MPC – Brief overview and software

Dr. Lilli Frison

Fraunhofer Institute for Solar Energy System ISE,
Department of Microsystems Engineering IMTEK,
University of Freiburg

July 12, 2023

Control challenges in a renewable-based energy system

Control challenges in a renewable-based energy system

Transition from a fossil fuel based to a renewable-based energy system poses new **(control) challenges**:

- high energy yield but fluctuating
- grid stability, supply reliability
- energy storage
- energy efficiency (e.g. buildings, heating networks, technologies)

Renewable-based energy system are highly diverse and complex systems:

- nonlinear (wind energy, hydraulics)
- mixed-integer (energy networks, plant dynamics, change of dynamics)
- fast and slow dynamics (seasonal storages)

$\rightarrow$ **Increasingly complex systems** (multi-energy networks, 4/5G district heating, integrated PVT-HP systems) and **technologies** (heat pumps, PV/PVT, seasonal storages, (airborne) wind energy) require **advanced control techniques**.

What is Model Predictive Control (MPC)

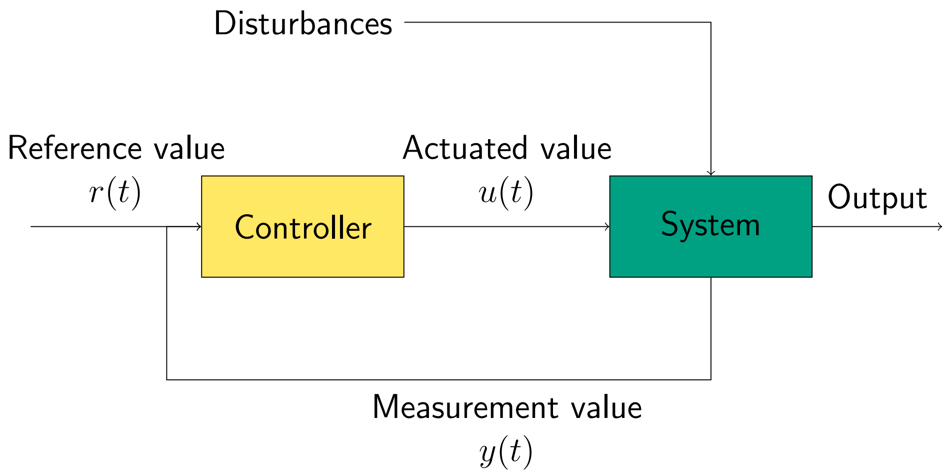
Feedback control

Feedback control

- Measure output of the process (”sensors”)
  $\rightarrow$ compare with set point (reference)
  $\rightarrow$ error is given to controller
  $\rightarrow$ controller generates manipulated variable (”actuator”) to control the process
- Traditional feedback control: PID (computes proportional, integral and differential terms) control

Model Predictive Control (MPC)

- Most relevant **higher control concept** in industry applications
- Real-time, repeated optimization to choose control
- Controller explicitly contains a model of the system (enables predictive operation & state constraints)



Block diagramm

MPC loop (receding horizon control)

- Model-based
- Predictive: **look into the future** using the **process model**
- Control: optimization-based feedback control

Algorithm:

1. Estimate current state $\hat{x}_k$.
2. Solve optimal control problem (OCP).
3. Apply only very first part of the computed optimal control trajectory.
4. Wait until new measurement becomes available at time $k+1$ and repeat.

