Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

Sim2Real Transfer

Spring 2021, CMU 10-403

Katerina Fragkiadaki

Paradox

The requirement of large number of samples for model-free RL, **only possible in simulation**, renders model-free RL a **model-based** framework: we can’t do without the simulator.

Choices

We want to learn manipulation and locomotion policies, what do we do?

1. We use a Physics simulator, where Physics rules between objects and/or particles have been hand-coded by engineers. We train our policies there with reinforcement (trial-and-error) and/or demonstrations. We then **transfer** them to the real world.

2. We directly learn policies in the real world.

Choices

We want to learn manipulation and locomotion policies, what do we do?

1. We use a Physics simulator, where Physics rules between objects and/or particles have been hand-coded by engineers. We train our policies there with reinforcement (trial-and-error) and/or demonstrations. We then **transfer** them to the real world.

2. We directly learn policies in the real world.

3. We combine simulators with deeply learned residuals for action or next state prediction to compensate for wrong simulation models-> residual Physics

Pros of Simulation

- We can afford many samples
- Safe: we do not want to deploy partially trained policies in the real world
- Avoids wear and tear of the robot
- We can explore creative robot configurations

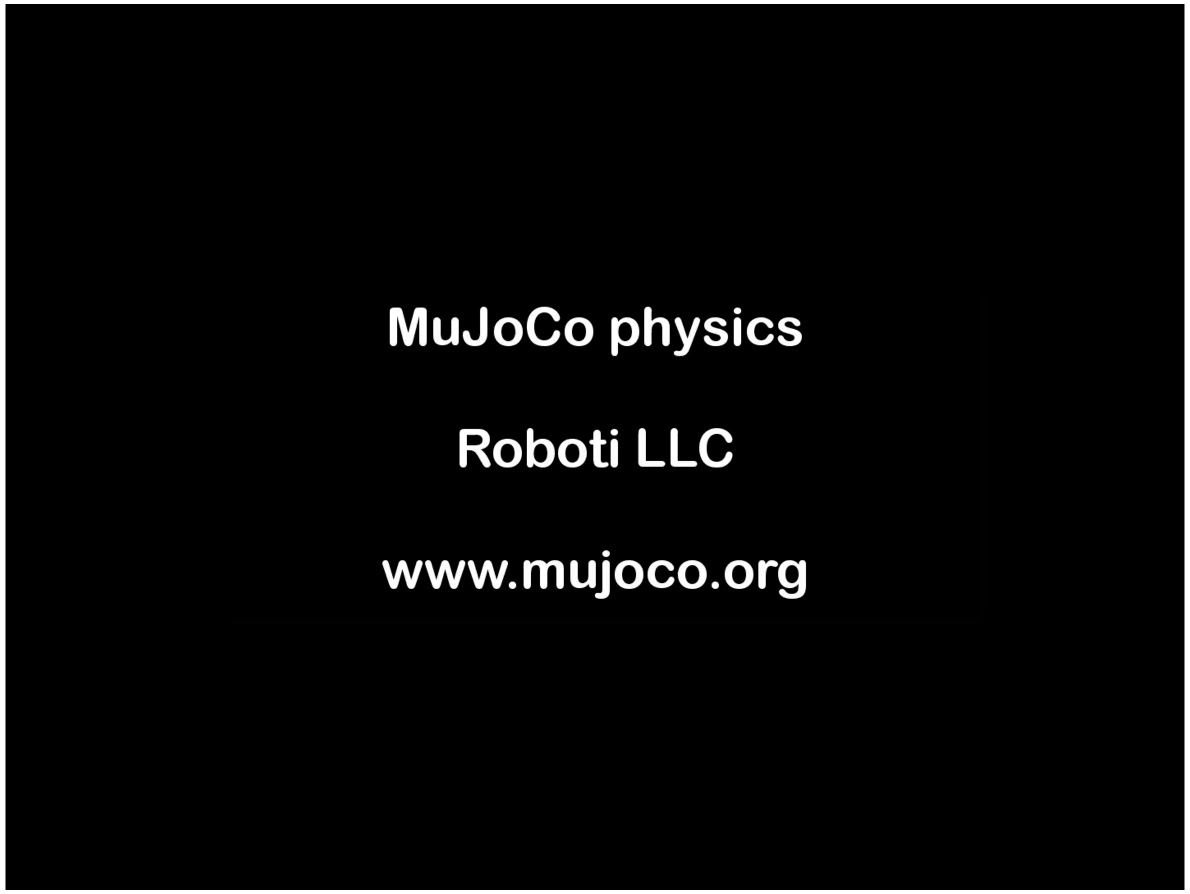
Cons of Simulation

- Under-modeling: It is hard to exactly replicate the real world and its physics and mechanics
- Large engineering effort into building the environment which we care to manipulate
- 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:
    1. observations
    2. dynamics

