Proc. of IEEE International Conference on Intelligent Robots and Systems, San Diego, CA, 2007.

Metrics for quantifying system performance in intelligent, fault-tolerant multi-robot teams

Balajee Kannan
Field Robotics Center, Robotics Institute
Carnegie Mellon University
Email: bkannan@cs.cmu.edu

Lynne E. Parker
Distributed Intelligence Laboratory
Dept. of Electrical Engineering and Computer Science
University of Tennessee
Email: parker@eecs.utk.edu

Abstract— **Any system that has the capability to diagnose and recover from faults is considered to be a fault-tolerant system. Additionally, the quality of the incorporated fault-tolerance has a direct impact on the overall performance of the system. Hence, being able to measure the extent and usefulness of fault-tolerance exhibited by the system would provide the designer with a useful analysis tool for better understanding the system as a whole. Unfortunately, it is difficult to quantify system fault-tolerance on its own for intelligent systems. A more useful metric for evaluation is the "effectiveness" [6] measure of fault-tolerance. The influence of fault-tolerance towards improving overall performance determines the overall *effectiveness* or quality of the system. In this paper, we outline application-independent metrics to measure fault-tolerance within the context of system performance. In addition, we also outline potential methods to better interpret the obtained measures towards understanding the capabilities of the implemented system. Furthermore, a main focus of our 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. We show the utility of the designed metrics by applying them to different fault-tolerance architectures implemented for multiple complex heterogeneous multi-robot team applications and comparing system performance. Finally, we contrast the developed metrics with the only other existing method (HWB method [6]) for evaluating (that we are aware of) effective fault-tolerance for multi-robot teams and rate them in terms of their capability to best interpret the workings of the implemented systems. To the best of our knowledge, this is the first metric that attempts to evaluate the quality of learning towards understanding system level fault-tolerance.**

I. INTRODUCTION

Increasingly, multi-robot teams are being used in functionally-distributed missions that require complex coordination among multiple robots performing numerous tasks such as planning, information sharing, and so forth in highly dynamic and potentially hazardous operating environments. In the last decade, several researchers have studied fault-tolerance for robotic systems (e.g., [1], [7], [11], [10]). However, still missing from this research are standard metrics for evaluating new and existing multi-robot fault-tolerance methods. In the absence of accepted metrics, it is difficult for a designer to calculate the true capability of an existing system, compare two different fault-tolerant strategies, or attempt to evaluate a newly proposed strategy.

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 [6], it is incomplete as an evaluation metric, since a system can also be effectively fault-tolerant through other reasoning methods. Additionally, in our earlier work ([12]), we detailed empirical results that show the inability of a system to identify all faults that may occur without incorporating an experience based online-learner. As a consequence, we can say that for a system to be truly robust to change, it needs to exhibit adaptability to counter the dynamic nature of its operating environment. 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. Based on this analysis, this paper addresses the following problem: *Develop application-independent 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.* The metrics were initially outlined in [9], but were not meaningfully evaluated using experimental data. In this current paper, we detail and extend the originally proposed metrics and present extensive new experimental data to validate them.

In Section II, we present a brief review of existing work on performance metrics and discuss their drawbacks as fault-tolerance measures. Section III formally defines the proposed problem and details the derivation of the proposed metrics. In order to evaluate the validity of the metrics, we apply them to two distinct physical multi-robot experiments in Section IV. Based on the obtained data, we attempt to understand and identify the capabilities and limitations of the multi-robot systems. In Section V, we compare and contrast the newly developed metrics with an alternate existing metric for the obtained data and offer concluding remarks in Section VI.

