Towards inferring environment models for control functions from recorded signal data

Henrik Peters, Falk Howar, Andreas Rausch
Clausthal University of Technology, Department of Informatics
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
Email: {henrik.peters, falk.howar, andreas.rausch}@tu-clausthal.de

***Abstract*—In the automotive domain, control functions (e.g., ACC or brake booster) are mainly validated through road tests by means of performing specific driving maneuvers. In many cases, however, there is only an indirect connection between the inputs at the system level (e.g., position of the brake pedal) and the inputs to a tested component (e.g., negative pressure of a brake booster). In order to validate that a software component was tested sufficiently, engineers have to analyze recorded data after road tests. We present an approach for inferring automata models from such data. These (small) automata are easier to analyze than hours of raw signal data: they exhibit specific states and transitions for different test scenarios, which allow engineers to understand how a function was exercised during a road test. Technically, we generate models in three steps: we (1) identify segments of consistent behavior, (2) classify these segments, and (3) generate automata models from sequences of classified segments. We evaluate the presented approach on speed and acceleration data from a small number of road tests.**

I. Introduction

Control functions in the automotive domain (e.g., of the engine control unit) are realized as software components to an ever increasing degree. Automotive software becomes more complex as these components have to be integrated and coordinated in the context of higher-level functions like adaptive cruise control (ACC) or a brake booster. The automotive industry has developed strategies for scaling software development to the required level of complexity in recent years (e.g., modularity, model-based development, and automated code generation). What is still missing today, are adequate strategies for testing components of control software [1].

One challenge when testing software components is that often test engineers do not have access to realistic inputs. In many cases, tests are thus done with quite generic artificial inputs (e.g., sine waves). These tests are only of limited help for assessing the correctness of an implementation. Road tests by means of performing specific driving maneuvers, however, are of limited use to software engineers as well: Frequently, during maneuvers there is only an indirect connection between the inputs at the system level (e.g., position of the brake pedal) and the actual inputs to a tested component (e.g., negative pressure of a brake booster). Engineers have to analyze recorded data (i.e., inputs to components) after road tests in order to validate that a component was tested appropriately. Going through hours of raw signal data in order to find specific behavior is a tedious and error-prone task.

In this paper, we present an approach for inferring abstract automata models for the environment of control components from data recorded during road tests. More precisely, we learn behavioral models for component inputs. These models enable engineers to assess the quality of test inputs more efficient: The models are concise and exhibit specific states and transitions for different test scenarios. This easily allows engineers to understand how a component was exercised during a road test.

Our contribution is a conceptual approach for inferring abstract automata models of inputs for control functions in three steps: First, *abstract traces* are inferred from time series data by (1) identifying and (2) classifying segments of data that correspond to different states of the environment. Then, (3) *automata models* are generated from these abstract traces.

We have implemented all three steps prototypically and evaluate the presented approach on speed and acceleration data from a small number of road tests.

Related Work. The three steps in our approach are similar to the steps in the approach presented in [6] for learning models that can help to detect anomalies in the operation of production plants. However, the realization of the three steps differs significantly: While in [6] the identification of system states is based on discrete events, it is based on computing segments of the quasi-continuous signals in our approach since. Moreover, in [6] the generation of automata models is based on an automata learning algorithm. We construct simpler and more concise automata that do not differentiate states based on future behavior. These concise models are sufficient for engineers to assess the quality of test inputs.

Another recent approach generates non-linear symbolic automata models from signal data using a combination of machine learning techniques (for model fitting) and an evolutionary algorithm (for finding symbolic representations of continuous functions) [3]. The method integrates learning of symbolic descriptions, states, and transitions.

Our approach, on the other hand, decomposes model inference into multiple steps. This will allow us to replace our current model construction by established automata learning algorithms in the future. This will become necessary when using generated models as a basis for generating realistic test inputs for components. One candidate algorithm is presented in [5] and infers deterministic probabilistic finite automata from sets of traces.

