

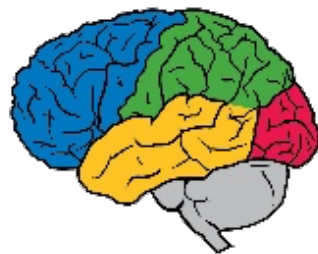
Deep
^

Model-Based Reinforcement Learning

Chelsea Finn



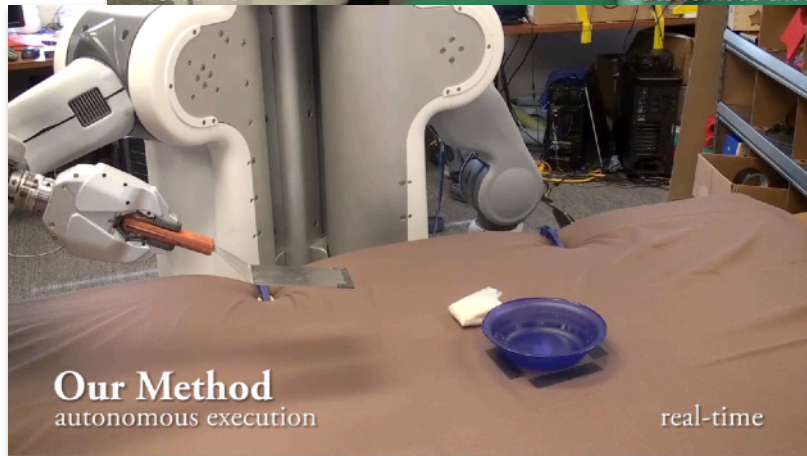
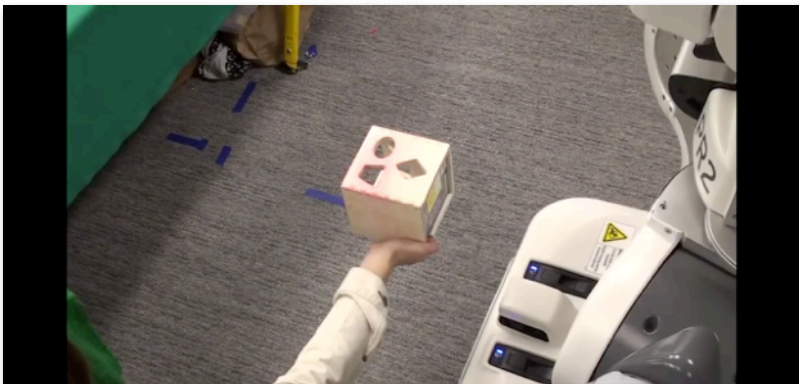
UC Berkeley



Google Brain



Stanford



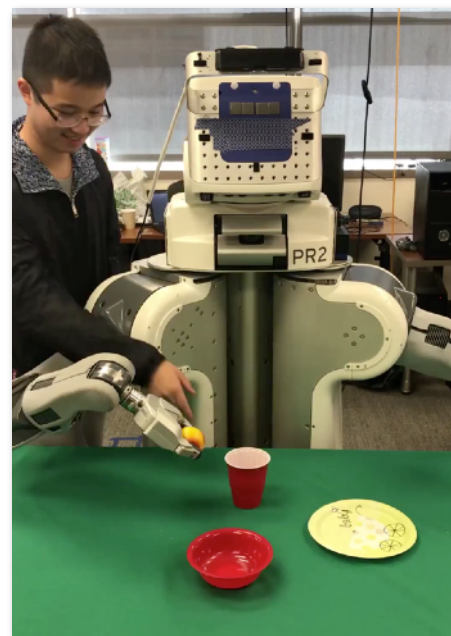
Levine*, Finn*, Darrell, Abbeel, JMLR '16
Finn, Tan, Duan, Darrell, Levine, Abbeel, ICRA '16



Finn, Goodfellow, Levine, NIPS '16
Finn & Levine, ICRA '17



Ebert*, Finn*, Dasari, Xie,
Lee, Levine, '18



Yu*, Finn*, Xie, Dasari, Zhang, Abbeel, Levine, RSS '18



Nagabandi, Clavera, Liu, Fearing,
Abbeel, Levine, Finn, '18

Outline

1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
3. Using local models & guided policy search
4. Handling high-dimensional observations

Outline

- 1. Why use model-based reinforcement learning?**
2. Main model-based RL approaches
3. Using local models & guided policy search
4. Handling high-dimensional observations

Why use model-based reinforcement learning?

- a model enables you to plan
- sample efficiency

RL approaches

gradient-free methods
(e.g. NES, CMA, etc.)



fully online methods
(e.g. A3C)



policy gradient methods
(e.g. TRPO)



replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)



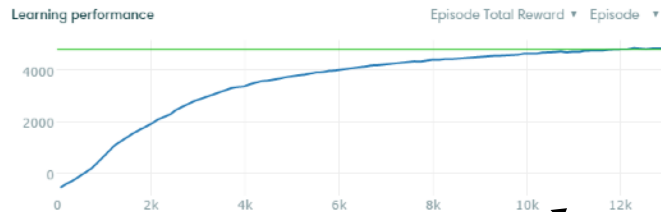
model-based deep RL
(e.g. guided policy search)



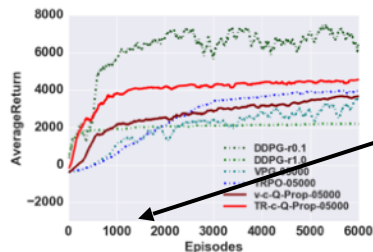
model-based "shallow" RL
(e.g. PILCO)

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans¹ Jonathan Ho¹ Xi Chen¹ Ilya Sutskever¹

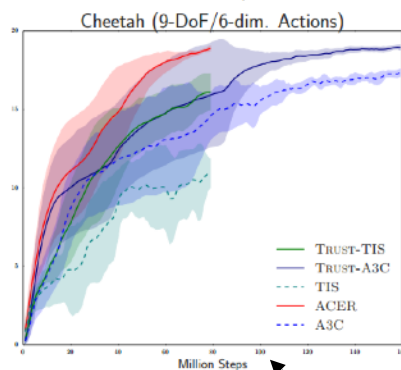


TRPO+GAE (Schulman et al. '16)



Gu et al. '16

	cart-pole	cart-double-pole	unicycle
state space	\mathbb{R}^4	\mathbb{R}^6	\mathbb{R}^{12}
# trials	≤ 10	20-30	≈ 20
experience	≈ 20 s	≈ 60 s-90 s	≈ 20 s-30 s
parameter space	\mathbb{R}^{305}	\mathbb{R}^{1816}	\mathbb{R}^{28}



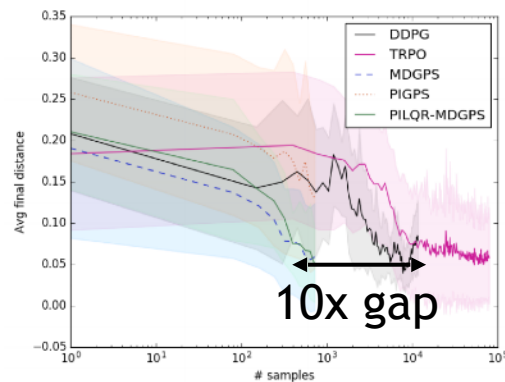
Wang et al. '17



100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)



Chebatar et al. '17 (note log scale)

about 20 minutes of
experience on a real
robot

Why use model-based reinforcement learning?

- a model enables you to plan
- sample efficiency
- transferability & generality

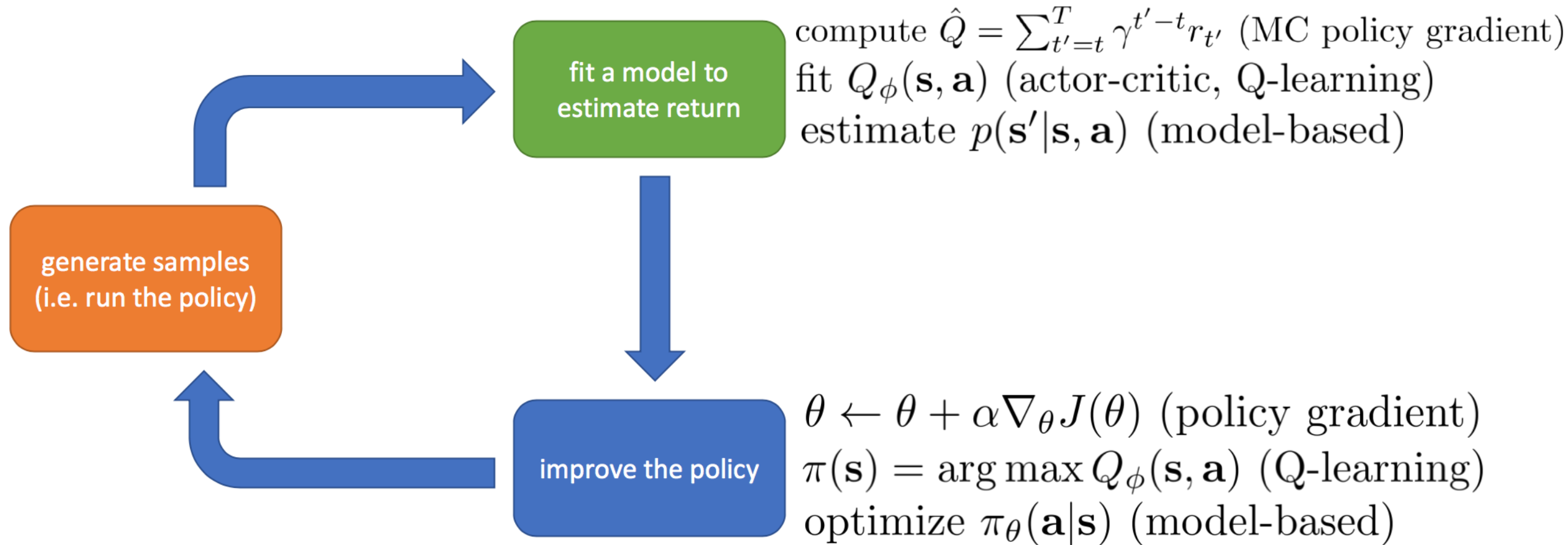
A model can be reused for achieving different tasks.

[more examples later]

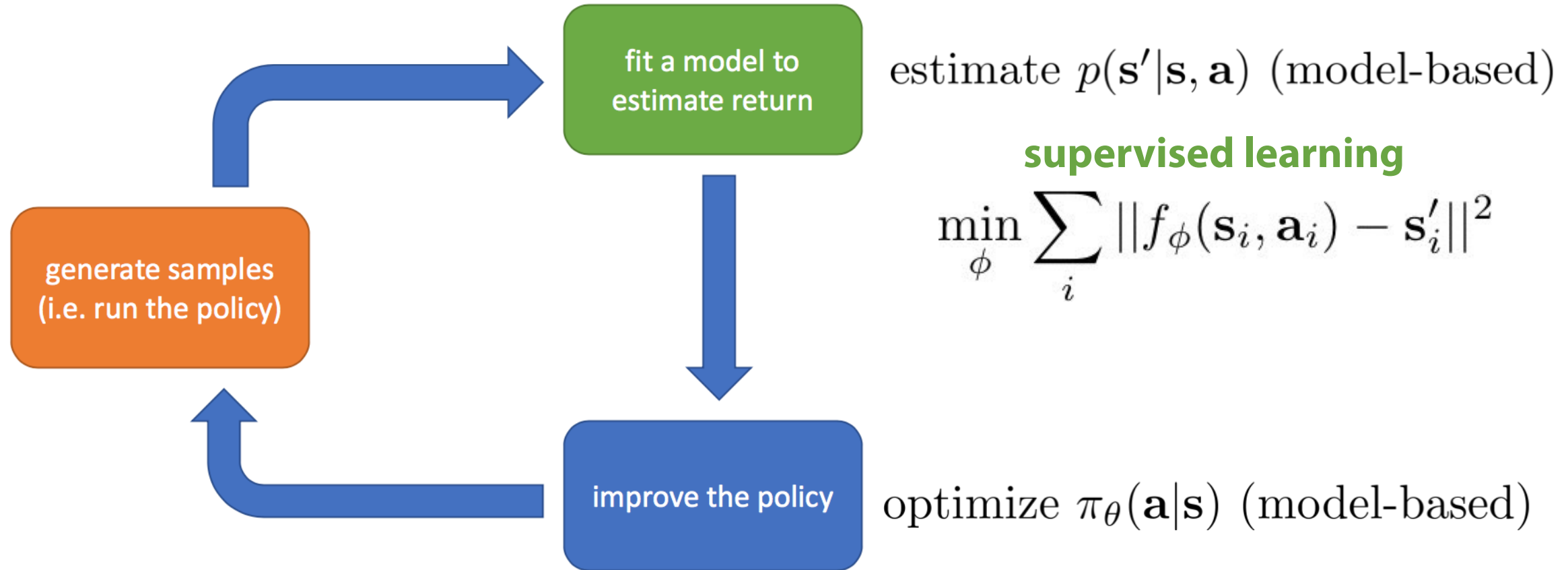
Outline

1. Why use model-based reinforcement learning?
- 2. Main model-based RL approaches**
3. Using local models & guided policy search
4. Handling high-dimensional observations

