Pergamon

0045-7906(94)E0004-Y

Computers Elect. Engng Vol. 20, No. 5, pp. 421–435, 1994
Copyright © 1994 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0045-7906/94 $7.00 + 0.00

EXPERT SYSTEM FRAMEWORK FOR FAULT DETECTION AND FAULT TOLERANCE IN ROBOTICS

M. L. VISINSKY, J. R. CAVALLARO and I. D. WALKER†
Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251-1892, U.S.A.

(Accepted in final revised form 28 June 1993)

Abstract—Fault tolerance is of increasing importance for modern robots. The ability to detect and tolerate failures enables robots to effectively cope with internal failures and continue performing assigned tasks without the need for immediate human intervention. To monitor fault tolerance actions performed by lower level routines and to provide higher level information about a robot's recovery capabilities, we present an expert system and critic which together form a novel and intelligent fault tolerance framework integrating fault detection and tolerance routines with dynamic fault tree analysis.

A higher level, operating system inspired critic layer provides a buffer between robot fault tolerant operations and the user. The expert system gives the framework the modularity and flexibility to quickly convert between a variety of robot structures and tasks. It also provides a standard interface to the fault detection and tolerance software and a more intelligent means of monitoring the progress of failure and recovery throughout the robot system. The expert system further allows for prioritization of tasks so that recovery can take precedence over less pressing goals. Fault trees are used as a standard database to reveal the components essential to fault detection and tolerance within a system and detail the interconnection between failures in the system. The trees are also used quantitatively to provide a dynamic estimate of the probability of failure of the entire system or various subsystems.

Key words: Robotics, fault tolerance, fault detection, expert systems.

1. INTRODUCTION

There is much ongoing interest in deployment of robots in remote and hazardous environments. Applications include hazardous waste management and clean-up [1] and space-based operations [2,3]. In these spheres of operation, the use of humans to perform tasks is limited to short periods of time, and the work is often dangerous as well as difficult. The inherent problems in having humans perform tasks safely in these environments make robots an attractive, if not mandatory, alternative. However, the remoteness of such environments, the safety-critical tasks required, and the difficulty of repairing the robots after failure, make fault tolerance and reliability more critical issues than in conventional applications [4]. Recently, there has been a surge of interest in robot fault tolerance, and the subject has been investigated from a number of points of view. Ongoing work includes analysis of redundant systems [5–10], modular software environments [11–14], fault tolerant control environments [15,16], error recovery [17,18], and fault detection [19–22].

To protect the robot from human error, we turned to operating system structures and developed a critic level for our framework [14]. The critic monitors the user's interaction with the robot and searches for any commands, obstacles or failure situations which might endanger the robot's mission. In earlier work, we developed and verified algorithms which successfully detect and tolerate sensor and motor failures in a specific robot system [21,23]. To apply these algorithms to a wider variety of robots, we proposed an expert system based supervisor which performs robotic fault tolerance using a standardized representation of the robot's fault structure and modularized detection and recovery actions applicable to common robot components [24]. The advantage of our approach is that, through the expert system, different methods for fault detection and tolerance, such as those described in the works listed above, may be conveniently and efficiently integrated, allowing the robot system to benefit from the best available approaches.

†Author for correspondence.

In this paper, we discuss how the expert system manipulates fault trees [25,26] in order to dynamically respond to detected failures in the robot system. In fault trees, the possible failure events for the system are connected by basic logic symbols to produce a flow chart of failures [26]. Some failure analysis programs use digraphs [27,28] to represent the failure interconnections. Digraphs are, however, easily transformed into fault trees for which there exist a wide variety of research efforts [4,23,25,29].

In the following section, a new robot fault tolerance framework is described focusing on the expert system and the fault tree database within the expert system. A discussion of quantitative failure probabilities maintained by the system follows in Section 3. The novel use of fault trees as a dynamic database for reconfiguration and repair, together with examples, is detailed in Section 4. Issues of fault tolerant user interfacing and a discussion of the resulting critic shell follow in Section 5. Section 6 contains conclusions and discussion.

2. FAULT TOLERANCE FRAMEWORK

In order to separate the various functions of the overall robotic system and to better monitor the interactions of the subsystems, we use models from the design of computer operating systems. Figure 1 shows the high level interactions among the various layers of the control architecture, incorporating an expert system framework.

2.1. Operating system architecture

Operating system software [30] allows for the scheduling and control of tasks desired by the user. The operating system presents a well-defined interface to the user and spares the user from having to know the particular details of the current configuration and underlying implementation. The user communicates with a shell which is wrapped around a kernel. This analogy implies that the shell protects the kernel and analyzes and translates random user commands. In Fig. 1, the user communicates commands to the critic/protocol conformance shell which validates the user

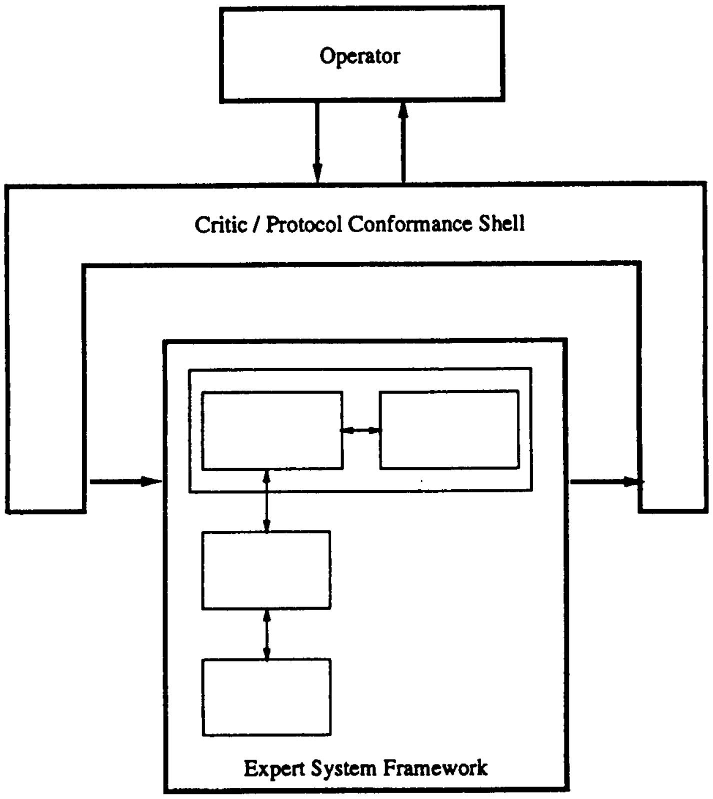


Fig. 1. Fault tolerant robotic operating system.

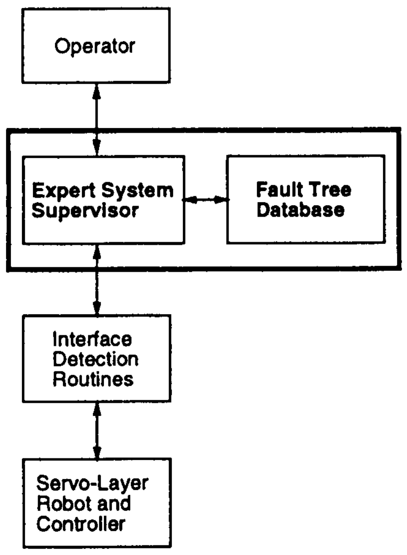

Operator
Expert System Supervisor
Fault Tree Database
Interface Detection Routines
Servo-Layer Robot and Controller

Fig. 2. Expert system supervisor within fault tolerance framework.

commands and passes low-level information to the expert system framework shown in greater detail in Fig. 2. As part of this standardization, the user and operating system agree on a language with a set of recognized commands. The commands are further defined in a specific protocol or standard. The designers of the system attempt to guarantee that the particular implementation will conform to the agreed upon protocol. Recently, there has been some work at Sandia National Labs to develop a robot-independent programming language [31]. In the following sections, we describe the fault tolerance framework and its capabilities in more detail, concentrating on the expert system and its fault tree database. The overlying critic shell is further detailed in Section 5.

