Ad Hoc Networks 170 (2025) 103781

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Simopticon: Automated optimization of vehicular platooning controllers

Per Natzschka, Burkhard Hensel *, Christoph Sommer

TU Dresden, Faculty of Computer Science, Germany

ARTICLE INFO

Dataset link: cms-labs.org/research/software/simopticon/, zenodo.org/records/13828790

MSC:
37M05
70Q05
68-04

Keywords:
Simulation
Optimization

ABSTRACT

Platooning control – the automatic distance control in a chain of vehicles – has seen more than 2000 publications in the past 30 years, along with a corresponding number of different algorithms. However, comparisons between such controllers have rarely been done. Moreover, a fair comparison requires that all controller parameters are chosen to be optimal, often manually, which is a labor intensive and hard to replicate process. In this article we demonstrate the benefits of a methodology for parameter selection that encompasses: an evaluator employing a common metric, a simulator component, and an optimizer, all integrated into an optimization framework – along with an open-source reference implementation. We also discuss the trade-offs of different optimization algorithms, both from the literature and custom-built, for parameter optimization of platooning controllers.

1. Introduction

There are many reasons to let vehicles form platoons, i.e., to form a chain of vehicles driving with automated distance control. Besides improving driver comfort, automatic control has a shorter reaction time and allows thus shorter inter-vehicle distances. That increases road capacity and reduces air drag, resulting in less energy consumption and less air pollution [1–4]. Since the 1990s, such distance control is often informed by wireless messages exchanged between the vehicles [5], because this allows using more knowledge and therefore shortening further the distance between the vehicles and improving safety. Platooning with wireless message exchange is usually called Cooperative Adaptive Cruise Control (CACC).

Over the years, a wide variety of control algorithms has been used, from simple equations based on sliding-mode control [6,7], to consensus controllers [8,9], to optimal control [10] and complex model-based predictive controllers [11,12]. Many modern controllers can even take inhomogeneous platoons (different vehicle types in the same platoon), transmission delays, or the loss of wireless packets into account. Yet, most publications compare the proposed algorithm only with one or two other controller types without discussing if these controllers have been optimized appropriately. Control algorithms contain parameters with which their behavior can be adjusted to a given platoon or control objective. These parameters have not only an influence on control performance, but also on *internal stability* and *string stability*.

Especially from a practical point of view, the question is open, which of the hundreds of control algorithms should be preferred, because detailed and fair comparisons of more than a few controllers are rare.

There is thus a very real necessity to move away from manual, case-by-case optimization of select controllers, which are becoming increasingly complex – and to instead move towards the fully automatic, reproducible optimization of platooning controllers based on a shared metric.

The aim of this article is to contribute to the answer of this question. In brief, the key contributions of this article are:

- We demonstrate the benefits of a methodology for automatic parameter optimization, illustrated in Fig. 1, that encompasses all of: an evaluator employing a common metric, a simulator component, and an optimizer.
- For the optimizer, we discuss the trade-offs of different optimization algorithms for parameter optimization of platooning controllers, both straightforward ones from the Monte Carlo and Random Neighbors families and a custom variant of the DIRECT family of algorithms we call Simopticon-DIRECT.
- We discuss our open-source reference implementation of the presented methodology in an extensible architecture with a straightforward metric, the mean cumulative mean squared error ($\varepsilon_{\mathrm{MCMSE}}$). Notably, instead of requiring yet another custom simulator, its simulator component employs the proven and established Plexe simulation framework, which in turn is based on OMNeT++, SUMO, and Veins.

* Corresponding author.
E-mail addresses: per.natzschka1@mailbox.tu-dresden.de (P. Natzschka), burkhard.hensel@tu-dresden.de (B. Hensel).
URL: https://www.cms-labs.org/people/sommer (C. Sommer).

https://doi.org/10.1016/j.adhoc.2025.103781
Received 3 December 2024; Received in revised form 16 January 2025; Accepted 21 January 2025
Available online 29 January 2025

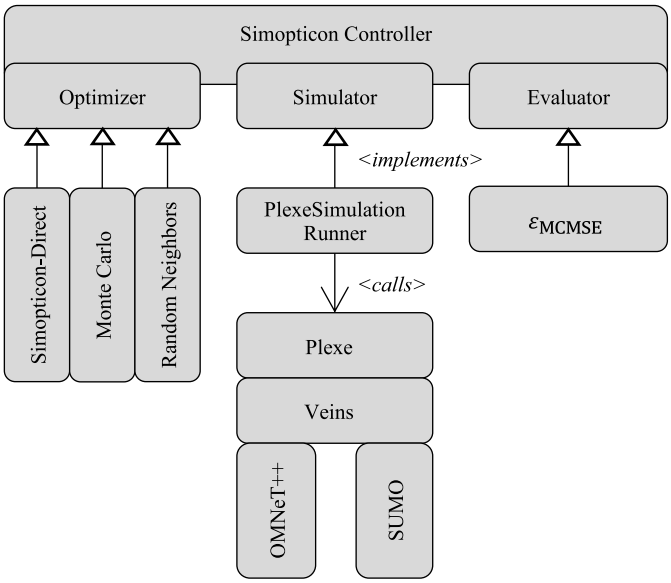


Fig. 1. Architecture of the optimization framework and open-source reference implementation discussed in this article.

- In case studies, we demonstrate that *a)* many common controllers have a lot of potential for optimization compared to their default parameterization, *b)* that the type of optimization strategy matters, and *c)* that the reference distance has an enormous influence on the control quality of some controllers.

It should be mentioned that parameter optimization is not directly necessary if controllers are used that learn their parameters during runtime (i.e., adaptive controllers [13] or methods based on reinforcement learning [14]). Comparable to this approach is also *self-organization*, where platoons homogenize their dynamic behavior [15]. Nevertheless, the by far largest part of published platoon controllers assumes constant dynamic behavior, and this article focuses on the optimization of such controllers. However, also self-learning approaches, like these mentioned before, have parameters (e.g., a learning rate) that might be optimized using Simopticon.

This article is structured as follows. We first discuss related work on platooning simulation, controller comparison, and optimization frameworks (Section 2). We then present the overall methodology and give an overview of the structure of the reference implementation used in the following studies (Section 3). We follow this with a detailed discussion of three optimization algorithms, two standard and one specifically adapted to platoon controller optimization (Section 4). We then discuss the results of a verification study of these algorithms, their performance for standard problems and the suitability of the chosen algorithm for platoon controller optimization (Section 5). We then present the results of a case study in which we demonstrate the impact of parameter optimization on the performance of a platoon controller, demonstrating how optimization drastically changes the perceived performance of five standard platoon controllers (Section 6). We finish with a brief summary and outlook on future work (Section 7).

A glossary together with the abbreviations used in this article is given in Table 1.

2. Related work

2.1. Simulation of platoons

Performance evaluation of platooning controllers is usually done by simulation. Several tools are available that have been used for simulating platoons. A broad overview is given by Segata et al. [16]. The most well known simulation software in the control domain is MATLAB. It is also often used for platooning [17–19]. DYNACAR has been used by Hidalgo et al. [20], TORCS by Ali et al. [18]. Several platooning models are based on CARLA, e.g., OpenCDA [15,21]. More alternatives are VISSIM [22], CarSim [23], HESTIA [24], and iTETRIS [25].

This variety of simulation tools shows that from the viewpoint of just simulating a platoon, it typically does not matter which simulation software is used, because most platoon control algorithms and vehicle models are simple enough to implement in *any* simulation software. Differences exist in features of the simulation tools that are not in each case relevant. Matlab is outstanding for mathematical purposes as it contains an immense set of mathematical features that help implement complicated methods and especially numerical stability analyses. On the other hand, Plexe, for instance, provides more detailed network models and the large set of features of the traffic simulation software SUMO. However, these features are not needed for the simple scenario of a straight road and simple periodic messages that are typical for platoon controller studies. Carla, as a third example, is mainly known for its highly realistic visualization, which is nice for presentations, but not really needed for the analysis of controllers. Simopticon has been designed to be as loosely coupled as possible to any simulation software, in order to be able to reuse it also for other simulators. The decision to start with Plexe as the first supported simulation software is mainly based on the experience of the authors in former works.

Plexe [16] is a platooning extension of the Car2x simulation framework Veins, that combines the traffic simulator SUMO with the network simulator OMNeT++. Plexe is open-source software designed for simulating platooning scenarios and testing platoon controllers. Plexe is based on the traffic simulator SUMO, the network simulator OMNeT++, and the Car2x simulator Veins. Most simulation details are implemented in C++ like in OMNeT++. Due to Veins, especially the network (typically using IEEE 802.11p as wireless communication technology) is modeled in detail with a lot of parameters. The main advantages of Plexe compared to the most often used software MATLAB is the availability of detailed network models, engine models, scenarios, and controller implementations in a unified framework, and the open-source license.

2.2. Comparison of platoon controllers

To the best of our knowledge, besides some basic case studies, no comparison of platoon controllers has been done. A comparison of basic platoon controllers has been made by Liu et al. [26] using the simulation framework Plexe. The focus was on comparing five controllers in a sinusoidal scenario with Gaussian noise. The parameters have been used as they are by default in Plexe. Hasan et al. [27] gave a comparison of two controllers regarding emergency brake using Plexe. Model Predictive Control (MPC) and CACC algorithms have been compared by Hidalgo et al. [20]. Some works [10,17,18,28,29] provide a comparison of a new controller with one or several others. Where Plexe has been used to simulate platoons, the default controller parameters have not been changed (as far as this information provided in the publications) [26,27,30,31].

2.3. Optimization frameworks

The optimization of simulation parameters has been addressed already many times and many such optimization frameworks have been developed for domains other than platooning [32–43]. The operating principle is usually equal: A parameter set is selected; a simulation is executed with that parameter set; metrics are computed from the simulation outputs; an optimizer decides based on the metrics which parameter set is the best current candidate; the parameters are changed according to an optimization strategy, and the loop starts again until a stopping criterion is fulfilled. To the best of our knowledge it has not yet been explored how this established strategy can be mapped to the domain of platooning controller optimization.

Table 1
Glossary and list of abbreviations used in this article.

CC	Cruise Control (*vehicle drives at a constant speed, independent of its predecessor*)	**BR**	Branin (*optimization benchmark function; see also the following functions*)
ACC	Adaptive Cruise Control (*vehicle attempts to follow its predecessor with a constant distance and/or time headway, based purely on sensors*)	**C6**	Six-Hump Camelback
		GP	Goldstein-Price
CACC	Cooperative Adaptive Cruise Control (*like ACC, but based on both sensors and wirelessly received information*)	**H3**	Hartman 3
		H6	Hartman 6
MPC	Model Predictive Control (*control strategy that uses a model of the system to predict future states and optimize a cost function*)	**S10**	Shekel 10
		S5	Shekel 5
		S7	Shekel 7
		SHU	Shubert

3. Methodology and reference implementation structure

Our methodology for parameter selection encompasses three distinct parts: An evaluator employing a common metric, a simulator component, and an optimizer. These three parts are integrated into an optimization framework. We also discuss a reference implementation of this methodology, called Simopticon.[1] It is an open-source framework providing functionality and interfaces for the iterative search of optimal simulation parameters.

Each iteration of Simopticon consists of three steps which are executed by three major components of Simopticon: *Optimizer*, *SimulationRunner*, and *Evaluation*; these correspond to the three parts of the methodology (Optimizer, Simulator, and Evaluator) illustrated in Fig. 1.

3.1. Evaluation

In the last step, the *Evaluation* must judge the performance of each simulated allocation by calculating a scalar performance value based on the acquired simulation data. This value is interpreted as a score of the evaluated parameter allocation and can be used by the *Optimizer* to determine the next set of allocations. In Simopticon, a lower performance score implies a better simulation performance. Therefore, the *Optimizer* searches the minimum of a blackbox function which can only be assessed by feeding it parameter allocations and obtaining the respective performance score. The optimization iteration is canceled when the optimization strategy in *Optimizer* finishes or the user interrupts the process. Simopticon allows for the use of arbitrary evaluation metrics.

