

Fault-Tolerance Based Metrics for Evaluating System Performance in Multi-Robot Teams

Balajee Kannan and Lynne E. Parker

Distributed Intelligence Laboratory, Department of Computer Science

The University of Tennessee, Knoxville, TN 37996-3450

Email: {balajee, parker}@cs.utk.edu

Abstract—The failure-prone complex operating environment of a standard multi-robot application dictates some amount of fault-tolerance to be incorporated into the system. Being able to identify the extent of fault-tolerance in a system would be a useful analysis tool for the designer. Unfortunately, it is difficult to quantify system fault-tolerance on its own. A more tangible metric for evaluation is the “effectiveness” [8] measure of fault-tolerance. *Effectiveness* is measured by identifying the influence of fault-tolerance towards overall system performance. In this paper, we explore the significance of the relationship between fault-tolerance and system performance, and develop metrics to measure fault-tolerance within the context of system performance. 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 measures of redundancy. The developed metrics are designed to be application independent and can be used to evaluate and/or compare different fault-diagnostic architectures. We show the utility of the designed metrics by applying them to a sample complex heterogeneous multi-robot team application and evaluating the effective fault-tolerance exhibited by the system.

I. INTRODUCTION

To scale the use of robots from simple tasks in static environments to large-scale applications, they must be able to effectively and robustly coordinate many different functionalities. Multi-robot teams applied to complex applications will typically require robot team members to perform multiple tasks such as planning, mapping, localization, formation-keeping, information sharing, and so forth. These functionalities are especially useful for applications involving dynamic environments such as urban search and rescue and future combat systems.

However, the nature of these operating environments are such that 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. Hence, it is important for these systems to exhibit some fault-tolerance, or the ability to diagnose and recover from encountered faults.

In the last decade, several researchers have studied fault-tolerance for robotic systems (e.g., [15], [1], [16], [5], [12], [9], [17], [14]). 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.

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. Based on this analysis of *effective* fault tolerance, this paper addresses the following problem: *Define a metric for calculating the usefulness of fault-tolerance towards system performance in multi-robot teams.*

The rest of this paper is organized as follows. We first present a brief review of the related work and discuss how the existing methods for evaluation are insufficient for multi-robot systems in Section II. Section III formally defines the above problem and details the derivation of the proposed metrics. In order to evaluate the validity of the metrics, we apply them to a set of experimental results obtained from a physical robot implementation of a sample complex heterogeneous application [10] in Section IV. We discuss the potential scope and significance of the new metrics in Section V and offer concluding remarks and comments on our ongoing work in Section VI.

II. RELATED WORK

The concept of metrics for quantifying performance is not new. In 1994, Cavallaro and Walker [4] recognized the lack of standard metrics and discussed the applicability of protocols based on NASA and military standards. Evans and Messina in [6] 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 [22] 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. Based on this concept, 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:

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

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:

$$MTTR = \frac{\text{No. of hours spent repairing}}{\text{No. of repairs}} \quad (2)$$

$$Availability = \frac{MTBF}{MTTR + MTBF} \cdot 100\% \quad (3)$$

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 [21], [20], 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. [13], 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 [23] approach for measuring *autonomy* for intelligent systems and Finkelstein’s Analytical Hierarchy Process (AHP [7]) 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 [19]. 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. [8]. 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 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:

$$eff = k_1(f)^2 + k_2(p)^2 \quad (4)$$

where *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 prior work on developing and evaluating fault-diagnostic architectures for multi-robot systems [20], [21] 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 mentioned in the previous section, 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. In this paper, we attempt to 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. We introduce our measures in the next section.

III. PROBLEM DEFINITION

Based on our earlier work on developing turn-key solutions¹ for fault-diagnosis [20], we evaluate system performance based on the following terms:

- 1) **Efficiency** — ability of the system to best utilize the available resources,
- 2) **Robustness** to noise — ability of the system to identify and recover from faults, and
- 3) **Learning** — ability to adapt to the changes in the environment by autonomously extracting and integrating

¹A *turn-key* solution, as defined by Carlson and Murphy [2], is one that can be implemented on different applications without the need for significant modifications.

useful system information during the course of task execution.

We now formally define the problem as follows. Given:

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

We assume:

- 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*, which are faults the designer can anticipate based on experience, application type and operating environment; *unknown*, which are faults not anticipated by the designer, but which can be diagnosed by the system based on experience and sparse information available during execution; and *undiagnosable*, which are faults that cannot be classified autonomously and need human intervention. In addition to diagnosis, the system can autonomously recover from *known* and *unknown* faults, whereas human assistance is required for it to recover from *undiagnosable* faults. The number of faults in each category are represented as f_{known}^i , $f_{unknown}^i$, and $f_{undiagnosable}^i$.
- 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: task success or task 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.
- Based on the importance of the task the designer builds a task-utility table, such as that shown in Table I, in which the summation of the terms ($\sum u$ and $\sum c$) are normalized between ranges of $[0, 1]$.

