A Systematic Testing Approach for Autonomous Mobile Robots Using Domain-Specific Languages

Martin Proetzsch[1], Fabian Zimmermann[2], Robert Eschbach[2], Johannes Kloos[2], and Karsten Berns[1]

[1] Robotics Research Lab, TU Kaiserslautern, 67653 Kaiserslautern, Germany
{proetzsch,berns}@cs.uni-kl.de
[2] Department of Testing and Inspections, Fraunhofer IESE, 67663 Kaiserslautern, Germany
{fabian.zimmermann,robert.eschbach,johannes.kloos}@iese.fraunhofer.de

Abstract. One aspect often neglected during the development of autonomous mobile robots is the systematic validation of their overall behavior. Especially large robots applied to real-world scenarios may cause injuries or even human death and must therefore be classified as safety-critical. In this paper, a generic approach to defining and executing purposeful test runs using domain-specific languages (DSLs) is presented. Test cases can be defined in an appropriate test description language (first DSL). These test cases can be derived automatically using a model-based testing approach, for which a test model has to be created. Hence, a second DSL for the creation of the test model is presented. It is further shown how the generated test cases are automatically executed and evaluated. The paper concludes with the application of the approach to the autonomous off-road robot RAVON.

1 Motivation and Introduction

In the field of autonomous mobile robots, systematic quality assurance is made complicated by the complexity of detection and motion generation systems. Since the failure of a robot's control software can have grave consequences, some basic testing is required before putting it into operation. This testing is often done unsystematically and in an ad-hoc manner. Furthermore, test cases have to be described on a very low and technical level of abstraction. Thus, testing requires complicated programming and adaptation of test cases for new versions.

Some previous research has been conducted to assure the quality of robot control software. There have been efforts to guarantee certain properties of systems using formal verification techniques [1,2]. However, the increasing number of states for complex control systems limits this approach to relatively small subsystems. In [3] fault trees are used for the verification of a mobile robot. Here, only failures identified during fault tree analysis are considered. Further approaches use an architecture with a safety layer to avoid hazards such as collisions [4]. This works for the construction of a new robot, but changing the existing architecture of RAVON would require considerable effort.

R. Dillmann et al. (Eds.): KI 2010, LNAI 6359, pp. 317–324, 2010.

Other contributions deal with performance measures [5,6]. Here, aspects such as usefulness of actions, smoothness and accuracy of paths, time to completion of tasks, and distance traveled are used for evaluating the progress made during system development. However, no information is given on how these techniques can be used to provide a generic approach for generating, executing, and evaluating test cases. A general overview of safety aspects in the field of artificial intelligence is provided in [7]. The paper at hand presents an approach for a high-level description of tests for autonomous mobile robots. It is based on domain-specific languages and leverages model-based testing. The approach is evaluated on the autonomous robot RAVON (Robust Autonomous Vehicle for Off-road Navigation).

2 Test Case Definition

2.1 Test Description Language

Defining test cases for a mobile robot in a generic way requires a test description language capable of expressing the relevant aspects. First, for a mobile robot, a list of *commands* to be executed has to be specified. As the application presented here concerns driving across unknown terrain, commands are expressed as a list of *target points* to be approached. During execution, several *events* can occur, e. g., moving obstacles or deliberatively disturbed sensor data. For a complete test run, a list of commands and events can be defined. For the chosen application, this is a *path* containing target points to reach and events occuring during the traversal.

Besides a definition of the commands, one crucial aspect during test cases is how malfunctions can be detected, i. e., a test oracle has to be defined. Since RAVON uses a behavior-based control system (cf. Section 3), no exact definition of correct system behavior exists. If RAVON should reach a target point, any sequence of actions that brings the robot to the target point within a certain period and without any collision can be seen as correct. Similar problems also occur when testing other autonomous robots. Thus, instead of defining a detailed expected response for each input (here, inputs are the coordinates of the next target point), a set of rules is given. There are two different kinds of rules, called *invariants* and *goals*. Invariants should always hold, while goals should be reached in the end. An example of an invariant is *no collision*, a possible goal is *the robot has reached the last target point*. A further distinction is made as to whether an occurring *violation* causes a test run to *terminate*. The current test case has succeeded if all goals are reached and no invariant has been violated.