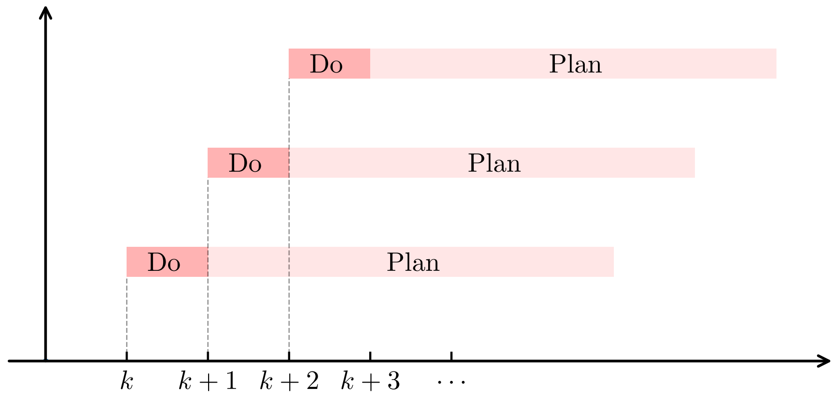
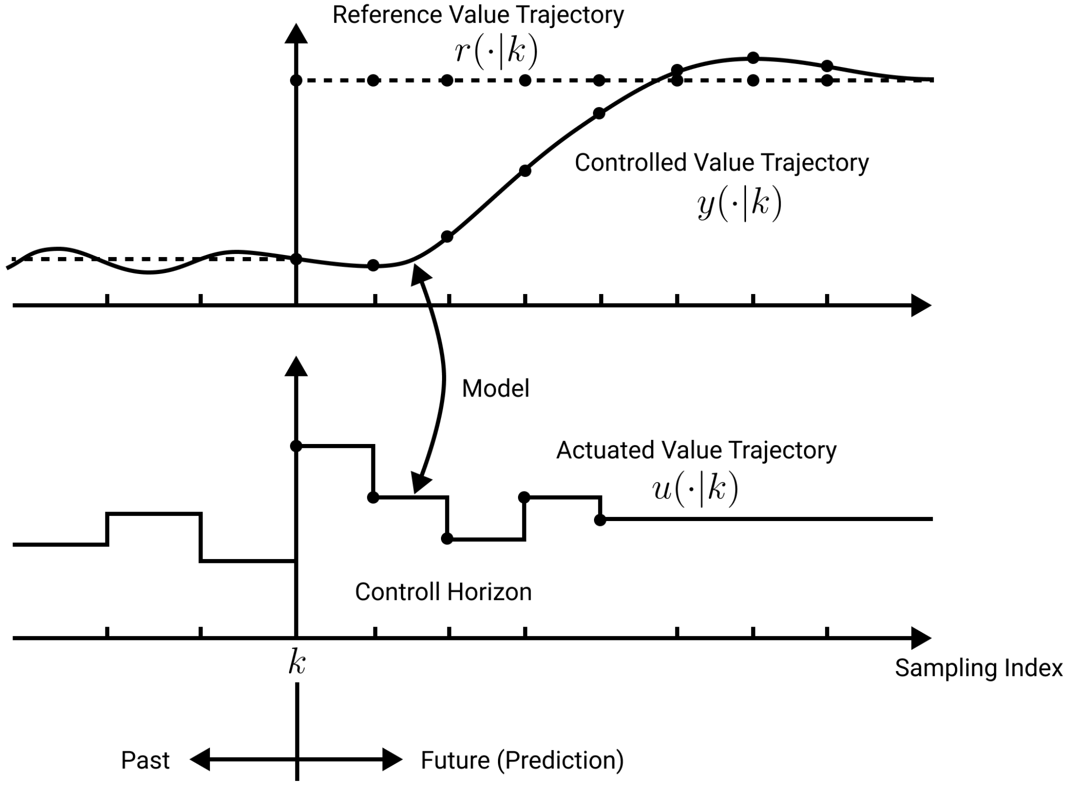


Figure: Receding horizon strategy introduces feedback in an MPC-based control loop.



Formulation of Continuous-Time Optimal Control Problems (OCP)

