universität freiburg

# Numerical Optimal Control: Smooth, Nonsmooth, and Robust

Moritz Diehl[1]

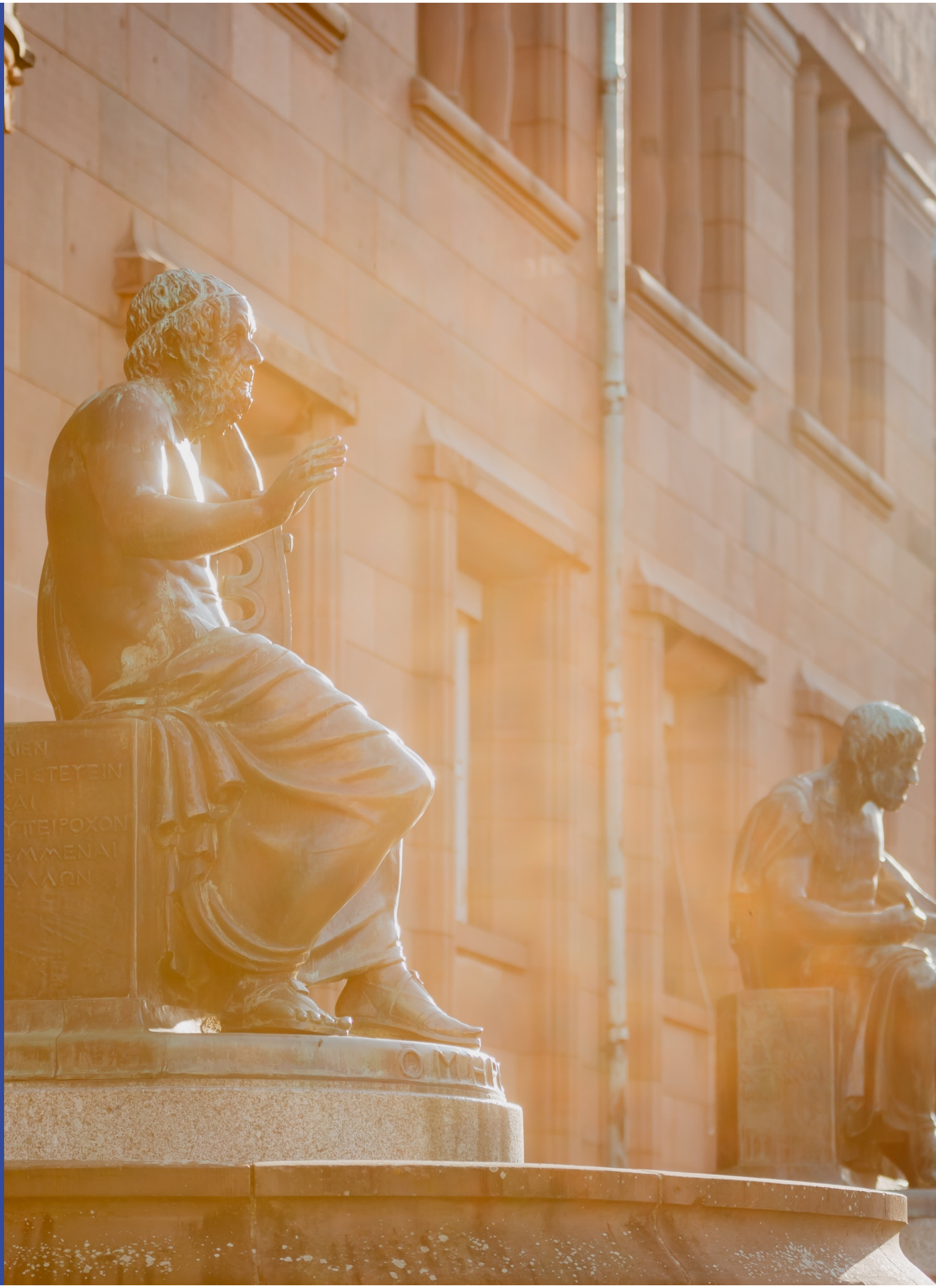
joint work with Armin Nurkanovic[1], Christian Dietz[1,2], Anton Pozharskiy[1], Gianluca Frison[1,3] Sebastian Albrecht[2]

[1] Department of Microsystems Engineering and Department of Mathematics, University of Freiburg, Germany

[2] Siemens Foundational Technology, Munich, Germany

[3] MOSEK ApS, Denmark

NTNU Trondheim, February 16, 2026



# Continuous-Time Optimal Control Problems (OCP)

## Continuous-Time OCP with Ordinary Differential Equation (ODE) Constraints

$$\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \textstyle\int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}$$

# Continuous-Time Optimal Control Problems (OCP)

## Continuous-Time OCP with Ordinary Differential Equation (ODE) Constraints

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& \quad 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& \quad 0 \geq r(x(T))
\end{aligned}
$$

Can in most applications assume convexity of all ”outer” problem functions: $L_{\mathrm{c}}, E, h, r$.

# Three Levels of Difficulty in Continuous-Time OCP

### Continuous-Time OCP

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)), \; t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

Three levels of difficulty:

# Three Levels of Difficulty in Continuous-Time OCP

## Continuous-Time OCP

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\ \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\ t \in [0,T] \\ & 0 \geq r(x(T)) \end{aligned}$$

Three levels of difficulty:

(a) Linear ODE: $f(x,u) = Ax + Bu$

# Three Levels of Difficulty in Continuous-Time OCP

## Continuous-Time OCP

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)), \; t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

Three levels of difficulty:

(a) Linear ODE: $f(x,u) = Ax + Bu$

(b) Nonlinear smooth ODE: $f \in \mathcal{C}^1$

# Three Levels of Difficulty in Continuous-Time OCP

### Continuous-Time OCP

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\ \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\ t \in [0,T] \\ & 0 \geq r(x(T)) \end{aligned}$$

Three levels of difficulty:

(a) Linear ODE: $f(x,u) = Ax + Bu$

(b) Nonlinear smooth ODE: $f \in \mathcal{C}^1$

(c) **Nonsmooth Dynamics (NSD):**

- $f$ not differentiable (NSD1),
- $f$ not continuous (NSD2), or even
- $f$ not finite valued, discontinuous state $x(t)$ (NSD3)

# Three Levels of Difficulty in Continuous-Time OCP

**Continuous-Time OCP**

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\ \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\ t \in [0,T] \\ & 0 \geq r(x(T)) \end{aligned}$$

Three levels of difficulty:

(a) Linear ODE: $f(x,u) = Ax + Bu$

(b) Nonlinear smooth ODE: $f \in \mathcal{C}^1$

(c) **Nonsmooth Dynamics (NSD):**

- $f$ not differentiable (NSD1),
- $f$ not continuous (NSD2), or even
- $f$ not finite valued, discontinuous state $x(t)$ (NSD3)

First focus on smooth cases (a) and (b).

# Three Levels of Difficulty in Continuous-Time OCP

**Continuous-Time OCP**

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

Three levels of difficulty:

(a) Linear ODE: $f(x,u) = Ax + Bu$

(b) Nonlinear smooth ODE: $f \in \mathcal{C}^1$

First focus on smooth cases (a) and (b).

# Recall: Runge-Kutta Discretization for Smooth Systems

## Ordinary Differential Equation (ODE)

$$\dot{x}(t) = \underbrace{f(x(t), u(t))}_{=:v(t)}$$

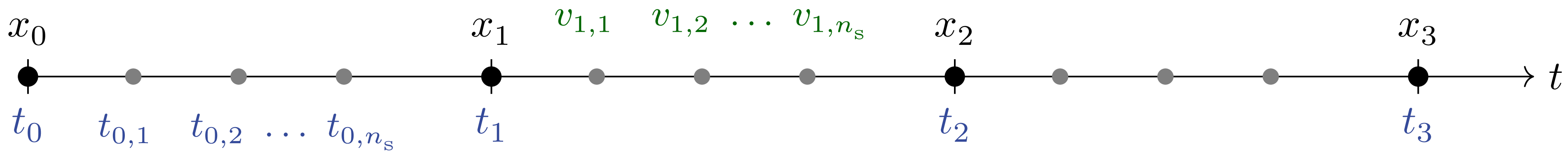
## Initial Value Problem (IVP)

$$\begin{aligned} x(0) &= \bar{x}_0 \\ v(t) &= f(x(t), u(t)) \\ \dot{x}(t) &= v(t) \\ & \quad t \in [0, T] \end{aligned}$$

## Discretization: $N$ Runge-Kutta steps of each $n_s$ stages

$$\begin{aligned} x_{0,0} &= \bar{x}_0, \qquad\qquad \Delta t = \tfrac{T}{N} \\ v_{k,j} &= f(x_{k,j}, u_k) \\ x_{k,j} &= x_{k,0} + \Delta t \textstyle\sum_{n=1}^{n_s} a_{jn} v_{k,n} \\ x_{k+1,0} &= x_{k,0} + \Delta t \textstyle\sum_{n=1}^{n_s} b_n v_{k,n} \\ & j = 1, \dots, n_s, \quad k = 0, \dots, N-1 \end{aligned}$$

For fixed controls and initial value: square system with $n_x + N(2n_s + 1)n_x$ unknowns, implicitly defined via $n_x + N(2n_s + 1)n_x$ equations.
(trivial eliminations in case of explicit RK methods)



# Direct Methods Transform OCP into Nonlinear Program (NLP)

### Continuous time OCP

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\ \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\ t \in [0,T] \\ & 0 \geq r(x(T)) \end{aligned}$$

- Direct methods "first discretize, then optimize"

# Direct Methods Transform OCP into Nonlinear Program (NLP)

## Continuous time OCP

$$\begin{aligned}\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T))\\ \text{s.t.} \quad & x(0) = \bar{x}_0\\ & \dot{x}(t) = f(x(t),u(t))\\ & 0 \geq h(x(t),u(t)),\ t \in [0,T]\\ & 0 \geq r(x(T))\end{aligned}$$

- Direct methods "first discretize, then optimize"

1. Parameterize controls, e.g. $u(t) = u_n, t \in [t_n, t_{n+1}]$.

# Direct Methods Transform OCP into Nonlinear Program (NLP)

### Continuous time OCP

$$\min_{x(\cdot),u(\cdot)} \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T))$$

$$\begin{aligned} \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\ t \in [0,T] \\ & 0 \geq r(x(T)) \end{aligned}$$

- Direct methods "first discretize, then optimize"

1. Parameterize controls, e.g. $u(t) = u_n, t \in [t_n, t_{n+1}]$.
2. Discretize cost and dynamics

$$L_\mathrm{d}(x_n, z_k, u_n) \approx \int_{t_n}^{t_{n+1}} L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t$$

Replace $\dot{x} = f(x,u)$ by

$$\begin{aligned} x_{n+1} &= \phi_f(x_n, z_n, u_n) \\ 0 &= \phi_\mathrm{int}(x_n, z_n, u_n) \end{aligned}$$

# Direct Methods Transform OCP into Nonlinear Program (NLP)

**Continuous time OCP**

$$\min_{x(\cdot),u(\cdot)} \quad \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T))$$

$$\begin{aligned} \text{s.t.} \quad x(0) &= \bar{x}_0 \\ \dot{x}(t) &= f(x(t),u(t)) \\ 0 &\geq h(x(t),u(t)),\ t \in [0,T] \\ 0 &\geq r(x(T)) \end{aligned}$$

- Direct methods "first discretize, then optimize"

1. Parameterize controls, e.g. $u(t) = u_n, t \in [t_n, t_{n+1}]$.
2. Discretize cost and dynamics

$$L_\mathrm{d}(x_n, z_k, u_n) \approx \int_{t_n}^{t_{n+1}} L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t$$

Replace $\dot{x} = f(x,u)$ by

$$x_{n+1} = \phi_f(x_n, z_n, u_n)$$

$$0 = \phi_\mathrm{int}(x_n, z_n, u_n)$$

3. Also discretize path constraints

$$0 \geq \phi_h(x_n, z_n, u_n),\ n = 0, \ldots N-1.$$

# Direct Methods Transform OCP into Nonlinear Program (NLP)

## Continuous time OCP

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)), \ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

- Direct methods "first discretize, then optimize"

## Discrete time OCP (an NLP)

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z},\mathbf{u}} \quad & \textstyle\sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k,z_k,u_k) + E(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0 \\
& x_{n+1} = \phi_f(x_n,z_n,u_n) \\
& 0 = \phi_{\mathrm{int}}(x_n,z_n,u_n) \\
& 0 \geq \phi_h(x_n,z_n,u_n), \ n = 0,\dots,N-1 \\
& 0 \geq r(x_N)
\end{aligned}
$$

Variables $\mathbf{x} = (x_0,\dots,x_N)$, $\mathbf{z} = (z_0,\dots,z_N)$ and $\mathbf{u} = (u_0,\dots,u_{N-1})$.
Here, $\mathbf{z}$ are the intermediate variables of the integrator (e.g. Runge-Kutta)

# Simplest Direct Transcription: Single Step Explicit Euler

(not recommended in practice, other Runge-Kutta methods are much more efficient)

## Continuous time OCP

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\ \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\ t \in [0,T] \\ & 0 \geq r(x(T)) \end{aligned}$$

- Direct methods: first discretize, then optimize

## Single Step Explicit Euler NLP, with $\Delta t = \frac{T}{N}$

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u}} \quad & \textstyle\sum_{k=0}^{N-1} L_{\mathrm{c}}(x_k,u_k)\Delta t + E(x_N) \\ \text{s.t.} \quad & x_0 = \bar{x}_0 \\ & x_{n+1} = x_n + f(x_n,u_n)\Delta t \\ & 0 \geq h(x_n,u_n),\ n = 0,\ldots,N-1 \\ & 0 \geq r(x_N) \end{aligned}$$

Variables $\mathbf{x} = (x_0,\ldots,x_N)$ and $\mathbf{u} = (u_0,\ldots,u_{N-1})$.
(single step explicit Euler has no internal integrator variables $\mathbf{z}$)

# 2nd Simplest Direct Transcription: Midpoint Rule

**Continuous time OCP**

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)), \;\; t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

**Midpoint Rule NLP, with $\Delta t = \frac{T}{N}$**

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z},\mathbf{u}} \quad & \sum_{k=0}^{N-1} L_\mathrm{c}(z_k,u_k)\Delta t + E(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0 \\
& x_{n+1} = x_n + f(z_n,u_n)\Delta t \\
& 0 = z_n - \frac{x_n + x_{n+1}}{2} \\
& 0 \geq h(z_n,u_n), \;\; n = 0,\ldots,N-1 \\
& 0 \geq r(x_N)
\end{aligned}
$$

Variables $\mathbf{x} = (x_0,\ldots,x_N)$, $\mathbf{z} = (z_0,\ldots,z_{N-1})$, and $\mathbf{u} = (u_0,\ldots,u_{N-1})$.

# Sparse NLP resulting from direct transcription

**Discrete time OCP (an NLP)**

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z},\mathbf{u}} \ & \textstyle\sum_{k=0}^{N-1} L_{\mathrm{d}}(x_k, z_n, u_k) + E(x_N) \\ \text{s.t.} \quad & x_0 = \bar{x}_0 \\ & x_{n+1} = \phi_f(x_n, z_n, u_n) \\ & 0 = \phi_{\mathrm{int}}(x_n, z_n, u_n) \\ & 0 \geq \phi_h(x_n, z_n, u_n), \ n = 0, \dots, N-1 \\ & 0 \geq r(x_N) \end{aligned}$$

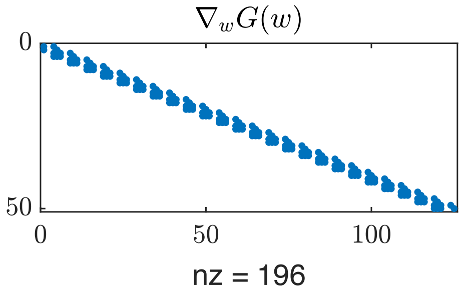
Variables $w = (\mathbf{x}, \mathbf{z}, \mathbf{u})$

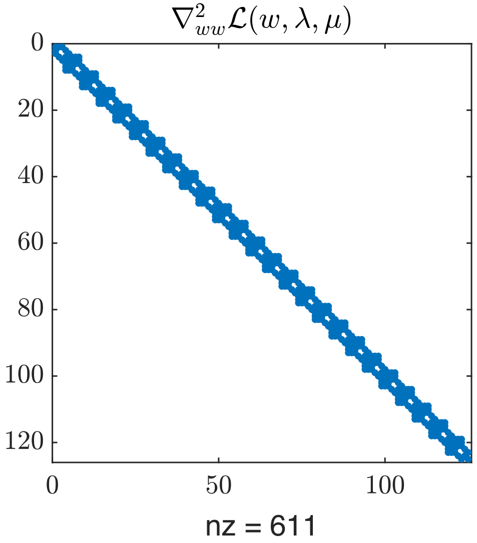
**Nonlinear Program (NLP)**

$$\begin{aligned} \min_{w \in \mathbb{R}^{n_x}} \ & F(w) \\ \text{s.t.} \ & G(w) = 0 \\ & H(w) \geq 0 \end{aligned}$$

Large and sparse NLP

# Sparse NLP resulting from direct transcription





Variables $w = (\mathbf{x}, \mathbf{z}, \mathbf{u})$

## Nonlinear Program (NLP)

$$\begin{aligned} \min_{w\in\mathbb{R}^{n_x}} \quad & F(w) \\ \text{s.t.} \quad & G(w) = 0 \\ & H(w) \geq 0 \end{aligned}$$

Large and sparse NLP

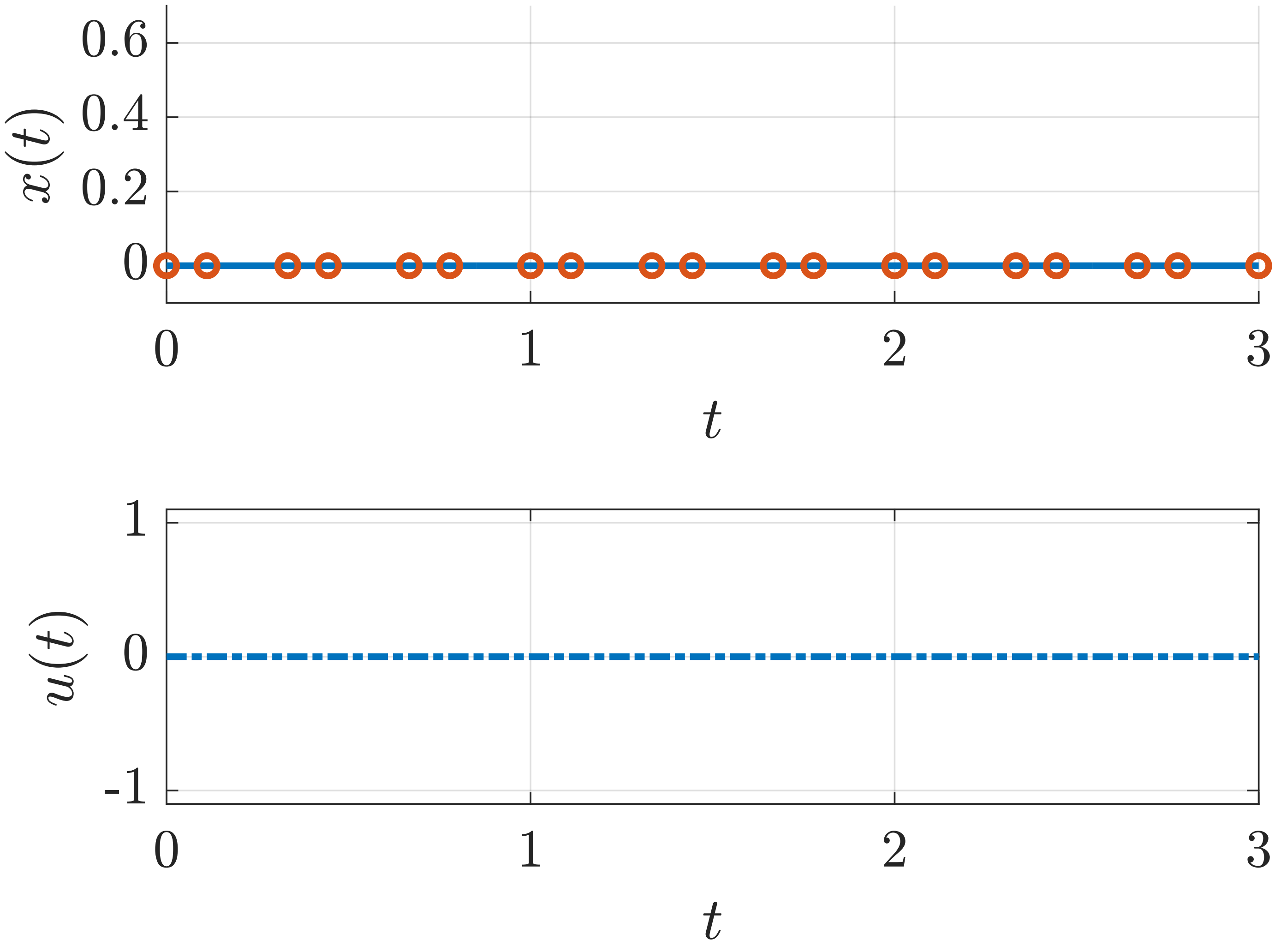
# Illustrative example of direct collocation with Newton-type optimization

### Illustrative nonlinear optimal control problem (with one state and one control)

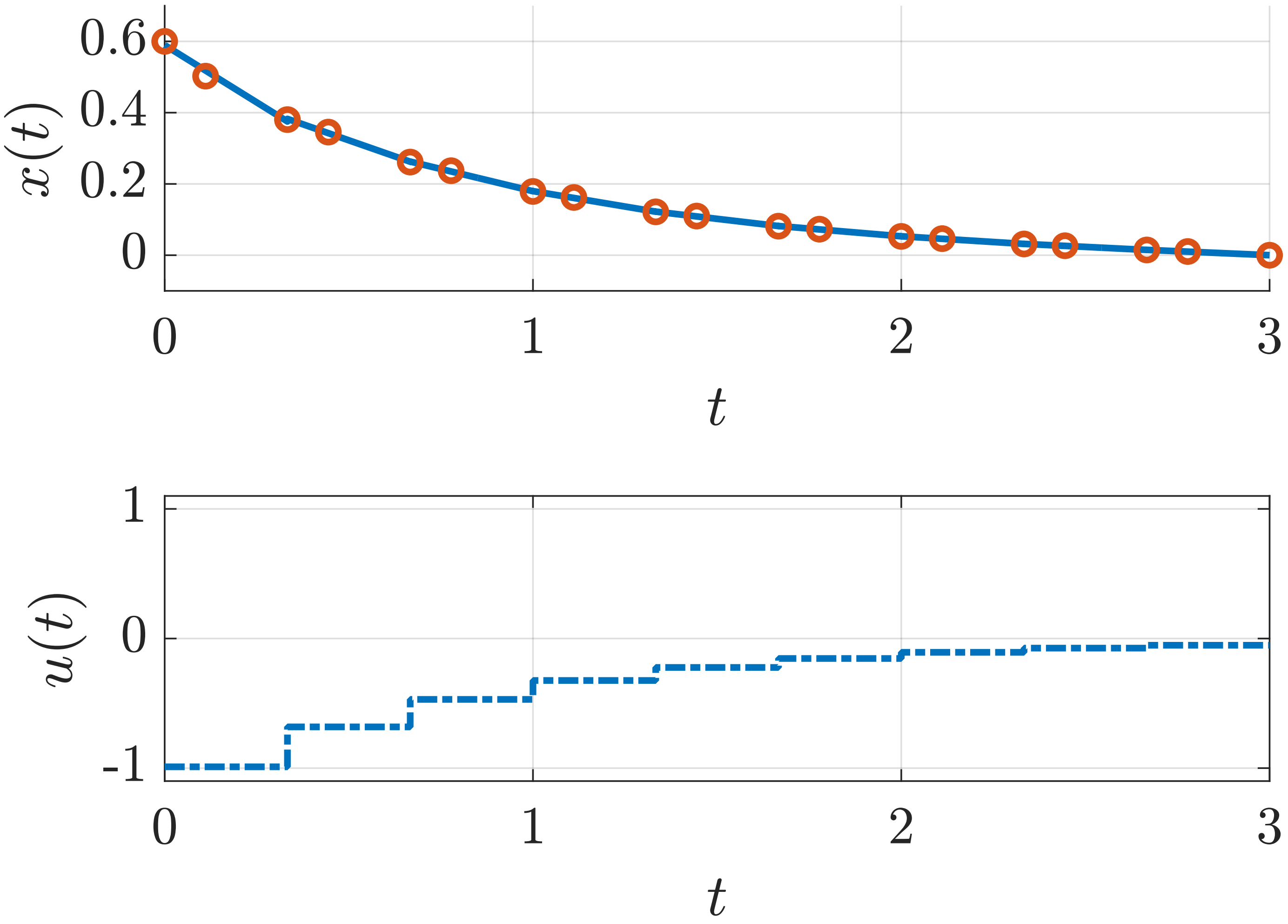
$$
\begin{aligned}
\underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad & \int_0^3 x(t)^2 + u(t)^2 \; dt \\
\text{subject to} \quad & \\
x(0) &= \bar{x}_0 && && \text{(initial value, } \bar{x}_0 = 0.6\text{)} \\
\dot{x} &= (1+x)x + u, && && \text{(ODE model)} \\
-1 \le u(t) &\le 1, && t \in [0,3] && \text{(bounds)} \\
x(3) &= 0 && && \text{(terminal constraint)}
\end{aligned}
$$

- choose $N = 9$ equal intervals and Radau-IIA collocation with $n_s = 2$ stages
- obtain nonlinear program with $n_x + (2n_s + 1)Nn_x + Nn_u$ variables
- initialize with zeros everywhere, solve with CasADi and Ipopt (interior point)

