To the Graduate Council:

I am submitting herewith a dissertation written by Balajee Kannan entitled "LeaF: A Learning-based Fault Diagnostic System for Multi-Robot Teams". I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Lynne E. Parker
Major Professor

We have read this dissertation
and recommend its acceptance:

Michael W. Berry

Bruce J. MacLennan

Itamar Elhanany

Accepted for the Council:

Carolyn Hodges
Vice Provost and
Dean of Graduate School

(Original signatures are on file with official student records.)

LeaF: A Learning-based Fault Diagnostic System for Multi-Robot Teams

Presented for the
Doctoral Degree
The University of Tennessee, Knoxville

Balajee Kannan
May 2007

Dedication

To my wonderful family who gave me the inspiration to dream and my amazing wife who showed me the path to achieve and ensured that I never strayed, all along with a sense of un-equivocal love and dedication. In a small way, may this Ph.D. dissertation inspire others to achieve their goals through hard work and perseverance.

Acknowledgements

This dissertation could not have been completed without the support and guidance of numerous people. First and foremost, I would like to express my deepest thanks to my advisor, Lynne Parker, for her infinite patience in dealing with me. I can not imagine a more inspirational person to guide me through the course of my research and without whose common-sense, knowledge and perceptiveness, I would never have completed my dissertation. I would like to thank the rest of my thesis committee, Dr. Bruce MacLennan, Dr. Itamar Elhanany and Dr. Michael Berry for their invaluable guidance and suggestions that helped broaden and improve this work. I would also like to thank Dr. Stacy Prowell and Murray Browne for their words of encouragement in times of need. I really enjoyed being a part of the Distributed Intelligence Laboratory (DI Lab) from its inception. Working with robots was an experience onto itself. I would like to give special thanks to my colleagues Michael Bailey, Xingyan Li, Yuanyuan Li and Yifan Tang, who have offered constant support and encouragement in addition to putting up with my eccentricities. I have enjoyed working with them tremendously.

Many administrative assistants and staffs have provided me great help in countless occasions. I would like to give thanks to Dorsey Bottoms, and Donna Bodenheimer, whose offices I probably frequented more than mine. Many thanks to Greg Shutt, Markus Iturriaga and the entire lab staff for patiently helping me with the robots.

I am constantly amazed by my parents, Venkatraman Kannan and Usha Kannan. They have always have encouraged me, bailed me out of trouble more than a few times in my formative years, and never ever lost faith in me even when I did. My sister Dharini Kannan who has been a constant source of inspiration through her hard-work and sincerely. My humble thanks to my mother and father-in-laws Rani and Dr. Gothandaraman, who took care of me as their own. Finally, words fail to express my gratitude for my staunchest supporter, my wife Akila, without whom I know I would not have been able to achieve any of my dreams. I pray that I have the fortitude and courage to always stand by her side as she has by mine. I hope, in some small way, that this dissertation makes all the people who have touched my life proud of what I have been able to accomplish because of them.

Abstract

The failure-prone complex operating environment of a standard multi-robot application dictates some amount of fault-tolerance to be incorporated into every system. In fact, the quality of the incorporated fault-tolerance has a direct impact on the overall performance of the system. Despite the extensive work being done in the field of multi-robot systems, there does not exist a general methodology for fault diagnosis and recovery. The objective of this research, in part, is to provide an adaptive approach that enables the robot team to autonomously detect and compensate for the wide variety of faults that could be experienced. The key feature of the developed approach is its ability to learn useful information from encountered faults, unique or otherwise, towards a more robust system. As part of this research, we analyzed an existing multi-agent architecture, CMM – Causal Model Method – as a fault diagnostic solution for a sample multi-robot application. Based on the analysis, we claim that a causal model approach is effective for anticipating and recovering from many types of robot team errors. However, the analysis also showed that the CMM method in its current form is incomplete as a turn-key solution. Due to the significant number of possible failure modes in a complex multi-robot application, and the difficulty in anticipating all possible failures in advance, one cannot guarantee the generation of a complete *a priori* causal model that identifies and specifies all faults that may occur in the system. Therefore, based on these preliminary studies, we designed an alternate approach, called LeaF: Learning based Fault diagnostic architecture for multi-robot teams. LeaF is an adaptive method that uses its experience to update and extend its causal model to enable the team, over time, to better recover from faults when they occur. LeaF combines the initial fault model with a case-based learning algorithm, LID – Lazy Induction of Descriptions — to allow robot team members to diagnose faults and to automatically update their causal models. The modified LID algorithm uses structural similarity between fault characteristics as a means of classifying previously un-encountered faults. Furthermore, the use of learning allows the system to identify and categorize unexpected faults, enable team members to learn from problems encountered by others, and make intelligent decisions regarding the environment. To evaluate LeaF, we implemented it in two challenging and dynamic physical multi-robot applications.

The other significant contribution of the research is the development of metrics to measure the fault-tolerance, within the context of system performance, for a multi-robot system. In addition to developing these metrics, we also outline potential methods to better interpret the obtained measures towards truly understanding the capabilities of the implemented system. The developed metrics are designed to be application independent and can be used to evaluate and/or compare different fault-tolerance architectures like CMM and LeaF. To the best of our knowledge, this approach is the only one that attempts to capture the effect of intelligence, reasoning, or learning on the *effective* fault-tolerance of the system, rather than relying purely on traditional redundancy based measures. Finally, we show the utility of the designed metrics by applying them to the obtained physical robot experiments, measuring the effective fault-tolerance and system performance, and subsequently analyzing the calculated measures to help better understand the capabilities of LeaF.

Contents

List of Tables

List of Figures

Glossary

f_l	Leaf node, 32
A_j	Task success metric, Measure of a successful task T_j, 41
B_j	Task failure metric, Measure of a failed task T_j, 41
D_f	*Similitude* or similarity set built by the learner based on matching symptoms between fault nodes, 29
PS	Problem set or set of all possible fault descriptions that can be encountered by the system, 29
P	Effective performance metric, 42
R	Team of robots, 40
S	Solution set or set of all corrective actions for the corresponding fault from the problem set, 29
T	Set of tasks, 40
V	Vertices set or set of all fault nodes from the causal model, 29
δ	Learning exhibited by a system, 43
ϵ	Efficiency of a system, 43
HWB_{eff}	Effective measure for the redundancy based fault tolerance metric proposed by Hamilton and Walker, 13
π	Path, defined as the sequence of inter-related nodes going from node f_i to node f_j, 30
ρ_s	System robustness, 42
c_j	Cost, the punishment associated with a failed task, 40
f_j	Fault node, 29
k_1 **and** k_2	Normalizing constants for fault-tolerance rating for the HWB metric, 13
u_j	Utility award associated with every successfully completed task, 40

Characteristic set(ψ)	Characteristic set is a collection of all the symptoms correlating to the fault nodes from the causal model, 30
CMM	Causal Model Method for fault diagnosis, v
E	Edge connecting nodes x and y in the causal model, 28
Effective fault tolerance	Influence of fault tolerance towards overall system performance, 3
f	The fault-tolerance rating for HWB metric, 13
Generate-and-test methodology	Series of simple verification tests, 18
LeaF	Learning based Fault diagnosis for multi-robot teams, v
POC (σ)	Probability of occurrence or the frequency with which a fault is seen in a system, 28
SDR	The DARPA Software for Distributed Robotics project, 3

Chapter 1

Introduction

1.1 Introduction

In recent years mobile robotics has carved a niche in various complex military and civilian applications such as urban search and rescue, and future combat systems. The most common use of heterogeneous multi-robot teams is to achieve functionally-distributed missions, in which the mission tasks require a variety of capabilities not possessed by any single robot team member. In these applications, team members must decide which robot should perform which task, based upon the unique capabilities of each robot. These applications require complex coordination among multiple robots performing multiple tasks such as planning, mapping, localization, formation-keeping, information sharing, and so forth. For multi-robot systems to be commercially and practically viable, the system should be efficient and robust. The operating environments for such applications are highly dynamic and possibly hazardous. The dynamic nature of these environments leads to a high likelihood of component fault. Despite extensive system design and testing, breakdowns and faults often develop during the course of regular action. A fault can cause the robot(s) to lose functionality, which in turn may lead to a drop in the overall performance of the system. In extreme cases, faults can lead the robot towards incorrect actions or dangerous situations [Carlson and Murphy, 2003]. Fault diagnosis is defined as the ability of the system to detect and identify faults that occur during the course of operation, whereas fault recovery refers to the ability of a system to recover from these diagnosed faults. Any system that has the capability to diagnose and recover from faults is considered to be a fault-tolerant system.

Unfortunately, fault diagnosis for large teams of robots is an extremely difficult problem due to the large number of system components that must operate properly to successfully accomplish the team's mission. Faults are extremely common and can stem from a number of sources: malfunctions, mis-calibration, mis-coordination and/or software errors. In addition to these, any unexpected changes in the operating environment of the robots can directly or indirectly have a detrimental effect on overall system performance — for example changing lighting conditions can wreak havoc with the best vision algorithms, and clutter in a key area of the environment, such as a doorway, may cause traffic deadlock and lead to one area of the environment being inaccessible. Physical sensor faults (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 fault due to an environmental condition or some other cause, such as a sensor anomaly.

Due to the significant number of possible failure modes in a multi-robot domain, it is useful to develop a general solution for fault-tolerance that can be applied to a new problem (new robot and/or task) without extensive modifications. Murphy and Carlson [Carlson and Murphy, 2004] identify four important factors in the success of a diagnostic system for mobile robots:

- Efficiency — ability of the system to best utilize the available resources,
- Dealing with uncertainty — ability of the system to adapt to the changes in the operating environment,
- Robustness to noise — ability of the system to identify and recover from faults,
- Dealing with sparse information — ability to extract and integrate useful system information during the course of task execution, without the need for a large number of examples or a long training time.

Existing methods for fault diagnosis are mostly application-driven and are incomplete when applied to a larger more dynamic domain like the ones in which multi-robots operate. A key current limitation of existing architectures is their inability to incorporate sparse information towards improving system performance. Another drawback of most robotic fault diagnostic architectures is that they are primarily designed for identifying single robot sensing failures. One research area, related to the field of robotics, where a significant amount of work on fault-tolerance for teams has been done is that of multi-agent systems. The techniques designed for multi-agent systems offer an intriguing possibility when translated towards multi-robot domains. The latter sections of this thesis detail the work regarding one such strategy: the Causal Model Methodology (CMM) [Horling et al., 2000b]. This method is studied in detail when applied to an illustrative multi-robot test application. Subsequent chapters evaluate the technique and identify the potential stumbling blocks of this technique.

As applications and environments vary, the lack of a standard metric for evaluating the fault-tolerance of a system forces the designers to develop application-specific fault diagnosis instead of being able to develop a more generic approach. Promising methods such as *effective measure* [Hamilton et al., 1996] tend to focus on fault-tolerance based on redundancy, thereby not providing a true measure of learning and fault-tolerance in the system. Therefore, there is a need for a general metric for measuring the extent of fault-tolerance of a system.

In this thesis, we outline an application independent autonomous learning-based approach for multi-robot fault diagnosis. In addition to developing a generic fault-diagnostic system, we attempt to identify the influence of fault-tolerance towards achieving overall system goals of efficiency and robustness. Towards that purpose, we develop metrics to measure the system performance and fault-tolerance exhibited by the system. The developed metrics are designed to be application-independent and can be applied to any fault-diagnostic architecture. In the subsequent sections, we detail the scope of the problem that we address and outline our approach to solve it.

1.2 Problem and proposed approach

The primary objective of our research is to outline a framework for developing a robust turn-key solution for fault diagnosis in complex teams of heterogeneous mobile robots. The generic approach would enable the robot team to autonomously detect and compensate for the wide variety of fault modes that could be experienced in an indoor environment. The high level goal of this research is to design a system capable of autonomously adapting itself based on the information obtained from encountered faults, thereby improving system robustness over time.

Identifying, classifying and analyzing the list of possible faults that might be encountered by the system gives us a good starting point about when, what, and where to apply diagnosis. However, despite extensive detailing, post-experimental analysis on a sample test application will nearly always identify some previously faults. For a robot team, the faults that occur during the course of operation provide an opportunity for the system to learn about its environment, thereby adapting to it. Based on this key principal, we design our fault diagnostic architecture called LeaF. LeaF uses a partial causal model for representing the various faults in the system. The usage of a partial causal model reduces the search space and decreases the overall size and complexity of the model, making it scalable for large teams. When an *unknown* fault is encountered, the system uses a case-based learning approach to adapt and categorize it and subsequently add it to the causal model for future use. The ability of the system to effectively learn from its own faults makes it a more robust and efficient fault-tolerance architecture than prior approaches.

As the influence of fault-tolerance towards overall performance determines the effectiveness of the system, it is useful to have a metric that can measure the effective fault-tolerance in achieving the tasks of the application. Based on this concept of effective fault tolerance, this thesis addresses the following problem: Develop metrics for calculating the influence of fault-tolerance towards system performance and identify potential methods for analyzing the obtained measures towards evaluating the true capability of a multi-robot system.

1.3 Motivating application

The motivating test application we are using to develop our approach is a prior research task that we performed as part of the DARPA/SDR (Software for Distributed Robotics) project[*], involving a large number (up to 80) of physical heterogeneous robots. This project defined an indoor locate-and-protect mission [Howard et al., 2006]. The robot team had a very strict set of goals: to autonomously explore and map a single story in a large indoor environment such as the one shown in Figure 1.1, detect a valued object, deploy a sensor network and use the network to track intruders within the building. Cost and power considerations dictated that the bulk of these robots should be relatively simple; i.e., they have no sensors for localization or obstacle avoidance, and minimal sensing for robot kin recognition (using a crude camera). The robots must, however, retain the ability to make maps of the environment, detect valued objects, and navigate safely — tasks that would

[*]The DARPA/SDR project was accomplished by researchers in University of Tennessee's Distributed Intelligence Laboratory from 2002–2004, in collaboration with researchers from the University of Southern California and SAIC (McLean, VA).

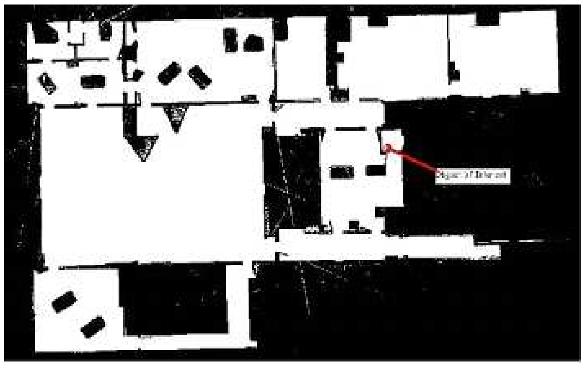


Figure 1.1: Test environment for the heterogeneous multi-robot application. The environment is approximately 45 by 25 meters in size, with an internal area of about 600 square meters.

normally require the use of expensive ranging sensors such as scanning laser range-finders or stereo vision. To solve this apparent conundrum, a heterogeneous approach [Parker et al., 2003] using complex behavior coordination (see Figure 1.2) was designed, utilizing three distinct classes of robots with varying capabilities.

The final composition of the team shown in Figure 1.3 consisted of three classes of robots: four (4) mapping robots equipped with scanning laser range-finders and a unique fiducial; three (3) helper robots equipped with scanning laser range-finders and cameras; and a large number (approximately 70) of simple sensor net robots equipped with a microphone and a crude camera. All of the robots had 802.11 WiFi, and a modified ad-hoc routing package (AODV) was used to ensure network connectivity.

With this heterogeneous team, the locate-and-protect mission was divided into two distinct phases: exploration and mapping, and deployment and detection. For the first phase, the mapping sub-team explored the environment, built an occupancy grid and located the valued object. Exploration was coordinated, and mutual observations were used to solve difficult correspondence problems (i.e., the mapping robots observed one other, and thereby identify loops in the environment). The second stage involved moving the simple mobile robots into deployment positions that were optimal for serving as a sensor network. Because these sensor-limited robots could not navigate safely on their own, we used complex heterogeneous teaming behaviors that allowed the small group of helper robots to deploy the simple robots (typically, 1–4 of these simple robots at a time) to their planned destinations using a combination of robot chaining and vision-based marker detection for autonomous tele-operation [Parker et al., 2003].

The unpredictable nature of the operating environment of the test application provides an ideal basis for testing our fault diagnostic system. As discussed earlier, the probability of encountering faults is high for a highly complex system such as that described here. For the current research described in this thesis, I attempt to recreate a similar scenario as the test application, using the available robots — Pioneers and Amigobots (see Figure 1.4) — from the University of Tennessee's Distributed Intelligence Laboratory (DI Lab). The exact details of the application are described in detail in subsequent chapters.

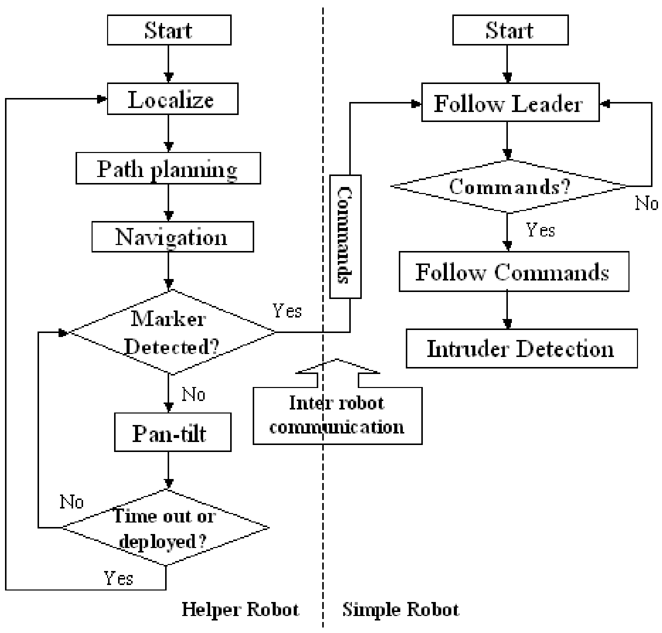


Figure 1.2: Cooperative behaviors that implement the heterogeneous deployment task.

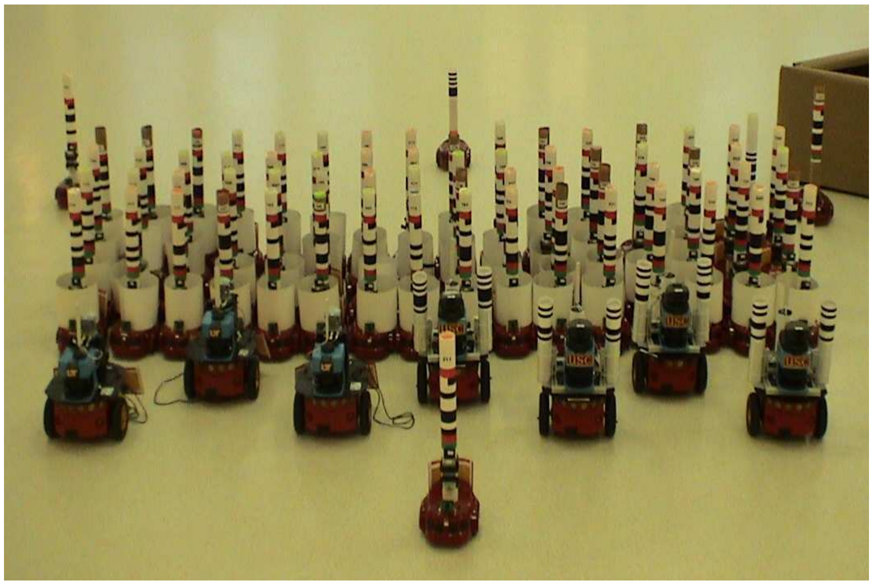


Figure 1.3: The heterogeneous robot team — mapper, helper and simple robots.

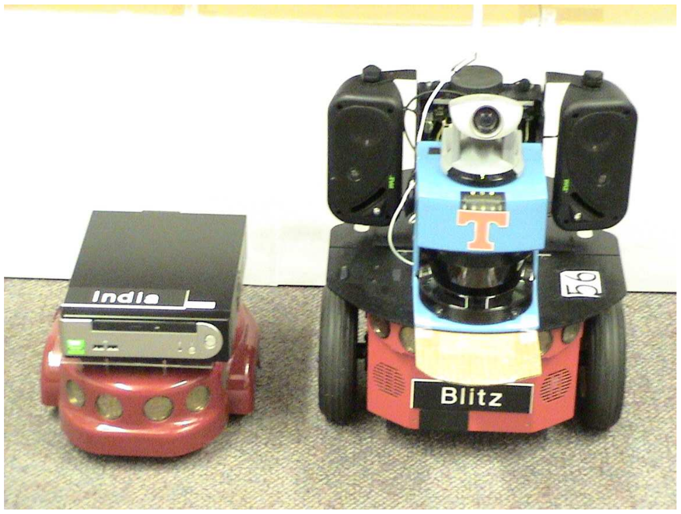


Figure 1.4: The two types of robots from the DI Lab used in this research — Pioneer (right) and Amigobot (left).

1.4 Contributions

The main contributions of this research to the multi-robot community are two-fold. First, this research designs and develops an application-independent, learning-based adaptive architecture for fault diagnosis and recovery that can be easily incorporated into a wide variety of existing systems to improve system performance over time. The unique feature of this system is its ability to learn from the encountered faults towards future fault tolerance. Second, the research introduces a new set of evaluation metrics to measure the extent of fault-tolerance towards system improvement over a period of time. In addition, by fusing existing methodologies from the fields of agents and physical robots towards a more complete fault diagnostic system, the aim is to bridge the gap between the two communities, providing a new and promising direction for future research in fault tolerance for autonomous teams.

1.5 Outline

The rest of this thesis is organized as follows. Chapter 2 discusses the previous work relevant to the issue of fault diagnosis. Chapter 3 details the preliminary work done in analyzing existing techniques for a sample multi-robot application and identifies the drawbacks of the existing systems. In Chapter 4, we describe our proposed solution LeaF. The architectural details of LeaF are provided along with preliminary experimental data validating it. Chapter 5 formally details the derivation of the proposed metrics. Chapter 6 presents the results from implementing LeaF and CMM for the physical robot experiments and applies the developed metrics to the obtained results towards performance analysis. Finally, we conclude with a summary of the contributions of this research to the multi-robot community.

Chapter 2

Related Work

Toyama and Hager [Toyoma and Hager, 1997] identify robustness as a key challenge in the face of uncertainty in complex, dynamic environments for intelligent agents. One major drawback of most current methods is that the programmer is responsible for accounting for all possible failures during the design stage. Atkins [Atkins et al., 1997] describes the inherent explosion of state space complexity in such dynamic environments that inhibits the ability of any designer to specify the correct response in each possible state in advance. Despite thorough consideration, it is not realistic to expect that all such failure modes will be accurately predicted. Several areas of related work apply to this research, including fault diagnosis [Cavallaro and Walker, 1994, Mahdavi and Bentley, 2003, Broxvall et al., 2004, Pell et al., 1997, Beetz and McDermott, 1997, Muscettola et al., 1997, Long et al., 2003, Lamine and Kabanza, 2000], robot assistance [Hamilton et al., 2001], learning [Visinsky, 1991, Gambardella and Versino, 1994, Mahadevan and Connell, 1991, Torras, 1994, Wermter et al., 2004], etc. In this chapter, we focus specifically on current and existing work that provides the background material for our research.

2.1 Fault diagnosis

According to a NASA survey conducted by Cavallaro and Walker [Cavallaro and Walker, 1994], the reliability and fault-tolerance efforts in robotics are application-specific. Unlike other domains such as engineering there do not exist any standards and protocols for implementing fault-tolerance. Most of the work in the area of fault-tolerance in robotics is aimed towards industrial robotics. Various techniques based on repeated testing for error recovery have been proposed and successfully demonstrated [Mahdavi and Bentley, 2003]. Bongard and Lipson [Bongard and Lipson, 2004] demonstrated an approach for automatically providing a diagnosis and recovery for unanticipated failure, specifically the recovery of locomotion of a severely damaged leg on quadrupedal and hexapedal robots. According to Carlson and Murphy [Carlson and Murphy, 2003], the mobile robot technology available today is often hand-built, making it nearly impossible for any designer to duplicate. As a consequence, diagnosis information on mobile robots is likely to be limited.

The experimental nature of the equipment and its uncertain interactions with the environment make detailed modeling for multiple failures, like that used in industrial robotics, difficult in multi-robot teams. This is the reason most researchers focus on single instance fault diagnosis systems for mobile robots [Muscettola et al., 1997] rather than multiple instance failures, namely [Long et al., 2003], [Lamine and Kabanza, 2000]. To better understand current work on fault tolerance for multi-robot systems, we classify it into two distinct yet related areas of research — multi-agent-based and robot-based. Multi-agent based systems integrated with distributed computing are becoming an essential part of AI programming. Such systems are designed as a possible replacement for existing systems such as monitoring, continuous diagnostic, process control, and decision support. A group of agents can be mapped to one or more systems to perform complex tasks including fault-tolerance. In the case of robot-based fault-tolerance, diagnosis is aimed at building a reliable, highly autonomous system consisting of a team of physical robots that operate in rich, uncertain environments. This necessitates robots that can reason about uncertainty, and diagnose and recover from unanticipated errors.

Fault-tolerance in multi-agent systems

Typically multi-agent tasks include modern transportation, industrial, agricultural, and fishing related tasks that can be performed by a group of autonomous agents. These tasks are parallelized with small amounts of coordinating communication at either the start or at the end. Some of the more popular examples include the ignorant swarms of Mataric [Mataric, 1993] and the swarms of Dudek et al. [Dudek et al., 1993]. Much of the work is geared towards equipping agents with functions to map the environmental conditions to coordinated actions or using learning to predict the future actions of other agents. Kaminka and Tambe [Kaminka and Tambe, 1998] present an approach for monitoring and diagnosis for multi-agent domains called SAM — Socially Attentive Monitoring. SAM uses social psychology-based fault detection, in which an agent utilizes other agents as a source of information for detecting failures. Social diagnosis is performed by reasoning about failures using an explicit model of teamwork and uses model sharing to alleviate inefficiencies in model representation.

A similar approach, called the CMM (causal model method [Horling et al., 2000b]), is a type of model-based parameter tuning approach for fault diagnosis that predefines a decision graph for detecting and diagnosing problems that occur during system operation. The ability of a model-based system to adapt depends on detecting failed expectations, and determining their cause so they may be corrected. To achieve this, CMM was designed to incorporate domain and coordination independent diagnosis capabilities into the individual agents within the multi-agent system. The initial design of CMM was specifically aimed to address performance issues in situation-specific coordination strategies. In this method, the strategy used for agent coordination must be tailored to meet the specifics of the current environment and the coordination situations an agent will encounter. Of all the approaches that we have seen, CMM appears to be the most feasible approach for implementation in a large multi-robot team environment. Unfortunately, when CMM is applied to a large test multi-robot application, certain significant shortcomings, like its inability to handle unexpected faults and to incorporate sparse information, are identified. On the other hand, the analysis of the CMM method led to an interesting observation: the partial causal model used by CMM for fault diagnosis provides us with a building block for systematically representing system faults towards designing a more robust system capable of handling unanticipated failures. We discuss the CMM model along with its shortcomings in more detail in Chapter 3.

Robot-based fault-tolerance

The literature on distributed fault recovery for autonomous robotics is limited. Despite the extensive work on fault detection and diagnosis, Long, et al., [Long et al., 2003] claim that the issue of how to handle sensing failures is largely un-investigated, and work on distributed sensing recovery towards improving system performance is scarce. ROSEY [Le, 2002] is an example of a fault diagnostic robot that is capable of tracing a causal chain from initial state to conclusion. Unfortunately, these methods are designed to be self-explanatory systems as opposed to a fast and efficient fault diagnostic approach for improved system performance.

A common approach to fault diagnosis is to endow the system with the ability to use knowledge-based techniques to reason about the state of the execution, detect anomalies, and to autonomously generate a recovery plan. A failure in perception, including the inability to acquire the perceptual data needed to perform the desired actions, often results in a robot unable to execute its goals. In this context, Murphy and Hershberger [Murphy and Hershberger, 1996] have suggested a two-step approach: a strategy for classifying sensor failures, and a recovery strategy. The sensor fusion effects architecture (SFX-EH) [Murphy and Hershberger, 1999] for handling sensing failures in autonomous mobile robots is based on this two-step methodology. SFX-EH is robust, efficient and to an extent is able to handle uncertainty. It uses extensions to the generate-and-test method to classify failures based on 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. SFX-EH uses the type of failure to determine the appropriate recovery strategy. The error handling approach presented is concerned primarily with a single autonomous robot based error recovery, with implicit fault detection. The hypotheses library used to identify faults does not easily extend to a multi-robot domain because of the inherently dynamic nature of the operating environment. Despite recent work by Long, et al. [Long et al., 2003], which looks at extending the SFX-EH architecture from a single robot to a small team of distributed robots, the architecture was primarily designed for single robot sensing fault diagnosis. Our studies involving post-experimental analysis of fault data for a sample test application, which is detailed in subsequent chapters, bring to light a type of fault that is specific to the multi-robot domain. This type of fault — a *coordination fault* — occurs due to the interaction of system components, even though individual sensors and behaviors are performing properly. SFX-EH, in its present form, is not designed to handle such faults that are specific to the multi-robot domain. Another drawback of the existing systems is their inability to incorporate sparse information towards improving system performance. Even with these shortcomings, the fault representation of the SFX-EH provides us with another useful building block for designing a more robust fault diagnosis system for multi-robot teams.

