applied sciences

Article

Autonomous Driving—A Crash Explained in Detail

Johannes Betz [1,*], Alexander Heilmeier [1], Alexander Wischnewski [2], Tim Stahl [1] and Markus Lienkamp [1]

1. Institute of Automotive Technology, Technical University of Munich, Boltzmannstr. 15, 85748 Garching b. München, Germany; heilmeier@ftm.mw.tum.de (A.H.); stahl@ftm.mw.tum.de (T.S.); lienkamp@ftm.mw.tum.de (M.L.)
2. Institute of Automatic Control, Technical University of Munich, Boltzmannstr. 15, 85748 Garching b. München, Germany; alexander.wischnewski@tum.de

* Correspondence: betz@ftm.mw.tum.de

Received: 2 October 2019; Accepted: 19 November 2019; Published: 26 November 2019

Abstract: Since 2017, a research team from the Technical University of Munich has developed a software stack for autonomous driving. The software was used to participate in the Roborace Season Alpha Championship. The championship aims to achieve autonomous race cars competing with different software stacks against each other. In May 2019, during a software test in Modena, Italy, the greatest danger in autonomous driving became reality: A minor change in environmental influences led an extensively tested software to crash into a barrier at speed. Crashes with autonomous vehicles have happened before but a detailed explanation of why software failed and what part of the software was not working correctly is missing in research articles. In this paper we present a general method that can be used to display an autonomous vehicle disengagement to explain in detail what happened. This method is then used to display and explain the crash from Modena. Firstly a brief introduction into the modular software stack that was used in the Modena event, consisting of three individual parts—perception, planning, and control—is given. Furthermore, the circumstances causing the crash are elaborated in detail.By presented and explaining in detail which software part failed and contributed to the crash we can discuss further software improvements. As a result, we present necessary functions that need to be integrated in an autonomous driving software stack to prevent such a vehicle behavior causing a fatal crash. In addition we suggest an enhancement of the current disengagement reports for autonomous driving regarding a detailed explanation of the software part that was causing the disengagement. In the outlook of this paper we present two additional software functions for assessing the tire and control performance of the vehicle to enhance the autonomous.

Keywords: Autonomous vehicle; autonomous system; intelligent transportation systems; vehicle crash; advanced driver assistance systems (ADAS); performance evaluation; path planning; automatic control

1. Introduction

A WHO statistic showed that in the year 2018, 1.34 million people worldwide lost their lives due to traffic accidents [1]. That is more than 3000 fatalities per year, in Germany only [2]. In retrospect, over the last decades, there was a continuous decrease of fatalities in car accidents, which can be traced back to laws like the seat belt obligation, higher fines, the prescribed use of motorcycle helmets, and speed reductions. In addition, this reduction comes also due to the increasing integration of active and passive safety functions in vehicles, such as airbags, anti-lock braking system (ABS), or electronic stability program (ESP) [3]. In particular, the active safety systems that support the driver in his driving task stand out. Accident statistics demonstrate that about 90% of traffic accidents are caused by human error and only 10% by technical errors [4]: carelessness, microsleep, inappropriate speed, or driving too close to other vehicles. This number of accidents can be decreased by replacing the driver with an autonomous system [5].

In the year 2019, the vehicles that are used with corresponding driver assistance systems to support the driver are predominantly assigned to autonomy level three [6].The goal is the development of an autonomous level five vehicle which can eliminate the influence of the driver completely. Nonetheless, from a technical point of view, a fully autonomous level five vehicle is still too complicated and is therefore focus of the current development in the automotive industry and research of universities. However, there is one crucial question that has to be discussed simultaneously: what if an accident happens while the vehicle is driving fully autonomously? Until today, we do not have much experience with crashes of autonomous vehicles and therefore we do not know how to evaluate and address the issues of the vehicle—or better said the issues of the software—that lead to an accident. It is important to set up metrics and explanations that have to be published afterwards similar to aviation: the countries in which the aircraft concerned have been designed, constructed, certified, or registered may participate in the investigation [7]. In addition to the country itself, each country has a Bureau of Aircraft Accident Investigation that is responsible for air accident and incident investigation. The evaluation of the respective incident investigated and the resulting conclusion and safety recommendation should not serve to clarify the question of guilt or liability. Rather, the sole aim of the technical investigation is to gain knowledge with which future accidents and incidents can be prevented [7].

To break new ground for autonomous vehicles in this research area we will present in this paper a detailed explanation of a crash conducted by an autonomous vehicle. In this research we used an autonomous racecar provided by the Roborace company [8]. This racecar is used to develop and test a holistic software stack for autonomous driving. With this racecar the the real-time capability, performance, and reliability of autonomous driving algorithms can be tested on high accelerations and high velocities. Testing on the safe space area on a racetrack gives the possibility that the gained experience can be used for further development of autonomous driving functions for conventional production vehicles. In this paper we will explain the structure of the software stack and the reasons for the accident. In addition we will give recommendations about how to address future autonomous crashes and what is mandatory regarding the software and safety enhancement.

Our key contributions are the following:

- Presentation of a general method for describing autonomous disengagements or crashes;
- detailed evaluation of the crash causes by using the new presented method;
- detailed description of the crash of an autonomous racecar while it was driven by the software;
- derivation of important safety features for autonomous driving;
- derivation of new software functions to prevent such a crash.

2. Related Work

In 2019, the so-called Roborace Season Alpha [8] took place at various locations with the aim of testing possible competition formats using the Devbot 2.0. The events could be classic race formats similar to Formula 1 (testing; qualifying; race), precision races similar to rally events, or hillclimb events. For these events, the Technical University of Munich (TUM) team developed an entire autonomous software stack that includes different software modules [9,10]: a perception part for recognizing static and dynamic objects, a dynamic trajectory and behavior planner for high-speed overtaking maneuvers, and a highly adapted control module. The basis for this software structure was inspired by Pendelton et al. [11]. It matches the special hardware structure of the Devbot but is also generic enough to transfer it to other vehicles. Searching in the state of the art in autonomous driving software, there are just a few holistic software stacks that depict all necessary autonomous driving functions. The Autoware software [12,13] is based on the robot operating system (ROS) and provides a rich set of self-driving modules composed of sensing, computing, and actuation capabilities. Although Autoware is used in different research projects and autonomous vehicles, it is unknown how the different software modules work together. Another stack is the Apollo software [14] which has a highly performant, flexible architecture that should help to accelerate the development, testing, and deployment of autonomous vehicles. The Apollo 5.0 software is supplied with modules for perception, localization and control of the vehicle and offers a reference hardware for getting started with the software framework. Similar to the Autoware software, it is unclear where and in which vehicles the Apollo software is used.

Only a few existing publications cover autonomous driving crashes or problems in autonomous vehicles. The department of motor vehicles in the US state California [15] has received 186 autonomous vehicle collision reports from different companies like Zoox, GM or Waymo. In these predefined three-page reports the companies give a short overview what happened in the collision. In addition they have to supply information about the weather (clear, cloudy, etc.), lighting (daylight, dusk, etc.), roadway surface (dry, wet, etc.), roadway conditions (holes, loose material, etc.), and the special movement of the vehicle (stopping, left turn, etc.). A detailed analysis and evaluation of these incidents are discussed in Favaro et al. [16,17]. Although this database gives a good overview of what can happen, it does not explain why it happened and what part of the software in the autonomous vehicle failed. A more detailed source are the disengagement reports for every year that are supplied by the same authority [18]. Here, every company hands in detailed statistics about the disengagements of its cars. In comparison to the collision reports it is explained which part of the software (e.g., wrong detection of an object, or faulty motion planning) failed. A detailed analysis of these reports in the year 2015 and 2016 has been done by Favaro et al. [19]. Breaking down the disengagements into macro-categories, they found out that 52% of the disengagements had to do with a failure of the system. In these system failures, they evaluated that 25% of the disengagements were caused by a software discrepancy, followed by a perception discrepancy (11.6%), and the motion planning not being ready (10.9%).

Besides these numbers that are recorded due to the development phases of level four and level five vehicles, we currently have companies that are enabling autonomous driving on level two, e.g., Tesla, Audi, or Uber. For example, the Tesla Autopilot enables traffic-aware cruise control, an automatic steering for keeping the car in the current lane, as well as automatic lane changes and side collision warning [20]. However, in 2016 the first death of a Tesla driver happened while he was driving with the autopilot when he crashed into a truck [21]. Although it was the driver's responsibility to keep an eye on the road and do not let the car drive completely alone, the car failed to detect the truck trailer as an obstacle because of its "white color against a brightly lit sky" and the "high ride height" [21]. In March 2018, Elaine Herzberg was the first recorded case of a pedestrian fatality involving an autonomous test car operated by Uber. Similar to the Tesla accident, the perception module of the vehicle failed to classify the woman walking on the road correctly: first, the car classified the woman as an unknown object, then as a vehicle, and finally as a bicycle, whose path it could not predict [22]. Just 1.3 s before the impact, the self-driving system realized that an emergency braking was needed [22]. Although this fatal crash could have been prevented by the safety driver, the software of an autonomous vehicle failed again. In addition, a detailed explanation from Uber why their software failed is again not available. The most detailed answers about what happened can be found on Wikipedia where different articles are combined [23].

The description of all this related work is necessary for understanding that in the scientific community almost no publications exist that explain autonomous driving software failures or crashes in detail. The only exception in this field is Fletcher et al. [24]. In the 2007 DARPA Urban Challenge the autonomous Land Rover "Talos" from the MIT team crashed with the autonomous vehicle "Skynet" from the Cornell team. Although it was a low-speed accident, the root cause could be identified in the system design of each vehicle software. Fletcher et al. [24] gives a system-level description of both autonomous vehicles with detail on the sub-systems and algorithms. In addition a brief summary of robot–robot interactions during the race is presented, followed by an in-depth analysis of both robots' behaviors leading up to and during the collision. To sum it up, autonomous driving is very present in the technical development and in the media, but as well as it is present in the current discussion it lacks detailed explanations why the software of an autonomous vehicle can fail.

