Domain Adaptation and Domain Randomization for Sim2Real RL

Xiong-Hui Chen

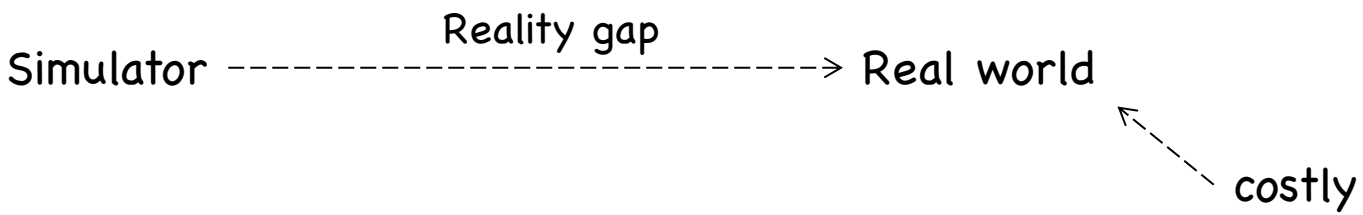
南京大學 NANJING UNIVERSITY

LAMDA Learning And Mining from DatA

Table of Contents

1. **Sim2Real Transfer for Reinforcement Learning**
2. Domain Randomization for Sim2Real RL
3. Domain Adaptation for Sim2Real RL
4. Our work: Cross-Modal Domain Adaptation with Sequential structure (CODAS)

Sim2Real Transfer for Reinforcement Learning



Setting:

1. Target: We would like to train an agent to maximize the rewards in the real world.
2. Query online samples in the real world are costly.
3. We have a simulator that is used to simulate the real world.
4. But the simulator has reality gaps to the real world.

Sim2Real Transfer for Reinforcement Learning

Derived Setting:

1. Target: We would like to train an agent to maximize the rewards in the real world.
2. Query **online samples** in the real world are **costly**.
    1. Different attitudes to "costly".
        1. Allow a few online samples (e.g., several steps or one episode) before deployment.
        2. Allow zero extra querying before deployment.
    2. Other sources of data?
        1. We have an offline dataset (collected by a human/rule-based policy).
        2. We have no extra real-world dataset.
3. We have **a simulator** that is used to simulate the real world.
    1. What is the ability of the simulator?
        1. Configurable (e.g., generate dynamics models with different friction coefficients)?
        2. Renderable (e.g., state -> image)?
4. But the simulator has **reality gaps** to the real world.
    1. The source of the gaps.
        1. Observations
        2. Dynamics models
        3. Reward function (e.g., different tasks)

Table of Contents

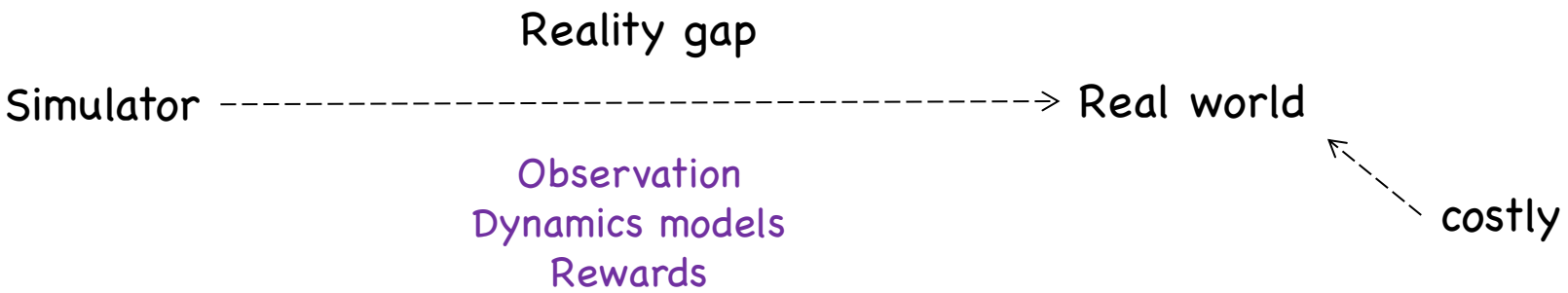
1. Sim2Real Transfer for Reinforcement Learning
2. **Domain Randomization for Sim2Real RL**
3. Domain Adaptation for Sim2Real RL
4. Our work: Cross-Modal Domain Adaptation with Sequential structure (CODAS)

Domain Randomization for Sim2Real RL

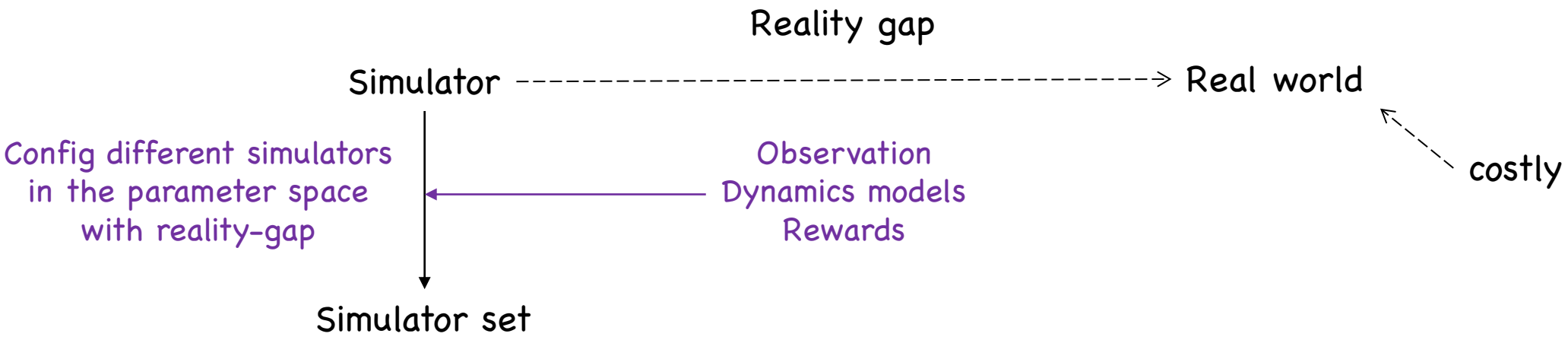
Derived Setting:

1. Target: We would like to train an agent to maximize the rewards in the real world.
2. Query **online samples** in the real world are **costly**.
    1. [-] Different attitudes to "costly".
        1. Allow a few online samples (e.g., several steps or one episode) before deployment.
        2. Allow zero extra querying before deployment.
    2. [-] Other sources of data?
        1. We have an offline dataset (collected by a human/rule-based policy).
        2. We have no extra real-world dataset.
3. We have **a simulator** that is used to simulate the real world.
    1. What is the ability of the simulator?
        1. [√] Configurable (e.g., generate dynamics models with different friction coefficients)?
        2. [-] Renderable (e.g., state -> image)?
4. But the simulator has **reality gaps** to the real world.
    1. [√] The source of the gaps.
        1. Observations
        2. Dynamics models
        3. Reward function (e.g., different tasks)

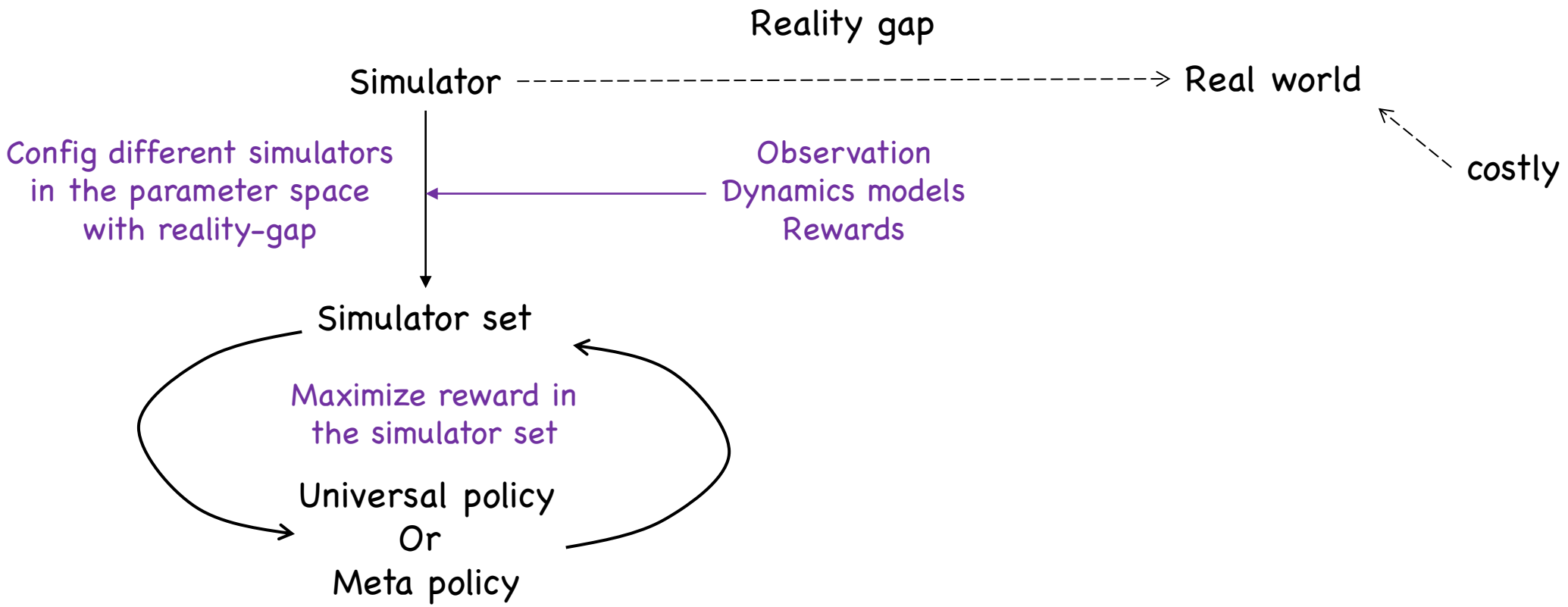
Domain Randomization for Sim2Real RL



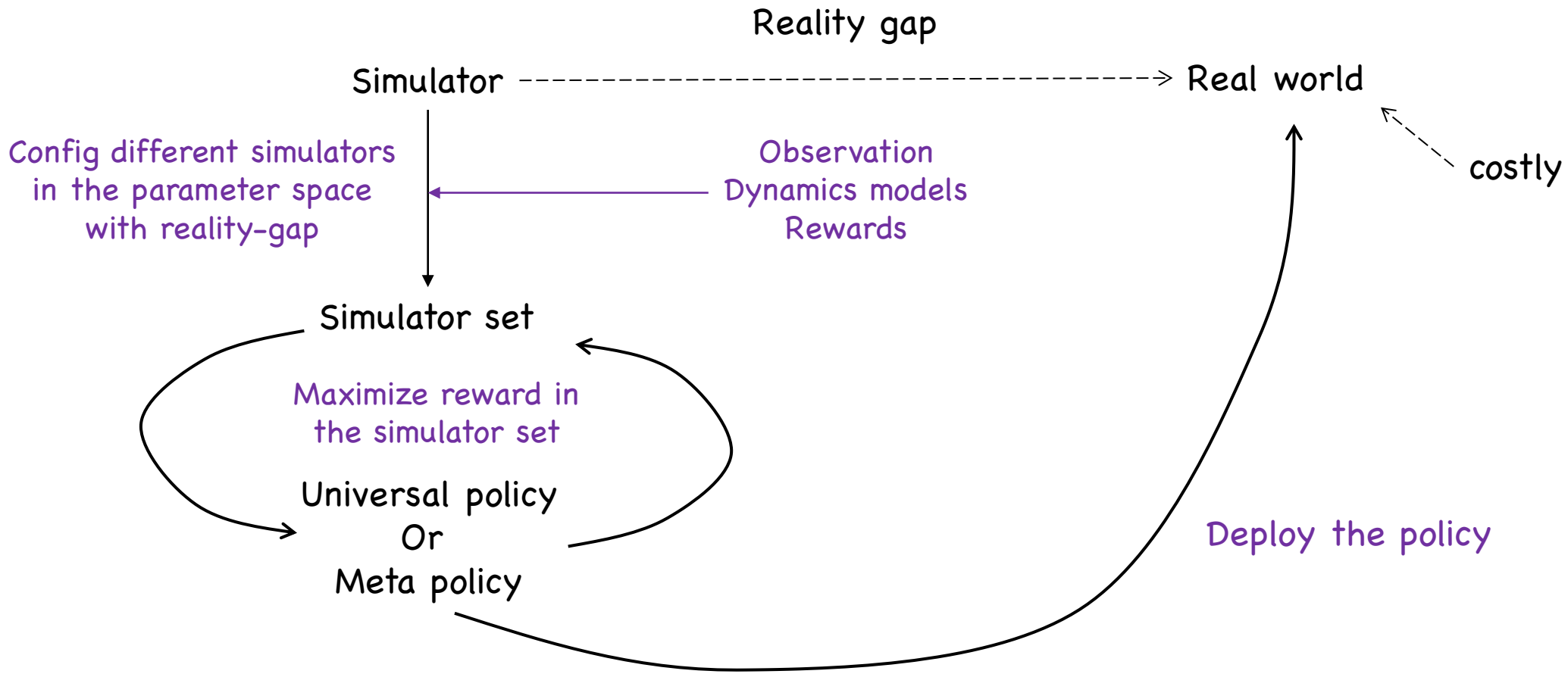
Domain Randomization for Sim2Real RL



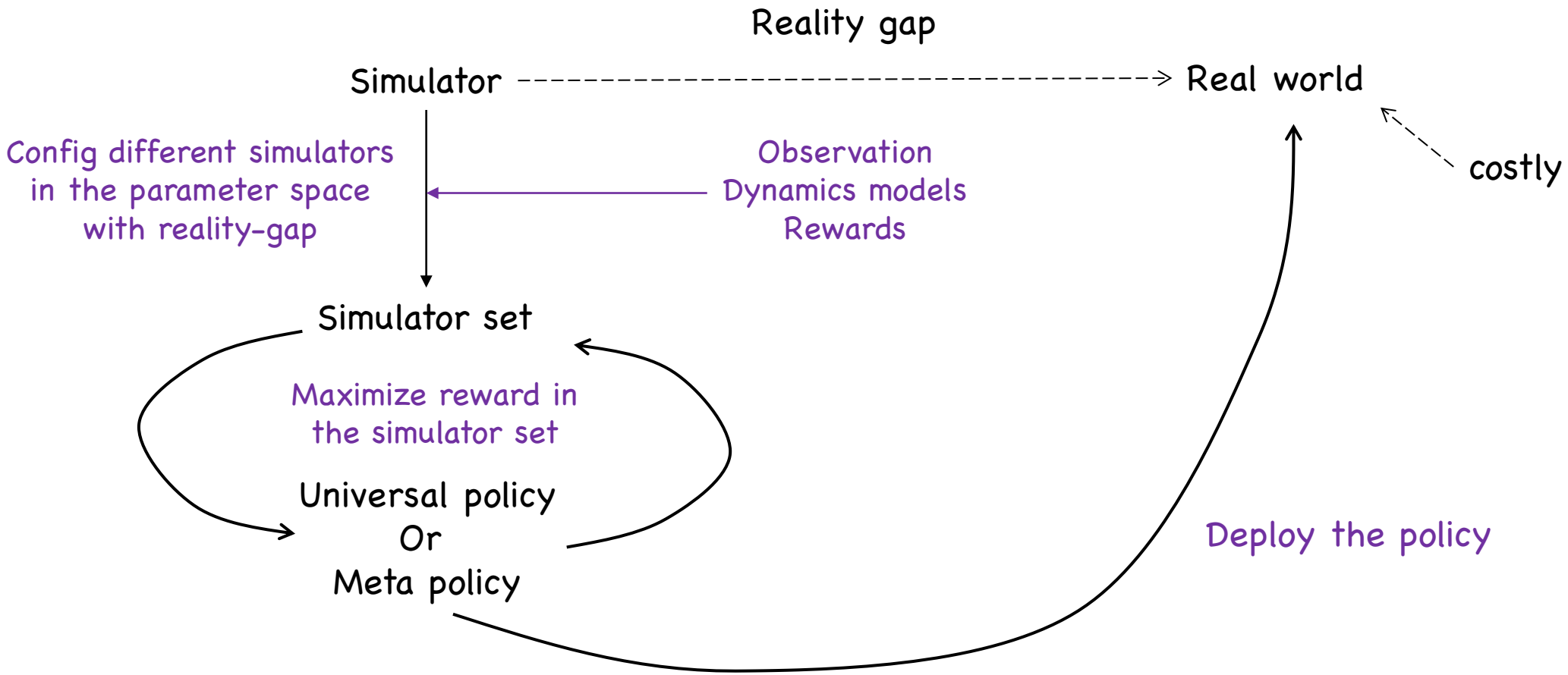
Domain Randomization for Sim2Real RL



Domain Randomization for Sim2Real RL



Domain Randomization for Sim2Real RL

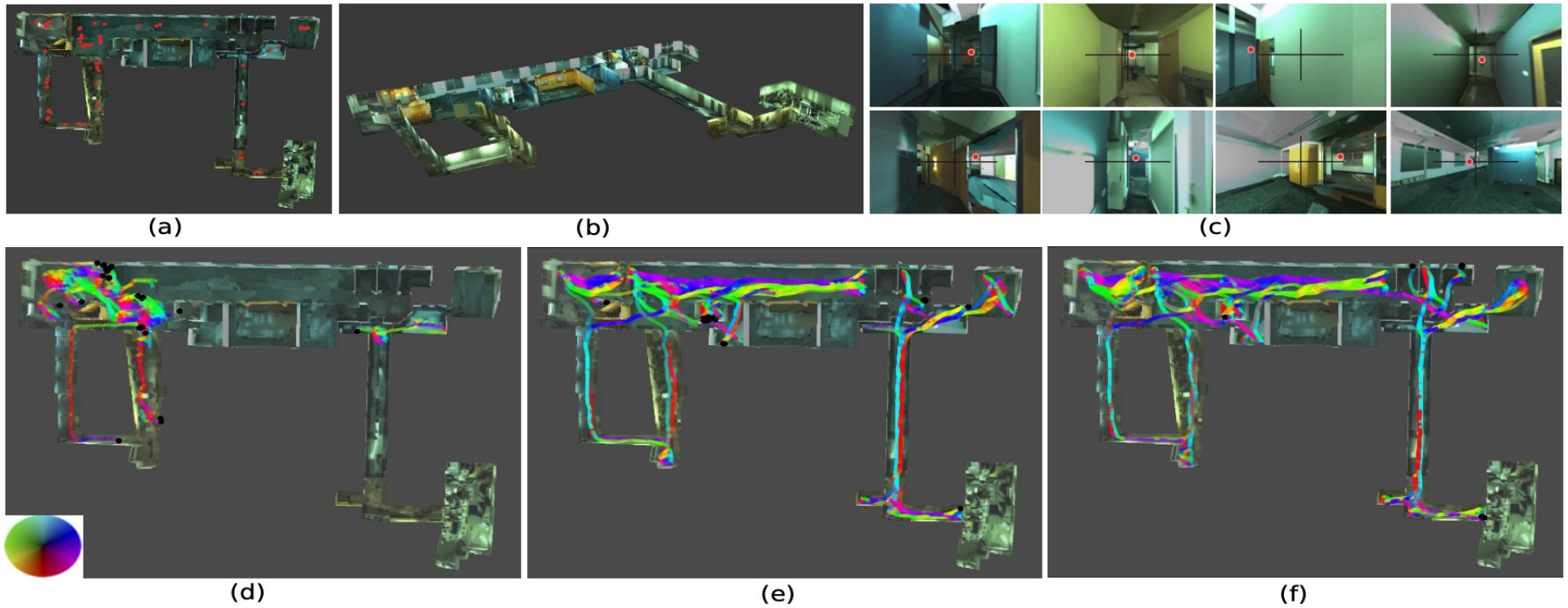


implicit assumption: the environment parameters of the simulators cover the one in the real world.

