Differentiable Planners

Saurabh Gupta

Solving a RL Problem

Structured Policies

Better Reward Signals

Sim2Real

Convert into a Supervised Training Problem

Better Optimization

Solve a Related but Supervision-rich Problem

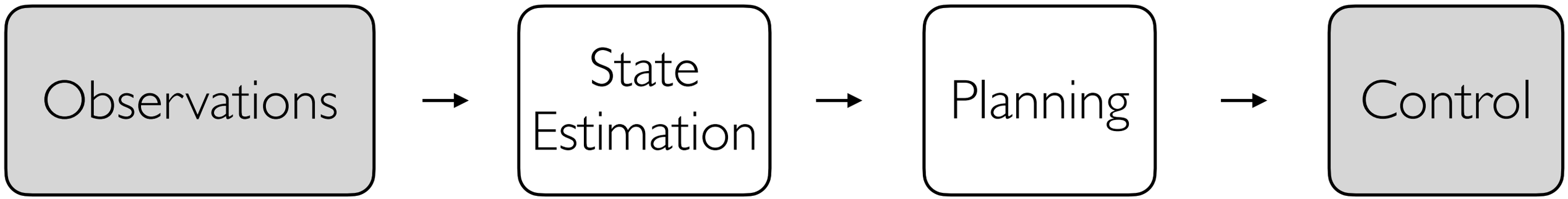
Build Models and Plan with Them

Model-free RL with sparse rewards

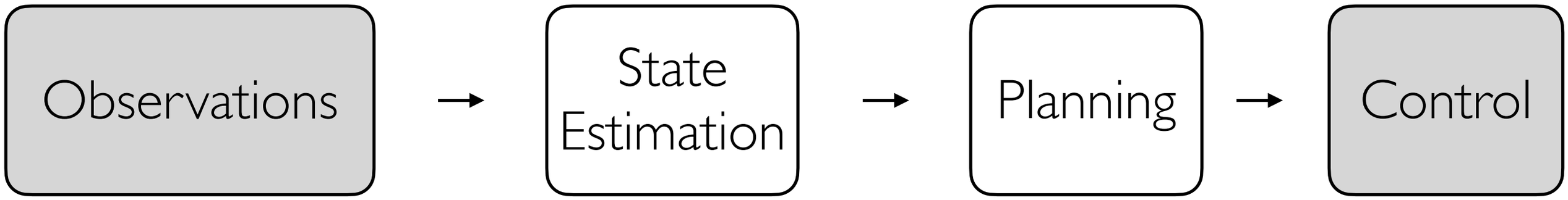
Known reward, known model. Model-based RL

Structured Policies

Borrow policy structures from classical pipelines, and inject learning.

