Fault Detection and Fault Tolerance in Robotics

Monica Visinsky, Ian D. Walker, and Joseph R. Cavallaro
Department of Electrical & Computer Engineering
Rice University
Houston, TX 77251-1892

Abstract

Robots are used in inaccessible or hazardous environments in order to alleviate some of the time, cost and risk involved in preparing men to endure these conditions. In order to perform their expected tasks, the robots are often quite complex, thus increasing their potential for failures. If men must be sent into these environments to repair each component failure in the robot, the advantages of using the robot are quickly lost. Fault tolerant robots are needed which can effectively cope with failures and continue their tasks until repairs can be realistically scheduled. Before fault tolerant capabilities can be created, methods of detecting and pinpointing failures must be perfected. This paper develops a basic fault tree analysis of a robot in order to obtain a better understanding of where failures can occur and how they contribute to other failures in the robot. The resulting failure flow chart can also be used to analyze the resiliency of the robot in the presence of specific faults. By simulating robot failures and fault detection schemes, the problems involved in detecting failures for robots are explored in more depth. Future work will extend the analyses done in this paper to enhance Trick, a robotic simulation testbed, with fault tolerant capabilities in an expert system package.

1 Introduction

In hazardous environments or environments which are not readily accessible to man, robots must be able to efficiently adapt to failures in both software and hardware in order to continue working until the problem can be realistically repaired. Before a robot can try to cope with a failure, however, it must first be able to detect and pinpoint the problem. This paper develops a basic fault tree analysis of robots in order to obtain a better understanding of where failures can occur and how they contribute to other failures or limitations in each robot. The resulting flow chart style picture of failures in a robot can also be used to analyze the resiliency of the robot in the presence of specific faults. Once a failure has been detected, the robot can reorganize its view of its internal structure in such a way as to hide or isolate the fault so the robot can continue working. The focus of this research is on finding real-time fault detection and fault tolerance methods which maintain as much of the robot's functionality as possible while not requiring that redundant or extra parts be added to the robot.

1.1 Previous Work

1.1.1 Redundancy Based Fault Tolerance

Previous work on fault tolerance in robotics has concentrated on dealing with faults in one specific part of the robot (mechanical failure in the motor, kinematic joint failure, etc.) with only token thought going to the more critical, systemwide effect of the failures. Relatively little focus has been given to the question of how to detect failures in robots. Previous research tends to concentrate on fault tolerance algorithms, especially those schemes which rely on duplicating parts such as joint motors [15,19] for their fault tolerant abilities. These redundancy based schemes are similar to several computer fault tolerant algorithms which are also based on redundancy of parts. One common computer fault tolerant algorithm is Triple Modular Redundancy (TMR) in which three processors all work on the same problem and compare their results. If one of the processors is faulty and its result does not agree with the results of the other two processors, the faulty processor is voted out of the final decision and the correct result is passed on to the rest of the system. This fault tolerance scheme fails, however, if more than one of the processors is faulty. Duplication of physical parts provides a backup in case the element performing the work fails. Redundancy of parts can also provide a useful means of checking to see if a component is in error.

For the equivalent redundancy based robot fault tolerant algorithms, two motors have been placed in each robot joint to provide a backup in case one motor stalls, runs away, or begins free-spinning. The fault tolerant advantages of redundancy have also led to adding extra parallel structures, such as seven legs when only six are needed, in order to allow many different configurations in the presence of a failure. Previous work by Tesar, et al at UT, Austin [15] and independently by Wu [19] with Lockheed at Johnson Space Center have explored the aforementioned method of duplicating motors. Two motors in a joint work together so as to provide one output velocity for the joint. When one of the motors breaks, the other one takes over the faulty motor's functions. The faulty motor must be isolated from the system or the second motor must be able to adjust its output to account for any transients introduced to the system by the failed motor. If the robot is performing a time-critical or delicate task, fault tolerance must allow the robot to get a run-away motor under control quickly before any damage to the environment or the robot occurs.

1.1.2 Structure-Independent Fault Tolerance

Many useful robots have already been created. In order to provide fault tolerance for these robots without redesigning them, algorithms need to be developed that will utilize the advantages of the existing structure and not require the addition of extra motors, sensors, or other components to the robot [16]. These algorithms should be easily adaptable to most robots regardless of the robot structure.

To avoid adding redundant parts for fault tolerance in computers, algorithms have been developed which reconfigure the data or code in a computer system among the working parts after one component has failed. Some computer fault tolerant systems handle a fault by allowing a graceful degradation in functionality or speed. The literature speaks of time redundancy [4] in which a computational cycle is lengthened so a fault-free part (or parts) will have enough time to handle the tasks of a faulty component. Other systems use set-switching or processor-switching schemes [4] for reconfiguration. In processor-switching, fault-free components can be collected to form a basic subpart of the configuration, such as a row of an array, until the full configuration is achieved. This method may, however, require many extra interconnections between components. In software, check bits and error correction codes help insure that data is successfully transmitted in the system and allow a reconstruction of the original data if a transmission line is faulty.

For robotics, however, little work has been done in developing algorithms for accommodating a failure using only the available physical parts. Many robots exist which do not have redundant motors or extensive sensors in the joints. Duplicating motors increases the size of the robot, the cost involved in building it, and the weight and inertia which affect the robot controller. It would thus be cost effective to find fault tolerance schemes that do not rely on a specific robotic architecture to continue working but reorganize the robotic algorithms in the controller or utilize the self-motion capability of robots with redundant joints. In order to develop these schemes, the advantages and capabilities of a general robot architecture must be researched.