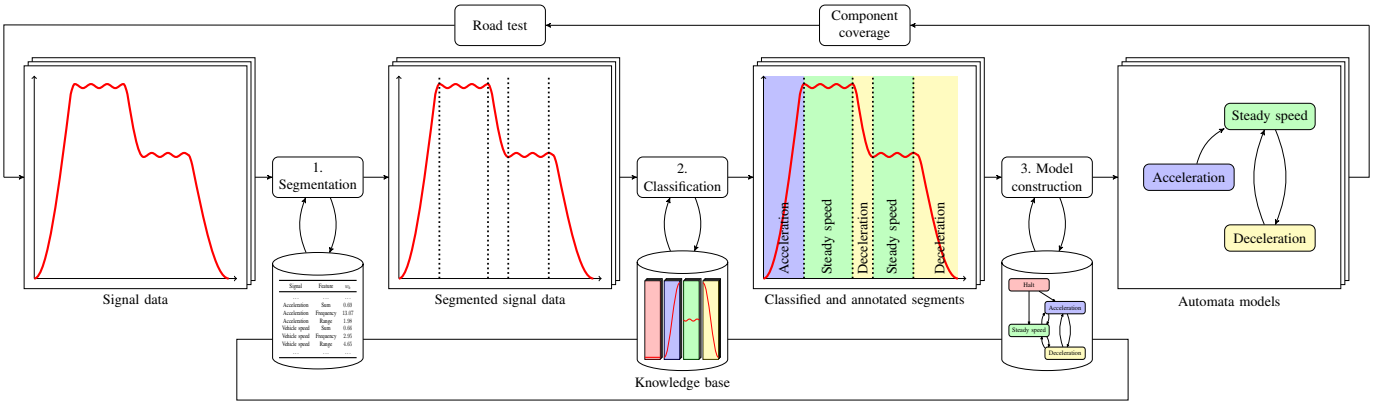

Fig. 1. Workflow of the presented approach: In a first step, segments are identified on a signal recorded during a road test; in a second step, segments are classified using already classified segments as a basis; in the third step, automata models are constructed from sequences of classified segments. Finally, the automata are used to assess whether a component was tested adequately in a road test.

Organization. The remainder of the paper is organized as follows. We detail our approach in the next section and discuss the prototypical implementation of the individual steps. In Section III we present results from the preliminary evaluation of our method before discussing conclusions and future work in Section IV.

II. Inferring Models from Signal Data

We infer models from recorded signal data in three steps as illustrated in Fig. 1. In the first step, we compute segments on an input signal to some component recorded during a road test. We search points in time at which the quasi-continuous behavior of the signal changes significantly. E.g., for the signal depicted in Fig. 1 we compute five segments: initially the signal increases, then remains relatively constant, then decreases a little, remains constant, and decreases further. In the second step, the identified segments are classified using the knowledge base of already classified segments as a basis. Classification uses a small number of domain-specific classes (e.g., 'halt', 'steady speed', 'acceleration', and 'deceleration' in our example). In the third step, we construct an automaton model from a sequence of classified segments by counting transitions between classes (i.e., states of the automaton). The automaton is then used to analyze the recorded test inputs at an abstract level—instead of manually evaluating hours of raw signal data.

The remainder of the section discusses the technical realization of the three steps in our current implementation. Let therefore a *signal* S be a sequence of pairs $(t, x) \in \mathbb{R}^2$, where t is a time and x is the signal data. For $S = (t_0, x_0) \dots (t_m, x_m)$ we assume that $t_0 = 0$ and that all t_i are equidistant, i.e., that $(t_i - t_{i-1}) = (t_{i-1} - t_{i-2})$ for $2 \leq i \leq m$.

a) Segmentation: We search for points in time at which certain characteristics of a signal change significantly. Formally, a *segmentation* $Seg = (s_1, \dots, s_n)$ of S is a partition of S into partial signals such that $S = s_1 \cdot \ldots \cdot s_n$, i.e., S is the concatenation of the segments. Our approach is inspired by the algorithm presented in [2]: We use two adjacent sliding windows to identify changing behavior in a signal. For each window we compute a set of features, e.g., sum of all values, contained frequencies, range of values, and bias-corrected sample variance. Segment boundaries are then points where features in both windows differ significantly. All features can be weighted in order to account for different types of signals. We train the weighting using an optimization algorithm and a manually performed optimal segmentation as the target.

b) Classification: After computing segments, we group computed segments into named classes (e.g., 'steady speed'), using already classified segments in a knowledge base as a basis. For a segmentation $Seg = (s_1, \dots, s_n)$ and a finite set of segment classes C, a *labeling* l of Seg is a function $l : Seg \to C$.