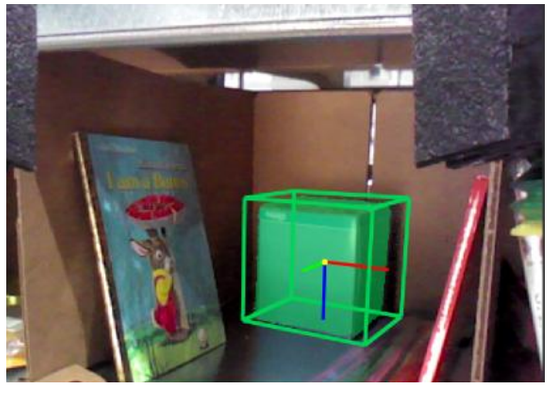


Typical Robotics Pipeline

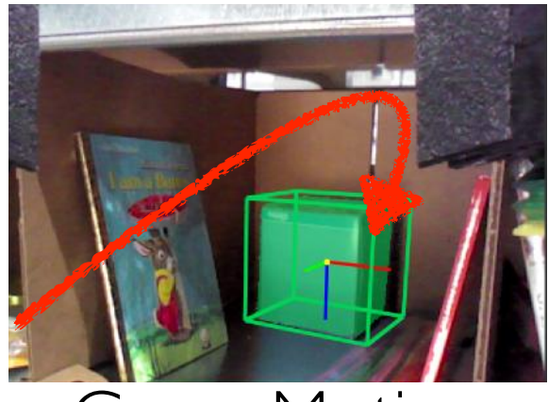


Manipulation

Observed Images

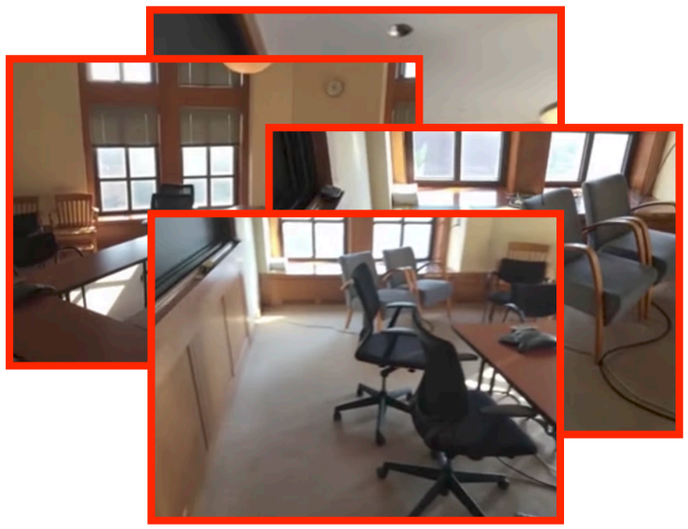


6DOF Pose



Grasp Motion Planning

Observations → State Estimation → Planning → Control

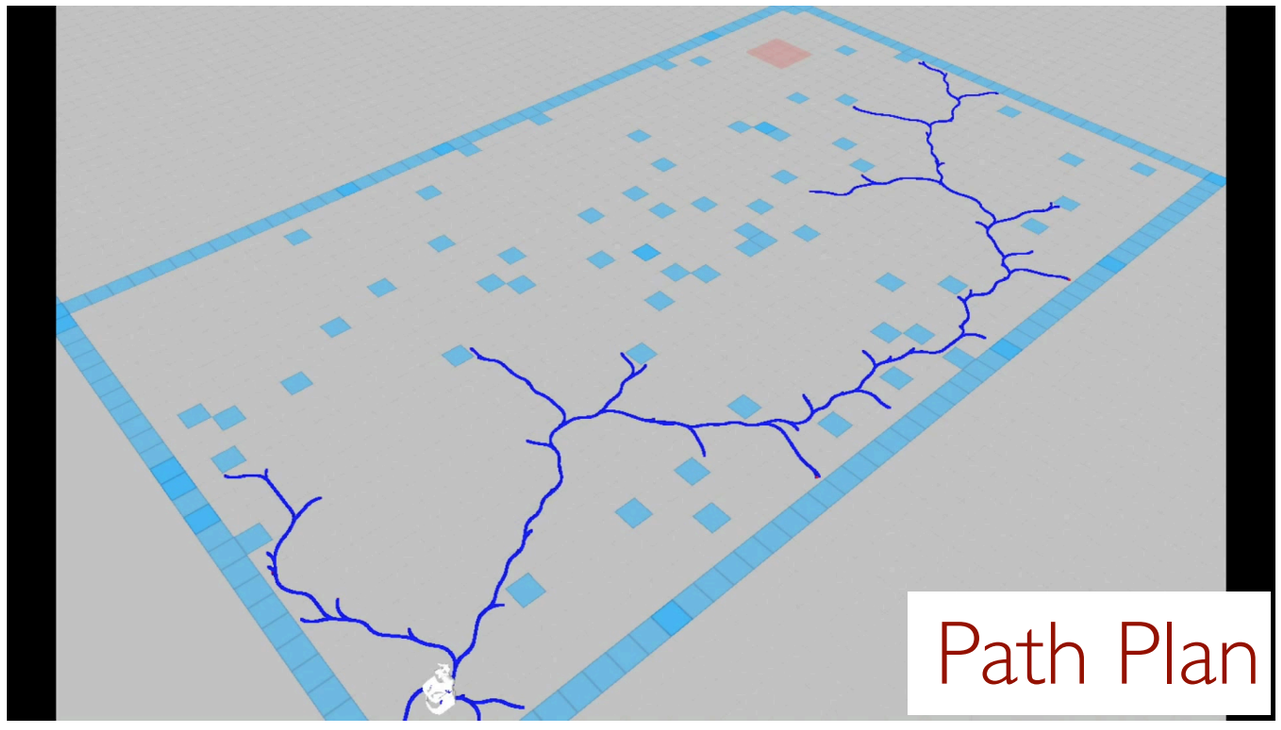


Observed Images

Mapping

Geometric Reconstruction

Planning



Path Plan

Hartley and Zisserman. 2000. Multiple View Geometry in Computer Vision

Thrun, Burgard, Fox. 2005. Probabilistic Robotics

Canny. 1988. The complexity of robot motion planning.

