University Joseph Fourier

Master 2 Internship at Gipsa-Lab

Evaluation of an optimization framework for fast bilinear systems

Author:
Peter Pflaum

Supervisor:
Mazen Alamir

June 17, 2013

Contents

1 Introduction

Since the 1980s model predictive control (MPC) has been applied as a successful control strategy that provides high performances in combination with a comfortable way of controller tuning. As MPC requires high computations it was restricted to systems with rather slow dynamics such as chemical processes (see for instance [10]) for a long time. With the continuously increasing computation capacities of recent processors a new field of research has emerged during the last decade which has the objective of applying MPC also to systems showing fast dynamics.

One key characteristic of this new stream of research is that the implicit assumption according to which the computation of the optimal future control sequence can be obtained in a fraction of the system's sampling time is no more valid. Many works providing strategies of distributing the optimization process over the system lifetime have been proposed in the recent years such as in [2] where a receding-horizon state feedback law is proposed that yields a sub-optimal solution of the minimum interception time problem. In [9], a real-time iteration framework addressing general nonlinear models uses a multiple shooting method and only a single iteration of a Newton like descent algorithm is applied in which the cost function is based on the KKT optimality conditions. In [12] a continuous formulation of the above idea is proposed in which the decision variable is given by $\mathbf{u}$ while the gradient method is used to decrease the cost function related to the KKT conditions through appropriate defnition of the derivative w.r.t. $\mathbf{u}$. In [5, 11] a distributed-in-time Sequential Quadratic Programming coupled with a trust region approach has been used together with a rather system dependent parametrized approach in order to deal with real-time requirement.

The work described in this contribution is dedicated to study and evaluate the performances of a new optimization framework for bilinear systems with fast dynamics which was proposed in [1] and which aligns with the works mentioned in the previous paragraph. Since this framework uses a pretty unusual approach to mimimize the cost function it is difficult to have an idea of its performances without having executed extensive simulations. These simulations are the first key part in this work. Furthermore the benefits of possible extensions to the basic framework such as parameterization strategies or a variable control updating period are examined. Finally a comparison to the established *ACADO optimization toolkit* [6, 7], a set of optimization algorithms with an MPC simulation environment, is done.

The fact that the proposed optimization framework applies to the class of bilinear systems makes it interesting for many applications. For instance in hydraulic systems, pipe systems in general, power electronic systems or in robotics, models of bilinear structure are very widely spread. Additionally, as the detailed lecture of this work will show, many nonlinear systems can be transformed into the desired bilinear form by coordinate transformations or simple extensions opening a considerable field of applications to this optimization framework. An example of such an approach to modeling is given in [13] where an optimal tracking control based on a bilinear system model is proposed for the famous and rather complex Tokamak system.

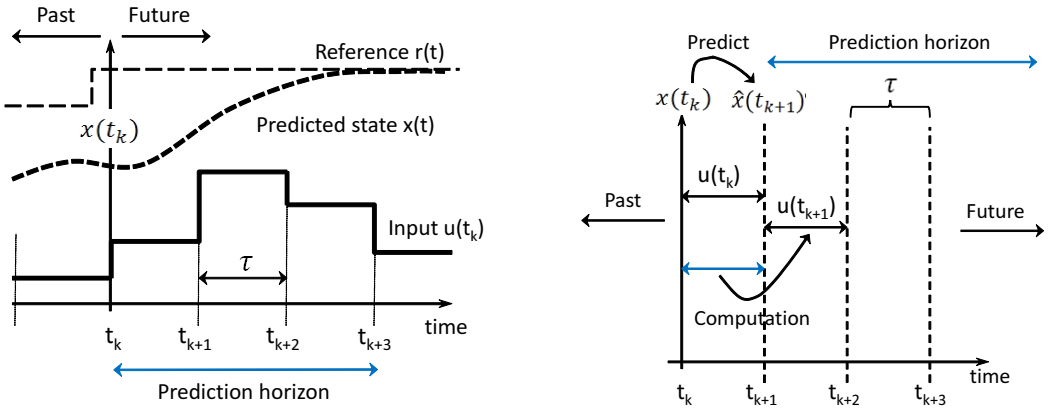
2 From NMPC to fast NMPC

In this section a brief recall of the principle of nonlinear model predictive control (NMPC) is provided, pointing out the difficulties arising when one aims to apply NMPC to fast systems.

Given a state space model of a nonlinear system:

$$\dot{x} = f(x, u) \quad ; \quad (x, u) \in \mathbb{R}^n \times \mathbb{U}(x) \tag{1}$$

where x, u are the state and the control vectors respectively. A MPC feedback is defined by the following cost function:



Ideal MPC

MPC for fast systems

Figure 1: Difference between the ideal MPC scheme where the optimal solution is computed in a fraction of the sampling time (left) and the fast MPC scheme where the active sampling period is not part of the prediction horizon (right).

$$J(\mathbf{u}, x_0) := \int_0^T [l_1(x(t)) + l_2(u(t))]dt \tag{2}$$

where $\mathbf{u} := \left(u^{(1)}...u^{(N)}\right) \in \mathbb{R}^{m \times N}$ is a piece-wise constant (pwc) control profile over the prediction horizon of length $T = N\tau$ (where τ is the sampling period).

The ideal MPC assumes that at each sampling instant $t_k = k\tau$, the optimal solution $\mathbf{u}_{opt}(t_k) = \left(u_{opt}^{(1)}...u_{opt}^{(N)}\right)$, resulting from the constrained optimization problem given in equation (3), can be obtained in a fraction of the sampling period so that during the same sampling period $[t_k, t_{k+1}]$, the first control vector $u_{opt}^{(1)}(t_k)$ can be applied.

$$\mathbf{u}_{opt}(t_k) = \underset{u}{\operatorname{argmin}}\, J(\mathbf{u}, x(t_k)) \tag{3}$$

Obviously this assumption is not true anymore when MPC is applied to fast systems. The computation time of the optimal solution is not negligible anymore compared to the sampling period or is even not sufficient at all to find the optimal solution $\mathbf{u}_{opt}(t_k)$.

This situation can be handled by distributing the numerical minimization of the cost function over time while providing at each sampling instant $k\tau$ some $\mathbf{u}(t_k)$ which is not the optimal solution $\mathbf{u}_{opt}(t_k)$, but the result of a limited number of iterations of some optimization process performed during the previous sampling period.

Doing so it is hoped that after several sampling periods the optimal ideal scheme can be asymptotically recovered which can be expressed as:

$$\lim_{k \to \infty} \|J(\mathbf{u}(t_k), x(t_k)) - J(\mathbf{u}_{opt}(t_k), x(t_k))\| = 0 \tag{4}$$

Note that (4) is likely to hold if the decrease of the cost function during one sampling period due to the optimization method is faster than the potential increase of the cost function due to the horizon shift and uncertainties.

Remark:

When applying NMPC to fast systems, the prediction horizon over which the control sequence is attempted to be optimized does not start at the current sampling instant $k\tau$ anymore, but at $(k+1)\tau$. This is due to the fact that the control input $\mathbf{u}^{(1)}(t_k)$ which is applied during $[t_k, t_{k+1}]$ results from the cost improvements that were done during the previous sampling period $[t_{k-1}, t_k]$ and consequently is not a degree of freedom of the new optimization problem.

This means in practice, that at instant $t_k = k\tau$ the state $\hat{x}(t_{k+1})$ at instant $(k+1)\tau$ is predicted based on the system model, the measured state $x(t_k)$ and the piece-wise constant control input $\mathbf{u}^{(1)}(t_k)$ applied during $[t_k, t_{k+1}]$. The actual prediction horizon which is subject to the optimization process during $[t_k, t_{k+1}]$ is then $[t_{k+1}, t_{k+1} + N\tau]$.

In Figure 1 this difference in the prediction horizon position between ideal and fast NMPC is illustrated.

3 Description of the provided framework

The focus of this internship is on implementing and evaluating a complete optimization framework for fast bilinear systems which is proposed in [1]. In this chapter the principle of this framework is introduced.

3.1 Problem formulation and notation

The class of systems to which this optimization algorithm addresses is the class of bilinear systems of the form:

$$\dot{x} = \left(\bar{A}_0 + \sum_{i=1}^{m} v_i \bar{A}_i \right) x + Bv \quad ; \quad x \in \mathbb{R}^n, \quad v \in \mathbb{V}(x) \tag{5}$$

.

The admissible set $\mathbb{V}(x)$ is assumed to be a hypercube of the form:

$$\mathbb{V}(x) = \prod_{i=1}^{m} [\underline{v}_i(x), \overline{v}_i(x)] \tag{6}$$

In [1] it is shown that every bilinear system as given in equation (5) can be transformed into the following so called canonical bilinear form to which the algorithm applies:

$$\dot{x} = \left(\sum_{i=1}^{m} u_i A_i \right) x \tag{7}$$

Note that the dimensions of the system in canonical bilinear form are not the same as for the original bilinear system in equation (5). The new dimensions are $A_i \in \mathbb{R}^{(n+m)\times(n+m)}$ and $\mathbb{U} = \mathbb{V} \times \{1\}$.