Maciejewski at Purdue University has quantified the effect of joint failure on the remaining dexterity of a kinematically redundant manipulator [10]. He calculates an optimal configuration of redundant arms to maximize the fault tolerance while minimizing the degradation of the system in the event of a failure. His method currently only provides fault tolerance if the robot is near this initial configuration and can try and arrange its joints to mimic the fault safe configuration as close as possible. Robot controllers may further attempt to keep the robot arm in a configuration where the joints are arranged to stay away from any possible singularities or uncomfortable positions in case a joint fails during the operation. These studies do not rely on adding extra motors or other components to the robot, but they also do not explore what the robot controller must do in order to utilize the remaining dexterity to continue its tasks.

This paper considers and analyzes systemwide failures (electromechanical, computer software/hardware, etc.) and their inter-relationship via fault trees. We focus on developing fault detection and fault tolerance schemes using only the components normally available to the robot. Previous work in fault tolerance forms a subset of our analysis, and our structure has the additional advantage of allowing the best results from more specific fault schemes to be embedded into our tree analysis.

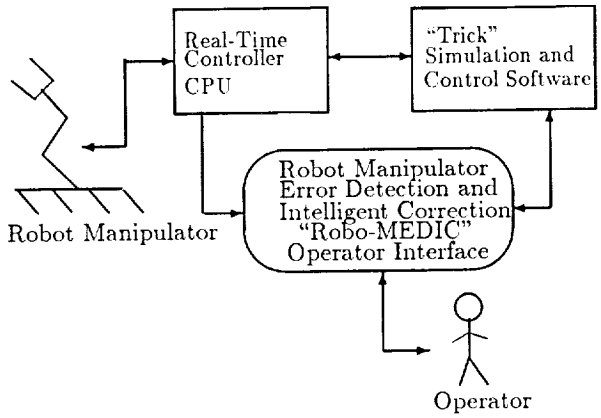

Figure 1: Robo-MEDIC Fault Tolerance Environment.

1.2 Riceobot and Robo-MEDIC

This paper begins by specifically analyzing the fault trees of the Rice University robot, the Riceobot, but the results apply to most robots. The fault tolerant algorithms developed from this analysis will be embedded into the CLIPS expert system environment [6]. This NASA-developed public domain software package is commonly used by government agencies and is running on our computer systems.

The resulting expert system package, Robo-MEDIC (Robot Manipulator Error Detection and Intelligent Correction) will provide diagnostic assistance to the operator and will interface with the control computer of the robot as shown in Figure 1. Robo-MEDIC will be able to use the fault trees as a flow chart of failures. Nodes in the trees will have some fault tolerant action associated with them that will allow the robot to take advantage of inherent backup or alternate paths charted by the fault tree. By maneuvering around the trees, Robo-MEDIC will perform fault tolerant recovery actions as a sequence of these smaller, simpler actions.

1.3 Fault Detection Simulator and Trick

In addition to the fault tree analysis, we are examining failures and testing fault detection schemes using a simulation of a generic four link, planar robot. We will be integrating the concepts derived from the simulator into Trick [1], a robotics software testbed developed at NASA Johnson Space Center by Leslie J. Quiocho and Robert Bailey.

The Trick software package already contains information to model the seven-joint Robotic Research Arms, the Space Shuttle RMS, and the full Riceobot with base and two arms. Data modules provided by Trick allow the user to build customized robots with various types of sensors, joints, and links. Our research is expanding the capabilities of the software to model fault detection and tolerance algorithms. The flexibility of the software allows the failure analysis developed in this paper to be extended to a variety of different robots.

Table I: Fault Tree Analysis Symbols

Symbol	Function
AND gate 	All inputs required to produce output event.
OR gate 	Any one input event causes the output event.
Rectangle 	A malfunction which results from a combination of fault events through logic gates.
Diamond 	A fault event for which the causes are left undeveloped.
Circle 	A basic fault event. This includes component failures whose frequency and failure mode are known.
Triangle 	A suppressed tree. The tree is detailed in another figure.

2 Robotic Fault Tree Analysis

2.1 Analysis Technique

Fault Tree Analysis (FTA) is a deductive method in which failure paths are identified by using a fault tree drawing or graphical representation of the flow of fault events [2]. FTA is a well-known analysis technique often used in industry for computer control systems and large industrial plants. Each event in the tree is a component failure, an external disturbance, or a system operation. The top event is the undesired event being analyzed and, in this research, is the failure of the entire robot. The events are connected by logic symbols to create a logical tree of failures. Some of the basic symbols are explained in Table I.

The explanation of the FTA technique in [2] promotes a top down development of the fault tree. The top event is broken down into primary events that can, through some logical combination, cause the failure at the top. This process is repeated to deeper levels until a basic event or an undeveloped event is reached. Some conditions or causes may be left undeveloped if the probability that they will occur is small enough to be ignored.

2.2 Failure Propagation/Probability Analysis

The information available in the fault trees may be enhanced by a quantitative analysis of the failures. Failure rates are assigned to each input event and propagated up the tree based on the rules of the connecting logic gates. The output of an OR gate is the sum of the inputs. The resulting probability of the combined input events is greater than the probability of an individual input event. The output of an AND gate is the product of its inputs. The probability of all the events occurring is less than the probability of any one occurring. The AND gate represents a redundant measurement or capability and is more desirable in the tree since the probability of a failure decreases through the combination of lower level events.

A Markov or semi-Markov model approach to probability analysis can also be developed based on the PAWS/STEM [3] and CARE III [14] reliability analysis packages. These packages can analyze simpler fault trees as well as Markov chains, but they are not necessarily optimized to handle the simpler structures [11]. These analysis tools were also not designed for robotics and thus would not take advantage of some of the commonality within robot structures. We will include some of the advantageous aspects of using Markov models in our CLIPS-based expert system, Robo-MEDIC.