2.2. Expert system kernel framework

The expert system kernel is initially divided into three layers [14,21]; a continuous servo layer, an interface monitor layer, and a discrete supervisor layer. The servo layer consists of the normal robot controller and the robot. The interface layer contains the motion planner and provides fault detection and some basic fault tolerance for the system. The supervisor layer provides higher level fault tolerance and general action commands for the robot.

Figure 2 highlights the position of the expert system as supervisor in the fault tolerance intelligent control framework discussed in [14,21]. The expert system prunes failed components or branches from a database of fault trees as failures are reported from the detection routines in the interface layer. The system immediately notifies the operator of failures in the robot during on-line operations so the operator may monitor the fault tolerance capabilities of the system and suggest alternative recovery actions. The expert system also suggests recovery actions based on the structure of the robot fault trees. The nodes in the tree then have an associated detection or recovery action module [21,32] depending on the type of component or event. By maneuvering around the trees, the expert system quickly performs fault tolerant recovery actions for a specific robot structure as a sequence of these smaller, generic actions. The user modifies the existing robot design or switches in another robot structure by manipulating the facts defining the fault trees in the database. Common components and systems are described in easily integrated libraries.

The expert system also recomputes the probability of failure of the robot system as failures occur. The probabilities are used by the operator to monitor the degradation of the system and to note when critical components have failed (see Section 3). A robot system responds quickly to internal failures, endangering the robot and its working environment. A user may not have time to respond to the failure, especially in systems with long communication delays such as in space-based robots.

The robot must take the initiative to cope with failures immediately until the user can verify the recovery action, suggest alternatives, or repair the component. The prioritization capabilities of the expert system are therefore utilized so that lower priority probability computations are treated as background processes which occur after more pressing recovery or tolerance actions have been completed.

We have implemented the fault tree database and supervisor in CLIPS ('C' Language Integrated Production System) [33], a NASA-developed expert system software package. This system provides an intelligent and flexible framework which easily accommodates fault detection and fault tolerance routines for many structurally diverse robots. Fault tree traversing schemes provide rules for the expert system supervisor layer. The expert system organizes the detection and tolerance algorithms into modular procedures and can be easily embedded within other programs to allow for future expansions.

2.3. Fault tree database

Fault trees contain details of the manipulator hardware, the control computer hardware and software. An operator initializes the fault tree database with facts defining the structure of a predetermined fault tree for the robot. We have developed detailed fault trees for our eight joint research robot at Rice [26], and for a manipulator of the PUMA geometry. For this paper, we focus on a simple four-link planar robot (Fig. 3) with dual sensors (for example, encoder/tachometer, double encoder, etc.) and a direct drive motor per joint. This structure provides a simple example of analytical [21,32] and kinematic redundancy with which the expert system can detect and tolerate component failures.

In the fault tree (Fig. 4) for the robot, we have concentrated on the critical functional components; the manipulator sensors and actuators. The failure events have been given a basic letter encoding to simplify the database facts. Event **A** is the failure of the entire robot, the top event. Failure events **B**, **C**, **D**, and **E** represent failures of the joints of the robot. Note that the suppressed trees under events **D** and **E** look much the same as the subtrees under **B** and **C**. The basic component failures **u** through **z** represent failures of sensors (**v**, **w**, **x**, **z**) or motors (**u**, **y**), the critical components in the joint subsystems. The OR-gate connecting **u** and **F** means that either event **u** or **F** occurring will cause **B** to occur, where the AND-gate connecting **v** and **w** indicates that both **v** and **w** must occur to invoke **F**.

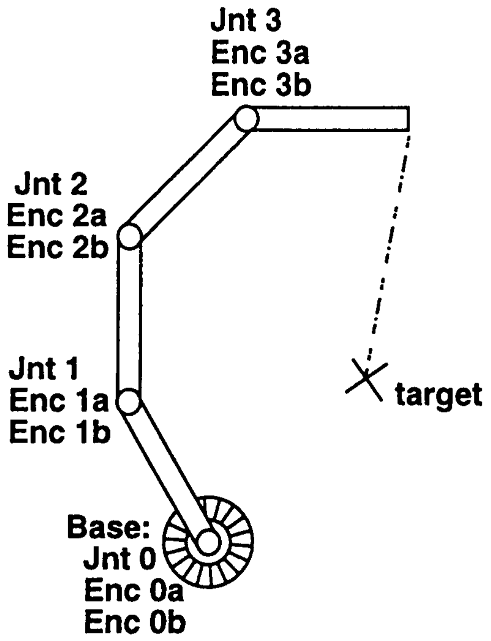
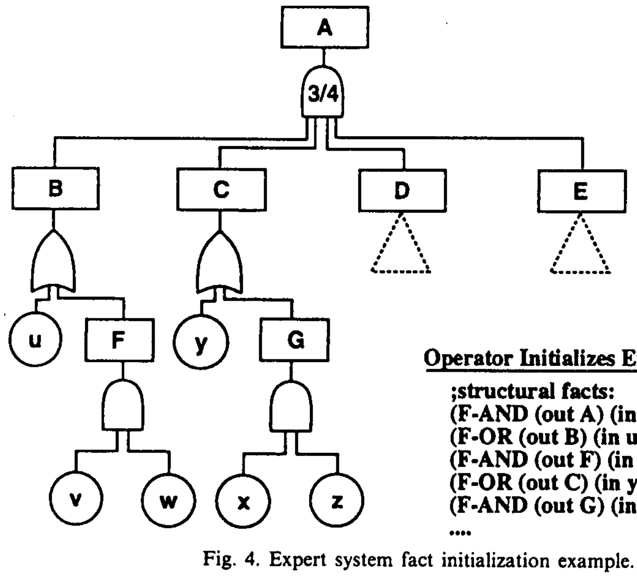


Fig. 3. Four-link robot example.



Operator Initializes Expert System with:

;structural facts:
(F-AND (out A) (in B C D E) (fail-with 3))
(F-OR (out B) (in u F))
(F-AND (out F) (in v w))
(F-OR (out C) (in y G))
(F-AND (out G) (in x z))
....

Fig. 4. Expert system fact initialization example.

While the robot is operational, the expert system supervisor maintains the database model of the trees, pruning branches as failures occur based on the rules for failure propagation through AND- and OR-gates. As failures spawn additional failures due to the robot structure, subsequent failures are automatically asserted by the expert system propagation rules, thus initiating a recursive propagation of failure events up through the trees.

When a sensor or motor failure is detected by the interface layer [21], a failure fact is asserted in the expert system. This fact consolidates with all structural facts containing as input the failed component to activate a specific failure propagation rule. If the component is an input to an OR-gate, the failure propagates through the gate and fails the output, thus pruning the OR-gate from the tree by modifying the gate status to 'failed'.

