In *Experimental Robotics VII, LNCIS 271*
D. Rus and S. Singh, editors, pages 353-362. Springer-Verlag Berlin Heidelberg, 2001.

Principled Communication for Dynamic Multi-Robot Task Allocation

Brian P. Gerkey and Maja J Matarić
Robotics Research Labs
University of Southern California
Los Angeles, CA, USA
{bgerkey|mataric}@cs.usc.edu

Abstract: In the pursuit of an efficient cooperative multi-robot system, the researcher must eventually answer the question "how should robots communicate?"; a natural way to attack this question is to decompose it into three simpler corollaries: "what should robots communicate?", "when should they communicate?" and "with whom should they communicate?". In this paper, we propose answers to these questions in the form of a general framework for inter-robot communication and, more specifically, advocate its use in dynamic task allocation for teams of cooperative mobile robots. We base our communication model on *publish/subscribe messaging* and validate our system by using it in a tightly-coupled multi-robot manipulation task and a loosely-coupled long-term experiment involving many robots concurrently executing different tasks.

1. Introduction

We posit that the long-term goal for collective robotics is to have a decentralized collection of cooperative robots that are anonymously taskable. That is, irrespective of which and how many robots are available and what if any differences exist among them, the group should be viewed as a pool of resources to whom work assignments, or tasks, can be delegated. For a variety of practical reasons, the robots must be individually autonomous, and so the delegation of tasks must be accomplished in a distributed manner such that this autonomy is ensured. Further, we are striving toward *long-term* autonomy, in which, for long periods of time (days or weeks or even longer), each robot is a self-sufficient entity either executing a task or awaiting a new assignment; human intervention should be required only in severe failure situations.

In our approach, each robot has a set of individual resources, or capabilities, which it can use to achieve tasks. These capabilities, which could be physical, like having a gripper, or ethereal, like having knowledge of the topological layout of a building, can and do vary over time. Thus, even if two robots are initially identically physically configured, they will, through experience and interaction with the world, become heterogeneous with respect to both their capabilities and their fitness for any given task. Each robot might have many different resources, and thus be capable of a variety of tasks. Also, any robot might fail at any time in a variety of ways. The problem we address here is how to intelligently allocate tasks, both simple and complex, to such a

dynamic, heterogeneous group of physically-embodied individuals working in an unpredictable environment.

Toward this end, we have implemented and tested a novel task allocation system which we call MURDOCH. This system is based on a principled, resource-centric, *publish/subscribe* communication model and makes extensive (but efficient) use of explicit inter-robot communication. Since our goal is not to exploit resources in a globally optimal fashion, but rather to investigate practical methods for allocating tasks to groups of autonomous and heterogeneous physical robots, we have built MURDOCH as a completely distributed system. As such, it offers a distributed approximation to a global optimum of resource usage which is equivalent to an instantaneous greedy scheduler.

We have tested this system in two very different domains: a tightly-coupled multi-robot physical manipulation task (see Section 6.1) and a loosely-coupled many-robot experiment in long-term autonomy (see Section 6.2).

2. Related Work

There has been a great deal of work on architectures for automated cooperation among agents. The Open Agent Architecture (OAA) [1] and RETSINA [2] are two such architectures. In both, the focus is on providing a maximally general environment in which very different agents, such as user interfaces and legacy database management systems, can interact and coordinate. Since we are concerned specifically with task allocation for physical mobile robots, we do not require the overhead (such as ontology specifications) that allows these systems to be so general. There is a component of task allocation among their agents, although the tasks are purely informational, rather than physical, in nature. Both architectures accomplish task allocation through a broker (called *facilitator* and *matchmaker*, respectively) which matches new task requests with agents that have previously advertised relevant capabilities. In MURDOCH, we completely distribute the matchmaking process and thus do away with the centralized broker. A different approach is taken in STEAM [3]; this model-based system relies on each agent's explicitly tracking the actions of both itself and its team and then applying *joint intention* principles to the resulting world model in order to make decisions. While this approach has been validated on software agents, it is not clear how, in a physically embodied world, an autonomous agent could gather and interpret the necessary data to perform the requisite tracking and modeling.

