Collecting Simulation Scenarios by Analyzing Physical Test Drives

Pascal Minnerup
fortiss GmbH
An-Institut der TU München
Munich, Germany

Tobias Kessler
fortiss GmbH
An-Institut der TU München
Munich, Germany

Alois Knoll
Robotics and Embedded Systems
TU München
Munich, Germany

***Abstract*—Ensuring safe operation of autonomous vehicles requires testing them including critical combinations of obstacle configurations plus sensor and actuator inaccuracies. A method for testing inaccuracy combinations has already been published by the authors. This paper enhances the capabilities of the method by automatically collecting scenarios from physical vehicle drives that are relevant for further analysis. For such situations, a state trace including all variables of the whole planning and control system is stored together with environment information. The stored data is the input for further analysis. An implementation of this approach is demonstrated using a simulation and a full size vehicle.**

I. Introduction

As driver assistance systems become more complex the set of possible behaviors consequently increases. Some behaviors are explicitly undesired, like a collision with an obstacle. Testing the system strives to ensure that such behavior does not occur. This paper focuses on testing the planning and control system, as it has a significant impact on the actual behavior of the vehicle. The planning and control system receives the obstacle map and the current vehicle state as input and has to output commands for the motor, the brake and the steering wheel leading to some specified behavior of the vehicle. It plans these commands based on an abstract model of the car and its environment. All three: the input, the reaction to the output and the actual behavior of the car deviate from the ideal model. The planning and control system has to cope with these differences by putting out commands that lead to desired behavior even in the worst case combination of these deviations.

Testing, whether the planning and control system fulfills this requirement is challenging, as there are many different scenarios to be tested and for each scenario there is a number of sensor and actuator inaccuracy combinations growing exponentially with the length of the simulation. Coping with the exponentially large set of inaccuracy combinations has been regarded in [1]. The effectivity of the concept proposed in that paper depends on the right scenarios being tested. Therefore, the present paper investigates a way to find these scenarios. Each scenario consists of the initial configuration of the vehicle, the obstacle distribution and potentially other factors like the currently followed reference path. Additionally to using requirement specifications, the present paper advocates to automatically collect relevant scenarios from physical drives performed with the target vehicle. The applied recording mechanism allows to set up a simulation in which sensor and actuator responses are variated for a systematic test.

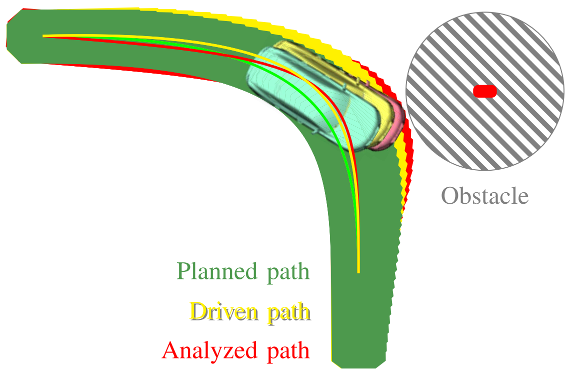

Fig. 1: Passing an obstacle scenario: The actually driven path (yellow) deviates from the planned path (green) but the worst case would have been the red path

Fig. 1 shows an example of a scenario that can be extracted from a physical test drive. The green curve and the green area are the planned path and the area the vehicle covers when following exactly the planned path. Due to sensor and actuator inaccuracies, the car actually follows the yellow path and therefore almost violates the safety distance (hatched circle) around a pedestrian (red rounded rectangle). An offline analysis shows that with worse inaccuracies (red covered area), the car might have actually entered the pedestrians safety distance. Thus in the worst case the vehicle would have hit the pedestrian.

II. Related work

Prior research includes work to extract scenarios that are worth analyzing from different sources like

- requirements specifications,
- detailed accident information or
- vehicle test drives.

A. *Requirements specifications*

The first available source of information for extracting simulation scenarios is the requirements specification. In the Darpa challenge, the competitors were required to navigate in a pre defined environment. This environment was distributed to the research team who based their simulation environments on it. For example the simulator of [2] was able to load these files and enhance the simulation by further details.

