User Manual of *ParNMPC* by Examples

A Code Generation Toolbox for Parallel Nonlinear Model Predictive Control with OpenMP

Haoyang Deng, Toshiyuki Ohtsuka
Department of Systems Science, Graduate School of Informatics
Kyoto University
January 21, 2018
Beta 2.0

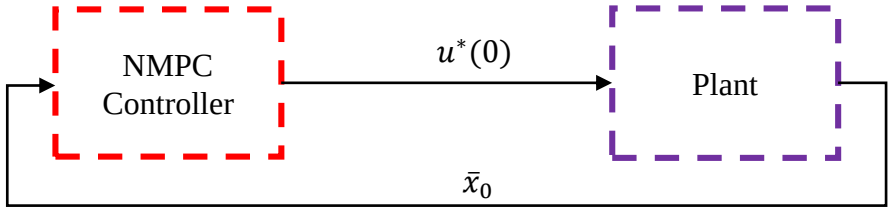
- *Highly Parallelizable*
- *Fast Rate of Convergence*
- *Parallel C/C++ Code Generation*
- *Closed-loop Simulation*
- *Easy to Use*

Contents

Introduction

ParNMPC

Simulation Loop



ParNMPC is a Matlab software toolbox developed to carry out closed-loop NMPC simulation and parallel C/C++ code generation with OpenMP. *ParNMPC* is based on a highly parallelizable Newton-type algorithm, which can utilize N cores at most to calculate the optimal control input. To use *ParNMPC*, the user has to define the NMPC Controller, the Plant and the Simulation Loop. Parameters and functions need to be defined are listed below.

NMPC Controller

$$\min_{x(\cdot),u(\cdot)} \int_0^T L\big(u(t),x(t),p(t)\big)dt$$

$$\text{s.t.}\quad \dot{x}(t) = f\big(u(t),x(t),p(t)\big),$$
$$x(0) = \bar{x}_0,$$
$$C\big(u(t),x(t),p(t)\big) = 0,\ t \in (0,T].$$

Functions:
- f: dynamic model for controller design
- L: cost function
- C: constraint

Parameters:
- x: state vector
- u: input vector
- p: parameter vector
- $\bar{x}_0$: initial state vector
- T: prediction horizon
- N: number of discretization grids

Plant

$$\dot{x}_{sim} = f_{sim}(u_{sim}, x_{sim}, p_{sim})$$

- f_{sim}: dynamic model of the plant
- p_{sim}: parameter in the plant model

Simulation Loop

Simulation parameters:
- Sampling interval T_s
- Simulation length

Introduction

- In practice, *ParNMPC* solves the discretized optimal control problem (OCP) given time-dependent parameters $\{p_i\}_{i=1}^N$ and current state $\bar{x}_0$, then apply the first optimal control input u_1^*.

$$\begin{aligned}
&\min_{x(\cdot),\,u(\cdot)} \int_0^T L\big(u(t),x(t),p(t)\big)dt \\
&\text{s.t. } \dot{x}(t) = f\big(u(t),x(t),p(t)\big), \\
&\quad x(0) = \bar{x}_0, \\
&\quad C\big(u(t),x(t),p(t)\big) = 0,\ t \in (0,T].
\end{aligned}
\quad \xrightarrow[\text{implicit Euler method}]{\text{Discretization with the}} \quad
\begin{aligned}
&\min_{X,\,U} \sum_{i=1}^N L(u_i,x_i,p_i)\Delta\tau \\
&\text{s.t. } x_i = x_{i-1} + f(u_i,x_i,p_i)\Delta\tau, \\
&\quad x_0 = \bar{x}_0, \\
&\quad C(u_i,x_i,p_i) = 0,\ i = 1,\cdots,N, \\
&\text{where } X = (x_1,x_2,\cdots,x_N), \\
&\quad U = (u_1,u_2,\cdots,u_N), \\
&\quad \Delta\tau = T/N.
\end{aligned}$$

