Distributed Multi-Agent Diagnosis and Recovery from Sensor Failures

Matthew T. Long and Robin R. Murphy
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620
Email: {*mtlong,murphy*}*@csee.usf.edu*

Lynne E. Parker
Department of Computer Science
University of Tennessee
Knoxville, Tennessee 37996–3450
Email: *parker@cs.utk.edu*

***Abstract*— This paper presents work extending previous research in sensor fault tolerance, classification, and recovery from a single robot to a heterogeneous team of distributed robots. This approach allows teams of robots to share knowledge about the working environment, sensor and task state, to diagnose failures and also communicate to redistribute tasks in the event that a robot becomes inoperable. Our work presents several novel extensions to prior art: distributed fault handling and task management in a dynamic, distributed Java framework. This research was implemented and demonstrated on robots in a lab environment performing a simplified search operation.**

I. Introduction

Autonomous mobile robots require robust sensing to enable useful control of planned actions. Failures in sensing can lead the robot towards incorrect actions or dangerous situations [1]. Unfortunately, these failures are extremely common and can stem from a number of sources: malfunctions, miscalibration, or errant planning. Unexpected changes in the vicinity of the robot can also have deleterious effects on the usability of the agent – changing lighting conditions can wreak havoc with the best vision algorithms and can even cause sensor "hallucinations" leading to unforeseen actions.

These types of sensing degradation can be extremely difficult, if not impossible, to model and classify; a robot may not necessarily be able to uncover the true cause of a failure, or sometimes even determine that a failure is taking place. Physical sensor failures (e.g., loss of power) can often be detected at the hardware level given a well-designed sensor package. With an environmental change, however, it can be difficult to differentiate between a sensing failure due to the environmental condition (e.g., lights turned off) or some other cause, such as occlusion of the sensed object. Unfortunately, each cause has a separate method to continue the task – in the first, the robot may wish to disregard the useless sensor data or switch to a different sensor, while in the second case, the agent may need to recover through the use of a new search behavior or additional reasoning about the world state. Correctly determining the proper recovery may require collaboration by other sensors, whether the sensors reside on the same agent, as demonstrated in [2] or on a peer, as demonstrated in this paper.

This paper is not concerned with error detection, save in the most general sense. It is assumed that failures will be detected properly and the appropriate exceptions raised. Instead, this paper is concerned with single- and multiple-agent sensor recovery from a detected fault.

In this paper, sensor failure handling is both a local and a distributed effort. A distributed environment enables access to more sensors and more information about the world, so a distributed environment can potentially allow a more accurate and reliable error classification. This system allows an agent to query a remote peer for sensor information and request that a peer execute a test or set of tests on sensors. The results of these tests are correlated with the results of local tests to enhance confidence about the state of local sensors. For example, it is very difficult for a robot with a single camera to correctly diagnose many failures – a physically damaged camera often responds identically to a camera that is in a darkened room. Correlating data and test results from a peer in the area can help distinguish between the two cases. A single robot should suffer gradual degradation in the face of sensing failures, and extensions into a distributed arena should not make the situation worse – the system should be tolerant of partial failure, latency and other distributed issues.

For most domains, the completion of a task is more important than the state of one individual robot. Rather than continue blindly on a task and jeopardize the overall mission, a failing robot may attempt to communicate its state to a peer and relinquish the task to a more suitable peer. Taking advantage of the dynamic nature of the environment, a robot can even call upon another robot that has no knowledge of how to perform that task, either by potentially using the remote robot's sensors and effectors directly – a robot "teleoperating" a peer – or by transmitting the knowledge of how to complete the task and allowing the peer to act autonomously.

II. Related Work

While related issues such as fault detection and diagnosis have been discussed in other works, the issue of how to handle sensing failures is largely uninvestigated, and work on distributed sensing recovery, in the sense denoted in this paper, is even more scarce.

Much of the work on error detection, classification and diagnosis, such as Soika [3] or Reed [4] is applicable to this domain, but does not address the issue of recovery. Tools such as analytical redundancy [5], [6] do not directly apply to this work, as these works assume that the sensing capabilities of a robot system can be completely modeled. In a distributed, dynamic and partially unknown environment, this is unlikely to be the case. However, work not reported in this paper is looking at incorporating analytical approaches for situations where models are known or valid with SFX-EH. The literature on distributed fault recovery for autonomous robotics is virtually nonexistent. However, systems such as Jini [7] and Lim's [8] distributed service concept for information dissemination provides a framework for distributed services and a foundation for this work. Both provide for distributed lookup of services and dynamic mobile code. Stroupe, Martin and Balch [9] fuse sensor data shared between multiple robots using Gaussian distributions. Hoover and Olsen [10] have implemented a sensor network consisting of a static camera sensor network and a mobile robot for robot navigation. This concept could be extended to use multiple autonomous robots in place of static cameras.

