TECHNICAL ARTICLE

Sensor Fusion and Motion Control for Autonomous Racing Cars

Technical University of Munich

speedgoat
real-time simulation and testing

Abstract

This whitepaper describes the use of Speedgoat hardware and the related Simulink toolchain for the *Roborace* research project at the Technical University of Munich (TUM). The team is developing an autonomous driving software stack capable of operating a racing vehicle close to its physical limits. Speedgoat real-time solutions are used within the car (a real-time rapid prototyping controller unit) and to set up a hardware-in-the-loop (HIL) simulation. In the following whitepaper, we provide an overview of the motion control and sensor fusion algorithms and present open source packages for these functions ready to be used with off-the-shelf Speedgoat hardware.

Introduction

Autonomous driving is a topic of significant interest to industry and academia peers. Applying theoretical findings to research-level prototypes and then production vehicles is a key challenge. To address this and facilitate rapid prototyping, we present a software stack for an autonomous vehicle capable of vehicle motion control and localization using sensor fusion and a real-time testing workflow. While the former can serve as a track-proven basis for future research projects, the latter speeds up the development and gives high-quality experimental results at significantly lower cost and effort than trials with a full-size vehicle prototype. The work completed by the TUM was supported by Speedgoat. Simulink® was chosen as the main modeling tool and Speedgoat real-time target machines together with Simulink Real-Time™ as the real-time prototyping and testing platform.

Roborace Research Project at the TUM

Racing has always provided incentive to develop new automotive technologies. High speeds and operation at the vehicle's handling limit enable engineers to identify shortcomings with current technology and therefore enhance hardware and software beyond state-of-the-art standards. One of the key challenges is the development of algorithms with low calculation periods but sufficiently accurate results. Furthermore, the dynamic and unstructured driving situations at the racetrack challenge state-of-the-art planning and control algorithms. The work done by the team enlarges the operational design domain (ODD) of these algorithms, resulting in improved robustness to the extreme conditions on the racetrack.

Source: TUM Roborace Team

Figure 1 - DevBot 2.0 at the Monteblanco racetrack

To benefit from these developments, the Chair of Automotive Technology and the Chair of Automatic Control of the TUM decided to take part in the Roborace competition for autonomous racing to benchmark their research on a full-size racing vehicle [1], called 'DevBot 2.0'. The vehicle has been developed by Roborace based upon a standard Le Mans Prototype (LMP) chassis and is driven by two electric motors on the rear axle. The main idea behind the competition is to equip teams with an identical car hardware platform so that the performance only depends on the algorithm design and the team performance. The autonomous driving software stack runs on an NVIDIA Drive PX2 and a Speedgoat Mobile real-time target machine. During the Season Alpha in 2019, the teams carried out different tasks, such as minimizing lap time, racing multiple vehicles on a circuit in classic race formats and optimizing on-track localization and driving precision. This diversity of challenges required the TUM team to develop a holistic software architecture. This article will focus on the motion control and the sensor fusion parts of the software stack. The reader is referred to [3] and [4] for further details on the remaining software modules and how they are interconnected with the algorithms presented.

Importance of Real-Time Prototyping and Testing for Control Software Development in AD

Driving at the handling limits presents significant challenges for autonomous driving software:

- Multiple sensor signals must be combined to determine accurate and reliable information about the current vehicle state. These signals are captured at different rates and resolutions. Merging sensor signals will subsequently be referred to as ***sensor fusion***.

- The vehicle motion dynamics must be stabilized around a target race trajectory. This requires feedback information to mitigate external disturbances like wind gusts, track inclination and tarmac irregularities. This continuous process of stabilization will subsequently be referred to as ***vehicle motion control***.

To optimize the performance of control and fusion algorithms, high sampling rates, deterministic timing and real-time execution are crucial.

To enable this, the research team at TUM used Simulink Real-Time and a Speedgoat Mobile real-time target machine for rapid control prototyping. Directly from within Simulink, this allowed the team to:

- Deploy and execute the controls in real-time

- Tune parameters and log data

- Access the required communication interfaces

With this, the team was able to quickly iterate, tune and test their control designs and minimize the time needed for hardware and software integration.

