GRAMPC documentation

(pronounced gramp-c [græmp'si:])

Bartosz Käpernick, Knut Graichen, Tilman Utz
Institute of Measurement, Control, and Microtechnology
University of Ulm
{bartosz.kaepernick,knut.graichen}@uni-ulm.de

February 10, 2014

Contents

1 Introduction

This manual describes the model predictive control tool GRAMPC that is suited for nonlinear systems with control constraints. GRAMPC uses an efficient projected gradient algorithm with an adaptive line search strategy in order to allow an MPC implementation for highly dynamical systems with sampling times in the (sub)millisecond sampling range and/or high-dimensional systems. GRAMPC is implemented as C code with an additional user interface (GUI) to MATLAB/SIMULINK. A fixed number of gradient iterations is used in each new MPC step and the current solution is used as new initialization (warm-start) to successively refine the predicted MPC trajectories over the single MPC steps.

This manuel describes the algorithm behind GRAMPC, the installation and implementation details as well as the usage of the MATLAB interface that allows for a user-friendly MPC tuning. In addition, several example problems for fast and high-dimensional systems are included in GRAMPC and are described in the last part of this document.

2 Installation

The following sections describe the installation procedure of GRAMPC for use in C and MATLAB on a Linux and MS Windows operating system. The essential software parts of GRAMPC are written in self-contained C code and hence do not require external libraries.

2.1 Installation on Linux

An appropriate C compiler is required in order to compile GRAMPC correctly. Most Linux distributions include the gcc compiler. If this is not the case, please check your Linux distribution on how to install a C compiler.

2.1.1 Installation of GRAMPC for use in C

In order to install GRAMPC, perform the following steps:

1. The current version of GRAMPC can be downloaded from the following web page https://sourceforge.net/projects/grampc/.

2. Unpack the archive grampc_v1.0.zip to an arbitrary location on your computer. After the unpacking procedure, a new directory with the following subfolders is created:
   - **doc**: manual of GRAMPC,
   - **src**: source files of GRAMPC,
   - **include**: header files of GRAMPC,
   - **libs**: contains a GRAMPC library after successful compilation,
   - **examples**: examples to run GRAMPC,
   - **matlab**: interface of GRAMPC to MATLAB with GUI.

   The GRAMPC directory additionally contains a makefile. In the remainder, the location of the new created folder will be denoted by <grampc_root>.

3. GRAMPC is compiled by running the following commands in a Linux terminal:

   ```
   $ cd <grampc_root>
   $ make clean all
   ```

   The dollar symbol "$" indicates the line prompt of the terminal. The clean target removes previously installed parts of GRAMPC. The make command compiles the source files and generates the library grampc.a, which can now be used to solve a suitable problem.

2.1.2 Installation of GRAMPC for use in Matlab

A recent version of the gcc compiler is required in order to use GRAMPC directly in MATLAB. Details on supported compilers for the current MATLAB version as well as previous releases can be found via the MATHWORKS homepage. The correct linkage of the compiler to MATLAB can be checked by typing

```
>> mex -setup
```

in the MATLAB terminal window and subsequently selecting the corresponding C compiler. The symbol "`>>`" denotes the MATLAB prompt. The `GRAMPC` installation under MATLAB proceeds in two steps:

1. After downloading and unpacking `GRAMPC` as described in Section 2.1.1, go to the MATLAB directory

   ```
   >> cd <grampc_root>/matlab
   ```

   which contains the following subfolders:

   - **bin**: contains object files of `GRAMPC` after a successful building process,
   - **examples**: examples to run `GRAMPC` in MATLAB,
   - **GUI**: MATLAB graphical user interface (GUI) for `GRAMPC`,
   - **simulink**: files for various MATLAB releases to use `GRAMPC` in SIMULINK,
   - **src**: MEX files which provide the interface between `GRAMPC` and MATLAB.

   In addition, the subfolder contains the m-file `make.m` to start the building process.

2. Build the necessary object files for `GRAMPC` by executing the make function. The compilation of the source files can be performed with the following options:

   - `>> make clean`: removes all previously built `GRAMPC` files,
   - `>> make`: creates the necessary object files to use `GRAMPC`,
   - `>> make verbose`: the object files are created in verbose mode, i.e. additional information regarding the building process are provided during the compilation,
   - `>> make debug`: the debug option creates the object files with additional information for use in debugging,
   - `>> make debug verbose`: activates the debug option as well as the verbose mode.

   After the compilation, `GRAMPC` can be used in MATLAB.

2.2 Installation on MS Windows

This section shortly addresses the installation procedure of `GRAMPC` on MS Windows systems, which are basically the same as under Linux.

2.2.1 Installation of `GRAMPC` for use in C

A convenient way to use `GRAMPC` in C under MS Windows is the Linux environment Cygwin. To install Cygwin, download the setup file from the web page `http://www.cygwin.com/` and follow the installation instructions. In the installation process when packages can be selected, you have to choose the `gcc` compiler and `make`. If Cygwin is properly installed, open a Cygwin terminal and follow the steps described in Section 2.1.1.

2.2.2 Installation of `GRAMPC` for use in Matlab

The MATLAB installation procedure of `GRAMPC` on MS Windows systems corresponds to the description in Section 2.1.2. Depending on your MATLAB version, you might have to download and install a suitable C compiler manually. Take a look at the MATHWORKS homepage to figure out which compiler is supported by your MATLAB version. The installation of `GRAMPC` then follows the two instruction steps in Section 2.1.2.

3 Problem formulation and algorithm

GRAMPC is a software package that implements a real-time model predictive control scheme for nonlinear systems by solving an optimal control problem on a moving horizon. This chapter describes the basic algorithm and implementational aspects that are of interest for the algorithmic settings of GRAMPC. Further details on the algorithm and stability properties can be found in [6, 5, 8].

3.1 Problem formulation

The MPC scheme that is implemented in GRAMPC is based on the optimal control problem (OCP)

$$\min_{u(\cdot)} \quad J(u, x_k) = V(T, x(T)) + \int_0^T L\left(\tau, x(\tau), u(\tau)\right) \, \mathrm{d}\tau \tag{3.1a}$$

$$\text{s.t.} \quad \dot{x}(\tau) = f\left(t_k + \tau, x(\tau), u(\tau)\right) \,, \quad x(0) = x_k \tag{3.1b}$$

$$u(\tau) \in \left[u^-, u^+\right], \quad \tau \in [0, T] \tag{3.1c}$$

with state $x \in \mathbb{R}^{N_x}$ and control $u \in \mathbb{R}^{N_u}$ subject to the box constraints (3.1c). The functional (3.1a) with the prediction horizon $T > 0$ contains the final cost $V : \mathbb{R}_0^+ \times \mathbb{R}^{N_x} \to \mathbb{R}_+^0$ and integral cost $L : \mathbb{R}_0^+ \times \mathbb{R}^{N_x} \times \mathbb{R}^{N_u} \to \mathbb{R}_+^0$ which usually are assumed being positive semi-definite functions. The nonlinear system function $f : \mathbb{R}_0^+ \times \mathbb{R}^{N_x} \times \mathbb{R}^{N_u} \to \mathbb{R}^{N_x}$ describes the system dynamics (3.1b). 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. All functions are assumed to be continuously differentiable in their arguments. The MPC formulation (3.1) uses the internal time τ within the prediction interval $[0, T]$. Please note that the terminal cost V as well as the integral cost L contain an explicit time dependency with regard to the internal MPC time. In addition, the considered dynamics (3.1b) features an explicit time dependency and hence the current sampling time t_k also has to be provided.

The optimal solution of (3.1) at sampling instant t_k is denoted by

$$u_k^*(\tau) := u^*(\tau; x_k) \,, \quad x_k^*(\tau) := x^*(\tau; x_k) \,, \quad \tau \in [0, T] \tag{3.2}$$

with the optimal cost value $J^*(x_k) := J(u^*, x_k)$. The subindex k indicates the solution to the corresponding sampling instance t_k with initial state x_k. In general, an MPC scheme aims at computing the optimal solution (3.2) in each sampling instance and uses the first part of the optimal control trajectory $u^*(t; x_k)$ as input for the system on the interval $\tau \in [0\,,\, \Delta t)$, i.e.

$$u(t_k + \tau) = u_k^*(\tau) \,, \quad \tau \in [0, \Delta t) \,. \tag{3.3}$$

In the next sampling step $t_{k+1} = t_k + \Delta t$, OCP (3.1) has to be solved again with the new initial state x_{k+1}. In the nominal case, the sampling point in the next MPC step is given by $x_{k+1} = x_k^*(\Delta t)$.

3.2 MPC algorithm

GRAMPC computes the solution of OCP (3.1) by a projected gradient method that relies on the respective first-order optimality conditions. To this end, define the Hamiltonian

$$H\left(\tau, x(\tau), u(\tau), \lambda(\tau)\right) = L(\tau, x(\tau), u(\tau)) + \lambda^\mathsf{T}(\tau) f\left(\tau + t_k, x(\tau), u(\tau)\right) \tag{3.4}$$

where $\lambda \in \mathbb{R}^{N_x}$ is the adjoint state or costate. The first-order optimality conditions for the optimal trajectories $u_k^*(\tau)$, $x_k^*(\tau)$ and $\lambda_k^*(\tau)$ follow from Pontryagin's Maximum Principle [9, 2]

$$\dot{x}_k^*(\tau) = f\left(\tau + t_k, x_k^*(\tau), u_k^*(\tau)\right), \qquad x_k^*(0) = x_k \tag{3.5a}$$

$$\dot{\lambda}_k^*(\tau) = -H_x\left(\tau, x_k^*(\tau), u_k^*(\tau), \lambda_k^*(\tau)\right), \qquad \lambda_k^*(T) = V_x\left(T, x_k^*(T)\right) \tag{3.5b}$$

$$u_k^*(\tau) = \underset{u\in[u^-,u^+]}{\arg\min}\ H\left(\tau, x_k^*(\tau), u, \lambda_k^*(\tau)\right), \quad \tau \in [0,T], \tag{3.5c}$$

where $H_x := \partial H/\partial x$ and $V_x := \partial V/\partial x$ are partial derivative functions. `GRAMPC` solves the optimality conditions (3.5) based on a real-time implementation of a projected gradient algorithm [3, 10] that is outlined in Table 3.1.

Table 3.1: Gradient algorithm of `GRAMPC`.

- Initialization of input trajectory $u_k^{(0)}(\tau) \in [u^-, u^+]$ for all $\tau \in [0,T]$.
- Gradient iterations for $j = 0, \ldots, N$:
  1) Forward time integration of $\dot{x}_k^{(j)}(\tau) = f\left(\tau + t_k, x_k^{(j)}(\tau), u_k^{(j)}(\tau)\right)$ with initial condition $x_k^{(j)}(0) = x_k$.
  2) Backward time integration of $\lambda_k^{(j)}(\tau) = -H_x\left(\tau, x_k^{(j)}(\tau), u_k^{(j)}(\tau), \lambda_k^{(j)}(\tau)\right)$ with terminal condition $\lambda_k^{(j)}(T) = V_x\left(T, x_k^{(j)}(T)\right)$.
  3) (Approximate) solution of the line search problem

$$\alpha^{(j)} = \underset{\alpha>0}{\arg\min}\ J\left(\psi\left[u_k^{(j)}(\tau) - \alpha g_k^{(j)}\right], x_k\right) \tag{3.6}$$

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

$$\psi(u) = \begin{cases} u^- & : u < u^- \\ u & : u \in [u^-, u^+] \\ u^+ & : u > u^+. \end{cases} \tag{3.7}$$

  4) Control update $u_k^{(j+1)}(\tau) = \psi\left[u_k^{(j)}(\tau) - \alpha^{(j)} g_k^{(j)}\right]$, $\tau \in [0,T]$.

In order to maintain real-time feasibility of the overall MPC algorithm, a fixed number of gradient iterations N is performed in each MPC step. Hence, in contrast to applying the optimal solution $u_k^*(\tau)$ (cf. (3.2)), the control that is injected to the system is given by

$$u(t_k + \tau) = u_k^{(N)}(\tau), \quad \tau \in [0, \Delta t). \tag{3.8}$$

The last iteration is additionally used in the next sampling instance to re-initialize the controls. Convergence and stability analysis regarding the projected gradient method as well as the (prematurely stopped) MPC scheme can be found in [3] and [6, 5], respectively.

The control update requires the computation of a suitable step size $\alpha^{(j)}$ by solving the line search problem (3.6). The next sections demonstrate two different ways to determine a suitable step size.

3.3 Adaptive line search

An appropriate way to determine the step size is the adaptive line search approach from [5], where a polynomial approximation of the cost (3.1a) is used and an adaptation of the search intervals is performed. More precisely, the cost function is evaluated at three sample points $\alpha_1 < \alpha_2 < \alpha_3$ with $\alpha_2 = \frac{1}{2}(\alpha_1 + \alpha_3)$ which are used to construct a quadratic polynomial of the cost according to

$$J\left(\psi\left[u_k^{(j)} - \alpha g_k^{(j)}\right], x_k\right) \approx \phi(\alpha) = c_0 + c_1\alpha + c_2\alpha^2, \quad \alpha \in [\alpha_1, \alpha_3]. \tag{3.9}$$

Subsequently, a step size $\alpha^{(j)}$ is computed by minimizing the cost approximation (3.9). If necessary, the interval $[\alpha_1, \alpha_3]$ is adapted for the next gradient iteration in the following way

$$[\alpha_1, \alpha_3] \leftarrow \begin{cases} \kappa\,[\alpha_1, \alpha_3] & \text{if } \alpha \geqslant \alpha_3 - \varepsilon_\alpha(\alpha_3 - \alpha_1) \text{ and } \alpha_3 \leqslant \alpha_{\max} \\ \frac{1}{\kappa}\,[\alpha_1, \alpha_3] & \text{if } \alpha \leqslant \alpha_1 + \varepsilon_\alpha(\alpha_3 - \alpha_1) \text{ and } \alpha_3 \geqslant \alpha_{\min} \\ [\alpha_1, \alpha_3] & \text{otherwise} \end{cases} \tag{3.10a}$$

$$\alpha_2 \leftarrow \frac{1}{2}(\alpha_1 + \alpha_3) \tag{3.10b}$$

with the adaptation factor $\kappa > 1$, the interval tolerance $\varepsilon_\alpha \in (0, 1)$ and the interval bounds $\alpha_{\max} > \alpha_{\min} > 0$. The modification (3.10) of the line search interval tracks the minimum point $\alpha^{(j)}$ of the line search problem in the case when $\alpha^{(j)}$ is either outside of the interval $[\alpha_1, \alpha_3]$ or close to one of the outer bounds α_1, α_3 as illustrated in Figure 3.1. The adaptation factor κ accounts for scaling as well as shifting of the interval $[\alpha_1, \alpha_3]$ in the next gradient iteration, if $\alpha^{(j)}$ lies in the vicinity of the interval bounds $[\alpha_1, \alpha_3]$ as specified by the interval tolerance ε_α. This adaptive strategy allows one to track the minimum of the line search problem (3.6) over the gradient iterations j and MPC steps k, while guaranteeing a fixed number of operations in view of a real-time MPC implementation.