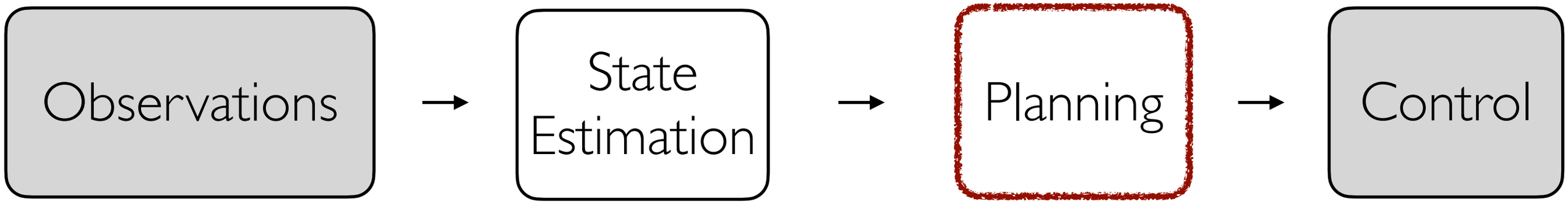
Kavraki et al. RA1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces.

Lavalle and Kuffner. 2000. Rapidly-exploring random trees: Progress and prospects.

Video Credits: Mur-Artal et al., Palmieri et al.

Structured Policies

Borrow policy structures from classical pipelines, and inject learning.

