A Versatile Implementation of the *TraderBots* Approach for Multirobot Coordination

M. Bernardine DIAS, Robert ZLOT, Marc ZINCK, Juan P. GONZALEZ, and Anthony (Tony) STENTZ
The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA
{mbdias, robz, mbz, jgonzale, axs}@ri.cmu.edu

Abstract. This paper reports details of a versatile implementation of the *TraderBots* approach: A market-based approach to multirobot coordination. The architectural layout, implementation details, and variety of features are described. Experimental results are presented using a team of Pioneer II DX robots engaged in exploration and distributed sensing tasks. Different features and strengths of the approach and the implementation are highlighted in the experimental results.

1. Introduction

The past decade has witnessed a growing emphasis in research topics highlighting coordination of multirobot systems. This emphasis is generated by the increasing demand for automation in application domains where a single robot is no longer capable of performing the necessary tasks, and/or multiple robots can accomplish the same tasks more efficiently. Thus, coordinating multiple robots to cooperatively complete a task is a difficult problem that has attracted much attention from the robotics research community in recent years. This paper details a versatile implementation of the *TraderBots* coordination approach, specifically geared towards coordinating multiple robots for cooperative tasks in dynamic environments. The *TraderBots* approach capitalizes on the strengths of market economies that enable many agents to collectively execute complex tasks with access to incomplete information under dynamic conditions.

Recently, negotiation-based and economy/market-based multirobot coordination has gained popularity. This work in multirobot coordination draws from the software agents literature that began with Smith's Contract Net Protocol [15], its extension by Sandholm and Lesser [14], and the general concepts of market-aware agents developed by Wellman and Wurman [19]. These concepts have since been extended to control a variety of multiagent (and more recently multirobot) systems. Golfarelli, Maio, and Rizzi [10] designed a swap-based negotiation protocol for multirobot coordination that restricted negotiations to task-swaps, and Botelho and Alami [2] produced an auction-based mechanism for task allocation in multirobot coordination applications. Stentz and Dias [16] proposed a more generally capable market-based approach for multirobot coordination which aims to opportunistically introduce pockets of centralized optimal planning into a distributed system, thereby exploiting the desirable properties of both distributed and centralized approaches. Thayer et al. [18], Gerkey and Mataric [9], and Zlot et al. [21] have since produced market-based multirobot coordination results. Zlot and Stentz [20], and Dias and Stentz [6] report a more complete review of multirobot coordination approaches. Dias and Stentz [6] provide a detailed description of the *TraderBots* approach; a brief overview of this approach follows:

Consider a team of robots assembled to perform a particular set of tasks, or a complex mission. Tasks such as exploration, distributed sensing, mapping, and

reconnaissance, where subtasks can be carried out in parallel, in dynamic environments, where prior information about the environment is imperfect, benefit most from the application of the *TraderBots* approach. Consider further, that each robot in the team is modeled as a self-interested trader, and the team of robots as an economy. The goal of the team is to complete the tasks successfully while minimizing overall costs and maximizing overall revenue. Costs can be estimated in terms of metrics such as distance traveled, time elapsed, energy expended, resources depreciated, or robot-hours consumed, and revenue can be awarded in accordance with metrics such as number of tasks completed, information gained, or number of samples retrieved, depending on the requirements of the overall mission and application domain. Each robot aims to maximize its individual profit. However, since all revenue is derived from satisfying team objectives, the robots' self-interest correlates to doing global good. Moreover, the global cost is determined by the summation of individual robot costs, and hence each deal made results in global cost reduction. The competitive element of the robots bidding for different tasks enables the systems to decipher the competing local information of each robot, while the currency exchange provides grounding for the competing local costs in terms of the global value of the tasks being performed.

The *TraderBots* approach has proven to be successful in efficient and robust multirobot coordination in previous implementations in simulation ([4], [7], [5]) and on physical robots ([21]). However previous publications have not disseminated implementation details sufficient for reproducing an implementation of the *TraderBots* approach on a robotic platform. The contributions of this paper are first, a detailed description of the most current and most versatile implementation to date of the *TraderBots* approach for multirobot coordination, and second, a demonstration of many previously un-implemented features of the *TraderBots* approach, including graceful handling of partial robot malfunctions and communication failures, accommodating new input from an operator during task execution, efficiently allocating tasks under dynamic conditions, dynamically generating tasks, incorporating new robots during execution, handling tasks that cannot be accomplished, and executing in unknown environments.

2. Implementation Details

An implementation of the *TraderBots* approach on a team of Pioneer robots enables the reported results. The details of the robotic system used in this implementation are presented next.

