A Dynamic Fault Tolerance Framework for Remote Robots

Monica L. Visinsky, *Member, IEEE*, Joseph R. Cavallaro, *Member, IEEE*, and Ian D. Walker, *Member, IEEE*

**Reprinted from
IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION
Vol. 11, No. 4, August 1995**

A Dynamic Fault Tolerance Framework for Remote Robots

Monica L. Visinsky, *Member, IEEE*, Joseph R. Cavallaro, *Member, IEEE*, and Ian D. Walker, *Member, IEEE*

Abstract—Fault tolerance is increasingly important for robots, especially those in remote or hazardous environments. Robots need the ability to effectively detect and tolerate internal failures in order to continue performing their tasks without the need for immediate human intervention. This paper presents a layered fault tolerance framework containing new fault detection and tolerance schemes. The framework is divided into servo, interface, and supervisor layers. The servo layer is the continuous robot system and its normal controller. The interface layer monitors the servo layer for sensor or motor failures using analytical redundancy based fault detection tests. A newly developed algorithm generates the dynamic thresholds necessary to adapt the detection tests to the modeling inaccuracies present in robotic control. Depending on the initial conditions, the interface layer can provide some sensor fault tolerance automatically without direction from the supervisor. If the interface runs out of alternatives, the discrete event supervisor searches for remaining tolerance options and initiates the appropriate action based on the current robot structure indicated by the fault tree database. The layers form a hierarchy of fault tolerance which provide different levels of detection and tolerance capabilities for structurally diverse robots.

I. INTRODUCTION

ROBOTS are being more frequently utilized in inaccessible or hazardous environments to alleviate some of the risk involved in sending humans to perform the tasks and to decrease the time and expense. However, the dangerous, long distance nature of these environs also makes it difficult, if not impossible, to send humans to repair a faulty robot. Communication delays between the robot and operator may be large enough to allow potentially tolerable faults to blossom into system-wide malfunctions forcing the mission to abort. Typically, human operators are integral in monitoring such system for problems [1], but they are not completely reliable in detecting failures [7]. The robot needs to independently detect and isolate internal failures and utilize the remaining functional capabilities to automatically overcome the limitations imposed by the failures. This type of autonomous fault tolerance is also useful for industrial robots in that it decreases down-time by tolerating failures (thus increasing the robot's life-span), identifies faulty components or subsystems

to speed up the repair process, and prevents the robot from damaging the products being manufactured. Providing robots with autonomous fault detection and tolerance lengthens their working life without requiring immediate human intervention for every failure situation [40].

In previous work, we have designed generalized robot fault tolerance algorithms which utilize the advantages of whatever structure exists [35], [36], [39], [41], [42] making them appropriate for many applications, even those with limited or specialized robots. The fault detection algorithms monitor the system for failures by comparing measured system outputs with the corresponding expected output derived from a model of the system behavior. Analytical redundancy methods were used to develop the original detection tests for robotics and are used in this paper to extend this derivation. Analytical redundancy utilizes the available functional redundancy in the robot model instead of relying solely on hardware redundancy which increases the weight and cost of the robot. We note, however, that hardware redundancy does provide additional survival choices and is a good design option, if possible, for new robots. To mask out the modeling inaccuracies in the system which bias the tests slightly, this paper also presents a new, efficient algorithm to generate dynamic, model-based thresholds for use with the detection tests.

In order to modularize our algorithms and enable them to more easily adapt to a wide variety of robot structures, this paper develops a multilayer intelligent control framework (see Fig. 1) similar to the hybrid dynamical system framework proposed in [9]. While our framework also has similarities to the broader NASREM architecture [2], our main goal is providing a structure to link our fault tolerance algorithms within the robotic control system. An additional feature of this framework is that it enables easier integration of other research such as more long-term analyses of failure situations [21], [22], [44] which can be performed as background processes so as not to interfere with the fast fault detection and tolerance algorithms. Research in providing automatic fault tolerance by duplicating components [31], [46] or using kinematic redundancy [19], [24] can provide the framework with a wider range of tolerance options for the robot. The framework can further integrate work being done on control-level fault tolerance [28], [31], [32] and detection of software failures [25], [33], [45].

Fault tolerant control of a robot system can be initially divided into three layers: 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

Manuscript received November 12, 1993; revised October 1994. This work was supported in part by the National Science Foundation under grants DDM-9202639 and MSS-9024391 and by DOE Sandia National Laboratory Contracts #18-4379A and #AL-3017. This work was further supported by a Mitre Corporation Graduate Fellowship and NSF Graduate Fellowship RCD-9154692.

M. L. Visinsky is with Dowell Schlumberger Corporation, Tulsa, OK 74134 USA.

J. R. Cavallaro and I. D. Walker are with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251-1892 USA.

IEEE Log Number 9410111.

Fig. 1. Fault tolerant robot system.

interface layer contains the trajectory 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.

We have developed a fault detection and tolerance simulator for a 4-link planar robot with two sensors (encoder and tachometer) per joint as well as for a sensor-enhanced 6 DOF PUMA robot. The simple planar system is easy to simulate and still provides both functional sensor redundancy (position information could be derived from the tachometer and velocity information from the encoder if necessary) and kinematic redundancy with which to test the tolerance methods. The 6 DOF PUMA robot simulation shows how the algorithms can be applied to a larger, more complex system. Kinematically redundant robots or robots with multiple sensors per joint can use the algorithms to obtain more extensive failure detection and tolerance capabilities. Our algorithms can, however, also provide basic detection capabilities for robots which have little or no redundancy with which to tolerate the failures. Table I summarizes the fault detection and fault tolerance capabilities our framework would provide for various robots based on their existing sensor and joint structure.

Section II describes the servo layer of the framework. The interface layer is explained in Section III, which also gives the derivation of the comparison tests used for robotic fault detection based on analytical redundancy methods. The problems associated with the modeling errors inherent in robotic systems are also discussed, followed by a brief description of our new model-based method for generating detection thresholds (ThMB) that mask out these dynamic modeling errors. Section IV explains the supervisor layer and the higher level robot fault tolerance which this layer can provide. Section V summarizes the layer interaction. Section VI shows results for the algorithms proposed in Sections III and IV applied to the sensor-enhanced 6 DOF PUMA robot. Finally, Section VII presents a summary of the conclusions of this paper.

TABLE I
MODULAR FAULT DETECTION AND FAULT TOLERANCE CAPABILITIES

Kinematic Redundancy	Sensor Redundancy	Example Robot	Framework Provided Capabilities per Joint
No	No	Puma	(A) Detect failure; Lock Motor; Work with Reduced Workspace or Stop Robot.
No	Yes	RMS, Sensor Enhanced Puma	(B) Detect Failures; Tolerate sensor failures; Reduce to (A) if all sensors fail.
Yes	No	Rice University Rice-obot	(C) Detect Failures; Tolerate motor failures; Reduce to (A) if kinematically redundant motors fail.
Yes	Yes	RRC	(D) Detect Failures; Tolerate sensor/motor failures; Reduce to (C) if all sensors fail. Reduce to (B) if kinematically redundant motors fail.

II. SERVO LAYER

A. Robot Control

The servo layer of the proposed framework includes the robot and the robot control computer. Rigid link robots are typically modeled using the following form of dynamic equations [27]:

$$\tau = [\hat{M}(\theta)]\ddot{\theta} + \hat{N}(\theta, \dot{\theta}) \quad (1)$$

where τ is the joint torque vector, $[\hat{M}]$ is the inertia matrix, and $\hat{N}$ represents the effects of Coriolis and centripetal forces, gravity, friction, etc. $[\hat{M}]$ and $\hat{N}$ depend on the real robot's parameters and are not known precisely by the controller. The robot controller uses estimates of the real parameters along with sensed estimates of joint positions to calculate the estimated $[\hat{M}]$ and $\hat{N}$. The PD "computed-torque" controller analyzed in this paper takes the form [27]:

$$\tau = [M(\theta)]\{\ddot{\theta}_d + [K_P](\theta_d - \theta) + [K_D](\dot{\theta}_d - \dot{\theta})\} + N(\theta, \dot{\theta}) \quad (2)$$

where $[K_P]$ and $[K_D]$ are system gain matrices with diagonal entries K_p and K_d corresponding to the current joint.

