Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Networked and Autonomous Model-scale Vehicles for Experiments in Research and Education ⋆

Patrick Scheffe * **Janis Maczijewski** * **Maximilian Kloock** *
Alexandru Kampmann * **Andreas Derks** * **Stefan Kowalewski** *
Bassam Alrifaee *

* *Chair for Embedded Software, RWTH Aachen University, 52074 Aachen, Germany (corresponding e-mail: scheffe@embedded.rwth-aachen.de)*

Abstract: This paper presents the µCar, a 1:18 model-scale vehicle with Ackermann steering geometry developed for experiments in networked and autonomous driving in research and education. The vehicle is open source, moderately costed and highly flexible, which allows for many applications. It is equipped with an inertial measurement unit and an odometer and obtains its pose via WLAN from an indoor positioning system. The two supported operating modes for controlling the vehicle are (1) computing control inputs on external hardware, transmitting them via WLAN and applying received inputs to the actuators and (2) transmitting a reference trajectory via WLAN, which is then followed by a controller running on the onboard Raspberry Pi Zero W. The design allows identical vehicles to be used at the same time in order to conduct experiments with a large amount of networked agents.

Keywords: Control education using laboratory equipment, Connected Vehicles, Autonomous Vehicles, Multi-vehicle systems, Embedded computer control systems and applications, Embedded computer architectures, Remote and distributed control

SUPPLEMENTARY MATERIAL

A demonstration video of this work is available at `https://youtu.be/84R28cSxWQU`.

The vehicle software, bill of materials and a production tutorial can be found on our website `https://cpm.embedded.rwth-aachen.de`.

1. INTRODUCTION

Research on networked and autonomous vehicles is ongoing since multiple decades. When new methods are developed, the necessity of testing them arises. This can be done with little effort in simulation as in Naumann et al. (2018). The meaningfulness of results in simulation is restricted, as only aspects of reality that are modeled are considered. More meaningful are experiments in true scale, but those require a high effort and are expensive, especially when testing methods on networked vehicles, as multiple test platforms are required. Midway between those options, methods can be tested on scaled testbeds. In scaled experiments, many challenges of the true-scale problem are apparent, e.g. communication delays and losses, synchronization problems or actuator dynamics. Another benefit compared to the true-scale experiment is that setting up the experiment is simpler and quicker, which allows for rapid development cycles.

The curriculum at a university should prepare students for research in networked and autonomous vehicles. This includes for example the design of algorithms for embedded hardware, designing controllers for nonlinear systems, or coupling of networked agents for collision avoidance. Seeing an algorithm one has developed running in an experiment fills students with enthusiasm about learning concepts of control by applying it to the cyber-physical mobility (CPM) system. The modified model-scale vehicle proposed in this paper enables those experiments.

This paper is structured as follows. Section 2 compares model-scale vehicles with Ackermann steering geometry from literature. Section 4 describes how we transform a model-scale race car to a networked and autonomous vehicle with off-the-shelf components, excluding a printed circuit board. The lab environment in which the vehicles operate is sketched in section 3. In section 5, examples are given to show in what form the vehicles can be used in control education.

2. EXISTING PLATFORMS

In the last decade, a number of model-scale testbeds have been developed. In Paull et al. (2017), 15 platforms for education and research with a cost lower than $300 are compared. These differ from the model-scale vehicle we present, as they are wheeled differential drive platforms or platforms with slip-stick forwards motion.

⋆ This research is supported by the Deutsche Forschungsgemeinschaft (German Research Foundation) within the Priority Program SPP 1835 “Cooperative Interacting Automobiles” (grant number: KO 1430/17-1) and the Post Graduate Program GRK 1856 “Integrated Energy Supply Modules for Roadbound E-Mobility”.

