SUPERVISORY CONTROL OF MULTIPLE ROBOTS BASED ON A REAL-TIME STRATEGY GAME INTERACTION PARADIGM

HANK JONES and MARTIN SNYDER

Aerospace Robotics Laboratory, Stanford University, Stanford, CA, USA 94305, hlj@arl.stanford.edu
Ethermoon Entertainment, Inc., Edgemont, PA, USA 19028, msnyder@ethermoon.com

Abstract

As robots are deployed beyond the laboratory and into the field, the method with which they interact with their operators is vital to efficient and optimal use. This paper describes the design and implementation of a supervisory control architecture that enables the straightforward operation of multiple complex robots.

The design was inspired by the flexibility and scalability of the real-time strategy (RTS) game interface paradigm. However, the basic RTS architecture had to be substantially adapted to accommodate the constraints of field robot operation. Initial experimental results show that this familiar and tested interaction paradigm is applicable for complex robotics systems.

Keywords

Supervisory control, multiple robots, human-system interaction, real-time strategy, RTS, agents

1 Introduction

Current methods of controlling robots fall into one of two categories: those designed for a single complex robot and those designed for distributed systems of simple robots. This research addresses the need for an interface and system architecture in a third category that provides supervisory control for a single operator to manage the conduct of multiple complex robots. This paper distinguishes between simple and complex robots by the latter's ability to perform extended missions that call for more than one kind of task.

There have been a wide variety of human-system interfaces for single complex robots. Autonomous helicopters have been controlled using point-and-click [12] and virtual dashboard [1] techniques; autonomous underwater vehicles and space vehicles have been directed using virtual environments [8] and high-level tasking [25, 27]; intelligent arms have been instructed using gestures [6] and graphical icons [13]; and many complex robots have simply been fully teleoperated [24].

However, if one operator were expected to command multiple italicized complex robots, none of these methods would easily scale to accommodate the additional complexity. Direct teleoperation would either overstress the operator or underutilize the robots [10]. Extensions of the robot interfaces of more automated systems, such as control panel, dashboard, or master-slave mechanisms, for single robots do not appear to naturally accommodate additional robots.

Research regarding the control of multiple robot systems that exhibit emergent or reactive behavior has only recently moved beyond simulation work to robotic implementations. Most architectures, such as AuRA [2] and ALLIANCE [20], focus on strengthening the autonomous capabilities of the robot teams rather than their operation by humans. The research that has incorporated the human operator, such as ROBODIS [26], RAVE [7], MokSAF [21], and MissionLab [3], have largely concentrated on methods of cooperative motion and task planning for surveillance and exploration, with the user available either for assistance or initial planning.

There have been a small number of other research programs that have focused on the human-system interaction for multiple complex robots. Purely virtual but complex robots were operated in DARPA's SIMNET [5], and a high-level tasking "playbook" interface has been developed and tested for future operation of uninhabited combat air vehicles [16]. The MAGIC2 system, developed for operational control of unmanned air vehicles [14], and the MACTA hybrid agent/reactive architecture [4] are the only two robotic systems that have demonstrated operation of multiple complex robots experimentally. MAGIC2 combines control panels for the control of unmanned aerial vehicles but appears to be limited to a maximum of four vehicles per operator. MACTA focuses on behavior scripts and their ability to satisfy human-designated goals, rather than human-system efficacy. Ultimately, no standard method of interaction has emerged.

The Aerospace Robotics Laboratory at Stanford University has developed complex robots capable of carrying out many different tasks given at a high level [25, 27]. The focus has been on developing human-robot teams, although the research to date has been limited to single robot-operator-task combinations. The purpose of our research was to design and implement a human-robot interaction that extends this past work to enable intuitive and scalable control of multiple complex robots by one user.

2 Objectives and Constraints

2.1 User-centered design

As a rule, systems built around autonomous robots often provide mechanisms for human involvement only as an afterthought. This tendency to push user interaction to the background is not only a problem in robotics; many other technological fields have similar histories during their early development [11]. In addition, the human-system interactions for robots are almost always designed by the engineers responsible for the robots' development. The resulting interface is thus often based on the engineer's model of the underlying system rather than the task that the robot was built to accomplish. [9] However, robotics could benefit from the lessons learned in other fields [23] that have embraced the concepts that constitute user-centered design.

User-centered design, a concept originated by Don Norman and Stephen Draper, is a philosophical approach to interface design that seeks to put the user first and the technology second. Typical user-centered design efforts incorporate end user input from the beginning to encourage the maximum possible ease of use and utility of the final product.

