Differentiable Predictive Control: From Offline Pre-Training to Safe Online Deployment

Ján Drgoňa
Associate Professor
Civil and Systems Engineering Department
The Ralph O'Connor Sustainable Energy Institute

State of the Art Control Methods

Model Predictive Control

95% of the industrial control applications are PID loops & expert driven rule-based control (RBC).
The rest is model predictive control (MPC).

Reinforcement Learning

Everywhere where you have reliable digital environment model, and you can use compute resources to solve the problem offline. Successful in GenAI, games, and robotics.

James B. Rawlings, David Q. Mayne, Moritz M. Diehl, Model Predictive Control: Theory and Design, Nob Hill Publishing, 2nd ed., 2017
Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2nd ed., 2018
Drgona J., et al. 2020. "All You Need to Know About Model Predictive Control for Buildings." Annual Reviews in Control, September 29, 2020
G. Dulac-Arnold, D. J. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning,", 2019.

Model Predictive Control (MPC)

MPC uses a **receding horizon** approach to compute the optimal control input at each time step by solving a **finite-horizon** optimal control problem in real-time via **online optimization** using numerical methods.

Finite-horizon optimal control problem formulation:

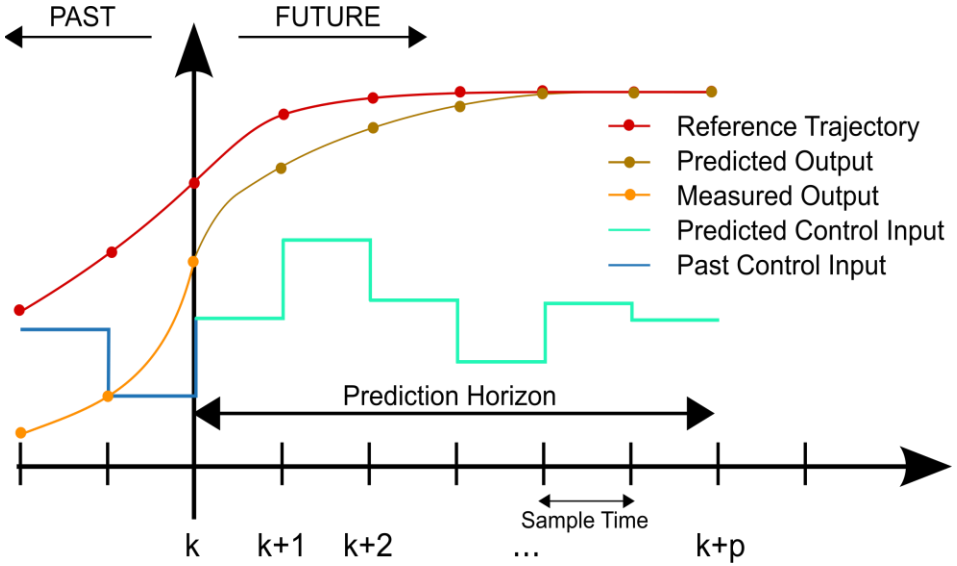
$$\min_{\mathbf{u}_0,\ldots,\mathbf{u}_{N-1}} \sum_{k=0}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k) + p_N(\mathbf{x}_N)$$

$$\text{s.t. } \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k),\ k \in \mathbb{N}_0^{N-1}$$

$$h(\mathbf{x}_k) \leq 0$$

$$g(\mathbf{u}_k) \leq 0$$

$$\mathbf{x}_0 = \mathbf{x}(t)$$



At each time step:

1. Measure the current state x_{current}.
2. Solve the finite-horizon optimization problem for N future steps.
3. Apply only the first control input u_0^*.
4. Move to the next time step and repeat.

James B. Rawlings, David Q. Mayne, Moritz M. Diehl, Model Predictive Control: Theory and Design, Nob Hill Publishing, 2nd ed., 2017

Model Predictive Control (MPC)

MPC uses a **receding horizon** approach to compute the optimal control input at each time step by solving a **finite-horizon** optimal control problem in real-time via **online optimization** using numerical methods.

Finite-horizon optimal control problem formulation:

$$\min_{\mathbf{u}_0,\ldots,\mathbf{u}_{N-1}} \sum_{k=0}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k) + p_N(\mathbf{x}_N)$$

$$\text{s.t. } \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k),\ k \in \mathbb{N}_0^{N-1}$$

$$h(\mathbf{x}_k) \leq 0$$

$$g(\mathbf{u}_k) \leq 0$$

$$\mathbf{x}_0 = \mathbf{x}(t)$$

At each time step:

1. Measure the current state x_{current}.
2. Solve the finite-horizon optimization problem for N future steps.
3. Apply only the first control input u_0^*.
4. Move to the next time step and repeat.

MPC software tools:

PYOMO, MATLAB, JUMP, GEKKO DYNAMIC OPTIMIZATION, CVXPY, CasADi, GUROBI OPTIMIZATION, robust optimal control toolbox mpc

Challenges: Computationally expensive online.

James B. Rawlings, David Q. Mayne, Moritz M. Diehl, Model Predictive Control: Theory and Design, Nob Hill Publishing, 2nd ed., 2017

Reinforcement Learning (RL)

RL is a **data-driven decision-making framework** where an agent interacts with an environment to learn an optimal policy that maximizes rewards over **infinite-horizon**. Most modern **policy-gradient methods** (actor–critic, PPO, TRPO, DDPG, TD3, offline RL) **learn a critic** and use it to **update the actor (policy)**.