- More specifically, *ParNMPC* solves the KKT conditions for the discretized OCP, i.e., the following nonlinear algebraic equations with respect to the costates $\{\lambda_i\}_{i=1}^N$, the multipliers $\{\mu_i\}_{i=1}^N$, the inputs $\{u_i\}_{i=1}^N$ and the states $\{x_i\}_{i=1}^N$:

$$\begin{bmatrix} x_{i-1} - x_i + f(u_i,x_i,p_i)\Delta\tau \\ C(u_i,x_i,p_i)\Delta\tau \\ H_u^T(\lambda_i,\mu_i,u_i,x_i,p_i)\Delta\tau \\ \lambda_{i+1} - \lambda_i + H_x^T(\lambda_i,\mu_i,u_i,x_i,p_i)\Delta\tau \end{bmatrix}_{i=1}^N = 0 \in \mathbb{R}^{N(2n_x+n_\mu+n_u)}$$

Here, $H(\lambda,\mu,u,x,p) \coloneqq L(u,x,p) + \lambda^T f(u,x,p) + \mu^T C(u,x,p)$ denotes the Hamiltonian, $\lambda \in \mathbb{R}^{n_x}$, $\mu \in \mathbb{R}^{n_\mu}$, $u \in \mathbb{R}^{n_u}$, $x \in \mathbb{R}^{n_x}$, $p \in \mathbb{R}^{n_p}$, $x_0 = \bar{x}_0$ and $\lambda_{N+1} = 0$.

Installation

1. Download ParNMPC-master.zip
2. Extract files (5 folders):
   - *CSTR*
   - *DoubleInvertedPendulum*
   - *Quadrotor*
   - *SemiActiveDamper*
   - *UserManual*
3. Go into one of the application folders, e.g., *Quadrotor*
4. Files and folders in *Quadrotor*:
   - Folders
     - *codegen* (Pre-generated C code)
     - *Functions* (Functions and scripts)
   - Files:
     - ParNMPC_Def.m (Define the simulation problem)
     - ParNMPC_Init.m (Solve the very first OCP)
     - ParNMPC.m (For simulation and code generation)
     - initialSolution.mat (Solution to the very first OCP)

- ParNMPC is application-oriented and each application folder is self-contained.

