Vehicular platooning experiments using autonomous slot cars [⋆]

Martin Lád [*] **Ivo Herman** [*] **Zdeněk Hurák** [*]

[*] *Faculty of Electrical Engineering, Czech Technical University in Prague. e-mail: (ladmarti@fel.cvut.cz, ivo.herman@fel.cvut.cz, hurak@fel.cvut.cz)*

Abstract: The paper reports on an affordable experimental platform for vehicular platooning. The experimental platoon consists of several autonomous slot cars (typical experiments take 5 to 20 slot cars), hence it fits into an indoor laboratory. Each car is equipped with an onboard controller and it can measure its own velocity, acceleration, and distances to its nearest neighbors. Furthermore, each car can communicate with other vehicles including the leader of the platoon. A convenient user interface allows to store, analyze and visualize the experimental data in Matlab. The platform can be used for demonstrating various decentralized and distributed control strategies for vehicular platoons, such as predecessor following, (a)symmetric bidirectional control or cooperative adaptive cruise control. Moreover, the phenomenon of string instability can be observed in experiments due to the fast dynamics of slot cars. The technical design details including the source codes and electronic schematics are shared with the public.

Keywords: Distributed control, vehicular platoons, predecessor following, bidirectional control.

1. INTRODUCTION

Distributed control of platoons of vehicles has been an active research topic for a few decades. Both theoretical and technological aspects have been investigated. What motivates this research are some promising properties of platoons, such as reduced fuel consumption, increased transportation capacity and safety. The research in platooning has accelerated recently with the advent of driverless vehicles.

Although many dozens if not hundreds of theoretical papers have been written on the topic of vehicular platoons, descriptions of experimental verifications are disproportionally rare in the literature. Among the projects/programs/activities with experimental verifications, the University of Berkley's PATH program was apparently the first, having started in late 1980s, see Milanes et al. (2014). In Europe, the EU funded research project named SARTRE, which included Volvo company among its industrial partners, was succesfully accomplised in 2012 by demonstrating a few driverless rides of a five-vehicle platoon, see Coelingh and Solyom (2012). Truck platooning was also demonstrated by KTH in Stockholm and Scania, see Alam et al. (2015). In the Netherlands, the experimental platooning was demonstrated in Eindhoven, see Naus et al. (2010).

Dominant control strategy in these experimental full-scale projects was based on a *time headway* and a *communication* of the states of a few preceding vehicles. On the other hand, there are other control strategies described in the literature. One research direction is the *fixed-distance control*. Examples are *predecessor following* analyzed in detail by Seiler et al. (2004), *symmetric bidirectional control* investigated by Barooah and Hespanha (2005), *asymmetric bidirectional control* examined by Barooah et al. (2009) and control with *different asymmetries in each state* elaborated on by Herman et al. (2016b). Except for predecessor following, we are not aware of any experimental verification of these control architectures. One of the reason for this absence of experiments is that real vehicles are usually not equipped with a rear-distance measuring device, which is required for bidirectional control architectures. Adding these extra sensors, as well as actually planning and conducting experiments with real vehicular platoons, may turn out unaffordable for most academic research teams. And yet the experiments are badly needed to keep the theoretical work well motivated.

And here comes our work. Our goal is to provide an affordable experimental platform for testing distributed control algorithms for vehicular platforms. Being based on toy-size racing slot cars, experiments with more than 10 cars are easily feasible even in a small lab/office. And yet the fast dynamics of the cars and the generous onboard computational power give the experiments a flavour of reality. Furthermore, since there is no need to be worried about the consequences of crashes between the cars, the platform can serve also as an educational tool.

The experimental platform is based on racing slot cars produced by *Carrera* used commonly as toys by kids and adults. Originally, these slot cars have no onboard controller. They are also free of any sensor or communication interface. Their velocity is controlled remotely by a human player varying the voltage on the conducting strips on the track. We upgraded the slot cars significantly by equipping them with custom-made electronics including powerful microprocessors, sensors and communication interface.

[⋆] I.H. was supported by the Czech Science Foundation within the project GACR 16-19526S.

The main computing unit is the popular and widespread *Raspberry Pi Compute Module*, which runs Debian Linux. Everything still fits within the small car, so the appearance is (almost) unchanged—see Fig. 1.

Fig. 1. Platoon of autonomous slot cars.

The main features of the platform are

- Small size: each car is some 13 cm in length and 4 cm in height. The simplest track required for experiments is 1 meter in diameter, so it easily fits an office desk.
- Low cost: the total cost of each autonomous car is approximately 300 €, including the bare slot car itself. The basic track costs about 70 €.
- Durability: since all the electronics is located inside the plastic cover of the slot car, it can easily withstand almost any crash between cars.
- Ease of controller design: the overall software architecture is such that a controller is a just plugin for the main application in the car. Hence it can be easily developed, deployed and run. Moreover, source codes for the most commonly used controllers (PI, PD, . . .) are already provided.
- Graphical application for the platoon setup: a Java application is provided for the platoon management. It allows to select the type of a controller, set its parameters and upload the controller to the onboard unit. It also allows to update the firmware in the cars conveniently.
- Matlab interface: the data measured by sensors at each car are sent to the human operator's PC running Matlab. Matlab can be used not only to visualize and analyze the measured data but it can also be used to issue commands to the platoon, such as a change in the leading vehicle's desired velocity.