In Table 1, an overview of recently developed model-scale vehicles with Ackermann steering geometry is given. Having a scaling factor of 1:43 and 1:24 respectively, the ORCA Racer (Liniger et al., 2014) and the Cambridge Minicar (Hyldmar et al., 2019) are smaller than the vehicle presented in this work. The ORCA Racer is based on the Kzosho dnano RC race car, but substitutes its original board with a custom printed circuit board (PCB). This board features an ARM Cortex-M4 processor, Bluetooth communication and an inertial measurement unit (IMU). The vehicles are designed to receive externally computed control inputs via Bluetooth, and apply these inputs with an onboard low-level controller (LLC). The Cambridge Minicar is based on the CMJ RC Cars Range Rover Sport. Its controlled by a Raspberry Pi Zero W. These vehicles are controlled by sending externally computed control inputs via broadband radio.
The Berkeley Autonomous Race Car (BARC) from Gonzales et al. (2016), the MIT Racecar from Karaman et al. (2017) and the F1/10 from O'Kelly et al. (2019) share the scale of 1:10. The mechanical base for all three vehicles is a Traxxas rally car. At this size, the vehicles are capable of carrying more computational power and more sensors additionally to an IMU. In the BARC, 4 rotary encoders are installed for speed measurement and a camera is mounted. Optionally, it is possible to install a lidar and a Global Navigation Satellite System (GNSS) receiver. The high-level controller (HLC) and main computing unit is an ODROID-XU4, the LLC, i.e. sensor read and actuator control, is performed with an Arduino Nano. The setup of the MIT Racecar and the F1/10 is similar. The speed is given by a VESC electronic speed controller, and optional sensors include a 3D stereo cameras and a lidar. The main computing element is the Nvidia Jetson Tegra X1. The greater computing power and additional sensors allow for onboard autonomy. This is also a reason why these setups cost around $1000. At the scale of 1:10, a lot of space is required for indoor experiments on cooperative driving with multiple vehicles. Due to the cost and the size of the platforms, indoor experiments with a large amount of vehicles are difficult.
At the largest scale of 1:5, the GATech Auto-Rally from Williams et al. (2016) and the IRT buggy from Reiter et al. (2014, 2017) are designed for outdoor experiments. The Auto-Rally is equipped with two forward facing cameras, a Lord Microstrain 3DM-GX4-25 IMU, a GNSS receiver, and wheel speed sensors. The computational power is provided by an Intel quad-core i7 processor, 16GB RAM, and an Nvidia GTX-750ti graphics card. With this elaborate hardware setup, the Auto-Rally is used for aggressive driving. The IRT buggy is designed for versatile use. It shares the separation of HLC and LLC in two hardware components with the BARC. Sensors include a GNSS-sensor, an IMU, and two rotary encoders at the rear wheels. Its modular setup allows for other sensors such as a lidar. Similar to the ORCA Racer, this platform is not open source.

The larger model-scale vehicles are equipped with sensors and computing power to allow autonomy. The µCar, as well as the ORCA Racer and the Cambridge Minicar are reliant on the interaction with a lab environment. This lab environment provides the positioning of the vehicles and therefore substitutes the GNSS of the real world

Table 1. Recent model-scale Ackermann-steering platforms

Vehicle name	Scale
ETHZ ORCA Racer	1:43
Cambridge Minicar	1:24
µCar	*1:18*
F1/10	1:10
BARC	1:10
MIT Racecar	1:10
GATech AutoRally	1:5
IRT buggy	1:5

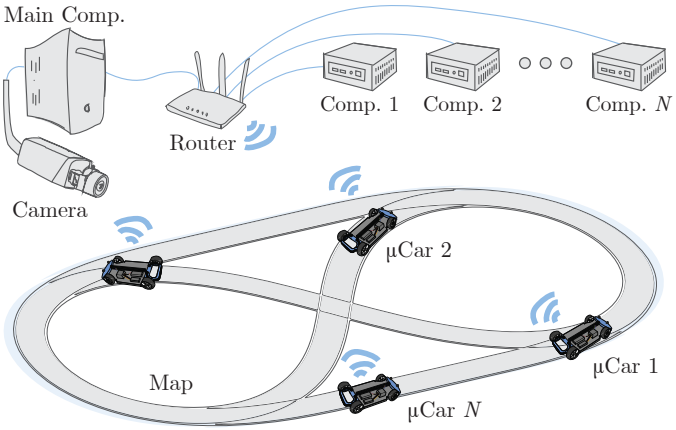

Fig. 1. CPM lab overview: vehicles communicate via WLAN with their respective computers and the indoor positioning system (IPS).

experiment. In the case of the Cambridge Minicar, this is done with an OptiTrack motion capture system that requires multiple cameras, while the lab environment of the ORCA Racer only uses one camera, similar as our CPM lab. In contrast to those two labs, in addition to the option of sending control inputs to the vehicle, a trajectory following mode exists, where an onboard controller determines the control inputs necessary to follow a given trajectory.