Testing the controls was another major challenge since the track time was very limited. While desktop simulation can help to run initial tests, real-time aspects and communication interfaces cannot be covered with this approach. To cover these aspects, the team used a Speedgoat Performance real-time target machine. By running their vehicle model in real-time and using the actual hardware interfaces, the team was able to test their controls early on under realistic conditions.

The major advantages of this approach are:

- Testing safety critical functions in advance without risking damage to the vehicle

- Finding errors that would otherwise appear on the track

- Optimizing the performance of the controls

This allowed the team to make the best use of limited track time.

Software and Hardware

To replicate the conditions under which the controls described in this article will eventually be used in the DevBot 2.0 as closely as possible, an identical real-time prototyping platform was used. This joint solution from MathWorks® and Speedgoat for deterministic real-time prototyping and testing consists of Simulink Real-Time and Speedgoat real-time target machines.

MathWorks provides Simulink Real-Time, which includes a real-time operating system as well as several host capabilities that enable real-time applications to be created from Simulink models, deployed on Speedgoat real-time target machines, and controlled directly from Simulink and MATLAB®.

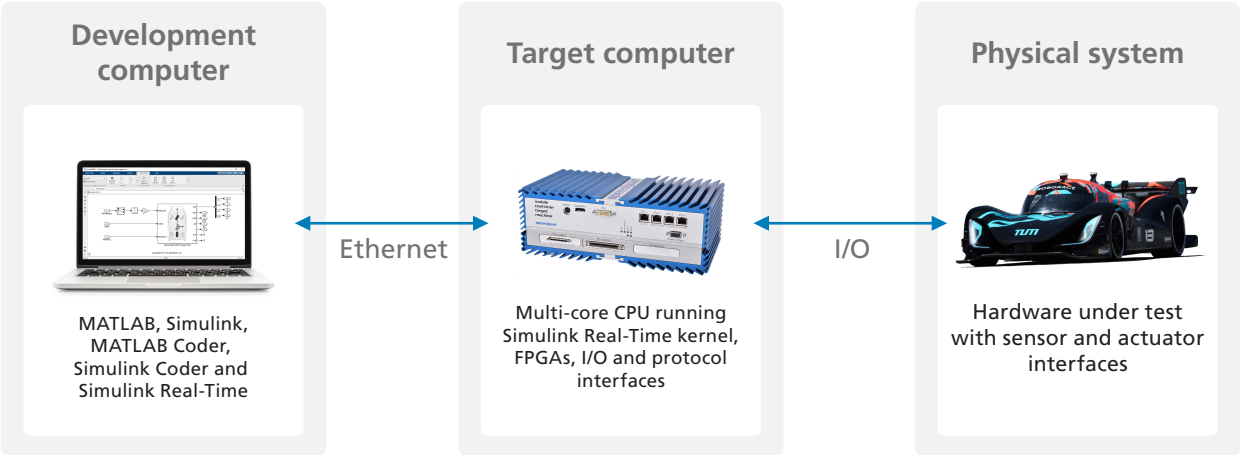

Figure 2 - Simulink Real-Time™ workflow

Speedgoat's solution for real-time simulation consists of the target machines that can be configured using Speedgoat's broad range of I/O and FPGA modules. These modules enable users to directly interface with the vehicle's networks, sensors, and actuators and, if needed, deploy algorithms onto FPGAs to achieve higher sampling rates.

The toolchain provided by Speedgoat and MathWorks enables the algorithms designed in Simulink to be directly transferred to a real-time platform offering the required interfaces, all from within Simulink. It is also the most direct path to simulating and testing algorithms in real-time.

Mobile real-time target machine

The TUM Roborace Team uses a Speedgoat Mobile real-time target machine in the DevBot 2.0.

The system is configured as follows:

- Processor: Intel i7 2.5 GHz Dual Core
- RAM: 4 Gb
- Modules: IO601 for CAN communication

