Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Sim2Real

Katerina Fragkiadaki

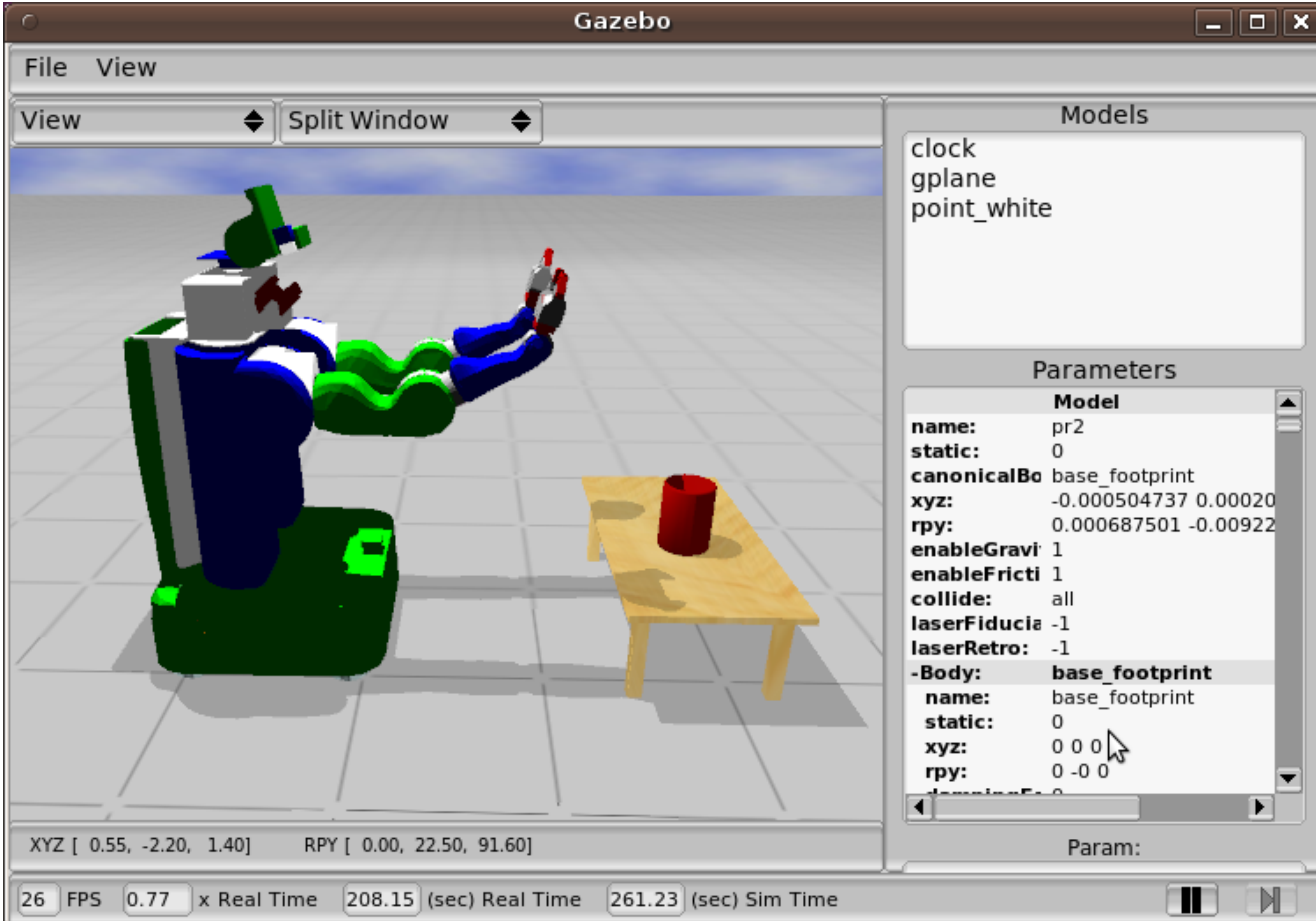
So far

The requirement of large number of samples for RL, only possible in simulation, renders RL a model-based framework, we can't really rely (solely) on interaction in the real world (as of today)

- In the real world, we usually finetune model and policies learnt in simulation

Physics Simulators

Mujoko, bullet, gazeebo, etc.

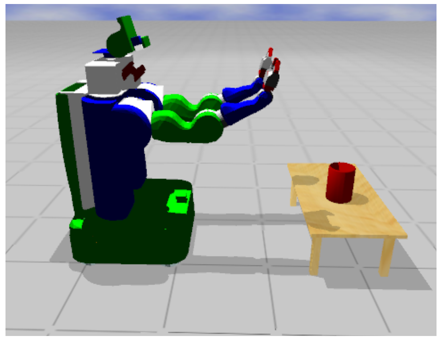


Pros of Simulation

- We can afford many more samples!
- Safety
- Avoids wear and tear of the robot
- Good at rigid multibody dynamics

Cons of Simulation

- Under-modeling: many physical events are not modeled.
- Wrong parameters. Even if our physical equations were correct, we would need to estimate the right parameters, e.g., inertia, frictions (system identification).
- Systematic discrepancy w.r.t. the real world regarding:
  - observations
  - dynamics



as a result, policies that learnt in simulation do not transfer to the real world

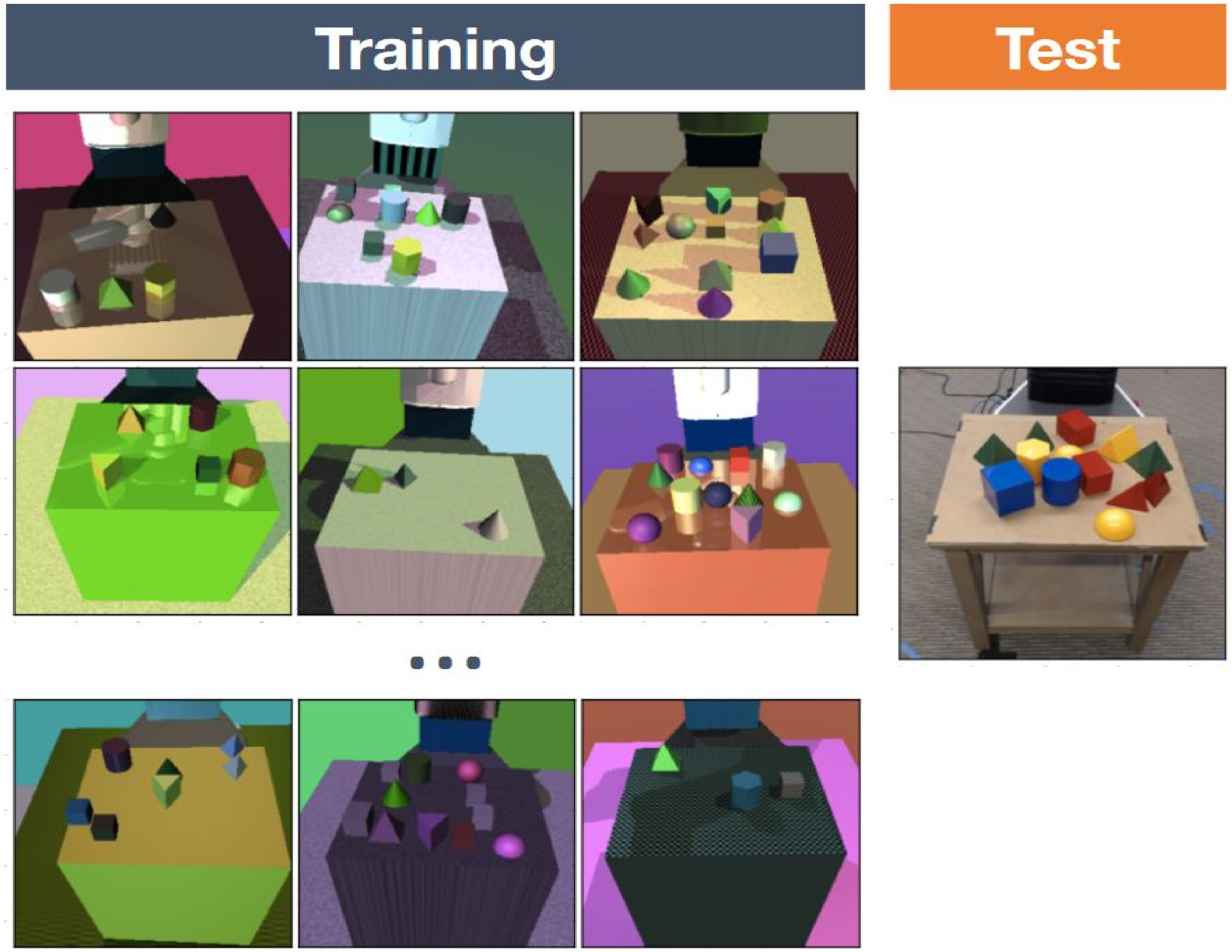
- Hard to simulate deformable objects (finite element methods are very computational intensive)

What has shown to work

- Domain randomization (dynamics, images)
  - With enough variability in the simulator, the real world may appear to the model as just another variation"
- Learning not from pixels but rather from label maps-> semantic maps between simulation and real world are closer than textures
- Learning higher level policies, not low-level controllers, as the low level dynamics are very different between Sim and REAL

