This paper appeared in the *Proceedings of the Conference of the IEEE Oceanic Engineering Society (OCEANS)*, Providence, RI, September, 2000.

Simulating an Autonomous Oceanographic Sampling Network: A Multi-Fidelity Approach to Simulating Systems of Systems*

Roy M. Turner
Elise H. Turner
Department of Computer Science
University of Maine
Orono, ME 04469 USA
{rmt,eht}@umcs.maine.edu

***Abstract*–Many useful ocean engineering systems, such as autonomous oceanographic sampling networks (AOSNs) are actually systems of systems. Such systems are complex and require simulation during design and development. However, adequately simulating such systems is also difficult. We have developed an approach to multi-fidelity simulation that allows systems of systems to be simulated at several points along the spectrum from high-level, low-fidelity to low-level, high-fidelity simulation. We have implemented this approach a simulator for the CoDA (Cooperative Distributed AOSN control) project. Such an approach provides simulation support throughout the lifetime of the project, from design through proof-of-concept and beyond.**

Introduction

Many systems that are useful in ocean engineering are *systems of systems*. For example, an autonomous oceanographic sampling network (AOSN) [1] is a system composed of autonomous underwater vehicles (AUVs) and instrument platforms, each of which is itself a system. Systems of systems are complex, not only because of the complexity of the individual systems, but also because of the complexity of the interactions between the component systems.

Because of the complexity, simulation is an important tool in designing new systems of systems. However, it can be difficult to create an effective simulation for such systems. Because of the many components, it takes a long time to create a traditional, detailed simulation of the entire system. In addition, a detailed simulation of the entire system may not allow designers to isolate specific mechanisms for interactions that should be evaluated. On the other hand, this level of detail is not always necessary to get important feedback that can impact design. However, detailed simulation is ultimately needed to truly evaluate the system.

Our approach is to create a multi-fidelity simulation for systems of systems. This approach includes several possible levels of simulation fidelity, ranging from a high level of abstraction to a high level of detail.

Many approaches allow simulation at an abstract or at a detailed level, or both. For example, simulations based on Petri nets or finite state machines often model the interaction of systems without modeling more than a highly abstract version of each component's state. At the other extreme, simulators such as CADCON [2] model the systems and their interactions in a highly detailed manner. Unfortunately, for these and most other simulators, there is nothing in between: there is no route from abstract, high-level simulation of system properties to highly-detailed, low-level simulation of system components.

Our approach differs by being capable of simulation at several levels of fidelity in a single simulator for the system. These levels are:

Protocol level: The aggregate properties of the entire system are simulated by focusing on the interaction protocols and treating the component systems as black boxes.

Technique level: Techniques are developed that carry out steps in the protocol level, but are not necessarily implemented as algorithms that will be executed by the components.

Algorithm level: Algorithms that a component will use to implement the techniques are selected and implemented.

Full agent in software level: All algorithms required to control the component are im-

*This work was funded in part by grants N0001-14-96-1-5009 and N0001-14-98-1-0648 from the Office of Naval Research. The content of this paper does not necessarily reflect the position or the policy of the U.S. government, and no official endorsement should be inferred.

plemented, and these components interact to carry out missions in a simulated world.

Full agent in hardware level:
Algorithms and interactions are tested using robots in the laboratory.

At any given time, different parts of the system to be simulated can be simulated at different levels of fidelity. For example, a robot in the laboratory may interact with another agent implemented completely in software and still others simulated at the protocol level or with only a few techniques implemented as algorithms. This provides a smooth path between an initial high-level, low-fidelity simulation and the ultimate low-level, high-fidelity simulation of the system. The approach allows us to quickly simulate the overall system properties without requiring commitments to implementation details. More importantly, it allows us to focus on an individual technique and evaluate it in the context of the full system without requiring the full system to be implemented in detail. The approach also modularizes the development process and allows incremental development and testing.

In the remainder of this paper, we will discuss this approach to simulation in more detail. We describe the challenging issues confronting such simulations and how our approach addresses them. We then present an implementation of our approach, the CoDA (Cooperative Distributed AOSN control) simulator [3; 4], which simulates autonomous oceanographic sampling networks.