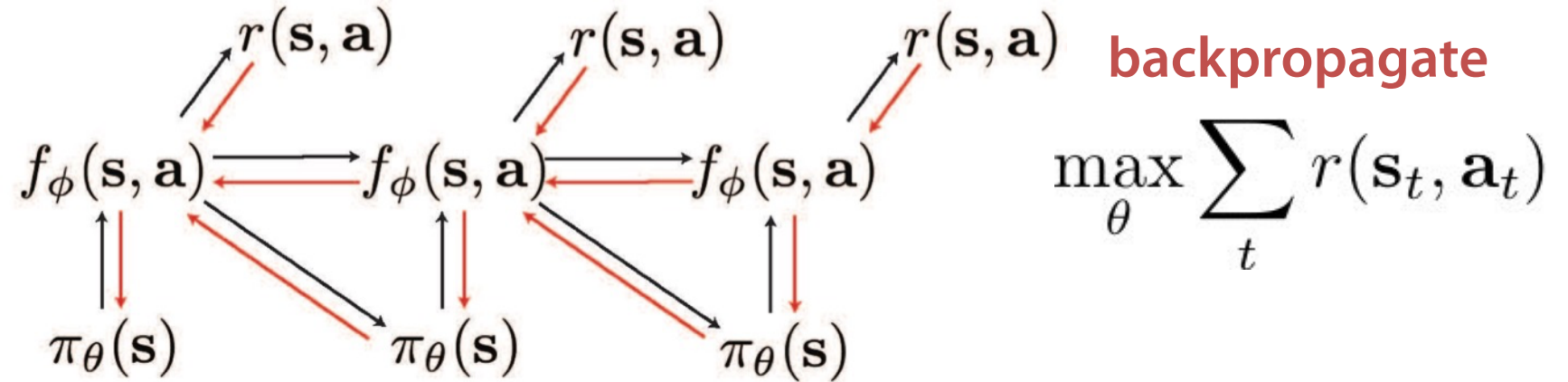
The Anatomy of a Reinforcement Learning Problem



Model-Based Reinforcement Learning



Backprop through model to optimize policy



Algorithm v0:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn model $f_{\phi}(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f_{\phi}(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. backpropagate through $f_{\phi}(\mathbf{s}, \mathbf{a})$ to choose actions.
or into policy to optimize $\pi_{\theta}(\mathbf{a}_t|\mathbf{s}_t)$.

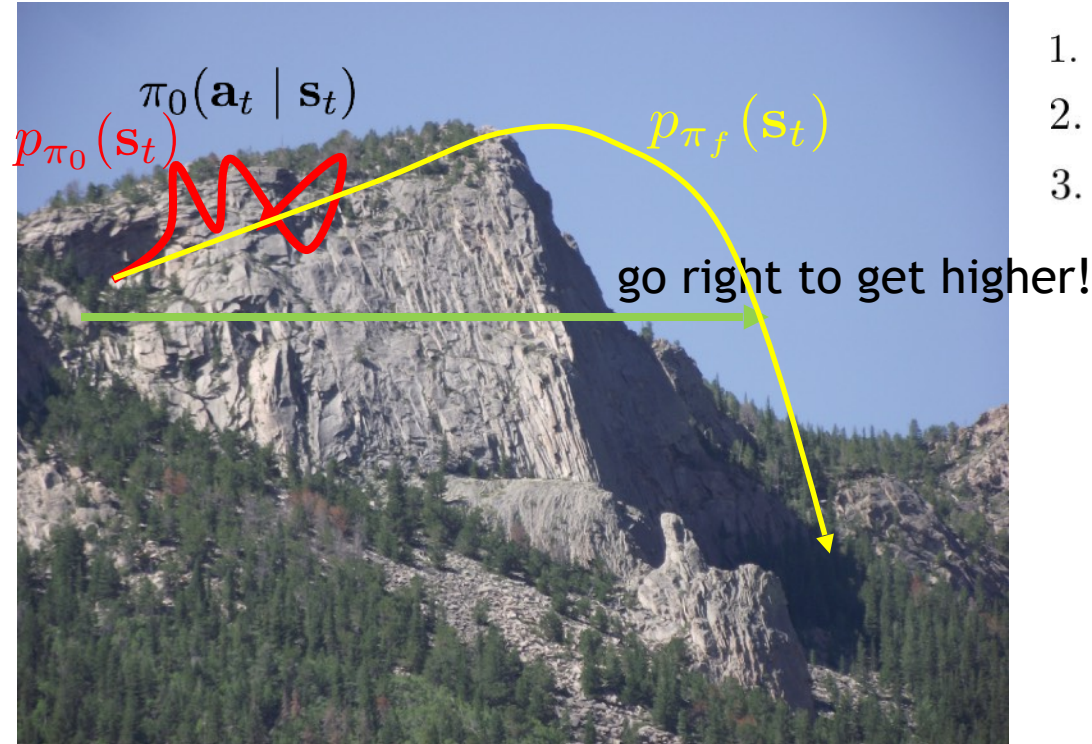
Does it work?

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 $\pi_0(\mathbf{a}_t | \mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn model $f_\phi(\mathbf{s}, \mathbf{a})$ to minimize $\sum \|f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. backpropagate through $f_\phi(\mathbf{s}, \mathbf{a})$ into policy to optimize $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$

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


- **State distribution mismatch**, problem becomes exacerbated as we use more expressive model classes

Can we do better?

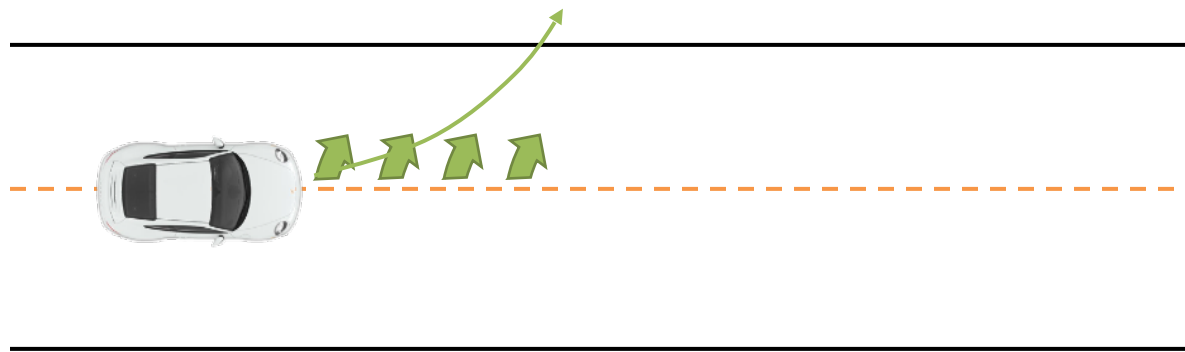
can we make $p_{\pi_0}(\mathbf{s}_t) = p_{\pi_f}(\mathbf{s}_t)$?

need to collect data from $p_{\pi_f}(\mathbf{s}_t)$

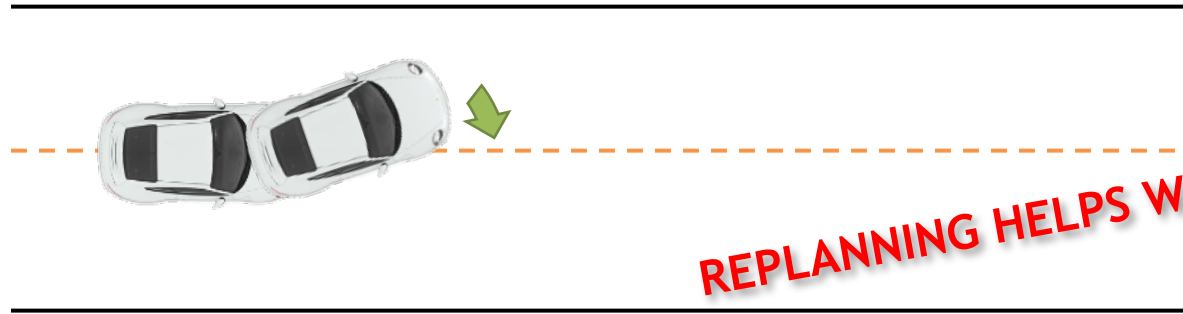
Algorithm v1:

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

What if we make a mistake?



Can you correct the mistake?



REPLANNING HELPS WITH MODEL ERRORS

Algorithm v2a:

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

every N steps

model-predictive control (MPC)

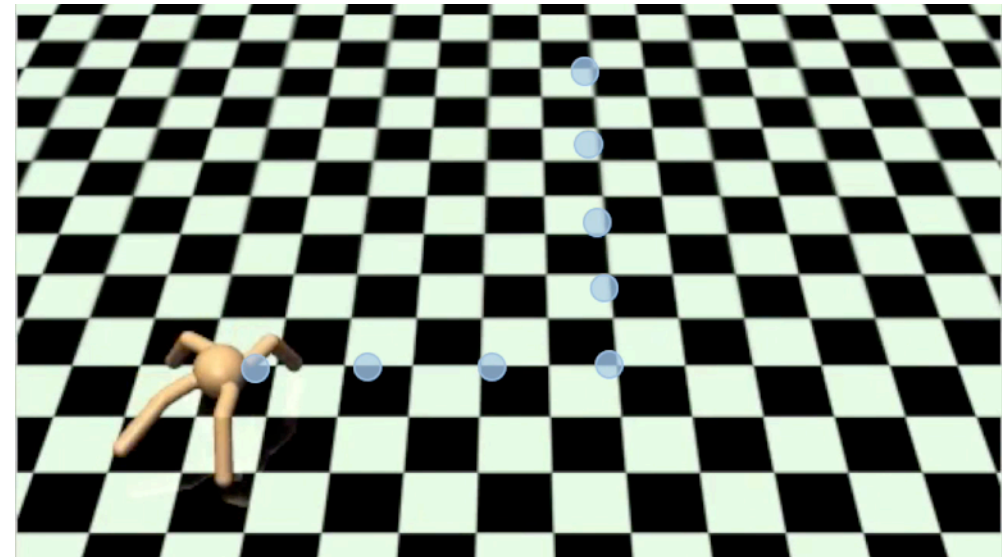
An alternative way to choose actions

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

every N steps

Can instead sample to choose actions:

- A. Sample action sequences from some distribution (e.g. uniformly at random)
- B. Run actions through model to prediction future
- C. Choose action leading to the best future



Summary so far

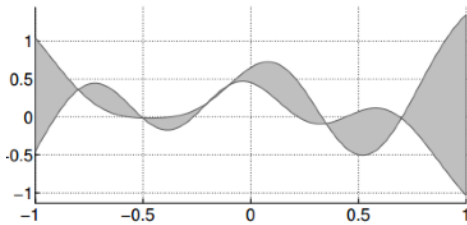
- **Version 0: collect random samples, train dynamics, plan**
 - **Pro:** simple, no iterative procedure
 - **Con:** distribution mismatch problem
- **Version 1: iteratively collect data, refit model**
 - **Pro:** simple, solves distribution mismatch
 - **Con:** still might make mistakes with imperfect model
- **Version 2: iteratively collect data using MPC (replan at each step)**
 - **Pro:** robust to small model errors
 - **Con:** computationally expensive, but have a planning algorithm available

Two ways to optimize policy w.r.t. model:

- backprop through model into policy
- sampling-based optimization

What kind of models can we use?

Gaussian process



GP with input (\mathbf{s}, \mathbf{a}) and output \mathbf{s}'

Pro: very data-efficient

Con: not great with non-smooth dynamics

Con: very slow when dataset is big

neural network

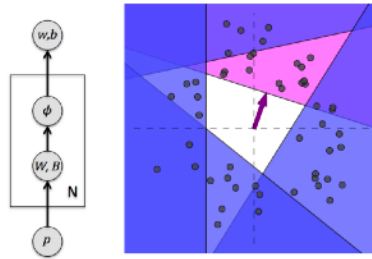


image: Punjani & Abbeel '14

Input is (\mathbf{s}, \mathbf{a}) and output is \mathbf{s}'

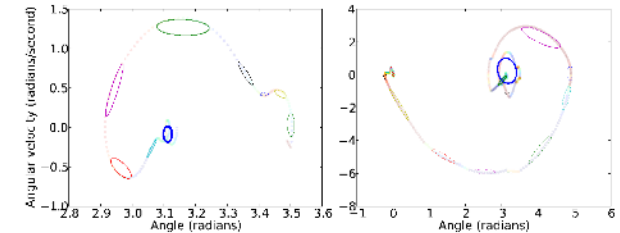
Euclidean training loss corresponds to Gaussian $p(\mathbf{s}' | \mathbf{s}, \mathbf{a})$

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 $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ tuples

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

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

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



video prediction?
more on this later

Outline

1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
- 3. Using local models & guided policy search**
4. Handling high-dimensional observations

The trouble with global models

Global model: $f_\phi(\mathbf{s}_t, \mathbf{a}_t)$ represented by a big neural network

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$

2. learn model $f_\phi(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$

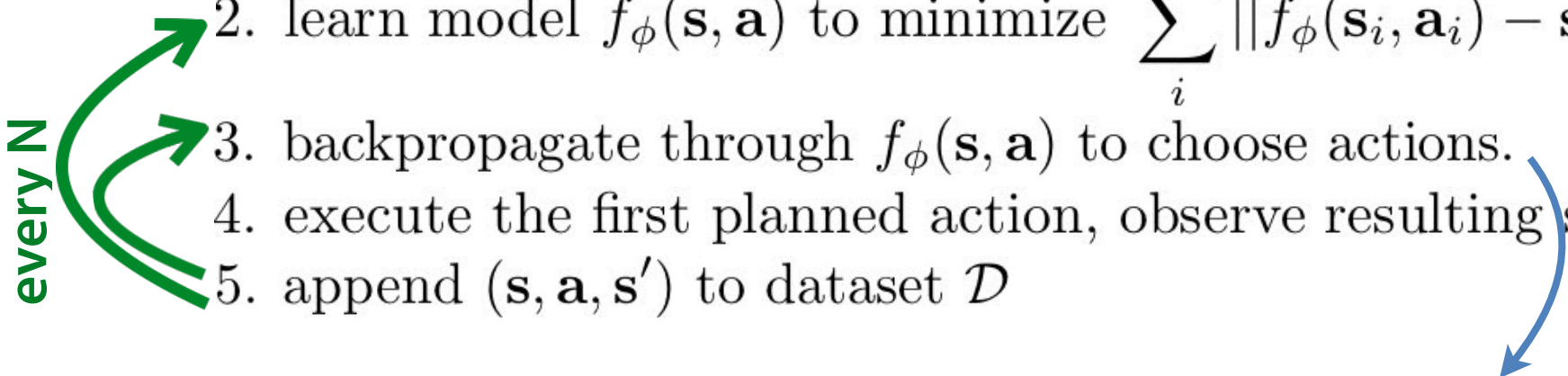
3. backpropagate through $f_\phi(\mathbf{s}, \mathbf{a})$ into policy to optimize $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

4. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ 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

Do we need to model everything?

