Chapter Six

Reliability and Fault Tolerance in Collective Robot Systems

Lynne E. Parker

6.1 Introduction

Collective robotic systems (or, equivalently, *multi-robot teams*) have many potential advantages over single-robot systems, including increased speed of task completion through parallelism; improved solutions for tasks that are inherently distributed in space, time, or functionality; cheaper solutions for complex applications that can be addressed with multiple specialized robots, rather than all-capable monolithic entities; and, increased robustness and reliability through redundancy [Parker (2008)].

Of course these advantages do not come for free. Indeed, collective robot systems often experience a variety of problems that do not arise in single-robot solutions. First, even though the individual robot cost and complexity may be less in a collective solution, determining how to manage the complete system may be more difficult and complex because of the lack of centralized control or of a centralized repository of global information [Goldman and Zilberstein (2004)]. Further, collective robotic systems may require increased communication to coordinate all the robots in the system [Xuan *et al.* (2001)]. Increasing the number of robots can lead to higher levels of interference [Goldberg and Mataric (1997)], as the robots must act without complete knowledge of their teammates' intentions. Additionally, collective systems will often experience increased uncertainty about the state of the system as a whole [Mataric (1995)].

If not properly handled, all of these challenging issues can lead to a collective system that is unreliable and faulty [Parker (1998)]. Fortunately, a number of techniques have been developed to realize the advantages of collective robotic systems while countering many of the possible disadvantages. This chapter discusses the challenges of achieving collective robotic systems that are reliable and fault tolerant, as well as methods for detecting anomalies in these systems, so that recovery procedures can be initiated. In Section 6.2, we introduce the problem of reliability and fault tolerance in collective robot systems, followed by a discussion in Section 6.3 regarding the causes of unreliability and faulty systems. Section 6.4 discusses both qualitative and quantitative metrics for measuring the reliability and fault tolerance of collective systems. General mechanisms for fault detection, diagnosis, and recovery in collective systems are studied in Section 6.5. Section 6.6 presents four case studies of research approaches that address important aspects of achieving collective robotic systems that are reliable and fault tolerant. The chapter concludes in Section 6.7 with a discussion of several open challenges that remain before truly reliable and fault tolerant robot collectives can be achieved.

Throughout this chapter, a *robot collective* can refer to any type of multi-robot system, from a swarm of simple homogeneous robots that exhibit emergent cooperation to a small number of heterogeneous robots that explicitly reason about their cooperation. While the mechanisms for achieving reliability and fault tolerance may differ across the various types of robot collectives, there are also many overlapping concepts. The objective of this chapter is to provide an overview of the possible approaches that may be appropriate for achieving reliability and fault tolerance in a variety of multi-robot systems.

6.2 Background

Even the most carefully designed and tested robots may behave abnormally in some situations; therefore, it is necessary for robots to monitor their performance so that deviations from expected behaviors can be promptly detected and handled.

Designing collective robot systems to be **reliable and robust** requires addressing of a variety of inter-related questions, including:

- How (or whether) to detect when robots have failed;
- How (or whether) to diagnose and identify robot failures; and,
- How (or whether) to respond to (i.e., recover from) robot failures.

In some cases, robot collectives may be designed to be inherently robust, meaning that no explicit reasoning needs to take place about faults. Instead, the cooperative control is designed to work in spite of certain failures, which may not have to be explicitly identified or diagnosed. Nevertheless, even in such systems, the designers typically have to analyze and address the many types of faults that may occur in the robot collective, so as to design robust control solutions.

Many terms are used in this discussion of unreliability and faulty systems in robot collectives. Some of the common terms are defined as follows:

- *Fault*: a deviation from the expected behavior of the robot system. In some applications, the determination of a fault is a binary designation in which a fault has either occurred or it has not. In other applications, however, it may be more meaningful to measure degrees of faults, along a continuum from a completely nominal system to a completely failed one.

- *Reliability*: the probability that a device, system, or process will perform its prescribed duty without failure for a specified period of time when operated correctly in a specified environment. Different measures of reliability can be given to individual robot components, to individual robots, or to the entire collective of robots. Of particular interest is measuring the reliability of the robot team as a whole, regardless of the reliability of the individual components or robot team members.
- *Fault Tolerance*, or *Robustness*: the capability of a system to continue operation, perhaps at a reduced level (i.e., graceful degradation), even when undesired changes in the internal structure or external environment occur. In the context of robot collectives, fault tolerance and robustness typically refer to the ability of the system to deal with failed robots.

6.3 Causes of Unreliability and Faulty Systems

One of the key motivations building for collective robot systems is to achieve increased overall reliability through the redundancy of multiple robots. The idea is that several individual robot failures could be overcome simply through redundancy, under the assumption that even if some number of robots fail, sufficient numbers of properly functioning robots will be available to successfully accomplish the application.

To realize this objective, the system must be designed with these faults in mind. There are many reasons why collective robotic systems can be faulty and unreliable. Some of these reasons are *internal* causes, which are issues within the individual robots or their software design, while others are *external* causes, which are due to environmental effects. Some of the more common causes of faults in robot collectives include:

- *Individual robot malfunctions*. As with most engineered systems, the more components there are in the system, the more opportunities there are for failure. Carlson and Murphy [Carlson and Murphy (205)] report on a wide variety of reasons why individual robots fail in the field. Their study focuses primarily on teleoperated robots that are remotely controlled by human operators in urban search and rescue, or military applications. The findings show that the reliability of these robots is low, with a mean time between failures (MTBF) between 6 and 20 hours. Many different causes of failure were discovered, which can be roughly categorized as physical failures, software control failures, or human control errors. The most common causes of failure reported were robot effector failures, unstable control systems, platforms designed only for a narrow range of operating conditions, limited wireless communication range, and insufficient bandwidth for video feedback. More broadly, their research found that the most unreliable components of the system were those that were custom-designed or hand-built (e.g., control and effector systems), while the most reliable components were simple and/or mass-produced (e.g., power and sensors). While this prior study is for a particular sub-domain of robotics, the findings are not unique to this sub-domain. Thus, any collective of robots will also have to deal with many types of individual robot failures.

- *Local perspectives that are globally incoherent*. By definition, distributed systems are composed of individual entities that maintain only a local perspective. While some non-local information may be shared between entities in the system, no individual entity will have complete knowledge across the entire domain of the collective system. Actions taken based only on local information could lead to globally incoherent solutions. There are several reasons for the global incoherence in this context. For example, the interaction of the local control approaches of individual entities may cause unexpected emergent consequences that are undesirable. It could be that the local control methods are incorrect, leading to global incoherence. Further, even if the local control approaches are proper for certain situations, it may be that unexpected events or incomplete design lead to failures in the coordination mechanisms. In all these cases, the result is a collective system that does not behave in a globally coherent fashion.
- *Interference*. Any system with multiple robots sharing the same physical space must deal with contention, or interference, that may arise when robots operate near each other. Without properly addressing physical workspace contention, the benefits of multiple robots can be erased by harmful inter-robot interference.
- *Software errors or incompleteness*. Complex software systems are notoriously difficult to validate. Large collective robot systems consist of multiple robots, each of which may be rather complex. Ensuring that the autonomous control software for these systems is accurate for all possible situations and environments to which the collective may be applied is currently beyond the state of the art. Thus, providing the robot collective with an ability to deal with unmodeled events is important for achieving robust solutions.
- *Communications failures*. Many collective robot systems can achieve increased performance by enabling robots to share partial information about their local state or environment. Work by Arkin, et al., [Arkin *et al.* (1993)] showed that even a small amount of shared state can significantly improve team performance. However, collective systems must also be designed to properly respond even when this communicated information is not available.

Thus, it should be clear that collective robot systems are quite likely to experience a variety of failures during operation. Even though the robot team may not need to explicitly identify and/or diagnose each problem that arises, the team as a whole must have some means for compensating for the failures that do occur. This places the burden on the system designer to ensure that the system is properly designed to overcome failures when they occur, so that the team can continue to successfully meet its application objectives.

6.4 Measuring Reliability and Fault Tolerance

Evaluating the success of a robot collective in achieving reliability and fault tolerance can be performed either qualitatively or quantitatively. Qualitative analyses are helpful for understanding the types of problems that might arise in a multi-robot system, whereas quantitative analyses can offer the possibility of comparing and contrasting alternative approaches according to the same performance metrics. The following subsections discuss these techniques in more detail.

