acados

A rapid prototyping tool for NMPC

Robin V, Dimitris K, Andrea Z, Rien Q, Niels v. D, Joris G, and Moritz Diehl

Systems Control and Optimization laboratory

October 4, 2016

How things were until now

www.acadotoolkit.org

Key Properties of ACADO Toolkit [Houska et al 2009]

- Open Source (LGPL)
- **A**utomatic **C**ontrol **A**nd **D**ynamic **O**ptimization
- User friendly interface close to mathematical syntax

How things were until now

www.acadotoolkit.org

Key Properties of ACADO Toolkit [Houska et al 2009]

- Open Source (LGPL)
- **A**utomatic **C**ontrol **A**nd **D**ynamic **O**ptimization
- User friendly interface close to mathematical syntax

Multiplatform support

- C++: Linux, OS X, Windows
- MATLAB

ACADO developers (past and current)

Moritz Diehl
Scientific advisor

Hans Joachim Ferreau
Main developer

Boris Houska
Main developer

Filip Logist
Multi-objective optimization

Rien Quirynen
Code generation

Dries Telen
Optimal Experimental Design

Mattia Valerio
Multi-objective optimal control

Milan Vukov
Code generation for MPC & MHE

ACADO Code Generation Toolkit

Pro

- Easy to use,

Con

- ... but codebase difficult to maintain

ACADO Code Generation Toolkit

Pro

- Easy to use,
- Fast NMPC/MHE solvers,

Con

- ... but codebase difficult to maintain
- ... but with a lot of global data

ACADO Code Generation Toolkit

Pro

- Easy to use,
- Fast NMPC/MHE solvers,
- Interfaced to external solvers,

Con

- ... but codebase difficult to maintain
- ... but with a lot of global data
- ... but hard coupling with new ones

ACADO Code Generation Toolkit

Pro

- Easy to use,
- Fast NMPC/MHE solvers,
- Interfaced to external solvers,
- Comes with own AD,

Con

- ... but codebase difficult to maintain
- ... but with a lot of global data
- ... but hard coupling with new ones
- ... but not compatible with CasADi

Code generation

Mathematical Formulation

$$\begin{aligned} \min_{x,u} \quad & \int_0^T x^2 + u^2 \mathrm{d}t \\ & \dot{x} = f(x,u) \\ \text{s.t.} \quad & x(0) = x_0 \\ & -1 \leq u \leq 1 \, . \end{aligned}$$

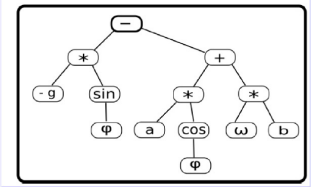
ACADO Syntax

```
DifferentialState    x;
Control              u;

DifferentialEquation f;
f << dot(x) == u + ...;

ocp.minLagrangeTerm( x*x+u*u );
ocp.subjectTo( f );
ocp.subjectTo( -1 <= u <= 1 );
```

Symbolic Structure Detection



Algorithm

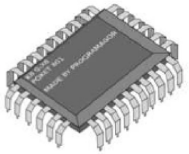
- **Multiple Shooting**
- **Real-Time Gauss Newton**
- **Online Active Set Strategy**

Optimized C-Code

```
r[1] = a[15]*c[17] + a[16]*c[19] + ... ;
r[2] = sin(a[1]*a[2]) + a[4] + ... ;
r[3] = cos(r[1])/exp(c[4])+ r[1] +... ;
```

Customized Solver Implemented on Chip/FPGA:

Measurement x_0



Optimal Decision u*

Code generation

Mathematical Formulation

$$\min_{x,u} \int_0^T x^2 + u^2 \mathrm{d}t$$
$$\text{s.t.} \quad \dot{x} = f(x,u)$$
$$x(0) = x_0$$
$$-1 \le u \le 1 \, .$$

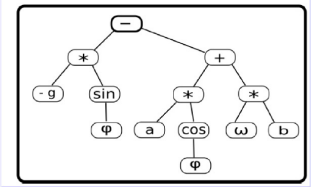
ACADO Syntax

```
DifferentialState     x;
Control               u;

DifferentialEquation f;
f << dot(x) == u + ...;

ocp.minLagrangeTerm( x*x+u*u );
ocp.subjectTo( f );
ocp.subjectTo( -1 <= u <= 1 );
```

Symbolic Structure Detection



Algorithm

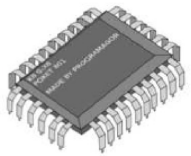
- **Multiple Shooting**
- **Real-Time Gauss Newton**
- **Online Active Set Strategy**

Optimized C-Code

```
r[1] = a[15]*c[17] + a[16]*c[19] + ... ;
r[2] = sin(a[1]*a[2]) + a[4] + ... ;
r[3] = cos(r[1])/exp(c[4])+ r[1] +... ;
```

Customized Solver Implemented on Chip/FPGA:

Measurement x_0



Optimal Decision u*

But is codegen really necessary?

Code generation

Why code generation?

- eliminate computations
- known dimensions and sparsity patterns
- no dynamic memory
- code reorganization, ...

Code generation

Why code generation?

- eliminate computations
- known dimensions and sparsity patterns
- no dynamic memory
- code reorganization, ...

But code generation

- increases code size
- creates unreadable code
- only really makes sense at the bottleneck $\rightarrow$ linear algebra

A different idea

Instead of code-generated code, we want

- maintainable,
- extensible,
- fast code,
- with efficient linear algebra kernels,
- available from MATLAB and Python.

$\longrightarrow$ acados

Interfaces

The structure of acados is based on internal interfaces:

- OCP QP interface,
- condensing interface,
- integrator interface,
- AD interface,
- Hessian regularization interface,
- ...

If a solver sticks to a certain interface, it can be swapped quickly for another one

$$\rightarrow \text{Rapid Prototyping}$$

Status

As of now: one NMPC example in C99 with

- Model: [Chen&Allgoewer 1998]
- Integrator: Runge-Kutta 4
- Condensing: N^2
- QP solver: qpOASES3.0

Status

As of now: one NMPC example in C99 with

- Model: [Chen&Allgoewer 1998]
- Integrator: Runge-Kutta 4
- Condensing: N^2
- QP solver: qpOASES3.0

$0.63\,\mathrm{ms}$ per iteration,
only 2x slower than ACADO code generated code,
not bad for first 'plain vanilla' attempt ☺

Codebase

Located at `https://github.com/acados/acados`

- Completely written in `C99`
- Each interface has its own header file
- Data passed around by pointers to `structs`

Codebase

Located at `https://github.com/acados/acados`

- Completely written in `C99`
- Each interface has its own header file
- Data passed around by pointers to `structs`

TODO:

- Interface with CasADi (and thus MATLAB/Python)
- Use BlasFEO's (former HPMPC) customly optimized linear algebra
- Implement many more new solvers
- Real-life examples
- ...