To develop a user-centered interaction for multiple robots, we observed the command and control of police Special Weapons and Tactics (SWAT) teams during field exercises. SWAT teams are a valuable environment for studying hierarchical human-human interaction for the command of teams, and are also a potential end user of autonomous robots. Some important lessons learned include:

- Commanders and subordinates naturally interact through dialogues about current positions, surroundings, and capabilities
- Comprehensive world knowledge for the purpose of conceiving commands was extremely difficult to construct and convey
- Interfaces to the system require a high level of flexibility

Given these lessons, we sought to develop a supervisory control architecture that replicated the dialogue interaction to some extent, avoided the assumption that complete world knowledge could be obtained, and flexibly adapted to changing robot capabilities and missions.

2.2 RTS user interaction

As discussed previously, there is no standard interaction for the operation of multiple complex robots. However, we did identify a suitable interaction paradigm for the command and control of multiple complex robots that has undergone extensive testing and refinement -- although only for simulated ideal robots. This interface genre, used most often in Real-Time Strategy (RTS) computer games, has the following pertinent characteristics:

- Implements an interface between a superior controller and multiple autonomous entities
- Third-person perspective of the environment is the dominant viewpoint
- Robots and objects are represented as icons that may be selected singly or in groups for subsequent action
- Operator uses a combination of mouse gestures and keystrokes for input

The concept behind Real-Time Strategy games is that one person is required to command many autonomous entities (Figure 1) to achieve strategic goals. Popular examples of the genre include Strifeshadow (http://www.ethermoon.com) and Starcraft (http://www.blizzard.com/starcraft/). Future robot applications should benefit from the existence of millions of operators with experience in this interaction paradigm. A Boeing-led study funded by the Department of Defense identified such an approach as the best method to overcome current human-system interface challenges [17].

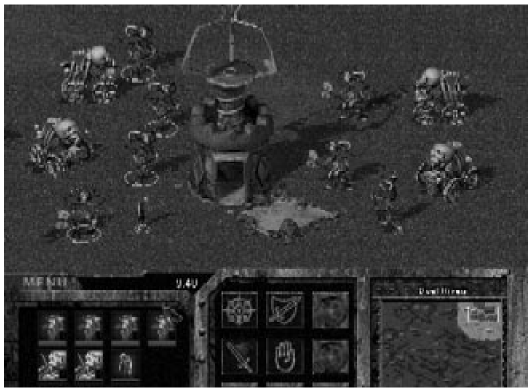

Figure 1. Example RTS interface (from Strifeshadow)

2.3 RTS control architecture

RTS control protocols allow for shared commands and multi-entity interaction so that game objects can easily be instructed to collaborate on certain tasks. While the implementations vary, most RTS games share the following characteristics:

- User interface and data representation are separate from the control architecture
- Entire game state is managed as a single simulation; all entities operate within the context of this shared state and contribute no information to the simulation state
- Command architecture supports immediate as well as high-latency and non-guaranteed connections

The control architecture is typically constructed in three parts: the data objects, the interface, and the command and data transfer objects necessary for communication. (Figure 2)

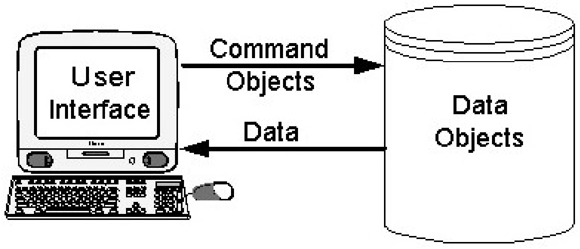

Figure 2. Generic RTS control architecture

2.4 Difficulties presented by field robots

Field robots, sometimes called telerobots, are robots that extend a person's sensing and/or manipulating capability to a location remote from that person. They usually have enough autonomous capability to control their own movement and act on high-level commands from their users.

The relationship between a field robot and its user must be emphasized – they do not operate 'within sight' of one another. More generally, no direct sensors provide a third-person perspective of the robot activity to the user. Thus, the computer link is the only conduit between human and robot and is of vital importance to the success of a mission. This characteristic is the key difference between field robots and the better-known industrial or personal robots.

