

# Fault Tolerance versus Performance Metrics for Robot Systems

Deirdre L. Hamilton, Ian D. Walker, and John K. Bennett  
Department of Electrical and Computer Engineering  
Rice University, Houston, TX 77005-1892

## Abstract

*The incorporation of fault tolerance techniques into robot systems improves the reliability, but also increases the hardware and computational requirements in the overall system. It is not always clear how to evaluate the merit, or ‘effectiveness’ of different fault tolerance approaches for a given application. In this paper, we present a new set of performance criteria designed to measure and compare the effectiveness of robot fault tolerance strategies. The measures, which are designed to evaluate fault tolerance/performance/cost tradeoffs, can also be used to evaluate pure performance or pure fault tolerance strategies. We show their usefulness using a variety of proposed fault tolerance approaches in the literature, focusing on multiprocessor control architectures.*

## 1 Introduction

The first robots were used to perform simple repetitive tasks in static environments, such as assembly line work in a large, unobstructed area. Control and safety requirements in these cases were relatively simple, repair straightforward, and fault tolerance was not a major consideration. However, robots are now being considered for use in many new arenas, such as medical applications, handling of hazardous waste, and uses in space. Fault tolerance is highly desirable in these environments because the results of system failures may be very dangerous and unrecoverable.

To provide an effective fault tolerant robotic system, every key subsystem must be capable of enduring some amount of failure. In the past few years, this issue has motivated significant work in the area of robot fault tolerance [17]. Researchers have considered the use of redundancy and safety checks, at the joint [9, 12] and actuator [13] levels, and also in the software [20] and control architecture [6, 14, 15]. Additional work has considered fault detection [18, 21] and error recovery [4].

This research has produced a series of potentially valuable techniques and architectures for robot fault tolerance. However, relatively few of these new techniques have found their way into actual applications. This is in part due to the lack of an accepted metric which can evaluate the benefits of adopting new robot fault tolerance methods relative to their cost of adoption. The inclusion of fault tolerance and safety features in robot designs typically incurs extra cost, and can decrease system performance.

Consider, for example, a multiprocessor control architecture that allows processor failure detection via data comparison and system recovery by processor reconfiguration or repair.

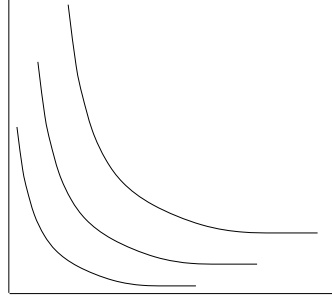


Figure 1: Design Trade-offs

Inclusion of data comparison routines increases the number of tasks in the control loop, increasing the controller execution time. Consequently, the trade-off between providing processor fault tolerance and providing the fastest possible control is unavoidable. This trade-off is illustrated intuitively in Figure 1 (performance represents the inverse of controller execution time). Maximum performance for  $x$  processors is attained with minimal fault tolerance incorporated. Providing maximum fault tolerance for a constant number of processors seriously degrades performance [6].

As another example, the incorporation of kinematic redundancy in a design allows some tolerance of joint failures (assuming the failed joints can be locked) [9]. If there are sufficient degrees of freedom remaining, the manipulator plan can be reconfigured to continue the task with the remaining joints. However, the extra joints add weight, cost, and complexity to the design. Once again, there is a trade-off between fault tolerance and the costs incurred in incorporating it. Performance measures which reflect and quantify this trade-off would be valuable in deciding whether to incorporate new fault tolerance strategies.

Another issue arises when attempting to compare different fault tolerant designs. It is not always obvious which design is best when fault tolerance as well as performance and cost are considered. For example, processor architectures are often compared in terms of performance (execution time). Consider, for example, one architecture designed to tolerate multiple sensor and processor failures and another architecture that cannot survive after one failure of any type, with the same controller execution time. Based on performance alone the two systems would be considered equivalent simply because their controller speeds are the same. On the other hand, fault tolerant architectures may be described in terms of the number of failures tolerated. However, two architectures, capable of tolerating the same number of failures (equal fault tolerance levels), but operating at different speeds should be differentiated. For these reasons, a means of comparison that considers both performance and fault tolerance should be used.

In this paper we present a new metric, which we term an *effectiveness measure*, designed to address some of these issues. The synthesis of performance measures for robotics is not new (see for example [3] for a good list of criteria). However, we believe that the measure described here is the first meaningful measure for fault tolerance/performance trade-offs proposed for robotics, although similar measures have been advocated and adopted in other areas [19]. The effectiveness measure, introduced in the following section, combines two sub-measures for fault tolerance and cost/performance, the selection of which are discussed in Sections 2.1 and 2.2, respectively. Each of these sub-measures can be used independently to analyze pure fault tolerance or pure cost/performance. We illustrate the utility of the

proposed measure in Section 3 using several examples, concentrating on a fault tolerant multiprocessor control architecture.

## 2 Effectiveness Measure

The metric presented below offers a method for rating fault tolerant architectures that also considers their performance and cost levels. Thus, the effectiveness measure has two terms: one based on a fault tolerance rating, the other derived from a performance/cost rating. These ratings are discussed in following subsections. The effectiveness measure should have a high value when both ratings are high and a low value when both are low. Furthermore, when either fault tolerance or performance is critical, the effectiveness measure should reflect its rating. The effectiveness is calculated as follows:

$$eff = k_1(f)^2 + k_2(p)^2 \quad (1)$$

where:

- $eff$  is the effectiveness rating,
- $f$  is the fault tolerance rating,
- $p$  is the performance/cost rating, and
- $k_1, k_2$  are constants reflecting relative importance.