2.2 Learning in robotics

As we mentioned in the previous sections, the inherent nature of the operating environment dictates the system ability to dynamically adapt in order to accurately diagnose existing and new faults. The capacity of a system to learn from experience, analytical observation, and other means, results in increased efficiency and effectiveness. Currently, learning is used as a method for bounded-rational agents/robots to adapt to an optimal strategy in a situation with incomplete information.

Reinforcement learning [Sutton, 1998] is one of the most widely used online learning techniques in robotics. With an online approach, the robot learns during action and acts during learning, which is neglected in supervised learning methods. With reinforcement learning, the learner perceives the state of its environment (or conditions at high level), and based on a predefined criterion chooses an action (or behavior). Mataric [Mataric, 1994] describes a methodology for automatically combining basic robotic behaviors into higher-level ones, through unsupervised reinforcement learning based on the agents' interactions with the environment. To make learning possible, a reformulation was proposed such that it elevates the level of system description from states and actions to conditions and behaviors. A popular form of reinforcement learning used is the one-step Q-learning [Watkins, 1989] algorithm in which the external world is modeled as a Markov decision process with discrete finite-time states.

The methods that implement learning for fault-diagnosis, such as Nolan and Madden's IFT [Madden and Nolan, 1999], depend heavily on models or classified raw data. Other work by Madden includes fault diagnosis based on the natural hierarchy of components and sub-components in electrical and/or mechanical systems [Madden, 1998]. Wang and Dai [Wang and Dai, 1999] use a control theoretic architecture with each model having its own learning-based diagnosis agent. The advantages of decomposing a complex learning task into smaller ones include formulating a complicated learning task as a series of smaller ones thereby reducing the extra information which the learning system may use. This leads to an improved performance of the system as smaller learning tasks typically have simpler, more fundamental concepts associated with them that can be learned easily. Based on the above concept, [Madden, 1998] describes a novel approach for systematically performing a set of machine learning tasks in order to build a monitoring system with a hierarchical structure which corresponds to that of the machinery being monitored. Successful multi-robot systems like ALLIANCE [Parker, 2000] use learning to improve the efficiency of the team's performance. The extended architecture, L-ALLIANCE, monitors the performance of the robot team members to achieve learning and to take advantage of the fault-tolerance capabilities of ALLIANCE to enable task-oriented solutions.

Most traditional methods of learning, like the ones we have seen, require a large amount of training data. Unfortunately, for the application domain that we are interested in, it may not be possible for the robot to obtain large amounts of *a priori* training data. Incorporating new information regarding faults may involve rebuilding the entire rule-base. This could have a detrimental effect towards system performance in terms of complexity and execution time.

2.2.1 Case-based reasoning

A more recent learning technique, called case-based reasoning [Kolodner, 1993], appears better suited to match the domain criteria. In this method, no explicit description of the target concept is learned with the training data; the system simply stores training examples for future use. Each case typically contains a description of the problem, plus a solution and/or the outcome. The system does not have a pre-defined model based on the training data and performs the computation during prediction time. This is different from other conventional learning methods which rely on a set of classical rules. When a query is raised, the learner searches the database for similar data points and builds an on-line local model. By not forming an *a priori* model of the data, the system can dynamically adapt to the environment. When a test example is given, the case-based reasoner tries to find the closest match in the training examples. Based on the small set of training examples correlating to the faults identified by the designer, we may be able to handle newer unexpected faults by using distance functions like k-nearest locally weighted regression [Atkins et al., 1997] to extrapolate data corresponding to the current fault to the training data.

Armengol et al. [Armengol and Plaza, 2001a] introduced a method called LID — Lazy Induction of Deductions — in which solutions are obtained by explanations derived from symbolic descriptions of the similar aspects between the current problem and existing training data. In this method, the system uses the notion of *structural similarity* [Borner et al., 1996] to identify similarity between the two cases. Furthermore, the concept of *anti-unification* [Plotkin, 1970] is then used to formally define the structural similarity. The resultant *similitude* term contains only the relevant cases based on the outlined criteria. The authors have successfully implemented LID for a biologically-inspired classification application for a team of agents [Plaza et al., 1996, Armengol and Plaza, 2005]. We explore the LID method in detail in Chapter 4.

2.2.2 Metrics

In the last decade, several researchers have studied fault-tolerance for robotic systems (e.g., [Mahdavi and Bentley, 2003, Bongard and Lipson, 2004, Mataric, 1993, Dudek et al., 1993, Kaminka and Tambe, 1998, Horling et al., 2000a, Murphy and Hershberger, 1996, Long et al., 2003]). However, still missing from this research are standard metrics for evaluating new and existing multi-robot fault-tolerance methods. In the absence of an accepted metric, it is difficult for a designer to calculate the true measure of a system capability. This is especially true when attempting to compare two different fault-tolerant strategies, and determining which strategy can achieve the best performance.

The concept of metrics for quantifying performance is not new. In 1994, Cavallaro and Walker [Cavallaro and Walker, 1994] recognized the lack of standard metrics and discussed the applicability of protocols based on NASA and military standards. Evans and Messina in [Evans and Messina, 2000] analyze the importance of defining universally accepted performance metrics for intelligent systems. The analysis outlines current efforts to develop standardized testing and evaluation strategies and argues the need for industry accepted metrics for inter-comparison of results and to avoid duplication of work. Extending the analysis of Evans and Messina, Pouchard in [Pouchard, 2000] explores metrics specific to the software agent perspective. Both sets of authors extend a challenge to the research community to actively work towards the process of developing standard metrics.

Traditional engineering methods that address fault tolerance predominantly deal with reliability analysis of systems and components. *Reliability* is defined as the probability with which a system will perform its specified function/task without failure under stated environmental conditions over a required lifetime. Stancliff et al., [Stancliff et al., 2006] present a quantitative analysis supporting the argument that larger teams of less-reliable robots perform certain missions more reliably than smaller teams of more-reliable robots. Based on the concept of reliability, Carlson and Murphy extensively analyze failure data for mobile robots in [Carlson and Murphy, 2005]. Using MTBF (Mean Time Between Failures) as a representation for average time to the next failure, *reliability* for mobile robots is calculated. The MTBF metric is defined as:

$$MTBF = \frac{\text{No. of hours robot is in use}}{\text{No. of failures encountered}} \tag{2.1}$$

Other metrics used for evaluation include MTTR (Mean Time Taken to Repair) and *Availability*, which measures the impact of failure on an application or project. These metrics are defined as:

$$MTTR = \frac{\text{No. of hours spent repairing}}{\text{No. of repairs}} \tag{2.2}$$

$$Availability = \frac{\text{MTBF}}{\text{MTTR} + \text{MTBF}} \cdot 100\% \tag{2.3}$$

The resulting study illustrates that the reliability among mobile robots is low, with failures occurring at regular time intervals, mainly due to the operating platform. This study is very helpful in providing a detailed analysis of the component failure rate in mobile robots, and in highlighting the complexity of the operating environment as a significant determining factor for failures. However, it does not capture other types of fault tolerance that may be present in a system. It is also difficult to compare the merits of differing robot team control architectures purely using the standard manufacturing metrics of MTBF and MTTR.

In our work on metrics, we want to capture the notion of reasoning and intelligence as it affects the fault tolerance of a system. As our earlier work shows [Parker et al., 2004], [Parker and Kannan, 2006], ultimately, multi-robot systems should be able to intelligently handle failures, and thus improve over time. Hence, it is important for any performance metric for a multi-robot system to measure the extent of intelligence exhibited by the system. Recently, there has been a renewed interest in exploring the problem of metrics for intelligent systems. Lee et al. [Lee et al., 2000], propose an engineering based approach for measuring system intelligence. In this method, learning is used to theoretically measure system intelligence through a formal analysis of system software architecture and hardware configurations. Other related works include Yavnai's [Yavnai, 2000] approach for measuring *autonomy* for intelligent systems and Finkelstein's Analytical Hierarchy Process (AHP [Finkelstein, 2000]) for measuring system intelligence.

Unfortunately, existing work does not apply or extend these measures to help evaluate system fault-tolerance. In fact, relatively little research has addressed the issue of metrics specific to the field of fault-tolerance in multi-robot teams. Most existing architectures are evaluated purely based on task-specific or architecture-specific quantities [Parker, 2001]. The consequences of such an evaluation are that the general characteristics of fault-tolerance, robustness, and so forth, are not explicitly identified, and instead are hidden in the application-specific measures.

The most promising work related to our objectives is the work of Hamilton, et al. [Hamilton et al., 1996]. Their approach outlines a metric (which we call the HWB metric) for calculating "effective" fault-tolerance for single robot manipulators by combining the observed fault-tolerance with a performance/cost rating. The measure has two distinct terms: the first is based on a fault-tolerance rating and the second term is derived from a performance/cost value, as follows:

$$HWB_{eff} = k_1(f)^2 + k_2(p)^2 \tag{2.4}$$

where HWB_{eff} is the calculated measure, f is the fault-tolerance rating, $p$is the performance/cost rating, and k_1 and k_2 are normalizing constants. Here, fault-tolerance is calculated as $f = m/n$, where m is number of tolerable subsystem failures and n is number of available subsystems. The performance/cost rating is given by $p = (S + R + C)/3$, where S is performance speed, R is recovery time rating, and C is the cost measure. The authors evaluated their metrics on a number of multiprocessor control architectures.

This proposed metric has a few shortcomings that restrict its applicability for the multi-robot domain. First, the system calculates the effect of robustness purely based on redundancy, and thus does not capture the use of intelligence or reasoning to compensate for failure. Our research in the development of LeaF [Parker and Kannan, 2006], [Parker et al., 2004] identifies online learning from faults as an integral component of successful fault-tolerant systems. Hence, it is imperative for a evaluation strategy to quantify learning as part of the fault-tolerance measure. Also, as previously mentioned, most multi-robot systems are task-centric rather than robot-centric. Hence, it is easier to evaluate the system if the metrics focus on task performance. As part of our research, we extend the concept of "effective" evaluation of fault-tolerance to multi-robot systems. The newly proposed metrics are task-centric and include measures to identify system intelligence or learning.

2.3 Summary

In this chapter, work relevant to the primary concept of the thesis is presented. Current methods like CMM, SFX-EH and LID, though successful for fault diagnosis in specific applications, have limitations when applied to a larger more dynamic domain like multi-robot team based applications. On the other hand, all these methods provide fundamental building blocks for a more effective fault diagnostic architecture. We plan to adapt the LID learning technique for our problem and combine it with the existing CMM model based approach to enable the robot system to train itself based on the encountered faults.

Chapter 3

Investigating the Causal Model Method (CMM)

Existing architectures for fault-tolerance, such as SFX-EH and CMM, are robust, efficient and to an extent are able to handle uncertainty. The inherent explosion of state space complexity in multi-robot environments inhibits the ability of any designer to specify the correct response for every possible state in advance. Despite thorough consideration, it is not realistic to expect that all such failure modes will be accurately predicted. The faults occurring during the course of normal operation provide an opportunity for the system to learn about its operating environment, thereby reducing future occurrences of failures. A key current limitation of existing systems is their inability to incorporate sparse information towards improving system performance. Among the different techniques we analyzed, the CMM described by Horling, et al., [Horling et al., 2000b,Hudlická and Lesser, 1987] appears to be the most feasible one for multi-robot applications.

In this chapter, we analyze the applicability of the CMM approach to the complex multi-robot application scenario defined in Chapter 1. The CMM approach is a type of model-based parameter tuning approach for fault diagnosis that predefines a decision graph for detecting and diagnosing problems that occur during system operation. We decided to analyze CMM for our multi-robot test application for the following reasons:

- The CMM uses diagnosis as a means to adapt to changes,
- A causal model is designed to handle multiple failures,
- The diagnosis can be done in a domain-independent manner, and
- The approach described by Horling and Lesser [Horling et al., 2000b] consists of two distinct and independent parts – a framework for task analysis, environment modeling, and simulation called TAEMS [Decker, 1996] and a causal model that is used for the fault diagnostic process.

Because of the inherent distributed nature of the multi-robot application, it is advantageous to modularize the system. Specifically, the implemented system is broken down into two distinct parts — a standard behavior based model for task allocation and completion, and a fault diagnosis model for diagnosing and recovering from faults in the system. Thus, the two distinct parts of the approach in [Horling et al., 2000b] fit well with our application scenario.

In our earlier research, we have successfully implemented and demonstrated a distributed behavior based structure for the test scenario [Parker et al., 2004, Parker et al., 2003]. While the obtained test results were quite encouraging, there remained important issues that needed to be addressed, such as the influence of fault diagnosis towards overall system efficiency and robustness. In order to better understand the CMM for the multi-robot domain, we apply it to the test scenario from [Parker et al., 2003]. Here we restrict the analysis to focus exclusively on implementing the causal model based structure for fault diagnosis and recovery. In the subsequent sections, we provide a brief overview of the CMM approach. The analysis consisted of two distinct stages — the first stage involves the motivation and the design behind the Phase I causal model. In Phase I, a causal model for fault representation is designed. The design of the model depends solely on the designer's experience in the operating environment to predict faults in the system. The designer attempts to identify, *a priori*, all possible faults for the application. Based on the results of the testing of the application, we analyze the data collected from the experimental runs. The subsequent stage involves the development of the Phase II causal model. The Phase II causal model builds on the Phase I model using the post-experimental data. The newer model is more detailed than the first model and attempts to incorporate additional fault information not present in the Phase I model. Based on the experimental data, the designer attempts to recreate a new model containing all possible fault nodes. Finally we compare the differences in the models and their implications for generating turn-key solutions for multi-robot fault diagnosis.

3.1 CMM

The causal model as described by Horling and Lesser consists of two mutually exclusive modules — a goal/task decomposition language called the TAEMS and a structure to describe the pertinent assumptions for diagnosis based on the causal model. The model has an *a priori* base set of assumptions about behaviors and availability of resources. Fault detection is defined as the ability of the agent(s) to recognize when an assumption becomes invalid. Given this invalid assumption, fault diagnosis is defined as the ability of the system to identify the resource behavior or sensor that is responsible for the failure. Prior knowledge about the expected behavior provides a comparison monitor for the subsequent actions of the system. The knowledge can be of two types: information about the agent's expected behavior and methods for detecting deviation. Further, the model encodes the information/task using a domain independent goal/task decomposition language called TAEMS [Vincent et al., 2000]. TAEMS provides an explicit representation of goals and pathways to achieve them. Method behavior and interactions between other methods and resources are also represented by means of the TAEMS structure. TAEMS provides an organized knowledge structure to be used by the agent(s) for successful task completion. The initial model as designed by Bazzan, Lesser and Xuan focused on designing the diagnosis closely to the underlying system architecture. Based on the work of Sugawara and Lesser [Sugawara and Lesser, 1998], the subsequent model used a causal model to organize the diagnostic process in addition to the TAEMS structure.

The original CMM was defined as a directed, acyclic graph, written left to right, used for organizing a set of diagnosis nodes. The model makes use of the information available within the TAEMS structure to detect faults. Within the TAEMS structure, each node is defined by means of a set of characteristics defining the node and its relation to other nodes. In addition, some nodes in the model are marked as triggerable, meaning that they periodically perform simple comparison checks to determine if a fault may have occurred. The system then checks for deviations from the expected value of the characteristics and if a deviation is detected, a flag is set for that error. This in turn triggers the diagnosis model to identify the exact source of the fault. This trigger-checking activity is a primary mechanism for initiating the diagnostic process.

In order to apply the CMM to the multi-robot domain, certain modifications were made to the original model. These include:

- The CMM was modified to be an undirected graph. This enables the system to incorporate new information anywhere within the causal model.
- All nodes were made triggerable. Every node actively monitors for a variation from known behavior knowledge.
- Behavior knowledge is represented by means of characteristics that better describe the fault; e.g., symptoms associated with the fault, influence of the fault towards task completion, etc.
- The existing TAEMS model was combined with the generate-and-test methodology [Lindsay et al., 1980] from the SFX-EH architecture to help trace the fault.
- Each node has a corrective action associated with it. A corrective action consists of a set of instructions that the robot has to successfully execute to recover from the encountered fault.

Figure 3.1 illustrates a simple causal model designed for a sample multi-robot application. Each node in the graph corresponds to a particular fault diagnosis with the level of precision increasing from left to right. As the nodes produce diagnosis, the generate-and-test strategy is used to find more detailed levels of diagnosis to further categorize the problem. Using this methodology, the robot generates tests about possible causes for failure and uses the results of the tests as a means for further classifying the encountered fault. The extreme right nodes in the graph, called the leaf nodes, provide the highest precision or the lowest abstraction of diagnosis based on available information for a given fault. Generally, the leaf nodes correspond to sensor level diagnosis.

Multiple faults are handled by incorporating a single fault node that uses the diagnosis of several other nodes for verification. The causal model allows easy addition and removal of nodes by the designer depending on the specifics of the diagnosis required for the system. The designer is also free to make the nodes as domain independent as possible. By making the design as generic as possible, the model can be applied to each agent or robot with little modification. On the other hand, the design and the precision of the model is also dependent on the individual capabilities, the computation power, sensors, etc., of each individual or type of robot in a multi-robot team.

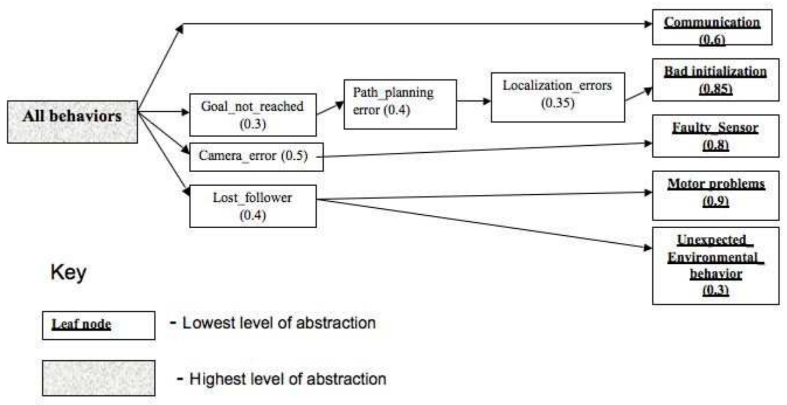

All behaviors
Goal_not_reached (0.3)
Path_planning error (0.4)
Localization_errors (0.35)
Camera_error (0.5)
Lost_follower (0.4)
Communication (0.6)
Bad initialization (0.85)
Faulty Sensor (0.8)
Motor problems (0.9)
Unexpected Environmental behavior (0.3)
Key
Leaf node - Lowest level of abstraction
- Highest level of abstraction

Figure 3.1: Phase I causal model for SDR – based on the failure recovery table from [Parker et al., 2004]

Therefore a system can have one or multiple causal models depending on the team composition. We explain the model in more detail in the following sections.

3.1.1 Phase I causal model

Using the above described CMM approach, we developed a Phase I causal model for fault diagnosis based upon our experience in testing and debugging the complex heterogeneous robot behaviors of the motivating example in Chapter 1. Figure 3.1 illustrates the implemented causal model for Phase I. Due to the computational constraints on the simpler sensor-limited robots, Phase I has one primary causal model implemented only on the helper robots. The comparison monitor used for fault detection was based on a simple case-based reasoning structure. A set of rules describing the fault characteristics were defined for the structure; in the event of a match, a flag was set, diagnosis performed, and a corrective action taken. Table A.1 (see appendix) lists the rules used to recognize the fault states predefined and implemented for this application. In addition, Table A.1 also lists the series of tests from the generate-and-test methodology used for improving the fault diagnosis. For our implementation, the characteristic set describing the fault was restricted to a single defining element, the symptom representing a fault. The symptom was derived from the system exception that was generated when a fault occurs. By comparing the symptoms of the encountered fault with the existing rule-base, we can identify the fault. Once the fault is identified, the robot, using the generate-and-test method, generates further tests about the cause of failure, eliminates redundancy and confounding effects, analyzes the test results, and attempts to determine as accurate a diagnosis as possible. The symptom-fault rule base along with the corresponding corrective action for the Phase I model is detailed in Table A.1.

As an example of this approach, consider the situation in which the simple sensor-limited robot has motor problems during deployment and stops. The helper robot uses the camera to periodically monitor the following sensor-limited robot during the course of deployment. The helper robot is also in constant communication with the simple robot, coordinating with the simple robot to maintain formation. Once a sufficient amount of time has elapsed and the helper robot realizes that the follower is not present behind it, an exception correlating to a *missing_robot* symptom is raised. The helper robot then attempts to coordinate with the simple robot and perform additional tests (see generate-and-test section of Table A.1) to better determine the possible fault mode. Additional tests identify no sensing or communication errors on either one of the two robots. Also during the course of testing, an *inconsistent map information* symptom is identified. Once it matches the symptom to the corresponding fault, *unexpected_environment_behavior*, as the primary cause for failure, it performs remedial actions which involves the helper attempting to activate the follower into acoustic sensor network detection mode (provided the follower robot is within an acceptable vicinity of its original goal and its other sensors are operational).

3.1.2 Experimental results

The application described in Chapter 1 was tested in the sample environment shown in Figure 1.1. The experiments consisted of repeated deployments of 1, 2, or 3 simple robots per team. The experiments were tightly controlled by a set of human evaluators who were not the system operators. Over the course of the experiment, various failures were encountered, some of which were expected and some that were unexpected. If a deployment failed on one experiment, the consequences of that failure were not corrected, except on rare occasions. Thus, the data reported here incorporates propagation of error from one experiment to the next. This is an important difference between single robot and multi-robot teams in terms of fault occurrence. In these experiments, a total of 61 simple robot deployments were attempted. The experimental data showed an overall deployment success rate of 60% - 90%, depending upon the environmental characteristics. In other words, for each attempt at deploying a simple robot, 60% - 90% of those robots successfully reached their planned deployment position. The reason for the low success rate is the complexity of the heterogeneous robot system. As discussed in Chapter 1, the system is composed of several non-trivial modules, including localization, path planning, navigation, helper following, visual marker detection, and inter-robot communication (refer to Table 3.1).

The completion of the entire deployment process depends upon the successful completion of all of the system modules while the robots are operating in cluttered environments along complex paths. Such an application provides us with a useful scenario for testing the causal model. Despite the limited number of failure states identified and implemented, the success rate of the helper robots making it back home autonomously in these rigorous experiments was 91% (over 45 trials). Table 3.1 depicts the probability of success of each individual module in this implementation and the overall system probability, based upon the experimental results. Each component/module is designed to be highly reliable, yet due to the overall interaction between the components, the overall system probability is only around 59%. In order to make each component highly reliable, each component must be able to identify, recognize and recover from errors.

Table 3.1: Overall System Success Rate, averaged over 45 trials

Localization	**Probability**	**Subsystem Success Rate**	**Experimental Success Rate**
Localization	p_1	0.83	
Path Planning	p_2	0.99 (est.)	
Navigation	p_3	0.95 (est.)	
Follow Helper	p_4	0.78 (est.)	
Marker Detection	p_5	0.98	
Communication	p_6	0.91	
Complete System	$p_1 \times p_2 \times p_3 \times$ $p_4 \times p_5 \times p_6$	0.54 (est.)	0.67 (2 robot depl.) 0.48 (1 robot depl.) 0.59 (combined over all trials)

3.1.3 Phase II causal model

The post-experimental analysis highlighted the shortcomings of the original fault diagnostic causal model. It also identified certain unexpected faults that the system was not equipped to handle. One such fault occurred in the communication protocol. The layout of the environment in Figure 1.1 from Chapter 1, was such that communication between the base station and the room in the upper right corner required the signal to pass outside the building and then back into the building, which affected the quality of the signal propagation. Also, the experiments were conducted during a snowstorm, which also likely interfered with WiFi signal propagation. This resulted in strong interference over the WiFi signals that were confined to that particular room alone. This, in turn, caused the robots to lose communication with the base and with other robots, leading to a failure in achieving its objective.

Based upon the lessons learned from testing, we developed a Phase II version of our causal model that incorporates the previously *unknown* faults that were identified during experimentation. Figure 3.2 illustrates this new Phase II causal model based on the modified symptom-fault rule base (see Table A.2). This model was not implemented on the robots as it was developed from post-experimental data analysis. However, it is instructional to step thorough this model to see how the robots could have used it for improved fault-tolerance. In the case of the above example, interference in communication would have triggered the *communication_error* node. Tracing through the Phase II model, communication failure could have been caused by one of the following three reasons:

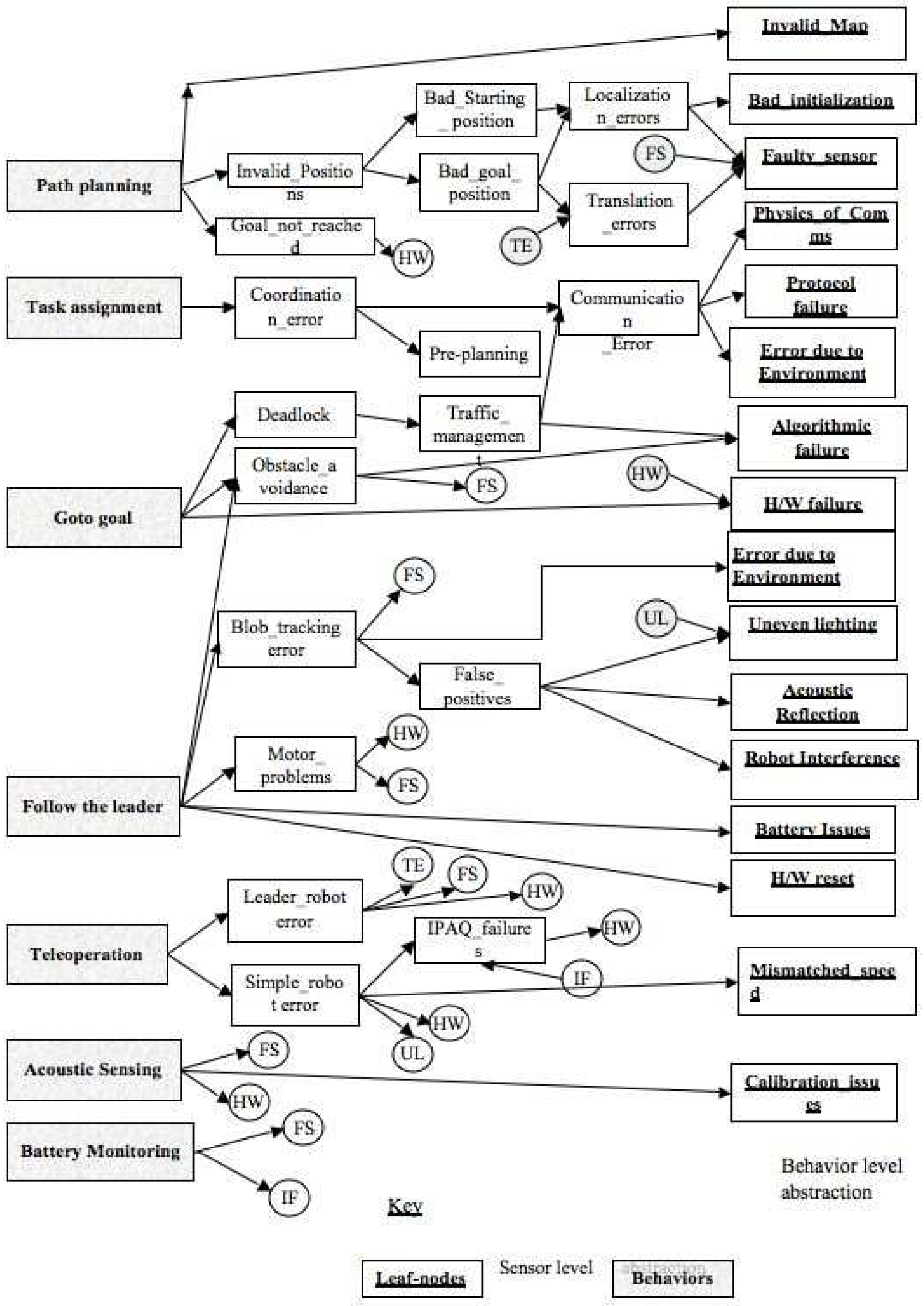


Figure 3.2: Post-Experimental, Phase II causal model for SDR — based on [Parker et al., 2004].

physics of communication, *protocol failure*, or *environmental conditions*. Once the potential failure node was identified, the robot could have performed a series of tests from the correlating generate-and-test strategy associated with the communication failure node. This would have included exploring the area, trying to re-establish communication with other robots. Since the problem was restricted to that particular room, the affected robot would have been able to establish communication with other robots as soon as it left that room. As mentioned in Chapter 1, the robots used communication as a means for coordinating observations. Cross-verification of the fault diagnosis information from other team members would have resulted in the robot eliminating physics of communication and communication protocol as a possible failure diagnosis. This would have left the robot to conclude that the loss in communication was due to an environmental anomaly. Comparing the two causal models in Figure 3.1 and Figure 3.2, we can conclude that it is unlikely that the designer will anticipate and incorporate every possible fault that the system may encounter. This is especially true in the case of large multi-robot team environments because of the inherent nature of the domain and the large number of behaviors and moving parts in the system.

