Arbeitsgruppe Eingebettete
Systeme und Betriebssysteme

Prof. Jörg Kaiser, Sebastian Zug


---

Seminar Mobile Service Robotik (MSR)
Kooperative verteilte Systeme

---

1. Überblick über kooperative Robotikanwendungen [2, 9]
2. (Entwicklungs)Frameworks für verteilte Robotiksysteme
   (a) Übersicht [8]
   (b) ROCI - Framework for perception and control [3]
   (c) OROCOS - Development Framework [1]
3. Anwendungen
   (a) Koordination heterogener Roboter [6]
   (b) Lokalisierung in instrumentierten Umgebungen [7]
   (c) Gemeinsame Umgebungserfassung mittels Virtueller Sensoren [5]
   (d) Lokalisation mittels unterschiedlicher fehlerbehafteter Sensoren [11]
   (e) Kooperative Kartenerstellung [10]
   (f) Interaction fliegender Roboter mit Sensornetzen [4]

## Literatur

[1] H. Bruyninckx. Open robot control software: the OROCOS project. *Language*, 551:14.

[2] Y.U. Cao, A.S. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1):7–27, 1997.

[3] L. Chaimowicz, A. Cowley, V. Sabella, and C.J. Taylor. ROCI: A distributed framework for multi-robot perception and control. In *Proceedings of the 2003 IEEE/RJS International Conference on Intelligent Robots and Systems*, pages 266–271. Citeseer, 2003.

[4] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. Deployment and connectivity repair of a sensor net with a flying robot. *Experimental Robotics IX*, pages 333–343.

[5] R. Grabowski, P. Khosla, and H. Choset. Development and deployment of a line of sight virtual sensor for heterogeneous teams. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 3024–3029. Citeseer, 2004.

[6] L. Iocchi, D. Nardi, M. Piaggio, and A. Sgorbissa. Distributed coordination in heterogeneous multi-robot systems. *Autonomous Robots*, 15(2):155–168, 2003.

[7] B. Jung and G.S. Sukhatme. Cooperative tracking using mobile robots and environment-embedded, networked sensors. In *the 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 206–211. Citeseer, 2001.

[8] J. Kramer and M. Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22(2):101–132, 2007.

[9] L.E. Parker et al. Current state of the art in distributed autonomous mobile robotics. *Distributed Autonomous Robotic Systems*, 4:3–12, 2000.

[10] S. Thayer, B. Digney, M. Diaz, A. Stentz, B. Nabbe, and M. Hebert. Distributed robotic mapping of extreme environments. In *Proceedings of SPIE*, volume 4195. Citeseer, 2000.

[11] R. Tinós, L. Navarro-Serment, and C. Paredis. Fault tolerant localization for teams of distributed robots. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2–1061. Citeseer, 2001.

Proceedings of the 2001 IEEE
International Conference on Robotics & Automation
Seoul, Korea • May 21-26, 2001

# Open Robot Control Software: the OROCOS project

Herman Bruyninckx*
Mechanical Engineering, Katholieke Universiteit Leuven, Belgium
http://www.mech.kuleuven.ac.be/~bruyninc

## Abstract

This paper introduces the OROCOS project (www.orocos.org), that aims at becoming a general-purpose and open robot control software package. OROCOS follows the Open Source development model that has been proven to work in many other general-purpose software packages, such as Linux, Apache, Perl, or LaTeX. The paper focuses on the long-term vision of this start-up project, motivates which strategic and innovative design decisions are to be taken (a CORBA(-like) component architecture being the most important one, [14]), and lists other projects on which OROCOS could build. The success of OROCOS depends critically on how many researchers and engineers can be motivated to contribute code, documentation and feedback to the project.

## 1 Introduction

The robot industry is about half a century old, and has always been keen to adapt standards and high-tech evolutions from electronics and mechanics, especially materials, acutators, and electronic communication. However, the industry's current *software* practices are very similar to those in the computer business some 20 years ago: every manufacturer has its own proprietary software, algorithms, data structures, as well as programming languages. Efforts to define open data format standards or programming languages have had very minor success, such that software exchange possibilities are really extremely low.

There are many reasons for this lack of interoperability and openness in the robotics industry; some of the more important ones are: (i) all manufacturers operate in niche markets (that is, a niche in *scope*, certainly not in *size*!) such as manufacturing or assembly; (ii) the largest share of the market is covered by large corporate customers that make long-term investments and demand high reliability, and not by very price-sensitive individual consumers with rapidly varying tastes; (iii) the academic "customers," interested in the development of robotic systems with a much wider and ambitious scope, represent only negligable market share.

As a result, robotics is still an industry with a very high "user lock-in," leading to the corresponding huge investment threshold for newcomers and low cross-fertilization rate of innovations. This situation has only become worse after the worldwide mergers of the last ten years or so, with only a handful of general purpose manufacturers remaining. On the other hand, many relatively small companies have appeared, which offer a wide range of innovative (but mutually incompatible) robotics products, that are aimed at the (growing) academic, special-purpose or home markets; for example, RWI/iRobot [7, 16], Nomadic [13], Khepera/CyberBotics [8, 27], etc. The repercussions of this commercial situation on academic robotics institutes is that exchange of software research results is almost inexistant, and independent "benchmark" comparisons of results are impossible.

In contrast to what has happened in robotics, the *computer business* scene has gone through a radically different evolution during the last couple of years: the appearance and growing maturity of the *Linux* operating system and other Free Software and Open Source software packages (the GNU tools, Apache, Gnome and KDE, Perl, etc.), together with the massive employment of the Internet and its open communication standards (TCP/IP, HTML, CORBA, etc.), has drastically lowered the monetary threshold for newcomers and consumers, *and* (or rather, *because*!) it has allowed the easy, cheap, instantaneous, and worldwide exchange of software and data.

There is no reason why this "democratization process" of the computer industry could not be applied to the robotics area too. However, until now, no (proprietary or Open) general-purpose robot control software package exists. The goal of this paper is to present such an emerging and ambitious effort, that goes under the name of *OROCOS* (*O*pen *RO*bot *CO*ntrol *S*oftware).

*Postdoctoral Fellow of the F.W.O–Vlaanderen. The financial support by the Belgian State—Prime Minister's Office—Science Policy Programme (IUAP), and by K.U.Leuven's Concerted Research Action GOA/99/04 are gratefully acknowledged.

The following Sections explain the long-term goals of OROCOS, the modern software engineering principles the project will use, the Open Standards with which its software and data formats will work, the business model companies will be able to use to generate an income based on OROCOS (in synergy with the project's Open Source nature!), and the practical, Internet-based day-to-day management and organisation of the project.

## 2 Goals and vision

The OROCOS project is all about developing robot control software

- under Open Source license, [15]. That is, the source code is freely available for study, use and modification. The GPL license (GNU General Public License, [21]) is a major candidate because it has proven to work for Linux. However, it forces all "derived code" to be released under the GPL too. Linux Torvalds solved this problem for the Linux kernel by explicitly allowing proprietary code to be linked to the kernel code; another possibility are less stringent licenses such as the LGPL (Lesser However, OROCOS has a competitive advantage over proprietary libraries (simply because the latter don't exist) so that a more compliant license is not really necessary [22].

- with extreme modularity and flexibility. Such that (i) users can build their own system *à la carte*, and (ii) developers can contribute to the modules they are interested in, without the need to delve into the code of the whole system.

- of the highest quality, from both the technical, documentation, and software engineering points of view. The education and documentation aspects receive special attention within the project: the OROCOS documentation should not be limited to the classical user and reference guides for the software, but, due to its Open Source nature, it can serve as a basis for educational purposes too, e.g., classroom notes illustrated with the examples from the OROCOS code and OROCOS projects. By starting the project with an extensive design brainstorming phase, we want to increase the chances that documentation is available together with the code, or even before it!

- independently of (but if possible compatible with) commercial robot manufacturers. Initially, especially the smaller and/or service-oriented businesses will be interested in using the OROCOS code, and in contributing to it. In the long run, the OROCOS code should become an inevitable *de facto* "Open Standard," comparable to the acceptance of Linux in the mainstream computer industry. One of the major gaps in the current robot industry is the lack of a common programming API (Application Programming Interface), and trajectory generation and motion control libraries; an independent project such as OROCOS is ideal to suggest roadmaps to fill this gap.

- for all sorts of robotic devices and computer platforms. (Note that the large majority of Open Source projects are not at all limited to the Linux platform, but run on Windows, MacOS, Unices, etc.) The largest obstacle in this context are the "off the shelf" robot hardware systems, because their software architecture seldom provides sufficient openness. However, many of those systems have already been "hacked" by the robotics research community, and the results can easily be added as the "Open Hardware" part of OROCOS. As soon as OROCOS proves to be a viable and reliable software project, new robotic systems will adapt it as (one possible) default control environment.

- *localized* for all languages. That means that the software should be straightforward to configure in such a way that all dialogues with users take place in the users' preferred language.

- featuring software *components* for kinematics, dynamics, planning, sensing, control, hardware interfacing, etc. The meaning of "component" in this context corresponds to what is described in so-called "middleware" software (CORBA, DCOM, or RIM), [4], i.e., a software object that can dynamically be added or removed from a network and that offers its services through a neutral, programming language independent interface (such as CORBA's Interface Description Language IDL, [14]).

The project aims at becoming *much more* than just a copy of existing commercial robot controllers or robot simulation/programming packages: these commercial offerings are, up to now, always limited to niches in the market (although these niches often represent billion euros economies!) and are very difficult to extend, to combine, or to run on any hardware or over a network. The difficulties come not only from limits on the scope

of the projects, but often also from the limitations imposed by commercial licenses.

The OROCOS project on the other hand, starts out from the opposite side: it wants to develop *shareable libraries* that are *platform independent*, implement *stand-alone component* examples that can be adapted and extended by end-users, and develop a *configurable run-time environment* from which to simulate and control all possible *distributed robotic systems*. The result should be able to appeal to *all* roboticists, whether they are students interested in extending the soccer server of RoboCup, [17], or a multinational team of engineers from different companies that have to get a five-finger robot hand work together with a redundant manipulator on a planetary rover under vision-guided teleoperation, where all parts come from different manufacturers.

These aims are very ambitious (but realistic), but they are only achievable in an Open Source effort: no proprietary developer can (or wants to, or has the man power to) cover the same broad scope. However, nothing prevents commercial companies from using (part of) the OROCOS software, or even contributing to it. It might be a cliché, but the Linux project is a proven example of a very ambitious undertaking that now receives contributions from both volunteers and professional software engineers, working in isolation or in the framework of multinational corporations.

The goals of this paper are: (i) to introduce the OROCOS project to the public, (ii) to explain the motivations behind its long-term strategy, (iii) to invite colleagues from all over the world to brainstorm about its *design*; and (vi) to raise the interest of "robot hackers" to contribute, review and extend code. Indeed, experience has learned that two requirements should be fulfilled before an Open Source project will succeed: (i) it should start with a *clear vision* of what it want to achieve; and then (ii) it needs a *critical mass of skilled contributors* to make that vision happen. Both requirements are surely fulfilled in robotics! The only(?) problem will be to manage these heterogeneous forces such that they share the same long-term goals.

## 3 Modules

A huge software project such as OROCOS should rely on a "divide and conquer" approach, in order to keep complexity manageable. The OROCOS code (and documentation!) base is to be divided into *modules*, or *libraries*. There are roughly three major types of modules:

- *Supporting modules*. That is, the software without functional robotics contents, but that is nevertheless needed to build a working robot control system. For example, 3D visualisation and simulation; numerical library; software component configuration tools; real-time operating system; inter-process communication; documentation writing tools; etc. These modules can almost completely be "recycled" from already existing Open Source projects, see Sect. 7.

- *Robotics modules*. That is, the software that implements specific robotics algorithms: kinematics and dynamics of (both general and specific) kinematic chains; servo controllers; Baysian and Neural Network estimators; motion planners for mobile robots, serial and parallel manipulators; etc. The Robotics modules make use of one or more of the *Supporting modules*.

- *Components*. That is, the CORBA objects with their IDL descriptions. These components are constructed out of the previous two types of modules, and they are the building blocks with which users "assemble" their (distributed) robot control software environment.

## 4 Software engineering approach

The Robotics and Component modules are the *innovative core* of the OROCOS project. "Innovative core" has two complementary meanings, i.e., in the areas of

1. *Research*: novel ideas and/or algorithms, i.e., not yet available in academic or commercial robot systems. Much novelty also lies in the ambitious *scope* of the applications.

2. *Software engineering*: many algorithms are, in one form or another, already in use in commercial robot systems for many years, but they cannot be accessed for improvement or reuse in other applications. However, pouring them in a sofware module that will function as, for example, a CORBA component, requires creative software engineering, and, in most cases, re-creation of the source code. This is an excellent occasion to apply modern software engineering principles, i.e., object-orientation, software patterns, and distributed components.

The common *design strategy* underlying the development of each OROCOS module is as follows: in

a first step, the long-term vision is openly discussed by means of the OROCOS mailinglist; the developers look for possible *Software Patterns* that are transferable from other domains or have emerged in robotics over the last couple of decades; and, finally, implementation starts and is iterated. Of course, in practice these steps occur most often in parallel. Open Source projects have proven their enormous flexibility in this respect: whenever rational arguments exist to recreate code or redesign parts of the system, there are no marketing or "backwards compatibility" limits to keep the community from doing it. In contrast to the commercial software industry, backwards compatibility is not such a big constraint, because (i) all older versions remain available, and (ii) the cost to upgrade is marginal.

A large component-based software system such as OROCOS will profit maximally from two of the major evolutions in software engineering: object-orientation and software patterns. The latter in particular will get much emphasis, because it is still almost completely absent from the robotics research scene, while a project such as OROCOS is the perfect initiative to fill this gap in an innovative way. The following paragraphs explain the major features of both software engineering concepts, but the interested reader is refered to the abundant literature for more details.

1. *Object orientation.* A large software system is divided into smaller "objects," with well-described *interfaces*, that other objects can use. The objects themselves are responsible for *how* they implement these interfaces.

2. *Software patterns.* The development of a new software system becomes faster and more reliable, whenever one can find a part of the whole system that shows the same behaviour as previously implemented software systems, or as a well-known behaviours from everyday life in human society. Such a recognizable behaviour is called a "*pattern*," [5].

   Well-known examples that are immediately useful in a robotics context are [1, 5]: the *Model–View–Control* pattern for graphical user interfaces; the *Producer–Consumer* model for distributing data over different, dynamically changing clients on a network.

   A software pattern is a set of objects and methods, whose functionality and inter-relationships have been proven to work in common software practice or in real human life. A pattern exists in a possibly very different domain than the one in which the software system is to be implemented. It need even not exist in the form of a software module at all, as long as the expected behaviour of the whole system is intuitively clear to the human developer and/or user, because they recognize it from their everyday experiences. The goal of introducing Software Patterns is to have users recognize these patterns in the system at hand, by giving the pattern a suitable, intuitive name, and then to implement the code in such a way that it satisfies the (often implicit) expectations that users have in the context of the intuitively named pattern. That is, the name of a pattern brings to live a whole *context* within the user's mind, and this context makes it easier to (i) understand the functionality of the different methods implemented in the software that comes with the pattern, and hence (ii) extend and/or maintain the current implementation of the pattern, even for people that were not involved in the original coding.

   Very few of the Patterns that implicitly appear in many robotic systems have already been made explicit, which means that there is still a large unexplored research area to be covered by OROCOS.

## 5 Standards

Open standards (in programming languages, data formats, communication protocols, and documentation) are of paramount importance for efficient sharing of software and for its cooperative development. Only an Open Source project can guarantee the optimal use of, and respect for, open standards; commercial software producers are often tempted to introduce proprietary protocols to keep competitors out of the market.

Here is a list of some of the most important (and most powerful) standards that the OROCOS project will use:

- The CORBA IDL for object embedding, [14]. Through the IIOP (Internet Inter-Orb Protocol), all different implementations of CORBA are interoperable. CORBA also has a real-time extension, which is very important for robot control; TAO, [19] is an open source implementation.
- The structured document language XML [26] for configuration and data files.
- Modelica [12] for modeling of dynamical systems. Most of the commercial packages for the simula-

tion of dynamical systems are co-developing and supporting this modeling format as their future open data format standard for the vendor-neutral exchange of models.

- DocBook [2] and LaTeX[9] for documentation. Both are platform-independent, Open Source, can be translated into many other formats, and as such they are the preferred text processing vehicles of the Linux Documentation Project too. DocBook is ideally suited for software manuals, while LaTeXis best for the advanced mathematics that occur everywhere in robotics.

The OROCOS does not want to impose a "standard" *programming language*. The philosophy behind this strategy is that the best tool for the job should be used. This can be C for real-time control, C++ for numerical algorithms, a scripting language such as Tcl/Tk for graphical interfaces or robot programs, etc. Anyway, the CORBA middleware is, by design, able to work in an heterogeneous programming language environment.

## 6 Business model

The previous Sections could give the false impression that only the robotics researchers in academia are to be interested in OROCOS. This is certainly not true, because all arguments behind proven business models in "classical" Open Source projects hold for OROCOS too:

- Robot service companies and robot users have less costs when they have to learn a "universal" programming paradigm and API. So, the investment threshold for OROCOS is lower than for proprietary alternatives.

- (Especially) small start-up businesses can focus on their core competencies (services or products) without having to invest enormous amounts of money in software.

- The above-mentioned companies (and many others that are mainly *users* of robot software) will pay developers to extend the existing code base with particular drivers, algorithms or features that they need for their own businesses. Through the open source license principle, this code will flow back into the major code tree, and can serve other people's needs too.

- The professional packaging and distribution of software is also a valid business model, as proven by the Red Hats, SuSEs and other players in the Linux market. Also professional documentation is a viable source of income.

- Companies that want to buy service contracts for their robots have more control: they are not "locked-in" by the proprietary software from one particular provider, and they can judge the quality of a given service company from looking at the code this company has already contributed to the project's code tree. In contrast to the commercial software world, Open Source developers are known by their individual names, because these occur in the copyright statements in the code or documentation.

The only "loss" resulting from Open Sourcing a large project is that no company will get tremendously rich from just writing and selling the software...

## 7 Supporting projects

OROCOS is very ambitious in its goals, but it can begin with an enormous head-start: almost all of its *Supporting Modules* (Sect. 3) are already available in other Open Source projects. Indeed, the re-use (of code, as well as of project management tools and experiences) is, almost by definition, the strong point of Open Source. Some (but certainly not all!) of the projects that OROCOS can built on are:

- *Octave*, [3], is a very rich numerical library, that uses the same foundations as, and in its scripting form is very compatible with, Matlab. It also contains ODE and DAE solvers, for the simulation of dynamical systems.

- *Real-Time Linux* [28] and/or *RTAI* [11], which are real-time kernel extensions to Linux; or the Linux-independent eCos [6].

- *Comedi*, [18], is a library for real-time device drivers, such as AD/DA cards.

- Various packages and open formats for 3D visualisation: VRML [25], Java3D [23], OpenGL with OpenInventor [20] or Coin3D [24], etc.

## 8 Organisation

Also for the practical organisation of a large software project such as OROCOS, one can profit maxi-

mally from the experiences and tools originating from other projects. In fact, a certain project management "Pattern" has arisen over the last couple of years, which describes the necessary components needed to successfully run an Open Source project: the project needs a small group of people that outline the strategic *vision*, and it needs a web page (www.orocos.org) for efficient *communication* with the core body of code developers. That web page carries a CVS code repository, one or more mailinglists (together with their archives!), a bug tracking tool (such as Bugzilla or GNATS), and lots of documentation (for example, in the form of an HOWTO guide, as is common practice in the Linux Documentation Project, [10]).

The major responsibilities of the project management team members are *not* in the first place writing code themselves, but rather: being responsive to contributing users; making their valid contributions to the code, the webpage, the documentation, etc., appear on the web page as soon as possible; keeping a low profile in (inevitable) "flame wars" on the mailing list while reminding participants of the common long-term goals; and being pro-active but moderate in the "advocacy" efforts.

## 9 Conclusions

Time has come to start with an Open Source robot control software project: the current commercial culture in robotics is still to "lock-in" users to proprietary software that uses no open software standards to work with code from other sources; the academic (and other!) robotics research community offers enough skilled users, not only to reach the critical mass needed for successful code writing, but also to provide long-term and long-reach design vision; there are many existing Open Source projects on which to build; and, last but not least, the concepts and practices behind running an Open Source project have been proven many times already, with, among many others, the Linux and Apache developments as prime examples.

This paper has introduced and motivated the long-term strategy behind OROCOS, the Open RObot COntrol Software project. This strategy encompasses innovative research subjects, as well as advanced software engineering goals.

## References


[1] Patterns home page. http://hillside.net/patterns/.

[2] DocBook Technical Committee. Docbook. http://www.oasis-open.org/docbook/.

[3] J. W. Eaton. GNU Octave. http://www.che.wisc.edu/octave/.

[4] W. Emmerich. *Engineering distributed objects*. Wiley Chichester, 2000.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, Reading, MA, 1995.

[6] R. Hat. Embedded Configurable Operating System. http://sources.redhat.com/ecos/.

[7] iRobot. http://www.irobot.com/ir/index.htm.

[8] K-Team. http://www.k-team.com/.

[9] L. Lamport. LaTeX: A document preparation system. http://www.latex-project.org/.

[10] D. S. Lawyer. Linux documentation project. http://www.linuxdoc.org/.

[11] P. Mantegazza. RTAI: the Real-Time Applications Interface. http://server.aero.polimi.it/projects/rtai/.

[12] Modelica Association. Modelica: Language design for multi-domain modeling. http://www.modelica.org/.

[13] Nomadic Technologies. http://www.robots.com/.

[14] Open Management Group. CORBA: Common Object Request Broker Architecture. http://www.corba.org/.

[15] OpenSource.org. The open source page. http://www.opensource.org/.

[16] Real World Interface. http://www.irobot.com/rwi/.

[17] RoboCup. Soccer server. http://ci.etl.go.jp/~noda/soccer/server/.

[18] D. Schleef. Comedi: Linux control and measurement device interface. http://stm.lbl.gov/comedi/.

[19] D. C. Schmidt. TAO, The Ace Orb. http://www.cs.wustl.edu/~schmidt/TAO.html.

[20] SGI. Open inventor. http://oss.sgi.com/projects/inventor/.

[21] R. M. Stallman. GNU Public Licence. http://www.fsf.org/copyleft/gpl.html.

[22] R. M. Stallman. Why you shouldn't use the Library GPL for your next library. http://www.fsf.org/philosophy/why-not-lgpl.html.

[23] Sun Microsystems. Java3D. http://java.sun.com/products/java-media/3D/.

[24] Systems in Motion AS. Coin3D. http://www.coin3d.org/.

[25] The Web3D Consortium. The VRML repository. http://www.web3d.org/vrml/vrml.htm.

[26] W3C. XML: Extensible Markup Language. http://www.w3.org/XML/.

[27] Webots. http://www.cyberbotics.com/.

[28] V. Yodaiken and M. Barabanov. Real-time linux. http://www.rtlinux.org.

# Cooperative Mobile Robotics: Antecedents and Directions*

Y. Uny Cao, Alex S. Fukunaga, and Andrew B. Kahng

UCLA Computer Science Department, Los Angeles, CA 90024-1596

## Abstract

There has been increased research interest in systems composed of multiple autonomous mobile robots exhibiting cooperative behavior. Groups of mobile robots are constructed, with an aim to studying such issues as group architecture, resource conflict, origin of cooperation, learning, and geometric problems. As yet, few applications of cooperative robotics have been reported, and supporting theory is still in its formative stages. In this paper, we give a critical survey of existing works and discuss open problems in this field, emphasizing the various theoretical issues that arise in the study of cooperative robotics. We describe the intellectual heritages that have guided early research, as well as possible additions to the set of existing motivations.

## 1 Preliminaries

There has been much recent activity toward achieving systems of multiple mobile robots engaged in collective behavior. Such systems are of interest for several reasons:

- tasks may be inherently too complex for a single robot to accomplish, or performance benefits can be gained from using multiple robots;
- building and using several simple robots can be easier, cheaper, more flexible and more fault-tolerant than having a single powerful robot for each separate task; and
- the constructive, synthetic approach inherent in cooperative mobile robotics can possibly yield insights into fundamental problems in the social sciences (organization theory, economics, cognitive psychology), and life sciences (theoretical biology, animal ethology).

The study of multiple-robot systems naturally extends research on single-robot systems, but is also a discipline unto itself: multiple-robot systems can accomplish tasks that no single robot can accomplish, since ultimately a single robot, no matter how capable, is spatially limited. Multiple-robot systems are

*This is an expanded version of a paper which originally appeared in the proceedings of the 1995 IEEE/RSJ IROS conference.

also different from other distributed systems because of their implicit "real-world" environment, which is presumably more difficult to model and reason about than traditional components of distributed system environments (i.e., computers, databases, networks).

The term *collective behavior* generically denotes any behavior of agents in a system having more than one agent. *Cooperative behavior*, which is the subject of the present survey, is a subclass of collective behavior that is characterized by cooperation. Webster's dictionary [Merriam-Webster, 1963] defines "cooperate" as "to associate with another or others for mutual, often economic, benefit". Explicit definitions of cooperation in the robotics literature, while surprisingly sparse, include:

1. "joint collaborative behavior that is directed toward some goal in which there is a common interest or reward" [Barnes and Gray, 1991];
2. "a form of interaction, usually based on communication" [Mataric, 1994a]; and
3. "[joining] together for doing something that creates a progressive result such as increasing performance or saving time" [Premvuti and Yuta, 1990].

These definitions show the wide range of possible motivating perspectives. For example, definitions such as (1) typically lead to the study of task decomposition, task allocation, and other distributed artificial intelligence (DAI) issues (e.g., learning, rationality). Definitions along the lines of (2) reflect a concern with requirements for information or other resources, and may be accompanied by studies of related issues such as correctness and fault-tolerance. Finally, definition (3) reflects a concern with quantified measures of cooperation, such as speedup in time to complete a task. Thus, in these definitions we see three fundamental seeds: the *task*, the *mechanism* of cooperation, and system *performance*.

We define cooperative behavior as follows: *Given some task specified by a designer, a multiple-robot system displays* cooperative behavior *if, due to some underlying mechanism (i.e., the "mechanism of cooperation"), there is an increase in the total utility of the system.* Intuitively, cooperative behavior entails some type of performance gain over naive collective behavior. The mechanism of cooperation may lie in the imposition by the designer of a control or communication structure, in aspects of the task specification, in the interaction dynamics of agent behaviors, etc.

In this paper, we survey the intellectual heritage and major research directions of the field of cooperative robotics. For this survey of cooperative robotics to remain tractable, we restrict our discussion to works involving mobile robots or *simulations* of mobile robots, where a *mobile robot* is taken to be an autonomous, physically independent, mobile robot. In particular, we concentrated on fundamental *theoretical* issues that impinge on cooperative robotics. Thus, the following related

subjects were outside the scope of this work:

- coordination of multiple manipulators, articulated arms, or multi-fingered hands, etc.
- *human-robot cooperative systems*, and user-interface issues that arise with multiple-robot systems [Yokota *et al.*, 1994] [Arkin and Ali, 1994] [Noreils and Recherche, 1991] [Adams *et al.*, 1995].
- the *competitive* subclass of collective behavior, which includes pursuit-evasion [Reynolds, 1994, Miller and Cliff, 1994] and one-on-one competitive games [Asada *et al.*, 1994]. Note that a cooperative team strategy for, e.g., work on the robot soccer league recently started in Japan [Kitano, 1994] would lie within our present scope.
- emerging technologies such as nanotechnology [Drexler, 1992] and Micro Electro-Mechanical Systems [Mehregany *et al.*, 1988] that are likely to be very important to cooperative robotics are beyond the scope of this paper.

Even with these restrictions, we find that over the past 8 years (1987-1995) alone, well over 200 papers have been published in this field of cooperative (mobile) robotics, encompassing theories from such diverse disciplines as artificial intelligence, game theory/economics, theoretical biology, distributed computing/control, animal ethology and artificial life.

We are aware of two previous works that have surveyed or taxonomized the literature. [Asama, 1992] is a broad, relatively succinct survey whose scope encompasses *distributed autonomous robotic systems* (i.e., not restricted to mobile robots). [Dudek *et al.*, 1993] focuses on several well-known "swarm" architectures (e.g., SWARM and Mataric's Behavior-based architecture – see Section 2.1.5) and proposes a taxonomy to characterize these architectures. The scope and intent of our work differs significantly from these, in that (1) we extensively survey the field of cooperative mobile robotics, and (2) we provide a taxonomical organization of the literature based on *problems* and *solutions* that have arisen in the field (as opposed to a selected group of architectures). In addition, we survey much new material that has appeared since these earlier works were published.

## Towards a Picture of Cooperative Robotics

In the mid-1940's Grey Walter, along with Wiener and Shannon, studied turtle-like robots equipped with light and touch sensors; these simple robots exhibited "complex social behavior" in responding to each other's movements [Dorf, 1990]. Coordination and interactions of multiple intelligent agents have been actively studied in the field of distributed artificial intelligence (DAI) since the early 1970's [Bond and Gasser, 1988], but the DAI field concerned itself mainly with problems involving software agents. In the late 1980's, the robotics research community became very active in cooperative robotics,

beginning with projects such as CEBOT [Fukuda and Nakagawa, 1987], SWARM [Beni, 1988], ACTRESS [Asama *et al.*, 1989], GOFER [Caloud *et al.*, 1990], and the work at Brussels [Steels, 1990]. These early projects were done primarily in simulation, and, while the early work on CEBOT, ACTRESS and GOFER have all had physical implementations (with $\leq 3$ robots), in some sense these implementations were presented by way of proving the simulation results. Thus, several more recent works (cf. [Kube and Zhang, 1992, Mataric, 1992b, Parker, 1992]) are significant for establishing an emphasis on the actual physical implementation of cooperative robotic systems. Many of the recent cooperative robotic systems, in contrast to the earlier works, are based on a behavior-based approach (cf. [Brooks, 1986]). Various perspectives on autonomy and on the connection between intelligence and environment are strongly associated with the behavior-based approach [Brooks, 1991], but are not intrinsic to multiple-robot systems and thus lie beyond our present scope. Also note that a recent incarnation of CEBOT, which has been implemented on physical robots, is based on a behavior-based control architecture [Cai *et al.*, 1995].

The rapid progress of cooperative robotics since the late 1980's has been an interplay of *systems*, *theories* and *problems*: to solve a given problem, systems are envisioned, simulated and built; theories of cooperation are brought from other fields; and new problems are identified (prompting further systems and theories). Since so much of this progress is recent, it is not easy to discern deep intellectual heritages from within the field. More apparent are the intellectual heritages from other fields, as well as the canonical task domains which have driven research. Three examples of the latter are:

- **Traffic Control.** When multiple agents move within a common environment, they typically attempt to avoid collisions. Fundamentally, this may be viewed as a problem of *resource conflict*, which may be resolved by introducing, e.g., traffic rules, priorities, or communication architectures. From another perspective, path planning must be performed taking into consideration other robots and the global environment; this multiple-robot path planning is an intrinsically *geometric* problem in configuration space-time. Note that prioritization and communication protocols – as well as the internal modeling of other robots – all reflect possible variants of the *group architecture* of the robots. For example, traffic rules are commonly used to reduce planning cost for avoiding collision and deadlock in a real-world environment, such as a network of roads. (Interestingly, behavior-based approaches identify collision avoidance as one of the most basic behaviors [Brooks, 1986], and achieving a collision-avoidance behavior is the natural solution to collision avoidance among multiple robots. However, in reported experiments that use the behavior-based approach, robots are never restricted to road networks.)

- **Box-Pushing/Cooperative Manipulation.** Many works have addressed the box-pushing (or couch-pushing) problem, for widely varying reasons. The focus in [Parker, 1994b] is on

task allocation, fault-tolerance and (reinforcement) *learning*. By contrast, [Donald *et al.*, 1994] studies two box-pushing protocols in terms of their intrinsic communication and hardware requirements, via the concept of information invariants. Cooperative manipulation of large objects is particularly interesting in that cooperation can be achieved without the robots even knowing of each others' existence [Sen *et al.*, 1994, Tung and Kleinrock, 1993]. Other works in the class of box-pushing/object manipulation include [Wang *et al.*, 1994] [Stilwell and Bay, 1993] [Johnson and Bay, 1994] [Brown and Jennings, 1995] [Kube and Zhang, 1992] [Kube *et al.*, 1993] [Kube and Zhang, 1993] [Mataric *et al.*, 1995] [Rus *et al.*, 1995] [Hara *et al.*, 1995] [Sasaki *et al.*, 1995].

- **Foraging.** In foraging, a group of robots must pick up objects scattered in the environment; this is evocative of toxic waste cleanup, harvesting, search and rescue, etc. The foraging task is one of the canonical testbeds for cooperative robotics [Brooks *et al.*, 1990] [Steels, 1990] [Arkin, 1992] [Goss and Deneubourg, 1992] [Lewis and Bekey, 1992] [Drgoul and Ferber, 1993] [Mataric, 1994a] [Arkin and Hobbs, 1993] [Beckers *et al.*, 1994]. The task is interesting because (1) it can be performed by each robot independently (i.e., the issue is whether multiple robots achieve a performance gain), and (2) as discussed in Section 3.2, the task is also interesting due to motivations related to the biological inspirations behind cooperative robot systems. There are some conceptual overlaps with the related task of materials handling in a manufacturing workcell [Doty and Aken, 1993]. A wide variety of techniques have been applied, ranging from simple stigmergy (essentially random movements that result in the fortuitous collection of objects [Beckers *et al.*, 1994] to more complex algorithms in which robots form chains along which objects are passed to the goal [Drgoul and Ferber, 1993]. [Beckers *et al.*, 1994] defines stigmergy as "the production of a certain behaviour in agents as a consequence of the effects produced in the local environment by previous behaviour". This is actually a form of "cooperation without communication", which has been the stated object of several foraging solutions since the corresponding formulations become nearly trivial if communication is used. On the other hand, that stigmergy may not satisfy our definition of cooperation given above, since there is no performance improvement over the "naive algorithm" – in this particular case, the proposed stigmergic algorithm *is* the naive algorithm. Again, group architecture and learning are major research themes in addressing this problem.

Other interesting task domains that have received attention in the literature include multi-robot security systems [Everett *et al.*, 1993], landmine detection and clearance [Franklin *et al.*, 1995], robotic structural support systems (i.e., keeping structures stable in case of, say, an earthquake) [Ma *et al.*, 1994], map making [Singh and Fujimura, 1993], and assembly of objects using multiple robots [Wang *et al.*, 1994].

### Organization of Paper

With respect to our above definition of cooperative behavior, we find that the great majority of the cooperative robotics literature centers on the *mechanism* of cooperation (i.e., few works study a task without also claiming some novel approach to achieving cooperation). Thus, our study has led to the synthesis of five "Research Axes" which we believe comprise the major themes of investigation to date into the underlying mechanism of cooperation.

Section 2 of this paper describes these axes, which are: 2.1 Group Architecture, 2.2 Resource Conflict, 2.3 Origin of Cooperation, 2.4 Learning, and 2.5 Geometric Problems. In Section 3, we present more synthetic reviews of cooperative robotics: Section 3.1 discusses constraints arising from technological limitations; and Section 3.2 discusses possible lacunae in existing work (e.g., formalisms for measuring performance of a cooperative robot system), then reviews three fields which we believe must strongly influence future work. We conclude in Section 4 with a list of key research challenges facing the field.

## 2 Research Axes

Seeking a *mechanism* of cooperation may be rephrased as the "cooperative behavior design problem": *Given a group of robots, an environment, and a task, how should cooperative behavior arise?* In some sense, every work in cooperative robotics has addressed facets of this problem, and the major research axes of the field follow from elements of this problem. (Note that certain basic robot interactions are not task-performing interactions per se, but are rather basic primitives upon which task-performing interactions can be built, e.g., following ([Connell, 1987, Donald *et al.*, 1994] and many others) or flocking [Reynolds, 1987, Mataric, 1994a]. It might be argued that these interactions entail "control and coordination" tasks rather than "cooperation" tasks, but our treatment does not make such a distinction).

First, the realization of cooperative behavior must rely on some infrastructure, the **group architecture**. This encompasses such concepts as robot heterogeneity/homogeneity, the ability of a given robot to recognize and model other robots, and communication structure. Second, for multiple robots to inhabit a shared environment, manipulate objects in the environment, and possibly communicate with each other, a mechanism is needed to resolve **resource conflicts**. The third research axis, **origins of cooperation**, refers to how cooperative behavior is actually motivated and achieved. Here, we do not discuss instances where cooperation has been "explicitly engineered" into the robots' behavior since this is the default approach. Instead, we are more interested in biological parallels (e.g., to social insect behavior), game-theoretic justifications for cooperation, and concepts of emergence. Because

adaptability and flexibility are essential traits in a task-solving group of robots, we view **learning** as a fourth key to achieving cooperative behavior. One important mechanism in generating cooperation, namely, task decomposition and allocation, is *not* considered a research axis since (i) very few works in cooperative robotics have centered on task decomposition and allocation (with the notable exceptions of [Noreils, 1993, Lueth and Laengle, 1994, Parker, 1994b]), (ii) cooperative robot tasks (foraging, box-pushing) in the literature are simple enough that decomposition and allocation are not required in the solution, and (iii) the use of decomposition and allocation depends almost entirely on the group architectures (e.g. whether it is centralized or decentralized). Note that there is also a related, geometric problem of optimizing the allocation of tasks spatially. This has been recently studied in the context of the division of the search of a work area by multiple robots [Kurabayashi *et al.*, 1995]. Whereas the first four axes are related to the *generation* of cooperative behavior, our fifth and final axis – **geometric problems** – covers research issues that are tied to the embedding of robot tasks in a two- or three-dimensional world. These issues include multi-agent path planning, moving to formation, and pattern generation.

## 2.1 Group Architecture

The *architecture* of a computing system has been defined as "the part of the system that remains unchanged unless an external agent changes it" [VanLehn, 1991]. The *group architecture* of a cooperative robotic system provides the infrastructure upon which collective behaviors are implemented, and determines the capabilities and limitations of the system. We now briefly discuss some of the key architectural features of a group architecture for mobile robots: centralization/decentralization, differentiation, communications, and the ability to model other agents. We then describe several representative systems that have addressed these specific problems.

### 2.1.1 Centralization/Decentralization

The most fundamental decision that is made when defining a group architecture is whether the system is centralized or decentralized, and if it is decentralized, whether the system is hierarchical or distributed. *Centralized* architectures are characterized by a single control agent. *Decentralized* architectures lack such an agent. There are two types of decentralized architectures: *distributed* architectures in which all agents are equal with respect to control, and *hierarchical* architectures which are locally centralized. Currently, the dominant paradigm is the decentralized approach.

The behavior of decentralized systems is often described using such terms as "emergence" and "self-organization." It is widely claimed that decentralized architectures (e.g., [Beckers *et al.*, 1994,

Arkin, 1992, Steels, 1994, Mataric, 1994a]) have several inherent advantages over centralized architectures, including fault tolerance, natural exploitation of parallelism, reliability, and scalability. However, we are not aware of any published empirical or theoretical comparison that supports these claims directly. Such a comparison would be interesting, particularly in scenarios where the team of robots is relatively small (e.g., two robots pushing a box), and it is not clear whether the scaling properties of decentralization offset the coordinative advantage of centralized systems.

In practice, many systems do not conform to a strict centralized/decentralized dichotomy, e.g., many largely decentralized architectures utilize "leader" agents. We are not aware of any instances of systems that are completely centralized, although there are some hybrid centralized/decentralized architectures wherein there is a central planner that exerts high-level control over mostly autonomous agents [Noreils, 1993, Lueth and Laengle, 1994, Aguilar *et al.*, 1995, Causse and Pampagnin, 1995].

### 2.1.2 Differentiation

We define a group of robots to be *homogeneous* if the capabilities of the individual robots are identical, and *heterogeneous* otherwise. In general, heterogeneity introduces complexity since task allocation becomes more difficult, and agents have a greater need to model other individuals in the group. [Parker, 1994b] has introduced the concept of *task coverage*, which measures the ability of a given team member to achieve a given task. This parameter is an index of the demand for cooperation: when task coverage is high, tasks can be accomplished without much cooperation, but otherwise, cooperation is necessary. Task coverage is maximal in homogeneous groups, and decreases as groups become more heterogeneous (i.e., in the limit only one agent in the group can perform any given task).

The literature is currently dominated by works that assume homogeneous groups of robots. However, some notable architectures can handle heterogeneity, e.g., ACTRESS and ALLIANCE (see Section 2.1.5 below). In heterogeneous groups, task allocation may be determined by individual capabilities, but in homogeneous systems, agents may need to *differentiate* into distinct roles that are either known at design-time, or arise dynamically at run-time.

### 2.1.3 Communication Structures

The communication structure of a group determines the possible modes of inter-agent interaction. We characterize three major types of interactions that can be supported. ([Dudek *et al.*, 1993] proposes a more detailed taxonomy of communication structures).

**Interaction via environment**

The simplest, most limited type of interaction occurs when the environment itself is the communication medium (in effect, a shared memory), and there is no explicit communication or interaction between agents. This modality has also been called "cooperation without communication" by some researchers. Systems that depend on this form of interaction include [Goss and Deneubourg, 1992, Beckers *et al.*, 1994, Arkin, 1992, Steels, 1990, Tung and Kleinrock, 1993, Tung, 1994, Sen *et al.*, 1994].

**Interaction via sensing**

Corresponding to arms-length relationships in organization theory [Hewitt, 1993], interaction via sensing refers to local interactions that occur between agents as a result of agents sensing one another, but without explicit communication. This type of interaction requires the ability of agents to distinguish between other agents in the group and other objects in the environment, which is called "kin recognition" in some literatures [Mataric, 1994a]. Interaction via sensing is indispensable for modeling of other agents (see Section 2.1.4 below). Because of hardware limitations, interaction via sensing has often been emulated using radio or infrared communications. However, several recent works attempt to implement true interaction via sensing, based on vision [Kuniyoshi *et al.*, 1994a, Kuniyoshi *et al.*, 1994b, Sugie *et al.*, 1995]. Collective behaviors that can use this kind of interaction include flocking and pattern formation (keeping in formation with nearest neighbors).

**Interaction via communications**

The third form of interaction involves explicit communication with other agents, by either directed or broadcast intentional messages (i.e. the recipient(s) of the message may be either known or unknown). Because architectures that enable this form of communication are similar to communication networks, many standard issues from the field of networks arise, including the design of network topologies and communications protocols. For example, in [Wang, 1994] a media access protocol (similar to that of Ethernet) is used for inter-robot communication. In [Ichikawa *et al.*, 1993], robots with limited communication range communicate to each other using the "hello-call" protocol, by which they establish "chains" in order to extend their effective communication ranges. [Gage, 1993] describes methods for communicating to many ("zillions") robots, including a variety of schemes ranging from broadcast channels (where a message is sent to all other robots in the system) to modulated retroreflection (where a master sends out a laser signal to slaves and interprets the response by the nature of the reflection). [Wang *et al.*, 1995] describes and simulates a wireless CSMA/CD ( Carrier Sense Multiple Access with Collision Detection ) protocol for the distributed robotic systems.

There are also communication mechanisms designed specially for multiple-robot systems. For example, [Wang and Beni, 1990] proposes the "sign-board" as a communication mechanism for distributed robotic systems. [Arai *et al.*, 1993] gives a communication protocol modeled after diffusion, wherein

local communication similar to chemical communication mechanisms in animals is used. The communication is engineered to decay away at a preset rate. Similar communications mechanisms are studied in [Lewis and Bekey, 1992, Drgoul and Ferber, 1993, Goss and Deneubourg, 1992]. Additional work on communication can be found in [Yoshida *et al.*, 1994], which analyzes optimal group sizes for local communications and communication delays. In a related vein, [Yoshida *et al.*, 1995a, Yoshida *et al.*, 1995b] analyzes optimal local communication ranges in broadcast communication.

### 2.1.4 Modeling of Other Agents

Modeling the intentions, beliefs, actions, capabilities, and states of other agents can lead to more effective cooperation between robots. Communications requirements can also be lowered if each agent has the capability to model other agents. Note that the modeling of other agents entails more than implicit communication via the environment or perception: modeling requires that the modeler has some representation of another agent, and that this representation can be used to make inferences about the actions of the other agent.

In cooperative robotics, agent modeling has been explored most extensively in the context of manipulating a large object. Many solutions have exploited the fact that the object can serve as a common medium by which the agents can model each other.

The second of two box-pushing protocols in [Donald *et al.*, 1994] can achieve "cooperation without communication" since the object being manipulated also functions as a "communication channel" that is shared by the robot agents; other works capitalize on the same concept to derive distributed control laws which rely only on local measures of force, torque, orientation, or distance, i.e., no explicit communication is necessary (cf. [Stilwell and Bay, 1993] [Hashimoto and Oba, 1993]). In a two-robot bar carrying task, Fukuda and Sekiyama's agents [Fukuda and Sekiyama, 1994] each uses a probabilistic model of the other agent. When a risk threshold is exceeded, an agent communicates with its partner to maintain coordination. In [Donald, 1993a, Donald, 1993b], the theory of information invariants is used to show that extra hardware capabilities can be added in order to infer the actions of the other agent, thus reducing communication requirements. This is in contrast to [Sen *et al.*, 1994], where the robots achieve box pushing but are not aware of each other at all. For a more complex task involving the placement of five desks in [Sugie *et al.*, 1995], a homogeneous group of four robots share a ceiling camera to get positional information, but do not communicate with each other. Each robot relies on modeling of other agents to detect conflicts of paths and placements of desks, and to change plans accordingly.

### 2.1.5 Representative Architectures

All systems implement some group architecture. We now describe several particularly well-defined representative architectures, along with works done within each of their frameworks. It is interesting to note that these architectures encompass the entire spectrum from traditional AI to highly decentralized approaches.

**CEBOT**

CEBOT (CEllular roBOTics System) is a decentralized, hierarchical architecture inspired by the cellular organization of biological entities (cf. [Fukuda and Nakagawa, 1987] [Fukuda and Kawauchi, 1993], [Ueyama and Fukuda, 1993b] [Ueyama and Fukuda, 1993a] [Fukuda and Iritani, 1995]). The system is dynamically reconfigurable in that basic autonomous "cells" (robots), which can be physically coupled to other cells, dynamically reconfigure their structure to an "optimal" configuration in response to changing environments. In the CEBOT hierarchy there are "master cells" that coordinate subtasks and communicate with other master cells. A solution to the problem of electing these master cells was discussed in [Ueyama *et al.*, 1993b]. Formation of structured cellular modules from a population of initially separated cells was studied in [Ueyama and Fukuda, 1993b]. Communications requirements have been studied extensively with respect to the CEBOT architecture, and various methods have been proposed that seek to reduce communication requirements by making individual cells more intelligent (e.g., enabling them to model the behavior of other cells). [Fukuda and Sekiyama, 1994] studies the problem of modeling the behavior of other cells, while [Kawauchi *et al.*, 1993a, Kawauchi *et al.*, 1993b] present a control method that calculates the goal of a cell based on its previous goal and on its master's goal. [Fukuda *et al.*, 1990] gives a means of estimating the amount of information exchanged between cells, and [Ueyama *et al.*, 1993a] gives a heuristic for finding master cells for a binary communication tree. A new behavior selection mechanism is introduced in [Cai *et al.*, 1995], based on two matrices, the priority matrix and the interest relation matrix, with a learning algorithm used to adjust the priority matrix. Recently, a Micro Autonomous Robotic System (MARS) has been built consisting of robots of 20 cubic mm and equipped with infrared communications [Mitsumoto *et al.*, 1995].

**ACTRESS**

The ACTRESS (ACTor-based Robot and Equipments Synthetic System) project [Asama *et al.*, 1989] [Ishida *et al.*, 1991] [Asama *et al.*, 1992] is inspired by the Universal Modular ACTOR Formalism [Hewitt *et al.*, 1973]. In the ACTRESS system, "robotors", including 3 robots and 3 workstations (one as interface to human operator, one as image processor and one as global environment manager), form a heterogeneous group trying to perform tasks such as object pushing [Asama *et al.*, 1991a] that cannot be accomplished by any of the individual robotors alone [Ishida *et al.*, 1994, Suzuki *et al.*, 1995].

Communication protocols at different abstraction levels [Matsumoto *et al.*, 1990] provide a means upon which "group cast" and negotiation mechanisms based on Contract Net [Smith, 1980] and multi-stage negotiation protocols are built [Asama *et al.*, 1994]. Various issues are studied, such as efficient communications between robots and environment managers [Asama *et al.*, 1991b], collision avoidance [Asama *et al.*, 1991c].

**SWARM**

A *SWARM* is a distributed system with a large number of autonomous robots [Jin *et al.*, 1994]. (Note that the work on SWARM systems began as work on Cellular Robotic Systems [Beni, 1988], where many simple agents occupied one- or two-dimensional environments and were able to perform tasks such as pattern generation and self-organization). *SWARM intelligence* is "a property of systems of non-intelligent robots exhibiting collectively intelligent behavior" [Hackwood and Beni, 1991]. *Self-organization* in a SWARM is the ability to distribute itself "optimally" for a given task, e.g., via geometric pattern formation or structural organization. SWARM exhibits a distributed architecture, usually with no differentiation among members (an exception is [Hackwood and Beni, 1992], where two different types of robots were used). Interaction takes place by each cell reacting to the state of its nearest neighbors. Mechanisms for self-organization in SWARM are studied in [Hackwood and Beni, 1992] [Beni and Wang, 1991] [Beni and Hackwood, 1992] [Hackwood and Beni, 1991] [Liang and Beni, 1995] [Jin *et al.*, 1994]. Examples for possible applications include large-scale displays and distributed sensing [Hackwood and Wang, 1988]. Communication primitives have been an important part of research in SWARM [Wang and Beni, 1988, Wang and Premvuti, 1995] (see Section 3.2 below for more details).

**GOFER**

The GOFER architecture [Caloud *et al.*, 1990, LePape, 1990] was used to study distributed problem solving by multiple mobile robots in an indoor environment using traditional AI techniques. In GOFER, a central task planning and scheduling system (CTPS) communicates with all robots and has a global view of both the tasks to be performed and the availability of robots to perform the tasks. The CTPS generates a plan structure (template for an instance of a plan) and informs all available robots of the pending goals and plan structures. Robots use a task allocation algorithm like the Contract Net Protocol [Smith, 1980] to determine their roles. Given the goals assigned during the task allocation process, they attempt to achieve their goals using fairly standard AI planning techniques. The GOFER architecture was successfully used with three physical robots for tasks such as following, box-pushing, and wall tracking in a corridor.

Other architectures that make significant use of concepts studied within the classical distributed paradigm are described in [Lueth and Laengle, 1994] [Noreils, 1990] [Noreils, 1993] [Noreils, 1992a]

[Noreils, 1992b] [Alani *et al.*, 1995] [Albus, 1993].

#### ALLIANCE/L-ALLIANCE

The ALLIANCE architecture was developed by Parker [Parker, 1994b, Parker, 1994a] in order to study cooperation in a heterogeneous, small-to-medium-sized team of largely independent, loosely coupled robots. Robots are assumed able to, with some probability, sense the effects of their own actions and the actions of other agents through perception and explicit broadcast communications. Individual robots are based on a behavior-based controller with an extension for activating "behavior sets" that accomplish certain tasks. These sets are activated by motivational behaviors whose activations are in turn determined by the robots' awareness of their teammates. L-ALLIANCE [Parker, 1994b] is an extension to ALLIANCE that uses reinforcement learning to adjust the parameters controlling behavior set activation. The ALLIANCE/L-ALLIANCE architecture has been implemented both on real robots and in simulation, and has been successfully demonstrated for tasks including box-pushing, puck-gathering, marching in formation, and simulations of hazardous waste cleanup and janitorial service.

#### Behavior-Based Cooperative Behavior

Mataric [Mataric, 1992c, Mataric, 1992a, Mataric, 1993, Mataric, 1994a] proposes a behavior-based architecture for the synthesis of collective behaviors such as flocking, foraging, and docking based on the direct and temporal composition of primitive basic behaviors (safe-wandering, following, aggregation, dispersion, homing). A method for automatically constructing composite behaviors based on reinforcement learning is also proposed. The architecture has been implemented both on groups of up to 20 real robots (the largest group reported in the works we surveyed) and in simulation.

Similar behavior-based architectures include the work by Kube et al, which is based on an Adaptive Logic Network, a neural network [Kube and Zhang, 1992, Kube and Zhang, 1993, Kube *et al.*, 1993, Kube and Zhang, 1994], the Tropism-Based Cognitive Architecture [Agah and Bekey, 1994], and an architecture based on "instinctive behaviors" [Dario *et al.*, 1991].

### 2.2 Resource Conflict

When a single indivisible resource is requested by multiple robots, *resource conflict* arises. This issue has been studied in many guises, notably the mutual exclusion problem in distributed algorithms and the multiaccess problem in computer networks. With multiple robots, resource conflict occurs when there is a need to share space, manipulable objects or communication media. Few works have dealt specifically with object sharing or sharing of communication media (i.e., sharing of communication media is usually achieved using very basic techniques – wireless LAN, straightforward time-division multi-

plexing, or broadcast over an RF channel; recently, [Wang and Premvuti, 1994, Yoshida *et al.*, 1995b, Yoshida *et al.*, 1995a] have considered some problems in sharing communications channels). We therefore center on the space sharing problem, which has been studied primarily via multiple-robot path planning (the "traffic control" formulation from above) and the collision and deadlock avoidance problems.

In a multi-robot system, each robot can conceivably plan a path that accounts for other robots and the global environment via configuration space-time, explicit models of other agents, or other techniques For example, [Fukuda and Sekiyama, 1994] proposes a "hierarchical prediction model" which essentially uses simulation to achieve collision avoidance. [Rude, 1994] considers the problem of crossing an intersection: event transforms into the local space-time coordinate frame of a robot are applied, and each robot (i) iteratively updates the local frame and its objects, (ii) evaluates collision risk, and (iii) generates a modified path depending on the collision risk. (See also Section 2.5). However, researchers considering real-world multi-robot systems typically conclude that planning paths in advance is impossible. Thus, robots are often restricted to prescribed paths or roads, with rules (much like traffic laws in the human world) and communications used to avoid collision and deadlock [Caloud *et al.*, 1990, Asama *et al.*, 1991a].

Grossman [Grossman, 1988] classifies instances of the traffic control problem into three types: (i) restricted roads, (ii) multiple possible roads with robots selecting autonomously between them, and (iii) multiple possible roads with centralized traffic control. When individual robots possess unique roads from one point to another, no conflict is possible; when there is global knowledge and centralized control, it is easy to prevent conflict. Thus, the interesting case is (ii), where robots are allowed to autonomously select roads. Analysis in [Grossman, 1988] shows that restricted roads are highly suboptimal, and that the autonomous road choice coupled with a greedy policy for escaping blocked situations is far more effective (cf. "modest cooperation" [Premvuti and Yuta, 1990], where robots are assumed to be benevolent for the common good of the system).

Solutions to the traffic control problem range from rule-based solutions to approaches with antecedents in distributed processing. In [Kato *et al.*, 1992], robots follow pre-planned paths and use rules for collision avoidance. Example rules include "keep-right", "stop at intersection", and "keep sufficient space to the robot in front of you". [Asama *et al.*, 1991c] solves collision avoidance using two simple rules and a communication protocol that resolves conflict by transmitting individual priorities based on the task requirement, the environment, and the robot performance. In [Yuta and Premvuti, 1992], the robots stop at an intersection and indicate both the number of robots at the intersection and the directions in which they are traveling. If deadlock is possible, each robot performs "shunting" (trying to obtain high priority) and proceeds according to the agreed-upon priorities. [Wang, 1991] takes a

distributed computing approach to traffic control, where the particular problem solved is to keep the number of robots traveling on any given path below a threshold value. Robots use a mutual exclusion protocol to compete for the right to travel on each path. Wang and Beni [Wang and Beni, 1990] adapt two distributed algorithms to solve two problems in their CRS/SWARM architecture. The "$n$-way intersection problem" is solved using an algorithm similar to mutual exclusion, and the "knot detection problem" is solved using an algorithm similar to distributed deadlock detection.

## 2.3 The Origin of Cooperation

In almost all of the work in collective robotics so far, it has been assumed that cooperation is explicitly designed into the system. An interesting research problem is to study how cooperation can arise without explicit human motivation among possibly selfish agents.

McFarland [McFarland, 1994] distinguishes between two significantly different types of group behaviors that are found in nature: *eusocial behavior* and *cooperative behavior*. Eusocial behavior is found in many insect species (e.g., colonies of ants or bees), and is the result of genetically determined individual behavior. In eusocial societies, individual agents are not very capable, but seemingly intelligent behavior arises out of their interactions. This "cooperative" behavior is necessary for the survival of the individuals in the colonies. [Werner and Dyer, 1992] studies the evolution of herding behavior in "prey" agents in a simulated ecology, where there is no *a priori* drive for cooperation. Recently, [McFarland, 1994, Steels, 1994] have laid the initial groundwork to address the problem of emergent cooperation in an ecological system inhabited by actual mobile robots. In their ecosystem, individual robots are selfish, utility-driven agents that must cooperate in order to survive (i.e., maintain some minimal energy level).

On the other hand, [McFarland, 1994] defines cooperative behavior as the social behavior observed in higher animals (vertebrates); i.e., cooperation is the result of interactions between selfish agents. Unlike eusocial behavior, cooperative behavior is not motivated by innate behavior, but by an intentional desire to cooperate in order to maximize individual utility.

Inspired by economics and game-theoretic approaches, [Bond and Gasser, 1988] [Genesereth *et al.*, 1986] [Rosenschein and Genesereth, 1985] [Rosenschein and Zlotkin, 1994] and others have studied the emergence of cooperation in selfish rational agents in the field of distributed artificial intelligence (DAI). A recent work in the robotics literature that adopts this game-theoretic approach is [Numaoka, 1993].

## 2.4 Learning

Finding the correct values for control parameters that lead to a desired cooperative behavior can be a difficult, time-consuming task for a human designer. Therefore, it is highly desirable for multiple-robot systems to be able to *learn* control parameter values in order to optimize their task performance, and to adapt to changes in the environment. Reinforcement learning [Barto *et al.*, 1983, Kaelbling, 1993] has often been used in cooperative robotics.

[Mataric, 1994a, Mataric, 1994b] propose a reformulation of the reinforcement learning paradigm using higher levels of abstraction (conditions, behaviors, and heterogeneous reward functions and progress estimators instead of states, actions, and reinforcement) to enable robots to learn a composite foraging behavior. [Parker, 1994b] uses standard reinforcement algorithms to improve the performance of cooperating agents in the L-ALLIANCE architecture by having the agents learn how to better estimate the performance of other agents. [Sen *et al.*, 1994] uses reinforcement learning in a two-robot box-pushing system, and [Yanco and Stein, 1992] applies reinforcement learning to learn a simple, artificial robot language. Other relevant works in multiagent reinforcement learning (done in simulation, in contrast to the above works which were implemented on actual robots) include [Whitehead, 1991, Tan, 1993, Littman, 1994].

In addition, techniques inspired by biological evolution have also been used in cooperative robotics. [Werner and Dyer, 1992] uses a genetic algorithm [Goldberg, 1989] to evolve neural network controllers for simulated "prey" creatures that learn a herding behavior to help avoid predators. [Reynolds, 1992] uses genetic programming [Koza, 1990] to evolve flocking behavior in simulated "boids."

## 2.5 Geometric Problems

Because mobile robots can move about in the physical world and must interact with each other physically, geometric problems are inherent to multiple-robot systems. This is a fundamental property that distinguishes multiple-robot systems from traditional distributed computer systems in which individual nodes are stationary. Geometric problems that have been studied in the cooperative robotics literature include multiple-robot path planning, moving to (and maintaining) formation, and pattern generation.

### (Multiple-Robot) Path Planning

Recall that multiple-robot path planning requires agents to plan routes that do not intersect. This is a case of resource conflict, since the agents and their goals are embedded in a finite amount of space. However, we note path planning separately because of its intrinsic geometric flavor as well as its historical importance in the literature.

Detailed reviews of path planning are found in [Fujimura, 1991, Latombe, 1991, Arai and Ota, 1992]. Fujimura [Fujimura, 1991] views path planning as either centralized (with a universal path-planner making decisions) or distributed (with individual agents planning and adjusting their paths). Arai and Ota [Arai and Ota, 1992] make a similar distinction in the nature of the planner, and also allow hybrid systems that can be combinations of on-line, off-line, centralized, or decentralized. Latombe [Latombe, 1991] gives a somewhat different taxonomy: his "centralized planning" is planning that takes into account all robots, while "decoupled" planning entails planning the path of each robot independently. For centralized planning, several methods originally used for single-robot systems can be applied. For decoupled planning, two approaches are given: (i) *prioritized planning* considers one robot at a time according to a global priority, while (ii) the *path coordination method* essentially plans paths by scheduling the configuration space-time resource. The work of [Erdmann and Lozano-Perez, 1986] is a typical decoupled approach where every robot is prioritized and robots plan global paths with respect to only higher-priority robots (e.g., the highest-priority robot plans only around the obstacles in the environment). Note that this is still a centralized method according to the terminology of [Fujimura, 1991, Arai and Ota, 1992]. On the other hand, [Yeung and Bekey, 1987] presents a distributed approach (per Fujimura's taxonomy) where each robot initially attempts a straight-line path to the goal; if an interfering obstacle is seen, then the robot will scan the visible vertices of the obstacle and move toward the closest one. In general, this continues until the goal is reached. Dynamically varying priorities are given to each robot (based on current need) to resolve path intersection conflicts, and conflicting robots can either negotiate among themselves or allow a global blackboard manager to perform this function.

Some recent works have addressed some nontraditional motion planning problems. For example, [Hert and Lumelsky, 1995] proposes an algorithm for path planning in tethered robots, and [Ota *et al.*, 1995] consider the problem of moving while grasping large objects.

**The Formation and Marching Problems**

The Formation and Marching problems respectively require multiple robots to form up and move in a specified pattern. Solving these problems is quite interesting in terms of distributed algorithms [Sugihara and Suzuki, 1990], balancing between global and local knowledge [Parker, 1994b], and intrinsic information requirements for a given task. Solutions to Formation and Marching are also useful primitives for larger tasks, e.g., moving a large object by a group of robots [Stilwell and Bay, 1993] [Chen and Luh, 1994a] [Chen and Luh, 1994b] or distributed sensing [Wang and Beni, 1988].

The Formation problem seems very difficult, e.g., no published work has yet given a distributed "circle-forming" algorithm that guarantees the robots will actually end up in a circle. For this problem, the best known solution is the distributed algorithm of [Sugihara and Suzuki, 1990], which guarantees

only that the robots will end up in a shape of constant diameter (e.g., a Reuleaux triangle can be the result). It is assumed that the $i^{th}$ mobile robot knows the distances $D_i$ and $d_i$ to its farthest and nearest neighbors, respectively; the algorithm attempts to match the ratios $D_i/d_i$ to a prescribed constant. No method of detecting termination was given. [Chen and Luh, 1994a, Chen and Luh, 1994b] extend the method of [Sugihara and Suzuki, 1990] to incorporate collision avoidance when the robots are moving. [Yamaguchi and Arai, 1994] approaches the shape-generation problem using systems of linear equations; starting at some initial location, each robot changes its $(x, y)$ position according to a linear function of its neighbors' positions and some fixed constant. Simulations of the method show that a group of initially collinear robots will converge into the shape of an arc.

We observe that the circle-forming problem, while quite simple to state, reveals several pitfalls in formulating distributed geometric tasks. For example, the ability of an individual agent to sense attributes of the formation must be carefully considered: too much information makes the problem trivial, but too little information (e.g., returns from localized sensors) may prevent a solution (e.g., robots may never find each other). Information lower bounds, e.g., for robots to be able to realize that they have achieved the prescribed formation, are also largely unexplored in the literature. Interestingly, we note that the algorithm of [Sugihara and Suzuki, 1990] can be slightly modified: rather than each robot seeking to achieve a prescribed ratio $D/d$, each robot could seek to achieve a prescribed angle (close to 90 degrees) subtended by its farthest neighbor and its closest neighbor to the right. This uses very similar sensing capabilities but guarantees the desired circular shape.