6.4.1 *Qualitative Analysis*

As noted by Winfeld and Nembrini [Winfield and Nembrini (2006)], robot collectives consisting of large numbers of simple robots (i.e., robot *swarms*) are commonly presumed to be robust by design for a number of reasons, including:

- Since robot swarms are distributed, they have no single point of failure;
- Robot swarms consist of simple robots that could be designed to be functionally and mechanically reliable;
- Because of the large numbers of robots, the swarm could be tolerant of environmental noise and uncertainty; and,
- Because of redundancy, the swarm may be tolerant to the failure of individual robots.

However, relatively little work has actually performed the analysis of robot collectives to verify that these benefits are indeed obtained from any particular robot team. While some algorithms for robot collectives have been shown to continue proper operation even when a robot fails (e.g., Lewis, et al., [Lewis and Tan (1997)] show that their formation control behavior is robust to robot failure), such an analysis is not always performed.

Winfield and Nembrini [Winfield and Nembrini (2006)] address this issue by proposing the use of a qualitative Failure Mode and Effect Analysis (FMEA) approach [Dailey (2004)] to evaluate the fault tolerance of a robot swarm. They apply their analysis to a swarm robot system performing a containment task, which requires the robot collective to physically surround, or contain, an object or area of interest, as illustrated in Fig. 6.1. In this application, only local wireless connectivity information is used to achieve swarm aggregation. Under this concept, robots within the swarm are wirelessly ”glued” together based on local connectivity. In this approach, connectivity information is transmitted between robots for only a single hop. Each robot broadcasts its own ID and the IDs of its imediate neighbors. If a robot loses a connect to a particular robot, and the number of remaining connected neighbors is less than a threshold, then the current robot assumes it is moving out of the swarm and turns arund. However, when the number of connections increases, the robot chooses a new direction randomly. The robot swarm is deemed *coherent* if any break in connectivity lasts less than a given constant time. From coherence arises two emergent behaviors: swarm aggregation and a connected ad hoc wireless network. The behaviors are completed with the use of short-range avoidance sensors and an emergent beacon taxis behavior. Altogether, this system design includes five emergent behaviors:

- Swarm aggregation
- Coherent ad hoc network
- Beacon taxis
- Obstacle avoidance
- Beacon encapsulation

For this particular wireless connectivity application, Winfield and Nembrini apply the FMEA approach to determine the robustness of the system design for the robot swarm. Under the FMEA approach, the designer attempts to identify all of the internal hazards (e.g., faults in robots or robot subsystems, such as motor failure, communications failure, sensor failures, and control system failures) and external hazards (e.g., environmental disturbances and communications noise) that threaten a robot collective. For each hazard, an analysis is performed to determine the impact of that hazard on the robot collective's performance.

In this specific case study of the containment application, the FMEA internal hazards identified were:

- H_1: Motor failure
- H_2: Communications failure
- H_3: Avoidance sensor(s) failure
- H_4: Beacon sensor failure
- H_5: Control systems failure
- H_6: All systems failure

For each of these identified hazards, the effect on each of the application behaviors was analyzed. For example, the effect of the H_1 motor failure hazard in a single robot is to anchor the entire swarm in a fixed region, which could be a potentially serious problem. On the other hand, the effect of the H_2 communications failure in an individual robot is relatively minor, leading to that robot being disconnected from the rest of the swarm. The effect of the H_3 avoidance sensor failure hazard on an individual robot is that that robot may collide with other robots or obstacles. The effect of the H_4 beacon sensor behavior is considered negligible, in that the worst possibility is a slight slowdown in the swarm taxis behavior. The effect of the H_5 control systems failure is determined to be that a single robot leaves the swarm, or that the robot becomes stationary or turning in the spot, leading to swarm anchoring. Finally, the effect of hazard H_6, total systems failure, is considered to be benign, as the affected robot will be treated as a static obstacle to be avoided.

In summary, this FMEA analysis shows that the robot collective is quite robust to many types of hazards due to parallelism, redundancy, and the distributed nature of the solution. However, while the collective is highly tolerant of the complete failure of some robots, the system was much more vulnerable to partially failed robots, such as robots whose motors fail, but whose other subsystems continue to operate properly. Such partially failed robots can have the effect of preventing the entire collective from achieving their task. From this analysis, Winfield and Nembrini concluded that: (1) any study of the fault tolerance in robot swarms must consider the effect of partial robot failure, and (2) future safety-critical swarms need explicit techniques designed into the system to counteract the effect of the partial robot failures. Winfield, et al., also propose the need for a new discipline of swarm engineering to achieve dependable swarms [Winfield *et al.* (2005)]; at present, however, such techniques are still in their infancy.

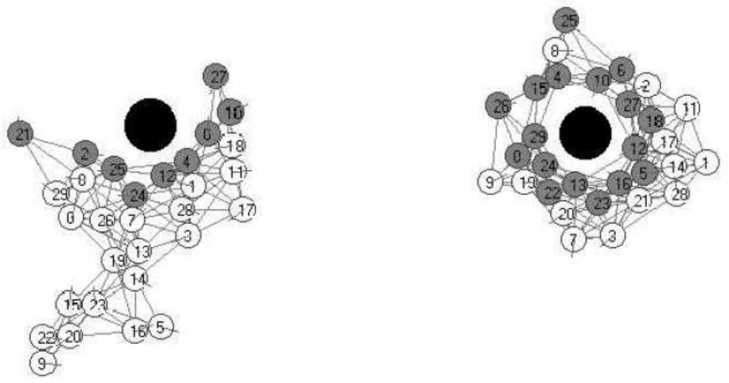

Figure 6.1. Robot collective performing the encapsulation task; left image shows encapsulation in progress, while right image shows the completed task (from [Winfield and Nembrini (2006)]).

6.4.2 *Quantitative Metrics*

Quantitative metrics are useful in most engineering systems for allowing the inter-comparison of alternative designs. This is also certainly true for measuring reliability and fault tolerance in intelligent robotic systems. Evans and Messina [Evans and Messina (2000)] analyzed the importance of defining universally accepted performance metrics for intelligent systems. They also 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.

Standard engineering metrics, such as MTBF (Mean Time Between Failures) and Availability have proven useful in analyzing the failures of individual robots. The MTBF metric is defined as:

$$\text{MTBF} = \frac{\text{Number of hours robot is in use}}{\text{Number of failures encountered}} \tag{6.1}$$

The availability metric is defined as:

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF + MTTR}} \times 100 \tag{6.2}$$

where MTTR = Mean Time To Repair is given by:

$$\text{MTTR} = \frac{\text{Number of hours spent repairing}}{\text{Number of repairs}} \tag{6.3}$$

Carlson and Murphy [Carlson and Murphy (205)] use these metrics to compare and contrast the performance of several types of physical robots in the field. While these metrics focus on individual robots, such metrics could also be useful for multi-robot systems.

Unfortunately, relatively little work has studied metrics for robot collectives. The challenge is that robot collectives cannot typically be viewed as a set of independent and redundant mechanical components. Instead, their interactions and intelligence must be taken into account to determine their overall system reliability and fault tolerance. In particular, the interactions between robots in collectives means the independence assumption between robots does not hold. Further, the intelligence and/or learning capabilities of robot collectives mean that the team may be able to use reasoning to overcome certain failure modes. Such capabilities are not captured in typical redundancy-based engineering metrics.

Because of these characteristics, it is non-trivial to develop quantitative metrics that can measure the degree of fault tolerance and reliability in robot collectives. As a result, most existing multi-robot 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 and reliability are not explicitly identified, and instead are hidden in the application-specific measures.

The following subsections outline some existing work on defining and evaluating metrics specifically tailored to robot collectives.

6.4.2.1 *Reliability Models*

Many traditional engineering methods that address fault-tolerance predominantly deal with reliability analysis of systems and components. 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.

Winfield and Nembrini [Winfield and Nembrini (2006)] have investigated a number of techniques for modeling the reliability of a robot collective, given the known reliability of individual components of the system [Elsayed (1996)]. One approach to model the reliability, R, of a group of n robots is to assume that they are independent and have an equal probability of failure, p. Then, the system failure probability is simply the product of the robot failure probabilities, $R = 1 - p^n$. However, this model is purely based on redundancy (similar to [Hamilton *et al.* (1996)]), and does not model the fact that the overall system will likely not function properly if very few of the robots remain operational.

A second approach to reliability modeling investigated in [Winfield and Nembrini (2006)] is a *load-sharing* approach, in which the failure of one component of the system increases the probability that another component will fail, with the probabilities escalating with each additional failure. However, Winfield and Nembrini argue that this approach may not always be appropriate, since the failure of one or more robots does not necessarily mean that the workload of other robots increases.

