IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 6 April 2023, accepted 30 April 2023, date of publication 3 May 2023, date of current version 10 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3272750

APPLIED RESEARCH

Learn to See Fast: Lessons Learned From Autonomous Racing on How to Develop Perception Systems

FLORIAN SAUERBECK[1], (Member, IEEE), SEBASTIAN HUCH[1], FELIX FENT[1], PHILLIP KARLE[1], DOMINIK KULMER[1], AND JOHANNES BETZ[2], (Member, IEEE)

[1]Department of Mobility Systems Engineering, TUM School of Engineering and Design, Institute of Automotive Technology, Technical University of Munich (TUM), 85748 Munich, Germany
[2]Department of Mobility Systems Engineering, Professorship of Autonomous Vehicle Systems, TUM School of Engineering and Design, Technical University of Munich (TUM), 85748 Munich, Germany

Corresponding author: Florian Sauerbeck (florian.sauerbeck@tum.de)

This work was supported in part by the Technical University of Munich, in part by the Bavarian Research Foundation (BFS), in part by the Leibniz Supercomputing Centre, in part by the FlexLab, and in part by the Allgemeiner Deutscher Automobil-Club (ADAC) Foundation.

ABSTRACT The objective of this work is to provide a comprehensive understanding of the development of autonomous vehicle perception systems. So far, most autonomy perception research has been concentrated on improving perception systems' algorithmic quality or combining different sensor setups. In our work, we draw conclusions from participating in the Indy Autonomous Challenge 2021 and its follow-up event in Las Vegas 2022. These were the first head-to-head autonomous racing competitions that required an entire perception pipeline to perceive the environment and the opposing surrounding vehicles. Our research includes quantitative results from collected vehicle data and qualitative results from simulation, video, and multiple race analysis. The Indy Autonomous Challenge was one of the few research projects that considered the entire autonomous vehicle. Therefore, our findings indicate insights on the system level, including hardware setup and full-stack software. We can demonstrate that different sensor modalities in the vehicle have strengths and weaknesses when they are deployed. Our results further show the difficulties and challenges that emerge when multi-modal perception systems must run in real-time on real-world autonomous vehicles. The most concise finding from our investigation is the summary of critical learnings when developing and deploying perception systems for autonomous systems. Given the background of the study, it was inevitable that our conclusions were influenced by driving on the racetrack and only one hardware setup available. Therefore, in the discussion, we draw further parallels to driving on public roads in dense traffic. More studies are needed to investigate the development and deployment of multi-modal perception systems for autonomous road vehicles with different hardware setups and various object detection, localization, and prediction algorithms. The novel contributions of this work are given by 12 lessons learned, summarized in 5 categories. These were derived and validated through a realized real-world application project. The videos of the final events in Indianapolis and Las Vegas can be watched here:
IAC: https://www.youtube.com/watch?v=ERTffn3IpIs&ab_channel=CNETHighlights
AC@CES: https://www.youtube.com/watch?v=df9f4Qfa0uU&ab_channel=CNETHighlights
Multiple modules of the software stack are open source: https://github.com/TUMFTM.

INDEX TERMS Autonomous racing, autonomous vehicles, perception systems, software development.

The associate editor coordinating the review of this manuscript and approving it for publication was Junho Hong.

I. INTRODUCTION

Competition stimulates not only business, but also research. This has been demonstrated by various competitions and

challenges in the past. In the field of autonomous driving, the DARPA Grand Challenges in the mid-2000s led the way for the entire industrial sector as we know it today [1]. Building on this, another competition was launched in 2021: The Indy Autonomous Challenge (IAC) and its successor, the Autonomous Challenge at CES in Las Vegas (AC@CES) in 2022 [2]. The goal was to advance the state of the art in autonomous vehicles (AVs) by taking them to the racetrack and teaching them to compete against each other in a high-speed wheel-to-wheel competition. The TUM Autonomous Motorsport race car can be seen in Fig. 1 during an overtake.

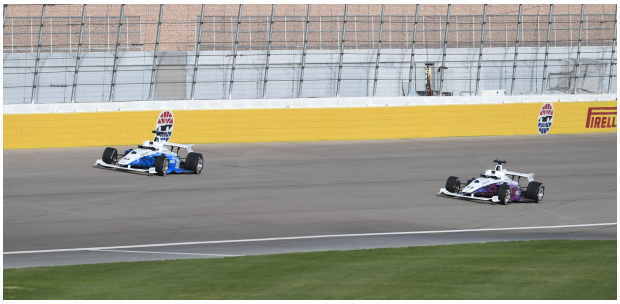


FIGURE 1. TUM overtaking Euroracing's AV-21 during the high-speed event at the Las Vegas Motor Speedway. Own foto.

A. STRUCTURE OF THIS PAPER

First of all, we want to introduce the structure of this paper for better comprehension during reading. In the Introduction, we present the necessary information about the challenges we are referring to, namely the IAC and the AC@CES. We also present the used vehicle platform and its sensor setup and motivate this work based on the experiences we had throughout the challenges. After the Introduction, we review related work which can be summarized into three subcategories: software and hardware, software only, and autonomous racing. On the basis of the previous work presented, we derive the research gap, and thus the novel contributions of this work. After giving a short overview of the challenges and the results we achieved, the 12 different lessons learned are presented, which can be categorized into five categories. Fig. 2 visually shows the learnings and their categorization. These lessons learned are the main contribution of this work. The Discussion section explains open research topics and discusses how our findings from racing can be translated into public traffic. In the end, everything is concluded comprehensively. The lessons learned are summarized and can be taken as advice for future robotics and autonomous driving projects.

B. INDY AUTONOMOUS CHALLENGE

The participants of the challenges were nine research teams from universities around the world. Each team participated with a copy of the autonomous race car developed specifically for the challenge, the *Dallara* AV-21. Thus, the whole hardware setup was the same for all teams and the scope of the challenge was limited to software. The teams' task was

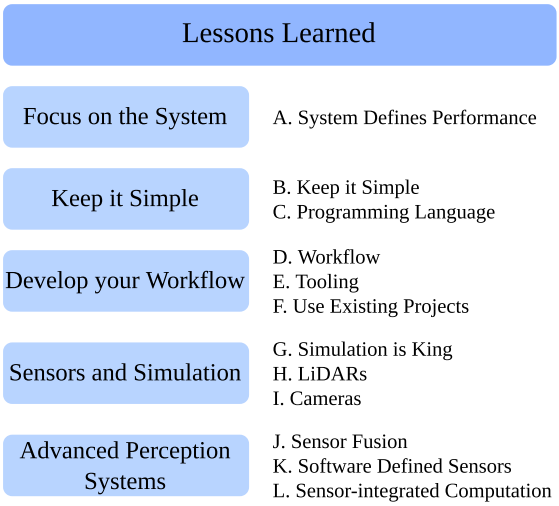


FIGURE 2. Main contribution of this work: the lessons learned presented.

to develop, deploy, validate, and test a complete autonomous driving stack, from sensor data to vehicle actuation. Because of the challenge's tight schedule, software development had to begin before the vehicles were built. This made the simulation of the vehicles and the environment an important task to be successful in the final competition.

The IAC took place on October 23, 2021, in Indianapolis, and the AC@CES was held during the CES in Las Vegas on January 07, 2022. The details and rules for both events can be found in [3] and [4].

C. VEHICLE PLATFORM

The specially developed *Dallara* AV-21 was based on a *Dallara* IL-15 chassis known from the Indy Lights Series [5]. The computing platform, perception sensors, and additional components such as low voltage (LV) batteries were installed inside the cockpit. Therefore, the driver seat and steering wheel were removed. All components of the system are listed in Table 1. The central computer from *ADLink* incorporated an 8-core *Intel Xeon E-2278GE* CPU with 64 GB RAM and an *Nvidia Quadro RTX8000* GPU with 48 GB of memory. All sensors were connected via Ethernet to a network switch, which transfers all data to the computer via a 40 Gbit QSFP+ interface.

The global navigation satellite system (GNSS) consisted of two dual-antenna setups with *NovAtel PwrPak7D* receivers to provide redundancy. Each of them contained a dedicated inertial measurement unit (IMU). For environmental and object detection, the system used camera, light detection and ranging (LiDAR) as well as radio detection and ranging (RADAR) sensors. The camera setup used a total of six cameras to cover a 360° area around the vehicle. The cameras that were aimed at the sides and rear of the vehicle had a field of view (FoV) of 102.8° horizontally and 77° vertically each. They used an *Edmund Optics* lens with a focal length of 3.5 mm. The two front cameras used lenses with a higher focal length

TABLE 1. The perception components of the AV-21. The FoV is given horizontally and vertically (h x v).

Device	Manufacturer	Model	Frame rate	FoV (h x v)
2x GNSS Receiver	NovAtel	PwrPak7D Receiver	20 Hz GNSS, 100 Hz IMU	
3x LiDAR	Luminar	H3	1 Hz - 30 Hz	120° x 0° - 30°
2x Front Camera	Allied Vision	Mako G319C, 12 mm focal length	up to 37.6 Hz at full resolution	34° x 24°
4x Side Camera	Allied Vision	Mako G319C, 3.5 mm focal length	up to 37.6 Hz at full resolution	102.8° x 77°
2x Side RADAR	Aptiv	MRR	10 Hz	90° x 5°
1x Front RADAR	Aptiv	ESR 2.5	10 Hz	90° x 4.4° (short range), 20° x 4.4° (long range)
Computing Platform	ADLink	AVA-3501		
Network Switch	Cisco	IE500		

(12 mm) to enable higher detection ranges. They had a FoV of 34° horizontally and 24° vertically and were placed in a stereo setup with a baseline of 24 cm. The three LiDARs also covered a FoV of 360° in total. Each of them had a horizontal FoV of 120° and a configurable vertical FoV between 0° and 30°. The RADARs were placed with one facing forward and one on each side at ± 90° horizontally. The front RADAR alternated between short-range and long-range detection.

Fig. 3 shows the arrangement of the perception sensors of the AV-21.

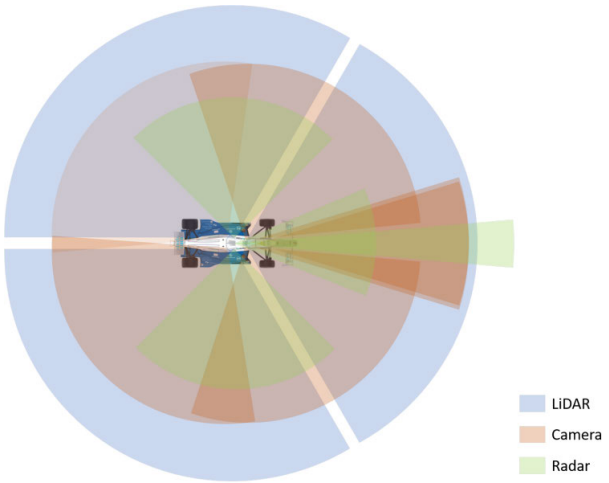


FIGURE 3. Perception sensor setup of the AV-21.

