Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Learning Local models, TRPO, Imitating Optimal Controllers

Katerina Fragkiadaki

Last lecture

Iterative-Linear Quadratic Regulator for continuous control: We assumed:

- known dynamics model
- we could measure the reward (state x was fully observed, thus also the distance from a desired state x*)

and we showed a local optimization process that would achieve the desired task from a specific initial state x_0 using iterative linear approximations of dynamics and quadratic approximations for the cost.

Learning global dynamics models using Neural Networks as the function class

This lecture

- Learning local dynamics models
- i-LQR with learn local models
- Trust region constraint for policy optimization: TRPO and i-LQR
- Learning *general* policies by imitating i-LQR local controllers
  - DAGGER
  - Guided policy search

Next lecture

- Differentiable model-based reinforcement learning
- Recurrent networks and optimal control
- Back-propagate directly to the policy using temporal unfolding- differentiable dynamics- back propagate through discrete actions (stochastic sampling on the forward pass), or through continuous actions (re-paramertization trick)

(Locally) Optimal Control

$$\min_{u_1,\ldots,u_T} \sum_{t=1}^{T} c(x_t, u_t) \text{ s.t. } x_t = f(x_{t-1}, u_{t-1})$$

$$\min_{u_1,\ldots,u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \cdots + c(f(f(...)...), u_T)$$

Differentiate and optimize.

Need derivates: $\frac{df}{dx_t}, \frac{df}{du_t}, \frac{dc}{dx_t}, \frac{dc}{du_t}$

In case f is linear and c quadratic, then we can using dynamic programming and get optimal solution!-> i-LQR, MPC extensions

If we knew the dynamics

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$

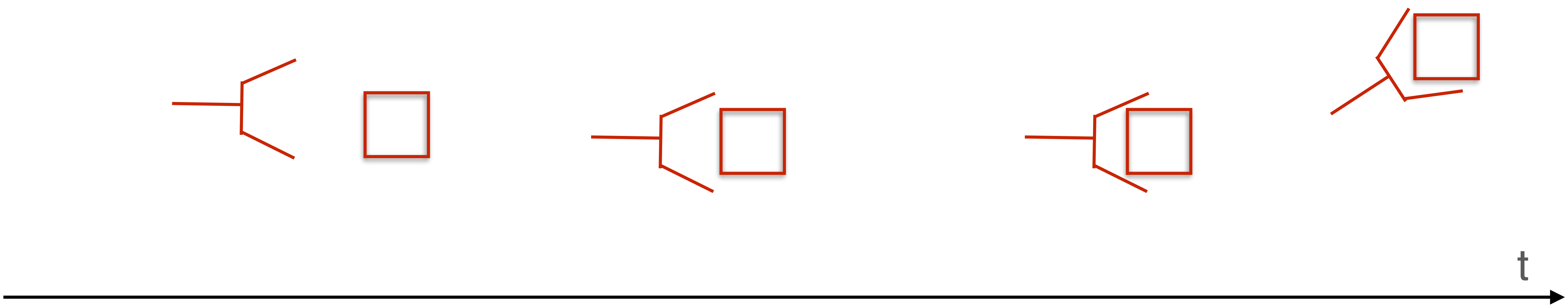
Global dynamics model would do. But we saw they are hard to fit/get them to generalize.

But if you use i-LQR, in any case it is a local optimization method, around reference trajectories! You don't need dynamics everywhere (at each iteration), only around the reference trajectory: $\hat{x}_t, \hat{u}_t$!

(Time varying) Local models of dynamics! Local linear approximations!

Time varying linear dynamics

reference trajectory $\hat{x}_t, \hat{u}_t, t = 1, ..., T$



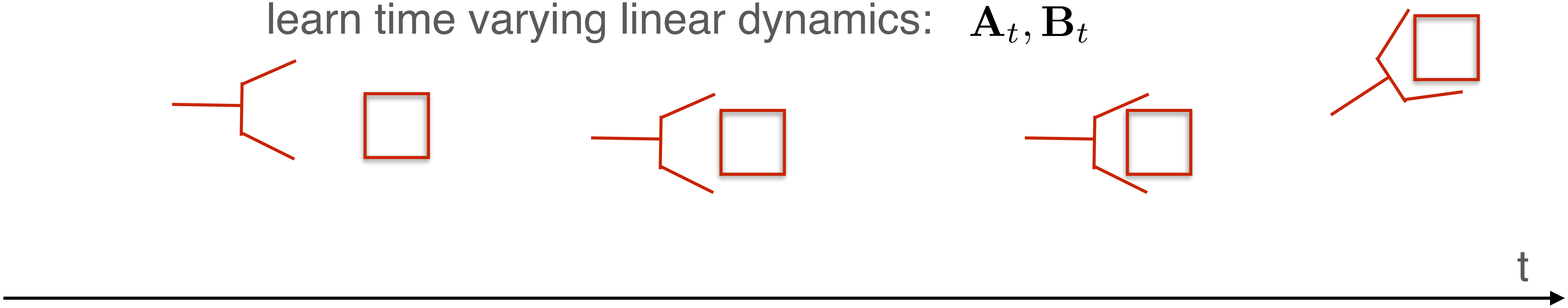
Time varying linear dynamics

$$f(x_t, u_t) \approx \mathbf{A}_t x_t + \mathbf{B}_t u_t$$

$$\mathbf{A}_t = \frac{df}{dx_t} \quad \mathbf{B}_t = \frac{df}{du_t}$$

reference trajectory $\hat{x}_t, \hat{u}_t, t = 1, ..., T$

learn time varying linear dynamics: $\mathbf{A}_t, \mathbf{B}_t$



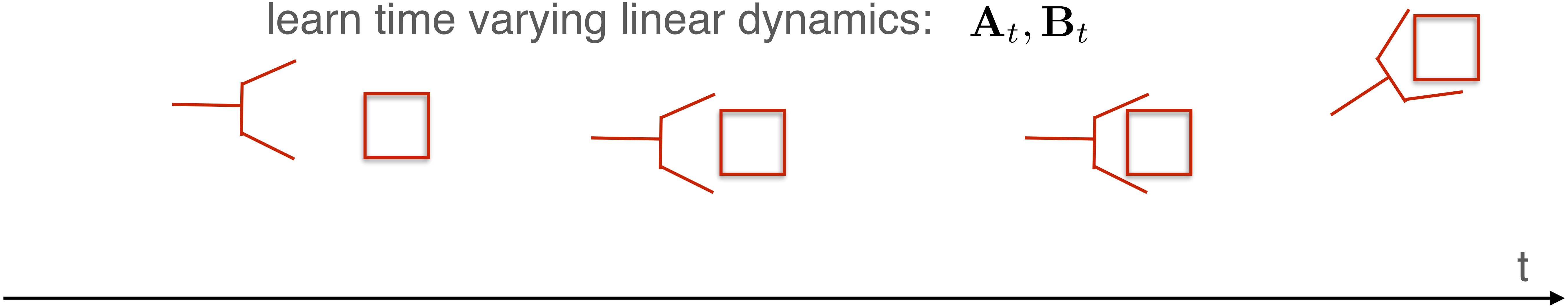
Time varying linear dynamics

$$f(x_t, u_t) \approx \mathbf{A}_t x_t + \mathbf{B}_t u_t$$

$$\mathbf{A}_t = \frac{df}{dx_t} \quad \mathbf{B}_t = \frac{df}{du_t}$$

reference trajectory $\hat{x}_t, \hat{u}_t, t = 1, ..., T$

learn time varying linear dynamics: $\mathbf{A}_t, \mathbf{B}_t$



How do I get the data to fit my linear dynamics at each time step?

We execute the controller u_t at state x_t to explore how the world works in the vicinity of the reference trajectory!

Which controller?

Which controller to collect samples with?

- We need a stochastic controller! Why?

Which controller to collect samples with?

- We need a stochastic controller! Why?
- Here is a good guess: add some noise to the output of iLQR:

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

Which controller to collect samples with?

- We need a stochastic controller! Why?
- Here is a good guess: add some noise to the output of iLQR:

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

- It turns out that setting $\Sigma_t = \mathbf{Q}^{-1}_{\mathbf{u}_t,\mathbf{u}_t}$ solves the following maximum entropy control problem:

$$\min \sum_{t=1}^{T} E_{(\mathbf{x}_t,\mathbf{u}_t)\sim p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t, \mathbf{u}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))]$$

Guided Policy Search, Levine and Colton 2013

Which controller to collect samples with?

- We need a stochastic controller! Why?
- Here is a good guess: add some noise to the output of iLQR:

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

- It turns out that setting $\Sigma_t = \mathbf{Q}^{-1}_{\mathbf{u}_t,\mathbf{u}_t}$ solves the following maximum entropy control problem:

$$\min \sum_{t=1}^{T} E_{(\mathbf{x}_t,\mathbf{u}_t)\sim p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t, \mathbf{u}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))]$$

- Remember, cost to go:

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2}\begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

- The above controller strikes the right balance between minimizing the cost and maximize exploration

Guided Policy Search, Levine and Colton 2013

Which controller to collect samples with?

$$\min \sum_{t=1}^{T} E_{(\mathbf{x}_t,\mathbf{u}_t)\sim p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))]$$

Guided Policy Search, Levine and Colton 2013

- Act as randomly as possible while minimizing the cost! What does this remind us of?

Which controller to collect samples with?

$$\min \sum_{t=1}^{T} E_{(\mathbf{x}_t,\mathbf{u}_t)\sim p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))]$$

Guided Policy Search, Levine and Colton 2013

- Act as randomly as possible while minimizing the cost! What does this remind us of?

