Copyright © 2014 IEEE

Reprinted from
Proceedings of the 2014 European Control Conference

Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The Gradient Based Nonlinear Model Predictive Control Software GRAMPC

Bartosz Käpernick and Knut Graichen

Abstract—This paper presents the nonlinear model predictive control (MPC) software **GRAMPC** (**GR**Adient based **MP**C – [**gr**æmp'si:]) which is suited for controlling nonlinear systems with input constraints in the (sub)millisecond range. **GRAMPC** is based on a real-time solution strategy in combination with a (projected) gradient method. It is written in plain C with an interface to MATLAB/SIMULINK and also provides a graphical user interface (GUI) in MATLAB for a convenient MPC design and tuning. The performance of **GRAMPC** is demonstrated by two examples from different technical fields. Additionally, some comparison results with established MPC software are provided.

I. INTRODUCTION

Model predictive control (MPC) has become a popular and powerful control methodology due to its ability to handle nonlinear multiple input systems as well as constraints [1], [2]. It is based on the solution of an underlying optimal control problem (OCP) which is solved online at each sampling instance and typically requires a considerable computational effort. This drawback usually limits the applicability of model predictive control to sufficiently slow or low-dimensional processes.

There exist several MPC approaches and software packages to cope with the problem of real-time applicability for linear and nonlinear systems. The presented methods in [3], [4], [5] are suited for linear, time discrete systems and use efficient solution strategies, interior point approaches tailored to convex multistage problems and first order methods. The software environment ACADO Toolkit [6] provides a module to generate optimized C code for real-time MPC which can be used to control nonlinear systems. An alternative nonlinear MPC approach developed in [7] employs a continuation method to trace the solution of the optimality conditions over a single MPC step based on a generalized minimum residual (GMRES) method.

This paper presents the portable and user-friendly nonlinear MPC software **GRAMPC** (**GR**Adient based **MP**C – [**gr**æmp'si:]). It is based on a (projected) gradient method from optimal control [8], [9] and includes a real-time solution strategy for controlling nonlinear systems with input constraints and real-time demands. **GRAMPC** is written in plain C without the use of external libraries resulting in a small code size and a high level of portability in view of different operating systems and platforms. It is well suited to handle fast and high-dimensional systems and also provides a user-friendly integration to MATLAB/SIMULINK with an additional graphical user interface (GUI) in MATLAB

which can be used in an online MPC design procedure. The performance and real-time applicability of **GRAMPC** is demonstrated by means of two examples from different technical areas.

The paper is organized as follows: Section II defines the general problem formulation as well as **GRAMPC**'s underlying algorithm. The structure of **GRAMPC** is discussed in Section III which also includes the setup of a problem formulation and the corresponding interaction with **GRAMPC**. Section IV is devoted to the simulation examples and Section V finally concludes the paper.

II. PROBLEM FORMULATION AND ALGORITHM

In the following, the problem formulation is introduced that can be handled with **GRAMPC**. The corresponding optimality conditions are used to derive a gradient based algorithm with two suitable line search strategies.

A. MPC formulation

Throughout this paper, an MPC scheme based on the following OCP is considered:

$$\min_{u(\cdot)} J(u, x_k) = V(T, x(T)) + \int_0^T L(\tau, x(\tau), u(\tau)) d\tau \quad (1a)$$

$$\text{s.t.} \quad \dot{x}(\tau) = f(t_k + \tau, x(\tau), u(\tau)), \quad x(0) = x_k \quad (1b)$$

$$u(\tau) \in [u^-, u^+], \quad \tau \in [0, T] \quad (1c)$$

with states $x \in \mathbb{R}^n$ and controls $u \in \mathbb{R}^m$. The functional (1a) contains the continuously differentiable terminal cost $V : \mathbb{R}_+ \times \mathbb{R}^n \rightarrow \mathbb{R}_+^0$ and integral cost $L : \mathbb{R}_+^0 \times \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}_+^0$ which are both positive semi-definite functions. The vector field $f : \mathbb{R}_+^0 \times \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ in (1b) describes the dynamics of a time-varying system and is also assumed to be continuously differentiable in its arguments. The initial condition $x(0) = x_k$ denotes the measured (or observed) state of the system at the sampling instance $t_k = t_0 + k\Delta t$ with sampling time Δt . Condition (1c) represents the input constraints and $T > 0$ is the prediction horizon.