The scoring function used in the following experiments is constructed as follows: It is based on the assumption that the best allocation ensures that the inter-vehicle gap is equal to the target gap at any given point in time for each vehicle. Thus, the performance of an allocation can be quantified by measuring the deviation of a vehicle with index i from its target gap $d_{\mathrm{ref},i}(t)$, which itself is a function of the speed $v_i(t)$. Plexe measures the inter-vehicle gap for each simulated vehicle at a fixed rate. Let T be the set of points in time when the gap is measured and $d_i : T \to \mathbb{R}$ denote the gap measurements between the ith and $(i-1)$th vehicle. The spacing error of each vehicle is calculated as the mean squared error. For a given scenario s and a given random seed r, the error $\varepsilon_{\mathrm{CMSE}}(s,r)$ in a simulation of n vehicles is accumulated by adding the respective mean squared errors:

$$\varepsilon_{\mathrm{CMSE}}(s,r) = \sum_{i=2}^{n} \frac{1}{|T|} \sum_{t \in T} \left(d_i(t) - d_{\mathrm{ref},i}(t) \right)^2 \tag{1}$$

Note that the vehicles are numbered from 1 to n, i.e., $i = 1$ is the platoon leader and therefore not relevant to the error calculation in Eq. (1). As was mentioned earlier, each parameter allocation may be simulated multiple times using different scenarios or random seeds. For each of those simulation runs the error $\varepsilon_{\mathrm{CMSE}}(s,r)$ is calculated as shown in Eq. (1), resulting in $|S| \cdot |R|$ error values, where S and R denote the simulated scenarios and random seed variations, respectively. To allow for a simple form of multi-objective optimization [44], i.e., optimizing the parameters to suit all simulated scenarios, the evaluation returns the arithmetic mean of the calculated error values as the mean cumulative mean squared error,

$$\varepsilon_{\mathrm{MCMSE}} = \frac{1}{|S| \cdot |R|} \sum_{s \in S} \sum_{r \in R} \varepsilon_{\mathrm{CMSE}}(s,r) \tag{2}$$

In controller parameter optimization, a multitude of different scenarios should be simulated to avoid overfitting of parameters to a small subset of the possible scenarios.

The $\varepsilon_{\mathrm{MCMSE}}$ metric allows for comparison of platoon stability based on gap measurements only. The main disadvantage of this simple method is that it does not account for collisions between vehicles. When a collision occurs, the simulation is stopped instantly and no further measurements are recorded. In this case, though, $\varepsilon_{\mathrm{MCMSE}}$ is comparatively high, since at least one vehicle i has a deviation of $-d_{\mathrm{ref},i}(t)$ from its target distance at the time of crash t. Nevertheless, there can be worse values of $\varepsilon_{\mathrm{MCMSE}}$ even if no collision occurs, e.g., if a gap measurement of a vehicle is $2d_{\mathrm{ref},i}(t)$ at some point t of the simulation. Thus, simulations with collisions do not necessarily lead to the highest (i.e., worst) possible evaluation score. This problem, however, did not affect the studies of this article since all minima found in simulation studies were far below the values calculated for simulations with colliding vehicles. The exploration of more sophisticated metrics that incorporate collisions and other requirements like string stability is left as future work. Such metrics could be implemented as a linear combination of different evaluation metrics.

3.2. SimulationRunner

The *SimulationRunner* automates the simulation of the selected allocations. In our case it automates the process of conducting platooning simulations with given control parameters using Plexe. Plexe currently contains five platoon controllers that are described in Section 6.1, but own controllers can be added. Each simulation project contains a text file containing adjustable parameters. These are not only controller parameters, but also parameters influencing the scenario, e.g., number of vehicles in the platoon, probability for packet losses, vehicle properties. In more detail, *SimulationRunner* is designed to create new Plexe simulation parameter files containing the requested parameter allocations based on a prototype file. Execution of the respective simulations is being parallelized to increase efficiency. This is especially important as Plexe allows the simulation of different platooning scenarios with the same parameter allocation. This – and repeating the same (stochastic) simulation with different random seeds a given number of times – lead to a potentially high number of simulation runs per sampled parameter allocation.

3.3. Optimizer

First, the *Optimizer* selects a set of parameter allocations that are simulated in the second step. The selection is based on previously sampled allocations and a blackbox optimization strategy.

[1] For download links, see the data availability statement at the end of the article.

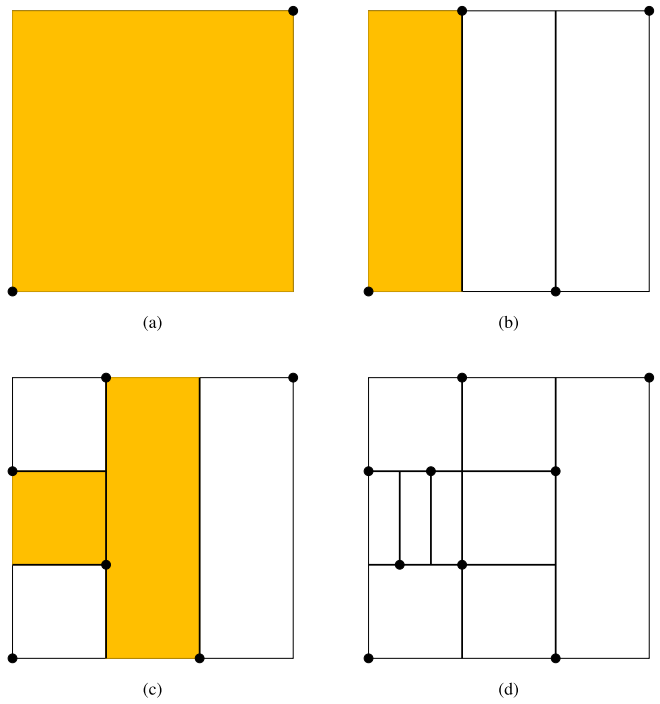


Fig. 2. Partition of a 2-dimensional hypercube.

A central controller uses the *Optimizer* to conduct the iterative optimization process using the respective components. A loose coupling of components makes the framework extendable with new implementations of the components, which can be used interchangeably. Similarly, each implementation of components has their own configuration file, since the available settings may differ between implementations. For example, the *SimulationRunner* implementation for Plexe may simulate different controllers that are selected in the respective configuration, but that option might not be relevant for other implementations.

We discuss the optimization strategies available to the *Optimizer* class in Section 4.

4. Optimization strategies

In the following we present three optimization strategies that we investigated for the optimization of platooning controllers.

Besides two simple stochastic optimizers, presented in Sections 4.2 and 4.3, respectively, the framework contains a deterministic, but complicated optimizer that is described in detail in Section 4.1. As in the practical experiments on platooning the simple stochastic algorithms showed a better convergence (see Section 6.4), readers interested only in the application might skip the long Section 4.1.

4.1. The Simopticon-Direct optimization algorithm

The Direct (DIviding RECTangles) algorithm is a well-known method of solving derivative-free global optimization problems that has seen a multitude of applications and adaptions [45,46] since its introduction in 1993 by Jones et al. [47]. Direct algorithms are deterministic and allow for Lipschitzian optimization without the need to specify a Lipschitz constant, i.e., no initial knowledge about the optimized problem is needed. Adaptions of the Direct algorithm often outperform other state-of-the-art derivative-free global optimization methods [48]. The main drawback of Direct algorithms is the "curse of dimensionality" [45], i.e., their bad performance in higher dimensionality. This, however, is not problematic when optimizing platoon controllers since they usually only have a small number of parameters. Therefore, we investigate the Direct algorithm family as a candidate for the optimization of platooning controllers.

However, one of the major strengths of Direct algorithms – their balancing of global and local search – can become a weakness when the basin of the global optimum has been found and local refinement should prevail [45]. This is especially problematic in the context of simulation optimization where every evaluation is computationally costly, i.e., unnecessary global exploration may severely slow down the optimization process. We therefore chose to adapt the Direct algorithm [47] to the specific use case of optimizing platooning controllers. To accomplish this, we combine the Direct variation by Liu et al. [49,50] (which adds local refinement phases) with the partitioning scheme introduced by Sergeyev and Kvasov [51] (which further improves performance in low dimensionality). We call this adaptation Simopticon-Direct. It is adapted – and, as we will show, particularly effective – for low-dimensional problems ($n < 4$), which is the case for the platooning controllers we are optimizing.

4.1.1. Search space partition

The search space for a controller with n parameters is, without loss of generality, normalized to the n-dimensional hypercube. The Direct algorithm partitions said search space into increasingly smaller hyperrectangles. In each iteration step, a subset of the current partition is selected (see Section 4.1.2) for further exploration. The rectangles of this subset are then split into smaller rectangles which in turn are sampled again. This way the algorithm approaches the global minimum by dividing rectangles where good values have been found and thereby sampling around the optimum more densely.

To be more precise, a selected rectangle is always split into thirds. For $n = 2$ dimensions, this is shown in Fig. 2 where the selected rectangles are highlighted in yellow. Rectangles are always split along one of their longest sides which ensures that each dimension is explored to the same extent throughout the optimization. If there are multiple longest sides (e.g., Fig. 2(a)), the split dimension may be chosen arbitrarily. Since the partition strategy of Sergeyev and Kvasov [51] is used, the split dimensions are selected in ascending order here.

While most Direct variations sample the centerpoint of newly added rectangles [49,50,52–58], Simopticon-Direct samples opposite vertices as proposed by Sergeyev and Kvasov [51]. In low dimensionality, this allows for a more precise selection of rectangles that are being split, since every rectangle can be judged based on two sampled values, instead of one. Counter-intuitively, sampling opposite vertices reduces the number of sampling points required to partition the search space into a given number of rectangles, instead of doubling it. This is, for example, visible in Fig. 2, where the sampling points are visualized by black dots. While the ratio of sampling points to rectangles is indeed $2:1$ in Fig. 2(a), it rapidly changes to $1:1$ in Fig. 2(d). This is due to sampling points being used as vertices of multiple rectangles.

4.1.2. Search strategy

One key advantage of the Direct algorithm is its deterministic, direct search strategy, i.e., its selection of rectangles that are sampled and divided further in each iteration step. The strategy is derived from Lipschitzian optimization, i.e., it judges the potential optimality of all rectangles in the partition based on an estimate of their lower bound. Let $f : [0,1]^n \to \mathbb{R}$ be the optimized function and assume that we know a Lipschitz constant K for f, i.e., we know a $K \in (0, \infty)$ such that for any $\vec{x}, \vec{x'} \in [0,1]^n$:

$$\left| f(\vec{x}) - f(\vec{x'}) \right| \leq K \cdot \left\| \vec{x} - \vec{x'} \right\| \tag{3}$$

This condition is not fulfilled if there are random influences on the simulated process, e.g., due to stochastic packet losses. Here we assume a deterministic behavior of the simulation, the other case is considered later.

Table 2
Levels of optimization.

Phase	Level j	ς_j	ϵ_j
Global	3	50%	10^{-5}
Local	2	100%	10^{-5}
	1	95%	10^{-7}
	0	4%	0

Following Sergeyev and Kvasov [51], a lower bound $\hat{M}$ of f on a rectangle r with vertices $\vec{a}, \vec{b} \in [0,1]^n$ can be estimated using a known Lipschitz constant K as

$$\hat{M}(r,K) = \frac{f(\vec{a}) + f(\vec{b})}{2} - K \cdot \left\| \vec{b} - \vec{a} \right\|, \tag{4}$$

where $\|\cdot\|$ denotes the Euclidean norm. The rectangle with the lowest lower bound is considered to be most likely to contain the optimal parameter allocation.

The problem of the estimation using Eq. (4) is its reliance on a known Lipschitz constant K. Since the algorithm is dealing with blackbox functions, that parameter is not known for most problems. Moreover, K acts as a weight between local and global search. If $K \to 0$, rectangles with lowest values at their vertices have the lowest estimation, which is equivalent to local search. Conversely, the largest rectangles have the lowest estimation for $K \to \infty$, which is equivalent to global search. Therefore, using a too high or too low estimation of the Lipschitz constant may lead to poorer performance. DIRECT circumvents this problem by factoring in all possible $K \in (0, \infty)$. Let $P = \{ r_1, r_2, \ldots, r_m \}$ be the current partition of the search space into m rectangles $r_1, \ldots, r_m$. The set $I \subseteq P$ of potentially optimal rectangles consists of all rectangles $r \in P$ that satisfy

$$\exists_{K \in (0,\infty)} : con_1 \wedge con_2 \tag{5}$$

$$con_1 \equiv \forall_{r' \in P} : \hat{M}(r,K) \leq \hat{M}(r',K) \tag{6}$$

$$con_2 \equiv \hat{M}(r,K) \leq \phi - \varepsilon \cdot \left| \phi - \tilde{f} \right| \tag{7}$$

Here, con_1 ensures that I only contains rectangles which have the lowest estimate $\hat{M}$ for a certain K. In Eq. (7), $\tilde{f}$ denotes the median and ϕ the lowest of all values sampled from f so far. con_2 is used to filter out rectangles r that satisfy con_1 for one K' if their lower bound $\hat{M}(r,K')$ cannot undermatch the current minimum ϕ by a percentage defined in the hyperparameter ε. This condition is used to prevent unnecessary costly sampling in rectangles that cannot – by estimation – yield a significant improvement. The set of potentially optimal rectangles I is calculated and sampled (as shown in Section 4.1.1) using the condition in Eq. (5) in each iteration step.