B. Sensors, Motors and Failure Modes

Using the dynamic equation (2), the robot controller computes the necessary torque to apply to each motor in order to move the robot from the given current position to the next desired position. Information about the current position or velocity of the robot joints is relayed to the controller from the robot's internal sensors. The robot computer uses the sensors to determine the errors between the desired and sensed values of joint positions, velocities, or forces so that the computer can compensate for these errors using feedback control. All sensors typically have some form of noise associated with them. This error is compounded by the inaccuracies between the chosen model parameters and the real robot parameters. These errors must be hidden by thresholds (as discussed in Section III-B) in order to focus the detection on real failures (not artifacts

caused solely by modeling inaccuracies). Further, if a sensor is damaged and the failure remains undetected, the controller will receive incorrect information about the joint and the resulting $[M]$ and $\underline{N}$ of (2) will be significantly in error.

The well-being of the sensors is thus critical for the robot in completing its task and we therefore focus our algorithms to provide the fastest possible detection of sensor failures. A frozen failure mode in which the sensor produces a constant value is fairly common. Other possible failure modes include free-spinning, biased, or erratic sensors [30]. We concentrate on the frozen failure mode for this paper with suggestions on appropriate modifications for the remaining modes. Motors are obviously critical to the robot task and the ability to lock a motor in the event of a failure is important in supporting the fault tolerance schemes of Section III. A failed joint which is allowed to swing freely pulls dynamically at the other joints and draws them off course. If a motor fails and locks in place, it will not affect the other joints through coupling although it will stop contributing to the completion of the task. If not already an automatic response to failures, locking may be implemented as a direct command from the interface to the robot. The paths of surviving joints are then modified to move the end effector to its goal (Section III-D).

III. INTERFACE MONITOR AND FAULT DETECTION LAYER

The interface layer contains the robot planner and fault detection routines. Based on the end effector target point or velocity vector passed down from the supervisor, the planner iteratively computes the next configuration for each joint in order to reach this desired goal. The new configuration is then sent to the robot controller in the servo layer (Section II-A).

The fault detection routines continually monitor the state of the robot in order to catch any failures in the system. The routines are unable to differentiate certain failures because only the effect of the failures can be gleaned through the robot's sensors. For example, if a motor is not driving a joint as it has been commanded to do by the controller, detection routines will be unable to determine whether a chain in the gear-train has broken or an internal gear on the motor gear-box has worn down. All the algorithms can see is that, according to the sensors, the desired motion of the joint is no longer being produced. The typically small number of sensors further limits the ability of the detection routines to isolate failures within the system. The tolerance actions we have chosen (switching to surviving sensors or shutting down the motor), however, have the positive effect of covering most of these substructure failures by eliminating or isolating their influence on the rest of the system [41].

A. Analytical Analysis of Detection Relations

To provide the interface with detection/isolation abilities, we must first develop tests which use the available data to watch for sensor or motor failures. It is important to identify as many useful independent tests as possible while eliminating redundant checks. There are established results for the generation of such detection tests or residuals for general dynamic systems using the concept of analytical redundancy

[29]. This section develops the results specifically for robotics along the lines of Chow and Willsky's [4] mathematical characterization of analytical redundancy for dynamic systems discretely modeled as:

$$\underline{x}(k+1) = [A]\underline{x}(k) + \sum_{j=1}^q b_j u_j(k) + W \quad (3)$$

$$y_j(k) = c_j \underline{x}(k) + V, \quad j = 1, \dots, m. \quad (4)$$

The vector $\underline{x} \in \mathbb{R}^N$ is the state vector. The scalar u_j is the known input to the j^{th} actuator and y_j is the scalar output of the j^{th} sensor. Of the coefficients, $[A] \in \mathbb{R}^{N \times N}$ is an $N \times N$ matrix, $b_j \in \mathbb{R}^N$ is a column vector, and $c_j^T \in \mathbb{R}^N$ is a row vector. $W \in \mathbb{R}^N$ and $V \in \mathbb{R}$ represent the noise from nonlinearities and inaccurate sensors. To simplify the derivation, we assume that the sensors read the exact values for the position and speed of the robot (thus $C = I$) and that W and V are zero. The effects of these matrices will be dealt with later.

In the following analysis, the number of sensors per joint is assumed to be two ($m = 2$) and there are two states ($N = 2$, position and velocity) per joint. With this assumption, we can show the improved isolation capabilities over the intuitive base case while still maintaining a realistic structure for the robot. The results can easily be extended to the general case of m sensors [35]. The sensors are chosen to be an encoder and a tachometer which enables us to examine the structure in [4] for temporal redundancy between two different types of sensors. For each joint, the number of actuators, q , is 1. The robot dynamic relations are derived from:

$$\underline{x}(k+1) = \begin{bmatrix} \theta(k+1) \\ \dot{\theta}(k+1) \end{bmatrix} = \begin{bmatrix} \theta(k) + (\Delta t)\dot{\theta}(k) \\ \dot{\theta}(k) + (\Delta t)\ddot{\theta}(k) \end{bmatrix} \quad (5)$$

$$\underline{y}(k) = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \underline{x}(k) = \begin{bmatrix} \theta(k) \\ \dot{\theta}(k) \end{bmatrix} \quad (6)$$

which can be factored into the following matrix format:

$$\underline{x}(k+1) = \begin{bmatrix} 1 & (\Delta t) \\ 0 & 1 \end{bmatrix} \underline{x}(k) + \begin{bmatrix} 0 \\ (\Delta t) \end{bmatrix} u(k) \quad (7)$$

$$\text{and } \underline{y}(k) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \underline{x}(k). \quad (8)$$

θ is the angular position of each joint and Δt is the length of time between each iteration k .

For dynamics (1), the control torque produces an output acceleration based on the equation:

$$\ddot{\theta}(k) = [\hat{M}(t)]^{-1} \underline{\tau}(k) - [\hat{M}(t)]^{-1} (\hat{\underline{N}}(t)). \quad (9)$$

Assuming the controller is reasonably efficient, the control input to the linearized system is essentially given by the feedback-modified acceleration

$$u(k) = \ddot{\theta}_d(k) + K_p(\theta_d(k) - \theta(k)) + K_d(\dot{\theta}_d(k) - \dot{\theta}(k)) \quad (10)$$

where θ_d is a vector of the desired angular positions for each joint. The effects of the typical modeling errors in the control

torques on fault detection will be incorporated at a later stage in the algorithm as discussed in Section III-B. We note that $\underline{x}(k) = [\theta(k) \ \dot{\theta}(k)]^T$ and thus move $\theta(k)$ and $\dot{\theta}(k)$ out of $u(k)$ and combine them with $\underline{x}(k)$ in (7) above. The input $u(k)$ now becomes

$$u_d(k) = \ddot{\theta}_d(k) + K_p \theta_d(k) + K_d \dot{\theta}_d(k) \quad (11)$$

which is only a function of desired values (modifying the results of the original derivation found in [35]). The resulting coefficient matrices in the dynamic equations (7) and (8) are then:

$$A = \begin{bmatrix} 1 & (\Delta t) \\ -K_p(\Delta t) & 1 - K_d(\Delta t) \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ (\Delta t) \end{bmatrix},$$

$$c_1 = [1 \ 0], \text{ and } c_2 = [0 \ 1].$$

We now investigate the sensor output (via the c_j) to find relationships between the sensor readings and inputs that hold in the fault free case. With $C_j(k) = [c_j^T, (c_j A)^T, \dots, (c_j A^k)^T]^T$, ($j = 1, 2; k = 0, 1, \dots$), the observable subspaces [4] for the encoder ($j = 1$) and the tachometer ($j = 2$) are spanned by the rows of:

$$C_1(n_1) = \begin{bmatrix} 1 & 0 \\ 1 & \Delta t \\ (1 - K_p \Delta t^2) & (2\Delta t - K_d \Delta t^2) \end{bmatrix}, \text{ and}$$

$$C_2(n_2) = \begin{bmatrix} 0 & 1 \\ -K_p \Delta t & (1 - K_d \Delta t) \\ c_{31} & c_{32} \end{bmatrix} \quad (12)$$

where $c_{31} = K_p(K_d \Delta t^2 - 2\Delta t)$ and $c_{32} = 1 + (K_d^2 - K_p)\Delta t^2 - 2K_d \Delta t$. Both of these matrices are of rank 2. These ranks imply that no new information can be gained after observing either the encoder or the tachometer for two time-steps [4]. (The feedback data pulled from $u(k)$ provides additional information about the tachometer.) By observing the two previous time steps and the current state of the system and using (12), we can find four constraints on the values of the sensors. We derive four linearly independent ω 's, forming rows of Ω in (13) which combine the observable data and provide tests for the fault-free case by satisfying:

$$[\Omega]_{4 \times 6} \begin{bmatrix} C_1(n_1) \\ C_2(n_2) \end{bmatrix} \underline{x}(k) = 0. \quad (13)$$