We compute l using the k-nearest neighbor method [8] based on already classified segments. For all segments, we compute feature vectors including different features, e.g., arithmetic mean, slope, maximum slope, minimum value, maximum value, maximum range, and bias-corrected sample variance. Similarity of segments is computed as Euclidean distance on normalized features. New segments are assigned to the type found most within the next k neighbors (or in case of non-uniqueness to the type of the next neighbor).

c) Model Construction: For a signal S, a segmentation Seg of S, and a labeling l of Seg, the *abstract trace* σ of S is the sequence $\sigma = l(s_1) \dots l(s_n)$. Most of the automated analyses we are currently performing work on abstract traces. We additionally transform those traces into more concise automata models, which are easier to analyze for engineers.

A model $M = \langle Q, I, F, T, l_T, l_Q \rangle$ is a labeled transition system where $Q \subseteq C$ is a set of states, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states, $T \subseteq Q \times Q$ is a set of transitions, and $l_Q : Q \to L$ and $l_T : T \to L$ provide state labels and transition labels (L is a set of labels).

We fold a trace σ into a model M by using the labels so that $l(s_1)$ becomes an initial state. The last label $l(s_n)$ of a trace becomes a final state. We introduce a transition t from q to q' if there is at least one transition from q to q' in σ, i.e., if for some index $1 \leq i \leq n$ in σ it holds

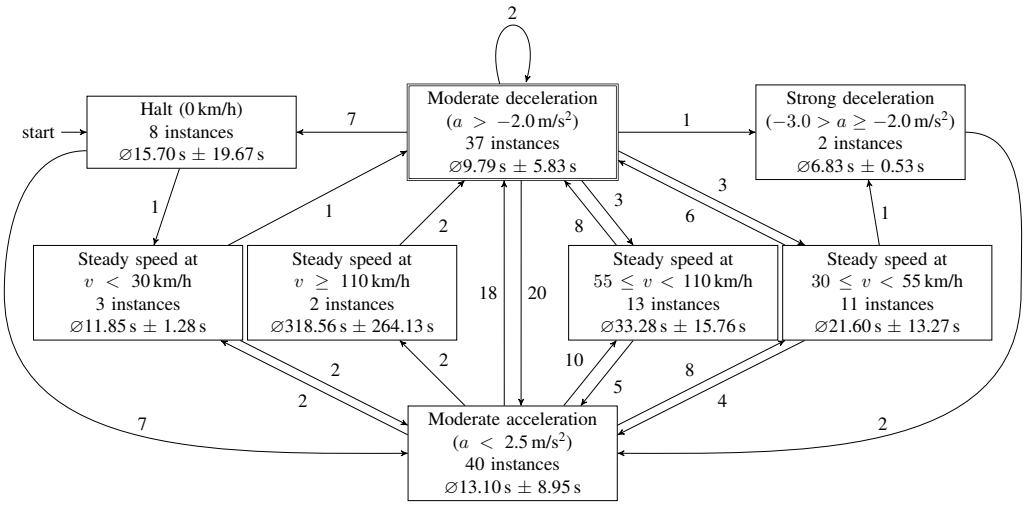

Fig. 2. Automaton model of the third road test. States are labeled with segment classes, number of instances in the analyzed trace, average duration, and std. deviation of duration. The initial state is marked with an extra incoming transition, the final state is marked by double lines. Transitions are annotated with their respective number of occurences.

that $l(s_{i-1}) = q$ and $l(s_i) = q'$. We label transitions with the number of occurrences in a trace and states with the number of occurrences, as well as the average duration and std. deviation of duration of corresponding segments. We do not define formal semantics for the automata models since we mainly use automata for visual validation of test cases.

We extend the above algorithm for constructing automata to sets of traces in the obvious way and maintain one automaton for all observed traces in the knowledge base (cf. Fig. 1). This *global* automaton contains all states and transitions observed during any of the road tests. It is used as a basis for computing state coverage and transition coverage of individual road tests.