4.1.3. Optimization levels

So far, the described algorithm does not differ much from the variation proposed by Sergeyev and Kvasov [51]. We now combine the described partitioning scheme with the leveled approach described by Liu et al. [49,50]. This approach tries to mitigate *global drag*, a major weakness of DIRECT algorithms: DIRECT algorithms are known for finding the basin of the global optimum quickly but being rather slow in locally refining that optimum. This is due to the search strategy sampling all potentially optimal rectangles while only one of them may contain the optimum. Especially for more refined partitions where the basin has been found and local search should prevail, DIRECT focuses on global optimization with equal vigor which leads to unnecessary sampling of large rectangles.

Simopticon-DIRECT mitigates the global drag by switching between different levels of optimization which are locally or globally biased as proposed by Liu et al. [49,50]. There are four different levels which are summarized in Table 2. Level 2 is the default level, where no bias is used. To shift towards a more local optimization, levels 1 and 0 narrow the selection of potentially optimal rectangles to the smallest rectangles. Let $P = \{ r_1, r_2, \ldots, r_m \}$ again be the current partition, sorted by size of the rectangles in ascending order. The size of a rectangle r_i can, for example, be measured as $\left\| \vec{b}_i - \vec{a}_i \right\|$ where $\vec{a}_i, \vec{b}_i \in [0,1]^n$ are the vertices of r_i. For levels $j = 0, 1$, only a subset $P'_{local} \subseteq P$ is used in rectangle selection, which is defined as

$$P'_{local} = \left\{ r_i \in P \middle| \left\| \vec{b}_i - \vec{a}_i \right\| \leq \left\| \vec{b}_k - \vec{a}_k \right\| \right\} \tag{8}$$

$$k = \left\lceil \varsigma_j \cdot |P| \right\rceil \tag{9}$$

Conversely, there is level $j = 3$ which narrows rectangle selection to the largest rectangles, thereby creating a global bias. This is inspired by Sergeyev and Kvasov [51] where a global phase is introduced to avoid getting stuck in local optima. In the global phase, only a subset $P'_{global} \subseteq P$ is used in rectangle selection, which is defined as

$$P'_{global} = \left\{ r_i \in P \middle| \left\| \vec{b}_i - \vec{a}_i \right\| \geq \left\| \vec{b}_k - \vec{a}_k \right\| \right\}, \tag{10}$$

where k has the meaning of Eq. (9).

Additionally, the local bias is reinforced in levels 1 and 0 by using smaller values for ε in rectangle selection (see Table 2). This leads to rectangles being filtered out less frequently by Eq. (7), i.e., rectangles are explored even though they are not estimated to yield major improvements.

Simopticon-DIRECT starts in the local phase with level 2. In a local phase, the level changes after each iteration in a *W-pattern* proposed by Liu et al. [49], i.e., 2-1-0-1-1-0-1-2. After every four iterations the algorithm checks if a significant improvement was made — otherwise it switches to a global phase and therefore to level 3. Let ϕ and ϕ' be the best sampled values before and after those four iterations, respectively, and let $\tilde{f}$ denote the median of all sampled values. Then the condition for a switch to a global phase is

$$\phi' \leq \phi - \varepsilon_3 \cdot \left| \phi - \tilde{f} \right| \tag{11}$$

When the algorithm switches to a global phase, it resides on level 3 until the condition in Eq. (11) is met again (with ϕ being the best sampled value at the beginning of the global phase) or until n iterations have been completed, whichever happens first. When a global phase finishes, the algorithm resumes the interrupted W-cycle, i.e., if the local phase was interrupted after iterations on levels 2-1-0-1, it resumes with levels 1-0-1-2 after the global phase. In all our studies we chose the maximum number of iterations in the global phase via the standard benchmark functions discussed in Section 5.1, which resulted in the best value for a maximum of $n = 35$ iterations.

4.2. The Monte Carlo optimization algorithm

Besides the previously described optimizer, also a simple Monte Carlo optimizer is investigated. Monte Carlo methods use random variables to determine the parameter sets to be simulated [59]. In this implementation, the Monte Carlo algorithm sets all parameters randomly in the global defined bounds using a uniform distribution. Therefore, for a large iteration count, they cover the parameter space more or less in an equal density, but they do not use knowledge of prior simulation runs in order to search close to good previous solutions. This avoids sticking in local optima, but it also makes the optimum search into a random procedure without any guarantee to reach the global optimum after a finite number of iterations. Since the step size is not reduced during optimization, the algorithm does not converge in the classical meaning of optimization. The only parameter of this optimizer implementation is the number of parallel simulation runs for speeding up the optimization procedure.

One further advantage of the Monte Carlo optimizer compared to the Direct optimizer is that it does not require the Lipschitz condition (3) and, so, is better suited for simulations with stochastic influences like packet losses. Our implementation of Monte Carlo uses a fixed random seed in order to allow for reproducible results, despite the stochastic nature of the method. As the algorithm takes all parameter sets from the full global parameter space, the influence of the random seed should be negligible.

Table 3
Benchmark functions [47] used for validation of the Simopticon-Direct, Monte Carlo, and *Random Neighbors* optimization algorithms.

Benchmark function	Abbr.	n	Minima	
			Local	Global
Branin	BR	2	3	3
Goldstein-Price	GP	2	4	1
Six-Hump Camelback	C6	2	6	2
Shubert	SHU	2	760	18
Hartman 3	H3	3	4	1
Shekel 5	S5	4	5	1
Shekel 7	S7	4	7	1
Shekel 10	S10	4	10	1
Hartman 6	H6	6	4	1

4.3. The Random Neighbors optimization algorithm

The *Random Neighbors* algorithm is an extension of the Monte Carlo algorithm. One drawback of the Monte Carlo algorithm is that is does not know already available knowledge about prior simulation results and searches always with the same probability in the full parameter scope. However, searching in the local surrounding of the current optimum has often been found to be a good idea, e.g., in the famous *simulated annealing* approach [60]. In order to use prior results, the *Random Neighbors* algorithm uses an adjustable percentage of simulations with parameters narrow around the current optimum, because it is probable that the simulation results there are better than in the average of the global parameter scope. In this article, 25% of all simulations are used to search in a region of ±5% around the current optimum for each parameter. The other 75% are usual Monte Carlo simulations. The parameters are adjustable in the software. As long as the current optimum is far away from the global optimum, this algorithm is worse than Monte Carlo optimization as there are unnecessarily many simulations around a local optimum that is not the global optimum. Besides, due to the stochastic nature, also this algorithm cannot guarantee to converge to the global minimum. Still, like the Monte Carlo optimizer it does not require the Lipschitz condition (3), what is relevant for optimizing stochastic simulations. A fixed random seed is used in our implementation to allow for reproducible results. The fact that 75% of the tested parameter sets are taken randomly from the full parameter space (and not only from a surrounding around the current optimum) should minimize the effect of the random seed selection.

5. Verification

5.1. Standard optimization models

To investigate the feasibility of Simopticon-Direct, Monte Carlo, and *Random Neighbors* optimization for low dimensional problems like the optimization of controller parameters, they have been used to optimize the problems listed in Table 3. The original Direct algorithm is used as a baseline since it has been validated on the same benchmarks by Jones et al. [47]. Here, convergence is defined similarly as by Jones et al. [47] via the percent error e of the current best solution ϕ from the global minimum f^*:

$$e = \frac{\phi - f^*}{|f^*|} \leq e' \tag{12}$$

If this condition holds for a given threshold e' after an iteration, the algorithm is considered to have converged on the global minimum.

Table 4 shows the number of evaluations until convergence is reached for $e' = 10^{-4}$. For low dimensional problems ($n < 4$), Simopticon-Direct outperforms the original in most cases. Especially for the Shubert (SHU) function, a great improvement can be noted. This is particularly interesting since SHU contains a multitude of local minima, which seem to distract original Direct more than the multi-level approach of Simopticon. On the other hand, problems of higher dimensionality pose a problem to Simopticon-Direct. While the difference is not excessive for the Shekel functions ($n = 4$), Simopticon-Direct fails to converge in a reasonable number of evaluations for Hartman 6 (H6). Sergeyev and Kvasov [51] report a similar pattern where their variation underperforms on Shekel 7 (S7), Shekel 10 (S10) and H6 by orders of magnitude. Since the variation in Simopticon is heavily based on work by Sergeyev and Kvasov [51], this issue has been inherited, but weakened for the Shekel functions.

Table 4
Optimizer comparison: number of evaluations until convergence is reached for $e' = 10^{-4}$. See Table 3 for details on the benchmarks used.

Benchmark	Evaluations			
	Direct [47]	Simopticon Direct	Monte Carlo	Random Neighbors
BR	195	**134**	17 114	1018
GP	**191**	210	57 713	572
C6	285	222	5 595	**104**
SHU	2967	**1 822**	6 145	3031
H3	199	**181**	938	3275
S5	**155**	237	–	6667
S7	**145**	227	–	5904
S10	**145**	258	–	2680
H6	**571**	13 443	–	1457

Table 5
Optimizer comparison: number of evaluations until convergence is reached for $e' = 10^{-2}$. See Table 3 for details on the benchmarks used.

Benchmark	Evaluations			
	Direct [47]	Simopticon Direct	Monte Carlo	Random Neighbors
BR	**63**	72	17 120	263
GP	**101**	168	57 728	104
C6	113	188	5 600	**48**
SHU	2883	1 780	6 160	**325**
H3	83	**26**	944	854
S5	**103**	182	–	712
S7	**97**	190	–	724
S10	**97**	202	–	1029
H6	**213**	13 299	–	247

Table 5 shows the same comparison for a weaker threshold $e' = 10^{-2}$. Monte Carlo optimization yields similarly bad results in both cases, even failing to converge in less than 10^7 evaluations for $n > 2$. *Random Neighbors* on the contrary needs much fewer evaluations for the weaker convergence criterion which means that the algorithm is fast to find the basin of the global optimum but slow in locally refining said optimum.

5.2. Plexe simulations

To verify the effectiveness of Simopticon-Direct for platoon controllers, it is used with Simopticon to optimize the parameters of the *PATH* controller [6]. The other optimization methods are compared to Simopticon-Direct later in Section 6. The *PATH* controller has three parameters C_1, ξ, and ω_n (see Section 6.1 for details). Simopticon is used to find the optimal parameter allocation $(C_1, \xi, \omega_n) \in [0, 1] \times [10, 25] \times [1, 12]$. Each parameter allocation is simulated in Plexe's *Sinusoidal* scenario. This scenario starts with a stable platoon of 8 vehicles driving at 100 km/h. After maintaining a constant speed for 5 s, the platoon leader's acceleration is varied between 1.45 m/s^2 and −1.45 m/s^2 in a manner that ensures its velocity matches a sine function with a frequency of 0.2 Hz, oscillating between 95 km/h and 105 km/h. This continues until either a crash occurs or the simulation time reaches 60 s. Each simulation of a parameter allocation is repeated 5 times and the respective error values are calculated as shown in Eq. (1) and arithmetically averaged to get the performance value $\varepsilon_{\text{MCMSE}}$ (cf. Eq. (2)).

