Model implementation of the simulation environment of voting algorithms, as a dynamic system for increasing the reliability of the control complex of autonomous unmanned objects

Igor Kovalev[1], *Vasiliy* Losev[1], *Mikhail* Saramud[1,*], and *Mariam* Petrosyan[1]

[1]Reshetnev Siberian State University of Science and Technology, Russian Federation

Abstract. In the article the question of application of redundant schemes for the development of fault-tolerant software, in particular multiversion programming, is considered with reference to the task increasing the reliability of the control complex of autonomous unmanned objects. Modifications of the basic voting algorithms by an agreed majority and its fuzzy version are proposed. The software implementation of the simulation environment that implements the described approach and the proposed modified algorithms is considered. The results of the simulation the dependence of the reliability indicators of the system on its input parameters are shown by authors.

1 Introduction

With the development of robotics in recent years, autonomous unmanned objects, including unmanned aerial vehicles (UAV), have become increasingly important. The impetus for this is the development of the technical base - the emergence of compact controllers with higher processing power, accurate sensors, cameras with high resolution, and the emergence of the market for finished products. For example, a circular laser scanner (lidar) is currently an affordable consumer product and number of manufacturers is already on the market. At the same time, electronic components are becoming more compact and energy-efficient every year - they consume less electrical power and produce less heat, thanks to the transition of semiconductor manufacturers to a more subtle process technology and the improvement of microarchitecture of chips. The second factor is the development of the algorithmic base, autonomous management has been an urgent task for several years already, and this direction is actively developing, new algorithms, methods and ready-to-use software solutions are emerging. One of the most important tasks in autonomous control is image recognition, since it is the output of the pattern recognition unit that is the main source of information about the world for the system, based on which a decision is made about the behavior of the entire object. There is also active development of unmanned vehicles, for which the correct recognition of road signs and markings is critical, which, in turn, are not always in favorable conditions for recognition (pollution, installation in different positions, at different angles to the roadway, partial overlapping). The faults in the recognition of images can lead to serious negative consequences, including the destruction of the controlled object; therefore, it is necessary to minimize the possibility of faults, thereby maximizing the reliability of the system. In recent years, many different algorithmic approaches developed, as well as hardware solutions for machine vision and pattern recognition. The presence of a variety of different image recognition algorithms, allows you to develop a redundant system that includes several versions of software modules with different algorithms. These modules, based on the same or different hardware capabilities (different cameras, lidars, range finders, pyrometers, etc.) make a decision in the classification of the image, and the execution environment makes the final decision based on the outputs of these versions. In this case, expediently in the similar system uses the multiversion programming scheme (MVP) with versions that perform the same functionality, but with considering different methods and a decision block that chooses the "correct" output. This block makes a decision on the outputs correctness based on the voting algorithms: the voting by an absolute majority, agreed by the majority, their fuzzy and weighted modifications, median voting.

2 Voting Algorithms

For improving of voting algorithms stability to the related faults of program modules [1], in comparison with unweighted analogs, are suggested the modifications of the basic voting algorithms agreed by a majority and a fuzzy vote by an agreed majority [2].

For each software version in a system creates the Boolean stack, of fixed length, in which “0” is added in the event, that the voting block decides that the version gives the wrong answer and “1” if it gives the right answer or in case of a fuzzy vote if the value of the version's output belongs to the winning class > 0, i.e. all

* Corresponding author: msaramud@gmail.com

versions that have added weight to the class that won the vote will be marked as correct.

As the stack length is fixed, new data will replaces the most oldest, i.e. the stack operates according to the principle FIFO (if the stack comes first, the stack will leave first). This action lets to input the forgetting element to the depth of a stack. For example, if the depth of a stack is 100 to the results of a work version are older that for 100 voices and won`t be counted.

The forgetting element is necessary to ensure the operative response of the system to changes in the behavior of versions when a certain version can significantly change its reliability in case of changing the input data stream [3], therefore is necessary to change rapidly its rating of version for correct distribution of the weights during voting [4]. The rating is determined by summing of all the elements of the stack. For example, if in stack of 100 elements length 97 values of "1" and 3 values of "0" are recorded, the weight of this version will be equal to 0.97.

