VOTING WITH DYNAMIC THRESHOLD VALUES FOR REAL-TIME FAULT TOLERANT CONTROL SYSTEMS

G. Latif-Shabgahi[+], M. O. Tokhi*, and M. Taghvaei[*]

[+] *ICT Dept, Technology Faculty, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK*
**Automatic Control and Systems Eng. Dept., The University of Sheffield, Mappin St., Sheffield, S1 3JD, UK*
(*Contact Author: g.r.latif@open.ac.uk*)

Abstract: Voting on the outputs of redundant modules with real number results (i.e., where correctly functioning redundant systems may arrive at slightly different yet correct outputs for identical inputs) is not straightforward. Such cases need inexact voting in which some discrepancies between the outputs of redundant modules are allowed. Documented inexact voters use a fixed threshold value that may cause problems in safety-critical systems. This paper introduces a voting algorithm with dynamic threshold value. The voter is implemented, and compared with its counterpart with a fixed threshold value. The experimental results show that the novel voter gives more correct yet less incorrect results than the conventional majority voter with a fixed threshold value in the examined error scenarios.

Key Words. Safety-Critical Systems, Fault Tolerance, Redundancy Control, Threshold, Reliability.

1. INTRODUCTION

Increasing reliability and safety is one of the primary concerns in many real-time systems. Examples include safety-critical systems (e.g., flight control, nuclear power plant control, and medical equipment such as blood gas analyser used in intensive care units), highly reliable applications (e.g., railway-interlocking system, and telecom switch), and distributed systems (where Byzantine agreement and clock synchronisation is required). Certain critical parts of such applications must be able to operate under faulty conditions. Such applications use redundancy to reduce the risk associated with relying upon any single component operating flawlessly. Triple Modular Redundancy, TMR, and 3-Version Programming, 3VP, are commonly-used techniques for masking faults/errors at hardware and software levels, respectively (Lala and Harper, 1994). The outputs from three identical modules (in general, an odd number of modules) operating in parallel with the same inputs are supplied to a voting module that arbitrates between them to produce a single output. The system, therefore, continues its predefined function even in the presence of some faults/errors. A voting module specifies how the voting result is obtained from the output of multiple modules and can be the basis for implementing a hardware voting network (see, for example, (Kim *et al*., 2002)) or a software voting routine. Various voting techniques have been used in fault tolerant systems; the most common of these are majority, plurality, median and weighted average voters. These voting schemes are indistinguishable when there is only one faulty module; however, they behave differently in the presence of multiple-module failures. In the context of voting, we are encountered to *exact* and *inexact* voting. In exact (bit-wise) voting, agreement means that redundant results are exactly the same; thus a 5-input exact majority voter produces an output when 3-out-of-5 of its inputs are equal. In inexact voting, some discrepancy between the inputs is allowed; agreement now means that the redundant results are not exactly the same, but the difference between

them is less than a particular threshold. The value of this threshold is application specific. Unfortunately, there is no analytical approach for setting the value of voting threshold, most designers use heuristics and the characteristics of the application to set the value. Moreover, the implementation of an inexact voter in the hardware domain is not straightforward in itself (Quintana *et al*., 2000).

1.1. Related Work

Exact voting on the results of redundant modules with real number outputs is not appropriate. For data derived directly from noisy sources, for analogue sensor outputs which are read by digital computers, or for the output of diversely implemented software programs (Avizienis and Kelly, 1984) which handle floating point arithmetic an exact match is generally impossible due to quantisation and/or rounding errors. Dealing with these cases needs *inexact (threshold)* voting. A number of inexact voting algorithms have been studied in the literature. Examples are formalised majority and plurality voters (Lorczak *et al.,* 1989), Predictor voters (Latif *et al*., 2002), Smoothing voter (Latif *et al*., 1998), and Maximum Likelihood Voter (Kim *et al*., 1998). Formalised majority and plurality voters are extensions of their exact counterparts to handle disagreed yet correct voter inputs. The predictor and smoothing voters extend the capability of the inexact majority voter for handling complete disagreement voting cases. They predict a value, based on the history record of the voter previous outputs, as the voter output in cases of complete disagreement between the redundant module results. Maximum likelihood voter has been suggested for multi-version software with finite output space, under assumption of failure independence. To estimate the correct result, it uses the reliability of each software module and determines the most likely correct module result. In inexact voting, the selection/generation of the voter output out of/from the agreed values is also important. Strategies like random selection (Lorczak *et al*., 1989), averaging, weighted averaging (Parhami, 1994), and mid-value selection have been used in practice. Recently, a fuzzy approach has been suggested by Kim *et al* (1998) that alleviates this problem by forming a fuzzy equivalence relation.