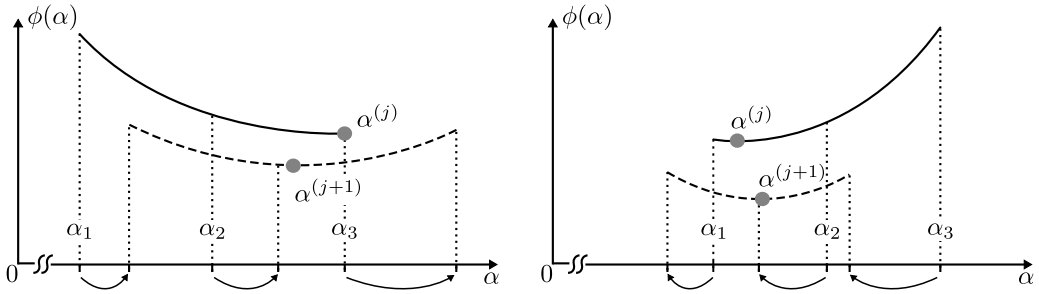

Figure 3.1: Adaptation of line search interval.

3.4 Explicit line search

An alternative way to determine the step size in order to further reduce the computational effort for time-critical problems is the explicit line search approach originally discussed in [1] and adapted in [8] for the optimal control case. The motivation is to minimize the difference between two consecutive control updates $u_k^{(j)}(\tau)$ and $u_k^{(j+1)}(\tau)$ in the unconstrained case and additionally assuming the same

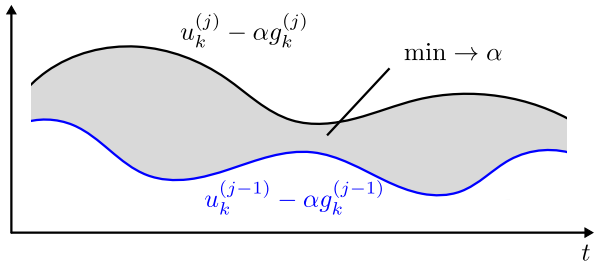

Figure 3.2: Motivation for the explicit line search strategy.

step size $\alpha^{(j)}$, i.e.[1]

$$\begin{aligned}\alpha^{(j)} &= \arg\min_{\alpha>0} \Big\| \underbrace{u_k^{(j+1)} - u_k^{(j)}}_{=:\Delta u_k^{(j+1)}} \Big\|^2_{L_m^2[0,T]} \\ &= \arg\min_{\alpha>0} \Big\| \underbrace{u_k^{(j)} - u_k^{(j-1)}}_{=:\Delta u_k^{(j)}} - \alpha \underbrace{\left(g_k^{(j)} - g_k^{(j-1)}\right)}_{=:\Delta g_k^{(j)}} \Big\|^2_{L_m^2[0,T]}\end{aligned} \tag{3.11}$$

with $||z||^2_{L_m^2[0,T]} = \langle z, z\rangle := \int_0^T z^\mathsf{T}(t) z(t) \mathrm{d}t$. Figure 3.2 illustrates the general idea behind (3.11). To solve (3.11), consider the following function

$$\begin{aligned} q(\alpha) :&= \left\| \Delta u_k^{(j)} - \alpha \Delta g_k^{(j)} \right\|^2_{L_m^2[0,T]} \\ &= \int_0^T \left(\Delta u_k^{(j)} - \alpha \Delta g_k^{(j)}\right)^\mathsf{T} \left(\Delta u_k^{(j)} - \alpha \Delta g_k^{(j)}\right) \mathrm{d}t \\ &= \int_0^T \left(\Delta u_k^{(j)}\right)^\mathsf{T} \Delta u_k^{(j)} \,\mathrm{d}t + \alpha^2 \int_0^T \left(\Delta g_k^{(j)}\right)^\mathsf{T} \Delta g_k^{(j)} \,\mathrm{d}t - 2\alpha \int_0^T \left(\Delta u_k^{(j)}\right)^\mathsf{T} \Delta g_k^{(j)} \,\mathrm{d}t. \end{aligned} \tag{3.12}$$

The minimum has to satisfy the stationarity condition

$$\frac{\partial q(\alpha)}{\partial \alpha} = 2\alpha \int_0^T \left(\Delta g_k^{(j)}\right)^\mathsf{T} \Delta g_k^{(j)} \,\mathrm{d}t - 2 \int_0^T \left(\Delta u_k^{(j)}\right)^\mathsf{T} \Delta g_k^{(j)} \,\mathrm{d}t = 0. \tag{3.13}$$

A suitable step size $\alpha^{(j)}$ then follows to

$$\alpha^{(j)} = \frac{\int_{t_k}^{t_k+T} \left(\Delta u_k^{(j)}\right)^\mathsf{T} \Delta g_k^{(j)} \,\mathrm{d}t}{\int_{t_k}^{t_k+T} \left(\Delta g_k^{(j)}\right)^\mathsf{T} \Delta g_k^{(j)} \,\mathrm{d}t} = \frac{\langle \Delta u^{(j)}, \Delta g_k^{(j)} \rangle}{\langle \Delta g_k^{(j)}, \Delta g_k^{(j)} \rangle}. \tag{3.14}$$

Another way to compute an appropriate step size for the control update can be achieved by reformulating (3.12) in the following way:

$$q(\alpha) = \left\| \Delta u_k^{(j)} - \alpha \Delta g_k^{(j)} \right\|^2_{L_m^2[0,T]} = \alpha^2 \left\| \frac{1}{\alpha} \Delta u_k^{(j)} - \Delta g_k^{(j)} \right\|^2_{L_m^2[0,T]} =: \alpha^2 \bar{q}(\alpha). \tag{3.15}$$

In the subsequent, the new function $\bar{q}(\alpha)$ is minimized w.r.t. the step size leading to a similar solution

$$\alpha^{(j)} = \frac{\langle \Delta u^{(j)}, \Delta u^{(j)} \rangle}{\langle \Delta u^{(j)}, \Delta H_u^{(j)} \rangle}. \tag{3.16}$$

[1] In the remainder, the function arguments are omitted where it is convenient to maintain readability.

In the GRAMPC implementation, both approaches (3.14) and (3.16) are available. In addition, the step size $\alpha^{(j)}$ is additionally bounded by an upper value $\alpha_{\max} > 0$. The adaptive or the explicit line search strategy, respectively, can be selected by the user via the algorithmic options (see Section 4.2).

4 Structure of GRAMPC

The aim of GRAMPC is to be portable and executable on different operating systems and hardware devices 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 of GRAMPC including the interfaces, algorithmic parameters and options as well as the resulting outputs is illustrated in Figure 4.1.

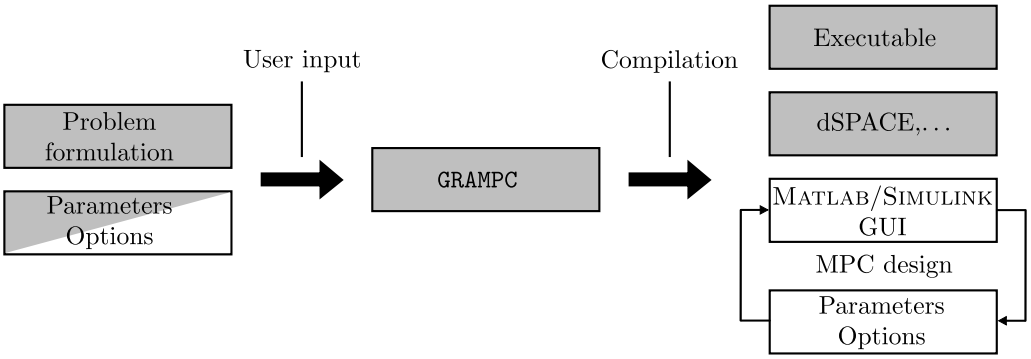

Figure 4.1: General structure of GRAMPC (gray – C code, white – MATLAB code)

4.1 Problem implementation

The problem formulation in terms of (3.1) has to be provided to GRAMPC via a C function template, c.f. Figure 4.1. For illustration purposes, consider the following double integrator example

$$\min_{u(\cdot)} \quad J(u,x_k) = x_1^2(T) + x_2^2(T) + \int_0^T x_2^2(\tau) + ru^2(\tau)\,\mathrm{d}\tau \tag{4.1a}$$

$$\text{s.t.} \quad \dot{x}_1(\tau) = cx_2(\tau)\,, \quad x_1(0) = x_{1,k} \tag{4.1b}$$

$$\dot{x}_2(\tau) = u(\tau)\,, \quad x_2(0) = x_{2,k} \tag{4.1c}$$

$$u(\tau) \in [u^-, u^+]\,, \quad \tau \in [0,T] \tag{4.1d}$$

which is implemented in the function template below.

Example (Problem formulation - double integrator)

```
void sysdim(typeInt *Nx, typeInt *Nu)
{
  *Nx = 2;  /* number of states   */
  *Nu = 1;  /* number of controls */
}

void sysfct(typeRNum *out, typeRNum t, typeRNum *x, typeRNum *u,
            typeRNum *pSys)
```

```
{
  out[0] = pSys[0]*x[1];  /* x1' = f1 = c*x2 */
  out[1] = u[0];          /* x2' = f2 = u    */
}

void sysjacx(typeRNum *out, typeRNum t, typeRNum *x, typeRNum *u,
             typeRNum *pSys)
{
  out[0] = 0.0;       /* df1dx1 */
  out[1] = 0.0;       /* df2dx1 */
  out[2] = pSys[0];   /* df1dx2 */
  out[3] = 0.0;       /* df2dx2 */
}

void sysjacu(typeRNum *out, typeRNum t, typeRNum *x, typeRNum *u,
             typeRNum *pSys)
{
  out[0] = 0.0;  /* df1du1 */
  out[1] = 1.0;  /* df2du1 */
}

void sysjacxadj(typeRNum *out, typeRNum t, typeRNum *x,
                typeRNum *adj, typeRNum *u, typeRNum *pSys)
{}

void sysjacuadj(typeRNum *out, typeRNum t, typeRNum *x,
                typeRNum *adj, typeRNum *u, typeRNum *pSys)
{}

void icostfct(typeRNum *out, typeRNum t, typeRNum *x, typeRNum *u,
              typeRNum *xdes, typeRNum *udes, typeRNum *pCost)
{
  out[0] = x[1]*x[1] + pCost[0]*u[0]*u[0];  /* L =  x2^2 + r*u^2 */
}

void icostjacx(typeRNum *out, typeRNum t, typeRNum *x, typeRNum *u,
               typeRNum *xdes, typeRNum *udes, typeRNum *pCost)
{
  out[0] = 0.0;     /* dLdx1 */
  out[1] = 2*x[1];  /* dLdx2 */
}

void icostjacu(typeRNum *out, typeRNum t, typeRNum *x, typeRNum *u,
               typeRNum *xdes, typeRNum *udes, typeRNum *pCost)
{
  out[0] = 2*pCost[0]*u[0];  /* dLdu1 */
}
```

```
void fcostfct(typeRNum *out, typeRNum t, typeRNum *x,
              typeRNum *xdes, typeRNum *pCost)
{
  out[0] = x[0]*x[0] + x[1]*x[1];  /* V = x1^2 + x2^2 */
}

void fcostjacx(typeRNum *out, typeRNum t, typeRNum *x,
               typeRNum *xdes, typeRNum *pCost)
{
  out[0] = 2*x[0];  /* dVdx1 */
  out[1] = 2*x[1];  /* dVdx2 */
}
```

The single functions have the following meaning:

- `sysdim`: This function contains the system dimension of the considered problem, i.e. the number of states as well as of the controls.
- `sysfct`: In this function the corresponding system dynamics is included where the variable `pSys` contains additional system parameters (see Section 4.3).
- `sysjacx` and `sysjacu`: The jacobians $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial u}$ allow for evaluating the optimality conditions (3.5) time efficiently. The jacobians must be implemented in vector form by arranging the successive rows of the jacobians.
- `sysjacxadj` and `sysjacuadj`: An alternative to `sysjacx` and `sysjacu` is to provide the product functions $(\frac{\partial f}{\partial x})^\mathsf{T}\lambda$ and $(\frac{\partial f}{\partial u})^\mathsf{T}\lambda$ that appear in the partial derivatives H_x and H_u of the Hamiltonian. For the double integrator example, these functions would take the form

  ```
  void sysjacxadj(typeRNum *out, typeRNum t, typeRNum *x,
                  typeRNum *adj, typeRNum *u, typeRNum *pSys)
  {
    out[0] = 0.0;              /* df1dx1*adj1 + df2dx1*adj2 */
    out[1] = pSys[0]*adj[0];   /* df1dx2*adj1 + df2dx2*adj2 */
  }

  void sysjacuadj(typeRNum *out, typeRNum t, typeRNum *x,
                  typeRNum *adj, typeRNum *u, typeRNum *pSys)
  {
    out[0] = adj[1];  /* df1du1*adj1 + df2dx2*adj2 */
  }
  ```

  instead of the empty function templates in the code above. Using `sysjacxadj` and `sysjacuadj` in `GRAMPC` is of advantage, if the jacobians $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial u}$ are sparsely populated and an explicit evaluation of $(\frac{\partial f}{\partial x})^\mathsf{T}\lambda$ and $(\frac{\partial f}{\partial u})^\mathsf{T}\lambda$ in the gradient algorithm would involve unnecessary zero multiplications.

  Whether the functions `sysjacx`, `sysjacu` or `dfdxadjfct`, `dfduadjfct` are provided by the user is indicated to `GRAMPC` via the algorithmic options (see Section 4.2).
- `fcostfct` and `icostfct`: The final and integral cost terms in (3.1a) can be provided by means of these functions. The variable `pCost` contains additional parameters.

- `fcostjacx`, `icostjacx` and `icostjacu`: To further accelerate the computation, the partial derivatives $\frac{\partial l}{\partial x}$, $\frac{\partial l}{\partial u}$, $\frac{\partial V}{\partial x}$ have to be provided by the user.

A template file `probfctTEMPLATE.c` for the problem formulation can be found in `<grampc_root>/examples`. This directory also contains a makefile for compiling purposes.

4.2 Algorithmic options

The algorithmic options of `GRAMPC` concern the numerical integrations in the gradient algorithm (Table 3.1), the line search implementation (Sections 3.3 and 3.4) as well as further settings. Table 4.1 summarizes all algorithmic options of `GRAMPC` in terms of the option name and type as well as default and possible values (if available).

