# Joint Evaluation of Mission Programming for Underwater Robots

Eve Coste-Manière
INRIA Sophia Antipolis
Eve.Coste-Maniere@inria.fr

Howard H. Wang, Stephen M. Rock
Stanford University
{lazarus,rock}@sun-valley.stanford.edu

Alexis Peuch, Michel Perrier, Vincent Rigaud
IFREMER Toulon
{peuch,mperrier,rigaud}@ifremer.fr

Michael J. Lee
MBARI
lemi@mbari.org

**Abstract**

*Underwater robotic systems require a control architecture that handles automatic control aspects and discrete-event management while considering real-time issues within a formalized framework in order to achieve a high degree of autonomy and robustness while performing missions. To extract generic characteristics of working implementations, we examine two methodologies developed independently by French and American research organizations to control their underwater robots. In particular, we study how a specific reactive and complex mission is encoded under the two different methodologies. Details of the methodologies along with mission programs created by each team to accomplish the evaluation mission are presented.*

## 1 Introduction

With the current state-of-the-art in underwater robotics, many aspects remain to be tuned in order to create an efficient and successful robotic application while achieving a high degree of autonomy and robustness. In an effort to foster greater cooperation and exchange of ideas between research institutions and across international borders, a joint study has begun which will compare and contrast the results of independent research programs. This paper presents preliminary investigations and evaluations in the field of control architectures between two American (MBARI, Stanford Aerospace Robotics Laboratory) and two French (INRIA, IFREMER) institutions. Two underwater, robotic vehicles are being used to support the study: the VORTEX vehicle (Figure 1) on the French side and the OTTER vehicle (Figure 2) for the American participants. The US Naval Postgraduate School is also participating in this study with their PHOENIX AUV. Each group performs research in control/software algorithms and architectures that enables these robots to be self-guided with minimal human input, i.e. *high-level control*.

A careful examination of the literature (see [5] for extended references) shows that almost every single organization working in robotics, underwater or land-based, has their own custom control/software architecture that is dedicated to some specific hardware system(s). It is easier to determine the general requirements of a control architecture by analyzing existing ones that are used to control real hardware in actual applications. By analyzing the response of the robots as they perform the same task or mission and by comparing the different programming styles, methods and philosophies, we hope to be able to conclude in more generic terms what are the attributes and capabilities of a high-level controlling program required for both autonomous and semi-autonomous underwater robots. Two main points are presented in the following sections. First, we identify some fundamental attributes of mission programming methodology and describe the ways in which they have been tackled by two different architectures. Then both approaches are illustrated through the presentation of the encoding of a generic underwater mission by both research teams using their particular methodologies.

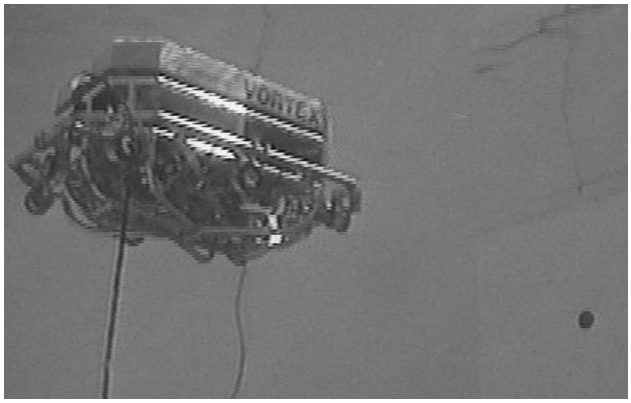


Figure 1: The VORTEX in its test pool

## 2 Mission Programming Methodology

Robotic systems achieve their fundamental scope and usefulness thanks to their continuous interaction with


Figure 2: The OTTER in its working environment

the external world. They attain a high degree of autonomy and robustness while performing missions if they can plan and execute actions, build and update maps, detect and react to defined events, and sense and adapt their behavior in response to the environment. Obviously, this cannot be performed with a single, unique action, however complex, that is dedicated to a particular control situation. In order to carry out complex missions, a well-formalized structure or "control architecture" that handles automatic control aspects, real-time issues, reactivity, discrete-event management and control-law switching within an integrated framework is therefore required.