3.2 Discussion and summary

The key current limitation of CMM and other such techniques is their inability to incorporate sparse information towards improving system performance. From the design, experiments, and post-experimental data analysis, we have formed the belief that multi-robot applications, such as the one we studied, are executed in environments with unbounded indeterminacy, meaning that it is impossible to enumerate all possible contingencies. We therefore hypothesize that the experimentation of a revised implementation that incorporates the newer Phase II causal model shown in Figure 3.2 would still result in the identification of additional faults that were not pre-defined in the causal model. For example, a minor change in the characteristics of the environment, such as a change in the inclination of the slope of the floor (which results in the camera mounted on the helper not being able to see the simple robot due to the non-planar paths of the two robots) could result in the designer having to build a completely new causal model. This is potentially an infinitely iterative process that can never be completely solved. The CMM model requires an enormous amount of fine-tuning by the designer for it to be truly robust in a specific environment. This prevents the system from ever truly becoming a turn-key solution for fault diagnosis. The model in its current stage is unable to adapt to different environments and the resulting faults. On the other hand, the causal model does provide a good basis for developing a more generic framework for fault diagnosis. The failures occurring during the course of normal operation provide an opportunity for the system to learn about its operating environments, thereby reducing future occurrences of failures.

Chapter 4

The LeaF Approach

The ability to learn from mistakes is a unique feature that enables humans to adapt to a wide variety of situations. Faults provide valuable information about the environment. Identifying and learning about the faults encountered can improve a system's performance. Our analysis of the causal model revealed that it cannot handle *unknown* faults that are not modeled *a priori* into the system. Further analysis revealed that a possible solution to overcome the existing drawbacks is to incorporate learning into the system to handle unexpected failures. Learning across the team members would allow the system to extend its causal model based on team experience. Hence, we focus our research on enabling the system to learn more about its working environment and as a consequence adapt to it. The specific objectives of learning include:

1. Using experience to refine the process of diagnosis,
2. Incorporating new edges in the causal model, based on information extracted from the faults, and
3. Enabling dynamic software reconfigurability based on information learned from other team members.

These capabilities would enable the system to handle situations such as bad communication in one area, bad lighting in one area, an environment too complex for navigation in one area, and a blocked passageway, without compromising the overall system goals. In addition, the system could use learning to incorporate the sparse information, obtained from the encountered faults, into the causal model. This information would be represented as new nodes and their relationship to other existing nodes. Adapting the causal model during experimentation eliminates the need for the designer to attempt to identify all possible faults *a priori.*

LeaF is designed as an adaptive method capable of using its experience to update and extend its causal model to enable the team, over time, to better recover from faults when they occur. A modified case-based learning algorithm is used to adapt and categorize a new fault and add it to the causal model for future use. In addition, the learner refines the current diagnosis process based on the frequency of occurrence of specific faults. The system consists of a hierarchical set of blocks or modules of predefined functionality, such as fault detection, diagnosis, recovery, learning, etc. The faults are detected and recovered from by each individual module with minimal communication with the other modules of the system; i.e., the decision making is distributed across each robot in the team based on its partial knowledge of the environment. This allows the system to be flexible in terms of modularity, fault-tolerance, and extendibility. More importantly, it eliminates the single point of failure that is associated with a centralized architecture. In addition, the design allows LeaF to be easily incorporated into most existing robot architectures without major modifications.

Our ultimate goal is to build a cross-architecture system capable of constantly learning from its own faults and those of other team members, making it a domain- and application-independent architecture. We believe that learning is the key characteristic needed to develop a turn-key solution for a fault diagnostic system for heterogeneous multi-robot teams performing complex tasks.

At a higher level, the entire process of fault representation and diagnosis can be viewed as a fully connected graph, with nodes representing faults and edges highlighting the relation between the faults. For such a system, the goal would be to autonomously learn the probability values associated with each edge that maximizes system performance for a specific application or environment.

4.1 LeaF Architecture

This section details the process of incorporating LeaF into a standard multi-robot application [Parker et al., 2004]. Figure 4.1 illustrates the combined architecture, where the upper portion depicts a basic behavior-based robot architecture while the bottom portion of the figure represents LeaF. The system consists of two main functional modules:

- *Behavior Control Module (BCM)* responsible for executing specific tasks or goals,
- *Fault-tolerance Module (FtM)* responsible for fault diagnosis and recovery.

Embedded within the fault-tolerance module is another important component — the case-based learner.

4.1.1 Flow of control

The robot starts out by receiving task information from the human operator and then proceeds to execute it. The task information is fed into the state machine which then determines the operating state based on the type of task.

State machine

The state machine consists of a set of states that the system operates in, and a transition function that maps current states to a next state. The set of states the system operates in are: **NORMAL**, **DIAGNOSIS** and **RECOVERY** in addition to a transition state called **WAIT** (see Figure 4.2). Any tasks, tests or recovery actions are fed through the state machine to streamline the control of sensors and effectors.

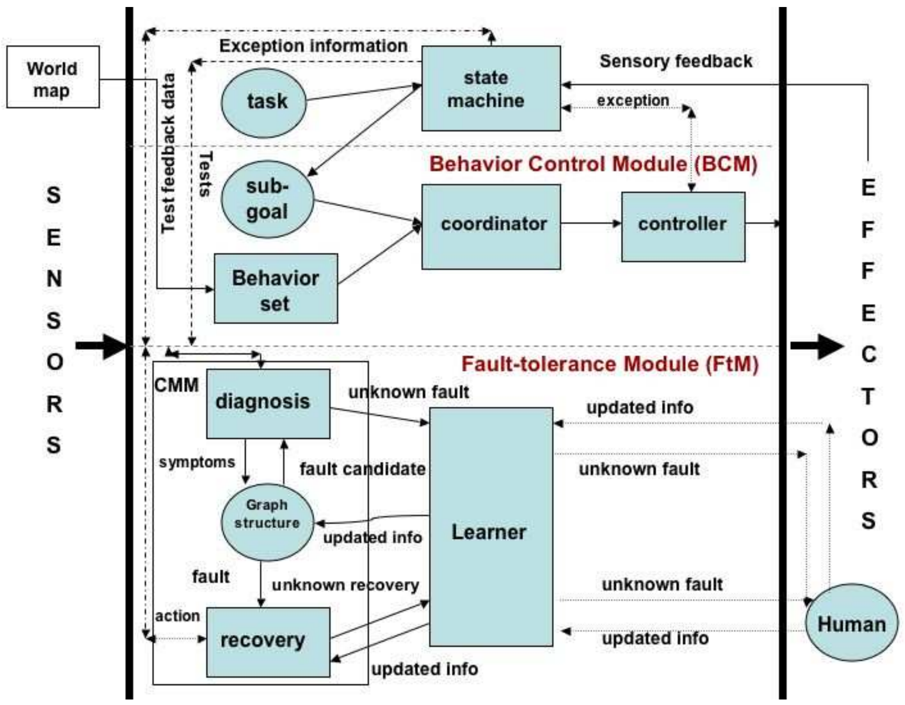


Figure 4.1: Architectural details of LeaF.

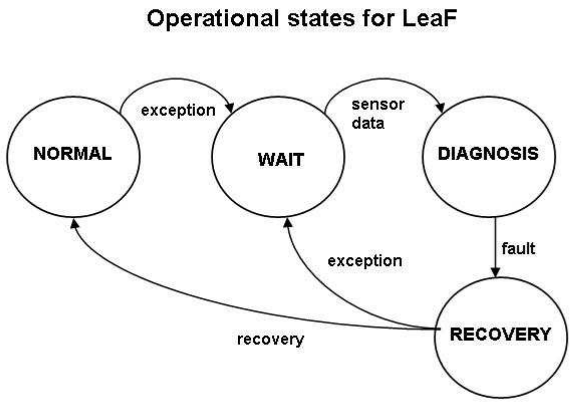


Figure 4.2: State diagram for modes of operation of LeaF.

The sensors feed back into the state machine, which determines the current operating state and transfers control to either the BCM or the Fault-tolerance module. The robot begins in the **NORMAL** state. In this state, the robot executes the assigned task based on the set of available behaviors, such as the ones in Figure 4.3 [Parker et al., 2004]. Once the state machine identifies the mode of operation and task, it passes the information to the BCM for execution. In the event of an unexpected variation in the behavior, an exception is raised. When an exception occurs, the BCM stops normal behavior execution, shifts the robot to a **WAIT** state and attempts to obtain all useful information regarding the problem, including information about sensors and behaviors. This information is represented by means of a top-down tree structure, with the level of abstraction decreasing from left to right. The information allows LeaF to identify new fault nodes that can be incorporated into the CMM. In addition, the BCM also saves the system's state at the time of exception. This is called the *safe point* of the system. All the collected information is then transferred to the state machine. Once the state machine receives all relevant information, it transitions the system to the **DIAGNOSIS** state. In this state, the robot tries to identify and classify the fault encountered. The FtM is responsible for fault identification, classification and subsequent recovery. Based on the gathered data, the diagnosis block of the FtM outlines a series of tests to identify potential failure nodes. To ensure a tight flow of control, the FtM itself does not execute the tests; instead it transfers the control back to the state machine. The state machine passes on the test to the BCM for execution. The feedback of the tests are filtered through the state machine before being sent back to the FtM. The diagnosis block compares the results of the tests with the behavior knowledge stored in the nodes of the CMM to identify the exact type of failure. The origin of exception helps restrict the search within the CMM. In the event of an *unknown* fault, the diagnosis block passes the information on to the Learner. The Learner attempts to diagnose the *unknown* fault based on a case-based learning algorithm. For certain difficult cases, it may be beneficial to request additional human input to short circuit the process. Once the fault is diagnosed, the control is transferred to the recovery block and the system transitions to the **RECOVERY** state. As both the diagnosis and the recovery modules fall under the FtM, control is transferred directly from one to another without having to pass through the state machine. This reduces redundancy and overall system complexity. The recovery block comes up with a corrective action for the diagnosed fault. A corrective action is defined as a sequence of steps that need to be executed successfully to facilitate the transition of the system from the **RECOVERY** state to the **NORMAL** state of operation. Once the recovery is successfully completed and the fault eliminated from the system, the system shifts back to the **NORMAL** state and transfers control to the BCM controller block. The module starts execution from the last *safe point* and the robot attempts to complete its assigned task. In the event the recovery mission fails to eliminate the error, the state machine temporarily shifts the robot into the **WAIT** state, gathers any relevant information and restarts the process of fault diagnosis. Figure 4.3 shows a similar state diagram implemented for the simple robots from the motivational example in Chapter 1.

4.1.2 Behavior Control Module (BCM)

As mentioned in detail in Chapter 1, complex cooperative behaviors are required for successful completion of the defined tasks (see Figure 1.2). Parker, et al. [Parker et al., 2003] implement one such method for the task of navigation-assistance. We believe that as the complexity of the application increases, so does the heterogeneity of the components involved. As behavior-based representations for robotics are parallel, distributed, and active, in order to accommodate the real-time demands of other parts of the system, they are ideally suited for the type of applications that we are interested in. However, more traditional control-theoretic approaches could also be used; LeaF does not require a particular type of robot control. In order for the architecture to be portable and modular, we separate the behavior-based control from the rest of the fault-tolerance module.

The behavior module consists of the coordinator and the controller. The controller contains the list of individual behaviors associated with the robot. Based on the task to be completed and the environment map, the coordinator arranges an order of execution for the behaviors. Currently the coordinator is hand-coded, but a future goal might be to incorporate a system such as ASyMTRe (Automated synthesis of multi-robot task solutions through software reconfiguration) [Tang and Parker, 2005] for automatically synthesizing task solutions. The controller translates high level commands into motor and sensory commands.

CMM

The CMM block contains the *Diagnosis* and *Recovery* sub-modules. As mentioned in Chapter 3, the causal model for LeaF is designed as an undirected graph, where the faults are represented as nodes of the graph [Horling et al., 2000b].

4.1.3 Fault-tolerance Module (FtM)

The fault-tolerance module combines the CMM with a case-based learner to help identify and subsequently recover from faults. The subsequent sections describe the individual sections of the FtM in detail.

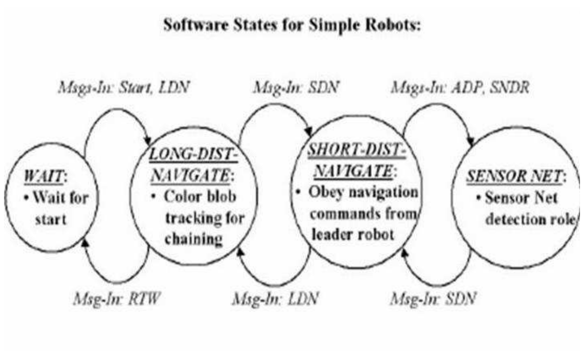


Figure 4.3: State diagram for normal operation of a simple sensor-limited robot.

The model has an *a priori* base set of assumptions about the faults and the symptoms associated with them (refer to Figure 3.1 and Table A.1). Once the fault is classified, recovery is straightforward. All nodes of the model have an action associated with them; when the diagnosis determines that the cause for a fault is one of the nodes, the corresponding corrective action is then selected and subsequently executed. The recovery strategy could vary from using an alternate logical sensor to graceful degradation depending on the type of fault diagnosed. The successful execution of the recovery strategy results in the robot switching back to the original task and operation. In the case of *unknown* faults the predefined causal model is inadequate for fault diagnosis and recovery. In such cases we use the learning algorithm to identify, diagnose and recover by extrapolating the fault type and associated action based on the existing sample space.

In most applications, some faults occur more commonly than others. One of the drawbacks of our CMM architecture is that it does not distinguish between diagnosis in terms of the frequency of occurrence. In order to better prioritize the faults, we associate a weight, called the probability of occurrence or POC (σ), with every edge of the CMM. For an encountered fault with an observed symptom, the σ for an edge E_1, connecting nodes x and y, is the probability with which edge E_1 is selected from current node x to the connected node y within the CMM and is denoted by $\sigma(x, y)$. The cumulative outgoing probability from any node sums to 1. The edges in the CMM are initialized by the designer. By allowing the designer to assign initialization values, the system takes advantage of prior experience of the designer's field experience. This reduces the need for an elaborate training stage. Alternately, the starting data set can be obtained by operating the robot in a training phase for a fixed period of time and calculating the probability of occurrence for the edges of the CMM. To improve the quality of the weighted value, the training data can be collected separately for individual team members. This ensures that robots weigh their individual CMM's specific to their functionality or task capability. Figure 4.4 illustrates the modified Figure 3.1 CMM having weighted nodes.

Learner

As described in the previous sections, previously *unknown* faults cannot be classified using the causal model. To categorize and recover from such types of errors, we propose to use a case-based learning [Aamodt and Plaza, 1994] approach to identify, categorize and add the new fault to the causal model for future use. As mentioned in Chapter 2, we want our multi-robot team to learn quickly and efficiently from the limited failures it encounters. Case-based reasoning (CBR) [Plaza and Arcos, 2002] is one such method that can be used to autonomously learn from experience. Using case-based reasoning, a new fault can be diagnosed by identifying its similarity to one or several previously classified faults stored in the causal model and by adapting known solutions instead of working out a new solution from scratch. As our work involves teams of robots performing individual tasks, we want to design the CBR in a decentralized fashion; that is, each robot has a copy of the causal model and the learner and attempts to identify the new fault by itself.

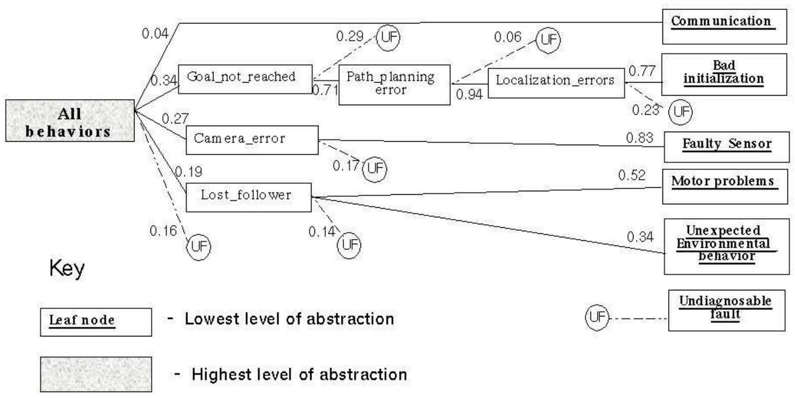


Figure 4.4: Modified causal model with assigned probability of occurrence.

In the event the robot's learner cannot classify the fault, the robot communicates the relevant information to a human operator for inference.

The identification of similarities between faults is done using a technique called Lazy Induction of Descriptions (LID [Armengol and Plaza, 2001a]). LID builds a symbolic similarity description, *similitude* [Armengol and Plaza, 2001b], between existing sample data and the encountered fault. Specifically, *similitude* is constructed by identifying prior cases that best share certain pre-defined characteristics with the encountered fault. Unlike other distance-based approaches for CBR, similitude measures the relevance between faults using relations among entities, rather than using arbitrary distance metrics. LID identifies relevance by only selecting the nodes with similar characteristics to that of the encountered fault. This approach provides a better reasoning for fault diagnosis and reduces overall system complexity and the time spent in fault diagnosis.

LID

We first describe the formal framework for LID summarizing [Armengol and Plaza, 2001a] and adapting it to our multi-robot domain. We then apply this technique to our sample team application, giving an example of how LID enables us to create an adaptive causal model.

For our domain, we define the problem space, PS, as the set of all possible fault descriptions that can be encountered by the system over the duration of the experiment. PS includes the faults that are identified by the designer before experimentation as well as those that are encountered during testing and experimentation. The solution set (S), pertaining to the problem space, is the set of all corrective actions for the corresponding fault from the problem set. Specifically,

- PS=$<V, E>$, where the vertices set $V = \{f_1, f_2, f_3, f_4, ..., f_m\}$ represents faults and the edges $E = \{\{f_1, f_2\}, \{f_1, f_3\}, \{f_2, f_3\}, ..., \{f_i, f_j\}\}$ represent relationships between the faults, according to the causal model.

- $S = \{s_1, s_2, s_3, s_4, ..., s_m\}$, where s_j is the corrective action associated with fault f_j.

Typically, fault relationships indicate increasing (or decreasing) order of specificity in the fault description.

- For a system consisting of n robots, the sample case base, C, is defined as:

  $C = \{C_1, C_2, C_3, ...C_n\}$, where $C_j = \{(PS_j, S_j)\}$ and PS_j is the problem space defined for robot R_j having a solution set of S_j.

For the LeaF system, PS_j represents all nodes and edges of the CMM for robot R_j and S_j constitutes the corrective action associated with the nodes. Also, a path $\pi(f_i, f_j)$ is defined as the sequence of inter-related nodes going from node f_i to node f_j in the CMM.

Interpreting the rule-base from Table A.1 and Figure 3.1 from Chapter 3, we can define the vertex and edge sets for our test application as follows:

- V = \{ *Goal_not_reached, Camera_error, Lost_follower, Path_planning_error, Localization_error, Communication, Bad_initialization, Faulty_sensor, Motor_problems, Unexpected_Environmental-behavior* \}

- *E*= \{\{ *Goal_not_reached, Path_planning_error*\}, \{*Path_planning_error, Localization_error*\}, \{*Camera_error, Faulty_sensor*\}, \{*Lost_follower, Motor_problems*\}, \{*Lost_follower, Unexpected_Environmental-behavior*\}, \{*Localization_error, Bad_initialization*\}\}

Also, a sample path from *Goal_not_reached* to *Bad_initialization* is given by:

- π(*Goal_not_reached, Bad_initialization*) = \{*Goal_not_reached, Path_planning_error, Localization_error, Bad_initialization*\}, and

As mentioned earlier, a fault is described by means of its characteristics. The collection of terms (ψ) defines the characteristics of the fault sorted in terms of their relevance. We define ψ as follows:

- $\psi = s[\psi_1, \psi_2, ...]$, where $\psi_1, \psi_2, ...$ are the fault characteristics and s is the application-specific sort function defined by the designer.

For LeaF, currently we restrict the characteristic defining the problem to a single term that represents the fault symptom.

Having introduced the above terms, we can then re-define *similitude* as finding nodes from the causal model that best describe the encountered fault based on the observed symptom. To classify an encountered fault, f_i, the adapted LID builds a similarity space, called D_f, such that:

- D_f contains only the most relevant characteristics of f_i.

- D_f induces a subset of the problem set that satisfies the description: $D_{sp} = \{f_1, f_2, ..., f_j, ... \mid \forall f_j \in V : \psi \in f_i \text{ and } \psi \in f_j\}$; D_{sp} is called the "discriminatory set" of D_f.

Hence, for LeaF, the similarity space is simply the set of all nodes in the causal model that have the same symptom as the encountered fault. To illustrate the concept of similitude, consider a scenario in which a helper robot encounters a fault, f_i, while teleoperating a simple robot. This fault has the symptom $\psi = \{$"inconsistent pose info"$\}$. Applying the above criteria to the Table A.1, the generated similitude term, D_f, is given by $\{Localization\_errors\}$.

Heuristic Assessment

In order for the robots to diagnose and recover from a fault, it is not enough to identify a set of similar cases. The solution should be similar to the problem in aspects that are important to the current task. In order to identify the relevance of the elements in the similarity set, LID uses a top-down strategy that determines which of the characteristics present in the fault are the most discriminatory. The process of using only the most discriminatory characteristics eliminates unrelated cases and narrows down the similarity space to contain only the most relevant cases to the encountered fault.

Unlike the original method, which builds a dynamic heuristic, the lack of prior knowledge of the distinct similarities between faults forces us to predetermine the most important characteristics for our causal model. As we are interested in diagnosing faults of varying abstraction, it is difficult to define generalized characteristics common to all of them. Fault symptom is one such characteristic that provides information that is common to all encountered faults. Even though a fault symptom by itself is not an exact indicator of what went wrong, it does provide the designer with enough information to identify the most likely event that could have gone wrong. The fault-symptom relation can be better described by means of the following:

- Every fault has a symptom associated with it,
- Symptoms are not unique to a fault, and
- There is a possibility existing symptoms may be an indicator for a previously *unknown* fault.

Future work involves analyzing the faults in the system to possibly design a more dynamic heuristic.

Once the characteristics are determined, they are sorted in terms of their relevance and are then used to build the similarity space, D_f. The initial similarity space is built by identifying and grouping all cases possessing the most relevant characteristic(s) to the encountered fault. Subsequent steps involve pruning the similarity space, based on the sorted characteristic set, until a single case that provides the most specific reasoning for the encountered fault can be identified. Algorithmically, this is determined by checking for one of two termination criteria, as detailed in Algorithm 1.

Algorithm 1 Terminating Criteria

```
1: if $\exists$ at least one leaf node ($f_l$) in the similarity space, $f_k \in D_f$ then
2:     STOP
3: else if Adding further characteristics of $f_i$ into $D_f$ does not produce a leaf node $f_l$ then
4:     Execute Algorithm 2
5:     STOP
```

When the termination occurs due to the second criterion it means that system is unable to build a structural similarity between the fault f_i and any of the leaf nodes of the causal model. Then, in order to prune the candidate cases down to just one most relevant case, we apply a diagnosis Algorithm (Algorithm 2) to D_f.

The diagnosis algorithm forms the core of the learner. The algorithm recursively applies the LID learning process to the causal model, reducing the search space until a node from the CMM that best describes the encountered problem, f_i, is selected. By selecting a leaf node, we attempt to identify a diagnosis with the lowest level of abstraction, thereby providing a more refined recovery strategy for the encountered fault. Unfortunately, the diagnosis may not always lead to the identification of a leaf node. For such cases, as the classification of the faults or nodes are done based on their structural similarities, it is reasonable to assume that if there exists a leaf node associated with the fault, it lies closer to the nodes described by the similarity set than the other nodes of the CMM. Based on this assumption, the diagnosis algorithm has the flexibility to expand the search space to include outlying nodes to the *similitude* term.

After initialization, the first step in the algorithm involves selecting the edge with the highest probability of occurrence for the observed symptom. Once the edge is identified, the next step involves replacing the current problem, f_i, with the selected node, f_j of set D_f. Once the problem is redefined, we recursively re-apply the LID strategy to the problem. This process is repeated until either a leaf node is eventually diagnosed or until all structurally similar nodes in the CMM are explored. If the system is unable to diagnose a leaf node, then the node from the similarity space having the highest path weight is selected as the initial diagnosis node.

Generally the recovery action associated with the selected leaf node solves the encountered fault and the system can then resume normal operations, but in some cases the recovery action does not eliminate the encountered fault. In such a scenario, the entire process of building the *similitude* term and applying the diagnosis algorithm is repeated with the last generated *similitude* term, from the previous iteration of the algorithm, acting as the initial *similitude* term for the latest application of the algorithm. Thus assuming a selected leaf node did not solve the problem, node f_q is added to the explored nodes list ($D_{explored}$) and Algorithm 2 is restarted with $D_{initial} = D_{final}$.

Also, it may be possible that the diagnosis corresponding to the encountered fault may be part of the CMM but does not have a connected edge from the initial diagnosis node. This would lead to selection of a non-leaf node as the diagnosis for the encountered fault. In order for LID to expand the similarity space to include other marginal cases from the CMM, the algorithm attempts to retrace a level of abstraction, i.e., explore a larger sub-tree of the entire CMM, and include it as part of the modified similarity space. The process of expanding the similarity space continues until a leaf node is identified or the path weight

Algorithm 2 Diagnosis algorithm for LID (adapted from [Armengol and Plaza, 2001a])

1.5emindent=0.10em

```
1: Initialize variables $D_{initial}$, $D_{final}$ and $D_{explored}$; Set $D_{initial} = D_f$, $D_{final} = \emptyset$, and $D_{explored} = D_f$
2: while $D_{initial} \neq \emptyset$ do
3:     Select some $f_j \in D_{initial}$
4:     Initialize top search node to $f_j$; $f_{top} = f_j$
5:     Select $f_j$ from $D_{initial}$. Initialize variable $f_{top} = f_j$
6:     Select a node, $f_q \in V : f_q \notin D_{explored}$ and $\exists$ an edge $E(f_{top}, f_q)$ that has the highest probability of occurrence ($sigma$)
7:     if $\exists$ a node $f_q$ then
8:         Apply generate-and-test methodology, from the CMM, on the selected node $f_q$
9:         if test result is a success then
10:            Replace fault $f_i$ with selected node $f_q$
11:            Re-build *similitude* term ($D_{f_q}$) for the new fault $f_q$
12:            Apply terminating criteria algorithm (Algorithm 1)
13:            if $\exists$ a leaf node $f_l \in D_f$ then
14:                $D_{final} = \{f_l\}$ and exit.
15:            else
16:                Calculate and store the path weight $W(\pi(f_j, f_q))$; $W(\pi(f_j, f_q)) = \{\sigma(f_j, f_{j+1}) + \sigma(f_{j+1}, f_{j+2}) + ... + \sigma(f_{q-1}, f_q)\}$
17:                Assign $f_{top} = f_q$,
18:            Add $f_q$ to explored set; $D_{explored} = D_{explored} + \{f_q\}$
19:            GOTO Step 4
20:    else
21:        if $f_{top} \in D_{initial}$ then
22:            Remove $f_{top}$ from $D_{initial}$; $D_{initial} = D_{initial} - \{f_{top}\}$
23:            GOTO Step 2
24:        else
25:            Back-trace a level of abstraction to explore alternate nodes; $f_{top} = \{f_q \mid \exists$ an edge $E(f_q, f_{top})$ and $f_q \in D_{explored}\}$
26:            GOTO Step 4
27: if $D_{final} = \emptyset$ then
28:    $D_{final} = \{f_q : f_q \in D_{explore}$ and $max(W(\pi(f_i, f_q)))\}$. Select $f_q$ as the diagnosis, i.e., the node in the CMM having the largest path weight from the initial diagnosis, based on the observed symptom.
29:    Expand similitude set to go up a level of abstraction; $D_f = \{\{f_1, f_2, ..., f_n, ....\} \mid f_n \in V$ and $\exists$ a connected edge $E(f_n, f_j)$ and $W(\pi(f_n, f_j)) \leq$ threshold-value, $\forall f_j \in D_f\}$
30: if $D_f \neq \emptyset$ then
31:    GOTO Step 1
32: if $f_j \in D_{final} = f_{root}$) then
33:    Refer to human operator for classification. (i.e., the fault could not be classified.)
```

calculated from the selected node to any node of the original similarity space D_f exceeds a threshold value. As long as the path weight is calculated to be lesser than the threshold value, the search process continues. Depending on the system requirements, the designer pre-defines the threshold value. If the system is unable to altogether build a similarity space for the encountered fault, the robot transfers control to a human for further analysis. The resultant diagnosis is then used to adapt the causal model for future use by the multi-robot team.