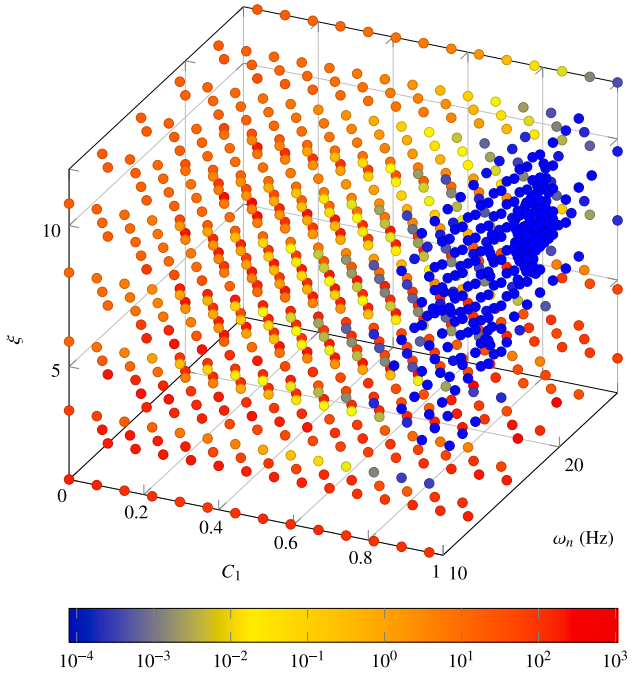


Fig. 3. Sampled locations C_1, ξ, ω_n and their corresponding ϵ_{MCMSE} values in the parameter space during one optimization run.

Fig. 3 shows the ϵ_{MCMSE} values sampled throughout the optimization process. It is evident that regions which yield bad values, e.g., $C_1 < 0.5$ are sampled evenly but in low granularity. Conversely, regions with good values are sampled in great detail as can be seen around the found optimum $(C_1, \xi, \omega_n) = (1, 86/9, 65/3\,\text{Hz})$. For this parameter allocation a very low ϵ_{MCMSE} value of 7.9587×10^{-5} is calculated. Fig. 3 shows a clear basin around that optimum, indicating that parameter allocations with $C_1 \approx 1$, $\xi \approx 9$, and $\omega_n \approx 20\,\text{Hz}$ lead to good performances of the *PATH* controller.

To validate the expressiveness of the ϵ_{MCMSE} calculation, Fig. 4 shows the correlation between platoon stability and low ϵ_{MCMSE} values by plotting the control error of the simulated vehicles. The control error is defined as the difference between the target gap $d_{\text{ref},i}(t)$ of the ith vehicle to its predecessor and the actual gap $d_i(t)$ and at a given point in time t:

$$e_c(t) = d_{\text{ref},i}(t) - d_i(t) \tag{13}$$

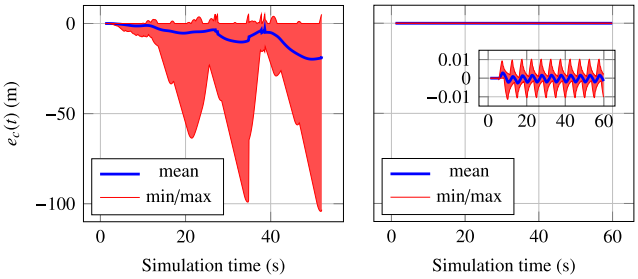
In this experiment, the target gap was set to $d_{\text{ref},i}(t) = 5\,\text{m}$.

Fig. 4(a) shows a simulation using the worst found parameter allocation $(C_1, \xi, \omega_n) = (2/27, 70/3, 53/9\,\text{Hz})$ which leads to ϵ_{MCMSE} value of 1061.87. It can be seen that bad allocations can lead to deviations from the target gap of up to 20 m in the mean and even more than 100 m for individual vehicles. Additionally, the simulation ends early after only 52 s, because a crash occurs in the simulation.

The opposite can be seen in Fig. 4(b) where the behavior of the best found parameter allocation is plotted. With the found parameters, the vehicles only deviate up to 11.4 mm from the target gap — on average only 2.85 mm. Moreover, the graph shows an alternating pattern that resembles a sine function with 0.2 Hz frequency — the same frequency in which the leading vehicle alternates its speed in the *Sinusoidal* scenario.

6. Case study

As a case study, we now apply the presented methodology to optimize the parameters of the controllers available in Plexe. We set out to demonstrate that, for the discussed example, the ranking of



(a) Suboptimal parameters (b) Optimized parameters

Fig. 4. Control error $e_c(t)$: Deviation of vehicles from target gap $d_{\text{ref},i}(t)$.

the controllers after optimization is very different to the ranking of the controllers with default parameterization, an important result for controller selection. Further, this section will show that both the optimum parameters and the controller ranking depend on the reference distance.

All simulations have been conducted with Simopticon 1.1.0, OMNeT++ 6.0.2, Veins 5.2, as well as the extensions of Plexe 3.1.1 and SUMO 1.18 linked from the data availability statement at the end of this article.

The simulations use the default vehicle model of Plexe, i.e., a linear, double-integrator, first-order lag model with a lag of $\tau = 0.5\,\text{s}$. The input of the vehicle model is the *desired acceleration* measured in m/s^2. The vehicles have a minimum and maximum acceleration that are $a_{\min} = -9\,\text{m/s}^2$ and $a_{\max} = 2.5\,\text{m/s}^2$.

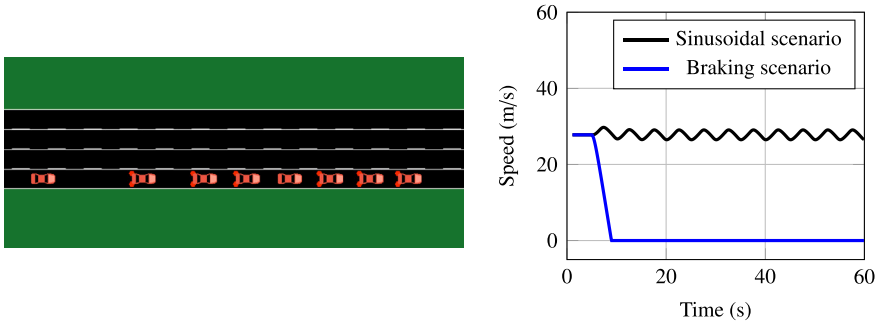
Also the communication system uses the default parameters of Plexe, i.e., an IEEE 802.11p network is used and the beaconing interval is 100 ms. The packet loss rate of the receivers' MAC layer is set to zero, reducing the stochastic behavior of the simulation results. However, besides the packet loss rate, also other features of Plexe are influenced by the random seed, e.g., the random backoffs, the antennas, and the vehicle mobility, but the influence of these aspects on the platoon performance is rather small using the default settings. The Lipschitz condition (3) is not fulfilled in the presence of random influences, so the Monte Carlo or Random Neighbors optimizers might be preferred. Nevertheless, only one repetition per parameter combination is used in the optimization runs shown below, because the effect of the number of repetitions on the optimization result is part of the ongoing research.

For the simulations, two default scenarios of Plexe are used. In both cases the platoon consists of 8 passenger cars driving on a straight road from left to right (Fig. 5(a)). In the *sinusoidal* scenario, the leader speed starts to oscillate around 100 km/h 5 s after the start of the simulation. In the *braking* scenario, the leader starts to break 5 s after the start, with a deceleration of $8\,\text{m/s}^2$, which is slightly below the maximum deceleration of $9\,\text{m/s}^2$, shown in Fig. 5(b). Using both scenarios enables implicit multi-objective optimization through ϵ_{MCMSE}, as discussed in Section 3.1. This approach ensures that the parameter allocations found are robust, avoiding overfitting to either the sinusoidal or braking scenario individually. Additionally, these two scenarios cover two very interesting situations of platooning: Emergency braking is the most challenging scenario for avoiding vehicle collisions and is therefore most interesting regarding safety, while the oscillations of the sinusoidal scenario make string stability issues most clearly visible.

6.1. Controller types

In this study, mainly the controllers available in Plexe are used. They are called *ACC*, *PATH*, *Ploeg*, *Consensus*, and *Flatbed*. All controllers but *ACC* use information from (simulated) Car2x messages whereas *ACC* simulates on-board distance measurements with Radar.

In this section, the control equations of these controllers are given. Although this content could also be read in the literature, we repeat



(a) Platoon in Plexe consisting of 8 vehicles (b) Leader speed

Fig. 5. Plexe scenarios used in simulation.

Table 6
Default spacing policy of Plexe controllers. '—' means that there is no parameter to adjust the value, it is hardcoded in the source code.

Controller	d_0 in m	Parameter	h in s	Parameter
ACC	2	–	0.3/1.2	*accHeadway*
PATH	5	*caccSpacing*	0	–
Ploeg	2	–	0.5	*ploegH*
Flatbed	5	*flatbedD*	0	–
Consensus	15	–	0.1	*headway*
Yan	20	*yanR*	0	–

these equations here for two reasons: First, the equations show the meaning of the parameters of each controller which are optimized by Simopticon. Second, the equations make clear why the control performance of some controllers depends on the reference distance, while for others, it does not.

The *reference distance*, also called *formation geometry* or *inter-vehicle spacing policy*, is called $d_{\text{ref},i}(k)$ for all controllers and in general defined as

$$d_{\text{ref},i}(k) = d_0 + h \cdot v_i(k) \tag{14}$$

with standstill distance d_0 and time headway h. $v_i(k)$ is the velocity of the vehicle with index i and k the sampling instant. For all controllers only one of these two parameters is adjustable, the other one is fixed. The default values of these parameters are shown in Table 6.

The *ACC* controller, taken from Ioannou and Chien [61] and Rajamani [7], is implemented in Plexe as

$$a_i(k) = \min\Big(a_{i,\max}, \max\Big(-a_{\text{decel}}, K_p \cdot (v_i(k) - v_{\text{des}})\Big)\Big), \tag{15}$$

if the distance to the previous vehicle is greater than 250 m (Cruise Control (CC)), and

$$a_i(k) = \frac{1}{h}\Big(v_i(k) - v_{i-1}(k) + \lambda\Big(d_{\text{ref},i}(k) - d_{i,i-1}(k)\Big)\Big), \tag{16}$$

otherwise. Due to the factor $1/h$, this controller requires a nonzero time headway h and can therefore not be used for a constant target distance. If the value computed by the CC controller is smaller than the value computed by the Adaptive Cruise Control (ACC), CC is used, even if the distance is below 250 m. So, in ACC mode there is only one controller parameter (λ), besides the adjustable headway h.

The *PATH* controller (that is called *CACC* in Plexe) is implemented according to Swaroop et al. [6]. It is often called the *PATH* controller because it has been developed in the *California PATH* project at UC Berkeley [6]. It is implemented as

$$\begin{aligned} a_i(k) = {} & \alpha_1 a_{i-1}(k) + \alpha_2 a_L(k) + \alpha_3\Big(v_i(k) - v_{i-1}(k)\Big) \\ & + \alpha_4\Big(v_i(k) - v_L(k)\Big) + \alpha_5\Big(d_{\text{ref},i}(k) - d_{i,i-1}(k)\Big), \end{aligned} \tag{17}$$

where index L references the platoon leader and $i-1$ the predecessor of the controlled vehicle. Its parameter selection in Plexe is based on the sliding surface method of controller design [7,62], which reduces the number of independent parameters from five, that is $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$, to three, that is, (C_1, ξ, ω_n):

$$\alpha_1 = 1 - C_1 \tag{18}$$

$$\alpha_2 = C_1 \tag{19}$$

$$\alpha_3 = -\Big(2\xi - C_1 \cdot \Big(\xi + \sqrt{\xi^2 - 1}\Big)\Big) \cdot \omega_N \tag{20}$$

$$\alpha_4 = -\Big(\xi + \sqrt{\xi^2 - 1} \cdot \omega_N \cdot C_1\Big) \tag{21}$$

$$\alpha_5 = -\omega_N^2 \tag{22}$$

The *Ploeg* controller, based on the work of Ploeg et al. [63], is implemented as

$$\frac{\Delta u_i(k)}{\Delta T} = \frac{1}{h} \cdot \Big(-u_i(k) + k_p e_i(k) + k_d \dot{e}_i(k) + u_{i-1}(k)\Big), \tag{23}$$

$$e_i(k) = d_i(k) - d_{\text{ref},i}(k), \tag{24}$$

$$\dot{e}_i(k) = v_{i-1}(k) - v_i(k) - h \cdot a_i(k), \tag{25}$$

$$a_i(k) = a_i(k-1) + \Delta u_i(k). \tag{26}$$

Also this controller requires a nonzero time headway h and cannot be used for a constant target distance. Note that this implementation differs from the original controller [63]: Besides the missing $\ddot{e}_i(k)$ part, the implementation in Plexe uses a first order lowpass system for the full controller while Ploeg [63] uses only a first order lowpass system to estimate the real acceleration from the transmitted desired acceleration. The Plexe version of the *Ploeg* controller has two parameters: k_p and k_d.

