An environment to support structural testing of autonomous vehicles

Vânia de Oliveira Neves, Márcio Eduardo Delamaro,
Paulo Cesar Masiero
Depto de Sistemas de Computação - ICMC
Universidade de São Paulo - São Carlos, SP - Brasil
{vaniaon,delamaro,masiero}@icmc.usp.br

***Abstract*—The software used to control autonomous vehicles is a type of embedded system that needs to undergo strenuous testing before deployment. Field testing is the final stage of testing ensuring that autonomous vehicles show the intended behavior. It usually does not take into consideration the code structure. In this context, a previously proposed testing model and a software tool to support structural testing in the context of autonomous vehicle field testing have been improved to support the generation of new input data from logs collected during field testing using strategies of combination and mutation. Two strategies and an exploratory study to assess their coverage according to the criteria all-nodes and all-edges are presented.**

I. Introduction

An Autonomous Mobile Robot (AMR) is an embedded system that has as main feature the ability to move and operate in semi- or fully autonomous mode [1]. An autonomous vehicle is a type of AMR that aims to allow it to move and make several maneuvers, such as passing another vehicle, park and obey traffic rules without the presence of a conductor, i.e., autonomously. They are critical systems and, as such, require high quality and must be tested carefully and thoroughly. The test of such software can use functional techniques, which is more common, or structural techniques based on code. There are several researches related to embedded systems that seek to increase the test coverage of the code without having to target a specific type of failure [2]. Offline simulation or execution is also a technique widely used for its lower cost and difficulties to test in real situations.

In this sense, the objective of this work is to propose an environment to support structural testing of software to control autonomous vehicles complementing field testing. This test is performed off-line based in one of the main inputs for this type of test: logs of point clouds. Specifically, the paper proposes an algorithm to generate new logs from logs collected in field tests and presents an exploratory study of mutation and combination strategies for point clouds, which are data structure obtained by processing the images received by the stereo camera of the vehicle.

This paper is organized as follows: Section II defines and discusses briefly the field of autonomous vehicles testing; Section III proposes heuristics to generate test data for autonomous vehicle software and a meta-model that supports this generation; Section IV defines two strategies for generating new test data using mutation and combination; Section V presents a tool that implements the model proposed in Section III; Section VI presents an exploratory study that uses the tool developed and a real software to control autonomous vehicles to assess the coverage obtained by the strategies proposed; and, finally, Section VII contains the paper's conclusions and future works.

II. Field testing of autonomous vehicles

According with Urmson et al. [3], the development of software to drive automatically an autonomous vehicle follows a cyclical process, as shown in Figure 1. Based on the requirements, algorithms to perform the required functionalities are developed and they are tested using different types of simulators in a cycle, until the technology is accepted. This occurs when the algorithms become sufficiently mature and tested. At this point the software and hardware of the controller are embedded in the test vehicle with the aim of assessing the system's interactions with the environment. These tests often reveal problems with the algorithms or implementation errors not discovered during the simulated tests.

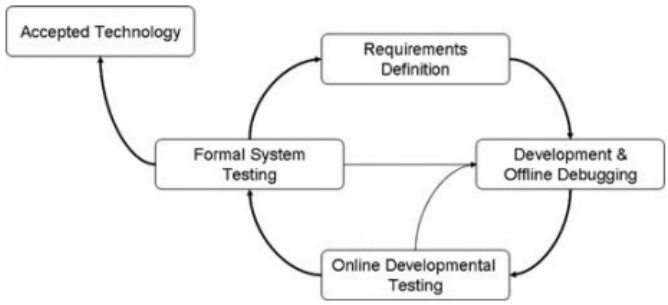

Fig. 1. Development process used to develop Boss[3])

Tests with whole systems (software, hardware and vehicle) are essentially functional and based on scenarios. Data can be collected during these tests and can be used to rerun the software (or part thereof) under different scenarios or situations, in a type of regression testing.

