty, imperfections and noise. Abstractions for motion commands and sensor management should provide convenient programming, for example guarded moves. Real time APIs to sensors and actuators must be provided to the programmer, and accurate time–stamps recorded for sensory data.

3.1.4 Control systems Programmers need libraries and tools for creating sophisticated controllers, with accurate notions of continuous time control (eg using graphical tools such as those provided with Matlab).

3.1.5 Concurrency and Real time Multithreading, real time operating system and networking must provide for interactions with other robots.

3.1.6 Autonomy and intelligence Autonomous, intelligent decision making facilities are needed (eg inference for problem solving, reasoning about time, and learning from a teacher/experience/simulation).

3.1.7 Sophisticated exception processing is needed, not just for errors, but to manage uncer-

tainty, for example averaging erroneous results or allowing retries (see [26]). Robust error handling, recovery, and fault–tolerance are essential.

3.1.8 Robotics oriented programming language abstractions should provide domain specific data and program abstractions, including libraries and operators for a variety of types (eg basic number, string, enumerated and logical types, I/O oriented types, time, probabilities as well as modern compound structures such as lists, sets, dictionaries, tuples, and first class functions). Some types should be tuned to analog and digital data I/O. Domains specific to robotics include positions and orientations, geometry and transforms, geometric paths with interpolation, path planning and control, three dimensional objects, maps, sensory data, vision, signal processing, kinematics, statics, dynamics, force control, temporal reasoning, learning and artificial intelligence. The infrastructure should bring together robotic systems developed in different domains, enabling common tools, interoperability and reuse [27].

3.1.9 Sound foundations Well–founded robot languages provide the possibility of verifying accuracy, safety and other characteristics of robot software (eg functional languages [6, 7, 27, 28]). Robot programs might be generated from a requirements specification [29].

3.2 HRI component

3.2.1 Rich set of human interaction modes Robot systems need comprehensive interaction techniques, including graphical user interfaces, speech, gesture interaction, screen based graphical visualisation of the robot's real and simulated behaviour, electronically augmented visual representations, and haptic interaction.

3.2.2 Robot components primarily designed for interaction Robot systems will need to find, recognise, and track the humans they interact with, and they will need good facilities for cooperating with humans (eg to physically manipulate objects). The robot must model and reason about the human programmer's intentions, and be able to recognise plans presented by the human [30, 31].

3.2.3 Interactive programming tools The development environment should provide the typical, full set of tools including intelligent editing, program stepping, breakpoints, browsing of code and data, dynamic tracing, intelligent logging and monitoring. In robot systems, where the human interacts directly with the robot, tools must be integrated in direct HRI, for example using programming/teaching techniques such as guiding, demonstration, instruction, text–generation, and text–skeleton (see [32, 33, 34]). At times a programmer will manipulate the robot directly, using teleoperation, with varying degrees of local autonomy for the robot depending on the bandwidth and latency of the communication link. The interactive system should provide virtual collaboration facilities such as metaphors, for working at a distance, with groups of programmers as well as groups of robots. The virtual world provided by simulations should enhance the programming environment, by providing more information and interaction, for example collaboration with other programmers and robots that are not present locally.

3.2.4 Robot information database The interactive tools should allow users to create, store and manipulate representations of robots, objects, environments, robot locations and paths in a convenient way. For example, enabling a programmer to easily name paths, objects and locations for a robot to manipulate or follow.

3.3 Infrastructure component

Many authors describe systems with strong architectural decisions, necessary in software production settings. Our work focuses on reuse for researchers via an easily accessible infrastructure. Software components are the eventual goal, but we usually don't know enough to create them until the research has matured to standard forms. The infrastructure must enable experimentation.

At the low level, the infrastructure should accommodate different languages and platforms, and promote collaboration, software reuse, and leverage between projects. At a higher level the infrastructure services should ideally be architecturally neutral so that we may experiment with different architectural styles. While it must be possible to program at different levels, such as the sensor/actuator, control, geometry planning, problem solving, and simulation levels, there should not be strong barriers between levels. For example, low level events may be important at the highest level.