A quantitative analysis provides a measure of the overall chance of a complete failure for each robot. The structure provided by the fault trees organizes the probabilities appropriately for the robot system and provides a simple map of how the probabilities relate to each other. Using the trees, robots of significantly different origin and structure can be compared for fault tolerant abilities and survivability. The integrated Robo-MEDIC expert system will provide diagnostic capabilities by using the fault trees and will alert the operator of an impending failure. It can be used for off-line comparisons of robots or for suggesting possible corrective actions to an operator and the low-level robot controller during real operations.

2.3 Fault Tree Pruning

A suggested drawback of FTA is that there is no way to ensure that all the causes of a failure have been evaluated [2]. The designer tends to identify the important or most obvious events that would cause a given failure. However, the events that are not modeled normally have a low probability of occurring and can be ignored or treated as a basic event without overly biasing the analysis.

Several failures may also be interconnected creating lateral branches or cycles in the fault tree. In some robots, one motion at a joint may be coupled with another motion such that failure of either motion causes the failure of the other. It is also difficult to determine the relationships between some failures. For example, the failure of all the internal feedback sensors at the elbow joint of a robot may make the robot blind to the elbow's position. The elbow has not actually failed, but the robot is unable to detect the results of any commands sent to the elbow. Thus, the sensor malfunction does not contribute to an elbow failure specifically but may cause a failure of the entire robot. Relationships like these make the tree complex and difficult to analyze. These problems can be overcome by working to simplify the tree. In the case of the coupled motions, the two failures can be considered as one with twice as likely a probability of occurring.

2.4 Riceobot Fault Trees

To provide a foundation for the analysis of general robots, we have chosen to analyze the arm of the Rice University Riceobot. The arm has eight degrees of freedom: three motions in the shoulder (z translation, pitch, and yaw), two motions in the elbow (roll and pitch), and three motions in the wrist (roll, pitch, and yaw). The results obtained from the Riceobot apply to most general robots especially since the Riceobot has a wide variety of commonly used link, joint, and motor arrangements.

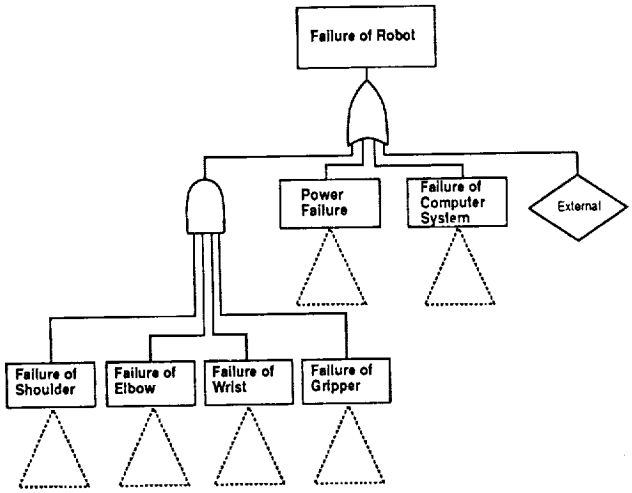

Figure 2: Top Level Fault Tree for Entire Riceobot.

Overall Robot Failure

Several fault trees have been developed and a few are reproduced in the following pages. The top event is obviously the failure of the entire robot (Figure 2). The primary causes of a robot failure are power failure, computer system failure, or a combination of failures of the joints. If the robot is fault tolerant, it can withstand the failure of several joints. By stablizing the faulty motion or joint in some manner (such as locking the joint), it is possible that the other motions can still provide some functional capability to the robot. This ability results in the AND gate combining the joint failures in Figure 2 and decreases the probability of a failure of the robot.

Joint Failures

The Riceobot has two directly driven motions: the shoulder z-direction motion and the wrist roll motion (Figure 3). The fault trees for these motions are quite simple since only the failure of the motor plays an important role in the failure of the motion. The other motions of the Riceobot depend on some form of gear-train assembly to allow the spatially separated motor to drive the joint. Failure of the gear-train can be caused by basic events as simple as a loosening of the chain or cable.

Motor and Sensor Failures

The probability of a motor failure is dependent on the type of motor used. The Riceobot contains both brushless DC and stepper motors. Each motor also has a gear box which may fail due to gear slippage or wear. A power failure affects all motors as well as any other electrically driven parts in the robot, but each motor could lose power separately if its specific power cables break. A motor failure could conceivably also be the cause of a sensor failure when sensors are mounted on the shafts of the motors. Sensors are also affected by incorrect calibration and external noise or vibrations (see Figure 4).

Computer System Failures

The computer system of the Riceobot consists of three main parts: (1) amplifiers which read from the optical encoders and drive the motors, (2) servo control chips which store information about the different motors and convert the desired angles into currents for each motor, and (3) an on-board host computer which is programmed in C and computes the desired angles for the desired motions (Figure 5). These three parts each contain at least one board filled with TTL chips, capacitors, power transistors, resistors, and other analog and digital circuit components. A failure of any one of these parts may not cause a