It is assumed, that OCP (1) possesses the optimal solution

$$u_k^*(\tau) := u^*(\tau; x_k), \quad x_k^*(\tau) := x^*(\tau; x_k), \quad \tau \in [0, T] \quad (2)$$

where the subindex k indicates the corresponding sampling time t_k with initial state x_k . In general, an MPC computes the optimal solution (2) in each sampling instance and the first part of the optimal control trajectory is used as input for the system on the interval $\tau \in [0, \Delta t)$, i.e.

$$u(t_k + \tau) = u_k^*(\tau), \quad \tau \in [0, \Delta t). \quad (3)$$

In the next sampling step $t_{k+1} = t_k + \Delta t$, OCP (1) has to be solved again with the new initial state x_{k+1} .

This work was funded by the Austrian Science Fund (FWF): P21253N22
B. Käpernick and K. Graichen are with the Institute of Measurement, Control, and Microtechnology, University of Ulm, Germany
{bartosz.kaepernick, knut.graichen}@uni-ulm.de

B. Optimality conditions and gradient algorithm

In order to derive the optimality conditions of OCP (1), the Hamiltonian¹

$$H(x, u, \lambda) = L(x, u) + \lambda^\top f(x, u) \quad (4)$$

with the costates $\lambda \in \mathbb{R}^n$ is introduced. The necessary optimality conditions for the optimal solutions u_k^* , x_k^* and λ_k^* can then be stated by means of Pontryagin's Maximum Principle [10], [11]

$$\dot{x}_k^* = f(x_k^*, u_k^*), \quad x_k^*(0) = x_k \quad (5a)$$

$$\dot{\lambda}_k^* = -H_x(x_k^*, u_k^*, \lambda_k^*), \quad \lambda_k^*(T) = V_x^*|_{t=T} \quad (5b)$$

$$u_k^* = \arg \min_{u \in [u^-, u^+]} H(x_k^*, u, \lambda_k^*), \quad \tau \in [0, T] \quad (5c)$$

where $H_x := \partial H / \partial x$ and $V_x := \partial V / \partial x$ denote partial derivatives w.r.t. the states x . The separated boundary conditions in (5a), (5b) are due to the OCP formulation (1) without terminal conditions.

The optimality conditions (5) can be solved by means of the (projected) gradient method [8], [9]. To this end, GRAMPC performs the following steps for solving OCP (1):

- Initialization of input trajectory $u_k^{(0)}$.
- Gradient iterations for $j = 0, \dots, N_{\text{grad}} - 1$
 - 1) Forward integration in time of (5a) to obtain $x_k^{(j)}$.
 - 2) Backward integration in time of (5b) to obtain $\lambda_k^{(j)}$.
 - 3) Solving the line search problem

$$\alpha^{(j)} = \arg \min_{\alpha > 0} J\left(\psi\left(u_k^{(j)} - \alpha g_k^{(j)}\right), x_k\right) \quad (6)$$

with search direction $g_k^{(j)} = H_u(x_k^{(j)}, u_k^{(j)}, \lambda_k^{(j)})$ and projection function

$$\psi(u) = \begin{cases} u^- : u < u^- \\ u : u \in [u^-, u^+] \\ u^+ : u > u^+ \end{cases} \quad (7)$$

- 4) Update control $u_k^{(j+1)} = \psi\left(u_k^{(j)} - \alpha^{(j)} g_k^{(j)}\right)$.

A limited number of gradient iterations N_{grad} is performed in each MPC step in order to achieve real-time feasibility of the approach, i.e. the control used as input for the system is

$$u(t_k + \tau) = u_k^{(N_{\text{grad}})}(\tau), \quad \tau \in [0, \Delta t] \quad (8)$$

where the last iteration is additionally used in the next sampling instance to re-initialize the controls. A convergence and stability analysis regarding the projected gradient method as well as the (prematurely stopped) MPC scheme can be found in [9] and [12], respectively.