The Model Predictive Control feedback is defined by the following cost function which the algorithm attempts to minimize:

$$J_\rho(\mathbf{u}, x) := \int_{t=0}^{T} \left[l_1(x) + l_2(u) + \rho \sum_{j=1}^{n_c} \left[max(0, g_j(x)) \right]^2 \right] \tag{8}$$

The constraints that have to be satisfied by the optimal solution $u_{opt}(t)$ are expressed in the following set of n_c inequality constraints:

$$g\left(x(t)\right) \leq 0 \in \mathbb{R}^{n_c} \tag{9}$$

One special characteristic of this optimization framework is that the state constraints are handled in the cost function by a variable weight ρ as we will see later.

As in most MPC formulations, the control profile $u(t)$ is assumed to be piecewise constant (p.w.c.) with sampling period τ so that the control input $u(t)$ over the prediction horizon length $T = N\tau$ can be expressed by the sequence

$$\mathbf{u} := \left(u^{(1)}...u^{(N)}\right) \tag{10}$$

so that:

$$u(t) = u^{(i)} \quad ; \quad i = int(t/\tau) + 1 \tag{11}$$

.

Under this piecewise constant control input the cost function can be rewritten as a discrete cost function:

$$J_\rho(\mathbf{u}, x(k)) := \sum_{i=1}^{N} \left[l_1(x^{\mathbf{u}}(i|k)) + l_2(u^{(i)}) + \rho \sum_{j=1}^{n_c} [max(0, g_j(x^{\mathbf{u}}(i|k))]^2 \right] \tag{12}$$

where $x^{\mathbf{u}}(i|k)$ represents the predicted state at the sampled time instant $i\tau$ under the pwc control sequence $\mathbf{u}$ and with the initial state of the prediction horizon $x(0|k) = x(k)$. Note that the parameter k represents the global instant of time $t = k\tau$ where the actual prediction horizon begins.

3.2 Description of the control updating method

3.2.1 Trajectory computation using second order Runge-Kutta integration

The most computation time consuming task of direct methods in solving MPC is the computation of the state trajectory based on a given control input profile. In this framework a second-order Runge-Kutta integration is used for this issue. The relative low precision compared to other methods such as higher order Runge-Kutta methods is compensated by the important benefit of fast computations. Additionally the sampling period and the prediction horizon are rather short in fast NMPC which makes it probable that the error due to the numerical integration does not exceed an acceptable range. Nevertheless the second order Runge-Kutta method could be exchanged with another method if it was required.

Given a sequence of control inputs $\mathbf{u}$, the resulting state trajectory is computed using the second order Runge-Kutta integration as follows:

$$x^{\mathbf{u}}(i|k) = \left[\Phi(u^{(i)})\right] \times ... \times \left[\Phi(u^{(1)})\right] x(0|k) \tag{13}$$

where $\Phi(u^{(i)})$ is the one step transition matrix representing the second order Runge-Kutta integration corresponding to input $u^{(i)}$. The transition matrix $\Phi(u^{(i)})$ is computed as follows:

$$\Phi(u^{(i)}) = \left[\mathbb{I} + \tau S_A(u^{(i)}) + \frac{\tau^2}{2} S_A(u^{(i)}) S_A(u^{(i)})\right] \tag{14}$$

with $\quad S_A(u^{(i)}) = \sum_{j=1}^{m} u_j^{(i)} A_j$.

Note that the transition matrices $\Phi(u^{(i)})$ only depend on the control input $u^{(i)}$. This is a characteristic of bilinear systems that is exploited by the optimization algorithm in order to reduce computation times as we will see later.

3.2.2 Principle of the optimization method

The main idea behind this optimization algorithm is to minimize the cost function by successively considering each single control component $u_l^{(i)}$ of the control input trajectory (i.e. one of the $m \times N$ piecewise constant control inputs lying in the prediction horizon $[0, N\tau]$) during one iteration and trying to improve the value for this control input component $u_l^{(i)}$ such that the cost function decreases.

Preliminary remarks:

This iterative optimization scheme is applying a distribution of the optimization process over several sampling periods. This means that after a horizon shift, the new optimization problem depends on improvements of the cost function that were achieved during the previous sampling periods. The therefore necessary memory of the algorithm is the input trajectory $\mathbf{u}$, the corresponding trust region radiuses $\alpha_l^{(i)}$ to each component $\mathbf{u}_l^{(i)}$ and the corresponding transition matrices $\mathbf{\Phi}^i = \Phi(u^{(i)})$.

When the horizon is shifted, a shift of the index in the stored parameters is performed such that for instance $u^{(i)}$ takes the value of the previous $u^{(i+1)}$ and $u^{(i+1)}$ takes the value of the previous $u^{(i+2)}$. This has the advantage that the components of $\mathbf{\Phi}^{(i)}$ never have to be computed all at a time. Only the component $\mathbf{\Phi}^i = \Phi(u^{(i)})$ corresponding to the control parameter $u^{(i)}$ which the algorithm attempts to improve has to be recalculated.

The order in which the $m \times N$ components of the input trajectory are visited by the algorithm is as follows: For a given instant $i\tau$ all m inputs are attempted to be optimized one after another before the m inputs at instant $(i+1)\tau$ are subject to the optimization and so on.

For components $u_l^{(i)}$ where $i > 1$, the resulting state trajectories due to variations of $u_l^{(i)}$ do not have to be calculated over the whole prediction horizon but only for the interval $[(i-1)\tau, N\tau]$, as the past state trajectory $x^{\mathbf{u}}(j|k)$ for $j = 1, ..., i-1$ is not affected by the modification.

Consequently, the cost which is computed is not the cost over the whole horizon, but only the partial cost over the partial horizon $[(i-1)\tau, N\tau]$. This means a great reduction in computation times for components $u^{(i)}$ lying further in the future of the prediction horizon.

Control updating algorithm

In the sequel the steps which are performed during one iteration (i.e. during the optimization of one component $u_l^{(i)}$) are introduced:

1. Take one component $u_l^{(i)}$ of the actually stored input trajectory. After that, build the interval $[u_{min}, u_{max}] := [u_l^{(i)} - \alpha_l^{(i)}, u_l^{(i)} + \alpha_l^{(i)}]$ where $\alpha_l^{(i)}$ is the trust region radius for component $u_l^{(i)}$. Then take $r+1$ equally distributed values in this interval and compute the resulting state trajectories and the corresponding cost function for each of them. Based on the obtained cost values, compute a r-th order polynomial approximation from these cost values. Using a finer grid of n_g values on the same interval $[u_{min}, u_{max}]$, determine the value $\bar{u}_l^{(i)}$ that minimizes this scalar polynomial.

2. If the value $\bar{u}_l^{(i)}$ decreases the cost function significantly, the corresponding value $u_l^{(i)}$ of the stored control input trajectory is updated with $\bar{u}_l^{(i)}$, as well as the corresponding transition matrix $\Phi(u^{(i)})$. Additionally the trust region size $\alpha_l^{(i)}$ for this parameter is increased in case the cost function value was improved. Otherwise, if the cost wasn't improved, $\alpha_l^{(i)}$ is decreased.

3. If the algorithm found a local minimum (this is determined by checking whether the trust region size for all input components $u_l^{(i)}$ is inferior than a predefined threshold ϵ_α) the constraints weight ρ is adapted. In case that no constraints are violated, ρ is decreased, otherwise it is increased.

4. Having executed the previous three steps for all m input components of $u^{(i)}$, index i is increased and the

same computations of the previous three steps are done for each of the m input components of $u^{(i+1)}$. After having visited the input components of $u^{(N)}$ (i.e. $i = N$), index i is reset to 1 instead of being increased.

5. Every time one iteration has been terminated (i.e. each time the steps 1-3 have been performed) the algorithm checks whether the actual control updating period is elapsed. If yes, the first control $u^{(1)}$ of the prediction horizon is applied and the horizon is shifted. This also means that the smallest theoretically achievable sampling period is limited by the time the algorithm needs to perform a single iteration. Note that in practice the algorithm needs to perform several iterations to obtain good performance.

4 Implementation to a simple car model

4.1 Introduction of the simple car model

The chosen model for implementing the algorithm is the model of a simple car shown in Figure (2). The corresponding state-space representation of the model is given in equation (15). This model can be easily transformed into a bilinear system by a change of coordinates which means that this model provides the required structure for the algorithm. Another reason for this choice is the small amount of states (4 states). Additionally this example is a very commonly known and demonstrative problem which makes it easy to show the basic functionalities and benefits of the proposed algorithm.

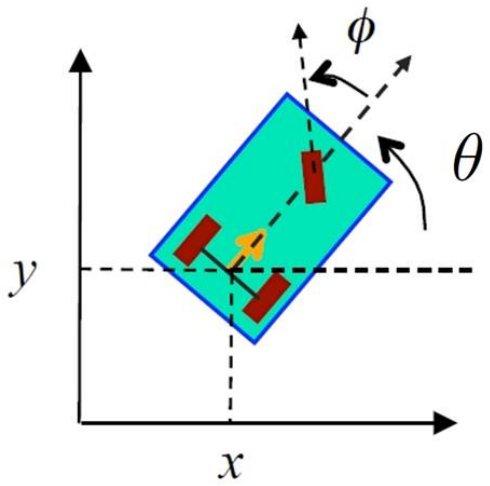


Figure 2: Sketch of the simple car model.