The platform is being developed within numerous student projects following the spirit of *open source* and *open hardware*. The code and the schematics are publicly available through a Git repository `https://gitlab.fel.cvut.cz/SlotcarPlatooning`.

A previous version of the experimental platform has already been reported to the community by Martinec et al. (2012). The current report reflects major design changes which now make the platform more scalable and also reproducible by other teams.

This paper is structured as follows. In the next section we introduce the basic hardware components of the car and in the third section we describe the software. The fourth section is dedicated to modelling of dynamics of a slot car and description of a basic control architecture. In the fifth section we present some experimental results, showing the ability of the experimental platform to reveal some basic properties of vehicle platoons. Concluding remarks are presented afterwards.

2. SLOT CAR HARDWARE

We begin by the description of the overall hardware configuration, see Fig. 2. The whole experimental platform consists of the individual slot cars, the track set, and, finally, a standard PC with Wi-Fi.

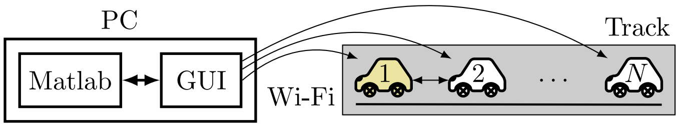


Fig. 2. Overall structure of the system. Although not shown, the cars also communicate among each other.

Each autonomous slot car is based on a 1:32 model of Ford Capri by Carrera. The drivetrain is very simple: a permanent magnet DC brushed motor which drives the rear axle through a 3:1 gear. No differential is present and the wheels are on a common shaft.

The car is kept on the track thanks to a guiding slot into which a pin or blade extends from the bottom of the car. This means that the car can move only forward or backward. The power supply is taken from the pair of metal strips along the slot. Since the side motion is severely restricted, the platform is only suitable for experiments related to longitudinal dynamics and control.

2.1 Electronics

We equipped the car with custom-made electronics, sketch of which is shown in Fig. 3.

The supply (rail) voltage ranges from 9 V to 15 V. We normally use 12 V. This is the voltage at which the experimental identification was conducted (see Sec. 4). There are several step-down voltage converters connected in a cascade: 12 V → 5 V → 3.3 V → 1.8 V.

For a short amount of time, the car may loose a connection with the power line, for instance, due to crash between cars, which would lead to a loss of data. This issue is solved by adding a supercapacitor (capacity 1 F; 5 V) which serves as a power backup for the 5 V and lower branches. This guarantees a power backup for cca 5 s, which seems to be enough for most emergency situations. Note that since the car has a significant power consumption, a regular capacitor is not sufficient for this task. Another option was to add a battery. However, a battery needs a regular maintenance, which is not needed for the super capacitor.

There are two main processing units: i) ARM Cortex M4 STM32 microcontroller (STM) and ii) Raspberry Pi Compute Module (RPi). The STM processor collects and processes the data from the onboard sensors and realized velocity-control loop, while the RPi module realizes the higher-level control loops and communication.

The motor is powered directly from the 12 V power branch. It is driven through an H-bridge motor driver by Texas

Instruments. The driving signal is generated by the STM and the H-bridge allows the STM to change the direction and the speed of rotation. The speed is controlled by generating a PWM (Pulse Width Modulation). Therefore any desired speed is achievable (limited by the power supply).

The communication interface is a standard USB Wi-Fi (IEEE 802.11g) module, connected to the RPi. It allows the cars to communicate with each other and with the PC.

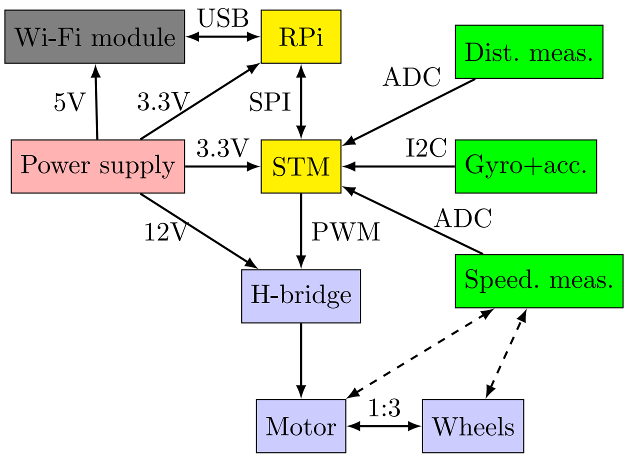


Fig. 3. Overall configuration of the electronics.

2.2 Sensors

Each car is equipped with several sensors. The quantities measured onboard each car are the car's acceleration (in all three directions), the car's angular velocity (around all the three body axes), the vehicle's translational velocity (along the slot), the distance to the car ahead and the car behind, and, finally, also the rail voltage.

The acceleration and the angular velocity vectors are measured by a 3-axis MEMS gyroscope and accelerometer. This device is connected to STM using an I2C bus.

