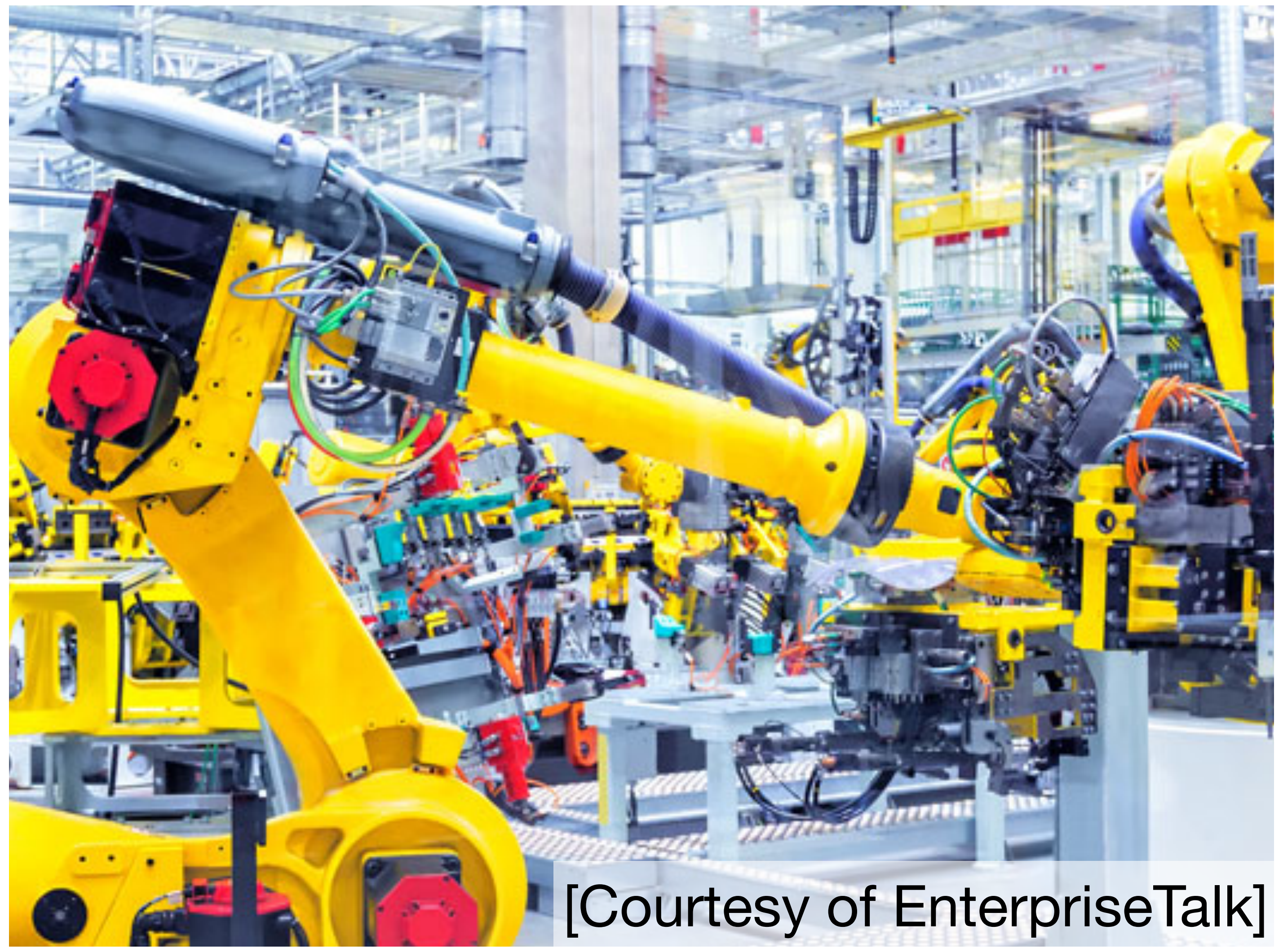
Real-time Decision-Making via Data-Driven Optimization

PRINCETON UNIVERSITY ORFE

Bartolomeo Stellato — Politecnico di Milano, June 16 2022

New Capabilities for Autonomous Systems

Automation



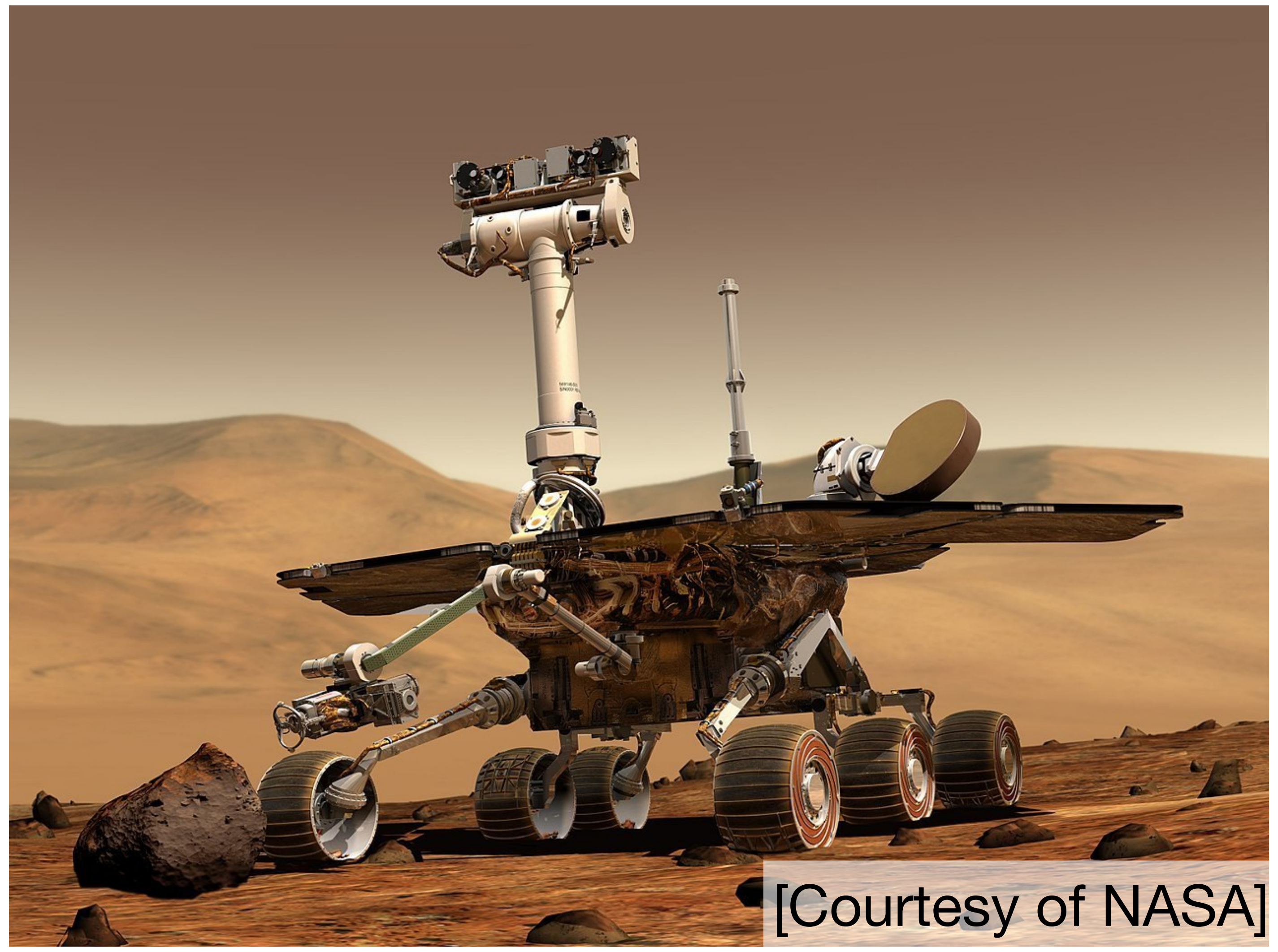
[Courtesy of EnterpriseTalk]

system limited to
pre-programmed behavior

faster
computing

accurate
sensing

Autonomy



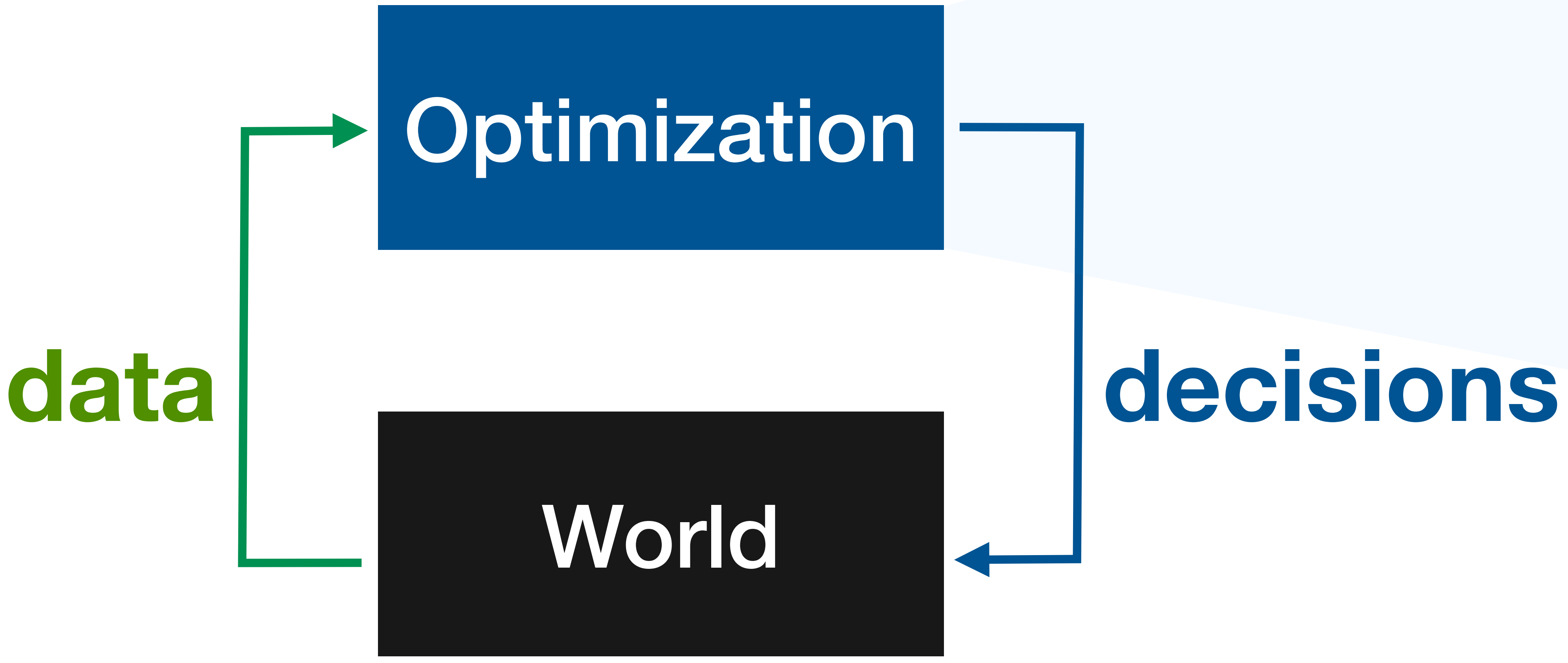
[Courtesy of NASA]

system reacts to
new not pre-programmed situations

How can an autonomous system **quickly** and **safely** react to unexpected conditions?