3. ENVIRONMENT: CPM LAB

The vehicles are used for experiments in a lab environment as visualized in Fig. 1, which we call CPM lab (Kloock et al., 2020a). This lab provides a driving area of about 4.5m × 4m. Communication between the vehicles and this environment is established through Data Distribution Service RTI Connext DDS. An IPS provides the vehicles with their pose (position and orientation) with a worst-case accuracy of 3.25 cm and 2.25°. A camera detects the position of the three LEDs on the vehicle. These LEDs define a vehicle's pose due to their arrangement on the vehicle in a non-equilateral triangle. The vehicle corresponding to a detected pose is identified with a signal code sent by the fourth LED on the vehicle as shown in Kloock et al. (2020b). Additionally, a reference trajectory or the actuator inputs for the vehicles are sent via WLAN. The vehicle returns its current state, which includes the estimated pose as well as sensor readings and actuator commands.

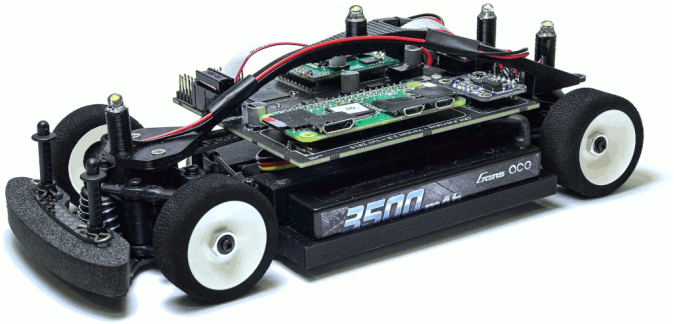

Fig. 2. The µCar, a 1:18 model-scale vehicle

Table 2. Components used in the µCar; cost rounded to the next integer

Item	Application	Cost [€]
XRAY M18 Pro	Mechanical platform	170
Gens ace 3500mAh	LiPo Battery	30
NF113LG-011	Motor	15
Hitec D89MW	Servo	50
PCB	Board	15
Raspberry Pi Zero W	MLC	18
8GB SD Card	Memory	7
ATmega2560	LLC	12
Pololu VNH5019	Motor Driver	23
DeboSens BNO055	IMU	34
Eletronic Parts		21
SUM		395

4. VEHICLE SETUP

The model-scale vehicle presented here is shown in Fig. 2. It is an Ackermann-steered, non-holonomic mobile robot in the scale of 1:18 compared to a typical passenger vehicle. Its length is 220 mm, its width 107 mm, its height 70 mm, its wheelbase $L = 150$ mm and its weight is 500 g. The vehicle has a maximum speed of 3.7 m/s. The power consumption in standby (without steering or acceleration) is 250 mW. In experiments, the battery powers the car for about five hours. Table 2 lists the components used in the model-scale vehicle. The cost calculation refers to an order of 20 vehicles, as a single PCB would cost 45 €, but ordering a panel cluster with 20 PCBs on one board reduces the price of a unit to 15 €. Assembling a vehicle takes one person around six hours of time.

Using an off-the-shelf mechanical platform allows for a quick start in building a networked and autonomous model-scale vehicle. We use the mechanical components from the XRAY M18 PRO LiPo. It is a 1:18 micro car that is designed for holding a battery, a servo motor for steering and a motor for propulsion. The motor drives all four wheels as the shaft is connected to each one with differentials. The minimum turning radius given by the mechanical design is approximately 0.3 m.

The vehicle's hardware architecture is illustrated in Fig. 3. A Raspberry Pi Zero W takes the role of the mid-level controller (MLC) on the vehicle. It is responsible for the communication via WLAN with the HLC, as described in section 3, and for clock synchronization using the Network Time Protocol. Additionally, the MLC fuses the sensor data to obtain accurate localization. The MLC also supplies the LLC with control inputs. This is either realized by forwarding control inputs received via WLAN, or by running a controller for trajectory following as described in the next paragraph. The tasks on the Raspberry are repeated in a frequency of 50 Hz, i.e. a time interval of 20 ms.