In stark contrast to these very general architectures are ALLIANCE [4] and BLE [5]. These two special-purpose systems provide coordination among multiple robots by having the execution of a behavior on one robot directly inhibit the execution of the same behavior on another. Although it is not strictly necessary, in practice robots in these systems share an identical internal (behavior-based) structure and the inhibition relationships between the relevant behaviors are hand-coded. Further, while a system built in either ALLIANCE or BLE may be well-suited to performing a single task, neither architecture has been demonstrated as being easily retaskable.

Whereas most of the architectures described above fundamentally communicate by point-to-point message passing, an alternate method is the broadcast-oriented blackboard model [6]. The standard blackboard system has many sim-

ilarities to the publish/subscribe system used in MURDOCH. In fact, one might say that a publish/subscribe system is an instantaneous blackboard in which no central state need ever be kept, with the tradeoff that the past history of the blackboard is lost. This is an acceptable tradeoff in our domain, since old information quickly becomes irrelevant in physical, real-time systems.

In order to allocate a given task, we use a simple auction in which each capable agent evaluates its own fitness for the task. A similar approach for deriving supply chains in a producer/consumer problem is described in [7].

3. The Communication Model

In implementing distributed control systems for teams of robots, researchers typically resort to *ad hoc* communication strategies. These specialized strategies are often implemented as hand-crafted, task-specific communication graphs. For example, in [4] and [5], communication channels are explicitly created among individual behaviors on the different robots. While this model is well-suited to the design of some special-purpose, single-task systems, it has not been demonstrated for the general case of controlling a large population, in which the members dynamically form teams to accomplish different tasks as they are presented to the system.

As an alternative to such special-purpose systems, we propose a principled communication model based on *publish/subscribe messaging*. Also known as *dissemination-oriented communication* and the *announce/listen metaphor* [8], publish/subscribe messaging is a commercially viable ([9]) message delivery paradigm that has been studied in a distributed systems context ([10]).

The unifying concept of publish/subscribe systems is that *messages are addressed by content rather than by destination.* This idea, often called *subject-based addressing*, is used to divide the network into a loosely-coupled association of anonymous data producers and data consumers. A data producer simply tags a message with a subject describing its content, and "publishes" it onto the network; any data consumers who have registered interest in that subject by "subscribing" will automatically receive the message. Data producers need not have any knowledge of which consumers, if any, are receiving their messages, and vice versa. This kind of communication represents a fundamental departure from the traditional communication model, in which each message is unicast from the sender to a single receiver at a specific known destination. As an aside, we have tailored this idea slightly so that when a message is published, it is addressed to a set of subjects, rather than just one. A data consumer will receive a message if the subjects in the message comprise any subset of the consumer's current subscription list. Although we have not optimized the subset matching algorithm in our implementation, others have investigated the topic [11].

4. Murdoch

Several important but hopefully admissible assumptions were made in the design of MURDOCH. First, the robots have a reasonable communication system. By "reasonable", we mean that, although the communication paths among the robots need not be reliable, they should work most of the time and they should

easily provide the meager bandwidth required by our system. At a higher level, we assume that the robots share a common vocabulary. Thus, when one robot communicates about a task called **push-box-on-the-right-end**, the others know what that means. Now, this assumption does not imply homogeneity, for the mapping from the task name **push-box-on-the-right-end** to an internal control system for actually achieving it could be different on each robot, either out of necessity (e.g., one robot has legs and the other wheels), or by design (e.g., we are comparing different pushing algorithms simultaneously). We also expect the robots to be fundamentally cooperative and honest. For example, when evaluating a fitness metric for a potential task, each robot faithfully reports its score and gracefully withdraws when beaten. Of course, since the fitness metric is likely based on the robot's noisy and imperfect sensory input, the reported score may not actually be "correct" (see Section 4.3).

4.1. Subject Namespace

The first step in creating a publish/subscribe system is designating the semantics of the subject namespace. Analogous to deciding the layout of a database, the interpretation of subjects will heavily influence the rest of the system. In MURDOCH, since we are allocating tasks among a group of potentially heterogeneous robots, we use subjects to represent their "resources". Resources can be physical devices (e.g., **camera**, **gripper**, **sonar**), higher-level capabilities (e.g., **mobile**, **door-opener**) or abstracted notions of current state (e.g., **idle**, **have-puck**, **currently-pushing-box**). Thus, if we have a task that involves going to some physical location and observing it, we can reach the appropriate robots by addressing a message to the set **{mobile camera idle}**. Since messages are addressed to subjects and subjects represent resources, all inter-robot communication will necessarily be resource-centric, which we believe to be fundamental in achieving our goals. The robots never interact with each other by name and in fact have no explicit knowledge of each others' existence; rather they only communicate about tasks and all messages are addressed in terms of the resources required to do the tasks.

