Learning Dynamical System Models from Data

CS 294-112: Deep Reinforcement Learning

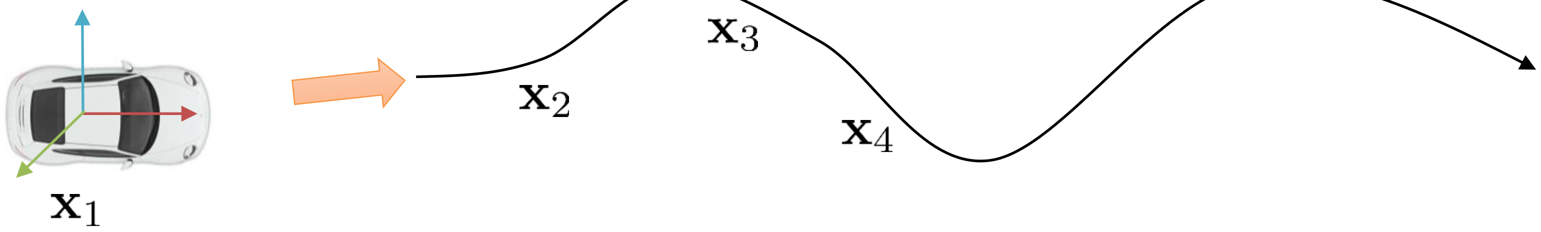
Week 3, Lecture 1

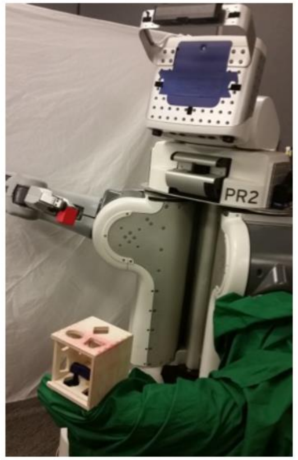
Sergey Levine

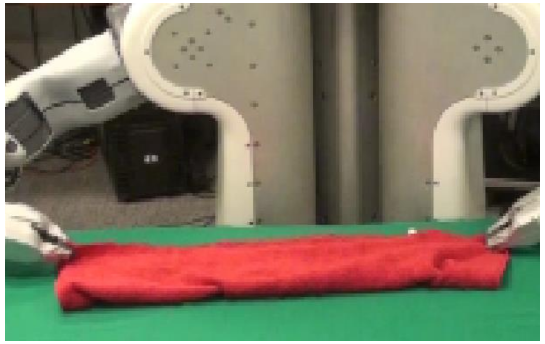
Overview

1. Before: learning to act by imitating a human
2. Last lecture: choose good actions autonomously by backpropagating (or planning) through *known* system dynamics (e.g. known physics)
3. Today: what do we do if the dynamics are *unknown*?
    a. Fitting global dynamics models ("model-based RL")
    b. Fitting local dynamics models
4. Wednesday: putting it all together to learn to "imitate" without a human (*e.g. by imitating optimal control*), so that we can train deep network policies autonomously

What’s wrong with known dynamics?







This lecture: learning the dynamics model

Today's Lecture

1. Overview of model-based RL
   - Learn only the model
   - Learn model & policy
2. What kind of models can we use?
3. Global models and local models
4. Learning with local models and trust regions

- Goals:
  - Understand the terminology and formalism of model-based RL
  - Understand the options for models we can use in model-based RL
  - Understand practical considerations of model learning

Why learn the model?

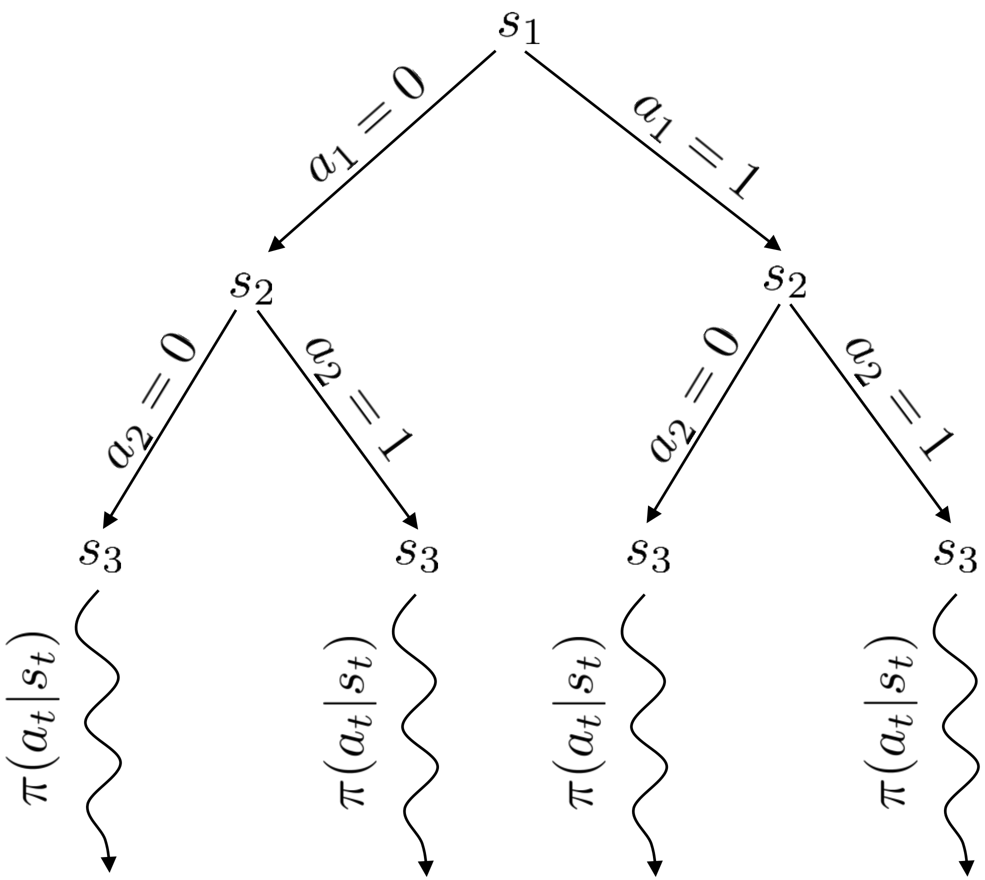
$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

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

usual story: differentiate via backpropagation and optimize!

need $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}, \frac{dc}{d\mathbf{x}_t}, \frac{dc}{d\mathbf{u}_t}$

Why learn the model?



Why learn the model?

If we knew $f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_{t+1}$, we could use the tools from last week.

(or $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ in the stochastic case)

So let's learn $f(\mathbf{x}_t, \mathbf{u}_t)$ from data, and *then* backpropagate through it!

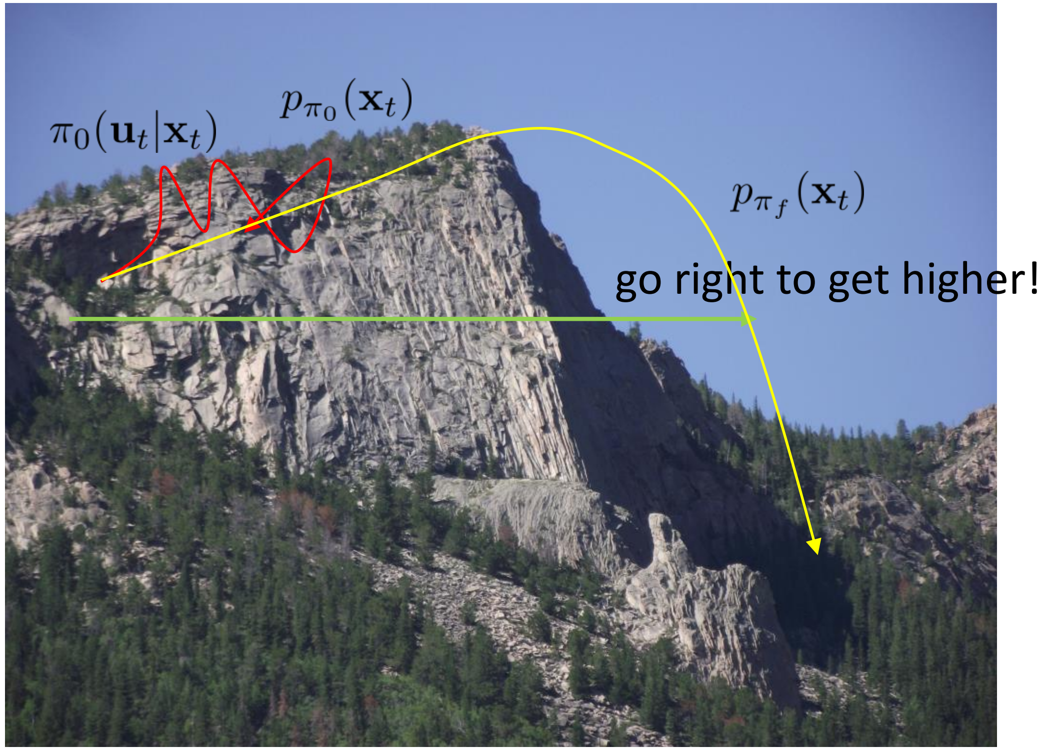
model-based reinforcement learning version 0.5:

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f(\mathbf{x}, \mathbf{u})$ to minimize $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. backpropagate through $f(\mathbf{x}, \mathbf{u})$ to choose actions (e.g. using iLQR)

Does it work? Yes!

- Essentially how system identification works in classical robotics
- Some care should be taken to design a good base policy
- Particularly effective if we can hand-engineer a dynamics representation using our knowledge of physics, and fit just a few parameters

Does it work? No!



1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f(\mathbf{x}, \mathbf{u})$ to minimize $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. backpropagate through $f(\mathbf{x}, \mathbf{u})$ to choose actions (e.g. using iLQR)

$$p_{\pi_f}(\mathbf{x}_t) \neq p_{\pi_0}(\mathbf{x}_t)$$

- Distribution mismatch problem becomes exacerbated as we use more expressive model classes

Can we do better?