2.1 Robotic Platform

The robot team (shown in Figure 1) consists of a homogenous set of off-the-shelf mobile robot platforms outfitted with additional sensing and computing. Serving as the mobility platform is an ActivMedia Pioneer II DX indoor robot. A Mobile Pentium 266 with MMX is the main processor. Attached is a 1-gigabyte hard drive for program and data storage and 802.11b wireless card for ad-hoc communication between robots. Encoder data from the drive wheels is collected onboard from which dead reckoning position (x, y, θ) is calculated. Encoders provide a relatively accurate measure of linear travel, but relatively inaccurate angle measurement, such that small errors in angle compound over time resulting in large displacement errors. A solution to these pose errors is the addition of alternate angle measurements using a fiber optic rate gyroscope (KVH E-Core 1000). The gyroscope provides highly stable and accurate angle measurement (four degrees drift per hour). Robots sense their environment using an 180˚ scanning laser range finder (SICK LMS 200). Horizontal scan-range-data is incorporated with position data to create a 2D map. In addition to providing information to the operator, the map is used for local navigation and cost estimation during trading.

Figure 1: Robot Team

2.2 Architecture

Design and implementation of the system supporting the *TraderBots* architecture was focused on extensibility and scalability. The system can be conceptualized as a 4-tier structure (as illustrated in Figure 2): hardware, hardware abstraction, autonomous navigation, and multi-robot (inter-robot) communication. The hardware layer consists of the motors, encoders, laser sensor and gyroscope. A process called *RobotCntl*, which serves as a hardware abstraction to higher-level processes, controls all components of the hardware layer. *RobotCntl* manages the state of the hardware, collects, timestamps, and provides access to data, and interprets and executes hardware control commands from higher-level processes.

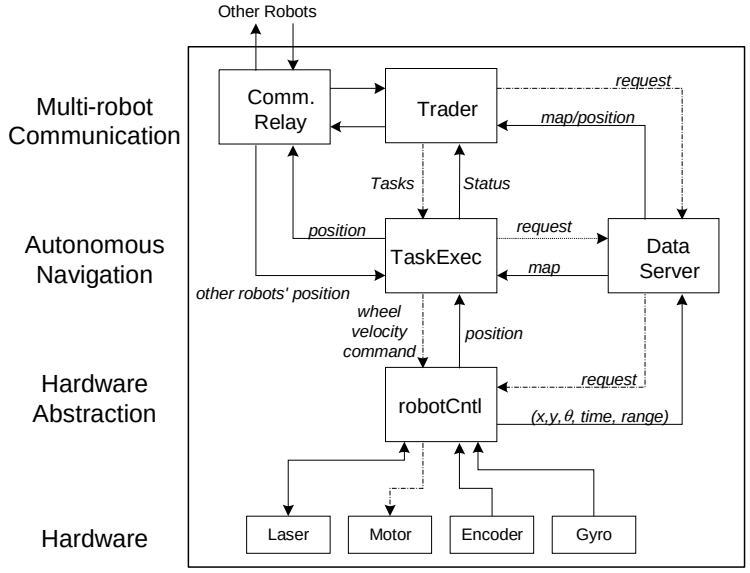

Figure 2: Architectural Layout

Two separate processes, *TaskExec* and *DataServer*, in conjunction with the hardware abstraction layer accomplish autonomous navigation. *TaskExec* executes local navigation with a map provided from *DataServer*. *DataServer* aggregates position and laser range data from the hardware abstraction level and provides maps to other processes that require map information. In addition to receiving map data from *DataServer*, *TaskExec* broadcasts its position to other robots and receives the position of other robots through the *CommRelay*. These positions are placed in *TaskExec*'s navigation map as obstacles to implement a collision avoidance mechanism between robots. At the highest level of control is the *Trader* process. The *Trader* is responsible for coordinating with other robots through *CommRelay* and determining task allocations. Once tasks are allocated, the *Trader* maintains a schedule for its commitments and periodically sends tasks to be executed to the *TaskExec*. The *Trader* also keeps a local map for cost estimation during trading.