Simulating Systems of Systems

Approaches to simulating systems of systems, which we will refer to hereafter as *multiagent systems*, have generally taken one of two paths. The first path is to simulate the aggregate properties of the system by concentrating on the interactions between the components. In this approach, the components themselves are treated as black boxes. For example, a Petri net or other mechanism may be used to simulate the evolving state of the entire system as actions are taken and messages are sent between components, but the actual decision-making processes of the agents that would produce the actions or messages are not simulated. This approach can give a very good idea of what the overall system will do, and it is very quick to implement, easy to use for simulation experiments, and easy to change as needed. However, it has the drawback of simulating the multiagent system at only a high level of abstraction. Ultimately, the behavior of the system will depend on the decision-making processes of its components as they interact with their local environment and the other components.

The other path is to simulate to a high level of detail the decision-making processes of the agents comprising the multiagent system—in fact, the actual agent control programs, or something very close to them, would likely themselves be used in the simulation. This approach has the benefit of yielding high-fidelity simulations of what the behavior of the actual system will be. However, it has its price. It is much more difficult to design and implement the agent control programs than it is to simply simulate what they might plausibly do in various situations. This kind of simulator takes much longer to implement. In cases where the agent control programs are themselves being designed, needed information about agent details may not be known or implementation of the simulator may be delayed waiting for the results of the control program research. Such a simulator takes longer to run, too, since the agent control programs may have to run in real time, whereas an aggregate property simulation can often run many times faster than real time.

The first approach is often taken for systems in which the focus is the interaction, not the control programs of the components and, generally, where the components themselves are rather simple. For example, this would be an appropriate kind of simulation for designing and debugging a new Internet protocol. The assumption for this kind of simulation is that either: the components being simulated can be adequately characterized by the rules, state transitions, or other means used to represent their actions; or that the components can be readily designed to some set of specifications that are being simulated. The second approach is often taken for systems in which the focus is on the agent-level control programs, the agents are seen to be very complex, or both. For example, this kind of simulation is the norm in artificial intelligence research, where the agents are very complex. The behavior of such agents generally cannot be reduced to a simple set of rules, nor can they be designed with confidence to meet a set of specifications without the aid of simulation.

Neither approach by itself is sufficient for a simulator supporting development of a multiagent system, such as an AOSN, in which attention has to be paid both to the interaction between protocols and to the complex nature of the components themselves. What is needed is a multi-fidelity approach to simulating multiagent systems, where "multi-fidelity" refers both to simultaneously simulating different components at different levels of fidelity as well as supporting different levels of simulation fidelity over time as development progresses.

Multi-fidelity Simulation

We have taken an approach to simulating systems of systems that calls for simulation of components and their interactions at several different levels, both over the course of the development of the multiagent system and simultaneously, with different components simulated at

different levels.

Early in the development of a multiagent system such as an AOSN, it is important to quickly simulate aggregate properties of the system. We refer to this level as the *protocol level*, since what is being simulated here is typically the protocols the components of the system use to communicate and cooperate with one another. This allows the researchers to quickly try different high-level approaches to controlling the system without the need to commit to or implement simulations of agent control program details. Many dead-ends can be detected and avoided by using this kind of simulator to predict general properties of the system, saving much effort that would otherwise later be wasted.

As work progresses, it is important to be able to try out different techniques for problems the agents must solve. For example, in an AOSN, mission tasks need to be matched up with AUVs or instrument platforms that will perform them. There are many ways of approaching this task assignment problem. The simulator should support easily trying these different techniques in the context of the full system. It should not at this stage, however, require a commitment to how the technique will ultimately be implemented aboard an actual agent. Such a commitment may not be possible at this stage, and in any case, it risks confounding properties of the technique with implementation details. We refer to this level of simulation fidelity as the *technique level*. For example, constrained heuristic search (CHS) [5] is an attractive candidate for task assignment. A distributed version, DCHS [6], exists and would likely be used in the AOSN. Implementing the distributed version would require developing the algorithms to be used by participating agents as well as knowledge of details of these agents' processing capabilities and means of communication. We can avoid this overhead by first implementing a CHS-based task assignment mechanism and testing it in the context of the AOSN simulation. If the technique is successful, DCHS can be implemented in the algorithm level.