Test with the vehicle in the environment is called by Trepagnier et al. [4] 'Field Test'. They claim that in field test 'the vehicle goes through strenuous tests designed not only to test the reliability and endurance of software and hardware, but also vehicle performance ... going through difficult terrain, difficult scenarios obstacles to be avoided and long run times'. Field tests cannot be remade using exactly the same input data

as they are collected online during the execution of the test, but these data can be used offline later in simulated executions.

Bacha et al. [5], Thrun et al. [6] and Raulskolb et al. [7] emphasize the importance of regression testing using data collected in field tests or tests based on simulations. They comment on the existence of visualization tools in the development environment, as well as rerunning the software from data collected and stored in log files. Field tests are final and decisive tests to ensure that an autonomous vehicle behaves as expected, but they do not take into account any information about the structure of the source code. It is possible that a vehicle has passed through several field tests successfully but important parts of the source code have never been performed.

III. Proposal of heuristics to support field test offline

Considering the literature of the area and through contacts with autonomous vehicles research groups, we realized that structural testing of software in this type of system, although important, is often neglected during the development process. One possible reason is that this technique relies heavily on testing tools available in the development environment.

Programs for autonomous vehicles undergo several successive refinements, many parts of the code or even the whole code of a unit may change, thus generating a large number of versions of a program, of test scenarios and of logs. To support structural testing in this domain, also taking into account these characteristics, the authors of this article defined a meta-model to represent the most important concepts of this phase of the development process [8]. A supporting tool based on this model was also developed.

A version of a program is defined in this model as a specific implementation of a program that differs in some part of the code, but keeps the same architectural elements. Test scenarios are test plans with a specific goal of testing a subset of features of a program under test subject to certain conditions. Logs are records of actual test data captured during a field test of a program version for a particular scenario. Test data captured during a field test are entries received by the vehicle, i.e.. the controller software, as for example: images from a camera and position received from a GPS. Combining a version and a log entry is possible to simulate the program execution in a simulation environment; the results may be displayed using various testing criteria and types of graphs (control flow and data) using different levels of granularity.

As a continuation of this work, the next step is to generate other input of test data with the goal of improving coverage according with certain criteria. For this generation, it would be interesting to use a reduced set of test data representative of data contained in a log [9]. A set of this type can be used, for example, as the initial population of search algorithms or regression testing. Thus, as described in greater detail by Neves et al. [10], five field tests have been conducted with an autonomous vehicle program controlled by a system (hardware and software) developed by the ICMC-USP robotics group [11], resulting in five logs. These logs were partitioned into segments (or subsequences) containing approximately five values of the same type of input data. The segments were defined with five point clouds each so that they represent a small part of the route taken during the test. The input data considered for this partitioning are point clouds.

The controller program was executed in a simulated situation using a log composed by segments as input and the coverage for criteria all-nodes and all-edges was calculated. The reasoning to create the heuristics to generate segments comes from the physical intuition of how the program behaves when executed: in general, for each cycle of reading a point cloud as input, if there are no obstacles in the way requiring, for example, detours and/or speed changes, it is natural that the program run the same computational commands and conditions evaluated do not change. As a minimum limit, each segment could have just one point cloud.

Analyzing the results obtained, it was noted that there are some segments that give a greater contribution to the coverage. These segments were interpreted as those in which there was some change in the conditions of the route that led the program to go change its route or its regular behavior in its execution flow. We gave the name of discriminatory segment to these special segments. According with the initial intuition, they occur in small numbers and can be good candidates to generate the initial population sought. Based on them seven heuristics have been created and tested, and among the ones that produced the best results two have been select [10]. These were heuristics H2 and H6 and contained the segments that better discriminated all-edges criterion. With the completion of this study, it was observed that with a small subset of these segments it is possible to obtain a near-complete coverage of the execution log. As an example, for log 5, which contains 2490 point clouds and produces a coverage of 95% for the criteria all-nodes for one of the methods of the program, the heuristic H6 consists of 13 segments (about 65 point clouds) and produces a coverage of 92%.