3. Research Design—A General Method for Describing Autonomous Disengagements

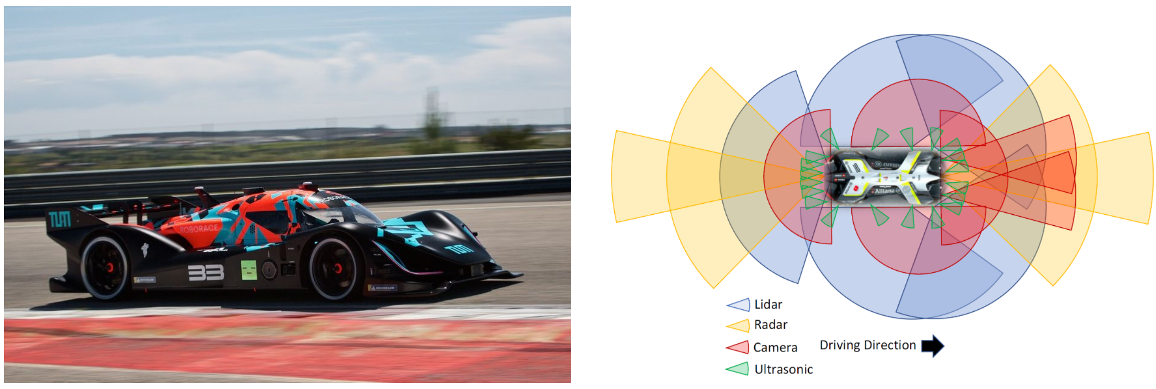
To sum it up, autonomous driving is very present in the current technical development and research of vehicle manufacturer and vehicle supplier, but as well as it is present in the current discussion it lacks detailed explanations why the software of an autonomous vehicle can fail. We addressed in the state of the art that we are missing a method that we can use to describe an autonomous disengagement, malfunction or crash in detail. With the knowledge from the sate of the art we present our research design for this paper. We propose for all upcoming research in the field of autonomous disengagements, malfunctions, and crashes the following general method that is valid for any case. If this general method is used, a comparison with other autonomous driving crashes can be derived more easily. In our method different steps need to be done:

1. **Background knowledge—autonomous vehicle presentation:** first of all, an overview of the autonomous vehicle must be given. It needs to be displayed what kind of vehicle was used (vehicle purpose e.g., passenger vehicle) and what are most important vehicle parameters (e.g., vehicle mass). In addition, it needs to be presented what kind of sensors and electrical control units (ECUs) are integrated in the vehicle. Therefore for others the comparison to their own vehicle can be assured.
2. **Background knowledge—autonomous software stack explanation:** The central element of an autonomous vehicle is its software. To understand a misbehavior of an autonomous vehicle it needs to be explained of what parts an autonomous software consists of. In addition it needs to be explained how these parts communicate with each other and what kind of interfaces are used.
3. **A step-by-step display of the crash chronology:** in the next step it needs to be explained what happened in the crash. First of all an overview of the situation (e.g., describing the scenario, describing the area) needs to be given. After that, a detailed step-by-step explanation of what happened is essential for further understanding. From the knowledge of a detailed step-by-step vehicle behavior further explanations can be derived and theories from observation can be generated.
4. **Software explanation failure in detail:** if the autonomous driving disengagement can be lead back to an software issue it needs be the explained in detail what went wrong in this software. Each software part that failed and has contributed its part to the crash needs to be explained here. Beside this, it needs to be displayed why the integrated safety systems did not work. In this section we use real vehicle measurement data (accelerations, velocities, and vehicle states) to give a quantitative view on the software failure.
5. **Discussion of the software failure:** after a detailed insight into the software failures are given, an additional discussion about the software failures must be done to derive conclusions for autonomous driving software development. It needs to discussed here what precautions to prevent this specific problem can be integrated, what kind of enhanced error detection mechanisms needs to be implemented in the software, and how previous real and simulation tests can be done to detect those errors earlier.
6. **Outlook for further software enhancements:** in the last part an outlook for further software developments for the explained software stack needs to be given. It should be displayed what kind of future algorithms and software parts will be used to enhance the software so this kind of failure will not happen anymore.

4. Background Knowledge

4.1. The Autonomous Vehicle—Roborace Devbot 2.0

The work described in this paper is part of a research project at the Technical University of Munich in which seven PhD students developed software that aimed for a level five functionality in an race car [25,26]. This race car (Devbot 2.0, see Figure 1) was developed and operated by the company Roborace [8].



(**a**) Devbot 2.0 (**b**) Devbot with sensors setup

Figure 1. Devbot 2.0 (**a**) for the Roborace Season Alpha 2019 with sensors setup (**b**) [8].

The Devbot 2.0 was based on an LMP 3 carbon fibre chassis and was a battery electric vehicle with a lithium polymer battery with a capacity of 40 kWh. With a weight of 1160 kg the car was powered by two electric motors that provided a combined power of 270 kW. It could reach a velocity of 220 km/h and a maximal lateral acceleration of 1.5 g. Besides the vehicle control systems, the car was equipped with a variety of sensors for localization and perception of the environment. It had two front and four surround cameras, five LIDAR systems (four around the front wheels and one in the back of the car), two radars (one in the front and one in the back), as well as 17 ultrasonic sensors all around the vehicle (Figure 1). The Devbot 2.0 had two main ECUs that were responsible for operating the vehicle. First of all a Nvidia Drive PX2 computing platform [27] was used to process the sensor data and run the perception and motion planning software parts. The second ECU was the Speedgoat Mobile real-time target machine [28] that was used as a main low-level controller.

4.2. The Autonomous Software Stack from TUM

For this autonomous racecar our team from TUM developed a holistic autonomous driving software. A thorough description is given in the papers [9,10]. An overview of the software architecture with its seven sub-modules is displayed in Figure 2 in sufficient detail.

Coming from sub-module one a map of the racetrack was created. This could be done either online/offline by using simultaneous localization and mapping (SLAM) algorithms. As an input data a LIDAR point cloud generated from the LIDAR sensors on the vehicle was used. The point cloud was processed afterwards by using either the GMAPPING or Google Cartographer algorithm. We adapted the parameters of both algorithms regarding the usage in large scales environments for creating a 2D occupancy grid of the track [29]. In addition we had the possibility to create the track map by measuring the boundaries of the track manually with high-precision GPS. Another alternative for creating the track map was by using a deep learning semantic segmentation algorithm that was fused with a monocular depth estimation and was presented in [30]. Because the online performance of the automatic map creation was currently not reliable enough the manual map creation method by measuring the GPS was used.

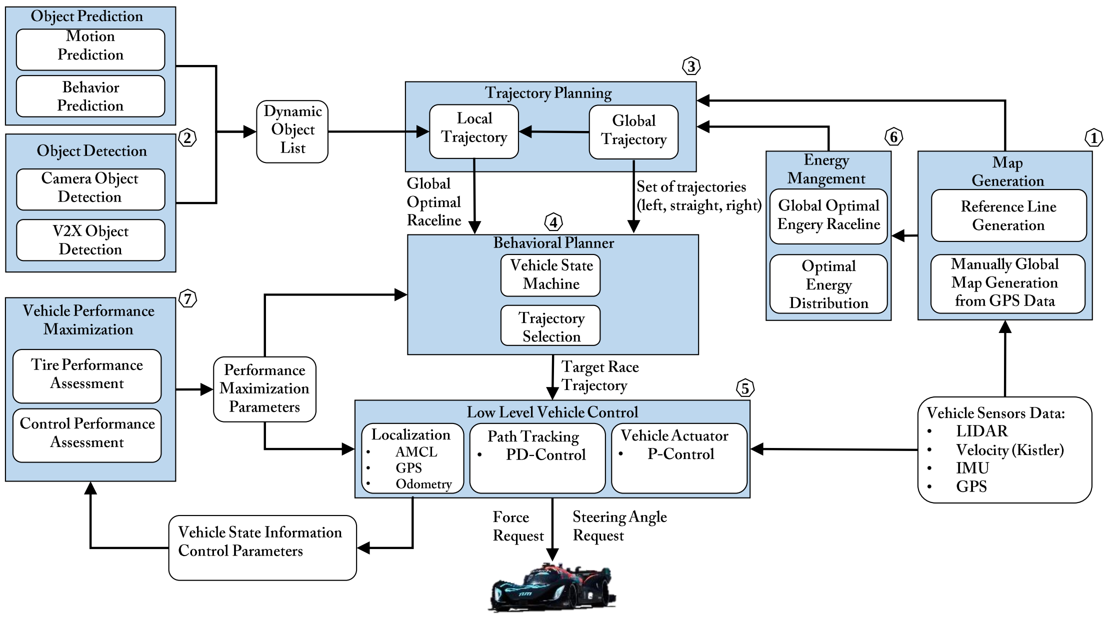

Object Prediction
Motion Prediction
Behavior Prediction
Object Detection
Camera Object Detection
V2X Object Detection
Dynamic Object List
Trajectory Planning
Local Trajectory
Global Trajectory
Global Optimal Raceline
Set of trajectories (left, straight, right)
Energy Mangement
Global Optimal Engery Raceline
Optimal Energy Distribution
Map Generation
Reference Line Generation
Manually Global Map Generation from GPS Data
Behavioral Planner
Vehicle State Machine
Trajectory Selection
Vehicle Performance Maximization
Tire Performance Assessment
Control Performance Assessment
Performance Maximization Parameters
Target Race Trajectory
Low Level Vehicle Control
Localization AMCL GPS Odometry
Path Tracking PD-Control
Vehicle Actuator P-Control
Vehicle Sensors Data: LIDAR Velocity (Kistler) IMU GPS
Vehicle State Information Control Parameters
Force Request
Steering Angle Request

Figure 2. The Technical University of Munich (TUM) team software architecture for the Roborace devbot.

With this map in submodule three a globally time-optimal trajectory around the track based on a minimum curvature trajectory planning presented in [31] was used. The trajectory contained the path on the track and the calculated velocity profile. When driving autonomously, the local trajectory planner used the global raceline as a reference for the fastest solution. However, if there were obstacles on the track or if the car wanted to perform an overtaking maneuver, it dynamically generated several local paths on the basis of pre-sampled splines [32]. The required information about objects on the track were gathered by a dynamic object list in submodule two. To complete the planning process, velocity profiles were calculated (see pseudocode in [31]) for each possible path based on a so called gg-diagram (Figure 3).

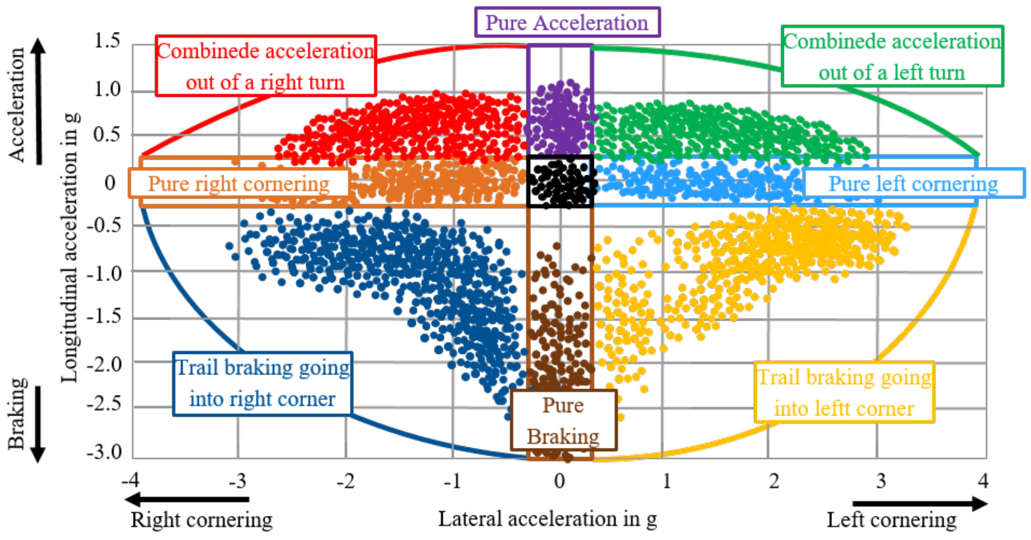


Figure 3. The gg-diagram explained in detail [33].