The vehicles were equipped with a state-of-the-art sensor suite and computing platform. Since most sensor suites of autonomous vehicles are equipped with fewer sensors, this also allowed us to draw conclusions for other vehicles with other arrangements.

D. MOTIVATION

At the overtaking competition in Las Vegas, our car spun out at a speed of about 270 km h^{-1}. The root cause of this lies in the perception system: Instead of one car, the perception pipeline predicted multiple opponent cars and initiated an evasive maneuver that could not be controlled at such a speed. Subsequent investigations showed that this failure could have been prevented in various ways: by adapting the object detection or the tracking and prediction pipeline. This led us to the question of how to find and implement an ideal perception system for a real-world application as given in this work.

II. RELATED WORK

Autonomous vehicles have already competed against each other in the past, starting with the DARPA Grand Challenges in the mid-2000s [1]. In contrast to many existing algorithmic challenges such as the KITTI benchmarks [6], the Argoverse competitions [7], or the Waymo Open Dataset Challenge [8], not only specific algorithms, but a whole running system had to be developed, deployed, and tested at the DARPA Grand Challenges.

A. DARPA CHALLENGES

An overview of autonomous vehicle research conducted through the DARPA Grand Challenges is provided in [9]. The Stanford Artificial Intelligence Laboratory, the winner of the first challenge, published its software and development structure in [10]. Since the publication explains the entire software stack of the team, the perception system was dealt with from a high-level perspective. Also, the DARPA Robotics Challenge (DRC) initiated robotics systems research. Here, the focus was not on the automotive application, but on legged robotic systems for rescue applications. In 2015, the Florida Institute for Human & Machine Cognition (IHMC) published its conclusions from the DRC trials [11].

B. AUTONOMOUS DRIVING SOFTWARE SURVEYS

In the field of autonomous vehicle research, Yurtsever et al. [12] gave a detailed overview of the state of the art in autonomous driving and autonomous vehicle systems. They summarized and compared existing survey papers. Different approaches were considered, from highly modular software stacks to end-to-end driving. Naz et al. [13] provided a survey of intelligent autonomous vehicle systems with a focus on state-of-the-art artificial intelligence (AI)-based algorithms. The survey of Ma et al. [14] also presented AI applications for autonomous driving. Perception algorithms were explained, however, the focus was on the whole autonomous driving software stack. An overview of deep learning perception

software with a focus on object detection and semantic segmentation was published by Jebamikyous and Kashef [15]. Other surveys focused more on hardware and mainly sensor technologies. Mohammed et al. [16] presented a review of different sensor technologies for different functionalities under various conditions. Vargas et al. [17] also focused on sensors and investigated their vulnerability under harsh weather conditions.

C. SENSOR SETUP AND FUSION

Research has been carried out in the field of sensor modalities and their optimal purposes of use. According to the current state of research, different sensors have to be fused to enable Society of Automotive Engineers (SAE) Level 5 autonomous driving in different situations, e.g. adverse weather conditions. Marti et al. [18] compared different sensor modalities and outlined their potential use cases. They came to the conclusion that different sensor modalities are suited for different applications in the field of autonomous driving, which leads to the need for sensor fusion. Wang et al. [19] and Yeong et al. [20] also highlighted the importance of fusing heterogeneous sensor data to create reliable perception systems and compared different implementation approaches on the algorithmic side. LiDAR sensors are currently the most precise range sensors, however, it is difficult to extract semantic descriptions from point clouds [21]. To reduce these problems, a multisensor data fusion is needed according to [22].

The current state of the art in sensor fusion was presented by Fayyad et al. [23]. Methods based on deep learning were presented and further research topics were derived. However, aside from the harsh weather conditions, they focused only on algorithmic improvements at the software level, without considering hardware or workflow limitations.

A detailed overview of perception systems and simulators for autonomous driving was presented by Rosique et al. [24]. Sensors and their operating principles were compared and evaluated, and methods for combining and simulating data were presented. However, a combined view at hardware and software is missing. Furthermore, their findings were not related to an implemented application.

D. FULL-STACK SOFTWARE

Devi et al. [25] gave an overview of the state-of-the-art software for autonomous driving with a focus on camera-based perception approaches. This work focused on the algorithmic side and did not include different sensor setups, development workflows, deployment, or hardware. Velasco-Hernandez et al. [26] also added hardware considerations to their survey. The main challenge, which was outlined, is the demand for more accurate and robust algorithms, powerful computation platforms for real-time capability, and system validation. Lin et al. [27] focused on the development of a whole autonomous driving stack that includes hardware and software constraints. To analyze the system performance, a software stack based on state-of-the-art algorithms was implemented. The identified limiting software modules were the perception parts of the pipeline, namely localization, object detection, and object tracking. According to the authors, computing power remains the bottleneck for autonomous vehicles, even preventing these systems from benefiting from improved sensors such as higher-resolution cameras.

Zong et al. [28] presented a whole hardware and software stack. They showed the design process and architecture of the entire autonomous vehicle with a focus on low-cost hardware. Initial perception results were presented. A section on the software development and deployment workflow is lacking.

Recently, Kowalczyk et al. [29] proposed a framework to characterize automotive datasets for the development of perception systems. With their method, existing datasets can be assessed and weaknesses identified. However, it did not propose a way to develop perception systems for future autonomous driving applications.

Additionally, investigations have been carried out on the influence of the perception stack on the performance of the system. Falanga et al. [30] determined the maximum speed of drones depending on the perception system and the latency used.

E. SAFETY

The field of safety and risk assessment partially focuses on system-level considerations. Since most crashes are due to perception failures, and even a high number and coverage of sensors cannot account for this, autonomous vehicle (AV) safety research is carried out in the perception domain [31]. An overview of the implications of the entire AV software stack on safety considerations was published by Wang et al. [32]. Taeihagh and Lim [33] assessed different risks stemming from autonomous vehicles and used this information to rank government strategies. Ren et al. [34] provided security guidelines for AVs, including the perception system. In [35], a camera perception pipeline was used to quantify the safety of AVs. To work with models of whole autonomous driving stacks, Tlig et al. [36] tried to model whole software stacks to build a safety concept upon. Almeaibed et al. [37] used a digital twin approach for future safety considerations. It is shown how perception approaches can mitigate the risk of cyber attacks. Also, the DARPA Grand Challenges have generated research in the area of safety for automated vehicles. McBride et al. [38] drew conclusions for the safety of AVs from the DARPA Grand Challenges.

F. FORMULA STUDENT DRIVERLESS

Within the Formula Student Driverless (FSD) competitions, student teams develop kart-sized race cars from scratch, including the vehicle platform and the autonomous system. This resulted in research in the field of hardware and software development for AVs. On the perception side, vehicles must

be able to detect cones that mark the track and map the racetrack online. Since there is no head-to-head racing, other cars do not have to be detected. KA-Raceing, the FSD team from the Karlsruhe Institute of Technology, presented its vehicle and software stack for the FSD competitions in [39] and [40]. The perception pipeline was mainly based on 3D LiDARs and was built on open-source algorithms, such as GraphSLAM [41] to build a track map. Nekkah et al. [39] also gave a brief insight into the development process and how the cars evolve from year to year.

In addition, AMZ Driverless, the FSD team of the ETH Zurich, published its approach [42]. They had parallel pipelines for camera and LiDAR perception and used fastSLAM [43] for mapping. Based on their results, some lessons learned were also presented. It is pointed out that the performance is defined by the whole system rather than by individual algorithms. Thus, slimmer algorithms might outperform more enhanced ones through better manageability [42].

G. INDY AUTONOMOUS CHALLENGE

The TUM Autonomous Motorsport team introduced its software for the IAC and the AC@CES in [3] and [4]. Based on this software, the results of this work were produced. Furthermore, [3] contained some lessons learned for autonomous vehicle systems in general. In contrast to that work, the present paper focuses on the perception system and the related lessons learned.

Team KAIST from the Korea Advanced Institute of Science and Technology published their full-stack approaches for the IAC [44]. They also explain their perception approach which is similar to the one presented by the TUM Autonomous Motorsport team and mainly focuses on LiDAR combined with a GNSS-based localization.

H. NOVEL CONTRIBUTIONS

Table 2 summarizes what topics are covered by the above-mentioned literature. Most research focuses on either algorithm development or sensor setups. The only works that have a scope similar to our work are from the DARPA challenges [10] and the Formula Student Driverless competition [39], [40], [42]. We are trying to close the research gap by providing a paper based on a realized application with a full state-of-the-art sensor and software setup.

It becomes clear that the systems engineering of AVs and especially their perception systems are a limiting factor for autonomous driving and an ongoing research topic. Currently, a holistic view of all aspects of engineering, development, testing, and deployment is lacking.

Building on our experiences from the IAC and the AC@CES, we want to present our main lessons learned from perception system development. This paper provides a system-level overview of the perception pipeline and derives lessons learned based on the results we obtained with it. Therefore, we will discuss the relevant topics from the hardware setup of the race cars to the software tools and development to the final target platform deployment.

Summarized, the novel contributions of this paper are the following:

- Development from scratch to a complete autonomous driving perception software stack for the final races,
- Development of a perception software pipeline plus a whole simulation environment,
- Consideration of the entire pipeline and processes: development, software, hardware, simulation, workflow, tooling, etc.,
- Real-world application: Everything described in this work was applied to a real race car.

III. EXPERIMENT AND RESULTS

The findings published here are based on the experiences of the TUM Autonomous Motorsport team during the concept, development, implementation, validation, and testing phases for the IAC and the AC@CES, respectively. During the challenges, the team proved that the entire software stack can reach speeds of up to 74 $\mathrm{m\,s^{-1}}$ (266.4 $\mathrm{km\,h^{-1}}$) and still safely overtake detected opposing cars [3], [4].

IV. LESSONS LEARNED

In the following, we present and explain the core findings we have gained about the development and deployment of a perception stack for an AV. In this work, we will not focus on detailed explanations of the algorithms but on the system level, how the modules work together, and how they influence the overall performance. Since the hardware was predetermined for the IAC and we could not design the perception hardware for our vehicle ourselves, we cannot consider the selection and arrangement of the sensors and hardware. However, we had the freedom to set up and configure the available hardware as we wanted.