Result: Policies learnt in simulation usually do not directly transfer to the real world

Simulators

MuJoCo: rigid and deformable body simulator on a CPU



http://www.mujoco.org/image/home/mujocodemo.mp4

FLEX: particle based simulator on a GPU for rigid / soft bodies, fluids, gas.

https://www.youtube.com/watch?v=1o0Nuq71gI4

Sim2Real:What has shown to work

- Domain randomization (dynamics, visuals)
- Intelligent adaptive domain randomization (dynamics, visuals)
- Residual Physics: combine analytic models with deep learning
- Visual Abstraction: Learning from label images as opposed to pixel images-> semantic maps between simulation and real world are closer than textures
- Action Abstraction: Learning higher level policies, not low-level controllers because the low level dynamics are very different between Simulation and Reality

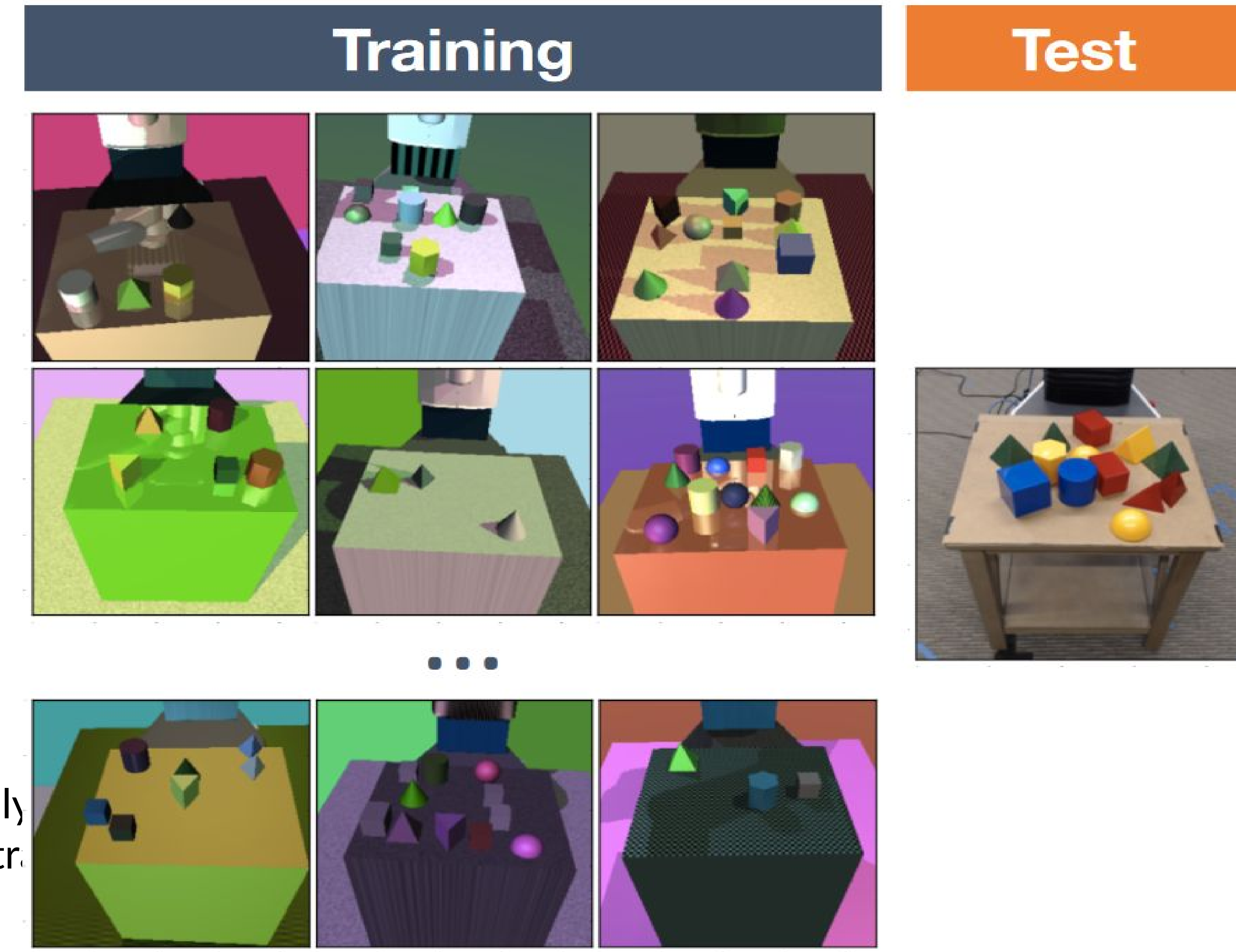
Sim2Real:What has shown to work

- Domain randomization (dynamics, visuals)
- Learning to adapt the textures of the simulator to match the real domain
- Learning to adapt the dynamics of the simulator to match the real domain
