2019 IEEE Intelligent Vehicles Symposium (IV)
Paris, France. June 9-12, 2019

Bridging the Gap between Open Source Software and Vehicle Hardware for Autonomous Driving

Tobias Kessler[1], Julian Bernhard[1], Martin Buechel[1], Klemens Esterle[1], Patrick Hart[1], Daniel Malovetz[1], Michael Truong Le[1], Frederik Diehl[1], Thomas Brunner[1] and Alois Knoll[2]

***Abstract*— Although many research vehicle platforms for autonomous driving have been built in the past, hardware design, source code and lessons learned have not been made available for the next generation of demonstrators. This raises the efforts for the research community to contribute results based on real-world evaluations as engineering knowledge of building and maintaining a research vehicle is lost. In this paper, we deliver an analysis of our approach to transferring an open source driving stack to a research vehicle.**

We put the hardware and software setup in context to other demonstrators and explain the criteria that led to our chosen hardware and software design. Specifically, we discuss the mapping of the Apollo driving stack to the system layout of our research vehicle, *fortuna*, including communication with the actuators by a controller running on a real-time hardware platform and the integration of the sensor setup. With our collection of the lessons learned, we encourage a faster setup of such systems by other research groups in the future.

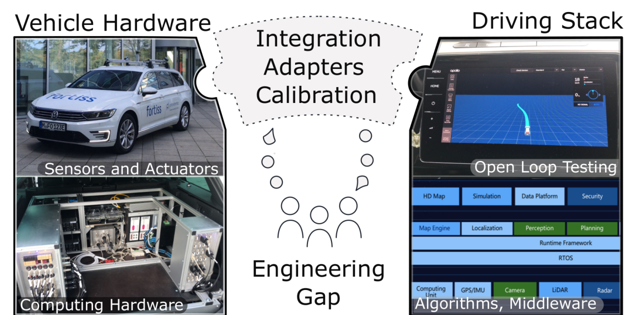

Fig. 1: Autonomous driving software has to be deployed to vehicle hardware. Know-how, code, and pitfalls shall be mirrored back to hardware and software design and the research community.

I. Introduction

The participants [1–4] of the Defense Advanced Research Projects Agency (DARPA) Urban Challenge demonstrated that full-sized research vehicles foster identification of essential research problems while testing and verifying algorithms in real-world scenarios. To fully leverage the potential of the research community in the field of autonomous driving, academic institutions should not only rely on OEMs for real-world experiments but evaluate algorithms on their own research platforms.

Despite the rich history of past research vehicles starting in the early 90s (see Section II-B), the source code has rarely been made open source. Apart from a collaborative approach by Levinson *et al.* [5] to summarize lessons learned on the algorithm side, only sparse knowledge has flown back to the community regarding how to build up an architecture including the inevitable pitfalls one will face. The fully autonomous Bertha Benz Memorial drive, for example, has been a celebrated milestone with significant scientific contributions [6]. However, as we outline in Table I, the hardware configuration, the sensor set, and others have not been published to protect the intellectual property of the OEM. The research community thus often remains unable to reproduce and verify new algorithms as the technical and organizational hurdles to build up and operate a research platform are very high.

[1]fortiss GmbH, An-Institut Technische Universität München, Munich, Germany
[2]Chair of Robotics, Artificial Intelligence and Real-time Systems, Technische Universität München, Munich, Germany

Only recently, open source driving stacks got attention, potentially enabling research groups around the world to solve real-world problems. *Driving stack* generally denotes the set of all software components that are necessary for fully autonomous driving. A prominent example is *Apollo*, an open source project funded and operated by Baidu [7]. A joint project of various Japanese universities has produced the open-source stack *Autoware* [8] that claims similar capabilities as Apollo. Both projects expect deployment on a specific exclusive set of supported hardware architectures.

However, transferring a software stack from a specific vehicle hardware architecture, including sensor setup, actuation interfaces, and computational hardware, to another is challenging due to missing standardized hardware configurations for autonomous vehicles. As an in-depth discussion of the architectures is often neglected in scientific publications, engineering knowledge is lost from one generation of researchers to another.

We provide a systematic analysis of the steps we applied to integrate the Apollo Driving Stack on our research vehicle with a detailed presentation of the lessons learned (see Fig. 1). We designed the vehicle as a research prototype for the development of cognitive systems and autonomous driving function prototyping. The presented modules bridge the gap between the hardware and driving stack. We specifically contribute

- a detailed discussion on how and for which reasons we modified a production vehicle equipped with state-of-the-art sensors and the access to control lateral and longitudinal motion, starting from a discussion on other

978-1-7281-0560-4/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 27,2022 at 11:02:16 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Hardware comparison of relevant research vehicles from an early demonstrator, the DARPA challenges and the Apollo reference vehicle. We use ♦ to indicate an existing component, ◊ to indicate its nonexistence and ? if such information is not public.