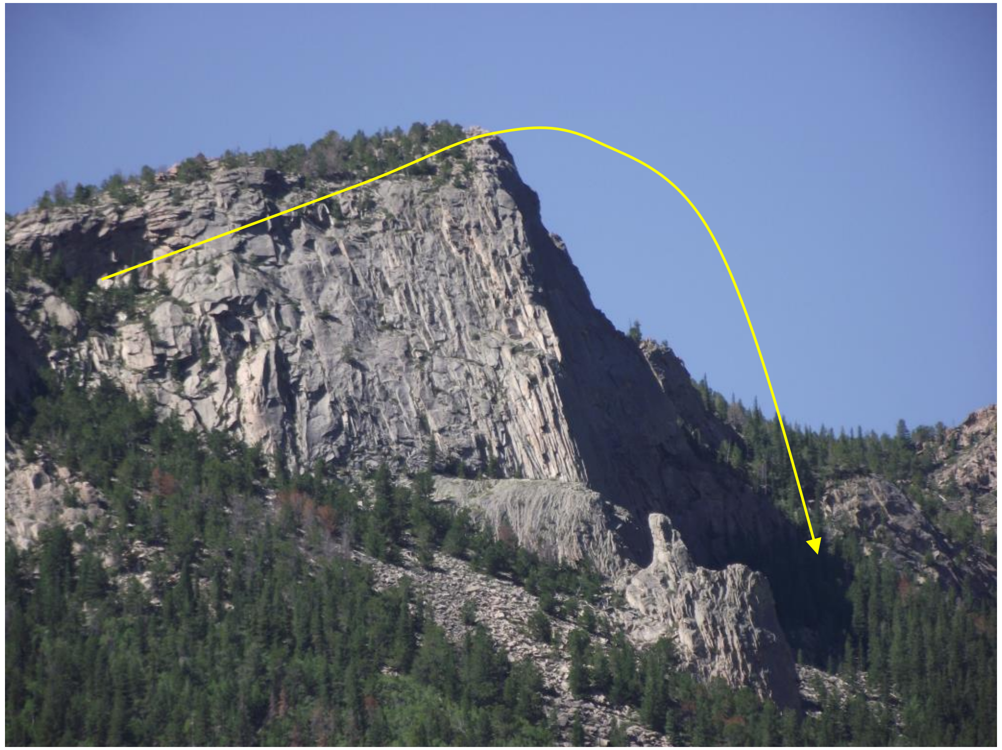
can we make $p_{\pi_0}(\mathbf{x}_t) = p_{\pi_f}(\mathbf{x}_t)$?

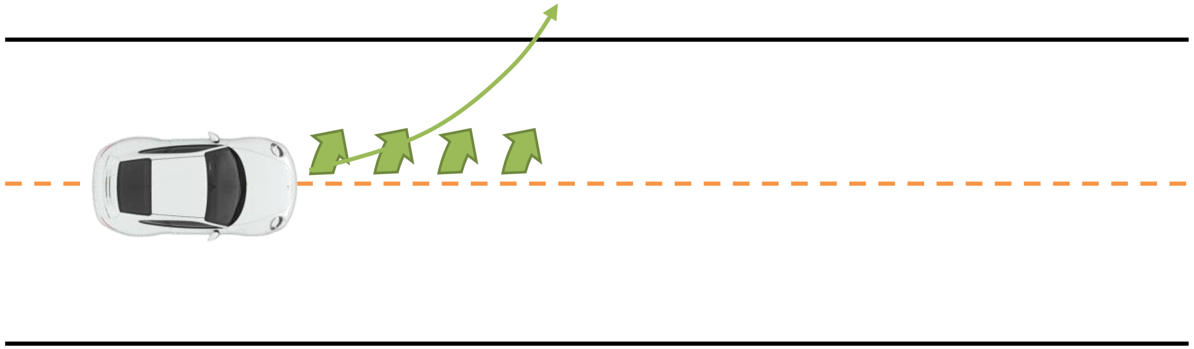
where have we seen that before? need to collect data from $p_{\pi_f}(\mathbf{x}_t)$

model-based reinforcement learning version 1.0:

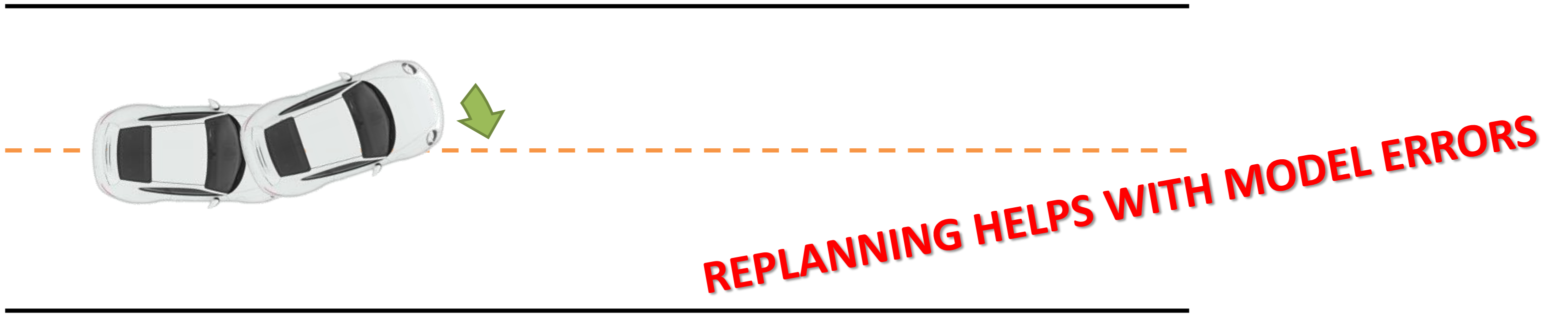
1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f(\mathbf{x}, \mathbf{u})$ to minimize $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. backpropagate through $f(\mathbf{x}, \mathbf{u})$ to choose actions (e.g. using iLQR)
4. execute those actions and add the resulting data $\{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_j\}$ to $\mathcal{D}$

What if we make a mistake?





Can we do better?



model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f(\mathbf{x}, \mathbf{u})$ to minimize $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. backpropagate through $f(\mathbf{x}, \mathbf{u})$ to choose actions (e.g. using iLQR)
4. execute the first planned action, observe resulting state $\mathbf{x}'$ (MPC)
5. append $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$ to dataset $\mathcal{D}$

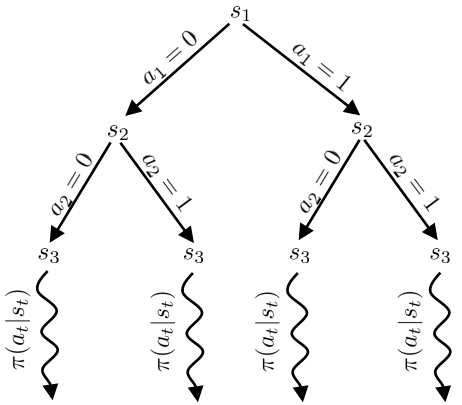
every N steps

That seems like a lot of work...

model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f(\mathbf{x}, \mathbf{u})$ to minimize $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. backpropagate through $f(\mathbf{x}, \mathbf{u})$ to choose actions (e.g. using iLQR)
4. execute the first planned action, observe resulting state $\mathbf{x}'$ (MPC)
5. append $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$ to dataset $\mathcal{D}$

every N steps



Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning

Xiaoxiao Guo
Computer Science and Eng.
University of Michigan
guoxiao@umich.edu

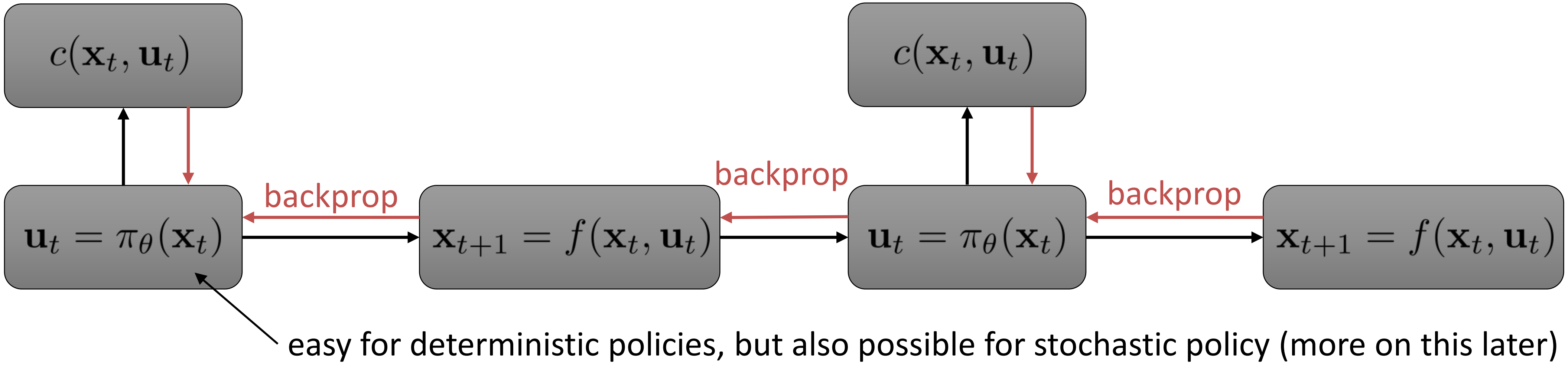
Satinder Singh
Computer Science and Eng.
University of Michigan
baveja@umich.edu

Honglak Lee
Computer Science and Eng.
University of Michigan
honglak@umich.edu

Richard Lewis
Department of Psychology
University of Michigan
rickl@umich.edu

Xiaoshi Wang
Computer Science and Eng.
University of Michigan
xiaoshiw@umich.edu

Backpropagate directly into the policy?



model-based reinforcement learning version 2.0:

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f(\mathbf{x}, \mathbf{u})$ to minimize $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. backpropagate through $f(\mathbf{x}, \mathbf{u})$ into the policy to optimize $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$
4. run $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, appending the visited tuples $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$ to $\mathcal{D}$

Summary

- Version 0.5: collect random samples, train dynamics, plan
  - Pro: simple, no iterative procedure
  - Con: distribution mismatch problem
- Version 1.0: iteratively collect data, replan, collect data
  - Pro: simple, solves distribution mismatch
  - Con: open loop plan might perform poorly, esp. in stochastic domains
- Version 1.5: iteratively collect data using MPC (replan at each step)
  - Pro: robust to small model errors
  - Con: computationally expensive, but have a planning algorithm available
- Version 2.0: backpropagate directly into policy
  - Pro: computationally cheap at runtime
  - Con: can be numerically unstable, especially in stochastic domains (more on this later)