- MaxEntIOC: be as random as possible while matching the feature counts of demonstrated paths

$$\max_P - \sum_\tau P(\tau) \log P(\tau)$$

$$\sum_\tau P(\tau) f_\tau = f_{\text{dem}}$$

Time varying linear dynamics

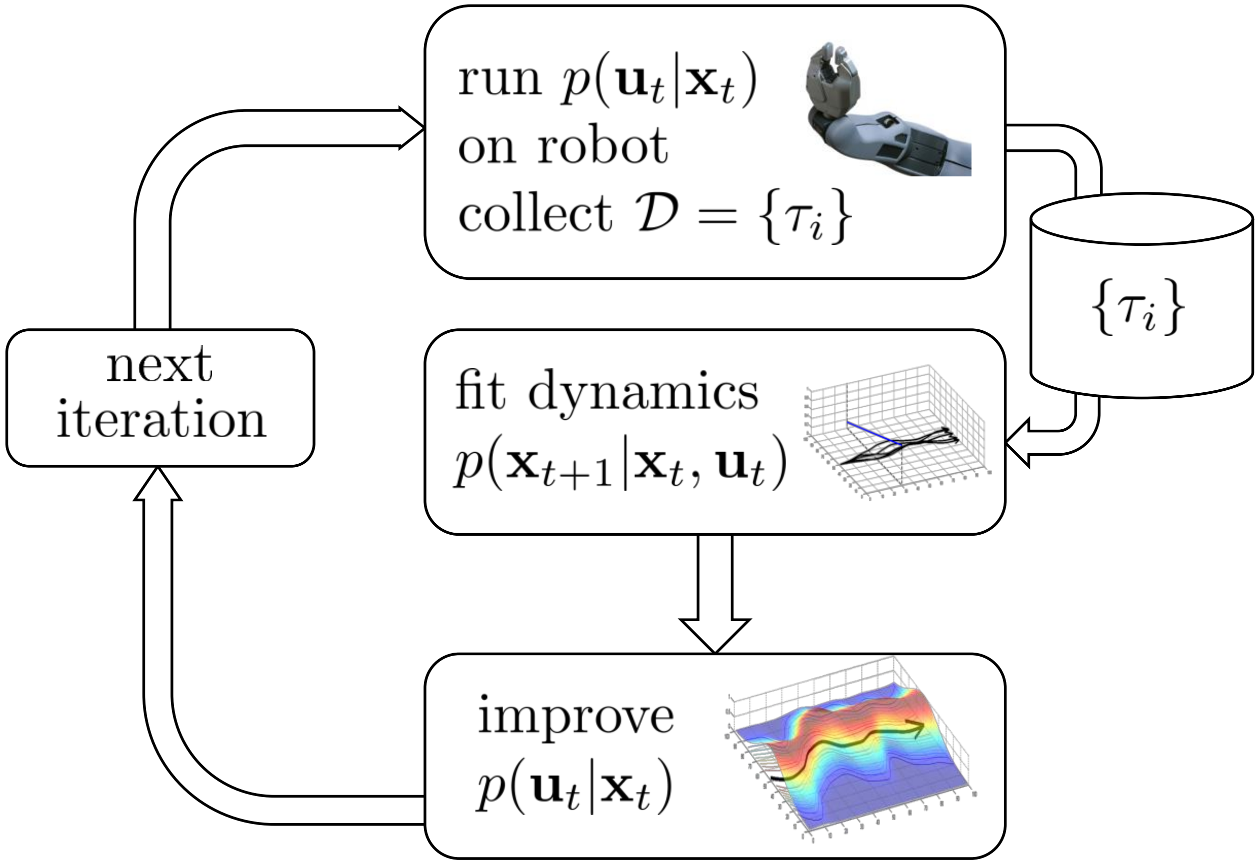
We iteratively fit dynamics and update the policy. Why such iteration is important?

So that the space (state, action distribution) our dynamics are estimated is similar to the one our policy visits (last lecture).

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t), \Sigma)$$

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx \mathbf{A}_t \mathbf{x}_t + \mathbf{B}_t \mathbf{u}_t$$

$$\mathbf{A}_t = \frac{df}{d\mathbf{x}_t} \qquad \mathbf{B}_t = \frac{df}{d\mathbf{u}_t}$$



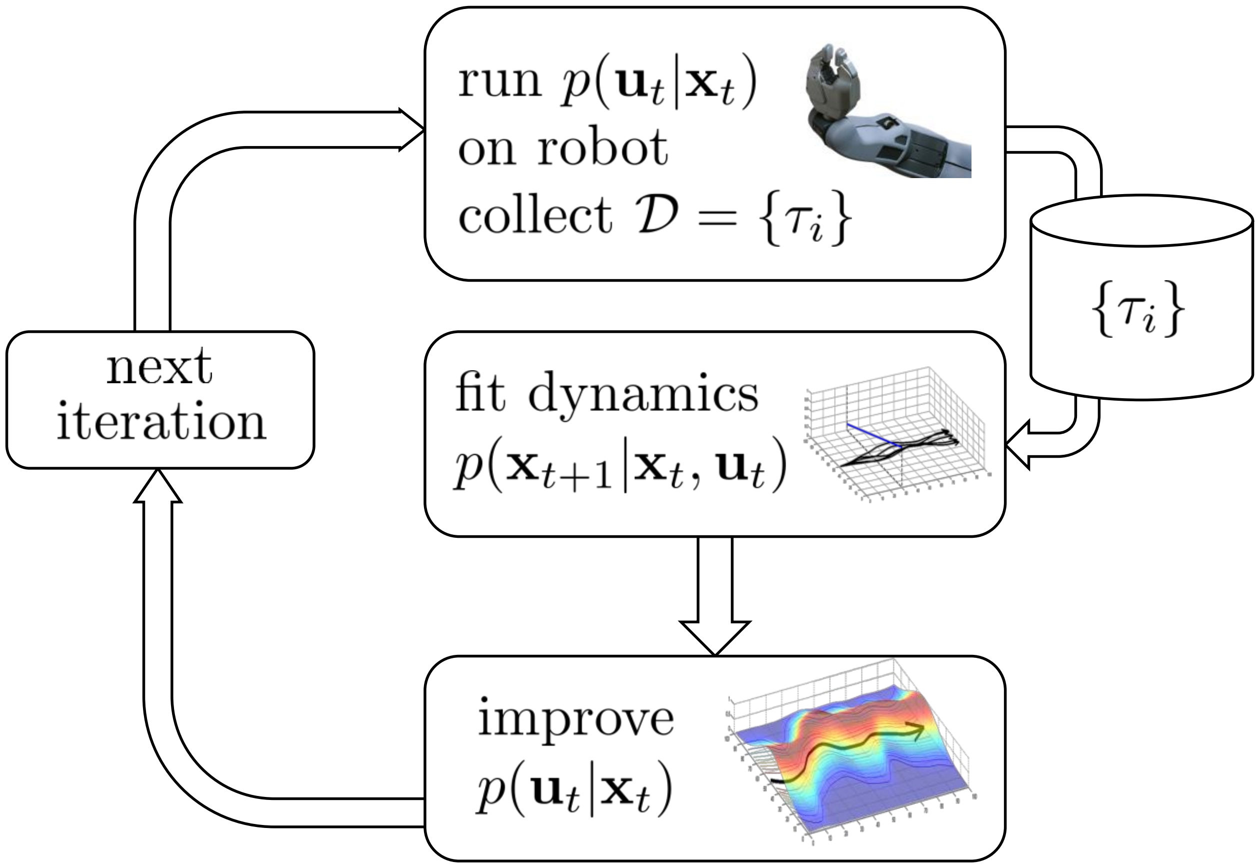
Fitting time varying linear dynamics

- Can we further improve sample complexity? Right now each sample (x_t, u_t, x_{t+1}) contributes in one linear model fitting.
- Instead of linear regression use Bayesian linear regression!

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t), \Sigma)$$

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx \mathbf{A}_t \mathbf{x}_t + \mathbf{B}_t \mathbf{u}_t$$

$$\mathbf{A}_t = \frac{df}{d\mathbf{x}_t} \qquad \mathbf{B}_t = \frac{df}{d\mathbf{u}_t}$$



Bayesian Linear regression

Let β be the weights of our linear regression model:

$$y = X\beta + \epsilon. \quad \epsilon_i \sim \mathrm{N}(0, \sigma^2).$$

$$p(y \mid X, \beta; \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}\|y - X\beta\|^2\right)$$

Bayesian Linear regression

Let β be the weights of our linear regression model:

$$y = X\beta + \epsilon. \quad \epsilon_i \sim \mathrm{N}(0, \sigma^2).$$

$$p(y \mid X, \beta; \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}\|y - X\beta\|^2\right)$$

By maximizing the log likelihood we get the MLE solution for the weights:

$$\hat{\beta} = (X^T X)^{-1} X^T y \qquad \hat{\beta} \sim \mathrm{N}\left(\beta, \sigma^2 (X^T X)^{-1}\right)$$

Bayesian Linear regression

Let β be the weights of our linear regression model:

$$y = X\beta + \epsilon. \quad \epsilon_i \sim \mathrm{N}(0, \sigma^2).$$

$$p(y \mid X, \beta; \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}\|y - X\beta\|^2\right)$$

By maximizing the log likelihood we get the MLE solution for the weights:

$$\hat{\beta} = (X^T X)^{-1} X^T y \qquad \hat{\beta} \sim \mathrm{N}\left(\beta, \sigma^2 (X^T X)^{-1}\right)$$

What if we assume the following prior for the weights:

$$\beta \sim \mathrm{N}(0, \Lambda^{-1})$$

Bayesian Linear regression

Let β be the weights of our linear regression model:

$$y = X\beta + \epsilon. \quad \epsilon_i \sim \mathrm{N}(0, \sigma^2).$$

$$p(y \mid X, \beta; \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}\|y - X\beta\|^2\right)$$

By maximizing the log likelihood we get the MLE solution for the weights:

$$\hat{\beta} = (X^T X)^{-1} X^T y \qquad \hat{\beta} \sim \mathrm{N}\left(\beta, \sigma^2 (X^T X)^{-1}\right)$$

What if we assume the following prior for the weights:

$$\beta \sim \mathrm{N}(0, \Lambda^{-1})$$

Then the posterior will be:

$$P(\beta | y, X; \sigma^2) \propto P(y | \beta; \sigma^2) P(\beta)$$

$$\beta \sim \mathrm{N}(\mu_n, \Sigma_n),$$

$$p(\beta \mid y; X, \sigma^2) \propto \exp\left(-\frac{1}{2\sigma^2}\|y - X\beta\|^2 - \frac{1}{2}\beta^T \Lambda \beta\right)$$

$$\mu_n = \left(X^T X + \sigma^2 \Lambda\right)^{-1} X^T y,$$
$$\Sigma_n = \sigma^2 \left(X^T X + \sigma^2 \Lambda\right)^{-1}.$$

http://www.cedar.buffalo.edu/~srihari/CSE574/Chap3/BayesianRegression.pdf

https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf

Bayesian Linear dynamics fitting

Fit a *Global* Gaussian Mixture Model using all samples (x_t, u_t, x_{t+1}) of all iterations and time steps. -> prior

Use current samples (from this iteration) and obtain Gaussian posterior for (x_t, u_t, x_{t+1}), which you condition to obtain $p(x_{t+1}|x_t, u_t)$.

Such prior results in 4 to 8 times less samples needed, despite the fact that it is not accurate enough by itself.

$$\Sigma = \frac{\mathbf{\Phi} + N\hat{\Sigma} + \frac{Nm}{N+m}(\hat{\mu} - \mu_0)(\hat{\mu} - \mu_0)^{\mathrm{T}}}{N + n_0}$$

$$\mu = \frac{m\mu_0 + n_0\hat{\mu}}{m + n_0}.$$

Posterior of mean and covariance where $\hat{\mu}, \hat{\Sigma}$ are the empirical means and covariances and Φ, μ_0, n_0, m an inverse Wishart prior

Learning Neural Network Policies with guided Policy Search under Unknown Dynamics, Levine and Abbeel 2014

Bayesian Linear dynamics fitting

Fit a *Global* Model of Dynamics by fitting a Neural Network using all samples (x_t, u_t, x_{t+1}) of all iterations and time steps, and across multiple manipulation tasks->multi-task learning.