One representation of the 4×6 Ω matrix is:

$$\Omega = \begin{bmatrix} 1 & -1 & 0 & (\Delta t) & 0 & 0 \\ (K_p \Delta t^2 - 1) & 0 & 1 & (K_d \Delta t^2 - 2\Delta t) & 0 & 0 \\ (K_p \Delta t) & 0 & 0 & (K_d \Delta t - 1) & 1 & 0 \\ -c_{31} & 0 & 0 & -c_{32} & 0 & 1 \end{bmatrix}. \quad (14)$$

Each row of Ω physically represents various comparisons among the encoder and tachometer readings. When Ω is directly multiplied by the appropriate state histories $\underline{x}_h(k) = [\theta(k) \ \theta(k+1) \ \theta(k+2) \ \dot{\theta}(k) \ \dot{\theta}(k+1) \ \dot{\theta}(k+2)]^T$, the result should be zero for the fault free case.

The sensor inputs ($\underline{x}$) are actual robot positions/velocities, however, and are not observable by fault detection routines.

The detection routines only see the sensor outputs $\underline{y}$ (see Fig. 1). We must transform the equations to obtain $\underline{x}$ in terms of the known $\underline{y}$, b , and $u(k)$. The result, multiplied by Ω , gives the parity vector which now defines the observable relationships for fault detection in terms of obtainable values. The general definition of the parity vector (using the notation of [4] and including the effects of the modified actuator input $u_d(k)$) is:

$$P(k) = \Omega \left\{ \begin{bmatrix} Y_1(k, n_1) \\ \vdots \\ Y_m(k, n_m) \end{bmatrix} - \begin{bmatrix} B_1(n_1) \\ \vdots \\ B_m(n_m) \end{bmatrix} U_d(k, n_0) \right\} \quad (15)$$

where

$$Y_i(k, n_i) = \begin{bmatrix} y_i(k) \\ y_i(k+1) \\ \vdots \\ y_i(k+n_i) \end{bmatrix},$$

$$B_i(n_i) = \begin{bmatrix} 0 & 0 & \cdot & \cdot & 0 \\ c_i b & 0 & \cdot & \cdot & 0 \\ \cdot & c_i b & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ c_i A^{n_i-1} b & c_i A^{n_i-2} b & \cdot & \cdot & c_i b \end{bmatrix} \quad (16)$$

and $n_0 = \max(n_1, n_2)$, for $i = 1, \dots, m$ and $q = \text{number of actuators} = 1$.

In the fault-free case, $P(k) = 0$. With simple noise (due to inexact linearization in the controller, for example), $P(k)$ becomes a random vector with zero mean [4]. With noise and failures, $P(k)$ will be biased away from zero indicating a failure. For this analysis, $P(k)$ is defined by the following matrices:

$$Y_1(k, n_1) = \begin{bmatrix} \theta(k) \\ \theta(k+1) \\ \theta(k+2) \end{bmatrix}, \quad Y_2(k, n_2) = \begin{bmatrix} \dot{\theta}(k) \\ \dot{\theta}(k+1) \\ \dot{\theta}(k+2) \end{bmatrix},$$

$$B_1(n_1) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \Delta t^2 & 0 \end{bmatrix}, \quad B_2(n_2) = \begin{bmatrix} 0 & 0 \\ \Delta t & 0 \\ \Delta t(1 - \Delta t K_d) & \Delta t \end{bmatrix},$$

$$\text{and } U_d(k, n_0) = \begin{bmatrix} u_d(k) \\ u_d(k+1) \end{bmatrix}.$$

Table II shows the tests found by substituting these matrices into (15) with $P(k) = 0$. Note that the input has been converted back into the original $u(k) = u_d(k) - K_p \theta(k) - K_d \dot{\theta}(k)$, and the second derivative of the tachometer value ($\ddot{\theta} = \frac{\dot{\theta}(k+2) - \dot{\theta}(k+1) - (\dot{\theta}(k+1) - \dot{\theta}(k))}{\Delta t^2}$) is compared with the jerk or first derivative of acceleration. The comparisons in Table II monitor the changes or histories of the state variables. The tests are fundamentally the only independent tests which can be performed using the input (u) and sensor output information (y_1 and y_2) on the system defined by (3) and (4) [4]. In the following section, we show how these tests can be effectively used in the inexact, dynamic world of robotics. The detection results are given in the example of Section VI.

TABLE II
ANALYTICAL REDUNDANCY DERIVED DETECTION TESTS

Encoder Reading versus Tachometer Reading:	$\frac{\theta(k+1)-\theta(k)}{\Delta t} = \dot{\theta}(k).$
Encoder Reading versus Computed Acceleration:	$\frac{(\theta(k+2)-\theta(k+1))-(\theta(k+1)-\theta(k))}{\Delta t^2} = u(k),$
Tachometer Reading versus Computed Acceleration:	$\frac{\dot{\theta}(k+1)-\dot{\theta}}{\Delta t} = u(k),$
Tachometer Reading versus Computed Jerk:	$\ddot{\theta} = \frac{(u(k+1)-u(k))}{\Delta t},$

TABLE III
DETECTION TESTS BASED ON EXPANDED ANALYTICAL REDUNDANCY DERIVATION

Encoder Reading versus Tachometer Reading:	$\frac{\theta(k+1)-\theta(k)}{\Delta t} - \frac{\Delta t}{2} u_d(k) = \dot{\theta}(k).$
Encoder Reading versus Computed Acceleration:	$\frac{(\theta(k+2)-\theta(k+1))-(\theta(k+1)-\theta(k))}{\Delta t^2} - \frac{u_d(k+1)}{2} = u(k),$
Tachometer Reading versus Computed Acceleration:	$\frac{\dot{\theta}(k+1)-\dot{\theta}}{\Delta t} = u(k),$
Tachometer Reading versus Computed Jerk:	$\ddot{\theta} = \frac{(u(k+1)-u(k))}{\Delta t},$

The discrete representation of the state space can be made more accurate or closer to the continuous time system by using more terms from a higher order series expansion. If we include the third term in the series (with a coefficient of $\frac{\Delta t^2}{2}$), the discrete state equation (7) would become:

$$\underline{x}(k+1) = \begin{bmatrix} 1 & (\Delta t) \\ 0 & 1 \end{bmatrix} \underline{x}(k) + \begin{bmatrix} \frac{\Delta t^2}{2} \\ (\Delta t) \end{bmatrix} u(k). \quad (17)$$

Only the coefficient vector b changes. This, in turn, modifies the B_i matrices:

$$B_1(n_1) = \begin{bmatrix} 0 & 0 \\ \frac{\Delta t^2}{2} & 0 \\ \frac{3\Delta t^2}{2} & \frac{\Delta t^2}{2} \end{bmatrix},$$

$$B_2(n_2) = \begin{bmatrix} 0 & 0 \\ \Delta t & 0 \\ \Delta t(1 - \Delta t K_d - \frac{\Delta t^2}{2} K_p) & \Delta t \end{bmatrix}.$$

The new tests (Table III) derived using these B_i matrices in (15) are very similar to the tests of Table II. The difference is that correction terms are added to the encoder-based tests representing the more accurate discretization of the state derivative. (As more of the higher order series terms are considered in the derivation, the correction terms become more complex and would also affect the remaining tests.)

