Feasibility Study and Benchmarking of Embedded MPC for Vehicle Platoons

Iñaki Martín Soroa
Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
i.martin.soroa@studenten.tue.nl

Amr Ibrahim
Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
a.ibrahim@tue.nl

Dip Goswami
Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
d.goswami@tue.nl

Hong Li
Car Infotainment and Driving Assistance, NXP Semiconductor, Eindhoven, The Netherlands
Hong.r.li@nxp.nl

Abstract

This paper performs a feasibility analysis of deploying Model Predictive Control (MPC) for vehicle platooning on an On-Board Unit (OBU) and performance benchmarking considering interference from other (system) tasks running on an OBU. MPC is a control strategy that solves an implicit (on-line) or explicit (off-line) optimisation problem for computing the control input in every sample. OBUs have limited computational resources. The challenge is to implement an MPC algorithm on such automotive Electronic Control Units (ECUs) with an acceptable timing behavior. Moreover, we should be able to stop the execution if necessary at the cost of performance.

We measured the computational capability of a unit developed by Cohda Wireless and NXP under the influence of its Operating System (OS). Next, we analysed the computational requirements of different state-of-the-art MPC algorithms by estimating their execution times. We use off-the-shelf and free automatic code generators for MPC to run a number of relevant MPC algorithms on the platform. From the results, we conclude that it is feasible to implement MPC on automotive ECUs for vehicle platooning and we further benchmark their performance in terms of MPC parameters such as prediction horizon and system dimension.

2012 ACM Subject Classification Applied computing → Command and control; Computer systems organization → Embedded systems

Keywords and phrases Model predictive control, vehicle platoon, embedded implementation, code generation

Digital Object Identifier 10.4230/OASIcs.ASD.2019.2

Funding This work is partially supported by the H2020 projects I-MECH (GA no. 737453) and oCPS (GA no. 674875).

1 Introduction

Vehicle platooning is an application based on Cooperative Adaptive Cruise Control (CACC) technology, which is an extension of Adaptive Cruise Control (ACC). In ACC, the vehicle senses the position of the preceding vehicle and adapts the speed to avoid a collision. CACC introduces V2V messages between different vehicles. These messages have much richer information including position, speed, acceleration or road intersection status among others. The richer information allows the vehicles to react faster to sudden changes in the preceding vehicles and therefore, the distance between the vehicles can be reduced which enables to achieve better fuel efficiency and road capacity [21].

Model Predictive Control (MPC) is an optimal control strategy capable of satisfying constraints on the states of the system (plant) and the control input. The main challenge of MPC is its high computational requirements since it requires to solve an optimisation problem at every time step (sample) [7]. The MPC technology is extensively used in the chemical industry where the dynamics is generally slower. With the advent of powerful computing abilities of modern processors, MPC is making its way into other sectors such as the automotive industry [10]. One of the applications of MPC in the automotive industry is vehicle platooning. MPC has already been applied to vehicle platooning without explicitly considering constraints on the computational resources and the V2V communication time that is present in a real implementation [6]. In a real implementation, the ECU of a vehicle, which is an embedded device with limited resources, needs to solve the MPC optimization fast enough to meet the timing requirements imposed by the V2V communication.

MPC has already been implemented on embedded platforms successfully for different applications. In [31] the authors use a simple embedded device with an ARM processor running at 48MHz with 64kB of RAM memory. They control a system consisting of 8 states, 2 inputs and a control horizon of length 20 achieving a sampling period of 4ms. In [31], Fast Gradient Method algorithm (FGM) was used with fixed point operations and a tuned level of sub-optimality specific for the plant. When using floating point operations and decreasing the control horizon length to 15, it achieves a sampling time of 8ms. In [9], the authors achieved a sampling frequency in the kHz range using a processor with a clock frequency of 1GHz with a dedicated floating point unit. When controlling a larger system they manage to reach a sampling time of 13ms.

A number of works approached the embedded MPC problem using hardware accelerators [5] [28] [25] [17] [14] [22] usually on a Field Programmable Gate Array (FPGA). These works attempt to achieve sampling rates in the kHz range or control very large systems, while the vehicle platooning problem does not require very short sampling times nor large predictive models.

In order to solve the MPC optimisation problem an algorithm needs to be used. There are mainly two categories of MPC algorithms – explicit and implicit. In explicit MPC the solution is computed off-line and given to the controller as a look up table which usually requires large memory capacity. In implicit MPC the solution is computed on-line at each sampling period [1]. In this paper we focus on implicit MPC. Implicit MPC is the most commonly used method and there are a number well-developed state-of-the-art algorithms. Almost all of them can be classified in one of the following categories – Inner Point Method (IPM) [16], Active Set Method (ASM) [8], and (Fast) Gradient Method ((F)GM) [17] [3]. We analysed the feasibility of these algorithms with a special focus on FGM.

In order to determine if it is feasible to implement MPC for vehicle platooning on an embedded device, timing constraints must be met. The MPC algorithm needs to be able to compute the solution fast enough for a problem with similar dimensions and constraints as in vehicle platooning, described below (Section 4.2). The time available for the execution of the MPC task depends on the message rate (or sampling rate) supported by the V2V communication and the execution time of the other tasks running on the device. Ideally the execution time would be deterministic or bounded, which can be achieved for some of the state-of-the-art algorithms.

We also analyse the trade-off that needs to be made to balance the control performance and the execution time of the MPC task. We investigate the impact of the length of the control horizon (used in the MPC optimization) on the execution time, the effect of the algorithm choice and provide a number of guidelines for choosing the processor.

