# A Self-Driving Car Architecture in ROS2

Michael Reke, Daniel Peter, Joschua Schulte-Tigges,
Stefan Schiffer and Alexander Ferrein
Mobile Autonomous Systems & Cognitive Robotics Institute
FH Aachen University of Applied Sciences,
Aachen, Germany
Email: reke@fh-aachen.de, peter@fh-aachen.de,
joschua.schulte-tigges@alumni.fh-aachen.de,
s.schiffer@fh-aachen.de, ferrein@fh-aachen.de

Thomas Walter and Dominik Matheis
Electronics System Development,
Advanced Safety Control
Hyundai Motor Europe Technical Center GmbH,
Ruesselsheim, Germany
Email: twalter@hyundai-europe.com,
dmatheis@hyundai-europe.com

***Abstract*—In this paper we report on an architecture for a self-driving car that is based on ROS2. Self-driving cars have to take decisions based on their sensory input in real-time, providing high reliability with a strong demand in functional safety. In principle, self-driving cars are robots. However, typical robot software, in general, and the previous version of the Robot Operating System (ROS), in particular, does not always meet these requirements. With the successor ROS2 the situation has changed and it might be considered as a solution for automated and autonomous driving. Existing robotic software based on ROS was not ready for safety critical applications like self-driving cars.**

**We propose an architecture for using ROS2 for a self-driving car that enables safe and reliable real-time behaviour, but keeping the advantages of ROS such as a distributed architecture and standardised message types. First experiments with an automated real passenger car at lower and higher speed-levels show that our approach seems feasible for autonomous driving under the necessary real-time conditions.**

***Index Terms*—Self-driving car, autonomous driving, architecture, robot operating system, ROS, ROS2, LKAS, V2X**

## I. Introduction

Compared to other robots, self-driving cars have different requirements in terms of real-time behaviour and calculation speed. They have to master a wide range of manoeuvre from precise path-finding in parking situations to high-speed driving on highways. Additionally, the deployed computer systems have to perceive a lot of traffic situations correctly to take the right driving decisions. Self-driving cars have to accomplish all these tasks with a very high demand on functional safety because of their direct interaction with people as drivers, passengers, pedestrians, and other road users. The widely used Robot Operating System (ROS) [1] cannot guarantee such a high reliability and real-time performance as required for automated driving applications covering all the mentioned situations.

However, ROS comes with a lot of features, which are beneficial when developing a general computer system for automated driving. The lively ROS community has many people contributing functionalities, e.g. hardware drivers for numerous sensors. A lot of great tools such as RViz or Gazebo are available for an efficient development process and ROS is open source, which gives a high confidence to the used software modules.

The second generation of Robot Operation System ROS2 provides the needed reliability and real-time performance while most of the advantages of ROS1 are still available (e.g. cf. [2]). Therefore, we decided to develop a software architecture for a self-driving car making use of ROS2.

Besides the requirements of real-time and calculation performance, the proposed architecture design should be largely vehicle independent. Having a control interface for steering, acceleration and braking is common to all automated vehicles, but the specification is very different. To deal with those differences we introduce a wrapper node, which we call *ROS2Car*. Additionally, we define nodes (computing entities in ROS) for all of the standard tasks in automated driving. We will give the details below. The architecture is evaluated in different scenarios to verify its usability and variability. We report our first evaluation results in this paper.

The rest of the paper is organised as follows. In the next section, we give an overview of the related work in autonomous driving architectures. In Section III we present our approach to a car-independent autonomous driving architecture for ROS2 focusing on the different aspects of such an architecture. We start with the requirements and then present our ideas how tasks such as mission and manoeuvre planning, vehicle control or localisation and perception could be dealt with inside the architecture. In Section IV, we present our first evaluation results with our autonomous driving platform, the Kia Niro, which already was driving up to speeds of about 80 km/h with our ROS2 software. We conclude in Section V.

