

The International Journal of Robotics Research

<http://ijr.sagepub.com>

Handling Sensing Failures in Autonomous Mobile Robots

Robin R. Murphy and Dave Hershberger


The International Journal of Robotics Research 1999; 18; 382

DOI: 10.1177/02783649922066286

The online version of this article can be found at:

<http://ijr.sagepub.com/cgi/content/abstract/18/4/382>

Published by:

 SAGE Publications

<http://www.sagepublications.com>

On behalf of:



Multimedia Archives

Additional services and information for *The International Journal of Robotics Research* can be found at:

Email Alerts: <http://ijr.sagepub.com/cgi/alerts>

Subscriptions: <http://ijr.sagepub.com/subscriptions>

Reprints: <http://www.sagepub.com/journalsReprints.nav>

Permissions: <http://www.sagepub.com/journalsPermissions.nav>

Robin R. Murphy

Computer Science and Engineering
University of South Florida
Tampa, FL 33620-5399, USA
murphy@csee.usf.edu

Dave Hershberger

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
hersh@ri.cmu.edu

Handling Sensing Failures in Autonomous Mobile Robots

Abstract

This article details the SFX-EH architecture for handling sensing failures in autonomous mobile robots. The SFX-EH uses novel extensions to the generate-and-test method to classify failures with only a partial causal model of the sensor/environment/task interactions for the robot. The generate-and-test methodology exploits the ability of the robot as a physically situated agent to actively test assumptions about the state of sensors, condition of the environment, and validity of task constraints. The SFX-EH uses the type of failure to determine the appropriate recovery strategy: reconfiguration of the logical sensor or logical behavior, recalibration of the sensor or actuator, and corrective actions. The system bypasses classification if all hypotheses lead to the same recovery strategy. Results of the SFX-EH running on two robots with different sensor suites and tasks are presented, demonstrating intelligent failure recovery within a modular, portable implementation.

1. Introduction

Autonomous mobile robots, as well as other situated agents, rely on robust sensing to control their actions. If their sensing is faulty, the agent may “hallucinate” and respond inappropriately. Degradations in sensing result from a number of sources, including random sensor errors, systematic sensor errors, and occlusions. Robustness permits the sensing system to either continue execution or allow graceful degradation in the face of unexpected events. Examples of unexpected events include: sensor malfunctions, environmental changes (e.g., the lights go out), and errant expectations (e.g., the robot has a faulty plan). These unexpected events can also be referred to as *sensing failures*, because they can cause the sensing system to misinterpret observations. Environmental changes and

sensor malfunctions are largely exogenous events that are extremely difficult to statistically predict or to explicitly model their impact on sensing.

Robust sensing consists of detecting a potential sensing failure, confirming the failure and identifying its source, and applying the appropriate recovery method. The issue of how to detect sensing failures has received some attention. Certain types of sensing failures can be detected at the behavioral layer (i.e., self-monitoring) (Ferrell 1993; Murphy and Arkin 1992) and/or the deliberative layer (i.e., global monitoring) (Hughes 1993; Noreils and Chatila 1995). These detection methods can be based on either the instantaneous sensor readings or more sophisticated methods such as the time history of signals, conflict with other sensors, and so on.

This article does not address fault detection; instead, it addresses the issue of how to handle sensing failures once they have been detected. Unlike detection, failure handling remains largely uninvestigated, with two notable exceptions. The trend exemplified by Payton and colleagues (1992) and Ferrell (1993) has been to concentrate solely on the recovery from sensing failures, ignoring any issues in classification. These approaches attempt to use the failure as an index into the recovery method, providing an immediate response. An immediate response can be critical for time-dependent robot missions. For example, a robot moving at a high rate of speed (e.g., an intelligent highway vehicle) may not be able to suddenly stop during deliberation. Likewise, the time the robot spends in hostile environments (e.g., Three Mile Island) may also be a concern.

However, in general, detection of a failure does not necessarily mean that the right response can be automatically employed. For example, in Murphy's (1991) work, three different problems that interfered with sensing in a security robot (sensor drift, incorrect placement of the robot, sensor malfunction) evinced that same symptom: a lack of consensus

between the observations. The appropriate response to each problem was significantly different (recalibrate the offending sensor, rotate the robot until it reached the correct view, and replace the damaged sensor with an alternative, respectively). However, the correct response was known once the cause was identified.

The cost of applying an inappropriate recovery strategy can be high. The recovery method may take time to instantiate (e.g., slew time to reposition sensors or initialize the software), plus the time it takes for the detection mechanism to determine that the recovery method has not fixed the faulty sensing. This time could be more than it would take to explicitly categorize the failure. During the time the wrong recovery method was employed, the robot could continue to act on faulty sensor information, exacerbating the potential for a mission failure. Therefore, methods that can quickly classify sensing failures are desirable.

As noted by Murphy and Hershberger (1996), classification of sensing failures in autonomous mobile robots is similar to, but has important differences from, automated diagnosis problems in fields such as medicine and geology. Unlike those domains that support full causal models of problems, it is impractical, if not impossible, to explicitly model all possible failure modes in a mobile robot. Explicit modeling of the interactions between sensors for known environments leads to a combinatorial explosion as seen by Vander, Velde, and Carignan (1984) and Weller, Groen, and Hertzberger (1989). Developing these models assumes *a priori* knowledge about the domain, which may not be available to robots exploring unknown or partially known terrains. Using probabilistic models of sensor failures has been dismissed, (1) because of the difficulty of gathering or generating the large number of necessary conditional probabilities, and (2) because low probability failures may be rejected in favor of higher probability failures, and thus invoke incorrect recovery schemes. While complete causal models are hard to obtain in robotics, partial models may be more accessible. For example, reactive and hybrid deliberative/reactive robots usually have implicit task-specific information rather than complete models of sensor/environment interaction. Constructing even a partial model is challenging. Physically situated agents, such as robots, are subject to environmental changes; the impact of environmental changes on an agent has been shown to be difficult to model (Howe and Cohen 1990). Another important difference between automated diagnosis and robotics stems from the fact that sensing failure handling is not the primary function of the robot. The primary function is to accomplish a task. This leads to a concern over the time dedicated to exception handling, as noted earlier. It also implies that exception handling must be integrated into the whole control system for the robot.

Our approach to classification and recovery of sensing failures has been to extend the generate-and-test methodology developed for medical diagnosis (Lindsay et al. 1980). The

generate-and-test method exploits the ability of the robot to acquire new information by engaging its environment (i.e., active perception) (Bajcsy 1988). Under the generate-and-test method, the robot generates hypotheses and tests about the cause of the sensor failure, orders the tests to eliminate redundancy and confounding effects, collects data, and determines the cause. The appropriate recovery method is linked to the cause.

Our modifications to the generate-and-test methodology allow it to operate with only a partial causal model of all the possible failure modes, sensor relationships, and environmental influences. Much of this model is organized around the sensors: environmental preconditions, sensor tests, and failure hypotheses. One key feature of our approach is that sensors and their diagnostics are unaware of other sensors. This modularity is essential to allow the classification and recovery scheme to be ported to new robot platforms and tasks. Another feature of our approach is that the type of failure serves to partition the search space associated with classification, reducing the time associated with diagnosis. The methodology further reduces execution time by skipping classification if unnecessary, consistent with the "do whatever works" philosophy of Payton and colleagues (1992) and Ferrell (1993). The control scheme also provides for default recovery schemes if the classification process is unsuccessful. These features obviate problems with fine granularity and exhaustive search associated with the generate-and-test method.

This article describes our approach and implementation on two different robots, with both complementary and redundant sensors. Our work has been partially presented in previous papers (Murphy and Hershberger 1996; Chavez and Murphy 1993); this article is intended to provide a complete discussion of the approach and implementation. New demonstrations are offered, with more emphasis on recovery rather than just classification. The article is laid out as follows. Section 2 provides a summary of related work in classifying sensing failures. Section 3 contributes an overview of the theoretical underpinnings of our approach. The implementation as the Exception Handling component of the Sensor Fusion Effects Architecture (SFX-EH) is detailed in Section 4. Section 5 attempts to describe the system in sufficient detail such that it can be replicated on other hybrid deliberative/reactive mobile-robot architectures. Demonstrations of the SFX-EH on two mobile robots, one with complementary sensors and the other with redundant sensors, are presented in Section 5, followed by a discussion of the results in Section 6.

2. Related Work

Our approach is most closely related to the sensing management system of Weller, Groen, and Hertzberger (1989), which uses local expert knowledge and modularity for sensing diagnosis. This provides a foundation for organizing our control scheme around partial causal models. Our control scheme

also incorporates aspects of recovery developed by Ferrell (1993) and Payton and colleagues (1992). The extensions of the generate-and-test strategy make use of general error detection and recovery work from other AI domains, especially planning (Doyle, Atkinson, and Doshi 1986; Hammond 1986; Kolodner 1987; Lee, Barnes, and Hardy 1983; Simmons and Davis 1987).

It is beyond the scope of this article to address the issues of detecting failures, automated generation of hypotheses, and diagnostic tests. We expect that our approach could be used in conjunction with detection methods such as those used by Ferrell (1993), Hughes (1993), Murphy and Arkin (1992), and Noreils and Chatila (1995). Likewise, a means of generating diagnostics from a qualitative model such as that described by Pearce (1988) would be a valuable addition. As with many robot systems, the overall behavior of the SFX-EH scheme is dependent on the often empirical understanding of the knowledge engineer at implementation. While our system mitigates the impact of unclassifiable failures with default recovery schemes (see Section 3.3), a more rigorous method of capturing diagnostics is certainly desirable.

2.1. Sensing Management for Autonomous Mobile Robots

The work by Weller, Groen, and Hertzberger (1989) is one of the strongest influences on our approach. Their efforts deal directly with detecting and recovering from sensor errors. Weller and colleagues' sensor system is broken down into sensor modules. Each module contains tests that verify the input data, the internal data used in computation, and the output data from the execution of the algorithms used to process the raw data. Environmental conditions dictate whether certain tests are to be performed or not. Error recovery is handled by modifying the raw sensor data or the algorithms that manipulate the raw data.