For the purpose of dealing with the global realization of a mission that requires planning, control and execution, successful architectures will integrate the planning aspects of top-down approaches with the real-time and physical aspects of the bottom-up approaches. This joint evaluation starts with the investigation of some key issues at the "mission" or "task" level where the merging can be done. This level serves as the coordinator between a human user or intelligent planner and the execution layer of the robotic system. In the following, we have identified some important functionalities that must be built into a mission-level programming methodology.

## 1. Specifying elementary robotic actions

These actions form the basic elements that are instantiated and manipulated by the mission program and reside at the lowest layer of the control hierarchy interacting directly with the sensors and actuators. Elementary robotic actions can be created using classical methods with discrete-sampling control theory or through techniques such as neural networks or fuzzy control. In many systems *Control laws* or *control modes* are other terms that are synonymous with elementary robotic actions. For any control architecture, formalized methods to specify, define, encode, and otherwise create an elementary action are needed to present a clean and uniform interface to the higher mission-programming level.

For VORTEX, the robot-task concept introduced by the ORCCAD [11] system has been adopted. Its simplified scheme is composed of a parameterized, control law and a logical behavior which handles the start, stop or typed exceptions during the life of the control law. Specifically, control laws implemented in PIRRAT (the real-time control software of the VORTEX vehicle) are encapsulated with a logical behavior imposed by the set of events that govern its execution. ESTEREL wrappers add the local behaviors to the control laws.

To design the various control laws, elementary modules (e.g. filters, estimators, etc.) are created and linked in a predefined block diagram configuration. Any type of control law, such as non-linear PID or task-function (sensor-based) control laws can be implemented with it. Real-time code can be generated after a simulation using the Matlab/Simulink software. The control laws are then executed in the PIRRAT real-time controller which can also handle and generate discrete events occuring during the execution of the control law by monitoring the vehicle state and its environment.

For OTTER, the corresponding capability is provided under the Task-Level Control paradigm (TLC) using the CONTROLSHELL real-time software framework [10]. CONTROLSHELL is used to create software *components* that represent the most basic modules of a control system. Examples of components are digital filters, analog-to-digital device drivers, PID and LQR controllers, estimators, and basically, any piece of a control system block diagram that is executed periodically in a sampled-data, digital system. With a graphical editor, one connects the inputs and outputs of each component to define how the data flows through the components. Then various *configurations* are defined by grouping different components together. These configurations are, in essence, different control modes for the system.

## 2. Switching of actions

In most, if not all, realistic, underwater missions, different control laws will be used to control the robot during the mission. For example, a robot may switch from a control law using a global long base-line position sensor to a control law defined for a relative sonar sensor as it moves from open water towards a cliff whose position is not well-known globally. Thus, switching the control mode of a robot is a basic function of the control architecture. The smooth, bumpless transfer of control between the termination of one control mode and the initiation of another is an established area of research. However, existing control

architectures do not always explicitly provide mechanisms to assist in making a smooth exchange of control modes.
For VORTEX, a particular protocol will be adopted and extended in order to gracefully and efficiently switch the executing robot-tasks when triggered by the mission program. Similar to the ORCCAD system [8], this switching mechanism will associate starting, nominal and ending methods to the various components. They will be judiciously combined to provide a correct exchange between actions.
For OTTER, higher-level entities can command the lower layer to switch configurations (control modes). As a part of the switch, the shutdown methods of all the modules making up the terminating configuration are automatically executed as well as the startup methods of initiating configuration. These methods provide the mechanism for the user to define a smooth transition during control mode changes.

### 3. Event driven behavior

Events are changes in the robot and environment states that occur asynchronously with respect to the execution of the mission program. Robotic systems are driven by events to complete complex actions (or missions in the case of underwater robots). Typical events are "obstacle detected" or "object grasped". A mission programmer must be able to define and link events to actions. The underlying control architecture manages the propagation of events to the parts of the mission program that need to execute upon event detection.
To describe the event driven behavior of the actions and missions performed by the VORTEX, the synchronous formalism [1] has been adopted through the use of the ESTEREL [2] language which provides an imperative style and a rich development environment to program reactive systems [4]. Because of the choice of the synchronous assumption, a mission program written in ESTEREL is then automatically translated at compile time into a finite automaton which ensures the global control of the application and the robot-tasks. Also, it is directly possible to benefit from tools that allow the user to check for and verify logical correctness [9] on finite automaton.
Observation functions are used to trigger events that connect the ESTEREL program with its real environment. Dedicated services between the mission level and the execution level [3] allow the transformation of physically meaningful signals into logical events as understood by the ESTEREL mission program.
For the OTTER, physical activity are connected to user-defined events and reactions to events through an enhanced finite-state machine (FSM) provided by CONTROLSHELL [10]. The FSM is created graphically and follows the paradigm of the Mealy machine where the work is accomplished during transitions between states. CONTROLSHELL's FSM extensions over basic finite-automata theory include allowing an event to trigger a transition to any one of a set of states that is based on the output of the transition. In addition, transitions based on values of variables are allowed, and the concept of finite-state subroutines is introduced.