Thus, the meta-model originally proposed by Neves et al. [8] was also enhanced and expanded to support the generation and storage of heuristics [10] and the generation of new test logs derived from existing logs, as can be seen in Figure 2.

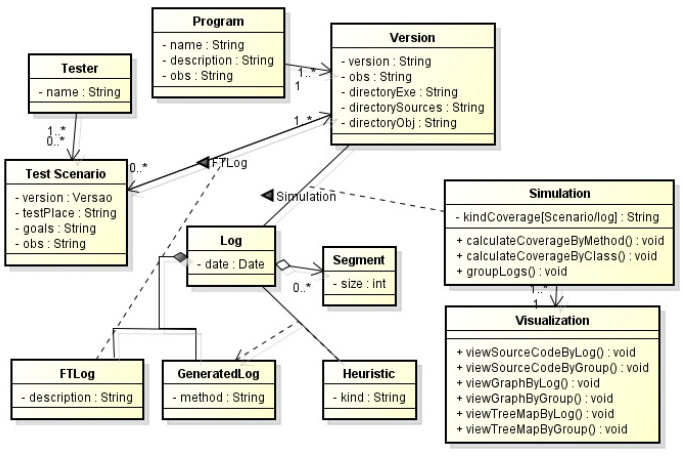

Fig. 2. Meta-model to support field test

This new model considers that a log can be of two types: log obtained from a field test (FTLog) and log generated according to some heuristic (GeneratedLog). Also, a Log may

be composed by zero to n segments, being n the number of point clouds in the segment.

The logs generated can be obtained by the composition of the segments obtained by some heuristic or by the choice of the tester. In addition, from these generated logs it is possible to obtain new logs applying changes in point clouds belonging to a segment. The next section explains in more detail how this is done.

IV. Generating new logs from heuristics based on combination and mutation

With the aim of finding good heuristics for combining point clouds to form other point clouds most likely to increase the coverage obtained with the initial log, an algorithm that uses the meta-model in the previous section and generates new log has been designed. The algorithm has four steps:

- Choose an initial log to be the base. Preferably a small log, so that the tester can create by selecting segments or by generating from a heuristic.
- For each segment of the chosen log apply a mutation strategy and a combination strategy on its point clouds and generate new point clouds.
- A particular point cloud generated is inserted (or merged) in the segment, in a location defined by the strategy, explained below, thus forming a new segment.
- The log generated is composed by these new segments containing new (or modified) point clouds. The coverage obtained according to the desired criterion is calculated by executing the new log generated in the same order of the original log.

The strategy of combination and mutation may be based in a physical intuition about the environment in which the field test was conducted. It could also be purely logical or abstract. Generically, let L' be the log base comprised by a set of S segments selected by a heuristic H and m be the number of segments of L'. S consists of a sequence of k point clouds (pc). It is determined by a function of order $n, n > 0$. In the context of this work an strategy St is functions of order 1 (mutation based in one point cloud) ($St(pc) \mapsto pc$) or 2 (combination based on two point clouds) ($St(pc, pc) \mapsto pc$). In the first case, the point cloud generated is inserted in the segment after the one which is its origin and in the second case it is inserted between the two point clouds that were used to generate it.

A new log L'' is a log generated from L' and consists of m segments S', where S' is generated considering a strategy of mutation or combination to create new point clouds modified pc'. Thus, S' contains pc and pc' merged as shown in Figure 3, where the clouds unfilled represent pc and the clouds filled represent PC'. Thus, the number of point clouds in each segment, original or modified, of S' is $2k$ for mutations and $2k-1$ for combinations. For this exploratory study a mutation strategy and a combination strategy have been used.