	1994	2007		2013	2015	2016		2018	2019
	VaMP [11]	Junior [1]	Boss [2]	Bertha [6]	RACE [10]	Halmstad [14]	Bertha [13]	Apollo [7]	*fortuna*
Camera	♦ front/rear	◊	♦ front	♦ stereo	♦ front	◊	♦ stereo/360°	♦ front	♦ 360°
Lidar	◊	♦ 64L	♦	◊	♦ 4L	◊	♦ 4L	♦ 64L	♦ 32L
Radar	◊	♦	♦	♦	♦	♦ (series)	♦	♦	♦
GPS	◊	♦	♦	♦	♦	♦ rtk	♦ rtk	♦ rtk	♦ rtk
INS	♦	♦	♦	?	♦	♦	♦	♦	♦
RT comp.	♦	?	◊	?	♦	♦	♦	◊	♦
PC	♦	♦	♦	?	♦	♦	♦	♦	♦
GPU	◊	◊	◊	?	◊	◊	♦	♦	♦

research demonstrators,

- how we used and adopted the open-source Apollo stack,
- the lessons learned as a guideline for the research community.

The paper is structured as follows: First, we compare our platform to previous research vehicles. Then, we present the challenges of transferring the Apollo stack to our vehicle. In the end, we detail the sensor and control architecture setups and algorithms adapted to run with Apollo on *fortuna*.

II. Relation to Other Research Platforms

Our research vehicle *fortuna* shall serve as a platform to study variants of automated driving algorithms. Our institute has previously studied architectural aspects with the vehicle demonstrator described in Section II-A. Various other autonomous vehicle research platforms have already been constructed. In Section II-B, we present a selected set of vehicles.

A. *fortuna* and the RACE Demonstrator in Contrast

The project RACE demonstrated a robust and reliant centralized electronic system architecture and a runtime environment which supports the integration of mixed-criticality components up to Automotive Safety Integrity Level D (ASIL D), while providing timing guarantees as well as plug and play capability [9]. Furthermore, the platform provides error detection and failure handling mechanisms on top of a real-time operating system and scheduler. The research vehicle set up within RACE was equipped with many prototypical components to demonstrate the capability of a central platform computer and its middleware [10]. Among these is a Steer-by-Wire system without mechanical fall-back, a prototypical intelligent brake actuator and non-production wheel hub motors. The sensor set on the other hand is limited, cf. Table I. It could be demonstrated that the E/E architecture meets the requirements for a future homologation as a fail-operational system, which is mandatory for SAE Level 5 automation. Nevertheless, the RACE vehicle demonstrator has no legal allowance to drive on public roads in Germany and is therefore not suited for autonomous driving algorithm design.

fortuna, on the other hand, shall serve as a platform for automated driving algorithm development, rapid prototyping and testing in real-world traffic scenarios, while always including a safety driver. Hence it needs to meet production vehicle standards for the manual driving functions, which build the fall-back for ensuring road safety. Furthermore, all safety-critical hardware parts are certified production vehicle components.

B. *fortuna* Compared to Other Research Vehicles

In this section, we discuss former research vehicles and compare their hardware and software setup to *fortuna*. We chose to include the vehicles of the two winning teams of the DARPA Urban Challenge and the Grand Cooperative Driving Challenge (GCDC), respectively, as these competitions presented two milestones in autonomous driving research. Table I depicts the hardware evolution of research prototypes in the past 15 years. We start by comparing the sensor setups.

1) Sensors: VaMP, developed by Thomanek and Dickmanns [11] in the 1990s, was limited to Adaptive Cruise Control (ACC) and Lane Change (LC) applications on highways due to reduced sensor capabilities with only four cameras and computation hardware of approximately 50 processing units. Fifteen years later, a successful choice for teams in the DARPA Urban Challenge was a fusion of high-end Lidar and Radar sensor data. However, back then perception systems could not rely on GPU-based acceleration and deep neural networks. Since then, Radar and Lidar sensors for detection and camera-based systems for classification have been used in urban environments. Recent advances in multi-sensor data fusion using machine learning methods have encouraged to use more advanced sensors and setups. With the Apollo reference vehicle [7] various scenarios have been demonstrated using different sensor setups. The DARPA Urban Challenge established high-precision Global Positioning System (GPS) as a standard. Since 2016, systems with Real-Time Kinematic (RTK) have benefited from increased localization accuracy.