4 SOME COMPONENTS OF A ROBOT PROGRAMMING SYSTEM

Our work includes: robot localisation, coverage and navigation [35, 36, 37, 38, 39]; robot arm control; speech command recognition; environment modelling and visualisation; machine learning, problem solving and instructable systems for teaching robots [30, 33, 34, 40, 41, 42]. Two recent projects include the development of an underlying infrastructure for our robot programming systems, and an associated simulation and visualisation tool, discussed below.

4.1 Distributed robot programming application infrastructure

Evan Woo's [43] three layer CORBA based, service broker application architecture provides a

distributed programming infrastructure, including tests on robots ranging from those with only a single microcontroller (LEGO Mindstorm, Khepera) to those with a considerably more sophisticated platform (B21r). This will improve the integration and leverage between projects. It adopts a CORBA based application architecture comprising three layers: application, infrastructure services, and middleware. The application layer contains components developed by researchers, which are registered as services. The infrastructure services layer is a broker, following the CORBA Trader specification. The broker provides the protocol between services as well as query facilities for clients. The middleware layer handles communications between services. Our reference implementation includes robot services for the B21r, Khepera, and LEGO Mindstorm robots, the service broker, a remote robot control application, plus a web–enabled version. Real–time facilities are not yet included. Although there are issues to overcome, in particular ORB compatibility and performance in a busy network, the application infrastructure provides a good framework for researchers.

4.2 Robot simulation and visualisation tool

Félix Trépanier's [44, 45] Graphical Simulation and Visualisation (GSV) tool aims to help humans visualise robot behaviours operating in different environments. It is integrated in a broader robot programming environment supported by the service based architecture mentioned above. The GSV tool can display any robot model controlled by a robot behaviour using a state-of-the-art game engine to render the virtual world thus giving an accurate 3D perspective of the behaviour of the robots in the virtual environment. The tool also uses registered services to control the simulated robots. The tool can register itself as a simulation service so robotic applications can use the tool without the intervention of a human. A simulation can be accessed by many researchers at the same time and the robots are controlled by behaviours, possibly from another research group. In addition, a simulation can be recorded and replayed. Performance measurements show the simulation of five robots each with typical behaviour in the same environment.

5 CONCLUSION

The design and development of robot programming environments face considerable barriers to creating convenient human–robot interaction for programming. A rich set of programming components is needed, along with sophisticated human–robot interaction techniques for programming, and a broad set of distributed infrastructure components. Our goal is to design and build a comprehensive, versatile system for programming robots.

References

[1] J. Chen and A. Zelinsky, "Programming by demonstration: removing sub-optimal actions in a partially known configuration space," in *Proc IEEE Intl. Conf. on Robotics and Automation (ICRA '01)*, vol. 4, May 2001, pp. 4096–4103.

[2] K. Ogawara, J. Takamatsu, H. Kimura, and K. Ikeuchi, "Generation of a task model by integrating multiple observations of human demonstrations," in *Proc IEEE Intl. Conf. on Robotics and Automation (ICRA '02)*, vol. 2, May 2002, pp. 1545–1550.

[3] X. Dai, G. Hager, and J. Peterson, "Specifying behavior in C++," in *Proc IEEE Intl. Conf. on Robotics and Automation (ICRA '02)*, vol. 1, May 2002, pp. 153–160.

[4] D. Hardin, M. Frerking, P. Wiley, and G. Bolella, "Getting down and dirty: device-level programming using the real-time specification for Java," in *Object-Oriented Real-Time Distributed Computing, 2002. (ISORC 2002). Proc. Fifth IEEE International Symposium on*, 2002, pp. 457–464.

[5] Y. Kanayama and C. Wu, "It's time to make mobile robots programmable," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '00)*, vol. 1, April 2000, pp. 329–34.