In defining the measure (1), it was desired to utilize as simple a form as possible while reflecting the physical characteristics of fault tolerance/performance trade-offs. Some robot tasks may require higher levels of fault tolerance than others, just as some may have greater performance requirements. Thus the effectiveness measure linearly trades off terms based on sub-performance measures for fault tolerance ( $f$ ) and performance/cost ( $p$ ). The fault tolerance rating is a function of the amount of redundancy incorporated (for e.g., the number of extra links for the robot and/or processors for the controller), and could include the reliability of the individual components of the robotic system. We will elaborate on this point in the next section. The performance/cost rating is further discussed in Section 2.2.

The terms,  $f^2$  and  $p^2$ , are assigned non-negative integer constant coefficients,  $k_1$  and  $k_2$ , that indicate their relative importance. The sum of these two constants is required to be ten. For example, if fault tolerance is extremely important, then  $k_1$  might be 9 and  $k_2$  would, therefore, be 1. By this method, the stunted performance will not greatly diminish the overall effectiveness of the system. The range for each constant is 0 – 10. For a situation in which fault tolerance is the only concern, this measure can still be used by setting  $k_1 = 10$  and  $k_2 = 0$ . Likewise, if the system has no fault tolerance, the effectiveness measure rates performance/cost only. From (1), the range of the overall effectiveness value is 0.0 – 10.0. Therefore, a total effectiveness rating of 5.0 or greater may be considered valuable based on a study of many cases.

In synthesizing (1), we selected real values for  $f$  and  $p$  between 0.0 and 1.0 and considered  $eff = k_1(f)^n + k_2(p)^n$ , with the value of  $n$  ranging 1 – 3. The values of  $n$  greater than 1 produce a nonlinear but gradual increase in the terms of the effectiveness rating for lower values of  $f$  and  $p$  and greater differentiation for higher values of  $f$  and  $p$ . Most of the  $eff$  values calculated for  $n = 1$  were sensible, though some were rather optimistic. On the other hand, some values calculated using  $n = 2$  were a bit conservative, while most were also reasonable. Most values calculated for  $n = 3$  seemed to be too low. We show two examples

$k_1$	$f$	$k_2$	$p$	$eff(n = 1)$	$eff(n = 2)$	$eff(n = 3)$
2	.1	8	.9	7.4	6.5	5.834
5	.1	5	.9	5.0	4.1	3.650

Table 1: Examples of effectiveness values for  $n = 1 - 3$

in Table 1. In the first row, the relative importance of performance is high ( $k_2 = 8$ ), while that of the fault tolerance rating is low ( $k_1 = 2$ ). The resulting effectiveness value for the case  $n = 1$  suggests that the sub-measures,  $f$  and  $p$ , have values that are acceptable for their relative significance (i.e. the more crucial measure, the performance/cost rating, is high). The value acquired for  $n = 2$  ( $eff = 6.5$ ) is somewhat low given that the performance rating is almost at its maximum value of 1.0. Then, in the second row of the table,  $k_1$  and  $k_2$  show the case when fault tolerance and performance are equally important. With a fault tolerance rating of only 0.1, an effectiveness rating of 5.0 is rather optimistic. However, a rating that is less than 5.0 is more reflective of the sub-ratings. We chose to introduce the squared values of  $f$  and  $p$  into (1) because it erred on the conservative side rather than the optimistic side, as demonstrated by these two examples.

Some examples with our choice of  $n = 2$  are shown in Table 2. In the first entry in the

$k_1$	$f$	$k_2$	$p$	effectiveness
1	.9	9	.9	8.10
1	.8	9	.2	1.00
5	.9	5	.5	5.30
7	.7	3	.5	4.18

Table 2: Examples of effectiveness values ( $n = 2$ )

table, the performance/cost rating is the most important term, as indicated by  $k_2$ . Since this rating and the fault tolerance rating are both quite high, the effectiveness rating is also high. The overall effectiveness for the second entry is very low because the most significant rating has a low value. In these examples, the effectiveness rating mirrors the performance/cost rating because  $k_2 = 9$ , i.e. the fault tolerance rating has little effect on the overall system effectiveness. In the third entry, the fault tolerance and performance/cost ratings are equally emphasized. The effectiveness rating has a medium value (min = 0, max = 10) because the rating for performance/cost is only 0.5, even though the fault tolerance rating is high. The fault tolerance rating is the primary term for the final example, though the performance/cost rating also has some significance. Here, the effectiveness rating is slightly below its mean value since the fault tolerance rating is not very high and the performance/cost rating is equal to its mean.

The measure can be used to assess the effectiveness of the robot system even in the presence of some failures. For example, consider the case of a fault tolerant multiprocessor architecture. We can measure  $eff$  at any point, as follows:

- $eff_{max} \Rightarrow$  when all processors are available
- $eff_{(-1)} \Rightarrow$  after one processor has been removed
- $\vdots$
- $eff_{-(max-1)} \Rightarrow$  after all but one processor have been lost ( $eff = k_1(0) + k_2(p)^2$ )

Sections 3.3 and 3.4 contain examples to demonstrate the usage of the effectiveness measure.

## 2.1 Fault Tolerance Rating

Detection of and recovery from component or subsystem failure requires some form of redundancy in general [10]. Redundancy in the software that performs the control calculations allows detection of processor failures through data comparisons, for example, and provides recovery via continuation of another software module. Hardware redundancy allows recovery from subsystem failures by replacing a failed subsystem with a functionally equivalent subsystem.