As techniques are selected for various agent control problems, implementation commitments will be made. It then will be possible to write the code that will run aboard the agent in the real world. It is likely that this will occur for some techniques before others, and so it may be desirable to test the implementations before the full agent control programs are ready to be tested. The simulator should support this, as well. This level of simulation, the *algorithm level*, entails some parts of the multiagent system being simulated with an extremely high level of fidelity to the ultimate system while the rest of the system is left at lower-fidelity levels (i.e., protocol and technique levels). For example, a DCHS-based task assignment mechanism might be implemented and used in the simulation while the rest of the agent is being simulated at the protocol level.

As work progresses, more and more of the agent control programs will be fully specified and implementable. This leads to the next level of simulation, the *full agent in software level*. This level is equivalent to the second traditional simulation approach mentioned at the start of this section; it can give very good predictions of the performance of the actual multiagent system. One agent may be implemented at this level while others are implemented at other levels, thus allowing this level to co-exist with others in the multi-fidelity simulator.

A simulator will ideally support yet another level. The highest-fidelity level of simulation is the *full agent in hardware* level. At this level, the agent control program is controlling portions of the actual hardware that will be fielded, ideally the entire agent hardware, while still remaining in a simulation setting. For example, one could imagine an AOSN simulator in which some AUVs are being simulated, while others are actually in the ocean, with their positions and and other data being integrated into the ongoing simulation.

It is important to note that a multi-fidelity simulator should not require all components to progress through these levels at the same rate. Indeed, not all components need to progress at all. For example, the simulator should allow researchers concentrating on task assignment or agent organization techniques to try out their solutions in the simulator before other pieces, such as communication modules or mission planners, are ready. Some agents may never be simulated at high fidelity. For example, this might occur when the system is to include components to whose internals the researchers do not have access, or when testing how the system would behave in the presence of agents that do not yet exist.

A multi-fidelity simulator provides a smooth migration path for research on multiagent system control from the earliest design phases, supported by simulation at the protocol level, through implementation and full proof-of-concept, supported by simulation at the full agent in hardware level. In order to support this, the simulator must have a flexible skeleton that the simulated components, regardless of their level of simulation, can "plug into." At the aggregate level, this may mean adding rules to simulate what the component would do under given conditions. At higher levels of fidelity, it will require well worked-out communication protocols and languages for the pieces to use to talk to the rest of the simulator, either directly, or more likely, via wrappers around the simulated components.

Whether a component is simulated at a high or a low level of fidelity should be transparent to the user. The only difference noticeable to the user of the simulation should be that the higher the overall fidelity, the better

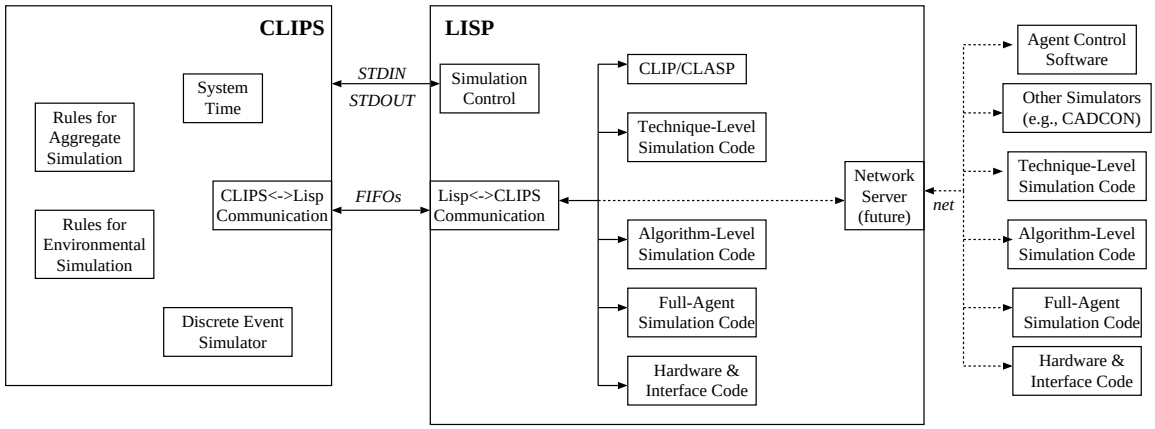

Figure 1: CoDA simulator.

the system's behavior matches the real-world behavior of the simulated system. The level of simulation of a component should be transparent as well to the other simulated components. For example, a full-agent simulation of an AUV mission controller should not need to know that a mooring is being simulated at the aggregate property level. Similarly, an AUV deployed in the ocean and communicating with the rest of the simulation should need only changes in the way it communicates (i.e., through the simulator rather than directly to other components) and changes to allow it to "see" other AUVs and instrument platforms.