Because AND-gates in a tree represent some form of functional redundancy in the robot [21], AND-gate facts have an associated 'fail-with' field (see Fig. 4) which informs the system of how many input failures will fail the AND-gate. If an input to an AND-gate fails and the 'fail-with' field of the gate is greater than one, then this level still has redundancy and only the failed input is pruned from the tree. The gate now fails with one less failure than before and has one less input. When the AND-gate fact reveals that the gate will fail with one failure, the gate is essentially an OR-gate and a failure of any remaining input will fail the output of the gate.

Facts associated with a pruned component or branch are not actually removed from the database but are simply modified to prevent certain rule activations. This allows the original tree to be easily rebuilt by the expert system when components are repaired/replaced. Asserting the 'failed' status only upwards in the tree from a failed component allows the expert system to continue monitoring the siblings of the faulty component (i.e. other inputs to the gate). For example, when component **u** fails in the tree of Fig. 4, the failure propagates through the OR-gate and fails **B** as well. Suppose that while **u** is down, component **v** fails. The expert system is able to handle the new failure appropriately for the AND-gate beneath **F**, which is still considered alive although it no longer contributes to system failure because node **B** has already failed. If **u** is now repaired and reinstated into the database, the failure of **v** is still known to the expert system. If the entire subtree under **B** had been 'pruned' as well when **u** originally failed, the failure of **v** would not have been correctly recorded at its onset.

The operator has the option to signal an abort at any time if it is deemed that the assigned task has been compromised. The expert system will use suggestions made by the operator or information learned by traversing the database to develop new tolerance strategies based on the existing structure of the robot. Alternate sensors or unburdened external sensors (such as cameras) of which the interface layer is originally unaware, can be switched into the detection set to monitor a joint for failures when the known or basic set of sensors has failed. Increasing the redundancy of the motors or sensors as in [8] and [10] would provide a wider range of survivor choices for the system.

2.4. Future enhancements

The expert system tasks can be gracefully expanded or updated to draw in other work in the rapidly growing field of robotic fault tolerance [6,8,10,13,18]. The expert system can utilize programs, such as those described in [6,18], to monitor the reachable workspace of the robot throughout its operation. When failures occur and a joint is shut down, the workspace shrinks due to the immobility of the failed joint. The expert system will compute the changes to the reachable workspace [18] and decide if the target position is still reachable by the robot. If the robot can no longer reach its assigned goal, the supervisor signals an abort and waits for further instructions from the operator.

The supervisor provides the operator with information on possible causes of failures by traversing the appropriate subtrees in the database to accumulate a list of components which may have caused a given failure. An operator can then use the list to direct and speed up future inspections and repairs. When pruning the trees of failed components, the system also searches for indications that a component is likely to fail because it is linked either structurally or functionally to a previously failed component. For example, when one half of a dual motor fails, the surviving half picks up the burden of the failed half's workload [8,10], increasing its chances of failing. Also, if a certain type of sensor has failed due to operating conditions, other sensors of that type may also fail in the near future. Advanced detection routines use such warnings to be more alert to the possibility of failures in these components while monitoring the system.

3. PROBABILITY OF FAILURE

The expert system maintains the probability of failure for each node within the tree. The operator initializes only the basic components (leaves) in the tree with appropriate probability facts. The expert system then initializes the probabilities for inner nodes of the tree by combining the basic component probabilities through the gates in the tree structure.

As detailed in [4,25], the probability of a component failure can be calculated from a failure rate for the component [29]. Given a constant failure rate λ and using the exponential distribution, the probability of failure at time t is [4]:

$$p(t) = 1 - e^{-\lambda t},$$

the reliability of the component in the system is given by

$$R(t) = 1 - p(t) = e^{-\lambda t},$$

and the mean time to failure (MTTF) is given as

$$\text{MTTF} = 1/\lambda.$$

If the failure rate is small, the probability of failure is often approximated as λt [25]. The expert system models component decay by using time-dependent probabilities [25]. A small update routine monitors the system time and modifies the basic probability facts during the life of the robot.

Various methods can be used to determine the failure rate λ. For example, in [34], the average failure rate λ_m for a D.C. motor is estimated as

$$\lambda_m = [(t^2/\alpha_B^3) + (1/\alpha_W)]$$

failures per 10^6 h, where t is the operating time period for which λ_m is the average failure rate, α_B is the bearing characteristic life, and α_W is the winding characteristic life of the device. Both α_B and α_W depend on the ambient temperature for the device, with expressions given in [34]. For an ambient temperature of 20°C, an operating period of 100 h, the data in [34] gives a failure rate of 6.3×10^{-7} failures per hour.

Also in [34], the average failure rate λ_r for a resolver is given as

$$\lambda_r = \lambda_b \pi_S \pi_N \pi_E$$

failures per 10^6 h, where λ_b is the base failure rate (exponentially related to ambient temperature), π_S is a factor related to the device size, π_N is related to the number of brushes, and π_E is an environmental factor. For a small resolver with four brushes and the same ambient temperature as the motor above in a (possibly mobile) ground-based environment, the failure rate λ_r is found from data in [34] to be 1.6×10^{-6} failures per hour.

In the following, we assume that failure rates have been found, and work with the failure probabilities in the system. Within the fault trees, these failure probabilities are combined through the logic gates using simple multiplication and addition [25]. The probability of failure for the output event of an AND-gate is the product of all the input probabilities and a conservative estimate of the output event probability for an OR-gate is the sum of the input probabilities.

Figure 5 shows two stages of the process for the subtree under component **B** in Fig. 4. In Fig. 5(a), a failure has occurred in motor **u** and the failure propagates to the top level following the rules described earlier. Failed components and events are shown with a grey background and failed gates and paths are shown with dashed lines. In Fig. 5(b), sensor **v** has failed (and **u** has not failed) and the modified probabilities propagate through the tree as shown. When a component fails, the expert system initiates the changes in the system probabilities that are affected by the failed node. The failed component is matched with all structure facts containing the component as input and new facts are asserted which detail the type of gate, the output event, and the inputs. These new facts activate probability-propagation rules which recompute the probability of the output event by recursively multiplying or adding (depending on the gate type) [25] the probabilities of the inputs.

Currently, the branch-in factor (number of inputs) of the gates in the trees is relatively small so that each recomputation step is not too expensive time-wise. An alternative method if the branch-in factor is high for a robot's fault trees is to divide (for AND-gates) or subtract (for OR-gates) the old value of the probability for the modified input and then multiply or add in the new value. This method limits the computation at each level to a single divide/multiply or subtract/add combination but requires additional memory space to save the old probabilities for each node in the tree. There is thus a tradeoff in time vs space which must be considered.

A quantitative analysis provides a measure of the overall chance of a complete failure for each robot system. The structure provided by the fault trees organizes the probabilities appropriately for the robot system and provides a simple map of how the events relate to each other. Using the trees, robots of a significantly different origin and structure are easily compared for fault tolerance and survivability. Static analysis of failure probabilities and their propagation through the fault trees allows the robot designer to focus on the components which are more likely to fail in the robot and which are thus more central to the monitoring software and fault detection routines.