Other teams conduct research about simulation in the requirements phase. The authors of [3] propose to analyze scenarios in the requirement specification phase and use Novelty Search in order to find maximally different behaviors of the software system.

The method proposed in the present paper includes analyzing scenarios based on the requirement specification and additionally collects scenarios from other sources.

B. Vehicle test drives

Additionally to requirements specifications, data can be collected in physical test drives. Automotive development software like EB Assist ADTF[1] or dSPACE ControlDesk[2] offer tools to record data from physical test drives and replay it in an offline environment. This is one of the standard procedures in industrial development projects. Similar techniques are applied in research projects. The authors of [4] highlight the ability to recreate environments in which their planning system failed. This does not include the state of the planning system. Their competitors ([5]) also focused on replaying and solving situations showing defects without varying collected scenarios. Another use case for collecting data for autonomous vehicles in physical test drives are benchmark suites like KITTI [6]. Physical test drives are also the foundation of the method described in the present paper. The collected data is used to simulate different variations of the observed scenario. In contrast, the approaches listed above are limited to replaying the exactly same scenario.

The author of [7] tries to directly prevent unsafe trajectories online using an abstract vehicle model. By supporting an arbitrary simulation environment, the method in the present paper aims to reduce the reality gap. This is also intended by the online simulation used in [8] for identifying faulty robots.

C. Detailed accident information

Finally, scenarios from previously physically driven situations can be obtained from detailed accident information gathered by different countries. One such data base is the GIDAS [9] (German In- Depth Accident Study) database in Germany. Other countries maintain similar databases, which the project described in [10] tries to harmonize.

The authors of [11] describe how to use the GIDAS data base for extracting information about what happened several seconds before a crash occured. This information is then used in a simulation in order to evaluate, whether different sensors would have been able to detect the opposing vehicle. Based on this method, [12] simulates the same accident scenarios with a different sensor equipment. In order to improve the quality of scenario reconstruction, [13] applies a Monte-Carlo analysis for a better estimation of parameters necessary for the simulation environment.

A disadvantage about current accident databases is the small amount of information available compared to data recorded in test drives.

III. Scenario Extraction

In test drives or daily usage, a vehicle can encounter situations that might lead to an accident but do not because the sensors and actuators perform better than the worst case. As sensors and actuators do not deliver their worst case performance in most cases, it is reasonable to assume that such scenarios occur more often than actual accidents. Hence, if such scenarios can be detected without a following accident, many defects can be resolved before an accident occurs.

As mentioned in the previous section, reviewing recorded data from test drives and checking it for unusual behavior is best practice in the automotive industry. However, the critical scenarios described above are not necessarily unusual concerning the occurred controller errors or steering activity. Plus, if the car moved close to an obstacle it might still have been within the planned safety distance and thus have behaved as specified. Finally, just from typically recorded data it is difficult to tell, whether the car might have behaved worse than observed provided slightly different conditions. The goal of the concept presented in this paper is to collect these scenarios, store information about them, reproduce the scenario in a simulation environment and systematically check how the system would have behaved in the presence of worse sensor and actuator inaccuracies. The collected scenarios can also be replayed with later or other versions of the planning and control system.

A. Sources for scenario extraction

There are various situations in different levels of the development process of the autonomous vehicle providing scenarios worth further analysis. Some groups of such situations are:

- Test drives executed by developers
- Test drives according to requirement specifications
- Drives performed in a prototype testing program
- Regular usage by customers

Developers regularly drive the experimental vehicle in order to test functions they are currently developing. The test drives are usually less systematic than independent vehicle tests but based on knowledge and experience that is not necessarily available during requirements specification or test drives by independent test experts.

The second group mentioned above are test drives based on the requirements specification. These test drives systematically cover the requirements and can be additionally supported by reference sensor systems or known environment maps.