The tight schedule of the IAC and the AC@CES enforced a fast and agile development workflow as the teams had only roughly 18 months from the first concept to the final race.

A. THE SYSTEM DEFINES THE PERFORMANCE

First of all, the most important thing we learned is that vehicle performance is determined by the overall system. Thus, even the best and most advanced algorithms will not improve system performance if they do not fit the overall concept of the system. The weakest link in the chain limits overall performance. Most of the time from concept generation to implementation to deployment and testing was spent on the application and debugging of the software. This increases not only the development time, but also the time needed to properly set up the software. Our experience shows that a simple algorithm with lower theoretical potential often performs better than advanced algorithms because more time can be invested in parameterization and optimization as well as integration into the overall software. The high potential of algorithms, which is usually presented in associated publications, cannot be reached in many cases under real-world conditions and without perfect fine-tuning for a specific dataset.

TABLE 2. Overview of related work for perception systems of autonomous vehicles. Most research focuses on algorithms and sensor setup.

Paper	Object Detection	Localization	Simulation	Sensor System and Hardware	Software Development	Testing	Real-World	Year
[10]	✓	✓		✓	(✓)	✓	✓	2006
[12]	✓	✓		✓				2020
[13]	✓	✓		✓				2022
[14]		✓	✓		✓	✓	✓	2020
[15]	✓			(✓)				2022
[16]	✓	✓		✓				2020
[17]				✓		✓		2021
[18]				✓				2019
[19]	✓			✓				2019
[20]				✓				2021
[21]	✓			✓				2020
[22]	✓	✓		✓				2018
[23]	✓	✓		✓				2020
[24]			✓	✓				2019
[25]	✓							2020
[26]	✓	✓		✓				2020
[27]	✓	✓		✓				2018
[28]	✓	✓		✓		✓	✓	2018
[29]				✓				2022
[39]	✓	✓	(✓)	✓		✓	✓	2020
[40]	✓	✓	✓	✓		✓	✓	2022
[42]	✓	✓	✓	✓		✓	✓	2020
[3]	✓	✓	(✓)	(✓)		✓	✓	2020
[44]	✓	✓	(✓)	(✓)		(✓)	✓	2020
Ours	(✓)	(✓)	✓	✓	✓	✓	✓	2023

Another important point about the overall performance of the system is the interaction of the individual software modules. Latency is often not considered in scientific publications. However, end-to-end latency has an enormous influence on software performance and robustness [45]. As a result of this, it is crucial to test the algorithms in conjunction with each other.

For example, we used several parallel object detection pipelines from different sensor modalities. These were fused and tracked in a separate module. When evaluating the performance of the object detection pipelines as standalone modules, the absence of objects as false negatives (FN) or outputting false positives (FP) has a huge impact on the conventional performance metrics such as mean average precision (mAP). However, in conjunction with the temporal relationships considered in the tracking module, many of these errors can be neglected, while other correlations, which might not be covered by standard metrics, show greater influences. Therefore, we tried to validate every software change in the context of the entire software stack.

B. KEEP IT SIMPLE

One of our key performance factors for the challenges was to keep the perception system simple and understandable. This allowed us to identify the most significant performance limitations and improve them accordingly. After evaluating various additional sophisticated approaches, the validation of these in the full software stack simulation revealed that there was no overall performance improvement mainly due to lower robustness, more complex parameterization, or higher end-to-end latency. Thus, we ended up winning the IAC with seemingly simple approaches on the sensing and perception side: Due to limited resources, our car did not have time synchronization between the computer and the sensors, nor were the sensors triggered synchronously. Therefore, we managed to develop robust software that allowed for maximum test time on track.

In terms of algorithms, we ended up using neither LiDAR nor visual localization algorithms. Localization was based on real-time kinematic (RTK)-corrected GNSS fused with raw IMU measurements using an Extended Kalman Filter (EKF).

For the detection of other vehicles, the main pipeline was based on LiDAR clustering with preceding point cloud filtering. A deep learning-based pipeline was also developed and deployed, but since the GPU did not work in the final race, object detection was based on LiDAR clustering fused with RADAR for long-range detections and velocity measurement in the tracking algorithm. A major advantage of conventional algorithms we experienced is the limited data dependence. Deep learning-based approaches are heavily dependent on available data. Thus, data collection has to be done before we can use these algorithms effectively. However, it has to be mentioned that with sufficient data, these algorithms will probably outperform simpler approaches. This can be seen in the object detection benchmarks [6], [7], [8].

Table 3 shows a short comparison of the deep learning-based *PointRCNN* and the CPU-based clustering algorithm. It can be seen that for our application and our limited labeled data sources, the clustering is capable of even outperforming the *PointRCNN*. However, it has to be mentioned that this comparison already excludes some FPs from clustering that are neglected as they are outside the track. We tuned the

TABLE 3. Comparison of PointRCNN and point cloud clustering.

Algorithm	Precision	Runtime	Hardware
PointRCNN	52.0 %	83.4 ms	Nvidia Quadro RTX8000 GPU
Clustering	59.0 %	89.1 ms	Intel Xeon E-2278GE CPU

clustering to fit within our overall perception concept. Therefore, we allowed more FPs as the tracking filtered them out in the next step. Also, with more data and more training iterations, the performance of the *PointRCNN* will further improve.

Due to the short time, limited human resources, and mainly limited compute and network resources, we did not use a camera object detection pipeline. This simplified the sensor fusion in bird's-eye view. In summary, our perception pipeline was not based on any deep learning algorithms during the final events. Due to the shift of resources to a better tuning of the simpler approaches, we still achieved a stable detection range of around 80 m.

To save time during software development, large parts of the software were written in *Python*. This enabled simpler ''quick-and-dirty'' implementations compared to *C++*. Also, debugging is easier and sometimes more meaningful as the code is not compiled but interpreted at runtime. The result is a compromise between the investment in development time and the performance at runtime of the implemented software.

An example of the development of an advanced module that was simplified to the relevant functionalities for optimal overall software performance is the localization module (Fig. 4). Here, the original localization module introduced in [46] is shown. The plan was to use a combination of LiDAR and camera with known track information and the possibility to map the track before racing. LiDAR was planned to detect the boundary and calculate distance and orientation relatively. The camera should use an adapted visual SLAM for longitudinal localization which requires features that could only be found outside the LiDAR range due to the racetrack environment with few features. With information on the track layout, a transformation into *xy*-coordinates is possible. The first tests on the track have revealed that a combination of GNSS and IMU is sufficient, takes only a small part of the setup time on the track, and additionally saves a lot of computational resources. Thus, the modules

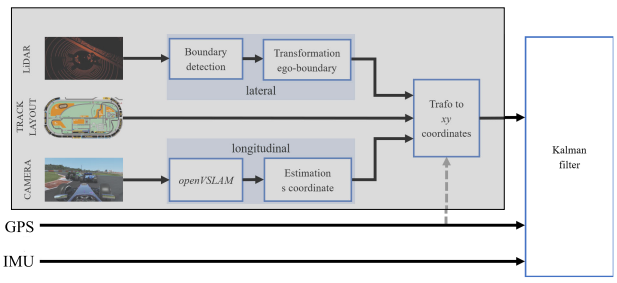


FIGURE 4. Originally planned localization algorithm and simplified final localization (without gray box).

within the gray box of Figure 4 were removed from the final software stack.

To be compliant with the 2D-based path planning algorithm, we even localized in 2D. Since the racetracks showed banking in turns of up to 20°, the *y*-acceleration had to be compensated so that the Extended Kalman Filter (EKF) could calculate as if it were on a flat surface. We made the assumption that the car drives parallel to the track boundaries and thus did not compensate for longitudinal acceleration.

The KAIST team took a similar approach for their perception pipeline [44]. They also relied on a relatively simple concept: GNSS-based localization and state estimation and a LiDAR detection pipeline consisting of ground filtering and clustering.

However, for other applications, such as drones, for example, and other circumstances, this approach might not work. Therefore, new assumptions will have to be found to simplify the system.

C. PROGRAMMING LANGUAGE

The choice of programming language is a key decision at the beginning of every software project. Due to the general constraint that we wanted to use the Robot Operating System (*ROS 2*) as a middleware, the decision was mainly between *Python* and *C++*. *ROS 2* allows writing separate nodes in different languages, so there was no global decision to be made. This allows ''offloading'' of runtime critical applications to *C++* without the need of rewriting everything. Here, a modular software approach has great benefit.

As mentioned in the previous chapter, compared to *C++*, in general, *Python* code is relatively easy to understand and to write, leading to an accelerated development process. Many packages are available and can be easily integrated. Developers do not have to worry about build systems, etc. This lowers the entrance barrier to robotic systems as less experienced developers have an easier start into this field. However, *Python* has major drawbacks and limitations. First, weaknesses in performance and power efficiency should be mentioned [47]. Also, in terms of safety, *C++* is advantageous as the memory is handled by the developer and the data types are fixed. Our car triggered emergency stops several times because the data types in the *Python* nodes did not match. This mainly occurred, when there were empty lists, for example for RADAR detections. Since *C++* nodes are compiled, these errors would occur at compile time and not at runtime. Furthermore, some performance-relevant implementations of *ROS 2* are currently not available in *Python*: Shared memory, zero-copy transport, and compilation of nodes into components, similar to the nodelets in *ROS 1*, are currently only available in *C++*. In general, *Python* can be said to allow fast development, but *C++* is significantly better in terms of performance and safety.

D. DEFINE YOUR WORKFLOW

To reduce development, deployment, and testing times, it was important for us to define a suitable workflow. Especially

larger teams working on extensive software projects benefit greatly from established well-designed workflows. The overall development workflow of TUM Autonomous Motorsport is depicted in Fig. 5. Each new commit on the *develop* branch is automatically tested via GitLab CI/CD functionality. If the tests are successful, *Docker* images for all software modules are automatically built and pushed to the container registry. These images could be deployed directly on the race car. Before that, they were all manually deployed to our Hardware-in-the-Loop (HiL) simulation environment and tested in race simulations. During the development phase, there were fixed software releases every month that were marked with GitLab tags. Later, when the actual race car was available, a new tag was released for each test session on a daily basis. The automatically built *Docker* images, tested through the CI pipeline, were manually deployed and tested on our HiL simulator, where different scenarios were covered. Thus, only fully tested images were deployed to the race car.