The combination strategy is binary and the axis x, y, and z of the new point cloud are calculated according with the

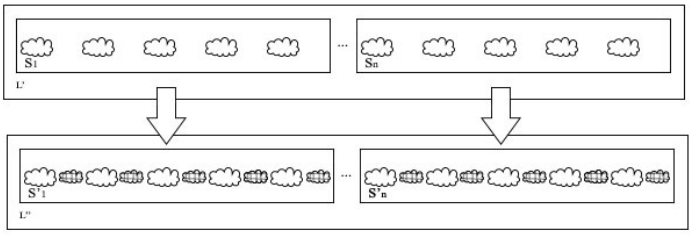

Fig. 3. Generation of new log diagrammatically (mutation)

average of the axis of the two point clouds. It is defined as follows, for i integers varying from 1 to $n-1$, where n is maximum number of point clouds in each segment:

$$AVG(pc_i(x1, y1, z1), pc_{i+1}(x2, y2, z2)) \mapsto pc'((x1 + x2)/2, (y1 + y2)/2, (z1 + z2)/2)$$

The mutation strategy is unary and inserts a point cloud NP'_i with a constant c added to the axis z of every cell of Np_i. This strategy increases (or decreases) the height of the z axis. It is defined as follow, for i in the interval $1...n$, as above:

$$HGT(pc_i(x, y, z)) \mapsto pc'(x, y, z + c)$$

After the generation of new segments S', the coverages obtained by log L'' are calculated for each test criterion desired. This coverage can be compared with the coverage of the base log, but more importantly, it is possible to check if any new test requirement was covered, i.e., a requirement not previously covered by the log base L'. We call this coverage the aggregate coverage of L' and L''. So, being nC the number of nodes covered, nT the total number of nodes, eC the number of covered edges, and eT the total number of edges on a Control Flow graph, the coverage of L' is calculated as follows:

$$Cov_{Nodes}(S_1, ..., S_n) = (\frac{nC}{nT})$$

$$Cov_{Edges}(S_1, ..., S_n) = (\frac{eC}{eT})$$

Similarly, the coverage of L'' is:

$$Cov_{Nodes}(S'_1, ..., S'_n) = (\frac{nC}{nT})$$

$$Cov_{Edges}(S'_1, ..., S'_n) = (\frac{eC}{eT})$$

Be NC' the set of requirements covered by the execution of log L' for a testing criterion, and be NC'' the set of requirements covered by the execution of log L'' for a testing criterion. The aggregated coverage of L' and L'' is:

$$Cov_{Ag_{Nodes}} = \frac{|NC - NC''|}{nT}$$

$$Cov_{Ag_{Edges}} = \frac{|NC - NC''|}{eT}$$

V. A tool to support offline test of autonomous vehicles

A tool has been implemented to support the meta-model proposed in Section III. In fact, it is an extension of the tool described by Neves et al. [8]. The extension of the tool allows: a) partition logs of different sizes using a parameter; b) select automatically discriminatory segments of a set of segments

of a certain size n from a log using a particular heuristic; c) generate a new log from these selected discriminatory segments or from a set of arbitrarily segments selected arbitrarily by the tester; d) generate new logs using combination and mutation strategies from the logs generated in c); e) simulate logs generated both c) and d); f) calculate the coverage of logs generated in c) and d) and compare them, calculating the aggregate coverage as well.

Figure 4 shows the sequence diagram for the activity of generating logs. First, the tester selects a log to be partitioned and next generate segments. Then, the tester can request the generation of a new log composed of discriminatory segments selected according to an informed heuristic or by manually selected segments. Next, combination and mutation strategies can be applied to generate a new log and execute a simulation to calculate the coverage obtained. Using the tool, the tester can view the percentage of the coverage obtained for the criteria all-nodes and all-edges of the classes and methods involved, the source code of the program highlighting the lines not executed, and the Control Flow Graph differentiating nodes and edges not executed. It is also possible to select two logs to compare coverage between them. When this option is chosen, the tool also calculates the percentage of aggregated coverage for the criteria all-nodes and all-edges.