All aforementioned voters use a fixed threshold value for reaching an inexact consensus. Knowing the bounds on the normal deviation between the results of redundant modules for a system's entire operational time allows the design of inexact voters with a fixed threshold value. However, use of such voters at the control level of safety-critical systems is problematic for several reasons: i) the selection of the threshold is critical; ii) some acceptable module results may be ignored when using a fixed threshold; and iii) voters with fixed threshold values are unable to vary their response in the face of different levels of disagreement in phased-mission and performance degradable systems. Soft threshold voter has been introduced in (Latif *et al*., 2003) for smoothing these problems. Instead of using a fixed threshold value for dividing the module results into '*agreed'* and '*disagreed'* groups and then obtaining a value for the voter output from the agreed values, it assigns a real value in the range 0.0 to 1.0 for any pair of module results from which the weighting value of any module result toward the voter output is obtained. This real value, in fact, expresses the '*degree of agreement'* of any two module results, and can be controlled to allow the user to change the behaviour of the soft voter between that of the two baseline majority and distance-based weighted average voters.

This paper introduces the concept of '*dynamic threshold'* for inexact voters. In a voter with dynamic threshold, the value of the threshold is determined based on a system's operational mode. More precisely, the *status of the system*, the trajectory of *input data* (the *output of modules = input data* to the voter), *task criticality*, and *input data values* are used to set a value for voting threshold in each operational mode. The organisation of this paper is as follows. Section 2 explains the needs for using dynamic threshold in inexact voting. Section 3 describes the experimental test harness and methodology. In section 4, the comparative safety and availability performance of the inexact majority voter with a fixed and dynamic threshold values are investigated. Finally, some conclusions are given in section 5.

2. INEXACT VOTING WITH DYNAMIC THRESHOLD

As mentioned in section 1, an inexact voter with a fixed threshold value may cause problems in many real time control systems. In multi-state safety-critical systems some of the operational modes are more critical than the others; in a flight control system, for example, take-off and landing modes are more fault/error-prone than the ascending, descending, and cruising modes. Thus the fault tolerant mechanism (and its related adjustments) used for high-critical operational modes must differ from that of the less-critical modes. In the case of using a TMR fault masking strategy, the former modes need a voting algorithm with a carefully selected threshold value (small enough to ensure that incorrect redundant module results cannot contribute toward voting) whereas the latter modes are likely to work correctly with a larger threshold value (to ensure that acceptable variant results are not discarded from voting). The use of state-based voting threshold values is also dictated by the range of data created by multiple redundant modules. Suppose that in the operational state A, the voter is faced with data from the interval [1 5], and in state B it is confronted with data from the interval [100 150]. Here, arbitrating between redundant data from the two

distinct intervals with an identical threshold value (e.g., 1.0) is questionable. Obviously, judging between redundant small numbers needs a smaller threshold value than arbitrating between the redundant large real numbers. That is, for state A the voter inputs *{1 2 3}* (with the deviation of 1.0 from each other) are more likely considered in disagreement whereas for state B, voter inputs {120, 121, 122} with the same deviation are considered in agreement. This is the basis for choosing a threshold value for the novel voter. The voter threshold is proportional to the expected numerical values of its inputs. The proportional coefficient is an application-specific parameter.

The internal structure of a voter with an adaptive threshold value is shown in Figure 1. Knowing the operational mode of the system and the range of values expected for that mode enables the designer to assign a voting threshold value for that mode.