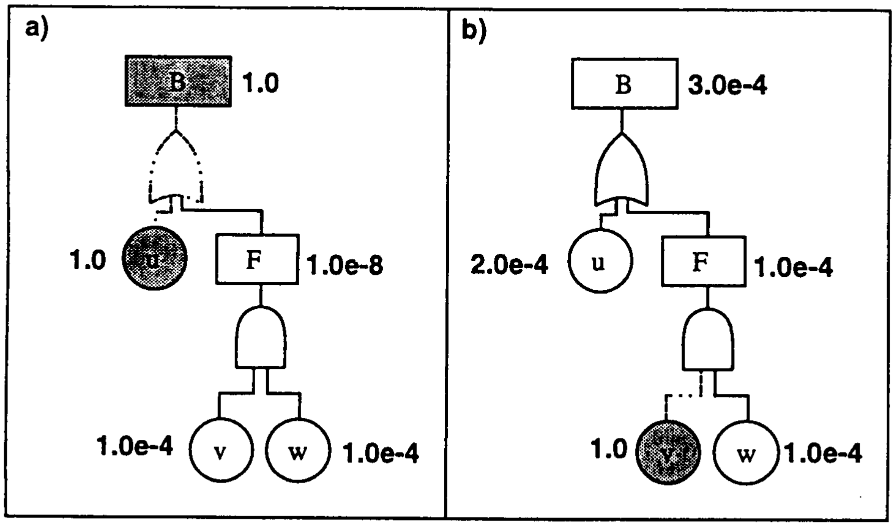


Fig. 5. Example of probability propagation in fault trees.

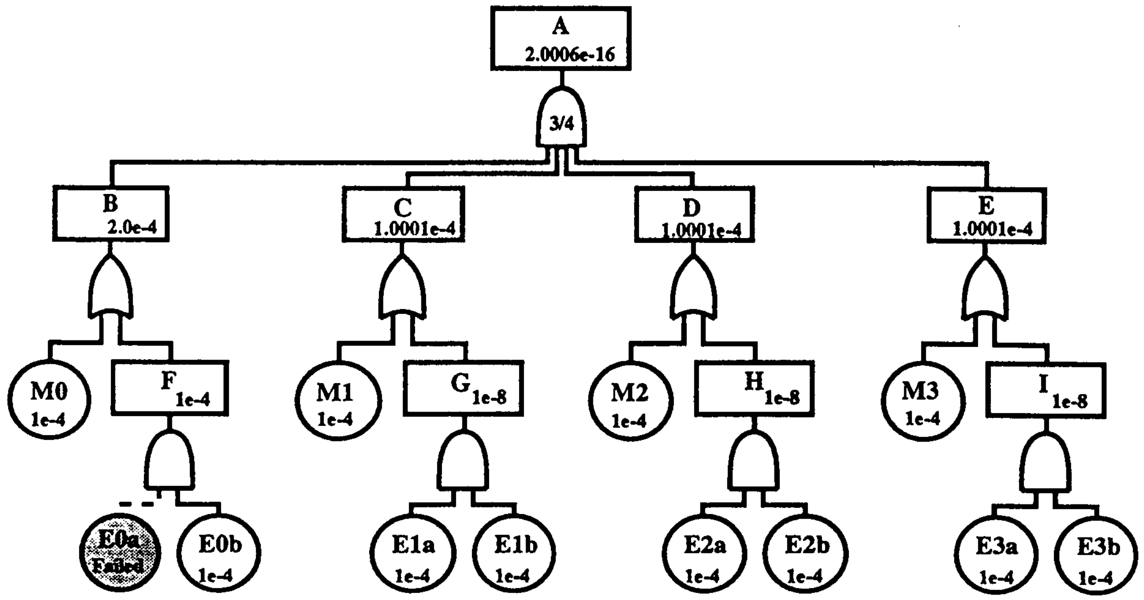
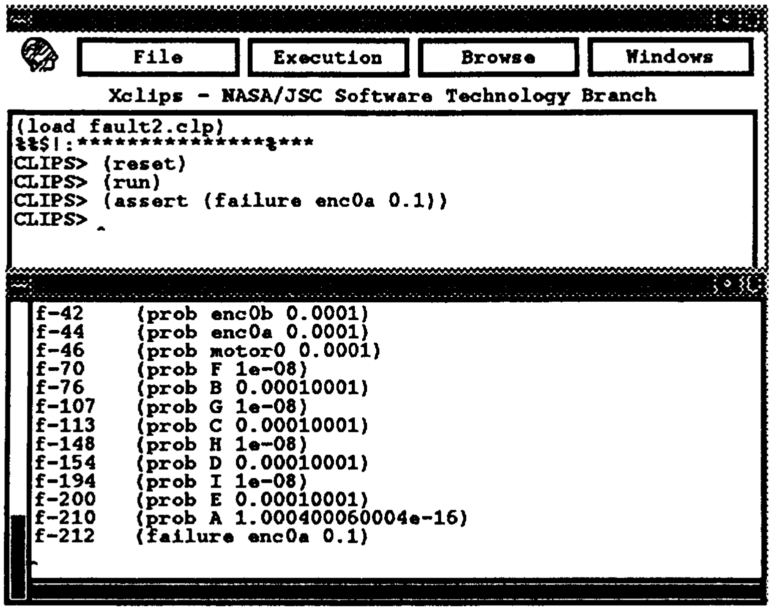


Fig. 6. Example failure of an encoder at joint_0.

By dynamically modifying the probabilities as failures occur, the expert system alerts the robot operator of any large changes in failure probabilities (several orders of magnitude) which may indicate a loss of redundancy or a loss of a component vital for the current task. The dynamic probability analysis will also be useful in directing which joint or component to use for a given task by promoting the use of those systems which are less likely to fail. The robot would then mimic optimal configurations [6,7] so as not to put undue stress on systems which are injured or 'sore'.

4. DYNAMIC FAULT TREES AND FAULT TOLERANCE

We use fault trees in a dynamic mode to see the effects of component or sub-system failure as the system is operating. The expert system is a key component here, providing the robot system with a structured, easily modifiable knowledge base to cope with failures on-line and also providing

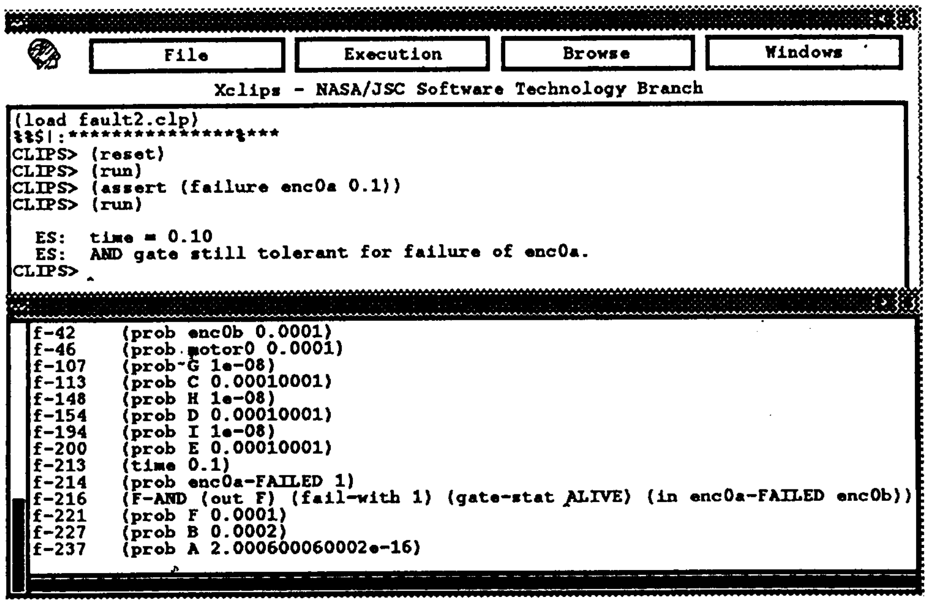


Xclips - NASA/JSC Software Technology Branch

```
(load fault2.clp)
%%$!:*****************%***
CLIPS> (reset)
CLIPS> (run)
CLIPS> (assert (failure enc0a 0.1))
CLIPS>
```

```
f-42     (prob enc0b 0.0001)
f-44     (prob enc0a 0.0001)
f-46     (prob motor0 0.0001)
f-70     (prob F 1e-08)
f-76     (prob B 0.00010001)
f-107    (prob G 1e-08)
f-113    (prob C 0.00010001)
f-148    (prob H 1e-08)
f-154    (prob D 0.00010001)
f-194    (prob I 1e-08)
f-200    (prob E 0.00010001)
f-210    (prob A 1.000400060004e-16)
f-212    (failure enc0a 0.1)
```