2) Communication Interfaces: Before 2016, research platforms did not focus on inter-vehicle cooperation or communication devices. The first GCDC [12] in 2016 confronted researchers with cooperative ACC scenarios and introduced

Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 27,2022 at 11:02:16 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Software stack characteristics of the discussed vehicle platforms.

	VaMP [11]	Junior [1]	Boss [2]	Bertha [6]	RACE [10]	Halmstad [14]	Bertha [13]	Apollo [7]
Application	German Highway	DARPA Urban Challenge		German Rural	Parking	Cooperative Driving Challenge		Various
Licensing	proprietary	partly open	proprietary	proprietary	proprietary	proprietary	proprietary	open
Middleware	?	publish/ subscribe IPC	publish/ subscribe IPC	?	RACE RTE	LCM	ROS	Cyber RT
OS on PC	?	Linux	?	?	PikeOS	Linux	Linux	Linux
Functional Safety	None	Watchdog module	Error Recovery	?	supporting ASIL D	Trust System	?	System Health Monitor
Controller on	?	PC	?	?	RACE DDC	MicroAutobox	realtime onboard comp.	PC

a V2V communication protocol. This challenge made appropriate communication devices necessary in the participating platforms. As fusing self-perceived sensor data and Vehicle-to-Vehicle (V2V) data proved to be challenging, Nunen *et al.* [12] selected to use the communication interface and a Radar, whereas Tas *et al.* [13] selected to use the Vehicle-to-Everything (V2X) inputs only. Similar competitors participated in the second GCDC [14], which also included lateral maneuvers with the need for communication and cooperation.

3) Control: Focusing on the hardware for control purposes, we observe no clear tendency towards a separation of control algorithms onto Real-Time computing system (RT comp.) with vehicle bus access and soft real-time PC hardware (see Table II). Since using real-time computing platforms increases robustness to software crashes or system failures and ensures safe communication with the production vehicle hardware, we employ such a setup.

4) Middleware: Nearly all research vehicles rely on a customized middleware layer between the specific hardware and software modules. A comparison is shown in Table II. The open source ROS has evolved into a popular middleware for autonomous driving prototypes. At the time of the DARPA Urban Challenge in 2007, ROS had not been released yet. The winning teams implemented a network-based publish/subscribe IPC system, which also formed the basis of ROS. Although no information was provided on the software architecture used for the Bertha Benz Memorial Drive [6], Bertha was running ROS during the DARPA Grand Cooperative Challenge in 2016 [13]. Baidu based the first versions of Apollo on an extended ROS by a downwards-compatible message protocol based on Google Protocol Buffers[1] and decentralized node management. Starting from version 3.5, ROS was replaced by Cyber RT, a custom middleware that claims to be more performant and easier to use. With Lightweight Communication and Marshalling (LCM) [15] serving as an alternative, no standard has evolved until now.

Considerations of functional safety are commonly neglected with research vehicles. Nevertheless, watchdogs and sanity checks usually handle algorithm and system errors.

[1]https://developers.google.com/protocol-buffers/

Fig. 2: The *fortuna* autonomous driving vehicle demonstrator: a modified VW Passat with Lidar sensors and antennas on the roof rack, additional cameras inside the vehicle and additional radar sensors integrated in the bumpers.

III. Hardware Setup

To meet the requirements of rapid driving algorithm prototyping, we chose to modify a production vehicle with non-production-vehicle hardware as well as providing access to production sensors and actuators. We refitted a 2018 Volkswagen Passat Variant GTE (see Fig. 2). This setup leverages the need for innovative and powerful hardware and the need for a safe and reliable base hardware setup. We did not aim for a hardware setup of a production vehicle in terms of redundancy or power consumption.

An architecture overview of *fortuna*'s additional hardware is depicted in Fig. 3. Fig. 4 provides an impression of the installed setup in the trunk. The modifications include additional sensors, interfaces to access production vehicle bus networks and four computers connected via an industrial gigabit Ethernet switch splitting the traffic into several virtual networks.

- One industry standard real-time rapid-prototyping control unit (a dSpace Micro Autobox II) with an IBM Power PC 900MHz CPU and 16MB RAM for control algorithms with various low-latency hardware interfaces (including CAN) running a real-time operating system. We run the low-level trajectory control on this computer as described in Section VI.
- Two PCs with Intel i7 3.4GHz quad-core CPU and 32GB RAM for sensor data processing, motion plan-

Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 27,2022 at 11:02:16 UTC from IEEE Xplore. Restrictions apply.