The rest of the paper is organised as follows. We describe the problem of vehicle platooning in Section 2. In Section 3, we analyse the characteristics and the performance of a platform suitable for being used for vehicle platooning. Next, we describe a possible implementation for vehicle platooning and we measure the overhead introduced by the other tasks that need to run on the selected platform in Section 4. We investigate a number of automatic C code generation tools available for MPC in Section 5. We analyse the computational requirements of different MPC algorithms in Section 6. Using the computational requirements of different algorithms and the performance of the platform, we provide an analytical estimation of the execution time on the selected platform in Section 7.1. We use the code generation tools to run a number of template MPC algorithms on the V2V wireless node in Section 7.2. Using the experimental and the analytical execution times we estimate the possible delays and sampling periods that can be achieved using MPC and the trade-offs that can be made, in Section 7.3. Finally we conclude in Section 8.

2 Vehicle platoons

2.1 V2V Communication and topology

The vehicle-to-vehicle communication (V2V) is performed following the standards of each country, most notably the standard of the EU, ETSI-ITS, and the standard of the USA, 1609 WAVE. Both standards are based on the IEEE 802.11p protocol stack. Under IEEE 802.11p, we can reach up to 10Hz message rate when the network usage is below 70%. The message rate can get as low as 1Hz under heavy traffic of vehicles with V2V communication devices [13].

In this paper we consider the Predecessor-Follower (PF) topology, where each vehicle receives messages from its predecessor (Fig. 1). Other topologies exist, such as Two-Predecessor-Follower (TPF) [30] and Leader-Predecessor-Follower (LPF) [30].

In Fig. 1, m_i is the message from the vehicle i including its speed, position and acceleration, and Δd_i is the error in distance (desired gap - actual gap) between the vehicles i and $i-1$.

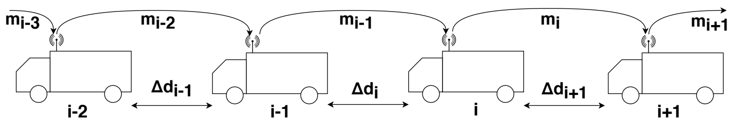

Figure 1 Predecessor-Follower topology.

2.2 Platoon model

The platoon model is distributed, each vehicle has a model of itself (vehicle model) and its relation with its predecessor (inter-vehicle dynamics).

The model of the vehicle i is obtained combining a simplified model of the longitudinal dynamics of the vehicle with the dynamics of a DC motor [27, 26], and it is given by:

$$\dot{x}_v^i = A_v^i x_v^i + B_v^i u_v^i \tag{1}$$

where A_v^i and B_v^i are the state and input matrices respectively, u_v^i is the duty cycle of the input of the motor and $x_v^i = [a^i \quad \dot{a}^i]^T$ is the state vector. Where a^i and $\dot{a}^i$ are the acceleration and the rate of change of the acceleration of the vehicle i, respectively. Moreover, the state matrix A_v^i and the input vector B_v^i of vehicle i are defined as:

$$A_v^i = \begin{pmatrix} 0 & 1 \\ \frac{-1}{\tau^i \tau_a^i} & \frac{-(\tau^i+\tau_a^i)}{\tau^i \tau_a^i} \end{pmatrix} \in \mathbb{R}^{2\times 2}, B_v^i = \begin{pmatrix} 0 \\ \frac{K^i K_a^i}{\tau^i \tau_a^i} \end{pmatrix} \in \mathbb{R}^{2\times 1}.$$

where τ^i, τ_a^i, K^i, K_a^i are model parameters of the vehicle i.

To obtain the platoon model under the PF topology, the inter-vehicle dynamics relate the vehicle i to the vehicle $i-1$. This is done by adding two new states, Δv^i and Δd^i, which represent the speed difference and the gap error between the vehicles, respectively. They are defined as $\Delta d^i = d^i - d_{des}^i$ and $\Delta v^i = v^{i-1} - v^i$, where Δd^i is the error between the actual gap (d^i) and the desired inter-vehicle gap (d_{des}^i) between the vehicle i and the vehicle $i-1$. Δv^i is the velocity error between the vehicle i and the vehicle $i-1$, where v^i denote for the velocity of the vehicle i. d^i and d_{des}^i are defined as $d_{des}^i = d_0 + \tau_h v^i$ and $d^i = q^{i-1} - q^i - L^i$, where d_0 is the gap between vehicles at standstill, τ_h is the constant headway time (the time the vehicle i needs to reach the position of the vehicle $i-1$ when $d_0 = 0$). L^i, q^i are the length and position of the vehicle i, respectively.

Combining the vehicle model with the inter-vehicle dynamics we obtain the platoon model:

$$\dot{x}_p^i = A_p^i x_p^i + B_p^i u_p^i + G_p^i a^{i-1} \qquad (2)$$

where $x_p^i = [a^i \quad \dot{a}^i \quad \Delta d^i \quad \Delta v^i]$ is the state vector, a^{i-1} is the acceleration of the preceding vehicle and A_p^i is the state matrix. The predictive model used for MPC will be obtained based on the platoon model in Eq.(2) (see Section 4.2).

3 Embedded platform: Cohda Wireless MK5 OBU

The Cohda Wireless MK5 is a platform developed by Cohda Wireless in partnership with NXP. It has been developed as a prototyping platform for V2V applications, such as CACC, and other Vehicle to Everything (V2X) applications.

3.1 Hardware

The platform has one main processor, NXP i.MX6 Dual Lite @ 800MHz (dual-core processor), paired with a communications co-processor, NXP MARS. It is equipped with 1GB of volatile memory. With a large volatile memory, memory is not a bottleneck and we are interested only in the computational power.

The platform has several ports and connectivity options. It can be connected to two 5.9GHz antennas, a GNSS antenna, μSD card, Ethernet port, CAN bus port and audio jack. On top of that it has a DC power connection.

3.2 Software

The platform uses an Ubuntu distribution of Linux as its Operating System (OS). It is not a Real-Time OS (RTOS). There are system applications available on the platform. The most relevant are the communication stacks of the EU and the USA standards.