Table 4.1: ALGORITHMIC OPTIONS.

option name	**type**	**option value**	
		possible values	**default value**
`MaxIter`	integer		2
`ShiftControl`	char	`on/off`	`on`
`ScaleProblem`	char	`on/off`	`off`
`CostIntegrator`	char	`trapezodial/simpson`	`trapezodial`
`Integrator`	char	`euler/modeuler/heun/ruku45`	`heun`
`IntegratorRelTol`	double		10^{-6}
`IntegratorAbsTol`	double		10^{-8}
`LineSearchType`	char	`adaptive/explicit1/explicit2`	`adaptive`
`LineSearchMax`	double		0.75
`LineSearchMin`	double		10^{-5}
`LineSearchInit`	double		$5.5 \cdot 10^{-4}$
`LineSearchIntervalFactor`	double		0.85
`LineSearchAdaptFactor`	double		3/2
`LineSearchIntervalTol`	double		10^{-1}
`JacobianX`	char	`sysjacx/sysjacxadj`	`sysjacxadj`
`JacobianU`	char	`sysjacu/sysjacuadj`	`sysjacuadj`
`IntegralCost`	char	`on/off`	`on`
`FinalCost`	char	`on/off`	`on`

- `MaxIter`: Denotes the number of gradient iterations N per MPC step.
- `ShiftControl`: Indicates if the initial input trajectory $u_{k+1}^{(0)}(\tau)$ is shifted in each MPC step, i.e. $u_{k+1}^{(0)}(\tau) = u_k^{(N)}(\tau + \Delta t)$ for $\tau \in [0, T - \Delta t]$, while the last time segment for $\tau \in (T - \Delta t, T]$ is extrapolated according to $u_{k+1}^{(0)}(\tau) = u_k^{(N)}(\tau)$, $\tau \in (T - \Delta t, T]$. The control shift is motivated by the principle of optimality for an infinite MPC horizon and will lead to a faster convergence behavior of the gradient algorithm for most MPC problems.
- `ScaleProblem`: Indicates if scaling is activated in `GRAMPC`. Scaling is recommended for improving the numerical conditioning when the states and controls values of the given MPC problem (3.1)

differ in several orders of magnitude. GRAMPC allows state and control scaling according to

$$\bar{x}(t) = \frac{x(t) - x_{\text{offset}}}{x_{\text{scale}}}, \quad \bar{u}(t) = \frac{u(t) - u_{\text{offset}}}{u_{\text{scale}}} \tag{4.2}$$

where $x_{\text{offset}} \in \mathbb{R}^n$ and $u_{\text{offset}} \in \mathbb{R}^m$ denote offset values and $x_{\text{scale}} \in \mathbb{R}^n$ and $u_{\text{scale}} \in \mathbb{R}^m$ are scaling values. These values are set via the algorithmic parameters of GRAMPC (see Section 4.3).

- CostIntegrator: Denotes the integration method for the computation of the integral cost $L(x(t), u(t))$ in (3.1a). The integration is either performed by means of the trapezodial rule (option value: trapezodial) or the Simpson rule (option value: simpson).
- Integrator: Specifies which integration scheme is used for the forward and backward integration of the system and adjoint dynamics (see algorithm in Table 3.1). So far, the following integration methods are implemented: Euler method (Option value: euler), modified Euler method (Option value: modeuler), Heun method (Option value: heun) and a Runge-Kutta method of 4th order (option value: ruku45). The first mentioned integrators use a fixed step size, whereas the Runge-Kutta method is a variable step size integrator.
- IntegratorRelTol: Denotes the relative tolerance of the Runge-Kutta integrator with variable step size.
- IntegratorAbsTol: Denotes the absolute tolerance of the Runge-Kutta integrator with variable step size.
- LineSearchType: Indicates the computation approach of the step size α as described in Sections 3.3 and 3.4, respectively. The step size is computed either by using the adaptive line search (3.9), (3.10) (option value: adaptive) or the explicit approaches (option value: explicit1/ explicit2), see (3.14) and (3.16).
- LineSearchMax: Denotes the maximum value $\alpha_{\max}$ of the step size α.
- LineSearchMin: Denotes the minimum value $\alpha_{\min}$ of the step size α.
- LineSearchInit: Indicates the initial value $\alpha_{\text{init}} > 0$ for the step size α. If the adaptive line search from Section 3.3 is used, the sample point α_2 is set to $\alpha_2 = \alpha_{\text{init}}$.
- LineSearchIntervalFactor: This option is only active for the adaptive line search (LineSearchType option value: adaptive). Using the initialized mid sample point $\alpha_2 = \alpha_{\text{init}}$, the interval factor $\beta \in (0, 1)$ specifies the interval bounds $[\alpha_1, \alpha_3]$ for the adaptive line search according to $\alpha_1 = \alpha_2(1 - \beta)$ and $\alpha_3 = \alpha_2(1 + \beta)$.
- LineSearchAdaptFactor: This option is only available for the adaptive line method (LineSearchType option value: adaptive) and specifies the factor $\kappa > 1$ for the interval adaptation (3.10) that determines how much the line search interval is scaled and shifted in the direction of the minimum step size value.
- LineSearchIntervalTol: This option is only available for the adaptive line search (LineSearchType option value: adaptive) and specifies the interval tolerance $\varepsilon_\alpha \in (0, 1)$ for which the interval adaptation (3.10) is performed.
- JacobianX: Specifies how the jacobian of the system function f w.r.t. the states x is provided by the user. The jacobian is either provided in matrix form (option value: sysjacx), i.e. sysjacx $\triangleq \frac{\partial f}{\partial x}$ or in vector form as the product with the corresponding adjoint states λ (option value: sysjacxadj), i.e. sysjacxadj $\triangleq \frac{\partial f}{\partial x}^{\mathsf{T}} \lambda$, see Section 4.1.

- `JacobianU`: Specifies how the jacobian of the system function f w.r.t. the controls u is provided by the user. The jacobian is either provided in matrix form (option value: `sysjacu`), i.e. sysjacu $\triangleq \frac{\partial f}{\partial u}$ or in vector form as the product with the corresponding adjoint states λ (option value: `sysjacuadj`), i.e. sysjacuadj $\triangleq \frac{\partial f}{\partial u}^{\mathsf{T}} \lambda$, see Section 4.1.
- `IntegralCost`: Indicates if the integral cost in the MPC formulation (3.1a) is given.
- `FinalCost`: Indicates if the final cost in the MPC formulation (3.1a) is given.

4.3 Algorithmic parameters

Besides the algorithmic options, `GRAMPC` provides several parameter settings, where some are problem specific and need to be provided, whereas other values are set to default values. Table 4.2 provides an overview of the algorithmic parameters of `GRAMPC`.

Table 4.2: ALGORITHMIC PARAMETERS.

parameter name	parameter type	default value
`u0`	double *	to be provided
`xk`	double *	to be provided
`xdes`	double *	to be provided
`udes`	double *	to be provided
`Thor`	double	to be provided
`dt`	double	to be provided
`tk`	double	0
`Nhor`	integer	30
`umax`	double *	$+\inf \cdot \mathrm{e}_{\mathrm{Nu}}$
`umin`	double *	$-\inf \cdot \mathrm{e}_{\mathrm{Nu}}$
`xScale`	double *	e_{Nx}
`xOffset`	double *	$0 \cdot \mathrm{e}_{\mathrm{Nx}}$
`uScale`	double *	e_{Nu}
`uOffset`	double *	$0 \cdot \mathrm{e}_{\mathrm{Nu}}$
`pCost`	double *	`NULL`
`NpCost`	integer	0
`pSys`	double *	`NULL`
`NpSys`	integer	0

The symbol $\mathrm{e}_n = [1, \ldots, 1]^{\mathsf{T}} \in \mathbb{R}^n$ denotes a column vector. The asterisk indicates pointer notation, `NULL` represents the null pointer. All parameters that must be provided by the user are listed at the beginning of Table 4.2, whereas the remaining parameters are optional and initialized to the coresponding default values.

- `u0`: Denotes the initial (constant) control vector u_0 that is used by `GRAMPC` to initialize the control trajectory in the first MPC step, i.e. $u_0^{(0)}(\tau) = u_0 = const.$ for $\tau \in [0, T]$.
- `xk`: Denotes the initial state vector $x(0) = x_k$ (initial condition) at the corresponding sampling time t_k for the MPC formulation (3.1).
- `xdes`: Denotes the desired setpoint vector for the states.

- udes: Denotes the desired setpoint vector for the controls.
- Thor: Denotes the prediction horizon for the optimization.
- dt: Denotes the sampling time of the considered system.
- tk: Denotes the current sampling instance t_k that is provided to the time-varying system dynamics (3.1b) of the MPC formulation (3.1).
- Nhor: Denotes the number of discretization points for the numerical integration.
- umax: Denotes a vector with upper bounds for the control inputs.
- umin: Denotes a vector with lower bounds for the control inputs.
- xScale: Denotes a vector with the scaling factors for each state, see (4.2).
- xOffset: Denotes a vector with the offset for each state, see (4.2).
- uScale: Denotes a vector with the scaling factors for each control, see (4.2).
- uOffset: Denotes a vector with the offset for each control, see (4.2).
- pCost: Denotes a vector with additional parameters with regard to the cost function (3.1a).
- NpCost: Denotes the number of elements of the cost parameter vector pCost.
- pSys: Denotes a vector with additional parameters for the system dynamics (3.1b).
- NpSys: Denotes the number of elements of the system parameter vector pSys.

4.4 Using GRAMPC in C

In order to provide a high level of portability of GRAMPC in view of different operating systems and hardware devices, the main components are written in plain C without the use of external libraries. This section describes the usage of GRAMPC in C including initialization, compiling and running the MPC framework.

4.4.1 Main components of GRAMPC

As illustrated in Figure 4.2, GRAMPC contains initializing as well as running files (including the gradient algorithm), different integrators for the system and adjoint dynamics (cf. Table 3.1) and functions to alter the options and parameters (cf. Section 4.2 and 4.3).

In more detail, GRAMPC comprises the following main files (see also Appendix B for the function interface)

- grampc_init.c: Function for initializing GRAMPC that also includes memory allocation as well as deallocation.
- grampc_run.c: Function for running GRAMPC including the implemented gradient algorithm with the two different line search strategies (cf. Chapter 3).
- grampc_mess.c: Function for printing information (e.g. errors) regarding the initialization as well as execution of GRAMPC.
- grampc_setopt.c: Function for setting algorithmic options (cf. Section 4.2).

- grampc_setparam.c: Function for setting algorithmic parameters (cf. Section 4.3).
- euler1.c: Simple Euler integration scheme with fixed step size.
- eulermod2.c: Modified Euler integration scheme with fixed step size.
- heun2.c: Heun integration scheme with fixed step size.
- ruku45.c: Runge-Kutta integration scheme of order 4 with variable step size.

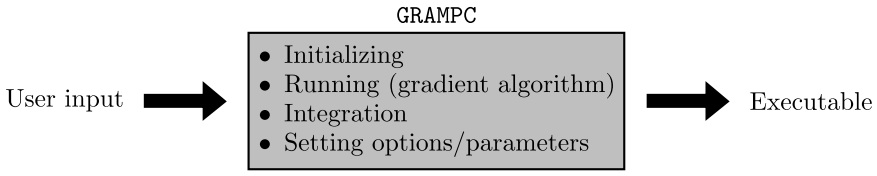

Figure 4.2: Main components of GRAMPC.

4.4.2 Initialization of GRAMPC

The overall initialization of GRAMPC is done via the routine

```
void grampc_init(typeGRAMPC **grampc)
```

where the overall structure variable grampc is initialized with the following substructures (see Appendix A for the definition of the data type):

- sol (data type typeGRAMPCsol): Contains the computed control input, the predicted state x_{k+1} in the next MPC step as well as the corresponding cost value $J(u_k^{(N)}, x_k)$.
- param (data type typeGRAMPCparam): Contains the parameter structure of GRAMPC.
- rws (data type typeGRAMPCrws): Contains the real time workspace of GRAMPC including MPC calculations along the prediction horizon.
- opt (data type typeGRAMPCopt): Contains the option structure of GRAMPC.

The definition of these data types can be found in Appendix A. The deallocation of the variables can be achieved by means of the function

```
void grampc_free(typeGRAMPC **grampc)
```

The single options in Table 4.1 are set via the three functions

```
void grampc_setopt_real(typeGRAMPC *grampc,
                        typeChar optName[], typeRNum optValue)

void grampc_setopt_int(typeGRAMPC *grampc,
                       typeChar optName[], typeInt optValue)

void grampc_setopt_string(typeGRAMPC *grampc,
                          typeChar optName[], typeChar optValue[])
```

for option values of type double, integer and string, respectively. An overview of the current options can be displayed by using

```
void grampc_printopt(typeGRAMPC *grampc)
```

Example (Setting options in C)
The relative tolerance of the Runge-Kutta integrator, the number of gradient iterations and the new integrator can be set in the following way

```
[...]
/* Declaration */
typeGRAMPC *grampc;
[...]

/* Initialization */
grampc_init(&grampc);
[...]

/* Setting options */
grampc_setopt_string(grampc, "Integrator", "ruku45");
grampc_setopt_real(grampc, "IntegratorRelTol", 0.001);
grampc_setopt_int(grampc, "MaxIter", 5);
[...]
```

Similar to setting the GRAMPC options, the parameters in Table 4.2 are set according to their data type with the three functions

```
void grampc_setparam_real(typeGRAMPC *grampc,
                          typeChar paramName[], typeRNum paramValue)

void grampc_setparam_int(typeGRAMPC *grampc,
                         typeChar paramName[], typeInt paramValue)

void grampc_setparam_vector(typeGRAMPC *grampc,
                            typeChar paramName[],
                            typeRNum *paramValue)
```

An overview of the parameters can be displayed by using

```
void grampc_printparam(typeGRAMPC *grampc)
```

Example (Setting parameters in C)
The initial conditions, the prediction horizon and the number of discretization points for a system with two states and one control input can be set in the following way:

```
[...]
/* Declaration */
typeGRAMPC *grampc;
[...]

/* Initialization */
grampc_init(&grampc);
[...]

/* Initial states & control */
typeRNum x0[2] = {2.0,-1.0};
```

```
typeRNum u0[1] = {0.0};
[...]

/* Setting parameters */
grampc_setparam_real(grampc, "dt", 0.01);
grampc_setparam_real(grampc, "Thor", 1.5);
grampc_setparam_int(grampc, "Nhor", 40);
grampc_setparam_vector(grampc, "xk", x0);
grampc_setparam_vector(grampc, "u0", u0);
[...]
```

4.4.3 Compiling and calling GRAMPC

GRAMPC can be integrated into an executable program after the problem formulation as well as options and parameters are provided. A makefile for compilation purposes is provided in the folder <grampc_root>/examples. The main calling routine for GRAMPC is

```
void grampc_run(typeGRAMPC *grampc)
```

The following code demonstrates how to integrate GRAMPC within an MPC loop by completing the double integrator example from Section 4.1.

Example (C code for running GRAMPC within an MPC loop)

```
#include "grampc.h"   /* contains all necessary header files */
int main()
{
  /* Declaration */
  typeGRAMPC *grampc;
  typeRNum x0[2] = {2.0,-1.0}; typeRNum xdes[2]  = {0.0,0.0};
  typeRNum u0[1] = {0.0};      typeRNum udes[1]  = {0.0};
  typeInt NpSys  = 1;          typeRNum pSys[1]  = {2.0};
  typeInt NpCost = 1;          typeRNum pCost[1] = {5.0};
  typeInt iMPC, iMPCmax;
  typeRNum tSim = 2.0;
  typeRNum tMPC = 0.0;

  /* Initialization and memory allocation */
  grampc_init(&grampc);

  /* Setting options */
  grampc_setopt_string(grampc, "Integrator", "ruku45");
  grampc_setopt_real(grampc, "IntegratorRelTol", 0.001);
  grampc_setopt_int(grampc, "MaxIter", 5);
  grampc_printopt(grampc);

  /* Setting mandatory parameters */
  grampc_setparam_vector(grampc, "u0", u0);
  grampc_setparam_vector(grampc, "xk", x0);
  grampc_setparam_vector(grampc, "udes", udes);
  grampc_setparam_vector(grampc, "xdes", xdes);
  grampc_setparam_real(grampc, "Thor", 1.5);
```

```
  grampc_setparam_real(grampc, "dt", 0.01);

  /* Setting optional parameters */
  grampc_setparam_real(grampc, "tk", tMPC);
  grampc_setparam_int(grampc, "Nhor", 40);
  grampc_setparam_int(grampc, "NpCost", NpCost);
  grampc_setparam_int(grampc, "NpSys", NpSys);
  grampc_setparam_vector(grampc, "pCost", pCost);
  grampc_setparam_vector(grampc, "pSys", pSys);
  grampc_printparam(grampc);

  /* MPC loop */
  iMPCmax = (typeInt)(tSim/MPC->dt);
  for (iMPC = 1; iMPC <=i MPCmax+1; iMPC++) {
    grampc_run(grampc);
    /* next control: grampc->sol->unext */
    /* next state:   grampc->sol->xnext */
    /* cost value:   grampc->sol->J     */
    tMPC = tMPC + grampc->param->dt;
    grampc_setparam_vector(grampc, "xk", grampc->sol->xnext);
    grampc_setparam_real(grampc, "tk", tMPC);
  }

  /* Memory deallocation */
  grampc_free(&grampc);
  return 0;
}
```

After initializing the structure variable `grampc` some options are set. Then the mandatory as well as some optional parameters are chosen. Finally, `GRAMPC` is repetitively executed until a defined simulation time is reached where also the current state of the system and the sampling time are provided.