II. RELATED WORK

Evans and Messina in [5] analyzed the importance of defining universally accepted performance metrics for intelligent systems, outlined current efforts to develop standardized testing and evaluation strategies and argued the need for industry accepted metrics for inter-comparison of results and to avoid duplication of work. 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., [16] 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 [3]. 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:

$$\text{MTBF} = \frac{\text{No. of hours robot is in use}}{\text{No. of failures encountered}} \tag{1}$$

Though the study is very helpful in providing a detailed analysis of the component failure rate in mobile robots, 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 like MTBF. Unfortunately, most existing architectures are evaluated purely based on task-specific or architecture-specific quantities [14]. 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.

Another key drawback of the current techniques like MTBF is their inability to identify the extent of adaptive learning exhibited by a fault-tolerant system. Our earlier experience on developing multi-robot fault-diagnostic architectures [15], [12], convinces us that it is unlikely that the human designer can anticipate every possible fault that the multi-robot system may encounter *a priori*. Thus, a multi-robot team can achieve a much higher level of fault-tolerance if it is able to autonomously adapt over time. Hence, it is important for any multi-robot performance metric to evaluate the extent of intelligence exhibited by the system. In our work on metrics, we want to capture the notion of reasoning and intelligence as it influences the overall system performance.

The most promising work related to our objectives is the work of Hamilton, et al. [6]. Their approach outlines a metric for calculating "effective" fault-tolerance for single robot manipulators by combining the observed fault-tolerance with a performance/cost rating. The key drawback of the approach is that 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.

III. Metrics for evaluating fault-tolerance and system performance

A. 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 [2] four important factors are essential for the success of a diagnostic system:

- **Efficiency** — ability of the system to optimize available system 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 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 $\langle R_i, T_j \rangle$. 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., $c_j = u_j$.

B. Measuring System Performance

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

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

$$A_j = u_j \tag{2}$$

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

$$A = \sum_{j:T_j \in X} u_j \tag{3}$$

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. 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, given by

$$\rho_j^i = \frac{f^i_{known_j} + f^i_{unknown_j}}{F_j^i} \tag{4}$$

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

Based on equation 4, the task failure metric for the i^{th} attempt of task T_j is:

$$B_j^i = c_j * \left(1 - \rho_j^i\right) \tag{5}$$

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

$$B_j = \sum_{i=1}^{q_j} (c_j * (1 - \rho_j^i)) \tag{6}$$

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 6 across all task failures, gives:

$$B = \sum_{j:T_j \in Y} (c_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i) \tag{7}$$

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.,

$$P = A - B \tag{8}$$

Substituting for A and B from equations 3 and 7 respectively, we obtain our desired effective performance metric:

$$P = \sum_{j:T_j \in X} u_j - \sum_{j:T_j \in Y} (c_j * q_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i) \tag{9}$$

P provides the designer with a measure of the system's effective performance. The measure results in P values in the range $[-P_{Max}, 1]$, where P_{Max} is an arbitrarily large number that can approach infinity. 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.

C. 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.

Combining individual task robustness measures from equation 4 for failed task attempts with the robustness value for the successful attempts, system robustness can be represented as:

$$\rho_s = \frac{\sum_{j:T_j \in Y} \sum_{i=1}^{q_j} \rho_j^i + \sum_{q:T_q \in X} \rho_q^1}{|X + Y|} \tag{10}$$

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 efficiency metric (ϵ), or the total task-execution time spent by a robot on a task, T_j. In other words, the efficiency of a task can be used to qualitatively assess a system's ability to handle failures, i.e.:

$$t_j = \mathrm{t}_{Normal_j} + \mathrm{t}_{Fault_j} + \mathrm{t}_{Recovery_j} \tag{11}$$

$$\epsilon_j = \frac{\mathrm{t}_{Normal_j}}{t_j} \tag{12}$$

We emphasize that efficiency is representative of the system's ability to best utilize its resources towards completing the assigned task and is not a reflection of the quality of the implemented solution.