We also use the evaluation platform reported in [30] and available in [29], which allows to quickly measure the string stability of the platoon and takes care of all the tasks required to execute CACC in a modular approach. The structure of this evaluation platform is further detailed in Section 4.

3.3 Performance evaluation

The performance is measured in millions of floating point operations per second (Mflop/s) and millions of fixed point operations per second (Mop/s). We evaluate the average performance and its distribution in percentiles.

The type of operations measured are fixed point and floating point additions, and fixed point and floating point multiplications. Most MPC algorithms use only these operations. In Table 1, the performance of the platform is shown.

Table 1 Performance of the Cohda Wireless MK5 platform.

	Fixed point addition	**Fixed point multiplication**	**Floating point addition**	**Floating point multiplication**
Fastest	230.1 Mop/s	214.0 Mop/s	113.0 Mflop/s	88.2 Mflop/s
95th percentile*	214.0 Mop/s	200.0 Mop/s	113.0 Mflop/s	88.2 Mflop/s
Average	174.0 Mop/s	156.0 Mop/s	112.0 Mflop/s	60.0 Mflop/s
5th percentile*	200.0 Mop/s	150.0 Mop/s	103.0 Mflop/s	78.9 Mflop/s
1st percentile*	107.0 Mop/s	88.2 Mop/s	50.8 Mflop/s	47.6 Mflop/s

*The kth percentile is the number larger than k% of the measurements.

We use an internal timer for the measurement which runs at a frequency of 1MHz, while the processor runs at 800MHz. Therefore the accuracy of our measurement is within 800 clock cycles. The first test uses a large number of operations in a loop. We used 8×10^9, 4×10^9 and 2×10^9 operations for each of the measured types. The different number of operations allows us to confirm that the execution time is linear to the number of operations. These measurements give us a notion of the average performance of the system.

As our system does not use a RTOS, during the time that the test executes (over 4 minutes in some cases) there are other tasks preempting the test. In order to measure the variability of the performance we designed a second test. In this test we measure the time needed to perform 3000 operations, and the measurement is repeated for 1 million times. We search for the fastest iteration and the 1st, 5th and 95th percentile. If performance requirement is higher, the device can be overclocked to reach 1GHz, and some secondary OS tasks such as Bluetooth can be shut down to remove their influence.

4 Overall architecture

In the Fig. 2 we can see a diagram showing how the system works. We use the platform in [30], with the MPC design in [15]. As we will use the PF strategy for the communication, the leader sends its state to the first vehicle, and the vehicle i sends its state to the vehicle $i+1$. In a real environment, the leader vehicle would be driven by a human driver, and the commands control the real vehicle. In a simulation we create a profile for the acceleration commands and actuate the model of the vehicle.

The vehicle i receives the state from the vehicle $i-1$ and uses a^{i-1} and x_p^i as inputs for the MPC controller (the upper layer), which computes the desired acceleration. The desired acceleration is used by the lower layer as reference value and outputs the duty cycle of the input to the motor, which controls the vehicle. The state of the vehicle (sensed or simulated) is given as an input to the two controllers and it is also sent to the next vehicle.

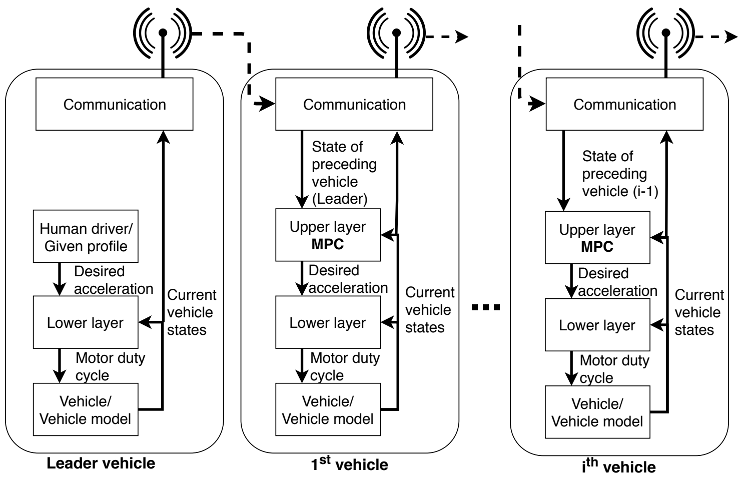

Figure 2 Overall architecture of the system.

4.1 Lower layer controller

The lower layer controller is a state-feedback controller and it runs at a faster rate than the upper layer with a sampling rate of 2ms [15]. The output of this controller is the motor duty cycle which controls the vehicle.

4.2 MPC: Upper layer controller

In MPC we solve an optimisation problem by defining the following quadratic cost function subject to specific constraints on inputs and states:

$$J = {x^i_{N+k|k}}^T P x^i_{N+k|k} + \sum_{j=0}^{N-1} ({x^i_{j+k|k}}^T Q x^i_{j+k|k} + {u^i_{j+k|k}}^T R u^i_{j+k|k})$$

$$\begin{aligned} \textbf{subject to} \quad x^i_{j+k+1|k} &= \Phi^i x^i_{j+k|k} + \Gamma^i u^i_{j+k|k} + \Psi^i a^{i-1}_{j+k|k},\ j = 0, ..., N-1 \\ x_{min} &\leq x^i_{j+k|k} \leq x_{max},\ j = 1, ..., N \\ u_{min} &\leq u^i_{j+k|k} \leq u_{max},\ j = 1, ..., N \end{aligned} \tag{3}$$

where J is the cost function, N is the length of the control horizon, $x^i_{j+k|k}$ is the predicted state vector of vehicle i after j steps computed at time k, where $x^i_{k|k}$ is the sensed state of vehicle i. $u^i_{j+k|k}$ is the computed input vector for the vehicle i for the j step, and Q, R and P are the weight matrices. It should be noted that a quadratic cost function is chosen so that the problem is convex and a global minimum can be found.