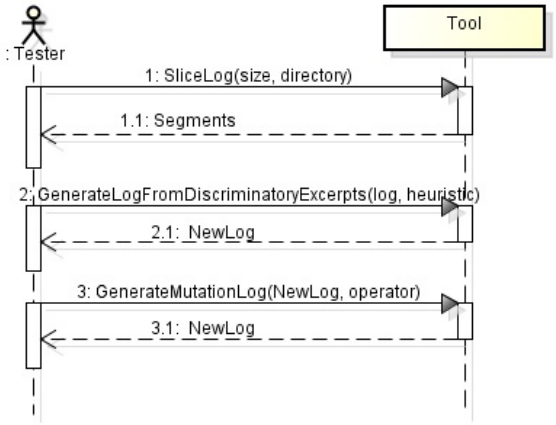

Fig. 4. Sequence Diagram to generate logs

VI. An exploratory study of the coverage obtained with logs generated from combination and mutation strategies

An exploratory study was initially conducted with two goals: to validate the tool developed based on the meta-model proposed in Section V and to assess the efficacy of the two combination and mutation strategies defined in Section IV, to understand how to solve the problem of increasing coverage for this type of software and eventually define and test other strategies.

The program used in the exploratory study aims to drive an autonomous vehicle in an urban area to get to a point defined by a GPS coordinate, avoiding collisions. It is provided with a stereo camera that generates two images that are processed by a global semi-stereo algorithm to producing a disparity map. This map is then converted to a 3D point cloud based on the camera parameters. The orientation of the camera relative to the ground is estimated by the RANSAC (Random Sample Consensus) method. A method to detect obstacles that classifies points based on elevations and relative differences in height is also used. This information is input to the Vector Field Histogram (VFH) method that is used to guide the vehicle to the point of arrival [11]. The class diagram of this program is shown in Figure 5.

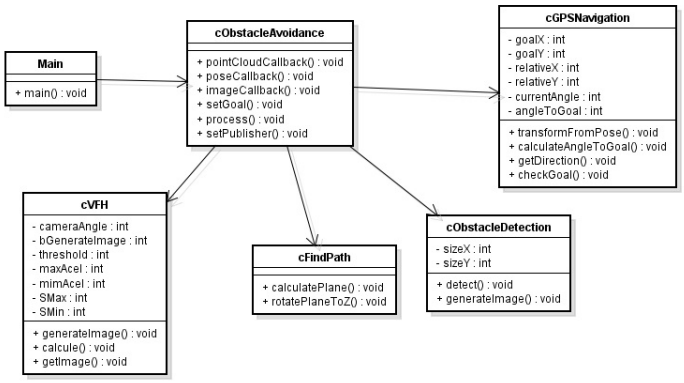

Fig. 5. Class Diagram

The program was implemented using the functionalities of the framework ROS[1]. Thus, the readings of the stereo camera and of the other input data of the program are published by ROS and received by the controller program via the method $main()$. This method sends the input data to the $cObstacleAvoidance$ class, which is responsible for the processes described in the paragraph above, and then make the decision on the actuation of the vehicle: accelerating, decelerating, braking, changing the angle of wheels etc. The two main methods of the program responsible for these calculations and decisions are $process()$, from the class $cObstacleAvoidance$ and $calculate()$, from the class $cVFH$. The first has 155 lines of code without comments and the second 153. The exploratory study and the analysis presented in this article are based on these two methods, because the others are simple and small, easily reaching high levels of coverage of nodes and edges.

After completion of the field testing and the selection of the discriminatory segments described in Section III, we used the tool to generate new logs of discriminatory segments selected by heuristic H2 and H6 (the second best [10]), for the methods under test. These logs were used as basis for the implementation of both combination and mutation strategies defined in Section IV thus creating a second version of the new logs. For the AVG strategy, the value of height constant used was 10. After the simulation of all new logs the coverage criteria all-nodes and all-edges, based on the Control Flow Graphs (GFC) of the methods under test were calculated. Tables I and II show respectively the coverage obtained for the all-nodes criterion and all-edges of the method $process()$. Similarly, Tables III and IV present, respectively, the coverage obtained for the criteria all-nodes and all-edges of the method to $calcule()$.