2.3 Communication

Communication between modules occurs in two ways: intra-robot (between modules on a single robot) and inter-robot (between modules on different robots). These two instances use different techniques reflecting their unique situations. Intra- robot communication happens between processes on one robot such as *TaskExec* and *RobotCntl* or *Trader* and *DataServer*. These links are assumed to be high-speed and reliable since the processes run on the same robot. The basic assumption is that this channel is high bandwidth, low latency and reliable. In this implementation we use a communication package called RTC (Real Time Communication) [12], which provides inter-process-communication between processes on the same machine or machines with reliable links. Inter-robot communication differs from intra-robot communication with respect to bandwidth and reliability. Inter-robot communications use wireless Ethernet that is orders of magnitude less capable in terms of bandwidth in comparison to intra-robot communication, and suffers from reliability problems due to radio interference. In order to avoid re-transmission problems in an unreliable wireless environment, we use UDP (User Datagram Protocol), a connectionless datagram protocol built on IP (Internet Protocol), for transmitting data between robots. All RTC messages destined for another robot are sent to the *CommRelay* and packaged as UDP messages. The UDP messages are then sent via UDP to the

destination robot and received by that robot's *CommRelay*. They are then converted back into RTC messages and sent to the appropriate modules using the intra-robot communication protocol.

As described above, communication between processes on different robots is realized through a point-to-point UDP-based message-passing scheme. Thus, each *RoboTrader* is not instantly able to determine which other *RoboTrader*s it is connected to at any given time. In order to keep track of which other traders are reachable, each *RoboTrader* sends out a periodic hunt signal to all existing robots whether they are known to be alive or not. All traders that receive the hunt signal record the sender as connected, and send an acknowledgement (ACK). The original sender waits a predetermined amount of time (10 seconds) for ACKs. The senders of any ACKs that arrive within the time interval are recorded as being connected, and at the end of the time interval all other traders are marked as disconnected. Additionally, the senders of any other signals (e.g. auction calls, bids) can opportunistically be marked as connected by the recipients of these messages.

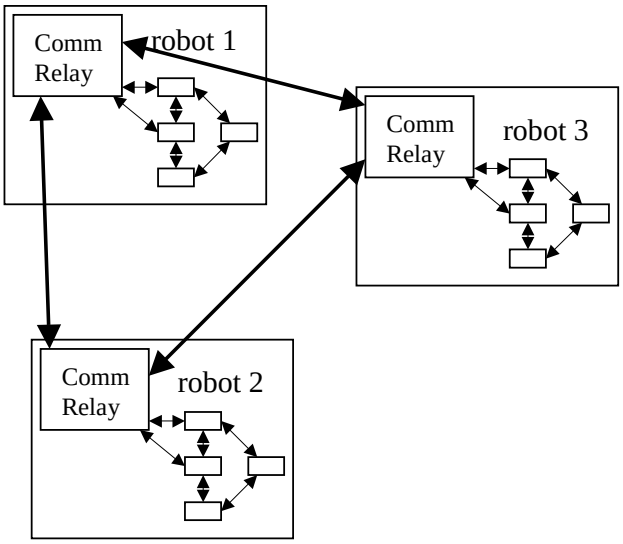

Figure 3: Inter-robot and Intra-robot Communication

It is possible for a connected *RoboTrader F* to become perceived as disconnected at a later time if a trader *T* who had detected that robot previously ceases to communicate with it. This can happen both in the case of a communication problem (out of range or a malfunction) or a robot death. When the disconnection occurs, *T* waits for a specified interval (1 minute) to attempt to reconnect to *F* either through the hunt-acknowledge protocol, or by receiving any other message from *F*. If no such message arrives, then T assumes that there is a problem with *F*. To handle the possible fault, *T* first asks the other connected traders if they can connect to *F*. This may be possible if *F* is out of communications range of *T*, but is within range of some other robot *R* that is also reachable by *T*. If any other traders are connected to *F*, then *T* reverts to believing that *F* is alive and begins the 1-minute disconnection timer once again (in case *F* suffers a fault before reconnecting to *T*). Otherwise, *T* assumes *F* has suffered a robot death and thus is out of commission until it receives a message from *F* again. The handling of robot death and other robustness issues in the *TraderBots* approach is reported in detail in a recent publication submission to the International Conference on Robotics and Automation [8].

2.4 Execution

The *TaskExec* module performs the execution level of the architecture. This module is in charge of monitoring and arbitration of tasks, allowing for sequential and/or parallel execution. The *TaskExec* module combines the virtues of SAUSAGES [11] and DAMN [13] to create a task network in which simultaneous tasks can have their outputs combined through an arbiter. The basic building blocks for the task network are tasks. Tasks share a common structure that allows them to be transparently called by the *TaskExec* independent of the specific function that the task performs. Thanks to this common structure, tasks can be dynamically added and removed from the task network.

The most important member functions of a task are:

- *startTask()*: this function is called once by the *TaskExec* before the task is executed for the first time.