In order to ensure the most individual and adaptable handling of the vehicle, we designed a custom PCB connecting the components. This PCB serves as an interface between the actuators, sensors and control electronics. The PCB with its components is shown in Fig. 4. This PCB embeds an ATmega2560 microcontroller with a 16 MHz clock rate. The microcontroller functions as the LLC, reading the sensor data and applying the control inputs to the actuators. The hardware separation in MLC and LLC introduces a hierarchical architecture, which creates a hardware abstraction layer. If the MLC needs to be changed, the interface to the hardware can remain the same.

At a frequency of 50 Hz, the MLC and the LLC exchange information via Serial Peripheral Interface (SPI). The MLC provides the control inputs, while the LLC returns the sensor readings. A TXB0104 bidirectional voltage-level translator was installed for level adaptation of the SPI bus. The 3.3V SPI level of the Raspberry is converted into a 5V SPI signal for the ATmega.

The IMU is a DeboSens BNO055 and provides the required sensor data using a 9-DOF sensor. The ATmega microcontroller can retrieve this data via the two wire Inter-Integrated Circuit bus.

The motor driver board VNH5019 drives the single brushed DC motor of the vehicle via an integrated H-Bridge. The ATmega controls the motor driver via a pulse width modulation signal with a frequency of 20 kHz. A current sensing output provides the ATmega with a signal which is proportional to the current applied to the motor. The power source is a 2000 mAh lithium-ion polymer (LiPo) battery which provides a 7.4 V voltage. This voltage is directly fed to the motor driver unit. Since the Raspberry and all the other components (except the motor driver unit) are specified to 5 V or 3.3 V respectively, the voltage is reduced by an NCP1117 low-dropout voltage regulator. To protect the LiPo battery as well as the electronic components a battery protection circuit was inserted.

Three Hall-effect sensors mounted on a separate odometer board measure the motor shaft rotation. A diametrically magnetised disc magnet is attached to the motor shaft in order to make the rotational motion of the axis measurable. With this setup, it is possible to distinguish six different motor angles per rotation. The digital signals of the Hall sensors are directly transmitted to three I/Os of the ATmega, which translates the signals into a rotation count.

Four LEDs are installed on the vehicle, which are also connected to the odometer board and controlled by the ATmega. The outer three LEDs are used for positioning with an IPS, while the inner one signals the vehicle's ID.

The vehicles can operate in the two different modes (1) external control and (2) trajectory following. If a trajectory is provided to the vehicle, the MLC determines the control inputs to follow that trajectory. The trajectory is provided