Use model predictive control with iLQR for computing the policy at every time step.

State is the robotic arm configuration and cost depends on a desired end-effector pose. No object involved in the state.

$$\bar{f}([\mathbf{x};\mathbf{u}]) \approx \bar{f}([\mathbf{x}_i;\mathbf{u}_i]) + \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]}^{\mathrm{T}}([\mathbf{x};\mathbf{u}] - [\mathbf{x}_i;\mathbf{u}_i])$$

$$\bar{\mu} = \begin{bmatrix} [\mathbf{x}_i;\mathbf{u}_i] \\ \bar{f}([\mathbf{x}_i;\mathbf{u}_i]) \end{bmatrix}$$

$$\bar{\Sigma} = \begin{bmatrix} \bar{\Sigma}_{\mathbf{xu},\mathbf{xu}} & \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]}^{\mathrm{T}} \bar{\Sigma}_{\mathbf{xu},\mathbf{xu}} \\ \bar{\Sigma}_{\mathbf{xu},\mathbf{xu}} \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]} & \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]}^{\mathrm{T}} \bar{\Sigma}_{\mathbf{xu},\mathbf{xu}} \frac{d\bar{f}}{d[\mathbf{x};\mathbf{u}]} + \bar{\Sigma}_{\mathbf{x}',\mathbf{x}'} \end{bmatrix}$$

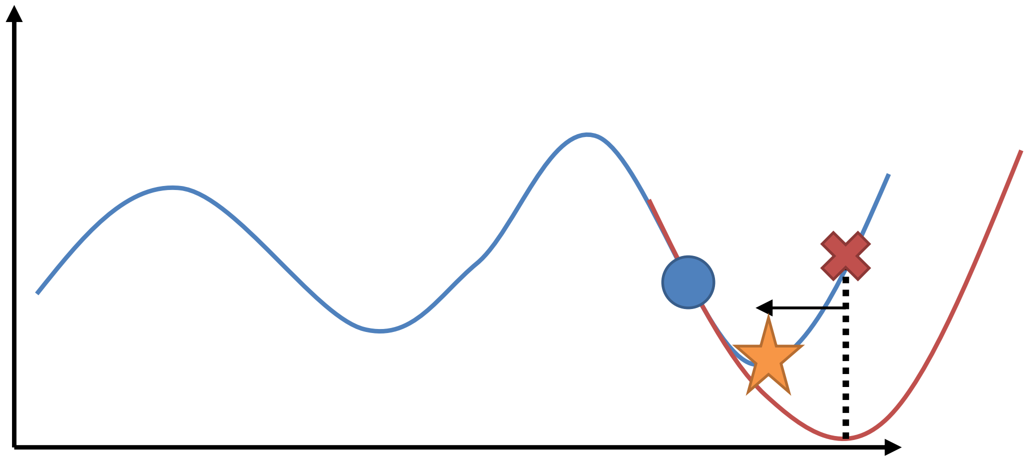
One shot Learning of Manipulation Skills with Online Dynamic Adaptation and Neural Network Priors, Fu et al.

Step-size in iterative LQR

Remember from the last lecture:

The quadratic approximation in invalid too far away from the reference trajectory

$$\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x}-\hat{\mathbf{x}})^T\mathbf{H}(\mathbf{x}-\hat{\mathbf{x}}) + \mathbf{g}^T(\mathbf{x}-\hat{\mathbf{x}})$$

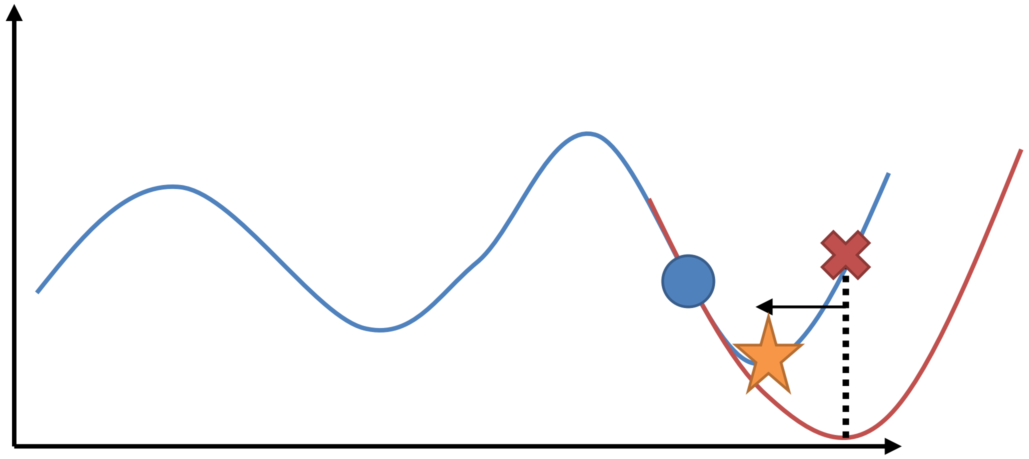


Step-size in iterative LQR

Remember from the last lecture:

The quadratic approximation in invalid too far away from the reference trajectory

$$\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x}-\hat{\mathbf{x}})^T\mathbf{H}(\mathbf{x}-\hat{\mathbf{x}}) + \mathbf{g}^T(\mathbf{x}-\hat{\mathbf{x}})$$



Instead of using the argmin we do a line search:

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{C}_t = \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

Run LQR backward pass on state $\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$

Run forward pass with real nonlinear dynamics and $u_t = \hat{u}_t + K_t(x_t - \hat{x}_T) + \alpha k_t$

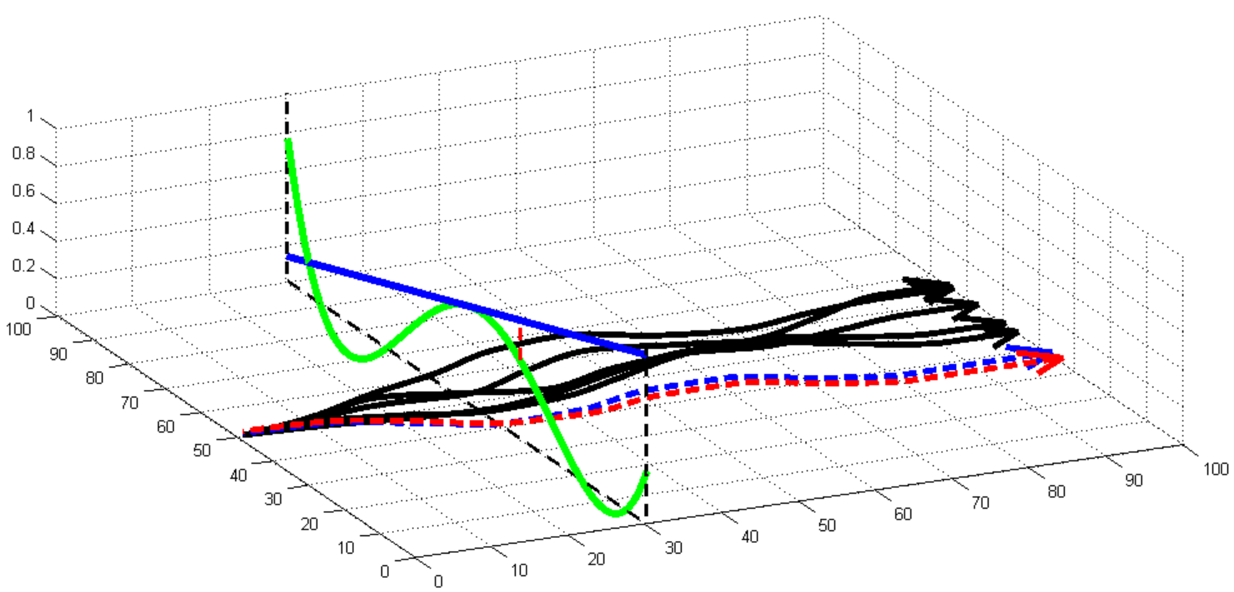
Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass

line search for α

Step-size in iterative LQR

- Both the quadratic cost approximation and the fitted linear dynamics are invalid too far *away from the reference trajectory*.
- We want the trajectory distributions not to change much from iteration to iteration of our policy.
- Constraint the KL divergence between trajectory distributions:

$$D_{\mathrm{KL}}(p(\tau)||\bar{p}(\tau)) \leq \epsilon$$



$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

$$p(\tau) = p(\mathbf{x}_1) \prod_{t=1}^{T} p(\mathbf{u}_t|\mathbf{x}_t) p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$$

KL-divergences between trajectories

KL divergence between trajectory distributions translates to KL divergence between policies.

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = E_{p(\tau)}[\log p(\tau) - \log \bar{p}(\tau)]$$

$$p(\tau) = p(\mathbf{x}_1)\prod_{t=1}^{T} p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t) \qquad \bar{p}(\tau) = \underline{p(\mathbf{x}_1)}\prod_{t=1}^{T} \bar{p}(\mathbf{u}_t|\mathbf{x}_t)\underline{p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)}$$

dynamics & initial state are the same!

KL-divergences between trajectories

KL divergence between trajectory distributions translates to KL divergence between policies.

$$D_{\text{KL}}(p(\tau)\|\bar{p}(\tau)) = E_{p(\tau)}[\log p(\tau) - \log \bar{p}(\tau)]$$

$$p(\tau) = p(\mathbf{x}_1)\prod_{t=1}^{T} p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t) \qquad \bar{p}(\tau) = \underline{p(\mathbf{x}_1)}\prod_{t=1}^{T} \bar{p}(\mathbf{u}_t|\mathbf{x}_t)\underline{p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)}$$

dynamics & initial state are the same!

$$\log p(\tau) - \log \bar{p}(\tau) = \log p(x_1) + \sum_{t=1}^{T} \log p(u_t|x_t) + \log p(x_{t+1}|x_t,u_t)$$
$$- \log p(x_1) + \sum_{t=1}^{T} - \log \bar{p}(u_t|x_t) - \log p(x_{t+1}|x_t,u_t)$$

KL-divergences between trajectories

KL divergence between trajectory distributions translates to KL divergence between policies.

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = E_{p(\tau)}[\log p(\tau) - \log \bar{p}(\tau)]$$

$$p(\tau) = p(\mathbf{x}_1)\prod_{t=1}^{T} p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t) \qquad \bar{p}(\tau) = \underline{p(\mathbf{x}_1)}\prod_{t=1}^{T} \bar{p}(\mathbf{u}_t|\mathbf{x}_t)\underline{p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)}$$

dynamics & initial state are the same!

$$\log p(\tau) - \log \bar{p}(\tau) = \log p(x_1) + \sum_{t=1}^{T} \log p(u_t|x_t) + \log p(x_{t+1}|x_t,u_t)$$
$$- \log p(x_1) + \sum_{t=1}^{T} - \log \bar{p}(u_t|x_t) - \log p(x_{t+1}|x_t,u_t)$$

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = E_{p(\tau)}\left[\sum_{t=1}^{T} \log p(\mathbf{u}_t|\mathbf{x}_t) - \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t)\right]$$

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\log p(\mathbf{u}_t|\mathbf{x}_t) - \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t)]$$

KL-divergences between trajectories

KL divergence between trajectory distributions translates to KL divergence between policies.