Domain randomization

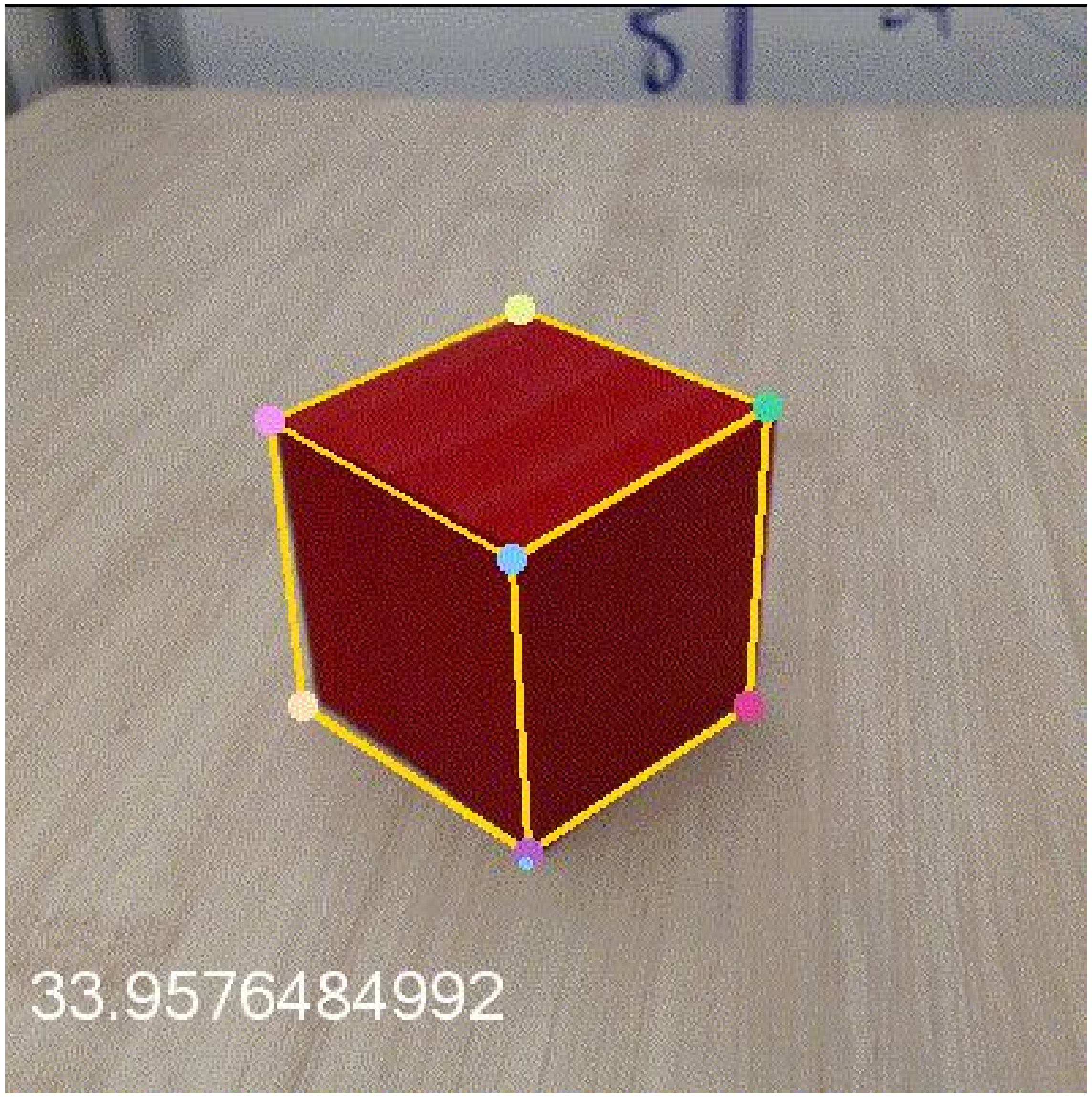
for detecting and grasping objects



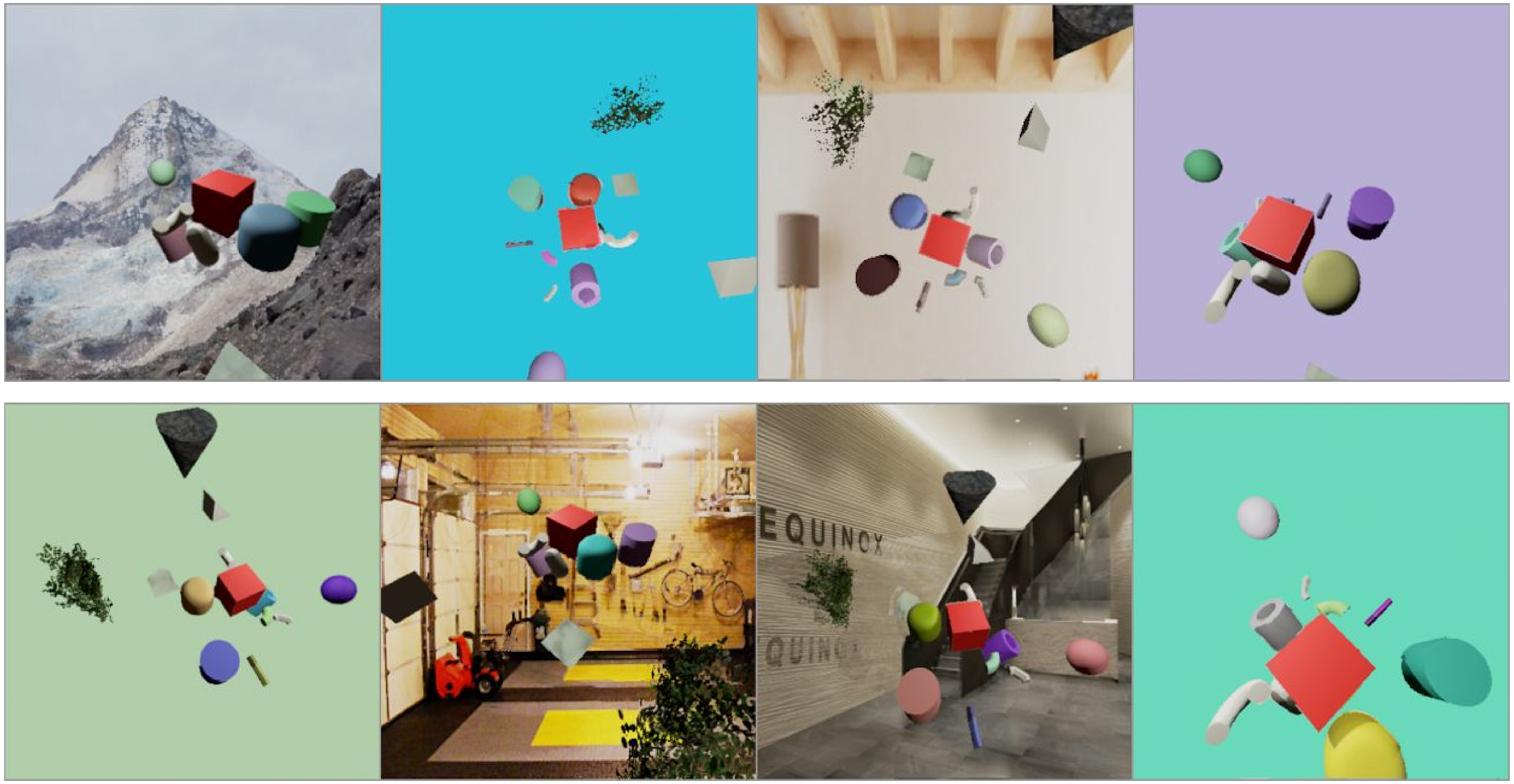
Tobin et al., 2
arXiv:1703.06

Let’s try a more fine grained task

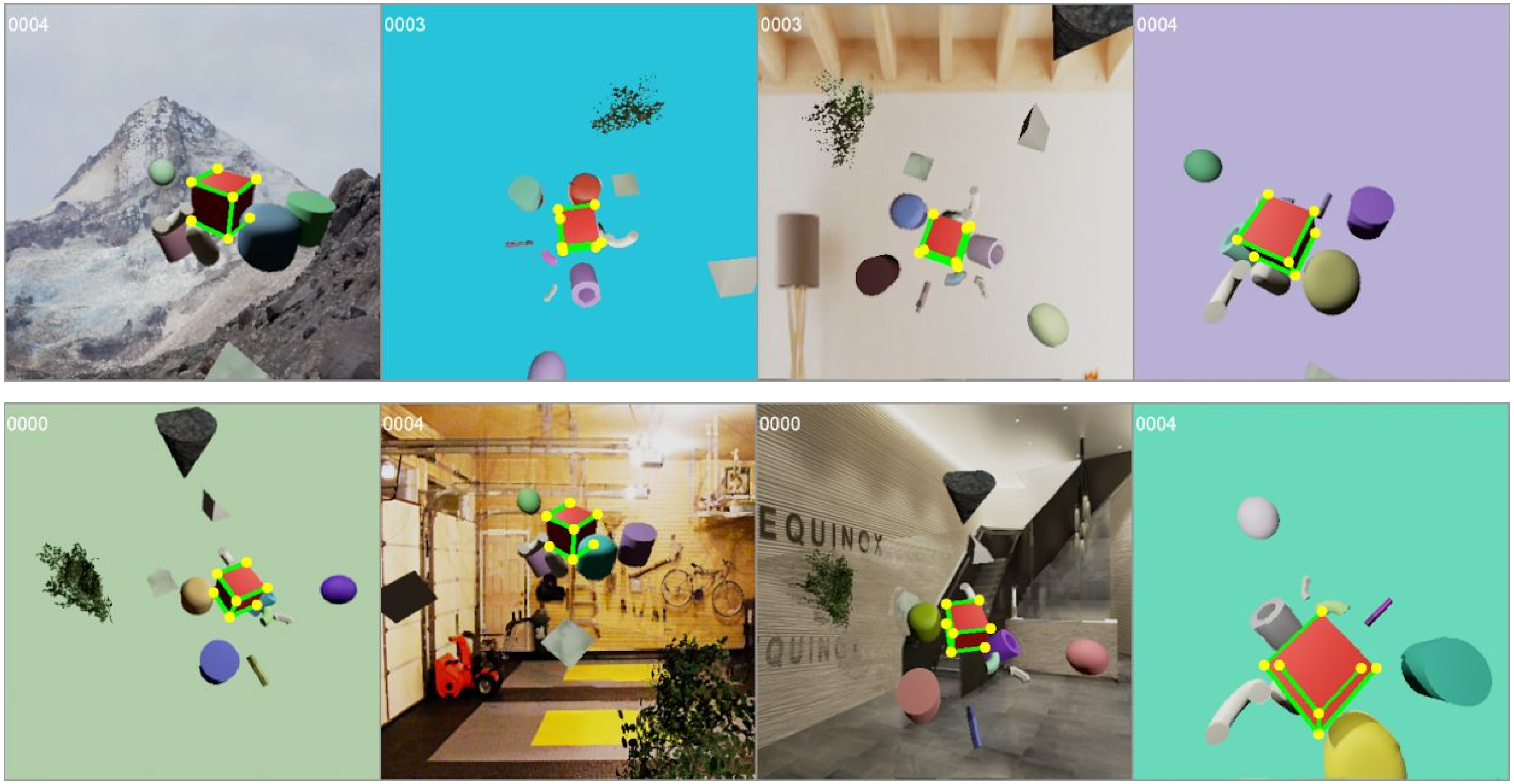
Cuboid Pose Estimation



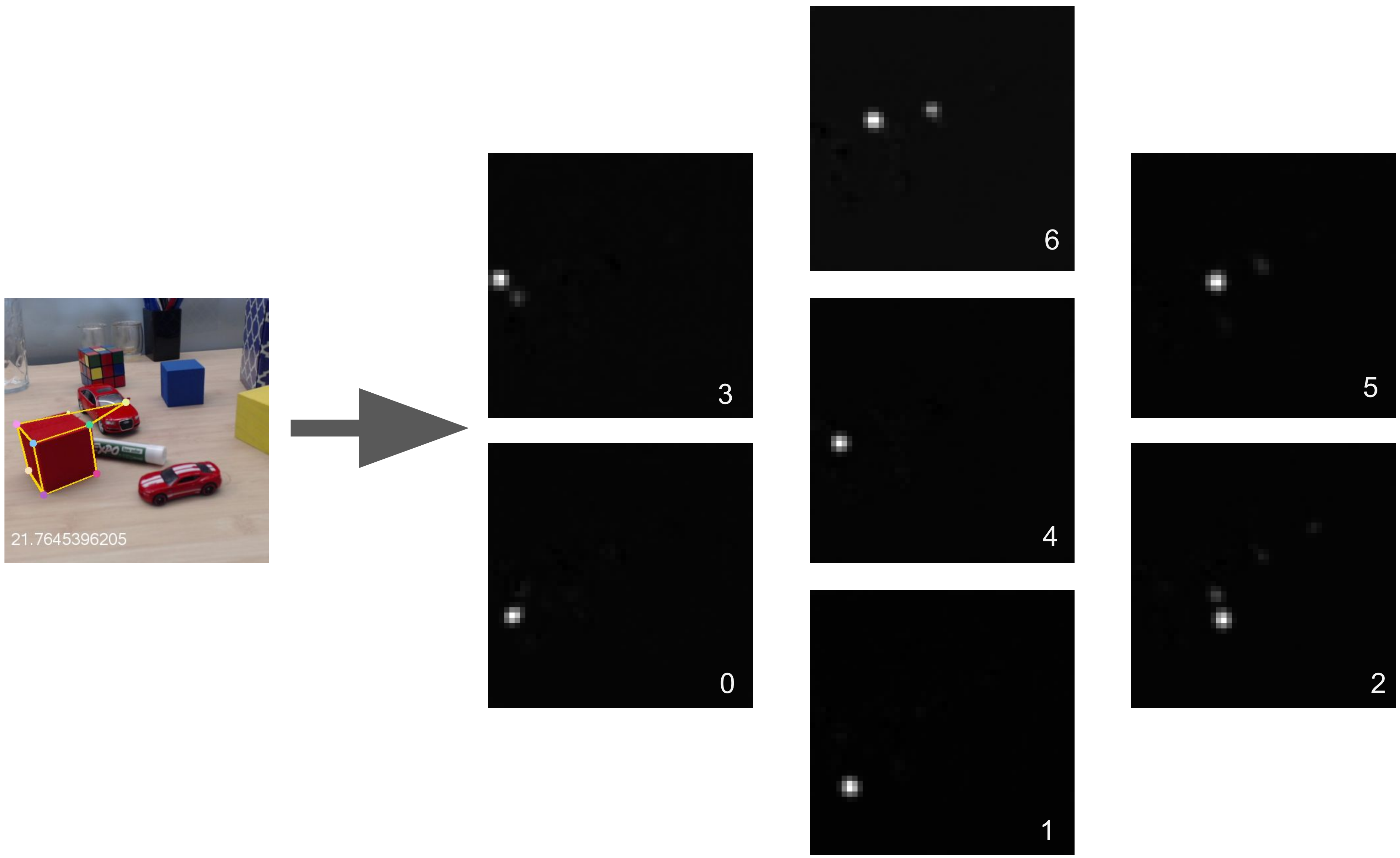
Data generation



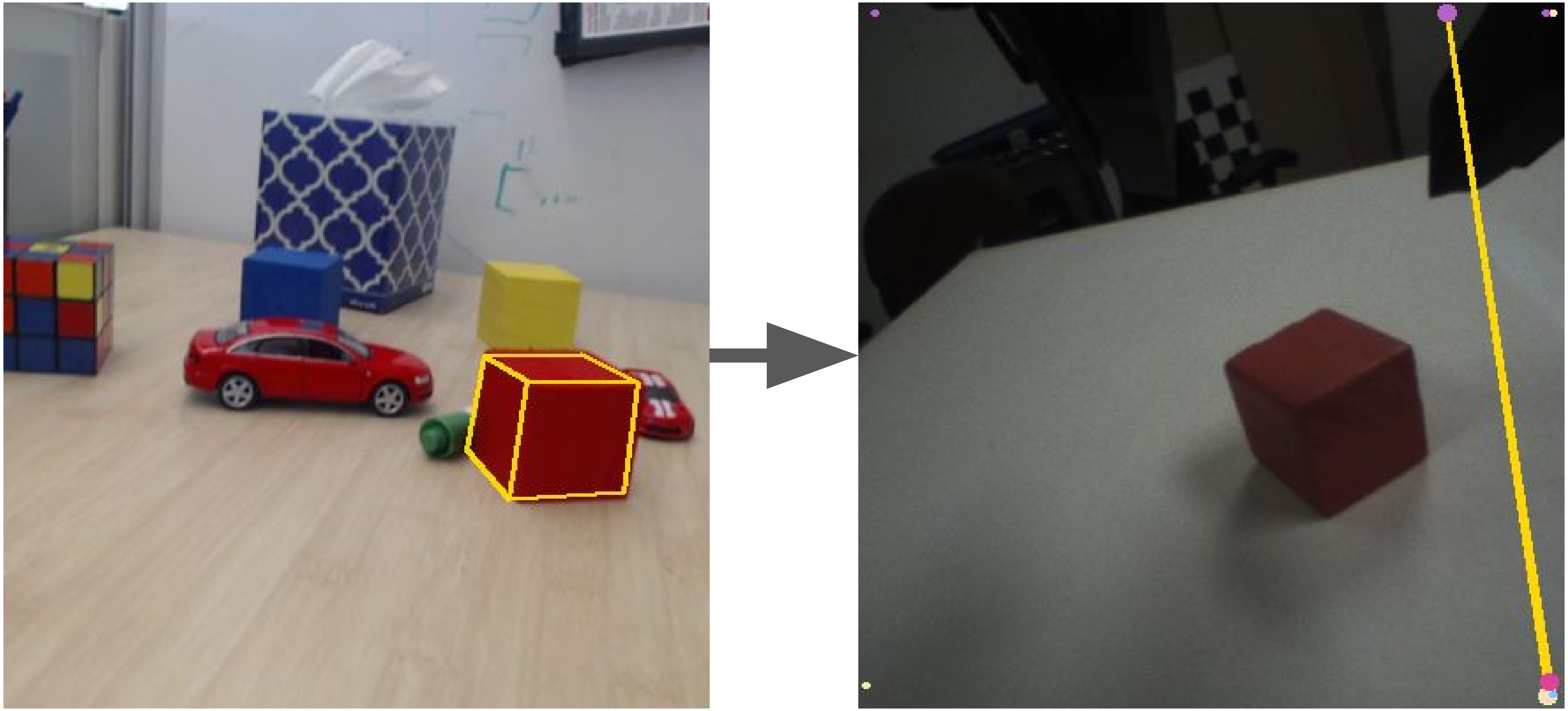
Data generation



Regressing to vertices

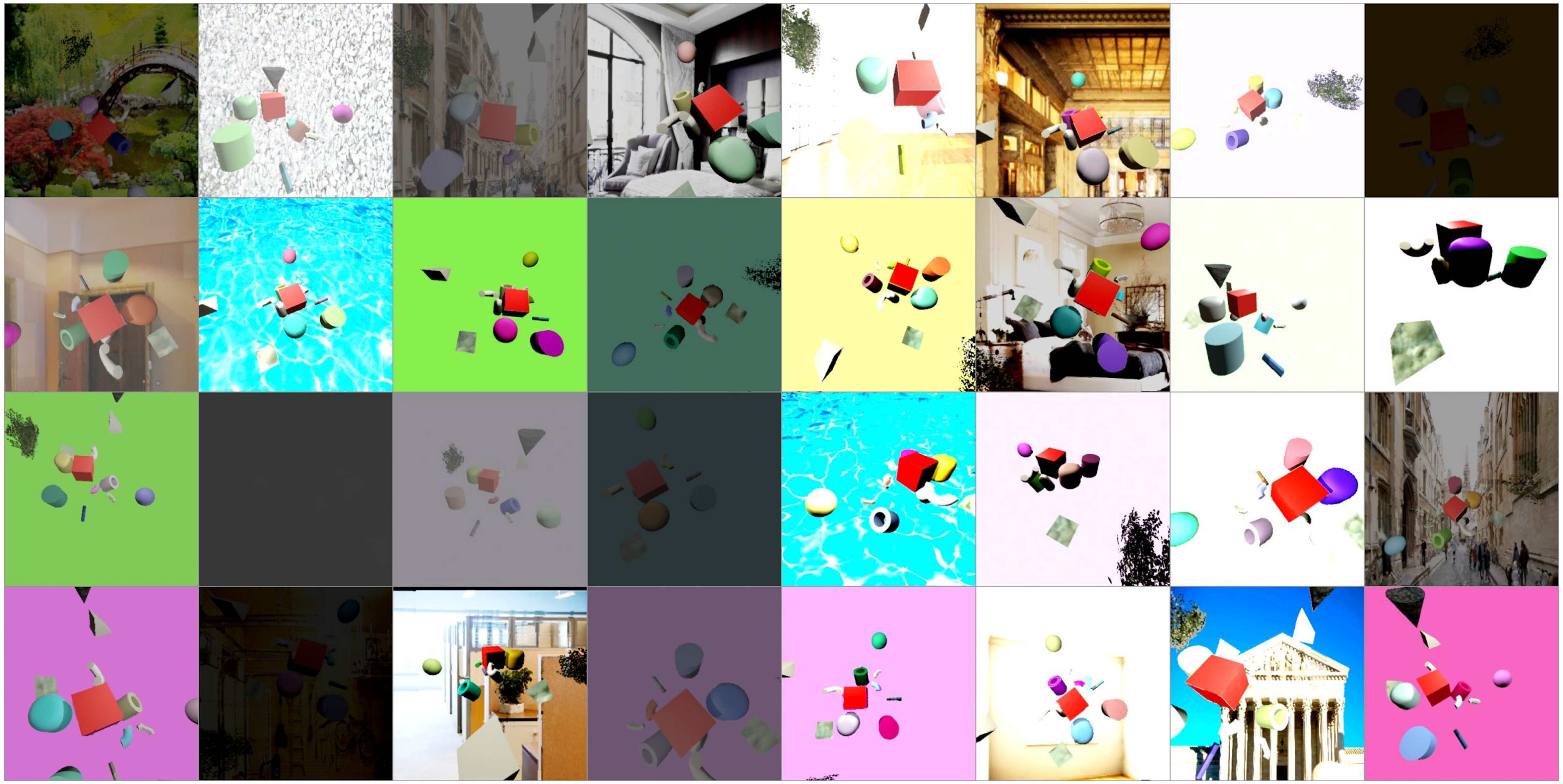


SIM2REAL

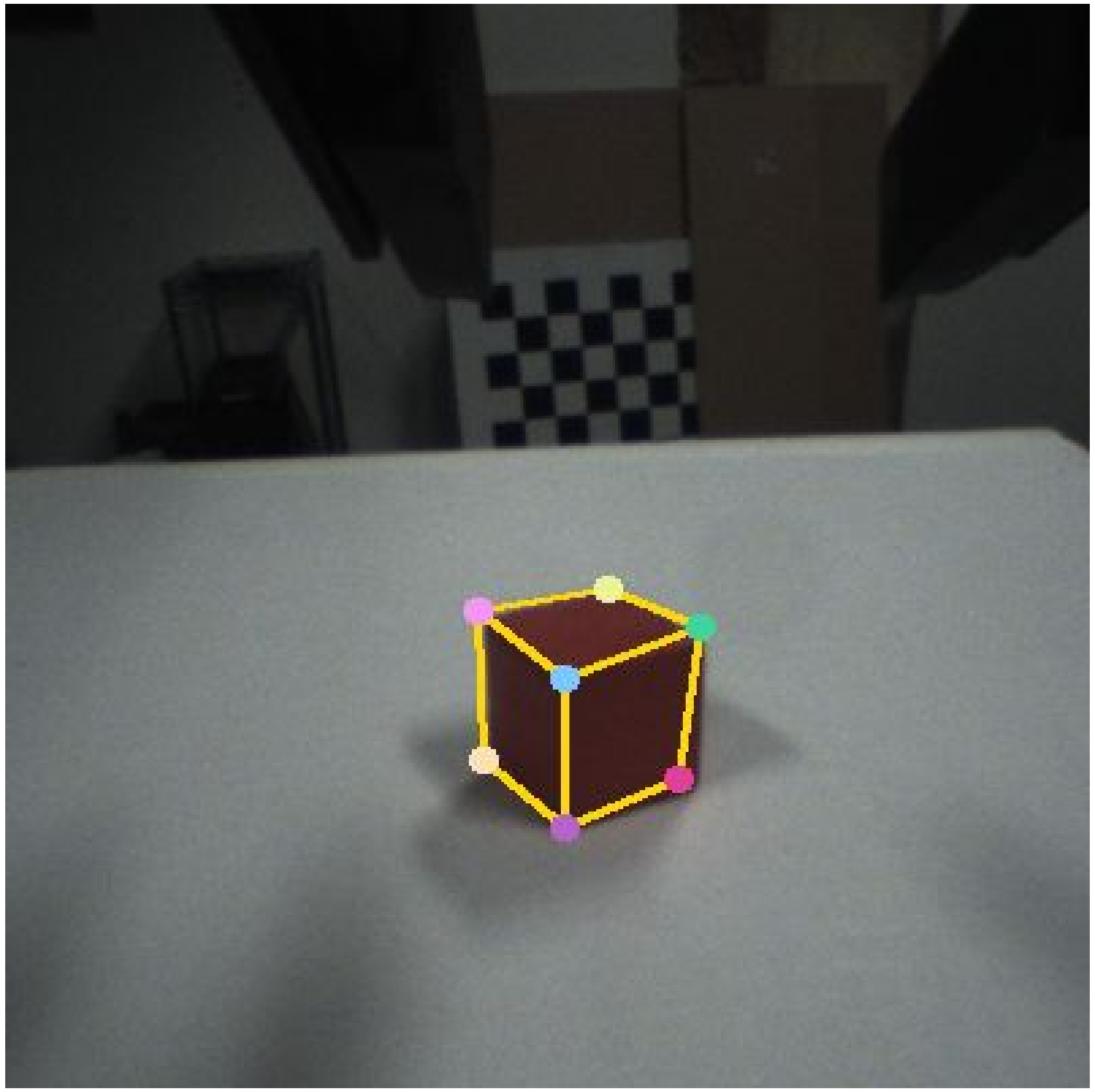


Data generation

Data - Contrast and Brightness

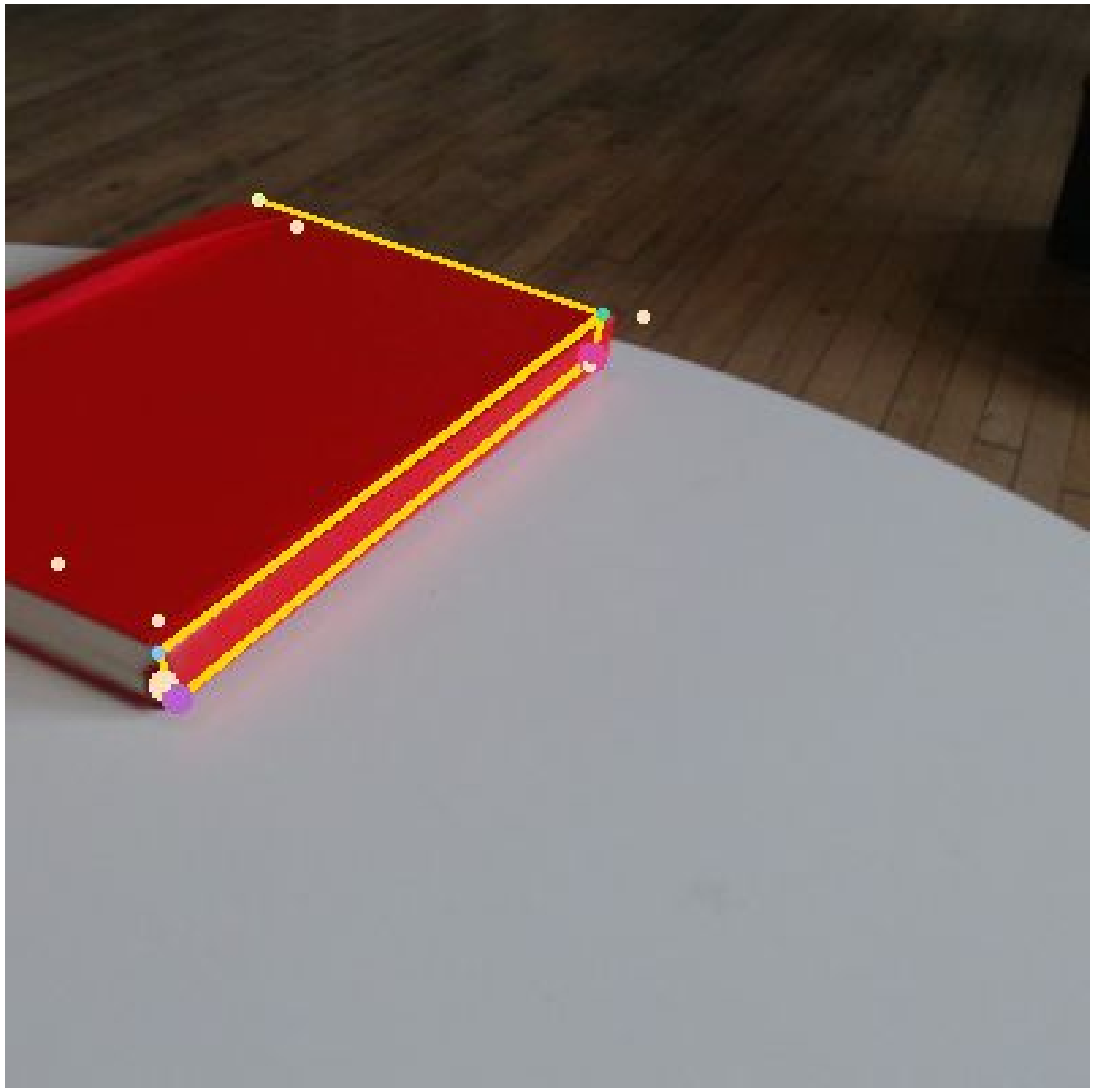


SIM2REAL



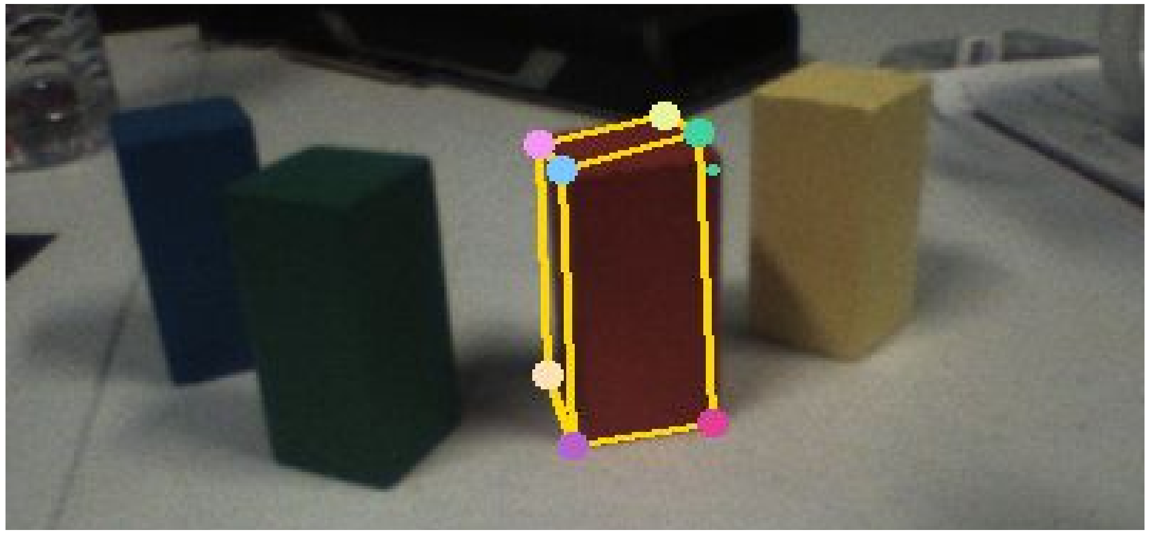
SIM2REAL

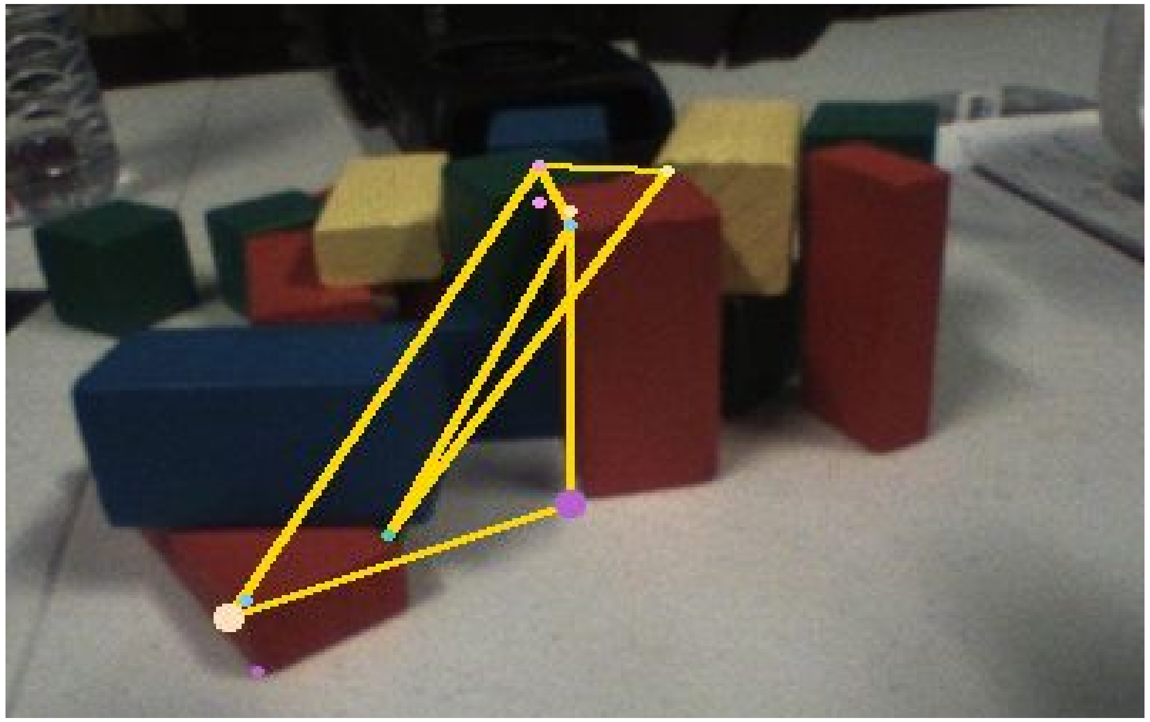
Surprising Result



SIM2REAL

Baxter's camera

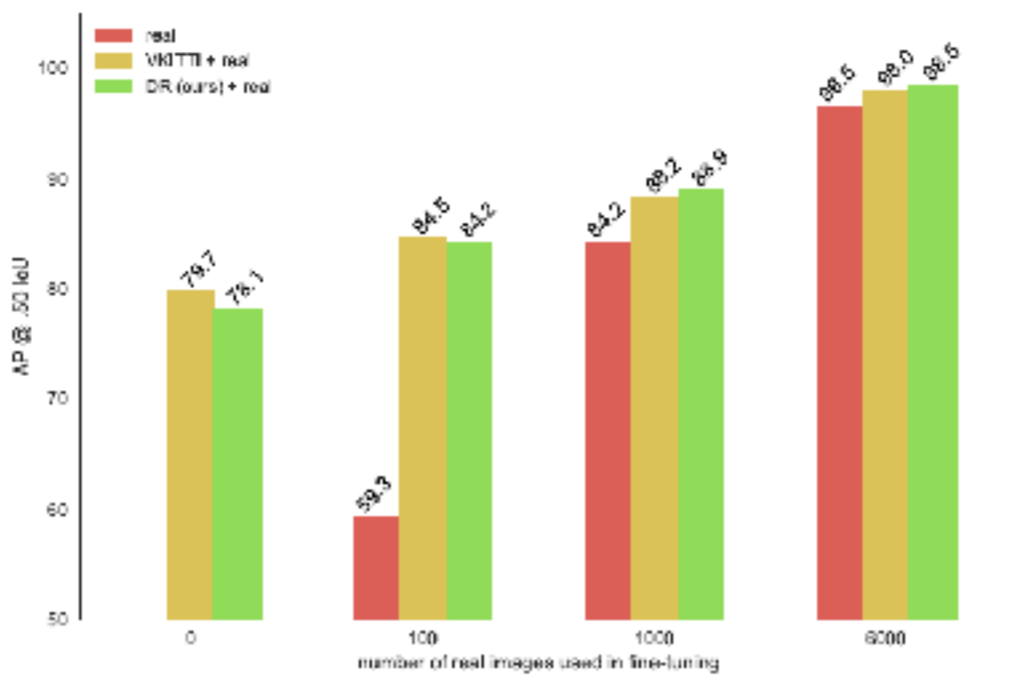




Car detection

VKITTI

domain rand



data generation

Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization, NVIDIA

Dynamics randomization

EPOpt: Learning Robust Neural Network Policies Using Model Ensembles

Aravind Rajeswaran[1], Sarvjeet Ghotra[2], Balaraman Ravindran[3], Sergey Levine[4]
aravraj@cs.washington.edu, sarvjeet.13it236@nitk.edu.in,