4.5 Using GRAMPC in Matlab/Simulink

The main components of `GRAMPC` are implemented in plain C to ensure a high level of portability. However, `GRAMPC` also provides a user-friendly interaction with MATLAB/SIMULINK with an additional graphical user interface (GUI) to allow for a convenient MPC design.

4.5.1 Interface to Matlab

Each main component of `GRAMPC` (cf. Section 4.4.1) has a related MEX routine (included in the directory `<grampc_root>/matlab/src`) as it is illustrated in Figure 4.3 that allows running `GRAMPC` in MATLAB/SIMULINK as well as altering options and parameters without recompilation. A related makefile to compile a given problem formulation for use in MATLAB/SIMULINK is provided in `<grampc_root>/matlab/examples`.

The MATLAB implemented version of the MPC example in Section 4.4.3 is given in the following lines.

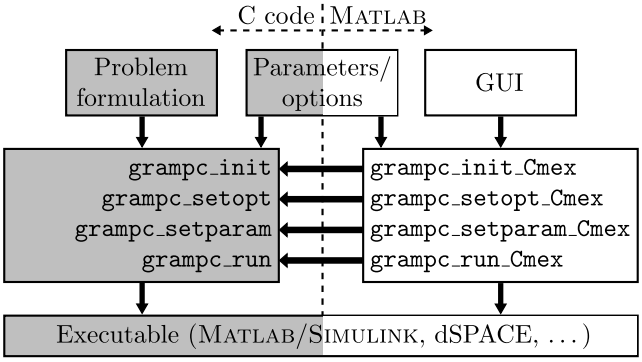

Figure 4.3: Interface of GRAMPC to MATLAB/SIMULINK (gray – C code, white – MATLAB code)

Example (Matlab code for running GRAMPC within an MPC loop)

```
function []=startMPC()

  x0     = [2.0,-1.0]; xdes  = [0.0,0.0];
  u0     = 0.0;        udes  = 0.0;
  NpSys  = 1;          pSys  = 2.0;
  NpCost = 1;          pCost = 5.0;
  tSim   = 2.0;
  tMPC   = 0.0;

  % Initialization
  grampc = grampc_init_Cmex();

  % Setting options
  grampc_setopt_Cmex(grampc, 'Integrator', 'ruku45');
  grampc_setopt_Cmex(grampc, 'IntegratorRelTol', 0.001);
  grampc_setopt_Cmex(grampc, 'MaxIter', 5);

  % Setting mandatory parameters
  grampc_setparam_Cmex(grampc, 'u0',u0);
  grampc_setparam_Cmex(grampc, 'xk', x0);
  grampc_setparam_Cmex(grampc, 'udes',udes);
  grampc_setparam_Cmex(grampc, 'xdes',xdes);
  grampc_setparam_Cmex(grampc, 'Thor', 1.5);
  grampc_setparam_Cmex(grampc, 'dt', 0.01);

  % Setting optional parameters
  grampc_setparam_Cmex(grampc, 'tk', tMPC);
  grampc_setparam_Cmex(grampc, 'Nhor', 40);
  grampc_setparam_Cmex(grampc, 'NpCost', NpCost);
  grampc_setparam_Cmex(grampc, 'NpSys', NpSys);
  grampc_setparam_Cmex(grampc, 'pCost', pCost);
  grampc_setparam_Cmex(grampc, 'pSys', pSys);

  % MPC loop
```

```
  iMPCmax=tSim/MPC.dt;
  for iMPC=1:iMPCmax+1
    [xnext,unext,J] = grampc_run_Cmex(grampc);
    tMPC            = tMPC + grampc.param.dt;
    grampc_setparam_Cmex(grampc, 'xk', xnext);
    grampc_setparam_Cmex(grampc, 'tk', tMPC);
  end

% --- END FUNCTION ---
```

As it was already discussed in Section (4.4.3) (related C example), the structure variable `grampc` is initialized before the options and mandatory as well as optional parameters are set. Then `GRAMPC` is started within an MPC loop where after the computation of the new controls, the predicted states and the corresponding cost function the current state of the system (new initial condition) and the sampling time are provided to `GRAMPC`.

4.5.2 Interface to Matlab/Simulink

`GRAMPC` also allows a MATLAB/SIMULINK integration via the S-function `grampc_run_Sfct.c` (also included in the directory `<grampc_root>/matlab/src`). A corresponding SIMULINK block can be found in the folder `<grampc_root>/matlab/simulink` for a number of MATLAB versions. The directory also contains the m-file `initData.m` which can be used for initializing `GRAMPC`'s options and parameters. Each of the MATLAB examples in the directory `<grampc_root>/matlab/examples` also includes a corresponding SIMULINK model for simulation and the build procedure of the MEX routines additionally compiles the S-function for the SIMULINK block.

The MATLAB/SIMULINK model of `GRAMPC` can be seen in Figure 4.4. The illustrated block `MPC` contains algorithmic components of `GRAMPC` (implemented within the S-function `grampc_run_Sfct.c`). It requires an input argument of data type `typeGRAMPC` comprising all user settings regarding parameters as well as options. Additionally, Figure 4.4 reveals that the MATLAB/SIMULINK block of `GRAMPC` has four input and three output signals, respectively. The input signals are the current state of the system (`xk`), the current sampling instance (`tk`) as well as the desired setpoints for the states (`xdes`) and the controls (`udes`). The output signals are the control for the next sampling time (`unext`), the predicted state (`xnext`) and the cost value (`Cost`). The S-function is also built during the compilation process, which can be performed as described in the previous sections.

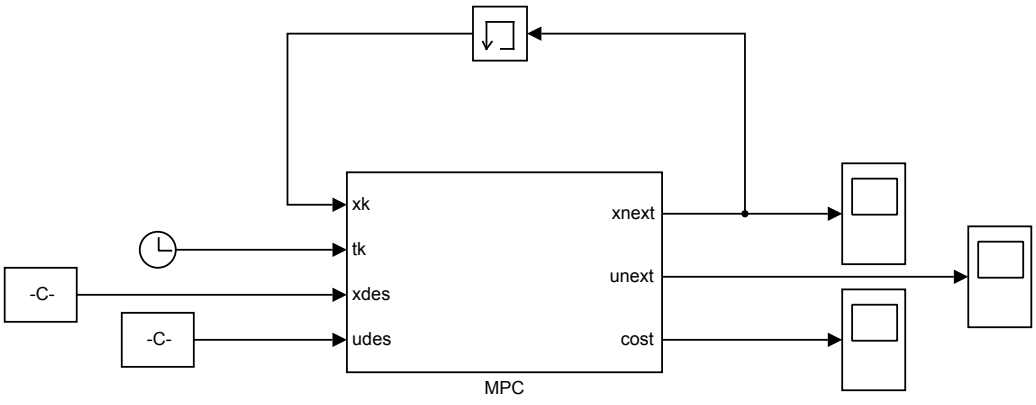

Figure 4.4: MATLAB/SIMULINK model of `GRAMPC`.

4.5.3 Graphical user interface (GUI)

In addition to the MATLAB/SIMULINK interface functions, a graphical user interface (GUI) for GRAMPC can be used to allow for a convenient way to parametrize the model predictive controller. The GUI directory <grampc_root>/matlab/GUI contains the MATLAB figure grampc_GUI.fig (containing the GUI layout) and the m-file grampc_GUI.m to be called to start the GUI, e.g. by typing

```
>> run('<grampc_root>/matlab/GUI/grampc_GUI.m')
```

in the command window. A snapshot of the GUI window is shown in Figure 4.5.

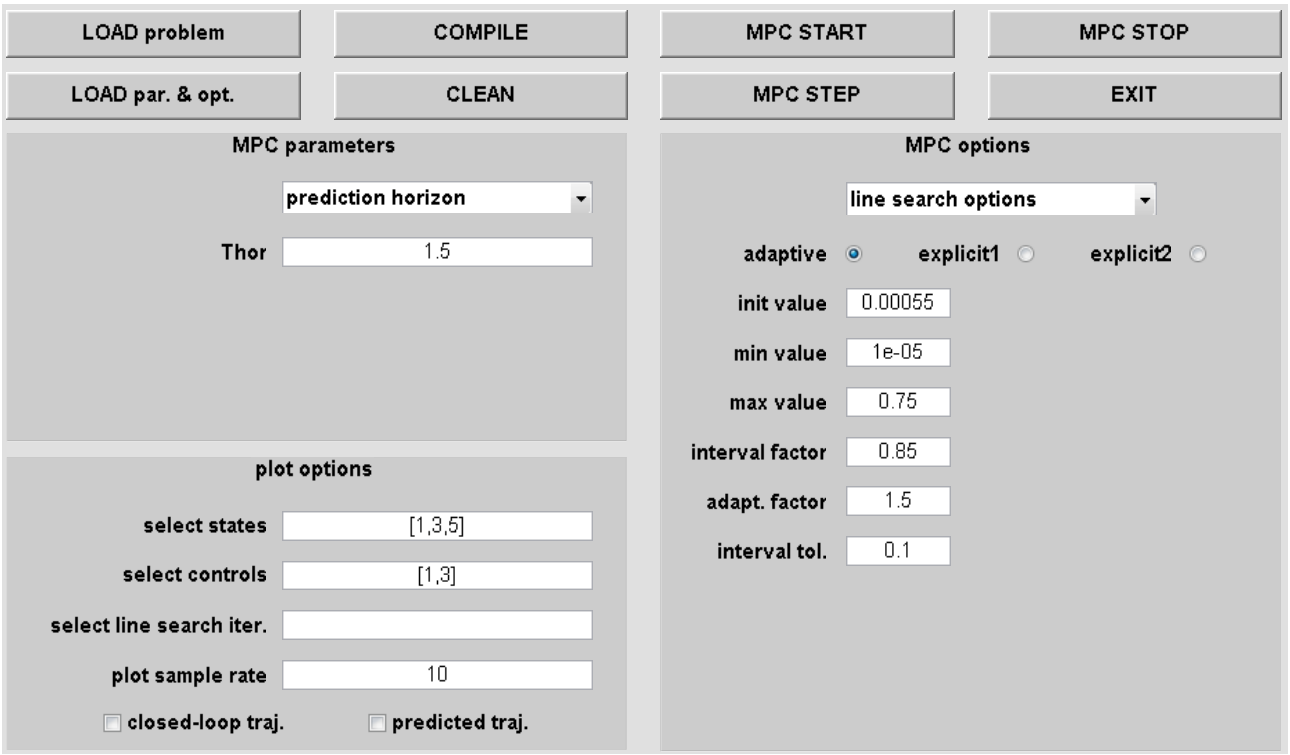

Figure 4.5: GRAMPC graphical user interface.

The GUI buttons have the following meaning:

- The button **LOAD problem** loads the problem formulation including the system representation and the cost functions (cf. Section 4.1). .
- The button **LOAD par. & opt.** reads the same initialization file initData.m as described in Section 4.5.2, where the options and parameters for the considered problem are set. A template file for this function can be found in <grampc_root>/matlab/simulink.
- The remaining buttons compile the problem implementation, clean previous built files, start and stop the MPC simulation and perform single (paused) MPC steps. The **EXIT** button closes the GUI.

The GUI is further divided into three blocks (ss Figure 4.5) with regard to GRAMPC's parameter and option settings as well as plot properties:

- The block **MPC parameters** contains all algorithmic parameters such as the prediction horizon (Thor), the sampling time (dt), the number of discretization points (Nhor) and the remaining parameters as described in Section 4.3.

- The options of GRAMPC are set in the **MPC options** block. Available choices for the selected option are depicted.
- The block **plot options** provides options to illustrate the simulation results. The actual closed-loop results of the MPC (**closed-loop traj.**) and the MPC predictions in each MPC step (**predicted traj.**) can be individually depicted.

  The fields **select states** and **select controls** allow to select individual components of the states and controls to be plotted (to be entered in MATLAB vector syntax).

  The field **select line search iter.** provides the additional option to select and plot the approximated cost value for the corresponding step sizes and gradient iteration (cf. Section 3.3). The results are illustrated within the plot of the predicted trajectories and hence can be displayed by activating the corresponding plot options.

  The field **plot sample rate** can be used to accelerate the MPC simulation, as it specifies how many MPC steps are computed before the plots are refreshed. For the default value 10, the MPC results are depicted for every 10-th sampling instance.

5 Examples

In this chapter the use of `GRAMPC` for C and MATLAB is illustrated by means of several examples from various physical domains and with varying complexity. The examples can be found in the directories `<grampc_root>/examples` (for use in C) and `<grampc_root>/matlab/examples` (for use in MATLAB) and the presented results were performed with MATLAB 2012b (64-bit) and Windows 7 (64-bit) on an Intel Core i5 CPU with 2.67 GHz and 4 GB memory

5.1 2D crane

Figure 5.1 shows the schematics of the crane example from [8]. The nonlinear system dynamics is

$$\begin{aligned}\ddot{s}_1 &= a_C\\ \ddot{s}_2 &= a_R\\ \ddot{\phi} &= -\frac{1}{s_2}\left(g\sin(\phi) + a_C\cos(\phi) + 2\dot{s}_2\dot{\phi}\right)\end{aligned}$$

with the gravitational constant g. The states $x \in \mathbb{R}^6$ are the cart position $x_1 = s_1$, the rope length $x_3 = s_2$, the angular deflection $x_5 = \phi$ and the corresponding velocities $x_2 = \dot{s}_1$, $x_4 = \dot{s}_2$ and $x_6 = \dot{\phi}$. The cart acceleration $u_1 = a_C$ as well as the rope acceleration $u_2 = a_R$ serve as controls $u \in \mathbb{R}^2$. The integral and final cost in (3.1a) are chosen according to

$$V(x) = \Delta x^\mathsf{T}\Delta x, \quad L(x,u) = \Delta x^\mathsf{T}\Delta x + 0.01\Delta u^\mathsf{T}\Delta u$$

with $\Delta x := x - x_{\mathrm{des}}$ and $\Delta u := u - u_{\mathrm{des}}$ denoting the distance to the desired setpoint $x_{\mathrm{des}} \in \mathbb{R}^6$, $u_{\mathrm{des}} \in \mathbb{R}^2$. The matrices are set to

$$P = Q = \mathrm{diag}(1,1,1,1,1,1), \quad R = \mathrm{diag}(0.01, 0.01)\,.$$

The initial states and controls and the desired setpoints follow to

$$\begin{aligned} x_0 &= [-2\,\mathrm{m}, 0, 2\,\mathrm{m}, 0, 0, 0]^\mathsf{T}, \qquad u_0 = [0,0]^\mathsf{T}\\ x_{\mathrm{des}} &= [2\,\mathrm{m}, 0, 0.4\,\mathrm{m}, 0, 0, 0]^\mathsf{T}, \quad u_{\mathrm{des}} = [0,0]^\mathsf{T}\,.\end{aligned}$$

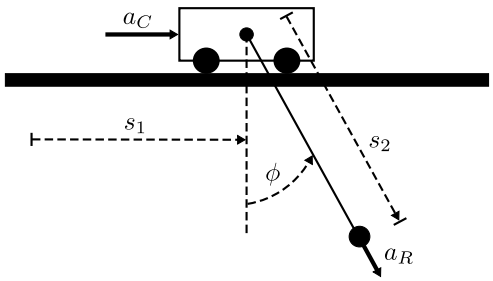

Figure 5.1: Schematics of the 2D crane.