Optimization for real-time decision-making

Sequential decisions over time



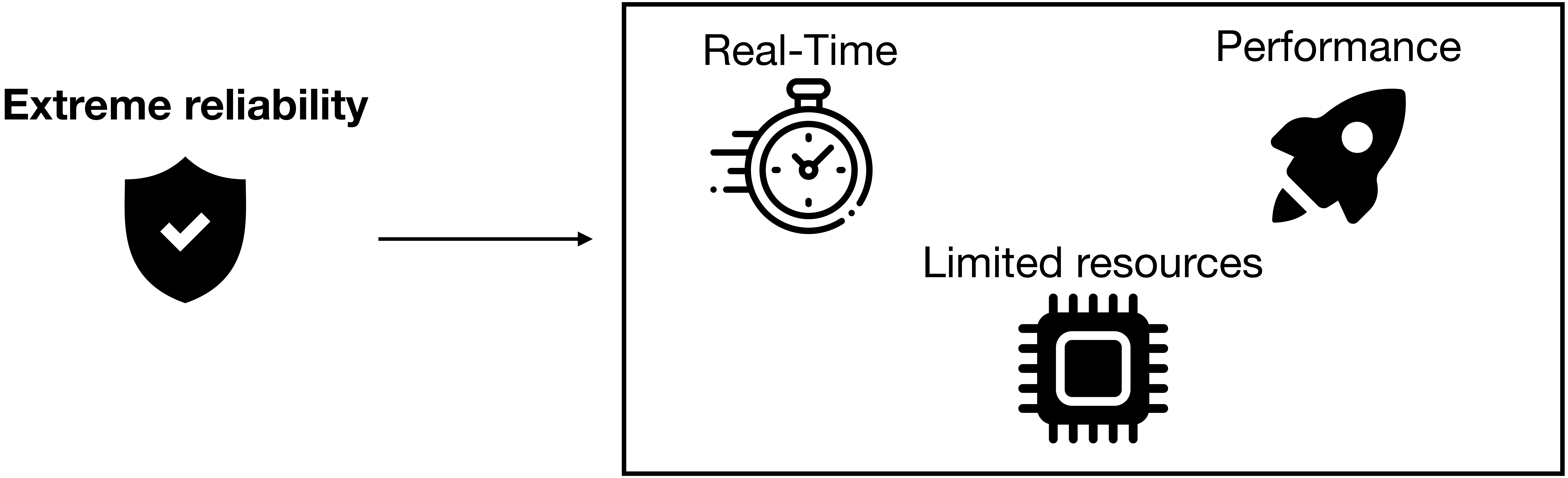
decisions

$$\begin{array}{ll}\text{minimize} & f(x, \theta)\\ \text{subject to} & g(x, \theta) \leq 0\end{array}$$

parameters

objective f: consumption, costs, distance
constraints g: dynamics, safety, budget
parameters θ: state, obstacles, target location

Challenges in real-time optimization

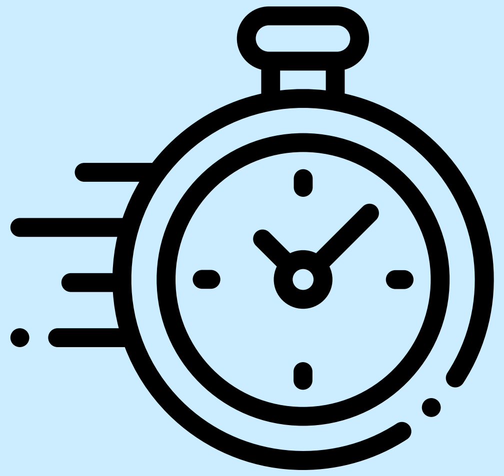


Today’s talk

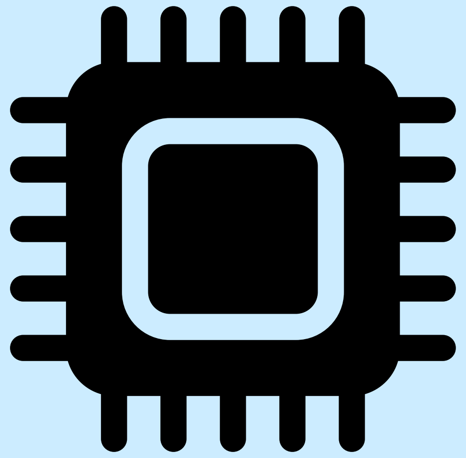
Real-time Decision-Making via Data-Driven Optimization

OSQP
Solver

Real-Time

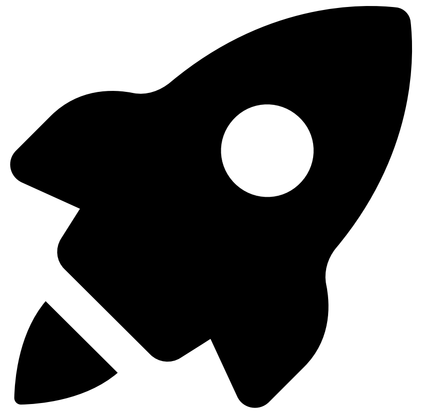


Limited resources



Learning
Convex Optimization
Control Policies

Performance




123.74
140
135
130
125
Tues.
142.78
Chg.
1.64
1.98
Day

First-order methods

Wide popularity

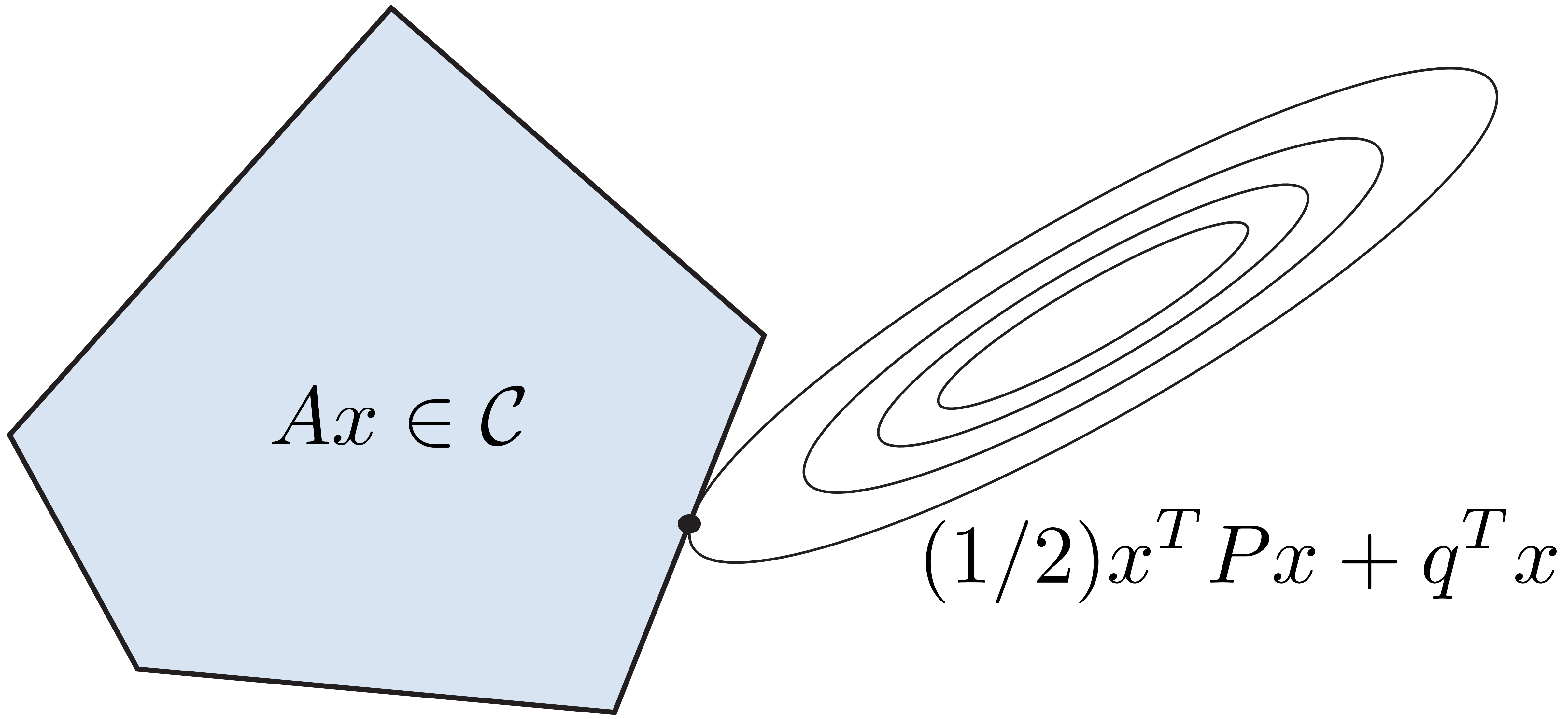
Pros	Cons		OSQP
Warm-starting	Low quality solutions		High-quality solutions
Large-scale problems	Can’t detect infeasibility	→	Detects infeasibility
Embeddable	Problem data dependent		Robust

The problem

minimize $\quad (1/2)x^T Px + q^T x$

subject to $\quad Ax \in \mathcal{C}$

Quadratic program: $\mathcal{C} = [l, u]$



ADMM

Alternating Direction Method of Multipliers

Splitting

minimize $\quad f(x) + g(x) \quad \longrightarrow$

$$\begin{array}{ll} \text{minimize} & f(\tilde{x}) + g(x) \\ \text{subject to} & \tilde{x} = x \end{array}$$

Iterations

$$\tilde{x}^{k+1} \leftarrow \underset{\tilde{x}}{\text{argmin}} \left(f(\tilde{x}) + \rho/2 \left\| \tilde{x} - (x^k - y^k/\rho) \right\|^2 \right)$$

$$x^{k+1} \leftarrow \underset{x}{\text{argmin}} \left(g(x) + \rho/2 \left\| x - (\tilde{x}^{k+1} + y^k/\rho) \right\|^2 \right)$$

$$y^{k+1} \leftarrow y^k + \rho \left(\tilde{x}^{k+1} - x^{k+1} \right)$$

How do we split the QP?

$$\begin{array}{lll} \text{minimize} & (1/2)x^T P x + q^T x & \\ \text{subject to} & Ax = z & f \\ & z \in \mathcal{C} & g \end{array}$$

Splitting formulation

$$\begin{array}{ll} \text{minimize} & \underbrace{(1/2)\tilde{x}^T P \tilde{x} + q^T \tilde{x} + \mathcal{I}_{Ax=z}(\tilde{x}, \tilde{z})}^{f} + \underbrace{\mathcal{I}_{\mathcal{C}}(z)}^{g} \\ \text{subject to} & \tilde{x} = x \\ & \tilde{z} = z \end{array}$$

Diagonal
step sizes

σ

ρ

Complete algorithm

Problem

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & l \leq Ax \leq u \end{array}$$

Algorithm

Linear system solve

$$x^{k+1} \leftarrow \text{Solve} \quad (P + \sigma + \rho A^T A)x = \sigma x^k - q + A^T(\rho z^k - y^k)$$

Easy operations

$$z^{k+1} \leftarrow \Pi(Ax^{k+1} + \rho^{-1} y^k)$$

$$y^{k+1} \leftarrow y^k + \rho(Ax^{k+1} - z^{k+1})$$

always solvable!