The translational speed is measured using two types of sensors. The first one is a standard (albeit home-made) incremental rotary encoder (IRC). The binary (black and white) disk is connected directly to the rear-axle shaft, and light pulses reflected from the disk are counted. Since there is only one encoder, it is not possible to obtain the direction of the rotation. The other mean of speed measurement is based on measuring the back-EMF (back-induced electromotoric force), denoted v_e, which is proportional to the motor angular speed as $v_e = k\omega$, where k is the so-called back-EMF constant (in SI units identical to the motor torque constant). The voltage is measured using the integrated analog-to-digital converter (ADC) on the microcontroller. Unlike the IRC, the back-EMF measurements also give a sign of the angular velocity. Since we have two measurements, each with a different precision and accuracy, they can be fused using an estimator, namely Kalman filter. This filter also provides an estimate of the friction force.

For distance measurements, we could not find any commercially available sensor which would satisfy the following requirements:

- small size, such that it fits to the car,
- wide field of view (important when going through turnings),
- no interference between the front and rear distance sensor.

It was possible to satisfy the first two requirements, however the third one seemed as a big obstacle. That is why we designed our own sensor. It is based on an infrared (IR) LED diode and a phototransistor. The diode emits square pulses with a given frequency and the phototransistor receives the signal. To get rid of disturbances such as the sunshine or the room lighting, the demodulation of the received signal exploits the correlation (synchronous detection)—the frequency of the transmitted signal is known. There are distinct frequencies for both front and the rear distance measurements: for the front it is $f_\mathrm{f} = 1111\,\mathrm{Hz}$ and for rear it is $f_\mathrm{r} = 1666\,\mathrm{Hz}$. Both are chosen to be sufficiently well separated and also not being the multiplies (higher harmonics) of 100 Hz, which is a frequency of the fluorescent tubes used in the lab.

In order to get rid of the dependence on the angle of reflection, each car receives the signal emitted by its two neighbors—the car ahead and the car behind. This gives a good precision, since we used wide angle IR diodes and phototransistors. The sketch of the distance measurement is in Fig. 4. The car with index $i-1$ emits the signal from its rear LED with the frequency f_f. The car i detects this signal at its front transistor and demodulates it. From the signal strength of the demodulated signal it calculates the distance. The inter-vehicle distance d is then obtained from the demodulated signal strength s as

$$d = \sqrt{c/s}, \tag{1}$$

where c is a constant obtained from calibration. The range of the sensor is approximately 70cm. Similarly, the car $i-1$ calculates its distance to the car i. Compared to the distance measurement schemes based on reflections (some commercially available hobby-grade distance sensors rely on it), the proposed scheme achieves 4 times stronger signal (the distance for the light to travel is halved).

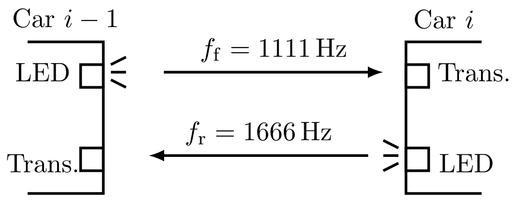

Car $i-1$
Car i
LED
$f_\mathrm{f} = 1111\,\mathrm{Hz}$
Trans.
$f_\mathrm{r} = 1666\,\mathrm{Hz}$
Trans.
LED

Fig. 4. Distance measurement using LEDs and phototransistors.

The disadvantage of this solution is that it is required to have a car ahead, transmitting the modulated signal. However, this does not cause any problems in our setting, since there is always a platoon leader. The leader drives independently, following a desired speed profile. Hence the leader itself does not need any front distance measurement.

3. SOFTWARE

The software splits into three main parts, corresponding to the three processors on which it runs: the STM microprocessor, the Raspberry Pi Linux computer and the PC. In this section we describe the architecture of these software parts.

3.1 Firmware running on the STM microprocessor

The main purpose of the code running on the STM (which we will call *firmware* here) is to read off the data measured by the sensors (pulses from the IRC, back-EMF voltage, angular rates and accelerations from a MEMS inertial multi-sensor, distance to the car ahead and behind, voltage on the conducting strips), preprocess them (distance measurement demodulation, fusion of speed sensors using Kalman filter), close some fast low-level feedback loops (speed of the slot car) and provide some of the measurements to the higher level controller running on the RPi computer. These tasks require very high sampling frequencies (for instance, for the distance demodulation the frequency is 60 kHz), which would be difficult to achieve with the RPi computer running Linux.

On the other hand, thanks to a built-in SPI bootloader on the STM microprocessor, firmware can be conveniently uploaded to the STM processor from the RPi computer.

The firmware for the STM is written in C language. It is relatively simple and will not require any modification by a regular user, unless there are some specific requirements.

3.2 Software running on Raspberry Pi computer

The main control-related role of the code running on the slot car onboard Raspberry Pi computer is to execute the *distance controller*. In order to accomplish this, the computer has to realize the communication with other slot cars and the PC.

The controller accepts as its inputs the full states of all the vehicles (obtained by wireless communication). Based on these inputs it calculates the reference (desired) speed for the speed controller. Alternatively, the controller can also set the duty cycle for the PWM voltage applied to the motor.

The choice of Raspberry Pi computer and Java language Raspberry Pi computer runs a light operating system—a Linux distribution called *Raspbian* (based on the popular Debian distribution, as the name reflects). Having an operating system allows the user to use many convenient services such as SSH for easy maintenance and debugging, FTP for firmware uploading and log files downloading, etc.