KL divergence constraints are important to ensure monotonic improvement of the policy behavior also in model-free environments.

Covariant policy search (Bagnell et all), Natural policy gradient (Kakade 2001), Relative entropy policy search (Peters et al. 2003), utilize such constraints when taking the policy gradient.

Theoretical guarantees for a general policy parametrization and a practical algorithm were given recently in the TRPO Schulman et al.

Trust Region Policy Optimization

Police gradients with monotonic guarantees!

- Police gradients: have a function approximation for the policy $\pi_\theta(u|x)$ and optimize use SGD. SGD is sufficient to learn great object object detectors for example. What is different in RL?
- Non-stationarity in RL: *Each time the policy changes the state visitation distribution changes.* And this can cause the policy to diverge!
- Contribution: theoretical and practical method of how big of a step our gradient can take.

Trust Region Policy Optimization, Schulman et al. 2015

Problem Setup

Problem: minimize expected cost of policy

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t), \text{ where}\right]$$

$$s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$$

- Suppose we execute policy π in the MDP, obtaining a set of trajectories.
  - Using these trajectories, can we construct loss function L that is a local approximation for the expected cost η?

A Neat Identity

Advantage function: $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$

Visitation distribution: $\rho_\pi(s) = (P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots)$

Expected cost of new policy can be written in terms of old one

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$$

Approximately Optimal Approximate Reinforcement Learning, Kakade and Langford 2002

Surrogate Loss Function

$\eta(\tilde{\pi})$ has complicated dependence on $\tilde{\pi}$ through $\rho_{\tilde{\pi}}(s)$

Define surrogate loss L, a local approximation to η

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$$

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\pi}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$$

Improvement Theorem

$$\eta(\tilde{\pi}) \leq L_\pi(\tilde{\pi}) + C D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}), \text{ where } C = \frac{2\epsilon\gamma}{(1-\gamma)^2}$$

$$\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'(a|s)}\left[A_\pi(s,a)\right]\right|$$

$$D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\mathrm{KL}}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s)).$$

Mixture policy update considered by Kakade and Langford:

$$\pi_{\mathrm{new}}(a|s) = (1-\alpha)\pi_{\mathrm{old}}(a|s) + \alpha\pi'(a|s).$$

Approximately Optimal Approximate Reinforcement Learning, Kakade and Langford 2002

Algorithm

- Optimize surrogate loss + KL penalty => guaranteed improvement to η

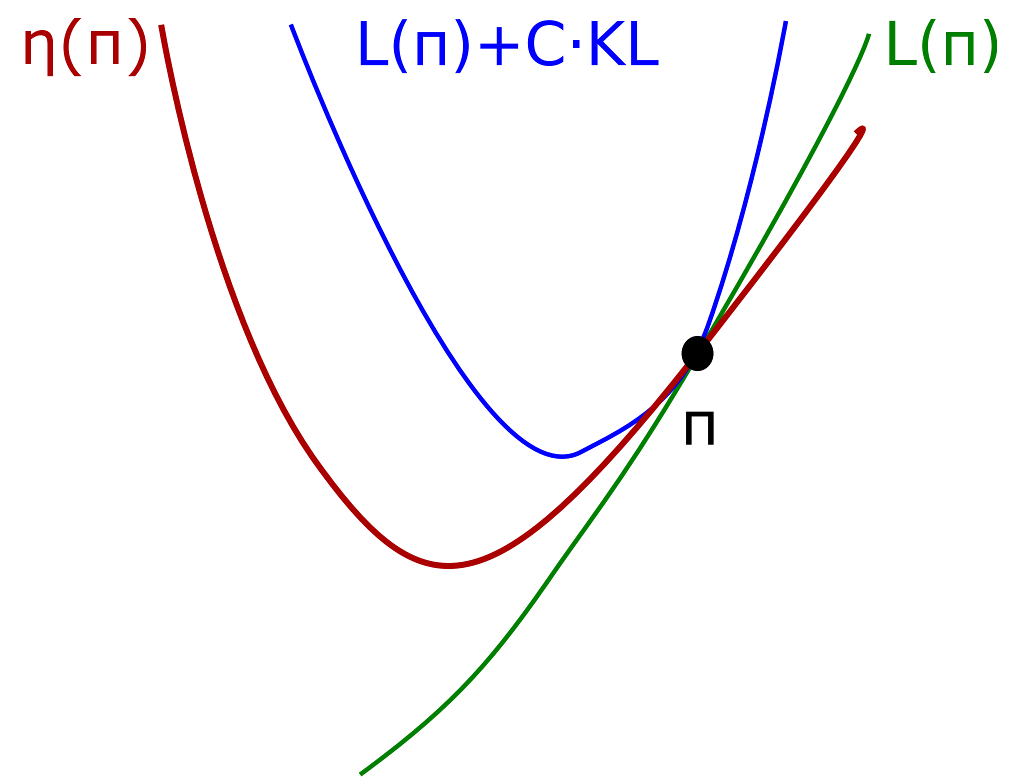


diagram John Schulman

Review

- Devised a surrogate loss L, which is a tractable local approximation to η

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$$

- KL-penalized surrogate loss majorities the true objective η

$$\eta(\tilde{\pi}) \leq L_\pi(\tilde{\pi}) + C D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}), \text{ where } C = \frac{2\epsilon\gamma}{(1-\gamma)^2}$$

- We don't have an algorithm yet: need to construct L_π from sampled data, and make approximations

Sampling

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$$

- Want to construct an objective that in expectation equals L plus a constant independent of $\tilde{\pi}$
  - Execute policy to sample states from ρ_π
  - Use empirical returns in place of A_π

Approximations

$$\underset{\theta}{\text{minimize}}\,[L_{\theta_{\text{old}}}(\theta) + C D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta)]$$

Approx #1: use trust region instead of penalty

$$\underset{\theta}{\text{minimize}}\, L_{\theta_{\text{old}}}(\theta)$$

$$\text{subject to } D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \leq \delta.$$

Approx #2: use mean KL instead of max KL

$$\underset{\theta}{\text{minimize}}\, L_{\theta_{\text{old}}}(\theta)$$

$$\text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leq \delta.$$

Relation to Policy Iteration and Natural Policy Gradient

Trust region policy optimization:

$$\underset{\theta}{\text{minimize}}\, L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \le \delta.$$

Policy Gradient:

$$\underset{\theta}{\text{minimize}} \left[\nabla_\theta L_{\theta_{\text{old}}}(\theta) \big|_{\theta=\theta_{\text{old}}} \cdot (\theta - \theta_{\text{old}}) \right]$$
$$\text{subject to } \frac{1}{2}\|\theta - \theta_{\text{old}}\|^2 \le \delta.$$

Natural Policy Gradient:

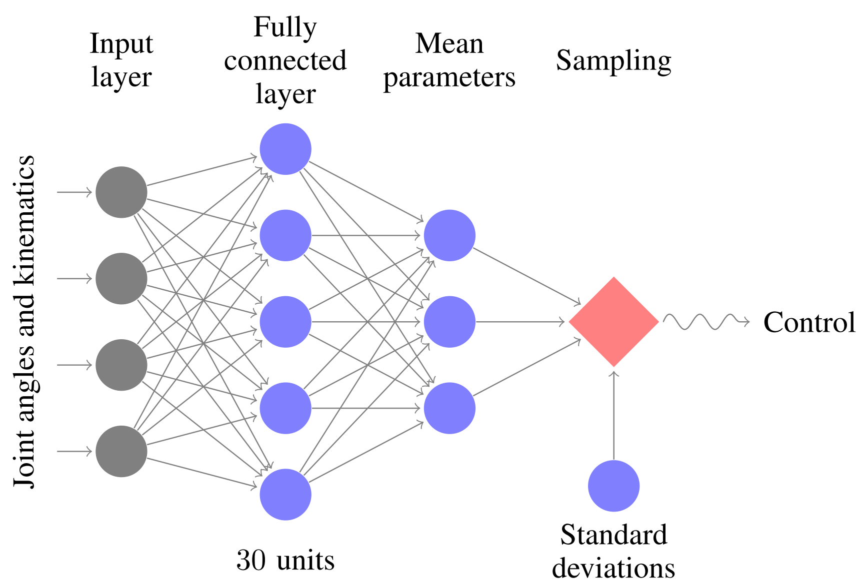
TRPO in limit as $\delta \to 0$

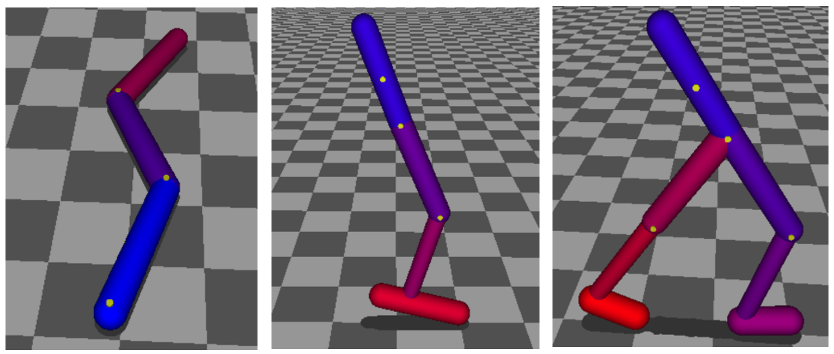
Relative entropy policy search (Peters et al. 2010) constraints the state-action marginals p(a,s) instead of p(a|s)

- How to solve this constrained optimization problem at every iteration?
- Authors used a direction search based on quadratic approximation of the constraint and then line search to find the step so that constraint is not violated and the surrogate cost goes down.

Experiments: Simulated Robot Control

Policy parametrization as a neural network





Cost function: move forward and don't fall over

diagram John Schulman

Constraint Optimization in iLQR

$$\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t)] \text{ s.t. } D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon$$

KL-divergences between trajectories:

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}\left[\log p(\mathbf{u}_t|\mathbf{x}_t) - \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t)\right]$$

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}\left[-\log \bar{p}(\mathbf{u}_t|\mathbf{x}_t)\right] + E_{p(\mathbf{x}_t)}[\underbrace{E_{p(\mathbf{u}_t|\mathbf{x}_t)}[\log p(\mathbf{u}_t|\mathbf{x}_t)]}]$$

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}\left[-\log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))\right]$$

KL-divergences between trajectories

We have the following constrained optimization problem:

$$\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t)] \text{ s.t. } D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon$$

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(x_t,u_t)} - [\log \bar{p}(u_t|x_t) - \mathcal{H}(p(u_t|x_t))]$$

Reminder: Linear-Gaussian solves $\min \sum_{t=1}^{T} E_{p(x_t,u_t)}[c(x_t,u_t) - \mathcal{H}(p(u_t|x_t))]$

$$p(u_t|x_t) = \mathcal{N}(K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t, \Sigma_t)$$

If we can get D_{KL} into the cost, we can just use iLQR!