This work is based off of earlier work on Murphy's SFX-EH architecture [11], particularly Murphy and Hershberger [2]. This article builds off of prior work and proposes distributed extensions. The error classification scheme now uses remote, distributed sensor data to verify local hypotheses about the world. This generates more information for validating hypotheses and the distributed environment allows higher-level error handling, including task transfer between agents.

Prior work has been done with SFX-EH [2], attempting to enhance real-time characteristics by pre-computing a shortest-time decision tree. This decision tree took timing information for tests and recovery methods along with a knowledge-engineer-supplied hypothesis library to find the quickest path to recovery.

However, this approach suffered from several flaws. These hypothesis libraries do not easily extend to a multi-agent environment, because of its inherently dynamic nature – there is no guarantee that there will be a peer available, much less the timing or latency of remote tests. The decision tree was generated at system start to avoid run-time delays, but was difficult to change dynamically. Additionally, the programmer was required to design and maintain the hypothesis library, which turned out to be non-intuitive and time-consuming. The new implementation does not follow this approach, but instead dynamically determines the proper ordering of tests at run-time, based on timing information of each test and inter-test dependencies.

III. Approach

This section discusses the approach taken in this paper to distributed sensor fault recovery. First, however, a general understanding of sensing failures and recovery is required as related to a single-agent system.

A. Recovery Strategies

The system handles errors in the following manner: reconfiguration of the perceptual schema, recalibration of the sensor, or other corrective actions.

Reconfiguration is perhaps the simplest recovery strategy, and is typically the first corrective action that the error classifier will take. After determining that the sensor has failed, the classifier will reactivate the failed perceptual schema, causing it to search for a new sensor logically equivalent to the failed sensor.

Recalibration is often performed as a background recovery task, since many times the recalibration of a sensor is a complex task. However, recalibration of sensors offers the possibility of more intelligent behavior. If a sensor is in a low-light situation, it may be possible to reconfigure the camera to automatically apply various image enhancements or filters to restore usability without the perceptual schemas being required to each manually apply image filters – it will all be done "under the hood" at the sensor level.

Other *corrective actions* can be taken as well to try to return sensors to active service. For example, if a sensor group is disabled by placing opaque cloth over a set of sensors the affected sensors might all report a low light situation. Correlating the low light over all the sensors might suggest a corrective action such as shaking the robot or backing up to possibly dislodge the offending item. Note that the error classifier would have no concept of "cloth" or "being covered" at this stage, but would simply know that a group of sensors were not working correctly.

However care must be taken that the error handler does not interfere with the overall usability of the robot. If the error handler were to suddenly reverse the robot without knowledge of the overall state of the world, such as an obstacle to the rear, it might put the robot into a worse situation or risk damaging other sensors. In this case, the error classifier needs to notify the task planner or mission controller that the action *should* be taken, but other concerns might override this suggestion or delay it until a later, more convenient time.

Using this model also suggests an area for multi-agent collaboration. A peer might be able to take corrective action that the failed robot might not. In the above

example, a peer might be able to remove an item obscuring a sensor or even turn on attached lights to add additional luminance to an area and enable a failed camera.

B. Distributed sensing issues

Extending the above failure handling concepts into a multi-agent realm adds several issues for consideration: logical sensors across multiple robots, distributed sensor correlation, and recovery.

Distributed sensing is challenging since the system is dynamic by design. The number of sensors available to an agent during the error handling is not fixed in any way. Since mobile robots are mobile and not fixed in one location, they can move into and out of communication range of other robots. This is especially important in many domains, such as search and rescue, where the environment seriously limits wireless communication. As robots enter a reasonable proximity and communication is established, each becomes aware of the capabilities of the others, and all can take advantage of cooperatively sharing sensors, data and other knowledge.

With the introduction of additional robotic agents, it is possible to stretch the concept of logical sensors across multiple agents. A color camera on a remote peer may be considered equivalent to a local color camera, with different execution characteristics and attributes, such as location and refresh time. However, logical remote sensors introduce a host of issues, such as partial failure and latency. While these issues have not yet been addressed in general, the current implementation can utilize remote sensors to validate hypotheses generated during the error processing.