In order to use MPC we must discretize the platoon model in Eq. (2) using Zero-Order Hold (ZOH). After the discretization, the predictive model for vehicle i becomes:

$$x^i_{j+k+1|k} = \Phi^i x^i_{j+k|k} + \Gamma^i u^i_{j+k|k} + \Psi^i a^{i-1}_{j+k|k},\ \ j = 0, ..., N-1 \tag{4}$$

where $x^i_{j+k|k} = [a^i_{j+k|k} \quad \delta a^i_{j+k|k} \quad \Delta d^i_{j+k|k} \quad \Delta v^i_{j+k|k}]$ represent the predicted states. $u^i_{j+k|k}$ is the desired acceleration (the optimal control inputs that must be computed). $a^{i-1}_{j+k|k}$ is the predicted acceleration of the preceding vehicle. We consider that the future evolution of the acceleration of the preceding vehicle is constant. Therefore, it does not affect the optimisation process. The predictive model has 4 states and 1 input variable. Each of the states and the desired acceleration (u^i) have an upper and a lower bound. Therefore we have 10 constraints.

The upper layer uses MPC. It receives the current state of the vehicle and the state of the preceding vehicle, and gives the lower layer a new acceleration reference. The upper layer runs with a sampling time of 100ms since the maximum message rate is 10Hz. At every sample in which a new message has been received, the MPC controller computes an optimal series of future N inputs (N is the horizon length). When the message rate is lower than the sampling rate, the MPC controller automatically updates the desired acceleration using the next value of the optimal series of inputs that were computed in the last sample in which a message had been received. As the message rate can drop to 1Hz, we need the length of the control horizon to be at least 10 while a higher N could improve the quality of the control. We consider $10 \leq N \leq 20$.

4.3 Execution time budget for MPC

In this section, we compute the maximum available execution time for the MPC algorithm considering a message rate of 10Hz. That is, the time available to execute the MPC algorithm after performing all the other system tasks – see Fig. 3. The platform needs to send and receive messages (communication task), and compute the result of the lower layer and upper layer controllers. The upper layer also has some overhead besides the MPC algorithm such as updating the value of some pointers and variables like the desired acceleration. Fig. 5 shows the tasks performed by each piece of hardware. The MARS co-processor sends and receives packages. The main processor creates the packages that need to be sent, processes the received packages, and executes the upper and lower layer controllers.

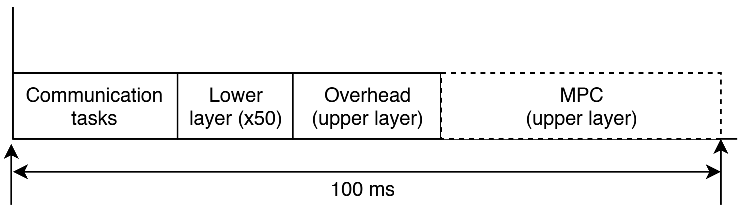

Figure 3 Execution time requirement when running on a single core.

In order to measure the contribution of each task to the total execution time, we use the platform developed in [29] which uses a PID controller for the upper layer and a state-feedback controller for the lower layer. We removed the logging functions and the PID controller, so that we get a minimal version of the platform, and added time stamps to analyse the latency of each task. Furthermore, we have modified the platform so that the tasks in the main processor are scheduled using POSIX threads with a Fixed-Priority Preemptive Scheduler (FPPS). For tasks with equal priority it follows a First In, First Out (FIFO) schedule. The

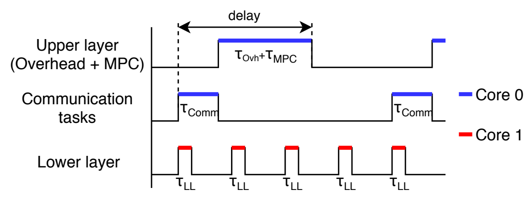

Figure 4 Typical execution without OS tasks.

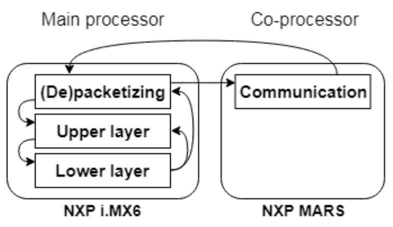

Figure 5 Task distribution and data flow on hardware.

OS has the highest priority, the lower layer and the (de)packetizing tasks are given an equal medium priority, while the upper layer has the lowest priority. The processor has two cores, making it possible to process two different tasks (threads) simultaneously. The kernel distributes the different tasks (including the OS tasks) between the 2 cores dynamically.

In Fig. 4 we can observe the expected execution of the tasks in the absence of OS tasks. The lower layer is represented at a lower frequency that the frequency implemented.

The results are shown in Table 2. We are mainly concerned about the tasks shown in Fig. 3. We present the measured maximum and average latencies after 100 runs and the execution frequency, i.e. how often does that task have to execute.

Table 2 Latency of other tasks that run on the platform.

	Maximum latency	Average latency	Execution frequency
Lower layer controller	0.045ms	0.0151ms	500Hz
Upper layer controller overhead	0.059ms	0.0126ms	10Hz
Communication tasks	1.063ms	0.3611ms	$\leq$ 10Hz

With the results in Table 2 we can obtain the execution time budget for the MPC task. In 100ms we need to perform the lower layer controller task 50 times $\left(\frac{500Hz}{10Hz}\right)$, and the communication and the upper layer controller tasks only once. We used the worst case latencies to compute the execution time available for the MPC algorithm in the worst case. The worst case latencies will occur when the OS tasks are running on both cores of the processor, therefore for the worst case analysis we assume that there is only one core available for all the tasks. e_{MPC}, e_{Comm}, e_{Ovh} and e_{LL} denote the maximum latency of the MPC task, the communications tasks, the overhead of the high level controller and the low level controller respectively.

$$e_{MPC} \leq 100ms - (e_{Comm} + e_{Ovh} + 50e_{LL}) = 96.628ms \tag{5}$$

From the above experiments, we conclude that the effects of other tasks are almost negligible and we obtained an upper bound for the execution time. How to respect this requirement is analysed in Section 6. The quality of the control is affected by the sensor-to-actuator delay. Therefore, the execution time should be as short as possible.