The code to be run on board of each slot car is written in Java. One reason behind this choice was to make the solution multi-platform (the code written for the slot car can be reused in the PC application). Another reason for this choice is a wide range of available libraries. For instance a library for communication (TCP/UDP stack), object serialization or file management. The RPi application periodically polls the STM processor for the measured data and communicates the control effort to it.

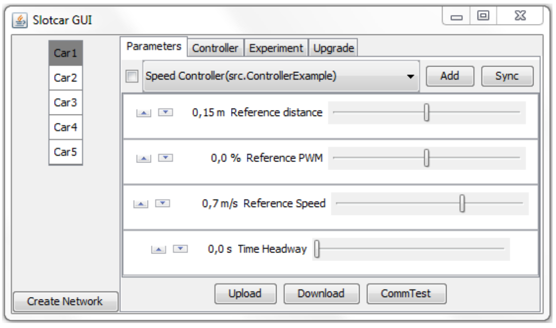
Although we do not use a real-time version of Linux, it appears that the operating system is capable of running the controller task with a sampling period of $T = 30\,\text{ms}$. However, we confess that have not conducted any systematic timing analysis yet.

Implementation of the inter-vehicle distance controller as a plugin The distance controller is implemented as a plugin. This gives an important advantage: with the structure of basic controller fairly simple, even a person with little programming experience can quickly implement his or her own controller. Being decoupled from the development of the main application, the control designer is freed from the need to understand the complicated infrastructure. In fact, the code for the controller looks similar to a Matlab S-function, with which many control engineers and students are familiar. A code snippet for a simple controller is given in Appendix A.

3.3 Graphical User Interface running on PC

The highest-level managment tasks for the platooning experiments are accomplished using a graphical user interface (GUI) running on a PC. The application allows to choose a type of a controller, set its parameters, set the reference values for each car, upload the controller to the car, as well as to start and stop the experiment. On the top of that, the applicatin allows to upload the cars' firmware (using FTP and SSH), to reset the onboard computer/controller and a few more system operations.

The GUI acts as a server, to which the clients (cars) connect. Each car, after booting up, connects to this server. After a car connects, the GUI updates the list of all connected cars and shares this list with the other cars. Once the car connects to the GUI, the application retrieves the car's setting—the control parameters (type of controller, desired distance, controller parameters). Then the user can modify the parameters and upload them back to the car. A screenshot of basic parameter setting using the GUI is in Fig. 5.



Speed Controller(src.ControllerExample)
0,15 m Reference distance
0,0 % Reference PWM
0,7 m/s Reference Speed
0,0 s Time Headway

Fig. 5. Screenshot of the GUI. On the left there is a list of connected cars.

The GUI has been mainly developed and tested on Windows. However, thanks to multi-platformness of Java, it also works in Linux.

We also implemented a Simulink model, which displays the states in the platoon and allows to change some control parameters of the platoon during the experiment. The Simulink model reads the data from the GUI and sends back the commands. Hence, it requires the GUI to get data. This solution enabled keeping the Simulink model very simple and did not require adding an extra code to the cars' software.

Ordering of the cars in the platoon In case the control algorithm requires a knowledge of the order of the cars in the platoon, an ordering procedure can be initiated using the GUI. The procedure is based on a moving the cars one by one. Starting off with a leader, the car moves ahead a given distance, which triggers a detection of a change of the inter-vehicle distance by some car, which then transmits its recognized order. The procedure repeats until the order of all cars is known.

3.4 Communication among the cars and with the GUI

The cars and the GUI share the same communication protocol, which is the popular User Datagram Protocol (UDP). The cars periodically broadcast their states. These are then received by all the other cars and by the GUI. The order, in which the cars broadcast, is based on the order of vehicles in the platoon (the leader starts). If a vehicle receives a packet from the preceding vehicle, it can share its own state—a *token ring policy* is implemented.

On the top of this token ring communication, a simple addressed (peer-to-peer) communication was implemented, which is only used between the GUI and the cars for managing the platoon configuration.

4. MODELLING AND CONTROL

4.1 A control-oriented model of a slot car

A slot-car is basically just a loaded brush-type permanent magnet DC motor. The total mass of the slot car (including the mass of the motor) constitutes the load which is steered along a trajectory determined by the slots. The equations can be written down almost by inspection or using the power bond graph in Fig. 6

$$\frac{\mathrm{d}i(t)}{\mathrm{d}t} = \frac{u(t)}{L} - \frac{R}{L}i(t) - \frac{k}{rnL}v(t) \tag{2}$$

$$\frac{\mathrm{d}v(t)}{\mathrm{d}t} = \frac{k}{mnr}i(t) - \frac{b}{m}v(t) \tag{3}$$

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = v(t). \tag{4}$$

The states are: the current i through the motor winding (in A), the velocity v of the car (in m/s), and the travelled distance x (in m). The control input is the voltage u (in V). In fact, the voltage signal comes in the pulse width modulated (PWM) form with the duty cycle D, where $D \in [0, 1]$. Hence u denotes just the low-frequency content. The rail voltage $U_\mathrm{r} = 12\,\mathrm{V}$ and $u(t) = U_\mathrm{r}\,D(t)$. The relevant physical parameters are in Table 1.