In the software, the diagram contained $a_{x,\max}$, the maximum lateral (pure cornering right and left), and $a_{y,\max}$, the maximal longitudinal (pure acceleration and braking) acceleration the car could handle. Combined acceleration cases were treated according to Equation (1), which defines the overall gg-diagram. The shape of the diagram could be changed by using the parameter shape coefficient b. We could change between a trapezoid ($b = 1$) and a circle ($b = 2$). All these parameters had to be defined according to the track surface and layout as well as the current weather conditions (dry, rainy, etc.). This definition and adjustment was done manually by the software programmers.

$$\left(\frac{|a_x|}{a_{x,\max}}\right)^b + \left(\frac{|a_y|}{a_{y,\max}}\right)^b \leq 1 \tag{1}$$

A set of driveable local trajectories is sent to the behavior planner in sub-module 4. It selects one trajectory based on several criteria such as safety and time loss. The final trajectory is then forwarded to the vehicle controller. The overall structure of the controller is displayed in Figure 4.

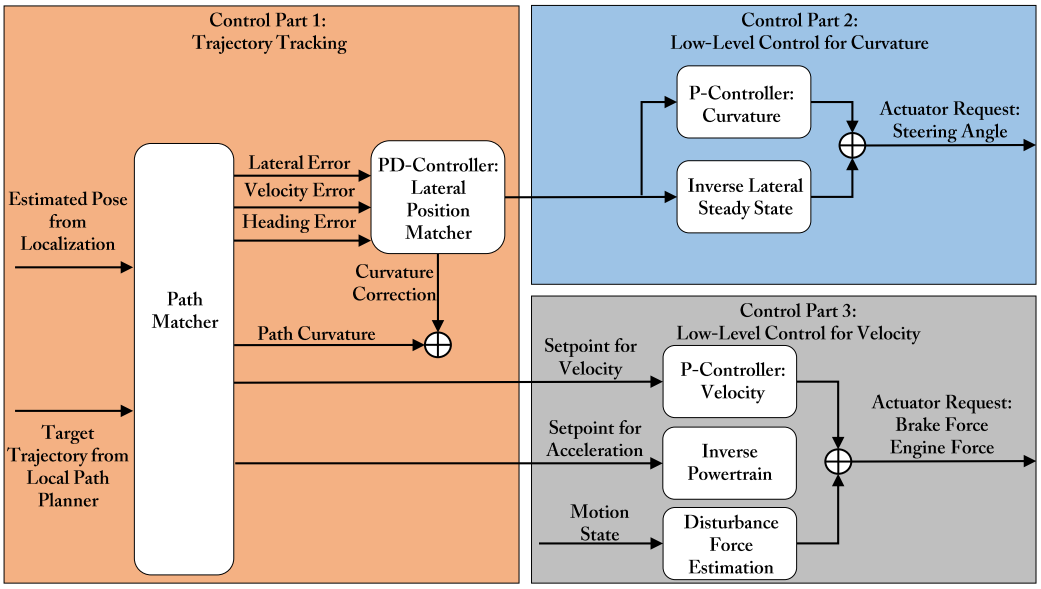

Control Part 1:
Trajectory Tracking
Estimated Pose from Localization
Target Trajectory from Local Path Planner
Path Matcher
Lateral Error
Velocity Error
Heading Error
PD-Controller: Lateral Position Matcher
Curvature Correction
Path Curvature
Control Part 2:
Low-Level Control for Curvature
P-Controller: Curvature
Inverse Lateral Steady State
Actuator Request: Steering Angle
Control Part 3:
Low-Level Control for Velocity
Setpoint for Velocity
P-Controller: Velocity
Setpoint for Acceleration
Inverse Powertrain
Motion State
Disturbance Force Estimation
Actuator Request: Brake Force Engine Force

Figure 4. Overall structure of the vehicle controller concept.

The controller architecture was split into a high-level trajectory tracking controller and a low-level vehicle controller. We separated these two controllers to increase the systems robustness [31]. Because we were driving with a racecar we had high non-linearities of the dynamics and a complex control allocation (split between feed forward and feedback parts and lateral and longitudinal couplings) at the limits of handling [31]. Therefore a high-level PD-controller was used for lateral position control that got information from the velocity heading error and the lateral deviation. The low-level controller was split into a curvature controller and velocity controller and for each of them a P-controller was used. The curvature could not be measured directly and the side slip angle derivatives were approximated by using a steady-state assumption [31]. The curvature controller was a proportional feedback controller with a fixed gain to add counter-steering in case of instabilities [31]. We used the same structure for the velocity controller. By using a proportional feedback we achieved stability and tracking of both velocity and acceleration setpoints [31]. This was combined with an inverse power-train model for estimating the driving resistances. The level of uncertainties were solved by adding a disturbance observer based on a steady-state Kalman filter. With the assumption that we had a constant disturbance upon our autonomous system we could calculate the necessary force request for the electrical engines and brakes.

Both controllers used the information for an estimated pose which was generated in the localization part in sub-module five. To make this localization robust and reliable the localization was split in two individual pipelines. First of all a LIDAR localization was used based upon the adpative Monte Carlo localization (AMCL) algorithm, a particle-filter-based localization approach, to perform very accurate lateral state estimations. We adapted and modified the basic AMCL regarding the high speeds of the racecar to achieve a reliable overall position estimation and presented this in [34]. In the second localization pipeline we used the high-precision GPS information and odometry information based on the velocity and IMU sensor. The data generated by the two pipelines was then fused in a Kalman filter based on a purely kinematic model. With this approach it was possible to provide a redundancy if one sensor failed. In addition we proved that the purely kinematic model outperformed a high-fidelity model in the high-speed scenario on the racetrack. Further explanations and results regarding the vehicle dynamics state estimation and localization of the TUM software can be found in [35].

Sub-module six included an energy management system, where we could adapt the electrical energy from the batteries and include thermal behavior of the battery and electrical engine. By integrating and optimization algorithm we could plan time-and-energy-optimal vehicle behavior. A first approach for this optimization is presented in the paper from Hermann et al. [36]. Sub-module seven included algorithms for assessing the tyre and a control performance of the vehicle. We will go more in depth in this specific submodule in the outlook of this paper.

5. A Step by Step Display of the Crash Chronology

5.1. The Crash in Modena—An Overview

As one of the teams participating in Roborace Season Alpha we had multiple test days before the season as well as additional practice sessions at the first Roborace Season Alpha event. Videos about the events can be found in the supplementary material. Until May 2019, the software stack has been tested on the real Devbot 2.0 vehicle for about 20 days and was used in one Season Alpha event. The second Roborace Season Alpha event was done on the Modena racetrack in Italy and had the classic Formula 1 race event format. First of all we had two days of testing with the vehicle on the racetrack. After that, a qualifing session was done to dermine the grind positions of the vehicle. At the end of this event a race (10 laps) between the autonomous vehicle was planned. The Modena racetrack from above is displayed in Figure 5.

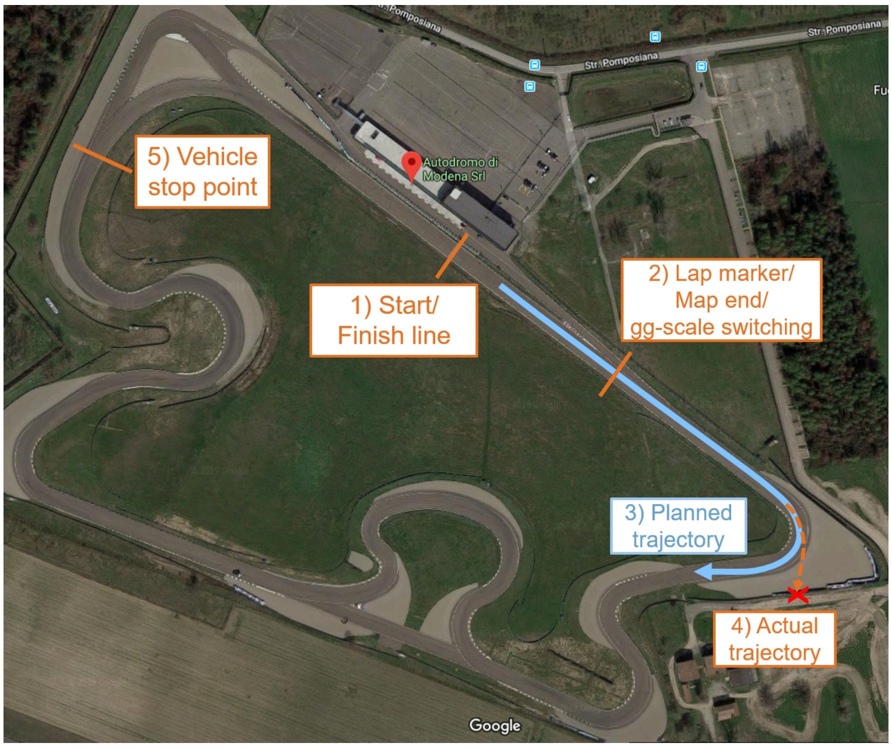


Figure 5. Modena racetrack with some important way-points.

During the practice session on the day before the race, the software parameters were adjusted manually to the track layout and track conditions as described in Section 4.1. This included functional tests as well as tuning of the gg-diagram. A setup of $a_x = a_y = 13.5\,\mathrm{m\,s^{-2}}$ had been determined as the maximum vehicle potential on this track during several 100 km/h runs. In addition, high velocity runs (about 200 km/h) were conducted with slightly lower limits. Furthermore, we compared the gg-data of the autonomous software with a real driver—the real driver achieved much higher longitudinal and lateral accelerations. Our conclusion was that our parametrization led to a vehicle behavior that drove the vehicle within its vehicle dynamic limits.