The third reliability model explored by [Winfield and Nembrini (2006)] is a *multi-state* approach, in which the robots can be assumed to be in one of three possible states: fully operational, partially operational (state s_o), or completely failed (state s_f). If the probability of robot failure in state s_o is p_o and the probability of failure in state s_f is p_f, then the reliability of one robot is given by $1 - p_o - p_f$. Thus, for n robots in the collective, the total system reliability, R, could be modeled as $R = (1 - p_o)^n - p_f^n$. This equation can then be used to find the optimal number of robots for the collective by taking the derivative with respect to n, equating to 0, and solving for n.

Winfield and Nembrini [Winfield and Nembrini (2006)] argue that further work on reliability modeling should study the k-out-of-n reliability model, in which k is the minimum number of robots needed to achieve acceptable performance. An additional avenue of study that they also recommend is a combined multi-state k-out-of-n reliability approach, such as [Huang *et al.* (2000)].

6.4.2.2 *Effectiveness Metrics*

Kannan and Parker [Kannan and Parker (2007)] have defined metrics for fault tolerance within multi-robot teams.

This work emphasizes the **effective fault tolerance** of the system, which takes into account the ability of an intelligent multi-robot system to overcome failures not only through redundancy, but also through intelligent reasoning about the cause of the failure.

This approach was informed by the work of Hamilton, et al. [Hamilton *et al.* (1996)], who outlined a metric for calculating the "effective" fault-tolerance for single robot manipulators by combining the observed fault tolerance with the performance benefits and costs of including the fault tolerant mechanisms. However, the work of Hamilton, et al., was based purely on redundancy, and does not capture the use of intelligence or reasoning to compensate for failure. It also has not been applied to multi-robot systems.

The work of Kannan and Parker [Kannan and Parker (2007)] defines metrics for evaluating the fault tolerance and efficiency of a multi-robot system. Rather than being predictive measures, these metrics are applied after the robot team has performed a set of tasks, to evaluate the team's performance in retrospect. The use of these metrics assumes that the robot team has a set of discrete tasks to perform, and that the outcome of each task attempt is either *success* or *failure*. Failed tasks can be re-attempted, either by the same robot team member or by other robots on the team. Additionally, multiple faults can occur while attempting a single task.

A fault tolerance metric is defined that is based on the ratio of successfully executed tasks to total tasks. An *efficiency* metric is also defined to measure the robot team's efficiency in handling failures, measured as the average ratio of the total time spent in accomplishing a task divided by the total time spent in fault detection, recovery, and task execution. Finally, a *learning* metric is defined in terms of the change over time of the efficiency of the robot team.

In more detail, the formal definition of the problem for which metrics have been developed [Kannan and Parker (2007)] is given as follows. First, it is presumed that a set of robots and tasks are given, defined as:

- 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.

In this context, the following assumptions are made:

- 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., $c_j = u_j$.

The total number of faults for an i^{th} attempt of task T_j is defined as the summation of all encountered faults during the course of task execution. That is, $F^i_j = f^i_{known_j} + f^i_{unknown_j} + f^i_{undiagnosable_j}$. F^i_j 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:

$$A_j = u_j \tag{6.4}$$

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

$$A = \sum_{j:T_j \in X} u_j \tag{6.5}$$

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, a task failure metric, B^i_j, is associated with 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, robustness can be used to quantify the extent of fault-tolerance in the system. The notation ρ^i_j gives the measure of robustness for the i^{th} attempt of task T_j, defined as:

$$\rho^i_j = \frac{f^i_{known_j} + f^i_{unknown_j}}{F^i_j} \tag{6.6}$$

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

Based on equation 6.6, the task failure metric for the i^{th} attempt of task T_j is:

$$B^i_j = c_j * \left(1 - \rho^i_j\right) \tag{6.7}$$

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

$$B_j = \sum_{i=1}^{q_j} (c_j * (1 - \rho^i_j)) \tag{6.8}$$

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 6.8 across all task failures, gives:

$$B = \sum_{j:T_j \in Y} (c_j) * (q_j - \sum_{i=1}^{q_j} \rho^i_j) \tag{6.9}$$

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.,

$$P = A - B \tag{6.10}$$

Substituting for A and B from equations 6.5 and 6.9 respectively, the desired effective performance metric is obtained:

$$P = \sum_{j:T_j \in X} u_j - \sum_{j:T_j \in Y} (c_j * q_j) * (q_j - \sum_{i=1}^{q_j} \rho_j^i) \tag{6.11}$$

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, it is important to identify additional individual metrics that help give a complete picture of the system.

In addition to outlining a measure for performance, it is desirable to identify the fault-tolerance exhibited by the system. As mentioned previously, system fault-tolerance is defined in terms of robustness, efficiency and learning. Combining individual task robustness measures from equation 6.6 for failed task attempts with the robustness value for the successful attempts, system robustness can be represented as:

$$\rho_s = \frac{\sum_{j:T_j \in Y} \sum_{i=1}^{q_j} \rho_j^i + \sum_{q:T_q \in X} \rho_q^1}{|X + Y|} \tag{6.12}$$

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.:

$$t_j = \mathrm{t}_{Normal_j} + \mathrm{t}_{Fault_j} + \mathrm{t}_{Recovery_j} \tag{6.13}$$

$$\epsilon_j = \frac{\mathrm{t}_{Normal_j}}{t_j} \tag{6.14}$$

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:

$$\epsilon = \frac{\sum_{j:T_j \in X} \frac{t_{Normal_j}}{t_j}}{X + Y} \tag{6.15}$$

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 a set of trials, an estimate of the learning exhibited by the system is obtained.

$$\delta_k = \epsilon_k^{'} - \epsilon_k^0 \tag{6.16}$$

$$\delta_u = \epsilon_u^{'} - \epsilon_u^0 \tag{6.17}$$

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.

Additionally, 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.

These metrics were applied to a robot team performing an assistive navigation and deployment task, and were used to compare a multi-robot control approach that was not able to learn about faults with one that could diagnose and learn from faults (i.e., the LeaF system [Parker and Kannan (2006)], discussed in more detail in Sec. 6.6.4). These metrics were able to capture the differences between these systems, whose robot components were the same, but whose intelligent reasoning software was different.

6.5 General Mechanisms for Fault Detection, Diagnosis, and Recovery

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, with few existing standards and protocols for implementing system-level fault-tolerance. The experimental nature of typical mobile robotic equipment and its uncertain interactions with the environment make detailed modeling for multiple failures, such as the approach used in industrial robotics, difficult in teams of mobile robots.

Nevertheless, several approaches have been developed for making collective robotic systems more reliable, fault tolerant, and capable of detecting, diagnosing, and recovering from failures. This section discusses several general techniques that have been proposed. We separate the discussion into three areas – fault detection, fault diagnosis and identification, and fault recovery.

6.5.1 *Fault Detection*

Fault detection for robots is a complex problem, for a number of reasons: the space of possible faults is very large; robot sensors, actuators, and environment models are uncertain; and, there is limited computation time and power available to the robots. Nevertheless, because of its importance, much prior work has been done in this area for individual robot systems. These individual robot techniques can typically be used in a robot collective to detect problems with individual robots. The next subsection describes some of these techniques for individual robots. The following subsections then outline techniques that have been developed for, and/or demonstrated on, robot collectives.

6.5.1.1 *Individual Robot Fault Detection*

The most popular method for providing fault detection in robot systems is based on motion control [Hashimoto *et al.* (2003); Visinsky *et al.* (1994); Lee *et al.* (2003)]. This method compares the values estimated by a pre-defined motion model and the current measurements to detect a fault. For example, in the Hannibal robot system [Ferrell (1994)], if the leg sensors do not agree with the set of plausible leg motions that are programmed for the leg, the robot generates a belief that the sensor is not working. This method only works, however, when the motion model of the robot is completely known.

Another widely used fault detection method is voting based on modular redundancy [Jackson *et al.* (2003); Chen *et al.* (2006)]. This method is commonly used in highly reliable systems in which more than one module works redundantly to perform the same task given the same input data. If one of the modules is faulty and its result does not agree with the results of the other modules, the faulty module is voted out of the final decision and the correct result is passed on to the rest of the system.

Analytical redundancy is another concept for fault detection that does not need redundant modules [Leuschen *et al.* (2002); Jeppesen and Cebon (2004); Garcła *et al.* (2000)]. By comparing the histories of sensor outputs versus the actuator inputs, results from dissimilar sensors can be compared at different times in order to check for failures.

In recent years, particle filter techniques for robot fault detection have become popular [Goel *et al.* (2000); Verma and Simmons (2006); Cai and Duan (2005)]. This method can estimate the robot and its environmental state from a sequence of noisy, partial sensor measurements. Most particle filter based fault detection methods work with a known set of possible fault types.