Example

To illustrate how the learner works in a multi-robot application, we now detail the steps taken by the learner to diagnose a sample fault with the help of an example. Consider the following simple scenario from our sample application.

Problem

A helper robot, R_1, is attempting to go from one fixed position to another in a pre-mapped environment. The problem space (PS_1) (in the form of the causal model) for the robot is shown in Figure 4.5. The symptoms (ψ) and the solution set (S_1) associated with the faults are detailed in Table A.1. Before actual experimentation, the system is put through a training phase and a probability of occurrence for each node is calculated and incorporated into the CMM. During the course of task execution, a fault f_i is encountered. The symptom associated with the fault is identified as $\psi = \{$"time out in goto_goal behavior"$\}$.

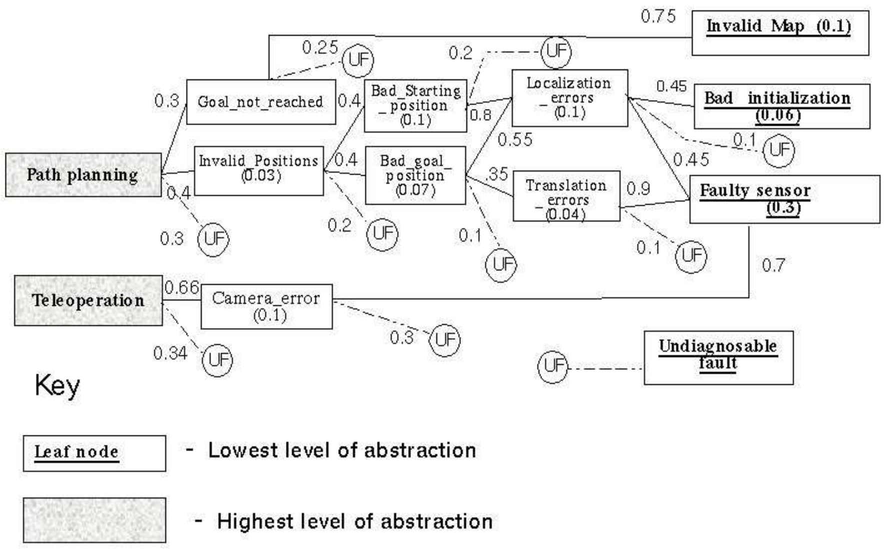


Figure 4.5: Causal model for robot R_1.

Stepping through the Algorithm

Once the exception is raised, the the state machine transfers control to the FtM for diagnosis and recovery*. Based on the encountered symptom and the information stored in the CMM (see Figure 4.5), the learner in the FtM attempts to build a similarity space, D_f, for the encountered fault f_i (see step 0 of Table 4.1). Once the similitude has been built, the learner checks for termination by applying the above mentioned criteria. Since D_f has only one node, *Goal_not_reached*, and it does not correspond to a leaf node, the learner invokes the diagnosis Algorithm 2.

The diagnosis algorithm starts out initializing variables D_{final}, $D_{explored}$, $D_{initial}$. As LID is based on the concept of *similitude*, the scope of the algorithm is restricted to a valid non-empty similarity space. If the initial *similitude* term ($D_{initial}$) is an empty set, then the diagnosis algorithm fails and a human user has to intervene to guide the system towards an eventual diagnosis.

Also, as the diagnosis algorithm involves modifications to the initial *similitude* term, the validity check is performed at regular intervals using a loop structure like *while*. In our example, as $D_{initial}$ is non-empty, the algorithm successfully passes through the *while* condition. Once inside the main body of the loop, the first step involves selecting a node from the initial *similitude* term and assigning it as the current top node (f_{top}) for exploration (see step 2 of Table 4.1)†. As we have only one node in $D_{initial}$, $f_{top} =$*Goal_not_reached*. The next step involves identifying a connected node from the f_{top} with the highest value of σ. From Figure 4.5 this corresponds to the node *Invalid_Map*. Once the node f_q is selected, to ensure that the diagnosis is proceeding in the right direction, the generate-and-test associated with f_q from Table A.1 is executed. The result of the test provides further information that can be used for diagnosis. Assuming the test to be successful for the above problem, the next step is to replace the fault symptom with the symptom correlating to the selected node f_q. This is done to further streamline the diagnosis by temporarily restricting the search process to involve only the sub-graph created by the selected node f_q. This is represented as the modified problem value for f_i in step 4 of Table 4.1. Once the problem is re-defined, the algorithm then builds a new *similitude* term called D_{f_q}. Once the *similitude* term is built, the terminating algorithm (Algorithm 1) is applied on D_{f_q} to check for leaf node. If a leaf node exists, that node is selected as the final diagnosis and the algorithm exits. Since the generated *similitude* term contains the leaf node, *Invalid_map*, the first terminating criterion from Algorithm 1 is satisfied and the recovery action associated with *Invalid_map* is executed. The selected leaf node that solves the fault along with the final *similitude* term D_{final} is shown in step 5 of Table 4.1. The corresponding recovery action associated with the diagnosis, obtained from Table A.1, is given by $s_i = \{$"Alert human to replace map"$\}$.

*Table 4.1 illustrates the step by step progression within the diagnosis algorithm that corresponds to the textual description.

†Since all selected nodes have a single characteristic they are equally similar to the fault.

Table 4.1: Step-wise illustration of Algorithm 2

Step	**Variable**	**Value**
Step 0	fault (f_i)	Symptom = “time out in goto goal”
	Similitude (D_f)	$\{Goal\_not\_reached\}$
Step 1	Initial *Similitude* set ($D_{initial}$)	$\{Goal\_not\_reached\}$
	Final *Similitude* set (D_{final})	$\{\emptyset\}$
	Explored *Similitude* set ($D_{explored}$)	$\{\emptyset\}$
Step 2	Top node (f_{top})	$\{Goal\_not\_reached\ \}$
Step 3	Connected node (f_q)	$\{Goal\_not\_reached\}$
Step 4	Modified f_i	Symptom is “Map not found”
	New *similitude* (D_{f_q})	$\{Invalid\_Map\}$
Step 5	Leaf node (f_l)	$\{Invalid\_Map\}$
	Final *similitude* (D_{final})	$\{Invalid\_Map\}$

Updating the case base

When a fault is encountered in the system, two stages are involved in the recovery process. First, the system diagnoses the fault, and second, useful information is extracted from the fault for future use. Using the LID technique for identifying *similitude* enables us to consider only the most relevant cases to the encountered fault. This enables us to quickly and accurately narrow down the nodes responsible for the fault. However, the trade off for such a method is that by ignoring other less relevant characteristics, we are throwing away potentially useful information regarding the system and the environment. Currently, as we restrict the heuristic set to a single characteristic, no useful information is thrown away. On the other hand, the diagnosis process may identify new nodes and/or different connections between existing nodes.

To incorporate information on the current fault, we update our original case set based on new relevant information extracted from fault f_i. Let $\delta = D_{final}$. The updated vertex set is represented as $V' = \{f_1, f_2, f_3..., f_n\} \cup \delta$ and the edge set is updated to hold any new relation between the newly added node and existing nodes. The updated edge set is given by

$$E = E + \{\{f_s, f_i\} \mid \forall f_s \in D_{initial} \& f_i \in D_{final}\} \tag{4.1}$$

In addition, the system needs to update the probability of occurrence associated with each node, taking into consideration the current diagnosed fault. Updating the node weights enables LID to dynamically adapt its search strategy towards more frequently occurring faults. This in turn reduces the time spent in fault diagnosis and enhances overall system performance. Node weights are given as follows,

$$\forall f_j \in \pi(f_i, f_l) \cdot \sigma(f_j, f_{j+1}) \leftarrow \frac{\sigma(f_j, f_{j+1}) * \text{ sample\_size}}{\text{sample\_size } + 1} \tag{4.2}$$

where f_i is the initial diagnosis node and f_l is the final diagnosis or the leaf node, and sample_size is the total number of faults that have been encountered by the system todate. This could include faults during initialization, testing and during normal operation. Sample_size can also be used to control the rate of convergence or learning for a system[‡].

Once the appropriate action has been executed and the case base updated, the robot coordinates with other team members regarding the newly diagnosed fault. The other robots attempt to identify the new fault based on their individual case bases. If the fault is not identified, then all additional information regarding the fault is requested. Once all information is exchanged, the new fault is then added and the case base updated. This type of cooperation between team members ensures all team members maintain current causal model information.

For our above detailed example, if we assume the selected recovery action solved the encountered failure, we must then attempt to identify any new information identified during the course of diagnosis and incorporate it back into the causal model shown in Figure 4.5.

[‡]A designer can use an arbitrary value for sample_size based on his/her experience in dealing with the operating environment.

As there are no new nodes or edges added, the vertex set and the edge set remain the same. Figure 4.6 represents the updated causal model with updated σ values representing the modified probability distribution.

4.2 Discussion

Chapter 5

Metrics for Evaluating System Performance and Fault Tolerance

Any system that has the capability to diagnose and recover from faults is considered to be a fault-tolerant system. Unfortunately, achieving fault-tolerance for large teams of robots is an extremely difficult problem due to the large number of interacting system components. The inherent explosion of state space complexity [Atkins et al., 1997] for multi-robot teams means that a practical system restricts the scope of fault-tolerance towards optimizing system performance. One possible way of measuring fault-tolerance is by defining the redundancy in a system, perhaps achieved through interchangeable components that can substitute for each other if one (or more) of the components fail. Most multi-robot applications are distributed in nature, and when robots are homogeneous, they can provide a natural redundancy to each other. However, while redundancy by itself is a useful measure, it is incomplete as an evaluation metric, since a system can also be effectively fault-tolerant through reasoning methods other than redundancy. Thus, it is preferred to have a metric that can measure the effective fault-tolerance as it influences overall system performance in achieving the tasks of the application. In addition to measuring the extent of fault-tolerance exhibited by a system, it is useful to have a metric that can evaluate the quality of the implemented fault-tolerance.

5.1 Problem Definition

As it is difficult to objectively evaluate metrics, it becomes important for a given metric to identify and subsequently evaluate the key features that define the fault-tolerance of a system. According to Murphy and Carlson [Carlson and Murphy, 2004] four important factors are essential for the success of a diagnostic system:

- **Efficiency** — ability of the system to optimize available time and resources towards task completion,
- **Robustness** to noise — ability of the system to identify and recover from faults, and
- **Learning** — There are two types of learning a system can exhibit,

– Dealing with uncertainty — ability of the system to adapt to the changes in the operating environment,

– Dealing with sparse information — ability to extract and integrate useful system information during the course of task execution, without the need for a large number of examples or a long training time.

The ability of the metrics to assess a system based on the above criteria, makes it useful for evaluating and comparing different fault-tolerance architectures based on their system performance. Based on Murphy and Carlson's hypothesis, we define a generic set of metrics capable of calculating the quantity and quality of fault-tolerance in the system. The formal definition of the problem is as follows. We are given:

- An autonomous robot team $R = \{R_1, R_2, R_3, ..., R_n\}$.
- A pre-defined set of tasks to be executed by the robot team $T = \{T_1, T_2, T_3, ..., T_m\}$, where each task T_j is executed by a separate robot R_i.

We assume the following:

- The task assignment is pre-defined by means of a set of pairings $< R_i, T_j >$. An individual task T_j is executed by the specific robot R_i.
- Faults can occur naturally during task execution or can be artificially introduced into the system.
- Faults are broadly categorized into three (3) types: *known*, faults the designer can anticipate; *unknown*, faults not anticipated by the designer, but which can be diagnosed by the system based on experience and available sparse information; and *undiagnosable*, faults that cannot be classified autonomously and need human intervention. The number of faults in each category are represented as f_{known}, $f_{unknown}$, and $f_{undiagnosable}$.
- The robots have three (3) functionally significant operating states: *Normal* state, in which a robot focuses all its system resources and operating time towards completing the assigned task; *Fault* state, in which a robot spends all available time and resources in attempting to identify the source of the encountered fault; and *Recovery* state, in which a robot spends its resources and operating time in executing the recovery action for the diagnosed fault.
- Once assigned to a robot, a task can have two possible outcomes: *success* or *failure*. Task success is defined as the ability of the robot to successfully complete its assigned task. A task failure is defined as the inability of the robot to complete its assigned task in the presence of faults.
- If a robot (R_j) fails to complete a task (T_j), then based on the system design, the system can either assign task T_j to a different robot R_i, re-assign T_j to the task queue of robot R_j, or remove task T_j from the system task list.
- Every task-assignment, $\langle R_i, T_j \rangle$, is considered a *task attempt* and is evaluated separately towards overall system performance.

- An award is associated with every successfully completed task, given by the *utility* component u_j ; the punishment associated with a failed task attempt is given by the *cost* component for task failure, c_j.
- Based on the importance of each individual task relative to the others, the designer builds a utility-cost table, in which the summation of the term $\sum u$ is normalized to 1.
- To ensure normalized metrics across differing systems, the cost value is tied to the corresponding task term, i.e., $c_j = u_j$.

5.2 Measuring System Performance

In developing our metric, we first define the total number of faults for the i^{th} attempt of task T_j as the summation of all encountered faults during the course of task execution. That is, $F_j^i = f_{known_j}^i + f_{unknown_j}^i + f_{undiagnosable_j}^i$.

Successful completion of task T_j is measured by means of a success metric, A_j:

$$A_j = u_j \tag{5.1}$$

Then, the system level measure of success (A) is calculated as:

$$A = \sum_{j:T_j \in X} u_j \tag{5.2}$$

where $X = \{T_j \mid \text{Task } T_j \in T \text{ was successfully completed}\}$. That is, the system level measure of success is the sum of the utilities of the tasks that were successfully completed.

Similarly, we associate a task failure metric, B_j^i, for each unsuccessful attempt of task T_j by a robot. On the other hand, as the performance is closely tied with the robot's ability to recover from faults, every failed task has a robustness component associated with it. The effect of the task failure metric towards performance is discounted by the extent of the robustness in the task, i.e., the higher the robustness, the lower the value of the task failure. In other words, we can use robustness to quantify the extent of fault-tolerance in the system. We define ρ_j^i as the measure of robustness for the i^{th} attempt of task T_j and is given by

$$\rho_j^i = \frac{f_{knownj}^i + f_{unknown_j}^i}{F_j^i} \tag{5.3}$$

That is, ρ_j^i gives the fraction of the faults that the system could successfully recover from.

Based on equation 5.3, the task failure metric for the i^{th} attempt of task T_j is:

$$B_j^i = c_j * \left(1 - \rho_j^i\right) \tag{5.4}$$

Grouping all failed attempts of a task T_j, we get the combined task failure metric (B_j) for a task T_j as:

$$B_j = (c_j * q_j) * \sum_{i=1}^{q_j} (1 - \rho_j^i) \tag{5.5}$$

where q_j is total number of failed attempts of task T_j. The upper bound of q is application specific and needs to be determined by the designer before implementation.

Extending equation 5.5 across all task failures, gives:

$$B = \sum_{j:T_j \in Y} (c_j * q_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i) \tag{5.6}$$

where $Y = \{T_j \mid \text{Task } T_j \in T \text{ failed}\}$.

Finally, the measure of performance can be obtained by subtracting the cost associated with a task failure from the utility for successful task completion, i.e.,

$$P = A - B \tag{5.7}$$

Substituting for A and B from equations 5.2 and 5.6 respectively, we obtain our desired effective performance metric:

$$P = \sum_{j:T_j \in X} u_j - \sum_{j:T_j \in y} (c_j * q_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i) \tag{5.8}$$

P provides the designer with a measure of the system's effective performance. The measure results in P values in the range $(-\infty, 1]$. A value of 1 indicates an optimal system performance, whereas P approaching $-\infty$ indicates a total system failure. As P by itself does not provide all the information necessary for evaluation, we need to identify additional individual metrics that help give a complete picture of the system.

5.2.1 Measuring Fault-tolerance

In addition to outlining a measure for performance, we are interested in identifying the fault-tolerance exhibited by the system. As mentioned above, we measure the system fault-tolerance in terms of robustness, efficiency and learning. These components provide a good metric for identifying the extent and usefulness of fault-tolerance towards improving overall system performance.

Combining individual task robustness measures from equation 5.3, system robustness can be represented as:

$$\rho_s = \frac{\sum_{j:T_j \in Y} \sum_{i=1}^{q_j} \rho_j^i + \sum_{q:T_q \in X} \rho_q^1}{X + Y} \tag{5.9}$$

A high value of ρ_s (an ideal system exhibits a ρ_s value of 1) indicates a highly robust system and a ρ_s value of 0 indicates a system with no robustness to faults.

As the ultimate goal of any fault-tolerance architecture is to achieve task success in the presence of failures, it is important that the system maximizes its usage of resources and time towards the completion of the assigned task. Towards that, it is necessary to define the system efficiency metric (ϵ), the total task-execution time spent by a robot on a successfully completed task, T_j. This is given by the summation of time spent in Normal (t_{Normal}), Fault (t_{Fault}) and Recovery ($t_{Recovery}$) states for that attempt, i.e.,

$$t_j = t_{Normal_j} + t_{Fault_j} + t_{Recovery_j} \tag{5.10}$$

$$\epsilon_j = \frac{t_{Normal_j}}{t_j} \tag{5.11}$$

Similar to the robustness measure, combining efficiency across the tasks gives:

$$\epsilon = \frac{\sum_{j:T_j \in X} \frac{t_{Normal_j}}{t_j}}{X + Y} \tag{5.12}$$

A more efficient system has a higher value of ϵ and an inefficient system has ϵ near 0.

Subsequently, the influence of learning exhibited by a system towards system performance can be measured by tracing the rate of change of diagnosis for the *known* and *unknown* types of faults[*]. By comparing the efficiency of the system over the set of trials[†], we get an estimate of the learning exhibited by the system.

$$\delta_k = \epsilon_k^{'} - \epsilon_k^0 \tag{5.13}$$

$$\delta_u = \epsilon_u^{'} - \epsilon_u^0 \tag{5.14}$$

where $\epsilon_k^{'}$ is efficiency metric for the *known* faults for final trial, ϵ_k^0 is the efficiency value for *known* faults for initial trial, $\epsilon_u^{'}$ is efficiency metric for the *unknown* faults for final trial and ϵ_u^0 is the efficiency value for *unknown* faults for initial trial. Typically, a negative value for P or a δ value close to 0 is a good indicator of the lack of adequate learning in the system. Additionally, plotting and tracing the efficiency rate over the course of normal operation can be used to identify the effectiveness of the implemented learning algorithm. Hence, after computing the metric values, we attempt to reason about the observed behavior of the evaluated system. The reasoning tools are especially useful for fine-tuning an implemented fault-tolerance architecture, leading to the development of more refined and effective solutions. We describe our approach in detail in Chapter 6. Finally, based on the above definitions for robustness, efficiency and learning, we can represent system level effective fault-tolerance as an unordered triple given by (ρ, ϵ, δ).

5.2.2 Discussion

In the previous section, we have detailed distinct and separate measures for calculating system performance and fault-tolerance. In justification, when measured separately neither of the two measures provide a complete assessment of the application in use. Using only

[*]As *undiagnosable* faults fall outside the realm of classification for any fault-tolerance architecture, we can ignore such faults and restrict our focus to the *known* and *unknown* types of faults.

[†]An experiment consists of a set of more than 1 trials.

system performance, we do not get a fair idea regarding the extent of fault-tolerance in the system. On the other hand, fault-tolerance by itself is not a strong enough measure for evaluating the intelligence in the system. The combined metrics provide a good basis for evaluating and comparing different fault-tolerance architectures based on their system performance.

As mentioned earlier, it is difficult to develop a truly objective set of metrics. The somewhat arbitrary nature of any particular metric leaves it open for subjective interpretation. Specifically, our metrics are designed as an empirical measure, i.e., they are calculated post-experimentation based on the collected data, to account for the ability of a system to adapt to unexpected changes in the environment. Though the metrics can be applied to any environment, static or dynamic, if the operating environment is modified, then there is a need to recompute the metrics for the new data. As a consequence it is difficult to theoretically prove the correctness of the developed metrics. However, by focusing on measuring the essential qualities that define a fault-tolerance system, that of robustness, learning and efficiency, we attempt to make the metrics less arbitrary and more objective in evaluating and comparing system performance of different fault-tolerance architectures.

Chapter 6

System Evaluation

6.1 Evaluation

In the previous chapters, we have introduced two different methods for multi-robot fault-tolerance (CMM and LeaF), reasoned on the drawbacks of CMM and discussed the need for a more robust, learning based method like LeaF. However, in order to meaningfully evaluate the two architectures, we need to implement and analyze their performances for sample multi-robot applications. Towards that,

- We implement CMM and LeaF for two distinct physical multi-robot experiments,
- Analyze the obtained data to compare the performance of the two systems, and
- Subsequently, we apply the set of developed metrics described in Chapter 5 to the obtained experimental results. The ensuing analysis will help identify the extent of intelligence or reasoning on the fault-tolerance influencing the overall performance of the system.

Additionally, in order to validate the developed metrics, we contrast it with the only other existing method [Hamilton et al., 1996] for evaluating (that we are aware of) effective fault-tolerance for multi-robot teams. The comparison highlights the inadequacies of the existing method and furthers the necessity for a more well-rounded metric capable of effectively rating a robot system, such as the one we have developed as part of this research.

This chapter is organized as follows. Sections 6.2 and 6.3 detail the experimental platform and the programming details respectively. In Section 6.4, we briefly discuss the initial simulation tests for exploring the applicability of LID in the multi-robot fault-tolerance domain. In Section 6.5, We use the Box pushing task as my first of the two implementation examples. The obtained results are compared and inferences are drawn. The two systems are then further evaluated based on the derived metrics. Section 6.6 details how the systems performed for an alternate physical robot experiment, the locate-and-protect task that serves as our motivational example (see Chapter 1). Finally, in Section 6.7, we apply the alternate set of metrics (HWB) to the obtained results from the two sets of experiments, and discuss the differences between the two metrics.

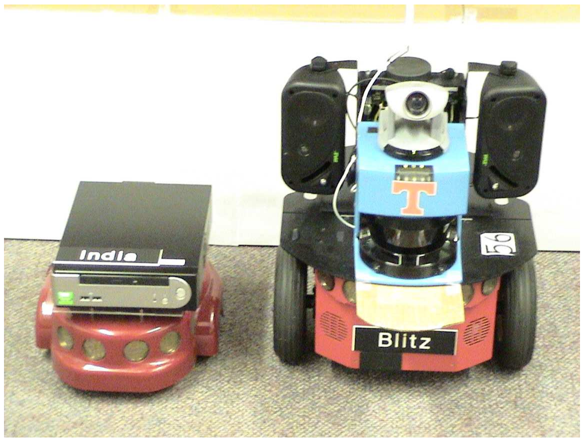


Figure 6.1: Amigobot and Pioneer used in experimentation.

6.2 Experimental Platform

For my experiments, we use Activmedia's Pioneer 3DX (Figure 6.1) and Amigobot robots as my hardware platform. The Pioneer 3DX is a differential-drive based wheeled mobile robot. The core of the robot's on-board processing unit is an Intel Pentium III processor (850MHz). As the predominant task of the robot is active sensing and navigation, it has a standard suite of sensors including a ring of 16 sonars (8 on front and 8 at the back), a SICK LMS-200 laser range-finder and a forward mounted Sony VC4 Pan-Tilt-Zoom camera. The Amigobots are a physically smaller sized robots equipped with an external Via 8000 processor that provides processing power up to 1.64 GHz. The sensor suite on the Amigobots includes a differential drive motor system, an external logitech web cam, connected via the sub interface, and a ring of sonars located at front and back. For the purposes of our experiments, we restrict the Amigobot to use only its camera and motor sensors. Information is sent in packets over a tethered RS232 serial connection to and from the Via boards to the Amigobots. Both the Pioneers and the Amigobots use a Orinoco gold wireless card (using 802.11 Ad-hoc network) for external communication.

6.3 Programming Details

Both fault-tolerance architectures, CMM and LeaF, are implemented using the standard C++ programming language structure developed on a Linux (Gentoo) platform. The fault-tolerance architecture code is meshed with the corresponding robot control code using the Player robot software (version 2.0.1). Player [Gerkey et al., 2003] is a Linux based network server that provides a simple interface to the robot's sensors and actuators over a network. Client programs written in C++ communicate over a TCP socket, reading data from sensors, and writing commands to actuators. As Player is an established open-source software in the robot community, it provides support to a variety of robot hardware. In addition, Player's modular architecture makes it easy to add support for new hardware.

6.4 Experiment 0: The LID Simulator

In order to ensure the validity of our adapted LID algorithm, we test it in simulation before applying to the physical robots. The setup involves simulating a single robot performing a random task when a fault occurs. The initial CMM is hand coded into the system. A fault exception is artificially introduced into the system. The symptom correlating to the exception is predefined and fed into the system along with the exception. This simulates the data gathering stage of the real robot. The simulator then applies the LID algorithm to the fault and identifies the new fault. Any useful information is extracted and added to the existing CMM. The fault diagnosis performance and the time taken for analysis are noted. The test is then repeated with the new CMM and the subsequent times are noted. The tests are performed for different faults and different starting CMM's. The experimental setup does not really provide a realistic approximation of the multi-robot environment and as a consequence the collected data bears no significance towards the overall performance of the developed system. However, the simulator does work as a proof of concept test and provides a developmental ground for implementing LeaF.

6.5 Experiment I: Box Pushing Task

In order to illustrate the usefulness of LeaF, we apply it to a standard problem in the multi-robot community: the box pushing task. The implemented box pushing algorithm is based on prior work done by Tang [Tang, 2006] as part of her dissertation work. As in the original implementation, we use a team of two Pioneer robots equipped with a laser-scanner, a ring of sonars, and a forward mounted camera. The experiment could be extended to include a larger team of robots. The fundamental idea behind the box pushing protocol used in the experiments is adapted from the work done by [Donald et al., 1994], [Parker, 1998]. The robots are placed on either end of a box and the task is to push the box over a pre-determined distance. The robots locate the goal position, indicated by a red blob, and calculate an appropriate pushing direction based on the relative orientation of the box* to the goal. In addition, based on the relative distance information, the robot constantly updates its speed profile and assumes an obstacle-free path from the starting position to the goal. For the experiment, the robots do not need any information about the orientation of the box or its team members (see Figure 6.2).

To ensure modularity and the re-usability of code, the overall task was broken down into sub-tasks that were completed by teaming three base-line behavior modules. The box pushing task is composed of three such base-line behaviors, namely: box push, communication and blob tracking. Table 6.1 shows the relation between the individual module and the set of sub-tasks and Figure 6.3 shows how these behaviors are combined with one another. Additionally, as the robots are initially placed close to the same face of the box, they can locate the correct positions on the box to push and align themselves with the box such that the orientations of the robots are perpendicular to the pushing face of the box.

*As the robots do not know the explicit orientation of the box, the robot's own pose information is used as an alternate.

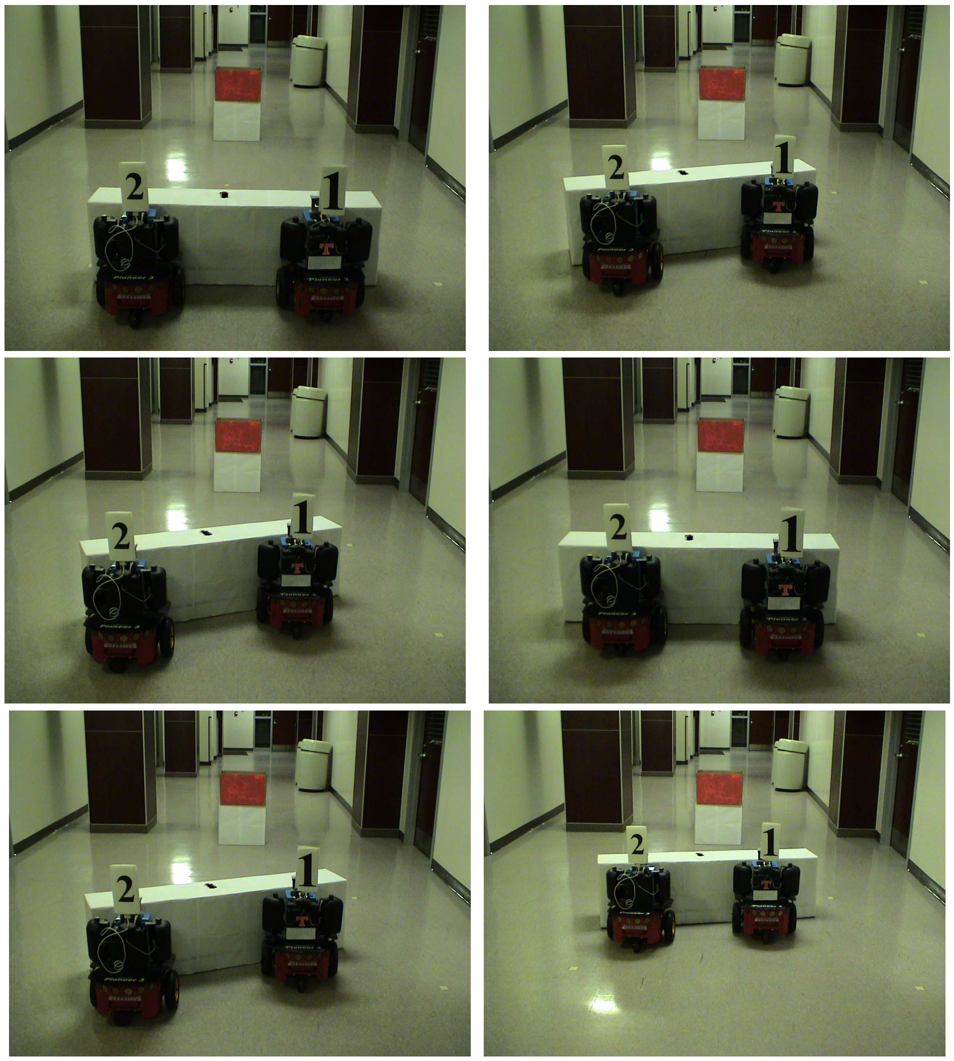


Figure 6.2: Example of a box pushing task – Robots 1 and 2 push a long box towards the goal indicated by a blob at the far end of the corridor. Robot 1 pushes one end of the box and waits until robot 2 has successfully pushed its end. The process continues until the goal is reached (read left to right, top to bottom).