These aspects are defined using a domain specific language (first DSL in our approach). This language contains a list of positions and events that are defined and referenced by indexes for defining paths. For the definition of execution and specification checks, predefined types of modules are referenced that are parameterized according to the demands. The presented test case description language is suitable for automatic generation by means of tools.

2.2 Automatic Generation of Test Cases

Although the language described in the last section can be used to manually define test cases, further improvement of the testing procedure can be achieved through automatic test case generation. Therefore, this paper deals with model-based statistical testing techniques (MBST) [8] for automatically deriving test cases based on a test model. The MBST approach has been evaluated in [9].

Test model. In MBST, a test model is constructed as a transition system. It contains all possible inputs to the system and usually the corresponding expected outputs. In our case of testing an autonomous robot, a state in the test model, i. e., a state in the transition system, represents a position of the robot on the map. This map is selected according to the expected area of deployment. If the area of deployment is unknown, a map containing the expected type of terrain should be chosen. A stimulus is the command to drive to a target point combined with an event that might occur while driving there. Positions where the robot should start its test drives are marked as start states. All states where a test drive might end are marked as exit states. If a transition from a state A to a state B exists, the command *drive to* B could occur in a test drive when the robot has reached target point A. For random test case generation, each transition is annotated with a probability according to a usage profile, i. e., the test model is a Markov chain.

Language for test model creation. Creating the test model is a complicated process usually done by test experts, e. g., using a technique called sequence-based specification [10,11]. A domain expert is usually not knowledgeable in the construction of test models, but has very detailed information about which configurations of the system will be especially critical. Since autonomous robots are a domain of scientific research, there are often no test experts available at all. Thus, it is recommended providing a domain-specific language that enables the domain expert to create the test model [12].

In our approach, the test model can be easily created by the roboticist as the domain expert. We provide a domain-specific language for the description of test models (the second DSL in our approach). Since the test model is a connected graph depicting the area the robot should drive through during testing, we provide an editor for drawing this graph as a domain-specific language. On the map, points are selected as potential target points by clicking. For each target point, a connection is drawn to all target points that could be reached from this point in the next test step. Target points that can serve as starting points are marked, as are points where a test case can end. The connections between target points can be annotated with a probability value. This probability value is used for the random generation of test cases according to a usage profile.

The annotation is done in the editor by left-clicking on the current transition. We provide probability values between 1 (unlikely) and 10 (very likely) with a default value of 5. The probability values of all outgoing transitions of each state are normalized. Furthermore, the robot expert can select different events

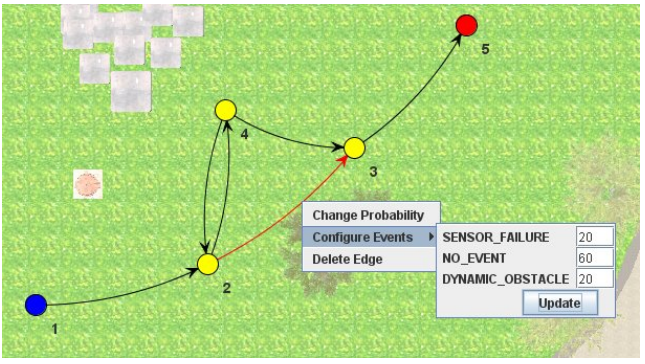

Fig. 1. Example of a test model created with the test model editor. Event probabilities are assigned to the transition from point 2 to point 3.

Fig. 2. The autonomous off-road robot RAVON

that can happen using a certain transition in a test step. If more than one event might happen on a drive represented by a certain transition, probability values have to be assigned to all possible events. If no event is explicitly selected and no probability distribution is explicitly defined, no event will occur, i. e., the probability of NO_EVENT is 100 %.