The event planned for this day was an autonomous race between two Devbot 2.0 cars. The grid position was determined by the time set on a flying lap with a single vehicle. Each car was granted a ten minute slot. During this session, the software lost control over the car after passing the start finish line on the last timed lap and crashed into the tire barriers near the first corner with a remaining speed of about 60 km/h. The crash resulted in a huge mechanical damage of the vehicle and destroyed nearly the complete left side of the Devbot.

5.2. The Crash in Modena—Qualifying Parameters

From previous tests, it was known that four laps could be completed within the given time-slot. During the run, a scale factor allowed adjustments of the gg-diagram in a lap-wise manner. The effect of scaling is visualized in Figure 6.

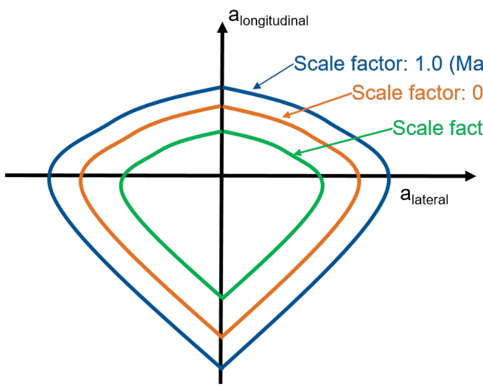


Figure 6. Scaling of the gg-diagram in the TUM software.

There were several reasons for the application of such a scaling factor. Firstly, a constant increase of the factor gave confidence about the current stability of the vehicle. Crucial vehicle signals (accelerations, heading, steering angle, lateral control error, etc.) could be observed live by the race engineer and the autonomous run could be aborted if necessary. Secondly, it allowed us to introduce a warm-up lap for heating up the tires and brakes. This was important for the first one or two laps because the gg-diagram was obviously tuned to fit warm tires. Thirdly, the scaling factor allowed other components to influence the trajectory planner as displayed in Figure 2. The vehicle performance maximization module (submodule 7) aimed at exploiting the vehicle capabilities and improving the control performance. Since these algorithms were still under development they were not applied during the real race and therefore the tuning of the scaling factor wass performed manually. For the Modena event the four laps were planned as follows:

- Lap one: Heating up the brakes and tires—gg-scale-factor = 0.8.
- Lap two: Increase speed and check vehicle stability—gg-scale-factor = 0.9.
- Lap three: Timed flying lap—gg-scale-factor = 0.925.
- Lap four: Cool down lap for brakes and tires—gg-scale-factor = 0.8.

5.3. The Crash in Modena—Step by Step Chronology of the Crash

The events that lead to the crash in a chronological order:

1. The car started the autonomous run with a scale factor of 0.8 on the start finish line of the track. No safety driver was inside the car. The track was dry with a surface temperature of 36 degrees Celsius. Tires were in good condition and had only completed 31 km. The live data stream displayed minor under- and over-steering in a few corners, which was not critical because of the cold tires. Lap one was completed successfully.
2. The car started lap two with a new gg-profile scaled to factor 0.9. On the start-finish-straight the car reached a maximum velocity of 195 km/h before starting to brake into corner one. Lap two was completed successfully.
3. The car started lap three with a new gg-profile scaled to factor 0.925. This scale factor was used for the first time this day which is why we left a gap to the maximum of the vehicle handling limits. On the start-finish-straight the vehicle reached a maximum velocity of 198 km/h before starting to brake into corner one. Lap three was completed successfully.
4. The car started the last lap with a new gg-profile scaled to factor 0.8 for the cool down of the vehicle. By entering corner one, the vehicle understeered. Around the apex of corner one, it went off the intended trajectory into the gravel without any request for slowing down (Figure 7). Soon after that, it hit a barrier in corner one.
5. The car ground to a halt because of major mechanical damages. The high-voltage (HV) light was green which indicated that the car was safe from an HV perspective. No one was injured and the car was recovered completely after the crash.

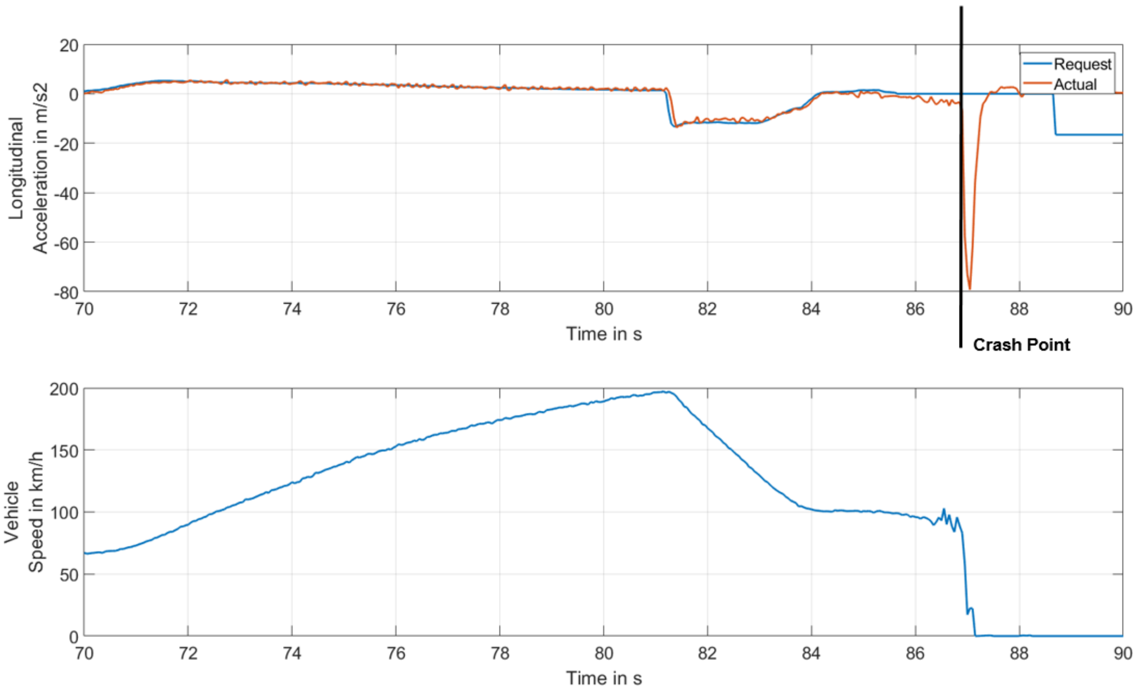


Figure 7. Crash explanation from data.

6. Software Explanation Failure in Detail

The accident was caused by a software failure which had not appeared in extensive simulation or during testing before, as it was very specific to the chosen software parameters and track layout. The live measurement data was working until the car was shutdown and the logs on the car could be rescued. This allowed us to analyze the crash in detail afterwards and have a extensive look at the data. With this data insight it was possible to exclude simple errors e.g., sensor was broken, wrong signaling, runtime error etc. The software failure could be devided in the following issues:

6.1. Velocity Planner Failure

The velocity profile generation algorithm turned out to be the root cause for the accident. Here is what happened:

1. After lap three (timed lap), the scale factor was switched from 0.925 to 0.8 (cool down). This switching happened after passing the last point of the official map created in sub-module one because this was assumed to be the start/finish line of the track. However, for Modena the final point of the map was at the end of the pitlane (map end in Figure 5) and therefore roughly 170 m after the real start/finish line which was the intended scale factor switching point. The car always stopped on a specific vehicle stop point before the pit entry and not after the start/finish line which is why we did not discover this important fact.
2. From this moment on, the 0.8-scaled "smaller" gg-diagram was used for the calculation of the velocity profile in the trajectory planner. Similar to the lap before, the software now requested a negative longitudinal deceleration (Figure 7).
3. As soon as the car started to turn into the corner the tires must not only transmit longitudinal but also lateral forces. This means that according to Equation (1) the possible deceleration must be reduced more and more while following the given path. Finally, no more deceleration is applied as soon as the pure lateral acceleration of the gg-diagram is fully used. Coming from a vehicle-dynamics perspective this behavior is reasonable. However, it strongly depends on the gg-diagram. In the current case there was remaining tire potential in reality which was not used because the software thought that it had already exploited all of it. The software stopped asking for further deceleration when the vehicle reached a speed of around 100 km/h, while in the previous lap it had slowed down to 65 km/h (Figure 7).
4. Usually it's not a problem if the car drives into the corner slightly too fast because the gg-diagram and/or the scaling factor leaves a small safety margin to the real vehicle handling limits. However, in this case the car was so fast that the real handling limits were exceeded. As soon as the tires could not transmit the required lateral force the vehicle under-steered off the intended trajectory. This led to a path into the gravel without any more request for slowing down.
5. In the analysis we found out that the scaling factor switching point was only a few meters too late to be able to brake down enough in front of corner one in combination with a slightly higher maximum velocity which was reached in lap three due to the scaling factor of 0.925.

The root cause of the problem could be traced down to decreasing the scale factor from 0.925 to 0.8 for the last lap. While this was done before on other tracks and in simulation, the specific track layout and the proximity of the switching point to the first corner made a difference in the behavior of the algorithm. In that situation we wanted to keep the tires in the best possible condition for the later race. However, we implemented some fall-back levels into our driving stack in the vehicle controller and trajectory planner interface display in Figure 8. The next section shows why they also failed.

6.2. Safety System Failure

The most important part of the safety system was placed in the vehicle controller [31] in submodule five of the TUM software. It is described in more detail in Figure 8. The control of the vehicle was split into path-matching and low-level control parts. The path was tracked using a PD-like controller with a velocity-based gain scheduling. The low-level curvature controller was a P-controller with a feed-forward term and outputted a steering angle request. The low-level-velocity controller consisted of a P-controller, a disturbance estimator, and a feed-forward term. It outputted an overall traction force which was eventually converted to torque requests for the electrical motors of the car.