4.2. Task Structure

Since we are concerned with task allocation, we must of course choose a representational structure in which we can describe (and the robots can understand) a given task. Although we do not currently use the Hierarchical Task Network (HTN) formalism [12] (we will soon transition to an HTN representation), the flexible hierarchical scheme that we do use is quite similar. Much like HTN's *compound tasks*, we have *high-level tasks*; these tasks are sufficiently high-level that a human user can readily understand and reason about them. High-level tasks can be (and often are) composed of what we simply call *tasks*; a task is an atomic unit of computation and control that a single robot will execute. Our tasks are akin sometimes to HTN *compound tasks* and sometimes to HTN *atomic tasks*, since the execution of our tasks can involve allocating yet more tasks.

Since in this work we are investigating task allocation, not task decomposition, we delegate to the task designer the work of defining a high-level task

in the form accepted by MURDOCH[1]. However, it is important to note that in practice the tasks we want the robots to achieve are structurally simple enough (typically just a single layer of decomposition even for a tightly-coupled task) that the designer is not mired in exploration of a vast space of combinations.

We simplify the job of writing the single-robot tasks, like **push-right-end-of-box** through the use of basis behaviors [14]. In fact, we follow a behavior-based model ([15], [14]) for controlling all the sensors and actuators on our robots. However, we do not require that the robots be controlled in a behavior-based manner; any control architecture that allows for the starting and stopping of task execution will fit within our model. Our behaviors are low-level controllers, such as **avoid-obstacles-with-sonar** and **visual-servo-to-color** and are each implemented as a separate operating system thread. A task, then, is defined in terms of a concurrent instantiation of a collection of properly parameterized behaviors. For example, we can define **push-right-end-of-box** as particular instances of **avoid-obstacles-with-sonar** and **visual-servo-to-color** (our box is brightly colored) and we can define **goto-goal** as a differently parameterized instance of **avoid-obstacles-with-sonar** paired with **servo-to-location**. While writing these individual behaviors is not a trivial matter, we believe that having a library of robust and parameterizable basis behaviors is well worth the work involved as the result is a natural way of specifying tasks.

4.3. Negotiation

At the heart of MURDOCH lies a simple distributed negotiation protocol that allocates tasks one by one via a sequence of one-round auctions. The process is triggered by the introduction of a task to the system. The task could be introduced in many ways, including a human user, a `cron`-style alarm, or an already ongoing task. In every case, the first step is for an agent (which is working on behalf of a user, alarm, or task) to publish an announcement message for the task. This message contains details about the task, such as its name, length, and a new subject on which to negotiate it. The announcement message is addressed to the set of subjects which represent the resources required to execute the task; thus only those robots currently capable of the task will receive the message. Of course, there may be more than one capable robot available for a single task and we need a method for deciding among them; in fact, this decision process is the very basis of achieving sensible task allocation. For this purpose, we employ metric functions, or simply *metrics*.

Metrics can take many forms, with the restriction that each one, when evaluated in the context of a specific robot, should return a scalar "score" representing that robot's fitness for the task. A metric is usually defined as some function of the robot's current state, although in general, a metric could perform any arbitrary computation, including inter-robot communication. As an example, if the task under consideration is to go to a certain location and pick up an object, one possible metric is to compute the Cartesian distance from the robot's current position to the goal position, with a shorter distance being

[1] Alternatively, an off-line planner, such as the one described in [13], endowed with knowledge of the preconditions, postconditions and interdependencies of the tasks involved could be employed here. Note that the planner would not be deriving specific motion sequences, but rather overall task structure; the behaviors themselves generate *in situ* trajectories.

better. Multiple metrics can be defined for a single task, with the final score being some combination of the individual scores; we have experimented with combining metrics through both simple sums and weighted sums, the latter of which provides for a prioritization of metrics. It is important to note that metrics, being functions of each robot's own sensor data, may not accurately represent the current state of the robots, possibly resulting in a non-optimal allocation of the task. Since finding an optimal allocation would require gathering global data, guaranteeing its accuracy, and centralizing control, we find our metric-based distributed approximation to be a parsimonious alternative.