-> the meta-policy or universal policy would make reasonable decisions in the real world

CAD2RL: Real single-image flight without a single real image

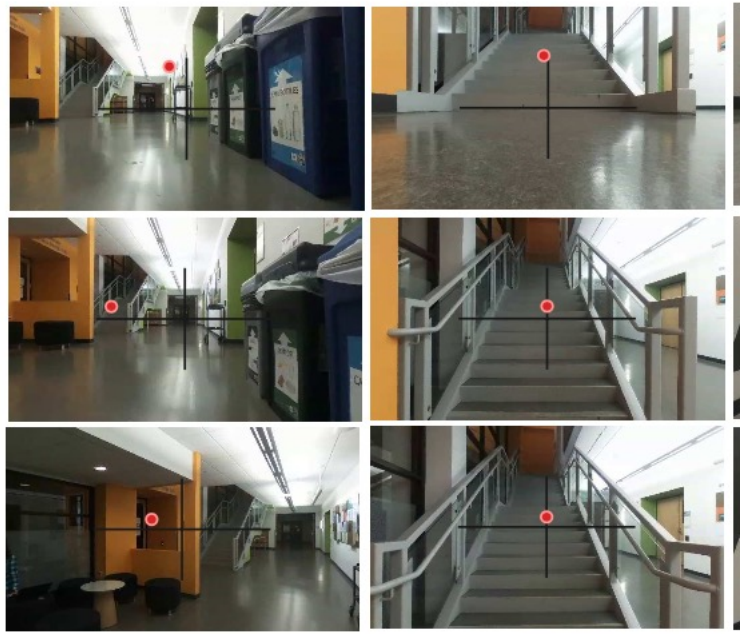
Indoor navigation and collision avoidance



CAD2RL: Real single-image flight without a single real image

Learning Architecture

- Policy representation: $\pi(a|I) \rightarrow v$



-> velocity command

CAD2RL: Real single-image flight without a single real image

Learning Architecture

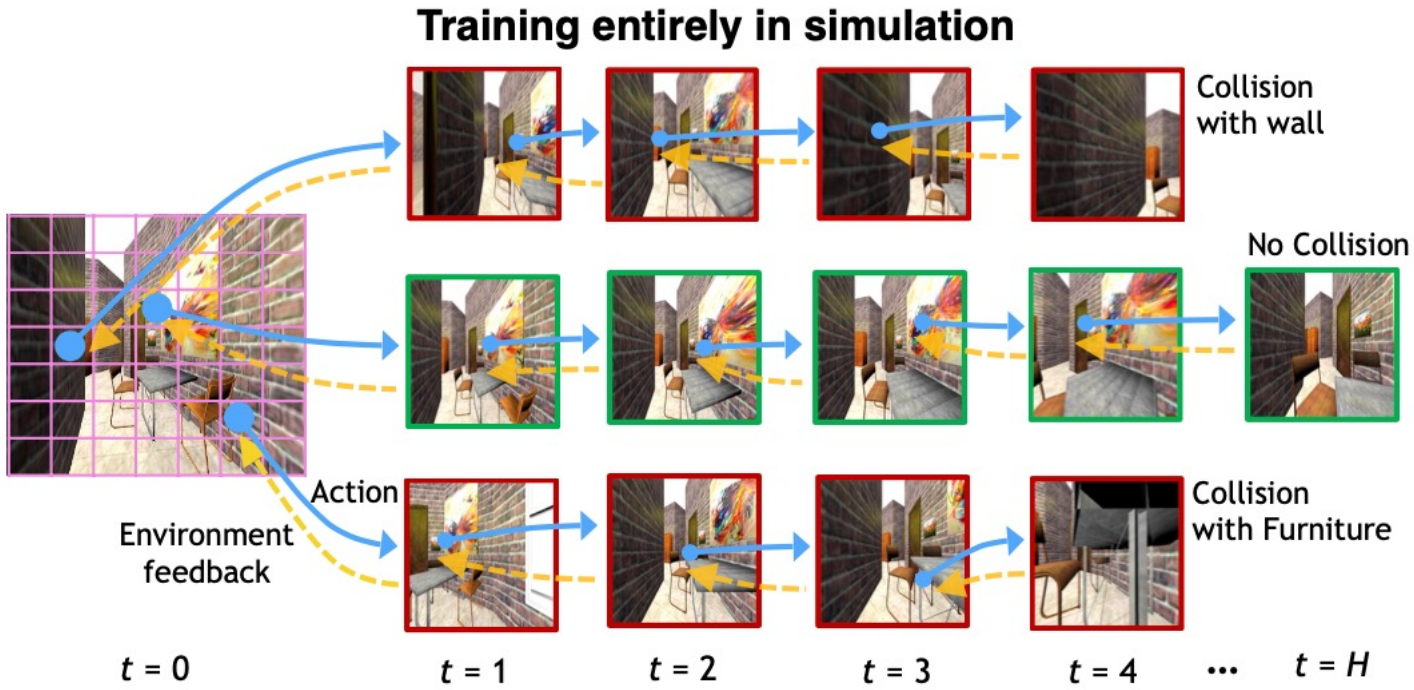
- Policy Learning: $\pi(a|I)$
  - Q(l|I,a)
    - Pretrain -> free space detection

In order to initialize our model with a reasonable starting policy, we use a heuristic pre-training phase based on free space detection. In this pretraining phase, the model is trained to predict $P(l|\mathbf{I}_t, \mathbf{a}_t)$, where $l \in \{0, 1\}$ is a label that indicates whether a collision detection raycast in the direction $\mathbf{v}_t$ corresponding to $\mathbf{a}_t$ intersects an obstacle. The raycast has a fixed length of 1 meter. This is essentially equivalent to thresholding the depth map by one meter. This initialization phase roughly corresponds to the assumption that the vehicle will maintain a predefined constant velocity $\mathbf{v}_t$. The model, which is represented by a fully convolutional neural network as described in Section III-D, is trained to label each bin with the collision label l, analogously to recent work in image segmentation [9]. The labels are obtained from our simulation engine, as described in Section IV.

CAD2RL: Real single-image flight without a single real image

Learning Architecture

- Policy Learning: $\pi(a|I)$
  - Q(l|I,a)
    - Pretrain -> free space detection
    - $\pi(I) = \arg\max Q(I, a)$
    - RL: update Q via MC rollout follow π

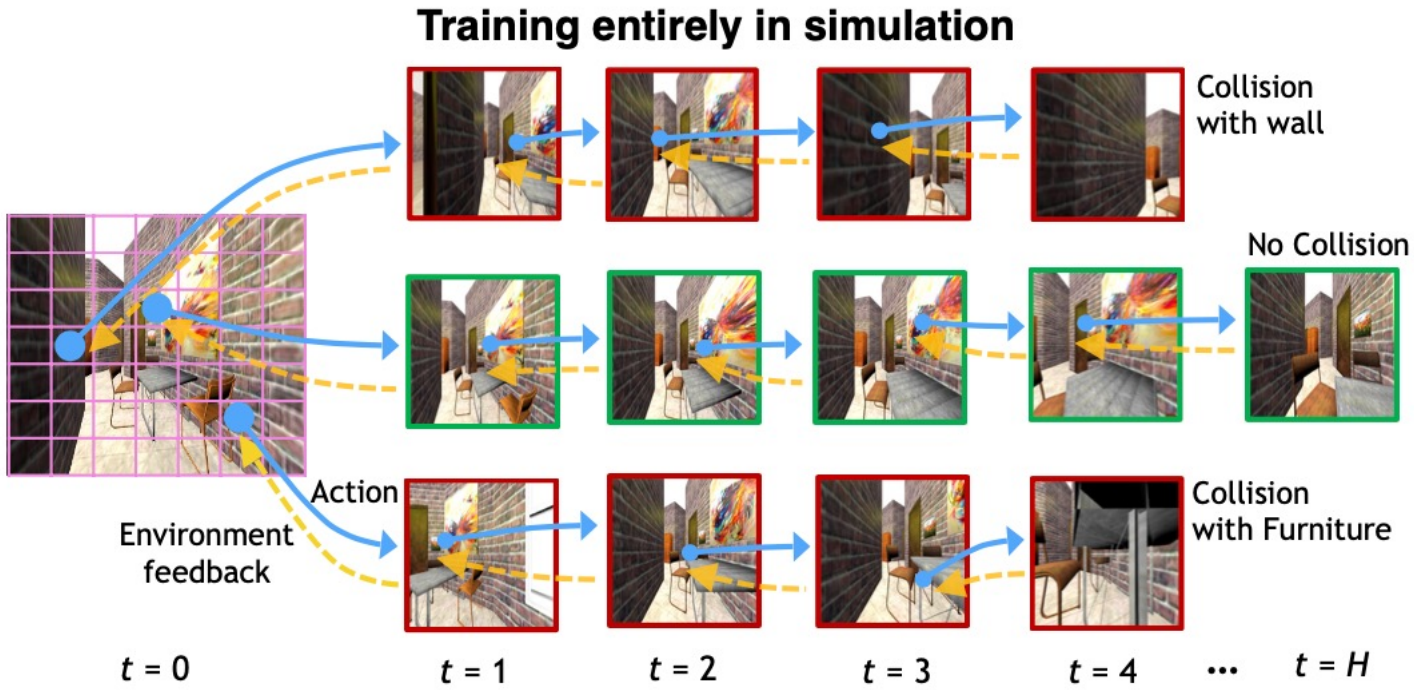


CAD2RL: Real single-image flight without a single real image

Learning Architecture

- Policy Learning: $\pi(a|I)$
  - Q(l|I,a)
    - Pretrain -> free space detection
    - $\pi(I) = \arg\max Q(I, a)$
    - RL: update Q via MC rollout follow π

H-branch : H = 5
Random reset: random location and with random orientation and generate a rollout of size



CAD2RL: Real single-image flight without a single real image

Domain Randomization

1. **furnitures**: We use furnitures with various type and size to populate the hallways.
2. **textures** : The walls are textured with randomly chosen textures(e.g. wood, metal, textile, carpet, stone, glass, etc.), and illuminated with lights that are placed and oriented at random.
3. **viewpoints** : In order to provide a diversity of viewpoints, we render pretraining images by flying a simulated camera with randomized height and random camera orientation.

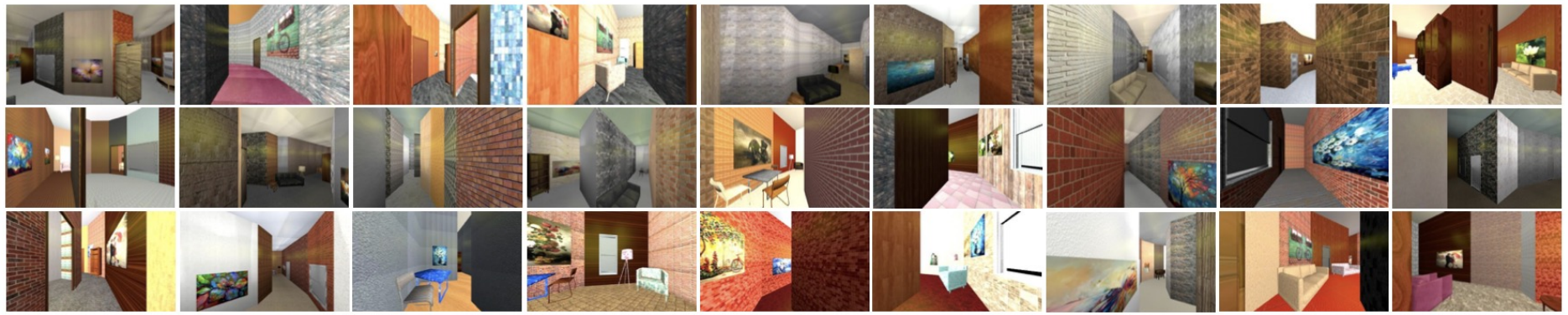


Fig. 2. Examples of rendered images using our simulator. We randomize textures, lighting and furniture placement to create a visually diverse set of scenes.

CAD2RL: Real single-image flight without a single real image

Domain Randomization

1. **furnitures**: We use furnitures with various type and size to populate the hallways.
2. **textures** : The walls are textured with randomly chosen textures(e.g. wood, metal, textile, carpet, stone, glass, etc.), and illuminated with lights that are placed and oriented at random.
3. **viewpoints** : In order to provide a diversity of viewpoints, we render pretraining images by flying a simulated camera with randomized height and random camera orientation.
4. **tasks**: represent a variety of structures that can be seen in real hallways, such as long straight or circular segments with multiple junction connectivity, as well as side rooms with open or closed doors

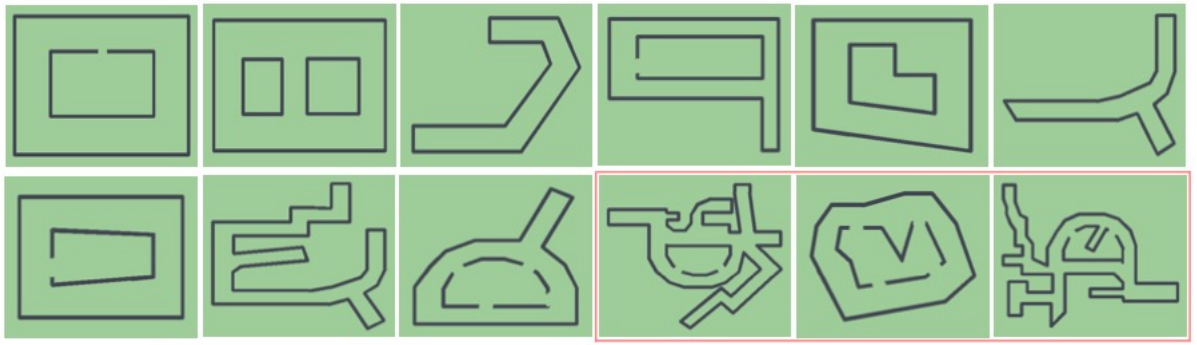


Fig. 4. Floor plans of the synthetic hallways. The last three hallways are used for evaluation while the first 9 are used during training.

CAD2RL: Real single-image flight without a single real image

Experiment
- Realistic Environment Evaluation

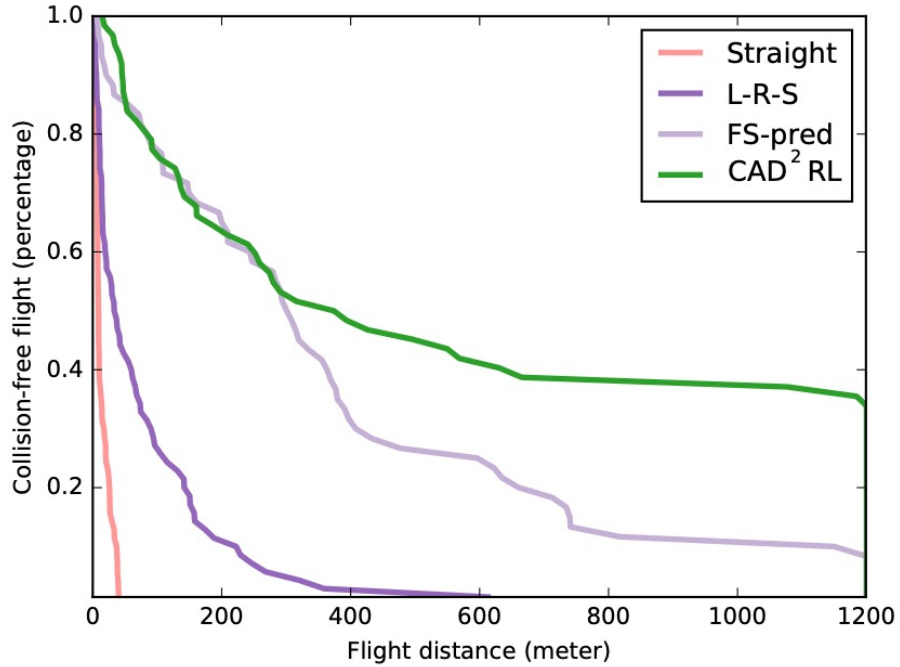


Fig. 5. Quantitative results on a realistically textured hallway. Our approach, CAD^{2}RL, outperforms the prior method (L-R-S) and other baselines.

CAD2RL is able to maintain a collision-free flight of 1.2 kilometers in about 40% of the cases, and substantially out- performs the model that is simply trained with supervised learning to predict 1 meter of free space in front of the vehicle (FS-pred) and SOTA algorithm LRS (a supervised learning algorithm).

CAD2RL: Real single-image flight without a single real image

Experiment
- Real World Flight Experiments

We ran experiments in two different buildings, Cory Hall and SDH (Sutardja Dai Hall), both located on the UC Berkeley campus.

CAD2RL experienced fewer collisions and has longer expected safe flight. This suggests that the CAD2RL policy makes fewer mistakes and is more robust to perturbations and drift.

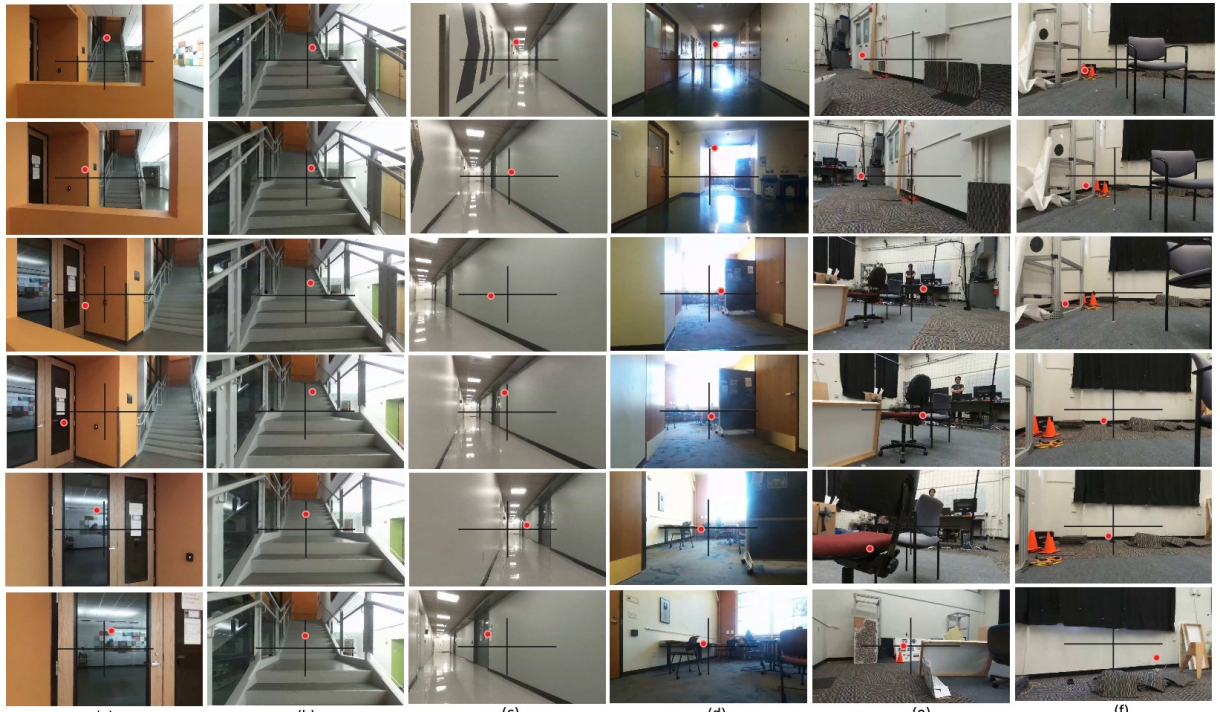


Fig. 7. Snapshots of autonomous flight in various real indoor scenarios. Frames ordered from top to bottom. Red dots show the commanded flight direction by CAD^{2}RL. (a) Flying near furniture, around corners, through a window; (b) Flying up a staircase; (c) Navigating in narrow corridors; (d) Navigating through junctions, fly through rooms; (e) Flying through a maze of random obstacles in a confined space; (f) Avoiding dynamic obstacles.

TABLE I
REAL WORLD FLIGHT RESULTS.

Environment	Traveled Distance (meters)	Travel Time (minutes)	Collision (per meter)	Collision (per minute)	Safe Flight (meters)	Safe Flight (minutes)	Total Collisions
Cory FS-pred	162.458	12.01	0.080	1.081	12.496	0.924	13
Cory CAD^{2}RL	163.779	11.950	0.0366	0.502	27.296	1.991	6
SDH FS-pred	53.492	4.016	0.130	1.742	7.641	0.573	7
SDH CAD^{2}RL	54.813	4.183	0.072	0.956	13.703	1.045	4

Sim-to-Real Transfer of Robotic Control with Dynamics Randomization

Task: Robotic Control

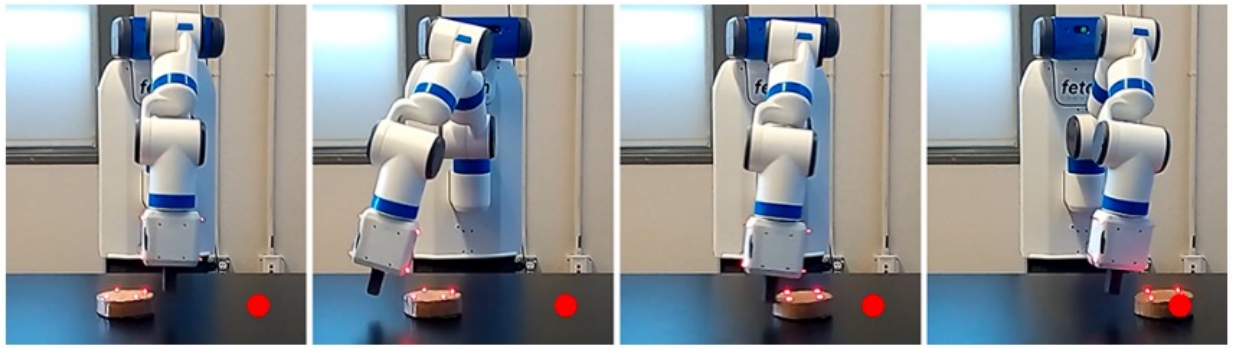


Fig. 1. A recurrent neural network policy trained for a pushing task in simulation is deployed directly on a Fetch Robotics arm. The red marker indicates the target location for the puck.