C. Line search strategies

A crucial role is the effective solution of the line search problem (6). An appropriate method to solve (6) is the adaptive line search approach from [8], where the cost (1a)

is evaluated at three sample points $\alpha_1 < \alpha_2 < \alpha_3$ in order to construct a quadratic polynomial approximation of the form

$$J\left(\psi\left(u_k^{(j)} - \alpha g_k^{(j)}\right), x_k\right) \approx \phi(\alpha) = c_0 + c_1 \alpha + c_2 \alpha^2. \quad (9)$$

Subsequently, the step size is computed according to

$$\alpha^{(j)} = \arg \min_{\alpha \in [\alpha_1, \alpha_3]} \phi(\alpha) \quad (10)$$

where the cost approximation (9) is minimized w.r.t. α . If the minimum point $\alpha^{(j)}$ is close to the bounds α_1 and α_3 or lies outside the interval $[\alpha_1, \alpha_3]$, then the interval is adapted by a scaling factor to approach the minimum point in the next gradient iteration, i.e.

$$[\alpha_1, \alpha_3] \leftarrow \begin{cases} \kappa [\alpha_1, \alpha_3] : \alpha^{(j)} \geq \bar{\alpha} \ \& \ \alpha_3 \leq \alpha_{\max} \\ \frac{1}{\kappa} [\alpha_1, \alpha_3] : \alpha^{(j)} \leq \underline{\alpha} \ \& \ \alpha_1 \geq \alpha_{\min} \\ [\alpha_1, \alpha_3] : \text{otherwise} \end{cases} \quad (11a)$$

$$\alpha_2 \leftarrow \frac{\alpha_1 + \alpha_3}{2} \quad (11b)$$

with $\bar{\alpha} = \alpha_1 + (1 - \varepsilon_\alpha)(\alpha_3 - \alpha_1)$ and $\underline{\alpha} = \alpha_1 + \varepsilon_\alpha(\alpha_3 - \alpha_1)$. The adaptation factor $\kappa > 1$ and the interval tolerance $\varepsilon_\alpha \in (0, 1)$ control the interval adaptation and hence the minimum tracking. Investigations revealed that bounding the step size interval $[\alpha_1, \alpha_3]$ maintained better convergence properties during each gradient step. In this regard, $\alpha_{\min}$ and $\alpha_{\max}$ denote the minimum and maximum value, respectively. In addition, the adaptation procedure is illustrated in Fig. 1.

The polynomial approximation of the cost function (9) is implemented efficiently in GRAMPC and is automatically performed in each gradient iteration. In the case of a non-quadratic cost function an appropriate fitting strategy is performed for the approximation. This approach provides good results for computing the step size $\alpha^{(j)}$ due to the fixed number of operations and the tracking of the minimum over the gradient iterations j and MPC steps k .

To further reduce the computational effort for time-critical problems, an alternative way to determine the step size is the explicit line search approach discussed in [13] that can be adapted to the optimal control formulation of the MPC scheme. The motivation is to minimize the difference between two consecutive control updates for the same step size and without considering constraints, i.e.

$$\alpha^{(j)} = \arg \min_{\alpha > 0} \left\| \Delta u_k^{(j)} - \alpha \Delta g_k^{(j)} \right\|_{L_m^2[0, T]}^2 \quad (12)$$

Fig. 1. Adaptation of line search interval.

¹In the remainder of the paper, the time argument is omitted where it is convenient to maintain readability.

Fig. 2. General structure of GRAMPC (gray – C code, white – MATLAB).

with $\Delta u_k^{(j)} := u_k^{(j)} - u_k^{(j-1)}$ and $\Delta g_k^{(j)} := g_k^{(j)} - g_k^{(j-1)}$. In the nominal case without any disturbances this approach provides only small deviations in the vicinity of the optimal solution. In view of (12), the minimum step size $\alpha^{(j)}$ can be determined by the condition

$$0 = \frac{\partial}{\partial \alpha} \left\| \Delta u_k^{(j)} - \alpha \Delta g_k^{(j)} \right\|_{L_m^2[0,T]}^2 \quad (13)$$

and hence for $\Delta g_k^{(j)} \neq 0$ the step size $\alpha^{(j)} = \alpha$ in the active gradient iteration follows to

$$\alpha^{(j)} = \frac{\int_0^T \left(\Delta u_k^{(j)} \right)^\top \Delta g_k^{(j)} dt}{\int_0^T \left(\Delta g_k^{(j)} \right)^\top \Delta g_k^{(j)} dt} =: \frac{\langle \Delta u_k^{(j)}, \Delta g_k^{(j)} \rangle}{\langle \Delta g_k^{(j)}, \Delta g_k^{(j)} \rangle}. \quad (14)$$