What if we know where our model is good and where it is bad?
i.e., model uncertainty

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

Take actions that lead to high reward *in expectation*.

helps avoid model exploitation ***Caveat:** still need to explore

To get model uncertainty:

- Gaussian Processes
- Bayesian neural networks
- Bootstrap ensembles

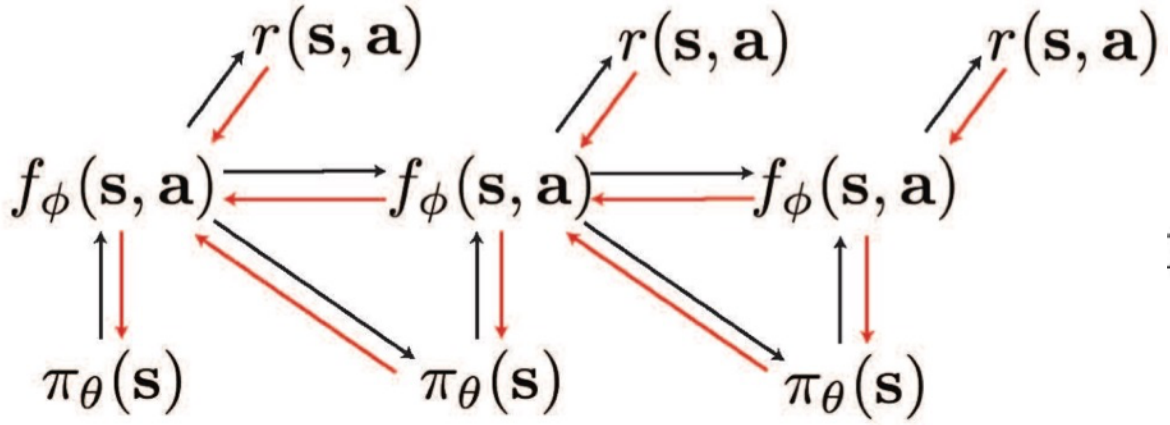
$$p(\phi|\mathcal{D})$$

Do we need to model everything?

In some tasks, the **model** is much more complex than the **policy**



Local models



backpropagate

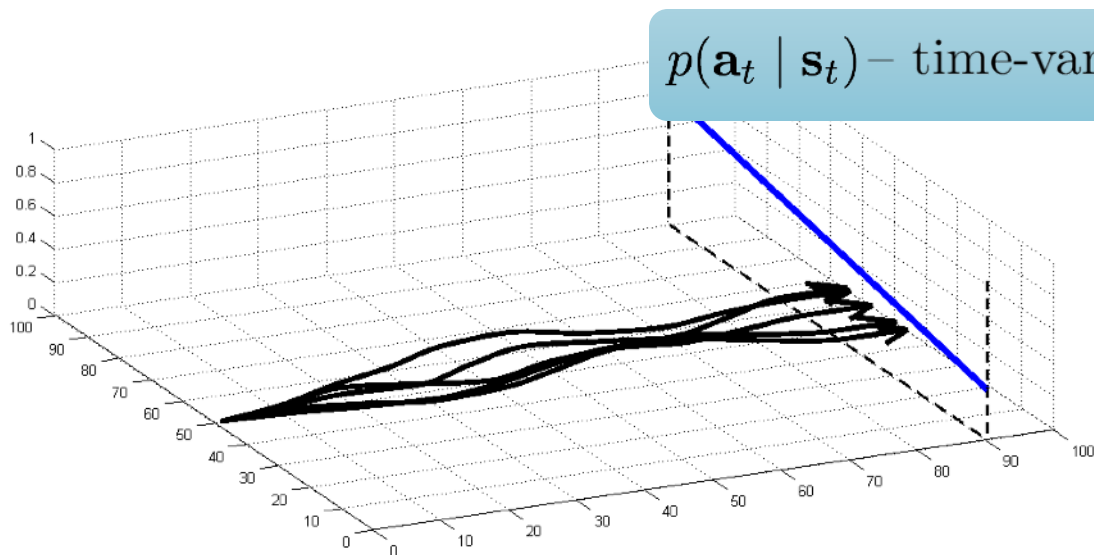
$$\max_{\theta} \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$$

need $\frac{df}{ds_t}, \frac{df}{da_t}, \frac{dr}{ds_t}, \frac{dr}{da_t}$

Local models

need $\frac{df}{ds_t}, \frac{df}{da_t}, \frac{dr}{ds_t}, \frac{dr}{da_t}$

idea: just fit $\frac{df}{ds_t}, \frac{df}{da_t}$, around current trajectory or policy!



can **execute** on the robot!

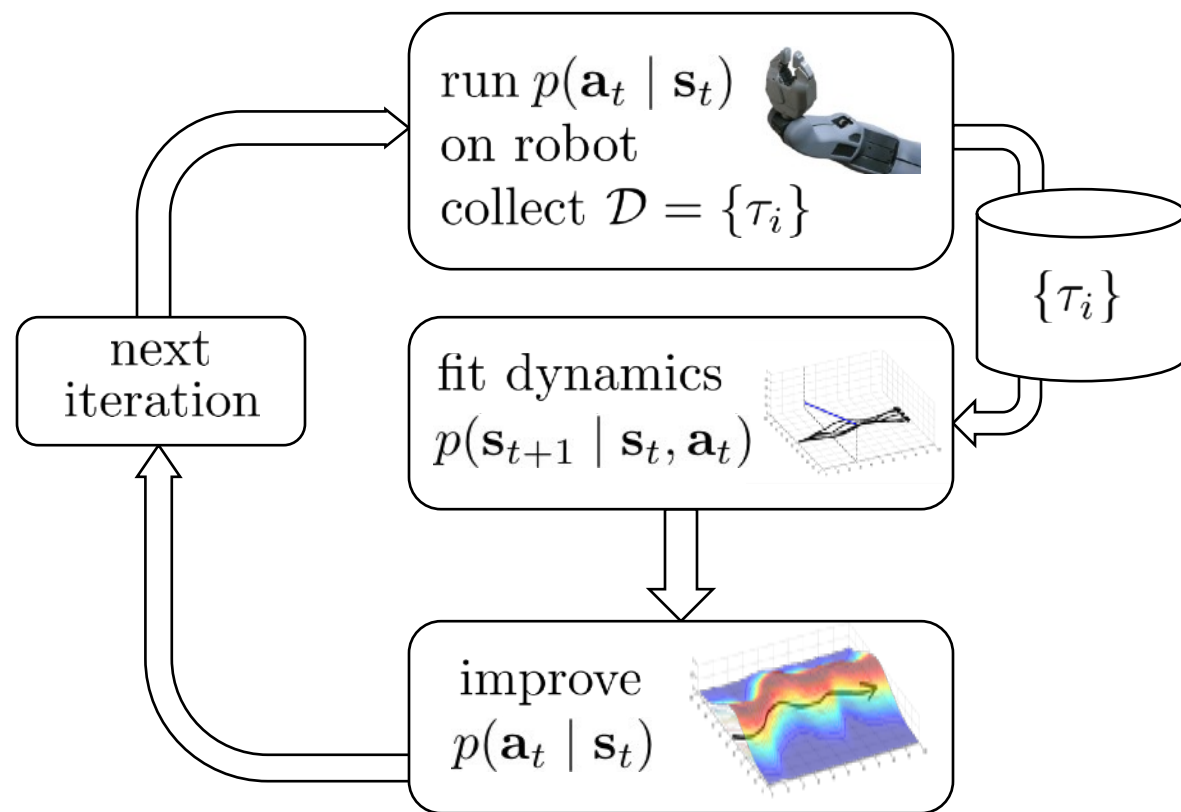
“local policy”

Local models

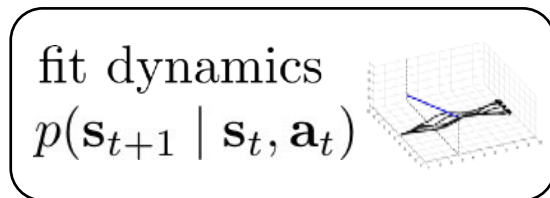
$$p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) = \mathcal{N}(f(\mathbf{s}_t, \mathbf{a}_t), \Sigma)$$

$$f(\mathbf{s}_t, \mathbf{a}_t) \approx \mathbf{A}_t \mathbf{s}_t + \mathbf{B}_t \mathbf{a}_t$$

$$\mathbf{A}_t = \frac{df}{d\mathbf{s}_t} \quad \mathbf{B}_t = \frac{df}{d\mathbf{a}_t}$$



How to fit the dynamics?



$$\{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})_i\}$$

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

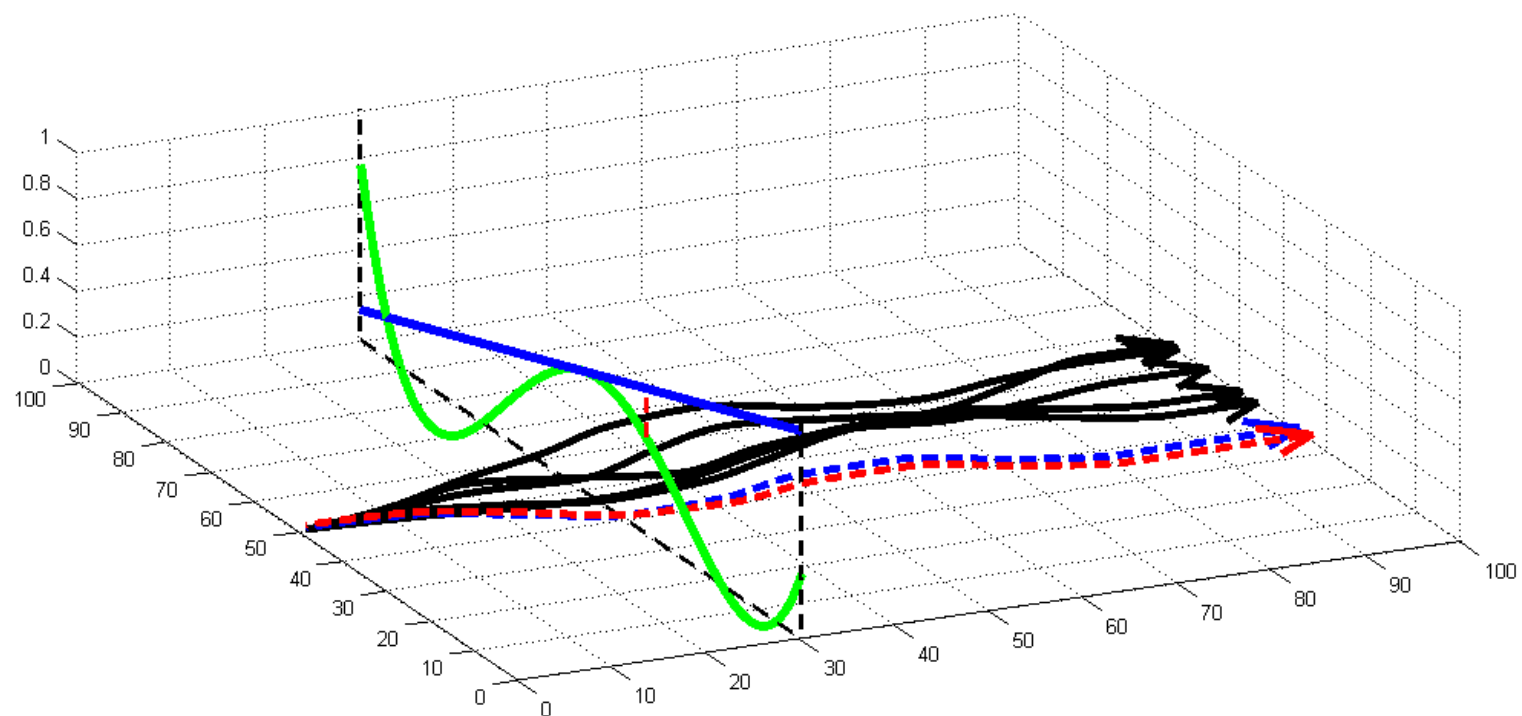
$$p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) = \mathcal{N}(\mathbf{A}_t \mathbf{s}_t + \mathbf{B}_t \mathbf{a}_t + \mathbf{c}_t, \mathbf{N}_t) \quad \mathbf{A}_t \approx \frac{df}{d\mathbf{s}_t} \quad \mathbf{B}_t \approx \frac{df}{d\mathbf{a}_t}$$

Can we do better?