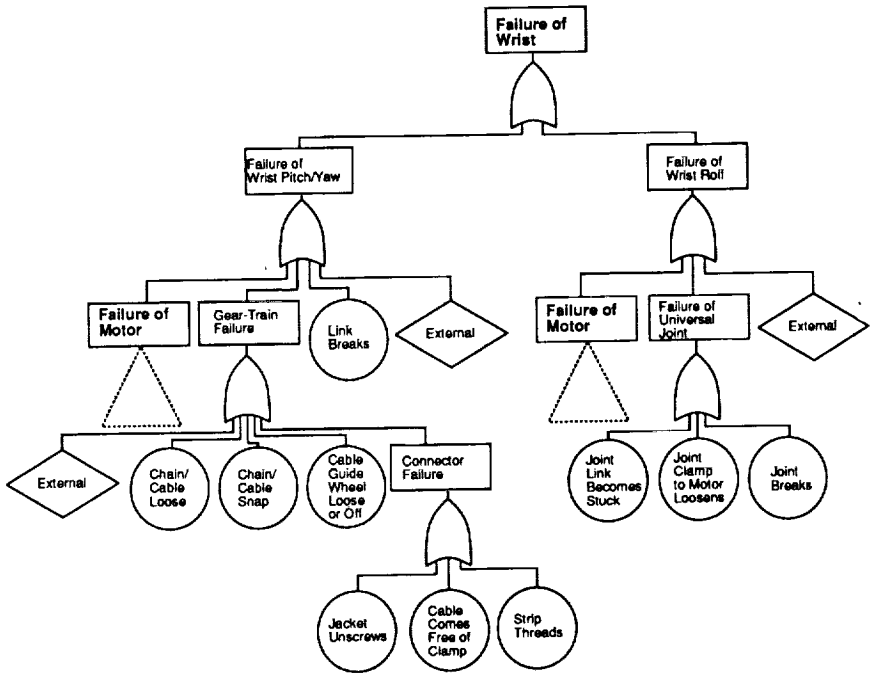

Figure 3: Sub-Level Fault Tree for Wrist System.

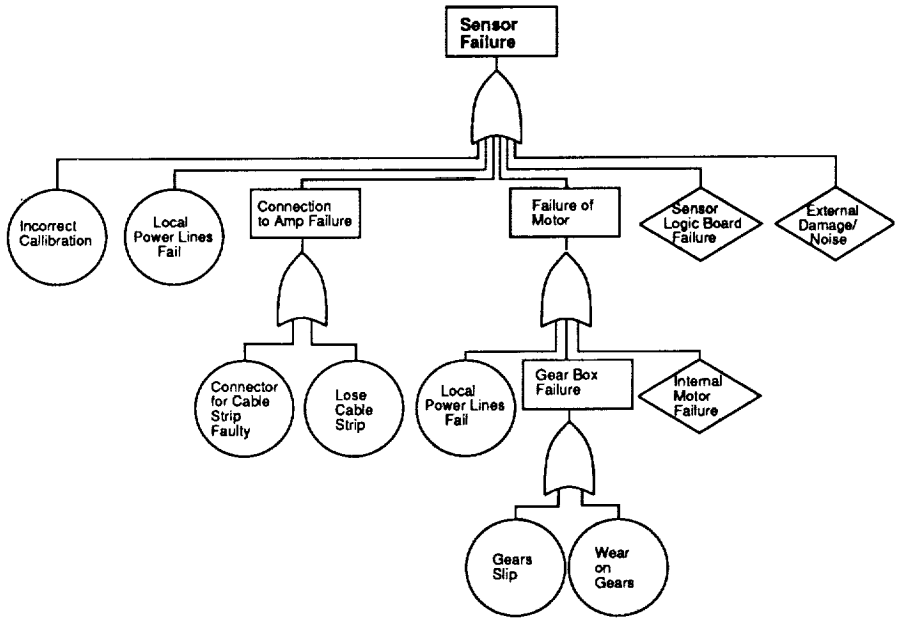

Figure 4: Sub-Level Fault Tree for Sensor System.

failure of the entire board; but if a board did fail, the robot would be unable to function. The robot cannot withstand the failure of all the servo controllers or all the amplifiers because it would no longer be able to communicate with the joints.

Detecting a non-terminal failure in the computer system requires some form of testing circuitry or the ability to poll components to see if they are still alive. The IEEE standard 1149.1 Test Access Port may be incorporated into any VLSI chip on the boards and could be used for active testing [9]. Radiation hardened circuits [18] should also be used in the computer system. Correction code bits can be used to check data transfers and could identify a bus failure if the bits were consistently wrong.

2.5 Derived Riceobot Fault Detection

The qualitative analysis of these fault trees has proven useful in pointing out some of the limitations of the Riceobot in regards to fault detection and fault tolerance. With only one sensor at each joint, the Riceobot represents the worst case scenario for detecting sensor and joint failures. The only option available to the fault detection software is to compare the sensed angles with the calculated desired angles. After accounting for a predetermined threshold to mask any precision errors in the calculations or sensing equipment and possibly adjust for load effects, any difference between the sensed and desired values must be considered the result of a failure. The computer is, however, unable to differentiate between a sensor malfunction and an actual joint failure due, for example, to a frozen motor. The computer must therefore shut down the joint and proceed with fault tolerance schemes based on a new model of the robot with fewer possible motions.

Fault detection for the Riceobot could be improved using the vision system. With the computer calculation and the sensor reading, the additional joint angle information would help distinguish between a sensor error and a real joint failure. The Riceobot could then function in the presence of one sensor failure. Using the vision system for this task, however, increases the load on the image processing software and may hinder the system's ability to perform its normal vision tasks.

3 Robot Fault Detection

The fault trees give an indication of the interaction between failures in a system. The trees also provide a map of alternate paths for detecting faults or bypassing failures. In order to expand on this information and to show how modeling errors or other uncertainties affect fault detection, we need to simulate the robot and the fault detection algorithm. Because of the Riceobot's lack of sensors and the complexity needed for its fault detection algorithm, we are initially simulating fault detection using a computer modeled planar, four link robot [7]. The current program will need to be expanded extensively for the Riceobot and will be accomplished by implementing the fault detection routine in the CLIPS expert system as part of Robo-MEDIC.