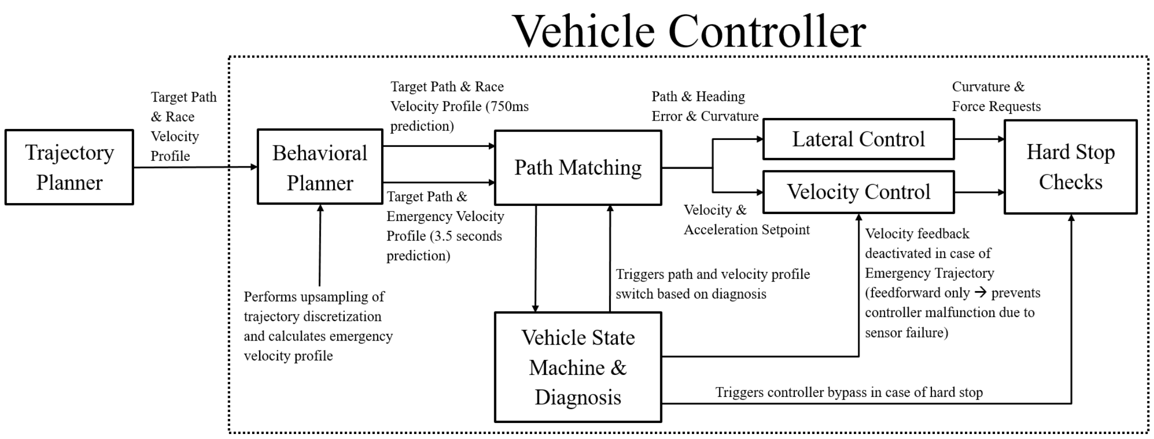

Vehicle Controller
Trajectory Planner
Target Path & Race Velocity Profile
Behavioral Planner
Target Path & Race Velocity Profile (750ms prediction)
Target Path & Emergency Velocity Profile (3.5 seconds prediction)
Performs upsampling of trajectory discretization and calculates emergency velocity profile
Path Matching
Path & Heading Error & Curvature
Lateral Control
Velocity Control
Velocity & Acceleration Setpoint
Curvature & Force Requests
Hard Stop Checks
Vehicle State Machine & Diagnosis
Triggers path and velocity profile switch based on diagnosis
Velocity feedback deactivated in case of Emergency Trajectory (feedforward only → prevents controller malfunction due to sensor failure)
Triggers controller bypass in case of hard stop

Figure 8. Low-level control architecture.

In addition to the race trajectory, an emergency trajectory was provided by the behavior planner in every time-step. It had an identical path as the race trajectory but a much longer foresight horizon and a modified velocity profile. The latter decelerated the car as fast as possible to a standstill. The main intention was to cover network and planning software failures. However, the mechanism failed due to two reasons:

1. The emergency gg-diagram was also scaled with the scaling factor.
2. The controller switched too late to the emergency trajectory.

The scaling of the emergency gg-diagram was originally not implemented when we first programmed this software part. However, during the previous race event in Monteblanco, Spain we suffered a problem when driving shortly behind another car in the so called "follow mode". Hereby, the emergency gg-diagram was used to calculate a velocity profile which stops closely behind the position where the car in front would stop if it fully braked. This was used later in the software to determine the currently required minimum distance to the car in front. Put simply, the velocity profile ran into infeasibility when the software shortly switched from the normal part of the follow mode velocity profile (which was scaled by the scaling factor) into the emergency part of the follow mode velocity profile (which was originally unscaled) and back again into the scaled part while approaching a corner. The problem was similar to the problem in Modena but in a completely different environment. As a quick fix we decided to also scale the emergency gg-diagram when we were in Monteblanco. However, we did not track the issue and therefore introduced one of the problems into the software.

However, this would not have made a huge difference because of the second problem: The vehicle controller [31] switched too late from race to emergency trajectory. This switching was done based on the following safety checks:

- Behavior planner communication timeout $\geq$ 300 ms.
- Non-critical sensor fusion failure (fusing IMU, wheel-speed and velocity sensor information).
- Vehicle sliding based on a chassis side slip threshold.
- Maximum lateral control error of 3 m for longer than 300 ms.

In addition, the software performed a "hard stop" (i.e., it applies full brake pressure) if one of the following conditions was fulfilled:

- Communication loss to the low-level ECU.
- Driving on the emergency trajectory without coming to a standstill for more than 5 s.

The controller switched to emergency as soon as the car reached a lateral control error of 3 m which happened right before the apex of corner one before leaving the track into the gravel (Figure 9).

At this point is was already too late to brake the car down to a standstill even with a 1.0 scaling factor. Neither the race trajectory nor the emergency trajectory applied any further deceleration (because the algorithm assumed that the tires' friction capacity was already exploited). Therefore, the velocity controller did not see a reason to slow down the car (Figure 9).

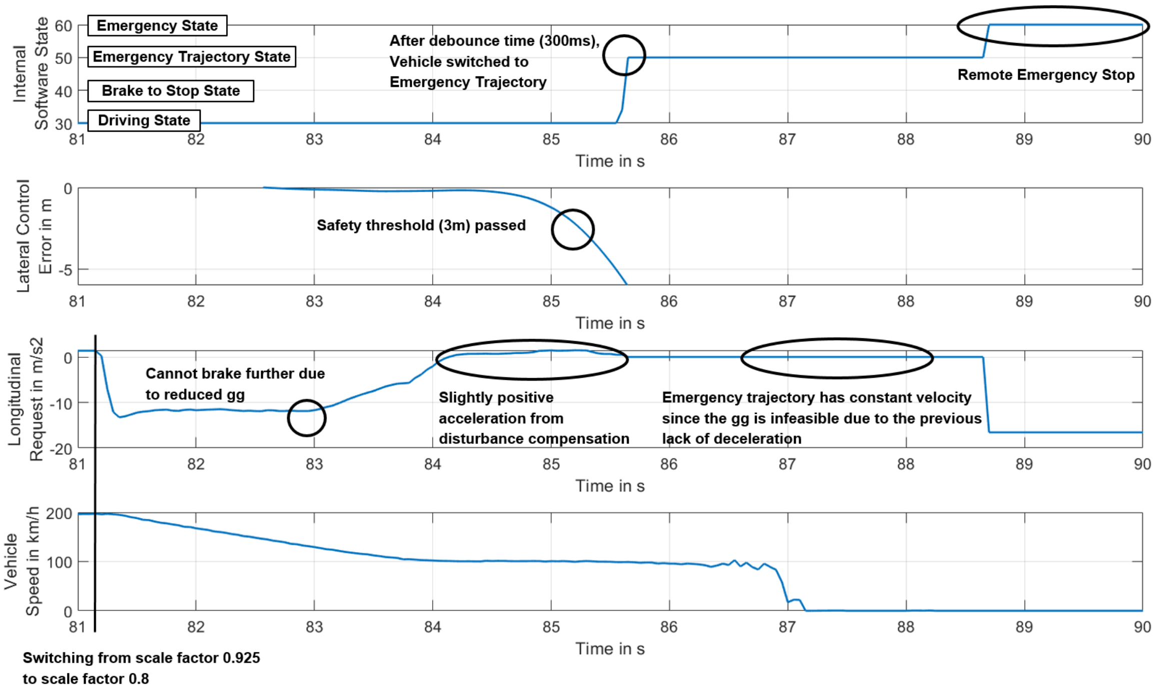


Figure 9. Crash explanation from data: safety system failure.

To sum it up, the Modena crash was an accumulation of coincidences together with some design weaknesses of the software. The crash did not happen because the vehicle or the engineers overestimated the physical limit. However, the presence of several hard constraints in conjunction with operating in the area of the physical limits made such incidents much more likely.

7. Discussion

In this paper we presented a research design for addressing and explaining autonomous disengagements, failures and crashes. We presented a general method that can be used by everyone to display their autonomous crash in detail on a high level (scenario and step by step explanation) as well as on a low level (software failure explanation in detail). In the previous chapters, we used this method to explain the crash of an autonomous car in detail. This crash occurred on a racetrack with a high-performance race car and provides therefore a special environment for autonomous driving. The discussion is focusing therefore on our specific use case and software stack. Nevertheless, the analysis of the crash leads to more general conclusions for autonomous driving software development—especially when it comes to road or highway scenarios with other traffic participants.

7.1. *Evaluation of the New Presented Method for Autonomous Disengagements*

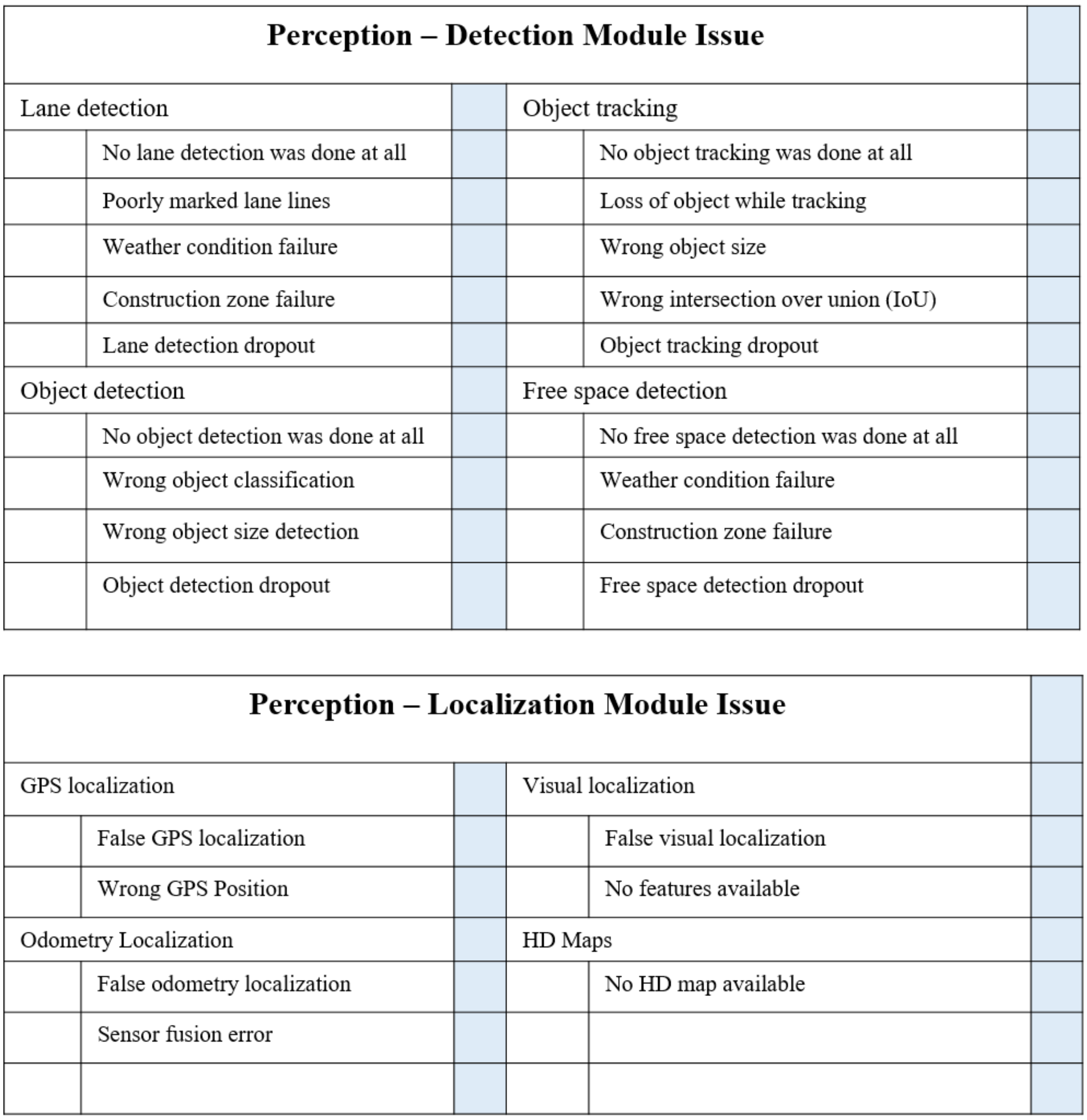
With state-of-the-art knowledge we presented in Section 3 our research design for addressing all upcoming research in the field of autonomous disengagements, malfunctions, and crashes. By presenting a general method that is used to go through the crash in detail, a comparison with other autonomous driving crashes can be derived more easily. In our method we use six different steps to provide the necessary background knowledge about the vehicle and software, and then describing the crash itself and the software failure in detail. After presenting this approach we used this method to describe an autonomous crash that happened with an autonomous racecar.