Table 6.1: Task module relationship table for box pushing task

Sub-task	**Modules**
Alignment	Blob tracking, Push box
Go to goal Task	Blob tracking, Communication, Push box

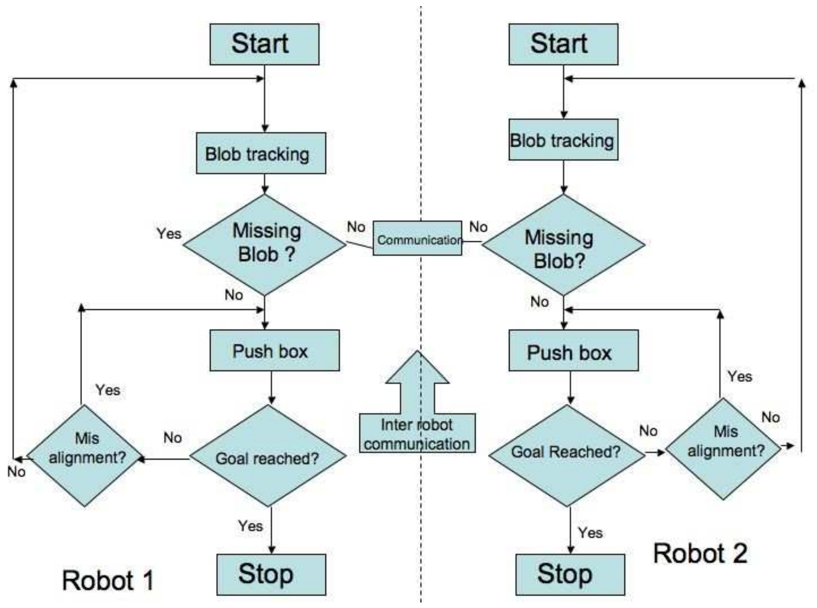


Figure 6.3: Behavior set for box pushing task.

The two robots pushing the box concurrently share their status with each other to synchronize their behaviors. The status information updates the robots as to whether its partner has pushed, aligned or completed the task.

6.5.1 Experimental setup

Two separate sets of experiments were performed. For the first set, I used CMM as the fault-diagnosis method, whereas LeaF is implemented as the fault diagnosis strategy for the second set of experiments. To ensure consistency, data was collected from over 18 trial runs for each set of experiments. Over the course of experiments, various failures were encountered, some of which were expected and others that were unexpected.

The relatively simple nature of the task ensured that, apart from the individual implementation detail for each architecture, the parameters of testing for both sets of experiments were kept the same. The rigidity of the experimental test-bed meant that the obtained results could then be compared directly with one another.

For the box pushing task, both LeaF and CMM are implemented on the robots using a rule-based approach that defines the various possible failure modes as represented in the causal model. Table A.3 lists the rules used to recognize the predefined failure states implemented for this task, along with the corresponding corrective actions. The corresponding causal models are shown in Figures 6.4 and 6.5. Both systems start with the same initial causal model; however, LeaF has a probability value associated with the edges of its causal model, based on their frequency of occurrence. The probability values associated with fault likelihoods of occurrence at the node level are initialized to be equal at the beginning of the experiments.

Also, the specifics of representation for the causal models are given as follows:

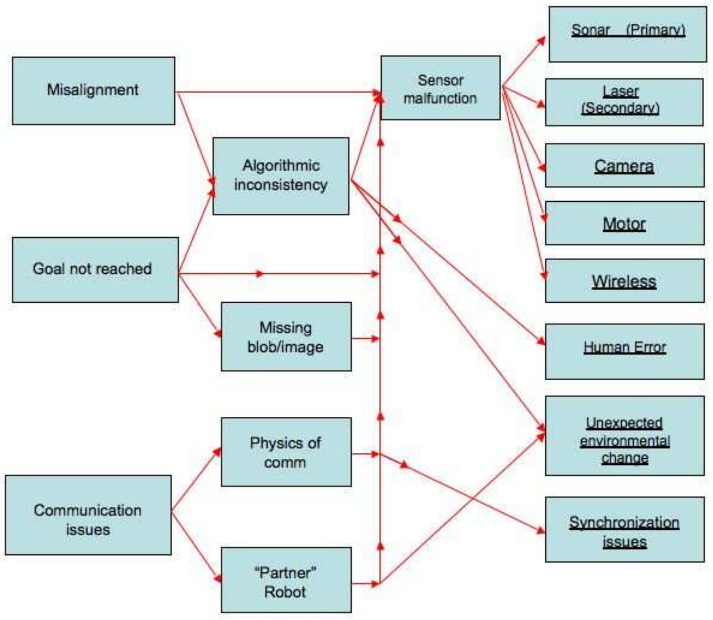


Figure 6.4: Initial causal model for CMM for the box pushing task.

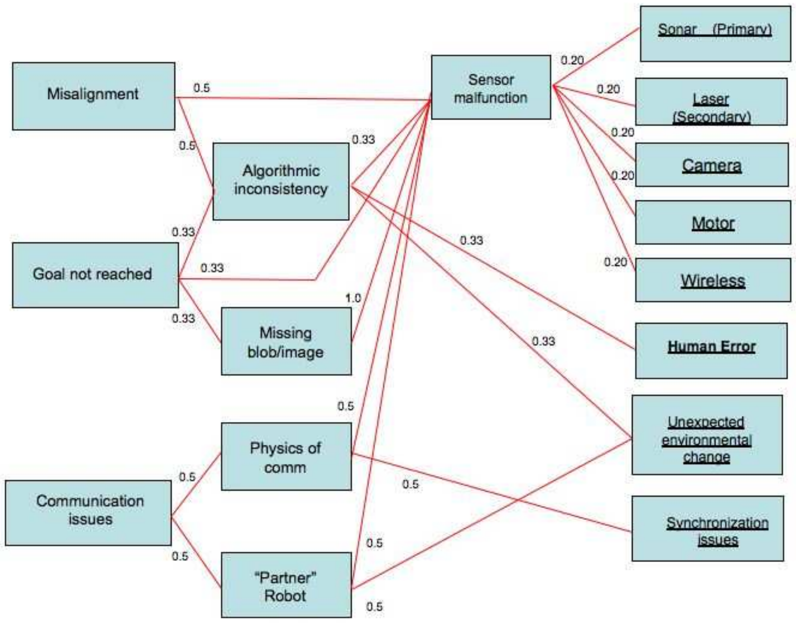


Figure 6.5: Initial causal model for LeaF for the box pushing task.

- V correlates to the vertex set representing all the available nodes for both systems,
- Set E_{LeaF} contains all the edges between the nodes for the LeaF system,

Interpreting Table A.3 and Figure 6.5, we can list the vertex and edge sets —

- V = { *misalignment, goal not reached, communication issues, algorithmic inconsistency, missing blob, physics of comms, "partner robot" errors, sensor malfunction, sonar, laser, camera, motor, wireless, human error, unexpected environmental change, synchronization issues*}
- E_{LeaF}= { {*misalignment, sensor malfunction*}, {*misalignment, algorithmic inconsistency*}, {*goal not reached, algorithmic inconsistency*}, {*goal not reached, sensor malfunction*}, {*goal not reached, missing blob*}, {*communication issues, physics of comms*}, {*communication issues, "partner" robot*}, {*communication issues, sensor malfunction*}, {*algorithmic inconsistency, sensor malfunction*}, {*algorithmic inconsistency, human error*}, {*algorithmic inconsistency, unexpected environmental change*}, {*missing blob, sensor malfunction*}, {*physics of comms, synchronization issues*}, {*physics of comms, sensor malfunction*}, {*"partner robot", sensor malfunction*}, {*"partner robot", unexpected environmental change*}, {*sensor malfunction, sonar*}, {*sensor malfunction, laser*}, {*sensor malfunction, camera*}, {*sensor malfunction, motor*}, {*sensor malfunction, wireless*}}

The problem space is then specified by $< V, E >$ with the symptoms (ψ) and the solution set (S_1) associated with the faults being detailed in Table A.3.

After each run of the task, diagnosis information, including the number of faults encountered, identified, type and reasoning for the fault, are collected. These are tabulated in Table B.1 along with additional information regarding type and number of faults encountered. The collected data provides good feedback on system performance over the length of the trials. In the experiments using LeaF, the probability values associated with fault likelihoods of occurrence are updated with experience; subsequent trials beyond the initial one made use of the updated values.

6.5.2 Discussion

Towards meaningful comparisons, trials with similar characteristics, in terms of faults seen, are grouped together. Thus Trial 8 from Table B.1 experiences communication and blob tracking errors for both LeaF and CMM. Additionally, as every trial is an independent event for the task of box pushing, the performance of one does not influence the performance of subsequent trials for CMM. However, LeaF does carry over the experience (good or bad) from one trial to another.

Analyzing the results from Table B.1 we can see the distribution of the various faults that occur during experimentation. Over the course of 15 trials, CMM encounters a total of 41 faults, whereas the count is 37 for LeaF. The entire set of encountered faults can be essentially classified into one of the following two types: known(k) or anticipated, and unknown(u)[†] or un-anticipated. From the initial causal model (see Figure 6.4), we can see

[†]Once a previously unknown error has been classified, LeaF adds it to the causal model and the error falls into the known category. For clarity however, we will still refer to those errors as unknown errors.

that *Synchronization issues* is an example of the known type of fault for the communication sub-module. In this case, there already exists a path from the top node (*communication issues*) to the leaf node (*Synchronization issues*). Alternately, in the case of an unknown fault there exists no path from the top-node or the sub-module to the fault node. *Communication failure due to human error* is an example of an unknown fault. The distribution of known to unknown errors is 33 to 8 for CMM and 25 to 12 for LeaF. In most cases, the errors occurred naturally during normal task operation in a non-sterile environment. In certain cases, the fault was introduced artificially introduced in to the system. For example, the sonar error was experimentally created by masking the incoming raw data (causing the *sonar fault* error).

Looking closely at the diagnosis time information from Table B.2, we can see that the LeaF system shows a distinct improvement in diagnosis time over the course of the trials, especially for certain types of faults. Consider for example, in Trial 4 (the first time an error was seen in the communication module) both CMM and LeaF had comparable diagnosis times (47 and 50). For subsequent failures, LeaF is able to reduce its diagnosis time to be considerable lower than CMM. In order to understand the reason for improved diagnosis times, we need to look at the timing data specific to faults seen during experimentation. From the results, we can see that initially both LeaF and CMM perform comparably for known set of faults like *synchronization issues* (Trials 4, 5, 6, etc.). However, as the number of trials increases or the system becomes more experienced, we note that the time for the LeaF diagnoses decreases. Specifically comparing values for the fault *synchronization issues* in Trial 14, CMM's time is pretty close to its original time of 50 seconds whereas LeaF diagnosis the fault in 24 seconds. This can be explained as the system improving the balance of the probability of occurrence values for the fault nodes for the box pushing task. As the system stabilizes, the process of fault diagnosis becomes more efficient, leading to faster diagnosis time and as a consequence faster execution times.

As it is impossible to anticipate all faults, how the system handles unknown faults is a good indicator of its overall performance. The ability of a system to gracefully degrade in the presence of un-recoverable faults is a key feature for a system to be truly application-independent, as it eliminates the need for extensive parameter-tuning. Also, re-analyzing the data from Tables B.1 and B.2 specific to the handling of unknown faults highlights the advantage of LeaF over CMM. Consider a specific example: in Trial 10, the systems encounter two different faults. As the robots start out, fault due to *Synhronization issue* is encountered. Both robots are successfully able to handle the faults (3), though LeaF has a lower diagnosis time. However, during the course of task execution the lights are turned off which results in blob tracking faults. The table shows how CMM system was unable to provide any useful diagnosis even after a long time interval (in excess of 600 seconds[‡]), whereas the LeaF system is able to diagnose the error relatively quickly. Additionally, LeaF shows progressive improvement in the diagnosis time for future occurrences of the fault (Trial 12). A latter re-occurrence of the fault in Trial 13 still shows the same diagnosis time as the POC values are yet to be balanced due to the relatively new nature of the encountered fault.

[‡]Experiments were stopped after 600 seconds to preserve robot battery life.

Figure 6.6 illustrates the stepwise classification sequence (based on the LeaF diagnosis algorithm from Chapter 4) for the above mentioned fault, f_i = *blob tracking failure due to environmental changes*. The symptom associated with the fault is identified as $\psi = \{$"Unable to find blob"$\}$. Based on the encountered symptom and the available causal model, LeaF attempts to match all nodes correlating to the encountered fault and build a similarity space, D_f, for the encountered fault f_i. Once the similitude has been built, LeaF checks the terminating criteria using Algorithm 1 from Chapter 4. Since the built similarity space, D_f, is restricted to a single non-leaf node, *Missing blob/image*, further classification is attempted using Algorithm 2. The node from the initial similarity space, $D_{initial}$ is then selected for further diagnosis (see Figure 6.6(a)).

The next step involves identifying a connected node, from the selected node, with the highest value of σ. Figure 6.6(b) indicates the selected node *sensor malfunction.* To ensure that the diagnosis is proceeding in the right direction, the generate-and-test associated with f_q from Table A.3 is executed. As the selected node is more of a filter than an actual hardware test, and the generate-and-test return is successful, the next step in the process is to continue the diagnosis process further into the graph. In order to streamline the diagnosis process, LeaF temporarily restricts the search process to involve only the sub-graph created by the node *sensor malfunction.* Re-applying the algorithm to the new sub-graph, the system builds a new *similitude* term called $D_{f_q} = \{$*sonar, laser, motor, camera, wireless*$\}$, and applies the terminating algorithm. As the new term has multiple leaf nodes with equal probability of occurrences, the leaf node selection is implementation specific. The order of execution implies that node *sonar* is the first selected leaf node (see Figure 6.6(c)). Algorithm 1 is applied on D_{f_q} to see if it is a leaf node. However, applying generate-and-test on the *sonar* node results in a failure. LeaF then backs up to the preceding level of the sub-graph, *sensor malfunction*, eliminates *sonar* from the list of *similitude* term D_{f_q}, and selects an alternate leaf node.

Unfortunately, as the fault is previously unknown the generate-and-test strategy fails for all the nodes (see Figures 6.6(d) through 6.7(a)). Having explored all the nodes in the sub-graph without finding a solution, LeaF then reverts back to the original graph further expanding the search space. However, as the original selected node *missing blob/image* has only one valid edge, to *sensor malfunction*, that has already been explored it needs to further expand the search space beyond the original *similitude* space. This results in the system backtracking to the top level module node *Goal not reached* (see Figure 6.7(b)). As before, the diagnosis algorithm is then invoked and subsequently the lone un-explored node from the most recent search space:

$D_{modified} = \{$*algorithm inconsistency, sensor malfunction, missing blob*$\}$
— is selected. Figure 6.7(c) shows LeaF selecting *algorithm inconsistency.* After unsuccessfully exploring a potential leaf node solution in *Human error* (see Figure 6.7(d)), the system eventually ends up with the leaf node *Unexpected environmental failure* as shown in Figure 6.7(e). With the success of the applied generate-and-test strategy for the selected node, LeaF settles on the node as the eventual diagnosis and executes the associated recovery action. The last step in the process requires LeaF to re-scan the entire causal model to determine if there exists a forward moving path from the initial diagnosis node *Missing blob/image* to the final leaf node *unexpected environmental change.* As there does not exist

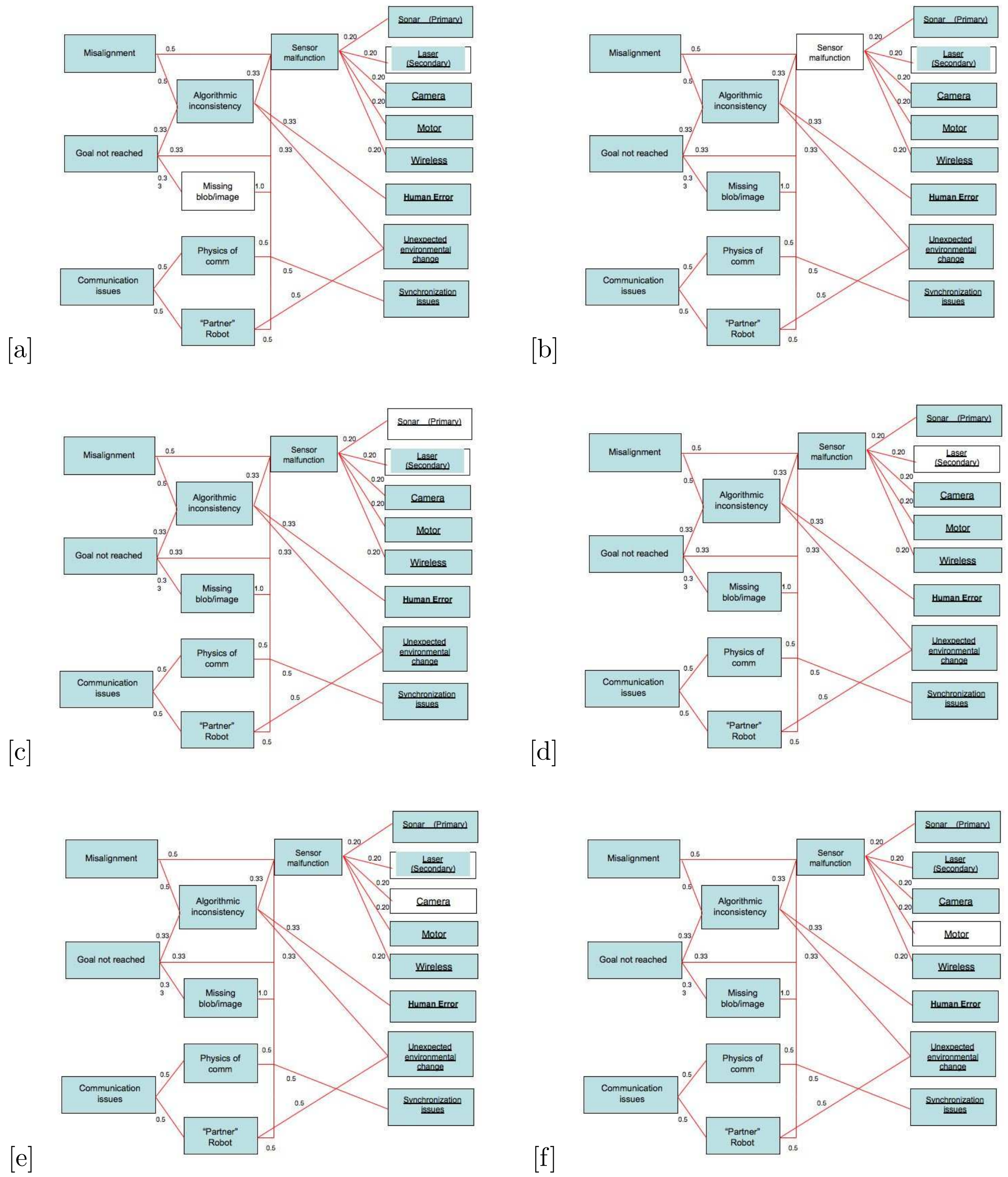


Figure 6.6: Part I – Trace through of the diagnosis process for the classification of an unknown fault: Steps a through f, the system scans through the causal model, effectively back-tracking before continuing the diagnosis process when a fault can not be diagnosed based on the initial symptom.

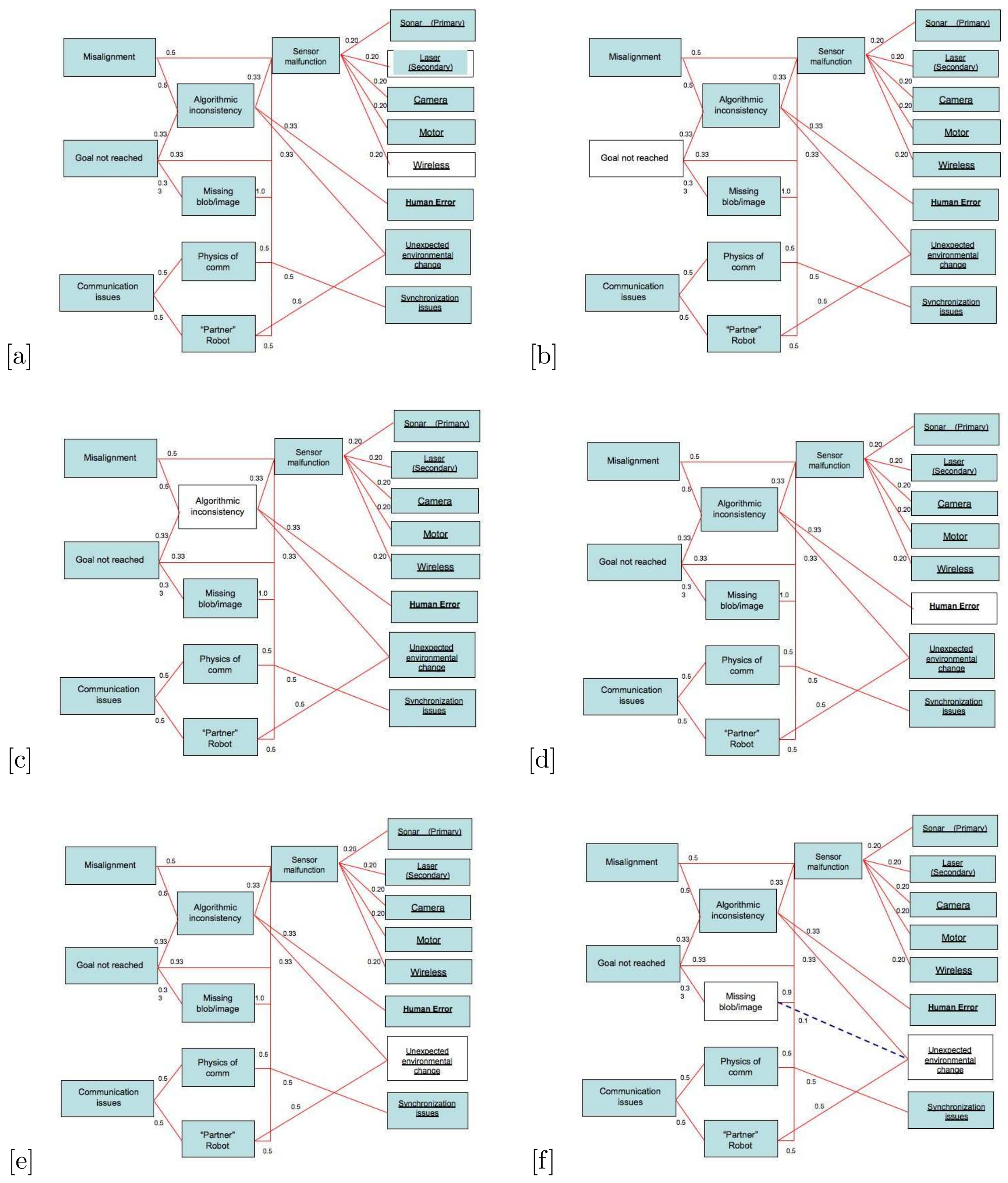


Figure 6.7: Part II – Trace through of the diagnosis process for the classification of an unknown fault.

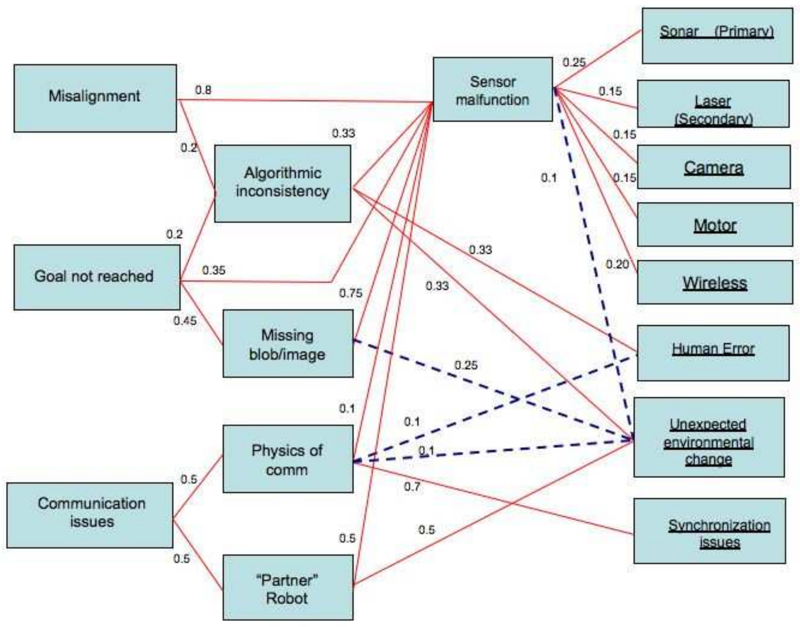


Figure 6.8: Final updated causal model for the box pushing task

one, an edge between the two nodes is created and the probability weights are accordingly adjusted. The updated causal model is shown in Figure 6.7(f).

A key point is that the static nature of the CMM design does not allow it to be flexible enough to adapt to the dynamics of the environment. As a consequence, the system does not make full use of the available information, whereas the ability of LeaF to adapt and expand its search space beyond the initial diagnosis makes it capable of identifying previously unknown errors. Once LeaF identifies the new error, it adds a new edge in the CMM, thus reducing the time taken for subsequent searches. Figure 6.8 shows the final causal model at the end of all trials.

Having looked at the timing information above, we can reason that LeaF exhibits two different types of learning behavior, one for known type of faults and the other for unknown type of faults. In the first one, using the probability of occurrence information, LeaF attempts to balance the graph towards maximizing system efficiency. This type of learning occurs constantly over the course of task execution from Trial 1 through Trial 15. This is highlighted in Figure 6.9 which traces the diagnosis time for one such fault through the course of the entire experiment. As the frequency of occurrence of the said fault increases, LeaF shows considerable improvement in diagnosis time whereas CMM maintains the initial value throughout. The other instance of learning can be considered to be an "on-demand" type and occurs only when an unknown types of fault is encountered. As a consequence, LeaF displays better utilization of resources at hand, which in this case is the time of operation. By reducing the diagnosis time, LeaF can utilize most of its execution time towards task completion.

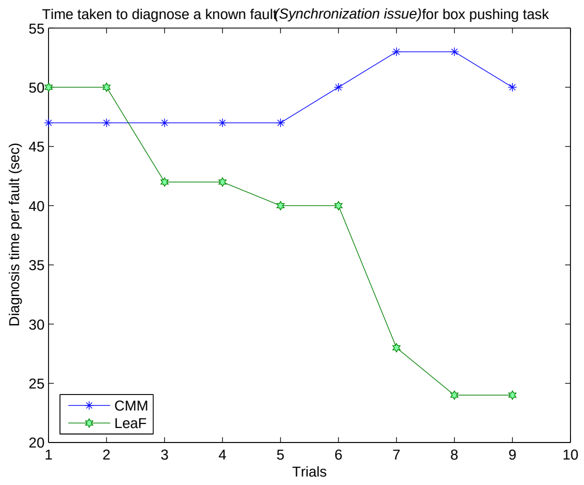


Figure 6.9: Rate of diagnosis for LeaF and CMM for the box pushing task.

6.5.3 Using Metrics for Evaluation

An alternate method for comparing the two systems is by applying the metrics that we have developed to the obtained results. The metrics provide a more objective, albeit a less detailed, analysis of the two systems, and attempt to quantify the extent of learning and fault-tolerance exhibited by the systems. Towards that we define the the utility/cost values associated with the corresponding tasks as shown in Table 6.2. Recall, the task-utility is constructed by the designer based on the importance of the sub-tasks such that the summation of the terms ($\sum u$ and $\sum c$) are normalized between ranges of $[0, 1]$. For box pushing, since both the sub-tasks carry equal importance towards the completion of the overall task, the values are distributed equally.