Code generation with OSQP

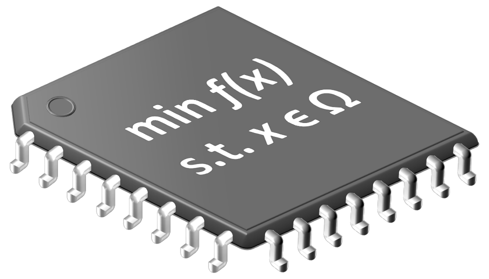
```
# Create OSQP object
m = osqp.OSQP()

# Initialize solver
m.setup(P, q, A, l, u,
        settings)

# Generate C code
m.codegen('folder_name')
```

→

Optimized C code

→

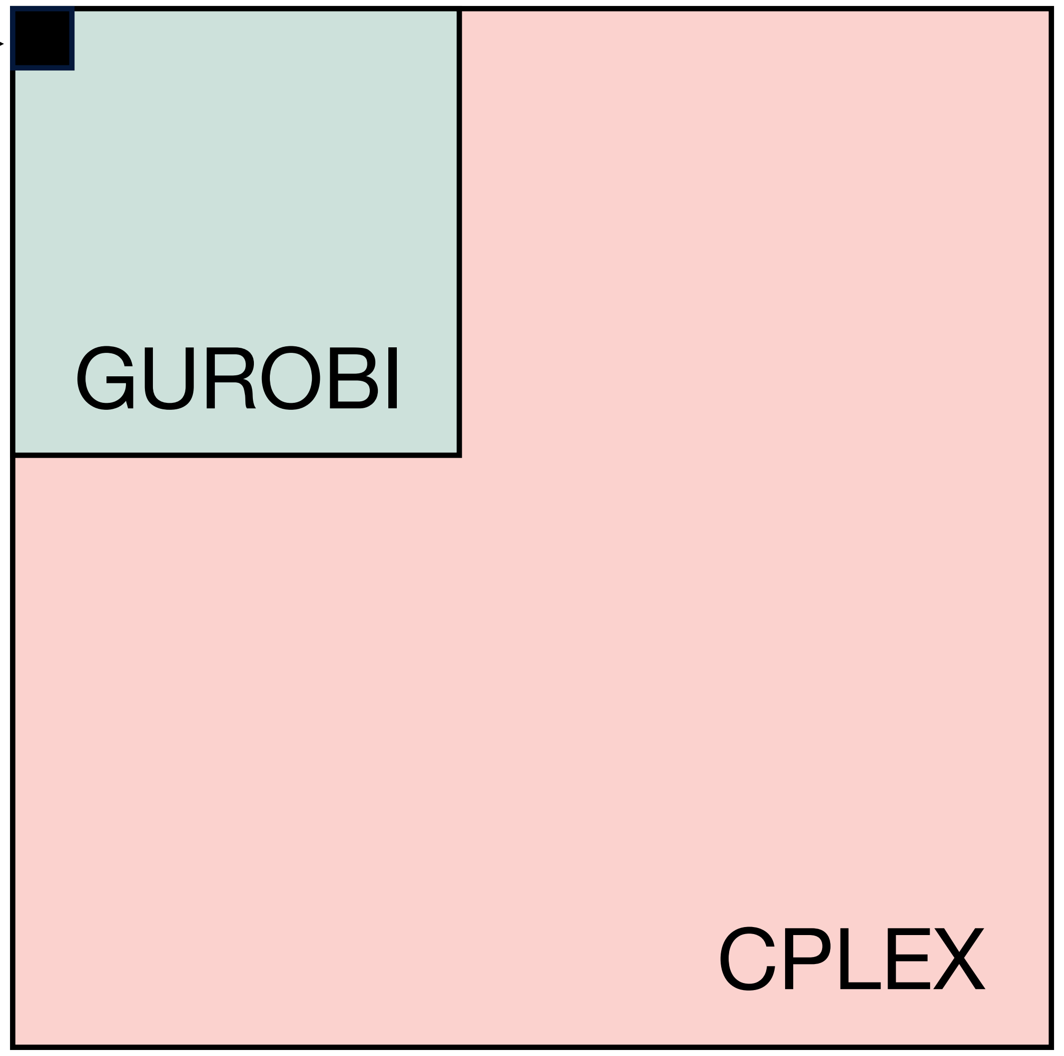
Embedded Hardware



It can be compiled into division-free!

Compiled code size ~80kb (low footprint)

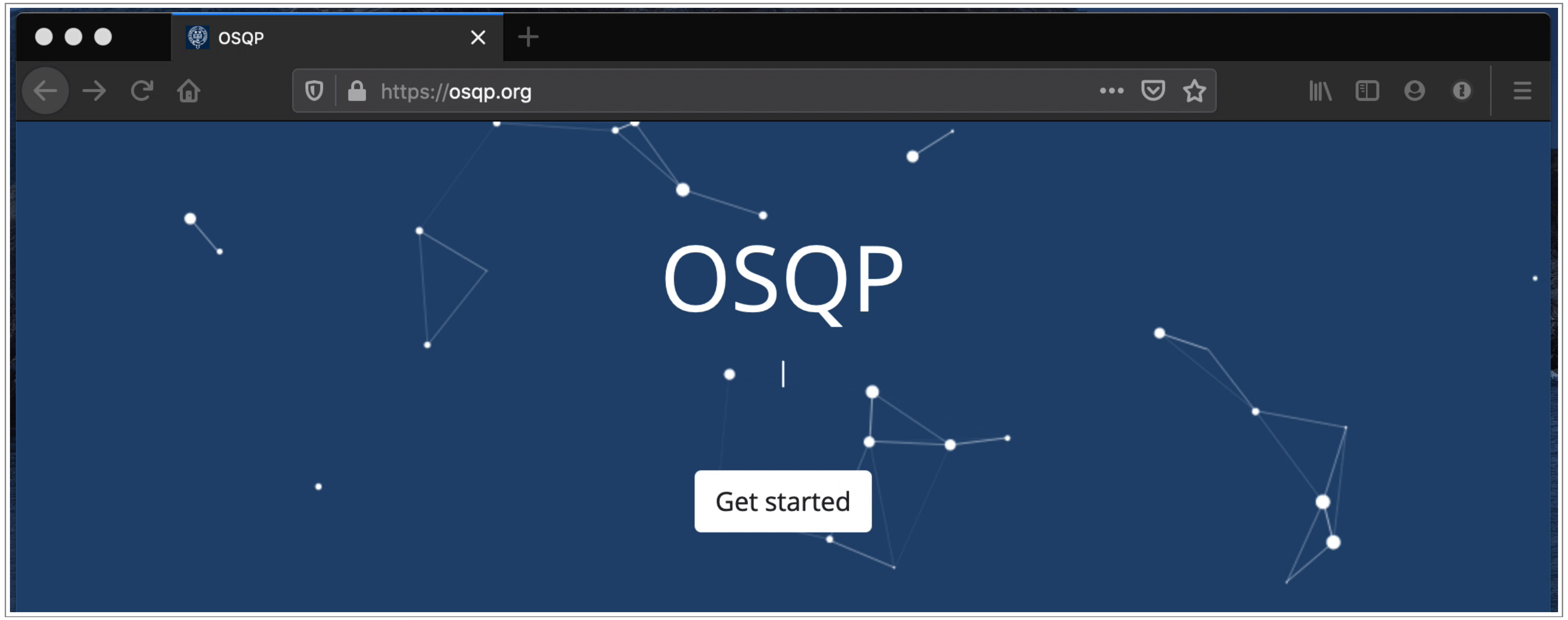
OSQP →



300x Reduction!

OSQP

Operator Splitting solver for Quadratic Programs



Embeddable
(can be division free!)

Supports
warm-starting

Detects
infeasibility

Solves large-scale
problems

Users

More than 16 million downloads!

MIT

Los Alamos
NATIONAL LABORATORY
EST. 1943

Berkeley
UNIVERSITY OF CALIFORNIA

NYU

TOYOTA
Let's Go Places

Google

UCDAVIS
UNIVERSITY OF CALIFORNIA

BLACKROCK®

KU LEUVEN

iit ISTITUTO ITALIANO DI TECNOLOGIA

ETH zürich

LUND UNIVERSITY

APTIV

Stanford University

Universität Stuttgart

[pepy.tech/project/osqp]

How do we ensure fast convergence?

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = z \\ & l \leq z \leq u \end{array}$$

Primal residual

$$r^k_{\text{prim}} = Ax^k - z^k$$

Dual residual

$$r^k_{\text{dual}} = Px^k + q + A^T y^k$$

Intuition

linear system solution with $\sigma = 0$

$$\left(P + \rho A^T A\right) x^{k+1} = -q + A^T(\rho z^k - y^k)$$

$\rho = \infty$ $\qquad\qquad$ $\rho = 0$

$$A^T A x^{k+1} = A^T z^k \qquad\qquad P x^{k+1} = -q - A^T y^k$$

Small primal residual $\qquad\qquad$ **Small dual residual**

What's the optimal ρ?

Constraint-wise step size

minimize $\quad (1/2)x^T P x + q^T x$
subject to $\quad l \leq Ax \leq u$

Tight constraints

$l_i = (Ax^\star)_i$ or $(Ax^\star)_i = u_i$

Diagonal step size

$\rho = (\rho_1, \ldots, \rho_m)$

Never tight

$l_i = -\infty$ and $u_i = \infty$

$\downarrow$

$\rho_i = 0$

Otherwise

$\downarrow$

Balance residuals

$$\rho_i^{k+1} \leftarrow \rho_i^k \sqrt{\|r_{\text{prim}}\| / \|r_{\text{dual}}\|}$$

Always tight

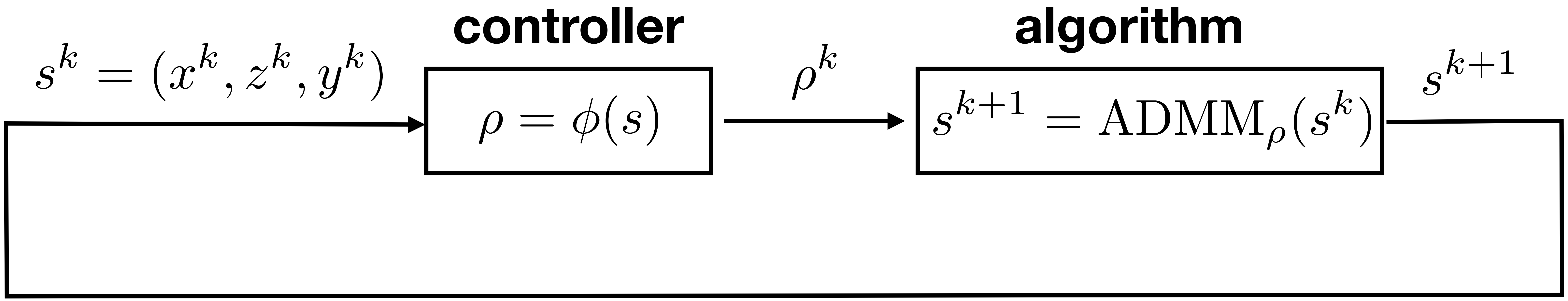
$l_i = u_i \neq \infty$

$\downarrow$

$\rho_i = \infty$

Can we learn a better update rule from data?