The primary advantages of this approach are that it emphasizes modularity and the use of local expert knowledge. This local expert knowledge is one way of encoding the domain-dependent information needed to streamline the exception-handling process. However, this approach has a significant disadvantage for a sensor-fusion application: it repairs exceptions by either modifying the raw sensor data or the sensor data-processing algorithms. Adjusting invalid data is not an acceptable method for recovery, mainly because there is no obvious method for adjusting the invalid data. Furthermore, the method should be expanded to include modification or removal of the sensors themselves if they are malfunctioning.

Our approach is complementary to the sensing architectures of Chen and Trivedi (1995) and Noreils and Chatila (1995). Noreils and Chatila concentrate on detecting failures by monitoring the motor behavior from the deliberative layer, rather than on general classification and recovery issues. Chen and Trivedi focus on the impact of sensing on planning, especially how to take advantage of new information, leaving the

issues of sensing failures largely unexplored. The SFX-EH could be incorporated into these architectures.

2.2. Recovery from Sensing Failures in Robotics

Gini and Gini (1983) have addressed the issue of error diagnosis and recovery from the perspective of assembly and manufacturing plans. Payton and colleagues (1992) have directly addressed the issue of sensors in fault-tolerant autonomous control of mobile robots, and demonstrated a system that "does whatever works" for an underwater vehicle simulator. The "do whatever works" approach monitors the output of concurrent behaviors, rating their performance. If a behavior is not producing the desired result, it receives a higher failure rating. As the failure rating increases, a redundant behavior is activated. The redundant behavior may use a complementary or a redundant sensor, as well as actuators. The approach is unconcerned with the cause of a particular behavior failure; instead, the focus is on the timely insertion of a behavior that can work. Another advantage of not identifying the failure is that a sensor that has failed for one behavior may work with another behavior, owing to different software, operating range, and so on.

This approach has some shortcomings for our application. It appears that it assumes that redundant behaviors have different sensors and actuators. In our system, many behaviors control the same drive, steering, and sensor-effector actuators. Therefore, it may be necessary to explicitly identify sensor failures (which can be handled by substituting a new sensor) from actuator failures (in which case, no change in sensor will correct the problem). Also, a high degree of redundancy in either the hardware or the software does not ensure that an alternative behavior will necessarily work. Redundant physical sensors may fail if the environmental conditions change; for example, if the lighting conditions are low, swapping to a backup camera will not solve the problem, and valuable time may have been lost slewing the camera into position and then encountering a failure. Ferrell (1993) also notes that a hardware failure that leads to poor performance in one behavior may impact other behaviors which use that hardware. We add that it may not be as easy to detect the impact on performance in the other behavior, leading to a global degradation in robot performance.

While our system is committed to classification, it incorporates aspects of the pragmatic "do whatever works" philosophy from Payton and colleagues (1992) by "short-circuiting" the classification process if all hypotheses lead to the same recovery method. This can be viewed as a "do whatever works, if it doesn't matter what the problem is" variation.

Our SFX-EH system originated with Chavez and Murphy's (1993) demonstration of a generate-and-test methodology for failures in sensor fusion. It was designed for behaviors that used multiple, concurrent sensors. The exception-handling mechanism consisted of two modules: error classification

and error recovery. The error-classification module generated hypotheses about the suspect sensor(s). It then tested these hypotheses with data from the local concurrent sensors. The error-recovery module could then replace the behavior or repair the sensing "plan" (a net of concurrent sensors and their fusion relationship) by pruning the failed sensor.

There are several major differences between the version of SFX-EH reported on in this article, and the older version by Chavez and Murphy (1993). First, the system can be used with behaviors that have only one sensor. This requires a global sensing manager that is aware of the sensing capabilities of the robot outside of the individual behavior. The current system is backwardly compatible with the old SFX-EH. Structurally, the two major representations, the sensor-availability bitmask and the exception-handling knowledge structure, have been significantly expanded and improved.

Ferrell (1993) has recently detailed a fault-tolerant network for Hannibal, a hexapod robot with over 100 physical sensors. Our research was done independently, but shares similar motivation and implementation details. Sensing failures are detected by a consensus monitor, which compares the output of one sensor to equivalent logical or "virtual" sensors. If a sensor reports a reading that is inconsistent with the consensus, that sensor's "pain" or injury level is excited. When the pain level exceeds a threshold, its output is masked, or suppressed.

Recovery from sensing failures takes four forms: retry, recalibration, reconfiguration, and reintegration. Retrying a sensor occurs naturally; since the pain activation does not immediately reach the threshold, the sensor is repeatedly retried until there is confidence that a failure exists. If the sensing anomaly is temporary, it should be smoothed over by the retries. A failed sensor may have a dynamic recalibration routine, which is applied. The effects of masking the failed sensor may also lead to reconfiguration. In this situation, redundant virtual sensors that were already operating assume complete control. This idea is similar to the pruning of a malfunctioning logical sensor from an execution net of concurrent equivalent sensors (Chavez and Murphy 1993). A sensor may be reintegrated if the consensus monitor determines that it is now agreeing with its equivalent sensors. Our system shares with Ferrell's (1993) system the recalibration-, reconfiguration-, and reintegration-recovery modes. The retry mode is assumed to be the responsibility of the detection process, and is outside the scope of this article.

2.3. General Error Recovery

Generate Test and Debug (GTD) (Doyle, Atkinson, and Doshi 1986; Simmons and Davis 1987) and CHEF (Hammond 1986) are planning systems that use similar approaches to error recovery in the geological and cooking domains, respectively. The GTD system uses a generate-and-test procedure supplemented with a debugger to repair denied hypotheses, creating

new ones. One interesting aspect of the GTD system is that it challenges each of the assumptions made in the interpretation. This is useful for sensor-fusion systems such as the SFX, which use a static plan, because of their dependence on assumptions. A static plan is one that is configured at the beginning of a task and is not permitted to change during execution. Perceptual systems that operate according to a static plan tend to be efficient, but will produce errors if the assumptions made during construction are violated.

The CHEF system is a case-based planner that operates in the cooking domain. It generates plans for new situations by altering stored plans that only partially match the present situation. The CHEF contains a plan-modification library that constrains steps that can be substituted for existing steps. This concept is useful because it suggests a way for exception handling to rapidly recover from sensing failures by modifying or repairing the current configuration, rather than by starting over.

A weakness of both the GTD and CHEF systems is that they rely heavily on domain-independent repair schemes and knowledge. This type of knowledge is not readily available in most sensor-fusion domains, where it is difficult to predict complex interactions between sensors. Furthermore, an exception-handling module must be fast to restore sensing as soon as possible. Domain-independent knowledge tends to cause the handler to make many guesses until the cause can be narrowed down and identified. Although it limits extensibility, basing exception handling on domain-dependent knowledge appears to be a more-realistic choice.

Hanks and Firby (1990) have presented a planning architecture that addresses exception handling. This architecture joins Firby's RAP architecture with Hanks's deliberation system. The action component handles the execution of plans and addresses exception handling for plan failures. Two types of plan failures can occur: an atomic action fails, or there are no applicable methods for an RAP corresponding to a plan step. Hanks and Firby indicate that either an alternate method is selected or the same method is run again. A plan step is retried until satisfied, or until the system assures itself that no available method can succeed. The system signals failure if the same method is run twice in the same world state without success. The failure is then passed up to the calling-plan step, which then deals with it.

The contribution of Hanks and Firby's (1990) approach to sensor fusion is to have an exception-handling mechanism propose a number of candidate "next steps," and then select one based on the satisfaction of preconditions for those steps. One major drawback is that no formal error-classification scheme is presented; the system recovers by either choosing another method randomly whose preconditions are currently satisfied, or by running the same method again. Given the need for rapid recovery, it is preferable to modify the current plan, thus saving the startup costs associated with instantiating different sensors.

3. The Approach

This section provides an overview of the two key components of the SFX-EH: classification and recovery. Classification and recovery assumes that an autonomous mobile robot accomplishes a task via independent behaviors. The robot may have one or more equivalent behaviors, or *logical behaviors* (Henderson and Grupen 1990), capable of accomplishing the task. Typically only one member of the set of logical behaviors is instantiated at a time for a task. Each behavior consists of two parts: a motor schema, which defines the pattern of motor activity, and a perceptual schema, which supplies the necessary perception. The perceptual schema component is treated as a *logical sensor* (Henderson and Shilcrat 1984). Multiple logical sensors may exist which are compatible with a particular motor schema for a behavior; the perceptual schema contains the list of logical sensors for that percept.

Classification is based on the generate-and-test paradigm. The basic algorithm has been extended to exploit the advantages of a situated agent that can interact with its environment as needed. The immediate objective of the SFX-EH is to find an appropriate recovery strategy as quickly as possible to continue the robot's mission. Each hypothesis has an associated recovery strategy. A recovery strategy can be directly employed if all of the generated hypotheses have the same recovery method. The recovery strategies fall into these categories: *reconfiguration* of either logical sensors or logical behaviors, *recalibration*, or general *corrective actions*.

3.1. Classification: The Generate-and-Test Method

The basic generate-and-test algorithm is described by Rich and Knight (1991) as the following:

1. generate a possible solution;
2. test to see if this is actually a solution by comparing with the goal; and
3. if the solution is found, quit; otherwise, return to step 1.

The generate-and-test method can be cast as an exhaustive search through all possible hypotheses that can be generated. The amount of search is dependent on the size of the problem space. If the problem space is large, then the generate-and-test strategy can become quite time consuming. Heuristics can be used to increase efficiency, but even then, overall effectiveness still may not be sufficient. A second disadvantage of the original generate-and-test methodology is that it requires a complete causal model. This would require the explicit enumeration of exogenous events, which by definition are difficult to predict. Also, causal models of the interactions between sensors and how their readings should correlate with each other may or may not be readily obtainable.

Despite its weaknesses, especially the potential for being time consuming, the generate-and-test method still offers advantages for sensor-exception handling. First, since it uses

an exhaustive search, it catches errors that occur infrequently. Second, the generate-and-test method allows the robot to actively collect additional information. Because robotic behaviors generally are reactive in the sense of Brooks's (1986) work, their perception is limited to local representations that are focused solely on the motor action. As a result, there is usually not enough information available to a behavior to locally isolate a failure cause. Active acquisition of additional information is critical to the success of error classification. Third, the tests do not require redundant sensors—information from other modalities can be used.