1, **Data collection** via sampling:

$$
\begin{aligned}
&x_t \sim \text{environment} \\
&u_t = \pi_\theta(x_t) \\
&x_{t+1},\ r_t \sim \mathcal{P}(\cdot \mid x_t, u_t) \\
&(x_t,\ u_t,\ r_t,\ x_{t+1}) \in \mathcal{D}
\end{aligned}
$$

2, **Critic Update:**

Q-function

$$Q^\pi(x,u) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_k \mid x_0 = x, u_0 = u, \pi\right]$$

Bellman target

$$y_t = r_t + \gamma Q_{\phi'}(x_{t+1},\ \pi_{\theta'}(x_{t+1}))$$

Critic loss function

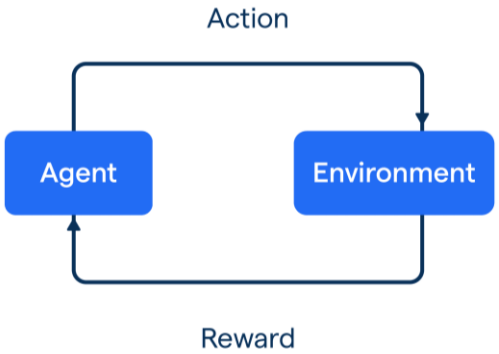
$$\mathcal{L}_{\text{critic}}(\phi) = \left(Q_\phi(x_t, u_t) - y_t\right)^2$$

3, **Actor update** via deterministic policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\left[\nabla_u Q_\phi(x,u)\big|_{u=\pi_\theta(x)} \cdot \nabla_\theta \pi_\theta(x)\right]$$

4, **Gradient descent updates**:

$$
\begin{aligned}
\phi' &\leftarrow \tau\phi + (1-\tau)\phi' \\
\theta' &\leftarrow \tau\theta + (1-\tau)\theta'
\end{aligned}
$$



Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2nd ed., 2018

Reinforcement Learning (RL)

RL is a **data-driven decision-making framework** where an agent interacts with an environment to learn an optimal policy that maximizes rewards over **infinite-horizon**. Most modern **policy-gradient methods** (actor–critic, PPO, TRPO, DDPG, TD3, offline RL) **learn a critic** and use it to **update the actor (policy)**.

1, **Data collection** via sampling:

$$
\begin{aligned}
&x_t \sim \text{environment} \\
&u_t = \pi_\theta(x_t) \\
&x_{t+1},\ r_t \sim \mathcal{P}(\cdot \mid x_t, u_t) \\
&(x_t,\ u_t,\ r_t,\ x_{t+1}) \in \mathcal{D}
\end{aligned}
$$

2, **Critic Update:**

Q-function

$$Q^\pi(x,u) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_k \mid x_0 = x, u_0 = u, \pi\right]$$

Bellman target

$$y_t = r_t + \gamma Q_{\phi'}(x_{t+1},\ \pi_{\theta'}(x_{t+1}))$$

Critic loss function

$$\mathcal{L}_{\text{critic}}(\phi) = \left(Q_\phi(x_t, u_t) - y_t\right)^2$$

3, **Actor update** via deterministic policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\left[\nabla_u Q_\phi(x,u)\big|_{u=\pi_\theta(x)} \cdot \nabla_\theta \pi_\theta(x)\right]$$

4, **Gradient descent updates**:

$$
\begin{aligned}
\phi' &\leftarrow \tau\phi + (1-\tau)\phi' \\
\theta' &\leftarrow \tau\theta + (1-\tau)\theta'
\end{aligned}
$$

RL software tools:

PyTorch TensorFlow

Gym

Challenges: Sample inefficient! Hard to obtain guarantees.

Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2nd ed., 2018

Model-Based Reinforcement Learning (MBRL)

MBRL aims to **improve sample efficiency** by **learning a predictive model** of the environment dynamics and using it to **simulate trajectories** for Q-function fitting and policy optimization. This framework is followed by methods such as MBPO, TD3-MB, MOPO, COMBO, DreamerV3, and Plan2Explore.

1, **Model rollouts** via sampling:

$$\hat{x}_0 = x \sim \mathcal{D}$$
$$\hat{u}_t \sim \pi_\theta(\cdot \mid \hat{x}_t)$$
$$\hat{x}_{t+1} = f_\psi(\hat{x}_t, \hat{u}_t)$$
$$\hat{r}_t = \hat{r}_\psi(\hat{x}_t, \hat{u}_t)$$

2, **Critic Update** using model rollouts:

Bellman target $y_t = \hat{r}_t + \gamma Q_{\phi'}(\hat{x}_{t+1}, \pi_{\theta'}(\hat{x}_{t+1}))$

Critic loss function $\mathcal{L}_{\text{critic}}(\phi) = (Q_\phi(\hat{x}_t, \hat{u}_t) - y_t)^2$

3, **Actor update** via deterministic policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\hat{x}_t \sim \mathcal{D}} \left[\nabla_u Q_\phi(\hat{x}_t, u)\big|_{u=\pi_\theta(\hat{x}_t)} \cdot \nabla_\theta \pi_\theta(\hat{x}_t) \right]$$

4, **Gradient descent updates**:

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$
$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

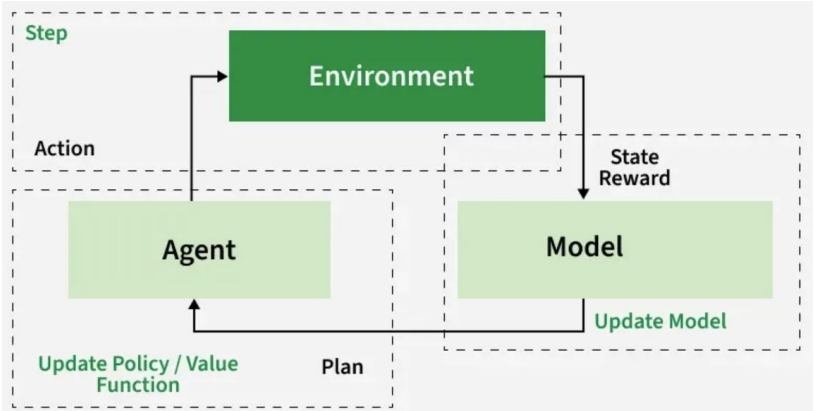


Image: www.geeksforgeeks.org/artificial-intelligence/model-based-reinforcement-learning-mbrl-in-ai/

Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2nd ed., 2018

Model-Based Reinforcement Learning (MBRL)

MBRL aims to **improve sample efficiency** by **learning a predictive model** of the environment dynamics and using it to **simulate trajectories** for Q-function fitting and policy optimization. This framework is followed by methods such as MBPO, TD3-MB, MOPO, COMBO, DreamerV3, and Plan2Explore.