As it can be seen from (14), the approach needs to store the control trajectory and the gradient from the previous iteration. Then two separate integrations must be performed along the prediction horizon. Thus, the explicit line search strategy can be implemented efficiently and provides an alternative way to compute a suitable step size $\alpha^{(j)}$ with regard to time critical applications.

III. STRUCTURE OF GRAMPC

The aim of GRAMPC is to be portable and executable on different operating systems and platforms without the use of external libraries. To this end, GRAMPC is implemented in plain C with a user-friendly interface to MATLAB/SIMULINK. The general structure is illustrated in Fig. 2.

A. Problem formulation for GRAMPC

The problem formulation (1) is provided to GRAMPC using a C function template, c.f. Fig. 2. It contains the system dimensions (number of states and controls), the nonlinear system function $f(x, u)$ and the terminal and integral cost $V(x)$ and $L(x, u)$, respectively. In order to evaluate the optimality conditions (5) time efficiently, the user has to provide the related Jacobians $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial u}$ and the partial derivatives $\frac{\partial L}{\partial x}$, $\frac{\partial L}{\partial u}$, $\frac{\partial V}{\partial x}$. Alternatively, the product functions $(\frac{\partial f}{\partial x})^\top \lambda$ and $(\frac{\partial f}{\partial u})^\top \lambda$ that appear in the partial derivatives H_x and H_u of the Hamiltonian can be provided to increase the numerical efficiency for sparsely populated Jacobians $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial u}$.

Fig. 3. Interface of GRAMPC to MATLAB/SIMULINK (gray – C code, white – MATLAB).

A code extract of the problem function for a simple double integrator is shown in Table I. The variables pSys and pCost are pointers containing optional parameters of the dynamics (1b) and the cost (1a).

TABLE I

CODE EXTRACT OF A PROBLEM FUNCTION FOR A DOUBLE INTEGRATOR.

```
void sysdim(typeRNum *Nx, typeRNum *Nu) {
    *Nx = 2; // number of states
    *Nu = 1; // number of controls
}
void sysfct(typeRNum *out, [...]) {
    out[0] = pSys[0]*x[1]; // x1' = c*x2
    out[1] = u[0];        // x2' = u1
}
[...]
void icostfct(typeRNum *out, [...]) {
    // l = x2^2 + r*u1^2
    out[0] = x[1]*x[1] + pCost[0]*u[0]*u[0];
}
[...]
```

B. Interaction with GRAMPC

After the setup of a problem formulation, the provided input data can be linked with GRAMPC's main components to build an executable program. Fig. 3 illustrates the main functions for initializing and running GRAMPC as well as for setting parameters and options, respectively. Parameter values are related to OCP (1) and are partly mandatory (e.g. initial values, setpoints) and optional (e.g. number of discretizations, input constraints) whereas the options concern the algorithmic behaviour of GRAMPC (e.g. number of gradient iterations, line search strategy). The gradient algorithm implemented in GRAMPC relies on the integration of the system and adjoint dynamics (5a) and (5b), respectively. Hence, different integrators with fixed (e.g. Euler forward method or Heun method) as well as variable integration steps (Runge-Kutta method of order 4) are provided. GRAMPC can directly be implemented on alternative real-time platforms since all components (such as the problem formulation and the integrators) are written in plain C without including external libraries. In this regard, a high level of portability can be achieved.

Additionally, each main function of GRAMPC has a related MEX-routine (cf. Fig. 3) which allows running GRAMPC

and altering parameters and options in MATLAB/SIMULINK without re-compilation, as illustrated in Fig. 2. In order to demonstrate the use of GRAMPC, another code extract of the MPC formulation in MATLAB for the double integrator is given in Table II.