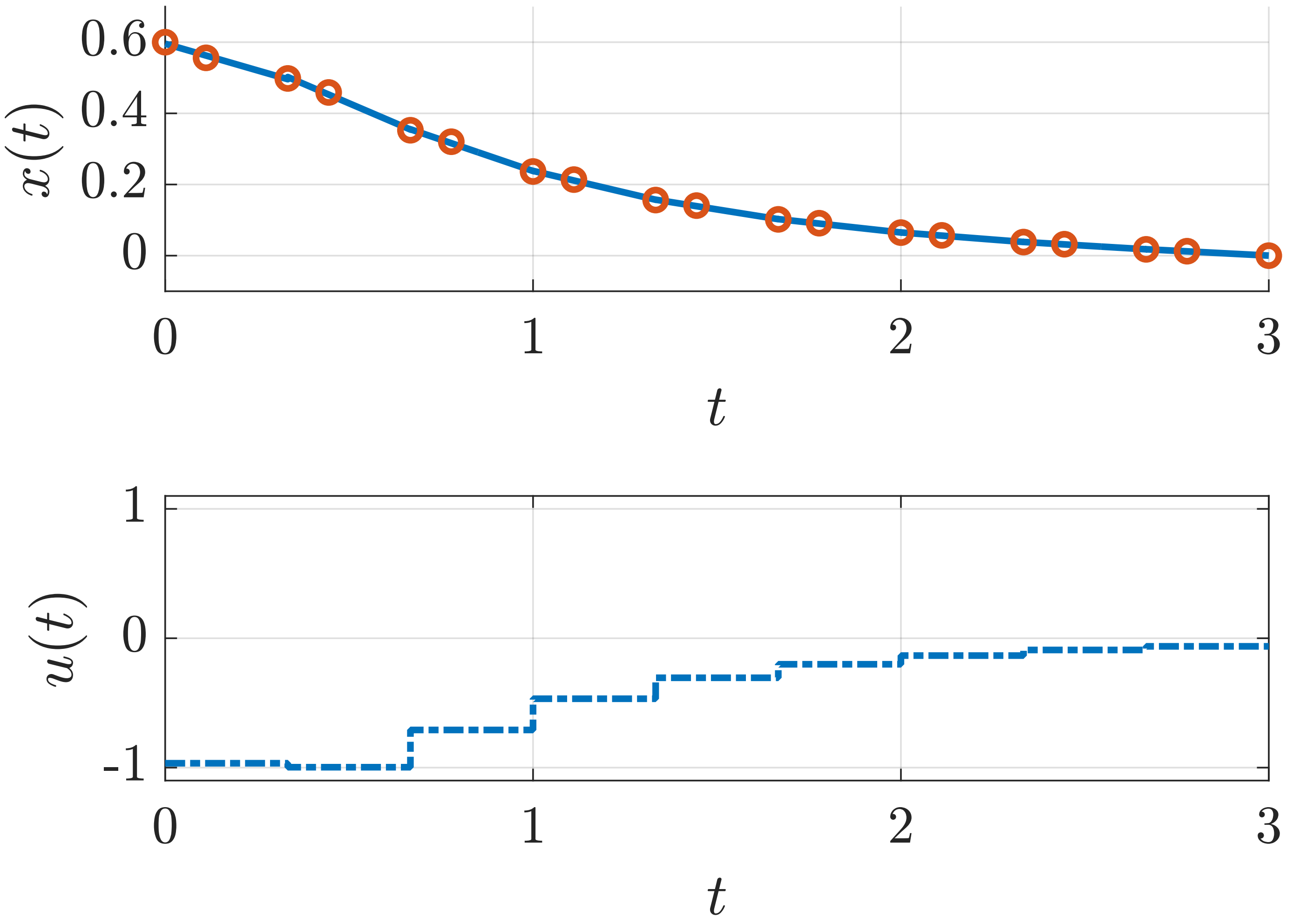
# Illustrative example: Initialization



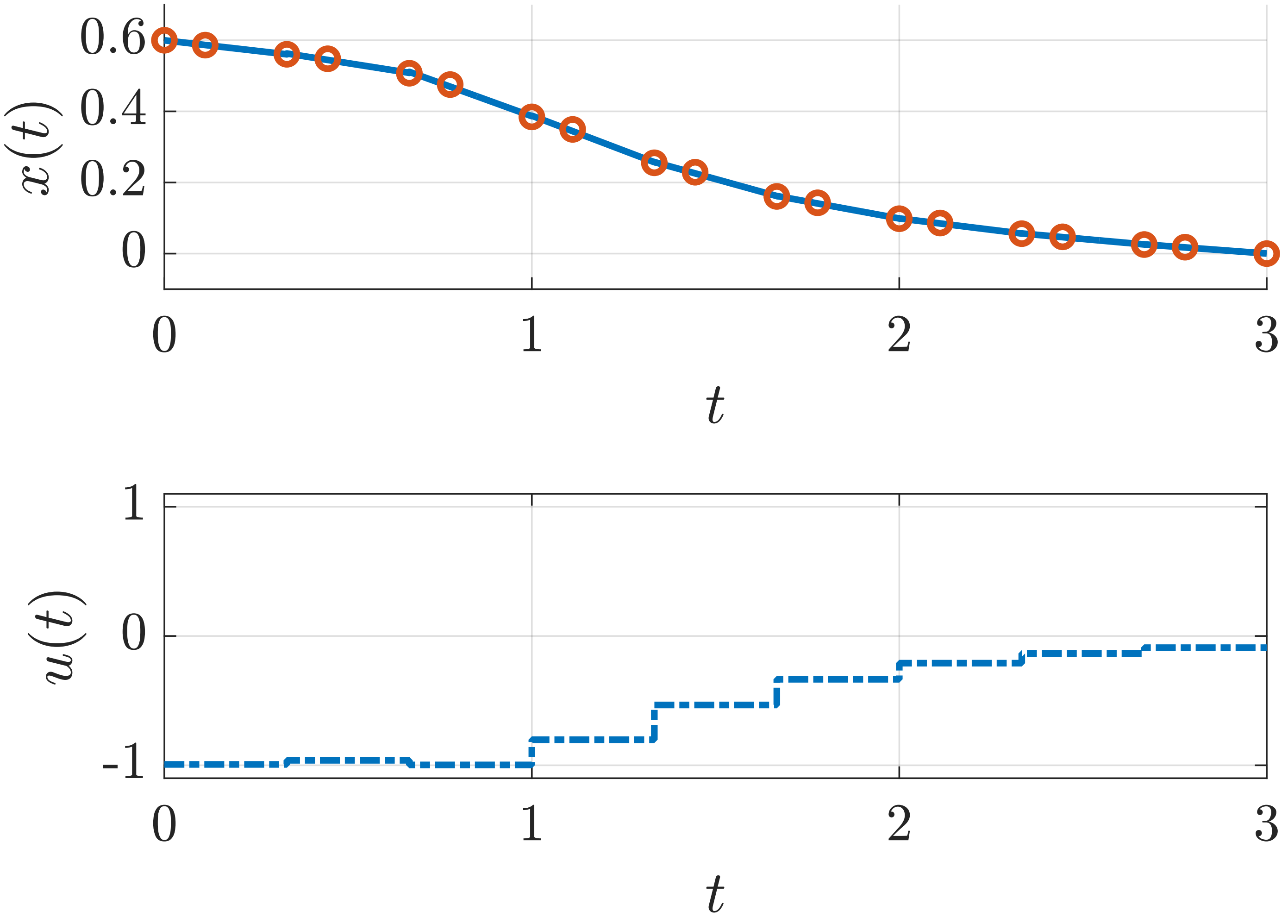
# Illustrative example: First Iterate



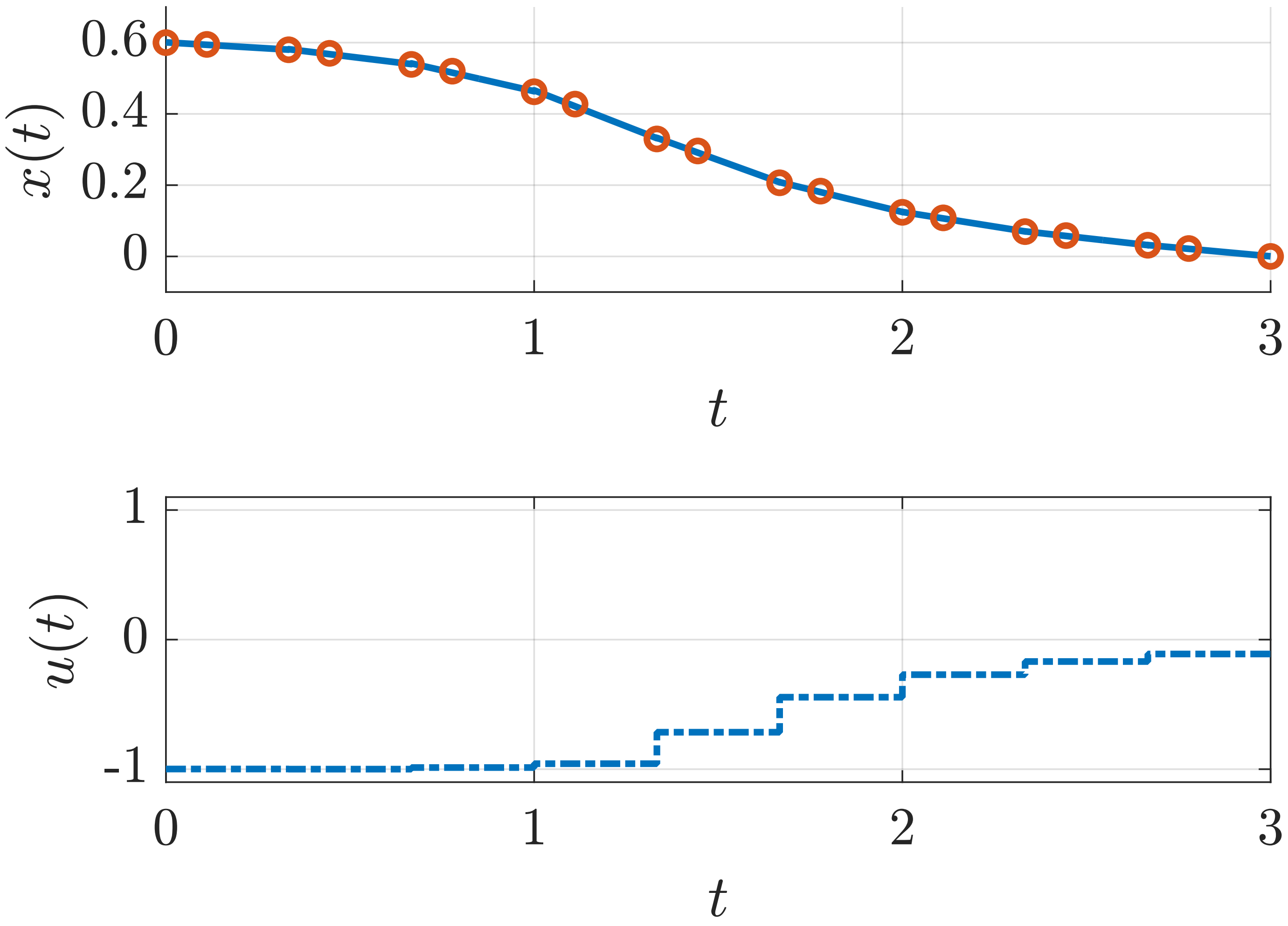
# Illustrative example: Second Iterate



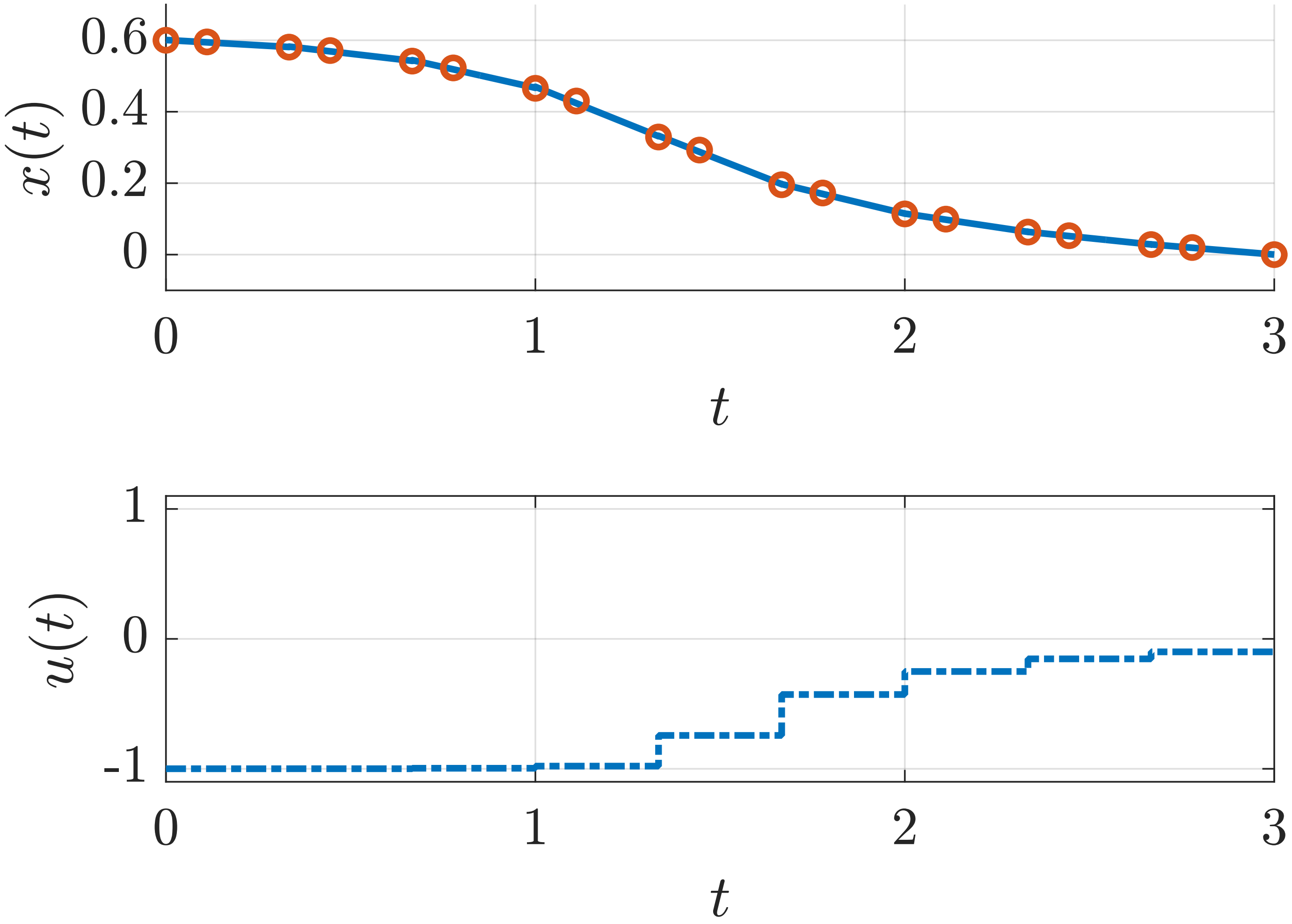
# Illustrative example: Third Iterate



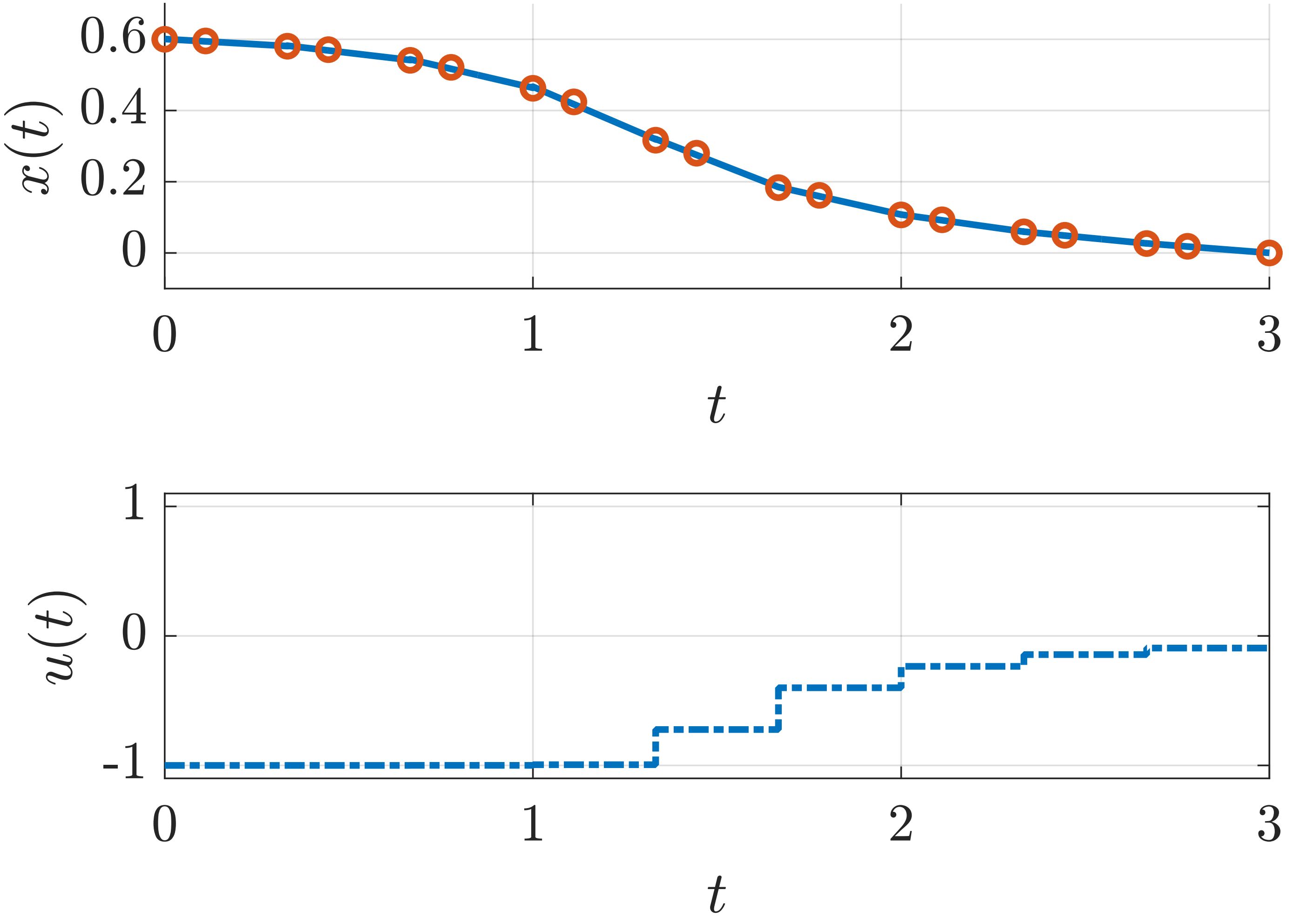
# Illustrative example: Fourth Iterate



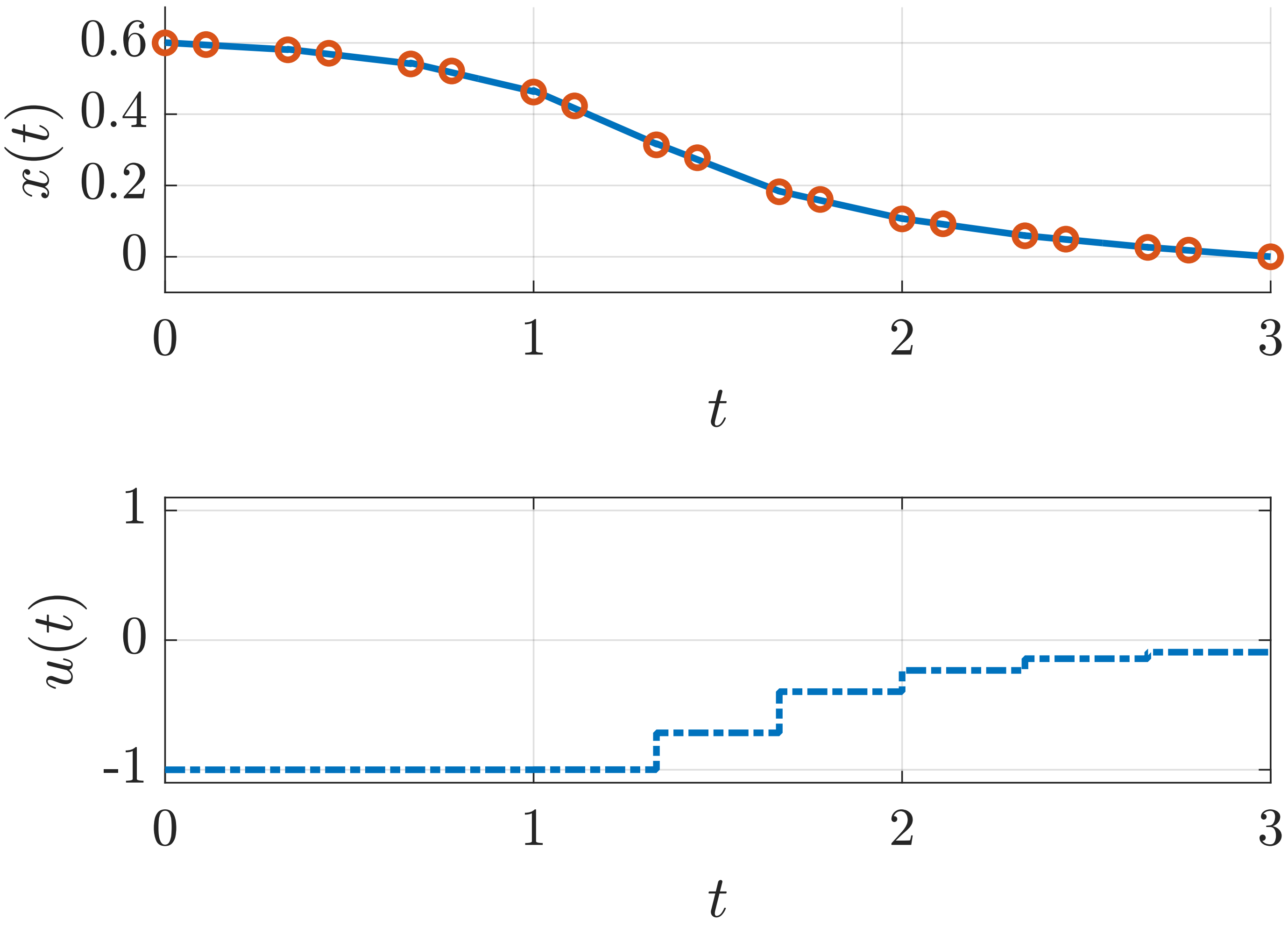
# Illustrative example: Fifth Iterate



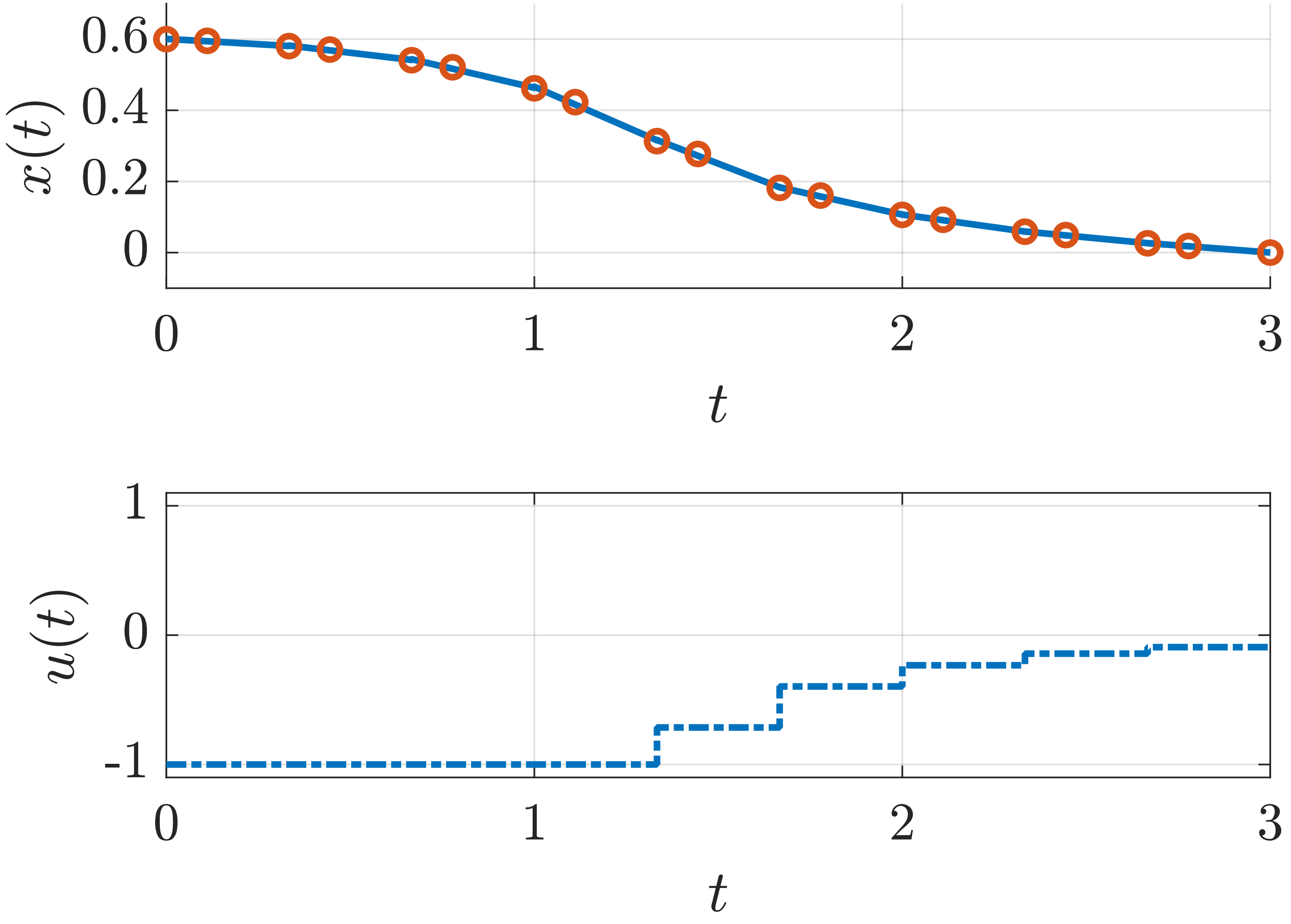
# Illustrative example: Sixth Iterate



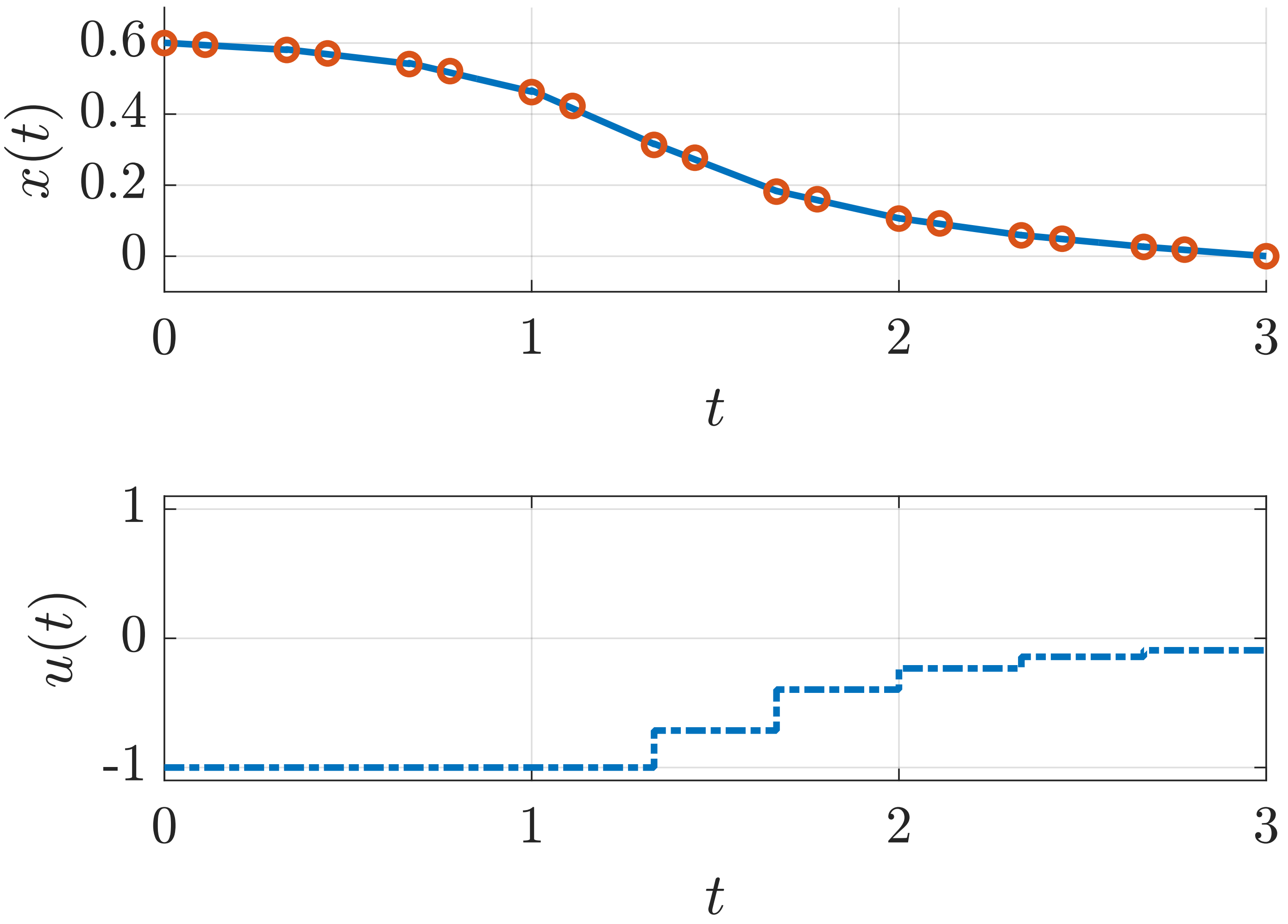
# Illustrative example: Seventh Iterate



# Illustrative example: Eighth Iterate



# Illustrative example: Solution after Nine Newton-type Iterations



# More Complex Example: Power Optimal Trajectories in Airborne Wind Energy (AWE)

formulated and solved daily by practitioners using open-source python package “AWEBox” [De Schutter et al. 2023]

For simple plane attached to a tether:

- 20 differential states (3+3 trans, 9+3 rotation, 1+1 tether)
- 1 algebraic state (tether force)
- 8 invariants (6 rotation, 2 due to tether constraint)
- 3 control inputs (aileron, elevator, tether length)

$$\text{Translational:} \quad \begin{bmatrix} m & 0 & 0 & x \\ 0 & m & 0 & y \\ 0 & 0 & m & z \\ x & y & z & 0 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \lambda \end{bmatrix} = \begin{bmatrix} F_x + m\left(\dot{\delta}^2 r_A + \dot{\delta}^2 x + 2\dot{\delta}\dot{y} + \ddot{\delta}y\right) \\ F_y + m\left(y\dot{\delta}^2 - 2\dot{x}\dot{\delta} - \ddot{\delta}(rA + x)\right) \\ F_z - gm \\ -\dot{x}^2 - \dot{y}^2 - \dot{z}^2 \end{bmatrix}$$

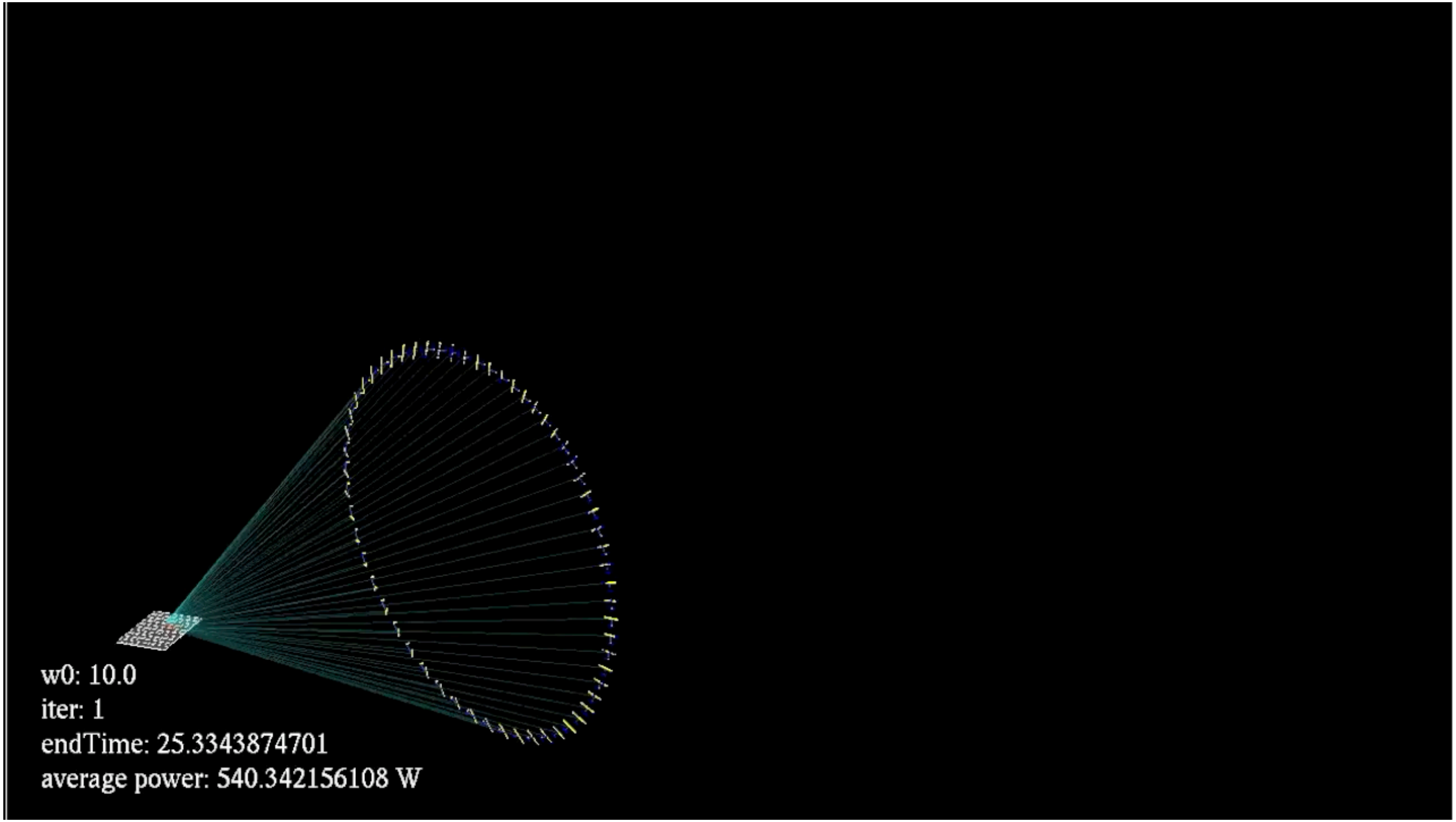
$$\text{Rotational:} \quad \dot{R} = R\omega_\times - R^T \begin{bmatrix} 0 \\ 0 \\ \dot{\delta} \end{bmatrix}, \qquad J\dot{\omega} = T - \omega \times J\omega, \qquad R = \begin{bmatrix} \vec{E}_x & \vec{E}_y & \vec{E}_z \end{bmatrix}$$

$$\text{Aero. coefficients:} \quad \vec{v} = \begin{bmatrix} \dot{x} - \dot{\delta}y \\ \dot{y} + \dot{\delta}(r_\mathrm{A} + x) \\ \dot{z} \end{bmatrix} - \vec{w}(x, y, z, \delta, t), \qquad \alpha = -\frac{\vec{E}_z^T \vec{v}}{\vec{E}_x^T \vec{v}}, \qquad \beta = \frac{\vec{E}_y^T \vec{v}}{\vec{E}_x^T \vec{v}}$$

$$\text{Aero. forces/torques:} \quad \vec{F}_\mathrm{A} = \frac{1}{2}\rho A \|\vec{v}\| (C_\mathrm{L} \vec{v} \times \vec{E}_y - C_\mathrm{D} \vec{v}), \quad \vec{T}_\mathrm{A} = \frac{1}{2}\rho A \|\vec{v}\|^2 \begin{bmatrix} C_\mathrm{R} \\ C_\mathrm{P} \\ C_\mathrm{Y} \end{bmatrix}$$

# Newton-Type Optimization Iterations for Power Optimal Flight

(video by Greg Horn, using CasADi and Ipopt as optimization engine)

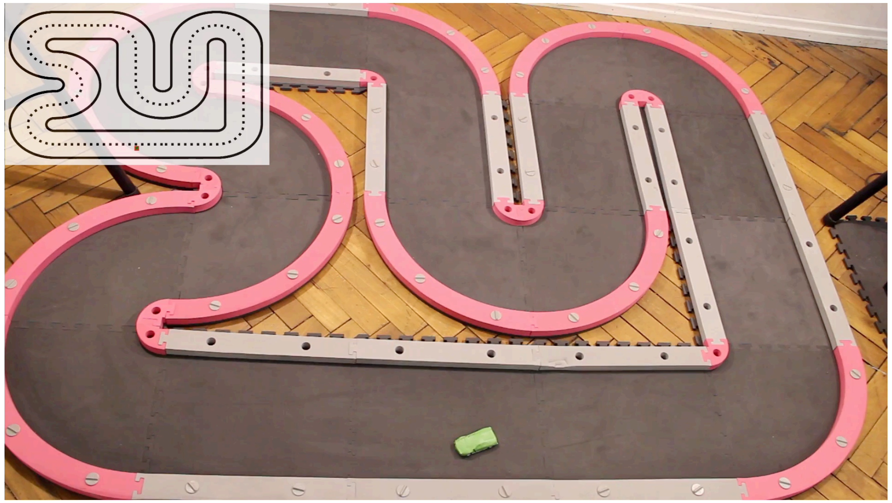


w0: 10.0
iter: 1
endTime: 25.3343874701
average power: 540.342156108 W

# Nonlinear Optimal Control often used for Model Predictive Control (MPC)

## One widely used nonlinear MPC package is `acados` [Verscheuren et al. 2021]

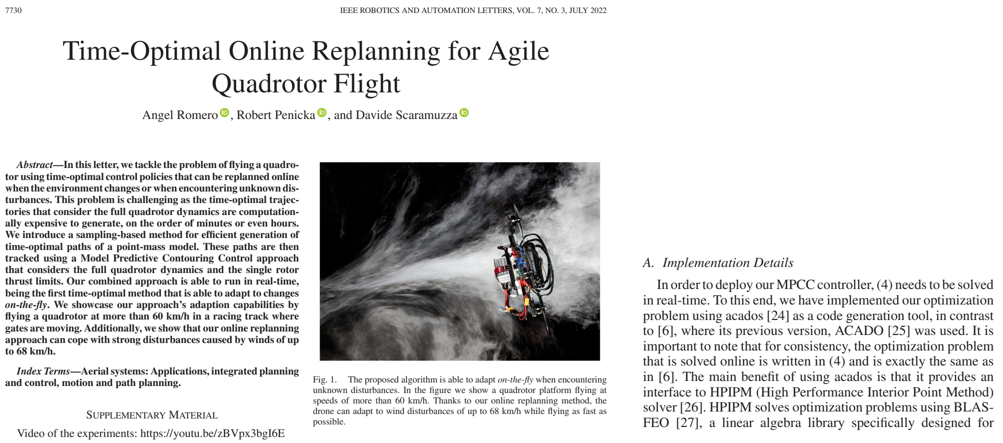
Example 1: Autonomous Driving (in Freiburg)



Example 2: Quadrotor Racing (U Zurich, Scaramuzza)

Paper: https://ieeexplore.ieee.org/abstract/document/9805699

Video: https://www.youtube.com/watch?v=zBVpx3bgI6E



7730 IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 7, NO. 3, JULY 2022

Time-Optimal Online Replanning for Agile Quadrotor Flight

Angel Romero, Robert Penicka, and Davide Scaramuzza

***Abstract*—In this letter, we tackle the problem of flying a quadrotor using time-optimal control policies that can be replanned online when the environment changes or when encountering unknown disturbances. This problem is challenging as the time-optimal trajectories that consider the full quadrotor dynamics are computationally expensive to generate, on the order of minutes or even hours. We introduce a sampling-based method for efficient generation of time-optimal paths of a point-mass model. These paths are then tracked using a Model Predictive Contouring Control approach that considers the full quadrotor dynamics and the single rotor thrust limits. Our combined approach is able to run in real-time, being the first time-optimal method that is able to adapt to changes *on-the-fly*. We showcase our approach's adaption capabilities by flying a quadrotor at more than 60 km/h in a racing track where gates are moving. Additionally, we show that our online replanning approach can cope with strong disturbances caused by winds of up to 68 km/h.**

***Index Terms*—Aerial systems: Applications, integrated planning and control, motion and path planning.**

SUPPLEMENTARY MATERIAL

Video of the experiments: https://youtu.be/zBVpx3bgI6E

Fig. 1. The proposed algorithm is able to adapt *on-the-fly* when encountering unknown disturbances. In the figure we show a quadrotor platform flying at speeds of more than 60 km/h. Thanks to our online replanning method, the drone can adapt to wind disturbances of up to 68 km/h while flying as fast as possible.

*A. Implementation Details*

In order to deploy our MPCC controller, (4) needs to be solved in real-time. To this end, we have implemented our optimization problem using acados [24] as a code generation tool, in contrast to [6], where its previous version, ACADO [25] was used. It is important to note that for consistency, the optimization problem that is solved online is written in (4) and is exactly the same as in [6]. The main benefit of using acados is that it provides an interface to HPIPM (High Performance Interior Point Method) solver [26]. HPIPM solves optimization problems using BLASFEO [27], a linear algebra library specifically designed for

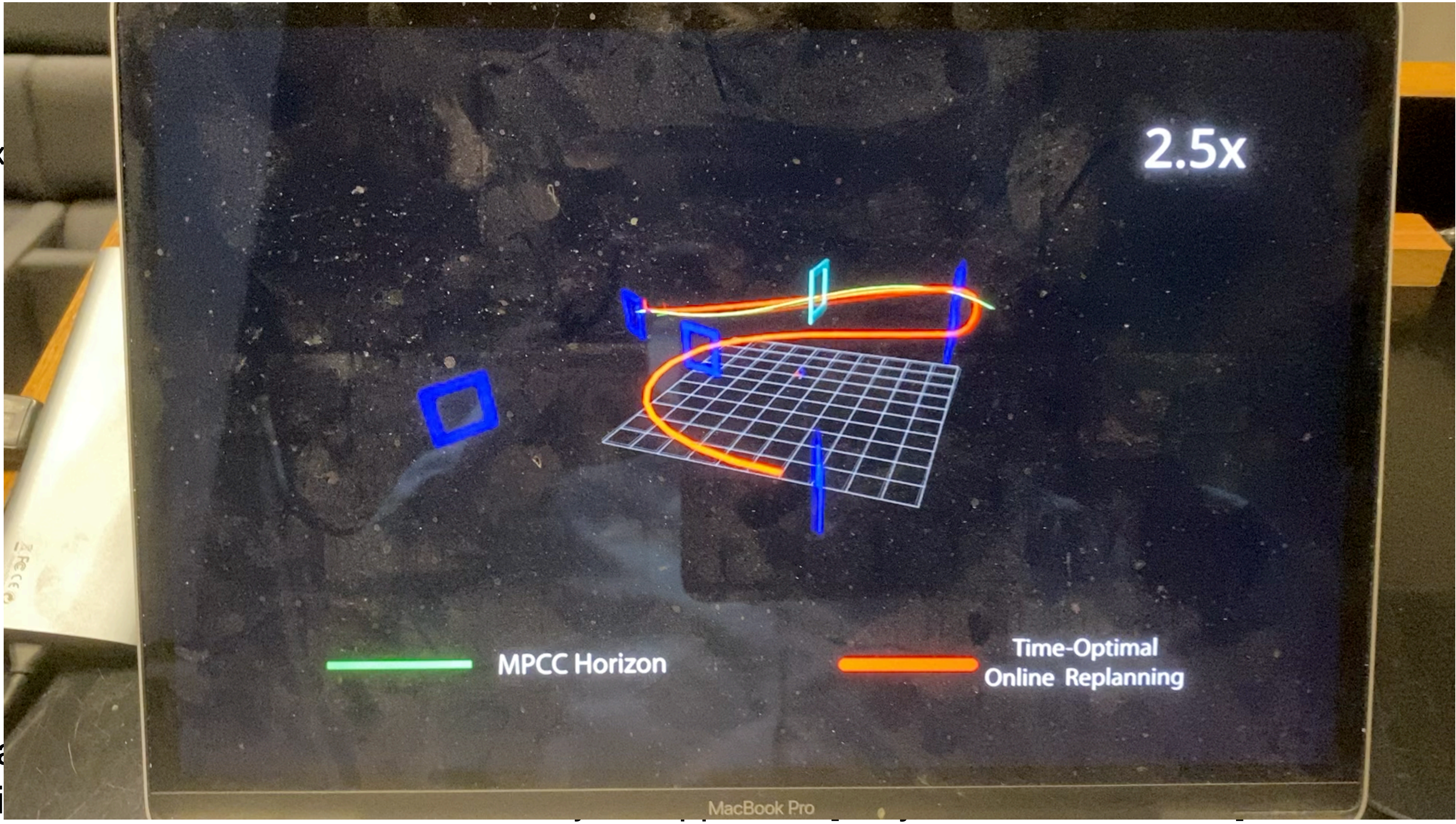
Latest **`acados`** development:
differentiable nonlinear MPC via adjoint approach [Frey et al. 2025, subm.]

# Nonlinear Optimal Control often used for Model Predictive Control (MPC)

## One widely used nonlinear MPC package is **acados** [Verscheuren et al. 2021]

Ex

za)



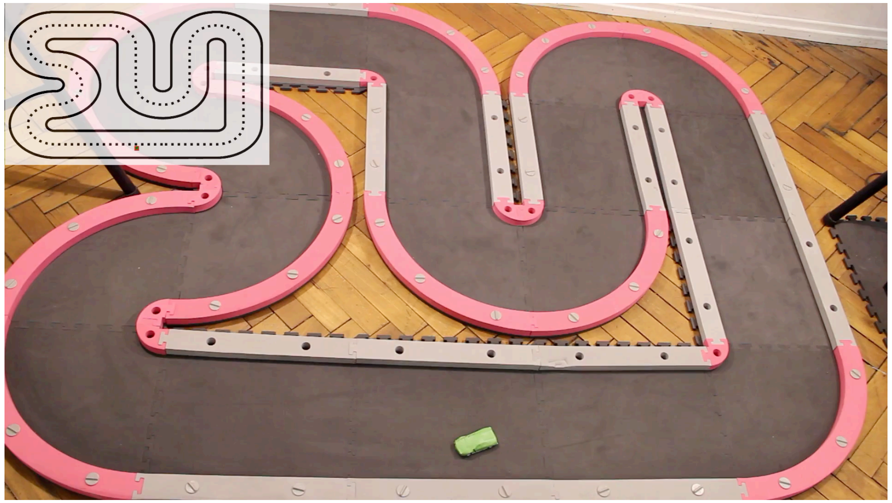
La

di

# Nonlinear Optimal Control often used for Model Predictive Control (MPC)

## One widely used nonlinear MPC package is `acados` [Verscheuren et al. 2021]

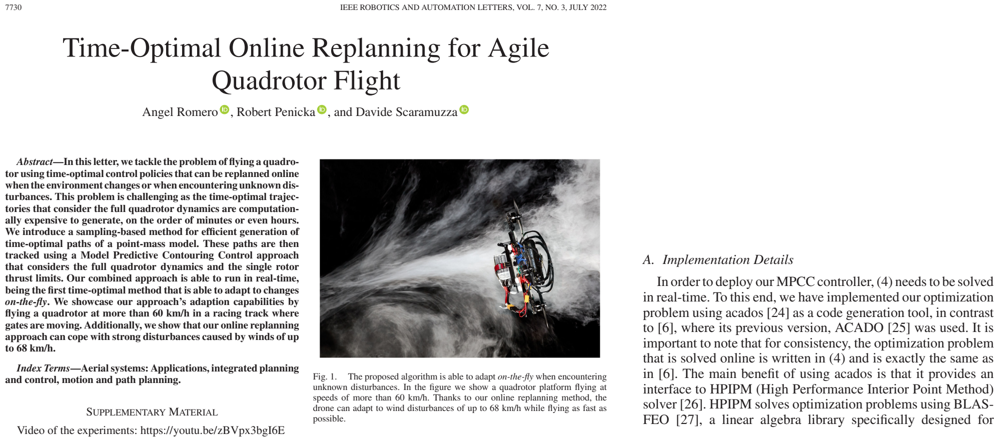
Example 1: Autonomous Driving (in Freiburg)



Example 2: Quadrotor Racing (U Zurich, Scaramuzza)

Paper: https://ieeexplore.ieee.org/abstract/document/9805699

Video: https://www.youtube.com/watch?v=zBVpx3bgI6E



7730 IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 7, NO. 3, JULY 2022

Time-Optimal Online Replanning for Agile Quadrotor Flight

Angel Romero, Robert Penicka, and Davide Scaramuzza

*Abstract*—**In this letter, we tackle the problem of flying a quadrotor using time-optimal control policies that can be replanned online when the environment changes or when encountering unknown disturbances. This problem is challenging as the time-optimal trajectories that consider the full quadrotor dynamics are computationally expensive to generate, on the order of minutes or even hours. We introduce a sampling-based method for efficient generation of time-optimal paths of a point-mass model. These paths are then tracked using a Model Predictive Contouring Control approach that considers the full quadrotor dynamics and the single rotor thrust limits. Our combined approach is able to run in real-time, being the first time-optimal method that is able to adapt to changes *on-the-fly*. We showcase our approach's adaption capabilities by flying a quadrotor at more than 60 km/h in a racing track where gates are moving. Additionally, we show that our online replanning approach can cope with strong disturbances caused by winds of up to 68 km/h.**

*Index Terms*—**Aerial systems: Applications, integrated planning and control, motion and path planning.**

SUPPLEMENTARY MATERIAL

Video of the experiments: https://youtu.be/zBVpx3bgI6E

Fig. 1. The proposed algorithm is able to adapt *on-the-fly* when encountering unknown disturbances. In the figure we show a quadrotor platform flying at speeds of more than 60 km/h. Thanks to our online replanning method, the drone can adapt to wind disturbances of up to 68 km/h while flying as fast as possible.

*A. Implementation Details*

In order to deploy our MPCC controller, (4) needs to be solved in real-time. To this end, we have implemented our optimization problem using acados [24] as a code generation tool, in contrast to [6], where its previous version, ACADO [25] was used. It is important to note that for consistency, the optimization problem that is solved online is written in (4) and is exactly the same as in [6]. The main benefit of using acados is that it provides an interface to HPIPM (High Performance Interior Point Method) solver [26]. HPIPM solves optimization problems using BLASFEO [27], a linear algebra library specifically designed for

Latest `acados` development:
differentiable nonlinear MPC via adjoint approach [Frey et al. 2025, subm.]