For Marching, [Chen and Luh, 1994a] employs positional constraint conditions in a group of robots that makes turns while maintaining an array pattern. In [Chen and Luh, 1994b] a leader-follower approach is used to solve a similar task. [Parker, 1993] studies the problem of keeping four marching robots in a side-by-side formation; this increases in difficulty when the leader has to perform obstacle avoidance or other maneuvers. Parker also defines the concepts of global goals and global/local knowledge. To study the effects of different distributions of global goals and global knowledge, four strategies are compared both in simulation and on mobile robots. Simplified instances of the Marching problem require robots to reliably follow each other and to move in a group (without tight constraints on their relative positions). Some works that address this problem (sometimes referred to as the herding/flocking problem) include [Reynolds, 1987, Mataric, 1994a, Hodgins and Brogan, 1994, Brogan and Hodgins, 1995, Milios and Wilkes, 1995]. A somewhat related problem is the problem of cooperative positioning (determining the locations of the robots in a group using limited information) [Kurazume and Nagata, 1994].

Related to the Formation problem is the pattern generation problem in Cellular Robotic Systems, multiple-robot systems which can "encode information as patterns of its own structural units"

[Beni, 1988]. Typically, one- or two-dimensional grids constitute the workspace, and sensing of neighboring cells is the only input. Within these constraints, a set of rules is devised and applied to all agents; a standard result is to show in simulation that convergence to some spatial pattern is guaranteed. The meaningful aspect of this work lies in providing a system with the capability of spatial self-organization: a CRS will reconfigure itself without intervention in certain situations or under certain conditions.

In [Wang and Beni, 1988, Liang and Beni, 1995], a CRS is characterized as an arbitrary number of robots in a one- or two-dimensional grid. The robots are able to sense neighboring cells and communicate with other robots via a signboard mechanism. Protocols are presented for creating different patterns, e.g., alternating robots and spaces in a one-dimensional grid; covering the top row of a two-dimensional grid by robots; or covering the boundary of a two-dimensional grid by robots. Egecioglu and Zimmermann [Egecioglu and Zimmermann, 1988] pose the "Random Pairing" problem, and seek a set of rules by which for any given number, a CRS will converge to a pattern such that there is a group of two robots with that number of vacant spaces between them (see also [Beni and Hackwood, 1992]). An analogous cellular approach is adopted by Genovese et al. [Genovese *et al.*, 1992], who describe the simulation of a system of pollutant-seeking mobile robots. The simulation uses a potential field mechanism to attract robots to the pollutant and to repulse robots from each other. The combined effect of these two forces yields a gradient pattern that "points" toward the source of the pollutant.

# 3 Perspectives

As an integrative engineering discipline, robotics has always had to confront technological constraints that limit the domains that can be studied. Cooperative robotics has been subject to these same constraints, but the constraints tend to be more severe because of the need to cope with multiple robots. At the same time, cooperative robotics is a highly interdisciplinary field that offers the opportunity to draw influences from many other domains. In this section, we first outline some of the technological constraints that face the field. We then mention some directions in which *cooperative robotics* might progress, and describe related fields that have provided and will continue to provide influences.

## 3.1 Technological Constraints

It is clear that technological constraints have limited the scope of implementations and task domains attempted in multiple-robot research systems.

One obvious problem that arises is the general problem of researchers having to solve various instances of the vision problem before being able to make progress on "higher-level" problems. Often, difficulties arising from having to solve difficult perceptual problems can limit the range of tasks that

can be implemented on a multiple-robot platform. For example, in cooperative robotics systems where modeling of other agents (see Section 2.1.4) is used, the lack of an effective sensor array can render the system unimplementable in practice. In addition, robot hardware is also notoriously unreliable; as a result, it is extremely difficult to maintain a fleet of robots in working condition. Again, collective robotics must deal with all of the hardware problems of single-robotic systems, exacerbated by the multiplicity of agents.

Due to the difficulties (such as those outlined above) encountered when working with real robots, much of collective robotics has been studied exclusively in simulation. Some researchers have argued (cf. [Brooks, 1991]) that by ignoring most of the difficulties associated with perception and actuation, simulations ignore the most difficult problems of robotics. By making overly simplistic assumptions, it is possible to generate "successful" systems in simulation that would be infeasible in the real world. (Conversely, mobile research robots can also come to "look like the simulator", i.e., circular footprint, sonar ring, synchro-drive is a common configuration.) Nevertheless, simulation must inevitably play a role in multi-agent robotics at some level. Although it is currently possible for researchers to study groups of 10-20 robots, it is unlikely that truly large-scale collective behavior involving hundreds or thousands of real robots will be feasible at any time in the near future. Thus, cooperative mobile robot researchers have used a variety of techniques to simulate perception, while using physical robots. For instance, the use of a global positioning system can in part compensate for the lack of vision, but can place severe environmental constraints under which robots can operate (because many objects and acoustic features of the environment can interfere with the GPS). For the basic problem of differentiating between other agents and all other objects in the environment. some researchers [Parker, 1994b] use radio communication to solve this problem. In other works [Donald, 1993a, Parker, 1994b] interaction via sensing is done by explicit radio communication. There are recent attempts to perform recognition via vision [Kuniyoshi *et al.*, 1994a, Kuniyoshi *et al.*, 1994b].

An approach taken by some researchers is to use simulations as prototypes for larger-scale studies, and small numbers of real robots as a proof-of-concept demonstration [Mataric, 1994a, Parker, 1994b]. On the other hand, some researchers, citing the necessity of working in the real world domain, have chosen to eschew simulations altogether and implement their theories directly on actual robots [Beckers *et al.*, 1994] [McFarland, 1994] [Steels, 1994]. In studies of locomotion in large herds of (upto 100) one-legged robots and simulated human cyclists, [Hodgins and Brogan, 1994] [Brogan and Hodgins, 1995] take an alternate approach of design a very physically realistic simulation. While this approach brings realism to actuation, the issue of perception is still simulated away; it is still unclear whether it will be feasible to realistically model sophisticated agents in more complex environments, or whether the effort will outweigh the benefits.

## 3.2 Towards a Science of Cooperative Robotics

The field of cooperative mobile robotics offers an incredibly rich application domain, integrating a huge number of distinct fields from the social sciences, life sciences, and engineering. That so many theories have been brought to bear on "cooperative robotics" clearly shows the energy and the allure of the field. Yet, cooperative robotics is still an emerging field, and many open directions remain. In this subsection, we point out some promising directions that have yet to be fully explored by the research community. By way of a preface, we also point out three "cultural" changes which may come as the field matures: (1) Because of the youth of the field, cooperative robotics research has been necessarily rather informal and "concept" oriented. However, the development of rigorous formalisms is desirable to clarify various assumptions about the systems being discussed, and to obtain a more precise language for discussion of elusive concepts such as cooperation (there are some exceptions, such as [Parker, 1994b], which presents a formalization of motivational behavior in the ALLIANCE architecture). (2) Formal metrics for cooperation and system performance, as well as for grades of cooperation, are noticeably missing from the literature. While the notion of cooperation is difficult to formalize, such metrics will be very useful in characterizing various systems, and would improve our understanding of the nature of agent interactions. Although [Mataric, 1994a] has suggested parameters such as agent density for estimating interference in a multi-robot system, much more work in this area is necessary. (3) Experimental studies might become more rigorous and thorough, e.g., via standard benchmark problems and algorithms. This is challenging in mobile robotics, given the noisy, system-specific nature of the field. Nevertheless, it is necessary for claims about "robustness" and "near-optimality" to be appropriately quantified, and for dependencies on various control parameters to be better understood. For example, we have noted that despite a number of claims that various decentralized approaches are superior to centralized approaches, we have not seen any thorough, published experimental comparisons between the major competing paradigms on a particular task. However, we note that recently, researchers have begun to empirically study quantitative measures of cooperation, trying to identify conditions under which mechanisms for cooperation are beneficial [Arkin *et al.*, 1993, Arkin and Hobbs, 1993, Parker, 1995].

Finally, several basic analogies remain incomplete, and must be revisited and resynthesized as the field matures. For instance, many multi-robot problems are "canonical" for distributed computation and are interesting primarily when viewed in this light. A typical example is moving to formation, which has been solved optimally in the computational geometry literature (it is the "geometric matching under isometry" problem [P.J. Heffernan, 1992]), but which is difficult in the distributed context due to issues like synchronization, fault-tolerance, leader election, etc. However, the distributed context can be selectively ignored, e.g., [Sugihara and Suzuki, 1990] use "human intervention" to perform what is essentially leader election (breaking symmetry in a circle of robots to choose vertices of the desired

polygonal formation). The introduction of such devices runs counter to the implicit assumption that it is the *distributed* problem that holds research interest.

More generally, it is likely that more structural and less superficial analogies with other disciplines will be needed in order to obtain "principled" theories of cooperation among (mobile) robots; integration of formalisms and methodologies developed in these more mature disciplines is likely to be an important step in the development of cooperative robotics. Disciplines most critical to the growth of cooperative robotics are: distributed artificial intelligence, biology, and distributed systems.

**Distributed Artificial Intelligence**

The field of distributed artificial intelligence (DAI) concerns itself with the study of distributed systems of intelligent agents. As such, this field is highly relevant to cooperative robotics. Bond and Gasser [Bond and Gasser, 1988] define DAI as "the subfield of artificial intelligence (AI) concerned with concurrency in AI computations, at many levels." Grounded in traditional symbolic AI and the social sciences, DAI is composed of two major areas of study: Distributed Problem Solving (DPS) and Multiagent Systems (MAS).

Research in DPS is concerned with the issue of solving a single problem using many agents. Agents can cooperate by independently solving subproblems (task-sharing), and by periodically communicating partial solutions to each other (result-sharing). DPS involves three possibly overlapping phases: (i) problem decomposition (task allocation), (ii) subproblem solution, and (iii) solution synthesis. Of these, problem decomposition has attracted the greatest interest among DAI researchers. The critical issue in task sharing is finding the appropriate agent to assign to a subproblem. This is nontrivial, since if the most appropriate agent for a subtask is not obvious, then the system must try to determine which of the many eligible agents should be assigned the task, and often there are too many eligible agents to attempt an exhaustive search. Perhaps the best known scheme for task allocation is the Contract Net Protocol [Smith, 1980], which has been used in the ACTRESS [Ishida *et al.*, 1994, Asama *et al.*, 1994, Ozaki *et al.*, 1993] and GOFER [Caloud *et al.*, 1990] projects.

One important assumption in DPS is that the agents are predisposed to cooperate. Research in DPS is thus concerned with developing frameworks for cooperative behavior between willing agents, rather than developing frameworks to enforce cooperation between potentially incompatible agents, as is the case with multiagent systems and distributed processing.

Multiagent Systems (MAS) research is the study of the collective behavior of a group of possibly heterogeneous agents with potentially conflicting goals. In other words, researchers in MAS discard the "benevolent agent" assumption of DPS [Genesereth *et al.*, 1986]. [Genesereth *et al.*, 1986] states the central problem of MAS research as follows: "in a world in which we get to design only our own

intelligent agent, how should it interact with other intelligent agents?" Therefore, areas of interest in MAS research include game-theoretic analysis of multi-agent interactions (cf. [Genesereth *et al.*, 1986, Rosenschein and Genesereth, 1985, Rosenschein and Zlotkin, 1994]), reasoning about other agents' goals, beliefs, and actions (cf. [Rosenschein, 1982, Georgeff, 1983, Georgeff, 1984]), and analysis of the complexity of social interactions [Shoham and Tennenholtz, 1992].

Work in MAS has tended to be theoretical and in very abstract domains. A common underlying assumption is that although the agents may be selfish, they are rational and highly deliberative. This is in stark contrast with research in swarm intelligence (see Section 2.5), in which individual agents are assumed to be relatively unintelligent.

However, the influence of DAI on cooperative robotics has been limited. This is in part because researchers in DAI have mostly concentrated on domains where uncertainty is not as much of an issue as it is in the physical world. Work in MAS has tended to be theoretical and in very abstract domains where perfect sensing is usually assumed; typical DPS domains are in disembodied, knowledge-based systems. Another assumption of DAI that has prevented its application in cooperative robotics is the assumption is that although agents may be selfish, they are rational and highly deliberative. However, achieving strict criteria of rationality and deliberativeness can often be prohibitively expensive in current robotic systems. Thus, it has been argued that DAI, while suited for unsituated, knowledge-based systems, will not succeed in the domain of cooperative robotics [Parker, 1994b, Mataric, 1994a]. However, we observe that direct comparisons of DAI and alternative paradigms are notably missing from the literature; such comparisons are needed to evaluate the true utility of DAI techniques in cooperative robotics. Also, as lower-level processes (perception and actuation) are better understood and implemented, and as computational power increases, the high-level results of DAI research may become increasingly applicable to collective mobile robotics.

**Distributed Systems**

A multiple-robot system is in fact a special case of a distributed system. Thus, the field of distributed systems is a natural source of ideas and solutions. [Beni, 1988] describes cellular robotics as belonging to the general field of distributed computing. It is noted, however, that distributed computing can only contribute general theoretical foundations and that further progress needs to be made concerning the application of such methods to collective robotics. [Wang and Beni, 1990] states, "a distributed computing system contains a collection of computing devices which may reside in geographically separated locations called sites." By noting the similarities with distributed computing, theories pertaining to deadlock [Wang and Beni, 1988, Wang and Beni, 1990, Lin and Hsu, 1995], message passing [Wang and Beni, 1990] and resource allocation [Wang, 1991], and the combination of the above as primitives [Wang, 1995, Wang and Premvuti, 1995], have been applied to collective robotics

in a number of works.

In work done on multiple AGV systems and Distributed Robotic Systems, deadlock detection and resource allocation methods are applied to allow many robots to share the limited resource of path space [Wang and Beni, 1990, Wang, 1991]. Pattern generation in a CRS may also rely on distributed computing to resolve conflicts [Wang, 1991, Wang and Beni, 1990]. Finally, [Wang, 1993, Wang, 1994] describe a task allocation algorithm where the robots vie for the right to participate in a task. See also the discussion in Section 2.1.1 and Section 2.5.

Broadcast communication, which is widely assumed in cooperative robotics, exhibits poor scaling properties. As robots become more numerous and widely distributed, techniques and issues from the field of computer networks become relevant. A rich body of research on algorithms, protocols, performance modeling and analysis in computer networks can be applied to cooperative robotics. There is currently a great amount of effort being put into studying networking issues related to mobile/nomadic/ubiquitous computing (cf. [Awerbuch and Peleg, 1991, Weiser, 1993, Kleinrock, 1995, Badrinath *et al.*, 1994]). Results from this field could be applied in a straightforward way to multi-robot systems.

Finally, distributed control is a promising framework for the coordination of multiple robots. Due to difficulty of sensing and communication, a parsimonious formulation which can coordinate robots having minimal sensing and communication capabilities is desirable. In an ideal scenario, maximal fault tolerance is possible, modeling of other agents is unnecessary, and each agent is controlled by a very simple mechanism. A distributed control scheme (known as the GUR game) developed originally by [Tsetlin, 1964] and recently studied in [Tung and Kleinrock, 1993, Tung, 1994] provides a framework in which groups of agents with minimal sensing capability and no communication are controlled by simple finite state automata and converge to optimal behaviors. [Tung, 1994] describes possible cooperative robotics applications in moving platform control and perimeter guarding.

**Biology**

Biological analogies and influences abound in the field of cooperative robotics. The majority of existing work in the field has cited biological systems as inspiration or justification. Well-known collective behaviors of ants, bees, and other eusocial insects [Wilson, 1971] provide striking existence proofs that systems composed of simple agents can accomplish sophisticated tasks in the real world. It is widely held that the cognitive capabilities of these insects are very limited, and that complex behaviors *emerge* out of interactions between the agents, which are individually obeying simple rules. Thus, rather than following the AI tradition of modeling robots as rational, deliberative agents, some researchers in cooperative robotics have chosen to take a more "bottom-up"

approach in which individual agents are more like ants – they follow simple rules, and are highly reactive (this is the approach taken in the field of Artificial Life). Works based on this insect-colony analogy include [Mataric, 1994a, Beckers *et al.*, 1994, Stilwell and Bay, 1993, Doty and Aken, 1993, Johnson and Bay, 1994, Deneubourg *et al.*, 1991a, Deneubourg *et al.*, 1991b]. The pattern generation of CRS's can also be considered as bottom-up (see Section 2.5), since each robot is designed as a very simple agent which follows a set of prespecified rules.

A more general, biological metaphor that is often used in cooperative robotics is the concept of a *self-organizing system* [Nicolis and Prigogine, 1977, Yates, 1987]. (Note that researchers from many fields have studied self-organization; it is by no means an exclusively biological concept. However, in the field of cooperative robotics, references to self-organization have often been made in a biological context.) The behavior of insect colonies described above can be characterized more generally as that of self-organizing systems. Representative work that is based on this concept includes [Wang and Beni, 1988, Steels, 1990, Hackwood and Beni, 1991, Hackwood and Beni, 1992, Beni and Hackwood, 1992]. Self-organization in multi-cellular biological systems has been an inspiration for [Hackwood and Wang, 1988, Beni, 1988, Egecioglu and Zimmermann, 1988, Genovese *et al.*, 1992]. Hierarchical organization of biological multi-cellular organisms (i.e., from cellular to tissue to organism level) has been used as a guiding metaphor for cellular robotics in the CEBOT project [Fukuda and Nakagawa, 1987].

Biological analogies have also influenced the choice of task domains studied in cooperative robotics. While foraging is a natural abstraction of some practical applications such as waste retrieval and search and rescue, one major reason that it has become identified as the canonical cooperative robotic task is that it is a natural task, given the group architectures resulting from analogies to insect colonies. Another example of this phenomenon is the flocking/herding task. It seems no accident that biological inspirations led to "natural" models of group motion, as opposed to more structured models of coordinated motion (such as moving in some arbitrary formation).

Finally, as we noted in Section 2.4, there have been some biological influences on the learning and optimization algorithms used to tune control parameters in multiple-robot systems.

# 4 Conclusions

We have synthesized a view of the theoretical bases for research in cooperative mobile robotics. Key research axes in the field were identified, particularly with respect to achieving a "mechanism of cooperation", and existing works were surveyed in this framework. We then discussed technological constraints and interdisciplinary influences that have shaped the field, and offered some general precepts for future growth of the field. Finally, we identified distributed artificial intelligence, biology, and

distributed systems as disciplines that are most relevant to cooperative robotics, and which are most likely to continue to provide valuable influences. Based on our synthesis, a number of open research areas become apparent. We believe that the following are among the major, yet tractable, challenges for the near future:

1. robust definitions and metrics for various forms of cooperation,
2. achieving a more complete theory of information requirements for task-solving in spatial domains, perhaps for the canonical tasks of pattern formation or distributed sensing (e.g., measures of pattern complexity, information lower bounds for pattern recognition and maintenance, abstraction of sensor models from the solution approach). The works of [Donald *et al.*, 1994, Rus *et al.*, 1995, Brown and Jennings, 1995] have begun to address this issue, in the context of object manipulation tasks; interestingly, [Brown and Jennings, 1995] observes that given a robot system, some tasks are *strongly cooperative* – the robots must act in concert to achieve the goal, and the strategy for the task is not trivially serializable.
3. principled transfer of the concepts of fault-tolerance and reliability from the field of distributed and fault-tolerant computing,
4. incorporation of recent ideas in distributed control to achieve oblivious cooperation, or cooperation without communication (e.g., when robots have minimal sensing and communication capabilities),
5. achieving cooperation within competitive situations (e.g., for robot soccer, or pursuit-evasion with multiple pursuers and evaders). An interesting open problem is how well solutions that have been developed in discretized abstractions of these domains (cf. [Levy and Rosenschein, 1991, Korf, 1992]) translate to the physical world.

## Acknowledgements

Partial support for this work was provided by NSF Young Investigator Award MIP-9257982; the UCLA Commotion Laboratory is supported by NSF CDA-9303148. The authors would like to thank B. Donald, T. Fukuda, M. Anthony Lewis, M. Mataric, J. Wang, and members of the UCLA Commotion Lab for helpful comments, suggestions, and discussions. Frank Meng assisted in the preparation of a previous version of this paper.

# References


[Adams *et al.*, 1995] J.A. Adams, R. Basjcsy, J. Kosecka, V. Kumar, R. Mandelbaum, M. Mintz, R. Paul, C. Wang, Y. Yamamoto, and X. Yun. Cooperative material handling by human and robotic agents: Module development and system synthesis. In *IEEE/RSJ IROS*, pages 200–205, 1995.

[Agah and Bekey, 1994] A. Agah and G. A. Bekey. Autonomous mobile robot teams. In *Conf. on Intelligent Robotics in Filed, Factory, Service and Space (CIRFFSS94)*, 1994.

[Aguilar *et al.*, 1995] L. Aguilar, R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Ten autonomous mobile robots (and even more). In *IEEE/RSJ IROS*, pages 260–267, 1995.

[Alani *et al.*, 1995] R. Alani, F. Robert, F. Ingrand, and S. Suzuki. Multi-robot cooperation through incremental plan-merging. In *IEEE ICRA*, pages 2573–2579, 1995.

[Albus, 1993] J. S. Albus. *A Control Architecture for Cooperative Intelligent Robots*, pages 713–743. 1993.

[Arai and Ota, 1992] T. Arai and J. Ota. Motion planning of multiple robots. In *IEEE/RSJ IROS*, pages 1761–1768, 1992.

[Arai *et al.*, 1993] T. Arai, E. Yoshida, and J. Ota. Information diffusion by local communication of multiple mobile robots. In *IEEE Conference on Systems, Man and Cybernetics*, pages 535–540, 1993.

[Arkin and Ali, 1994] R. Arkin and K. Ali. Integration of reactive and telerobotic control in multi-agent robotic systems. In *Proc. Simulation of Adaptive Behavior*, 1994.

[Arkin and Hobbs, 1993] R. Arkin and J. Hobbs. Dimensions of communication and social organization in multi-agent robotic systems. In *Proc. Simulation of Adaptive Behavior*, 1993.

[Arkin *et al.*, 1993] R. C. Arkin, T. Balch, and E. Nitz. Communication of behavioral state in multi-agent retrieval tasks. In *IEEE ICRA*, volume 3, pages 588–594, 1993.

[Arkin, 1992] R. C. Arkin. Cooperation without communication: Multiagent schema-based robot navigation. *Journal of Robotic Systems*, 9(3):351–364, 1992.

[Asada *et al.*, 1994] M. Asada, E. Uchibe, S. Noda, S. Tawaratsumida, and K. Hosoda. Coordination of multiple behaviors acquired by a vision-based reinforcement learning. In *IEEE/RSJ IROS*, 1994.

[Asama *et al.*, 1989] H. Asama, A. Matsumoto, and Y. Ishida. Design of an autonomous and distributed robot system: ACTRESS. In *IEEE/RSJ IROS*, pages 283–290, 1989.

[Asama *et al.*, 1991a] H. Asama, M. K. Habib, I. Endo, K. Ozaki, A. Matsumoto, and Y. Ishida. Functional distribution among multiple mobile robots in an autonomous and decentralized robot system. In *IEEE ICRA*, pages 1921–6, 1991.

[Asama *et al.*, 1991b] H. Asama, K. Ozaki, Y. Ishida, M. K. Habib, A. Matsumoto, and I. Endo. Negotiation between multiple mobile robots and an environment manager. In *IEEE ICRA*, pages 533–538, 1991.

[Asama *et al.*, 1991c] H. Asama, K. Ozaki, H. Itakura, A. Matsumoto, Y. Ishida, and I. Endo. Collision avoidance among multiple mobile robots based on rules and communication. In *IEEE/RSJ IROS*, pages 1215–1220, 1991.

[Asama *et al.*, 1992] H. Asama, Y. Ishida, K. Ozaki, M. K. Habib, A. Matsumoto, H. Kaetsu, and I. Endo. A communication system between multiple robotic agents. In M. Leu, editor, *Proc. the Japan U.S.A. Symposium on Flexible Automation*, pages 647–654, 1992.

[Asama *et al.*, 1994] H. Asama, K. Ozaki, Y. Ishida, K. Yokita, A. Matsumoto, H. Kaetsu, and I. Endo. Collaborative team organization using communication in a decentralized robotic system. In *IEEE/RSJ IROS*, 1994.

[Asama, 1992] H. Asama. Distributed autonomous robotic system configurated with multiple agents and its cooperative behaviors. *Journal of Robotics and Mechatronics*, 4(3), 1992.

[Awerbuch and Peleg, 1991] B. Awerbuch and D. Peleg. Concurrent online tracking of mobile users. *Computer Communication Review*, 21(4):221–233, 1991.

[Badrinath *et al.*, 1994] B.R. Badrinath, A. Acharya, and T. Imielinski. Structuring distributed algorithms for mobile hosts. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 21–24, June 1994.

[Barnes and Gray, 1991] D. Barnes and J. Gray. Behaviour synthesis for co-operant mobile robot control. In *International Conference on Control*, pages 1135–1140, 1991.

[Barto *et al.*, 1983] A.G. Barto, R.S. Sutton, and C.J.C.H. Watkins. Learning and sequential decision making. In M. Gabriel and J. Moore, editors, *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pages 539–603. MIT Press, 1983.

[Beckers *et al.*, 1994] R. Beckers, O. E. Holland, and J. L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In *Proc. A-Life IV*. MIT Press, 1994.

[Beni and Hackwood, 1992] G. Beni and S. Hackwood. Stationary waves in cyclic swarms. In *IEEE International Symposium on Intelligent Control*, pages 234–242, 1992.

[Beni and Wang, 1991] G. Beni and J. Wang. Theoretical problems for the realization of distributed robotic systems. In *IEEE ICRA*, pages 1914–1920, 1991.

[Beni, 1988] G. Beni. The concept of cellular robotic system. In *IEEE International Symposium on Intelligent Control*, pages 57–62, 1988.

[Bond and Gasser, 1988] A. H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, 1988.

[Brogan and Hodgins, 1995] D.C. Brogan and J.C. Hodgins. Group behaviors for systems with significant dynamics. In *IEEE/RSJ IROS*, pages 528–534, 1995.

[Brooks *et al.*, 1990] R. A. Brooks, P. Maes, M. J. Mataric, and G. More. Lunar base construction robots. In *IEEE/RSJ IROS*. IEEE, July 1990.

[Brooks, 1986] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[Brooks, 1991] R. A. Brooks. Intelligence without reason. In *Proc. Intl. Joint Conf. Artificial Intelligence*, pages 569–595, 1991.

[Brown and Jennings, 1995] R.G. Brown and J.S. Jennings. A pusher/steerer model for strongly cooperative mobile robot cooperation. In *IEEE/RSJ IROS*, pages 562–568, 1995.

[Cai *et al.*, 1995] A.-H. Cai, T. Fukuda, F. Arai, T. Ueyama, and A. Sakai. Hierarchical control architecture for cellular robotic system. In *IEEE ICRA*, pages 1191–1196, 1995.

[Caloud *et al.*, 1990] P. Caloud, W. Choi, J.-C. Latombe, C. Le Pape, and M. Yin. Indoor automation with many mobile robots. In *IEEE/RSJ IROS*, pages 67–72, 1990.

[Causse and Pampagnin, 1995] O. Causse and L.H. Pampagnin. Management of a multi-robot system in a public environment. In *IEEE/RSJ IROS*, pages 246–252, 1995.

[Chen and Luh, 1994a] Q. Chen and J. Y. S. Luh. Coordination and control of a group of small mobile robots. In *IEEE ICRA*, pages 2315–2320, 1994.

[Chen and Luh, 1994b] Q. Chen and J. Y. S. Luh. Distributed motion coordination of multiple robots. In *IEEE/RSJ IROS*, pages 1493–1500, 1994.

[Connell, 1987] J. Connell. Creature design with the subsumption architecture. In *Proc. AAAI*, pages 1124–1126, 1987.

[Dario *et al.*, 1991] P. Dario, F. Ribechini, V. Genovese, and G. Sandini. Instinctive behaviors and personalities in societies of cellular robots. In *IEEE ICRA*, pages 1927–1932, 1991.

[Deneubourg *et al.*, 1991a] J. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. The dynamics of collective sorting: Robot-like ants and ant-like robots. In *Proc. Simulation of Adaptive Behavior*, 1991.

[Deneubourg *et al.*, 1991b] J-L. Deneubourg, G. Theraulaz, and R. Beckers. Swarm-made architectures. In Verela and Bourgine, editors, *Proc. European Conference on Artificial Life*, pages 123–133. MIT Press, 1991.

[Donald *et al.*, 1994] B. R. Donald, J. Jennings, and D. Rus. Analyzing teams of cooperating mobile robots. In *IEEE ICRA*, pages 1896–1903, 1994.

[Donald, 1993a] B. R. Donald. Information invariants in robotics: I. state, communication, and side-effects. In *IEEE ICRA*, pages 276–283, 1993.

[Donald, 1993b] B. R. Donald. Information invariants in robotics: II. sensors and computation. In *IEEE ICRA*, volume 3, pages 284–90, 1993.

[Dorf, 1990] R. Dorf. *Concise International Encyclopedia of Robotics: Applications and Automation*. Wiley-Interscience, 1990.

[Doty and Aken, 1993] K. L. Doty and R. E. Van Aken. Swarm robot materials handling paradigm for a manufacturing workcell. In *IEEE ICRA*, volume 1, pages 778–782, 1993.

[Drexler, 1992] K.E. Drexler. *Nanosystems: Molecular Machinery, Manufacturing, and Computation.* John Wiley and Sons, Inc., 1992.

[Drgoul and Ferber, 1993] A. Drgoul and J. Ferber. From tom thumb to the dockers: Some experiments with foraging robots. In *Proc. Simulation of Adaptive Behavior*, 1993.

[Dudek *et al.*, 1993] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for swarm robots. In *IEEE/RSJ IROS*, pages 441–447, 1993.

[Egecioglu and Zimmermann, 1988] O. Egecioglu and B. Zimmermann. The one dimensional random pairing problem in a cellular robotic system. In *IEEE International Symposium on Intelligent Control*, pages 76–80, 1988.

[Erdmann and Lozano-Perez, 1986] M. Erdmann and T. Lozano-Perez. On multiple moving objects. In *IEEE ICRA*, pages 1419–1424, 1986.

[Everett *et al.*, 1993] H. Everett, G.A. Gilbreath, T.A. Heath-Pastore, and R.T. Laird. Coordinated control of multiple security robots. *Mobile Robots VIII*, 2058:292–305, 1993.

[Franklin *et al.*, 1995] D.E. Franklin, A.B. Kahng, and M.A. Lewis. Distributed sensing and probing with multiple search agents: toward system-level landmine detection solutions. In *Detection Technologies for Mines and Minelike Targets, Proceedings of SPIE, Vol.2496*, pages 698–709, 1995.

[Fujimura, 1991] K. Fujimura. *Motion Planning in Dynamic Environments.* Springer-Verlag, New York, NY, 1991.

[Fukuda and Iritani, 1995] T. Fukuda and G. Iritani. Construction mechanism of group behavior with cooperation. In *IEEE/RSJ IROS*, pages 535–542, 1995.

[Fukuda and Kawauchi, 1993] T. Fukuda and Y. Kawauchi. *Cellular Robotics*, pages 745–782. Springer-Verlag, 1993.

[Fukuda and Nakagawa, 1987] T. Fukuda and S. Nakagawa. A dynamically reconfigurable robotic system (concept of a system and optimal configurations). In *International Conference on Industrial Electronics, Control, and Instrumentation*, pages 588–95, 1987.

[Fukuda and Sekiyama, 1994] T. Fukuda and K. Sekiyama. Communication reduction with risk estimate for multiple robotic system. In *IEEE ICRA*, pages 2864–2869, 1994.

[Fukuda *et al.*, 1990] T. Fukuda, Y. Kawauchi, and H. Asama. Analysis and evaluation of cellular robotics (CEBOT) as a distributed intelligent system by communication amount. In *IEEE/RSJ IROS*, pages 827–834, 1990.

[Gage, 1993] D. Gage. How to communicate to zillions of robots. In *Mobile Robots VIII, SPIE*, pages 250–257, 1993.

[Genesereth *et al.*, 1986] M. R. Genesereth, M. L. Ginsberg, and J. S. Rosenschein. Cooperation without communication. In *Proc. AAAI*, pages 51–57, 1986.

[Genovese *et al.*, 1992] V. Genovese, P. Dario, R. Magni, and L. Odetti. Self organizing behavior and swarm inteligence in a pack of mobile miniature robots in search of pollutants. In *IEEE/RSJ IROS*, pages 1575–1582, 1992.

[Georgeff, 1983] M. Georgeff. Communication and interaction in multi-agent planning. In *Proc. AAAI*, pages 125–129, 1983.

[Georgeff, 1984] M. Georgeff. A theory of action for multi-agent planning. In *Proc. AAAI*, pages 121–125, 1984.

[Goldberg, 1989] D. Goldberg. *Genetic Algorithms in search, optimization, and machine learning.* Addison Wesley, 1989.

[Goss and Deneubourg, 1992] S. Goss and J. Deneubourg. Harvesting by a group of robots. In *Proc. European Conference on Artificial Life*, 1992.

[Grossman, 1988] D. Grossman. Traffic control of multiple robot vehicles. *IEEE Journal of Robotics and Automation*, 4:491–497, 1988.

[Hackwood and Beni, 1991] S. Hackwood and G. Beni. Self-organizing sensors by deterministic annealing. In *IEEE/RSJ IROS*, pages 1177–1183, 1991.

[Hackwood and Beni, 1992] S. Hackwood and G. Beni. Self-organization of sensors for swarm intelligence. In *IEEE ICRA*, pages 819–829, 1992.

[Hackwood and Wang, 1988] S. Hackwood and J. Wang. The engineering of cellular robotic systems. In *IEEE International Symposium on Intelligent Control*, pages 70–75, 1988.

[Hara *et al.*, 1995] F. Hara, Y. Yasui, and T. Aritake. A kinematic analysis of locomotive cooperation for two mobile robots along a genereal wavy road. In *IEEE ICRA*, pages 1197–1204, 1995.

[Hashimoto and Oba, 1993] M. Hashimoto and F. Oba. Dynamic control approach for motion coordination of multiple wheeled mobile robots transporting a single object. In *IEEE/RSJ IROS*, pages 1944–1951, 1993.

[Hert and Lumelsky, 1995] S. Hert and V. Lumelsky. Moving multiple tethered robots between arbitrary configurations. In *IEEE/RSJ IROS*, pages 280–285, 1995.

[Hewitt *et al.*, 1973] C. Hewitt, P. Bishop, I. Greif, B. Smith, T. Matson, and R. Steiger. Actor induction and meta-evaluation. In *ACM Symposium on Principles of Programming Languages*, pages 153–168, 1973.

[Hewitt, 1993] C. Hewitt. Toward an open systems architecture. In *Information Processing 89. Proceedings of the IFIP 11th World Computer Congress*, pages 389–92, 1993.

[Hodgins and Brogan, 1994] J. Hodgins and D. Brogan. Robot herds: Group behaviors for systems with significant dynamics. In *Proc. A-Life IV*, 1994.

[Ichikawa *et al.*, 1993] S. Ichikawa, F. Hara, and H. Hosokai. Cooperative route-searching behavior of multi-robot system using hello-call communiction. In *IEEE/RSJ IROS*, pages 1149–1156, 1993.

[Ishida *et al.*, 1991] Y. Ishida, I. Endo, and A. Matsumoto. Communication and cooperation in an autonomous and decentralized robot system. In *IFAC int. Symp. on Distributed Intelligent Systems*, pages 299–304, 1991.

[Ishida *et al.*, 1994] Y. Ishida, H. Asama, S. Tomita, K. Ozaki, A. Matsumoto, and I. Endo. Functional complement by cooperation of multiple autonomous robots. In *IEEE ICRA*, pages 2476–2481, 1994.

[Jin *et al.*, 1994] K. Jin, P. Liang, and G. Beni. Stability of synchronized distributed control of discrete swarm structures. In *IEEE ICRA*, pages 1033–1038, 1994.

[Johnson and Bay, 1994] P. J. Johnson and J. S. Bay. Distributed control of autonomous mobile robot collectives in payload transportation. Technical report, Virginia Polytechnic Institute and State University, Bradley Dept. of Elec. Engr., 1994.

[Kaelbling, 1993] L. P. Kaelbling. *Learning in Embedded Systems.* MIT Press, 1993.

[Kato *et al.*, 1992] S. Kato, S. Nishiyama, and J. Takeno. Coordinating mobile robots by applying traffic rules. In *IEEE/RSJ IROS*, pages 1535–1541, 1992.

[Kawauchi *et al.*, 1993a] Y. Kawauchi, M. Inaba, and T. Fukuda. A principle of distributed decision making of cellular robotic system (CEBOT). In *IEEE ICRA*, volume 3, pages 833–838, 1993.

[Kawauchi *et al.*, 1993b] Y. Kawauchi, M. Inaba, and T. Fukuda. A relation between resource amount and system performance of the cellular robotic system. In *IEEE/RSJ IROS*, pages 454–459, 1993.

[Kitano, 1994] H. Kitano. personal communication, 1994.

[Kleinrock, 1995] L. Kleinrock. Nomadic computing - an opportunity. *Computer Communications Review*, Computer Communications Review 1995.

[Korf, 1992] R. Korf. A simple solution to pursuit games. In *Proc. 11th International Workshop on Distributed Artificial Intelligence*, 1992.

[Koza, 1990] J. Koza. *Genetic Programming: On the Programming of Computers By the Means of Natural Selection.* MIT Press, 1990.

[Kube and Zhang, 1992] C. R. Kube and H. Zhang. Collective robotic intelligence. In *Proc. Simulation of Adaptive Behavior*, pages 460–468, 1992.

[Kube and Zhang, 1993] C. R. Kube and H. Zhang. Collective robotics: From social insects to robots. *Adaptive Behavior*, 2(2):189–219, 1993.

[Kube and Zhang, 1994] C. R. Kube and H. Zhang. Stagnation recovery behaviours for collective robotics. In *IEEE/RSJ IROS*, pages 1883–1890, 1994.

[Kube *et al.*, 1993] C. R. Kube, H. Zhang, and X. Wang. Controlling collective tasks with an ALN. In *IEEE/RSJ IROS*, pages 289–293, 1993.

[Kuniyoshi *et al.*, 1994a] Y. Kuniyoshi, N. Kita, S. Rougeaux, S. Sakane, M. Ishii, and M. Kakikura. Cooperation by observation - the framework and basic task patterns. In *IEEE ICRA*, pages 767–774, 1994.

[Kuniyoshi *et al.*, 1994b] Y. Kuniyoshi, J. Riekki, M. Ishii, S. Rougeaux, N. Kita, S. Sakane, and M. Kakikura. Vision-based behaviors for multi-robot cooperation. In *IEEE/RSJ IROS*, pages 925–931, 1994.

[Kurabayashi *et al.*, 1995] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. An algorithm of dividing a work area to multiple mobile robots. In *IEEE/RSJ IROS*, pages 286–291, 1995.

[Kurazume and Nagata, 1994] R. Kurazume and S. Nagata. Cooperative positioning with multiple robots. In *IEEE ICRA*, pages 1250–1257, 1994.

[Latombe, 1991] J. Latombe. *Robot Motion Planning*. Kluwer Academic, Boston, MA, 1991.

[LePape, 1990] C. LePape. A combination of centralized and distributed methods for multi-agent planning and scheduling. In *IEEE ICRA*, pages 488–493, 1990.

[Levy and Rosenschein, 1991] R. Levy and J.S. Rosenschein. A game theoretic approach to distributed artificial intelligence and the pursuit problem. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 129–146, 1991.

[Lewis and Bekey, 1992] M.A. Lewis and G.A. Bekey. The behavioral self-organization of nanorobots using local rules. In *IEEE/RSJ IROS*, pages 1333–1338, 1992.

[Liang and Beni, 1995] P. Liang and G. Beni. Robotic morphogenesis. In *IEEE ICRA*, pages 2175–2180, 1995.

[Lin and Hsu, 1995] F.-C. Lin and J. Y.-J. Hsu. Cooperation and deadlock-handling for an object-sorting task in a multi-agent robotic system. In *IEEE ICRA*, pages 2580–2585, 1995.

[Littman, 1994] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the International Machine Learning Conference*, pages 157–163, 1994.

[Lueth and Laengle, 1994] T.C. Lueth and T. Laengle. Task description, decomposition and allocation in a distributed autonomous multi-agent robot system. In *IEEE/RSJ IROS*, pages 1516–1523, 1994.

[Ma *et al.*, 1994] S. Ma, S. Hackwood, and G. Beni. Multi-agent supporting systems (MASS): Control with centralized estimator of disturbance. In *IEEE/RSJ IROS*, pages 679–686, 1994.

[Mataric *et al.*, 1995] M.J. Mataric, M. Nilsson, and K.T. Simsarian. Cooperative multi-robot box-pushing. In *IEEE/RSJ IROS*, pages 556–561, 1995.

[Mataric, 1992a] M. J. Mataric. Designing emergent behaviors: From local interactions to collective intelligence. In J.-A. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats 2, Second International Conference on Simulation of Adaptive Behavior (SAB-92)*, pages 432–441. MIT Press, 1992.

[Mataric, 1992b] M. J. Mataric. Distributed approaches to behavior control. In *SPIE - Sensor Fusion V*, volume 1828, pages 373–382, 1992.

[Mataric, 1992c] M. J. Mataric. Minimizing complexity in controlling a mobile robot population. In *IEEE ICRA*, pages 830–835, May 1992.

[Mataric, 1993] M. J. Mataric. Kin recognition, similarity, and group behavior. In *Fifteenth Annual Cognitive Science Society Conference*, pages 705–710. Lawrence Erlbaum Associates, June 1993.

[Mataric, 1994a] M. Mataric. *Interaction and Intelligent Behavior*. PhD thesis, MIT, EECS, May 1994.

[Mataric, 1994b] M. Mataric. Reward functions for accelerated learning. In *Proceedings of the International Machine Learning Conference*, pages 181–189, 1994.

[Matsumoto *et al.*, 1990] A. Matsumoto, H. Asama, Y. Ishida, K. Ozaki, and I. Endo. Communication in the autonomous and decentralized robot system: ACTRESS. In *IEEE/RSJ IROS*, pages 835–840, 1990.

[McFarland, 1994] D. McFarland. Towards robot cooperation. In *Proc. Simulation of Adaptive Behavior*, 1994.

[Mehregany *et al.*, 1988] M. Mehregany, K.J. Gabriel, and W.S. Trimmer. Integrated fabrication of polysilicon mechanisms. *IEEE Trans. Electron Devices*, 35(6):719–723, 1988.

[Merriam-Webster, 1963] Merriam-Webster. *Webster's 7th Collegiate Dictionary*. Merriam-Webster, Inc., 1963.

[Milios and Wilkes, 1995] G. Dudek M. Jenkin E. Milios and D. Wilkes. Experiments in sensing and communication for robot convoy navigation. In *IEEE/RSJ IROS*, pages 268–273, 1995.

[Miller and Cliff, 1994] G. F. Miller and D. Cliff. Protean behavior in dynamic games: Arguments for the co-evolution of pursuit-evasion tactics. In D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *Proc. Simulation of Adaptive Behavior*, 1994.

[Mitsumoto *et al.*, 1995] N. Mitsumoto, T. Fukuda, K. Shimojina, and A. Ogawa. Micro autonomous robotic system and biologically inspired immune swarm strategy as a multi agent robotic system. In *IEEE ICRA*, pages 2187–2192, 1995.

[Nicolis and Prigogine, 1977] G. Nicolis and I. Prigogine. *Self-Organization in Nonequilibrium Systems*. Wiley-Interscience, 1977.

[Noreils and Recherche, 1991] F. Noreils and A. Recherche. Adding a man/machine interface to an architecture for mobile robots. In *IEEE/RSJ IROS*, 1991.

[Noreils, 1990] F. Noreils. Integrating multirobot coordination in a mobile robot control system. In *IEEE/RSJ IROS*, pages 43–49, 1990.

[Noreils, 1992a] F. R. Noreils. Coordinated protocols: An approach to formalize coordination between mobile robots. In *IEEE/RSJ IROS*, pages 717–724, July 1992.

[Noreils, 1992b] F. R. Noreils. Multi-robot coordination for battlefield strategies. In *IEEE/RSJ IROS*, pages 1777–1784, July 1992.

[Noreils, 1993] F. R. Noreils. Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *The International Journal of Robotics Research*, 12(1), February 1993.

[Numaoka, 1993] C. Numaoka. Collective alteration of strategic types with delayed global information. In *IEEE/RSJ IROS*, pages 1077–1084, 1993.

[Ota *et al.*, 1995] J. Ota, N. Miyata, T. Arai, E. Yoshida, D. Kurabayashi, and J. Sasaki. Transferring and regrasping a large object by cooperation of multiple mobile robots. In *IEEE/RSJ IROS*, pages 543–548, 1995.

[Ozaki *et al.*, 1993] K. Ozaki, H. Asama, Y. Ishida, A. Matsumoto, K. Yokota, H. Kaetsu, and I. Endo. Synchronized motion by multiple mobile robots using communication. In *IEEE/RSJ IROS*, pages 1164–1169, July 1993.

[Parker, 1992] L. E. Parker. Adaptive action selection for cooperative agent teams. In *Second Annual International Conference on Simulation of Adaptive Behavior*, pages 442–450. MIT Press, December 1992.

[Parker, 1993] L. E. Parker. Designing control laws for cooperative agent teams. In *IEEE ICRA*, volume 3, pages 582–587, 1993.

[Parker, 1994a] L. E. Parker. ALLIANCE: an architecture for fault tolerant, cooperative control of heterogeneous mobile robots. In *IEEE/RSJ IROS*, pages 776–783, 1994.

[Parker, 1994b] L. E. Parker. *Heterogeneous Multi-Robot Cooperation*. PhD thesis, MIT EECS Dept., February 1994.

[Parker, 1995] L. E. Parker. The effect of action recognition and robot awareness in cooperative robotic teams. In *IEEE/RSJ IROS*, pages 212–219, 1995.

[P.J. Heffernan, 1992] S. Schirra P.J. Heffernan. Approximate decision algorithms for point set congruence. In *8th Annual Compuational Geometry*, pages 93–101, 1992.

[Premvuti and Yuta, 1990] S. Premvuti and S. Yuta. Consideration on the cooperation of multiple autonomous mobile robots. In *IEEE/RSJ IROS*, pages 59–63, 1990.

[Reynolds, 1987] C. W. Reynolds. Flocks, herds and schools: a distributed behavioural model. *Computer Graphics*, 21(4):71–87, 1987.

[Reynolds, 1992] C. Reynolds. An evolved, vision-based behavioral model of coordinated group motion. In *Proc. Simulation of Adaptive Behavior*, 1992.

[Reynolds, 1994] C. Reynolds. Competition, coevolution and the game of tag. In *Proc. A-Life IV*, 1994.

[Rosenschein and Genesereth, 1985] J.S. Rosenschein and M.R. Genesereth. Deals among rational agents. In *Proc. Intl. Joint Conf. Artificial Intelligence*, pages 91–99, 1985.

[Rosenschein and Zlotkin, 1994] J.S. Rosenschein and G. Zlotkin. *Rules of Encounter:designing conventions for automated negotiation among computers*. MIT Press, 1994.

[Rosenschein, 1982] J.S. Rosenschein. Synchronization of multi-agent plans. In *Proc. AAAI*, pages 115–119, 1982.

[Rude, 1994] M. Rude. Cooperation of mobile robots by event transforms into local space-time. In *IEEE/RSJ IROS*, pages 1501–1507, 1994.

[Rus *et al.*, 1995] D. Rus, B. Donald, and J. Jennings. Moving furniture with teams of autonomous robots. In *IEEE/RSJ IROS*, pages 235–242, 1995.

[Sasaki *et al.*, 1995] J. Sasaki, J. Ota, E. Yoshida, D. Kurabayashi, and T. Arai. Cooperating grasping of a large object by multiple mobile robots. In *IEEE ICRA*, pages 1205–1210, 1995.

[Sen *et al.*, 1994] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proc. AAAI*, pages 426–431, 1994.

[Shoham and Tennenholtz, 1992] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial agent societies (preliminary report). In *Proc. AAAI*, pages 276–281, 1992.

[Singh and Fujimura, 1993] K. Singh and K. Fujimura. Map making by cooperating mobile robots. In *IEEE ICRA*, volume 2, pages 254–259, 1993.

[Smith, 1980] R. Smith. The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Trans. Computers*, pages 1104–1113, 1980.

[Steels, 1990] L. Steels. Cooperation between distributed agents through self-organization. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 175–195, 1990.

[Steels, 1994] L. Steels. A case study in the behavior-oriented design of autonomous agents. In *Proc. Simulation of Adaptive Behavior*, 1994.

[Stilwell and Bay, 1993] D. J. Stilwell and J. S. Bay. Toward the development of a material transport system using swarms of ant-like robots. In *IEEE ICRA*, volume 6, pages 766–771, 1993.

[Sugie *et al.*, 1995] H. Sugie, Y. Inagaki, S. Ono, H. Aisu, and T. Unemi. Placing objects with multiple mobile robots – mutual help using intention inference. In *IEEE ICRA*, pages 2181–2186, 1995.

[Sugihara and Suzuki, 1990] K. Sugihara and I. Suzuki. Distributed motion coordination of multiple mobile robots. In *Proc. IEEE International Symposium on Intelligent Control*, 1990.

[Suzuki *et al.*, 1995] T. Suzuki, H. Asama, A. Uegaki, S. Kotosaka, T. Fujita, A. Matsumoto, H. Kaetsu, and I. Endo. An infra-red sensory system with local communication for cooperative multiple mobile robots. In *IEEE/RSJ IROS*, volume 1, pages 220–225, 1995.

[Tan, 1993] M. Tan. Multi-agent reinforcement learning: independent vs. cooperative agents. In *Proceedings of the International Machine Learning Conference*, 1993.

[Tsetlin, 1964] M.L. Tsetlin. *Finite Automata and Modeling the Simplest Forms of Behavior*. PhD thesis, V.A. Steklov Mathematical Institute, 1964.

[Tung and Kleinrock, 1993] B. Tung and L. Kleinrock. Distributed control methods. In *Proceedings of the 2nd International Symposium on High Performance Distr ibuted Computing*, pages 206–215, 1993.

[Tung, 1994] Y-C. Tung. *Distributed Control Using Finite State Automata*. PhD thesis, UCLA Computer Science Department, 1994.

[Ueyama and Fukuda, 1993a] T. Ueyama and T. Fukuda. Knowledge acquisition and distributed decision making. In *IEEE ICRA*, volume 3, pages 167–172, 1993.

[Ueyama and Fukuda, 1993b] T. Ueyama and T. Fukuda. Self-organization of cellular robots using random walk with simple rules. In *IEEE ICRA*, volume 3, pages 595–600, 1993.

[Ueyama *et al.*, 1993a] T. Ueyama, T. Fukuda, F. Arai, Y. Kawauchi, Y. Katou, S. Matsumura, and T. Uesugi. Communication architecture for cellular robotic system. *JSME International Journal, Series C*, 36:353–360, 1993.

[Ueyama *et al.*, 1993b] T. Ueyama, T. Fukuda, F. Arai, T. Sugiura, A. Sakai, and T. Uesugi. Distributed sensing, control and planning - cellular robotics approach. In *IMACS*, pages 433–438. Elsevier Science Publ. (North-Holland), 1993.

[VanLehn, 1991] K. VanLehn, editor. *Architectures for Intelligence: The 22nd Carnegie Mellon Symposium on Cognition*. Lawrence Erlbaum Associates, 1991.

[Wang and Beni, 1988] J. Wang and G. Beni. Pattern generation in cellular robotic systems. In *IEEE International Symposium on Intelligent Control*, pages 63–69, 1988.

[Wang and Beni, 1990] J. Wang and G. Beni. Distributed computing problems in cellular robotic systems. In *IEEE/RSJ IROS*, pages 819–826, 1990.

[Wang and Premvuti, 1994] J. Wang and S. Premvuti. Resource sharing in distributed robotic systems based on a wireless medium access protocol (CSMA/CD-W). In *IEEE/RSJ IROS*, pages 784–791, 1994.

[Wang and Premvuti, 1995] J. Wang and S. Premvuti. Distributed traffic regulation and control for multiple autonomous mobile robots operating in discrete space. In *IEEE ICRA*, pages 1619–1624, 1995.

[Wang *et al.*, 1994] Z.-D. Wang, E. Nakano, and T. Matsukawa. Cooperating multiple behavior-based robots for object manipulation. In *IEEE/RSJ IROS*, pages 1524–1531, 1994.

[Wang *et al.*, 1995] J. Wang, S. Premvuti, and A. Tabbara. A wireless medium access protocol ( csma/cd-w ) for mobile robot based distributed robotic system. In *IEEE ICRA*, pages 2561–2566, 1995.

[Wang, 1991] J. Wang. Fully distributed traffic control strategies for many-AGV systems. In *IEEE/RSJ IROS*, pages 1199–1204, 1991.

[Wang, 1993] J. Wang. DRS operating primitives based on distributed mutual exclusion. In *IEEE/RSJ IROS*, pages 1085–1090, 1993.

[Wang, 1994] J. Wang. On sign-board based inter-robot communication in distributed robotic systems. In *IEEE ICRA*, pages 1045–1050, 1994.

[Wang, 1995] J. Wang. Operating primitives supporting traffic regulation and control of mobile robots under distributed robotic systems. In *IEEE ICRA*, pages 1613–1618, 1995.

[Weiser, 1993] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):74–84, 1993.

[Werner and Dyer, 1992] G. Werner and M. Dyer. Evolution of herding behavior in artificial animals. In *Proc. Simulation of Adaptive Behavior*, 1992.

[Whitehead, 1991] S. Whitehead. A complexity analysis of cooperative mechanisms in reinforcement learning. In *Proc. AAAI*, pages 607–613, 1991.

[Wilson, 1971] E.O. Wilson. *The insect societies*. Harvard University Press, 1971.

[Yamaguchi and Arai, 1994] H. Yamaguchi and T. Arai. Distributed and autonomous control method for generating shape of multiple mobile robot group. In *IEEE/RSJ IROS*, pages 800–807, 1994.

[Yanco and Stein, 1992] H. Yanco and L. Stein. An adaptive communication protocol for cooperating mobile robots. In *Proc. Simulation of Adaptive Behavior*, pages 478–485, 1992.

[Yates, 1987] F.E. Yates, editor. *Self-Organizing Systems: The Emergence of Order*. Plenum Press, 1987.

[Yeung and Bekey, 1987] D. Yeung and G. Bekey. A decentralized approach to the motion planning problem for multiple mobile robots. In *IEEE ICRA*, pages 1779–1784, 1987.

[Yokota *et al.*, 1994] K. Yokota, T. Suzuki, H. Asama, A. Masumoto, and I. Endo. A human interface system for the multi-agent robotic system. In *IEEE ICRA*, pages 1039–1044, 1994.

[Yoshida *et al.*, 1994] E. Yoshida, T. Arai, J. Ota, and T. Miki. Effect of grouping in local communication system of multiple mobile robots. In *IEEE/RSJ IROS*, pages 808–815, 1994.

[Yoshida *et al.*, 1995a] E. Yoshida, M. Yamamota, T. Arai, J. Ota, and D. Kurabayashi. A design method of local communication area in multiple mobile robot system. In *IEEE ICRA*, pages 2567–2572, 1995.

[Yoshida *et al.*, 1995b] E. Yoshida, M. Yamamoto, T. Arai, J. Ota, and D. Kurabayashi. A design method of local communication range in multiple mobile robot system. In *IEEE/RSJ IROS*, pages 274–279, 1995.

[Yuta and Premvuti, 1992] S. Yuta and S. Premvuti. Coordinating autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots. In *Proc. of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems, Raleigh, NC, July 7-10, 1992*, pages 1566–1574, July 1992.

# ROCI: A Distributed Framework for Multi-Robot Perception and Control

Luiz Chaimowicz, Anthony Cowley, Vito Sabella, Camillo J. Taylor

GRASP Laboratory – University of Pennsylvania, Philadelphia, PA, USA, 19104

{chaimo, acowley, vsabella, cjtaylor}@grasp.cis.upenn.edu

**Abstract**

This paper presents ROCI, a framework for developing applications for multi-robot teams. In ROCI, each robot is considered a node which contains several modules and may export different types of services and capabilities to other nodes. Each node runs a kernel that mediates the interactions of the robots in a team. This kernel keeps an updated database of all nodes and the functionalities that they export. Multi-robot applications can be built and changed dynamically, connecting modules that may be running in different nodes over the network. As an example, we present an obstacle avoidance task implemented using our framework and also discuss the use of ROCI in a multi-robot scenario.

## 1 Introduction

As sensors, actuators, microprocessors and wireless networks become cheaper and more ubiquitous it has become increasingly attractive to consider employing teams of small robots to tackle various sensing and manipulation tasks. In order to exploit the full capabilities of these teams, we need to develop effective models and methods for programming distributed ensembles of sensors and actuators.

Applications for distributed dynamic robotic teams require a different programming model than the one employed for most traditional robotic applications. In the traditional model, the programmer is faced with the task of developing software for a single processor interacting with a prescribed set of sensors and actuators. He or she can typically assume that the configuration of the target system is completely specified before the first line of code is written. On the other hand, when developing code for multi-robot dynamic teams, we must account for the fact that the number and type of robots available at runtime cannot be predicted. We expect to operate in an environment where robots will be added and removed continuously and unpredictably. Further, we must expect an environment where the robots will have heterogeneous capabilities; for example, some may be equipped with camera systems, others with range sensors or specialized actuators, some agents may be stationary while others may offer specialized computational resources. This implies that the program must be able to identify and marshal all of the resources required to carry out the specified task automatically.

This paper presents ROCI (Remote Objects Control Interface), a self-describing, objected oriented, strongly typed programming framework that allows the development of robust applications for dynamic multi-robot teams. The building blocks of ROCI applications are self-contained, reusable modules. Basically, a module encapsulates a process which acts on data available on the module's inputs and presents its results as outputs. Thus, complex tasks can be built connecting inputs and outputs of specific modules. These connections are made through a pin architecture that provides a strongly typed, network transparent communication framework. A good analogy is to consider each of these modules as an integrated circuit (IC), that has inputs and outputs and does some processing. Complex circuits can be built wiring several ICs, and individual ICs can be reused in different circuits.

The core control element in the ROCI architecture is the ROCI kernel. There is a copy of the kernel running in every entity that is part of the ROCI network (robots, remote sensors, etc.). These entities are considered ROCI nodes and any information acquired or processed in a certain node can be exposed to others. The kernel is responsible for managing network and maintaining an updated database of all the nodes and services in the ROCI network. The kernel is also responsible for handling module and task allocation and injection. It allows applications to be specified and executed dynamically, by connecting available pins and transferring code libraries to the nodes.

ROCI incorporates some features that are already present in modern distributed software environments such as the Open Agent Architecture [7] and the Grid Computing [4]. Some frameworks for cooperative robotics have already included advances such as hier-

archical and reusable objects [1], distributed sensing and actuation capabilities [5], abstraction and modularity [8], and task decomposition [9]. The use of modern programming languages [2] and graphical interfaces for task specification [6] are also important advances. But, in spite of that, most of the programming architectures for distributed robots still rely on traditional programming models and are specific for certain types of robots and control architectures. Thus, we believe that ROCI will certainly be a valuable contribution to the multi-robot programming field.

This paper is organized as follows: the next section describes the ROCI framework, giving details about its architecture and its main features. Section 3 shows the implementation of an obstacle avoidance task using ROCI and Section 4 describes the use of ROCI in a multi-robot scenario. Finally, in Section 5 we conclude the paper and discuss the next steps in this work.

## 2 ROCI Architecture

### 2.1 Introduction

ROCI is a dynamic, self-describing, object-oriented, strongly typed programming framework for distributed sensors and actuators. It provides programmers with a network transparent framework of strongly typed modules - assemblies of metadata, byte code, and machine code that can consume, process and produce information. ROCI modules are injectable (they can be automatically downloaded and started on a remote machine), reusable, browseable, support automatic configuration via XML, and provide strongly typed pin based communications. These features, coupled with a dynamic database of available nodes and network services, allows a programmer to write code that utilizes networks of robots as resources instead of independent machines.

ROCI is developed in C# using the Microsoft .NET platform. Modules are not limited to this language however, and several of our own modules are written in mixtures of C# and C++. The system makes use of XML to provide basic configuration options, and object reflection to enforce type safety and autodiscovery.

### 2.2 Modules and Tasks

The building blocks of a ROCI application are ROCI modules. A module is a computational block that encapsulates a process, taking an input, performing some operation on it, and making the result of that operation available as an output. There is no specification in the code relating to where input should come from or where output should go; the only specification is the type of data this computational block deals with. To create a new module, the application developer has to decide on the types of data to be input and output and then inherit from the ROCI module parent class, implementing a few virtual functions related to the allocation and de-allocation of resources required by the new module. Since the modules are designed with no knowledge of their runtime environment, they can be wired into the ROCI network with a great deal of flexibility.

ROCI modules are further organized into tasks. A ROCI task is a way of describing an instance of a collection of ROCI modules to be run on a single node and how they interact at runtime. Tasks represent a family of modules that work together to accomplish some end goal – a chain of building blocks that transforms input data through intermediate forms and into a useful output. A task can be defined in an XML file which outlines the modules that are needed to achieve the goal, and the connectivity between these modules. Tasks can also be defined and changed dynamically, by starting new modules and connecting them with the outputs and inputs of other modules. As will be explained in the next section, the connection between modules is made using pins. Pins can connect modules within the same task on the same computer, between tasks on the same computer, or between two tasks on different computers. Figure 1 shows an example of this organization.

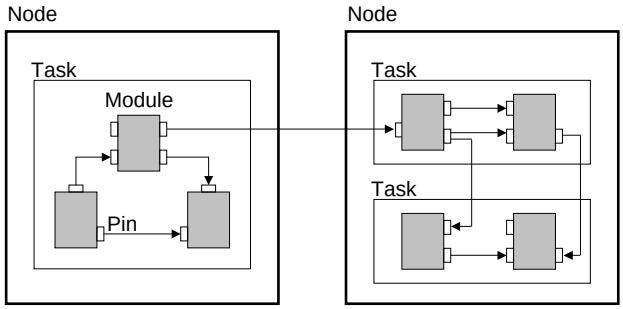



Figure 1: Roci Architecture: tasks are composed of modules and run inside nodes. Communication through pins can be seamlessly done between modules within the same task, modules in different tasks, or in different nodes.

### 2.3 Pin Based Communication

The wiring that connects ROCI modules is the pin communications architecture. Pin communications in ROCI are designed to be network transparent yet high performance. Basically, a pin provides the developer with an abstract communications endpoint. These endpoints can either represent a data producer or a data consumer. Pins in the system are nothing more than strongly typed fields of a module class, and thus

connecting a producer pin to a consumer pin is as simple as setting a reference to the producer in the consumer's field. Modules' pins are automatically exposed and discovered by the ROCI kernel through reflection – an important feature when programming dynamic networks.

Whenever a consumer pin registers itself to a producer pin, the ROCI subsystem determines whether the modules are within the same task domain. If they are, the consumer pin is assigned a reference to the producer pin. If not, ROCI creates a Remote Procedure Call channel and assigns a proxy reference to the consumer.

As the producer generates data it assigns it's pin an updated reference to the latest data. This assignment causes the pin to fire messages to all of the registered consumers in the network, alerting them to the availability of fresh data. A typical usage of this system is for a module to make a blocking call to one of its input pins which returns once the input pin has gotten new data from the output pin it is registered to. Alternatively, a module can ask its input pin to copy over whatever data is available immediately, whether it is new or not. Pin data is time stamped, allowing the consumers to determine how current their data is. Once all the consumers have completed processing their data, the managed environment in which ROCI runs automatically marks the data for garbage collection, freeing the programmer from any memory management issues which may arise with complex producer/consumer interconnections.