As noted by Murphy and Hershberger (1996), the SFX-EH makes five novel extensions to the generate-and-test method for classifying sensor failures in autonomous mobile robots. First, the problem space is constrained by the symptom (e.g., missing observation, lack of consensus between multiple observations, highly uncertain evidence, etc.) to reduce the search. This reduces the actual time spent in classification. Second, the exception handler generates all possible hypotheses and tests associated with a symptom at one time. Portions of the tests associated with the hypotheses may be redundant, and can be removed at this time. Cycles in the testing requirements can be detected as well. Third, the tests are ordered to ensure correctness. If additional sensors are being used in the tests to corroborate observations or verify the condition of the environment, the sensors must first be tested (if possible) to confirm that they are operational. Otherwise, confounding may occur. Fourth, the list of tests is examined and redundant tests are removed to speed up testing. Fifth and most important, the list of hypothetical causes and tests are generated from partial causal models of the sensors and the task. This model is discussed in more detail below.

3.1.1. Partial Causal Models

The SFX-EH system does not have a single structure that serves as the partial causal model. Instead, the knowledge engineer takes the list of logical behaviors for a task (already available for robot control) and the list of logical and physical sensors (which contain local diagnostic tests) and constructs a hypothesis library. The hypothesis library enumerates possible failure causes, and points to the appropriate sensor (and tests) and to the recovery method. The hypothesis library does go beyond manual encoding of knowledge; it infers when it can use a redundant sensor as collaboration for a suspect sensor. Note that each sensor-data structure contains only knowledge about itself.

Modularity was an important consideration in the design of the SFX-EH. To this end, the partial causal models are local to each sensor. That is, the information about a sensor is independent of information about another sensor. This allows sensors and their associated models to be installed on new robots or systems without incurring significant development time building new simulation models of the system with the

added sensor. Even with this concentration on information local to each sensor, the global approach is not abandoned: global interactions between sensors are inferred by the system based on sensor preconditions.

The hypothesis space searched by the generate-and-test algorithm for sensor-fusion exception handling is constrained to a finite number of built-in candidate-error hypotheses. These hypotheses are constructed at the time the system is integrated with all of its sensors, and are written by the person designing the system. However, as is seen in Section 4, these hypotheses may be either coarse- or fine-grained, reflecting the designer's level of understanding about the problem domain. The ability of one sensor to test a failure hypothesis about another sensor is inferred, similar to the GTD system (Simmons and Davis 1987), where the debugger challenges the preconditions of the sensor for the behavior. Note that in this application, the challenge is part of the initial hypothesis-generation step rather than a debugging step. Examples of preconditions are sufficient ambient light (do both cameras show equivalent lighting?) and adequate power supply (are the ultrasonics underpowered?).

3.2. General Categories of Failures

The SFX-EH system considers three general categories of failures: sensor malfunctions, environmental change, and errant expectations. These failures are described below.

3.2.1. Sensor Malfunctions

Sensor malfunctions manifest themselves as erroneous or missing data. Detection of erroneous data requires domain-specific knowledge in the form of the properties of the individual sensors. The error-classification module relies on the use of diagnostic routines to determine which sensor, if any, has malfunctioned. The sensor-diagnostic routines determine the correct functionality of the sensors. Because the perceptual process is waiting for a response from the diagnostic routines, the diagnostic routines must be responsive. In general, it is more important to have adequate routines that respond rapidly than highly accurate routines that respond slowly.

3.2.2. Environmental Change

Environmental change is an unanticipated change in the environmental state which negatively affects the performance of a set of sensors. Because a sensor operates at maximal efficiency within a limited band of values for each environmental property that impacts that sensor's performance, a change in the value of an environmental property can significantly reduce the performance of specific sensors.

For sensor-fusion exception handling, "relevant" environmental changes affect the performance of a configuration of sensors observing a percept. Therefore, the perceptual system must be able to reconfigure its sensors if an environmental change renders a configuration useless. The primary diffi-

culty is detecting which "relevant" environmental change has occurred.

Environmental change is detected by comparing the current value of an environmental attribute to the value (or range of values) in which the affected sensors perform well. If there is no match, then an environmental change has occurred that has degraded the affected sensors' performance. The environmental attribute values for "good" sensor performance are stored as *environmental preconditions*. Challenges (verification) of the environmental preconditions determine if an environmental change has occurred.

Checking for environmental change may require additional sensing. The additional sensing is performed by spawning off another perceptual process which directs the required additional sensing to determine the current value of an environmental attribute. The spawned perceptual process must involve only a single sensor type, or else the perceptual subsystem has the potential to fall into a disastrous recursive loop, caused by recurring state failures and fusion requests to investigate environmental changes. Because the original error occurred within sensor fusion, it is not logical to utilize additional fusion to diagnose the error.

Determination of applicable environmental attributes requires a significant amount of domain-specific knowledge. Knowledge of the behavior of the specific sensors mounted on the robot and of the environment that the robot will operate in is required. Utilization of domain-specific knowledge increases the complexity in the design of the environmental preconditions, but allows for more efficient execution, because environmental attributes are not checked for values that will never be encountered within the robot's domain (environment of execution).

3.2.3. Errant Expectation

If neither a sensor malfunction nor an environmental change is discovered, then errant expectation, the third cause of a sensor-fusion failure, is assumed. An error in expectation means that the percept that the robot is attempting to locate is not there because: (1) it moved or changed configuration, or (2) the robot is not actually located where it thinks it is.

Recovery from errant expectation is beyond the scope of both the exception-handling mechanism and the perceptual subsystem. Errant expectation is a planning problem, and requires intervention by the planner. In order for the planner to replan intelligently, information must be obtained from the perceptual subsystem to provide assistance. Both the interface between the perceptual system and the planner and recovery from errant-expectation errors are outside the scope of this paper, and are suggested as future work.

3.3. Recovery: Categories of Recovery Strategies

Once the sensing failure is classified, recovery is straightforward, since the logical sensor and behavior scheme explicitly represents equivalences between sensing processes. The

search for an alternative degenerates to a table lookup. If the sensing failure is due to either a malfunction or an environmental change, error recovery attempts to replace the logical sensor with an alternative. The alternative must satisfy any new preconditions discovered by the classification process. For example, if the reason for a sensing failure with a video camera is because the ambient lighting is extremely low, then a logical sensor using a redundant video camera is not considered. If there is no viable alternative logical sensor, the error-recovery process declares a mission failure, and passes control to the planner portion of the robot.

The recovery strategies are: *reconfiguration* by keeping the basic motor behavior but substituting an alternative logical sensor, and reconfiguration by replacing the entire behavior with a new logical behavior; *recalibration* of the sensor or actuator; and *corrective actions*. In the case of reconfiguration, the short-term recovery strategies check to see if the physical sensor for the candidate alternative logical sensor or behavior is operating prior to actual instantiation to prevent new failures. Recalibration is accomplished by routines local to a sensor, similar to that discussed by Ferrell (1993). One example of a recalibration routine is to realign the pan actuator for a camera, which may slip over time or be knocked aside. A novel recovery strategy is to attempt a corrective action. Corrective actions are stored as routines local to a logical sensor. They are particularly useful for situations where the failure classification is ambiguous or an alternative logical sensor or behavior is not available. For example, a camera lens may become streaked with mud, covered by debris, and so on. Reconfiguration may be impossible, because there are no alternatives. Recalibration may not be able to compensate for occluded or distorted images; indeed, there may be no test for this case, even if a hypothesis exists. In these situations, the system may employ a corrective-action routine such as shaking the camera in an attempt to fix the (unknown) problem. If the corrective action works, the robot can continue with the task. If the action does not work, the robot is no worse off than before, because there was no reconfiguration or recalibration option available.

In the case of successive failures, it may be advantageous to retry a logical sensor that has previously failed, as discussed by Ferrell (1993). If the hardware has failed, it may be due to a loose connection, which has fortuitously improved. Likewise, the environment may have become more favorable. Therefore, a previously failed logical sensor may be considered by the recovery strategy. The sensor is tested with the test that it failed; if it passes, it is reinstantiated.

4. Implementation

This section describes the implementation of the SFX-EH. The two robots on which it has been tested are described in Section 5. The SFX-EH software is all written in C++ to

facilitate modularity and portability. However, the mechanisms could have been written in CLOS. Section 4.1 briefly reviews the CSM architecture. The SFX-EH specifics are then given in Section 5.1, consisting of an overview of the exception-handling mechanism followed by detailed descriptions of important exception-handling components.

4.1. Architecture Overview

The CSM architecture is a hybrid deliberative/reactive architecture focusing on the integration of sensing with action. It is called the *sensor-fusion effects* architecture (SFX). Figure 1 shows a simplified version of it that was used for this work. Information about the complete architecture is presented in an earlier work (Murphy and Mali, forthcoming). The *deliberative* layer handles all activities that require knowledge about the robot's task. It consists of two main components: the *task manager* and the *sensing manager*. The task manager runs and manages the behaviors. The sensing manager allocates and maintains sensing resources, and is the focus of this research.

The *reactive*, or behavioral, layer is responsible for executing the behaviors according to the plan. Each behavior is divided into two schemas: a *perceptual schema* (PS) and a *motor schema* (MS). The motor schema controls actuators (i.e., drive and steering motors) based on information about the world provided by the perceptual schema. Each perceptual schema is responsible for providing this information reliably to its corresponding motor schema. Toward this end, each PS has a list of logical sensors that it can choose from. This list is called the *percept model*. The logical sensor represents one or more real sensors coupled with a processing algorithm. The algorithm can process the sensor data into a form common to the logical sensors in the percept model, giving them equiv-

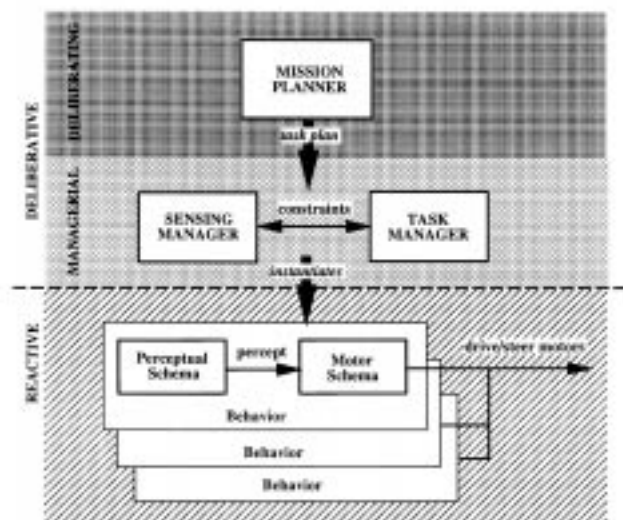


Fig. 1. Overview of the CSM SFX architecture.

alent interfaces to the perceptual schema, and thus yielding interchangeability.