5 Automatic C code generation for MPC

In order to facilitate implementing MPC algorithms on embedded platforms, automatic C code generators are developed. These tools take a description of the desired MPC problem and generate the necessary C code to solve it with a given optimisation algorithm.

Code generators are used academically and in the industry [4]. There are many tools available. Some of them are commercial (paid) tools, e.g., ODYS [24] or FORCES [11] while others (e.g., µAO-MPC[32], CVXGEN [20], FiOrdOs [12], jMPC [2]) are free. We used CVXGEN, µAO-MPC and FiOrdOs in this paper since they are free and they allow to stop executions providing a sub-optimal solution.

- **CVXGEN** has been developed in the University of Stanford. It allows to describe an optimisation problem in general terms, the problem description includes the dimensions of the different matrices and vectors, and some properties such as being positively definite, or diagonal. It does not need the exact values of each entry of the matrices. The algorithm used is based on CVX, a solver for MATLAB. The tool is online based and free for academic use [20].

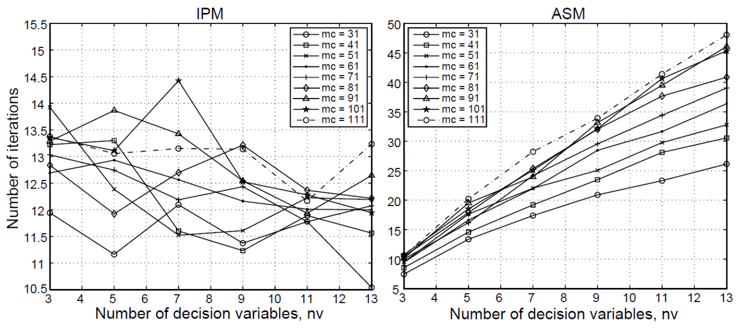

Figure 6 Number of iterations as a function of the number of variables (n_v) and constraints (m_c), reproduced from [19].

- **µAO-MPC** has been developed in the Otto von Guericke University of Magdeburg. This tool is a very similar to CVXGEN in its usage and flexibility. It uses a FGM algorithm for obtaining the solution. The tool can be downloaded for free and it works using Python, which is also a free tool [32].
- **FiOrdOs** has been developed in ETH Zurich. This tool requires a full description of the problem, with all the entries of the matrices before it can generate the code. It uses a FGM algorithm. The tool is a free toolbox for MATLAB [12].

6 MPC algorithms and computational requirements

In order to estimate the execution time of different algorithms, we need to know their complexity. The number of computations per iteration is deterministic in most algorithms, but the number of iterations depends on the convergence speed of the problem and the initial conditions making the total execution time unpredictable.[1]

- **IPM** reaches the solution in steps towards solving the Karush-Kuhn-Tucker equations, making few but computationally heavy iterations [19]. For IPM we used the estimate of the number of flops shown in [19] which is also shown in Table 3. We use the Gauss-Jordan elimination with pivoting method for solving the linear systems using the estimate given in [19]. We assumed that division operations are equivalent to 10 multiplications. For the number of iterations, we took an approximate value based on Fig. 6, reproduced from [19], with 13 iterations.
- **ASM** tries to guess the constraints that are active in the solution (Active Set) and does it by adding the constraints one by one on every iteration [19]. For ASM we use the estimate of the number of flops found in [19]. We made the same choices as for IPM. For the number of iterations, we can find a direct linear relationship between the number of decision variables, n_v, and the number of iterations when looking at Fig. 6, see Table 3.
- **GM** computes the gradient of the cost function in the current point and next, it moves one step in that direction. It repeats the process until it finds the minimum. In the fast variants, FGM, a sub-optimal solution is accepted as a trade-off for a faster computation time. An important advantage of FGM is that it can give an output at any point in time, making it possible to bound the execution time. These methods require the cost function to be quadratic [17] [3].

[1] In [23] an upper bound for the number of iterations of some algorithms is found. However, it requires knowledge of the exact values of the predictive model and the bound is significantly larger than the observed number of iterations [31].

For FGM we use the estimations provided in [18], which analyses several different FGM algorithms. It analyses Bemporad's and Richter's algorithms, each of them with 2 alternative formulations, which give different computational requirements. The equations used can be seen in Table 3. These estimates don't specify the type of operations. We assume that 50 iterations are needed, based on the experiments performed in [18], but those values are based on a different problem than the one used in [19], therefore they might not be comparable.

Table 3 Formulas used to compute the number of flops [8] [19] [18].

Algorithm	**Flops per iteration**	**Number of iterations**
IPM	$2n_v^2(n_c+1)+n_v(7n_c+2)+14n_c+1+$ $Ma(n_v)+Mm(n_v)+10(3n_c+1+Md(n_v))$	13
ASM	$2n_v^2+2n_v(2n_c+1)-n_c+Ma(n_v+0.5n_c)+$ $Mm(0.5n_c+n_v)+10(n_c+Md(n_v+0.5n_c))$	$2.5\times n_v$
Bemporad's FGM u-formulation	$N^2n_u(2n_u+3n_y+3n_c)$	50
Bemporad's FGM xu-formulation	$N(4n_x^2+6n_xn_u)+6N(N_c+n_y)(n_x+n_u)+$ $4Nn_u(n_x+n_u)$	50
Richter's FGM uy-formulation	$2N^2(n_y^2+n_un_y)$	50
Richter's FGM xuy-formulation	$2N(n_y+n_x)(5n_x+2n_u+n_y)$	50