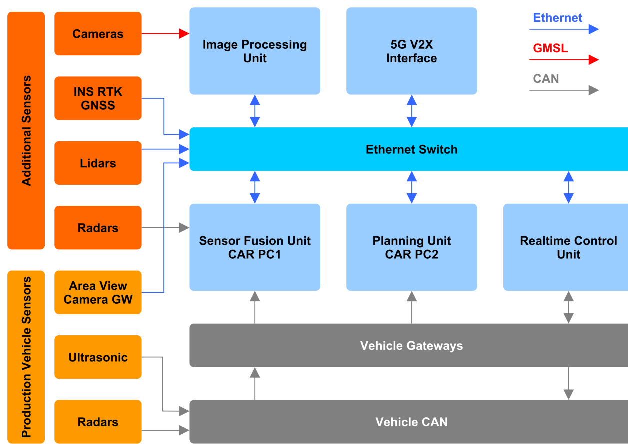

Fig. 3: Schematic overview of the hardware setup and the interfaces between the components.

Fig. 4: The additional hardware installed in the trunk showing (clockwise) the power supply, the two car PCs, the Drive PX 2, the iNat FSSG, the Micro Autobox, the CAN gateways, the Ethernet switch, and the CAN patch panel.

ning, human-machine interfaces and further software components running Ubuntu Linux. These PCs run the driving stack as described in Section IV.

- One Nvidia Drive PX 2 AutoChauffeur with two Pascal GPUs running Nvidia Drive Works for accessing camera images. The platform is mainly used for vision-based perception and neural network inference as sketched in Section V.

The hardware setup includes a 12V backup battery with additional power management and a connection to the high-voltage system of the production vehicle. A prototype cellular 5G interface realizes V2X connectivity.

Proprietary gateways enable access to the CAN buses of the production vehicle, which allows reading sensor and vehicle information. Write access enables automated driving through steer-by-wire. The production vehicle sensor data include object lists detected by the radar sensors of the ACC system and kinematic vehicle state information. Also, raw data from the ultrasonic sensors and camera images from the Area View surround view cameras are available.

More in detail, the vehicle is equipped with the following additional sensors, cf. Fig. 2, allowing a 360° Field Of View (FOV) avoiding blind spots.

- Three Velodyne Lidars: one VLP-32C with 32 layers in a central horizontal position on the rooftop and two VLP-16 with 16 layers at each side of the vehicle roof, inclined to scan the areas at each side of the vehicle. We regard Lidar sensors as mandatory for automated driving above SAE Level 3. The setup was chosen to provide a sound point cloud density in combination with a sufficiently broad sensor range for various scenarios.
- Five Sekonix cameras: two front-facing cameras, one with 60° FOV, and one with 120° FOV, one camera to each side and one rear camera, all with 120° FOV
- Four Smartmicro UMRR-146 radars: two facing forwards and two backward, integrated into the bumpers with access to raw sensor data
- Inertial Navigation System (INS): iMAR iNAT FSSG-1, a fiber optic gyro (FOG) based INS supporting RTK with integrated Global Navigation Satellite System (GNSS) receiver offering a localization precision of up to 2cm. The device can serve as a positioning unit and also as high precision reference localization for algorithm validation. As we consider a very high and reproducible localization measurement as essential for benchmarking autonomous driving functions, we decided on this industry-standard but non-automotive production grade device.

Key switches are installed for safety reasons and allow to power and enable the reading access measurement system and to enable writing CAN access for longitudinal and lateral control. An emergency shutdown button allows the safety driver to return to a production vehicle mode.

In contrast to the Apollo reference vehicle [7] or the Bertha vehicle [13], we chose to equip the vehicle with more than one computer to separate functionality. This setup comes at the price of network communication overhead and necessary design decisions on how to connect sensors and devices. For example, we connect the 360° FOV camera setup to the Drive PX 2 hardware via Gigabit Multimedia Serial Link (GMSL). This wiring hinders us from running image-based perception algorithms on one of the PCs.

IV. Driving Stack

We chose to base the software setup on the Apollo driving stack developed and maintained by Baidu [7]. Baidu claims that the stack contains all necessary modules for SAE Level 5 autonomy [7]. When starting this work, we used version 2.5, the newest at that time. As of now, we migrated our modules to version 3.5. The stack has a very modular structure and brings its custom-built middleware to exchange information. Since the Apollo stack is embedded in a growing open-source community and many companies have joined the Apollo board, we chose to use Apollo over other open-source stacks such as Autoware [8] or the Junior Driving Stack [1].

For a comprehensive description of the Apollo software design and architecture, the reader is referred to the docu-

Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 27,2022 at 11:02:16 UTC from IEEE Xplore. Restrictions apply.

mentation in the open source repository[2]. This section will focus on the extensions we needed to implement to run Apollo on our vehicle.