- *runOnce()*: this function is called once each execution cycle, for as long as the task is active
- *endTask()*: this function should be called by the task itself, when its termination criterion has been met. The TaskExec will also call it if the task executes beyond its assigned termination time.

The *TaskExec* is the executor of the task network. It starts, executes, monitors and terminates tasks as required. It also allows for dynamic changes in the task network and operates as follows:

- Process inputs from sensors, and put them in maps and data structures that are accessible to all tasks
- Check start-conditions of all tasks. If the start conditions of one or more tasks are satisfied, start the tasks by calling the *startTask()* member of the tasks. Tasks will be considered active from this moment until their termination. There are two kinds of start conditions for a task: (1) at a specified time, and (2) after completion of its predecessor: It is also possible to condition the start on the successful termination of the task, and specify a different task to be executed if the predecessor fails.
- Call *runOnce()* for all the tasks that are active. Each task processes the changes in the world and generates an output, or a vote. If a task has complete control of a resource, the output can be a direct command to the resource. If the task has shared control of a resource, the output of the task will be a vote on the desired behavior of the controlled resource. If a task has finished, it calls *endTask()*, to indicate its termination.
- Check termination conditions for all active tasks. If a task remains active beyond its scheduled execution time, the *TaskExec* will terminate the task.

Tasks can have control of two types of resources: exclusive and shared. Exclusive resources (for example science instruments on a space exploration robot) are unique to a task, and can be controlled directly from the task. Shared resources (for example motors and multi-purpose sensors) are common to several tasks, and need to be arbitrated to perform an action. The most common kind of arbitrated resources are steering angle and speed. In the current implementation, all the tasks that participate in the selection of a steering angle and speed share a set of arcs with a different curvature and speed associated to each one of them. When the *runOnce()* function is called, the tasks issue votes on each one of the arcs. After all the tasks have been called for the current execution cycle, the *TaskExec* combines the votes from all the active tasks and executes the arc corresponding to the winning vote. The *RoboTrader* sends sequences of tasks to the *TaskExec* to be executed. In the current implementation the *TaskExec* maintains a single execution queue, and new tasks are added to the end to the current execution queue. If the queue was empty before the arrival of new tasks, the new tasks are executed immediately. The *TaskExec* reports success or failure of an executed task to the *RoboTrader* when a task terminates.

2.5 Trading

Trading is a key component of the *TraderBots* approach. A *RoboTrader* assigned to each robot is responsible for opportunistically optimizing the tasks the robot commits to executing. An *OpTrader* serves as an interface agent between the operator and the robot team. Each trader maintains a portfolio in which it keeps track of its commitments, schedule, currently executing tasks, and tasks it trades to others. Two forms of contract types are allowed during trading: subcontracts and transfers. If the contract type is a subcontract, it implies the auctioneer is interested in monitoring the progress of the task and will hence expect a report when the task is completed; payment is made only after the subcontracted task is completed. Note that a subcontracted task can be traded in turn to another robot, but only as another subcontract. Each robot only needs to keep track of the robot it won the subcontract from and the robot it subcontracted the task to. Once the task is executed, the completion of the task is reported along the chain of robots linked by the subcontracts until the initial auctioneer is notified. If on the other hand, the contract type is a transfer, payment is made as soon as the task is traded, and no further communication

concerning that task is necessary between the auctioneer and bidder. Each trader has an internal alarm that prompts it to auction all tasks in its schedule periodically. Note that tasks being executed are removed from the schedule and hence cannot be traded. This implementation decision was based on the assumption that a task cannot be transferred once it is started. For application domains where this assumption is not true, this restriction can be removed. In contrast, in application domains where idle time for robots is highly costly, introducing a larger execution window by sending a higher number of tasks to the TaskExec and removing them from future auctions will be more suitable. A trader initiates an auction by sending out a call for bids. Traders within communication range compute and submit bids to this auction. Once the specified deadline expires, the auctioneer resolves the call by making a profit-maximizing allocation based on the bids it received. If a trader receives an award for a bid it submitted, it accepts or rejects that award based on its current state. Note that an award is binding after it has been accepted. Two methods of call resolution are used in the current implementation of *TraderBots*. The *RoboTrader*s assign at most the single most profitable bid submitted to the auction. The *OpTrader*, and *RoboTrader*s who discover they are in a fault state due to a malfunction, use a greedy algorithm for resolving calls so that tasks are allocated more rapidly; this greedy allocation is done because they cannot execute the tasks themselves and in the case of a malfunction, because the robot can expect a robot death with higher probability and hence aims to reassign tasks quickly. The greedy algorithm assigns the most profitable bid submitted by each trader that participates in the auction while assuring that no task gets assigned more than once and no bidder gets assigned more than one task during each auction. All bids are limited to single-task bids in this implementation. Dias and Stentz explore the comparative advantages of different negotiation strategies in a previous publication [7].