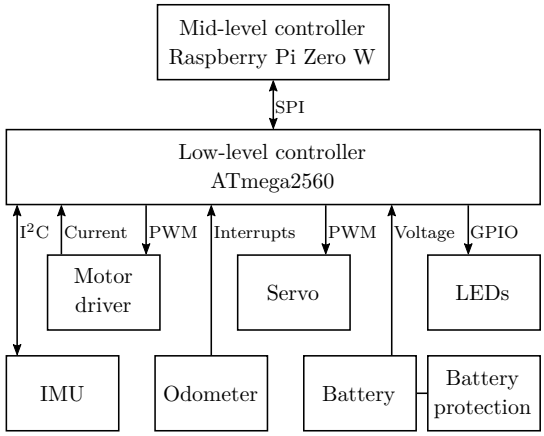

Fig. 3. Vehicle hardware architecture

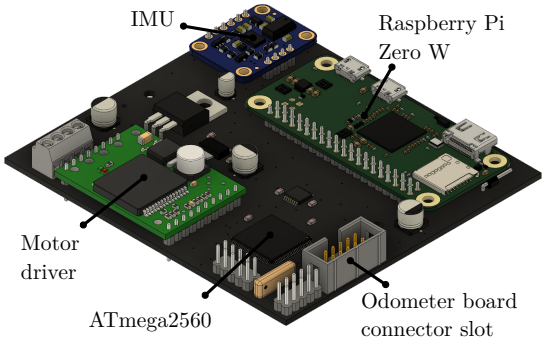

Fig. 4. The PCB on the vehicle with several components installed

as a list of tuples (t, x, y, v_x, v_y). Usually, a trajectory point is understood to be a tuple of time and position. The controller needs reference trajectory points at controller-specific points in time. If the time step between trajectory points is larger than the control time step, the MLC interpolates the trajectory to determine a sensible reference point. By fixing the derivative of the trajectory in each point, the MLC is enabled to interpolate between trajectory points with cubic Hermite splines. While it is possible to substitute a reference trajectory, further trajectory points can also be appended to an existing reference trajectory. If the MLC receives control inputs, it switches to directly applying those to the actuators. This behavior allows for manual control of the vehicle with a gamepad or a keyboard for example. It is also possible to compute control inputs depending on the vehicle state and reference trajectory externally and send those via WLAN.

5. THE VEHICLES IN CONTROL EDUCATION

The vehicle's hierarchical architecture allows students to work at different levels of abstraction.

(1) It is possible to learn the basics of embedded programming when working with the LLC (the ATmega2560). At this level, students need to understand microcontroller unit data sheets in order to determine how to read sensors and control actuators correctly in C-code.
(2) At the level of the MLC (the Raspberry Pi Zero W), tasks like trajectory control or sensor fusion can be tackled. Measurements of multiple sensors need to be fused for vehicle localization in the proposed setup, which reflects the real world application. The IPS provides absolute positioning, but its measurement data is transmitted to the vehicle via WLAN, which makes the measurements relatively slow and also unreliable. On the other hand, onboard sensors like the IMU and the odometer are fast and accurate for short distances, but need a reference. A controller for trajectory following can be implemented as simple as a PID-controller, or more advanced as a model predictive control (MPC). The µCar currently uses MPC for trajectory following. Restrictions by the limited computation power of the MLC still apply, which motivates efficient algorithms and a programming language like C++.
(3) On the highest abstraction level, ideas can be developed on an external PC with programming languages common in optimization (e.g. MATLAB, Python). It is possible to work on trajectory planners as well as on external controllers for the vehicles, depending on which mode of operation one wishes to use.

The modularity allows to focus on one specific part of networked and autonomous vehicles. It is possible to provide necessary interfaces with working components, so the content to be taught can be chosen freely and appropriately.

A system model is the prerequisite of many aspects in control, e.g. simulation or controller design. The purpose of the model defines its requirements. For simulation, the goal might be to represent the system as truthfully as possible, while for a controller using MPC the ability for fast computation might be necessary. In the following, we give an example of how a dynamical system model for the model-scale vehicles can be obtained. The goal of this endeavor is to illustrate how the vehicles might serve as a platform to control engineering education.

5.1 Vehicle dynamics model

In this example, we aim for a model that is suitable for MPC of a vehicle's pose and velocity on embedded hardware. The model needs to be simple enough for quick computation, while accurate enough for predicting the states. We propose a kinematic bicycle model with some added terms to account for various errors.

The model has the states $\boldsymbol{x}(t) \in \mathbb{R}^4$ and inputs $\boldsymbol{u}(t) \in \mathbb{R}^2$

$$\begin{aligned} \boldsymbol{x}(t) &= \begin{pmatrix} x(t) & y(t) & \psi(t) & v(t) \end{pmatrix}^{\mathrm{T}}, \\ \boldsymbol{u}(t) &= \begin{pmatrix} m(t) & d(t) \end{pmatrix}^{\mathrm{T}}, \end{aligned} \tag{1}$$

where $x(t)$ and $y(t)$ are the x- and y-position respectively, $\psi(t)$ is the yaw angle, $v(t)$ the speed at the vehicle rear axle, $m(t)$ the dimensionless motor command and $d(t)$ the dimensionless steering command. The model used to describe the vehicle's dynamics is a non-linear kinematic bicycle model according to Rajamani (2011). Similar to Alrifaee (2017), it is assumed that no slip occurs on the front and rear wheels, and no forces act on the vehicle. The velocity dynamics are described with a PT_1 behavior, which results in the following equations