To adapt Apollo and to run it on our research vehicle, we focused on modifying the localization, perception and controller modules. Additionally, we developed several adapters to connect the vehicle hardware with the Apollo stack. However, we aimed to introduce minimal changes to the Apollo stack to maintain the original functionality. Furthermore, we developed additional modules to verify the functionality of the stack on the research vehicle, such as a mocked localization and a mocked planner. We are thus able to send trajectories with various lengths, speeds and steering angles, making them an ideal validation tool.

Since we use different hardware for localization than the Apollo reference vehicle, we had to develop a customized localization adapter. As the INS/GNSS provides highly accurate and consistent measurements, we decided to feed an already filtered position into the stack. However, we did not change the localization module to not lose any existing functionalities such as a watchdog that detects irregularities.

Our sensor setup is different from the recommended Apollo hardware-setup, and we re-implemented parts of the perception pipeline. We describe these adaptions in the perception modules and adapters in greater detail in Section V.

We chose to replace the controller used by Apollo and run it on a separated real-time platform instead. This separation facilitates a higher level of safety by separating control tasks in real-time execution for other software applications on different hardware platforms. Therefore, we developed an adapter communicating between the Apollo stack and the real-time system along with a trajectory tracking controller suitable for real-time execution. This software architecture will be described in greater detail in Section VI. We observed that this separation yields more stable system performance and enables us to optimally use the different benefits of the modular computing hardware setup. The development of the autonomous function stack as a mixed-criticality system on a single computer is of high scientific relevance but addresses other research aspects than driving function development.

In summary, Apollo provides prebuilt functionality and a well-structured code-base. For the stack to run on our research vehicle, we had to change drivers, add adapters and customize modules.

V. Perception and Sensor Setup

This section describes how we calibrated the multi-sensor setup, integrated it into Apollo and further how we implemented a basic sensor fusion algorithm. An example is shown in Fig. 5.

Up to this point, we calibrated all five Sekonix cameras and all three Velodyne Lidars towards a base link coordinate system which is located in the center of the rear axis. The section will conclude with a short discussion on the integration of the custom sensor setup into the Apollo framework.

[2]https://github.com/ApolloAuto/apollo

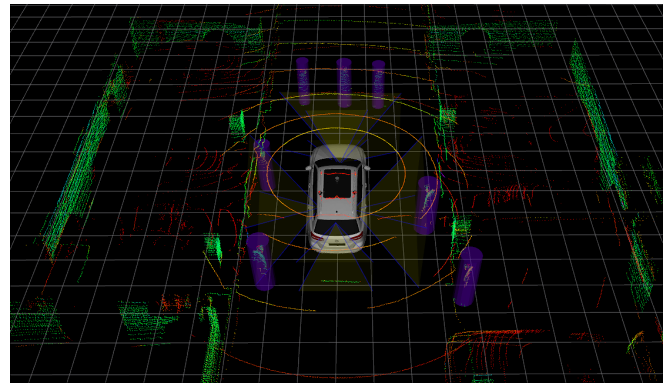

Fig. 5: Lidar pointcloud with 3D pedestrian detections showing the perception components.

To calibrate the sensors we calculated the position of the central Lidar followed by a semi-manual calibration of each camera and the side-facing Lidars with respect to the central Lidar.

A. Ground Plane Estimation

For estimating the ground plane, it is essential to handle outliers. Especially point clouds generated from Lidar sensors are sensitive to distance and angle of incidence of emitted rays. This explains why ground points further away from the sensor suffer from higher noise as the incidence angle gets sharper. To tackle this problem, we implemented a ground plane estimation algorithm based on Random sample consensus (RANSAC) [16]. The model needed for the estimation is a plane which consists of 4 degrees of freedom $\vec{n}\vec{x}^T + d = 0$ with the normal vector $\vec{n} \in \mathbb{R}^3$ and an offset $d \in \mathbb{R}$ of the plain. A point $\vec{x} \in \mathbb{R}^3$ lies on the plane if the equation evaluates to true.

On each RANSAC iteration, at least three points are randomly selected to form a plane. All the remaining points are evaluated on the plane's equation by thresholding the plane-point distance. The points on the plane are saved in the consensus set. After convergence, all points in the final set are used to estimate the final ground plane.

This algorithm assumes that a large portion of the point cloud is the ground plane. Therefore, wide and flat locations – such as parking lots – are preferred.