Upon receipt of a task announcement message, each capable robot participates in a one-round auction, determining its fitness by evaluating the indicated metrics and broadcasting its score back to the others. Everyone immediately knows who is best suited (i.e., who won the auction) and so the losers can go back to listening for new tasks while the winner begins the task. Since each task will always be claimed by the most capable robot at the time, MURDOCH acts as an instantaneous greedy task scheduler. Thus we suffer from the well-known problems of greedy algorithms; they are manifested in our domain as situations in which, although sufficient resources exist to achieve a given set of tasks, the order in which they are presented causes resources to be exploited in a non-optimal manner such that not all the tasks are actually achieved. Centralized broker and matchmaker systems avoid this pitfall by analyzing concurrent tasks before allocating them; of course, this kind of planning will not help in what we view as the common case in which single tasks are input stochasticly from some outside source, such as a human user. As a distributed alternative, we are investigating the inclusion of simple optimizations in the metrics themselves; for example, a specialist who has few resources to offer could always increase its own score by some amount such that it will defeat a generalist who has more resources and might be put to better use on a later task.

5. Experimental Platform

MURDOCH is implemented on our group of seven ActivMedia Pioneer 2-DX mobile robots. The robots are equipped with a variety of sensors and each is configured differently. The sensors include: ultrasonic rangefinders, laser rangefinders, compasses, color cameras, and tactile bumpers. With regard to actuators, each is equipped with two drive wheels and a passive caster (steering is differential), some have grippers, and some have pan-tilt-zoom camera units.

Internally, each robot houses a Pentium-based computer running Linux; MURDOCH runs on this platform and communicates with the robot via a server[2] developed at the USC Robotics Research Labs. Inter-robot communication is provided by way of wireless Ethernet; the topology is such that every machine (robot or not) on the network can communicate freely with every other machine.

[2]The server, Player [16], runs on the Pioneer's computer and offers a unified TCP socket-based interface to all the sensors and actuators attached to the robot. Player is freely available under the GNU Public License. Consult `http://fnord.usc.edu/player` for more details.

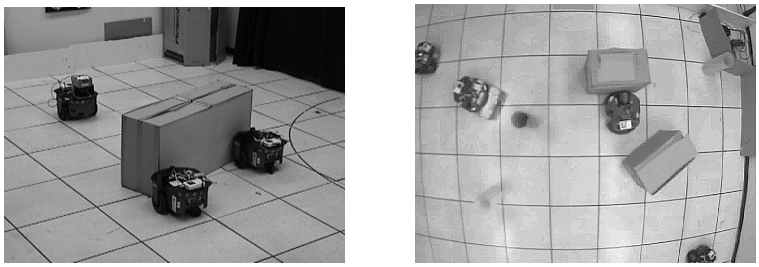

Figure 1. *Our two experimental task domains. In the cooperative box-pushing task (left), the robot on the left (the watcher) is coordinating the actions of the other two robots (pushers). In the other experiment (right), a group of robots concurrently execute a variety of single-agent tasks, including box-pushing and target-tracking.*

6. Experimental Task Domains

We validated our approach in two different task domains, one featuring tightly-coupled coordination of multiple robots engaged in a cooperative box-pushing task, and the other loosely-coupled coordination of many robots executing a collection of independent single-robot tasks. For more information about these tasks, and videos of experiments, consult: `http://robotics.usc.edu/~agents/projects/pub-sub.html`.

6.1. Tightly-Coupled Task Allocation

Our experimental setup (see Figure 1) is as follows: one large box must be moved from a start location to a goal location by the collective pushing efforts of a group of robots. The robots do not have manipulators with which they could grip the box, and the box is large enough and characterized by sufficiently complex physical properties that a single robot cannot predictably manipulate the box by pushing in the middle of a side. We chose a cooperative pushing algorithm inspired by the distributed manipulation work of [17] and [18]. We have two pushers and a single watcher, equipped with a laser rangefinder and color camera. The watcher drives in front of the box and toward the goal, which it finds with the camera, all the while monitoring the orientation of the box with the laser. Depending on the current orientation of the box, the watcher generates the appropriate pushing tasks to rotate and translate the box as necessary. It is important to note that the pushers are not inherently "handed"; they have no preference for pushing a particular end of the box and are in fact both executing the same control program. When a new pushing task is announced, each available pusher executes a vision-based metric that reflects how well-positioned it is to achieve the particular task.