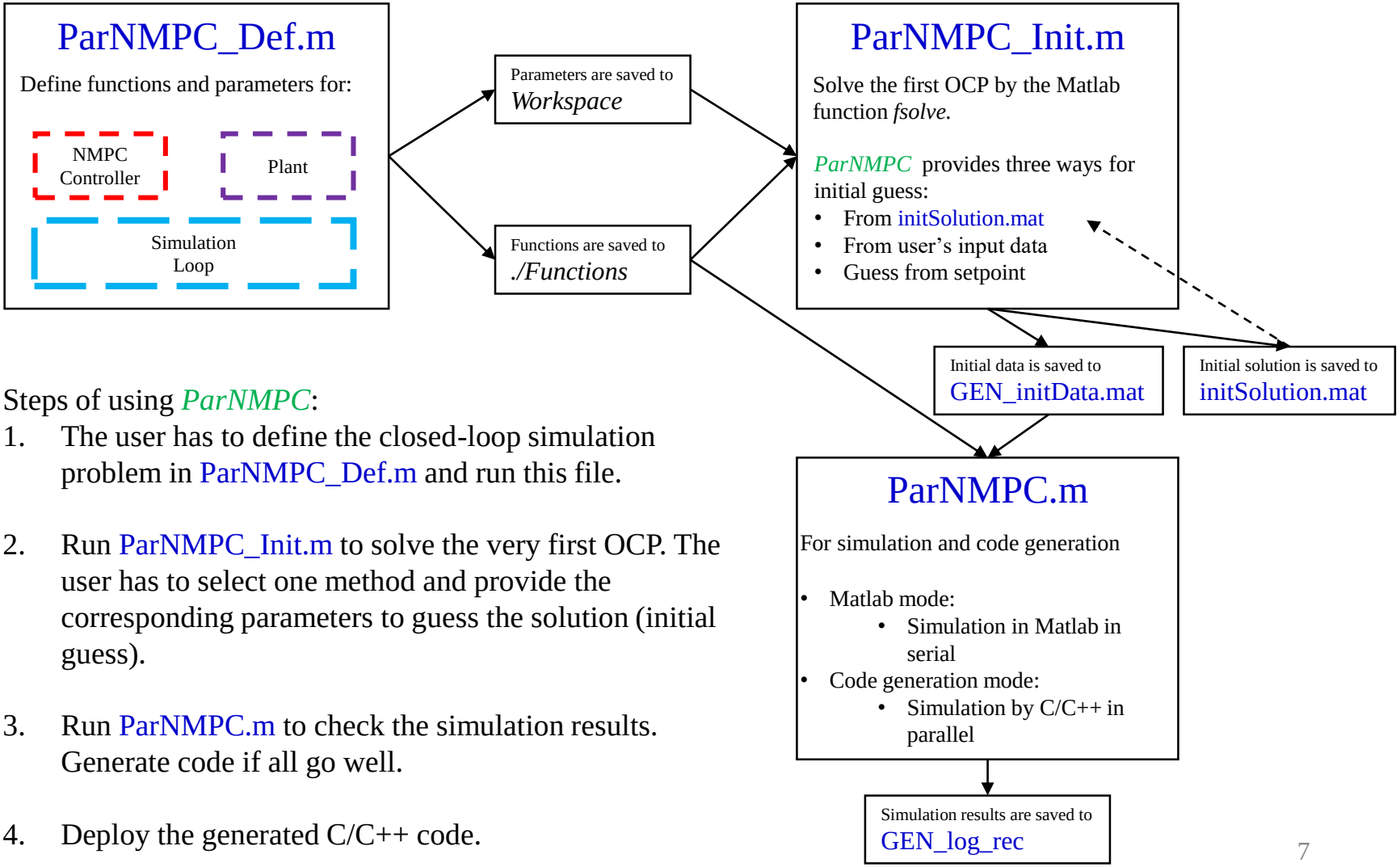
System Requirements

- MATLAB 2016a or later
  - MATLAB Coder
  - Optimization Toolbox
  - Parallel Computing Toolbox
  - Symbolic Math Toolbox

- Compiler
  - C/C++ Compilers supporting OpenMP for code generation
    - See Supported and Compatible Compilers

- Tested Environments:
  - Windows 10 + MATLAB 2016a + Microsoft Visual C++ 2015 Professional
  - Windows 10 + MATLAB 2017b + Microsoft Visual C++ 2017 Community

Flow of *ParNMPC*



Steps of using *ParNMPC*:

1. The user has to define the closed-loop simulation problem in ParNMPC_Def.m and run this file.

2. Run ParNMPC_Init.m to solve the very first OCP. The user has to select one method and provide the corresponding parameters to guess the solution (initial guess).

3. Run ParNMPC.m to check the simulation results. Generate code if all go well.

4. Deploy the generated C/C++ code.

Getting Started
Semi-Active Damper

Simulation Loop:

- Sampling interval: 0.01 s
- Simulation length: 20 s

Plant:

- Same as in NMPC:

$$f_{sim}(u,x,p_{sim}) = \begin{bmatrix} x_2 \\ ax_1 + bx_2u_1 \end{bmatrix}$$

NMPC Controller:

- Dynamics:

$$f(u,x,p) = \begin{bmatrix} x_2 \\ ax_1 + bx_2u_1 \end{bmatrix}, a = b = -1$$

- Constraints (Converted into equality constraint by introducing dummy input u_2 (see *ref*)):

$$-1 \le u_1 \le 1 \longrightarrow C(u,x,p) = u_1^2 + u_2^2 - 1$$

- Stage cost function:

$$L(u,x,p) = \frac{1}{2}(\|x - x_{ref}\|_Q^2 + \|u - u_{ref}\|_R^2 - r \cdot u)$$

$Q = \text{diag}(10,10)$, $R = \text{diag}(0.01,0.01)$, $r = [0,0.1]$

$x_{ref} = [0,0]^T$, $u_{ref} = [0,1]^T$

- Initial state: $\bar{x}_0 = [1,0]^T$
- Prediction horizon: 2 s
- Number of discretization grids: $N = 36$
- Max number of iterations: 5
- Tolerance: 1e-5

Ref: Toshiyuki Ohtsuka. "A continuation/GMRES method for fast computation of nonlinear receding horizon control." *Automatica* 40.4 (2004): 563-574.

Getting Started

Semi-Active Damper – Problem Definition

Step 1:
Define functions and parameters as follows in ParNMPC_Def.m and run ParNMPC_Def.m.