Simple car dynamics:

$$\begin{aligned} \dot{x} &= u_1 \cos\theta \\ \dot{y} &= u_1 \sin\theta \\ \dot{\Phi} &= u_2 \\ \dot{\theta} &= \frac{1}{L} \tan\Phi u_1 \end{aligned} \tag{15}$$

This model can be easily converted into the so called chained form which is a subclass of bilinear systems:

$$\begin{aligned} \dot{z_1} &= v_1 \\ \dot{z_2} &= v_2 \\ \dot{z_3} &= z_2 v_1 \\ \dot{z_4} &= z_3 v_1 \end{aligned} \tag{16}$$

by applying the following change of coordinates:

$$\begin{aligned} z_1 &= x \\ z_2 &= \frac{1}{L}\frac{1}{\cos^3\theta}\tan\Phi \\ z_3 &= \tan\theta \\ z_4 &= y \end{aligned} \tag{17}$$

and

$$\begin{aligned} v_1 &= \cos\theta u_1 \\ v_2 &= \frac{Lu_2 + 3\sin^2\Phi\sin\theta\cos\theta u_1}{\cos^2\theta\cos^3\Phi} \end{aligned}$$

The so obtained chained form can finally be transformed into the canonical bilinear form to which the algorithm applies by extending the system by two additional states of constant values $z_5 = z_6 = 1$.

$$\dot{z} = (v_1 A_1 + v_2 A_2) z \tag{18}$$

where:

$$A_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Remark:

Due to the transformation from the original system to the chained form, the possible range of the system's state is artificially limited because of a singularity in term z_2 of equation (17). It must be guaranteed that $\cos^3\theta$ does not become zero. This can be achieved by limiting the car's direction to angles in the range of $\left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$.

4.2 Main functionality demonstration

In this section the implementation of the algorithm for the previously given simple car model is described, and the resulting performance is evaluated from a qualitative point of view. Note that the implementation of the simple car model in the whole section is uniquely done in MATLAB code which means that the computation speed is much lower than if it was coded in a language like C, C++ or Jave. In section 6 the twin pendulum system will be implemented in C++ code which allows higher computation speeds and consequently gives a more realistic idea of the algorithm's performance limits.

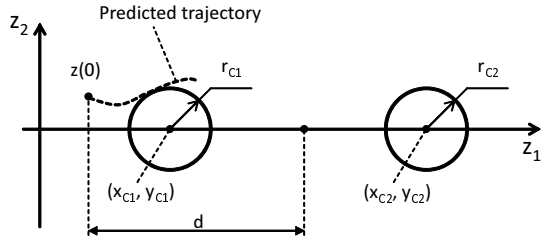


Figure 3: Scenario based on which the cost function given in equation (19) is defined. The figure shows the 2-dimensional plane in which the car is moving. $z(0)$ is the actual car position. (x_{C1}, y_{C1}) and (x_{C2}, y_{C2}) are the centres of the two obstacle circles and r_{C1} and r_{C2} the corresponding radiuses.

Parameter	Value	Description
β^+_{trust}	4	factor to increase the trust region $\alpha_l^{(i)}$
β^-_{trust}	0.25	factor to decrease the trust region $\alpha_l^{(i)}$
β^+_{constr}	2	factor to increase the constraint weight ρ
β^-_{constr}	0.5	factor to decrease the constraint weight ρ
α_{max}	10	upper bound on trust region size $\alpha_l^{(i)}$
α_{min}	0.01	lower bound on trust region size $\alpha_l^{(i)}$
ρ_{max}	100000	upper bound on constraint weight ρ
ρ_{min}	100	lower bound on constraint weight ρ
ϵ_α	0.1	threshold for stationary point
n_g	20	finer grid to search the minimum on polynomial
r	2	order of the polynomial approximation
γ	0.00001	threshold indicating a significant cost decrease

Table 1: Parameters for tuning the algorithm's dynamic behaviour

In a first step the main functionality of the algorithm is demonstrated with a special focus on how constraints are handled. Therefore a scenario is implemented where the car's objective is to follow the x-axis while avoiding to run into two obstacles which are positioned on the x-axis. In Figure (3) this scenario is illustrated graphically.

In terms of the cost function given in equation (12) this scenario is expressed as follows:

$$\begin{aligned} l_1(z) &= Q\left[(z_1 - (z_1(0) + d))^2 + z_4^2\right] \\ g_j(z) &= r_{Cj}^2 - \left[(z_1 - x_{Cj})^2 + (x_4 - y_{Cj})^2\right] \quad ; \quad j = \{1, 2\} \end{aligned} \tag{19}$$

where term $l_1(z)$ of equation (19) represents a weight on the distance of the predicted car position from a point lying on the x-axis in a distance d from the actual car position. This term expresses the car's interest in moving along the x-axis. Term $g_j(z)$ of equation (19) expresses the weight in case that the obstacle j in form of a circle with radius r_{Cj}, x-coordinate x_{Cj} and y-coordinate y_{Cj}, is violated.

Apart from the general MPC parameters such as the sampling period and the prediction horizon length, the algorithm contains some specific parameters which can be used to tune the algorithm's dynamic behaviour. These specific parameters concerning the dynamic trust region size adaption and the dynamic constraints weight adaption have been chosen once for all in this report and are given in Table (1).

Figure 4 presents the resulting car trajectory for a prediction horizon of $T = 2sec$ at a sampling period of $\tau = 0.1sec$. The resulting number of degrees of freedom is $n_{dof} = m(T/\tau) = 40$. The algorithm is clearly able to make the car follow the x-axis while avoiding the obstacles. The graphic on the right shows how the

constraints weight ρ is dynamically adapted by the algorithm. At the beginning the algorithm is at a stationary point (i.e. the optimal trajectory is determined and $\|\alpha\|_\infty < \epsilon_\alpha$). From the moment where the obstacle appears in the prediction horiozon, ρ increases until the algorithm indicates that it isn't at a stationary point anymore. After having passed at the first obstacle, the algorithm arrives at a stationary point again and ρ is decreased as no obstacle lies in the prediction horizon. Short time after that the same happens again when the second obstacle appears in the horizon.

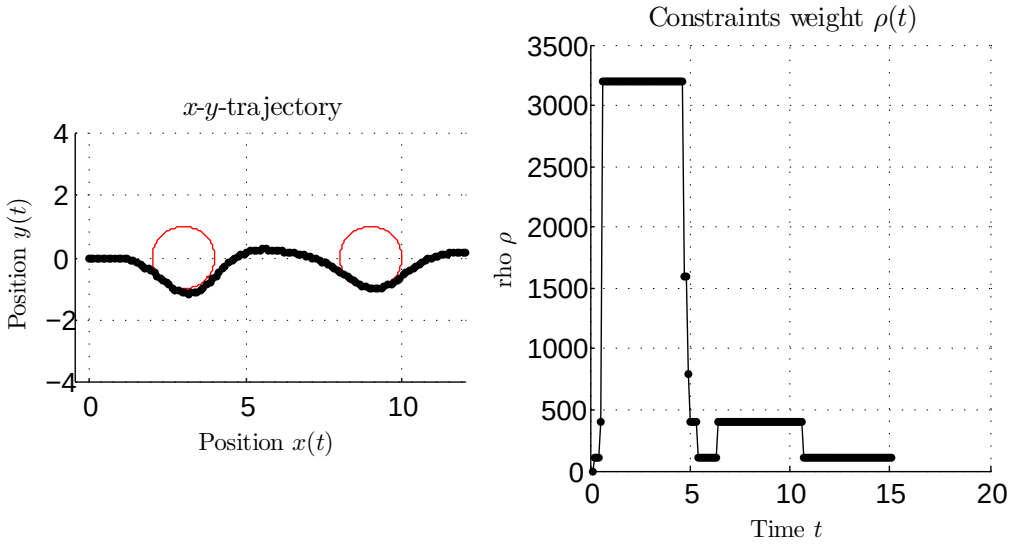


Figure 4: Simulation of the scenario where the car's objective is to follow the x-axis while avoiding to run into the obstacles (red circles). On the right the corresponding dynamic adaption of the constraints weight ρ is shown. When the circles appear in the horizon, ρ clearly increases and decreases once the obstacle has been circumnavigated.

4.3 Impact of reduced number of iterations

The objective of this section is to demonstrate the impact of the computation speed, i.e. the number of iterations performed during one control updating period (:= sampling period τ), on the stability of the algorithm.