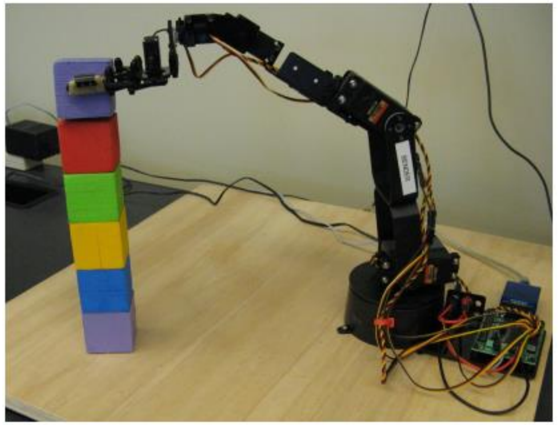
Case study: model-based policy search with GPs

Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning

Marc Peter Deisenroth
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA

Carl Edward Rasmussen
Dept. of Engineering
University of Cambridge
Cambridge, UK

Dieter Fox
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA



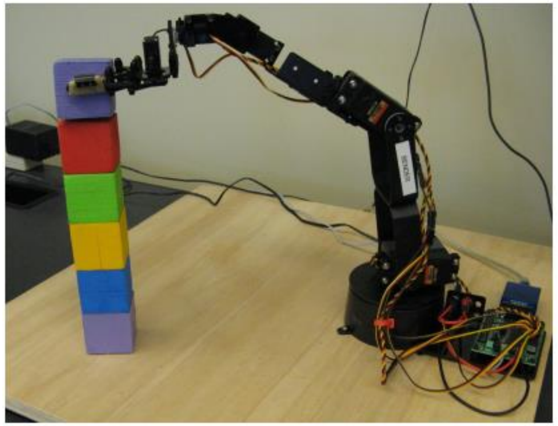
Case study: model-based policy search with GPs

Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning

Marc Peter Deisenroth
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA

Carl Edward Rasmussen
Dept. of Engineering
University of Cambridge
Cambridge, UK

Dieter Fox
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA



1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn GP dynamics model $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$ to maximize $\sum_i \log p(\mathbf{x}'_i|\mathbf{x}_i, \mathbf{u}_i)$
3. backpropagate through $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$ into the policy to optimize $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$
4. run $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, appending the visited tuples $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$ to $\mathcal{D}$

Case study: model-based policy search with GPs

3. backpropagate through $p(\mathbf{x}'|\mathbf{x},\mathbf{u})$ into the policy to optimize $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

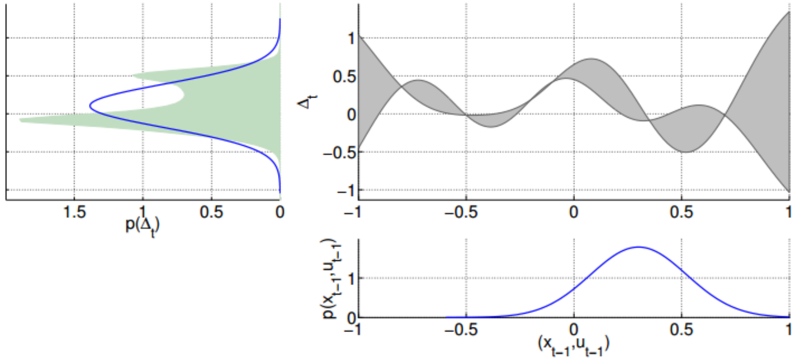
Given $p(\mathbf{x}_t)$, use $p(\mathbf{x}'|\mathbf{x},\mathbf{u})$ to compute $p(\mathbf{x}_{t+1})$

If $p(\mathbf{x}_t)$ is Gaussian, we can get a (non-Gaussian) $\bar{p}(\mathbf{x}_{t+1})$ in closed form

Project non-Gaussian $\bar{p}(\mathbf{x}_{t+1})$ to Gaussian $p(\mathbf{x}_{t+1})$ using moment matching

$E_{\mathbf{x}\sim p(\mathbf{x})}[c(\mathbf{x})]$ easy if c is nice and $p(\mathbf{x})$ Gaussian

Write $\sum_t E_{\mathbf{x}\sim p(\mathbf{x}_t)}[c(\mathbf{x}_t)]$ and differentiate

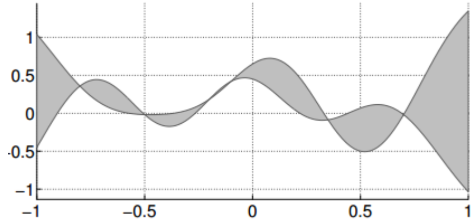


Marc Peter Deisenroth, Carl Edward Rasmussen, Dieter Fox

Learning to Control a Low-Cost Manipulator
using Data-efficient Reinforcement Learning

What kind of models can we use?

Gaussian process



GP with input $(\mathbf{x}, \mathbf{u})$ and output $\mathbf{x}'$

Pro: very data-efficient

Con: not great with non-smooth dynamics

Con: very slow when dataset is big

neural network

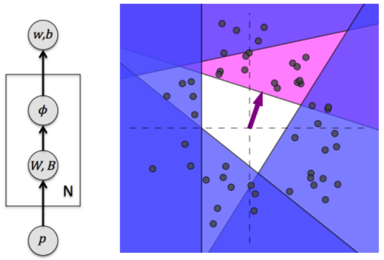


image: Punjani & Abbeel '14

Input is $(\mathbf{x}, \mathbf{u})$, output is $\mathbf{x}'$

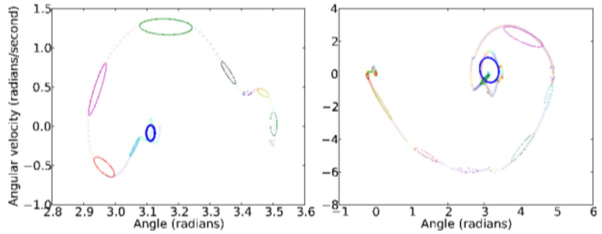
Euclidean training loss corresponds to Gaussian $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$

More complex losses, e.g. output parameters of Gaussian mixture

Pro: very expressive, can use lots of data

Con: not so great in low data regimes

other



GMM over $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$ tuples

Train on $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$, condition to get $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$

For i^{th} mixture element, $p_i(\mathbf{x}, \mathbf{u})$ gives region where the mode $p_i(\mathbf{x}'|\mathbf{x}, \mathbf{u})$ holds

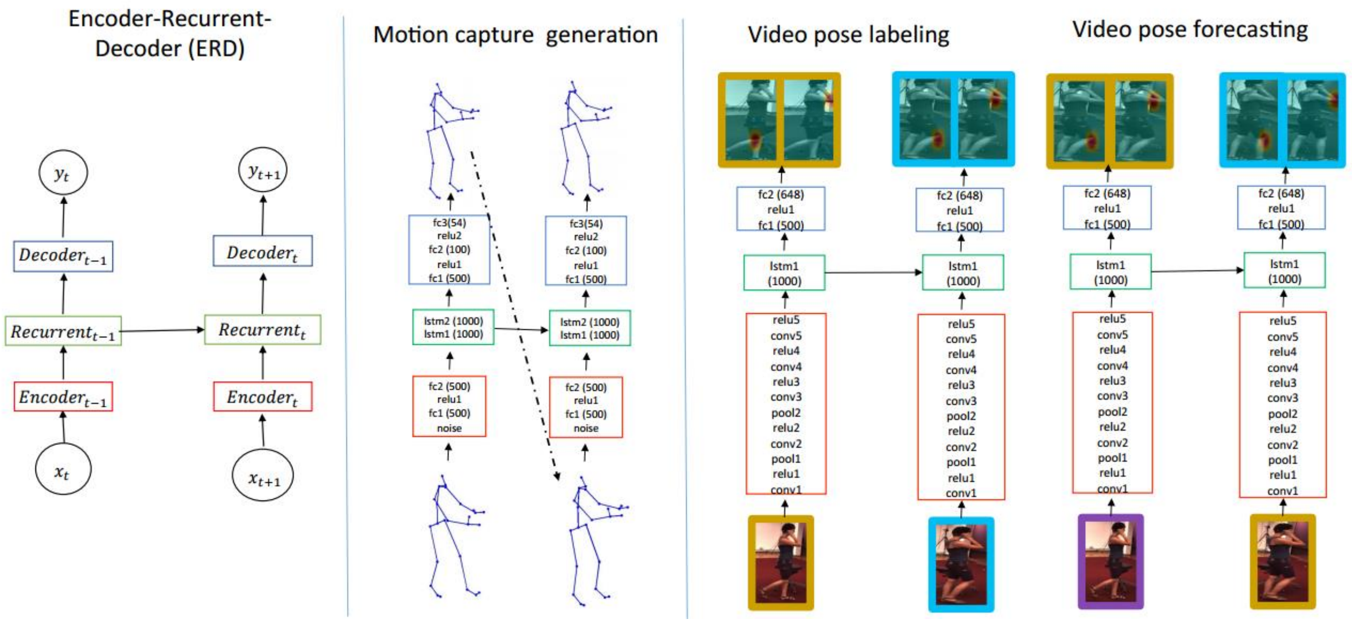
other classes: domain-specific models (e.g. physics parameters)