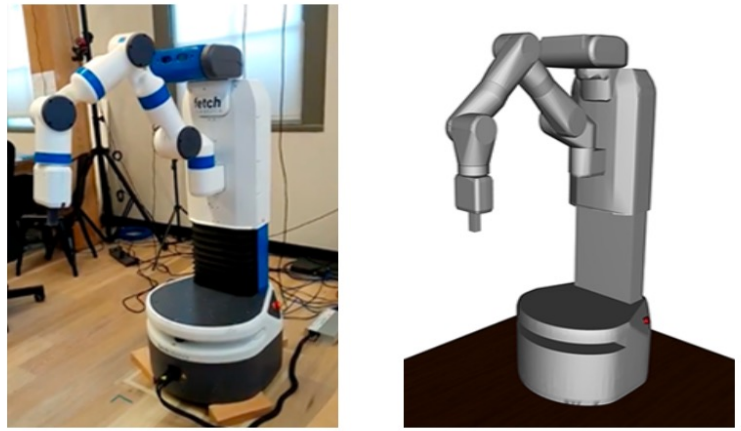


Fig. 2. Our experiments are conducted on a 7-DOF Fetch Robotics arm. **Left:** Real robot. **Right:** Simulated MuJoCo model.

1. Action: 7-DOF Fetch Robotics arm
2. Task: The goal for each episode specifies a random target position on the table that the puck should be moved to. The reward is binary with r = 0 if the puck is within a given distance of the target, and r = −1 otherwise.
3. State: The state is represented using the joint positions and velocities of the arm, the position of the gripper, as well as the puck's position, orientation, linear and angular velocities.

Sim-to-Real Transfer of Robotic Control with Dynamics Randomization

Realty gap in dynamics-parameter space

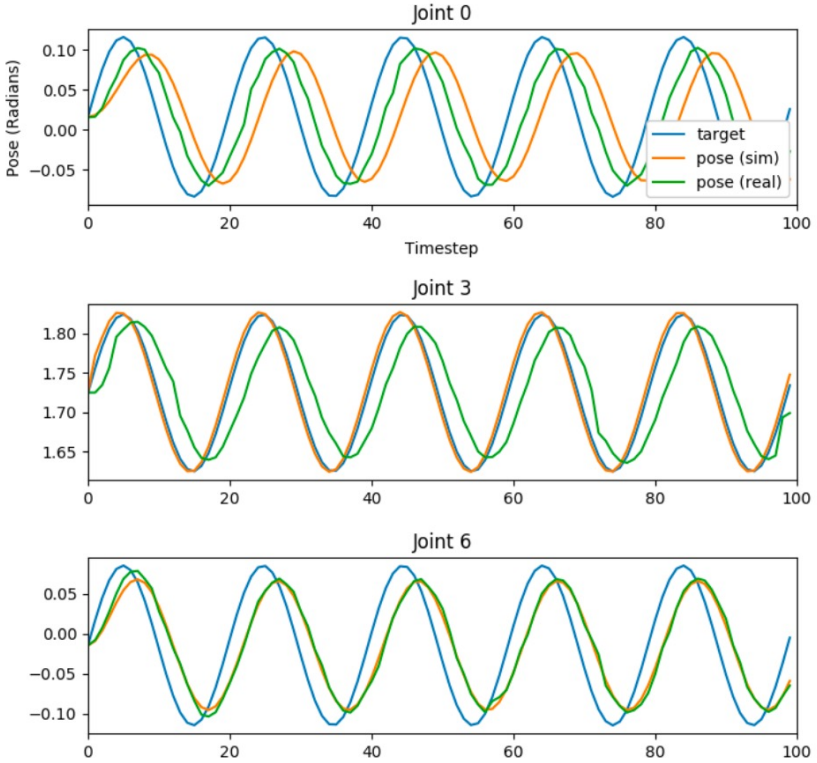


Fig. 5. Joint trajectories recorded from the simulated and real robot when executing the same target trajectories. The joints correspond to the shoulder, elbow, and wrist of the Fetch arm.

Given the same target trajectory, the pose trajectories of the simulated and real robot differ significantly, with varying degrees of mismatch across joints.

Sim-to-Real Transfer of Robotic Control with Dynamics Randomization

Dynamics Randomization

During training, rollouts are organized into episodes of a fixed length. At the start of each episode, a random set of dynamics parameters μ are sampled according to ρ_μ and held fixed for the duration of the episode. The parameters which we randomize include:

- Mass of each link in the robot's body
- Damping of each joint
- Mass, friction, and damping of the puck
- Height of the table
- Gains for the position controller
- Timestep between actions
- Observation noise

which results in a total of 95 randomized parame The

Parameter	**Range**
Link Mass	$[0.25, 4]\times$ default mass of each link
Joint Damping	$[0.2, 20]\times$ default damping of each joint
Puck Mass	$[0.1, 0.4]kg$
Puck Friction	$[0.1, 5]$
Puck Damping	$[0.01, 0.2]Ns/m$
Table Height	$[0.73, 0.77]m$
Controller Gains	$[0.5, 2]\times$ default gains
Action Timestep λ	$[125, 1000]s^{-1}$

TABLE I

DYNAMICS PARAMETERS AND THEIR RESPECTIVE RANGES.

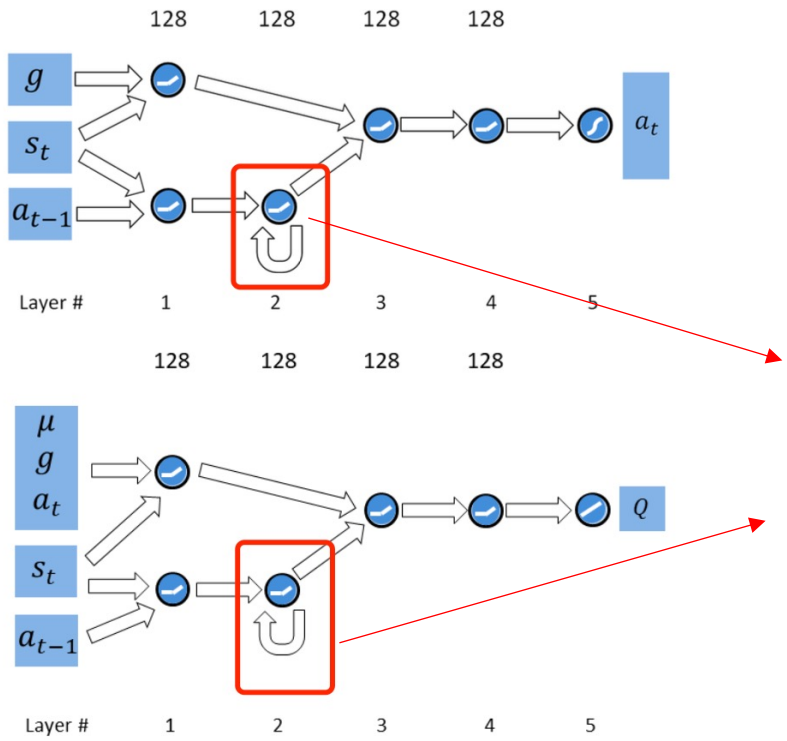
Sim-to-Real Transfer of Robotic Control with Dynamics Randomization

Adaptive Policy Training

The objective is then modified to maximize the expected return across a distribution of dynamics models ρ_μ,

$$\mathop{\mathbb{E}}_{\mu\sim\rho_\mu}\left[\mathbb{E}_{\tau\sim p(\tau|\pi,\mu)}\left[\sum_{t=0}^{T-1} r(s_t,a_t)\right]\right]$$

By training policies to adapt to variability in the dynamics of the environment, the resulting policy might then better generalize to the dynamics of real world.



RNN: Learn to represent the environment

Fig. 4. Schematic illustrations of the policy network (**top**), and value network (**bottom**). Features that are relevant for inferring the dynamics of the environment are processed by the recurrent branch, while the other inputs are processed by the feedforward branch.

Sim-to-Real Transfer of Robotic Control with Dynamics Randomization

Experiments

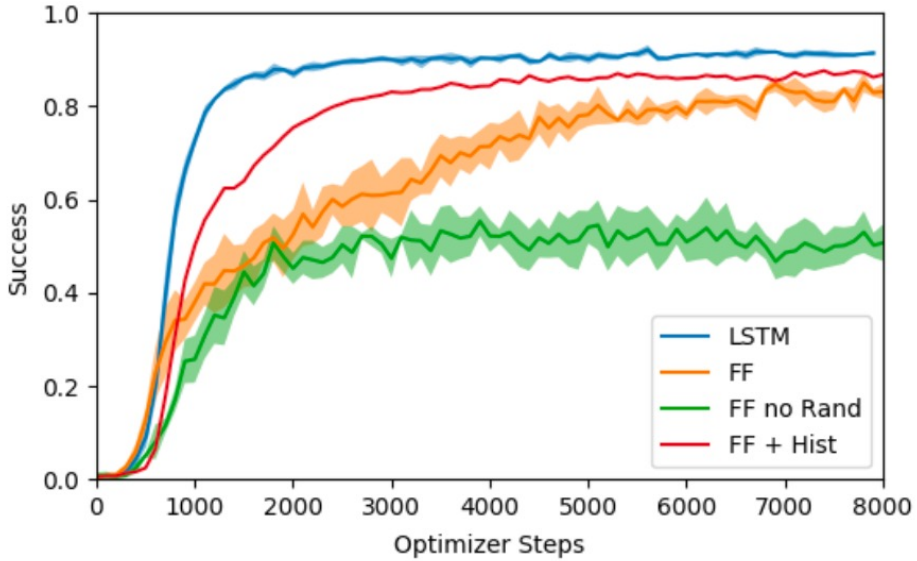


Fig. 6. Learning curves of different network architectures. Four policies are trained for each architecture with different random seeds. Performance is evaluated over 100 episodes in simulation with random dynamics.

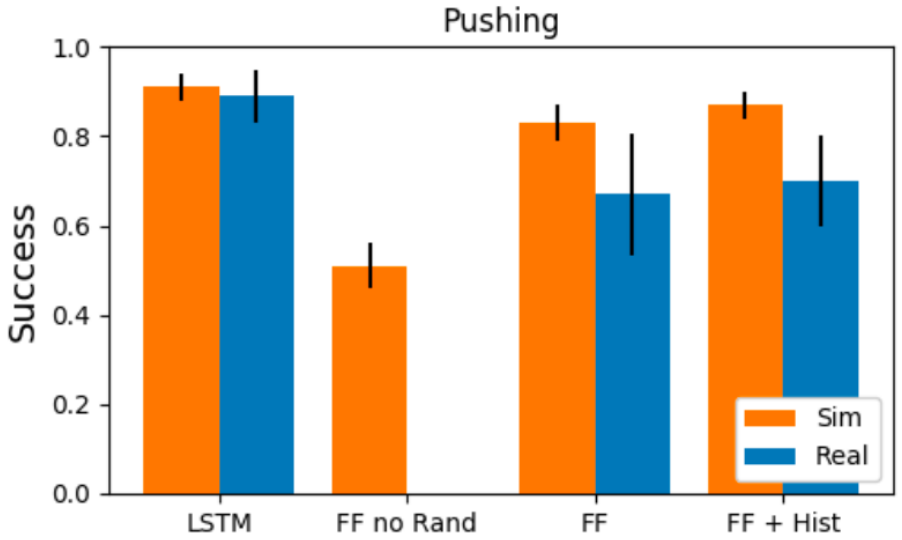


Fig. 7. Performance of different models when deployed on the simulated and real robot for the pushing task. Policies are trained using only data from simulation.

Sim

Real

A Solid Application: Solving Rubik's Cube with a Robot Hand

More details: https://zhuanlan.zhihu.com/p/89312919

Table of Contents

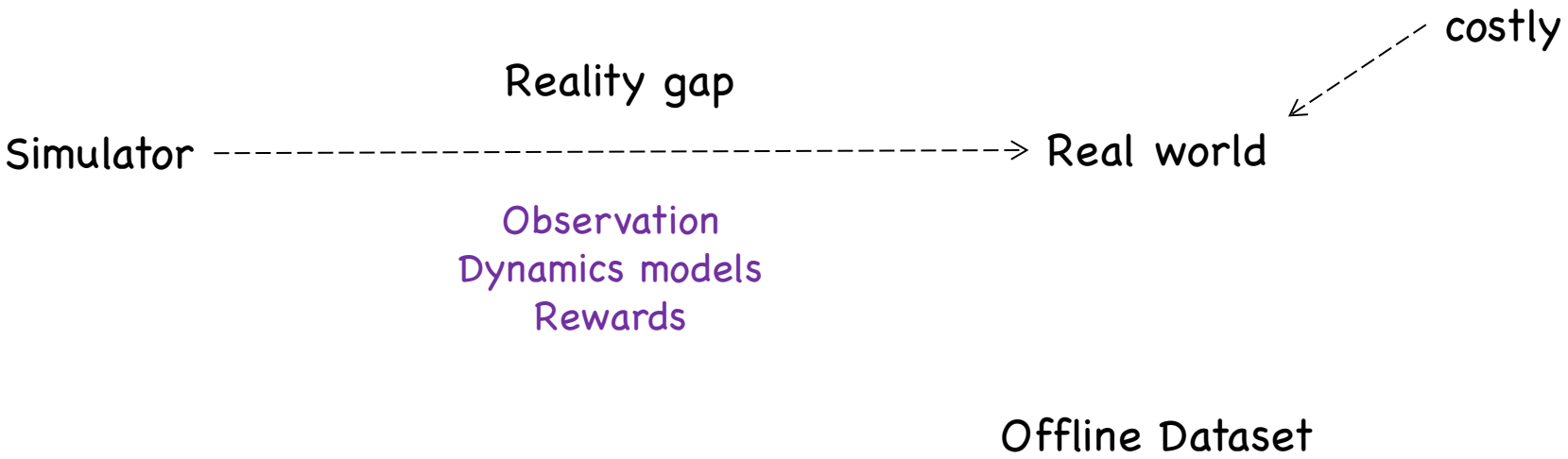
1. Sim2Real Transfer for Reinforcement Learning
2. Domain Randomization for Sim2Real RL
3. **Domain Adaptation for Sim2Real RL**
4. Our work: Cross-Modal Domain Adaptation with Sequential structure (CODAS)

Domain Adaptation for Sim2Real RL

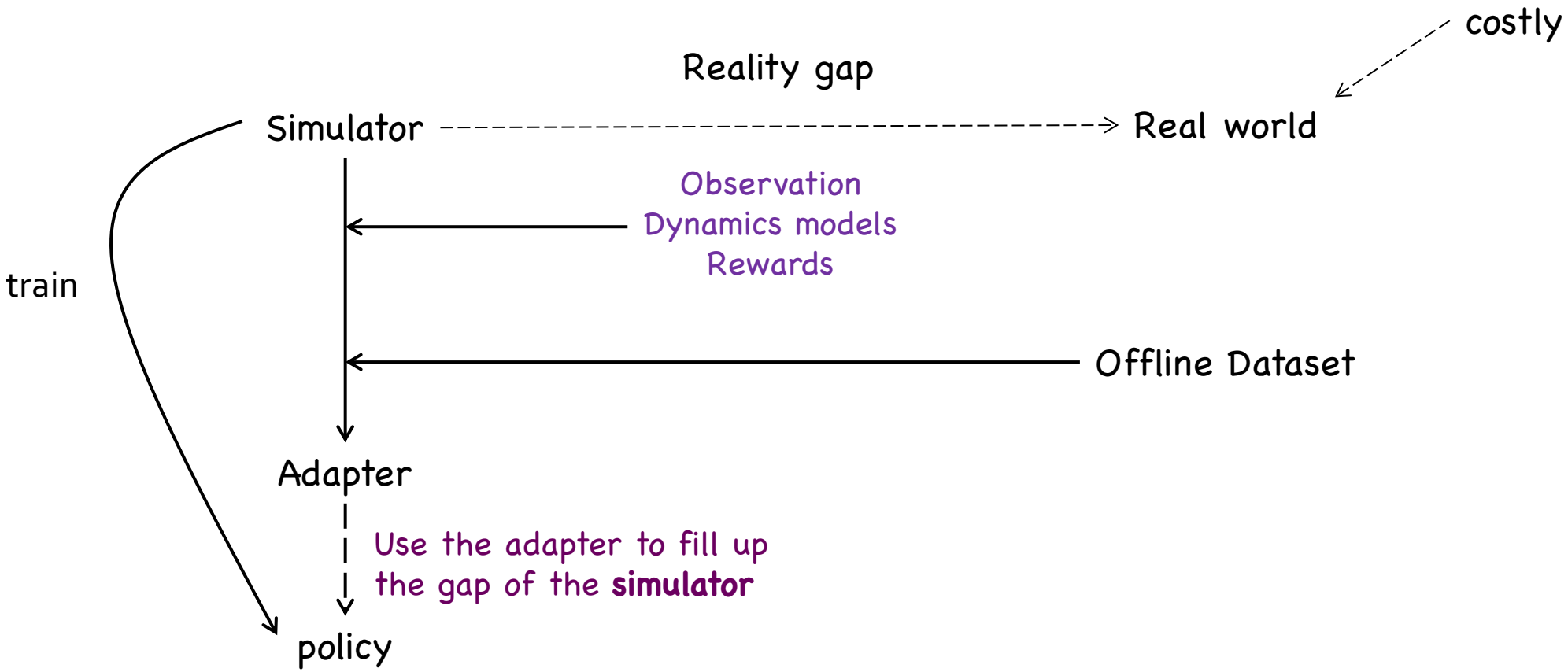
Derived Setting:

1. Target: We would like to train an agent to maximize the rewards in the real world.
2. Query **online samples** in the real world are **costly**.
    1. [-] Different attitudes to "costly".
        1. Allow a few online samples (e.g., several steps or one episode) before deployment.
        2. Allow zero extra querying before deployment.
    2. Other sources of data?
        1. [√] We have an offline dataset (collected by a human/rule-based policy).
        2. [x] We have no extra real-world dataset.
3. We have **a simulator** that is used to simulate the real world.
    1. [-] What is the ability of the simulator?
        1. Configurable (e.g., generate dynamics models with different friction coefficients)?
        2. Renderable (e.g., state -> image)?
4. But the simulator has **reality gaps** to the real world.
    1. [-] The source of the gaps.
        1. Observations
        2. Dynamics models
        3. Reward function (e.g., different tasks)