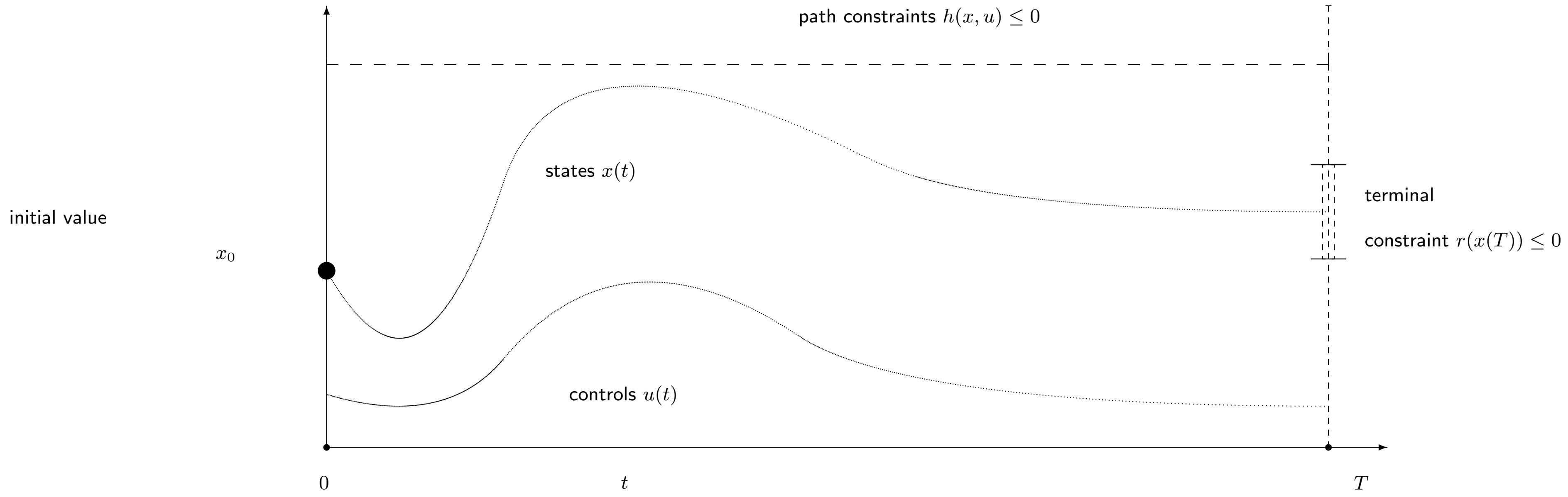
Problem is defined by

- Objective that is minimized (running + final cost)
- Internal system model to predict system behavior
- Constraints that have to be satisfied

$$\underset{x,\,u}{\text{minimize}} \quad \int_0^T L(x(t),u(t))\,dt \; + \; E\,(x(T))$$

subject to

$$\begin{aligned} x(0) &= x_0, && && \text{(fixed initial value)} \\ \dot{x}(t) &= f(x(t),u(t)), && t\in[0,T], && \text{(ODE model)} \\ h(x(t),u(t)) &\le 0, && t\in[0,T], && \text{(path constraints)} \end{aligned}$$



Different MPC variants (there exist many more)

Form of model equations:

Linear MPC (LMPC)

$$\begin{aligned}
\underset{\substack{x_0,\dots,x_N\\ u_0,\dots,u_{N-1}}}{\text{minimize}} \quad & \frac{1}{2}x_N^\top P_N x_N + \frac{1}{2}\sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k \\
\text{subject to} \quad & \\
x_0 &= \hat{x}_0, \\
x_{k+1} &= Ax_k + Bu_k, \qquad k = 0,\dots,N-1, \\
Cx + Du &\leq 0, \qquad ,
\end{aligned}$$

Nonlinear MPC (NMPC)

$$\begin{aligned}
\underset{x,\,u}{\text{minimize}} \quad & \int_0^T L(x(t),u(t))\,dt \;+\; E\,(x(T)) \\
\text{subject to} \quad & \\
x(0) &= x_0, \\
\dot{x}(t) &= f(x(t),u(t)), \qquad t \in [0,T], \\
h(x(t),u(t)) &\leq 0, \qquad t \in [0,T],
\end{aligned}$$