Subsystems that are tolerant of component failures are sometimes described in terms of the number of failures that can be tolerated. Therefore, this number must be included in the fault tolerance rating. However, we propose to use the fraction of components that may fail rather than the number (note that this will keep the value of  $f$  within the 0 – 1.0 range). This way, a multiprocessor architecture that can lose 4 of 5 processors has a slightly higher rating than the architecture that can lose 3 of 4. Similarly, a manipulator that can lose 3 of 6 joints (such as a nonredundant manipulator engaged in a task where only position, not orientation, of the end effector is important) would have a higher fault tolerance rating than one that could only lose 1 of 7 joints.

For a subsystem providing fault tolerance via redundancy, the fault tolerance rating is:

$$f = m/n \tag{2}$$

where:

- $f$  is the fault tolerance rating,
- $n$  is the number of available subsystems, and
- $m$  is the number of subsystem failures tolerable

For most subsystems,  $m$  will usually be  $(n - 1)$  (processors, sensors, actuators, for e.g.) because only one component of the subsystem is needed to perform the task. However, in some cases  $m$  will have some other value because multiple components are required for proper functionality (such as joints).

By this method, it would seem desirable to begin with a large number of redundant components, regardless of how many are needed to meet performance requirements. However, there is obviously a limit to the amount of complexity that is desired. For example in a multiprocessor system, parallel execution requires synchronization, and there is a point at which the addition of processors is detrimental to performance. There is a minimum response time requirement if reasonable control is to be maintained. Therefore, the maximum (optimal) number of processors (or subsystems in general) used will be driven by performance/cost requirements as much as the desire for a high fault tolerance rating. This value will vary for different architectures and tasks. In Section 2.2, we discuss some of the issues involved in selecting a performance/cost rating.

Note that the above expression for  $f$  assumes that we are considering fault tolerance via redundancy of one particular type of subsystem. In fact, the effectiveness measure framework easily extends to handle the analysis when several different redundant components or safety checks are present in the robot system. The extension to multiple subsystems, which uses the technique of fault trees, is discussed in Section 4.

## 2.2 Performance/Cost Rating

The choice of an appropriate sub-measure  $p$  for performance/cost is less straightforward than that for fault tolerance. Different subsystems will have different performance and cost considerations. Speed, or response time, is a primary performance issue for many subsystems. For sensors and actuators, accuracy is of great importance; and flexibility (or rigidity) may be a major concern for manipulator links. For some systems, financial cost of components will be high and restrict the fault tolerance options. In other cases, financial considerations will be less of an issue, and size or weight may be the main factors influencing the evaluation. A sample formulation for  $p$  is:

$$p = (S + R + C)/3 \quad (3)$$

where:

- $p$  is the performance rating,
- $S$  is speed performance (including fault detection),
- $R$  is the recovery time rating, and
- $C$  is the cost of incorporating fault tolerance

for the given subsystem. The terms contributing to the performance/cost rating would be mapped to the range  $[0, 1]$  in order to maintain the range requirement for  $p$  and to reconcile the unit differences between the terms. The choices for mapping would be determined per application, as demonstrated in the example in Section 3.3.

The fault detection and recovery times are important because they contribute to the control loop execution time in fault tolerant systems (i.e. they affect the granularity of control). These times may vary with the number of redundant components and the method of fault detection used. Since fault detection methods are executed in each loop of the control cycle, the fault detection time is always a part of the loop execution time. Therefore, it is included in the speed performance term. Recovery time, however, is only a factor when a failure has been detected. Furthermore, the recovery time may vary as more components of a subsystem are lost. Detection and recovery times are both applicable to any subsystem (sensors, processors, etc. ). However, regardless of where the failure occurs, it is the control loop execution time that is affected. Therefore, we only include the rating  $R$  when dealing with the controller subsystem. Since the recovery time negatively affects performance, a function should be chosen that decreases the value of  $R$  as the recovery time increases. An exponential function could be chosen to give the desired result:

$$R = e^{-10t_r}$$

where  $t_r$  is the time required to recover from a detected failure. In the above equation, the coefficient of the recovery time,  $t_r$ , determines how dramatically the recovery time rating decreases. We chose a value of 10 for this example. The specific function can be chosen to fit the design goals.

The only requirement for use of the effectiveness measure is that a measure be defined for  $p$  that takes real values in the range  $[0, 1]$ . Here, designs whose performance/cost is unacceptable should rate a value of 0, with increasingly better designs rating monotonically higher scores towards 1. We anticipate that such performance sub-measures should arise naturally for each application. Indeed, we believe that it is preferable for engineers familiar with each individual application to define their own particular performance/cost measure  $p$ .

Our effectiveness measure is designed to provide a framework within which to evaluate various (robot-dependent) fault tolerance options in the context of specific application-dependent measures  $p$ .

In the next section, a specific performance metric  $p$  (related to controller execution time) is given for the case of a multiprocessor control architecture. In this case, fault tolerance is provided by duplication of processors (single subsystem redundancy), and (2) is used for the fault tolerance sub-measure  $f$ .

### 3 Example: Fault Tolerant Multiprocessor Control

To demonstrate the features and potential of the effectiveness measure, we return to the example of a multiprocessor control architecture. This type of system features fault tolerant capabilities via component (processor) redundancy, a clearly defined performance goal (maximum sampling rate), and serves as a good example for fault tolerance/performance investigation.