1, **Model rollouts** via sampling:

$$\hat{x}_0 = x \sim \mathcal{D}$$
$$\hat{u}_t \sim \pi_\theta(\cdot \mid \hat{x}_t)$$
$$\hat{x}_{t+1} = f_\psi(\hat{x}_t, \hat{u}_t)$$
$$\hat{r}_t = \hat{r}_\psi(\hat{x}_t, \hat{u}_t)$$

2, **Critic Update** using model rollouts:

Bellman target $y_t = \hat{r}_t + \gamma Q_{\phi'}(\hat{x}_{t+1}, \pi_{\theta'}(\hat{x}_{t+1}))$

Critic loss function $\mathcal{L}_{\text{critic}}(\phi) = (Q_\phi(\hat{x}_t, \hat{u}_t) - y_t)^2$

Why to learn the critic if we have accurate and differentiable model and parametric reward functions?

3, **Actor update** via deterministic policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\hat{x}_t \sim \mathcal{D}} \left[\nabla_u Q_\phi(\hat{x}_t, u)\big|_{u=\pi_\theta(\hat{x}_t)} \cdot \nabla_\theta \pi_\theta(\hat{x}_t) \right]$$

4, **Gradient descent updates**:

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$
$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

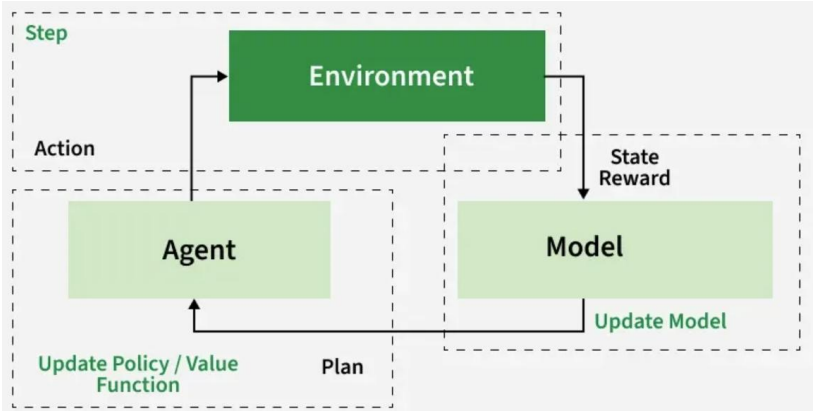


Image: www.geeksforgeeks.org/artificial-intelligence/model-based-reinforcement-learning-mbrl-in-ai/

Differentiable Predictive Control (DPC)

DPC is closely related to **MBRL** in that both leverage **model of dynamics**, but DPC utilizes **differentiable** closed-loop models and cost functions, allowing direct policy gradients **without requiring a learned critic**.

Empirical risk minimization problem:

$$\min_{\theta} \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\xi}_k} \left(\sum_{k=0}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\xi}_k) + p_N(\mathbf{x}_N) \right)$$

$$\text{s.t. } \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\xi}_k), \;\; k \in \mathbb{N}_0^{N-1}$$

$$\mathbf{u}_k = \pi_\theta(\mathbf{x}_k, \boldsymbol{\xi}_k)$$

$$h(\mathbf{x}_k, \boldsymbol{\xi}_k) \leq 0$$

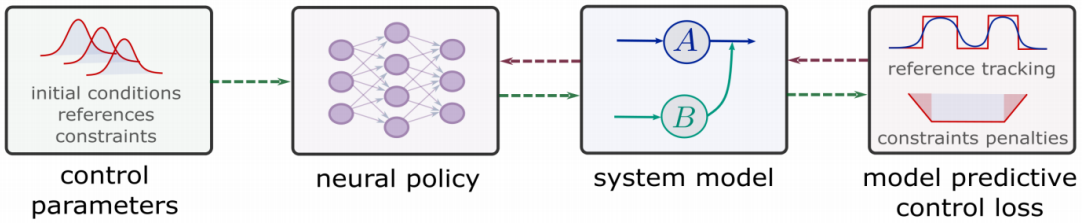
$$g(\mathbf{u}_k, \boldsymbol{\xi}_k) \leq 0$$

Critic parametrized by differentiable MPC-like loss function:

$$\mathcal{L}_{\text{DPC}} = \frac{1}{mN} \sum_{i=1}^{m} \left(\sum_{k=0}^{N-1} \left(\ell(\mathbf{x}_k^i, \mathbf{u}_k^i, \boldsymbol{\xi}_k^i) + p_x(h(\mathbf{x}_k^i, \boldsymbol{\xi}_k^i)) + p_u(g(\mathbf{u}_k^i, \boldsymbol{\xi}_k^i)) \right) + p_N(\mathbf{x}_N^i) \right)$$

Policy gradient via automatic differentiation:

$$\nabla_\theta \mathcal{L}_{\text{DPC}} = \left(\frac{\partial \ell}{\partial \mathbf{x}} + \frac{\partial p_x}{\partial \mathbf{x}} + \frac{\partial p_N}{\partial \mathbf{x}} \right) \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial \theta} + \left(\frac{\partial \ell}{\partial \mathbf{u}} + \frac{\partial p_u}{\partial \mathbf{u}} \right) \cdot \frac{\partial \mathbf{u}}{\partial \theta}$$



$\ell(\cdot)$: stage cost
$p_x(\cdot)$, $p_u(\cdot)$: constraints penalties
$p_N(\cdot)$: terminal cost
$\frac{\partial \mathbf{x}}{\partial \mathbf{u}}$: differentiable dynamics
$\frac{\partial \mathbf{u}}{\partial \theta}$: differentiable policy

J. Drgoňa, A. Tuor and D. Vrabie, *Learning Constrained Parametric Differentiable Predictive Control Policies With Guarantees, in IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2024*

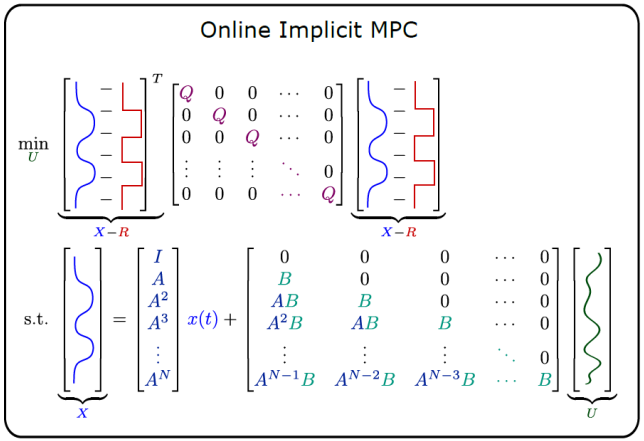
J. Drgoňa, et al., *Differentiable Predictive Control: An MPC Alternative for Unknown Nonlinear Systems using Constrained Deep Learning*, Journal of Process Control, 2022