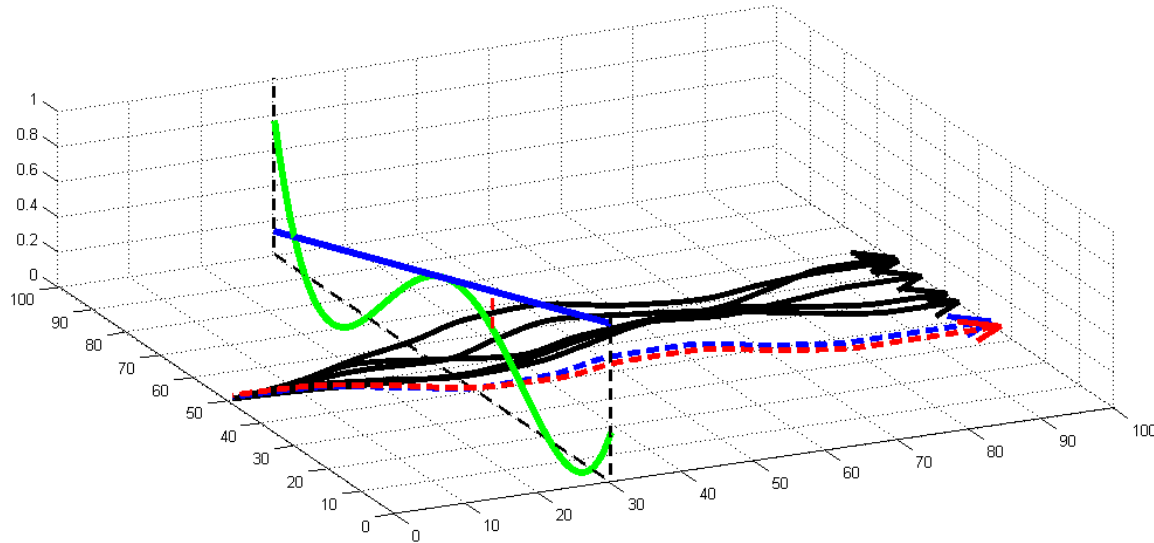
Version 2.0: fit $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ using *Bayesian* linear regression

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
 $p(\mathbf{a}_t | \mathbf{s}_t)$

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

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

Local Models Approach Summary

Levine & Abbeel NIPS '14

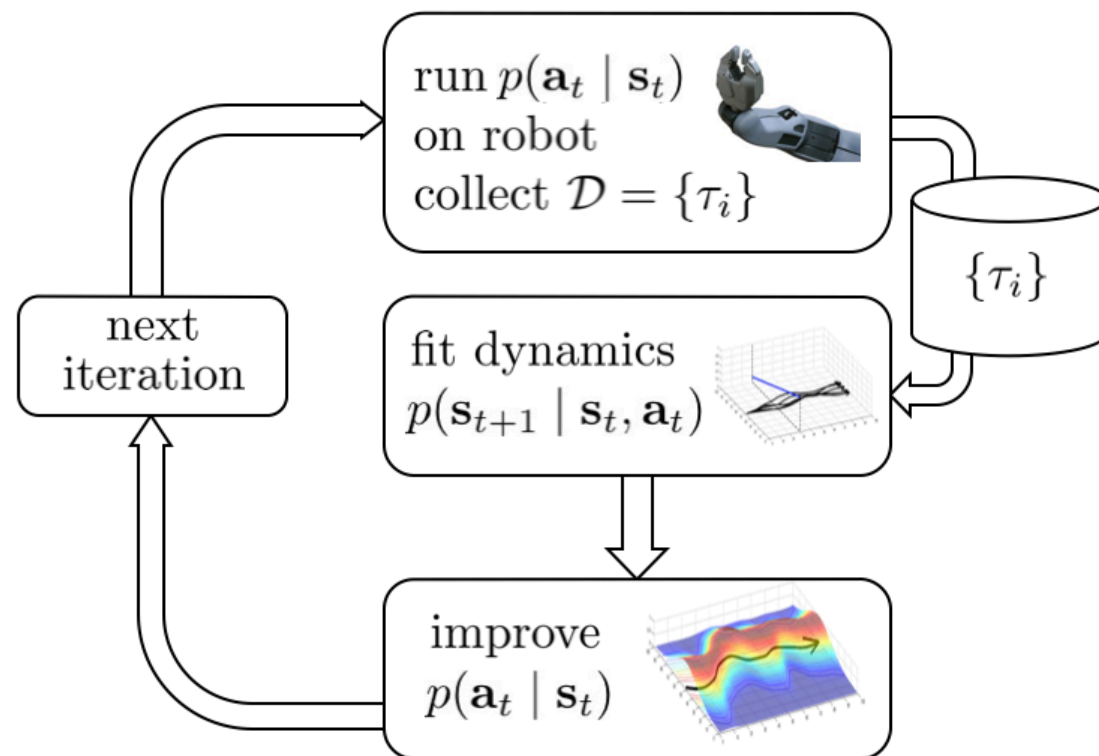
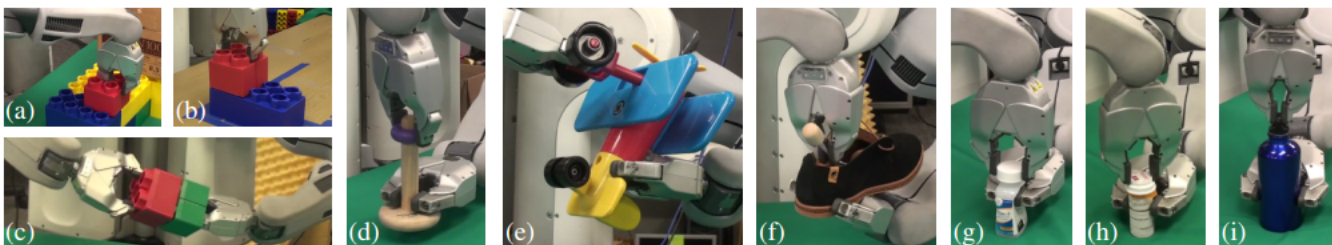
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn local model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
e.g. using linear regression
3. update local policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ using local model f_ϕ with KL constraint.
4. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, putting visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ in \mathcal{D}

e.g. using iterative LQR

Case study: local models & iterative LQR

Learning Contact-Rich Manipulation Skills with Guided Policy Search

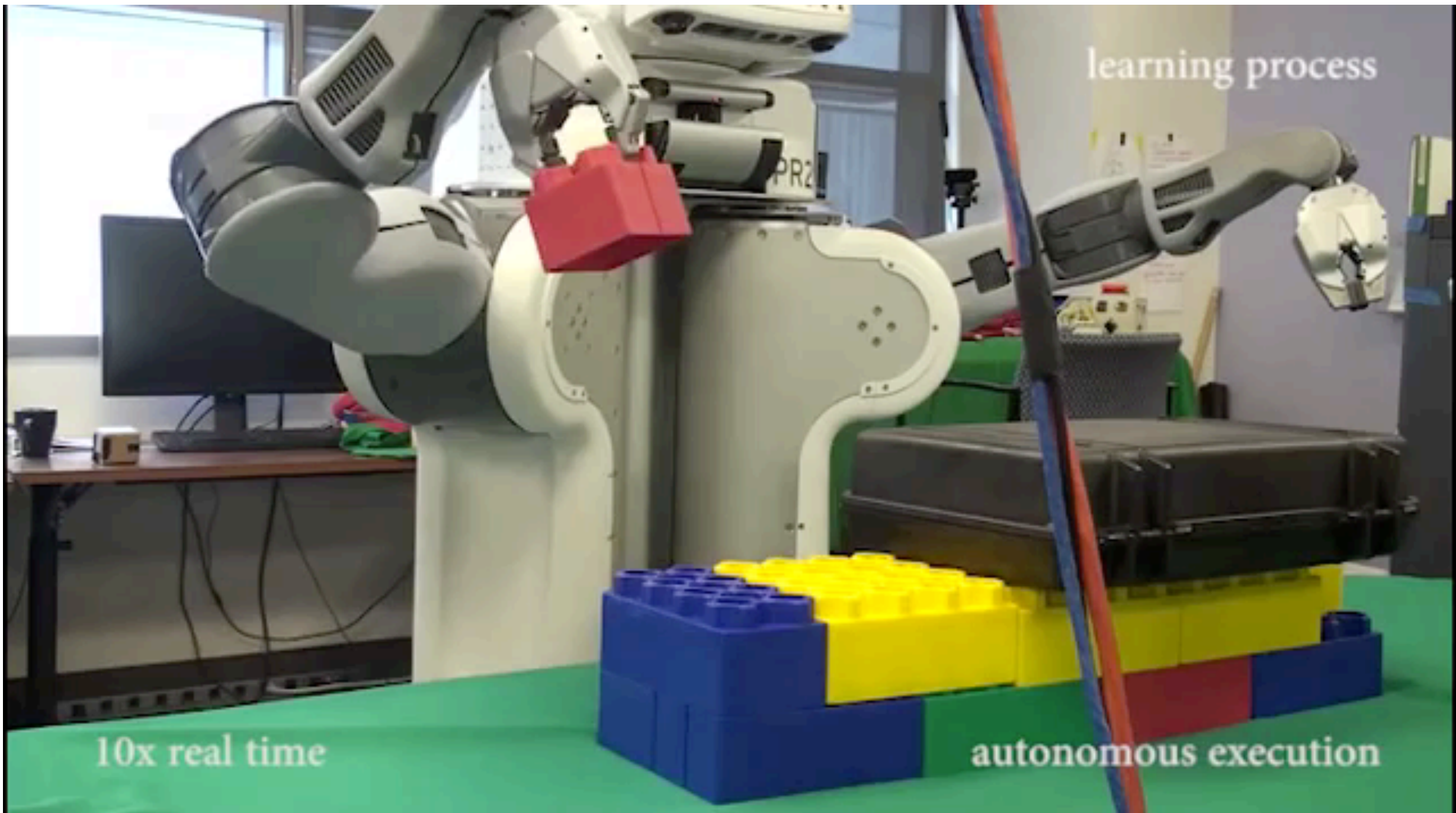
Sergey Levine, Nolan Wagener, Pieter Abbeel