After verifying that the simple case of continuous straight-line pushing was successful, we stressed the system in various ways. First, we moved the goal far enough off to one side that simultaneous action by both pushers would have been insufficient to move the box to the goal; the box had to be rotated as well. In this situation, the watcher senses that the box is out of alignment with respect to the goal and consequently allocates only the appropriate single pushing task which would correct the alignment; they then return to the simple case of straight pushing. We also conducted experiments with a moving goal

and even changing the watcher task dynamically to a different goal. Since the watcher algorithm involves no path planning, but rather constantly reacts to current sensor data, moving and changing goals do not constitute a special case in the algorithm. We were able to reliably lead the whole group (robots and box) along many sufficiently widely curved paths around the lab.

In order to test the fault-tolerance capability of our box-pushing system, we purposefully introduced failure by turning off one pusher while the robots were in the middle of a coordinated box movement. With the removal of one potential pusher, nothing changes with respect to the algorithm that the remaining robots follow; only the observed behavior changes. Since the remaining pusher is the only one capable of new pushing tasks, it continues pushing but switches sides many times as new corrective pushing tasks are allocated by the watcher, and eventually is able to see-saw the box to the goal. Thus, after a robot failure, the system exploits the other resources and degrades naturally to a less efficient pushing method. Of course, when the remaining resources are insufficient for the task, the task is deemed unachievable and an informative error message is returned to the user. However, if resources again become available, they are immediately put to use. We verified this behavior by turning off one robot, letting the other continue for a while, then reintroducing the failed robot on the opposite side from which it had started; the new pusher was automatically brought into the team, and the pushers had swapped roles. In summary, the system properly handled all failure classes for which we tested.

6.2. Loosely-Coupled Task Allocation

In the pursuit of long-term autonomy, we also experimented with MURDOCH in a very different task domain. In this domain (see Figure 1), we ran a large group, consisting of 6 heterogeneous robots and 1 desktop computer connected to an overhead camera, over a long period of time (approximately 3 hours), all the while randomly injecting new tasks of random lengths into the system. Each machine was controlled by a copy of the same program; this program simply queried its host for the list of currently available devices, then made the proper resource subscriptions. For example, the robots equipped with both cameras and lasers subscribed to {`camera laser mobile`}, while the desktop computer only subscribed to {`overhead-camera`}. We used four different tasks. The first, **object-tracking**, requires `camera` and `mobile`. The task is to find and track from a safe distance a certain colored object. The second, **sentry-duty**, requires `camera`, `laser`, and `mobile`. The task is to find a certain colored object, then turn about and remain still, watching for any motion with the laser and setting off an intruder alarm if motion is detected. Our third task, **cleanup**, requires `camera`, `bumpers`, and `mobile`. The task for the robot is to find each small box of a certain color and use its tactile bumpers to push the box to the edge of the room, thereby cleaning the room. The final task, **monitor-object**, requires only `overhead-camera`. The task is to monitor the positions of various colored objects, such as boxes and robots, from the overhead view, and log the information for later review.

Each robot also runs a battery-monitoring behavior that checks the current charge whenever the robot is idle, in between tasks. At that time, if the battery is low enough, the robot will unsubscribe from all subjects (thereby

removing itself from consideration for future tasks) and go to a clearly marked charging station. After charging for some time, the robot is freed to reenter the experiment.

We observed the following system behavior. Over the course of 3 hours, the 7 agents (6 robots and 1 computer) successfully executed 49 tasks and returned to charge their batteries 12 times. The same control program executed on each robot for the length of the experiment, with robots periodically idle (only executing passive collision avoidance), executing some task, or charging. Some of the randomly generated tasks were unachievable due to a lack of resources, because all the capable robots were either charging or otherwise engaged. In these situations, an error was returned, suggesting that the task be reintroduced later. We are currently implementing a smarter user-agent that will persistently reintroduce unachievable tasks on the user's behalf (following a randomized exponential backoff algorithm) until they are allocated. We also purposefully induced failures in this domain. Since the tasks are all single-robot jobs, a robot failure meant that the task had only to be automatically reallocated to another robot. When resources permitted this reallocation, it was done as expected; otherwise the task was deemed unachievable.