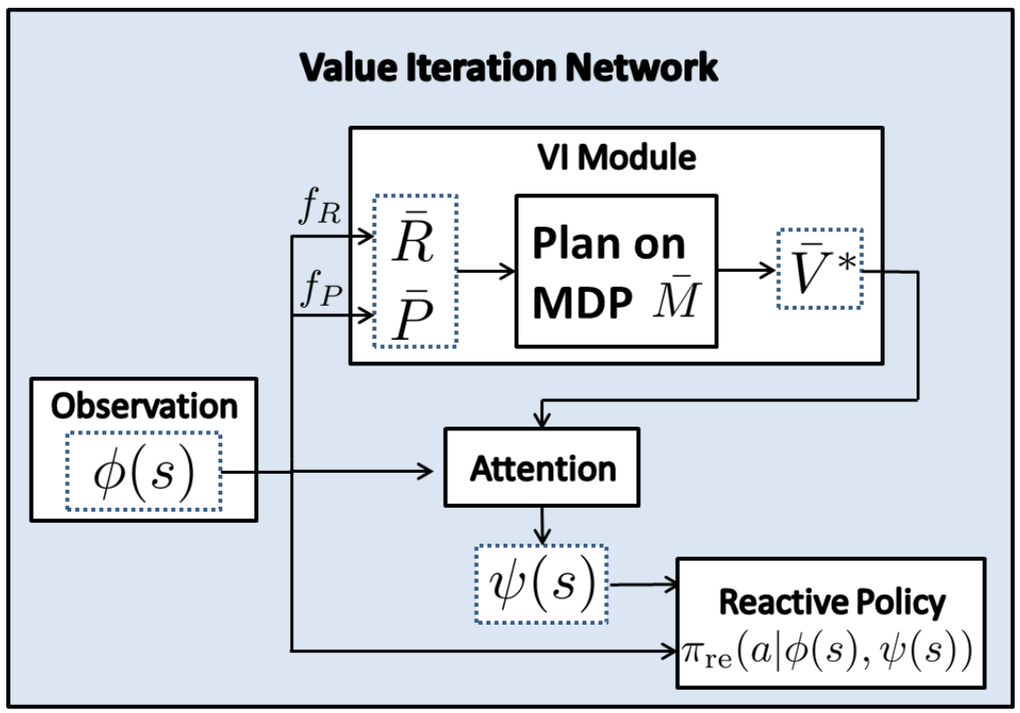


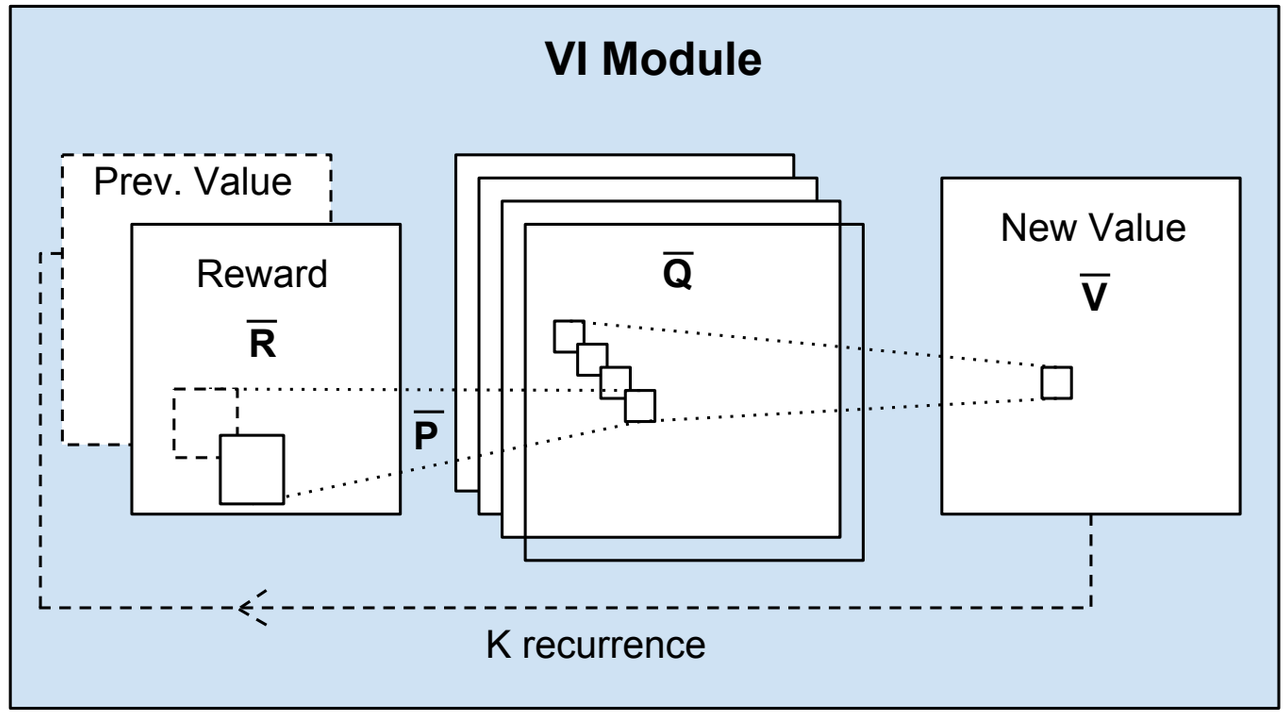
If we can differentiate through the path planning step, we can train state estimators, directly for outputting good control.

- Analytically differentiate through a planner
  - eg: Differentiating through value iteration
  - eg: Differentiating through MPC (this paper)
- Train a planner (and differentiate through it)
  - eg: Value iteration networks
  - eg: Spatial Planning using Transformers