## II. Related Work

The first research into self-driving cars were undertaken in the late 1980s already with the PROMETHEUS project (e.g. [3]). Within this project it was shown that it's possible to drive a motorway controlled by a computer program, though still under restricted lighting and weather conditions. Like today, the prevalent sensors were camera systems. In the late 2000s the topic became a hot research topic again with the DARPA Grand and Urban Challenges (e.g. [4], [5]). In 2009,

978-1-7281-4162-6/20/$31.00 ©2020 IEEE

the Tartan R acing team won the Urban Challenge showing that it is possible to drive autonomously in an urban environment. The DARPA challenges also been led to the GoogleX project, which gave the whole field quite a push. In [6], a brief overview of the self-driving car activities on the different continents is given.

During these works several software architectures have been described. Following some comparison works of Berger et al. [7]–[10] these software architectures for self-driving cars are mainly subdivided into modules, which break down the overall task into different functionalities such as localisation, perception and vehicle control. As described below, we also stick to this principles and our software architecture is also divided in a similar way.

Today, there are several projects for autonomous driving software, which use their own proprietary software frameworks. The Chinese Apollo project [11] started using ROS, but they changed the original ROS system mainly wrt. communication functionalities. Apollo does not use ROS2. Another proprietary platform is Polysync Core, which provides a bridge to ROS, but does not support ROS2. Additionally, there is the NVIDIA Drive Platform [12], which is primarily based on the NVIDIA Pegasus hardware, which we also use but on the software side, there is no link to ROS.

The Japanese initiative *The Autoware Foundation* introduced *Autoware.ai* [13], which is based on ROS1 [14]. This original project will not be further developed, but is currently transferred to ROS2 under the name *Autoware.auto* [15]. The functional software architecture of *Autoware* is similar to our proposed architecture, but the *Autoware* project focuses currently only on different parking scenarios as a first use case. This leads to some particular design decisions. For instance, they primarily concentrate on localisation with Lidar information comparing to HD maps. In parking scenarios this is useful, but it is very computation intensive [16]. This is not well suited for driving at higher speeds. Additionally, they focus on sensor fusion at point cloud level, which is also mainly useful in parking scenarios. In comparison to Autoware, with our approach we concentrate on a more general sensor fusion at an object level in order to generate global cost maps or occupancy grids. Currently, a research group led by Kato, one of the Autoware founders, is working on a promising new distributed computing system based on GPUs and FPGA [17] to extend the overall computing power and to divide the whole task into sub-tasks. As described in the next section, we also investigated how data for certain sub-tasks could be processed independent from the main processors, for instance, by a MobilEye camera system that comes with its own processing power.

On the commercial side, there are also some software frameworks, such as AUTOSAR [18] and AUTOSAR Adaptive or Elektrobits ADTF, which are used in autonomous driving projects, but none of them has such a huge contribution community like ROS.

The design of software architecture proposed here was also developed with respect to experiences from some European research projects. In the light of road safety and traffic management the European Union established the Horizon 2020 research program, where research & development activities are carried out in the field of automated and cooperative automated driving. HMETC (Hyundai Motor Europe Technical Center) is actively participating in dedicated research projects on cooperated automated driving, where vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication based on the ITS-G5 WiFi standard is used to coordinate the traffic flow in order to increase traffic safety & efficiency in urban environments. In the MAVEN (Managing Automated Vehicles Network) project the focus is urban environment traffic management. This includes infrastructure advises for an automated vehicle. Within the research project TransAID (Transition Areas For Infrastructure-Assisted Driving) the focus is on transitional areas where an automated driving vehicle, for instance, has to hand over control to the driver (Transition of Control—ToC) due to a limited level of automation in the vehicle that cannot handle an upcoming traffic situation (e.g. road works, weather conditions, etc.).

## III. Architecture

In this section, we introduce our software architecture. We start with a requirements analysis before we go into the details of the implementation and the functional subtasks.

### *A. Requirements*

Following the taxonomy of Lichter [19] the requirements for a software architecture can be subdivided into usability related (functional) and maintainability related (non-functional) requirements. Functional requirements describe the requested performance of the system and represent the user oriented view. Non-functional requirements describe the key points of the developer oriented view.