B. Semi-Manual Lidar-to-Image and Lidar-to-Lidar Calibration

For calibrating each camera to the vehicle base link coordinate system, we estimated the position of each camera separately towards the central Lidar following the work of Dhall *et al.* [17][3]. We selected more than six 3D-2D point correspondences manually in the point cloud and the corresponding image. These points are then used to solve for the projection matrix $\mathbf{P} = [\mathbf{R}|\vec{t}]$, in which the rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ is represented by its yaw, pitch, and roll angles. The vector $\vec{t} \in \mathbb{R}^3$ is the resulting translation from

[3]Implementation of https://github.com/agarwa65/lidar_camera_calibration

Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 27,2022 at 11:02:16 UTC from IEEE Xplore. Restrictions apply.

the central Lidar coordinate system to the camera origin. The solution for the overdetermined linear equation system was then estimated by a Sequential Least Squares Programming optimization algorithm.

We calibrated the two side-facing Lidars by using the Iterative Closest Point (ICP) [18] algorithm to estimate the six Degrees of Freedom (DoF) transformation between the point clouds of each side-facing Lidar and the central Lidar, with a rough initial estimate. To further improve the calibration quality manual fine tuning was performed.

C. Perception Pipeline and Apollo Integration Challenges

As a first integration step, we set up a fusion of camera detection and Lidar point clouds for object detection based on frustums [19] independent of Apollo using standard ROS making use of the 360° FOV of camera and lidar. A state-of-the-art deep learning based object detector [20] was used to detect pedestrians in all cameras. The bounding box was then projected into 3D Euclidean space resulting in four lines representing the corners of the 2D bounding box. The area inside those lines is called frustum. All points outside this frustum were removed, and the resulting points were clustered using DBSCAN [21]. For simplicity, we assume that the target object is not occluded and consists of sufficiently many points so that the closest cluster to the ego-vehicle can be chosen as a detection candidate. Afterward, a cylinder is fitted to the cluster leading to the detection. Fig. 5 visualizes the algorithm.

We selected *fortuna's* sensor set before Apollo was released and also with highway scenarios in focus. It is thus independent of the Apollo perception pipeline inputs. We faced several challenges while integrating the sensor set when we started using Apollo 2.5. With more recent versions (3.5 as of now) the integration barriers decreased, and mainly configuration work was necessary.

Since the cameras are connected to the Drive PX 2, Apollo's perception components need to be evaluated directly on the device to avoid costly forwarding of raw camera data to another PC. We consider this future work as well as the usage of other sensors mentioned in Section III. As a proof of concept connect a standard off-the-shelf USB camera to the car PC along with the central Velodyne-32 Lidar sensor and run the Apollo perception module out of the box.

VI. Trajectory Control

This section describes how we implemented and tested our trajectory tracking controller. As main contributions, we consider the description of the communication setup between the non-real-time trajectory planner, the real-time trajectory controller and the vehicle control units (cf. Section VI-B and Section VI-C). Also, we show how we extended a trajectory control algorithm from literature. Section VI-A also states the architecture of the component. Furthermore, we briefly outline our test and analysis tooling in Section VI-D.

A. Control Algorithm

The control algorithm is based on the work of Werling *et al.* [22] and shown as a block diagram in the middle part of Fig. 6. Based on the driving situation we either apply a full trajectory or a path tracking mode. The suitable control strategy and parametrization is selected using the received trajectory.

In the trajectory tracking case, we interpolate using the time on the given trajectory. As a common time base for the trajectory planner and the controller, we use the localization time signal. In the path tracking mode, the point with the closest Euclidean distance to the current vehicle pose on the trajectory is extracted as the reference point. The trajectory planner has to ensure that a sufficient backward horizon of the trajectory is available to guarantee a valid result of this interpolation.

A full trajectory tracking is especially valuable when driving on longer road segments with sufficient time and space to cope with sensor and actuator errors. When maneuvering, time aspects are of minor importance, and the situation is less dynamic. Trajectory tracking can also yield suboptimal final poses. The errors and their derivatives are extracted from the tracking point in a Frenet reference frame. Using input substitution and backstepping asymptotic stability of the control law can be proven [22].

For the sake of driving smoothness and planner error tolerance, we limit the absolute values and rates of all control signals.

A separated software module, implemented as finite state automaton, activates and parametrizes the different controller components based on the received trajectory and a host PC HMI component (omitted in Fig. 6). With this, we achieve a full separation of control algorithm code from functional execution logic.

We perform basic consistency checking of each trajectory and localization signal in terms of data and time validity. Furthermore, we detect actuator failures and vehicle interface errors. In case of an error, the control is handed over to the safety driver. More sophisticated errors like a bad tracking quality are expected to be treated by higher level components.

B. Trajectory Planner Communication Interfaces

The control algorithm is running on a rapid prototyping hard real-time computation platform whereas a PC-like hardware executes the trajectory generation components in a soft real-time context. As the trajectory is available to the controller over a particular horizon, no real-time communication between planner and controllers has to be implemented. Consequently, delays or packet loss in communication become acceptable. Also, short planner computation delays still result in smooth motions. In case of a planner software failure, the controller can still evaluate the last valid trajectory and can trigger an emergency stop or hand over the control to the safety driver.