Figure 1 shows the determination of event probabilities for the transition between point 2 and point 3. Here, in 60 % of the times that a drive from point 2 to point 3 occurs as a test step, no event will happen. In 20 %, a dynamic obstacle will occur in front of RAVON; in another 20 %, we inject a sensor failure.

Generation of test cases. As soon as the test model has been created, test cases are automatically generated following different generation strategies. To test each relevant input sequence at least once, coverage criteria are used, e. g., transition coverage. This means that each transition is visited at least once, i. e., each possible movement from a target point to its neighbors with all possible events occurring is executed. To derive more test cases, the strategy of random test case generation based on the probability profile of the transitions is used. The benefit of this strategy is that a larger number of test cases can be derived. These test drives often contain circles. Thus, the robot is tested in many different, sometimes unexpected situations. For the generation of test cases, the tool jumbl [10] is used.

3 Method Application for Ravon

The application platform used here is the off-road robot RAVON, see Fig. 2. The characteristics of RAVON (2.4 m × 1.4 m × 1.8 m, 750 kg) make it indispensable to consider safety aspects. Therefore, dedicated hardware for low-level emergency stops is integrated, i. e., safety bumpers, emergency stop buttons, and a safety chain connecting the involved components. Besides this, RAVON is equipped with

several sensor systems for obstacle detection, i. e., fixed laser scanners, a panning laser scanner, and stereo camera systems. RAVON's control system is implemented following the behavior-based control architecture iB2C (integrated Behavior-Based Control) [13]. In iB2C, a standardized interface of behaviors allows adjustment of a behavior's relevance, uniform combination of conflicting commands, and abstract evaluation of a behavior's internal state.

A major aspect in the context of performance analysis is the automation of test runs such that experiments can be executed repetitively. This way, basic safety properties of a control system can be evaluated. Due to the high effort of real-world experiments and missing information about the ground truth, preliminary tests have to be based on a simulation environment providing suitable details of the environment and the robot hardware. Here, SimVis3D [14] is used, which supports the simulation of actors, sensors, and environmental properties. That way, it is possible to check fundamental static and dynamic properties of robotic systems without possible damage to the hardware. As the real system is simulated at the lowest possible level, the sensor processing mechanisms and the control system remain unchanged both for simulated and real-world experiments. A predefined hardware interface allows the seamless exchange of hardware simulation and real system.

3.1 Test Run Automation

Based on the description of test cases presented in Section 2.1, automated test runs have been executed to validate the performance of RAVON's control system with respect to reaching predefined positions without violating the safety margins. The execution of commands and the checks of specifications are realized as an iB2C network (cf. Section 3) that is automatically built up according to the test case description. Based on this modular approach, extensions are easily configurable. Furthermore, the standardized interface of behaviors offers the possibility to observe the current state of the test run system during run-time. Figure 3 shows an example of the proposed structure. The depicted behavior network is connected to the upper interface of the control system and replaces the inputs of an operator.

The *Test run supervisor* behavior deals with coordinating the sequence of runs for a given environment and given specifications. The *Test run execution* behavior generates commands to be executed by the control system, i. e., a sequence of positions to be reached. During system execution, the violation of the defined specifications is evaluated by behaviors. For the given application, invariant violation behaviors observe collisions, elapsed time, and the mobility of the robot. Further criteria for evaluating safety properties can be excessive inclination, excessive velocity, or unstable steering motions. The maximum fusion behavior *(F) Invariant violation* gives an abstract representation of whether any of the properties is violated.