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

$$\epsilon = \frac{\sum_{j:T_j \in X} \frac{t_{Normal_j}}{t_j}}{X+Y} \tag{13}$$

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[1]. By comparing the efficiency of the system over a set of trials[2], we get an estimate of the learning exhibited by the system.

$$\delta_k = \epsilon_k^{'} - \epsilon_k^0 \tag{14}$$
$$\delta_u = \epsilon_u^{'} - \epsilon_u^0 \tag{15}$$

where $\epsilon_k^{'}$ is the efficiency metric for the *known* faults for the final trial, ϵ_k^0 is the efficiency value for *known* faults for the initial trial, $\epsilon_u^{'}$ is efficiency metric for the *unknown* faults for the final trial and ϵ_u^0 is the efficiency value for *unknown* faults for the 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. Currently, we are in the process of expanding the definition of δ to better account for the stochastic nature of the operating environment.

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. The reasoning tools are especially useful for fine-tuning an implemented fault-tolerance architecture, leading to the development of more refined and effective solutions.

D. Discussion

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. Future work intends to focus on exploring potential means to overcome this drawback.

[1]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.

[2]An experiment consists of a set of more than 1 trial.

IV. Evaluation of Metrics

In order to meaningfully validate the metrics and to show its application-independence, we apply them to the results obtained from two distinct physical robot experiments for different fault-tolerance architectures. The first architecture, the Causal Model Method (CMM) [7], uses a model-based approach that predefines a decision graph for detecting and diagnosing problems that occur during system operation, for fault-tolerance. The original CMM as described by Horling and Lesser [7] was defined as a directed, acyclic graph (DAG) used for organizing a set of diagnosis nodes. Each node in the graph corresponds to a particular diagnosis with the precision of reasoning increasing as the depth of the DAG increases. The nodes periodically perform simple comparison checks to determine if a fault may have occurred. Any deviations from the expected value of the characteristics 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.

The second architecture that we implemented, called LeaF [12], 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 existing CMM strategy with case-based learning algorithm to adapt and categorize a new fault and add it to the causal model for future use. The unique aspect of the proposed architecture is its ability to extract useful information from previously encountered faults. 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. Much more detail on LeaF can be found in [12] and [8].

A. Experiment I: Box Pushing Task

The first experiment we performed was the box pushing algorithm of [4], as implemented in work by Tang [13]. As in the original implementation, a team of two Pioneer robots equipped with a laser-scanner, a ring of sonars, and a forward mounted camera 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[3] to the goal (see Figure 1). 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 of box push, communication and blob tracking. Table I shows

TABLE I
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

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

the relation between the individual module and the set of sub-tasks.

Two separate sets of experiments were performed, with CMM as the fault-diagnosis method for one set and LeaF as the fault diagnosis strategy for the other set of experiments. To ensure consistency, data was collected from over 15 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 parameters of testing for both sets of experiments were kept the same to ensure the obtained results could be compared directly with one another. We define the the utility/cost values associated with the corresponding tasks as shown in Table II. For the box pushing task, since both the sub-tasks carry equal importance towards the completion of the overall task, the values are distributed equally. The calculated performance ratings are enumerated in Table III.

1) Discussion: Based purely on the performance values, we can claim that LeaF performs better than CMM. Looking closely at the value of robustness it can be said from the data that LeaF can handle twice as many faults as CMM. The ability of LeaF to handle unexpected faults is the reason for the higher robustness rating (Table II). Looking purely at the net efficiency value for the two systems is a little deceiving. The differences in the efficiency values between the two systems are better illustrated in Figure 2[4]. We can see clearly from the graph that LeaF better utilizes operating time and resources. In fact, LeaF displays learning through the course of the trials, resulting in improved values for efficiency across the system. As a caveat, despite the improvement over CMM, the low system efficiency rating for LeaF indicates the need for a more streamlined implementation of the diagnosis process. Additionally, for Trial 2, both systems are unable to handle the encountered failure. The ability of LeaF to learn from experience is captured in Figure 3. The graph traces the diagnosis time for one type of *known* 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. By reducing the diagnosis time, LeaF can utilize most of its execution time towards task completion.