Fig. 7. CLIPS example: initial status.



```
(load fault2.clp)
%%$|:*****************%***
CLIPS> (reset)
CLIPS> (run)
CLIPS> (assert (failure enc0a 0.1))
CLIPS> (run)

  ES:  time = 0.10
  ES:  AND gate still tolerant for failure of enc0a.
CLIPS>
```

```
f-42     (prob enc0b 0.0001)
f-46     (prob motor0 0.0001)
f-107    (prob G 1e-08)
f-113    (prob C 0.00010001)
f-148    (prob H 1e-08)
f-154    (prob D 0.00010001)
f-194    (prob I 1e-08)
f-200    (prob E 0.00010001)
f-213    (time 0.1)
f-214    (prob enc0a-FAILED 1)
f-216    (F-AND (out F) (fail-with 1) (gate-stat ALIVE) (in enc0a-FAILED enc0b))
f-221    (prob F 0.0001)
f-227    (prob B 0.0002)
f-237    (prob A 2.000600060002e-16)
```

Fig. 8. CLIPS example: failure asserted.

a link with the system operator. Our expert system is configured with fault trees for both the four-link and PUMA arms. For the purposes of illustration, we consider the simpler case of the four-link planar robot example here (see Fig. 3).

With all components functioning, the probability of failure of the robot, represented by the top node A, is 1.0004×10^{-16}. Figure 6 shows the fault tree annotated with the probabilities which the expert system tracks for the robot. In this example, an encoder at joint_0 has failed. The interface detection layer routines detect the failure and automatically reconfigure to rely solely on the surviving sensor. The interface alerts the supervisor that the failure has occurred. As the interface was able to tolerate the failure, the expert system simply propagates the failure and the changing probabilities through the tree. If the interface had been unaware of the alternate sensor or if there were no obvious alternate sensor, the expert system would search the database for other possibly external options [24]. Because of the sensor redundancy at the joint, the failure probability of the top node A increases only slightly after the failure (a factor of two as opposed to orders of magnitude).

Figures 7–9 show the progression of this example using the X-windows interface for CLIPS. The upper window in each figure is the general interactive window through which the user can direct and monitor the expert system. The lower window lists the facts currently in the database. Not all of the facts are listed in these figures so as to highlight the changes as the robot status is updated.

Figure 7 shows the initial status of the robot before the failure has occurred. The robot's tree structure is loaded in from a file and initialized with the first reset and run commands. The 'prob' facts in the fact window show the initial probabilities for the first joint branch and all of the upper nodes in the system. Note that the probability of the top node A is the last probability fact in the list (fact f-210 in the figure). The fact containing the direction to fail encoder 0a at time 0.1 is also shown.

In Fig. 8, time 0.1 is reached causing the instigation of the failure. The expert system finds the tree structure (fact f-216) containing enc0a and prunes the encoder by making it 'enc0a-FAILED'. Since the branch containing enc0a is an AND-gate and the remaining sensor is still alive, the failure does not propagate any further up the tree. Several probabilities, however, do change up the tree and are shown at the bottom of the fact list in Fig. 8. The failure probability at the top level is now 2.0006×10^{-16} (fact f-237). In the top window of Fig. 8, the expert system alerts the operator of the failure, noting the time and the fact that the robot was able to tolerate the failure.

The user might now wish to explore alternative replacement strategies for the failed encoder at joint_0 by considering replacing it with a more reliable unit with failure probability $p = 1.0 \times 10^{-6}$.

The effect of this replacement strategy is shown in Fig. 9. The branch containing the encoder is modified again to reflect the repair and the fact that the branch now has increased redundancy (see the 'fail with' field in fact f-242 vs fact f-216 in Fig. 8). The probabilities are again changed and the operator notes that, while the replacement encoder is 100 times more reliable, the relative improvement in overall system reliability is quite small ($A = 1.0003 \times 10^{-16}$ vs the original $A = 1.0004 \times 10^{-16}$). This is due to the redundant sensor configuration of the joint. The user might therefore opt not to replace the failed encoder with this possibly more expensive unit. In any case, the expert system is able to deal with both off-line design analysis and real repairs to the on-line system.

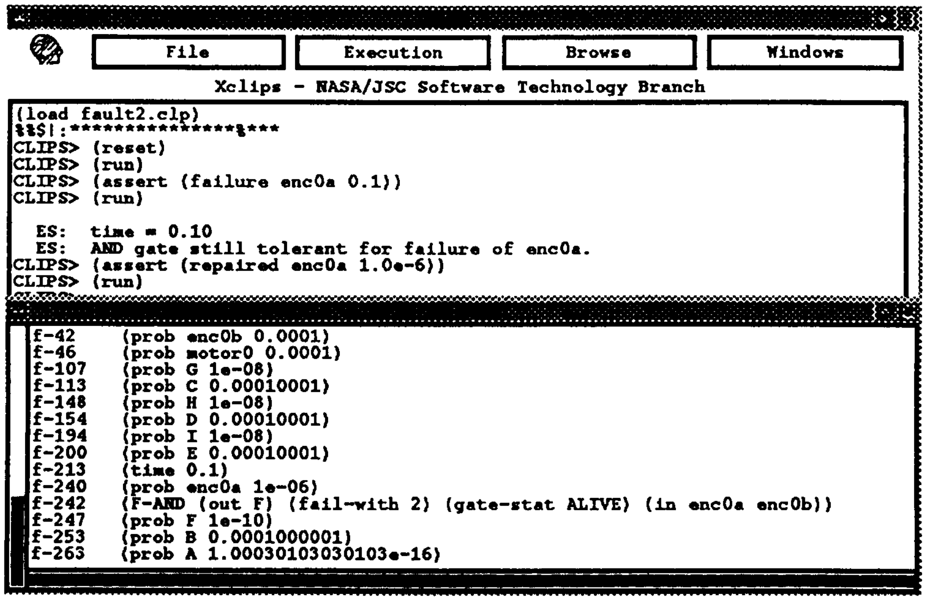
5. CRITIC/PROTOCOL CONFORMANCE SHELL

In this section, we develop and formalize the protection capabilities of the shell for our example robot application. Computer operating systems provide various levels of user privilege and enforce the concept of a 'Super User' and a 'Normal User'. The shell understands that certain commands are potentially dangerous to the integrity of the system and are therefore privileged and can only be executed by the more knowledgeable 'Super User'. In our fault tolerant robot application, we develop a shell which contains a 'critic'. Following fault tolerance reconfiguration, this critic checks for conformance to the original plan provided by the user. The critic also checks for unavoidable obstacles and will halt the robot to protect it from imminent damage. Only the super user can override the critic and force it to continue.