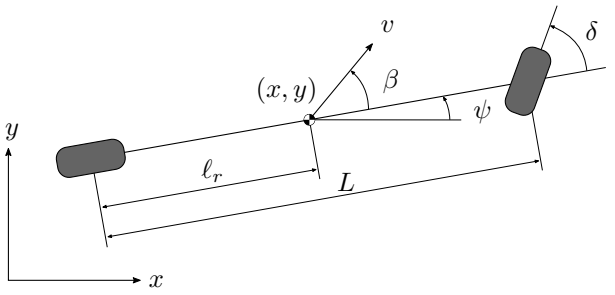

Fig. 5. Kinematic bicycle model of the vehicle

$$\begin{aligned}
\dot{x}(t) &= v(t) \cdot \cos\big(\psi(t) + \beta(t)\big),\\
\dot{y}(t) &= v(t) \cdot \sin\big(\psi(t) + \beta(t)\big),\\
\dot{\psi}(t) &= v(t) \cdot \frac{1}{L} \cdot \tan\delta(t),\\
\dot{v}(t) &= -\frac{1}{T_v} \cdot v(t) + \frac{K_v}{T_v} \cdot v_{\text{in}}\big(m(t)\big),\\
\beta(t) &= \tan^{-1}\left(\frac{\ell_r}{L}\tan\delta(t)\right).
\end{aligned} \tag{2}$$

The model variables are illustrated in Fig. 5. The time t is omitted in the figure and in the following for the sake of readability. ℓ_r is the distance from the rear axle to the vehicle's reference point, L is the distance between front and rear axle, δ is the steering angle which is related to the steering command d, K_v and T_v are the gain and time constant of the velocity's PT_1 behavior and v_{in} is the input velocity, which is modelled as a function of the motor command m. The change of the vehicle's x- and y-position is dependent on the velocity v at the vehicle's reference point. The odometer measures the motor shaft rotation, which corresponds to the velocity at a certain point A along the roll axis of the vehicle. From the fact that the angular velocity $\dot{\psi}$ is equal at every point of the vehicle, we get

$$\dot{\psi} = \frac{v}{R} = \frac{v_A}{R_A}, \tag{3}$$

where R is the radius of the circular movement at the vehicle center, R_A is the radius of the circular movement at an arbitrary point A along the roll axis and v_A is the corresponding velocity. With (3), Pythagoras' theorem and the distance ℓ_A from the rear axle to point A we obtain

$$v = v_A \cdot \sqrt{\frac{1 + \left(\frac{\ell_r}{L} \cdot \tan\delta\right)^2}{1 + \left(\frac{\ell_A}{L} \cdot \tan\delta\right)^2}}. \tag{4}$$

In order to simplify computational tasks on the model, we can approximate some terms with Taylor series at the point $\delta = 0$. The side slip angle β due to steering is approximated with a first-order Taylor series

$$\beta(\delta) = \frac{\ell_r}{L} \cdot \delta + \mathcal{O}(\delta^3). \tag{5}$$

Equation (4) is simplified with a second-order Taylor approximation:

$$v = v_A \cdot \left(1 + \frac{\ell_r^2 - \ell_A^2}{L^2} \cdot \delta^2 + \mathcal{O}(\delta^4)\right). \tag{6}$$

Now substituting the model's variables with parameters and introducing some parameters to account for various inaccuracies, the parameterized bicycle model is given by:

$$\begin{aligned}
\dot{x} &= p_1 \cdot v \cdot \big(1 + p_2 \cdot (d + p_8)^2\big)\\
&\quad \cdot \cos\big(\psi + p_3 \cdot (d + p_8) + p_9\big),\\
\dot{y} &= p_1 \cdot v \cdot \big(1 + p_2 \cdot (d + p_8)^2\big)\\
&\quad \cdot \sin\big(\psi + p_3 \cdot (d + p_8) + p_9\big),\\
\dot{\psi} &= p_4 \cdot v \cdot (d + p_8),\\
\dot{v} &= p_5 \cdot v + p_6 \cdot \operatorname{sign}(m) \cdot |m|^{p_7}.
\end{aligned} \tag{7}$$

An extra parameters introduced is p_1, which compensates the calibration error between IPS speed and odometer speed. p_2 and p_3 substitute the model parameters in (6) and (5) respectively. p_4 contains the model parameter $\frac{1}{L}$ as well as the conversion of steering command to steering angle. p_5 substitutes $-\frac{1}{T_v}$ in the velocity's PT_1 model. The steady state velocity is modeled as a power function, where the constant factor is represented by p_6 and the exponent by p_7. Since the exponential function is defined for positive real bases, the absolute value of the motor command m is used as the base and the sign of m is multiplied. p_8 is an extra parameter introduced to correct steering misalignment, while p_9 accounts for a yaw calibration error in the IPS.