The *Flatbed* controller (developed by Ali et al. [18]) is implemented as

$$\begin{aligned} a_i(k) = {} & -K_a a_i(k-1) + K_v\Big(v_{i-1}(k) - v_i(k)\Big) \\ & + K_p\Big(d_{i,i-1}(k) - d_{\text{ref},i} - h_{\text{des}}\Big(v_i(k) - v_L(k)\Big)\Big). \end{aligned} \tag{27}$$

However, the original work by Ali et al. [18] computed the jerk W with this equation, not the acceleration $a_i(k)$.

Plexe contains also a *Consensus* controller, to be more specific a version developed by Santini et al. [9]. However, this controller's implementation is not parameterizable in the way the other controllers can be parameterized. Instead of making the parameters of the existing implementation adjustable, we decided to implement another version of the consensus controller based on Yan et al. [8], because it has a similar control strategy, but a more flexible and simple way for parameterization and a stronger focus on network issues like delays and packet losses. We call it the *Yan* controller. The basic idea of the consensus controller is that each vehicle compares its own position, speed, and acceleration with the values of all or a subset of all other

Table 7
Control performance ϵ_{MCMSE} before and after optimization. The default reference distance parameters belong to Eq. (14). With respect to Eq. (14), reference distance 5 m means a fixed distance of $d_0 = 5\,\text{m}$ and $h = 0$ for controllers working with a constant reference distance (*PATH*, *Flatbed*, and *Yan*) as well as $d_0 = 2\,\text{m}$ and $h = 0.108\,\text{s}$ for controllers that require a nonzero headway (*ACC* and *Ploeg*). Reference distance 35 m means $d_0 = 35\,\text{m}$ and $h = 0$ for controllers working with a constant reference distance as well as $d_0 = 2\,\text{m}$ and $h = 1.188\,\text{s}$ for controllers that require a nonzero headway.

Controller	Default ref. dist.	Unoptimized (using default parameters)		Optimized for ref. dist. 5 m		Optimized for ref. dist. 35 m	
		ϵ_{MCMSE} (m^2)	Rank	ϵ_{MCMSE} (m^2)	Rank	ϵ_{MCMSE} (m^2)	Rank
ACC	$d_0 = 2\,\text{m},\ h = 1.2\,\text{s}$	2.405	4 ••	11.037	5 •	5.628×10^{-7}	1 •••••
PATH	$d_0 = 5\,\text{m},\ h = 0$	0.639	2 ••••	4.683×10^{-3}	4 ••	4.686×10^{-3}	5 •
Ploeg	$d_0 = 2\,\text{m},\ h = 0.5\,\text{s}$	0.0642	1 •••••	4.016×10^{-4}	2 ••••	1.195×10^{-6}	2 ••••
Flatbed	$d_0 = 5\,\text{m},\ h = 0$	0.692	3 •••	9.129×10^{-5}	1 •••••	9.131×10^{-5}	3 •••
Yan	$d_0 = 20\,\text{m},\ h = 0$	14.730	5 •	4.030×10^{-4}	3 •••	4.036×10^{-4}	4 ••

vehicles. Using a simplified notation, the *Yan* controller is implemented as

$$\begin{aligned} u_i(k) = -\sum_{j=1}^{n} \alpha_{ij} \Bigg[& \Big(x_i(k) - x_j(k) - d_{ij}\Big) \\ & + \beta \cdot \Big(v_i(k) - v_j(k)\Big) + \gamma \cdot \Big(a_i(k) - a_j(k)\Big) \Bigg] \\ & - \kappa \Bigg[\Big(x_i(k) - x_0(k) - d_{i0}\Big) + \eta \cdot \Big(v_i(k) - v_0(k)\Big) \\ & + \Xi \cdot \Big(a_i(k) - a_0(k)\Big) \Bigg], \end{aligned} \tag{28}$$

where x_i is the position of vehicle i, d_{ij} the reference distance between the vehicles i and j (the sum of the reference distances $d_{\text{ref},i}$ and the vehicle lengths of all vehicles in between), κ represents the weighting of the leader, β and η the weighting of the velocity difference and γ and Ξ the weighting of the acceleration difference. Via the adjacency matrix (delivering α_{ij}) the topology can be specified. In the case study, a predecessor-leader-following topology is used, i.e., all α_{i0} and $\alpha_{i,i-1}$ are equal to 1 and all other α_{ij} are equal to zero.

6.2. Stability and string stability

Stability is a fundamental requirement in control applications and stability proofs build the core of most publications about control algorithms. In the case of platooning, besides *internal stability* [64] or *individual vehicle stability* [63] of each vehicle also *string stability* for the whole platoon is typically proven. There are a lot of string stability definitions [65]. Roughly speaking, a platoon is called string stable if there is no frequency for which speed oscillations are amplified from one vehicle to the next one. However, in this article, no stability proofs are given, because controllers are optimized based on a performance measure, not based on theoretic stability criteria.

The currently implemented optimizers of Simopticon take only simulation results into account, no theoretical stability checks. In most cases a control loop with optimal simulation results will surely be stable because in unstable control loops oscillations will lead to poor control performance. Nevertheless, a simulation is a case study and no proof for stability.

Future versions of Simopticon might also take stability conditions into account. Stability conditions typically limit the parameter space that is allowed to be used in the optimization procedure. However, if the global optimum lies in the region allowed by the stability criterion, it is not absolutely necessary to take these conditions into account; it only might speed up the convergence to the optimum. On the other hand, if the optimum lies outside of the region allowed by the stability condition, this condition should be revisited; maybe it bases on simplified assumptions that are not held by complex simulations. Alternatively, the stability condition might have higher priority because potential stability issues of a parameter set might be hidden in a concrete simulated scenario.

It is possible to integrate stability checks either in the optimizers or in the evaluation modules of Simopticon, but this has not been done yet. Both for internal or individual stability as well as string stability such conditions could be included.

For example, the *PATH* controller is string stable for $\xi \geq 1$ and $C_1 < 1$ and the spacing error converges to zero [7]. These are simple conditions that can be taken into account during optimization by limiting the parameter bounds appropriate to that. In principle, such stability conditions can be implemented as part of either the optimizer or evaluation modules, so that parameter combinations that do not fulfill them can be discarded. If they are implemented as part of the optimizer, they can be checked before starting a simulation, improving optimization performance. But note that sufficient (but not necessary) conditions might exclude the best solutions regarding control performance from the parameter space. As there are many string stability definitions, in the best case *all* of them should be checked.

6.3. Comparison before optimization

In Plexe, all controllers have default parameters. The control performance ϵ_{MCMSE} for each controller using the default settings is shown in Table 7 on the left-hand side. This is the sum of the metrics for both scenarios, i.e., the sinusoidal scenario and the braking scenario. As Plexe contains two configurations of the *ACC* controller (only differing in the value of the headway h), the one using $h = 1.2\,\text{s}$ has been used, because only this one is string stable. The corresponding inter-vehicle distances are visualized in Figs. 6 and 8 for the sinusoidal and braking scenario, respectively, and the corresponding control errors in Figs. 7 and 9. Inter-vehicle distance and control error show a similar behavior, but the control error is more important for the optimization while the distances make the influence of the controller-dependent reference distance more visible.

Since the reference distance also has an influence on the control performance (at least for the controllers requiring $h > 0$), a comparison between different controllers based on different reference distances is not fair.

6.4. Optimization

As already mentioned, the behavior of some controllers depends on the reference distance (spacing policy). This is especially true for the *ACC* controller which is only string stable for a time headway larger than 1 s [7]. Therefore, two reference distances (i.e., setpoints) have been selected to show optimization results for these distances. The first distance is selected as 5 m, because that is the default distance in Plexe for the controllers *PATH* and *Flatbed*. The second reference distance is set to 35 m, because this allows a headway larger than 1 s for the *ACC* controller.

Besides the controller parameters, also the reference distance could be optimized instead of showing two simulation studies for a different reference distance. However, the control performance of *CACC*, *Flatbed*, and *Yan* can be expected to be mostly independent of the reference distance, that is, only slightly influenced by numeric aspects and an increased packet loss rate for a higher reference distance, as confirmed by the following simulations. On the other hand, for *ACC* and *Ploeg* the control performance massively improves with increasing reference

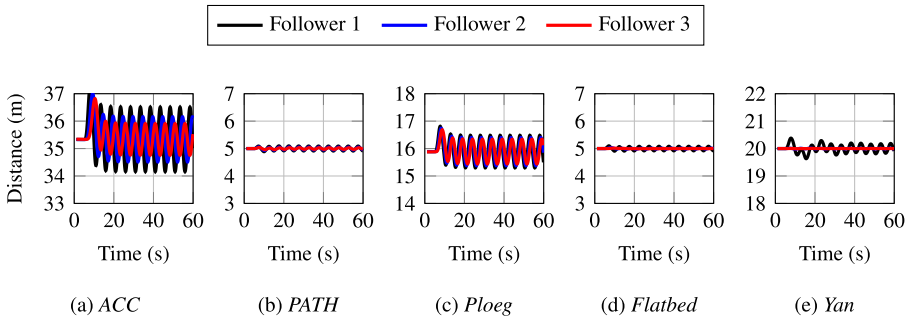


Fig. 6. Simulation results of unoptimized controllers for the original reference distance and the sinusoidal scenario. Shown is the distance in front of each vehicle to its predecessor.

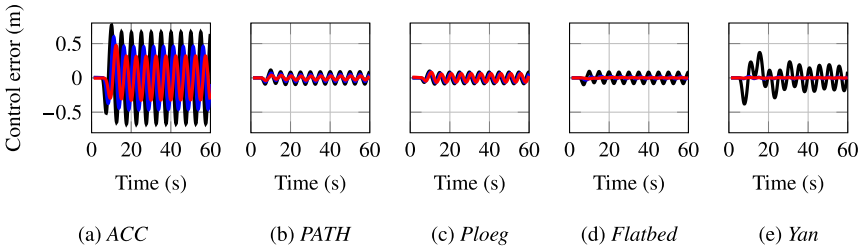


Fig. 7. Simulation results of unoptimized controllers for the original reference distance and the sinusoidal scenario. Shown is control error (i.e., the difference between the reference distance and the real distance in front of each vehicle to its predecessor).

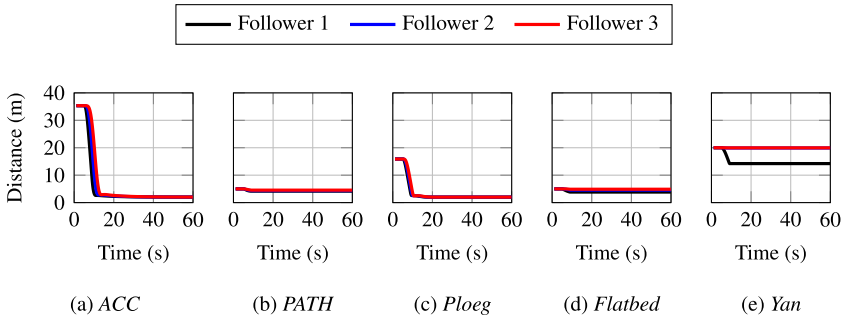


Fig. 8. Simulation results of unoptimized controllers for the original reference distance and the braking scenario. Shown is the distance in front of each vehicle to its predecessor.

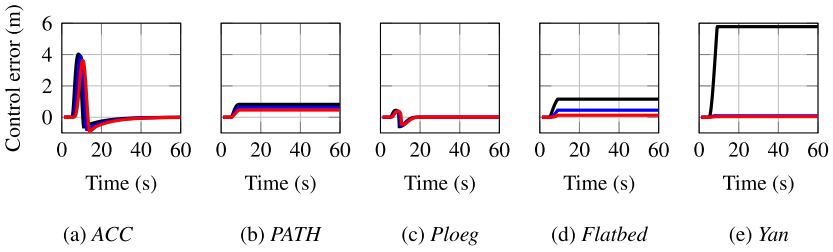


Fig. 9. Simulation results of unoptimized controllers for the original reference distance and the braking scenario. Shown is control error (i.e., the difference between the reference distance and the real distance in front of each vehicle to its predecessor).

distance (see Section 6.5), only limited by increasing packet losses at large distances for the *Ploeg* controller (not for *ACC* as this does not use communication). But having a very large gap between the vehicles is contrary to the main goals of platooning. So, if the reference distance is optimized automatically, some partial metric for penalizing large distances should be added. Its weighting, though, would be very subjective; therefore, this is out of scope of this article.