We do not expand on the more complex discretization of the state space in this paper as the resulting tests are not as intuitive as the basic comparison tests of Table II. In the fault detection code, the correction terms would be more easily implemented using the basic tests from Table II. The detection thresholds of Section III-B may also be affected by the correction terms in such a way as to make threshold generation procedures based on the AR derivation more time consuming and less effective for robotic use. Future research would have to address these concerns before the expanded derivation could be utilized.

Two apparent problems with the ideal tests are that the sensors are not perfect (for example, the encoder truncates values) and some sensed values must be differentiated to compare them to the values from functionally equivalent sensors (the encoder would be differentiated twice to compare it to the computed acceleration). The inherent loss of precision and noise in

the sensors are compounded through the differentiation. The fault tolerance algorithms further the problem by replacing faulty sensors with data from functionally equivalent sensors (i.e., when an encoder fails, its data is replaced by the position information derived from integrating tachometer readings). To eliminate the differentiation problem, the encoder reading is compared to the expected position derived from the planner or to the integrated tachometer reading which is more stable. The RMS uses similar tests to detect encoder failures [1] but takes several time steps to verify the error. The controller is designed to adjust to small errors in the system. We therefore accept the slight increase in noise caused by differentiation/integration compared to the disastrous consequences of allowing faulty data into the system.

B. Implementation of Analytical Redundancy for Robotics

In their pure form, the ideal tests derived from analytical redundancy cause an unacceptable number of false alarms due to the sensor and modeling errors which arise, for example, from linearization of the robot equations and inaccuracies in model parameters such as link inertia or mass. An acceptable bound or threshold for the difference between the desired value and the sensor reading must be chosen [23] to mask out these inaccuracies. Typically, the thresholds are determined empirically by monitoring a fault-free run and setting the threshold larger than the noted maximum deviation from the desired path. This method results in a constant threshold based on a specific trajectory. The effects of the modeling errors and sensor noise, however, fluctuate dynamically as the motion of the robot changes and as failures occur. The thresholds therefore need to be dynamic in order to cope with the ever changing robot status while remaining small enough to catch any failure-induced errors that the feedback controller would be unable to handle.

In searching for an efficient method for generating robotic detection thresholds, we analyzed the Reachable Measurement Intervals (RMI) algorithm developed by Horak, *et al.* [14]–[16] to compute detection bounds for aircraft systems. RMI provides conditions for finding the extreme possible system outputs $y(t)$ for each input $u(t)$ and state $x(t)$ given bounded parameter variations in the coefficient matrices of the

dynamic equations (A, b , and W from (3)). The procedure is started at a known state in the past history of the trajectory and run for a specific window of iterations to reach the current time. This allows RMI to take into account the global effects of the parameter uncertainty. We initially implemented the algorithm for a single-joint (pendulum) robot to demonstrate the procedure and then explored the expandability of the algorithm and the effects of joint coupling with a two-joint simulation [37], [43]. RMI provided bounds that were generally smaller and much more dynamic than the empirically determined thresholds. Through the examples, we were also able to isolate and view the effects of gravity and joint coupling on the thresholds.

RMI, however, is designed for systems with primarily linear dynamics and thus requires special tailoring to fit the nonlinear dynamics of robot manipulators. Our analysis showed that RMI is also not readily scalable to larger robots. The RMI equations become unreasonably complex to implement for robots of useful size. These complex equations must be recomputed at each iteration for each set of uncertain parameter extremes in order to determine the maximum and minimum output. Increasing the number of joints in the robot geometrically increases the number of comparisons needed during the run. RMI further requires a large amount of memory to maintain the various matrix sets used in finding the maximum and minimum variances. Some savings are possible due to repetition within the matrices, but the memory requirements may be too large to justify the autonomy needed for quick, on-board fault detection, especially in costly space robotics.

In addition, a history of the effects are built-up by beginning the algorithm at a given distance in the past and working back up to the present iteration, computing the maximum and minimum for each step. Thus, the time needed to compute the RMI bounds for robots with many uncertain parameters or multiple joints prohibits a real-time response to failures for the detection algorithms. If the robot cannot make a fast decision about a possible failure, the errors rapidly escalate and endanger the robot and the environment. Previous analysis [35] has shown that with even a simple stuck sensor failure, it only takes a few iterations of the controller before the robot begins to swing wildly out of control (see also Figs. 7 and 11).

To overcome these concerns, we present a possible alternative to the RMI concept which still utilizes the idea of finding the maximum variance possible due to bounded parameter errors at each state iteration. We have exploited the similarities between RMI and the analytical redundancy (AR). With our new method, we found that only two time steps of history were needed for adequate fault detection in robots (Section III-A). Our new model-based threshold (ThMB) algorithm is a local optimization as compared to the global optimizations of the RMI method. It is possible that a smaller error at the current iteration will ultimately result in a larger error later in the trajectory than the error currently used in the calculations. Despite this problem, the use of only local history allows ThMB to perform much fewer calculations at each iteration and it is thus able to produce a threshold much faster than RMI (the window used in our implementation of RMI took

one hundred internal iterations to produce the thresholds for each robot controller cycle). This fast response is essential for robotic fault detection. ThMB can easily be made less prone to false alarms by assuming slightly larger bounds for the parameter errors than necessary.

A useful facet of rigid link robot dynamics is the linearity with respect to parameters in the nonlinear equations of motion. As in numerous adaptive control strategies [27], we can separate the parameter-based coefficients from the joint variables to produce a linear relationship of the form

$$\tau = [\hat{M}(\theta)]\ddot{\theta} + \hat{N}(\theta, \dot{\theta}) = [Y(\theta, \dot{\theta}, \ddot{\theta})]\hat{p} \quad (18)$$

where $\hat{M}$ and $\hat{N}$ are as before. The matrix Y contains known functions of the state variables $(\theta, \dot{\theta}, \ddot{\theta})$ and $\hat{p}$ is a vector of the parameter coefficients.

Parameter adaptive control methods [17] are conducive to determining the bounds on the tracking error ($e = \theta_d - \theta$, where θ_d is the desired trajectory) due to modeling inaccuracies because of the separation of the parameters in the control equations. The inverse dynamics control law (as in (2)) uses the given or nominal estimates of the system parameters in $\hat{M}$ and $\hat{N}$:

$$\tau = M(\theta)[\ddot{\theta}_d + K_d\dot{e} + K_p e] + N(\theta, \dot{\theta}). \quad (19)$$

The real parameters in the robot vary from these estimates but are within the range given by the parameter extremes: $p_{min} \leq \hat{p}_i \leq p_{max}$ where $\hat{p}_i$ is the i^{th} component of $\hat{p}$. The resulting robot equation is

$$\ddot{\theta} = \hat{M}^{-1}\tau - \hat{M}^{-1}\hat{N}. \quad (20)$$

Substituting the torque from (19) into (20), we get

$$\hat{M}\ddot{\theta} = M[\ddot{\theta}_d + K_d\dot{e} + K_p e] + (\hat{N} - \tilde{N}). \quad (21)$$

Changing $\ddot{\theta}_d$ into $\ddot{e}$ by adding and subtracting $M\ddot{\theta}$ on the right hand side and rearranging the equation leads to:

$$\ddot{e} + K_d\dot{e} + K_p e = M^{-1}(\tilde{M}\ddot{\theta} + \tilde{N}) \quad (22)$$

where $\tilde{M} = \hat{M} - M$ and $\tilde{N} = \hat{N} - N$. Using (18), we can modify the right hand side to separate the parameters from the state variables:

$$\ddot{e} + K_d\dot{e} + K_p e = M^{-1}[Y]\tilde{p}. \quad (23)$$

Note that the inaccuracies in the models are contained entirely in the parameter vector $\tilde{p} = \hat{p} - p$.