We will solve it with dual gradient descent

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } C(\mathbf{x}) = 0$$

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x})$$

$$g(\lambda) = \inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$$

$$\lambda \leftarrow \arg\max_{\lambda} g(\lambda)$$

How to maximize? Compute gradients!

Digression: dual gradient descent

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } C(\mathbf{x}) = 0$$

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x})$$

$$g(\lambda) = \inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$$

$$g(\lambda) = \mathcal{L}(\mathbf{x}^\star(\lambda), \lambda)$$

$$\frac{dg}{d\lambda} = \cancel{\frac{d\mathcal{L}}{d\mathbf{x}^\star}} \frac{d\mathbf{x}^\star}{d\lambda} + \frac{d\mathcal{L}}{d\lambda}$$

if $\mathbf{x}^\star = \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$, then $\frac{d\mathcal{L}}{d\mathbf{x}^\star} = 0!$

Digression: dual gradient descent

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } C(\mathbf{x}) = 0$$

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x})$$

$$g(\lambda) = \mathcal{L}(\mathbf{x}^\star(\lambda), \lambda)$$

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$$

$$\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$$

1. Find $\mathbf{x}^\star \leftarrow \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$
2. Compute $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

DGD with iterative LQR

This is the constrained problem we want to solve:

$$\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t)] \text{ s.t. } D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon$$

$$D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}\left[-\log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))\right]$$

$$\mathcal{L}(p,\lambda) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t) - \lambda \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \lambda\mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))] - \lambda\epsilon$$

DGD with iterative LQR

$$\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t)] \text{ s.t. } D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon$$

$$\mathcal{L}(p,\lambda) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t) - \lambda \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \lambda \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))] - \lambda\epsilon$$

1. Find $p^* \leftarrow \arg\min_p \mathcal{L}(p,\lambda)$
2. Compute $\dfrac{dg}{d\lambda} = \dfrac{d\mathcal{L}}{d\lambda}(p^*,\lambda)$
3. $\lambda \leftarrow \lambda + \alpha \dfrac{dg}{d\lambda}$

DGD with iterative LQR

1. Find $p^\star \leftarrow \arg\min_p \mathcal{L}(p, \lambda)$

$$\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t) - \lambda \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \lambda\mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))] - \lambda\epsilon$$

Reminder: Linear-Gaussian solves $\min \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))]$

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

$$\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}\left[\frac{1}{\lambda}c(\mathbf{x}_t,\mathbf{u}_t) - \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))\right]$$

Just use LQR with cost $\tilde{c}(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{\lambda}c(\mathbf{x}_t,\mathbf{u}_t) - \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t)$

DGD with iterative LQR

$$\min_{p} \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[c(\mathbf{x}_t,\mathbf{u}_t)] \text{ s.t. } D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon$$

1. Set $\tilde{c}(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{\lambda}c(\mathbf{x}_t,\mathbf{u}_t) - \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t)$
2. Use LQR to find $p^{\star}(\mathbf{u}_t|\mathbf{x}_t)$ using $\tilde{c}$
3. $\lambda \leftarrow \lambda + \alpha(D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) - \epsilon)$

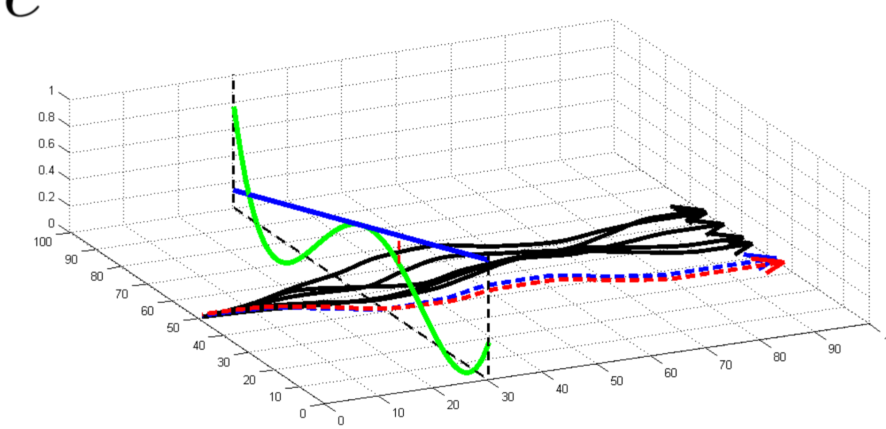


diagram Sergey Levine

So far..

- Learning local linear dynamics models
- Using KL divergence constraints for global and local policy search

Learning general policies by imitating local controllers

- Each iLQR controller achieves the task from a specific initial state x_0
- We want to learn general policies by mimicking such controllers. Why?
  - This policy will succeed under different forms of initial conditions. We hope with optimal controllers in the loop to do better than simple trial and error and require less human demonstrations than imitating human experts directly. However, ti will require measuring the cost at training time.
- Those general policies can be: a non parametric nearest neighbor local controller selection or a neural network policy $\pi_\theta(x)$

Imitating local controllers with DAGGER

Dataset AGGregation: bring learner's and expert's trajectory distributions closer by labelling additional data points resulting from applying the current policy

1. train $\pi_\theta(u_t|o_t)$ from human data $\mathcal{D}_{\pi^*} = \{o_1, u_1, ..., o_N, u_N\}$
2. run $\pi_\theta(u_t|o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, ..., o_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions u_t
4. Aggregate: $\mathcal{D}_{\pi^*} \leftarrow \mathcal{D}_{\pi^*} \cup \mathcal{D}_\pi$
5. GOTO step 1.

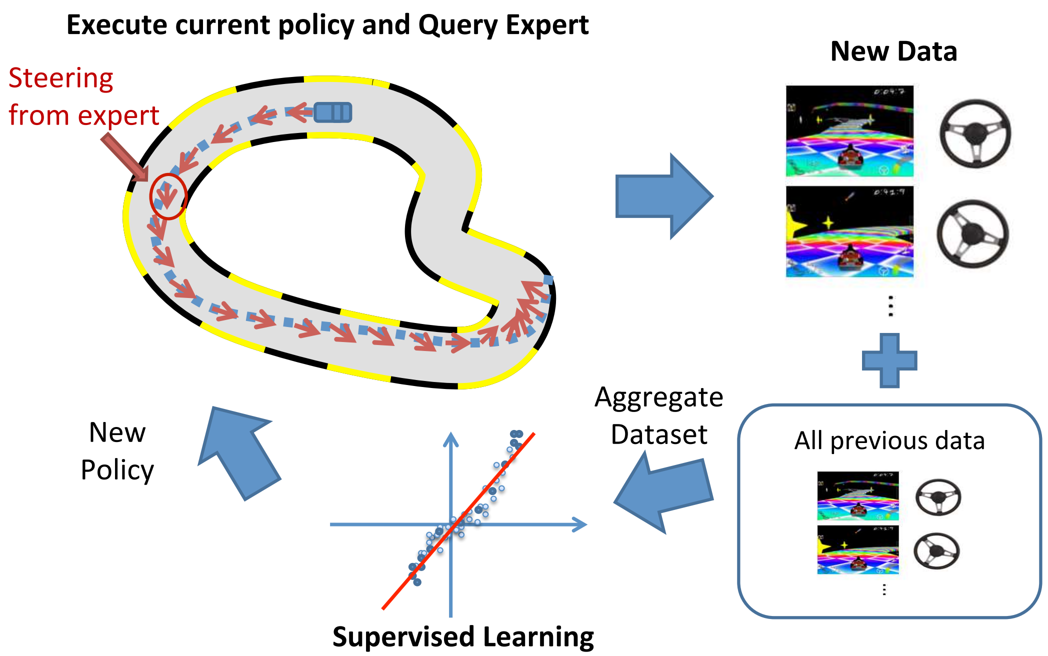
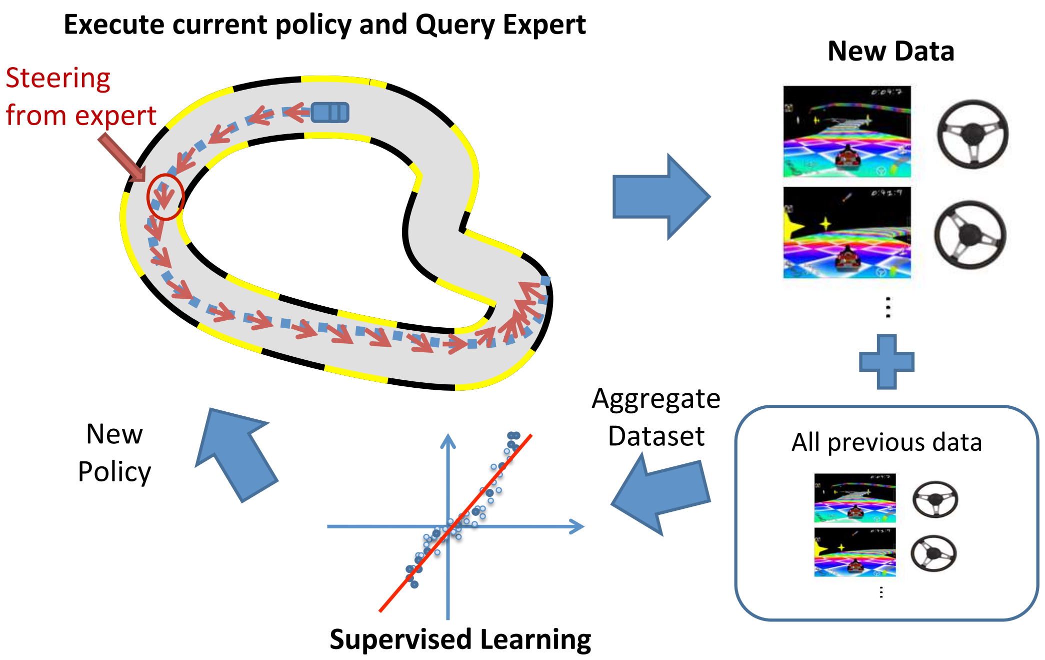


diagram: DAGGER paper

Imitating local controllers with DAGGER

Dataset AGGregation: bring learner's and expert's trajectory distributions closer by labelling additional data points resulting from applying the current policy

1. train $\pi_\theta(u_t|o_t)$ from human data $\mathcal{D}_{\pi^*} = \{o_1, u_1, ..., o_N, u_N\}$
2. run $\pi_\theta(u_t|o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, ..., o_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions u_t
4. Aggregate: $\mathcal{D}_{\pi^*} \leftarrow \mathcal{D}_{\pi^*} \cup \mathcal{D}_\pi$
5. GOTO step 1.