Step size choice as a control problem



Stage cost

$$\ell(s) = \begin{cases} 1 & \text{if not converged} \\ 0 & \text{if converged} \end{cases}$$

Cumulative cost

$$J = \mathbf{E} \sum_{k=1}^{\infty} \gamma^k \ell(s^k)$$

Train with
Deep Policy Gradient methods (TD3)

Constraint-wise control policy

$$\phi(s) = \begin{bmatrix} \phi_c(s_1) \\ \phi_c(s_2) \\ \vdots \\ \phi_c(s_m) \end{bmatrix}$$

Per-constraint update rule

$$\rho_i = \phi_c(s_i)$$

Per-constraint state

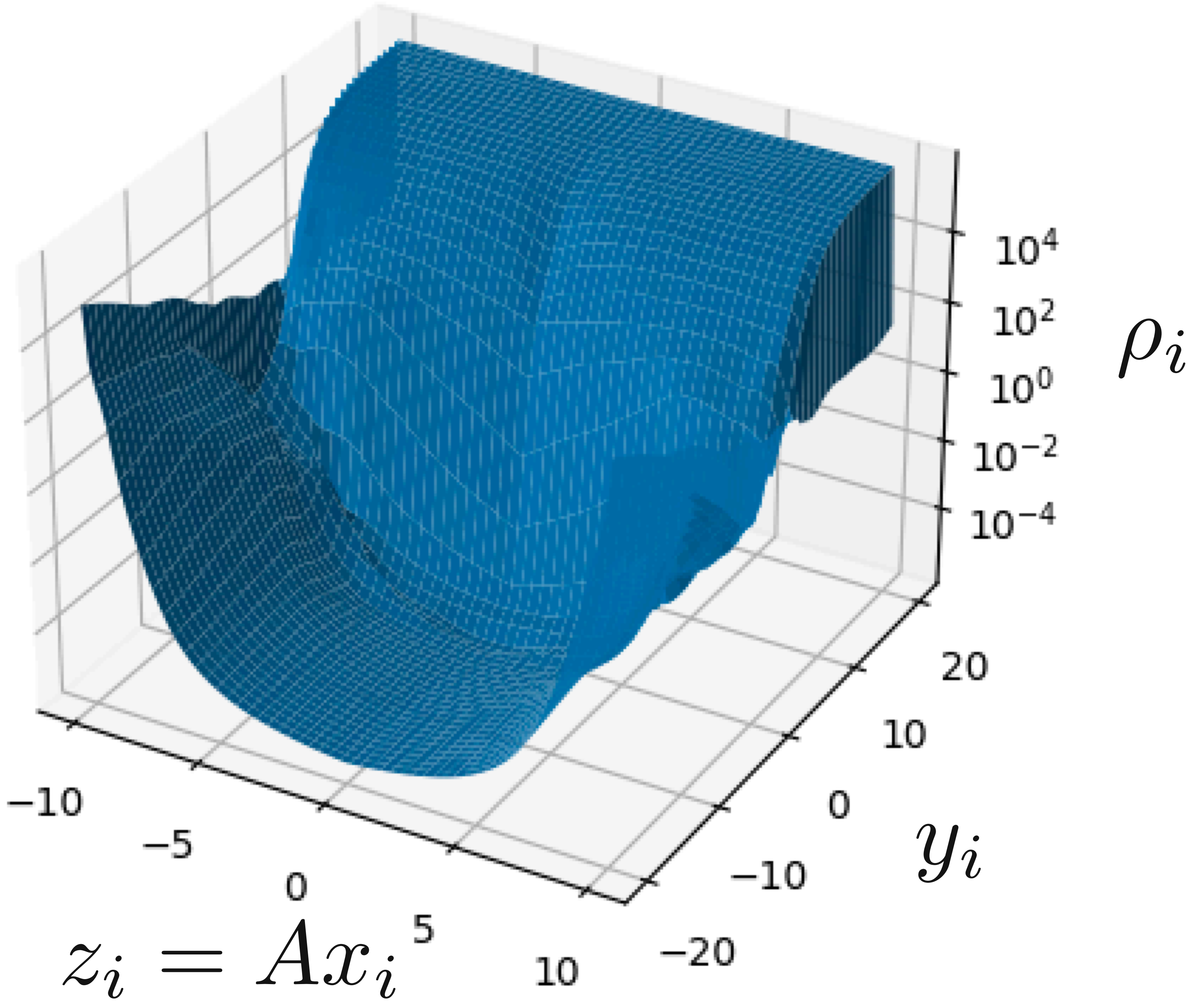
$$s_i = \begin{bmatrix} \min(z_i - l_i, u_i - z_i) \\ (Ax)_i - z_i \\ y_i \end{bmatrix} \begin{matrix} \textbf{slacks} \\ \textbf{infeasibility} \\ \textbf{dual variable} \end{matrix}$$

Generalize to different dimensions

Low-dimensional state per constraint

Small NN policy $\phi_c(s_i)$

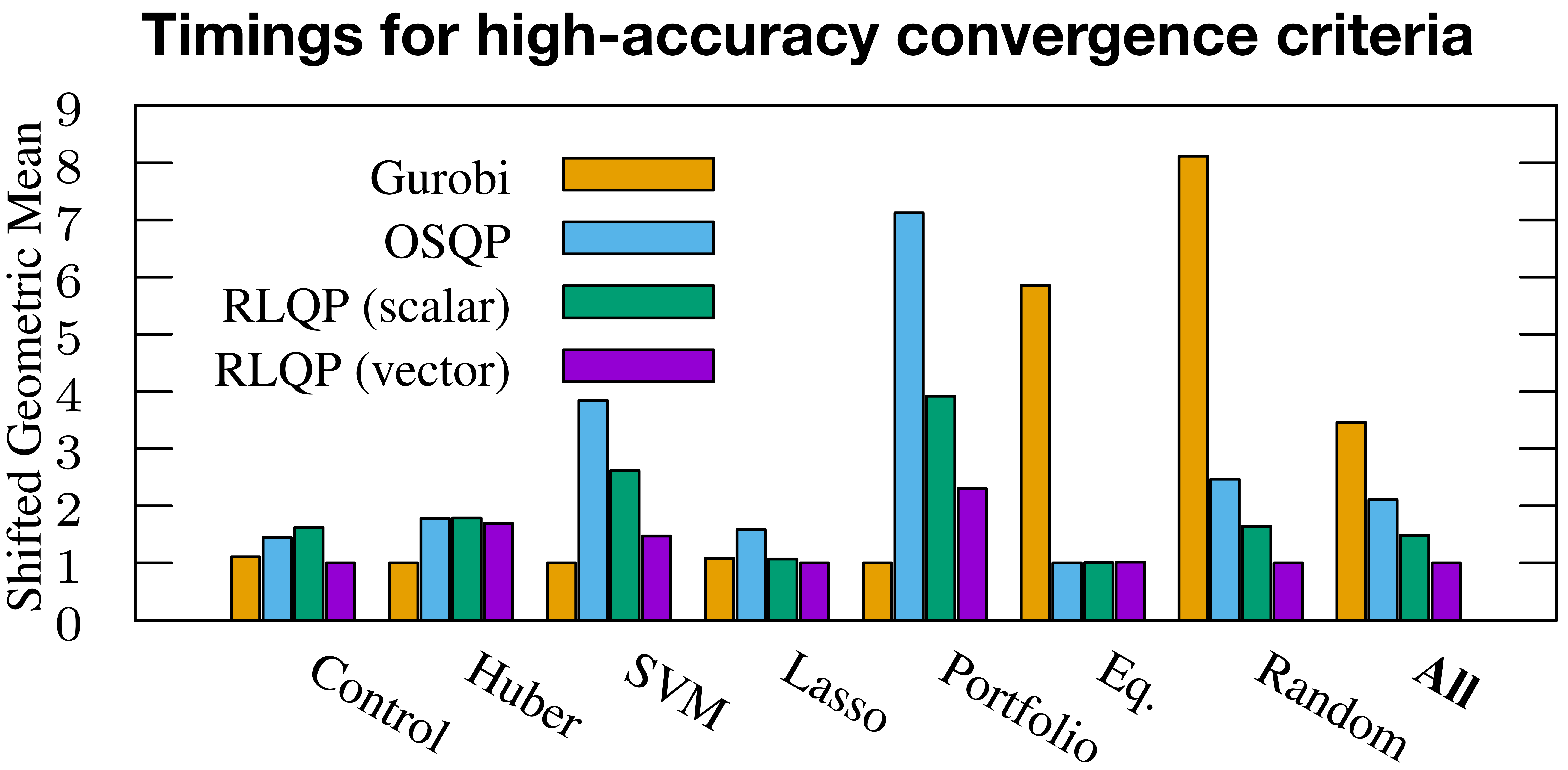
Visualize learned policy



Interpretable policy

High step size ρ_i
when we reach bounds
$z_i \approx l_i$ and $z_i \approx u_i$.

Performance with step size learning

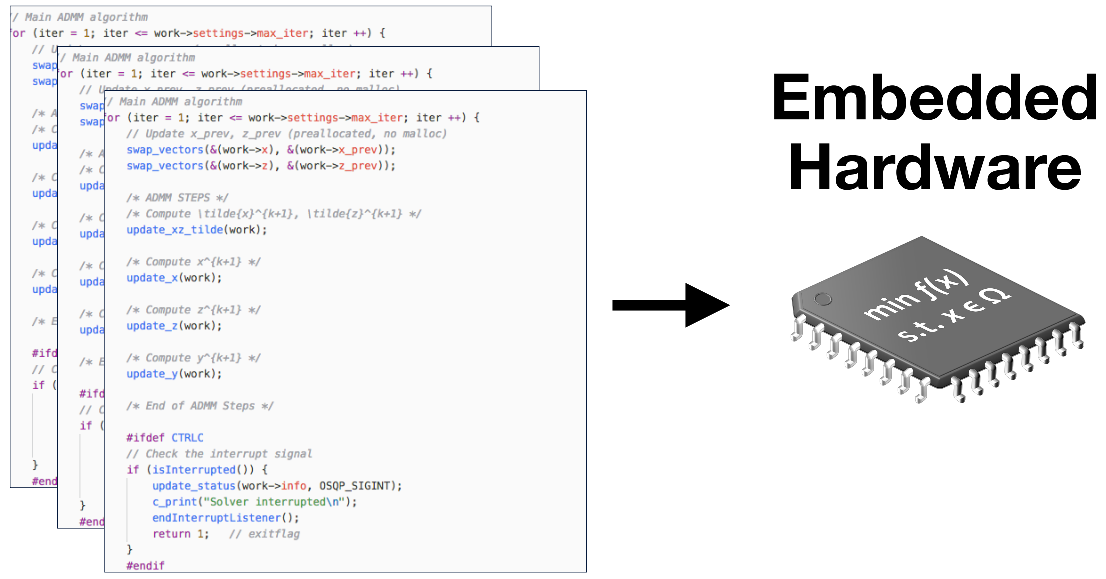


Up to 3x faster than Gurobi

OSQP 1.0 (this summer!)