We have made only one restriction with the software implementation; the version weight cannot be equal to one, what can happen in practice. Quite reliable enough versions give the correct answer 100, 1000, 10000, etc. times in a row. And without a limit they would get the whole stack of units (or TRUE), which would give them a rating of one, but these situations should not be allowed, since in case of an incorrect answer by this version, it would get a weight of one. Although the right answer of remaining N-1 versions on a weight only approaches one and as a result loses the vote. If a reliability rating of the version is equal to one, it isn`t make sense, because if we have a reliable software module, the sense of the system will be lost [5]. Thus, we have made restrictions in calculating the rating of the version with each vote.

2.1 Software implementation of the simulation environment

When voted by an agreed majority, the voting block receives responses from each version (or in our program is simulation). If the output value doesn't match the previously obtained one, a new class is created, if the value matches the existing class [6-7], then this class weight will be recalculated as

$$P_{total} = P_{class} + (1 - P_{class}) * P_{version} \quad (1)$$

After all the versions gave an answer (response time limit in this model is absent [8]), the weights of the resulting classes are compared. Thus, the class with the greatest weight wins, but as it is seen, is not always a class, which the largest number of versions "voted". The weight of each version has a value, which allows considering the reliability of each separate version separately [9].

After determining the correct exit version of the version, those who voted for it get a weight "1" on the stack, and the versions voted "0" differently.

In case of fuzzy voting by an agreed majority, while voting one more pass occurs in the program when versions whose output value have not been matched the class value. This value is different for it no more than the tolerance "E" and, in this way, it belongs to the class is equally >0 that adds weight to the class:

$$P_{total} = P_{class} + (1 - P_{class}) * P_{version} * K_{implement} \quad (2)$$

where

$$K_{implement} = 1 - \left(\left| X_{class} - \frac{X_{version}}{E} \right| \right) \quad (3)$$

X - Class value and version, which gave non-matching answer, but a significance that enters the tolerance E [10].

In this studied system, the version simulations give three types of faults: a random fault, a simulating fault in the module, as related fault, and an inaccuracy to the answer is close to the correct one, remote from it is no more than the tolerance, but not equal to it. This type of a fault simulates "inaccuracy" is connected with rounding faults with a lack of bit capacity, inaccurate digitization of analog sensor outputs, etc. We take into account the situation when the algorithm version worked correctly, but it gave an inaccurate response due to rounding faults, digitization, etc. [11] The fault appears with the probability given for each version and each data stream. In the event of a fault, the following checks occur: if it's not first fault in current voting, in this case an related fault will be generated with predetermined probability, i.e. a value matching the value of the past fault returns.

Further, an "inaccuracy" is generated with a given the probability and the output is not equal to right has been returned, but remote from it by no more than a given deviation of E. If the previous probabilities do not work, then a random fault will returned, simulating a malfunction in the current version of the module.

From the results of the program, it is seen that in the absence of "inaccuracies", there is no difference in the voting methods; a random fault is always too "far" from the correct answer to change the weight of classes [12]. Thus, we can conclude, that if a system does not have possible places of occurrence of "inaccuracies" that is digitization of analog signals, lack of digit capacity for the mathematical operations, etc. then using of a more resource-intensive fuzzy voting algorithm will not give advantages [13].

However, if the possibility of such "inaccuracies" is present that the fuzzy algorithm will increase the reliability of a system. The only drawback will be the same evaluation for both the versions giving the ideally correct output, and the versions algorithmically correct, but having "inaccuracies".

The developed system stimulates the multiversions separate function that returns with in advance set probabilities either the correct answer or numerical result of one of three types of mistakes to "inaccuracy", "failure" and "related".

These probabilities are specified by the program for 3-successively varying input data streams, allowing to model different versions of the response to a change in the input data stream. After a certain iteration, the probability distribution changes, and it is possible to track the system's response to this [14], check the correctness of the operation, the reaction rate depending on the algorithm chosen, the stack length and other system parameters.

The system permits to change the version numbers N in the range from three to night, the probability P of each version for 3 different data streams, the depth of the weight stack of versions. It influences the speed of the system response to the change in version work, set the number of iterations for each thread, the probability of the related faults, inaccuracies, and the E tolerance for a fuzzy voting algorithm. As a result of the simulation, the system gives the number of faults allowed by each algorithm, for fuzzy voting, the "failures" and "inaccuracies" are calculated separately, the sum of the weights of the winning voting classes and the number of version related faults.