In order to participate in an auction, robots need to calculate the costs of tasks. A robot announcing an auction must determine its reservation price, i.e. the highest price it is willing to pay to subcontract or purchase a task. A robot bidding in an auction must calculate the expected cost of the tasks being offered. These valuations are based on marginal costs – the difference in between the cost of the current schedule with those tasks and the cost of the schedule without those tasks. For a single task, an auctioneer's valuation is the savings resulting from removing that task from its schedule and reordering the remaining tasks in the schedule in an efficient manner. A bidder's marginal cost for a single task is the estimated cost of efficiently inserting the task into its schedule. When a trader initiates an auction, the call is sent to all robots marked as connected by that trader. This allows the trader to clear the auction as soon as it receives bids from all connected robots, rather than waiting for the auction deadline.

3. Experiments, Results, and Discussion

Many application domains demand high quality performance from multirobot systems. Dias and Stentz [6] identify several requirements for a successful multirobot coordination approach. Some of these requirements are used to evaluate the versatility of the current implementation of the *TraderBots* approach. The following characteristics are examined: the robustness to malfunctions, the ability to execute a task with incomplete information about the environment, the ability to deal with imperfect communication, the ability to dynamically handle new instructions from the operator during execution, the flexibility to execute different types of tasks, and the ability to accommodate the addition of a robot to the team during operation. The chosen application is a distributed sensing problem where robots are tasked with gathering sensory information from various designated locations of interest. Figure 4a shows a graph of the 25mx45m area in which 30 cities (tasks), illustrated as triangles, are assigned to a team of robots to visit. Figure 4b shows a photograph of the cluttered dynamic environment, graphed in Figure 4a, where the reported experiments were carried out. This translates into a version of the traveling salesman problem (TSP) with the robots being represented by multiple salesmen following paths instead of tours (i.e. without the requirement that robots need to return to their starting locations) and where all the robots can start from different base locations – this is known as the multi-depot traveling salesman path problem (MD-TSPP). The tasks can be considered as cities to be visited where the costs are computed as the time taken to traverse between

cities. A task is completed when a robot arrives at a city. The global task is complete when all cities are visited by at least one robot. The global cost is computed as the summation of the individual robot cost, and the goal is to complete the global task while minimizing the number of robot-hours consumed. When the robots are not executing tasks, they remain stationary at their current locations.