Several data-driven techniques have been developed to allow a robot to learn normal models of operation, which can then be used for detecting faults. Commonly used techniques to extract system knowledge from data include decision trees [Yang *et al.* (2001)], artificial neural networks [Sadeghi *et al.* (2005)] and Bayesian networks [Zhou and Sakane (2002); Delage *et al.* (2006)]. Matsuura, et al., [Matsuura and Yoneyama (2004)] present a fault detection method based on Bayesian networks that does not require previous information about the dynamic system. In their work, a Bayesian network is learned from a series of data acquired under normal conditions, and faults are detected as low probability values.

Other fault detection methods construct a system behavior model in the form of a set of rules generated by applying pattern clustering and association. This approach has been used in some complex systems [Yairi *et al.* (2001)] outside of robotics. Hybrid control systems can be used to identify generic patterns of continuous and discrete event dynamical systems [Branicky *et al.* (1998)]. A generic framework for hybrid systems includes transitions between continuous and discrete states.

In more recent years, researchers have taken inspiration from biological immune systems [Timmis *et al.* (2004)] to develop techniques for robot fault detection [Canham *et al.* (2003)]. For example, in [Canham *et al.* (2003)], the authors recognize that biological systems are amazingly robust, able to survive injury, damage, wear and tear, and to withstand continual attack from infectious pathogens. Their approach is based on the concept that during the life of the individual, it is able to differentiate between self (i.e., that which is normally present) and non-self (i.e., that which is normally absent). Their work defines detectors that consider a segment of the feature space, determining whether the value of the system lies in the self, or non-self, region. Learning techniques are used to determine the appropriate size of the detectors. This approach has been successfully applied for fault detection in two robot applications – one for an obstacle avoiding robot, and a second for a robot motion controller.

6.5.1.2 *Health Signal in Robot Collectives*

A commonly used and simple technique for fault detection in robot collectives is the use of a *health signal* to determine if robot team members are functioning properly. The justification for such an approach is that it is very common for robots to experience total software failure, resulting in a non-functioning robot. If robots are designed to periodically broadcast a health signal indicating their presence and continued operation, other robots can presume that such robots are operating properly, and without fault. Examples of the use of this technique in robot collectives include [Parker (1998); Christensen *et al.* (2009)], which are summarized as case studies in Secs. 6.6.1 and 6.6.2.

6.5.1.3 *Data-Driven Models in Robot Collectives*

If it were possible to analytically model the expected interactions among the team members in a robot collective, such models could be used to detect problems when they occur. However, it is often quite difficult to generate comprehensive analytical models of robot collectives, as there are too many unknowns in these complex systems. An alternative approach is to build data-driven models of robot collectives, which are based on statistical techniques that make use of sensor, control, and/or communicated data during "normal" operations of the robot collective. The team experiences during task execution can then be compared to the built statistical models to determine when problems have arisen. An example of the use of this technique in robot collectives is given in [Li and Parker (2007, 2009)], which is also summarized as a case study in Sec. 6.6.3.

6.5.2 ***Fault Diagnosis and Identification***

The fault diagnosis and identification process goes beyond simply detecting that a fault has occurred, focusing instead on determining the specific cause of a detected fault. The topic of fault diagnosis and identification has been studied both for individual robots, and for robot collectives. The following subsections outline some common techniques in these areas.

6.5.2.1 *Individual Robot Fault Diagnosis and Identification*

Many of the techniques mentioned in Sec. 6.5.1 for fault detection in single robots also have the ability to diagnose and identify the faults that have occurred. For example, the decision-theoretic particle filter approach of [Verma and Simmons (2006)] models all known failure modes (such as slipping wheels, faulty encoders, stuck wheels, etc.). When the system determines with high probability that the robot is in a failure state, the identity of that state is also known, which corresponds to the specific cause of the failure.

The work of Liu and Coghill [Liu and Coghill (2005)] presents a model-based approach for online fault diagnosis in individual manipulator robots, called the First Priority Diagnostic Engine (FPDE). This approach is based on defining the range of acceptable values of key variables of the system. When variables are out of bounds, an interval filter is used to distinguish between actual faults and noisy measurements. This is followed by a component-based reasoner that performs further analysis about orientation and translational faults.

Kaminka and Tambe present an approach for monitoring and diagnosis for multi-agent domains called SAM — Socially Attentive Monitoring [Kaminka and Tambe (1998)]. 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.

Another technique that addresses fault diagnosis, as well as detection and recovery, is the work of Murphy and Hershberger [Murphy and Hershberger (1999)], who suggested a two-step approach: a strategy for classifying sensor failures, and a recovery strategy. Their Sensor Fusion Effects Architecture (SFX-EH) for handling sensing failures in autonomous mobile robots is based on this two-step methodology, using extensions to the generate-and-test method (which was originally developed for medical diagnosis [Lindsay *et al.* (1980)]) to classify failures based on a partial causal model of the sensor/environment/task interactions for the robot. In this approach, the robot generates hypotheses about the causes of sensor failures and then executes tests to determine which hypotheses are correct. Recovery methods are then linked to each possible cause of failure.

6.5.2.2 *Causal Models in Robot Collectives*

Autonomous systems usually can benefit from prior domain knowledge that is built into the system. One way to provide this domain knowledge for diagnosing and identifying faults is the Causal Model Method (CMM) [Hudlická and Lesser (1987)], which identifies common reasons for faults occurring in a system, modeled as a decision graph. The CMM was initially designed 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.

The SFX-EH approach mentioned in the previous subsection is an example technique that makes use of partial causal models. Even though SFX-EH was primarily designed for single robot fault diagnosis, work by Long, et al. [Long *et al.* (2003)], has investigated extending the SFX-EH architecture from a single robot to a small team of distributed robots. While the original SFX-EH architecture achieved real-time constraints by pre-computing shortest-time decision trees, this approach does not easily scale to multi-robot teams because it is difficult to update the decision trees dynamically, as would be necessary in a multi-robot application. Instead, the multi-robot version dynamically determines the proper ordering of tests at runtime, based on timing information and interdependencies of tests, which can be distributed across multiple robots. In this approach, robots share knowledge about the working environment and sensor and task states. Additionally, robots communicate with each other to diagnose failures and to redistribute tasks in case a robot becomes inoperable.

While static causal models have been shown to be beneficial in robot collectives for determining likely causes of faults that occur in the team, a major drawback of such causal models is that they require the designer to anticipate all possible faults that may occur during execution. In practice, this is very difficult to do, as the inherent explosion of state space complexity [Atkins *et al.* (1997)] for multi-robot teams operating in dynamic environments inhibits the ability of any designer to anticipate all possible failure modes in advance. To deal with these challenges, one approach is to begin with a manually-defined causal model, but allow the robot team to adapt the model over time as new types of faults are experienced. This adaptive causal model approach has been reported in [Parker and Kannan (2006)], and is summarized in Sec. 6.6.4 as a case study.

6.5.3 *Fault Recovery*

Fault recovery strategies in multi-robot collectives can vary significantly. In swarm-type robot collectives, some algorithms can be shown to be inherently robust to a limited number of robot failures, thus not requiring any explicit actions for fault recovery. In some cases, it may not be necessary to identify the failed robot, whereas other cases do require knowledge of which robot has failed. In the latter case, a technique such as the synchronous flashing light approach described in Sec. 6.6.2 could be used to identify the specific robots that have failed.

Furthermore, in some situations, it may not be necessary to identify the specific failure in order to successfully recover from the failure. Instead, alternative behaviors could be tried until the desired performance is achieved. An example of this approach is the work of Payton, et al., [Payton *et al.* (1992)], which handles undiagnosable events by activating a redundant behavior that uses a complementary or redundant sensor and/or actuator. This approach does not attempt to determine the cause of a failure.

In the ALLIANCE architecture [Parker (1998)] (which is described in more detail in Sec. 6.6.1), problems with the performance of individual robots are detected, but no attempt is made to determine the specific cause of the fault. Instead, the recovery strategy is for another available robot to take over the task from the failing robot.

In other types of robot collectives, especially those involving smaller numbers of robots that intentionally cooperate, more explicit means for fault recovery are needed. Some of the more intentional techniques for individual robots outlined in Secs. 6.5.1 and 6.5.2 can also be used for fault recovery. Most notably, the SFX-EX approach [Murphy and Hershberger (1999); Long *et al.* (2003)] associates with each failure type a recovery mechanism. In this approach, three recovery strategies are defined: *reconfiguration*, which either replaces a sensor with an alternative logical sensor, or replaces the entire behavior with a new behavior; *recalibration*, which recalibrates the sensor or actuator; and, *corrective actions*, which are stored subroutines to be executed to achieve fault recovery.