Equation (23) represents the focus of our new Model-Based Threshold (ThMB) generation algorithm. Here, all the unknown portions of the dynamics have been shifted into the parameter error vector $\tilde{p}$. By selecting appropriate combinations of the extreme variations for the parameters so as to mimic the extreme effects of the real values ($\hat{p}$) present in $\tilde{p}$, we can calculate the possible maximum errors in the positive and negative direction for the next iteration of both $\dot{e}$ and e . Notice that in the ThMB method, the inherent nonlinearity of the robot dynamics is explicitly contained in the derivation, in contrast to RMI, simplifying not only the derivation, but also the resulting equations.

S_ReadFromSnsr command which has the appropriate directions within the accompanying structure. If there are no more available sensors, the interface receives an S_ShutDownMtr command which tells it to shut down the motor in joint i using the I_ShutDownMtr signal. The state now becomes JOINT-FAILED.

The algorithm remains in the JOINT-FAILED state until the task completes or aborts or the motor status is reset at which point it moves back to the IDLE state. Failures are currently considered permanent and only the operator can reset a component's status. If the concept of temporary failures is included, the algorithm could reset the status when a component passed the appropriate test.

D. Fault Tolerance of Joint Failures

To enable the robot to follow the desired path in the presence of joint failures, the supervisor layer of the framework must alert the interface-level planner when a joint has failed. A kinematically redundant robot can withstand joint failures up to the degree of kinematic redundancy without limiting the end effector range of motion although the dexterity may decrease [20], [26]. For the robot, the kinematic relationship $\underline{v} = [J_n(\underline{\theta}_n)]\dot{\underline{\theta}}_n$ ($J_n \in \mathbb{R}^{m \times n}$, $\underline{\theta}_n \in \mathbb{R}^n$) becomes $\underline{v} = [J_{n-k}(\underline{\theta}_{n-k})]\dot{\underline{\theta}}_{n-k}$ ($J_{n-k} \in \mathbb{R}^{m \times (n-k)}$, $\underline{\theta}_{n-k} \in \mathbb{R}^{n-k}$), formed with the column(s) corresponding to the failed element(s) deleted from J_n and the locked position of the failed joint(s) used appropriately in the resulting J_{n-k} . In order to maintain the desired trajectory $\underline{v}$ in a failure situation, the interface planner replaces the n joint solution $\dot{\underline{\theta}}_n = [J_n(\underline{\theta}_n)]^+ \underline{v}$ (where $J^+ = J^T(JJ^T)^{-1}$) by $\dot{\underline{\theta}}_{n-k} = [J_{n-k}(\underline{\theta}_{n-k})]^+ \underline{v}$, with the appropriate k column(s) of J_n removed and the locked position(s) of the k failed joint(s) used in the Jacobian calculations. This approach plans end effector motion $\underline{v}$ using the available (reduced) joint set $\underline{\theta}_{n-k}$, $(n-k) \geq m$.

In detecting and isolating sensor failures, the interface layer uses the redundant information from surviving sensors to provide basic single sensor fault tolerance for the robot. Additionally, the interface can detect multi-sensor failures or motor failures and initiate the actions necessary to tolerate the resulting joint failures. The robot can maintain its range of motion in the presence of joint failures up to the degree of kinematic redundancy of the robot. The interface works with the planner to keep the robot arm tracking the new desired values towards the target goal.

IV. SUPERVISOR AND FAULT TOLERANCE LAYER

We have constructed an expert system based supervisor [36], [39], [41] which performs robotic fault tolerance using fault trees, a standardized representation of the robot's fault structure. The supervisor prunes failed components or branches from the fault trees as failures are reported from the detection routines in the interface layer. The changing tree information is graphically displayed to give the operator an idea of how the failures are interacting and propagating through the system. The system further textually 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 supervisor also maintains the probability of failure for each node within the trees. The expert system uses time-varying probabilities [34] to model component decay during the life of the robot. As detailed in [34] and [39], the probability of a component failure can be easily calculated from the standard failure rate for the component [11], [12]. Within the fault trees, these failure probabilities are combined through logic gates connecting the nodes using simple multiplication and addition.

The 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 significantly different origin and structure can be 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.

By dynamically modifying the probabilities as failures occur, the expert system can alert the robot operator of any large changes (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 task-oriented decisions on 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 [21] so as not to put undo stress on systems which are injured or "sore."

A. Supervisor States

The supervisor layer interacts with the operator and works with the interface to provide fault tolerance of sensor and motor failures for the robot system. Fig. 4 shows the generalized state diagram for the supervisor layer of the framework. All states but the IDLE state represent meta-states with several procedures being performed within each state. Again, signals prefaced by an "S_" or an "I_" originate in the supervisor or interface, respectively. Signals prefaced by an "O_" are from the operator. Transition conditions without a preface are local to the supervisor.

The supervisor initially receives from the operator a target position for the end effector and a list of which internal sensors to use for monitoring each joint as well as the O_Start signal. The supervisor transitions on the signal from the IDLE state to the PROCESS state where it digests the command data, sends the starting velocities to the interface-level planner, and signals the interface with the appropriate set of S_ReadFromSnsr commands to start monitoring the task. The supervisor then waits in this state until a signal is received from the interface or the operator signals an abort. The supervisor moves back into the IDLE state from the PROCESS state if the interface signals that the task has been completed with the I_Done signal.

Fig. 4. State diagram for supervisor.

When the supervisor receives an *I_LogSnsrFail* signal, it accesses the accompanying data structure and records the failure by modifying a database. The supervisor also alerts the operator of the failure. The operator has the option to signal an abort at any time if it is deemed that the task has been compromised. The supervisor does not determine an alternate component until an *I_AllSnsrFail* signal is also received from the interface. The supervisor will then transition to the CHECK FOR SENSORS state and use information learned by traversing the database to develop new tolerance strategies based on the existing structure of the robot. If there are alternate sensors available or external sensors which can be used to monitor a joint, the supervisor signals the interface with the *S_ReadiFromSnsr* command and corresponding data. The supervisor then transitions back into the PROCESS state. If there are no available replacements, the supervisor transitions to the CHECK-FOR-MOTORS state.

The supervisor also transitions to the CHECK-FOR-MOTORS state in response to an *I_MtrFail* signal. The supervisor again modifies its records and alerts the operator of the failure. The supervisor now checks to ensure that the robot can continue working with one less joint. If the robot is kinematically redundant and has not yet dropped down to the base number of joints for the robot, the supervisor moves back to the PROCESS state and sends an *S_ShutDownMtr* command to the interface. In the presence of failed joints, the supervisor command *S_ShutDownMtr* signals the interface-level planner to keep the robot on the desired course using the techniques of Section III-D. If the robot cannot complete its mission, the supervisor signals the interface to shut down the entire robot with an *S_Abort* command. The supervisor also alerts the operator and transitions into the IDLE state to await further instruction from the operator. At this stage the operator would need to repair the robot or alter the task required before restarting the robot. The component failure condition tables would need to be reset when a component is repaired or replaced.

B. Alternative Supervisor Sub-Modules

The supervisor provides higher levels of intelligence to the basic tolerance of sensor and motor failures provided by the

interface layer fault detection algorithms. The work done by Tesar *et al.* [28] and Wu *et al.* [46] on redundant motors would increase the redundancy of the system and provide a wider range of survivor choices for the supervisor or interface. There may be alternate sensors available or external sensors (such as cameras) of which the interface layer is originally unaware that can be used to monitor a joint for failures when the known or basic set of sensors has failed. The expert system will be able to utilize these alternatives and develop new fault tolerance strategies based on information learned by traversing the database.

The supervisor can be further enhanced to monitor the reachable workspace of the robot throughout its operation [26]. While it waits for a signal in the PROCESS state, the supervisor can spawn a process to determine the area reachable by the end effector with the robot in its current state. When motor failures occur and a joint is shut down, the workspace shrinks due to the immobility of the failed joint. The supervisor computes the changes to the reachable workspace [26] and decides 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 alerts the operator. The supervisor then waits in the IDLE state for direction from the operator.