# Next Challenge: Nonsmooth Optimal Control

# Next Challenge: Nonsmooth Optimal Control

**Continuous-Time OCP**

$$\min_{x(\cdot),u(\cdot)} \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T))$$

$$\begin{aligned} \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\ t \in [0,T] \\ & 0 \geq r(x(T)) \end{aligned}$$

Three levels of difficulty:

(a) Linear ODE: $f(x,u) = Ax + Bu$

(b) Nonlinear smooth ODE: $f \in \mathcal{C}^1$

(c) **Nonsmooth Dynamics (NSD):**

- $f$ not differentiable (NSD1),

# Next Challenge: Nonsmooth Optimal Control

**Continuous-Time OCP**

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_\mathrm{c}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\ \text{s.t.} \quad & x(0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t),u(t)) \\ & 0 \geq h(x(t),u(t)),\ t \in [0,T] \\ & 0 \geq r(x(T)) \end{aligned}$$

Three levels of difficulty:

(a) Linear ODE: $f(x,u) = Ax + Bu$

(b) Nonlinear smooth ODE: $f \in \mathcal{C}^1$

(c) **Nonsmooth Dynamics (NSD):**

- $f$ not differentiable (NSD1),
- $f$ not continuous (NSD2), or even

# Next Challenge: Nonsmooth Optimal Control

**Continuous-Time OCP**

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^T L_{\mathrm{c}}(x(t),u(t))\,\mathrm{d}t + E(x(T)) \\
\text{s.t.} \quad & x(0) = \bar{x}_0 \\
& \dot{x}(t) = f(x(t),u(t)) \\
& 0 \geq h(x(t),u(t)),\ t \in [0,T] \\
& 0 \geq r(x(T))
\end{aligned}
$$

Three levels of difficulty:

(a) Linear ODE: $f(x,u) = Ax + Bu$

(b) Nonlinear smooth ODE: $f \in \mathcal{C}^1$

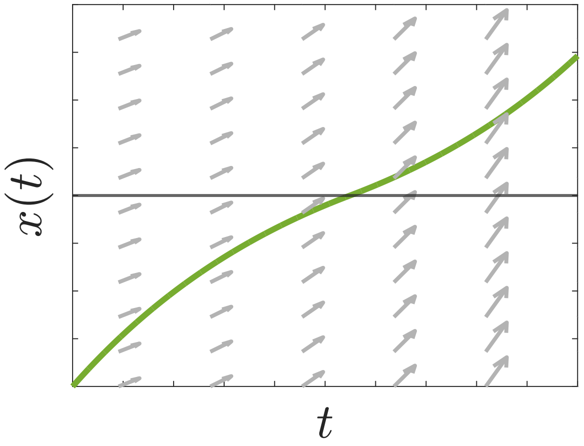
(c) **Nonsmooth Dynamics (NSD):**

- $f$ not differentiable (NSD1),
- $f$ not continuous (NSD2), or even
- $f$ not finite valued, discontinuous state $x(t)$ (NSD3)

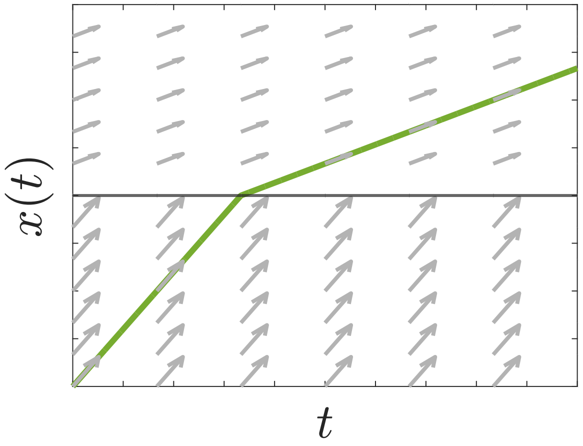
# Nonsmooth differential equations - hybrid systems

## Classification of Nonsmooth Dynamics (NSD)

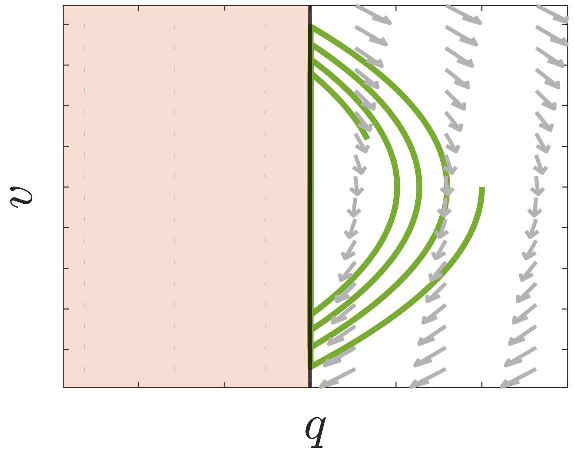
Ordinary differential equation (ODE) with a nonsmooth right-hand side (RHS).



**NSD1**
non-differentiable RHS



**NSD2**
discontinuous RHS



**NSD3**
state dependent jump

# Nonsmooth differential equations - hybrid systems

## Classification of Nonsmooth Dynamics (NSD)

Ordinary differential equation (ODE) with a nonsmooth right-hand side (RHS).

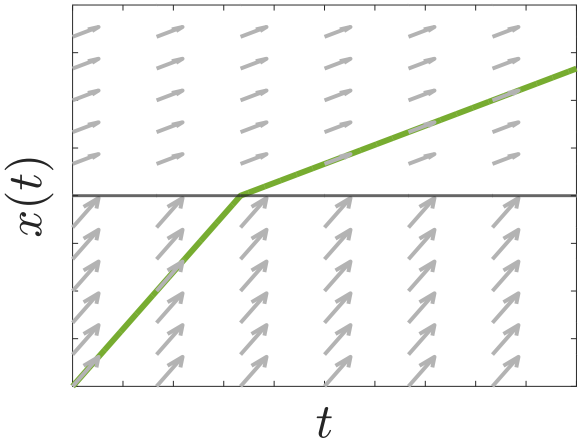
Continuous activation functions in the RHS

$$\dot{x} = 1 + \max(0, x)$$

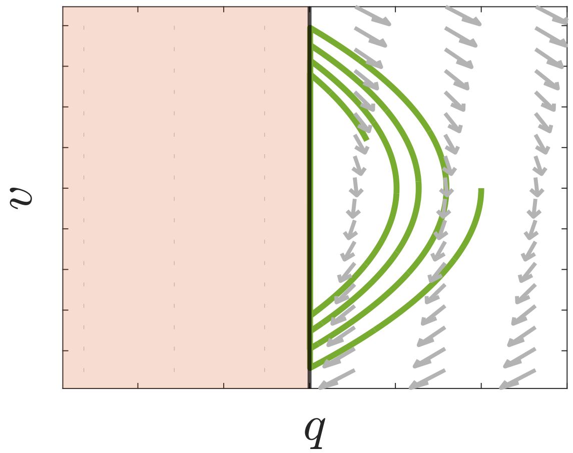
Continuous non-diff. ODEs

$$\dot{x} = 1 + |x|$$

**NSD1**
non-differentiable RHS



**NSD2**
discontinuous RHS



**NSD3**
state dependent jump

# Nonsmooth differential equations - hybrid systems

## Classification of Nonsmooth Dynamics (NSD)

Ordinary differential equation (ODE) with a nonsmooth right-hand side (RHS).

Continuous activation functions in the RHS

$$\dot{x} = 1 + \max(0, x)$$

Continuous non-diff. ODEs

$$\dot{x} = 1 + |x|$$

**NSD1**
non-differentiable RHS

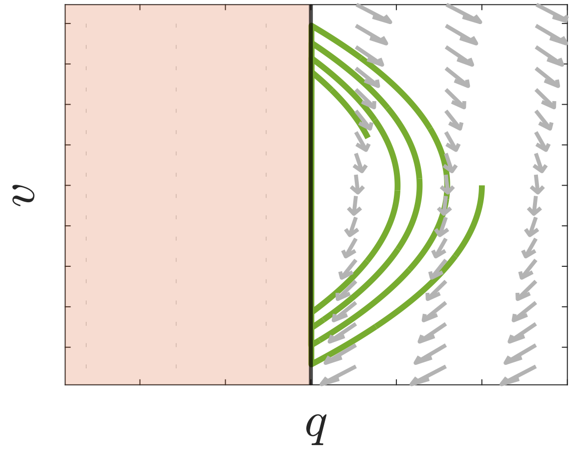
Piecewise smooth systems

$$\dot{x} = f_i(x), \text{ if } x \in R_i$$
$$i = 1, \ldots, m$$

Projected dynamical systems

$$\dot{x} = \mathrm{P}_{\mathcal{T}_C(x)}(f(x))$$

**NSD2**
discontinuous RHS



**NSD3**
state dependent jump

# Nonsmooth differential equations - hybrid systems

## Classification of Nonsmooth Dynamics (NSD)

Ordinary differential equation (ODE) with a nonsmooth right-hand side (RHS).

Continuous activation functions in the RHS

$$\dot{x} = 1 + \max(0, x)$$

Continuous non-diff. ODEs

$$\dot{x} = 1 + |x|$$

**NSD1**

non-differentiable RHS

Piecewise smooth systems

$$\begin{aligned} &\dot{x} = f_i(x), \text{ if } x \in R_i \\ &i = 1, \ldots, m \end{aligned}$$

Projected dynamical systems

$$\dot{x} = \mathrm{P}_{\mathcal{T}_C(x)}(f(x))$$

**NSD2**

discontinuous RHS

Rigid bodies with impacts and friction

$$\begin{aligned} \dot{q} &= v \\ M(q)\dot{v} &= f_{\mathrm{v}}(q, v) + J_{\mathrm{n}}(q)\lambda_{\mathrm{n}} \\ 0 \leq \lambda_{\mathrm{n}} &\perp f_{\mathrm{c}}(q) \geq 0 \\ &\text{(state jump law for } v\text{)} \end{aligned}$$

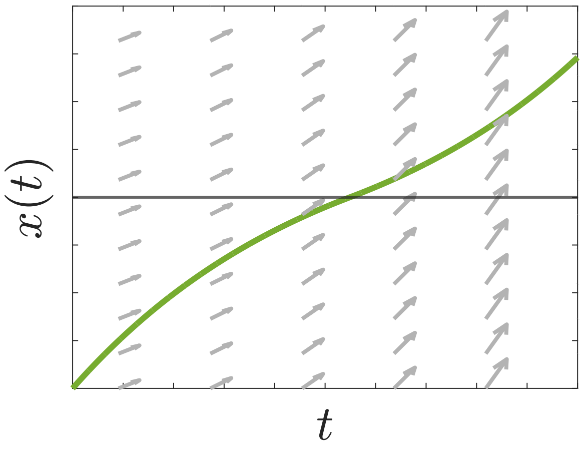
**NSD3**

state dependent jump

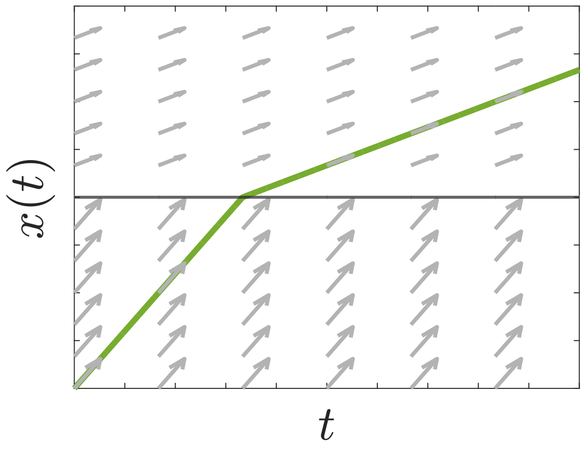
# Nonsmooth differential equations - hybrid systems

## Classification of Nonsmooth Dynamics (NSD)

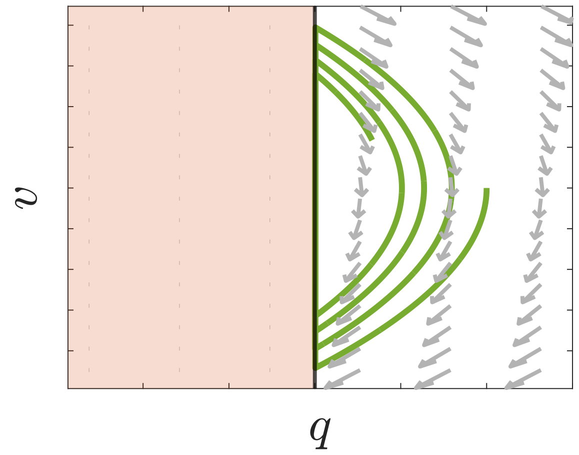
Ordinary differential equation (ODE) with a nonsmooth right-hand side (RHS).



**NSD1**
non-differentiable RHS



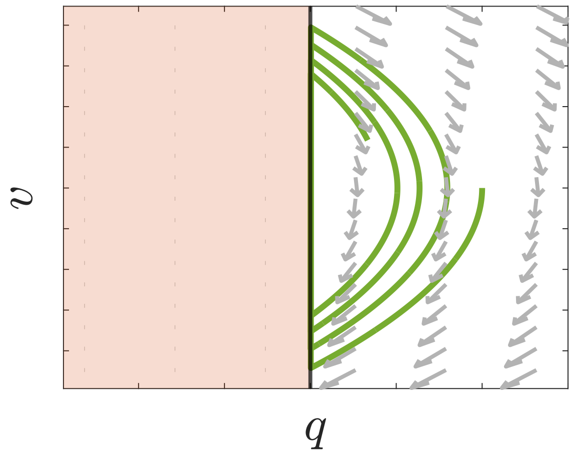
**NSD2**
discontinuous RHS



**NSD3**
state dependent jump

# Nonsmooth differential equations - hybrid systems

## Classification of Nonsmooth Dynamics (NSD)

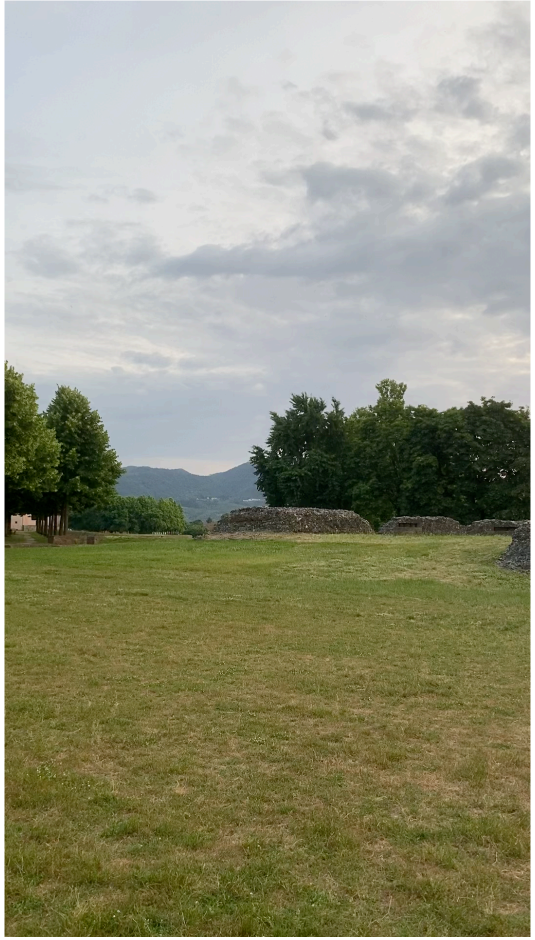


**NSD3**

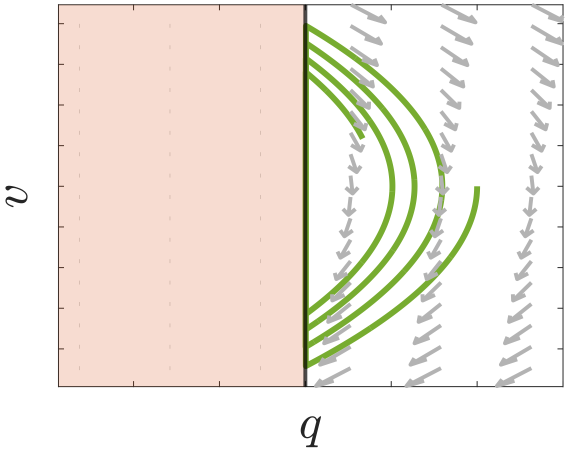
state dependent jump

# Nonsmooth differential equations - hybrid systems

## Classification of Nonsmooth Dynamics (NSD)



Bouncing Ball (NSD3)

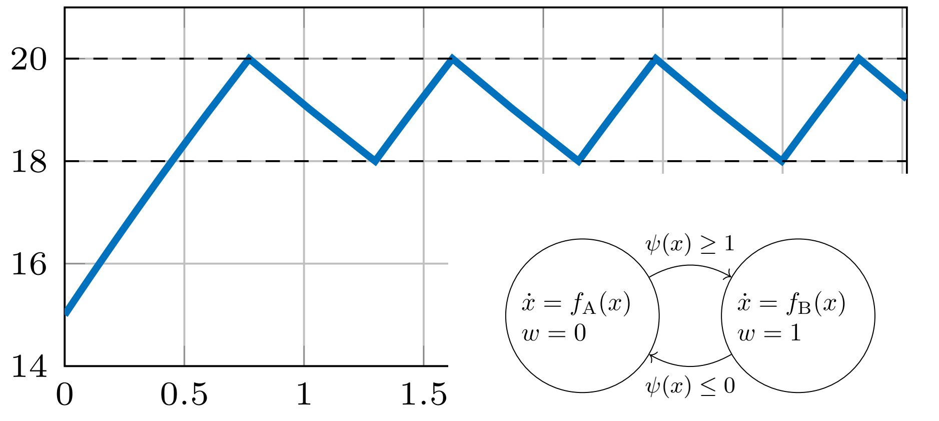


**NSD3**

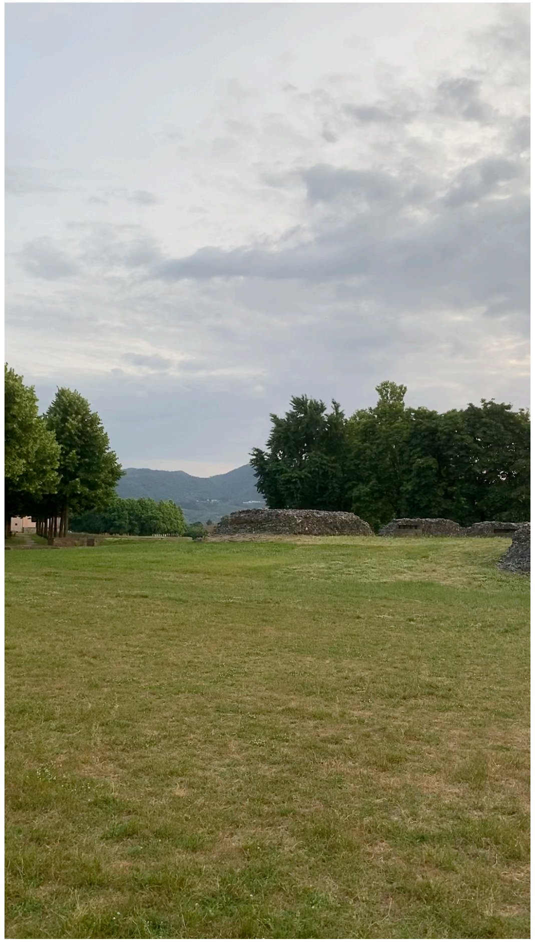
state dependent jump

# Nonsmooth differential equations - hybrid systems

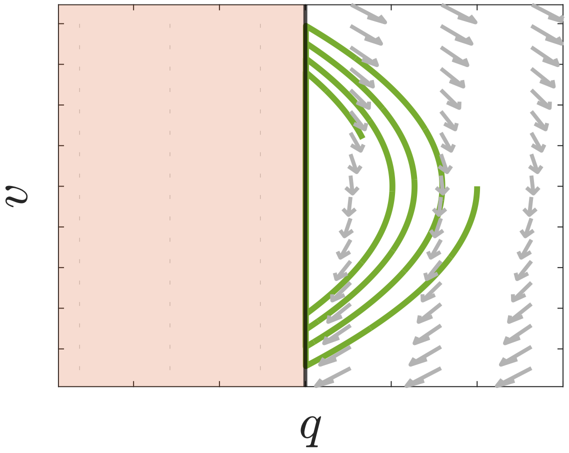
## Classification of Nonsmooth Dynamics (NSD)



State Machine in Hysteresis Control (NSD3)



Bouncing Ball (NSD3)

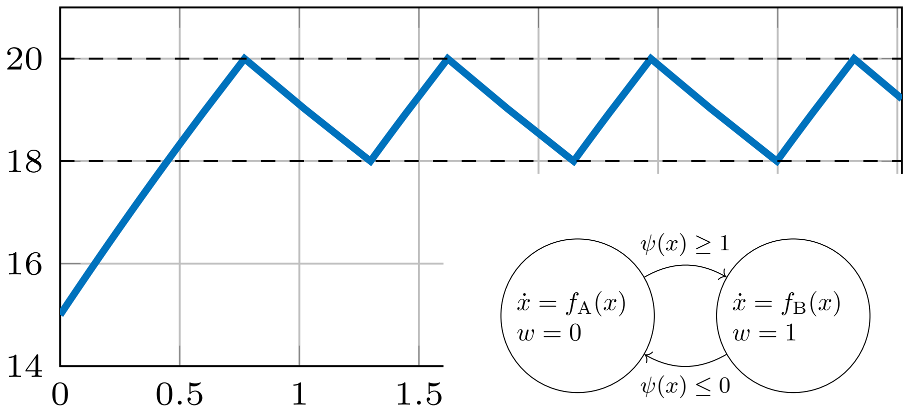


**NSD3**

state dependent jump

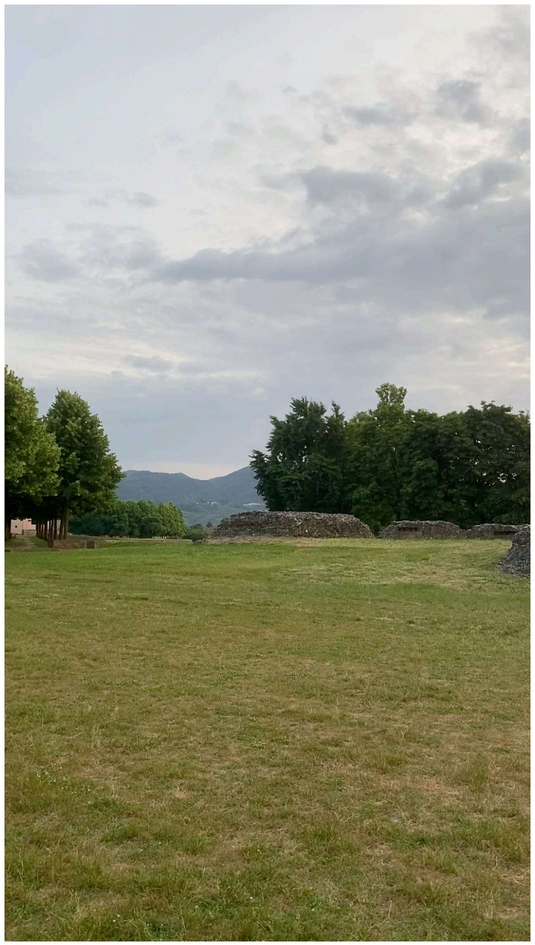
# Nonsmooth differential equations - hybrid systems

## Classification of Nonsmooth Dynamics (NSD)

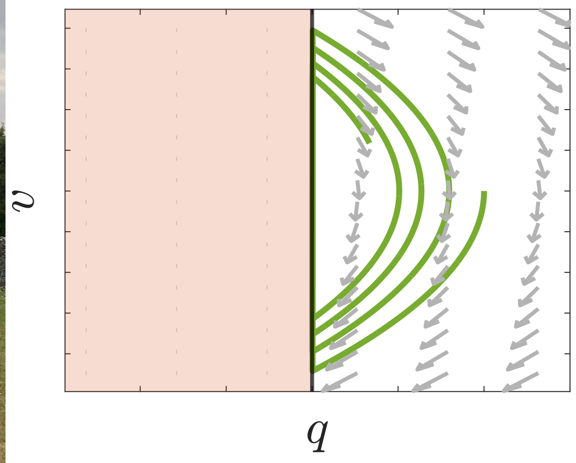


State Machine in Hysteresis Control (NSD3)

Walking Robot (unitree at LAAS, NSD3)



Bouncing Ball (NSD3)



**NSD3**

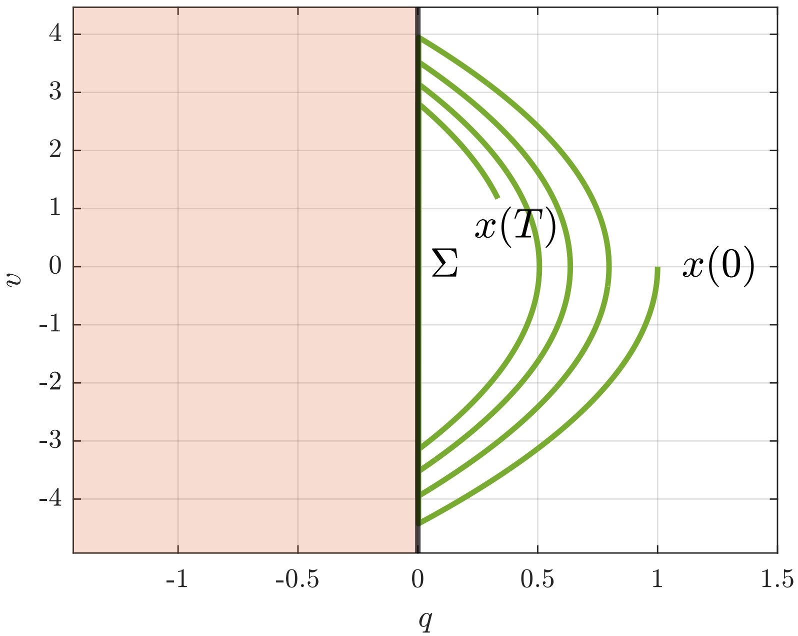
state dependent jump

# NSD3 state jump example: bouncing ball

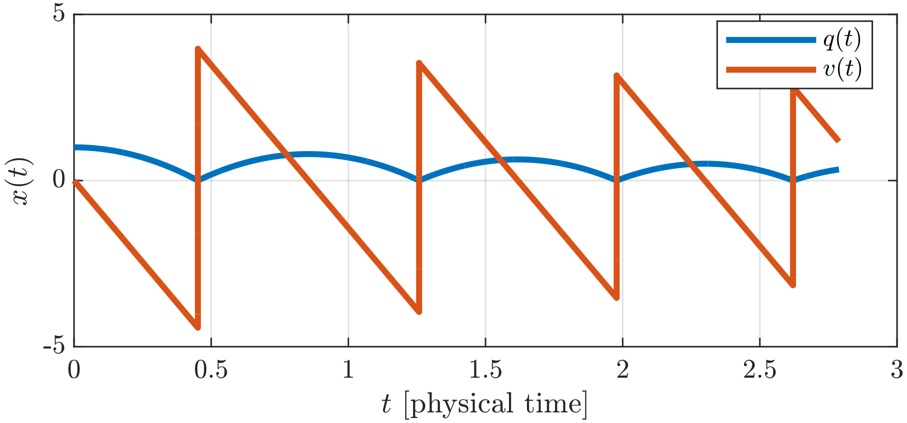
Bouncing ball with state $x = (q, v)$:

$$\begin{aligned} \dot{q} = v,\ m\dot{v} = -mg, \quad & \text{if } q > 0 \\ v(t^+) = -0.9\, v(t^-), \quad & \text{if } q(t^-) = 0 \text{ and } v(t^-) < 0 \end{aligned}$$

Phase plot of bouncing ball trajectory:



Time plot of bouncing ball trajectory:



# Can Newton-Type Optimization be Useful for NSD3 Systems ?

# Can Newton-Type Optimization be Useful for NSD3 Systems ?

Surprisingly, Yes !

# Can Newton-Type Optimization be Useful for NSD3 Systems ?

Surprisingly, Yes !

Some recent progress in Nonsmooth Optimal Control:

# Can Newton-Type Optimization be Useful for NSD3 Systems ?

Surprisingly, Yes !

Some recent progress in Nonsmooth Optimal Control:

- Can transform many NSD3 systems into (easier) NSD2 via **time-freezing**

# Can Newton-Type Optimization be Useful for NSD3 Systems ?

Surprisingly, Yes !

Some recent progress in Nonsmooth Optimal Control:

- Can transform many NSD3 systems into (easier) NSD2 via **time-freezing**
- Can discretize NSD2 systems with highly accurate **Finite Elements with Switch Detection (FESD),** removing spurious local minimisers

# Can Newton-Type Optimization be Useful for NSD3 Systems ?

Surprisingly, Yes !

Some recent progress in Nonsmooth Optimal Control:

- Can transform many NSD3 systems into (easier) NSD2 via **time-freezing**
- Can discretize NSD2 systems with highly accurate **Finite Elements with Switch Detection (FESD),** removing spurious local minimisers
- Can solve the resulting **Mathematical Programs with Complementarity Constraints (MPCC)** via homotopy of NLP (e.g. based on IPOPT)

# Can Newton-Type Optimization be Useful for NSD3 Systems ?

Surprisingly, Yes !

Some recent progress in Nonsmooth Optimal Control:

- Can transform many NSD3 systems into (easier) NSD2 via **time-freezing**
- Can discretize NSD2 systems with highly accurate **Finite Elements with Switch Detection (FESD),** removing spurious local minimisers
- Can solve the resulting **Mathematical Programs with Complementarity Constraints (MPCC)** via homotopy of NLP (e.g. based on IPOPT)
- Can use open-source software **NOSNOC,** from MATLAB and Python

`github.com/nosnoc/nosnoc`

# Can Newton-Type Optimization be Useful for NSD3 Systems ?

Surprisingly, Yes !

Some recent progress in Nonsmooth Optimal Control:

- Can transform many NSD3 systems into (easier) NSD2 via **time-freezing**
- Can discretize NSD2 systems with highly accurate **Finite Elements with Switch Detection (FESD),** removing spurious local minimisers
- Can solve the resulting **Mathematical Programs with Complementarity Constraints (MPCC)** via homotopy of NLP (e.g. based on IPOPT)
- Can use open-source software **NOSNOC,** from MATLAB and Python

`github.com/nosnoc/nosnoc`

PhD and Postdoc Work by **Armin Nurkanovic**

# First: Time Freezing

# First: Time Freezing

**Question: could we transform NSD3 systems into (easier) NSD2 systems?**

# First: Time Freezing

**Question: could we transform NSD3 systems into (easier) NSD2 systems?**

Time Freezing Reformulation based on three ideas:

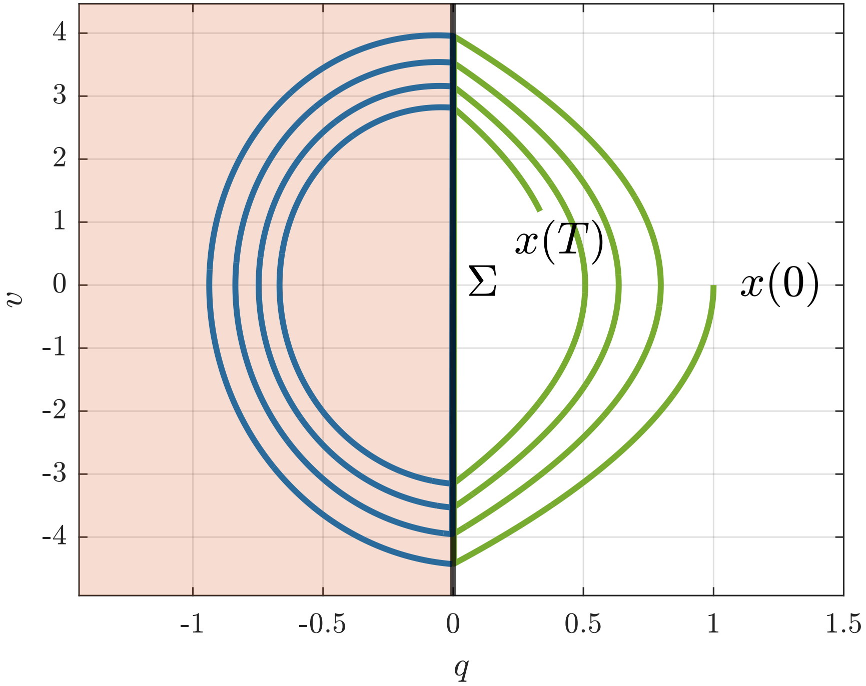
1. mimic state jump by **auxiliary dynamic system** $\dot{x} = f_{\mathrm{aux}}(x)$ on prohibited region
2. introduce a **clock state** $t(\tau)$ that stops counting when the auxiliary system is active
3. adapt speed of time, $\frac{\mathrm{d}t}{\mathrm{d}\tau} = s$ with $s \geq 1$, and **impose terminal constraint** $t(T) = T$

# The time-freezing reformulation

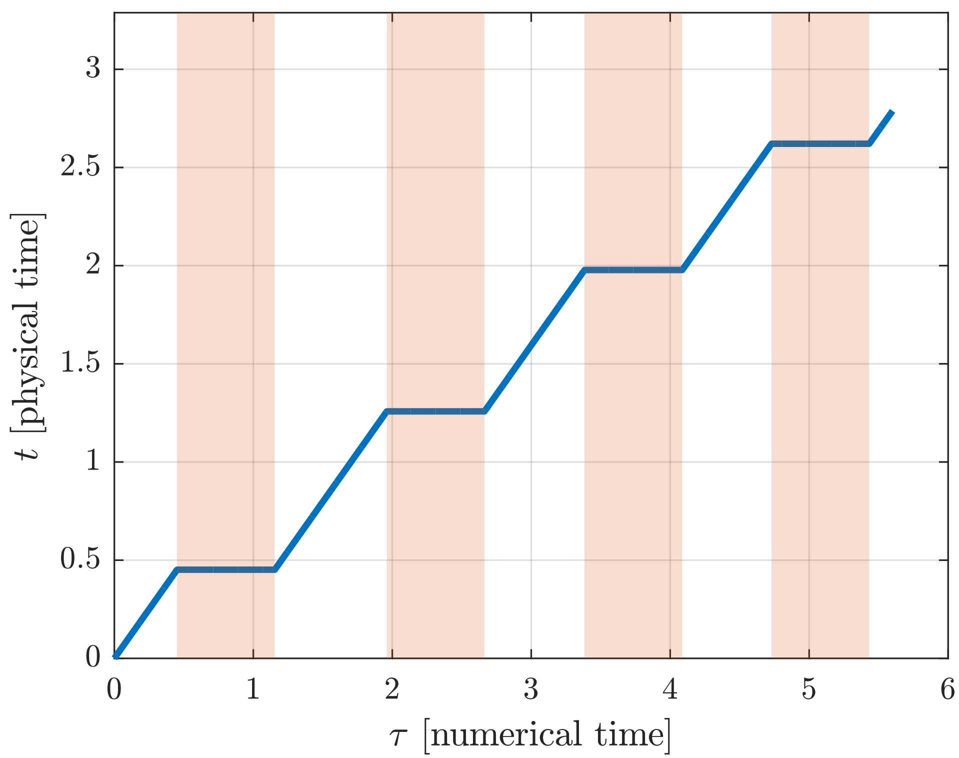
Augmented state $(x, t) \in \mathbb{R}^{n+1}$ evolves in **numerical time** $\tau$. Augmented system is nonsmooth, of NSD2 type:

$$\frac{\mathrm{d}}{\mathrm{d}\tau}\begin{bmatrix} x \\ t \end{bmatrix} = \begin{cases} s\begin{bmatrix} f(x) \\ 1 \end{bmatrix}, & \text{if } \ c(x) \geq 0 \\ \begin{bmatrix} s f_{\mathrm{aux}}(x) \\ 0 \end{bmatrix}, & \text{if } \ c(x) < 0 \end{cases}$$

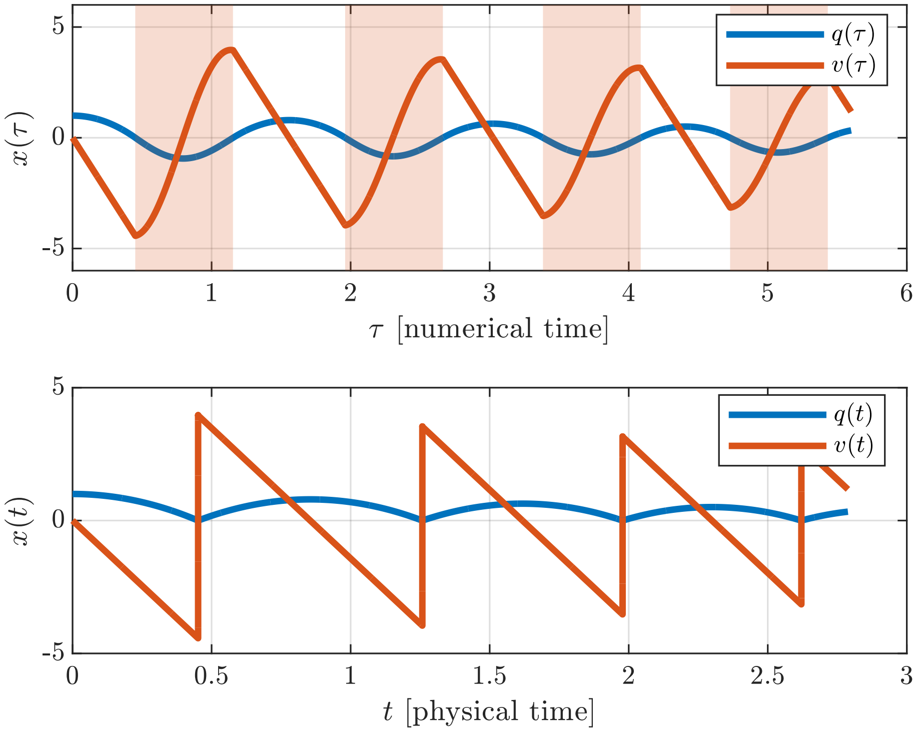
- During normal times, system and clock state evolve with adapted speed $s \geq 1$.
- Auxiliary system $\frac{\mathrm{d}x}{\mathrm{d}\tau} = f_{\mathrm{aux}}(x)$ mimics state jump while time is frozen, $\frac{\mathrm{d}t}{\mathrm{d}\tau} = 0$.



# Time-freezing for bouncing ball example



Evolution of physical time (clock state) during augmented system simulation ($s = 1$).



We can recover the true solution by plotting $x(\tau)$ vs. $t(\tau)$ and disregarding "frozen pieces".

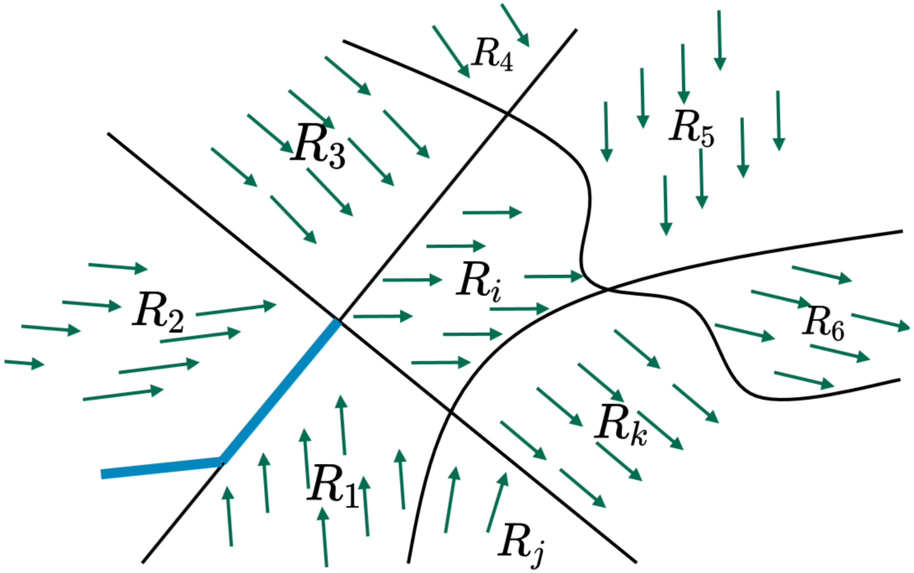
# Second: How to Optimise Switched (NSD2) Systems ?

# Regard Switched Systems that may include Sliding Modes

Regard **discontinuous** right-hand side, piecewise smooth on disjoint open regions $R_i \subset \mathbb{R}^{n_x}$

## Discontinuous ODE (NSD2)

$$\dot{x} = f_i(x,u), \text{ if } x \in R_i, \quad i \in \{1,\dots,n_f\}$$



Numerical aims:

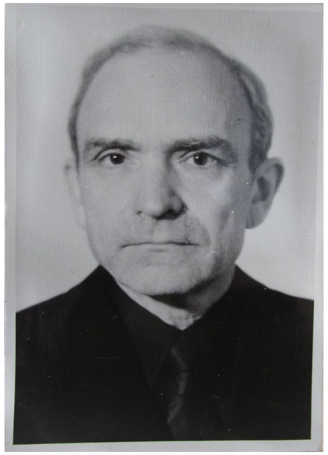
1. exactly detect switching times
2. obtain exact sensitivities across regions
3. appropriately treat evolution on boundaries (sliding mode → Filippov convexification)

# Filippov Convexification

Dynamics not yet well-defined on region boundaries $\partial R_i$. Idea by A.F. Filippov (1923-2006): replace ODE by differential inclusion, using convex combination of neighboring vector fields.