Usability (functional) oriented requirements:

- **Real-Time performance:** Automated Driving is basically a control task, where the temporal boundaries for the inner control loop are given by the lateral and longitudinal dynamic of the vehicle. The period of this inner vehicle dynamic control loop is with around 10 ms normally the shortest within the complete cascaded control structure. To ensure a stable control result, the jitter has to be in the range of a few microseconds.
  Additionally, one of the basic requirements for any driving automation system is to be better in the sense of faults than any human driver. However, this is only possible if the surrounding traffic including all obstacles are perceived faster and with a higher confidence than any human could do. While a human needs at least 300 ms to perceive a new traffic situation and to react accordingly [20], this gives a maximum boundary of about 100 ms for detecting objects and a proper reaction by the automation system.
- **Reliability:** Not only the life of the vehicle passengers is affected by an driving automation system, but also the life of any other road user. In terms of functional safety the

hazard and risk analysis can not give any other demand than ASIL D in accordance to ISO 26262 [21]. Only the list of safety goals has to be discussed for the particular vehicle. While ISO 26262 only focuses on the fail-safe behavior of systems, the upcoming ISO/PAS 21448 [22] will also give a frame for the *Safety of the intended functionality (SOTIF)*.

- **Adaptability and Interoperability:** The software architecture should have the claim to be used in different cars and in different traffic scenarios, e.g. use in different countries or the use in restricted areas, where piloted driving is allowed, recommended or needed. To reach this requirement, the software has to provide mechanisms to integrate the corresponding communication to infrastructure elements. Additionally the communication with other vehicles and road users has to be integrated to consider those behavior in the vehicle control. Additionally the software has to distinguish between various driving situations and has to switch between the needed driving manoeuvres.

Maintainability (non-functional) oriented requirements:

- **Flexibility and Variability:** The software architecture should be used for different vehicles in different environments. The architecture will be mainly used for passenger cars on public roads, but it should be possible to use the architecture e.g. for construction vehicles in off-road situations. To reach this the architecture requires abstraction elements for different vehicles and environments.
- **Re-Usability and Extensibility:** The software architecture is intended to be used for different current vehicles, but it should be possible to integrate also future developments. Especially in the field of sensor systems there are a lot of changes, improvements and extensions. To be prepared for those developments the software architecture should provide an independence of the different sensor systems and software modules.
- **Open Community:** Automated and even more autonomous driving is a huge challenge, which cannot be mastered by one isolated development group. Hence, a vital and open community is essential to work and to contribute successful in this field.

## *B. Overview and Implementation*

With respect to the functional and non-functional requirements the second generation of the robot operating system (ROS2) is an ideal framework, because on the one hand ROS in general is a modular framework and on the other hand it provides real-time performance since ROS2.

As mentioned in Section III-A, it is important to meet the temporal boundaries given by the vehicle-control-loop. Therefore, it is necessary to strictly use the real-time capabilities of ROS2, which means to implement all ROS nodes following the real-time rules [23]:

- Since kernel version 2.6.11 Linux supports real-time in general with the official kernel patch PREEMPT_RT.
- Dynamic memory management is not allowed in real-time applications, so all objects in all nodes have to be statically allocated.
- The complete communication within the ROS2 network has to be realised by using the real-time middleware, which follows the *Data Distribution Service* (DDS) standard defined by the Object Management Group (OMG).

Based on these implementation guidelines, we designed the automated driving software architecture as a distributed and modular system, consisting of several ROS2 nodes. The distribution of nodes is oriented first to the integrated devices such as sensor systems (e.g. having a separate node for each used sensor system). Second, the distribution orients to the different tasks in autonomous driving. Each task (e.g. path control) is encapsulated in a separate node. Following these architecture design rules, the requirements in variability and re-usability are fulfilled.

Figure 1 shows a functional plan of the architecture, where each white box represents a ROS2 node.