6.6 Case Studies

A number of robot collectives have been developed that demonstrate some degree of fault tolerance and reliability. This section explores four (4) case studies that successfully illustrate some of the techniques outlined in this chapter. The first case study on ALLIANCE demonstrates the use of health signals and simple robot capability modeling for fault detection and recovery. The second case study on fault detection using synchronous flashing lights illustrates a technique for identifying failing robots in swarm robot collectives via a type of health signal that does not require wireless communication. The third case study on SAFDetection makes use of a data-driven approach to model the normal behavior of a collective robot system, which can then be used to detect faults that occur in the team. The final case study on LeaF illustrates the use of adaptive causal models for fault diagnosis and recovery in multi-robot teams.

6.6.1 *ALLIANCE: Dealing with Faults through Robot Modeling*

One of the earliest works addressing fault tolerance in robot collectives is the ALLIANCE architecture [Parker (1998)], which enables robots to learn simple models about the capabilities of other robots; these models can then be used to allow a robot to determine that a task is not being solved properly, or that a robot has failed (or seriously degraded). At the time of its development, the primary method for task allocation in robot teams depended upon inter-robot communication and negotiation, via mechanisms such as the Contract Net protocol [Smith (1980)]. However, such negotiation techniques typically provided no mechanism for the team to continue its work when communication failed. As wireless communication failure is typical in multi-robot applications, the ALLIANCE approach was designed to ensure that robots could continue to make progress on their tasks, even if the robots were not able to talk with each other, or if the robots failed. However, if wireless communications were available, the ALLIANCE architecture would make use of it by having robots periodically broadcast a health signal that indicated the robot's identity and current activity.

Further, ALLIANCE was designed to emphasize the importance of robots demonstrating their ability to successfully achieve their tasks, in terms of the effect the actions had on the world itself. Thus, even if a robot were to announce that it was performing a particular task, other robots would continue to monitor that robot's effect on the world, to ensure that the expected changes are occurring as a result of that robot performing the task. To accomplish this, the ALLIANCE approach built simple models of the expectations for how quickly tasks should be performed when being worked on by particular robots. If the environment did not change quickly enough, according to these simple models, other robots may decide to take over that task, since the original robot was not able to demonstrate its ability to actually accomplish the task through its effect on the world.

The ALLIANCE approach (shown in Fig. 6.2.), builds on the subsumption architecture [Brooks (1986)] by adding behavior sets and motivations for achieving action selection without explicit negotiations between robots. Behavior sets group low-level behaviors together for the execution of a particular task. The motivations consist of levels of impatience and acquiescence that can raise and lower a robot's interest in activating a behavior set corresponding to a task that must be accomplished.

In this approach, the initial motivation to perform a given behavior set is equal to zero. Then, at each time step, the motivation level is recalculated based on (1) the previous motivation level, (2) the rate of impatience, (3) whether the sensory feedback indicates the behavior set is needed, (4) whether the robot has another behavior set already activated, (5) whether another robot has recently begun work on this task, and (6) whether the robot is willing to give up the task, based on how long it has been attempting the task. Effectively, the motivation continues to increase at some positive rate unless one of four situations occurs: (1) the sensory feedback indicates that the behavior set is no longer needed, (2) another behavior set in the robot activates, (3) some other robot has just taken over the task for the first time, or (4) the robot has decided to acquiesce the task. In any of these four situations, the motivation returns to zero. Otherwise, the motivation grows until it crosses a threshold value, at which time the behavior set is activated and the robot can be said to have selected an action. When an action is selected, cross-inhibition within that robot prevents other tasks from being activated within that same robot. When a behavior set is active in a robot, the robot broadcasts its current activity to other robots at a periodic rate.

The L-ALLIANCE extension [Parker (2000)] allows a robot to adapt the rate of change of the impatience and acquiescence values depending on the quality with which that robot is expected to accomplish a given task. The result is that robots that have demonstrated their ability to better accomplish certain tasks are more likely to choose those tasks in the future. Additionally, if problems occur during team performance, then robots may dynamically reallocate their tasks to compensate for the problems.

This approach was demonstrated on a team of three heterogeneous robots performing a mock clean-up task, two robots performing a box-pushing task, and

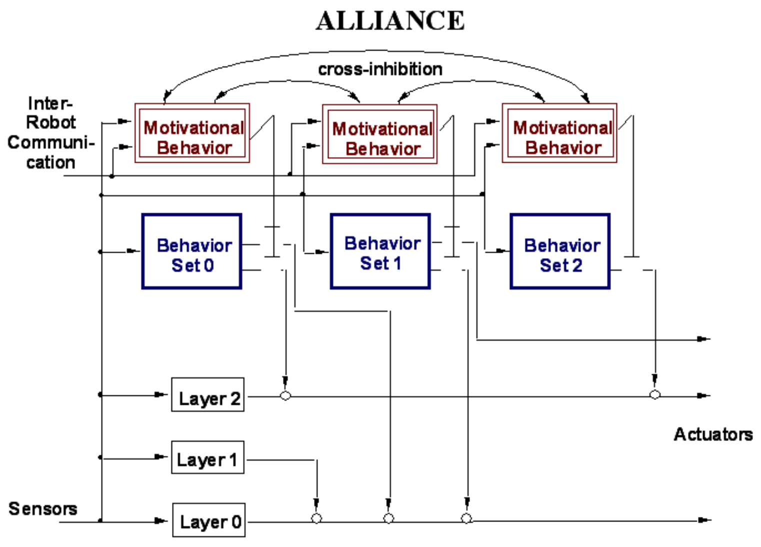

Figure 6.2. The ALLIANCE architecture (from [Parker (1998)]).

four robots performing a cooperative target observation problem. The approach has also been demonstrated in simulation on a janitorial service task and a bounding overwatch task. Examples of the types of faults and unexpected events that could be detected, and which resulted in an automatic reallocation of tasks between robot team members, included a robot becoming trapped (and thus not able to complete its task in the expected time frame), robots completely failing (or being removed from the team), and a change in robot team composition.

6.6.2 *From Fireflies to Fault Tolerant Swarms*

While swarm-type multi-robot collectives are envisioned as systems that can achieve fault tolerance through redundancy, it is often necessary to explicitly manage the failures that might occur. The work of Christensen, et al., [Christensen *et al.* (2009)] addresses these concerns by developing a decentralized system that detects non-operational robots in a swarm by engineering a flashing light system on the robots. This flashing light system is reminiscent of some species of fireflies who can synchronize their flashing. This robotic approach creates the ability for operational robots to flash in unison; failed robots can thus be detected as those that are not flashing in synchrony with the rest of the robot team. This approach, thus, is similar to the health signal discussed in Sec. 6.5.1.2.

The robotic swarm in this work consists of several *s-bots* (see Fig. 6.3.), that can physically connect with each other to cross navigational hazards or to cooperatively transport objects. Each robot also has several colored LEDs that can be detected by the onboard omnidirectional camera on each robot. The synchronized flashing is achieved by the use of pulse-coupled oscillators, which influence other oscillators during short, periodic pulses. As described in [Christensen *et al.* (2009)], the activation of each oscillator increases over time until it reaches a threshold, at which time the oscillator fires and its activation returns to zero. The cycle then repeats. When an oscillator observes a nearby flash, it increases its own activation by a small amount. Theoretical results have shown that a population of pulse-coupled oscillators almost always transitions to a state in which the oscillators fire in unison [Mirollo and Strogatz (1990); Lucarelli and Wang (2004)]. Once the synchronization has been achieved, any robot that is not flashing in synchrony can be assumed to have experienced a failure. Robots can also intentionally stop flashing when they detect an internal error.

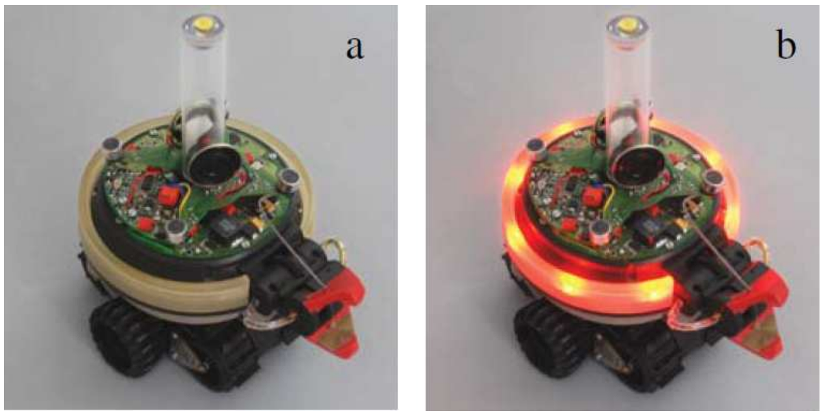

Figure 6.3. The s-bots of [Christensen *et al.* (2009)]. Image (a) shows the s-bot with no LEDs illuminated; image (b) shows the s-bot with its red LEDs illuminated (from [Christensen *et al.* (2009)]).