4.2. Exception-Handling Control Schema

Robust sensing in the reactive layer is made possible by the exception-handling control schema implemented in the sensing manager. The basic process of handling exceptions is shown in Figure 2. When a perceptual schema or any of its subcomponents detects an *exception*, or a potential sensing failure, it sends information about it in the form of an *exception-handling knowledge structure* (EHKS) to the *error handler* in the sensing manager. If the exception can be classified, the type of the error is sent to the *error-recovery* unit. The error-recovery unit either requests the task manager to instantiate different behaviors (if the current behaviors are no longer viable), or it passes information about which sensor is unusable back to the perceptual schema. In the second case, the perceptual schema consults its list of possible sensor configurations and requests a new one from the *sensor-allocation manager*. Finally, if none of the requested sensors are available and usable, the perceptual schema sends a *behavior fault* to the task manager. In the case where the error handler cannot classify the exception, an *errant expectation* is sent to the task manager, signifying that the perceptual system appears to be working and a plan failure should be considered.

4.3. The Exception-Handling Knowledge Structure

The exception-handling knowledge structure (EHKS) merits further discussion because it plays a key role in diagnosing failures: keeping track of where faults are detected. Figure 3 shows the three fields of the EHKS and what they contain in two different situations. In Figure 3a, a problem has been

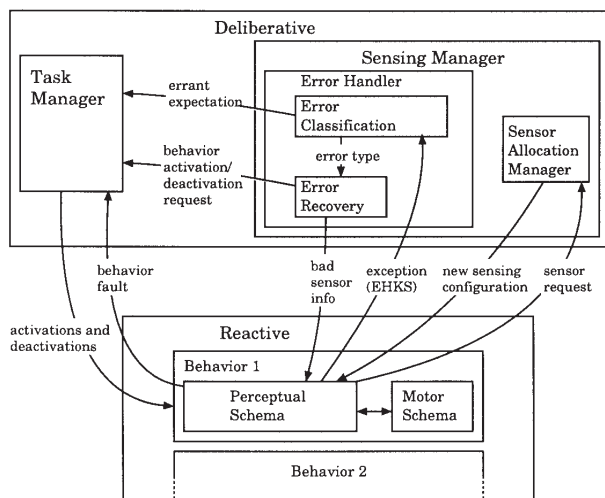


Fig. 2. The basic exception-handling process.

detected in the execution code for *perceptual schema 1*. This implicates all the sensors that were providing input at the time of the fault, as shown by the gray area. For example, if sonar and visual input are both being used by a PS, and they begin to show an inconsistency with each other, either one may be at fault. In Figure 3b, a problem has been detected in *sensor 1*. In this case, only sensor 1 is implicated. The system has no knowledge about which perceptual schema is associated with the fault, nor is such knowledge necessary. For example, if a camera stops sending a valid video signal, it does not matter to the diagnostic system how the video was being used. Creation of an EHKS occurs at each place in the system where faults are detected. For example, when a loss of video signal is detected in the camera-sensor object's code, an EHKS is created with only a pointer to the camera-sensor object. The EHKS is then thrown to the sensing manager, and error handling is initiated.

Once the sensing manager's error-handling function is called with an EHKS, it must first generate a list of sensors that are suspected to have caused the problem. If the failure was detected by the sensor object (a bad video signal, for example), the sensor field of the EHKS will have a pointer to the only sensor that is suspect—the one that detected the failure. If the failure happened in code unique to the logical sensor, it will create an EHKS with a pointer to the logical sensor. In this case, the suspect sensor list becomes a list of all sensors used by the logical sensor that detected the problem. Similarly, if the perceptual schema signaled the fault, all of the sensors from all of the logical sensors being used by the PS will be in the list.

4.4. Hypotheses

The list of suspect sensors gives the system a way of pruning the search space of possible causes of the problem (or hypotheses). At system startup, a global list of hypotheses (the hypothesis library) is created. Most hypotheses are manually coded at the time of adapting the software to a new platform, and refer to specific sensor and test objects, but some are more generic. Later, during error handling, a list of candidate hypotheses is generated from the hypothesis library using the suspect sensor list. These hypotheses are then tested by running tests of the physical sensors. The subsequent recovery from the error is described in Section 4.5.

4.4.1. The Hypothesis Library

The hypothesis library stores hypotheses about failure modes of sensors. To avoid finding and storing information about all possible failure modes, hypotheses are only programmed for failure modes that it is useful to be able to distinguish between. For example, there are many things that could cause the video signal from a camera to be interrupted, but distinguishing between them is unnecessary, because the only way to recover from any of them is to give up on that camera and try to use another sensor.

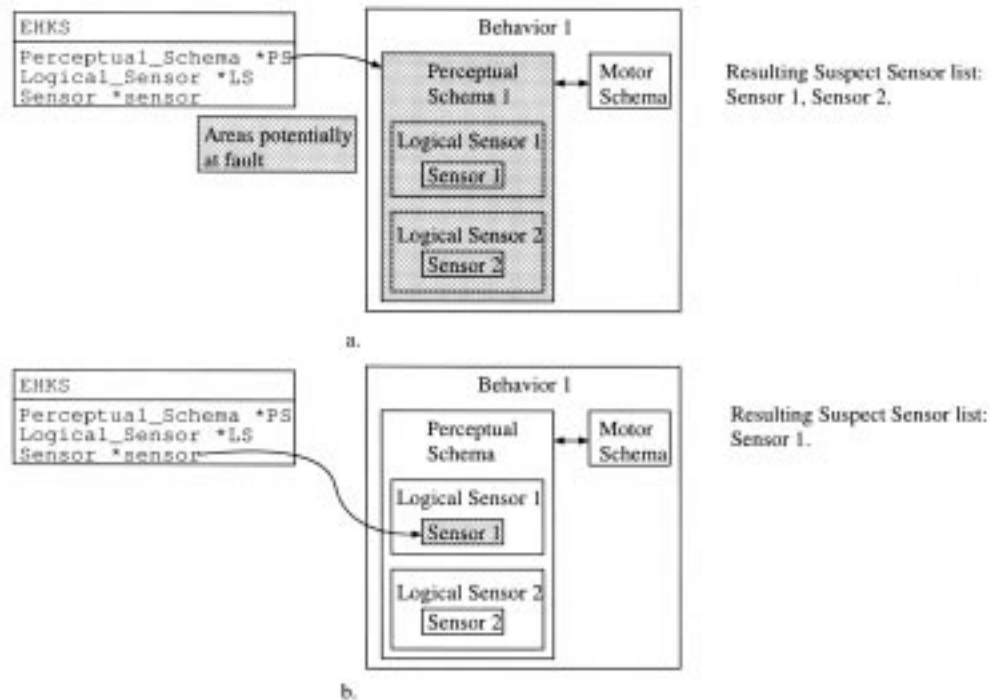


Fig. 3. The EHKS connections in two different situations: when a fault was detected in the perceptual schema (a); and when a fault was detected in the sensor code itself (b).

The hypothesis library is represented as a list of hypotheses. There are two types of hypotheses: regular *hypotheses*, and *environmental change hypotheses*. Each regular hypothesis contains an evaluation function, pointers to sensors, tests, and a recovery method. Each hypothesis also provides a list of sensors that would become unusable if it were true (its *affected sensor* list). For regular hypotheses, this usually contains the one sensor that the hypothesis is concerned with. For environmental change hypotheses, this list usually contains all the sensors of a given type, i.e., “vision.” Tests are used by the evaluation functions to determine the truth or falsehood of the hypothesis. Environmental change hypotheses are listed first in the hypothesis library to avoid being masked by other simpler hypotheses.

4.4.2. Hypothesis Generation and Testing

Because all hypotheses for a given system are explicitly listed at startup, the hypothesis-generation portion of the error-handling function need only prune the list down to hypotheses that could possibly be true, given the error. This is done by listing all hypotheses with affected sensors in the suspect sensor list generated from the EHKS.

Testing each possible hypothesis is the next task for the error handler. The ordering of the hypothesis library simplifies this to finding the first hypothesis that evaluates to true. Each evaluation function computes a logical combination of

test results, such as “test 1 passed and test 2 failed,” which implies the truth of the hypothesis in question. Only those tests that are needed to find a true hypothesis are run, and each of these is run only once for a given sensing-failure episode. In the case where a hypothesis runs tests on multiple sensors, such as an environmental change hypothesis, the *corroborating* sensors are first validated by running a hardware test for each. If no hypotheses evaluate to true, the sensing manager sends an errant expectation to the task manager.

This whole process of testing hypotheses can be short-circuited if the recovery methods of all possible hypotheses are the same. This is demonstrated in a simple way on Clementine 2 when there is a problem with the sonar. Because there is only one hypothesis about the sonar, *Sonar_Broken_Hyp*, there is no need to test it. The recovery method, which switches out the sonar-based behavior and switches in the vision-based behavior, is run immediately.

4.5. Error Recovery

If the hypothesis-testing code finds a true hypothesis, or if the testing is short-circuited, the sensing manager attempts to recover from the sensing failure. Recovery has two stages. The first stage is to mark all the sensors in the hypothesis’s affected-sensor list as “bad.” This prevents these sensors from being used again by any behavior without first getting retested. The second part is to run the recovery method stored

with the hypothesis. In some cases, this second step will be empty, since the sensor getting marked “bad” will cause the PS to try to allocate another one. In cases where a perceptual schema has only one possible input sensor, which is now marked “bad,” it may ask the task manager to deactivate this behavior and possibly activate another (i.e., reconfiguration). In other cases, such as reconfiguration or corrective action, the recovery method may be able to fix the problem with the sensor (by shaking a camera to remove obscuring debris, for example). At that point, the recovery method must reset the status of the sensor to “good.”