[6] J. Peterson, G. Hager, and A. Serjentov, "Composable robot controllers," in *Computational Intelligence in Robotics and Automation, 2001. Proc 2001 IEEE International Symposium on*, 2001, pp. 149–154.

[7] P. Hudak, A. Courtney, H. Nilsson, and J. Peterson, "Arrows, robots, and functional reactive programming," in *Summer School on Advanced Functional Programming 2002, Oxford University*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2003, to Appear.

[8] R. Bischoff, A. Kazi, and M. Seyfarth, "The morpha style guide for icon-based programming," in *Robot and Human Interactive Communication, 2002. Proc. 11th IEEE International Workshop on*, 2002, pp. 482–487.

[9] "Lego mindstorms," http://mindstorms.lego.com /eng/products/ris/rissoft.asp.

[10] A. Bredenfeld and G. Indiveri, "Robot behavior engineering using dd-designer," in *Robotics and Automation, 2001. Proc 2001 ICRA. IEEE International Conference on*, vol. 1, 2001, pp. 205–210.

[11] D. Miller and R. Lennox, "An object-oriented environment for robot system architectures," *IEEE Control Syst. Mag.*, vol. 11, no. 2, pp. 14–23, February 1991.

[12] B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, and M. Balabanovic, "A domain-specific software architecture for adaptive intelligent systems," *IEEE Trans. Software Eng.*, vol. 21, no. 4, pp. 288–301, April 1995.

[13] E. Coste-Manière and N. Turro, "The MAESTRO language and its environment: specification, validation and control of robotic missions," in *Proc. IROS '97, Intelligent Robots and Systems*, vol. 2, 7–11 September 1997, pp. 836–41.

[14] S. A. Schneider, V. W. Chen, and G. Pardo-Castellote, "The ControlShell component-based real-time programming system," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '95)*, vol. 3, 21–27 May 1995, pp. 2381–8.

[15] M. Piaggio, A. Sgorgbissa, and R. Zaccaria, "Ethnos: a light architecture for real-time mobile robotics," in *Proc IROS'99 Intell Robots & Sys's*, vol. 3, 1999, pp. 1292–97.

[16] S. Flavell, S. Winter, D. Wilson, and E. Hart, "Design of a distributed software architecture for an intelligent planning system," *IEE Proceedings – Control Theory and Applications*, vol. 143, no. 2, pp. 125–31, March 1996.

[17] J. Beck, J. Reagin, T. Sweeny, R. Anderson, and T. Garner, "Applying a component-based software architecture to robotic workcell applications," *IEEE Trans. Robot. Automat.*, vol. 16, no. 3, pp. 207–17, June 2000.

[18] O. Stasse and Y. Kuniyoshi, "PredN: achieving efficiency and code re-usability in a programming system for complex robotic applications," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA '00)*, vol. 1, April 2000, pp. 81–7.

[19] S. Fleury, M. Herrb, and R. Chatila, "Gen^{en}oM: a tool for the specification and the implementation of operating modules in a distributed robot architecture," in *Proc. IROS '97, Intelligent Robots and Systems*, vol. 2, 7–11 September 1997, pp. 842–9.

[20] C. Schlegel and R. Wörz, "The software framework SMARTSOFT for implementing sensorimotor systems," in *Proc. IROS '99 Intelligent Robots and Systems*, vol. 3, 1999, pp. 1610–6.

[21] "Orocos," http://www.orocos.org.

[22] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro — middleware for mobile robot applications," *IEEE Trans. Robot. Automat.*, vol. 18, no. 4, pp. 493–7, Aug 2002.

[23] D. Schmidt, "Adaptive communication environment (ACE)." [Online]. Available: http://www.cs.wustl.edu/ schmidt/ACE.html

[24] "Mca2." [Online]. Available: http://mca2.sf.net

[25] "CLARAty," http://robotics.jpl.nasa.gov/tasks/claraty/homepage.html.

[26] I. Cox and N. Gehani, "Exception handling in robotics," *IEEE Computer*, vol. 22, no. 3, pp. 43–49, March 1989.