Furthermore, before the final customers use the product, a limited set of almost final prototypes is produced and used. These prototypes can still contain some extra devices for gathering development data. Plus, the prototypes are used in

[1]http://automotive.elektrobit.com/products/eb-assist/adtf/

[2]http://www.dspace.com

a similar way as by end customers and hence create similar data before market start.

Finally, scenarios can be gathered during usage by the final customers in an anonymized form. Data collected from this source cannot prevent the distribution of faulty vehicles, but it can help eliminating defects before they lead to undesired behavior experienced by a customer. This source is particularly useful, as it provides the largest amounts of data.

This paper focuses on collecting scenarios in which all sensors and actuators work properly, except for inaccuracies. Another usage of the proposed concept is to collect scenarios in which some sensors or actuators failed completely. Software engineers can then analyze, whether the fail safe concept would also have worked with different inaccuracies of the remaining sensors and actuators. The same principle can be applied to unexpected behaviors of other traffic participants: Was the autonomous driving function still in control of the situation?

For all groups mentioned above, the process for extracting scenarios suggested in this paper consists of four stages:

1) Record all necessary data
2) Identify relevant scenarios
3) Update the error model
4) Analyze the identified scenarios

These stages are explained in the following sub sections.

B. Record data

Without the concept proposed in this paper, the typically recorded data includes data visible at the components' interfaces. This includes the position and velocity of the car, the applied actuator actions or the information sent over the vehicle network, like CAN messages. This information can be replayed in a simulation environment in order to review how the system components behave in the exactly same scenario. However, it is more challenging to use this data to create modified scenarios, which is necessary for systematically analyzing the effect of sensor and actuator inaccuracies. Therefore, we suggest to use the concept proposed in [1] for recording data. In this concept, the whole state of the planning and control system including internal variables is stored and can be restored in a simulation environment. This way, no state variable that might be important is missed. Plus, when initializing the simulation no default values for non stored information have to be found. Fig. 2 shows the scenario also depicted in Fig. 1. At every marker position (small gray circles), the state of all planning components is saved.

In the pure analysis usage of the cited concept, the saved and loaded components are identical. For scenario extraction, the saved and restored configurations differ in the representation of the environment as depicted in Fig. 3. During the recorded drive, the environment is the *Physical Environment* (bottom left green box) of the car. During analysis it is a *Simulation Environment* (top left red box). Therefore the *Scenario Extractor* (bottom red box) distinguishes between saving and restoring of the *Planning and Control* system (right boxes) and the *Perceived/Simulation Environment*. The *Planning and Control*

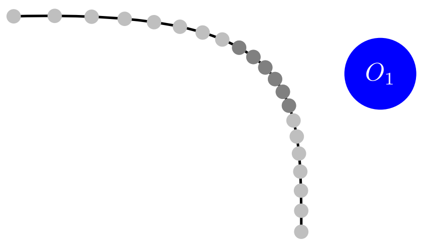

Fig. 2: Points (gray) at which the planning and control system is saved while following the path described in Fig. 1. The dark grey states are regarded by the analysis, as the car is close to the obstacle.

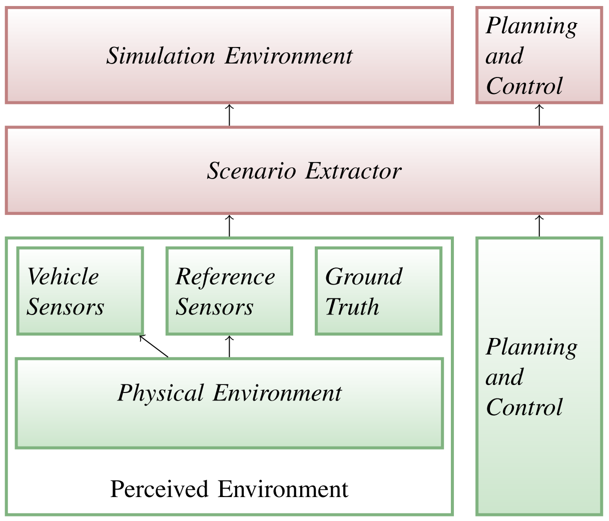

Fig. 3: The software system and its environment during the physical drive (green) are converted to a representation in the simulation environment (red). The perceived environment can be converted based on different sensor measurements or prior known ground truth information.

system can be stored and restored the same way as for a pure analysis usage. The state of the *Simulation Environment* has to be computed corresponding to the perceived environment.