### 4. Sequencing of actions

Complex missions or robotic tasks are usually a sequence of elementary robotic actions that when executed successfully accomplish a specific goal. These actions are commonly guarded against anomalous conditions with alternative action sequences in case the primary one fails. The main purpose of the mission programming methodology is to provide the user with an intuitive and powerful interface with which to link actions (sometimes executing concurrently) together along with facile exception handling and making the use of the switching mechanism introduced previously.
So far, for VORTEX, the mission programming language makes use of the ESTEREL language. To fully benefit from the ESTEREL language with no need to know the intricate aspects of the language, high-level primitives will be implemented in order to provide classical functionalities and to render the mission programming process easier such as sequencing, parallelism, conditional branching, and iteration. The control structures are identified as textual primitives. They are used to write a high-level mission program that handles primitive actions, increasing the degree of complexity for example by adopting the robot procedure approach [6]. Then, the program is in turn translated into ESTEREL.
For OTTER, a mission is decomposed into phases that are represented by states in the finite-state machine. The transition between states are specified by expected events that indicate the completion of a mission phase. Typically, during a transition a new elementary robot action is initiated (by changing configurations of the low-level controllers and desired robot state), and processes monitoring the progress of the next phase of the mission are spawned. Since there may be multiple ways that an action may end, different events corresponding to possible conclusions can be specified to trigger alternative subsequent actions. A global state exists to catch all events that must be monitored during the entire mission such as low batteries or water leakage.
While it is expected that any control architecture that has been implemented on actual working robots has an instantiation of this level of control, we submit that the most important property of this level (and furthermore, of any level in a hierarchy) is the *formal definition of the abstraction* it provides. In this way, a uniform interface to the adjacent layers of control will yield to convenient and flexible mechanisms used to create, modify, and execute the entities at that level.

## 3 Common Mission Evaluation

Once the two programming methodologies have been placed into a common *theoretical* framework, the

equivalence and disparity of the *practical* application of the two paradigms in actual experiments will be needed to draw general conclusions. Thus, we have defined a generic mission to be encoded under the two architectures that will be executed the underwater robot testbeds of each team. This generic mission features many of the functionalities that future AUV missions for science and industry will need, and thus, exercises the capability of a mission programming system to control a robot to accomplish real underwater missions.

### 3.1 Description of the Generic Mission

We chose a mission profile related to subsea intervention that could be effectively performed at sea with complex and heterogeneous systems. This mission scenario (depicted in Figure 3) involves tasks in a structured environment [8] that require the integration of aspects of teleoperation together with full autonomous control within the same framework. Reactivity handling and reliability are of prime importance (e.g. obstacle avoidance, user interaction).

The vehicle transits to its starting position upon user command. Then it proceeds towards a wall and starts following it. A visual target (such as a pipe to be inspected) is located on the wall. When the target appears within the vehicle camera's field-of-view, the user commands the vehicle to visually servo upon the target, and thus simulating an inspection task. Then, the vehicle is teleoperated to imitate an operational action such as "go and fetch a tool". When the target reappears within the workspace, the vehicle is again servoed visually upon the target image, mimicking a manipulation task. After a while, the user commands the vehicle to go back home.

The entire mission is performed under user supervision who can issue an "abort" command causing the vehicle to start a homing procedure. During mission execution, the presence of an obstacle will cause the vehicle to station keep until the obstacle is removed from its nominal path. Then, the vehicle proceeds and continues the mission. More drastic exceptions (such as alarms for water leakage) can occur at any time during the mission execution and must be safeguarded.