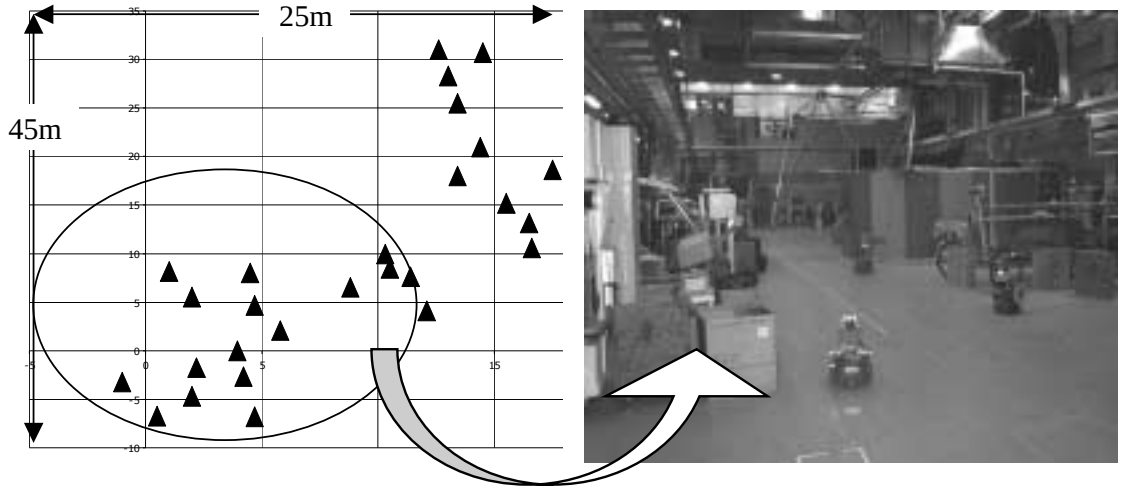

Figure 4: Distributed Sensing Tasks and Experimental Environment

Each robot is responsible for optimizing its own local schedule (i.e. given a set of tasks, the robots attempt to find the optimal TSPP solution to their local problem instance). In general, the TSPP is NP-hard, so approximation algorithms are often used when large problem instances are encountered. Additionally, the problem encountered is an online variant of the TSPP – cities are arriving whenever a robot is awarded a task in an auction and are being removed whenever a task is traded to another robot. When adding a task to the tour, it is inserted into the tour at the location that results in the smallest increase in marginal cost. Insertion heuristics have been shown to have constant factor approximation guarantees for some point orderings, but in general they have a performance guarantee of ($\lceil \log n \rceil + 1$) for n-city tours [1]. Tours are also optimized as a whole whenever tasks are added or removed. If the number of tasks is at most 12, the optimal solution is computed using a depth first search-based algorithm. If the number of tasks exceeds 12, computing the optimal solution is too time-intensive, and hence a minimum spanning tree-based 2-approximation algorithm is used [3] if the resulting tour has a lower cost than the current tour. Tour optimization is also performed whenever a task is completed or failed as costs between cities may have changed due to new map information from recent sensor readings. In the implemented TSPP scenario, all valuations are derived from inter-point distance costs. These costs are estimated using a D* path planner [17] with the robot's local map as input.

Experiment 1: This experiment measures the performance, averaged over a set of 3 runs, of the nominal case for 4 robots engaged in a distributed sensing task.

Experiment 2: This experiment investigates how a partial robot malfunction (simulated by killing the *TaskExec* process) at a random time during a run affects the nominal performance. Reported results are averaged over a set of 3 runs for 4 robots engaged in a distributed sensing task.

Experiment 3: This experiment investigates how communication failures (simulated by deleting 10% of messages passed between robots) affect the nominal performance. Reported results are averaged over a set of 3 runs for 4 robots engaged in a distributed sensing task.

Experiment 4: This experiment investigates the effect of new operator input during execution (where 4 random tasks are cancelled at random times during execution). Reported results are averaged over a set of 3 runs for 4 robots engaged in a distributed sensing task.

Experiment 5: This experiment investigates the performance for the 3-robot case using a random allocation for comparison with the *TraderBots* allocation. Reported results are averaged over a set of 3 runs for 3 robots engaged in a distributed sensing task.

Experiment 6: This experiment investigates the performance for the 3-robot case using a greedy allocation for comparison with the *TraderBots* allocation. Reported results are averaged over a set of 3 runs for 3 robots engaged in a distributed sensing task.

Experiment 7: This experiment investigates the nominal performance for the 3-robot case, for comparisons in efficiency with random and greedy allocations reported in experiments 5 and 6. Reported results are averaged over a set of 3 runs for 3 robots engaged in a distributed sensing task.

Experiment 8: This experiment investigates the effect of adding a new robot to the team at a random time during execution. Reported results are averaged over a set of 3 runs for a distributed sensing task.

Experiment 9: This experiment investigates the flexibility of the implemented *TraderBots* approach by applying it to an exploration task where a set of 4 robots dynamically generate locations to be visited in order to cooperatively build a map of a previously unknown world. Reported results are averaged over a set of 3 runs, each of 5-minute duration.

	# Robots	**Tasks Assigned**	**Tasks Handled**	**Team Cost**
Experiment 1	4	30	30	154.4
Experiment 2	4	30	30	150.3
Experiment 3	4	30	30	190.0
Experiment 4	4	30-4	27	140.9
Experiment 5	3	30	30	232.1
Experiment 6	3	30	30	162.0
Experiment 7	3	30	30	139.0
Experiment 8	3+1	30	30	139.1
Experiment 9	4	22	22	154.8

Table 1: Experimental Results

Table 1 reports the results of experiments 1-9. Experiment 1 shows a team of robots able to accomplish the 30 assigned tasks with a cumulative cost of ~150 robot-seconds nominally. Experiment 2 shows that an induced malfunction in one of the robots is handled gracefully without loss to solution efficiency. Experiment 3 shows that a 10% loss in communication raises the solution cost to ~190 robot-seconds, mainly due to repeated tasks because of lost acknowledgements, but does not prevent the handling of any of the tasks. Experiment 4 results in a drop in the solution cost due to the cancellation of 4 tasks during the experiments – note that on average only 3 of the tasks were cancelled before execution since tasks were chosen at random to be cancelled and a task chosen for cancellation was on average completed prior to the time of cancellation. Experiment 8 shows that the *TraderBots* approach can accommodate the addition of a robot during operations. However, the new robot was started in a different start position from the previous experiments and hence the resulting costs cannot be compared. Experiment 9 shows the results when the robots are tasked with generating suitable observation points in order to