This is an end-to-end, grey-box model for the vehicle dynamics. The model parameters are not measured directly, but optimized to best fit the vehicle behavior as shown in section 5.3.

5.2 Model discretization

The model is discretized with the explicit Euler method, as follows:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta t \cdot f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}). \tag{8}$$

Here, f is obtained from the continuous vehicle dynamics model (7). This discretization is chosen for its simplicity and computational efficiency. Measurements are taken in time intervals of $\Delta t = 0.02\,\text{s}$. This short time interval compensates partly for the inaccuracies introduced by the first order Euler method. This discretization is also used during the parameter identification.

5.3 Parameter identification

Since the dynamics of nonholonomic vehicles are nonlinear, model identification procedures for nonlinear systems need to be used. Identifying the vehicle dynamics can be achieved by formulating the task as an optimal parameter estimation problem. The optimization tries to find a set of model parameters that best reproduce the measurement data. A measurement vector at timestep k contains:

$$\hat{\boldsymbol{x}}_k = (\hat{x}_k \quad \hat{y}_k \quad \hat{\psi}_k \quad \hat{v}_k)^{\mathrm{T}}. \tag{9}$$

Here, $\hat{x}$ and $\hat{y}$ is the IPS x- and y-position respectively, $\hat{\psi}$ is the IPS yaw angle and $\hat{v}$ the odometer speed.

The optimization problem is then given as

$$\begin{aligned}
\underset{\boldsymbol{x}_k^j, \boldsymbol{p}}{\text{minimize}} \quad & \sum_{j=1}^{n_{\text{experiments}}} \sum_{k=0}^{n_{\text{timesteps}}} E(\boldsymbol{x}_k^j - \hat{\boldsymbol{x}}_k^j)\\
\text{subject to} \quad & \boldsymbol{x}_{k+1}^j = \boldsymbol{x}_k^j + \Delta t \cdot f(\boldsymbol{x}_k^j, \boldsymbol{u}_k^j, \boldsymbol{p})\\
& k = 0, \ldots, (n_{\text{timesteps}} - 1)\\
& j = 1, \ldots, n_{\text{experiments}},
\end{aligned} \tag{10}$$

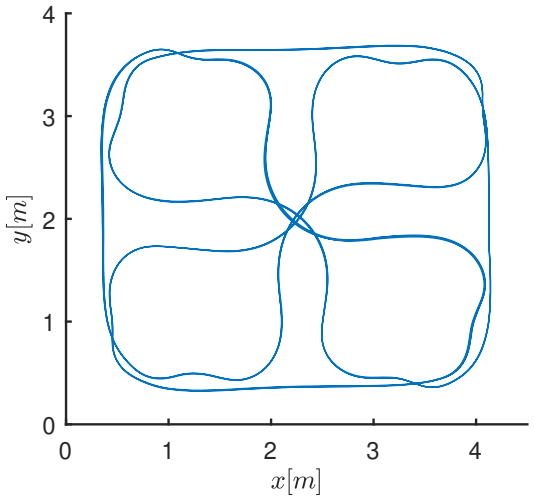

Fig. 6. Driven trajectory for measurement data collection

where $\boldsymbol{u}_k^j$ are the known inputs, f is the discrete vehicle model as in (8), $\boldsymbol{p}$ is the vector of model parameters p_1 to p_9, Δt is a constant timestep of 0.02 s and E is the error penalty function. Since the vehicle pose lives in $SO(2)$, an adequate error metric needs to be used. We used a weighted quadratic error function and account for the period of 2π in the yaw error function using $\sin^2(\Delta\psi/2)$.

This kind of optimization problem is not well suited for identifying the delay times. The optimization problem is therefore solved multiple times for combinations of delay times in an outer loop. The delays that create the lowest objective value are taken as the solution.