## Filippov Differential Inclusion

$$\dot{x} \in F_{\mathrm{F}}(x,u) := \Bigg\{ \sum_{i=1}^{n_f} f_i(x,u)\,\theta_i \;\Bigg|\; \sum_{i=1}^{n_f} \theta_i = 1, \\ \theta_i \geq 0, \quad i = 1, \ldots n_f, \\ \theta_i = 0, \quad \text{if } x \notin \overline{R_i} \quad \Bigg\}$$



Aleksei F. Filippov
(1923-2006)
image source: wikipedia

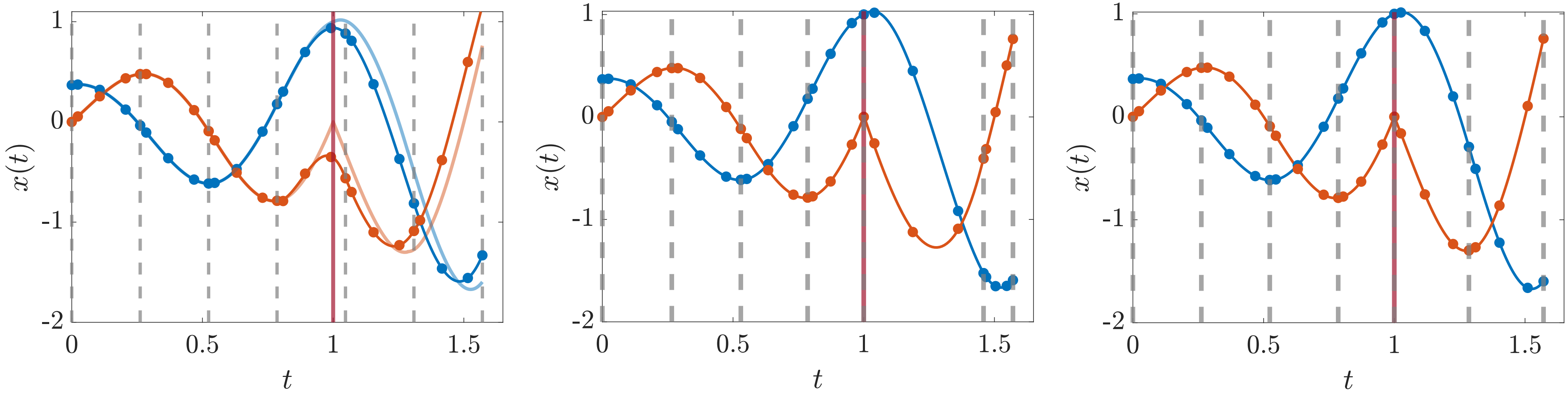
- for interior points $x \in R_i$ nothing changes: $F_{\mathrm{F}}(x,u) = \{f_i(x,u)\}$
- Provides meaningful generalization on region boundaries.
  E.g. on $\overline{R_1} \cap \overline{R_2}$ both $\theta_1$ and $\theta_2$ can be nonzero

# Finite Elements with Switch Detection (FESD)

Introduced in [Nurkanović et al., 2024], implemented in [Nurkanović and Diehl, 2022], extended in [Nurkanović et al., 2024, Nurkanović et al., 2024, Pozharskiy et al., 2024].

FESD is a novel DCS discretization method based on three ideas:

- make step sizes $h_n$ free, ensure $\sum_{n=0}^{N-1} h_n = T$ (cf. [Baumrucker and Biegler, 2009])
- allow switches only at element boundaries, enforce via **cross-complementarities**,
- remove spurious degrees of freedom via **step equilibration**.



conventional discretization

variable step sizes and cross-complementarities

FESD discretization with step equilibration

# Conventional DCS and FESD discretization

## Time-stepping discretization

$$
\begin{aligned}
x_{0,0} &= \bar{x}_0, \quad h = T/N \\
x_{n+1,0} &= x_{n,0} + h \textstyle\sum_{i=1}^{n_{\mathrm{s}}} b_i v_{n,i} \\
x_{n,i} &= x_{n,0} + h \textstyle\sum_{j=1}^{n_{\mathrm{s}}} a_{i,j} v_{n,j} \\
v_{n,i} &= F(x_{n,i}, u_{n,i})\, \theta_{n,i} \\
0 &= g(x_{n,i}) - \lambda_{n,i} - e\mu_{n,i} \\
0 &\leq \theta_{n,i} \perp \lambda_{n,i} \geq 0 \\
1 &= e^\top \theta_{n,i}
\end{aligned}
$$

for $i = 1, \ldots, n_{\mathrm{s}}$

and $n = 0, \ldots, N-1$

## FESD discretization with step equilibration

$$
\begin{aligned}
x_{0,0} &= \bar{x}_0, \; \textstyle\sum_{n=0}^{N-1} h_n = T \\
x_{n+1,0} &= x_{n,0} + h_n \textstyle\sum_{i=1}^{n_{\mathrm{s}}} b_i v_{n,i} \\
x_{n,i} &= x_{n,0} + h_n \textstyle\sum_{j=1}^{n_{\mathrm{s}}} a_{i,j} v_{n,j} \\
v_{n,i} &= F(x_{n,i}, u_{n,i})\, \theta_{n,i} \\
0 &= g(x_{n,i'}) - \lambda_{n,i'} - e\mu_{n,i'} \\
0 &\leq \theta_{n,i} \perp \lambda_{n,i'} \geq 0 \quad \text{(cross-complementarities)} \\
1 &= e^\top \theta_{n,i} \\
0 &= \nu(\theta_{n'}, \theta_{n'+1}, \lambda_{n'}, \lambda_{n'+1}) \cdot (h_{n'} - h_{n'+1})
\end{aligned}
$$

for $\quad i = 1, \ldots, n_{\mathrm{s}} \quad$ and $\quad n = 0, \ldots, N-1$

and $\quad i' = 0, 1, \ldots, n_{\mathrm{s}} \quad$ and $\quad n' = 0, \ldots, N-2$

- $N$ extra variables $(h_0, \ldots, h_{N-1})$ restricted by one extra equality
- Additional multipliers $\lambda_{n,0}, \mu_{n,0}$ are uniquely determined
- Indicator function $\nu(\theta_{n'}, \theta_{n'+1}, \lambda_{k'}, \lambda_{k'+1})$ only zero if a switch occurs

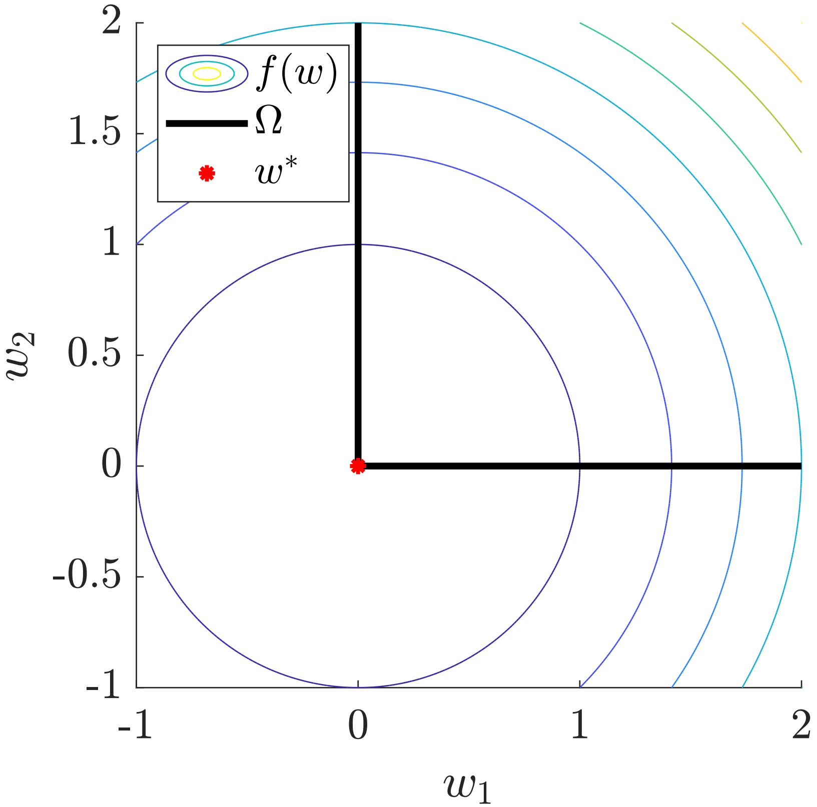
# Numerical methods for MPCCs

## MPEC

$$
\begin{aligned}
\min_{w\in\mathbb{R}^n} \quad & f(w) && \text{(3a)}\\
\text{s.t.} \quad & g(w) = 0, && \text{(3b)}\\
& h(w) \geq 0, && \text{(3c)}\\
& 0 \leq w_1 \perp w_2 \geq 0, && \text{(3d)}
\end{aligned}
$$

$$w = (w_0, w_1, w_2) \in \mathbb{R}^n,\ w_0 \in \mathbb{R}^p,\ w_1, w_2 \in \mathbb{R}^m,$$

$$\Omega = \{x \in \mathbb{R}^n \mid g(w) = 0, h(w) \geq 0,\ 0 \leq w_1 \perp w_2 \geq 0\},$$



- Standard NLP methods solve the KKT conditions.
- MPECs violate constraint qualifications, and the KKT conditions may not be necessary.
- There are many stationary concepts for MPECs, and not all are useful.

# Numerical methods for MPCCs

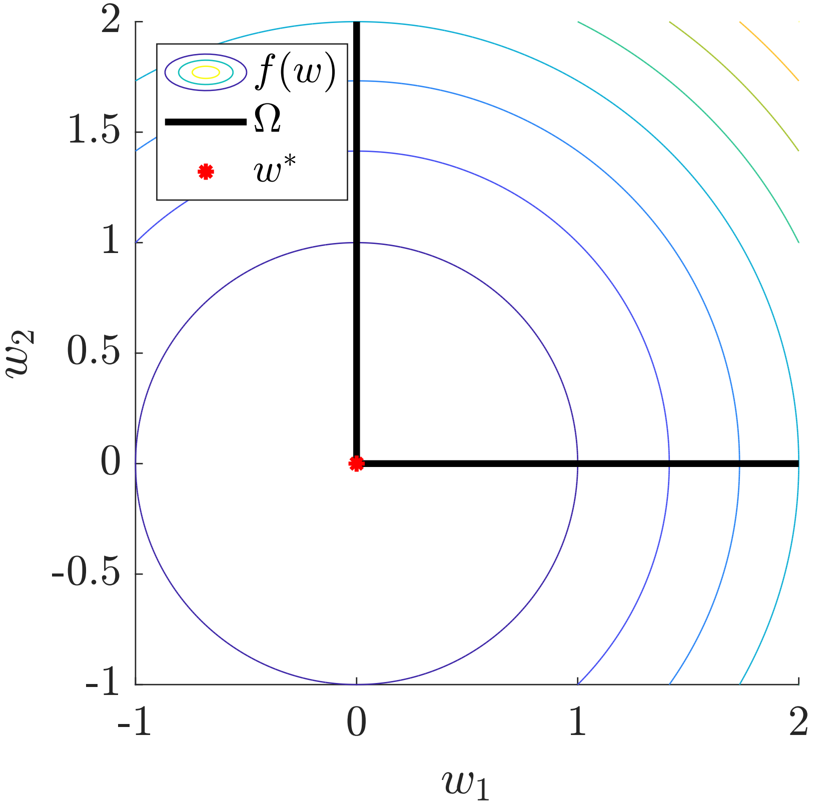
## MPEC

$$\min_{w\in\mathbb{R}^n} \quad f(w) \tag{3a}$$

$$\text{s.t.} \quad g(w) = 0, \tag{3b}$$

$$h(w) \geq 0, \tag{3c}$$

$$0 \leq w_1 \perp w_2 \geq 0, \tag{3d}$$



$$w = (w_0, w_1, w_2) \in \mathbb{R}^n,\ w_0 \in \mathbb{R}^p,\ w_1, w_2 \in \mathbb{R}^m,$$

$$\Omega = \{x \in \mathbb{R}^n \mid g(w) = 0, h(w) \geq 0,\ 0 \leq w_1 \perp w_2 \geq 0\},$$

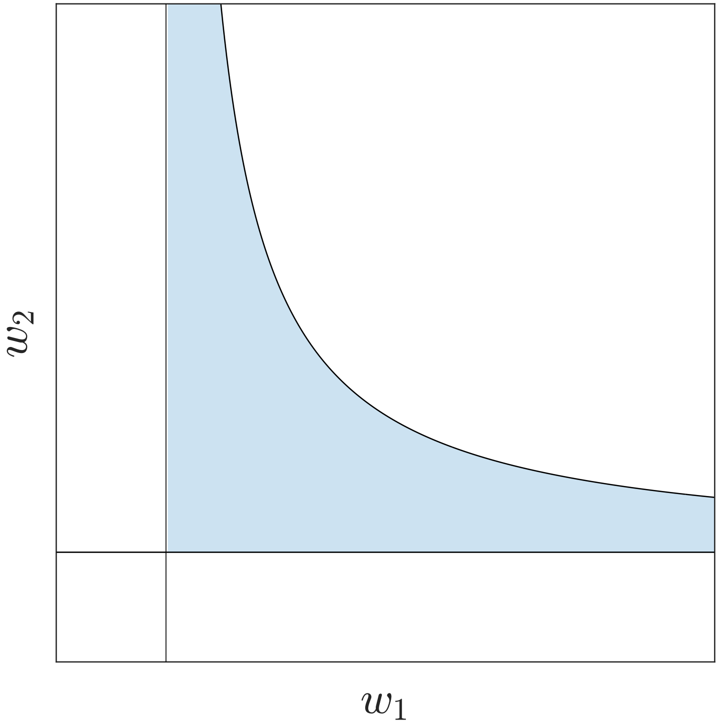
- Standard NLP methods solve the KKT conditions.
- MPECs violate constraint qualifications, and the KKT conditions may not be necessary.
- There are many stationary concepts for MPECs, and not all are useful.
- **Workaround/main idea**: solve a (finite) sequence of more regular problems.

# Scholtes' global relaxation method

The easiest to implement and the most efficient regularization method [Scholtes, 2001].

### Reg($\sigma^k$)

$$
\begin{aligned}
\min_{w \in \mathbb{R}^n} \quad & f(w) \\
\text{s.t.} \quad & g(w) = 0, \\
& h(w) \geq 0, \\
& w_1, w_2 \geq 0, \\
& w_{1,i} w_{2,i} \leq \sigma^k, \ \ i = 1, \dots, m.
\end{aligned}
$$

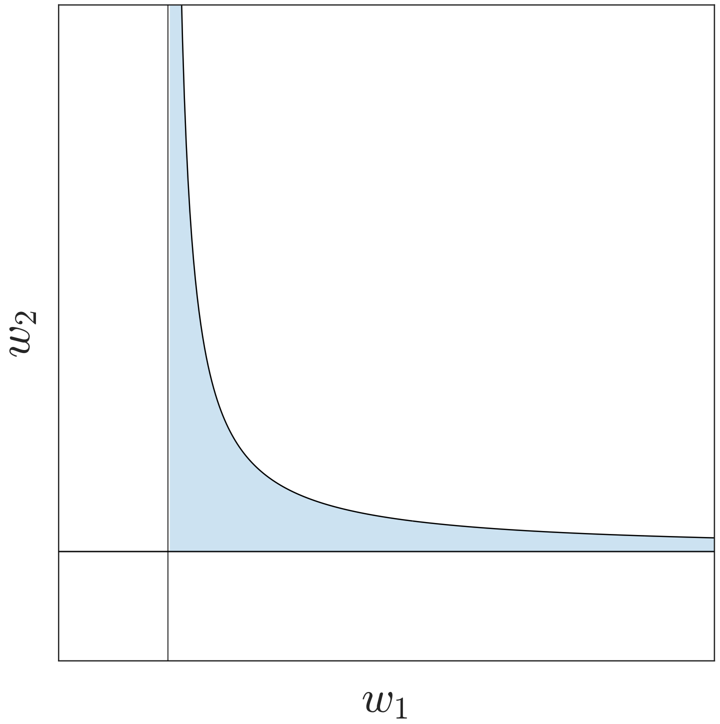


# Scholtes' global relaxation method

The easiest to implement and the most efficient regularization method [Scholtes, 2001].

**Reg($\sigma^k$)**

$$
\begin{aligned}
\min_{w \in \mathbb{R}^n} \quad & f(w) \\
\text{s.t.} \quad & g(w) = 0, \\
& h(w) \geq 0, \\
& w_1, w_2 \geq 0, \\
& w_{1,i} w_{2,i} \leq \sigma^k, \;\; i = 1, \dots, m.
\end{aligned}
$$

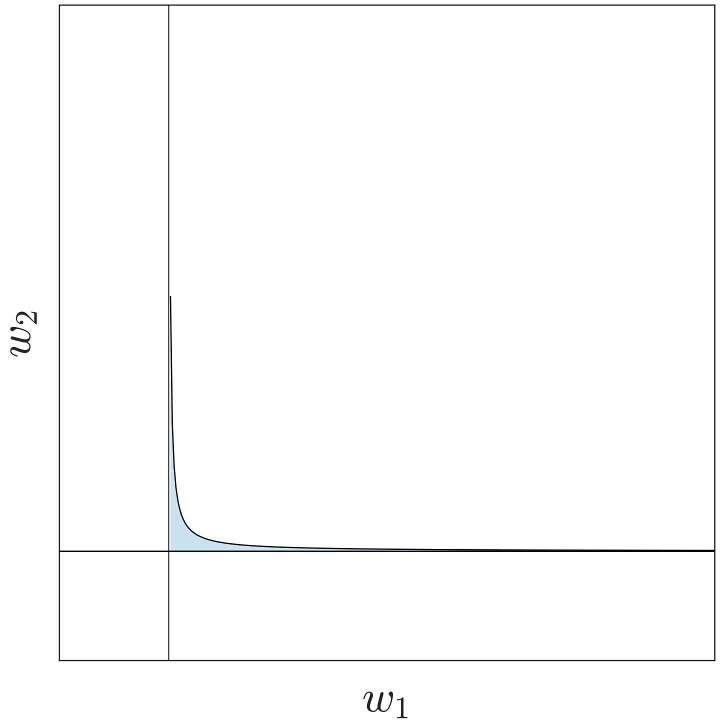


# Scholtes' global relaxation method

The easiest to implement and the most efficient regularization method [Scholtes, 2001].

### Reg($\sigma^k$)

$$
\begin{aligned}
\min_{w\in\mathbb{R}^n} \quad & f(w) \\
\text{s.t.} \quad & g(w) = 0, \\
& h(w) \geq 0, \\
& w_1, w_2 \geq 0, \\
& w_{1,i} w_{2,i} \leq \sigma^k, \;\; i = 1, \ldots, m.
\end{aligned}
$$



# Optimal control problem - benchmark example

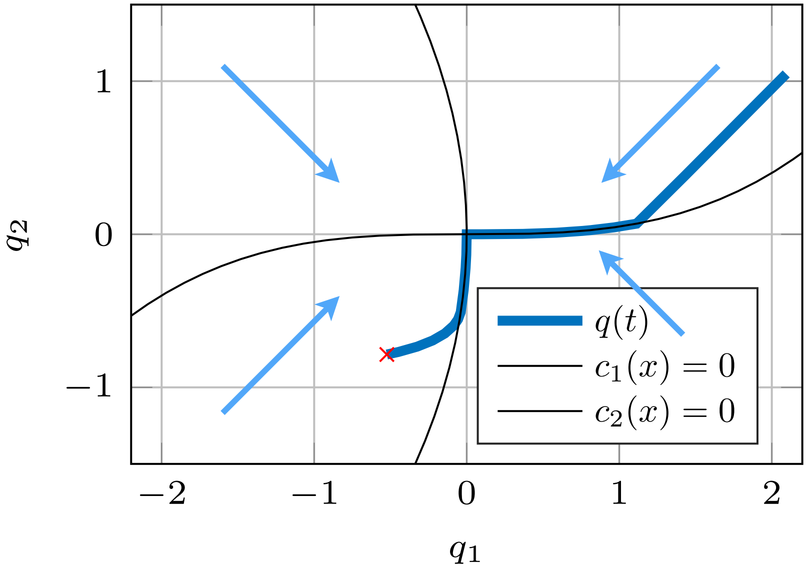
Benchmark example with entering/leaving sliding mode.

States $q, v \in \mathbb{R}^2$ and control $u \in \mathbb{R}^2$, $x = (q, v)$

## OCP with sliding modes

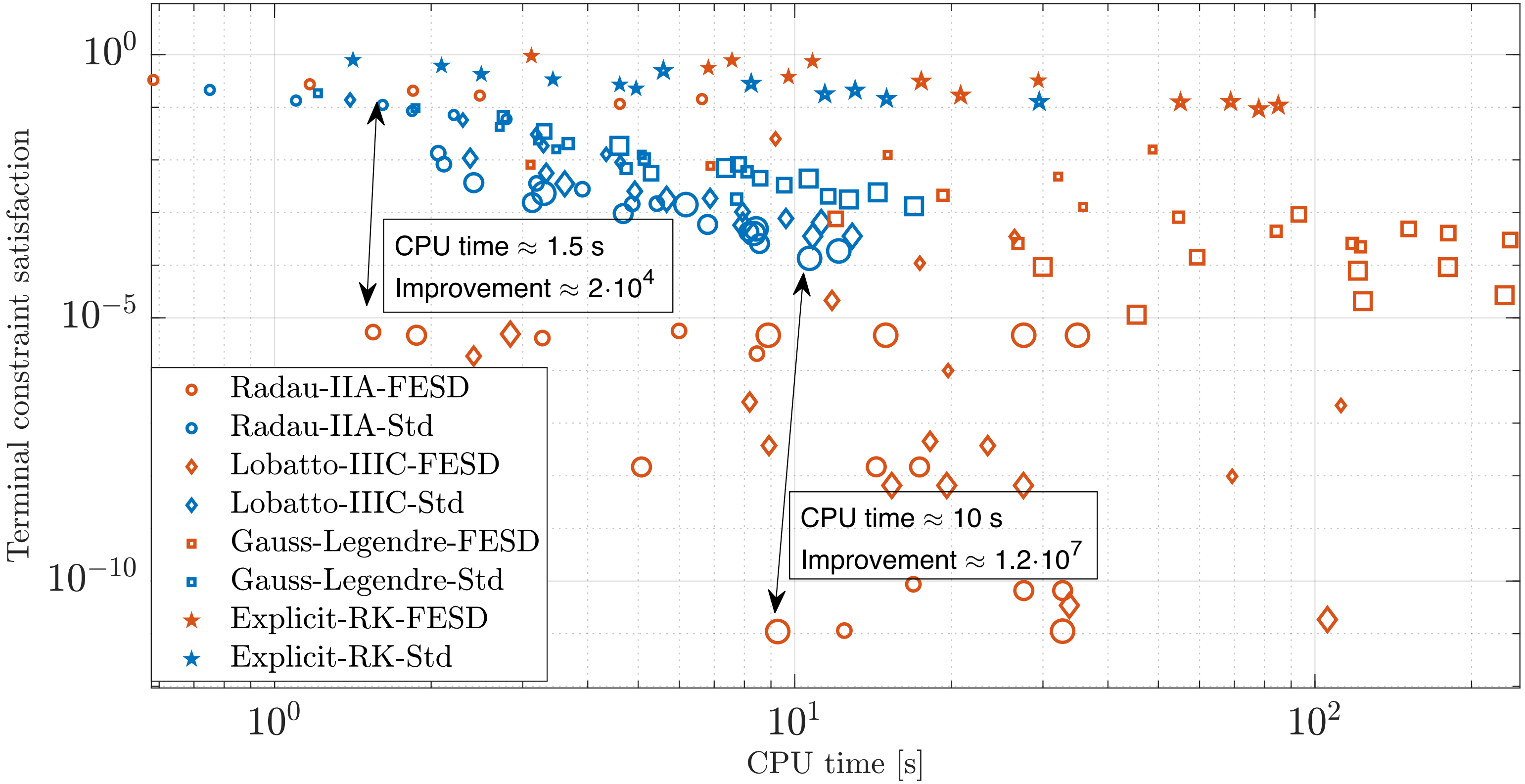
$$\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \int_0^4 u(t)^\top u(t) + v(t)^\top v(t)\,\mathrm{d}t \\
\text{s.t.} \quad & x(0) = (\frac{2\pi}{3}, \frac{\pi}{3}, 0, 0) \\
& \dot{x}(t) = \begin{bmatrix} -\mathrm{sign}(c(x(t))) + v(t) \\ u(t) \end{bmatrix} \\
& -2e \le v(t) \le 2e \\
& -10e \le u(t) \le 10e \quad t \in [0,4], \\
& q(T) = (-\frac{\pi}{6}, -\frac{\pi}{4})
\end{aligned}$$

Switching functions $c(x) = \begin{bmatrix} q_1 + 0.15 q_2^2 \\ 0.05 q_1^3 + q_2 \end{bmatrix}$



# FESD vs standard IRK benchmark run with nosnoc

Benchmark on an optimal control problem with nonlinear sliding modes. Bigger marker = higher order.

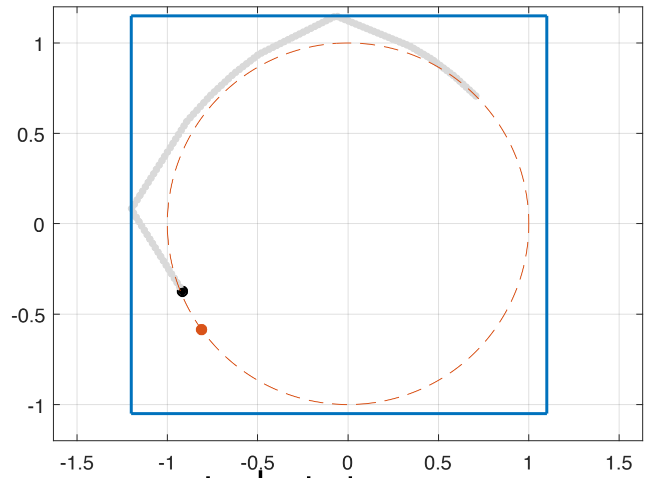


**FESD orders of magnitude more accurate than time-stepping for same CPU time.**

# Now: apply FESD and MPCC to Time-Freezing Problems

# A tracking OCP example with Time-Freezing and FESD in NOSNOC

Regard bouncing ball in two dimensions driven by bounded force: $\boxed{\ddot{q} = u}$



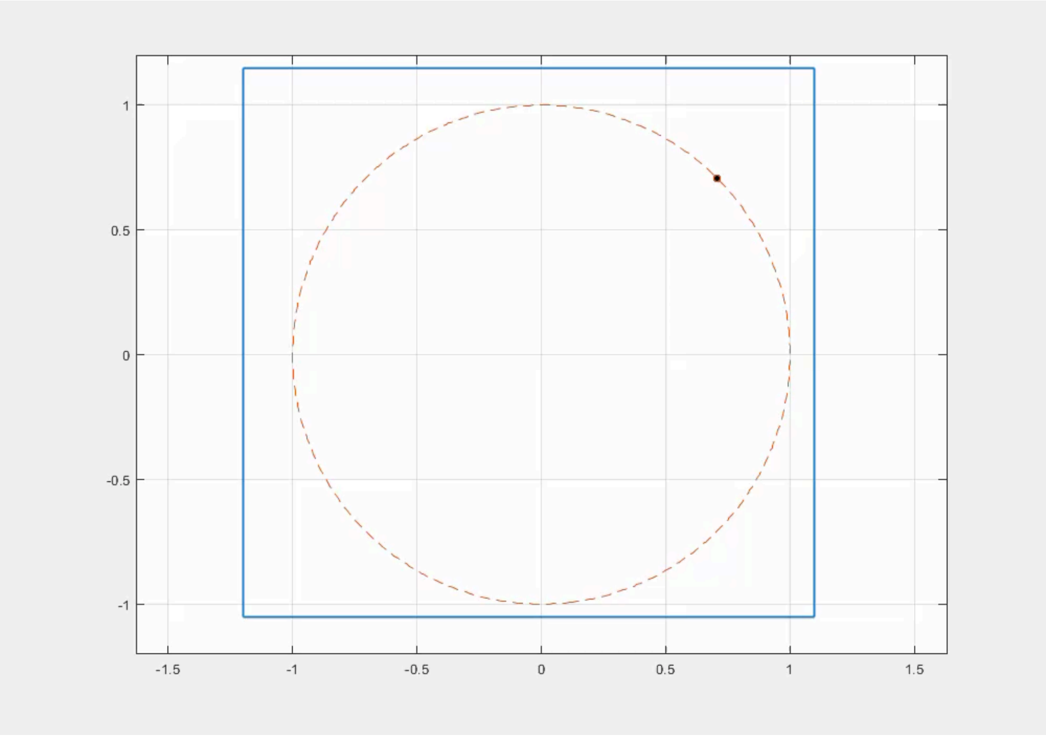
- augmented state $x = (q, \dot{q}, t) \in \mathbb{R}^5$
- $n_f = 9$ regions (8 with auxiliary dynamics for state jumps)

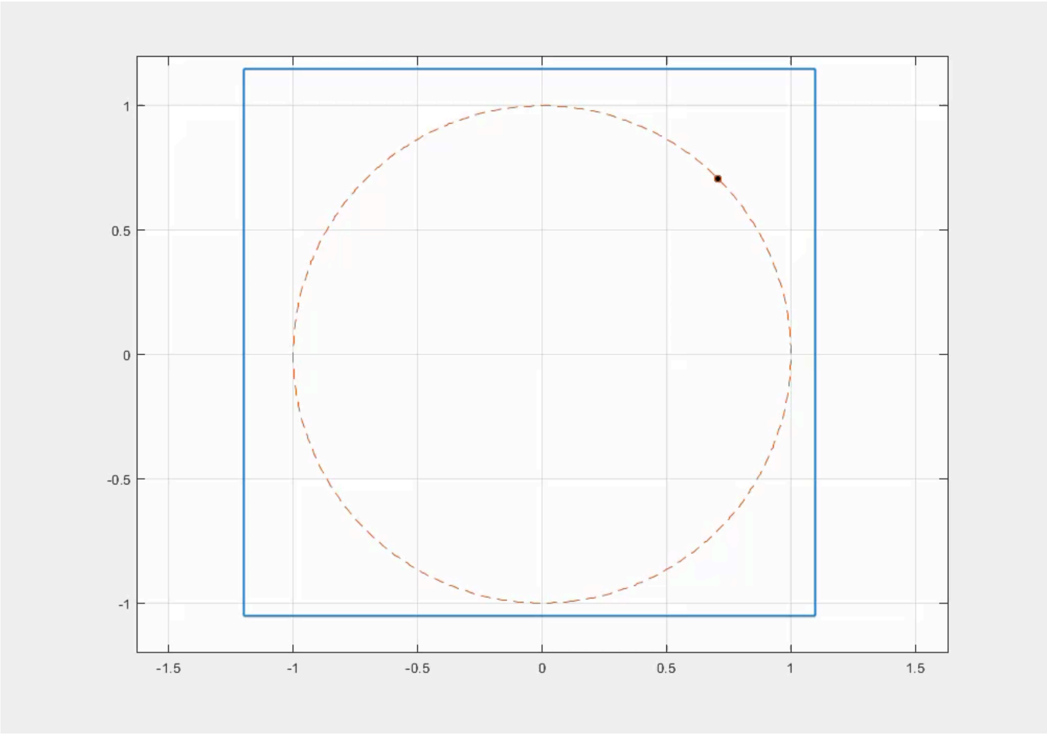
$$
\begin{aligned}
\min_{\substack{x(.),u(.),s(.),\\ \theta(.),\lambda(.),\mu(.)}} \quad & \int_0^T (q - q_{\mathrm{ref}}(\tau))^\top (q - q_{\mathrm{ref}}(\tau))\, s(\tau)\, \mathrm{d}\tau \\
\text{s.t.} \quad & x(0) = x_0, \quad t(T) = T, \\
& x'(\tau) = \sum_{i=1}^{n_f} \theta_i(\tau) f_i(x(\tau), u(\tau), s(\tau)), \\
& 0 = g(x(\tau)) - \lambda(\tau) - \mu(\tau) e, \\
& 0 \leq \lambda(\tau) \perp \theta(\tau) \geq 0, \\
& 1 = e^\top \theta(\tau), \\
& \|u(\tau)\|_2^2 \leq u_{\max}^2, \\
& 1 \leq s(\tau) \leq s_{\max}, \ \ \tau \in [0, T].
\end{aligned}
$$

$$q_{\mathrm{ref}}(\tau) = (R\cos(\omega\, t(\tau)), R\sin(\omega\, t(\tau))).$$

# A tracking OCP example with Time-Freezing and FESD in NOSNOC

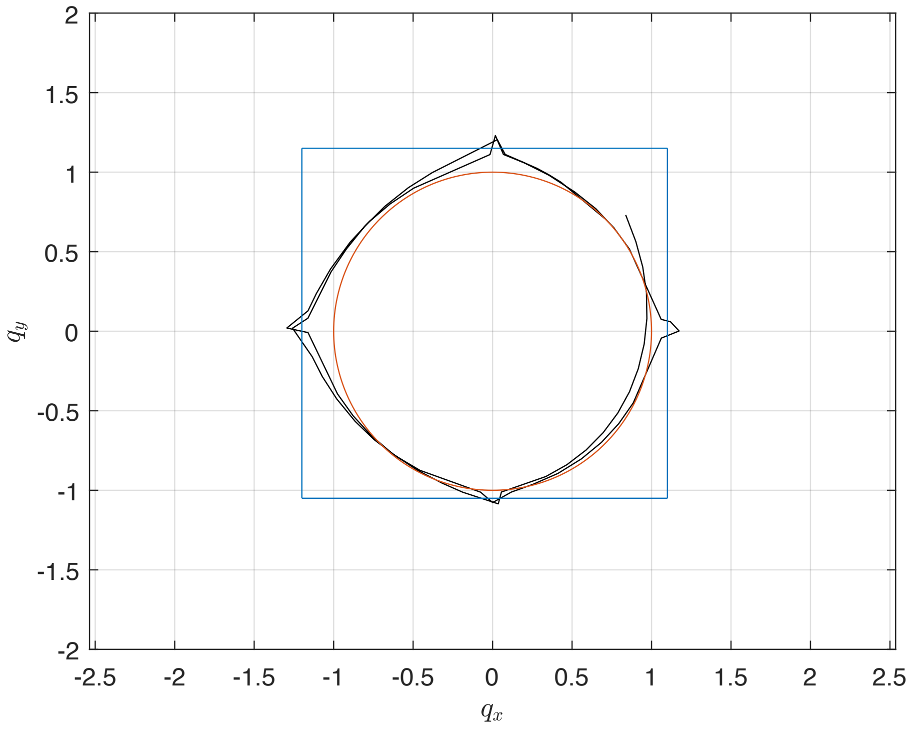
Regard bouncing ball in two dimensions driven by bounded force: $\boxed{\ddot{q} = u}$



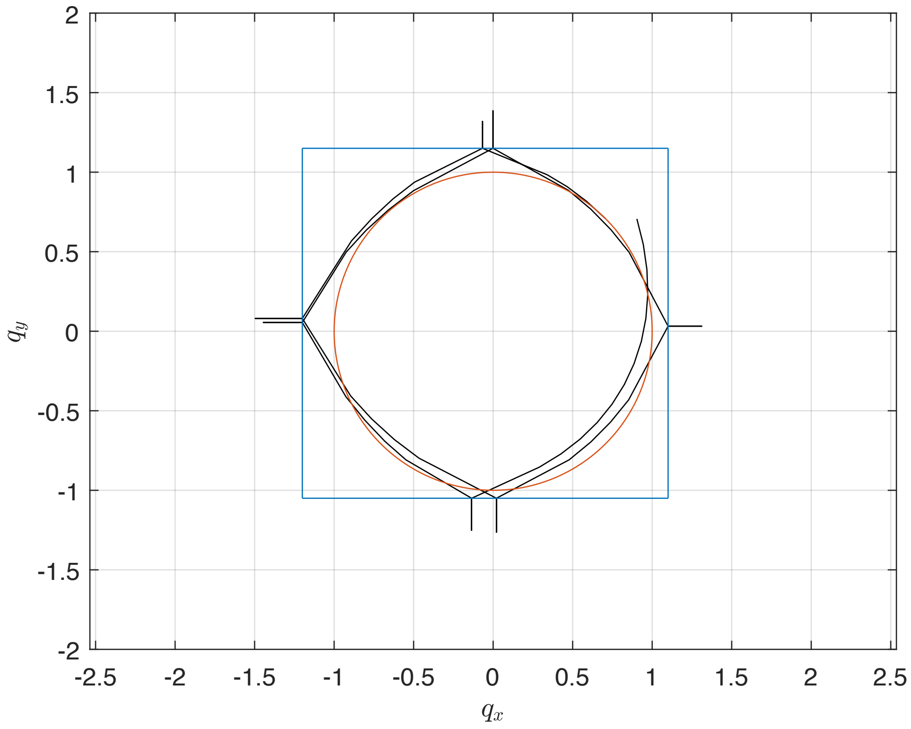


# Homotopy: first iteration vs converged solution

## Geometric trajectory



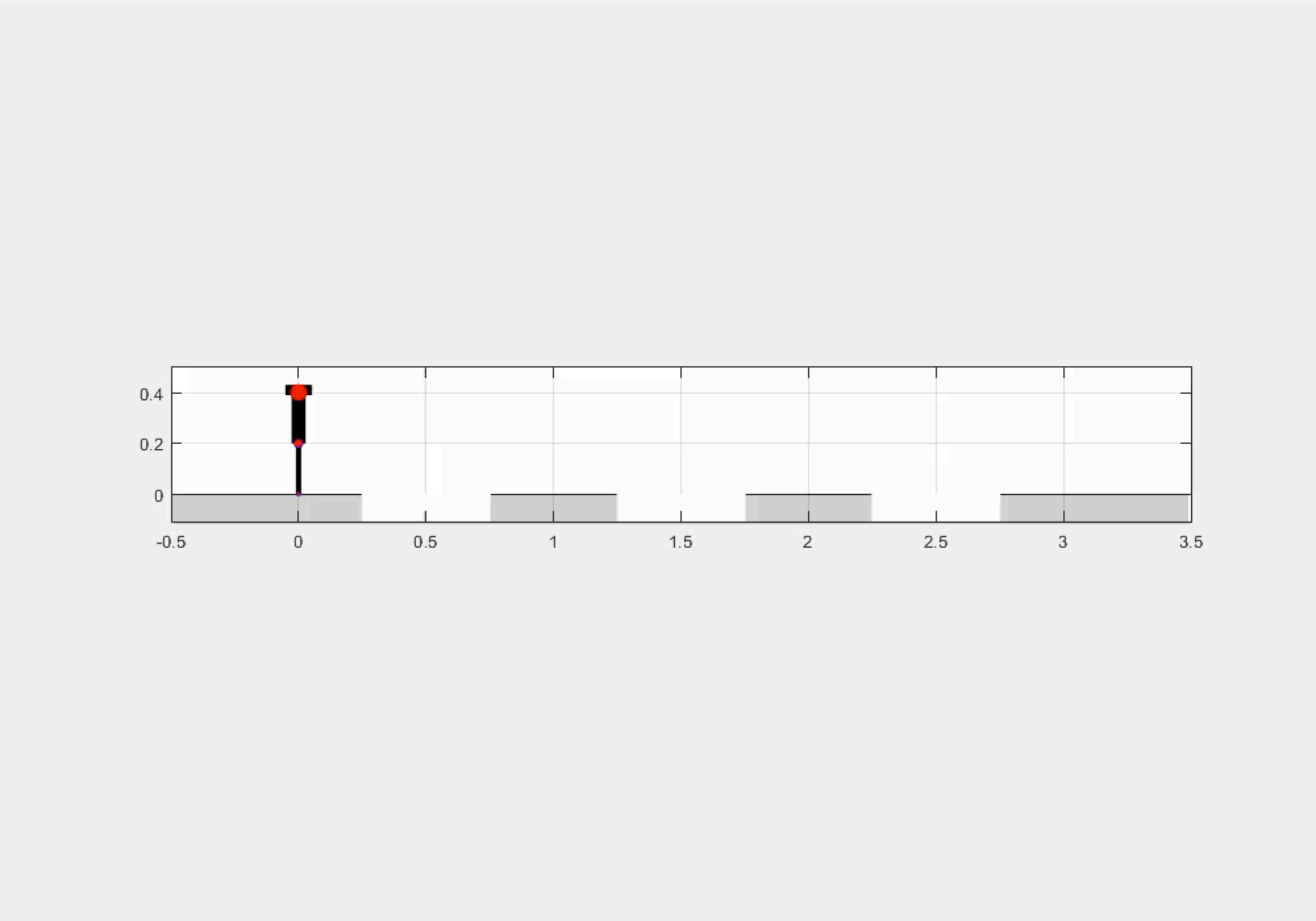
After the first homotopy iteration



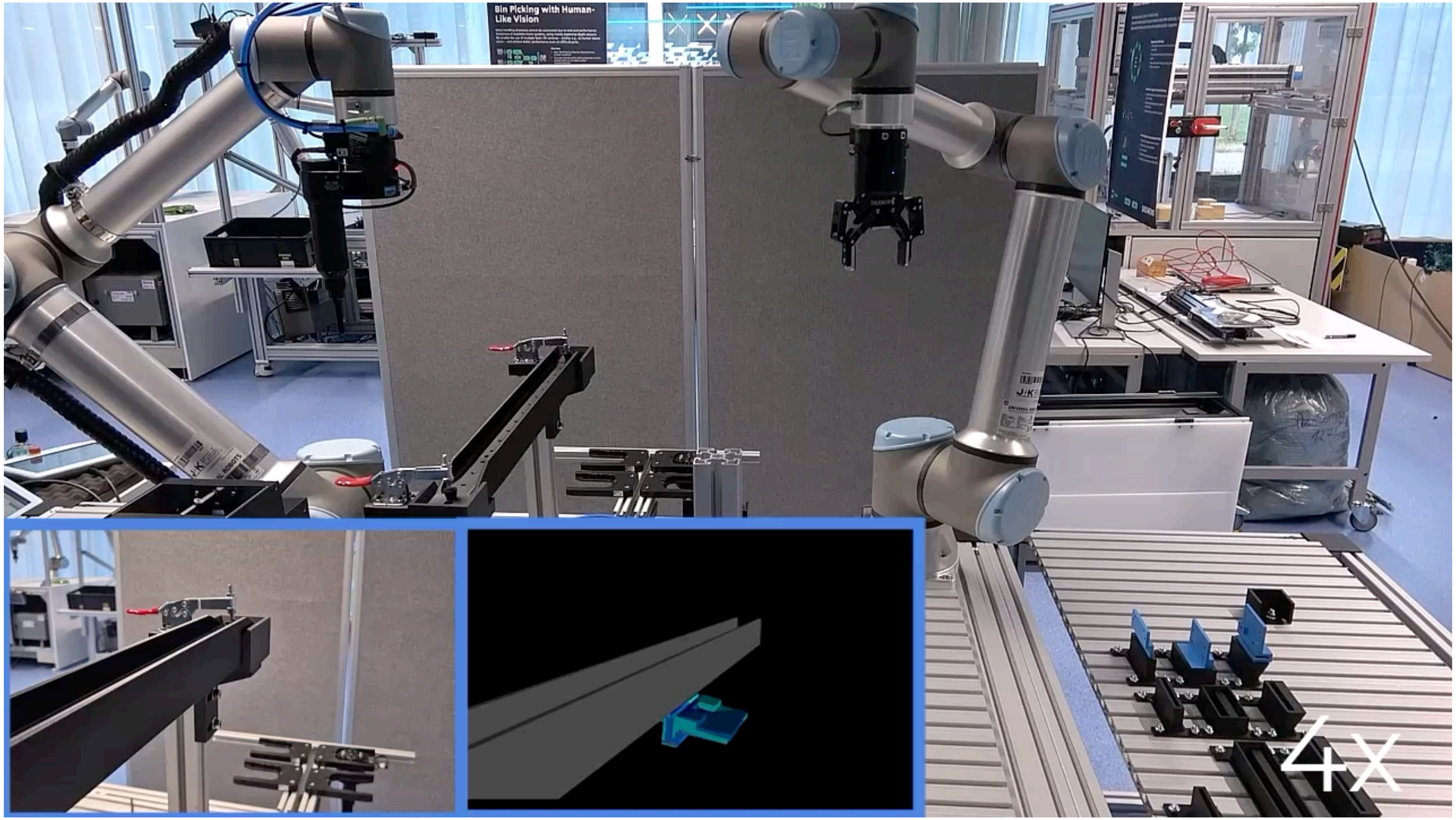
The solution trajectory after convergence

# Hopping robot - move with minimal effort from start to end position

Homotopy initialized with start position everywhere. Optimizer finds creative solution.