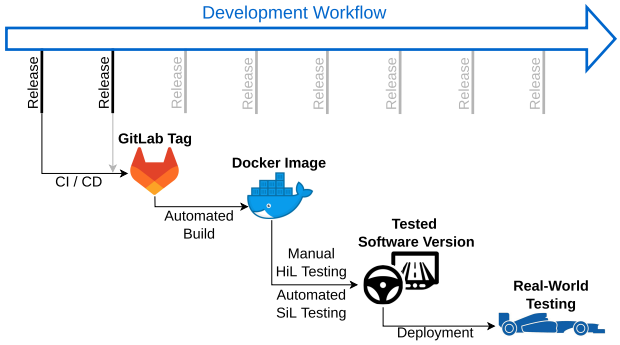


FIGURE 5. Software development, release, deployment, and testing workflow.

We decided to base our software design on container virtualization to avoid *"it-works-on-my-machine"* kind of problems with dependency management. In addition, this workflow allows the deployment and launch of ready-built containers instead of having to build the software stack on the target machine. To allow developers to easily exchange individual software modules, we used *Docker* to run the modules as a kind of microservice, as shown in Fig. 6. Since the message types between the modules were fixed, it was easy to swap and compare individual containers with their previous versions. A custom base image was used to base all images on. This prevented compatibility issues between software parts used by several modules, such as message types for communication between individual nodes. To launch and orchestrate the containers and their included software, we used *Docker Compose*. It allowed us to allocate sufficient resources for each software component by allocating specific cores for the tasks. The resources needed for each software module were measured before assembling all of the modules. This workflow turned out to be suited for our use case. However, for other projects or other software setups, different development and deployment processes have to be evaluated. Also, the fixed allocation of computing resources is not only a benefit but can also limit software performance in specific situations, where a certain module needs more resources. Another critical point that will have to be addressed in future works is the overhead introduced by containerization. In our case, it was negligible, however, it can make a difference in other projects.

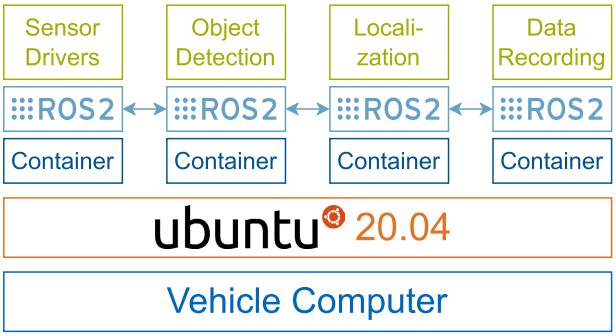


FIGURE 6. Applications running in *Docker* containers as microservices.

Establishing our development and deployment workflow from the beginning of the concept phase allowed us to win the IAC even though we were handed our own vehicle only six weeks before the race.

E. TIME INVESTED IN TOOLING PAYS BACK

Proper tooling has been an important accelerator for the progress of our software development from the beginning. Visualization is important when working with sensor data. Therefore, we mainly used *ROS 2 RViz*. For plotting recorded data from *rosbags*, the standard *ROS 2* data recording format, and also for live visualization of our telemetry, we made extensive use of the open-source tool *PlotJuggler*.[1] In addition to these standardized tools, each software module had its own data recording pipeline and data playback tooling. To allow smooth operation for each user, these tools were virtualized in *Docker* containers. *Shell* scripts and parameter files took care of the launch process of these tools.

For automatic labeling of detected vehicles, GNSS data was shared between teams. Based on these data and the timestamps contained, we developed a tool that automatically generates ground-truth data for each frame with an opponent vehicle. This was done by transforming the GNSS coordinates of the other vehicle into the local ego coordinate system and generating a 3D bounding box around it with the known vehicle dimensions. In the second step, the labels were refined more precisely before the quality of the labels was manually checked and assessed. Therefore, the 3D point clouds and bounding boxes were visualized and the label was manually shifted to perfectly fit the opponent race car if necessary.

Although we had to invest a lot of time in the development process of these tools, this time was saved many times in the end, as the entire development and software optimization

[1] https://github.com/facontidavide/PlotJuggler

process was greatly accelerated. It will lead to an accelerated evolution of the software as those tools support it. For example, we could use the automatically generated bounding box labels to develop more advanced deep learning-based object detection algorithms.

F. USE WHAT IS ALREADY THERE

To minimize development overhead, especially for software and debug tools, we have resorted to many available open-source projects. Small adjustments for the specific use case can improve the development process and also the quality of the final project. For our case, in particular due to the tight schedule, we had to use our resources as well as possible to develop a fast and robust software stack for autonomous racing.

ROS 2 provides an ideal platform for the development of robotic projects of different scales. It can handle individual software modules and control the communication between them. This also makes it ideal for team collaboration. Many existing nodes and tools such as *RViz* can be easily integrated into the workflow. As mentioned above, *PlotJuggler* has been an important tool for many teams. It allows the analysis of recorded *rosbags* and visualization of the contained data. It was also used as a live telemetry display to visualize current sensor and vehicle data while the car was on track. *Foxglove*[2] provides great capabilities for replaying and visualizing data directly from *rosbag* files. For the deployment of our software modules, we relied on *Docker* and *Docker Compose*.

The great possibilities that existing open-source algorithms offer can be seen by a quick look at our perception and especially object detection pipelines. The camera object detection pipeline was based on Yolo v5[3] [48], and the two LiDAR pipelines were based on a clustering algorithm from the PCL library[4] and *Autoware*[5] and a deep learning pipeline based on PointRCNN[6] [49]. An overview of the originally planned object detection pipeline is shown in Fig. 7. Originally, the primary and main pipeline was planned to be a deep learning-based LiDAR algorithm, supported by deep learning-based camera detections. The LiDAR clustering was planned as a backup detection pipeline and for debris detection. The algorithm is a euclidean cluster extraction that uses pre-filtered non-ground points as input. First, these points are brought into a Kd-tree representation. Then, the algorithm iterates through all points and searches for neighbors within a defined radius [50]. The clustered points were planned to be compared with the geometry of the race car to decide if the object is another car or something else. All of the mentioned algorithms are available open source which gave us the possibility to implement multiple approaches and compare them. If we had to implement entire object detection algorithms from scratch, the decision on the final concept would have had to be made significantly earlier. In the end, the main pipeline was LiDAR clustering, even without geometry matching. Using this easier algorithm made us independent of real-world test data and training data, and made object detection less dependent on the fine-tuning of the algorithm and training process. This, again, shows the importance of another major lesson learned: keep it simple.

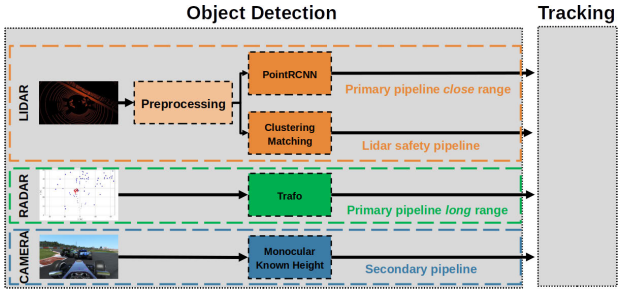


FIGURE 7. Initially planned object detection pipeline. The main pipeline used in the final races was LiDAR clustering, originally only planned as backup and debris detection.

G. SIMULATION IS KING

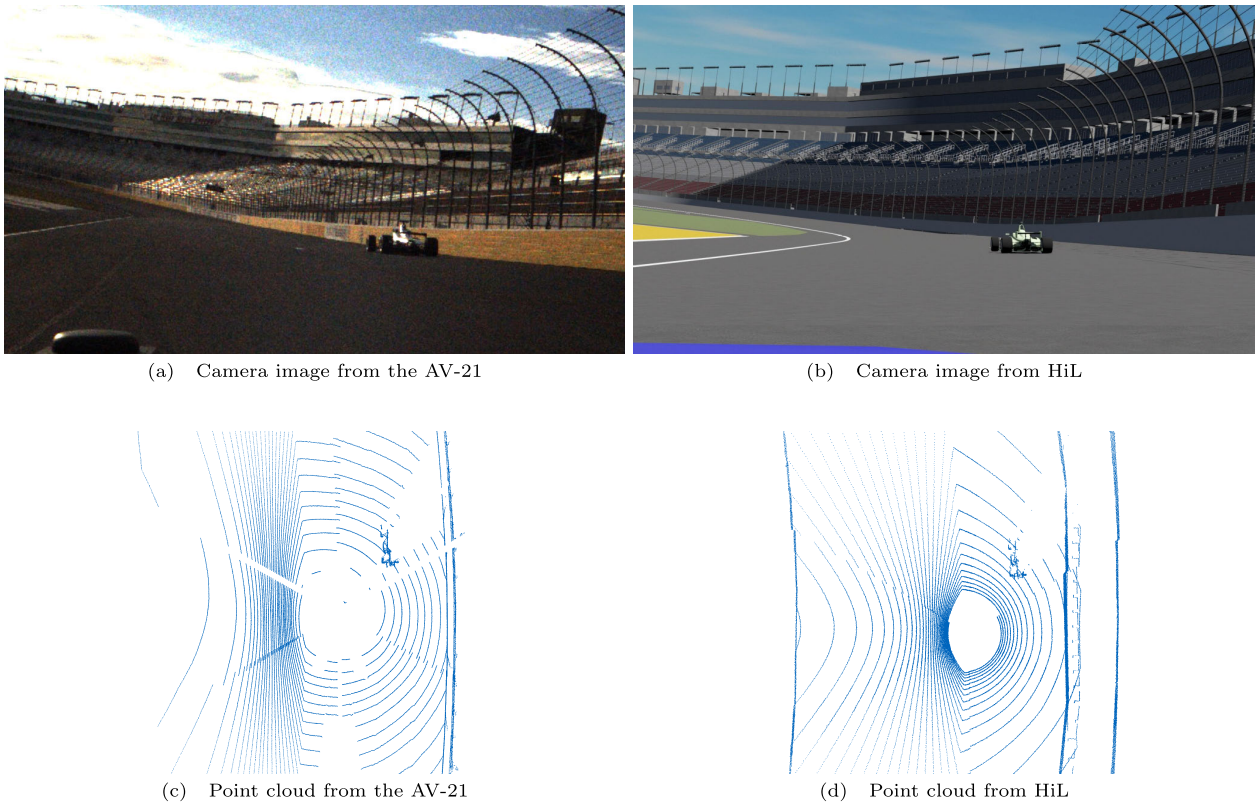
Due to the tight schedule of the challenges, it was crucial to start developing the software pipeline long before the final configuration of the vehicle was known. To enable different levels of simulation, we developed three simulation environments:

- Module-specific simulations, e.g. to replay a single module: Module-in-the-Loop (MiL)
- Software-in-the-Loop (SiL)
- Hardware-in-the-Loop (HiL)

In SiL, we were able to replay the entire software stack except for the perception. To also simulate perception modules, we developed sensor models based on *Unity* [3]. The high flexibility of the simulation environment allowed us to always update the vehicle model with the current configuration planned by the organizers and to evaluate the current performance of our perception software. This progressing simulation of the full-stack software meant that we already had executable software before we touched our race car for the first time.