The system performance measure from Chapter 5 is given by:

$$P = \sum_{j:T_j \in X} u_j - \sum_{j:T_j \in y} (c_j * q_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i)$$

From the collected experimental values in Tables B.1, we can calculate

$$\begin{aligned}
P_{CMMTrial1} &= u_{gotogoal} + u_{alignment} - 0 \\
P_{CMMTrial2} &= u_{gotogoal} + u_{alignment} - 0 \\
P_{CMMTrial3} &= u_{alignment} - c_{gotogoal} * (1 - \rho_{gotogoal}) \\
&\vdots \qquad \vdots
\end{aligned}$$

Table 6.2: Utility-cost table for the experiments

Task	**CMM**	
	Utility	**Cost**
Go_to_goal Task	0.50	0.50
Alignment Task	0.50	0.50

$$\begin{aligned}
P_{CMM} &= \frac{P_{Trial1}+P_{Trial2}+P_{Trial3}+...+P_{Trial15}}{No.of Trials} \\
P_{CMM} &= \frac{1.0+1.0+(0.5-(0.5(1-0.5)))+...}{15} \\
&= 0.25
\end{aligned}$$

Similarly calculating for LeaF gives us,

$$\begin{aligned}
P_{LeaFTrial1} &= u_{gotogoal} + u_{alignment} - 0 \\
P_{LeaFTrial2} &= u_{gotogoal} + u_{alignment} - 0 \\
P_{LeaFTrial3} &= u_{alignment} + u_{alignment} - 0 \\
P_{LeaFTrial4} &= 0 - (c_{gotogoal} * (1 - \rho_{gotogoal}) \\
&\quad + c_{alignment} * (1 - \rho_{alignment})) \\
&\vdots \\
P_{LeaF} &= \frac{P_{Trial1}+P_{Trial2}+P_{Trial3}+...+P_{Trial15}}{No.of Trials} \\
P_{LeaF} &= \frac{1.0+1.0+1.0+...}{15} \\
&= 0.7
\end{aligned}$$

Furthermore, we calculate values for ϵ (see Table B.3 for timing information) and ρ,

$$\begin{aligned}
\rho_s &= \textstyle\sum_{j:T_j \in Y} \sum_{i=1}^{q_j} \rho_j^i + \sum_{q:T_q \in X} \rho_q^1 \\
\rho_{CMMTrial1} &= 0; \\
\rho_{CMMTrial2} &= 0; \\
\rho_{CMMTrial3} &= \rho_{gotogoal}; \\
&\vdots
\end{aligned}$$

$$
\begin{aligned}
\rho_{CMM} &= \tfrac{0.0+0.0+0.6+...+0.0}{15} \\
&= 0.400 \\
\rho_{LeaFTrial1} &= 0; \\
\rho_{LeaFTrial2} &= 0; \\
\rho_{LeaFTrial3} &= \rho_{gotogoal}; \\
&\vdots \quad \vdots \\
\rho_{LeaF} &= \tfrac{0.0+0.0+0.6+...+0.0}{15} \\
&= 0.734 \\
\epsilon &= \textstyle\sum_{j:T_j\in X} \frac{t_{Normal_j}}{t_j} \\
\epsilon_{CMMTrial1} &= 1.0; \\
\epsilon_{CMMTrial2} &= 1.0; \\
\epsilon_{CMMTrial3} &= 0 \\
&\vdots \quad \vdots \\
\epsilon_{CMM} &= \tfrac{1.0+1.0+0+...+0.0}{15} \\
&= 0.2567 \\
\epsilon &= \textstyle\sum_{j:T_j\in X} \frac{t_{Normal_j}}{t_j} \\
\epsilon_{LeaFTrial1} &= 1.0; \\
\epsilon_{LeaFTrial2} &= 1.0; \\
\epsilon_{LeaFTrial3} &= 0 \\
&\vdots \quad \vdots \\
\epsilon_{LeaF} &= \tfrac{1.0+1.0+0+...+0.0}{15} \\
&= 0.3791
\end{aligned}
$$

The calculated performance ratings are enumerated in Table 6.3. In addition to calculating system performance, we also calculate the respective fault-tolerance values. Comparing the two systems, we can see that LeaF outperforms CMM in all three categories. For the box pushing task, LeaF has a performance rating that is twice as good as the CMM.

Table 6.3: Performance Evaluation table for box pushing task

System	**P** $\in (-\infty, 1]$	$\rho \in [0,1]$ **per trial**	$\epsilon \in [0,1]$ **per trial**	δ_{known} **per trial**	$\delta_{unknown}$ **per trial**
CMM	0.25	0.400	0.2567	0	0
LeaF	0.7	0.734	0.3791	1.01	2.83

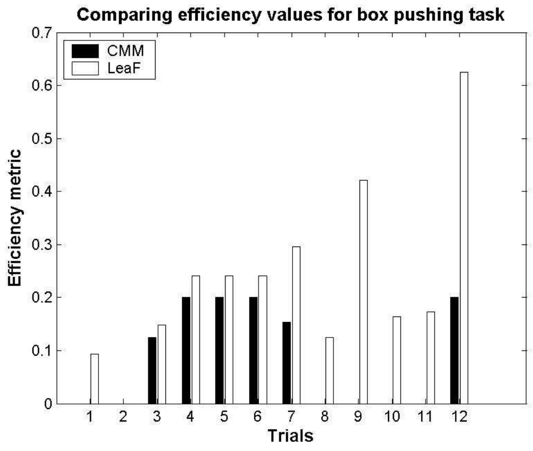


Figure 6.10: Efficiency rating discrepancy between LeaF and CMM for the box pushing task. The missing values in the graph (Trials 1, 2, 8, 9, 10, 11) correlate to an efficiency rating of 0; i.e., the system was unable to diagnose the fault and had to be manually rebooted.

Even though the numerical values are comparable, it is probably reasonable to expect the rate of learning of the system, and as a consequence the system performance, to slow down as it approaches the peak value of 1. Also, it can be said from the data that LeaF exhibits twice as much robustness to failure when compared to CMM. The ability of LeaF to handle un-expected faults is the reason for the higher robustness rating. On initial analysis, it appears that the efficiency of the two systems are comparable. However, looking purely at the net efficiency value is a little deceiving. The fact that the systems encounter a larger number of known type of faults than the unknown type means that the system does not have enough sample data for the new faults to show a higher overall efficiency rating. The discrepancies in the efficiency values between the two systems are illustrated in Figure 6.10[§].

We can see clearly from the graph that LeaF better utilizes operating time and resources. In fact, LeaF displays consistent learning through the course of the trial, resulting in improved values for efficiency across the system. Observing efficiency values for Trials 8, 10 and 11 for the two systems shows the ability of LeaF to not only diagnose unknown faults but to work towards improving efficiency for future instances of the said fault. However this is typical of a multi-robot system. Hence, despite the improvement over CMM, the low system efficiency rating for LeaF indicates the need for a more streamlined implementation of the diagnosis process.

For experiments involving independent trials, it would be incorrect to compare the learning exhibited by the systems based purely on performance for individual trials. Also, as there is potential for two distinct types of learning occurring during the course of operation, we need a more analytical method to identify the extent of learning, if any, exhibited by the systems. We already looked at one type of learning, for the known fault in the previous section. Let us take a closer look at the individual execution times for each trial (see Table B.3) to see if any further information regarding the alternate types of learning can

[§]For trials that are stopped after 600 seconds, the ϵ value is considered to be 0, as there is no possibility of the system ever coming out of the diagnosis process to resume normal operations.

be extracted. Considering only known faults like *synchronization issues, camera fault, etc*, from Tables B.1 and B.3, we can calculate the rate of change of the diagnosis time. For example, the diagnosis time for *synchronization issue* fault changes from 50 seconds (initial) to a final time of 17 seconds, a change of 33 seconds over the course of 23 trials. Similarly, we can calculate the rate of change for other faults. The net change per trial highlights the extent of learning exhibited per trial. Similarly, the rate of change of diagnosis for unknown fault is also calculated. An unknown fault may occur in only certain trials and may not be present in all, so the $\delta_{unknown}$ is calculated only from the first time the fault is diagnosed, instead of from Trial 1. This is because, once the fault has been diagnosed and an edge added, it becomes part of the *known* class of faults. Finally, comparing the values of P, ρ, and δ for the two systems, we can say that LeaF is a more suitable architecture for the box pushing application.

6.6 Experiment II – Indoor locate-and-protect task

6.6.1 The SDR Project

The physical robot experiment described in Chapter 1 was the motivating problem behind my research. The test application is a large-scale locate-and-protect mission involving a large team of physical heterogeneous robots. The robot team has a very strict set of goals/tasks: autonomously deploy a sensor network and use the network to track intruders within the building. The composition of the team consisted of two classes of robots: three (3) lead robots equipped with scanning laser range-finders and cameras; and a large number (approximately 70) of sensor-limited robots equipped with a microphone and a crude camera. All of the robots had 802.11 WiFi, and a modified ad-hoc routing package (AODV) was used to ensure network connectivity.

Figure 6.11 shows these robots performing one such deployment, using the behavior set shown in Figure 1.2. This scenario involves a complex combination of cooperative and single-robot behaviors, including laser-based localization, path planning, obstacle avoidance, vision-based autonomous tele-operation, simple vision-based following, and wireless *ad hoc* mobile communication. These behaviors are run on two physically different types of robots in a cluttered environment (like a hall-way in an office building), leading to a wide variety of possible failure modes. Table 6.4 shows the relationship between the individual modules and the defined set of tasks.

Similar to the box pushing task, we build a rule base representing the different potential failure states. Table A.4 lists the rules used to recognize the predefined failure states implemented for this task, along with the corresponding corrective actions. The corresponding causal model is shown in Figure 6.12. The experiments consisted of repeated deployments of 1 and 2 simple robots per team. Over the course of the experiment, various failures were encountered, some of which were expected and some that were totally unexpected. If a deployment failed on one experiment, the consequences of that failure were not corrected, except on rare occasions.

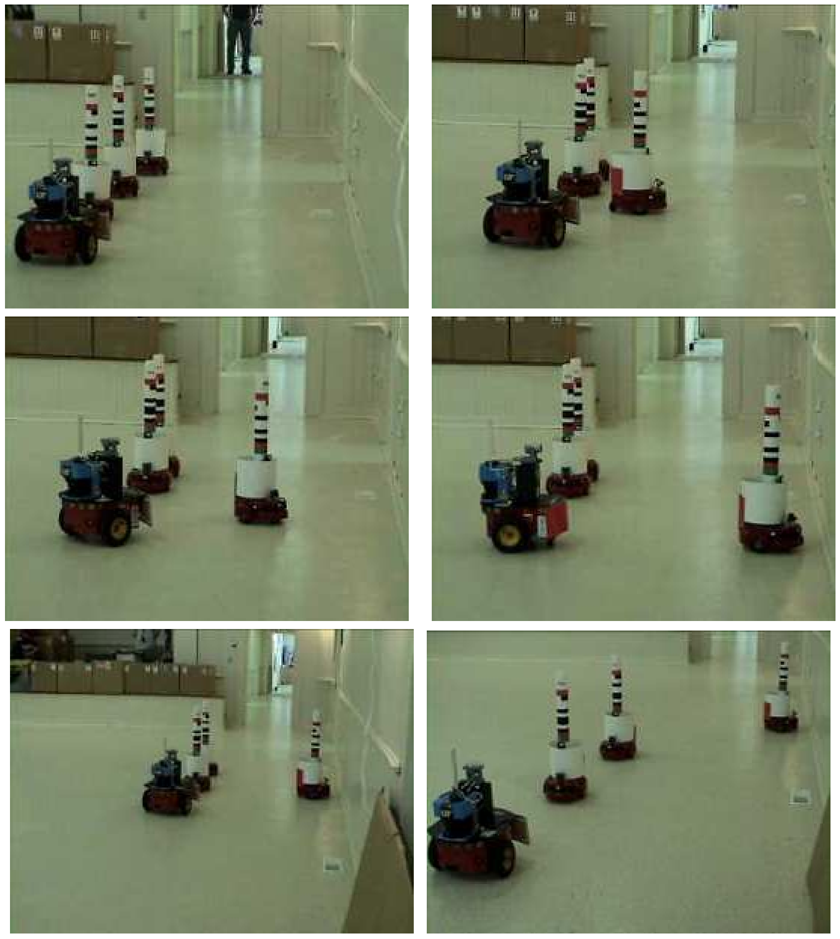


Figure 6.11: Deployment of a sensor robot using assistive navigation: the lead robot first guides and then directs the sensor robot into position (read left to right, top to bottom).

Table 6.4: Task module relationship table for CMM and LeaF

Task	**Modules**
Go_to_goal Task	Localization, Path_planning, Navigation
Tele-operation Task	Marker Detection, Communication
Re charging Task	Localization, Path_planning, Navigation, Marker Detection, Communication
Follow_the_leader Task	Blob Tracking
Return_home Task	Localization, Path_planning, Navigation

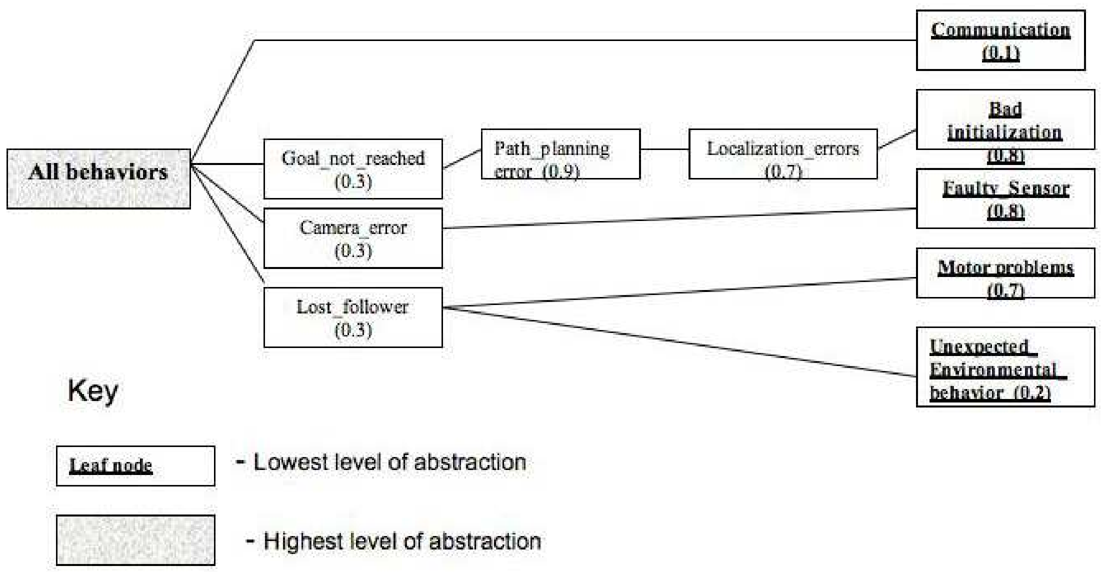

All behaviors
Goal_not_reached (0.3)
Camera_error (0.3)
Lost_follower (0.3)
Path_planning error (0.9)
Localization_errors (0.7)
Communication (0.1)
Bad initialization (0.8)
Faulty Sensor (0.8)
Motor problems (0.7)
Unexpected Environmental behavior (0.2)
Key
Leaf node - Lowest level of abstraction
- Highest level of abstraction

Figure 6.12: Initial causal model for the deployment behaviors in the locate-and-protect scenario.

Thus, the data collected incorporates propagation of error from one experiment to the next.

Recall from Chapter 3, the experimental data obtained for the DARPA/SDR project Over a period of 45 runs, the system showed an overall deployment success rate of 60% - 90%, depending upon the environmental characteristics. In other words, for each attempt at deploying a simple robot, 60% - 90% of those robots successfully reached their planned deployment position. Column 2 of Table 3.1 depicts the probability of success of each individual module in this implementation and the overall system probability, based upon the experimental results. The probability values are used to calculate individual and collective task robustness. To better understand the pure empirical data, we apply the developed metrics on the obtained results. During the evaluation process certain constraints had to be accounted for, most important of which was incorporating the disparity in the task/utility value associated with helper and sensor-limited robots. This is shown in Table 6.5.

For these experiments, ρ is a measure based on the total probability of task success. Task success is given by the product of the individual sub-modules completion probabilities for a specified task. Also, as the faults propagate from one run to the next, the entire set of trials (45) is considered as a single continuous experiment.

Table 6.6 shows the evaluated values for system performance, robustness and efficiency. The ability of CMM to handle expected faults is the reason for the higher robustness rating.

Table 6.5: Utility-cost table for SDR

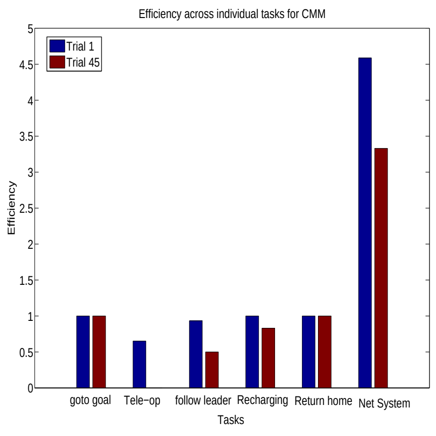
Task	**Utility**	**Cost**
Go_to_goal Task	0.20	0.20
Tele-operation Task	0.20	0.20
Re charging Task	0.15	0.15
Follow_the_leader Task	0.15	0.15
Return_home Task	0.30	0.30

However, the performance metric indicates a negative value, which shows that for the concerned application the implemented fault-tolerance does not optimize system performance. An alternate technique could potentially be used to further improve performance. The negative value for learning illustrates our earlier point that the performance degrades over the course of experimentation. It is to be noted that the lack of learning does not indicate a failure of the system to learn, instead it merely highlights that the system was not designed to be a learning system and hence its failure to adapt to unexpected changes during the course of task-execution. This means that for multi-robot domains, a system must often display a significant amount of active learning just for it to maintain its initial performance levels.

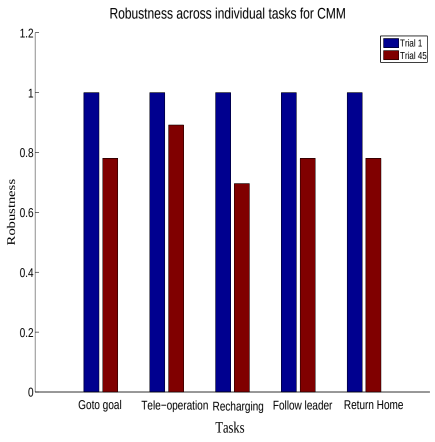
The efficiency and robustness graphs (Figures 6.13(a) and 6.13(b)) for CMM show that despite a high initial value the system shows a steady decline over the course of the trials ending with a significant decline in its measures. Finally, the rate of system learning is illustrated in Figure 6.20. The system performance progressively degrades over the course of the trials. Unlike the box pushing task, the trials for the SDR experiment are sequential, i.e., the outcome of one trial influences future trial success. As a consequence, increasing number of components combined with a lack of learning results in a sharp downward curve. In fact, we can make the claim that for such systems, the architecture should display significant learning just to maintain the initial performance rating.

Table 6.6: Evaluation table for CMM over 45 trials

System	$\mathbf{P} \in (-\infty, 1]$	$\rho \in [0, \infty]$ **per task**	$\epsilon \in [0, \infty]$ **per task**	δ
CMM	-9.012	2.95 per task	2.89 per task	-9.012



(a) Efficiency measured over time for CMM



(b) Robustness measured over time for CMM

Figure 6.13: Part I: Metrics illustrated over the period of operation for the SDR project.

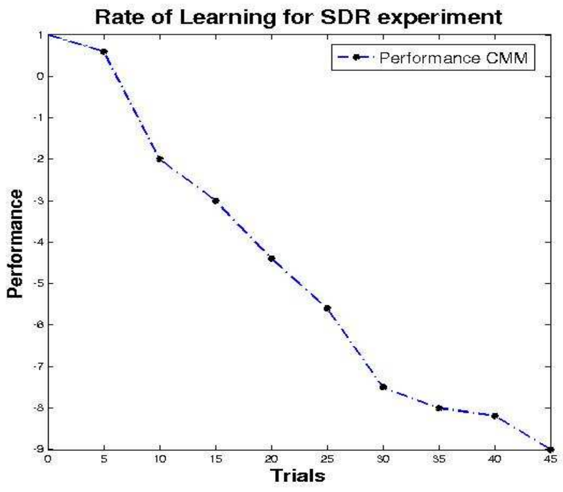


Figure 6.14: Part II: Learning illustrated over the period of operation of the SDR project.

6.6.2 Experiment IIb: Deployment task

Unlike the relatively simple nature of the box pushing task, certain restrictions, including the lack of sufficient hardware and access to the original test environments, makes it infeasible to recreate the SDR test scenario for both sets of experiments for my thesis work.

Also, it is important to note that without an accepted standard testing environment, like the one described in [Jacoff et al., 2003] for Urban search and rescue, it is impossible to recreate the exact operating environments used for testing CMM.

Hence, we redefine the problem to better suit the available resources. Based on the post-experimental analysis outlined in Chapter 3, we identified the deployment module from the SDR experiment as the sub-task with the highest probability of failure. Hence, we attempt to recreate the scenario focusing on the deployment task from the motivating problem.

For system validation, the SDR application involves a small team (2-5) of mobile, physically heterogeneous cooperating robots for an indoor deployment task. The same composition of the team members was maintained from the original SDR experiment. Figure 6.16 shows a simplified single-robot deployment task performed on the first floor of the Claxton building (see Figure 6.15 for building layout). In order to enable a fair comparison between the two systems, we re-implemented CMM in addition to developing LeaF for the deployment task.

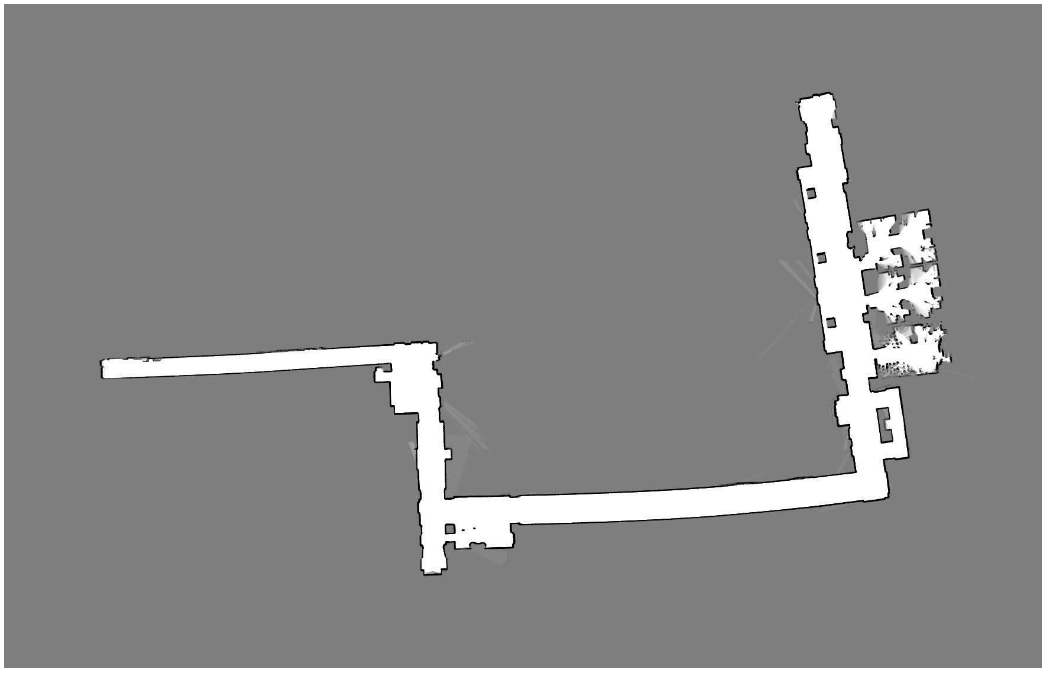


Figure 6.15: Layout of first floor of Claxton complex.

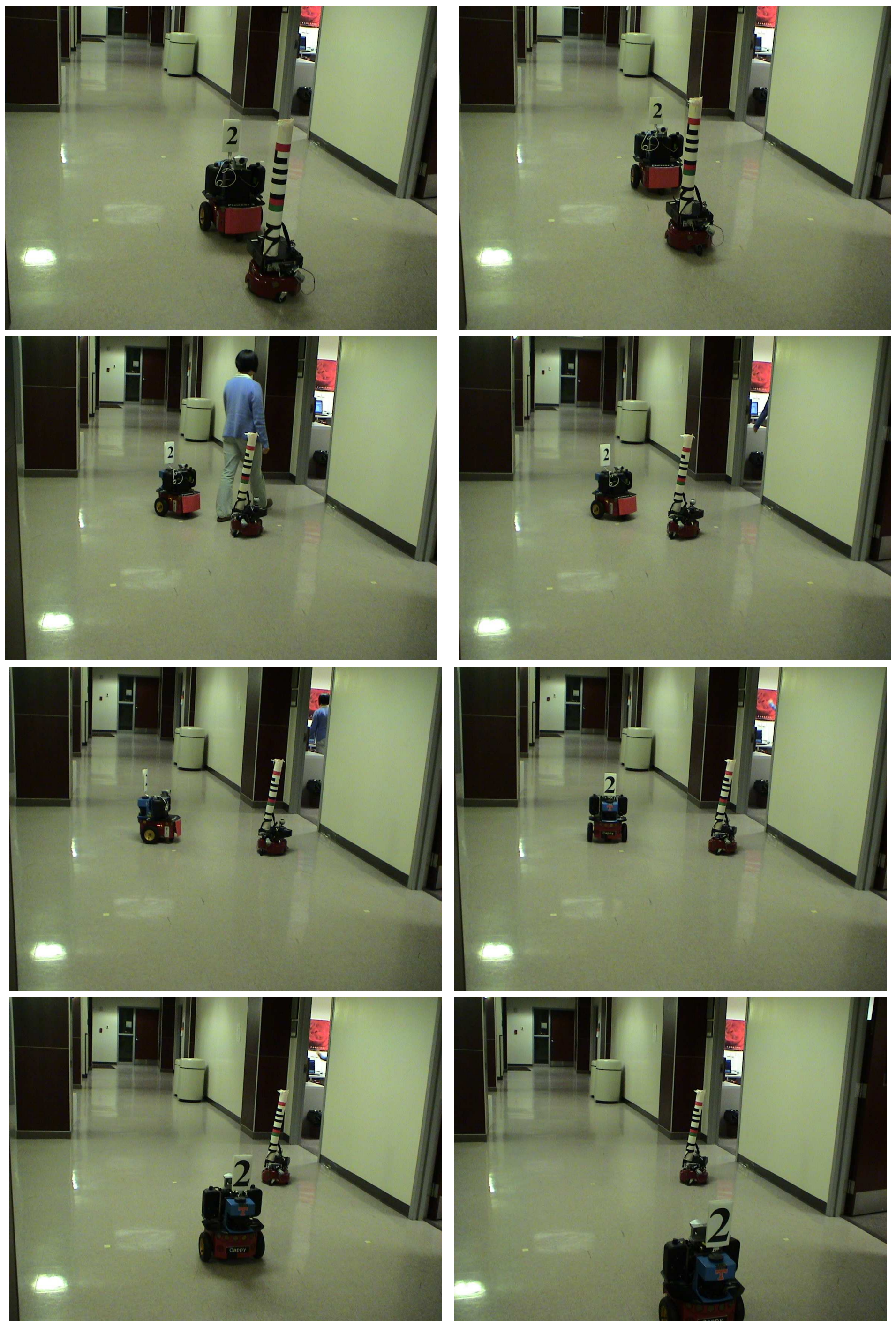


Figure 6.16: Deployment of a sensor robot using assistive navigation: the lead robot reaches a pre-specified way-point, tele-operates the sensor robot into position and heads back to the starting position (read left to right, top to bottom).

The corresponding initial causal models are shown in Figures 6.17 and 6.18. The experiments consisted of repeated deployments of a single simple robot into the environment. Similar to the CMM testing, various failures, expected and unexpected, when encountered were handled autonomously with minimal human intervention. The consequences of failure were not corrected, and the team was allowed to learn from its own mistakes, whenever possible. Unlike, the original task, each run was considered a separate, discrete event. In these experiments, a total of 15 simple robot deployments were attempted. After performing the experiments and collecting data, we analyze the results. Similar to the box pushing task, trials with similar characteristics, in terms of faults seen, are grouped together. LeaF treats the entire set of experiments as one long trial and carries over the experience (good or bad) information from one trial to another.