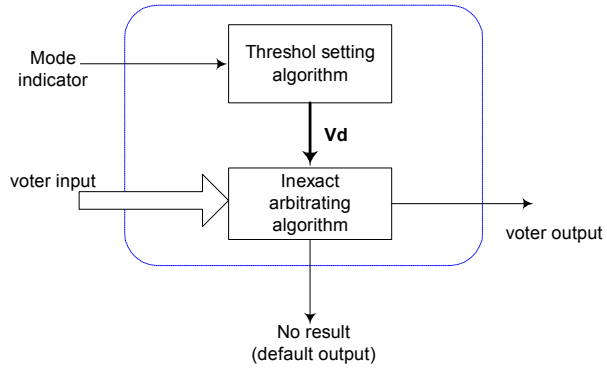

Fig. 1. An inexact voter with a dynamic threshold

The structure is clarified by using a hypothetical flight control example shown in Table 1. In this example, it is assumed that in high-critical modes (take-off and landing) the voter is encountered with small input values (from the range [1 5]), consequently a small threshold value has been set for this mode ($\frac{1}{10}.x_{e_{\max}} = 0.5$, where $x_{e_{\max}}$ is the upper band of the range). For less-critical modes (e.g., cruising mode) the voter is encountered with large numbers (from the interval [10 20]), and therefore, a large threshold vale is chosen ($\frac{1}{7}.x_{e_{\max}} = 2.85$).

Table 2 shows the comparative outputs of an inexact majority voter with dynamic threshold values (set based on Table 1) with those of a voter with a fixed threshold value (=0.6) for a stream of inputs. The first column shows the sample number in brackets as well as the notional correct output of the voter in that sample, the next three columns indicate saboteurs' perturbed outputs (voter inputs), the fifth column is the output of the fixed threshold majority voter, and the last column is the output of the dynamic threshold majority voter.

Table 1. An example for setting a voter threshold

Operational mode	Mode indicator	Range of voter Outputs	Voter threshold
Take-off	A	$0< x_e \leq 5$	$\frac{1}{10}.x_{e_{\max}}$
Ascend	B	$5< x_e \leq 10$	$\frac{1}{8}.x_{e_{\max}}$
Descend	B	$5< x_e \leq 10$	$\frac{1}{8}.x_{e_{\max}}$
Cruise	C	$10< x_e \leq 20$	$\frac{1}{7}.x_{e_{\max}}$
Landing	A	$0< x_e \leq 5$	$\frac{1}{10}.x_{e_{\max}}$

For samples [1] to [4] in which the voter functions in the mode A, and the inputs are taken from the range [1 5], the dynamic threshold is 0.5 (=1/10*5). In the first and second samples, both voters have reached agreement between the variants, and the result of the second variant has been selected as output. For the third sample, the fixed threshold voter gives an output (correct output) but the dynamic threshold voter produces no output. In this case, the selection of a small value for dynamic threshold results in discarding the good variant results (i.e., 2.8 and 3.4) from being voted; this is, in fact, the disadvantage of choosing a small value for voting threshold. Both of the voters give *no-output* for the fourth sample.

Table 2. Outputs of majority voters with fixed threshold (Fx-thr) & dynamic threshold (Dy-thr) values for 12 cases

exp. output		var-1	var-2	var-3	Fx-thr	Dy-thr
[1]	1	*1*	*1.2*	*1.6*	1.2	**1.2**
[2]	2	*1.8*	*2.1*	*2.9*	2.1	**2.1**
[3]	3	*2.8*	*3.4*	*4*	3.4	**---**
[4]	4	*4.7*	*4*	*3.2*	---	**---**
[5]	6	*6*	*6.5*	*7*	6.5	**6.5**
[6]	7	*7*	*7.8*	*6.2*	---	**7**
[7]	8	*8.9*	*8*	*10*	---	**8.9**
[8]	9	*9*	*10.5*	*7.5*	---	**---**
[9]	10	*10*	*10.4*	*10.7*	10.4	**10.4**
[10]	12	*12*	*12.9*	*13.8*	---	**12.9**
[11]	13	*13*	*10*	*16*	---	**---**
[12]	20	*18*	*19*	*20*	---	**19**