Even in the early phases, where the simulation results did not come close to the real-world data, the use of simulation enforced running software with working communication between all modules and tolerable runtimes for real-time execution. In later phases with more realistic simulation, synthetic data was also crucial for including new features in the working software stack. Each new feature was tested in HiL in various scenarios and compared to previous performance. When all tests were passed, a *Docker* image was built and pushed to our *GitLab* container registry. Only previously tested software versions were deployed to the real vehicle and validated on the racetrack. The time shift between the racetracks in the United States and the simulation team in

[2]https://foxglove.dev/
[3]https://github.com/ultralytics/yolov5
[4]https://pcl.readthedocs.io/en/latest/cluster_extraction.html
[5]https://www.autoware.org/
[6]https://github.com/sshaoshuai/PointRCNN



(a) Camera image from the AV-21

(b) Camera image from HiL

(c) Point cloud from the AV-21

(d) Point cloud from HiL

FIGURE 8. Comparison of real-world data and simulated data.

Germany allowed complementary working shifts to make the most efficient use of time.

Also for the assessment of non-functional performance, simulation was of great benefit. This allowed us to determine interfaces between the software modules early and thus make the individual development processes more independent. Even before we knew the hardware specifications of the final computer and were able to measure the computation times of our algorithms, we were able to determine the relative times compared to other modules. In this way, we were able to identify computational bottlenecks at an early stage of the development process and adjust the software architecture accordingly.

What we identified to have the greatest room for improvement in terms of perception simulation are sensor models, especially the camera and the RADAR model, and the environment model. LiDAR models based on ray tracing with additional noise provide relatively realistic data for non-deep learning algorithms. However, deep learning algorithms exhibit completely different behavior, even if the simulated data look comparable to the human eye. A comprehensive analysis of the sim-to-real gap in our LiDAR simulation was conducted by Huch et al. [51]. Future research is needed to reduce this gap and make simulated sensor data even more valuable for perception algorithm development. To the best of our knowledge, no physical RADAR model is currently publicly available in any open-source simulation environment for autonomous driving. Our camera model only depicted the environment from the camera's perspective. Camera-typical effects, such as motion blur, were not taken into account. Fig. 8 shows a comparison of simulated and real sensor data. The quality of the simulated camera data highly depends on the 3D environment model, since textures mainly determine the appearance of the image. A major drawback of all simulation environments is the high computational effort which makes it impossible to generate synthetic data for the whole sensor setup in real-time. Thus, to simulate closed-loop in real-time, a simplified perception simulation and inference had to be used. The whole realistic sensor setup could be simulated slower to generate synthetic datasets for algorithm validation.

H. LIDARS PROVIDE GOOD DATA 'OUT OF THE BOX'

During the challenges, we took full advantage of the comparably easy-to-use LiDAR data. LiDAR sensors output the captured information directly in the real-world 3D frame. Thus, no additional assumptions have to be made to transform the data from a 2D plane into the real-world frame as for cameras. This also leads to less dependence on the quality of the internal calibration. Typically, LiDARs do not require internal calibration. Only LiDAR-to-LiDAR and/or LiDAR-to-chassis calibrations need to be done.

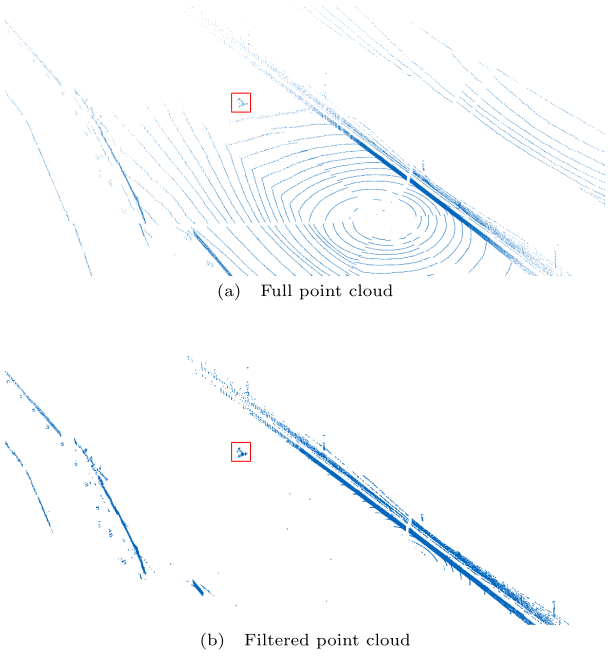


FIGURE 9. LiDAR point cloud with an opponent vehicle at a distance of 50 m.

To maximize the performance of LiDAR-based algorithms and the information content of the sensor data, intelligent pre-processing of the point clouds was crucial for us. The pre-filtering consisted of a geometric filter (conditional removal), a voxel filter, and a ground filter to reduce the number of points. Fig. 10 shows the reduction of contained points throughout the pre-filtering steps. The filtered point cloud contains less than 20 % of the points in the raw input point cloud. The computation time for this step is around 22 ms [3].

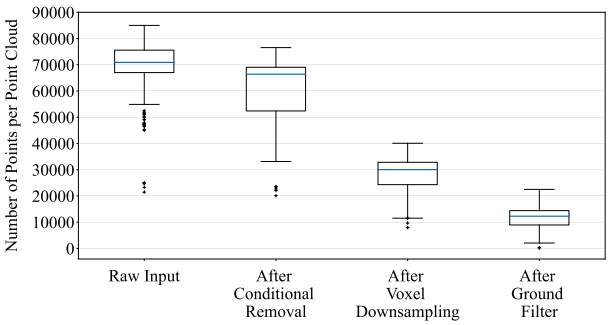


FIGURE 10. Reduction of points in point clouds for the final race of the CES [3].

The complete point cloud (a) and the point cloud after filtering (b) are depicted in Fig. 9. Unnecessary information is removed, and it becomes less computationally expensive to detect the opposing race car. The ground filter is based on the one integrated in the *Autoware* software stack. It iterates from the ego vehicle through the reflection rings (as seen in Fig. 9) and checks if the next ring is within a certain slope to determine whether it also belongs to the ground or another object. This approach is highly dependent on parametrization, especially to work in difficult situations like the banked turns. More research is needed to develop ground filtering algorithms that are more general and perform better.

Due to the representation of the data in point clouds and the smaller amount of data compared to cameras for a 360° FoV around the ego vehicle, no compression is required for the communication or recording of LiDAR data.

The specific information representation of the sensor data contained in LiDAR point clouds also makes it possible to obtain decent perception results with conventional algorithms without the need for deep neural networks (DNNs). In our case, a clustering algorithm applied to the pre-filtered point clouds provided stable detection results over a range of about 80 m. A major advantage of these algorithms is their adaptability from simulation data to sensor data. Since clustering is not as data-centric as DNNs, the dependence on the data source is limited, as no training data are needed.

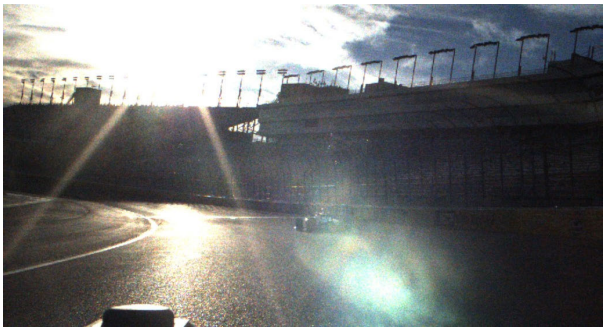
Since the data are sparser, LiDAR algorithms show a runtime benefit on conventional computing resources. This makes them more independent of hardware accelerators or GPUs. Furthermore, many existing open-source algorithms and tools in *ROS 2* can be used directly with the default *PointCloud2* data format.

I. CAMERAS ARE HARD TO HANDLE. HOWEVER, THEY HAVE GREAT POTENTIAL

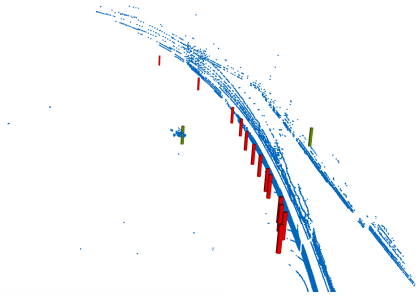
During the final events in Indianapolis and Las Vegas, most teams did not have a camera perception pipeline in use. The main reason for this was the limited time available to develop and set up the perception system. Unlike the LiDARs, the camera drivers did not output directly usable sensor data.

A big issue when processing camera image data is enormous data throughput. In our case, we had to provide a Gigabit interface for each of the six cameras. The switch, the network, and the central computer must be able to handle this amount of data. Additionally, when recording *rosbags*, the writing process on the drive can be limiting. To prevent this, data compression is required. Compression can be performed on dedicated hardware, the CPU, or the GPU. However, this requires computational resources that may also be needed for perception algorithms and adds latency.

Another topic that complicates the integration of cameras into a perception system compared to LiDARs is calibration. To compensate for distortions, etc., cameras require internal calibration. This is usually done by using dedicated checkerboards. Algorithms can be quite sensitive to the quality of the calibration. The exposure has to be set properly to obtain good images under different lighting conditions. Automatic exposure can be slow and fixed exposure can make it difficult to detect vehicles in the shade or in direct sunlight. Fig. 11 shows the comparison of camera, LiDAR, and RADAR data under unfavorable lighting conditions. Due to reflections and problems with automatic exposure, it is difficult to detect the car in the camera image even though it is only around 35 m



(a) Camera image



(b) LiDAR point cloud (blue) and RADAR point cloud (red: static objects, green: dynamic objects)

FIGURE 11. Comparison of camera, RADAR, and LiDAR data in harsh lighting conditions.