Different processor configurations provide varied choices for hardware fault tolerance. “Time redundancy” involves reconfiguration of existing units following processor failure [2]. The tasks of the failed processor are executed on one of the remaining processors. Including spare processors is an alternative to time redundancy. These processors would not be included in the working set of processors unless a failure occurs. The advantage of this method is that the number of processors performing the task does not decrease as long as spare processors are available.

Similar to the options for hardware fault tolerance, there are varying levels of software fault tolerance. To provide the highest level of fault tolerance, each processor in an architecture conceptually executes the entire control algorithm. With complete redundancy in the algorithm, data can be compared after every instruction, if chosen, to detect failures. More frequent comparisons allow early failure detection, but also reduce performance. Fewer comparisons allow faster program execution at the possible cost of not detecting a failure as promptly. When fewer comparisons are made, less software redundancy is required. Thus, in this lower level of fault tolerance each processor can execute a different portion of the entire program, allowing the program to be completed more quickly.

The recovery time for the different levels of software fault tolerance will vary. Since each processor is already executing much of the control program for the architecture with a high fault tolerance level, reconfiguration will simply entail getting correct results from a working processor. However, when each processor is only executing a relatively small portion of the total algorithm, reconfiguration may require redoing the work of the failed processor on a neighbor. Thus, the recovery time for a controller with a high fault tolerance level should be shorter than it would be for a controller with a lower fault tolerance level.

The time required for reconfiguration for the different approaches to hardware fault tolerance will also vary. In the case of time redundancy, the recovery time is dependent upon the software configuration. If each processor has the entire program loaded, then a working processor simply performs the tasks that would have been done by the failed processor. If this is not the case, then the program of the failed processor must be loaded onto a working processor. When spare processors are available, the code may have to be loaded onto a spare processor; or the program may have already been running on the extra processor, though it was not being used. Whatever the architecture, the recovery time should be minimized so that the control output is delayed as little as possible.

### 3.1 Fault Tolerance Rating

We consider a time redundant architecture with varying degrees of software fault tolerance in our analysis. Thus, the number of failures that can be tolerated is  $n - 1$ , where  $n$  is the number of processors available when execution begins. Assuming one processor is capable of performing the task, the fraction of potential failures is always  $q = (n - 1)/n$ . (The situation where more than one working processor will be required to maintain a satisfactory sampling rate is taken into account within the performance rating). If one processor performs the task too slowly to maintain control, the performance rating will be appropriately affected.

### 3.2 Performance Rating

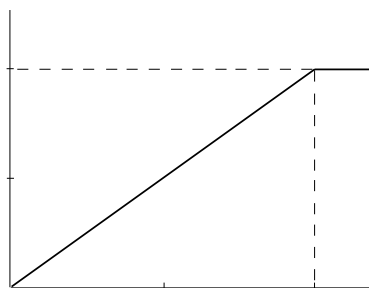


Figure 2: Map of performance rating

The performance rating is a function of the control sampling frequency as shown for one example in Figure 2. For this case, we used timing data for the PUMA robot simulation that is discussed in Section 3.3. We chose to give the uniprocessor controller in our example a performance rating close to 0.25. This corresponds to a sampling rate of 40 Hz being rated at 0.5, and any sampling rate greater than 80 Hz being rated 1.0, or completely satisfactory. However, this choice will vary in practice for different applications, and the choice here is made for the purposes of illustration.

In certain environments, it is important to maintain a fine granularity of control (high sampling rate) of the robot. To do this, the control loop execution time must be as short as possible. This addresses the question “Why would an architecture with a maximum of 8 processors only be able to tolerate 4 failures?” Though one processor can always perform the task, it may not be able to complete the control loop quickly enough to keep the robot sufficiently close to the desired path. The controller frequency should be mapped to the performance rating so that loop execution times that are too slow result in low performance ratings.

### 3.3 Numerical Example

We next give a numerical example of the use of our effectiveness measure for evaluating multiprocessor control architectures. In previous work [6], we demonstrated the tradeoff in performance between parallel robot control code with no processor fault tolerance and parallel code incorporating various levels of processor fault tolerance. We first developed a serial control program as a benchmark for analysis of three approaches to parallel robot control for a PUMA robot.

The first parallel controller that we developed had no tolerance of processor failures. This controller provides maximum speedup (given the algorithm used) because there is no software redundancy incorporated. For this PUMA controller, we chose the number of processors,  $n = 10$  to achieve a speedup of about 421% over the serial version.

The second controller considered incorporated total software redundancy. Each processor executes the exact same algorithm in parallel and compared results of some computations at various synchronization points. We used six processors operating in parallel for this controller (one per joint). This program dramatically exposed the time overhead of adding processor fault tolerance.

Finally, to demonstrate the concept of controller speedup and processor fault tolerance in cooperation, a third version was developed that can be considered a fault tolerant version of the first parallel approach. In this version, each processor was given a subset of the control equations, as in the first parallel program. However, some computation was performed on every processor and the result compared at chosen synchronization points. With fewer failure detection points, a failure may not be detected as quickly. Therefore, more work may have to be repeated by a working processor following the failure detection. Thus, this controller offers improved performance over the second approach, at the expense of using less information to detect a processor failure (a lower level of fault tolerance).