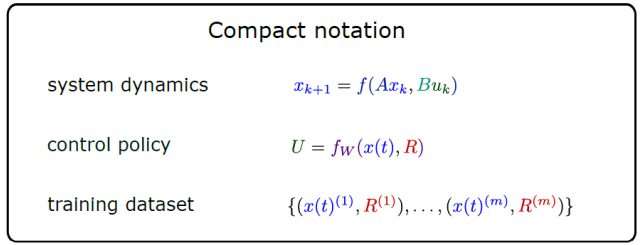
Differentiable Predictive Control (DPC)

DPC is also closely related to **explicit MPC**, as it solves a **parametric optimal control problem**; but instead of on intractable multi-parametric programming, it employs direct **gradient-based** policy optimization.

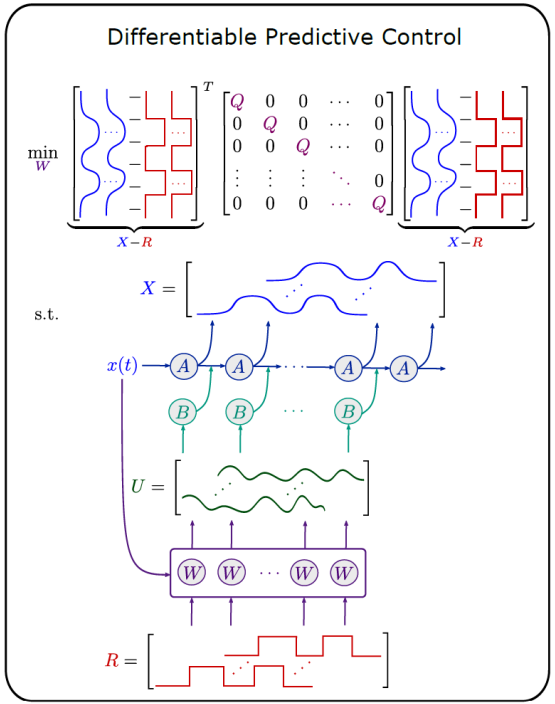
There is a **structural equivalence** between **single shooting** formulation of MPC problem and the **unrolled closed-loop system** dynamics in DPC.

But instead of online optimization as in implicit MPC, the **DPC learns explicit policy offline**, similarly to RL.





structural equivalence



J. Drgoňa, A. Tuor and D. Vrabie, *Learning Constrained Parametric Differentiable Predictive Control Policies With Guarantees, in IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2024*

J. Drgoňa, et al., *Differentiable Predictive Control: An MPC Alternative for Unknown Nonlinear Systems using Constrained Deep Learning*, Journal of Process Control, 2022

Stochastic Differentiable Predictive Control

DPC naturally generalizes to handle parametric stochastic optimal control problem formulations via **joint sampling of control parameters and uncertainties**.

$$\min_{\theta} \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\xi}_k, \boldsymbol{\omega}_k} \left(\sum_{k=0}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\xi}_k) + p_N(\mathbf{x}_N) \right)$$

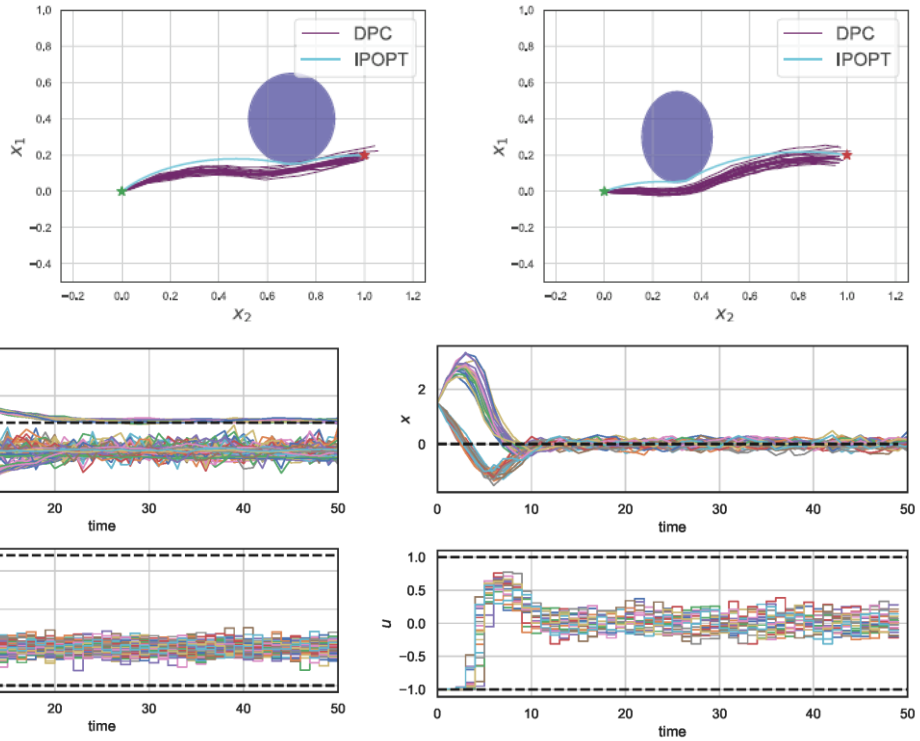
$$\text{s.t. } \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\xi}_k, \boldsymbol{\omega}_k), \;\; k \in \mathbb{N}_0^{N-1}$$