In Table 3 the new variables have the following meaning:

- n_x: Number of states of the plant model, in this case 4.
- n_y: Number of outputs of the plant model, in this case 1.
- n_u: Number of inputs of the plant model, in this case 1.
- n_c: Number of inequality constraints, in this case 10.
- n_t: Parameter computed as $n_t = n_u + n_y + n_x$
- n_v: Parameter computed as $n_v = N \cdot n_u$
- $Ma(x)$: Number of additions needed to solve the linear system of equations, computed as $Ma(x) = 0.5(x-1)x(x+1)$ when using Gauss-Jordan elimination.
- $Mm(x)$: Number of multiplications needed to solve the linear system of equations, computed as $Mm(x) = 0.5x^2(x+1)$ when using Gauss-Jordan elimination.
- $Md(x)$: Number of divisions needed to solve the linear system of equations, computed as $Md(x) = x$ when using Gauss-Jordan elimination.

The effect of varying the control horizon length for a system with 4 states, 1 input, 1 output and 10 inequality constraints can be seen in Fig. 7. We can observe that the complexity of some algorithms grows exponentially while in others it grows linearly. Depending on the control horizon chosen for the application, different algorithms are recommendable.

7 Performance analysis

7.1 Estimated execution time

Using the specifications of the predictive model and the constraints in Section 4.2, taking $N = 15$, and the formulas given in Table 3, we can obtain the number of operations needed to solve MPC for vehicle platooning. Combining them with the performance of the device (Table 1) we can estimate the execution time for each algorithm.

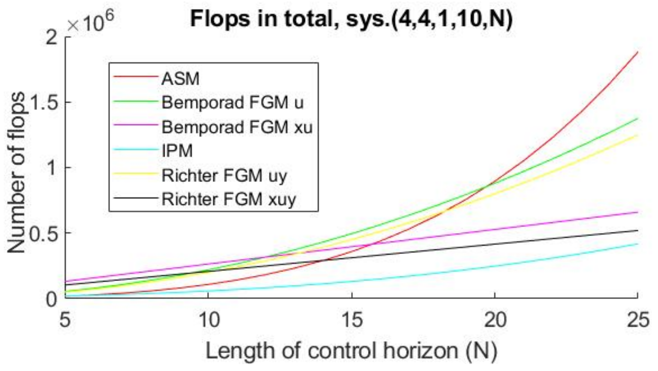

Figure 7 Effect of varying the length of the control horizon-Total number of flops

Table 4 Execution time estimation for different algorithms for the model described in Section 4.2 and $N = 15$.

	Per iteration			Total		
	Flops	Time [ms] (multiplications[1])	Time [ms] (mixed[2])	Flops	Time [ms] (multiplications[1])	Time [ms] (mixed[2])
ASM	9.56×10^3	0.1593	0.1112	35.85×10^4	5.9750	4.1686
Bemporad's FGM u-formulation	9.9×10^3	0.1650	0.1151	49.5×10^4	8.2500	5.7558
Bemporad's FGM xu-formulation	7.92×10^3	0.1320	0.0921	39.6×10^4	6.6000	4.6047
IPM	10.11×10^3	0.1685	0.1176	13.14×10^4	2.1907	1.5284
Richter's FGM uy-formulation	9×10^3	0.1500	0.1047	45×10^4	7.5000	5.2326
Richter's FGM xuy-formulation	6.24×10^3	0.1040	0.0726	31.2×10^4	5.2000	3.6279

[1] This measurement assumes that all the operations are multiplications.
[2] This measurement assumes that half of the operations are additions and the other half multiplications.

In Table 4 we show the number of operations and execution time in total and per iteration for all the considered algorithms. Two different execution times are given, one under the assumption that all the operations are multiplications and the other assuming mixed operations, i.e. half of the operations are additions and the other half are multiplications.

7.2 Code generation experiments

The theoretical estimations can be too optimistic, as they assume that the data is always available, which is equivalent to having a infinitely fast memory. Using the code generation tools described in Section 5 we run an experiment (Appendix A) on the Cohda platform, obtaining a real execution time. The three algorithms are considered. We use approaches based on several iterations and sub-optimality levels. Each algorithm converges to the solution at a different speed. Therefore they need a different number of iterations. We determined the number of iterations as the minimum necessary to reach a value within 0.001 units of the solution for a very large number of iterations, which is assumed to be the optimal solution. This is equivalent to an error smaller that 1% in the problem used (Appendix A).

For CVXGEN the number of iterations varied with the control horizon length, being 63 for N=10, 56 for N=15 and 59 for N=20. µAO-MPC and FiOrdOs use a dual approach, with an inner and outer loop. For µAO-MPC we needed 30 iterations for the inner loop and 30 for the outer loop. For FiOrdOs the number of iterations of the inner loop is 1, and 125 for the outer loop for all the horizons. The time displayed in Table 5 is the average between 100 solutions. We expect that the number of iterations needed to solve the vehicle platooning problem will be comparable to the number of iterations used in this problem.

Table 5 Execution time of different automatic code generation tools for different control horizon lengths.

Horizon length	CVXGEN	µAO-MPC	FiOrdOs
N=10	15.52 ms	106.150 ms	32.286 ms
N=15	26.02 ms	211.526 ms	73.632 ms
N=20	45.35 ms	358.030 ms	125.968 ms

7.3 Feasibility analysis

Following the execution diagram in Fig. 4, the delay from the sampling instant until the new control input is computed, depends on the execution times of the upper layer and the communication tasks. The execution time is variable due to the OS, making it impossible to obtain an exact value. We approximate it as:

$$delay \approx \tau_{MPC} + \tau_{Comm} + \tau_{Ovh} \tag{6}$$

where τ_{MPC}, τ_{Comm} and τ_{Ovh} are the average execution time of the MPC task, the communication task and the overhead of the upper layer, respectively.