Maciejewski's work on kinematic fault tolerance [20] can be integrated into the fault tree database to provide the planner with information on new optimal configurations for the robot in the presence of failures. If the fault trees are pruned due to detected failures, a new optimal configuration could be drawn up to provide a more dynamic preventive maintenance measure.

V. LAYER INTERACTION

We have introduced three layers for fault tolerant robot control with a more formal definition of the layers appearing in Appendix A. The interactive nature of these layers is given below. The structure is similar to that in [9], but has several important differences, notably in the interface structure. Complete and perfect observations are assumed in [9]. Our application is more structured than that in [9] as fault detection routines form an integral part of the interface.

The interface layer isolates the supervisor layer and the human operator from the continuous control algorithms of the plant. The interface runs two continuous functions (sensor monitoring and inverse kinematics) and returns discrete event or failure information to the supervisor. At each iteration of the control, an independent process computes the next set of thresholds for fault detection based on the current state and passes them to the interface. The operator receives a signal when the robot completes the task or when there is a failure. Operators only need to provide direction when a failure situation makes the task impossible, although they may intervene at any time.

As an example of the interaction of this three layer framework, let us assume that the supervisor wants the interface layer to start a task using as controller feedback data only the values derived from the encoder reading at each joint. At some point in the run, the encoder for Joint 2 fails and the test

Fig. 5. Propagation of a sensor failure among framework layers.

monitoring this sensor is violated. When the failure causes a detection threshold to be violated in the tests, the interface sends (see Appendix A) an $I_AllSnsrFail$ signal along with the $I_LogSnsrFail$ signal which tells the supervisor that encoder 2 has failed and the interface knows of no more sensors to use. Fig. 5 shows the supervisor subsequently removing the encoder for Joint 2 from a fault tree database. The database reveals that a tachometer is still available to functionally replace the encoder as they are connected by an AND gate and the failure does not move any farther up the tree. The supervisor can therefore command the interface to read the information from this alternate sensor as long as the operator does not request an abort.

In Fig. 6, the Motor 0 failure is detected by the interface. The interface sends an $I_MtrFail$ signal (via η) to the supervisor. The trees show that the failure would propagate through the OR gate all the way up to the top level AND gate. The supervisor therefore removes the entire Joint 0 subtree. Since the robot has kinematic redundancy enabling it to withstand as many as two joint failures as noted by the fraction in the joint level AND gate, fault tolerant options are still available because only one joint has failed so far. In signaling the interface layer to shut down the faulty joint, the supervisor initiates the joint fault tolerance scheme present in the planner routine.

VI. EXAMPLES

The first robot used to demonstrate the fault tolerance framework was a simulated four link robot originally developed by Hamilton [13] and executed on a Silicon Graphics Personal Iris computer. All joints were rotational and moved in the same plane providing two degrees of kinematic redundancy with a simple structure. The simulator further modeled two sensors (an encoder and a tachometer) per joint in order to

Fig. 6. Propagation of a joint failure among framework layers.

Fig. 7. 6 DOF PUMA robot with failure.

utilize the sensor fault tolerance routines. The simulation has been expanded and currently tests the algorithms on a 6 DOF PUMA robot (see Fig. 7) which, in the fault-free case, uses encoders to provide joint position data and tachometers to provide joint velocity data. The simulator produces a dynamic display of the robot that changes color as failures occur.

The structure of the simulator is shown in Fig. 8. In this implementation, the user instigates failures and performs the actions of the supervisor. The controller is given in (2) and uses the dynamic equations defined in Section II-A as a model for the robot [13]. Inaccuracies on the order of 1% are injected into the robot parameters within $\hat{M}$ and $\hat{N}$ to simulate modeling errors [38], [43]. We do not explicitly model friction in the equations, but friction-based effects could be

Fig. 8. Robot simulator flow chart.

added to the nonlinearities included in $\hat{N}$ as mentioned in Section II-A. The optical encoders estimate the joint positions by truncating the value of each angle given each encoder's precision. Joint velocities pass through a first order filter based on a predetermined motor lag time to produce the tachometer reading. Both sensors are derived from modules of the NASA Trick simulation package [3].

In the framework of Fig. 1 and Fig. 8, the estimates of the angles and velocities are actually passed into the interface layer (Section III) which checks for failures using the new methods described here and in [35]. This layer either passes to the servo layer controller what it considers to be good estimates of the position, velocity, and acceleration $(\theta, \dot{\theta}, \ddot{\theta})$ or instigates the appropriate restructuring strategy based on the available sensor set or instruction from the supervisor layer (Sections III and IV). A planner in the interface produces the desired trajectory through a velocity level inverse kinematics routine [5]. The algorithm computes the Jacobian matrix, J , using the sensed angles and the known link lengths. The pseudo-inverse is computed as $J^+ = J^T(JJ^T)^{-1}$. The desired velocities are then derived from the equation $\dot{\theta}_d = [J^+]v_d$.

The servo-level robot takes the computed torque from the controller and determines the resulting per joint robot angle accelerations based on (1). The robot's position, velocity, and acceleration are then sent to the sensor routines. The robot position is also sent to the TDM graphics simulator which displays the motion on the screen [35]. This is the same graphics program as used by NASA's Trick simulation package. The path of the end effector is displayed and, when a failure is instigated by the simulator, a small red flag appears. Fault-free joints are displayed in green. A joint (and its associated link) will change color when a failure within the joint is detected. The delay between the instigation of a failure (when the flag appears) and the detection of the failure (when the joint changes color) can thus be readily observed in the graphics. The simulator also outputs messages to the screen when failures are instigated or detected. A dynamic display of the robot's fault tree showing (in color) the changes due to failure and tolerance actions has also been recently included in the graphics.

A. Results

In this section, we discuss an example of the fault detection and tolerance algorithms as applied to the PUMA robot.

Fig. 9. PUMA joint 2 response when Encoder 2 fails at time-step 0.5.

Fig. 10. PUMA joint 0 response when Encoder 2 fails at time-step 0.5.

Results for the planar, four-link robot are given in [35]. The first example shows the results for a single stuck encoder failure in joint 2 at time-step 0.5. Fig. 9 shows how the undetected failure causes the joint position (dashed line) to drift off its desired path (ooo line). Because the inaccurate sensor reading is used in the control calculations for all the joints, it affects the other joints as well, to various degrees. When the failure is detected, the controller is able to keep the robot very close to the desired values (solid line, overlapped by ooo line) by obtaining position information from the surviving sensor. Fig. 10 shows how the failure in joint 2 can affect the position of joint 0 (dashed line) versus its desired path (ooo line) if the failure remains undetected. The solid line shows the path of joint 0 if the failure is detected and tolerated.

Fig. 11 displays the end effector path in 3-D for this failure situation. The desired motion (ooo line) of the end effector is quite small. When the failure is detected and tolerated (solid line), the robot tracks the desired trajectory and completely avoids the wild motion. The interface-level detection routines notice the failure when the encoder stops producing the

Fig. 11. PUMA end effector response when Encoder 2 fails at time-step 0.5.

expected value (modulo the current dynamic threshold for that sensor) and then tolerate the failure by relying only on the surviving tachometer.

If the joint 2 encoder failure remains undetected, the end effector quickly begins to deviate from its path (dashed line around $t = 0.5$) and swings wildly out of control. The full deviant motion is shown in the graphics display of Fig. 7. The undetected failure caused the robot to swing through itself and the floor (there are no hard stops in the simulation). Figs. 7 and 11 show that although an undetected local failure affects the expected motion of the *joint* smoothly (it just drifts away in Fig. 9), the deviation of all the affected joints combined can cause drastic changes in the end effector position. Our algorithms are very effective (as shown in the figures) at quickly detecting and tolerating sensor and motor failures thus completely eliminating the resulting wild motions.