collectively build a map of the environment. Figure 5a shows an example of a map collaboratively generated by the robots. Figures 5b, 5c, and 5d show a graphical comparison of allocations made using random, greedy, and *TraderBots* approaches in experiments 5-7. Note that the robot paths overlap the least in the *TraderBots* allocation and they overlap the most in the random allocation as expected. This result is reflected in the corresponding costs shown in Table 1. In previous publications Dias and Stentz report comparisons, in simulation, of the efficiency of the *TraderBots* allocation to the optimal allocation [7], to a centralized allocation [5], and to a fully distributed allocation [5].

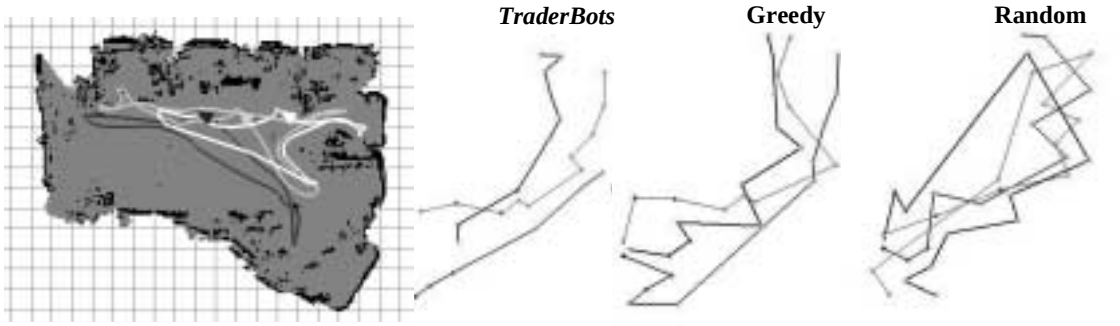

Figure 5: Generated Map From Exploration Task and Comparison of Allocations

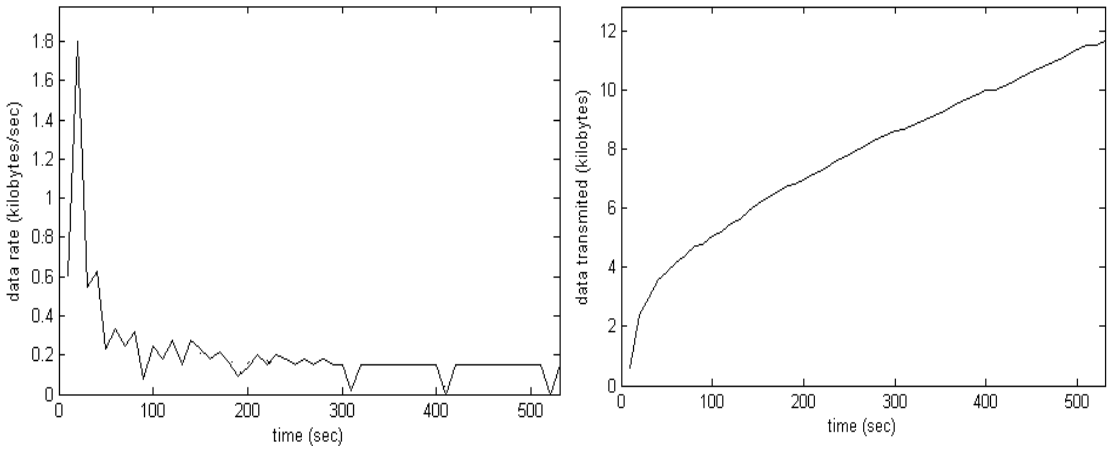

Figure 6: Trading-Related Communication Traffic

Figure 6 shows an analysis of communication traffic sent by one trader using the *TraderBots* approach. Figure 6a illustrates the evolution of the data rate, in kilobytes/second, over time, in seconds, and Figure 6b shows the cumulative data transmitted, in kilobytes, over the same time period. Message types are auctions, bids, hunts, awards, and acknowledgments. When the robot is first deployed, communication peaks as a result of the initial auction. As tasks are executed and knowledge is gathered, tasks continue being traded at a lower rate. The steady state communication rate is due to the continuing trading mechanism (auctions, bids, awards) and messages to maintain knowledge of trader state (hunts, acknowledgments).