[4]For trials that are stopped after 600 secs, 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.

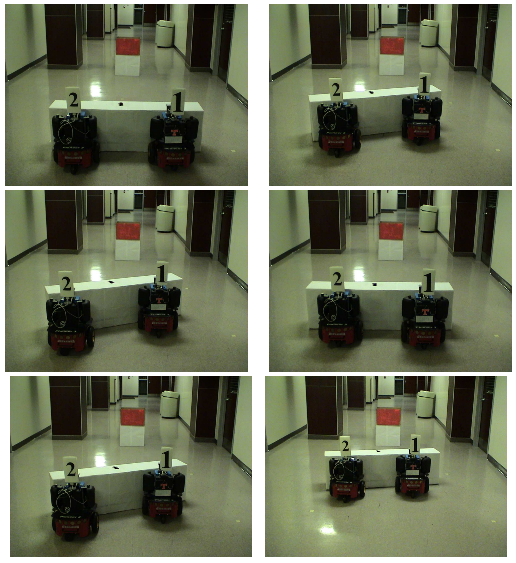


Fig. 1. Example of a box pushing task – Robots 1 and 2 alternately push a box towards the goal indicated by a blob at the far end of the corridor (read left to right, top to bottom).

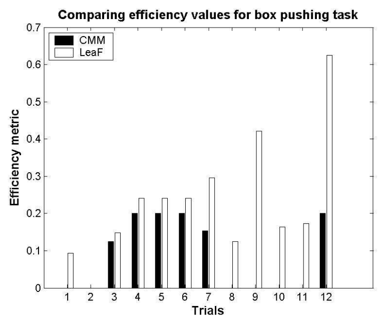


Fig. 2. Efficiency rating differences between LeaF and CMM for the box pushing task. The missing values in the graph (Trials 1, 2, 8, 9, 10, 11) correlate to an efficiency rating of 0, i.e., the system was unable to diagnose the fault and had to be manually rebooted.

B. Experiment II – Deployment task

The second physical robot experiment we performed was based on the idea of "assistive navigation" between robots. For this deployment task we use a small team (2-5) of mobile, physically heterogeneous cooperating robots for an indoor deployment task. The robot team has a very strict set of goals/tasks: autonomously deploy a sensor robot within an indoor environment. The composition of the team consisted of two classes of robots: lead robot equipped with scanning laser range-finders and cameras; and 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.

Because these sensor-limited robots could not navigate safely on their own, complex heterogeneous teaming behaviors were used that allowed the small group of helper robots to deploy the sensor-limited robots to their planned destinations [15]. Table IV shows the relation between the individual modules and the defined set of tasks. Figure 4 shows these robots performing one such deployment task. Similar to the box pushing task, two (2) sets of experiments were performed. The consequences of failure were

TABLE II
UTILITY-COST TABLE FOR THE EXPERIMENTS

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

TABLE III

PERFORMANCE EVALUATION TABLE FOR BOX PUSHING TASK

System	$\mathbf{P} \in (-\infty, 1]$	$\rho \in [0,1]$ **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

not corrected, and the team was allowed to learn from its own mistakes, whenever possible. Each run was considered a separate, discrete trial. In these experiments, a total of 15 single robot deployments were attempted. The distribution of faults for the two system were 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. We start by defining the utility/cost table values associated with the corresponding tasks. For deployment, since the most important criterion is for the robot to return home, the module Return Home is assigned the highest utility/task value. Similarly, based on prior experience with multi-robot systems, we determine the order of importance for the remaining modules and assign values (see Table V).