The prediction horizon, the sample time, the number of discretization points as well as gradient iterations and the boundaries for the control inputs are

$$T = 1.5\,\mathrm{s}, \quad N_{\mathrm{hor}} = 40, \quad u_{\max} = \left[+2\,\mathrm{m/s^2}, +2\,\mathrm{m/s^2}\right]^\mathsf{T}$$
$$\Delta t = 1\,\mathrm{ms}, \qquad N = 2, \qquad u_{\min} = \left[-2\,\mathrm{m/s^2}, -2\,\mathrm{m/s^2}\right]^\mathsf{T}.$$

Furthermore, the Euler forward integrator and the explicit line search strategy with (3.16) are used. The entries of P, Q and R are comprised in the variable `pCost`. The initialization is then given in the following way (demonstrated for use in C here)

```
[...]
// declaration
typeGRAMPC *grampc;
typeRNum x0[6]={-2.0,0.0,2.0,0.0,0.0,0.0};typeRNum u0[2]={0.0,0.0};
typeRNum xdes[6]={2.0,0.0,0.4,0.0,0.0,0.0};typeRNum udes[2]={0.0,0.0};
typeRNum umax[2]={2.0,2.0}; typeRNum umin[2]={-2.0,-2.0};
typeRNum pCost[14]={1.0,1.0,1.0,1.0,1.0,1.0,    // P
                    1.0,1.0,1.0,1.0,1.0,1.0,    // Q
                    0.01,0.01};                 // R
typeInt NpCost=14;
[...]
// initialization
grampc_init(&grampc);
// options
grampc_setopt_string(grampc,"Integrator","euler");
grampc_setopt_string(grampc,"LineSearchType","explicit2");
grampc_printopt(grampc);
// parameters
grampc_setparam_real(grampc,"Thor",Thor);
grampc_setparam_real(grampc,"dt",dt);
grampc_setparam_int(grampc,"Nhor",Nhor);
grampc_setparam_vector(grampc,"umax",umax);
grampc_setparam_vector(grampc,"umin",umin);
grampc_setparam_int(grampc,"NpCost",NpCost);
grampc_setparam_vector(grampc,"pCost",pCost);
grampc_setparam_vector(grampc,"xk",x0);
grampc_setparam_vector(grampc,"u0",u0);
grampc_setparam_vector(grampc,"xdes",xdes);
grampc_setparam_vector(grampc,"udes",udes);
grampc_printparam(grampc);
[...]
```

The problem formulation in the corresponding directory `<grampc_root>/examples/Crane_2D` can be compiled by means of the provided makefile. Running `GRAMPC` then displays the following results

```
$ ./startMPC
-- MPC OPTIONS --
                MaxIter: 2
           ShiftControl: on
           ScaleProblem: off
```

```
          CostIntegrator: trapezodial
              Integrator: euler
        IntegratorRelTol: 1.00e-06
        IntegratorAbsTol: 1.00e-08
          LineSearchType: explicit2
           LineSearchMax: 0.750000
           LineSearchMin: 0.000010
          LineSearchInit: 0.000550
LineSearchIntervalFactor: 0.850000
   LineSearchAdaptFactor: 1.500000
   LineSearchIntervalTol: 0.100000
              JacobianX: sysjacxadj
              JacobianU: sysjacuadj
           IntegralCost: on
              FinalCost: on
-- MPC PARAMETER --
     Nx: 6
     Nu: 2
     xk: [-2.00,0.00,2.00,0.00,0.00,0.00]
     u0: [0.00,0.00]
   xdes: [2.00,0.00,0.40,0.00,0.00,0.00]
   udes: [0.00,0.00]
   Thor: 1.50
     dt: 0.0010
     tk: 0.0000
   Nhor: 40
  pCost: [1.00,1.00,1.00,1.00,1.00,1.00,
          1.00,1.00,1.00,1.00,1.00,0.01,0.01]
 NpCost: 14
   pSys: []
  NpSys: 0
   umax: [2.00,2.00]
   umin: [-2.00,-2.00]
 xScale: [1.00,1.00,1.00,1.00,1.00,1.00]
xOffset: [0.00,0.00,0.00,0.00,0.00,0.00]
 uScale: [1.00,1.00]
uOffset: [0.00,0.00]
MPC running ...
MPC finished. Average computation time: 0.087 ms.
```

The related closed-loop trajectories for the state, the controls as well as the cost value can be seen in Figure 5.2.

5.2 3D crane

A more involved model of a 3D rotary crane presented in [4] is considered as another example. The schematics of the crane are illustrated in Figure 5.3. The related system dynamics are given by

$$\ddot{s}_1 = u_1, \quad \ddot{s}_2 = u_2, \quad \ddot{\phi}_1 = u_3,$$

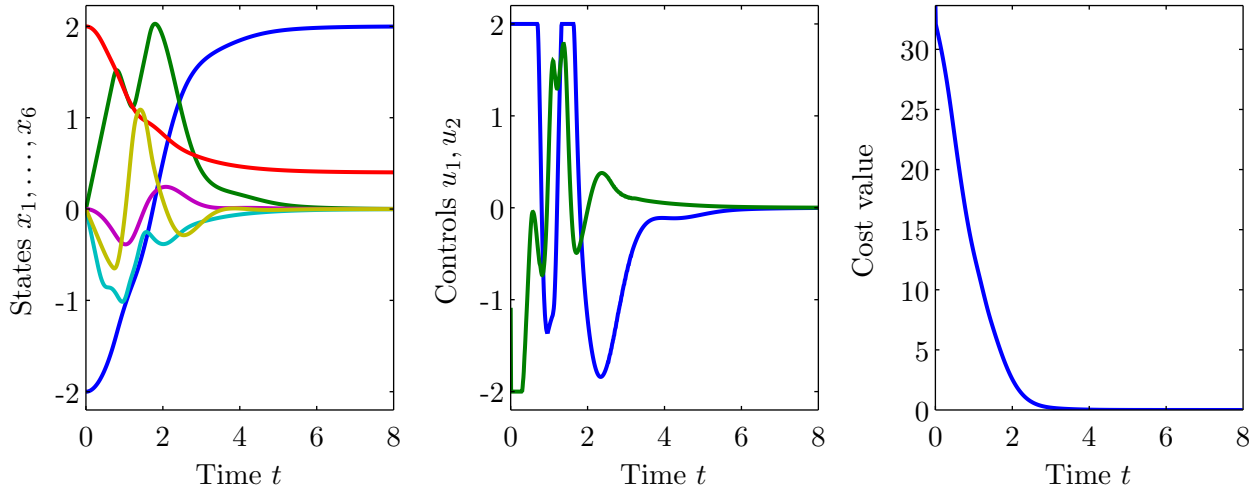

Figure 5.2: Simulation results for the 2D crane.

$$
\begin{aligned}
\ddot{\phi}_2 =& \frac{1}{s_2 \cos\phi_3}\Big(-2\dot{s}_2\dot{\phi}_1 \cos\phi_2 \sin\phi_3 - 2\dot{s}_2\dot{\phi}_2 \cos\phi_3 - 2\dot{\phi}_1\dot{\phi}_3 s_2 \cos\phi_2 \cos\phi_3 + 2s_2\dot{\phi}_2\dot{\phi}_3 \sin\phi_3 \\
& - s_1\dot{\phi}_1^2 \cos\phi_2 + s_2\dot{\phi}_1^2 \sin\phi_2 \cos\phi_2 \cos\phi_3 - g \sin\phi_2 + u_1 \cos\phi_2 - u_3 s_2 \cos\phi_2 \sin\phi_3 \Big), \\
\ddot{\phi}_3 =& \frac{1}{s_2}\Big(-2\dot{s}_1\dot{\phi}_1 \cos\phi_3 - 2\dot{s}_2\dot{\phi}_3 + 2\dot{s}_2\dot{\phi}_1 \sin\phi_2 + 2s_2\dot{\phi}_1\dot{\phi}_2 \cos\phi_2 \cos^2\phi_3 + s_1\dot{\phi}_1^2 \sin\phi_2 \sin\phi_3 \\
& - s_2\dot{\phi}_2^2 \sin\phi_3 \cos\phi_3 + s_2\dot{\phi}_1^2 \cos^2\phi_2 \sin\phi_3 \cos\phi_3 - g \cos\phi_2 \sin\phi_3 - u_1 \sin\phi_2 \sin\phi_3 \\
& + (s_2 \sin\phi_2 - s_1 \cos\phi_3)\, u_3 \Big) .
\end{aligned}
$$

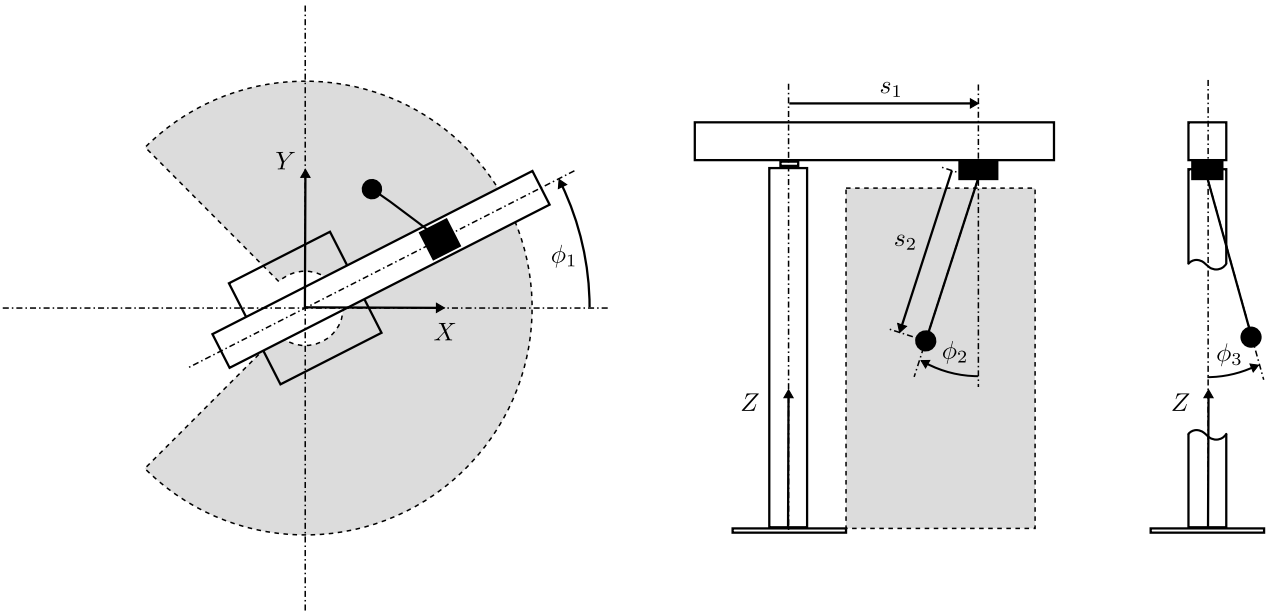

Figure 5.3: Schematics of the 3D crane [4].

The states $x = [s_1, s_2, \phi_1, \phi_2, \phi_3, \dot{s}_1, \dot{s}_2, \dot{\phi}_1, \dot{\phi}_2, \dot{\phi}_3]^\mathsf{T} \in \mathbb{R}^{10}$ are the cart position, the rope length, the deflection angles and the corresponding velocities. The controls $u = [\ddot{s}_1, \ddot{s}_2, \ddot{\phi}_1]^\mathsf{T} \in \mathbb{R}^3$ are the cart and the rope acceleration as well as the angular acceleration of ϕ_1. The final and the integral cost are

$$V(x) = \Delta x^\mathsf{T} P \Delta x, \quad L(x,u) = \Delta x^\mathsf{T} Q \Delta x + \Delta u^\mathsf{T} R \Delta u$$

where $P \in \mathbb{R}^{10\times10}$, $Q \in \mathbb{R}^{10\times10}$ and $R \in \mathbb{R}^{3\times3}$ are positive-definite and diagonal matrices and $\Delta x := x - x_{\text{des}}$ and $\Delta u := u - u_{\text{des}}$ denote the distance to the desired setpoint $x_{\text{des}} \in \mathbb{R}^{10}$, $u_{\text{des}} \in \mathbb{R}^3$ (cf. Section 5.1). The matrices are chosen to

$$P = Q = \text{diag}(1,1,1,1,1,0.1,0.1,0.1,0.1,1), \quad R = \text{diag}(0.01, 0.01, 0.01),$$

where the values are comprised in the related variable `pCost`. The initial states and controls as well as the desired setpoints for a considered crane maneuver are

$$\begin{aligned} x_0 &= [0.7\,\text{m}, 0.7\,\text{m}, -60\,\text{deg}, 0, 0, 0, 0, 0, 0, 0]^\mathsf{T}, \quad u_0 = [0,0,0]^\mathsf{T}, \\ x_{\text{des}} &= [0.2\,\text{m}, 0.25\,\text{m}, 60\,\text{deg}, 0, 0, 0, 0, 0, 0, 0]^\mathsf{T}, \quad u_{\text{des}} = [0,0,0]^\mathsf{T}. \end{aligned}$$

The prediction horizon, the sample time, the number of discretization points as well as the gradient iterations and the boundaries for the control inputs are set to

$$\begin{aligned} T = 1.5\,\text{s}, \quad N_{\text{hor}} = 40, \quad u_{\max} &= \left[+2\,\text{m/s}^2, +2\,\text{m/s}^2, +2\,\text{rad/s}^2\right]^\mathsf{T} \\ \Delta t = 2\,\text{ms}, \quad N = 2, \quad u_{\min} &= \left[-2\,\text{m/s}^2, -2\,\text{m/s}^2, -2\,\text{rad/s}^2\right]^\mathsf{T}. \end{aligned}$$