$$\mathbf{u}_k = \pi_\theta(\mathbf{x}_k, \boldsymbol{\xi}_k)$$

$$\Pr(h(\mathbf{x}_k, \boldsymbol{\xi}_k) \leq 0) \geq \beta$$

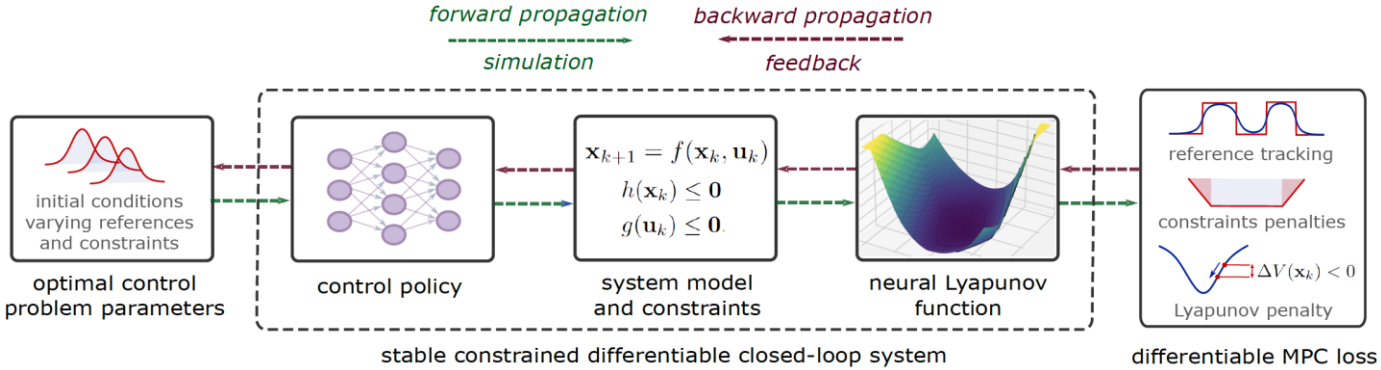
$$\Pr(g(\mathbf{u}_k, \boldsymbol{\xi}_k) \leq 0) \geq \beta$$

Can handle both parametric and additive uncertainties, and wide range of task parametrizations such as reference tracking, stabilization, time-varying constraints, or obstacle avoidance.



Ján Drgoňa, Sayak Mukherjee, Aaron Tuor, Mahantesh Halappanavar, Draguna Vrabie, Learning Stochastic Parametric Differentiable Predictive Control Policies, IFAC-PapersOnLine, Volume 55, Issue 25, 2022

DPC with Neural Lyapunov Functions



Joint learning of neural policy and Lyapunov function:

$$\min_{\theta,\phi} \frac{1}{mN} \sum_{i=1}^{m} \sum_{k=0}^{N-1} \left(\ell(\mathbf{x}_k^i, \mathbf{u}_k^i) + Q_V p_V(\mathbf{x}_{k+1}^i, \mathbf{x}_k^i) + \right.$$

$$\left. Q_x p_x(\mathbf{x}_k^i) + Q_u p_u(\mathbf{u}_k^i) \right),$$

$$\text{s.t. } \mathbf{x}_{k+1}^i = f(\mathbf{x}_k^i, \mathbf{u}_k^i), \ k \in \mathbb{N}_0^{N-1},$$

$$[\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}]^i = \boldsymbol{\pi}_\theta(\mathbf{x}_k^i),$$

$$\mathbf{x}_0^i \in \mathbb{X} \subseteq \mathbb{R}^{n_x} \sim \mathcal{D}.$$

Neural Lyapunov function candidate via input-convex neural network (ICNN) architecture:

$$\mathbf{z}_1 = \sigma_0 \left(\mathbf{W}_0 \mathbf{x} + \mathbf{b}_0 \right),$$

$$\mathbf{z}_{i+1} = \sigma_i \left(\mathbf{U}_i \mathbf{z}_i + \mathbf{W}_i \mathbf{x} + \mathbf{b}_i \right), i = 1, \ldots, k-1,$$

$$\mathbf{g}(\mathbf{x}) \equiv \mathbf{z}_\mathbf{k},$$

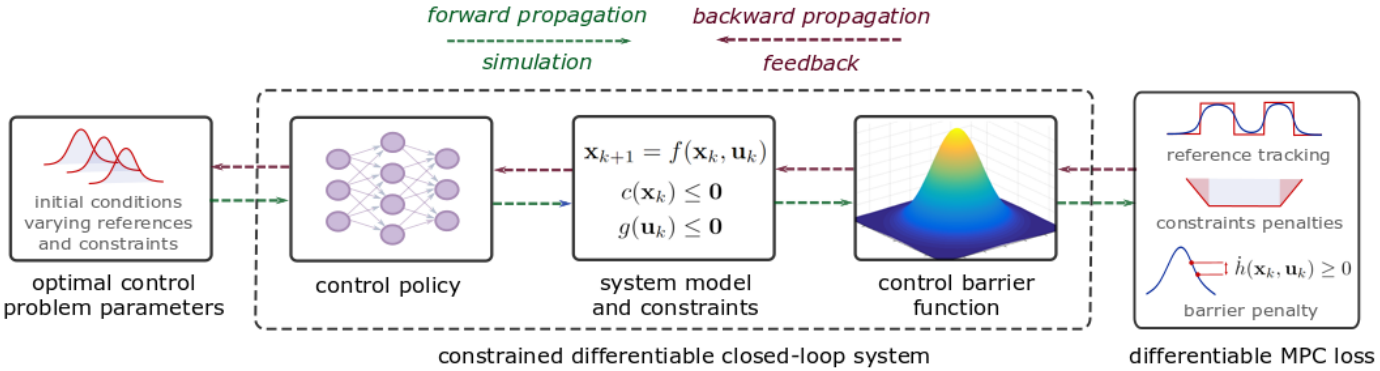
$$V(\mathbf{x}) = \sigma_{k+1}(\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{0})) + \epsilon \|\mathbf{x}\|_2^2,$$

Discrete-time Lyapunov penalty:

$$p_V(\mathbf{x}_{k+1}, \mathbf{x}_k) = ||\texttt{ReLU}(V_\phi(\mathbf{x}_{k+1}) - V_\phi(\mathbf{x}_\mathbf{k}))||_2$$

S. Mukherjee, J. Drgoňa, A. Tuor, M. Halappanavar, D. Vrabie, Neural Lyapunov Differentiable Predictive Control, CDC, 2022
Gaurav Manek, J. Zico Kolter, Learning Stable Deep Dynamics Models, NeurIPS 2019

DPC with Control Barrier Functions



Let's consider perturbed control-affine system:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u} + \boldsymbol{w}(t)$$