The robot consists of four cylindrical links connected end-to-end. All joints are rotational and move in the same plane. A simulated optical encoder and tachometer were added for each joint. The fault tree for this robot is relatively simple (Figure 6). We have not included the possibility of link breakage or global power failure in the simulation. The motors are in essence direct drive with no gear-trains and fail only in a locked mode. These conditions pruned each joint subtree down from the complexity of the Riceobot trees to an easily simulated failure situation.

It is interesting to note that it is the fault detection software which allows the joint to survive in the presence of a single sensor failure thus creating the AND gate under each motor failure in the tree. If both sensors at a joint fail, the host computer is blind to that joint and the fault detection routine forces a motor failure to prevent the joint from moving too far with-

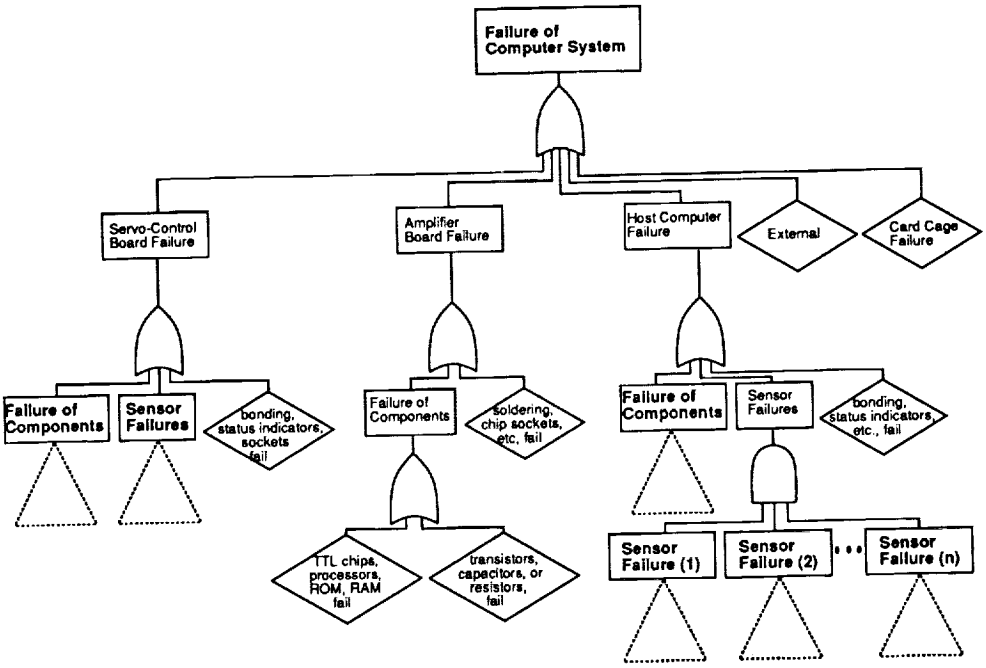

Figure 5: Sub-Level Fault Tree for Computer System.

out computer supervision. Thus, the fault detection algorithm makes the dual sensor failure subtree a cause of the motor failure events to protect a blind joint.

3.1 Fault Detection Simulator

The structure of the simulator is shown in Figure 7. The flow of information is from the simulated host computer through the robot and then the sensors to the fault detection program and finally back to the host computer. The host computer uses the desired angles computed by a planner routine and the estimated present position of the robot derived from the sensors to calculate the torque necessary to move each link to its desired destination. The controller is a standard proportional-derivative (PD) computed torque type controller. The robot routine then takes the calculated torques and determines the new position, velocity, and acceleration for each joint. The optical encoders estimate the positions by truncating the value of each angle based on each encoder's precision. The tachometers pass the velocities through a first order filter based on a predetermined motor lag time. The sensors are modules from the Trick simulation package and represent our initial efforts at integration with Trick. These estimates of the angles and velocities are passed into the fault detection procedure which checks for failures. Then, the procedure either passes to the host what it considers good estimates of the position, velocity, and acceleration or signals the motor of a joint with two bad sensors to shut down.

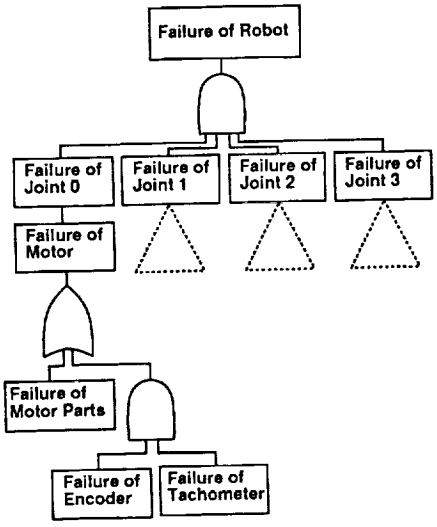

Figure 6: Four Link, Planar Robot Fault Tree.

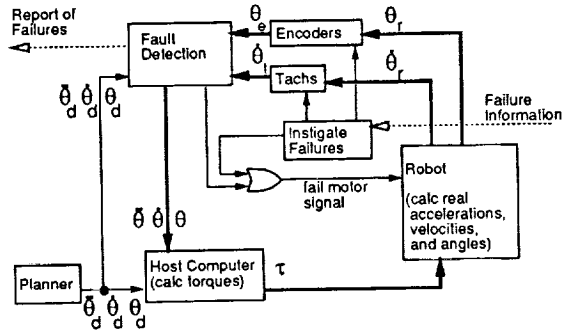

Figure 7: Fault Detection Simulator Flow Chart.