Improved embedded code generation

```
# Create OSQP object
m = osqp.OSQP()

# Initialize solver
m.setup(P, q, A, l, u,
        settings)

# Generate C code
m.codegen('folder_name')
```

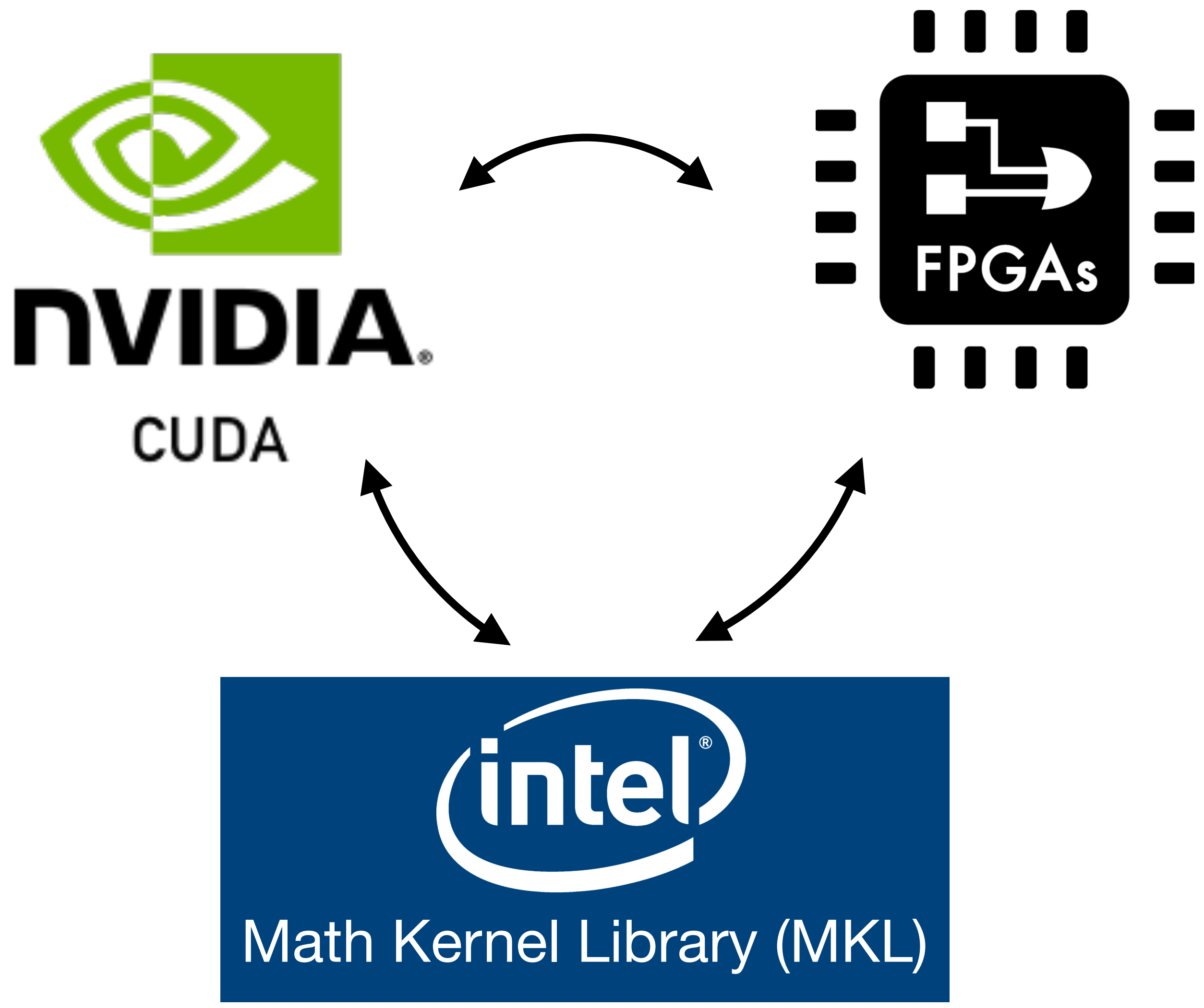


Code generation from C to C

Differentiable layers

PyTorch

Modular linear algebra



OSQP features

Features

- Robust
- Embeddable (can be division free!)
- Detects infeasibility
- Supports warm-starting

Learning for optimization

Integrate RL and ADMM to dynamically tune parameters

- Faster convergence
- Very low overhead
- Interpretable policy

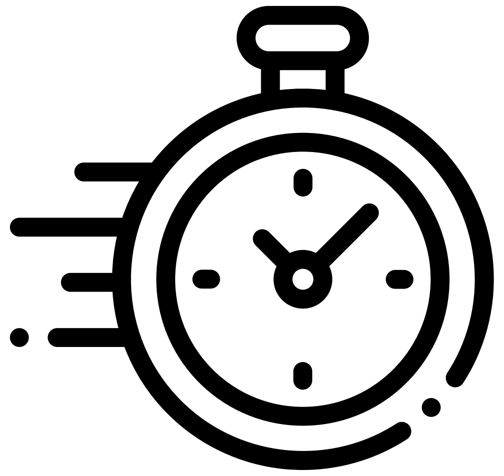
Today's talk

Real-time Decision-Making via Data-Driven Optimization

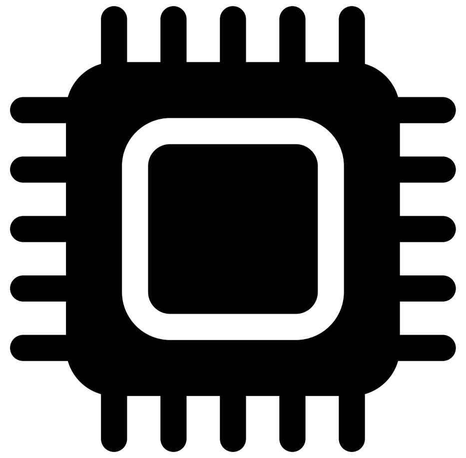
OSQP
Solver

Learning
Convex Optimization
Control Policies

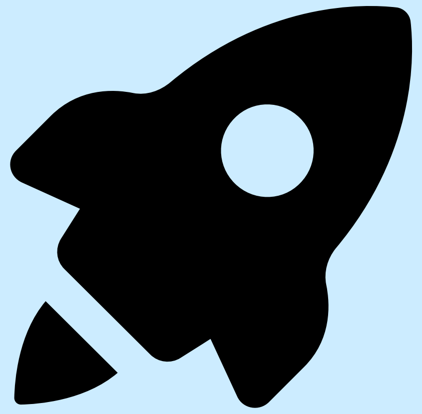
Real-Time



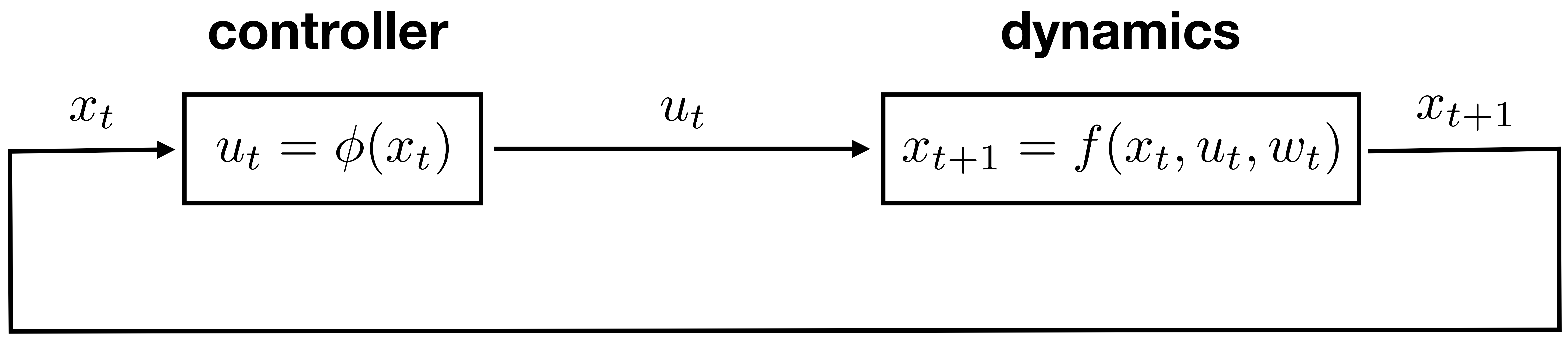
Limited resources



Performance



Control loop



x_t state
u_t input
w_t (random) disturbance

$\phi(x_t)$ **control policy**

Explicit vs implicit control policies

Explicit

Complete control specification

Example: PI Controller

$$u_t = -K_P e_t - K_I \sum_{\tau=0}^{t} e_\tau$$

Implicit (optimization-based)

Designer specifies goal and requirements → **Optimizer** computes the action

Example: LQR Controller

dynamics: $x_{k+1} = Ax_t + Bu_t + w_t$

stage cost: $x^T Q x + u^T R u$

$$\begin{aligned} u_t &= \operatorname*{argmin}_u u^T R u + (Ax_t + Bu)^T P (Ax_t + Bu) \\ &= Kx_t \end{aligned}$$

Convex optimization control policies (COCPs)

$$
\begin{array}{rll}
u_t = & \underset{u}{\text{argmin}} & f(x_t, u, \theta) \\
& \text{subject to} & g(x_t, u, \theta) \le 0 \\
& & A(x_t, \theta)u = b(x_t, \theta)
\end{array}
$$

x_t state
θ parameters to tune
f, g convex functions

Many control policies are COCPs

Examples

- Linear Quadratic Regulator (LQR)
- Model predictive control (MPC)
- Actuator allocation
- Resource allocation
- Portfolio trading

Advantages

Interpretable

Satisfy constraints

Handle varying dynamics

Efficient and reliable (even division-free: OSQP)

Judging COCPs

Given a policy, state and input trajectories form a ***stochastic process***

Trajectories

$$X = (x_0, \ldots, x_{T-1}, x_T)$$
$$U = (u_0, \ldots, u_{T-1})$$
$$W = (w_0, \ldots, w_{T-1})$$

$\longrightarrow$

Policy cost

$$J(\theta) = \mathbf{E}\, \psi(X, U, W)$$

Approximate $J(\theta)$ from data (monte carlo simulation)

$$\hat{J}(\theta) = \frac{1}{K} \sum_{i=1}^{K} \psi(X^i, U^i, W^i)$$

COCP Example: dynamic programming

Time-separable cost

$$\psi(X, U, W) = \sum_{t=0}^{T-1} g(x_t, u_t, w_t)$$

Optimal policy as $T \to \infty$

$$\phi(x_t) = \underset{u}{\text{argmin}}\, \mathbf{E}\left(g(x_t, u, w_t) + V(f(x_t, u, w_t))\right)$$

Value function

COCP if

- f affine in x and u
- g convex in x and u
- V is convex

COCP Example: approximate dynamic programming

$$\phi(x_t) = \underset{u}{\text{argmin}}\, \mathbf{E}\left(g(x_t, u, w_t) + \hat{V}(f(x_t, u, w_t))\right)$$

Approximate value function

COCP if

- f affine in x and u
- g convex in x and u
- $\hat{V}$ is convex

(even when V is not)

Controller tuning problem

Goal

minimize $J(\theta)$

Nonconvex and difficult to solve

Traditional approaches

- **Hand-tuning** (few parameters, simple dependencies)
- **Derivative-free method** (very slow)

Learning scheme

Auto-tuning

Stochastic gradient descent

$$\theta^{k+1} = \theta^k - \underbrace{t^k}_{\text{step size}} \underbrace{\nabla_\theta \hat{J}(\theta^k)}_{\text{stochastic gradient from simulation}}$$