VII. CONCLUSIONS

Fault detection and fault tolerance for robots are important issues which have only recently begun to receive attention in the literature. The space program and programs involved in other hazardous environments have increased the need for reliable robots which can withstand system failures without requiring immediate repairs [6], [8], [23]. The increased use of robots in industry and the medical arena will require greater fault tolerance from a safety perspective. The main objectives of this paper have been to develop a multilayered control framework which provides fault tolerance for a variety of robots and to introduce new analytical redundancy-based fault tolerance techniques and detection threshold generation algorithms for robotics. The emphasis is on a modular system that will be able to use the existing structure of a robot as the basis for fault detection and tolerance, without requiring specific types or numbers of joints, motors, or other components.

The framework is divided into a servo layer, an interface/fault detection layer, and a supervisor layer. The interface and supervisor layers work together to tolerate as many sensor and motor failures within the robot as possible without compromising the goal of the assigned task. The theory of analytical redundancy provided a unique method for deriving

the detection tests based on the available functional redundancy in the robot structure. We further developed a new detection threshold generation algorithm that is more efficient and practical for robotic application than previous methods designed for aircraft systems. An operator directs the initial formation and conditions for a given task and expects reports on internal failures. The framework can be embedded in an expert system or a simple algorithmic program.

We have implemented the fault tolerance supervisor described in this paper with the NASA-developed CLIPS [10] expert system software package. (The details are presented in [36] and [39].) This system provides a more intelligent and flexible platform for the supervisor layer. A fault tree database for the supervisor has also been developed for the PUMA and four-link robots. This expert-system based framework can be used to provide fault tolerance to the wide variety of robots which have already been developed to perform tasks for those programs which may be far from human control for long periods of time such as the space or hazardous waste removal programs.

APPENDIX A

FORMAL DESCRIPTION OF LAYERS

Servo: The plant model for robot manipulators is $\ddot{\tau} = \hat{M}(\theta)\ddot{\theta} + \hat{N}(\theta, \dot{\theta})$ (dynamics), with $\theta \in \mathbb{R}^n$, $\tau \in \mathbb{R}^n$, and $\hat{M} \in \mathbb{R}^{n \times n}$ ($\hat{M} > 0$). It produces $\underline{y} = \underline{y}(\theta, \dot{\theta})$ (sensor observations), subject to $\underline{x} = f(\theta)$ and $\underline{v} = [J(\theta)]\dot{\theta}$ (kinematics), where $\underline{y}(\cdot) : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^p$ ($p \geq m$), $\underline{x} \in \mathbb{R}^m$ ($m \leq n$), and $[J] = \frac{\partial f}{\partial \theta}$ as per [18]. The input $\tau(\cdot)$ is restricted to a prespecified set of functions due to joint torque constraints. Output $\underline{y}(\cdot)$ depends on the type of sensors and takes the form of truncation, first order filters, etc.

Supervisor:

1) It is a finite state machine (S, ε, μ) where

- i) S is the state set $S = S_R \cup S_C$, with S_R containing states which accept interface response events, and S_C containing states which produce interface commands.
- ii) ε is the event set $\varepsilon = C \cup R$, where C is the set of supervisor commands, and R is the set of interface responses.
- iii) μ is the transition (partial) function $S_R \times R$ to S_C , and $S_C \times C$ to S_R .

2) The state persists between event occurrences and changes only as a result of command or response events which may occur at any time; the states do not keep track of time.

Note that the supervisor's discrete and time-independent nature allow supervisor functions to be performed by a machine separate from the controller/planner. The supervisor may also monitor other tasks, respond to operator commands, and intervene in the task at any time.

Interface:

- 1) Continuously monitors supervisor for commands and plant for manipulator sensor output.

- 2) Performs planning (inverse kinematics) based on input from the supervisor (working joint information) and plant (trusted sensor information) and uses the desired end effector velocity $\underline{v}_d$ to compute the desired joint trajectory. Formally, this is a map $\delta: C \times Y \times \mathbb{R}^m \rightarrow \mathbb{R}^n$, where $\delta(c, y, \underline{v}_d) = \underline{\theta}_d$ is the nominal input to the controller. Commands c denote which joints are operational and, for which, motion can be planned.
- 3) Returns to the supervisor a response based on the last command and the subsequent plant output. Performs fault detection by accepting sensor readings as input from the plant, testing their values using analytical redundancy, and sending failure reports to the supervisor. Formally, this is a map $\eta: Y \times \mathbb{R}^n \times \mathbb{R}^n \rightarrow R$, where $\eta(y, \underline{\theta}_d, \underline{\theta}_a) = r$ is the response to the supervisor. The output can be a response denoting successful completion or fault information.
- 4) Maps the validated sensor data and nominal trajectory into the plant controller input based on the last observation and any supervisor input. Formally, this is a map $\varphi: C \times Y \times \mathbb{R}^n \rightarrow U$, where $\varphi(c, y, \underline{\theta}_d) = (\underline{\theta}, \underline{\dot{\theta}}, \underline{\ddot{\theta}}_d)$ is the validated control data. Supervisor inputs may be information on available redundant sensors or commands to shut down motors.