The main information needed for recreating the simulation state is

- the vehicle state and
- the environment obstacle map.

The vehicle state consists of the position, the orientation, the speed and the acceleration of the vehicle. Additionally it includes dynamic effects like drift and more detailed properties, like for example the exact motor state. Some of this information is not available for the scenario extraction and some information can not be used by a specific simulation environment. The remaining information is extracted and used to set the simulation state. Furthermore, the environment obstacle map is needed, which tells where the obstacles are. There are three potential sources of this information:

- The vehicle sensor system,
- a reference sensor system or
- some ground truth information.

The vehicle sensor system is available in all scenarios described at the beginning of section III. It can measure the basic vehicle state and detect static and dynamic obstacles. The disadvantage is that the quality including the level of detail of the measured values is limited. The effect of slightly wrong measurements is that a slightly different scenario will be analyzed which might still produce valuable results. The second source of information is a reference sensor system. For example, prototype vehicles may be equipped with high quality laser sensors, better differential GPS or other highly accurate sensors. These sensor values are not available for the tested planning and control system, but can be used for storing the current vehicle state. Finally, for some tested scenarios, the exact environment map is available, for example, because the map of the surrounding building is known or the whole test drive happened in a controlled environment. This is the most accurate information, but available only for a limited set of the tested scenarios. Additionally, for many locations, 3D-geodata can be used for scenario reconstruction as suggested in [14].

The previous paragraphs regard different sources for extracting environment information. Additionally, the environment can be modeled in different levels of abstraction. For example, the red rounded rectangle in Fig. 1 depicts a pedestrian. This pedestrian can be represented by a complex pedestrian simulation. Alternatively, it can be modeled as an entity that can randomly move $2\,\mathrm{m}$ in any direction before the driver assistance system reacts. This would be the same effect as a static circle shaped obstacle with a radius of $2\,\mathrm{m}$.

C. *Identify relevant scenarios*

Not all recorded scenarios need to be analyzed when searching for a specific undesired behavior. For the undesired behavior "collision with an obstacle", scenarios need to be considered, in which the vehicle gets close to any obstacle. In the analysis framework proposed in [1], the same mechanism used to identify undesired behaviors can be used for identifying scenarios relevant for further analysis. For recognizing scenarios that get close to an obstacle, the *Scenario Extractor* performs a collision check between the obstacle map and the vehicle shape enlarged by a parameterizable distance. The distance should be chosen large enough to avoid sorting out relevant scenarios. A larger distance leads to more scenarios being analyzed, but not to wrong results. Being close to an obstacle is one example of an event triggering further analysis. If such an event is triggered, the *Scenario Extractor* stores the time at which the event happened. In Fig. 2 the event occurred in the middle of the dark gray states. The analysis will then consider some seconds before and after this stored time. This is depicted by the dark gray circles.

The behavior "collision with an obstacle" is a typical analysis goal. However, other undesired behaviors can be specified as well. For example an aborted parking attempt or excessive steering in a non dangerous situation. Furthermore, different triggers for identifying scenarios relevant for these behaviors can be used. One example is the amount of steering applied per second.

D. *Update Error model*

The analysis is based on an error model. For example, the performed acceleration might happen later and with a different intensity than requested. Both, the delay and the intensity are limited by a parameter of the error model. The first step after recording relevant scenarios is to check, whether the observed behavior of the vehicle can be explained by the vehicle model and the currently assumed error model. If not, the model needs to be updated and scenarios analyzed based on the previous error model need to be analyzed again.