Value iteration

Value iteration networks

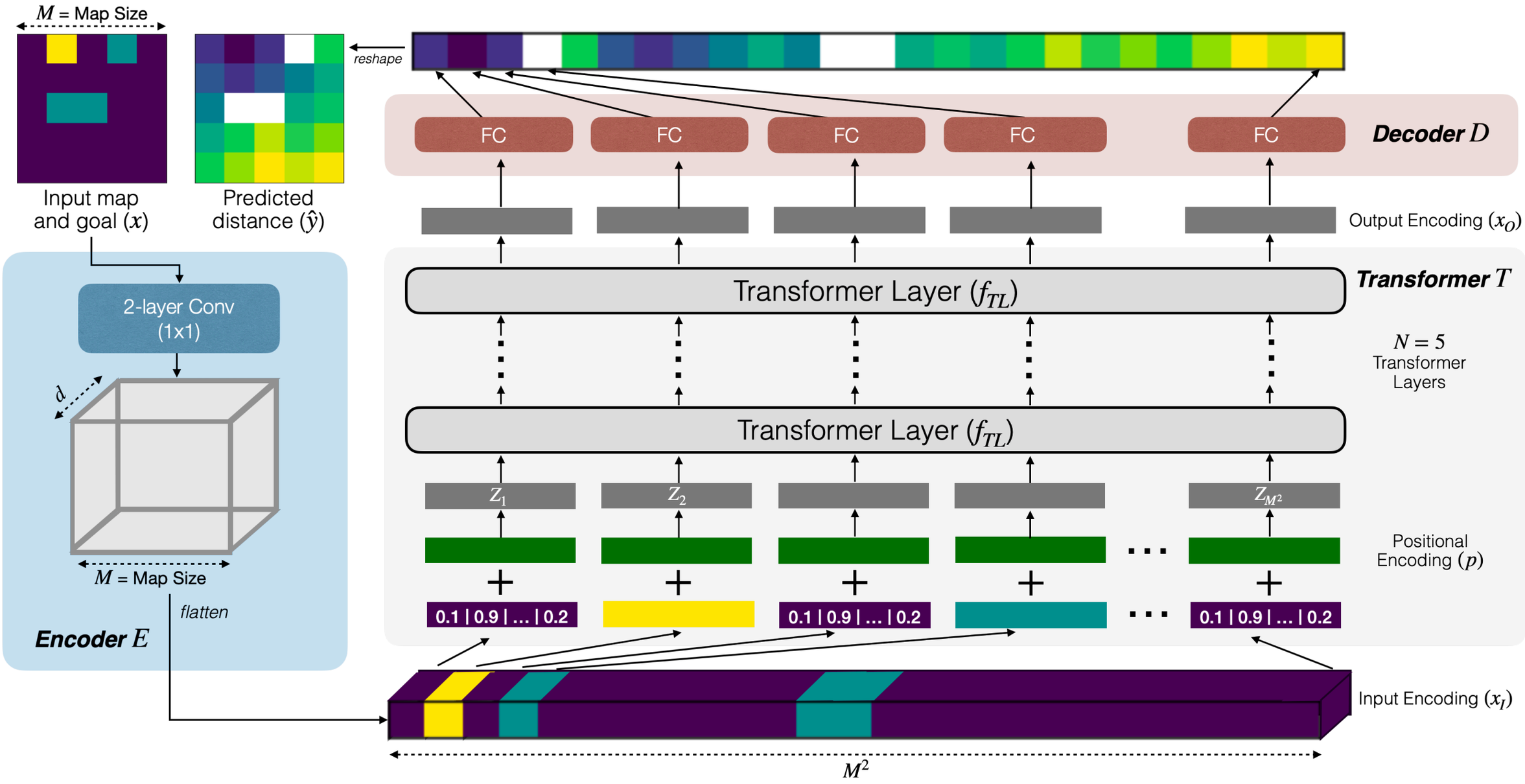




express value iteration computation as a convolutional network (with max-pooling across channels)

Tamar et al. NeurIPS 2016. Value Iteration Networks

Spatial Planning using Transformers



long-range value prediction via transformers

Open Review 2020. Differentiable Spatial Planning using Transformers link

Differentiable MPC for End-to-end Planning and Control

Brandon Amos[1] **Ivan Dario Jimenez Rodriguez**[2] **Jacob Sacks**[2]
Byron Boots[2] **J. Zico Kolter**[13]
[1]Carnegie Mellon University [2]Georgia Tech [3]Bosch Center for AI

- $$\tau^\star_{1:T} = \underset{\tau_{1:T}}{\operatorname{argmin}} \sum_{t=1}^{T} \frac{1}{2} \tau_t^\top C_t \tau_t + c_t^\top \tau_t \quad \text{subject to} \quad x_1 = x_{\text{init}},\ x_{t+1} = F_t \tau_t + f_t.$$
- $\theta = \{C, c, F, f\}$, $\tau^\star_{1:T} = \{x_t, u_t\}_{1:T}$
- Paper is about, how to compute gradient: $$\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial \tau^\star_{1:T}} \frac{\partial \tau^\star_{1:T}}{\partial \theta}.$$

Builds upon OptNet

$$z_{i+1} = \underset{z}{\text{argmin}} \ \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z$$
$$\text{subject to} \ A(z_i) z = b(z_i)$$
$$G(z_i) z \leq h(z_i)$$

-
- Optimization layer:
  - inputs $z_i$describe a constrained optimization problem
  - output z_{i+1} is the minimizer of the optimization problem

Amos and Kolter ICML 2017. OptNet: Differentiable Optimization as a Layer in Neural Networks

Builds upon OptNet

$$z_{i+1} = \underset{z}{\text{argmin}} \ \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z$$
$$\text{subject to} \ A(z_i) z = b(z_i)$$
$$G(z_i) z \leq h(z_i)$$

-
- Compute Lagrangian, with Lagrange multipliers, $\lambda \geq 0$, ν:

$$L(z, \nu, \lambda) = \frac{1}{2} z^T Q z + q^T z + \nu^T (Az - b) + \lambda^T (Gz - h)$$

- For a convex optimization problem, at optima, KKT conditions, complementary slackness must hold:

$$Qz^\star + q + A^T \nu^\star + G^T \lambda^\star = 0$$
$$Az^\star - b = 0$$
$$D(\lambda^\star)(Gz^\star - h) = 0,$$

-

See, CVX Book Chapter 5. Stephen Boyd.