In samples [5] to [9] the value of dynamic threshold is set to 1.25 (=1/8*10), and the voter works in mode B. In sample [5] both of the voters give an (correct) output. In samples [6] and [7] the fixed threshold voter gives no outputs whereas the dynamic threshold voter produces an (correct) output. These two cases indicate the benefits of using an appropriate dynamic threshold value (as well as the problem of choosing a small fixed threshold value) for an inexact voter; the fixed threshold voter has discarded good values from voting. Both of the voters give no output for the next

sample. In samples [10]-[12] the voters function in mode C, and, hence, the value of the dynamic threshold is set to 2.85 (=20/7). The fixed threshold voter produces no result for these samples, whereas the dynamic threshold voter gives correct outputs for samples [10] and [12].

3. TEST HARNESS FRAMEWORK

The experimental test harness is shown in Figure 2. The input generator produces one notional correct result in each voting cycle. This sequence of numbers identical correct results expected from redundant modules. Copies of the notional correct result are presented to each saboteur in every voting cycle. The saboteurs can be programmed to introduce selected module error amplitudes, according to selected random distributions. The symptom of errors appears to the voter as numerical input values. A comparator is used to check for agreement between the notional correct result and the output of the voter under test at any voting cycle. However, for simplicity, issues associated with ensuring synchronisation of the inputs to the voter and to the saboteurs are ignored.

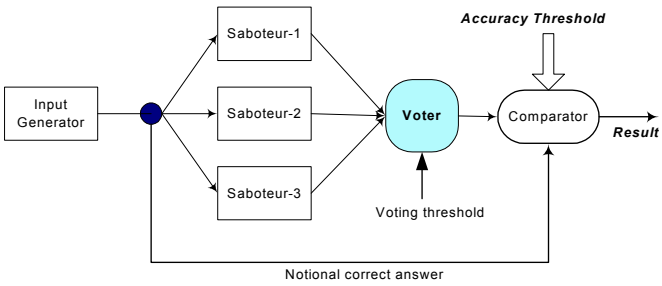

Fig. 2. Experimental harnesses

A *voter threshold (*dynamic or fixed*), VT,* is used to determine the maximum acceptable divergence of voter inputs in each voting cycle from the notional correct result, and an *accuracy threshold*, *AT*, is used in comparator to determine if the distance between the notional correct result and the voter output is within acceptable limits. In this framework, the accuracy threshold is chosen equal to the voter threshold in each voting cycle. A voter result which has a distance from the notional correct answer less than the accuracy threshold is taken as a correct output, otherwise it is considered as an incorrect output. This is a valid assumption in a many real-time systems in which the discontinuity between consecutive correct variant results is small (Bennett, 1994). Hence, the presence a large discontinuity is indicative an error and can be detected by the acceptance tests. Where the voter cannot reach an agreement between the outputs of saboteurs, it produces a of default value that moves the system toward a fail-safe or fail-stop state. Such voter output is called a disagreed (benign) result. It is also assumed that all voters perform correctly. This assumption is made due to the fact that the voting algorithm is usually a simpler program than the modules it monitors. Figure 3 indicates the classification of a voter outputs in this test harness. It is obvious that, from the viewpoint of system safety, agreed–correct results are of interest whereas the agreed-incorrect results are dangerous and catastrophic outputs.