For this issue a tracking scenario is chosen where the car tries to follow a given reference trajectory of trapezoidal shape. In terms of the cost function given in equation (12) this scenario is expressed as follows:

$$\begin{aligned} l_1(x(i|k)) &= Q\left[x_4(i|k) - y_{ref}(x_1(i|k))\right]^2 \\ l_2(u(i|k)) &= -R\left[u_1(i|k)\right]^2 \end{aligned} \tag{20}$$

Term $l_1(x(i|k))$ of equation (20) represents a weight on the car's distance from the reference trajectory and term $l_2(u(i|k))$ which has a negative sign represents the interest of the car to increase its speed. This second term is necessary because otherwise the car would find a global minimum as soon as it is on the reference trajectory and thus would not move on anymore.

Figure 5 shows the resulting trajectories of the car for different computation speeds. The sampling period in this scenario is $\tau = 0.025$ and the prediction horizon is $T = 0.75sec$ which means a number of degrees of freedom of $n_{dof} = 60$. The computation speed is expressed in terms of the number of iterations (one iteration := improvement of one component $u_l^{(i)}$) that can be performed during one sampling period τ.

Note that the way the amount of iterations per sampling period is restricted in the following simulations is not done by directly fixing the number of iterations, but by limiting the time the algorithm runs per sampling period. In reality the number of performed iterations is strongly varying from sampling period to sampling period since the computation of an iteration takes much longer if an input component $u^{(i)}$ for small values of i is optimized than for high values of i (see section 3.2.2). This means that when talking about iterations in the sequel, the values have to be understood as the amount of iterations performed during one sampling period in average over the whole simulation time.

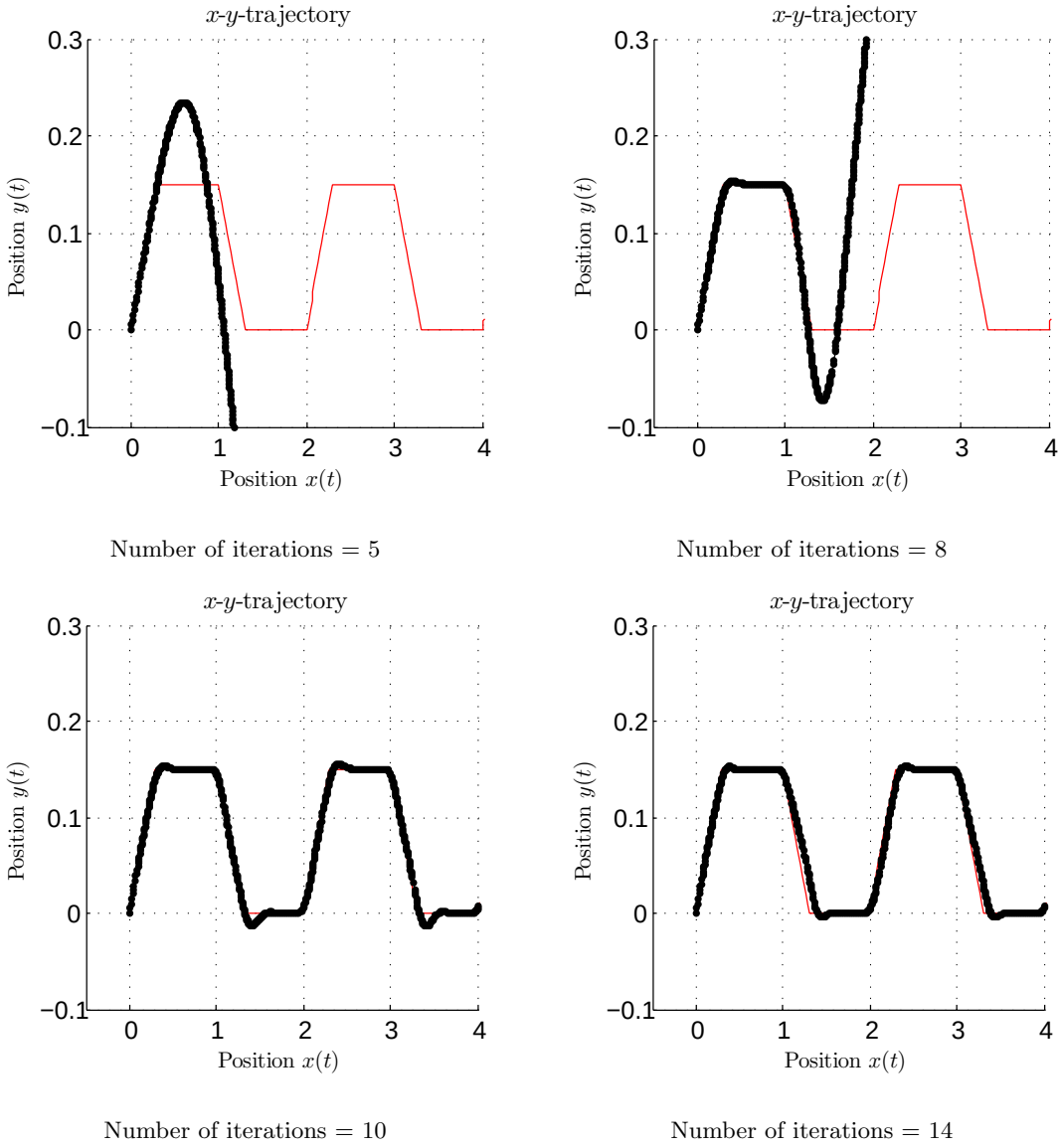
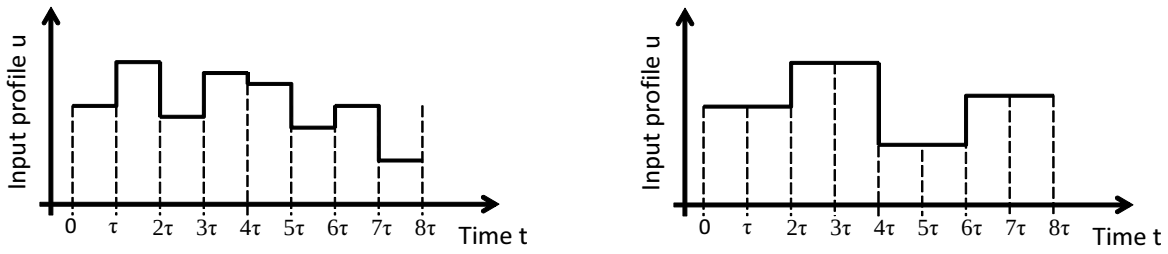


Figure 5: Tracking behaviour for different numbers of iterations performed during one sampling period. When 8 or less iterations are performed during one sampling period the behaviour is unstable (see the two top graphs) and it becomes stable if the number of iterations per sampling interval is higher than 8.

It can be clearly seen that in this specific tracking scenario, the system shows good performances if more than 8 iterations are performed during one sampling period. The two graphs at the bottom confirm that in order to obtain stable behaviour, only a fraction of the 60 input components $u_l^{(i)}$ (here: > 8 iterations) must be visited



No parametrization Parametrization $p = 2$

Figure 6: Difference between unparametrized and parametrized input profile.

during one sampling period. This is a clear proof that the algorithm allows the distribution of an optimization problem over several sampling periods if the problem is too large to be solved during a single sampling period.

4.4 Performance improvement using parametrization

In section 4.3 it was shown that the behaviour becomes unstable if the computation speed is not sufficient to perform enough iterations during one sampling period. For the tracking scenario shown in Figure 5 the algorithm was unstable if less than 9 iterations per sampling period were performed.

In the depicted tracking scenario this is not a real problem since the computation speed in real-time was sufficient to perform 14 iterations per sampling period which is clearly enough to guarantee stability. In more challenging systems this can become a limiting factor for using this optimization method in its pure form.

A general approach to solve this kind of instability problem due to insufficient computation speed in MPC is to reduce the number of degrees of freedom of the problem. One way to achieve this is to choose a shorter prediction horizon. However this is not a good idea in many cases, because a shorter prediction horizon often leads to instability itself.

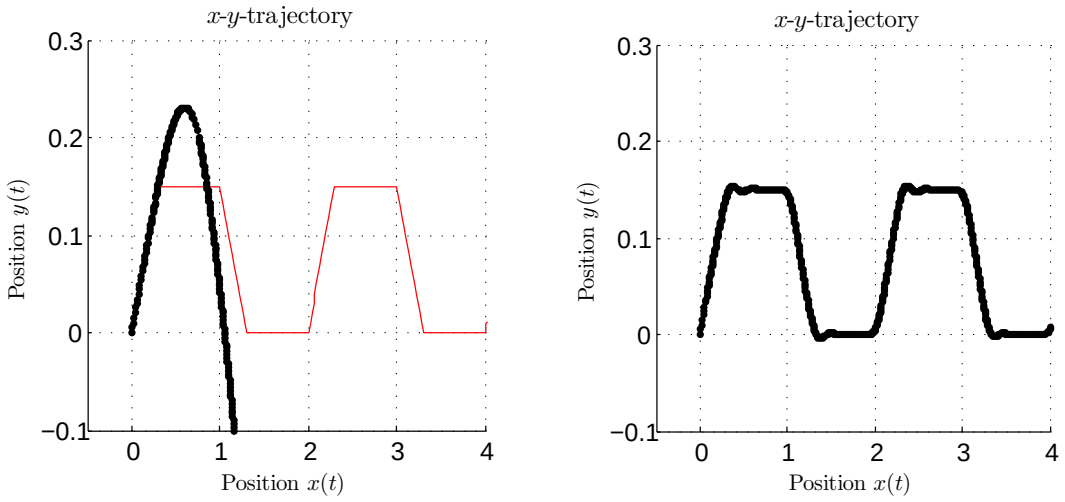
An alternative way to reduce the number of degrees of freedom of a system is to use some parametrization method which allows to maintain the same horizon length for the price of loosing some flexibility in the control profile. Very often this loss of flexiblity in the control profile has a negligible impact on the stability and controllability of the system. For this reason, a parametrization method is proposed in this section and applied to the same scenario as in section 4.3.