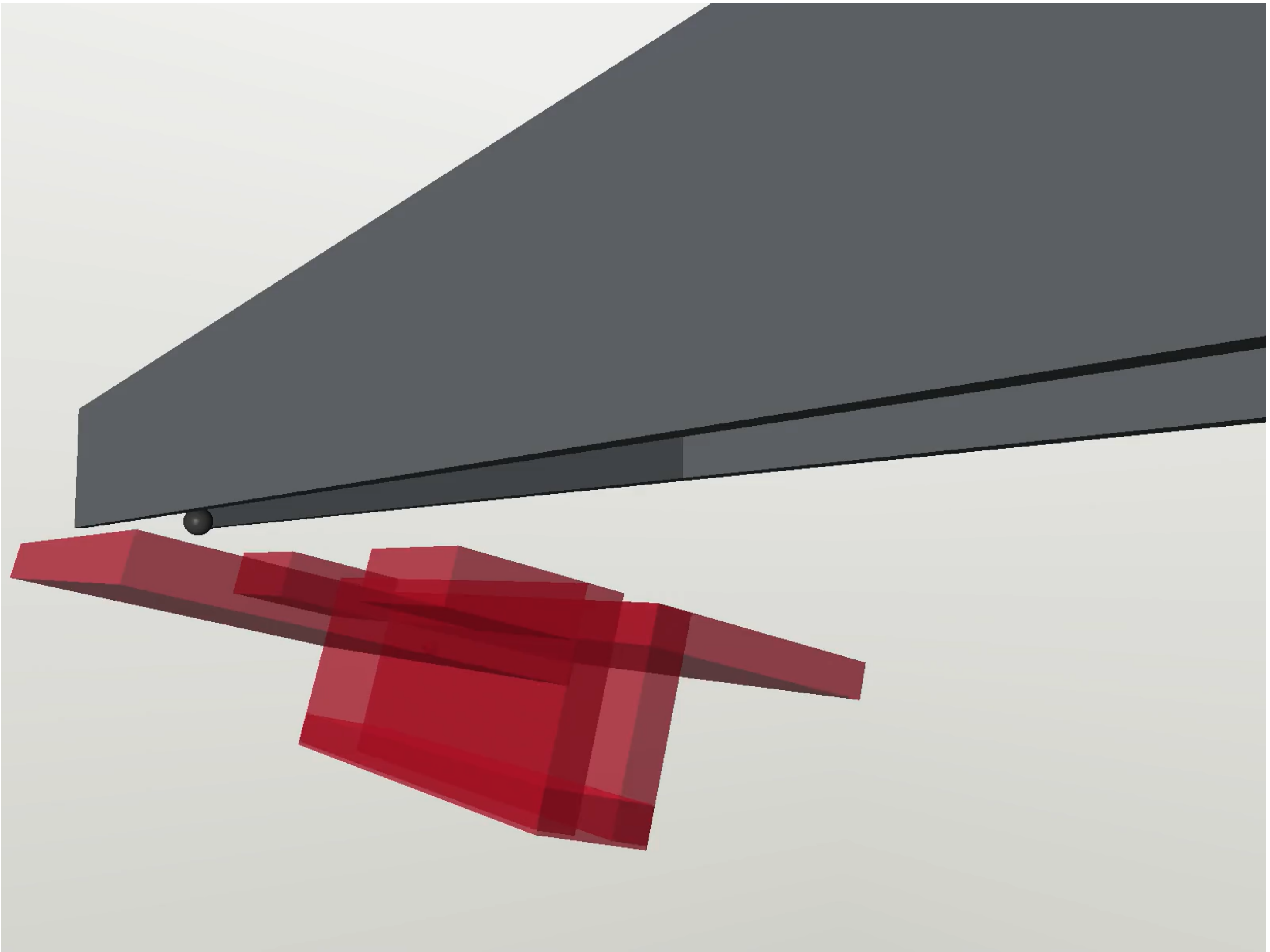
# Real-World Application: Assembly Robots at Siemens in Munich (NSD3)



Christian Dietz (MSc Mathematics) industrial PhD student at University of Freiburg, supervised by Armin Nurkanovic, Sebastian Albrecht, and MD

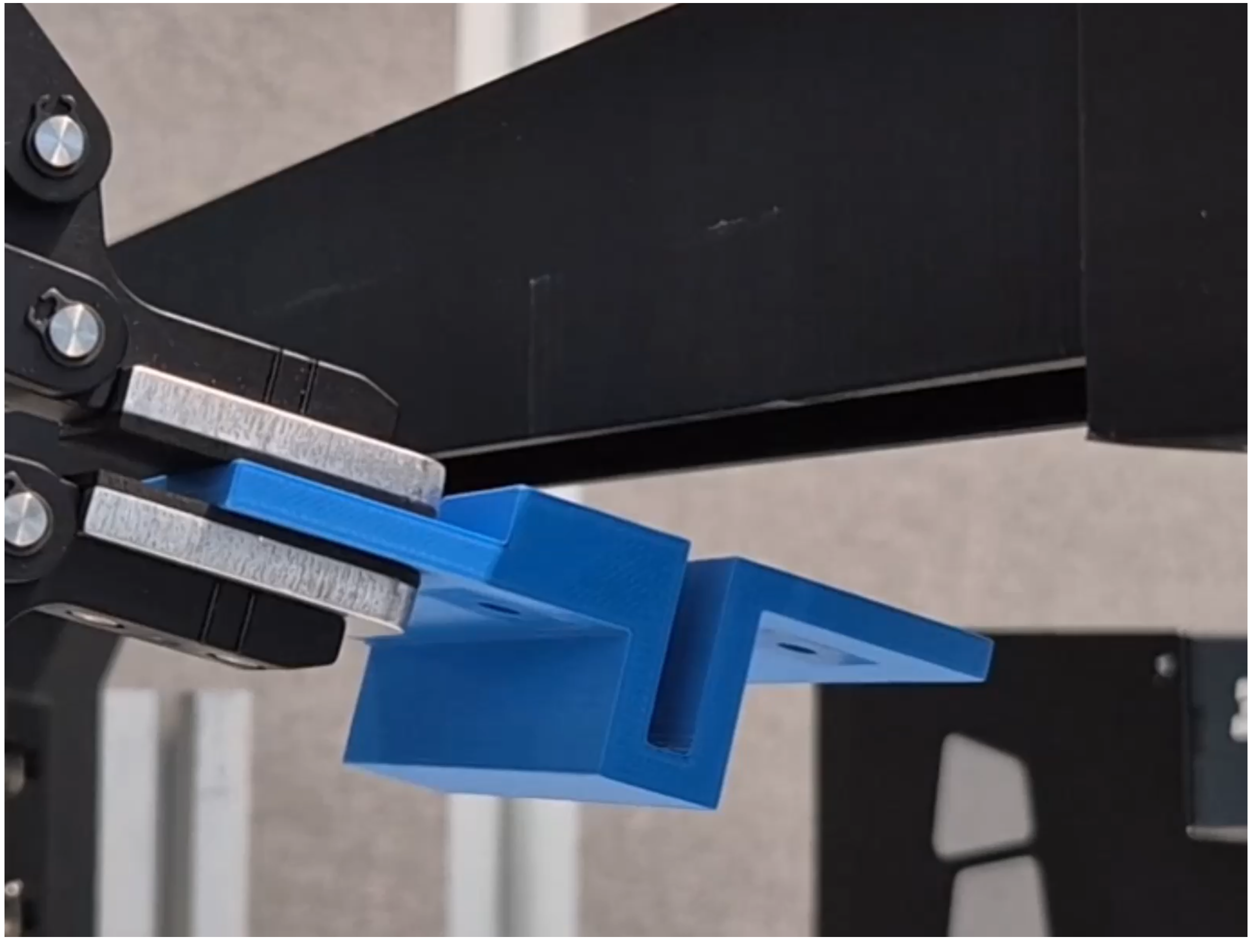
# Dream: Use Optimal Control to Move Robot to Desired End Position

(simulated solution of optimal control problem, L2-control penalty)



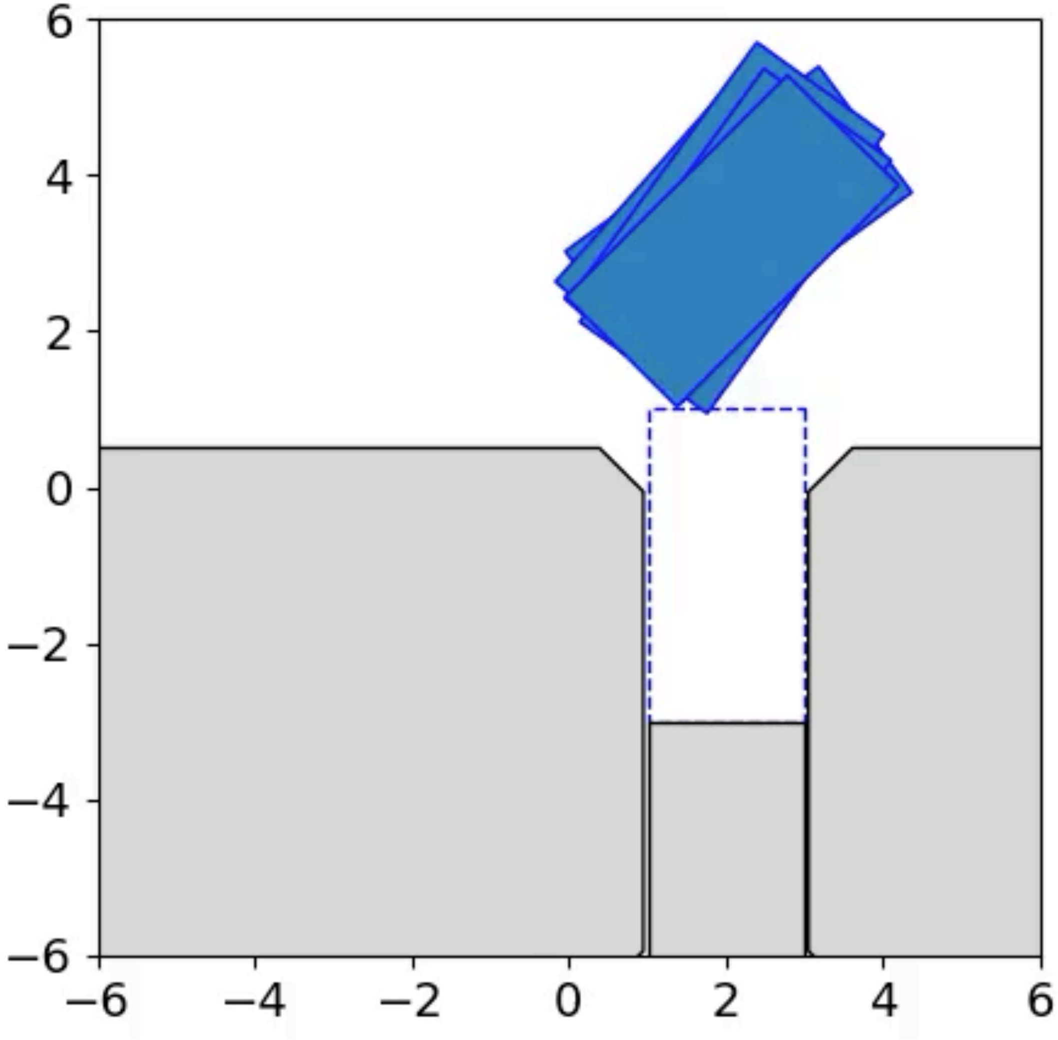
# Dream: Use Optimal Control to Move Robot to Desired End Position

(experiment)

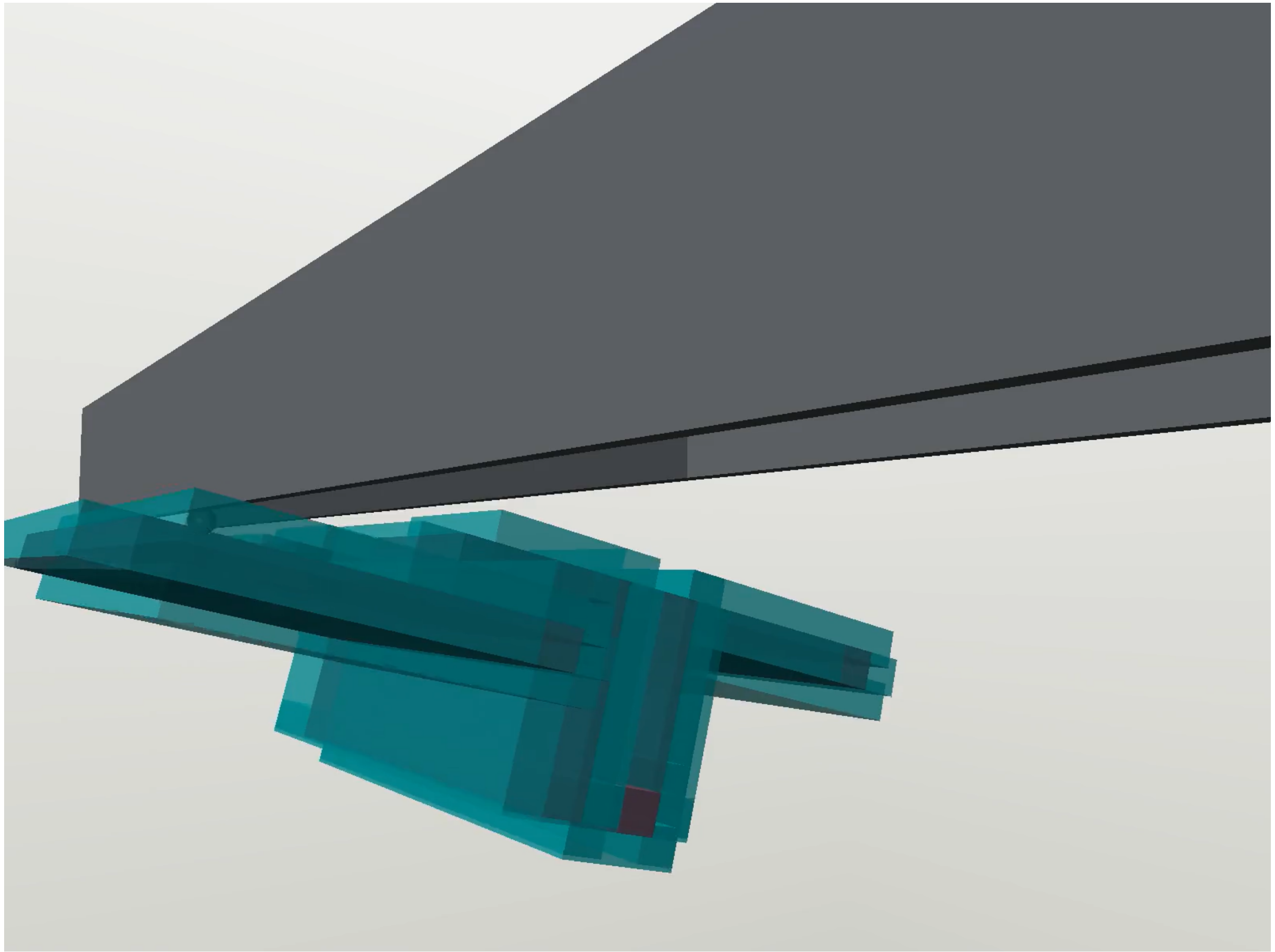


# Dream: Use Optimal Control to Move Robot to Desired End Position
(experiment)

# Idea: robustify motions by optimizing several reference trajectories simultaneously



# Robust Optimal Control Solution (5 scenarios)
(simulation)

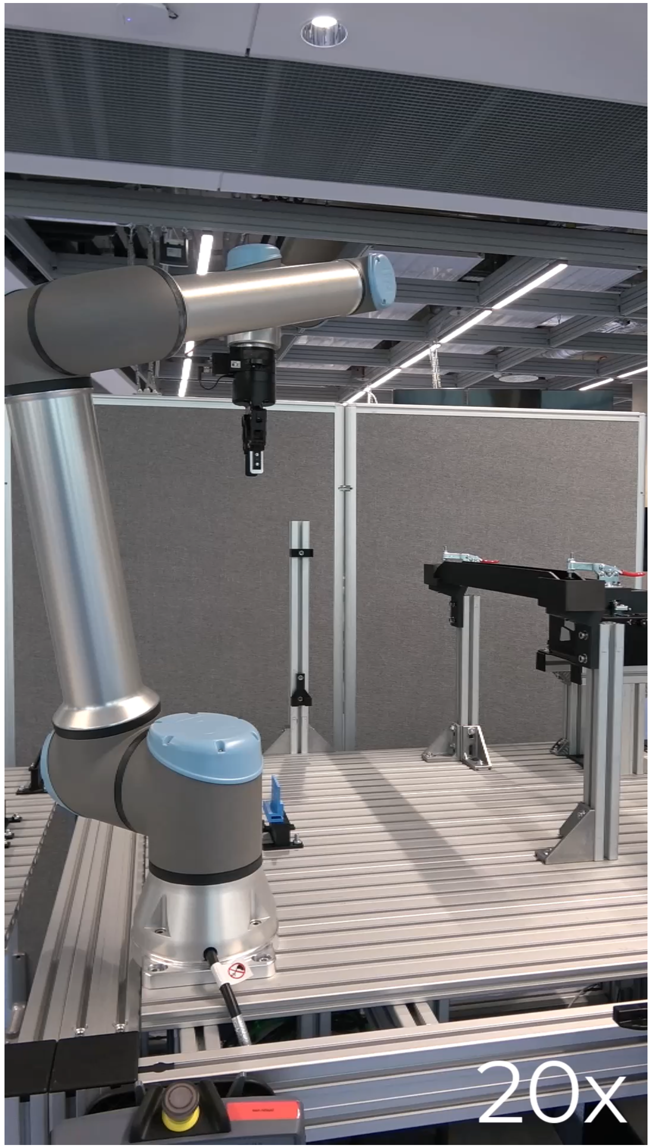


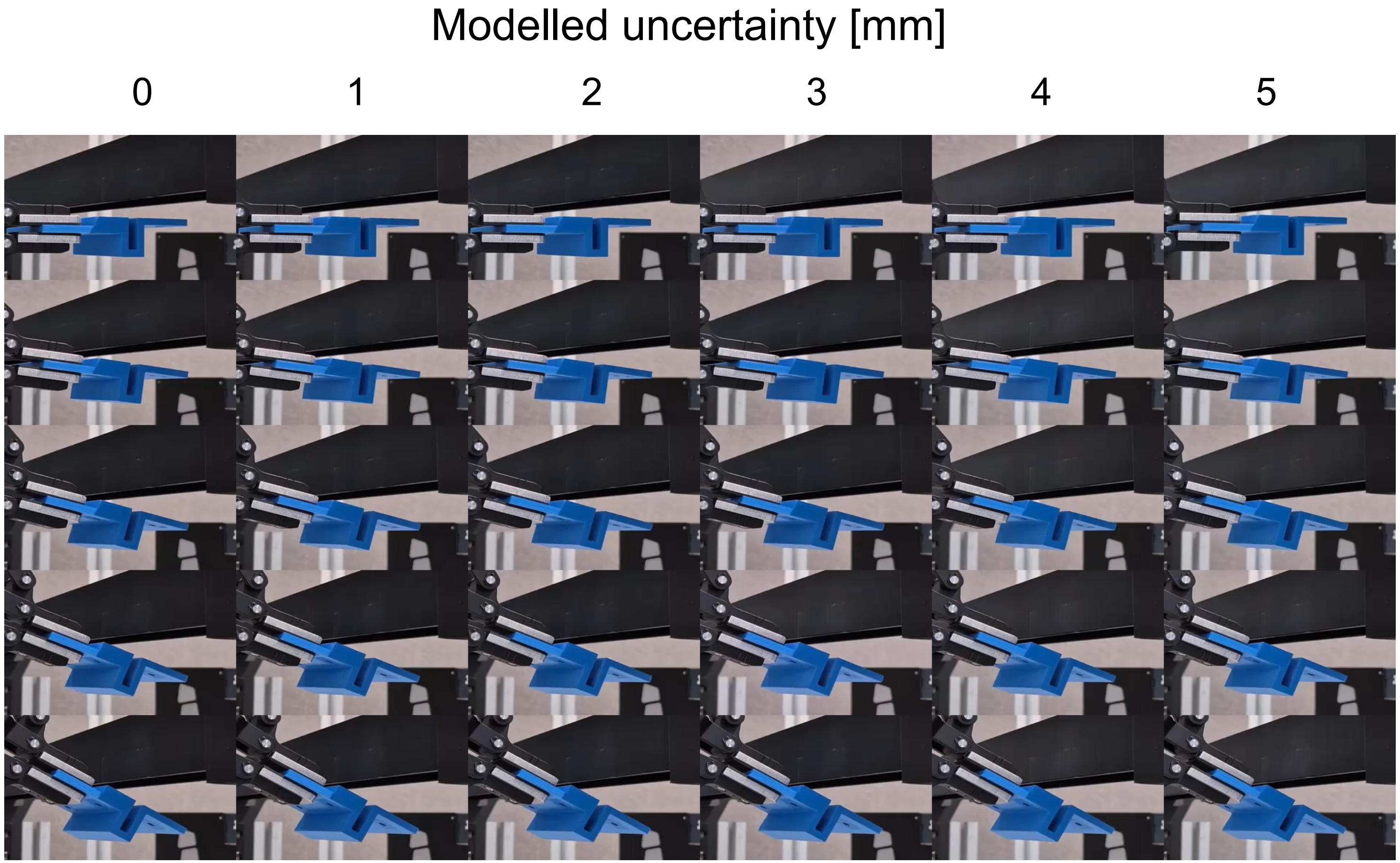
# Robust Optimal Control Solution (5 scenarios)

(experiment)

# Tracking references on real robotic system

with artificially introduced model-reality mismatch of 3mm





# Conclusions

- Newton-type optimization can address seemingly combinatorial optimization problems in nonsmooth optimal control (recent advances are time freezing and FESD)
- Mathematical Programs with Complementarity Constraints (MPCC) are a powerful tool for “disciplined nonsmooth programming”
- Derivatives remain a crucial optimization ingredient also when the nonconvexity of problems increases

Thank you!

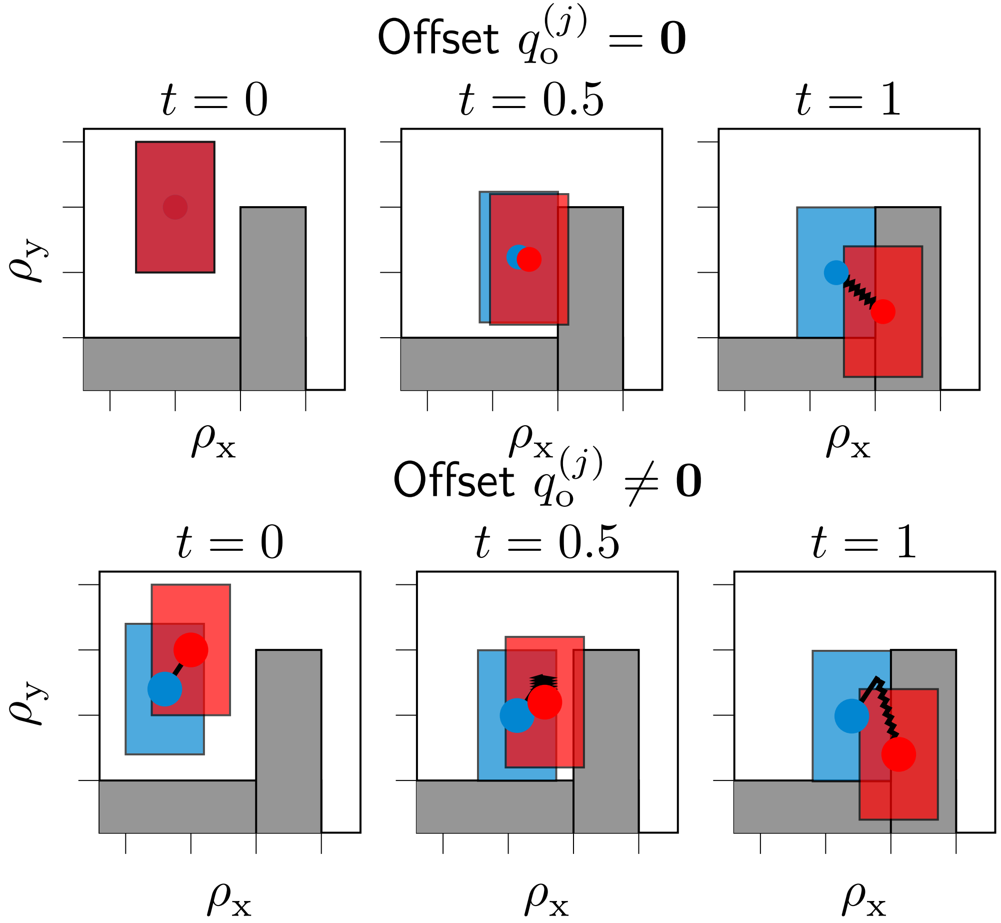
# APPENDIX 1 - Details on Siemens Assembly Robot Optimization

# Impedance law for reliable motion execution on real systems

- To achieve closed-loop execution on a real system, we utilize an impedance law as control strategy
- The goal of the planning algorithm is to determine a desired trajectory which results in robust assembly motions if it is tracked by the impedance controller
- For a given desired trajectory $x_{\mathrm{d}} = (q_{\mathrm{d}}, \nu_{\mathrm{d}})$, a trajectory $x_{\mathrm{e}}^{(j)} = (q_{\mathrm{e}}^{(j)}, \nu_{\mathrm{e}}^{(j)})$ in the ensemble is controlled by the impedance force

$$u_j = D(\nu_{\mathrm{d}} - \nu_{\mathrm{e}}^{(j)}) + K((q_{\mathrm{d}} \oplus q_{\mathrm{o}}^{(j)}) \ominus q_{\mathrm{e}}^{(j)}),$$

with gain matrices $D, K$ and a fixed offset $q_{\mathrm{o}}^{(j)}$.



*Blue rectangle represents $q_{\mathrm{e}}^{(j)}$, red rectangle $q_{\mathrm{d}}$.*

# How to formulate and solve OCP for assembly robot at Siemens?

1. Divide colliding bodies each into rigidly connected convex polyhedra
2. Define Signed Distance Function (SDF) between polyhedra
3. Compute Contact Normal of SDF (unique if slightly smoothed)
4. Formulate Complementarity Lagrangian System Model (NSD3)
5. Discretize OCP with Time-Stepping Method (Implicit Euler, Fixed Steps)
6. Solve resulting Mathematical Program with Complementarity Constraints (MPCC) via Scholtes' Relaxation Method
7. Play open-loop control trajectory on real robot

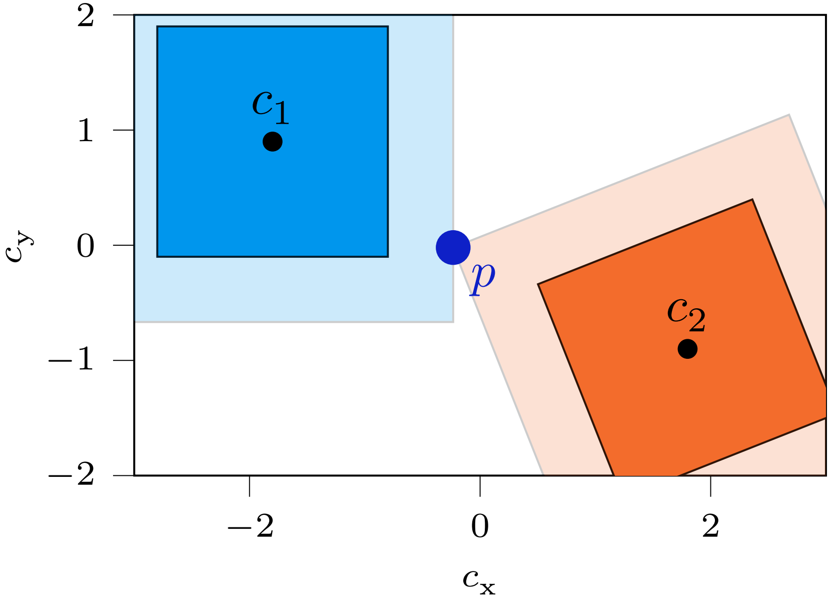
# How to formulate and solve OCP for assembly robot at Siemens?

1. Divide colliding bodies each into rigidly connected convex polyhedra
2. **Define Signed Distance Function (SDF) between polyhedra**
3. **Compute Contact Normal of SDF (unique if slightly smoothed)**
4. Formulate Complementarity Lagrangian System Model (NSD3)
5. Discretize OCP with Time-Stepping Method (Implicit Euler, Fixed Steps)
6. Solve resulting Mathematical Program with Complementarity Constraints (MPCC) via Scholtes' Relaxation Method
7. Play open-loop control trajectory on real robot

# Optimization-based signed distance function (SDF) for polytopes

Halfspace representation of polytopes for $n_\mathrm{w} \in \{2, 3\}$:

$$\mathcal{P}_1 = \{p \in \mathbb{R}^{n_\mathrm{w}} \mid G_1 p \leq h_1\},\ \mathcal{P}_2 = \{p \in \mathbb{R}^{n_\mathrm{w}} \mid G_2 p \leq h_2\}.$$



Associating degrees of freedom:

- $\rho_i$ center of mass of $i$-th polytope
- $\xi_i$ orientation of $i$-th polytope
- System configuration: $q = (\rho_1, \xi_2, \rho_2, \xi_2)$
- $R(\xi_i)$ - rotation matrices

Calculating the SDF as growth distance:

$$\begin{aligned} \Phi_0(q) = \min_{p,\alpha} \quad & \alpha \\ \text{s.t.} \quad & G_1 R(\xi_1)^\top (p - \rho_1) \leq (1+\alpha) h_1, \\ & G_2 R(\xi_2)^\top (p - \rho_2) \leq (1+\alpha) h_2. \end{aligned}$$

# Smoothing the signed distance function

The optimization-based SDF is given by a parametric linear program

$$\begin{aligned}\Phi_0(q) = \min_{z} \quad & c^\top z \\ \text{s.t.} \quad & A(q)z \leq b(q),\end{aligned}$$

with primal variables $z = (p, \alpha)$.
Perturbed KKT conditions as considered in interior-point methods with barrier parameter $\tau > 0$ are given by

$$\begin{aligned} 0 &= c + A(q)^\top \lambda, \\ y &= b(q) - A(q)z, \\ \lambda_i y_i &= \tau, \quad i = 1, \dots, m, \\ \lambda &> \mathbf{0}, y > \mathbf{0}, \end{aligned}$$

$\lambda$ are Lagrange multipliers and $y$ are inequality constraint slacks.

# Smoothing the signed distance function (1)

By writing the equality conditions compactly the perturbed KKT conditions are denoted by

$$
\begin{aligned}
F_\tau(\gamma; q) &= \mathbf{0}, \\
\lambda > \mathbf{0}, y &> \mathbf{0},
\end{aligned}
$$

with primal, dual and slack variables $\gamma = (z, \lambda, y)$.

## Proposition

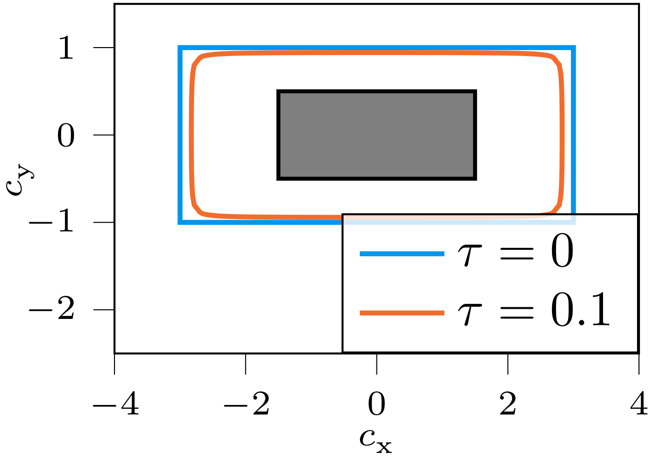
*The solution $\gamma_\tau = (z_\tau, \lambda_\tau, y_\tau)$ of the perturbed optimality conditions exists and is unique.*[1]

This implies that the distance function defined by

$$
\Phi_\tau(q) = \{\alpha \mid F_\tau(\gamma_\tau; q) = \mathbf{0}, \lambda_\tau > \mathbf{0}, y_\tau > \mathbf{0}\},
$$

is well-defined for $\tau > 0$.

Level lines $\Phi_\tau(q) = 1$, $q$ point mass



[1] C. Dietz, S. Albrecht, A. Nurkanović, M. Diehl. *Smoothed Distance Functions for Direct Optimal Control of Contact-Rich Systems*. European Control Conference (ECC) 2025.

# Smoothing the signed distance function (1)

By writing the equality conditions compactly the perturbed KKT conditions are denoted by

$$F_\tau(\gamma; q) = \mathbf{0},$$
$$\lambda > \mathbf{0}, y > \mathbf{0},$$

with primal, dual and slack variables $\gamma = (z, \lambda, y)$.

## Proposition

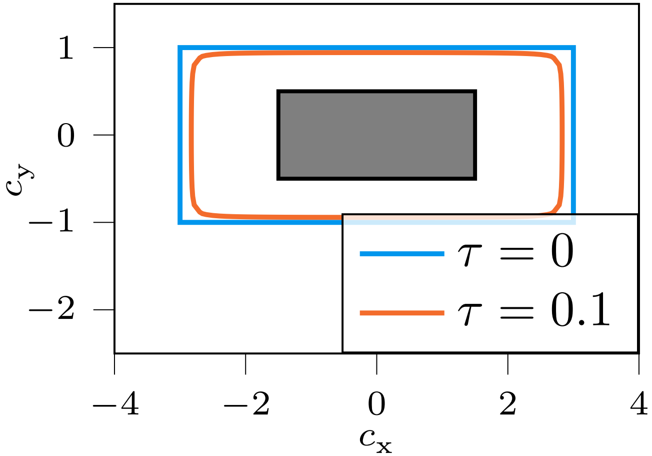
*The solution $\gamma_\tau = (z_\tau, \lambda_\tau, y_\tau)$ of the perturbed optimality conditions exists and is unique.*[1]

This implies that the distance function defined by

$$\Phi_\tau(q) = \{\alpha \mid F_\tau(\gamma_\tau; q) = \mathbf{0}, \lambda_\tau > \mathbf{0}, y_\tau > \mathbf{0}\},$$

is well-defined for $\tau > 0$.