4. Future Work

Work in progress and future extensions to this implementation of the TraderBots approach include task abstraction using trees, escaping local minima using clustered bids, graceful handling of robot death, efficient handling of heterogeneity, enhancing scalability, and developing a more systematic comparative analysis of different multirobot coordination approaches.

5. Acknowledgments

This document describes a multirobot architecture, implementation, and test bed used for conducting research on two active projects. This work was sponsored in part by the U.S. Army Research Laboratory, under contract "Robotics Collaborative Technology Alliance" (contract number DAAD19-01-2-0012) and in part by NASA, under contract "Heterogeneous Multi-Rover Coordination for Planetary Exploration" (contract number NCC2-1243). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, NASA, or the U.S. Government.

The authors wish to acknowledge the contributions of the other members of the CTA Multirobot research group, Bart Nabbe, Nidhi Kalra, Dave Ferguson, and Andres S. Perez-Bergquist, which enabled this implementation.

6. References

[1] Bafna, V., Kulyanasundaram, B., and Pruhs, K., "Not All Insertion Methods Yield Constant Approximate Tours in the Euclidean Plane", *Theoretical Computer Science*, 125(2), pp.345-353, 1994.

[2] Botelho, S. S. C., and Alami, R., "M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement", *Proceedings of the International Conference on Robotics and Automation*, 1999.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., "Introduction to Algorithms (Second Edition)", *MIT-Press*, 2001.

[4] Dias, M. B. and Stentz, A., "A Free Market Architecture for Distributed Control of a Multirobot System", proceedings of the 6th International Conference on Intelligent Autonomous Systems (IAS-6), 2000.

[5] Dias, M. B., and Stentz, A., "A Comparative Study between Centralized, Market-Based, and Behavioral Multirobot Coordination Approaches", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[6] Dias, M. B., and Stentz, A., "A Market Approach to Multirobot Coordination", *Technical Report, CMU-RI -TR-01-26, Robotics Institute, Carnegie Mellon University*, 2001.

[7] Dias, M. B., and Stentz, A., "Opportunistic Optimization for Market-Based Multirobot Control", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[8] Dias, M. B., Zlot, R., M., Zinck, M., and Stentz, A., "Robust Multirobot Coordination in Dynamic Environments", *IEEE International Conference on Robotics and Automation (ICRA) 2004*, Submitted 2003.

[9] Gerkey, B. P. and Matarić, M. J., "Sold! Market methods for multi-robot control", *IEEE Transactions on Robotics and Automation Special Issue on Multi-Robot Systems*, Submitted 2001.

[10] Golfarelli, M., Maio, and D., Rizzi, S., "A Task-Swap Negotiation Protocol Based on the Contract Net Paradigm", *Technical Report, 005-97, CSITE (Research Centre For Informatics And Telecommunication Systems), University of Bologna*, 1997.

[11] Gowdy, J., "SAUSAGES: Between Planning and Action", *Technical Report CMU-RI-TR-94-32, Robotics Institute, Carnegie Mellon University*, 1994.

[12] Real-Time Communications (RTC), http://www.resquared.com/RTC.html.

[13] Rosenblatt, J., "DAMN: A Distributed Architecture for Mobile Navigation", *Doctoral Dissertation, Technical Report CMU-RI-TR-97-01, Robotics Institute, Carnegie Mellon University*, January, 1997.

[14] Sandholm, T., and Lesser, V., "Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework", *Proceedings of the first International Conference on Multiagent Systems (ICMAS-95)*, 1995.

[15] 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, 1980.

[16] Stentz, A. and Dias, M. B., "A Free Market Architecture for Coordinating Multiple Robots", *Technical Report, CMU-RI-TR-99-42, Robotics Institute, Carnegie Mellon University*, 1999.

[17] Stentz, A., "Optimal and Efficient Path Planning for Partially-Known Environments", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1994.

[18] Thayer, S. M., Dias, M. B., Digney, B. L., Stentz, A., Nabbe, B., and Hebert, M., "Distributed robotic mapping of extreme environments", *Proceedings of SPIE: Vol. 4195: Mobile Robots XV and Telemanipulator and Telepresence Technologies VII*, 2000.

[19] Wellman, M. P., and Wurman, P. R., "Market-Aware Agents for a Multiagent World", *Robotics and Autonomous Systems*, Volume 24, 1998.

[20] Zlot, R., and Stentz, A., "Efficient Market-based Multirobot Coordination for Complex Tasks", *The International Journal or Robotics Research (IJRR)*, Submitted 2003.

[21] Zlot, R., Stentz, A., Dias, M. B., and Thayer, S., "Multi-Robot Exploration Controlled By A Market Economy", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.