Fortunately, a distributed environment enables access to more sensors and more information about the world, so a distributed environment allows a more accurate and reliable error classification. This system allows an agent to query a remote peer for sensor information and request that a peer execute a test or set of tests on sensors. The results of these tests can be correlated with the results of local tests to enhance confidence about the state of local sensors. For example, it is very difficult for a robot with a single camera to correctly diagnose many failures – a physically damaged camera often responds identically to a camera that is in a darkened room. Correlating data and test results from a peer in the area can help distinguish between the two cases.

IV. Implementation

The implementation consists of a number of components, both traditional single-agent and new multi-agent. Important components are the Sensor Fusion Effects (SFX) core architecture, multi-agent extensions to the architecture, error handler and classifier, and remote communication subsystems. Additional discussion is given regarding the implementation language and other supporting techologies.

The local error handler and SFX architecture have been implemented in previous work, however the extensions that allow remote communication and remote error handling and classification are novel. The SFX architecture was chosen as a base for this work since many of the concepts of single-agent error recovery had been implemented previously in prior work [2]. This work adds extensions to handle multi-agent testing, communication and collaboration. One key feature of SFX that is heavily utilized in the fault-recovery process is the use of *logical* sensors. Behaviors in SFX consist of two parts: a perceptual schema to generate a percept and a motor schema to act on the percept. Any sensor and algorithm that is able to generate the same percept for a behavior is said to be a logical sensor. These logical sensors exist on a per-perceptual-schema basis and all the logical sensors used by a perceptual schema generate the same percept. Each may have other intangibles, such as execution speed or latency that can influence the desirability of that logical sensor for the behavior. The importance of this will be seen shortly.

Another module of note in the SFX architecture is the *Sensing Manager*. This manager is responsible for resolving conflicts related to sensors, such as prioritizing and allocating sensors among all schemas that need the sensor. The error classifier and error handling module is part of the Sensing Manager, since it is responsible for testing and reallocating failed sensors.

A. Classification

Error classification in SFX-EH uses a variant of the "Generate, Test and Debug" (GTD) algorithm [2], [12]. This is best shown by example, and is discussed in detail in Sections V-C and V-D.

B. Error Handler

A new Java based framework was implemented, utilizing the Jini and RMI technologies to handle remote method calls and services. Additionally, work on previous error handlers [2] was extended to take advantage of remote sensors and data.

Fig. 1 shows an updated overview of the architecture, as it relates to distributed error handling, with changes to the base SFX-EH denoted by the shaded areas. The additions for this project are the use of remote test data, and possibly even remote sensors in a behavior, although this last is still future work.

C. EHKS

The primary data structure used by the error handler is the *Exception Handling Knowledge Structure*. As a fault is detected, a run-time exception is generated and an

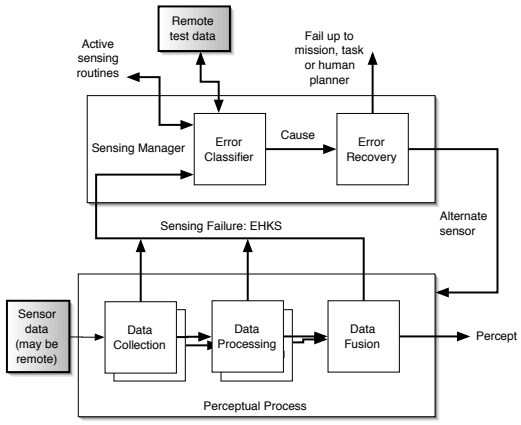

Fig. 1. The SFX architecture, with distributed error handling

EHKS is generated. This EHKS is filled with information from the appropriate portions of the SFX framework, which is later used for classification. The nature of the error determines the SFX modules that are relevant and the information that is stored at any given time. For example, if a hardware fault is detected at a low level, a minimal amount of information is required for the classifier to diagnose and recover from the problem: the faulty sensor and the containing perceptual schemas and behavior. However, if the error occurs during sensor fusion of data from multiple sensors, it may not be immediately clear with which sensor the problem lies. In this case, more information must be generated and stored for use by the classifier, and the work of the classifier is more difficult.