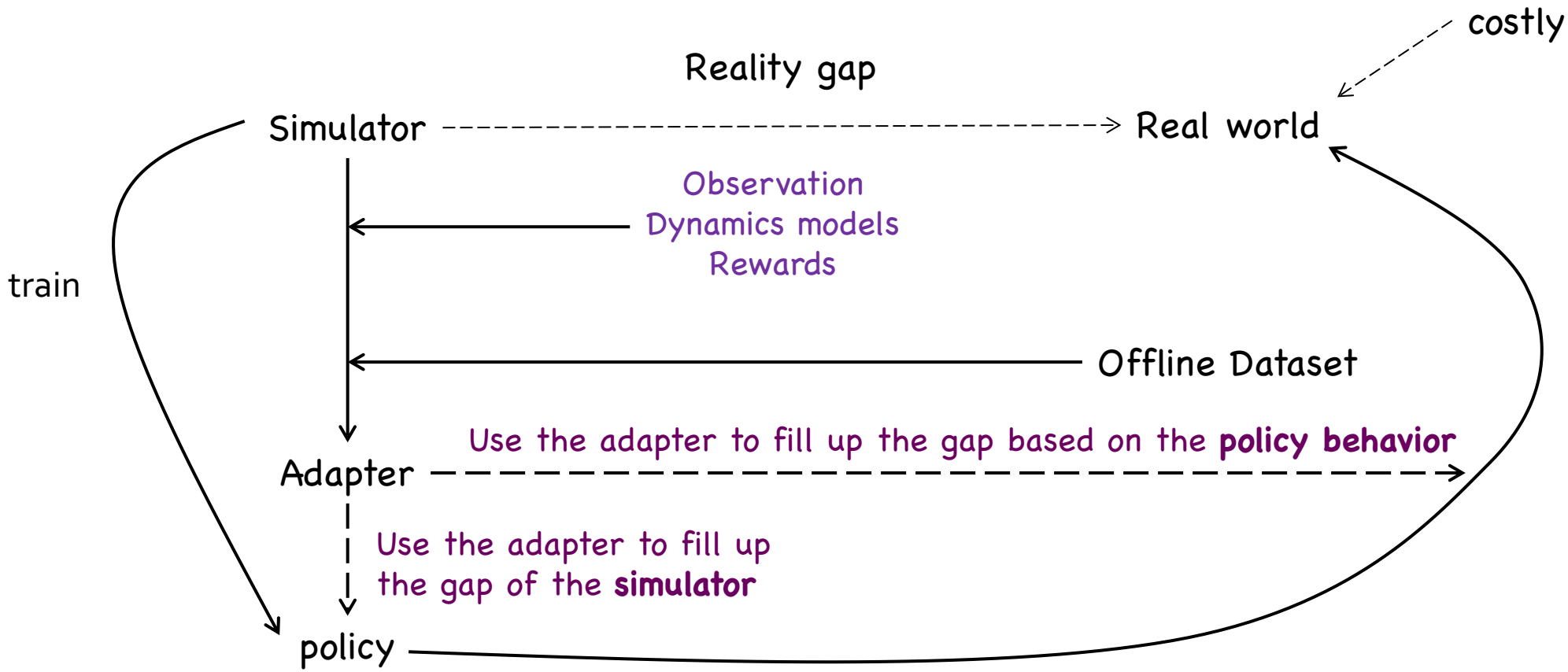
Domain Adaptation for Sim2Real RL



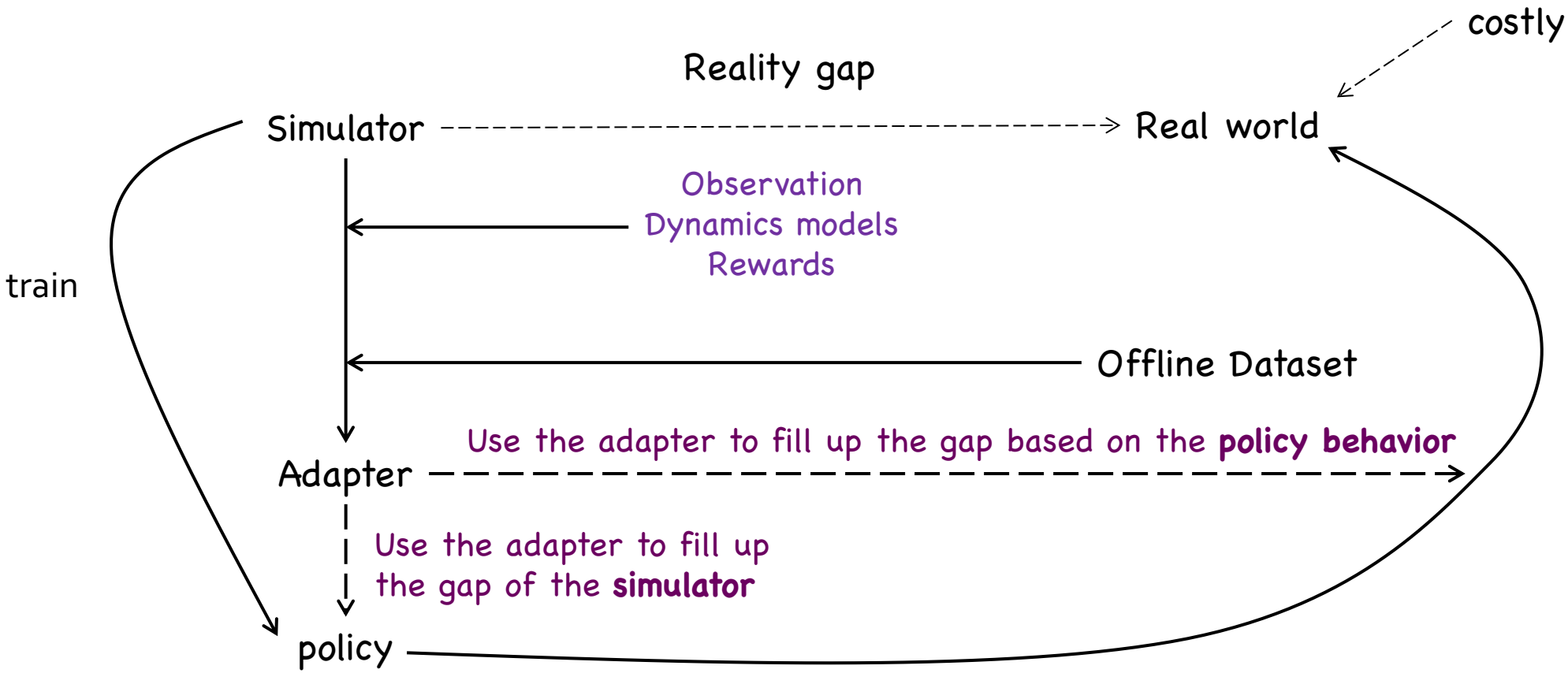
Domain Adaptation for Sim2Real RL



Domain Adaptation for Sim2Real RL



Domain Adaptation for Sim2Real RL



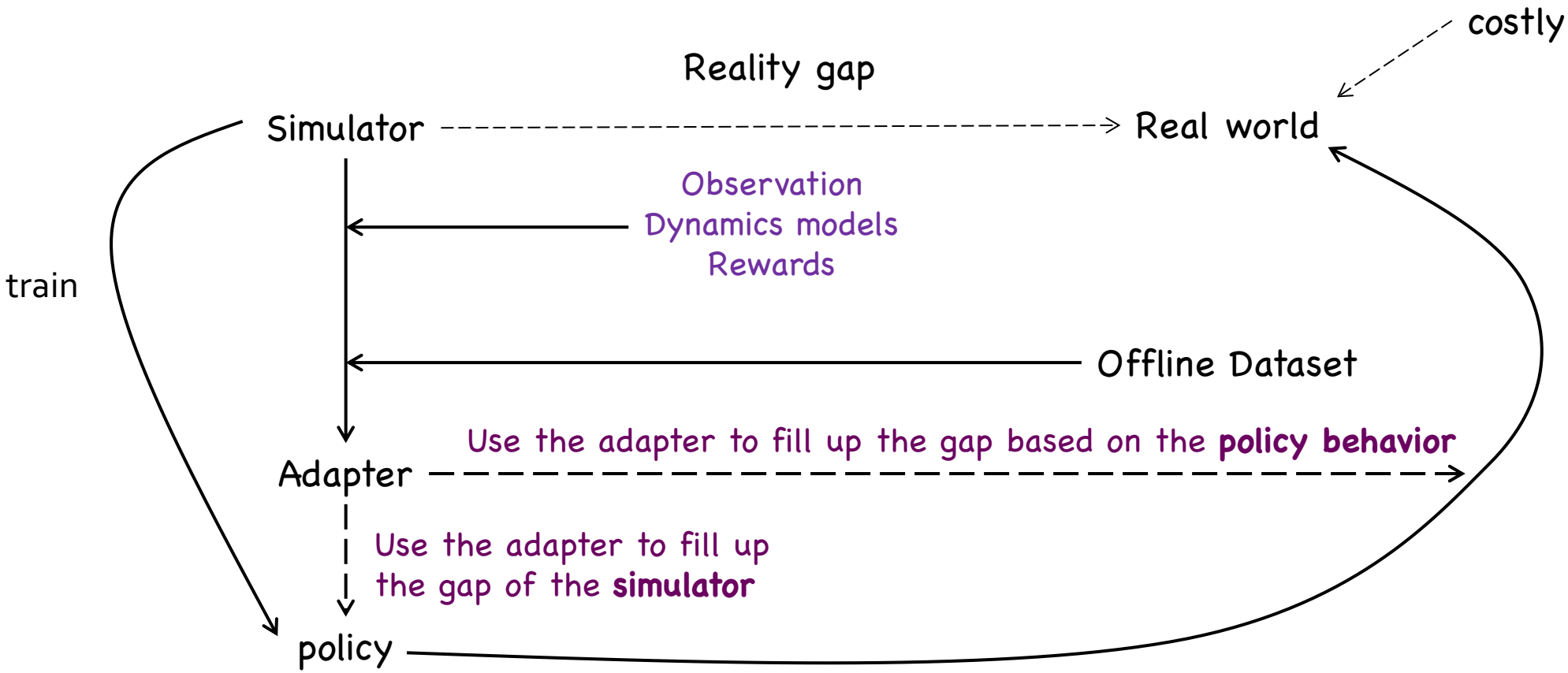
The adapter can be designed in many different ways:

$\pi(a|Adapt(o_s|o_r))$, $\pi(a|s) + Adapt(a|s)$

$T(s'|s,a) + Adapt(s'|s,a)$

...

Domain Adaptation for Sim2Real RL



The adapter can be designed in many different ways:

$\pi(a|Adapt(o_s|o_r))$, $\pi(a|s) + Adapt(a|s)$

$T(s'|s,a) + Adapt(s'|s,a)$

...

Virtual to Real Reinforcement Learning for Autonomous Driving

Observation gap in autonomous driving

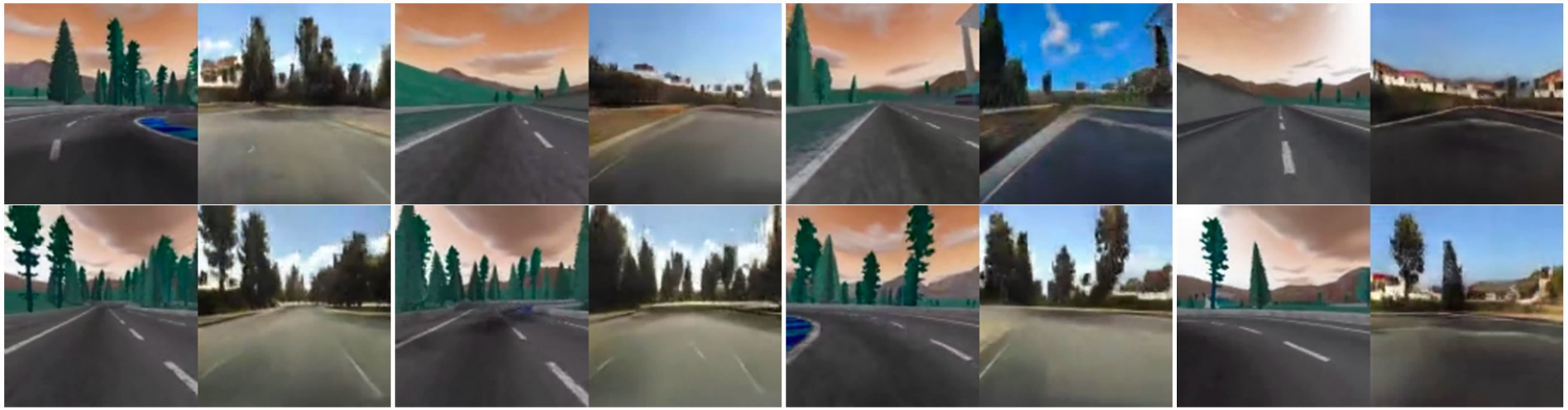


Figure 4: Examples of Virtual to Real Image Translation. Odd columns are virtual images captured from TORCS. Even columns are synthetic real world images corresponding to virtual images on the left.

Virtual to Real Reinforcement Learning for Autonomous Driving

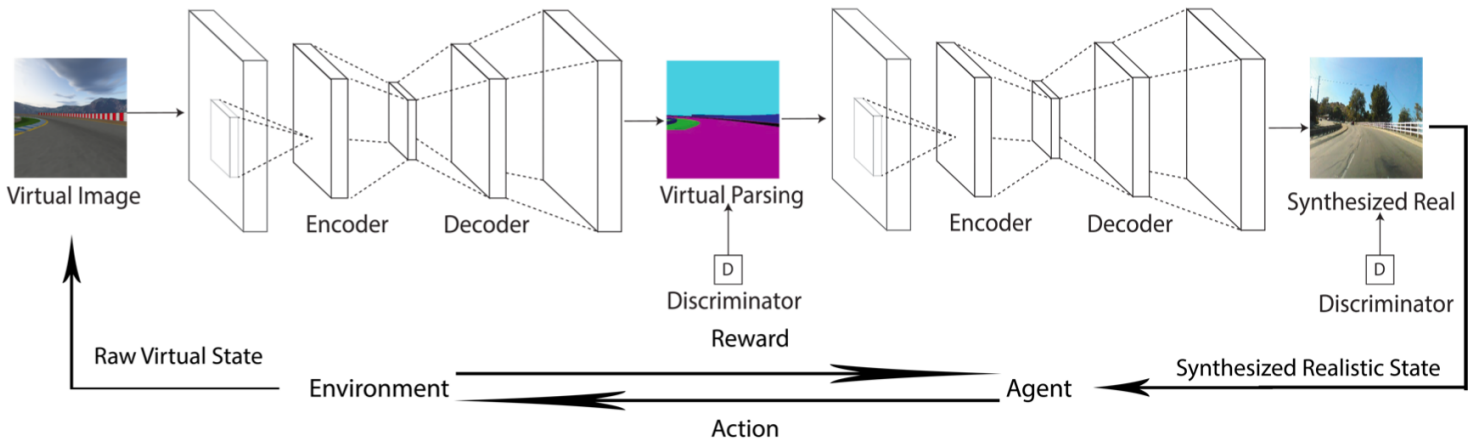


Figure 1: Framework for virtual to real reinforcement learning for autonomous driving. Virtual images rendered by a simulator (environment) are first segmented to scene parsing representation and then translated to synthetic realistic images by the proposed image translation network (VISRI). Agent observes synthetic realistic images and takes actions. Environment will give reward to the agent. Since the agent is trained using realistic images that are visually similar to real world scenes, it can nicely adapt to real world driving.

$$\begin{aligned}\mathcal{L}_{cGAN}(G,D) =& \mathbb{E}_{x,s\sim p_{data}(x,s)}[\log D(x,s)] \\ &+ \mathbb{E}_{x\sim p_{data}(x),z\sim p_z(z)}[\log(1-D(x,G(x,z)))],\end{aligned} \tag{1}$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,s\sim p_{data}(x,s),z\sim p_z(z)}[\|s-G(x,z)\|_1]. \tag{2}$$

Therefore, the overall objective for the image-to-image translation network is,

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G), \tag{3}$$

Virtual to Real Reinforcement Learning for Autonomous Driving

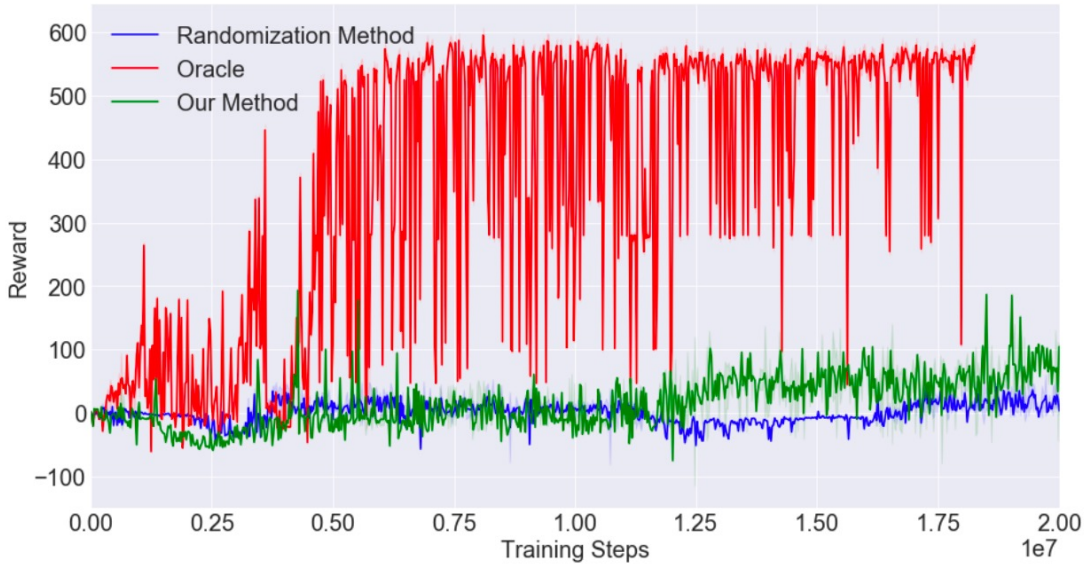


Figure 5: Transfer learning between different environments. Oracle was trained in Cgtrack2 and tested in Cgtrack2, so its performance is the best. Our model works better than the domain randomization RL method. Domain randomization method requires training in multiple virtual environments, which imposes significant manual engineering work.

Table 1: Action prediction accuracy for the three methods.

Accuracy	Ours	B-RL	SV
Dataset in [5]	43.40%	28.33%	53.60%

Sim-to-Real Transfer with Neural-Augmented Robot Simulation

Dynamics gap in robotics

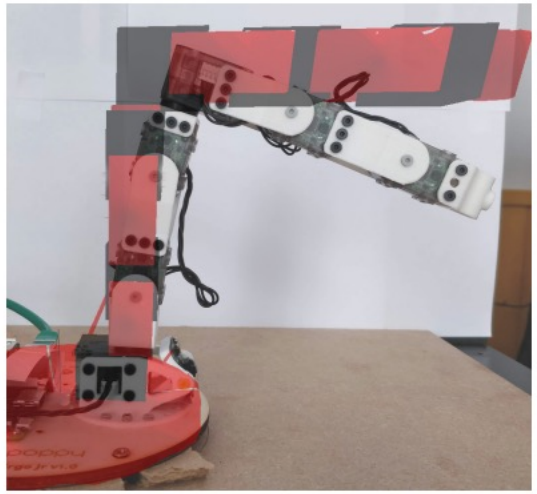
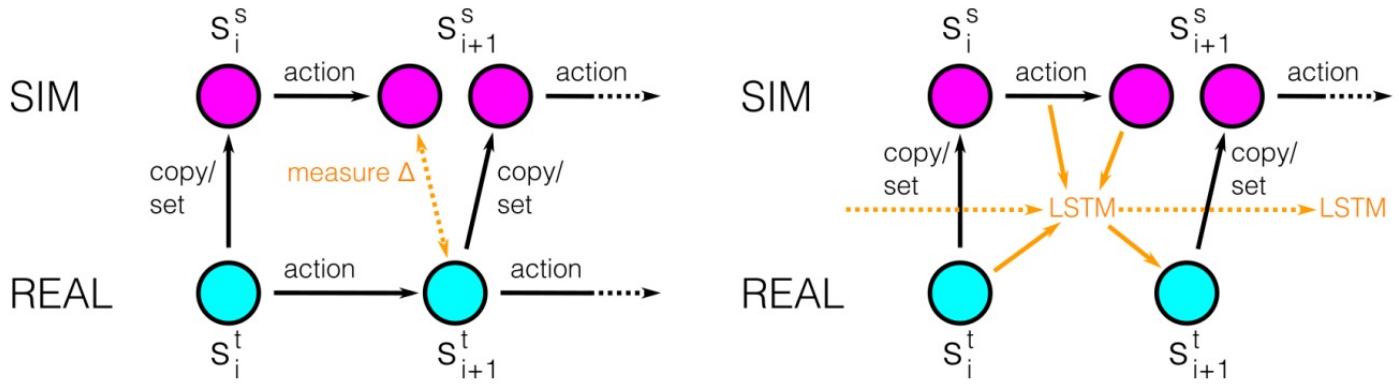


Figure 1: Impact of backlash: when setting both the simulated robot (red) and the real robot (white) to the resting position, the small backlash of each joint adds up to a noticeable difference in the end effector position.