TABLE IV

TASK MODULE RELATIONSHIP TABLE FOR CMM AND LEAF

Task	Modules
Go to goal Task	Localization, Path planning, Navigation
Teleoperation Task	Marker Detection, Communication
Recharging Task	Localization, Path planning, Navigation, Marker Detection, Communication
Follow the leader Task	Blob Tracking
Return home Task	Localization, Path planning, Navigation

Table VI compares the system performance metric for CMM and LeaF. LeaF shows a marked improvement in the per trial measures of robustness, and efficiency. Despite LeaF's ability to recover from all the encountered faults over the course of the experiments, it is important to note that the value for robustness for LeaF is misleading. From our earlier work [12], we know that the larger the number of interacting components in an environment, higher is 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. Figures 5(a) and 5(b) plot the diagnosis curve for the two sets of faults from

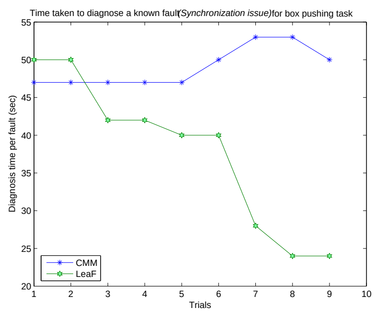


Fig. 3. Time of diagnosis for LeaF and CMM for the box pushing task.

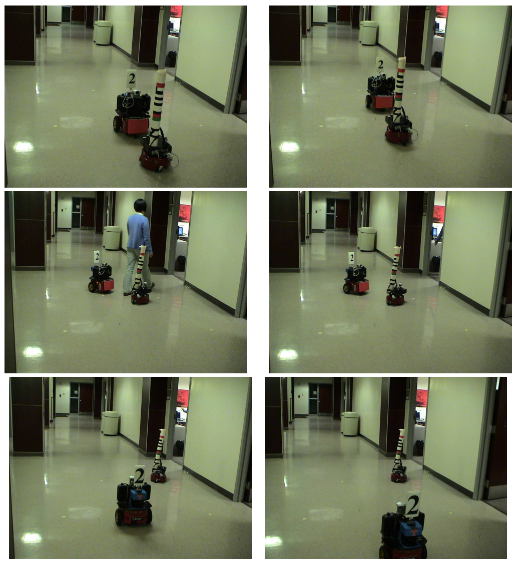


Fig. 4. Deployment of a sensor robot using assistive navigation: the lead robot reaches a pre-specified way-point, teleoperates the sensor robot into position and heads back to the starting position (read left to right, top to bottom).

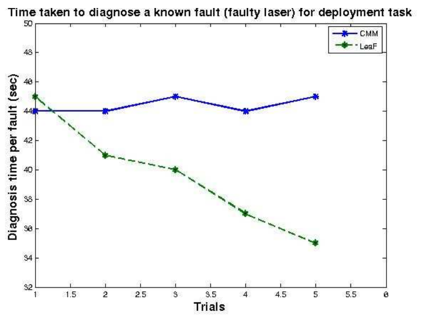
known and *unknown* category. For the *known* type of faults, the diagnosis times for CMM do not decrease over the course of trials and remain steady at best. However, in the case of LeaF, it is interesting to note that the system uses experience as a means to improve performance steadily over trials. For the case of *unknown* faults, from the graph in Figure 5(b) we can see that after a few instances of the unknown fault, LeaF is able to better diagnose the problem[5]. The use of case-based reasoner in LeaF ensures that the system is constantly learning about the new fault. 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. Based on the analyzed data and our prior work with diagnostic systems [12], 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. It is also 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. On the other hand, increasing the number of components, combined with a lack of learning, results in a sharp downward performance curve. In fact, we can make the claim that for such systems, the architecture should display significant learning just to

[5]The actual times only represent the quality of implementation and not the quality of learning.