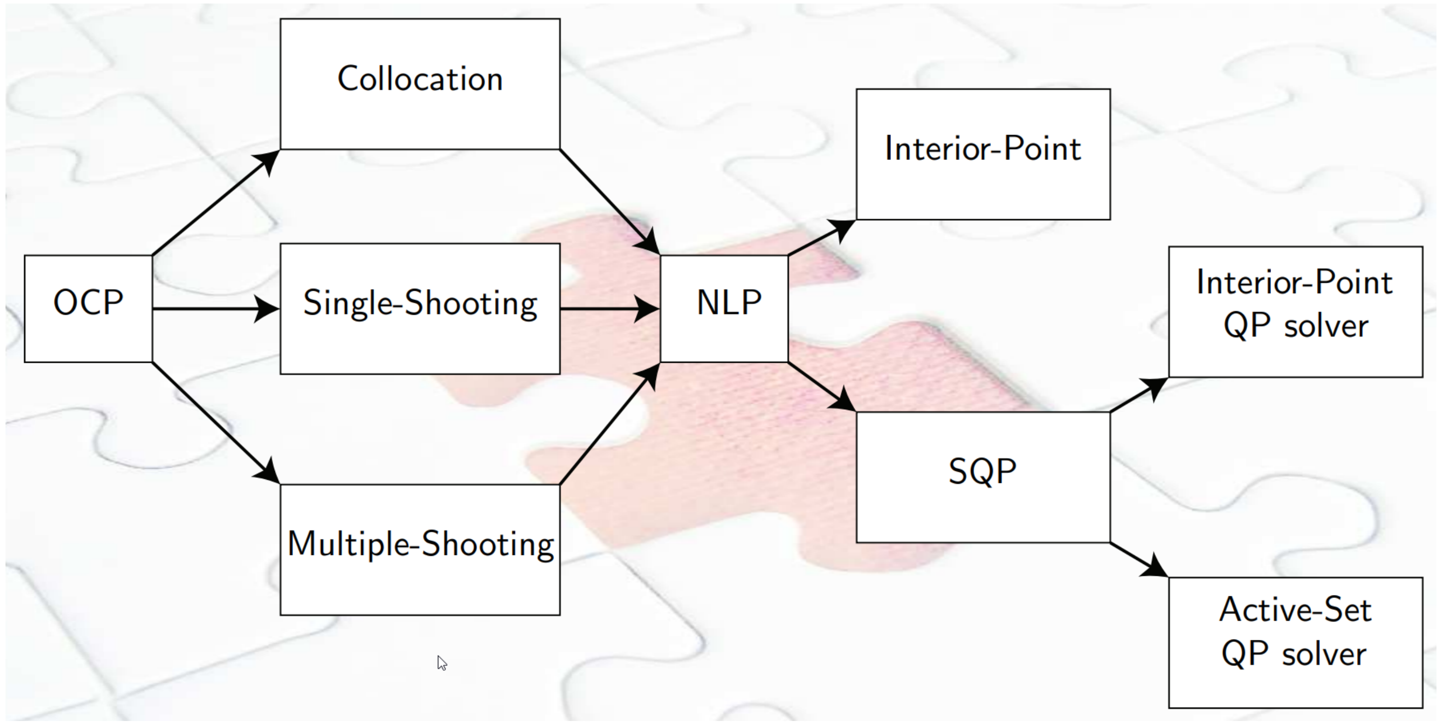
Objective function:

Tracking MPC: Follow a given reference trajectory, e.g., keep the room temperature near the set-point

Trajectory planning MPC: Compute optimal trajectory, which minimizes a cost function, e.g. optimal energy storage charging profile during the day

Solution methods for MPC

How to implement and solve OCPs?



- Use higher-level OCP solver, e.g. `rockit` to discretize and solve conitnuous-time OCP

$\longrightarrow$ Works for standard OCPs (without special dynamics (mixed-integer,...))

- Discretize OCP, e.g. using `CasADi`, and solve with specialized optimization solvers, e.g. `IPOPT` (NLP solver), `Bonmin` or `SCIP` (MILP solver)
- LMPC: specialized solution methods using QP solvers

Direct multiple shooting

- first step (in all direct methods): parameterization of the control function $u(t)$ on a fixed grid $0 = t_0 < t_1 < \ldots < t_N = T$, e.g. piecewise constant:

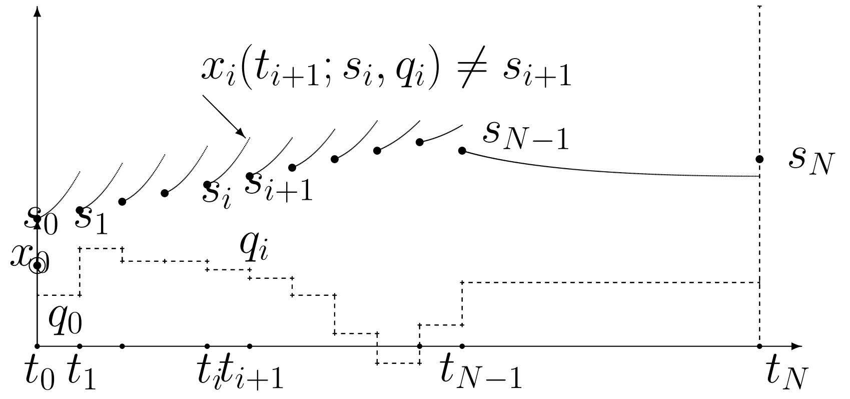
$$u(t; q) = q_i \quad \text{if} \quad t \in [t_i, t_{i+1}], i = 0, \ldots, N-1.$$