One can set a pin's input in two distinct ways that are each useful in different situations. On a lower level, pins can be connected dynamically during program execution. This can be accomplished by querying nodes on the ROCI network for available pins – usually with some type constraint – and may involve dynamically creating local pins to bind to the discovered remote pins. However, the simpler way of binding pins together is via the XML descriptions that define ROCI tasks.

Strongly typed pins enforce that only pins of the same type are connected to each other. The exchange of strongly typed objects instead of raw data eliminates potential software bugs increasing the robustness of the system. Robustness is also a consequence of the self-describing nature of pins. Since we can find out the exact type of a pin instance, we can dynamically guarantee that it will only be connected to a compatible pin.

### 2.4 ROCI Kernel

The kernel is the core control element of ROCI. The kernel manages the Remote Procedure Call (RPC) system, the real-time network database, module and task allocation and injection, and a Web Services like interface for remote monitoring and control. There is a copy of the kernel running in every entity that is part of the ROCI network (robots, remote sensors, etc.).

The RPC system provides interfaces for module management, injection and communication, as well as providing a web-based interface to the current status of the network. The real time database contains information on all of the modules, tasks and communications channels within the network. ROCI's database can be used to locate an appropriate sensor or actuator to solve a problem, find software modules that are needed in local computation, and identify computer utilization and congestion across the network. The database provides modules with a bird's eye view of the environment, allowing them to locate and utilize each and every hardware and software resource on the network.

It is important to mention that ROCI's kernel exposes its interfaces through Simple Object Access Protocol (SOAP), a cross-platform RPC standard. This standard allows non-ROCI programs and utilities to easily interoperate with and utilize the resources of the ROCI network.

### 2.5 ROCI Browser

The job of presenting this network of functionality to the user falls upon the ROCI Browser. The browser's job is to give a human user command and control over the network as well as the situational awareness necessary to make informed decisions about network operations. The ROCI network is presented hierarchically: the human operator can browse nodes on the network, tasks running on each node, the modules that make up each task, and even certain pins within those modules. The browser can be used to monitor the status of running tasks or even to tap into and display the outputs of pins for which display routines exist.

Using the browser, the user can also decide to start or stop a task running on any node on the network. When the user requests a task be started on a given node, the kernel running locally on that node first ascertains whether or not the proper versions of component modules are available locally (strong versioning is a useful feature of the .NET Framework). If they are not, it queries the network for a node that does have the modules in question and downloads them automatically. Once the byte code of the modules that comprise the task all exists locally, the task is loaded.

One of the important features of the ROCI architecture is that modules and pins are self-describing

entities. Thus, when the user browses through the tasks, he or she can immediately have a complete description of the modules and pins in use. Given this information, the browser can automatically start appropriate modules locally to tap into the remote data for visualization or processing purposes. This can be very useful for debugging purposes during development and for situational awareness during deployed execution.

## 3 Simple Obstacle Avoider in ROCI

As mentioned, an application in ROCI may be composed of multiple tasks. Tasks can be specified connecting several building blocks (modules), each one offering a specific service. The connections determine the data flow from one block's outputs to another block's inputs. In fact, these connections can be made seamlessly between modules in different tasks, even if they are running on different nodes.

To demonstrate this, we have developed and successfully executed a simple obstacle avoidance task using one of our ClodBuster robots equipped with an omnidirectional camera and IEEE 802.11b wireless network [1]. In this task, the robot's heading is computed based on a range map constructed from an omnidirectional edge image of the environment (Figure 2). The edge image and the heading direction can be displayed simultaneously by another task running on a remote computer.

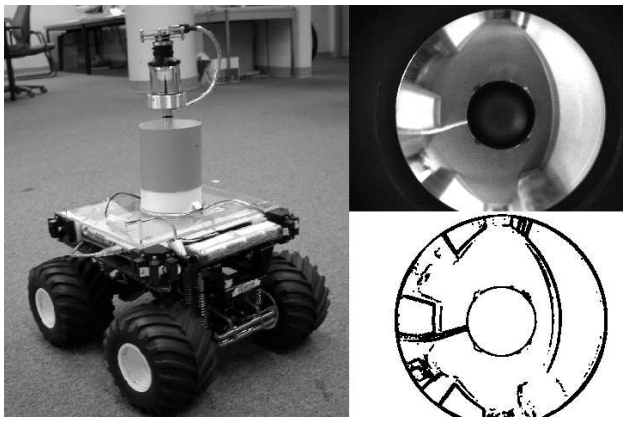


Figure 2: A Clodbuster robot and an image captured by the omnicam, before and after the edge extraction.

A ROCI diagram of the present application can be seen in Figure 3. It is composed of three tasks, two running in the robot and one running in a remote computer. The main task is the Obstacle Avoider that is comprised of 5 modules: the OmniCam captures an image and exports it to other modules through a video pin. The Edge Detector processes this image and makes it available for the Range Mapper that computes a desired bearing for the robot. The Range Mapper also receives calibration parameters from the OmniCam and exports a video pin containing the input image with the heading direction highlighted. The bearing is exported and used by the Robot Controller module which generates inputs to the Grasp Board that is the interface to the servo motors. Running in the same node, there is also a task that reduces the resolution and subsamples the video stream exported by the Range Mapper so it can be better transmitted over the network. Finally, there is a Video Preview running on another computer that allows a remote operator to observe the video broadcast by the robot.

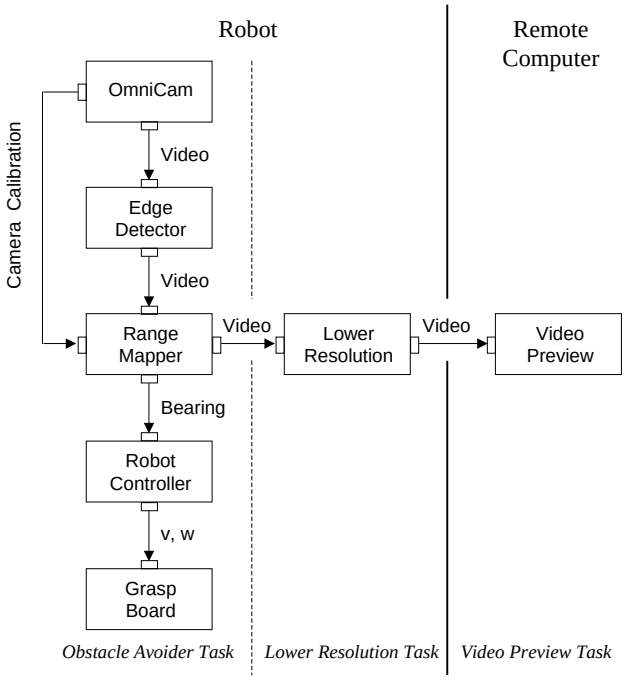



Figure 3: Diagram of the tasks and modules for the obstacle avoider implemented in ROCI.

We need to reinforce that each one of these modules is self-contained and can run independently of the others. Consequently, different applications can be specified and performed reusing some of these modules and adding new ones. For example, the Omnicam module could be used in a tracking task connected to some feature extractor module or the Edge Detector could receive input from a regular camera in a different task specification. The idea is to have a library of modules that can be executed by robots according to their capabilities. Also, during execution, connections between modules can be dynamically made or

changed allowing modules to receive different inputs from different sources. For example, in the application described above, a remote operator could repin the input of the Lower Resolution module to any one of the modules that outputs a video pin. The operator has this information, since modules and pins are self-describing and the network database is continuously updated. Consequently, the operator could observe images directly from the Omnicam module or from any other module that is exporting a video pin at the moment.

As will be discussed in the next section, these capabilities are especially important in multi-robot applications in which the number of robots, communication and sensor constraints may change dynamically during execution.

## 4 Multi-Robot Scenario

Let us consider a multi-robot task in which $n$ robots must perform a visual reconnaissance of a certain area. Each robot $i$ is equipped with a GPS and an omnidirectional camera and its objective is to send its position $\boldsymbol{x_i}$ and an image captured from that position to a base station. In ROCI, this task could be specified by three modules, as shown in Figure 4. The Camera captures images and exports them as a video pin. It also outputs the camera parameters which are not being used by any module at this moment. There is another module to get the robot's GPS coordinates and a third one that simply processes the video and the robot's position and sends it to the base station. Initially, each robot can act independently from its teammates (it is a loosely coupled task), but the ROCI kernel running in each robot continuously updates its database, keeping track of the other nodes on the network.

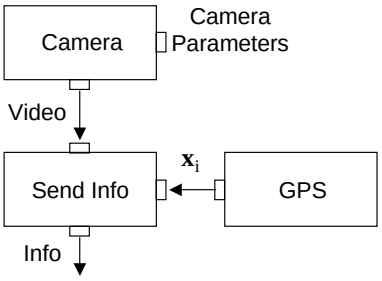



Figure 4: Diagram of a reconnaissance task running in each robot.

Now, suppose that one of the robots (for example, robot $k$) loses its GPS information. In order to continue performing its task, it should find a way of replacing that information source. Since the other robots in the team may still be able to compute their positions, robot $k$ can rely on a cooperative localization scheme to localize itself [3]. The cooperative localization works as follows: each robot $j$ in the neighborhood of $k$ computes the position of the other robots in its field of view based on its GPS and camera image and exports this information. Then, based on the position estimates received from its neighbors, robot $k$ will be able to localize itself.

Using the ROCI framework, it is easy for robot $k$ to start the cooperative localization dynamically. First of all, it has an updated list of the other robots in the network that can perform localization since the ROCI kernel maintains this information. So, robot $k$ will be able to inject a new localizer module in some of its teammates and dynamically pin it to the modules that are already running to get the information that it needs (Figure 5a). The localizer will receive information from the camera (image and calibration parameters) and the GPS and export the estimated position of all the robots that are visible $(\boldsymbol{x_{j1}}, \boldsymbol{x_{j2}}, \ldots, \boldsymbol{x_{jk}})$. Robot $k$ will also start running a local module called Position Estimation to get the position estimates from its neighbors and compute its position, automatically repinning the input of the Send Info module to the output of this new module, as shown in Figure 5b. Thus, this dynamic reconfiguration allows the robot that lost its GPS to continue executing its task.

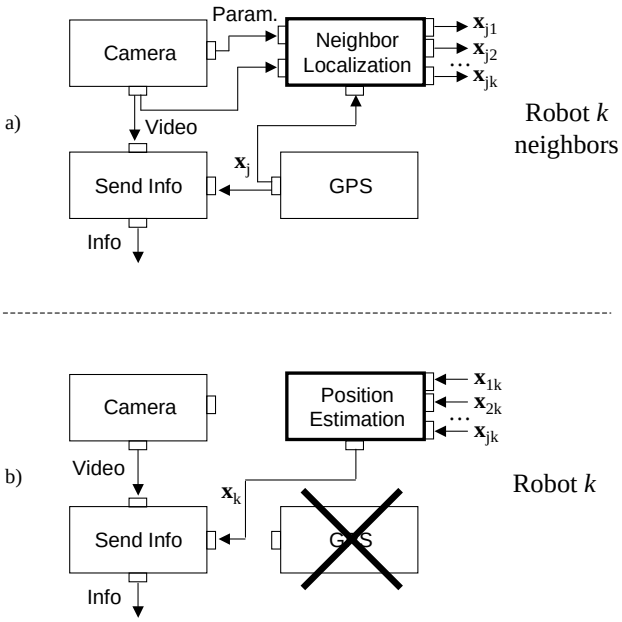



Figure 5: Scenario if robot $k$ loses its GPS: a) robot $k$ injects a new localizer module on its neighbors and b) robot $k$ starts a local module and dynamically repins its other modules to use the new information.

It is important to note that some aspects of the cooperative task were not detailed in the description above. For example, we did not define exactly the concept of neighborhood in terms of sensing and communication in this paper. Also, we did not give any details about the cooperative localization or about the controllers and coordination techniques that should be used in this task. These points should be specified in the implementation of the task, but our main objective here is to present the ROCI framework, showing how it allows multi-robot teams to adapt to dynamic changes that typically occur during the execution of cooperative tasks.

## 5 Conclusion

In this paper we presented ROCI, a programming framework for distributed ensembles of sensors and actuators. Applications in ROCI are composed of tasks and can be built dynamically by connecting several modules, which gives a great flexibility to the programmer. Modules are completely self-contained and can be reused in different tasks and applications. A pin architecture is used to connect modules. These connections can be made seamlessly between modules in different tasks, even if they are running on different nodes, creating a network transparent programming environment. The ROCI framework is strongly typed, allowing the development of more robust yet high performance applications. Two of the ROCI's main features are the ROCI kernel and the self-describing nature of modules and pins. Together, they allow the creation of an updated view of network nodes, services, and data, providing situational awareness for users and applications. This is a key requirement for programming dynamic distributed multi-robot teams.

Our present and future work is directed towards implementing several multi-robot applications using the ROCI framework. Under the DARPA's MARS project, we are developing a new team of robots (both aerial and terrestrial) that will be fully programmed and controlled using ROCI. Several multi-robot capabilities are being developed for this team, such as outdoors navigation, cooperative localization, stereo obstacle avoidance and communication sensitive behaviors. Also, our multi-robot team will have to interact with other robots programmed in different frameworks, more specifically Player [5] and MissionLab [6]. We are working on a common interface between ROCI and these systems that will use SOAP and XML to exchange data between different platforms. This project will provide an excellent test bed for the ROCI framework, and we expect to have important results very soon.


### Acknowledgment

This work was in part supported by: DARPA MARS NBCH1020012 and NSF ITR (ANTIDOTE) CCR02-05336.


## References


[1] R. Alur, A. Das, J. Esposito, R. Fierro, G. Grudic, Y. Hur, V. Kumar, I. Lee, J. Ostrowski, G. Pappas, B. Southall, J. Spletzer, and C. Taylor. A framework and architecture for multirobot coordination. In D. Rus and S. Singh, editors, *Experimental Robotics VII, LNCIS 271*. Springer Verlag, 2001.

[2] T. Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, College of Computing - Georgia Institute of Technology, 1998.

[3] A. Das, J. Spletzer, V. Kumar, and C. J. Taylor. Ad hoc networks for localization and control. In *Proceedings of the IEEE Conference on Decision and Control*, 2002.

[4] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[5] B. Gerkey, R. Vaughan, K. Stoy, A. Howard, G. Sukhatme, and M. Mataric. Most valuable player: A robot device server for distributed control. In *Proceedings of the IEEE/RJS International Conference on Intelligent Robots and Systems*, pages 1226–1231, 2001.

[6] D. MacKenzie, R. Arkin, and J. Cameron. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–52, 1997.

[7] D. Martin, A. Cheyer, and D. Moran. The open agent architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.

[8] J. Peterson, G. Hager, and P. Hudak. A language for declarative robotic programming. In *Proceedings of 1999 IEEE International Conference on Robotics and Automation*, pages 1144–1151, 1999.

[9] R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings of the 1998 IEEE/RJS International Conference on Intelligent Robotics and Systems*, pages 1931–1937, 1998.

# Deployment and Connectivity Repair of a Sensor Net with a Flying Robot

P. Corke[1] and S. Hrabar[2] and R. Peterson[3] and D. Rus[4] and S. Saripalli[2] and G. Sukhatme[2]

[1] CSIRO ICT Centre
Australia, peter.corke@csiro.au
[2] Center for Robotics and Embedded Systems
University of Southern California, Los Angeles, California, USA
shrabar@robotics.usc.edu, srik@robotics.usc.edu, gaurav@robotics.usc.edu
[3] Dartmouth Computer Science Department, Hanover, NH 03755 USA,
rapjr@cs.dartmouth.edu
[4] Computer Science and Artificial Intelligence Laboratory
MIT, Cambridge MA 02139, USA, rus@csail.dartmouth.edu

**Abstract.** We consider multi-robot systems that include sensor nodes and aerial or ground robots networked together. Such networks are suitable for tasks such as large-scale environmental monitoring or for command and control in emergency situations. We present a sensor network deployment method using autonomous aerial vehicles and describe in detail the algorithms used for deployment and for measuring network connectivity and provide experimental data collected from field trials. A particular focus is on determining gaps in connectivity of the deployed network and generating a plan for repair, to complete the connectivity. This project is the result of a collaboration between three robotics labs (CSIRO, USC, and Dartmouth.)

## 1 Introduction

We wish to develop distributed networks of sensors and robots that perceive their environment and respond to it. To perform such tasks there needs to exist a synergy between mobility and communication. Sensor networks provide robots with faster and cheaper access to data beyond their perceptual horizon. Conversely robots can assist a sensor network by deploying it, by localizing network elements post deployment [6], by making repairs or extensions as required, and acting as "data mules" to relay information between disconnected sensor clusters.

In this paper we describe our algorithms and experiments for deploying sensor networks using an autonomous helicopter. The static sensor nodes are Mica Motes and the mobile node is the autonomous helicopter. Once on the ground, the sensors establish an ad-hoc network and compute their connectivity map in a localized and distributed way. If the network is disconnected, a localized algorithm determines waypoints for the helicopter to drop additional nodes at.

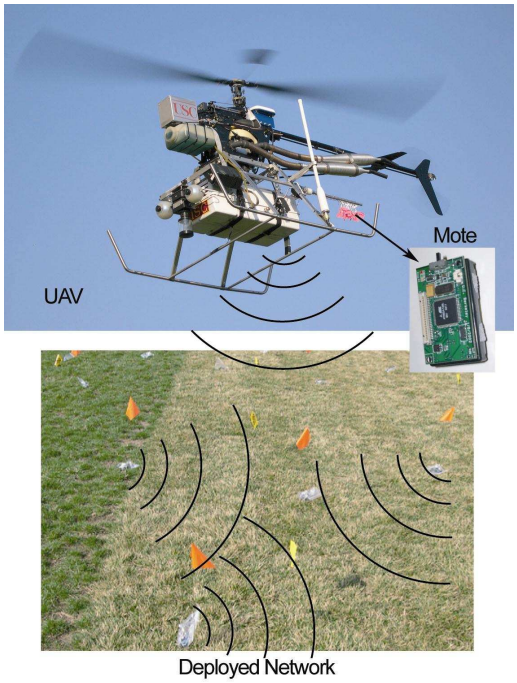



**Fig. 1.** AVATAR Autonomous Helicopter with a sensor interface for deploying sensors

## 2 Related Work

Our work builds on important previous work in sensor networks [8, 11, 14] and unmanned aerial vehicles [3, 16]. It bridges the two communities by integrating autonomous control of flying vehicles with multi-hop message routing in ad-hoc networks. Autonomous aerial vehicles have been an active area of research for several years. Autonomous model helicopters have been used as testbeds to investigate problems ranging from control, navigation, path planning to object tracking and following. Flying robot control is a very challenging problem and our work here builds on successes with hovering and control for two autonomous helicopters [3,17]. Several other teams are working on autonomous control and other varied problems with helicopters. A good overview of the various types of vehicles and the algorithms used for control of these vehicles can be found in [17] . Recent work has included autonomous landing [16, 19], aggressive maneuvering of helicopters [9] and pursuit-evasion games [21].

Research in sensor networks has been very active in the recent past. An excellent general introduction on sensor networks can be found in [8]. An overview of hardware and software requirements for sensor networks can be found in [12] which describes the Berkeley Mica Motes. Algorithms for positioning a mobile sensor network

includes even dispersal of sensors from a source point and redeployment for network rebuilding [2, 13]. Other important contributions include [1, 4, 10, 15, 18].

In [6] we describe a decentralized and localized algorithm called *robot-assisted localization* for localizing a sensor network with a robot helicopter. In [7] we describe an algorithm called *network-assisted navigation* in which a sensor network guides a robot helicopter. In [5] we describe an algorithm and preliminary experiments for deploying a sensor network with a robot helicopter. Here we extend this work to include deployment and connectivity repair and discuss our field experiments using a autonomous helicopter and a 55-node sensor network.

## 3 Approach

Our approach consists of three phases. In the first phase, an initial autonomous network deployment is executed. In the second phase, the entire network measures its connectivity topology. If this topology does not match the desired topology, a third phase is employed in which waypoints for the helicopter are computed at which additional sensors are deployed. The last two phases can be run at any point in time to detect the potential failure of sensor nodes and ensure sustained connectivity.

### 3.1 Deployment Algorithm

Given a desired network topology for the deployed network deployed, and a deployment scale (usually the inter-sensor distance between the nodes in the network), we embed the topology in the 3-dimensional hyper-plane at the given location and extract desired node locations from the resulting embedding. The resulting locations are the $(x, y, z)$ co-ordinates where the sensors need to be deployed.These are given as way-point inputs to the helicopter controller. The helicopter then flies to each of these way-points autonomously, hovers at each of them and then deploys a sensor at the specified location.

### 3.2 Connectivity Measurement Algorithms

Two methods were used to measure network connectivity: a ping-based connectivity measure and a token-passing based measure. For the ping-based measure, a Mote sensor that has been specially modified to add physical user interface controls (a potentiometer and switch) is used to control and configure the sensor side of the ping connectivity tests prior to Algorithm 1 executing.

For the token based connectivity measure each node assumes its network ID as its token. All nodes broadcast and trade tokens as described in Algorithm 2. Tokens are only propagated amongst nodes in connected regions. Thus, disconnected regions will have differing token values. This algorithm is run automatically at 30 second intervals.

Slight differences in connectivity were observed when comparing the ping and token measurements of connectivity and were found to result from the differences

**Algorithm 1** Ping connectivity algorithm for ground deployed motes.

**Wait** for experiment configuration/start message
**Initialization:** Set configuration *mode* = air-to-ground, ground-to-ground, or ground-to-air.
Set *count* = number of ping iterations.
**Send** a multi-hop forwarding of start message to other motes.

**Thread 1**
**for** i=1 to *count* **do**
  **if** *mode* = ground-to-ground OR *mode* = ground-to-air **then**
    broadcast a ping message.
  Sleep a random interval

**Thread 2**
**while** Listen for messages **do**
  **if** message is a ping **then**
    **if** *mode* = air-to-ground OR *mode* = ground-to-ground **then**
      reply to ping.
  **else if** Message is a ping reply. **then**
    tabulate reply.

**Termination:** broadcast counts of replies per mote ID in response to download message.

in message length. Pings are very short messages (1 byte payload) while token messages are longer (10 byte payload). The longer message length increases the chance of collisions and reduces the probability of reception of token messages.

### 3.3 Connectivity Repair Algorithm

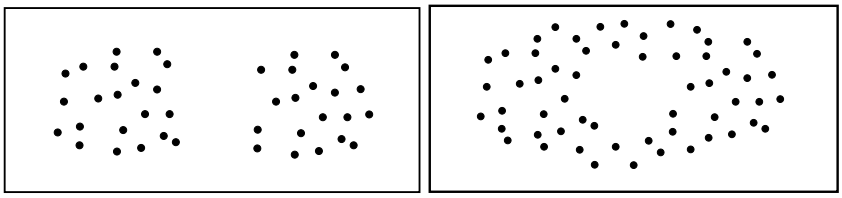


**Fig. 2.** (Left)Two disconnected components in a sensor network field. (Right) A single network which is not fully connected.

The token based connectivity algorithm is a localized and distributed algorithm for computing connected components in the deployed network. Each node ends up with one token that denotes the group to which it belongs. These tokens are collected by the helicopter during a sweep of the field. If more than one token is collected, the network is not connected and new sensor deployments are needed. The locations of the collected tokens can be used to determine the repair regions.

We have developed two algorithms for repairing network connectivity. In the first algorithm, the robot helicopter estimates the location of the gap between two

disconnected components by estimating the locations of the fringe nodes (see Figure 2(Left)). The repair locations are interpolated between the fringes, based on average sensor communication range, which is known.

**Algorithm 2** Distributed algorithm for identifying the connected components in a sensor network. All the nodes in one connected component will have the same $component$ value as a result of this protocol.

**for** each node in the sensor network **do**
  $component$ = id
**for** each node in the sensor network **do**
  broadcast node id.
  **while** listen for $new_{id}$ broadcasts **do**

    **if** received id $>$ $component$ **then**
      $component = new_{id}$
    broadcast $new_{id}$
Helicopter collects all $component$ values
Helicopter determines unique $component$ values as the number of connected components.

In the second algorithm the sensor field computes a potential field to regions of "dark" sensors (see Figure 2(Right)) discovered within it and guides the helicopter there using the potential field algorithm in [7]. This second algorithm handles both complete disconnections and holes in the middle of the sensor field.

For our field experiments we used a hand computed version of the first algorithm described above, averaging the fringe locations to determine a center and averaging the fringe gap distance to determine interpolated repair locations used in the autonomous repair deployment phase.

The general connectivity matching problem remains open. This problem reduces to computing subgraph embeddings which is intractable for the optimal case. We hope to identify a good approximation.

## 4 Experiments and Results

We have implemented the deployment algorithms on a hardware platform that integrates hardware and software from three labs: USC's autonomous helicopter, Dartmouth's sensor network, and CSIRO's interface between a helicopter and a sensor network. Over January 23–25 the three groups met at USC and conducted joint experiments which demonstrate, for a desired network topology, (1) autonomous deployment of a 40 node sensor network with a robot helicopter, (2) autonomous and localized computation of connectivity maps (3) autonomous determination of disconnected network components and autonomous repair of the disconnections.

### 4.1 The Experimental Testbed

The experimental testbed consists of three parts (a) An autonomous helicopter (b) "Mote" sensors and (c) Helicopter-sensor interface. The helicopter [20] is a gas-powered radio-controlled model helicopter fitted with a PC-104 stack augmented with sensors (Figure 1). Autonomous flight is achieved using a *behavior-based* control architecture [16]. Our sensor network platform is the Berkeley Mica Mote [12]. The operating system support for the Motes is provided by TinyOS, an event-based operating system. Our testbed consists of 50 Mote sensors deployed in the form of a regular $11 \times 5$ grid, see Figure 4.1. An extra Mote sensor is fitted to the helicopter to allow communications with the deployed sensor network and is connected to the helicopter's Linux-based computer. For further details the reader is referred to [5]. Several applications were run onboard the helicopter, depending on the experiment. The `ping` application sends a broadcast message with a unique id once per second and logs all replies along with the associated Mote identifier. This data allows us to measure air-ground connectivity. The `gps` application receives GPS coordinates via a network socket from the helicopter navigation software and broadcasts it. Simple algorithms in each Mote are able to use these position messages to refine an estimate of their location [6].

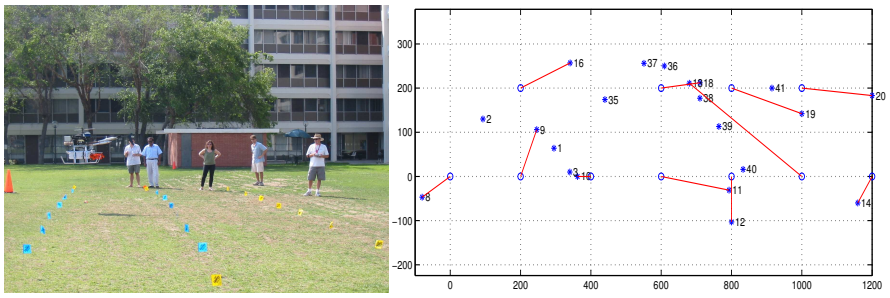



**Fig. 3.** (Left) The sensor network field with flags marking desired sensor locations. (Right) The locations of a sensor network deployed autonomously by the robot helicopter. The desired locations are denoted by $*$ and they are on a grid. The actual locations are denoted by $o$.

### 4.2 Experimental Results

Our field experiments have been performed on a grass field on the USC campus (see Figure 3(left)). We marked a $11 \times 5$ grid on the ground with flags. We used an empirical method to determine the spacing of the grid. We established that on that ground, the Mote transmission range was 2.5 meters. We selected the grid spacing at 2 meters so that we would guarantee communication between any neighbors in the field.

### 4.3 Deployment and Connectivity Results

Figure 3(Right) shows the desired and actual location of the deployed sensors. The deployment error has multiple causes: (a) error in release location compared to desired location (error due to inherent error present in GPS). (b)error in location of markers on ground compared to release location (wind, downwash, and bounce induced error.)

After being deployed the ground sensors establish autonomously an ad-hoc network whose connectivity topology is shown in Figure 4(Top Left). Although there was error in the deployed location, the resulting network is fully connected. We then manually removed 7 nodes down the center of the network to simulate node failure and create a disconnection in the network. The network automatically computed a new token connectivity map as described in Algorithm 2. Figure 4(Top Right) shows the disconnected components as computed by the token algorithm. Finally the robot helicopter autonomously deployed new nodes to repair connectivity resulting in the connectivity map shown in Figure 4(Bottom). Note that some network links were lost in the final graph. Besides some nodes failing due to being out in the hot sun for a day, the introduction of new nodes results in changes in message timing which changes collision rates and hence overall connectivity, even for nodes remote from the area of repair. Mote communication is inherently unreliable as well. The communication range is dependent on relative antenna orientation, shielding (eg. obstacle between two Motes), ground moisture, current receiver autogain levels, etc. The communication links are asymmetric and congestion is a significant concern. We believe that error, uncertainty, and asymmetry are significant factors that should be explicitly included in any model and approach for networked robotics.

### 4.4 Localization Results

During localization the flying robot followed a preprogrammed path, see Figure 6(Left). The computer onboard the helicopter obtained its current coordinates and broadcast this via the mote attached to the helicopter once every 100ms. Each ground mote recorded all the X,Y broadcasts it received and used them to compute a centroid based location for itself. Figure 6(Right) shows the helicopter height. Figure 5 shows the location of each of the the motes broadcasts received. It is clear that the motes do not receive messages uniformly from all directions. We speculate that this is due to the non-spherical antenna patterns for transmitter and receiver motes, as well as non-uniform height of the helicopter itself during flights.

## 5 Conclusion

We have described control algorithms and experimental results from sensor network deployment, localization and subsequent repair of the sensor network with an autonomous helicopter. By sprinkling sensor nodes, we can reach remote or dangerous environments such as rugged mountain slopes, burning forests, etc. We believe that

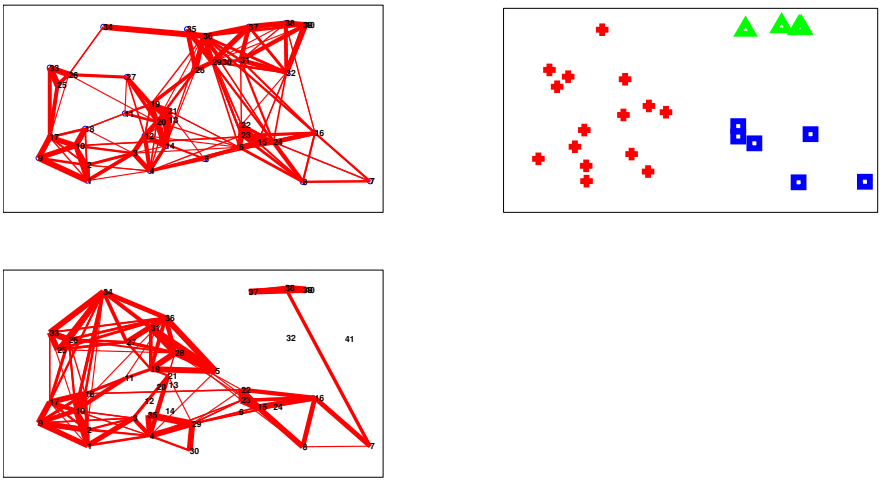


**Fig. 4.** (Top Left) Connectivity of the initial deployment. (Top Right) Token groups showing connected components after several nodes were removed from the field. (Bottom) Connectivity after the deployment of additional sensor nodes to repair connectivity.

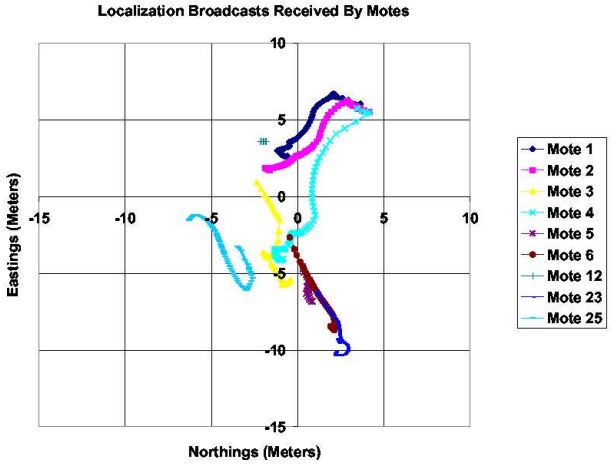



**Fig. 5.** Location broadcasts heard by some of the motes in the network.

this kind of autonomous approach will enable the instrumentation of remote sites with communication, sensing, and computation infrastructure, which in turn will support navigation and monitoring applications. From what we've learned in these experiments we plan to develop systems for automatic network repair. This will

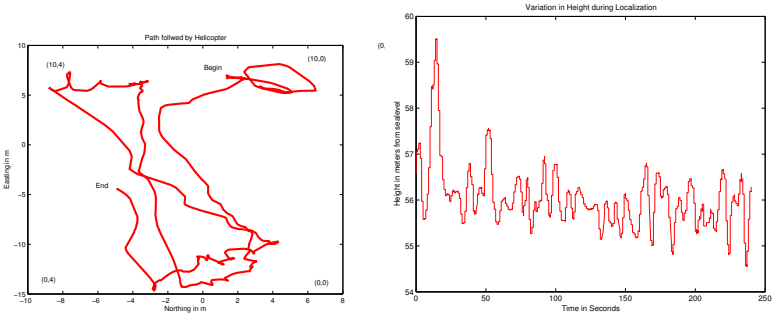



**Fig. 6.** (Left) The path taken by the helicopter while broadcasting location messages. (Right) The height of the helicopter during the process.

require the ground sensors and helicopter to cooperate to identify network disconnections and guide the helicopter to appropriate locations for autonomous sensor deployment.

## Acknowledgment

Support for this work was provided through the Institute for Security Technology Studies, NSF awards EIA-9901589, IIS-9818299, IIS-9912193, EIA-0202789 and 0225446, ONR award N00014-01-1-0675 and DARPA Task Grant F-30602-00-2-0585. This work is also supported in part by NASA under JPL/caltech contract 1231521, by DARPA under grants DABT63-99-1-0015 and 5-39509-A (via UPenn) as part of the Mobile Autonomous Robot Software (MARS) program. Our thanks to our safety pilot Doug Wilson for keeping our computers from (literally) crashing.

## References

1. J. Agre and L. Clare. An integrated architeture for cooperative sensing networks. *Computer*, pages 106 – 108, May 2000.
2. M.A. Batalin and G.S. Sukhatme. Spreading out: A local approach to multi-robot coverage. In *Distributed Autonomous Robotic Systems 5*, pages 373–382, 2002.
3. G. Buskey, J. Roberts, P. Corke, P. Ridley, and G. Wyeth. Sensing and control for a small-size helicopter. In B. Siciliano and P. Dario, editors, *Experimental Robotics*, volume VIII, pages 476–487. Springer-Verlag, 2003.
4. Y. Chen and T. C. Henderson. S-NETS: Smart sensor networks. In *Seventh International Symposium on Experiemental Robotics*, Hawaii, Dec. 2000.
5. P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 1143–8, 2004.
6. P. Corke, R. Peterson, and D. Rus. Networked robots: Flying robot navigation with a sensor network. In *ISRR*, 2003.
7. P. Corke, R. Peterson, and D. Rus. Coordinating aerial and ground robots for navigation and localization. In *Submitted to Distributed Autonomous Robotic Systems*, 2004.

8. D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *ACM MobiCom 99*, Seattle, USA, August 1999.
9. V. Gavrilets, I. Martinos, B. Mettler, and E. Feron. Control logic for automated aerobatic flight of miniature helicopter. In *AIAA Guidance, Navigation and Control Conference*, Monterey, CA, USA, Aug 2002.
10. P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, IT-46(2):388–404, March 2000.
11. J. Hill, P. Bounadonna, and D. Culler. Active message communication for tiny network sensors. In *INFOCOM*, 2001.
12. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. In *ASPLOS*, 2000.
13. A. Howard, M.J. Mataric, and G.S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable so lution to the area coverage problem. In *Distributed Autonomous Robotic Systems 5*, pages 299–308, 2002.
14. Q. Li, M. DeRosa, and D. Rus. Distributed algorithms for guiding navigation across sensor networks. In *MOBICOM*, 2003.
15. G. J. Pottie. Wireless sensor networks. In *IEEE Information Theory Workshop*, pages 139–140, 1998.
16. S. Saripalli, J. F. Montgomery, and G. S. Sukhatme. Visually-guided landing of an unmanned aerial vehicle. *IEEE Transactions on Robotics and Automation*, 19(3):371–381, June 2003.
17. S. Saripalli, J. M. Roberts, P. I. Corke, G. Buskey, and G. S. Sukhatme. A tale of two helicopters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, USA, Oct 2003. (To appear).
18. A. Scaglione and S. Servetto. On the interdependence of routing and data compression in multi-hop sensor networks. In *ACM Mobicom*, Atlanta, GA, 2002.
19. O. Shakernia, Y. Ma, T. J. Koo, and S. S. Sastry. Landing an unmanned air vehicle:vision based motion estimation and non-linear control. In *Asian Journal of Control*, volume 1, pages 128–145, September 1999.
20. University of Southern California Autonomous Flying Vehicle Homepage. http://www-robotics.usc.edu/˜avatar.
21. R. Vidal, O. Shakernia, H. J. Kim, D. Shim, and S. Sastry. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation*, Oct 2002.

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

# Development and Deployment of a Line of Sight Virtual Sensor for Heterogeneous Teams

Robert Grabowski, Pradeep Khosla and Howie Choset

Carnegie Mellon University
Electrical and Computer Engineering, and Mechanical Engineering Department
Pittsburgh, Pennsylvania 15213
{grabowski, pkk, choset}@cs.cmu.edu

***Abstract* — For a team of cooperating robots, geometry plays a vital role in operation. Knowledge of line of sight to local obstacles and adjacent teammates is critical in both the movement and planning stages to avoid collisions, maintain formation and localize the team. However, determining if other robots are within the line of sight of one another is difficult with existing sensor platforms – especially as the scale of the robot is reduced. We describe a method of exploiting collective team information to generate a virtual sensor that provides line of sight determination, greater range and resolution and the ability to generalize local sensing. We develop this sensor and apply it to the control of a tightly coupled, resource-limited robot team called Millibots.**

**Keywords-component; mobile robot teams; sensing; heterogeneous control**

## I. INTRODUCTION

Robots are versatile machines that can be programmed to react collectively to sensor information in a variety of tasks that range from surveillance and reconnaissance to rescue support. Despite this versatility, a single robot cannot always realize all applications. On the other hand, a team of robots can coordinate action and sensing to extend a collection of individual entities to a single, cohesive group. To facilitate this coordination, a robot team must be able to manage its formation to exchange information and leverage the proximity of the others.

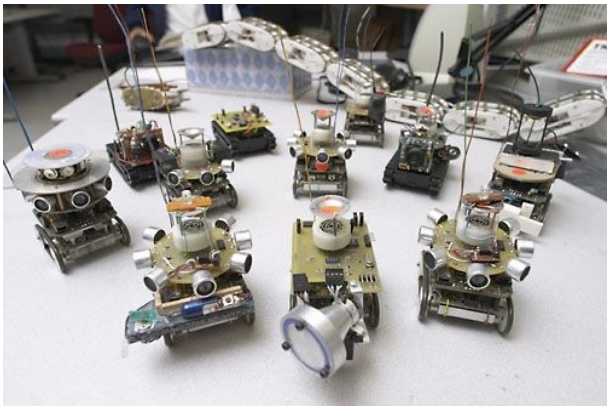


Figure 1. The Millibot Team – a heterogeneous collection of small-scale robots designed on the 5cm scale.

Formation control is essential in many aspects of team coordination from communications [1][2] to sensor coverage [7][11] to localization [4][9][12]. A critical component of formation control is line of sight. Line-of-sight is defined as an open, obstacle-free path between two points in space and must be wide enough to allow the passage of information signals such as light, video or ultrasonics. Unfortunately, local sensing is not always sufficient to directly determine the property of line of sight. Local sensors are often limited in their range and resolution and are incapable of discriminating between robot and obstacle. Even when a robot has access to a local map, it still may not have acquired sufficient information to make the determination on its own. This is especially true as the scale of the robot is decreased, the number of available sensors is restricted and the range of local sensing is reduced.

Coordinating multiple robots is a management issue as well. Conventional formation control is based on the idea that each robot is equipped with roughly the same sensing capabilities. Heterogeneous team control must take into account for the differences in the sensing and processing capabilities of each robot. In some cases, this sensing may be rudimentary and not able to provide the necessary local information needed to navigate on its own [7][11]. The problem becomes even more compounded when the composition and number of the team is dynamic.

Our work is primarily motivated by the control and coordination of a team of heterogeneous, resource-limited robots, called Millibots [7]. These are small-scale robots on the order of 5cm a side that are designed to operate in unknown or partially known environments. Their small size gives them access to tight, inaccessible areas while making them easier to conceal, deploy and manage. However, their small scale and dynamic heterogeneous composition makes conventional control strategies difficult to apply.

We address coordination of multiple, heterogeneous robots by developing the concept of a 'virtual' sensor. Robot teams have the advantage that they can collectively share information. They are able to fuse range information from a variety of different platforms to build a global occupancy map that represent a single collective view of the environment. A virtual sensor is simply an abstraction of the team's occupancy map. We call this a virtual sensor because it has all the properties of a real sensor with respect to that robot's navigation and planning but is derived from information already processed and not from the physical interaction of a sensor and its

surroundings. However, when employed by the individual, the information derived from a virtual sensor can be treated in the same fashion as a real sensor.

In Section III, we develop the virtual sensor and show how it can provide essential line of sight information to obstacles, open space and other robots regardless of the platform being employed. In section IV, we show how this generalization aids in local, sensor-based planning by providing information with greater range and resolution than existing local sensors. We then show how it can be extended to the planning stage with respect to maintaining line of sight to multiple members during and after movement. Finally, in Section V, we show how the virtual sensor allows the generalization of existing sensors in such a way as to allow homogenous control laws to be applied to a heterogeneous team.

## II. Related Work

The concept of recasting local sensing is not new. Borenstein proposed recasting a robot's local map in terms of a polar sensor in the development of his Vector Field Histogram [5]. From the vector field, he is able to apply potential field methods for avoiding obstacles and navigating through open space. This method was designed primarily to support local, sensor-based navigation of a single robot and was not intended to support multiple robots.

Banos utilizes a high-resolution laser range sensor to express the robot's centric view as a collection of polylines by connecting the ends of each range measurement together into a contour [3]. As the robot generates new views, he combines polylines to form a composite representation of the environment. While a powerful geometrical means for combining robot views, this method lacks a means for separating other robots from the environment as well as a means for assessing new plans based on the line of sight to other robots. Moreover, it relies on high resolution sensing which becomes problematic for robot teams with more limited sensing modalities.

Nelson describes the generation of an enhanced range sensor by utilizing vision to evaluate the height information in a video image [10]. The relationship between range and image height is made possible by regulating the dimensions of the walls in the environment. Moreover, line of sight to other robots is obtained by assigning unique colors to each robot. While this method solves both the range and line of sight issue, it requires the construction of a controlled environment and is not suitable for applications in unknown environments.

## III. The Virtual Sensor

To develop the virtual sensor, we first recast the team occupancy map as a polar occupancy map with respect to robot. The polar map is constructed as a grid with the columns representing range and the rows representing bearing (Figure 2b). We make the transformation by mapping the value of the occupancy map cells to the corresponding polar occupancy cells for each cell in the polar map.

From the polar map, we generate a polar contour that records the closest features in the map with respect to the center of the robot. Features captured by the polar contour include the closest obstacle and the closest free space boundary. The polar contour is stored as a single linear array with the number of cells equal to the column width of polar map (Figure 2c). The indices of array correspond to the angle and their values are the range to the closest feature This reduced representation allows rapid calculation of line of sight as well as a compact representation for faster communications.

To determine the range to the closest features, we scan each of the columns of the polar map from bottom (closest to the robot) to the top (maximum range) until we detect a cell that transitions from open (low occupancy) to either closed (high occupancy) or unexplored. If a transition from open to high occupancy is detected, the scan is terminated and the range is

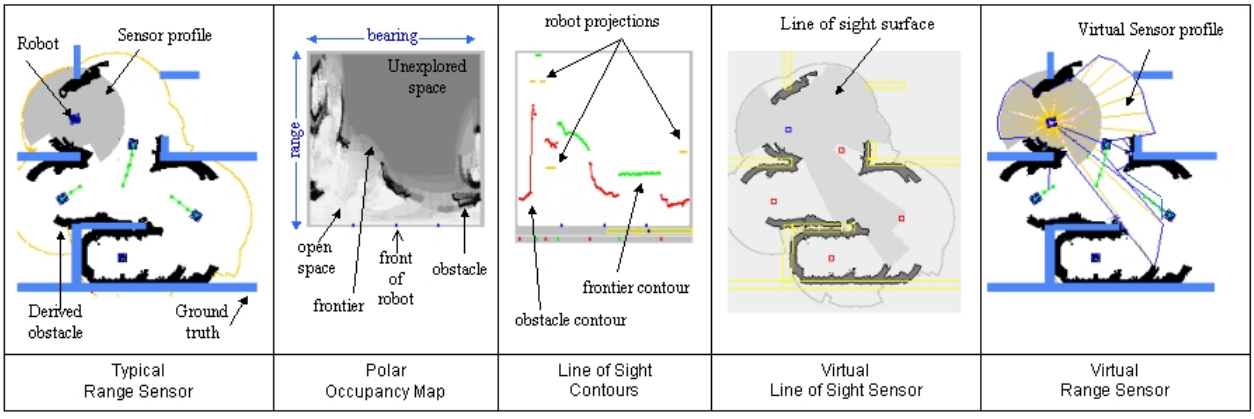



Figure 2. Generating the Virtual Sensor. - **a)** A robot's local sensors may not have the resolution to determine line of sight to other members (dark gray profile) **b)** First step is to map the team's occupancy map into the individual polar map with the target robot at the center - axes are range and bearing. **c)** We process the polar map to generate a polar contour - For a given bearing, we mark the closest transition from open space to form obstacle contours and frontier boundaries. Additionally, we project the profile of adjacent teammates onto the map **d)** We test each point in space against the contour map. All points within this region are within line of sight of the target robot. **e)** We utilize the values of a contour map to generate a virtual range sensor with greater range and resolution of the robot's individual sensor. This generalized profile is the same for any robot regardless of underlying sensor platform.

recorded in the array. These cells represent the closest obstacle points within the line of sight of the robot.

If a transition from open to unexplored is detected, the range to this feature is recorded. This transition represents the closest frontier point for that given angle. A frontier represents the boundary between known and unknown space and is used to guide exploration. To allow the storage of range information to two classes of features (frontier and obstacle points) in a single array, we store frontier points as negative values.

The polar contour stores the distance to the closest features within a robot's line of sight. However, its power comes from the ability to test line of sight to arbitrary points in space. We can readily establish whether a point in space is within line of sight of the robot by determining the angle and range to the point with respect to the robot. If the point is above the contour (range greater for a given bearing) it is beyond that robot's line of sight. If it is below the contour it is within line of sight.

Ironically, while robot range sensors, such as sonar, are good at detecting the distance to local obstacles, there are some cases where they are unable to detect the presence of other robots - even when they are within range. Such is the case of the Millibots where the sonar sensor platforms are mounted on top of the robot. These platforms present a small cross section (effective reflection area of the robot) to ultrasonic bursts and consequently do not always reflect sufficient energy to be detected. However, we can leverage team knowledge to account for undetected robots by artificially projecting their profile onto the polar contour. The positions of the robots are obtained from the team's localization solutions and mapped into the polar map.

Instead of simply treating these robots as a point source, we account for their size as well. We project these teammates onto an individual robot's polar map by generating a line whose width is a function of the size of the actual robot and the distance from the center of the target robot. Figure 2c shows the projection of team members onto the target robot's contour map. Where the robot's projection is closer than any other feature, it replaces that feature as the closest point. Now the team can move reliably even when it cannot directly detect its neighbors. Moreover, we have a method for determining the reliability of line of sight information. That is, we can assess whether a participating robot is partially obscured. This ability to account for the projected width of a robot (as opposed to a point source) is one of the advantages of utilizing a polar contour over determining line of sight by using simple ray tracing techniques.

## IV. Virtual Sensor Planning

Line of sight is not just a detection issue; it must be considered during planning. Not only does a robot need to know which robots are within line of sight and which are not, it needs to make the same assessment for future cases in the planning stage. We can exploit the virtual sensor to aid in local robot planning as well as coordinated team planning. For example, localization effectiveness is often a property of the number of robots that participate. Therefore, planning must also account for the number of robots that will be within line of sight after the robot moves. If the new position cannot be adequately resolved due to lack of participation after such a move, the team may select an alternative move or opt to reposition others first.

### *A. Local Planning*

Researchers are already exploiting the increased resolution of maps generated by fusing multiple robot sensors to aid in exploration and localization. However, this has traditionally been a one-way process. Few are exploiting this map information to augment an individual robot's perspective. Consequently, each robot is left to develop plans based primarily on local sensors. We approach information exchange from the other perspective. That is, we use the higher fidelity of the team map to support sensing and local planning.

In addition to line of sight determination, we can recast the virtual sensor in the form of an extended range sensor to aid in local robot planning. To accomplish this, we utilize the range values of a robot's polar contour as an element of a virtual sensor array (an imaginary range sensor) (Figure 2e). The angular and range resolution of the virtual sensor is a product of the number of rows and columns of the polar map and not the underlying sensor. Consequently, the range and resolution of the virtual sensor is generally greater than the underlying robot sensor.

One immediate advantage to this formulation is the ability to perform local sensor-based planning. With greater resolution provided by the virtual sensor, a robot is able to generate finer resolution plans than possible with its original sensors. Many techniques exist that allow a robot to reliably navigate through known spaces utilizing only local sensors [3][5][6]. However, the success of these methods is partially a function of the resolving ability of the sensors. For example, many robots have the luxury of supporting a high-resolution laser rangefinder or an array of 16 or more Polaroid sonar sensors. However, the typical Millibot sensor array utilizes only 8 sonar range sensors each with a range of 30cm. Consequently, it has a harder time applying local control laws to maintain a path or follow the contour of local obstacles. On the other hand, a virtual sensor has a derived range and resolution based on the resolution of the polar map and not the underlying sensor. Consequently, it can augment existing sensing to produce better motion plans for movement through the space.

Robots also utilize features, such as obstacle profiles and frontiers, extracted from the virtual sensor to navigate. One popular method in robot exploration is frontier expansion where the robot is directed towards existing boundaries between open and unexplored space. However, it has been shown that specular reflection and general sensor failure can complicate the proper generation of frontiers [8]. Instead of directing the robot to viable search areas, specular reflection produces erroneous frontiers resulting in plans that direct the robot through obstacles. If the obstacle cannot be traversed, the attempt fails wasting valuable time and resources. On the other hand, a virtual sensor identifies obstacle boundaries and frontiers with respect to the position of an individual robot. As such, it ignores potential erroneous information generated beyond the local line of sight of that robot. Consequently, it is not sensitive to the failures induced by specular reflection.

Obstacle profiles and frontier boundaries are extracted from the virtual sensor by clustering similar points along the polar contour into contiguous objects. Adjacent cells of positive values from the polar contour represent the profiles of obstacles while adjacent cells of negative values represent the profiles of frontiers. Consequently, exploration can be accomplished by directing the robot towards the center of clustered frontier points.

### *B. Line of Sight Participation Planning*

Local control laws are useful for many aspects of robot navigation and planning. However, sometimes a robot has to coordinate its movements with respect to others in order to operate effectively. For example, Millibot requires that at least two other robots be within line of sight after any given move to generate a reliable position estimate. Before a Millibot moves to a new position, it must first assess its chances at localizing.

To support this planning, we first examine the line of sight region generated by a single robot. These are all the points directly viewable with respect to that robot (Figure 2d). Now, if we view the same line of sight region from the perspective of another robot, we have a way of predicting future line of sight constraints for the moving robot. Viewed in such a way, one robot's line of sight region represents all the places a second robot can move and still maintain a connection with the first. The same process can be applied to each of the robots in the team.

Consider the scenario given in Figure 3. Four robots have mapped a given space and have localized into the positions shown (Figure 3a). We now wish to move the robot under test to a new area in order to explore or provide mission specific sensing. The three remaining robots, denoted Ra, Rb, Rc, remain stationary to support localizing the robot after the move. For localization to be successful, the moved robot has to obtain valid range information from at least two of the other robots. The question is where can that robot move and still be in line of sight with respect to the remaining robots.

Again we answer this question by utilizing the virtual sensors of each of the stationary robots. Figure 3b shows the virtual sensor generated by each of these robots and their corresponding line of sight regions. Each reading provides local information about whether the new position will be within its line of sight (Figure 3c). However, we get a composite assessment by projecting each of these regions simultaneously onto a common map (Figure 3d). Points where the regions overlap once represent common areas that are within the line of sight of at least two robots. Regions that overlap twice represent places where all three robots will be within line of sight of the new position. Assessing regions in terms of multiple overlaps provides a basis for coordinating multiple robots. The same procedure can be applied for any operation that requires a robot to maintain line of sight to one or more robots including localization, communications or surveillance. In practice, every point in space does not have to be evaluated as in the way illustrated in Figure 3. Instead candidate movement points can be generated and evaluated by some other criteria.

### *C. Combining with a Localization Metric*

Line of sight and the number of participants is not the only factor in achieving good localization. The geometry of the formation is also a critical factor. Some formations naturally

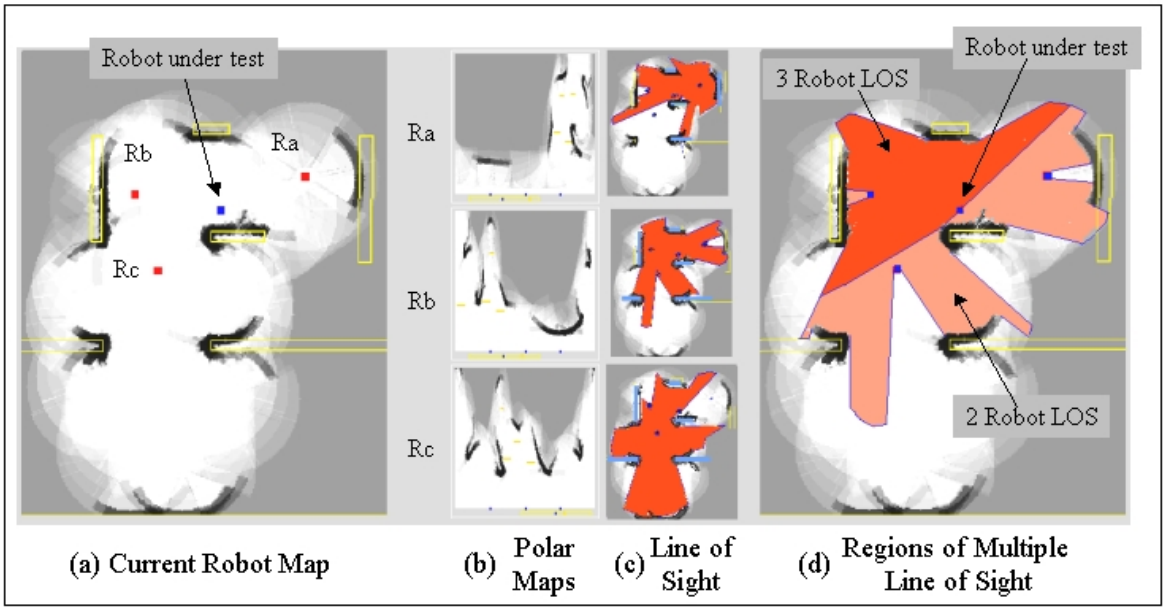



Figure 3. Line of Sight Participation - a) Given three participating robots, we want to find the places in the map where at least two are within line of sight. b) Each participating robot generates a polar plot by generating local polar map. c) Line of sight projected for each participating robot. d) Combine line of sight projections - the darker the overlay, the greater the number of participants. The robot under test can move anywhere within the shown projections and still be in line of sight of at least two robots.

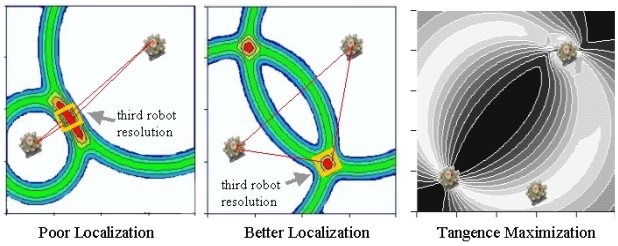



Figure 4. Localization Metric - a) Localization is poor when the team geometry approaches collinear. b) Localization is best when the angle between the two readings form a right angle. c) A metric plot showing utility of maximizing the angle between the tangents of intersecting measurements Lighter areas represent regions in space that generate better localization readings.

lend themselves to better position estimation than others. For example, in a three-robot system, resolution of a third robot is best if we maximize the angles between range measurement pairs of the participating robots. In localization, range measurement pairs can be visualized as an annulus with a center at the originating robot and a radius equal to the range measurement between the two robots. In a three-robot system, two range pairs are obtained to localize that robot. For the best localization, we maximize the angle between the tangents of the two intersecting range pairs so that their combined distributions lie in a confined region of space. If the placement of the third robot is poorly selected (for example if the three robots are collinear) the position estimate of the robot is also poor and spread over a large region of space (Figure 4a). However, if we maximize the angle between the tangents of the range measurements (Figure 4b), the same combined distribution is confined to a smaller region in space resulting in a better position estimate.

Given this knowledge, we can utilize geometry in the planning stage to guide robot movement as to best satisfy its line of sight constraints while taking into account the best location for maximizing its position estimate. To accomplish this, we develop a metric that maximizes the tangents of the range pairs. Figure 4c shows the application of this metric to the space around the set of robots. The lighter areas represent regions in space that generate well-localized position estimates while the darker areas generate poor estimates. Coupled with the line-of-sight assessment, candidates are selected that maximize localization resolution while ensuring an adequate number of robots remains within line-of-sight of each other.

A similar approach is applied to deal with the multitude of team constraints. For example, in the control of the Millibots, we cast a series of random points about a robot and pose each point as a candidate position for movement. From the list of candidates, we test each against conflicting constraints that include line of sight, obstacle clearance, travel distance, exploration gain etc. The point with the highest overall utility is then selected and the robot is directed to that point.

## V. Heterogeneous Sensing

As if reduced range and resolution were not enough of a handicap, some robot teams must contend with the composition of heterogeneous sensing. Such is the case for the Millibot team where the scale of the robots does not support a single robot type. Robots must distribute and coordinate sensing to achieve the necessary degree of perception. For example, some robots are equipped with sonar sensors that provide range information to obstacles while others are equipped with mission dependent sensing like heat detectors or video cameras.

Conventional team control is accomplished by treating each robot as if they were interchangeable. Control is a matter of managing the physical locations of the robots but not the individual resources of those robots. Heterogeneous composition complicates this methodology and introduces complexity in the control process. Uniformity of sensing on an individual robot is an issue as well. Local navigation strategies often assume a uniform sensor distribution about the robot. However, in some cases, a robot is equipped with a variety of sensors each with different sensing characteristics.

One example is the Dirrsbot (Figure 5a). This Millibot contains three forward-looking sonars similar to others in the group each with a range of 30cm. However, it gets its name from two side-looking, digital infrared range sensors (dirrs). These sensors have a range of 80cm but a narrow 5-degree field of view. Not only is the sensor profile for this robot non-uniform, it does not fully cover the area about the robot. Integrating this robot requires specialized routines for almost every aspect of operation. On the other hand a virtual sensor is posed as a uniform array with respect to the robot (Figure 5c). Moreover, since the virtual sensor for each robot is derived from the fusion of the same team map, each robot can utilize the same navigation strategies.

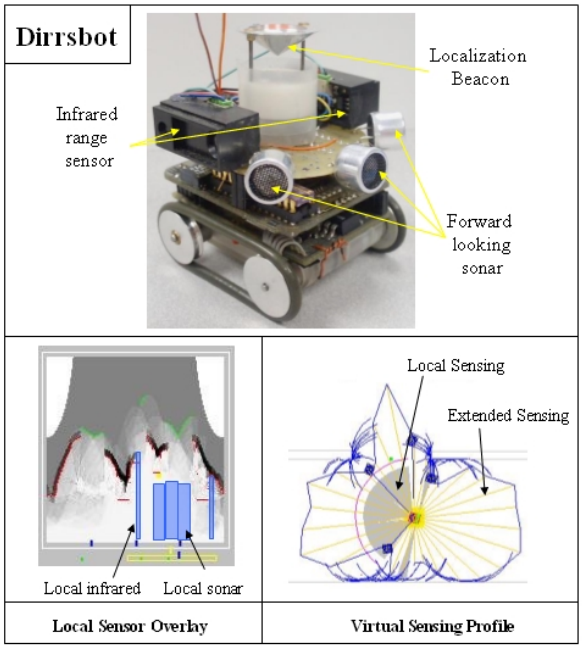



Figure 5. Supporting Heterogeneous Sensing – a) A heterogeneous Millibot equipped with forward-looking sonar and side-looking infrared. b) Correlation between local and virtual sensor. c) Comparison of non-uniform local sensor and uniform virtual range sensor.

Sensor failure also complicates reliable local navigation. Sensor failure and specular reflection often result in readings that indicate a clear path even when the path is obstructed. However, if we correlate the current sensor readings with the derived virtual sensor readings (Figure 5b), we have a means for putting local readings in context and rejecting suspect readings. This additional confidence in sensing equates directly to competence in moving through a desired space without incident.

In some cases, a robot simply does not have a means for sensing its own surroundings. One example is a Millibot equipped with only a video camera and localization beacon. Even when the robot needs to navigate in known space, it has no means for generating local plans without involving the operator. However, by utilizing the virtual sensor, the same robot can operate as if equipped with range sensors.

Virtual sensors provide an added benefit in that they allow the possibility of the generalization of local sensor platforms into a ubiquitous representation. Robots with long-range and short-range sensors generate similar virtual sensors as do robots with non-uniform sensor distribution or no sensors at all. This ability to generalize robot sensing is instrumental in applying generalized robotic algorithms to heterogeneous robot platforms.

## VI. CONCLUSION

In this paper we have shown how to generate a virtual sensor for each robot that is an abstraction of an individual robots sensor and the team's occupancy map. The formulation of this sensor allows a local determination of line of sight local obstacles and frontiers as well as other robots in the team. This assessment is critical for establishing formation to support many aspects of team coordination including localization, communications and coverage. Equally important is the ability of determining when line of sight has been violated to reject potentially erroneous signals still present due to multipath reflections.

Line of sight assessment is facilitated by expressing the range and bearing to local features in the form of a polar contour. Line of sight to arbitrary points in space is determined by testing whether the desired point lies above or below this contour.

The utility of the virtual sensor moves from sensing to planning by viewing the contour of one robot with respect to the all the others. In this fashion, one robot's line of sight region represents all the areas a second robot can move and still remain within the line of sight of the first. The same process can be simultaneously applied to the other robots in the team to satisfy formations that require line of sight to multiple robots.

Finally we show the added utility of virtual sensing when applied to teams with heterogeneous sensor composition. The virtual sensor provides a means of recasting a robot's local sensing in terms of the collective sensing stored in the team's occupancy map. Consequently, the derived virtual sensor is the same for any robot regardless of the underlying sensor platform. This ubiquitous representation of sensing allows the application of conventional homogenous control techniques on heterogeneous teams.

## REFERENCES


[1] Anderson, S., Simmons, R., Goldberg, D., "Maintaining Line of Sight Communications Networks between Planetary Rovers," Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems Las Vegas, Nevada · October 2003.

[2] Arkin, R., Balch, T., "Line-of-Sight Constrained Exploration for Reactive Multiagent Robotic Teams", 7th International Workshop on Advanced Motion Control, AMC'02 , Maribor, Slovenia, July 2002.