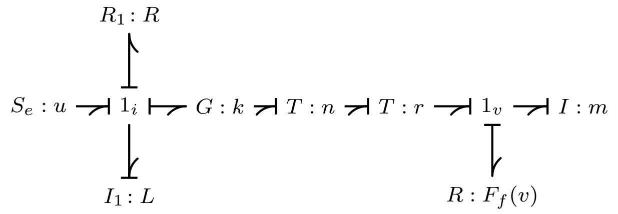


Fig. 6. Bond graph of the one-dimensional electromechanical dynamics of a slot-car as a loaded permanent magnet DC motor with a permanent magnet.

Table 1. Parameters for the vehicle model.

Physical parameter	Symbol	Value
Resistance of the motor winding	R	4.9Ω
Inductance of the motor winding	L	$2\,\mathrm{mH}$
Torque constant	k	$0.005\,\mathrm{Nm\,A^{-1}}$
Mass of the slot car	m	$0.4\,\mathrm{kg}$
Friction coefficient (linear friction model)	b	$0.25\,\mathrm{Nm^{-1}\,s}$
Radius of the wheel	r	$0.01\mathrm{m}$
Gear ratio	n	$1/3$

Note that while the linear model turns out nearly perfect for most aspects here, it fails badly when describing the friction phenomenon. Here the friction comes from three sources—friction induced by the angular motion of the rotor shaft, friction in the slot, and the rolling friction. It is well known that the rolling friction does not depend on the velocity but only depends on the normal force (here it is not only the weight but also the attractive force of magnets that push the slot car against the track). Introducing some nonlinearity into the model seems inevitable. But temporarily, the linear model of a friction is used to get a transfer function as rough models of the overall dynamics.

The electric current dynamics is very fast compared to the mechanical dynamics of the velocity, so we can neglect it by setting $\frac{\mathrm{d}i(t)}{\mathrm{d}t} = 0$ in (2). Separating $i(t)$ and plugging it to (3), we get

$$\frac{\mathrm{d}v(t)}{\mathrm{d}t} = \frac{U_\mathrm{r}k}{Rmnr}d(t) - \frac{b}{m}v(t) - \frac{k^2}{Rmr^2n^2}v(t) \tag{5}$$

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = v(t). \tag{6}$$

Transfer functions from the input voltage are

$$G(s) = \frac{v(s)}{D(s)} = \frac{\frac{kU_\mathrm{r}}{Rmnr}}{s + \frac{b}{m} + \frac{k^2}{Rmr^2n^2}} \tag{7}$$

$$H(s) = \frac{x(s)}{D(s)} = \frac{1}{s}G(s). \tag{8}$$

After plugging in the values of the parameters,

$$G(s) = \frac{5.28}{0.58s + 1}. \tag{9}$$

Using an identification from a step response (see Fig. 7), the transfer function was corrected to

$$G(s) = \frac{5.1}{0.58s + 1} \tag{10}$$

In experiments it was observed that there is a very large static (dry, Coulomb) friction, which amounts up to 28% of the torque (hence force) corresponding to the maximum PWM duty cycle. A simple static friction model comes in the form of a *dead-zone* with the range $[-0.28, 0.28]$ appended to the controller.

4.2 Controllers

The control architecture is in Fig. 8. It exhibits a cascade structure—the innermost loop takes care of tracking the desired speed while the outer loop takes care of tracking the desired inter-vehicle distances.

Speed controller The speed controller is implemented in the STM processor. It is a discrete PI controller with a

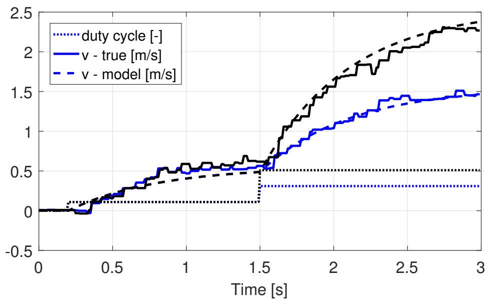


Fig. 7. Comparison of true velocity and output of the model (10). Two different steps at time 1.5 s are shown. The value of the dead-zone was already subtracted from the input d.

sampling period $T_{\mathrm{s,v}} = 0.005\,\mathrm{s}$. Its output, which is the PWM signal duty cycle, is saturated to $[-1, 1]$. The controller implements an anti-windup in the form of clamping. The transfer function of the controller is

$$C_v(z) = \frac{z - 0.975}{z - 1}. \tag{11}$$

Distance controller The distance controller is implemented in the RPi computer. It can take many forms. In this demonstration paper we evaluate a simple bidirectional control law, which accepts a single input—the weighted regulation error as in

$$e = (d_i - d^{\mathrm{ref}}) + \epsilon(d_{i+1} - d^{\mathrm{ref}}), \tag{12}$$

where $d_i = x_{i-1} - x_i$ is the distance to the car ahead, $d_{i+1} = x_i - x_{i+1}$ is the distance to the car behind, d^{ref} is the reference distance the cars should keep among each other, and ϵ is a *constant of asymmetry*. This constant weighs the contribution of the rear spacing error. When $\epsilon = 0$, the controller only uses the distance to the car ahead (the so-called *predecessor following*), when $\epsilon = 1$, the car weighs the rear spacing error with the same weight as the front error (so called *symmetric bidirectional control*) and when $0 < \epsilon < 1$, we have *asymmetric bidirectional control*. The control law (12) was used in Herman et al. (2016a).