- shooting methods use embedded ODE solver to eliminate the continuous time dynamic system by forward simulation
- resulting discretized NLP with x obtained by forward simulation:

$$\underset{s,\,q}{\text{minimize}} \quad \sum_{i=0}^{N-1} l_i(s_i, q_i) \; + \; E\left(s_N\right)$$

subject to

$$\begin{array}{rcll}
x_0 - s_0 & = & 0, & & \text{(initial value)} \\
x_i(t_{i+1}; s_i, q_i) - s_{i+1} & = & 0, & i = 0, \ldots, N-1, & \text{(continuity)} \\
h(s_i, q_i) & \leq & 0, & i = 0, \ldots, N, & \text{(discretized path constraints)} \\
r\left(s_N\right) & \leq & 0. & & \text{(terminal constraints)}
\end{array}$$



Direct collocation

- parameterize the control function as before
- also discretize states by polynomial approximation of state trajectory on a finer collocation grid (collocation points)
- on each interval $[t_i, t_{i+1}]$ $\dot{x} = f(x, u_i)$ is approximated by $x(v_i, t) = \sum_{k=0}^{K} \underbrace{v_i^k}_{\text{coefficients}} \underbrace{P_i^k(t)}_{\text{polynomials}}$
- constraints are given by conditions on collocation grid
- resulting fully discretized large-scale NLP:

$$\underset{s,v,q}{\text{minimize}} \quad \sum_{i=0}^{N-1} l_i(s_i, v_i, q_i) \; + \; E\left(s_N\right)$$

subject to

$$\begin{array}{rcll}
s_0 - x_0 & = & 0, & & \text{(fixed initial value)} \\
c_i(s_i, v_i, q_i) & = & 0, & i = 0, \ldots, N-1, & \text{(collocation conditions)} \\
p_i(t_{i+1}; v_i) - s_{i+1} & = & 0, & i = 0, \ldots, N-1, & \text{(continuity conditions)} \\
h(s_i, q_i) & \leq & 0, & i = 0, \ldots, N-1, & \text{(discretized path constraints)} \\
r\left(s_N\right) & \leq & 0. & & \text{(terminal constraints)}
\end{array}$$

Software

CasADi at a glance

What is CasADi?

A general-purpsose software framework for quick, yet efficient, implementation of algorithms for numeric optimization

In particular

Facilitates the solution of optimal control problems (OCPs) using a variety of different methods by implementing algorithmic differentiation (AD) and defining interfaces to NL solvers