3.2 Host Computer Model

The simulated host computer uses the following dynamics equation as a model for the robot [7]:

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

where $\underline{\tau}$ is the joint torque vector, $[M]$ is the inertia matrix, and $\underline{N}$ is the Coriolis and centrifugal torque vector. The $[M]$ matrix and $\underline{N}$ vector are computed based on the estimated angles from the sensors. Since the robot is planar, gravity is orthogonal to the plane of motion and there are no resultant gravity torques to consider. Friction is also neglected in this model.

The PD controller for this model becomes:

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

The matrices $[K_P]$ and $[K_D]$ are the position and derivative gains, respectively, and are used to control tracking and steady state errors by feedback control. For critical damping, the gains become:

$$[K_D] = 2\omega, \ [K_P] = \omega^2, \quad (3)$$

where ω is the natural frequency input by the user. The natural frequency is typically set to 1 for most runs of the simulator.

3.3 Robot Model

The robot simulation takes the computed torque from the simulated host computer and determines the resulting four robot angle accelerations based on the equation:

$$\ddot{\underline{\theta}} = [\hat{M}]^{-1}\underline{\tau} - [\hat{M}]^{-1}(\underline{\hat{N}}). \quad (4)$$

Here, $[\hat{M}]$ and $\underline{\hat{N}}$ are the inertia matrix and Coriolis and centrifugal torque vector as before but are now based on the actual robot angles instead of the sensed angles. The matrices have also been injected with a small constant error to simulate modeling inaccuracies.

The joint angle, θ, and its first derivative are estimated by the equations:

$$\dot{\theta}_i = \dot{\theta}_{i-1} + (\Delta t)\ddot{\theta}_i, \quad (5)$$

$$\theta_i = \theta_{i-1} + (\Delta t)\dot{\theta}_i. \quad (6)$$

If a motor failure has occurred, $\ddot{\theta}$ and $\dot{\theta}$ are set to zero to simulate the effects of a locked motor. The position thus remains constant. Only the locked motor is currently simulated, but other failure modes could result in runaway motors or free-spinning motors.

The robot's position, velocity, and acceleration calculated in this procedure are sent to the sensor routines. The robot position is also sent to the graphics simulator which displays the motion on the screen. This is the same graphics program used by the Trick simulation package.

3.4 Fault Detection Capabilities

3.4.1 Failure Modes

If a sensor breaks and the failure goes undetected, the host computer will be performing its calculations using erroneous information. In this simulator, the encoders break in a frozen mode continuously reporting the last value read before the failure. The tachometers fail by continuously reporting zero velocities and thus constant positions. With these failure modes, the host sees the error between the sensed angle and the desired angle grow for the joint with the faulty sensor, and the control equations increase the appropriate output torque to the robot to try and compensate for the error. The joint with the faulty sensor swings wildly off course because the host keeps trying harder and harder to get the broken sensor value to match the desired value. Since the calculations for all the joints are based on knowledge of where the other joints are located, all of the other output torques are also computed incorrectly and the joints all stray from their desired paths.

When a motor fails, it locks in position and the joint is then unable to move. If a motor failure goes undetected, the sensors are still reading the correct information. In reality, the motor failure would probably result in a sensor failure as well, but the result would still be that both sensors are reporting a constant joint angle. The control equations try to push the broken joint closer to the desired value but the frozen motor does not respond to the torques. Since the sensors are still reporting the actual position of the joint, all the other calculations are based on correct data and the other joints can continue with their normal motions. The plan must be modified, however, to get the end effector to its desired location.

3.4.2 Thresholds

These two undetected failures reveal the importance of getting accurate sensor readings and of detecting a sensor failure quickly. A frozen motor is not as critical a failure in most cases and can be dealt with at a more leisurely pace. Since the sensors are not perfectly accurate, an acceptable threshold for the error between sensor reading and desired value must be chosen. Unfortunately, even during normal operation, the error between the actual angle and the desired angle can be relatively large especially at the beginning of a run before the controller has had time to bring the error under control. Choosing the maximum error found during a failure-free run results in a threshold that is so large, it may take several time steps to notice the error from a broken sensor. By the time the failed sensor is detected, the robot controller has already been infected with the erroneous information and the robot is either off course or has damaged itself.

Fortunately, the error between the angles recorded by the two sensors during normal operation is very small even after integrating the tachometer reading to get the angular position. Modeling errors and errors induced by unpredicted loads affect both sensors in a similar manner. Thus, a tight threshold can be chosen for a comparison of the two sensed positions. If this threshold is exceeded, the fault detection software assumes that one of the sensors has failed and appropriately chooses one as the working sensor from which to take the recorded data. The larger thresholds from the typical error between the sensed and

desired angles are still monitored, however. The large thresholds provide a means of checking for a motor failure.

The pseudo-code for these checks is reproduced below. The angle θ_d and its derivatives are the desired values. The variables θ_t, $\dot{\theta}_t$, and $\ddot{\theta}_t$ are the values derived from the tachometer reading. The results based on the encoder are θ_e, $\dot{\theta}_e$, and $\ddot{\theta}_e$. Finally, θ and its derivatives are the values sent to the robot controller.

```
If ((encoder working) and (tachometer working)){
    θ = θ_e, θ̇ = θ̇_t, θ̈ = θ̈_t
    If ((|θ_t − θ_e|) >= threshold){
        if (encoder working){
            tachometer = failed
            θ = θ_e, θ̇ = θ̇_e, θ̈ = θ̈_e
        }else{
            encoder = failed
            θ = θ_t, θ̇ = θ̇_t, θ̈ = θ̈_t
        }
    }else{
        if ((|θ_d − θ_t|) >= tachometer-threshold)
            tachometer = failed
        if ((|θ_d − θ_e|) >= encoder-threshold)
            encoder = failed
    }
}
If ((tachometer == failed) and (encoder != failed)){
    if ((|θ_d − θ_e|) < encoder-threshold){
        θ = θ_e, θ̇ = θ̇_e, θ̈ = θ̈_e
    }else{
        encoder = failed
        motor = failed
        send stop motor signal to robot
    }
}
If ((encoder == failed) and (tachometer != failed)){
    if ((|θ_d − θ_t|) < tachometer-threshold){
        θ = θ_t, θ̇ = θ̇_t, θ̈ = θ̈_t
    }else{
        tachometer = failed
        motor = failed
        send stop motor signal to robot
    }
}
```