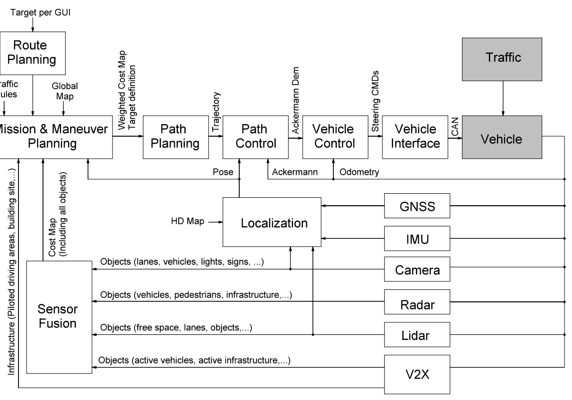
Target per GUI
Route Planning
Traffic Rules
Global Map
Mission & Maneuver Planning
Weighted Cost Map
Target definition
Path Planning
Trajectory
Path Control
Ackermann Dem
Vehicle Control
Steering CMDs
Vehicle Interface
CAN
Traffic
Vehicle
Pose
Ackermann
Odometry
Infrastructure (Piloted driving areas, building site,...)
Cost Map (Including all objects)
HD Map
Localization
GNSS
IMU
Camera
Radar
Lidar
V2X
Sensor Fusion
Objects (lanes, vehicles, lights, signs, ...)
Objects (vehicles, pedestrians, infrastructure,...)
Objects (free space, lanes, objects,...)
Objects (active vehicles, active infrastructure,...)

Fig. 1. Function-plan of the self-driving car architecture

The vehicle interface is abstracted within a node, which is individual to every supported vehicle. Every sensor system (GNSS, IMU, camera, radar, lidar, V2X) is connected to the ROS2 system via a corresponding node, which also includes the necessary intelligence for object detection. Based on the information of all sensor systems, a cost map is generated by a sensor fusion node. To follow a globally generated route the detailed missions and manoeuvres are planned under recognition of the cost map and external areal information, which are gathered by V2X connections. According to the selected manoeuvre a local trajectory is generated, which the vehicle has to follow. This trajectory can be seen as a demand value for a closed loop path controller with the current position of the vehicle as the actual value. The vehicle position is determined by a GNSS sensor, which is corrected by an inertial measurement unit (IMU) and additional position information of the camera and lidar systems, which are partly matched to a HD map.

## *C. Mission and Manoeuvre Planning*

The complete task of autonomous driving has to be subdivided into various manoeuvres to cover all traffic situations.

The granularity of these manoeuvres is different depending on the situation. For example, parking can be divided in the manoeuvres *looking for parking space, drive vehicle in the parking space* and *secure vehicle*, while hourly driving on a highway is covered by only one highway manoeuvre.

The inputs for the mission and manoeuvre planing are on the one hand the output route from a route planning system and a global map. On the other hand, another input is the current traffic and environmental situation of the vehicle as a local cost map including all relevant objects. Based on the traffic rules and the information, which are sent by infrastructure elements via V2X the next manoeuvre is chosen and planned.

The output of the mission and manoeuvre planning is a weighted cost map with detailed information of the roads and the environment, which surrounds the vehicle, and a target definition.

### *D. Vehicle Control*

Depending on the weighted cost map and the target definition, a trajectory is generated by the path planner. For this, the weighted cost map consists of all areas, where the vehicle is allowed to drive and it also consists of all obstacles, which have to be avoided. The calculation of this trajectory has to consider the geometric dimensions and the particular lateral and longitudinal dynamics of the vehicle.

The trajectory is used as the demand value for the path controller, which uses the position and current dynamic of the vehicle as actual value. As output the path controller requests an acceleration, a velocity and a steering angle. Depending on the vehicle, but also depending on the current manoeuvre the path controller uses different control algorithms to follow the trajectory. This is necessary, because the calculation power of the deployed computer system is always fixed, but the speed of the vehicle and the demands in terms of steering accuracy are different, while performing different manoeuvres (e.g. low speed and high accuracy for parking and high speed, but lower accuracy for driving on highway). All control strategies have to recognise the kinematics of the actual vehicle, which is an Ackermann kinematic in most times.

The vehicle controller realises a controlled constant speed using the known longitudinal vehicle dynamics as a feed-forward control part. For the lateral vehicle control the dynamic parts of the vehicles steering system are compensated.

The output of the vehicle control are messages, which are translated to e.g. CAN messages for the particular vehicle control interface.

By using this complex modularised vehicle control system the requirement of flexibility and variability is fulfilled, because for every type of vehicle only some of the parts have to be exchanged.

### *E. Localisation and Perception*

The localisation and the perception is based on sensor systems, which can vary very much between different vehicles. To deal with these variations the software architecture provides a separate node for every sensor systems.