4.5.1. Sensor Allocation

If the failed sensor is not actually fixed by the recovery method, there is one final step in handling a sensing failure: allocating a new sensor. When a sensor in use is marked “bad” and the perceptual schema using it remains active, the PS must attempt to allocate another sensor to replace it. The process of allocating a sensor starts when a PS tries to use a “bad” sensor. The PS makes a list of all the sensors (good or bad) that it can use, with the most-preferred sensors listed first. The list is passed to the sensing manager’s *Request_Sensor()* function. Calling this function has three possible outcomes:

- A “good” sensor is available, and is allocated and returned.
- No “good” sensors are available, but a “bad” sensor has started working again, which is detected by re-running whichever test first detected the failure. The sensor is marked “good,” allocated, and returned. In this case, the entire “good” list is first checked, and only after no good sensors are found are the bad sensors tested. Furthermore, bad sensors are not retested more than once during a given period of time. This prevents “thrashing”—a state in which the system rapidly switches back and forth between multiple failing sensors. Future versions of the SFX-EH will include a frequency-of-failure measurement that can be used to further reduce the potential for thrashing.
- No sensors are available or none pass their tests. In this case, a *behavior fault* is generated, since the behavior requesting one of these sensors presumably cannot run without any sensors.

The allocation mechanism implemented to date is a simple one, in which a sensor can only be allocated to one PS at a time, and the first PS to claim a given sensor has exclusive rights to it until that PS voluntarily gives it up. There are many possible improvements to this scheme, some of which are discussed in Section 6.

The sensor-allocation function uses a simple data structure to keep track of sensor allocations, called the sensor allocation table. This table stores a mapping from perceptual schemas to sensors. A list of perceptual schemas is kept, and with

each entry a list of allocated sensors is stored. An additional list of sensors keeps track of which sensors are not allocated to anything and are thus available for use. When a PS is deactivated, it relinquishes all of its allocated sensors.

When this process of handling a sensing failure is complete, the robot is using a repaired sensor or a replacement sensor, a behavior has been changed, or the program has halted because no sensors remain to perform the task. In any of these cases, the failure has been handled as gracefully as possible.

5. Demonstrations

The SFX-EH system was demonstrated on two mobile-robot platforms. These were Clementine 2 (C2), a Power Wheels toy jeep with a 100-MHz Pentium PC attached (Fig. 4a), and Clementine, a Denning/Branch MRV-4 powered by a 66-MHz 486 PC (Fig. 4b). Clementine 2 has a panning color camera mounted in the center of the vehicle, and a panning sonar in the front. Clementine has two color cameras mounted facing front and back (180° apart). The demonstrations showed the SFX-EH quickly classifying and recovering from errors on robots with complementary and redundant sensors.

5.1. The C2 Demonstration

The implementation on Clementine 2 demonstrated complementary sensors being used to insure robust sensing. It also served to exercise the library, to show some more-advanced recovery methods, and to show the portability of the system to new platforms. This section first describes the code specific to the implementation on C2, then presents the results of the demonstration.

5.1.1. Demonstration-Specific Code for C2

The test domain for the C2 demonstration was hall-following. There were two behaviors: a visual hall-following behavior and a sonar-based hall-following behavior, each of which had one available sensor (see Fig. 5). Only one of these behaviors is needed at a time, and either should suffice for the majority of the hallway used. Therefore, if one sensor should fail, the robot needs another way to complete its task, so a recovery method tells the task manager to suspend the current behavior and instantiate the other one to complete its task.

The visual hall-following behavior uses a camcorder mounted on a panning mast to see the baseboards of the hallway walls, which are dark on a light background. The sonar-based hall-following behavior uses a sonar mounted on a panning motor attached to the front bumper of the robot.

The tests, hypotheses, and recovery methods used in the demonstration on C2 are shown in Figure 6. The lines between the tests and the hypotheses show how the *evaluate()* functions of the hypotheses work. For this demonstration, all of the

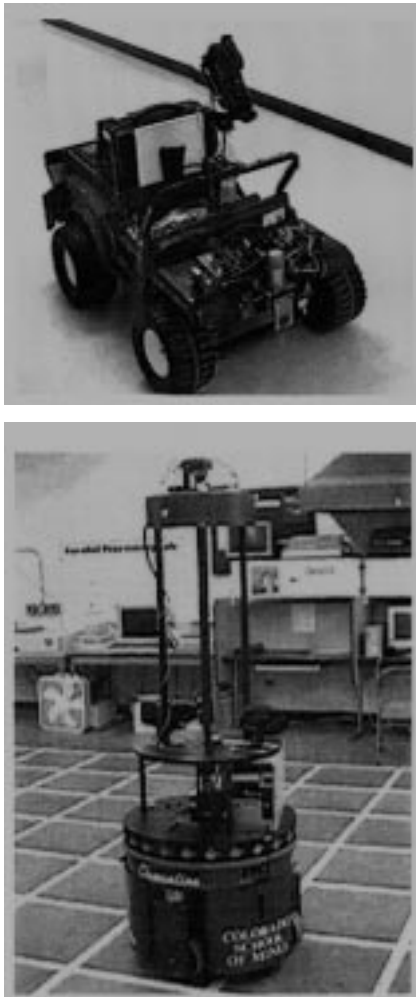


Fig. 4. Clementine 2 (C2), a Power Wheels jeep with one camera and one sonar (a); and Clementine, a Denning/Branch MRV-4 with two cameras (b).

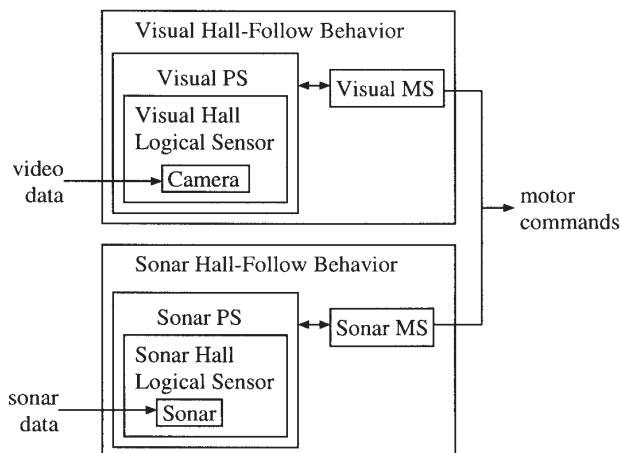


Fig. 5. Tests, hypotheses, and recovery methods for Clementine 2 (actuator tests are not shown).

evaluation functions were conjunctions of their corresponding test results.

The first hypothesis shown is the *sonar broken* hypothesis, which is classified as a sensor malfunction. Since it is the only hypothesis related to the sonar, it is the only one generated when a sonar problem is detected. Therefore, the testing process is short-circuited, and the recovery method can be run immediately. In this case, the recovery consists of switching to the visual hall-following behavior. The sonar test is only run when the sonar sensor is marked "bad" from a previous problem but is needed because of a problem with the visual behavior. The mechanism for retesting a sensor that is marked "bad" is described in Section 4.5.1.

The *camera broken* hypothesis is similar to the sonar broken hypothesis, and is also a sensor malfunction, but test short-circuiting cannot happen because when the camera is the suspect sensor, all five vision-based hypotheses are generated and there are three different possible recovery methods.

The *lights out* hypothesis is of particular interest, because it is an environmental change hypothesis. As explained in Section 4.4.1, an environmental change hypothesis has no specific sensors to test; instead, it tests all the sensors of a given type. In this case, it tests all visual sensors with the brightness test.

The *camera lens obscured* hypothesis (a sensor malfunction) is associated with a more complicated test scenario. For its evaluation to succeed, the camera test must pass, the brightness test must pass, and the toes-visible test must fail. The toes-visible test works by looking for the drawing of a foot which is attached to the back of the computer. The *shake camera* recovery method then shakes the camera back and forth via the panning motor in an attempt to dislodge whatever is blocking the lens. For the demonstration, a cloth was draped over the camera, and the shaking typically cleared the lens after the first or second try.

The *camera misaligned* hypothesis (another sensor malfunction) depends on the camera working and the toes-visible test passing, but the camera-alignment test failing. (The camera-alignment test works by comparing the angle turned to see the foot with 180° .)

In the case where the lines of the baseboards are no longer visible or are not there at all, the robot cannot use vision to navigate. This situation is presumed by the *lines gone* hypothesis when the toes are visible, the camera is properly aligned, and the lines are still not visible. This is classified as an environmental change hypothesis. In this case, the recovery method is the same as when the camera is broken or the lights are out. In all of these situations, vision is inappropriate, and the robot falls back to the sonar-based behavior.

In addition to the above hypotheses, there is also a pair of actuator-failure hypotheses: *sonar-pan failure* and *camera-pan failure*. Each has a corresponding test that commands a simple pan motion and measures the response. These tests and hypotheses are difficult to demonstrate on the robot, because

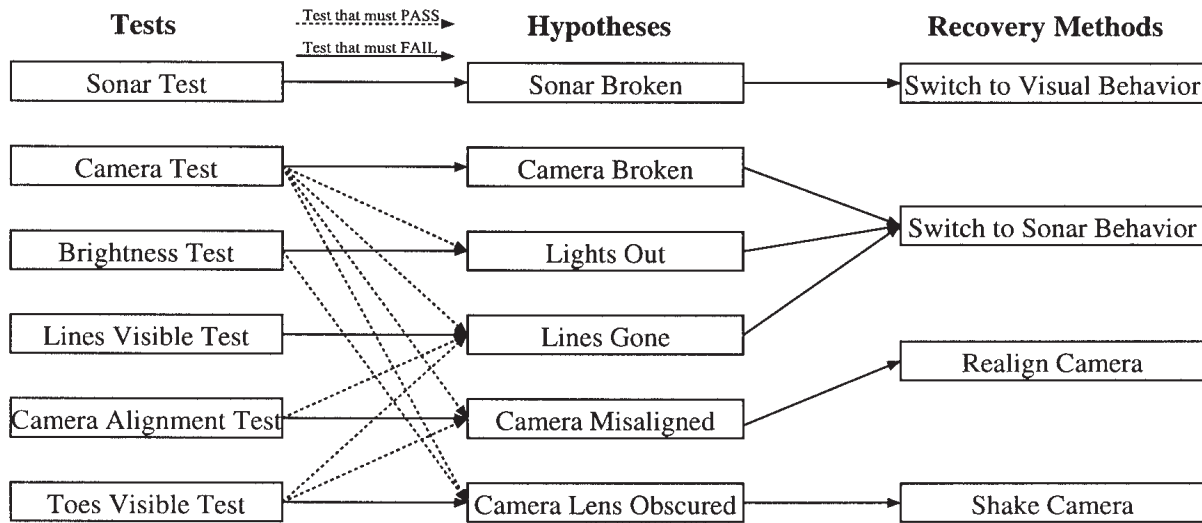


Fig. 6. Tests, hypotheses, and recovery methods for C2.

they are triggered by real hardware problems internal to the panning-motor controllers. Because of this and because they do not serve to demonstrate any otherwise novel feature of the system, they are not shown in the diagrams or mentioned elsewhere.