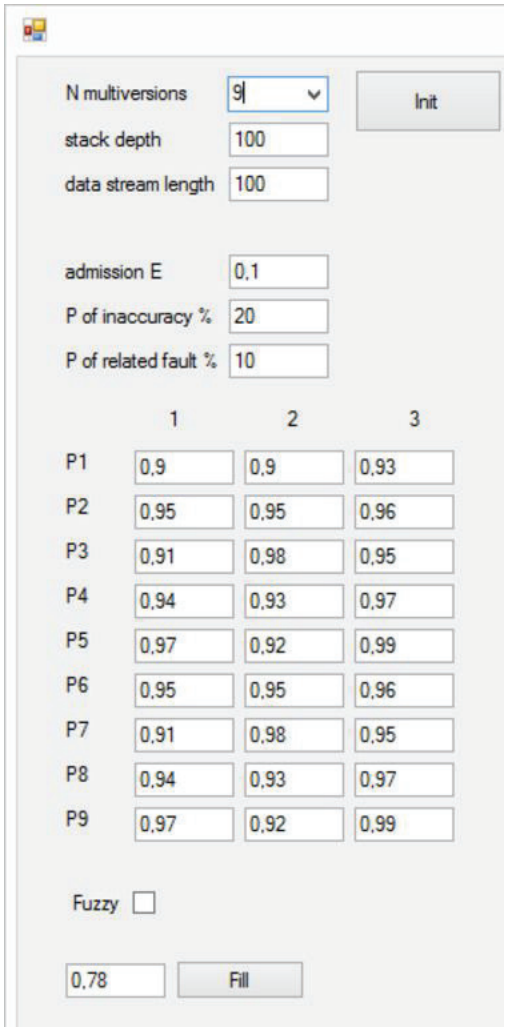

Fig. 1. The interface of the simulation environment: simulation parameters.

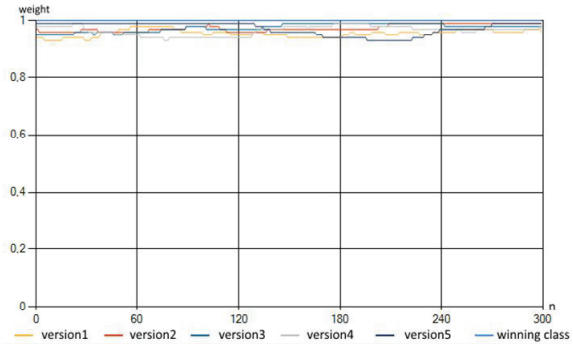

Fig. 2. Results of the simulation model implementation of the fuzzy voting algorithm by an agreed majority (versions are potentially reliable).

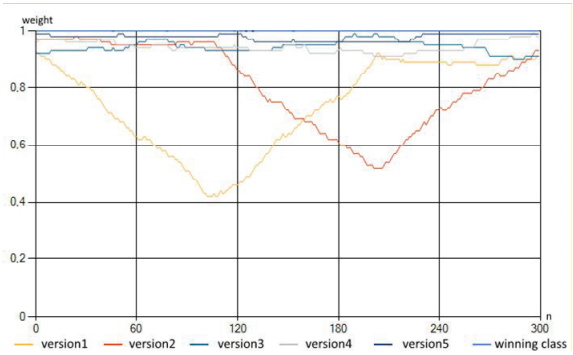

Fig. 3. Results of the simulation model implementation of the voting algorithm by an agreed majority (in the first version of the first data set a failure and in the second version of the second data set with 50% probability).

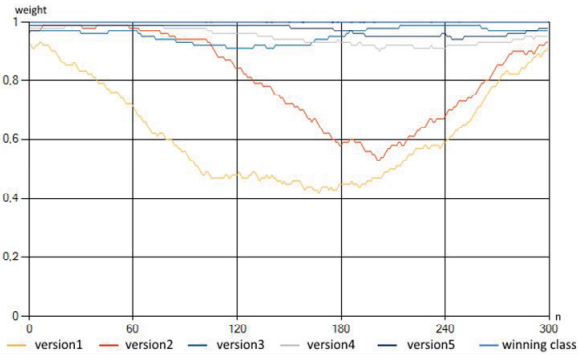

Fig. 4. Results of the simulation model of the implementation of the voting algorithm by an agreed majority (in the first version of the first data set a failure and in the second version of the second data and in the second version only second data with 50% probability).