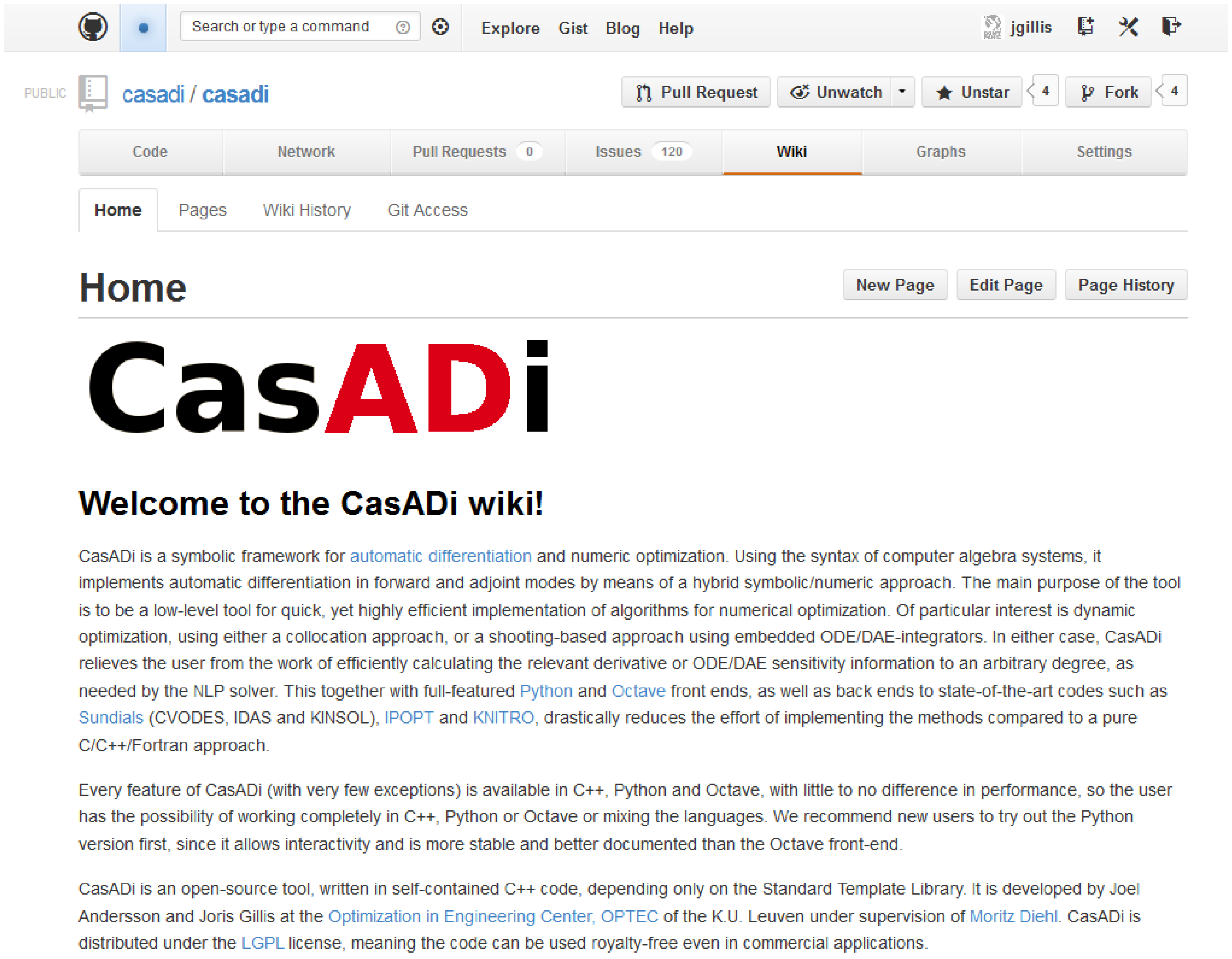
- *Facilitates*, not actually *solve* the OCPs
- Faciltates implementation of direct collocation and shooting methods for discretization

Source Code

- Free & open-source (LGPL)
- Use from C++ or Python
- Project started in December 2009 at KU Leuven by Andersson under the supervision of Prof. Moritz Diehl
  - Original motivation: Solve OCPs with models from *Modelica*
- Since 2012, a growing number of users – now a mature project

Where CasADi lives

`casadi.org` $\rightarrow$ `github.com`



Home

Welcome to the CasADi wiki!

CasADi is a symbolic framework for automatic differentiation and numeric optimization. Using the syntax of computer algebra systems, it implements automatic differentiation in forward and adjoint modes by means of a hybrid symbolic/numeric approach. The main purpose of the tool is to be a low-level tool for quick, yet highly efficient implementation of algorithms for numerical optimization. Of particular interest is dynamic optimization, using either a collocation approach, or a shooting-based approach using embedded ODE/DAE-integrators. In either case, CasADi relieves the user from the work of efficiently calculating the relevant derivative or ODE/DAE sensitivity information to an arbitrary degree, as needed by the NLP solver. This together with full-featured Python and Octave front ends, as well as back ends to state-of-the-art codes such as Sundials (CVODES, IDAS and KINSOL), IPOPT and KNITRO, drastically reduces the effort of implementing the methods compared to a pure C/C++/Fortran approach.

Every feature of CasADi (with very few exceptions) is available in C++, Python and Octave, with little to no difference in performance, so the user has the possibility of working completely in C++, Python or Octave or mixing the languages. We recommend new users to try out the Python version first, since it allows interactivity and is more stable and better documented than the Octave front-end.

CasADi is an open-source tool, written in self-contained C++ code, depending only on the Standard Template Library. It is developed by Joel Andersson and Joris Gillis at the Optimization in Engineering Center, OPTEC of the K.U. Leuven under supervision of Moritz Diehl. CasADi is distributed under the LGPL license, meaning the code can be used royalty-free even in commercial applications.