E. *Analyze identified scenarios*

After recording relevant scenarios and updating the error model, the recorded scenarios are analyzed using the framework proposed in [1]. For doing this, the framework loads the search tree built while recording the scenarios. This is possible, because the recording step described previously stores the full state of the planning and control system using the mentioned framework. The states included in the search tree are put into an "Open States" data structure deciding which states are restored and analyzed first. This data structure can for example be grid based. The proposed algorithm will restore each node to a state for the planning and control system and for the simulation environment. Then, it will command the simulation environment to show a new kind of error pattern. After a specified time interval, the resulting state of the planning and control system and the simulation environment is stored and put into the "Open States" data structure. This method allows to efficiently test the planning and control system against combinations of sensor and actuator inaccuracies. Further details are explained in [1]. For the nodes already stored during the scenario extraction, the state of the simulation environment has to be extracted from sensor measurements. For this process, the simulation components can set their states based on sensor information. The analysis framework derives this information from the measurements during the physical test drive. For example, the vehicle model simulation can set its state based on the position information derived from vehicle sensors.

Additionally to analyzing the exact obstacle configuration experienced in the physical test drive, a future step would be to vary the extracted scenarios based on optimization algorithms as suggested in [15].

The result of running the analysis framework is the potentially empty set of sensor and actuator error combinations leading to undesired behaviors. The software engineer can use this information as pointed out in the next section.

IV. Using the Gained Information

If the analysis finds no undesired behavior, the system could have coped with worse error combinations and no further action is required. If it determines that some error

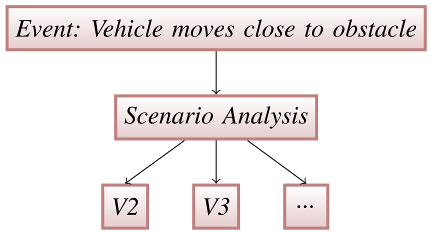

Fig. 4: A vehicle experiences a relevant scenario, which is analyzed by some computing center. The results are distributed to all vehicles.

combination could have provoked undesired behavior some reaction is necessary: On the one hand, the problem has to be fixed by adjusting either the planning and control system or the error model. On the other hand, the gained information can be useful for other vehicles. Fig. 4 depicts how the information can be spread: Some vehicle recognizes a relevant situation and uploads the recorded data about this situation to a central computer cluster, which analyzes the scenario. If this cluster determines that the encountered situation might have been dangerous it generates a suggested counter measure. The generation of the counter measure may happen either fully automatically or be supported by human decisions. Reasonable counter measures include

- updating the vehicle software,
- adapting parameters of the vehicle software
- warning of the particular geographic position, or
- deactivating the corresponding function

Updating the vehicle software or software parameters is the solution if the situation was dangerous due to some software defect or wrongly calibrated parameters. For development vehicles, this will typically be the case. For production vehicles, the dangerous situation might be due to the specific geographic properties. For example a road curve might be very slippery. In this case other vehicles should be warned of this position. If neither a software adjustment, nor avoiding geographic positions can avoid a significant danger, the final option is to deactivate the corresponding driver assistance function until a solution is found.

Additionally, the gained information can support the development process of the next vehicle generation. Critical scenarios can be added to a regression test suite of the development branch. Plus, for large changes it can be checked, whether the system still produces satisfactory results in all previously analyzed scenarios. The tests can also be applied to vehicles with different hardware including different vehicle dynamics and different sensor and actuator sets.

V. Experimental Results

The proposed system has been tested by recording a $188\,\mathrm{s}$ test sequence and analyzing the data in a simulation environment. For this sequence, a full size vehicle with a planning and control system based on [16] was used. In the first step, the vehicle is commanded to execute the tasks depicted in

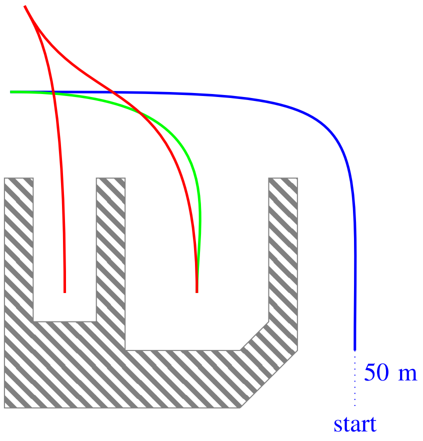

Fig. 5: Sequence with three consecutive moves (blue, green, red) around some obstacle (hatched area) in which the scenario extraction is tested.