[3] Banos, H., Mao, E., Latombe, J.C., Murali, T.M., and Efrat A., "Planning Robot Motion Strategies for Efficient Model Construction." Robotics Research -- The 9th Int. Symposium, J.M. Hollerbach and D.E. Koditschek (eds.), Springer, pp. 345-352, 2000.

[4] Bisson, J., Michaud, F., Létourneau, D., "Relative Positioning of Mobile Robots Using Ultrasonics," Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems Las Vegas, Nevada · October 2003.

[5] Borenstein, J., and Koren, Y. "The Vector Field Histogram - fast Obstacle Avoidance for Mobile Robots," IEEE Transactions on Robotics and Automation 7(3): 278 -288. 1991

[6] Choset, H., Nagatani, K., "Topological Simultaneous Localization and Mapping (SLAM): Toward Exact Localization Without Explicit Localization," In IEEE Transactions on Robotics and Automation, Vol. 17, No. 2, April, 2001, pp. 125 - 137.

[7] Grabowski, R., Navarro-Serment, L. E., Paredis, C.J.J., and Khosla, P. "Heterogeneous Teams of Modular Robots for Mapping and Exploration," Autonomous Robots - Special Issue on Heterogeneous Multirobot Systems.

[8] Grabowski, R., Khosla, P., Choset, H., "Autonomous Exploration via Regions of Interest," Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems Las Vegas, Nevada · October 2003.

[9] Navarro, L., Paredis, C., Khosla, P., "A Beacon System for the Localization of Distributed Robotic Teams," in Proceedings of the International Conference on Field and Service Robotics, Pittsburgh, PA, August 29-31, 1999.

[10] Nelson, A., Grant, E., Barlow, G., and Henderson, T., "A Colony of Robots Using Vision Sensing and Evolved Nueral Controllers," Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems Las Vegas, Nevada · October 2003.

[11] Parker, L., Kannan, B., Fu, X., and Tang, Y., "Heterogeneous Mobile Sensor Net Deployment Using Robot Herding and Line-of-Sight Formations," ," Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems Las Vegas, Nevada · October 2003.

[12] Rekleitis, Y., Dudek, G., Milios, E., "Multi-Robot Collaboration for Robust Exploration," Annals of Mathematics and Artificial Intelligence, 2001, volume 31, No. 1-4, pp. 7-40.

Autonomous Robots 15 (2), pp. 155-168, 2003

# Distributed coordination in heterogeneous multi-robot systems

Luca Iocchi and Daniele Nardi
*Dipartimento di Informatica e Sistemistica*
*Università di Roma "La Sapienza"*
*Via Salaria 113, 00198, Roma, Italy*
*Email: {iocchi,nardi}@dis.uniroma1.it*

Maurizio Piaggio and Antonio Sgorbissa
*Dept. of Communication, Computer and System Sciences*
*Università di Genova*
*Via Opera Pia 13, 16125 Genova, Italy*
*Email: {piaggio,sgorbissa}@dist.unige.it*

**Abstract.** Coordination in multi-robot systems is a very active research field in Artificial Intelligence and Robotics, since through coordination one can achieve a more effective execution of the robots' tasks. In this paper we present an approach to distributed coordination of a multi-robot system that is based on dynamic role assignment. The approach relies on the broadcast communication of *utility functions* that define the capability for every robot to perform a task and on the execution of a coordination protocol for dynamic role assignment. The presented method is robust to communication failures and suitable for application in dynamic environments. In addition to experimental results showing the effectiveness of our approach, the method has been successfully implemented within the team of heterogeneous robots Azzurra Robot Team in a very dynamic hostile environment provided by the RoboCup robotic soccer competitions.

**Keywords:** Multi-Robot Systems, Distributed Coordination, Dynamic task Assignment, Communication

## 1. Introduction

Coordination in Multi-Robot Systems (MRS) is nowadays one of the most interesting areas of research in Artificial Intelligence and Robotics [14, 34, 27]. In fact, multi-robot systems are being developed from both a biological and an engineering perspective. In the first case the goal is to simulate the properties of biological systems, while in the second case the goal is to improve the effectiveness of a robotic system both from the viewpoint of the performance in accomplishing certain tasks [14] and in the robustness and reliability of the system [34].

A significant boost to the work on MRS has recently been given also by the Robot competitions and RoboCup, in particular. In fact, the design of MRS is regarded as one of the major scientific challenges to

be developed in the RoboCup environment [27]. In particular, according to the organization of RoboCup, the real robot leagues as well as the simulated league provide different settings, where the design of MRS is realized according to different hypotheses. The most distinguishing feature, as compared with previous work on MRS, is that the RoboCup environment is highly dynamic and hostile due to the presence of an opponent team.

In this paper we present a distributed approach to coordination in MRS that has been originally developed within the Azzurra Robot Team (ART) of robotic soccer players participating in the RoboCup competitions in the mid-size category [13]. Nonetheless, the methods and techniques presented here are rather general and suitable for application in other environments where similar underlying assumptions are satisfied.

ART [31] is composed of different robotic players developed in various Italian universities. Each university in the team is responsible for the development of one or more robots which differ in the mechanics (robot base and kickers), in the hardware (processor type and speed) and in the control software (different techniques and approaches for perception, reasoning and motion control) from the robots of the other research groups. Because of this kind of organization, coordination among the ART robots requires not only a distributed coordination protocol, but also a very flexible one, that allows to accommodate the various configurations that can arise by forming teams with players equipped with different basic features. Due to the intrinsic heterogeneity of the team, the dynamic environment in which the robots act, and the complexity of the task (playing soccer), our main focus in the development of such a MRS has been the implementation of a coordination framework suitable for decomposing the complexity of the task to be carried out into different sub-tasks and for dynamically assigning these tasks to the robots in the MRS.

Given the very wide scope of the proposals on MRS and the relatively youth of the research area, a common framework for the work on MRS is difficult to identify. A MRS cannot be simply regarded as a generalization of the single robot case and the proposed approaches need to be more precisely characterized in terms of assumptions about the environment and in terms of the internal system organization. Classifications of the work on MRS has been presented by Dudek et al. (1996) [14], by Cao et al. (1997) [11], by Iocchi et al. (2001) [25], and by Balch et al. (2002) [7]. In [14] the classification of MRS is focused on the communication and computation aspects, while in [11] several application domains for MRS are described and many systems are classified according to the task they are realized for. In [25] MRS are analyzed by

addressing their deliberation and reactivity capabilities. Other specific goals of the research in MRS include also the issue of explicit versus implicit communication [5] or exploring specific coordination strategies for specific problems [21].

In order to properly characterize the present proposal, it is worth highlighting the hypotheses underlying the MRS described by the present work that can be summarized as follows:

- *Communication-based coordination*: the usage of communication among the robots to improve team performances, allowing the robots to acquire more information and to self-organize in a more reliable way.
- *Autonomy in coordination*: the robots are capable to perform their task, possibly in a degraded way, even in case of partial or total lack of communication.
- *Distributed coordination*: the communication capabilities, combined with the autonomy requirement, require that each robot, while interacting with the others, must rely on local control (hence the system is distributed).
- *Heterogeneity*: the robots are heterogeneous both from hardware and software viewpoints, they can usually perform the same tasks but with different performance.
- *Highly dynamic, hostile environment*: the robots must be able to perform the assigned task in the presence of external and dynamic changes in the environment.

The building blocks of our proposal are a communication layer and a coordination protocol. The former provides for suitable inter-robot information exchange, as well as, proper interoperation within each robot computational model. The latter has been designed to deal both with roles (defender, attacker etc.) and with strategies (defensive, offensive). While the strategic level may be demanded to an external selection, roles are dynamically assigned [44, 42] to the robots during the game.

In the paper we also address the problem of evaluating the performance of MRS [3, 4], and are able to show through the analysis of the game logs that, in the operation environment, both the space coverage and the role exchange are effectively accomplished.

The paper is organized as follows. In the next section, we describe the architecture of the MRS, then the communication infrastructure and the coordination protocol. We then present the experimental results

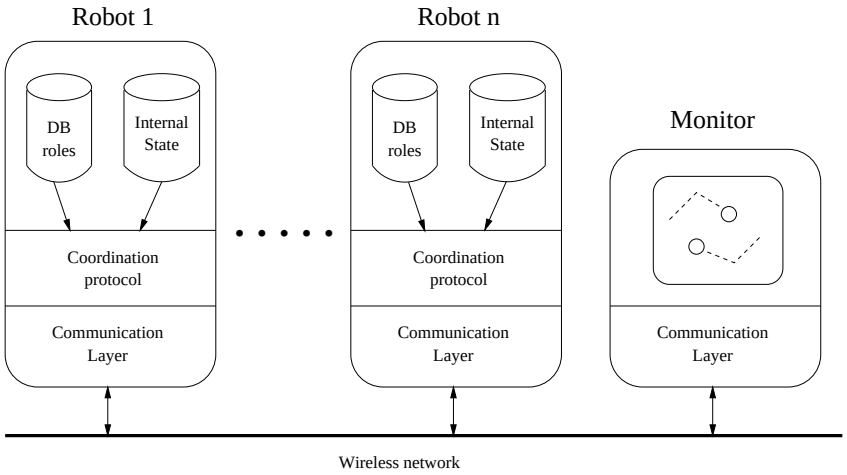



*Figure 1.* System architecture for robot coordination

on coordination of the ART robotic players, address related work and possible application domains of the approach presented, finally we draw some conclusions, specifically discussing both the main features of the proposed approach and some issues that deserve further investigation.

## 2. System Architecture

Our approach to multi robot coordination is based on a coordination protocol run by all the robots of the team. This is obtained by implementing on every robot of the team a common subsystem that is responsible for communication on a (usually wireless) network and to execute the coordination protocol.

The main features of the proposed approach are that the team may easily include heterogeneous robots and that coordination is distributed, since each robot can take decisions on its own behavior based on its knowledge of the situation and no robot can take direct decisions about the behavior of other robots.

In Fig. 1 the system architecture for multi robot coordination is shown. The coordination module for every robot is organized into two layers: the communication layer that is responsible for exchanging data among the robots in the network, and the coordination protocol that performs a negotiation with the other robots and provides other modules of the robot with information about the actions that the robot must perform according to the team goals.

The robots may also be connected to a monitor that is typically used to visually inspect the behavior of the team and also to send general commands (like start or stop) to all the robots.

An important building block, within such an architecture, is ETHNOS (Expert Tribe in a Hybrid Network Operating System) [37] a real-time software architecture and a programming environment dedicated to the development of distributed intelligent robotic applications. In ETHNOS, in order to allocate and guarantee system time for communication activity, the communication infrastructure is fully integrated in the real time software architecture for robot control.

In ETHNOS all the concurrent activities required for the control of a mobile robot are demanded to entities called experts. As an architecture for the development of real-time robotics applications, ETHNOS can schedule three different kind of experts:

1. periodic experts, i.e. real-time tasks which execute at a fixed rate

2. sporadic experts, i.e. real-time tasks which execute in subordination to specific conditions,

3. background experts, i.e. non real-time tasks which execute when the scheduler has no pending requests coming from periodic and sporadic expert to satisfy (for a more exhaustive description of ETHNOS scheduling policy [36]).

In the development of the ART MRS ETHNOS has played a twofold role: on some of the robots is used as the underlying operating system, in other robots it simply implements the communication layer. In the former case ETHNOS has been used as a basis for the implementation of different cognitive models on individual robots. In the latter case, the ETHNOS communication facilities ported as a C-library on other control environments, provide a flexible and efficient communication layer suitable for effective implementation of a heterogeneous team of robots.

Summarizing, each of the communicating units in the architecture shown in Fig. 1, is based either on the ETHNOS system or on some other control systems enriched with the ETHNOS layer for communication. We have three other kinds of control environments, one of them being the general purpose system SAPHIRA [28], and two specialized systems used in other robotic platforms.

While the features of ETHNOS are described in detail elsewhere [37], in the next section we focus on the communication layer that provides the infra-structure for our MRS. The coordination method and the the coordination protocol are described in Section 4.

## 3. Communication Infra-structure

As already mentioned, the communication infra-structure of our MRS has been developed by relying on ETHNOS, which implements a suitable inter-expert message-based communication protocol (the EIEP - Expert Information Exchange Protocol [39]) fully integrated with the central system. The EIEP allows the system developer to decouple the different experts in execution, to reach, as close as possible, the limit situation in which the single expert is not aware of what and how many other experts are running. In this way an expert can be added, removed or modified at run-time without altering the other components of the system.

Expert decoupling is achieved by eliminating any static communication link. The EIEP is essentially an efficient implementation of a blackboard [15] in a network distributed environment. In fact, the EIEP is based on an asynchronous publish-subscribe messaging paradigm. The single expert subscribes to the particular message types, known a priori. When another expert publishes any message of the subscribed types, the subscriber receives it transparently regardless of the particular machine on which they are produced or on which an expert subscribed. This allows EIEP to be used uniformly both to design the control method within the experts of a single robot and to support transparent inter-robot communication. In fact, in ETHNOS the different experts are also allowed to subscribe to communication clubs. It is the responsibility of the system to dynamically and transparently distribute the messages to the appropriate club members.

This general methodology has been used in different applications (i.e. service robotics [38] and RoboCup [37]) and adapted in an application-dependent way by appropriately defining the number and type of clubs used. Specifically, in this RoboCup application, we are allowing the robots to communicate in a single club - the team club - (to which all of them have subscribed) and with an external supervisor (the coach) that monitors the activity of all the robots.

The EIEP also deals with low level network communication. In fact, since in RoboCup (and in general in mobile robotics) network communication is often wireless (i.e. radio link, Wavelan, etc.), due to noise or interference transmission packets are sometimes lost. In this context, both TCP-IP and UDP-IP based communication cannot be used: the former because it is intrinsically not efficient in a noisy environment; the latter because it does not guarantee the arrival of a message, nor any information on whether it has arrived or not. For this reason we have also designed a protocol for this type of applications,

called EEUDP (Ethnos Extended UDP), which extends UDP with the required features.

The EEUDP allows for the transmission of messages with different priorities. The minimum priority corresponds to the basic UDP multicast (there is no guarantee on the message arrival) and it is used for data of little importance or data that is frequently updated (for example the robot position in the environment that is periodically published). The maximum priority is similar to TCP because the message is sent until its reception is acknowledged. However, it differs from TCP because it does not guarantee that the order of arrival of the different messages is identical to the order in which they have been sent (irrelevant in ETHNOS applications because every message is independent of the others), which is the cause of the major TCP overhead. Different in-between priorities allow the re-transmission of a message until its reception is acknowledged for different time periods (i.e. 5 ms, 10 ms, etc.).

## 4. Coordination

The coordination method we present in this section is based on a coordination protocol, that is implemented on every robot of the team and is used by the robots in order to coordinate their activities for accomplishing a global team task.

The approach we adopted is a *formation/role* system similar to the ones described in [44, 42, 35, 45]. A *formation* decomposes the task space defining a set of *roles*. Each robot has the knowledge and capabilities necessary to play any role, therefore robots can switch their roles on the fly, if needed. Notice that, in an heterogeneous team, the implementation choices for each role can be different, thus having possibly different behaviors for a role, depending on the robot.

For instance, in the RoboCup environment, a basic formation could be the one where a robot takes the role of attacking going to get the ball control, another one that of defending and a third one that of supporting the attack. However, other formations are possible depending on the kind of strategy adopted (offensive, defensive) and on the need to handle special situations such as for example the malfunctioning of the goal keeper.

The coordination protocol is used by the robots in order to select the appropriate formation according to the environment conditions and to make a decision on the roles assumed by the robots in the formation.

### 4.1. Coordination protocol

The coordination protocol is based on broadcast communication of some data by every robot. These data are processed by each robot in order to establish the formation that the team will adopt and the role assigned to the robot. The computation of the coordination protocol is distributed, because each robot must process the information coming from the others in order to identify the team formation and its own role. The protocol is robust because it relies on a little amount of transmitted data.

The coordination protocol is based on the concept of *utility functions*, that are defined off-line before the actual operation of the MRS in the environment. These functions are evaluated periodically during the robot mission and exchanged among the robots. The coordination protocol includes two steps that are periodically executed on-line during the MRS mission: role assignment and formation selection. In our current implementation the protocol is executed every 100 ms (that corresponds to the perception cycle). A less frequent execution of the protocol may lead to less reactivity in role exchanging.

In the following sections we will first describe the definition of the utility functions and then the two steps that define the coordination protocol.

#### Definition of utility functions

A set of *utility functions* is defined in order to provide a quantitative measure of the effectiveness estimated by each robot in assuming a given role within the current formation. A utility function is thus associated to one role and the coordination protocol is based on exchanging the values computed for these functions among the robots and on taking a distributed decision on role assignment from these values.

More formally, a utility function $f_r^i(..)$ for a robot $i$ and a role $r$ is a function that, given the information about the status of the robot, returns a value that indicates how well can this robot play the role $r$. In other words, a utility function for a role should return higher values when the robot is in a good situation to play the role, and lower values when the robot is in a bad situation to fulfill it.

The definition of the utility functions is an important step in the design and realization of the MRS, and, since they are deeply related to the application domain of the MRS, it is not easy to develop a general methodology for defining them. The designer of the MRS must anyhow take into account two considerations: i) there are some variables or conditions that characterize the state of the robot that are relevant for the execution of the task associated to a role; ii) some parameters

of the utility functions must be experimentally evaluated, since they also depend from the characteristics of the individual robots (in an heterogeneous team).

Based on the above considerations we have performed the following steps for defining our utility functions:

1. identify the variables that are relevant for the execution of the task associated to a role;
2. define the utility function as a linear combination of these variables;
3. perform a set of systematic experiments in order to determine the coefficients of the utility functions.

For example, the utility function for the role *Attacker* in our RoboCup soccer robots is a linear combination of the following variables: distance to the ball, direction to the ball, distance to the opponent goal, direction to the opponent goal, presence of obstacles in the trajectory from the robot to the ball. The coefficients for this linear combination have been derived for every robot with the experiments described in Section 5. In particular we remark that the experiments with real robots have been fundamental for an effective coordination in our heterogeneous team, since due to the diversity of the robots, the coefficients of the utility functions are different for every robot.

### Role assignment

Role assignment is the first step of the coordination protocol and it is accomplished through explicit communication of the values of the utility functions (specific for every role), that every robot evaluates given its current local information about the environment. By comparing these values, each member of the team is able to establish the same set of assignments (robot$\leftrightarrow$role) to be immediately adopted.

More specifically, suppose we have $n$ robots $\{R_1, ..., R_n\}$ and $m$ roles $\{r_1, ..., r_m\}$. The roles are ordered by the MRS designer with respect to importance in the global task to be performed, i.e. assigning $r_i$ has higher priority than assigning $r_{i+1}$. Moreover, for each role we define a "percentage of role covering" $P_j$, that denotes how many robots of the team should be assigned to this role. For example, if we have two roles and we want to assign the first role to 60% of the team robots and the second role to the remaining 40% of the robots we have $P_1 = 0.6$ and $P_2 = 0.4$. In this way it is possible to assign a role to a robot if $n > m$. The use of the percentage values $P_j$ may require a little care when rounding them with respect to the total number of robots. A simple solution we adopted for our team is to define $P_j$ as $x/n$ ($x$ being

an integer value); while a different choice may be appropriate when a larger number of robots are present.

Let $f_j^i()$ be the value of the utility function, computed by robot $R_i$ for the role $r_j$ and $A(i) = j$ denote that the role $r_j$ is assigned to the robot $R_i$.

The method for dynamic role assignment requires that each robot $R_p$ computes the following algorithm:

**for each** role $r_j$ **do**
  compute and broadcast $f_j^p()$;
**for each** robot $R_i$ $(i \neq p)$
  **for each** role $r_j$ **do**
    collect $f_j^i()$;
$\mathcal{L} = \emptyset$; /* Empty the list of assigned robots */
**for each** role $r_j$ **do**
  **for** c = 1 to $P_j \times n$ **do**
    **begin**
      $h = argmax_{(i \notin \mathcal{L})}\{f_j^i()\}$ ;
      /* the robot $R_h$ has the highest value of $f_j^i()(i \notin \mathcal{L})$ */
      **if** $h = p$ **then** $A(p) = j$; /* the role for $R_p$ is $r_j$ */
      $\mathcal{L} = \mathcal{L} \cup \{h\}$;
    **end**

It is easy to see that every role is assigned to at most $P_j \times n$ robots and that every robot is assigned to only one role. The reason is that at every cycle of the algorithm a different assignment $A(i) = j$ is done: $j$ changes after $P_j \times n$ cycles and robots already included in the set $\mathcal{L}$ of assigned robots cannot be chosen for further assignments.

In particular, the first role (i.e. the one with the highest priority), will be assigned to those robots that have the best utility values for the role $r_1$, the second role to those among the remaining robots that have the best utility values for the second role, and so on. While in the case of a complete lack of communication all the robots will assume the most important role ($r_1$).

It is important to notice here that the utility functions we have defined in our approach do not represent a metric evaluation of the performance of the MRS (as in [35]), but the best way to accomplish every single task (independently from the others). Therefore, even though our algorithm can be considered as a greedy approach to the *Heterogeneous Robot Action Selection Problem (HRASP)* defined in [35], we remark that our utility functions are not semantically equivalent to the metric evaluation functions. Moreover, while HRASP has been proved to

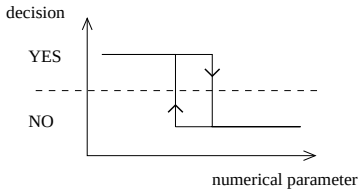



*Figure 2.* The hysteresis mechanism in decisions

be NP-hard [35] and thus heuristics must be developed to cope with this problem, our objective is not to compute a role assignment that maximize the sum of the utility functions, but to assign a priority to every role and to guarantee that high priority roles are assigned to the robots that are in the best conditions to perform the associated tasks.

This issue is very important in many dynamic and adversarial environments and especially in the RoboCup environment, in which this work has been originally developed. Consider a situation in the RoboCup environment in which two robots coordinate on the two roles: *attacker* and *defender*. For an effective game it is important that the role *attacker* is assigned to the robot that is in the best condition to reach the ball before the opponents. Now consider the following values for the utility functions of these two roles: $f_a^1(..) = 10, f_d^1(..) = 6$ for robot 1 and $f_a^2(..) = 6, f_d^2(..) = 1$ for robot 2. Our algorithm will assign robot 1 to the role *attacker* and robot 2 to the *defender*, while if we had chosen to maximize the sum of the utility functions we would have the opposite assignment. However, in this scenario, the second choice may be ineffective since it does not take into account the different priorities of the two roles and may lead to a situation in which robot 2 will fail to reach the ball and to act as *attacker*, while robot 1 (that was in the best condition to get the ball and attack) will perform a defensive action possibly loosing the opportunity to control the ball.

In order to obtain an effective application of the above method, an important issue to be dealt with is the stability of decisions with respect to possible oscillations of the numerical parameters on which they depend upon (see also [24]).

The method we adopted to stabilize decisions is based on the notion of *hysteresis* (see Fig. 2), which amounts to smoothing the changes in the parameter values. This technique prevents a numerical parameter's oscillation from causing oscillations in high level decisions. In the case of coordination, for instance, if at a certain instant robot $R_i$ covers role $r_j$, its utility function $f_j^i()$ for role $r_j$ returns a higher value.

Another critical factor for the correct operation of coordination is the capability of each element to realize a sudden difficulty in perform-

ing its task. For instance, a robot that is moving toward the ball can get stuck on its way. Once it has realized this circumstance, all its utility functions must return low values so that the the role can be assigned to other robots.

Finally, if all the robots possess the same data (i.e. communications are working correctly), they will compute the same assignment, but in case of a great loss of transmitted data due to interferences, the robots may have slightly inconsistent data. Therefore, there could be roles temporarily assigned to more than one robot or not assigned at all. However, holes in data transmission last in general fractions of a second. So, if we assume that the values of the "utility functions" do not change sharply, the correct use of the hysteresis method guarantees that the roles will be correctly assigned almost always (as shown by the experimental data we have collected during the games and are discussed in the next section).

### Formation selection

The robots have at their disposal a number of predefined formations and rules to select the formation to adopt, on the basis of the environment configuration. Since each robot status do not necessarily coincide with those of the others (because of possible communication failures or different views of the world), the robots may choose different formations. Therefore the formation selection algorithm is based on a voting scheme that allows for changing the formation only in presence of the absolute majority of votes.

**for each** robot $R_i$ of the team **do**
  **begin**
    collect $voted\_formation[i]$;
    $votes[voted\_formation[i]] = votes[voted\_formation[i]] + 1$;
  **end**
**if** there is a formation $f$ such that $votes[f] > n/2$ **then**
  $selected\_formation = f$;

This voting scheme resolves the conflict arising by lack of absolute majority by leaving the formation unchanged. The algorithm also limits significantly the risk of oscillations in formation selection as well as the frequency of changes. Moreover, to ensure the stability of decisions, the formation selection is accomplished at a lower frequency to that of role selection (i.e. once per second).

## 5. Applications and experiments

A successful coordination of the team depends on the effectiveness of the coordination protocol and on a suitable calibration of some parameters, such as the coefficients of the utility functions. Calibration of these parameters typically requires a significant experimental work.

In addition, in an heterogeneous team of robots the experimental phase is particularly demanding, since the exchanged information are computed and interpreted differently by each element of the team. For example, consider the evaluation of reachability of the ball: each robot may have a mobile base with different capabilities and a set of behaviors with specific speed characteristics and, that notwithstanding, robots must calculate comparable numerical values.

In this section we describe methods and tools that have been used for developing and evaluating the approach described above and we present the results of our experiments in a competitive and highly dynamic environment such as the one provided by the RoboCup matches.

### 5.1. Methods and tools

The experimentation of the coordination protocol must be done in stages which require the use of different tools: a simulator, experimentation without play, experimentation during actual games, and analysis of log files of the games.

The first and easier experimental setting is provided by a simulator. Even though simulation cannot provide a precise characterization of all the aspects that influence the performance of the robot in the real environment, it is very useful both for verifying the correctness of the protocol and for computing a first estimation of the coefficients of the utility functions, that will be refined with subsequent experiments involving real robots.

First experiments with real robots may be done without playing, with steady robots and moving the ball and opponents. At this stage one needs to adjust the discrepancies arising from differences in heterogeneous robots' implementations. In particular, we compare the sensing capabilities of the robots and adjust the coefficients of their utility functions. Notice that while these aspects could be, in principle, resolved through the simulator, this requires a rather complex simulation model that is very difficult to build. Similarly, learning techniques could be used, but they are outside the scope of the present work.

The other experimental phase involving the robots consists in looking at the game and in singling out the failures of the coordination system. For example, a typical task is that of identifying situations

where the most suitable player does not move towards the ball (take role *Attacker*) and adjust the parameters to restore the expected behavior.

To this end, an analysis of the log files generated during the games is very useful for identifying misbehaviors of the coordination system. In this respect, we have developed a 3D viewer for experimental data that allows for displaying several information about one or more robots from the log files of real games or portion of them. By analyzing the data collected by the robots during the game through our graphical tool for 3D navigation of these data, we are able to detect several interesting features of coordination as well as unexpected behaviors of the robots, such as unassigned roles, oscillations in role exchange, overlapping roles, etc. Specifically, we have used this tool also to further refine the utility functions.

### 5.2. The RoboCup Environment

The coordination method presented in the previous section has been implemented within the Azzurra Robot Team, the Italian national team of heterogeneous robotic soccer players, participating in the RoboCup competitions in the Middle-Size League [31].

Coordination in the RoboCup environment is an important issue because of the possibility of implementing effective team strategies for the game. In fact, coordination among the ART players requires not only a distributed coordination protocol, but also a very flexible one, that allows the coach to accommodate the various configurations that can arise by forming teams with different basic features. The performance of the ART team provided substantial evidence that basic coordination among the team players has been successfully achieved. In several situations where two team players were close to the ball, they were able to smoothly switch their roles and managed to get ball possession without obstructing each other; in addition, they have generally occupied the field in a satisfactory way, as shown in the coordination analysis presented in Section 5.4.

The coordination protocol used in this setting is based on a set of formations and a set of roles for every formation. The formation that has been mostly used is the *standard formation* that is described below. Other formations have been considered in order to deal with special situations, like goalkeeper out of the game.

In the *standard formation* we have defined 3 roles for the 3 players: *Attacker, Defender, Support*. The *utility functions* for these roles are defined as a linear combination of distance from the ball, position and orientation of the robot in the field, and obstacles in the path towards the ball. Since we have 3 roles for 3 robots the *percentage role covering*

$P_j$ are set to 1/3, so that every role is assigned to one robot. Moreover, the roles have priorities and thus if for example one robot is out or does not communicate with the others the first two roles (*Attacker, Defender*) are assigned to the remaining two robots, leaving the *Support* role unassigned.

The algorithm for dynamic role assignment has been used for effective role switching between robots during the games and the hysteresis in the utility functions have reduced the possibility of too frequent role switching. The performance has also been evaluated through a set of systematic experiments on the robots and by the data collected during actual games in the official games of RoboCup 1999 and 2000. We have reported the results of the analysis on communication and coordination in the next sections.

### 5.3. Communication analysis

The reliability of communication with the EIEP has been experimentally verified. ETHNOS system allocates a maximum guaranteed and dedicated time to network communication. Since ETHNOS schedules all tasks in real-time according to the Rate Monotonic scheduling policy, the dedicated time value is computed automatically on the basis of the schedulability analysis so that the real time execution of the whole set of tasks (i.e. user-defined and communication tasks) is guaranteed. Thus, the dedicated time depends on the computational load of the tasks in execution as well as on the processor speed.

In Fig. 3 the diagrams represent different machines (with different processing power) corresponding to two robots (Relé - an AMD-K6 200MHz - and Homer - a AMD-K6 233Mhz -) and a monitor (an Intel Pentium II 300 Mhz), connected using radio ethernet (2 Mbit bandwidth Wavelans). The top line in the diagram indicates the calculated time available each 100 ms for communication purposes. Clearly, this value decreases as the number of tasks in execution increases (and, consequently, the computational load). The bottom line indicates the time spent in communication, which also increases with the number of experts (this is because in this experiment we have assumed that the activity of every expert involves either transmission or reception of messages). In this way it is always possible to determine a priori whether the system is capable of both communicating information and executing in real time.

Finally, it is worth considering the transmission complexity which determines the bandwidth requirements. In general, if we have a system composed of $n$ robots and $m$ roles we have a complexity $O(mn)$ corresponding to the number of utility values that have to be transmitted. In

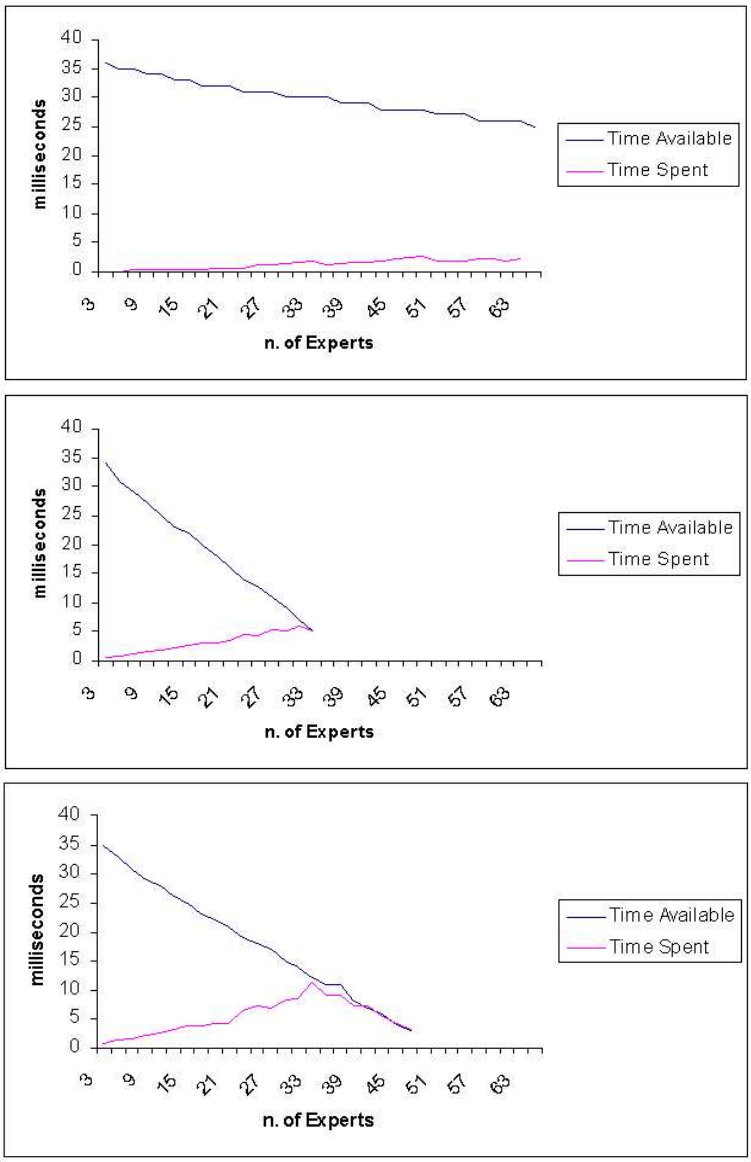



*Figure 3.* Network Communication in ETHNOS. From top to bottom: the monitor, Relé, Homer

fact, according to ETHNOS broadcast communication strategy, if the number of roles is fixed the complexity linearly increases with respect to the number of robots and viceversa. This typically occurs in many applications in which the number of roles is significantly lower than the number of robots and it is fixed and related to the specific application

Table I. Robots' roles.

| Match | Attacker | Support | Defender |
|---|---|---|---|
| 1 | 82.9% | 39.5% | 98.2% |
| 2 | 84.6% | 98.0% | 80.8% |
| 3 | 87.6% | 38.5% | 90.0% |
| 4 | 89.5% | 81.8% | 96.9% |
| 5 | 93.5% | 84.1% | 98.9% |
| Avg. | 87.6% | 68.3% | 93.0% |

Table II. Robot position in the field.

| Match | Forward | Middle | Backward |
|---|---|---|---|
| 1 | 78.8% | 68.7% | 97.8% |
| 2 | 65.4% | 68.8% | 98.6% |
| 3 | 53.2% | 54.5% | 99.3% |
| 4 | 23.5% | 80.4% | 93.1% |
| 5 | 64.1% | 72.1% | 98.9% |
| Avg. | 57.0% | 68.9% | 97.5% |

domain. In addition, the size of messages is limited and the bandwidth required is thus very low.

### 5.4. Coordination analysis

The evaluation of the coordination protocol in this setting has been carried out by analyzing a set of log files acquired during the matches in the European Championship 2000. We present here the results of this analysis both for the communication and for the coordination layer.

A quantitative analysis of the coordination protocol has been worked out through the collection of the log files of various games.

Table I describes the percentage of time in which a role is assigned to at least one robot. Since the roles are ordered in terms of importance within a formation, some roles are more likely to be assigned. In particular *Attacker* and *Defender* were almost always assigned, while for instance the role *Support* was not assigned when there were only two players in the field. Therefore, with respect to a static assignment of roles, dynamic assignment provides a good distribution of the roles, but

with the advantage of selecting the more appropriate robot for every role depending on the current situation of the environment.

Table II shows the coverage of the field by the three middlefield robots (i.e. the percentage of time in which at least one robot was in a zone of the field). It is interesting to notice that the field has been properly covered, even if there is not an explicit split of the field in sections assigned to the robots. In particular, the defensive area has been occupied by at least one robot almost at every time.

Another analysis shows that during the game there is an average of one role switch every 10 seconds. Due to occasional loss of transmitted data, we noticed that about 1/10 of the role switches generates roles' oscillation, lasting about 300 ms before stabilization.

Finally, by a comparison between the logs and a visual review of the game, we have discovered that one relatively frequent source of failure in coordination arises from situations in which a robot is not actually able to correctly evaluate the utility function for a certain role. This situation arise when there is a problem in the vision system which causes the robot to see the ball very close to it, while it is not. This robot gets the role *Attacker* with the goal of going to the ball, but it will be unsuccessful while blocking the exchange of roles. This behavior shows that the coordinated system is highly influenced by the individual robot failure. In such cases the overall performance of the team is not good, since a robot that is actually in a better position to go will not assume the correct role. The problem has been successfully addressed by decreasing the utility function when the robot does not make progress in its task. This is obtained by a task-dependent evaluation of the progress of the robot in performing the task. For example, the evaluation of the progress of the task associated to our role *Attacker* is given by the computation of the distance to the ball that must decrease in time. If the distance to the ball does not decrease the utility function for the role *Attacker* returns a lower value, allowing other robots to possibly take the role.

## 6. Related work

Recently, there has been a growing interest in multi-agent robotic systems, as witnessed by the great number of scientific publications in the field (for a survey see [14, 11, 25, 7]). Research has mostly been focused on specific aspects (distributed system architectures, communication modalities, coordinated motion, cooperation and competition, etc.), on particular environments (military [32, 29], service [17] [16], extra-terrestrial planetary exploration domain [30], etc.).

In this section we identify those aspects of the MRS discussed in the literature that are more closely related to our work. Subsequently, we will address RoboCup related approaches.

From the architectural point of view our approach requires only a common communication layer, whereas significantly different architectural models can be adopted. This is witnessed by the different architectures (behaviors based, hybrid, fuzzy-logic based) of the robots in the ART team. Other approaches impose more rigid constraints. For example in [10, 33] subsumption-based reactive control is adopted for robot societies consisting of up to 20 agents. ALLIANCE [34] and BLE [45] provide coordination among the robots by an inhibition mechanism that is used by one robot to disable the execution of a behavior on other robots. Robots in these systems are designed in a behavior-based architecture. A more general approach is proposed in [2, 6] where the AuRa hybrid architectural model is extended to handle multi robot coordination in hostile environment exploration for urban warfare and for navigation in military formation. Also the MURDOCH architecture [20] defines a distributed communication system based on a publish/subscribe paradigm that may be implemented on different robotic architectures.

Heterogeneity is a fundamental feature of our approach and in fact the robots in the ART team differ not only in the control architecture, but also in the mechanics, sensors and actuators as well as in the robot shape and appearance. It is important to emphasize that in our approach heterogeneity is a requirement that the robotic system must handle (the robots to be coordinated have been developed independently and beforehand by separate research groups) and that every robot is able to perform every task, but possibly with different performance. Other approaches in literature have typically dealt with heterogeneity in a different way. For example in [41, 35]. heterogeneity is specifically designed and adopted to better accomplish a given task.

Our approach differs from many other proposed systems also in the role of coordination to address the given task. In fact we are not simply interested in finding a spatial distribution of the robots in order to optimize the execution of a cooperative task (as in typical coverage problems [19, 23], such as mine collecting [1] or floor cleaning [26]); instead, we aim at finding a solution for the more general problem, where robots are given a set of criteria and an arbitration algorithm to negotiate the role that each of them must assume (in our definition, selecting a role means choosing and activating the group of concurrent behaviors that are associated with that specific role). This problem, often related to the presence of a robotic or human opponent, has been considered relevant in military applications for Urban Warfare. In this

scenario a group of robots are deployed into an unknown, possibly hostile environment (for example a building occupied by enemy forces) with the purpose of exploring it and report to a base station what they have discovered (as in the DARPA Tactical Mobile Robot domain [8]).

The problem of dynamic task assignment has been specifically addressed also in some recent works [45, 20, 43, 40]. The work in [45] describes an approach to distributed task assignment similar to the one presented here. In order to assign a task every robot collects a value (that can be seen as the result of our utility functions) from all the other robots, the robot with the maximum value can send an inhibition signal to the other robots in order to disable their execution of the behavior associated to this task. As already mentioned before, this mechanism is based on a Port-Arbitrated Behavior-Based architecture. In the MURDOCH system [20] tasks are organized in a structure similar to a Hierarchical Task Network and task allocation is computed by a sequence of "one-round auctions". When a task is to be executed by the MRS an auction mechanism allows for deciding which robot is in the best condition to perform the task by means of the evaluation of some *metric functions*, that are similar to our utility functions. A difference with respect to our system is that we evaluate our utility functions periodically during the mission, so that if one robot happens to be in a better condition to do a task a role switching is performed. The work in [43] addresses the issue of "module conflict", that is needed for avoiding that two or more robots execute incompatible tasks. The authors define the concept of *module*, that is slightly different from our definition of *role*. While their modules are associated to the basic tasks that can be executed by the robots, our roles define high-level tasks that must be accomplished by the robots (possibly executing simpler subtasks). Therefore, the association between robots and roles is one-to-one in our approach, while it is many-to-many in [43]. The module conflict resolution mechanism in [43], that is used for dynamic task assignment, is based on the evaluation of a correlation function that defines incompatibility among the modules. This function is defined by the user for every pair of modules. Finally, in [40] tasks are organized in Distributed Organizational Task Networks (DOTN) and task allocation is performed by a heuristic search in the space of possible assignments between robots and DOTN. Task assignment is not based on the evaluation of a function for measuring the quality of task allocation, but on minimizing the dependencies among the tasks assigned to different robots, so that each robot can perform its tasks in a more independent way.

Among the proposals for robot coordination developed in the RoboCup environment, we may find both centralized and distributed ap-

proaches. Centralized ones are mostly adopted in the Small-size League where a global reconstruction of the environment is possible (because of the availability of a global vision system) and when a team of robots can reliably reconstruct global information about the environment from local sensors. For example the CS Freiburg team in the Middle-size League [22] makes use of very precise measurements given by laser scanners to reconstruct a global representation of the field and a centralized coordination is realized by dynamic selection of the robot that acts as the leader. However, possible communication failures, that are very common in the RoboCup environment, as well as the general difficulty of reconstructing a global reliable view of the environment, require full autonomy on each robot and distributed approaches are usually preferred in this context. A distributed approach that is not based on explicit communication is described in [18], in which two robots are able to exchange the ball by using direct communication of their roles and a field-vector based collision avoidance that takes into account in a proper way the role of the robots. Other behavior based reactive MRS are presented in [9, 12]. Our approach exhibits the following two features: on one hand, it exploits explicit communication among the robots for exchanging information about the status of the environment in order to achieve a distributed agreement on the actions to be performed; on the other hand, it is robust to possible communication failures, since in these cases the system degrades its performance, but it still keeps on the execution of the task.

## 7. Conclusions

In this article we have described an approach to distributed coordination of a team of heterogeneous mobile robots, whose main features are: explicit communication among the robots, autonomy in coordination and heterogeneity of robots with similar capabilities. The approach was originally developed for the RoboCup environment, and it is thus suitable for highly dynamic and hostile environments.

The successful application of the method has proved its effectiveness in a challenging scenario for MRS coordination. In fact, the coordination method described here has been a critical factor in the performance of the ART robot team for the Stockolm 1999 and Amsterdam 2000 RoboCup competitions, where the ART has classified in the second place for both of these competitions. ART soccer team was a highly heterogeneous team, since different players were designed by different research groups exploring different hardware and software solutions. The proposed system architecture allowed to easily integrate together

the results of different researchers across different groups and to easily implement protocols for inter-robot coordination (by hiding to the programmers the details of communication). It is important to emphasize that in our approach heterogeneity is a requirement that the robotic system must handle (the robots to be coordinated have been developed independently and beforehand by separate research groups). Moreover, as a difference with other heterogeneous MRS [41, 35], in our case all the robots are able to perform all the tasks, but with different performance that is taken into account in the coordination protocol. We believe that the availability of robots with different features and designed by different manufacturers is a realistic scenario in the next future.

Another important contribution of this article is in the definition of methods and tools for an experimental evaluation of the performance of the proposed coordination method. In particular the analysis of communication packets has proved the effectiveness and reliability of the EIEP for robots' information exchanging, while the analysis on the coordination protocol has showed that several properties (like area coverage and role assignment), that are important for the environment in which coordination has been applied, have been substantially achieved.

One critical aspect of distributed coordination in a MRS is the robustness of the method, with respect to failures and malfunctioning of the robots. The method described in this paper made the team sufficiently robust to perform successfully during the competitions. However we believe that the methods for achieving robustness of the team performance deserve further investigation. Another important issue is the evaluation of the effectiveness of coordination methods. We are currently addressing this aspect both by improving the techniques and tools proposed in the paper and by addressing the analysis of a wider set of properties that are related to coordination and team work.

## Acknowledgments

We would like to thank all the members of ART, who contributed to the development of the team. The experimentation of robotic team work has been greatly supported by the whole team. We would like to thank Claudio Castelpietra and Alessandro Scalzo particularly for their contribution to the implementation of the proposed method. We acknowledge the support of Consorzio Padova Ricerche, CNR, University of Rome "La Sapienza", and University of Genova.

## References


1. Acar, E., Y. Zhang, H. Choset, M. Schervish, A. G. Costa, R. Melamud, D. Lean, and A. Graveline: 2001, 'Path Planning for Robotic Demining and Development of a Test Platform'. In: *Field and Service Robotics.*
2. Arkin, R., T. Collins, and T. Endo: 1999, 'Tactical Mobile Robot Mission Specification and Execution'. In: *Mobile Robots XIV.* Boston, USA, pp. 150–163.
3. Balch, T.: 1999, 'The impact of diversity on performance in multi-robot foraging'. In: *Proc. of Agents '99.*
4. Balch, T.: 2000, 'Hierarchic social entropy: an information theoretic measure of robot team diversity'. *Autonomous Robots* **8**(3).
5. Balch, T. and R. C. Arkin: 1995, 'Communication in Reactive Multiagent Robotic Systems'. *Autonomous Robots* **1**(1), 27–52.
6. Balch, T. and R. C. Arkin: 1998, 'Behavior-based Formation Control for Multi-robot Teams'. *IEEE transactions on robotics and automation* **14**(6).
7. Balch, T. and L. E. Parker (eds.): 2002, *Robot Teams: From Diversity to Polymorphism.* A K Peters Ltd.
8. Blitch, J.: 1999, 'Tactical Mobile Robots for Complex Urban Environments'. In: *Mobile Robots XIV.* Boston, USA, pp. 116–128.
9. Brendenfeld, A. and H. U. Kobialka: 2000, 'Team Cooperation using Dual Dynamics'. In: *Proc. of ECAI2000 Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems.*
10. Brooks, R., P. Maes, M. Mataric, and G. More: 1990, 'Lunar Base Construction Robots'. In: *IEEE International Workshop on Intelligent Robots and Systems.* Tsuchiura, Japan, pp. 389–392.
11. Cao, Y. U., A. Fukunaga, and A. Kahng: 1997, 'Cooperative Mobile Robotics: Antecedents and Directions'. *Autonomous Robots* **4**, 1–23.
12. Carpin, S., C. Ferrari, F. Montesello, E. Pagello, and P. Patuelli: 2000, 'Scalable Deliberative Procedures for Efficient Multi-robot Coordination'. In: *Proc. of ECAI2000 Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems.*
13. Castelpietra, C., L. Iocchi, D. Nardi, M. Piaggio, A. Scalzo, and A. Sgorbissa: 2000, 'Coordination among heterogenous robotic soccer players'. In: *Proc. of International Conference on Intelligent Robots and Systems (IROS'2000).*
14. Dudek, D., M. Jenkin, E. Milios, and D. Wilkes: 1996, 'A Taxonomy for Multi-Agent Robotics'. *Autonomous Robots* **3**(4), 375–397.
15. Engelore, R. and T. M. (eds): 1988, *Blackboard systems.* Addison-Wesley.
16. Evans, J.: 1995, 'HelpMate: a Service Robot Success Story'. *ServiceRobot: An International Journal* **1**(1), 19–21.
17. Everett, H. R. and D. Gage: 1996, 'A Third Generation Security Robot'. In: *SPIE Mobile Robot and Automated Vehicle Control Systems.* Boston, USA, pp. 20–21.
18. Ferraresso, M., C. Ferrari, E. Pagello, R. Polesel, R. Rosati, A. Speranzon, and W. Zanette: 2000, 'Collaborative Emergent Actions Between Real Soccer Robots'. In: *RoboCup-2000: Robot Soccer World Cup IV.*
19. Fontan, M. and M. Mataric: 1998, 'Territorial multi-robot task division'. *IEEE Transactions on Robotics and Automation* **15**(5).
20. Gerkey, B. P. and M. J. Matarić: 2000, 'Principled Communication for Dynamic Multi-Robot Task Allocation'. In: *Proceedings of the International Symposium on Experimental Robotics.* Waikiki, Hawaii.

21. Guibas, L. J., J. Latombe, S. LaValle, and D. Lin: 1999, 'A Visibility-Based Pursuit-Evasion Problem'. *International Journal of Computational Geometry and Applications* **9**, 471–494.
22. Gutmann, J.-S., T. Weigel, and B. Nebel: 1999, 'Fast, Accurate, and Robust Self-Localization in the RoboCup Environment'. In: *RoboCup-99: Robot Soccer World Cup III*. pp. 304–317.
23. H., C.: 2001, 'Coverage for robotics - A survey on recent results'. *Annals of Mathematics and Artificial Intelligence* **31**, pp 113–126. Kluwer academic Publishers, printed in the Netherlands.
24. Hannebauer, M., J. Wendler, P. Gugenberger, and H. Burkhard: 1998, 'Emergent Cooperation in a Virtual Soccer Environment'. In: *Proc. of DARS-98*.
25. Iocchi, L., D. Nardi, and M. Salerno: 2001, 'Reactivity and Deliberation: a survey on Multi-Robot Systems'. In: E. P. M. Hannebauer, J. Wendler (ed.): *Balancing Reactivity and Deliberation in Multi-Agent Systems (LNAI 2103)*. Springer, pp. 9–32.
26. Jung, D. and A. Zelinsky: 2000, 'Grounded Symbolic Communication Between Heterogenoeus Cooperating Robots'. *Autonomous Robots* **8**(3).
27. Kitano, H., M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara: 1998, 'RoboCup: A Challenge Problem for AI and Robotics'. In: *Lecture Note in Artificial Intelligence*, Vol. 1395. pp. 1–19.
28. Konolige, K., K. Myers, E. Ruspini, and A. Saffiotti: 1997, 'The Saphira Architecture: A Design for Autonomy'. *Journal of Experimental and Theoretical Artificial Intelligence* **9**(1), 215–235.
29. Lee, J., M. Huber, E. Durfee, and P. Kenny: 1994, 'M-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications'. In: *AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space*.
30. Miller, D.: 1990, 'Multiple Behavior-Controlled MicroRobots for Planetary Surface Missions'. In: *IEEE International Conference on Systems, Man, and Cybernetics*. Los Angeles, USA, pp. 289–292.
31. Nardi, D., G. Adorni, A. Bonarini, A. Chella, G. Clemente, E. Pagello, and M. Piaggio: 1999, 'ART-99: Azzurra Robot Team'. In: *RoboCup-99: Robot Soccer World Cup III*.
32. Noreils, F. R.: 1992, 'Battlefield Strategies and Coordination between Mobile Robots'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 1777–1784.
33. Parker, L.: 1993, 'Adaptive Action Selection for Cooperative Agent Teams'. In: *Proc. of 2nd International Conference on the Simulation of Adaptive Behavior*. Honolulu, USA, pp. 442–450.
34. Parker, L. E.: 1998, 'ALLIANCE: An Architecture for fault Tolerant Multirobot Cooperation'. *IEEE Transactions on Robotics and Automation* **14**(2), 220–240.
35. Parker, L. E.: 2000, 'Lifelong Adaption in Heterogeneous Multi-Robot Teams: Response to Continual Variation in Individual Robot Performance'. *Autonomous Robots* **8**(3), 239–267.
36. Piaggio, M., A. Sgorbissa, and R. Zaccaria: 2000a, 'Pre-emptive versus non-pre-emptive real time scheduling in intelligent mobile robotics'. *Journal of Experimental and Theoretical Artificial Intelligence* **12**, 235–245.
37. Piaggio, M., A. Sgorbissa, and R. Zaccaria: 2000b, 'A programming environment for real time control of distributed multiple robotic systems'. *Advanced Robotics* **14**(1), 75–86.

38. Piaggio, M., A. Sgorbissa, and R. Zaccaria: 2001, 'Autonomous Navigation and Localization in Service Mobile Robotics'. In: *Proc. of IROS'01.*
39. Piaggio, M. and R. Zaccaria: 1997, 'An Information Exchange Protocol in a Multi-Layer Distributed Architecture'. In: *IEEE Proc. Hawaii International Conference on Complex Systems.*
40. Shen, W. M. and B. Salemi: 2002, 'Distributed and Dynamic Task Reallocation in Robot Organizations'. In: *Prooceedings of the 2002 IEEE Int. Conference on Robotics and Automation.* Washington, DC.
41. Simmons, R., D. Apfelbaum, D. Fox, R. P. Goldman, K. Z. Haigh, D. J. Musliner, M. Pelican, and S. Thrun: 2000, 'Coordinated Deployment of Multiple, Heterogeneous Robots'. In: *Proc. of International Conference on Intelligent Robots and Systems (IROS'2000).*
42. Stone, P. and M. Veloso: 1999, 'Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork'. *Artificial Intelligence* **110**(2), 241–273.
43. Uchibe, E., T. Kato, M. Asada, and K. Hosoda: 2001, 'Dynamic Task Assignment in a Multiagent/Multitask Environment based on Module Conflict Resolution'. In: *Proced. of the 2001 IEEE International Conference on Robotics and Automation.* Seoul, Korea.
44. Veloso, M. and P. Stone: 1998, 'Individual and Collaborative Behaviors in a Team of Homogeneous Robotic Soccer Agents'. In: *Proceedings of the Third International Conference on Multi-Agent Systems.* pp. 309–316.
45. Werger, B. B. and M. J. Mataric: 2000, 'Broadcast of Local Eligibility for Multi-Target Observation'. In: *Proc. of DARS.*

*Address for Offprints:* Kluwer Prepress Department
P.O. Box 990
3300 AZ Dordrecht
The Netherlands

In *the 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation*

# Cooperative Tracking using Mobile Robots and Environment-Embedded, Networked Sensors*

**Boyoon Jung and Gaurav S. Sukhatme**

*boyoon|gaurav@robotics.usc.edu*

**Robotic Embedded Systems Laboratory**
**Robotics Research Laboratory**
**Department of Computer Science**
**University of Southern California**
**Los Angeles, CA 90089-0781**

## Abstract

We study the target tracking problem using multiple, environment-embedded, stationary sensors and mobile robots. An architecture for robot motion coordination is presented which exploits a shared topological map of the environment. The stationary sensors and robots maintain region-based density estimates which are used to guide the robots to parts of the environment where unobserved targets may be present. Experiments in simulation show that the region-based approach works better than a naive target following approach when the number of targets in the environment is high.

## 1 Introduction

Autonomous target tracking has many potential applications; e.g. surveillance, security, etc. Mobile robot-based trackers are attractive for two reasons: they can potentially reduce the overall number of sensors needed and they can adapt to the movement of the targets (e.g. follow targets to occluded areas). The robot-based target tracking problem (CMOMMT: Cooperative Multirobot Observation of Multiple Moving Targets), has been formally defined in [1] and has received recent attention in the robotics community[2, 3].

The CMOMMT problem is defined as follows. Given a bounded, enclosed region $S$, a team of $m$ robots $R$, a set of $n$ targets $O(t)$, and a binary variable $In(o_j(t), S)$ defined to be true when target $o_j(t)$ is located within region $S$ at time $t$, and $m \times n$ matrix $A(x)$ is defined where

$$a_{ij}(t) = \begin{cases} 1 & \text{if a robot } r_i \text{ is monitoring target } o_j(t) \\ & \text{in } S \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

and *the logical OR* operator is defined as

$$\bigvee_{i=1}^{k} h_i = \begin{cases} 1 & \text{if there exists an } i \text{ such that } h_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

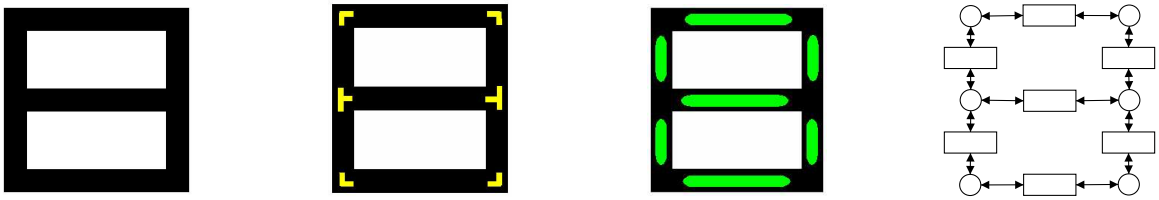
The goal of the CMOMMT is to maximize the observation.

$$Observation = \frac{\sum_{t=0}^{T} \sum_{j=1}^{m} \bigvee_{i=1}^{k} a_{ij}(t)}{T \times m} \qquad (1)$$

In [1, 2], the ALLIANCE architecture was used to coordinate robots in the CMOMMT task; role assignment among mobile robots was achieved implicitly through one-way communication. However, it was assumed that the observation sensors had a perfect field-of-view and a known global coordinate system. Experiments were performed in a bounded, enclosed spatial region, and an indoor global positioning system was utilized as a substitute for vision or range-sensor-based tracking. In [3], an approach to a similar problem using the BLE (Broadcast of Local Eligibility) technique was presented which used a real video camera to track moving objects, and one-way communication for explicit role-assignment. However, the environment in [3] was very simple, and movements of targets were pre-programmed. Each target was also identified a priori. In [4], a Variable Structure Interacting Multiple Model (VS-IMM) estimator combined with an assignment algorithm for tracking multiple ground targets was described.

In this paper, we consider a more realistic office-like environment. It consists of corridors; offices will be added in the near future. The major difference from previous research is to utilize environment-embedded, stationary sensors installed at fixed positions in the environment. These sensors are used to track moving targets in their sensor range, and broadcast target location information over a wireless channel. The mobile robots are used to explore regions which are not covered by the fixed sensors. The robots also broadcast the tracked target location information. We

*This work is sponsored in part by DARPA grant DABT63-99-1-0015 and NSF grants ANI-9979457 and ANI-0082498



(a) Environment (b) Landmarks (c) Regions (d) Topological map

Figure 1: A structured environment segmented into landmarks and regions

present a region-based strategy for robot coordination which uses a topological map, and compare it to a naive 'target-following' strategy using the observation metric similar to Equation 1. Our results show that the region-based strategy works better than the naive strategy when the number of targets in large.

In Section 2, the region-based method and system architecture is described. The simulation environment and experimental results are discussed in Section 3. Concluding remarks and future work is discussed in Section 4.

## 2 Region-based Robot Coordination

When the environment is an empty open space, the main challenge is to assign targets to a fixed number of robots based on the distances between robots and targets. However, when the environment has structure (e.g. office-type environment), it is important to disperse robots properly. We propose a region-based approach for this purpose.

### *2.1 Assumptions*

We make several assumptions about the environment and robot capabilities. First, a topological map of the environment is assumed to be given. Previous research on map building is extensive [5, 6, 7]. In [5], a simple, modular, and scalable behavior-based technique for incremental on-line mapping is presented, and in [6], a simple, yet robust cooperative mapping method using multiple robots is presented. In [7], a probabilistic approach to building large-scale maps of indoor environments with mobile robots is presented. In this paper, the data structures from [6] have been adopted to build a topological map.

The second assumption is that global communication between robots and the fixed sensors is allowed. However, this does not imply two-way communication, like a negotiation. We only use one-way broadcast among sensors and robots; whenever a sensor detects a moving target, the sensor broadcasts the estimated position of the target. Perfect communication is not necessary either; a small rate of packet loss will not degrade the performance of the system.

Third, the initial position of the mobile robots is assumed to be known for localization. Since odometry is used for localization in the experiments reported here, the initial positions of the mobile robots must be known. However, localization information is used only for estimating the positions of moving objects, not for navigation. Navigation is based on a landmark detector, not global positioning.

### *2.2 The Region-based Method*

The basic idea of the region-based approach is that the environment can be divided into several (topologically simple) regions using landmarks as demarcaters. In Figure 1 (a), a simple office-type environment that consists of corridors is shown, (b) shows landmarks, (c) shows how the environment can be divided into regions by the landmarks, and (d) is a topological map of the environment.

Assuming that a topological map is given, we need to decide which region "needs" more robots and which region does not. In order to answer this question, each region is assigned two properties: a *robot density* ($D_r$) and a *target density* ($D_t$). They are defined as follows:

$$D_r(r) = \frac{\text{The number of robots in region } r}{\text{Area of region } r} \quad (2)$$

$$D_t(r) = \frac{\text{The number of targets in region } r}{\text{Area of region } r} \quad (3)$$

Robot density indicates how many robots[1] are in a region, and target density indicates how many tracked targets are in a region. The definition for $D_t(r)$ is not complete as stated in Equation (3); as we explain later, $D_t(r)$ can also assume negative values if no moving objects are detected in region $r$. Both values are normalized by area. If a region has low robot density and high target density, the region needs more mobile robots, and vice versa.

Sometimes, a robot must stay in its current region even though there is another region that needs more robots; for example, when it is the only robot tracking objects in its region or when there are too many

[1]Embedded stationary sensors are counted as robots when robot density is calculated.

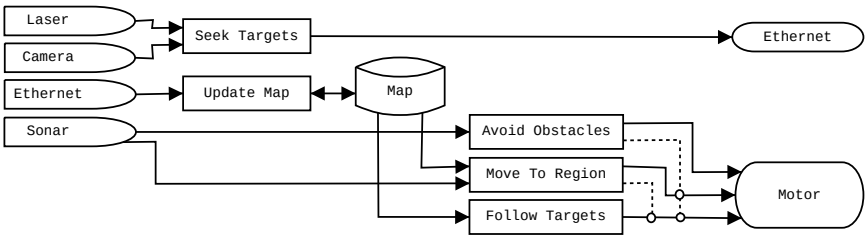



Figure 2: System architecture for mobile robots and embedded sensors

moving objects in the region. Therefore, each robot must check its *availability* on the basis of the following criteria:

$$D_t(r_c) < 0 \tag{4}$$

$$\frac{D_t(r_c)}{D_r(r_c)} < \theta \tag{5}$$

Equation (4) models the situation when the robot has observed the current region $r_c$, but couldn't find any target in it, and equation (5) models the situation when there are more than enough robots in the current region $r_c$. This is signified by the ratio in Equation (5) being less than a prespecified threshold $\theta$. If the situation falls under one of the above criteria, the robot is available and decides to move to another region.

Another problem is how to choose the most urgent region to be observed. The two density properties of each region are used to choose one. The following equations show how these properties are used:

$$D_r(r_i) = 0 \wedge D_t(r_i) > 0 \tag{6}$$

$$\frac{D_t(r_i)}{D_r(r_i)} \geq 1.0 \tag{7}$$

$$D_r(r_i) = 0 \wedge D_t(r_i) = 0 \tag{8}$$

Equation (6) means that a region $r_i$ has moving objects which are not being observed. Equation (7) means that a region has too many objects to be tracked by the current number of robots, and Equation (8) means that a region is not being observed currently. These rules are prioritized; Equation (6) has the highest priority, and Equation (8) has the lowest one. A region for which a higher priority rule is applicable must be observed first. If there are two or more regions with the same score, the region closest to the current robot position is selected to be observed.

### *2.3 System Architecture*

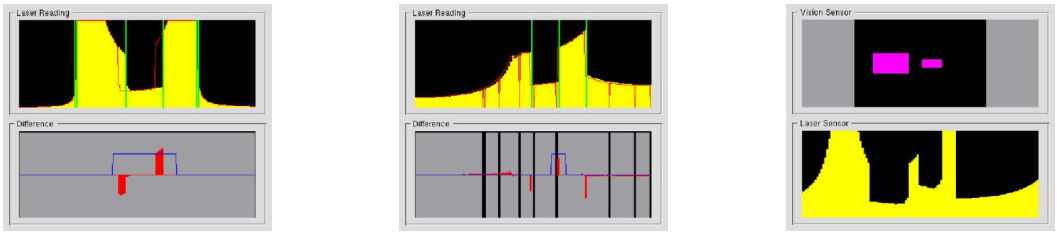
Figure 2 shows a behavior-based control architecture for the mobile robots which uses the density estimate for role assignment. There are five modules in the controller: one for detecting moving targets and four for dispersing robots according to the criteria discussed in the previous section. The embedded sensors have exactly the same system architecture as the mobile robots, but only one module, *Seek-Targets*, is activated.