Using these programs to demonstrate our effectiveness measure yields the results shown in Table 3. For these examples:

$$f = (n - 1)/n$$

$$p = \begin{cases} 1/(80t) & \text{if } t \geq 12.5\text{ms} \\ 1.0 & \text{if } t < 12.5\text{ms} \end{cases}$$

where  $t$  is the controller cycle time. Recall that the performance rating is maximum (1.0) for cycle times less than 12.5ms. We do not include a measure of the recovery time  $R$  because the recovery time was not measured in [6].

	Num. Failed	Exec. time(ms)	$k_1$	$k_2$	$f$	$p$	$eff$
Serial		42.83	0	10	0	0.292	0.85
Parallel - no fault tol.		8.21	0	10	0	1.00	10.0
Parallel - high fault tol.	0	50.00	9	1	5/6	0.250	6.31
	1	50.83	9	1	4/5	0.246	5.82
Parallel - lower fault tol.	0	12.29	5	5	9/10	1.00	9.05
	1	12.43	5	5	8/9	1.00	8.95
	2	13.91	5	5	7/8	0.899	7.87

Table 3: Effectiveness measure for our previous work

### 3.4 Related Examples

In this section, we use our effectiveness measure to rate various robot control architecture examples from the literature (Table 4). The architectures chosen range from a uniprocessor system [5] to multiprocessors [1, 8, 11, 22]. All simulate control of a PUMA manipulator using Newton-Euler inverse dynamics.

Drake and Hsia [5] implemented two versions of the NE inverse dynamics algorithm on a DSP chip. One version includes optimizations afforded by the PUMA structure. The second algorithm is a more general version of the inverse dynamics. For the PUMA-specific routine, the execution time reported is 256  $\mu s$ . The general version is executed in 362  $\mu s$ .

Hashimoto et al. [8] proposed a parallel inverse dynamics algorithm and presented a multiprocessor architecture for its implementation. This algorithm eliminates the backward recursion that is required by the conventional NE inverse dynamics algorithm. The architecture consists of a host computer and multiple transputers for computation. The control algorithm was tested on a PUMA 560 for which the three wrist links were fixed on the third link. The computation time for the inverse dynamics was 464.2  $\mu s$  for this configuration. The sampling period for control of all six joints was estimated to be 0.8 ms for eight transputers and 1.4 ms when using four transputers.

Zalzala and Morris [22] presented a multiprocessor architecture for implementation of their on-line trajectory generator. This architecture employs multiple transputers for managing their algorithm, in addition to transputers performing calculations in parallel. For a four-transputer configuration, the execution time is 2.46 ms. With thirty-seven transputers, the execution time drops to 255  $\mu s$ .

Nigam and Lee [11] proposed a parallel pipeline architecture, which utilizes six processors for computation and a seventh for system supervision. The performance of this system using several different processors was reported. The best performance seen for the control computations was 0.6 ms. This was achieved on a MC68020 using integer arithmetic rather than floating point for increased speed. The authors indicated that the redundancy affords greater reliability, but they do not discuss fault tolerance in detail.

Ahmad and Li [1] presented a new scheduling algorithm for robot control tasks. The architecture chosen for this work consists of multiple arithmetic processing units connected to one host processor. This system was simulated with and without consideration of contention for access to the host processor. With contention modeled, their optimal time of 1.213 ms was achieved with six processing units.

The multiprocessor architectures described in these papers do not directly address fault tolerance. However, disregarding speed requirements, any task executed on multiple processors can be implemented on a single processor. Therefore, for our comparisons, we assign an importance of one ( $k_1 = 1$ ) to the fault tolerance ratings. For consistency, we assign an importance of one to the fault tolerance rating for the uniprocessor case as well. Some multiprocessor architectures utilize a system supervisor, or scheduler, or some other single point of failure. We only consider the redundant processing units for determining the fault tolerance rating. The number of processors used is indicated in parentheses in the table. Since no recovery times are available for these examples, we neglect that value. The performance rating will be mapped to a frequency range as it was in our previous example. The saturation frequency for these examples is taken to be 4 kHz.

Thus, for these examples:

$$f = (n - 1)/n$$

$$p = \begin{cases} 1.0 & \text{if } t < 0.25\text{ms} \\ 1/(3950t) - 0.013 & \text{if } 20\text{ms} > t \geq 0.25\text{ms} \\ 0.0 & \text{if } t \geq 20\text{ms} \end{cases}$$

yielding the results shown in Table 4 for the first case (when no failures have occurred).

	Exec. time(ms)	$k_1$	$k_2$	$f$	$p$	$eff$
Drake/Hsia (1)	0.256	1	9	0	0.976	8.573
Hashimoto et. al. (8)	0.8*	1	9	7/8	0.304	1.596
Hashimoto et. al. (4)	1.4*	1	9	3/4	0.168	0.817
Zalzala/Morris (37)	0.255	1	9	36/37	0.980	9.593
Zalzala/Morris (4)	2.46	1	9	3/4	0.090	0.636
Nigam/Lee (6)	0.6	1	9	5/6	0.409	2.202
Ahmad/Li (6)	1.213	1	9	5/6	0.196	1.040

Table 4: Effectiveness measure examples (\* estimates)

## 4 Discussion

In the discussion thus far, we have considered an effectiveness measure for specific subsystems of the overall robot system (the controller, joint construction, etc.). For these subsystems, individual fault tolerance sub-measures can be defined in a relatively straightforward fashion. However, in operational systems designed for fault tolerance, there will be numerous robot subsystems with redundant or safety systems. We would of course like to reflect the benefits (and costs) of each of these fault tolerance options in our overall effectiveness measure.