A. 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 . An award is associated with every successfully completed task, given by the utility component u_j .

$$A_j = u_j \quad (5)$$

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

$$A = \sum_{j:T_j \in X} u_j \quad (6)$$

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. The punishment associated with a failed task attempt is given by the cost component for task failure, c_j . 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. We define ρ_j^i as the measure of robustness for the i^{th} attempt of task T_j and is given by

$$\rho_j^i = \frac{f_{known_j}^i + f_{unknown_j}^i}{F_j^i} \quad (7)$$

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

Based on equation 7, the task failure metric for the i^{th} attempt of task T_j is:

$$B_j^i = c_j * (1 - \rho_j^i) \quad (8)$$

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

$$B_j = (c_j * q_j) * \sum_{i=1}^{q_j} (1 - \rho_j^i) \quad (9)$$

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.

TABLE I
UTILITY-COST TABLE FOR SYSTEM TASKS

Task	Utility	Cost for task failure
T_1	u_1	c_1
T_2	u_2	c_2
...
T_m	u_m	c_m

Simplifying,

$$B_j = (c_j * q_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i) \quad (10)$$

Extending equation 10 across all task failures, gives:

$$B = \sum_{j:T_j \in Y} (c_j * q_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i) \quad (11)$$

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.,

$$P = A - B \quad (12)$$

Substituting for A and B from equations 6 and 11 respectively, we obtain our desired effective performance metric:

$$P = \sum_{j:T_j \in X} u_j - \sum_{j:T_j \in Y} (c_j * q_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i) \quad (13)$$

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. However, P by itself does not provide the all the information necessary for validation. Hence, we need to identify additional metrics that help give a complete picture of the system.

B. Measuring Fault-tolerance

In addition to outlining a measure for performance, we are interested in identifying the fault-tolerance in the system. Based on Murphy and Carlson's observation from the previous section, 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, system robustness can be represented as:

$$\rho_s = \sum_{j:T_j \in Y} \sum_{i=1}^{q_j} \rho_j^i \quad (14)$$

A high value of ρ_s (an average system exhibits a ρ_s value close to 1.5) indicates a highly robust system and a ρ_s value of 0 indicates a system with no robustness to faults.

In order to define the system efficiency metric (ϵ), we need to measure the total time (t_j) 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.,

$$t_j = t_{Normal_j} + t_{Fault_j} + t_{Recovery_j} \quad (15)$$

Then, we can define ϵ as:

$$\epsilon = \sum_{j:T_j \in X} \frac{t_{Normal_j}}{t_j} \quad (16)$$

Similar to the robustness measure, a more efficient system has a higher value of ϵ and an inefficient system has ϵ near 0. The influence of learning towards system performance can be measured as an empirical quantity. Comparing system performances with and without learning gives us a good estimate of the learning in the system.

$$\delta = P - P' \quad (17)$$

where P is a system with learning and P' is a system with no learning.

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) .

IV. EVALUATION OF METRICS

To give a better understanding of the range of values for the metrics, we apply them to the following simple example scenarios.

A. Scenario 1

Consider a sample multi-robot application comprised of 10 individual tasks to be completed by a team of 10 functionally similar robots. We make the assumptions that the robots encounter one failure per task and the task/utility weights are evenly distributed. Then we define these measures as follows:

$$\forall i. u_i = c_i \quad (18)$$

$$\sum_{i=1}^{10} u_i = \sum_{i=1}^{10} c_i = 1 \quad (19)$$

The time spent by the robot in Normal operation mode is assumed to be t secs. Also, as it takes a very small fraction of time to diagnose task failure from the time a fault is discovered, we assume this time to be negligible and ignore it.

TABLE II
EVALUATION TABLE FOR SCENARIO 1

System	Case	P	ρ	ϵ	δ
S_1	Best case	1	0	10	0
	Average case	0.5	0	5	0
	Worst case	-1	0	0	0
S_2	Best case	1	0	10	0
	Average case	0.4	0	7.5	0
	Worst case	-0.4	0	5	0

To best illustrate the variations in the values, we choose three specific cases to evaluate, namely:

- 1) Best-case, where the system encounters no failures,

- 2) Average-case, where the system encounters at least one failure in half the number of executed tasks, and
- 3) Worst-case, where there is at least one failure in all cases.