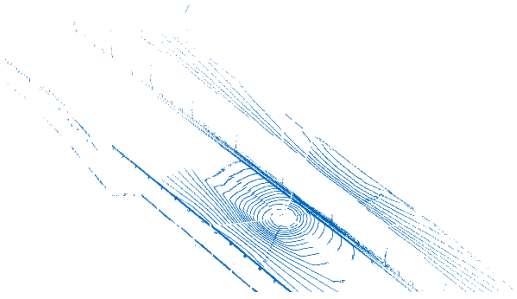
ahead. The LiDAR and also the RADAR have no problem detecting the car, as they are not operating with visible light.

When object detection is running on camera images, it is state of the art to execute the detection algorithm on each image stream separately which costs a lot of computing power. However, it is hard to fuse multiple camera images into a single image. This is because the data are not acquired in the real-world-3D frame as LiDAR point clouds. Therefore, with LiDARs, it is easy to transform different sensors into a common frame. Since cameras record data in their own 2D coordinate system, assumptions have to be made to fuse images and also estimate depth. On the algorithmic side, even a simple implementation of object detection requires DNNs. There are no basic algorithms such as LiDAR point clustering that can handle a complex task such as the detection of opponent race cars on the track properly. This also means that GPUs are crucial for efficient perception algorithms based on cameras - even more crucial than for LiDAR data. What makes development even more challenging is simulation. Currently, available simulators cannot produce realistic camera images.

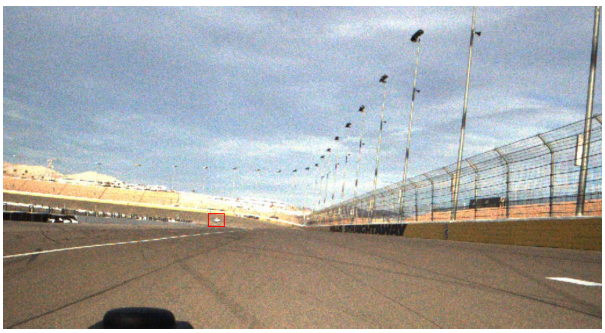
In summary, we can say that, compared to LiDARs, cameras are generally more difficult to:

- fuse,
- calibrate,
- simulate,
- recover 3D data,
- and handle the data load.

What we noticed on the other side during the challenges was the enormous potential that cameras offer. Fig. 12 shows data from the three LiDARs (top) and the front left camera (bottom). In the camera image, the safety car can still be seen, although it is more than 400 m ahead of the ego vehicle. The LiDARs do not receive any measured points from that distance. It should be mentioned here that this is also due to the high focal length of the front camera. The usage of different lenses allows adapting a camera sensor for its specific use case, e.g. using a telephoto lens for long-range and a fisheye lens for short-range detection. Furthermore, depending on the settings, cameras offer higher resolution and color detection. This allows the environment to be represented in more detail.



(a) LiDAR point cloud. Safety car out of range of the sensor



(b) Camera image. Safety car marked red

FIGURE 12. Range difference of camera and LiDAR.

In summary, we can say that range-measuring sensors such as LiDAR and RADAR offer an easier starting point to extract relevant information from the data. They are also less sensitive to external conditions as they are active sensors and offer algorithms which are less data-dependent. Having said that, cameras offer huge potential due to their low price, optical adaptability, high resolution, and color output.

J. FUSION OVERCOMES LIMITATIONS

The only way to combine the advantages of sensors is to fuse their data in a multi-modal perception pipeline. Generally, there are three options to do so:

- early fusion on raw sensor data,
- mid-level fusion on extracted feature level,
- late fusion on object level.

We decided on an EKF-based late fusion approach incorporated into the object tracking module, which was introduced

in [52]. This approach provides more flexibility, as it is not dependent on a single pipeline but can handle the available detection results. With it, heterogeneous detection pipelines with different measured features, update frequencies, sensor ranges, and detection accuracies can be used to output an optimized unique object list. Additionally, the fusion enables the completion of measured features. As an example, we fused the precise position measurement of the LiDAR detection with the RADAR detection, which measured the speed of objects. Furthermore, it is less dependent on a precise calibration as the raw sensor data are not transformed, but only the already detected bounding boxes are. Thus, the detection algorithms themselves do not depend on external calibration. This tracking method supported our modular software approach, as it could grow with the detection pipelines.

By means of the implemented fusion method, the tracking of surrounding objects with less than 0.1 m positional residuals and speed residuals below $0.15\,\mathrm{m\,s^{-1}}$ at the AC@CES at object speeds of up to $75\,\mathrm{m\,s^{-1}}$ were achieved. The fused detection range was up to 105 m to the front and 57 m to the rear [52].

For the next evolutions of our autonomous racing software stack, also other fusion methods will be taken into account. Especially early and mid-level fusion promise to outperform separate detection pipelines when applied correctly. However, this will increase implementation and test time and also increase the requirements for the data and the external sensor calibration, and thus, the late fusion was chosen for the presented approach.

K. BENEFIT FROM SOFTWARE CONFIGURABLE SENSORS

The AV-21 was equipped with three LiDAR sensors, each covering a horizontal field of view (FoV) of 120°, resulting in a total horizontal FoV of 360°. Since the LiDAR sensors were identical, only one driver was needed to transform each sensor's individual point clouds directly into the vehicle coordinate system and merge the three point clouds into a single point cloud. The sensor driver also provided the ability to dynamically configure the settings of each sensor, which we took advantage of. For our sensor, it was possible to adjust the scan rate, scan FoV, FoV center, and vertical scan pattern during runtime.

All three sensors operated at a constant scan rate of 20 Hz and a constant vertical FoV of 17.5° for the front and 20° for the left/right LiDAR, respectively. The scan pattern defines the vertical line distribution and has presets such as Uniform, Gaussian, or Exponential. We chose a Gaussian scan pattern that provided an increased resolution around a specified vertical region of interest (ROI). The ROI of the front LiDAR sensor was dynamically adapted on the basis of the vehicle's position on the track to react to different banking angles on straights and turns. On the front and back straights of the Las Vegas Motor Speedway, the banking angle is only about 6°. Here, a narrow ROI aligned to the horizon is important for a high detection range. However, with a banking of up to 20° in turns, this narrow ROI aligned with the horizon could result in the absence of LiDAR beams hitting vehicles at a higher distance in turns. Therefore, we increased the ROI width (increased standard deviation of the Gaussian distribution) and shifted the ROI center upward. In this way, we could scan the entire area in front of our vehicle in a turn. When entering the straight again, we resetted the ROI width and center.

The capability of dynamic sensor configuration provided a greater overall detection range throughout the track. In general, object detection performance can benefit from dynamically configurable sensors, also for road vehicles. For example, in combination with an IMU, the sensor's FoV can be adjusted to be constantly aligned parallel to the road surface and not be affected by the vehicle's pitch due to acceleration or deceleration. Additionally, the beam configuration can be dynamically adapted to different tasks, such as object detection or localization. In the future, sensor configuration can also be included in software optimization problems, as already shown [53].

L. SENSOR INTEGRATED COMPUTATION POWER CAN BE BENEFICIAL

An autonomous vehicle is a complex system that has to manage several computationally expensive tasks simultaneously. Therefore, computers in self-driving cars often exceed their limits, or algorithms have to be adapted to computing resources. Another problem is network traffic. When all raw sensor data are transmitted, also a lot of unusable data are transmitted. For systems with a large number of sensors, the network may reach its limits. Thus, the sensor data have to be reduced by a lower resolution, frequency, etc.

To mitigate the effects of these problems, it can be of great benefit to integrate the first processing steps directly into the sensors. By doing this, the amount of unused transmitted information can be reduced, computing power of the vehicle computer can be saved, transmission times of the data can be minimized and the amount of recorded data can also be reduced. The research progress in the field of low-power AI accelerators, based on field programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs), will boost this development, as lightweight machine learning tasks can also be directly executed on sensors. However, there are also disadvantages to this concept. The complexity of the system increases as the distributed software has to be developed and deployed. Therefore, centralized deployment of the vehicle computer is not possible. In addition, the hardware and programming limitations of the computing platform have to be considered.

For future perception systems, the technology of sensor-integrated computing power will definitely be interesting and should be pursued further.

V. DISCUSSION

In this work, we showed the importance of system and workflow considerations for the overall performance of autonomous vehicle perception systems. Our lessons learned and findings were generated and validated through the IAC

and the AC@CES real-world applications. In the following section, we want to discuss our work. Therefore, we will discuss how perception system performance can be evaluated and how the learnings from autonomous racing can be translated into public transport.

A. PERCEPTION SYSTEM EVALUATION

It is difficult to quantify perception system performance. Generally, the measurability of system performance will be an important research topic in the future. In the current state of research, there are many metrics that quantify the performance of individual algorithms and compare different approaches. At this point, there is no standardized evaluation metric for autonomous vehicles. Some new evaluation metrics that cover a few of the problems have recently been suggested [54], [55], [56]. However, none of them could establish as a new standard, since not a single one covers all of the problems.

In industry, the most widely used metric is kilometers or miles driven per disengagement. However, this does not include the complexity of the environment, and it also does not allow one to track the reasons for disengagements, and thus identify the weak points of the current software stack. Also in the scientific and research world, more focus should be placed on system-level results. Even the best working and most sophisticated algorithms will not contribute to overall performance if they are not properly integrated into the autonomous system. Sensitivities should not only be quantified for the output of a specific task (e.g. mAP for object detection) but also for the whole system: What benefit does a higher mAP bring, do we prefer precision or runtime, etc.

In our work, we took the vehicle hardware set-up as given and only configured it according to our needs, as partially explained in Section IV-K. Future research projects should consider the whole system as a problem to solve. In this way, perfect synergies between hardware and software can be found. On the one hand, it is desirable to design modular software that can be deployed to any hardware setup. On the other hand, hardware and software cannot be separated without compromise, as they depend heavily on each other. To validate the performance of the whole perception system, the hardware and hardware-software implications will have to be looked at.

B. WAY TO GO TO LEVEL 5

The autonomous driving application considered in this work represents a highly limited operational design domain (ODD), as we

- only drove in good weather conditions,
- knew the racetrack before racing,
- only had to consider objects of one type, and thus did not need any classification,
- and all other teams had to stick to strictly defined race rules.

Only the strict limitation of the application made these events possible. When aiming for Level 5 autonomous driving, the ODDs will have to be gradually increased. The limitation of working only in good weather conditions is still a big topic in autonomous driving. The importance of ODDs to reach high levels of automation was pointed out by Velasco-Hernandez et al. [26]. They emphasize the need of a well defined life cycle development from ODD definition to the final product release. Girdhar et al. [57] gave an overview of the implications of perception systems for autonomous vehicles if the goal is Level 5. Weather is one of the biggest problems for autonomous vehicles today [58], [59].