The EHKS deserves some comment. The EHKS contains fields for the *failure step*, which denotes what the robot was doing when the error occurred. Errors are commonly detected during the data acquisition step, the data processing step, or during sensor fusion. The *error* is a description of what went wrong, and serves as a hint to the classifier. The error might show that there was low confidence in the sensor data, or that there was a conflict between multiple sensors. The remainder of the EHKS is filled with the following: *bodies of evidence* that show the state of relevant sensors when the error occurred, *environmental preconditions* that might effect the sensors, and a list of *frames* that give the classifier access to the state of the sensors, perceptual schemas and behaviors that might affect classification.

D. Local and Distributed Execution of the Error Handler

Fig. 2 shows the flow of an error through the system. The Sensing Manager initially passes the EHKS to the Error Classifier for processing. The Error Classifier begins the GTD process, creating a set of hypotheses that are individually tested to validate or deny the hypothesis.

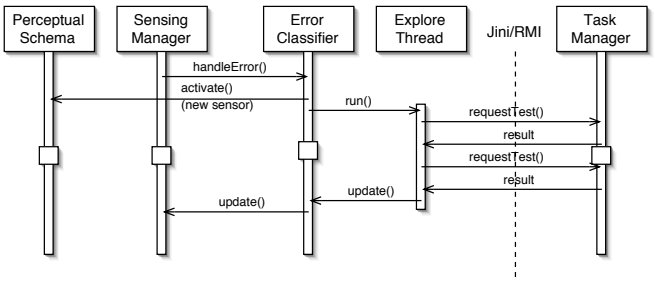

Fig. 2. Remote test states

The local error classification attempts to short-circuit most of the testing, performing the most basic hardware tests and trying to limit the number of more expensive environmental tests run. This can be done if there is a sufficiently diverse sensor payload – the classifier assumes failure initially, and logically equivalent sensors that use a different sensing modality will be selected first. This initial testing determines a quick-and-dirty recovery that will allow the robot to continue its task. An exploration thread is also spawned for more in-depth testing of the hypotheses.

Once the local testing is complete, the sensing manager and perceptual schemas are updated to use the new sensor, and the failed sensors are disabled. The exploration thread then, as system resources allow, will continue testing of the failed sensors. This thread also examines the remote services offered by peers, and queries those offered services to correlate remote data with local test results.

If the exploration thread classifies the error more accurately than the initial best-guess, then it may be advantageous to update the sensing manager and perceptual schemas with a different sensor. The initial guess may have altered the a schema to use a sonar over a possibly-failed camera, but later testing may reveal that the camera is working correctly and may provide better results than the sonar.

E. Java / JINI Implementation

The Java programming language is well suited to the dynamic and distributed nature of the task. Jini, a Java technology for service discovery and registration, was used for broadcasting the services, such as sensors and behaviors, offered by an agent, and RMI, Java's Remote Method Invocation, was used as the underlying technology for remote method calls and mobile objects and code. Java also has a rich set of libraries that simplify the task of system development.

Fortunately Java and many of its core packages were created to simplify development. For example, Jini and RMI (Remote Method Invocation) are intended to allow an agent to export local services and activate services offered

by remote agents. With Jini, complex distributed actions can be developed relatively simply.

Jini works by instantiating a number of lookup services on a network. As each agent activates, it looks for a lookup service and notifies each that it has a number of services that are available for use by a peer. As the network grows, agents become aware of services offered by others within the network (the *federation* in Jini-speak). The registration process is dynamic, and agents that are interested in certain services can be notified that new services have been instantiated or old services are no longer available.

From the SFX standpoint, many of the core SFX modules can be exported in this fashion and made available to others. Sensors, tests and behaviors are among the most obvious services that can be easily used by a remote client, and this implementation does indeed use these three services.

V. Demonstrations

The following sections present two demonstrations of the goals of this paper. The first demonstrates the *execution of the error classification and recovery unit on a single system*, and the second shows that a *physically distributed peer can aid in error diagnosis and error recovery*. These demonstrations provide existence proofs that multi-agent diagnosis and recovery is possible in this context.

A. Test Hardware

The demonstrations use two robots from Oak Ridge National Labs: *Augustus* and *Theodosius*. Both robots are RWI ATRV-Mini robot bases and have a color camera mounted on a pan-tilt unit as a primary sensor, as well as the standard complement of core sensors: sonar, power, compass and encoders for position. Unfortunately, there were no robots with physically redundant sensors available. To simulate and test this important capability of the error handler, each camera and sonar sensor was instantiated twice internally – this essentially creates two distinct virtual sensors on the same hardware. This is valid simulation since the internal errors generated are the same, whether generated by an actual or a simulated hardware failure. The benefit of this is that an operator can disable a virtual sensor through the operator console, and the error handler will be able to recover to the other virtual sensor.