Define a safe set and its boundary:

$$\mathcal{C}(t) = \{\boldsymbol{x} \in \mathbb{R}^{n_x} : h(\boldsymbol{x}, t) \geq 0\}$$

$$\mathcal{A}(t) = \{\boldsymbol{x} \in \mathbb{R}^{n_x} : h(\boldsymbol{x}, t) \in [-b, a]\}$$

Control Barrier Function (CBF) condition:

$$\frac{d}{dt}h(x) = \nabla h(x)^\top \left(f(x) + g(x)u\right) \geq -\alpha(h(x))$$

Learned DPC policy with CBF safety filter:

$$u_k(x_k, k\Delta t) := \begin{cases} \pi_\theta(x_k, k\Delta t), & \text{if } x_k \notin \mathcal{A}(k\Delta t) \\ u_k^*(x_k, k\Delta t), & \text{if } x_k \in \mathcal{A}(k\Delta t) \end{cases}$$

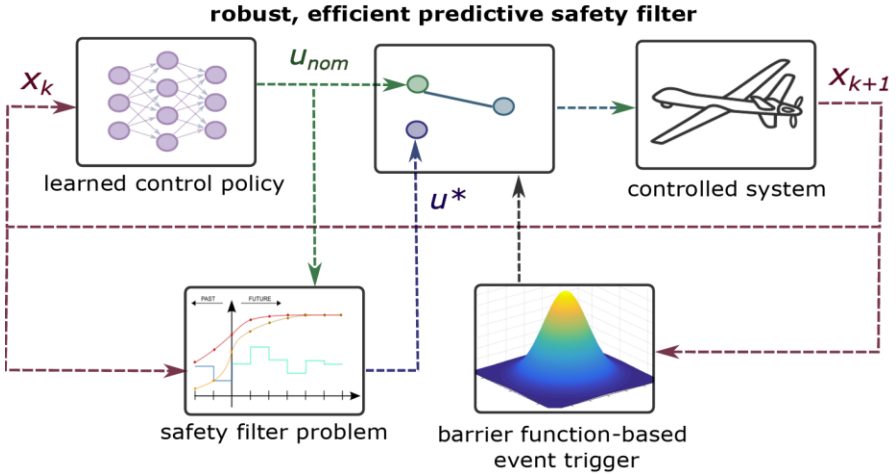
CBF safety filter implemented as QP:

$$u_k^*(x_k, k\Delta t) = \arg\min_{u \in \mathcal{U}} \|u - \pi_W(x_k, k\Delta t)\|_2^2$$

$$\text{subject to} \quad \nabla h(x)^\top \left(f(x) + g(x)u\right) \geq -\alpha(h(x))$$

W. S. Cortez, J. Drgoňa, A. Tuor, M. Halappanavar, D. Vrabie, Differentiable Predictive Control with Safety Guarantees: A Control Barrier Function Approach, CDC, 2022
A. D. Ames, et al., Control Barrier Functions: Theory and Applications, ECC, 2019

DPC with Predictive Safety Filters



Predictive safety filter (PSF) problem:

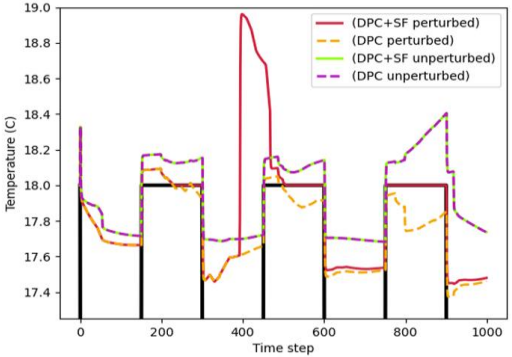
$$\begin{aligned} \min_{u_0,\ldots,u_{N-1}} \quad & \sum_{k=0}^{N-1} \|u_k - u_\theta\|^2 \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k), \\ & x_k \in \mathbb{X}, \quad u_k \in \mathbb{U}, \quad \forall k \in \mathbb{Z}_0^{N-1}, \\ & x_N \in \mathcal{X}_{\text{safe}}. \end{aligned}$$

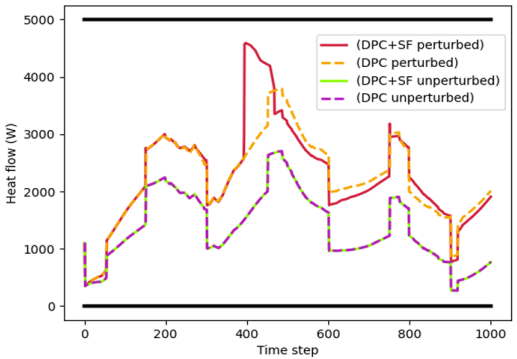
N-step robust barrier function event trigger:

$$\text{Trigger PSF at time } k \quad \text{if} \quad h(x_k) \leq \bar{h}$$

N-step ahead robust safety condition:

$$\delta\bar{h}_N(\boldsymbol{x}_k, \boldsymbol{u}_k, k) := h(\boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, k), k+1) - \mathcal{L}_h^x \mathcal{L}_d (\mathcal{L}_f^x)^{N-1} \geq 0$$

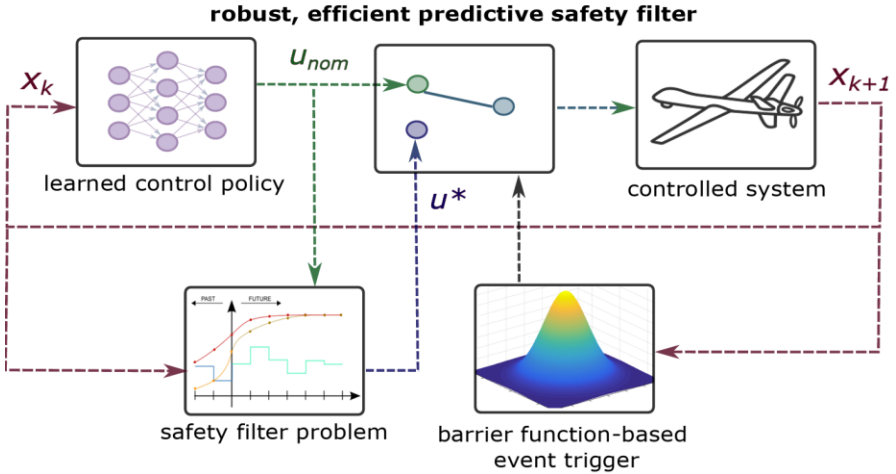




Wenceslao Shaw Cortez, Jan Drgona, Draguna Vrabie, Mahantesh Halappanavar, A Robust, Efficient Predictive Safety Filter, arXiv:2311.08496, 2024