The handling of more complicated scenarios is another major challenge. More scalable and automatable mapping approaches will have to be developed. In our application, the map was still manually refined. For the mapping of large cities, this is not feasible. In addition, each vehicle will have to be able to detect changes in the map on the go and publish them so that other vehicles are informed. Traffic participants on public roads are not as predictable as opposing race cars during strictly regulated events. Different types of participants exhibit different behaviors: Pedestrians, bicycles, cars, trucks, etc. Driving in urban areas is a different task from driving on the highway and may need to be addressed separately at this point.

The requirement that autonomous vehicles must be able to manage these completely different challenges underlines the importance of data-driven deep learning algorithms, as not every single scenario can be covered with conventional algorithms. The use of deep learning leads to the problem of scalability and a chicken-and-egg problem: to develop such algorithms, we need huge amounts of data that depict a lot of edge cases, which only appear rarely; to collect data and have many autonomous vehicles driving on public roads, exactly these algorithms are required. The only solution to this dilemma is to develop better generalizing algorithms and refine the usage of data. Current models are not able to transfer the learnings from one domain to another, e.g., from simulation to reality, from an American city to a European one, or even from one sensor setup to a different one. Domain adaptation and scalability of deep learning remain key challenges in the field of autonomous driving. However, as explained above, more complex scenarios will require more complex algorithms and these will have to be data-driven. The creation of novel and more versatile publicly available data sets will be the key to quickly overcoming these hurdles.

In summary, autonomous driving has to be solved by solving sub-problems and merging solutions. Therefore, challenges such as the IAC and the AC@CES can be of great benefit to the entire community. However, many key problems remain unsolved and the road to level 5 autonomous vehicles is still long.

VI. CONCLUSION

The experiences and results of the TUM Autonomous Motorsport team throughout the IAC and the AC@CES showed

that perception systems have a huge impact on the performance of autonomous vehicles. System design and analysis are still open research topics. Since system performance is often limited by a conglomerate of causes rather than by specific algorithms, this topic should be of greater importance in future research.

To improve in these areas, well-defined and well-proven workflows are important. Additionally, simulation environments as well as well-thought strategies for testing and validation will have to be established and standardized. This applies not only to race cars but also to road vehicles. When developing novel perception systems, it is a good way to start with the available basic algorithms and first set up a running system. In the next step, more advanced algorithms can be included in a working system to improve and validate the system performance further. We could show that different sensors provide different advantages and disadvantages. For limited ODDs, it might be sufficient to rely heavily on one or two specific modalities. However, if we want to strive toward Level 5 autonomous driving and cover different use cases, applications, and circumstances, sensor fusion is inevitable.

Summarizing our entire findings as concisely as possible, we want to conclude this paper with the most important learnings. These can be taken as advice when starting a robotic project.

A. FOCUS ON THE SYSTEM

Every component should be evaluated for the influence it has on the overall system performance, rather than on the performance of a single algorithm. An early and clear definition of software interfaces allows for easy enhancement of the software stack and integration of novel approaches. A modular overall software pipeline makes it easier to gradually enhance the software stack and individual modules.

B. KEEP IT SIMPLE

This accounts for hardware and software. For a first working prototype, choose a sensor that allows easy development of a first processing pipeline. In our case, this was the LiDAR. Also, for algorithms, it is recommended to start with conventional and understandable solutions. Many of them are already available, for example on *GitHub*[7] and can be quickly integrated. Often, it is worth starting with a *Python* node to get a first working implementation. Later, the software can be extended with advanced approaches.

C. DEVELOP YOUR WORKFLOW

Establishing a well-thought workflow can be a real boost for a robotic project. This includes the whole pipeline, from software development, testing, CI/CD, and simulation, to deployment and validation on the target hardware. Tools such as *Docker* and *GitLab*-integrated tools (e.g. container registry, or build and test automation) allow to set up an advanced workflow even for smaller projects. In addition, many tools for software and data visualization, e.g. *PlotJuggler*, are available open source.

[7]https://github.com/

D. SIMULATION IS KING

Usually, software development has to be started before the final hardware is ready to be used. Simulation is the only way to test a robotic system before it can be evaluated in a real-world application. By doing so, first findings can also be taken into account for hardware development. Moreover, it forces developers to have an early version of a working software stack with defined interfaces. Thus, the transition to the actual application is accelerated significantly. Another point is that simulation allows for conducting a multitude of experiments that could be done in reality.

E. ADVANCED PERCEPTION SYSTEMS

Currently, perception is the limiting factor for autonomous vehicles. Advanced sensors incorporate functionalities that allow them to adapt to different situations. These can be software-definable specifications such as FoV, frame rate, or resolution, but also sensor integrated compute power for a first data pre-processing. However, each sensor modality still faces physical limitations. The only way to develop a perception system capable of Level 5 autonomous driving is to compensate for the disadvantages of single modalities using sensor fusion. The level of data fusion depends on sensor setup and application and will have to be investigated more detailed in future research.

ACKNOWLEDGMENT

This article was written by the Autonomous Motorsport Perception Team, Technical University of Munich (TUM). The authors would like to thank all the other team members and students who contributed to the performance throughout the competitions. Without the whole team, such research would not be possible! They would like to thank the Indy Autonomous Challenge organizers, Juncos Hollinger Racing, and all other participating teams for the countless efforts to make the Indy Autonomous Challenge and all of those experiments with multiple full-scale autonomous racing vehicles possible. Furthermore, this project was made possible with the generous support and contributions of the basic research funds of the Institute of Automotive Technology and the Chair of Automatic Control of TUM as well as several private donors and sponsors.

REFERENCES

[1] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA Grand Challenge: The Great Robot Race*, vol. 36. Berlin, Germany: Springer, 2007.

[2] (2022). *Indy Autonomous Challenge*. [Online]. Available: https://www.indyautonomouschallenge.com/

[3] J. Betz, T. Betz, F. Fent, M. Geisslinger, A. Heilmeier, L. Hermansdorfer, T. Herrmann, S. Huch, P. Karle, M. Lienkamp, B. Lohmann, F. Nobis, L. Ögretmen, M. Rowold, F. Sauerbeck, T. Stahl, R. Trauth, F. Werner, and A. Wischnewski, "TUM autonomous motorsport: An autonomous racing software for the Indy Autonomous Challenge," *J. Field Robot.*, vol. 40, no. 4, pp. 783–809, Jun. 2023.

[4] A. Wischnewski, M. Geisslinger, and J. Betz, "Indy Autonomous Challenge—Autonomous race cars at the handling limits," in *Proc. 12th Int. Munich Chassis Symp.* Cham, Switzerland: Springer, 2022, pp. 163–182.

[5] (2022). *Indy Lights—Dallara IL15*. [Online]. Available: https://www.dallara.it/en/racing/indy%20lights

[6] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3354–3361.

[7] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, D. Ramanan, P. Carr, and J. Hays, "Argoverse 2: Next generation datasets for self-driving perception and forecasting," in *Proc. Neural Inf. Process. Syst.*, 2021, pp. 1–27.

[8] P. Sun, H. Kretzschmar, and X. Dotiwalla, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2446–2454.

[9] S. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. Eng, D. Rus, and M. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, Feb. 2017.

[10] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, and G. Hoffmann, "Stanley: The robot that won the DARPA grand challenge," *J. Field Robot.*, vol. 23, no. 9, pp. 661–692, Sep. 2006.

[11] M. Johnson, B. Shrewsbury, S. Bertrand, T. Wu, D. Duran, M. Floyd, P. Abeles, D. Stephen, N. Mertins, and A. Lesman, "Team IHMC's lessons learned from the DARPA robotics challenge trials," *J. Field Robot.*, vol. 32, no. 2, pp. 192–208, 2015.

[12] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.

[13] N. Naz, M. K. Ehsan, M. R. Amirzada, M. Y. Ali, and M. A. Qureshi, "Intelligence of autonomous vehicles: A concise revisit," *J. Sensors*, vol. 2022, pp. 1–11, Apr. 2022.

[14] Y. Ma, Z. Wang, H. Yang, and L. Yang, "Artificial intelligence applications in the development of autonomous vehicles: A survey," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 2, pp. 315–329, Feb. 2020.

[15] H.-H. Jebamikyous and R. Kashef, "Autonomous Vehicles Perception (AVP) using deep learning: Modeling, assessment, and challenges," *IEEE Access*, vol. 10, pp. 10523–10535, 2022.

[16] A. S. Mohammed, A. Amamou, F. K. Ayevide, S. Kelouwani, K. Agbossou, and N. Zioui, "The perception system of intelligent ground vehicles in all weather conditions: A systematic literature review," *Sensors*, vol. 20, no. 22, p. 6532, 2020.

[17] J. Vargas, S. Alsweiss, O. Toker, R. Razdan, and J. Santos, "An overview of autonomous vehicles sensors and their vulnerability to weather conditions," *Sensors*, vol. 21, no. 16, p. 5397, Aug. 2021.

[18] E. Marti, M. A. de Miguel, F. Garcia, and J. Perez, "A review of sensor technologies for perception in automated driving," *IEEE Intell. Transp. Syst. Mag.*, vol. 11, no. 4, pp. 94–108, 2019.

[19] Z. Wang, Y. Wu, and Q. Niu, "Multi-sensor fusion in automated driving: A survey," *IEEE Access*, vol. 8, pp. 2847–2868, 2020.

[20] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, p. 2140, 2021.

[21] Y. Li and J. Ibanez-Guzman, "LiDAR for autonomous driving: The principles, challenges, and trends for automotive LiDAR and perception systems," *IEEE Signal Process. Mag.*, vol. 37, no. 4, pp. 50–61, Jul. 2020.

[22] J. Van Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow," *Transp. Res. C, Emerg. Technol.*, vol. 89, pp. 384–406, Apr. 2018.

[23] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran, "Deep learning sensor fusion for autonomous vehicle perception and localization: A review," *Sensors*, vol. 20, no. 15, p. 4220, Jul. 2020.

[24] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, p. 648, 2019.

[25] S. Devi, P. Malarvezhi, R. Dayana, and K. Vadivukkarasi, "A comprehensive survey on autonomous driving cars: A perspective view," *Wireless Pers. Commun.*, vol. 114, pp. 2121–2133, May 2020.

[26] G. Velasco-Hernandez, D. J. Yeong, J. Barry, and J. Walsh, "Autonomous driving architectures, perception and data fusion: A review," in *Proc. IEEE 16th Int. Conf. Intell. Comput. Commun. Process. (ICCP)*, Sep. 2020, pp. 315–321.