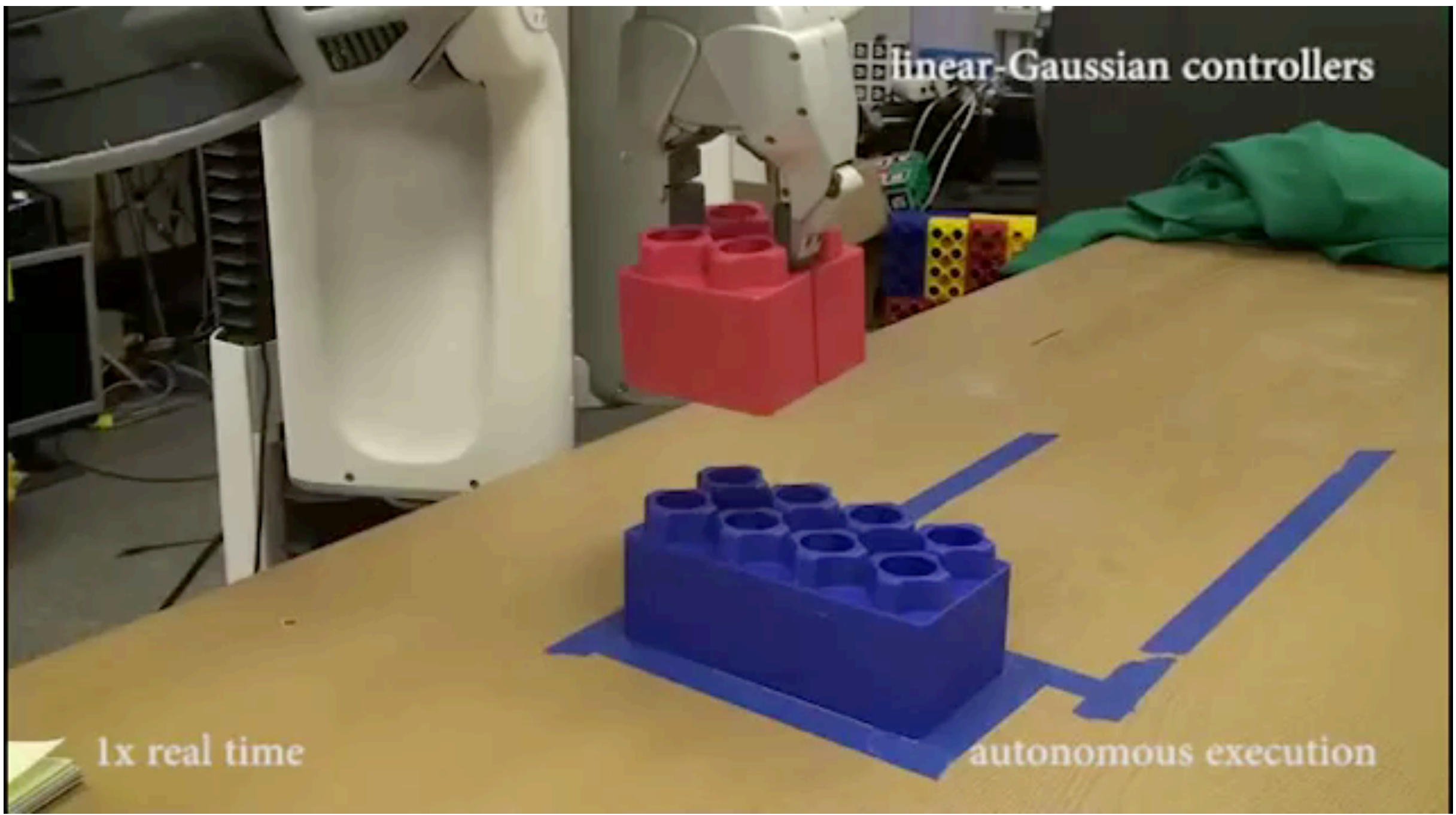
learning process

10x real time

autonomous execution



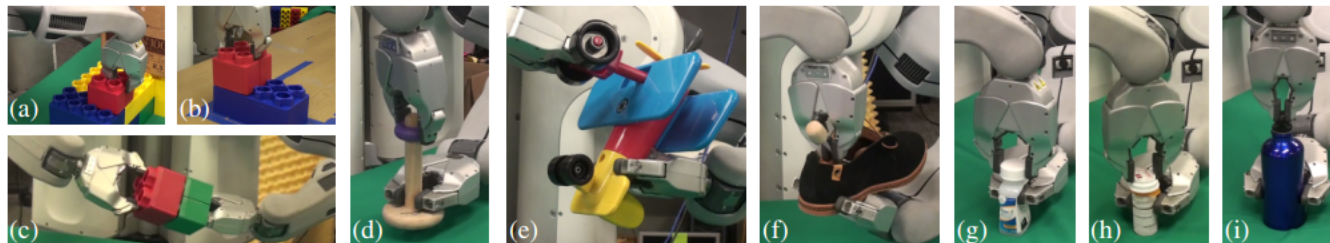
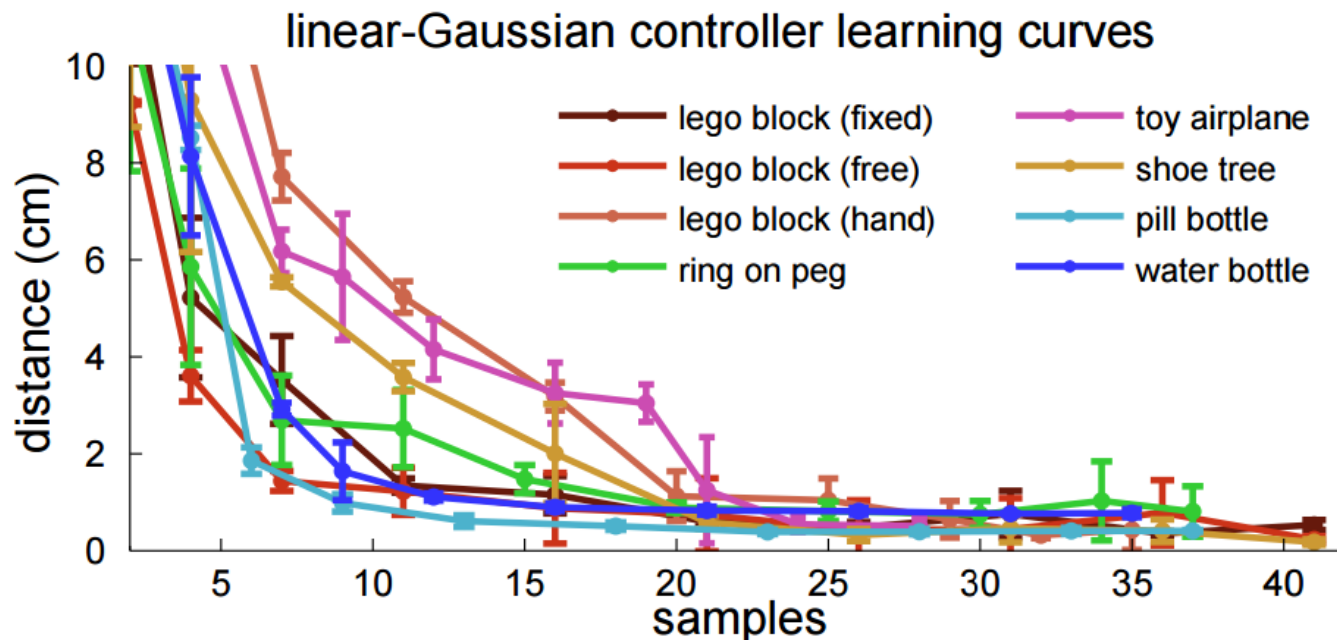
linear-Gaussian controllers



1x real time

autonomous execution

Case study: local models & iterative LQR



Local Models Approach Summary

Levine & Abbeel NIPS '14

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn local model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
e.g. using linear regression
3. update local policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ using local model f_ϕ with KL constraint.
4. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, putting visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ in \mathcal{D}

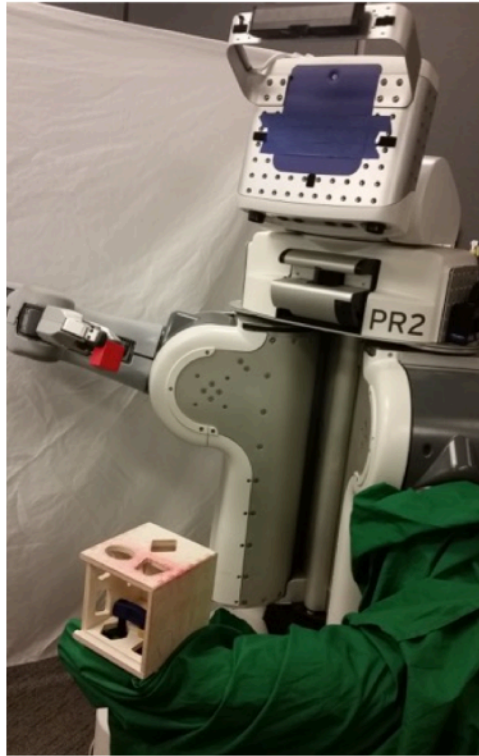
e.g. using iterative LQR

end result: single local policy

Guided policy search: supervise one global policy using multiple local policies

Case study: guided policy search

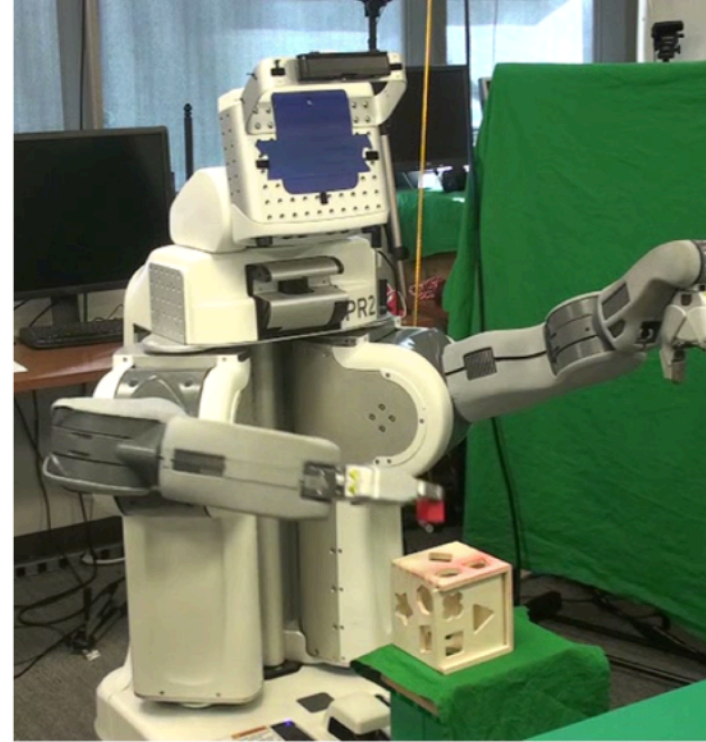
Training time



$$\mathbf{s}_t \longrightarrow \mathbf{a}_t$$

target pose known

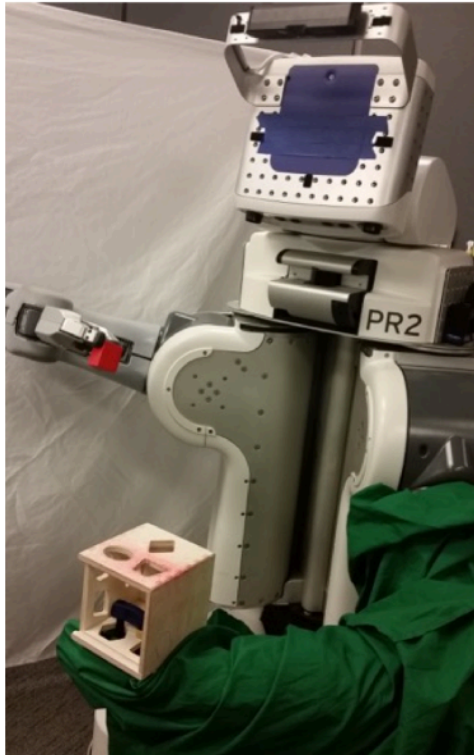
Test time



$$\mathbf{o}(\mathbf{s}_t) \longrightarrow \mathbf{a}_t$$

Case study: guided policy search

Training time

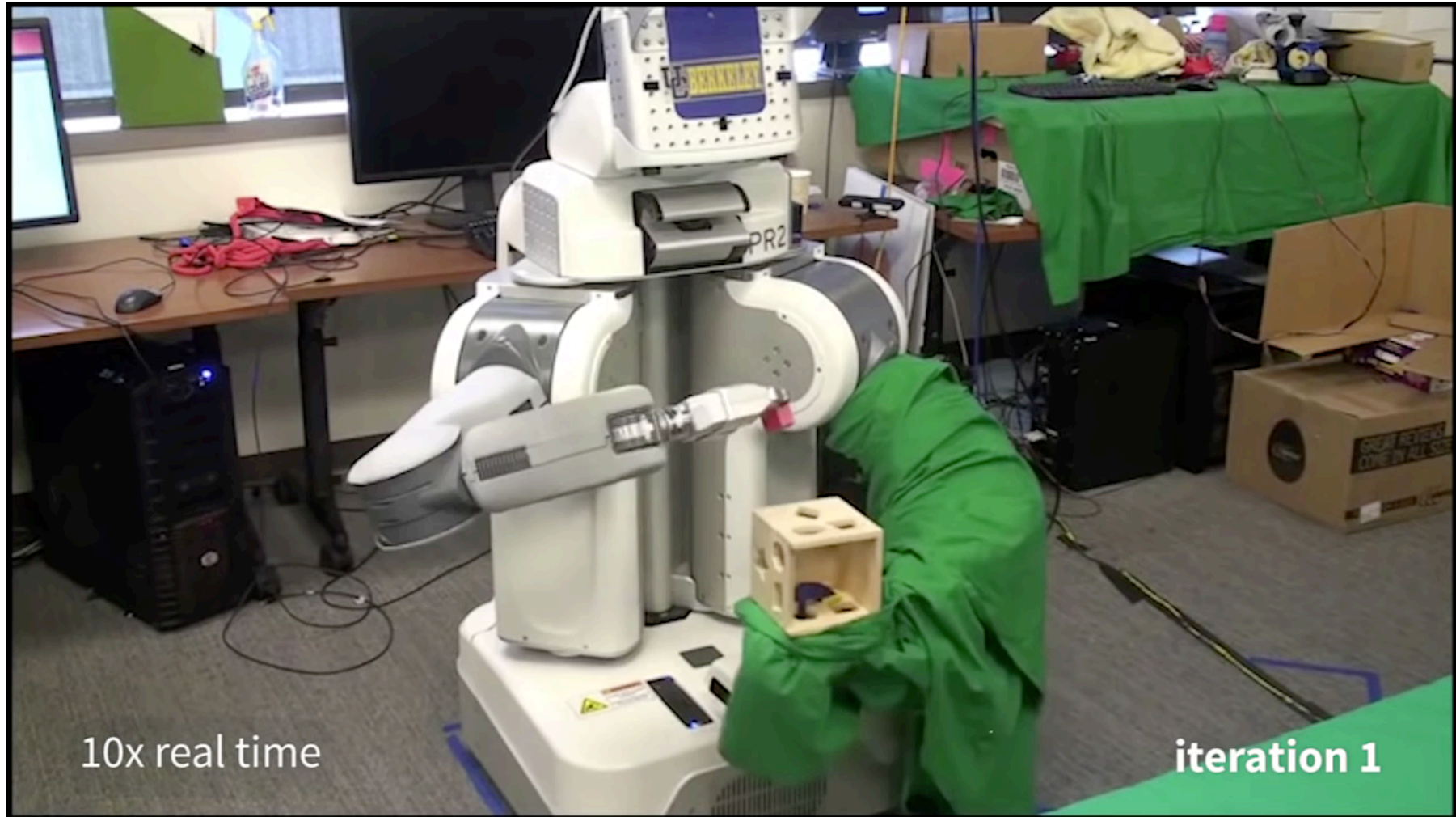


- take samples for each target position
- fit local model and solve for local policy for each target position
- use supervision from local policies to train global neural network policy w/ vision

$$\mathbf{s}_t \longrightarrow \mathbf{a}_t$$

Guided Policy Search: learning

(Levine*, Finn*, et al. JMLR '16)