5.1.2. C2 Demonstration Results

This section shows the C2 robot being exposed to various failures and then recovering as it travels down the hall. The process of each recovery is explained in detail.

Camera Misalignment

The first failure, shown in Figure 7, shows the camera mount being misaligned and the robot recovering. This could happen in applications where there are low overhead obstacles that bump the camera, such as tree or shrub limbs, or the ceiling of a crawlspace in a rubble pile. For the demonstration it was knocked to the side by hand (Fig. 7b). If this happens while the visual behavior is operating, as in Figure 7, the visual hall logical sensor stops detecting any lines, and throws an exception. The sensing manager's *Handle_Error()* function is called, and the process of handling the error begins.

Figure 8 shows details of the process of handling the error. The box at the far left shows that the list of candidate hypotheses is generated from the suspect sensor *camera*, as described in general in Section 4.4.2. The table in the figure shows the incremental execution of the tests as each hypothesis is evaluated. Tests are only shown in the table in the step where they are first run because subsequent hypothesis evaluations use their results without rerunning them. The box at the far right shows that the recovery method for the *camera misaligned* hypothesis consists only of realigning the camera.

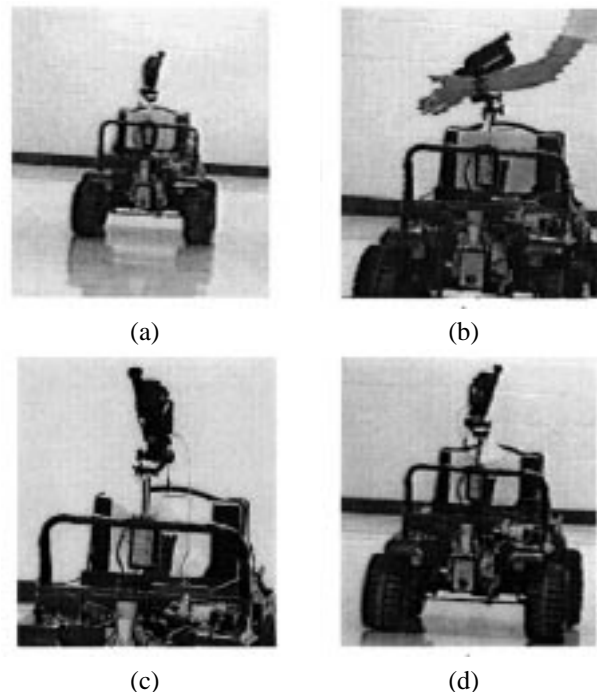


Fig. 7. Camera misalignment and recovery: normal camera operation (a); camera knocked sideways (b); toes-visible test passing (c); and a return to normal camera operation (d).

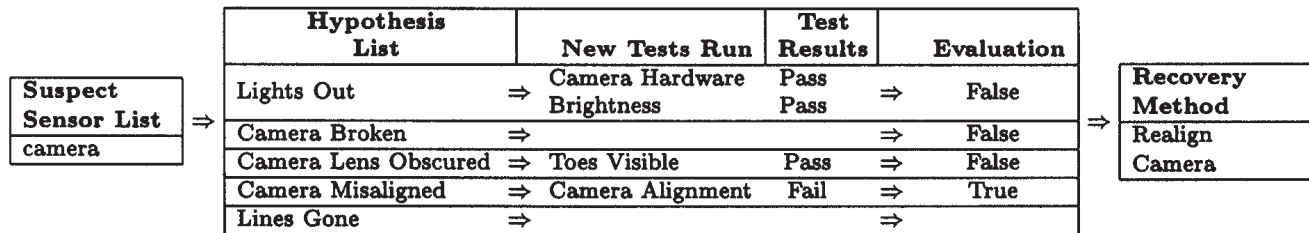


Fig. 8. Handling a camera misalignment: candidate hypothesis list generation, hypothesis evaluation, and recovery.

Lost Video Signal

Figure 9 shows what happens when the robot loses its video signal while using the visual follow-hall behavior. This is a common problem on C2 and other robots, typically caused by video cables working loose or camera batteries running low, but also potentially caused by destruction of the camera itself. For the demonstration, the video cable was simply unplugged from the camera.

Figure 10 shows the process of recovering from the loss of the video signal. Notice that only one test was needed to find a valid hypothesis, because of the incremental evaluation of the test results.

Sonar Failure

Figure 11 shows the results of the sonar failing to detect return signals when using the sonar follow-hall behavior. This failure can happen when the robot moves into an area with cloth-covered office partitions or another sound-absorbing material, or when it moves out of a hallway and into a large open area. In this case, the sonar transducer was simply covered with a cloth.

Figure 12 shows how the system handles the sonar failure. Because there is only one candidate hypothesis, there is only one possible recovery method, and the testing pro-

cess is short-circuited. When the recovery method consists of switching to a different behavior, the new behavior is responsible for making sure its sensor is operational. In the case shown, the camera was in working condition, but it was still labeled "bad" from its previous failure. The perceptual schema within the visual hall-follow behavior detected that its current sensor (the camera) was labeled "bad," and called Request_Sensor() in the sensing manager. As there were no replacement sensors for the visual hall-follow behavior, Request_Sensor() reran the test, which caused the camera's "bad" status (Camera_HW_Test). In this case, the test passed, because the video cable had been plugged back in; so the visual behavior was able to continue. If the problem had not been fixed, there would have been no behaviors with working sensors available, and a behavior fault would have been signaled to the task manager.

Camera Lens Obscured

Figure 13 shows how the robot reacts when its camera lens becomes obscured. This could happen from dust, sand, or mud getting on the lens, or even a piece of equipment or garbage falling on the camera. For the demonstration, a shirt was dropped over the camera.

The process of recovery is shown in Figure 14. The shirt did not block enough light for the camera-brightness test to fail, and instead the toes-visible test failed, confirming the camera-obscured hypothesis. The shake-camera recovery method works by driving the camera panning motor back and forth through 250° of rotation twice per second. It does this for six shakes, then pans the camera looking for the image of the toes. If it does not find them, it shakes the camera again, giving up after four tries. If it does find them, it realigns the camera, labels it "good," and returns.

5.2. Clementine Demonstration

The SFX-EH software was also implemented on Clementine, a Denning/Branch MRV-4 mobile robot, to show how redundant sensors are handled. For the demonstration, there was one behavior: track-target, with two identical color video

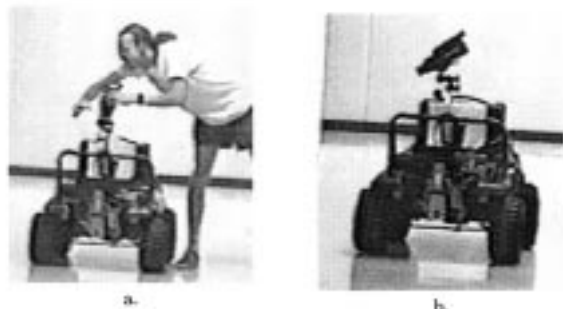


Fig. 9. Camera-hardware failure and recovery: the camera's video cable is unplugged (a); sonar behavior takes over (b). (The camera turns to the side to indicate it is not in use.)

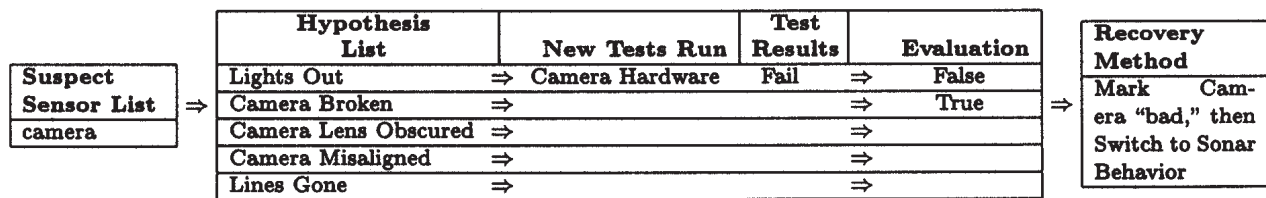


Fig. 10. Handling a lost video signal: candidate-hypothesis list generation, hypothesis evaluation, and recovery.



Fig. 11. Sonar failure and recovery: normal sonar operation (a); cloth absorbs many sonar pulses (b); and camera behavior is activated (c).

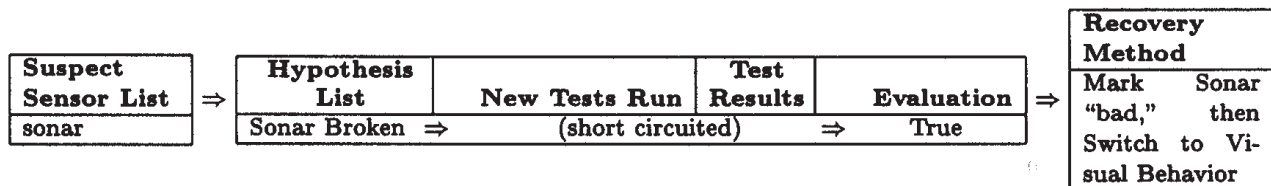


Fig. 12. Handling a sonar failure: candidate-hypothesis list generation, hypothesis evaluation, and recovery.



Fig. 13. Camera cover-up and recovery: normal camera operation (a); shirt is placed over the camera (b); toes-visible test fails (c); shake-camera recovery method runs (d); and camera lens has been uncovered, so robot resumes following the hall (e).