When the leader moves with a constant velocity, its position is a linearly growing signal—a ramp signal. It follows from internal model principle applied within the domain of distributed systems Wieland et al. (2011); Lunze (2012) that in order to track the leader, each car is required to have at least two integrators (two poles at the origin) in the open loop (Yadlapalli et al. (2006)). There is already one integrator in the vehicle model: the integration of velocity to position. Therefore, it is necessary to have at least one integrator in the controller. Therefore, we selected a discrete PI controller

$$C_{x,1}(z) = \frac{3z + 2.976}{z - 1}. \tag{13}$$

Two integrators in the open loop usually mean slow and oscillatory transients. When we only require tracking of the leader's velocity and the steady-state distance error can be nonzero, it suffices to have only one integrator in the open loop. An instance of such a controller is the following filtered PD controller

$$C_{x,2}(z) = \frac{15z - 7.5}{z - 0.5}. \tag{14}$$

For both controllers the sampling period was $T_{\mathrm{s,d}} = 0.03\,\mathrm{s}$. Since we can rely on a wireless communication among the vehicles, it is possible to achieve a zero steady-state distance error. The leader has to communicate its velocity $v_0(t)$ to all the cars. The communicated velocity can be added to the desired velocity produced by the feedback controller, thus implementing a feedforward controller. The control law for the ith car becomes

$$v_i^{\mathrm{ref}}(t) = v_0(t) + r_i(t), \tag{15}$$

where r_i is the output of the distance controller (14). A similar control law was used in Barooah et al. (2009); Lin et al. (2012), where the leader's velocity was called a *desired velocity of the platoon*.

5. EXPERIMENTAL VERIFICATION

In this section we give an appetizer of how the proposed experimental platform can be used as a testbed for showing some properties of distributed control for vehicular platoons.

In all experiments shown here, we were interested in the responses of the distances among cars to changes in the leader's desired velocity. The leader has a reference velocity

$$v_{\mathrm{ref}} = \begin{cases} 0.0\,\mathrm{m/s} & \text{for } t \in [0, 1]\,\mathrm{s}, \\ 0.3\,\mathrm{m/s} & \text{for } t \in [1, 19]\,\mathrm{s}, \\ 0.7\,\mathrm{m/s} & \text{for } t \in [19, 34]\,\mathrm{s}, \\ 0.3\,\mathrm{m/s} & \text{for } t \in [34, 50]\,\mathrm{s}. \end{cases} \tag{16}$$

All other cars should follow the leader with the desired distance $d^{\mathrm{ref}} = 0.15\,\mathrm{m}$.

The first experiment compares the distance controllers from the previous section in a predecessor following scenario. That is, $\epsilon = 0$. In Fig. 9a, after the initial growth of the distances, the PI controller (13) was able to achieve the desired inter-vehicle distance. On the other hand, having a PD controller in Fig. 9b, the transient was better, but the steady-state distances were about $0.25\,\mathrm{m}$ for $v_{\mathrm{ref}} = 0.3\,\mathrm{m/s}$ and $0.30\,\mathrm{m}$ for $v_{\mathrm{ref}} = 0.7\,\mathrm{m/s}$, instead of desired $0.15\,\mathrm{m}$. This confirms that the open loop with only one integrator cannot achieve a zero steady-state error. Instead, it behaves as a time headway spacing policy, where the inter-vehicle distance is proportional to velocity. When the leader's velocity was available to all the cars for feedforward (Fig. 9c), the transient was short and the achieved distance was equal to the desired one. The change in desired velocity at $t = 19\,\mathrm{s}$ and $t = 34\,\mathrm{s}$ did not have almost any influence on the error in distance. Note that there are turns on the track, which act as disturbances. That is why that around $t = 15\,\mathrm{s}, t = 25\,\mathrm{s}, t = 40\,\mathrm{s}$ there are some oscillations in the plots.

For bidirectional control the transient was longer and errors were larger too, see Fig. 10. In this figure we also show the response of the model of the platoon. Although the model still needs to be improved (mainly modelling of the friction), it captures the reality reasonably well.

For the predecessor following architecture there is an apparent string instability in Fig. 11 for $v_{\mathrm{ref}} = 0.3\,\mathrm{m/s}$. The higher the index of the car, the higher the peak in velocity. This is a consequence of having two integrators in the open loop, for which the string instability was proved in (Seiler et al., 2004, Thm. 1).