Sim-to-Real Transfer with Neural-Augmented Robot Simulation

Neural-Augmented Robot Simulation



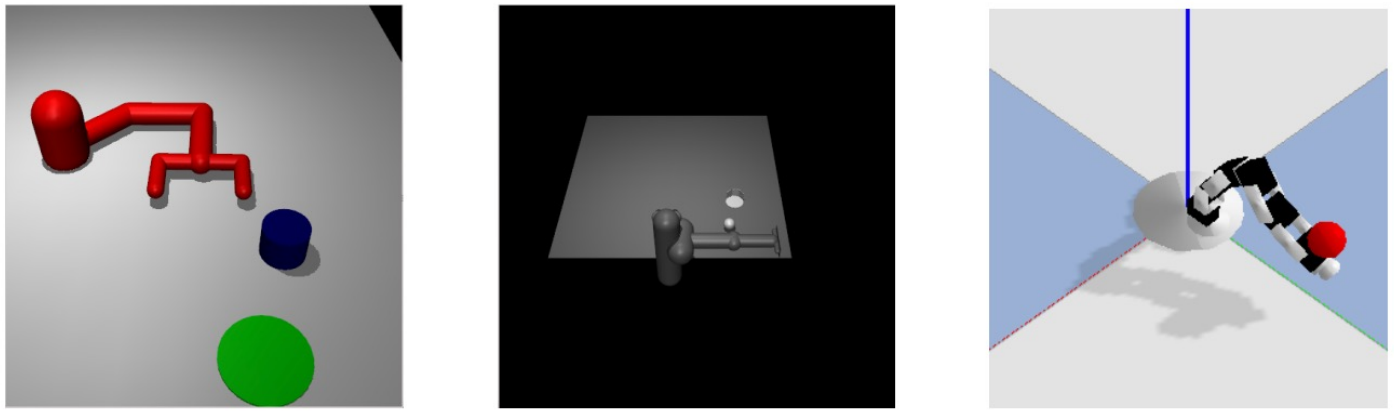
(a) Training phase (b) Policy Learning Phase

Figure 2: Left: Overview of the method for training the forward dynamics model. By gathering state differences when running the same action in simulation and on the real robot. Right: When the forward model is learned, it can be applied to simulator states to get the corresponding real state. This correction model (Δ) is time-dependent (implemented as LSTM). The policy learning algorithm only has access to the "real" states.

After collecting the data the model ϕ, an LSTM [7], is trained to predict s^t_{i+1}. The difference between two states is usually small, so the network outputs a correction term $\phi(s^t_i, a_i, h, s^s_{i+1}) = s^t_{i+1} - s^s_{i+1}$ where h is the hidden state of the network. We also compare our approach with a forward model trained without using information from the source domain, $\psi(s_i, a_i, h) = s^t_{i+1} - s_i$. We normalize the inputs and outputs, and the model is trained with maximum likelihood using Adam [30].

Sim-to-Real Transfer with Neural-Augmented Robot Simulation

Experiment:
- MuJoCo

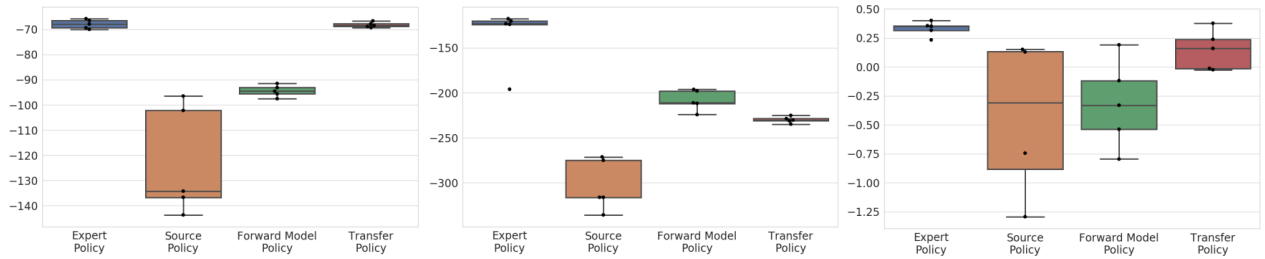


(a) Pusher (b) Striker (c) ErgoReacher

Figure 3: Benchmark simulation environments used

Sim-to-Real Transfer with Neural-Augmented Robot Simulation

Experiment:
- MuJoCo



(a) Pusher (b) Striker (c) ErgoReacher

Figure 5: Comparison of the different methods described when deployed on the target environment, for the Pusher (5a) and the Striker (5b).

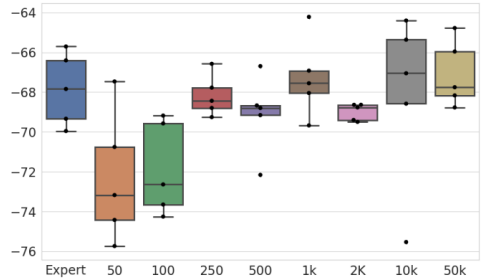
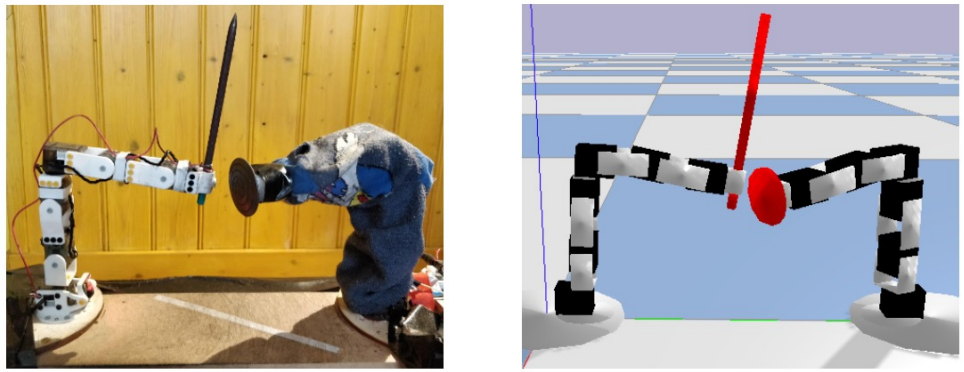


Figure 6: We compare the results of our method when the number of trajectories used to train the model ϕ varies on the Pusher

- *Expert policy*: policy trained directly in the target environment
- *Source policy*: transferring a policy trained in source environment without any adaption
- *Forward model policy*: a forward dynamic model ψ is trained using an LSTM and data collected from the target domain then a policy trained using only this model (without insight from the source domain)
- *Transfer policy*: the policy trained using NAS

Sim-to-Real Transfer with Neural-Augmented Robot Simulation

Experiment:
- Physical Robot

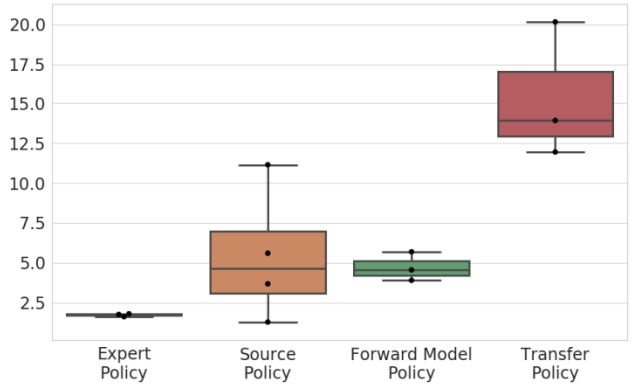


(a) Physical Robots (b) Simulation

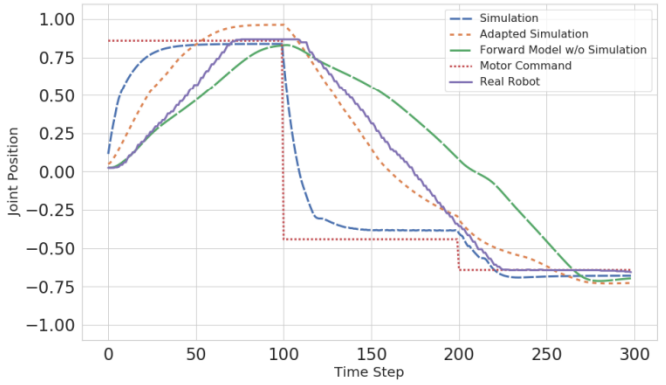
Figure 7: The *ErgoShield* environment in reality and simulation. The attacker (sword, left side) has to hit the randomly-moving defender (shield, right side) as often as possible in 10s. In the left image the defender robot is covered in a sock to mitigate the attacker getting stuck too often. The joint angles were compensated for backlash in the left image to highlight similarities.

Sim-to-Real Transfer with Neural-Augmented Robot Simulation

Experiment:
- Physical Robot



(a) Method Comparison (Real Robot)



(b) Example Single Joint Position and Estimate over Time

Figure 8: Results of different simulation to real transfer approaches. Left: comparison of average rewards of 20 rollouts of 3 policies per approach. Right: comparison of single joint behavior when receiving a target position (violet dotted) in simulation (green dashed), on the real robot (blue solid), and estimates from the forward model (red, dashed) and our method (yellow, dotted)

1. hit detection is not perfect (as in simulation) and since not every hit gets detected the reward is sparser
2. since the attacker robot frequently gets their sword stuck in the opponent, in themselves, or in the environment, exploration is not as easy as in simulation

Learning to Drive from Simulation without Real World Labels

Overall framework

Learn a latent space to represent
The observations in the two domains

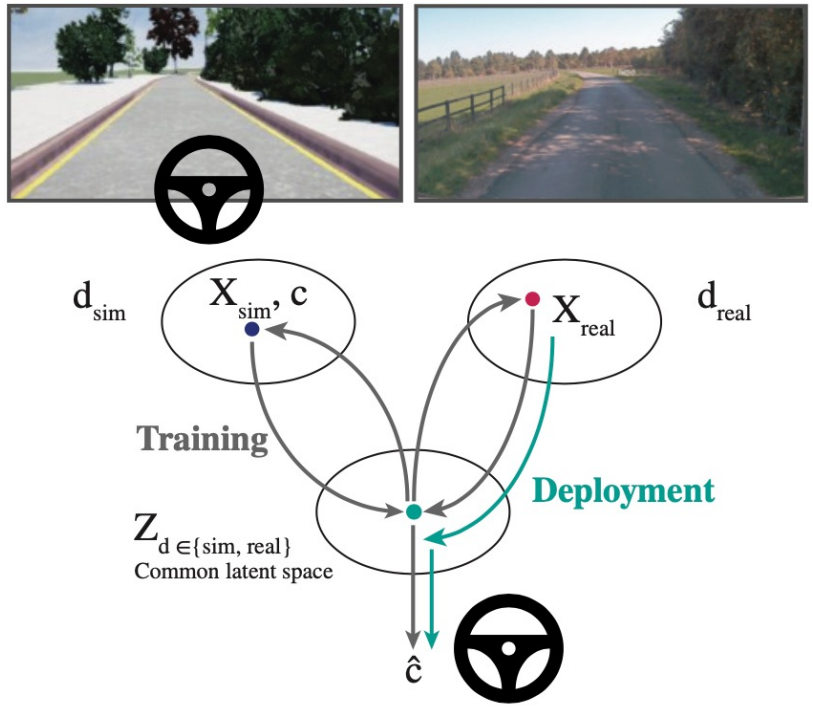


Fig. 1: We constructed a model for end-to-end driving (vision to action) for lane following by learning to translate between simulated and real-world imagery ($X_{sim,real}$ from domains $d_{sim,real}$), jointly learning a control policy from this common latent space Z using labels c from an expert driver in simulation. This method does not require real-world control labels (which are more difficult to acquire), yet learns a policy predicting a control signal $\hat{c}$ which can be applied successfully to a real-world driving task.

Learning to Drive from Simulation without Real World Labels

Two directions generator

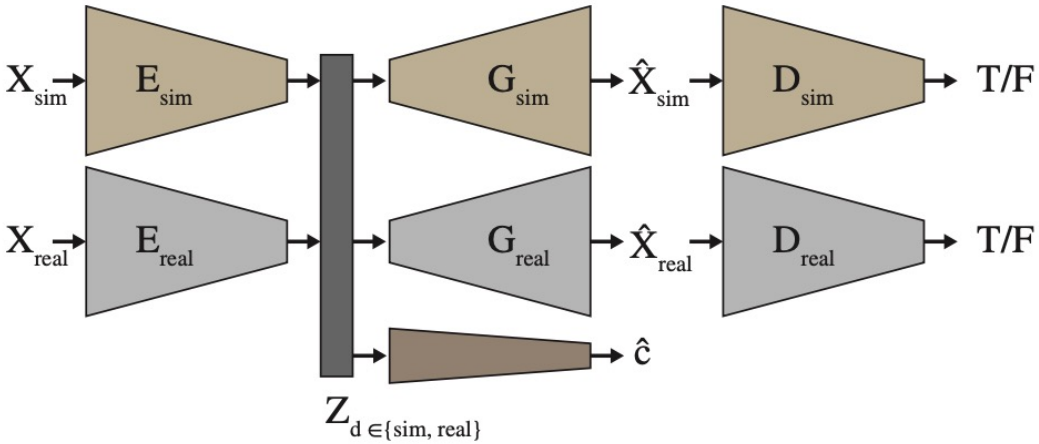


Fig. 2: Model architecture for domain-transfer from a simulated domain to real-world imagery, jointly learning control and domain translation. The encoders $E_{sim,real}$ map input images from their respective domains to a latent space Z which is used for predicting vehicle controls $\hat{c}$. This common latent space is learned through direct and cyclic losses as part of learning image-to-image translation, indicated conceptually in Figure 3 and in Section III-B.

Learning to Drive from Simulation without Real World Labels

Two directions generator: Image Reconstruction Loss

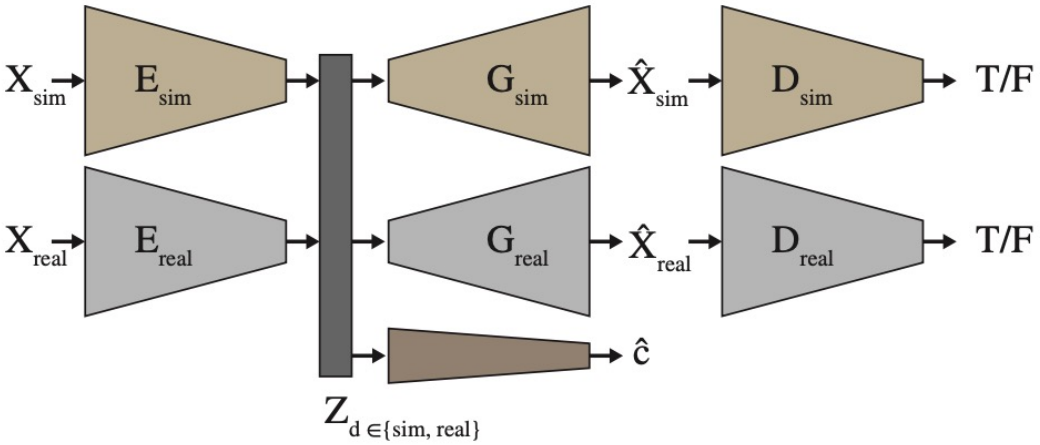
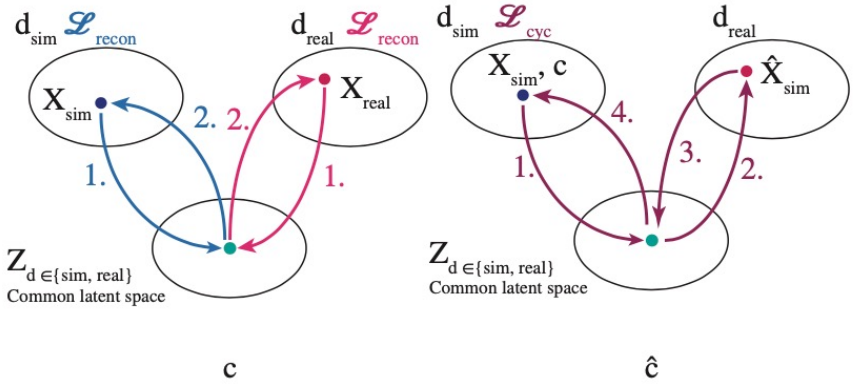
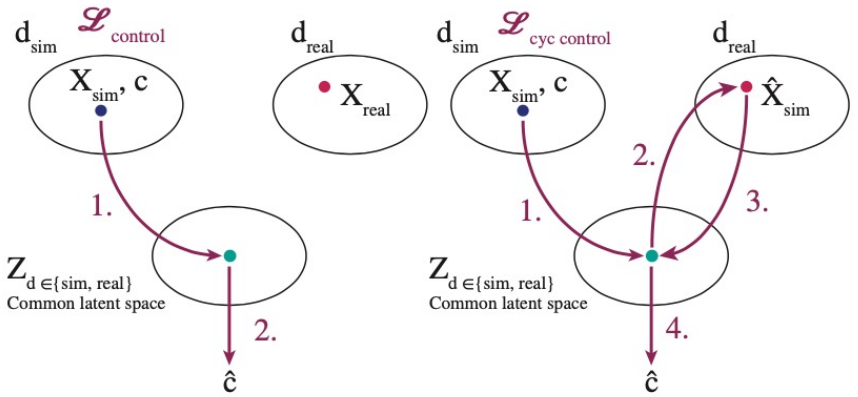


Fig. 2: Model architecture for domain-transfer from a simulated domain to real-world imagery, jointly learning control and domain translation. The encoders $E_{sim,real}$ map input images from their respective domains to a latent space Z which is used for predicting vehicle controls $\hat{c}$. This common latent space is learned through direct and cyclic losses as part of learning image-to-image translation, indicated conceptually in Figure 3 and in Section III-B.



(a) Reconstruction loss $\mathcal{L}_{recon}$ (b) Cyclic reconstruction loss $\mathcal{L}_{cyc}$



(c) Control loss $\mathcal{L}_{control}$ (d) Cyclic control loss $\mathcal{L}_{cyc\ control}$

Learning to Drive from Simulation without Real World Labels

Two directions generator: Image Reconstruction Loss

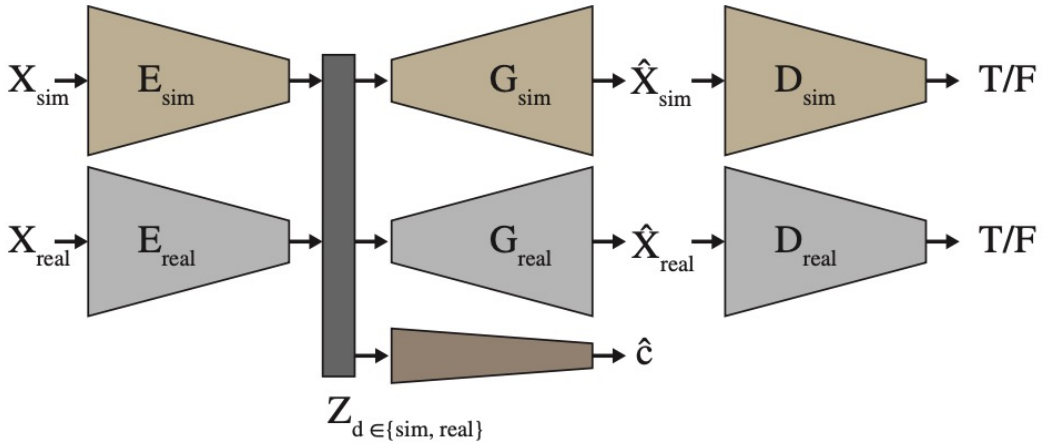
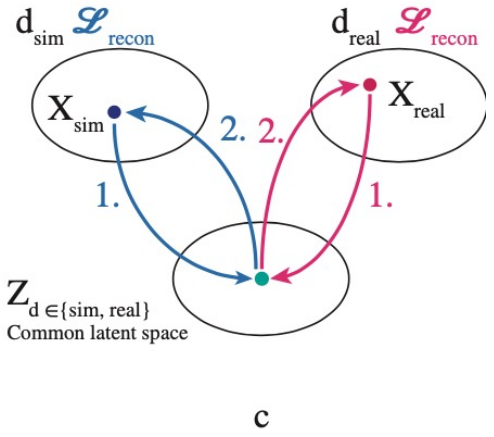


Fig. 2: Model architecture for domain-transfer from a simulated domain to real-world imagery, jointly learning control and domain translation. The encoders $E_{sim,real}$ map input images from their respective domains to a latent space Z which is used for predicting vehicle controls $\hat{c}$. This common latent space is learned through direct and cyclic losses as part of learning image-to-image translation, indicated conceptually in Figure 3 and in Section III-B.



c

1) Image Reconstruction Loss: For a given domain d, E_d and G_d constitute a VAE. To improve image translation we used an L1 Loss $\mathcal{L}_{recon}$ between an image X_d and the reconstructed image after passing it through the corresponding VAE, $X_d^{recon} = G_d(E_d(X_d))$, as shown in Figure 3a.