video prediction? more on this later in the course

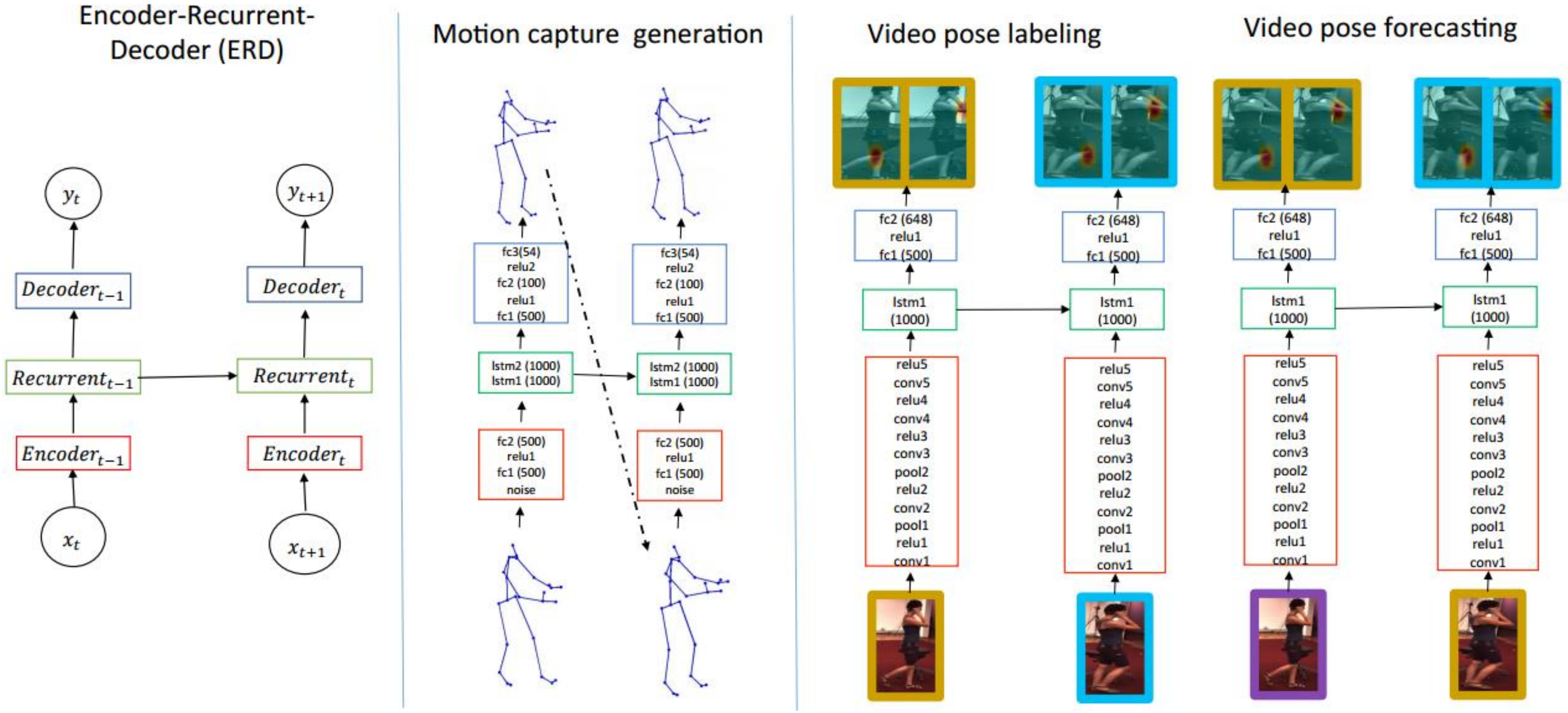
Case study: dynamics with recurrent networks

Recurrent Network Models for Human Dynamics

Katerina Fragkiadaki Sergey Levine Panna Felsen Jitendra Malik
University of California, Berkeley
Berkeley, CA
{katef,svlevine@eecs,panna@eecs,malik@eecs}.berkeley.edu

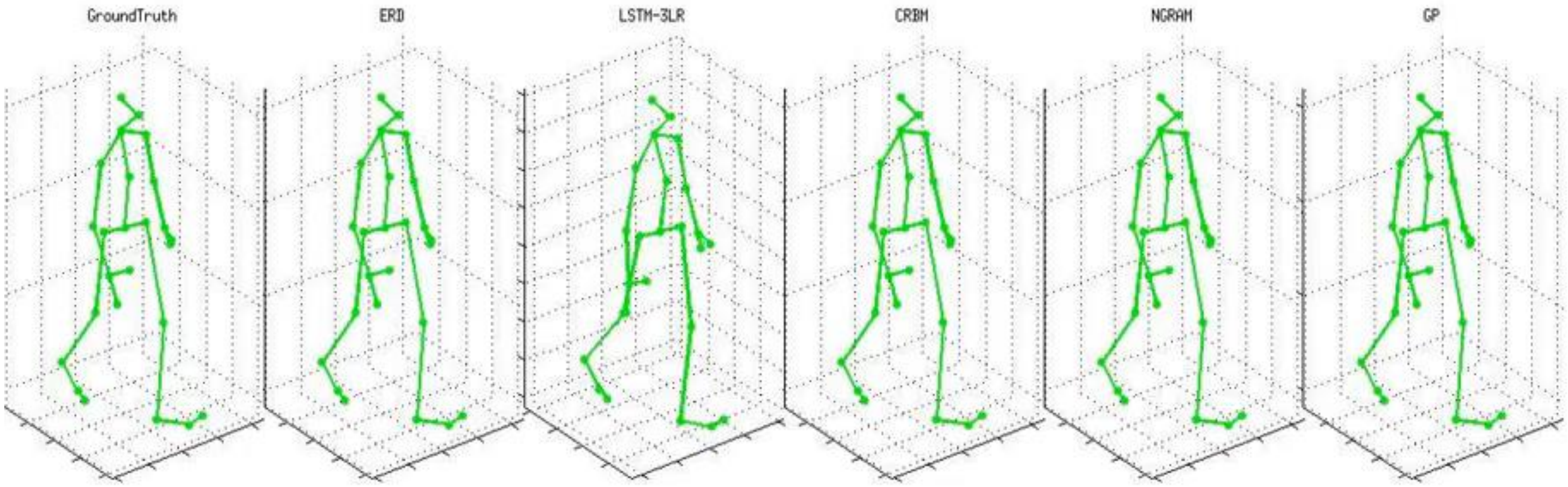


Case study: dynamics with recurrent networks



Other related work on learning human dynamics

- Conditional Restricted Boltzmann Machines (Taylor et al.)
- GPs and GPLVMs (Wang et al.)
- Linear and switching linear dynamical systems (Hsu & Popovic)
- Many others…
- Will compare:
  - ERD (this work)
  - LSTM with three layers
  - CRBM (probabilistic model trained with contrastive divergence)
  - Simple n-gram baseline
  - GP model


GroundTruth
ERD
LSTM-3LR
CRBM
NGRAM
GP

The trouble with global models

Global model: $f(\mathbf{x}_t, \mathbf{u}_t)$ represented by a big neural network

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f(\mathbf{x}, \mathbf{u})$ to minimize $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. backpropagate through $f(\mathbf{x}, \mathbf{u})$ to choose actions (e.g. using iLQR)
4. execute those actions and add the resulting data $\{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_j\}$ to $\mathcal{D}$

- Planner will seek out regions where the model is erroneously optimistic
- Need to find a very good model in most of the state space to converge on a good solution

The trouble with global models

- Planner will seek out regions where the model is erroneously optimistic
- Need to find a very good model in most of the state space to converge on a good solution
- In some tasks, the model is much more complex than the policy

Local models

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

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

usual story: differentiate via backpropagation and optimize!

need $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}, \frac{dc}{d\mathbf{x}_t}, \frac{dc}{d\mathbf{u}_t}$

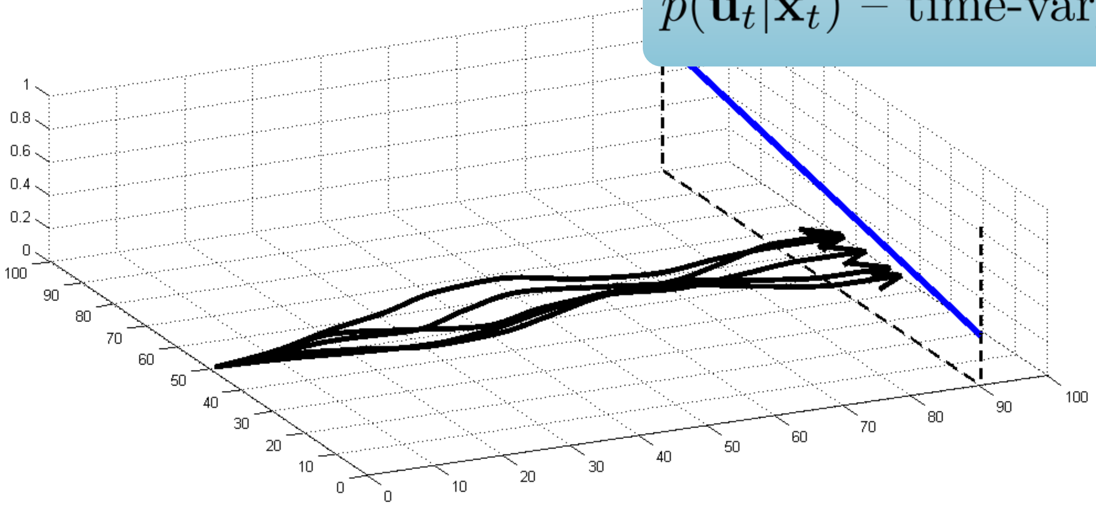
Local models

need $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}, \frac{dc}{d\mathbf{x}_t}, \frac{dc}{d\mathbf{u}_t}$

idea: just fit $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}$ around current trajectory or policy!

$p(\mathbf{u}_t|\mathbf{x}_t)$ – time-varying linear-Gaussian controller

can **execute** on the robot!

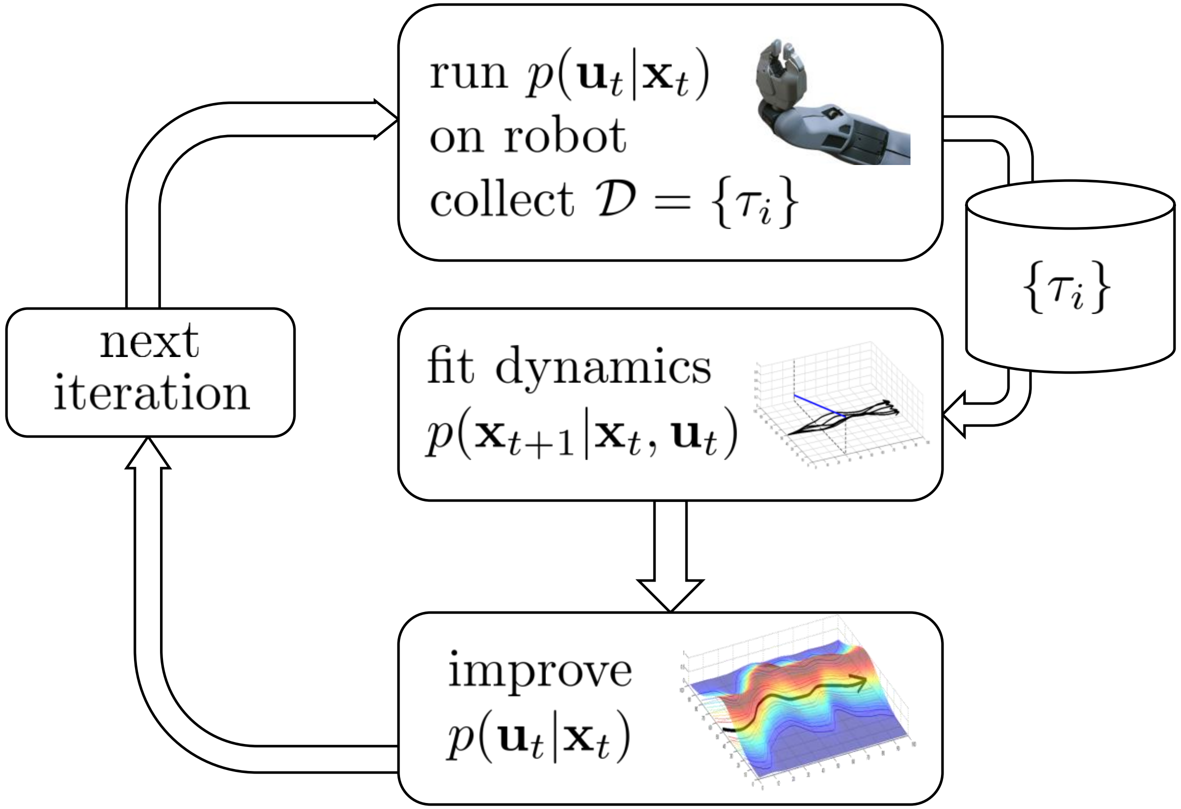


Local models

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

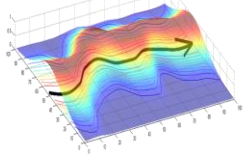
$$f(\mathbf{x}_t, \mathbf{u}_t) \approx \mathbf{A}_t \mathbf{x}_t + \mathbf{B}_t \mathbf{u}_t$$