III. Evaluation

We have evaluated the approach presented in the previous section using data recorded during six road tests. We recorded acceleration a and vehicle speed v with an overall length of more than four hours. Similar routes were driven in all tests, containing segments of city traffic as well as highway and freeway driving. Data was recorded using VAG-COM Diagnostic System (VCDS) [7] via the on-board diagnosis (OBD) connector of the vehicles.

Setup. We manually performed segmentation and classification of all recorded data and used this information as a reference when evaluating the single steps. Segmentation was done on the acceleration signal, classification was done using the speed signal, using the segments computed on the acceleration signal. The manually computed segmentation from one road test was used to train the settings of the segmentation algorithm, using simulated annealing. For the classification step we used the following types of segments:

1) Halt (0 km/h),
2) Moderate acceleration ($a < 2.5\,\text{m/s}^2$),
3) Strong acceleration ($a \geq 2.5\,\text{m/s}^2$),
4) Moderate deceleration ($a > -2.0\,\text{m/s}^2$),
5) Strong deceleration ($-3.0 > a \geq -2.0\,\text{m/s}^2$),
6) (Almost) full braking ($a \leq -3.0\,\text{m/s}^2$),
7) Steady speed at $v < 30$ km/h,
8) Steady speed at $30 \leq v < 55$ km/h,
9) Steady speed at $55 \leq v < 110$ km/h, and
10) Steady speed at $v \geq 110$ km/h.

When performing the automated classification of segments for a road test, we used the data from all other road tests as knowledge base. Finally, we generated one big model from all road tests and used it as a reference for analyzing smaller models for the individual tests.

Results. Fig. 3 shows the computed segment boundaries of a representative excerpt of one road test. Correctly found segment boundaries are shown as dashed lines, dotted lines mark boundaries not detected by the optimized settings, and dashdotted lines show incorrectly found boundaries. The results show that segmentation works reliably on the analyzed data. The example demonstrates that most boundaries are identified correctly.

Table I shows the results of the classification using the k-nearest neighbor method and different values of k. We report the duration of a road test, the number of segments per test, and the percentage of correctly classified segments for $k \in \{1, 3, 5\}$: as can be seen, $85\,\%$ to $95\,\%$ of segments are classified correctly in each experiment. It is notable that in each case the best results were obtained for a value of $k = 1$. This can be attributed to the relatively small number of already classified segments.

Table II shows the coverage of the global automaton by each road test. As can be seen, only four of six road tests cover all states. Moreover, no road test covers all observed transitions; transition coverage ranges from $44\,\%$ to $76\,\%$.

Finally, Fig. 2 shows the automaton model generated from the abstract trace of road test no. 3. The automaton is a concise representation of the trace: the model shows, e.g., the

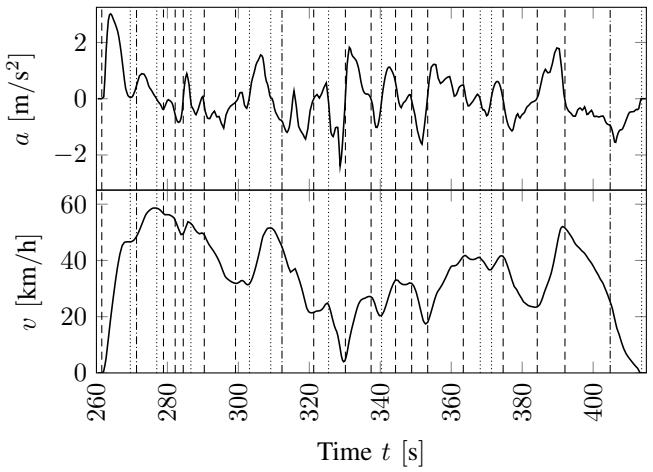

Fig. 3. Signal with computed segments: Speed v, Acceleration a. Dashed lines: correctly identified segment boundaries; dotted lines: missed boundaries; dashdotted lines: incorrect boundaries.

38 transitions between the states 'Moderate acceleration' and 'Moderate deceleration' very compactly.

Summarizing, already the basic implementations that we have used for the individual steps perform quite well and yield interesting results. The computed traces and models can help engineers validate test inputs for components that are generated through road tests. They show, e.g., that certain states or transitions are not covered by a road test.