This synchronization approach has been successfully demonstrated on up to 100 robots in simulation, and on 10 physical robots. Studies showed that moving robots synchronized faster than static robots, and that the speed of synchronization was inversely proportional to the robot density. In the most favorable situations, the synchronization time was around 1 to 1.5 minutes. Because the synchronization takes some time, the preferred approach is to introduce a warm-up time period (of about two minutes), in which the robots ignore asynchrony. After this time, the robots can treat another robot that is not flashing in unison as a failed robot. This approach was shown to successfully detect robot faults in physical robot experiments.

6.6.3 *SAFDetection: Sensor-Based Modeling for Fault and Anomaly Detection*

The SAFDetection system (<u>S</u>ensor <u>A</u>nalysis for <u>F</u>ault <u>Detection</u>) [Li and Parker (2007, 2009)] is a data-driven approach for modeling and detecting faults in robot collectives that perform tightly-coupled multi-robot team tasks. Unlike motion model based methods, the SAFDetection approach does not require knowledge of the internal control of the robot system, nor of advance knowledge of the possible fault types, unlike the particle filter approaches [Verma and Simmons (2006); Goel *et al.* (2000); Cai and Duan (2005)]. Additionally, no functionally redundant modules are required and no requirement is made for specifying the relationship between the measured variables, unlike the analytical redundancy methods [Leuschen *et al.* (2002); Jeppesen and Cebon (2004); Garcła *et al.* (2000)]. Instead, the SAFDetection approach is a data-driven technique that learns a probabilistic robot state transition diagram from the histories of robot sensor data during normal operation based on a clustering algorithm. This model is then used online, together with the online sensor data, to detect faults in a real-time fashion. Since this approach does not require *a priori* motion models to be built by the designer, SAFDetection is viewed as a *black box* technique that can be used as a wrapper around a variety of different robot team behaviors, without needing to change the internal control software.

SAFDetection has two implementations – one is centralized, which regards the complete multi-robot team as one monolithic robot with a unified set of sensors, and the second is distributed, which allows each robot to build its own model from its perspective of the robot collective's task execution. Fig. 6.4. shows the structure of the centralized version, while Fig. 6.5. shows the structure of the distributed version for the training stage.

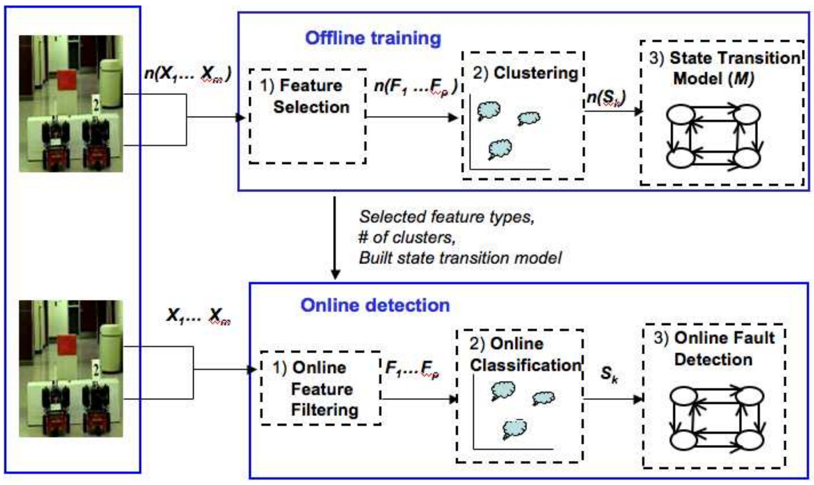

Figure 6.4. The structure of the centralized SAFDetection approach (from [Li and Parker (2009)]).

The SAFDetection approach is a training-classification based method. In the training stage, a history of sensor data (i.e., training data) during normal operation is clustered into different states by using the Fuzzy C-Means (FCM) clustering algorithm [Klawonn and Keller (1997)]. A state transition diagram that represents the probabilistic transitions between these states is then learned from the training data. In the classification stage, the on-line sensor data is compared with the state transition model and three types of faults can be detected. If the sensor data does not belong to any of the states learned from the normal data, a *hard fault* is detected. If the sensor data belongs to one of the states but the observed state transition deviates significantly from the learned state transitions, a *logic fault* is detected. In a

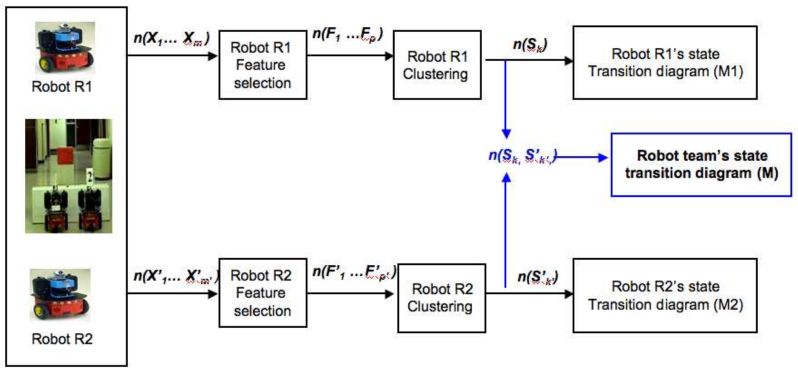
Robot R1
Robot R2
n(X1,... Xm)
n(X'1,... X'm')
Robot R1 Feature selection
Robot R2 Feature selection
n(F1 ...Fp)
n(F'1 ...F'p')
Robot R1 Clustering
Robot R2 Clustering
n(Sk)
n(S'k')
n(Sk, S'k')
Robot R1's state Transition diagram (M1)
Robot R2's state Transition diagram (M2)
Robot team's state transition diagram (M)

Figure 6.5. The training stage in the distributed SAFDetection approach (shown for 2-robot team, for clarity; the approach scales to larger teams) (from [Li and Parker (2009)]).

similar manner, when this approach is used in a robot collective, the inconsistency between robots can be detected as a *coalition fault*. If no fault is detected, the sensor data is classified as normal and is used to update the probabilistic state transition diagram.

In more detail, the clustering approach during learning makes use of pre-defined features computed from the sensor data, rather than all of the available raw sensor data. Selecting the correct components of the feature vector is a non-trivial problem, since the computational demands will be intractable if the system makes use of every possible feature in the system. Many methods can be used to reduce the feature dimension, including Principal Components Analysis (PCA) and Singular Value Decomposition (SVD), which are two of the common linear algebra techniques for continuous data dimensionality reduction. SAFDetection makes use of PCA for this purpose.

Once the sensor feature data is obtained from the normal operation of the robot team performing the task, this data must be clustered into states, reflecting the various operational modes of the robot team. SAFDetection makes use of the Fuzzy C-Means clustering algorithm (FCM) [Klawonn and Keller (1997)] for this purpose. Since FCM is a fuzzy algorithm, a single data pattern may belong to several clusters, having different membership values in each cluster. This property of fuzziness is advantageous when dealing with the noisy or partial data of typical robot applications. One limitation of FCM is that it requires knowledge of the number of clusters c, which is typically unknown before clustering is performed. Thus, SAFDetection iteratively runs the FCM algorithm over several trials with varying cluster numbers and selects the number of clusters that gives the best clustering quality, using the measurement defined by Xie [Xie and Beni (1991)].

Once the states are determined from the clustering step, a probabilistic state transition diagram is built to capture the normal robot team transitions between states. This diagram is similar to a Markov model, in that it records states and the transition probabilities between each pair of states. In addition, the SAFDetection state transition diagram also includes the mean and standard deviation value of the time duration of the system in each state (i.e., before the system transits to another state). This model could be replaced with a variable length Hidden Markov Model, although the modeling process would likely take longer.

Once the state transition diagram is built, it can be used to detect faults online. The on-line sensor data and its cluster membership values are sent to the fault detection module and three types of faults can be detected. If the membership value does not show clearly which cluster the data belongs to, a *hard fault* is detected, meaning that the robot has entered an unknown state. If the sensor data shows that a robot has remained in particular state for an unusually long time, a *logic fault* is detected, indicating that the robot is stuck in that state. Finally, if a state transition occurs and the observed state transition does not exist in the learned state transition diagram, a logic fault is detected because the robot has changed its state in an unknown manner. If one robot team member detects a hard or logic fault, but this fault is not detected by another team member, a *coalition fault* is detected indicating inconsistency between team robots (i.e., in their perception of the current team state).