A review of the typical field robot applications might compare favorably to the list of scenarios currently employed in RTS games. One might even reason at first glance that a practical human-system interface might be accomplished by simply "plugging in" a robot system to an RTS game architecture. However, further consideration reveals the following significant differences between the typical RTS and robot environments:

- The distributed sensors of the robots will often disagree and send conflicting inputs that may not be readily integrated into a comprehensive world model
- The real-world nature of robot systems does not allow the shared-state consistency found in the standard RTS environment
- Disturbances to the robots and objects in the world may be completely unobservable to the interface
- Commands must be conceived and given in the robot frame of reference
- Relationships between robots, their capabilities, and objects in their world change dynamically

An RTS system assumes knowledge of the complete state of all robots and objects in the environment. There is no confusion about an object's identity or location. However, in actual robot operational systems, the assumption that robots will sense infallibly and agree on everything is unrealistic. Systems used to command robots must recognize that robots will disagree about fundamental aspects of their environment yet be able to handle these discrepancies adroitly. The practical outcome is that there is no conclusive global model of the world that the robot or operator can use for command purposes.

Finally, a valuable trait of a well-designed interface is that it only affords tasks and objects to the user that the robot is capable of satisfying. However, robots may change as its engineers provide upgrades, or change dramatically as the realities of a deployment effect its second-to-second capabilities. This can be problematic, since robot capabilities not available on the interface are effectively nonexistent, and those capabilities on the interface that the robot cannot accomplish will be a great source of frustration to the user. [18]

3 Design and implementation

We performed our initial implementation on the free-flying robots of the Stanford University Aerospace Robotics Laboratory (http://arl.stanford.edu) shown in Figure 3. To implement the control paradigm required by the RTS architecture, we needed to change the robots' command and control structure, which had been designed for the operation of a single robot performing a single task. The commands to the robots now have to distinguish between many robot, task and object combinations. Much of the necessary new functionality was realized by creating a community of agents to handle communication between the robots and the user interface. The basic system structure is shown in Figure 4.

3.1 Object-Based Task-Level Control

Many of the fundamental issues of field robot operation were addressed through the development of Object-Based Task-Level Control (OBTLC) at the

Stanford's Aerospace Robotics Laboratory. Originally created for controlling field robot systems in space applications, OBTLC technologies have also been applied to autonomous underwater vehicles [27], flexible manipulators [22], and factory workcells [19].

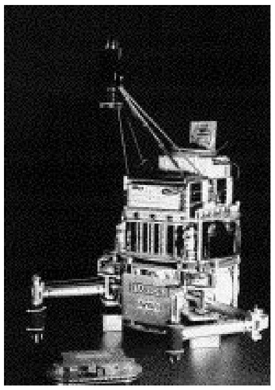
Figure 3. Experimental platform

In OBTLC, the robot handles its own low-level control locally while the operator provides high-level task commands to the robot. The operator specifies treatment of an object, rather than motions of the robot. (In the context of OBTLC, an *object* is a physical entity around which a task command may be constructed.) For instance, the user might command a robot to put a rock into a sample container, but would not specify the timing or trajectory of the task. The result is a human-robot team where the responsibilities are suited to the capabilities of the human and the robot.

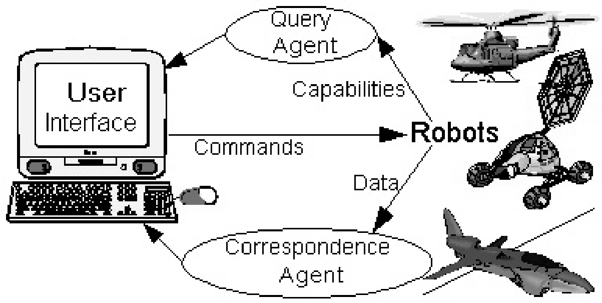

Figure 4. Basic System Diagram

However, OBTLC requires that the robot maintain a list of objects that it senses, and a list of tasks that it is capable of performing. The robot must then maintain knowledge of which tasks are possible for each individual object, resulting in a clutter of interrelationships. When considering the command of many such robots, the organization of just these relationships is a daunting task.

3.2 Agents

To sort through the complication of a deployment of many OBTLC-capable robots, we introduced an agent architecture for communication between operator and robots. For instance, an agent listens to the communication traffic and maintains interlinked lists of robot capabilities and sends the appropriate ones to the interface when queried. As a result, the user is only afforded functions that a robot is capable of performing at that moment.

The agent architecture we employed was the Open Agent Architecture developed at SRI (http://www.ai.sri.com/~oaa). OAA is focused on building distributed communities of agents, where *agent* is defined as any software process that meets the conventions of the OAA society. An agent satisfies this requirement by registering the services it can provide in an acceptable form, by being able to speak the Interagent Communication Language (ICL), and by sharing functionality common to all OAA agents such as the ability to manage data in certain ways [15].