Under the selected communication protocol, there is no use on having a faster sampling rate than 10Hz, but other communication protocols could be used. Therefore we will compute the maximum achievable sampling rate as the inverse of the delay. We consider that it is feasible to use an algorithm if its execution time is below the budget computed in Section 4.3.

Looking at the theoretical estimates, the execution time of IPM is the shortest, while the shortest FGM algorithm is Richter's algorithm using the xuy-formulation. For the FGM algorithms we don't know the type of operations. Therefore, we use the estimate in the case that all the operations are multiplications. For IPM the number of multiplications and additions are very similar [19] and we use the mixed estimation. When considering the experimental execution times, the best results are for CVXGEN for all the tested horizon lengths. In Table 6 we present the results for the selected algorithms allowing us to make the following observations. First, there is a significant difference in the execution time when using different algorithms or different control horizon lengths. Second, the delay is almost equal to the execution time of the MPC algorithm. Finally, all the selected algorithms are feasible to be used for this problem.

7.4 Trade-off analysis

From the complexity of the different algorithms (Section 6) we observe that the size of the predictive model and the length of the control horizon have a big impact on the complexity of the algorithm. Generally a longer control horizon length and a more accurate predictive model (which usually results in a larger model) give a better control performance, but the improvement might not be sufficient to overcome the negative effects of increasing the sensor-to-actuator delay and reducing the sampling rate.

When choosing hardware for MPC applications, the variability in the execution time must be taken into account. In all the implicit MPC algorithms the number of iterations varies depending on the initial conditions (making the total execution time vary), therefore it is not

Table 6 Feasibility analysis, execution time, delay and the maximum sampling period for the selected algorithms.

	Execution time (ms)	Delay (ms)	Maximum sampling period (Hz)	Feasible
IPM $N = 15$	1.5284	1.9021	526	Yes
Richter's FGM xuy-formulation $N = 15$	5.2000	5.5737	179	Yes
CVXGEN $N = 10$	15.5200	15.8937	63	Yes
CVXGEN $N = 15$	26.0200	26.6368	38	Yes
CVXGEN $N = 20$	45.3500	46.1027	22	Yes

enough to select a processor capable of meeting the timing constraints for the average case. The processor depends on the requirements of the application, i.e. the MPC algorithm must be guaranteed to execute within the timing constraints 90% of the times. To provide such guarantees, it is required to perform multiple experiments under different initial conditions and obtain a probabilistic distribution of the execution time.

Finally, the MPC task can be parallelized when running on a multi-core processor. This can improve the execution time of MPC but it must be done ensuring that the task in charge of receiving new messages is able to run. Every received message needs to be processed and it must be recalled that every vehicle broadcasts several messages per second, making it possible to receive several hundreds of messages per second when there is traffic.

8 Conclusion

In this paper we analysed the feasibility of employing embedded MPC for vehicle platooning and provided an overview of the trade-offs that can be done. We obtained a bound for the maximum execution time admissible when taking into consideration the other system tasks that run on the platform. We have shown that it is feasible in two different ways. First, we analysed the computational complexity of different MPC algorithms and compared it to the performance of the device, obtaining a theoretical execution time. Second, we used automatic C code generation tools to measure the real execution time of MPC algorithms for different control horizon lengths. We compared the execution times to the execution time bounds, showing that it is feasible to use embedded MPC for vehicle platooning. In this process, we have benchmarked the performance of various MPC algorithms with respect to parameters such as the horizon length and the number of the states.

References

1 Alessandro Alessio and Alberto Bemporad. A Survey on Explicit Model Predictive Control. In *Nonlinear Model Predictive Control, LNCIS 384*, pages 345–369. Springer-Verlag, Berlin Heidelberg, 2009.

2 Auckland University of Technology. jMPC Toolbox. URL: `http://www.i2c2.aut.ac.nz/Resources/Software/jMPCToolbox.html`.

3 Alberto Bemporad and Panagiotis Patrinos. Simple and Certifiable Quadratic Programming Algorithms for Embedded Linear Model Predictive Control. *IFAC Proceedings Volumes*, 45(17):14–20, 2012.

4 Daniele Bernardini. ODYS and GM bring online MPC to production! | ODYS. URL: `http://www.odys.it/odys-and-gm-bring-online-mpc-to-production/`.

5 L.G. Bleris, P.D. Vouzis, M.G. Arnold, and M.V. Kothare. A co-processor FPGA platform for the implementation of real-time model predictive control. In *American Control Conference (ACC)*, pages 1912–1917, 2006.

6 Catalin Braescu, Razvan C. Rafaila, Alexandru Tiganasu, Anca Maxim, and Constantin F. Caruntu. Distributed model predictive control algorithm for vehicle platooning. *International Conference on System Theory, Control and Computing (ICSTCC)*, pages 657–662, 2016.

7 Eduardo F. Camacho and Carlos Bordons Alba. *Model Predictive Control*. Springer-Verlag London, London, 2007.

8 Gionata Cimini and Alberto Bemporad. Exact Complexity Certification of Active-Set Methods for Quadratic Programming. *IEEE Transactions on Automatic Control*, 62(12):6094–6109, 2017.

9 J Currie, A Prince-Pike, and D I Wilson. Auto-code generation for fast embedded Model Predictive Controllers. *International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pages 116–122, 2012.

10 Alexander Domahidi, Hans Joachim Ferreau, Stefan Almer, Helfried Peyrl, and Juan Luis Jerez. Survey of industrial applications of embedded model predictive control. *European Control Conference (ECC)*, pages 601–601, 2016.