Choosing which sensor has failed and which is still working when the tight tolerance is exceeded is the most difficult task. Intuitively, one would expect the sensor with a reading closer to the desired value to be the working sensor and would switch to obtaining all the information from that sensor. However, experience has shown that the fault detection software choses the correct sensor only when the desired values are increasing. If the desired angles are decreasing in value, it consistently picks the failed sensor as the working one.

This problem is a result of the time it takes the controller to bring the errors under control and the failure modes for the sensors. Both the encoders and the tachometers fail by reporting a constant angular position either directly or by integration of a zero velocity. First, let us assume the sensors always read less than the desired value. If a sensor fails and gets stuck at a specific value while the desired values are increasing, the error will grow and the fault detection routine should take the angle information from the sensor that reads closer to the desired value. However, if the sensor fails while the desired values are decreasing, the desired values are approaching the failed value. The error starts decreasing and the surviving sensor is often the one whose absolute error is larger. The opposite relationships hold if both sensors are reading values greater than the desired angle. A failed sensor would then have the smaller error during an increase in desired angles and the larger error during a decrease in desired angles.

The various sensor failure situations that arise in the presence of increasing desired values are listed in Table II. The variable d_t is the tachometer error or the absolute difference between the desired value and the tachometer value. Similarly, d_e is the encoder error. The bold faced entries are the actual sensor failures which occur given the specified orderings of the angles and sensed angle errors. The entries enclosed in parentheses are the action taken by our current fault detection algorithm which takes into account the ordering situations described in the preceding paragraph. The intuitive, more naive algorithm would always choose the encoder as the survivor for the case where $d_t > d_e$ and would always choose the tachometer otherwise. The table for decreasing desired values would look similar to Table II but would shut down the joint in the opposite column.

Table II: Failure Situations and Detection Actions for Increasing Desired Angles

Angle Ordering	Error Ordering	
	$d_t > d_e$	$d_e > d_t$
$\theta_d < \theta_t, \theta_e$	**Encoder Failed** (choose tach)	**Tach Failed** (choose encoder)
$\theta_t, \theta_e < \theta_d$	**Tach Failed** (choose encoder)	**Encoder Failed** (choose tach)
$\theta_t < \theta_d < \theta_e$	**Encoder or Tach** (Shut Down Joint)	**Encoder Failed** (choose tach)
$\theta_e < \theta_d < \theta_t$	**Tach Failed** (choose Encoder)	**Encoder or Tach** (Shut Down Joint)

By checking whether the desired values are increasing or decreasing and performing the appropriate comparisons to choose the surviving sensor, the fault detection algorithm can correctly isolate the failed sensor in 75% of the cases instead of 50% for the naive algorithm. The remaining 25% of the cases are inconclusive as either sensor failure could produce the same sensed angle ordering for the given order of the angles. In the case where $\theta_e < \theta_d < \theta_t$ and θ_d is increasing, for example, an encoder failure would result in the encoder error growing larger while the tachometer error still tracks the desired value. Thus, d_e would most likely be greater than d_t. However, if the tachometer failed, the desired value approaches the static tachometer value, and d_e would again be greater than d_t (assuming the angle ordering does not change). Both failures result in the same ordering of the sensor errors. The algorithm shuts down the appropriate joint to avoid choosing the wrong sensor which would feed erroneous information to the controller and cause other joints to swing off course.

The fault detection algorithm thus provides the robot with fault tolerance of most single sensor failures by obtaining the angle information solely from the surviving sensor. Through the detection and isolation of a sensor failure, the algorithm is able to make use of the redundant information provided by the other sensor. When the algorithm cannot isolate the failure, it still protects the system from the hazards of faulty sensor readings.

In general, our simple fault detection simulator is capable of detecting for each joint a single sensor failure, a single sensor failure followed by a motor failure, or a motor failure. The simulator will eventually detect a second sensor failure and will catch the single failures it has missed, but it has allowed enough erroneous sensor readings through to the controller that other joints have been knocked off course and fail as well. In order to improve the fault detection algorithm, we must switch from the hard-coded voting scheme presented above to other forms of analytical redundancy which use filters [12,17], adaptive thresholds [8], or parity relations [5]. Willsky gives a thorough review of the various methods of analytical redundancy in [17] and [5]. Unfortunately, the amount of uncertainty and modeling errors present in most robotic control systems makes several of the proposed methods impractical. The generation of residuals using parity relations is one example of a method which would be unsuitable for robotic applications [8]. Most of the work in analytical redundancy has been focused on failure detection in aircraft, power generation system and other mechanical systems [13]. The algorithms for these systems will need to be modified for robotics applications.

4 Conclusions and Future Work

In this paper we have presented new results in fault tree analysis and fault detection for robot manipulators. This research sets the stage for significantly enhanced activity in fault tolerance for robotics. Once a failure can be detected and isolated, a fault tolerant expert system like Robo-MEDIC can proceed with the appropriate actions to make use of the existing robot structure, redundancy, and alternate paths. There already exist a variety of computer fault tolerance schemes which can provide a starting point for creating these structurally independent robotic fault tolerance algorithms.