A precise localisation is essential for an accurate vehicle control behavior. Within the software architecture it is abstracted in one node, for which the GNSS and the IMU sensors deliver the main input signals. Using a Kalman filter, an absolute position of the vehicle in geographical coordinates is calculated. Because GNSS is an often uncertain or unavailable signal e.g. in tunnels or under bridges, it has to be corrected by IMU sensor data. But if the satellite link is permanently disturbed, even this correction cannot deliver a useful position. To compensate this, it is essential to facilitate new localisation methods e.g. by high density maps, with additional position information of characteristic objects such as buildings, traffic lights and the like. These objects will be recognised by sensor systems such as a camera or Lidar, and allow for an additional localisation information by a comparison with the HD map. The output of the localisation node is always the current position of the vehicle.

To enable autonomous driving, lots of objects have to be perceived by the vehicles with a precise position and geometric expansion measurement with an acceptable error of only a few centimeters. Additionally, all measurements and data processing have to be carried out in real-time to enable a stable control behaviour. A highly reliable perception of all relevant objects is essential for a safe driving behaviour. This is only possible by fusing the object lists of all used sensor systems to calculate a local cost map, which consists of all objects as needed by the manoeuvre planning as described in Subsection III-C.

## IV. Evaluation

### *A. Current State of Work*

The overall goal of our work is to develop a general software architecture for autonomous driving vehicle. We described the software architecture on a coarse level, which have to be refined and finally coded node by node. Up to now, we implemented a first set of nodes for two KIA Niros (Fig. 2) as test vehicles, which we equipped with slightly the same hardware. Our first goal was to evaluate the core technical challenges like real-time behavior, capability of sensor integration and capability of vehicle control at different speed levels.

Fig. 2. One of the KIA Niros, which has been automated at FH Aachen.

### *B. Vehicle Setup*

Figure 3 shows the hardware setup of the test vehicles. The core ROS2 software is running on an i7 Intel NUC PC, which is connected to the vehicle via a CAN (Control Area Network) interface. Also the GNSS, IMU and radar

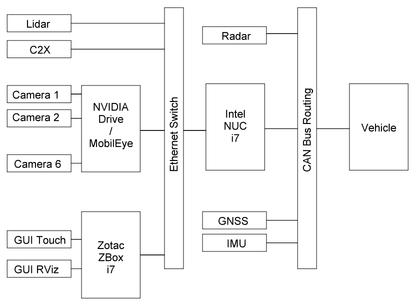

Fig. 3. Hardware Setup of the KIA Niro

sensors are connected via CAN to the ROS2 system. Some of the base odometry signals have to be exchanged between the sensor systems. The radar sensor e.g. needs the vehicles speed for a correct object classification and to calculate the correct relative speed of the different objects. To ensure this communication a CAN bus router was deployed, which allows to route signals from one message on a particular bus to another message e.g. with different ID or scaling on another bus.
An ethernet switch connects the control PC to the lidar sensor and the V2X communication box. To control and observe the complete system via GUIs an additional i7 PC (ZBox) is connected via ethernet. Especially the camera data processing with algorithms for object recognition and classification needs a lot of calculation power. This is realised by a dedicated NVIDIA Drive hardware with GPUs, where all cameras are connected to, or, alternatively, by an integrated MobilEye camera system. The data exchange of the camera system with the remaining network is also realised via ethernet.

### C. Real-time Performance

The real-time behaviour of the system is the most important criterion for security and stability. Therefore ROS2 with its real-time capabilites was chosen. We implemented a lane keeping assist system (LKAS) scenario and tested it under different configurations using recordings of the MobilEye CAN as input data. The CAN measurement contains data about lanes, their position and curvature. A control PC records the CAN data via a CAN to USB interface. In the first step, camera data are published cyclically with 33 Hz by the camera node and subscribed and processed by the node vehicle control node which in turn calls the LKAS algorithm. From the lane data, the LKAS algorithm determines the vehicle's steering angle and publishes it to the ROS2CAR node. We measured the throughput with the software tool Vector CANoe, expecting a difference time of 30 ms between the messages. Since the standard Linux kernel is not designed for real-time operations, a jitter is to be expected. We tested the jitter with different kernel setups, shown in Figure 4.