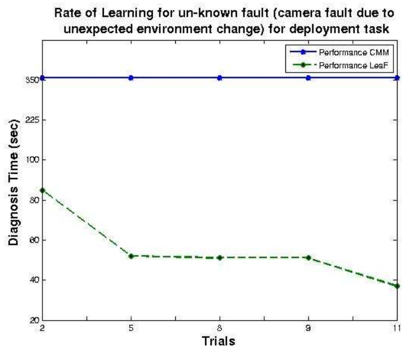
TABLE VI

EVALUATION TABLE FOR CMM AND LEAF FOR DEPLOYMENT 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	0.1938	0.6133	0.1814	0.0	0.0
LeaF	0.855	1.0	0.5221	0.1	1.39



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



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

Fig. 5. Performance illustrated over the period of operation for CMM and LeaF.

TABLE V

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
Teleoperation 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

maintain the initial performance rating. It is to be noted that the lack of learning is not always an indicative of 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. Additionally, the CMM 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. Thus the CMM performs well for static situations, but starts degrading when any dynamic changes are made to the operating environment, whereas the LeaF is a more competent architecture for static as well as dynamic systems.

V. 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. In this section, we apply the Hamilton-Walker-Bennett (HWB) [6] metric to the obtained results from the two different physical robot experiments and subsequently analyze the effectiveness for each one in helping a designer better understand an implemented system. The metric is defined as follows:

$$HWB_{eff} = k_1(f)^2 + k_2(p)^2; \tag{16}$$

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 ineffective system and 1 represents a system with an ideal balance of fault-tolerance and performance. Translating to the multi-robot domain,

$$HWB_{eff} = \begin{array}{c} \displaystyle\sum_{j=1}^{m} u_j(\rho_{redundancy_j})^2 + \\ c_j\left(1 - \dfrac{\frac{t_{diagnosis_j} + t_{Recovery_j}}{t_j} + 1}{2}\right)^2 \end{array} \tag{17}$$

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 sub-systems or sub-tasks,
- Performance speed is inversely mapped to the system efficiency from our metrics model, and
- The cost value is always set to 1, as we never take into consideration the actual physical cost towards overall

system performance.

TABLE VII
CALCULATED VALUE FOR EFFECTIVE FAULT-TOLERANCE FOR BOTH METRICS FOR THE DIFFERENT TEST SYSTEMS

Experiment	**System**	**HWB**	**Kannan-Parker**[6]
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

Towards a fair comparison, we scale our metric values to the same range $[0, 1]$ as the *HWB* measure. We illustrate the calculated performance values for both systems in Table VII. Comparing the two metric values, it immediately becomes clear that 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. The predominant reason for this is that 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 [2] 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. To summarize, the results from our experiments indicate that our metric gives more informative indicators of the system performance, but that more research is needed to determine to make a more general comparison between the two metrics.

VI. CONCLUSIONS AND FUTURE WORK

As new techniques in fault-tolerance are being explored [12], existing methods do not provide a complete measure of system performance for multi-robot teams. In this paper, we present an evaluation metric to measure the extent of fault-tolerance towards system improvement over a period of time. Furthermore, we evaluate two different 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. To the best of our knowledge, this is the first metric that attempts to evaluate the quality of learning towards understanding system level fault-tolerance. We intend to focus future efforts towards overcoming the purely empirical nature of the developed metrics.

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

REFERENCES

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

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

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

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

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

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

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

[8] B. Kannan. *LeaF: A Learning-based Fault Diagnostic System for Multi-Robot Teams*. Ph.D. Dissertation, University of Tennessee, Department of Computer Science, 2007.

[9] B. Kannan and L. Parker. Fault-tolerance based metrics for evaluating system performance in multi-robot teams. In *Proceedings of Performance Metrics for Intelligent Systems Workshop*, 2006.

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

[11] R. Murphy and D. Hershberger. 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, 1996.

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

[13] L. Parker and F. Tang. Building multi-robot coalitions through automated task solution synthesis. *Proceedings of the IEEE, special issue on Multi-Robot Systems*, 94(7):1289–1305, 2006.

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

[15] L. E. Parker, B. Kannan, F. Tang, and M. Bailey. 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, 2004.

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