The SAFDetection approach was implemented on a physical robot team of Pioneer robots performing a cooperative box pushing task, as well as variants of the multi-robot leader-follower task. Sensor features identified for this task included minimum laser range, index of minimum laser range, robot speed, robot turn rate, etc. Fifteen (15) trials of the robots performing the assigned tasks in normal operation were used by the robots to learn the model of normal operation. The experimental results showed that this approach could successfully detect a variety of multi-robot team faults during the online task execution. Example faults detected online for the box pushing task include a robot getting stuck against an obstacle, removing sight of the goal (to which the box should be pushed), and communication failures. Example faults detected online for the multi-robot following task include robots losing track of the leader, or robots tracking spurious (non-robot) objects. Results also showed that, while the centralized SAFDetection approach worked for smaller tasks (involving two robots), it was not able to handle larger tasks involving five robots. However, the distributed SAFDetection approach was shown to scale well for to larger team sizes.

6.6.4 *LeaF: Adaptive Causal Models for Fault Diagnosis*

An example system for diagnosing and identifying faults in multi-robot teams is the LeaF (for Learning-based Fault diagnosis) system [Parker and Kannan (2006)], which uses an adaptive causal model for representing possible faults in the system. The pre-specified partial causal model provides the team with knowledge from the designer in advance of the application. When a new fault is encountered, the system uses a case-based reasoning (CBR) approach to attempt to extract a relevant recovery action from prior experience. The causal model is then adapted to include the new fault information, making it available for future diagnosis and recovery. The ability of the system to effectively learn from its own faults makes it a more robust and efficient team that is better suited for practical application.

Fig. 6.6. shows the LeaF architecture, which combines typical robot control processes with modules for adaptive fault diagnosis and recovery. In this architecture,

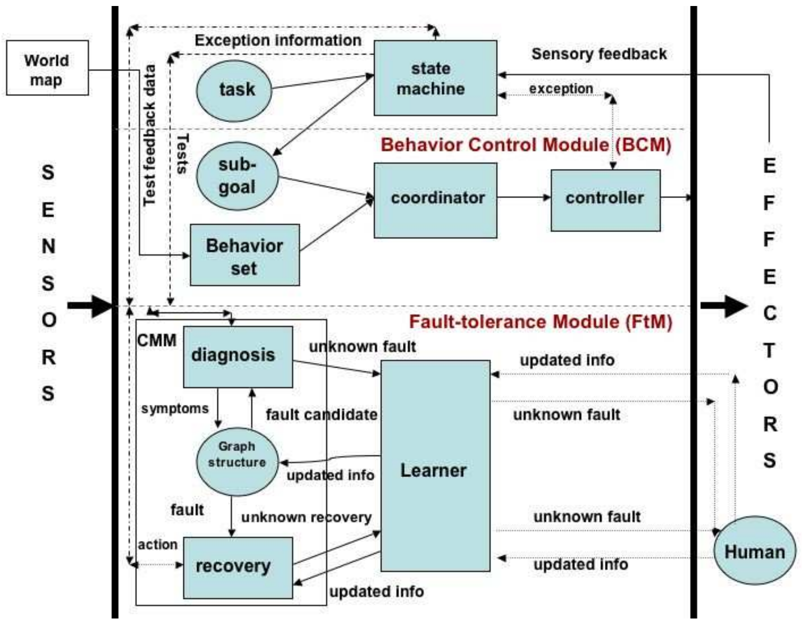

Figure 6.6. Architectural details of LeaF (from [Parker and Kannan (2006)]).

the *Behavior Control Module* (BCM) contains application-dependent behaviors that enable the robot to accomplish its objectives within the given application. The *Fault-tolerance Module* (FtM) is responsible for fault diagnosis and recovery, and learning to adapt the causal model from experience. This module consists of two main blocks – the causal model (CMM) and the Learner. The CMM block contains the *Diagnosis* and *Recovery* sub-modules, and are based on an extended version of the SFX-EH architecture [Murphy and Hershberger (1999)] to diagnose the failure. Using SFX-EH, the robot generates tests about the possible cause for failure, analyzes the resulting data and attempts to diagnose the failure by comparing with the nodes of the causal model. Since all nodes of the causal model have an action associated with them, when the cause for a fault is determined, the corresponding corrective action in the causal model is selected and executed.

When unknown faults occur, the predefined causal model is inadequate for fault diagnosis and recovery. Using case-based reasoning, a new fault can be diagnosed by identifying one or several previously classified faults stored in the causal model to which it is similar and by adapting known solutions, rather than working out a new solution from scratch. In this approach, each capable robot maintains a copy of the causal model (perhaps in collaboration with its teammates) and the LeaF learner, and attempts to identify the new fault. In the event the LeaF learner cannot classify the fault, the robot communicates the relevant information to a human operator for assistance in updating the causal model.

The identification of similarities between faults is achieved using Armengol and Plaza's technique called Lazy Induction of Descriptions (LID) [Armengol and Plaza (2001a)]. LID builds a symbolic similarity description, *similitude* [Armengol and Plaza (2001b)], for the fault, finding the best match to one or more nodes in the causal model. 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 reduces the overall system complexity and the time spent in fault diagnosis.

The LeaF system was applied to a heterogeneous robot team of 3-5 robots performing a variety of tasks, such as assistive navigation, path planning, localization, etc., using a variety of sensors, including laser range sensors and cameras, as well as wireless communication. The LeaF approach was compared to a fixed causal model (CMM) to determine the benefit of the learning capability for fault diagnosis. Specific comparisons were made for a laser fault (a previously known type of fault) and a camera fault due to environmental changes (a previously unknown fault type); both of these faults prevented the cooperation from proceeding properly. The results showed that the CMM approach performs comparably with the LeaF system in the case of the known fault. However, if the same error is encountered repeatedly, the learning techniques in LeaF allowed the speed of diagnosis to increase, due to the increasing probability of occurrence of this particular fault in the system. In the case of a previously unknown fault, the CMM system was unable to provide any useful diagnosis even after a long time interval, whereas the LeaF system shows progressive improvement over time. Once LeaF identifies the new error, it adds the new information into the causal model, thus reducing the time taken for subsequent searches.

One limitation of LeaF is that it cannot address undiagnosed faults that are not closely related to any of the existing faults in the causal model. This type of situation is handled in LeaF by communicating available information on the fault to a human for further evaluation.

6.7 Open Challenges

While reliability and fault tolerance has been studied in individual robot systems, much work remains for understanding these issues in the context of robot collectives. A variety of open challenges remain. Some of the most important issues include the following:

- *Metrics:* Application-independent metrics are needed that can evaluate the robustness and reliability of an arbitrary robot collective [Kannan and Parker (2007); Parker (2001)].
- *Partial robot failures:* General techniques for inferring the impact of partially failed robots is needed for a wide variety of robot collective applications, along with designed techniques for addressing these failures [Winfield and Nembrini (2006)].
- *Reliability modeling:* More expressive quantitative models for representing the reliability of robot collectives is needed; these models should lead to techniques for determining the minimum number of robots needed for acceptable robot team performance [Winfield and Nembrini (2006)].

- *Combining domain knowledge with data-driven approaches:* Additional modeling techniques are needed that can combine prior domain knowledge with data-driven techniques for fault detection, diagnosis, and recovery in robot collectives [Li and Parker (2009)].
- *New concepts for fault tolerance:* A better understanding is needed of how the experiences gained from years of study in conventional robotics can contribute to novel concepts for fault tolerance, such as those inspired from biological viruses and immunological systems.

Solutions to these and related problems will lead to robot collectives that can reliably be applied to a wide variety practical applications, in a manner that demonstrates a high degree of robustness and fault tolerance.

Bibliography

Arkin, R. C., Balch, T. and Nitz, E. (1993). Communication of behavioral state in multi-agent retrieval tasks, in *Proceedings of the 1993 International Conference on Robotics and Automation*, pp. 588–594.

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

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 (Springer-Verlag), pp. 44–58.

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)*, pp. 571–576.

Branicky, M., Borkar, V. and Mitter, S. (1998). A unified framework for hybrid control: Model and optimal control theory, *IEEE Transactions on Automatic Control* **43**, 1, pp. 31–45.

Brooks, R. A. (1986). A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* **RA-2**, 1, pp. 14–23.