step size

stochastic gradient from simulation

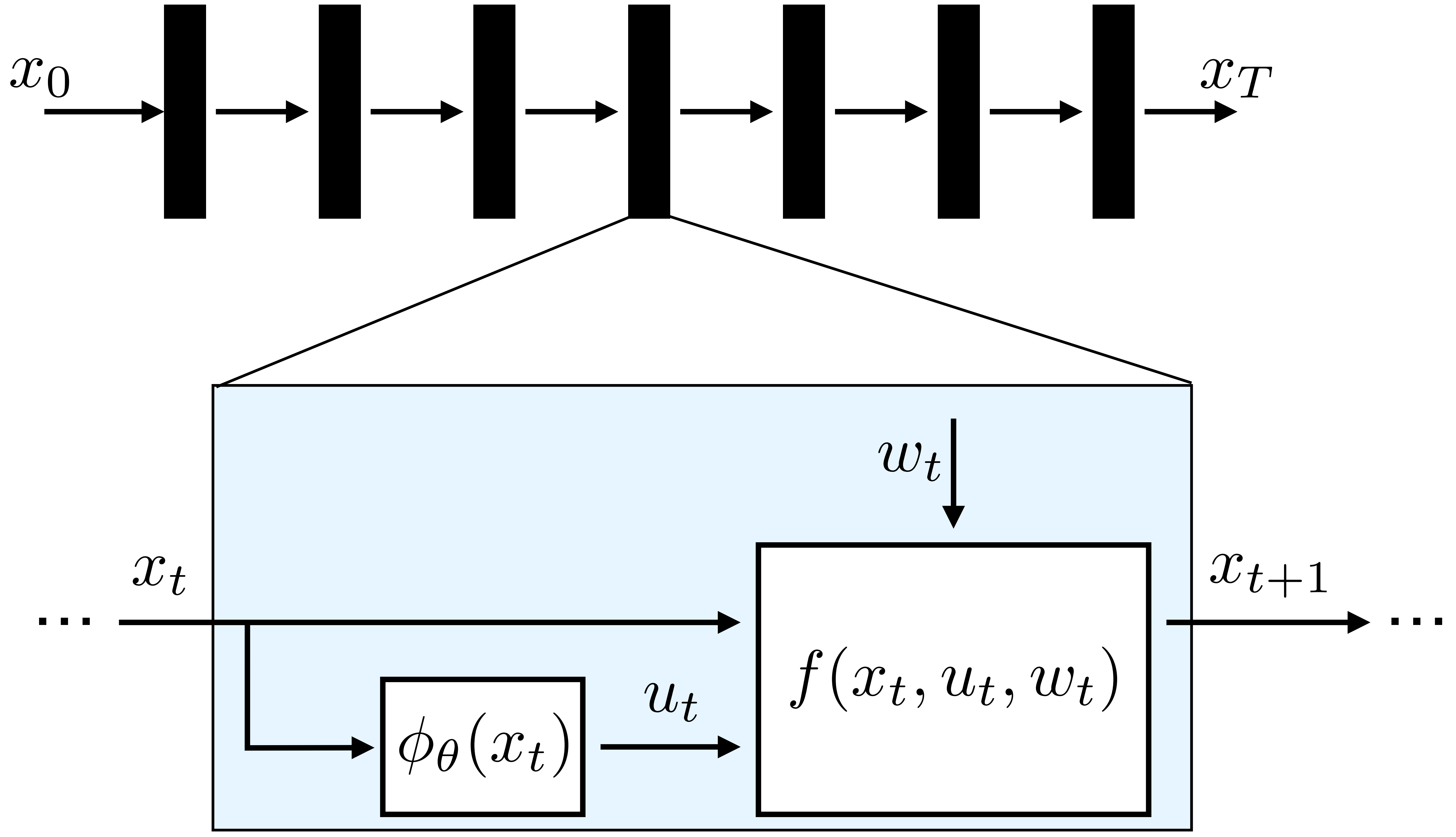
Generalization

Split simulation data in training, validation and testing

Non differentiable $\hat{J}(\theta)$?

Still get a descent direction (common in NN community)

Implementation



Automatic differentiation

- **Build** computation graph (simulate forward in time)
- **Backpropagate** using PyTorch

How do you backpropagate through ϕ_θ ?

Differentiating through convex optimization problems

$$\begin{array}{ll} \text{minimize} & f(x, \theta) \\ \text{subject to} & g(x, \theta) \leq 0 \end{array}$$

- x variable
- θ parameter

Optimality conditions

stationarity $\longrightarrow$ $\nabla_x f(x, \theta) + D_x g(x, \theta)^T y = 0$

complementary slackness $\longrightarrow$ $\mathbf{diag}(y) g(x, \theta) = 0$

$$g(x^\star, \theta) \leq 0$$

$$y^\star \geq 0$$

$\longrightarrow$ $F(z^\star(\theta), \theta) = 0$

primal/dual solution

$$z^\star = (x^\star(\theta), y^\star(\theta))$$

Goal

Compute $Dz^\star(\theta)$

Differentiating through convex optimization problems

$$F(z^\star(\theta), \theta) = 0$$

primal/dual solution

$$z^\star = (x^\star(\theta), y^\star(\theta))$$

Implicit function theorem

$$D_z F(z^\star, \theta) D z^\star(\theta) + D_\theta F(z^\star, \theta) = 0$$

$$\downarrow$$

$$Dz^\star(\theta) = -\left(D_z F(z^\star, \theta)\right)^{-1} D_\theta F(z^\star, \theta)$$

($D_z F(z^\star, \theta)$ must be invertible)

one linear system solution

We plug $Dz^\star(\theta)$ in AD (automatic differentiation)

PyTorch

Box-constrained LQR

Problem setup

- dynamics: $x_{t+1} = Ax_t + Bu_t + w_t$
- actuator limit: $\|u_t\|_\infty \leq 1$
- stage cost: $x_t^T Q x_t + u_t^T R u_t$

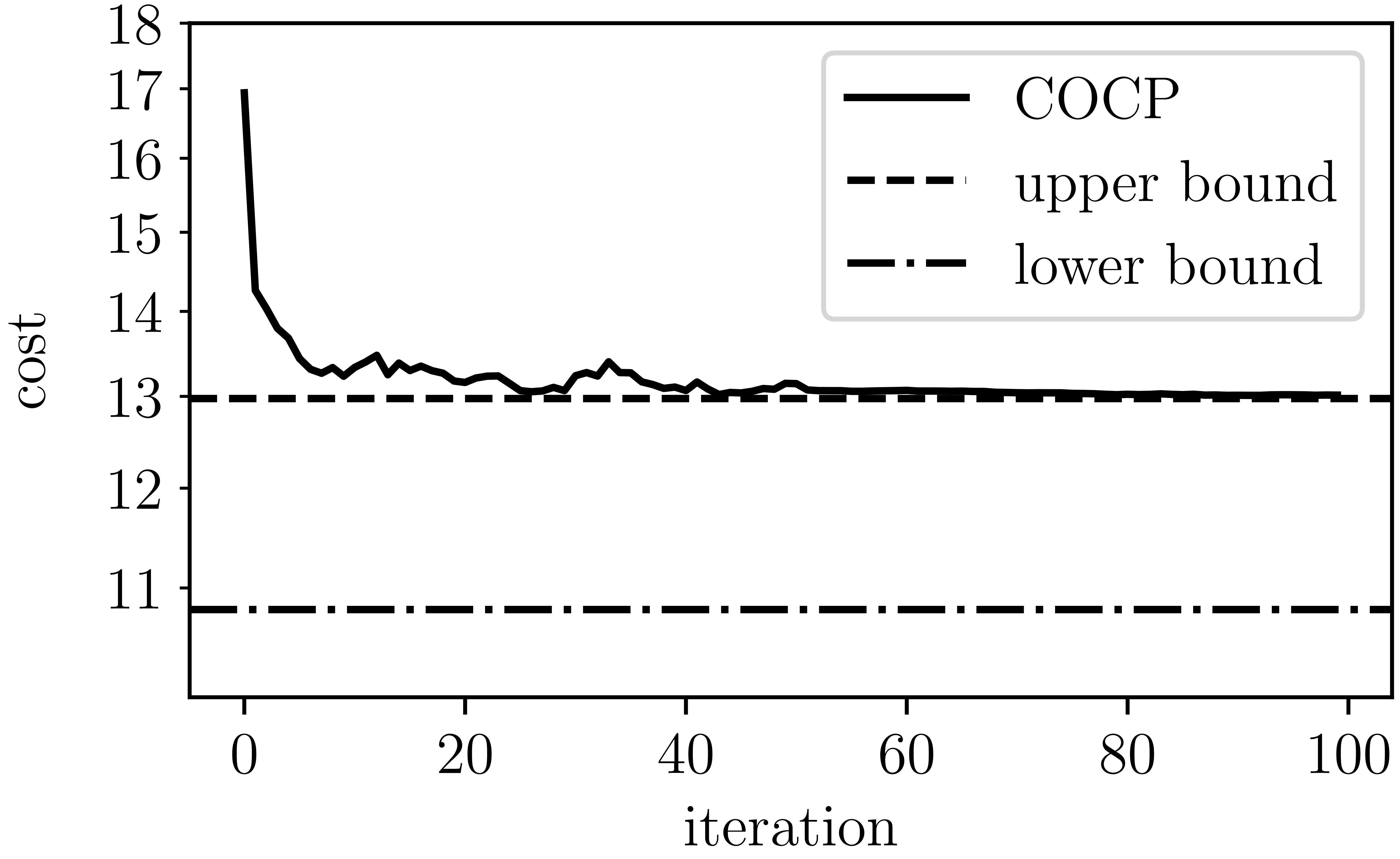
COCP Policy (QP)

$$u_t = \begin{array}{ll} \underset{u}{\text{argmin}} & u^T R u + \|\boxed{\theta}(Ax_t + Bu)\|_2^2 \\ \text{subject to} & \|u\|_\infty \leq 1 \end{array}$$

parameters

Box-constrained LQR

Performance



Standard upper/lower bounds from SDPs

Hard to generalize
(other dynamics, disturbances, etc)

Vehicle tracking curved paths

lateral deviation

heading deviation

path curvature

State: $x_t = (\Delta s_t, \Delta\psi_t, v_t, v_t^{\text{des}}, \kappa_t)$

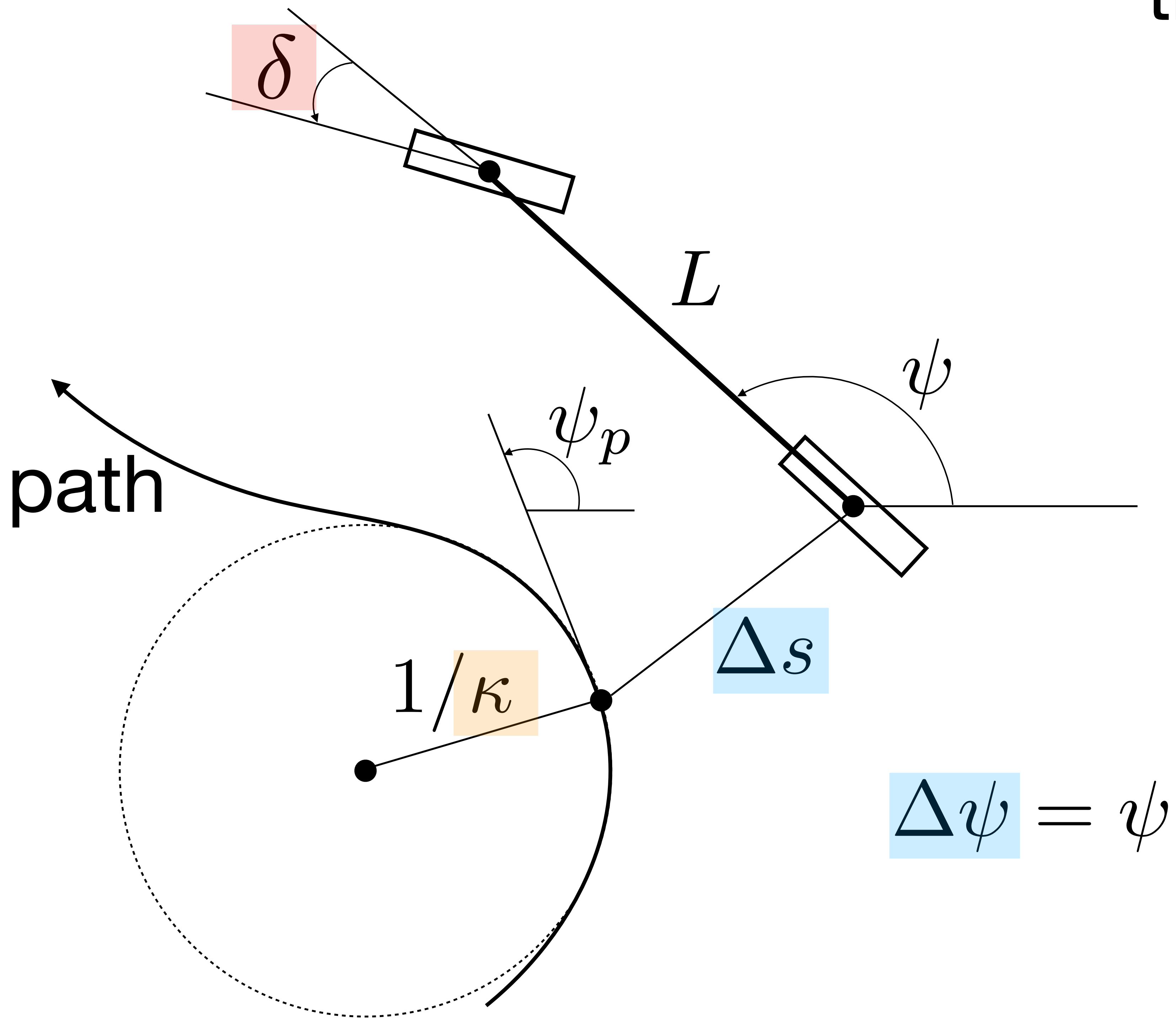
true and desired velocities

acceleration

Input: $u_t = (a_t, z_t)$

turn

$z_t = \tan(\delta_t) - L\kappa_t$



$\Delta\psi = \psi - \psi_p$

Dynamics

$$x_{t+1} = f(x_t, u_t, w_t)$$

Vehicle tracking curved paths

Cost and constraints

Stage cost

track velocity

small control effort

$$\underbrace{(v_t - v_t^{\mathrm{des}})^2}_{\text{track velocity}} + \underbrace{\Delta s_t^2 + \Delta \psi_t^2}_{\text{track path}} + \lambda\underbrace{(|a_t| + z_t^2)}_{\text{small control effort}}$$