Table II illustrates the values obtained for two different hypothetical architectural implementations – one with no built-in fault-tolerance (S_1) and another with some redundancy-based fault-tolerance (S_2). For this scenario, the task/utility weights are evenly distributed, i.e., $u_1 = c_1, u_2 = c_2, \dots$. When a fault is encountered during task execution in the first architecture, robot(s) do not have the capability to recover and report a failed task. In the case of the second architecture, if and when a failure occurs, the task is assumed to have failed and is reassigned to another team member for execution. The task reassignment continues until all robots in the team have had an opportunity to complete the task. We further make the assumption that on average it takes $\frac{n}{2}$ attempts to successfully recover from an encountered fault. Finally, we assume there is 50% probability of the system successfully recovering from an encountered error.

Looking at the values for system performance in Table II we can infer that, for the average-case, the architecture with zero fault-tolerance (S_1) performs better than the architecture with some fault-tolerance (S_2). For this specific application, as the cost/utility values are equivalent, the inability of the system to handle failures does not significantly affect system performance (P) for the average-case. In fact, the time and resources spent in fault-diagnosis and recovery by S_2 adversely affects its performance. However, with increasing number of faults the ability of the system to handle failures becomes important. This is indicated by the worst-case performance of the two architectures. Table II shows S_2 edging S_1 as the number of failures in the system increases. On the other hand, a system with a higher task completion rate will have a higher value for efficiency (ϵ), as reflected in Table II. Finally, a δ value of zero highlights the fact that neither system exhibits any kind of learning.

B. Scenario 2

Consider an alternate multi-robot application comprised of 10 individual tasks to be completed by a team of 10 functionally similar robots. Similar to the above scenario, we make the assumptions that the robots encounter one failure per task. In contrast to Scenario 1, the task/utility weights are not evenly distributed with a higher utility associated for task success than the cost for a task-failure, as given by:

$$\forall i, j. u_i = u_j \quad (20)$$

$$\forall i, j. c_i = c_j \quad (21)$$

$$\sum_{i=1}^{10} u_i > \sum_{i=1}^{10} c_i \quad (22)$$

$$\sum_{i=1}^{10} u_i = 1 \quad (23)$$

$$\sum_{i=1}^{10} c_i = 0.5 \quad (24)$$

The time spent by the robot in Normal operation mode is assumed to be t secs. Also, as it takes a very small fraction of time to diagnose task failure from the time a fault is discovered, we assume this time to be negligible and ignore it.

To maintain consistency, we choose the same three specific cases to evaluate that we discussed above, namely:

- 1) Best-case, where the system encounters no failures,
- 2) Average-case, where the system encounters at least one failure in half the number of executed tasks, and
- 3) Worst-case, where there is at least one failure in all cases.

TABLE III
EVALUATION TABLE FOR SCENARIO 2

System	Case	P	ρ	ϵ	δ
S_1	Best case	1	0	10	0
	Average case	0.25	0	5	0
	Worst case	-0.5	0	0	0
S_2	Best case	1	0	10	0
	Average case	0.575	0	7.5	0
	Worst case	0.15	0	5	0

The difference in the performance of the two systems S_1 and S_2 is highlighted Table III. Unlike in the previous scenario, system S_2 has a consistently higher performance rating than system S_1 . As more emphasis is placed on task success than on system failure (i.e., higher utility value), the ability to recover from failures and to complete the assigned task directly impacts system performance (P). Similar to the previous scenario, system S_2 has higher efficiency (ρ) values than system S_1 . Hence, comparing the values of P , ρ , and δ for the two systems, we can say S_2 is a more suitable architecture for the application.

C. Scenario 3

Finally, we apply the metrics to the experimental results obtained for the physical robot implementation of a complex heterogeneous application [10]. This 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: to autonomously explore and map a single story in a large indoor environment, detect a valued object, deploy a sensor network and use the network to track intruders within the building.