Simulation Loop:

```
Ts = 0.01;        % sampling interval
simuLength = 20; % simulation length
```

Plant:

```
% Definition of dimension
pSimDim  = 0; % dimension of the parameter in the
simulation model

% Definition of system dynamics f_sim(u,x,p_sim)
fSim([u;x;pSim]) =
      [x(2); a * x(1) + b * x(2) * u(1)];

% init pSim
pSimVal = zeros(pSimDim,1);
```

NMPC Controller:

```
% Definition of dimensions
xDim  = 2; % dimension of states
uDim  = 2; % dimension of inputs including dummy inputs
muDim = 1; % dimension  of constraints
pDim  = 0; % dimension of parameters for the controller

% Definition of system dynamics f(u,x,p)
a = -1;
b = -1;
f([u;x;p]) = [x(2); a * x(1) + b * x(2) * u(1)];

% Definition of equality constraint C(u,x,p)
uMax = 1;
uMin = -1;
uBar = (uMax + uMin)/2;
C([u;x;p]) = (u(1) - uBar)^2 + u(2)^2 - (uMax - uBar)^2;

% Definition of cost function L(u,x,p)
Q = diag([10, 10]);
R = diag([0.01, 0.01]);
r = [0,0.1];
xRef = [0;0];
uRef = [0;uMax];
L([u;x;p]) =  0.5*(x-xRef).'*Q*(x-xRef)...
             +0.5*(u-uRef).'*R*(u-uRef)...
             -0.5*r*u;

% Definition of NMPC settings
x0Value = [1;0]; % init state
T  = 2; % prediction horizon
N  = 24; % number of discretization grids
pVal = zeros(pDim,N); % parameters
MaxIterNum = 5; % max num of iterations per update
tolerance = 1e-5; % tolerance of the KKT conditions to terminate
```

Getting Started

Semi-Active Damper – Initialization

Step 2:
Define the initialization method by `initMethod = INIT_BY_FILE` in ParNMPC_Init.m and run ParNMPC_Init.m, solution will be initialized from file initialSolution.mat.

- If initialSolution.mat is not available, another method should be selected by either `initMethod = INIT_BY_INPUT` or `INIT_BY_REF_GUESS`, and the corresponding parameters should be provided.
- The optimal solution will be automatically saved to initialSolution.mat.

Getting Started

Semi-Active Damper – Simulation & Code Generation

Step 3:

1. Run ParNMPC.m to validate the closed-loop simulation. The results will be saved to GEN_log_rec.mat.
   - This step is not in parallel. Only the generated C/C++ code is in parallel.
   - If the simulation doesn't go well, it is most likely that the first OCP is not well solved, or the measured state and time-dependent parameter change dramatically.

2. Code generation:
   1. Set up your compiler by *mex -setup*.
   2. Open the MATLAB Coder app.
   3. Select ParNMPC.m.
   4. Click Generate.

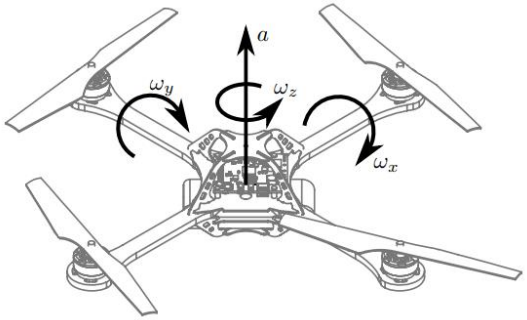
Getting Started
Semi-Active Damper – Deployment

Step 4:

1. Open Visual Studio.
2. Create an empty Win32 Console Application project.
3. Change to Release x64 mode.
4. Add *.h and *.c files in *\codegen\lib\ParNMPC* to project.
5. Add *.h and *.c files in *\codegen\lib\ParNMPC\examples* to project.
6. Add the folder *\codegen\lib\ParNMPC* to *Properties* → *C/C++* → *General* → *Additional Include Directories*.
7. *Properties* → *C/C++* → *Language* → *Open MP Support: Yes (/openmp).*
8. Compile and run.