TABLE II
CODE EXTRACT OF AN MPC FORMULATION FOR GRAMPC.

```
[...]
% INITIALIZATION
grampc = grampc_init_Cmex();
% MANDATORY PARAMETERS
grampc_setparam_Cmex(grampc,'xk',[0,1]);
grampc_setparam_Cmex(grampc,'u0',0);
grampc_setparam_Cmex(grampc,'xdes',[0,0]);
grampc_setparam_Cmex(grampc,'udes',0);
grampc_setparam_Cmex(grampc,'Thor',1.5);
grampc_setparam_Cmex(grampc,'dt',0.01);
% OPTIONAL PARAMETERS
grampc_setparam_Cmex(grampc,'umin',-0.2);
grampc_setparam_Cmex(grampc,'umax',+0.2);
% NO. OF ADDITIONAL SYSTEM AND COST PARAMETERS
grampc_setparam_Cmex(grampc,'NpSys',1);
grampc_setparam_Cmex(grampc,'NpCost',1);
% VALUES OF ADDITIONAL SYSTEM AND COST PARAMETERS
grampc_setparam_Cmex(grampc,'pSys',2);
grampc_setparam_Cmex(grampc,'pCost',0.1);
% OPTIONS
grampc_setopt_Cmex(grampc,'MaxIter',2);
grampc_setopt_Cmex(grampc,'Integrator','heun');
% MPC LOOP
for i = 1:iMPC
    % RUN GRAMPC
    [x,u,J] = grampc_run_Cmex(grampc);
    [...]
    % SET NEW INITIAL STATE
    grampc_setparam_Cmex(grampc,'xk',xnew);
    [...]
end
[...]
```

First of all GRAMPC is initialized by defining the structured variable `grampc` containing all default options, parameters and additional information with regard to an MPC iteration, cf. Table II. In the following lines, parameters and options can be selected and finally GRAMPC is started and updated with a new initial state in each MPC iteration. After GRAMPC has finished its computation, it provides the predicted new state, the new control and the corresponding cost value for the current sampling time as output arguments. Moreover, the determined state at the next sampling instant is set by means of the parameter function.

In addition, a MATLAB GUI is also provided for parameterizing the controller in an interactive way allowing a fast and convenient design procedure. The interface is shown in Fig. 4 and is basically divided in three parts with regard to GRAMPC's parameter and option settings as well as plot properties to present the corresponding results. Before the MPC can be run, a problem formulation and an initialization file (containing initial user parameters and options) must be loaded. Then the MPC can either be started for an entire time interval (simulation time) or just for a single MPC step and the resulting closed-loop and predicted trajectories can be illustrated.

IV. DEMONSTRATION OF GRAMPC BY TWO EXAMPLES

In this section two examples from different technical fields are considered in order to demonstrate the performance

and applicability of GRAMPC for both complex and highly dimensional and nonlinear problems.

A. Quasi-linear diffusion-convection-reaction system

An example for a large-scale system is the quasi-linear diffusion-convection-reaction system presented in [14]. It is described by the partial differential equation (PDE)

$$\partial_t \theta(z, t) = \partial_z [(q_0 + q_1 \theta(z, t)) \partial_z \theta(z, t) - \nu \theta(z, t)] + (r_0 + r_1 \theta(z, t)) \theta(z, t), \quad z \in (0, 1), \quad t > 0 \quad (15)$$

with the boundary and initial conditions

$$\partial_z \theta(z, t)|_{z=0} = 0 \quad (16)$$

$$\partial_z \theta(z, t)|_{z=1} + \theta(1, t) = u(t), \quad t > 0 \quad (17)$$

$$\theta(z, 0) = \theta_0(z), \quad z \in [0, 1], \quad (18)$$

where $\partial_z := \frac{\partial}{\partial z}$ and $\partial_t := \frac{\partial}{\partial t}$ are partial derivatives w.r.t. z and t . The variable z denotes the spatial coordinate and the parameters are $q_0 = 2$, $q_1 = -0.05$, $\nu = 1$, $r_0 = 1$ and $r_1 = 0.2$. By applying finite differences on an equidistant grid, an ordinary differential equation (ODE) of the form (1b) can be derived where the number of states corresponds to the number of discretizations. To this end, a large-scale formulation with $n = 100$ states is considered in the following. Note that the states, the control, the time and all parameters are normalized which lead to non-dimensional system dynamics.