The shell, shown in Fig. 1, is composed of two Finite State Machines, the User/Executive FSM and the Critic FSM. The User/Executive FSM handles the interaction with the user and initializes subordinate FSMs. The Critic FSM is responsible for providing protection and privilege services in our proposed intelligent control architecture [14].

5.1. User/Executive interaction FSM

The top level FSM describes the user interaction with the robot (Fig. 10). The robot waits in the executive idle state, EX_Idle, for input on the command queue from the operator in the form of the user plan. A robot independent command language, containing commands such as MOVE, APPROACH, OPEN_GRIPPER, CLOSE_GRIPPER, DEPART, MOVE_HOME, and STOP, as described in [31], would be recognized. The executive main loop takes a command from the queue and, when ready to run, enables the critic FSM. If the critic FSM reports successful completion,



```
(load fault2.clp)
%%$|:****************%***
CLIPS> (reset)
CLIPS> (run)
CLIPS> (assert (failure enc0a 0.1))
CLIPS> (run)

  ES:  time = 0.10
  ES:  AND gate still tolerant for failure of enc0a.
CLIPS> (assert (repaired enc0a 1.0e-6))
CLIPS> (run)
```

```
f-42    (prob enc0b 0.0001)
f-46    (prob motor0 0.0001)
f-107   (prob G 1e-08)
f-113   (prob C 0.00010001)
f-148   (prob H 1e-08)
f-154   (prob D 0.00010001)
f-194   (prob I 1e-08)
f-200   (prob E 0.00010001)
f-213   (time 0.1)
f-240   (prob enc0a 1e-06)
f-242   (F-AND (out F) (fail-with 2) (gate-stat ALIVE) (in enc0a enc0b))
f-247   (prob F 1e-10)
f-253   (prob B 0.0001000001)
f-263   (prob A 1.00030103030103e-16)
```

Fig. 9. CLIPS example: component repaired.

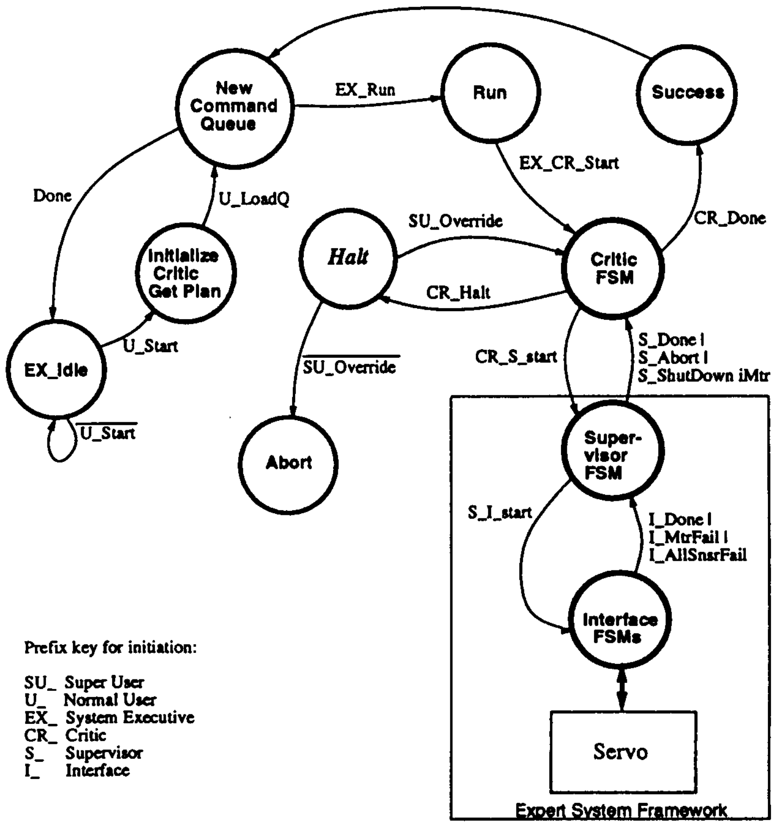


Fig. 10. Top level state diagram for full system showing User/Executive interaction.

then the next command is taken from the job queue until a STOP or done command appears in the queue. The executive then returns to EX_Idle.

When the executive starts the critic FSM with the EX_CR_Start signal, a number of actions occur at different levels to initiate control of the servo level robot. The hierarchy is displayed in Fig. 10. The state transition diagrams in Figs 10 and 11 show in greater detail the interaction among the subordinate finite state machines. Only the most important start, done, and failure signals are shown in Fig. 10.

The Supervisor and Interface FSMs and the servo level robot are discussed in detail in [21] and constitute the expert system kernel framework discussed in Section 2.2. Each sub-FSM has a lot of activity associated with it. Each Interface FSM (of which there is one per joint) monitors its joint for failures, performs basic fault tolerance of single and multiple component failures, and alerts the supervisor to changes in the robot status. The Supervisor (expert system) FSM performs more long term analyses of the system, searches for higher level or less obvious tolerance options, and maintains the tree database.

If a problem occurs in the robot which cannot be resolved with the expert system framework tolerance capabilities, the critic FSM will output a halt signal, CR_Halt. This protection mechanism will stop the robot and require user intervention. A 'Super User' may override the warning and cause the robot to resume a possibly dangerous task. A 'Normal User' may not override the critic and the task will abort. Additional levels of privilege could be added to such a system.

5.2. Critic FSM

The concept of a critic that will make judgments and recommendations to the user has been used in a number of automated systems. For example, systems exist for the analysis and criticism of text and VLSI circuit designs [35]. In Fig. 11, our robot fault tolerance critic is described as a finite state machine. The critic is initialized with the user plan by the executive FSM each time a new task is begun. The critic is responsible for the safety of the robot system and monitors the user's plan and the intelligent robot kernel.

After initialization, when the executive sends a start signal, EX_CR_Start, the critic moves from the ideal state, CR_Idle, and initializes the expert system supervisor FSM to carry out the next command in the user plan. Obstacle checking [12] is then performed by the critic until the supervisor indicates that the command is complete through S_Done or that a workspace reducing failure has occurred through S_ShutDowniMtr or the critic discovers an obstacle avoidance problem. If the obstacle checking routine reports an unsolvable problem to the critic, then the critic issues the CR_Halt signal and moves to the ERROR-HALT state. The override or abort options are as described in the above section on the user interface.

In response to the supervisor signal S_ShutDowniMtr, the critic initiates a process to recompute [18] the area reachable by the end effector with the robot in its current state using failure status information from the expert system database (see Section 2.4). If the robot can no longer reach its assigned goal, the critic signals an abort and alerts the user. The critic then waits in the ERROR-HALT state for direction from the user or super user.

The process of checking to see if the fault reconfigured robot can still complete its plan could be approached as a conformance testing problem [36]. The user plan was developed with a model of the robot which contained a specific number of sensors and joints. When faults cause joints to be lost, the reduced robot is viewed as a subset of the original robot. Test sequences can be developed to determine if the reduced robot conforms to the original robot specification with respect to the user's original plan.