Cai, Z. and Duan, Z. (2005). A multiple particle filters method for fault diagnosis of mobile robot dead-reckoning system, in *IEEE International Conference on Intelligent Robots and Systems*, pp. 481–486.

Canham, R., Jackson, A. H. and Tyrrell, A. (2003). Robot error detection using an artificial immune system, in *2003 NASA/DoD Conference on Evolvable Hardware (EH'03)*, pp. 588–594.

Carlson, J. and Murphy, R. R. (205). How UGVs physically fail in the field, *IEEE Transactions on Robotics* **21**, 3, pp. 423–437.

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*, pp. 282–286.

Chen, W., Gong, R. and Dai, K. (2006). Two new space-time triple modular redundancy techniques for improving fault tolerance of computer systems, in *IEEE International Conference on Computer and Information Technology*.

Christensen, A., O'Grady, R. and Dorigo, M. (2009). From fireflies to fault-tolerant swarms of robots, *IEEE Transactions o Evolutionary Computation* **13**, 4.

Dailey, K. W. (2004). *The FMEA Handbook* (DW Publishing).

Delage, E., Lee, H. and Ng, A. Y. (2006). A dynamic Bayesian network model for autonomous 3D reconstruction from a single indoor image, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2418–2428.

Elsayed, E. A. (1996). *Reliability Engineering* (Addison Wesley Longman).

Evans, J. and Messina, E. (2000). Performance metrics for intelligent systems, in *Performance Metrics for Intelligent Systems (PerMIS) Proceedings*, Vol. Part II.

Ferrell, C. (1994). Failure recognition and fault tolerance of an autonomous robot, *Adaptive behavior* **2**, 4, p. 375.

Garcła, F. J., Miguel, L. J. and Pern, J. R. (2000). Fault-diagnostic system using analytical fuzzy redundancy, *Engineering Applications of Artificial Intelligence* **13**, pp. 441–450.

Goel, P., Dedeoglu, G., Roumeliotis, S. and Sukhatme, G. (2000). Fault detection and identification in a mobile robot using multiple model estimation and neural network, in

IEEE International Conference on Robotics and Automation, pp. 2302–2309.

Goldberg, D. and Mataric, M. J. (1997). Interference as a tool for designing and evaluating multi-robot controllers, in *Proceedings, AAAI-97*, pp. 637–642.

Goldman, C. V. and Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis, *Journal of Artificial Intelligence Research* **22**, pp. 143–174.

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*, pp. 3073–3080.

Hashimoto, M., Kawashima, H. and Oba, F. (2003). A multi-model based fault detection and diagnosis of internal sensor for mobile robot, in *IEEE International Conference on Robotics and Automation*, pp. 3787–3792.

Huang, J., Zuo, M. J. and Wu, Y. (2000). Generalized multi-state *k*-out-of-*n*: G systems, *IEEE Transactions on Reliability* **48**, 1.

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

Jackson, A. H., Canham, R. and Tyrrell, A. M. (2003). Robot fault-tolerance using an embryonic array, in *NASA/DoD Conference on Evolvable Hardware*, pp. 91–100.

Jeppesen, B. and Cebon, D. (2004). Analytical redundancy techniques for fault detection in an active heavy vehicle suspension, *Vehicle System Dynamics* **42**, pp. 75–88.

Kaminka, G. 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-98)*.

Kannan, B. and Parker, L. E. (2007). Metrics for quantifying system performance in intelligent, fault-tolerant multi-robot systems, in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*.

Klawonn, F. and Keller, A. (1997). Fuzzy clustering and fuzzy rules, in *7th International Fuzzy Systems Association World Congress*, pp. 193–198.

Lee, I. S., Kim, J. T. and Lee, J. W. (2003). Model-based fault detection and isolation method using ART2 neural network, *International Journal of Intelligent Systems* **18**, pp. 1087–1100.

Leuschen, M. L., Cavallaro, J. R. and Walker, I. D. (2002). Robotic fault detection using nonlinear analytical redundancy, in *IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 456–463.

Lewis, M. A. and Tan, K. H. (1997). High precision formation control of mobile robots using virtual structures, *Autonomous Robots* **4**, 4, pp. 387–403.

Li, X. and Parker, L. E. (2007). Sensor analysis for fault detection in tightly-coupled multi-robot team tasks, in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

Li, X. and Parker, L. E. (2009). Distributed sensor analysis for fault detection in tightly-coupled multi-robot team tasks, in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

Lindsay, R. K., Buchanan, E. A., Feigenbaum, E. A. and Ledergerg, J. (1980). *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project* (McGraw-Hill).

Liu, H. and Coghill, G. M. (2005). A model-based approach to robot fault diagnosis, *Knowledge-Based Systems* **18**, 4–5, pp. 225–233.

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

Lucarelli, D. and Wang, I. J. (2004). Decentralized synchronization protocols with nearest neighbor communication, in *Proc. 2nd International Conference on Embedded Networked Sensor Systems* (ACM), pp. 62–68.

Mataric, M. J. (1995). Issues and approaches in the design of collective autonomous agents, *Robotics and Autonomous Systems* **16**, 2-4, pp. 321–331.

Matsuura, J. P. and Yoneyama, T. (2004). Learning Bayesian networks for fault detection, in *Proceedings of the IEEE Signal Processing Society Workshop*, pp. 133–142.

Mirollo, R. E. and Strogatz, S. H. (1990). Synchronization of pulse-coupled biological oscillators, *SIAM J. Appl. Math.* **50**, 6, pp. 1645–1662.

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

Parker, L. E. (1998). ALLIANCE: An architecture for fault-tolerant multi-robot cooperation, *IEEE Transactions on Robotics and Automation* **14**, 2, pp. 220–240.

Parker, L. E. (2000). Lifelong adaptation in heterogeneous teams: Response to continual variation in individual robot performance, *Autonomous Robots* **8**, 3.

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

Parker, L. E. (2008). Multiple mobile robot systems, in *Springer Handbook of Robotics* (Springer Berlin Heidelberg), p. Chapter 40.

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)*.

Payton, D. W., Keirsey, D., Kimble, D. M., Krozel, J. and Rosenblatt, J. K. (1992). Do whatever works: A robust approach to fault-tolerant autonomous control, *Journal of Applied Intelligence* **2**, 3, pp. 225–250.

Sadeghi, M. H., Raflee, J. and Arvani, F. (2005). A fault detection and identification system for gearboxes using neural networks, in *International Conference on Neural Networks and Brain*, pp. 964– 969.

Smith, R. G. (1980). The Contract Net Protocol: high-level communication and control in a distributed problem solver, *IEEE Transactions on Computers* **C-29**, 12.

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)*.

Timmis, J., Knight, T., Castro, L. D. and Hart, E. (2004). An overview of artificial immune systems, in *Computation in Cells and Tissues: Perspectives and Tools for Thought. Natural Computation Series* (Springer), pp. 51–86.

Verma, V. and Simmons (2006). Scalable robot fault detection and identification, *Robotics and Autonomous Systems* **54**, pp. 184–191.

Visinsky, M. L., Cavallaro, J. R. and Walker, I. D. (1994). Robotic fault detection and fault tolerance: A survey, *Reliability Engineering and System Safety* **46**, pp. 139–158.

Winfield, A. F. T., Harper, C. J. and Nembrini, J. (2005). Towards dependable swarms and a new discipline of swarm engineering, in *Simulation of Adaptive Behavior Workshop on Swarm Robotics, Lecture Notes in Computer Science 3342* (Springer-Verlag Berlin Heidelberg), pp. 126–142.

Winfield, A. F. T. and Nembrini, J. (2006). Safety in numbers: Fault-tolerance in robot swarms, *International Journal of Modelling, Identification, and Control* **1**, 1, pp. 30–37.

Xie, X. L. and Beni, G. (1991). A validity measure for fuzzy clustering, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**, 8, pp. 841–847.

Xuan, P., Lesser, V. and Zilberstein, S. (2001). Communication decisions in multi-agent cooperation: model and experiments, in *Proceedings of the fifth international conference on Autonomous agents*.

Yairi, T., Kato, Y. and Hori, K. (2001). Fault detection by mining association rules from housekeeping data, in *International Symposium on Artificial Intelligence,Robotics and Automation in Space*.

Yang, Q., Yin, J. and Ling, C. (2001). Postprocessing decision trees to extract actionable knowledge, in *IEEE International Conference on Data Mining*, pp. 685–688.

Zhou, H. and Sakane, S. (2002). Sensor planning for mobile robot localization using Bayesian network inference, *Advanced Robotics* **16**, 8, pp. 751–771.