Methodology

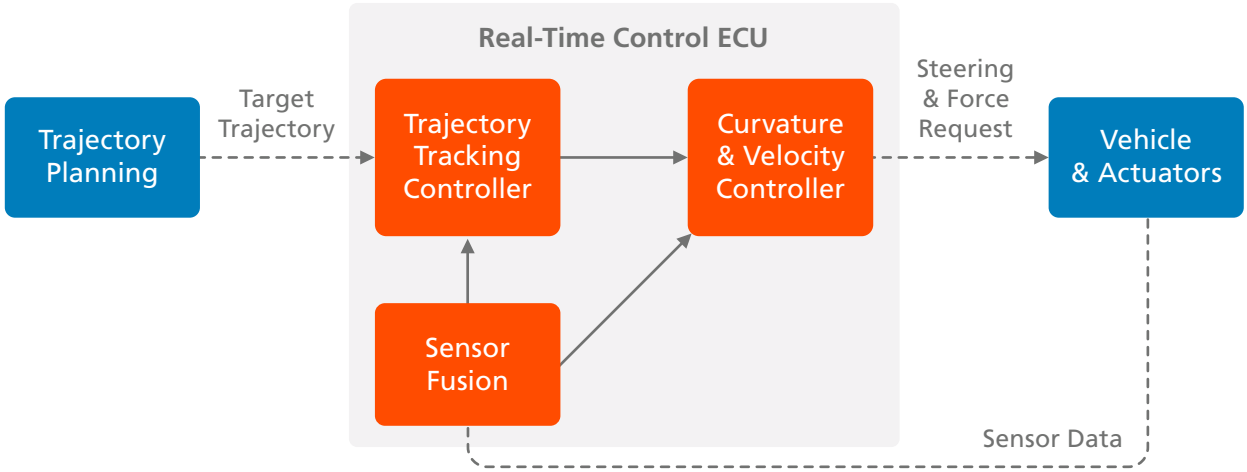

Figure 3 - Software architecture

This section describes the real-time control software of the TUM autonomous driving software stack. Its high-level architecture [3-4] is depicted in Figure 3. The software is split into a planning and a control module. As stated above, the control software module is executed on a real-time control unit to guarantee accurate timing in dynamic driving maneuvers. The real-time control software itself is divided into sensor fusion and vehicle motion control.

Sensor Fusion

The sensor fusion module is based on an Extended Kalman Filter combining measurements from different sensors. It allows the fusion of vehicle acceleration as well as yaw rate measurements (coming from an inertial measurement unit (IMU)), velocity measurements (e.g. an optical flow sensor or wheel speed sensors) and multiple localization sources. The localization signals may come from GPS, LIDAR-SLAM or Visual-SLAM. The minimum setup required for operation is one localization source and one IMU sensor. On the Devbot 2.0, a GPS and an IMU sensor are used.

The filter combines a dynamic system model and the sensor measurements. The model takes the current accelerations and yaw rate as inputs, and predicts the upcoming states using Newtonian mechanics. The velocity and position measurements are used to correct the predicted vehicle state. This simple model is superior to more complex vehicle dynamics models (e.g. a single-track model) as it does not introduce model bias in the nonlinear tire region due to inevitable parameter mismatch. In the sensor setup described, which is similar to others commonly used for autonomous driving, the choice of this simple model improves state estimation quality, rather than making it worse by introducing incorrect assumptions about tire or vehicle parameters. At the same time, the model is computationally cheap and delivers robust performance independent from the driving scenario. Details and theoretical background on this concept can be found in [2].