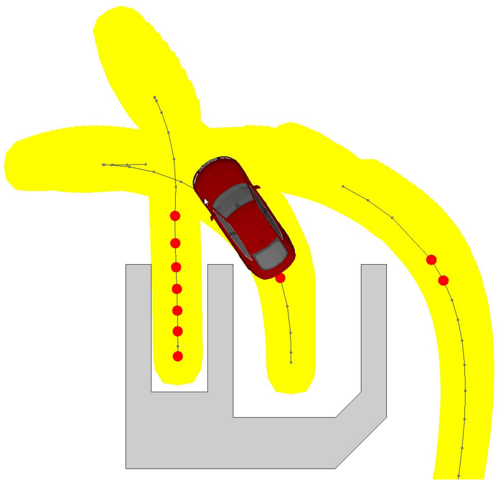

Fig. 6: Visualization of the data recorded on a physical vehicle and loaded into a simulation environment. The covered area of the vehicle (yellow) never intersects with the obstacles (gray area), but sometimes gets close to it (red circles).

Fig. 5. The hatched gray area represents the obstacle map. The blue, green and red colored lines are the trajectory followed by the vehicle in the first, second and third move. At first the vehicle drives $50\,\mathrm{m}$ (blue dotted line) at a low speed without any nearby obstacles until it reaches the obstacle area and turns left (blue line). After some waiting time, it parks into a broad parking lot (green line) and, finally, it parks into a narrow parking lot (red line). The executed actions correspond to a non systematic test sequence that might be executed by a developer. The stored data requires $9.16\,\mathrm{MB}$ hard disk storage.

Fig. 6 shows the visualization of the physical vehicle driving sequence loaded into the simulation environment. The yellow area has been covered by the car during the test drive. At the depicted point in time, the distance between the vehicle and the obstacle map (gray area) is lower than $0.4\,\mathrm{m}$, therefore this

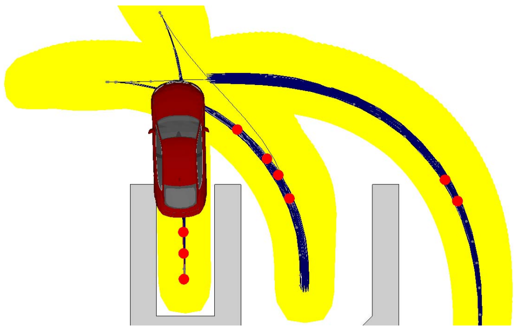

Fig. 7: Actuator inaccuracies lead to different trajectories (blue lines) covering the yellow area and leading to a collision in the narrow parking lot.

point in time is identified to be relevant for further analysis (compare Section III-C). One second later is still identified to be relevant as depicted by the red circle behind the car. Additionally to these states two more time intervals have been identified as relevant (red circles): One at the first left turn and one in the narrow parking lot. For each of these time intervals $15\,\mathrm{s}$ before and $5\,\mathrm{s}$ after the interval are analyzed using different combinations of inaccuracies. This is depicted by the black lines before and after the red circles.

Next, the recorded data is analyzed as explained in Section III-E. The stored states include the correct internal states of all planning and control components. For the controller, this includes the current reference trajectory and the tracking status. The planning components remember the planned path and also the previously computed reference trajectory. Hence, no initial state has to be specified manually and the state of the planning and control system in the replaying environment equals the state in the recording environment. Based on these saved states, the framework takes $1.5\,\mathrm{h}$ to simulate $100.000\,\mathrm{s}$ around time points that have been identified as relevant. The blue lines in Fig. 7 represent the trajectories the vehicle followed for different combinations of actuator inaccuracies. In the narrow parking lot, one of these inaccuracy combinations result in the collision depicted in Fig. 7. For the other two situations, the algorithm finds no collision, hence the area covered by the vehicle during the analysis (yellow) does not intersect with the obstacle map.