Analyzing tables it can be noticed that for some logs new testing requirements were covered, although the increase in coverage is small. This result can be better visualized in Figure

[1] www.ros.org

TABLE I. COVERAGE OBTAINED FOR THE METHOD $process()$ AND THE CRITERION ALL-NODES

Log	FT Log	H2-Base	H2-Agg-AVG	H2-Agg-HGT	H6-Base	H6-Agg-AVG	H6-Agg-HGT
1	60%	63%	63%	63%	63%	63%	63%
2	71%	74%	74%	74%	74%	74%	74%
3	70%	63%	63%	63%	63%	63%	63%
4	71%	74%	74%	74%	74%	75%	75%
5	71%	63%	63%	63%	63%	72%	72%
average	68,6%	67,4%	67,4%	67,4%	67,4%	69,4%	69,4%

TABLE II. COVERAGE OBTAINED FOR THE METHOD $process()$ AND THE CRITERION ALL-EDGES

Log	FT Log	H2-Base	H2-Agg-AVG	H2-Agg-HGT	H6-Base	H6-Agg-AVG	H6-Agg-HGT
1	46%	48%	48%	48%	48%	48%	48%
2	56%	58%	58%	58%	58%	59%	59%
3	54%	48%	48%	48%	48%	48%	48%
4	56%	59%	59%	59%	59%	60%	60%
5	56%	48%	48%	48%	48%	56%	56%
average	53,6%	52,2%	52,2%	52,2%	52,2%	54,2%	54,2%

TABLE III. COVERAGE OBTAINED FOR THE METHOD $calcule()$ AND THE CRITERION ALL-NODES

Log	FT Log	H2-Base	H2-Agg-AVG	H2-Agg-HGT	H6-Base	H6-Agg-AVG	H6-Agg-HGT
1	91%	91%	91%	91%	91%	91%	92%
2	93%	93%	93%	93%	93%	93%	93%
3	93%	91%	92%	92%	91%	92%	92%
4	90%	90%	90%	90%	93%	93%	93%
5	95%	92%	93%	93%	92%	93%	93%
average	92,4%	91,8%	91,8%	91,8%	92%	92,4%	92,6%

TABLE IV. COVERAGE OBTAINED FOR THE METHOD $calcule()$ AND THE CRITERION ALL-EDGES

Log	FT Log	H2-Base	H2-Agg-AVG	H2-Agg-HGT	H6-Base	H6-Agg-AVG	H6-Agg-HGT
1	74%	74%	74%	74%	74%	74%	75%
2	78%	77%	77%	77%	78%	78%	78%
3	77%	73%	74%	75%	73%	75%	75%
4	74%	73%	74%	74%	77%	77%	77%
5	81%	75%	78%	78%	75%	79%	78%
average	76,8%	74,4%	75,4%	75,6%	75,4%	76,6%	76,6%

6, which shows the average of the coverage obtained by the five logs used in the study.

Figure 7 presents a diagram that relate the coverage obtained by the log created by the field test, the base log and logs generated by using the strategies of mutation and combination for the all-nodes criterion (a) and for the all-edges criterion (b) of the $process()$ method. By analyzing such diagram, it can be noticed that the AVG and HGT strategies obtained the same results for all logs and both criteria. This may be explained since in the $process()$ method there is no command decision involving the coordinates x, y, z of the point cloud, which were the entries that have been modified by these strategies.

Figure 8 shows similar diagrams but considering the method $calcule()$. They show that strategies AVG and HGT obtained different coverages for some logs. While AVG got better coverage for the all-edges criterion of log 5 using heuristic H6, HGT was better considering the criteria all-nodes and all-edges of log 1 and heuristic H6 and considering the criterion all-edges of log 2 using heuristic H2. It is possible to note that the HGT strategy tends to be better than AVG for this