The composition of the team shown in Figure 1 consisted of three classes of robots: Four (4) mapping robots equipped

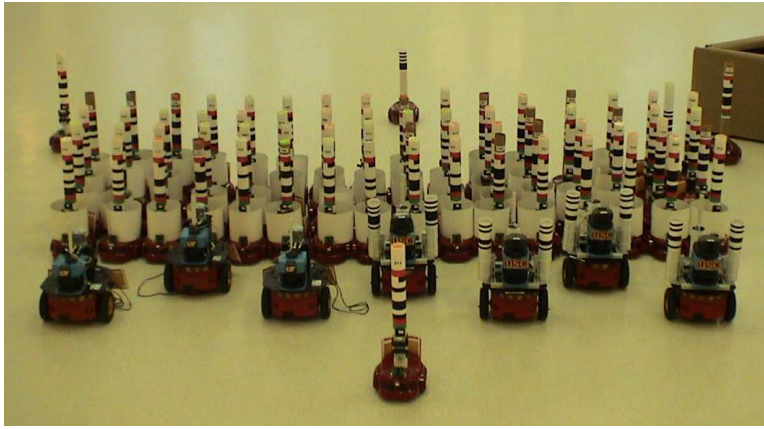


Fig. 1. The heterogeneous robot team — mapper, helper and simple robots.

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 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.

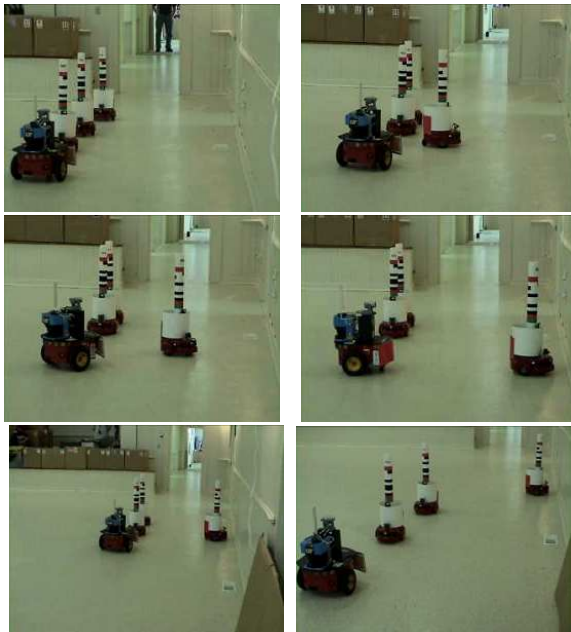


Fig. 2. Deployment of a sensor robot using assistive navigation: the lead robot first guides and then directs the sensor robot into position (read left to right, top to bottom).

In order to perform the task of locate-and-protect, the sensor-limited mobile robots had to be moved into deployment positions that were optimal for serving as a sensor network. 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 (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 [21]. Table IV shows the relation between the individual modules and the defined set of tasks. Figure 2 shows these robots performing one such deployment task. The 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 in a heavily cluttered environment, leading to a wide variety of faults.

To handle the encountered faults, a fault diagnostic system based on causal model methodology was implemented (see [20] for more details). The experiments consisted of repeated deployments of 1, 2, or 3 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. Thus, the data collected incorporates propagation of error from one experiment to the next. In these experiments, a

TABLE IV
TASK MODULE RELATIONSHIP TABLE FOR SCENARIO 3

Task	Modules
Go_to_goal Task	Localization, Path_planning, Navigation
Deployment Task	Marker Detection, Communication
Recharging Task	Localization, Path_planning, Navigation, Marker Detection, Communication
Follow_the_leader Task	Marker Detection
Return_home Task	Localization, Path_planning, Navigation

TABLE V
OVERALL SYSTEM SUCCESS RATE, AVERAGED OVER 45 TRIALS

Module	Subsystem Success Rate	Experimental Success Rate
Localization	0.83	
Path Planning	0.99 (est.)	
Navigation	0.95 (est.)	
Follow_the_leader	0.78 (est.)	
Marker Detection	0.98	
Communication	0.91	
Complete System	0.54 (est.)	0.67 (2-robot depl.) 0.48 (1-robot depl.) 0.59 (combined over all trials)
Helper Robot returning home		0.91 (over all trials)

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. Table V 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.

TABLE VI
UTILITY-COST TABLE FOR SCENARIO 3

Task	Utility	Cost for task failure
Go_to_goal Task	0.15	0.08
Deployment Task	0.15	0.08
Recharging Task	0.15	0.08
Follow_the_leader Task	0.15	0.08
Return_home Task	0.3	0.18

TABLE VII
EVALUATION TABLE FOR SCENARIO 3

System	P	ρ	δ
SDR	-5.4283	3.976	0

In order to better understand the quality of performance of the described system, we apply our metric 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 VI.