The initialization and execution of `GRAMPC` is now demonstrated for use in MATLAB. To this end, the provided m-file `startMPC.m` in the example directory `<grampc_root>/matlab/examples/Crane_3D` for running `GRAMPC` contains an initialization structure according to

```
[...]
% options
grampc=grampc_init_Cmex();
grampc_setopt_Cmex(grampc,'LineSearchType','adaptive');
% User parameter
% mandatory paramters
user.param.x0=[0.7,0.7,-1.0472,0.0,0.0,0.0,0.0,0.0,0.0,0.0];
user.param.u0=[0.0,0.0,0.0];
user.param.xdes=[0.2,0.25,1.0472,0.0,0.0,0.0,0.0,0.0,0.0,0.0];
user.param.udes=[0.0,0.0,0.0];
% optional parameters
user.param.umax=[2.0,2.0,2.0];
user.param.umin=[-2.0,-2.0,-2.0];
user.param.Thor=1.5;
user.param.Nhor=40;
user.param.dt=0.002;
user.param.NpCost=23;
user.param.pCost=[1.0,1.0,1.0,1.0,1.0,0.1,0.1,0.1,0.1,1.0,... // Q
                  0.01,0.01,0.01,...                          // R
                  1.0,1.0,1.0,1.0,1.0,0.1,0.1,0.1,0.1,1.0];   // P
grampc_setparam_Cmex(grampc,'xk',user.param.x0);
grampc_setparam_Cmex(grampc,'u0',user.param.u0);
```

```
grampc_setparam_Cmex(grampc,'xdes',user.param.xdes);
grampc_setparam_Cmex(grampc,'udes',user.param.udes);
grampc_setparam_Cmex(grampc,'umax',user.param.umax);
grampc_setparam_Cmex(grampc,'umin',user.param.umin);
grampc_setparam_Cmex(grampc,'Thor',user.param.Thor);
grampc_setparam_Cmex(grampc,'Nhor',user.param.Nhor);
grampc_setparam_Cmex(grampc,'dt',user.param.dt);
grampc_setparam_Cmex(grampc,'NpCost',user.param.NpCost);
grampc_setparam_Cmex(grampc,'pCost',user.param.pCost);
[...]
```

A compilation section is also included within the m-file featuring the following lines of code

```
[...]
% Compilation
PROBFCT = 'probfct_CRANE_3D.c';
COMPILE = 1;
DEBUG   = 0;
VERBOSE = 0;
CLEAN   = 0;

if COMPILE || CLEAN
  if exist('make','file') ~= 2
    curPath = cd;
    cd('..');
    addpath(cd);
    cd(curPath);
  end
end

if CLEAN
  make clean;
  vec    = [];
  grampc = [];
  return;
end

if COMPILE
  if VERBOSE && DEBUG
    make(PROBFCT,'verbose','debug');
  elseif VERBOSE
    make(PROBFCT,'verbose');
  elseif DEBUG
    make(PROBFCT,'debug');
  else
    make(PROBFCT);
  end
end
[...]
```

PROBFCT indicates the name of the problem function containing the considered problem formulation. The corresponding flags COMPILE, DEBUG, VERBOSE control the compilation procedure with regard to the additional use of the debug as well as verbose mode whereas CLEAN cleans up previous compiled files. The make file is included in the related directory <grampc_root>/matlab/examples of the 3D crane example. After compiling successfully the problem formulation and running GRAMPC the results illustrated in Figure 5.4 can be achieved.

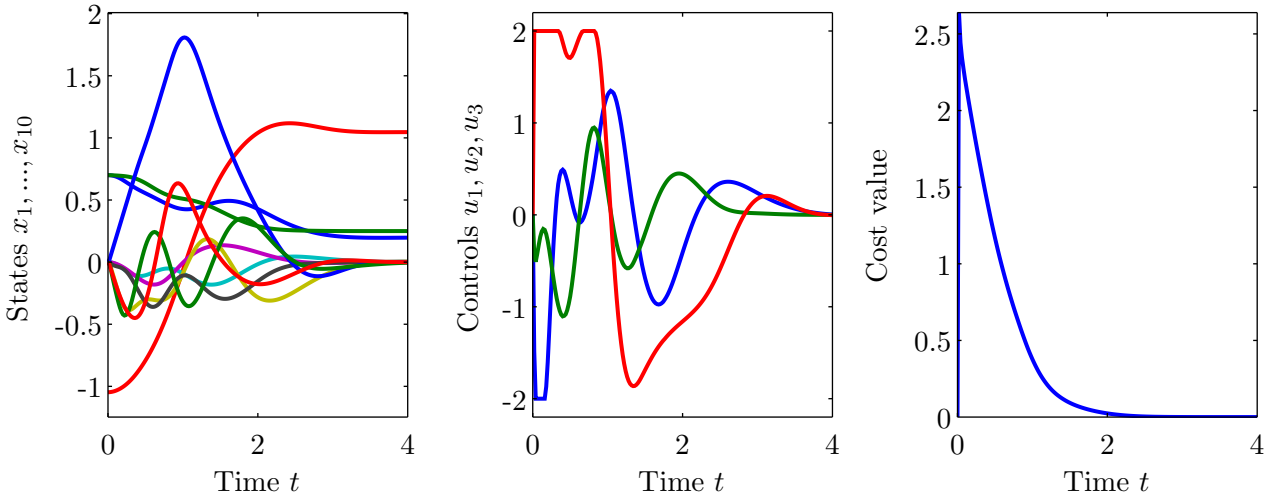

Figure 5.4: Simulation results for the 3D crane.

5.3 Quadrotor

Figure 5.5 shows the setup of a quadrotor that is considered in this section to be controlled by means of GRAMPC. The system dynamics follow to [7]

$$\begin{aligned}
\ddot{X} &= a\left(\cos\gamma\sin\beta\cos\alpha + \sin\gamma\sin\alpha\right), & \ddot{\gamma} &= \frac{1}{\cos\beta}\left(\omega_X\cos\gamma + \omega_Y\sin\gamma\right)\\
\ddot{Y} &= a\left(\cos\gamma\sin\beta\sin\alpha - \sin\gamma\cos\alpha\right), & \ddot{\beta} &= -\omega_X\sin\gamma + \omega_Y\cos\gamma\\
\ddot{Z} &= a\cos\gamma\cos\beta - g, & \ddot{\alpha} &= \omega_X\cos\gamma\tan\beta + \omega_Y\sin\gamma\tan\beta + \omega_Z
\end{aligned}$$

where (X, Y, Z) denote the translational position of the quadrotor to the inertial coordinate system O and (γ, β, α) describe the orientation of the vehicle with regard to the body-fixed coordinate system V. The states are $x = [X, \dot{X}, Y, \dot{Y}, Z, \dot{Z}, \gamma, \beta, \alpha]^\mathsf{T} \in \mathbb{R}^9$ containing the translational position, the

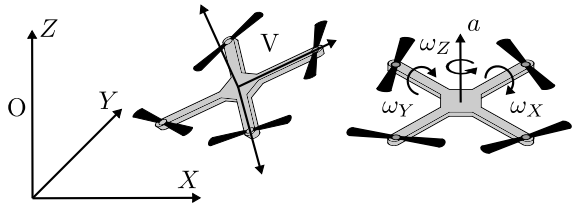

Figure 5.5: Setup of the quadrotor.

correspoding velocity and the orientation of the quadrotor. The controls $u = [a, \omega_X, \omega_Y, \omega_Z]^\mathsf{T} \in \mathbb{R}^4$ are the mass-normalized thrust a an the angular rates $(\omega_X, \omega_Y, \omega_Z)$. The final and integral cost functions are again considered to be quadratic, i.e.

$$V(x) = \Delta x^\mathsf{T} \Delta x, \quad L(x,u) = \Delta x^\mathsf{T} \Delta x + \Delta u^\mathsf{T} \Delta u.$$

The initial states and controls and the desired setpoints are

$$\begin{aligned} x_0 &= [-3\,\mathrm{m}, 0, -3\,\mathrm{m}, 0, -3\,\mathrm{m}, 0, 0, 0, 0]^\mathsf{T}, & u_0 &= [9.81\,\mathrm{m/s^2}, 0, 0, 0]^\mathsf{T} \\ x_\mathrm{des} &= [0, 0, 0, 0, 0, 0, 0, 0, 0]^\mathsf{T}, & u_\mathrm{des} &= [9.81\,\mathrm{m/s^2}, 0, 0, 0]^\mathsf{T}. \end{aligned}$$

The prediction horizon, the sampling time, the number of discretization points as well as gradient iterations and the boundaries for the control inputs are

$$\begin{aligned} T &= 1.5\,\mathrm{s}, \quad N_\mathrm{hor} = 40, \quad u_\mathrm{max} = \left[+11\,\mathrm{m/s^2}, +1\,\mathrm{rad/s}, +1\,\mathrm{rad/s}, +1\,\mathrm{rad/s}\right]^\mathsf{T} \\ \Delta t &= 2\,\mathrm{ms}, \quad N = 2, \quad u_\mathrm{min} = [0, -1\,\mathrm{rad/s}, -1\,\mathrm{rad/s}, -1\,\mathrm{rad/s}]^\mathsf{T} \end{aligned}$$

The files for compiling and running `GRAMPC` can be found in `<grampc_root>/examples/Quadrotor` for use in C and in `<grampc_root>/matlab/examples/Quadrotor` for use in MATLAB, respectively. The simulation results are shown in Figure 5.6.

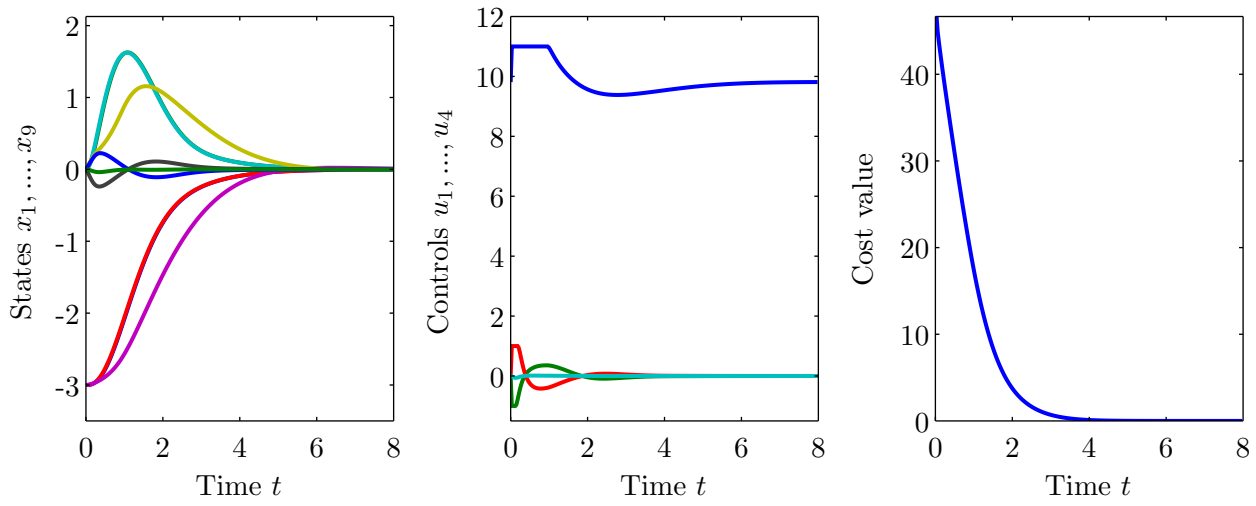

Figure 5.6: Simulation results for the quadrotor.

5.4 Quasi-linear diffusion-convection-reaction system

An example for a large-scale system is the quasi-linear diffusion-convection-reaction system presented in [13]. It is described by the partial differential equation (PDE)

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

with the boundary and initial conditions

$$\begin{aligned} \partial_z \theta(z,t)|_{z=0} &= 0 \\ \partial_z \theta(z,t)|_{z=1} + \theta(1,t) &= u(t), \quad t > 0 \\ \theta(z,0) &= \theta_0(z), \quad z \in [0,1]. \end{aligned}$$

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 (3.1b) can be derived where the number of states corresponds to the number of discretizations. To this end, a large-scale formulation depending on the number of discretization points can be realized. Note that the states, the control, the time and all parameters are normalized which lead to non-dimensional system dynamics.

The final as well as the integral cost are chosen quadratically, i.e.

$$V(x) = 0.5\Delta x^\mathsf{T}\Delta x, \quad L(x,u) = 0.5\Delta x^\mathsf{T}\Delta x + 0.005\Delta u^2.$$

In order to determine the initial values (x_0, u_0) and the desired setpoints $(x_{\text{des}}, u_{\text{des}})$, the related stationary ODE

$$0 = \partial_z\left[(q_0 + q_1\theta(z,\tau))\partial_z\theta(z,\tau) - \nu\theta(z,\tau)\right] + (r_0 + r_1\theta(z,\tau))\theta(z,\tau), \quad z \in (0,1)$$

with the corresponding boundary conditions for $\tau = \{0, T_{\text{SP}}\}$

$$\begin{aligned} \theta(0,0) &= 1, & \partial_z\theta(0,0) &= 0 \\ \theta(0,T_{\text{SP}}) &= 2, & \partial_z\theta(0,T_{\text{SP}}) &= 0 \end{aligned}$$

has to be solved. T_{SP} indicates the transition time for the setpoint change $(x_0, u_0) \rightarrow (x_{\text{des}}, u_{\text{des}})$. The prediction horizon, the sampling time, the number of gradient iterations and the boundaries for the control inputs are

$$\begin{aligned} T &= 0.2, & & & u_{\max} &= +3 \\ \Delta t &= 0.005, & N &= 2, & u_{\min} &= -3 \end{aligned}$$

Due to stiffness of the system dynamics, the Runge-Kutta integrator with variable step size used in connection with the explicit line search strategy.

The implementation for `GRAMPC` can be found in `<grampc_root>/examples/Reactor_PDE` (C) and `<grampc_root>/matlab/examples/Reactor_PDE` (MATLAB). The achieved results are illustrated in Figure 5.7.

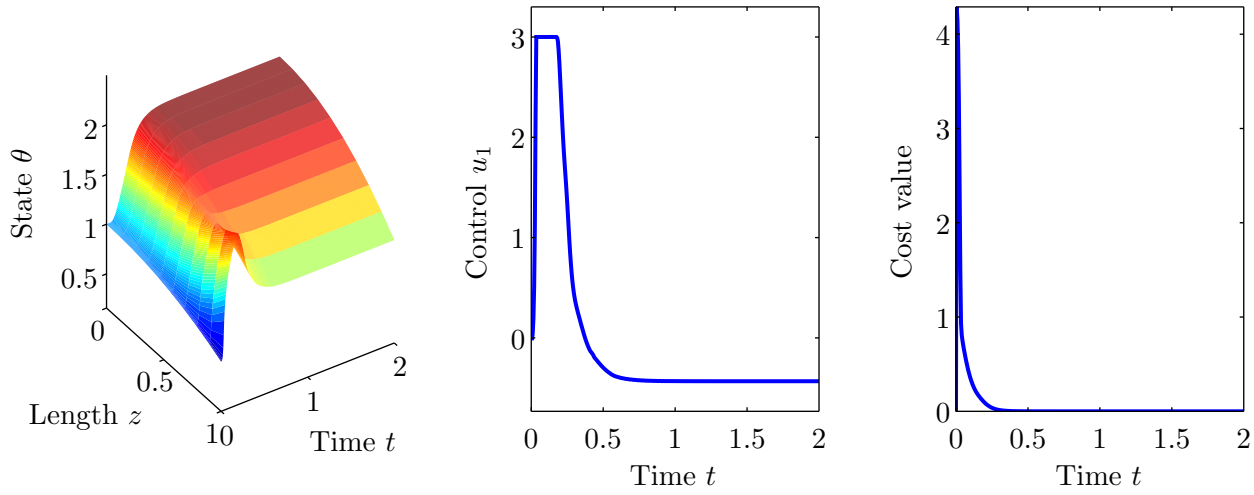

Figure 5.7: Simulation results for the diffusion-convection-reaction system.