In order to parametrize a piecewise constant control input profile one can imagine different possibilities such as an *exponential parametrization* or a *Fourier's parametrization*, both described in [3, p. 12]. However in a first step a more simple and more intuitive one is chosen.

The idea of the chosen parametrization is to group several consecutive control input components and always keep them at the same value. Thus the number of degrees of feedom is divided by the number of grouped input components: $n_{dof} = n_{dof,0}/p$ where $n_{dof,0}$ is the number of degrees of freedom without parametrization and p is the number of grouped input components. For instance if $p = 2$ and the algorithm attempts to improve the input component $u_l^{(1)}$, $u_l^{(2)}$ would always take the same value. Figure 6 shows possible control profiles without parametrization and with a parametrization of $p = 2$.

Figure 7 shows the same tracking scenario as studied in section 4.3. In contrast, the number of iterations per sampling period (i.e. the computation speed) is not changing here, but the parametrization does. The left graph shows the resulting trajectory without parametrization (same result as the top left graph in Figure 5) which is clearly unstable. On the right graph one can see the resulting car trajectory when a parametrization with

$p = 2$ is implemented. This trajectory is clearly stable which means that the chosen parametrization approach is very performant for the chosen simple car model in combination with the implemented tracking scenario.



No parametrization

Parametrization $p = 2$

Figure 7: Resulting state trajectories with and without parametrization. The number of iterations per samling period is $N = 5$ in both cases.

In this section it was shown that an appropriate parametrization method can lead to strong improvements of the convergence speed and is a promising extension to the algorithm which might enable it to control much more complex as well as faster systems than the examined simple car.

5 Benefit of C coding

When investigating on an optimization algorithm for fast NMPC, the computation speed of the implemented scheme is one crucial factor limiting the algorithm's performances. For this reason this section is dedicated to showing the important benefit of coding the algorithm in C++ instead of MATLAB.

Many control engineers (including me) are not very comfortable with writing code in C++. In most cases this is also not necessary since MATLAB is more adapted to their needs anyway. However it does occur from time to time that there is a need of coding some pieces of code in C++. For this reason MATLAB developed the *MATLAB Coder* toolbox. This toolbox allows to automatically translate code written in MATLAB to C or C++. The resulting code is structured in a readable way and even comments added to the MATLAB code are maintained at the good places in the C++ files. In order to use this automatic code translation one simply has to respect some rules such as not to use specific MATLAB functions. Additionally it is recommended to use as few variable-sized arrays in the code as possible since dynamic memory allocation has a negative impact on the computation speed. Doing so the resulting C++ code accelerates the original MATLAB code by an important factor in many cases.

Measurement of the acceleration factor between C++ and MATLAB

In the following the acceleration factor between MATLAB and C++ is measured for the computation of 1000 iterations of the algorithm in the implementation for the simple car model discussed in section 4. In order to achieve a maximal computation speed using C++, only fixed sized variables were used so that the C++ code

Parameter	Value
Prediction horizon length T	1 sec
Sampling period τ	50 msec
Degree of freedom DOF	40

Table 2: Algorithm parameters for the statically coded C++ algorithm of the simple car model

does not use any dynamic memory allocation. For achieving this issue a bunch of changes has to be done in the code so that finally all dimensions of the problem are of fixed size.

Table 2 shows the algorithm configuration chosen for the static coding.

Resulting acceleration factor:

$$Acceleration = \frac{t_{MATLAB}}{t_{C++}} = \frac{1.7071\ msec}{0.1447\ msec} = 11.8 \tag{21}$$

Equation 21 shows that doing the same amount of computations in C++ instead of MATLAB can accelerate the computation by factor 11,8. This already is a very important gain in computation speed. However it is likely that if the C++ implementation was coded manually instead of using the automatic code translation from the MATLAB Coder, the computation time could even be reduced by a stronger factor, since the MATLAB Coder is not especially optimized for generating fast C++ code. Whether this assumption is true won't be subject to further investigations in this work as this would require an expertise in C++ coding which is not available for the moment.

6 Twin pendulum model

6.1 Introduction of the twin pendulum model

The twin pendulum is an extension of the famous simple inverted pendulum which has been subject to many research works in nonlinear control engineering such as in [8]. The twin pendulum consists of a cart whose movement on a horizontal rod can be controlled. On the cart, two pendulums of different mechanical characteristics (mass, inertia, length) are fixed so that they can rotate freely. Figure 8 shows the structure of the system. The objective is to steer the two pendulums to the upward position and to maintain them there.

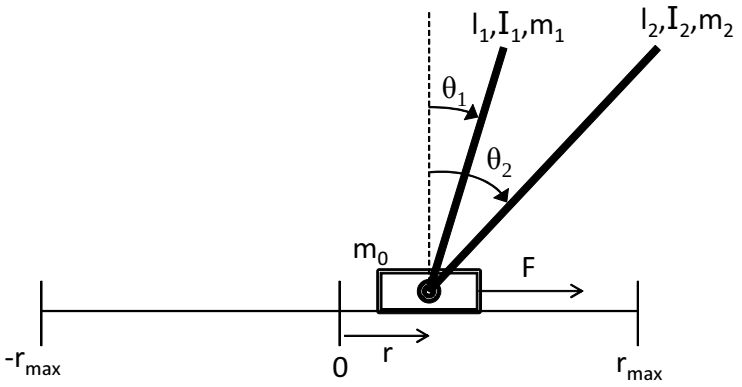


Figure 8: Mechanical structure of the twin pendulum system

Compared to the simple car model used in section 4 the twin pendulum is a much more complex system. After transforming the system to bilinear form (see equation (24)) the state space consists of 8 states while the simple car model has only 4 states. Moreover, only one control variable is available for the twin pendlum whereas the simple car model disposes of two controls. This already indicates that the twin pendulum is a much more challenging system to be controlled than the simple car.

The dynamic equations representing the model of the twin pendulum are as folows:

$$\begin{aligned} \ddot{r} &= u \\ \ddot{\theta_1} &= -\alpha_1 \cos\theta_1 u + \beta_1 \sin\theta_1 \\ \ddot{\theta_2} &= -\alpha_2 \cos\theta_2 u + \beta_2 \sin\theta_2 \end{aligned} \tag{22}$$

where:

$$\alpha_i = \frac{m_i l_i}{m_i {l_i}^2 + I_i} \quad ; \quad \beta_i = g\alpha_i \quad ; \quad i \in \{1, 2\}$$

Twin Pendulum in state space representation:

In state space representation, the system can be written as follows with state $\mathbf{x}(t) = \begin{bmatrix} r & \dot{r} & \theta_1 & \dot{\theta}_1 & \theta_2 & \dot{\theta}_2 \end{bmatrix}^\top$.

$$\dot{\mathbf{x}}(t) = \begin{cases} x_2 \\ u \\ x_4 \\ -\alpha_1 \cos x_3 u + \beta_1 \sin x_3 \\ x_6 \\ -\alpha_2 \cos x_5 u + \beta_2 \sin x_5 \end{cases} \tag{23}$$

Transformation to bilinear form:

Obviously this system is not of the desired bilinear form to which the optimization framework [1] addresses. But similiar to the Simple Car model from section 4, it can be easily transformed into the bilinear form. Extending the state by two additional states and adding two artificial and imposed inputs to the system leads to the bilinear representation:

$$\dot{\mathbf{x}}(t) = \begin{cases} x_2(t) \\ u_1(t) \\ x_4(t) \\ -\alpha_1 x_7(t) u_1(t) + \beta_1 u_2(t) \\ x_6 \\ -\alpha_2 x_8(t) u_1(t) + \beta_2 u_3(t) \\ -x_4(t) u_2(t) \\ -x_6(t) u_3(t) \end{cases} \tag{24}$$

where the following extensions were used:

$$\begin{aligned} x_7(t) &= \cos x_3(t) \\ x_8(t) &= \cos x_5(t) \\ u_2(t) &= \sin x_3(t) \\ u_3(t) &= \sin x_5(t) \end{aligned}$$

Finally the system is subject to constraints on the state x_1, limiting the range the cart can move on the rod, as well as to control constraints on the input acceleration u.

$$\begin{aligned} r(t) &\in [-r_{max}, r_{max}] \\ u(t) &\in [-u_{max}, u_{max}] \end{aligned} \tag{25}$$

6.2 Controller design

The objective of the controller is to steer the two pendulums to the upward position ($x_d := (0,0,0,0,0,0)^T$) while respecting the constraints on the cart position $r(t)$ as well as the constraint on the control input value $u(t)$. A possible control strategy expressed in terms of equation (12) can be written as follows:

$$\begin{aligned} l_1(x) &= \begin{cases} (x-x_d)^T Q (x-x_d) & if \quad |x_1| < r_{max} \\ (x-x_d)^T Q (x-x_d) + x_1^2 \rho & else \end{cases} \\ l_2(u) &= 0 \end{aligned} \tag{26}$$

Term $l_1(x)$ of equation (26) represents a weight on the distance of the system's state from the desired upward position x_d. Additionally the constraint violation of constraint $r \in [-r_{max}, r_{max}]$ is also handled in $l_1(x)$, since otherwise the notation would not have been conform with the general notation from equation (12). The handling of the constraints on the control input $u(t)$ is implemented in the parametrization strategy described in the following.