Vehicle Motion Control

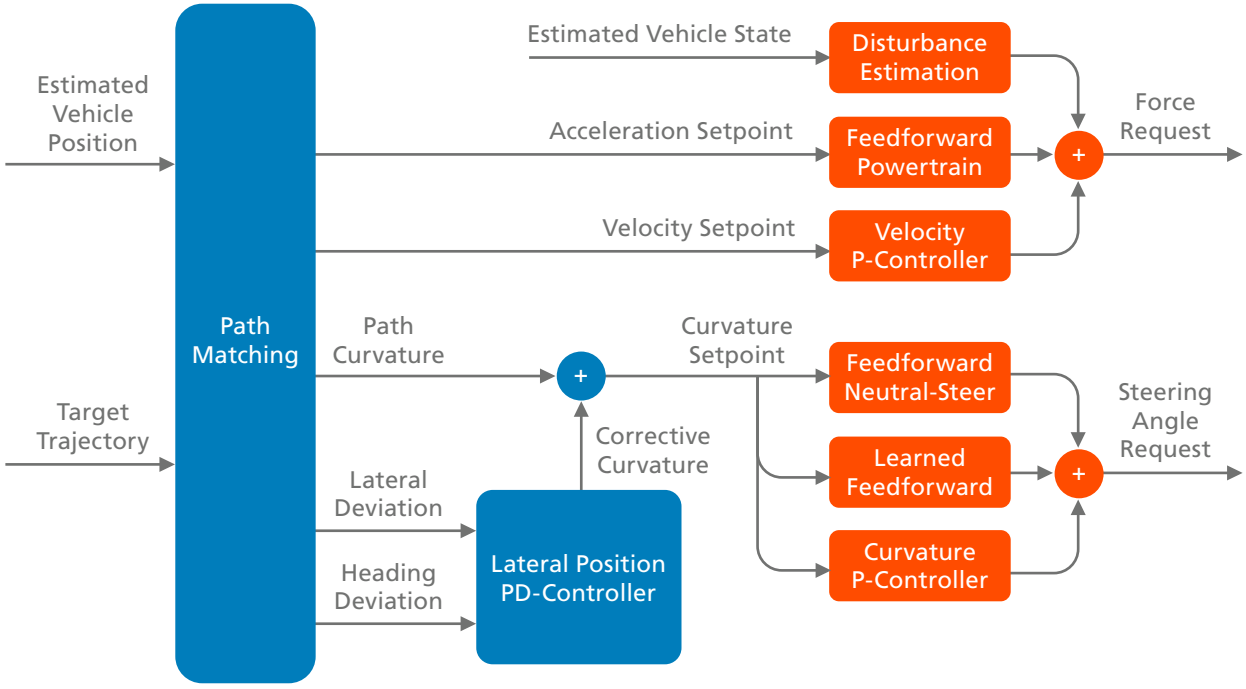

Figure 4: Vehicle motion control structure

The vehicle motion control part of the software generates appropriate steering and overall traction force requests based on the target trajectory and the current vehicle position. Its overall structure is depicted in Figure 4. The vehicle speed and the deviation from the target path are controlled. Furthermore, the lateral dynamics are stabilized via a low-level curvature controller. All controllers are based upon a two degrees of freedom architecture, separating the control request generation into a feedforward part and a feedback part [3].

The setpoint for the curvature controller is derived from a feedforward term for the current path curvature and a feedback part based on the lateral deviation and the velocity heading error. The curvature setpoint is converted into a steering angle using a neutral steer assumption from the basic vehicle kinematics. Furthermore, a learning component monitors the relation between the steering angle, vehicle velocity and driven curvature and adds a correction signal to the neutral steer assumption based on past data. The learning component compensates under- and oversteering as well as small steering actuator miscalibrations. The curvature controller is completed by a proportional feedback gain to reduce the influence of external disturbances and model uncertainty.
The longitudinal control task is covered by a velocity controller. It consists of a feedforward part, based on a stationary inverse powertrain model which compensates for driving resistances such as aerodynamic drag, and a proportional feedback combined with a disturbance estimation. The latter estimates the difference between the expected acceleration and actual measured acceleration and applies correction signals if they are not equal. This helps to mitigate unmodeled effects, such as track inclination.

Vehicle Dynamics Simulation

The simulation environment used for developing and evaluating the controller is based on a dual-track model. This choice is common for controller development, as it considers the most significant effects for vehicle motion control. It includes the combined lateral and longitudinal nonlinear tire behavior, the lateral and longitudinal load transfer for the tires as well as the wheel dynamics. The Vehicle Dynamics Blockset (VDBS) developed by MathWorks provides the building blocks required to create such models as well as several reference applications to get started easily. The results presented in this article were obtained using the passenger vehicle model from the Double-Lane Change Maneuver reference application from MathWorks. The vehicle model in conjunction with the TUM software stack were used to build a real-time simulation. An advanced version of this simulation model with modifications to meet the requirements of autonomous racing is also available in the open source software repository [5].

Real-Time Deployment to Speedgoat machines

This section outlines how the sensor fusion and vehicle control pipeline can be modeled within Simulink and then deployed to the Speedgoat Mobile real-time target machine using Simulink Real-Time. The models are divided into the three parts depicted in Figure 5: The vehicle simulation (tires, vehicle body, actuators and sensors), the vehicle control software and the trajectory planning emulation (provides the interface to the trajectory planning software and basic functionality for development purposes). This article and the corresponding software stack present two different variants for effective real-time testing: The first variant deploys all software components onto a single real-time system and leverages loopback hardware interfaces for communication between them. The second variant deploys the trajectory planning emulation and vehicle simulation on a separate real-time system. The actual vehicle control software can therefore be tested in a classic HIL setting. Example models for both setups are available from the software repository [5].