ravi@cse.iitm.ac.in, svlevine@eecs.berkeley.edu
[1] University of Washington Seattle
[2] NITK Surathkal
[3] Indian Institute of Technology Madras
[4] University of California Berkeley

Ideas:

- Consider a distribution over simulation models instead of a single one for learning policies robust to modeling errors that work well under many ``worlds”. Hard model mining
- Progressively bring the simulation model distribution closer to the real world.

Policy Search under model distribution

Learn a policy that performs best in expectation over MDPs in the source domain distribution:

$$\boxed{\mathbb{E}_{p\sim\mathcal{P}}}\left[\mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{T-1}\gamma^t r_t(s_t,a_t)\,\middle|\,p\right]\right]$$

p: simulator parameters

Policy Search under model distribution

Learn a policy that performs best in expectation over MDPs in the source domain distribution:

$$\mathbb{E}_{p\sim\mathcal{P}}\left[\mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{T-1}\gamma^t r_t(s_t,a_t)\,\middle|\,p\right]\right]$$

p: simulator parameters

Hard world model mining

Learn a policy that performs best in expectation over the worst \epsilon-percentile of MDPs in the source domain distribution

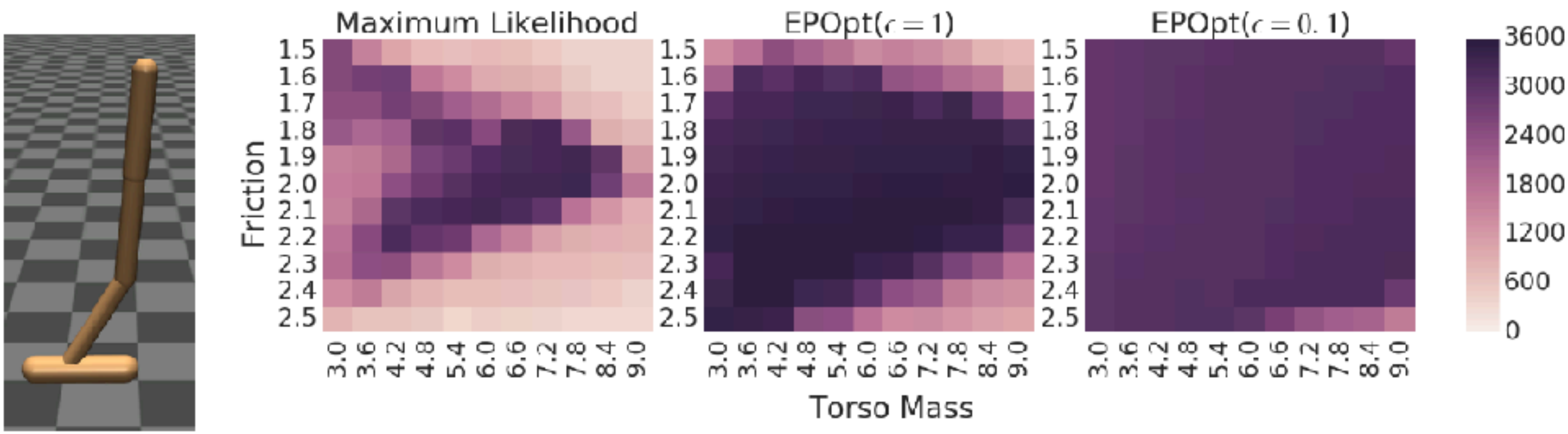
$$\max_{\theta,y}\int_{\mathcal{F}(\theta)}\eta_{\mathcal{M}}(\theta,p)\mathcal{P}(p)dp \qquad s.t.\quad \mathbb{P}\left(\eta_{\mathcal{M}}(\theta,P)\leq y\right)=\epsilon$$

Hard model mining

Algorithm 1: EPOpt–ϵ for Robust Policy Search

Input: ψ, θ_0, $niter$, N, ϵ
for *iteration* $i = 0, 1, 2, \ldots niter$ **do**
    for $k = 1, 2, \ldots N$ **do**
        sample model parameters $p_k \sim \mathcal{P}_\psi$
        sample a trajectory $\tau_k = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{T-1}$ from $\mathcal{M}(p_k)$ using policy $\pi(\theta_i)$
    end
    compute $Q_\epsilon = \epsilon$ percentile of $\{R(\tau_k)\}_{k=1}^{N}$
    select sub-set $\mathbb{T} = \{\tau_k : R(\tau_k) \leq Q_\epsilon\}$
    Update policy: $\theta_{i+1} = \text{BatchPolOpt}(\theta_i, \mathbb{T})$
end

Hard model mining results



Hard world mining results in policies with high reward over wider range of parameters

Adapting the source domain distribution

Sample a set of simulation parameters from a sampling distribution S.