An important tuning parameter is the weighting vector Q on the different states of the system. It was chosen as $Q := (1, 0.001, 10, 0.5, 10, 0.5)^T$. The highest weight is on the distance of the pendulum angles θ_1 and θ_2 from the upward position so that it is guaranteed that the strongest interest of the controller is to steer the two pendulums close to the upward position. The relative high weight on the cart position ($Q_1 = 1$) has the objective to avoid that the ρ-weighted term x_1^2 becomes active too often as this leads to a hybrid controller-like behaviour which might cause instability. However this can not always be avoided as the bottom left graph in Figure 9 shows.

Local stabilizing controller

During the effected simulations it appeared that the implemented NMPC controller is not suitable to maintain both pendulums in the upright position once the state is very close to x_d. This is not surprising since the whole controller design was focused on the swing-up process. For this reason the system switches to a stabilizing linear controller once the system is close to the state x_d. This linear controller is implemented as a simple discrete LQR-controller.

In order to avoid constraints violation due to the linear controller, the trajectory that would result after the switch is precomputed for some sampling periods in order to see whether any constraints would be violated. In case constraints on r or on u were violated, the switch is not performed. In this contribution a more detailed description of the linear controller design is not provided as it is not in the center of interest in this work. However the detailed description can be found in [3, p. 162-165] where exactly the same strategy is proposed.

Parametrization strategy

Since the twin pendulum is a strongly under-actuated system with only the acceleration $\ddot{r}$ as an input, steering the two pendulums from the downward in rest position to the upward position while respecting the constraints on the input and on the state is a very challenging task which, as one can imagine, might require a long prediction horizon.

In section 4.4 the potential benefits of using parametrization techniques have been demonstrated. However

Admissible range	Description
$p_1 \in [0.01, 0.5]$	The amplitude is positive and upper bounded
$p_2 \in [-3, 0]$	Exponential part can only decrease the amplitude
$p_3 \in [0.2, 6]$	The frequency of the sinusoidal part is bounded
$p_4 \in [-\pi, \pi]$	Bounds on the phase shift
$p_5 \in [-0.1, 0.1]$	The offset is upper- and lower bounded

Table 3: Admissible ranges for the 5 components $p_1, ..., p_5$ of the parametrization profile.

the demonstrated parametrization where several sequal control inputs were grouped and then treated as one degree of freedom is not a convenient method for the twin pendulum, because a more complex control profile is desirable when using a long prediction horizon and also taking into account that the system's dynamics have sinusoidal behaviour.

From these considerations one can imagine that a parametrization of some sinus-like shape might be a convenient choice. The finally chosen parametrization is given in equation (27) which has a sinus of variable frequency (p_3) and with a variable phase shift (p_4) as a basis. Additionally, the sinusoidal base is multiplied by an exponential function (p_2), allowing to increase or decrease the amplitude over the prediction horizon and a variable amplitude (p_1) allowing to amplify or damp the whole input profile. Finally an additional offset (p_5) is added to the sinusoidal profile. This is necessary, because the cart would only be able to move to one direction otherwise which is due to the characteristics that the cart position results from the double integration of the input.

Parameterization equation:

$$u(t) = \left[p_1 e^{p_2 t} \sin(p_3 t + p_4)\right] + p_5 \tag{27}$$

Using this parametrization method, the algorithm is not attempting to decrease the cost function by variations on the input components $u_l^{(i)}$ directly, but by variations of the whole control profile $u(t)$ through the parametrization parameters $\{p_1, ..., p_5\}$. This also leads to the fact that for every parameter variation, the cost function has to be computed for the whole prediction horizon, since every parameter has an effect on the predicted state and control trajectory over the whole horizon. This means that the positive characteristic of the basic algorithm, allowing to reduce the computation time by working on the partial cost when variations on input components $u_l^{(i)}$ for values $i > 1$ are done, is lost in the extension with this sinusoidal parametrization strategy. On the other hand, the degree of freedom is reduced that importantly ($n_{dof} = 5$) while maintaining a flexible control profile over the whole prediction horizon, that this loss is more than compensated.

As mentioned above the control input $u(t)$ is subject to the constraint $u(t) \in [-u_{max}, u_{max}]$. This constraint is respected by restricting the admissible range for the parameters $\{p_1, ..., p_5\}$ as given in Table 3. Since the term containing $\{p_1, ..., p_4\}$ in equation (27) can only take the maximal value $p_{1,max} = 0.5$ (the exponential part (p_2) can only decrease the term over time and the sinusoidal part (p_3, p_4) is limited to 1), the constraint on $u(t)$ is respected if u_{max} is chosen as $u_{max} > (p_{1,max} + p_{5,max} = 0.6)$.

6.3 Simulation results

For the simulation of the system the same mechanical parameters as in [3, p. 167] are used since they correspond to an actually available experiment of the twin pendulum in Gipsa-lab. The parameters are given in Table 4.

Figure 9 shows the simulated closed-loop behaviour of the system using a prediction horizon of $T = 3sec$ and a sampling period of $\tau = 0.03sec$. The initial state of the system is chosen such that both pendulums are in the downward position with initial angular velocities 0. Also the initial cart position and speed are 0. $x(0) = [0, 0, \pi, 0, \pi, 0]^T$. After about 32 seconds both pendulums arrive at the desired upright position (see top left curve). The curves of the cart position (top right) and of the control input (bottom left) show that

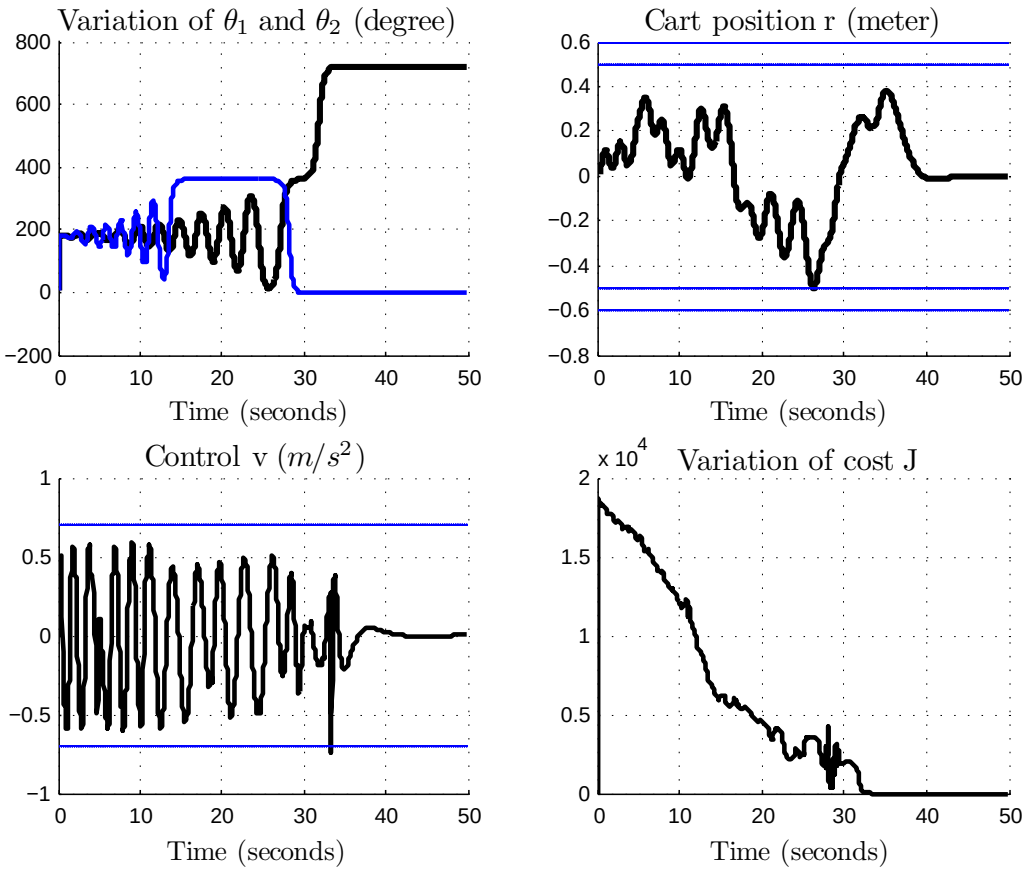


Figure 9: Closed-loop behaviour of the twin pendulum system. Initial state $x(0) = (0, 0, \pi, 0, \pi, 0)^T$, sampling period $\tau = 0.03sec$, prediction horizon length $T = 3sec$. The curve of the cart position $r(t)$ (top right) shows that the constraint r_{max} is well respected. The same is true for the constraint on the control input $u(t)$ (bottom left).

the constraints are respected during the whole swing-up process. The development of the cost $J(t)$ is almost continuously decreasing until it arrives at 0 which corresponds to the situation where the system is in the desired state with both pendulums in the upright position.