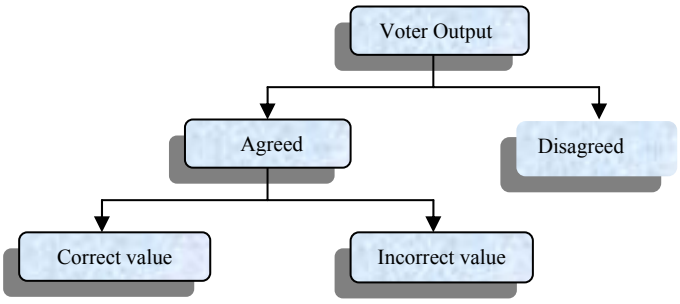

Fig. 3. Voter output classification

3.1. Experimental Method

A stream of input data with the sinusoidal profile: *u=50. sin (t)+100* sampled at 0.1 sec feed both the saboteurs and comparator. For inexact majority voter with a fixed threshold value, the threshold is set to 0.5, and for the majority voter with dynamic threshold values the setting $VT = \frac{1}{100}.x$ (*x* is the value of input data at each voting cycle) is made. The former setting is the simplified form of the threshold setting mechanism discussed in section 2. However, the accuracy threshold is always set to a value equal to the voter threshold. Random errors with uniform distribution from the interval $[-e_{max} \; +e_{max}]$ are injected into the all saboteurs to simulate modules errors. e_{max} is selected 2. This injection simulates the effects of permanent errors in the system. The outputs of saboteurs are presented to the voter under test. In every voting cycle the output of the voter, *y,* is compared with a copy of input data x_o. Based on the numerical distance between *y* and x_o values, the output of the voter is interpreted as correct, incorrect, or disagreed value. For each voter the results of 10^4 *(= n)* system runs are classified. In this way, n_c correct results, n_{ic} incorrect outputs and n_b disagreed results are collected. It is obvious that for all voters $n_c + n_{ic} + n_b = n$ and for weighted average $n_b = 0$. These data are, then, used for evaluation and comparison of voters. Two performance measures are defined for this purpose: *safety* and *reliability*.

1. *Safety (S)*: Since from a safety viewpoint the smallest number of agreed but incorrect outputs is desirable for a given voter, the safety measure

can be defined as: $S = (1-n_{ic} / n)$. Thus $S \in [0\ 1]$ and ideally $S=1$.

2. *Reliability (R)*: A voter which produces more correct results among its total outputs can be interpreted as more reliable voter. Reliability is defined as the ratio of correct voter outputs to the number of voting actions: $R = n_c / n$. Thus $R \in [0\ \ 1]$ and ideally $R=1$.

Each performance criterion can be plotted versus a parameter of the test harness such as error amplitude, accuracy threshold or versus a voter parameter such as voter threshold. In this framework, the safety and reliability performance of voters versus the size of injected errors are examined.

4. EXPERIMENTAL RESULTS

The safety and reliability performance of two versions of the inexact majority voter (one with a fixed threshold value, and the other with dynamic threshold values) are examined in this section versus the size of injected errors.

4.1. Experiment 1: Comparing the Reliability and Safety of 3-input Voters

Figures 4 shows the plot of safety and reliability of 3-input voters versus error amplitude, e_{max}. In the examined error scenarios, the voter with dynamic threshold gives higher safety (less incorrect outputs) and higher reliability (more correct outputs) than the voter with a fixed threshold value. For example, with $e_{max}=1$, the dynamic threshold voter has a 23% better safety and about 27% higher reliability performance than the fixed-threshold voter.

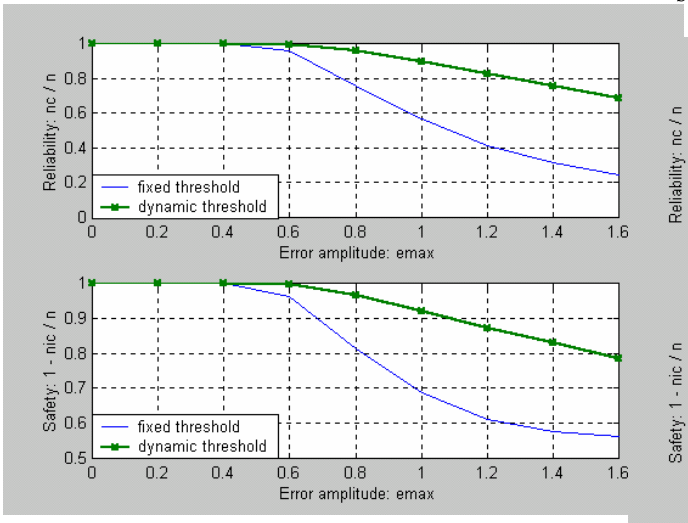

Fig. 4. Reliability and Safety of 3-input voters versus error amplitude

Figure 5 indicates, more clearly, the superiority of the dynamic threshold voter (in terms of the number of correct, incorrect, and agreed outputs) in the error point $e_{max}=0.8$.