### 3.2 Mission Program Implementation

We will now discuss how the generic mission is encoded using the two programming methodologies described previously. Whether using the French Esterel/Pirrat or the American TLC/ControlShell, automatic control considerations are first taken into account in order to design the various control laws needed at the execution level [7, 12]. Then, the logical and temporal arrangements of the elementary actions needed to drive the robot to accomplish the mission are completed using the tools provided by the two different methodologies.

#### Implementation with Esterel/Pirrat

**The Robot-Tasks:** Ten robot-tasks have been derived from seven control laws of various complexity

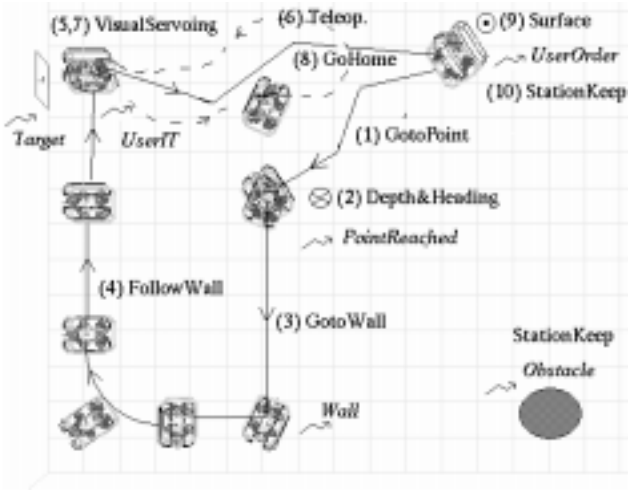


Figure 3: Top view of the nominal mission

including sonar and vision-based control. Robust control is used for several of the control laws and PID control approach [7] has been successfully adopted for robust control considerations.

The functionalities involved for each robot-task are briefly detailed in the following: With the *Depth&Heading*, *Surface* robot-tasks, the vehicle is controlled to reach the desired depth and heading. In the *GotoPoint*, *Homing* and *StationKeeping* robot-tasks, the control law is fed by the Cartesian position of the vehicle expressed in the pool frame and estimated by fusioning telemetric distances provided by the acoustic sounders and the vehicle heading angle. With the *FollowWall* and *GotoWall* robot-tasks, the relative position and orientation of the vehicle are controlled using telemetric distances provided by the two frontal acoustic sounders. The *VisualServoing* robot-task controls the vehicle to stabilize in front of a given target (motionless or not) at a specified distance. Here, the image processing unit feeds the control law with the position error information of the target with respect to the center of the video image. Finally, the vehicle is controlled manually with the *Teleoperation* action.

For each robot-task relevant pre and post conditions (e.g. position reached) have been carefully identified. Also, typed exceptions are raised when critical situations are encountered (e.g. target lost, bad distance); they are handled by the mission program to stop the mission. More details can be found in [3].

**The Mission Program:** The set of primitive goals associated with the generic mission and the sequence of their execution result from an application of goal-driven-like reasoning by the mission developer. The mission originates with a single high-level goal: *"Find Target and Operate with obstacle avoidance"*. This top-level goal is then reduced by introducing simpler but more specific subgoals. Four subgoals have been identified: *"GotoPoint&Depth"*, *"FindTarget"*, *"Teleoperation/Manipulation"* and finally *"Homing"*. They

are handled sequentially during the nominal execution of the program. Taken together, they will satisfy the top goal. Each subgoal consists of the sequencing of various robot-tasks guarded by some safety conditions corresponding to various exceptions raised by the environment or by the actions themselves. The mission is built incrementally with each increment being a stand alone program.
We illustrate the programming of the mission through the example of the simple phase required to ensure the safety of the vehicle if an obstacle is detected. Each robot-task involved in the mission is guarded by the logical handling of the OBSTACLE event. Whenever it occurs, the robot-task is stopped and the station-keeping action (our current recovery procedure) is started. When the obstacle has been removed, the original robot-task is restarted. This behavior can be automatically extended within the robot-procedure context [6]. Then, it is simple and natural to program this mission controller in ESTEREL by sequencing all the guarded robot-tasks and finally wrapping them with the wait of potential events (e.g. WATER_LEAKAGE) that lead to the global abortion of the mission.

```
SafeRobotTask
  Loop
   trap EXCEPTION in
     AnyRobotTask(param);
   ||
     await OBSTACLE;
     exit EXCEPTION
   handling EXCEPTION
     AnyRecoveryProcedure(param);
   end
  endLoop
```

The ESTEREL programming, simulation (XES) and proofs environment was extensively used during the design phase of the program, the compilation phase resulting automatically in the finite automaton that will control the mission. We checked, using MAUTO and AUTOGRAPH [9], that the logical behavior induced by the occurrence of an obstacle was the expected one, first, for all robot-tasks and then in all mission phases. The reduced view of this automaton corresponding to this behavior is shown in Figure 4.