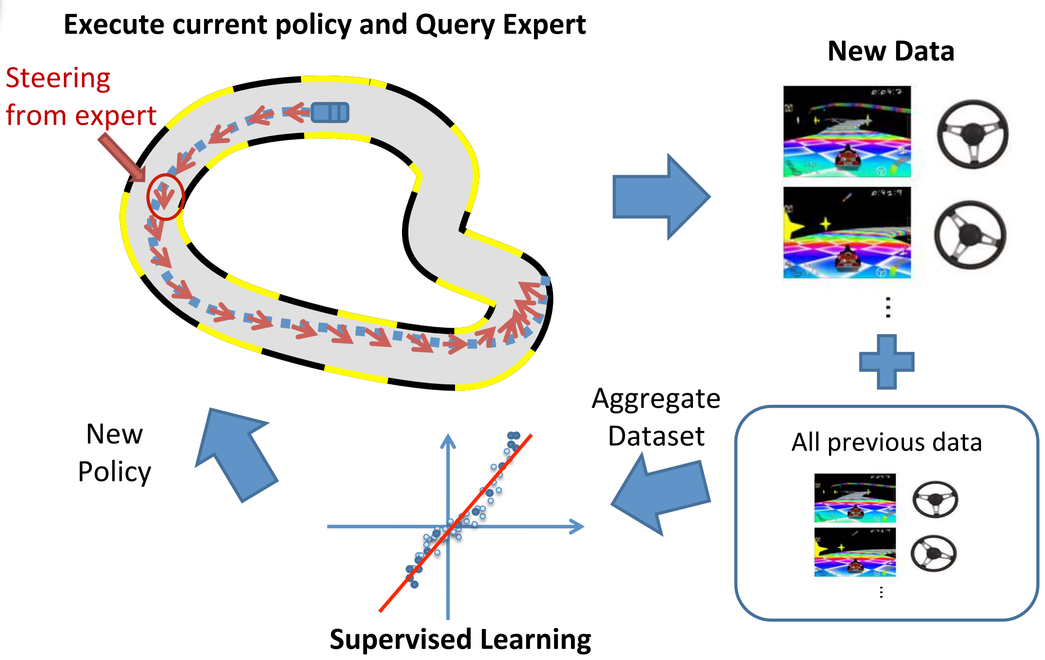


- repeatedly query the expert

Imitating local controllers with DAGGER

Dataset AGGregation: bring learner's and expert's trajectory distributions closer by labelling additional data points resulting from applying the current policy

1. train $\pi_\theta(u_t|o_t)$ from controller data $\mathcal{D}_{\pi^*} = \{o_1, u_1, ..., o_N, u_N\}$
2. run $\pi_\theta(u_t|o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, ..., o_M\}$
3. Ask controller to label $\mathcal{D}_\pi$ with actions u_t
4. Aggregate: $\mathcal{D}_{\pi^*} \leftarrow \mathcal{D}_{\pi^*} \cup \mathcal{D}_\pi$
5. GOTO step 1.

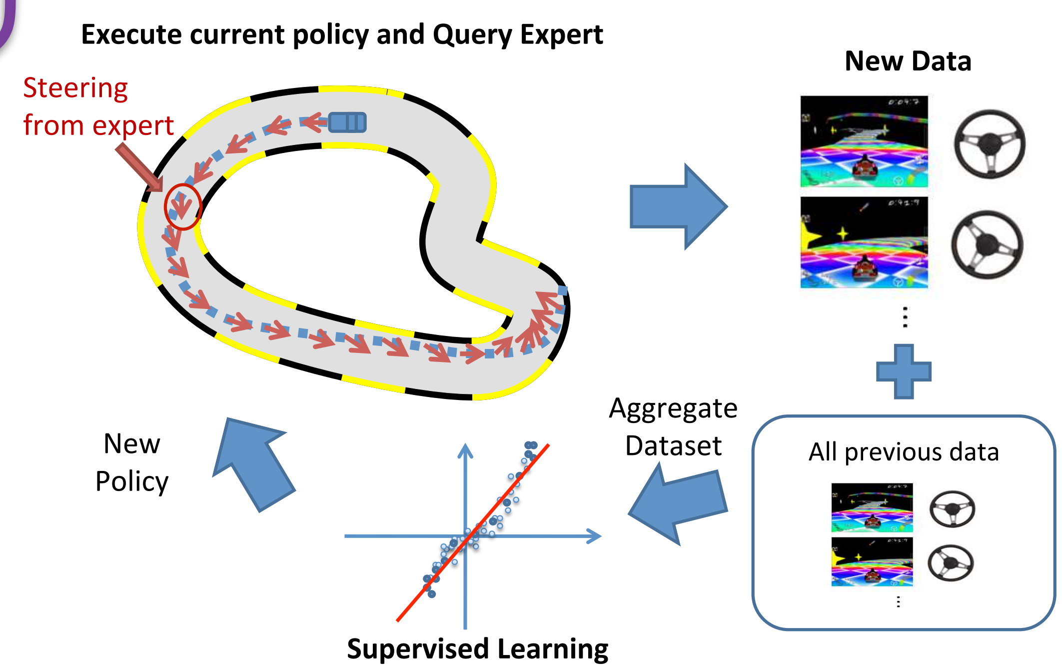


- repeatedly query the expert

Imitating local controllers with DAGGER

Dataset AGGregation: bring learner's and expert's trajectory distributions closer by labelling additional data points resulting from applying the current policy

1. train $\pi_\theta(u_t|o_t)$ from controller data $\mathcal{D}_{\pi^*} = \{o_1, u_1, ..., o_N, u_N\}$
2. run $\pi_\theta(u_t|o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, ..., o_M\}$
3. Ask controller to label $\mathcal{D}_\pi$ with actions u_t
4. Aggregate: $\mathcal{D}_{\pi^*} \leftarrow \mathcal{D}_{\pi^*} \cup \mathcal{D}_\pi$
5. GOTO step 1.



- execute an unsafe/partially trained policy
- repeatedly query the expert

DAGGER

- DAGGER assumes that the learner can imitate the expert. The expert comes close to the learner by matching the state distributions.

- Guided policy search does not require to execute a partially trained policy on hardware. The teacher further adapts to actions the learner can imitate.

Guided policy search, Levine and Koltun 2013

Guided Policy Search

- Impose constraints on trajectory optimization:

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T,\mathbf{x}_1,\dots,\mathbf{x}_T,\theta} \sum_{t=1}^{T} c(\mathbf{x}_t,\mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1},\mathbf{u}_{t-1}) \quad \left.\right\} \text{generic trajectory optimization, solve however you want}$$

$$\text{s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

Guided policy search, Levine and Koltun 2013

Solve it using dual gradient descent

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } C(\mathbf{x}) = 0$$

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x})$$

$$g(\lambda) = \mathcal{L}(\mathbf{x}^\star(\lambda), \lambda)$$

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$$

$$\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$$

1. Find $\mathbf{x}^\star \leftarrow \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$
2. Compute $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

A small tweak to DGD: augmented Lagrangian

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } C(\mathbf{x}) = 0$$

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x})$$

$$\bar{\mathcal{L}}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x}) + \rho ||C(\mathbf{x})||^2$$

- Still converges to correct solution
- When far from solution, quadratic term tends to improve stability
- Closely related to alternating direction method of multipliers (ADMM)

1. Find $\mathbf{x}^\star \leftarrow \arg\min_{\mathbf{x}} \bar{\mathcal{L}}(\mathbf{x}, \lambda)$
2. Compute $\frac{dg}{d\lambda} = \frac{d\bar{\mathcal{L}}}{d\lambda}(\mathbf{x}^\star, \lambda)$
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

Constraining trajectory optimization with dual gradient descent

$$\min_{\tau,\theta} c(\tau) \text{ s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

Lagrangian:

$$\mathcal{L}(\tau,\theta,\lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)$$

Augmented Lagrangian:

$$\bar{\mathcal{L}}(\tau,\theta,\lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t) + \sum_{t=1}^{T} \rho_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)^2$$

Stochastic (Gaussian) GPS

$$\min_{p,\theta} E_{\tau \sim p(\tau)}[c(\tau)] \text{ s.t. } p(\mathbf{u}_t|\mathbf{x}_t) = \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$$

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

$$\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\tilde{c}(\mathbf{x}_t, \mathbf{u}_t)] \text{ s.t. } D_{\text{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon$$

1. Optimize $p(\tau)$ with respect to some surrogate $\tilde{c}(x_t, u_t)$
2. Optimize θ with respect to some supervised objective
3. Increment or modify dual variables λ