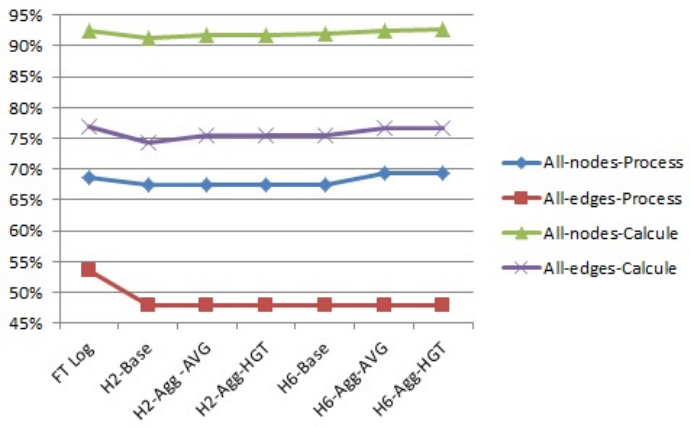

Fig. 6. Diagram with the averages of coverages obtained

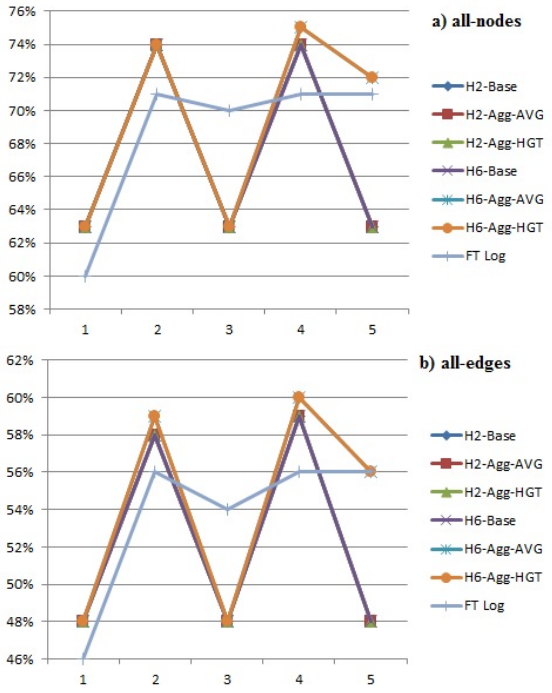

Fig. 7. Diagram showing log coverage obtained from the combination of heuristic and strategy for the method $process()$ and criteria all-nodes and all-edges

method. This can be explained because $calculate()$ considers the z axis of the point cloud to do processing. However, one can not come to any conclusion since the number of logs used in this study is small.

Other interesting point to observe is that for most logs of the method $process()$ and for some of the logs of method $calcule()$, the coverage reached by all of the criteria all-nodes and all-edges considering the base logs was larger than the coverage reached by the logs generated in the field test (FT-log). Since the coverage of the method $calcule()$ reached by the base log created in the field test was already high, the chances of generating new logs that enhanced coverage were smaller than in the method $process()$, which reached a lower coverage for the same log.

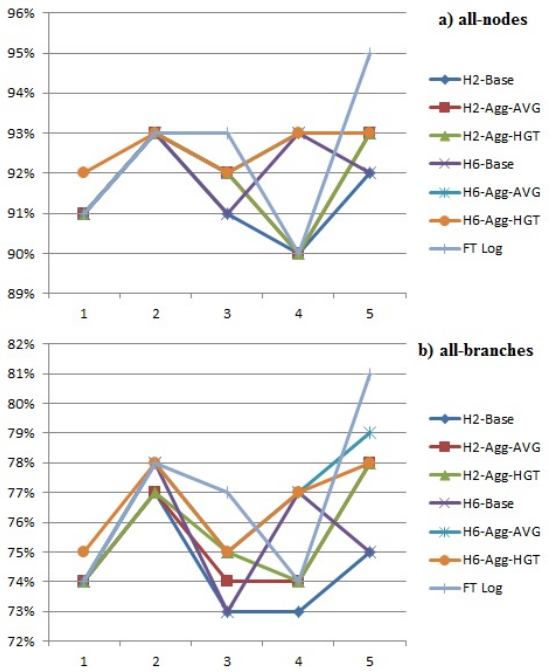

Fig. 8. Diagram showing log coverage obtained from the combination of heuristic and strategy for the method $calcule()$ and criteria a) all-nodes and b) all-edges

VII. Conclusions and Future Work

This work presents an environment that supports structural testing of autonomous vehicles. A meta-model defined in an earlier work and that stores information of test scenarios, logs and versions of a program has been enhanced to support the generation of heuristics to select discriminatory segments of execution logs and also to generate new logs modified according to strategies of combination and mutation, which have also been defined in this paper.