The Apollo trajectory planner outputs a collision-free trajectory and transfers a sampled representation on a defined horizon to the trajectory controller. The controller performs no more collision checks allowing it to run at a high frequency.

Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 27,2022 at 11:02:16 UTC from IEEE Xplore. Restrictions apply.

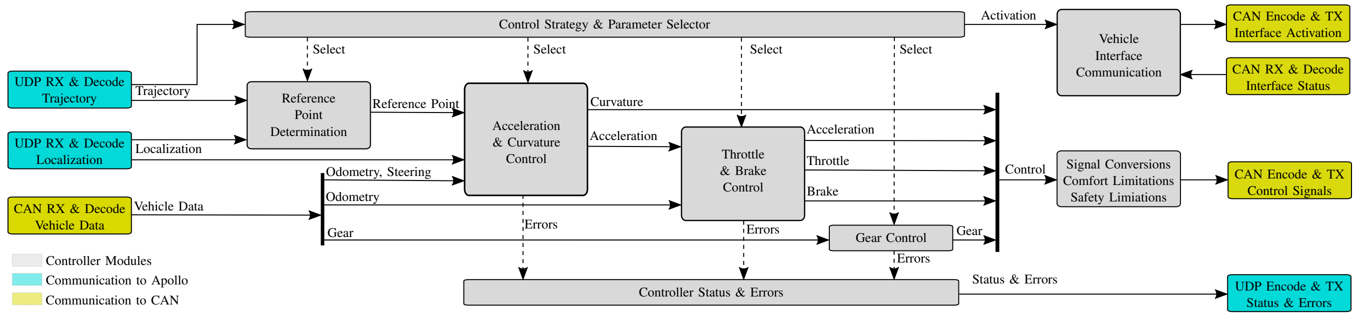

Fig. 6: Architecture overview of the trajectory tracking controller with the longitudinal/lateral vehicle control flow from the left to the right and the module parametrization and error handling flow from top to bottom.

Fig. 6 depicts the interfaces of the controller and the communication channels. The communication to the trajectory planner and the localization module is realized using Ethernet with a UDP protocol. With the interpolation method described in Section VI-A, no assumptions on the time or spatial distance of trajectory points are required. As an encoding format of the UDP messages, we use the protocol buffer definitions for trajectory and localization from Apollo.

C. Vehicle Communication Interfaces and Execution Platform

In contrast to the Apollo reference vehicle, we separate the computers for planning, perception and other driving functions from the closed loop vehicle control. That way, in case of timing problems on the computers or Ethernet network outtakes, we still can keep up the control loop on a short horizon. The trajectory controller is executed on a dSpace Micro Autobox II having a cycle time of 10 ms. Only this real-time hardware holds access to the vehicle controls.

The controller does not directly influence the vehicle actuators but computes high-level control values like an acceleration and steering wheel angle command. To control the vehicle motion a subsequent vehicle gateway control unit uses these high-level control values and actuates the production vehicle's control unit interfaces. The main interfaces are

- the acceleration interface from the production vehicle's automatic cruise control,
- the steering wheel angle interface from the production vehicle's park assistance system.

For low speed and maneuvering scenarios with reverse driving segments, the gear selection, throttle, and brake are actuated directly. No modification of any production control units is necessary. The implementation of the low-level control interfaces and the actuation of the production vehicle control units are realized on private CAN buses. These are proprietary and out of the scope of this work.

D. Implementation, Test, and Analysis

We modeled the controller and the CAN and Ethernet interfaces in Mathworks Simulink with certain code parts embedded as C-Code S-Functions.

For a seamless module test without the real-time hardware, we execute the model on a development PC. Mocking the same interfaces as the vehicle CAN bus enables us to run open or closed loop tests with the controller running in Simulink on a computer while receiving trajectory and localization from Apollo or recorded data. Furthermore, code generation to implement a virtual controller as Apollo node is planned to automate the test setup further.

To debug errors, it has been proven valuable to record all input signals and internal controller states using the toolchains offered of the different platforms [23]. A unified analysis and plotting framework has been implemented to analyze the recorded data.

VII. Conclusion

In this work, we pointed out the challenges, pitfalls, and lessons learned we encountered while integrating and running the open-source driving stack Apollo on our research vehicle *fortuna*. The research platform is an ideal basis for further research on functional autonomous driving software components, especially with the legal allowance to drive on German roads.