B. Test Domain

The scenario for this demonstration is a team of robots searching a room for a target orange-colored cone. The team consists of two robots, Augustus and Theodosius. The test domain for the demonstration is a search mission.

Both robots have the same complement of basic behaviors: **find-object**, **track-object**, **move-to-object**, and **avoid-obstacles**. However, the behaviors are activated differently on each agent, depending on the overall mission state.

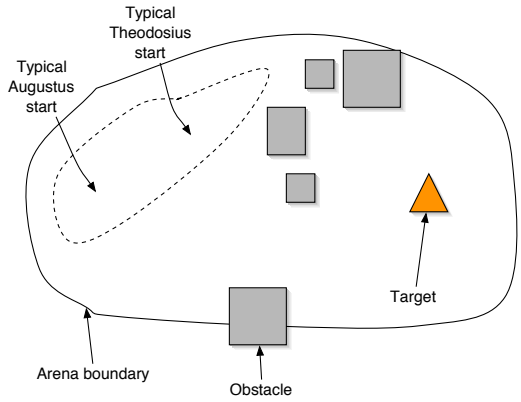

Fig. 3. A sketch of the arena used for the demonstration

The robots were set up in the Center for Engineering Science and Advanced Research (CESAR) robotics lab at the Oak Ridge National Labs. They were placed in an arena approximately 40' by 25', with opaque blocks placed within. The robots were placed roughly 25' from the target cone, and due to the initial placement, could sometimes see the cone at start and sometimes needed to search prior to visual acquisition. Fig. 3 shows the typical layout of the test arena. For brevity, Augustus typically started seeing the cone, while Theodosius could not.

The cone is detected through a color segmentation of appropriate values in HSI color-space. The resulting segmented image is passed through a basic connected-components algorithm to find the boundaries of the cone and provides a directional percept to the navigational schemas. The sonar are also utilized to find obstacles and other hazards. Currently there is no high-level mapping behavior, and the robots are purely reactive. Behavior activation is controlled by the TaskManager portion of the SFX framework and is based on the world state.

C. Single Agent Diagnosis and Recovery

The purpose of this demonstration is to show the execution of the error classifier in the single-agent case. The demonstration uses a single robot, Augustus, exploring the arena for the target. The active behavior during the demonstration was the track-object behavior, but could have been any other behavior.

In this demonstration, the operator introduces a hardware failure in a color camera. Since the camera hardware is too expensive to actually damage, this fault is introduced remotely, and a hardware fault is simulated in the driver software.

Diagnosis and recovery proceeds as follows:

a) The fault is detected in the low-level camera driver software.

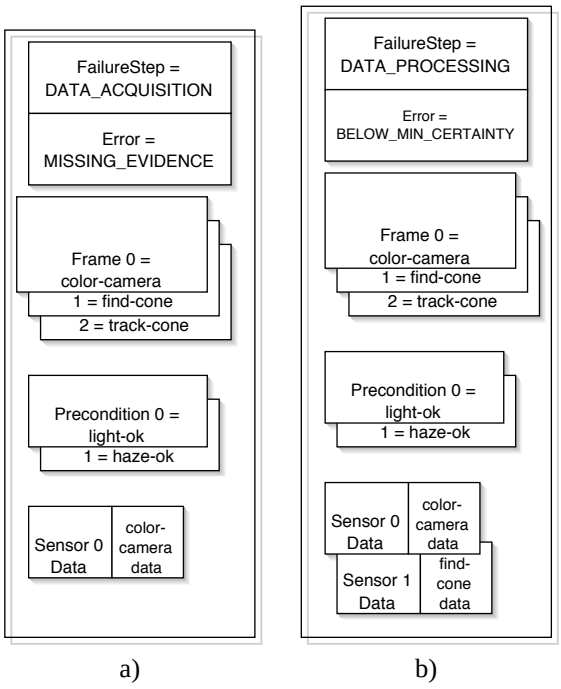

Fig. 4. EHKS generated by sensor failure for a) singe-agent and b) multi-agent recoveries

b) Initially, a stub EHKS is generated and a Java exception is thrown. As the exception is passed up the call stack, each execution frame adds information to the EHKS. The complete EHKS is shown in Fig. 4a.