K. P. Wabersich, M. N. Zeilinger, A predictive safety filter for learning-based control of constrained nonlinear dynamical systems, Automatica, 2021

DPC with Predictive Safety Filters

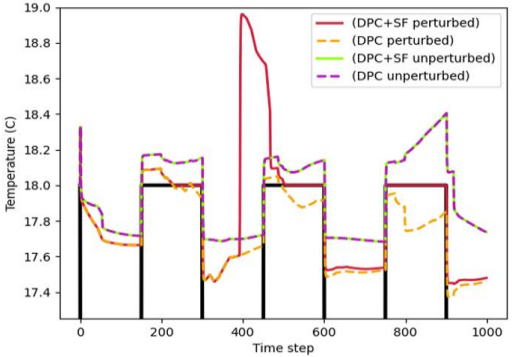


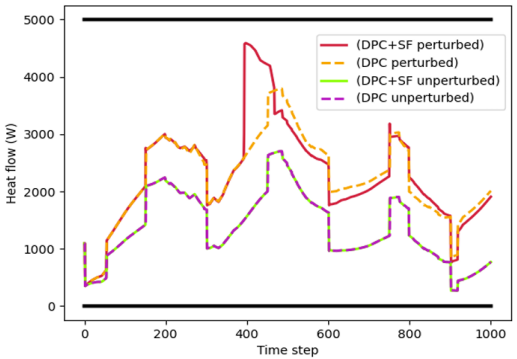
Predictive safety filter (PSF) problem:

$$\begin{aligned} \min_{u_0,\ldots,u_{N-1}} \quad & \sum_{k=0}^{N-1} \|u_k - u_\theta\|^2 \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k), \\ & x_k \in \mathbb{X}, \quad u_k \in \mathbb{U}, \quad \forall k \in \mathbb{Z}_0^{N-1}, \\ & x_N \in \mathcal{X}_{\text{safe}}. \end{aligned}$$

N-step robust barrier function event trigger:

$$\text{Trigger PSF at time } k \quad \text{if} \quad h(x_k) \leq \bar{h}$$

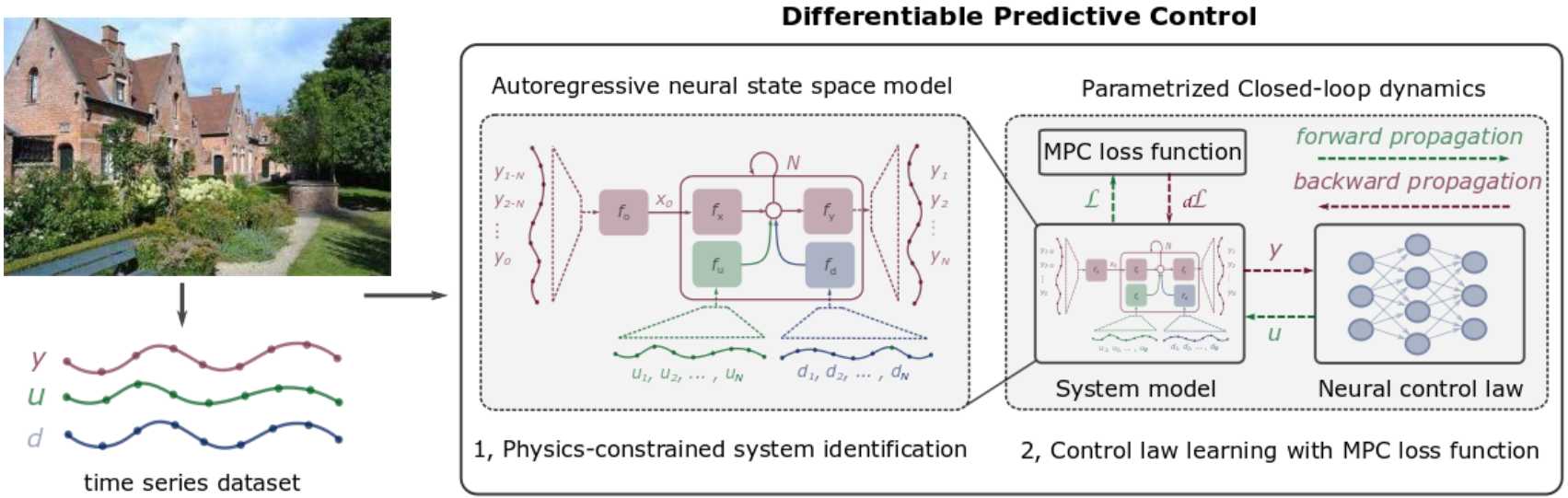




Just like RL or approximate MPC, DPC learns neural control policy and is compatible with most modern safety filter methods for learning-based control.

Wenceslao Shaw Cortez, Jan Drgona, Draguna Vrabie, Mahantesh Halappanavar, A Robust, Efficient Predictive Safety Filter, arXiv:2311.08496, 2024
K. P. Wabersich, M. N. Zeilinger, A predictive safety filter for learning-based control of constrained nonlinear dynamical systems, Automatica, 2021