Learning to Drive from Simulation without Real World Labels

Two directions generator: Cyclic Reconstruction Loss

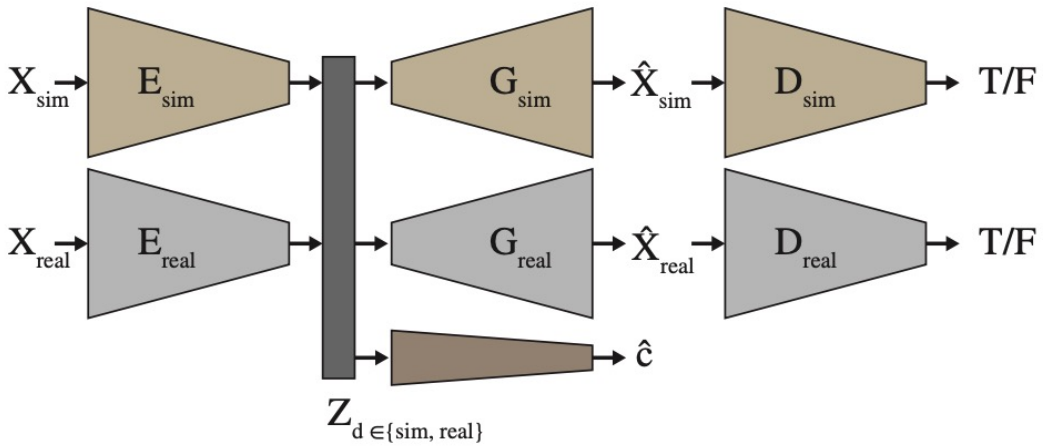
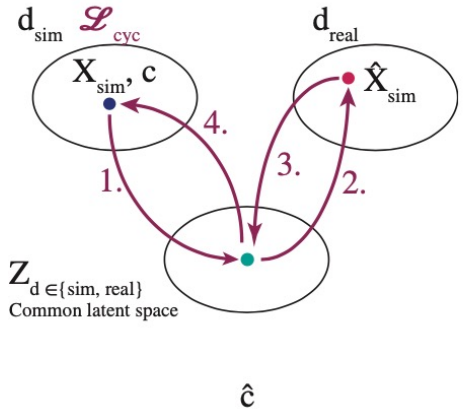


Fig. 2: Model architecture for domain-transfer from a simulated domain to real-world imagery, jointly learning control and domain translation. The encoders $E_{sim,real}$ map input images from their respective domains to a latent space Z which is used for predicting vehicle controls $\hat{c}$. This common latent space is learned through direct and cyclic losses as part of learning image-to-image translation, indicated conceptually in Figure 3 and in Section III-B.



2) Cyclic Reconstruction Loss: Assuming a shared latent space implies the cycle-consistency constraint, which says that if an image is translated to the other domain and then translated back, the original image should be recovered [38], [15]. We applied a cyclic consistency loss $\mathcal{L}_{cyc}$ to the VAEs, given by an L1 loss between an image X_d and the image after translating to the other domain, d', and back, $X_d^{cyc} = G_d(E_{d'}(G_{d'}(E_d(X_d))))$, see Figure 3b.

Learning to Drive from Simulation without Real World Labels

Two directions generator: Control Loss

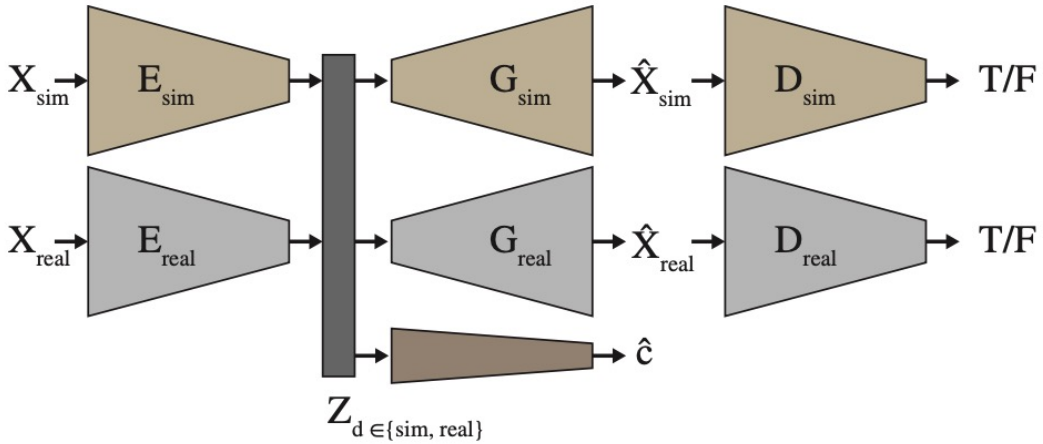
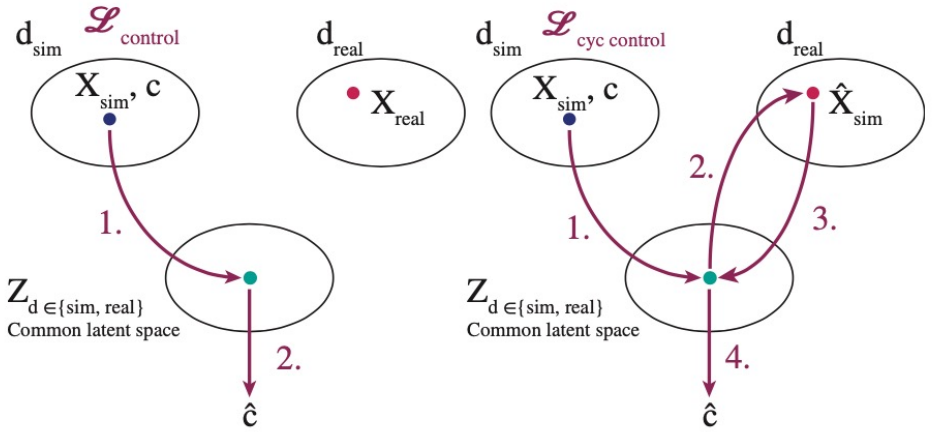


Fig. 2: Model architecture for domain-transfer from a simulated domain to real-world imagery, jointly learning control and domain translation. The encoders $E_{sim,real}$ map input images from their respective domains to a latent space Z which is used for predicting vehicle controls $\hat{c}$. This common latent space is learned through direct and cyclic losses as part of learning image-to-image translation, indicated conceptually in Figure 3 and in Section III-B.



(c) Control loss $\mathcal{L}_{control}$ (d) Cyclic control loss $\mathcal{L}_{cyc\ control}$

3) Control Loss: To guide our model to learn features that are useful for driving, we also used a control loss $\mathcal{L}_{control}$, which is an L1 loss between the controller's predicted steering $\hat{c} = C(E_d(X_d))$ and the ground truth given by the autopilot, c, shown in Figure 3c.

Learning to Drive from Simulation without Real World Labels

Two directions generator: Adversarial Loss

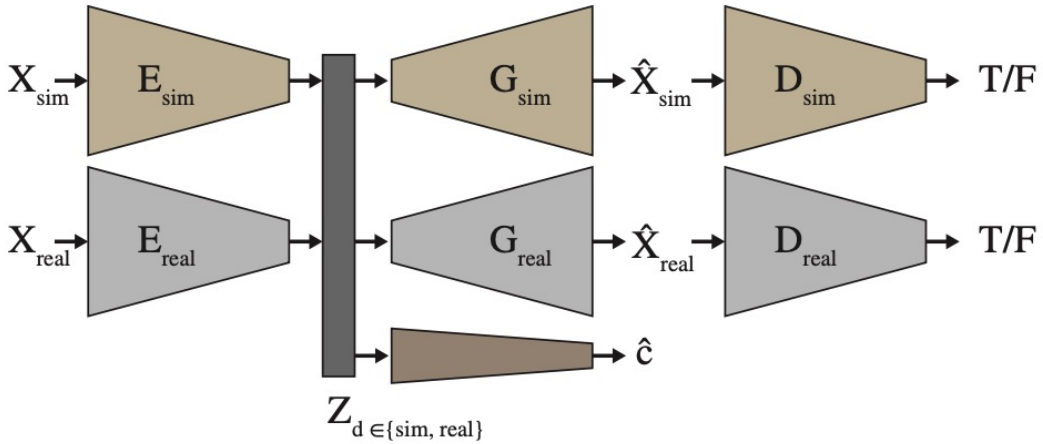


Fig. 2: Model architecture for domain-transfer from a simulated domain to real-world imagery, jointly learning control and domain translation. The encoders $E_{sim,real}$ map input images from their respective domains to a latent space Z which is used for predicting vehicle controls $\hat{c}$. This common latent space is learned through direct and cyclic losses as part of learning image-to-image translation, indicated conceptually in Figure 3 and in Section III-B.

4) Adversarial Loss: Both the image-translator and the discriminators were optimised with the Least-Squares Generative Adversarial Network (LSGAN) objective proposed by [17]. The discriminator (3) and generator (4) adversarial losses ensured that translated images resembled those from the chosen domain.

$$\mathcal{L}_{LSGAN}(D) = \mathbb{E}_{X\sim p_{data}}[(D(X)-1)^2] + \mathbb{E}_{Z\sim p_Z(Z)}[(D(G(Z))-0)^2] \quad (3)$$

$$\mathcal{L}_{LSGAN}(G) = \mathbb{E}_{Z\sim p_Z(Z)}[(D(G(Z))-1)^2] \quad (4)$$

Learning to Drive from Simulation without Real World Labels

Two directions generator: Perceptual Loss

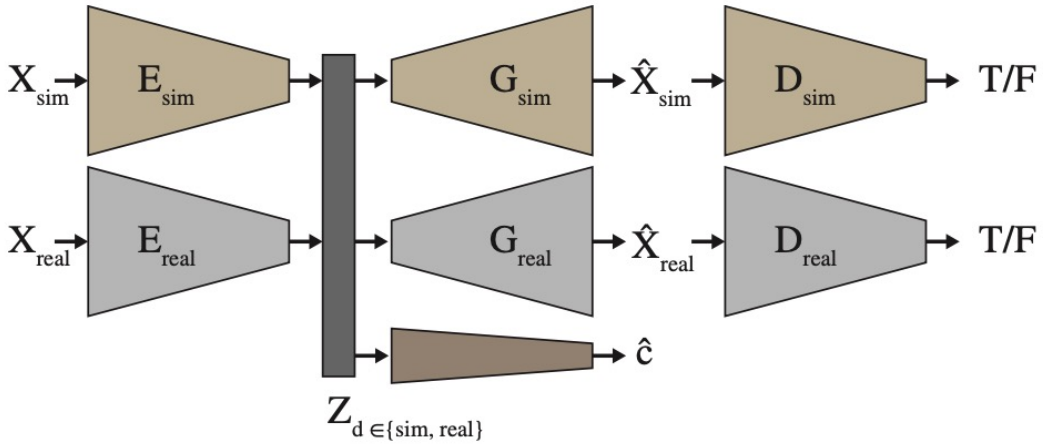


Fig. 2: Model architecture for domain-transfer from a simulated domain to real-world imagery, jointly learning control and domain translation. The encoders $E_{sim,real}$ map input images from their respective domains to a latent space Z which is used for predicting vehicle controls $\hat{c}$. This common latent space is learned through direct and cyclic losses as part of learning image-to-image translation, indicated conceptually in Figure 3 and in Section III-B.

5) Perceptual Loss: To encourage consistent semantic content across the two domains, we employed the use of a pre-trained VGG [27] model, which was applied to both the original and translated images. The perceptual loss $\mathcal{L}_{perceptual}$ was expressed as the difference between the features extracted from the last convolutional layer in the VGG16 model for a given input image and its translated counterpart. Extracted features were normalised via Instance Norm (IN) [31] following the result from Huang *et al.* [10] demonstrating that applying IN before computing the feature distance makes the perceptual loss more domain-invariant.

Learning to Drive from Simulation without Real World Labels

Two directions generator: Latent Reconstruction Loss

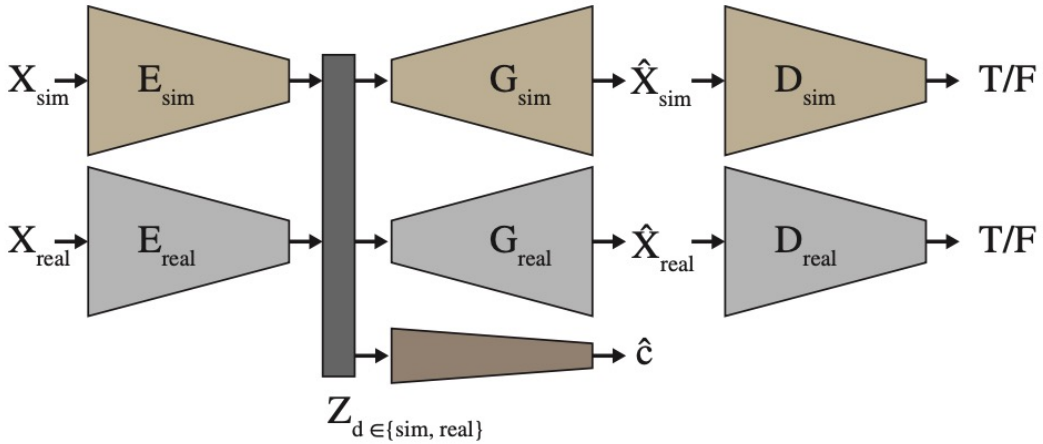


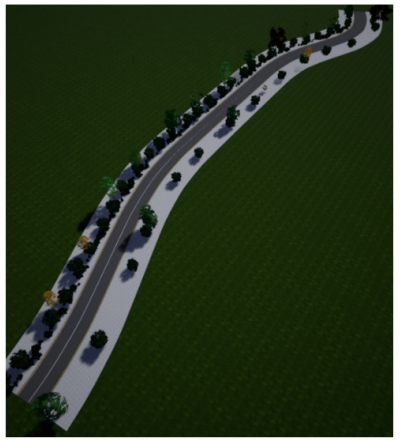
Fig. 2: Model architecture for domain-transfer from a simulated domain to real-world imagery, jointly learning control and domain translation. The encoders $E_{sim,real}$ map input images from their respective domains to a latent space Z which is used for predicting vehicle controls $\hat{c}$. This common latent space is learned through direct and cyclic losses as part of learning image-to-image translation, indicated conceptually in Figure 3 and in Section III-B.

6) Latent Reconstruction Loss: Ideally we wanted to encode the semantic content of the images within the latent space such that Z is independent of the domain from which an image came. We therefore applied a latent reconstruction loss $\mathcal{L}_{Zrecon}$, an L1 loss between the latent representation of an image Z_d and the reconstruction of the latent representation after it was decoded to the other domain and then encoded once more, $Z_d^{recon} = E_{d'}(G_{d'}(Z_d))$.

Learning to Drive from Simulation without Real World Labels

Experiment

(a) The 250m real-world rural driving route, coloured in blue.



(b) A procedurally generated curvy driving route in simulation.

(c) Real-world rural road.

(d) Simulated road.

Fig. 4: Rural setting: aerial views of the real-world (a) and simulated driving routes (b), along with example images from each domain showing the perspective from the vehicle (c, d).

(a) Simulated urban world.

(b) The real-world autonomous vehicle used for all experiments.

(c) An example of the real-world urban roads used for closed loop testing in Cambridge, UK.

Fig. 6: Urban setting: illustrations of the simulated and real-world domains for urban road scenes. Despite quite a large appearance change from the cartoon-like simulated world to the real-world, our model is able to successfully transfer the learned control policy to drive in in the real world, with no real world labels.

Learning to Drive from Simulation without Real World Labels

Experiment

TABLE II: Open-loop control metrics on the simulation and real test datasets from Table I.

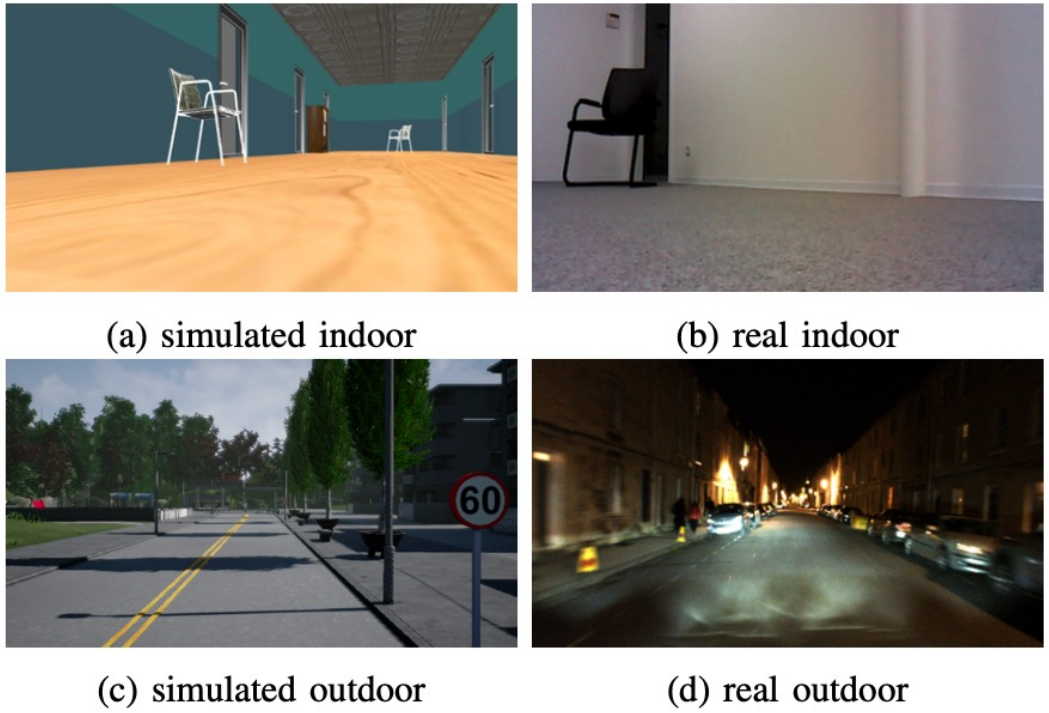
	Simulation		Real	
	MAE	Bal-MAE	MAE	Bal-MAE
Drive-Straight	0.043	0.087	**0.019**	0.093
Simple Transfer	0.05	0.055	0.265	0.272
Real-to-Sim Translation	-	-	0.261	0.234
Sim-to-Real Translation	-	-	0.059	**0.045**
Latent Feature ADA	0.040	0.047	0.032	0.071
Ours	**0.017**	**0.018**	0.081	0.087

TABLE III: On-vehicle performance, driving 3km on the rural driving route in 4a. For policies unable to drive a lap with ≤ 1 intervention, we terminated after one 250m lap (†).

	Mean distance / intervention (metres)
Drive-straight	$23^{\dagger}$
Simple Transfer	$9^{\dagger}$
Real-to-Sim Translation	$10^{\dagger}$
Sim-to-Real Translation	$28^{\dagger}$
Latent Feature ADA	$15^{\dagger}$
Ours	No intervention over 3km

VR Goggles for Robots: Real-to-sim Domain Adaptation for Visual Control

Observation adaptation for indoor-robot and outdoor autonomous driving



(a) simulated indoor (b) real indoor

(c) simulated outdoor (d) real outdoor

Fig. 6: Samples from the simulated environment (left) and the real world (right) used in our indoor (top) and outdoor (bottom) navigation experiments.

VR Goggles for Robots: Real-to-sim Domain Adaptation for Visual Control

VG Goggles

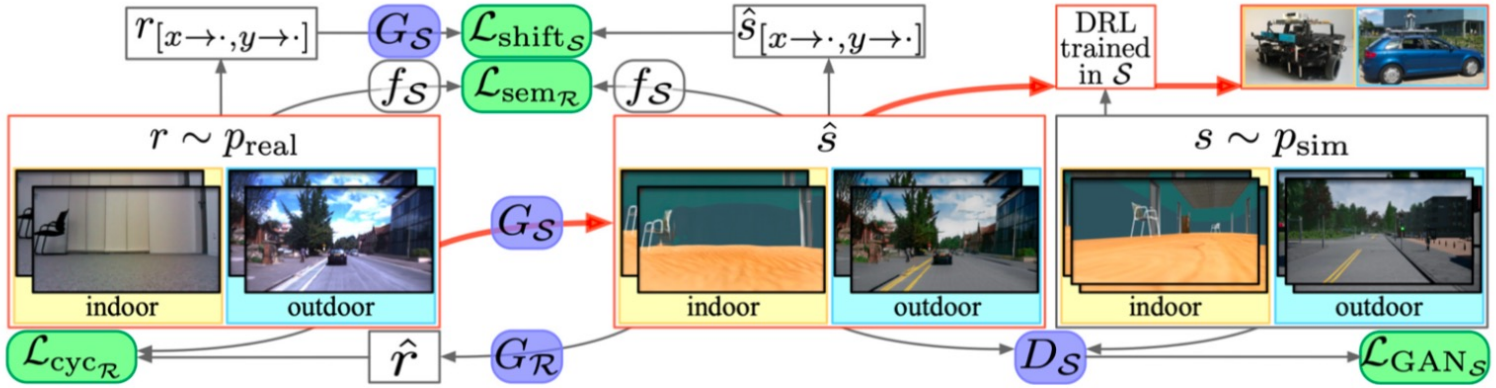


Fig. 1: The *VR-Goggles* pipeline. We depict the computation of the losses $\mathcal{L}_{\text{GAN}_{\mathcal{S}}}$, $\mathcal{L}_{\text{cyc}_{\mathcal{R}}}$, $\mathcal{L}_{\text{sem}_{\mathcal{R}}}$ and $\mathcal{L}_{\text{shift}_{\mathcal{S}}}$. We present both *outdoor* and *indoor* scenarios, where the adaptation for the *outdoor* scene is trained with the semantic loss $\mathcal{L}_{\text{sem}}$ (since its simulated domain *CARLA* has ground truth semantic labels to train a segmentation network $f_{\mathcal{S}}$), and the *indoor* one without (since its simulated domain *Gazebo* does not provide semantic ground truth). The components marked in *red* are those involved in the final deployment: a real sensor reading is captured ($r \sim p_{\text{real}}$), then passed through the generator $G_{\mathcal{S}}$ to be translated into the simulated domain $\mathcal{S}$, where the DRL agents were originally trained; the translated image $\hat{s}$ is then fed to the DRL policy, which outputs control commands. For clarity, we skip the counterpart losses $\mathcal{L}_{\text{GAN}_{\mathcal{R}}}$, $\mathcal{L}_{\text{cyc}_{\mathcal{S}}}$, $\mathcal{L}_{\text{sem}_{\mathcal{S}}}$ and $\mathcal{L}_{\text{shift}_{\mathcal{R}}}$.

