PolyMPC: An efficient Tool for Embedded Model Predictive Control for Fast Mechatronic Systems

Petr Listov, Michael Spieler, Colin Jones

École polytechnique fédérale de Lausanne

ABSTRACT

Efficient solvers for real-time Nonlinear Model Predictive Control (NMPC) are needed to run on embedded hardware with highly constrained computational resources. Most existing solvers are based on second-order methods which are prohibitively expensive for some applications. This project explores a Sequential Quadratic Programming (SQP) strategy using an ADMM based Quadratic Program (QP) solver. It is implemented as a generic nonlinear solver in form of a header-only module, which integrates into PolyMPC, an open-source C++ library for real-time NMPC. A pseudospectral collocation based approximation method is used to efficiently solve the OCP, while forward mode automatic differentiation simplifies the problem construction. We leverage the flexibility of templated C++ with the Eigen linear algebra library to solve OCPs without relying on dynamic memory allocation. The implementation is suitable to run on a microcontroller with Floating Point Unit (FPU), which was tested on a path following problem of a two-line soft-wing kite.

KEY FEATURES AND PRIMER

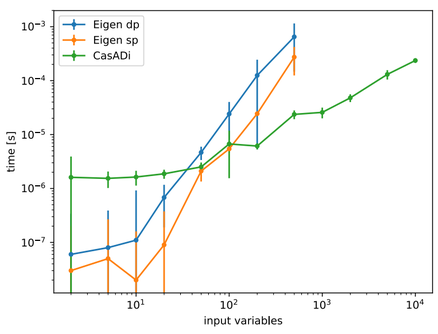
Efficiency: compile-time polymorphism and generic programming in C++ are used to avoid expensive calling of virtual methods.

Modularity: the software does not use any custom defined modelling language, but rather relies on popular dynamic optimization and linear algebra frameworks. This allows one to use implemented tracking and path-following predictive controllers, as well as utilize the building blocks of the algorithms such as the polynomial interpolation, collocation of differential equations and quadrature rule approximation of integrals.

Usability: PolyMPC does not use code generation, algorithms are implemented in a compact and readable way.

Automatic Differentiation: CasADi (forward and backward modes) and Eigen AutoDiff module (only forward mode, can work on microcontrollers).

$$f(x_1, \dots, x_m) = \sum_{i=1}^{m} \left[(a - x_i)^2 + b(x_{i+1} - x_i^2)^2\right]$$



Deployment: in-house SQP and QP solvers, no third party libraries (except Eigen) and dynamic memory allocation for embedded applications.

Memory Optimization: support for direct and indirect linear algebra solvers, in-place LDLT decomposition, cheap damped BFGS Hessian updates, support single and double precision.

```
#include "control/nmpc.hpp"
#include "kinematic_kite_model.hpp"
#include "polynomials/ebyshev.hpp"
#include "solvers/sqp.hpp"

using Problem = polympc::OCProblem<Kite<double>, Lagrange<double>,
                                   Mayer<double>>;
using Approximation = Chebyshev<3, GAUSS_LOBATTO, double>;
using controller_t = polympc::nmpc<Problem, Approximation, sqp::SQP>;

int main(int argc, char **argv)
{
    controller_t kite_controller;

    controller_t::State x = {-1, -1, 1.5};
    controller_t:Control u;
    controller_t::Parameters p(0.5);

    kite_controller.setStateBounds(xl, xu);   ///set the bounds
    kite_controller.setControlBounds(ul, uu);
    kite_controller.setParameters(p);

    kite_controller.computeControl(x);
    kite_controller.getOptimalControl(u);
}
```

Modifier les styles du texte du masque

SIMULATION RESULTS

Kite model equations:

$$\begin{bmatrix} L & 0 \\ 0 & L\cos\theta \end{bmatrix} \begin{matrix} \dot{\theta} \\ \dot{\varphi} \end{matrix} = \bar{R}_{NK} \begin{bmatrix} 1 & 0 & -E \\ 0 & 0 & 0 \end{bmatrix} R_{NK}^T R_{GN}^T v_\omega - \bar{R}_{NK} \begin{bmatrix} Ez \\ 0 \end{bmatrix}$$

$$\dot{\gamma} = u_\gamma$$

Where

$$R_{GN} = \begin{bmatrix} -\sin\theta\cos\varphi & -\sin\varphi & -\cos\theta\cos\varphi \\ -\sin\theta\sin\varphi & \cos\varphi & -\cos\theta\sin\varphi \\ \cos\theta & 0 & -\sin\theta \end{bmatrix} \quad R_{NK} = \begin{bmatrix} \bar{R}_{NK} & 0 \\ 0 & 1 \end{bmatrix} \quad \bar{R}_{NK} = \begin{bmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{bmatrix}$$