Guided Policy Search: learned behaviors

(Levine*, Finn*, et al. JMLR '16)



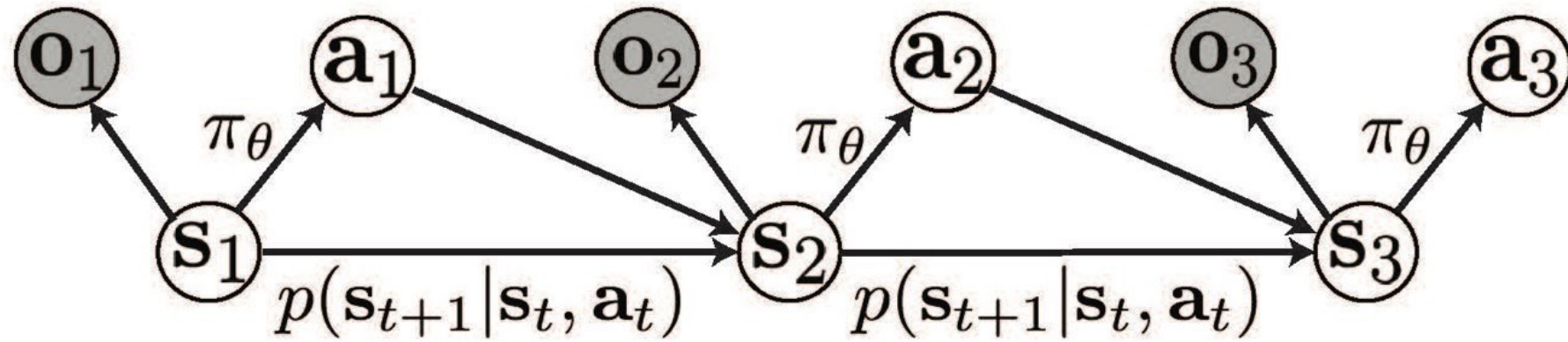
< 300 trials = 25 min of robot time (per task)

+ efficiently learn complex vision-based skills - requires state during training

Outline

1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
3. Using local models & guided policy search
- 4. Handling high-dimensional observations**

Only access to high-dimensional observations (i.e. images)?



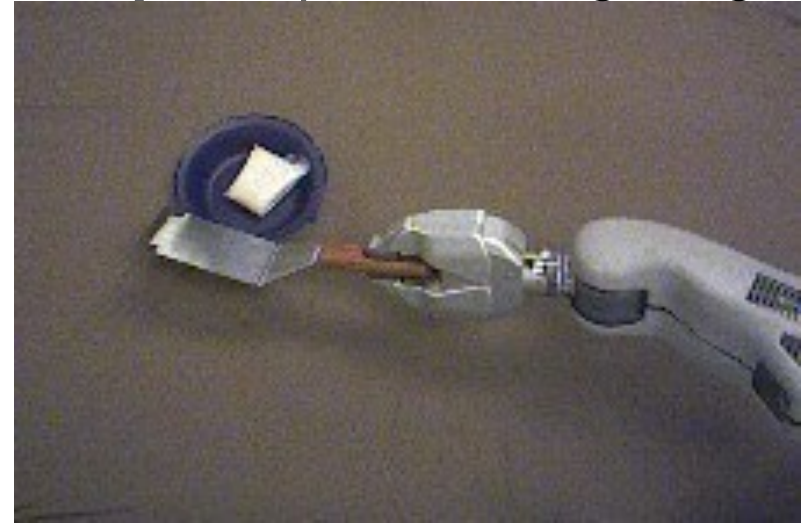
also: no reward signal with only observations

Only access to high-dimensional observations (i.e. images)?



also: no reward signal with only observations

one option: provide image of goal

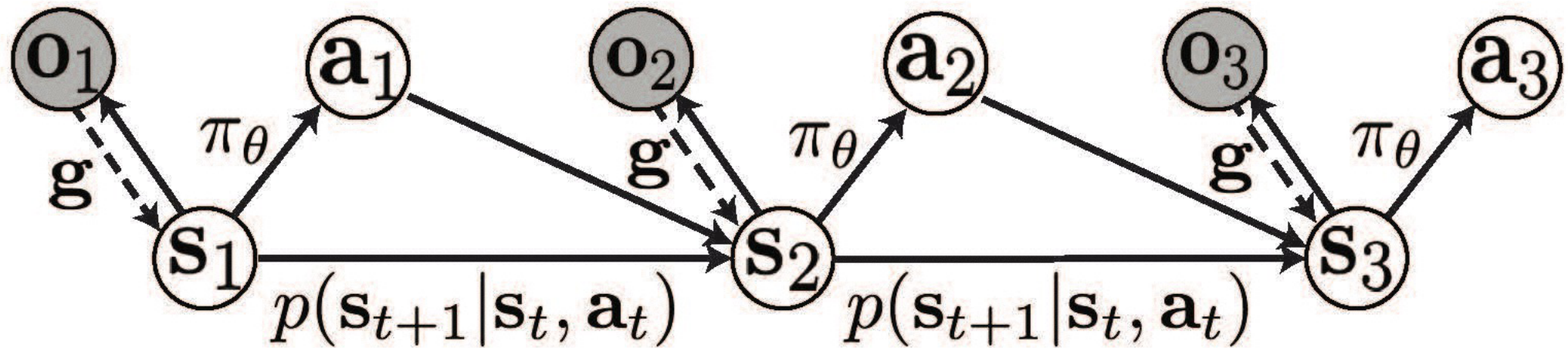


Approaches

1. Learn model in latent space
2. Learn model of observations (e.g. video)
3. Inverse models [won't cover]

Learning in Latent Space

Key idea: learn embedding $g(\mathbf{o}_t)$, then learn model in latent space



Learning in Latent Space

Key idea: learn embedding $\mathbf{s}_t = g(\mathbf{o}_t)$, then do model-based RL in latent space

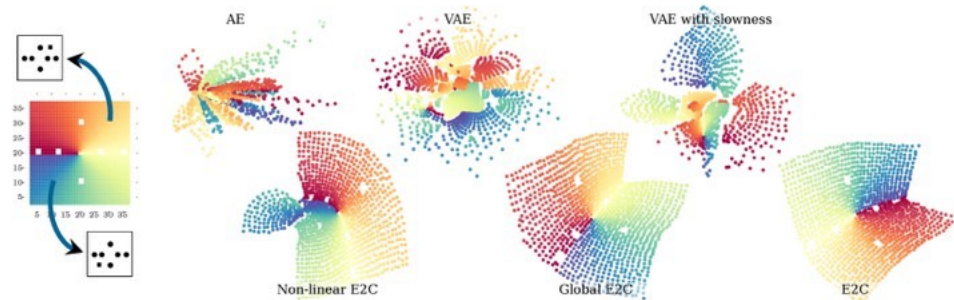
Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images

Deep Spatial Autoencoders for Visuomotor Learning

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, Pieter Abbeel

Manuel Watter* Jost Tobias Springenberg*
Joschka Boedecker
University of Freiburg, Germany
{watterm, springj, jboedeck}@cs.uni-freiburg.de

Martin Riedmiller
Google DeepMind
London, UK
riedmiller@google.com



NIPS 2015

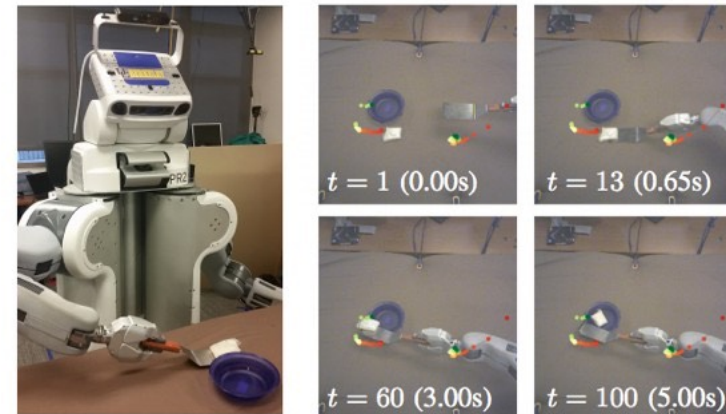


Fig. 1: PR2 learning to scoop a bag of rice into a bowl with a spatula (left) using a learned visual state representation (right).

ICRA 2016

Learning in Latent Space

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{o}_t)$ (e.g., exploratory policy) to collect $\mathcal{D} = \{(\mathbf{o}, \mathbf{a}, \mathbf{o}')_i\}$
2. learn latent embedding of observation $\mathbf{s}_t = g(\mathbf{o}_t)$ and dynamics model $\mathbf{s}' = f_\phi(\mathbf{s}, \mathbf{a})$
3. use model $f_\phi(\mathbf{s}, \mathbf{a})$ to optimize policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. run $\pi_\theta(\mathbf{a}_t|g(\mathbf{o}_t))$, appending visited tuples $(\mathbf{o}, \mathbf{a}, \mathbf{o}')$ to \mathcal{D}

What is reward for optimizing policy?

reward signal: $r(\mathbf{o}, \mathbf{a}) = r(\mathbf{a}) + \|g(\mathbf{o}) - g(\mathbf{o}_{\text{goal}})\|$

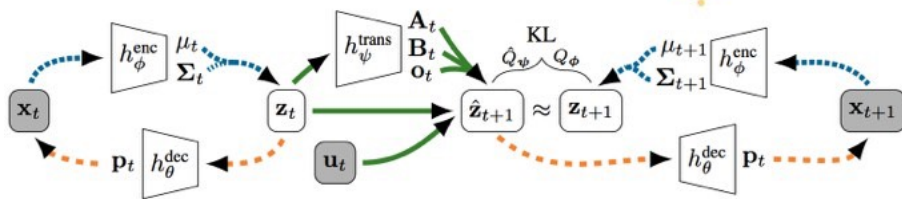
Aside: If you have reward observations (i.e. video games), can simply fit a reward model instead.

Learning in Latent Space

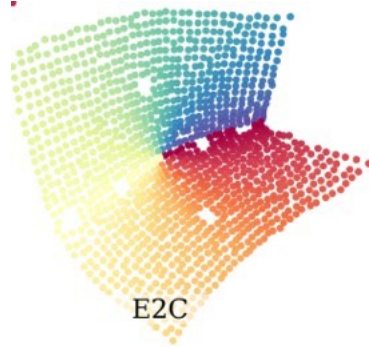
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., exploratory policy) to collect $\mathcal{D} = \{(\mathbf{o}, \mathbf{a}, \mathbf{o}')_i\}$
2. learn latent embedding of observation $\mathbf{s}_t = g(\mathbf{o}_t)$ and dynamics model $\mathbf{s}' = f_\phi(\mathbf{s}, \mathbf{a})$
3. use model $f_\phi(\mathbf{s}, \mathbf{a})$ to optimize policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. run $\pi_\theta(\mathbf{a}_t|g(\mathbf{o}_t))$, appending visited tuples $(\mathbf{o}, \mathbf{a}, \mathbf{o}')$ to \mathcal{D}

How to optimize latent embedding g ?