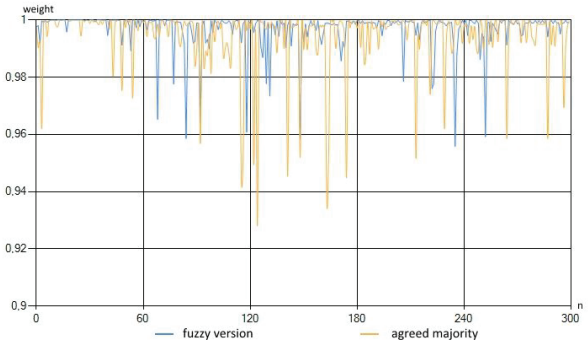

Fig. 5. Results of the simulation model: weights of the winning classes for all voting algorithms.

The system's reaction to the change in version behavior is visible on the graphs (Figures 2-4) when a reliable version starts to give faults or on the contrary, a low-weight version no longer makes a mistake. In an article shows the graphics for a stack of 100 deep and three consecutive data streams of 100 votes for each. Overall, it is evident that the system responds quickly to changes in the behavior of the versions and, through a number of votes equal to the depth of the version stack, receive estimated weights that are accurate in their probability of a correct answer.

Based on the simulation results, the program selects the optimal algorithm based on the total number of errors allowed and the sum of the weights of the classes. Output numerical results are also presented on the form: the total number of errors, the number of failures and inaccuracies for a fuzzy voting algorithm, the sum of the weights of the winning classes.

For the convenience of debugging on the form, the system parameters are displayed at the last failure: the iteration number, the values of the winning class, its weight. This can help in identifying inadequate operation of voting algorithms on a certain set of input parameters.

If the system can uniquely choose the best voting algorithm based on the simulation results, the user is given a pop-up window with the selected result, if the best algorithm can not be uniquely determined on the basis of the received data, the user is asked to make a decision independently based on the modeling data presented by the system on the form.

This simulation environment supports further refinement, it is possible to add new decision algorithms, for example - t/(n-1) algorithm, in order to compare it with classical algorithms and algorithms modified by us under the same simulation conditions.

2.2 Modeling results

Table 1. Results of the system performance for different quantities of multiversions (N 3-6).

N		3	4	5	6
Precise	Number of faults	49	22	7	0
	Total weights	269,43	286,53	296,05	299,21
Imprecise	Number of faults	88	45	13	2
	Total weights	279,45	289,04	297,70	299,70
Related faults		16	22	42	48

The system's reaction to the change in version about the last recorded fault at the output of the voting block to the iteration number, the output value, and the weight of this class. The system permits to build the weight graphics of each version and the winning classes along the axes of the iteration number and weight. In addition, we have an opportunity to change a scale for clarity, for example when examining the weights of the winning classes, the values do not fall below 0.9 and on a scale from zero to one represent a practically straight line flat in the field of one. For better perception in this mode, the scale changes to a range from 0.9 to one along the weight axis.

A form for convenience shows the information about the last recorded fault at the output of the voting block to the iteration number, the output value, and the weight of this class. The system permits to build the weight graphics of each version and the winning classes along the axes of the iteration number and weight. In addition, we have an opportunity to change a scale for clarity, for example when examining the weights of the winning classes, the values do not fall below 0.9 and on a scale from zero to one represent a practically straight line flat in the field of one. For better perception in this mode, the scale changes to a range from 0.9 to one along the weight axis.

Table 2. Results of the system performance for different quantities of multiversions (N 7-9).

N		7	8	9
Precise	Number of faults	3	0	0
	Total weights	298,67	299,21	299,21
Imprecise	Number of faults	4	2	2
	Total weights	299,46	299,70	299,70
Related faults		84	89	123

Tables 1-2 summarizes the simulation results for different numbers of multiversions from 3 to 9, with the probability of all versions in all streams equal to P = 0.7, the depth of the stack 100, the tolerance E = 0.1, the probability of inaccuracy P = 20%, the probability of related faults P = 10%.