It is also important to allow the user to specify at what level he or she would like the simulation to run. Although components cannot be simulated at higher fidelity than their development stage allows, it will often be useful to run the simulation at lower fidelity, for example, to run experiments faster than real time.

Finally, it is important that, at all levels, the simulator provide data collection and simulation monitoring services to the user.

A Multi-Fidelity Simulator

In this section, we discuss an example multi-fidelity simulator, the CoDA simulator. The CoDA (Cooperative Distributed AOSN control) project [7; 3; 8] has as its goal developing intelligent control techniques for autonomous oceanographic sampling networks. The current phase of the project focuses on autonomous organization and reorganization of the AOSN as well as the problem of task assignment. Our ideas about multi-fidelity simulation grew out of this project's need for a way to rapidly prototype communication and cooperation protocols that would also allow the organization and task assignment work to be folded into the simulator as it matured.

The architecture of the current simulator, herein just called CoDA, is shown in Fig. 1. The skeleton of CoDA, including the simulation controller, is written in Lisp and runs under Unix (Linux and Solaris). Protocol-level simulation is currently done in the rule-based expert system language CLIPS [9]. CLIPS also maintains the simulated system time, and it provides discrete event simulation and simple environmental simulation. CLIPS is started and controlled by the Lisp-based simulation controller.

Communication between CLIPS and Lisp is done in two ways. First, CLIPS' standard input and standard output streams are written and read by Lisp to send commands and receive information, respectively. Second, Unix FIFOs (named pipes) are used to transfer information between Lisp and CLIPS. For example, pipes are used when CLIPS requests that Lisp carry out some higher-fidelity simulation such as running a task assignment algorithm.

Protocol-level simulation is done by CLIPS. Rules are used to represent what agents would do under different situations. The rules' actions simulate what the agents do when following the cooperation protocols, but not how they arrived at the decisions necessary to do follow them. Fig. 2 shows a portion of the output from this level of the simulator as the AOSN self-organizes. (A more complete description of this process can be found elsewhere [3].)

As long as the simulator has any portion that is being simulated at the protocol level, CLIPS performs other simulation functions, such as maintaining the system time. This is done by using the discrete event simulator (DES), implemented as CLIPS rules and objects. Protocol-level simulation rules post events that should occur at some time in the future. When the rules reach quiescence, DES moves the system time to the next event and runs the event. For example, communication via acoustic link is currently modeled this way. When an agent sends a message, rules determine the travel time for the message through the water before it arrives at its

```
[CLIPS] 00:00:04.0 (MLO) new agent VIP10 broadcasting organization-present?.
[CLIPS] 00:00:04.0 (MLO) new agent VIP10 setting timer 1 to wait for replies.
[CLIPS] 00:00:04.0 (MLO) new agent VIP11 broadcasting (non-CDPS) organization-present?.
    [...DES moves time ahead to simulate acoustic link message transit time...]
[CLIPS] 00:00:05.05 (MLO) VIP11: received organization-present? message from VIP10
[CLIPS] 00:00:33.01 (MLO) VIP16: received organization-present? message from VIP13
[CLIPS] 00:00:33.02 (MLO) VIP14: received organization-present? message from VIP13
    [...]
[CLIPS] 00:00:34.0 (MLO) VIP10: waited long enough for organization-present? replies.
[CLIPS] 00:00:34.0 (MLO) VIP10: initiating MLO formation with agents = (VIP10 VIP13)
[CLIPS] 00:00:34.0 (MLO) VIP10: broadcasting first initiate-MLO message.
    [...]
[CLIPS] 00:01:05.05 (MLO) VIP13: completed MLO formation.
[CLIPS] 00:01:05.05 (SIM) MLO formed, contains members (VIP10 VIP13 VIP9).
[CLIPS] 00:01:05.05 (SIM) switching context -> MLO discovery
```

Figure 2: Output from the protocol level of simulation. (Edited slightly for clarity.)

destination and posts an event for that amount of time in the future. At that time, the recipient will receive the message. (For broadcasts, travel time is determined to all agents and corresponding events are posted.)

As portions of the simulation proceed from the protocol level toward more high-fidelity simulations, more of the simulation functionality will migrate out of CLIPS. When CoDA is running completely in a high-fidelity mode, then CLIPS will need do little.