For these experiments, ρ is a measure based on the total probability of task success. Task success is given by the product of the success probabilities of individual sub-modules

for the specified task. Also, as the faults propagate from one run to the next, the entire set of trials (61) is considered as a single continuous experiment. From the collected experimental values in Tables VI and V, we get

$$A = ((.83 * .99 * .95) * .15) + ((.98 * .91) * .15) + \dots$$

$$B = ((.08 * 61) * (1 - (.83 * .99 * .95))) + ((.08 * 61) * (1 - (.98 * .91))) + \dots$$

Table VII shows the evaluated values for system performance and fault-tolerance. The system displays a high amount of robustness. 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 Table also indicates a total lack of any learning in the fault-tolerance design², which is consistent with the analysis that was performed separately [20]. On the other hand, 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. Instead, an adaptive method is needed to enable the robot team to use its experience to update and extend its causal model to enable the team, over time, to better recover from faults when they occur.

In our other ongoing work [20], we are developing an extended causal model methodology, called LeaF, that enables the system to learn from experience and adapt its model, thereby improving its ability to diagnose and recover from unexpected faults. The unique aspect of the proposed architecture is its ability to extract useful information from previously encountered faults. Specifically, LeaF is designed as a distributed model that uses one or more partial causal models for representing the various faults in the system. 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. In addition, 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. 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. Ultimately the goal is to build a cross-architecture system capable of learning from its own faults and those of other team members, making it a domain- and application-independent architecture.

²Since the experimental results did not have information regarding the time spent in handling faults, we do not calculate the efficiency metric for this system.

V. 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 one of the two measures provide a complete assessment of the application in use. Using only 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 systems. However, the two metrics when viewed in context with each other helps the designer compare and contrast performances of different architectures in order to select the most appropriate one for the application in question. The ability to compare systems can help identify potential shortcomings, leading to the development of more refined and effective solutions. This also reduces the amount of time and resources spent in duplicating existing work.

VI. CONCLUSIONS AND FUTURE WORK

As new techniques in fault-tolerance are being explored [20], existing methods do not provide a complete measure of system performance for multi-robot teams. Hence, there is a need for a more generic evaluation method for multi-robot systems. 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 evaluated a large-scale multi-robot application based on the defined metrics. Specifically, the research provides a qualitative measure for identifying system fault-tolerance in terms of efficiency, robustness and the extent of learning. 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.

As part of our ongoing research, we plan to apply the metrics to our newly proposed fault-tolerance architecture, LeaF (Learning-based Fault diagnosis), and compare the results with those of other existing architectures, such as CMM [11], SFX-EH [18], and so forth. The observations will help us further evaluate refine our approach.

ACKNOWLEDGMENTS

Parts of this research were sponsored by DARPA/IPTO's Software for Distributed Robotics program, Science Applications International Corporation, and the University of Tennessee's Center for Information Technology Research. This paper does not reflect the position or policy of the U.S. Government and no official endorsement should be inferred.

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. R. Murphy. Reliability analysis of mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.
- [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] J. Cavallaro and I. Walker. 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, March 1994.
- [5] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robust positioning with a multi-agent robotic system. In *Proceedings of IJCAI-93 Workshop on Dynamically Interacting Robots*, pages 118–123, 1993.
- [6] J. Evans and E. Messina. Performance metrics for intelligent systems. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II, August 2000.
- [7] R. Finkelstein. A method for evaluating IQ of intelligent systems. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II, August 2000.
- [8] 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.
- [9] 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.
- [10] A. Howard, L. E. Parker, and G. S. Sukhatme. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment, and detection. *International Journal of Robotics Research*, 2006.
- [11] E. Hudlická and V. R. Lesser. Modeling and diagnosing problem-solving system behavior. *IEEE Transactions on Systems, Man, and Cybernetics*, 17:407–419, 1987.
- [12] G. A. Kaminka and M. Tambe. What is wrong with us? Improving robustness through social diagnosis. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 97–104, 1998.
- [13] S. Lee, W. Bang, and Z. Bien. Measure of system intelligence: An engineering perspective. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II, August 2000.
- [14] M. Long, R. 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.
- [15] S. Mahdavi and P. Bentley. An evolutionary approach to damage recovery of robot motion with muscles. In *European Conference on Artificial Life (ECAL)*, pages 248–255, 2003.
- [16] M. J. Mataric. 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, 1993.
- [17] 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.
- [18] R. R. Murphy and D. Hershberger. Handling sensing failures in autonomous mobile robots. *The International Journal of Robotics Research*, 18:382–400, 1999.
- [19] 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.
- [20] L. E. 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, to appear.
- [21] 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.
- [22] L. Pouchard. Metrics for intelligence: a perspective from software agents. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II, August 2000.
- [23] A. Yavnai. Metrics for system autonomy. Part I: Metrics definition. In *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, volume Part II, August 2000.