Watter et al.'15

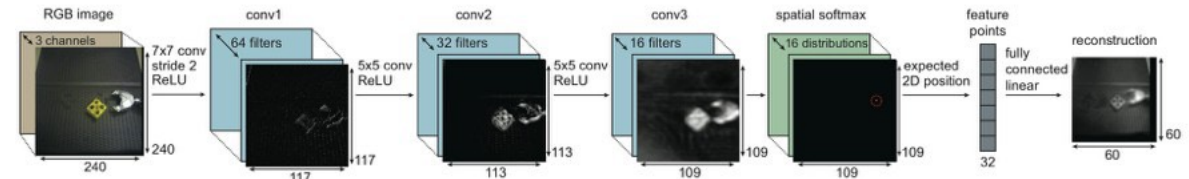


learn embedding & model **jointly**



E2C

Finn et al.'16



embedding is **smooth and structured**

Learning in Latent Space

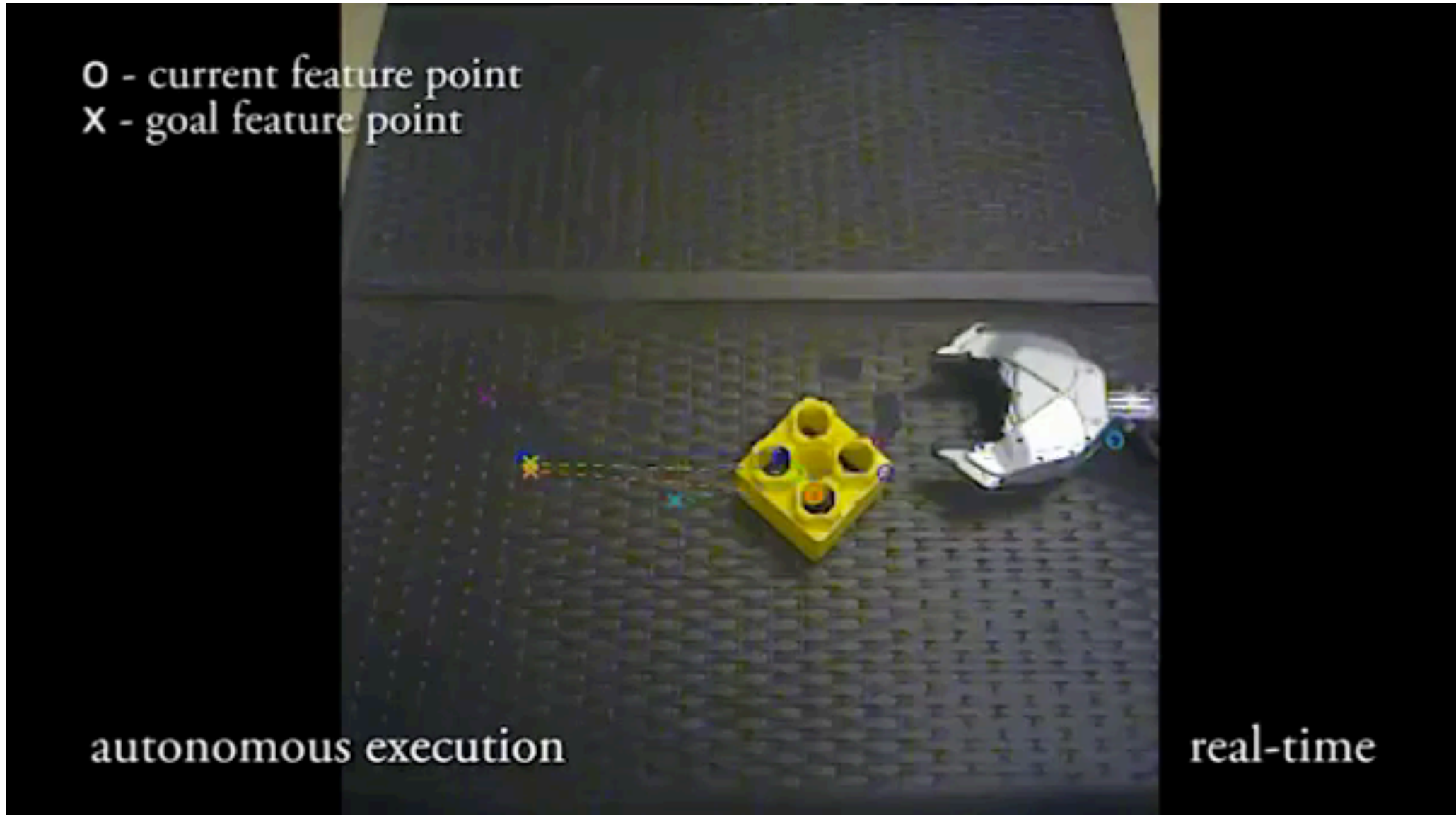


Goal state for trajectory optimization

~300 trials = ~25 min of robot time (per task)

Watter et al. NIPS '15

Learning in Latent Space



125 trials = 11 min of robot time (per task)

Finn et al. ICRA'16

Learning in Latent Space

Pros:

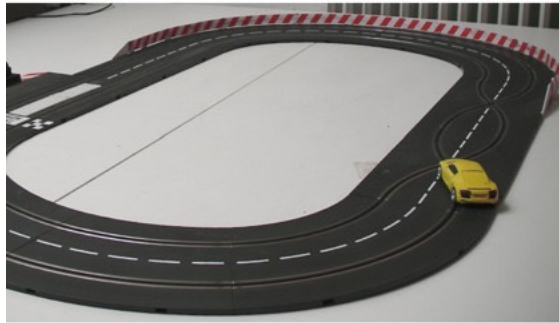
- + Learn complex visual skills very efficiently
- + Structured representation enables effective learning

Cons:

- Reconstruction objectives might not recover the right representation

Aside: Low-dimensional embedding can also be useful for model-free approaches

model-free RL in latent space



FQI in latent space
Lange et al. '12

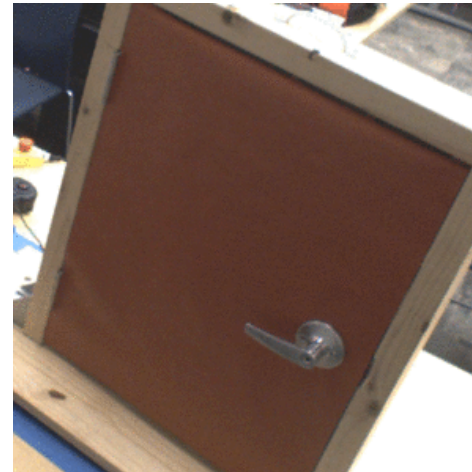


TRPO in latent space
Ghadirzadeh et al. '17

use embedding for reward function

Sermanet et al. RSS '17

video demonstration



acquire reward using
ImageNet features

learned policy



+ model-free RL

If you have a reward, you can predict it to form better latent space

Jaderberg et al. '17, Shelhamer et al. '17

Modeling directly in observation space

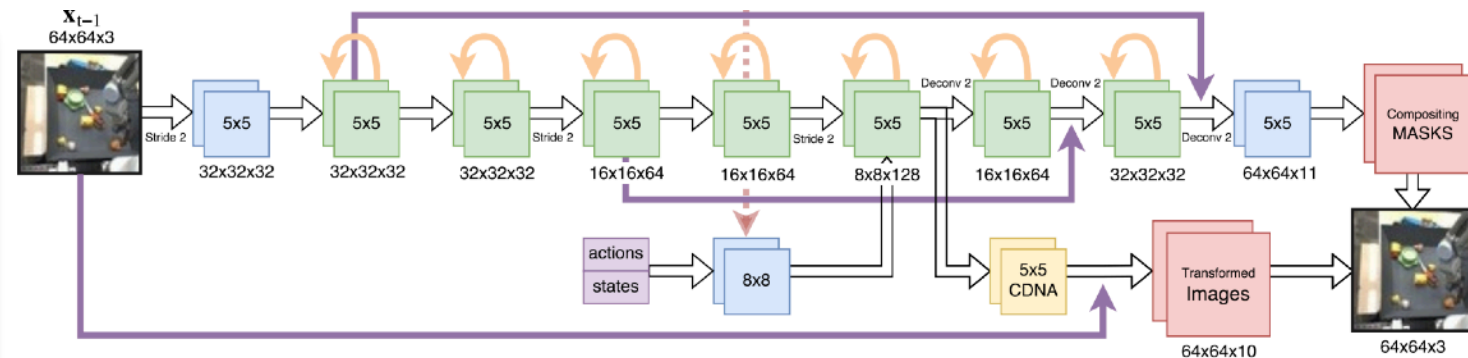
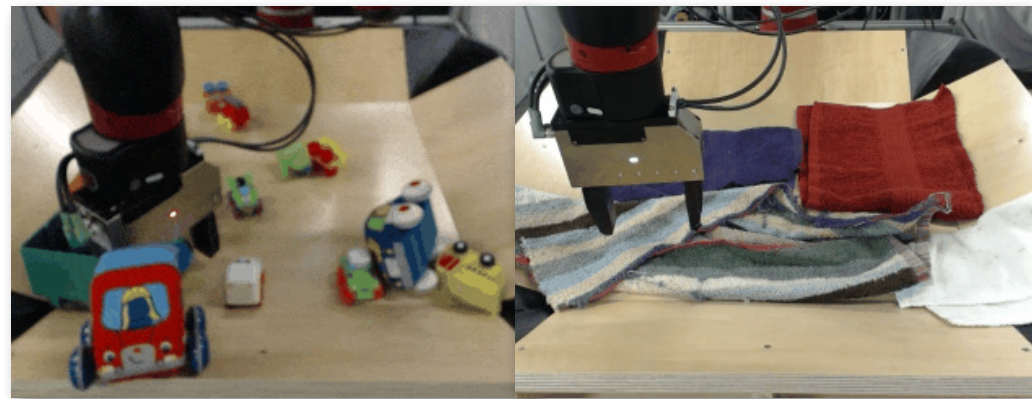
Recall MPC

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

every N steps

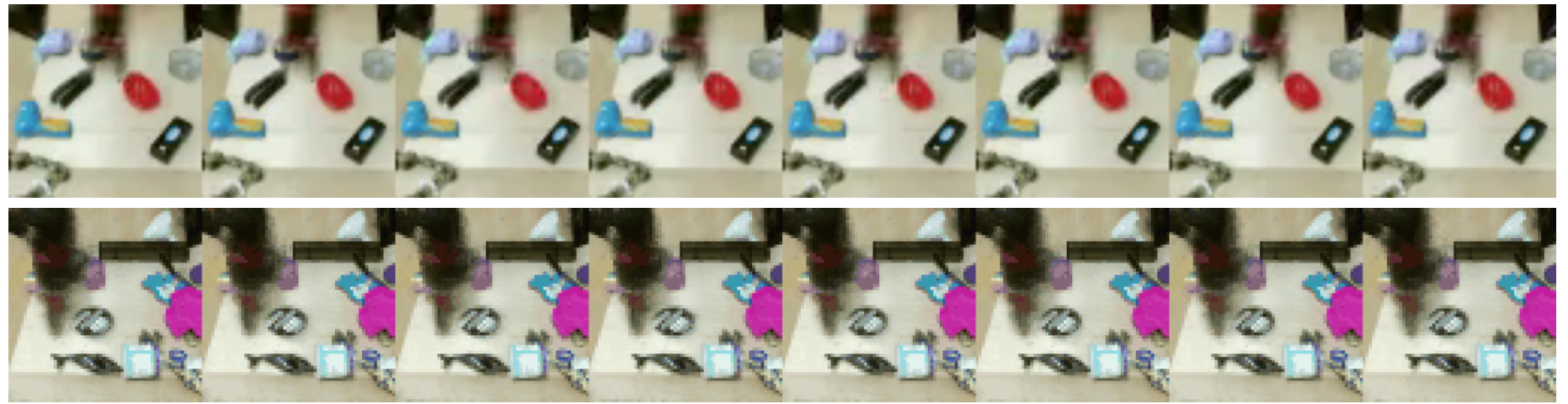


action-conditioned video prediction



Learn to predict

$$\mathbf{I}_t, \mathbf{a}_{t:t+H} \rightarrow \mathbf{I}_{t:t+H}$$



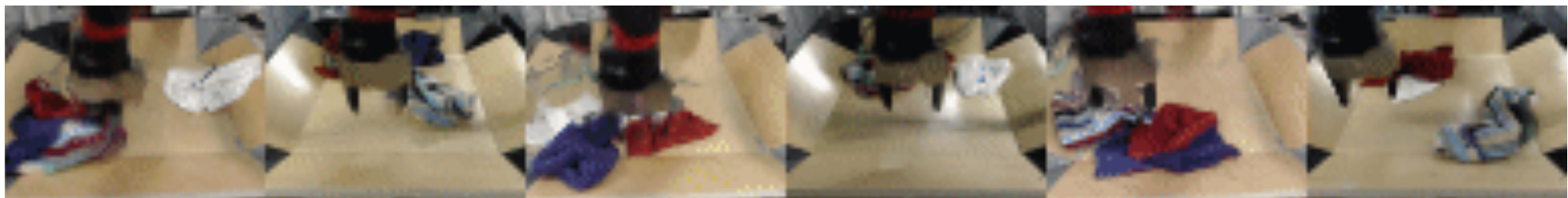
Contrast to:



Models capture **general purpose** knowledge about the world

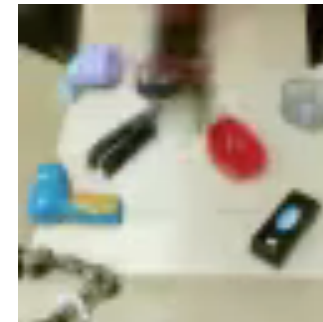
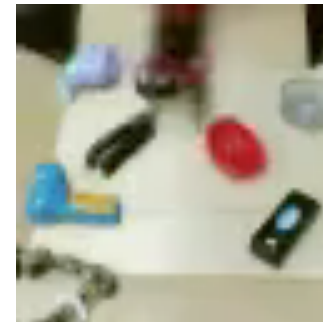
Use **all** of the available supervision signal.

Also: No assumptions about task **representations**.



Planning with Visual Foresight

1. Consider potential action sequences
2. Predict the future for each action sequence
3. Pick best future & execute corresponding action
4. Repeat 1-3 to replan in real time

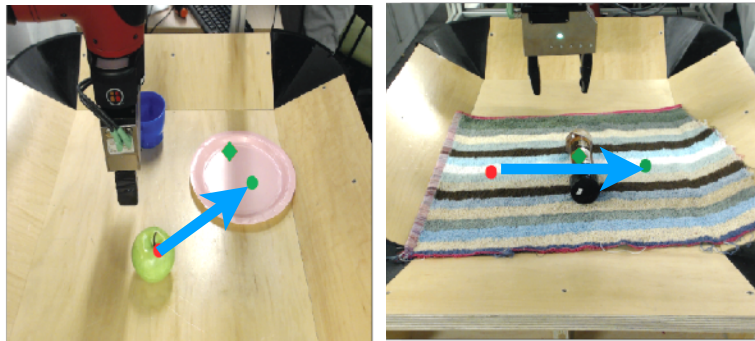


visual “model-predictive control” (MPC)

Overall System: Collect data, Train predictive model, Plan to achieve goals

Which future is the best one?