Getting Started

Quadrotor



Simulation Loop:

- Sampling interval: 0.01 s
- Simulation length: 20 s

Plant:

- Same as in NMPC:

$$\ddot{X} = a(\cos\gamma \sin\beta \cos\alpha + \sin\gamma \sin\alpha)$$
$$\ddot{Y} = a(\cos\gamma \sin\beta \sin\alpha - \sin\gamma \cos\alpha)$$
$$\ddot{Z} = a\cos\gamma \cos\beta - g$$
$$\dot{\gamma} = (\omega_X \cos\gamma + \omega_Y \sin\gamma)/\cos\beta$$
$$\dot{\beta} = -\omega_X \sin\gamma + \omega_Y \cos\gamma$$
$$\dot{\alpha} = \omega_X \cos\gamma \tan\beta + \omega_Y \sin\gamma \tan\beta + \omega_Z$$

NMPC Controller:

- States:

$$x = [X \quad \dot{X} \quad Y \quad \dot{Y} \quad Z \quad \dot{Z} \quad \gamma \quad \beta \quad \alpha]^T \in \mathbb{R}^9$$

- Dynamics:

$$\ddot{X} = a(\cos\gamma \sin\beta \cos\alpha + \sin\gamma \sin\alpha)$$
$$\ddot{Y} = a(\cos\gamma \sin\beta \sin\alpha - \sin\gamma \cos\alpha)$$
$$\ddot{Z} = a\cos\gamma \cos\beta - g$$
$$\dot{\gamma} = (\omega_X \cos\gamma + \omega_Y \sin\gamma)/\cos\beta$$
$$\dot{\beta} = -\omega_X \sin\gamma + \omega_Y \cos\gamma$$
$$\dot{\alpha} = \omega_X \cos\gamma \tan\beta + \omega_Y \sin\gamma \tan\beta + \omega_Z$$

- Constraints (Converted to equality constraints by introducing dummy inputs u_5, u_6, u_7, u_8):

$$\begin{bmatrix} 0 \\ -1 \\ -1 \\ -1 \end{bmatrix} \leq \begin{bmatrix} a \\ \omega_X \\ \omega_Y \\ \omega_Z \end{bmatrix} \leq \begin{bmatrix} 11 \\ 1 \\ 1 \\ 1 \end{bmatrix} \longrightarrow C(u,x,p) = \begin{bmatrix} (a-5.5)^2 + u_5^2 - 5.5^2 \\ \omega_X^2 + u_6^2 - 1 \\ \omega_Y^2 + u_7^2 - 1 \\ \omega_Z^2 + u_8^2 - 1 \end{bmatrix}$$

- Stage cost function:

$$L(u,x,p) = \frac{1}{2}(\|x - x_{ref}\|_Q^2 + \|u - u_{ref}\|_R^2 - r \cdot u)$$

$$Q = \text{diag}(10,1,2,1,10,1,1,1,1),\ R = \text{blkdiag}(I_4, 0.01 \cdot I_4)$$

$$r = [\text{zeros}(1,4), 0.1 \cdot \text{ones}(1,4)]$$

$$u_{ref} = [g, 0,0,0,3.4167,1,1,1]^T$$

$$x_{ref}(t) = \text{zeros}(9,1),\ t \in [0,10]$$

$$x_{ref}(t) = [1.5,0,1.5,0,1.5,0,0,0,0]^T,\ t \in (10,20]$$

- Initial state: $\bar{x}_0 = [1,0,1,0,1,0,0,0,0]^T$
- Prediction horizon: 1 s
- Number of discretization grids: $N = 24$
- Max number of iterations: 5
- Tolerance: 5e-3

Ref: Markus Hehn and Raffaello D'Andrea. "A flying inverted pendulum." *IEEE International Conference on Robotics and Automation (ICRA), 2011.*

Getting Started

Quadrotor – Problem Definition

Step 1:
Define functions and parameters as follows in ParNMPC_Def.m and run ParNMPC_Def.m.

Simulation Loop:

```
Ts = 0.01;       %sampling interval
simuLength = 10; %simulation length in seconds
```

Plant:

```
% Definition of dimensions
pSimDim  = 0; % dimension of the parameter in the simulation model

% Definition of system dynamics f_sim(u,x,p_sim)
fSim([u;x;pSim]) = [x(2);...
            u(1)*(cos(x(7))*sin(x(8))*cos(x(9)) + sin(x(7))*sin(x(9)));...
            x(4);...
            u(1)*(cos(x(7))*sin(x(8))*sin(x(9)) - sin(x(7))*cos(x(9)));...
            x(6);...
            u(1)*cos(x(7))*cos(x(8)) - g;...
            (u(2)*cos(x(7)) + u(3)*sin(x(7)))/cos(x(8));...
            -u(2)*sin(x(7)) + u(3)*cos(x(7));...
            u(2)*cos(x(7))*tan(x(8)) + u(3)*sin(x(7))*tan(x(8)) + u(4)];

% Init pSim
pSimVal = zeros(pSimDim,1);
```