A tool that implements the proposed model was presented and an exploratory study using this tool was also presented. The aim of this study was to better understand the process of generating mutations and combinations of point clouds and how to build a support tool to support this process. The result was promising because it was perceived that requirements not executed by the field test were covered by the logs generated, even using simple strategies for combination and mutation.

From this study we intend to define and evaluate new and more complex strategies of combination and mutation, considering other sizes of segments and also other software for controlling autonomous vehicles, including water and aerial vehicles. In addition, we intend to improve the model and the proposed tool to include the generation of new logs using search based algorithms, and creating logs derived that satisfy a fitness function, as for example how to reach a specific node or edge, or to execute a given command in order to increase the coverage.

Akcnowledgements

The authors acknowledge the financial support from the brazilian research agencies FAPESP, CAPES, CNPq and INCT-SEC.

References

[1] D. F. Wolf, E. V. Simões, F. S. Osório, and O. T. Junior, "Robótica móvel inteligente: Da simulação Às aplicações no mundo real," vol. 1, 2009.

[2] J. C. Costa and J. C. Monteiro, "Coverage-directed observability-based validation for embedded software," *Trans. on Design Automation of Electronic Systems*, vol. 18, no. 2, pp. 1–20, 2013.

[3] C. Urmson, J. Anhalt, D. Bagnell *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, pp. 425–466, August 2008.

[4] P. G. Trepagnier, J. Nagel, P. M. Kinney, C. Koutsougeras, and M. Dooner, "Kat-5: Robust systems for autonomous vehicle navigation in challenging and unknown terrain." *J. Field Robotics*, vol. 23, no. 8, pp. 509–526, 2006.

[5] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson, S. Cacciola, P. Currier, A. Dalton, J. Farmer, J. Hurdus, S. Kimmel, P. King, A. Taylor, D. V. Covern, and M. Webster, "Odin: Team victortango's entry in the darpa urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 467–492, 2008.

[6] S. Thrun, M. Montemerlo *et al.*, *Journal of Field Robotics*, vol. 23, pp. 661–92, September 2006.

[7] F. W. Raulskolb *et al.*, "Caroline: An autonomously driving vehicle for urban environments," *J. Field Robotics*, vol. 25, no. 9, pp. 674–724, 2008.

[8] V. d. O. Neves, M. E. Delamaro, and P. C. Masiero, "Structural testing of autonomous vehicles," in *Proc. of the 21th Int. Conf. on Software Engineering and Knowledge Engineering*, ser. SEKE'13. Boston, USA: Knowledge Systems Institute, 2013, pp. 200–05.

[9] Y. Yu, J. A. Jones, and M. J. Harrold, "An empirical study of the effects of test-suite reduction on fault localization," in *Proc. of the 30th Int. Conf. on Software Engineering*. New York, USA: ACM, 2008, pp. 201–10.

[10] V. d. O. Neves, M. E. Delamaro, and P. C. Masiero, "Heuristics for the selection of the initial population of search based data generation algorithms for autonomous vehicles controller software," in *SAST 2004 - 8th Brazilian Workshop on Systematic and Automated Software Testing*. Porto Alegre, BR: SBC, 2014, p. to be published (in Portuguese).

[11] C. C. T. Mendes and D. F. Wolf, in *Real Time Autonomous Navigation and Obstacle Avoidance Using a Semi-global Stereo Method*. New York, USA: ACM, 2013, pp. 235–36.