Table B.4 sequentially lists all the faults encountered from Trial 1 through 15. The distribution of faults is as follows: CMM – 23 faults, out of which only 6 were of type unknown, and LeaF – 35 faults, 13 of which belonged to the unknown category. On further review, the discrepancy was due to a large number of marker detection failures (5) in one particular trial (Trial 9). Oddly, the communication module did not encounter any faults. Also, we had one trial in LeaF that encountered 21 synchronization issues. This was due to the synchronization thread being accidently killed before the start of process execution. Despite the large number of faults, the average time for diagnosis was close to 25 seconds. We threw out the error as it was anomalous and skewed the existing data. Among all the modules in operation, it is the marker detection module that encounters the maximum number of faults. In the case of CMM, marker detection leads to a total of 16 errors of which 10 are recovered. This is consistent from what we saw for SDR, where the marker detection module had the lowest probability of success (78%). This can be attributed to the fact that the helper robot attempts to maintain line of sight with the simple robot in order to read and process its marker information. Hence, it is imperative that the

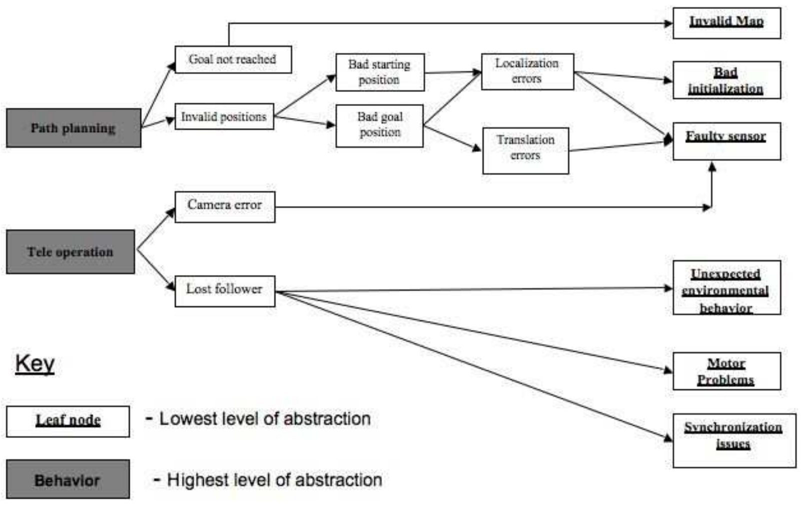


Figure 6.17: Initial causal model for CMM for the deployment task.

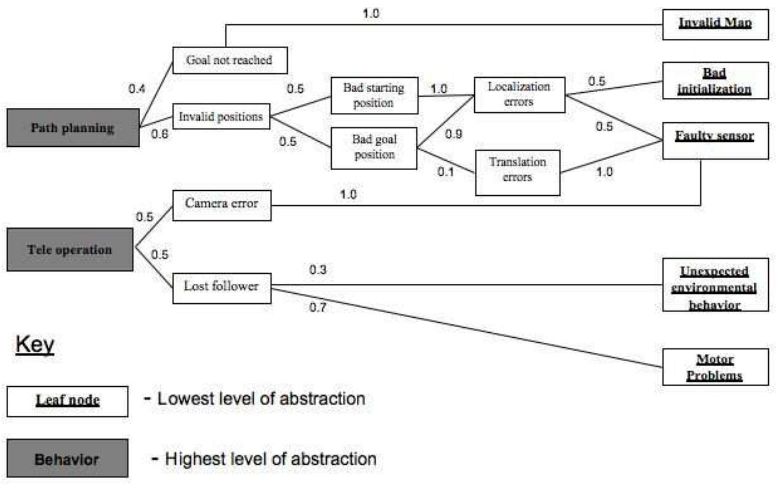


Figure 6.18: Initial causal model for LeaF for the deployment task.

robot be able to recover from such errors. Unfortunately, the inability to recover from marker detection[**] results in the robot being able to complete its task only on 6 trials. On the other hand, LeaF encounters marker detection based failures 21 times, of which all are recovered. The drawback to an approach like LeaF is that until the system balances the weights, there is a possibility that the system will perform worse than CMM. Consider Trials 12 through 15; they involve blob tracking error due to the simple robot's inability to keep up with the helper robot. Now looking at the timing values, we can see that CMM outperforms LeaF by about 20 seconds per diagnosis. Since the causal model for LeaF is still imbalanced, it will explore other alternate nodes before settling on the correct diagnosis. On the other hand, it just so happens that the implementation of CMM leads to select the correct diagnosis node, *synchronization issues*. However, once enough sampling data has been collected LeaF will perform at least as well as CMM does. The final updated causal model for the task is given in Figure 6.19.

6.6.3 Using Metrics for Evaluation

The measures are calculated similar to the previous experiment. We start by defining the utility/cost table values associated with the corresponding tasks as shown in Table 6.7. For deployment, since the most important criterion is for the robot to return home (from SDR), the module Return Home is assigned the highest utility/task value. Similarly, based on my prior experience with multi-robot systems, we determine the order of importance for the remaining modules and assign values. By letting the designer determine the task/utility value, we can make best use of the human's on-field experience.

As before, from the collected experimental values in Tables B.4 and B.5, we can calculate P, ϵ and ρ.

[**]To save battery life, we halted the trials after 360 seconds of diagnosis.

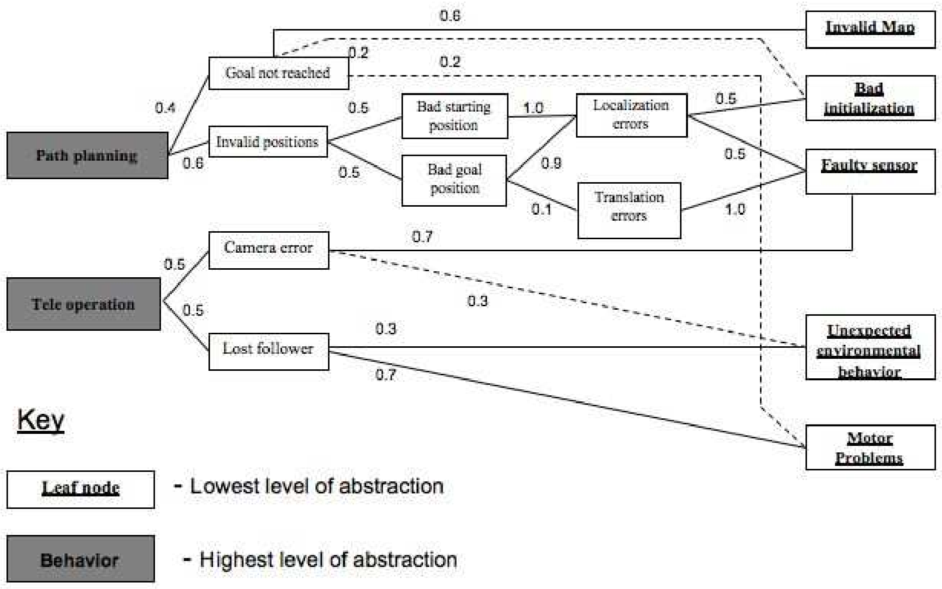


Figure 6.19: Final causal model for LeaF for the deployment task.

Table 6.7: Utility-cost table for the deployment task

Task	**CMM**		**LeaF**	
	Utility	**Cost**	**Utility**	**Cost**
Go to goal Task	0.25	0.25	0.25	0.25
Tele-operation Task	0.25	0.25	0.25	0.25
Follow the leader Task	0.20	0.20	0.20	0.20
Return home Task	0.30	0.30	0.30	0.30

$$\begin{aligned}
P &= \textstyle\sum_{j:T_j\in X} u_j - \sum_{j:T_j\in y}(c_j * q_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i) \\
P_{CMM}Trial1 &= u_{gotogoal} + u_{teleop} + u_{followtheleader} + u_{returnhome} - 0; \\
P_{CMM}Trial2 &= u_{gotogoal} + u_{followtheleader} + u_{returnhome} - c_{teleop} * (1 - \rho_{gotogoal}); \\
P_{CMM}Trial3 &= u_{gotogoal} + u_{teleop} + u_{followtheleader} + u_{returnhome} - 0; \\
\vdots &= \vdots \\
P_{CMM} &= \tfrac{P_{Trial1}+P_{Trial2}+P_{Trial3}+...+P_{Trial15}}{No.ofTrials} \\
P_{CMM} &= \tfrac{1.0+(0.75-(0.25(1-0)))+1.0+...}{15} \\
&= 0.1938; \\
P_{LeaF}Trial1 &= u_{gotogoal} + u_{teleop} + u_{followtheleader} + u_{returnhome} - 0; \\
P_{LeaF}Trial2 &= u_{gotogoal} + u_{teleop} + u_{followtheleader} + u_{returnhome} - 0; \\
P_{LeaF}Trial3 &= u_{gotogoal} + u_{teleop} + u_{followtheleader} + u_{returnhome} - 0; \\
\vdots &= \vdots \\
P_{LeaF} &= \tfrac{P_{Trial1}+P_{Trial2}+P_{Trial3}+...+P_{Trial15}}{No.ofTrials} \\
&= 0.855 \\
\text{and} & \\
\rho_s &= \textstyle\sum_{j:T_j\in Y} \sum_{i=1}^{q_j} \rho_j^i + \sum_{q:T_q\in X} \rho_q^1 \\
\rho_{CMMTrial1} &= 1; \\
\rho_{CMMTrial2} &= \rho_{gotogoal}; \\
\rho_{CMMTrial3} &= 1; \\
\vdots &= \vdots \\
\rho_{CMM} &= \tfrac{1.0+1.0+0.0+...+1.0}{15} \\
&= 0.6133 \\
\rho_{LeaFTrial1} &= 1; \\
\rho_{LeaFTrial2} &= 1; \\
\rho_{LeaFTrial3} &= 1; \\
\vdots &= \vdots \\
\rho_{LeaF} &= \tfrac{1.0+1.0+1.0+...+1.0}{15} \\
&= 1.0 \\
\text{and,} & \\
\epsilon &= \textstyle\sum_{j:T_j\in X} \frac{t_{Normal_j}}{t_j} \\
\epsilon_{CMMTrial1} &= 0.48; \\
\epsilon_{CMMTrial2} &= 0;
\end{aligned}$$

$$\begin{aligned}
\epsilon_{CMMTrial3} &= 0.48 \\
\vdots &= \vdots \\
\epsilon_{CMM} &= \frac{0.48+0.0+0.48+...+0.74}{15} \\
&= 0.1814 \\
\epsilon_{LeaFTrial1} &= 0.48; \\
\epsilon_{LeaFTrial2} &= 0.65780; \\
\epsilon_{LeaFTrial3} &= 0.48 \\
\vdots &= \vdots \\
\epsilon_{LeaF} &= \frac{0.48+.6578+0.48+...+0.67}{15} \\
&= 0.5221
\end{aligned}$$

Discussion

Post-experimental analysis of the data provides the designer with a useful tool for understanding the capabilities and limitations of a multi-robot system. Table 6.8 compares the system performance metric for CMM and LeaF. The tabulated values also show a marked improvement in the per task measures of robustness, and efficiency. As mentioned earlier, the values for the last 4 trials show CMM to perform much better than LeaF. Beyond the initial diagnosis, node selection in CMM can be considered to be random. This means in certain cases, based on implementation, CMM will perform better than LeaF, but in the long run LeaF will outperform CMM. It is important to note that the value for robustness is misleading. As, we have seen from SDR, the larger the number of interacting components the higher the probability of failure. We believe eventually LeaF will also encounter fault(s) that it does not handle well, similar to the sonar case in the box pushing task.

Similar to the box pushing task, it is difficult to extract learning information directly from the performance values. So, in order to understand the learning exhibited by the system, we examine the timing information for diagnosis for the different individual faults.

Table 6.8: Evaluation table for CMM and LeaF

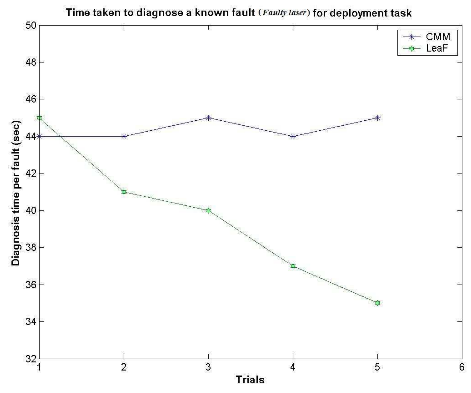
System	**P** $\in (-\infty, 1]$	$\rho \in [0,1]$ **per trial**	$\epsilon \in [0,1]$ **per trial**	δ_{known} **per trial**	$\delta_{unknown}$ **per trial**
CMM	0.1938	0.6133 per trial	0.1814 per trial	0.0	0.0
LeaF	0.855	1.0	0.5221	0.1	1.39

Table 6.9: Time taken to diagnose faults for deployment task

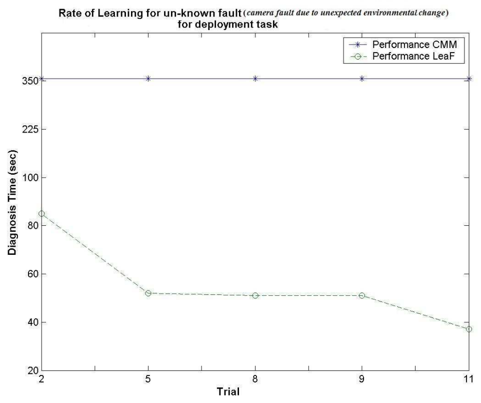
Fault name	**Trial number**	**Time (secs) for CMM**	**Time (secs) for LeaF**
faulty laser	1	44	45
	2	44	41
	3	45	40
	4	44	37
	5	45	35
camera failure due to environmental changes	1	360+	65
	2	360+	32
	3	360+	17

The timing information over the set of trials for two such faults, one of known type (*Laser fault*) and the other of unknown type (*camera fault due unexpected environmental change*) are illustrated in Table 6.9. Figures 6.20(a) and 6.20(b) plot the diagnosis curve for two sets of faults *laser fault* and *camera fault due to environmental changes*. For the *laser fault*, the diagnosis times for CMM do not decrease over the course of trials and remain steady at best. However, in the case of LeaF, it is interesting to note that the system uses experience as a means to improve performance steadily over trials. The more interesting case study is the one correlating to *camera fault due to environmental changes*. From the graph in Figure 6.20(b) we can see that after a few instances of the unknown fault, LeaF is able to better diagnose the problem. However, what is hidden is the fact that LeaF is constantly learning about the new fault by updating the POC values. Every time the POC becomes large enough for it precede an alternate edge, we see a jump in the diagnosis times. The actual times only represent the quality of implementation and not the quality of learning. We can hypothesize that over the course of future trials the system would show significant improvement in performance, before stabilizing on the fastest possible diagnosis implemented.

It is also worthwhile to monitor the speed of stabilization for LeaF. Table 6.10 shows the adaptation of the probability values in the second fault case (i.e., camera failure due to environmental changes when the lights are turned off), along with the time taken to diagnose the fault. The data shows a rapid change in the probability values corresponding to the increased likelihood of this particular failure occurring.



(a) Tracing diagnosis time over the set of trials for deployment task.



(b) Tracing diagnosis time for unknown type of fault for deployment task.

Figure 6.20: Performance illustrated over the period of operation for CMM and LeaF.

Table 6.10: Probability adaptation for environmental state change (light being switched off).

Fault name	**Trial number**	**No. of faults**	**Starting probability (%)**	**Ending probability (%)**	**Time taken to diagnose fault (secs)**
camera failure due to environmental changes	1	5	0	33	65
	2	5	33	47	24
	3	6	47	63	17
	4	4	63	76	17
	5	6	76	84	17
	6	2	63	88	17

While this may be desirable in the case of permanent environmental state changes, it also has the potentially negative effect of causing the team to forget the fault probability profile for the previous environmental state (i.e., when the lights are on). Finally, from the obtained data we can infer that the fault-tolerance architecture LeaF adapts faster and more efficiently towards achieving system goals and is a more suitable system for the multi-robot deployment task. Based on the analyzed data and our prior work with diagnostic systems [Parker and Kannan, 2006], it is our belief that the performance trend for an intelligent system would steadily improve over time once the system has gained experience about the most common previously un-modeled faults. Finally, we make the inference that the CMM system will perform well for static situations, but the performance starts degrading when any dynamic changes are made to the operating environment, whereas the LeaF is a more competent architecture for static as well dynamic systems.

6.7 Comparing metrics

As metrics are highly subjective, it is useful to compare any newly designed metrics with other existing ones for well defined, standard applications. This helps illustrate the usefulness, or the lack of, for the metrics. A review of the related work in the field of fault-tolerance metrics identified only one alternate work that could be potentially used for measuring the extent of fault-tolerance system performance exhibited by a multi-robot system. In this section, we apply Hamilton-Walker-Bennett metric to the obtained results from the two different physical robot experiments. Subsequently, we compare the two metrics and analyze the effectiveness for each one in helping a designer better understand an implemented system.

The metric is defined as follows:

$$HWB_{\mathit{eff}} = k_1(f)^2 + k_2(p)^2; \tag{6.1}$$

where k_1 and k_2 are normalizing constants, f is the redundancy based system fault-tolerance and p is the system performance. The calculated value for effective performance lies between the ranges of $[0, 1]$, with 0 indicating an in-effective system and 1 represents a system with an ideal balance of fault-tolerance and performance. Translating to the multi-robot domain,

$$HWB_{\mathit{eff}} = \sum_{j=1}^{m} u_j(\rho_{redundancy_j})^2 + c_j\left(1 - \frac{\frac{t_{diagnosis_j} + t_{Recovery_j}}{t_j} + 1}{2}\right)^2 \tag{6.2}$$

where m is the total set of tasks in the application, $\rho_{redundancy_j}$ is the correlating redundancy based fault-tolerance, and t_j is the total task execution time.

As the HWB metrics were originally designed for the multi-processor domain, in order to apply them to the results from the multi-robot domain, certain interpretations need to be made. These include:

- Each sub-system from the multi-processor environment can be equated to sub-tasks in the multi-robot environment,

- The normalizing constants used for determining the extent of fault-tolerance or performance can be mapped to the defined task-utility table,
- Only redundancy based faults can be used to calculate the fault-tolerance,
- System fault-tolerance is the summation of fault-tolerance across all individual subsystems or sub-tasks,
- Performance speed is inversely mapped to the system efficiency from our metrics model, and
- Finally the cost value is always set to 1, as we never take into consideration the actual physical cost towards overall system performance.

purely empirical nature of the developed metrics towards a more exact structure of metric definition.

We apply the HWB metric to the results obtained from the two physical robot experiments. We compare the obtained values with our metric and attempt to theorize if each of the metrics adequately explains the implemented systems. Similar to the way we fleshed out our metrics, HWB can be calculated. Hence, for box pushing task,

$$\begin{aligned}
HWB_{eff} &= \frac{HWB_{CMMTrial1}+HWB_{CMMTrial2}+HWB_{CMMTrial3}+...HWB_{CMMTrial5}}{15} \\
HWB_{eff} &= \frac{1.0+1.0+0.5+...+1.0}{15} = 0.3466 \\
HWB_{eff} &= \frac{1.0+1.0+0.5+...+1.0}{15} = 0.4232
\end{aligned}$$

Towards a fair comparison, we scale our metrics values to the same range ($[0, 1]$) as the *HWB* measure. We illustrate the calculated performance values for both systems in Table 6.11. Comparing the two metric values, it immediately becomes clear that the HWB metric does a poor job of distinguishing between the two systems. Numerically speaking, the values for HWB metric are lower than those of of our metric. The low numerical values are not a fair representation of the two systems. We can also infer from the table that for the HWB metric the granularity of distinction between the two systems is not nearly as high as that of our metric. The extent of relative difference becomes important when comparing similar types of systems.

Table 6.11: Calculated value for effective fault-tolerance for both metrics for the different test systems

Experiment	**System**	**HWB**	**Kannan-Parker**[‖]
Box pushing	CMM	0.3466	0.594
Box pushing	LeaF	0.4232	0.8122
Deployment	CMM	0.5548	0.5718
Deployment	LeaF	0.6830	0.9043

[‖]A logarithmic scale of the form $x = \log_2 y + 1$ is used to the scale the metrics to the range $[0, 1]$

The predominant reason for this is by considering purely redundant faults, the system does not account for other types of coordination or operational failures. In the case of a redundancy-based manipulator system, a larger number of team members results in an improved value for redundancy and as a consequence a higher performance rating, whereas for a multi-robot system as the complexity of the system increases, the performance curve goes down. Additionally, the HWB metric does not provide a means to evaluate and measure the adaptability or learning exhibited by a multi-robot system. This is reflected in the negative performance values for LeaF for both tasks. A negative measure is generally a good indicator for performance degradation over time or trials, whereas based on our analysis we know that the performance of LeaF improves over the course of the trials. As the HWB metric is a single quantitative measure, it cannot identify the influence of learning towards improving system performance exhibited by each system. From Murphy's hypotheses we know that learning constitutes an integral part of a robust fault-tolerance architecture and an inability to measure it prevents the metric from truly evaluating the capabilities of each system. As a consequence, a designer looking purely at the numerical values will not be able to determine which of the two systems, for either task, performs better. Hence, we can claim that the HWB metric in its current form is infeasible to be used for the multi-robot domain.

6.8 Conclusions and Future work

In this chapter, we have presented results from extensive physical robot implementations that compare and contrast CMM and LeaF. The subsequent analysis shows that a static causal model approach, like CMM, is insufficient for guaranteeing robust solutions, since it is practically impossible to pre-define all possible error modes in a complex multi-robot application. These results also show the need for a more adaptive approach (LeaF) for handling diverse fault conditions.

In addition, as new techniques in fault-tolerance are being explored, existing methods do not provide a complete measure of system performance for multi-robot teams. In this chapter, we evaluate two multi-robot applications based on the defined metrics. Specifically, the research provides a quantitative measure for identifying system fault-tolerance in terms of efficiency, robustness and the extent of learning. The research also provides the designer with analytical methods for understanding the metrics and the implemented system.

Chapter 7

Summary and Conclusions

7.1 Summary of Contributions

As part of my dissertation, I make several major research contributions to the multi-robot community. The most significant one is the development of an adaptive causal model method (adaptive CMM) for fault diagnosis and recovery in complex multi-robot teams called LeaF. It is particularly useful for cooperating teams of multi-robot systems interacting in a dynamic environment. The specific objectives of LeaF include:

1. Using experience as a means to better understand the system towards streamlining future fault diagnosis,
2. Using the full extent of available knowledge via the causal model to diagnose previously un-encountered faults, and
3. Absorbing the new information back into the existing causal model for future reference.

The LeaF approach, along with its predecessor CMM approach, has been implemented on two different physical robot experiments involving multi-robot navigation and deployment and cooperative multi-robot box pushing. In addition, the proof of concept for LeaF was implemented by means of a simulation setup, where the user had the option for controlling the type of faults encountered in the system.

The implemented results along with subsequent analysis have shown the following:

- How the causal model approach can be implemented in a large-scale multi-robot team and successfully enable the team to diagnose and recover from pre-defined errors. We have presented results from extensive physical robot implementations that illustrated the effectiveness of this approach.
- That a static causal model approach is insufficient for guaranteeing robust solutions, since it is practically impossible to pre-define all possible error modes in a complex multi-robot application, and
- That an adaptive causal model method like LeaF is robust enough to handle different types of failures in a dynamic environment.

Another significant contribution of my research is the development of metrics to measure fault-tolerance within the context of system performance. In addition, I have also outlined potential methods to better interpret the obtained metrics towards understanding the capabilities of the implemented system. The developed metrics are designed to be application independent and can be used to evaluate and/or compare different fault-tolerance architectures. Furthermore, a main focus of my approach is to capture the effect of intelligence, reasoning, or learning on the *effective* fault-tolerance of the system, rather than relying purely on traditional redundancy based measures.

The unique feature of LeaF, its ability to learn from the encountered faults towards future fault tolerance, provides a foundation for developing a turn-key solution for a fault diagnostic system for heterogeneous multi-robot teams performing complex tasks. Additionally, the presented evaluation metric provides the designer with numerical as well as analytical methods for understanding the intricacies of a fault-tolerant architecture. To the best of my knowledge, this is the first work that attempts to build a fault-tolerance architecture from the perspective of a multi-robot system. Finally, I also believe my metric is the first such one that attempts to evaluate the quality of learning towards understanding system level fault-tolerance.

7.2 Future Work

Several promising open research problems fall within the scope of research on multi-robot fault-tolerant system:

- *Taking advantage of the distributed nature of the approach.*

  Though the architecture can be developed as a distributed one, we do not take full advantage of the strengths of the distributed system. In addition to cooperating with each other to complete a task, robots could cooperatively solve faults. This would further expand the search area without significantly increasing the causal model size or taxing a single robot's resources. Extending the system to multiple robots could result in grids of cooperating robots capable of solving large complex problems in a short period of time. I have done some initial testing on using distributed sharing of fault diagnostic resources and hope to address the problem in the near future.

- *Adding information as nodes in addition to edges of the causal model.*

  A limitation of the current approach is its inability to solve for un-diagnosed faults that cannot be mapped to any of the existing ones under the causal model; i.e., the encountered case lies outside the existing solution set. Currently, this situation is handled by transferring all available information to a human for further evaluation. This situation could be potentially overcome in some cases if the robot had the capability of adding nodes in addition to edges within the causal model.

- *Adding predictive capabilities to the model using something like a HMM or other Bayesian model.*

  Prevention is better than cure. The purpose of the fault prediction would be to anticipate fault(s) before their occurrence based on the similitude between the current normal state and a prior fault state. By inferring the resultant similitude set in terms of a known environment component, such as a map update feature, the robot might be able to prevent the occurrence of a fault.

- *Human-robot interaction.*

  The ability to utilize human knowledge to solve previously un-solvable faults would greatly improve system efficiency and robustness. This involves determining the extent of human interaction — when, where and how often — to balance the efficiency and robustness of the system. To incorporate an extensive human-robot interaction is beyond the scope of this research; however I would like to explore the possibility at a later date.

Bibliography

Bibliography

[Aamodt and Plaza, 1994] Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. In *AICom - Artificial Intelligence Communications*, volume 7, pages 39–59.

[Armengol and Plaza, 2001a] Armengol, E. and Plaza, E. (2001a). Lazy induction of descriptions for relational case-based learning. In L. De Raedt, P. F., editor, *Machine Learning: EMCL 2001*, Lecture Notes in Artificial Intelligence, pages 13–24. Springer-Verlag.

[Armengol and Plaza, 2001b] Armengol, E. and Plaza, E. (2001b). Similarity assessment for relational cbr. In *Case-based reasoning research and development: ICCBR 2001*, Lecture notes in artificial intelligence, pages 44–58. Springer-Verlag.

[Armengol and Plaza, 2005] Armengol, E. and Plaza, E. (2005). Using symbolic similarity to explain cbr in classification tasks. In *Proceedings of AAAI fall symposium on explanation-aware computing*, pages 1–9. AAAI Press.

[Atkins et al., 1997] Atkins, E. M., Durfee, E. H., and Shin, K. G. (1997). Detecting and reacting to unplanned-for world states. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI)*, pages 571–576.

[Beetz and McDermott, 1997] Beetz, M. and McDermott, D. (1997). Expressing transformations of structured reactive plans. In *Recent Advances in AI Planning. Proceedings of the 1997 European Conference on Planning*, pages 64–76. Springer Publishers.

[Bongard and Lipson, 2004] Bongard, J. and Lipson, H. (2004). Automated damage diagnosis and recovery for remote robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3545–3550.

[Borner et al., 1996] Borner, K., Pippig, E., Tammer, E., and Coulon, C. (1996). Structural similarity and adaptation. In Smith, I. and Faltings, B., editors, *Proceedings of the third european workshop on case-based reasoning EWCBR-96*, volume 1168 of *Lecture notes in computer science*, pages 58–75. Springer.

[Broxvall et al., 2004] Broxvall, M., Coradeschi, S., Karlsson, L., and Saffiotti, A. (2004). Have another look: On failures and recovery planning in perceptual anchoring. In *Proceedings of the ECAI-04 Workshop on Cognitive Robotics*, Valencia, ES. Online at http://www.aass.oru.se/~asaffio/.

[Carlson and Murphy, 2003] Carlson, J. and Murphy, R. R. (2003). Reliability analysis of mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

[Carlson and Murphy, 2004] Carlson, J. and Murphy, R. R. (2004). An investigation of MML methods for fault diagnosis in mobile robots. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS).*

[Carlson and Murphy, 2005] Carlson, J. and Murphy, R. R. (2005). How UGVs physically fail in the field. *IEEE Transactions on Robotics*, 21(3):423–437.

[Cavallaro and Walker, 1994] Cavallaro, J. and Walker, I. (1994). A survey of NASA and military standards on fault tolerance and reliability applied to robotics. In *Proceedings of AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space*, pages 282–286.

[Decker, 1996] Decker, K. (1996). TAEMS: A framework for environment centered analysis & design of coordination mechanisms. In *Foundations of Distributed Artificial Intelligence*, pages 429–448. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science.