Figure 4 and Table I show the results of the four tests over time. For the different test cases, the same 30 s timeframe (showing a highway situation) was selected to ensure

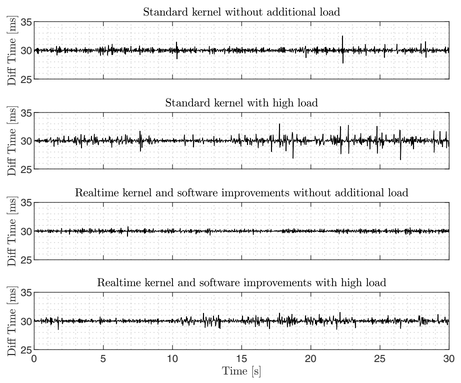

Fig. 4. Jitter measurement in four different system states

reproducibility. The first two graphs show the time intervals of the incoming CAN message on the standard Linux kernel without and with additional load on the system. The system reacts on high load with a worse performance and therefore it is not deterministic and not safe. After the optimisation of the ROS2 nodes regarding the CAN interface and the use of the real-time kernel for Linux, the delays in the third graph became considerably better. Under load, however, the fourth graph shows that the jitter is increasing again. Table I lists the average, minimum and maximum values of the measurements.

TABLE I
ANALYSIS OF THE JITTER MEASUREMENT

| | Mean [ms] | Min [ms] | Max [ms] |
|---|---|---|---|
| St Kernel w/o high Load | 29.9995 | 27.7580 | 32.5350 |
| St Kernel w/ high Load | 29.9988 | 26.6090 | 33.0010 |
| Rt Kernel w/o high Load | 29.9996 | 29.0900 | 30.7890 |
| Rt Kernel w/ high Load | 29.9991 | 28.4890 | 31.5090 |

As expected the measurements show a much better real-time performance of the system when using the real-time kernel, but it also shows that just using ROS2 is not enough (which confirms the results of [2]). Creating a distributed system for autonomous driving based on ROS2, which can really guarantee real-time boundaries, is only possible by carefully paying attention to the mentioned Linux scheduling guidelines, thread priorities and memory management. In our next steps we have to further improve and extend our system with respect to the hard real-time conditions, which should be far below 1 ms.

### D. Slow-speed and High-speed Evaluation

To evaluate our system in different driving scenarios we successfully used our system at different speed-levels.

Vehicles have to be moved at slow speed for, say, parking in an underground parking garage. In those situations the GNSS sensor often gives no suitable information and the localisation has to be realised by other sensor systems such as Lidar. These sensor systems measure point clouds, which may require a longer processing time. But due to the slower speed, also more

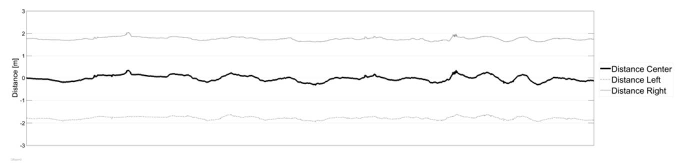

Fig. 5. Car position in between left and right lane during LKAS testing

time is available and more accurate localisation methods can be used, such as Monte Carlo localisation.

To evaluate the software architecture for those situations we used a curb-side bounded test road of about 300 m. We tried to detect the curb-sides via laser beams of the lidar as single sensor and gave us the task to keep the vehicle in the middle of road. Only the vehicle-internal measurement of the speed is additionally used for localisation. Our tests were successful and it was possible to keep the vehicle on track, without relying on any GNSS data. When driving on highways, cars are moving at higher speeds of 60-200 km/h. To evaluate the software architecture also at this speed level we tested our KIA Niro at an old airport near Darmstadt, Germany, with an LKAS algorithm. The input sensor for this setup was a MobileEye. It has a built-in lane detection where the lane information are transmitted via CAN. In our software implementation, the camera node gathers this information from CAN and publishes this periodically, so that the vehicle control node (LKAS) can work in a fixed time-step. The controller periodically calculates the error due to the inputs given by the MobileEye. The lateral vehicle controller uses a PID controller and publishes the steering angle. Our ROS2Car node transmits the steering command to the Vehicle Interface (see Figure 1). The test results are presented in Figure 5. It shows the position of the car continuously within the lane markings. The goal of the LKAS is not to keep the car in the exact center of the lane but rather to keep the car in between the lane markings. Additionally, the architecture allows us to evaluate the ToC-Scenario described in Section II. The vehicle therefore has to be capable to make a lane-change and stop afterwards. For this scenario, a distance controller was developed to keep the vehicle at a constant speed. The distance measurements from the MobileEye and the desired speed are parameters of the vehicle control node.