The fault tree analysis for the Riceobot has proven useful in pointing out some trouble spots for fault detection and fault tolerance. Even without a quantitative analysis, the importance of certain components and the severity of different failures are revealed in the fault trees. For robots, the good health of the internal sensors is shown to be extremely desirable. Erroneous data from even one sensor at a joint can cause the whole robot to deviate drastically from its course if the failure is not detected quickly. Without the sensors, the robot also loses much of its capability to detect faults. Developing methods for early detection of sensor malfunctions thus has a high priority in this research.

By simulating relatively simple fault detection situations, we are gaining a better understanding of how to satisfy this need for early detection. Our simulator has shown that to avoid false alarms due to modeling errors and noise we are forced to implement large thresholds which let some failures go undetected for too long. Other relationships must be developed concerning the information available in order to improve the fault detection algorithms. We will embed the algorithms in our expert system and integrate the simulation into the Trick simulation package to create a more flexible fault detection and fault tolerance simulator.

The analysis of the fault trees in this paper will be useful in creating fault tolerant algorithms. Through an analysis of the structures, fault tolerance schemes can be developed which will attempt to maintain the health of the internal nodes in the presence of failures in their children. These algorithms will rely only on the available components or structure of the robot and can be developed from the knowledge amassed during the fault detection simulations. The fault tolerant algorithms will also be tested on the enhanced Trick robotic simulation package.

Acknowledgments

This work was supported in part by the National Science Foundation under grants MIP-8909498 and MSS-9024391 and in part by a Mitre Corporation Graduate Fellowship and an NSF Graduate Fellowship.

References

1. Bailey, R. W. and Quiocho, L. J., TRICK SIMULATION ENVIRONMENT DEVELOPER'S GUIDE, NASA JSC Automation and Robotics Division, Beta-release edition, February 1991.
2. Bloch, H. P. and Geitner, F. K., AN INTRODUCTION TO MACHINERY RELIABILITY ASSESSMENT, Van Nostrand Reinhold, 1990.
3. Butler, R. W. and Stevenson, P. H., "The PAWS and STEM Reliability Analysis Programs," NASA TECHNICAL MEMORANDUM 100572, NASA Langley Research Center, March 1988.
4. Chean, M. and Fortes, J. A., "A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays," IEEE COMPUTER, 23(1):55–67, January 1990.
5. Chow, E. Y. and Willsky, A. S., "Analytical Redundancy and the Design of Robust Failure Detection Systems," IEEE TRANSACTIONS ON AUTOMATIC CONTROL, AC-29(7):603–614, July 1984.
6. Giarratano, J. and Riley, G., EXPERT SYSTEMS: PRINCIPLES AND PROGRAMMING, PWS-Kent Publishing Company, Boston, MA, 1989.
7. Hamilton, D., "Elec 590 Project Report," Advisors: I.D. Walker and J.K. Bennett, Rice University, Department of Electrical and Computer Engineering, Houston, Texas, April 1991.
8. Horak, D. T., "Failure Detection in Dynamic Systems with Modeling Errors," JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS, 11(6):508–516, November-December 1988.
9. Kota, K. and Cavallaro, J. R., "Run-Time Fault Detection and Isolation with the IEEE Std 1149.1 Test Access Port," In IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, San Diego, CA, May 1992. To be Submitted.

10. Maciejewski, A. A., "Fault Tolerant Properties of Kinematically Redundant Manipulators," In PROCEEDINGS 1990 IEEE CONFERENCE ON ROBOTICS AND AUTOMATION, pages 638–642, Cincinnati, OII, May 1990.
11. Martensen, A. L. and Butler, R. W., "The Fault-Tree Compiler," NASA TECHNICAL MEMORANDUM 89098, NASA Langley Research Center, January 1987.
12. Merrill, W. C., DeLaat, J. C., and Bruton, W. M., "Advanced Detection, Isolation, and Accommodation of Sensor Failures - Real-Time Evaluation," JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS, 11(6):517–526, November-December 1988.
13. Stengel, R. F., "Intelligent Failure-Tolerant Control," IEEE CONTROL SYSTEMS, 11(4):14–23, June 1991.
14. Stiffler, J. J. and Bryant, L. A., "CARE III Phase II Report - Mathematical Description," NASA CONTRACTOR REPORT 3566, NASA Langley Research Center, 1982.
15. Tesar, D., Sreevijayan, D., and Price, C., "Four-Level Fault Tolerance in Manipulator Design for Space Operations," In PROC. OF THE FIRST INTERNATIONAL SYMPOSIUM ON MEASUREMENT AND CONTROL IN ROBOTICS, Houston, TX, June 1990.
16. Walker, I. D. and Cavallaro, J. R., "Fault Tolerant Robotic Architectures and Algorithms," In SIAM INTERNATIONAL CONFERENCE ON INDUSTRIAL AND APPLIED MATHEMATICS, Washington, DC, July 1991.
17. Willsky, A. S., "A Survey of Design Methods for Failure Detection in Dynamic Systems," AUTOMATICA, 12:601–611, 1976.
18. Winokur, P. S., "Radiation-Hardened Circuits for Robotics," PROC. FOURTH TOPICAL MEETING ON ROBOTICS AND REMOTE SYSTEMS, pages 311–315, February 1991.
19. Wu, E., Diftler, M., Hwang, J., and Chladek, J., "A Fault Tolerant Joint Drive Systems for the Space Shuttle Remote Manipulator System," In IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, Sacremento, CA, April 1991.