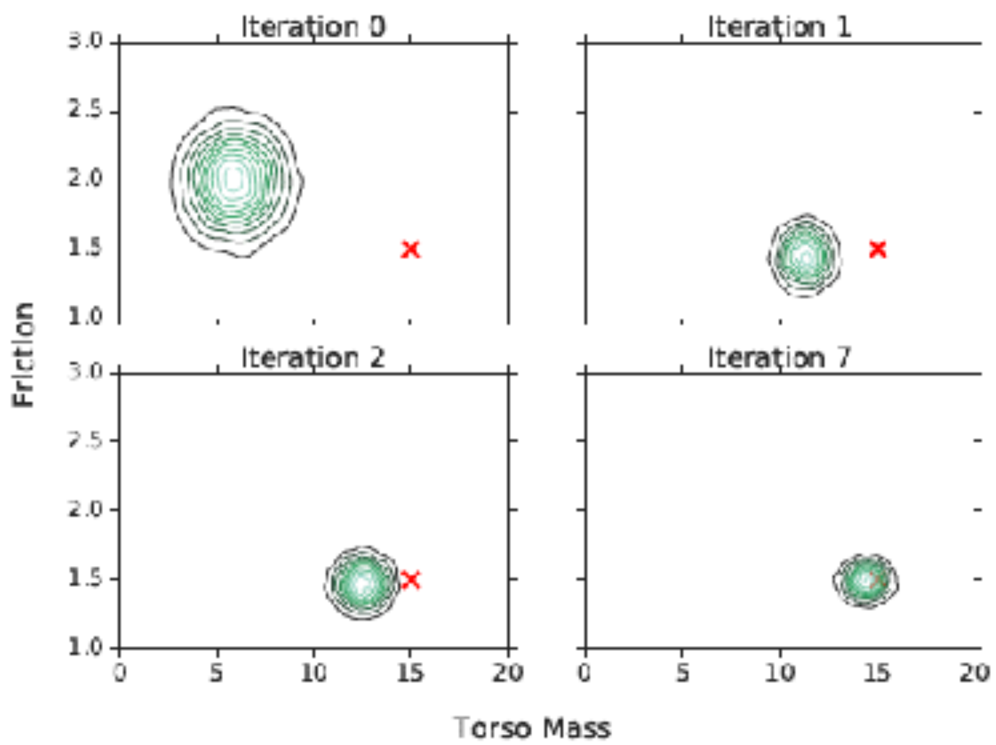
Posterior of parameters p_i:

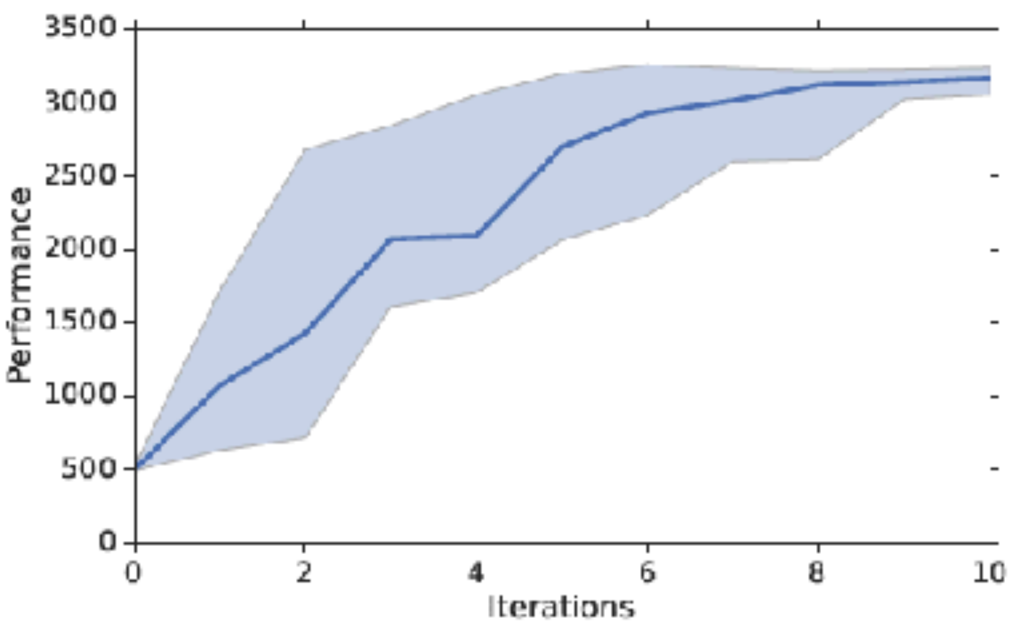
$$\mathbb{P}(p_i|\tau_k) \propto \prod_t \mathbb{P}(S_{t+1} = s_{t+1}^{(k)}|s_t^{(k)}, a_t^{(k)}, p_i) \times \frac{\mathbb{P}_P(p_i)}{\mathbb{P}_S(p_i)}$$

Fit a Gaussian model over simulator parameters based on posterior weights of the samples

fit of simulation parameter samples: how probable is an observed target state-action trajectory, the more probable the more we prefer such simulation model

Source Distribution Adaptation

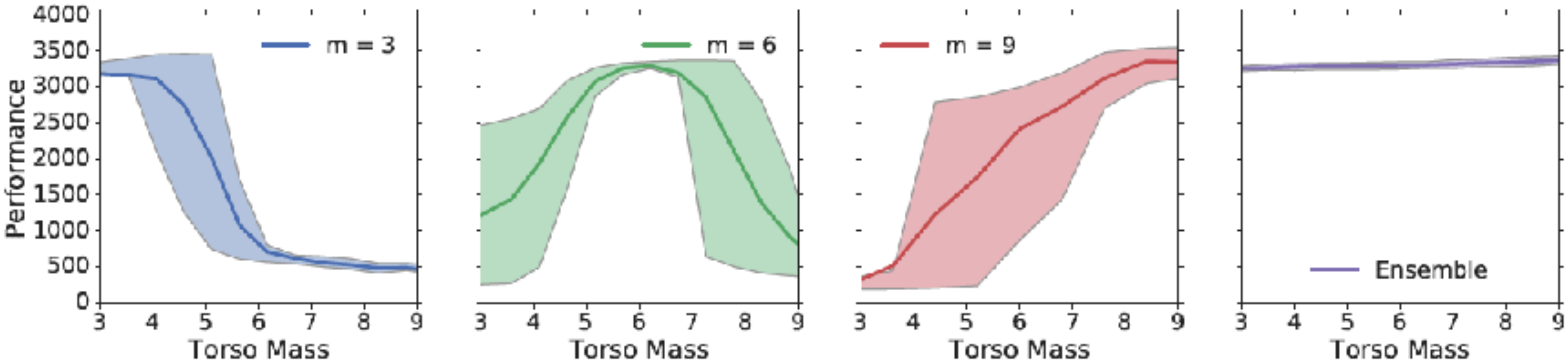




Performance on hopper policies

trained on single source domains

trained on Gaussian distribution of mean mass 6 and standard deviation 1.5



Driving Policy Transfer via Modularity and Abstraction

Matthias Müller
Visual Computing Center
KAUST, Saudi Arabia

Alexey Dosovitskiy
Intelligent Systems Lab
Intel Labs, Germany

Bernard Ghanem
Visual Computing Center
KAUST, Saudi Arabia

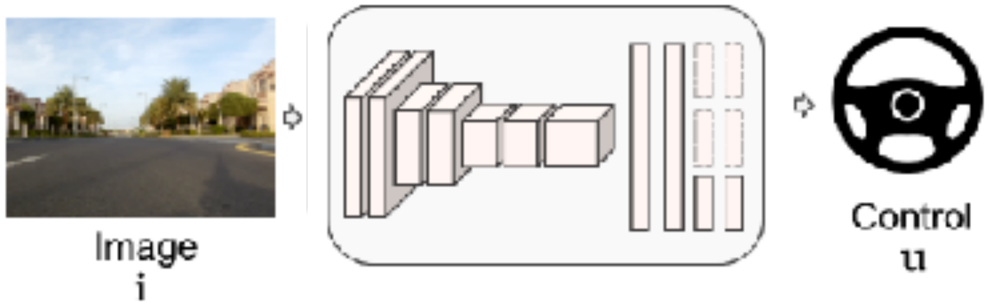
Vladlen Koltun
Intelligent Systems Lab
Intel Labs, USA

Idea: the driving policy is not directly exposed to raw perceptual input or low-level vehicle dynamics.

Main idea

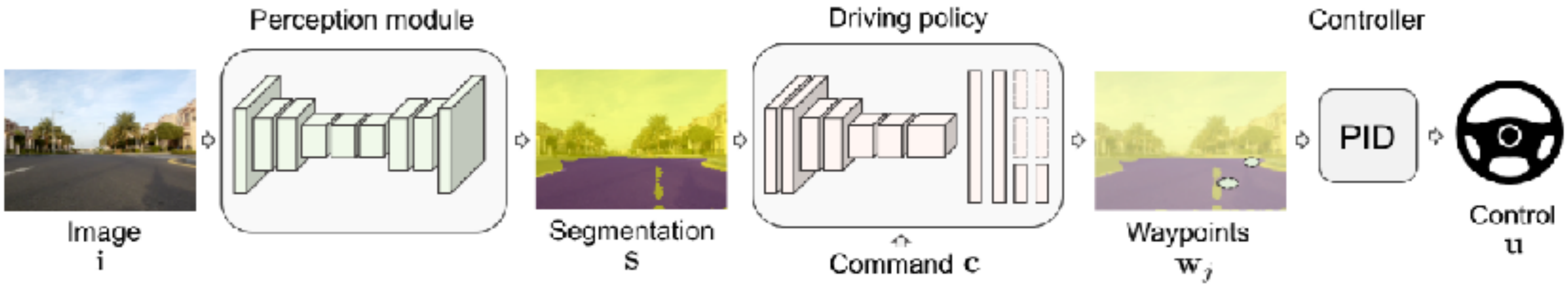
pixels to steering wheel learning is not SIM2REAL transferable

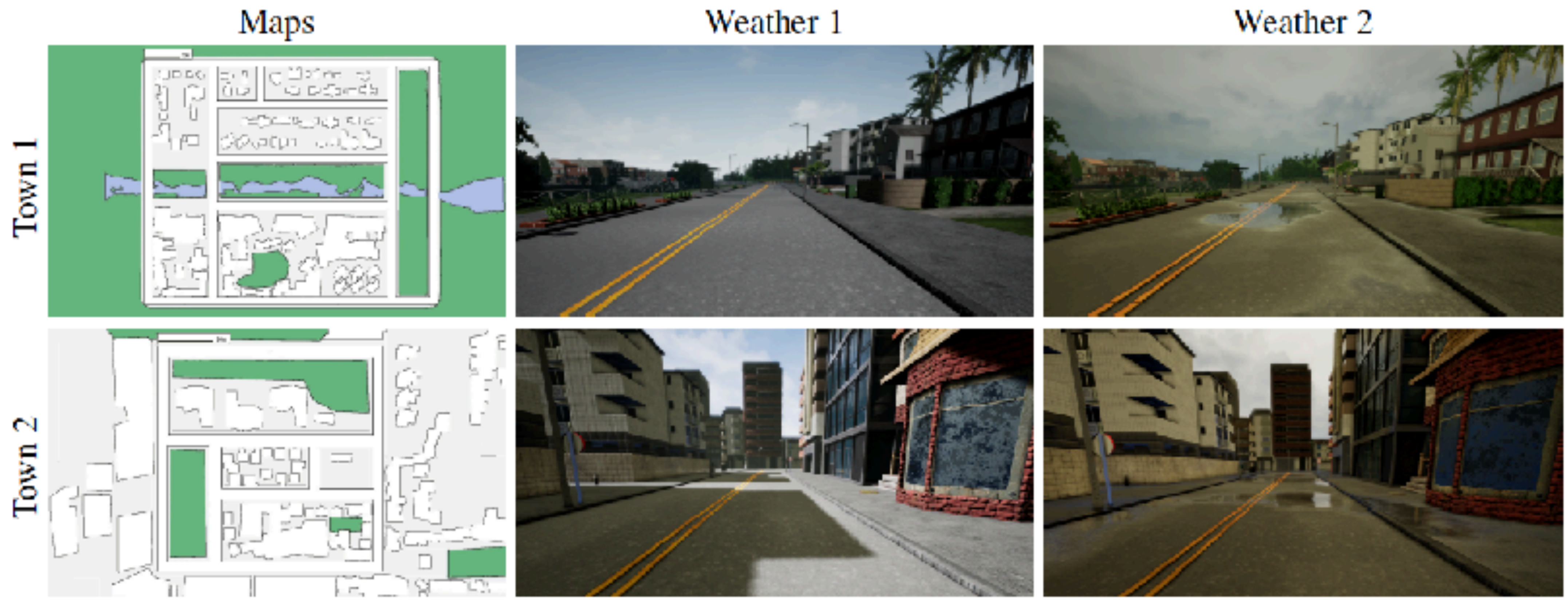
- textures/car dynamics mismatch



label maps to waypoint learning is SIM2REAL transferable

- label maps are similar between SIM and REAL and a low-level controller will take the car from waypoint to waypoint





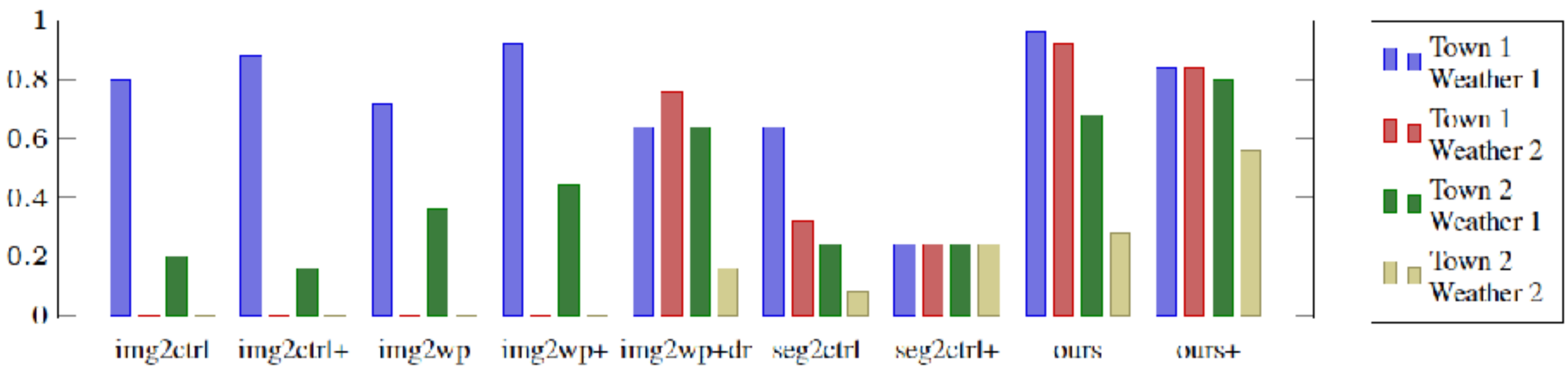


Figure 4: Quantitative evaluation of goal-directed navigation in simulation. We report the success rate over 25 navigation trials in four town-weather combinations. The models have been trained in Town 1 and Weather 1. The evaluated models are: *img2ctrl* – predicting low-level control from color images; *img2wp* – predicting waypoints from color images; *seg2ctrl* – predicting low-level control from the segmentation produced by the perception module; *ours* – predicting waypoints from the segmentation produced by the perception module. Suffix '+' denotes models trained with data augmentation, and '+dr' denotes the model trained with domain ramdomization.

Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Maximum Entropy Reinforcement Learning

CMU 10703

Katerina Fragkiadaki

Parts of slides borrowed from Russ Salakhutdinov, Rich Sutton, David Silver

RL objective

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t} R(s_t, a_t) \right]$$

MaxEntRL objective

Promoting stochastic policies

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=1}^{T} \underbrace{R(s_t, a_t)}_{\text{reward}} + \alpha \underbrace{\mathrm{H}(\pi(\cdot \mid s_t))}_{\text{entropy}} \right]$$

Why?

- Better exploration
- Learning alternative ways of accomplishing the task
- Better generalization, e.g., in the presence of obstacles a stochastic policy may still succeed.

Principle of Maximum Entropy

Policies that generate similar rewards, should be equally probable.

We do not want to commit to one policy over the other.

Why?

- Better exploration
- Learning alternative ways of accomplishing the task
- Better generalization, e.g., in the presence of obstacles a stochastic policy may still succeed.

Haarnoja et al., Reinforcement Learning with Deep Energy-Based Policies

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')\big(R - V(s_i;\theta'_v) + \beta \nabla_{\theta'} H(\pi(s_t;\theta'))\big)$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

“We also found that adding the entropy of the policy π to the objective function improved exploration by discouraging premature convergence to suboptimal deterministic policies. This technique was originally proposed by (Williams & Peng, 1991)”

Mnih et al., Asynchronous Methods for Deep Reinforcement Learning

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// *Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$*
// *Assume thread-specific parameter vectors θ' and θ'_v*
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t | s_t; \theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i | s_i; \theta')\big(R - V(s_i; \theta'_v) + \beta \nabla_{\theta'} H(\pi(s_t; \theta'))\big)$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i; \theta'_v)\right)^2 / \partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

This is just a regularization: such gradient just maximizes entropy of the current time step, not of future timesteps.

Mnih et al., Asynchronous Methods for Deep Reinforcement Learning

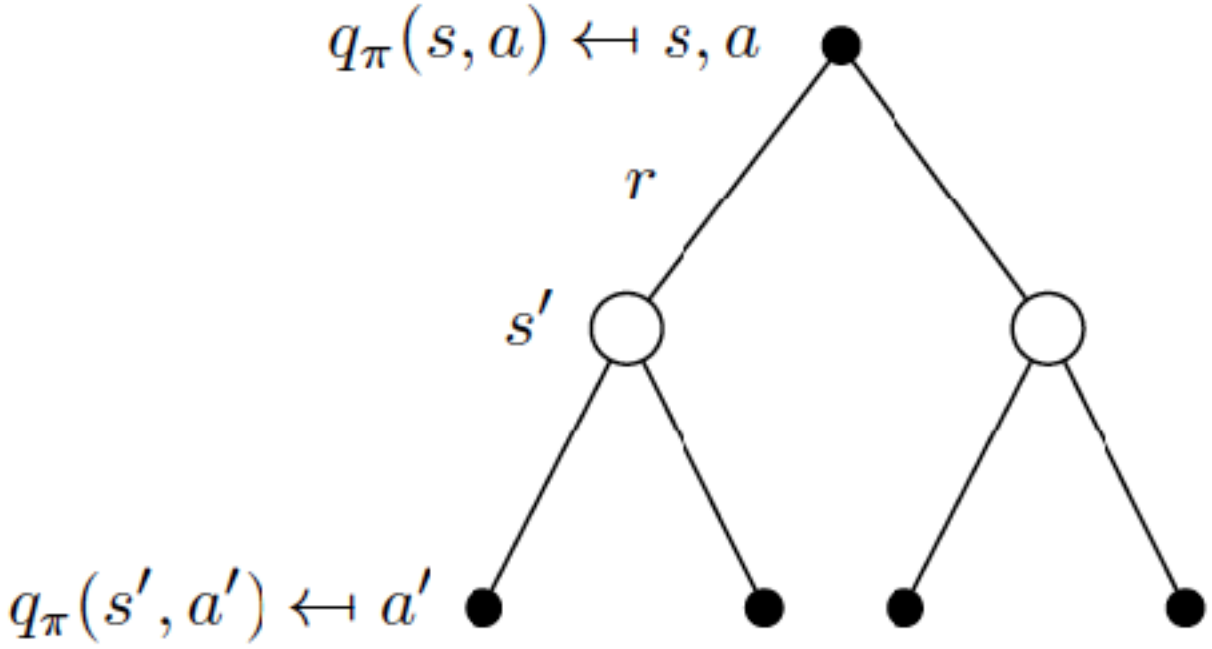
MaxEntRL objective

Promoting stochastic policies

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=1}^{T} \underbrace{R(s_t, a_t)}_{\text{reward}} + \alpha \underbrace{\mathrm{H}(\pi(\cdot \mid s_t))}_{\text{entropy}} \right]$$

How can we maximize such an objective?

Recall:Back-up Diagrams

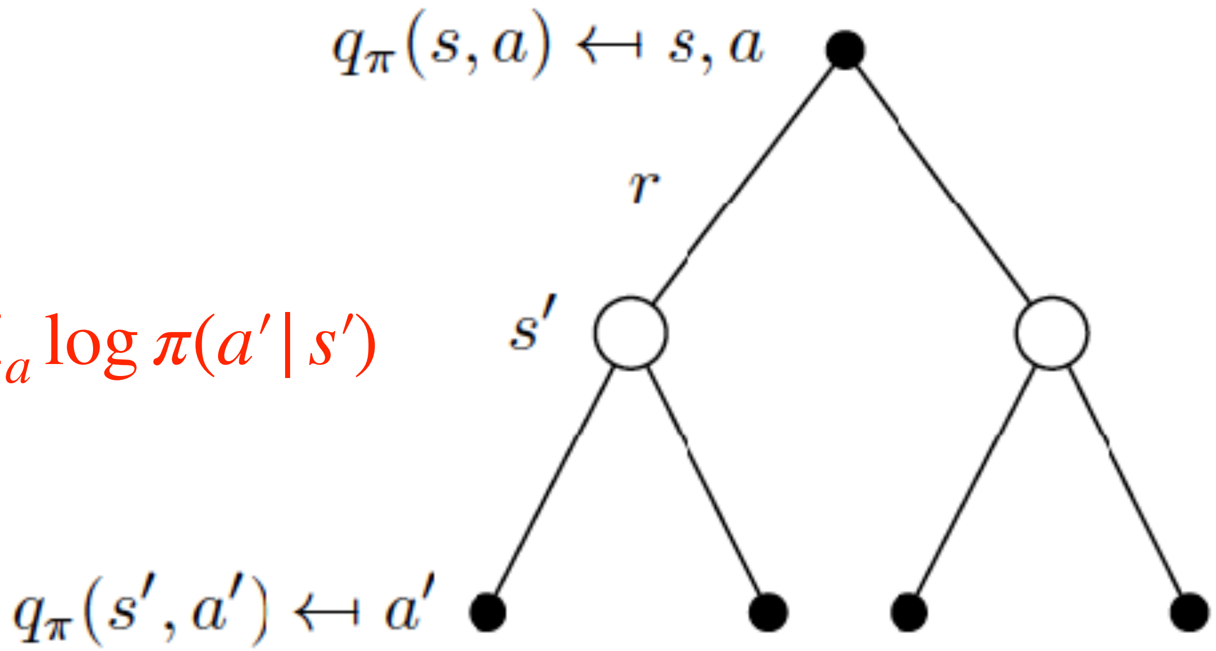


$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s') q_\pi(s',a')$$

Back-up Diagrams for MaxEnt Objective

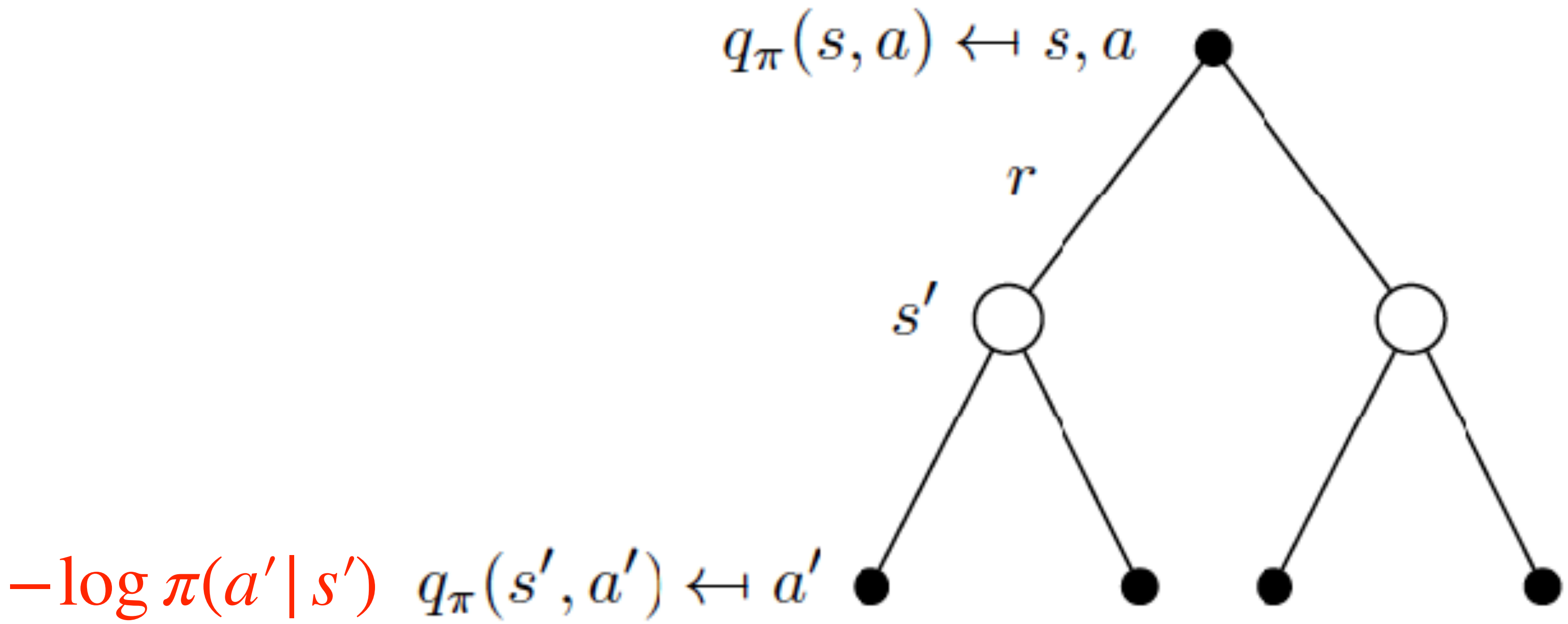
$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=1}^{T} \underbrace{R(s_t, a_t)}_{\text{reward}} + \alpha \underbrace{\mathrm{H}(\pi(\cdot \mid s_t))}_{\text{entropy}}\right]$$

$$H(\pi(\cdot \mid s')) = -\mathbb{E}_a \log \pi(a' \mid s')$$



Back-up Diagrams for MaxEnt Objective

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=1}^{T} \underbrace{R(s_t, a_t)}_{\text{reward}} + \alpha \underbrace{\mathrm{H}(\pi(\cdot \mid s_t))}_{\text{entropy}}\right]$$