7. Conclusions

In this paper, we presented the details of a dynamic task allocation system based on a principled publish/subscribe messaging model that requires all inter-robot communication to be resource-centric. This system was implemented and tested on physical robots in both a short-term tightly-coupled task domain and a long-term loosely-coupled task domain. We demonstrated how the system is extremely reactive to changes in the environment, including abrupt failures of robots and random introduction of new tasks. MURDOCH is completely distributed, with no single point of congestion or failure, with the tradeoff that its task allocation solutions are always greedy.

As part of continuing work in the box-pushing domain, we will increase the granularity of our control of the box by dividing the box's alignments into more categories and parameterizing each pushing task by both time and velocity. Our hope is to successfully transport the box in a tightly-constrained corridor environment. As for the loosely-coupled task domain, we are currently designing a more ambitious experiment in which we will run the robots for much longer, possibly days, and record relevant data for an objective performance analysis of the system.

8. Acknowledgments

The research reported here was conducted at the Interaction Lab, part of the Robotics Research Lab at the University of Southern California Computer Science Department. The work is supported by Office of Naval Research Grants N00014-00-1-0140 and N0014-99-1-0162, Jet Propulsion Laboratory Contract No. 1216961, and DARPA Grant DABT63-99-1-0015. We thank Richard Vaughan and Andrew Howard for important insights concerning this work.

References

[1] David L. Martin, Adam J. Cheyer, and Douglas B. Moran. The open agent architecture: A framework for building distributed software system. *Applied*

Artificial Intelligence, 13(1):91–128, Jan–Mar 1999.

[2] Katia Sycara, Keith Decker, Anandeep Pannu, Mike Williamson, and Dajun Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, December 1996.

[3] Milind Tambe. Agent architectures for flexible, practical teamwork. In *Proc. of the Natl. Conf. on Artificial Intelligence (AAAI)*, Providence, Rhode Island, July 1997.

[4] Lynne E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[5] Barry Brian Werger and Maja J Matarić. Broadcast of local eligibilty for multi-target observation. In *Proc. of the Intl. Symp. on Distributed Autonomous Robotic Systems (DARS)*, Knoxville, Tennessee, October 2000.

[6] Daniel D. Corkill. Blackboard systems. *AI Expert*, 6(9):40–47, September 1991.

[7] William E. Walsh and Michael P. Wellman. A market protocol for decentralized task allocation. In *Proc. of the Intl. Conf. on Multi Agent Systems (ICMAS)*, Paris, France, July 1998.

[8] Steven McCanne. Scalable multimedia communication with Internet multicast, light-weight sessions, and the MBone. Technical Report CSD 981002, UC Berkeley, March 1998.

[9] Arvola Chan. Transactional publish/subscribe: The proactive multicast of database changes. In *Proceedings of ACM SIGMOD Conf. on Management of Data*, page 521, Seattle, WA, June 1998.

[10] Guruduth Banavar et al. An efficient multicast protocol for content-based publish-subscribe systems. In *Proc. of the Intl. Conf. on Distributed Computing Systems*, Austin, Texas, June 1999.

[11] Marcos K. Aguilera et al. Matching events in a content-based subscription system. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 53–61, Atlanta, Georgia, May 1999.

[12] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proc. of the Natl. Conf. on Artificial Intelligence (AAAI)*, Seattle, WA, July 1994.

[13] Kutluhan Erol, James Hendler, and Dana S. Nau. UCMP: A sound and complete procedure for Hierarchical Task-Network planning. In *Proc. of the Intl. Conf. on Artificial Intelligence Planning Systems*, Chicago, IL, June 1994.

[14] Maja J Matarić. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artifical Intelligence*, 9(2–3):323–336, 1997.

[15] Ronald C. Arkin. *Behavior-Based Robotics.* MIT Press, Cambridge, MA, 1998.

[16] Brian P. Gerkey, Kasper Støy, and Richard T. Vaughan. Player robot server. Technical Report IRIS-00-392, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, November 2000.

[17] Bruce Donald, Jim Jennings, and Daniela Rus. Information invariants for distributed manipulation. *The Intl. Journal of Robotics Research*, 16(5):673–702, October 1997.

[18] Bruce Donald, Jim Jennings, and Daniela Rus. Minimalism + distribution = supermodularity. *Journal of Experimental and Theoretical Artifical Intelligence*, 9(2–3):293–321, 1997.