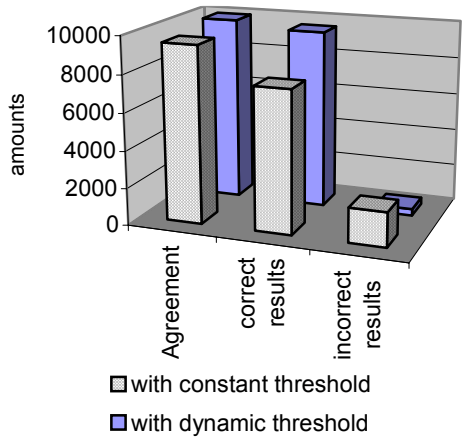

Fig. 5. Outputs of 3-input voters when $e_{max}=0.8$ for 10^4 runs

4. 2. Experiment 2: Comparing the Reliability and Safety for 5-input Voters

This experiment shows the comparative safety and reliability performance of 5-input dynamic and fixed threshold voters versus the amplitude of injected errors. Figure 6 indicates the results. Firstly, comparing this figure with figure 4 shows the superiority of the 5-input voters to their counterpart 3-input versions in terms of safety. The 5-input dynamic threshold voter, for example, gives higher safety (less incorrect outputs) than the 3-input dynamic threshold voter for all error scenarios. However, such safety improvement is achieved at the cost of decreasing their reliability performance as seen from the reliability plots in figures 4 and 6.

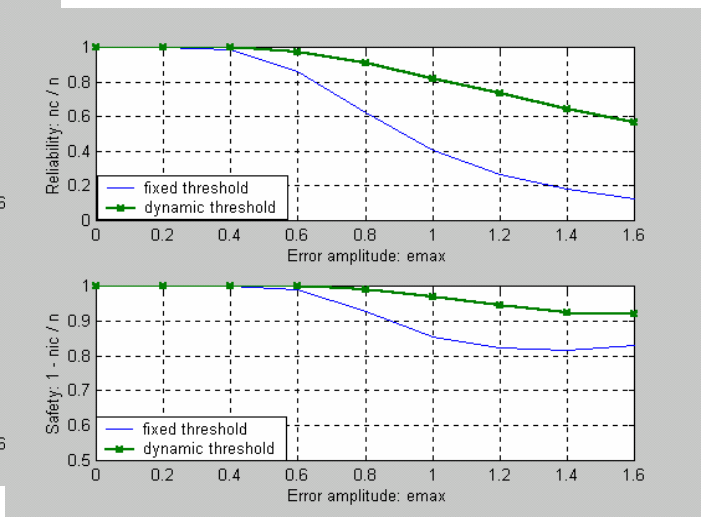

Fig. 6. Reliability and Safety of 5-input voters versus error amplitude

That is, the reliability of the 5-input voter (with dynamic or fixed threshold) is less than that of its 3-input counterpart voter. Secondly, Figure 6 also shows that the 5-input voter with dynamic threshold values gives higher safety and reliability than the 5-input voter with a fixed threshold value.

Figure 7 compares the number of correct, incorrect, and agreed outputs of the examined 5-input voters at error point $e_{max}=0.8$. The dynamic threshold voter is superior to the fixed threshold voter, and this superiority is much better than that of the 3-input voters (when comparing figures 5 and 7).