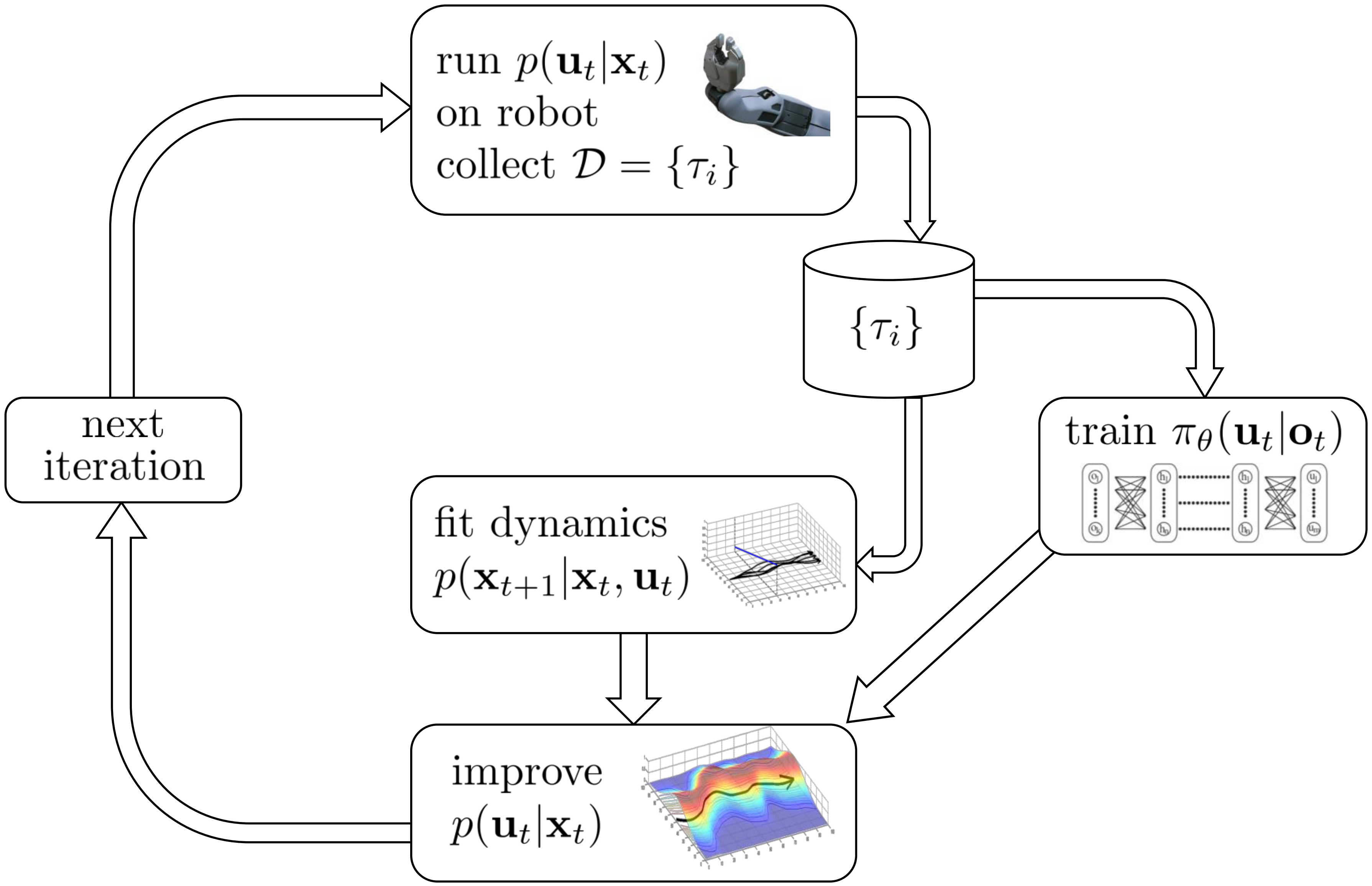
GPS with dual gradient descent

$$\min_{\tau,\theta} c(\tau) \text{ s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

$$\bar{\mathcal{L}}(\tau,\theta,\lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t) + \sum_{t=1}^{T} \rho_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)^2$$

1. Find $\tau \leftarrow \arg\min_\tau \bar{\mathcal{L}}(\tau,\theta,\lambda)$ (e.g. via iLQR)
2. Find $\theta \leftarrow \arg\min_\theta \bar{\mathcal{L}}(\tau,\theta,\lambda)$ (e.g. via SGD)
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

Guided policy search



Guided policy search

1. Find $\tau \leftarrow \arg\min_{\tau} \bar{\mathcal{L}}(\tau, \theta, \lambda)$ (e.g. via iLQR)
2. Find $\theta \leftarrow \arg\min_{\theta} \bar{\mathcal{L}}(\tau, \theta, \lambda)$ (e.g. via SGD)
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

- Can be interpreted as constrained trajectory optimization method
- Can be interpreted as imitation of an optimal control expert, since step 2 is just supervised learning
- The optimal control "teacher" adapts to the learner, and avoids actions that the learner can't mimic

DAGGER vs. GPS

- Dagger does not require an adaptive expert
  - Any expert will do, so long as states from learned policy can be labeled
  - Assumes it is possible to match expert's behavior up to bounded loss
    - Not always possible (e.g. partially observed domains)

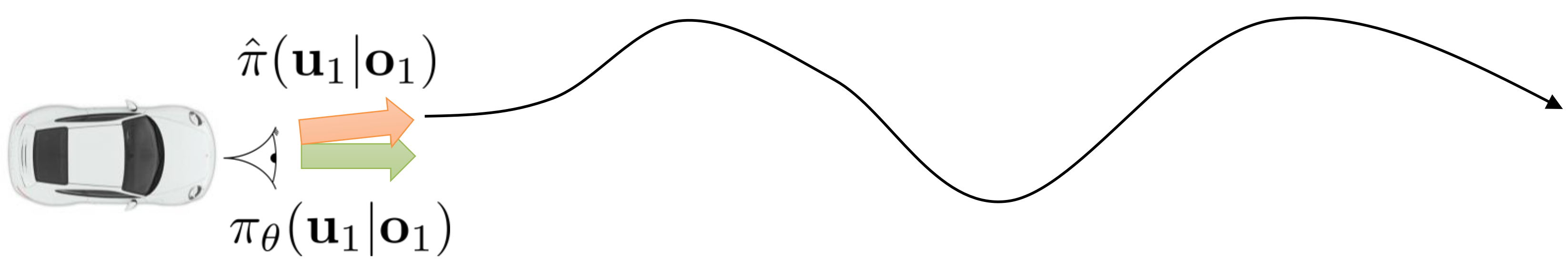
- GPS adapts the "expert" behavior
  - Does not require bounded loss on initial expert (expert will change)
  - It does require initial state resets!

Imitating MPC: PLATO algorithm

1. Train $\pi_\theta(u_t|o_t)$ from controller data $\mathcal{D} = \{o_1, u_1, ..., o_N, u_N\}$
2. Run $\hat{\pi}(u_t|o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, ..., o_M\}$
3. Ask computer to label $\mathcal{D}_\pi$ with actions u_t
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

simple stochastic policy: $\hat{\pi}(u_t|x_t) = \mathcal{N}(K_t x_t + k_t, \Sigma_{u_t})$

$$\hat{\pi}(u_t|x_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(x_{t'}, u_{t'})] + \lambda D_{\mathrm{KL}}(\hat{\pi}(u_t|x_t) \| \pi_\theta(u_t|o_t))$$



Imitating MPC: PLATO algorithm

simple stochastic policy: $\hat{\pi}(u_t|x_t) = \mathcal{N}(K_t x_t + k_t, \Sigma_{u_t})$

$$\hat{\pi}(u_t|x_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(x_{t'}, u_{t'})] + \lambda D_{\mathrm{KL}}(\hat{\pi}(u_t|x_t) \| \pi_\theta(u_t|o_t))$$

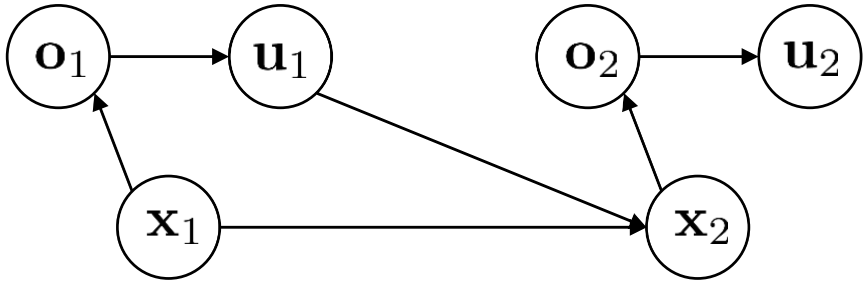
$\pi_\theta(u_t|o_t)$ Learner: trained from observations!

$$\overset{*}{\pi}(u_t|x_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(x_{t'}, u_{t'})]$$

Replanning = Model Predictive Control (MPC)

$\pi_\theta(u_t|o_t)$ - control from images

$\hat{\pi}(u_t|x_t)$ - control from states



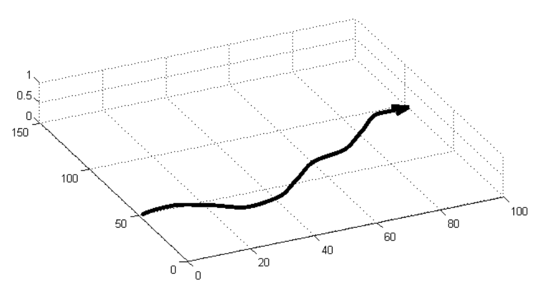
PLATO: Policy learning using Adaptive trajectory optimization, Kahn et al. 2016

Observability at train and test time

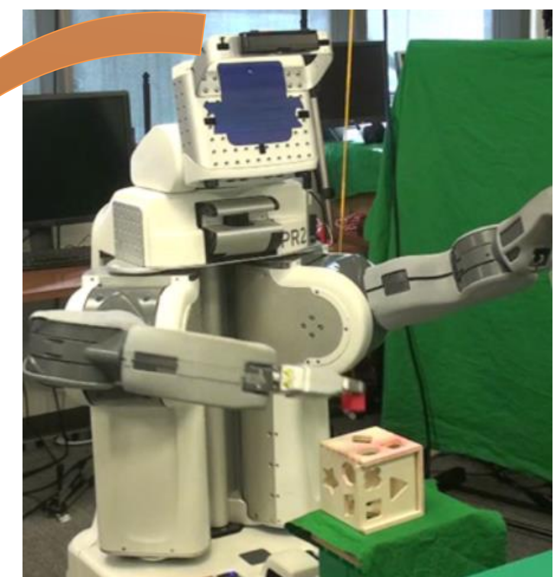
$$\min_{p,\theta} E_{\tau \sim p(\tau)}[c(\tau)] \text{ s.t. } p(u_t|x_t) = \pi_\theta(u_t|o_t)$$