As shown in Fig. 1, technique-level simulation can occur in two places in CoDA, within Lisp and external to it. Currently, it is done within Lisp. In order to try out a new technique, the protocol-level simulation of the agent using the technique must be modified to request that CoDA run the technique when it is needed. This entails modifying a few rules and defining a request type, then modifying a Lisp variable to link the request type to the function to call when the request is made. For example, the current simulator uses a technique that is a modified version of constrained heuristic search (CHS) in order to do task assignment. Requests were defined to transfer information from the CLIPS protocol-level simulation into Lisp data structures (e.g., "add-task," "add-alternative," etc.), and a request was added to invoke CHS ("solve-problem"). CLIPS rules pertaining to task assignment were modified to send Lisp these messages and wait for Lisp's reply. CoDA's Lisp portion, when it receives the message, calls the CHS top-level function, then sends the results to CLIPS. Fig. 3 shows a portion of this process.

At the present time, we are working on an algorithm-level simulation of the task-assignment mechanism. This will be a distributed version of CHS. It will be added to the simulator in much the same way as was sketched above for the technique-level CHS.

Although Fig. 1 shows full-agent and hardware-level simulation components as also being possible to include within Lisp, most likely these will be external to the main portion of the CoDA simulator. As shown, entities external to CoDA can also be used to implement technique-level and other levels of simulation. In the near future, we will add a network communication module to CoDA. This will communicate using sockets with simulated components. This will allow the pieces of the simulation to be written in virtually any language and run on multiple, possibly remote, systems. For example, we anticipate using the Orca AUV mission controller [10] to control simulated AUVs in CoDA. Orca will likely run in its own Lisp process on another machine and communicate with the agent "body" by exchanging messages with CoDA. CoDA may simulate the body internally, or that, too, may be external. For example, Orca's commands to the AUV may be passed along to another external simulator, such as CADCON [2], which can provide high-fidelity AUV simulation, or they may be passed along to actual robots connected to CoDA via the Internet.

Experiment control, as well as data gathering and analysis facilities are also provided by CoDA. CoDA uses CLIP/CLASP (Common Lisp Instrumentation Package/Common Lisp Analytical Statistics Package) [11]. This allows experiments to be defined in data files, then run under the control of CoDA. Data are gathered based on the experiment definitions and can be analyzed statistically by CLASP or exported for use by other statistics programs.

The current CoDA simulator is being extended in several ways. The network server mentioned above will soon be added. This will allow graphical user interfaces (GUIs), for example, to be easily added to CoDA. One

```
Received request to start new TLO construction; deleting old problem(s).
New TLO problem will be referred to as `"gen1099"'.
[CLIPS] 00:03:11.0 (MLO) VIP30: creating new search problem for assignments, gen1099
[CLIPS] 00:03:11.0 (MLO) VIP30: Assigning tasks.
Attempting to solve problem "gen1099".
Problem "gen1099" is solved.
Sending to CLIPS: problem status of "gen1099" is SUCCESS.
[CLIPS] 00:03:11.0 (MLO) VIP30: status of problem gen1099 is SUCCESS.
[CLIPS] 00:03:11.0 (SIM) Creating object gen1132 to represent the TLO.
[CLIPS] 00:03:11.0 (MLO) TLO created.
```

Figure 3: Output of a technique-level simulation. A TLO is the task-level organization that conducts the mission; MLO is the meta-level organization that designs the TLO.

subproject currently under way will create a Java/VRML (Virtual Reality Modeling Language) GUI that receives information from CoDA and generates and maintains a virtual reality model of the simulated system.

The addition of the network server will allow another major extension, the addition of hardware and full-agent simulation abilities. Our work is currently being extended to simulate a system of autonomous agents that are either real robots (Pioneer land robots) or simulated robots [12]. The real robots will be augmented with the ability to simulate AUVs; for example, they will each have a virtual depth that the simulator maintains for them in addition to their real XY location. This simulator connects the robots and simulated robots, as well as their control software, via the network. With this extension, CoDA will have the ability to simulate an AOSN at any of the levels of fidelity discussed above, from the protocol level to the full agent in hardware level, either with all components at the same level or with components at different levels.