track path

Constraints

maximum acceleration

$$|a_t| \leq a_{\max}$$

maximum turn

$$|z + L\kappa_t| \leq \tan(\delta_{\max})$$

Vehicle tracking curved paths

COCP as a Convex QP

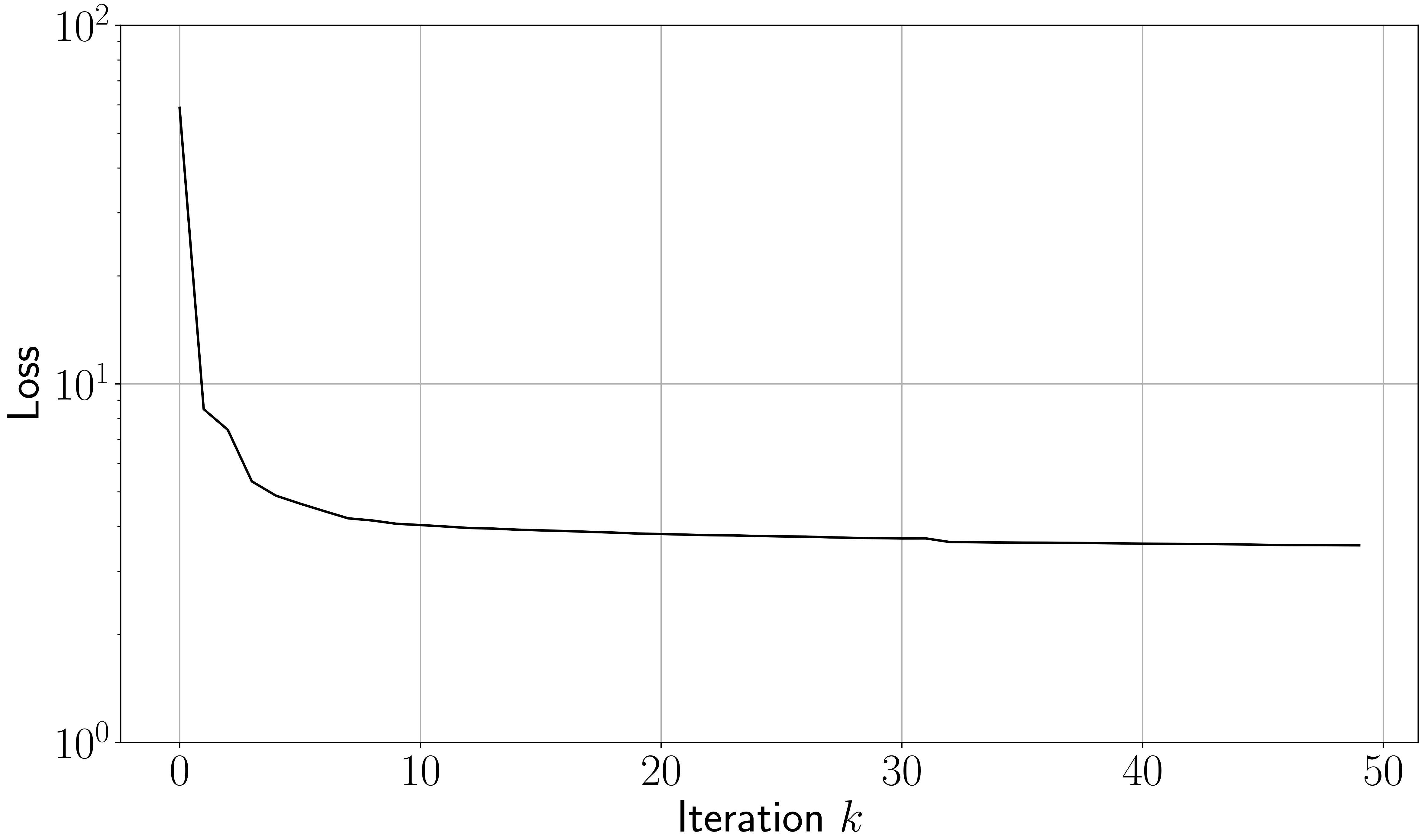
parameters

$$
\begin{array}{rll}
u_t = (a_t, z_t) = \phi(x_t) = & \underset{u}{\text{argmin}} & |a| + z^2 + \|Sy\|_2^2 + q^T y \\
& \text{subject to} & x_t = (\Delta s_t, \Delta \psi_t, v_t, v_t^{\text{des}}, \kappa_t) \\
& & u = (a, z) \\
\textbf{next state} \longrightarrow & & y = f(x_t, u, 0) \\
& & |a| \leq a_{\max} \\
& & |z + L\kappa_t| \leq \tan(\delta_{\max})
\end{array}
$$

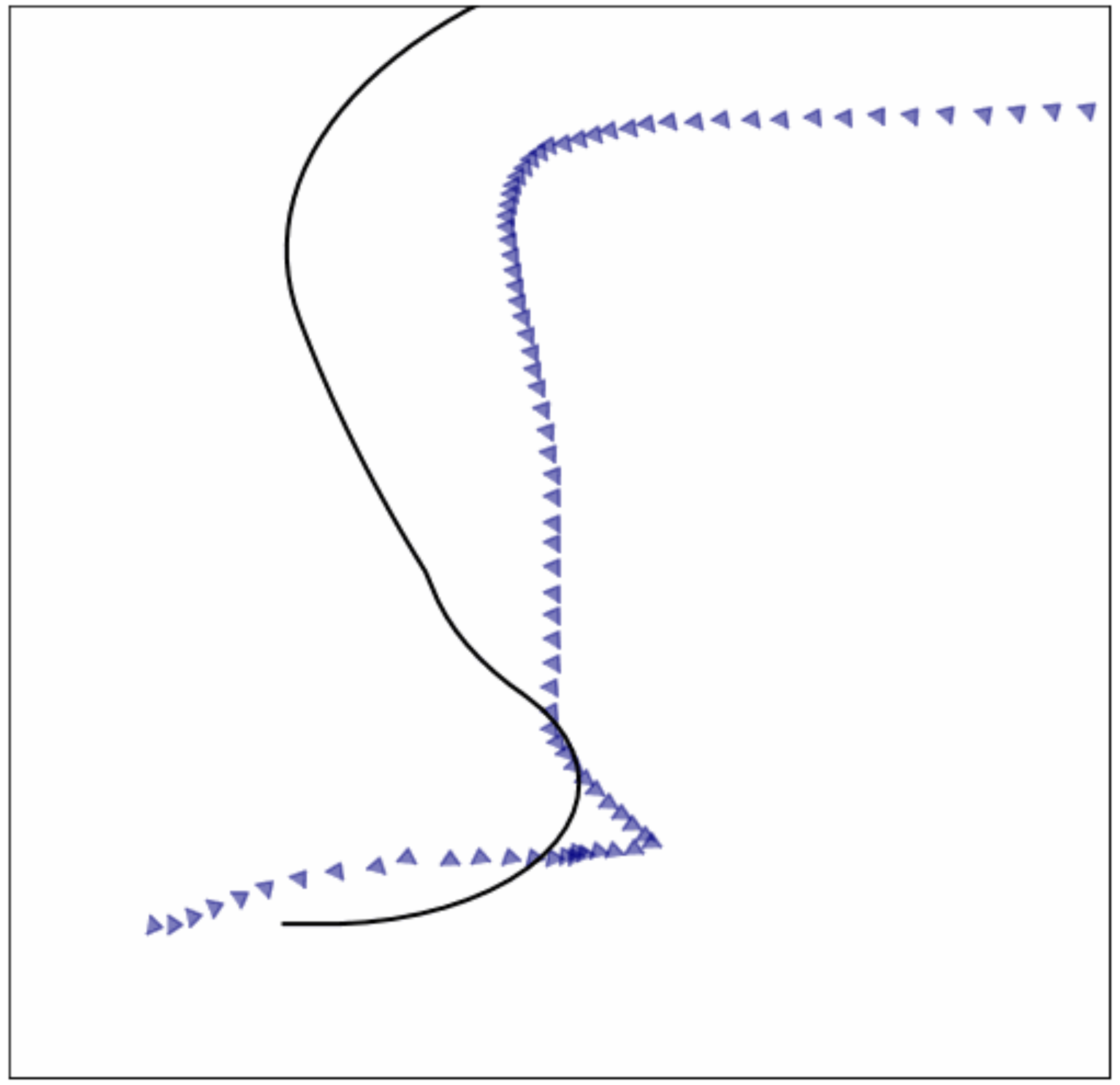
Vehicle tracking curved paths

Results

Validation loss



Tracking behavior

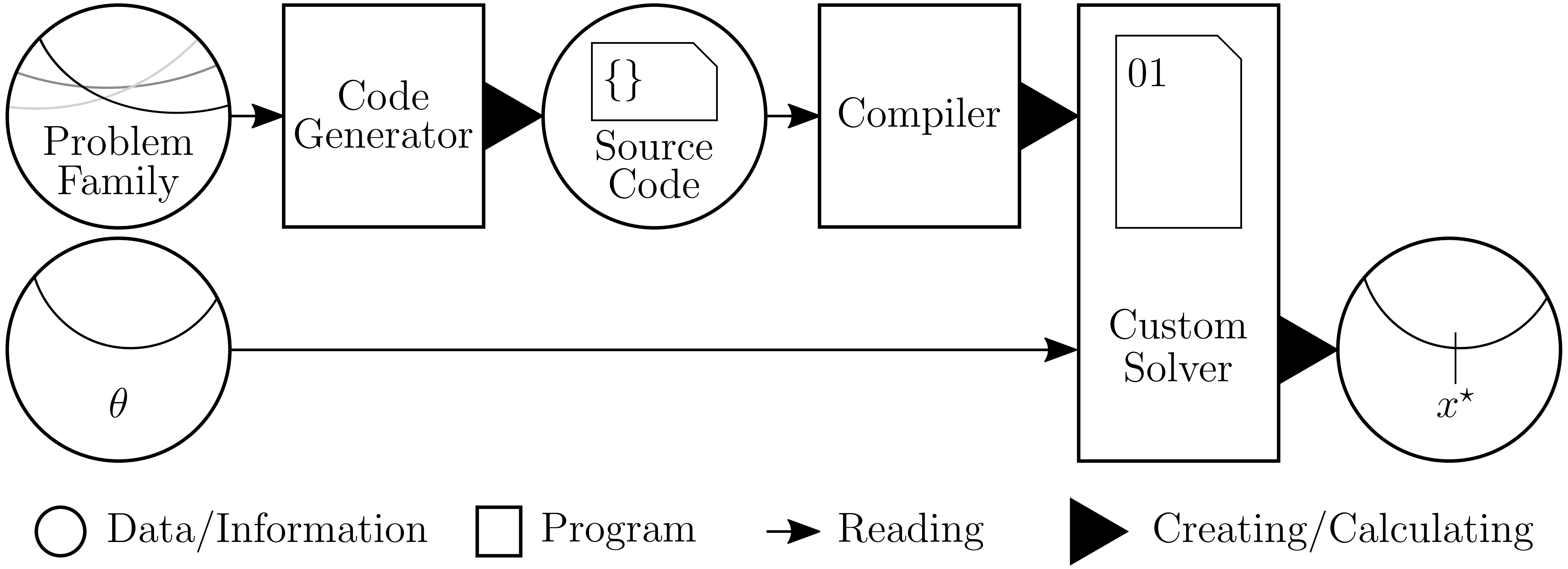


Good trajectory in very few iterations

Code generation for parametric convex optimization

CVXPYgen

`https://pypi.org/project/cvxpygen/`



It supports convex quadratic (OSQP), and conic (ECOS, SCS) optimization.