training time

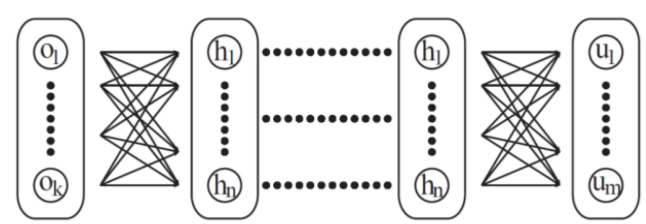
$\mathbf{x}_t \rightarrow \mathbf{u}_t$



test time



$\mathbf{o}_t \rightarrow \mathbf{u}_t$



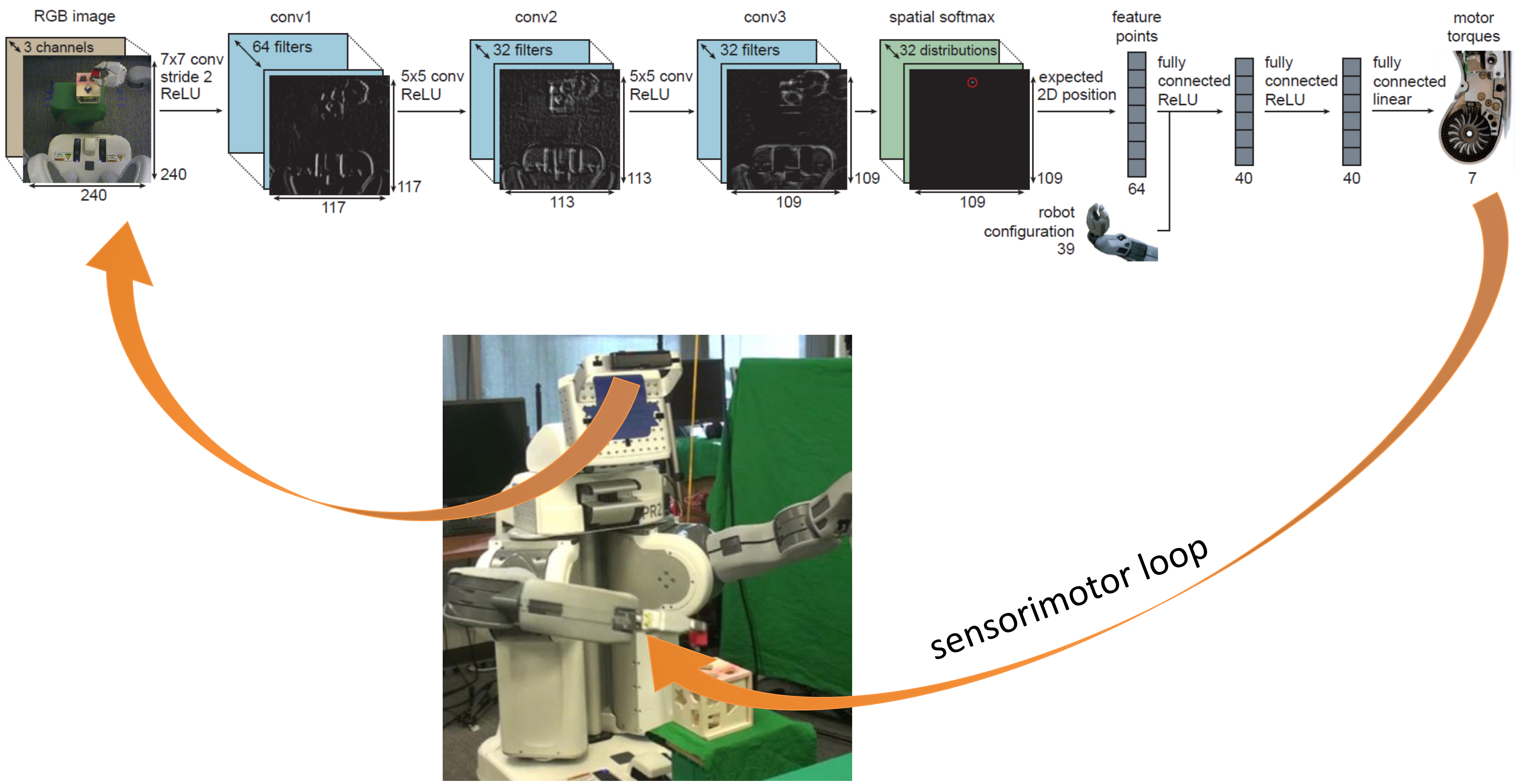
PLATO: Policy learning using Adaptive trajectory optimization, Kahn et al. 2016

Example: End-to-End training of Deep Visuomotor Policies

- Learning Neural Network general policies using direct RGB (no object detector, pose estimator or trackers) as input and trajectory optimization as supervision
- State: positions and velocities of joints, not object pose.
- Tasks: Swimmer, octopus etc, and peg insertion into a hole
- The RGB input is transformed to a set of x,y key points at the final layer to avoid overfitting
- Pretraining of the video CNN using object pose regression
- The environment is fully observable at training time (e.g., objects at known positions so that we know the desired state of the robotic arm), but not at test time
- Train and test environments are overall similar, due to the small amount of training data that can be collected in real world with instrumented training scenarios

End-to-End training of Deep Visuomotor Policies, Levine et al

Example: End-to-End training of Deep Visuomotor Policies



End-to-End training of Deep Visuomotor Policies, Levine et al.

Example: End-to-End training of Deep Visuomotor Policies

End-to-End training of Deep Visuomotor Policies, Levine et al

Example: Learning Dexterous Manipulation Policies from Experience and Imitation

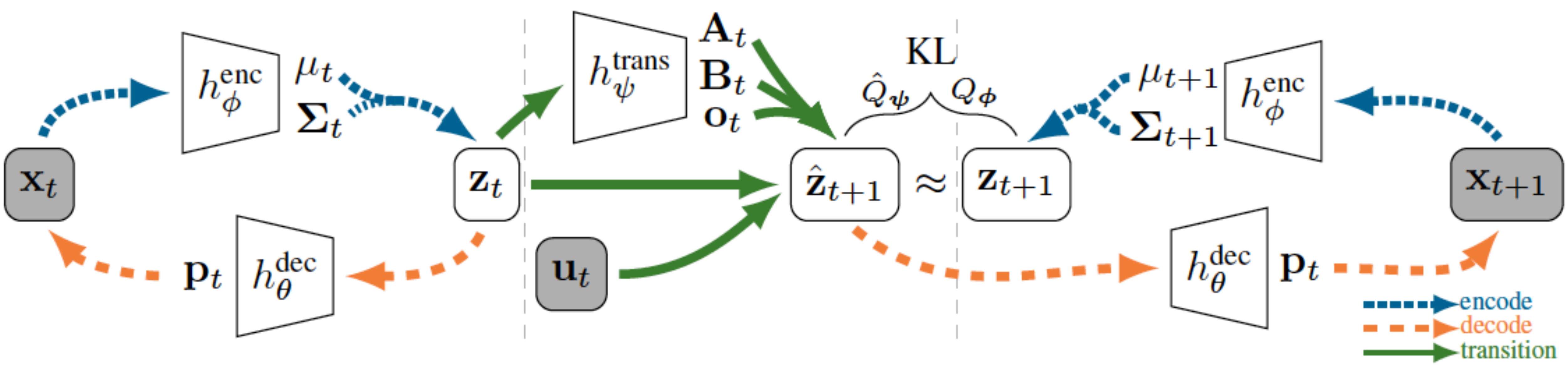
- Learning Neural Network and nearest neighbor based general policies using pose state as input and trajectory optimization as supervision
- State: positions and velocities of joints and objects—optitrack motion capture is used for object tracking at training time, and at test time for NNeib policy
- Tasks: dexterous manipulation, hard because of contact!
- iLQR fails without initialization from a demonstration!
- Nearest Neighbor using the object pose to determine which local policy to follow—requires saving all local controllers and knowing object pose (for effective matching)
- Neural net: can learn a mapping directly from on board sensing to actions, no vision, using GPS

Example: Learning Dexterous Manipulation Policies from Experience and Imitation

Learning Dexterous Manipulation Policies from Experience and Imitation, Kumar et al

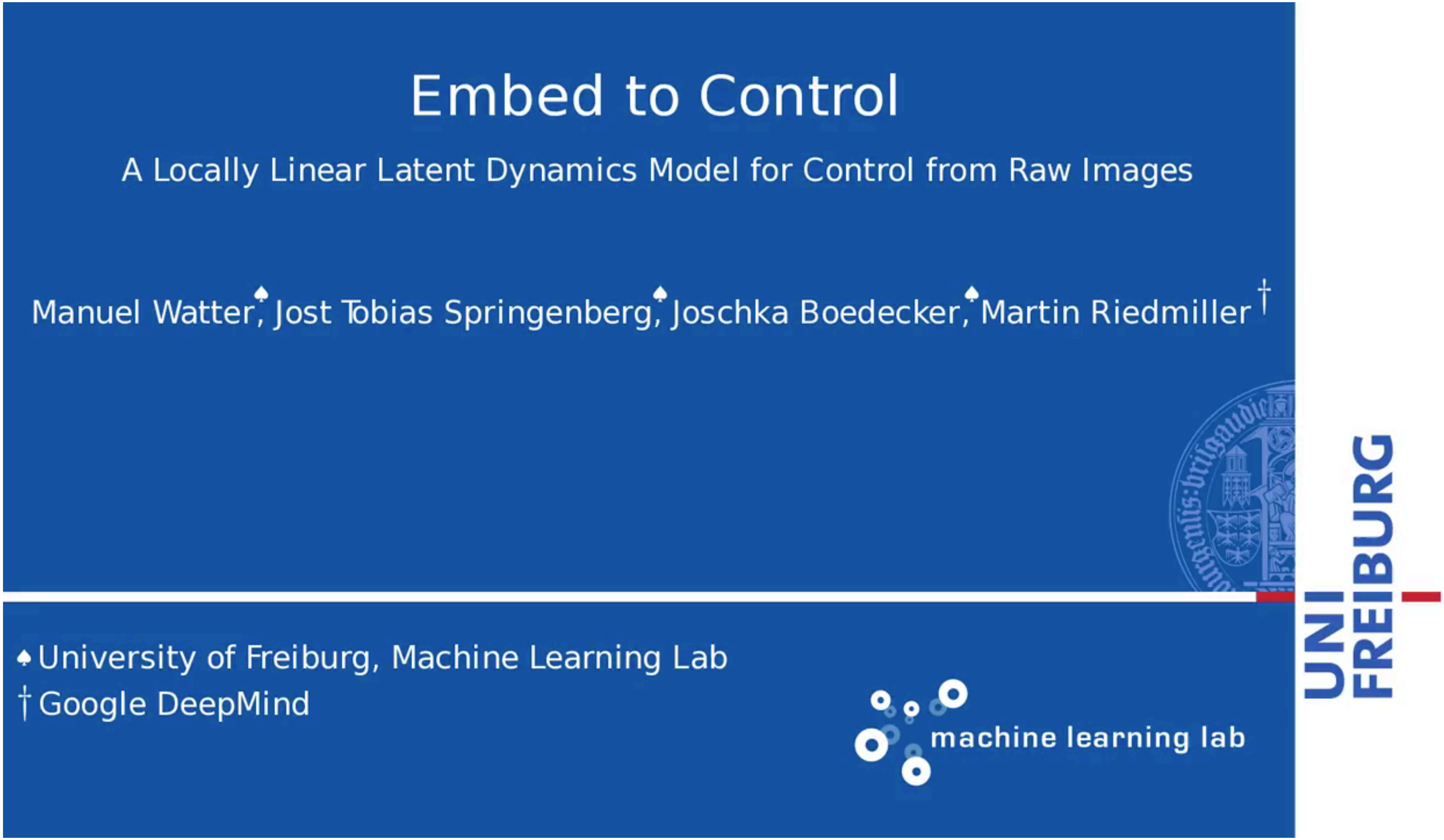
Embed to Control: A Locally Linear latent Dynamics model for Control from raw Images

- Infer a low-dimensional latent state space in which optimal control (LQR) can be used.
- Latent state should: 1) reconstruct the input image 2) predict the next state and then next observation 3) prediction should be locally linearizable



Embed to Control: A Locally Linear latent Dynamics model for Control from raw Images, Water et al.

Embed to Control: A Locally Linear latent Dynamics model for Control from raw Images



Embed to Control: A Locally Linear latent Dynamics model for Control from raw Images, Water et al.

Summary

- Learning local dynamics models
- i-LQR with learn local models
- Trust region constraint for policy optimization: TRPO and i-LQR
- Learning *general* policies by imitating i-LQR local controllers
    - DAGGER
    - Guided policy search

Next lecture

- Differentiable model-based reinforcement learning
- Recurrent networks and optimal control
- Back-propagate directly to the policy using temporal unfolding- differentiable dynamics- back propagate through discrete actions (stochastic sampling on the forward pass), or through continuous actions (re-paramertization trick)