11 Embotech. FORCES Pro code generator. URL: `https://www.embotech.com/forces-pro`.

12 ETH Zurich. FiOrdOs - Code Generation for First-Order Methods. URL: `http://fiordos.ethz.ch/dokuwiki/doku.php`.

13 European Telecommunications Standards Institute. Intelligent Transport Systems (ITS); Harmonized Channel Specifications for Intelligent Transport Systems operating in the 5 GHz frequency band, 2012.

14 S. Gopi, V. M. Vaidyan, and M. V. Vaidyan. Implementation of FPGA based model predictive control for MIMO systems. In *IEEE Conference on Systems, Process Control (ICSPC)*, pages 21–24, 2013.

15 Amr Ibrahim, Chetan Belagal Math, Dip Goswami, Twan Basten, and Hong Li. Co-simulation Framework for Control, Communication and Traffic for Vehicle Platoons. *Euromicro Conference on Digital System Design (DSD)*, pages 352–356, August 2018.

16 Stephen J. Wright. Applying new optimization algorithms to model predictive control. *International Conference on Chemical Process Control – CPC V*, pages 147–155, 1996.

17 Juan L Jerez, Paul J Goulart, Stefan Richter, George A Constantinides, Eric C Kerrigan, and Manfred Morari. Embedded Predictive Control on an FPGA using the Fast Gradient Method. *European Control Conference (ECC)*, pages 3614–3620, 2013.

18 Dimitris Kouzoupis. Complexity of First-Order Methods for Fast Embedded Model Predictive Control. Master's thesis, ETH Zurich, 2014.

19 Mark S. K. Lau, S. P. Yue, K. V. Ling, and J. M. Maciejkowski. A comparison of interior point and active set methods for FPGA implementation of Model Predictive Control. *Proceedings of the European Control Conference*, pages 3–8, 2009.

20 Jacob Mattingley and Stephen Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012. `doi:10.1007/s11081-011-9176-9`.

21 V. Milanés, S. E. Shladover, J. Spring, C. Nowakowski, H. Kawazoe, and M. Nakamura. Cooperative Adaptive Cruise Control in Real Traffic Situations. *IEEE Transactions on Intelligent Transportation Systems*, 15(1):296–305, 2014.

22 Khalil Mohamed, Ahmed El Mahdy, and Mohamed Refai. Model Predictive Control Using FPGA. *International Journal of Control Theory and Computer Modeling*, 5(2), 2015.

23 Yurii Nesterov. *Introductory Lectures on Convex Optimization*, volume 87 of *Applied Optimization*. Springer US, 2004.

24 ODYS. ODYS QP solver, 2018. URL: `http://www.odys.it/qp/`.

25 Yasser Shoukry, M. Watheq El-Kharashi, and Sherif Hammad. MPC-On-chip: An embedded GPC coprocessor for automotive active suspension systems. *IEEE Embedded Systems Letters*, 2(2):31–34, 2010.

26 M Tsujii, H Takeuchi, K Oda, and M Ohba. Application of self-tuning to automotive cruise control. In *American Control Conference, 1990*, pages 1843–1848. IEEE, 1990.

27 A Galip Ulsoy, Huei Peng, and Melih Çakmakci. *Automotive control systems*. Cambridge University Press, 2012.

28 Adrian G Wills, Geoff Knagge, and Brett Ninness. Fast Linear Model Predictive Control Via Custom Integrated Circuit Architecture. *IEEE Transactions on control systems technology*, 20(1):59–71, 2012.

29 Sijie Zhu. NXP Platoon Control Algorithm Evaluation Platform. URL: `https://github.com/sijiezhu/NXP-Platoon-Control-Algorithm-Evaluation-Platform`.

30 Sijie Zhu. Model-in-the-loop Experiments and Analysis of Platoon Control Algorithms. Master's thesis, Technische Universiteit Eindhoven, 2018. URL: `https://research.tue.nl/en/studentTheses/model-in-the-loop-experiments-and-analysis-of-platoon-control-alg`.

31 P. Zometa, M. Kogel, T. Faulwasser, and R. Findeisen. On Time versus Space and Related Problems. In *American Control Conference (ACC)*, pages 57–64, 2012.

32 P Zometa, M Kögel, and R Findeisen. μAO-MPC Documentation. Technical report, Otto von Guericke Univertität Magdeburg, 2016. URL: `http://ifatwww.et.uni-magdeburg.de/syst/research/muAO-MPC/doc/muaompc-1.0.pdf`.

A Parameters used in experimental analysis

$$Q = \begin{bmatrix} 1.89 & 0 & 0 & 0 \\ 0 & 1.90 & 0 & 0 \\ 0 & 0 & 1.13 & 0 \\ 0 & 0 & 0 & 1.21 \end{bmatrix} \Gamma = \begin{bmatrix} 1.75 \\ 1.90 \\ 0.69 \\ 1.61 \end{bmatrix} x_0 = \begin{bmatrix} 0.20 \\ 0.83 \\ -0.84 \\ 0.04 \end{bmatrix} x_{max} = \begin{bmatrix} 0.61 \\ 0.23 \\ -0.55 \\ -1.10 \end{bmatrix}$$

$$P = \begin{bmatrix} 1.44 & 0 & 0 & 0 \\ 0 & 1.03 & 0 & 0 \\ 0 & 0 & 1.46 & 0 \\ 0 & 0 & 0 & 1.65 \end{bmatrix} \Phi = \begin{bmatrix} -0.88 & 0.71 & 0.36 & -1.90 \\ 0.24 & -0.96 & -0.34 & -0.87 \\ 0.77 & -0.24 & -1.37 & 0.18 \\ 1.12 & -0.77 & -1.11 & -0.45 \end{bmatrix} x_{min} = \begin{bmatrix} -1.63 \\ -100 \\ -100 \\ -100 \end{bmatrix}$$

$R = 1.05 \qquad u_{max} = 1.38 \qquad u_{min} = -0.49 \qquad \Psi = [0 \quad 0 \quad 0 \quad 0]$