IV. Conclusion

In this paper we have presented a method for validating test inputs for components of control software in the automotive domain. One challenge when testing these components is that test inputs are often generated through road tests by means of performing specific driving maneuvers. In many cases there is only an indirect connection between the inputs at the system level (e.g., position of the brake pedal) and the inputs to a tested component (e.g., negative pressure of a brake booster). In order to validate that a software component was tested sufficiently, engineers have to analyze recorded data after road tests. We have presented an approach for inferring automata models from data recorded during road tests. These concise automata are easier to analyze than hours of raw signal data: they exhibit specific states and transitions for different test scenarios, which allows engineers to understand how a component was exercised during a road test.

TABLE I
Classification rate for Segments using k-nearest neighbor

Road Test	Length [min]	Segments	$k=1$	$k=3$	$k=5$
1	45.73	150	93.33 %	92.00 %	91.33 %
2	41.86	198	88.89 %	87.88 %	86.36 %
3	39.57	117	96.58 %	95.73 %	95.73 %
4	34.96	99	93.94 %	89.90 %	86.87 %
5	39.20	146	91.78 %	91.78 %	90.41 %
6	44.60	214	88.79 %	86.45 %	87.85 %

TABLE II
Coverage of Global Automaton by Individual Road Tests

Road Test	State Coverage	Transition Coverage
1	100 %	54 %
2	90 %	62 %
3	80 %	44 %
4	100 %	62 %
5	100 %	66 %
6	100 %	76 %

As a next step, we want to address robustness of the individual methods for identifying and classifying segments, as well as for computing automata models. A bigger case study has to show how much the individual steps can be automated: E.g., the initialization of appropriate weights for the segmentation currently requires much effort. Improving these weights can be automated in the future. Moreover, segmentation may benefit from using information from the classification, e.g., by searching for matching parts of signals.

Classification currently relies on a knowledge base with a set of already classified segments. In order to classify segments without a knowledge base (e.g., for classifying signal data for the first time) variants of the k-means methods [4] may be used for determining initial segment classes.

Finally, we want to infer models that can be used for generating test cases or as a basis for more formal approaches (e.g., model checking). This will require finding a learning algorithm and a learning model that produce models with formal semantics and support statements about the quality of models. Furthermore, classification may benefit from using improved automata models as an additional input, e.g., for probabilities of subsequent transitions. Another interesting topic is the correlation between different signals: models that relate outputs and inputs could be used for closed-loop testing.

References

[1] Abbaspour Asadollah, Sara & Inam, Rafia & Hansson, Hans, *A Survey on Testing for Cyber Physical System*, 27th IFIP WG 6.1 International Conference (ICTSS), p. 194-207, 2015.

[2] Azami, H. & Mohammadi, K. & Behzad, B., *An Improved Signal Segmentation Using Moving Average and Savitzky-Golay Filter*, Journal of Signal and Information Processing, http://doi.org/10.4236/jsip.2012.31006, 2012.

[3] Ly, Daniel L. & Lipson, Hod, *Learning Symbolic Representations of Hybrid Dynamical Systems*, Journal of Machine Learning Research, 13:3585-3618, 2012.

[4] MacKay, David, *An Example Inference Task: Clustering*, Information Theory, Inference and Learning Algorithms, Cambridge University Press, ISBN 0-521-64298-1, p. 284-292, 2003.

[5] Mao, H. & Chen, Y. & Jaeger, M. & Nielsen, T.D. & Larsen, K.G. & Nielsen, B.: *Learning probabilistic automata for model checking*, 8th International Conference on Quantitative Evaluation of Systems (QEST), 2011.

[6] Niggemann, Oliver & Stein, Benno & Maier, Alexander & Vodencarevic, Asmir & Kleine Büning, Hans, *Learning Behavior Models for Hybrid Timed Systems*, 26th AAAI Conference on Artificial Intelligence, p. 1083-1090, 2012.

[7] Ross-Tech, LLC: *VAG-COM Diagnostic System (VCDS)*, http://www.ross-tech.com, last accessed at 12-09-2015.

[8] Runkler, Thomas A., *Data Analytics – Models and Algorithms for Intelligent Data Analysis*, Springer Vieweg, Wiesbaden, ISBN 978-3-8348-2588-9, 2012.