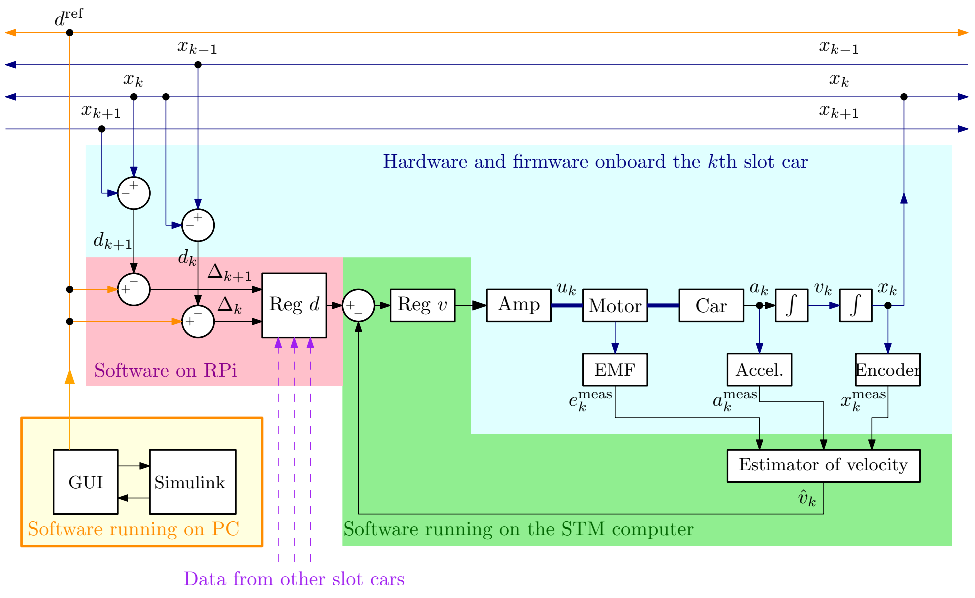
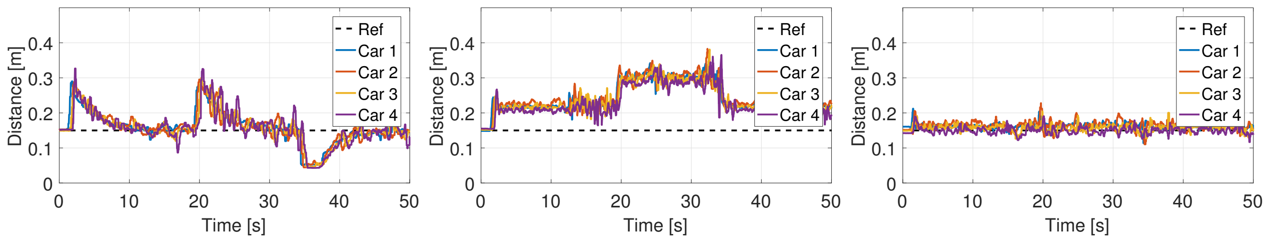


Fig. 8. Overall control system architure—cascade control structure. The distance controller implemented on the RPi can also accept some other inputs through a wireless communication with other slot cars.



(a) $C_{x,1}$ from (13) (b) $C_{x,2}$ from (14) (c) $C_{x,2}$ with feed-forward from (15)

Fig. 9. Performance of several control laws for predecessor following and changes in the leader's desired velocity (16).

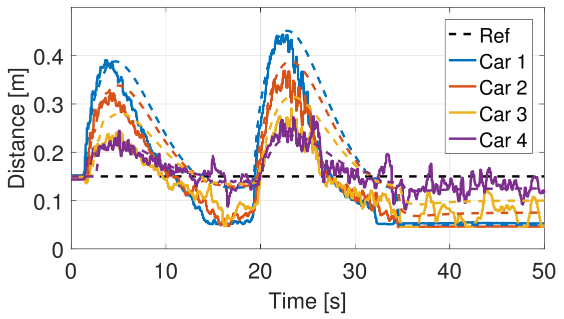


Fig. 10. Response to steps in leader's desired velocity with PI controller $C_{x,1}$ (13) and $\epsilon = 1$. The response of the mathematical model of the platoon uses dashed lines.

A short video documenting some (early) experiments with the slot car platoon described in this text is in `https://youtu.be/TBFM7v2_VAk`.

6. CONCLUSION

In this paper we described our experimental platform for demonstrating properties of distributed control systems for vehicular platoons. The slot cars are equipped with powerful onboard processors, sensors and communication interface. The car can control itself based on its local measurements but the measured/estimated states of the other cars can also be made available to it through wireless communication among the cars. The platoon is conveniently configured and controlled from Simulink. The experimental data are available for further analysis in Matlab.

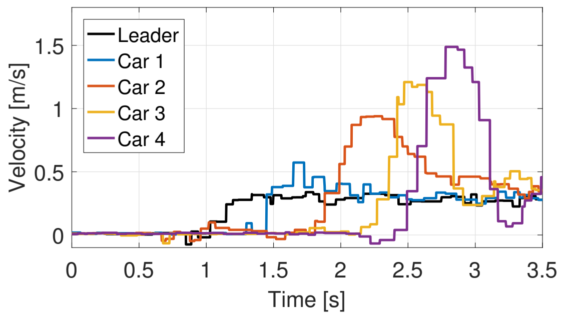


Fig. 11. Response to step in leader's desired velocity with PI controller $C_{x,1}$ (13) and $\epsilon = 0$.

The experiments shown briefly in this paper suggest that the proposed platform is suitable for experimental demonstration of some effects of distributed control of vehicular platoons. For instance, string instability is apparent for a platoon with two integrators in the open loop, while nonzero steady-state error appears when only one integrator is used. Centralized information, such as leader's velocity, improves the transient response, as expected.