$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s, a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s')\big(q_\pi(s', a') - \log(\pi(a' \mid s'))\big)$$

(Soft) policy evaluation

Soft Bellman backup equation:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} T(s' \mid s, a') \sum_{a'} \pi(a' \mid s') \left(q_\pi(s',a') - \log(\pi(a' \mid s') \right)$$

Bellman backup equation:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s, a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s') q_\pi(s',a')$$

Soft Bellman backup update operator-unknown dynamics:

$$Q(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}, a_{t+1}} \left[Q(s_{t+1}, a_{t+1}) - \log \pi(a_{t+1} \mid s_{t+1}) \right]$$

Bellman backup update operator-unknown dynamics:

$$Q(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}, a_{t+1}} Q(s_{t+1}, a_{t+1})$$

Soft Bellman backup update operator is a contraction

$$Q(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}, a_{t+1}} \left[Q(s_{t+1}, a_{t+1}) - \log \pi(a_{t+1} \mid s_{t+1}) \right]$$

$$\begin{aligned} Q(s_t, a_t) &\leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho}[\mathbb{E}_{a_{t+1} \sim \pi}[Q(s_{t+1}, a_{t+1}) - \log \pi(a_{t+1} \mid s_{t+1})]] \\ &\leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho, a_{t+1} \sim \pi} Q(s_{t+1}, a_{t+1}) + \gamma \mathbb{E}_{s_{t+1} \sim \rho} \mathbb{E}_{a_{t+1} \sim \pi}[-\log \pi(a_{t+1} \mid s_{t+1})] \\ &\leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho, a_{t+1} \sim \pi} Q(s_{t+1}, a_{t+1}) + \gamma \mathbb{E}_{s_{t+1} \sim \rho} H(\pi(\cdot \mid s_{t+1})) \end{aligned}$$

Rewrite the reward as:

$$r_{soft}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho} H(\pi(\cdot \mid s_{t+1}))$$

Then we get the old Bellman operator, which we know is a contraction

Soft Bellman backup update operator

$$Q(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}, a_{t+1}} \left[Q(s_{t+1}, a_{t+1}) - \log \pi(a_{t+1} \mid s_{t+1}) \right]$$

$$\begin{aligned} Q(s_t, a_t) &\leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho}[\mathbb{E}_{a_{t+1} \sim \pi}[Q(s_{t+1}, a_{t+1}) - \log \pi(a_{t+1} \mid s_{t+1})]] \\ &\leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho, a_{t+1} \sim \pi} Q(s_{t+1}, a_{t+1}) + \gamma \mathbb{E}_{s_{t+1} \sim \rho} \mathbb{E}_{a_{t+1} \sim \pi}[-\log \pi(a_{t+1} \mid s_{t+1})] \\ &\leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho, a_{t+1} \sim \pi} Q(s_{t+1}, a_{t+1}) + \gamma \mathbb{E}_{s_{t+1} \sim \rho} H(\pi(\cdot \mid s_{t+1})) \end{aligned}$$

We know that:

$$Q(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho}[V(s_{t+1})]$$

Which means that:

$$V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - \log \pi(a_t \mid s_t)]$$

Soft Policy Iteration

Soft policy iteration iterates between two steps:

1. Soft policy evaluation: Fix policy, apply Bellman backup operator till convergence

$$q_\pi(s,a) = r(s,a) + \mathbb{E}_{s',a'}\left(q_\pi(s',a') - \log(\pi(a' \mid s')\right)$$

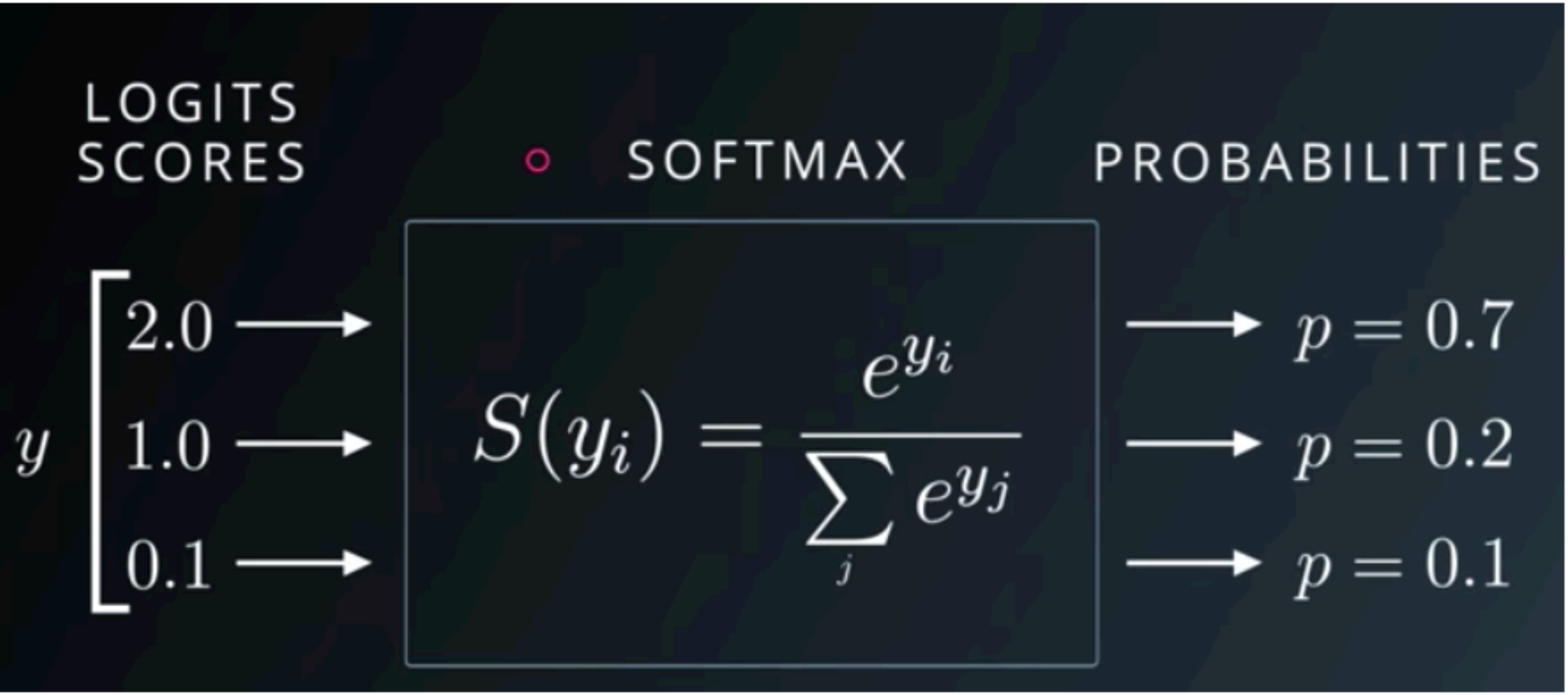
This converges to q_π

2. Soft policy improvement: Update the policy:

$$\pi' = \arg\min_{\pi_k \in \Pi} D_{KL}\left(\pi_k(\cdot \mid s_t) \,||\, \frac{\exp(\mathrm{Q}^\pi(\mathrm{s_t}, \cdot))}{Z^\pi(s_t)}\right)$$

Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

SoftMax


LOGITS
SCORES
SOFTMAX
PROBABILITIES
y
2.0
1.0
0.1
$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$
$p = 0.7$
$p = 0.2$
$p = 0.1$

Soft Policy Iteration

Soft policy iteration iterates between two steps:

1. Soft policy evaluation: Fix policy, apply Bellman backup operator till convergence

$$q_\pi(s,a) = r(s,a) + \mathbb{E}_{s',a'}\left(q_\pi(s',a') - \log(\pi(a' \mid s'))\right)$$

This converges to q_π

2. Soft policy improvement: Update the policy:

$$\pi' = \arg\min_{\pi_k \in \Pi} D_{KL}\left(\pi_k(\cdot \mid s_t) \,\middle|\middle|\, \frac{\exp(\mathrm{Q}^\pi(\mathrm{s_t}, \cdot))}{Z^\pi(s_t)}\right)$$

Leads to a sequence of policies with monotonically increasing soft q values

This so far concerns tabular methods. Next we will use function approximations for policy and action values

Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Soft Policy Iteration - Approximation

Use function approximations for policy, state and action value functions

$V_\psi(s_t) \qquad Q_\theta(s_t) \qquad \pi_\phi(a_t \mid s_t)$

1. Learning the state value function:

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t \sim \pi_\phi} \left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \right] \right)^2 \right]$$

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(\mathbf{s}_t) \left(V_\psi(\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, \mathbf{a}_t) + \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \right)$$

Soft Policy Iteration - Approximation

Use function approximations for policy, state and action value functions

$V_\psi(s_t)$ $Q_\theta(s_t)$ $\pi_\phi(a_t | s_t)$

2. Learning the state-action value function:

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) \right)^2 \right]$$

$$\hat{Q}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \, \mathbb{E}_{\mathbf{s}_{t+1} \sim p} \left[V_{\bar{\psi}}(\mathbf{s}_{t+1}) \right]$$

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\mathbf{a}_t, \mathbf{s}_t) \left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - r(\mathbf{s}_t, \mathbf{a}_t) - \gamma V_{\bar{\psi}}(\mathbf{s}_{t+1}) \right)$$

Soft Policy Iteration - Approximation

Use function approximations for policy, state and action value functions

$$V_\psi(s_t) \qquad Q_\theta(s_t, a_t) \quad \pi_\phi(a_t \mid s_t)$$

3. Learning the policy:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathrm{D_{KL}} \left(\pi_\phi(\,\cdot\,|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(Q_\theta(\mathbf{s}_t, \cdot)\right)}{Z_\theta(\mathbf{s}_t)} \right) \right]$$

$$Z_\theta(s_t) = \int_{\mathcal{A}} \exp(Q_\theta(s_t, a_t)) da_t$$

independent of \phi

$$\nabla_\phi J_\pi(\phi) = \nabla_\phi \mathbb{E}_{s_t \in D} \mathbb{E}_{a_t \sim \pi_\phi(a|s_t)} \log \frac{\pi_\phi(a_t \mid s_t)}{\frac{\exp(Q_\theta(s_t, a_t))}{Z_\theta(s_t)}}$$

$$\nabla_\phi J_\pi(\phi) = \nabla_\phi \mathbb{E}_{s_t \in D} \mathbb{E}_{a_t \sim \pi_\phi(a|s_t)} \log \frac{\pi_\phi(a_t \mid s_t)}{\exp(Q_\theta(s_t, a_t))}$$

The variable w.r.t. which we take gradient parametrizes the distribution inside the distribution.

Soft Policy Iteration - Approximation

Use function approximations for policy, state and action value functions

$$V_\psi(s_t) \qquad Q_\theta(s_t) \qquad \pi_\phi(a_t \mid s_t)$$

3. Learning the policy:

$$\nabla_\phi J_\pi(\phi) = \nabla_{\phi} \mathbb{E}_{s_t \in D} \mathbb{E}_{a_t \sim \pi_{\phi}(a \mid s_t)} \log \frac{\pi_\phi(a_t \mid s_t)}{\exp(Q_\theta(s_t, a_t))}$$

Reparametrization trick. The policy becomes a deterministic function of Gaussian random variables (fixed Gaussian distribution):

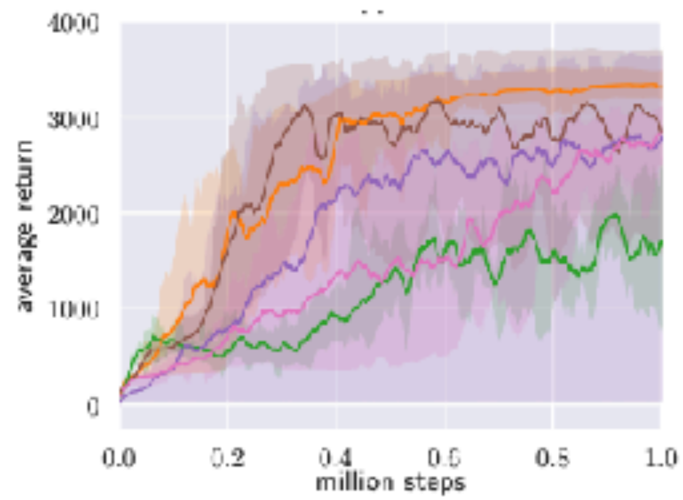
$$a_t = f_\phi(s_t, \epsilon) = \mu_\phi(s_t) + \epsilon \Sigma_\phi(s_t), \quad \epsilon \sim \mathcal{N}(0,I)$$

$$\nabla_\phi J_\pi(\phi) = \nabla_{\phi} \mathbb{E}_{s_t \in D, \epsilon \sim \mathcal{N}(0,I)} \log \frac{\pi_\phi(a_t \mid s_t)}{\exp(Q_\theta(s_t, a_t))}$$