Behaviors evaluating the achievement of goals deal with aspects that have to be valid at least once during a test run. For the given example, the evaluation concerns whether a target position is reached, see Figure 3. Here, the

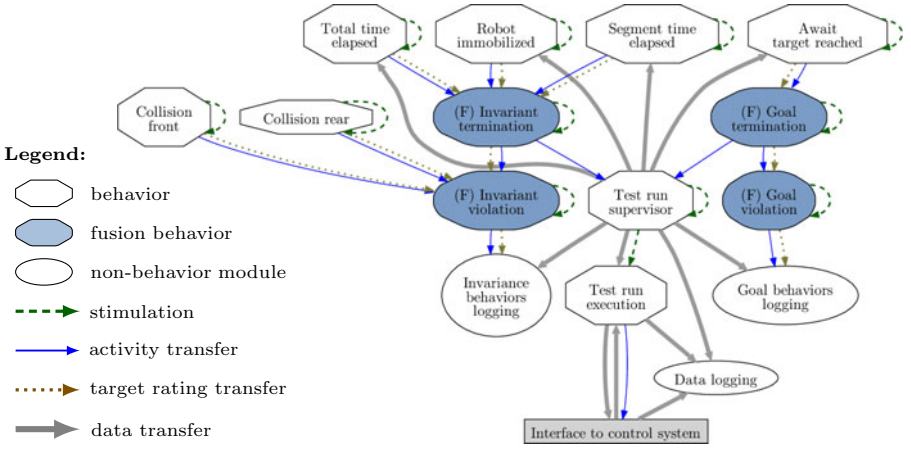

Fig. 3. Behavior network for test run automation

activity of the maximum fusion behavior *(F) Goal violation* therefore is an abstract measure for the accomplishment of all goals. Specifications that are relevant for terminating a test run are also implemented as separate behaviors (the behaviors *Await target reached*, *Robot immobilized*, *Total time elapsed*, and *Segment time elapsed*). This information is transferred to the *Test run supervisor* behavior via fusion behaviors. Their activity output is used to determine when to proceed with the next run. After finishing the last experiment, the program is automatically terminated. For each test run that is finished, the data collected during the experiments are saved and evaluated in terms of specification violations. The proposed approach is applicable both for simplified scenarios to test subsystems and for complex scenarios to validate the cooperation of system components. Although targeted towards simulation environments, the approach can also be used for test runs in real scenarios if the environmental properties can be sufficiently defined. In this case, the specification of invariant properties and goals has to be based on data available on-board.

3.2 Test Run Experiments and Results

The approach presented here has been applied to RAVON to evaluate the performance and safety properties in the given simulation scenario. Several overnight experiments were executed and yielded data for further evaluation. Figure 4 shows the test model and the trace of the robot's position during one of the test runs as an overlay. A selection of the statistical evaluation of the test runs is presented in Fig. 5. Here, for some of the path segments, the number of executions and the number of failed traversals are given. In this context, *failed* corresponds to a violation of any of the given specifications.

The most significant result is the high number of fails for paths 8 to 5. A closer evaluation reveals that the robot collides with the bushes during positioning maneuvers. Here, the vehicle follows a strategy of penetrating possibly traversable

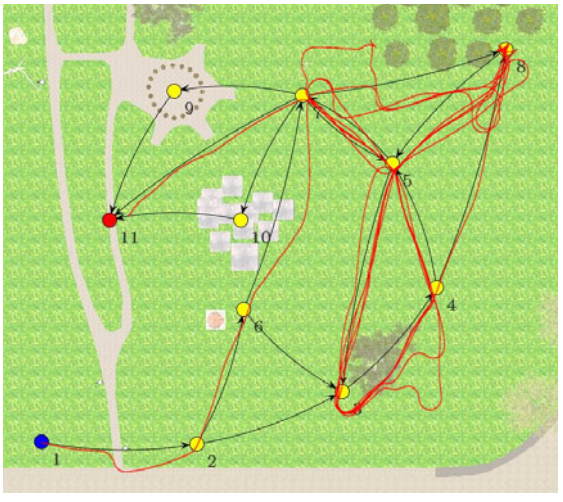

Fig. 4. Test model with overlaid pose trace of an exemplary run from one test case