Agents are useful at handling the transactions between robots and the operator because they can provide a consistent interface regardless of the number or type of robots. The agent used most often is the Query Agent, which listens to robot-operator traffic and sends its own requests for information to establish lists of robots, tasks, and objects that the operator might be interested in knowing. For instance, a robot only knows which objects it sees and the tasks it can perform on each of those objects. The Query Agent gathers this information and assembles lists of all tasks that a robot can perform at a given time, or a list of which robots sense a particular object. The interface can then access these lists through a standard call without requiring it to implement its own information-collection procedures.

Another agent under development, the Correspondence Agent, monitors robot output to determine which objects seen by two different robots are actually the same. This correspondence is based on object characteristics and location and is accessed by the interface to allow the operator to give commands for cooperative efforts.

3.3 Command process

Commands are formed through an iterative process (Figure 5) that ultimately affords only the most current capabilities of the robots to the user. A screen shot of the graphical user interface during an example task assignment is shown in Figure 6.

For example, assume that an operator's goal is to have a robot relocate a science instrument. The robots have periodically published a list of every object that they sense. The interface shows graphical representations of the robots and these objects. The operator clicks on a robot to select it, and then she clicks on an object sensed by the robot. The interface queries the robot to determine the tasks that the robot can perform on that object, which in the example shown is Retrieve, Move, and Reset, and presents the tasks to the operator in a pop-up dialog box (Figure 7) next to the object. A command packet is constructed by the interface and sent to the robot, which adds it to its queue of upcoming tasks.

Data: Interface to Robots	*Data: Robots to Interface*
	State info for all robots
Selected: <Robot A>	
	Visible for Robot A: <Obj1>, <Obj2>
Selected: <Obj1>	
	Possible for <Robot A> and <Obj1>: <Task1>, <Task2>
Command: <Robot A> <Task1> <Obj1>	
	Acknowledge Task

Figure 5. Command formation process

For commands to multiple robots, the process is much the same. The only difference is that the object being operated on must be sensed by all robots and must have been determined to be the same object by the Correspondence Agent.

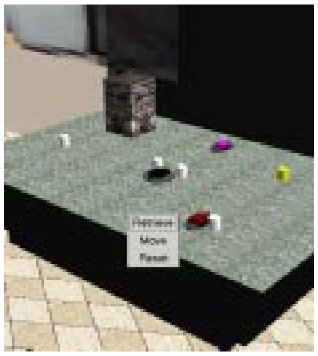
Figure 6. Screen shot of GUI

One of the most important aspects of this architecture is that tasks are conceived and commands are given in the local frame of the robot. Rather than formulate tasks based on a world model that may be inconsistent with the sensors of the robot, we decided to maintain each local model of the robot separately. This concept recognizes that the robot is best suited for controlling itself locally, and commands should only provide the minimum amount of information necessary to accomplish a task. Consequently, the robot is only asked to act on objects that it senses, and interactions with objects can take place in relative coordinate frames rather than global frames. What the user sees may or may not indicate variations between these local contexts. The interface may highlight or suppress this information depending on its application.

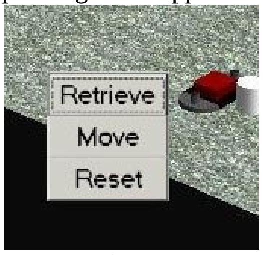

Figure 7. Close-up of Task Selection Window

There are two types of tasks -- those that are preemptive of all other tasks and those that can be added to the queue of tasks the robot must perform. The task descriptions that the robot provides also note whether the task requires some modifying information such as a destination.

4 Experimental results

Experimentation utilizing this new architecture was recently begun using the free-flying space robots at the Stanford University Aerospace Robotics Laboratory. These three robots float on an air bearing on a large granite table to emulate the zero-gravity, zero-drag dynamics of space with high fidelity in two dimensions. The robot shown in Figure 3 has cold-gas thrusters for propulsion, two manipulators with pneumatic grippers, on-board computation and power, and wireless communications. The robot is capable of autonomous navigation, picking up and placing objects, catching and throwing objects, escorting and monitoring tasks, and simple construction primitives.

Thus far, only one robot has been converted to communicate with the agent architecture and accept commands from the new interface. Initial tests have been very positive, as new and experienced users have found the system intuitive.

New visitors to the laboratory were able to immediately operate the robot after very little training and felt comfortable utilizing all of the robot capabilities. These users were told only that selecting a robot and then clicking on objects in the environment would provide enough information to operate the robots. The users could then see the task dialog box when they clicked on an object visible to the selected robot, and they readily chose from the options presented.