The low reliability index of all versions as 0.7 is taken for clarity of simulation results, since with quite reliable versions (P> 0.95) even with the number of multiversions N = 3, both voting algorithms give no faults for 300 votes [15-16]. What we have seen data is a kind of a perfect example of the system reliability increases overall, despite each models extremely unreliable software modules [14,17,18] throw an fault of 30% cases, we observe that in a system with 9 versions all 300 responses are correct.

The fuzzy algorithm shows more faults, because sometimes voting is won a class with a value close to the right one (no further than in E from the correct one), but not equal to it [19]. This answer is considered by the system as a fault. However, the versions that added weight to the winning class still get to stack one, increasing their own weights. Except that a large number of faults, the sum of the weights of the fuzzy algorithm is higher.

3 Conclusions

The results obtained with the help of developing simulation environment of modeling demonstrate the effectiveness of the proposed voting algorithms. In addition, it proves the possibility of creating a reliable system of the unreliable software modules and it is very important in the issue of pattern recognition and classification. Since in current time, there are not fault-free algorithms in this field and the multiversion approach will significantly increase the accuracy of recognition and classification. This approach increases the reliability of the on-board software management system for autonomous unmanned objects.

The reported study was funded by Russian Foundation for Basic Research, Government of Krasnoyarsk Territory, Krasnoyarsk Region Science and Technology Support Fund to the research project № 16-47-242143.

References

1. D.E. Eckhardt, L.D. Lee, *IEEE Trans. Soft. Eng.*, **SE-11**, no. 12, 1511-1517, (1985)
2. J. L. Gersting, R. L. Nist, D. B. Roberts, R. L. Van Valkenburg, *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, 253 – 262, **2** (1991)
3. K. E. Grosspietsch, *Proceedings 16th International Parallel and Distributed Processing Symposium*, (2002)
4. L. Chen, A. Avizienis, *Proc. Int. Symp. Fault-Tolerant Computing FTCS-8*, 3-9, (1978)
5. J.-C. Laprie, J. Arlat, C. Beounes, K. Kanoun, *IEEE Computer*, **23**, no. 7,39-51 (1990), Reprinted in Fault-Tolerant Software Systems: Techniques and Applications, H.Pham (ed.), *IEEE Computer Society Press*, 5-17 (1992)
6. Z. Aghajani, M. A. Azgomi, *2009 International Conference on Innovations in Information Technology*, 304 – 308 (2009)
7. G. Latif-Shabgahi, S. Bennett, *Proceedings 25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium*, **2**, 113 – 120, (1999)
8. I.V. Kovalev, P.V. Zelenkov, V.V. Losev, D.I. Kovalev, N.V. Ivleva, M.V. *IOP Conf. Series: Materials Science and Engineering,* 173 (2017) 012025.
9. J. Knight, B. Littlewood (eds.), *IEEE Softw.*, January (1994)
10. Wang Ping A *2007 8th International Conference on Electronic Measurement and Instruments*, 2-666 - 2-669, (2007)
11. S.S. Brilliant, J.C. Knight, N.G. Leveson, *IEEE Trans. Soft. Eng.*, **16**, no.2, 238-247 (1990)
12. J.C. Knight, and N.G. Leveson, *IEEE Transactions on Software Engineering*, **SE-12**, no. 1, 96-109, (1986)
13. R. Frullini, A. Lazzari, *Proceedings of the International Conference on Railway Safety Control and Automation Towards 21st Century*, London, U.K., 292-299 (1984)
14. M.R. Lyu (ed.), *Software Fault Tolerance* (John Wiley & Sons, Chichester, U.K., January 1995)
15. I. Lee, D. Tang, R.K. lyer, and M.C. Hsueh, *IEEE Trans. Rel.*, **42**, no. 2, 238-249 (1993)
16. A. Athavale, "Performance Evaluation of Hybrid Voting Schemes," M.S. thesis, North Carolina State University, Department of Computer Science, (1989)
17. A. Avizienis, *IEEE Trans. Soft. Eng.*, **SE-11**, no. 12, 1491-1501 (1985)
18. A. Avizienis, and L. Chen, *Proceedings COMPSAC,* 77, 149-155 (1977)
19. D.F. McAllister, C.E. Sun, and M.A. Vouk, *IEEE Trans. Rel.*, **39**, no. 5, 524-534, (1990)