[Donald et al., 1994] Donald, B., Jennings, J., and Rus, D. (1994). Analyzing teams of cooperating mobile robots. In *Proceeding of IEEE conference on robotics and automation*, pages 1896–1903.

[Dudek et al., 1993] Dudek, G., Jenkin, M., Milios, E., and Wilkes, D. (1993). Robust positioning with a multi-agent robotic system. In *Proceedings of IJCAI-93 Workshop on Dynamically Interacting Robots*, pages 118–123.

[Evans and Messina, 2000] Evans, J. and Messina, E. (2000). Performance metrics for intelligent systems. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II.

[Finkelstein, 2000] Finkelstein, R. (2000). A method for evaluating IQ of intelligent systems. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II.

[Gambardella and Versino, 1994] Gambardella, L. and Versino, C. (1994). Learning high-level navigation strategies from sensor information and planner experience. In *Proceedings of Perception to Action (PerAc'94)*, Chicago.

[Gerkey et al., 2003] Gerkey, B., Vaughan, R., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems.

[Hamilton et al., 1996] Hamilton, D., Walker, I., and Bennett, J. (1996). Fault tolerance versus performance metrics for robot systems. In *Proceedings of the IEEE International Conference on Robotics and Automation(ICRA)*, pages 3073–3080.

[Hamilton et al., 2001] Hamilton, K., Lane, D., Taylor, N., and Brown, K. (2001). Fault diagnosis on autonomous robotic vehicles with recovery: An integrated heterogeneous-knowledge approach. In *Proceedings of the IEEE International Conference on Robotics and Automation(ICRA)*, pages 3232–3237.

[Horling et al., 2000a] Horling, B., Lesser, V., Vincent, R., Bazzan, A., and Xuan, P. (2000a). Diagnosis as an integral part of multi-agent adaptability. In *Proceedings of DARPA Information Survivability Conference and Exposition*, pages 211–219.

[Horling et al., 2000b] Horling, B., Lesser, V. R., Vincent, R., Bazzan, A., and Xuan, P. (2000b). Diagnosis as an integral part of multi-agent adaptability. In *Proceedings of DARPA Information Survivability Conference and Exposition*, pages 211–219.

[Howard et al., 2006] Howard, A., Parker, L. E., and Sukhatme, G. S. (2006). Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment, and detection. *International Journal of Robotics Research*, 25:431–447.

[Hudlická and Lesser, 1987] Hudlická, E. and Lesser, V. R. (1987). Modeling and diagnosing problem-solving system behavior. *IEEE Transactions on Systems, Man, and Cybernetics*, 17:407–419.

[Jacoff et al., 2003] Jacoff, A., Messina, E., Weiss, B., Tadokoro, S., and Nakagawa, Y. (2003). Test arenas and performance metrics for urban search and rescue robots. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 1016–1022.

[Kaminka and Tambe, 1998] Kaminka, G. A. and Tambe, M. (1998). What is wrong with us? Improving robustness through social diagnosis. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 97–104.

[Kolodner, 1993] Kolodner, J. (1993). *Case-based reasoning*. Morgan Kaufmann Publishers.

[Lamine and Kabanza, 2000] Lamine, K. and Kabanza, F. (2000). History checking of temporal fuzzy logic formulas for monitoring behavior-based mobile robots. In *Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 312–319.

[Le, 2002] Le, C. (2002). System transparency through self-explanation. Technical report, AAAI Fall Symposium, North Falmout, Massachusetts.

[Lee et al., 2000] Lee, S., Bang, W., and Bien, Z. (2000). Measure of system intelligence: An engineering perspective. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II.

[Lindsay et al., 1980] Lindsay, R. K., Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J. (1980). *Applications of artificial intelligence for chemical inference: The Dendral Project*. McGraw-Hill, Inc., New York, NY, USA.

[Long et al., 2003] Long, M., Murphy, R. R., and Parker, L. E. (2003). Distributed multiagent diagnosis and recovery from sensor failures. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, pages 2506–2513.

[Madden, 1998] Madden, M. (1998). Hierarchically structured inductive learning for fault diagnosis. In *Proceedings of 13th International Conference on Applications of Artificial Intelligence in Engineering*, Galway.

[Madden and Nolan, 1999] Madden, M. and Nolan, P. (1999). Monitoring and diagnosis of multiple incipient faults using fault tree induction. *IEEE Proceedings - Control Theory and Applications*, 146:204–212.

[Mahadevan and Connell, 1991] Mahadevan, S. and Connell, J. (1991). Automatic programming of behavior-based robots using reinforcement learning. In *National Conference on Artificial Intelligence*, pages 768–773.

[Mahdavi and Bentley, 2003] Mahdavi, S. and Bentley, P. (2003). An evolutionary approach to damage recovery of robot motion with muscles. In *European Conference on Artificial Life (ECAL)*, pages 248–255.

[Mataric, 1993] Mataric, M. J. (1993). Designing emergent behaviors: From local interactions to collective intelligence. In *Animals to Animats 2: Proceedings of the second international conference on simulation of adaptive behaviour*, pages 432–441. MIT Press.

[Mataric, 1994] Mataric, M. J. (1994). Interaction and intelligent behavior. Technical Report AITR-1495.

[Murphy and Hershberger, 1996] Murphy, R. and Hershberger, D. (1996). Classifying and recovering from sensing failures in autonomous mobile robots. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI)*, volume 2, pages 922–929.

[Murphy and Hershberger, 1999] Murphy, R. R. and Hershberger, D. (1999). Handling sensing failures in autonomous mobile robots. *The International Journal of Robotics Research*, 18:382–400.

[Muscettola et al., 1997] Muscettola, N., Smith, B., Fry, C., Chien, S., Rajan, K., Rabideau, G., and Yan, D. (1997). Board planning for new millennium deep space one autonomy. In *Proceedings of the IEEE Aerospace Conference*.

[Parker, 1998] Parker, L. (1998). ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. In *IEEE Transactions on Robotics and Automation*, volume 14, pages 220–240.

[Parker, 2000] Parker, L. (2000). Lifelong adaptation in heterogeneous multi-robot teams: Response to continual variation in individual robot performance. *Autonomous Robots*, 8(3):239–267.

[Parker et al., 2004] Parker, L., Kannan, B., Tang, F., and Bailey, M. (2004). Tightly-coupled navigation assistance in heterogeneous multi-robot teams. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 1016–1022.

[Parker, 2001] Parker, L. E. (2001). Evaluating success in autonomous multi-robot teams: Experiences from ALLIANCE architecture implementations. *Journal of Theoretical and Experimental Artificial Intelligence*, 13:95–98.

[Parker and Kannan, 2006] Parker, L. E. and Kannan, B. (2006). Adaptive causal models for fault diagnosis and recovery in multi-robot teams. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*. Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS).

[Parker et al., 2003] Parker, L. E., Kannan, B., Fu, X., and Tang, Y. (2003). Heterogeneous mobile sensor net deployment using robot herding and line-of-sight formations. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*.

[Pell et al., 1997] Pell, B., Bernard, D., Chien, S., Gat, E., Muscettola, N., Nayak, P., Wagner, M., and Williams, B. (1997). An autonomous spacecraft agent prototype. In *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 253–261, New York. ACM Press.

[Plaza and Arcos, 2002] Plaza, E. and Arcos, J. (2002). Advances in case-based reasoning. In *Proceedings of 6th European Conference on Case-Based Reasoning, ECCBR-02*, Lecture notes in artificial intelligence, pages 306–320.

[Plaza et al., 1996] Plaza, E., Mntaras, R., and Armengol, E. (1996). On the importance of similitude: An entropy-based assessment. In Smith, I. and Faltings, B., editors, *Proceedings of the third european workshop on case-based reasoning EWCBR-96*, volume 1168 of *Lecture notes in computer science*, pages 324–338. Springer.

[Plotkin, 1970] Plotkin, G. (1970). A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press.

[Pouchard, 2000] Pouchard, L. (2000). Metrics for intelligence: a perspective from software agents. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II.

[Stancliff et al., 2006] Stancliff, S., Dolan, J., and Trebi-Ollennu, A. (2006). Mission reliability estimation for multi-robot team design. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*.

[Sugawara and Lesser, 1998] Sugawara, T. and Lesser, V. (1998). Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*.

[Sutton, 1998] Sutton, R. S. (1998). Reinforcement learning: Past, present and future. In *Second Asia-Pacific Conference on Simulated Evolution and Learning (SEAL)*, pages 195–197.

[Tang, 2006] Tang, F. (2006). *ASyMTRe: Building coalitions for heterogeneous multi-robot teams*. Dissertation, University of Tennessee, Knoxville.

[Tang and Parker, 2005] Tang, F. and Parker, L. (2005). ASyMTRe: Automated synthesis of multi-robot task solutions through software reconfiguration. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

[Torras, 1994] Torras, C. (1994). Neural learning for robot control. In *13th European Conference on Artificial Intelligence (ECAI)*, pages 814–822.

[Toyoma and Hager, 1997] Toyoma, K. and Hager, G. D. (1997). "if at first you don't succeed ...". In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI)*, pages 3–9.

[Vincent et al., 2000] Vincent, R., Horling, B., and Lesser, V. (2000). Experiences in simulating multi-agent systems using TAEMS. *The Fourth International Conference on MultiAgent Systems (ICMAS 2000).*

[Visinsky, 1991] Visinsky, M. (1991). Fault detection and fault tolerance methods for robotics. Master's thesis, Rice University.

[Wang and Dai, 1999] Wang, H. and Dai, G. (1999). Open architecture framework for distributed real-time fault diagnosing. In *Proceedings of the SPIE - The International Society for Optical Engineering*, pages 93–100.

[Watkins, 1989] Watkins, C. (1989). *Learning from Delayed Rewards.* PhD thesis, King's College, Oxford.

[Wermter et al., 2004] Wermter, S., Weber, C., Elshaw, M., Panchev, C., Erwin, H., and Pulvermller, F. (2004). Towards multimodal neural robot learning. *Robotics and Autonomous Systems Journal.*

[Yavnai, 2000] Yavnai, A. (2000). Metrics for system autonomy. Part I: Metrics definition. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II.

Appendices

Appendix A

Tabulation of the rule-base relationship for the experiments

Table A.1: Rule-base for Phase I CMM of SDR describing the symptom-fault relation and the associated corrective actions

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
All behaviors	Goal_not_reached	Time out in goto goal behavior	Perform path planner test	Reset path planner and re-attempt goto_goal behavior
	Camera_error	Image not found	Perform camera tests	If fault persists, leave simple robot(s) in wait state, send camera failure feedback to human operator and return home
	Lost_follower	Missing robot	Establish communication with simple robot and instruct it to perform motor tests	Check if simple robot is close enough to goal; if so, change simple robot state to sensor detection, and return home

Table A.1 – continued from previous page

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
	Path_planning error	Stuck in path planning algorithm	Check starting position and ending position	Re-attempt path planning.
	Localization_errors	Inconsistent pose information	Perform laser test, attempt to re-calibrate laser	Check if simple robot is close enough to goal; if so, change simple robot state to sensor detection, send laser data to human and return home
	Communication	No acknowledgment signal	Attempt transmission to simple robot, other robots and human operator	Move away from current location and re-attempt transmission; If communication error persists, return home
	Bad_Initialization	Incorrect starting pose	Perform position test	Reset starting pose values
	Faulty_sensor	Bad sensor readings	Perform sensor test	Alert human to replace sensor
	Motor_problems	Motor test failed	Perform simple robot status check	Change simple robot state to sensor detection and proceed

Table A.1 – continued from previous page

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
	Unexpected environmental behavior	Inconsistent map information	Compare current environment state with known map information for deviations	If simple robot state close to goal, change state to sensor detection and proceed; else, leave simple robot in wait state and proceed

Table A.2: Rule-base for Phase II CMM of SDR describing the symptom-fault relation and the associated corrective actions

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
Path planning	Invalid_positions	Invalid position information	Check starting and goal positions	Re-attempt path planning
	Goal_not_reached	Time out in goto goal behavior	Check sensor data to ensure all sensors are working correctly	Re-attempt goto_goal behavior
	Bad_starting_position	Incorrect starting position	Check starting position information from initial config file	Reset starting position and re-attempt path_planning
	Bad_goal_position	Incorrect goal position	Check goal position from initial config file	Reset goal position and re-attempt path_planning

Table A.2 – continued from previous page

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
	Localization_errors	Inconsistent pose information	Perform laser test, recalibrate laser	Check if simple robot is close enough to goal; if so, change simple robot state to sensor detection and inform human to replace laser
	Translation_errors	Inconsistent pose information	Request human user to check the translation algorithm for logical inconsistencies	Reset algorithm and re-attempt behavior
	Invalid_map	Map not found	Perform map test	Request human operator to replace map
	Bad_Initialization	Incorrect starting pose	Test position values	Reset starting pose values
	Faulty_sensor	Bad sensor readings	Perform sensor test	Alert human to replace sensor
Task assignment	Coordination_error	Missing robot	Pan around using camera searching for robot, communicate with other robots for information on lost robot	Change simple robot state to sensor detection and return home
	Communication_error	No acknowledgment signal	Attempt transmission to simple robot, other robots and human operator	Move away from current location and re-attempt transmission; If communication error persists, return home

Table A.2 – continued from previous page

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
	Physics_of_comm	Change in environment state	Attempt transmission to simple robot, other robots and human operator, if error persists compare current environment state with known prior information for deviations	Request human user to reset communication
	Protocol failure	Inconsistent communication feedback data	Perform communication protocol test	Request human user to reset protocol
	Error_due_to _environment	Change in environment state	Check current environment state with past states	Explore neighboring area, re-establish connection with other robots
Goto goal behavior	Deadlock	Time out in goto goal behavior	Coordinate with other robots, update position information and re-apply traffic management algorithm	Transfer control to human to solve deadlock
	Obstacle_avoidance	Time out in goto goal behavior	Perform motion test in all directions to check for obstacle	Communicate and coordinate with other robots to identify and navigate for un-expected obstacles in the path

Table A.2 – continued from previous page

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
	Traffic_management	Time out in goto goal behavior	Coordinate with other robots, update position information and perform traffic management test	Request human user to reset traffic-management algorithm
	Algorithmic failure	Inconsistent data upon testing	Run simple algorithm test	Request user to replace algorithm
	Hardware_failure	No sensor data	Perform simple hardware test	Alert human to replace hardware
Follow the leader	Blob_tracking_error	Blob image not found	Camera test	Coordinate with helper robot and attempt to use the camera to find blob
	Motor_problems	Increasing distance of separation between helper and following simple robot	Establish communication with simple robot and instruct it to perform motor tests	Check if simple robot is close enough to goal; if so, change simple robot state to sensor detection, and return home
	False_positives	Incorrect tracking information	Take image, repeat process and perform quality measure test	Coordinate and realign simple robot to track the correct blob

Table A.2 – continued from previous page

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
	Error_due_to _environment	Change in environment state	Compare current environment state with known map information for deviations	Check if simple robot is close to goal; if so, change simple robot state to sensor detection and proceed; else, leave simple robot in wait state and proceed
	Uneven_lighting	Sporadic blob tracking	Perform camera light test	Reset light parameters and allow for longer sensor reading times
	Acoustic_reflection	Noisy sensor data	Perform sensor test, repeat process and perform quality measure test	Average over multiple readings
	Robot_interference	Incorrect blob tracking	Test camera, test blob tracking	Coordinate with simple robot, realign robot and reset blob tracking
	Battery_issues	Battery alarm	Perform battery status test	Go back to starting position for battery recharge
	Hardware_reset	Incorrect current data	Perform hardware test	Reinitialize sensor from last stored state information
Tele operation	Helper_robot_error	Exception in Helper robot behavior	Perform hardware, sensor test and algorithm test	Trace through and identify source of helper exception

Table A.2 – continued from previous page

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
	Simple_robot_error	Exception in simpler robot behavior	Perform hardware sensor test	Trace through and identify source of simple robot exception
	Mismatched_speed	Error in speed synchronization	Perform distance measure test over a fixed time period	Coordinate with Helper robot to maintain formation
Acoustic sensing	Calibration_issues	Incorrect data	Perform simple sensor test	Re calibrate acoustic sensors from initialization files

Table A.3: Rule-base for box pushing task describing the symptom-fault relation and the associated corrective action

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
Mis alignment	Sensor malfunction	Incorrect sensor information	Perform correlating sensor tests	Report to human unable to accurately identify troublesome sensor
	Sonar	Inconsistent sensor information	Cross verify pose information from sonar with that of laser	Switch to alternate sensor (laser)
	Laser	Inconsistent sensor information	Perform laser test and check sensor data with that of team member	Align to blob, backup away from box and inform failure to team member
	Motor	Time out in goto_goal	Perform simple motor test	Indicate to human and team member and quit

Table A.3 – continued from previous page

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
	Wireless	Unable to contact team member	Perform wireless test with external member (human)	Backup and go back home
	Algorithmic Inconsistency	Unable to complete test	Cross check sensor information with human and team member	Inform human, and ask for help
	Goal not reached	Timeout in going to goal	Perform algorithm test	Unable to diagnose failure and return home
Blob tracking	missing blob	Unable to find blob	Perform sensor test	Get blob information from team
	Camera	Unable to find blob	Perform blob-test	Get blob information from team member
Communication issues	Physics of communication	Unable to contact team member	Perform synchronization test	If fault persists, turn back and go home, else re-synchronize with team member
	"Partner" robot error	Missing blob information	Establish communications with partner robot and attempt to obtain blob information	Communicate to team member and return home

Table A.4: Rule-base for deployment task describing the symptom-fault relation and the associated corrective action

Behavior module	**Fault Node**	**Symptom**	**Generate-and-test strategy**	**Corrective action**
Path planning	Goal_not_reached	Time out in goto goal behavior	Perform path planner test	Reset path planner and re-attempt goto_goal behavior
	Invalid positions	Invalid position information	Check starting and goal positions	Re-attempt path planning
	Bad_starting position	Incorrect starting position	Check starting position information from initial config file	Reset starting position and re-attempt path_planning
	Bad_goal position	Incorrect goal position	Check goal position from initial config file	Reset goal position and re-attempt path_planning
	Localization errors	Inconsistent pose information	Perform laser test, re-calibrate laser	Check if simple robot is close enough to goal; if so, change simple robot state to sensor detection and inform human to replace laser
	Translation errors	Inconsistent pose information	Request human user to check the translation algorithm for logical inconsistencies	Reset algorithm and re-attempt behavior
	Invalid_map	Map not found	Perform map test	Request human operator to replace map

Table A.4 – continued from previous page

Behavior module	Fault Node	Symptom	Generate-and-test strategy	Corrective action
	Bad_Initialization	Incorrect starting position	Test position values	Reset starting pose values
	Faulty_sensor	Bad sensor readings	Perform sensor test	Alert human to replace sensor
Tele operation	Camera error	Image not found	Perform camera tests	If fault persists, leave simple robot(s) in wait state, send camera failure feedback to human operator and return home
	Lost_follower	Missing robot	Establish communications with simple robot and instruct it to perform motor tests	Check if simple robot is close enough to goal; if so, change simple robot state to sensor detection, and return home
	Motor problems	Motor test failed	Perform simple robot status check	Change simple robot state to sensor detection and proceed

Appendix B

Tabulation of experimental results

Table B.1: Fault diagnosis data for CMM and LeaF for the box pushing task

Trial No.	**Module**	**Faults (k/u)**		**Diagnosed (k/u)**		**Additional Notes**
		CMM	LeaF	CMM	LeaF	
1	No Error	0	0	0		No errors encountered during operation
2	No Error	0	0	0		No errors encountered during operation
3	Communication	5 (4/1)	2 (2/0)	5 (3/2)	5(3/2)	Synchronization issues and unanticipated interference from alternate laptop
4	Communication	2 (2/0)	2 (2/0)	3 (3/0)	3 (3/0)	Synchronization issues among robots
4	Box push	1(1/0)	1(1/0)	1(1/0)	1(1/0)	Sonar data filtered out
5	Communication	4(4/0)	4(4/0)	3(3/0)	3(3/0)	Synchronization issues among robots
5	Box push	1 (0/1)	0(0/0)	1(0/1)	0(0/0)	Mis-aligned box leading to robot continuing the pushing task without box

Table B.1 – continued from previous page

Trial No.	Module	Faults (k/u)		Diagnosed (k/u)		Additional Notes
		CMM	LeaF	CMM	LeaF	
6	Communication	4(4/0)	4(4/0)	1(1/0)	1(1/0)	Synchronization issues among robots during communication
7	Communication	2(2/0)	2(2/0)	1(1/0)	1(1/0)	Synchronization issues among robots during communication
8	Blob tracking	1(1/0)	1(1/0)	1(1/0)	1(1/0)	Malfunctioning camera
8	Communication	3(3/0)	3(3/0)	2(2/0)	2(2/0)	Synchronization and wireless issues among robots
9	Communication	3(2/1)	2(2/0)	3(2/1)	3(2/1)	Synchronization issues and unanticipated interference from alternate laptop
10	Communication	4(4/0)	4(4/0)	1(1/0)	1(1/0)	Synchronization issues among robots
10	Blob tracking	1(0/1)	0(0/0)	1(0/1)	1(0/1)	Un-expected lighting change
11	Communication	1(0/1)	0(0/0)	1(0/1)	1(0/1)	Human error
12	Communication	3(3/0)	3(3/0)	2(2/0)	2(2/0)	Synchronization issues among robots
12	Blob tracking	1(0/1)	0(0/0)	5(0/5)	5(0/5)	Uneven lighting
13	Communication	2(2/0)	2(2/0)	1(1/0)	1(1/0)	Synchronization issues among robots
13	Blob tracking	1(0/1)	0(0/0)	3(3/0)	3(3/0)	Uneven lighting and un-expected changes
14	Communication	3(3/0)	3(3/0)	2(2/0)	2(2/0)	Synchronization issues among robots during communication
15	No Error	0	0	0		No errors encountered during operation

Table B.2: Time taken to diagnose faults for box pushing task

Module	**Trial number**	**Diagnosis Time (s)**		**Diagnosis**	
		CMM	LeaF	CMM	LeaF
No fault	1,2,14	0	0	None	
Communication	3	600	188	Not classified	Unexpected environment change
Communication	4	47	50	Synchronization issues	Synchronization issues
Box push	4	30	30	sonar fault	sonar fault
Box push	5	600	600	Not classified	Unexpected environment change
Communication	5	47	50	Synchronization issues	Synchronization issues
Communication	6,7	47	42	Synchronization issues	Synchronization issues
Communication	8	47	40	Synchronization issues	Synchronization issues and wireless fault
Blob tracking	8	40	50	camera fault	camera fault
Communication	9	600	170	Not classified	Unexpected environment change
Communication	10	50	40	Synchronization issues	Synchronization issues
Blob tracking	10	600	157	Not classified	Unexpected environment change

Table B.2 – continued from previous page

Module	**Trial number**	**Diagnosis Time (s)**		**Diagnosis**	
		CMM	LeaF	CMM	LeaF
Communication	11	600	55	Not classified	Unexpected environment change
Communication	12	53	28	Synchronization issues	Synchronization issues
Blob tracking	12	600	85	Not classified	Unexpected environment change
Communication	13	53	24	Synchronization issues	Synchronization issues
Communication	14	50	24	Synchronization issues	Synchronization issues
Blob Tracking	13	600	85	Not classified	Unexpected environment change

Table B.3: Execution and diagnosis time for box pushing task

Trial number	**Diagnosis Time (s)**		**Execution Time (s)**	
	CMM	LeaF	CMM	LeaF
1,2,15	0	0	40	40
3	600	390	600	430
4	600	600	600	600
5	280	230	320	270
6,7	160	126	200	166
8	220	95	260	135

9	600	210	600	250
10	600	280	600	320
11	600	55	600	95
12	600	184	600	244
13	600	182	600	232
14	160	24	200	64

Table B.4: Fault diagnosis data for CMM and LeaF for the deployment task

Trial No.	**Module**	**Fault (k/u)**		**Diagnosed (k/u)**		**Time(s)**		**Additional Notes**
		CMM	LeaF	CMM	LeaF	CMM	LeaF	
1	Localization	1(1/0)	1(1/0)	1(1/0)	1(1/0)	44	45	Laser was interfered with by a human standing directly in front of it.
1	Marker detection	3(3/0)	3(3/0)	2(2/0)	2(2/0)	50	47	Attempting to keep the simple robot in its line-of-sight resulted in the helper robot momentarily lost track of the simple robot.
1	Path planning	1(1/0)	1(1/0)	1(1/0)	1(1/0)	35	35	Laser was interfered with by a human standing directly in front of it.

Table B.4 – continued from previous page

Trial No.	Module	Faults (k/u)		Diagnosed (k/u)		Time(s)		Additional Notes
		CMM	LeaF	CMM	LeaF	CMM	LeaF	
2	Marker detection	1(0/1)	0(0/1)	2(2/2)	2(2/2)	360	65	The lighting in the room was tampered with by a human.
3	Marker detection	2(2/0)	2(2/0)	4(4/0)	4(4/0)	50	30	Attempting to keep the simple robot in its line-of-sight resulted in the helper robot momentarily lost track of the simple robot.
3	Localization	1(1/0)	1(1/0)	1(1/0)	1(1/0)	44	45	Laser was interfered with by a human standing directly in front of it.
4	Marker detection	3(2/1)	3(2/0)	2(2/0)	2(2/0)	50	27	Helper robot momentarily lost track of the simple robot.
4	Localization	1(1/0)	1(1/0)	1(1/0)	1(1/0)	44	45	Laser was interfered with by a human standing directly in front of it.
5	Marker detection	1(0/1)	0(0/0)	3(0/3)	3(0/3)	360	32	The lighting in the room was tampered with by a human.
6	Marker detection	2(2/0)	2(2/0)	1(1/0)	1(1/0)	50	28	Helper robot momentarily lost track of the simple robot.

Table B.4 – continued from previous page

Trial No.	**Module**	**Faults (k/u)**		**Diagnosed (k/u)**		**Time(s)**		**Additional Notes**
		CMM	LeaF	CMM	LeaF	CMM	LeaF	
6	Localization	1(1/0)	1(1/0)	1(1/0)	1(1/0)	44	45	Laser was interfered with by a human standing directly in front of it.
7	Localization	1(1/0)	1(1/0)	1(1/0)	1(1/0)	45	37	Improper initialization of pose information.
7	Navigation	1(0/1)	0(0/0)	1(0/1)	1(0/1)	360	52	Improper initialization of pose information.
8	Marker detection	2(1/1)	1(1/0)	1(1/0)	1(1/0)	360	0	The lighting in the room was tampered with
9	Marker detection	1(0/1)	0(0/0)	5(0/5)	5(0/5)	360	31	The lighting in the room was tampered with
10	Navigation	1(0/1)	0(0/0)	1(0/1)	1(0/1)	360	87	Motor problem at start up, motors were left in locked state.
11	Marker detection	1(0/1)	0(0/0)	1(0/1)	1(0/1)	360	17	The lighting in the room was tampered with by a human.
12	Blob tracking	1(1/0)	1(1/0)	1(1/0)	1(1/0)	30	59	Speed synchronization issues between robots
13	Blob tracking	1(1/0)	1(1/0)	2(2/0)	2(2/0) par	30	57	Speed synchronization issues between robots
14	Blob tracking	1(1/0)	1(1/0)	1(1/0)	1(1/0)	30	63	Speed synchronization issues between robots

Table B.4 – continued from previous page

Trial No.	**Module**	**Faults (k/u)**		**Diagnosed (k/u)**		**Time(s)**		**Additional Notes**
		CMM	LeaF	CMM	LeaF	CMM	LeaF	
15	Blob tracking	1(1/0)	1(1/0)	1(1/0)	1(1/0)	30	59	Speed synchronization issues between robots

Table B.5: Execution and diagnosis time for deployment task

Trial number	**Diagnosis Time (s)**		**Execution Time (s)**	
	CMM	LeaF	CMM	LeaF
1	130	129	250	249
2	360	360	65	190
3	130	250	115	235
4	131	251	107	227
5	360	360	32	155
6	136	260	108	230
7	360	360	80	200
8	360	360	32	155
9	360	360	186	328
10	360	360	87	215
11	360	360	17	143
12,13,14,15	30	154	60	179

Vita

Balajee Kannan joined the Department of Computer Science at the University of Tennessee-Knoxville (UTK) as Graduate Student in August 2000. He received the M.S. degree in computer science from UTK in 2003, and his B.E. degree in computer engineering from University of Madras, Madras, India in 2000.

He has been part of the Distributed Intelligent Laboratory at UTK since Fall of 2002, and was part of the successful Software for Distributed Robotics project funded by DARPA. His current research focuses on developing fault-tolerance techniques to enable multiple robots to succeed in dynamic environments and to subsequently evaluate them using application-independent metrics. Other research interests include human-robot cooperation, outdoor robotics, and social and educational robotics. He is a student member of the IEEE, ACM, UPE, SIGMA XI, and ADC. Post-graduation, he is looking forward to joining the Robotics Institute at the Carnegie Mellon University, in the capacity of a research engineer.