Software Design Methodology for Simulink

The software is designed using several advanced Simulink design techniques, which will be described below in addition to the reasons for using these techniques within the development workflow.

Most importantly, the use of **Simulink Projects** enables the vehicle control software and the vehicle simulation to be separated into different software repositories. The vehicle control software project references the vehicle simulation project. This allows the models managed in the simulation repository to be used within the controller development repository. Simulink Project manages the MATLAB path and allows a defined and reproducible working environment to be created for development and code generation on multiple development computers.

Referenced Models and **Bus Definitions** help make the software modular. The software consists of approximately 20 Simulink models (*.slx), which either provide a defined functionality or can be used to create different models. Their interfaces are specified independently from the models by utilizing Bus Definitions. This ensures consistent interfaces and is checked during the model update process. The separation into multiple files provides a natural separation of concerns, a well-known software development paradigm, and furthermore eases version control. Another technique to improve version control capabilities is to outsource complex functionality to .m-Files which can be called using Simulink MATLAB Function blocks. This enables text-based comparison and merge tools to be used in the version control software.

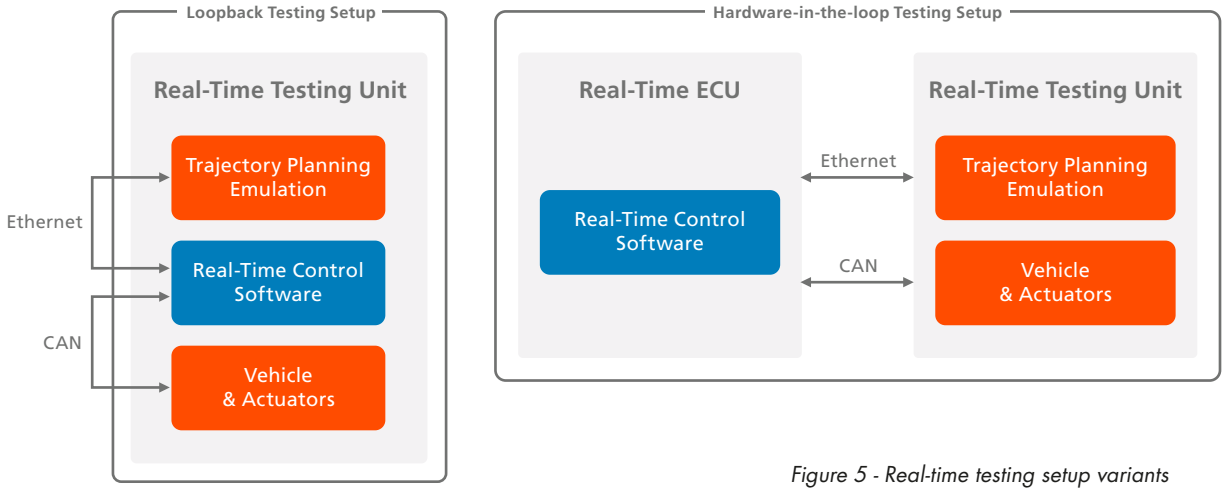

Figure 5 - Real-time testing setup variants

Finally, the software uses **Data Dictionaries (DD)**. These files can be thought of as a private workspace belonging only to the model to which they are attached. They allow the designer to separate algorithm development from data storage and can store multiple different parametrizations per algorithm. This is useful in case several vehicles are running the same code but require different control parameters. Furthermore, DDs may reference other DDs. A prominent use case for this is the interface DD for the Bus Definitions or a vehicle parameter DD.

Deployment to Speedgoat Real-Time Target Machine via Simulink Real-Time

This section describes the configuration of the loop-back simulation example setup (see Figure 5). A similar procedure can be applied for the HIL variant. The I/O modules of the target machines used for the examples slightly differ from the ones in the DevBot, however the configuration approach remains the same.