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



What controller to execute?

improve
$p(\mathbf{u}_t|\mathbf{x}_t)$



iLQR produces: $\hat{\mathbf{x}}_t$, $\hat{\mathbf{u}}_t$, $\mathbf{K}_t$, $\mathbf{k}_t$

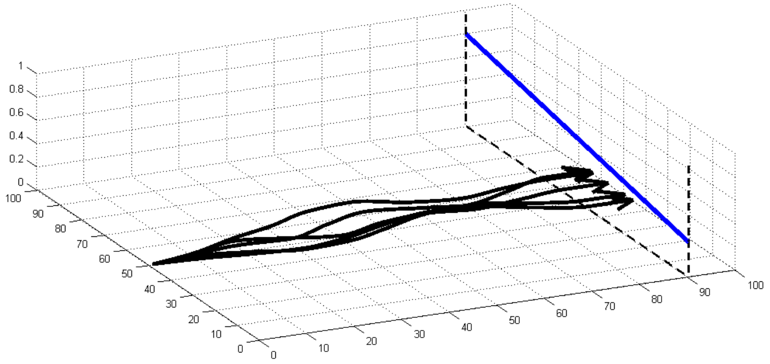
$$\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t$$

Version 0.5: $p(\mathbf{u}_t|\mathbf{x}_t) = \delta(\mathbf{u}_t = \hat{\mathbf{u}}_t)$

Doesn't correct deviations or drift

Version 1.0: $p(\mathbf{u}_t|\mathbf{x}_t) = \delta(\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t)$

Better, but maybe a little too good?



Version 2.0: $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$

Add noise so that all samples don't look the same!

What controller to execute?

Version 2.0: $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$

Set $\Sigma_t = \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1}$

$Q(\mathbf{x}_t, \mathbf{u}_t)$ is the cost to go: total cost we get after taking an action

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

$\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}$ is big if changing $\mathbf{u}_t$ changes the Q-value a lot!

If $\mathbf{u}_t$ changes Q-value a lot, don't vary $\mathbf{u}_t$ so much

Only act randomly when it minimally affects the cost to go

What controller to execute?

Version 2.0: $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$

Set $\Sigma_t = \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1}$

Standard LQR solves $\min \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t)$

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

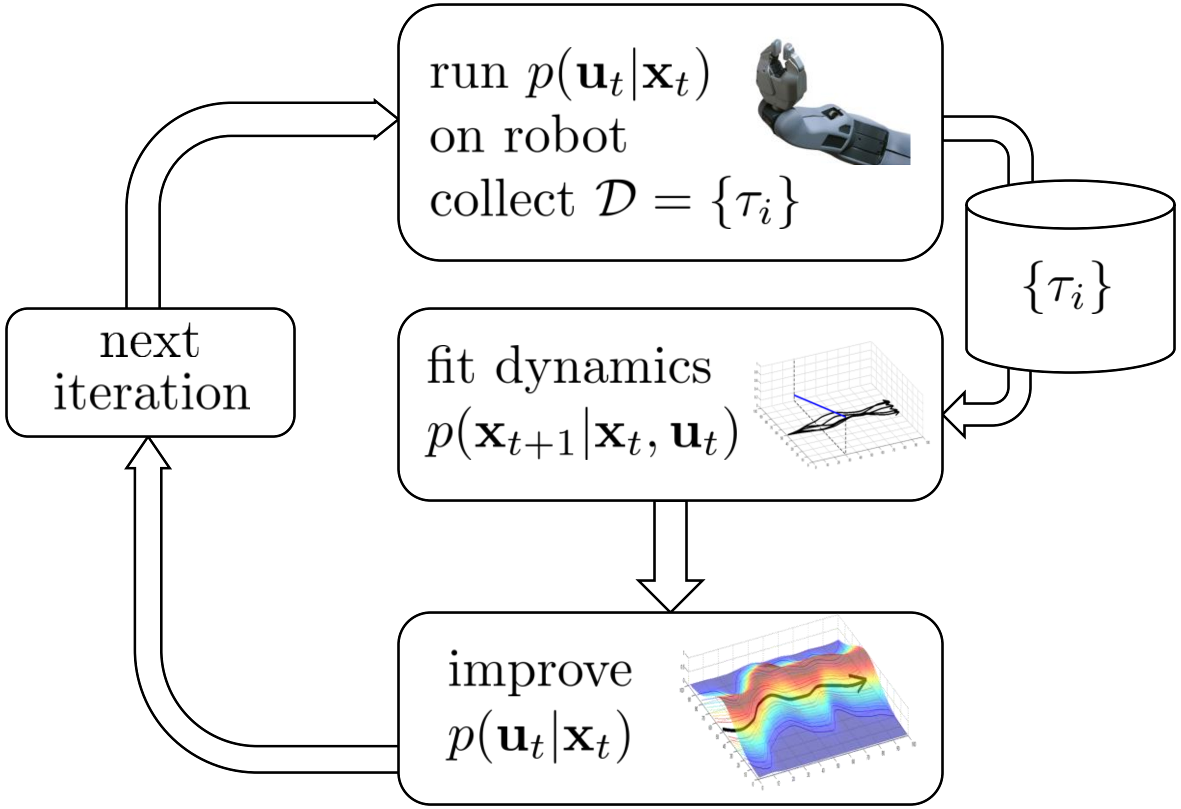
This is the *maximum entropy* solution: act as randomly as possible while minimizing cost

Local models

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

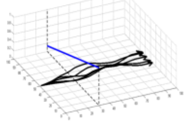
$$f(\mathbf{x}_t, \mathbf{u}_t) \approx \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t$$

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



How to fit the dynamics?

fit dynamics
$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$



$\{(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})_i\}$

Version 1.0: fit $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ at each time step using linear regression

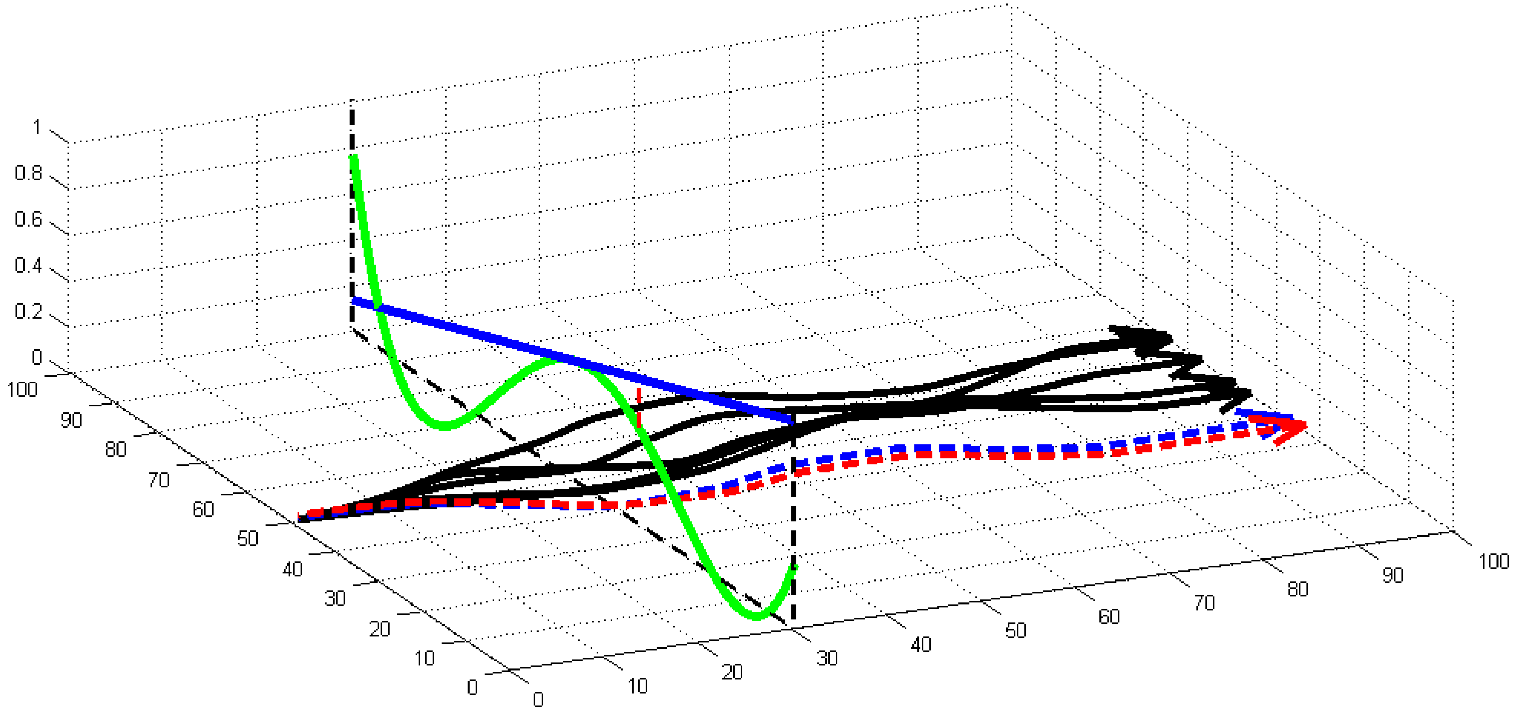
$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{A}_t \mathbf{x}_t + \mathbf{B}_t \mathbf{u}_t + \mathbf{c}, \mathbf{N}_t) \qquad \mathbf{A}_t \approx \frac{df}{d\mathbf{x}_t} \quad \mathbf{B}_t \approx \frac{df}{d\mathbf{u}_t}$$

Can we do better?