CycleGAN

$$\begin{aligned}\mathcal{L}_{\text{GAN}_{\mathcal{R}}}(G_{\mathcal{R}}, D_{\mathcal{R}}; \mathcal{S}, \mathcal{R}) =& \mathbb{E}_{p_{\text{real}}}\left[\log D_{\mathcal{R}}(r)\right] + \\ & \mathbb{E}_{p_{\text{sim}}}\left[\log(1 - D_{\mathcal{R}}(G_{\mathcal{R}}(s)))\right], \\ \mathcal{L}_{\text{GAN}_{\mathcal{S}}}(G_{\mathcal{S}}, D_{\mathcal{S}}; \mathcal{R}, \mathcal{S}) =& \mathbb{E}_{p_{\text{sim}}}\left[\log D_{\mathcal{S}}(s)\right] + \\ & \mathbb{E}_{p_{\text{real}}}\left[\log(1 - D_{\mathcal{S}}(G_{\mathcal{S}}(r)))\right],\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{\text{cyc}_{\mathcal{R}}}(G_{\mathcal{S}}, G_{\mathcal{R}}; \mathcal{R}) &= \mathbb{E}_{p_{\text{real}}}\left[||G_{\mathcal{R}}(G_{\mathcal{S}}(r)) - r||_1\right], \\ \mathcal{L}_{\text{cyc}_{\mathcal{S}}}(G_{\mathcal{R}}, G_{\mathcal{S}}; \mathcal{S}) &= \mathbb{E}_{p_{\text{sim}}}\left[||G_{\mathcal{S}}(G_{\mathcal{R}}(s)) - s||_1\right].\end{aligned}$$

VR Goggles for Robots: Real-to-sim Domain Adaptation for Visual Control

VG Goggles

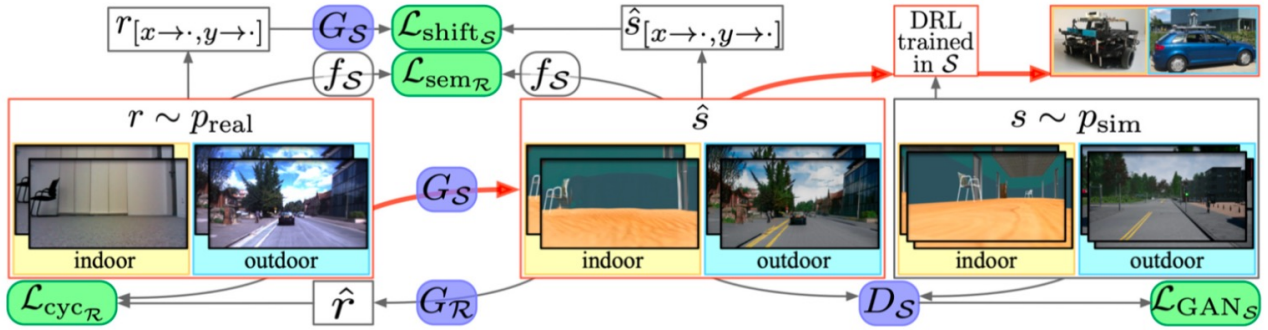


Fig. 1: The *VR-Goggles* pipeline. We depict the computation of the losses $\mathcal{L}_{\text{GAN}_{\mathcal{S}}}$, $\mathcal{L}_{\text{cyc}_{\mathcal{R}}}$, $\mathcal{L}_{\text{sem}_{\mathcal{R}}}$ and $\mathcal{L}_{\text{shift}_{\mathcal{S}}}$. We present both *outdoor* and *indoor* scenarios, where the adaptation for the *outdoor* scene is trained with the semantic loss $\mathcal{L}_{\text{sem}}$ (since its simulated domain *CARLA* has ground truth semantic labels to train a segmentation network $f_{\mathcal{S}}$), and the *indoor* one without (since its simulated domain *Gazebo* does not provide semantic ground truth). The components marked in *red* are those involved in the final deployment: a real sensor reading is captured ($r \sim p_{\text{real}}$), then passed through the generator $G_{\mathcal{S}}$ to be translated into the simulated domain $\mathcal{S}$, where the DRL agents were originally trained; the translated image $\hat{s}$ is then fed to the DRL policy, which outputs control commands. For clarity, we skip the counterpart losses $\mathcal{L}_{\text{GAN}_{\mathcal{R}}}$, $\mathcal{L}_{\text{cyc}_{\mathcal{S}}}$, $\mathcal{L}_{\text{sem}_{\mathcal{S}}}$ and $\mathcal{L}_{\text{shift}_{\mathcal{R}}}$.

Semantic Loss

Assuming that for images from domain $\mathcal{S}$, the ground truth semantic labels $\mathbf{Y}$ are available, a semantic segmentation network $f_{\mathcal{S}}$ can be obtained by minimizing the *cross-entropy* loss $\mathbb{E}_{s\sim\mathcal{S}}[\text{CrossEnt}(\mathbf{Y}_s, f_{\mathcal{S}}(s))]$. We further assume that the ground truth semantic for domain $\mathcal{R}$ is lacking (which is the case for most real scenarios), meaning that $f_{\mathcal{R}}$ is not easily obtainable. In this case, we use $f_{\mathcal{S}}$ to generate "semi" semantic labels for domain $\mathcal{R}$. Then semantically consistent image translation can be achieved by minimizing the following losses, which imposes consistency between the semantic maps of the input and that of the generated output:

$$\mathcal{L}_{\text{sem}_{\mathcal{R}}}(G_{\mathcal{S}}; \mathcal{R}, f_{\mathcal{S}}) = \mathbb{E}_{p_{\text{real}}}[\text{CrossEnt}(f_{\mathcal{S}}(r), f_{\mathcal{S}}(G_{\mathcal{S}}(r)))].$$
$$\mathcal{L}_{\text{sem}_{\mathcal{S}}}(G_{\mathcal{R}}; \mathcal{S}, f_{\mathcal{S}}) = \mathbb{E}_{p_{\text{sim}}}[\text{CrossEnt}(f_{\mathcal{S}}(s), f_{\mathcal{S}}(G_{\mathcal{R}}(s)))],$$

VR Goggles for Robots: Real-to-sim Domain Adaptation for Visual Control

Experiment

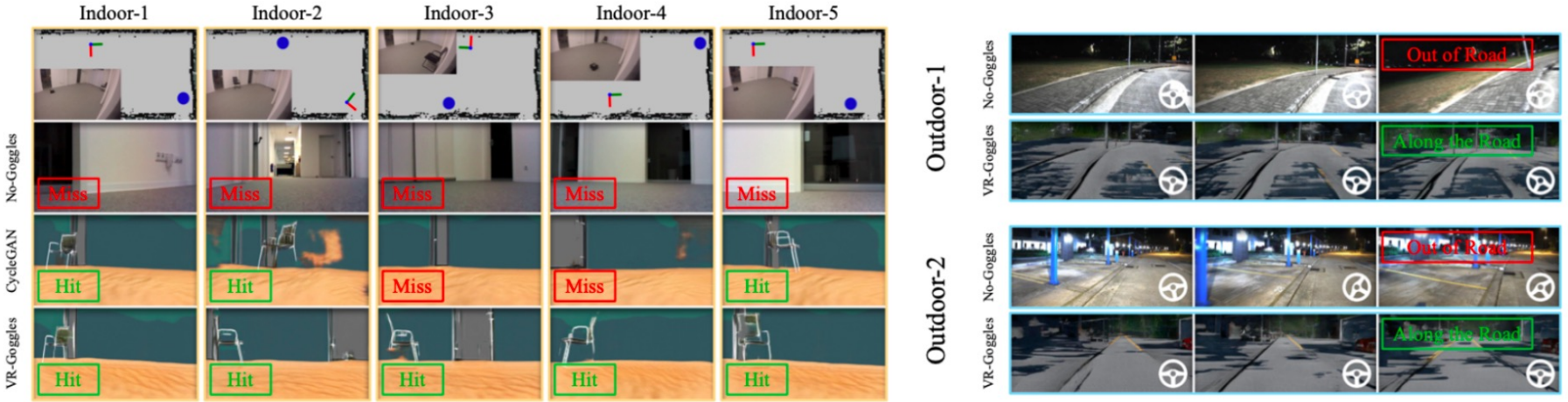


Fig. 5: Real-world visual control experiments. ***Indoor*** (yellow):. A navigation policy is firstly trained in a simulated environment (Fig. 6a) that is able to navigate to chairs based on visual inputs. Without retraining or finetuning, our proposed *VR-Goggles* enables the mobile robot to directly deploy this policy in a real office environment (Fig. 6b), achieving 100% success rate in a set of real-world experiments. Here *Miss* refers to test runs where the agent stays put or rotate in place and simply ignores the chair even when they are in sight as the policy trained in the simulation could not cope with the drastically visually different inputs (*No-Goggles*), or due to the inconsistency of the translated subsequent outputs which hinders the successful fulfilment of the goal-reaching task (*CycleGAN*). *Hit* refers to frames where the agent captures the chair in sight and outputs commands to move towards it. ***Outdoor*** (cyan): An autonomous driving policy (via conditional imitation learning [28]) is trained in *Carla* daytime (Fig. 6c), a *VR-Goggles* model is trained to translate between *Carla* daytime and *Robotcar* nighttime (Fig. 6d), which enables the real-world nighttime deployment of the trained policy.

Domain Randomization and Domain Adaptation are not Mutually Exclusive

[1] DARLA: Improving Zero-Shot Transfer in Reinforcement Learning

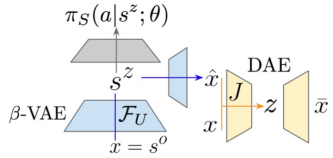
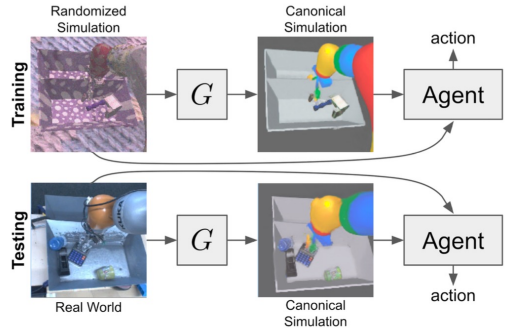


Figure 1. Schematic representation of DARLA. Yellow represents the denoising autoencoder part of the model, blue represents the β-VAE part of the model, and grey represents the policy learning part of the model.

[2] Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks



[3] Meta Reinforcement Learning for Sim-to-real Domain Adaptation

[4] Unsupervised Domain Adaptation with Dynamics-Aware Rewards in Reinforcement Learning

Table of Contents

1. Sim2Real Transfer for Reinforcement Learning
2. Domain Randomization for Sim2Real RL
3. Domain Adaptation for Sim2Real RL
4. **Our work: Cross-Modal Domain Adaptation with Sequential structure (CODAS)**

The Framework of Unsupervised Domain Adaptation in Sim2Real RL

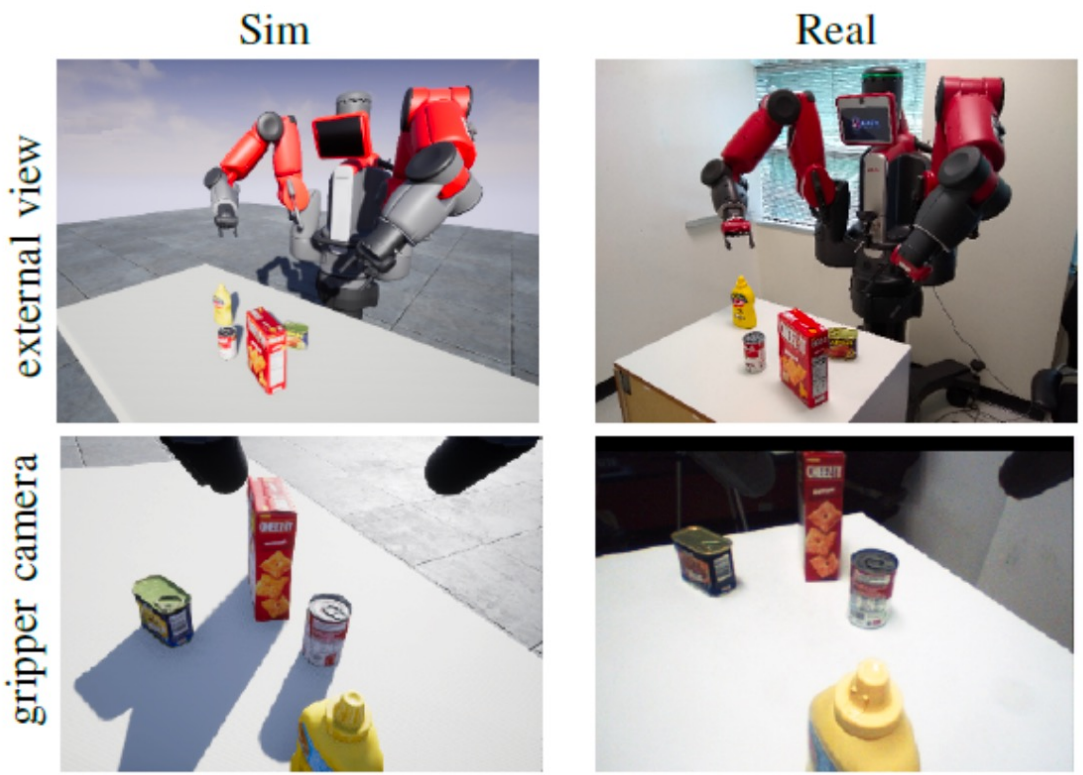
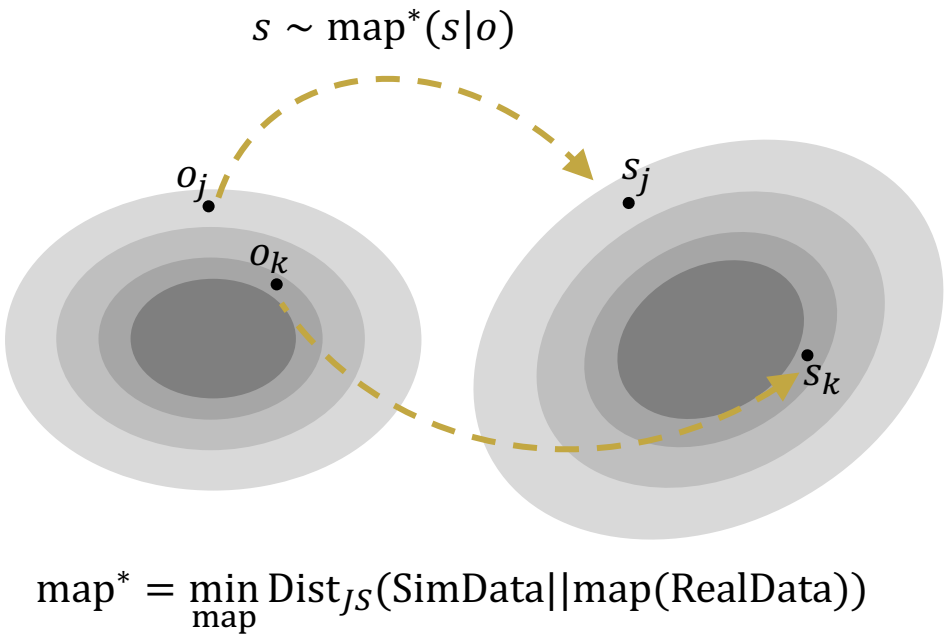


Fig. An example of reality-gap which is the core challenge in Sim2Real RL [1]



$$\text{map}^* = \min_{\text{map}} \text{Dist}_{JS}(\text{SimData}||\text{map}(\text{RealData}))$$

-> trained via GAN-style algorithms

Fig. The framework of Unsupervised Domain Adaptation for Sim2Real RL

Unsupervised domain adaptation (UDA) learns a mapping function to align the data distribution of the source and the target domain to handle the challenge of reality-gap on observation-space for Sim2Real RL

Cross-Modal UDA: A cost-efficient Framework for Sim2Real RL

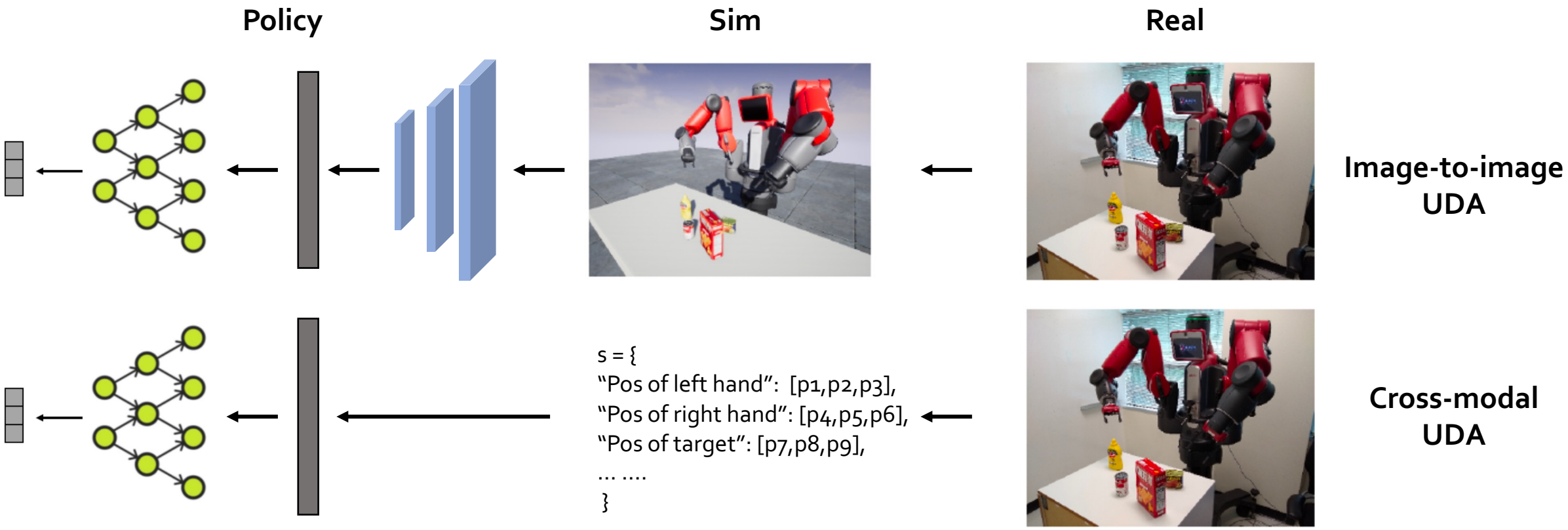


Image-to-image UDA introduce three extra cost, which is ignored in discussion in previous work.

1. human labor of building a visual simulator
2. huge computation resource required by running the simulator
3. inferior policy training on visual simulator

⇨ **Can be solved in Cross-modal UDA**

Ill-posedness of the raw objective in current UDA solutions

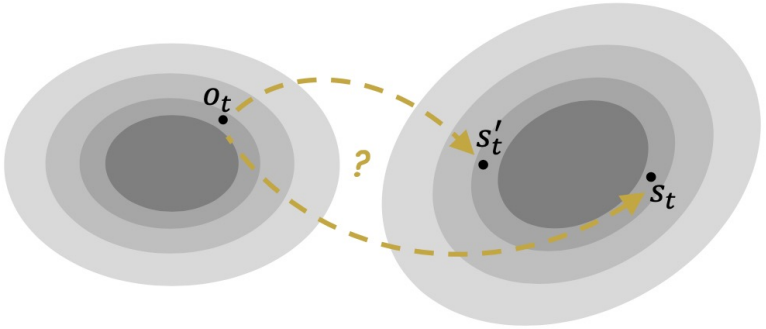


Fig. An example of ill-posedness of the distribution minimization objective in current UDA algorithms

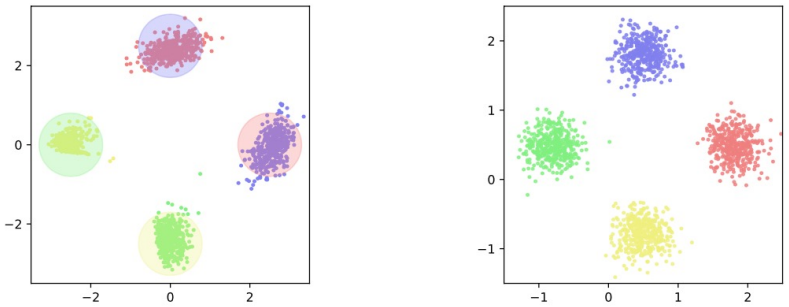


Fig. A toy example of ill-posed UDA

Since s_t and s_t' have similar probabilities, mapping an instance o_t to anywhere of a similar probability in the source domain is "reasonable" if we only consider distribution matching.