Similarly, performance/cost sub-measures  $p_i$  can be synthesized for the subsystems (notice that there may be a variety of different types of performance/cost modeled in the  $p_i$ , depending on the type of subsystem). These sub-measures could be found by using the techniques proposed in this paper or otherwise. The question arises as to how to combine the sub-measures for the various subsystems in the overall robot to obtain a meaningful effectiveness measure for the system.

In fact, there is a natural and straightforward way to combine these sub-measures which builds on established techniques already in use for robot fault tolerance analysis. We can use the well-understood technique of fault tree analysis [16], which graphically combines the logical effect of faults in subsystems and delineates their effect on the overall system [17]. Fault trees are often used in reliability analysis and are likely to be available for fault tolerant robot designs [17].

We utilize fault trees in two ways to analyze the fault tolerance of an architecture. The first approach considers each level of the tree separately, then combines the ratings as described below. The second method, which is discussed later, takes the component reliabilities into account. The difference between the two techniques is that the first is concerned with the amount and location of functional redundancy existing in the robot system, while the second uses cut sets to determine the probability of system failure (and no failure) occurring.

The logic in fault trees is typically displayed using AND and OR gates. We consider each in turn in the following, where we assume a fault tree has been constructed for the robot of interest. For subsystems combined via an AND gate, there is a degree of fault tolerance since all the subsystems must fail to pass a failure to the next level. The subsystems are basically providing redundancy, either through direct duplication of components (such as processors), or through functionally redundant components (such as having an encoder and resolver on the same joint). For the logic of AND gates, the method introduced in the previous sections (equation 2) can be applied to find a single fault tolerance sub-measure  $f_i$  (the AND gate represents the fact that there are backup systems).

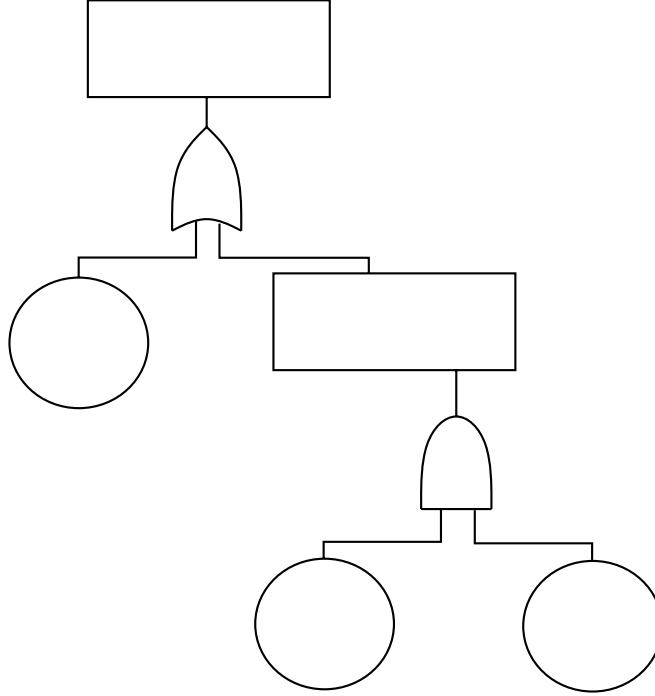


Figure 3: Robot joint with redundant sensors

The inputs to OR gates represent different functional parts of the robot system. At these gates, failure of any input subsystem propagates to the next level of the tree. There is no redundancy to protect from the failure. Thus the next (higher) level is dependent on any fault tolerance built into the subsystems entering into the OR gate, and measured by the individual  $f_i$ 's of the subsystems below. Therefore these  $f_i$ 's below must be combined to analyze the fault tolerance capability of the higher level (output of the OR gate).

The top event of the fault tree for one subsystem may be a primary event (bottom level) for another fault tree. By breaking the fault tree for the overall robot system into the component subsystem fault trees, we can analyze each of the sub-trees individually. Since the final fault tolerance rating is also required to be within the range  $[0, 1]$ , each subsystem rating is multiplied by a constant fraction. The value of the constant is determined by the level of the subsystem in the overall fault tree, such as  $c^{-i}$ , where  $i$  is the level of the subsystem in the overall tree. For example, the ratings of the actuator and sensor subsystems might be multiplied by  $2^{-3}$ , while the rating of the joint subsystem (at the next higher level) is multiplied by  $2^{-2}$ . Then the final fault tolerance rating for the entire robot system is calculated as the sum of the ratings for the subsystems, multiplied by  $2^{-1}$ .

As an example, consider a robot joint module with a single actuator and dual (identical) redundant sensors, as shown in Figure 3. The joint is said to have failed for this simple model if either the actuator or the sensor systems are inoperative. Thus 'joint failure' would be the output of an OR gate in a fault tree for this joint, with the inputs to the OR gate being 'actuator failure' and 'sensor failure'. Since there are redundant sensors, there is sensor fault tolerance which, using the approach detailed in Section 2, would be rated  $1/2$  (because one of the two sensors could fail and the unit continue). Since there is no actuator fault tolerance, the sub-measure  $f_a$  for the actuator system would be 0. Moving up to the next level (Joint Failure), the fault tolerance rating for the entire subsystem would be  $f = 2^{-i}(0 + 1/2)$  for the choice  $c = 2$ . Assuming that the fault tree in Figure 3 is a sub-tree of a larger fault tree,