More about CasADi

- Central feature I: general-purpose implementation of AD
- Central feature II: solve standard problems conveniently
  - QPs, NLPs, Rootfinding problems, Initial-value problems (IVP) in ODE/DAE
  - Provides convenient interfaces to mixed-integer and other specialized solvers, e.g. BONMIN for solving MINLP, which is disctributed with CasADi

Matrices in CasAD

- CasADi is *everything-is-a-matrix* (cf. MATLAB)
- All matrices are *sparse*
- Syntax is MATLAB inspired

`SX.sym("x",2,3)`	`2-by-3 symbolic primitive`
`SX.zeros(4,5)`	`dense 4-by-5 matrix with all zeros`
`SX.sparse(4,5)`	`sparse (empty) 4-by-5 matrix`
`SX.eye(4)`	`4-by-4 identity matrix`

Symbolic framework of CasADi

What you need to know

- CasADi allows you to symbolic expressions using syntax similar to e.g. *Symbolic Math Toolbox* for MATLAB or *SymPy*.

```
from casadi import *
x = SX.sym("x")          Variable x with display name "x"
f = sqrt(x**2 + 10)
g = sin(x)
```

$f = \sqrt{x^2+10}$

$g = \sin(x)$

- These functions are then used to define *functions* . . .

```
F = SXFunction([x],[f,g])   Defines
F.init()
```

$$F : \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R}\times\mathbb{R} \\ (x) & \mapsto & (f,g) \end{array}$$

- . . . that can e.g. be automatically differentiated using *algorithmic differentiation* (AD)

```
J = F.jacobian()   Defines
```

$$J : \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R}\times\mathbb{R}\times\mathbb{R} \\ (x) & \mapsto & \left(\frac{\partial f}{\partial x}, f, g\right) \end{array}$$

Two symbolic types with (almost) the same syntax

- SX: *Expression graph* with scalar-valued operations
- Low overhead, for simple functions

```
x = SX.sym("x",2,2)
f = sin(x**2 + 10)
print x, f
```

$$\begin{bmatrix} x_0 & x_2 \\ x_1 & x_3 \end{bmatrix}, \begin{bmatrix} \sin x_0^2 + 10 & \sin x_2^2 + 10 \\ \sin x_1^2 + 10 & \sin x_3^2 + 10 \end{bmatrix}$$

- MX: *Expression graph* with matrix-valued operations
- Larger overhead, but more generic

```
x = MX.sym("x",2,3)
f = sin(x**2 + 10)
print f
```

$$x, \quad \sin x^2 + 10$$

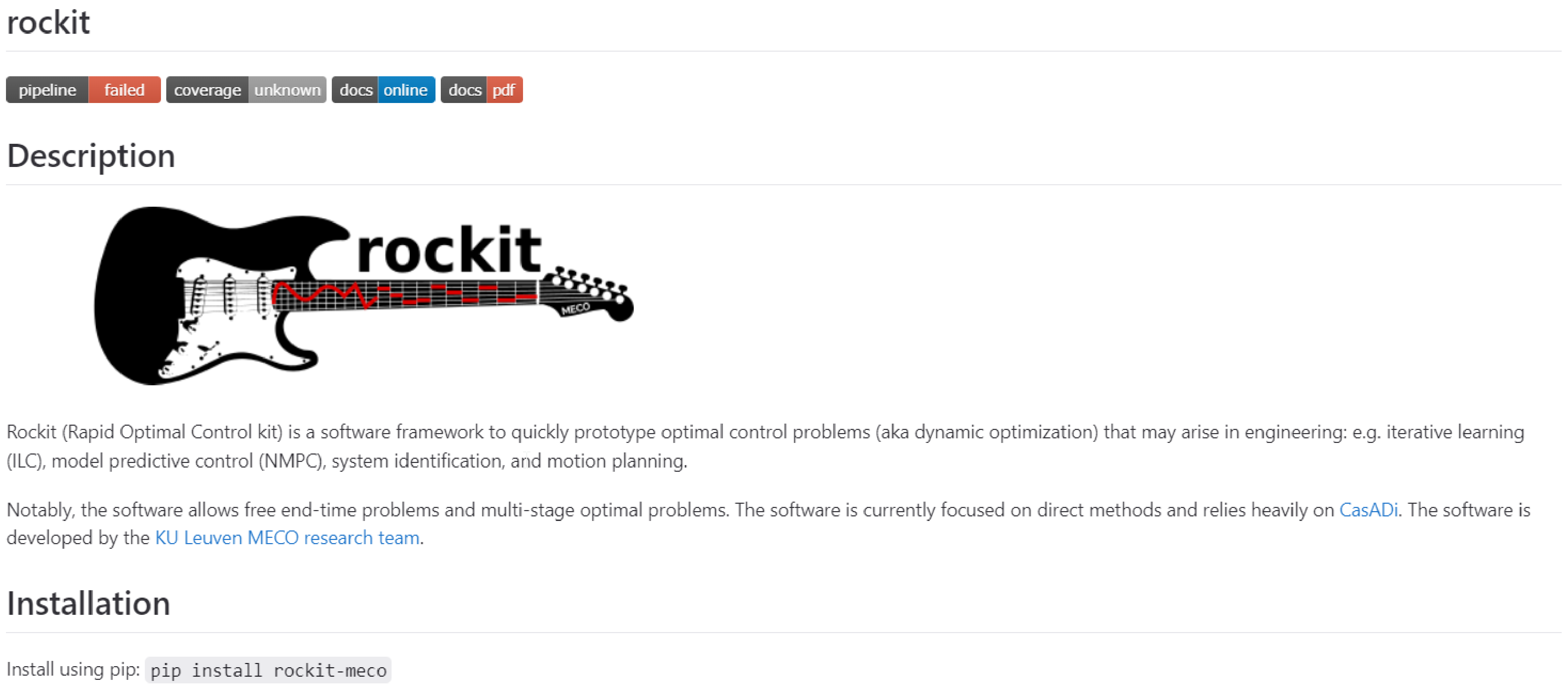
Why?