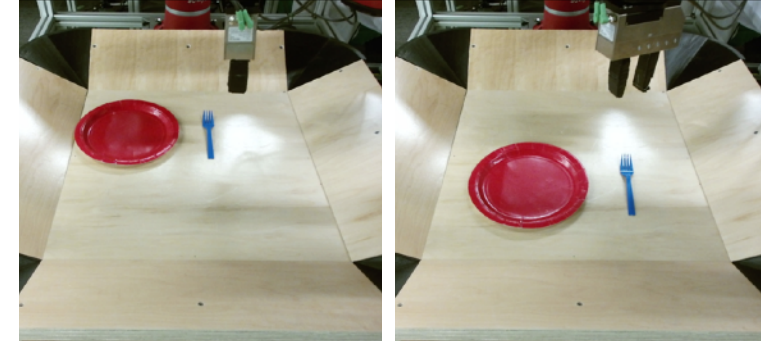
Human specifies a goal by:



Selecting where pixels should move.



Providing an image of the goal.



Providing a few examples of success.

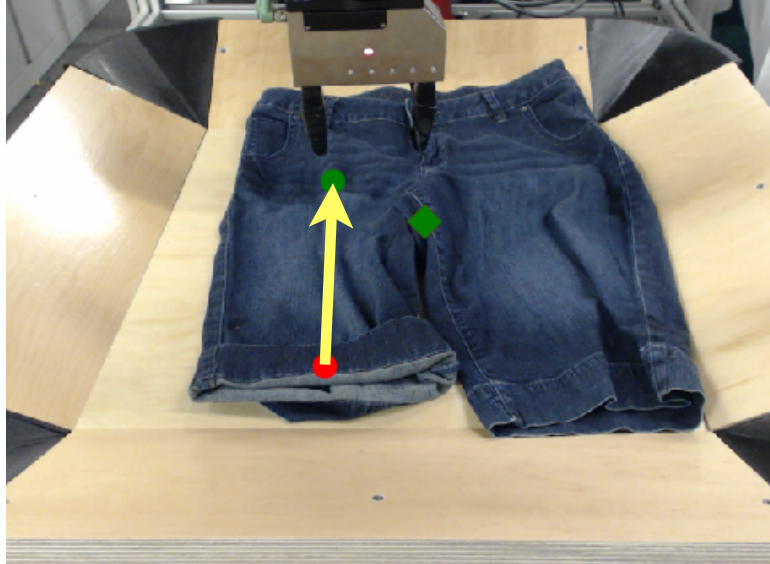
Finn & Levine ICRA '17

Ebert, Lee, Levine, Finn CoRL '18

Xie, Singh, Levine, Finn CoRL '18

Modeling directly in Observation Space

Specify goal



Visual MPC execution



Visual MPC
w.r.t. goal

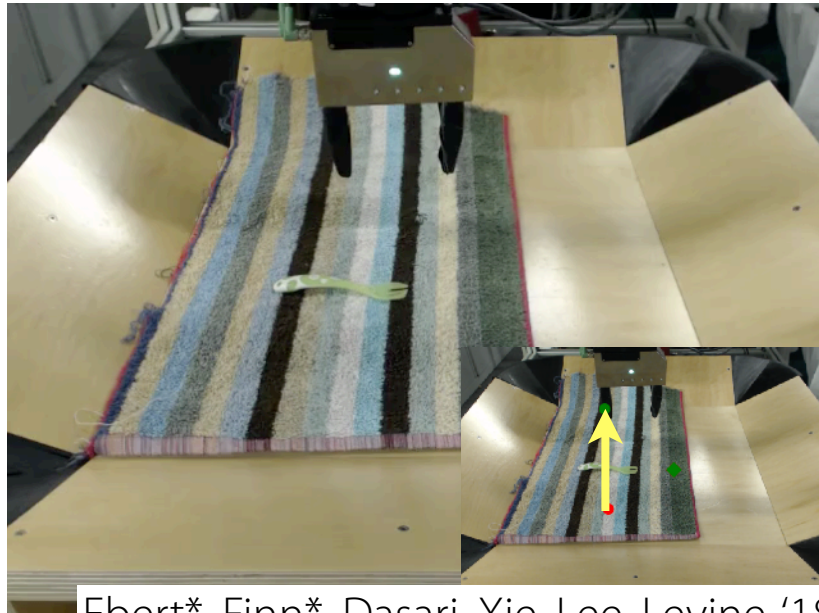
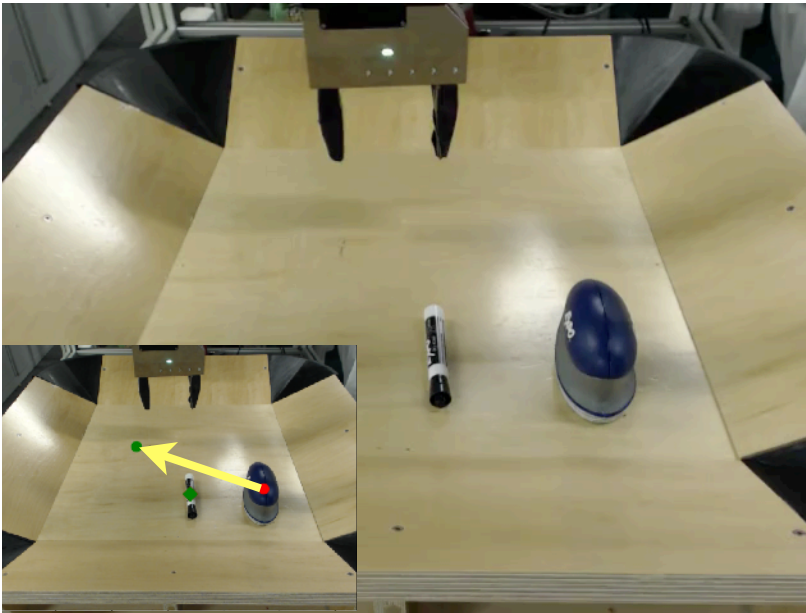
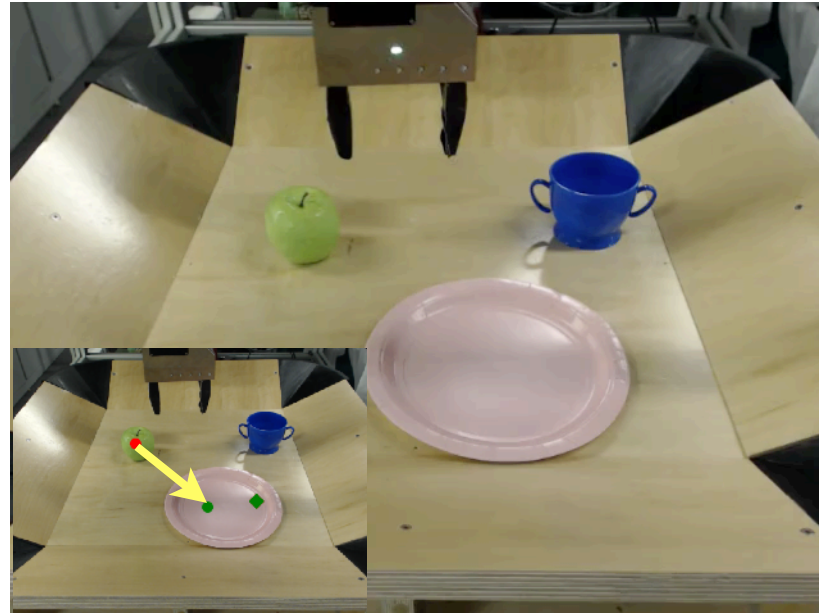


~2 weeks of *unsupervised* robot time

Only human involvement: programming initial motions and providing objects to play with.

Planning with a **single model** for many tasks

Video speed: 2x



Modeling directly in observation space

Pros:

- + Entirely self-supervised
- + Learn for a variety of tasks
- + More efficient than single-task model-free learning

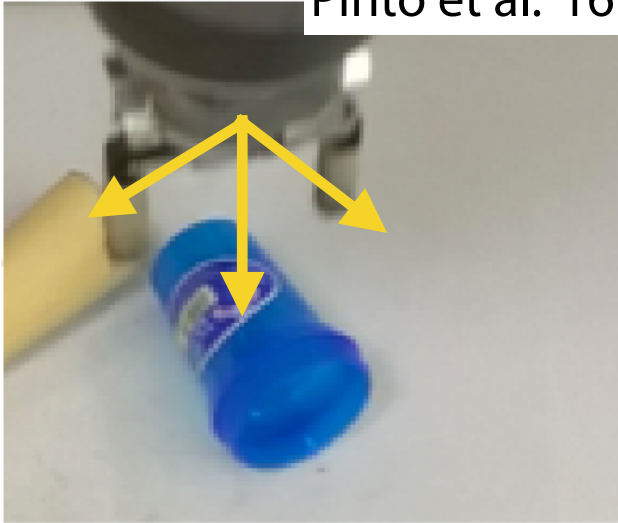
Cons:

- Can't [yet] handle as complex skills as model-free methods

Predict alternative quantities

If I take a set of actions:

Pinto et al. '16



Will I successfully grasp?

Will I collide?

Kahn et al. '17



What will health/damage/etc. be?

Pros:

+ Only predict task-relevant quantities!

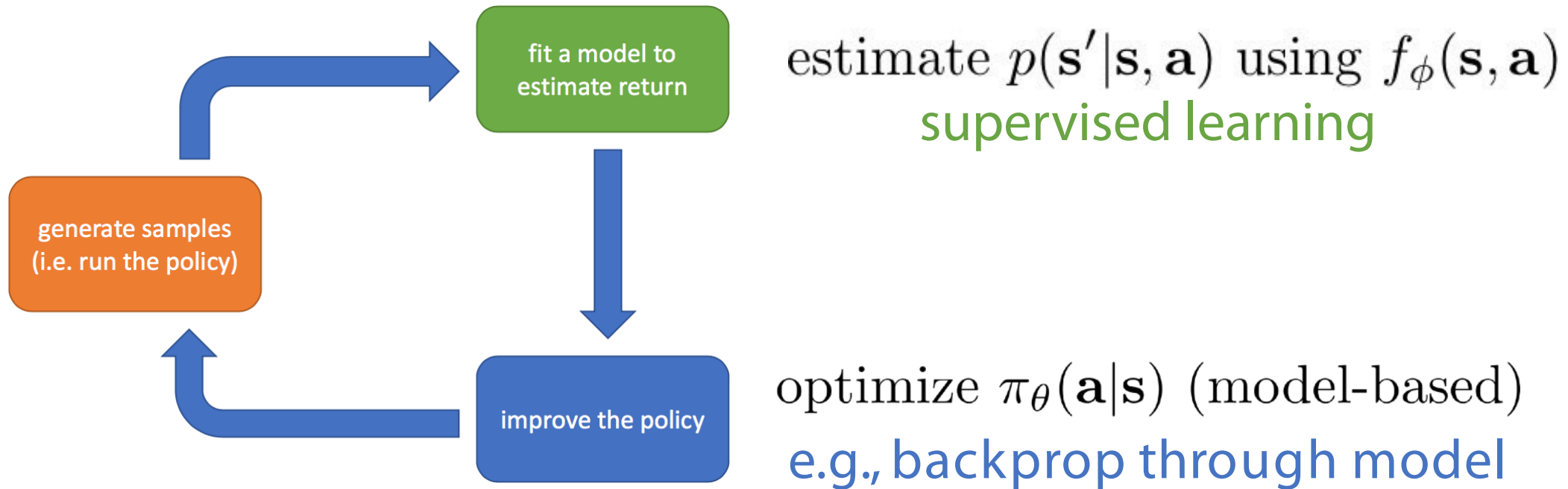
Cons:

- Need to manually pick quantities, must be able to directly observe them

Outline

1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
3. Using local models & guided policy search
4. Handling high-dimensional observations

Model-based RL Review



Correcting for model errors:

refit model with new data, replan with MPC, use local models or uncertainty

Model-based RL from raw observations:

learn latent space, typically with unsupervised learning, or
model & plan directly in observational space

Model-Based vs. Model-Free Algorithms

Models:

- + Easy to collect data in a scalable way (self-supervised)
- + Possibility to transfer across tasks
- + Typically require a smaller quantity of supervised data
- Models don't optimize for task performance
- Sometimes harder to learn than a policy
- Often need assumptions to learn complex skills (continuity, resets)

Model-Free:

- + Makes little assumptions beyond a reward function
- + Effective for learning complex policies
- Require a lot of experience (slower)
- Not transferable across tasks

Ultimately we will want both!

Challenges & Frontiers

Long-horizon prediction & planning

- Structured latent representations
- need:** - Uncertainty
- Compositionality

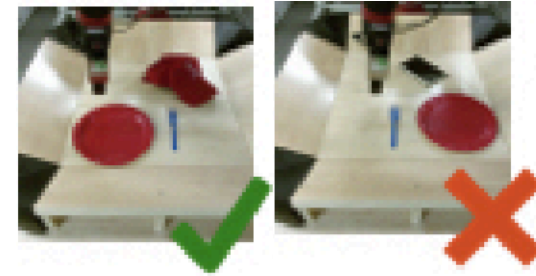


Janner, Levine, Freeman, Tenenbaum, Finn, Wu '18

Exploration (models can help!)

Stadie et al. arXiv '15, Oh et al. NIPS '16, Burda et al. '18

Internal reward representations



Combining elements of model-based & model-free

- **use roll-outs from model as experience:** Sutton '90, Gu et al. ICML '16, Kurutach et al. ICLR '18
- **model-free policy with planning capabilities:** Tamar et al. NIPS '16, Pascanu et al. '17
- **model-based look-ahead:** Guo et al. NIPS '14, Silver et al. Nature '16, Buckman et al. NIPS '18

Questions?