The measurement data used in the parameter optimization is shown in Fig. 6. This data is sliced into parts of 100 consecutive data points, i.e. time intervals of 2 s, which are fed to the optimization problem as experiments. The resulting parameters are

$$\begin{aligned}\boldsymbol{p} = (\ & 1.00 \quad -0.12 \quad 0.21 \quad 3.56 \\ & -1.42 \quad 6.90 \quad 1.34 \quad 0.03 \quad -0.01).\end{aligned} \tag{11}$$

The delays identified are 1 timestep for the IPS data, the local measurement information and the motor actuation, and 8 timesteps for the steering actuation.

6. CONCLUSION

This paper presented how a regular RC race car can be transformed to a networked and autonomous vehicle with mainly off-the-shelf components. The vehicles are used for teaching in multiple courses at RWTH Aachen University at the moment. We are eager to see the impact of applying concepts on real control systems on the students' learning experience.

Currently, a fleet of 20 vehicles is being built up. This should enable students and researchers alike to perform various experiments on networked and autonomous driving in moderately large scale networked systems.

REFERENCES

Alrifaee, B. (2017). *Networked Model Predictive Control for Vehicle Collision Avoidance*. Ph.D. thesis, RWTH Aachen University.

Gonzales, J., Zhang, F., Li, K., and Borrelli, F. (2016). Autonomous drifting with onboard sensors. In *Proceedings of the 13th International Symposium on Advanced Vehicle Control (AVEC)*.

Hyldmar, N., He, Y., and Prorok, A. (2019). A fleet of miniature cars for experiments in cooperative driving. *arXiv e-prints*, arXiv:1902.06133.

Karaman, S., Anders, A., Boulet, M., Connor, J., Gregson, K., Guerra, W., Guldner, O., Mohamoud, M., Plancher, B., Shin, R., et al. (2017). Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at mit. In *Integrated STEM Education Conference (ISEC)*. IEEE.

Kloock, M., Maczijewski, J., Scheffe, P., Kampmann, A., Mokhtarian, A., Kowalewski, S., and Alrifaee, B. (2020a). Cyber-physical mobility lab an open-source platform for networked and autonomous vehicles. *arXiv preprint arXiv:2004.10063*.

Kloock, M., Scheffe, P., Tülleners, I., Maczijewski, J., Kowalewski, S., and Alrifaee, B. (2020b). Vision-based real-time indoor positioning system for multiple vehicles. *IFAC-PapersOnLine*.

Liniger, A., Domahidi, A., and Morari, M. (2014). Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*. doi: 10.1002/oca.2123.

Naumann, M., Poggenhans, F., Lauer, M., and Stiller, C. (2018). Coincar-sim: An open-source simulation framework for cooperatively interacting automobiles. In *Intelligent Vehicles Symposium (IV)*. doi:10.1109/IVS.2018.8500405.

O'Kelly, M., Sukhil, V., Abbas, H., Harkins, J., Kao, C., Vardhan Pant, Y., Mangharam, R., Agarwal, D., Behl, M., Burgio, P., and Bertogna, M. (2019). F1/10: An Open-Source Autonomous Cyber-Physical Platform. *arXiv e-prints*, arXiv:1901.08567.

Paull, L., Tani, J., Ahn, H., Alonso-Mora, J., Carlone, L., Cap, M., Chen, Y.F., Choi, C., Dusek, J., Fang, Y., Hoehener, D., Liu, S.Y., Novitzky, M., and Franzoni, I. (2017). Duckietown: an open, inexpensive and flexible platform for autonomy education and research. In *International Conference on Robotics and Automation (ICRA)*. IEEE.

Rajamani, R. (2011). *Vehicle dynamics and control*. Springer Science & Business Media.

Reiter, M., Alrifaee, B., and Abel, D. (2014). Model scale experimental vehicle as test platform for autonomous driving applications. In *FISITA World Automotive Congress*.

Reiter, M., Wehr, M., Sehr, F., Trzuskowsky, A., Taborsky, R., and Abel, D. (2017). The IRT-buggy – vehicle platform for research and education. *IFAC-PapersOnLine*. doi:10.1016/j.ifacol.2017.08.2200.

Williams, G., Drews, P., Goldfain, B., Rehg, J.M., and Theodorou, E.A. (2016). Aggressive driving with model predictive path integral control. In *International Conference on Robotics and Automation (ICRA)*. IEEE.