Even the selection of a controller type could be part of the optimization strategy. This is not done in our optimization study because we want to show the different optimization results for the different controllers. Nevertheless, if the optimizer would also optimize the controller type, it should deliver the controller with the lowest ϵ_{MCMSE} of our simulation study, depending on the reference distance. For doing that, an extension of the optimization framework would be beneficial because the controllers have different parameters so that the parameter space is much larger if several controllers are taken into account, but for each controller only a few parameters have an influence on the control performance. Considering the efficiency of such optimization tasks is not implemented in Simopticon yet.

Several controllers (*ACC* and *Ploeg* in Plexe) require a headway $h > 0$, so that the reference distance is not constant but depends on the speed. In order to make such controller types comparable, the reference

Table 8
Optimized parameters. With respect to Eq. (14), reference distance 5 m means a fixed distance of $d_0 = 5$ m and $h = 0$ for controllers working with a constant reference distance (*PATH*, *Flatbed*, and *Yan*) as well as $d_0 = 2$ m and $h = 0.108$ s for controllers that require a nonzero headway (*ACC* and *Ploeg*). Reference distance 35 m means $d_0 = 35$ m and $h = 0$ for controllers working with a constant reference distance as well as $d_0 = 2$ m and $h = 1.188$ s for controllers that require a nonzero headway.

Controller	Parameter	Original value	Parameter limits	Optimum (ref. 5 m)	Optimum (ref. 35 m)
ACC	λ	0.1 s^{-1}	0 … 2000 s^{-1}	3.755 91 s^{-1}	1595.90 s^{-1}
PATH	C_1	0.5	0 … 1	0.842	0.842
	ξ	1	1 … 1000	115.35	115.35
	ω_n	0.2 Hz	0 … 2 Hz	0.0143 Hz	0.0143 Hz
Ploeg	k_p	0.2 s^{-2}	0 … 100 000 s^{-2}	48 038.8 s^{-2}	59 682.7 s^{-2}
	k_d	0.7 s^{-1}	0 … 100 000 s^{-1}	7494.06 s^{-1}	6972.18 s^{-1}
Flatbed	K_a	2.4	0 … 80	9.41	9.41
	K_v	0.6 s^{-1}	0 … 1000 s^{-1}	139.48 s^{-1}	139.48 s^{-1}
	K_p	12.0 s^{-2}	500 … 1000 s^{-2}	628.70 s^{-2}	628.70 s^{-2}
	h_{des}	4.0 s	0 … 20 s	1.523 s	1.523 s
Yan	β	1.2 s	0 … 200 s	182.062 s	182.062 s
	γ	2.5 s^2	0 … 50 s^2	37.4966 s^2	37.4966 s^2
	κ	1.2	0 … 200	135.464	135.464
	η	1.1 s	0 … 10 s	1.265 31 s	1.265 31 s
	Ξ	6.5 s^2	0 … 10 s^2	0.060 078 9 s^2	0.060 078 9 s^2

distance d_0 is set to 2 m (fitting to the Plexe value for the *ACC* and *Ploeg* controllers) and the time headway h such that the reference distance at the mean speed is 5 m and 35 m, respectively. Assuming a mean speed of 100 km/h, this results in a time headway h of 0.108 s and 1.188 s, respectively.

The parameter limits have been selected in several manual trials so that the found optimum does not lie at the bounds of the limits: If, after an optimization run, the optimum was found at one of the limits, the parameter range was increased and the optimization was started again.

For the optimization, the Monte Carlo optimizer is used with (at least) 1000 evaluations (i.e., random parameter combinations) for each controller. Each evaluation contains 2 simulations (both of the scenarios) without repetitions. The used metric is $\varepsilon_{\text{MCMSE}}$ as defined in (2). The optimized parameters for both reference distances of 5 m and 35 m can be found in Table 8. Roughly 5 to 8 h were necessary to run 1000 evaluations, depending on the used computer resources.

For one example (the optimization of the *Flatbed* controller for a reference distance of 35 m) the progression of the best found solution (the performance measure $\varepsilon_{\text{MCMSE}}$) as a function of the evaluation number is shown in Fig. 10 for all three optimization strategies. In this example, different to the test functions discussed in Section 5, Simopticon-Direct converges slower than the other two optimizers. The reason might be the large number of local optima compared to the test functions. As stated above, in general only 1000 evaluations have been used — although it can be seen that the optimum improves significantly also later. The reason for taking 1000 evaluations is mainly the processing time together with the fact that it is never clear if the global optimum has been reached or not, even for extremely high iteration numbers. Besides, it should be emphasized that this is only a case study and does not claim to deliver perfect results.

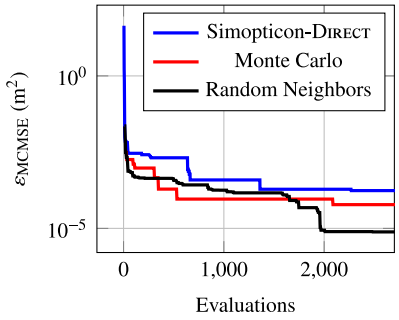


Fig. 10. Comparison of three optimization algorithms for the optimization of the *Flatbed* controller for a reference distance of 35 m. Shown is the evaluation of the current optimum as a function of the number of evaluations.

6.5. Comparison after optimization

The performance evaluation for each of the optimized controllers is shown in Table 7 for a reference distance of 5 m and for a reference distance of 35 m, respectively. According to the $\varepsilon_{\text{MCMSE}}$ values, the *Flatbed* controller seems to be the best controller for 5 m, and the *Ploeg* controller for 35 m.

We will first analyze the case with a reference distance of 5 m. The distance between the vehicles as a function of time and the control error are shown in Fig. 11 for the sinusoidal scenario and in Fig. 13 for the braking scenario, respectively.

In the sinusoidal scenario, the *Flatbed* controller shows only distance errors up to 1.6 cm, which is negligible from a practical point of view, taking into consideration the reference distance of 5 m (the error is lower than 0.3% of the reference distance). The *PATH* controller shows distance errors up to 6 cm, where the distance error is for the first follower much larger than for the subsequent vehicles in the platoon. For the *Ploeg* controller there are oscillations with an amplitude larger than 10 cm that result from the time headway $h > 0$, i.e., from the speed-dependent reference distance. To make the control error more comparable it is reasonable to plot the control error instead of the distance (lower row of Fig. 11). However, this hides the obvious issue that a constant target distance is not met by this type of controller. The *Yan* controller has distance errors up to 1.6 cm. The worst behavior can be seen for the *ACC* controller. This simulation run contains a vehicle collision at the time instance where the distance reaches zero. Plexe finishes the simulation also in this case, but this is not (yet) considered by the evaluation script. This could be fixed relatively simply by checking the simulation duration or the minimum distance in the evaluation code, but this case occurs only if the optimizer finds no parameter setting with acceptable performance like for the *PATH*, *Ploeg*, *Flatbed*, or *Yan* controllers. So even if that fix would be made, comparable results to the other controllers cannot be expected. The main problem is that the *ACC* controller is only string stable for a time headway larger than 1 s – which is not fulfilled for a reference distance of 5 m and a mean speed of 100 km/h.

In the braking scenario the distance errors of the *Flatbed* controller are even below 0.7 cm. The *ACC* controller shows a vehicle collision here as well. The difference between controllers with a fixed reference distance and controllers with a nonzero time headway is more obvious here – while the final vehicle distance for the former controllers is roughly 5 m, for the latter ones it is roughly 2 m, according to the

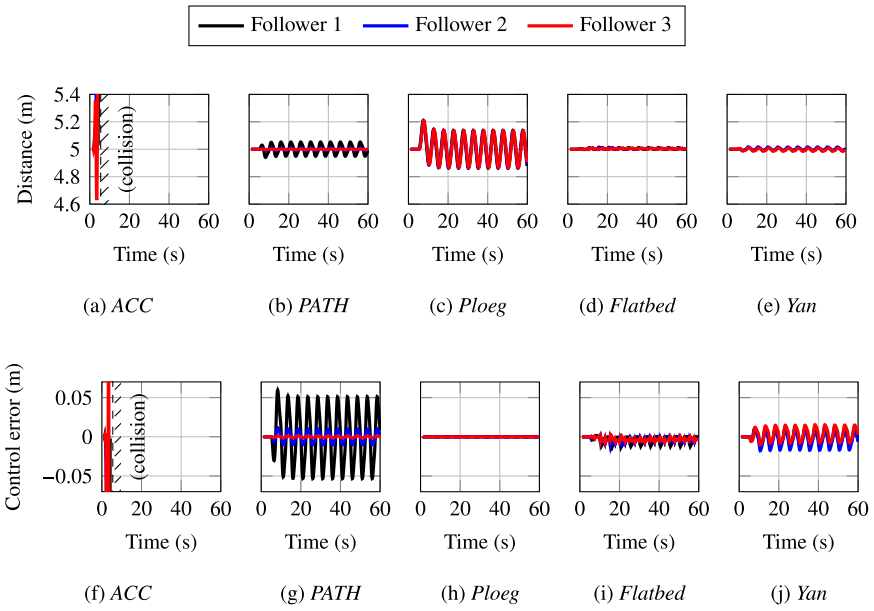


Fig. 11. Simulation results of optimized controllers for a reference distance of 5 m and the sinusoidal scenario. Shown is the distance in front of each vehicle to its predecessor (top) and control error (bottom).

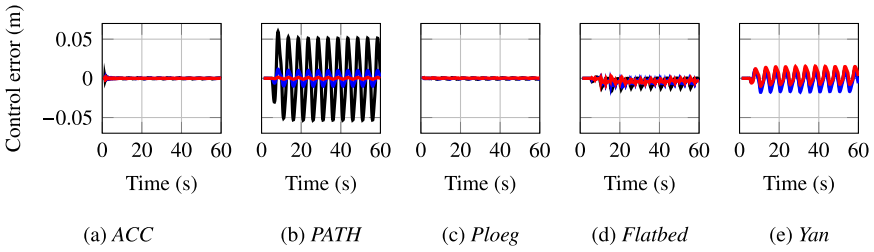


Fig. 12. Simulation results of optimized controllers for a reference distance of 35 m and the sinusoidal scenario. Shown is the control error.

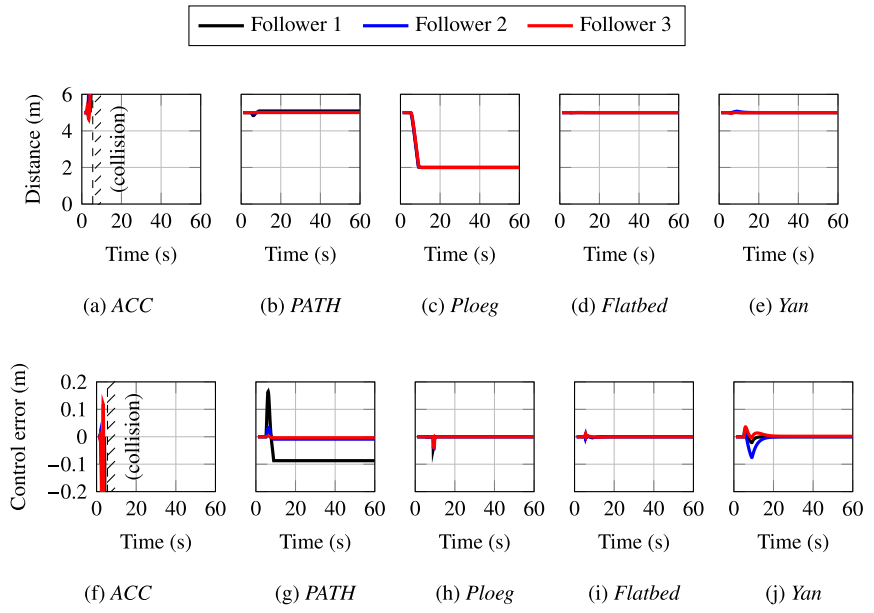


Fig. 13. Simulation results of optimized controllers for a reference distance of 5 m and the braking scenario. Shown is the distance in front of each vehicle to its predecessor (top) and control error (bottom).