VI. Conclusion and Outlook

In the present paper we presented a new concept for generating test scenarios that are useful for the development of autonomous vehicles. The method helps to monitor both prototype and consumer vehicles. In these scenarios, the autonomous driving system is tested against combinations of sensor and actuator inaccuracies. This way, critical scenarios can be detected, even if they did not results in undesired behavior in the observed case. The concept can also be used to mark dangerous locations for other vehicles. The proposed method was demonstrated based on a real vehicle test and a simulation environment.

This paper focuses on analyzing the possible effect of inaccurate, but well working sensors and actuators on situations collected in test drives. Additionally, the effect of sensor failures could be analyzed. If one sensor fails, the remaining sensors and actuators are still inaccurate leading to different possible results. This could be investigated systematically. Furthermore, unexpected behavior of pedestrians could trigger a more detailed analysis, whether this might have lead to an accident.

References

[1] P. Minnerup and A. Knoll, "Testing autonomous driving systems against sensor and actuator error combinations," *Intelligent Vehicles Symposium Proceedings*, 2014.

[2] J. Rojo *et al.*, "Spirit of Berlin: An Autonomous Car for the DARPA Urban Challenge Hardware and Software Architecture," *Retrieved*, vol. 12, no. 02, p. 2010, 2007.

[3] A. J. Ramirez, A. C. Jensen, B. H. C. Cheng, and D. B. Knoester, "Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings*, ser. ASE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 568–571.

[4] A. Bacha *et al.*, "Odin: Team VictorTango's entry in the DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 467–492, 2008.

[5] B. J. Patz, Y. Papelis, R. Pillat, G. Stein, and D. Harper, "A practical approach to robotic design for the DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 528–566, 2008.

[6] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Jun. 2012, pp. 3354–3361.

[7] M. Althoff, "Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars," Ph.D. dissertation, Technische Universität München, Munich, Feb. 2010.

[8] A. G. Millard, J. Timmis, and A. F. Winfield, "Run-time detection of faults in autonomous mobile robots based on the comparison of simulated and real robot behaviour," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sep. 2014, pp. 3720–3725.

[9] D. Otte, C. Krettek, H. Brunner, and H. Zwipp, "Scientific approach and methodology of a new in-depth investigation study in germany called gidas," *Technical Conference on the Enhanced Safety of Vehicles*, 2003.

[10] D. Ockel, J. Bakker, and R. Schoeneburg, "An initiative towards a simplified international in-depth accident database," *Berichte der Bundesanstalt fuer Strassenwesen*, 2013.

[11] C. Erbsmehl and L. Hannawald, "Simulation realer Unfalleinlaufszenarien der German In-Depth Accident Study (GIDAS). Erstellung und Nutzen von" pre crash scatter plots"," *VDI-Berichte*, 2008.

[12] C. Erbsmehl, "Simulation of real crashes as a method for estimating the potential benefits of advanced safety technologies," *Conference on the Enhanced Safety of Vehicles*, 2009.

[13] D. P. Wood and S. O'Riordain, "Monte Carlo Simulation Methods Applied to Accident Reconstruction and Avoidance Analysis," SAE Technical Paper, Tech. Rep., Mar. 1994.

[14] B. Bartels, C. Erbsmehl, and L. Hannawald, "Reconstruction of accidents based on 3D-geodata," *Berichte der Bundesanstalt fuer Strassenwesen*, 2013.

[15] O. Buhler and J. Wegener, "Automatic testing of an autonomous parking system using evolutionary computation," *Society of Automotive Engineers Inc.*, pp. 115–122, 2004.

[16] D. Lenz, P. Minnerup, C. Chen, and E. Roth, "Mehrstufiges Planungskonzept fuer pilotierte Parkhausfunktionen," in *30. VDI/VW-Gemeinschaftstagung "Fahrerassistenz und Integrierte Sicherheit 2014"*, Wolfsburg, Germany, 2014.