c) When the exception reaches the topmost execution frame, the SensingManager's error handler is invoked with the EHKS, and diagnosis and recovery can begin.

d) The generate-test-debug stage begins error classification and handling. The classifier immediately determines that the fault is hardware-related, and short-circuits the bulk of the testing. The classifier first executes quick hardware tests on the faulty camera and its equivalent sensors.

e) The classifier then activates the backup camera. With a valid backup available and active, the behavior reactivates and the mission continues.

f) A testing thread is instantiated since the possibility exists that further information can be extracted from the failed sensor. The execution of this thread does not occur in the primary classification and recovery step.

g) The test cycle ends with diagnosis and recovery complete.

In this demonstration there is no distributed component to the diagnosis, since there is only one active agent. This demonstration is intended to show the correct execution of the error handler in the most simple case; further demonstrations evaluate the distributed component.

Fig. 5. Placing a box on the cameras triggers an environmental failure

The recovery consists of reactivating the track-cone behavior with the find-cone perceptual schema using the alternate, good camera. Note that this equivalent camera has not been fully tested – only basic hardware tests are run. The camera is assumed to work, however it may generate a later fault if there is an environmental condition that prevents correct functioning.

Further testing is performed by the testing thread, which can be scheduled to execute as system resources permit. If the failed sensor ever recovers, it will be returned to service.

D. Multi-Agent Diagnosis and Recovery

The second demonstration shows the cooperative, distributed capabilities of the error classification scheme. This demonstration uses multi-agent sensor testing and communication to share sensor state and manage tasks on physically distributed agents. This demonstration uses both Augustus and Theodosius, with Augustus assuming the role of primary search robot and Theodosius held in reserve.

The sensor failures are generated through a physical environmental failure on all visual sensors situated on the primary search robot. This is accomplished by placing an opaque box over the camera, which completely blocks visual perception, shown in Fig. 5. Diagnosis and recovery for this error is more complicated than previously encountered errors.

The error is processed as follows:

a) As before, a recognized error triggers the generation of a full EHKS. There are significant differences, however. The box over the cameras does not trigger an immediate error, but rather lowers the light level sufficiently and causes the active perceptual schema to fail to detect

needed visual cues. Once the perceptual schema is unable to generate a percept, the schema enters an error state that begins the error process.

b) The EHKS differs substantially as well, since there is no immediate and apparent source of the failure. In a schema using multiple sensors and sensor fusion, the EHKS would contain all the sensors that might have failed, but in this demonstration is limited to the camera that is faulty. Additionally, the type of error is set to "BELOW_MINIMUM_CERTAINTY" to denote that the schema failed through uncertainty as to the target's location and the failure step was set to "DATA_PROCESSING". As the exception passes up the call stack, the frame information is stored for use by the error handler. (Fig. 4b)

c) When the exception reaches the topmost execution frame, the SensingManager's error handler is invoked with the EHKS, and diagnosis and recovery can begin.

d) Within the error classifier, the primary step involves generating the list of hypotheses to explain the fault. Here, two hypotheses have been generated with the EHKS, and will form the core of the testing. The hypothesis **light-ok** seeks to verify that the ambient light level is sufficient for the camera. The second hypothesis, **haze-ok** requires sufficient ambient light available, and verifies that the camera can distinguish objects in the scene. This failure could be caused by haze, film, water or some other substance on the lens.

e) Once the list has been filled, tests are executed. All test dependencies must be met before a given test can execute. For example, before **light-ok** can be executed, a set of hardware tests will run that check power and connection to the low-level driver. Since the sensor payload contains a second visual sensor, the second sensor is tested as well to verify the hypothesis.

f) As a result of the tests, the system determines that the light-level is too low for any available sensor, and notes that an environmental change has caused the problem.

g) Since there is no equivalent sensor to activate, the **track-cone** behavior is deactivated.

h) However, the possibility exists that further information can be extracted from the failed sensors, or that the environment may change again to reactivate the sensors. To this end, a testing thread is created and marked for later execution. This test execution does not occur in the primary classification and recovery step.

i) With diagnosis and recovery complete, the initial test cycle ends.