In order to complete the formulation of an underlying OCP (1) for GRAMPC, the corresponding cost (1a) is set to

$$V(x) = \Delta x^T P \Delta x, \quad L(x, u) = \Delta x^T Q \Delta x + r \Delta u^2 \quad (19)$$

with $Q = P = 0.5 \cdot I_n$ and $r = 0.005$. The matrix I_n denotes the unity matrix and the variables $\Delta x = x - x_{SP}$ and $\Delta u = u - u_{SP}$ describe the distance to a desired setpoint (x_{SP}, u_{SP}) . Moreover, the input constraints are $u^\pm = \pm 3$.

In the following simulation, GRAMPC is used to control the system from an initial state x_0 to a setpoint (x_{SP}, u_{SP}) . The MPC parameters of GRAMPC are chosen to $T = 0.2$ (prediction horizon) and $\Delta t = 0.005$ (sampling time) where no units are specified at this point according to the normalization of the system. Due to stiffness of the dynamics, GRAMPC's Runge-Kutta integrator with variable step size is used in connection with the explicit line search strategy. The

Fig. 4. GRAMPC's GUI for use in MATLAB.

Fig. 5. Model predictively controlled quasi-linear diffusion-convection-reaction system with $n = 100$ states.

simulation is run on a standard PC with MATLAB 2013a (64-bit) and Windows 7 (64-bit) on an Intel Core i5 CPU with 2.67 GHz and 4 GB memory.

Fig. 5 shows the closed-loop trajectories for the large-scale setup with $n = 100$ states and $N_{\text{grad}} = 2$ gradient iterations where only each 10-th state is plotted for a clear presentation of the results. It can be observed that the input constraints are satisfied and that a good control performance can be achieved with GRAMPC. The required average and maximum computation time for a single MPC step consisting of the gradient as well as the line search updates amounts to 623 ms and 701 ms, respectively. The rather large computation times are mainly caused by using an integrator with a variable step size due to the stiffness of the system.

B. Quadrotor

Fig. 6 shows the configuration of a quadrotor [15]. The states $x = [X, \dot{X}, Y, \dot{Y}, Z, \dot{Z}, \gamma, \beta, \alpha]^T$ of the system are the translational position (X, Y, Z) to the inertial coordinate system O , the related velocities $(\dot{X}, \dot{Y}, \dot{Z})$ and the Euler angles (γ, β, α) describing the orientation of the vehicle with regard to the body-fixed coordinate system V . The mass-normalized thrust a and the angular rates $(\omega_X, \omega_Y, \omega_Z)$ serve as controls $u = [a, \omega_X, \omega_Y, \omega_Z]^T$. The nonlinear dynamics is given by [15]

$$\begin{aligned}\ddot{X} &= a(C\gamma S\beta C\alpha + S\gamma S\alpha), \quad \dot{\gamma} = (\omega_X C\gamma + \omega_Y S\gamma) / C\beta \\ \ddot{Y} &= a(C\gamma S\beta S\alpha - S\gamma C\alpha), \quad \dot{\beta} = -\omega_X S\gamma + \omega_Y C\gamma \\ \ddot{Z} &= aC\gamma C\beta - g, \quad \dot{\alpha} = \omega_X C\gamma T\beta + \omega_Y S\gamma T\beta + \omega_Z\end{aligned}\quad (20)$$

with the abbreviated functions $Sz := \sin z$, $Cz := \cos z$, $Tz := \tan z$ and the gravitational constant g .