Data-driven DPC for Building Energy System Optimization



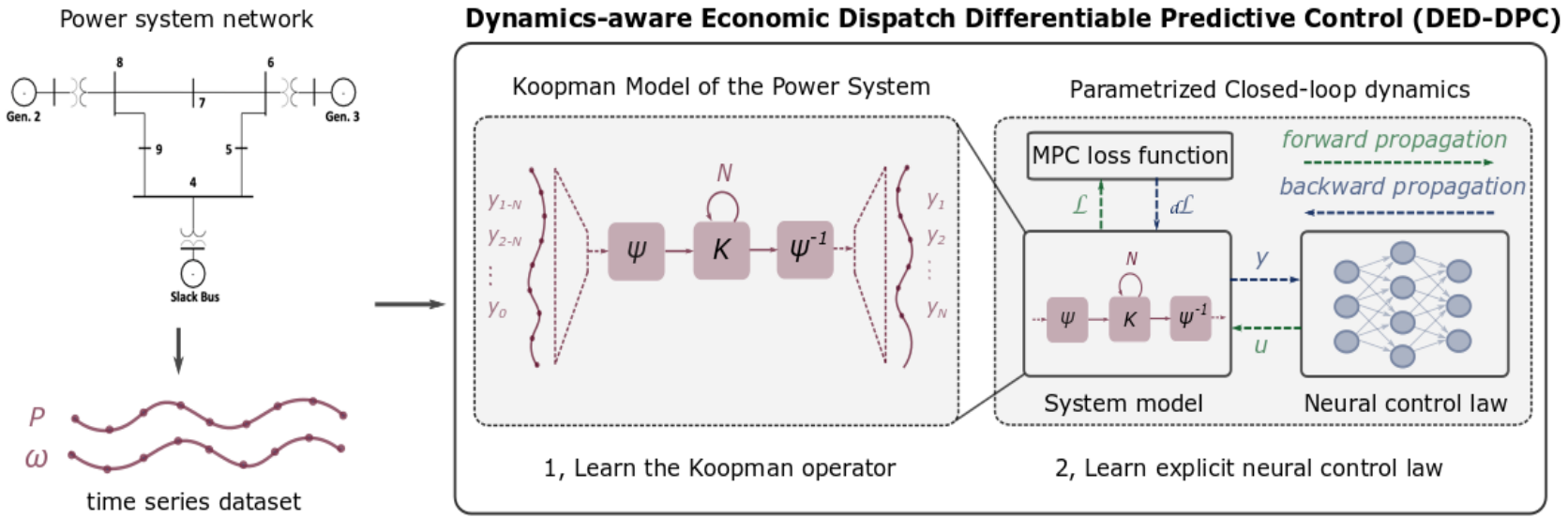
Benefits of DPC vs MPC

Modeling and optimal control design is roughly **10-times faster** and requires **less modeling expertise**.
Real-time decisions are made **orders of magnitude faster** than traditional model-based approaches.

J. Drgona, et al., Physics-constrained deep learning of multi-zone building thermal dynamics, Energy and Buildings, 2021

J. Drgona, et al., Deep Learning Explicit Differentiable Predictive Control Laws for Buildings, IFAC NMPC 2021

Data-driven DPC for Power System Optimization



Benefits of DPC vs MPC

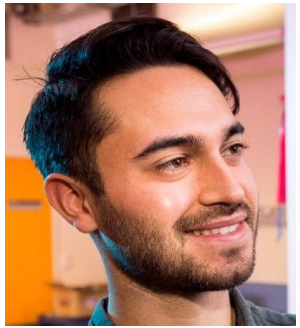
Fast prototyping by re-using code template from building control project.

Real-time decisions are made **orders of magnitude faster** than traditional model-based approaches.

Ethan King, et al., Koopman-based Differentiable Predictive Control for the Dynamics-Aware Economic Dispatch Problem, American Control Conference 2022

Acknowledgements

Aaron Tuor



Wenceslao Shaw Cortez

Ethan King

Sayak Mukherjee

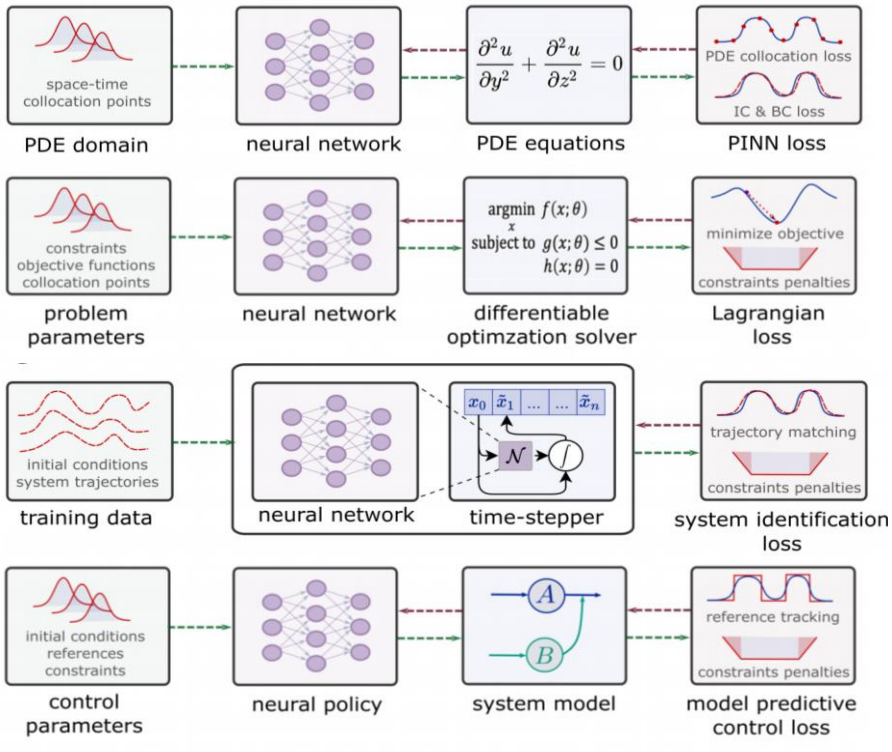
Mahantesh Halappanavar

Draguna Vrabie

U.S. DEPARTMENT OF ENERGY

Pacific Northwest NATIONAL LABORATORY

Try DPC Yourself in Neuromancer!



Open-source library in PyTorch

- Physics-informed Neural Networks
- Learning to optimize
- Neural differential equations
- Differentiable predictive control

github.com/pnnl/neuromancer

Summary

- **Differentiable Predictive Control**
  - MBRL algorithm inspired by explicit MPC
  - Differentiable system models and rewards
  - Avoids learning the critic
  - Direct policy optimization
  - Compatible with modern safety filters
- **Ongoing Research**
  - UQ and Robust DPC
  - Offset-free and online adaptive DPC
  - Dealing with mixed-integer decisions
  - Dealing with vanishing and exploding gradients
  - Control of PDEs
  - Applications in large-scale energy systems

Energy at Hopkins

drgona.github.io jdrgona1@jh.edu github.com/pnnl/neuromancer