[27] I. Pembeci, H. Nilsson, and G. Hager, "Functional reactive robotics: an exercise in principled integration of domain-specific languages," in *Proc. Fourth ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming*, October 2002.

[28] I. Navarro, K. Lundqvist, and N. Leveson, "An intent-specifications model for a robotic software control system," in *Digital Avionics Systems, 2001. DASC. The 20th Conference*, vol. 2, 2001, pp. 8E1/1–8E1/12.

[29] K. Czarnecki and U. W. Eisenecker, *Generative Programming: methods, tools, and applications*. Addison-Wesley, 2000.

[30] J. D. Lewis and B. A. MacDonald, "Machine learning under felicity conditions: exploiting pedagogical behavior," in *Proceedings of AI / CS '92*, Limerick, Ireland, September 1992.

[31] W. I. Yared and T. B. Sheridan, "Plan recognition and generalization in command languages with application to telerobotics," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, no. 2, pp. 327–338, March/April 1991.

[32] P. D. Summers and D. D. Grossman, "XPROBE: An experimental system for programming robots by example," *International Journal of Robotics Research*, vol. 3, no. 1, pp. 25–39, 1984.

[33] B. A. MacDonald and D. Pauli, "Adaptive robot training by programming and guiding," *Journal of Intelligent Manufacturing Systems*, vol. 4, pp. 385–404, 1993.

[34] B. A. MacDonald, "Instructable systems," *Knowledge Acquisition*, vol. 3, pp. 381–420, December 1991.

[35] D. C. Yuen and B. A. MacDonald, "An evaluation of sequential monte carlo technique for simultaneous localisation and map-building," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, Taipei, May 2003, to appear.

[36] ——, "Natural landmark based localisation system using panoramic images," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'02)*, vol. 1, Washington DC, May 2002, pp. 915–20.

[37] ——, "A comparison between extended kalman filtering and sequential monte carlo technique for simultaneous localisation and map-building," in *Proc. Australian Conference on Robotics and Automation*, Auckland, New Zealand, November 2002.

[38] ——, "Considerations for the mobile robot implementation of panoramic stereo vision system with a single optical centre," in *Proc. Image and Vision Computing New Zealand (IVCNZ)*, Auckland, New Zealand, November 2002, pp. 335–340.

[39] S. Wong and B. A. MacDonald, "A topological coverage algorithm for mobile robots," in *International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, October 2003, to appear.

[40] B. A. MacDonald and J. Baltes, "Learning, planning and understanding human instructions," in *Proceedings of the Machine Learning Workshop at AI/GI/VI'94*, Banff, Alberta, Canada, May 1994, pp. vii–1–vii–10, available as Research Report No. 94/539/08, Department of Computer Science, University of Calgary, 2500 University Drive NW, Calgary, Alberta, Canada, T2N 1N4.

[41] N. O. Schüler and B. A. MacDonald, "Learning repetition in string transformations," in *Proceedings of the Tenth Canadian Artificial Intelligence conference*. Canadian Society for the Computational Studies of Intelligence, May 1994, pp. 39–46.

[42] J. Baltes and B. A. MacDonald, "A distributed architecture for an instructable problem solver," in *Proceedings of the 27th Hawaii International Conference on Systems Sciences*, Wailea, Maui, 4–7 January 1994, pp. 63–72, emerging Paradigms for Intelligent Systems minitrack in the Decision Support and Knowledge–based Systems Track.

[43] E. Woo, B. A. MacDonald, and F. Trépanier, "Distributed mobile robot application infrastructure," in *International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, October 2003, to appear.

[44] F. Trépanier, "A graphical simulation and visualisation tool for distributed mobile robotics applications," Master's thesis, University of Auckland, New Zealand, 2003.

[45] F. Trépanier, "Developing intelligent robots using computer games," in *Proc. Australian Conference on Robotics and Automation*, Auckland, New Zealand, November 2002, poster. Supervised by Bruce MacDonald.