NMPC Controller:

```
% Definition of dimensions
xDim  = 9; % dimension of states
uDim  = 8; % dimension of inputs including dummy inputs
muDim = 4; % dimension  of constraints
pDim  = xDim+uDim; % dimension of parameters for the controller

% Definition of system dynamics f(u,x,p)
g = 9.81;
f([u;x;p]) = [x(2);...
              u(1)*(cos(x(7))*sin(x(8))*cos(x(9)) + sin(x(7))*sin(x(9)));...
              x(4);...
              u(1)*(cos(x(7))*sin(x(8))*sin(x(9)) - sin(x(7))*cos(x(9)));...
              x(6);...
              u(1)*cos(x(7))*cos(x(8)) - g;...
             (u(2)*cos(x(7)) + u(3)*sin(x(7)))/cos(x(8));...
             -u(2)*sin(x(7)) + u(3)*cos(x(7));...
              u(2)*cos(x(7))*tan(x(8)) + u(3)*sin(x(7))*tan(x(8)) + u(4)];

% Definition of equality constraint C(u,x,p)
uMax = [11;1;1;1];
uMin = [0;-1;-1;-1];
uBar = (uMax + uMin)/2;
C([u;x;p]) = [(u(1) - uBar(1))^2 + u(5)^2 - (uMax(1)-uBar(1))^2;...
              (u(2) - uBar(2))^2 + u(6)^2 - (uMax(2)-uBar(2))^2;...
              (u(3) - uBar(3))^2 + u(7)^2 - (uMax(3)-uBar(3))^2;...
              (u(4) - uBar(4))^2 + u(8)^2 - (uMax(4)-uBar(4))^2];

% Definition of cost function L(u,x,p)
Q = diag([10, 1, 2, 1, 10, 1, 1, 1, 1]);
R = diag([1, 1, 1, 1 ,...
          0.01, 0.01, 0.01, 0.01 ]);
r = [0,0,0,0,...
     1,1,1,1]*0.1;
L([u;x;p]) =  0.5*(x-p(1:xDim))'*Q*(x-p(1:xDim))...
             +0.5*(u-p(pDim-uDim+1:pDim))'*R*(u-p(pDim-uDim+1:pDim))...
             -0.5*r*u;

% Definition of NMPC settings
x0Value = [1;0;1;0;1;0;0;0;0]; % initial state
T = 1.0; % prediction horizon
N  = 24; % number of discretization grids
xRefConstant = [0;0;0;0;0;0;...
                0;0;0];
uRefConstant = [g;0;0;0;...
                sqrt(-(g - uBar(1))^2 + (uMax(1)-uBar(1))^2);...
                uMax(2);uMax(3);uMax(4)];
pVal = repmat([xRefConstant;uRefConstant],1,N); % init parameters
MaxIterNum = 5; % max num of iterations per update
tolerance = 5e-3; % tolerance of the KKT conditions to terminate
```

Getting Started

Quadrotor – Initialization

- Step 2: refer to *Semi-Active Damper – Initialization*

Quadrotor– Simulation & Code Generation

- Step 3: refer to *Semi-Active Damper – Simulation & Code Generation*

Quadrotor– Deployment

- Step 4: refer to *Semi-Active Damper – Deployment*

Possible Reasons Why *ParNMPC* Fails

1. The first OCP is not well solved. *ParNMPC* utilizes the warm-start strategy and highly depends on the solution of the first OCP. *ParNMPC* may fail even when the norm of the KKT conditions (first-order optimality) for the first OCP is small enough. In general, a good solution is smooth enough. In order to find a good solution to the first OCP, the method of continuation can be employed. For example, the solution of the first OCP can be initialized from the solution of an easy-to-solve problem, which has less active constraints or a shorter prediction horizon.

2. Dramatic changes of the measured state or time-dependent parameter. *ParNMPC* converges locally and requires that the current solution is close to the last solution. The current version of *ParNMPC* doesn't contain any globalization techniques such as line-search and trust-region methods. Make sure that the state and parameter don't vary too much with respect to time.

3. Ill-conditioned problem. It is usually cased by the settings of the weighting matrices.

License

ParNMPC is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

ParNMPC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this program. If not, see http://www.gnu.org/licenses/.