#### 2.3.1 Seek-Targets

*Seek-Targets* detects moving objects and broadcasts their estimated positions. As shown in Figure 2, two trackers have been developed: a laser-based tracker and a vision-based tracker. Target tracking is a well studied problem, especially in computer vision [8, 9, 10]. Our trackers are simple by design since our focus is on robot role-assignment.

The laser-based tracker uses the SICK laser rangefinder. It reads the laser rangefinder at 10 Hz and analyzes the data to *find moving objects* using scan differencing between consecutive laser readings. A big difference is attributed to a moving object. For accurate tracking, a simple edge detection algorithm is used. Figure 3 (a) and (b) shows two example actual laser readings. The upper window shows two consecutive laser readings and edges, and the lower window shows the difference between the two readings and a detected moving object.

The idea can be implemented without any limitation for stationary embedded sensors, but several limitations exist for mobile robots carrying laser rangefinders. In the mobile robots' case, simply comparing two consecutive laser readings is not correct because the robot actually moves during the scan process. Figure 4 shows two different positions of a robot when the laser was used. In order to compare these scans correctly, the old reading must be transformed to the new coordinate system. However, during the transformation, there may be several parts of the scan that have no valid data because of rounding errors or two exceptions. The first exception is when the old reading contains the maximum value, which means there is an empty region in front of it. The second exception is a corner occlusion. When there is a corner, the old scan does not have any information behind a corner, but the new scan may have (the fan-shaped region in Figure 4). Therefore, these areas must be ignored. The lower window in Figure 3 (b) shows these ignored regions; only the gray region is compared to




(a) Laser (Sensors) (b) Laser (Robots) (c) Vision

Figure 3: Moving-object tracker: various sensory readings

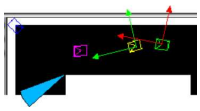


Figure 4: Coordinate transformation

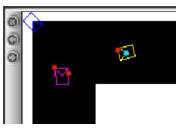


Figure 5: Grouping

calculate a difference. For example, in Figure 3 (a), there is no ignored region because the embedded sensors never move, but there are several ignored regions in Figure 3 (b).

The vision-based tracker uses a camera and a laser rangefinder. A color-blob detector was used to simplify the vision problem. It finds the existence and direction of colored objects using a camera, and measures the distance to objects using a laser rangefinder. Figure 3 (c) shows corresponding camera and laser readings taken from a single robot.

When moving objects are detected, the *Seek-Targets* behavior broadcasts their estimated positions over the network.

#### 2.3.2 Update-Map

*Update-Map* maintains an internal map. It reads broadcast packets about target locations, and puts them in a queue. By counting the packets in the queue, it can estimate the number of robots and the number of targets in each region. However, before counting them, a proper grouping strategy is required. Figure 5 shows a situation that requires grouping; A stationary sensor and a robot both detect a moving target. The mobile robot broadcasts the position of the target, and the embedded sensor does the same, but the position estimates are different. These two estimated positions of the moving object must be grouped as one target. In addition, the embedded sensor would recognize the robot as a moving object because it cannot distinguish a robot from moving objects. This estimated position must be grouped with the robot's position, and removed from the target list.

The robot density and the target density of each region are updated using Equations (2) and (3). The range of robot density is from 0.0 to 1.0, and the range of target density is from -1.0 to 1.0. According to Equation (3), target density cannot be negative. *Update-Map* uses the negative range of target density in order to mark empty regions. Whenever a robot cannot find any moving objects, it sets the target density of the current region to -1.0, which means that the region does not have any moving objects. By using the negative range, a robot can distinguish a region that does not have any moving object from a region that has not been observed. If target density is negative, *Update-Map* increases it slowly over time to 0.0 because the environment is dynamic. When target density becomes 0.0, it means the system has forgotten that there was no target in the region; the robots may now try to observe the region again if needed.

#### 2.3.3 Avoid-Obstacles

*Avoid-Obstacles* allows a robot to navigate without collision. It uses the eight front sonars to detect an obstacle. Each sonar uses a different range to detect obstacles, and constructs a virtual oval-shaped region in front of the robot. When any obstacle enters the region, *Avoid-Obstacles* reduces the speed in inverse-proportion to the distance to the obstacle, and turns away from the obstacle. In addition, *Avoid-Obstacles* stops a robot in place when a moving object approaches it, instead of actively avoiding it.

#### 2.3.4 Move-To-Region

*Move-To-Region* disperses robots all over the environment. The algorithm for it is divided into three steps: checking robot availability, finding the most urgent region, and moving to the region. First, this behavior checks if a robot itself is free to move to another region. Equations (4) and (5) are the criteria to decide if a robot is available for observing other

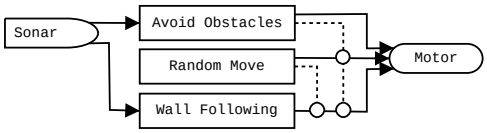



Figure 6: System architecture for target simulation

regions. If available, the behavior finds a region to be observed urgently on the basis of the internal map. It simply examines the internal map, and finds one using the prioritized scoring policy (Equations (6), (7), and (8)). If there are two or more regions that have the same score, the closer region is selected as the most urgent region. Once a starting region and a goal region are decided, a simple graph search is performed to find the shortest path. (The internal maps consist of nodes (landmarks) and regions as shown in Figure 1(d).) A robot follows the shortest path to move to the goal region.

#### 2.3.5 Follow-Targets

The *Follow-Targets* behavior causes robots to follow detected targets. In order to make robots follow more than one target at the same time, *Follow-Targets* calculates the center of mass of detected targets and follows this point, not the targets themselves. The worst case is when two targets move in opposite directions. This does not happen often in our narrow corridor environment.

## 3 Experimental Results

To test our region-based cooperative target tracking approach, several experiments have been performed using a multiple robot simulator.

### *3.1 Stage and Player*

Player [11] is a server and protocol that connects robots, sensors and control programs across the network. Stage [12] simulates a population of Player devices, allowing off-line development of control algorithms. Player and Stage were developed at the USC Robotics Research Labs and are freely available under the GNU Public License from *http://robotics.usc.edu/player/*.

### *3.2 Target Simulation*

Because Stage supports only mobile robots, moving targets in the environment were simulated using robots. The target movements are intended to crudely simulate human movements in an office environment, especially in corridors, like wall-following, turning, staying in place with other targets, etc. Figure 6 shows the control architecture of moving targets.

*Wall-Following* uses two pairs of side sonars. Target motion is divided into two parts: speed control and direction control. *Wall-Following* sets the speed to a maximum value, and uses a proportional controller to align the target parallel to a wall using the front and rear sonars. *Random* was added to make targets' movements somewhat unpredictable. Currently, only one random move is being generated, turning around. However, due to interactions with other targets and robots, each target's moves are quite complicated and unpredictable. *Avoid-Obstacles* is the same module used in the robot controller. The only difference is that a target never stops in place; it always actively avoids obstacles.

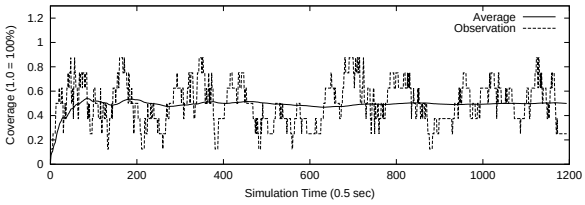



Figure 7: Convergence of the average value

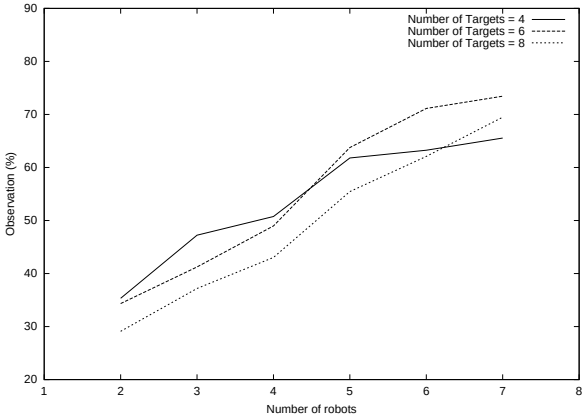



Figure 8: Performance of the region-based method

### *3.3 Experimental Results*

The simulation experiments were done with various configurations in order to evaluate the region-based approach. A performance metric for the CMOMMT task was proposed in [1, 2], and the metric (Equation 1) is used to evaluate performance. Each trial ran for 10 minutes. Figure 7 shows the average observation rate over time which stabilizes after 6–7 minutes.

The difference between the actual position of a target and its position as reported by the sensors was small. The average error was approximately 4 cm.

#### 3.3.1 Performance Evaluation

The performance of the system varies according to the number of sensors, and the number of moving objects. In our experiments, a total of 18 different configurations were tested. We changed the number

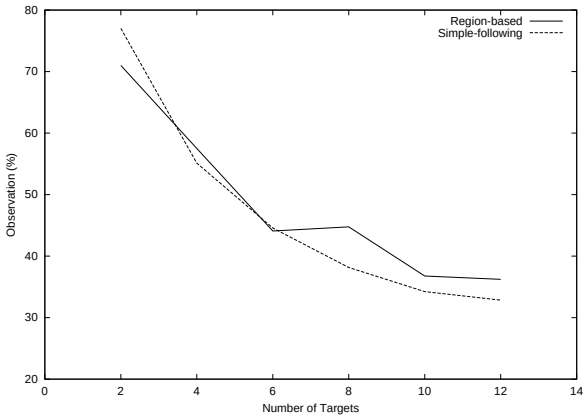



Figure 9: Comparison to a simple following method

of sensors from 2 to 6, and the number of targets from 4 to 8 in steps of 2. Figure 8 shows the tracking results. As expected, the more the sensors, the better the tracking performance. One interesting fact observed through the experiments is that the performance improves whenever sensors are added, but this improvement tails off when the number of sensors is greater than the number of objects.

#### 3.3.2 Comparison to a simple strategy

The region-based method was compared to a simple target-following method. In order to implement the simple method, we inhibited the *Move-To-Region* module. The robots follow walls, but after finding moving targets, the robots follow their center of mass. We changed the number of moving targets from 2 to 12 in steps of 2, and four mobile robots were used for all cases. Figure 9 shows the results. When the number of objects is small, the simple method occasionally showed better performance because robots do not give up following objects to explore other regions that may be more urgent. However, as the number of targets is increased, the region-based method showed better performance because *Move-To-Region* causes robots to move to regions that have more objects.

## 4 Conclusion and Future Work

Autonomous target tracking systems have many real–world applications. We have presented a region-based tracking system, which is especially well suited to structured environments. The system utilizes embedded sensors, like surveillance cameras already installed in buildings. Initial experiments indicate that our approach shows better performance when there are many targets to be tracked.

As future work, research on the ratio of mobile robots to embedded sensors is a topic we plan to address. In addition, real robot experiments are planned for the near future. Because Player provides exactly the same interface for a real Pioneer robot as a virtual robot in Stage, the control programs written for simulation can be used for real robot experiments without major modification.

### References

[1] Lynne E. Parker, "Cooperative motion control for multi-target observation," in *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1997, pp. 1591–1598.

[2] Lynne E. Parker, "Cooperative robotics for multi-target observation," *Intelligent Automation and Soft Computing, special issue on Robotics Research at Oak Ridge National Laboratory*, vol. 5, no. 1, pp. 5–19, 1999.

[3] Barry B. Werger and Maja J. Mataric, "Broadcast of local eligibility for multi-target observation," in *Proceedings of Distributed Autonomous Robotic Systems*, 2000.

[4] Yaakov Bar-Shalom and William Dale Blair, Eds., *Multitarget-Multisensor Tracking: Applications and Advances*, vol. 3, Artech House, 2000.

[5] Goksel Dedeoglu, Maja J. Mataric, and Gaurav S. Sukhatme, "Incremental, on-line topological map building with a mobile robot," in *Proceedings of Mobile Robots*, Boston, MA, 1999, vol. XIV, pp. 129–139.

[6] Goksel Dedeoglu and Gaurav S. Sukhatme, "Landmark-based matching algorithm for cooperative mapping by autonomous robots," in *Distributed Autonomous Robotic Systems (DARS)*, Knoxville, Tennessee, 2000.

[7] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Machine Learning and Autonomous Robots (joint issue)*, vol. 31 & 5, pp. 29–53 & 253–271, 1998.

[8] Isaac Cohen and Gerard Medioni, "Detecting and tracking objects in video surveillance," in *Proceeding of the IEEE Computer Vision and Pattern Recognition 99*, Fort Collins, June 1999.

[9] Stephen S. Intille, James W. Davis, and Aaron F. Bobick, "Real-time closed-world tracking," in *Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition*, June 1997, pp. 928–934.

[10] Alan J. Lipton, Hironobu Fujiyoshi, and Raju S. Patil, "Moving target classification and tracking from real-time video," in *Proceeding of the IEEE Workshop on Applications of Computer Vision*, 1998.

[11] Brian Gerkey, Kasper Stoy, and Richard T. Vaughan, "Player robot server," Institute for Robotics and Intelligent Systems Technical Report IRIS-00-391, University of Southern California, 2000.

[12] Richard T. Vaughan, "Stage: A multiple robot simulator," Institute for Robotics and Intelligent Systems Technical Report IRIS-00-393, University of Southern California, 2000.

# Development Environments for Autonomous Mobile Robots: A Survey

James Kramer and Matthias Scheutz
Artificial Intelligence and Robotics Laboratory
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
Email: {jkramer3,mscheutz}@nd.edu

**Abstract**

*Robotic Development Environments* (RDEs) have come to play an increasingly important role in robotics research in general, and for the development of architectures for mobile robots in particular. Yet, no systematic evaluation of available RDEs has been performed; establishing a comprehensive list of evaluation criteria targeted at robotics applications is desirable that can subsequently be used to compare their strengths and weaknesses. Moreover, there are no practical evaluations of the usability and impact of a large selection of RDEs that provides researchers with the information necessary to select an RDE most suited to their needs, nor identifies trends in RDE research that suggest directions for future RDE development.

This survey addresses the above by selecting and describing nine open source, freely available RDEs for mobile robots, evaluating and comparing them from various points of view. First, based on previous work concerning agent systems, a conceptual framework of four broad categories is established, encompassing the characteristics and capabilities that an RDE supports. Then, a practical evaluation of RDE *usability* in designing, implementing, and executing robot architectures is presented. Finally, the *impact* of specific RDEs on the field of robotics is addressed by providing a list of published applications and research projects that give concrete examples of areas in which systems have been used. The comprehensive evaluation and comparison of the nine RDEs concludes with suggestions of how to use the results of this survey and a brief discussion of future trends in RDE design.

## 1 Introduction

Robots, unlike many software agents, operate under real-world, real-time constraints where sensors and effectors with specific physical characteristics need to be controlled. To facilitate research in autonomous robotics and help architecture designers in managing the complexity of embodied agents, several robot development environments (RDEs) have been developed that support various aspects of the agent development process, ranging from the design of an agent architecture, to its implementation on robot hardware, to executing it on the robot.

While several frameworks for comparing agent systems have been proposed, some of them specifically for RDEs (see Section 3), there is currently no survey available that (1) provides a set of conceptual RDE*features* comprehensive enough to serve as a basis for a conceptual evaluation that does justice to the specific aims with which most RDEs have been developed, (2) applies the conceptual criteria *systematically* to a large selection of RDEs, (3) augments the theoretical evaluation with a practical *usability* evaluation that includes architecture design, implementation, and execution within each RDE on a robot, with special emphasis on ease of use and performance, (4) includes the *impact* of the RDE in terms of categorized published work using it, an indicator of an RDE's prevalence in and influence on the robotics field, and (5) provides a principled way of combining the three evaluations (conceptual, practical, and impact) to obtain an overall measure of how well an RDE can be adapted to the particular needs of researchers choosing among available systems or RDE developers considering future directions of system development.

This paper addresses all five points. Starting with a set of constraints used for the selection of RDEs to be examined (including a rationale for excluding certain RDEs), Section 2 introduces nine general purpose, freely available RDEs. Section 3 reviews previous work concerning agent system and RDE comparisons, establishing four categories of criteria corresponding to typical stages of application development for autonomous mobile robots. The RDEs are

then systematically evaluated according to the criteria in Section 4. Section 5 contains a practical evaluation based on designing, implementing, and running a simple architecture and some more complex architectural components in each RDE. The subsequent discussion in Section 6 ties together the conceptual and practical evaluations and augments them with one possible evaluation of the impact of each RDE, also suggesting a principled method for using the results of this survey by both researchers and RDE developers. Section 7 summarizes the results and extrapolates to identify some future trends in RDE development.

# 2 Autonomous Mobile Robot Systems

A complete accounting and systematic comparison of all RDEs is clearly impossible within the confines of a survey paper, not only because of the number of RDEs available and the release of new systems, but also due to the scope of robotics as a discipline. To make the task manageable, a group of qualifying constraints is used to limit the selection to a specific subset of representative RDEs. First, we consider only open source packages unencumbered by licensing costs and available for free download. CyberBotics *Webots* (Michel, 2004; Webots, 2005), *White Box Robotics* (WhiteBoxRobotics, 2005), and Evolution Robotics' *ERSP* (ERSP, 2004) are excluded as commercial packages. Also excluded are BERRA (Lindstrom et al., 2000) and CLARAty (Volpe et al., 2001; Nesnas et al., 2003, 2006) due to download unavailability. Systems are also required to generalize beyond specific hardware platforms, but provide more specificity than a general framework. So, while Lego *Mindstorms* (LEGO, 2005) is a widely-used robotics platform with many related packages available, we do not consider it (or projects such as *CotsBots* (Bergbreiter & Pister, 2003; CotsBots, 2005) or *Modular Controller Architecture* (MCA2, 2005)) due to specificity in relation to a single platform or custom hardware construction. Conversely, LAAS's GenoM (Fleury et al., 1997; Mallet et al., 2002; GenoM, 2004) is excluded as a framework for the generic definition of robot components. Finally, there must be at least one cohesive application developed in the system (i.e., a repository of components is not considered for inclusion). To our knowledge, this requirement is not met by *Orocos* (Bruyninckx, 2001; OROCOS, 2005), *The Rossum Project* (Rossum, 2004), *Nomadic* (Sprouse, 2005), *Dave's Robotic Operating System* (Austin, 2004), the *Open Automation Project* (Walters, 2003), or *YARP* (Metta et al., 2006). Similarly, this excludes some projects that, at the time this research was begun, were either just being developed (e.g., Orca (A. Brooks et al., 2005; Orca, 2005)) or in a pre-release stage (e.g., the RObotics and Learning Environment (ROLE) (ROLE, 2005)).[1]

Given the above constraints, nine RDEs have been selected[2], listed in Table 1. The following synopses give an overview of the systems' use and operation, including a broad system description, the stated purpose of the system, the platforms on which it runs, the release version, and a summary of notable features. To characterize the strengths of the systems more completely, the end of each subsection lists publications from particular robotics research subareas, determined by the presentation groupings established in the 2001-2005 *Proceedings of the IEEE International Conference on Robotics and Automation* (ICRA), which have been divided into three categories:

- *Single robot*: SLAM, Planning/Navigation, Learning, Hierarchical Behavior, and Education
- *Human-Robot Interaction*: Task Allocation, Learning, and Assistive Robotics
- *Multi-robot*: Sensing, Exploration, Mapping, Localization, Planning, Coordination, Formation, and Task Allocation

Only a single publication represents a sub-area; citations are also used in evaluating an RDE's *impact* in Section 6.

## 2.1 TeamBots

TeamBots (Teambots, 2004; Balch & Ram, 1998) (which supersedes JavaBots) is a Java-based collection of application programs and Java packages for multi-agent mobile robotics research. Although it is no longer under active development (the most recent version available, 2.0e, was released in April 2000), it is included due to its appearance in both (Jia et al., 2004) and (Orebäck & Christensen, 2003) and because it has found wide use in both research and education. The main author of TeamBots is now affiliated with the laboratory that develops MissionLab (see Section 2.6); for the above reasons, this description will be brief.

[1] Exclusion of the listed systems is only indicative of not meeting the specified constraints; further examination is encouraged.

[2] Almost all of the selected RDEs are under constant revision and more recent versions might be available.

| RDE | Originating Institution | More Information |
|---|---|---|
| TeamBots, v.2.0e | Carnegie-Mellon University | http://www.teambots.org/ |
| ARIA, v.2.4.1 | MobileRobots, Inc. | http://robots.mobilerobots.com/ |
| Player/Stage, v.1.6.5, 1.6.2 | University of Southern California | http://playerstage.sourceforge.net/ |
| Pyro, v.4.6.0 | Bryn Mawr College<br>Swarthmore College<br>University of Massachusetts<br>SRI International | http://emergent.brynmawr.edu/pyro/?page=Pyro |
| CARMEN, v.1.1.1 | Carnegie-Mellon University | http://carmen.sourceforge.net/ |
| MissionLab, v.6.0 | Georgia Institute of Technology | http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab/ |
| ADE, v.1.0beta | University of Notre Dame | http://ade.sourceforge.net/ |
| Miro, v.CVS-March 17, 2006 | University of Ulm | http://smart.informatik.uni-ulm.de/MIRO/ |
| MARIE, v.0.4.0<br>FlowDesigner, v.0.9.0<br>RobotFlow, v.0.2.6 | Université de Sherbrooke | http://marie.sourceforge.net/<br>http://flowdesigner.sourceforge.net/<br>http://robotflow.sourceforge.net/ |

Table 1: RDEs Selected for this Survey

A highly touted feature of TeamBots, stemming from a strict separation of hardware interfaces and control code, is the use of the same control code both in simulation and for an actual robot. While the only robot platforms supported are Probotic's Cye and Nomad 150 robots, there are many example simulation environments and control systems available. The simulator was developed to be extremely flexible, supporting multiple, heterogeneous robot platforms and control systems simultaneously. In addition, the *Clay* package allows hierarchical behavior specification, specifically targeting schema-based control. Inter-robot communication is supported via sockets and serial ports only. A notable inclusion is the Java CMU-Vision package, which supports frame captures and blob-tracking.

TeamBots publications include those from the hierarchical behavior (Balch, 2000) and education (Balch, 2002) subareas.

## 2.2 Advanced Robotics Interface for Applications (ARIA)

*ARIA* (MobileRobots, Inc., 2005; LaFary & Newton, 2005), the base software that comes packaged with the purchase of MobileRobots (*neé* ActivMedia) robots, is a set of C++ classes available for free download. At the lowest level, ARIA provides *system architecture* capacities; that is, software that describes the structure of a robot (including its sensors, effectors, and physical specifications) and implements the low-level interaction between software and hardware components. At a higher level of abstraction, it also includes some sensory interpretation functionality, basic actions (analogous to behaviors), and an elementary action resolver.

Although freely available, ARIA is a product of MobileRobots, Inc. and thus only supports MobileRobots platforms, using *robot parameter files* as the means of defining the characteristics of a robot. This includes information about the robot body (e.g., the robot's radius), the sensors (e.g., the number and position of sonar), and the effectors (e.g., the maximum velocity). The parameters are used by ARIA for various calculations (e.g., the "RobotRadius" parameter is used to determine the robot's turn limits). In support of distributed computing, ARIA provides the *ArNetworking* package as a wrapper around socket communications. In addition, the Simplified Wrapper and Interface Generator (SWIG, 2004) development tool is used to provide Java and Python support.

Supporting software is available, but is limited in some cases by licensing or purchase requirements. The *MobileSim* 2-dimensional simulator, a modified version of Player/Stage's Stage simulator (see Section 2.3, is freely available. A demo version of the ActivMedia Color Tracking Software (ACTS) is available for free download, but has restricted functionality that disallows integration with ARIA (not true of the licensed version). Additional open source software includes the *ArSpeech* components that provide interfaces to Sphinx speech recognition and both Festival and Cepstral speech production packages, *SonARNL* for sonar-based localization, *Mapper3 Basic* for map creation and editing, and *VisLib* for single camera object tracking. Also available, but restricted to license and/or purchase are *MobileEyes* (which provides a remote robot display and control GUI), *ARNL* (which provides laser-based mapping

and localization), and *Mapper3* (which augments the basic mapper package with laser support and automated map creation from sensor logs).

There are no publications available from projects that have used ARIA[3].

## 2.3 Player/Stage

The Player/Stage (Gerkey et al., 2001, 2003, 2005) project is designed to be a *programming interface*, specifically avoiding being a development environment. Rather than treating a robot as the primary unit of agency, it instead focusses on *devices*, or the various sensors and effectors. A "collection" of devices is typically, although not necessarily, located on a single robot. Supported platforms include MobileRobots, RWI/iRobot's RFLEX-based, Segway, Acronname, Botrics, Evolution Robotics, and K-Team robots, while components that are packaged along with the download include vector field histogram goal-seeking/obstacle avoidance, adaptive Monte-Carlo localization, and a wavefront propagation path planner and interfaces for ACTS (see Section 2.2), CMVision, Festival, a service discovery mechanism, and others.

In this software, *Player* refers specifically to the device and server interface. Devices are independent of one another and "register" with a Player *server* to become accessible to *clients*. Each client uses a separate socket connection to a server for data transfer, thus preserving concurrent operation of devices and the servicing of multiple requests. Minimal constraints are placed on the use of devices; in a very real sense, usage is only a *communication protocol*, leaving the client the freedom (and by extension, the work) of designing and implementing a control architecture. *Stage*, which is the second part of the software, is a device simulator. Client control code uses the same programming interface when used in conjunction with either simulated or physical devices.

A stated design goal of the device model used in Player/Stage is the separation of *interface* and *function*. The fact that servers communicate via a socket interface means that client programs can be written using any language with socket support. According to (Gerkey et al., 2003), currently available libraries include those written in C, C++, Tcl, Python, Java, and Common LISP. Due to the prevalent use of socket communication, the Player system inherently fits the distributed computing paradigm. Client code is able to operate on any host that has network connectivity, enabling location independence. A side effect of the device model and its networked basis is that combinations of devices can be formed to create novel types of agents (e.g., one composed of only sonar devices from many different robots). An additional feature of the device model is that the frequency of sensor and effector updates are independent, providing clients the ability to make full use of the data generated by devices that operate at a high frequency, while not being hindered by those that are slower.

*Stage* is a graphical, 2-dimensional simulator that models devices in a user defined environment. Specifically designed to support research in multi-robot systems through its use of socket-based communication, it also forms the foundation for ARIA's MobileSim simulator. In addition to Stage, a high-fidelity, 3-dimensional simulator called *Gazebo* is available. In both cases, client code uses the same interface on real robots as in the simulator. The authors mention that the device model makes it easy to simulate non-existent devices (for instance, a type of sonar that penetrates walls to some extent) for further research in device design and use.

Player/Stage publications include those from the SLAM (Wolf & Sukhatme, 2005), learning (Provost et al., 2004), education (Matarić, 2004), HRI task allocation (Tews et al., 2003), multi-robot sensing (Jung & Sukhatme, 2002), multi-robot exploration (Howard et al., 2002), multi-robot mapping (Howard et al., 2004), multi-robot localization (Howard et al., 2003), multi-robot planning (Howard et al., 2004), multi-robot coordination (Jones & Matarić, 2004), multi-robot formation (Fredslund & Matarić, 2002), and multi-robot task allocation (Gerkey & Matarić, 2002).

## 2.4 Python Robotics (Pyro)

Pyro (Blank et al., 2003, To appear; Pyro, 2005) is a robot programming environment aimed at, but not limited to, educational purposes, leading to specific choices in its design. One goal is to provide a top-down approach to the design of controllers, insulating students from low-level details of implementation while preserving access to the low-level if it is desired. Some of the abstractions include: range sensors, robot units, sensor groups, motion control, and devices, which encapsulate lower levels. This includes "wrapping" Player/Stage (see Section 2.3) and ARIA (see Section 2.2) functionality, so that any component written for those systems are also available to Pyro users. A large selection of platforms are supported, including K-Team Kheperas and Hemission, MobileRobots Pioneer, Handyboard, Sony Aibo, and all robots supported by Player/Stage (see Section 2.3).

[3]While publications exist for ARIA's ancestral software, the authors were explicitly requested to not refer to it.

Educational modules exist to demonstrate control paradigms (e.g., neural networks, evolutionary algorithms, vision processing, and reactive, behavior-based, finite state machines, etc.). Python, an interpreted language, was chosen as the basis of the system due to its ease of use for beginning students, while permitting more knowledgeable designers to write more advanced code. While it is acknowledged that using an interpreted language leads to slower operation, the trade-off between usability and performance is consciously made. Construction of graphical visualization of robot operation are explicitly supported through the use of pre-defined facilities and Python's OpenGL interface. Another goal is to design control code that operates on many different robots with no modification. An example of this is the use of "robot units" that replace traditional measurements such as meters.

Pyro publications include those from the learning (Blank et al., 2002), education (Blank et al., To appear), and HRI task allocation (Desai & Yanco, 2005) subareas.

## 2.5 Carnegie Mellon Robot Navigation Toolkit (CARMEN)

CARMEN (Montemerlo et al., 2003b, 2003a) is an open source collection of (mobile) robot control software written in the C programming language that is meant to provide a "consistent interface and a basic set of primitives for robotic research". Oriented towards single robot control, it uses a three layer agent "architecture", in which the first layer is the hardware interface, providing low-level control and integration of sensor and motion data, the second layer is concerned with basic robot tasks such as navigation, localization, object tracking, and motion planning, and the third layer is the user-defined application, which relies on the primitives of lower layers. Modularity is a primary concern, supported by the Inter-Process Communication System (IPC) communication protocol/software (discussed in more detail below). Besides supporting a number of robot platforms (including MobileRobots, Nomadic Technologies Scout and XR4000, Segway, iRobot ATRV, ATRVjr, and B21R) and navigation primitives (map-making, Monte-Carlo particle filter localization (Thrun et al., 2000), and Markov decision process path planning (Konolige, 2000)), CARMEN also provides configuration tools, a simulator, and graphical displays and editors.

IPC (Simmons, 1994, 2004) provides high-level support for connecting processes using TCP/IP sockets and sending data between processes, including opening and closing sockets, registering, sending, and receiving messages, which may be anonymous publish/subscribe or client/server type communications. The IPC library contains functions to marshall (serialize) and unmarshall (de-serialize) data, handles data transfer between machines with different Endian conventions, invoke user-defined handlers when a message is received, and invoke user-defined callbacks at set intervals. In essence, IPC performs a function similar to a naming service for components; besides providing the means to define message abstractions used for communication over a network, it also encourages extensibility (in that components are self-contained) and fault-tolerance (in that failure of a component ceases communication, but does not actively interrupt other components in the system).

CARMEN components generally take the form of a single executable, such as **pioneer** (for a MobileRobots Pioneer robot), **laser** (for a SICK laser range finder), or **localize** (for robot localization using a pre-made map). Particular platform definitions are contained in "base" specifications, which are then abstracted to a generic "robot" configuration that includes basic parameters such as body length and width, sonar offsets, maximum velocities, etc. Parameters for each component are stored in a human readable text file repository, but a graphical editor can be used to modify parameters at run-time. In addition, each component relies on a set of IPC message definitions to which other components can subscribe, allowing component distribution through an IPC server.

CARMEN publications include those from the SLAM (Thrun et al., 2000), learning (Osentoski et al., 2004), HRI assistive robotics (Pineau et al., 2002), and multi-robot coordination (Simmons et al., 2000) subareas.

## 2.6 MissionLab

MissionLab (MacKenzie et al., 1997; MissionLab, 2003) is a set of software tools for executing military-style plans using individual or teams of real and simulated robots. Developed as part of the DARPA Mobile Autonomous Robot Software (MARS) project, the main stated goal is to control the motion of robots in highly dynamic, unpredictable, and possibly hostile environments. Collaboration and coordination of robot teams is based on the *Societal Agent* theory, which views abstract "assemblages" of agents as agents themselves and whose behavior, in turn, is the aggregate of coordinated "primitive" behaviors of "atomic" agents. Assemblages are hierarchical, while behavior coordination is achieved through finite state automata (either competitive or temporally sequenced) or vector summation cooperation.

The *Configuration Description Language* (CDL) is used to recursively define abstract societal agents (called *configurations*), usually accomplished using the graphical *CfgEdit* tool. A configuration can be bound to a specific set of

robots and devices; robot choices include MobileRobots Pioneer, iRobot ATRVjr and Urban, Evolution Robotics ER-1, and Nomad 150/200. CDL is compiled to *Configuration Network Language* (CNL) code, which is then compiled to C++ code and finally compiled to machine code, resulting in a robot executable. The executable contains a communication module (called *HClient*) to interface with an *HServer*, an abstract control interface used for all robot hardware via IPT communication software. (IPT supports distributed computing and is related to the IPC communication software, described in Section 2.5; both are derived from the *Task Control Architecture* (TCA, Simmons, 1994) project.) Developers also have the option of using the higher level *Command Description Language* (CMDL) to describe robot missions, which is a mechanism for providing high-level input to robot behaviors.

As a primary concern of MissionLab is usability, the graphical interface is quite extensive, allowing non-experts to write control code without any programming. Logging consists of writing a robot's position, velocity, heading, and the current state of the robot with respect to time to a disk file, while debugging toggles are used to display program output to a console. A unique feature relative to other systems is the inclusion of "Motivational Variables" (anger, fear, hunger, curiosity) that simulate emotionality. Developers can also assign user-defined "Personalities" to robots. Finally, there is an extensive set of components available with Missionlab, including a case based reasoner, Q-learning, graphical behavior building tool, D* Lite planner, and human/robot interaction interfaces.

MissionLab publications include those from the learning (Arkin et al., 2003), hierarchical behavior (MacKenzie & Arkin, 1993), HRI task allocation (MacKenzie & Arkin, 1998), multi-robot planning (Endo et al., 2004), multi-robot coordination (MacKenzie et al., 1997), multi-robot formation (Balch & Arkin, 1999), and multi-robot task allocation (Arkin et al., 1999) subareas.

## 2.7 APOC Development Environment (ADE)

ADE (Andronache & Scheutz, 2004a, 2004b; Scheutz, 2006) is a programming environment that combines (1) support for developing and implementing agent architectures with (2) the infrastructure necessary for distributing architectural components. An explicit goal is to combine features of multi-agent systems (by treating architectural components as "agents" in a MAS-sense) with those of a programming environment and toolkit for complex agent design and implementation. ADE is a Java implementation of the APOC (Activating, Processing, Observing Components) (Scheutz & Andronache, 2003; Scheutz, 2004) universal agent architecture framework, which provides arbitrary levels of (possibly hierarchical) component abstraction and interconnection. Communication among ADE components relies on Java's Remote Method Invocation (RMI) facilities. ADE provides infrastructural components for an enhanced naming service, connection mediation and monitoring, security features (access control and authentication), and the ability to store the run-time state of the system, which in turn allows for the detection and recovery from component failures.

While ADE is limited to MobileRobots robots and Arrick Robotics' Trilobot, a set of abstractions for typical robotic sensors and effectors provide the means for extending support to other platforms. Configuration files can take the form of either text or XML files and include both an abstract architecture description and/or the run-time specification of component distribution. Graphical representations of individual components exist, accessible via a distributed, multi-user GUI, which provides a view of the complete agent architecture and the means to control individual components. Logging facilities allow any component in an ADE system to write to multiple files. ADE provides several predefined components include components for behavior definition, vision processing, speech recognition and production, a general-purpose rule interpreter, a Prolog interface, and "wrappers" to incorporate external software. It also includes a Java implementation of a Player (see Section 2.3) client that interfaces with the Stage 2-dimensional robot simulator and other Player/Stage components, in addition to an interface to the simulator packaged with the now defunct Saphira (Konolige et al., 1997; Konolige, 2002) system.

ADE publications include those from the planning/navigation (Kramer & Scheutz, 2003), hierarchical behavior (Scheutz & Andronache, 2004), HRI task allocation (Scheutz et al., 2004), assistive robotics (Scheutz et al., 2006), multi-robot sensing (Andronache & Scheutz, 2004a), and multi-robot coordination (Scheutz, 2006).

## 2.8 Middleware for Robots (Miro)

Miro (Utz et al., 2002; Miro, 2005) is a distributed, object oriented framework for mobile robot control that is meant to facilitate heterogeneous software integration and foster portability and maintainability of robot software. Core components have been developed in C++ for Linux based on Common Object Request Broker Architecture (CORBA) technology using the adaptive communication environment (ACE, Schmidt, 1994) as its communication framework.

Due to the programming language independence of CORBA, further components can be written in any language and on any platform that provides CORBA implementations.

Miro currently supports three platforms: iRobot B21, MobileRobots Pioneer, and the custom-built Sparrow. Abstraction interfaces include odometry, motion, rangesensor (sonar, infrared, bumper, laser), stall, video, pantilt, GUI buttons, and speech. Components exchange data based on subscriptions, which allow for event driven notification. Defined messages include those for odometry, rangesensor (scanevent, groupevent, bunchevent), sonar, infrared, bumper, stall, and GUI buttons. Miro includes a "behavior engine" for reactive behavior specification, which allows hierarchical decomposition of timed, event and task behavior sets into "policies". There are two types of policy transitions, local and global, that can be edited via a graphical interface; global policies preempt behaviors, local do not. The configuration of hardware, data subscriptions, and logging specification is stored in XML files. Two types of logging are defined, "log levels" and "log categories", that allow developers to vary the granularity of log data, while a graphical LogPlayer allows the replay of logged data.

Miro publications include those from the SLAM (Kraetzschmar et al., 2004), planning/navigation (Kraetzschmar et al., 2000), learning (Fay et al., 2004), hierarchical behavior (Utz et al., 2005), HRI assistive robotics (Gassull, 2001), multi-robot sensing (Utz et al., 2004), and multi-robot coordination (Utz et al., 2004) subareas.

### 2.9 Mobile and Autonomous Robotics Integration Environment (MARIE)

MARIE (Côté et al., 2004, 2006; Côté, 2005) is a programming environment that is specifically designed with the integration and distribution of robot applications, components, and tools in mind. For brevity, "MARIE" is used throughout this description and the rest of the survey to signify the MARIE software **and** the related FlowDesigner (Valin & Létourneau, 2004) and RobotFlow (Michaud & Létourneau, 2004) packages (described below), unless clarification is necessary. MARIE is implemented in C++ and the integration aspect of MARIE proper uses (but is not dependent upon) the Adaptive Communication Environment (ACE, Schmidt, 1994) communication framework. Following the *mediator* design pattern (Gamma et al., 1994), MARIE provides a centralized component that connects a variety of (possibly different) software. There are four functional components: application adapters, communication adapters, communication managers, and application managers. Application adapters act as proxies between the central component and applications. Communication adapters translate data between application adapters, while communication managers create and manage the links. Finally, application managers coordinate system states and configure and control system components on any one processing node. In keeping with the aim of integrating software, components have been developed for Player/Stage (see Section 2.3), CARMEN (see Section 2.5), and ARIA (see Section 2.2).

FlowDesigner is a data-flow processing library coupled with a graphical display that allows developers to create reusable "software blocks" linked together in a (possibly hierarchical) network. Available libraries include support for signal processing, audio processing (DSP), vector quantization, neural networks, fuzzy logic, an Octave plug-in, and RobotFlow. RobotFlow is a mobile robotics toolkit for FlowDesigner that includes support for MobileRobots Pioneer2 robots and other hardware devices, behaviors, finite state machines, vision processing (color training, tracking, etc.) and the interfaces for use with MARIE.

MARIE publications include those from the planning/navigation (Beaudry et al., 2005), education (Michaud, 2005), HRI assistive robotics (Labonté et al., 2005), multi-robot localization (Rivard, 2005), multi-robot coordination (Guilbert et al., 2003), and multi-robot formation (Lemay et al., 2004) subareas.

## 3 A Conceptual Framework for Comparing RDEs

Several comparisons of agent systems and agent development environments have been proposed in the recent literature. For software agents, they are typically concerned with various aspects of multi-agent systems (MAS), including comparing *agent platforms* (Altmann et al., 2001; Nguyen et al., 2002; Laukkanen, 1999; Nowostawski et al., 2000; Ricordel & Demazeau, 2000), *agent development kits* (Bitting et al., 2003), *mobile agent systems* (Silva et al., 2001), or *agent environments* (Eiter & Mascardi, 2002). There are also comparisons of *general agent systems* and *agent architectures per se* (Sloman, 1998; Logan, 1998; Sloman & Scheutz, 2002). Comparisons that concern robotic agents in particular have addressed *mobile robotic architectures* (Orebäck & Christensen, 2003) and *robot programming environments* (MacDonald et al., 2003; Jia et al., 2004; Biggs & MacDonald, 2003). Common to all is the need to establish an appropriate set of *criteria* that serves as a basis for the comparison. Clearly, the choice of criteria is critical, for, as pointed out in (Ricordel & Demazeau, 2000), "any criteria is relevant to a specific outside need".

We briefly review some of this prior work to situate our proposed evaluation criteria, giving a general overview of the conceptual breakdown in each and why each proves insufficient for the purposes of this paper. To avoid ambiguities and equivocations among the different terms used, we will adhere to the following terminology for the rest of this paper:

- *Platform:* the hardware on which an application will be executing; this includes the sensors, actuators, computers, operating system(s), and other hardware or software intimately tied to hardware.
- *Component:* a functionally independent part of an agent or system.
- *Architecture:* the structure and interaction of components; if necessary, a distinction will be made between system and agent architectures.
- *Agent:* the sum of the software and hardware required for an individual robot to perform its task. In particular, we will not consider infrastructure or strictly software agents (e.g., a *naming service* or *communication agent*) as agents *per se*, as is done in the field of multi-agent systems. These are instead considered functional *components* that are part of the broader environment or application.
- *Programming Environment:* the tools, infrastructure, and components that are not left for implementation by the developer. The term *system* will be used interchangeably in this context.

The most general and, for our purposes, pertinent, framework is (Eiter & Mascardi, 2002). Although founded in MAS research, the classification is intended to be comprehensive, establishing a framework for all types of agent systems. Additionally, the authors provide a practical method for *choosing* an appropriate system for a task selected by an application designer. Criteria are divided into five categories: (1) *agent attitudes*, (2) *software engineering support*, (3) *implementation concerns*, (4) *technical issues*, and (5) *economical aspects*[4]. While the *software engineering*, *implementation*, and *technical issues* categories usually have a prominent role in discussions of RDEs, the *agent attitudes* aspect is often omitted–not because it is unimportant or ignored, but because features therein often form the task, or object, of investigation. Yet, according to Eiter, the attitudes category is comprised of features that discriminate between agent and non-agent software: they are either *basic* (i.e., "close to the very core of agenthood") or *advanced* (i.e., "desirable but not of central interest"). Hence, an agent development environment (and by extension an RDE) should, at least in part, be evaluated with respect to the degree to which it supports these attitudes. While Eiter and Mascardi's categorization is comprehensive, it lacks some details of considerable importance for evaluating RDEs. In fact, this is explicitly acknowledged with the disclaimer, "other features and criteria should be taken into account" for the unique issues that arise in the development of *physical agents* (e.g., support for devices, real-time operation, etc.).

In their framework proposal, Jia et al., 2004 isolate three high-level categories for analysis of an RDE: (1) *openness*, (2) *abstraction*, and (3) *modularity*. *Openness* refers to extensibility: a programming environment should support the addition and evolution of hardware and software. *Abstraction* forms the basis on which *openness* is built, providing a well-defined application programming interface (API) that allows a developer to work at a level beyond the hardware (see also Vaughan et al., 2003). Different from *abstraction*, which is focussed on hardware, *modularity* concerns software, promoting good design and reusability. While these three categories address the design and implementation of autonomous mobile robotic applications (as demonstrated by their in-house development of the *Frontier-1* robot), they are too general to address specific concerns of RDEs (e.g., real-time support, hardware-dependence of a robotic platform, debugging tools, etc.).

MacDonald et al., 2003 give a detailed and comprehensive description of RDE features in three categories: (1) *robot programming* (both at the system and task level, which enable programmers to describe robot behavior), (2) *infrastructure* (which supports the execution of behavior descriptions), and (3) *human-robot interaction* (HRI, which allows interaction with the robot programming area; see also Biggs & MacDonald, 2003). The proposed features will be largely included in our comparison, but there are some issues concerning the analysis, organization, and application to various aspects of RDEs. For one, the boundaries of the categories overlap to such an extent as to be unclear. For instance, *infrastructure* conflates the facilities provided by the environment with both the *programming* and the *agent architecture* categories. Similarly, the broad scope of the *HRI* (human/robot interaction) category largely overlaps the robot programming category, yet contains individual features that are too specific for a general system comparison

[4]Eiter's *economical aspects* category will not be considered here, except for the *documentation* criterion, as the selected RDEs are both open source and research-oriented. Related considerations, such as the cost of application development, RDE maintenance or modification, training, etc. are, however, addressed by the usability evaluation in Section 5.

(i.e., excluding systems that are not especially intended nor designed for HRI). Moreover, the proposed categorization is not structured in a way that is easily amenable to a systematic comparison (e.g., conceptually different items are subsumed under the rubric "robot programming").

The study closest in intent to this survey is Orebäck & Christensen, 2003, which attempts to establish the characteristics of a "common software architecture" for mobile robot systems. In particular, seven categories (*hardware abstraction*, *scalability*, *overhead*, *control model*, *software*, *tools and methods*, and *documentation*) are proposed as a basis for comparing RDEs, covering an extensive range of features. However, while the proposed framework is generally suitable, the actual comparison is limited to only three RDEs (TeamBots (Teambots, 2004), Saphira (Konolige et al., 1997), and BERRA (Lindstrom et al., 2000)) and does not adhere strictly to the conceptual framework. Rather, criteria are grouped into six areas that mostly, but not always, correspond to the categories as defined, in some instances leaving out or introducing new criteria.

While all of the above studies agree that the main purpose of an RDE is to provide appropriate tools and abstractions that help the agent designer, they fail to provide a comprehensive, yet succinct conceptual framework that allows for a systematic comparison of RDEs. Based on the three typical stages in the development process of a robotic agent architecture[5] (*design*, *implementation*, and *execution*), we propose four categories of criteria for RDE comparison, categorized in terms of:

**F1:** *Specification*, which includes formalisms, methodologies, and design tools,

**F2:** *Platform support*, which is related to the hardware and its low-level interface (e.g., the operating system),

**F3:** *Infrastructure*, which refers to components and capabilities that are part of the RDE, but not the "agent architecture proper", and

**F4:** *Implementation*, which includes aspects of application development (including predefined components used in an agent architecture).

Of the four categories, three reflect features relevant to specific development stages (e.g., *specification* features are central to *design*, *implementation* features pertain to implementation, as the name suggests, and *platform* features play a role in the execution). The fourth category, *infrastructure*, is added to explicitly distinguish aspects of an RDE that are separate and distinct from the agent architecture (e.g., distribution mechanisms that are integral to system operation, yet usually transparent to the agent designer).

We note in advance that the four categories are comprised of features that an RDE objectively has or does not have, with an emphasis on the software engineering aspects of its functional characteristics and capabilities. These criteria alone are not sufficient for a full evaluation of an RDE and are supplemented with additional criteria in Sections 5 and 6. The different types of evaluation can be distinguished by an identifying prefix; criteria in this section are denoted by **F**, followed by a category and item number. It is crucial to note that even though the expanded criteria list provides a comprehensive foundation for RDE evaluation, it is impossible in principle to address every concern a developer might have. A remedy for the situation is discussed in Section 6.

## F1: Specification

The specification of a robotic agent or application occurs in the design stage and concerns issues such as the application domain(s), software engineering, and determination of an appropriate agent architecture. To preserve the focus on RDEs, the criteria presented are somewhat broad, but are sufficient to address the prevailing concerns.

**F1.1:** *Architectural Primitives.* An RDE provides various types of predefined functional component and/or knowledge primitives useful in robotic applications (e.g., behaviors, methods of control, tasks, objects, etc.), or the means to create, organize, and manipulate them.

**F1.2:** *Software engineering.* Software engineering support promotes the creation of high-quality software. Enabling modularization and code reuse, it can be accomplished through the use of stated design principles, explicit frameworks or tools, methodologies, or formalisms, and includes application verification, prototyping, and the abstractions mentioned in (Jia et al., 2004), (Orebäck & Christensen, 2003) and (Vaughan et al., 2003).

[5] Our stages are similar to (Ricordel & Demazeau, 2000), although we subsume the *analysis* category as part of the *design stage*.

**F1.3:** *Architecture neutrality.* An RDE may be associated with a particular theoretical foundation that promotes a specific agent/application architecture, separate from implementational concerns. Alternatively, it may be *architecture neutral*, leaving the choice to the designer or even providing the means to compare application implementations using different agent architectures.

## F2: Platform Support

Robotic applications necessarily incorporate real-world sensors and effectors; thus, they require a more diverse set of hardware than software-only systems. The principles of *abstraction*, *modularity*, and *openness*, as put forth in (Jia et al., 2004), are of particular importance to this item, promoting application use across varying platforms.

**F2.1:** *Operating system.* An RDE may be compatible with one or many operating systems, but must be compatible with the designer's choice. This can become a major obstacle when certain libraries or components are implemented only for a particular operating system.

**F2.2:** *Hardware support.* "Hardware support" refers to the variety of sensors and effectors that are available in an RDE, such as cameras, sonar, and laser devices. Since the number of standard (that is, common and non-custom) devices is limited and widely used on different platforms, we will refer instead to particular robot manufacturers. In support of increased modularity, ease of device modification, and addition of custom devices, a hierarchy of device abstraction is often specified, allowing control code to be easily ported and executed on different robots.

**F2.3:** *Simulator.* Simulation of the physical world allows developers to test applications, model currently unavailable hardware, and replay actual application execution. Simulators can be low- or high-fidelity, approximating an environment to some lesser or greater degree, and can also be two- or three-dimensional. Some simulators have the ability to include multiple robots in a single simulation or to mix real and simulated robots in an environment.

**F2.4:** *Configuration method.* The configuration of a robot is often changed to meet the demands of various applications. This information may be incorporated into the source code (requiring compilation to effect changes) or in configuration files that can be easily modified, either with a text editor or a graphical interface.

## F3: Infrastructure

*Infrastructure* refers to RDE functionality that affects multiple components (or the system as a whole) and is not tailored to individual architectural components, application domains, or particular stages of application/agent development. For example, *logging facilities* can be used with any or all components, are often invaluable as debugging tools during the implementation stage, and provide records of an execution instance for later performance analysis. In some cases, however, it may be impossible to determine whether a feature is due to a specific component or part of the infrastructure by function alone. For instance, the graphical representation of components might be implemented on an *ad hoc* basis, removing it from consideration as infrastructure. A system must provide generic mechanisms that supply these capabilities for them to be considered as infrastructure.

**F3.1:** *Low-level communication.* Inter-process communication (such as memory mapping, pipes, or sockets), basic networking protocols (such as UDP, TCP/IP, etc.), and mid-level protocols (such as IIOP or RMI) are part of the system infrastructure. These capabilities are often dictated by the platform being used, as their availability is contingent on the operating system and/or programming language.

**F3.2:** *Logging facilities.* Log files of application operation can be used for debugging, repetition of an application execution, or gathering performance statistics. Logging mechanisms can have various levels of flexibility, including fixed (which generally captures all data produced by components) vs. configurable data content, local vs. remote logging, file name selection, single vs. multiple data streams and/or files, or the ability to start and stop logging at run-time.

**F3.3:** *Debugging facilities.* While logging facilities can suffice for basic debugging, robust debugging tools can be invaluable during application implementation. Such tools can range from low-level code editors, to mid-level graphical representations of sensors and effectors, to high-level graphical behavior or task modification, possibly allowing run-time suspension, modification, and restarting.

**F3.4:** *Distribution mechanisms.* Distribution mechanisms, as part of the infrastructure, are required for multi-host applications. Typically, distribution capabilities are enabled by middleware (e.g., Poggi et al., 2002), either as a generic component implementation framework (such as CORBA, 2005, SOAP, 2003, etc.) or in the form of particular components (such as an *agent naming service*, *directory facilitator*, *broker agents*, or other components that provide similar functionality).

**F3.5:** *Scalability.* As robotic applications grow in scope and capabilities, a developer must be concerned with how an RDE handles increasing complexity. The term "scalability" can refer to many different aspects of a system, some of which are addressed more specifically by other criteria. For instance, *architectural primitives* (criteria F1.1) and a *high-level language* (F4.1.2) includes facilities for managing complex actions and behaviors, *software engineering* (F1.2) takes into account modularization that promotes system organization, while *distribution mechanisms* (F3.4) encompasses mechanisms used to add computational hosts. Additional concerns might include the overhead involved with message passing, both within a single host and among connected hosts, task allocation for multi-robot applications, or other concerns. Scalability is used here in a broad sense as a general system property, inclusive of the above.

**F3.6:** *Component mobility.* "Mobility" refers to the potential to relocate components at run-time. In robotic applications, however, it is somewhat constrained by possible dependence on the location of the requisite hardware. When an application is distributed across many hosts, component mobility can be used for dynamic resource allocation or run-time system reconfiguration, assuming there are mechanisms that allow reconnection to data sources. Ultimately, these capabilities would be automatic, adjusting operation with a changing computing environment.

**F3.7:** *System monitor/management.* A system monitor displays the status of multiple application components, often in graphical form. An extension of simple monitoring can allow for the management of the components' operation, ranging from starting and stopping to adjustment of parameters. Such extensions are often implemented as part of individual components, which are treated separately as an *implementation characteristic* in Subsection 3 and do not qualify as part of the infrastructure.

**F3.8:** *Security.* An application executing on a single robot may not need any security mechanism, but distribution across many hosts raises such concerns. Predefined components for encryption, authentication, and access control can be available for ready integration into applications. (A related discussion of security concerns in the multi-agent system RETSINA can be found in Singh & Sycara, 2004.)

**F3.9:** *Fault-tolerance.* Repeated failures of both hardware and software are common in robotic applications. The system infrastructure may incorporate generic mechanisms for *failure detection*, or be structured such that disruptions due to failed components do not halt the entire application. Extending this concept, mechanisms for *failure recovery* may exist that enable components to automatically recover from failures with no outside intervention (for instance, see Melchior & Smart, 2004).

## F4: Implementation

In practice, an important reason for selecting a particular RDE is to facilitate the implementation of an agent architecture. We subdivide implementation features into two areas: (1) *implementation characteristics*, which are somewhat abstract and refer to implementation concerns that are not predefined components, and (2) *predefined components*, which perform some specific function that can be directly incorporated into an architecture.

### F4.1: Implementation Characteristics

**F4.1.1:** *Programming language.* Architecture implementation necessitates the use of programming languages, such as C or Java. An RDE that is itself implemented in the particular language used for the application guarantees compatibility; however, an RDE may also supply interfaces or wrappers that interface easily with other languages.

**F4.1.2:** *High-level language.* Some programming environments integrate higher-level languages, such as The Behavior Language (Brooks, 1990), COLBERT (Konolige, 1997), or GRL (Horswill, 2000) for behavior description or

ACL (FIPA-ACL, 2002) or KQML (Mayfield et al., 1996) for agent communication. These high level languages can be used within an agent architecture (e.g., to facilitate data transfer between components) or in multi-robot applications.

**F4.1.3:** *Documentation.* The usability of an RDE is greatly enhanced by the inclusion of well-documented code and user manuals that may include the system's API specification, answers to frequently asked questions, trouble-shooting guides, instructions concerning custom extensions, etc.

**F4.1.4:** *Real-time operation.* Real-time constraints are often critical in designing and operating robot architectures. Real-time capabilities of an RDE are generally dependent on the operating system and/or programming language.

**F4.1.5:** *Graphical interface.* An RDE may supply pre-implemented graphical interfaces that enhance individual component visualization during application execution, including displays related to various sensors, effectors, behaviors, robot control, navigational plans, etc. Additionally, RDEs may define a standardized method of adding such displays.

**F4.1.6:** *Software integration.* RDEs may provide tools that facilitate the integration of external software, either at the component level (e.g., a localization routine) or a complete application-as-component (e.g., speech production), greatly enhancing development time and effort. A notable development in this area is "wrappers" for components of other robotic systems that promote the integration, sharing, and reuse of components.

#### F4.2: Predefined Components

Predefined components are analogous to software libraries; since the list is open-ended and will most assuredly expand in the future, we deviate from the format used thus far and give a necessarily incomplete list of common components with corresponding citations. Furthermore, the list assumes a fairly high-level viewpoint, necessary to maintain an acceptable level of commonality among systems.

Currently, most RDEs include predefined components for *map-making* (**F4.2.1**), *localization* (**F4.2.2**, e.g., Thrun, 2003), *route planning* (**F4.2.3**, e.g., (Konolige, 2000)), *speech recognition* (**F4.2.4**), *speech production* (**F4.2.5**), and *vision processing* (**F4.2.6**, with various capabilities such as blob tracking, edge detection, motion tracking, etc.). Some less common components are *rule interpreters* (**F4.2.7**, e.g., JESS, 2003 or Sloman, 2002), *planners* (**F4.2.8**, e.g., Maes, 1990; Jensen & Veloso, 1998; Stentz, 2002), *neural networks* (**F4.2.9**, e.g., Koker et al., 2004), and *machine learning* (**F4.2.10**, e.g., Vijayakumar et al., 2002; R. Russell, 2004). Even less common, and therefore not included in the evaluation criteria, are support for *instruction/teaching* (e.g., Skubic & Volz, 1998; Bentivegna & Atkeson, 2002), *human robot interaction facilities* (e.g., Fong et al., 2003), *affect* (e.g., Pfeifer, 1988; Moshkina & Arkin, 2003; Scheutz et al., 2006), and *coordination mechanisms* (e.g., Hoff & Bekey, 1995; Chaimowicz et al., 2003; Dias & Stentz, 2003).

# 4 RDE Feature Criteria Evaluations

For each of the RDEs in Section 2, a value has been assigned for the criteria from Section 3, determined using the system's documentation and verified based on usage experience (a synopsis of experimental implementations and the usability evaluation is provided in Section 5). Three types of assignments are made: (1) *binary*, signified by  for *no* and $\surd$ for *yes*, (2) *ternary*, signified by $\square$ for *not supported*, $\boxminus$ for *partially supported*, and $\boxplus$ for *well supported*, and (3) *listings*, which are text descriptions. Table 2 shows the values assigned to each system for each criteria, while further explanation is given in the text. The following shorthand column headings are used to designate particular systems: **TB**–TeamBots, **AR**–ARIA, **P/S**–Player/Stage, **Py**–Pyro, **C**–CARMEN, **ML**–MissionLab, **AD**–ADE, **Mi**–Miro, and **MA**–MARIE.

## F1: Specification

**F1.1** ***Architectural Primitives:*** To attain a *somewhat supported* value, a system must provide at least one form of robot control. Systems that provide additional, likely more complex, methods of robot control receive a *well*

*supported* value. Player/Stage does not provide any predefined control methods, following their policy of providing only the framework for implementing robot control and so receives a *not supported* value. ARIA provides a set of basic actions and an elementary priority-based action resolver. CARMEN provides a Markov decision process planner as part of its navigation component. Each receives a *somewhat supported* value. The rest of the systems are considered *well supported*. TeamBots provides schema-based motor control, finite state machine (FSM) sequencing, and hierarchical behaviors via the *Clay* behavior configuration system. Pyro provides both subsumption and fuzzy blending of behaviors, while MissionLab provides schema-based control, behavior sequencing and artifacts. ADE provides access to a general-purpose rule interpreter, schema-based and subsumption-based behavior primitives, a Prolog interface, and a distributed neural-network style component model based on APOC (Scheutz & Andronache, 2003). Miro includes a custom "behavior engine", based on that introduced in (Brooks, 1991). MARIE, via the RobotFlow and FlowDesigner packages, provides hidden Markov models, fuzzy blending, FSMs, an interface to Octave software, and other primitives.

| **Category** | **Criteria** | **TB** | **AR** | **P/S** | **Py** | **C** | **ML** | **AD** | **Mi** | **MA** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Specification** | F1.1 Architectural Primitives | ⊞ | ⊟ | □ | ⊞ | ⊟ | ⊞ | ⊞ | ⊞ | ⊞ |
| **F1** | F1.2 Software engineering | ⊟ | ⊟ | ⊟ | ⊟ | ⊟ | ⊞ | ⊞ | ⊟ | ⊟ |
| | F1.3 Architecture Neutrality | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| | F2.1 Operating System | J | U,W | U,W | U,W | U | U | J | U,W | U |
| **Platform** | F2.2 Hardware Support | □ | □ | ⊞ | ⊞ | ⊞ | ⊟ | □ | ⊟ | ⊞ |
| **F2** | F2.3 Simulator | ⊟ | ⊟ | ⊞ | ⊞ | ⊟ | ⊟ | ⊟ | ⊟ | ⊞ |
| | F2.4 Configuration Method | □ | ⊟ | ⊟ | ⊟ | ⊞ | ⊞ | ⊞ | ⊟ | ⊞ |
| | F3.1 Low-level Communication | S | S | S | S | I | I | R | C | S |
| | F3.2 Logging Facilities | □ | ⊞ | ⊞ | ⊟ | ⊞ | ⊟ | ⊞ | ⊞ | ⊟ |
| | F3.3 Debugging Facilities | □ | ⊟ | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ |
| **Infrastructure** | F3.4 Distribution Mechanisms | □ | ⊟ | ⊟ | □ | ⊟ | ⊟ | ⊞ | ⊞ | ⊞ |
| **F3** | F3.5 Scalability | □ | ⊟ | □ | □ | ⊟ | ⊞ | ⊞ | ⊟ | ⊟ |
| | F3.6 Component Mobility | □ | □ | ⊟ | □ | ⊟ | ⊟ | ⊞ | ⊟ | ⊟ |
| | F3.7 Monitoring/Management | □ | □ | ⊞ | ⊞ | □ | ⊞ | ⊞ | □ | ⊞ |
| | F3.8 Security | □ | ⊟ | □ | □ | □ | □ | ⊞ | □ | □ |
| | F3.9 Fault-tolerance | □ | □ | ⊟ | □ | ⊟ | ⊟ | ⊞ | ⊟ | ⊟ |
| | F4.1.1 Programming Language | Java | C++ | C++ | Pyth | C | C++ | Java | C++ | C++ |
| | F4.1.2 High-level Language | √ | √ | | √ | | √ | | √ | √ |
| | F4.1.3 Documentation | □ | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ | ⊟ | □ | ⊟ |
| | F4.1.4 Real-time Operation | | | | | | | | | |
| | F4.1.5 Graphical Interface | □ | □ | ⊞ | ⊞ | ⊟ | ⊟ | ⊞ | ⊟ | ⊞ |
| **Implementation** | F4.1.6 Software Integration | □ | □ | ⊟ | ⊞ | ⊟ | □ | ⊟ | ⊟ | ⊞ |
| **F4** | F4.2.1 Map-making | | √ | √ | √ | √ | | √ | | √ |
| | F4.2.2 Localization | √ | √ | √ | √ | √ | | √ | √ | √ |
| | F4.2.3 Route Planning | √ | √ | √ | | √ | √ | √ | | √ |
| | F4.2.4 Speech Recognition | | √ | √ | √ | | | √ | √ | |
| | F4.2.5 Speech Production | | √ | √ | √ | | | √ | √ | √ |
| | F4.2.6 Vision Processing | √ | √ | √ | √ | | √ | √ | √ | √ |
| | F4.2.7 Rule Interpreters | | | | | | √ | √ | | √ |
| | F4.2.8 Planners | | | | | | √ | | | √ |
| | F4.2.9 Neural Networks | | | | √ | | | | √ | √ |
| | F4.2.10 Learning | √ | | | √ | | √ | | | |

Table 2: Feature Criteria Evaluation by RDE

**F1.2** ***Software engineering:*** To attain a *somewhat supported* mark, an RDE must explicitly state design principles, be implemented using an object oriented programming language (e.g., C++ or Java), or make use of a high-level object language (e.g., CORBA). An explicit theoretical foundation yields a *well supported* mark. TeamBots, Player/Stage, ARIA, CARMEN, Pyro, MARIE, and Miro are of the former type, while the use of *Societal Agent*

theory in MissionLab and the APOC formalism in ADE provides the basis for receiving a *well supported* value.