By mixing, construct expressions (functions) that are both fast and generic

rockit

An Optimal Control Problem abstraction class, built on top of CasADi

`https://gitlab.kuleuven.be/meco-software/rockit`



rockit

pipeline failed | coverage unknown | docs online | docs pdf

Description

Rockit (Rapid Optimal Control kit) is a software framework to quickly prototype optimal control problems (aka dynamic optimization) that may arise in engineering: e.g. iterative learning (ILC), model predictive control (NMPC), system identification, and motion planning.

Notably, the software allows free end-time problems and multi-stage optimal problems. The software is currently focused on direct methods and relies heavily on CasADi. The software is developed by the KU Leuven MECO research team.

Installation

Install using pip: `pip install rockit-meco`

Example - How to solve OCPs with rockit?

```
# Start an OCP environment with a time horizon of 10s
ocp = Ocp(t0=0, T=10)

# Define two scalar states
x1 = ocp.state()
x2 = ocp.state()

# Define one piecewise constant control input
u = ocp.control()

# Specify differential equations for states
ocp.set_der(x1, (1 - x2**2) * x1 - x2 + u)
ocp.set_der(x2, x1)

# Lagrange objective term: signals in an integrand
ocp.add_objective(ocp.integral(x1**2 + x2**2 + u**2))
# Mayer objective term: signals evaluated at t_f = t0_+T
ocp.add_objective(ocp.at_tf(x1**2))

# Path constraints
ocp.subject_to(x1 >= -0.25)
ocp.subject_to(-1 <= (u <= 1 ))

# Boundary constraints
ocp.subject_to(ocp.at_t0(x1) == 0)
ocp.subject_to(ocp.at_t0(x2) == 1)
```

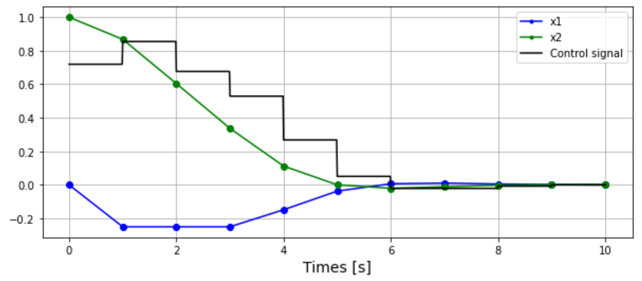
```
# Pick an NLP solver backend
ocp.solver('ipopt')

# Pick a solution method
method = MultipleShooting(N=10, intg='rk')
ocp.method(method)

# Solve
sol = ocp.solve()
```

$$\min_{x,u} \int_0^{10} x_1(t)^2 + x_2(t)^2 + u(t)^2 \, dt \quad + \quad x_1(10)$$

$$\begin{aligned} \text{s.t.} \quad & \dot{x_1} = (1 - x_2^2)x_1 - x_2 + u \\ & \dot{x_2} = x_1 \\ & x_1(0) = 0 \\ & x_2(0) = 1 \\ & x_1(t) \geq 0.25 \\ & -1 \leq u(t) \leq 1 \end{aligned}$$


Times [s]
x1
x2
Control signal