6. CONCLUSIONS AND DISCUSSION

This paper introduces a new approach for integrating fault detection and fault tolerance in robotics. The operating system-inspired critic discussed above provides a buffer between robot fault

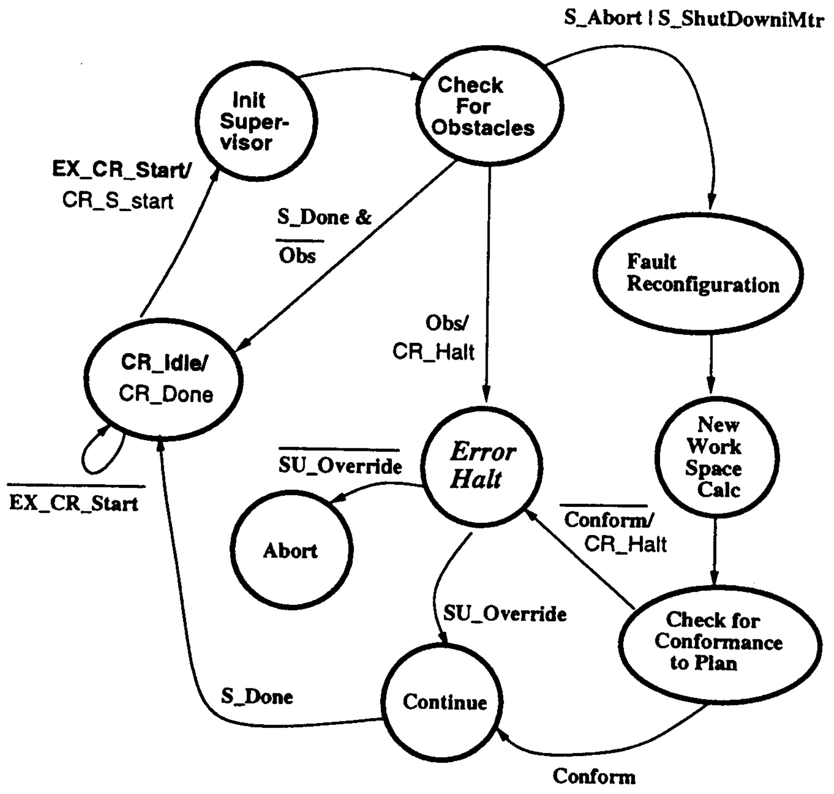


Fig. 11. State diagram for critic.

tolerant operations and the user by filtering user commands and ensuring that no damage is done to the robot or the environment unless it is verified. Through the use of an expert system, different strategies for failure mode analysis, fault detection, tolerance, reconfiguration, and repair can be linked into one framework. The key to the approach is the integration of fault trees into the expert system.

Fault trees are used statically to reveal the structural and functional redundancy of the system and to help develop the fault detection and tolerance algorithms for a specific robot system. When embedded in the expert system, the trees can be used to evaluate and re-design potential robots for fault tolerance at the design stage. The importance of adding redundant components, such as those discussed in [6–10], can be evaluated.

In addition, the expert system supervisor uses the fault trees dynamically as a database of run-time fault detection and recovery data which reveals functional replacements for faulty components and indicates possible causes or effects of failures. The fault trees within the database keep track of available fault tolerance built into the robot [6,7,9] (such as the redundant actuator modules of [8] and [10]). The expert system also maintains dynamic failure probabilities for the system as an indication of system weak spots.

A variety of fault detection routines (such as those suggested by [19–22], etc.) run in parallel and interact with the expert system at the interface level. Following detection of a failure, the interface level routines attempt to handle the failure using alternate components. If the interface level cannot find a way to tolerate the failure, or if an inappropriate command is given, the expert system invokes higher level routines for reconfiguration and recovery, such as in [6,7,16–18]. Since the expert system is based on the 'C' language, it is easily expandable to incorporate a variety of detection and tolerance schemes. It can also be easily integrated into an overall modular robot control architecture [11–14].

The critic and expert system framework described by this paper provides a novel implementation for a protected, flexible, intelligent, and modularly expandable fault tolerance framework for robotics. A version of the expert system package has been integrated with the IGRIP graphics environment and the Sandia robot control architecture, and is currently running at Sandia National Laboratories. We have also integrated the system with various fault detection algorithms in a simulation testbed at Rice, and, in future work, will move the code into a hardware testbed.

Acknowledgements—This work was supported in part by the National Science Foundation under grants DDM-9202639 and MSS-9024391 and by DOE Sandia National Laboratory Contract No. 18-4379A. Further support was provided by a Mitre Corporation Graduate Fellowship and NSF Graduate Fellowship RCD-9154692.

REFERENCES

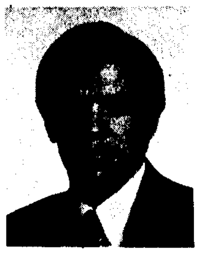
1. *Environmental Restoration and Waste Management Robotics Technology Development Program Robotics 5-Year Plan.* Vols 1-3. DOE/CE-0007T. Department of Energy, Washington, D.C. (1990).
2. A. Cohen and J. D. Erickson, Future uses of machine intelligence and robotics for the space station. *IEEE J. Robot. Automat.* **RA-1(3)**, 117–123 (1985).
3. N. M. Shehad and D. T. Wick, Simulation and control environment: an integrated environment for fault tolerant testing of space robots. In *Proceedings of Fourth International Symposium on Robotics and Manufacturing*, pp. 195–200. Sante Fe, N.M. (1992).
4. B. S. Dhillon, *Robot Reliability and Safety*. Springer, New York (1991).
5. J. T. Chladek, Fault tolerance for space based manipulator mechanisms and control systems. In *International Symposium on Measurement and Control in Robotics*, Vol. 1, pp. D3.3.1–D3.3.3. Houston, Tex. (1990).
6. A. A. Maciejewski, Fault tolerant properties of kinematically redundant manipulators. In 1990 *IEEE Conference on Robotics and Automation*, pp. 638–642. Cincinnati, Ohio (1990).
7. A. A. Maciejewski, The design and control of fault tolerant robots for use in hazardous or remote environments. In *Proceedings of the Fourth ANS Topical Meeting on Robotics and Remote Systems*, pp. 633–642. Albuquerque, N.M. (1991).
8. D. Tesar, D. Sreevijayan and C. Price, Four-level fault tolerance in manipulator design for space operations. In *NASA First International Symposium on Measurement and Control in Robotics*, Vol. 3, p. J3.2.1. Houston, Tex. (1990).
9. G. Toye, *Management of non-homogeneous functional modular redundancy for fault tolerant programmable electro-mechanical systems.* Ph.D. Thesis, Stanford University (1989).
10. E. Wu, M. Diftler, J. Hwang and J. Chladek, A fault tolerant joint drive systems for the space shuttle remote manipulator system. In 1991 *IEEE International Conference on Robotics and Automation*, pp. 2504–2509. Sacramento, Calif. (1991).
11. P. T. Boissiere and R. W. Harrigan, Telerobotic operation of conventional robot manipulators. In 1988 *IEEE International Conference on Robotics and Automation*, pp. 576–583. Philadelphia, Penn. (1988).