REFERENCES

- [1] PDRS MAL 2102. "Payload Deployment and Retrieval System Malfunction Workbook," Technical Manual, NASA Johnson Space Center, Houston, TX, Apr. 1988.
- [2] J. S. Albus, H. G. McCain, and R. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," NIST Tech. Note 1235, US Dept. Commerce, National Inst. of Standards and Technology, Gaithersburg, MD, Apr. 1989.
- [3] R. W. Bailey and L. J. Quicho, *Trick Simulation Environment Developer's Guide* (Beta-release ed.), Houston, TX: NASA JSC Autom. and Robotics Div., 1991.
- [4] E. Y. Chow and A. S. Willsky, "Analytical redundancy and the design of robust failure detection systems," *IEEE Trans. Automat. Contr.*, vol. AC-29, pp. 603-614, July 1984.
- [5] A. S. Deo and I. D. Walker, "Robot subtask performance with singularity robustness using optimal damped least squares," in *1992 IEEE Int. Conf. Robot. and Automat.*, 1992, pp. 434-441.
- [6] *Environmental Restoration and Waste Management 5-Year Plan, Fiscal Years 1994-1998 (DOE/S-00097F)*, Dept. Energy, Washington, DC, 1993, vol. 1-2.
- [7] J. V. Draper, S. Handel, and C. C. Hood, "The impact of partial joint failure on teleoperator task performance," in *Proc. 4th ANS Topical Meet. on Robot. and Remote Syst.*, 1991, pp. 433-439.
- [8] W. Fisher and C. Price, "Space station freedom external maintenance task team final report," NASA, Johnson Space Center, 1990.
- [9] A. Göllü and P. Varaiya, "Hybrid dynamical systems," in *IEEE Conf. Decision and Contr.*, 1989, pp. 2708-2712.
- [10] J. Giarratano and G. Riley, *Expert Systems: Principles and Programming*. Boston: PWS-Kent, 1989.
- [11] V. H. Guthrie, J. W. Gebbie, and H. M. Paula, "Bravo 2.0: Reliability, availability, and maintainability (RAM) analysis and risk assessment software," DOE Small Business Innovative Research (SBIR) Program Rep., Phase II - Final Rep. (JBFA-111-92), JBF Assoc., Inc., Knoxville, TN, Jan. 1993.
- [12] 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, TN, Mar. 1989.
- [13] D. L. Hamilton, J. K. Bennett, and I. D. Walker, "Parallel fault-tolerant robot control," in *1992 SPIE Conf. Cooperative Intell. Robot. in Space III*, 1992, pp. 251-261, Boston, MA, Nov. 1992.
- [14] D. T. Horak, "Failure detection in dynamic systems with modeling errors," *AIAA J. Guidance, Contr., and Dynam.*, vol. 11, no. 6, pp. 508-516, Nov.-Dec. 1988.
- [15] D. T. Horak and B. H. Allison, "Experimental implementation and evaluation of the RMI failure detection algorithm," in *Proc. 1987 Amer. Cont. Conf.*, 1987, pp. 1803-1810.
- [16] D. T. Horak and D. M. Goblirsch, "Reachable outputs in systems with bounded parameter uncertainties: Application to failure detection," in *Proc. 1986 Amer. Cont. Conf.*, 1986, pp. 301-308.
- [17] R. Iscrmann *et al.*, "Model based fault diagnosis and supervision of machines and drives," in *Proc. IFAC 11th Triennial World Congr.*, 1990, pp. 1-12.
- [18] K. Kreutz, "On manipulator control by exact linearization," *IEEE Trans. Automat. Contr.*, vol. 34, pp. 763-767, July 1989.
- [19] C. L. Lewis and A. A. Maciejewski, "Dexterity optimization of kinematically redundant manipulators in the presence of joint failures," *Int. J. Comput. and Elect. Eng.*, vol. 20, no. 3, pp. 273-288, 1994.
- [20] A. A. Maciejewski, "Fault tolerant properties of kinematically redundant manipulators," in *1990 IEEE Conf. Robot. and Automat.*, 1990, pp. 638-642.
- [21] ———, "Kinetic limitations on the use of redundancy in robotic manipulators," *IEEE Trans. Robot. and Automat.*, vol. 7, no. 2, pp. 205-210, Apr. 1991.
- [22] W. C. Merrill, J. C. DeLaat, and W. M. Bruton, "Advanced detection, isolation, and accommodation of sensor failures—Real-time evaluation," *J. Guidance, Contr., and Dynam.*, vol. 11, no. 6, pp. 517-526, Nov.-Dec. 1988.
- [23] M. M. Moya and W. M. Davidson, "Sensor-driven, fault-tolerant control of a maintenance robot," in *1986 IEEE Int. Conf. Robot. and Automat.*, 1986, pp. 428-434.
- [24] C. J. J. Paredis, W. K. F. Au, and P. K. Khosla, "Kinematic design of fault tolerant manipulators," *Int. J. Comput. and Elect. Engin.*, vol. 20, no. 3, pp. 211-220, 1994.
- [25] H. Pham, Ed., *Fault-Tolerant Software Systems: Techniques and Applications*. Los Alamitos, CA: IEEE Computer Society, 1992.
- [26] A. K. Pradeep, P. J. Yoder, R. Mukundan, and R. J. Schilling, "Crippled motion in robots," *IEEE Trans. Aerospace and Electron. Syst.*, vol. 24, no. 1, pp. 2-13, Jan. 1988.
- [27] M. W. Spong and M. Vidyasagar, *Robot Dynamics and Control*. New York: Wiley, 1989.
- [28] D. Sreevijayan, S. Tosunoglu, and D. Tesar, "Architectures for fault tolerant mechanical systems," in *1994 IEEE Mediterranean Electrotech. Conf.*, 1994, pp. 1029-1033.
- [29] R. F. Stengel, "Intelligent failure-tolerant control," *IEEE Cont. Syst. Mag.*, vol. 11, no. 4, pp. 14-23, Jun. 1991.
- [30] F. Tangorra, A. D. Montgomery, and R. F. Grasmeder, "Independent orbiter assessment analysis of the remote manipulator system," NASA Working Papers (NAS 1.26:185585 and NAS 1.26:185592), NASA, STS Orbiter Projects Office, Jan.-Feb. 1987-1988.
- [31] Y. Ting, S. Tosunoglu, and D. Tesar, "A control structure for fault tolerant operation of robotic manipulators," in *1993 IEEE Int. Conf. Robot. and Automat.*, 1993, pp. 684-690.
- [32] G. Toye, "Management of non-homogeneous functional modular redundancy for fault tolerant programmable electro-mechanical systems," Ph.D. dissertation, Dept. Mechan. Engineering, Stanford Univ., Stanford, CA, July 1989.
- [33] K. S. Tso, M. Hecht, and N. I. Marzwell, "Fault tolerant robotic system for critical applications," in *1993 IEEE Int. Conf. Robot. and Automat.*, 1993, pp. 691-696.
- [34] 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, US Nuclear Regulatory Commission, Washington, DC, Jan. 1981.
- [35] M. L. Visinsky, "Dynamic fault detection and intelligent fault tolerance methods for robotics," Ph.D. dissertation, Dept. Electrical and Computer Eng., Rice Univ., Houston, TX, Apr. 1994.
- [36] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker, "Expert system framework of fault detection and fault tolerance for robots," in *Robot. and Manufact.: Recent Trends in Res., Educat., and Applicat., Proc. Fourth Int. Symp. Robot. and Manufact.*, 1992, pp. 793-800.
- [37] ———, "Dynamic sensor-based fault detection for robots," in *1993 SPIE Conf. Telemanipulator Technol. and Space Robot.*, 1993, vol. 2057, pp. 385-396.
- [38] ———, "Adaptive fault detection and tolerance for robots," in *Proc. First World Automat. Congr.*, 1994, pp. 205-210.
- [39] ———, "Expert system framework of fault detection and fault tolerance in robotics," *Int. J. Comput. and Elect. Eng.*, vol. 20, no. 5, pp. 421-435, 1994.
- [40] ———, "Robotic fault detection and fault tolerance: A survey," *Reliability Eng. and Syst. Safety*, vol. 46, no. 2, pp. 139-158, 1994.
- [41] ———, "Chapter 3, Robotic fault tolerance: Algorithms and architec-

- tures," in *Robotics and Remote Systems for Hazardous Environments*, vol. 1. Englewood Cliffs, NJ: Prentice-Hall, 1993, pp. 53-73.
- [42] —, "Layered dynamic fault detection and tolerance for robots," in *1993 IEEE Int. Conf. Robot. and Automat.*, 1993, pp. 180-187.
- [43] —, "New dynamic model-based fault detection thresholds for robot manipulators," in *1994 IEEE Int. Conf. Robot. and Automat.*, 1994, pp. 1388-1395.
- [44] D. A. Walker. "Creating reliability in the design process: An evaluation approach, design tool, and example application," Master's thesis, Dept. Mechan. Engineering, Univ. Tennessee, Knoxville, TN, Aug. 1990.
- [45] T. S. Wikman, M. S. Branicky, and W. S. Newman, "Reflexive collision avoidance: A generalized approach," in *1993 IEEE Int. Conf. Robot. and Automat.*, 1993, pp. 31-36.
- [46] E. C. Wu, J. C. Hwang, and J. T. Chladek, "Fault tolerant joint development for the space shuttle remote manipulator system: Analysis and experiment," *IEEE Trans. Robot. and Automat.*, vol. 9, no. 5, pp. 675-684, Oct. 1993.

Monica L. Visinsky (S'90-M'95) received the B.S. degree in electrical engineering and computer science from Rice University, Houston, TX, in 1990. She received the M.S. and the Ph.D. degrees in electrical engineering from Rice University, Houston, TX, in 1992 and 1994.

She is currently with Dowell Schlumberger, in Tulsa, OK. Her research interests include fault tolerance in computer and robotic systems, redundant robotic manipulators, artificial intelligence, and intelligent system design and implementation.

Dr. Visinsky received the MITRE Graduate Fellowship in 1991 and the three-year NSF Graduate Fellowship starting in 1991. She is a member of AIAA, National Space Society, Tau Beta Pi, and Eta Kappa Nu.

Joseph R. Cavallaro (S'78-M'82) received the B.S. degree from the University of Pennsylvania, Philadelphia, PA, in 1981, the M.S. degree from Princeton University, Princeton, NJ, in 1982, and the Ph.D. degree from Cornell University, Ithaca, NY, in 1988, all in electrical engineering.

From 1981 to 1983, he was with AT&T Bell Laboratories, Holmdel, NJ. In 1988 he joined the faculty of Rice University, Houston, TX, where he is an Associate 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 (S'84-M'89) 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. degree in 1989, both in electrical engineering, from the University of Texas at Austin.

Since 1989 he has been with the Department of Electrical and Computer Engineering at Rice University, in Houston, Texas, where he is currently an Associate Professor. 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 Beta Alpha Phi, Eta Kappa Nu, Phi Kappa Phi, and Tau Beta Pi.