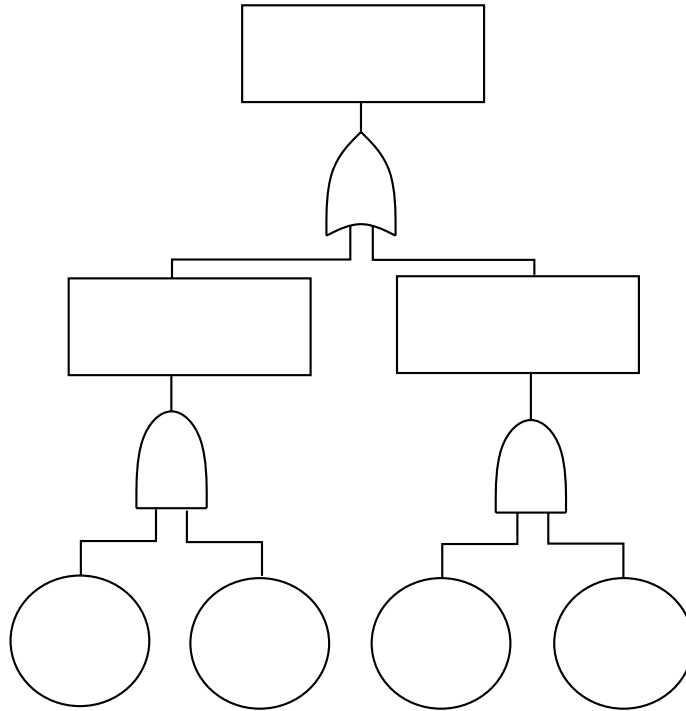


Figure 4: Robot joint with redundant sensors and actuators

$i$  might be 2 or greater for the joint level.

In the case of a subsystem that has both actuator and sensor redundancy, as shown in Figure 4, the fault tolerance rating at the joint level would be  $f = 2^{-i}(1/2 + 1/2)$  for  $c = 2$ .

In the above fashion, the fault tolerance sub-measures for individual subsystems can be analytically combined to achieve an overall fault tolerance measure for the complete system. The structure of the robot system is implicitly included in this analysis via the use of fault trees, which have already been used for robot reliability analysis [17].

The second approach to achieving a fault tolerance measure that was mentioned previously is to use the well-established theory of cut sets with component failure probabilities [16]. Basic events (primary failures) are those that are at the bottom level of the fault tree - not the result of other failures modeled in the tree. A cut set is a set of basic events that will cause the top event in the fault tree to occur.

We can use the relative importance of cut sets and those events which do not cause top event failure to derive a measure for fault tolerance as follows. First, all possible event combinations are determined, and the cut sets are separated from the sets that do not cause the top event to occur. Summing the probabilities of those sets that are not cut sets gives a probability that the subsystem failures will not occur - a fault tolerance measure that uses the reliability figures for the individual components.

For instance, using the example shown in Figure 3, let the actuator failure event be represented by 'A' and the sensor failure events be 'B' and 'C', respectively. The cut sets are: A, AB, AC, BC, ABC. The sets B and C do not result in joint failure, nor does the empty set. Using data for electric motors and optical encoders from the Nonelectronic Parts reliability data (NPRD-95) and based on 1000 hours of operation, the probabilities for the basic events are  $p(A) = 0.00924$ ,  $p(B) = p(C) = 0.0155$ . From these values, we can determine the probability of each set occurring, as shown in the table below. The sum of the probabilities of the first three sets (those that do not cause joint failure) is the fault

set	probability
$\emptyset$	$Pr\{\bar{A}\}Pr\{\bar{B}\}Pr\{\bar{C}\} = 0.96028$
$B$	$Pr\{\bar{A}\}Pr\{B\}Pr\{\bar{C}\} = 0.01512$
$C$	$Pr\{\bar{A}\}Pr\{\bar{B}\}Pr\{C\} = 0.01512$
$A$	$Pr\{A\}Pr\{\bar{B}\}Pr\{\bar{C}\} = 0.00896$
$AB$	$Pr\{A\}Pr\{B\}Pr\{\bar{C}\} = 0.00014$
$AC$	$Pr\{A\}Pr\{\bar{B}\}Pr\{C\} = 0.00014$
$BC$	$Pr\{\bar{A}\}Pr\{B\}Pr\{C\} = 0.00024$
$ABC$	$Pr\{A\}Pr\{B\}Pr\{C\} = 0.00000$

Table 5: Probability analysis

tolerance rating. For the example in Figure 3, the rating is 0.99052. In comparison, the fault tolerance rating for the example of Figure 4 is 0.99967.

The two methods presented for determining the fault tolerance rating of a system are complementary. The first method examines the structure of the system (amount and location of fault tolerance), while the latter looks at the reliability of the components used. This second method can be used to differentiate between systems with the same design but different component reliabilities.

## 5 Conclusions

As robots are chosen to perform more complicated, remote, and dangerous tasks, performance and fault tolerance issues take on increased importance. Therefore, rating a robot system based solely on one of these characteristics is no longer sufficient. The effectiveness metric presented in this paper combines performance/cost and fault tolerance ratings in chosen proportions in order to rank fault tolerant robot systems in a task-oriented manner. This approach can also be used to effectively rate architectures where no fault tolerance is incorporated.