Builds upon OptNet

- Get gradients via implicit differentiation:

$$\mathrm{d}Qz^\star + Q\mathrm{d}z + \mathrm{d}q + \mathrm{d}A^T\nu^\star +$$
$$A^T\mathrm{d}\nu + \mathrm{d}G^T\lambda^\star + G^T\mathrm{d}\lambda = 0$$
$$\mathrm{d}Az^\star + A\mathrm{d}z - \mathrm{d}b = 0$$
$$D(Gz^\star - h)\mathrm{d}\lambda + D(\lambda^\star)(\mathrm{d}Gz^\star + G\mathrm{d}z - \mathrm{d}h) = 0$$

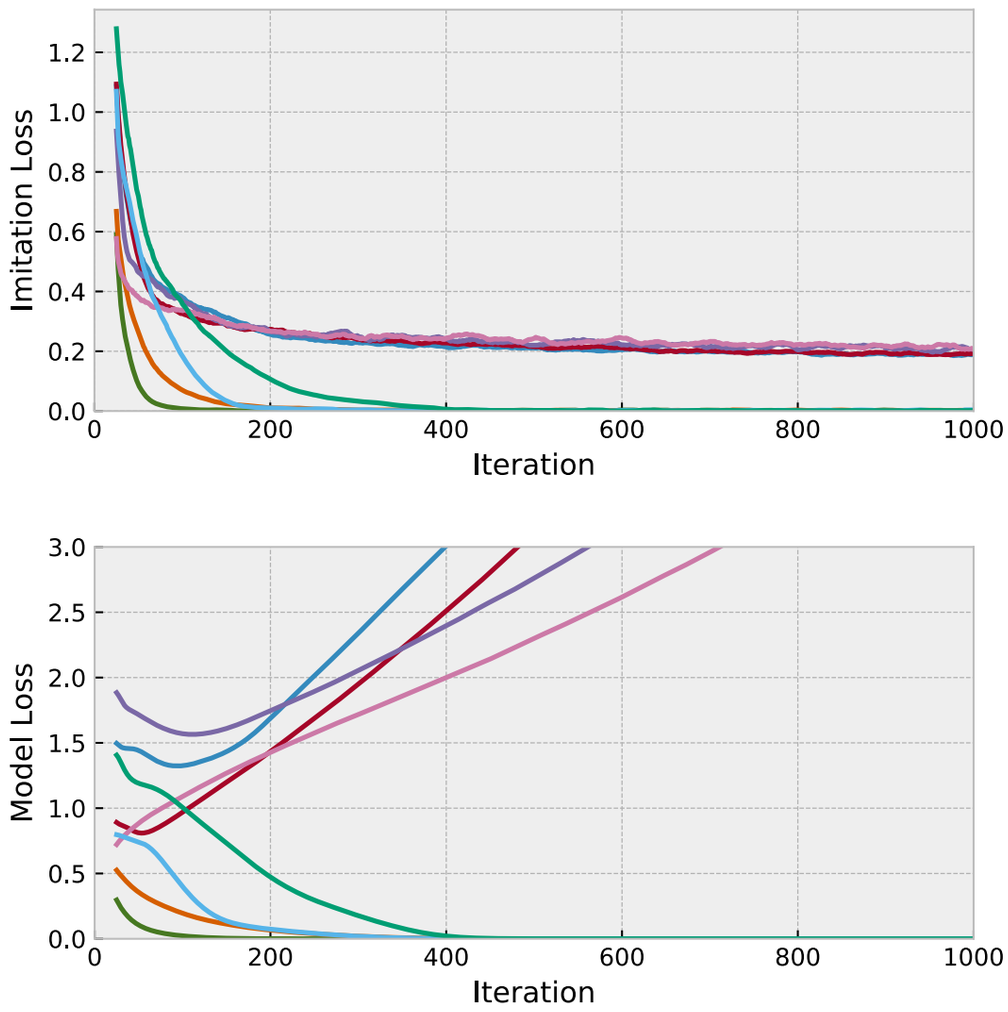
- Rearrange:

$$\begin{bmatrix} Q & G^T & A^T \\ D(\lambda^\star)G & D(Gz^\star - h) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{d}z \\ \mathrm{d}\lambda \\ \mathrm{d}\nu \end{bmatrix} =$$
$$- \begin{bmatrix} \mathrm{d}Qz^\star + \mathrm{d}q + \mathrm{d}G^T\lambda^\star + \mathrm{d}A^T\nu^\star \\ D(\lambda^\star)\mathrm{d}Gz^\star - D(\lambda^\star)\mathrm{d}h \\ \mathrm{d}Az^\star - \mathrm{d}b \end{bmatrix}.$$

$$\begin{bmatrix} Q & G^T & A^T \\ D(\lambda^\star)G & D(Gz^\star - h) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathsf{d}z \\ \mathsf{d}\lambda \\ \mathsf{d}\nu \end{bmatrix} =$$
$$- \begin{bmatrix} \mathsf{d}Qz^\star + \mathsf{d}q + \mathsf{d}G^T\lambda^\star + \mathsf{d}A^T\nu^\star \\ D(\lambda^\star)\mathsf{d}Gz^\star - D(\lambda^\star)\mathsf{d}h \\ \mathsf{d}Az^\star - \mathsf{d}b \end{bmatrix}.$$

- Rearrange:
- Can compute partial derivatives: $\partial z / \partial b$, by solving system of equations
- Better still, can directly compute $\dfrac{\partial l}{\partial b} = \dfrac{\partial l}{\partial z}\dfrac{\partial z}{\partial b}$
- $$\begin{bmatrix} d_z \\ d_\lambda \\ d_\nu \end{bmatrix} = - \begin{bmatrix} Q & G^T D(\lambda^\star) & A^T \\ G & D(Gz^\star - h) & 0 \\ A & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \left(\frac{\partial \ell}{\partial z^\star}\right)^T \\ 0 \\ 0 \end{bmatrix}$$
- $$\begin{aligned} \nabla_Q \ell &= \frac{1}{2}(d_z z^T + z d_z^T) & \nabla_q \ell &= d_z \\ \nabla_A \ell &= d_\nu z^T + \nu d_z^T & \nabla_b \ell &= -d_\nu \\ \nabla_G \ell &= D(\lambda^\star)(d_\lambda z^T + \lambda d_z^T) & \nabla_h \ell &= -D(\lambda^\star) d_\lambda \end{aligned}$$