A cost (1a) is chosen with the terminal and integral part

$$V(x) = \Delta x^T \Delta x, \quad L(x, u) = \Delta x^T \Delta x + \Delta u^T \Delta u \quad (21)$$

Fig. 6. Quadrotor setup and control inputs.

with $x_{\text{SP}} = 0$ and $u_{\text{SP}} = [9.81 \text{ m/s}^2, 0, 0, 0]^T$ for a hovering position of the quadrotor. Additionally, the quadrotor dynamics (20) are subject to the input constraints

$$u^+ = [11 \text{ m/s}^2, 1 \text{ rad/s}, 1 \text{ rad/s}, 1 \text{ rad/s}]^T \quad (22a)$$

$$u^- = [0, -1 \text{ rad/s}, -1 \text{ rad/s}, -1 \text{ rad/s}]^T. \quad (22b)$$

For simulation studies, the control task is to stabilize the origin (i.e. $x = 0$) starting from the initial state

$$x_0 = [-3 \text{ m}, 0, -3 \text{ m}, 0, -3 \text{ m}, 0, 0, 0, 0]^T \quad (23)$$

which corresponds to moving the quadrotor by 3 m in each translational direction. The sampling time, the prediction horizon and the number of gradient iterations per MPC step are set to $\Delta t = 2 \text{ ms}$, $T = 1.5 \text{ s}$ and $N_{\text{grad}} = 2$. The numerical integration of the system and adjoint dynamics (5a) and (5b), respectively, are performed by GRAMPC's Euler forward integration method with 30 discretization points.

The simulation with the setup described above was performed on a standard PC (see IV-A for more details) as well as on a dSPACE DS1103 system (with a PPC 750 GX CPU with 1 GHz and 96 MB memory) to demonstrate the real-time feasibility. In this regard, the predicted states of the MPC were used as feedback. In addition to the nominal stabilization case where the states can perfectly be determined without any disturbance or measurement noise, a noisy case is also illustrated where the states are subject to zero-mean Gaussian noise with variance $\sigma = 10^{-2}$ in order to demonstrate robustness issues.

Fig. 7 shows the simulation results for the model predictively controlled quadrotor with the adaptive line search strategy². It can be seen that the input constraints (22) are satisfied and that a very good control performance can be achieved. It can also be observed that the trajectories for the noisy case provide good performance as well.

Table III shows the computation times of GRAMPC running on the PC and the dSPACE system, respectively. To this end, the SIMULINK block of GRAMPC was used for the simulation and the PC computation times were measured by means of the SIMULINK profiler. Although a direct comparison with alternative MPC solvers is quite difficult due to different solution approaches and algorithms (different options/parameters), Table III also provides results for established linear as well as nonlinear MPC software. Note however, that a thorough benchmark is not the aim of the paper and hence the results intend to demonstrate roughly where GRAMPC stands in terms of performance. As regards the comparison, the code generation tool included in the ACADO Toolkit³ [6], AutoGenU⁴ [7], the FMPC code⁵ [3] as well as FORCES and FiOrdOs⁶ [4], [5] were applied for stabilizing the origin in an MPC fashion. All MPC solvers were also implemented within an s-function for use in MATLAB/SIMULINK where the computation times were measured using the SIMULINK

²The results with the explicit line search are almost identical and hence are omitted here due to the lack of space.

³Explicit Euler forward integrator with 20 steps and 10 control intervals.

⁴30 integration steps with a maximum of four iterations per MPC step.

⁵Three newton iterations, 10 prediction intervals and the related barrier parameter set to 10^{-3} .

⁶10 prediction intervals and a maximum of 20 iterations (FiOrdOs).

Fig. 7. Model predictively controlled quadrotor without (top) and with measurement noise (bottom).

profiler for multiple MPC runs. The illustrated cost values are determined by using the computed closed-loop trajectories along the entire simulation time in combination with (21). Table III reveals that all computation times are well below the sampling time. It can also be observed that GRAMPC is very competitive in view of the control performance (cost value) as well as the computation speed for the quadrotor example. Hence, the two gradient iterations provide a good tradeoff between the required computational burden and accuracy of the trajectories. Additionally, the presence of measurement noise barely affects the computation times but leads to a poorer control performance as the cost value of the closed-loop trajectories indicates (see also Fig. 7).

V. CONCLUSION

This paper described the gradient based MPC software GRAMPC which is well suited to control nonlinear and input constrained systems in terms of real-time demands. The main

TABLE III
COMPUTATION TIME AND COST VALUE FOR THE QUADROTOR.