First, two real-time UDP interfaces and the CAN interface of the IO614 module must be configured as depicted in Figure 6. We will use 10.0.1.0 as the IP for the controller interface and 10.0.2.0 as the IP for the trajectory planning interface. The CAN interface of the controller is Module ID 1 – Port 1 and the interface of the vehicle simulation is Module ID 1 – Port 2. This results in the structure depicted in Figure 3. In the examples provided, sensor signals can be grouped into CAN messages and therefore be easily changed directly in Simulink. The UDP messages are handled in a similar way.

The complete real time application runs with a sample rate of two milliseconds and uses the ode2-solver. This configuration has proven to offer a good trade-off between accuracy and computational speed. The controller itself operates at four milliseconds and uses the fixed-step discrete solver.

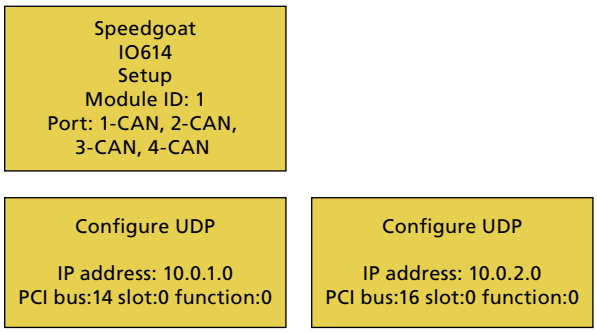

Figure 6 - Hardware configuration of loop-back testing model

Results

The VDBS double-lane change example model together with the software stack running on the HIL setup mentioned above accurately predicts the driving behavior on the Monteblanco racetrack, Spain (see Figure 7). The open source repository [5] provides several racetracks to enable controller development. The required target trajectories have been created using a software package developed by the Chair of Automotive Technology at TUM, which can be accessed at GitHub. Please see the Readme of this software package for further details.

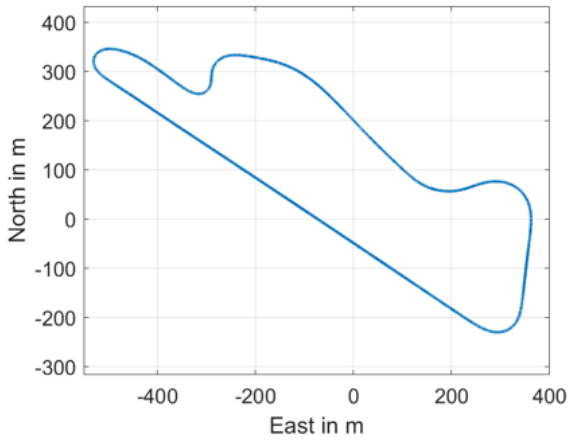

Figure 7 - Monteblanco racetrack, Spain

Vehicle Motion Control Performance

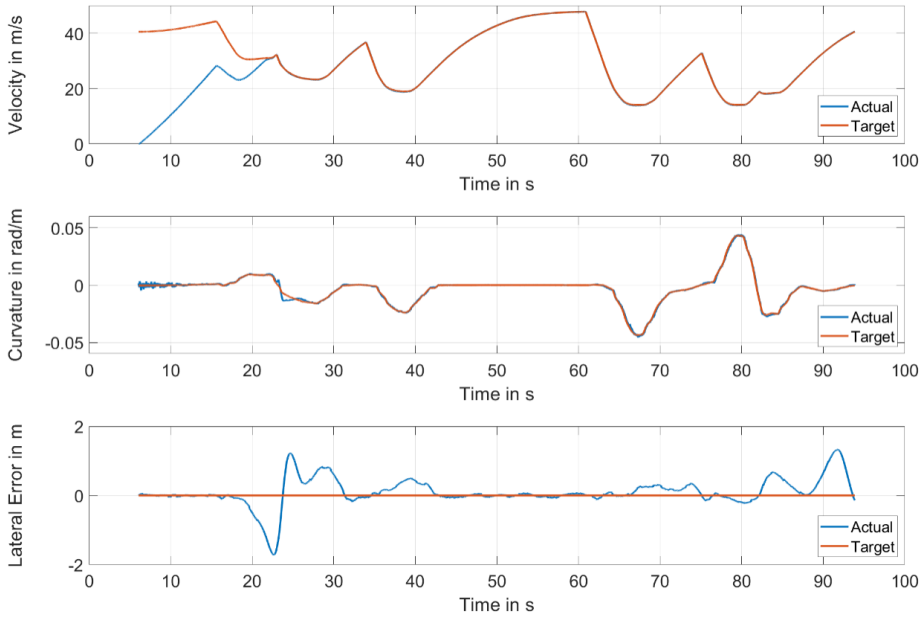

Figure 8 - Trajectory tracking performance

The tracking performance of the vehicle motion controller for the standard VDBS vehicle is depicted in Figure 8. The figure shows good tracking performance under challenging vehicle dynamics conditions with longitudinal and lateral accelerations up to $9_m / s^2$, even though the vehicle under control is not a racecar.