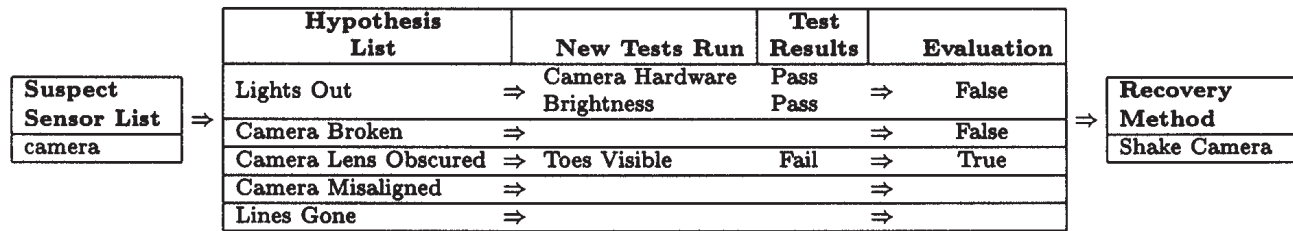


Fig. 14. Recovery when the camera lens is obscured: candidate-hypothesis list generation, hypothesis evaluation, and recovery.

cameras mounted on the robot. One faced forward and the other backward, as shown in Figure 4b. Because Clementine can drive either forward or backward, and because the head with the cameras turns with the wheels, she can use either camera as a forward-facing camera. The track-target behavior had a percept model with two logical sensors, one for each real camera. At any given time it would use one of these to track a square purple target and drive toward it. When the camera in use was damaged or somehow obscured, it switched to the remaining camera by turning 180° and reversing the drive-motor direction.

5.2.1. Clementine Demonstration-Specific Code

The hypotheses, tests, and recovery methods used for the demonstration on Clementine are shown in Figure 15. The first hypothesis is an environmental change hypothesis: *lights out*. For it to be true requires that both cameras fail their tests. In this case, it is important that it be evaluated before the other hypotheses, because both of the other hypotheses use subsets of its test results. There are not separate video signal tests and brightness tests in this implementation, because of a limitation in the video hardware. Instead, our camera tests simply check the average pixel intensity of the images, and serve both purposes. The recovery method for the lights out hypothesis must throw a behavior fault to the task manager, as there are no other sensors for the track-target behavior to use.

The two *camera broken* hypotheses (sensor malfunctions) are nearly identical to their counterpart on the C2 system. Each has one test for input, and each has the same recovery method: switch to the other sensor. In this arrangement however, the recovery does not need to activate and deactivate behaviors. Since the `Handle_Error()` function marks the affected sensor "bad," and the PS will notice that a sensor in use is bad and request a new one, the recovery method does not actually need to do anything.

5.2.2. Clementine Demonstration Results

This section shows the robot Clementine undergoing failures and recovering as it moves toward its target. The process of recovering from each failure is described in detail.

Single Camera Failure

Figure 16 shows what happens when Clementine's active camera is covered with a box while tracking its target. The recovery process is shown in Figure 17.

Target not in View

Figure 18 shows the robot responding to a situation where the target has moved out of view. This failure corresponds to any failure where the low-level behavior is not programmed to handle the situation at hand. The track-target behavior on Clementine is only programmed to handle situations where the target is in view, so an error is detected.

The testing and recovery process is shown in Figure 19. This time, both camera tests pass, but since this situation does not correspond to any known hypotheses, an errant expectation is sent to the task manager. The task manager in this case holds knowledge indicating that errant expectations from the track-target behavior should trigger a small pan of the camera, exposing it to a different field of view. This is repeated until the track-target behavior finds the target.

Dual Camera Failure

Sometimes both cameras will fail simultaneously, or one will fail while the other is inoperative. Possible causes of this type of failure are environmental changes, such as lights being turned out or the sun going down, or multiple hardware failures. This was demonstrated on Clementine by unplugging both cameras' video cables.

Figure 20 shows the process of diagnosing a dual camera failure. The lights-out hypothesis is found to be true. The true cause of the failure, simultaneous hardware failures, was not modeled, but because the recovery method would have been the same, it was not necessary. The recovery in this case was to send a behavior fault to the task manager, because the visual target-tracking behavior cannot work without video input. The task manager then stopped the robot and terminated the program because the target-tracking behavior was the only one available. In a more-involved application, the task manager might select a different behavior to accomplish the same goal, or select a different goal to pursue.

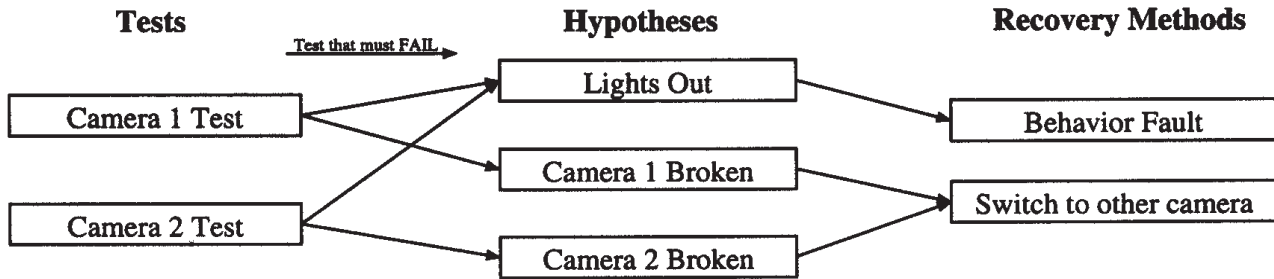


Fig. 15. Tests, hypotheses, and recovery methods for Clementine.

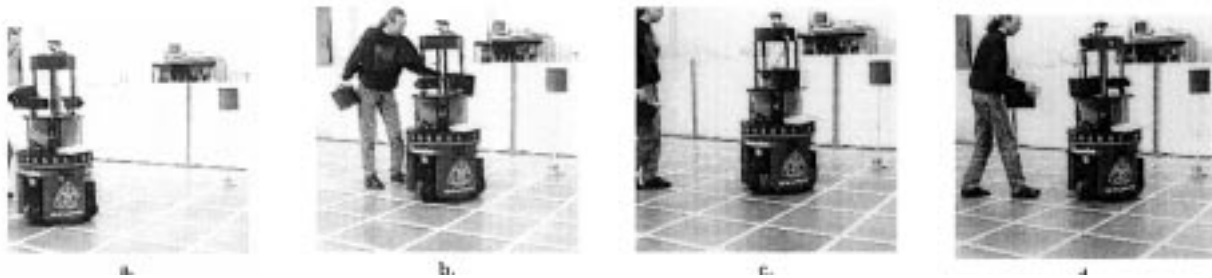


Fig. 16. Camera 1 failure and recovery: normal camera 1 operation (a); camera 1 covered with a box (b); switching to camera 2 (c); and resumption of tracking with camera 2 (d).

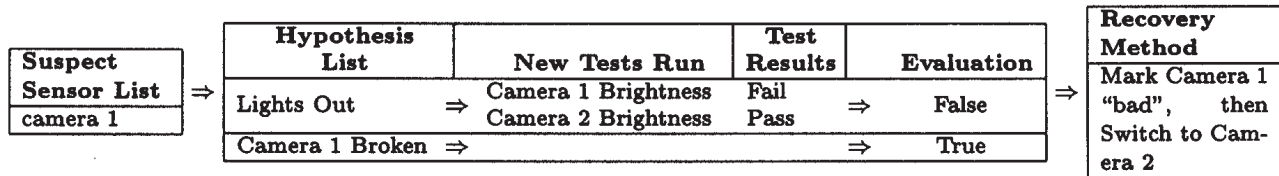


Fig. 17. Handling a single camera failure: candidate-hypothesis list generation, hypothesis evaluation, and recovery.



Fig. 18. Recovery when the target is not in view: initially, camera 1 cannot see the target (a); the robot turns until the target is seen (b); and the robot proceeds toward the target (c). The target is in the middle of the right edge of each photo.

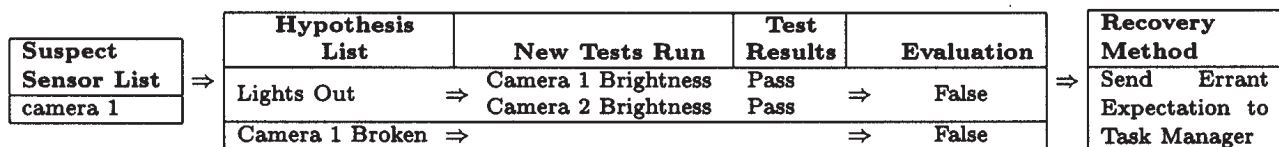


Fig. 19. Recovering when the target is not in view: candidate-hypothesis list generation, hypothesis evaluation, and recovery.

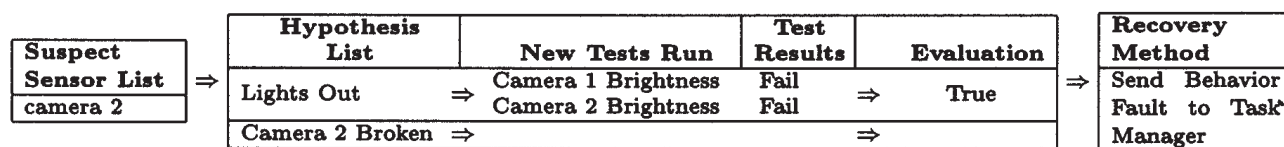


Fig. 20. Handling a dual camera failure: candidate-hypothesis list generation, hypothesis evaluation, and recovery.

6. Conclusions

This article has demonstrated how the SFX-EH architecture can classify and recover from sensing failures. The demonstrations highlight the advantages and limitations of the modified generate-and-test methodology, and of the current SFX-EH implementation.

The extended generate-and-test approach has several advantages. It requires only a partial causal model of the sensors, task, and environment. This makes the technique useful for autonomous mobile robots that are operating in partially known or unknown terrains, where it is impossible to extrapolate all possible exogenous events. It also allows the domain knowledge to be partitioned into more maintainable and portable modules. The knowledge of hypotheses, tests, recalibration routines, and corrective actions specific to a sensor can be stored with that sensor. Sensors are not aware of the presence of other sensors; relationships such as the ability of one sensor to corroborate the other are inferred by the SFX-EH from common preconditions. The SFX-EH was demonstrated on robots with complementary and redundant sensors, reinforcing the generalizability of this technique.