12. P. T. Boissiere and R. W. Harrigan, An alternative control structure for telerobotics. In *Proceedings of the NASA Conference on Space Telerobotics*, pp. 141–150. Pasadena, Calif. (1989).
13. D. B. Stewart, R. A. Volpe and P. K. Khosla, Integration of real-time software modules for reconfigurable sensor-based control systems. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 325–332. Raleigh, N.C. (1992).
14. I. D. Walker and J. R. Cavallaro, Dynamic fault reconfigurable intelligent control architectures for robotics. In *Proceedings 1993 Fifth American Nuclear Society Meeting on Robotics and Remote Handling*, pp. 305–312. Knoxville, Tenn. (1993).
15. D. L. Hamilton, J. K. Bennett and I. D. Walker, Parallel fault-tolerant robot control. In *1992 SPIE Conference on Cooperative Intelligent Robotics in Space III*, pp. 251–261. Boston, Mass (1992).
16. T. Wikman and W. Newman, Reflex control for robot system preservation and reliability. In *Proceedings of Fourth International Symposium on Robotics and Manufacturing*, pp. 979–986. Sante Fe, N.M. (1992).
17. B. R. Donald. *Error Detection and Recovery in Robotics*. Springer, New York (1989).
18. A. K. Pradeep, P. J. Yoder, R. Mukundan and R. J. Schilling, Crippled motion in robots. *IEEE Trans. Aerosp. Electron. Syst.* **24(1),** 2–13 (1988).
19. B. Freyermuth, An approach to model based fault diagnosis of industrial robots. In *1991 IEEE International Conference on Robotics and Automation*, pp. 1350–1356. Sacramento, Calif. (1991).
20. M. L. Visinsky, J. R. Cavallaro and I. D. Walker, Dynamic sensor-based fault detection for robots. In *1993 SPIE Conference on Telemanipulator Technology and Space Robotics*, Vol. 2057, pp. 385–396. Boston, Mass. (1993).
21. M. L. Visinsky, I. D. Walker and J. R. Cavallaro, Layered dynamic fault detection and tolerance for robots. In *1993 IEEE International Conference on Robotics and Automation*, pp. 180–187. Atlanta, Ga (1993).
22. J. Wünnenberg and P. M. Frank, Dynamic model based incipient fault detection concept for robots. In *Proceedings of the IFAC 11th Triennial World Congress*, pp. 61–66. Tallinn, Estonia (1990).
23. M. L. Visinksy, I. D. Walker and J. R. Cavallaro, Fault detection and fault tolerance in robotics. In *Proceedings of the NASA Space Operations, Applications, Research Symposium*, pp. 262–271. Houston, Tex. (1991).
24. M. L. Visinsky, J. R. Cavallaro and I. D. Walker, Expert system framework of fault detection and fault tolerance for robots. In *Robotics and Manufacturing: Recent Trends in Research, Education, and Applications. Proceedings of the Fourth International Symposium on Robotics and Manufacturing*, pp. 793–800. Santa Fe, N.M. ASME Press, New York (1992).
25. W. E. Vesley, F. F. Goldberg, N. H. Roberts and D. F. Haasi, Fault tree handbook. NUREG 0492, Systems and Reliability Research Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, D.C. (1981).
26. M. L. Visinsky, I. D. Walker and J. R. Cavallaro, Chapter 3, robotic fault tolerance: algorithms and architectures. In *Robotics and Remote Systems for Hazardous Environments*, Vol. 1, pp. 53–73. Prentice–Hall, N.J. (1993).
27. V. V. Dixit, EDNA: expert fault digraph analysis using CLIPS. In *Proceedings of the First CLIPS Conference*, pp. 118–124. NASA Conference Publication 10049. Houston, Tex. (1990).
28. R. W. Stevenson and R. W. McNenny, Fault isolation in space station freedom systems from the space station control center. In *NASA Joint Applications in Instrumentation Process and Computer Control (JAIPCC) 1992 Symposium: Technologies for New Exploration.* Clear Lake, Tex. (1992).
29. V. H. Guthrie and D. K. Whittle, RAM analysis software for optimization of servomanipulator designs. DOE Small Business Innovative Research (SBIR) Program Report JBFA-101-89, JBF Assoc. Inc., Knoxville, Tenn. Performed for Oak Ridge National Laboratory (1989).
30. J. Peterson and A. Silberschatz, *Operating System Concepts*. Addison-Wesley, Reading, Mass. (1983).
31. P. J. Eicker, D. J. Miller and D. R. Strip, Intelligent systems and technologies for manufacturing. *AT&T Tech. J.* 10–22 (1991).
32. E. Y. Chow and A. S. Willsky, Analytical redundancy and the design of robust failure detection systems. *IEEE Trans. Automat. Contr.* **AC-29(7)**, 603–614 (1984).
33. J. Giarratano and G. Riley, *Expert Systems: Principles and Programming*. PWS-Kent, Boston, Mass. (1989).
34. MIL-HDBK-217F, Reliability prediction of electronic equipment. Technical Report, DOD, Rome Laboratory, Griffiss AFB, N.Y. (1990).
35. B. G. Silverman, Survey of expert critiquing systems: practical and theoretical frontiers. *Commun. ACM* **35(4),** 106–127 (1992).
36. M. Ü. Uyar, A. Lapone and K. K. Sabnani, Algorithmic verification of ISDN network layer protocol. *AT&T Tech. J.* 17–31 (1990).

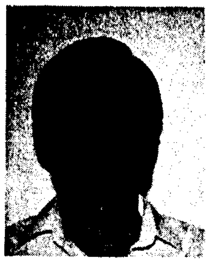
AUTHORS' BIOGRAPHIES

Monica L. Visinsky—Miss Visinsky was born in Houston, Texas on 25 July, 1968. She received the B.S. degree in electrical engineering and the B.A. degree in computer science from Rice University, Houston, Tex. in 1990. She received the M.S. degree, again in electrical engineering, in 1992 from Rice University, Houston, Tex, where she is currently pursuing the

Ph.D. degree. Her research interests include fault tolerance in computer and robotic systems, redundant robotic manipulators, artificial intelligence, and intelligent system design and implementation. Miss Visinsky received the MITRE Graduate Fellowship in 1991 and the 3 yr NSF Graduate Fellowship starting in 1991. She is a member of the IEEE, Tau Beta Pi, and Eta Kappa Nu.



Joseph R. Cavallaro—Dr Cavallaro was born in Philadelphia, Pennsylvania on 22 August, 1959. He received the B.S. degree from the University of Pennsylvania, Philadelphia, Pa, in 1981, the M.S. degree from Princeton University, Princeton, N.J. in 1982, and the Ph.D. degree from Cornell University, Ithaca, N.Y. in 1988, all in electrical engineering. From 1981 to 1983, he was with AT&T Bell Laboratories, Holmdel, N.J. In 1988 he joined the faculty of Rice University, Houston, Tex., where he is an Assistant Professor of Electrical and Computer Engineering. His research interests include computer arithmetic, fault tolerance, VLSI design and microlithography, and VLSI architectures and algorithms for parallel processing and robotics. Dr Cavallaro is a recipient of the NSF Research Initiation Award 1989–1992 and the IBM Graduate Fellowship 1987–1988, and is a member of IEEE, Tau Beta Pi and Eta Kappa Nu.



Ian D. Walker—Dr Walker was born in Epping, England, on 20 August, 1962. He received the B.Sc. degree in Mathematics from the University of Hull, England, in 1983. He received the M.S. degree in 1985, and the Ph.D. in 1989, both in Electrical Engineering, from the University of Texas at Austin. Since 1989 he has been an Assistant Professor in the Department of Electrical and Computer Engineering at Rice University, in Houston, Tex. His research interests are in the areas of robotics and control, particularly robotic hands and grasping; fault tolerant robot systems; and kinematically redundant robots. Dr Walker is a member of IEEE, Beta Alpha Phi, Eta Kappa Nu, Phi Kappa Phi, and Tau Beta Pi.