6.4 Comparison with ACADO optimization toolkit

The objective of this section is to provide an idea of where to classify the proposed optimization framework's performance in comparison with other numerical optimization solutions for NMPC problems.

For this reason the *ACADO optimization tookit* (see [6], [7]) is used as a reference to compare the algorithm's performances to. ACADO provides a set of optimization algorithms which address to a very large range of nonlinear systems and can be considered as one of the state-of-the-art optimization toolkits for nonlinear systems at the moment. It also provides the possibility to implement an NMPC controller and to apply it to a simulated process in real-time. This feautre is going to be used in this contribution.

As the optimization framework which is subject in this work and ACADO use very different principles of numerical optimization, the first difficulty in comparing their performances to each other is to find a scenario that allows to do some resonable comparison of their performances. The scenario consists of the system to be

Parameter	Value	Unity
m_0	2.0	kg
m_1	0.2	kg
m_2	0.1	kg
l_1	1.0	m
l_2	0.5	m
I_1	0.1	kgm^2
I_2	0.0125	kgm^2

Table 4: Mechanical parameters of the twin pendulum system.

controlled and of the configurations of the two optimization methods which have to be chosen in a way that they provide as many essential common characteristics as possible.

Comparison scenario

The chosen dynamic system which is subject to the two optimization frameworks is the classical simple inverted pendulum. It is the same system as the one described in section 6.1 with the difference that only one pendulum is mounted to the cart and can rotate freely. The objective of the controller is to steer the pendulum from the initial downward position to the upward position and to maintain it there.

The dynamic equation of this system in state space representation with state $\mathbf{x}(t) = \begin{bmatrix} r & \dot{r} & \theta & \dot{\theta} \end{bmatrix}^\top$ is as follows:

$$\dot{\mathbf{x}}(t) = \begin{cases} x_2 \\ u \\ x_4 \\ -\alpha \cos x_3 u + \beta \sin x_3 \end{cases} \tag{28}$$

The reason why the inverted pendulum is chosen instead of the *twin pendulum* from the previous sections is that for the inverted pendulum it is not necessary to use a specific parameterization such as the one from equation (27). Instead, the parameterization method using grouped input components (see section 4.4) is sufficient in that case. The benefit of using this parameterization is that it makes the scenario more similiar for the two optimization methods since ACADO is only able to handle piecewise constant control profiles without further restrictions to its shape. Another advantage of the simple inverted pendulum is that it is not necessary to implement a stabilizing linear controller for the case where the pendulum is close to the upward stable position, because the MPC controller is able to stabilize the pendulum itself.

The NMPC configuration is chosen equal for both optimization methods with a prediction horizon length of $T = 1.8$ seconds and a sampling period of $\tau = 30$ milliseconds. These configurations are going to remain fixed in the following discussion.

Variation of the degree of freedom

Having fixed the prediction horizon length and the sampling period, the remaining variable parameter for both optimization methods is the division of the control profile into a certain number of piecewise constant pieces. The number of pieces into which the control profile is departed, is considered as the degree of freedom (n_{dof}) of the optimization problem. In the following the degree of freedom is going to be varied in the range $n_{dof} \in [5, 12]$.

Remarks:

- For the ACADO NMPC controller the amount of iterations performed by the algorithm during one sampling period is fixed to 1. This means that during every sampling period the ACADO controller

performs exactly the minimal possible amount of computations which are necessary to provide a valid control input at the end of the sampling period. If the computation time for a single iteration was higher than the sampling period τ, the controller would not be real-time compatible.

- As shown in section (4.3), achieving the control objective with the component-wise optimization approach requires a certain amount of iterations that have to be performed during each sampling period. For this reason, the number of iterations performed during one sampling period is now fixed to the experimentally determined value $N_{iterations} = 11$ which guarantees that for all variations of the degree of freedom that are simulated in this section, the control objective of steering the pendulum to the upward position is achieved.

- The comparison between the two optimization frameworks is finally done in terms of computation time which each of the two controllers needs to simulate the system over a simulated time of $T_{sim} = 40$ seconds. Note that as long as the measured computation time is lower than the simulated time period T_{sim}, the controllers would be able to achieve these performances in real-time. For the following simulations, this is always the case. Note also that the computation times obtained for the ACADO controller are the absolute minimum values for each n_{dof} whereas for some n_{dof}, the component-wise optimization controller might also achieve the control objective if even less iterations per sampling period were performed.

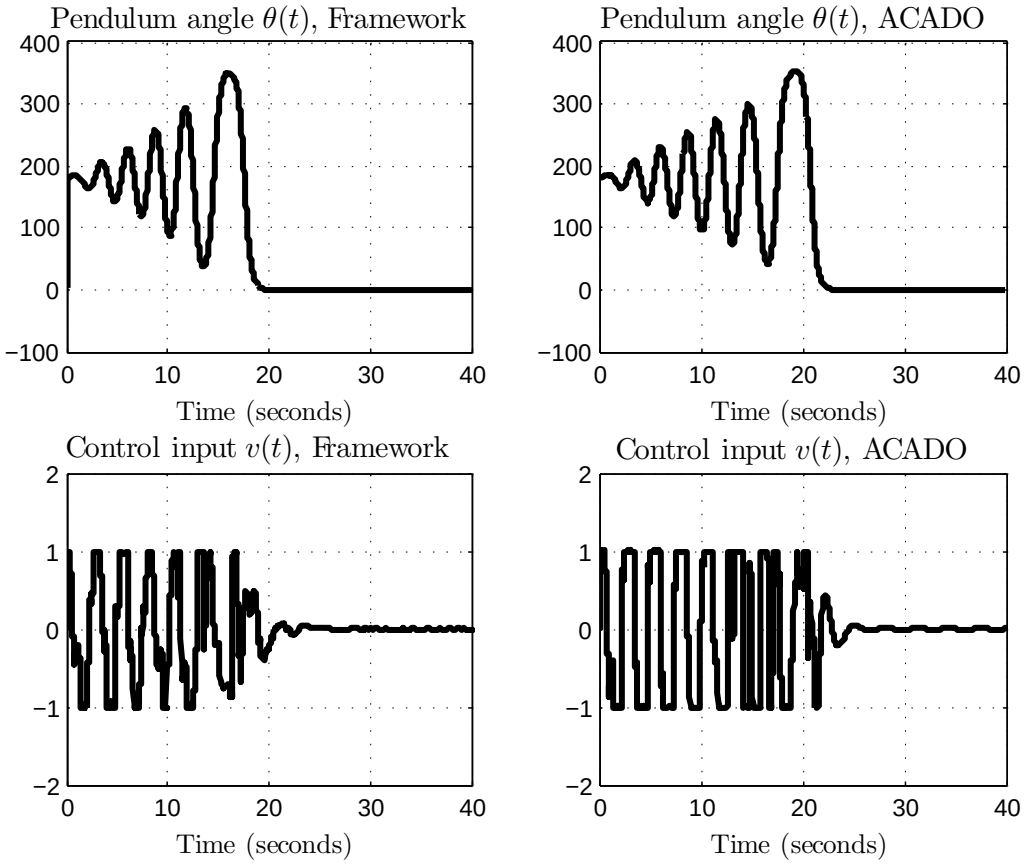


Figure 10: State- and control trajectories of the inverted pendulum for both optimization methods and for a fixed degree of freedom $n_{dof} = 6$. The resulting plots show that both algorithms determine results which are very close to each other. This shows that both methods are able to minimize the cost function with comparable precision.

Figure 10 shows the resulting state and control trajectories from the two optimization methods for a fixed degree of freedom of $n_{dof} = 6$. This shows clearly that both controllers are able to steer the pendulum to the upward position in about the same amount of time. Also in the determined control profile similarities can be seen, especially when looking at the period where the pendulum is close to its final position. Note that this figure is representative for all simulated variations of the degree of freedom in the range $n_{dof} \in [5, 12]$.

Figure 11 shows the simulation times that were measured for the ACADO optimization algorithm and for the proposed optimization framework for different degrees of freedom in the control profile. A first observation one can make is that the simulation time using ACADO is generally longer than for the optimization framework. Additioally one can see that the simulation time of ACADO increases with the degrees of freedom whereas for the optimization framework the simulation time is hardly influenced by the different degrees of freedom of the control profile. The reason why one iteration of ACADO takes so much longer is that ACADO is using a Newton like descent method which takes all degrees of freedom of the system over the whole prediction horizon into account. This means that in each iteration, ACADO has to perform a time consuming preparation step in which all the dependencies of the problem (system dynamics, constraints, complexity of the control profile) have to be considered. The big advantage of ACADO is that once the preparation step is completed, it takes only one iteration to determine a local minimum.