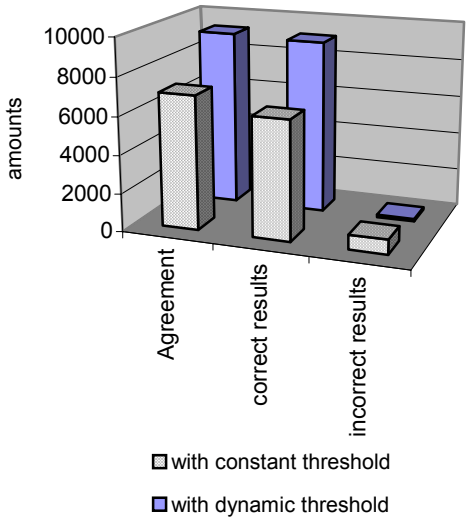

Fig. 7. Outputs of 5-input voters with $e_{max}=0.8$ for 10^4 runs

5. CONCLUSIONS

In using NMR or N-version programming systems, it is possible that correct modules may arrive at slightly different yet correct outputs for an identical input. Inexact voting is used to deal with this problem. An important requirement of an inexact voter is to choose the threshold value by which the consensus of inputs is examined. Since there is no mathematical way of determining the threshold value, most designers use heuristics and the characteristics of the application to set the value. The paper introduced a scheme for inexact voting with dynamic threshold suitable for real-time systems with different safety-critical modes. The performance of the inexact majority voter with a fixed threshold value has been compared with that of the newly introduced inexact majority voter with dynamic threshold values. The focus was on the number of correct and incorrect outputs of voters after *n* voting action in a fault injection environment. Two performance criteria, safety and reliability, were defined, and the behaviour of voters was examined in the presence of permanent errors. The experimental results showed that the reliability and safety of 3 and 5-input majority voters with dynamic threshold values are higher than those with fixed threshold values. The experimental results also showed that the safety of the 5-input majority voter is always higher than that of the 3-input majority voter, however, the reliability of 5-input majority voter is lower than that of the 3-input voter.

REFERENCES

Avizienis, A., and Kelly, J. P. (1984). Fault-tolerance by design diversity. *IEEE Computer Magazine,* **17**(7), 67-80.

Bennett, S. (1994). *Real-Time Computer Control-An Introduction.* 2nd Edition, Prentice-Hall Int. (UK).

Kim H, Jeon H. J., Lee K, and Lee H. (2002). The design and evaluation of all voting triple modular redundancy system. *Proc. Ann. Reliability and Maintainability Symp.*, 438-444.

Kim, K., Vouk, M. A., and McAllister, D. F. (1998). Fault tolerant software voters based on fuzzy equivalence relations. *Proc. IEEE Aerospace Conference*, **4**, 5-19.

Lala, J. H. and Harper, R. E. (1994). Architectural principles for safety-critical real-time applications. *Proc. of the IEEE*, **82**(1), 25-39.

Latif-Shabgahi, G., Bass, J. M., and Bennett, S. (1998). Complete Disagreement in redundant real-Time control applications. *Proc. 5th IFAC Workshop on Algorithms and Architectures for Real-Time Control,* AARTC'98, Cancun, Mexico, April 15-17, 259-264.

Latif-Shabgahi, G., S. Bennett, and J. M. Bass (2002). Voting algorithms in multiple error scenarios for real-time control applications, *Proc. IFAC B'02 World Congress*, July 21-26, Barcelona, Spain.

Latif-Shabgahi, G., Bennett, S., Hirst, A. J., and De Leon Martinez, A. D. (2003). Soft threshold voting scheme for safety-critical computer control applications", *Proc. of WRTP'03: 27th IFAC/IEEE Workshop on Real-Time Programming*, 14-17 May, Lagow, Poland.

Leung, Y. W. (1995). Maximum likelihood voting for fault tolerant software fault finite output space. *IEEE Trans. on Reliability*, **44**(3), 419-427.

Lorczak, P. R., Caglayan, A. K., and Eckhardt, D. E. (1989). A theoretical investigation of generalised voters. *Proc. FTCS'19: IEEE 19th Ann. Int. Symp. on Fault- Tolerant Computing Systems,* Chicago, IL, 444-451.

Parhami, B. (1994). Voting algorithms. *IEEE Trans. on Reliability*, **43**(4), 617-629.

Quintana, J.M., Avedillo, M.J., Rodriguez-Villegas, E., and Rueda, A. (2000). Efficient /spl nu/MOS realization of threshold voters for self-purging redundancy", *Proc. of 13th Symp. on Integrated Circuits and Systems Design*, Sept. 18-24, Manaus, Brazil.