From	To	Executed	Failed
2	3	50	0
2	6	54	0
3	4	241	3
4	8	106	3
7	10	1	1
7	11	101	3
7	5	30	0
7	8	54	2
7	9	1	0
8	5	160	16
9	11	1	0

Fig. 5. Selection of test run results

structures by slowly driving forward while observing the deflection of a bumper system. This approach has been implemented to allow traversing vegetated areas like grassland. As this happens at a low velocity, it is tolerable with respect to safety. Another interesting aspect is the reachability of points 9 and 10. Due to the low probability, each of them is only approached once. As expected, the result shows that point 10 cannot be reached. The test runs also reveal that the robot succeeds in approaching point 9 although it is blocked by obstacles. This results from the fact that during the approach, the minimal distance between the robot and the target point is less than a given tolerated distance used to mark a point as reached.

The tool support and the facilitated definitions of the test run execution procedure have proven suitable for rapidly adjusting the test run execution to new requirements. That way, several results regarding errors in the control system have been achieved. Furthermore, this approach has been used to validate the simulation environment by executing stress tests on it. In conclusion, contributions to the improvement of many different aspects of complex systems have been achieved.

4 Conclusion and Future Work

The approach presented in this paper provides a possibility for domain experts to easily define a test model that is used to generate an arbitrary number of purposeful test cases. As an automatic evaluation of test run results is provided, this system can be used for automatically checking system properties to find errors introduced during the development process. Future work will include an

extension of the set of possible events and available types of violation detection behaviors.

Acknowledgments. This work was partly funded by the German Federal Ministry of Education and Research (BMBF) in the context of the project ViERforES (No.: 01 IM08003).

References

1. Sharygina, N., Browne, J., Xie, F., Kurshan, R., Levin, V.: Lessons learned from model checking a nasa robot controller. Form. Methods Syst. Des. 25(2-3), 241–270 (2004)
2. Kim, M., Kang, K.C.: Formal construction and verification of home service robots: A case study. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 429–443. Springer, Heidelberg (2005)
3. Lankenau, A., Meyer, O.: Formal methods in robotics: Fault tree based verification. In: Proc. Quality Week Europe, Brussels, Belgium (1999)
4. Lankenau, A., Röfer, T.: A safe and versatile mobility assistant. reinventing the wheelchair. IEEE Robotics and Automation Magazine, 29–37 (2001)
5. Rosenblatt, J.: Damn: A distributed architecture for mobile navigation. Journal of Experimental and Theoretical Artificial Intelligence 9, 339–360 (1997)
6. Goldberg, D., Mataric, M.: Design and evaluation of robust behavior-based controllers for distributed multi-robot collection tasks. Robot Teams: From Diversity to Polymorphism (2001)
7. Lüth, C., Krieg-Brückner, B.: Sicherheit in der künstlichen intelligenz. Künstliche Intelligenz 21(1), 51–52 (2007)
8. Prowell, S.: Using markov chain usage models to test complex systems. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS 2005, p. 318c (January 2005)
9. Hussain, T., Eschbach, R.: Statistical testing of iec 61499 compliant software components. In: Proceedings of INCOM 2009 (2009)
10. Prowell, S., Poore, J.: Foundations of sequence-based software specification 29(5), 417–429 (May 2003)
11. Lin, L., Prowell, S., Poore, J.: The impact of requirements changes on specifications and state machines. Softw. Pract. Exper. 39(6), 573–610 (2009)
12. Kloos, J., Eschbach, R.: Generating system models for a highly configurable train control system using a domain-specific language: A case study. In: Proceedings of A-MOST 2009 (2009)
13. Proetzsch, M., Luksch, T., Berns, K.: Development of complex robotic systems using the behavior-based control architecture iB2C. Robotics and Autonomous Systems 58(1) (2010)
14. Braun, T., Wettach, J., Berns, K.: A customizable, multi-host simulation and visualization framework for robot applications. In: 13th International Conference on Advanced Robotics (ICAR 2007), Jeju, Korea, August 21-24, pp. 1105–1110 (2007)