**F1.3** ***Architecture neutrality:*** All the systems under consideration are neutral with regard to agent architectures, although MissionLab has a strong association with the AuRA architecture (Arkin & Balch, 1997) and CARMEN has been described by its authors as an example of the 3T hybrid architecture (Montemerlo et al., 2003b). However, neither enforces the use of the associated architecture, and can therefore be considered agent architecture neutral.

## F2: Platform Support

**F2.1** ***Operating System:*** Compatible operating systems have been determined according to information from system documentation and do not necessarily discount those not listed. If a system, such as TeamBots or ADE, is implemented in Java, the assumption is made that it will execute on any computer platform for which a Java Virtual Machine of the required type is implemented. Similarly, no differentiation is made among the various "flavors" of UNIX, although each system has at least been tested in a Linux environment. Player/Stage, ARIA, and Pyro run on both UNIX and Windows. CARMEN, MissionLab, Miro, and MARIE run on UNIX systems. Letter codes in Table 2 are as follows: J=Java, U=UNIX, W=Windows.

**F2.2** ***Hardware Support:*** Hardware support, as used here, refers to specific robot manufacturers/platforms. We assume a relatively limited pool of sensors and effectors are used across platforms (such as SICK LMS lasers), although all systems allow specification of custom sensors and/or effectors. To attain a *somewhat supported* value, a system must support at least three different platforms; more than five earns a *well supported* value. ARIA supports only MobileRobots robots, ADE supports MobileRobots and Arrick Trilobot, TeamBots supports Cye and Nomad 150 robots, Miro supports MobileRobots, iRobot B21, and their in-house Sparrow platforms, MissionLab supports MobileRobots, iRobot, Evolution Robotics ER-1, and Nomad robots, MARIE supports MobileRobots platforms natively, in addition to all platforms available through ARIA, Player/Stage, and CARMEN via its adapters, Player/Stage supports MobileRobots, iRobot, Segway, Acroname, Botrics, Evolution Robotics, and K-Team platforms, CARMEN supports MobileRobots, Aibo, Nomadics, iRobot, and Segway platforms, and Pyro supports MobileRobots, Aibo, Cye, iRobot, Khepera, Nomad, and Segway platforms.

**F2.3** ***Simulator:*** To attain a *somewhat supported* value, an RDE must at least provide a low-fidelity, 2-dimensional simulator (which may or may not support multi-robot simulations). To attain a *well supported* value, an RDE must provide a high-fidelity, 3-dimensional simulator that supports multi-robot simulations and may be used to model robotic mechanisms. CARMEN includes a 2-dimensional simulator that supports low-fidelity single-robot simulation that can, with some manipulation of the IPC communications, be used for multi-robot simulations. TeamBots, ARIA, MissionLab and ADE provide low-fidelity, multi-robot simulators. MissionLab also supplies a low-fidelity 3-dimensional simulator (although the manual states that its use will halt the system). A major component of Player/Stage, as indicated by its name, is the Stage 2-dimensional simulator, which is low-fidelity and supports multiple robots. Also available is the *Gazebo* high-fidelity 3-dimensional simulator, which elevates Player/Stage to *fully supported* status, along with Pyro and MARIE, which provide interfaces to Stage, Gazebo, and the ARIA and CARMEN simulators.

**F2.4** ***Configuration Method:*** A system, such as TeamBots, that embeds configuration in source code has a *not supported* status. If a system stores configuration in a text file (possibly XML), it receives a *somewhat supported* value. Player/Stage, ARIA, Pyro, and Miro all use text files, of which Miro supports XML. A system that provides a graphical means of accessing and modifying configuration settings gets a *well supported* value, which includes CARMEN, MissionLab, and ADE. MARIE also receives a *well supported* value due to the graphical interfaces included with FlowDesigner and RobotFlow; MARIE itself uses XML configuration files.

## F3: Infrastructure

**F3.1** ***Low-level Communication:*** All systems considered provide socket support and common networking protocols. TeamBots, Player/Stage, ARIA, and Pyro use direct socket connections as their primary method of communication. CARMEN and MissionLab use IPC and IPT, respectively, which adds a level of abstraction to general TCP/IP sockets. ADE uses Java's RMI, while Miro relies on CORBA's IIOP. MARIE makes use of shared

memory or sockets, relying on ACE for the latter. Letter codes in Table 2 are as follows: S=Socket, I=IPC/IPT, R=RMI, C=CORBA IIOP.

**F3.2** ***Logging Facilities:*** All systems provide some means of monitoring component operation as console output or graphical display, which forms the baseline for the value assignment (i.e., a *not supported* value). To gain a *somewhat supported* value, at the very least a system must supply a predefined logging facility; to gain a *well supported* value, a system must allow for remote data capture, run-time starting and stopping of logging, and dynamically configurable data capture that can be recorded in one or more files in one or more locations. TeamBots provides only simple console/graphical output. Logging in ARIA, Player/Stage, CARMEN, ADE, and Miro is *well supported*, while logging in Pyro, MissionLab, and MARIE is *somewhat supported*.

**F3.3** ***Debugging Facilities:*** To attain a *somewhat supported* status, a system must allow non-simulated application interruption and restart in conjunction with the ability to obtain information about component data. To qualify as *well supported*, a system must allow run-time suspension, modification, and replacement of arbitrary components. TeamBots is the only RDE receiving a *not supported* value. ARIA receives a *somewhat supported* value. Player/Stage, Pyro, CARMEN, MissionLab, ADE, MARIE, and Miro all qualify as *well supported*, but MissionLab and MARIE excel due to their integrated and extensive graphical interfaces. MissionLab is unique in that it also uses the included case-based reasoner to analyze a "mission" after completion to identify the source of operational errors. Also notable is ADE's ability to dynamically compile and replace components at run-time using the ADE class loader.

**F3.4** ***Distribution Mechanisms:*** To be elevated from a *not supported* to *somewhat supported* value, a system must include a component that functions as middleware. To qualify as *well supported*, an agent framework that treats components as independent agents is required. Neither TeamBots nor Pyro provide middleware mechanisms and receive a *not supported*. The IPC and IPT software used by CARMEN and MissionLab and the Player server in Player/Stage act as a centralized naming service, while the *ArNetworking* package provided in ARIA fills a similar function. ADE, MARIE, and Miro each specifically incorporate enhanced middleware functionality as part of their infrastructure, earning a *well supported* value.

**F3.5** ***Scalability:*** To use "scalability" as a general reflection of an RDE's properties, a combination of criteria from categories F1, F3, and F4 (specification, infrastructure, and implementation) is used. To earn a *well supported* value, a system must provide scalability support in all categories (as defined below); a *somewhat supported* value indicates support in any two categories, while support for a single category or none at all receives a *not supported* value. In the specification category, an RDE must have a *well supported* value for either *architectural primitives* or *software engineering* (criteria F1.1 or F1.2). For support in the infrastructure category, a system must earn at least a *somewhat supported* value for *distribution mechanisms* (F3.4), while the implementation category is comprised of satisfaction of at least one of *high-level language*, *rule interpreters*, or *planners* (F4.1.2, F4.2.7, and F4.2.8, respectively).

**F3.6** ***Component Mobility:*** To receive a *somewhat supported* value, an RDE must provide architectural components to operate independently of one another in addition to continuing system operation when a component is removed, restarted, and reconnects. To attain a *well supported* value, mechanisms must be in place that can perform this task automatically. TeamBots, ARIA, and Pyro all use a fixed run-time system architecture that does not allow mobility and so receive a *not supported* value. The portability of devices in Player/Stage allows manual component relocation at run-time, as do the modules in CARMEN and the object structure found in MARIE and Miro, while MissionLab provides mechanisms to upload robot executables to remote hosts. Each of these systems receives a *somewhat supported* status. ADE provides mechanisms for saving state, automatic component start-up, and automatic component re-location due to detected failures, earning a *well supported* value.

**F3.7** ***System Monitoring/Management:*** To gain *somewhat supported* status, an RDE must provide an interface that gives access to all components in the system architecture. To gain *well supported* status, an RDE must also provide mechanisms to manage all components. (Note that graphical representations of a robot's sensors, effectors, or other individual components do not qualify as *infrastructure*; see the *Graphical Interface* in Section 4). None of TeamBots, Miro, nor CARMEN provide coherent system-wide facilities. ARIA supplies the MobileEyes GUI for robot display and control, but source code is not freely available and thus earns a *not supported* status. Pyro,

Player/Stage, MissionLab, ADE, and MARIE all have graphical interfaces that not only display component status, but also allow component control.

**F3.8** ***Security:*** To gain a *somewhat supported* value, a system must provide a way to securely authenticate components. To gain a *well supported* value, an RDE must also provide access control and encryption. None of TeamBots, Player/Stage[6], Pyro, CARMEN, MissionLab, Miro, or MARIE use security mechanisms. MARIE and Miro both might inherit security features from their use of ACE for component communication, but do not exploit its availability. ARIA provides authentication services as part of the ArNetworking package, earning a *somewhat supported* value. ADE explicitly addresses all three aspects of security (encryption, authentication, and access control).

**F3.9** ***Fault-tolerance:*** To achieve *somewhat supported* status, a system must isolate components such that failure of a single component does not cause the entire application to fail. To receive *well supported* status, an RDE must also provide mechanisms in support of failure recovery. None of the TeamBots, ARIA, or Pyro RDEs provide component isolation. Player/Stage, CARMEN, and MissionLab, through their reliance on IPC software, each isolates components, while MARIE and Miro's use of ACE objects serve the same purpose. It is worth noting that MissionLab also incorporates a case-based reasoning wizard for the purpose of repairing a mission post-execution (Moshkina et al., 2006) and that ACE implements the Fault Tolerant CORBA specification, although neither MARIE nor Miro have yet incorporated it. ADE provides both fault detection and fault recovery at the component level through its use of heartbeats between `ADEServer` s and clients.

## F4: Implementation

### F4.1 System Implementation Characteristics

**F4.1.1** ***Programming Language:*** Both TeamBots and ADE are written in Java, while CARMEN and Pyro are implemented in C and Python, respectively. Player/Stage, ARIA, MissionLab, MARIE, and Miro are implemented in C++.

**F4.1.2** ***High-level Language:*** To qualify as supporting a high-level language, an RDE must supply a structured method for controlling a robot (e.g., a behavior or agent communication language). TeamBots supplies the Clay behavior hierarchy, ARIA provides action specification via the *ArAction* class, MissionLab provides both CDL and CMDL, Pyro and MARIE supply both a set of foundational behavior classes and finite state automata, and Miro provides a "behavior engine" for behavior specification. None of Player/Stage, CARMEN, nor ADE provide a high-level language.

**F4.1.3** ***Documentation:*** To attain a *somewhat supported* value, an RDE must have well documented source code and a publication outlining its use. If an RDE also supplies a manual that describes how to use the system (including installation instructions, guidelines for developing applications and extending capabilities into new areas, solutions to common problems, and example code), it receives a *well supported* value. While TeamBots, ADE, Miro, and MARIE all provide some level of documentation, both web-based and in source code, it is either incomplete or they do not provide finished manuals that detail their use. Player/Stage, ARIA, CARMEN, and MissionLab all have complete and detailed manuals available, while Pyro provides the equivalent through its extensive online documentation.

**F4.1.4** ***Real-time Operation:*** None of the systems directly provide real-time support, although MissionLab has the mechanisms in place for use with a purchased license of proprietary software from Honeywell.

**F4.1.5** ***Graphical Interface:*** To obtain a *somewhat supported* value, an RDE must supply graphical interfaces for visualizing component operation or designing control code without actual programming. To receive a *well supported* value, an RDE must provide both items just mentioned, in addition to a standard method for creating new displays. Only TeamBots does not provide a graphical display for a robot at run-time (although it does supply a graphical simulator facility), and so receives a *not supported* value. While the MobileEyes GUI is available with ARIA, source code is not freely available and thus has to be classified as *not supported*. CARMEN provides

[6]While the Player server in Player/Stage can optionally be set to require authentication, it is explicitly acknowledged that the authentication is *not* for security, as keys are passed in plain text.

*ad hoc*, component specific graphical interfaces, but does not provide standard methods for adding visualization nor graphical control code tools, and so receives a *somewhat supported* value. MissionLab and Miro provide interfaces that allow developers to design control code without actual programming, but do not provide a standard method for defining new displays, also earning them a *somewhat supported* value. Player/Stage, ADE, Pyro, and MARIE all provide both implemented displays and a standardized method of creating new displays, earning *well supported* values.

**F4.1.6** ***Software Integration:*** To attain *somewhat supported* status, an RDE must provide a standard API or mechanism for incorporating "outside" software, generally using socket connections (with the recognition that translation code will always have to be written). Providing additional codified facilities that interface with other RDEs, thereby allowing their software to be "dropped into" the environment, elevates the status to *well supported*. Neither TeamBots, ARIA, nor MissionLab provide such standard APIs or mechanisms. Player/Stage and CARMEN provide such APIs (for their devices and modules, respectively), Pyro and ADE explicitly include steps to "wrap" external software, MARIE supplies a variety of APIs and mechanisms for integration, while Miro relies on writing TAO interfaces in Interface Device Language (IDL), used to produce C++ code. Each receives at least a *somewhat supported* value. MARIE and Pyro also provide translation facilities such that components written for CARMEN, Player/Stage, or ARIA can be used and earn a *well supported* value.

### F4.2 Predefined Components

As mentioned earlier, any list of predefined components is open-ended and therefore necessarily incomplete. We limit this list to components that are commonly available and only list the RDEs that include them. Furthermore, no quantitative evaluation is given; the intent is not to establish a full taxonomy, but to provide a high-level indication of system functionality. It should also be noted that both ARIA and MissionLab provide some of the following components so long as they are licensed; due to the limitation of this survey to open source software, such components have been excluded.

**F4.2.1** ***Map-making:*** ARIA provides the *Basic Mapper* software, which can be used to manually construct maps. Player/Stage, Pyro, CARMEN, ADE, and MARIE all include map-making facilities, which are combined with localization.

**F4.2.2** ***Localization:*** TeamBots provides a landmark-based localization component, while Miro provides particle filter localization. ARIA provides sonar-based localization, but full localization facilities must be purchased. Player/Stage, Pyro, CARMEN, ADE, and MARIE all include localization facilities, which are combined with map-making.

**F4.2.3** ***Route Planning:*** TeamBots and ADE provide schema-based navigation, ARIA supplies a navigator integrated with its localization package, Player/Stage provides a wavefront propagation route planner, CARMEN uses a Markov decision process planner, MissionLab relies on geometric map analysis and an A* graph search, and MARIE integrates Player/Stage and CARMEN navigation components.

**F4.2.4** ***Speech Recognition:*** Player/Stage, ARIA, ADE, Pyro, and Miro all provide speech recognition support through integration of outside software such as Sphinx (Sphinx, 2004) or Sonic (Pellom & Hacioglu, 2003).

**F4.2.5** ***Speech Production:*** Player/Stage, ARIA, ADE, Pyro, MARIE, and Miro all provide speech production support through integration of outside software such as Festival (Festival, 2004).

**F4.2.6** ***Vision Processing:*** MissionLab and Miro have basic image/video capture capabilities, but Miro also provides stereo image capture and many video filters. TeamBots includes CMVision software, which can capture images and perform blob detection. ARIA has two vision packages available, the ActivMedia Color Tracking Software (ACTS) and VisLib. ACTS is a blob detection package, while VisLib includes image filters, blob detection, and object tracking. Player/Stage supports both ACTS and CMVision. Pyro includes image/video capture, blob, edge, and motion detection, assorted filters, and stereoscopic tools, implemented in C++ for speed reasons. ADE includes both an ACTS interface and custom blob detection, object tracking, and face/emotion detection. MARIE, via the RobotFlow software, provides custom image capture, blob detection, movement detection, text/symbol extraction routines, and supports OpenCV.

**F4.2.7** ***Rule Interpreters:*** ADE includes an interface to POP-Rulebase (Sloman, 2002) and Prolog, while MARIE and MissionLab both include custom rule interpreters.

**F4.2.8** ***Planners:*** While item F4.2.3 specifically covers navigation planners, these are considered too task-specific to qualify under the general rubric of "planner". MissionLab and MARIE both supply generic planners.

**F4.2.9** ***Neural Networks:*** Pyro, Miro, and MARIE all include neural network software.

**F4.2.10** ***Learning:*** TeamBots provides both reinforcement and Q-learning components, Pyro has a reinforcement learning module, while MissionLab includes integrated case based reasoning and Q-learning components. We do not include neural network software under the *learning* heading, as it appears as a separate category above.

# 5 RDE Usability Evaluations

The systematic comparison of RDEs with respect to their supported features based on a conceptual framework is one important part of an RDE evaluation. Another important part is RDE *usability*, for the extent to which an RDE can be easily installed and used in research is ultimately a decisive factor for its adoption. Yet, surprisingly, there is only one previous study (Orebäck & Christensen, 2003) that provides a practical RDE evaluation. And while a robotic architecture was actually implemented and executed on a robot in Orebäck & Christensen, 2003, their study is very limited in scope (only three RDEs were evaluated according to a small set of criteria based on a single application) and does not provide a methodology for systematic comparisons and subsequent evaluations that ties together conceptual, practical, and impact factors. Consequently, the conclusions Orebäck and Christensen (2003) arrived at have limited applicability.

We believe that a comprehensive evaluation needs to encompass at least the three categories of usability criteria:

**U1: Installation.** Basic steps required to obtain a usable system, evaluated in terms of the required time and effort.

**U2: Basic usability.** Implementation and execution of a simple "lowest common denominator" architecture for RDE comparison, focussing on "low-level" sensor and effector access and allows for an investigation of architectures that reside on a single host.

**U3: Advanced usability.** Usage of individual, predefined, "high-level" components that would commonly form sub-architectures of a complex, distributed architecture; an effort is made to explore uncommon (and possibly unique) "high-level" system features.

The following subsections describe the three categories in more detail. As was done with features in Sections 3 and 4, a set of criteria is defined and subsequently evaluated. Due to the variability of each criteria's subject, value meanings are specified per item; in general they can be interpreted as: $\square$ for *below average*, $\boxminus$ for *average*, and $\boxplus$ for *above average*. While an attempt has been made to adhere to a ternary value assignment, a value of *na* is used to indicate that a specific item was not examined in sufficient depth to assign a value for some reason (e.g., incompatible hardware, difficulties with prerequisite software, etc.). *na* values will not be included in an RDE's score. Results are shown in Table 3, where the following shorthand column headings are used to designate particular systems: **TB**–TeamBots, **AR**–ARIA, **P/S**–Player/Stage, **Py**–Pyro, **C**–CARMEN, **ML**–MissionLab, **AD**–ADE, **Mi**–Miro, and **MA**–MARIE.

All systems were installed on at least two computers out of a selection of five: two laptops, two desktops, and the onboard PC of an ActivMedia PeopleBot P2DXe robot (shown on the right in Figure 1). None of the computers were the same make and model, with varying CPUs (850MHz Pentium III, 1.3GHz and 2.0GHz Pentium M, 2.3GHz Pentium 4, and a 1.8GHz AMD Athlon) and memory capacities (from 128MB-1GB), although all used Linux (either Debian or Fedora distributions) running a 2.6.x kernel. Various supporting hardware included microphones, speakers, a Firewire camera, and both wired and wireless Ethernet networking. All non-simulated experiments were conducted on the robot, which also has a pan-tilt unit, sonar, bumpers, and a SICK LMS200 laser range finder.

## U1: Installation

Prior to actually using an RDE, it must be properly installed. Since we believe that installation difficulties might often be a deterrent for potential users, we give "Installation" its own category and criteria.

| Category | Criteria | TB | AR | P/S | Py | C | ML | AD | Mi | MA |
|---|---|---|---|---|---|---|---|---|---|---|
| Install | U1.1 Documentation | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ |
| | U1.2 Non-RDE Installation | ⊞ | ⊞ | ⊞ | ⊞ | ⊞ | ⊟ | ⊟ | □ | □ |
| | U1.3 RDE Installation | ⊞ | ⊞ | ⊞ | ⊞ | ⊟ | □ | ⊞ | □ | ⊟ |
| | U1.4 Installation Usability | ⊞ | ⊟ | ⊞ | ⊞ | ⊟ | □ | ⊟ | □ | □ |
| Low-level | U2.1 Documentation | □ | ⊟ | ⊞ | ⊞ | ⊞ | ⊟ | ⊟ | ⊟ | ⊟ |
| | U2.2 Architecture Implementation | □ | ⊟ | ⊞ | ⊞ | ⊞ | ⊟ | ⊞ | □ | □ |
| | U2.3 Architecture Execution | ⊞ | ⊞ | ⊟ | ⊞ | □ | ⊞ | ⊞ | ⊟ | □ |
| | U2.4 Graphical Tools | □ | ⊟ | ⊟ | ⊞ | ⊟ | ⊞ | ⊞ | ⊟ | ⊟ |
| | U2.5 Overhead (memory, CPU) | ⊞† | ⊟ | ⊟ | ⊟ | ⊟ | ⊟† | ⊞ | ⊟† | ⊞ |
| | U2.6 "Low-level" Usability | □ | ⊟ | ⊞ | ⊞ | ⊞ | ⊟ | ⊟ | □ | □ |
| High-level | U3.1 Documentation | *na* | ⊟ | ⊞ | ⊞ | ⊞ | ⊟ | ⊟ | □ | ⊟ |
| | U3.2 Predefined Components | *na* | ⊟ | ⊞ | ⊞ | □ | ⊞ | ⊟ | ⊟ | ⊞ |
| | U3.3 Task Implementation | *na* | ⊟ | ⊞ | ⊞ | ⊟ | ⊞ | ⊞ | ⊟ | ⊟ |
| | U3.4 Distribution | *na* | ⊟ | ⊟ | □ | ⊟ | ⊞ | ⊞ | ⊟ | ⊟ |
| | U3.5 Graphical Tools | *na* | □ | ⊟ | ⊟ | □ | ⊞ | ⊟ | □ | ⊞ |
| | U3.6 System Integration | *na* | □ | ⊟ | ⊞ | ⊟ | ⊞ | ⊞ | ⊟ | ⊞ |
| | U3.7 "High-level" Usability | *na* | □ | ⊞ | ⊞ | ⊟ | ⊞ | ⊟ | □ | ⊟ |

Table 3: Usability Criteria Evaluation by RDE (a † indicates that execution of the "low-level" architecture was done in simulation due to difficulties running it on the robot).

**U1.1** ***Documentation:*** *Installation documentation* refers specifically to how well the documentation described the installation process, including required preparatory steps and supporting software, minimum system specification, a clearly laid-out sequence of instructions, a list of known or potential issues, inclusion of mailing list or contact addresses, and references to further information. Satisfying more than five of the above requirements receives a ⊞ value, three or four receives a ⊟ value, while less than three receives a □.

**U1.2** ***Non-RDE Installation:*** In all cases, installation required additional supporting software. In some cases, this is limited to a single package (e.g., an adequate Java system), while in others, a large set of additional software is required to enable all available features. A value of ⊞ indicates that installing non-RDE software (including, if necessary, determining what supporting software was required, actual compilation and installation, and any needed debugging) took less than three hours, a ⊟ indicates less than two days, while a □ indicates more than two days. Due to the level of detail necessary to cover the variety and scope of additional software packages, we do not address many of the related issues encountered, although some additional information is given as part of criteria U1.4.

**U1.3** ***RDE Installation:*** *RDE installation* refers to the steps necessary to have a usable system, assuming all supporting software has been installed. Values are the same as non-RDE installation (U1.2): a ⊞ value indicates that installation took less than three hours, a ⊟ indicates less than two days, and a □ indicates two or more days were required to attain a usable system.

**U1.4** ***Installation Usability:*** *Usability*, as related to the installation procedure, is the overall (and ultimately subjective) impression of the experience. Values are assigned relative to the other systems, thus three systems each receive □, ⊟, and ⊞ values. The following notes provide selected information (presented in no particular order) gathered during the implementation process that help explain the evaluations:

- Pyro has a bootable "LiveCD" available, which should avoid installation issues altogether. However, actual robots rarely have a CD drive, making this irrelevant for non-simulated use. The packages in the Pyro `yum` repository conflicted in some cases, but manual installation was done without issue.
- The version of MissionLab available required the use of `gcc` version 3.2 or below and related libraries, which, due to its age and incompatibility with current versions, was the cause of time-consuming installation issues.

**FUNCTION** simple_architecture
$turning, forward = 0$
**while** true **do**
  $R \leftarrow getRanges()$
  **for all** $r \in R$ **do**
    $turning = turning + \frac{\alpha \times cos(r_{angle})}{r_{distance} \times r_{distance}}$
    $forward = forward - \frac{\alpha \times sin(r_{angle})}{r_{distance} \times r_{distance}}$
  **end for**
  $forward = forward + \beta$
  $setVelocity(forward, turning)$
**end while**

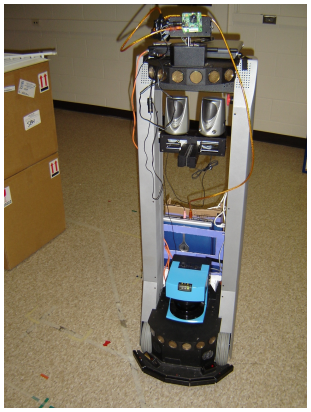


Figure 1: **Left:** The "simple" architecture algorithm implementing a wander behavior with obstacle avoidance. At each time step, a set of polar range readings $R = (r_1, r_2, \ldots, r_n)$ is obtained, where $-\pi \leq r_{angle} \leq \pi$ ($r_{angle} = 0$ is straight ahead) and $r_{distance}$ is relative to the center of the robot. The rotational velocity $turning$ is calculated by summing the $x$ component $cos(r_{angle})$ of polar readings, divided by square of the distance $r_{distance}$ to account for obstacles, multiplied by some system-dependent scalar $\alpha$. The translational velocity $forward$ is calculated similarly, using the $y$ component $sin(r_{angle})$, subtracted from the total to make it repulsive, then adding a constant $\beta$ for default forward movement. **Right:** The robot on which experiments were performed.

- Installation of ACE/TAO software, required for Miro and MARIE, was particularly time-consuming, particularly due to an initial misconfiguration that required multiple manual de- and re-installation. Miro requires a particular ACE/TAO configuration installation, which is not documented.
- Installation of supporting packages for ADE includes a hardware interface for Java, Player/Stage for simulation, a secure shell client and server for distribution, and assorted other packages to attain the full complement of system functionality.
- Few, if any, installation issues were experienced with TeamBots, ARIA, Player/Stage, and CARMEN. The issues that were encountered mostly concerned platform configuration (e.g., appropriate privileges and permissions, default hardware settings that differed from the particular configuration in use, etc.).

## U2: Basic Usability

*Basic usability* in this context means to be able to implement and execute a simple robotic architecture to be able to test a minimal set of capabilities supplied by each RDE. The implemented architecture consists of a basic wander behavior that incorporates obstacle avoidance. Only motor control and range finder sensors were accessed, in as direct a manner as possible, providing a "lowest common denominator" for RDE comparison. The basic algorithm, which uses a potential field method, is shown on the left side of Figure 1. Note that because the robots available to the authors are not supported by TeamBots, the architecture was implemented but execution could only be done in simulation. Similarly, MissionLab and Miro were also run in simulation due to repeated failures. Each time an installation was not successful, the error was located, fixed, and another attempt was made. Once simulated architecture execution was possible, a few additional attempts were made to run the architecture on the robot; when these also failed, simulated results were deemed acceptable. Systems that were only evaluated in simulation are marked with a † in Table 3.

**U2.1** ***Documentation:*** In relation to the "low-level" architecture, *documentation* refers to information that enhances basic usability (e.g., APIs, example code, a frequently asked question list, etc.). A ☐ value indicates that it was difficult to find information concerning either basic functionality (e.g., how to send a command to the robot) or a solution to a relatively simple problem (e.g., how to start architecture execution). A ⊟ value indicates that information was available but required some effort to locate, while a ⊞ value indicates that very little effort was required to find installation and implementation information.

**U2.2** ***Architecture Implementation:*** *Architecture implementation* refers to the process of writing the program that performs the wander behavior with obstacle avoidance. A $\square$ value indicates that implementing the simple architecture was a major undertaking (requiring more than two days), a $\boxminus$ value indicates that substantial effort was involved (requiring more than 5 hours), and a $\boxplus$ value indicates that implementation required an expected amount of effort (less than 5 hours). It is important to note that because the simple architecture was a low-level implementation that circumvented or avoided the abstractions and/or tools supplied by some RDEs (e.g., TeamBots' Clay system, ARIA's predefined actions, or MissionLab's behavior libraries), the value is not necessarily indicative of what might be considered "normal" usage, which is considered instead as part of "high-level" usability.

**U2.3** ***Architecture Execution:*** *Architecture execution* refers to the effort required to start and stop a fully implemented architecture (including both the control code and supporting software, if they are separate). A $\square$ value indicates a sequence of more than four steps, perhaps requiring multiple terminal connections that each require separate initialization. A $\boxminus$ value indicates that two to four steps are required, sometimes mitigated by the GUI. A $\boxplus$ value indicates that startup and shutdown requires a single step[7].

**U2.4** ***Graphical Tools:*** For the "low-level" architecture, *graphical tools* refer to the non-command line interface presented by an RDE. A value of $\boxplus$ is given if the system provides both a display of basic operational information and a graphical manner of starting and stopping architecture execution. A value of $\boxminus$ is given if an RDE provides either, while a value of $\square$ indicates that neither are provided.

**U2.5** ***Operational Overhead:*** *Operational overhead* refers to the memory and CPU ($M$ and $C$, respectively) resources used during architecture execution[8]. Initial conditions were kept constant by rebooting the computer for each system, turning the swap file off, then executing the architecture three times, recording measurements at one second intervals. Each execution run is divided into two phases: (1) *startup* (denoted by a subscript $S$), demarcated by the time just prior to system startup until robot movement is first detected and (2) *execution* (denoted by a subscript $E$), which begins when robot movement is detected, continuing for 90 seconds.

Figure 2 shows the average (with standard deviation bars) and maximum values across all three runs. The average values form the basis for calculating evaluation scores by dividing the range in thirds that are assigned values of 0 for the top third, 1 for the middle third, and 2 for the lower third. More specifically, $M_S$ and $M_E$ receive: <20MB = 2, <40MB = 1, and >40MB = 0. For $C_S$ and $C_E$, <33% = 2, <66% = 1, and >66% =0. The values shown in Table 3 are the sum of $M_S$, $M_E$, $C_S$, and $C_E$, where a total of 6 or better receives a $\boxplus$, a 4 or 5 receives a $\boxminus$, and 3 or less receives a $\square$. It is interesting to note that, in most cases, the operational overhead displays the classic memory vs. CPU usage tradeoff in both the startup and execution phases.

**U2.6** ***"Low-level" Usability:*** *Usability*, as related to the "low-level" architecture, is the overall (and ultimately subjective) impression of the experience. Values are assigned relative to the other systems, thus three systems each receive $\square$, $\boxminus$, and $\boxplus$ values. The following notes provide selected information (presented in no particular order) gathered during the implementation process that help explain the evaluations:

- Execution for TeamBots, MissionLab, and Miro were performed in simulation (denoted by a $^\dagger$ in Table 3) due to unsupported laser hardware in TeamBots and difficulties in hardware communication for MissionLab and Miro.
- Although RobotFlow supplies components for interfacing with a Pioneer and SICK lasers, execution for MARIE used a Player server as the hardware interface so that MARIE functionality was included in the performance measurements.
- Inclusion of an easily accessible simulator was extremely useful in implementing and debugging an architecture; Player/Stage and CARMEN are particularly strong in relation to simulator integration, while Miro was relatively difficult to access.
- The structure of TeamBots is such that implementations that deviate from the included software (e.g., non-Clay behaviors or unsupported hardware) are very difficult to program.

[7] We do not consider placing a sequence of commands in a shell script for execution as a single step.

[8] While disk space usage and bandwidth are important, neither is considered. We exclude disk space due to the variability of packages required, while bandwidth is not addressed due to the single-host nature of the "low-level" architecture.

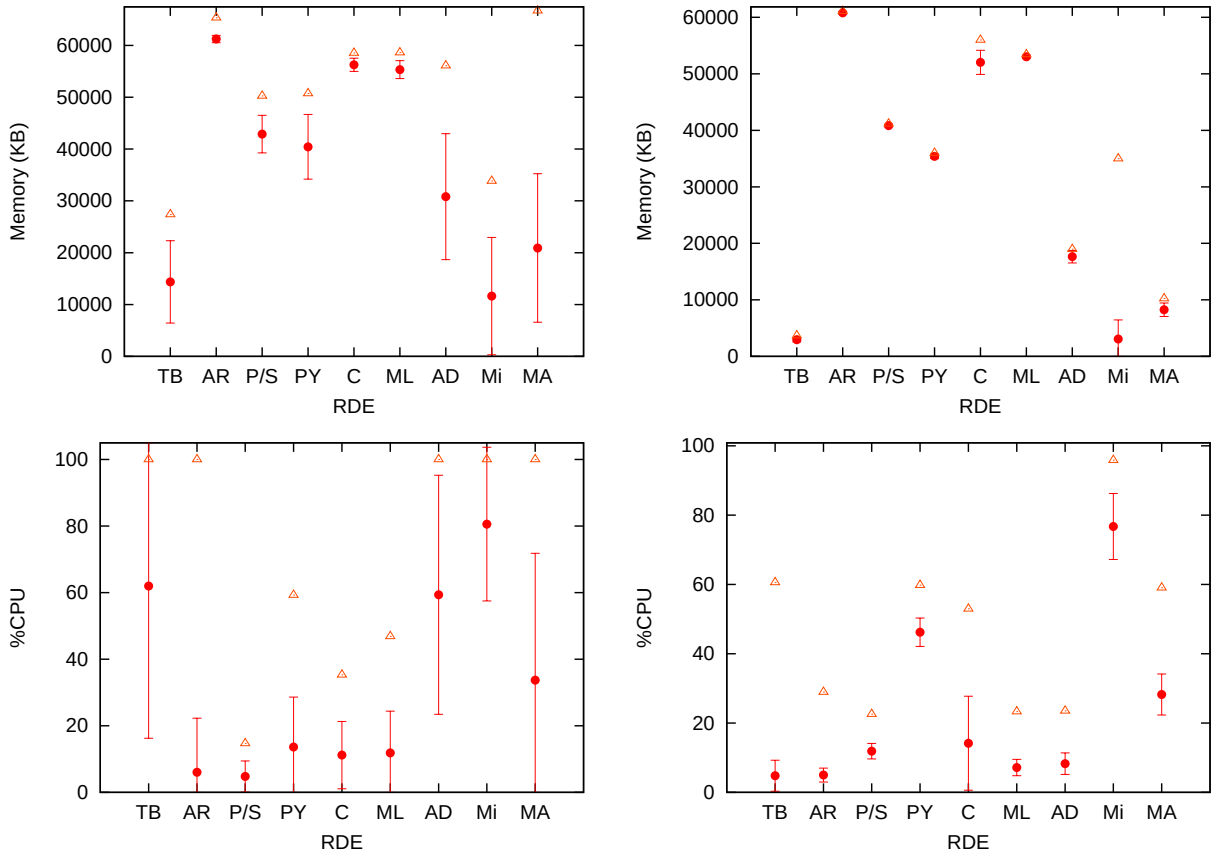



Figure 2: Operational overhead of each RDE executing the "low-level" architecture. **Top Row:** Memory usage (in KB), showing average (with standard deviation bars) and maximum values. The left-hand figure shows the startup phase, while the right shows the execution phase. **Bottom Row:** CPU usage (as a % of system load), again showing average and maximum values. As with the memory overhead, startup is on the left and execution is on the right.

- CARMEN and MissionLab require some knowledge of IPC messages to implement custom components or write routines that access the available components.
- Player/Stage, Pyro, and ADE provide a good balance of high-level component abstractions while also preserving accessibility to low-level sensor and effector interfaces.
- Player/Stage and Pyro provide a varied base of example code combined with relatively easy configuration and relevant documentation.
- Both MissionLab and CARMEN provide excellent documentation; MissionLab has a clear and complete user manual, while CARMEN's documentation is structured and organized in a manner that made it easy to find desired information.
- Although ARIA provides very good documentation, the "low-level" nature of sensor and effector access used in this architecture deviates from the standard methods addressed therein, leading to a more difficult implementation.
- While both the FlowDesigner and MissionLab GUIs are extensive and polished, some low-level tasks are easier to perform using a text editor (in other words, the GUI was actually a hindrance in some respects).

## U3: Advanced Usability

While the "low-level" architecture provides a lowest common denominator for subjective RDE evaluation, the implementation of what we will refer to as "high-level subarchitectures" provides a basis for evaluating an RDE under expected normal usage (although with a focus on distributed computing which becomes necessary due to architecture complexity and real-time processing constraints). Because there are few areas in which all RDEs provide the same capabilities in the same way, the underlying idea is to obtain an estimate of the effort required to distribute a single complex architecture over at least two hosts. A model example of such an architecture is DIARC (Scheutz et al., 2006), which provides a foundation for experiments in human-robot interaction.

The "fundamental" components examined are: (1) vision processing (monocular, that uses blob-detection), (2) speech recognition, (3) speech production, and (4) the provided robot control primitives, although attempts were made to use of some (possibly unique) components and capabilities (e.g., MissionLab's selection of behaviors, ADE's autonomous distribution, etc.). The implementation consisted of installing, configuring, and testing individual components, then connecting at least two of them across a network as an indication of ease of distribution, with the assumption that connections among the other components will require similar effort. No attempt was made to implement these tasks in TeamBots, due to both the limited number of predefined components and because the robots available to the authors are not supported.

**U3.1** ***Documentation:*** In relation to the varied tasks, *documentation* refers to information concerning the advanced RDE functionality (e.g., explanation of available components and their use, example code, etc.). In particular, the focus was on the components required for the varied tasks (vision, speech recognition, speech production, and robot control) and their distribution. A $\square$ value indicates either missing or incomplete information for at least two basic functionalities. A $\boxminus$ value indicates that information was either missing or incomplete for only one basic functionality or that the supplied documentation for any basic component was minimal or unclear. A $\boxplus$ value indicates that documentation was complete, easy to locate, and highly readable.

**U3.2** ***Predefined Components:*** *Predefined components* refers to the integrated capabilities included with an RDE. A $\square$ value indicates that an RDE was either (1) missing at least one component required to implement all "fundamental" functionality for the envisioned architecture (robot control, vision, speech production, and speech recognition) or (2) did not provide at least two additional components for each missing "fundamental" component. A $\boxminus$ value indicates that the basic components were available or at least two other components were available for each missing basic component. A $\boxplus$ value indicates that the RDE comes packaged with more than four predefined components beyond what is required for a $\boxminus$ value, in addition to other tools or functionality. Of particular note are Player/Stage, Pyro, MissionLab and MARIE: Player/Stage supports the largest number of devices, Pyro and MissionLab include various additional functionalities (e.g., Pyro includes working examples from S. Russell & Norvig, 2002, MissionLab includes a case-based reasoner for post-execution analysis), while MARIE provides many signal processing and vision tools via the RobotFlow and FlowDesigner packages.

**U3.3** ***Task Implementation:*** *Task implementation* refers to the effort required to implement the subarchitectures, both in terms of accessing individual components and their distribution. As with the "low-level" architecture, a $\square$ value indicates that implementation was a major undertaking (requiring more than two days), a $\boxminus$ value indicates that substantial effort was involved (more than five hours), and a $\boxplus$ value indicates that implementation required an expected amount of effort (of less that five hours). It is imperative to note that the "high-level usability" case made use of the abstractions and/or tools supplied by some RDEs (which sometimes proved detrimental), which may account for differences from the value given for the "low-level" architecture.

**U3.4** ***Distribution:*** *Distribution* refers to the ease of locating components across hosts once the distribution mechanisms have been implemented, accounting for both startup/shutdown procedures and relocation of components. A $\boxplus$ value indicates that the system provided facilities for automatic login and component startup/shutdown, in addition to providing the means to reconfigure component location. A $\boxminus$ value indicates that only one of those specifications was met, and a $\square$ value indicates that neither is supported. Of note is that RDEs in which components must be implemented with network capabilities (e.g., components in CARMEN or Miro and servers in Player/Stage or ADE) all have a relative advantage.

**U3.5** ***Graphical Tools:*** In relation to the "high-level subarchitectures", *graphical tools* refers to both the presentation of an integrated display and the means to graphically implement robot control procedures. This differs from

the "low-level" architecture in that (1) an emphasis is placed on integration, such that architecture display and system control is consolidated and (2) a user does not have to write low-level programs. A value of ⊞ is given if an RDE provides both, a value of ⊟ if only one is provided, and □ if neither is provided.

**U3.6** ***System Integration:*** *System integration* refers to the separate issues of (1) easily connecting and controlling components in a complex architecture and (2) coherent system usage in terms of providing a consistent interface to the complete system that allows access to and control of individual components. A ⊞ value is assigned if both objectives are met, a ⊟ if only one is satisfied, and a □ if neither.

**U3.7** ***"High-level" Usability:*** As with the "low-level" architecture, *usability* is the overall (and subjective) impression of the experience of using each RDE. Values are assigned relative to the other systems, thus three systems each receive ⊟ and ⊞ values, while two receive a □ value (because TeamBots was not evaluated in terms of the high-level tasks). The following notes provide selected information (presented in no particular order) gathered during the implementation process that help explain the evaluations:

- Both ARIA and MissionLab have additional components that were not considered here as they require licensing and were not included in the downloadable package.
- While ARIA's documentation is in general very good, the distribution package, `ArNetworking`, lacks complete documentation. For some basic tasks, MARIE's documentation is sparse, limited to "node" listings, while ADE and Miro are comparably spotty.
- Pyro's interface is well integrated, allowing access to various conceptually separate parts of an application (i.e., the *server*, *robot*, *devices*, and *"brain"*), while also providing the ability to enter Python commands at run-time. However, non-graphical usage is not quite as polished (for instance, hanging when exiting the system).
- The user interface provided by MissionLab is especially suited to task specification implemented by non-programmers, matching its objective of providing a high-level view of robot control.
- Systems that require the strict use of abstractions in support of their component model (e.g., the ACE/TAO in Miro and FlowDesigner networks in MARIE) or, to a lesser extent, require some level of abstraction removed from actual source code (e.g., IPC/IPT communications in CARMEN and MissionLab and RMI in ADE) can be either beneficial or detrimental to some degree. For instance, Miro's requirement that all components are CORBA objects makes component integration and distribution extremely simple, but the implementation of an arbitrary component is made more difficult.

# 6 Discussion

The evaluations presented in Sections 4 and 5 provide the foundation for comparing RDEs, both at a conceptual level and from a practical perspective. Augmented with an impact evaluation (to be described below), we envision the results of this survey being useful to the robotics community in at least two ways: (1) by providing *researchers* with a practical means of selecting an RDE that will most closely match their requirements and (2) by giving RDE *developers* an overview of the innovations being made in other systems, possibly suggesting improvements and extensions to their own system.

## 6.1 Researchers

To perform a comprehensive comparison of the RDEs presented, three separate measures are given: (1) a summary of the evaluations from Section 4 concerning an RDE's *features*, (2) a summary of the evaluations from Section 5 concerning an RDE's *usability*, and (3) an estimate on the influence an RDE has had on the robotics field (i.e., its *impact*), which is gauged by the breadth of publications from different research areas (as listed at the end of each description from Section 2) and the number of other RDEs that provide interoperability interfaces with a system. A researcher examining a group of RDEs to find one that best fits their needs might conduct evaluations using either *qualitative* or *quantitative* measures.

A qualitative evaluation is highly contingent on the user's purpose; on the one hand, very specific capabilities may be required, while on the the other, needs may be highly abstract or only loosely defined. For instance, an application

designer who has a large body of already written Octave software and desires to use it with a robot might look at the system descriptions in Section 2 and find that MARIE already has an Octave plug-in. Conversely, educators establishing an "Introduction to Robotics" class might consider the items from the usability evaluation in Section 5 to be of overriding importance; an examination of the values given to the *documentation* criteria (U1.1, U2.1, and U3.1) might lead them to limit attention to Player/Stage, Pyro, and CARMEN.

More likely, however, is that a *mixture* of characteristics is desired. For example, a developer might require a system that provides a simulated environment and a fair level of distribution facilities, with a preference for a system that supplies extensive GUI capabilities and usability oriented towards non-programmers. An examination of Section 4 would lead to considering criteria F2.3 (*Simulator*), F3.4 (*Distribution Mechanisms*), F4.1.2 (*High-level Language*), and F4.1.5 (*Graphical Interface*), while Section 5 would U2.4 (*Graphical Tools*) and U3.1-U3.7 (*High-level Usability*), making MissionLab the most likely choice.

To arrive at a quantitative evaluation, an assignment of values to criteria must be made, which can then be applied to a selection (or all) of the criteria. To arrive at numerical comparison scores, the three categories of criteria mentioned above are retained, yielding a *feature* score $F$, *usability* score $U$, and *impact* score $I$, which can be summed to give a *total* score $T$. Within each category, values of 0 or 2 are assigned to binary-valued criteria and values of 0, 1, or 2 to ternary-valued criteria (listing features are not scored)[9].

In formal terms, the selected RDEs form the set $S = \{TB, AR, P/S, Py, C, ML, AD, Mi, MA\}$ and are assigned a score that is the sum of the category scores $F$, $U$, and $I$. Given a number of individual criteria within each category (e.g., $F_{1.1}$ denotes *Architectural Primitives*, while $U_{1.1}$ denotes *Installation Documentation*), category scores are comprised of the sum of individual criteria values $F_i$, $U_j$, and $I_k$, where $m$, $n$, and $o$ are the total number of feature, usability, and impact criteria and $1 \leq i \leq m$, $1 \leq j \leq n$, and $1 \leq k \leq o$. In the simplest case, where all criteria are given equal weight, the RDE comparison scores are given by the formula:

$$T_{s \in S} \leftarrow \sum_{i=0}^{m} F_i^s + \sum_{i=0}^{n} U_i^s + \sum_{i=0}^{o} I_i^s$$

Should a quantitative evaluation that weights some categories or criteria more or less than others be desired, weights can be assigned at both a coarse-grained level (for each category, $\alpha, \beta$, and $\gamma$) and a fine-grained level (for each criterion within a category, $W_{F_i}, W_{U_j}$, and $W_{I_k}$). The resultant formula for establishing the total comparison scores is:

$$T_{s \in S} \leftarrow \alpha \sum_{i=0}^{m} W_{F_i} F_i^s + \beta \sum_{i=0}^{n} W_{U_i} U_i^s + \gamma \sum_{i=0}^{o} W_{I_i} I_i^s$$

where $\alpha + \beta + \gamma = 3$, $\sum W_{F_i} = m$, $\sum W_{U_i} = n$, and $\sum W_{I_i} = o$. This method will give an objective comparison in that scores are not biased for or against any particular RDE, although the choice of features and their assigned weights are based on the particular application requirements.

The quantitative evaluation that does not weight any criteria or category more than another follows, supplemented by a discussion of each category score and the totals. The tabulated scores are shown in Tables 4, 5, and 6, respectively, and summarized in Table 7.

MARIE and ADE have the highest score (44 and 43, respectively) in terms of the *Feature* score $F$. The *Implementation* category contributes more than half of the $F$ score, exerting the largest influence. This is quite acceptable, as it corresponds to the expectation that the purpose of an RDE is to provide appropriate tools and abstractions that facilitate application development. MARIE and Pyro have the highest implementation scores (23 and 22), indicative of their wide selection of components (partially due to its interoperability with other RDEs) and advanced GUI capabilities. Pyro is assigned the third highest $F$ score (37) due to the *Infrastructure* category, which accounts for over 25% of the final score. ADE, which has a score of 18 in the implementation category, ends up with the highest $F$ score (43) due to the infrastructure category, as it the only RDE that earns a *well supported* value for each criterion therein.

Of particular note is the fact that the RDEs with the highest scores (Pyro, ADE, and MARIE) all have explicit interoperability interfaces with other systems. While providing a boost in total feature score, we also note that this makes them reliant, to some degree, on the availability of the other systems for certain features, in addition to potentially affecting their stability (in that changes to the other RDEs may impact their operation). We also note again that comparing RDEs in terms of $F$ alone does not provide a full picture of evaluation, leaving out aspects such as *system usability*, discussed next.

[9]Binary and ternary values range from 0 to 2 so as to not introduce a bias towards ternary criteria.

| | Feature Category | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Specification (6) | | Platform (6) | | Infrastructure (16) | | Implementation (30) | | Total (58) | |
| **RDE** | Score | % | Score | % | Score | % | Score | % | Score | % |
| **TB** | 5 | 83 | 1 | 17 | 0 | 0 | 10 | 33 | 16 | 28 |
| **AR** | 4 | 67 | 2 | 33 | 6 | 38 | 14 | 47 | 26 | 45 |
| **P/S** | 3 | 50 | 5 | 83 | 9 | 56 | 17 | 57 | 34 | 59 |
| **Py** | 5 | 83 | 5 | 83 | 5 | 31 | 22 | 73 | 37 | 64 |
| **C** | 4 | 67 | 5 | 83 | 8 | 50 | 10 | 33 | 27 | 47 |
| **ML** | 6 | 100 | 4 | 67 | 10 | 63 | 15 | 50 | 35 | 60 |
| **AD** | 6 | 100 | 3 | 50 | 16 | 100 | 18 | 60 | 43 | 74 |
| **Mi** | 5 | 83 | 3 | 50 | 9 | 56 | 14 | 47 | 31 | 53 |
| **MA** | 5 | 83 | 6 | 100 | 10 | 63 | 23 | 77 | 44 | 76 |

Table 4: The raw *Feature* score $F$ and %, broken down by categories from Section 4 for each RDE

As noted in Section 3, an appropriate set of criteria must be considered to serve as the basis for comparing RDEs. Not only do applications have markedly different characteristics that may impact the designer's choice of RDE, but users tend to have different needs and working styles. The *Usability* score $U$ attempts to address the practical aspects of RDE usage by adding criteria relevant for the actual implementation and execution of robotic architectures (in particular, the two classes of tasks described in the previous section), the results of which are summarized in Table 5.

| | Usability Category | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Installation (8) | | "Low-level" (12) | | "High-level" (14) | | Total (34) | |
| **RDE** | Score | % | Score | % | Score | % | Score | % |
| **TB**$^{\dagger}$ | 8 | 100 | 4 | 33 | *na* | *na* | 12 | 35 |
| **AR** | 7 | 88 | 7 | 58 | 4 | 29 | 18 | 53 |
| **P/S** | 8 | 100 | 9 | 75 | 11 | 79 | 28 | 82 |
| **Py** | 8 | 100 | 11 | 92 | 11 | 79 | 30 | 88 |
| **C** | 6 | 75 | 8 | 67 | 6 | 43 | 20 | 59 |
| **ML** | 3 | 38 | 8 | 67 | 13 | 93 | 24 | 71 |
| **AD** | 6 | 75 | 10 | 83 | 10 | 71 | 26 | 76 |
| **Mi** | 2 | 25 | 4 | 33 | 4 | 29 | 10 | 29 |
| **MA** | 3 | 38 | 4 | 33 | 10 | 71 | 17 | 50 |

Table 5: The raw *Usability* score $U$ and %, broken down by categories from Section 5 for each RDE (a $^{\dagger}$ indicates some criteria were not included).

Pyro, Player/Stage, ADE, and MissionLab have the highest usability scores (30, 28, 26, and 24, respectively). From a usability point of view, this indicates that each has fulfilled their stated purpose: Pyro is aimed at novice users for educational purposes, Player/Stage is a flexible and adaptable programming interface, ADE combines robotic development with a MAS infrastructure, while MissionLab provides military personnel with non-programming methods of controlling robots. On the other hand, the low score given to Miro can be attributed to its reliance on the ACE/TAO communication framework[10] and incomplete documentation. In addition, it is necessary to point out that both ARIA and MissionLab's scores would be higher if the restriction to open-source components was lifted.

It is interesting to note that MissionLab and MARIE have the widest discrepancy in score between usability categories, each scoring relatively highly for "high-level" usability but low on the "low-level" architecture. Personal experience determined that much of the difference can be attributed to predefined components (both their number and usage) and their integration into a cohesive user interface. Both provide a comprehensive graphical method for connecting components, but suffer from either not providing a graphical interface to all parts of the system (e.g., MARIE requires shell scripts for startup/shutdown and the definition of communication channels) or by their orientation to-

[10]The impact of ACE/TAO is acknowledged in the user manual thusly: "The CORBA environment and the Miro framework seem to raise the bar for an easy entry into robot programming. While this can hardly be denied they facilitate tremendously the task of writing distributed programs."

wards very high-level tasks.

| Application Area | TB | AR† | P/S | Py | C | ML | AD | Mi | MA |
|---|---|---|---|---|---|---|---|---|---|
| **SLAM** | | | √ | | √ | | | | √ |
| **Planning/Navigation** | | | | | | | √ | √ | √ |
| **Learning** | | | √ | √ | √ | √ | | | √ |
| **Hierarchical Behavior** | √ | | | | | √ | √ | | √ |
| **Education** | √ | | √ | √ | | | | √ | |
| **HRI – Task Allocation** | | | √ | √ | | √ | √ | | |
| **HRI – Learning** | | | | | | | | | |
| **HRI – Assistive Robotics** | | | | | √ | | √ | √ | √ |
| **Multi-robot Sensing** | | | √ | | | | √ | | √ |
| **Multi-robot Exploration** | | | √ | | | | | | |
| **Multi-robot Mapping** | | | √ | | | | | | |
| **Multi-robot Localization** | | | √ | | | | | √ | |
| **Multi-robot Planning** | | | √ | | | √ | | | |
| **Multi-robot Coordination** | | | √ | | √ | √ | √ | √ | √ |
| **Multi-robot Formation** | | | √ | | | √ | | √ | |
| **Multi-robot Task Allocation** | | | √ | | | √ | | | |
| **Research areas** (out of 16) | 2 | 0† | 12 | 3 | 4 | 7 | 6 | 6 | 7 |
| **Interoperability facilities** | 0 | 2 | 4 | 0 | 1 | 0 | 0 | 0 | 0 |
| **Total score** (out of 20) | 2 | 2† | 16 | 3 | 5 | 7 | 6 | 6 | 7 |
| **Total %** | 10 | 10† | 80 | 15 | 25 | 35 | 30 | 30 | 35 |

Table 6: The raw *Impact* score $I$ and % for each RDE, the sum of "Application Area References" citations from Section 2 and the other RDEs that provide interoperability interfaces (a † indicates some criteria were not included).

Finally, an oblique way to determine the strengths of an RDE is to establish an *Impact* score $I$ that reflects the influence it has had on the robotics field. The number of research areas in which there are publications serve as an indicator of successful usage, as does the recognition that widely used systems are most likely to have other RDEs provide interoperability interfaces. Table 6 summarizes the robotics research subareas and citations given in Section 2 for each RDE, in addition to giving a count of the number of RDEs that interoperate with it. We reiterate that a single publication is used to satisfy research in any subarea, simply to provide an idea of the breadth of research areas in which it has been used; we also note the likelihood that some relevant publications were not included, as the particular RDE used is sometimes not mentioned in a publication. Because criteria are all either binary in nature or a simple count, total scores are a simple sum of items, deviating slightly from the previous convention of assigning 2 points to a $\surd$. Player/Stage clearly has had the largest impact, reflected by the fact that its score (16) is more than double that of the next highest system. It is also necessary to point out that ARIA's impact score is deceptively low (indicated by the † symbol), due to the fact that the authors were asked not to include references to its ancestral software.

Two overall comparison scores $T_\dagger$ and $T_{all}$ are shown in Table 7, where $T_\dagger$ does not include criteria that were unevaluated for any RDE and $T_{all}$ does. Overall scores are the sum of the feature $F$, usability $U$, and impact $I$ scores. Player/Stage has the highest total score $T_{all}$ (78 out of a possible 112), partially due to having the highest $I$ score, even though it had the fourth highest $F$ score and second highest $U$ score. The next three highest scoring RDEs (ADE with 75, which had the highest $F$ score; Pyro with 70, which had the highest $U$ score; and MARIE with 68, which had the second highest $F$ score) are all relatively new systems; in addition to providing some level of interoperability interfaces with other RDEs (thus capitalizing on prior innovations), we believe that part of their score can be attributed to identifying areas of application development that can be improved, partially based on the examples of already established RDEs (discussed in more depth in the next section). Of note is that when the unevaluated criteria are removed, the top four RDEs (ADE, Pyro, Player/Stage, and MARIE, respectively) remain the same.

We reiterate that the total scores $T_\dagger$ and $T_{all}$ may not reflect the particular requirements of a particular person or group and that while the evaluations here are comprehensive, they necessarily miss some criteria that may be important for a specific designer or application. Such items can be added at will to further refine and customize the evaluations, adjusting the evaluation formula given earlier.

| | Score | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Removed † Criteria | | | | | | | | All Criteria | | | | | | | |
| | $F$ (58) | | $U$ (20) | | $I$ (4) | | Total (82) | | $F$ (58) | | $U$ (34) | | $I$ (20) | | Total (112) | |
| **RDE** | Raw | % | Raw | % | Raw | % | Raw | % | Raw | % | Raw | % | Raw | % | Raw | % |
| **TB**† | 16 | 28 | 12 | 60 | 0 | 0 | 28 | 34 | 16 | 28 | 12 | 35 | 2 | 10 | 30 | 27 |
| **AR**† | 26 | 45 | 14 | 70 | 2 | 50 | 42 | 51 | 26 | 45 | 18 | 53 | 2 | 10 | 46 | 41 |
| **P/S** | 34 | 59 | 17 | 85 | 4 | 100 | 55 | 67 | 34 | 59 | 28 | 82 | 16 | 80 | 78 | 70 |
| **Py** | 37 | 64 | 19 | 95 | 0 | 0 | 56 | 68 | 37 | 64 | 30 | 88 | 3 | 15 | 70 | 63 |
| **C** | 27 | 47 | 14 | 70 | 1 | 25 | 42 | 51 | 27 | 47 | 20 | 59 | 5 | 25 | 52 | 46 |
| **ML** | 35 | 60 | 11 | 55 | 0 | 0 | 46 | 56 | 35 | 60 | 24 | 71 | 7 | 35 | 66 | 59 |
| **AD** | 43 | 74 | 16 | 80 | 0 | 0 | 59 | 72 | 43 | 74 | 26 | 76 | 6 | 30 | 75 | 67 |
| **Mi** | 31 | 53 | 6 | 30 | 0 | 0 | 37 | 45 | 31 | 53 | 10 | 29 | 6 | 30 | 47 | 42 |
| **MA** | 44 | 76 | 7 | 35 | 0 | 0 | 51 | 62 | 44 | 76 | 17 | 50 | 7 | 35 | 68 | 61 |

Table 7: The *Total* comparison score $T$ and % for each RDE, comprised of the sum of feature $F$, usability $U$, and impact $I$ scores. The left-hand columns under the **Removed † Criteria** heading do not include criteria unevaluated for any RDE, while columns under the **All Criteria** include all criteria, using a value of zero for unevaluated items.

## 6.2 RDE Maintainers and Developers

The selected RDEs in this survey are, as defined by the constraints of system selection, open source projects. While their availability is of obvious benefit to users, individual RDE maintainers can also potentially reap some benefit by examining other systems. Hopefully, this will facilitate the transfer of techniques and tools (e.g., Vaughan et al., 2003; Montemerlo et al., 2003b; Hattig et al., 2003; RADISH, 2004) across environments and promote progress in the field of robotics as a whole. Using the *Feature* and *Usability* comparison scores from the previous section as a basis (the *Impact* score is not considered, as it is not directly controlled by RDE maintainers), it becomes possible to not only make specific improvement suggestions, but also to make some high-level points. We note here that no suggestions are made for TeamBots because it is no longer under active development.

To begin, we examine the *Feature* scores shown in Table 4 by discussing each category. In the *Specification* category, all RDEs score at least 50%, while 8 of the 9 score 67% or higher, indicating that all supply adequate support for application design. Considering the *Platform* category, only two systems score less than 50%: TeamBots, which is no longer being actively developed, and ARIA, which has been developed in support of a proprietary platform and thus has unique objectives. Increasing the hardware support in MissionLab, ADE, and Miro would yield scores of 67% or higher for all of the remaining systems, such that all could be considered to have adequate platform support. In terms of the *Implementation* category, only CARMEN and MissionLab score below 50%. Recalling the information found in Table 2, it is evident that the score is substantially due to supporting a limited number of predefined components (although it is important to note that some components are available with MissionLab if licensed). ARIA's score is similarly affected by licensing issues, in that an integrated GUI is available; inclusion would put its score at about 60%. ADE, Miro, and MARIE all have inadequate documentation, which would improve their *Feature* score, while also increasing their *Usability* scores. The last category factoring into the feature score is *Infrastructure*, discussed very briefly here due to its inclusion in Section 7. ARIA, Pyro, and CARMEN all score 50% or below; again, ARIA's licensing requirements affect its score to some degree, leading to a *not supported* value for the *Monitoring and Management* criterion. The most prevalent unsupported criterion is *Security*, which only ARIA and ADE support at all. As noted earlier, Player/Stage, Miro, and MARIE all have the potential to easily incorporate security (Player/Stage by utilizing its authentication mechanism and Miro/MARIE by leveraging the ACE/TAO framework). The next least supported criteria are *Component Mobility* and *Fault-tolerance*, related to *Distribution Mechanisms*. Suggestions for improvements in these areas is beyond the scope of this survey, and we again refer to Section 7 for more.

Three categories make up the *Usability* score, *Installation*, the "low-level" architecture, and "high-level" subarchitectures. Three RDEs are at or below 50% in installation (MissionLab, Miro, and MARIE), two in the "low-level" category (Miro and MARIE), and three in the "high-level" category (ARIA, CARMEN, and Miro). Discounting unavailable software, ARIA's individual scores are well distributed among criteria, indicating that while each could be improved, a good overall mix is established. MissionLab's installation score is low due to its reliance on older versions of `gcc`; a new release would most likely greatly improve its score. As with the *Implementation* category mentioned

above, CARMEN would benefit greatly from additional predefined components. Miro's framework, relying as it does on CORBA, has great potential in terms of usability; however, additional tools and documentation are required to "lower the bar" that, as they say in their user manual, has been placed quite high. In MARIE's case, the lowest individual criteria scores are concentrated in the "low-level" architecture. In particular, the necessity of manually specifying component connections lowers usability, as does what might be considered a high learning curve.

From the above, one broad point that should be clear is that interoperability tools are of great utility, allowing one RDE to incorporate any component functionality developed in another RDE. Interoperability has been discussed in the literature (e.g., Baum et al., 2002; Nesnas et al., 2003; Côté et al., 2005), and while no single technique has been accepted, MARIE's foundations suggest a direction for future standards. While Pyro provides a large base of interoperability tools, MARIE's stated intention is to provide a well-grounded framework that is not only compatible with existing systems, but also provides the conceptual basis for adding interoperability in the future. The benefits of this approach are apparent when considering the available pre-defined component list; components available in Player/Stage, ARIA, and CARMEN are also available in Pyro and MARIE.

Finally, to gain acceptance, an RDE must pay attention to usability and the quality of its user and developer interfaces (see Steinfeld, 2004 for a general treatment). An area that is receiving increasing attention is robotics education. In addition to opening up the field to new people, an RDE that caters to novices should, almost by definition, promote usability. Pyro is particularly strong in this respect. Another aspect of usability concerns those who are not interested in going beyond the functionality already provided by an RDE, but rather use the already established components to implement their own applications without programming. In this sense, the FlowDesigner package (related to MARIE) provides a graphical method for defining data flow throughout an application. Taking this a step further, MissionLab provides graphical tools not concerned with robot particulars at all, but only with their actions. Additionally, the MissionLab developers have conducted many usability studies (e.g., MacKenzie & Arkin, 1998; Collins et al., 2000; Endo et al., 2004; Moshkina et al., 2006) in conjunction with system development.