We integrated our vehicle's hardware into the Apollo software stack by adding novel adapters, such as for the localization and vehicle control. Decoupling the controller from the other software stack components enables a safe control – also in case of prototype software or hardware failures.

However, we experienced notable engineering barriers while integrating Apollo with our vehicle hardware and sensor set. We separated the perception pipeline and sensor calibration from the Apollo stack and tailored it to our sensor set. Also our non-centralized computation hardware led to modifications in the driving stack. Nevertheless, Apollo proved to be a good choice to base our driving functions on due to its modularity and modern software design that made the integration of our custom components possible.

Future work will focus on a detailed evaluation on how Apollo performed with our hardware setup, including the discussion if this vehicle setup was a reasonable choice.

One of our next research goals is to modify the existing open source driving software stack to enforce a reliable vehicle behavior and the source code contribution to the community. We plan to address limitations of the ISO26262 norm for functional safety regarding autonomous driving.

Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 27,2022 at 11:02:16 UTC from IEEE Xplore. Restrictions apply.

References

[1] M. Montemerlo, J. Becker, S. Bhat, *et al.*, "Junior: The Stanford entry in the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, Sep. 2008.

[2] C. Urmson, J. Anhalt, D. Bagnell, *et al.*, "Autonomous driving in urban environments: Boss and the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, Aug. 2008.

[3] S. Kammel, J. Ziegler, B. Pitzer, *et al.*, "Team AnnieWAY's autonomous system for the 2007 DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 615–639, Sep. 2008.

[4] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Journal of Field Robotics*, vol. 25, no. 9, pp. 640–673, Sep. 2008.

[5] J. Levinson, J. Askeland, J. Becker, *et al.*, "Towards Fully Autonomous Driving: Systems and Algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 3–8.

[6] J. Ziegler, P. Bender, M. Schreiber, *et al.*, "Making bertha drive – an autonomous journey on a historic route," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[7] Baidu, *Apollo: an open autonomous driving platform*, 2018.

[8] S. Kato, S. Tokunaga, Y. Maruyama, *et al.*, "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems," *Proceedings - 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018*, pp. 287–296, 2018.

[9] S. Sommer, A. Camek, K. Becker, *et al.*, "RACE: A centralized platform computer based architecture for automotive applications," *2013 IEEE International Electric Vehicle Conference, IEVC 2013*, pp. 1–6, 2013.

[10] M. Buechel, J. Frtunikj, K. Becker, *et al.*, "An Automated Electric Vehicle Prototype Showing New Trends in Automotive Architectures," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, Sep. 2015, pp. 1274–1279.

[11] F. Thomanek and E. D. Dickmanns, "Autonomous road vehicle guidance in normal traffic," vol. 3, pp. 11–15, 1995.

[12] E. van Nunen, M. R. J. A. E. Kwakkernaat, J. Ploeg, and B. D. Netten, "Cooperative Competition for Future Mobility," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1018–1025, 2012.

[13] O. S. Tas, N. O. Salscheider, F. Poggenhans, *et al.*, "Making Bertha Cooperate – Team AnnieWAY's Entry to the 2016 Grand Cooperative Driving Challenge," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 4, pp. 1262–1276, 2018.

[14] M. Aramrattana, J. Detournay, C. Englund, *et al.*, "Team Halmstad Approach to Cooperative Driving in the Grand Cooperative Driving Challenge 2016," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 4, pp. 1248–1261, Apr. 2018.

[15] A. S. Huang, E. Olson, and D. C. Moore, "LCM: Lightweight Communications and Marshalling," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Oct. 2010, pp. 4057–4062.

[16] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[17] A. Dhall, K. Chelani, V. Radhakrishnan, and K. M. Krishna, "LiDAR-Camera Calibration using 3D-3D Point correspondences," *Computing Research Repository (CoRR)*, vol. abs/1705.0, May 2017.

[18] P. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, Feb. 1992.

[19] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D Object Detection from RGB-D Data," *Computing Research Repository (CoRR)*, vol. abs/1711.0, Nov. 2017.

[20] J. Huang, V. Rathod, C. Sun, *et al.*, "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jul. 2017, pp. 3296–3297.

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[22] M. Werling, L. Gröll, and G. Bretthauer, "Invariant Trajectory Tracking With a Full-Size Autonomous Road Vehicle," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 758–765, 2010.

[23] P. Minnerup, D. Lenz, T. Kessler, and A. Knoll, "Debugging Autonomous Driving Systems Using Serialized Software Components," *IFAC-PapersOnLine*, vol. 49, no. 15, pp. 44–49, 2016.

Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 27,2022 at 11:02:16 UTC from IEEE Xplore. Restrictions apply.