Informal surveys of advanced users have found that the operators trust the robot more because of the assurance that the task affordances are recent and valid. They also compare the list of tasks against their expectations for quick debugging of the robot during experiments.

5 Conclusions and future work

This research has led to the development of a control architecture that leverages a familiar human-computer interaction in use by many millions of users. Most of the standard interaction primitives had been resolved through generations of game development, so we were able to concentrate on the issues that resulted from using robot hardware.

While implementing the RTS interface for this robotic system, we realized that the underlying architecture is easily expandable to other interfaces that allow a dialogue to communicate current affordances. For instance, voice-only systems and text-based screens should also provide the necessary affordances to control multiple robots in some situations. We plan to explore these options and compare them to the graphical representation method.

References

1. M. Adams et al, "An Automation-Centered Human-System Integration Architecture for Autonomous Vehicles," *AUVSI '98*, 1998.
2. R. Arkin & T. Balch, "AuRA: Principles and Practice in Review," *Journal of Experimental and Theoretical Artificial Intelligence*, 1997.
3. R. Arkin et al, "Tactical Mobile Robot Mission Specification and Execution," *Mobile Robots XIV*, 1999.
4. R. Aylett et al, "Supervising multiple cooperating mobile robots," *Autonomous Agents 97*, 1997.
5. D. Brock et al, "Coordination and Control of Multiple Autonomous Vehicles," *Proc. of the 1992 IEEE Conference on Robotics and Automation*, 1992.
6. D. Cannon, Point-and-Direct Telerobotics: *Object Level Strategic Supervisory Control in Unstructured Human-Machine System Environments*, PhD Thesis, Stanford University, 1992.
7. K. Dixon et al, "RAVE: A Real and Virtual Environment for Multiple Mobile Robot Systems," *Proc. of the 1999 International Conference on Robotics and Systems*, 1999.
8. S. Fleischer & S. Rock, "Underwater Vehicle Control from a Virtual Environment Interface," *Proc. of the Symposium on Interaction 3D Graphics*, 1995.
9. D. Gentner & J. Grudin, "Why Good Engineers (Sometimes) Create Bad Interfaces," *CHI '90 Proceedings*, 1990.
10. R. Gilson et al, "Key Human Factor Issues for UAV/UCAV Mission Success," *AUVSI '98*, 1998.
11. J. Grudin, "The Computer Reaches Out: The Historical Continuity of Interface Design," *CHI '90 Proceedings*, 1990.
12. H. Jones et al, "Human-Robot Interaction for Field Operation of an Autonomous Helicopter," *Mobile Robots XIII*, 1998.
13. D. Lees, *A Graphical Programming Language for Service Robots in Semi-Structured Environments*, PhD Thesis, Stanford University, 1994.
14. "MAGIC2 – Multiple Aircraft GPS Integrated Command and Control System," *AUVSI '98*, 1998.
15. D. Martin et al, "The open agent architecture: a framework for building distributed software systems," Applied Artificial Intelligence, 13(1), 1999.
16. C. Miller et al, "'Tasking' Interfaces to Keep the Operator in Control," *5th Annual Symposium on Human Interaction with Complex Systems*, 2000.
17. C. Monson et al, "The Development of Specifications and Guidelines for the Design of Crew Stations for UAV Systems," *AUVSI '98*, 1998.
18. D. Norman, *The Design of Everyday Things*, New York: Basic Books, 1988.
19. G. Pardo-Castellote, *Experimental Integration of Planning and Control for a Intelligent Manufacturing Workcell*, PhD Thesis, Stanford University, 1995.
20. L. Parker, "ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation," *IEEE Trans. Of Robotics and Automation*, 14 (2), 1998.
21. T. Payne et al, "Varying the User Interaction within Multi-Agent Systems," *Proc. of the 4th Int'l Conference on Autonomous Agents*, 2000.
22. H. Schubert, "Space construction: an experimental testbed to develop enabling technologies," *Telemanipulator and Telepresence Technologies IV*, 1997.
23. T. Sheridan, "Speculations on Future Relations Between Humans and Automation," in *Automation and Human Performance*, 1996.
24. T. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*, Cambridge, MA: MIT Press, 1992.
25. H. Stevens et al, "Object-Based Task-Level Control: A Hierarchical Control Architecture for Remote Operation of Space Robots," *Proceedings of the AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space*, 1994.
26. H. Surmann & M. Theissinger, "ROBODIS: A dispatching system for multiple autonomous service robots," *Proc. of Field and Service Robotics*, 1999.
27. H. Wang, *Experiments in Intervention Autonomous Underwater Vehicles*, PhD Thesis, Stanford University, 1996.