# 7 Conclusion & Outlook

This survey has evaluated nine RDEs with respect to an extensive set of relatively common criteria supporting the development of robotic applications. Results were then compiled and used to compare the systems according to three types of score (*Feature*, *Usability*, and *Impact*), providing robotic architecture designers with information useful in picking an RDE for themselves. Finally, the comparisons provided the foundation for suggesting potential areas of improvement to RDE maintainers based on features currently found in other systems. In conclusion, we extrapolate from the results and attempt to identify some likely future trends.

The comparison of different RDEs suggests that common features will increasingly be expected in all systems, strengthened by the interoperability mechanisms found in some recent systems (e.g., Pyro and MARIE). In addition to creating a set of (possibly *de facto*) standards, this will lead to an increasing number of predefined components that can be expected in any given RDE. Furthermore, we expect the list of predefined components given in Section 3 to continue to expand, both in relation to high-level functionality (e.g., various types of robot control) and more specific low-level functionality (e.g., "vision processing" will split into separate categories such as monocular vs. stereo vision processing). We feel that a similar trend will develop in relation to RDE *infrastructure* (see Section 3), such that users expect inclusion of a suite of tools that implement various non-architectural functions. This suspicion is borne out by a cursory examination of the origination of RDEs. Early systems (e.g., TeamBots, 1998) provide little in the way of infrastructure: an application in TeamBots is the sum of the Java classes that implement it. Player/Stage (2001) incorporates a minimal amount of infrastructure; the authors acknowledge and deliberately reject this trend, making the system "free from the computational and programmatic overhead that is generally associated with the practical application" of such mechanisms (Gerkey et al., 2003). More recent RDEs, such as MARIE (2004) and ADE (2004), explicitly incorporate substantial infrastructure into their design and use, with the stated aims, respectively, of improving interoperability and distribution.

The necessity of providing infrastructural interoperability and distribution is illustrated by a quote from the authors of the GRACE project (Simmons et al., 2003): "One of the more difficult parts of the Challenge for us was determining how to integrate a vast amount of software that had been developed by the participating institutions, mostly on different hardware platforms." Such mechanisms should immediately bring to mind multi-agent systems (MAS) research, which has found particular traction in the robotics field in the form of multi-robot applications (such as the citations listed at the bottom of Table 6; see also K. P. Sycara & Zeng, 1996; Altmann et al., 2001; Dias & Stentz, 2003; Gerkey

& Matarić, 2004). We suggest that it will be critical for future RDEs to incorporate other aspects of MAS research, including, but not limited to security (e.g., Singh & Sycara, 2004) and system-wide management facilities (such as those discussed in Bellifemine et al., 1999; K. Sycara et al., 2003).

A final trend we expect to take shape in RDEs in the future is the prominent promotion of *autonomic computing* functionality (e.g., Bantz et al., 2003). On the one hand, we expect improvements in low-level system characteristics that are transparent to users such as *fault tolerance* (e.g., Varakantham et al., 2002; Long et al., 2003; Melchior & Smart, 2004). ADE, for example, already explicitly incorporates features for monitoring, relocating, and restarting of components integrated into its infrastructure. Moreover, MARIE and Miro can potentially take advantage of recent advances in middleware, e.g., the Fault Tolerant CORBA specification (see Chapter 23 in CORBA, 2005), which incorporates mechanisms that promote robust system operation. On the other hand, we expect that development of high-level AI techniques that enhance a robot's apparent intelligence will increasingly find inclusion in RDEs, like the tools found in MissionLab. We expect that robot learning (e.g., R. Russell, 2004; Blank et al., 2005), findings from human-robot interaction (HRI) research (e.g., Salter et al., 2005; Fong et al., 2006; Moshkina et al., 2006; Scheutz et al., 2006), and the study of social robotics (e.g., Bruce et al., 2002; Breazeal, 2003) will become commonplace.

In sum, we believe that the increase in capability of robotic applications will soon require extensive infrastructure support, with expanding development of support for autonomic computing in the future. Such tools and techniques will be used not only for the development and debugging of robotic architectures, but also for the execution and maintenance of robotic architectures as part of application deployment. If true, the choice of one RDE over another will be based on more than just the development support offered, but increasingly on the features it provides for the long-term operation of robotic applications. Furthermore, and perhaps more significantly, the integration of system infrastructure with the development of intelligent robotic architectures will lead to robots that display ever greater levels of autonomy.

# References


*Activmedia robotics mobilerobots developer support.* (2005). http://robots.mobilerobots.com/.

Altmann, J., Gruber, F., Klug, L., Stockner, W., & Weippl, E. (2001, June). Using mobile agents in real world: A survey and evaluation of agent platforms. In T. Wagner (Ed.), *Proceedings of the second international workshop on infrastructure for agents, MAS, and scalable MAS at the 5th international conference on autonomous agents* (pp. 33–39). Montreal, Canada: ACM Press.

Andronache, V., & Scheutz, M. (2004a). ADE - a tool for the development of distributed architectures for virtual and robotic agents. In *Proceedings of the 4th international symposium "from agent theory to agent implementation"*.

Andronache, V., & Scheutz, M. (2004b). Integrating theory and practice: The agent architecture framework APOC and its development environment ADE. In *Proceedings of AAMAS 2004*.

Arkin, R., & Balch, T. (1997). AuRA: principles and practice in review. *JETAI*, *9*(2-3), 175-189.

Arkin, R., Collins, T., & Endo, T. (1999, Sept). Tactical mobile robot mission specification and execution. *Mobile Robots XIV*, 150–163.

Arkin, R., Endo, Y., Lee, B., MacKenzie, D., & Martinson, E. (2003, March). Multistrategy learning methods for multirobot systems. In *Proceedings of the 2nd international workshop on multi-robot systems* (pp. 137–150). Washington, D.C.

Austin, D. (2004). *Dave's robotic operating system.* http://dros.org/.

Balch, T. (2000). Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, *8*(3), 209–238.

Balch, T. (2002). *Teambots proposal.* http://www.cs.cmu.edu/ trb/robocupjr/.

Balch, T. (2004). *Teambots.* http://www.teambots.org/.

Balch, T., & Arkin, R. (1999). Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, *20*(5).

Balch, T., & Ram, A. (1998). Integrating robotics research with JavaBots. In *Working notes of the AAAI 1998 spring symposium.*

Bantz, D., Bisdikian, C., Challener, D., Karidis, J., Mastrianni, S., Mohindra, A., et al. (2003). Autonomic personal computing. *IBM Systems Journal*, *42*(1), 165–176.

Baum, W., Bredenfeld, A., Hans, M., Hertzberg, J., Ritter, A., Schönherr, F., et al. (2002, June). Integrating hetero-

geneous robot and software components by agent technology. *Robotik 2002 Leistungsstand - Anwendungen - Visionen - Trends*, 655-660.

Beaudry, E., Brosseau, Y., Côté, C., Raïevsky, C., Létourneau, D., Kabanza, F., et al. (2005). Reactive planning in a motivated behavioural architecture. In *Proceedings american association for artificial intelligence conference* (pp. 1242–1247).

Bellifemine, F., Poggi, A., & Rimassa, G. (1999, April). JADE - a FIPA-compliant agent framework. In *Proceedings of the 4th international conference and exhibition on the practical application of intelligent agents and multi-agents* (pp. 97–108). London.

Bentivegna, D., & Atkeson, C. (2002). *Learning how to behave from observing others.* (SAB02 Workshop on Motor Control in Humans and Robots: on the interplay of real brains and artificial devices)

Bergbreiter, S., & Pister, K. (2003, Oct). CotsBots: An off-the-shelf platform for distributed robotics. In *Proceedings of 2003 IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)* (Vol. 3, pp. 1632–1637).

Bergbreiter, S., & Pister, K. (2005). *CotsBots: An off-the-shelf distributed robot platform.* http://www-bsac.eecs.berkeley.edu/projects/cotsbots/.

Biggs, G., & MacDonald, B. (2003, December). A survey of robot programming systems. In *Proceedings of the australasian conference on robotics and automation.* Brisbane, Australia.

Bitting, E., Carter, J., & Ghorbani, A. (2003, May). Multiagent systems development kits: An evaluation. In *Proceedings of the 1st annual conference on communication networks and services research (CNSR 2003)* (pp. 101–107). Moncton, Canada.

Blank, D., Kumar, D., & Meeden, L. (2002). A developmental approach to intelligence. In S. Conlon (Ed.), *Proceedings of the thirteenth annual midwest artificial intelligence and cognitive science society conference.*

Blank, D., Kumar, D., Meeden, L., & Marshall, J. (2005). Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture. *Cybernetics and Systems*, *36*(2).

Blank, D., Kumar, D., Meeden, L., & Yanco, H. (2003). Pyro: A python-based versatile programming environment for teaching robotics. *Journal on Educational Resources in Computing*, *3*(4), 1–15.

Blank, D., Kumar, D., Meeden, L., & Yanco, H. (To appear). Pyro: A python-based versatile programming environment for teaching robotics. *ACM Journal on Educational Resources in Computing (JERIC).*

Breazeal, C. (2003). Towards sociable robots. *Robotics and Autonomous Systems*, *42*(3-4), 167–175.

Brooks, A., Kaupp, T., Makarenko, A., Oreb ack, A., & Williams, S. (2005). Towards component-based robotics. In *IEEE/RSJ international conference on intelligent robots and systems (IROS 2005)* (pp. 163–168).

Brooks, R. (1990). *The behavior language: User's guide* (Tech. Rep. No. AIM-1227). Massachusettes Institute of Technology.

Brooks, R. (1991). Intelligence without representation. *Artificial Intelligence Journal*, *47*, 139–159.

Bruce, A., Nourbakhsh, I., & Simmons, R. (2002). The role of expressiveness and attention in human-robot interaction. In *Proceedings of the IEEE international conference on robotics and automation.*

Bruyninckx, H. (2001). Open robot control software: the OROCOS project. In *Proceedings of IEEE international conference on robotics and automation (ICRA) 2001* (Vol. 3, pp. 2523–2528).

Bruyninckx, H. (2005). *The OROCOS project.* http://www.orocos.org/.

Chaimowicz, L., Cowley, A., Sabella, V., & Taylor, C. (2003, Oct). ROCI: A distributed framework for multi-robot perception and control. In *Proceedings of 2003 IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)* (Vol. 3, pp. 266–271).

*The CMU sphinx group open source speech recognition engines.* (2004). http://cmusphinx.sourceforge.net/html/cmusphinx.php. The Sphinx Group at Carnegie Mellon University.

Collins, T., Arkin, R., Cramer, M., & Endo, Y. (2000, July). Field results for tactical mobile robot missions. In *Unmanned systems 2000.* Orlando, FL.

*Common object request broker architecture (CORBA/IIOP).* (2005). http://www.omg.org/technology/documents/corba_spec_catalog.htm Object Management Group.

Côté, C. (2005). *Mobile and autonomous robotics integration environment (MARIE).* http://marie.sourceforge.net/.

Côté, C., Brosseau, Y., Létourneau, D., Raievsky, C., & Michaud, F. (2006). Robotic software integration using MARIE. *International Journal on Advanced Robotics Systems*, *3*(1), 55–60.

Côté, C., Létourneau, D., Michaud, F., & Brosseau, Y. (2005). *Software design patterns for robotics: Solving integration problems with MARIE.* Submitted for workshop to ICRA2005.

Côté, C., Létourneau, D., Michaud, F., Valin, J., Brosseau, Y., Raievsky, C., et al. (2004). Programming mobile robots

using RobotFlow and MARIE. In *Proceedings IEEE/RSJ international conference on robots and intelligent systems.*

Desai, M., & Yanco, H. (2005, August). Blending human and robot inputs for sliding scale autonomy. In *Proceedings of the 14th IEEE international workshop on robot and human interactive communication.* Nashville, TN.

Dias, M., & Stentz, A. (2003, Oct). A comparative study between centralized, market-based, and behavioral multirobot coordination approaches. In *Proceedings of 2003 IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)* (Vol. 3, pp. 2279–2284).

Eiter, T., & Mascardi, V. (2002). Comparing environments for developing software agents. *AI Communications*, *15*(4), 169–197.

Endo, Y., MacKenzie, D., & Arkin, R. (2004, May). Usability evaluation of high-level user assistance for robot mission specification. *IEEE Transactions on Systems, Man, and Cybernetics*, *34*(2), 168–180.

*ERSP 3.0 robotic development platform.* (2004). http://www.evolution.com/products/ersp/. Evolution Robotics.

Fay, R., Kaufmann, U., Schwenker, F., & Palm, G. (2004). Learning Object Recognition in a NeuroBotic System. In H. Groß, K. Debes, & H. Böhme (Eds.), *3rd workshop on selforganization of adaptive behavior SOAVE 2004* (pp. 198–209). Dusseldorf: VDI.

*The festival speech synthesis system.* (2004). http://www.cstr.ed.ac.uk/projects/festival/. Centre for Speech Technology Research.

*FIPA ACL message structure specification (SC00061G).* (2002). http://www.fipa.org/specs/fipa00061/. Foundation for Intelligent Physical Agents.

Fleury, S., Herrb, M., & Chatila, R. (1997). Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *International conference on intelligent robots and systems* (Vol. 2, p. 842-848). IEEE.

Fleury, S., & Mallet, A. (2004). *LAAS open software for autonomous systems.* http://softs.laas.fr/openrobots/tools/genom.php.

Fong, T., Kunz, C., Hiatt, L., & Bugajska, M. (2006). The human-robot interaction operating system. In *Proceedings of the ACM conference on human-robot interaction (HRI2006).* ACM.

Fong, T., Nourbakhsh, I., & Dautenhahn, K. (2003). A survey of socially interactive robots. *Robotics and Autonomous Systems*, *42*, 143–166.

Fredslund, J., & Matarić, M. (2002, Oct). A general, local algorithm for robot formations. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-Robot Systems*, *18*(5), 837–846.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Gassull, G. (2001). *Communication services and user interfaces for tele-operating mobile robots via the internet*. Master's thesis, University of Barcelona and University of Ulm, Neuroinformatics.

Gerkey, B., Howard, A., & Vaughan, R. (2005). *Player/Stage.* http://playerstage.sourceforge.net/.

Gerkey, B., & Matarić, M. (2002, Oct). Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-Robot Systems*, *18*(5), 758–768. (Also Technical Report IRIS-01-399)

Gerkey, B., & Matarić, M. (2004). Are (explicit) multi-robot coordination and multi-agent coordination really so different? In *Proceedings of the AAAI spring symposium on bridging the multi-agent and multi-robotic research gap* (pp. 1–3).

Gerkey, B., Vaughan, R., & Howard, A. (2003, June). The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics* (pp. 317–323). Coimbra, Portugal.

Gerkey, B., Vaughan, R., Støy, K., Howard, A., Sukhatme, G., & Matarić, M. (2001, October). Most valuable player: A robot device server for distributed control. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems* (pp. 1226–1231). Wailea, Hawaii.

Guilbert, N., Beauregard, M., Michaud, F., & Lafontaine, J. de. (2003). Emulation of collaborative driving systems using mobile robots. In *Proceedings ieee conference on systems, man, and cybernetics* (pp. 856–861).

Hattig, M., Horswill, I., & Butler, J. (2003, Oct). Roadmap for mobile robot specifications. In *Proceedings of 2003 IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)* (Vol. 3, pp. 2410–2414).

Heckel, F. (2005). *ROLE robotics development environment.* http://www.cse.wustl.edu/ fwph/role/.

Hoff, J., & Bekey, G. (1995). An architecture for behavior coordination learning. In *IEEE international conference on neural networks.*

Horswill, I. (2000). Functional programming of behavior-based systems. *Autonomous Robots*, *9*(1), 83–93.

Howard, A., Matarić, M., & Sukhatme, G. (2002). An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots Special Issue on Intelligent Embedded Systems*, *13*(2), 113–126.

Howard, A., Matarić, M., & Sukhatme, G. (2003, Sep). Putting the 'I' in 'Team': An ego-centric approach to cooperative localization. In *IEEE international conference on robotics and automation* (pp. 868–892). Taipei, Taiwan.

Howard, A., Parker, L., & Sukhatme, G. (2004, Jun). The SDR experience: Experiments with a large-scale heterogenous mobile robot team. In *9th international symposium on experimental robotics 2004.* Singapore.

Howard, A., & Roy, N. (2004). *Robotics data set repository (RADISH).* http://radish.sourceforge.net/index.php.

Jensen, R., & Veloso, M. (1998, Oct). Interleaving deliberative and reactive planning in dynamic multi-agent domains. In *Proceedings of the AAAI fall symposium on integrated planning for autonomous agent architectures.* AAAI Press.

*JESS - the expert system shell for the java platform.* (2003). http://herzberg.ca.sandia.gov/jess/. Sandia National Laboratories.

Jia, J., Chen, W., & Xi, Y. (2004). Design and implementation of an open autonomous mobile robot system. In *Proceedings of IEEE international conference on robotics and automation (ICRA) 2004* (Vol. 2, pp. 1726–1731).

Jones, C., & Matarić, M. (2004, Sep). Automatic synthesis of communication-based coordinated multi-robot systems. In *IEEE/RSJ international conference on intelligent robots and systems* (pp. 381–387). Sendai, Japan.

Jung, B., & Sukhatme, G. (2002, Nov). Tracking targets using multiple robots: The effect of environment occlusion. *Autonomous Robots*, *13*(3), 191–205.

Kaupp, T. (2005). *Orca robotics.* http://orca-robotics.sourceforge.net/.

Koker, R., Oz, C., Cakar, T., & Ekiz, H. (2004, December). A study of neural network based inverse kinematics solution for a three-joint robot. *Robotics and Autonomous Systems*, *49*(3-4), 227-234.

Konolige, K. (1997). COLBERT: A language for reactive control in saphira. In *Proceedings of the german conference on artificial intelligence* (pp. 31–52). Freiburg, Germany.

Konolige, K. (2000). A gradient method for realtime robot control. In *Proceedings of the IEEE/RSJ international conference on intelligent robotic systems (IROS).*

Konolige, K. (2002, April). *Saphira robot control architecture* (Tech. Rep.). Menlo Park, CA: SRI International.

Konolige, K., Myers, K., Ruspini, E., & Saffiotti, A. (1997). The Saphira architecture: A design for autonomy. *Journal of experimental & theoretical artificial intelligence: JETAI*, *9*(1), 215–235.

Kraetzschmar, G., Gassull, G., & Uhl, K. (2004, July). Probabilistic quadtrees for variable-resolution mapping of large environments. In M. I. Ribeiro & J. Santos Victor (Eds.), *Proceedings of the 5th IFAC/EURON symposium on intelligent autonomous vehicles.*

Kraetzschmar, G., Sablatnög, S., Enderle, S., Utz, H., Simon, S., & Palm, G. (2000, October). Integration of Multiple Representations and Navigation Concepts on Autonomous Mobile Robots. In H. Groß, K. Debes, & H. Böhme (Eds.), *Workshop SOAVE-2000: Selbstorganisation von adaptivem verhalten* (Vol. 10/643). Ilmenau, Germany: VDI Verlag.

Kramer, J., & Scheutz, M. (2003). GLUE - a component connecting schema-based reactive to higher-level deliberative layers for autonomous agents. In R. Weber (Ed.), *Proceedings of the 16th international FLAIRS conference* (p. 22-26). AAAI Press.

Labonté, D., Michaud, F., Boissy, P., Corriveau, H., Cloutier, R., & Roux, M. (2005). *Evaluation methodology of user interfaces for teleoperated mobile robots in home environments.* (Submitted to IEEE International Conference on Robotics and Automation)

LaFary, M., & Newton, C. (2005, October). *Aria html documentation.*

Laukkanen, M. (1999). *Evaluation of FIPA-compliant agent platforms*. Unpublished master's thesis, Lappeenranta University of Technology.

*LEGO.com educational division – mindstorms for schools.* (2005). http://www.lego.com/eng/education/mindstorms/default.asp. LEGO.

Lemay, M., Michaud, F., Létourneau, D., & Valin, J. (2004). Autonomous initialization of robot formations. In *IEEE international conference on robotics and automation.*

Lindstrom, M., Orebäck, A., & Christensen, H. (2000, April). BERRA: A research architecture for service robots. In *Proceedings of international conference on robotics and automation (ICRA)* (Vol. 4, pp. 3278–3283).

Logan, B. (1998). Classifying agent systems. In B. Logan & J. Baxter (Eds.), *Proceedings of AAAI-98 conference*

*workshop on software tools for developing agents.* Menlo Park, California: American Association for Artificial Intelligence.

Long, M., Murphy, R., & Parker, L. (2003, October). Distributed multi-agent diagnosis and recovery from sensor failures. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, *3*, 2506–2513.

Lucas, G. (2004). *The rossum project.* http://rossum.sourceforge.net/.

MacDonald, B., Yuen, D., Wong, S., Woo, E., Gronlund, R., Collett, T., et al. (2003, September). Robot programming environments. In *ENZCon2003 10th electronics New Zealand conference.* University of Waikato, Hamilton.

MacKenzie, D., & Arkin, R. (1993, Nov). Formal specification for behavior-based mobile robots. *Mobile Robots VIII*, 94–104.

MacKenzie, D., & Arkin, R. (1998, April). Evaluating the usability of robot programming toolsets. *The International Journal of Robotics Research*, *17*(4), 381–401.

MacKenzie, D., Arkin, R., & Cameron, J. (1997). Multiagent mission specification and execution. *Autonomous Robots*, *4*(1), 29–52.

Maes, P. (1990). Situated agents can have goals. In P. Maes (Ed.), *Designing autonomous agents* (pp. 49–70). MIT Press.

Mallet, A., Fleury, S., & Bruyninckx, H. (2002). A specification of generic robotics software components: future evolutions of GenoM in the Orocos context. In *International conference on intelligent robotics and systems.* IEEE.

Matarić, M. (2004, Mar). Robotics education for all ages. In *Proceedings, AAAI spring symposium on accessible, hands-on AI and robotics education.*

Mayfield, J., Labrou, Y., & Finin, T. (1996). Evaluation of KQML as an agent communication language. In M. Wooldridge, J. P. Müller, & M. Tambe (Eds.), *Proceedings on the IJCAI workshop on intelligent agents II: Agent theories, architectures, and languages* (Vol. 1037, pp. 347–360). Springer-Verlag.

Melchior, N., & Smart, W. (2004). A framework for robust mobile robot systems. In D. W. Gage (Ed.), *Proceedings of SPIE: Mobile robots XVII* (Vol. 5609).

Metta, G., Fitzpatrick, P., & Natale, L. (2006). YARP: Yet another robot platform. *International Journal on Advanced Robotics Systems*, *3*(1), 43–48.

Michaud, F. (2005). *Engineering education and the design of intelligent mobile robots for real use.* (Submitted to International Journal of Intelligent Automation and Soft Computing, Special Issue on Global Look at Robotics Education)

Michaud, F., & Létourneau, D. (2004). *Robotflow: Open source robotics toolkit for flowdesigner.* http://robotflow.sourceforge.net/.

Michel, O. (2004). Webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, *1*(1), 39–42.

*Miro - middleware for robots.* (2005). http://smart.informatik.uni-ulm.de/MIRO/index.html. Robotics Group, University of Ulm.

*Missionlab v6.0.* (2003). http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab/. Mobile Robot Laboratory.

*Modular controller architecture.* (2005). http://mca2.sourceforge.net/.

Montemerlo, M., Roy, N., & Thrun, S. (2003a). *CARMEN, Carnegie Mellon Robot Navigation Toolkit.* http://carmen.sourceforge.net/.

Montemerlo, M., Roy, N., & Thrun, S. (2003b). Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit. In *IROS 2003* (Vol. 3, p. 2436-2441). Las Vegas, NV.

Moshkina, L., & Arkin, R. (2003). On TAMEing robots. In *IEEE international conference on systems, man and cybernetics* (Vol. 4, pp. 3949–3959).

Moshkina, L., Endo, Y., & Arkin, R. (2006). Usability evaluation of an automated mission repair mechanism for mobile robot mission specification. In *Proceedings of the ACM conference on human-robot interaction (HRI2006).* ACM.

Nesnas, I., Simmons, R., Gaines, D., Kunz, C., Diaz-Calderon, A., Estlin, T., et al. (2006). CLARAty: Challenges and steps toward reusable robotic software. *International Journal on Advanced Robotics Systems*, *3*(1), 23–30.

Nesnas, I., Wright, A., Bajracharya, M., Simmons, R., & Estlin, T. (2003). CLARAty and challenges of developing interoperable robotic software. In *Proceedings of 2003 IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)* (Vol. 3, pp. 2428–2435).

Nguyen, G., Dang, T., Hluchy, L., Balogh, Z., Laclavik, M., & Budinska, I. (2002, Jun). *Agent platform evaluation and comparison* (Tech. Rep.). Bratislava, Slovakia: Pellucid 5FP IST-2001-34519.

Nowostawski, M., Bush, G., Purvis, M., & Cranefield, S. (2000). Platforms for agent-oriented software engineering. In J. Dong, J. He, & M. Purvis (Eds.), *Proceedings of APSEC 2000* (pp. 480–488). IEEE Computer Society Press.

Orebäck, A., & Christensen, H. (2003). Evaluation of architectures for mobile robotics. *Autonomous Robots*, *14*(1), 33–49.

Osentoski, S., Manfredi, V., & Mahadevan, S. (2004). Learning hierarchical models of activity. In *IEEE/RSJ international conference on robots and systems (IROS 2004).*

Pellom, B., & Hacioglu, K. (2003). Recent improvements in the CU SONIC ASR system for noisy speech: The SPINE task. In *Proceedings of IEEE international conference on acoustics, speech, and signal processing (ICASSP).*

Pfeifer, R. (1988). Artificial intelligence models of emotion. In V. Hamilton, G. H. Bower, & N. H. Frijda (Eds.), *Cognitive perspectives on emotion and motivation, volume 44 of series d: Behavioural and social sciences* (pp. 287–320). Netherlands: Kluwer Academic Publishers.

Pineau, J., Montemerlo, M., Pollack, M., Roy, N., & Thrun, S. (2002). Towards robotic assistants in nursing homes: challenges and results. In T. Fong & I. Nourbakhsh (Eds.), *Workshop notes (WS8: Workshop on robot as partner: An exploration of social robots), IEEE international conference on robots and systems.* Lausanne, Switzerland: IEEE.

Poggi, A., Rimassa, G., & Turci, P. (2002). What agent middleware can (and should) do for you. *Applied Artificial Intelligence*, *16*(9-10), 677-698.

Provost, J., Kuipers, B., & Miikkulainen, R. (2004). Self-organizing perceptual and temporal abstraction for robot reinforcement learning. In *AAAI-04 workshop on learning and planning in markov processes.*

*Pyro, python robotics.* (2005). http://emergent.brynmawr.edu/pyro/?page=Pyro. Python Robotics.

Ricordel, P., & Demazeau, Y. (2000, Dec). From analysis to deployment: A multi-agent platform survey. In *Engineering societies in the agents world* (Vol. 1972, pp. 93–105). Springer-Verlag.

Rivard, F. (2005, June). *Localisation relative de robots mobiles opérant en groupe* (Tech. Rep.). Mémoire de maîtrise, Département de génie électrique et de génie informatique, Université de Sherbrooke.

Russell, R. (2004). Mobile robot learning by self-observation. *Autonomous Robots*, *16*(1), 81–93.

Russell, S., & Norvig, P. (2002). *Artificial intelligence: A modern approach* (2 ed.). Prentice Hall.

Salter, T., Michaud, F., Dautenhahn, K., Létourneau, D., & Caron, S. (2005). Recognizing interaction from a robot's perspective. In *Proceedings IEEE international workshop on robot and human interactive communication* (pp. 178–183).

Scheutz, M. (2004). *APOC - an architecture for the analysis and design of complex agents.* (Forthcoming In Darryl Davis, editor, *Visions of Mind*)

Scheutz, M. (2006, April). ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures. *Applied Artificial Intelligence*, *20*(4-5).

Scheutz, M., & Andronache, V. (2003). APOC - a framework for complex agents. In *Proceedings of the AAAI spring symposium* (pp. 18–25). AAAI Press.

Scheutz, M., & Andronache, V. (2004). Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems. *IEEE Transactions of System, Man, and Cybernetics Part B: Cybernetics*, *34*(6).

Scheutz, M., Andronache, V., Kramer, J., Snowberger, P., & Albert, E. (2004). Rudy: A robotic waiter with personality. In *Proceedings of AAAI robot workshop* (p. forthcoming). AAAI Press.

Scheutz, M., Schermerhorn, P., Kramer, J., & Middendorff, C. (2006). The utility of affect expression in natural language interactions in joint human-robot tasks. In *Proceedings of the ACM conference on human-robot interaction (HRI2006).* ACM.

Schmidt, D. (1994). The ADAPTIVE communication environment: An object-oriented network programming toolkit for developing communication software. In *12th annual sun users group conference* (pp. 214–225). San Francisco, CA.

Silva, A., Romao, A., Deugo, D., & Silva, M. da. (2001). Towards a reference model for surveying mobile agent systems. *Autonomous Agents and Multi-Agent Systems*, *4*, 187–231.

Simmons, R. (1994, Feb). Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, *10*(1), 34–43.

Simmons, R. (2004). *Inter process communication (IPC).* http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/TCA/www/ipc/.

Simmons, R., Apfelbaum, D., Fox, D., Goldmann, R., Haigh, K., Musliner, D., et al. (2000). Coordinated deployment of multiple heterogeneous robots. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS).*

Simmons, R., Goldberg, D., Goode, A., Montemerlo, M., Roy, N., Sellner, B., et al. (2003). GRACE: an autonomous robot for the AAAI robot challenge. *AI Mag.*, *24*(2), 51–72.

*Simplified wrapper and interface generator.* (2004). http://www.swig.org/.

Singh, R., & Sycara, K. (2004). *Securing multi agent societies* (Tech. Rep. No. CMU-RI-TR-04-02). Robotics Institute, Carnegie Mellon.

Skubic, M., & Volz, R. A. (1998). Learning force-based assembly skills from human demonstration for execution in unstructured environments. In *Proceedings of international conference on robotics and automation (ICRA98)* (pp. 1281–1288).

Sloman, A. (1998, July). What's an AI toolkit for? In B. Logan & J. Baxter (Eds.), *Proceedings of the AAAI-98 workshop on software tools for developing agents* (pp. 1–10).

Sloman, A. (2002, Feb). *Help poprulebase.*

Sloman, A., & Scheutz, M. (2002). A framework for comparing agent architectures. In *Proceedings of UK workshop on computational intelligence* (pp. 169–176).

*SOAP version 1.2.* (2003, June). http://www.w3.org/TR/soap12/. W3C XML Protocol Working Group.

Sprouse, J. (2005). *Nomadic.sourceforge.net.* http://nomadic.sourceforge.net/.

Steinfeld, A. (2004). Interface lessons for fully and semi-autonomous mobile robots. In *Proceedings of IEEE international conference on robotics and automation (ICRA) 2004* (Vol. 3, pp. 2752–2757).

Stentz, A. (2002, July). CD*: A real-time resolution optimal re-planner for globally constrained problems. In *Proceedings of AAAI 2002* (p. 605).

Sycara, K., Paolucci, M., Velsen, M. V., & Giampapa, J. (2003). The RETSINA MAS infrastructure. *Autonomous Agents and Multi-Agent Systems*, *7*(1), 29–48.

Sycara, K. P., & Zeng, D. (1996). Coordination of multiple intelligent software agents. *International Journal of Cooperative Information Systems*, *5*(2/3), 181-212.

Tews, A., Matarić, M., & Sukhatme, G. (2003, Sep). A scalable approach to human-robot interaction. In *IEEE international conference on robotics and automation* (p. 1665-1670). Taipei, Taiwan.

Thrun, S. (2003). Robotic mapping: A survey. In G. Lakemeyer & B. Nebel (Eds.), *Exploring artificial intelligence in the new millennium* (pp. 1–35). San Francisco, CA, USA: Morgan Kaufmann.

Thrun, S., Fox, D., Burgard, W., & Dellaert, F. (2000). Robust monte carlo localization for mobile robots. *Artificial Intelligence*, *128*(1-2), 99–141.

Utz, H., Kraetzschmar, G., Mayer, G., & Palm, G. (2005, August). Hierarchical behavior organization. In *Proceedings of IROS 2005.* Edmonton, Canada.

Utz, H., Sablatnög, S., Enderle, S., & Kraetzschmar, G. (2002, August). Miro – middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, *18*(4), 493–497.

Utz, H., Stulp, F., & Mühlenfeld, A. (2004). Sharing belief in teams of heterogeneous robots. In D. Nardi, M. Riedmiller, & C. Sammut (Eds.), *RoboCup-2004: The eighth RoboCup competitions and conferences.* Springer Verlag.

Valin, J., & Létourneau, D. (2004). *Flowdesigner.* http://flowdesigner.sourceforge.net/.

Varakantham, P., Gangwani, S., & Karlapalem, K. (2002). On handling component and transaction failures in multi agent systems. *SIGecom Exch.*, *3*(1), 32–43.

Vaughan, R., Gerkey, B., & Howard, A. (2003, Oct). On device abstractions for portable, resuable robot code. In *Proceedings of IROS 2003* (pp. 2121–2427). Las Vegas, Nevada.

Vijayakumar, S., D'souza, A., Shibata, T., Conradt, J., & Schaal, S. (2002). Statistical learning for humanoid robots. *Autonomous Robots*, *12*(1), 55–69.

Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., & Das, H. (2001, March). The CLARAty architecture for robotic autonomy. In *Proceedings of the 2001 IEEE aerospace conference.*

Walters, D. (2003). *Open automation project (OAP).* http://oap.sourceforge.net/.

*Webots 5.* (2005). http://www.cyberbotics.com/. Cyberbotics.

*White box robotics.* (2005). http://whiteboxrobotics.com/. White Box Robotics.

Wolf, D., & Sukhatme, G. (2005, Jul). Mobile robot simultaneous localization and mapping in dynamic environments. *Autonomous Robots*, *19*(1), 53–65.

# Current State of the Art in Distributed Autonomous Mobile Robotics

Lynne E. Parker

Center for Engineering Science Advanced Research
Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge TN 37831-6355, USA
email: ParkerLE@ornl.gov

**Keywords:** Distributed robotics, survey, cooperative robotics, multi-robot systems

**Abstract.** As research progresses in distributed robotic systems, more and more aspects of multi-robot systems are being explored. This article surveys the current state of the art in distributed mobile robot systems. Our focus is principally on research that has been demonstrated in physical robot implementations. We have identified eight primary research topics within multi-robot systems — biological inspirations, communication, architectures, localization/mapping/exploration, object transport and manipulation, motion coordination, reconfigurable robots, and learning – and discuss the current state of research in these areas. As we describe each research area, we identify some key open issues in multi-robot team research. We conclude by identifying several additional open research issues in distributed mobile robotic systems.

## 1 Introduction

The field of distributed robotics has its origins in the late-1980's, when several researchers began investigating issues in multiple mobile robot systems. Prior to this time, research had concentrated on either single robot systems or distributed problem-solving systems that did not involve robotic components. The topics of particular interest in this early distributed robotics work include:

- *Cellular (or reconfigurable) robot systems*, such as the work by Fukuda, *et al.* [20] on the Cellular Robotic System (CEBOT) and the work on cyclic swarms by Beni [8],
- *Multi-robot motion planning*, such as the work by Premvuti and Yuta [38] on traffic control and the work on movement in formations by Arai, *et al.* [2] and Wang [48], and
- *Architectures for multi-robot cooperation*, such as the work on ACTRESS by Asama, *et al.* [4].

Since this early research in distributed mobile robotics, the field has grown dramatically, with a much wider variety of topics being addressed. This paper

examines the current state of the art in autonomous multiple mobile robotic systems. The field of cooperative autonomous mobile robotics is still so new that no topic area within this domain can be considered mature. Some areas have been explored more extensively, however, and the community is beginning to understand how to develop and control certain aspects of multi-robot teams. Thus, rather than summarize the research into a taxonomy of cooperative systems (see Dudek [18] and Cao [12] for previous related summaries), we instead organize this research by the principal topic areas that have generated significant levels of study, to the extent possible in a limited space. As we present the review, we identify key open research issues within each topic area. We conclude by suggesting additional research issues that have not yet been extensively studied, but appear to be of growing interest and need in distributed autonomous multi-robot systems.

## 2 Biological Inspirations

Nearly all of the work in cooperative mobile robotics began after the introduction of the new robotics paradigm of behavior-based control [10,3]. This behavior-based paradigm has had a strong influence in much of the cooperative mobile robotics research. Because the behavior-based paradigm for mobile robotics is rooted in biological inspirations, many cooperative robotics researchers have also found it instructive to examine the social characteristics of insects and animals, and to apply these findings to the design of multi-robot systems[1].

The most common application of this knowledge is in the use of the simple local control rules of various biological societies – particularly ants, bees, and birds – to the development of similar behaviors in cooperative robot systems. Work in this vein has demonstrated the ability for multi-robot teams to flock, disperse, aggregate, forage, and follow trails (e.g., [30,15,17]). The application of the dynamics of ecosystems has also been applied to the development of multi-robot teams that demonstrate emergent cooperation as a result of acting on selfish interests [32]. To some extent, cooperation in higher animals, such as wolf packs, has generated advances in cooperative control. Significant study in predator-prey systems has occurred, although primarily in simulation [7,21]. Competition in multi-robot systems, such as found in higher animals including humans, is beginning to be studied in domains such as multi-robot soccer [46,29].

These areas of biological inspiration and their applicability to multi-robot teams seem to be fairly well understood. More recently identified, less well understood, biological topics of relevance include the use of imitation in higher

[1] For a more detailed analysis of various types of biological systems – what Tinbergen called *differentiating* (e.g., ants) and *integrative* (e.g., birds) – and their relationship to cooperative robotics work (i.e., "swarm" vs. "intentional"), see [35].

animals to learn new behaviors, and the physical interconnectivity demonstrated by insects such as ants to enable collective navigation over challenging terrains.

## 3 Communication

The issue of communication in multi-robot teams has been extensively studied since the inception of distributed robotics research. Distinctions between implicit and explicit communication are usually made, in which implicit communication occurs as a side-effect of other actions, or "through the world", whereas explicit communication is an specific act designed solely to convey information to other robots on the team. Several researchers have studied the effect of communication on the performance of multi-robot teams in a variety of tasks, and have concluded that communication provides certain benefit for particular types of tasks [27,6]. Additionally, these researchers have found that, in many cases, communication of even a small amount of information can lead to great benefit [6].

More recent work in multi-robot communication has focused on representations of languages and the grounding of these representations in the physical world [22,23]. Additionally, work has extended to achieving fault tolerance in multi-robot communication, such as setting up and maintaining distributed communications networks [51] and ensuring reliability in multi-robot communications [34]. While progress is being made in these more recent issues of communication, much work remains to enable multi-robot teams to operate reliably amidst faulty communication environments.

## 4 Architectures, Task Planning, and Control

A great deal of research in distributed robotics has focused on the development of architectures, task planning capabilities, and control. This research area addresses the issues of action selection, delegation of authority and control, the communication structure, heterogeneity versus homogeneity of robots, achieving coherence amidst local actions, resolution of conflicts, and other related issues. Each architecture that has been developed for multi-robot teams tends to focus on providing a specific type of capability to the distributed robot team. Capabilities that have been of particular emphasis include task planning [1], fault tolerance [36], swarm control [31], human design of mission plans [26], and so forth.

A general research question in this vein is whether specialized architectures for each type of robot team and/or application domain are needed, or whether a more general architecture can be developed that can be easily tailored to fit a wider range of multi-robot systems. Relatively little of the previous work has been aimed at unifying these architectures. Perhaps an all-encompassing architecture would be too unwieldy to implement in practical

applications. It remains to be seen if a single general architecture for multi-robot teams can be developed that is applicable to a much wider variety of domains than is currently possible with existing architectures.

## 5 Localization, Mapping, and Exploration

An extensive amount of research has been carried out in the area of localization, mapping, and exploration for single autonomous robots. Only fairly recently has much of this work been applied to multi-robot teams. Almost all of the work has been aimed at 2D environments. Additionally, nearly all of this research takes an existing algorithm developed for single robot mapping, localization, or exploration, and extends it to multiple robots, as opposed to developing a new algorithm that is fundamentally distributed. One exception is some of the work in multi-robot localization, which takes advantage of multiple robots to improve positioning accuracy beyond what is possible with single robots [42,19].

As is the case with single robot approaches to localization, mapping, and exploration, research into the multi-robot version can be described using the familiar categories based on the use of landmarks [14], scan-matching [11], and/or graphs [40,39], and which use either range sensors (such as sonar or laser) or vision sensors. While the single robot version of this problem is fairly well understood, much remains to be studied in the multi-robot version. For example, one question is the effectiveness of multi-robot teams over single-robot versions, and to what extent adding additional robots brings diminishing returns. This issue has begun to be studied (see [39]), but much much remains to be determined for the variety of approaches available for localization, mapping, and exploration.

## 6 Object Transport and Manipulation

Enabling multiple robots to cooperatively carry, push, or manipulate common objects has been a long-standing, yet difficult, goal of multi-robot systems. Many research projects have dealt with this topic area; fewer of these projects have been demonstrated on physical robot systems. This research area has a number of practical applications that make it of particular interest for study.

Numerous variations on this task area have been studied, including constrained and unconstrained motions, two-robot teams versus "swarm"-type teams, compliant versus non-compliant grasping mechanisms, cluttered versus uncluttered environments, global system models versus distributed models, and so forth. Perhaps the most demonstrated task involving cooperative transport is the pushing of objects by multi-robot teams [43,45]. This task seems inherently easier than the carry task, in which multiple robots must grip common objects and navigate to a destination in a coordinated fashion

[49,24]. A novel form of multi-robot transportation that has been demonstrated is the use of ropes wrapped around objects to move them along desired trajectories [16].

Nearly all of the previous work in this area work involves robots moving across a flat surface. A challenging open issue in this area is cooperative transport over uneven outdoor terrains.

## 7 Motion Coordination

A popular topic of study in multi-robot teams is that of motion coordination. Research themes in this domain that have been particularly well studied include multi-robot path planning [52], traffic control [38], formation generation [2], and formation keeping [5,48]. Most of these issues are now fairly well understood, although demonstration of these techniques in physical multi-robot teams (rather than in simulation) has been limited. More recent issues studied within the motion coordination context are target tracking [37], target search [25], and multi-robot docking [33] behaviors.

Nearly all of the previous work has been aimed at 2D domains, although some work has been aimed at 3D environments [52]. One of the most limiting characteristics of much of the existing path planning work is the computational complexity of the approaches. Perhaps as computing processor speed increases, the computational time will take care of itself. In the meantime, this characteristic is a limiting factor to the applicability of much of the path planning research in dynamic, real-time robot teams.

## 8 Reconfigurable Robotics

Even though some of the earliest research in distributed robotics focused on concepts for reconfigurable distributed systems [20,8], relatively little work has proceeded in this area until the last few years. More recent work has resulted in a number of actual physical robot systems that are able to reconfigure. The motivation of this work is to achieve function from shape, allowing individual modules, or robots, to connect and re-connect in various ways to generate a desired shape to serve a needed function. These systems have the theoretical capability of showing great robustness, versatility, and even self-repair.

Most of the work in this area involves identical modules with interconnection mechanisms that allow either manual or automatic reconfiguration. These systems have been demonstrated to form into various navigation configurations, including a rolling track motion [53], an earthworm or snake motion [53,13], and a spider or hexapod motion [53,13]. Some systems employ a cube-type arrangement, with modules able to connect in various ways to form matrices or lattices for specific functions [9,54,44,47].

Research in this area is still very young, and most of the systems developed are not yet able to perform beyond simple laboratory experiments. While the potential of large numbers of robot modules has, to some extent, been demonstrated in simulation, it is still uncommon to have implementations involving more than a dozen or so physical modules. The practical application of these systems is yet to be demonstrated, although progress is being made in that direction. Clearly, this is a rich area for continuing advances in multi-robot systems.

## 9 Learning

Many multi-robot researchers believe that an approach with more potential for the development of cooperative control mechanisms is autonomous learning. While a considerable amount of work has been done in this area for multi-agent learning [50], somewhat less work has been accomplished to date in multi-robot learning. The types of applications that are typically studied for this area of multi-robot learning vary considerably in their characteristics. Some of the applications include predator/prey [7,21], box pushing [28], foraging [31], multi-robot soccer [46,29,41], and cooperative target observation [37].

Particularly challenging domains for multi-robot learning are those tasks that are *inherently* cooperative – that is, tasks in which the utility of the action of one robot is dependent upon the current actions of the other team members. Inherently cooperative tasks cannot be decomposed into independent subtasks to be solved by a distributed robot team. Instead, the success of the team throughout its execution is measured by the combined actions of the robot team, rather than the individual robot actions. This type of task is a particular challenge in multi-robot learning, due to the difficulty of assigning credit for the individual actions of the robot team members. Multi-robot learning in general, and inherently cooperative task learning in particular, are areas in which significant research for multi-robot systems remains.

## 10 Additional Open Issues in Distributed Autonomous Mobile Robotics

It is clear that since the inception of the field of distributed autonomous mobile robotics less than two decades ago, significant progress has been made on a number of important issues. The field has a good understanding of the biological parallels that can be drawn, the use of communication in multi-robot teams, and the design of architectures for multi-robot control. Considerable progress has been made in multi-robot localization/mapping/exploration, cooperative object transport, and motion coordination. Recent progress is beginning to advance the areas of reconfigurable robotics and multi-robot

learning. Of course, all of these areas have not yet been fully studied; we have identified key open research challenges for these areas in the previous sections.

Several other research challenges still remain, including:

- How do we identify and quantify the fundamental advantages and characteristics of multi-robot systems?
- How do we easily enable humans to control multi-robot teams?
- Can we scale up to demonstrations involving more than a dozen or so robots?
- Is passive action recognition in multi-robot teams possible?
- How can we enable physical multi-robot systems to work under hard real-time constraints?
- How does the complexity of the task and of the environment affect the design of multi-robot systems?

These and other issues in multi-robot cooperation should keep the research community busy for many years to come.

## Acknowledgments

This work is sponsored by the Engineering Research Program of the Office of Basic Energy Sciences, U. S. Department of Energy. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U. S. Government purposes. Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the U.S. Dept. of Energy under contract DE-AC05-00OR22725.

## References

1. R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi-robot cooperation in the Martha project. *IEEE Robotics and Automation Magazine*, 1997.
2. T. Arai, H. Ogata, and T. Suzuki. Collision avoidance among multiple robots using virtual impedance. In *Proceedings of the Intelligent Robots and Systems (IROS)*, pages 479–485, 1989.
3. Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.
4. Hajime Asama, Akihiro Matsumoto, and Yoshiki Ishida. Design of an autonomous and distributed robot system: ACTRESS. In *Proceedings of IEEERSJ International Workshop on Intelligent Robots and Systems*, pages 283–290, Tsukuba, Japan, 1989.
5. T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, December 1998.
6. Tucker Balch and Ronald C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):1–25, 1994.

7. M. Benda, V. Jagannathan, and R. Dodhiawalla. On optimal cooperation of knowledge sources. Technical Report BCS-G2010-28, Boeing AI Center, August 1985.
8. Gerardo Beni. The concept of cellular robot. In *Proceedings of Third IEEE Symposium on Intelligent Control*, pages 57–61, Arlington, Virginia, 1988.
9. H. Bojinov, A. Casal, and T. Hogg. Emergent structures in moduluar self-reconfigurable robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1734–1741, 2000.
10. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.
11. W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 476–481, 2000.
12. Y. Uny Cao, Alex Fukunaga, Andrew Kahng, and Frank Meng. Cooperative mobile robotics: Antecedents and directions. In *Proceedings of 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '95)*, pages 226–234, 1995.
13. A. Castano, R. Chokkalingam, and P. Will. Autonomous and self-sufficient conro modules for reconfigurable robots. In *Proceedings of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
14. G. Dedeoglu and G. Sukhatme. Landmark-based matching algorithm for cooperative mapping by autonomous robots. In *Proceedings of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
15. J. Deneubourg, S. Goss, G. Sandini, F. Ferrari, and P. Dario. Self-organizing collection and transport of objects in unpredictable environments. In *Japan-U.S.A. Symposium on Flexible Automation*, pages 1093–1098, 1990.
16. B. Donald, L. Gariepy, and D. Rus. Distributed manipulation of multiple objects using ropes. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 450–457, 2000.
17. Alexis Drogoul and Jacques Ferber. From Tom Thumb to the Dockers: Some experiments with foraging robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 451–459, 1992.
18. Gregory Dudek et al. A taxonomy for swarm robots. In *Proceedings of 1993 IEEE International Conference on Intelligent Robots and Systems (IROS '93)*, pages 441–447, 1993.
19. D. Fox, W. Burgard, H. Kruppa, and S. Thrun. Collaborative multi-robot exploration. *Autonomous Robots*, 8(3), 2000.
20. T. Fukuda and S. Nakagawa. A dynamically reconfigurable robotic system (concept of a system and optimal configurations). In *Proceedings of IECON*, pages 588–595, 1987.
21. Thomas Haynes and Sandip Sen. Evolving behavioral strategies in predators and prey. In Gerard Weiss and Sandip Sen, editors, *Adaptation and Learning in Multi-Agent Systems*, pages 113–126. Springer, 1986.
22. L. Hugues. Collective grounded representations for robots. In *Proceedings of Fifth International Conference on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
23. David Jung and Alexander Zelinsky. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots*, 8(3), July 2000.

24. O. Khatib, K. Yokoi, K. Chang, D. Ruspini, R. Holmberg, and A. Casal. Vehicle/arm coordination and mobile manipulator decentralized cooperation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 546–553, 1996.
25. S. M. LaValle, D. Lin, L. J. Guibas, J-C. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *submitted to 1997 International Conference on Robots and Automation*, 1997.
26. D. MacKenzie, R. Arkin, and J. Cameron. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–52, 1997.
27. Bruce MacLennan. Synthetic ethology: An approach to the study of communication. In *Proceedings of the 2nd interdisciplinary workshop on synthesis and simulation of living systems*, pages 631–658, 1991.
28. S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. In *Proceedings of AAAI-91*, pages 8–14, 1991.
29. S. Marsella, J. Adibi, Y. Al-Onaizan, G. Kaminka, I. Muslea, and M. Tambe. On being a teammate: Experiences acquired in the design of RoboCup teams. In O. Etzioni, J. Muller, and J. Bradshaw, editors, *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 221–227, 1999.
30. Maja Mataric. Designing emergent behaviors: From local interactions to collective intelligence. In J. Meyer, H. Roitblat, and S. Wilson, editors, *Proc. of the Second Int'l Conf. on Simulation of Adaptive Behavior*, pages 432–441. MIT Press, 1992.
31. Maja Mataric. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, 1994.
32. David McFarland. Towards robot cooperation. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson, editors, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 440–444. MIT Press, 1994.
33. B. Minten, R. Murphy, J. Hyams, and M. Micire. A communication-free behavior for docking mobile robots. In *Proceedings of Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
34. P. Molnar and J. Starke. Communication fault tolerance in distributed robotic systems. In *Proceedings of Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
35. L. E. Parker. Adaptive action selection for cooperative agent teams. In Jean-Arcady Meyer, Herbert Roitblat, and Stewart Wilson, editors, *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 442–450. MIT Press, 1992.
36. L. E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.
37. L. E. Parker. Multi-robot learning in a cooperative observation task. In *Proceedings of Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
38. Suparerk Premvuti and Shin'ichi Yuta. Consideration on the cooperation of multiple autonomous mobile robots. In *Proceedings of the IEEE International Workshop of Intelligent Robots and Systems*, pages 59–63, Tsuchiura, Japan, 1990.

39. N. S. V. Rao. Terrain model acquisition by mobile robot teams and n-connectivity. In *Proceedings of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
40. I. Rekleitis, G. Dudek, and E. Milios. Graph-based exploration using multiple robots. In *Proceedings of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
41. P. RIley and M. Veloso. On behavior classification in adversarial environmentts. In *Proceedings of Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
42. S. Roumeliotis and G. Bekey. Distributed multi-robot localization. In *Proceedings of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
43. D. Rus, B. Donald, and J. Jennings. Moving furniture with teams of autonomous robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 235–242, 1995.
44. D. Rus and M. Vona. A physical implementation of the self-reconfiguring crystalline robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1726–1733, 2000.
45. Daniel Stilwell and John Bay. Toward the development of a material transport system using swarms of ant-like robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 766–771, 1993.
46. P. Stone and M. Veloso. A layered approach to learning client behaviors in the robocup soccer server. *Applied Artificial Intelligence*, 12:165–188, 1998.
47. C. Unsal and P. K. Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1742–1747, 2000.
48. P. K. C. Wang. Navigation strategies for multiple autonomous mobile robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 486–493, 1989.
49. Z. Wang, Y. Kimura, T. Takahashi, and E. Nakano. A control method of a multiple non-holonomic robot system for cooperative object transportation. In *Proceedings of Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
50. Gerhard Weiss and Sandip Sen, editors. *Adaption and Learning in Multi-Agent Systems*. Springer, 1996.
51. A. Winfield. Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. In *Proceedings of Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.
52. A. Yamashita, M. Fukuchi, J. Ota, T. Arai, and H. Asama. Motion planning for cooperative transportation of a large object by multiple mobile robots in a 3d environment. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3144–3151, 2000.
53. M. Yim, D. G. Duff, and K. D. Roufas. Polybot: a modular reconfigurable robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 514–520, 2000.
54. E. Yoshida, S. Murata, S. Kokaji, and K. Tomita dn H. Kurokawa. Micro self-reconfigurable robotic system using shape memory alloy. In *Proceedings of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000 – this volume)*, 2000.

# Distributed Robotic Mapping of Extreme Environments

S. M. Thayer*, M. Bernardine Dias, B. Nabbe, B. Digney, M. Hebert, and A. Stentz

Robotics Institute, Carnegie Mellon University

5000 Forbes Avenue, Pittsburgh, PA 15213

## ABSTRACT

In the extreme environments posed by war fighting, fire fighting, and nuclear accident response, the cost of direct human exposure is levied in terms of injury and death. Robotic alternatives must address effective operations while removing humans from danger. This is profoundly challenging, as extreme environments inflict cumulative performance damage on exposed robotic agents. Sensing and perception are among the most vulnerable components. We present a distributed robotic system that enables autonomous reconnaissance and mapping in urban structures using teams of robots. Robot teams scout remote sites, maintain operational tempos, and successfully execute tasks, principally the construction of 3-D Maps, despite multiple agent failures. Using an economic model of agent interaction based on a free market architecture, a virtual platform (a robot colony) is synthesized where task execution does not directly depend on individual agents within the colony.

**Keywords:** Distributed Robotics, Free Market Architecture, Cooperative Stereo, 3-D Mapping, Robot Colony

## 1. MOTIVATION

Military Operations in Urban Terrain (MOUT) pose fierce constraints such as limited visibility, complex and expansive fortifications, limited intelligence, and the presence of native populations and other non-combatants that prohibit deployment of large forces [1,2]. Further, the use of asymmetric threats, e.g. biological and chemical agents, against both land forces and indigenous populations in urban settings is an increasing likelihood [3]. These conditions place land forces and non-combatants in a highly non-deterministic, dangerous, confrontational, and volatile environment. An effort to identify and improve the ability of ground forces to project sufficient force and safeguard non-combatants is underway. This program, called the MOUT Advanced Concept Technology Development (ACTD), focuses on improving operational effectiveness in urban areas [4, 5].

The development of robotics technology will enable minimally invasive and precise MOUT operations that reduce risk to both ground forces and non-combatants by removing soldiers from dangerous and sometimes confrontational tasks [6]. Potential tasks for robotic systems include mine sweeping, reconnaissance, security and monitoring presence, and communications infrastructure [7]. DARPA has undertaken the task of enabling distributed robotics technology to execute urban reconnaissance missions. As a part of this effort, DARPA's Software for Distributed Robotics (SDR) program is looking at revolutionary approaches to the development of multi-agent robotic systems within this task domain. Under the auspices of the SDR Program, our team is producing software technology designed to construct interior maps of urban structures using groups of homogenous mobile robots. This concept software will be demonstrated on surrogate, simplistic mobile robotic platforms and eventually ported to mission capable mobile robotic systems.

Our approach to the problem focuses initially on the production of reliable, accurate, and fault-tolerant software by exploiting the redundancy inherent in a group of homogenous robots, or colony. Our methodology is to model multi-agent dynamics in terms of economic activity; thus, the motivation for robot effort is directly coupled to the rewards of task-execution [8]. We will incorporate learning and perception to build systems that degrade gracefully when faced with component or agent failures, that learn to execute tasks more efficiently than is possible as individual agents, and that deliver timely and accurate intelligence with respect to urban reconnaissance operations.

## 2. APPROACH

Some tasks, such as robotic exploration or reconnaissance, can be effective only if carried out by a team of robots. A robot team can accomplish a given task more quickly than a single agent can by dividing the task into sub-tasks and executing them

* Correspondence: Email sthayer@ri.cmu.edu : Phone (412)-268-7816, Fax (412) 268-5895

concurrently. A team can also make effective use of specialists (e.g., an agent that scouts an area, picks up objects, hauls payload) rather than require that each robot be a generalist, capable of performing all tasks but expert at none.
Difficulty arises, however, in coordinating all of these robots to perform a single, global task. One approach is to consider the robot team as a single robot "entity" with many degrees of freedom. A central computer coordinates the group optimally to perform the specified task. The problem is that optimal coordination is computationally difficult; the best known algorithms are exponential in complexity, so this approach is intractable for teams larger than a few robots. Further, the approach assumes that all information about the robots and their environment can be transmitted to a single location for processing and that this information does not change during the construction of an optimal plan. These assumptions are unrealistic for applications such as exploration in an unknown environment where communication is limited and robots behave in unpredictable ways. Another weakness with this approach is that it produces a highly vulnerable system. If the leader (the central planning unit) malfunctions, a new leader must be available or the entire team is disabled. A system that surmounts a single point of failure is crucial in applications such as urban reconnaissance.

Local and distributed approaches address the problems that arise with centralized, globally coordinated approaches. Each robot operates largely independently, acting on information that is locally available through its sensors. The robots are better able to respond to unknown or changing environments, since they sense and respond to the environment locally. A robot may coordinate with other robots in its vicinity, perhaps to divide a problem into sub-problems or to work together on a sub-task that cannot be accomplished by a single robot. Typically, little computation is required, since each robot need only plan and execute its own activities. Also, little communication is required, since the robots communicate only with others in their vicinity. The disadvantage of distributed approaches is that the solutions can be grossly suboptimal.

Consider an economic system for coordinating robots. In market economies, individuals are free, within given bounds, to exchange goods and services and enter into contracts at will. Individuals are in the best position to understand their needs and the means to satisfy them. At times they cooperate with other members of the society to achieve an outcome greater than that possible by each member alone. At times they compete with other members to provide goods or services at the lowest possible cost, thus eliminating waste and inefficiency. Combining the best of both approaches, market economies are robust and nimble like distributed approaches, yet allow for some centralization for optimization when time and computational resources permit.

In the past decade there has been growing interest in multi-agent systems. Mataric [18] presents a comprehensive summary of some of the principal efforts in this area of research. Jensen and Veloso [15], Švestka and Overmars [28], and Brumitt and Stentz [13] are examples of the centralized approach to control a multi-robot system organized hierarchically. A number of researchers have developed biologically inspired, locally reactive, behavior-based systems to carry out simple tasks [9, 10, 11, 17]. These distributed systems have found applications in many different domains. Additional review of recent approaches to multi-robot teaming can be found in [8]. To the best of our knowledge, we are the first to use a free market architecture for controlling a team of self-interested agents that perform tasks and solve problems.

## 3. FREE MARKET ARCHITECTURE[1]

### 3. 1 Revenues and Costs in the Economy

When a team of robots is modeled as an economy to perform a task, the goal of the team is to execute the task well while minimizing costs. A function, ***trev***, is needed that maps possible task outcomes onto revenue values. Another function, ***tcost***, is needed that maps possible schemes for performing the task onto cost values. As a team, the goal is to execute some plan ***P*** such that profit, ***trev(P) – tcost(P)***, is maximized. Individual cost and revenue functions must also be designed to provide a means for distributing the revenue and assessing costs to individual robots.

The sum of individual revenues and costs will determine the team's revenues and costs. However, the distribution is not even: individuals are compensated in accordance with their contribution to the overall task, based on factors that are within the control of each. An individual that maximizes its personal production and minimizes its personal cost receives a larger share of the overall profit. Therefore, by acting strictly in their own self-interests, individuals maximize not only their own profit but also the overall profit of the team.

### 3.2 Determining Price via the Bidding Process

[1] See [8] for a more detailed description of the free market architecture.

The team's revenue function is not the only source of income for the robots. A robot can also receive revenue from another robot in exchange for goods or services. In general, two robots have incentive to deal with each other if they can produce more aggregate profit together than apart. Such outcomes are win-win rather than zero-sum. The *price* dictates the payment amount for the good or service. Since many factors that could affect the price may be hidden or complex, a common approach to determining price is to *bid* for a good or service until a mutually acceptable price is found. A plausible bidding strategy is to start by bidding a price that is personally most favorable, and then successively retreat from this position until a price is mutually agreed upon.

Note that a given robot can negotiate several potential deals at the same time. It bids the most favorable price for itself for all of the deals, successively retreats from this position with counter bids, and closes the first deal that is mutually acceptable. Note also that a deal can be multi-party, requiring that all parties agree before any part of the deal is binding. The negotiated price will tend toward the intersection of the supply and demand curves for a given service. If a service is in high demand or short supply, the price will be high. This information will prompt other suppliers to enter the fray, driving the price down. Likewise, if demand is low or supply high, the low price will drive suppliers into another line of business. Thus, price serves to optimize the matching of supply to demand. Finally, it is important to note that price and bidding are low bandwidth mechanisms for communicating aggregate information about costs.

**3.3 Opportunistic Optimization, Learning, and Adaptation**

Conspicuously absent from the free market system is a rigid, top-down hierarchy. Instead, the robots organize themselves in a way that is mutually beneficial. Thus, robots will cooperate if they have complementary roles, i.e., if all robots involved can make more profit by working together than by working individually. Conversely, robots will compete if they have the same role, i.e., if the amount of profit that some can make is negatively affected by the services provided by the others. The flexibility of the market model allows the robots to cooperate and compete as necessary to accomplish the assigned task. Since the aggregate profit amassed by the individuals is directly tied to the success of the task, this self-organization yields the best results. But this does not preclude centralized optimization; instead, it is opportunistic as circumstances permit.

One of the greatest strengths of the market economy is its ability to deal successfully with changing conditions. Since the economy does not rely on a hierarchical structure for coordination and task assignment, the system is highly robust to changes in the environment, including destroyed or injured robots. Disabling any single robot will not jeopardize the system's performance. By adding escape clauses for "broken deals," any tasks undertaken by a robot that malfunctions can be re-bid to other robots, and the entire task can be accomplished. It is also important to realize that a robot does not need to remain idle if only a portion of its functionality is disabled. The market architecture allows a partially disabled robot to bid out the use of its available resources opportunistically. For example, if a robot gets stuck while executing a task, it can still use its computing resources as needed while attempting to recruit another robot to mobilize itself. Another key to optimality is that a robot can always sub-contract a task it undertook if the conditions change and it becomes more profitable to outsource (i.e., get a different robot to carry out the task). Thus, the market model allows the robots to deal with dynamic environments in an opportunistic and adaptive manner.

Within the market economy, a robot can learn new behaviors and strategies as it executes missions. This learning applies to both individual behaviors and negotiations as well as to the entire team. Individual robots may learn that certain strategies are not profitable, or that certain robots are apt to break a contract by failing to deliver the goods or proper payment. Individuals may also learn to identify areas where the risk of injury is high and share this knowledge with members of the colony. The robot team may learn that certain types of robots are in over-supply, indicated by widespread bankruptcy or an inability to make much money. Conversely, the robot team may learn that certain types of robots are in under-supply, evidenced by excessive profits captured by members of the type. Thus, the population can learn to exit members of one type and recruit members of another.