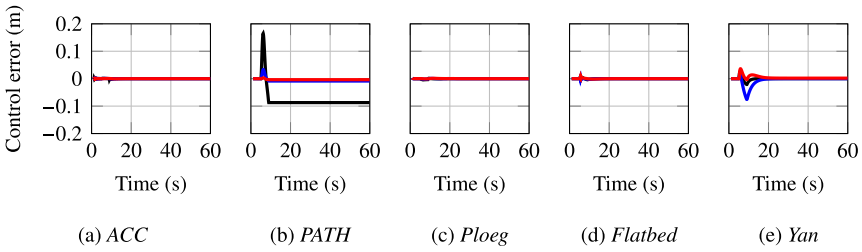


Fig. 14. Simulation results of optimized controllers for a reference distance of 35 m and the braking scenario. Shown is the control error.

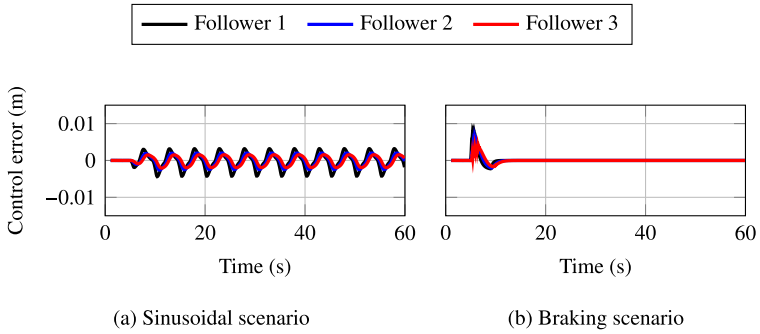


Fig. 15. Simulation results of optimized flatbed controller for a reference distance of 5 m. Shown is the control error.

Table 9
Optimization of the *Flatbed* controller with the *Random Neighbors* strategy for a reference distance of 5 m and increased number of evaluations.

Parameter	Original value	Parameter limits	Optimum
K_a	2.4	0 … 80	3.897 27
K_v	0.6 s^{-1}	0 … 1000 s^{-1}	92.5263 s^{-1}
K_p	12.0 s^{-2}	500 … 1000 s^{-2}	916.569 s^{-2}
h_{des}	4.0 s	0 … 20 s	0.463 197 s

different values of the stand-still distance d_0.

The simulation results for a reference distance of 35 m are shown in Figs. 12 and 14. For the controllers *PATH*, *Flatbed* and *Yan*, the simulation results are nearly identical to those for the 5 m case (with the only difference that the distance oscillates around 35 m instead of 5 m), also resulting in nearly identical ϵ_{MCMSE} values in Table 7. The reason is that the control algorithms work independent of the absolute reference distance: only relative errors to that reference distance are taken into account. The conclusion of this (expected) result is that it is possible to decrease the reference distance further without degrading the control performance or a risk of vehicle collisions.

In contrast to the other controllers, the *ACC* and *Ploeg* controllers work much better for 35 m than for 5 m. This can be seen most clearly in Table 7 and in the braking scenario (Fig. 14 compared to Fig. 13). The performance measures are even better than that for the other controllers, but the oscillations around the reference distance of 35 m are high due to the velocity-dependent spacing policy. Due to the large reference distance, there are no vehicle collisions at all: for none of the controllers and neither for the sinusoidal nor for the braking scenario.

Due to the very good behavior of the *Flatbed* controller for the 5 m reference case, this controller is optimized with an extended number of evaluations (see also Fig. 10). The parameters after optimization are shown in Table 9, the performance is $\epsilon_{MCMSE} = 7.207{\times}10^{-6}\,m^2$. The simulation using the best found parameter set is shown in Fig. 15. Distance errors less than 1 cm are reached in both scenarios (sinusoidal and braking scenario).

7. Conclusion

In this article we demonstrated the benefits of a methodology for parameter selection that encompasses all of: an evaluator employing a common metric, a simulator component, and an optimizer – along with an open-source reference implementation. We also discussed the trade-offs of different optimization algorithms, both from the literature and custom-built, for parameter optimization of platooning controllers.

While the state of the art regarding platoon controllers focuses on stability proofs, the optimization presented in this article delivers a concrete set of parameters that minimizes a defined cost function. The case studies with Plexe showed that all integrated controllers have a lot of potential for optimization compared to their default parameterization. The case study showed also that the type of optimization strategy matters. A clear outcome of the case study is also that the reference distance has an enormous influence on the control quality of some of the controllers, thus also the selection of a control strategy should depend on the reference distance.

The article and the provided open-source software lays the foundation for the optimization of more platoon controllers and a fair comparison of controllers. Future work will also focus on optimization in the presence of packet losses and other stochastic model parts as well as the influence of network properties.

CRediT authorship contribution statement

Per Natzschka: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **Burkhard Hensel:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Methodology, Investigation, Data curation, Conceptualization. **Christoph Sommer:** Writing – review & editing, Visualization, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We share the reference implementation of the optimization framework as open-source software. It was used to generate all discussed results and can be used to reproduce them. The software is available at cms-labs.org/research/software/simopticon/. Its output is available at zenodo.org/records/13828790 [66].

References

[1] V.A. Sujan, P. Jones, A. Siekmann, Heavy duty commercial vehicle platooning energy benefits for conventional and electrified powertrains, in: 2022 International Conference on Connected Vehicle and Expo, ICCVE, IEEE, Lakeland, USA, 2022, http://dx.doi.org/10.1109/ICCVE52871.2022.9742749.

[2] Y. Han, T. Kawasaki, S. Hanaoka, The benefits of truck platooning with an increasing market penetration: A case study in Japan, MDPI Sustain. 14 (15) (2022) http://dx.doi.org/10.3390/su14159351.

[3] C. Zhao, X. Duan, L. Cai, P. Cheng, Vehicle platooning with non-ideal communication networks, IEEE Trans. Veh. Technol. 70 (1) (2021) 18–32, http://dx.doi.org/10.1109/TVT.2020.3046165.

[4] A.A. Hussein, H.A. Rakha, Vehicle platooning impact on drag coefficients and energy/fuel saving implications, IEEE Trans. Veh. Technol. 71 (2) (2022) 1199–1208, http://dx.doi.org/10.1109/TVT.2021.3131305.

[5] S. Sheikholeslam, C.A. Desoer, Longitudinal control of a platoon of vehicles, in: 1990 American Control Conference (ACC 1990), IEEE, San Diego, USA, 1990, pp. 291–296, http://dx.doi.org/10.23919/ACC.1990.4790743.

[6] D. Swaroop, J.K. Hedrick, C.C. Chien, P. Ioannou, A comparision of spacing and headway control laws for automatically controlled vehicles, Inf. Veh. Syst. Dyn. 23 (1) (1994) 597–625, http://dx.doi.org/10.1080/00423119408969077.

[7] R. Rajamani, Vehicle Dynamics and Control, second ed., Springer, New York, 2012, http://dx.doi.org/10.1007/978-1-4614-1433-9.

[8] M. Yan, Y. Tang, P. Yang, Consensus based control algorithm for nonlinear vehicle platoons in the presence of time delays and packet losses, in: 2018 Chinese Automation Congress, CAC 2018, IEEE, Xi'an, China, 2018, pp. 3339–3344, http://dx.doi.org/10.1109/CAC.2018.8623770.

[9] S. Santini, A. Salvi, A.S. Valente, A. Pescapé, M. Segata, R. Lo Cigno, A consensus-based approach for platooning with intervehicular communications and its validation in realistic scenarios, IEEE Trans. Veh. Technol. 66 (3) (2017) 1985–1999, http://dx.doi.org/10.1109/TVT.2016.2585018.

[10] Z. Wang, Y. Gao, C. Fang, L. Liu, H. Zhou, H. Zhang, Optimal control design for connected cruise control with stochastic communication delays, IEEE Trans. Veh. Technol. 69 (12) (2020) 15357–15369, http://dx.doi.org/10.1109/TVT.2020.3040321.

[11] H. Zeng, Z. Ye, D. Zhang, Robust model predictive control based cooperative control of uncertain connected vehicle platoon systems, in: 9th International Conference on Control, Automation and Robotics, ICCAR 2023, IEEE, Beijing, China, 2023, pp. 256–261, http://dx.doi.org/10.1109/ICCAR57134.2023.10151715.

[12] J. Wang, X. Li, J.H. Park, G. Guo, Distributed MPC-based string stable platoon control of networked vehicle systems, IEEE Trans. Intell. Transp. Syst. 24 (3) (2023) 3078–3090, http://dx.doi.org/10.1109/TITS.2022.3221382.

[13] M.P. Aghababa, M. Saif, B. Shafai, Adaptive control of a vehicular platoon with unknown parameters and input variations, in: 21st IFAC World Congress, Vol. 53, Berlin, Germany, 2020, pp. 13876–13881, http://dx.doi.org/10.1016/j.ifacol.2020.12.900.

[14] M. Li, B. Wang, S. Wang, Z. Ke, Serial distributed reinforcement learning for enhanced multi-objective platoon control in curved road coordinates, Elsevier Expert. Syst. Appl. 269 (2025) http://dx.doi.org/10.1016/j.eswa.2025.126493.

[15] D. Liu, S. Mair, K. Yang, S. Baldi, P. Frasca, M. Althoff, Resilience in platoons of cooperative heterogeneous vehicles: Self-organization strategies and provably-correct design, IEEE Trans. Intell. Veh. 9 (1) (2024) 2262–2275, http://dx.doi.org/10.1109/tiv.2023.3317977.

[16] M. Segata, S. Joerer, B. Bloessl, C. Sommer, F. Dressler, R.L. Cigno, Plexe: A platooning extension for Veins, in: 2014 IEEE Vehicular Networking Conference, VNC 2014, IEEE, Paderborn, Germany, 2014, pp. 53–60, http://dx.doi.org/10.1109/VNC.2014.7013309.

[17] F. Acciani, P. Frasca, G. Heijenk, A.A. Stoorvogel, Stochastic string stability of vehicle platoons via cooperative adaptive cruise control with lossy communication, IEEE Trans. Intell. Transp. Syst. 23 (8) (2022) 10912–10922, http://dx.doi.org/10.1109/TITS.2021.3097199.

[18] A. Ali, G. Garcia, P. Martinet, The flatbed platoon towing model for safe and dense platooning on highways, IEEE Intell. Transp. Syst. Mag. 7 (1) (2015) 58–68, http://dx.doi.org/10.1109/MITS.2014.2328670.

[19] A. Tiganasu, C. Lazar, C.F. Caruntu, Cyber physical systems - oriented design of cooperative control for vehicle platooning, in: 2017 21st International Conference on Control Systems and Computer Science, CSCS, IEEE, Bucharest, Romania, 2017, pp. 465–470, http://dx.doi.org/10.1109/CSCS.2017.72.

[20] C. Hidalgo, R. Lattarulo, C. Flores, J. Pérez Rastelli, Platoon merging approach based on hybrid trajectory planning and CACC strategies, MDPI Sens. 21 (8) (2021) 2626, http://dx.doi.org/10.3390/s21082626.

[21] R. Xu, Y. Guo, X. Han, X. Xia, H. Xiang, J. Ma, OpenCDA: An open cooperative driving automation framework integrated with co-simulation, in: 2021 IEEE International Intelligent Transportation Systems Conference, ITSC, IEEE, 2021, pp. 1155–1162, http://dx.doi.org/10.1109/ITSC48978.2021.9564825.

[22] R. Zhang, A. Abraham, S. Dasgupta, J. Dauwels, Constrained model predictive control using kinematic model of vehicle platooning in VISSIM simulator, in: 2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV, IEEE, Singapore, 2018, pp. 721–726, http://dx.doi.org/10.1109/ICARCV.2018.8581314.

[23] X. Song, L. Chen, K. Wang, D. He, Robust time-delay feedback control of vehicular CACC systems with uncertain dynamics, MDPI Sens. 20 (6) (2020) http://dx.doi.org/10.3390/s20061775.

[24] S. Hallé, B. Chaib-draa, A collaborative driving system based on multiagent modelling and simulations, Elsevier Transp. Res. Part C: Emerg. Technol. 13 (4) (2005) 320–345, http://dx.doi.org/10.1016/j.trc.2005.07.004.