The method itself provides a clear and straight forward structure. It is clear what needs to be done in each step. In addition the method can be seen as holistic: each important part for understanding the crash in detail is addressed. On the one hand this costs time and effort; on the other hand this is necessary to derive an overall explanation of the crash. In our specific case of the Modena crash we figured out that a first explanation of the scenario (here, the racetrack) was important to figure out special occasions. Without this we would have been focusing more on the software itself but would not have realized all interrelations. With the background knowledge about the software architecture and the autonomous vehicle it was easy to display the software failure step by step. The links between what happens in the complex software architecture and what leads to a specific vehicle behavior can be understood better. Beside this the detailed insights in real vehicle data gives a better understanding of vehicle dynamics and vehicle states.

To sum it up, the presented method in Section 3 was used to explain an autonomous vehicle crash in detail. The methods works well for displaying critical autonomous vehicle behavior and gives a detailed overview of the software. In addition the clear and straight forward structure gives other researchers the possibility to easily display their autonomous crash or disengagement, too. Therefore a companion of crashes from different vehicles can be done more easily.

7.2. Software Disengagement Reports

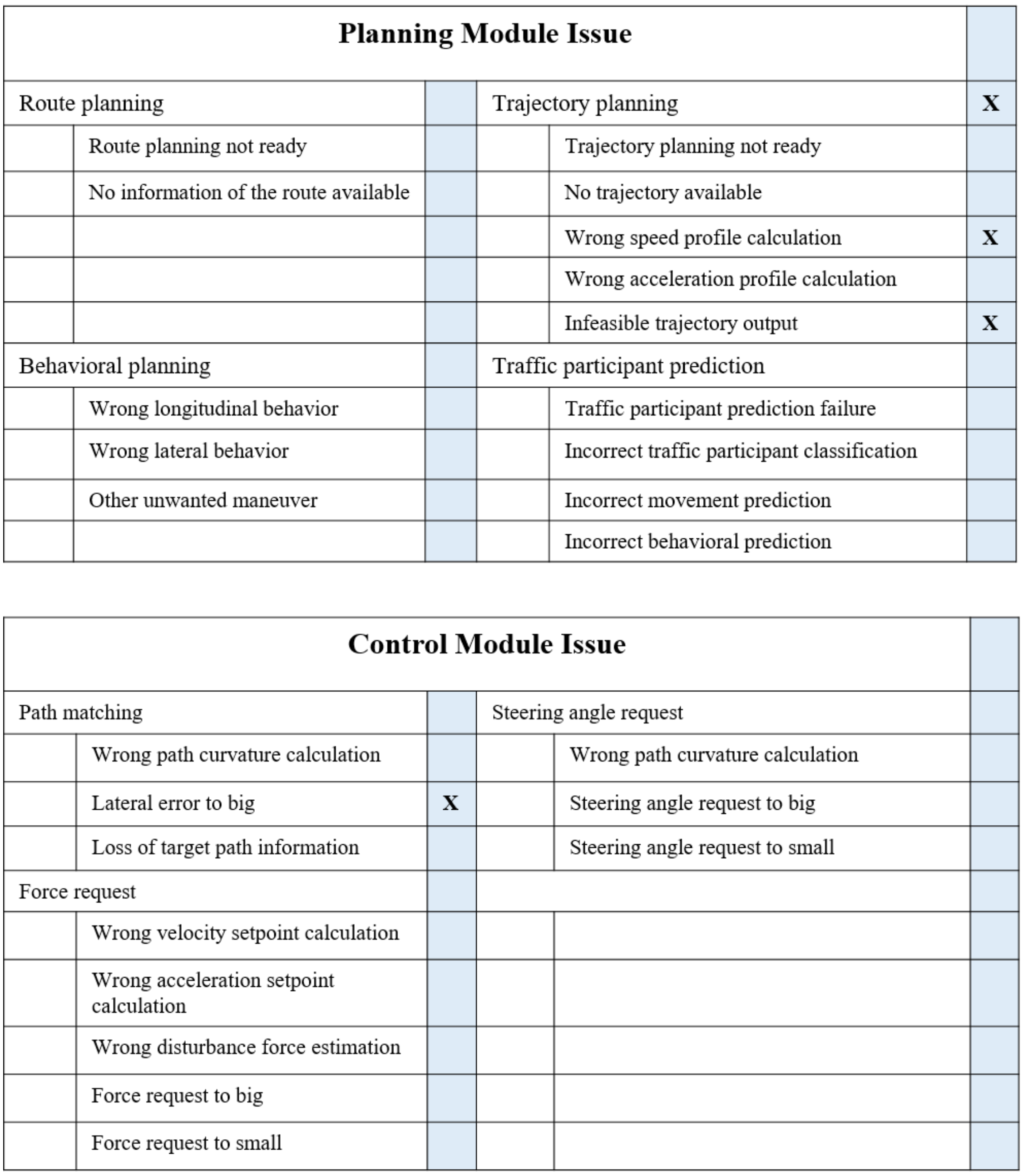
In the related work section, we presented an overview of the current publications and reports that address disengagements and collisions with autonomous vehicles. We stated that in these reports the disengagement itself was displayed and that some additional information like weather, lighting, or roadway surface condition [15] was integrated, but detailed information about the software issues were not available. This is mainly because the companies do not want to address their software problems publicly. Coming from the critics in the state of the art, the current disengagement reports should be extended by detailed information about the software error. Therefore, we propose new categories in which the software problem that leads to the crash or disengagement can be classified. In addition, we propose that is necessary that more reports and analytics covering autonomous driving crashes have to be published. A proposal for additional categories is shown in Figures 10 and 11.



Perception – Detection Module Issue					
Lane detection			Object tracking		
	No lane detection was done at all			No object tracking was done at all	
	Poorly marked lane lines			Loss of object while tracking	
	Weather condition failure			Wrong object size	
	Construction zone failure			Wrong intersection over union (IoU)	
	Lane detection dropout			Object tracking dropout	
Object detection			Free space detection		
	No object detection was done at all			No free space detection was done at all	
	Wrong object classification			Weather condition failure	
	Wrong object size detection			Construction zone failure	
	Object detection dropout			Free space detection dropout	

Perception – Localization Module Issue					
GPS localization			Visual localization		
	False GPS localization			False visual localization	
	Wrong GPS Position			No features available	
Odometry Localization			HD Maps		
	False odometry localization			No HD map available	
	Sensor fusion error				

Figure 10. New proposal for disengagement report categories regarding software issues with the Modena crash as an example—perception modules.



Planning Module Issue			
Route planning		Trajectory planning	X
Route planning not ready		Trajectory planning not ready	
No information of the route available		No trajectory available	
		Wrong speed profile calculation	X
		Wrong acceleration profile calculation	
		Infeasible trajectory output	X
Behavioral planning		Traffic participant prediction	
Wrong longitudinal behavior		Traffic participant prediction failure	
Wrong lateral behavior		Incorrect traffic participant classification	
Other unwanted maneuver		Incorrect movement prediction	
		Incorrect behavioral prediction	

Control Module Issue			
Path matching		Steering angle request	
Wrong path curvature calculation		Wrong path curvature calculation	
Lateral error to big	X	Steering angle request to big	
Loss of target path information		Steering angle request to small	
Force request			
Wrong velocity setpoint calculation			
Wrong acceleration setpoint calculation			
Wrong disturbance force estimation			
Force request to big			
Force request to small			

Figure 11. New proposal for disengagement report categories regarding software issues with the Modena crash as an example—Planning and control modules.

7.3. Precautions to Prevent this Specific Problem

Before discussing improvements for the development process of autonomous driving software, we want to point out some precautions which we took to prevent this specific problem to rise again:

1. Prevent down scaling of the gg-diagram if this leads to an infeasible velocity profile.
2. Use the unscaled gg-diagram for the emergency trajectory to be able to fully utilize the vehicle handling limits.
3. Tighten the condition for switching to the emergency trajectory to 1.5 m lateral error
4. Use the real start finish line instead of the end of the reference line as switching point for the scaling factor.
5. Evaluate different velocity planner concepts that are allow the integration of boundary conditions and implement them (future work).

6. The controller checks both received trajectories if they are feasible. This is done by calculating the total acceleration of the car on the basis of the requested longitudinal acceleration, an unscaled gg-diagram and the curvature profile. Furthermore, the emergency trajectory must always end at standstill. If one or both conditions is not met, it immediately switches to the last valid emergency trajectory. The main idea behind this step is, that it is easier to verify a result instead of producing a correct result and a redundancy for ensuring the constraints is introduced.