In this demonstration, the testing thread executes after a short delay – production systems should use a more sophisticated scheduling algorithm. The execution of the testing thread also activates the distributed component of the error handling architecture. During this comprehensive testing, the Jini lookup service is contacted when an environmental test is executed, and remote sensors that can test the environmental condition being examined are polled. This remote test data is examined along with the local test results to aid in diagnosis. During this demonstration, the cameras on the second robot report that the light level is normal, at odds with the local test results.

Recognizing the remote sensors are functioning has little direct impact on the first robot's availability; there is no direct local recovery for this state as yet. However, the second robot is notified of the Augustus' failure and takes over the initial task. While the first robot is out of commission, and remains so for the duration of the demonstration, the target is successfully located by the second robot, Theodosius.

The current implementation assumes the robots are located within the same region, and that the actual environment is equivalent for both robots. The set of sensors that is available for remote queries is determined dynamically during runtime and may vary between executions of the error handler and testing threads, as agents move into or out of communication range.

The demonstrations presented here are intended to validate the extensions to SFX-EH by showing correct behavior of a single-agent system, followed by fault diagnosis and recovery in a multi-agent environment. In all cases, the system diagnosed and recovered from operator-introduced faults to complete the mission.

VI. Conclusions and Future Work

The multi-agent approach to sensor fault tolerance has several advantages. Since no agent has a full causal model of the world, or completely redundant hardware and software, no agent alone is able to diagnose a failure or even determine that a failure has occurred with complete accuracy. This being the case, the more information that can be gathered to corroborate or disprove a hypothesis, the more confidence an agent can have in the results. Corroborating evidence can be gathered from multiple sensors on a a single robotic platform, but can also be requested from similar sensors on other agents in the vicinity.

A benefit of the approach used is that the available and logical sensors for both the local platform and remote peers is determined at runtime. Each time an error is handled the appropriate action is taken, which might be different, even for the same error, depending on the

environment, peers in communication range, and their sensors.

Current work only scratches the surface of the potential uses for multi-agent collaboration. This implementation only uses remote sensors on peers in a limited manner – for corroborating environmental hypotheses. However, there are many more uses. Sharing sensors – having a remote sensor logically replacing a local sensor – raises issues such as localization and coordinate transformation that are not yet fully addressed and is left for future work.

Since the current implementation has only been implemented and tested in a lab environment, the work covered in this paper must be tested and used in more robust settings. Operating outdoors and in non-static environments will provide more complex interactions as well as more opportunities for sensing failures. Additionally, this work must be extended to larger robot teams to truly examine distributed multi-robot error recovery.

VII. Acknowledgments

This work has been conducted under a grant from DOE (DE-FG02-01ER45904).

VIII. References

[1] J. Carlson and R. R. Murphy, "Reliability analysis of mobile robots," in *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, 2003, p. In publication.

[2] R. R. Murphy and D. Hershberger, "Handling sensing failures in autonomous mobile robots," *The International Journal of Robotics Research*, vol. 18, no. 4, pp. 382–400, March 1999.

[3] M. Soika, "A sensor failure detection framework for autonomous mobile robots," in *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1997, pp. 1735–1740.

[4] N. Reed, "Constructing the correct diagnosis when symptoms disappear," in *Proceedings Fifteenth National Conference on Artificial Intelligence AAAI-98*, 1998, pp. 151–156.

[5] M. L. Leuschen, J. R. Cavallaro, and I. D. Walker, "Robotic fault detection using nonlinear analytical redundancy," in *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, 2002.

[6] D. Um and V. Lumelsky, "Fault tolerance via analytic redundancy for a modularized sensitive skin," in *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1999.

[7] K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, A. Wollrath, and B. O'Sullivan, *The Jini Specification*. Addison-Wesley Pub Co, 1999.

[8] A. Lim, "Distributed services for information dissemination in self-organizing sensor networks," *Journal of the Franklin Institute*, vol. 38, no. 6, pp. 707–727, 2001.

[9] A. W. Stroupe, M. C. Martin, and T. Balch, "Distributed sensor fusion for object position estimation by multi-robot systems," *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pp. 1092–1098, 2001.

[10] A. Hoover and B. Olsen, "Sensor network perception for mobile robotics," in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, 2000.

[11] R. R. Murphy, *Introduction to AI Robotics*. The MIT Press, 2000, ch. 7, pp. 268–274.

[12] G. T. Chavez and R. R. Murphy, "Exception handling for sensor fusion," *SPIE Sensor Fusion VI*, pp. 142–153, September 1993.