## V. CONCLUSIONS

In this paper a new software architecture for autonomous driving based on ROS2 is presented and discussed. First we defined requirements for a software architecture and we proposed a possible safe, reliable and real-time-capable solution. The architecture was successfully evaluated in terms of real-time and usability for different automated driving scenarios. It is shown, that a good real-time behavior of a ROS2 system is possible, but only if in the implementation all the special real-time coding demands are considered.
In future work we will improve our system in terms of even better real-time performance and we want to evaluate it on different vehicle platforms. Additionally we will extent our system by supporting more sensor systems and more different driving manoeuvres.

## REFERENCES

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.
[2] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *Proceedings of the 13th International Conference on Embedded Software*. ACM, 2016, p. 5.
[3] E. D. Dickmanns, "The development of machine vision for road vehicles in the last decade," in *Intelligent Vehicle Symposium, 2002. IEEE*, vol. 1. IEEE, 2002, pp. 268–281.
[4] S. Thrun, "Toward robotic cars," *Communications of the ACM*, vol. 53, no. 4, pp. 99–106, 2010.
[5] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*. springer, 2009, vol. 56.
[6] M. Daily, S. Medasani, R. Behringer, and M. Trivedi, "Self-driving cars," *Computer*, vol. 50, no. 12, pp. 18–23, 2017.
[7] C. Berger and M. Dukaczewski, "Comparison of architectural design decisions for resource-constrained self-driving cars-a multiple case-study," *Informatik 2014*, 2014.
[8] S. Behere and M. Toerngren, "A functional reference architecture for autonomous driving," *Information and Software Technology*, vol. 73, pp. 136–150, 2016.
[9] S. Behere and M. Torngren, "A functional architecture for autonomous driving," in *2015 First International Workshop on Automotive Software Architecture (WASA)*. IEEE, 2015, pp. 3–10.
[10] C. Berger, J. Hansson, *et al.*, "Cots-architecture with a real-time os for a self-driving miniature vehicle," 2013.
[11] Baidu Apollo consortium, "Apollo Auto, An open autonomous driving platform." [Online]. Available: https://github.com/ApolloAuto/apollo
[12] NVIDIA Corporation, "NVIDIA DRIVE - Autonomous Vehicle Development Platforms." [Online]. Available: https://developer.nvidia.com/drive
[13] The Autoware Foundation, "Autoware.ai." [Online]. Available: https://www.autoware.ai/
[14] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.
[15] The Autoware Foundation, "Autoware.auto." [Online]. Available: https://www.autoware.auto/
[16] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.
[17] H. Chishiro, K. Suito, T. Ito, S. Maeda, T. Azumi, K. Funaoka, and S. Kato, "Towards Heterogeneous Computing Platforms for Autonomous Driving," in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE, 2019, pp. 1–8.
[18] S. Fuerst, J. Moessinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkaemper, G. Kinkelin, K. Nishikawa, and K. Lange, "Autosar–a worldwide standard is on the road," in *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, vol. 62, 2009, p. 5.
[19] Jochen Ludewig and Horst Lichter, *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. Dpunkt.Verlag GmbH, 2007.
[20] M. Mitschke and H. Wallentowitz, "Bremsung," in *Dynamik der Kraftfahrzeuge*. Springer, Jan. 2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-658-05068-9_9
[21] *ISO 26262: Road vehicles – Functional safety*, International Organization for Standardization Std., 2011.
[22] *ISO/PAS 21448:2019 Road Vehicles - Safety Of The Intended Functionality*, International Organization for Standardization Std., 2019.
[23] J. Kay, "Proposal for Implementation of Real-time Systems in ROS 2." [Online]. Available: https://design.ros2.org/articles/realtime_proposal.html