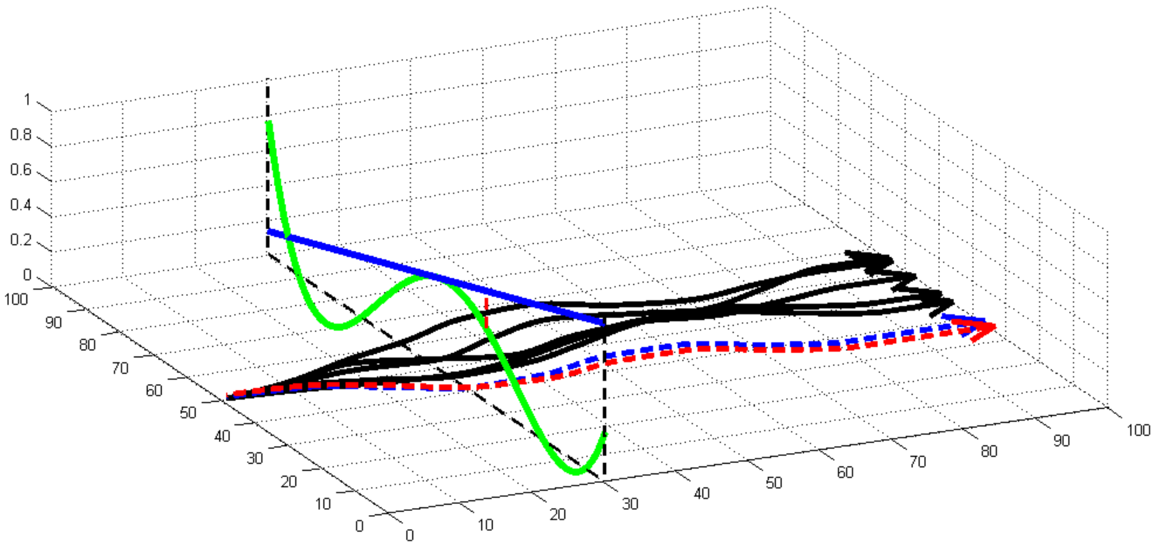
Version 2.0: fit $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ using *Bayesian* linear regression

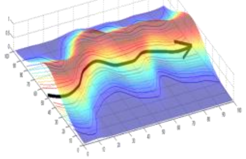
Use your favorite *global* model as prior (GP, deep net, GMM)

What if we go too far?



How to stay close to old controller?



improve $p(\mathbf{u}_t|\mathbf{x}_t)$ 

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

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

What if the new $p(\tau)$ is "close" to the old one $\bar{p}(\tau)$?

If trajectory distribution is close, then dynamics will be close too!

What does "close" mean? $D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon$

KL-divergences between trajectories

- Not just for trajectory optimization – really important for model-free policy search too! More on this in later lectures

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

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

dynamics & initial state are the same!

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

$$- \log p(\mathbf{x}_1) + \sum_{t=1}^{T} -\log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \log p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)$$

KL-divergences between trajectories

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

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

$$- \log p(\mathbf{x}_1) + \sum_{t=1}^{T} - \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$$

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

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

KL-divergences between trajectories

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

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

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

KL-divergences between trajectories

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

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

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

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

But how?

We want a constraint: $D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon$

Digression: dual gradient descent

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

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

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

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

how to maximize? Compute the gradient!

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

this is the hard part, everything else is easy!

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

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

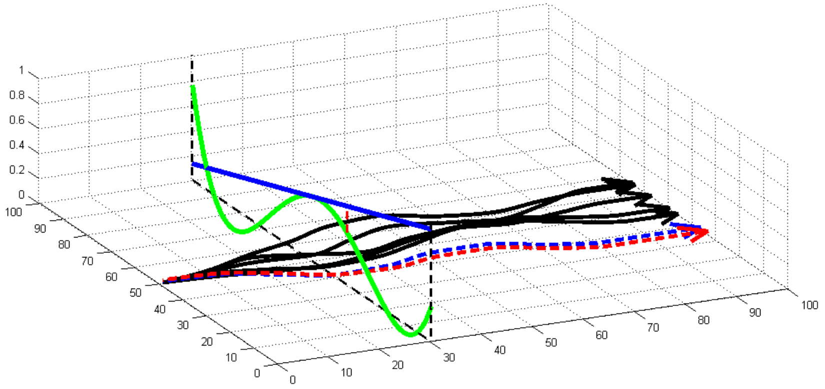
$$\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)} \left[\frac{1}{\lambda} c(\mathbf{x}_t, \mathbf{u}_t) - \log \bar{p}(\mathbf{u}_t|\mathbf{x}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))\right]$$

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. $\lambda \leftarrow \lambda + \alpha(D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) - \epsilon)$



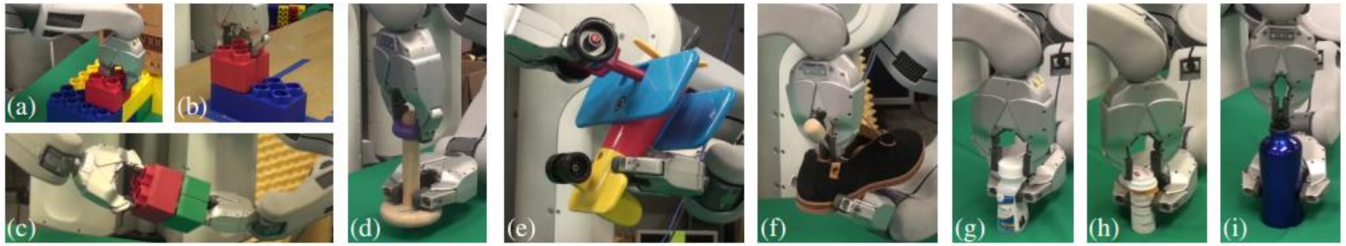
Trust regions & trajectory distributions

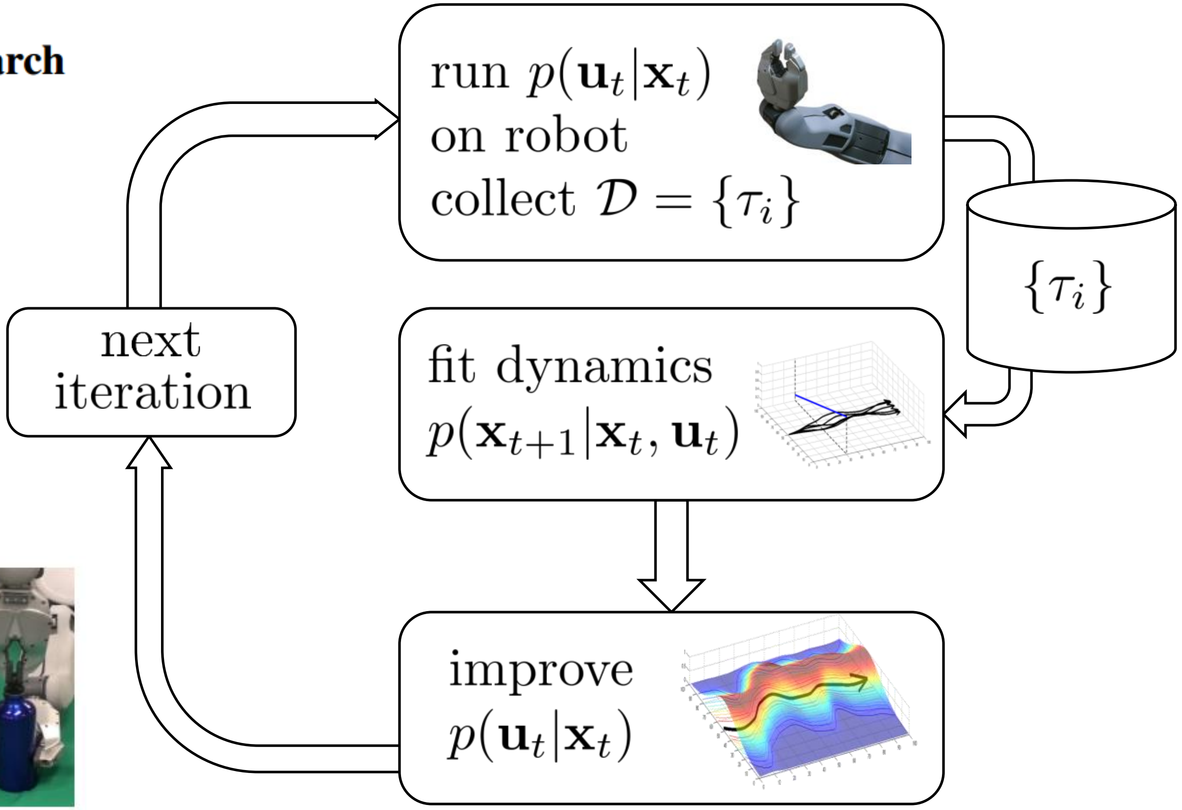
- Bounding KL-divergences between two policies or controllers, whether linear-Gaussian or more complex (e.g. neural networks) is really useful
- Bounding KL-divergence between policies is equivalent to bounding KL-divergences between trajectory distributions
- We'll use this later in the course in model-free RL too!