Example

$$\begin{array}{ll} \text{minimize} & \|Gx - h\|^2 \\ \text{subject to} & x \geq 0 \end{array}$$

$$\theta = (G, h)$$

```
import cvxpy as cp
from cvxpygen import cpg

# model problem
x = cp.Variable(n, name='x')
G = cp.Parameter((m,n), name='G')
h = cp.Parameter(m, name='h')
p = cp.Problem(cp.Minimize(cp.sum_squares(G@x-h)),
               [x>=0])
# generate code
cpg.generate_code(p)
```

Learning COCPs summary

Interpretable

Satisfy constraints

Handle varying dynamics

Efficient and reliable (code generation)

Easy to learn from data

Future work

- Hybrid (mixed-integer) control policies
- Stochastic policies with safety-guarantees

Conclusions

Acknowledgements

Goran Banjac

Paul Goulart

Alberto Bemporad

Ken Goldberg

Stephen Boyd

Akshay Agrawal

Shane Barratt

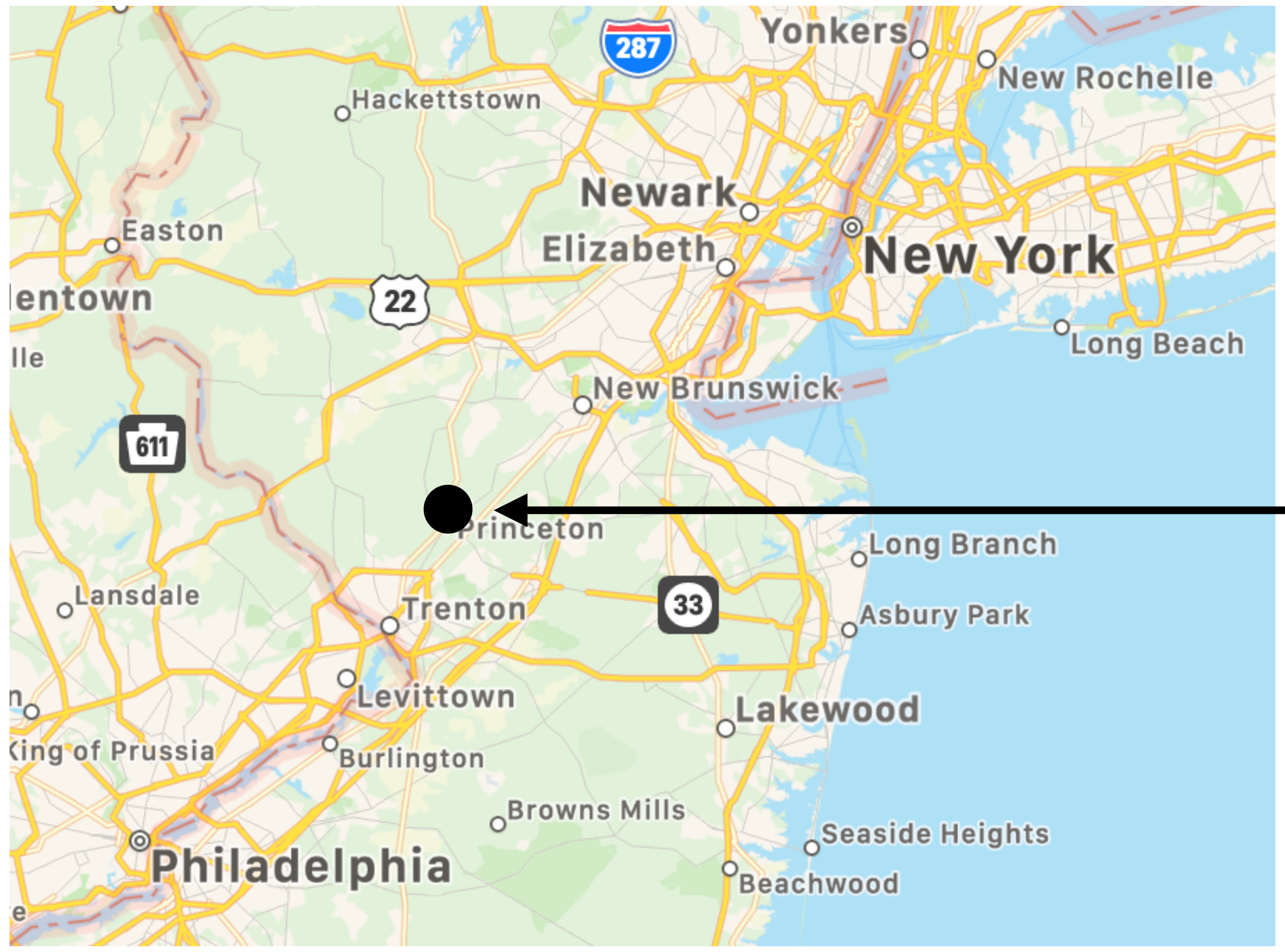
Jeff Ichnowski

Max Schaller

Paras Jain

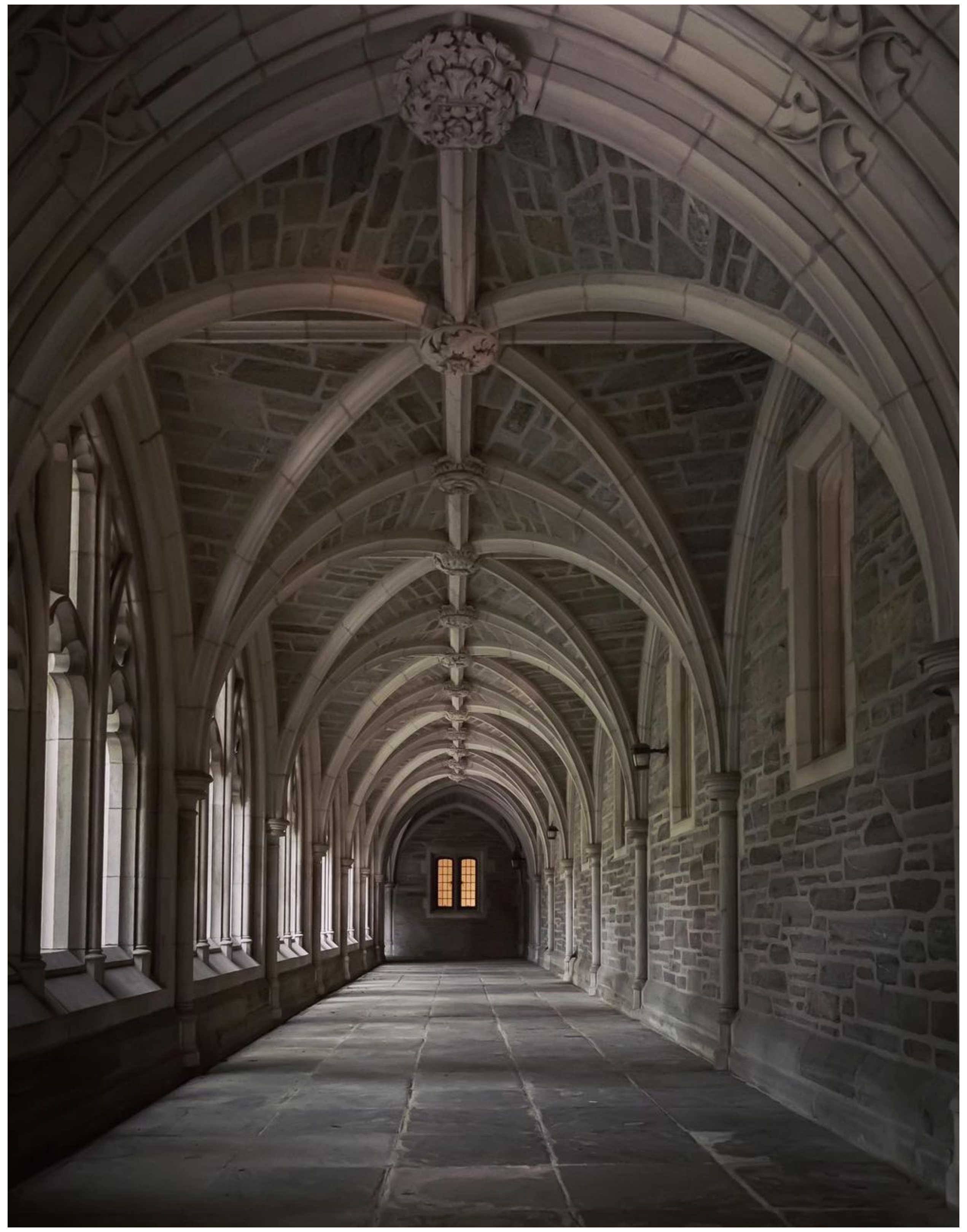
Francesco Borrelli

Come for a visit!



Princeton

Department of
Operations Research and Financial Engineering
orfe.princeton.edu
engineering.princeton.edu



References

OSQP (osqp.org)

[**Accelerating Quadratic Optimization with Reinforcement Learning.** Ichnowski, Jain, *Stellato*, Banjac, Luo, Gonzalez, Stoica, Borrelli, and Goldberg. NeurIPS 2021]

[**OSQP: An Operator Splitting Solver for Quadratic Programs.** *Stellato*, Banjac, Goulart, Bemporad, and Boyd. Mathematical Programming Computation 2020]

[Infeasibility detection in the alternating direction method of multipliers for convex optimization. Banjac, Goulart, *Stellato*, and Boyd. Journal of Optimization Theory and Applications 2019]

[Embedded code generation using the OSQP solver. *Stellato*, Banjac, Stellato, Moehle, Goulart, Bemporad, and Boyd. IEEE Conf. on Decision and Control 2017]

Learning COCPs

[**Embedded Code Generation with CVXPY.** Schaller, Banjac, Diamond, Agrawal, *Stellato*, and Boyd. IEEE Control Systems Letters 2022] (https://pypi.org/project/cvxpygen/)

[**Learning Convex Optimization Control Policies.** Agrawal, Amos, Barratt, Boyd, and Stellato. Learning for Dynamics and Control (L4DC) 2020] (https://github.com/cvxgrp/cocp)

[Differentiable convex optimization layers. Agrawal, Amos, Barratt, Boyd, Diamond, and Kolter. NeurIPS 2019]

Conclusions

Real-time and embedded optimization

will soon **be applied everywhere**

Thanks to

Efficient and reliable optimizers

Data-driven control policies

stellato.io

bstellato@princeton.edu

@b_stellato

github.com/bstellato