In image-to-image UDA, current methods rely on additional constraints on modality consistency to handle the problem implicitly

- special model structure [2], e.g. U-Net, Cycle-GAN;
- auxiliary losses, e.g. geometry consistency [3];

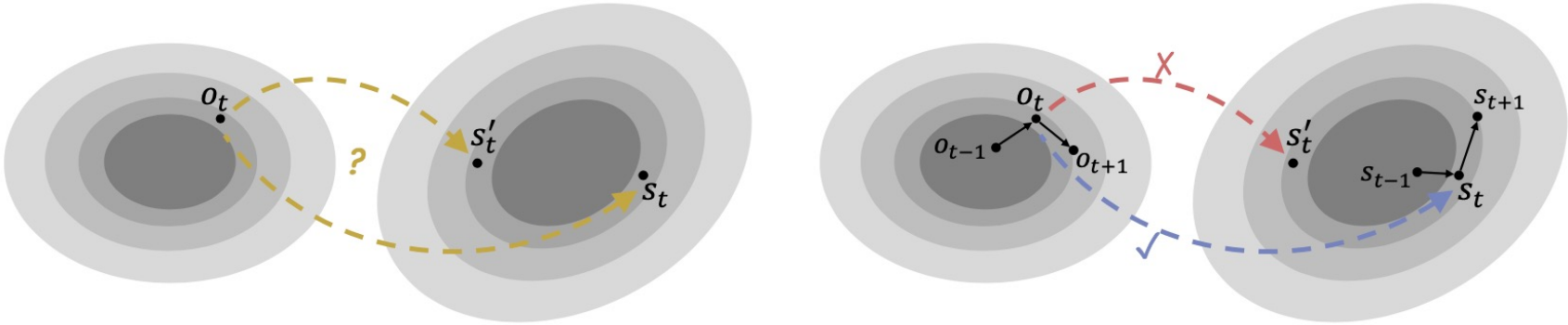
However, these constraints cannot hold anymore in Cross-modal UDA setting.

Our research question: Can we handle the ill-posedness of the objective directly?

Table of Contents

1. Background and Motivation
2. **Cross-Modal Domain Adaptation with Sequential structure (CODAS)**
3. Experiment
4. Take-home Messages

Any other potential way to handle the ill-posedness of the objective?



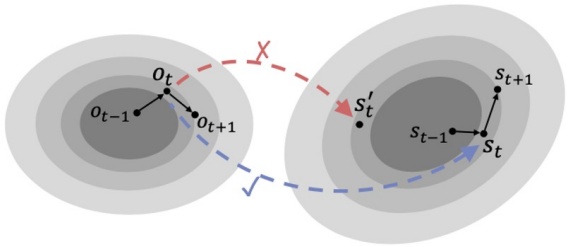
(a) Mapping only considering state-distribution matching

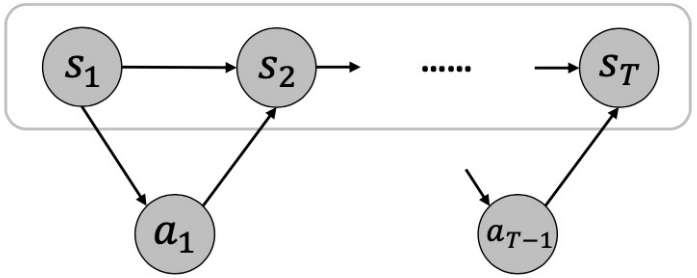
(b) Mapping with sequential structure

If we can make use of **the sequential structure in the Markov Decision Processes**, the historical information will give us the ability to identify the difference between $s_t{}'$ and s_t,

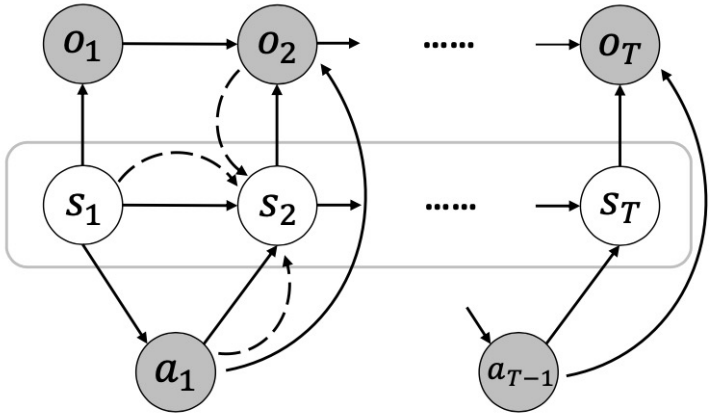
then the proposed ill-posedness of the objective will be fixed.

Reformulate the objective of UDA in RL based on the framework of variational inference





(a) Generation process in the source domain



(b) Generation and inference process in the target domain

Fig. The generation and inference process of UDA based on the framework of variational inference

$$\min_{\phi} \mathbb{E}_{\tau^o} \left[D_{\mathrm{KL}} \left[q_\phi(\tau^s \mid \tau^o) \,||\, p(\tau^s \mid \tau^o) \right] \right]$$

$$\Downarrow (ELBO)$$

$$\max_{\phi,\theta} \mathbb{E}_{\tau^o} \left[\mathbb{E}_{\hat{\tau}^s \sim q_\phi(\tau^s \mid \tau^o)} \left[\log p_\theta \left(\tau^o \mid \hat{\tau}^s \right) \right] - D_{\mathrm{KL}} \left[q_\phi \left(\tau^s \mid \tau^o \right) \,||\, p \left(\tau^s \right) \right] \right]$$

Differentiable Optimization Objectives

$$\max_{\phi,\theta} \mathbb{E}_{\tau^o}\left[\mathbb{E}_{\hat{\tau}^s \sim q_\phi(\tau^s|\tau^o)}\left[\log p_\theta\left(\tau^o \mid \hat{\tau}^s\right)\right] - D_{\mathrm{KL}}\left[q_\phi\left(\tau^s \mid \tau^o\right) \| \, p\left(\tau^s\right)\right]\right]$$

Reconstruction loss

Trajectory-distribution mismatch loss

Discriminator loss

$$\max_{\phi,\theta} \mathbb{E}_{\tau^o \sim \mathcal{D}^o}\Big[\sum_{t=1}^{T} \mathbb{E}_{\hat{s}_t \sim q_\phi(s_t|\hat{s}_{t-1},a_{t-1},o_t)}\Big[\log p_\theta(o_t \mid \hat{s}_t, o_{t-1}, a_{t-1}) - \lambda_D \log\left(1 - D_{\omega^*}(\hat{s}_t, a_t, h_{t-1})\right)\Big]\Big],$$

$$\text{s.t. } \omega^* = \arg\max_{\omega} \; \mathbb{E}_{\tau^s \sim \mathcal{D}^s}\Big[\sum_{t=1}^{T} \log D_\omega(s_t, a_t, h_{t-1})\Big]$$

Embedded Dynamics Model for Stable Training

Inference Function only outputs a small Δs_t. The main part is from **Embedded DM** (p_ϕ).

$$\hat{s}_t = p_\varphi(s_{t-1}, a_{t-1}) + \alpha \Delta s_t,$$
$$\text{where} \Delta s_t \sim q_\phi(\Delta s \mid s_{t-1}, a_{t-1}, o_t)$$

The parameters of **Embedded DM** are copied from a DM trained by:

$$\min_{\varphi} \mathbb{E}_{(s,a,s') \sim D^s \cup D^{\hat{s}}} \left[\left(p_\varphi(s, a) - s' \right)^2 \right]$$

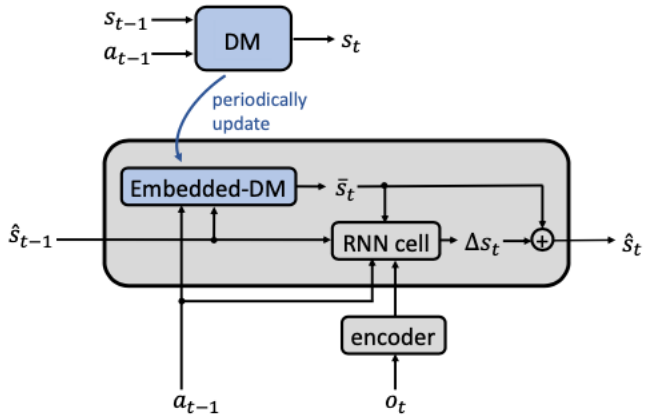


Figure: Detailed Structure of the Inference Function with Embedded DM

Table of Contents

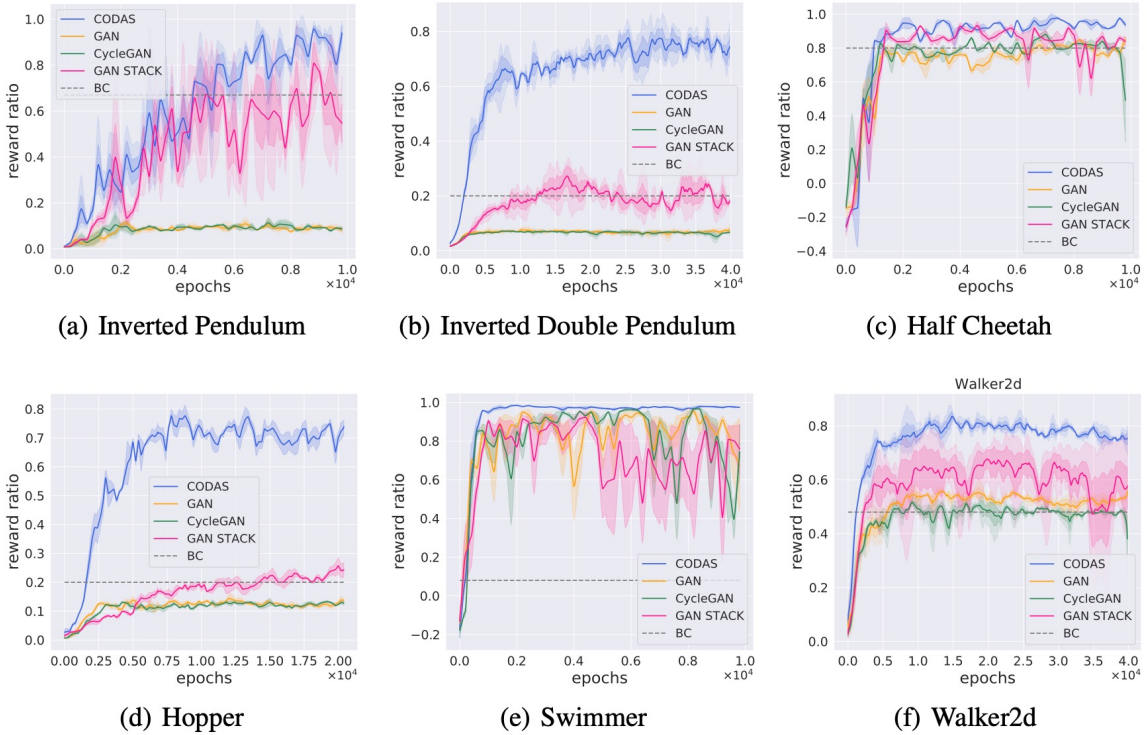
1. Background and Motivation

2. Cross-Modal Domain Adaptation with Sequential structure (CODAS)

3. **Experiment**

4. Take-home Messages

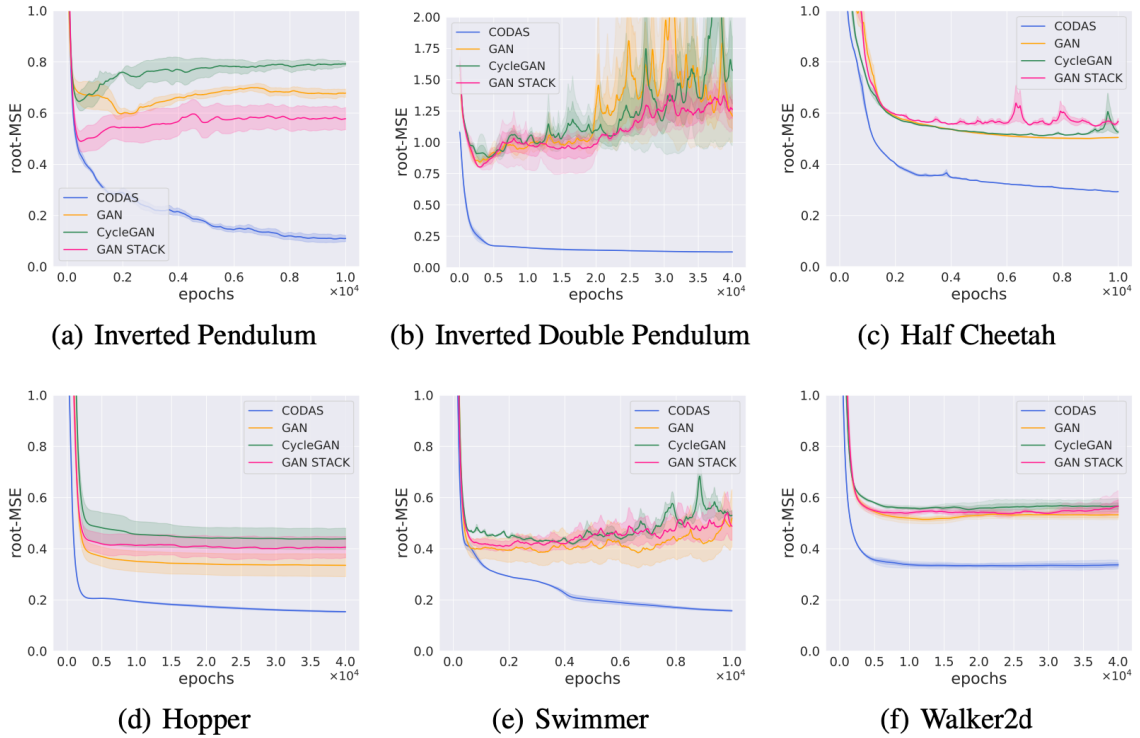
Comparative Evaluation in MuJoCo Tasks



(a) Inverted Pendulum (b) Inverted Double Pendulum (c) Half Cheetah

(d) Hopper (e) Swimmer (f) Walker2d

Figure 4: Training curves of different methods on MuJoCo.



(a) Inverted Pendulum (b) Inverted Double Pendulum (c) Half Cheetah

(d) Hopper (e) Swimmer (f) Walker2d

Figure 6: Root mean squared error between mapped states and ground-truth states. The solid lines denote the mean value. The shadows denote the standard deviation.

For all of the tasks, CODAS can map the correct states (i.e., with the smallest MSE-loss to the oracle states) and the performance of the deployment policies reach reasonable performance (75%~100%)

Visualization of the Learned Mapping on MuJoCo Tasks

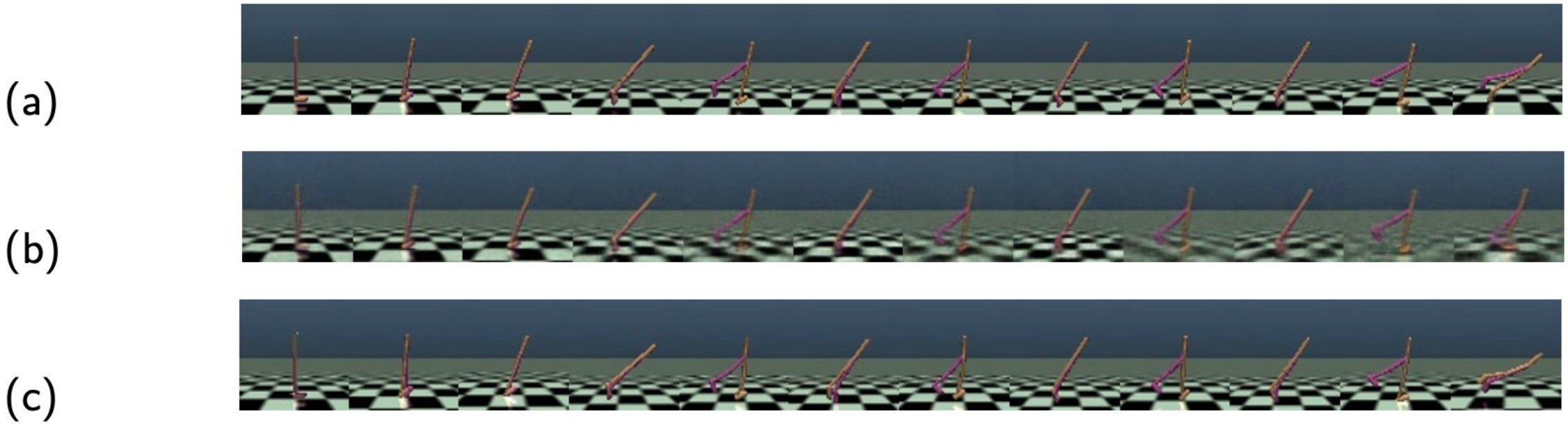


Fig. A visual illustration of (a) original images, (b) reconstructed images, and (c) re-rendered images of the mapped states in Hopper.

Both reconstructed images and re-rendered images match the original ones well. Re-rendered images can even match the original ones well in the last falling frames which are sparse in the dataset.

Performance on Robot Hand Manipulation Tasks

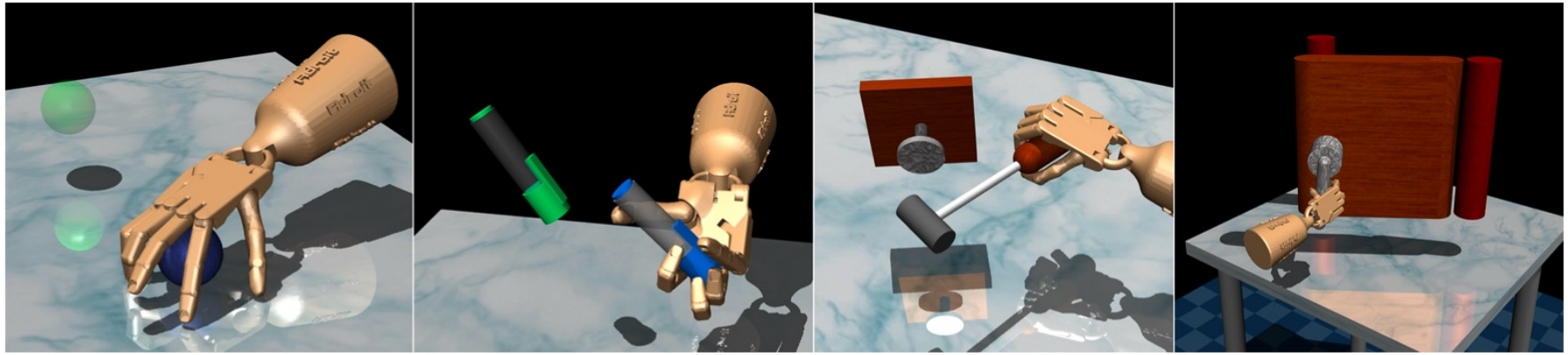


Table: The reward ratio in hand-manipulation tasks

Tasks	hammer	pen	door	relocate
Reward Ratio	0.820	0.701	0.886	0.090

In three out of four tasks, CODAS yields reasonable mapping functions for policy deployment.

Take-home Message

KEY point:

The Formulation of Variational Inference which considered the sequential structure in MDP can handle the ill-posedness of the objective and solve the UDA problem without relying on the knowledge of modality consistency.

Future work:

1. CODAS solve the UDA problem in a general way, it can be adopted to image-to-image UDA in theory;
2. If the ill-posedness of the CODAS objective still exist?
3. In the current formulation, we assume the policies/dynamics in the source and target domains are the same, which might not hold in real-world applications. By modeling the mismatching of dynamics models and data-collected policies into the CODAS framework, we can build a more practical UDA algorithm.

>> Thanks