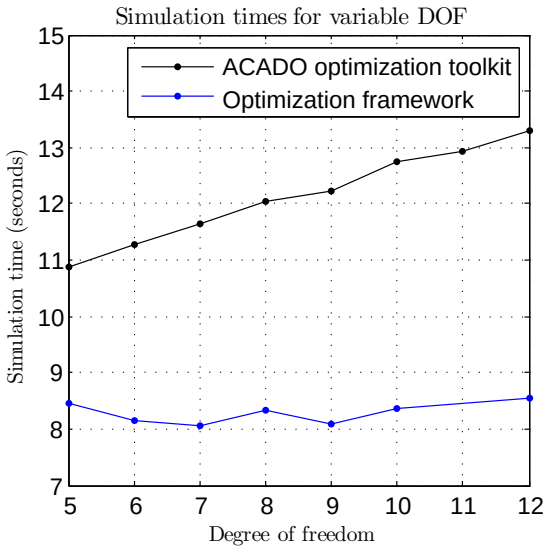


Figure 11: Time needed to simulate the inverted pendulum for 40 seconds using the ACADO optimization toolkit and the examined optimization framework for bilinear systems for different degrees of freedom of the piece-wise constant control profile. It can be seen that the computation times of ACADO increase with the degree of freedom whereas this is not the case for the examined optimization framework. Another benefit of this framwork compared to ACADO is that the minimal computation time which is necessary to achieve the objective of steering the pendulum to the upward position is smaller, allowing faster sampling periods in general.

The conclusions one can make based on this comparison between the proposed optimization framework and the ACADO optimization toolkit can be summarized as follows:

- The shortest possible computation time of ACADO is fixed by the computation time of one iteration. Note that one iteration is sufficient for ACADO to achieve good performance which is not the case for the proposed optimization framework since the computations performed during one iteration of the framework are fundamentally different from those performed in one iteration of ACADO. Note also that the settings used for ACADO's numerical optimization method are the default settings. However these settings seem to be those that provide the fastest possible computation since variations of the possible settings such as the integrator tolerance or the integrator type did not lead to a considerable reduction of the computation time.
- The shortest possible computation time for the proposed optimization framework is theoretically lower bounded by the time it needs to improve one input component (:= one iteration). However in reality, several iterations have to be performed during one sampling period to achieve stability. For the depicted example it was shown that the computation time to compute the necessary amount of iterations such

that the control objective is achieved, is smaller than the minimal possible computation time of ACADO (time one iteration needs). This means that for this system the framework requires a lower amount of computations to meet the control objective than ACADO does. Thus one could imagine a system where ACADO would not be able to meet the real-time requirements since its computation time for one iteration is higher than the sampling period, but the optimization framework would.

- ACADO's computation time for one iteration increases with the complexity of the control profile (degree of freedom). In opposition to this, the curve for the optimization framework shows that the impact of the degree of freedom on the computation time is negligible. This is logical also from a theoretical point of view, because the same amount of iterations is performed during each sampling interval independently of the degree of freedom which should also lead to about the same computation times.

- The fact that the computation times of the optimization framework were generally lower than for ACADO in this scenario is not a general characteristic. It is very likely that for other systems the optimization framework needs more iterations per sampling period to achieve the control objective while ACADO would still achieve it with one iteration. In that case the computation time of the optimization framework would increase while ACADO's computation time stayed the same. However it is promising that for this first and randomly chosen inverted pendulum example the proposed optimization framework could show better performances in terms of computation time than the state-of-the-art ACADO optimization toolkit.

7 Conclusion

7.1 Summary

In this work the fast NMPC optimization framework for bilinear systems proposed in [1] was studied and an evaluation of its performances was done.

In section 4 the algorithm was applied to a model of simple car dynamics. In a first step its main functionality was demonstrated pointing out how the weight on constraints is dynamically adapted. In a second step it was shown that a certain number of iterations per sampling period is required in order to achieve stability. Finally, for the case where the computation time does not suffice to perform enough iterations per sampling period, a parameterization strategy was proposed leading to a reduction in the degree of freedom of the control profile and to an important increase in stability.

In section 5 the benefit in terms of computation time when coding the algorithm in C++ instead of MATLAB was shown. It is remarkable that the achieved acceleration of the algorithm's computation speed was obtained using the *MATLAB Coder* which generates C++ code automatically from MATLAB code and does not require any knowledge about coding in C++.

In section 6 the optimization framework was applied to the model of a twin pendulum which is a much more challenging system than the simple car model. It was shown that using a specific control parameterization, the objective of steering both pendulums to the upward position could be achieved. Furthermore a comparison of the algoritm with the *ACADO optimization toolit* was performed. It could be demonstrated that for the depicted system of a simple inverted pendulum the algorithm's performance is comparable to the one of *ACADO* which can be considered as one of the state-of-the-art optimization toolkits.

7.2 Comments and future work

- Extensions to the optimization framework other than the parameterization methods shown in this contribution could be studied with the objective of improving the global convergence speed. One possible and

promising extension would be a variable control updating period (proposed in [4] where it is used in combination with a fast gradient based descent method). This would allow the algorithm to work on the same optimization problem for several sampling intervals and maybe lead to a globally faster decrease of the cost function than when shifting the horizon after each sampling period. A first attempt of implementing this to the simple car model could not lead to considerable improvements but it is likely that the reason lay rather in the implemented scenario's structure than in the method itself.

- Since the computation speed of the algorithm is a crucial factor that limits the range of possible applications of the proposed optimization framwork, more rigorous investigations on how to code the algorithm most effectively should be done. Although a high gain in computation speed could already be achieved by coding the algorithm in C++ using the *MATLAB Coder*, it is likely that coding the algorithm manually in C++ by someone who possesses expert knowledge in effective C++ coding would lead to considerably higher computation speeds.

- One aspect of the algorithm which was not considered at all in this work is the fact that it does not only apply to convex control profiles but also to control profiles which are nonconvex such as for instance a discrete control profile allowing only a limited number of control values. Interesting results can be expected when the algorithm is applied to systems of this kind. A large field of application one can imagine are for example power-electronic circuits where the admissible control values would be the configuration of the electronic switches. In that case one could imagine that due to the restricted number of control configurations, taking advantage of the controller memory could importantly reduce the computations if for instance the possible transition matrices $\Phi(\mathbf{u})$ were precomputed and stored.

References

[1] M. Alamir, "A framework for fast NMPC implementation on bilinear systems," *not yet published*, 2012.

[2] ——, "Nonlinear receding horizon sub-optimal guidance law for the minimum interception time problem," *Control Engineering Practice*, vol. 9, no. 1, pp. 107–116, 2001.

[3] ——, *Stabilization of Nonlinear Systems Using Receding-horizon Control Schemes*. London: Springer, 2006, vol. 339.

[4] ——, "Monitoring control updating period in fast gradient based NMPC," *CoRR*, vol. abs/1209.4922, 2012.

[5] M. Alamir and A. Murilo, "Swing-up and stabilization of a twin-pendulum under state and control constraints by a fast NMPC scheme," *Automatica*, vol. 44, no. 5, pp. 1319 – 1324, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109807004517

[6] D. Ariens, B. Houska, H. Ferreau, and F. Logist, "Acado: Toolkit for automatic control and dynamic optimization," Optimization in Engineering Center (OPTEC), http://www.acadotoolkit.org/.

[7] ——, *ACADO for Matlab User's Manual*, 1st ed., Optimization in Engineering Center (OPTEC), May 2010, http://www.acadotoolkit.org/.

[8] K. J. Astrom and K. Furuta, "Swinging up a pendulum by energy control," *13th IFAC World Congress*, 1996.

[9] M. Diehl, H. Bock, and J. Schlder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, 2005. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/S0363012902400713

[10] J. W. Eaton and J. B. Rawlings, "Model-predictive control of chemical processes," *Chemical Engineering Science*, vol. 47, no. 4, pp. 705 – 720, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/000925099280263C

[11] Murilo, A., Alamir, M., and Ortner, P., "A real-time implementable NMPC output feedback for a diesel engine air path," *Oil Gas Sci. Technol. - Rev. IFP Energies nouvelles*, vol. 66, no. 4, pp. 613–625, 2011. [Online]. Available: http://dx.doi.org/10.2516/ogst/2011120

[12] T. Ohtsuka, "A continuation/gmres method for fast computation of nonlinear receding horizon control," *Automatica*, vol. 40, no. 4, pp. 563 – 574, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109803003637

[13] Y. Ou, C. Xu, E. Schuster, T. Luce, J. Ferron, M. Walker, and D. Humphreys, "Optimal tracking control of current profile in tokamaks," *Control Systems Technology, IEEE Transactions on*, vol. 19, no. 2, pp. 432–441, 2011.