		Average comp. [μ s]		Cost [-]	
		nominal	noisy	nominal	noisy
GRAMPC	adapt.	91 (919) ^a	97 (920) ^a	45.150	48.459
GRAMPC	expl.	72 (546) ^a	70 (545) ^a	45.090	50.319
ACADO	(nonl.)	286	294	45.112	48.269
AutoGenU	(nonl.)	246	—	49.949	—
FORCES	(lin.)	476	472	46.202	48.831
FMPC	(lin.)	617	621	45.680	48.476
FiOrdOs	(lin.)	222	212	46.997	49.527

^aComputation times for the dSPACE system.

components are written in plain C without the use of external libraries in order to obtain a high level of portability in view of different operating systems and platforms. In addition, GRAMPC features an interface to MATLAB/SIMULINK with a GUI for a convenient MPC design procedure in MATLAB.

The performance of GRAMPC was demonstrated for a large-scale quasilinear diffusion-convection-reaction system and a quadrotor model. The simulation results illustrate the good control performance that was achieved by means of GRAMPC with computation times in the range of milli- and microseconds, respectively. Comparison results with established MPC software were also provided confirming GRAMPC's competitiveness. A dSPACE simulation and a scenario with measurement noise for the quadrotor example demonstrated the real-time applicability and robustness properties of GRAMPC.

GRAMPC is licensed under the GNU Lesser General Public License (version 3) and can be downloaded from <http://sourceforge.net/projects/grampc>. Future work concerns the extension of GRAMPC to account for state constraints and the integration of new algorithmic options, e.g. the fixed point scheme [16] for numerical acceleration.

REFERENCES

- [1] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.
- [2] E. Camacho and C. Bordons, *Model Predictive Control*. London: Springer, 2003.
- [3] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.
- [4] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *Proceedings of the 51th IEEE Conference on Decision and Control*, 2012, pp. 668–674.
- [5] C. N. Jones, A. Domahidi, M. Morari, S. Richter, F. Ullmann, and M. Zeilinger, "Fast predictive control: Real-time computation and certification," in *Proceedings of the 4th IFAC Nonlinear Predictive Control Conference*, 2012, pp. 94–98.
- [6] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.
- [7] T. Ohtsuka, "A continuation/GMRES method for fast computation of nonlinear receding horizon control," *Automatica*, vol. 40, no. 4, pp. 563–574, 2004.
- [8] K. Graichen and B. Käpernick, "A real-time gradient method for nonlinear model predictive control," in *Frontiers of Model Predictive Control*, T. Zheng, Ed. InTech, 2012, pp. 9–28, <http://www.intechopen.com/articles/show/title/a-real-time-gradient-method-for-nonlinear-model-predictive-control>.
- [9] J. C. Dunn, "On l^2 sufficient conditions and the gradient projection method for optimal control problems," *SIAM Journal on Control and Optimization*, vol. 34, no. 4, pp. 1270–1290, 1996.
- [10] D. E. Kirk, *Optimal Control Theory: An Introduction*. Mineola: Dover Publications, 1970.
- [11] L. D. Berkovitz, *Optimal Control Theory*. New York: Springer, 1974.
- [12] K. Graichen and A. Kugi, "Stability and incremental improvement of suboptimal MPC without terminal constraints," *IEEE Transactions on Automatic Control*, vol. 55, no. 11, pp. 2576–2580, 2010.
- [13] J. Barzilai and J. M. Borwein, "Two-point step size gradient method," *IMA Journal of Numerical Analysis*, vol. 8, no. 1, pp. 141–148, 1988.
- [14] T. Utz, K. Graichen, and A. Kugi, "Trajectory planning and receding horizon tracking control of a quasilinear diffusion-convection-reaction system," in *Proceedings of the 8th IFAC Symposium on Nonlinear Control Systems*, 2010, pp. 587–592.
- [15] M. Hehn and R. D'Andrea, "A flying inverted pendulum," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2011, pp. 763–770.
- [16] K. Graichen, "A fixed-point iteration scheme for real-time model predictive control," *Automatica*, vol. 48, no. 7, pp. 1300–1305, 2012.