7.4. Enhanced Error Detection Mechanisms

Software has to meet a variety of requirements, which often works very well as long as there is no error. However, if errors occur in hardware or software, it quickly becomes dangerous for the passengers of the car. This is why it is important to make software safer and error-tolerant through structured procedures and certain methods. Even if this can never be achieved 100 percent, emergency strategies can be implemented that mitigate the consequences of an error. The ISO 26262 standard deals with the attainment of functional safety for road vehicles and therefore also includes measures and recommendations for the development of software functions [37]. The automotive safety integrity level (ASIL) classifies hazardous situations caused by system or component failures in four groups. To certify a system for use in a high ASIL situation, it is necessary to integrate error detection mechanisms at software level. In our software we will use the following:

- Interval test of input and output data
- Plausibility checks through models or comparison of different data sources
- Detection of data errors by error detection algorithms and multiple data storage
- Selected deactivation of faulty functions
- External monitoring devices, e.g., watchdog
- Independent redundancies with different software structures

However, with growing complexity of the software deployed on autonomous vehicles (especially when compared to nowadays advanced driver assistance systems (ADAS)), the current standards for functional safety reach their limits. Besides the existing methods in the software development of autonomous driving functions, further approaches tackling the arising challenges have to be established. Coming from the perspective that this crash happened because of an improper velocity profile, we propose the integration of two new mechanisms enhancing safety:

1. Safety layer for the trajectory planner: We propose the integration of a safety layer for the trajectory planner. The idea is to have a second planner that runs as a redundancy for the actual trajectory planner. To get reliable results this second planner should be developed independently and on another basis than the main planner. This safety layer can then be used for the inspection of the feasibility of the generated velocity profiles. In addition, it should secure that the emergency trajectory always ends at 0 km/h. Additional conditions can be integrated. If any of the conditions is not matched, the safety layer switches immediately to the last valid emergency trajectory. To make the approach even more robust the safety layer and the main planner could run on different hardware. The Devbot has a Nvidia Drive PX2 as main ECU. This ECU offers two seperate GPU/CPU combinations that can run software individually [27]. In addition, an AURIX microcontroller is integrated as decisive safety compute element.
2. *Supvervisor submodule*: We propose the development of an additional runtime verification module, called "supervisor". The scope of this sub-module is to safeguard autonomous driving functions. Thereby, the goal is to identify and monitor an exhaustive list of features contributing to a safe operation, i.e., trajectory. Each of these features is then tracked by an evaluation metric. Based on the full list of metrics, a classification function decides on the binary safety state of a corresponding trajectory ("safe" or "unsafe"). Since an extensive list of features is surveyed this way, the supervisor module ensures the trajectory planner to stay within safe bounds. As a side

product, approval of a complex trajectory planner can be eased up. The development of such a supervisor is part of future research in our team.

7.5. Enhanced Simulation

Winner et al. [5] stated that real driving is currently the most important method for approval of the vehicle and its autonomous functions. The problem is, that the number of real testing kilometres depends on how many kilometres there is a fatal accident after: "for the motorway pilot, the current figures from the Federal Statistical Office show a comparative value of 662 million kilometres between two accidents with fatalities. Accordingly, 6.62 billion test kilometres would have to be completed on the motorway in order to meet the conditions described." [38]. Because driving this enormous distance is not applicable and realistic, we develop and test in simulations. For the software stack development we set up a hardware in the loop (HiL) simulator that consists of the vehicle ECUs and a two-track vehicle dynamics model. This simulation gives us the possibility to make enhanced tests with our software stack in different test scenarios. In addition to the software development we enhance the well-know safeguarding concepts towards autonomous driving:

- In the future we will focus more on Software in the Loop (SiL) development than on model in the loop (MIL) development. This is mainly because autonomous driving functions are connected to different subfunctions and have influences on other functions like described in this paper and displayed in our software architecture.
- Reuse of tests in all development phases: We need the same tests and same scenarios for all development stages.
- Automated scenarios testing: The automated simulation of different traffic scenarios on a variety of road networks with different environmental conditions is the most important part in the future autonomous driving software development. Therefore, the intelligent selection of test scenarios like displayed in [39,40] is of decisive importance. A large number of tests to cover variations of scenarios and random tests must be developed. In addition, we need a consistent management of test parameters and result analysis.
- Faster development processes: we need to design a continuous integration (CI) system for the environment development and the build and integration processes used in the software development. This is necessary in order to build, test, and integrate software modules in large scale environments in a short time.
- Data analysis tools for algorithm analysis: as described in the paper, the necessary tools for evaluating software issues fast and detailed are not available yet. We need tools that can explain software issues with unambiguous metrics without surveying gigabyte of data first. In addition we need special tools for machine learning algorithm decisions (traceability)

8. Outlook

In this paper a method for explaining and describing autonomous disengagements, crashes, and malfunctions is presented. We use this method afterwards to describe and autonomous vehicle crash. We explained what caused the issue inside the software and gave different proposal for necessary developments. Future work will focus on the integration of enhanced error detection mechanisms. The proposed safety layer for the planner and the supervisor sub-module are currently under development and will be integrated in the software stack by the beginning of 2020. In addition, we set up a continuous integration process that automatically tests the entire software stack every night with selected test cases. In the future we will enhance this continuous integration regarding more test cases and faster testing. Last but not least we are currently developing the vehicle performance maximization module were we integrate the tire and control performance assessment. For real testing, we will enhance our software stack and test in on different vehicles (e.g., Roborace). In addition, we are currently setting up a road legal car which is capable to run the TUM Roborace software stack on normal streets. Having access to two vehicles we will make further evaluation about the capability of driving fast and secure.

Another important part of the future development will be an enhancement of the software sub-modules regarding the vehicle performance maximization. Coming from the described crash in Modena we figured out that besides the safety failures in our software we had a significant problem in planning the correct trajectory. Although the gg-scale factor gives us lots of flexibility it was also the critical factor for crashing the vehicle. As explained in Section 5, the gg-scale factor is similar to the friction coefficient μ which displays the maximum transferable force between tire and road surface. This means, as long as the car does not detect the really available current μ coefficient during autonomous driving it will never be able to plan a fully safe trajectory, especially when it comes to evasive maneuvers or critical situations the car needs to know the exact friction limits to drive the vehicle at the handling limits. To address this issue we propose the additional functions we displayed in sub-module seven in Figure 2:

1. Tire performance assessment: the main goal of the tire performance assessment is to obtain information about the friction potential of the vehicle's tire-road contact, to store these data in backend maps and to provide it where required [10]. With this knowledge, the dynamic-trajectory-planning algorithms can calculate an adjusted path and velocity profile taking the vehicle's local maximum lateral and longitudinal acceleration into account [10]. By using vehicle-internal state variables (accelerations, forces, and torques) and an extended Kalman filter for estimating (partially) unobserved state variables, the current tire-road friction potential can be estimated. Furthermore, the algorithm collects data over a certain period of time to calculate a mean value of the friction potential for fixed road segments. A detailed explanation of this proposed software can be found in [10,41].
2. Control performance assessment: in our current software architecture we plan under ideal and simplified conditions. Obviously, this leads to a mismatch between planning and realization of the planned trajectory. To mitigate this uncertainty, it is required to include safety margins in terms of maximum accelerations as well as safety distance to the track boundaries [10]. However, it is unclear a priori how to set them as they depend on the controller performance and the environment conditions [10]. To overcome this problem, a data-driven algorithm is proposed which scales the allowed maximum accelerations while at the same time monitoring several safety constraints [10]. As safety constraints we chose the peak lateral tracking error, the longitudinal wheelslip and the difference between front and rear sideslip angle [10]. By using a Gaussian process regression, we learn those constraints using real data and can improve the estimates while driving. A detailed explanation of this proposed software can be found in [42].

9. Materials

We make our autonomous software stack publicly available at: https://github.com/TUMFTM.

10. Conclusions

In this paper a method for explaining and describing autonomous disengagements, crashes, and malfunctions is presented. We use this method afterwards to describe and autonomous vehicle crash. By describing the background of the used autonomous vehicle and software stack that was used, we explained in detail what happened in the crash. Afterwards a detailed insight in the software issues was given. We figure out that the crash was caused by an issue in the path planning software. More or less the crash is owed to the circumstance that many unforeseen coincidences came together. The general method we display in this paper was evaluated as a very good format to address autonomous disengagements are crashes. By using a detailed insight in the crash chronology and in the software structure it is possible to figure out cause of the crash. Interrelations between software modules or specific scenario occurrences are integrated in this method as well as detailed vehicle data investigations.

Nevertheless, the detailed review of this crash leads to many important conclusions in the autonomous software developed. These conclusions can be used for further research and by autonomous driving software developers. With the presentation of the state of the art it became clear that the autonomous driving development community misses such a general method to display an autonomous disengagement. We hope that this report will be imitated by others to speed up the development and increase the safety of autonomous driving.

Supplementary Materials: Videos demonstrating the system on the racetrack can be found here: Roborace - Monteblanco Episode 01: https://www.youtube.com/watch?v=eE27DGge1u4; Roborace - Monteblanco Episode 02: https://www.youtube.com/watch?v=yJX-CP9tIoc; Roborace - Monteblanco Episode 03: https://www.youtube.com/watch?v=yoTQlaoVuHg. A video demonstrating the software development on the HiL Simulator can be found here: How To Simulate AI For Autonomous Racing - Roborace: https://www.youtube.com/watch?v=d1nLdsC2cK4&t.

Author Contributions: J.B. as the first author initiated the idea of this paper and contributed essentially to the processing of the crash in detail. A.H. (global trajectory planning, behavior planning), A.W. (vehicle control), and T.S. (local trajectory planning) contributed equally to different parts of the software architecture and of the processing of the crash in detail. M.L. made an essential contribution to the conception of the research project. He revised the paper critically for important intellectual content. He gave final approval of the version to be published and agrees to all aspects of the work. As a guarantor, he accepts the responsibility for the overall integrity of the paper. Conceptualization, J.B.; Data curation, A.W., A.H., T.S.; Formal analysis, J.B., A.W., A.H., T.S.; Funding acquisition, J.B., M.L.; Investigation, J.B., A.W., A.H., T.S.; Methodology, J.B., A.W., A.H., T.S.; Project administration, J.B., M.L.; Resources, J.B., M.L.; Software, J.B., A.W., A.H., T.S.; Supervision, J.B., M.L.; Validation, A.W., A.H., T.S.; Visualization, J.B., A.W., A.H., T.S.; Writing—original draft, J.B.; Writing—review and editing, J.B., A.W., A.H., T.S., M.L.

Funding: We would like to thank the Bayerische Forschungsstiftung (AZ-1310-17) for funding us in the research project rAIcing. We would like to thank Continental Engineering Services and the TÜV Süd for funding us with research projects. The work described in this paper was also supported by the basic research fund of the Chair of Automotive Technology from the Technical University of Munich. This work was supported by the German Research Foundation (DFG) and the Technical University of Munich (TUM) in the framework of the Open Access Publishing Program.