Level lines $\Phi_\tau(q) = 1$, $q$ point mass



**How to obtain the contact normal $\nabla_q \Phi_\tau(q)$ ?**

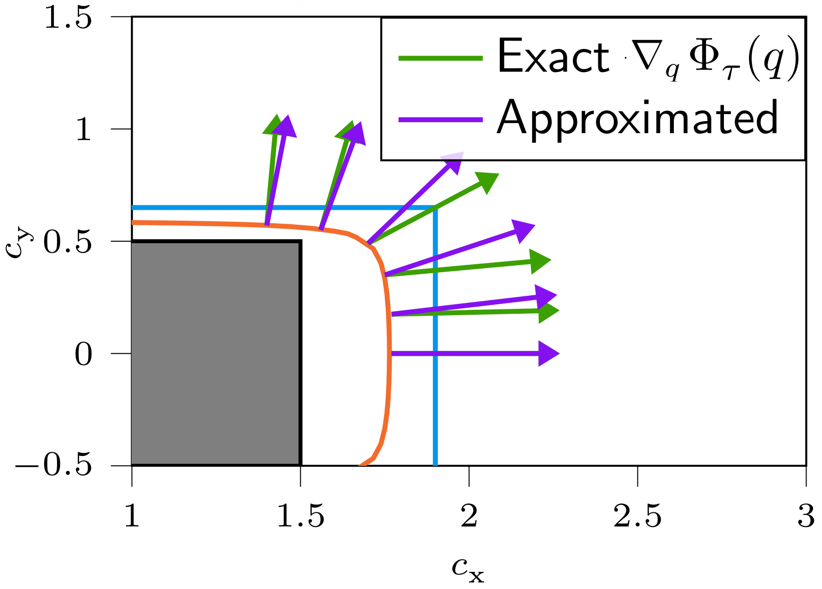
[1] C. Dietz, S. Albrecht, A. Nurkanović, M. Diehl. *Smoothed Distance Functions for Direct Optimal Control of Contact-Rich Systems*. European Control Conference (ECC) 2025.

# Contact normal approximation for the smooth SDF

Proposed approximation: $n_\tau(q) = \dfrac{-\nabla_q y(z_\tau, q)\lambda_\tau}{\|\nabla_q y(z_\tau, q)\lambda_\tau\|_2} \approxeq \nabla_q \Phi_\tau(q)$ (exact for $\tau$=0)

# Contact normal approximation for the smooth SDF

Proposed approximation: $n_\tau(q) = \dfrac{-\nabla_q y(z_\tau, q)\lambda_\tau}{\|\nabla_q y(z_\tau, q)\lambda_\tau\|_2} \approxeq \nabla_q \Phi_\tau(q)$ (exact for $\tau$=0)



# Contact normal approximation for the smooth SDF

## Recap on definitions

The SDF is given by

$$\begin{aligned}\Phi_0(q) = \min_{z} \quad & c^\top z \\ \text{s.t.} \quad & A(q)z \leq b(q),\end{aligned} \tag{1}$$

with inequality constraint slacks

$$y(z,q) = b(q) - A(q)z.$$

We additionally define

- $\bar{Z}(q)$ denotes the set of all primal optimal solutions to (1)
- $\bar{\Lambda}(q)$ denotes the set of all corresponding dual optimal solutions

- Modelling of contact-rich systems requires definition of a contact normal vector
- Normally the contact normal is chosen as the gradient of the SDF (results in third-order sensitivities in Newton-type optimization!)

Directional derivatives at an exact solution:[2]

$$\partial_d \Phi_0(q) = \min_{z \in \bar{Z}(q)} \max_{\lambda \in \bar{\Lambda}(q)} -d^\top \nabla_q y(z,q)\lambda,$$

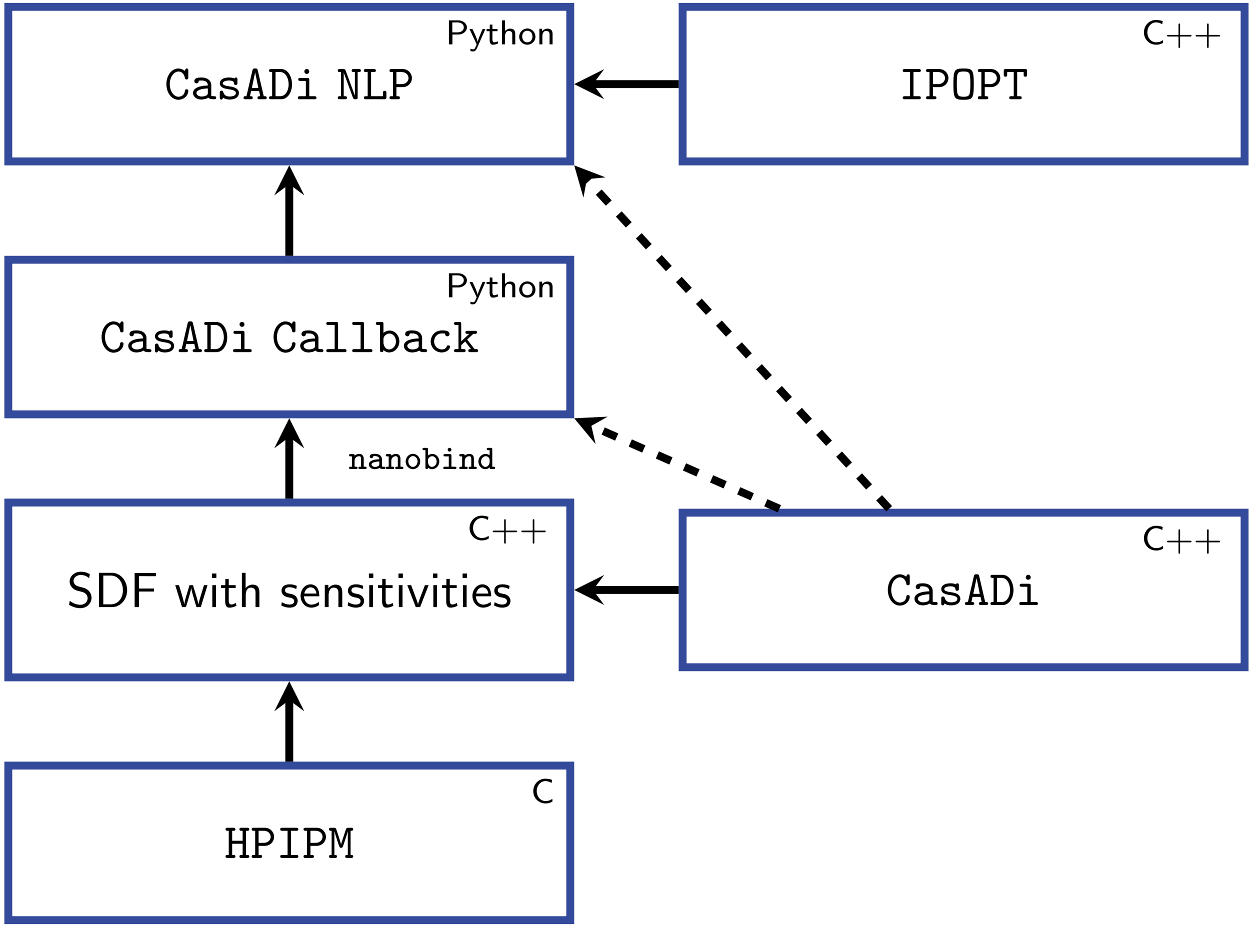
Proposed contact normal approximation:

$$n_\tau(q) = \frac{-\nabla_q y(z_\tau, q)\lambda_\tau}{\|\nabla_q y(z_\tau, q)\lambda_\tau\|_2}.$$

[2] W. Hogan. *Directional derivatives for extremal-value functions with applications to the comple*

# SDF implementation

- Numerical experiments use the CasADi toolbox through its Python interface and IPOPT as solver
- The SDF is specified through CasADi's Callback class
- HPIPM is used to solve the distance problems up to barrier parameter $\tau > 0$
- A C++ wrapper is used to efficiently manage HPIPM structures and parallel computing
- C++ code is interfaced back to Python by using the nanobind library



# How to formulate and solve OCP for assembly robot at Siemens?

1. Divide colliding bodies each into rigidly connected convex polyhedra
2. Define Signed Distance Function (SDF) between polyhedra
3. Compute Contact Normal of SDF (unique if slightly smoothed)
4. **Formulate Complementarity Lagrangian System Model (NSD3)**
5. **Discretize OCP with Time-Stepping Method (Implicit Euler, Fixed Steps)**
6. Solve resulting Mathematical Program with Complementarity Constraints (MPCC) via Scholtes' Relaxation Method
7. Play open-loop control trajectory on real robot

# Robust contact-implicit trajectory optimization

Continuous-time contact-rich dynamical system:

$$
\begin{aligned}
&\dot{q} = \nu, \\
&M\dot{\nu} = u + \sum_{i=1}^{n_{\mathrm{d}}} n_{\tau,i}(q)\lambda_{\mathrm{n},i}, \\
&0 \leq \Phi_{\tau,i}(q) \ \perp \ \lambda_{\mathrm{n},i} \geq 0, \quad i = 1, \dots, n_{\mathrm{d}}, \\
&\text{Multi impact law.}
\end{aligned}
$$

- $\nu \in \mathbb{R}^{n_{\mathrm{q}}}$ system velocity
- $M \in \mathbb{R}^{n_{\mathrm{q}} \times n_{\mathrm{q}}}$ inertia matrix
- $u \in \mathbb{R}^{n_{\mathrm{u}}}$ control input
- $n_{\mathrm{d}} \in \mathbb{N}$ object pairs with smooth SDF $\Phi_{\tau,i}$ and corresponding contact normals $n_{\tau,i}$

# Implicit-Euler time-stepping discretization

Time-stepping discretization:

$$
\begin{aligned}
& q_{k+1} = q_k + h\nu_{k+1}, \\
& \nu_{k+1} = \nu_k + hM^{-1}(u_k + \sum_{i=1}^{n_\mathrm{d}} n_{\tau,i}(q_{k+1})\lambda_{\mathrm{n},k,i}), \\
& \Phi_{\tau,i}(q_{k+1})\lambda_{\mathrm{n},k,i} \leq \sigma,\ \ i = 1, \ldots, n_\mathrm{d}, \\
& 0 \leq \Phi_{\tau,i}(q_{k+1}),\ 0 \leq \lambda_{\mathrm{n},k,i},\ \ i = 1, \ldots, n_\mathrm{d},
\end{aligned}
$$

with time-step $h > 0$ and using Scholtes' relaxation to relax complementarity constraints with $\sigma > 0$.

# Discretization and cost function for robust motion generation

Contact-rich system with quaternion dynamics:

$$
\begin{aligned}
&\dot{q}_{\mathrm{d}} = Q(q_{\mathrm{d}})\nu_{\mathrm{d}}, \\
&\text{For } j = 1, \dots, n_{\mathrm{s}} : \\
&\dot{q}_{\mathrm{e}}^{(j)} = Q(q_{\mathrm{e}}^{(j)})\nu_{\mathrm{e}}^{(j)}, \\
&M\dot{\nu}_{\mathrm{e}}^{(j)} = u_j + \sum_{i=1}^{n_{\mathrm{d}}} Q(q_{\mathrm{e}}^{(j)})^\top n_{\tau,i}(q_{\mathrm{e}}^{(j)})\lambda_{\mathrm{n},i}^{(j)}, \\
&\lambda_{\mathrm{n},i}^{(j)}\Phi_{\tau,i}(q_{\mathrm{e}}^{(j)}) \leq \sigma,\ i = 1, \dots, n_{\mathrm{d}} \\
&0 \leq \lambda_{\mathrm{n},i}^{(j)},\ 0 \leq \Phi_{\tau,i}(q_{\mathrm{e}}^{(j)}),\ i = 1, \dots, n_{\mathrm{d}}, \\
&u_j = D(\nu_{\mathrm{d}} - \nu_{\mathrm{e}}^{(j)}) + K((q_{\mathrm{d}} \oplus q_{\mathrm{o}}^{(j)}) \ominus q_{\mathrm{e}}^{(j)}),
\end{aligned}
$$

- Discretization through $N_{\mathrm{cnt}}$ intervals with $N_{\mathrm{sim}}$ simulation intervals per control interval
- Total simulation steps $N_{\mathrm{tot}} = N_{\mathrm{cnt}} N_{\mathrm{sim}}$
- On each simulation interval an implicit Euler time-stepping discretization is utilized
- On each control interval a constant $\nu_{\mathrm{d},k},\ k = 1, \dots, N_{\mathrm{cnt}}$ is used
- Cost function for terminal state $\bar{x} = (\bar{q}, \bar{\nu})$:

$$
\begin{aligned}
\mathrm{cost} = &\sum_{k=1}^{N_{\mathrm{cnt}}} 0.001\|\nu_{\mathrm{d},k,\mathrm{trs}}\|_2^2 + 0.01\|\nu_{\mathrm{d},k,\mathrm{ang}}\|_2^2 \\
&+ 1\|\bar{\rho} - \rho_{\mathrm{d},N_{\mathrm{tot}}}\|_2^2 + 10(1 - (\bar{\xi}^\top \xi_{\mathrm{d},N_{\mathrm{tot}}})^2) \\
&+ \sum_{j=1}^{n_{\mathrm{e}}} 100\|\bar{\rho} - \rho_{\mathrm{e},N_{\mathrm{tot}}}^{(j)}\|_2^2 + 1000(1 - (\bar{\xi}^\top \xi_{\mathrm{e},N_{\mathrm{tot}}}^{(j)})^2)
\end{aligned}
$$

# Contact-rich dynamics in three-dimensional space

Contact-rich system with quaternion dynamics:

$$\dot{q}_{\mathrm{d}} = Q(q_{\mathrm{d}})\nu_{\mathrm{d}},$$

For $j = 1, \ldots, n_{\mathrm{s}}$ :

$$\dot{q}_{\mathrm{e}}^{(j)} = Q(q_{\mathrm{e}}^{(j)})\nu_{\mathrm{e}}^{(j)},$$

$$M\dot{\nu}_{\mathrm{e}}^{(j)} = u_j + \sum_{i=1}^{n_{\mathrm{d}}} Q(q_{\mathrm{e}}^{(j)})^\top n_{\tau,i}(q_{\mathrm{e}}^{(j)})\lambda_{\mathrm{n},i}^{(j)},$$

$$\lambda_{\mathrm{n},i}^{(j)}\Phi_{\tau,i}(q_{\mathrm{e}}^{(j)}) \leq \sigma,\ \ i = 1, \ldots, n_{\mathrm{d}}$$

$$0 \leq \lambda_{\mathrm{n},i}^{(j)},\ \ 0 \leq \Phi_{\tau,i}(q_{\mathrm{e}}^{(j)}),\ \ i = 1, \ldots, n_{\mathrm{d}},$$

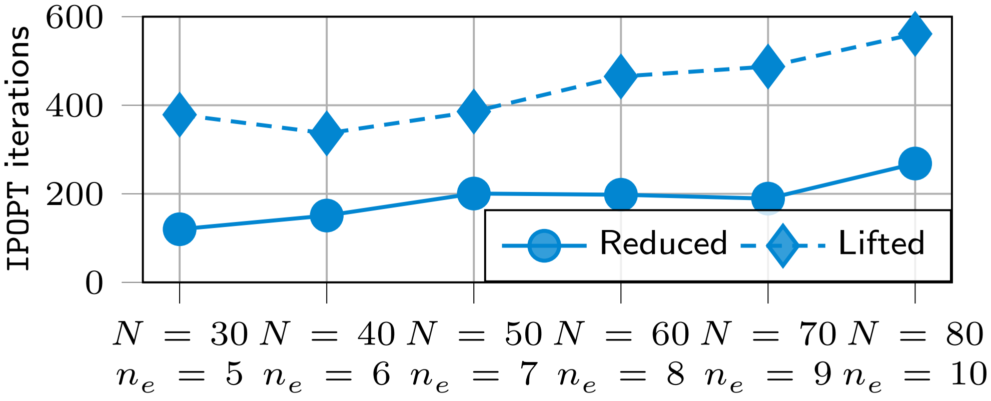
$$u_j = D(\nu_{\mathrm{d}} - \nu_{\mathrm{e}}^{(j)}) + K((q_{\mathrm{d}} \oplus q_{\mathrm{o}}^{(j)}) \ominus q_{\mathrm{e}}^{(j)}),$$

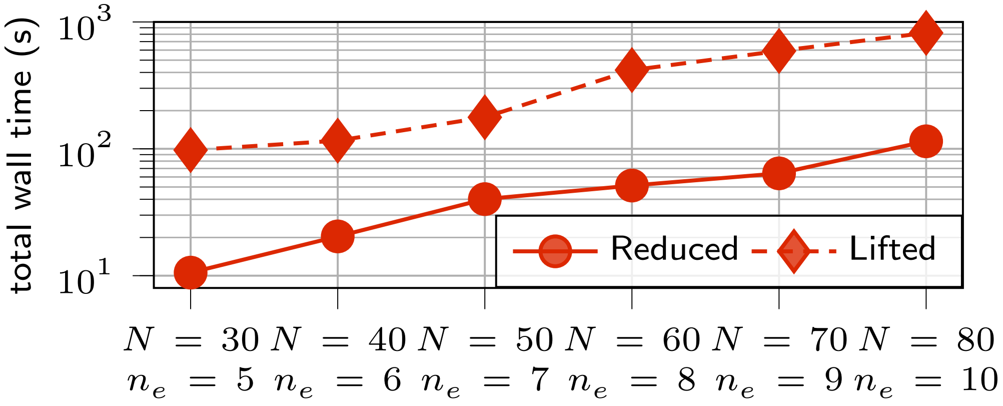
- Position $q = (\rho, \xi)$, with $\rho \in \mathbb{R}^3$ translational position and $\xi \in \mathbb{R}^4$ quaternion orientation

- $q_{\mathrm{d}}, \nu_{\mathrm{d}}$ desired position and velocity (control input)
- $q_{\mathrm{e},j}, \nu_{\mathrm{e},j}$ position and velocity of particle $j$
- $Q(\cdot)$ $7 \times 6$ matrix required to describe quaternion dynamics
- $D, K$ gain matrices, describe the spring-damper behaviour of the feedback controller
- $q_{\mathrm{o}}^{(j)}$ fixed offsets required to achieve robustness
- Quaternion multiplication is denoted by $\oplus$ as well as $\ominus$ for multiplication with conjugation

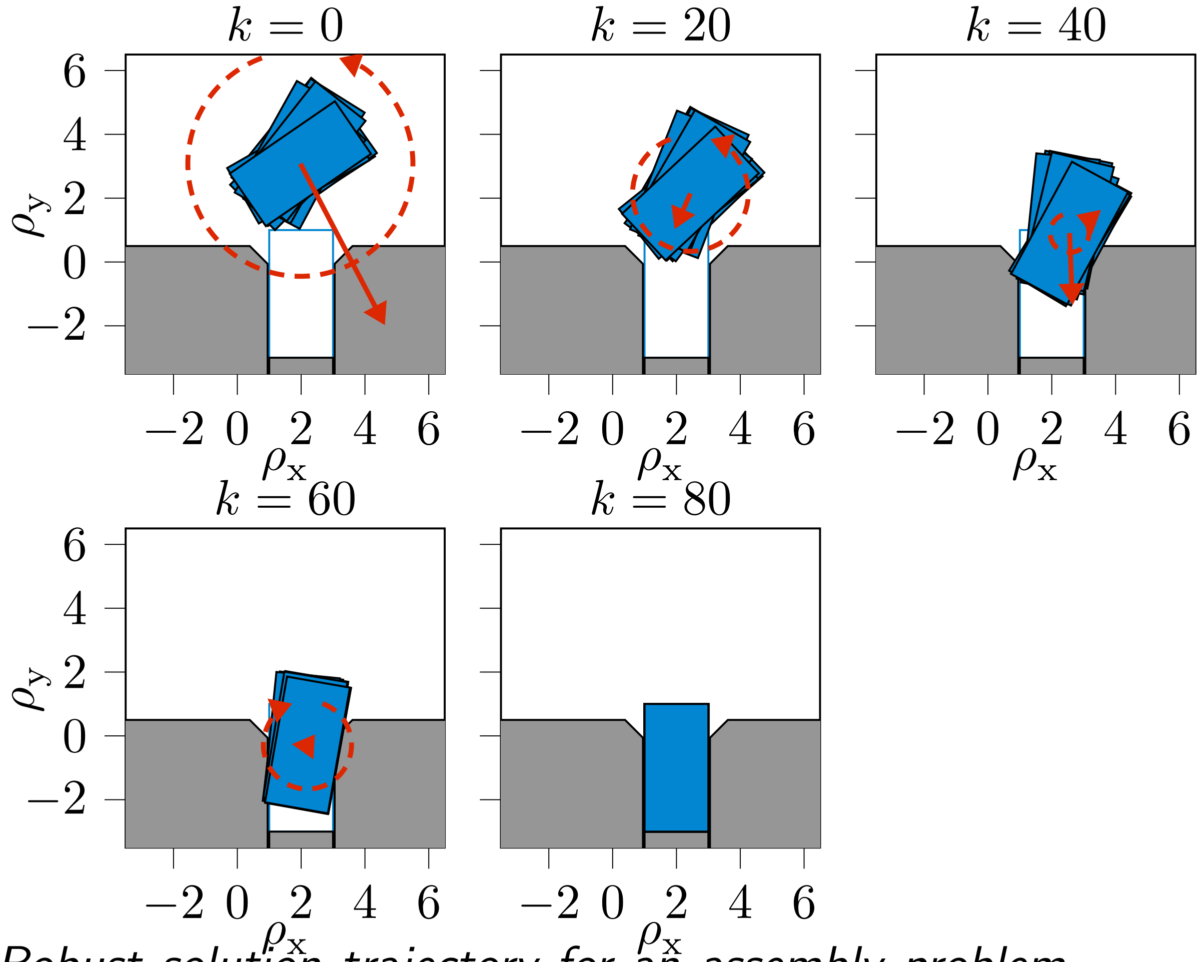
# Computational performance comparison of reduced and lifted SDF implementations

- The SDF $\Phi_{\tau,i}$ can be either evaluated as proposed by using HPIPM or by adding the perturbed KKT conditions directly in the optimal control problem (reduced or lifted implementation)
- We compare computational performance on a two-dimensional peg-in-hole problem for different trajectory lengths $N$ and number of simultaneously simulated trajectories $n_\mathrm{e}$
- Using the reduced modelling with external SDF evaluation results in less IPOPT iterations and less total wall time for all considered problem sizes





# Example robust trajectory for peg in hole



*Robust solution trajectory for an assembly problem. Orange arrows indicate applied control forces $u_k$.*

# Additional References Relevant to Siemens Assembly Robot Problem

*Growth-Distances:*

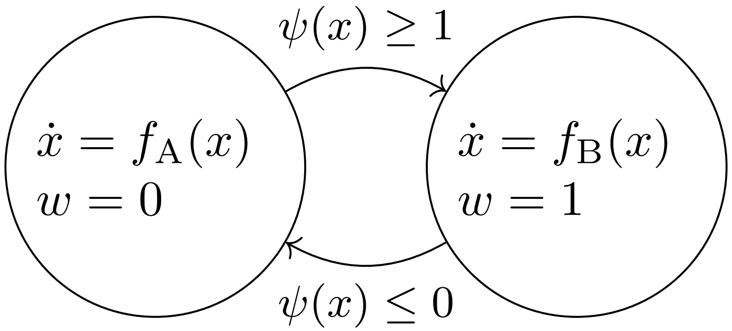
Chong Jin Ong and Elmer G. Gilbert, "Growth distances: New measures for object separation and penetration," in IEEE Transactions on Robotics and Automation, vol. 12, no. 6, pp. 888-903, 1996.

*Ensemble Trajectories for Robustified Robot Control:*

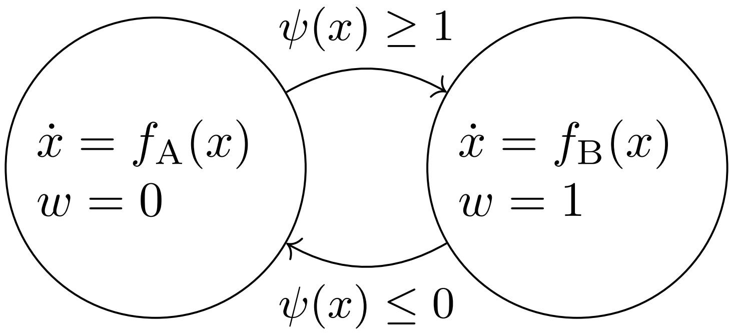
Igor Mordatch, Kendall Lowrey and Emanuel Todorov, "Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5307-5314, 2015.

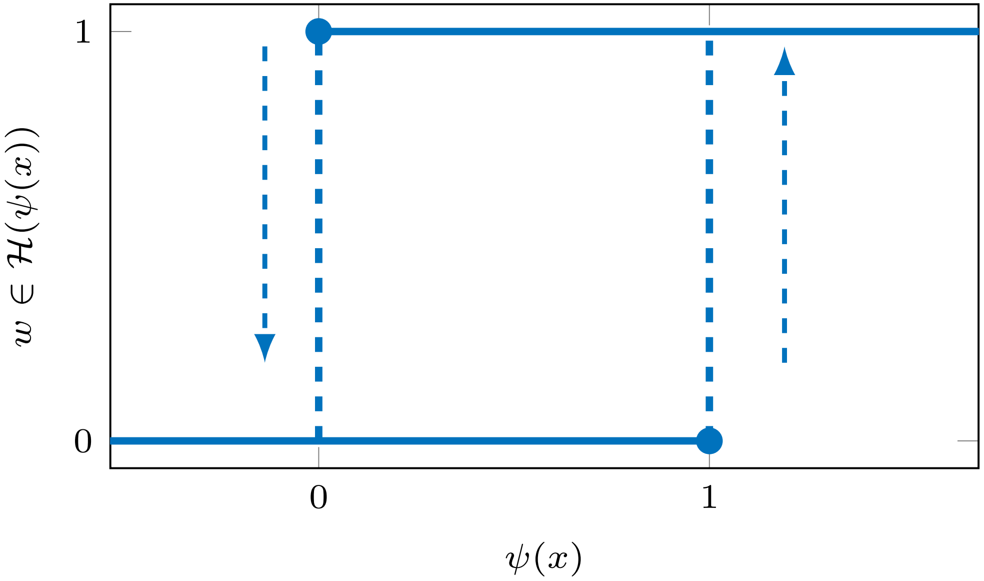
# APPENDIX 2 - Time Freezing for Automata (Hysteresis)

# Hybrid systems and finite automaton



# Hybrid systems and finite automaton

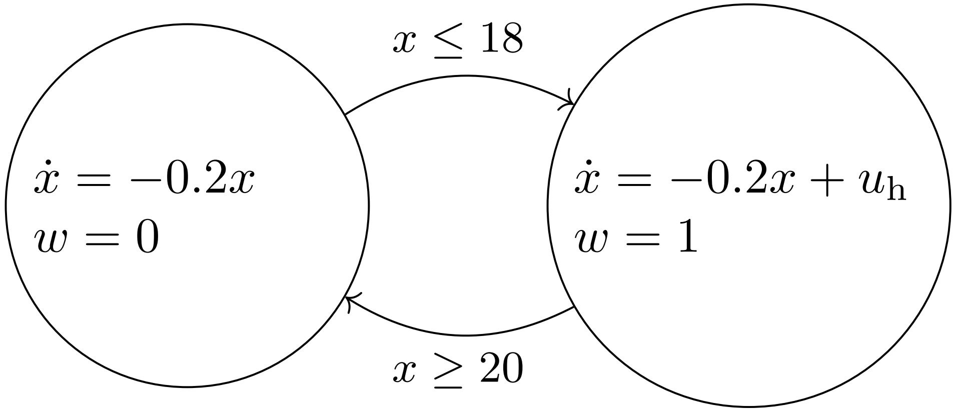




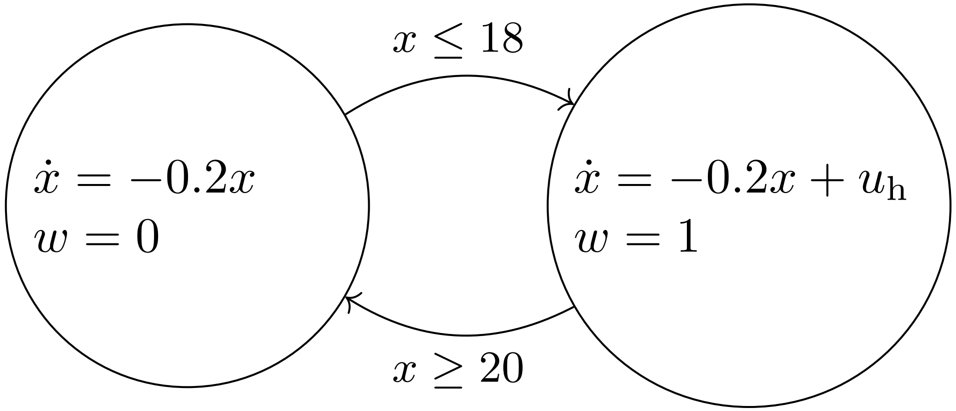
## Hybrid system with hysteresis (*incomplete description*)

$$\dot{x} = f(x, w) = (1 - w) f_{\mathrm{A}}(x) + w f_{\mathrm{B}}(x)$$

# Tutorial example: thermostat with hysteresis



# Tutorial example: thermostat with hysteresis

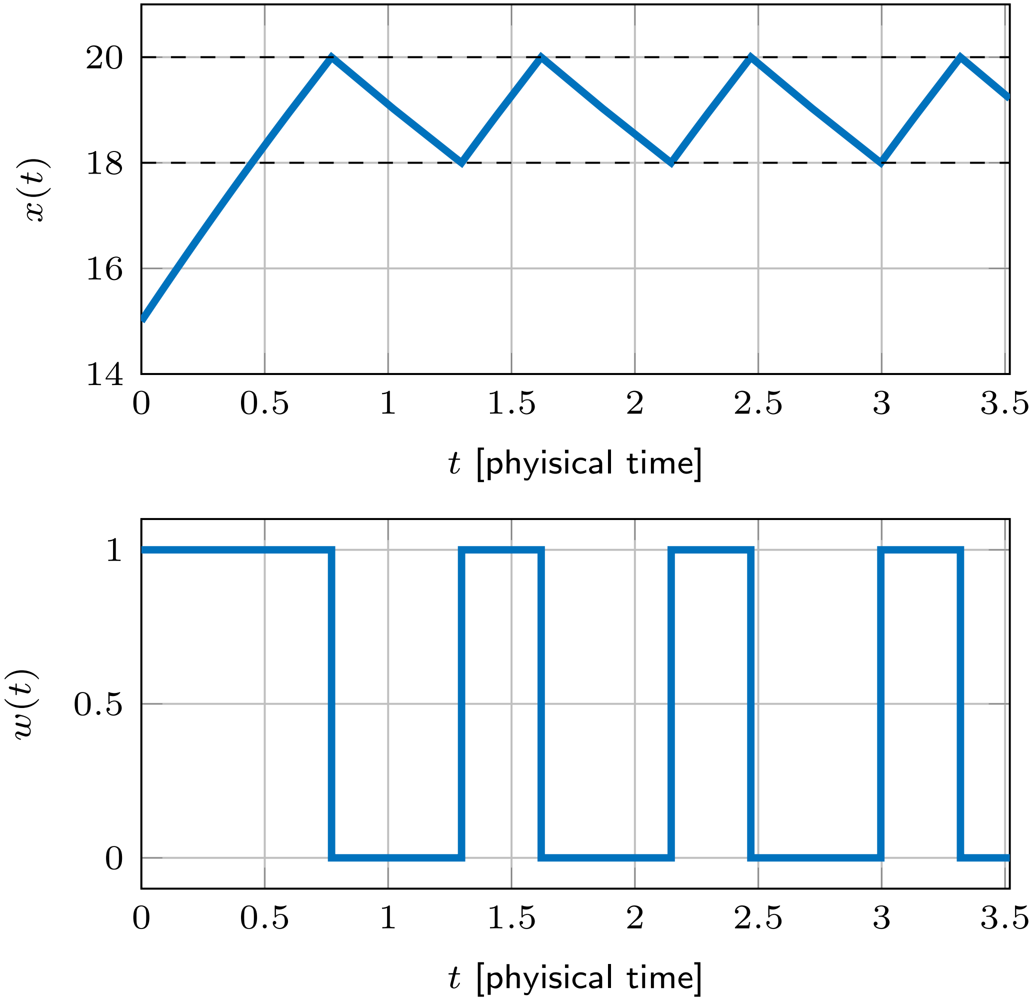


$x \leq 18$

$\dot{x} = -0.2x$
$w = 0$

$\dot{x} = -0.2x + u_{\mathrm{h}}$
$w = 1$

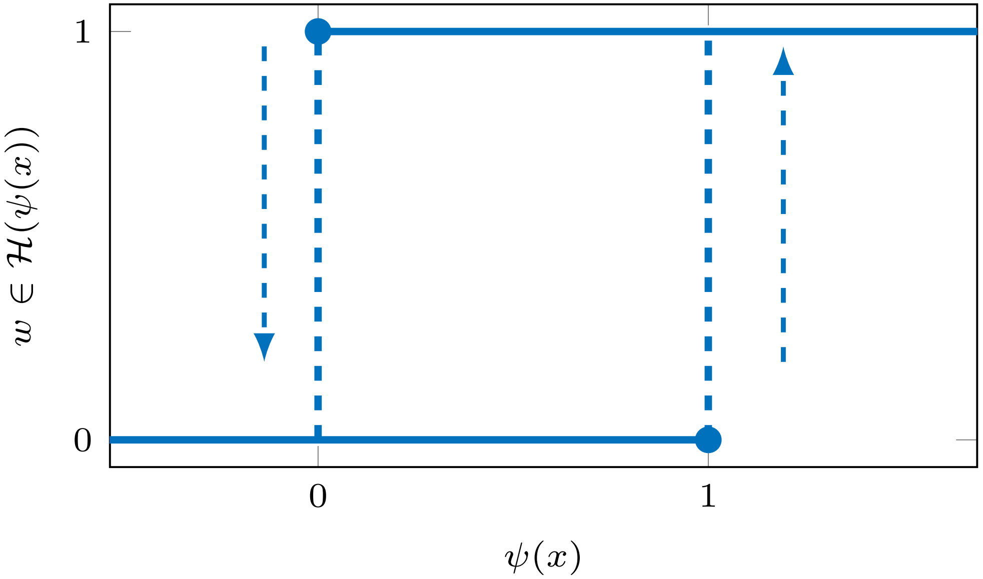
$x \geq 20$



# Hysteresis: a system with state jumps

## Hybrid system with hysteresis

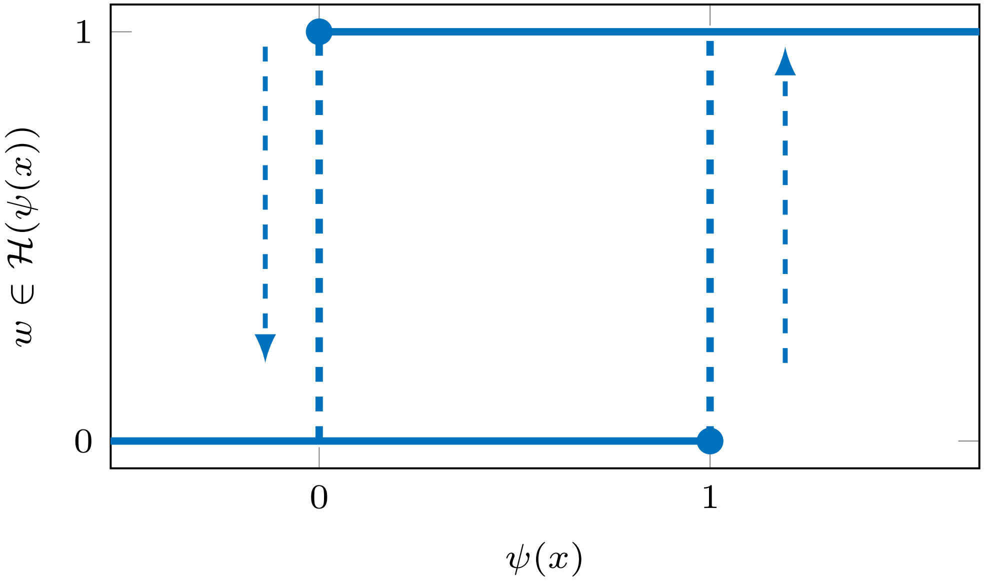
$$
\begin{aligned}
\dot{x} &= f(x,w) = (1-w)f_{\mathrm{A}}(x) + w f_{\mathrm{B}}(x)\\
\dot{w} &= 0
\end{aligned}
$$



# Hysteresis: a system with state jumps

## Hybrid system with hysteresis

$$\dot{x} = f(x,w) = (1-w)f_{\mathrm{A}}(x) + w f_{\mathrm{B}}(x)$$
$$\dot{w} = 0$$

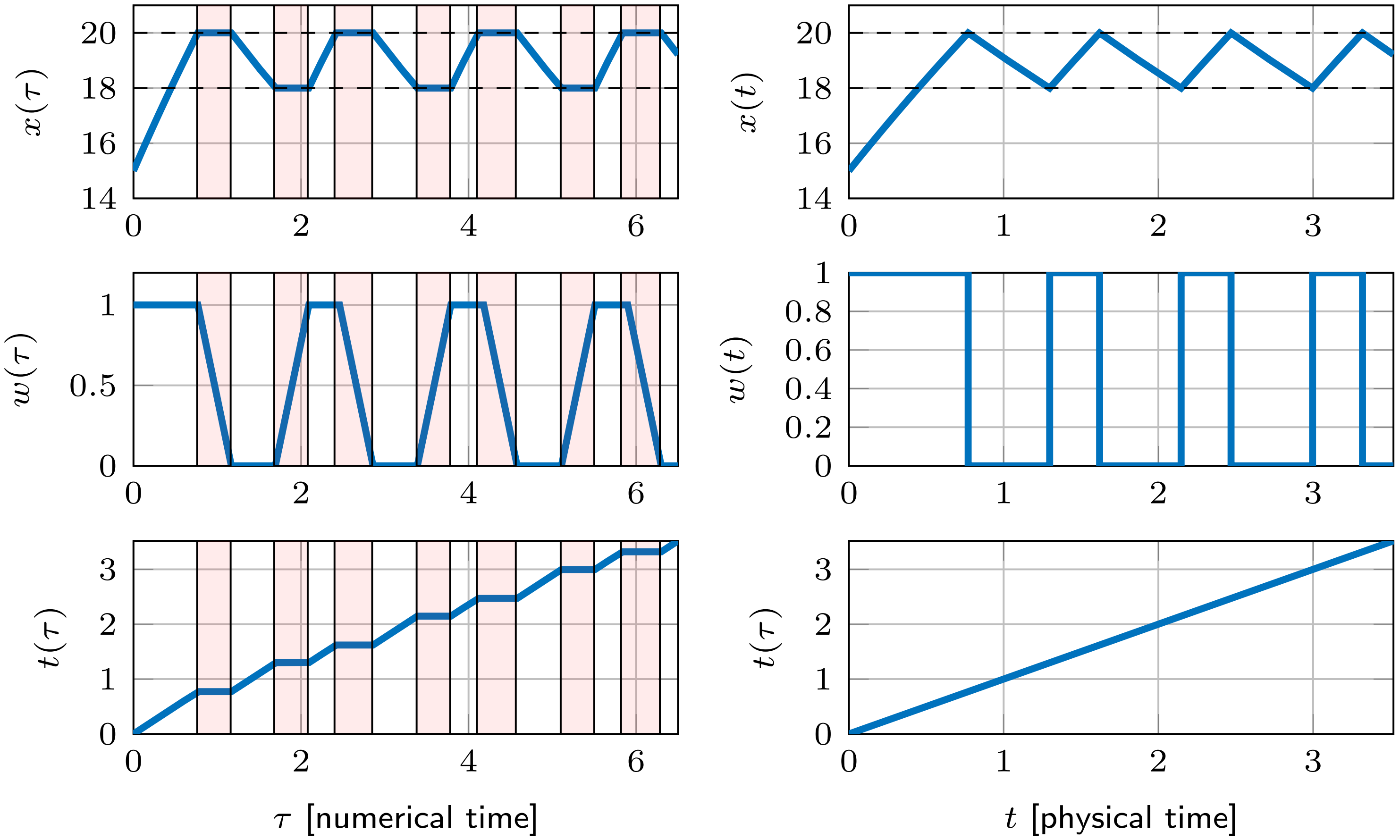


## The State Jump Law

1. if $w(t_{\mathrm{s}}^-) = 0$ and $\psi(x(t_{\mathrm{s}}^-)) = 1$, then $x(t_{\mathrm{s}}^+) = x(t_{\mathrm{s}}^-)$ and $w(t_{\mathrm{s}}^+) = 1$
2. if $w(t_{\mathrm{s}}^-) = 1$ and $\psi(x(t_{\mathrm{s}}^-)) = 0$, then $x(t_{\mathrm{s}}^+) = x(t_{\mathrm{s}}^-)$ and $w(t_{\mathrm{s}}^+) = 0$

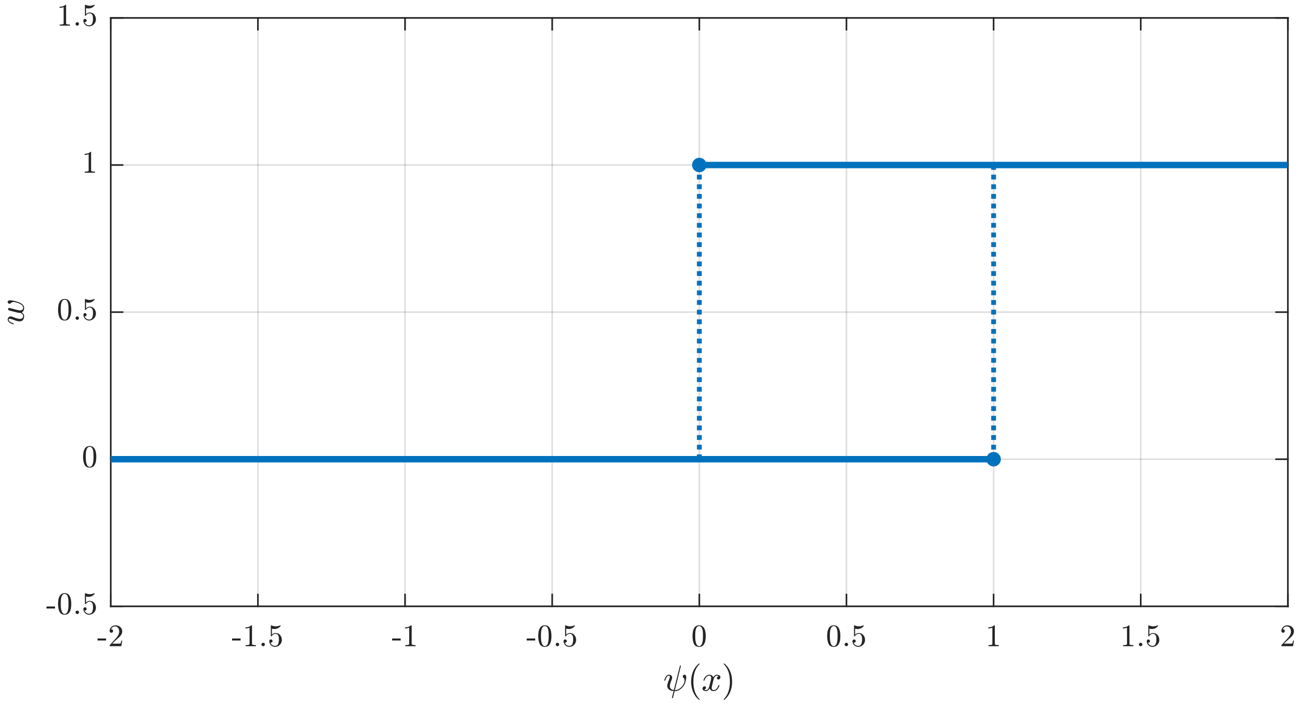
**Remember**: $w(t)$ is now a discontinuous differential state!