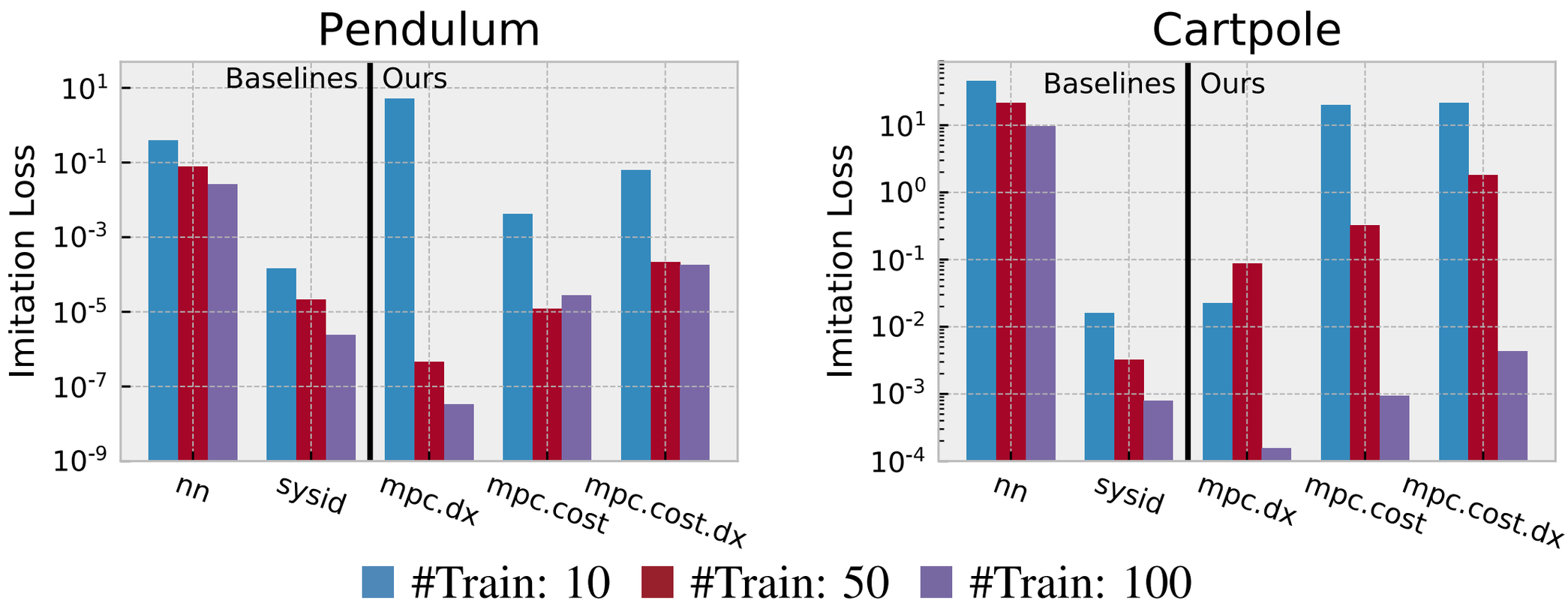
Experiments: Learning LQR Models



- Learning dynamics function ($\boldsymbol{A}$ and $\boldsymbol{B}$ matrices) for LQR.

Amos and Kolter ICML 2017. OptNet: Differentiable Optimization as a Layer in Neural Networks

Experiments: Learning in non-convex problems



- nn: LSTM to predict next action
- sysid: known cost, estimates parameters for 1 step dynamics functions
- mpc.dx: known cost, estimates parameters for 1 step dynamics functions
- mpc.cost: known dynamics, estimates cost parameters
- mpc.cost.dx: estimates both cost and dynamics function parameters

Experiments: SysID with a non-realizable expert

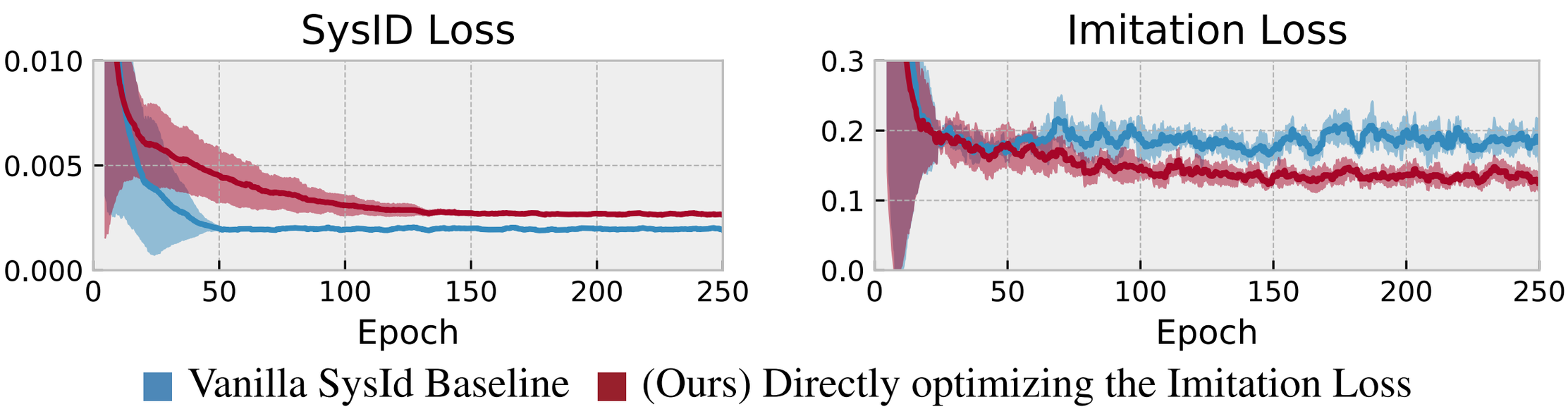


Figure 5: Convergence results in the non-realizable Pendulum task.

- sysid: known cost, estimates parameters for 1 step dynamics functions
- mpc.dx: known cost, estimates parameters for 1 step dynamics functions
- SysID does better at estimating parameters, but worse at imitating

Thank you