Conclusion

Systems of systems, such as autonomous oceanographic sampling networks, are important in ocean science and engineering. Such systems are complex and difficult to design and implement without simulation. Unfortunately, adequately simulating such systems is itself difficult. High-level, low-fidelity simulators do not predict the system's behavior in detail, yet low-level, high-fidelity simulators are labor intensive and require a long time to build.

We have here argued for multi-fidelity simulation for systems of systems. Such a simulator can provide a smooth transition path from high-level, low-fidelity simulations to low-level, high-fidelity simulations, thus supporting the development process from design through proof-of-concept and beyond. We have found several levels of simulation useful. *Protocol level* simulation concentrates on the aggregate properties of the components of the system as they interact using their cooperation protocols, but does not make any commitments to *how* they follow the protocols. *Technique level* simulation makes some commitment to the techniques to be used by the components, but does not commit to the actual implementation. *Algorithm level* simulation commits to the algorithms that will actually carry out the techniques and uses implementations of those algorithms in the simulation. *Full agent in software level* simulation simulates the decision-making processes of complete agents. *Full agent in hardware level* simulation makes use of actual hardware implementations of the agents (e.g., robots). At any given time, the various simulated components may be in different levels of simulation.

We have implemented a multi-fidelity simulator for the CoDA project to simulate AOSNs. So far, this simulator incorporates the protocol and technique levels, and the algorithm level is currently being worked on. An extension will add functionality to bring the full agent in software and full agent in hardware levels into CoDA. The CoDA simulator allowed us to rapidly prototype cooperation protocols for AOSN self-organization and reorganization as well as techniques for task assignment. In the near future, we will be using the simulator to develop and test algorithms for task assignment and techniques and algorithms for organizational structure selection. With the extension, we will be able to test and further develop our AOSN control approach using full agent simulation, both in software and in hardware, as a final step before in-water testing.

Acknowledgments

The authors thank the other members of the Cooperative Distributed Problem Solving (CDPS) Research Group, part the Maine Software Agents and Artificial Intelligence Laboratory (MaineSAIL), and our colleagues at the Autonomous Undersea Systems Institute (AUSI)

for their insightful comments and other help over the course of this work.

References

[1] T. B. Curtin, J. G. Bellingham, J. Catipovic, and D. Webb. Autonomous oceanographic sampling networks. *Oceanography*, 6(3), 1993.

[2] S. G. Chappell, R. J. Komerska, L. Peng, and Y. Lu. Cooperative AUV Development Concept (CADCON) – an environment for high-level multiple AUV simulation. In *Proceedings of the 11th International Symposium on Unmanned Untethered Submersible Technology (UUST99)*, Durham, NH, August 1999. The Autonomous Undersea Systems Institute, Lee, NH.

[3] R. M. Turner and E. H. Turner. Organization and reorganization of autonomous oceanographic sampling networks. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation (ICRA'98)*, pages 2060–2067, Leuven, Belgium, May 1998.

[4] E. H. Turner and R. M. Turner. A constraint-based approach to assigning system components to tasks. *International Journal of Applied Intelligence*, 10(2/3):155–172, 1999.

[5] M. S. Fox, N. Sadeh, and C. Baykan. Constrained heuristic search. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, 1989.

[6] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, 1991.

[7] R. Turner, E. Turner, and D. Blidberg. Organization and reorganization of autonomous oceanographic sampling networks. In *Proceedings of the 1996 IEEE Symposium on Autonomous Underwater Vehicle Technology*, pages 407–413, Monterey, CA, June 1996.

[8] R. M. Turner and E. H. Turner. A two-level, protocol-based approach to controlling autonomous oceanographic sampling networks. Unpublished; submitted to the *IEEE Journal of Oceanic Engineering*.

[9] J. C. Giarratano. *CLIPS User's Guide*. NASA, Information Systems Directorate, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX, 1993.

[10] R. M. Turner. Intelligent control of autonomous underwater vehicles: The Orca project. In *Proceedings of the 1995 IEEE International Conference on Systems, Man, and Cybernetics*. Vancouver, Canada, 1995.

[11] S. D. Anderson, D. L. Westbrook, M. Schmill, A. Carlson, D. M. Hart, and P. R. Cohen. *Common Lisp Analytical Statistics Package: User Manual*. Department of Computer Science, University of Massachusetts, 1995.

[12] C. Grunden. Master's thesis. Unpublished; to be completed August, 2000.