The largest lateral deviations occur in a demanding left-right, high-speed combination right after the back straight. A real-world evaluation of the control software for a Roborace DevBot on the Berlin 2019 Formula-E racetrack is presented in [3].

Sensor Fusion Performance

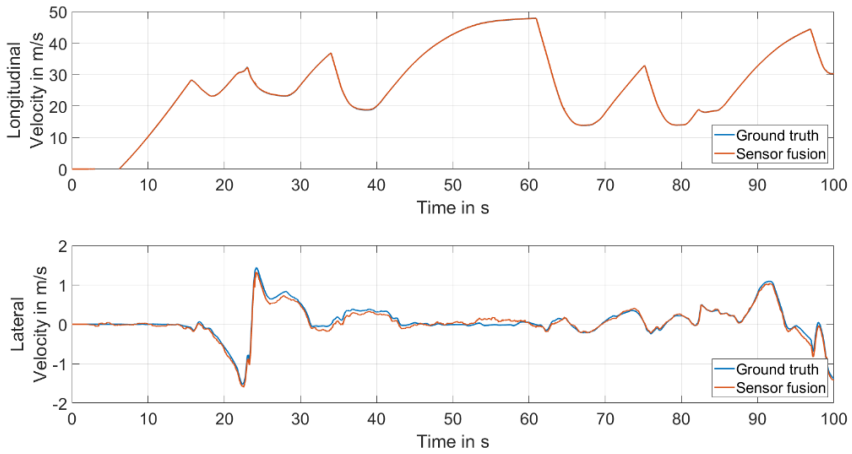

Figure 9 - Sensor fusion performance

The sensor fusion can be easily assessed in a HIL simulation, because ground truth data is available. The software does not have access to velocity measurements in this setup and therefore estimates the longitudinal and lateral velocities from the fusion of the localization and the IMU signals. Figure 9 shows a comparison of the estimated and the actual velocity obtained from ground truth logging from the simulation model. A detailed evaluation of the sensor fusion algorithm using real-world data is presented in [2].

Conclusion

We have presented a real-time control software stack capable of autonomous driving at the limits of handling. It consists of a sensor fusion and a vehicle motion control part. In fact, we have illustrated two concepts for real-time testing: A basic version running on a single real-time target machine and a full HIL setup for testing the vehicle control unit under realistic conditions. The open-source software stack [5] has examples for both setups and is therefore a good starting point for further development.

References

[1] *What can we learn from autonomous level 5 Motorsport?*, J. Betz, A. Wischnewski, A. Heilmeier, F. Nobis, T. Stahl, L. Hermansdorfer, B. Lohmann and M. Lienkamp, Proceedings of Chassis-Tech 2018
[2] *Vehicle Dynamics State Estimation and Localization for High Performance Race Cars*, A. Wischnewski, T. Stahl, J. Betz and Boris Lohmann, Proceedings of the IAV 2019 Conference
[3] *Minimum Curvature Trajectory Planning and Control for an Autonomous Race Car*, A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp and B. Lohmann, Vehicle System Dynamics
[4] *A Software Architecture for the Dynamic Path Planning of an Autonomous Racecar at the Limits of Handling*, J. Betz, A. Wischnewski, A. Heilmeier, F. Nobis, L. Hermansdorfer, T. Stahl, T. Herrmann and M. Lienkamp, Proceedings of the IEEE ICCVE 2019
[5] TUM Racing Software Stack, https://github.com/TUMFTM

Photo Credits:

TUM Roborace Team: https://www.mw.tum.de/en/ftm/main-research/intelligent-vehicle-systems/roborace-autonomous-motorsport/

The Authors:

Alexander Wischnewski, Timo Strässle, and Michael Lüthy