### Implementation with TLC/ControlShell

The following configurations are the main control modes that have been identified for the common mission. We note that the parameterization of these configurations make them useful in many different phases of the mission. For example, in the trajectory following mode, when a trajectory is finished, OTTER automatically maintains its last commanded position. Thus commanding OTTER to station keep is equivalent to changing the control mode to be *trajectory following* with its current position as the end point of the trajectory. Similarly the vision servo mode can be used both to maintain a constant position relative to an object or to move in a desired trajectory relative to the object.

- *trajectory following* : given a set of way points, a smooth trajectory is generated to drive the robot using global position sensors.
- *manual control* : user inputs of desired force are mapped into commands and fed to the thrusters.
- *vision servo* : using on-board cameras and real-time vision processing, the position vector to an underwater target is sensed and used by the control system to maintain a relative position.

The generic mission has been divided into nine phases. These phases are encoded as states in a FSM (see Figure 5). The FSM encoding reflects the sequential nature of this particular mission, i.e. there is only a single transition into and out of many of the states. In other missions where there may be many possible outcomes of a phase, one would see more states with multiple transitions to other states. Each transition is triggered by an event. This event could be a user command such as "CMD_START_MISSION" or the signal indicating the completion of an action such as "END_TRAJECTORY". Also, changes in monitored system state or in the external state can generate events, e.g. "LOW_BATTERIES" or "OBSTACLE_DETECTED". When an event that is defined for the current state of the system is detected, then the associated transition routine is executed. Mainly, these routines change the executing configuration of the low-level control system or modify its parametric values. The routines also spawn monitoring processes that are needed to detect changes and generate events for the next phase of the mission, e.g. finding a target.
A notable feature of the mission FSM is in the implementation of obstacle avoidance. Here the global state "ANY_STATE" is used such that whenever an "OBSTACLE_DETECTED" event occurs, whatever the state of the system may be at the time, a FSM subroutine is executed to handle the exception condition. The "obstacleAvoidance" procedure is truly a subroutine in that after the obstacle-avoidance algorithm finishes, the state of the FSM returns to the original state in which the "OBSTACLE_DETECTED" event occurred.

## 4 Results/Conclusions

Preliminary experimental results of the VORTEX robot performing the mission can be found in [3]. Later this year, we expect OTTER to perform the common mission, and at that time, we will analyze the results of the two experiments in tandem.
However, the direct comparison of robot performance is not the focus of this research. The objective of this joint effort resides in the investigation of the generic components of an architecture as addressed differently by the participants in order to highlight and retain their advantages. The focus is on the mission programming level for robotic systems operating in a dynamic environment. The aim is to provide–at the task level–generic and efficient programming methodologies for rigorous mission specification with a gateway to teleoperation for online user intervention. Future research will focus on compatible architectures between which we can exchange layers and components of common abstraction levels, from planning to execution.