**3.4 Constructing the Free Market Architecture**

Consider a team of robots assembled to perform a task. A software agent, representing the user, negotiates with the team to perform the task. The user desires the team to perform the task well while minimizing costs. To accomplish this, the user defines a revenue function that maps possible task outcomes onto revenue values, and a cost function that maps possible schemes for performing the task onto cost values. With the aim of maximizing their individual profits, the robots bid on parts of the task. They receive revenue for performing their subtasks, and they are charged for costs incurred during execution. Once they have received their assignments, the robots can continue to negotiate among themselves, buying and selling subtasks to minimize costs. If the task changes during execution, perhaps due to new information, new subtasks are created that are bid out to the robots. If a robot malfunctions or dies during execution, its subtasks are bid out to the surviving robots. Robot resources, such as sensing, communication, and computing, can be bought and sold in the marketplace to concentrate resources where they are most productive.

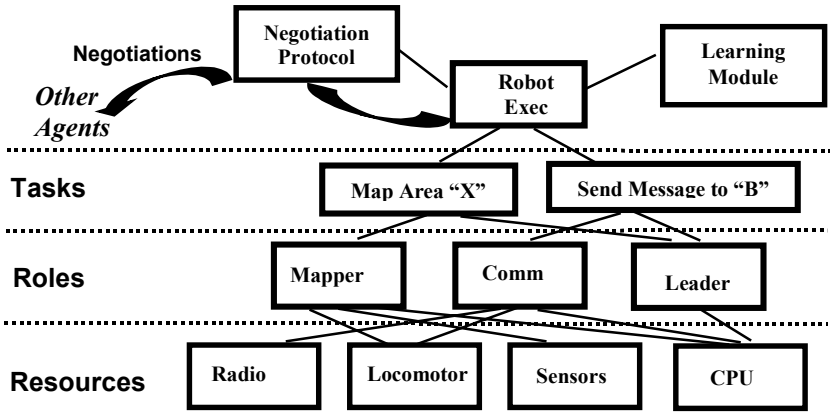



**Figure 1: Robot software architecture for each individual in a robot colony**

As shown in Figure 1, the architectural for each robot is layered. In the bottom layer are the resources under the robot's control, such as sensors, effectors, locomotors, computers, and communication devices. These resources are available to the robot to perform its tasks, and any unused resources can be sold to other robots in the colony. The next layer consists of the robot's roles for accomplishing tasks. Roles are application-specific software modules that implement particular robot capabilities or skills, such as collecting sensor data or hauling materials. The roles utilize resources (in the layer below) to execute their tasks. Roles execute tasks that match their specific capabilities. They receive assignments by bidding on tasks offered by other robots. As they execute their tasks, they may generate other tasks or subtasks to be bid out to the other robots.

At the top layer in the architecture, the robot executive coordinates the activities of the robot and its interactions with other robots. The executive bids on tasks for the robot to perform and offers tasks for sale. It matches tasks to roles, schedules the roles to run, resolves contention for resources, and offers unused resources for sale to other robots. The executive is equipped with a learning module that enables it to perform better over time by learning which negotiation strategies are the most effective.

## 4. COOPERATIVE STEREO MAPPING ROLE

One important role for individuals within robot colonies is distributed mapping. This role is illustrated in Figure 2, where robots 1 and 2, each equipped with one camera, are controlled by the planner so that the fields of view of their cameras provide maximum coverage of the environment. Assuming that the planner has also controlled the robots so that the fields of view overlap (at positions A and B for example), it is possible to reconstruct the 3-D geometry of the environment at those positions by processing the images from the two robots. Reconstructing 3-D maps from two robots is the cooperative stereo approach to map reconstruction. Collecting 3-D maps reconstructed from different positions provides a complete representation of the environment built cooperatively (the mapped area is shown in yellow or lightly shaded for b/w prints in Figure 2). This representation is suitable for:

- **Viewing:** The user can see the environment from different viewpoints, measure distances to or sizes of objects of interest, and generate floor plans for future use.
- **Route planning:** The location of potential obstacles can be derived from the 3-D maps. The obstacles can be detected at far greater range than can be achieved by conventional safeguarding sensors such as sonar. This information can be used directly by a route planner.

- **Coverage planning:** A difficult problem in exploration by multiple robots is to decide where to look next in order to ensure coverage of the environment. Using the maps, it is possible to reason about the relative positions of surfaces in order to plan paths for coverage. In particular, it is possible to detect parts of the environment that are occluded by surfaces visible from the recorded robot positions, and which should be explored.

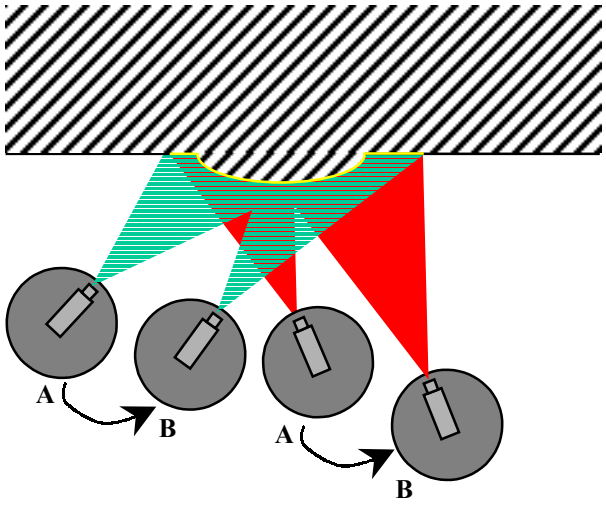



**Figure 2: Distributed mapping scenario with two robots cooperatively constructing a stereo map.**

It is important to note that while in principle it is possible to recover 3-D data from a single robot with stereo cameras, using cooperating robots has important advantages. One obvious advantage is the savings in hardware and computation time since a single camera is needed on each robot and the full load of the stereo computation does not reside on any given robot. A more important advantage, however, is that stereo reconstruction accurate enough for rendering can be achieved from a long range by taking advantage of the fact that the robots may be separated by a wide baseline. To be useful, the system must generate at least qualitatively correct reconstruction up to 10-20m, which is not possible using the short baseline that is typical on small robots. To illustrate this point, the graph in Figure 3 shows the error in depth $Z$ as a function of $Z$ for the type of camera currently used with a field of view of 60 degrees and for different baseline values $B$=0.1, 0.2,0.5,1,2m. The error is plotted using the standard stereo error formula $\Delta Z = Z^2/Bf$. The graphs clearly show that reconstruction past a few meters from the robot is not practical for short baselines and that baselines on the order of one meter or more become necessary. Such large baselines can be achieved only through the use of multiple robots.

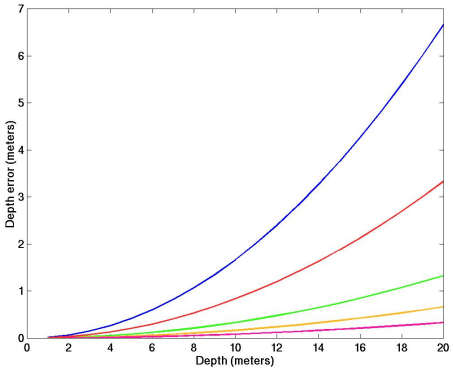



**Figure 3: Error in reconstructed depth as a function of distance from the camera for different baselines.**

Robust recovery of the scene geometry from multiple robots and extraction of information from the maps involves:

**Cooperative Calibration:** Although the relative positions and orientations of robots 1 and 2 are known to some degree from dead-reckoning and other means of internal pose estimation, the accuracy of the relative pose between them is typically not sufficient for stereo reconstruction. Cooperative calibration recovers relative poses of robot cameras automatically from image features.

**Spare Feature Estimation:** Once the robots are mutually calibrated, the next task is to compute the 3-D positions of sparse features in the scene.

**Dense Reconstruction:** The final step in reconstructing scene geometry from robots 1 and 2 is to convert the sparse structure to a dense reconstruction of the environment suitable for rendering and texture mapping.

### 4.1 Cooperative Calibration

Given images of a scene from arbitrary robot positions, the first task is to recover the relative poses of the cameras. These poses must be computed precisely since an error of a few image pixels in mapping points from one image to the next may lead to large errors in the 3-D reconstruction. In particular, the pose estimates from dead reckoning are not precise enough for image matching. The general approach is to extract point features from the images, find correspondences between the two sets of features and recover the relative poses of the cameras from the feature matches. Typical features used for calibration are shown in Figure 4. These features were extracted using an interest operator derived from the Harris operator [43].

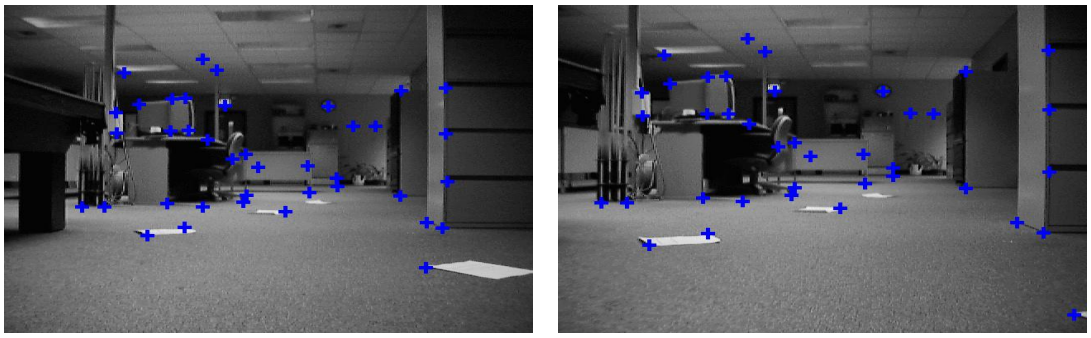


**Figure 4: Typical sparse features extracted from indoor images using the Harris operator.**

#### *4.1.1 Feature matching using photometric invariants*

Instead of using local correlation to match the features, we use a technique based on photometric invariants that was originally introduced in [37] and used for image matching in [42]. Given a feature location $\mathbf{p} = (x,y)$, the general idea is to compute the derivatives up to order 3 ( $I_x$, $I_{xx}$, $I_{xy}$, etc.) of the image in a neighborhood around **p.** Then, combine the derivatives into quantities that are invariant to rotation and translation in the image. For example, the quantity $I_x^2 + I_y^2$ is the magnitude of the gradient, which is clearly invariant by rotation. For derivatives up to order 3, it can be shown that 8 such invariants exist so that each feature $\mathbf{m}^i_j$ can be described by a 8-vector $\mathbf{v}^i_j$ which characterizes the local distribution of intensity and is geometrically invariant.

Invariance to scale is also important since the scale of the features may vary substantially across images in general poses. Although direct scale invariance is not possible, the derivatives can be easily computed at different scales since they are computed by convoluting derivatives of Gaussian filters of variance, for example, $I_x = G_x(\sigma) \otimes I$, where $\sigma$ is the variance of the Gaussian filters. The vectors $\mathbf{v}^i_j$ are computed over a range of values of $\sigma$ corresponding to a range of image scale factors. Features that have similar invariant vectors are matched. The similarity between vectors is defined as the weighted sum of squared differences between their coordinates. A similar approach is used in [39] for recovering epipolar geometry.

#### *4.1.2 Using Prior Pose Estimates*

The second enhancement to the standard image-based calibration algorithm is to take advantage of the fact that we do have information about the relative poses from the robots' own internal pose estimation systems. In fact, we not only have an approximate estimate of the robot pose but also an estimate of the uncertainty of the pose. The situation is illustrated in

Figure 5. Using camera 1 as the reference camera, the true pose of camera 2 with respect to camera 1 is (**R**,**T**); the relative pose reported by the positioning systems is (**R'**,**T'**); and the error between the true and estimated poses is (dθ,d**T**) where dθ is the error in orientation in the plane of travel of the robot. Although (dθ,d**T**) is not known, the positioning system can provide a bound on $d\theta_{max}$ – the maximum angular deviation from the true orientation – and $d\mathbf{T}_{max}$ – the maximum position error which places all the possible positions of camera 2 in a disk centered at **T** (Figure 5(a)). Those bounds define a region in which the true pose (**R**,**T**) may lie: (**R**,**T**) is in the region defined by all the possible (**T'**+d**T**) and **R**(θ'+dθ) with $\|d\mathbf{T}\| < d\mathbf{T}_{max}$. Given a feature position in image 1, $\mathbf{m}_1$, the set of corresponding epipolar lines in image 2 is defined by the parameter vectors (**T**'+d**T**) × **R**(θ'+dθ) $\mathbf{m}_1$ for all possible values of d**T** and dθ. This set of lines defines a search region for the correspondences with $\mathbf{m}_1$ – shown in green (gray shaded area for b/w prints) in Figure 5(b) around the initial estimate of the epipolar line shown in red (darker line for b/w prints). If the initial position error is small, the search problem is reduced to searching in the vicinity of the epipolar line and the calibration amounts to a small adjustment of **R** and **T**. For larger position errors, the region covers a larger area of the image but still eliminates most of the outliers in practice.

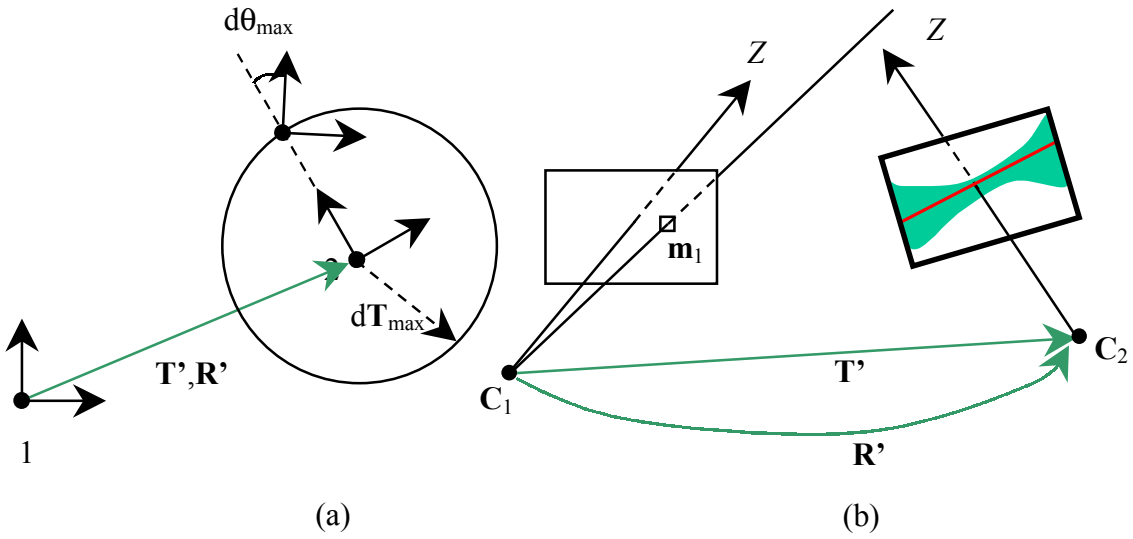



**Figure 5: Using prior pose estimations to limit search**

**4.2 Sparse Feature Estimation**

Given the calibration information from the two robots, the 3-D locations of each feature can be recovered by triangulation. In this case, the coordinates of each feature are computed independently. We have also applied global estimation techniques, which attempt to refine robot positions and feature positions by minimizing the distance between the image features and the image projection of the corresponding points – bundle adjustment technique [33]. Such global techniques become beneficial only when a substantial number of images are used. In our case, the experiments show that those techniques do not substantially affect the result and that straight triangulation is sufficient.

For example, Figure 6 (left) shows the depth reconstructed of matched features of Figure 4, as well as the same result (right) (right) in which the 3-D points corresponding to the features are shown in an oblique view. The lines are drawn from the point features to the ground plane. A few objects are indicated to facilitate visual registration of the data. This first reconstruction step provides a representation of the sparse structure of the environment. This sparse information can be reasonably used for planning or partial occlusion reasoning.

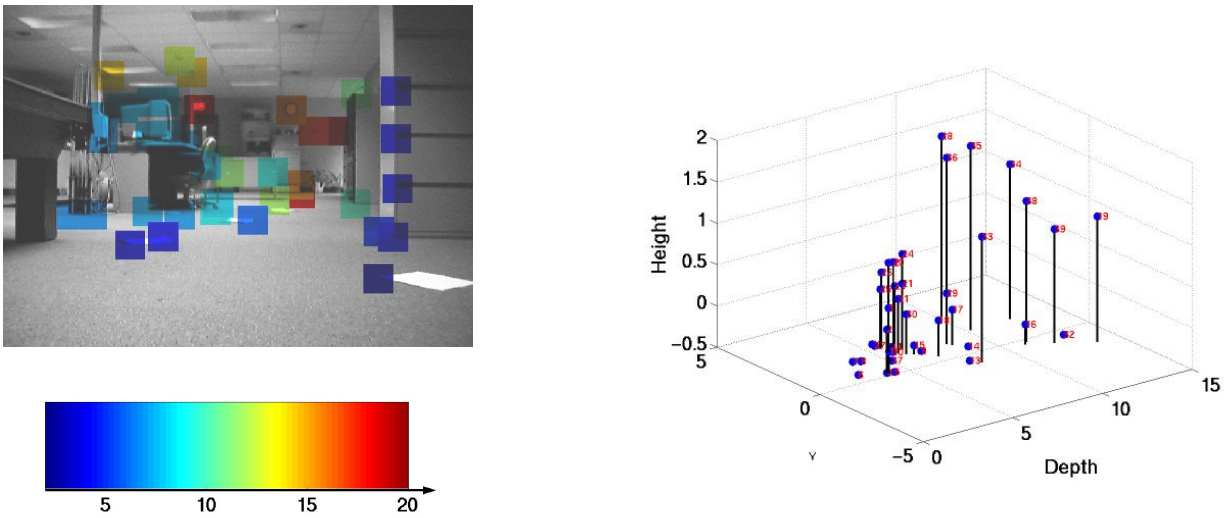



**Figure 6: (Left) Sparse depth map computed from the features of Figure 4. The depth is indicated by a color-coded square centered at each feature. The color scale is indicated below the figure. (Right) Oblique view of the reconstructed 3-D points corresponding to features of Figure 4.**

### 4.3 Dense Reconstruction

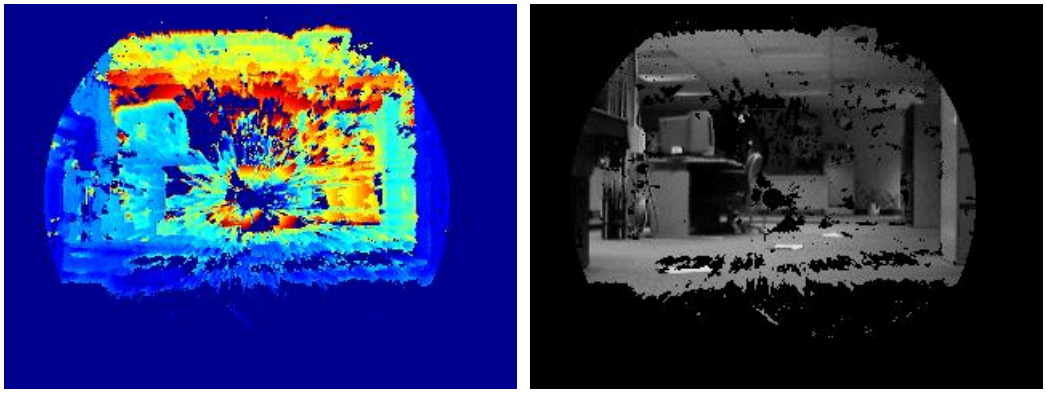


**Figure 7: Color-coded depth map after filtering (left) and corresponding mask (right)**

Sparse feature estimation provides useful information for visualization and planning but is insufficient for realistic rendering of the environment, which requires the recovery of the coordinates of a dense set of pixels in the image. Given two robot positions and assuming that the calibration problem has been solved using the techniques previously described, the problem is equivalent to the standard problem of stereo except that the images are taken from different robots instead of from a single camera rig, as is normally the case.

The general approach to stereo is to rectify the images so that the epipolar lines are the scanlines of the images and to search along the scanlines for matching locations based on a local comparison criterion such as correlation or sum of squared differences (SSD.) The rectification is critical because searching along arbitrary scanlines is not computationally practical. While in principle this approach is feasible, there are two key differences with standard stereo from fixed cameras, which are both related to the fact that the relative positions and orientations of the robots are arbitrary. First, as noted, the baseline between the robots may be large. Although necessary for accuracy reasons, this complicates the matching between images because of potentially large distortions between the two images. Second, the search for correspondences is complicated by the fact that rectification may lead to unacceptable warping of the images. Some preliminary results are shown in Figures 7 and 8(a) and 8(b).

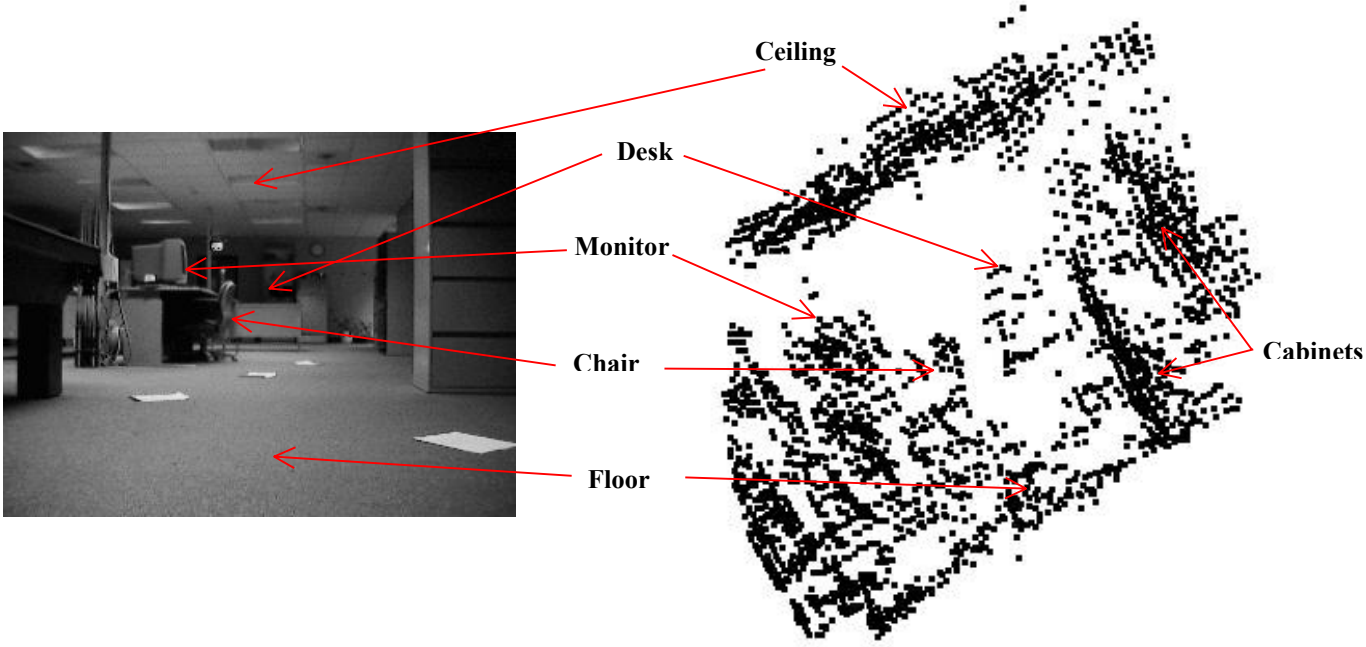



**Figure 8: 3-D View of the point cloud constructed by our cooperative stereo mapping**

## 5. RESULTS

This architecture was first verified by tasking a team of mobile robots with distributed mapping. [2] The mobile robots, located at different starting positions in a known simulated world, were assigned the task of visiting a set of pre-selected observation points. The robot colony was structured as illustrated below:

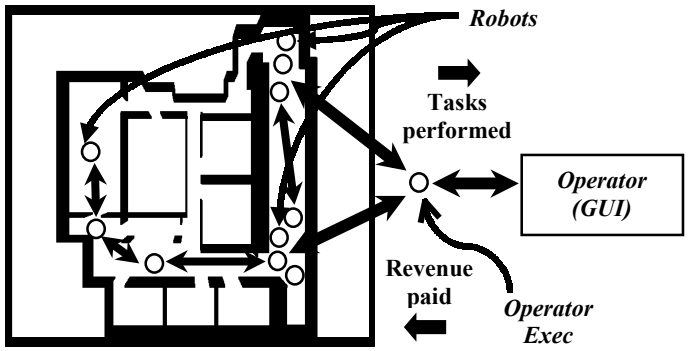



**Figure 9: Organizational structure for a colony of robots engaged in distributed mapping**

This problem is equivalent to the distributed traveling salesman problem, where the observation points are the cities to visit. Each agent was equipped with a map of the world, which enabled it to calculate the cost associated with visiting each of these cities. The costs were the lengths of the shortest paths between cities in an eight-connected grid, interpreted as money. The interface between the human operator and the team of robots was a software agent, the *operator executive* (*exec*). The *exec*

[2] Detailed descriptions of the implementation and results can be found in [8].

conveyed the operator's commands to the members of the team, managed the team revenue, monitored the team cost, and carried out the initial city assignments. Being a self-interested agent, the *exec* aimed to assign cities quickly while minimizing revenue flow to the team. In our initial implementation, the *exec* adopted a greedy algorithm for assigning tasks. Once the *exec* had completed the initial city assignments, the robots negotiated among themselves to subcontract city assignments. Only single-city deals were considered, and the robots continued to negotiate among themselves until no new, mutually profitable deals were possible. Thus, negotiations ceased once the system settled into a local minimum of the global cost. Dias and Stentz published preliminary results[8].

We also tested system response to dynamic conditions in a scenario similar to exploration of a partially known world. The operator designated a set of 14 observation points, or cities, to be visited in a simulated world. Four robots negotiated, won tours, and set off to explore. Whenever a robot approached a doorway, a new observation point was triggered inside the "newly observed" room. When a new goal was triggered, the robots ceased their current tours and began negotiations to determine new tours that were most profitable in light of the new information. This resulted in an additional six cites for exploration, for a total of 20 cites. Through this exploration scenario.[3], the robots evolved by adapting to dynamic conditions.

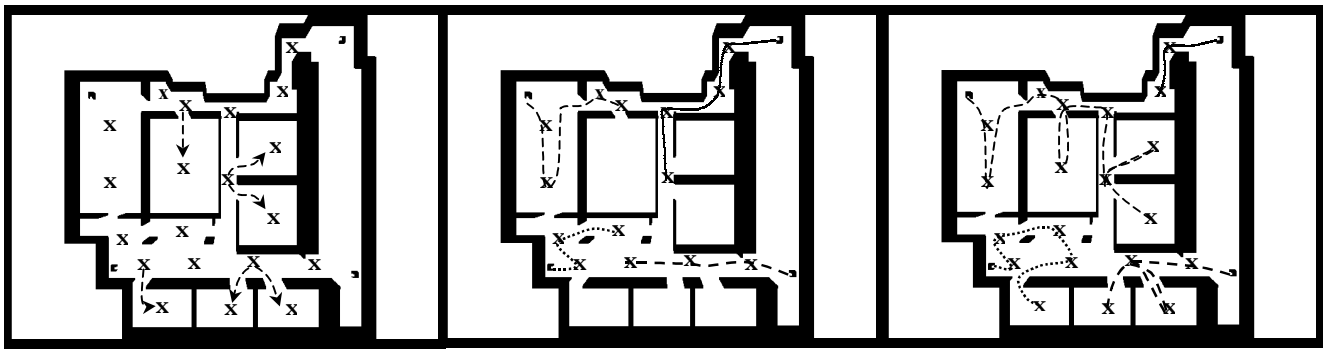


**Figure 10: Results from dynamic re-negotiations.**

## 6. SUMMARY

We have outlined the design of a colony of mobile robots capable of providing maps from the interior of urban structures. Traditionally, new paradigms are brittle and may not convey any advantage when deployed outside of the laboratory environment [9]. Our primary mission is to exploit multiple robots as a means to develop capability that is robust to individual failure much in the same way as traditional military units operate. These units exploit redundancy such that critical capability is preserved when losses are incurred; consequently, they can continue to operate when faced with multiple casualties.

Our architecture is unique in that task execution is only loosely coupled to the existence of individual robots. This is an important technological leap that must be developed in order to place robot colonies into extreme environments to perform dangerous tasks. Colony performance can be enhanced by the development of accelerated learning through "on-line" and instantaneous exchange of experiences. The ubiquitous connectivity within a colony provides the infrastructure to promote the exchange of successful skills and behaviors in a trivial fashion. Through our research, apprenticeship in established robot colonies will be measured in "seconds of download time."

Secondary goals of this effort are to equip this software architecture such that new applications can be readily programmed. As a first order enabler, the design of this software infrastructure has been abstracted from the task. In the worst case, new applications would require coding with provided application programmer interfaces (APIs). To eliminate this requirement, we are also pursuing the construction of task archives and behavioral seeding technologies that would enable robots to learn new tasks by downloading "seeds" from an archive. These seeds would grow into fully functional, perhaps even optimal, task level capabilities within our fertile distributed learning infrastructure. Once matured, this functionality would be stored in a

[3] A movie of this evolution is available at http://www.frc.ri.cmu.edu/projects/colony/frcworld.shtml

task archive for future mission capability. Thus, a process of iterative refinement for colonies of robots would be available. This archive of robot-derived experience, knowledge, and data is a first step in creating accelerated cultural learning and knowledge acquisition for robots, a distinctly human phenomenon.

## 7. FUTURE WORK

The results described above provide the building blocks for geometric environment reconstruction from multiple robots. Many issues must be addressed before they can be assembled into an effective mapping system. First, there are many choices of parameters for rectification and stereo – for example, size of matching window or interpolation mode – and extensive experiments are needed to determine the optimal choices in a typical environment. Secondly, there is the need to incorporate maps from multiple pairs of robots into composite maps. The multi-map approach will support addressing of maps from different locations, but registration of individual maps is necessary to ensure coherence of the representation throughout the environment. Map integration can be accomplished by one of two means. First, the individual 3-D maps from robot pairs can be iteratively modified to yield an optimal combined map. An alternative that we have used successfully is to find correspondences between the individual maps constructed from pairs of robots and to estimate the relative poses between the maps based on the correspondences. A global optimization approach is then used to find the optimal set of poses and correspondences between maps. This is the approach we proposed in [36]. We will apply both approaches to the multi-robot map building problem and will design the integration system based on the evaluation of those approaches. In addition, we are examining the following:

**Interactive Queries:** An operator-designated region, line, or point of interest from an approximate floor plan is used to guide retrieval and reconstruction from local stereo matching.

**Generating Floor Plans:** A cross section from an integrated map consisting of all the points from all the cooperative stereo maps taken in that location.

**View-Based Rendering:** generating realistically rendered views from arbitrary positions.

## ACKNOWLEDGEMENTS

This research was sponsored in part by DARPA under contract "Cognitive Colonies" (contract number N66001-99-1-8921, monitored by SPAWAR). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The authors thank the additional members of the Cognitive Colonies group for their valuable contribution: Charles Smart, Brian Frederick, Vanessa De Gennero, and Dean Boustead.

## REFERENCES

1. Brig. General J. R. Groves, Jr. "Operations in Urban Environments" *Military Review*, August 1998.
2. MOUT ACTD Program Overview. http://mout.actd.org
3. Lt. Gen. J. Rhodes, "Jane's Defense Weekly Interview" *Jane's Defense Weekly*, Feb 4, 1998.
4. MOUT ACTD Fact Sheet. http://mout.actd.org
5. MOUT ACTD Operational Requirements. http://mout.actd.org/req.html
6. SFC L. Lane, "Robots out Front" *Soldier's Online*, April 1995.
7. D. G. Knichel, "Robotics Insertion Technology" May 1997.
8. M. B. Dias and A. X. Stentz, "A Free Market Architecture for Coordinating Multiple Robots" CMU-RI-TR-99-42, Carnegie Mellon University, December 1999
9. L. W. Grau, "Bashing the Laser Range Finder with a Rock" *Military Review*, May 1997.
10. Arkin, R. C., "Cooperation without Communication: Multiagent Schema-Based Robot Navigation" *Journal of Robotic Systems*, Vol. 9, No.3, pp. 351-364, 1992.
11. Arkin, R. C., Balch, T., "AuRA: Principles and Practice in Review" *Journal of Experimental & Theoretical Artificial Intelligence*, Vol. 9, No. 2/3, pp.175-188, 1997.
12. Brooks, R. A., "Elephants Don't Play Chess" *Robotics and Autonomous Systems*, Vol. 6, pp.3-15, 1990.
13. Brumitt, B. L., Stentz, A., "Dynamic Mission Planning for Multiple Mobile Robots" *Proceedings of the IEEE International Conference on Robotics and Automation*, No. 3, pp. 2396-2401, 1996.
14. Golfarelli, M., Maio, D., Rizzi, S., "A Task-Swap Negotiation Protocol Based on the Contract Net Paradigm" Technical Report CSITE, No. 005-97, 1997.
15. Jensen, R. M., Veloso, M. M., "OBDD-based Universal Planning: Specifying and Solving Planning Problems for Synchronized Agents in Non-Deterministic Domains" *Lecture Notes in Computer Science*, No. 1600, pp. 213-248, 1999.
16. Johnson, N. F., Jarvis, S., Jonson, R., Cheung, P., Kwong, Y. R., Hui, P. M., "Volatility and Agent Adaptability in a Self-Organizing Market" *Physica A*, Vol. 258, No. 1-2, pp. 230-236, 1998.

17. Lux, T., Marchesi, M., "Scaling and Criticality in a Stochastic Multi-Agent Model of a Financial Market" *Nature*, Vol. 397, No. 6719, pp. 498-500, 1999.
18. Mataric, M. J., "Issues and Approaches in the Design of Collective Autonomous Agents" *Robotics and Autonomous Systems*, Vol. 16, pp. 321-331, 1995.
19. Pagello, E., D'Angelo, A., Montsello, F., Garelli, F., Ferrari, C., "Cooperative Behaviors in Multi-Robot Systems through Implicit Communication" *Robotics and Autonomous Systems*, Vol. 29, No. 1, pp. 65-77, 1999.
20. Parker, L. E., "ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation" *IEEE Transactions on Robotics and Automation*, Vol. 14, No.2, pp. 220-240, 1998.
21. Sandholm, T., "An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations" *Proceedings, Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 256-262, 1993.
22. Sandholm, T., Lesser, V., "Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework" *Proceedings, First International Conference on Multiagent Systems (ICMAS-95)*, pp. 328-335, 1995.
23. Schneider-Fontán, M., Mataric, M. J., "Territorial Multi-Robot Task Division" *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 5, 1998.
24. Schneider-Fontán, M., Mataric, M. J., "A Study of Territoriality: The Role of Critical Mass in Adaptive Task Division" *Proceedings, From Animals to Animats 4, Fourth International Conference on Simulation of Adaptive Behavior (SAB-96)*, MIT Press/Bradford Books, pp. 553-561, 1996.
25. Schwartz, R., Kraus, S., "Negotiation On Data Allocation in Multi-Agent Environments" *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp.29-35, 1997.
26. Shehory, O., Kraus, S., "Methods for Task Allocation via Agent Coalition Formation" *Artificial Intelligence Journal*, Vol.101, No:1-2, pp.165-200, May, 1998.
27. Smith, R., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver" *IEEE Transactions on Computers*, Vol. C-29, No. 12, December, 1980.
28. Švestka, P., Overmars, M. H., "Coordinated Path Planning for Multiple Robots" *Robotics and Autonomous Systems*, Vol. 23, No. 4, pp. 125-133, 1998.
29. Tambe, M., "Towards Flexible Teamwork" *Journal of Artificial Intelligence Research*, Vol. 7, pp. 83-124, 1997.
30. Veloso, M., Stone, P., Bowling, M., "Anticipation: A Key for Collaboration in a Team of Agents" Submitted to the *3rd International Conference on Autonomous Agents*, pp. 1-16, 1998.
31. Wellman, M., Wurman, P., "Market-Aware Agents for a Multiagent World" *Robotics and Autonomous Systems*, Vol. 24, 1998.
32. Zeng, D., Sycara, K., "Benefits of Learning in Negotiation" *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp.36-41, 1997.
33 P. Beardsley, A. Zisserman, D. Murray, "Sequential Updating of Projective and Affine Structure from Motion" *International Journal of Computer Vision*, Vol. 23, No. 3, 1997.
34 P. Chang, M. Hebert, "Omnidirectional Structure from Motion" *Proc. Workshop on Omnidirectional Vision*, IEEE Press, 2000.
35 O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint* MIT press, 1993.
36 D. Huber, O. Carmichael, M. Hebert, "3-D Map Reconstruction from Range Data" *Proc. IEEE International Conference on Robotics and Automation*, 2000.
37 J.J. Koenderink, A.J. van Doorn, "Representation of Local Geometry in the Visual System" *Biological Cybernetics*, 55:367-375, 1987.
38 S. Laveau, O.D. Faugeras "3-D scene representation as a collection of images" *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, 1994.
39 P. Montesinos, V. Gouet, R. Deriche, D. Pele, "Differential Invariants for Color Images" *Proc. 14th International Conference on Pattern Recognition*, 1998.
40 M. Pollefeys, R. Koch, L. Van Gool, "A Simple and Efficient Rectification Method for General Motion*" Proc. IEEE International Conference on Computer Vision*, Corfu, 1999.
41 S. Roy, J. Meunier, I. Cox, "Cylindrical Rectification to Minimize Epipolar Distortion*"Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 1997.
42 C. Schmid, R. Mohr, "Local Grayvalue Invariants for Image Retrieval*" IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530-534, 1997.
43 C. Schmid, R. Mohr, "Comparing and Evaluating Interest Points" *Proc. International Conference on Computer Vision*, 1997.
44 P. Sturm and R. Hartley, "Triangulation" *Computer Vision and Image Understanding*, 68(2), 1997.
45 Z. Zhang, R. Deriche, O. Faugeras, Q.T. Luong, "A Robust Technique for Matching Two Uncalibrated Images Through the Recovery of the Unknown Epipolar Geometry" *Artificial Intelligence Journal*, Vol. 78, 1995.

**Proceedings of the 2001 IEEE/RSJ**
**International Conference on Intelligent Robots and Systems**
**Maui, Hawaii, USA, Oct. 29 - Nov. 03, 2001**

# Fault Tolerant Localization for Teams of Distributed Robots

Renato Tinós
*Department of Electrical Engineering*
*EESC*
*University of São Paulo*
*São Carlos, SP 13560-970, Brazil*
*tinos@sel.eesc.sc.usp.br*

Luis E. Navarro-Serment and Christiaan J.J. Paredis
*Institute for Complex Engineered Systems*
*Dept. of Electrical and Computer Engineering*
*Carnegie Mellon University*
*Pittsburgh, Pennsylvania 15213-3890*
*{lenscmu, paredis}@cmu.edu*

## Abstract

*To combine sensor information from distributed robot teams, it is critical to know the locations of all the robots relative to each other. This paper presents a novel fault tolerant localization algorithm developed for centimeter-scale robots, called Millibots. To determine their locations, the Millibots measure the distances between themselves with an ultrasonic distance sensor. They then combine these distance measurements with dead reckoning in a maximum likelihood estimator.*
*The focus of this paper is on detecting and isolating measurement faults that commonly occur in this localization system. Such failures include dead reckoning errors when the robots collide with undetected obstacles, and distance measurement errors due to destructive interference between direct and multi-path ultrasound wave fronts.*
*Simulations show that the fault tolerance algorithm accurately detects erroneous measurements and significantly improves the reliability and accuracy of the localization system.*

## 1. Introduction and Related Work

No longer confined to industrial applications, robots are more often entering the human environment. Toy robots, robotic wheelchairs, surgical robots, and robots in hospitals and nursing homes all come in close contact with humans. Some research suggests that, by 2010, the number of robots in homes will reach 5 million [5]. As a result, a failure in a robotic system can not only cause unacceptable economic losses but also put the safety of the people in its environment at risk [18].

This situation is aggravated by the fact that robot failures are relatively common. Even in well-structured industrial environments, the recorded mean time to failure for manipulators ranges from only 500 to 2500 hours [4]. We expect that robot failures will be even more common in household robots for which, most likely, rigorous preventive maintenance schedules will not be enforced. According to Parker [15], it is due to the lack of research on fault tolerance and adaptivity of robot teams that robot autonomy and multi-robot cooperation have not yet been adequately demonstrated.

This paper investigates fault tolerance issues in a group of robot systems that are particularly vulnerable to failures. The CMU Millibots are very small (7x7x7cm) and inexpensive robots that contain little or no redundancy within each robot but rely on collaboration within a team to identify and overcome failures. These Millibots provide a good test-bed for fault tolerance as they reflect the limited reliability and capabilities of future inexpensive household robots.

Fault tolerance is usually achieved in two steps: The system first detects and isolates the faults, after which it reconfigures itself to overcome the faults. Generally, for individual mobile robots, this approach requires considerable redundancy in sensing, actuation, communication, and computation, resulting in large, complex, and expensive systems. For multi-robot systems, redundancy is also available at the team level. For instance, sensing capabilities on one robot may be replaced by a combination of sensing modalities on other robots. As a result, fault tolerance can still be implemented for robots with limited capabilities (and possibly without any redundancy) such as the Millibots.

In this paper, the problem of fault tolerance in the localization system of the CMU Millibots is addressed. The objective of the Millibots is to collaboratively map and explore an unknown environment. When a robot moves, its new position is estimated from a combination of dead reckoning measurements and distance measurements between robots. Faults in these measurements produce incorrect position estimates, and correspondingly, errors in the maps of the environment. This paper presents a method to detect and overcome such errors, based on the

information redundancy in the dead reckoning and distance measurements.

## 2. The Millibots

The Millibots are configured from modular components including communication, computation, mobility, camera, sonar, and IR modules [10]. Assembling different types of modules creates specialized robots that collaborate to accomplish a given task. Because of their small size (7x7x7 cm), the computational and sensing capabilities of Millibots are limited. Higher-level functions such as mapping and localization are provided by a larger robot or team leader.

The knowledge of the position and orientation of each Millibot is crucial to achieve accurate mapping and exploration of the environment. Conventional localization systems, based on dead reckoning, GPS, landmark recognition or map-based positioning [17], do not offer a viable solution due to limitations in size, power, or sensing of the Millibots. To overcome these problems, a novel method was developed that utilizes dead reckoning and ultrasonic distance measurements between robots [14].

The Millibot localization system is based on trilateration [2], i.e., determination of the position based on distance measurements to known landmarks or beacons [11, 12]. GPS is an example of a trilateration system; the position of a GPS unit on earth is calculated from distance measurements to satellites in space. Similarly, the Millibot localization system determines the position of each robot based on distance measurements to stationary robots with known positions.

As is illustrated in Figure 1, the distance between two robots is measured using synchronized ultrasound and RF pulses. A conical reflector mounted above a low-cost transducer allows the Millibots to detect and transmit

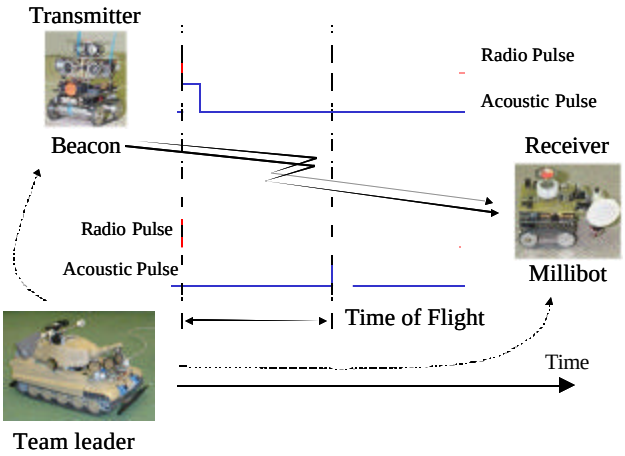



**Figure 1: Ultrasonic distance measurement.**

ultrasonic pulses in any direction [10]. Periodically, each robot that serves as a beacon emits simultaneously a radio frequency (RF) pulse and an ultrasonic pulse. Using the RF pulse for synchronization, the distance to the beacon is measured as the time-of-flight of the ultrasonic pulse multiplied by the speed of sound (343m/s at 20°C). The team leader coordinates the pinging sequence to ensure that beacon signals from multiple robots do not interfere with one another. To improve the accuracy, this procedure is repeated several times and the sample mean is utilized to estimate the distance to the beacon. A similar multi-modal localization system is being developed by Girod *et al.* [7, 8].

All the Millibots transmit their distance measurements to the team leader who calculates the new robot positions. A maximum likelihood algorithm determines the most likely position of the robot given the measured distances to the current beacons. Assuming that the dead reckoning and distance measurements are normally distributed random variables, the likelihood of being located at a position $(x,y)$ is given by

$$P(x,y \mid x_d, y_d, \bar{r}_b(1), \bar{r}_b(2), \cdots, \bar{r}_b(m)) = N(x - x_d, s_x^2) N(y - y_d, s_y^2) \prod_{i=1}^{m} N(r(i) - \bar{r}_b(i), s_b^2) \quad (1)$$

where $N(p,s^2)$ is a normal distribution with zero mean and variance of $s^2$ evaluated at $p$, $(x_d,y_d)$ is the position measured through dead reckoning, $r(i)$ is the distance from the beacon $i$ to the Millibot, $m$ is the number of beacons, and $\bar{r}_b(i)$ is the sample mean of the distance measurements from beacon $i$ to the Millibot.

The estimated new position of the Millibot is given by the value of $(x,y)$ that maximizes the probability density function in Equation (1), and is computed using the BFGS non-linear optimization algorithm [6]. The algorithm is initialized with the dead reckoning estimate $(x_d,y_d)$ or an estimate based on the closed form trilateration expression derived in [13]. In general, only a few iterations are necessary to reach the optimum value because of its proximity to the starting point.

It seems that similar results could have been achieved with a Kalman filter, as is commonly used in many other localization schemes [11, 12]. However, because the state equations are non-linear, they require an *Extended* Kalman Filter (EKF) obtained by linearizing the state equations around the estimated state. This linearization is only an acceptable approximation in a small neighborhood around the current state, so that the EKF only produces good results when the state varies slowly relative to the rate at which sensor data is incorporated. Because the Millibots use their sonar beacons only after relatively large steps, this condition is violated, causing the Kalman filter to diverge quickly.

On the other hand, the maximum likelihood approach described above does not assume linearity of the state equations allowing the optimization to converge to the true optimal estimate. For that reason and only in this application, it tends to perform better than Kalman filters, even though it does not take any cross-correlations or uncertainty in the beacon positions into account as would have been possible with an EKF.

## 3. Fault Modes and Effects Analysis

Although the localization algorithm described above has the potential to provide very accurate position estimates [14], practice has shown that it is susceptible to multiple failures, some of which occur relatively often. Before developing a fault tolerance scheme, we analyze these failure modes using Fault Modes and Effects Analysis (FMEA) [1].

### 3.1. Incorrect ultrasonic distance measurements

We have identified three different causes for erroneous distance measurements.

A first failure occurs when the ultrasonic pulse is either not emitted by the beacon or not received by the Millibot. This could happen due to a failure in the transducers, the circuitry, or the communication with the team leader. These faults do not occur often and result in a clearly identifiable effect: failing to register an ultrasonic pulse.

Incorrect distance measurements also result when there is an obstacle between the beacon and the Millibot blocking the ultrasonic pulse. The effect, in this case, is more complicated because an ultrasonic pulse that bounces around the obstacle (multi-path) may still be detected, resulting in a distance measurement larger than the actual distance.

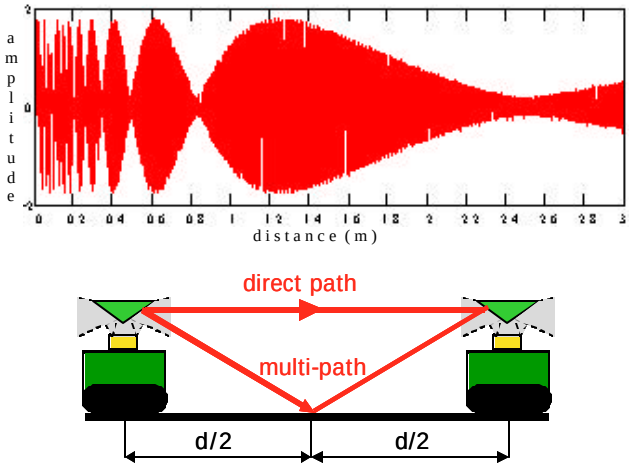



**Figure 2: Destructive interference.**

The same effect can occur due to destructive interference. As is illustrated in Figure 2, at certain distances between the robots, the wave propagated in the direct path interferes destructively with the wave that bounces off the floor. For the Millibots this failure mode is especially pronounced at distances of 0.5m and 0.8m (the interference at 2.5m, shown in Figure 2, is outside the range of the sensor). The effect, again, is that a secondary echo is measured instead, resulting in too large a distance measurement.

### 3.2. Incorrect dead reckoning measurements

The motion commands for the Millibots are executed by a local PID controller that receives feedback from optical encoders. Although the control loop is sufficiently accurate to ensure that the motors actually execute the desired movement (within a certain variance due to thread slippage), the following failures can still occur.

A first group of faults is a result of hardware failures of the actuators, mechanical transmissions, wheels, encoders or controllers. These faults are rather uncommon and result in a movement to the wrong position, no movement at all, or continuous movement without stopping. A more common failure mode occurs when the Millibots run into an undetected obstacle. The effect varies from stopping before reaching the desired position to falling over. Due to thread slippage, these failures cannot be detected through the encoder readings.

In conclusion, both groups of faults result in a discrepancy between the actual position and the estimated position based on dead reckoning.

## 4. Fault Tolerance

As pointed out in the previous section, faults affecting localization occur commonly in the Millibot system—especially the destructive interference failure mode occurs often. To ensure accurate position estimation, it is critical that these faults are detected and isolated so that they can be taken into account by the estimation algorithm. Such a fault tolerance scheme is presented in this section.

### 4.1. Fault Detection and Isolation

Past research in fault detection and isolation (FDI) has focused on faults in individual mobile robots with redundant sensors [9, 16]. For example, encoder readings are compared with integrated gyroscope measurements to detect faulty estimates of the robot orientation. The Millibots, however, do not have this level of sensing redundancy. Instead, they take advantage of the information redundancy in the combined dead reckoning and ultrasonic distance measurements for the entire team of robots.

Based on the dead reckoning information, we can compute the expected distance from the moving Millibot to each of the beacon Millibots. Assuming that the distance traveled and the distance to the beacons is relatively large, this expected distance is approximately normally distributed. The ultrasonic distance measurement is also a sample from a population that is approximately normally distributed (the discretization error in the ultrasonic sensor is much smaller than the measurement error). Our FDI scheme is based on statistical tests that verify whether the two normally distributed distance measurements (based on dead reckoning and ultrasonic pulses) are consistent, i.e., have the same expected value. If they are not consistent, a fault has occurred.

The test is based on the following statistical properties. Consider two random variables, $(x_1, x_2)$, from two different populations both with a normal distribution. It can be shown that the difference of the sample means, $(\bar{x}_1 - \bar{x}_2)$, is also normally distributed with a mean and variance equal to [3]:

$$m_{\bar{x}_1-\bar{x}_2} = m_1 - m_2 \qquad s_{\bar{x}_1-\bar{x}_2} = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \qquad (2)$$

where $n_1$ and $n_2$ are the sample sizes. Or, as a direct corollary:

$$z = \frac{(\bar{x}_1 - \bar{x}_2) - m_{\bar{x}_1-\bar{x}_2}}{s_{\bar{x}_1-\bar{x}_2}} = \frac{(\bar{x}_1 - \bar{x}_2) - (m_1 - m_2)}{\sqrt{s_1^2/n_1 + s_2^2/n_2}} \qquad (3)$$

is normally distributed with zero mean and unit variance.

To test whether both populations, $x_1$ and $x_2$, have the same expected value $(m_1 = m_2)$, we can use the following hypothesis:

$$H_0\text{: } m_1 - m_2 = 0 \text{ if } -z_{1-\alpha/2} \le z \le z_{1-\alpha/2} \qquad (4)$$

where $\alpha$ is the level of significance, and $z$ is

$$z = \frac{(\bar{x}_1 - \bar{x}_2)}{\sqrt{s_1^2/n_1 + s_2^2/n_2}}\ . \qquad (5)$$

An alternative hypothesis is

$$H_1\text{: } m_1 - m_2 \ne 0 \text{ if } z \le -z_{1-\alpha/2} \text{ or } z \ge z_{1-\alpha/2}\,. \qquad (6)$$

For a given $\alpha$, the value of $z_{1-\alpha/2}$ can be found in the standard normal tables.

We now apply this test to the two distance measurements: $r_d(i)$, the distance from the beacon $i$ to the Millibot based on dead reckoning, and $r_b(i)$, the corresponding ultrasonic distance measurement. The hypothesis variable, $z(i)$, is then:

$$z(i) = \frac{r_d(i) - \bar{r}_b(i)}{\sqrt{s_{r_d(i)}^2 + s_b^2/n_b(i)}} \quad , \quad i = 1, 2, \cdots, m \qquad (7)$$

where $n_b(i)$ is the number of independent ultrasonic measurements by beacon $i$. Assuming that the variances on the position coordinates, $x$ and $y$, are small, the variance $s_{r_d(i)}^2$ can be obtained from:

$$s_{r_d(i)}^2 = \left(\frac{\partial r_d(i)}{\partial x}\right)^2 s_x^2 + \left(\frac{\partial r_d(i)}{\partial y}\right)^2 s_y^2\,. \qquad (8)$$

**Table 1: Hypothesis tests for the FDI procedure.**

| Hyp. | Fault | Region |
|---|---|---|
| $H_0$ | No faults | $-z_{1-\alpha/2} \le z(i) \le z_{1-\alpha/2}$ for i = 1, 2, …, m |
| $H_1$ | Incorrect ultrasonic distance measurement for beacon j | $-z_{1-\alpha/2} \ge z(j)$, $-z_{1-\alpha/2} \le z(i) \le z_{1-\alpha/2}$ for i = 1, 2, …, m, i ≠ j |
| $H_2$ | Incorrect dead reckoning information | otherwise |

Table 1 summarizes the different fault scenarios and corresponding statistical hypotheses: If no faults occur, the variable $z(i)$ of every beacon is small, confirming the hypothesis $H_0$. If an error in the ultrasonic distance measurement between the Millibot and beacon $j$ occurs (hypothesis $H_1$), $z(j)$ is negative and large (negative because an erroneous measurement is always larger than the true distance, as explained in Section 3). Since we assume that only one error occurs at a time, the other variables, $z(i)$ with $i \ne j$, will all be small. If this is not the case, then we conclude that an error in the dead reckoning measurement has occurred (hypothesis $H_2$).

## 4.2. Reconfiguration

After the fault has been detected and isolated, the localization algorithm is easily reconfigured by ignoring the erroneous measurement. If an incorrect distance measurement for beacon $j$ is detected, Equation (1) is modified to

$$P(x, y \mid x_d, y_d, \bar{r}_b(1), \bar{r}_b(2), \cdots, \bar{r}_b(m)) = N(x - x_d, s_x^2) N(y - y_d, s_y^2) \prod_{\substack{i=1 \\ i \ne j}}^{m} N(r(i) - \bar{r}_b(i), s_b^2) \qquad (9)$$

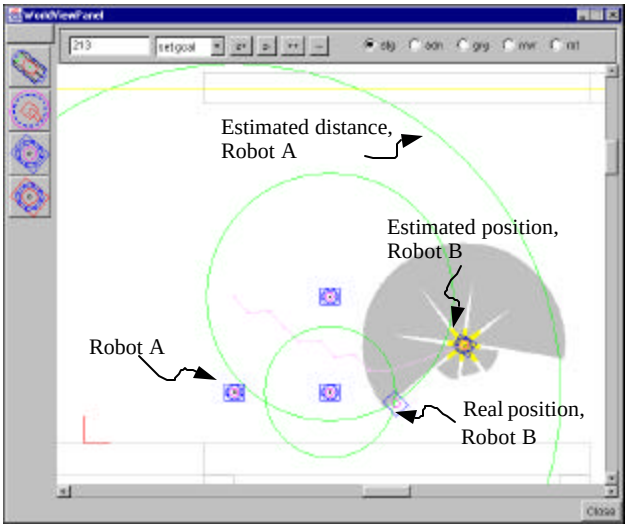



**Figure 3: Faulty position estimation for Robot B when an erroneous ultrasonic measurement occurred with respect to beacon A.**

If incorrect dead reckoning measurements are detected, Equation (1) becomes

$$P(x, y \mid \bar{r}_b(1), \bar{r}_b(2), \cdots, \bar{r}_b(m)) = \prod_{i=1}^{m} N\left(r(i) - \bar{r}_b(i), s_b^2\right) \quad (10)$$

in which the dead reckoning information is not utilized. This requires that at least three Millibots serve as beacons for a unique maximum to exist.

## 5. Results

A series of experiments have been conducted to test the effectiveness of a team of Millibots to explore and map a given area [10]. In these experiments, a human operator who controlled the Millibot team could plan the individual robot motions to avoid distances at which destructive interference occurs. The robot motions were also planned to avoid ill-conditioned configurations, such as collinear beacons. The experiments showed that the localization algorithm performs well when no faulty distance measurements occur. However, it becomes very difficult to avoid these faults (i.e. avoid collinearity and destructive interference) with more than four robots on a team.

We performed several simulations to test the fault tolerance algorithms. The Figures 3 and 4 show snapshots from a GUI that controls four Millibots in a mapping task. The simulations performed without the fault tolerance system (Figure 3) show that significant errors result when incorrect distance estimates are considered in the localization algorithm. The gray region around the robot B indicates the area covered by the sonar sensors used to detect objects; errors in the estimated position translate into significant mapping errors.

The simulations in which the fault tolerance system was active show that the faults described in Section 3 could be detected correctly. One such scenario is illustrated in Figure 4. In this simulation, destructive interference occurred in the distance measurement between the robots A and B. The values of $z$ in Equation (7) were -6.03 for beacon A, and 0.41 and –0.04 for the other beacons. For a significance of $a$=0.01, $z_{1-a/2}$ equals 2.326 resulting in a confirmation of hypothesis $H_1$ (since –6.03 < -2.326). As a result, the localization algorithm ignored the erroneous distance measurement to robot A and estimated the robot position based only on the dead reckoning information and the distance measurements to the other two robots.

The fault tolerance algorithm has the additional advantages that it is not computationally expensive, and that it provides additional quantitative information to the human operator with respect to the performance of the localization system. This information can be used to improve the individual robot motion operation.

## 6. Conclusion

Due to destructive interference of ultrasonic pulses or collisions with undetected obstacles, faults occur commonly in the Millibot localization system. It is therefore important to employ fault tolerance mechanisms to improve its reliability and accuracy.

This paper introduced an FDI system based on statistical hypothesis testing that can identify which of the measurements (distance measurements and dead

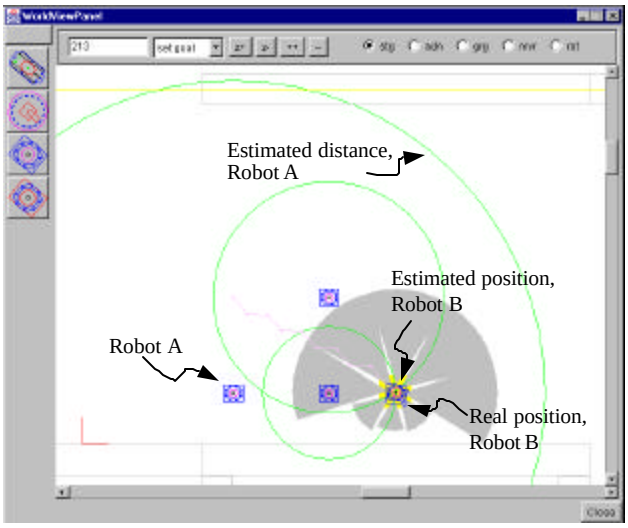



**Figure 4: Correct position estimate for robot B. The FDI procedure detected the erroneous distance measurement with respect to beacon A.**

reckoning) is incorrect. Because of the structure of the maximum likelihood estimator, the localization algorithm can be easily modified to omit these erroneous measurements. Simulations showed that the fault tolerance procedure successfully detected and compensated for incorrect measurements, thus improving the accuracy and reliability of the localization system.

In the future, the fault tolerance procedure should be tested with the real Millibots. In particular, research should focus on incorporating of fault tolerance procedures into the real-time Millibot path-planning system. The problem of multiple simultaneous faults should be addressed too, along with a characterization of the reliability of the localization system under this scenario.

## Acknowledgements

The authors would like to thank Robert Grabowski, who developed the Millibot simulator, for his contributions. The Millibots project is funded in part by the Distributed Robotics program of DARPA/ETO under contract DABT63-97-1-0003 and, by the Institute of Complex Engineered Systems at Carnegie Mellon University. Renato Tinós is supported by FAPESP under grant 98/15732-5.

## 7. References

[1] Birolini, A., *Reliability Engineering: Theory and Practice*, Third ed. New York: Springer, 1999.

[2] Borenstein, J., Everett, H. R., and Feng, L., *Navigating Mobile Robots: Sensors and Techniques*. Wellesley, MA: A.K. Peters, 1996.

[3] Chase, W. and Bown, F., *General Statistics*, Fourth ed. New York: John Wiley & Sons, 2000.

[4] Dhillon, B. S., *Robot Reliability and Safety*. New York: Springer-Verlag, 1991.

[5] Dhillon, B. S. and Fashandi, A. R. M., "Robotic Systems Probabilistic Analysis," *Microelectronics and Reliability*, vol. 37, pp. 211-224, 1997.

[6] Fletcher, R., *Practical Methods of Optimization*, Second ed. New York: John Wiley & Sons, 1987.

[7] Girod, L., "Development and Characterization of an Acoustic Rangefinder," University of California, Los Angeles, Los Angeles, CA, Technical Report USC-CS-00-728, April 2000.

[8] Girod, L. and Estrin, D., "Robust Range Estimation Using Acoustic and Multimodal Sensing," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, Maui, Hawaii, 2001.

[9] Goel, P., Dedeoglu, G., Roumeliotis, S. I., and Sukhatme, G. S., "Fault Detection and Identification in a Mobile Robot Using Multiple Model Estimation and Neural Network," *IEEE International Conference on Robotics and Automation*, Leuven, Belgium, 1998.

[10] Grabowski, R., Navarro-Serment, L. E., Paredis, C. J. J., and Khosla, P. K., "Heterogeneous Teams of Modular Robots for Mapping and Exploration," *Autonomous Robots*, vol. 8, pp. 293-308, 2000.

[11] Kleeman, L., "Optimal Estimation of Position and Heading for Mobile Robots Using Ultrasonic Beacons and Dead-Reckoning," *1992 IEEE International Conference on Robotics and Automation*, Nice, France, pp. 2582-2587, 1992.

[12] Leonard, J. F. and Durrant-Whyte, H. F., "Mobile Robot Localization by Tracking Geometric Beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 376-382, 1991.

[13] Manolakis, D. E., "Efficient Solution and Performance Analysis of 3-D Position Estimation by Trilateration," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, pp. 1239-1248, 1996.

[14] Navarro-Serment, L. E., Paredis, C. J. J., and Khosla, P. K., "A Beacon System for the Localization of Distributed Robotic Teams," *International Conference on Field and Service Robotics*, Pittsburgh, pp. 232-237, 1999.

[15] Parker, L. E., "ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 220-240, 1998.

[16] Roumeliotis, S. I., Sukhatme, G. S., and Bekey, G. A., "Sensor Fault Detection and Identification in a Mobile Robot," *1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Victoria, Canada, pp. 1383-1387, 1998.

[17] Stuck, E. R., Manz, A., Green, D. A., and Elgazzar, S., "Map Updating and Path Planning for Real-Time Mobile Robot Navigation," *1994 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Munich, Germany, pp. 753-360, 1994.

[18] Visinsky, M. L., Cavallaro, J. R., and Walker, I. D., "Robotic Fault Detection and Fault Tolerance: A Survey," *Reliability Engineering and System Safety*, vol. 46, pp. 139-158, 1994.