A practical advantage of our implementation of the generate-and-test method is that it allows the hypotheses to be at several levels of granularity. A hardware problem with a physical sensor (e.g., framegrabber not responding) or an unspecified environmental event (e.g., lighting is not sufficient for cameras) can be identified. In the latter case, it is important to note that the robot has no notion that the lights have been turned off, only that something in the environment has changed which impacts all sensors using visible light above a certain threshold. This information could be used by a higher cognitive agent in the robot to reason about the contravening state of the world, but that is a separate, parallel activity from recovery at the reactive level.

Another advantage of the SFX-EH is that the classification process can be short-circuited, similar to that described by Payton and colleagues (1992), reducing the time spent on problem solving. It should be noted that in our experience, the actual execution of the tests and recovery strategy took up almost all of the time associated with the sensing failure. This suggests that the system could afford to spend more time in the hypothesis and test-generation phase to reduce the number

(or resources) of the tests. This is an area we are currently investigating.

The SFX-EH supports a broad range of recovery strategies: reconfiguration, recalibration, and corrective action. The demonstrations in this article were by no means complete; for example, we expect that more sophisticated recalibration routines and corrective actions could be designed for other tasks. It is worth noting that recalibration and corrective actions are critical recovery methods for robots with a small number of general-purpose sensors. These types of robots may not have much physical redundancy, and so re-establishing sensing quality, rather than just ignoring it, is important. There is no guarantee that a sensor will recover by itself, and repeatedly retrying a sensor may just waste valuable time and resources on a computationally bound robot such as a planetary rover. There is also no guarantee that a sensor will have a recalibration routine for the specific failure. Therefore, general corrective actions (such as shaking the camera) are useful for attempting to re-establish adequate sensing, and also serve as default recovery methods when the results of hypothesis generation and testing are ambiguous.

Although not discussed in this article, the SFX-EH control scheme is suitable for logical sensors that fuse observations from multiple sensors (sensor fusion). The original version of SFX-EH (Chavez and Murphy 1993) was developed for sensor fusion, where there were several concurrent sensors for the same behavior. This simplified error detection allowed one sensor to guide the recalibration of another sensor, and the recovery strategy was essentially eliminating the faulty sensor from the fusion mix. The current version of SFX-EH is backwardly compatible. However, since single sensor/behavior systems are more common in practice, we have concentrated on them in this article. Another expected advantage of the SFX-EH approach is that it can be extensible to other situated agents, such as intelligent process controllers in manufacturing. We are currently transferring this research to process controllers for the power industry, where remote sensors are used to monitor power transmission and consumption.

The SFX-EH will encounter situations where it cannot resolve the sensing failure, either because it cannot classify the problem and/or no recovery strategy exists. This requires the advent of a higher cognitive agent, with more general problem skills and the ability to redefine the robot's mission,

possibly preempting tasks that use the failed behaviors and logical sensors. At this time, SFX-EH does not have such a higher cognitive agent; instead, the robot just halts and waits for the user. The role of a human supervisor in those situations where the exception-handling process fails "upward" has been investigated (Murphy and Rogers 1996; Rogers, Murphy, and Ericson 1997). The approach in those efforts is to treat the robot as a physically situated agent and the human as the cognitive agent. A third computational agent, the intelligent assistant, is added to facilitate visual problem solving, hypothesis management, and recovery for the human.

The SFX-EH also has significant limitations. First, it does depend in part on the designer's domain knowledge. It is more general than Ferrell's (1993) or Payton's (Payton et al. 1992) systems, but it is not domain-independent. Another limitation of the SFX-EH system is that it assumes that a sensing failure has only one cause. Multiple events could transpire to produce a single symptom. The SFX-EH would recover based on the first cause it could identify, and then the recovery method might fail due to the other causes. The sensing-allocation table would keep track of the identified failures, but there is currently no mechanism to combine previous failures with current ones to imply new hypotheses. Finally, the SFX-EH system assumes that all sensors are available for use in testing. In a robot performing concurrent behaviors, this may not be true, and some mechanism is needed to coordinate sensing allocation.

We are researching many improvements to the SFX-EH system. One thrust is concentrating on reducing the time spent in classification. One approach is to exploit any available knowledge about the frequency of failures. The generate-and-test algorithm currently used does not rank the hypotheses by the probability of occurrence. This is both a strength and weakness: it is advantageous in that the search generates and considers all hypotheses, allowing extremely infrequent problems to be identified. However, in a system where the time available for problem solving is limited, the generate-and-test method may introduce unnecessary delays that could be prevented by ranking the hypotheses. The second thrust is focusing on the issues in sensor allocation. Sensor allocation must redirect resources to the exception-handling activities without jeopardizing the other behaviors. Sensor allocation must also participate in the recovery process. We are incorporating allocation routines into the sensing-manager module of the SFX-EH. The sensing manager is responsible for noticing when sensors have overlapping but not identical functionality, and how to exploit this; e.g., a laser rangefinder can serve as an image device and a range sensor.

Acknowledgments

This research is supported in part by NSF grant IRI-9320318, ARPA grant AO#B460, and NASA/JSC contract NAS.9-19040 while the authors were at the Colorado School of

Mines. The authors would like to thank Greg Chavez for his efforts on an earlier version of this article, and Charlie Ozinga and the anonymous reviewers for their helpful comments.

References

- Bajcsy, R. 1988. Active perception. *Proc. IEEE* 76(8).
- Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE J. Robot. Automat.* 1(1):1-10.
- Chavez, G. T., and Murphy, R. R. 1993. Exception handling for sensor fusion. *SPIE Sensor Fusion VI*, pp. 142-153.
- Chen, C. X., and Trivedi, M. M. 1995. Task planning and action coordination in integrated sensor-based robots. *IEEE Trans. Sys. Man Cybernet.* 25(4):569-591.
- Doyle, R. J., Atkinson, D. J., and Doshi, R. S. 1986. Generating perception requests and expectations to verify the execution of plans. *Proc. of the Natl. Conf. on Art. Intell.*, pp. 81-88.
- Ferrell, C. 1993. Robust agent control of an autonomous robot with many sensors and actuators. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.
- Gini, M., and Gini, G. 1983. Error diagnosis and repair in sensorial robots. *Proc. of the Advanced Robot. Intl. Conf.*, pp. 11-27.
- Hammond, K. J. 1986. Chef: A model of case-based planning. *Proc. of the Natl. Conf. on Art. Intell.*, pp. 267-271.
- Hanks, S., and Firby, R. J. 1990. Issues and architectures for planning and execution. *Proc. of a Workshop on Innovative Approaches to Planning, Scheduling and Control*.
- Henderson, T., and Grupen, R. 1990. Logical behaviors. *J. Robot. Sys.* 7(3):309-336.
- Henderson, T., and Shilcrat, E. 1984. Logical sensor systems. *J. Robot. Sys.* 1(2):169-193.
- Howe, A. E., and Cohen, P. R. 1990. Responding to environmental change. *Proc. of a Workshop on Innovative Approaches to Planning, Scheduling and Control*.
- Hughes, K. 1993. Sensor confidence in sensor integration tasks: A model for sensor performance measurement. *SPIE Applications of Artificial Intelligence XI: Machine Vision and Robotics*, pp. 169-193.
- Kolodner, J. L. 1987 (June). Extending problem-solver capabilities through case-based inference. *Proc. of the 4th Intl. Workshop on Machine Learning*, pp. 167-178.
- Lee, M. H., Barnes, D. P., and Hardy, N. W. 1983 (November). A control and monitoring system for multiple-sensor industrial robots. *Robot Vision and Sensory Controls 3rd Intl. Conf.*, p. 471.
- Lindsay, R. K., Buchanan, E. A., Feigenbaum, E. A., and Lederberg, J. 1980. *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project*. New York: McGraw-Hill.
- Murphy, R. R. 1991. State-based sensor fusion for surveillance. *SPIE Sensor Fusion IV*, pp. 331-345.

- Murphy, R. R., and Arkin, R. C. 1992. Sfx: An architecture for action-oriented sensor fusion. *1992 IEEE/RSJ Intl. Conf. on Intell. Robots and Sys. (IROS)*. Los Alamitos, CA: IEEE, pp. 1079–1086.
- Murphy, R. R., and Hershberger, D. 1996. Classifying and recovering from sensing failures in autonomous mobile robots. *American Assoc. for Art. Intell., 13th Natl. Conf. AAAI Press*, pp. 922–929.
- Murphy, R. R., and Mali, A. Forthcoming. Lessons learned in integrating sensing into autonomous mobile robot architectures. *J. Exp. Theoret. Art. Intell.*
- Murphy, R. R., and Rogers, E. 1996. Cooperative assistance for remote robot supervision. *Presence* 5(2):224–240.
- Noreils, F. R., and Chatila R. G. 1995. Plan-execution monitoring and control architecture for mobile robots. *IEEE Trans. Robot. Automat.* 11(2):255–266.
- Payton, D. W., Keirsey, D., Kimble, D. M., Krozel, J., and Rosenblatt, J. K. 1992. Do whatever works: A robust approach to fault-tolerant autonomous control. *J. Appl. Intell.* 2:225–250.
- Pearce, D. 1988 (St. Paul, MN). The induction of fault-diagnosis systems for qualitative models. *AAAI-88, Proc. of the Seventh Natl. Conf. on Art. Intell.*, pp. 353–357.
- Rich, E., and Knight, K. 1991. *Artificial Intelligence*, 2nd ed. New York: McGraw-Hill.
- Rogers, E., Murphy R. R., and Ericson, B. 1997. Agent-based expert assistance for visual problem solving. *Autonomous Agents 97*, pp. 156–163.
- Simmons, R., and Davis, R. 1987 (August). Generate, test and debug: Combining associational rules and causal models. *Proc. of the 10th Intl. Joint Conf. on Art. Intell.*, pp. 1071–1078.
- Vander, W., Velde, E., and Carignan, C. R. 1984. Number and placement of control system components considering possible failures. *J. Guidance Control* 7(6).
- Weller, G. A., Groen, F. C. A., and Hertzberger, L. O. 1989. A sensor-processing model incorporating error detection and recovery. In Henderson, T.C. (ed.) *Traditional and Nontraditional Robotic Sensors*. Springer-Verlag.