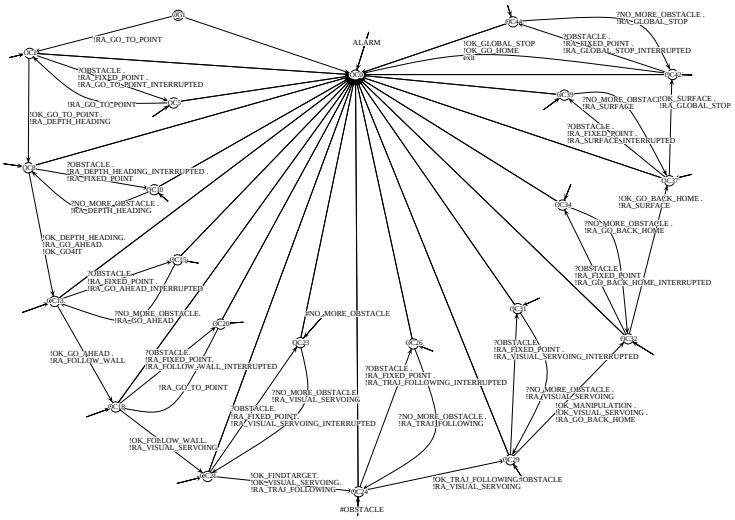


Figure 4: The control automaton: observation of obstacle and actions

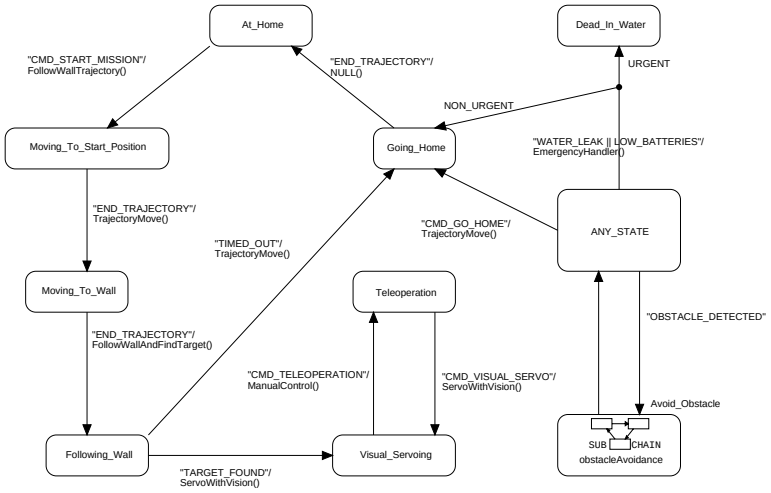


Figure 5: The Fsm encoding the mission strategy

# References


[1] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, Sept. 1991.

[2] F. Boussinot and R. de Simone. The ESTEREL language. *Proceedings of the IEEE, Special Issue*, 79(9), Sept 1991.

[3] E. Coste-Manière, A. Peuch, and M. Perrier. Mission programming: Application to underwater robots. In *Fourth International Symposium On Experimental Robotics: ISER'95*, Palo Alto, USA, June/July 1995.

[4] D. Harel and A. Pnueli. On the development of reactive systems, logic and models of concurrent systems. *Springer-Verlag*, pages 477–498, 1985.

[5] J.-M. Hasemann. Robot-control architectures: application requirements, approaches and technologies. In *SPIE:Intelligent Robots and Computer Vision*, Philadelphia, Sept 1995.

[6] K. Kapellos. *Environnement de programmation des applications robotiques réactives*. PhD thesis, Ecole des Mines de Paris, November 1994.

[7] M. Perrier, V. Rigaud, C. C. de Witt, and R. Bachmayer. Performance oriented robust nonlinear control for subsea robots: Experimental validation. In *IEEE International Conf on Robotics and Automation*, San Diego, May 1994.

[8] V. Rigaud and e.a. Underwater intelligent operation and navigation: Union main objectives of the project. In *Proc of the joint US/Portugal workshop : Intl Prgm Dev in Undersea Robotics and Intelligent Control*, Lisboa, March 1995.

[9] V. Roy and R. de Simone. Auto and autograph. In R. Kurshan, editor, *Workshop on Computer Aided Verification*, New Brunswick, June 1990.

[10] S. A. Schneider, V. W. Chen, and G. Pardo-Castellote. ControlShell: A Real-Time Software Framework. In *Proc of the AIAA/NASA Conf on Intl Robots in Field, Factory, Service and Space*, pages (II)870–7, Houston, March 1994.

[11] D. Simon, B. Espiau, E. Castillo, and K. Kapellos. Computer-aided design of a generic robot controller handling reactivity and real-time control issues. *IEEE Trans on Control Systems Technology*, 1(4):213–229, Dec 1993.

[12] H. H. Wang, R. L. Marks, T. W. McLain, S. D. Fleischer, D. W. Miles, G. A. Sapilewski, S. M. Rock, M. J. Lee, and R. C. Burton. OTTER: a testbed submersible for robotics research. In *ANS 6th Topical Meeting on Robotics and Remote Systems*. American Nuclear Society, February 1995.