The effectiveness measure could be very useful in helping to advance fault tolerance techniques in robotics, in that it provides a way to quantify the benefits of adding fault tolerance options to robots. It has the feature that an analyst can (by specifying  $k_1$  and  $k_2$  in (1)) tune the measure to reflect the degree of importance of fault tolerance versus performance/cost. In addition, if an engineer or system designer has specific measures for performance or cost, these can be used (as  $p$ ) within the effectiveness measure framework proposed in this paper to reflect the particular concerns of the application of interest. Thus the effectiveness measure framework proposed in this paper, while covering the spectrum of fault tolerance/performance/cost trade-offs, is general enough to allow tailoring for specific applications by those engineers with specific knowledge of the application.

To date no method of analysis exists that explicitly considers the fault tolerance/performance trade-offs for robotic systems. Any comparisons of fault tolerant robot systems must be somewhat subjective. Our approach removes some of the subjectivity by offering a method for rating both the fault tolerance and the performance of systems individually, then combining those ratings in a uniform manner. Since the sub-measures are combined in the same way

for all systems being analyzed, the resulting values can be compared.

## Acknowledgements

This work was supported in part by NASA Graduate Student Fellowship NGT-70251, in part by the National Science Foundation under grant #IRI-9526363, and in part by DOE Sandia National Laboratory Contract #AL-3017.

## References

- [1] S. Ahmad and B. Li. Robot Control Computation in Microprocessor Systems with Multiple Arithmetic Processors Using a Modified DF/IHS Scheduling Algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1167–1178, 1989.
- [2] M. Chean and J.A.B. Fortes. A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays. *Computer*, 23(1):55–69, January 1990.
- [3] K. Cleary and D. Tesar. Incorporating Multiple Criteria in the Operation of Redundant Manipulators. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 618–624, Cincinnati, OH, 1990.
- [4] B.R. Donald. *Error Detection and Recovery in Robotics*. Springer-Verlag, New York, 1989.
- [5] B.W. Drake and T.C. Hsia. Implementation of a Unified Robot Kinematics and Inverse Dynamics Algorithm on a DSP Chip. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1193–1199, Raleigh, NC, 1992.
- [6] D.L. Hamilton, J.K. Bennett, and I.D. Walker. Parallel Fault-Tolerant Robot Control. In *Proceedings of the 1992 SPIE Conference on Cooperative Intelligent Robotics in Space III*, pages 251–261, Boston, MA, 1992.
- [7] D.L. Hamilton, I.D. Walker, and J.K. Bennett. Fault Tolerance versus Performance Metrics for Robot Systems. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 3073–3080, Minneapolis, MN, 1996.
- [8] K. Hashimoto, K. Ohashi, and H. Kimura. An Implementation of a Parallel Algorithm for Real-Time Model-Based Control on a Network of Microprocessors. *International Journal of Robotics Research*, 9(6):37–47, 1990.
- [9] C.L. Lewis and A.A. Maciejewski. An Example of Failure Tolerant Operation of a Kinematically Redundant Manipulator. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 1380–1387, San Diego, CA, 1994.
- [10] V.P. Nelson. Fault-Tolerant Computing: Fundamental Concepts. *Computer*, 23(7):19–25, July 1990.
- [11] R. Nigam and C.S.G. Lee. A Multiprocessor-Based Controller for Control of Mechanical Manipulators. *IEEE Journal of Robotics and Automation*, RA-1(4):173–182, 1985.
- [12] C.J. Paredis and P.K. Khosla. Mapping Tasks into Fault Tolerant Manipulators. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 696–703, San Diego, CA, 1994.

- [13] D. Sreevijayan, S. Tosunoglu, and D. Tesar. Architectures for Fault-Tolerant Mechanical Systems. In *Proceedings of the 7th IEEE Mediterranean Electrotechnical Conference (MELECON)*, pages 1029–1033, Antalya, Turkey, 1994.
- [14] Y. Ting, S. Tosunoglu, and D. Tesar. A Control Structure for Fault Tolerant Operation of Robotic Manipulators. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 684–690, Atlanta, GA, 1993.
- [15] K.S. Tso, M. Hecht, and N.I. Marzwell. Fault-Tolerant Robotic System for Critical Applications. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 691–696, Atlanta, GA, 1993.
- [16] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasi. *Fault Tree Handbook*. NUREG 0492, Systems and Reliability Research Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, D.C., 1981.
- [17] M.L. Visinsky, J.R. Cavallaro, and I.D. Walker. Robotic Fault Detection and Fault Tolerance: A Survey. *Reliability Engineering and System Safety*, 46(4):139–158, 1994.
- [18] M.L. Visinsky, J.R. Cavallaro, and I.D. Walker. A Dynamic Fault Tolerance Framework for Remote Robots. *IEEE Transactions on Robotics and Automation*, 11(4):477–491, 1995.
- [19] F. Wang, K. Ramamritham, and J.A. Stankovic. Determining Redundancy Levels for Fault Tolerant Real-Time Systems. *IEEE Transactions on Computers*, 44(2):292–301, 1995.
- [20] T.S. Wikman, M.S. Branicky, and W.S. Newman. Reflex Collision Avoidance: A Generalized Approach. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 31–36, Atlanta, GA, 1993.
- [21] J. Wunnenberg and P.M. Frank. Dynamic Model Based Incipient Fault Detection Concept for Robots. In *Proceedings of the 1990 IFAC World Congress*, pages 61–66, Tallinn, Estonia, 1990.
- [22] A.M.S. Zalzala and A.S. Morris. Distributed Robot Control on Transputer Network. *1991 IEE Proceedings-Part E, Comput. Digit. Tech. (UK)*, 138(4):169–176, July 1991.