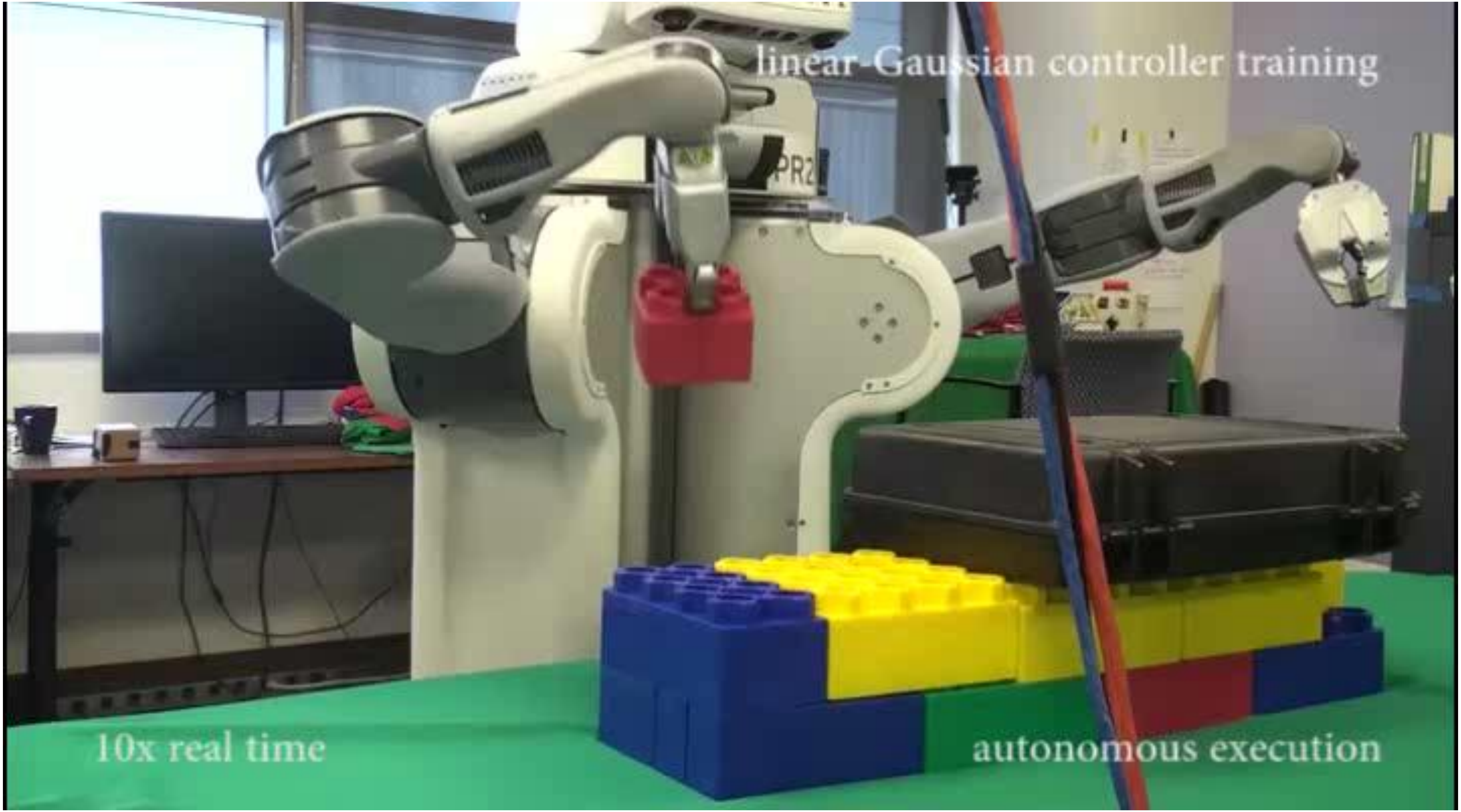
Case study: local models & iterative LQR

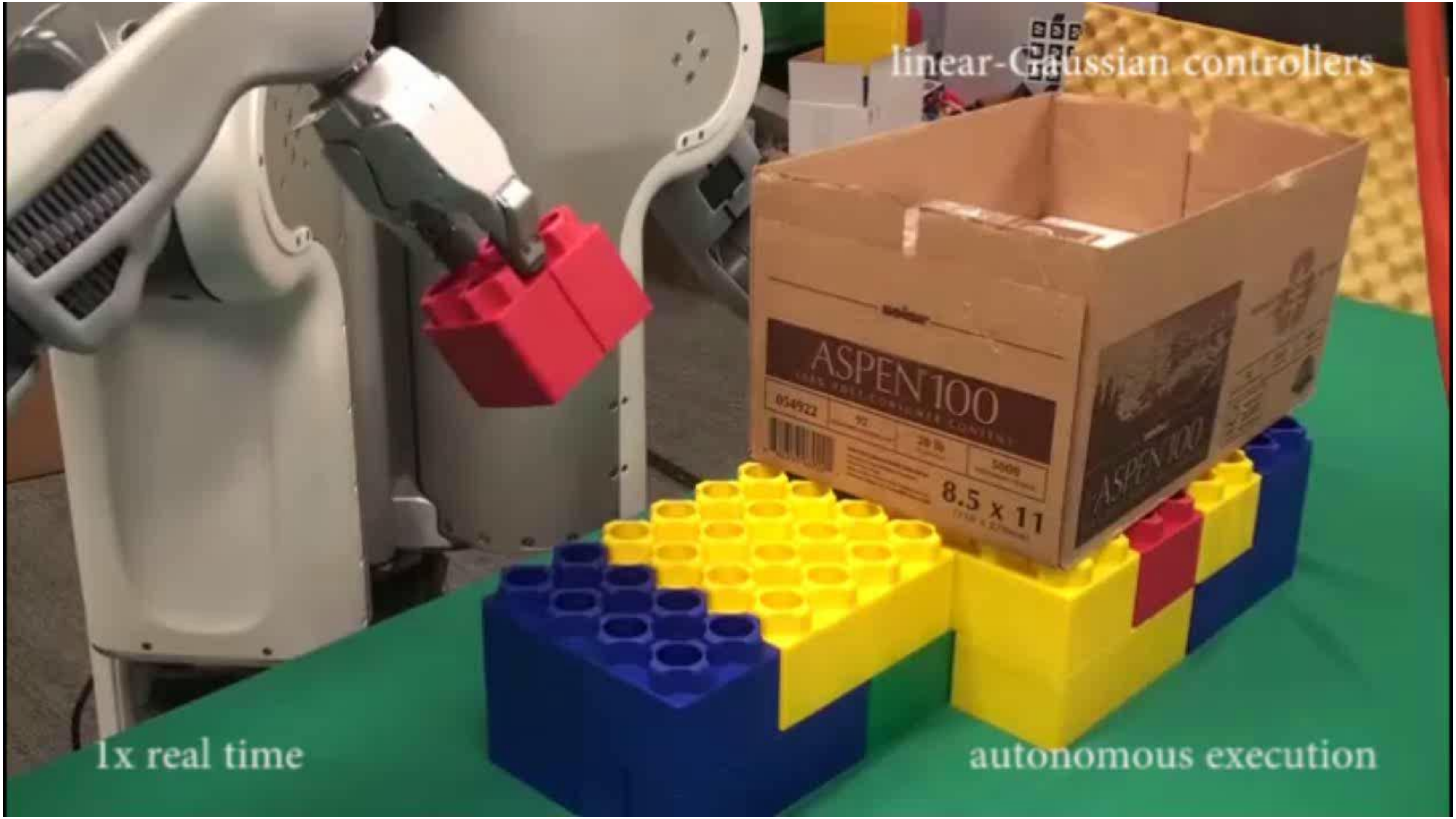
Learning Contact-Rich Manipulation Skills with Guided Policy Search