Table 5.1: Parameters of the CSTR example.

Paramter	Unit	Value	Paramter	Unit	Value
E_1	-	9758.3	E_2	-	8560.0
κ_1	h^{-1}	30.828	κ_2	h^{-1}	86.688
κ_3	K/kJ	0.1	κ_4	$\mathrm{m}^3\mathrm{K/kJ}$	$3.522 \cdot 10^{-4}$
T_{in}	°C	104.9	c_{in}	$\mathrm{mol/m}^3$	$5.1 \cdot 10^3$
k_{10}	h^{-1}	$1.287 \cdot 10^{12}$	k_{20}	$\mathrm{m}^3/(\mathrm{mol\,h})$	$9.043 \cdot 10^6$
ΔH_{AB}	kJ/mol	4.2	ΔH_{BC}	kJ/mol	-11.0
ΔH_{AD}	kJ/mol	-41.85			

5.5 Continuous stirred tank reactor

In this section the continuous stirred tank reactor (CSTR) from [11] is discussed. The schematics of the CSTR is given in Figure 5.8 and the corresponding nonlinear dynamics follow to

$$\begin{aligned}
\dot{c}_A &= -k_1(T)c_A - k_2(T)c_A^2 + (c_{\mathrm{in}} - c_A)\,u_1 \\
\dot{c}_B &= k_1(T)c_A - k_1(T)c_B - c_B u_1 \\
\dot{T} &= -\delta\left(k_1(T)c_A\Delta H_{AB} + k_1(T)c_B\Delta H_{BC} + k_2(T)c_A^2\Delta H_{AD}\right)\alpha\,(T_c - T) + (T_{\mathrm{in}} - T)\,u_1 \\
\dot{T}_c &= \beta\,(T - Tc) + \gamma u_2
\end{aligned}$$

where c_A and c_B denote the concentrations of the reactant A and the product B and T and T_c are the temperatures in the reactor and in the cooling jacket, respectively. The functions $k_1(T)$ and $k_2(T)$ are

$$k_i(T) = k_{i,0}\exp\left(\frac{-E_i}{T/^\circ\mathrm{C} + 273.15}\right), \quad i = 1, 2.$$

Moreover, the controls $u = [q, \dot{Q}]^\mathsf{T} \in \mathbb{R}^2$ are the normalized flow rate through the reactor and the cooling power applied to the cooling jacket. The concentrations and the temperatures are comprised in the state vector $x = [c_A, c_B, T, T_c]^\mathsf{T} \in \mathbb{R}^4$. The model parameters are given in Table 5.1. The weighting matrices for the quadratic final and integral cost

$$V(x) = \Delta x^\mathsf{T} P \Delta x, \quad L(x, u) = \Delta x^\mathsf{T} Q \Delta x + \Delta u^\mathsf{T} R \Delta u$$

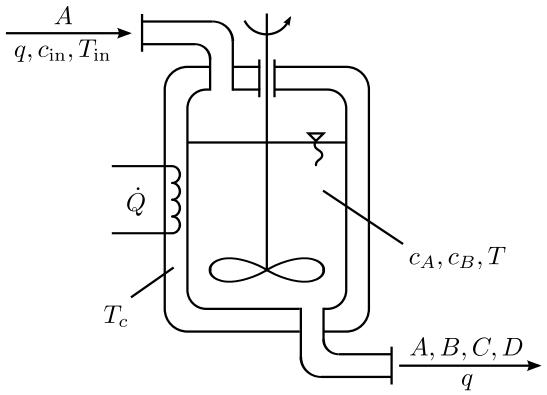

Figure 5.8: Setup of the CSTR example.

are chosen to

$$P = Q = \mathrm{diag}\left(0.2, 1.0, 0.5, 0.2\right), \quad R = \mathrm{diag}\left(0.5, 5 \cdot 10^{-3}\right).$$

The initial values and desired setpoints are given by

$$\begin{aligned} x_0 &= [2.02\,\mathrm{kmol/m^3}, 1.07\,\mathrm{kmol/m^3}, 100\,^\circ\mathrm{C}, 97.1\,^\circ\mathrm{C}]^\mathsf{T}, & u_0 &= [5\,\mathrm{h^{-1}}, -2540\,\mathrm{kJ\,h^{-1}}]^\mathsf{T} \\ x_{\mathrm{des}} &= [1.37\,\mathrm{kmol/m^3}, 0.95\,\mathrm{kmol/m^3}, 110\,^\circ\mathrm{C}, 108.6\,^\circ\mathrm{C}], & u_{\mathrm{des}} &= [5\,\mathrm{h^{-1}}, -1190\,\mathrm{kJ\,h^{-1}}]^\mathsf{T}. \end{aligned}$$

Further MPC parameters and options are chosen to

$$\begin{aligned} T &= 1200/3600\,\mathrm{h}, & N_{\mathrm{hor}} &= 40, & u_{\max} &= \left[35\,\mathrm{h^{-1}}, 0\right]^\mathsf{T} \\ \Delta t &= 1.0/3600\,\mathrm{h}, & N &= 2, & u_{\min} &= \left[3\,\mathrm{h^{-1}}, -9000\,\mathrm{kJ\,h^{-1}}\right]^\mathsf{T}. \end{aligned}$$

Moreover, the states and controls are scaled by means of the following values

$$\begin{aligned} x_{\mathrm{scale}} &= 5 \cdot [10^2\,\mathrm{mol/m^3}, 10^2\,\mathrm{mol/m^3}, 10^1\,^\circ\mathrm{C}, 10^1\,^\circ\mathrm{C}]^\mathsf{T}, & u_{\mathrm{scale}} &= [16\,\mathrm{h^{-1}}, 4500\,\mathrm{kJ\,h^{-1}}]^\mathsf{T} \\ x_{\mathrm{offset}} &= 5 \cdot [10^2\,\mathrm{mol/m^3}, 10^2\,\mathrm{mol/m^3}, 10^1\,^\circ\mathrm{C}, 10^1\,^\circ\mathrm{C}]^\mathsf{T}, & u_{\mathrm{offset}} &= [19\,\mathrm{h^{-1}}, -4500\,\mathrm{kJ\,h^{-1}}]^\mathsf{T}. \end{aligned}$$

In addition, the explicit line search is used in combination with the initial value $\alpha_{\mathrm{init}} = 10^{-6}$ and the Euler forward integrator. The files for compiling and running `GRAMPC` can be found in `<grampc_root>/examples/Reactor_CSTR` (C) and `<grampc_root>/matlab/examples/Reactor_CSTR` (MATLAB), respectively. The achieved results are for the CSTR example are presented in Figure 5.9.

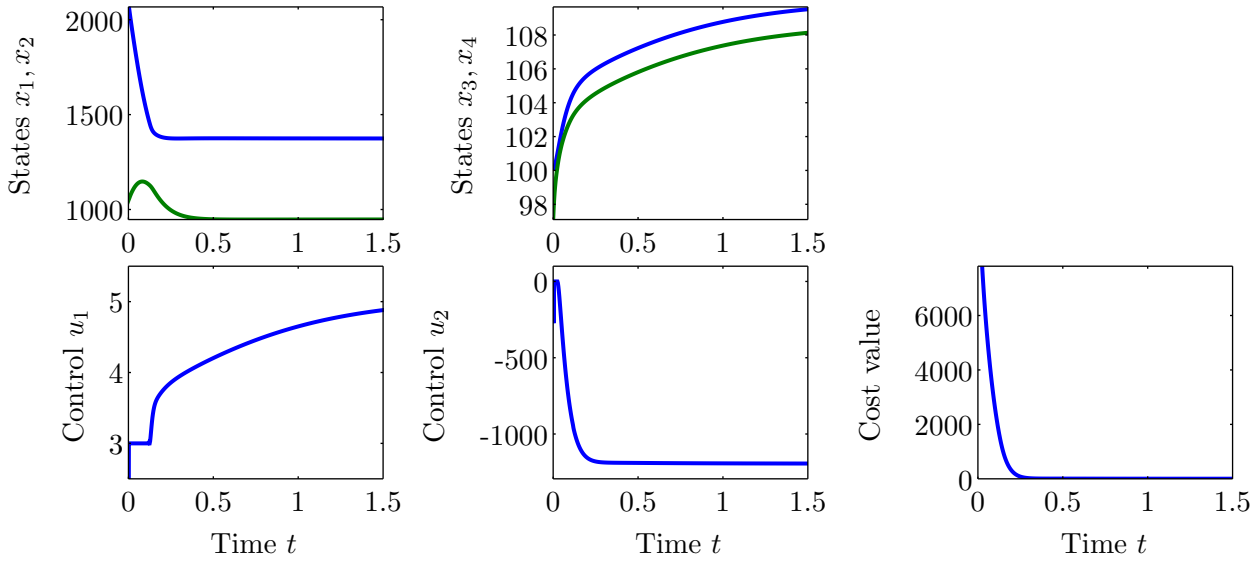

Figure 5.9: Simulation results for the CSTR.

5.6 Coupled mass-spring-damper system

The last considered example is the coupled mass-spring-damper system that is shown in Figure 5.10. The r equal masses m are connected by spring and damper elements with the same parameters c and d. The states $x = [y_1, \ldots, y_r, \dot{y}_1, \ldots, \dot{y}_r]^\mathsf{T} \in \mathbb{R}^{2r}$ are the positions of the masses as well as the

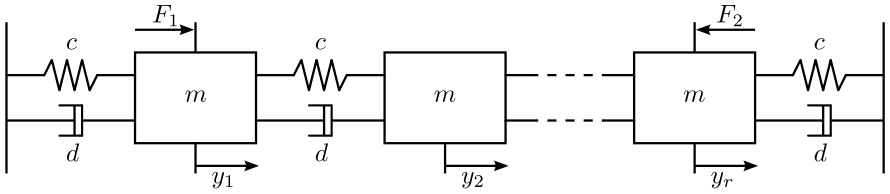

Figure 5.10: Setup of the mass-spring-damper system with a variable number of elements.

related velocities. The forces F_1 and F_2 acting at the first and the last element serve as control inputs $u = [F_1, F_2]^\mathsf{T} \in \mathbb{R}^2$. The linear system dynamics are given in the following way

$$\begin{aligned}
\ddot{y}_1 &= \frac{c}{m}(-2y_1 + y_2) + \frac{d}{m}(-2\dot{y}_1 + \dot{y}_2) + \frac{1}{m}F_1 \\
\ddot{y}_2 &= \frac{c}{m}(y_1 - 2y_2 + y_3) + \frac{d}{m}(\dot{y}_1 - 2\dot{y}_2 + \dot{y}_3) \\
\ddot{y}_{r-1} &= \frac{c}{m}(y_{r-2} - 2y_{r-1} + y_r) + \frac{d}{m}(\dot{y}_{r-2} - 2\dot{y}_{r-1} + \dot{y}_r) \\
\ddot{y}_r &= \frac{c}{m}(y_{r-1} - 2y_r) + \frac{d}{m}(\dot{y}_{r-1} - 2\dot{y}_r) - \frac{1}{m}F_2
\end{aligned}$$

with the parameters $m = 1\,\mathrm{kg}$, $c = 1\,\mathrm{kg/s^2}$ and $d = 0.2\,\mathrm{kg/s}$. For the simulation, $r = 5$ mass elements are considered resulting in $n = 10$ states. The final and integral cost are

$$V(x(t_k + T)) = x^\mathsf{T}x, \quad L(x, u) = x^\mathsf{T}x + 0.01u^\mathsf{T}u.$$

The initial values and desired setpoints follow to

$$\begin{aligned}
x_0 &= [1\,\mathrm{m}, 0, 0, 0, 1\,\mathrm{m}, 0, 0, 0, 0, 0]^\mathsf{T}, & u_0 &= [0, 0]^\mathsf{T} \\
x_{\mathrm{des}} &= [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^\mathsf{T}, & u_{\mathrm{des}} &= [0, 0]^\mathsf{T}.
\end{aligned}$$

The prediction horizon, the sampling time, the number of discretization points as well as gradient iterations and the input constraints are set to

$$\begin{aligned}
T &= 10\,\mathrm{s}, \quad N_{\mathrm{hor}} = 30, \quad u_{\max} = \left[1\,\mathrm{m/s^2}, 1\,\mathrm{m/s^2}\right]^\mathsf{T} \\
\Delta t &= 5\,\mathrm{ms}, \quad N = 2, \quad u_{\min} = \left[-1\,\mathrm{m/s^2}, -1\,\mathrm{m/s^2}\right]^\mathsf{T}.
\end{aligned}$$

The files to compile the problem formulation and to run `GRAMPC` can be found in the directories `<grampc_root>/examples/Mass_Spring_Damper` (C) and `<grampc_root>/matlab/examples/Mass_Spring_Damper` (MATLAB). The achieved simulation results are given in Figure 5.11.

5.7 Vertical take-off and landing aircraft

The last considered example is the vertical take-off and landing aircraft (VTOL) as it is presented in [12] and illustrated in Figure 5.12. The dynamics is

$$\ddot{x}_c = -\sin\theta u_1 + \varepsilon\cos\theta u_2, \quad \ddot{y}_c = \cos\theta u_1 + \varepsilon\sin\theta u_2 - 1, \quad \ddot{\theta} = u_2$$

where the states $x = [x_c, \dot{x}_c, y, \dot{y}_c, \theta, \dot{\theta}]^\mathsf{T}$ are the normalized position of the center of mass, the roll angle and the corresponding velocities. Moreover, the controls $u = [u_1, u_2]^\mathsf{T}$ are the normalized thrust

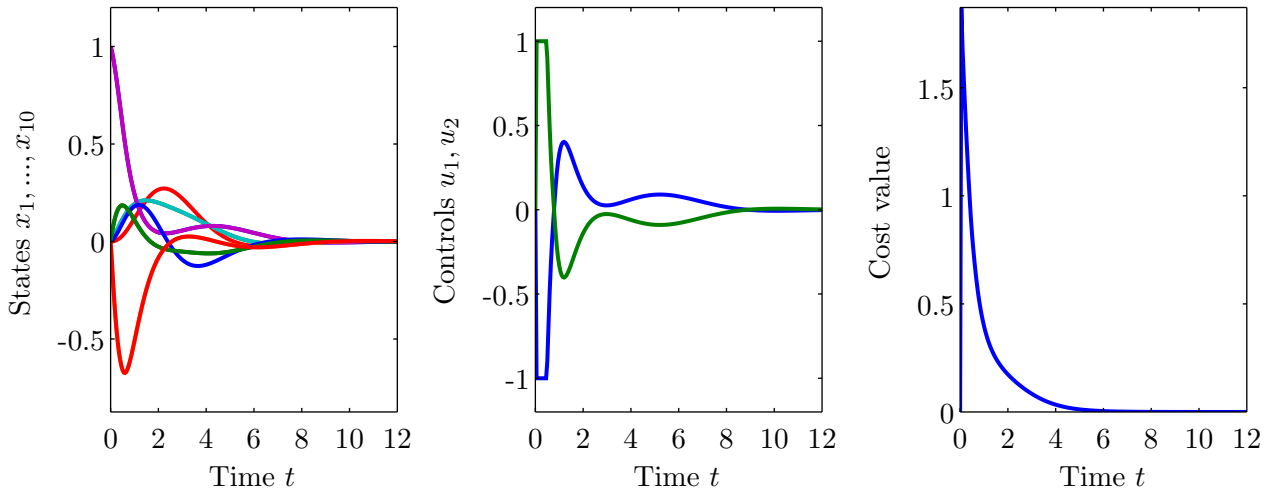

Figure 5.11: Simulation results for the mass-spring-damper system.