[25] A. Soua, O. Shagdar, J.-M. Lasgouttes, Toward efficient simulation platform for platoon communication in large scale C-ITS scenarios, in: 2018 International Symposium on Networks, Computers and Communications, ISNCC, IEEE, Rome, Italy, 2018, http://dx.doi.org/10.1109/ISNCC.2018.8530962.

[26] F. Liu, Y. Long, H. Zhou, Analysis of autonomous vehicle platoon with disturbance based on PLEXE, in: 2019 Chinese Automation Congress, CAC 2019, IEEE, Hangzhou, China, 2019, pp. 134–139, http://dx.doi.org/10.1109/CAC48633.2019.8996256.

[27] S. Hasan, A. Balador, S. Girs, E. Uhlemann, Towards emergency braking as a fail-safe state in platooning: A simulative approach, in: 90th IEEE Vehicular Technology Conference, VTC2019-Fall, IEEE, Honolulu, Hawaii, 2019, http://dx.doi.org/10.1109/VTCFall.2019.8891254.

[28] A. Elahi, A. Alfi, H. Modares, H_∞ consensus of homogeneous vehicular platooning systems with packet dropout and communication delay, IEEE Trans. Syst. Man Cybern.: Syst. 52 (6) (2022) 3680–3691, http://dx.doi.org/10.1109/TSMC.2021.3071994.

[29] J.W. Bentley, P. Snitzer, E. Stegner, D.M. Bevly, M. Hoffman, Comparing the Performance of Different Heavy Duty Platooning Control Strategies, Technical Paper 2023-01-0895, SAE, 2023, http://dx.doi.org/10.4271/2023-01-0895.

[30] S. Hasan, M.A. Al Ahad, I. Sljivo, A. Balador, S. Girs, E. Lisova, A fault-tolerant controller manager for platooning simulation, in: 2019 IEEE International Conference on Connected Vehicles and Expo, ICCVE, IEEE, Graz, Austria, 2019, pp. 1–6, http://dx.doi.org/10.1109/ICCVE45908.2019.8965220.

[31] M. Segata, R.L. Cigno, T. Hardes, J. Heinovski, M. Schettler, B. Bloessl, C. Sommer, F. Dressler, Multi-technology cooperative driving: An analysis based on PLEXE, IEEE Trans. Mob. Comput. 22 (8) (2023) 4792–4806, http://dx.doi.org/10.1109/TMC.2022.3154643.

[32] D. Subramanian, J.F. Pekny, G.V. Reklaitis, A simulation–optimization framework for addressing combinatorial and stochastic aspects of an R&D pipeline management problem, Elsevier Comput. Chem. Eng. 24 (2) (2000) 1005–1011, http://dx.doi.org/10.1016/S0098-1354(00)00535-4.

[33] X. Wan, S. Orçun, J.F. Pekny, G. Reklaitis, A simulation based optimization framework to analyze and investigate complex supply chains, in: 8th International Symposium on Process Systems Engineering, Vol. 15, Elsevier, Kunming, China, 2003, pp. 630–635, http://dx.doi.org/10.1016/S1570-7946(03)80615-6.

[34] Y. Pan, M. Zhou, Z. Chen, H. Tan, A simulation optimization framework for shipment planning at RDC considering time and quantity consolidation with uncertain demands, in: International Conference on Services Systems and Services Management, ICSSSM11, IEEE, Tianjin, China, 2011, http://dx.doi.org/10.1109/ICSSSM.2011.5959536.

[35] E.N. Pérez, Y.M. Méndez-Vázquez, M.C. Ríos, A Simulation-Optimization strategy to deal simultaneously with tens of decision variables and multiple performance measures in manufacturing, in: 2015 Winter Simulation Conference, WSC, IEEE, Huntington Beach, CA, USA, 2015, pp. 2295–2306, http://dx.doi.org/10.1109/WSC.2015.7408341.

[36] M. Carillo, G. Cordasco, F. Serrapica, V. Scarano, C. Spagnuolo, P. Szufel, SOF: Zero configuration simulation optimization framework on the cloud, in: 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP, IEEE, Heraklion, Greece, 2016, pp. 341–344, http://dx.doi.org/10.1109/PDP.2016.22.

[37] L. Atorf, C. Schorn, J. Rossmann, C. Schlette, A framework for simulation-based optimization demonstrated on reconfigurable robot workcells, in: 2017 IEEE International Systems Engineering Symposium, ISSE, IEEE, Vienna, Austria, 2017, pp. 1–6, http://dx.doi.org/10.1109/SysEng.2017.8088278.

[38] B. Hensel, K. Kabitzsch, Softwareunterstützung zur effizienten parameterbestimmung für modulare modelle, in: 5. Kolloquium des SFB/Transregio 96, Chemnitz, Germany, 2017, pp. 185–196.

[39] B. Hensel, S. Schroeder, K. Kabitzsch, New coordination software for parameter identification applied to thermal models of an actuator strut, Hindawi J. Comput. Eng. 2017 (2017) 1–14, http://dx.doi.org/10.1155/2017/3169785.

[40] B. Hensel, S. Schroeder, K. Kabitzsch, Parameter identification software for various thermal model types, in: 1st Conference on Thermal Issues in Machine Tools, Dresden, Germany, 2018, pp. 23–33.

[41] B. Hensel, K. Kabitzsch, Software supporting parameter optimization of finite element models, in: 20th UKSim-AMSS International Conference on Computer Modelling and Simulation, UKSim 2018, IEEE, Cambridge, United Kingdom, 2018, pp. 55–60, http://dx.doi.org/10.1109/UKSim.2018.00022.

[42] X. Feng, Z. Zhao, C. Zhang, Simulation optimization framework for online deployment and adjustment of reconfigurable machines in job shops, in: 2020 IEEE International Conference on Industrial Engineering and Engineering Management, IEEM, IEEE, Singapore, 2020, pp. 731–735, http://dx.doi.org/10.1109/IEEM45057.2020.9309782.

[43] M. Gillis, R. Urban, A. Saif, N. Kamal, M. Murphy, A simulation–optimization framework for optimizing response strategies to epidemics, Elsevier Oper. Res. Perspect. 8 (2021) http://dx.doi.org/10.1016/j.orp.2021.100210.

[44] K. Miettinen, Introduction to multiobjective optimization: Noninteractive approaches, in: J. Branke, K. Deb, K. Miettinen, R. Słowiński (Eds.), Multiobjective Optimization, Springer, 2008, pp. 1–26, http://dx.doi.org/10.1007/978-3-540-88908-3_1.

[45] D.R. Jones, J.R.R.A. Martins, The DIRECT algorithm: 25 years Later, Springer J. Glob. Optim. 79 (3) (2020) 521–566, http://dx.doi.org/10.1007/s10898-020-00952-6.

[46] L. Stripinis, R. Paulavičius, DIRECTGO: A new DIRECT-type MATLAB toolbox for derivative-free global optimization, ACM Trans. Math. Software 48 (4) (2022) 41:1–41:46, http://dx.doi.org/10.1145/3559755.

[47] D.R. Jones, C.D. Perttunen, B.E. Stuckman, Lipschitzian optimization without the Lipschitz constant, Springer J. Optim. Theory Appl. 79 (1) (1993) 157–181, http://dx.doi.org/10.1007/BF00941892.

[48] L.M. Rios, N.V. Sahinidis, Derivative-free optimization: a review of algorithms and comparison of software implementations, Springer J. Glob. Optim. 56 (3) (2012) 1247–1293, http://dx.doi.org/10.1007/s10898-012-9951-y.

[49] Q. Liu, J. Zeng, G. Yang, MrDIRECT: a multilevel robust DIRECT algorithm for global optimization problems, Springer J. Glob. Optim. 62 (2) (2014) 205–227, http://dx.doi.org/10.1007/s10898-014-0241-8.

[50] Q. Liu, G. Yang, Z. Zhang, J. Zeng, Improving the convergence rate of the DIRECT global optimization algorithm, Springer J. Glob. Optim. 67 (4) (2016) 851–872, http://dx.doi.org/10.1007/s10898-016-0447-z.

[51] Y.D. Sergeyev, D.E. Kvasov, Global search based on efficient diagonal partitions and a set of Lipschitz constants, SIAM J. Optim. 16 (3) (2006) 910–937, http://dx.doi.org/10.1137/040621132.

[52] H. Liu, S. Xu, X. Wang, J. Wu, Y. Song, A global optimization algorithm for simulation-based problems via the extended DIRECT scheme, Taylor Fr. Eng. Optim. 47 (11) (2015) 1441–1458, http://dx.doi.org/10.1080/0305215X.2014.971777.

[53] J. Gablonsky, C. Kelley, A locally-biased form of the DIRECT algorithm, Springer J. Glob. Optim. 21 (1) (2001) 27–37, http://dx.doi.org/10.1023/A:1017930332101.

[54] D. Finkel, C.T. Kelley, An Adaptive Restart Implementation of DIRECT, Technical Report CRSC-TR04-30, North Carolina State University. Center for Research in Scientific Computation, Raleigh, 2004.

[55] J. Mockus, On the Pareto optimality in the context of Lipschitzian optimization, Vilnius Univ. Press Inform. 22 (4) (2011) 521–536, http://dx.doi.org/10.15388/Informatica.2011.340.

[56] L. Stripinis, R. Paulavičius, J. Žilinskas, Improved scheme for selection of potentially optimal hyper-rectangles in DIRECT, Springer Optim. Lett. 12 (7) (2018) 1699–1712, http://dx.doi.org/10.1007/s11590-017-1228-4.

[57] G. Liuzzi, S. Lucidi, V. Piccialli, A DIRECT-based approach exploiting local minimizations for the solution of large-scale global optimization problems, Springer Comput. Optim. Appl. 45 (2) (2010) 353–375, http://dx.doi.org/10.1007/s10589-008-9217-2.

[58] A. Tavassoli, K. Haji Hajikolaei, S. Sadeqi, G.G. Wang, E. Kjeang, Modification of DIRECT for high-dimensional design problems, Taylor Fr. Eng. Optim. 46 (6) (2014) 810–823, http://dx.doi.org/10.1080/0305215X.2013.800057.

[59] D.P. Kroese, T. Taimre, Z.I. Botev, Handbook of Monte Carlo Methods, Wiley, Hoboken, NJ, 2011, http://dx.doi.org/10.1002/9781118014967.

[60] M.J. Kochenderfer, T.A. Wheeler, Algorithms for Optimization, The MIT Press, Cambridge, Massachusetts, 2019.

[61] P.A. Ioannou, C.C. Chien, Autonomous intelligent cruise control, IEEE Trans. Veh. Technol. 42 (4) (1993) 657–672, http://dx.doi.org/10.1109/25.260745.

[62] R. Rajamani, C. Zhu, Semi-autonomous adaptive cruise control systems, in: 1999 American Control Conference, ACC 1999, IEEE, San Diego, California, USA, 1999, pp. 1491–1495, http://dx.doi.org/10.1109/ACC.1999.783618.

[63] J. Ploeg, B.T.M. Scheepers, E. van Nunen, N. van de Wouw, H. Nijmeijer, Design and experimental evaluation of cooperative adaptive cruise control, in: 14th International IEEE Conference on Intelligent Transportation Systems, ITSC 2011, IEEE, Washington, DC, USA, 2011, pp. 260–265, http://dx.doi.org/10.1109/ITSC.2011.6082981.

[64] S.E. Li, Y. Zheng, K. Li, Y. Wu, J.K. Hedrick, F. Gao, H. Zhang, Dynamical modeling and distributed control of connected and automated vehicles: Challenges and opportunities, IEEE Intell. Transp. Syst. Mag. 9 (3) (2017) 46–58, http://dx.doi.org/10.1109/MITS.2017.2709781.

[65] S. Feng, Y. Zhang, S.E. Li, Z. Cao, H.X. Liu, L. Li, String stability for vehicular platoon control: Definitions and analysis methods, Elsevier Annu. Rev. Control. 47 (2019) 81–97, http://dx.doi.org/10.1016/j.arcontrol.2019.03.001.

[66] B. Hensel, P. Natzschka, P. Hempel, Simopticon - Software and Simulations [Dataset], Zenodo, 2024, http://dx.doi.org/10.5281/zenodo.13828790, URL: https://zenodo.org/records/13828790.