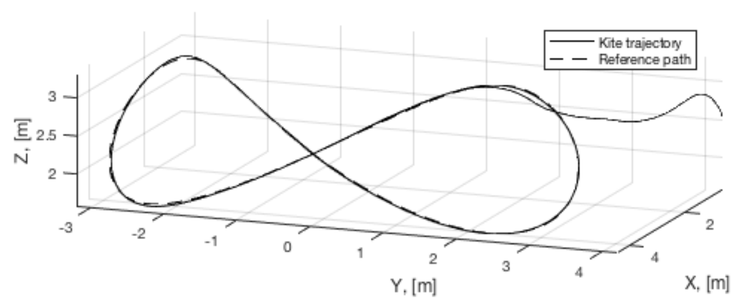
Path equations:

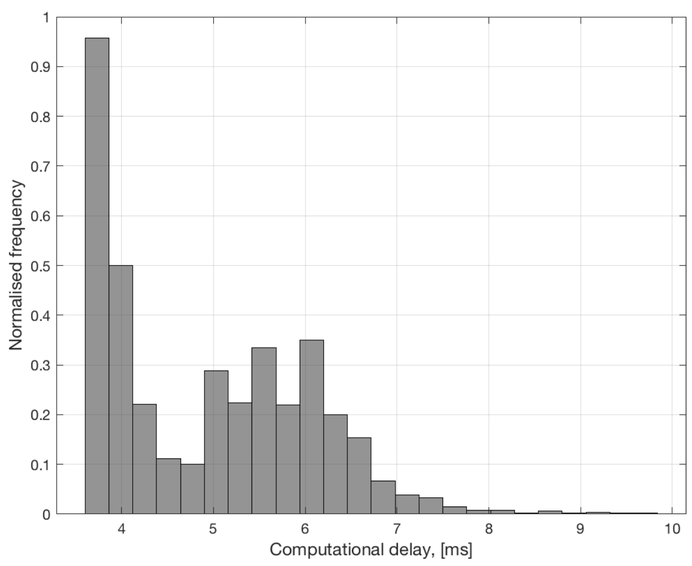
$$\theta(\tau) = h + a\sin(2\tau)$$
$$\varphi(\tau) = 4a\cos(2\tau)$$

State and control constraints:

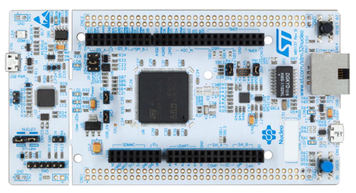
$$0 \le \theta(t) \le \frac{\pi}{2} \qquad -\pi \le \gamma(t) \le \pi$$
$$-\frac{\pi}{2} \le \varphi(t) \le \frac{\pi}{2} \qquad -5 \le u_\gamma \le 5$$

The simulator was run at 100 Hz, and the controller node was set to run at 50Hz with the following parameters: number of collocation points- 12; number of subintervals- 3, prediction horizon- 1.5 seconds.





TESTING ON EMBEDDED PLATFORMS



Name	Odroid XU4	Nucleo-F767ZI
Platform	Samsung Exynos5422	STM32F767ZI
CPU	8x ARM Cortex™-A15/A7	ARM Cortex™-M7
Architecture	ARMv7-A (32bit)	ARMv7E-M (32bit)
Acceleration	FPU, NEON SIMD, DSP	FPU (DP+SP), DSP
Clock	2GHz	216MHz
RAM	2GB LPDDR3	512KB SRAM
Storage	16GB eMMC/SDCard	2MB flash
OS	Ubuntu 18.04	bare metal
Dimensions	83mm x 58mm	133mm x 70mm
Power consumption	10W - 20W	<1.5W

Platform	Solve time [ms]	Factor
Intel Core i7 2.8 GHz	5.83	1.0
ARM Cortex-A15	19.21	3.3
ARM Cortex-M7	349.00	59.9

REFERENCES

1. Diwale S., Alessandretti A., Lymperopoulos I., Jones C. N., Nonlinear Adaptive Controller for Airborne Wind Energy Systems. In: American Control Conference: 4101-4106, IEEE; 2016.