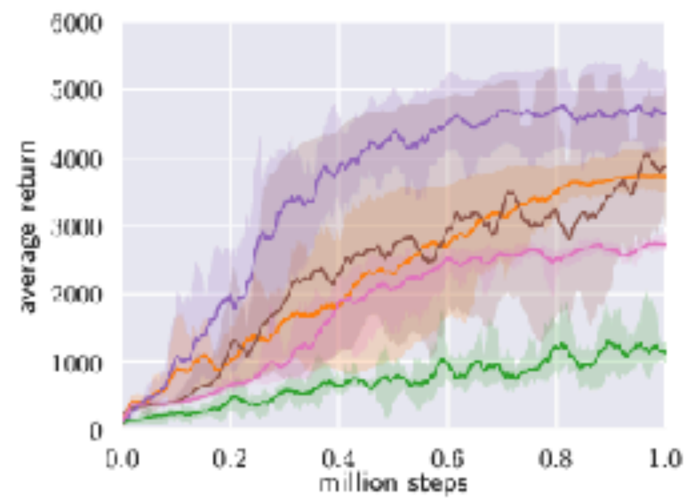
Soft Policy Iteration - Approximation

Algorithm 1 Soft Actor-Critic

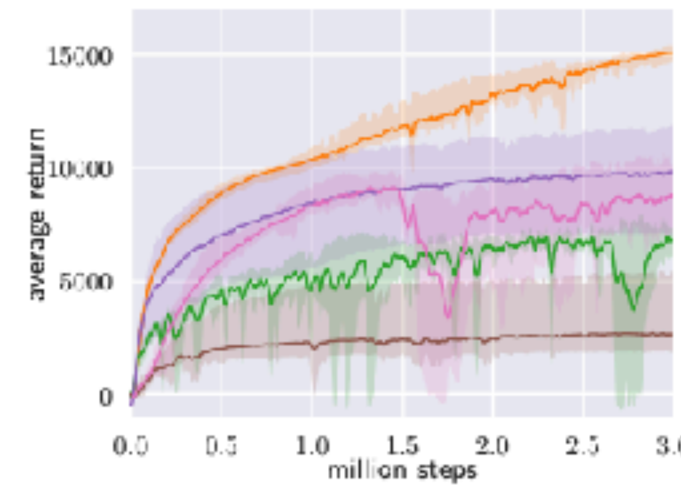
Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for



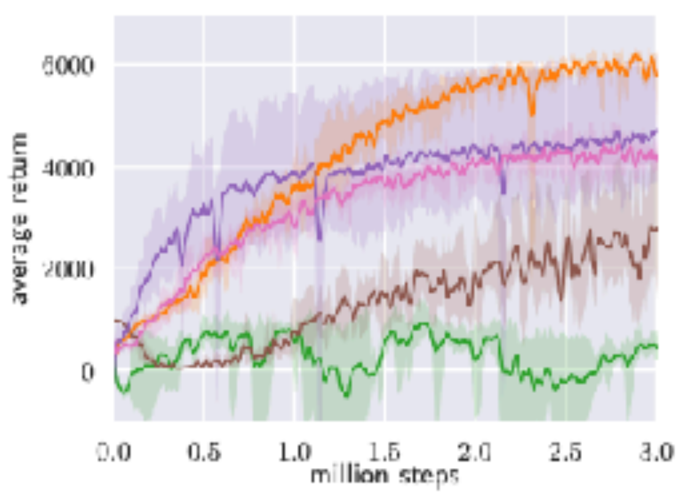
(a) Hopper-v1



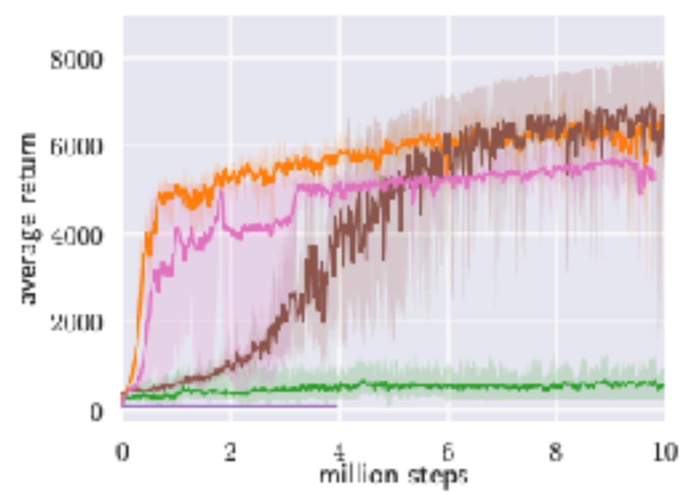
(b) Walker2d-v1



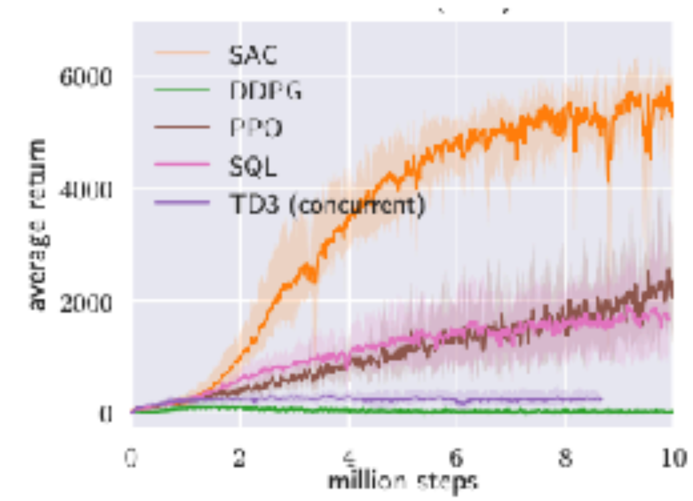
(c) HalfCheetah-v1



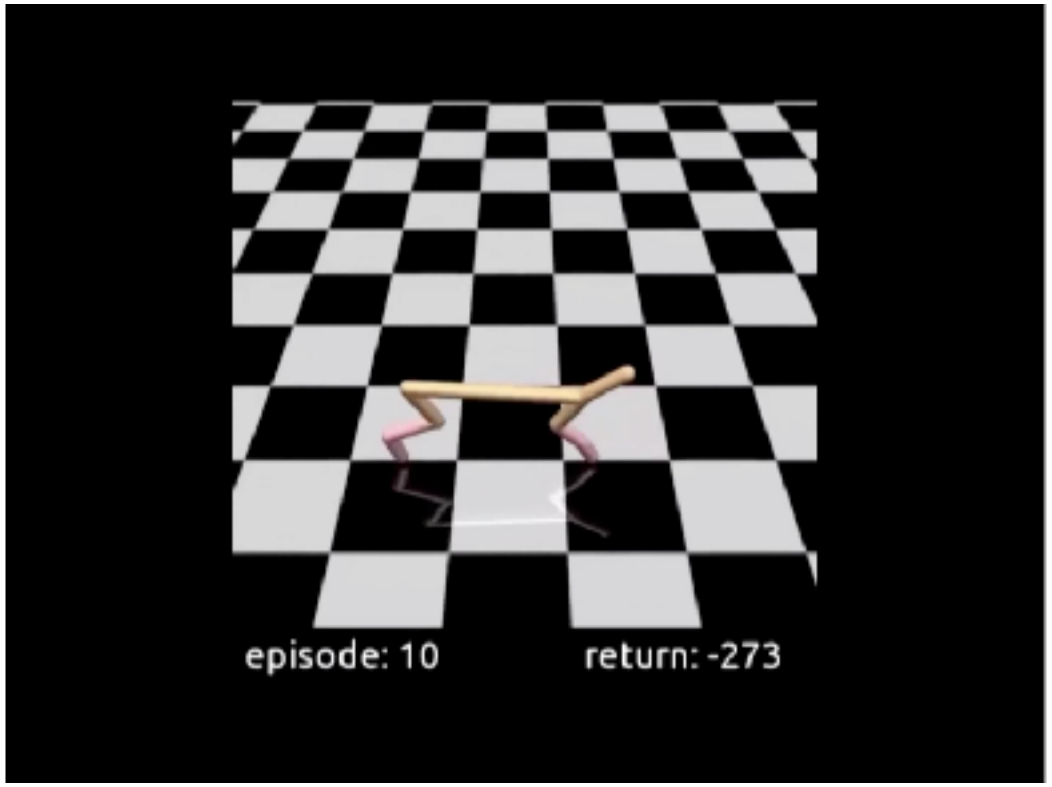
(d) Ant-v1



(e) Humanoid-v1



(f) Humanoid (rllab)



Composability of Maximum Entropy Policies

Imagine we want to satisfy two objectives at the same time, e.g., pick an object up while avoiding an obstacle. We would learn a policy to maximize the addition of the the corresponding reward functions:

$$r_{\mathcal{C}}(\mathbf{s}, \mathbf{a}) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} r_i(\mathbf{s}, \mathbf{a})$$

MaxEnt policies permit to obtain the resulting policy's optimal Q by simply adding the constituent Qs:

$$Q^*_{\mathcal{C}}(\mathbf{s}, \mathbf{a}) \approx Q_{\Sigma}(\mathbf{s}, \mathbf{a}) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} Q^*_i(\mathbf{s}, \mathbf{a})$$

We can theoretically bound the suboptimality of the resulting policy w.r.t. the policy trained under the addition of rewards. We cannot do this for deterministic policies.

Composable Deep Reinforcement Learning for Robotic Manipulation, Haarnoja et al.

Composability of Maximum Entropy Policies

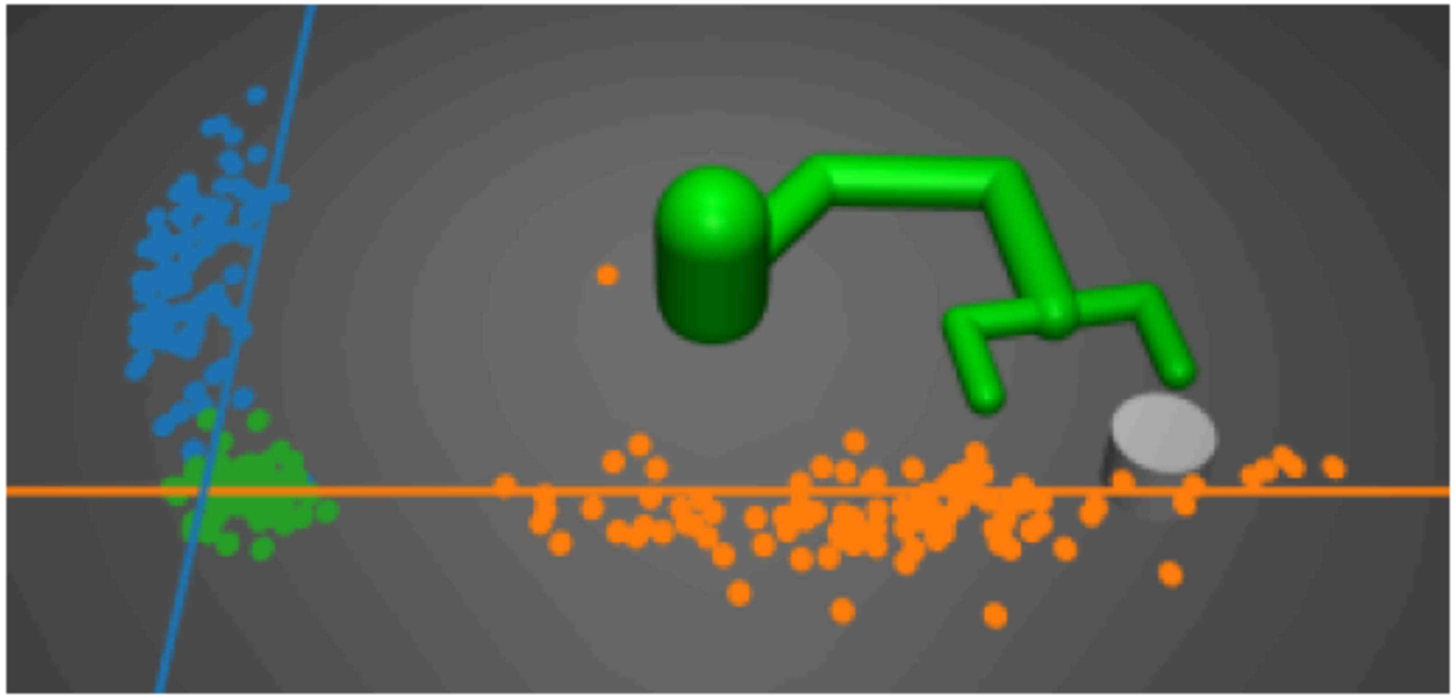


Fig. 2. Two independent policies are trained to push the cylinder to the orange line and blue line, respectively. The colored circles show samples of the final location of the cylinder for the respective policies. When the policies are combined, the resulting policy learns to push the cylinder to the lower intersection of the lines (green circle indicates final location). No additional samples from the environment are used to train the combined policy. The combined policy learns to satisfy both original goals, rather than simply averaging the final cylinder location.

Composable Deep Reinforcement Learning for Robotic Manipulation, Haarnoja et al.

Composable Deep Reinforcement Learning for Robotic Manipulation

Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou,
Murtaza Dalal, Pieter Abbeel, and Sergey Levine

Berkeley Artificial Intelligence Research
UC Berkeley

https://youtu.be/wdexoLS2cWU