Sergey Levine, Nolan Wagener, Pieter Abbeel

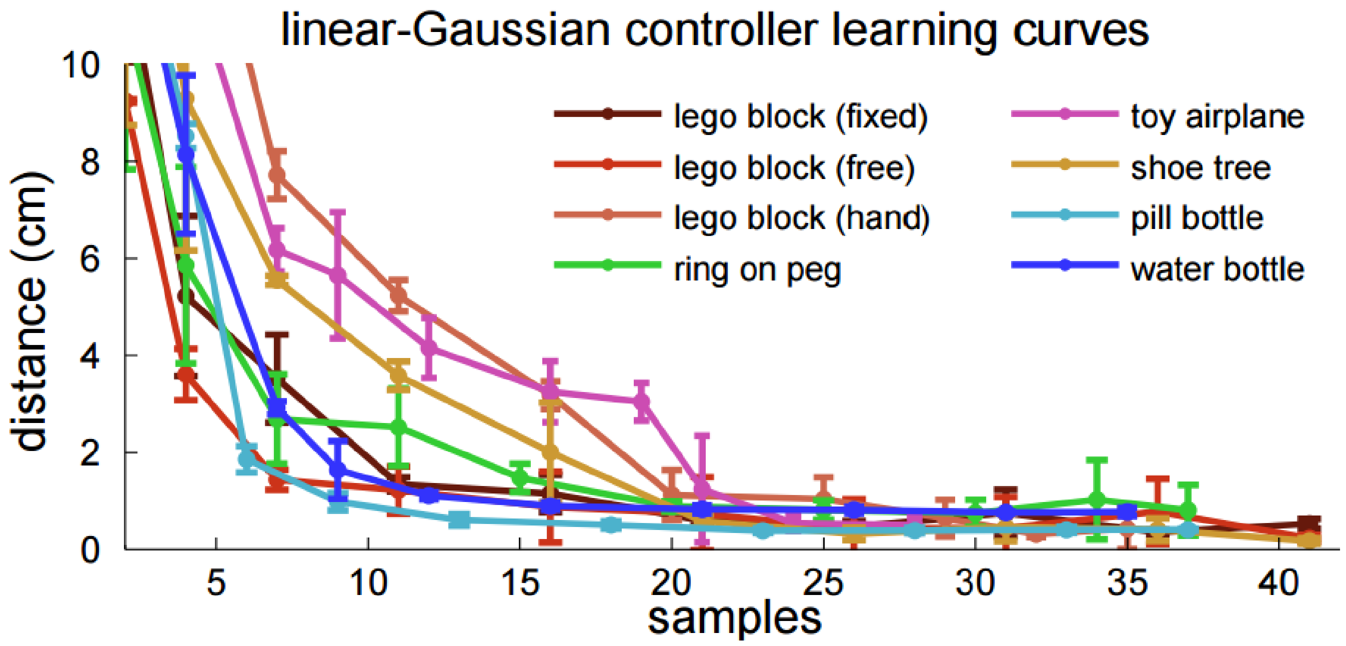


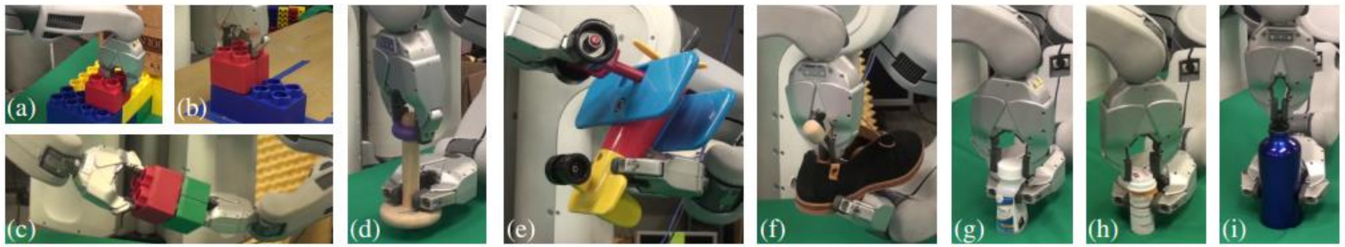



linear-Gaussian controller training
10x real time
autonomous execution


linear-Gaussian controllers
ASPEN 100
8.5 x 11
1x real time
autonomous execution

Case study: local models & iterative LQR

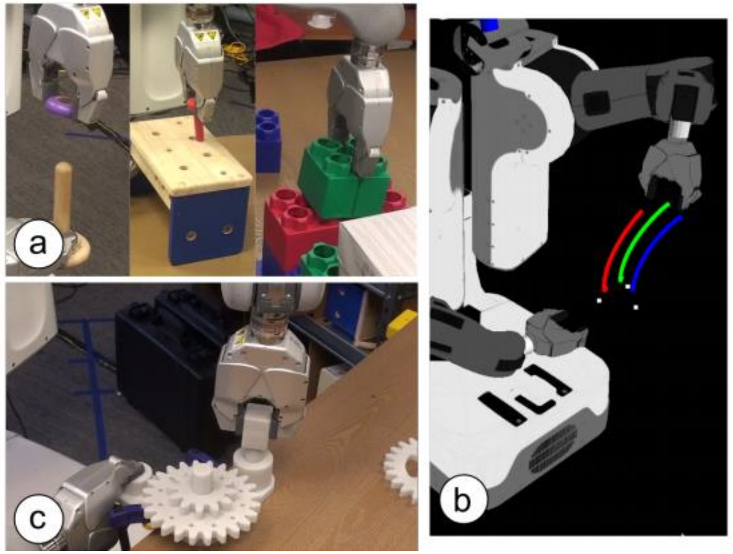


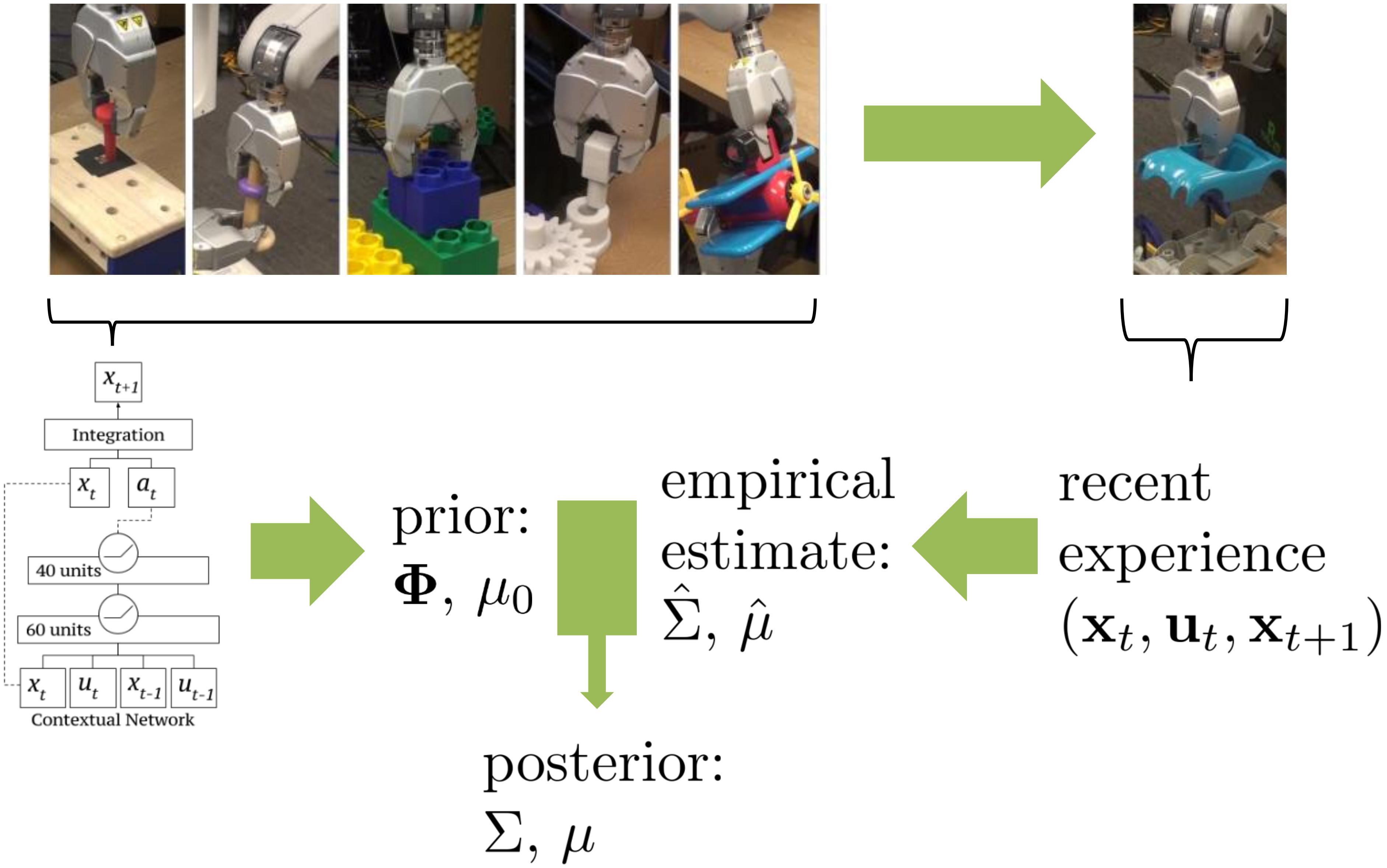


Case study: combining global and local models

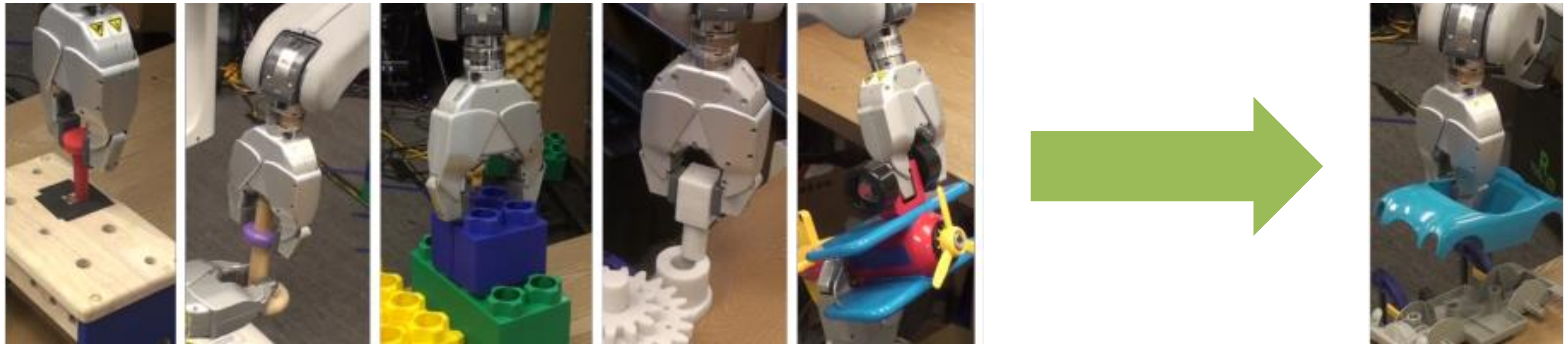
One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors

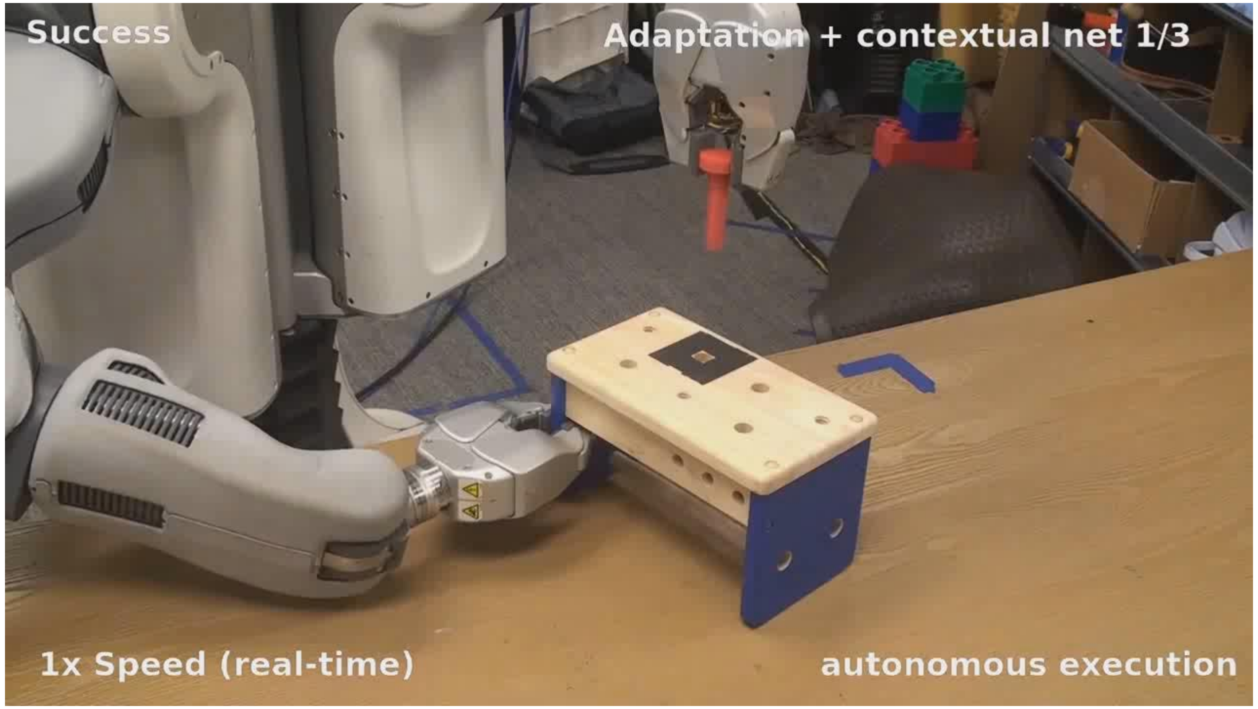
Justin Fu, Sergey Levine, Pieter Abbeel





x_{t+1}
Integration
x_t
a_t
40 units
60 units
x_t
u_t
x_{t-1}
u_{t-1}
Contextual Network
prior:
$\mathbf{\Phi}, \ \mu_0$
empirical estimate:
$\hat{\Sigma}, \ \hat{\mu}$
recent experience
$(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$
posterior:
$\Sigma, \ \mu$




Success
Adaptation + contextual net 1/3
1x Speed (real-time)
autonomous execution