[27] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proc. 23rd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2018, pp. 751–766.

[28] W. Zong, C. Zhang, Z. Wang, J. Zhu, and Q. Chen, "Architecture design and implementation of an autonomous vehicle," *IEEE Access*, vol. 6, pp. 21956–21970, 2018.

[29] P. Kowalczyk, M. Komorkiewicz, P. Skruch, and M. Szelest, "Efficient characterization method for big automotive datasets used for perception system development and verification," *IEEE Access*, vol. 10, pp. 12629–12643, 2022.

[30] D. Falanga, S. Kim, and D. Scaramuzza, "How fast is too fast? The role of perception latency in high-speed sense and avoid," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1884–1891, Apr. 2019.

[31] A. S. Mueller, J. B. Cicchino, and D. S. Zuby, "What humanlike errors do autonomous vehicles need to avoid to maximize safety?" *J. Saf. Res.*, vol. 75, pp. 310–318, Dec. 2020.

[32] J. Wang, L. Zhang, Y. Huang, and J. Zhao, "Safety of autonomous vehicles," *J. Adv. Transp.*, vol. 2020, pp. 1–13, Oct. 2020.

[33] A. Taeihagh and H. S. M. Lim, "Governing autonomous vehicles: Emerging responses for safety, liability, privacy, cybersecurity, and industry risks," *Transp. Rev.*, vol. 39, no. 1, pp. 103–128, Jul. 2018.

[34] K. Ren, Q. Wang, C. Wang, Z. Qin, and X. Lin, "The security of autonomous driving: Threats, defenses, and future directions," *Proc. IEEE*, vol. 108, no. 2, pp. 357–372, Feb. 2020.

[35] B.-J. Kim and S.-B. Lee, "Safety evaluation of autonomous vehicles for a comparative study of camera image distance information and dynamic characteristics measuring equipment," *IEEE Access*, vol. 10, pp. 18486–18506, 2022.

[36] M. Tlig, M. Machin, R. Kerneis, E. Arbaretier, L. Zhao, F. Meurville, and J. Van Frank, "Autonomous driving system: Model based safety analysis," in *Proc. 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2018, pp. 2–5.

[37] S. Almeaibed, S. Al-Rubaye, A. Tsourdos, and N. P. Avdelidis, "Digital twin analysis to promote safety and security in autonomous vehicles," *IEEE Commun. Standards Mag.*, vol. 5, no. 1, pp. 40–46, Mar. 2021.

[38] J. R. McBride, J. C. Ivan, D. S. Rhode, J. D. Rupp, M. Y. Rupp, J. D. Higgins, D. D. Turner, and R. M. Eustice, "A perspective on emerging automotive safety applications, derived from lessons learned through participation in the DARPA grand challenges," *J. Field Robot.*, vol. 25, no. 10, pp. 808–840, Oct. 2008.

[39] S. Nekkah, J. Janus, M. Boxheimer, L. Ohnemus, S. Hirsch, B. Schmidt, Y. Liu, D. Borbély, F. Keck, K. Bachmann, and L. Bleszynski, "The autonomous racing software stack of the KIT19d," 2020, *arXiv:2010.02828*.

[40] A. Alvarez, N. Denner, Z. Feng, D. Fischer, Y. Gao, L. Harsch, S. Herz, N. Le Large, N. Bach, C. Rosero, S. Schaefer, A. Terletskiy, L. Wahl, S. Wang, J. Yakupova, and Y. Haocen, "The software stack that won the formula student driverless competition," in *Proc. 2nd Workshop Opportunities Challenges Auto. Racing*, 2022, pp. 1–6.

[41] S. Thrun and M. Montemerlo, "The graph SLAM algorithm with applications to large-scale mapping of urban structures," *Int. J. Robot. Res.*, vol. 25, nos. 5–6, pp. 403–429, 2006.

[42] J. Kabzan, M. I. Valls, V. J. F. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, and A. Bühler, "AMZ driverless: The full autonomous racing system," *J. Field Robot.*, vol. 37, no. 7, pp. 1267–1294, 2020.

[43] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using FastSLAM," in *Proc. IEEE Int. Conf. Robot. Autom.*, Sep. 2003, pp. 1985–1991.

[44] C. Jung, A. Finazzi, H. Seong, D. Lee, S. Lee, B. Kim, G. Gang, S. Han, and D. H. Shim, "An autonomous system for head-to-head race: Design, implementation and analysis; Team KAIST at the Indy Autonomous Challenge," 2023, *arXiv:2303.09463*.

[45] T. Betz, P. Karle, F. Werner, and J. Betz, "An analysis of software latency for a high-speed autonomous race car—A case study in the Indy Autonomous Challenge," *SAE Int. J. Connected Automated Vehicles*, vol. 6, no. 3, pp. 1–11, Feb. 2023.

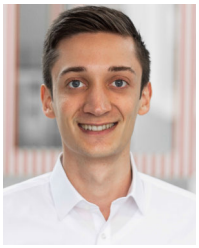
[46] F. Sauerbeck, L. Baierlein, J. Betz, and M. Lienkamp, "A combined LiDAR-camera localization for autonomous race cars," *SAE Int. J. Connected Automated Vehicles*, vol. 5, no. 1, pp. 61–71, Jan. 2022.

[47] P. Fua and K. Lis, "Comparing Python, go, and C++ on the N-queens problem," 2020, *arXiv:2001.02491*.

[48] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 779–788.
[49] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 770–779.
[50] V. M. Raju, V. Gupta, and S. Lomate, "Performance of open autonomous vehicle platforms: Autoware and Apollo," in *Proc. IEEE 5th Int. Conf. Converg. Technol. (ICT)*, Mar. 2019, pp. 1–5.
[51] S. Huch, L. Scalerandi, E. Rivera, and M. Lienkamp, "Quantifying the LiDAR sim-to-real domain shift: A detailed investigation using object detectors and analyzing point clouds at target-level," *IEEE Trans. Intell. Vehicles*, early access, Mar. 2, 2023, doi: 10.1109/TIV.2023.3251650.
[52] P. Karle, F. Fent, S. Huch, F. Sauerbeck, and M. Lienkamp, "Multi-modal sensor fusion and object tracking for autonomous racing," *IEEE Trans. Intell. Veh.*, pp. 1–13, 2023.
[53] N. Vodisch, O. Unal, K. Li, L. Van Gool, and D. Dai, "End-to-end optimization of LiDAR beam configuration for 3D object detection and localization," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 2242–2249, Apr. 2022.
[54] G. Volk, J. Gamerdinger, A. V. Bernuth, and O. Bringmann, "A comprehensive safety metric to evaluate perception in autonomous systems," in *Proc. IEEE 23rd Int. Conf. Intell. Transp. Syst. (ITSC)*, Sep. 2020, pp. 1–8.
[55] J. Philion, A. Kar, and S. Fidler, "Learning to evaluate perception models using planner-centric metrics," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 14055–14064.
[56] R. Philipp, J. Rehbein, F. Grun, L. Hartjen, Z. Zhu, F. Schuldt, and F. Howar, "Systematization of relevant road users for the evaluation of autonomous vehicle perception," in *Proc. IEEE Int. Syst. Conf. (SysCon)*, Apr. 2022, pp. 1–8.
[57] M. Girdhar, Y. You, T.-J. Song, S. Ghosh, and J. Hong, "Post-accident cyberattack event analysis for connected and automated vehicles," *IEEE Access*, vol. 10, pp. 83176–83194, 2022.
[58] S. Zang, M. Ding, D. Smith, P. Tyler, T. Rakotoarivelo, and M. A. Kaafar, "The impact of adverse weather conditions on autonomous vehicles: How rain, snow, fog, and hail affect the performance of a self-driving car," *IEEE Veh. Technol. Mag.*, vol. 14, no. 2, pp. 103–111, Jun. 2019.
[59] R. Heinzler, P. Schindler, J. Seekircher, W. Ritter, and W. Stork, "Weather influence and classification with automotive LiDAR sensors," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2019, pp. 1527–1534.

FLORIAN SAUERBECK (Member, IEEE) received the B.Sc. degree from the University of Erlangen–Nuremberg, in 2016, and the M.Sc. degree in electrical engineering from the Technical University of Munich (TUM), Munich, Germany, in 2020, where he is currently pursuing the Ph.D. degree in mechanical engineering with the Institute of Automotive Technology. His research interests include 3D perception, localization, and mapping, with a focus on real-world applications.

SEBASTIAN HUCH received the B.Eng. degree from Baden–Wuerttemberg Cooperative State University (DHBW), Stuttgart, Germany, in 2016, and the M.Sc. degree from the Technical University of Darmstadt, Germany, in 2018. He is currently pursuing the Ph.D. degree in mechanical engineering with the Institute of Automotive Technology, Technical University of Munich (TUM), Germany. His research interests include LiDAR simulation, LiDAR perception, and LiDAR domain adaptation for autonomous driving.



FELIX FENT received the B.Sc. and M.Sc. degrees from the Technical University of Munich (TUM), Munich, Germany, in 2018 and 2020, respectively, where he is currently pursuing the Ph.D. degree in mechanical engineering with the Institute of Automotive Technology. His research interests include radar-based perception, sensor fusion, and multi-modal object detection approaches, with a focus on real-world applications.

PHILLIP KARLE received the B.Sc. and M.Sc. degrees from the Technical University of Munich (TUM), Munich, Germany, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree in mechanical engineering with the Institute of Automotive Technology. His research interests include multi-object tracking, scenario understanding, motion prediction, and related applications for autonomous driving, with a focus on real-world applications.

DOMINIK KULMER received the B.Sc. and M.Sc. degrees from the Technical University of Munich (TUM), Munich, Germany, in 2019 and 2022, respectively, where he is currently pursuing the Ph.D. degree in mechanical engineering with the Institute of Automotive Technology. His research interests include sensor fusion, localization, and mapping, with a focus on real-world applications.

JOHANNES BETZ (Member, IEEE) received the B.Eng. degree from the University of Applied Science Coburg, in 2011, the M.Sc. degree from the University of Bayreuth, in 2012, and the Ph.D. and M.A. degrees in philosophy from the Technical University of Munich (TUM), in 2019 and 2021, respectively. He is currently an Assistant Professor with the Department of Mobility Systems Engineering, TUM. He is one of the founders of the Autonomous Motorsport Team, TUM. His research interests include developing adaptive dynamic path planning and control algorithms, decision-making algorithms that work under high uncertainty in multi-agent environments, and validating the algorithms on real-world robotic systems.