The platform is under an ongoing development. The to-do list contains conducting experiments with more cars (we aim at 20 to 30), improving the speed measurement/estimation and the estimation of the frictional force. Nonetheless, we are convinced that the proposed platform at its current state can already serve in research and education. We are eager to share the details with the interested collaborators and partners. The source codes and electronics schematics are shared through a publicly accessible GIT repository.

7. ACKNOWLEDGEMENT

The authors acknowledge the help of Dan Martinec, Filip Svoboda, Filip Richter, Jan Moravec, Jaromír Dvořák, Anastasia Vlasova and Alexander Dubeň, who all contributed to the software and hardware design and implementation.

Appendix A. CONTROLLER CODE EXAMPLE

An example code in the Java language for a P controller with a feedforward from the leader's velocity is shown in Listing 1.

```java
@ClassInfo(label = "P controller with FF")
public class ControllerP_FF extends Controller {

  @FieldInfo(label="Controller P constant")
  public float kp = 5f;

  public ControllerP_FF() {
    super(OutputType.SPEED);
    init();
  }

  public long getMiliSecPeriod() {
    return 30;
  }

  public void init() {
// in this case, nothing needs to be initialized
  }

  public float step(CarImage me, CarList cars) {
    float distRef = me.getReferenceDistance();
    float distMeas = me.getDistanceFront();
    float ctrEffort = kp*(distMeas - distRef);
    float ldrSpeed = cars.getPos(0).getSpeed();
    float totalEffort = ctrEffort + ldrSpeed;
    float desVel = Nonlin.sat(totalEffort, -1f, 1f);
    return desVel;
  }
}
```

Listing 1. Code for a simple proportional controller augmented with a feedforward of the leader's velocity.

Some parts of the code are worth mentioning. Any controller parameter can be made available to be edited by the user in the GUI just by declaring the corresponding property public (here "kp"—the P controller gain). The label which is displayed in the GUI is given in the "@FieldInfo" modifier. The controller description is given by "@ClassInfo". The most important method is "step". It calculates the control effort from the measured data ("me") and data received by communication ("cars"). The leader's index in the platoon is always 0, as can be seen from the obtaining the leader's speed. The output is saturated when calculating the desired velocity for the velocity controller ("desVel"). A simple controller template is also provided at GIT.

REFERENCES

Alam, A., Mårtensson, J., and Johansson, K.H. (2015). Experimental evaluation of decentralized cooperative cruise control for heavy-duty vehicle platooning. *Control Engineering Practice*, 38, 11–25.

Barooah, P. and Hespanha, J.P. (2005). Error Amplification and Disturbance Propagation in Vehicle Strings with Decentralized Linear Control. In *Proceedings of the 44th IEEE Conference on Decision and Control*, 4964–4969. IEEE.

Barooah, P., Mehta, P.G., and Hespanha, J.P. (2009). Mistuning-Based Control Design to Improve Closed-Loop Stability Margin of Vehicular Platoons. *IEEE Transactions on Automatic Control*, 54(9), 2100–2113.

Coelingh, E. and Solyom, S. (2012). All aboard the robotic road train. *IEEE Spectrum*, 49(11), 34–39.

Herman, I., Martinec, D., Hurák, Z., and Sebek, M. (2016a). Scaling in bidirectional platoons with dynamic controllers and proportional asymmetry. *IEEE Transactions on Automatic Control*, PP(99), 1–6.

Herman, I., Martinec, D., and Veerman, J.J.P. (2016b). Transients of platoons with asymmetric and different Laplacians. *Systems & Control Letters*, 91, 28–35.

Lin, F., Fardad, M., Jovanović, M.R., and Jovanovic, M.R. (2012). Optimal control of vehicular formations with nearest neighbor interactions. *IEEE Transactions on Automatic Control*, 57(9), 2203–2218.

Lunze, J. (2012). Synchronization of Heterogeneous Agents. *IEEE Transactions on Automatic Control*, 57(11), 2885–2890.

Martinec, D., Sebek, M., and Hurak, Z. (2012). Vehicular platooning experiments with racing slot cars. In *2012 IEEE International Conference on Control Applications (CCA)*, 166–171.

Milanes, V., Shladover, S.E., Spring, J., Nowakowski, C., Kawazoe, H., and Nakamura, M. (2014). Cooperative adaptive cruise control in real traffic situations. *IEEE Transactions on Inteligent Transportation Systems*, 15(1), 296–305.

Naus, G., Vugts, R., Ploeg, J., van de Molengraft, R., and Steinbuch, M. (2010). Cooperative adaptive cruise control, design and experiments. In *American Control Conference 2010*, 1, 6145–6150.

Seiler, P., Pant, A., and Hedrick, K. (2004). Disturbance Propagation in Vehicle Strings. *IEEE Transactions on Automatic Control*, 49(10), 1835–1841.

Wieland, P., Sepulchre, R., and Allgöwer, F. (2011). An internal model principle is necessary and sufficient for linear output synchronization. *Automatica*, 47(5), 1068–1074.

Yadlapalli, S., Darbha, S., and Rajagopal, K. (2006). Information Flow and Its Relation to Stability of the Motion of Vehicles in a Rigid Formation. *IEEE Transactions on Automatic Control*, 51(8), 1315–1319.