Acknowledgments: First, we want to thank the Roborace team for giving us the opportunity to work with them and using their Devbot vehicles for our research. In addition, we want to thank the company Speedgoat for supporting us with hardware for our hardware-in-the-loop simulator.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ABS	Anti-lock Braking System
ADAS	Advanced Driver Assistance System
AMCL	Adpative Monte Carlo Localization
ASIL	Automotive Safety Integrity Level
CI	Continuous Integration
CPU	Central Processing Unit
ECU	Electrical Control Unit
ESP	Electronic Stability Program
GPU	Graphical Processing Unit
HiL	Hardware in the Loop
HV	High Voltage
IMU	Inertial Measurement Unit
LMP	Le Mans Prototype
ROS	Robot Operating System
TUM	Technical University of Munich
WHO	World Health Organization

References

1. World Health Organization. *Road Traffic Injuries 2018*; World Health Organization: Geneva, Switzerland, 2019. Available online: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries (accessed on the 23 August 2019).
2. Statistisches Bundesamt. Verkehrsunfälle und Verunglückte im Zeitvergleich (ab 1950). 2019. Available online: https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Verkehrsunfaelle/Tabellen/liste-strassenverkehrsunfaelle.html (accessed on the 2 September 2019).
3. Johanning, V. *Car IT kompakt Das Auto der Zukunft—Vernetzt und Autonom Fahren*; Springer Vieweg: Wiesbaden, Germany, 2015.
4. Minx, E. *Autonomes Fahren : Wo Wir Heute Stehen und Was Noch Zu Tun Ist*; Axel Springer SE, Corporate Solutions: Berlin, Germany, 2015.
5. Winner, H. Introducing autonomous driving: An overview of safety challenges and market introduction strategies. *Automatisierungstechnik* **2018**, *66*, 100–106, doi:10.1515/auto-2017-0106.
6. SAE International. Auotmated Driving—Levels of Driving Automation. Available online: https://www.sae.org/misc/pdfs/automated_driving.pdf (accessed on 26 September 1988).
7. Wikipedia. Bundesstelle für Flugunfalluntersuchung. 2019. Available online: https://de.wikipedia.org/wiki/Bundesstelle_für_Flugunfalluntersuchung (accessed on 18 September 2019).
8. Roborace Ltd. Roborace. 2019. Available online: https://roborace.com/ (accessed on 23 August 2019).
9. Betz, J.; Wischnewski, A.; Heilmeier, A.; Nobis, F.; Stahl, T.; Hermansdorfer, L.; Lienkamp, M. A Software Architecture for an Autonomous Racecar. In Proceedings of the 2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring), Kuala Lumpur, Malaysia, 28 April–1 May 2019; doi:10.1109/vtcspring.2019.8746367.
10. Betz, J.; Wischnewski, A.; Heilmeier, A.; Nobis, F.; Stahl, T.; Hermansdorfer, L.; Lienkamp, M. A Software Architecture for the Dynamic Path Planning of an Autonomous Racecar at the Limits of Handling. In Proceedings of the 8th IEEE International Conference on Connected Vehicles and Expo (ICCVE 2019), Graz, Austria, 4–8 November 2019.
11. Pendleton, S.; Andersen, H.; Du, X.; Shen, X.; Meghjani, M.; Eng, Y.; Rus, D.; Ang, M. Perception, Planning, Control, and Coordination for Autonomous Vehicles. *Machines* **2017**, *5*, 6, doi:10.3390/machines5010006.
12. Kato, S.; Takeuchi, E.; Ishiguro, Y.; Ninomiya, Y.; Takeda, K.; Hamada, T. An Open Approach to Autonomous Vehicles. *IEEE Micro* **2015**, *35*, 60–68, doi:10.1109/mm.2015.133.
13. Kato, S.; Tokunaga, S.; Maruyama, Y.; Maeda, S.; Hirabayashi, M.; Kitsukawa, Y.; Monrroy, A.; Ando, T.; Fujii, Y.; Azumi, T. Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems. In Proceedings of the 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), Porto, Portugal, 11–13 April 2018; doi:10.1109/iccps.2018.00035.
14. Apollo. Apollo Open Platform. 2019. Available online: http://apollo.auto/ (accessed on the 23 August 2019).
15. State of California—Department of Motor Vehicles. Report of Traffic Collision Involving an Autonomous Vehicle (OL 316). 2019. Available online: https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/autonomousveh_ol316 (accessed on 23 August 2019).
16. Favarò, F.M.; Nader, N.; Eurich, S.O.; Tripp, M.; Varadaraju, N. Examining accident reports involving autonomous vehicles in California. *PLoS ONE* **2017**, *12*, e0184952, doi:10.1371/journal.pone.0184952.
17. Favarò, F.; Eurich, S.; Nader, N. Autonomous vehicles' disengagements: Trends, triggers, and regulatory limitations. *Accid. Anal. Prev.* **2018**, *110*, 136–148, doi:10.1016/j.aap.2017.11.001.
18. State of California - Department of Motor Vehicles. Autonomous Vehicle Disengagement Reports 2018. 2019. Available online: https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/disengagement_report_2018 (accessed on 23 August 2019).
19. Favaro, F.M.; Eurich, S.O.; Nader, N. Analysis of Disengagements in Autonomous Vehicle Technology. In Proceedings of the 2018 Annual Reliability and Maintainability Symposium (RAMS), Reno, NV, USA, 22–25 January 2018; doi:10.1109/ram.2018.8463123.
20. Tesla. Tesla Autopilot. 2019. Available online: https://www.tesla.com/de_DE/autopilot (accessed on 23 August 2019).

21. Electrek. Understanding the Fatal Tesla Accident on Autopilot and the NHTSA Probe. 2019. Available online: https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/ (accessed on 23 August 2019).
22. The economist. Why Uber's Self-Driving Car Killed a Pedestrian. 2019. Available online: https://www.economist.com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian (accessed on 23 August 2019).
23. Wikipedia. Death of Elaine Herzberg. 2019. Available online: https://en.wikipedia.org/wiki/Death_of_Elaine_Herzberg (accessed on 23 August 2019).
24. Fletcher, L.; Teller, S.; Olson, E.; Moore, D.; Kuwata, Y.; How, J.; Leonard, J.; Miller, I.; Campbell, M.; Huttenlocher, D.; et al. The MIT-Cornell collision and why it happened. *J. Field Robot.* **2008**, *25*, 775–807, doi:10.1002/rob.20266.
25. Technical University of Munich - Chair of Automotive Technology. Research Project Roborace: Autonomous Motorsport. 2018. Available online: https://www.ftm.mw.tum.de/en/main-research/vehicle-dynamics-and-control-systems/roborace-autonomous-motorsport/ (accessed on 25 May 2019).
26. Betz, J.; Wischnewski, A.; Heilmeier, A.; Nobis, F.; Stahl, T.; Hermansdorfer, L.; Lienkamp, M. What can we learn from autonomous level-5 motorsport? In *9th International Munich Chassis Symposium 2018*; Springer Fachmedien Wiesbaden: Wiesbaden, Germany, 2018; doi:10.1007/978-3-658-22050-1_12.
27. Nvidia. Nvidia Drive Platform. 2019. Available online: https://www.nvidia.com/en-us/self-driving-cars/drive-platform/ (accessed on 9 September 2019).
28. Speedgoat. Mobile Real-Time Target Machine. 2019. Available online: https://www.speedgoat.com/products-services/real-time-target-machines/mobile (accessed on 4 January 2019).
29. Nobis, F.; Betz, J.; Hermansdorfer, L.; Lienkamp, M. Autonomous Racing: A Comparison of SLAM Algorithms for Large Scale Outdoor Environments. In Proceedings of the 2019 3rd International Conference on Virtual and Augmented Reality Simulations (ICVARS 19), Perth, Australia, 23–25 February 2019; doi:10.1145/3332305.3332319.
30. Palafox, P.R.; Betz, J.; Nobis, F.; Riedl, K.; Lienkamp, M. SemanticDepth: Fusing Semantic Segmentation and Monocular Depth Estimation for Enabling Autonomous Driving in Roads without Lane Lines. *Sensors* **2019**, *19*, 3224, doi:10.3390/s19143224.
31. Heilmeier, A.; Wischnewski, A.; Hermansdorfer, L.; Betz, J.; Lienkamp, M.; Lohmann, B. Minimum curvature trajectory planning and control for an autonomous race car. *Veh. Syst. Dynam.* **2019**, 1–31, doi:10.1080/00423114.2019.1631455.
32. Stahl, T.; Alexander, W.; Betz, J.; Lienkamp, M. Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios. In Proceedings of the 2019 IEEE 22nd International Conference on Intelligent Transportation Systems (ITSC 2019), Auckland, New Zealand, 27–30 October 2019, in press.
33. Racecar Engineering. OptimumG GG-Diagram. 2019. Available online: https://www.racecar-engineering.com/latestissue/july-2018-issue/attachment/opitmumg-g-g-diagram/ (accessed on 23 August 2019).
34. Stahl, T.; Wischnewski, A.; Betz, J.; Lienkamp, M. ROS-based localization of a race vehicle at high-speed using LIDAR. *E3S Web Conf.* **2019**, *95*, 04002, doi:10.1051/e3sconf/20199504002.
35. Wischnewski, A.; Stahl, T.; Betz, J.; Lohmann, B. Vehicle Dynamics State Estimation and Localization for High Performance Race Cars. *IFAC-PapersOnLine* **2019**, *52*, 154–161, doi:10.1016/j.ifacol.2019.08.064.
36. Herrmann, T.; Christ, F.; Betz, J.; Lienkamp, M. Energy Management Strategy for an Autonomous Electric Racecar using Optimal Control. In Proceedings of the 2019 IEEE 22nd International Conference on Intelligent Transportation Systems (ITSC 2019), Auckland, New Zealand, 27–30 October 2019, in press.
37. Gosavi, M.A.; Rhoades, B.B.; Conrad, J.M. Application of Functional Safety in Autonomous Vehicles Using ISO 26262 Standard: A Survey. In Proceedings of the SoutheastCon 2018, Saint Petersburg, FL, USA, 19–22 April 2018; doi:10.1109/secon.2018.8479057.
38. Wachenfeld, W.; Winner, H. Die Freigabe des autonomen Fahrens. In *Autonomes Fahren: Technische, Rechtliche und Gesellschaftliche Aspekte*; Maurer, M., Gerdes, J.C., Lenz, B., Winner, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 439–464, doi:10.1007/978-3-662-45854-9_21.
39. Ponn, T.; Fratzke, D.; Gnandt, C.; Lienkamp, M. Towards certification of autonomous driving: Systematic test case generation for a comprehensive but economically-feasible assessment of lane keeping assist algorithms. In Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2019), Heraklion, Greece, 3–5 May 2019; pp. 333–342.

40. Ponn, T.; Gnandt, C.; Diermeyer, F. An Optimization-based Method to Identify Relevant Scenarios for Type Approval of Automated Vehicles. In Proceedings of the ESV—International Technical Conference on the Enhanced Safety of Vehicles, Eindhoven, The Netherlands, 10–13 June 2019.
41. Hermansdorfer, L.; Betz, J.; Lienkamp, M. A Concept for Estimation and Prediction of the Tire-Road Friction Potential for an Autonomous Racecar. In Proceedings of the 2019 IEEE 22nd International Conference on Intelligent Transportation Systems (ITSC 2019), Auckland, New Zealand, 27–30 October 2019, in press.
42. Wischnewski, A.; Betz, J.; Lohmann, B. A Model-Free Algorithm to Safely Approach the Handling Limit of an Autonomous Racecar. In Proceedings of the 8th IEEE International Conference on Connected Vehicles and Expo (ICCVE 2019), Graz, Austria, 4–8 November 2019, in press.