# Tutorial example: thermostat and time-freezing



# Time-freezing: the state space

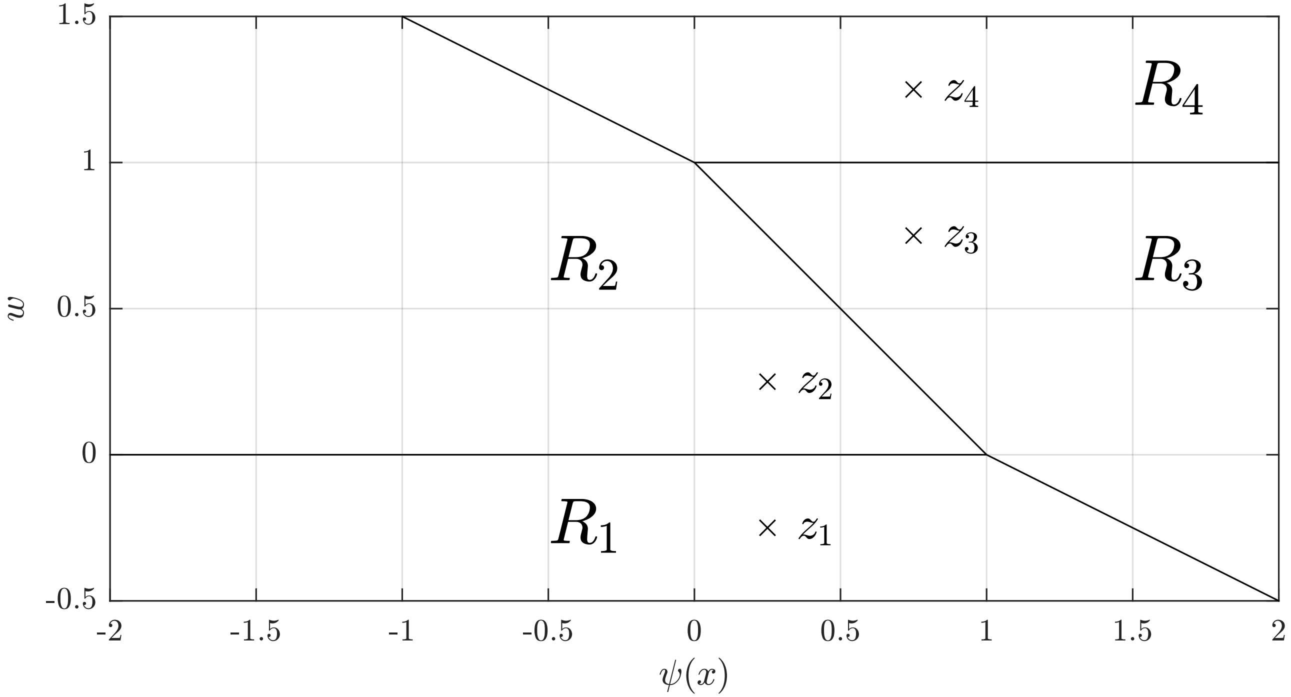
## A look at the $(\psi(x), w)-$plane



- Everything except the blue solid curve is prohibited in the $(\psi, w)-$ space (use $1^{\text{st}}$ principle of time-freezing)
- The evolution happens in a lower-dimensional space $\implies$ *sliding mode* (use $4^{\text{th}}$ principle of time-freezing)

# Time-freezing: partitioning of the space

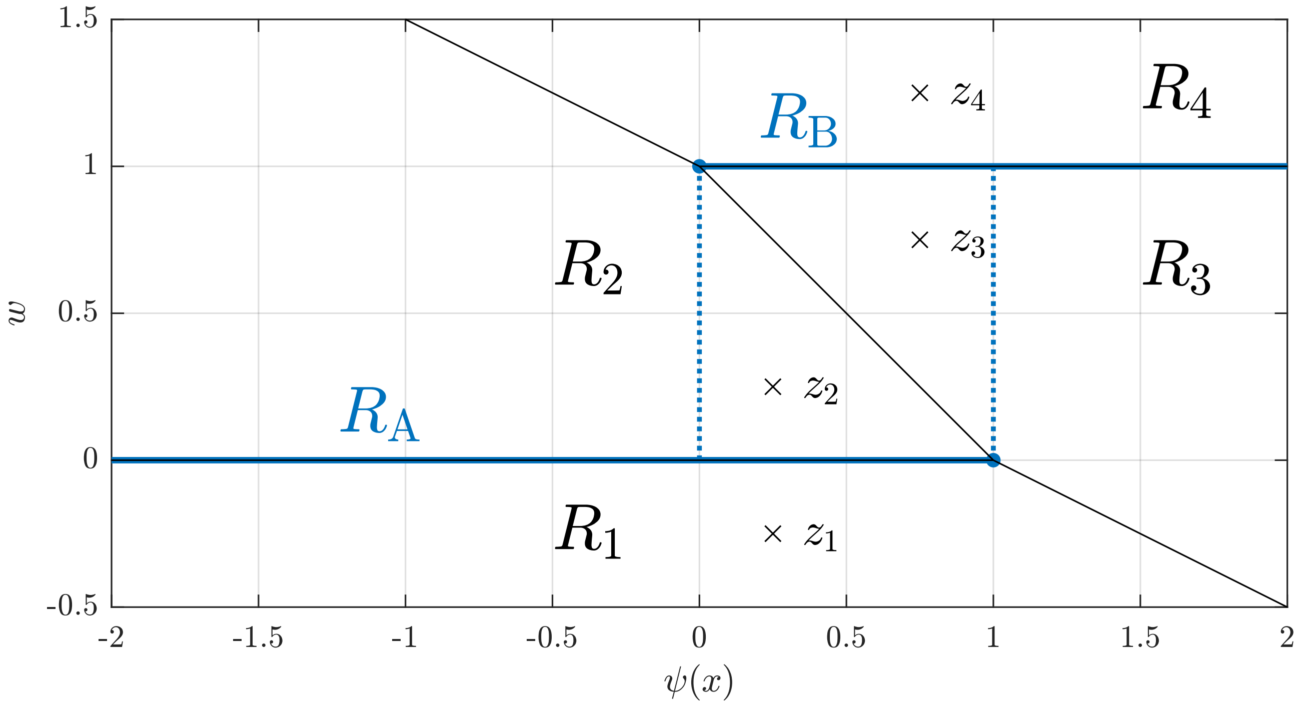
An efficient partition leads to less variables in FESD



- Partition the state space into *Voronoi regions*:
  $R_i = \{z \mid \|z - z_i\|^2 < \|z - z_j\|^2,\ j = 1, \dots, 4, j \neq i\},\ z = (\psi(x), w)$

# Time-freezing: partitioning of the space

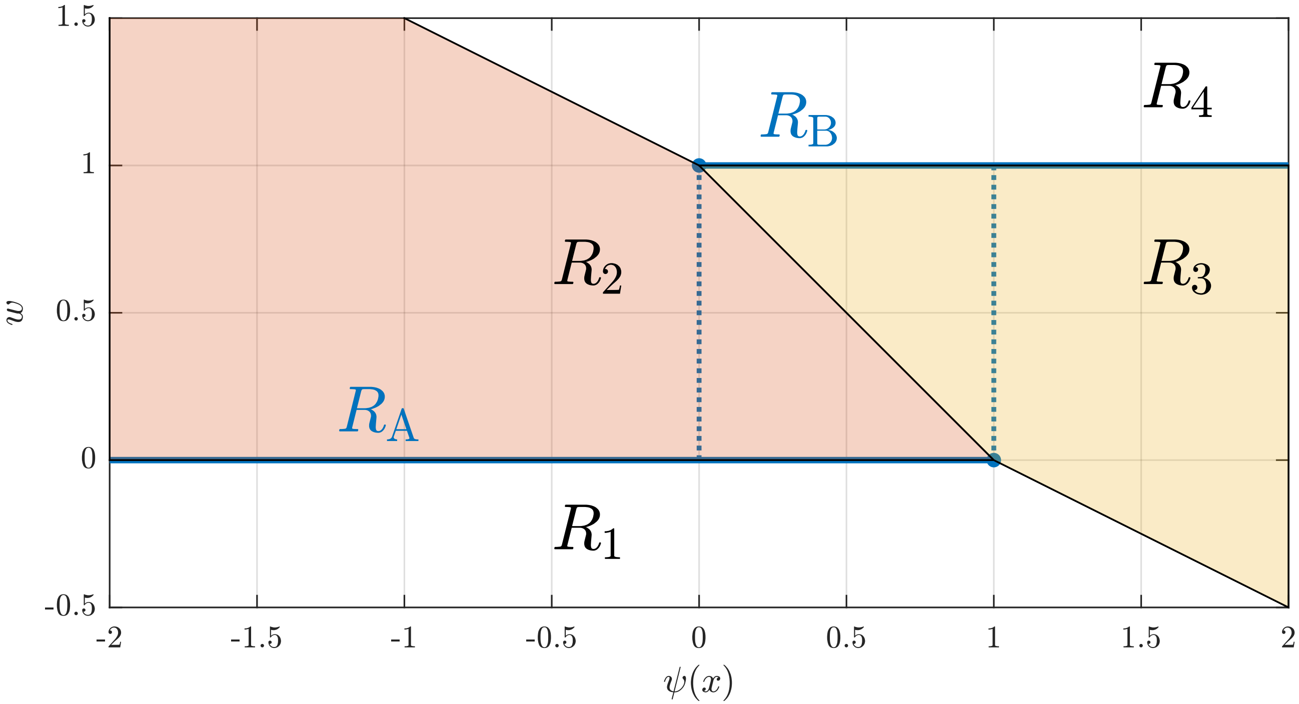
An efficient partition leads to less variables in FESD



- Partition the state space into *Voronoi regions*:
  $R_i = \{z \mid \|z - z_i\|^2 < \|z - z_j\|^2,\ j = 1, \dots, 4, j \neq i\},\ z = (\psi(x), w)$
- Feasible region for initial *hybrid system with hysteresis* on the region boundaries

# Time-freezing: auxiliary dynamics

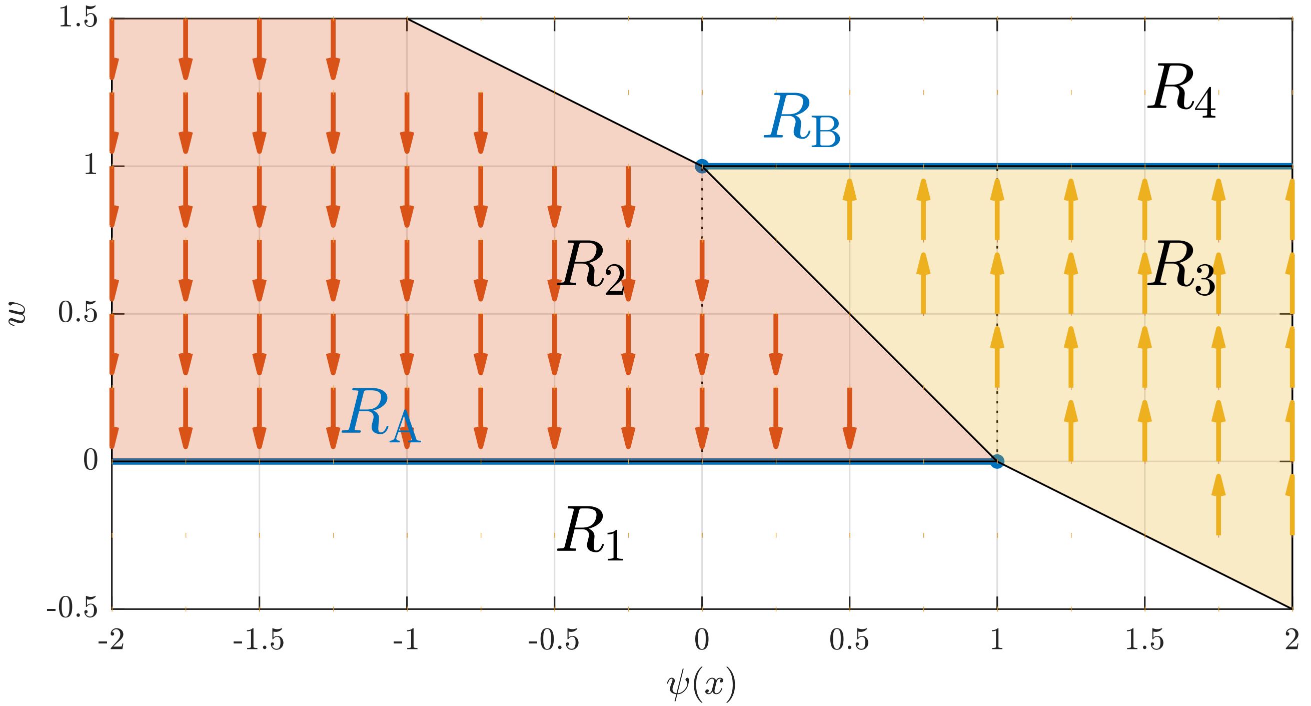
## To mimic state jumps in finite numerical time



- Use regions $R_2$ and $R_3$ to define auxiliary dynamics for the state jumps of $w(\cdot)$

# Time-freezing: auxiliary dynamics

To mimic state jumps in finite numerical time



- Use regions $R_2$ and $R_3$ to define auxiliary dynamics for the state jumps of $w(\cdot)$
- Evolution in $w-$direction happens only for $\psi \in \{0, 1\}$

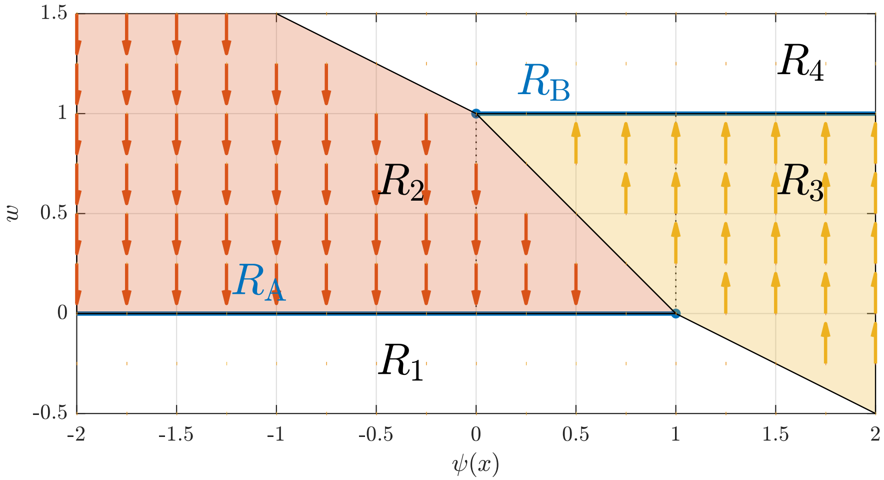
# Time-freezing: auxiliary dynamics

The new state space of the system is $y = (x, w, t) \in \mathbb{R}^{n_x+2}$

## Auxiliary dynamics

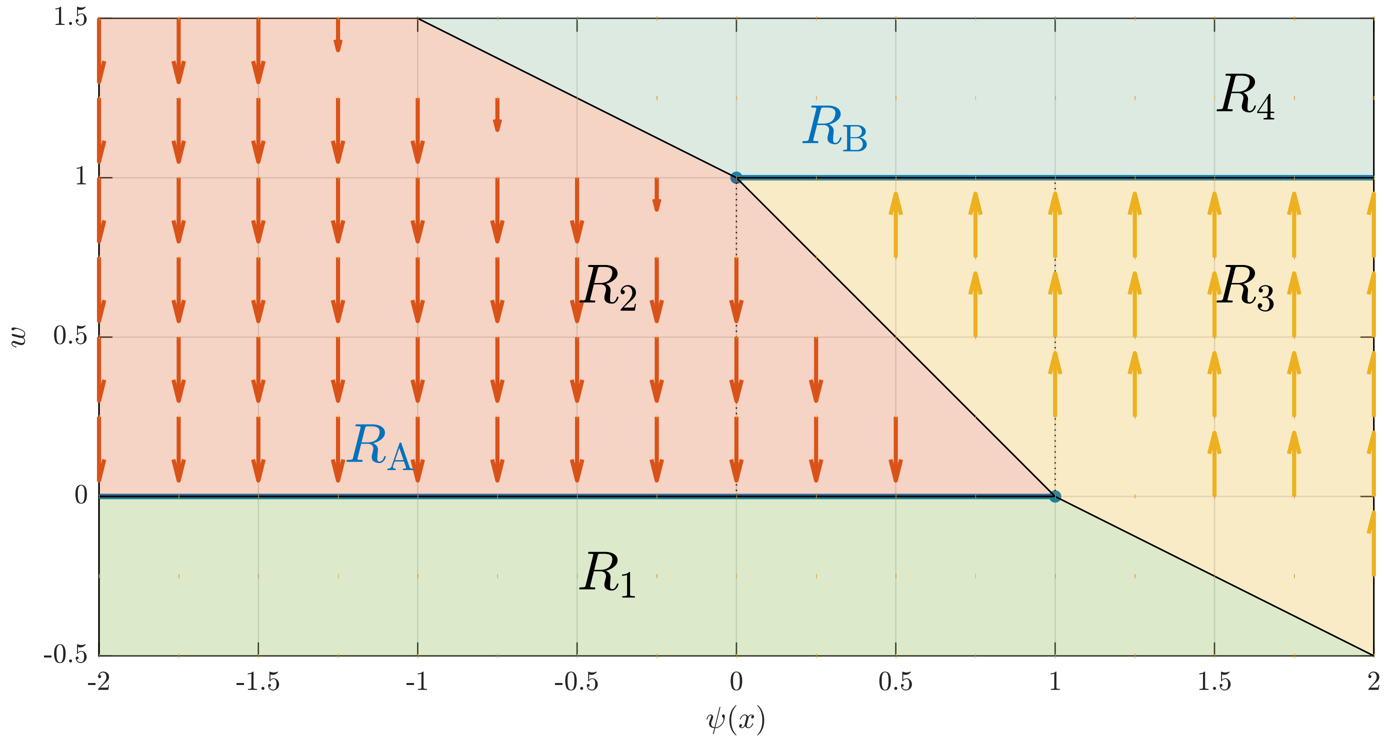
$$f_2(y) = \begin{bmatrix} 0 \\ -a \\ 0 \end{bmatrix}, \quad f_3(y) = \begin{bmatrix} 0 \\ a \\ 0 \end{bmatrix}$$

$a > 0$



# Time-freezing: DAE forming dynamics

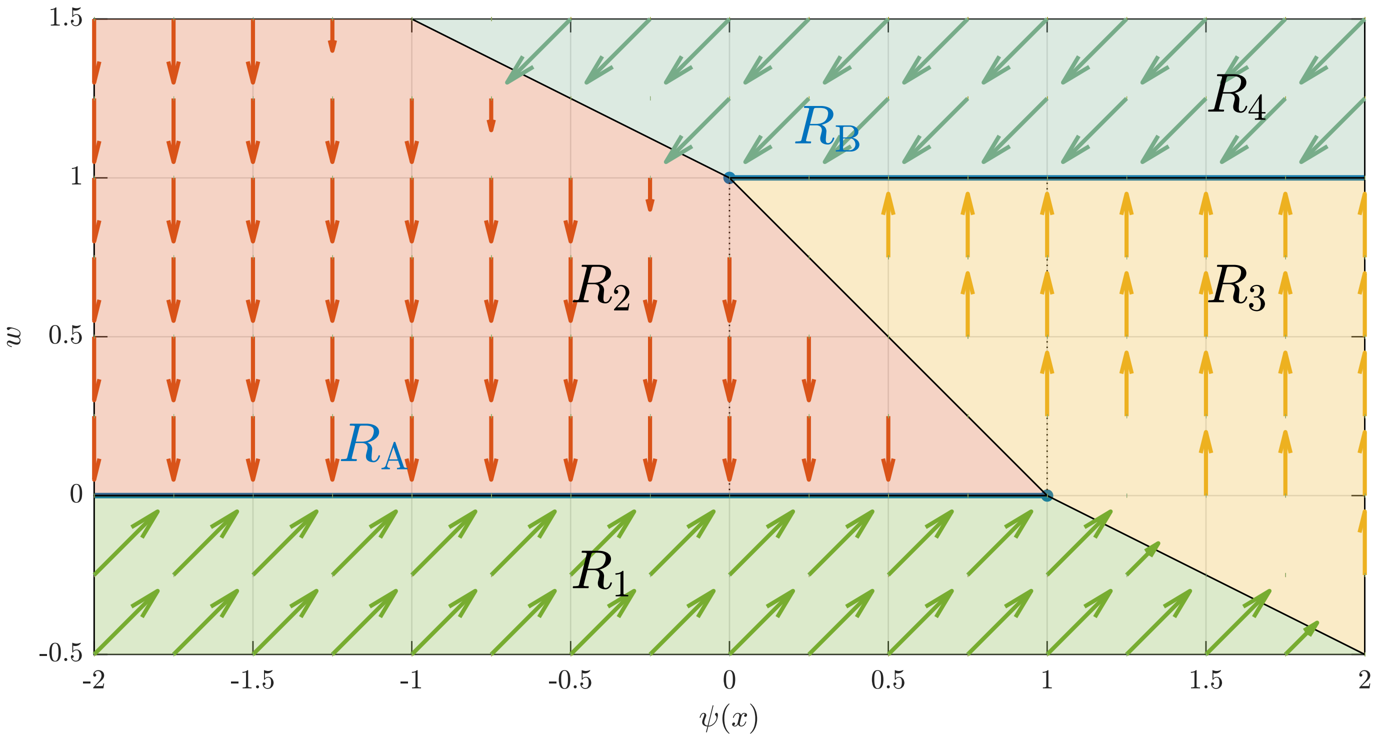
Stop the state jump and construct suitable sliding mode



- Dynamics in $R_1$ and $R_4$ stops evolution of auxiliary ODE - similar to inelastic impacts

# Time-freezing: DAE forming dynamics

Stop the state jump and construct suitable sliding mode



- Dynamics in $R_1$ and $R_4$ stops evolution of auxiliary ODE - similar to inelastic impacts
- Sliding modes on $R_{\mathrm{A}} := \partial R_1 \cap \partial R_2$ and $R_{\mathrm{B}} := \partial R_3 \cap \partial R_4$ match $f_{\mathrm{A}}(y)$ and $f_{\mathrm{B}}(y)$, resp.

# Time-freezing: summary

## DAE-forming dynamics

$$y = (x, w, t)$$

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = f_1(y) = \begin{bmatrix} 2f_{\mathrm{A}}(x) \\ a \\ 2 \end{bmatrix}$$

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = f_4(y) = \begin{bmatrix} 2f_{\mathrm{B}}(x) \\ -a \\ 2 \end{bmatrix}$$

- In total four regions $R_i$ , $i = 1, 2, 3, 4$ and evolution of original system is the **sliding mode**

# Time-freezing: summary

## DAE-forming dynamics

$$y = (x, w, t)$$

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = f_1(y) = \begin{bmatrix} 2 f_{\mathrm{A}}(x) \\ a \\ 2 \end{bmatrix}$$

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = f_4(y) = \begin{bmatrix} 2 f_{\mathrm{B}}(x) \\ -a \\ 2 \end{bmatrix}$$

- In total four regions $R_i$ , $i = 1, 2, 3, 4$ and evolution of original system is the **sliding mode**
- Regions $R_2$ and $R_3$ equipped with aux. dynamics to mimic state jump

# Time-freezing: summary

## DAE-forming dynamics

$$y = (x, w, t)$$

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = f_1(y) = \begin{bmatrix} 2f_{\mathrm{A}}(x) \\ a \\ 2 \end{bmatrix}$$

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = f_4(y) = \begin{bmatrix} 2f_{\mathrm{B}}(x) \\ -a \\ 2 \end{bmatrix}$$

- In total four regions $R_i$ , $i = 1, 2, 3, 4$ and evolution of original system is the **sliding mode**
- Regions $R_2$ and $R_3$ equipped with aux. dynamics to mimic state jump
- Regions $R_1$ and $R_4$ equipped with DAE-forming dynamics to recover original dynamics in sliding mode

# Time-freezing: summary

## DAE-forming dynamics

$$y = (x, w, t)$$

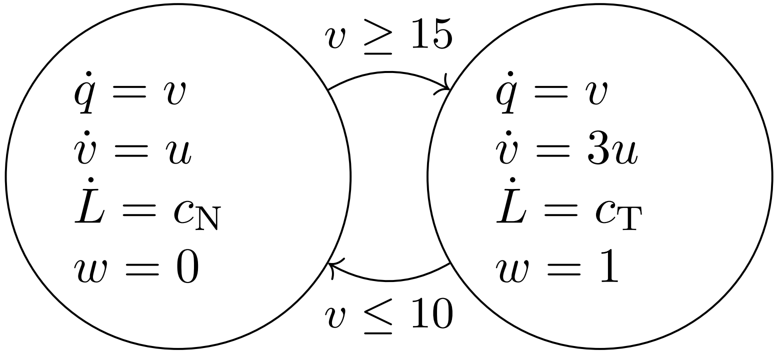
$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = f_1(y) = \begin{bmatrix} 2f_{\mathrm{A}}(x) \\ a \\ 2 \end{bmatrix}$$

$$\frac{\mathrm{d}y}{\mathrm{d}\tau} = f_4(y) = \begin{bmatrix} 2f_{\mathrm{B}}(x) \\ -a \\ 2 \end{bmatrix}$$

- In total four regions $R_i$ , $i = 1, 2, 3, 4$ and evolution of original system is the **sliding mode**
- Regions $R_2$ and $R_3$ equipped with aux. dynamics to mimic state jump
- Regions $R_1$ and $R_4$ equipped with DAE-forming dynamics to recover original dynamics in sliding mode
- E.g., $w' = 0 \implies \theta_1 f_1(y) + \theta_2 f_2(y) = f_{\mathrm{A}}(y)$ (sliding mode)
- Conclusion: we have a PSS and can treat it with FESD

# Time optimal control of a car with a turbo accelerator

## Example from [Avraam, 2000] solved with NOSNOC



$$\dot{q} = v \qquad \dot{v} = u \qquad \dot{L} = c_{\mathrm{N}} \qquad w = 0$$

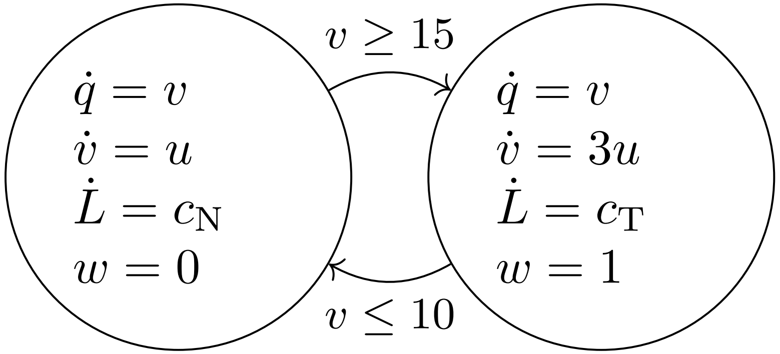
$$v \geq 15$$

$$v \leq 10$$

$$\dot{q} = v \qquad \dot{v} = 3u \qquad \dot{L} = c_{\mathrm{T}} \qquad w = 1$$

# Time optimal control of a car with a turbo accelerator

Example from [Avraam, 2000] solved with NOSNOC



$\dot{q} = v$
$\dot{v} = u$
$\dot{L} = c_{\mathrm{N}}$
$w = 0$

$v \geq 15$

$v \leq 10$

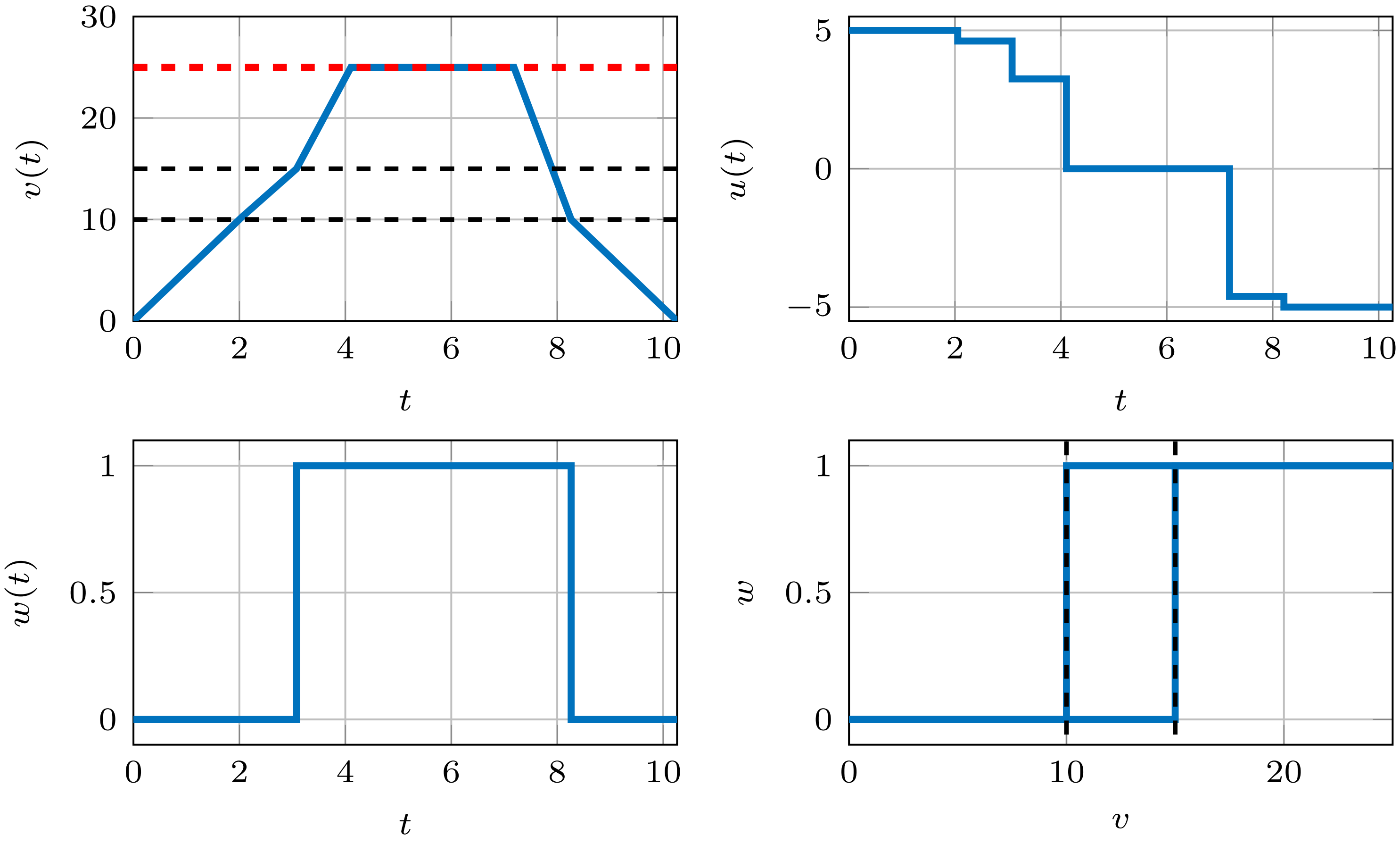
$\dot{q} = v$
$\dot{v} = 3u$
$\dot{L} = c_{\mathrm{T}}$
$w = 1$

## Time optimal control problem

$$
\begin{aligned}
\min_{y(\cdot),u(\cdot),s(\cdot)} \quad & t(\tau_{\mathrm{f}}) + L(\tau_{\mathrm{f}}) \\
\text{s.t.} \quad & y(0) = (z_0, 0) \\
& y'(\tau) \in s(\tau) F_{\mathrm{TF}}(y(\tau), u(\tau)) \\
& -\bar{u} \leq u(\tau) \leq \bar{u} \\
& \bar{s}^{-1} \leq s(\tau) \leq \bar{s} \\
& -\bar{v} \leq v(\tau) \leq \bar{v} \ \tau \in [0, \tau_{\mathrm{f}}] \\
& (q(\tau_{\mathrm{f}}), v(\tau_{\mathrm{f}})) = (q_{\mathrm{f}}, v_{\mathrm{f}})
\end{aligned}
$$

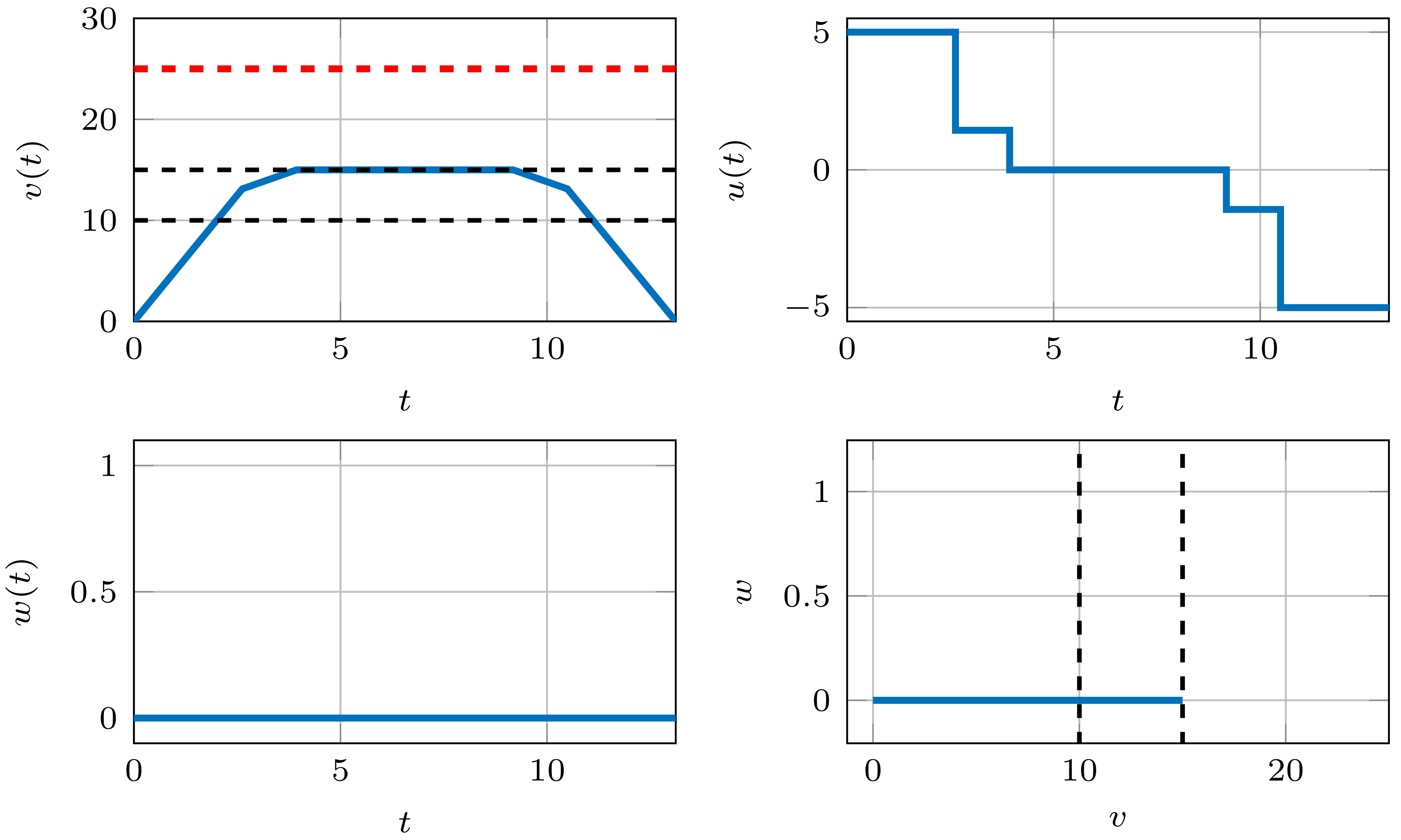
# Scenario 1: turbo and nominal cost the same