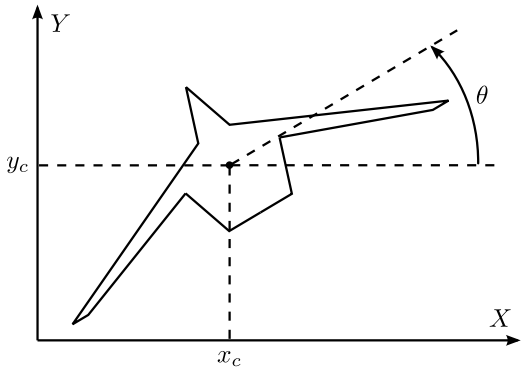

Figure 5.12: Setup of the VTOL example.

and the rolling moment. The parameter ε takes the coupling between the rolling moment and the acceleration of the aircraft into account and is set to $\varepsilon = 0.1$. For the VTOL, the final as well as the integral cost function are defined by

$$V(x) = \Delta x^\mathsf{T} \Delta x, \quad L(x,u) = \Delta x^\mathsf{T} \Delta + 0.01 \Delta u^\mathsf{T} \Delta u.$$

The initial values and desired setpoints are given by

$$\begin{aligned} x_0 &= [100/9.81\,\mathrm{s^2/kg}, 0, 200/9.81\,\mathrm{s^2/kg}, 0, 0, 0]^\mathsf{T}, & u_0 &= [1,0]^\mathsf{T} \\ x_{\mathrm{des}} &= [0, 0, 100/9.81\,\mathrm{s^2/kg}, 0, 0, 0], & u_{\mathrm{des}} &= [1,0]^\mathsf{T}. \end{aligned}$$

The prediction horizon, the sample time, the number of discretization points as well as gradient iterations and the boundaries for the control inputs are

$$\begin{aligned} T &= 2.5\,\mathrm{s}, \quad N_{\mathrm{hor}} = 40, \quad u_{\max} = [3, 6] \\ \Delta t &= 2\,\mathrm{ms}, \quad N = 2, \quad u_{\min} = [0, -6]. \end{aligned}$$

In addition, the explicit line search strategy is used with $\alpha_{\mathrm{init}} = 0.01$, $\alpha_{\max} = 0.01$ and $\alpha_{\min} = 0.001$. All necessary files are included in the directories `<grampc_root>/examples/VTOL` (C) and `<grampc_root>/matlab/examples/VTOL` (MATLAB). The obtained results are shown in Figure 5.13.

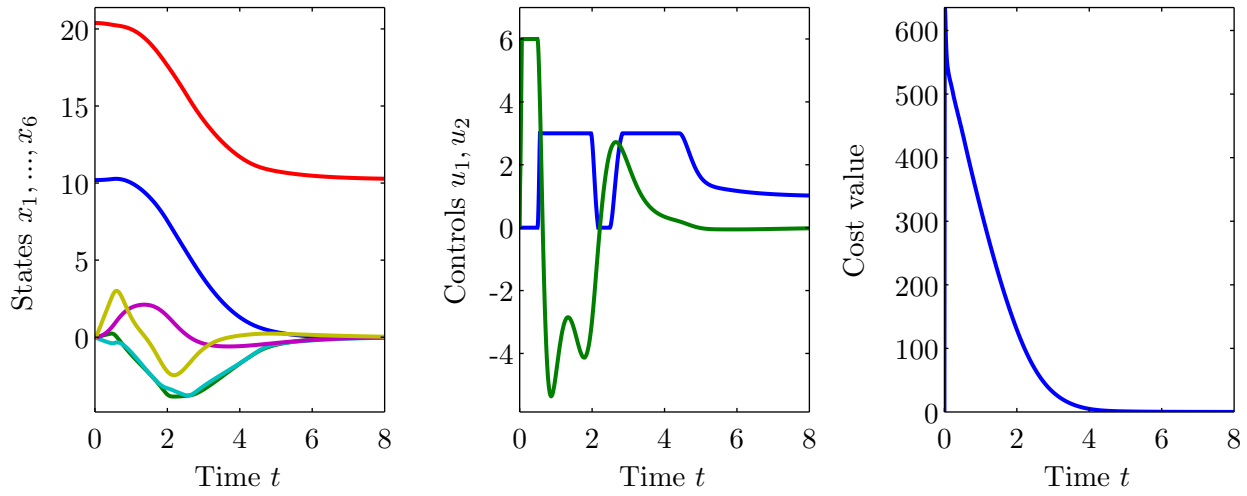

Figure 5.13: Simulation results for the VTOL.

Appendix

A GRAMPC data types

GRAMPC main structure:

```
typedef struct
{
  typeGRAMPCopt *opt;
  typeGRAMPCparam *param;
  typeGRAMPCrws *rws;
  typeGRAMPCsol *sol;
} typeGRAMPC;
```

GRAMPC option structure:

```
typedef struct
{
  typeInt MaxIter;
  typeChar ShiftControl[VALUE_ONOFF];
  typeChar ScaleProblem[VALUE_ONOFF];
  typeChar CostIntegrator[VALUE_COSTINTMETHOD];
  typeChar Integrator[VALUE_INTEGRATOR];
  typeRNum IntegratorRelTol;
  typeRNum IntegratorAbsTol;
  typeChar LineSearchType[VALUE_LSTYPE];
  typeRNum LineSearchMax;
  typeRNum LineSearchMin;
  typeRNum LineSearchInit;
  typeRNum LineSearchIntervalFactor;
  typeRNum LineSearchAdaptFactor;
  typeRNum LineSearchIntervalTol;
  typeChar JacobianX[VALUE_JACOBIANX];
  typeChar JacobianU[VALUE_JACOBIANU];
  typeChar IntegralCost[VALUE_ONOFF];
  typeChar TerminalCost[VALUE_ONOFF];
} typeGRAMPopt;
```

GRAMPC parameter structure:

```
typedef struct
{
  typeInt Nx;
  typeInt Nu;
  typeRNum *xk;
  typeRNum *u0;
  typeRNum *xdes;
  typeRNum *udes;
```

```
  typeRNum Thor;
  typeRNum dt;
  typeRNum tk;
  typeInt Nhor;
  typeRNum *pCost;
  typeRNum *pSys;
  typeRNum *umax;
  typeRNum *umin;
  typeRNum *xScale;
  typeRNum *xOffset;
  typeRNum *uScale;
  typeRNum *uOffset;
  typeInt NpSys;
  typeInt NpCost;
} typeGRAMPCparam;
```

GRAMPC real workspace structure:

```
typedef struct
{
  typeRNum *t;
  typeRNum *x;
  typeRNum *adj;
  typeRNum *u;
  typeRNum *dHdu;
  typeRNum *lsAdapt;
  typeRNum *uls;
  typeRNum *lsExplicit;
  typeRNum *uprev;
  typeRNum *dHduprev;
  typeRNum *J;
  typeRNum *rwsScale;
  typeRNum *rwsGradient;
  typeRNum *rwsCostIntegration;
  typeRNum *rwsAdjIntegration;
  typeRNum *rwsIntegration;
} typeGRAMPCrws;
```

GRAMPC solution structure:

```
typedef struct
{
  typeRNum *xnext;
  typeRNum *unext;
  typeRNum *J;
} typeGRAMPCsol;
```

B GRAMPC function interface

File grampc_init:

```
void grampc_init(typeGRAMPC **grampc);
void grampc_free(typeGRAMPC **grampc);
```

File grampc_run**:**

```
void grampc_run(typeGRAMPC *grampc);

void dHdufct(typeGRAMPC *grampc);

void inputproj(typeRNum *u, typeGRAMPC *grampc);

void lsearch_fit2(typeRNum *kfit, typeRNum *Jfit,
                  typeRNum *k, typeRNum *J);

void interplin(typeRNum *varint, typeRNum *tvec, typeRNum *varvec,
               typeRNum tint, typeInt Nvar, typeInt Nvec,
               typeInt searchdir);

void MatAdd(typeRNum *C, typeRNum *A, typeRNum *B,
            typeInt n1, typeInt n2);

void MatMult(typeRNum *C, typeRNum *A, typeRNum *B,
             typeInt n1, typeInt n2, typeInt n3);

void minfct(typeRNum *amin, typeInt *amini,
            typeRNum *a, typeInt Na);

void maxfct(typeRNum *amax, typeInt *amaxi,
            typeRNum *a, typeInt Na);

void Wadjsys(typeRNum *s, typeRNum *adj, typeRNum *t,
             typeRNum *x, typeRNum *u, typeGRAMPC *grampc);

void Wsys(typeRNum *s, typeRNum *x, typeRNum *t,
          typeRNum *dummy, typeRNum *u, typeGRAMPC *grampcs);

void intCostTrapezodial(typeRNum *s, typeRNum *t, typeRNum *x,
                        typeRNum *u, typeGRAMPC *grampc);

void intCostSimpson(typeRNum *s, typeRNum *t, typeRNum *x,
                    typeRNum *u, typeGRAMPC *grampc);
```

File grampc_setparam**:**

```
void grampc_setparam_real(typeGRAMPC *grampc, typeChar paramName[],
                          typeRNum paramValue);

void grampc_setparam_int(typeGRAMPC *grampc, typeChar paramName[],
                         typeInt paramValue);

void grampc_setparam_vector(typeGRAMPC *grampc, typeChar paramName[],
                            typeRNum *paramValue);

void grampc_printparam(typeGRAMPC *grampc);
```

File grampc_setopt**:**

```
void grampc_setopt_real(typeGRAMPC *grampc, typeChar optName[],
                        typeRNum optValue);

void grampc_setopt_int(typeGRAMPC *grampc, typeChar optName[],
                       typeInt optValue);

void grampc_setopt_string(typeGRAMPC *grampc, typeChar optName[],
                          typeChar optValue[]);

void grampc_printopt(typeGRAMPC *grampc);
```

File grampc_mess**:**

```
void grampc_error(char *mess);

void grampc_error_addstring(char *mess, char *addstring);

void grampc_error_optname(char *optname);

void grampc_error_optvalue(char *optname);

void grampc_error_paramname(char *paramname);

void grampc_error_paramvalue(char *paramname);
```

File probfct**:**

```
void sysdim(typeInt *Nx, typeInt *Nu);

void sysfct(typeRNum *out, typeRNum t, typeRNum *x,
            typeRNum *u, typeRNum *pSys);

void sysjacxadj(typeRNum *out, typeRNum t, typeRNum *x,
                typeRNum *adj, typeRNum *u, typeRNum *pSys);

void sysjacuadj(typeRNum *out, typeRNum t, typeRNum *x,
                typeRNum *adj, typeRNum *u, typeRNum *pSys);

void sysjacx(typeRNum *out, typeRNum t, typeRNum *x,
             typeRNum *u, typeRNum *pSys);

void sysjacu(typeRNum *out, typeRNum t, typeRNum *x,
             typeRNum *u, typeRNum *pSys);

void icostfct(typeRNum *out, typeRNum t, typeRNum *x,
              typeRNum *u, typeRNum *xdes, typeRNum *udes,
              typeRNum *pCost);

void icostjacx(typeRNum *out, typeRNum t, typeRNum *x,
               typeRNum *u, typeRNum *xdes, typeRNum *udes,
               typeRNum *pCost);
```

```
void icostjacu(typeRNum *out, typeRNum t, typeRNum *x,
               typeRNum *u, typeRNum *xdes, typeRNum *udes,
               typeRNum *pCost);

void fcostfct(typeRNum *out, typeRNum t, typeRNum *x,
              typeRNum *xdes, typeRNum *pCost);

void fcostjacx(typeRNum *out, typeRNum t, typeRNum *x,
               typeRNum *xdes, typeRNum *pCost);
```

File `euler1`**:**

```
void intsysEuler(typeRNum *y,
                 typeInt pInt,
                 typeInt Nint,
                 typeRNum *t,
                 typeRNum *x,
                 typeRNum *u,
                 typeGRAMPC *grampc,
                 void (*pfct)(typeRNum *,typeRNum *,typeRNum *,
                              typeRNum *,typeRNum *,typeGRAMPC *));
```

File `eulermod2`**:**

```
void intsysModEuler(typeRNum *y,
                    typeInt pInt,
                    typeInt Nint,
                    typeRNum *t,
                    typeRNum *x,
                    typeRNum *u,
                    typeGRAMPC *grampc,
                    void (*pfct)(typeRNum *,typeRNum *,typeRNum *,
                                 typeRNum *,typeRNum *,typeGRAMPC *));
```

File `heun2`**:**

```
void intsysHeun(typeRNum *y,
                typeInt pInt,
                typeInt Nint,
                typeRNum *t,
                typeRNum *x,
                typeRNum *u,
                typeGRAMPC *grampc,
                void (*pfct)(typeRNum *,typeRNum *,typeRNum *,
                             typeRNum *,typeRNum *,typeGRAMPC *));
```

File ruku45**:**

```
void intsysRuKu45(typeRNum *y,
                  typeInt pInt,
                  typeInt Nint,
                  typeRNum *t,
                  typeRNum *x,
                  typeRNum *u,
                  typeGRAMPC *grampc,
                  void (*pfct)(typeRNum *,typeRNum *,typeRNum *,
                               typeRNum *,typeRNum *,typeGRAMPC *));
```

Bibliography

[1] J. Barzilai and J. M. Borwein. Two-point step size gradient method. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[2] L. D. Berkovitz. *Optimal Control Theory*. Springer, New York, 1974.

[3] J. C. Dunn. On l^2 sufficient conditions and the gradient projection method for optimal control problems. *SIAM Journal on Control and Optimization*, 34(4):1270–1290, 1996.

[4] K. Graichen, M. Egretzberger, and A. Kugi. A suboptimal approach to real-time model predictive control of nonlinear systems. *at - Automatisierungstechnik*, 58(8):447–456, 2010.

[5] K. Graichen and B. Käpernick. A real-time gradient method for nonlinear model predictive control. In T. Zheng, editor, *Frontiers of Model Predictive Control*, pages 9–28. InTech, 2012. http://www.intechopen.com/books/frontiers-of-model-predictive-control.

[6] K. Graichen and A. Kugi. Stability and incremental improvement of suboptimal MPC without terminal constraints. *IEEE Transactions on Automatic Control*, 55(11):2576–2580, 2010.

[7] M. Hehn and R. D'Andrea. A flying inverted pendulum. In *IEEE International Conference on Robotics and Automation*, pages 763–770, 2011.

[8] B. Käpernick and K. Graichen. Model predictive control of an overhead crane using constraint substitution. In *Proceedings of the 2013 American Control Conference*, pages 3979–3984, 2013.

[9] D. E. Kirk. *Optimal Control Theory: An Introduction*. Dover Publications, Mineola, 1970.

[10] M. S. Nikol'skii. Convergence of the gradient projection method on optimal control problems. *Computational Mathematics and Modeling*, 18(2):148–156, 2007.

[11] R. Rothfuss, J. Rudolph, and M. Zeitz. Flatness based control of a nonlinear chemical reactor model. *Automatica*, 32(10):1433–1439, 1996.

[12] S. Sastry. *Nonlinear Systems – Analysis, Stability and Control*. Springer, New-York, 1999.

[13] 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*, pages 587–592, 2010.