$c_\mathrm{N} = c_\mathrm{T}$



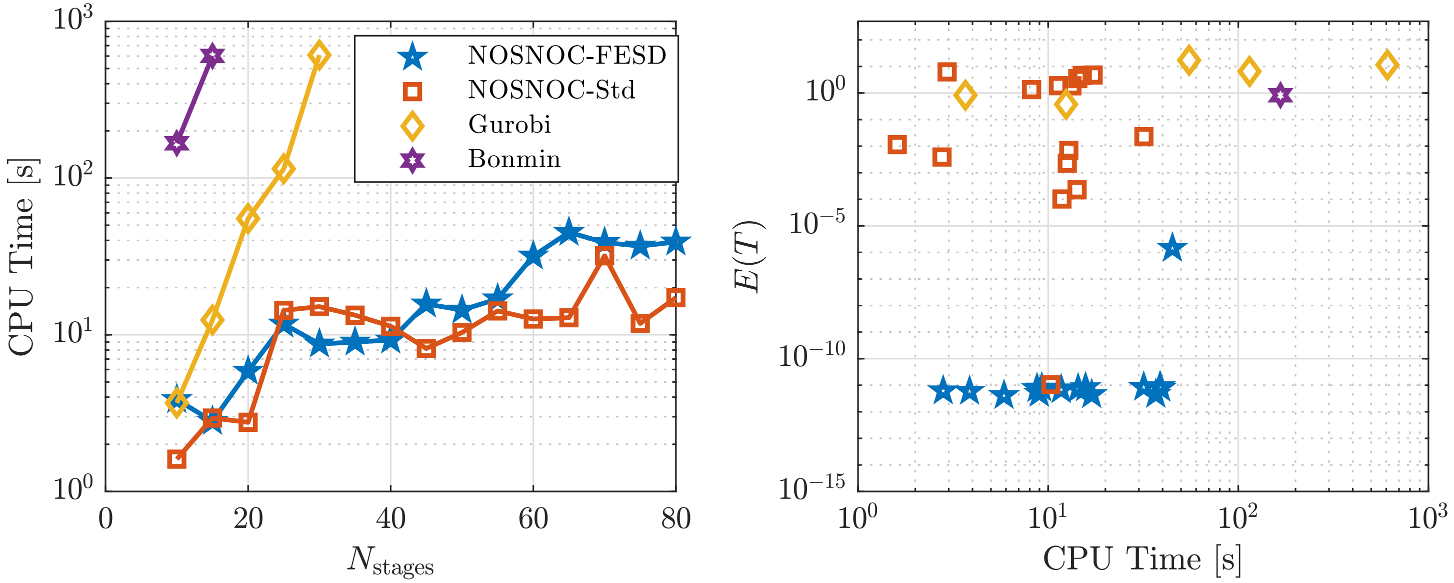
# Scenario 2: Turbo is Expensive

$c_{\mathrm{N}} < c_{\mathrm{T}}$



# NOSNOC vs MILP/MINLP formulations

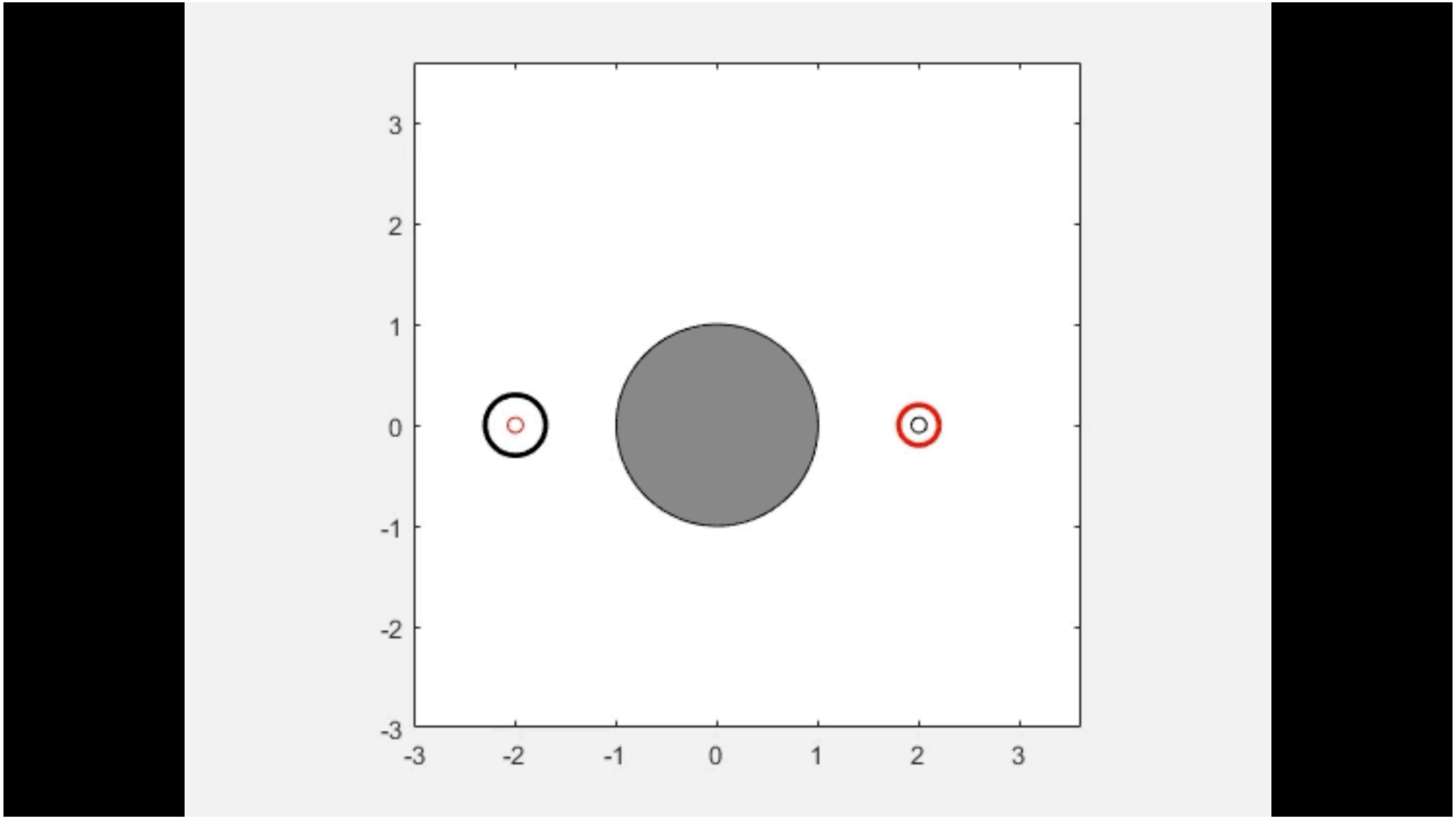
## Benchmark on time-optimal control problem of a car with turbo



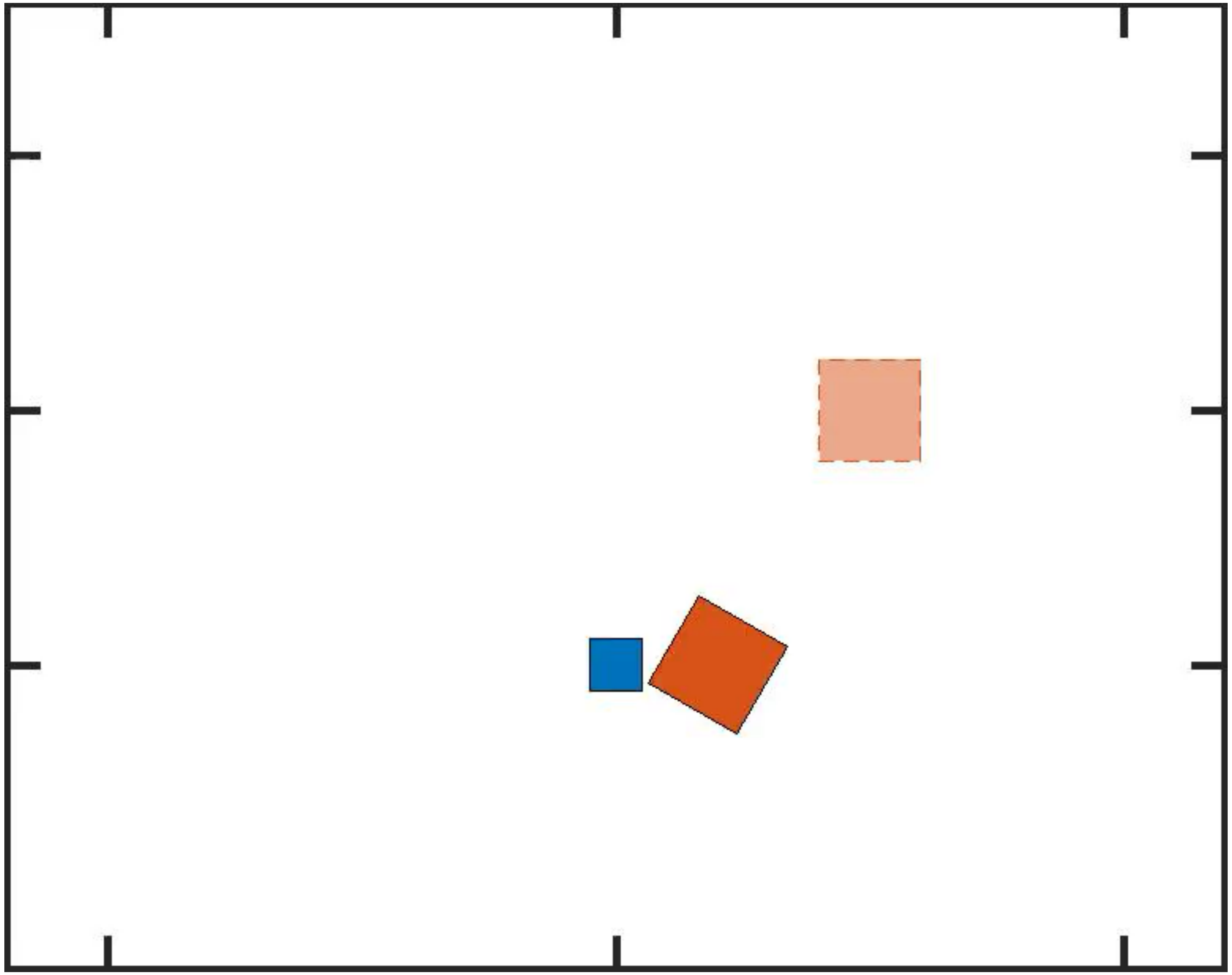
- compare CPU time as function of number of control intervals $N$ (left) and solution accuracy (right)
- MILP (Gurobi): solve problem with fixed $T$ until indefeasibly happens with grid search in $T$
- MILP/MINLP and NOSNOC-Std no switch detection $=$ low accuracy

# APPENDIX 3 - More NOSNOC Examples and Time-Freezing

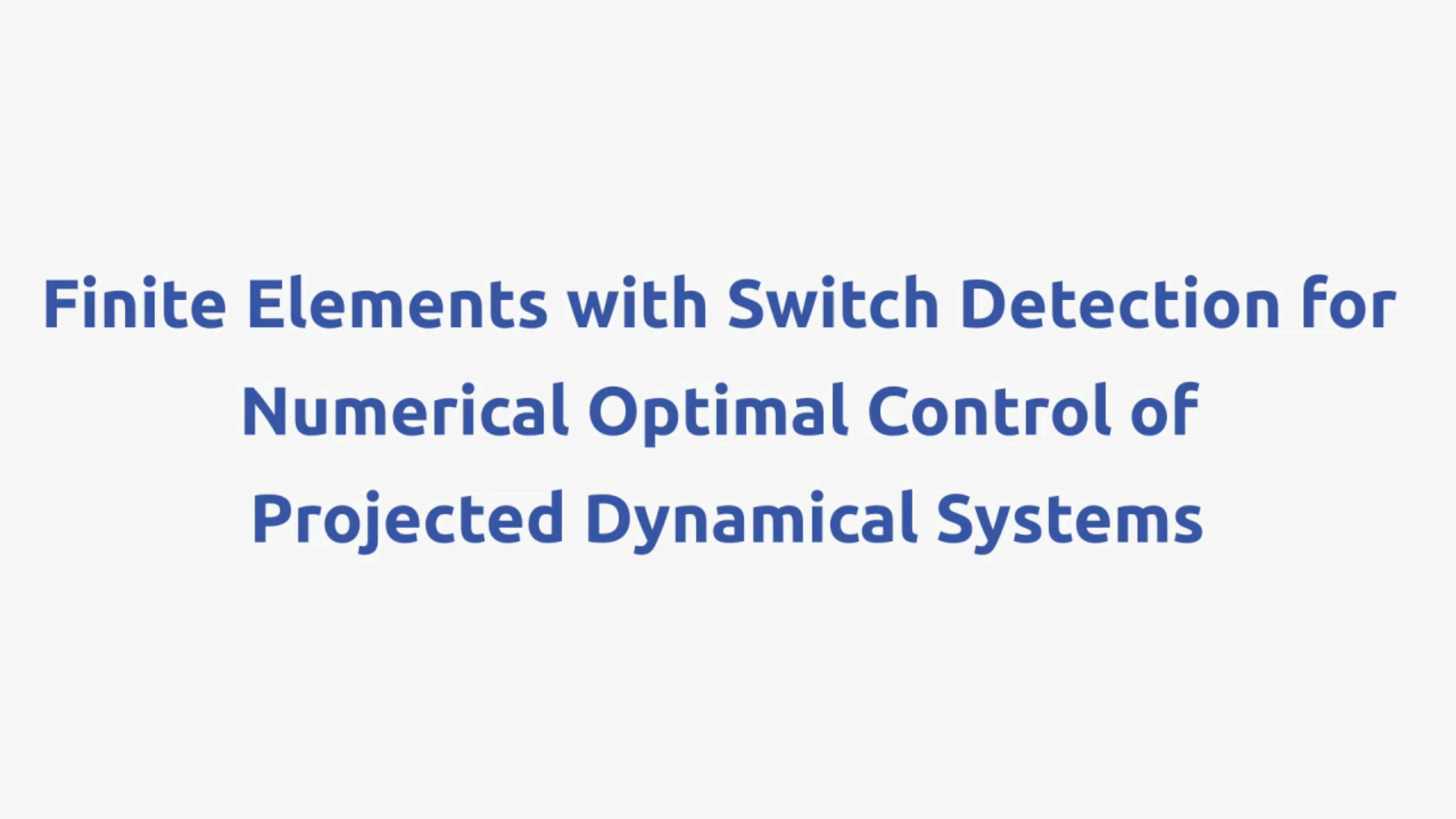
# NOSNOC examples



# NOSNOC examples



# NOSNOC examples

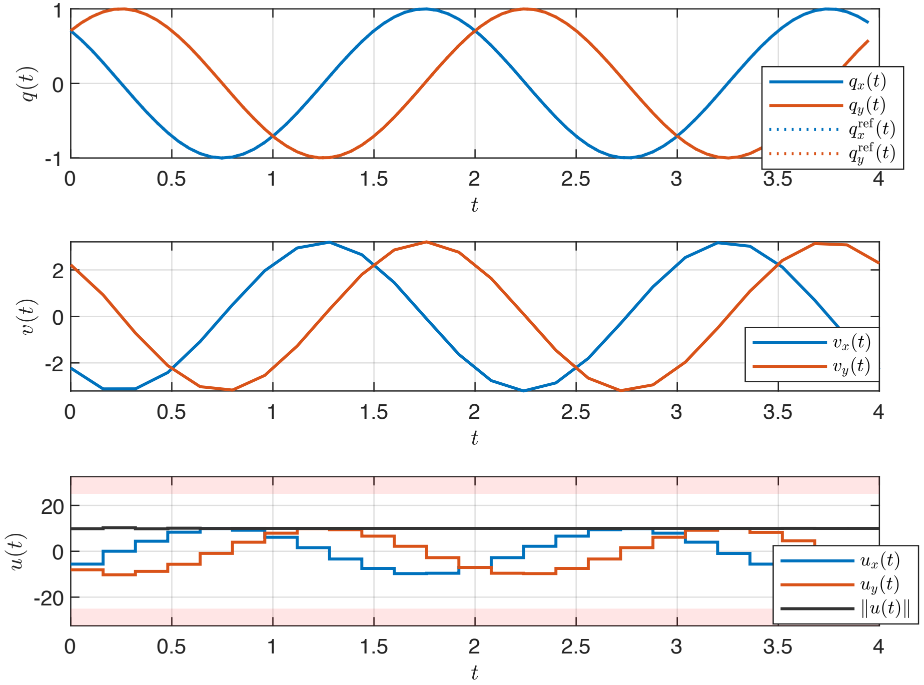


Finite Elements with Switch Detection for Numerical Optimal Control of Projected Dynamical Systems

# Results with slowly moving reference

For $\omega = \pi$, tracking is easy: no jumps occur in optimal solution.

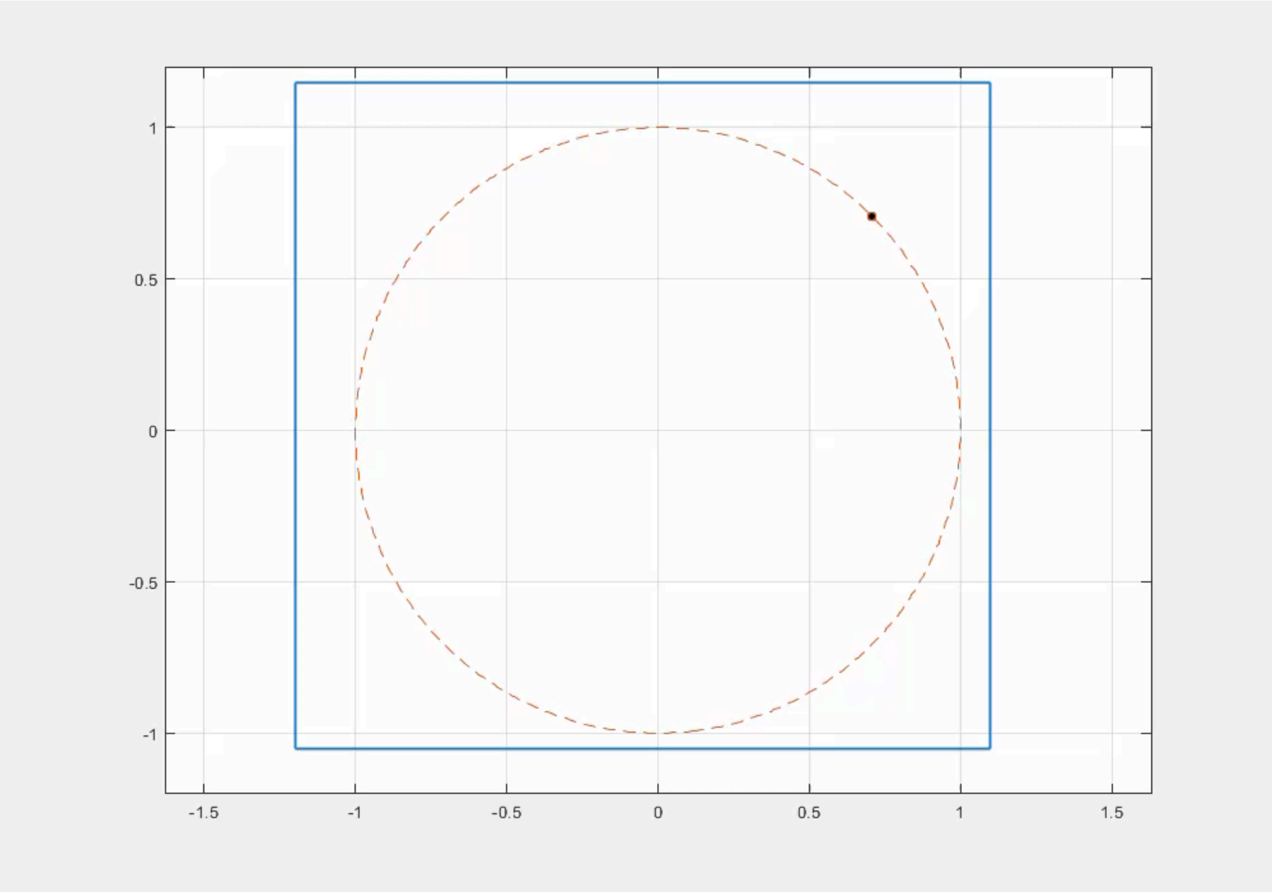
- Regard time horizon of two periods
- $N = 25$ equidistant control intervals
- use FESD with $N_{\mathrm{FE}} = 3$ finite elements with Radau 3 on each control interval
- each FESD interval has one constant control $u$ and one speed of time $s$
- MPCC solved via $\ell_\infty$ penalty reformulation and homotopy
- For homotopy convergence: in total 4 NLPs solved with `IPOPT` via `CasADi`



States and controls in physical time.

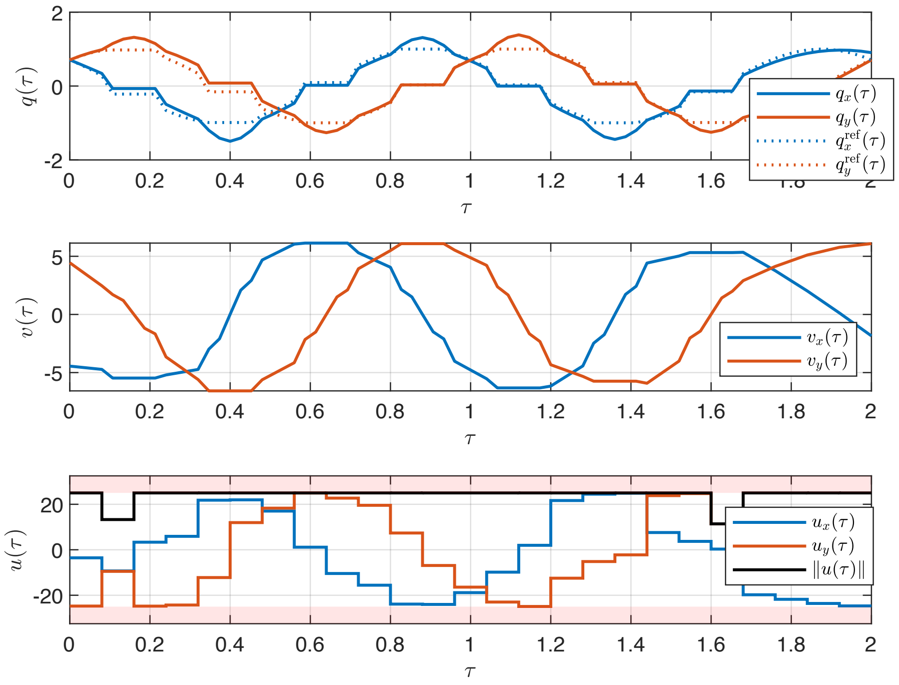
# Results with slowly moving reference - movie

For $\omega = \pi$, tracking is easy: no jumps occur in optimal solution.

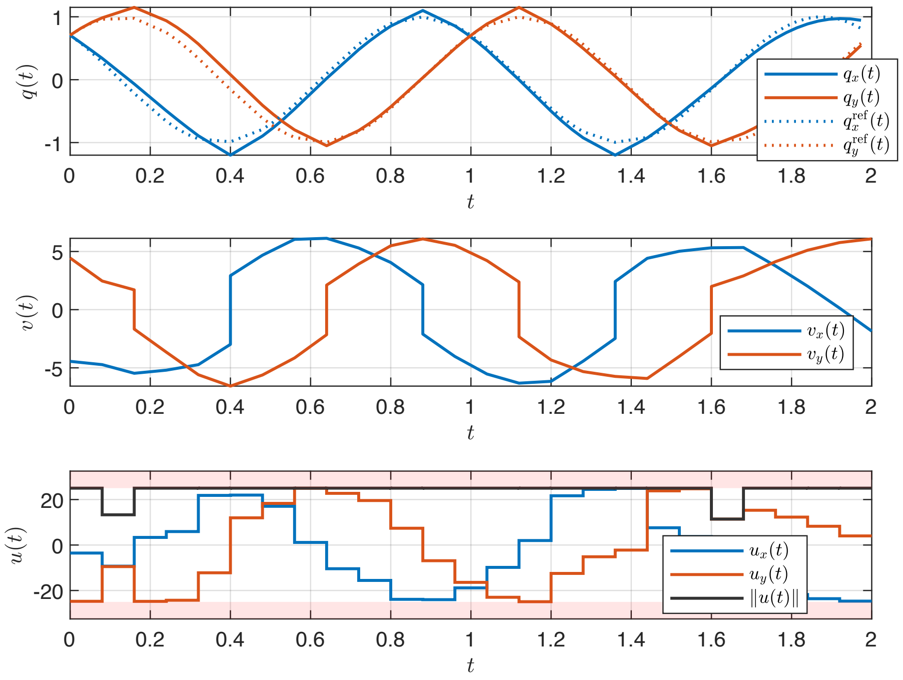


# Results with fast reference

For $\omega = 2\pi$, tracking is only possible if ball bounces against walls.



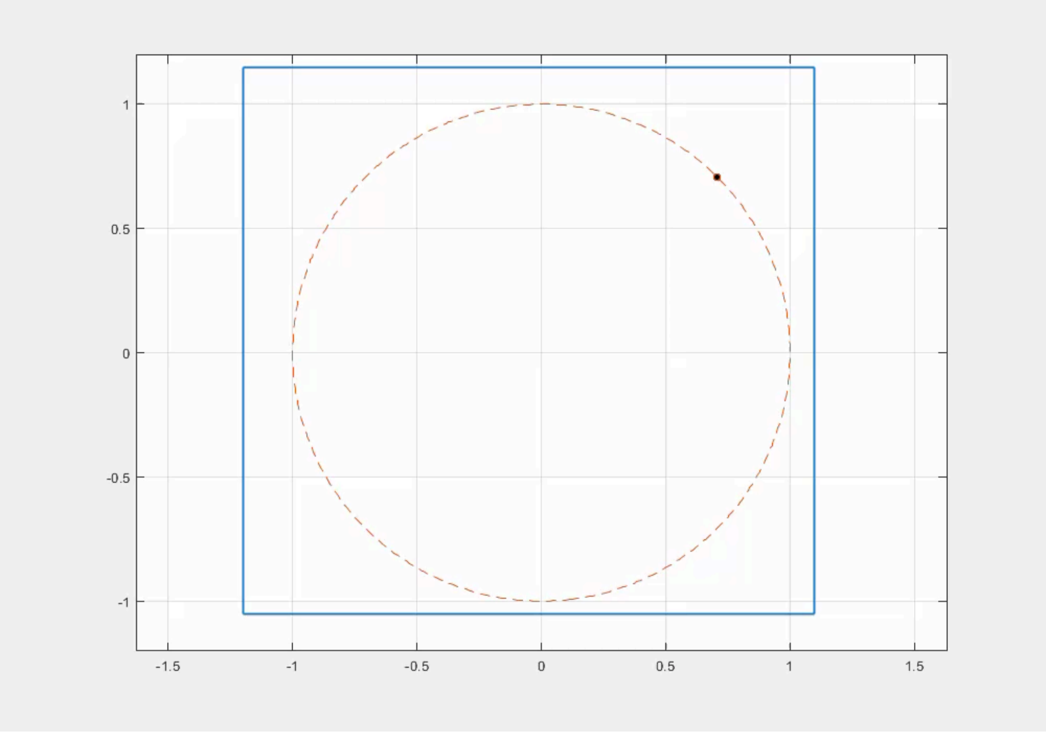
States and controls in numerical time.

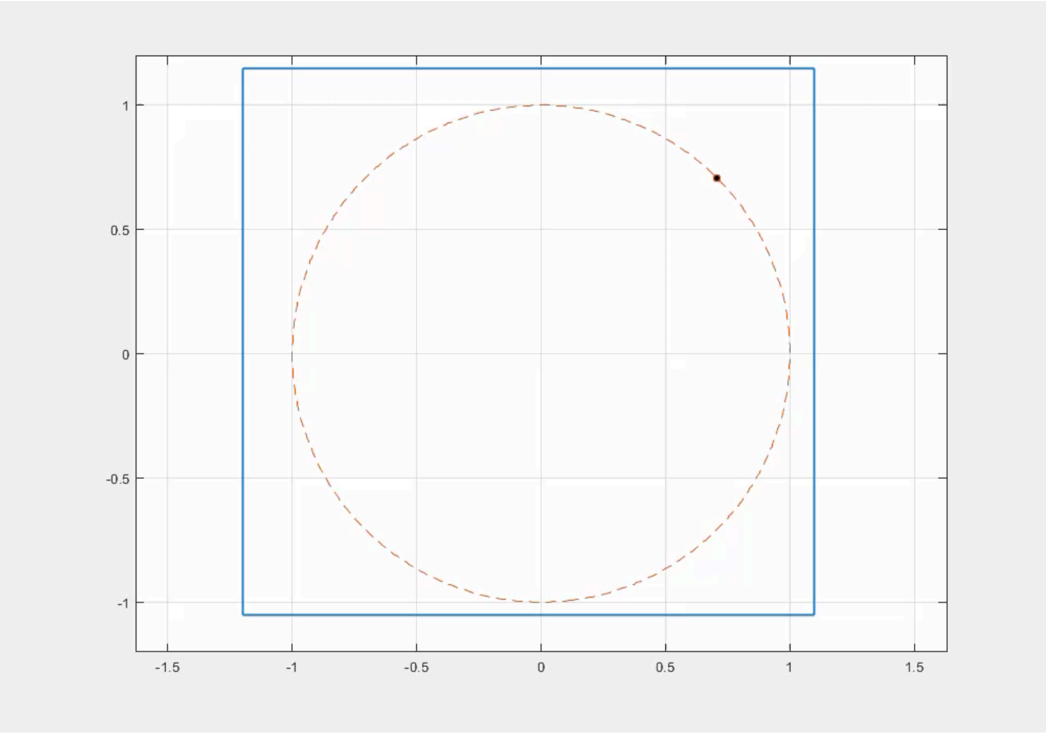


States and controls in physical time.

# Results with fast reference - movie

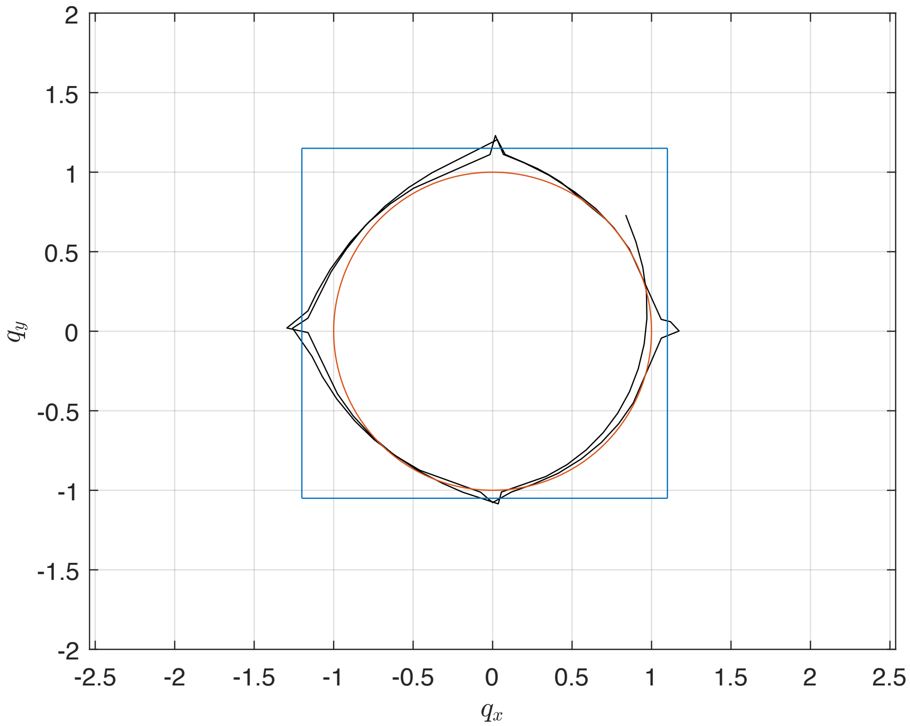
For $\omega = 2\pi$, tracking is only possible if ball bounces against walls.



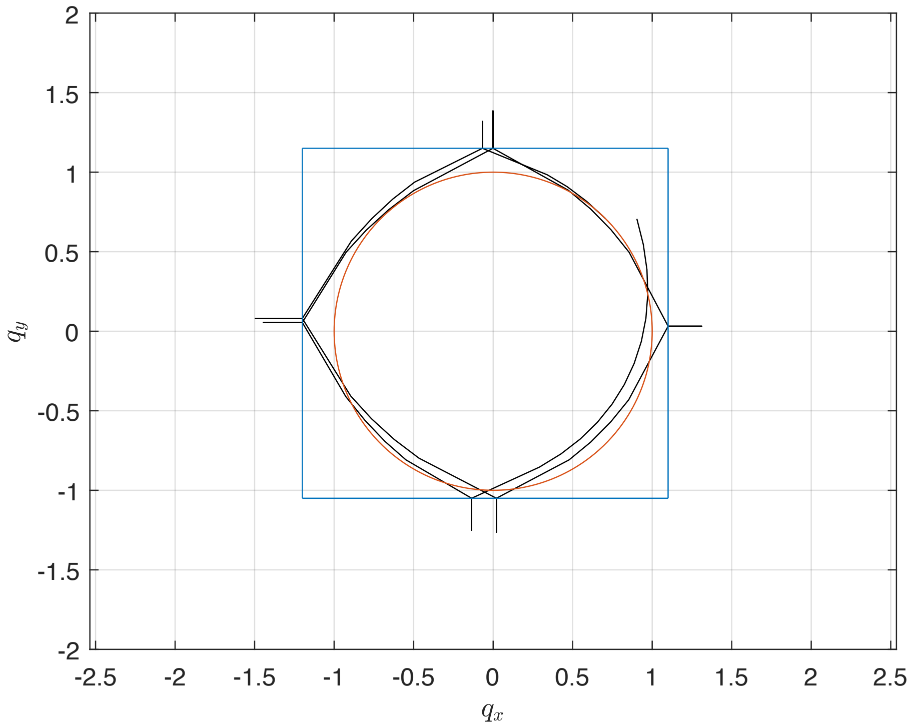


# Homotopy: first iteration vs converged solution

## Geometric trajectory



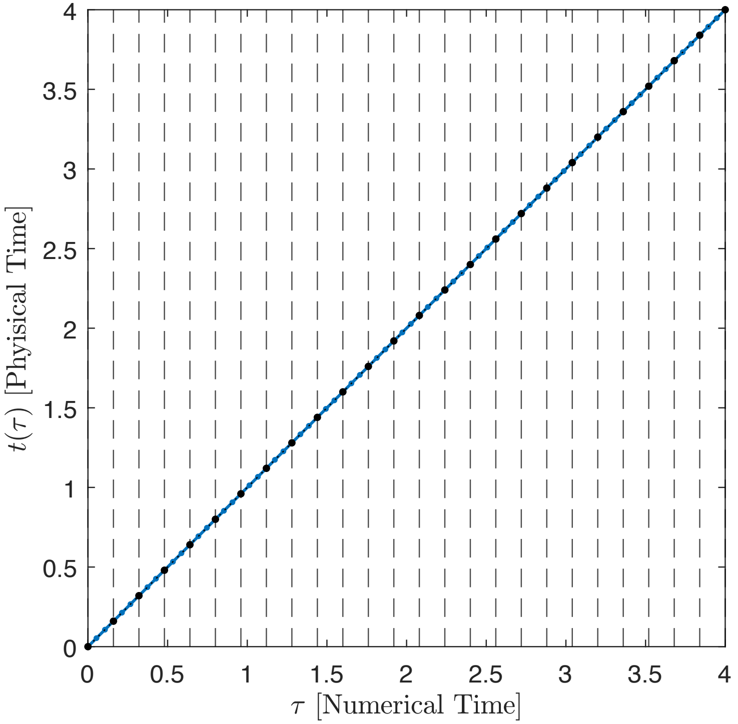
After the first homotopy iteration



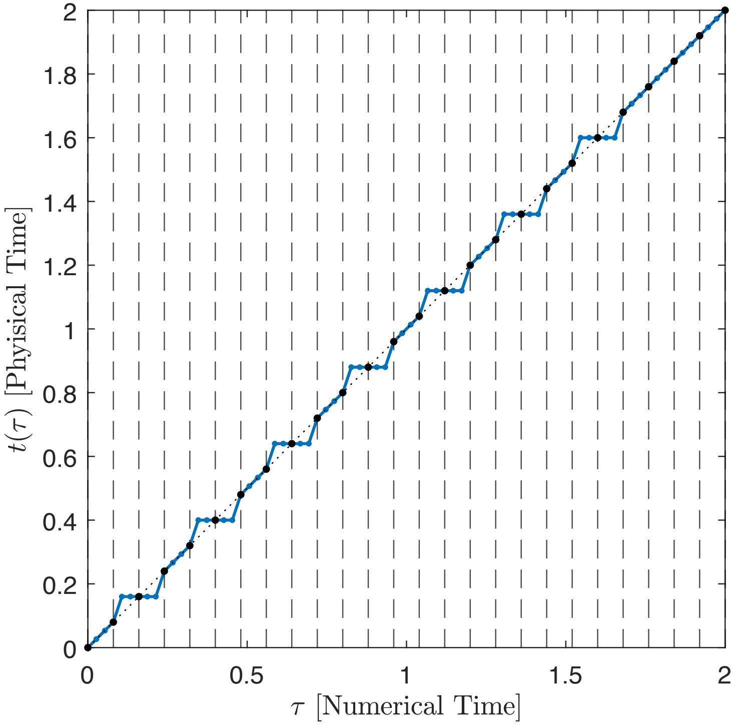
The solution trajectory after convergence

# Physical vs. Numerical Time

for $\omega = \pi$



for $\omega = 2\pi$


$t(\tau)$ [Physical Time]
$\tau$ [Numerical Time]