- Learning from label images as opposed to pixel images-> semantic maps between simulation and real world are closer than textures
- Abstraction: Learning higher level policies, not low-level controllers because the low level dynamics are very different between Simulation and Reality

What has shown to work

- Domain randomization (dynamics, visuals)
- Learning to adapt the textures of the simulator to match the real domain
- Learning to adapt the dynamics of the simulator to match the real domain
- Learning from label images as opposed to pixel images-> 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

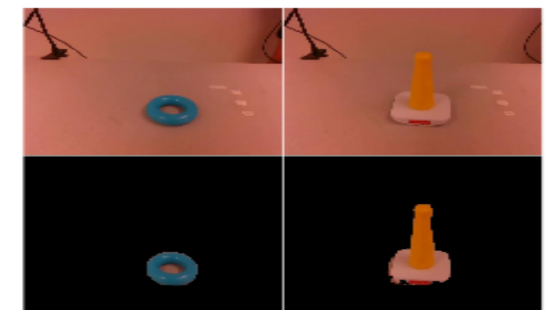


We create (automatically ... era viewpoints. We use
the simulation data to tr...

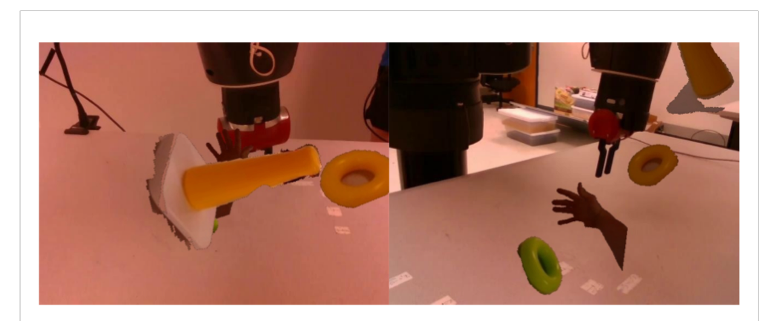
Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World, *Tobin et al.*

Data dreaming

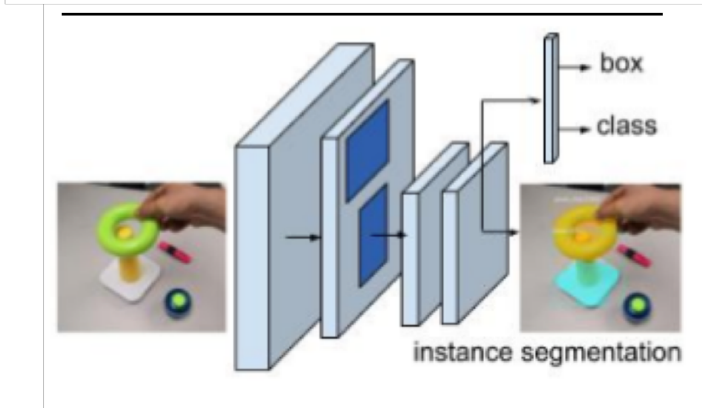
1. **Obtaining object masks**
   - background subtraction gives ground truth object masks



2. **Creating synthetic labelled data**
   - Massive augmentation of ground truth masks by random transformations/occlusions and random backgrounds



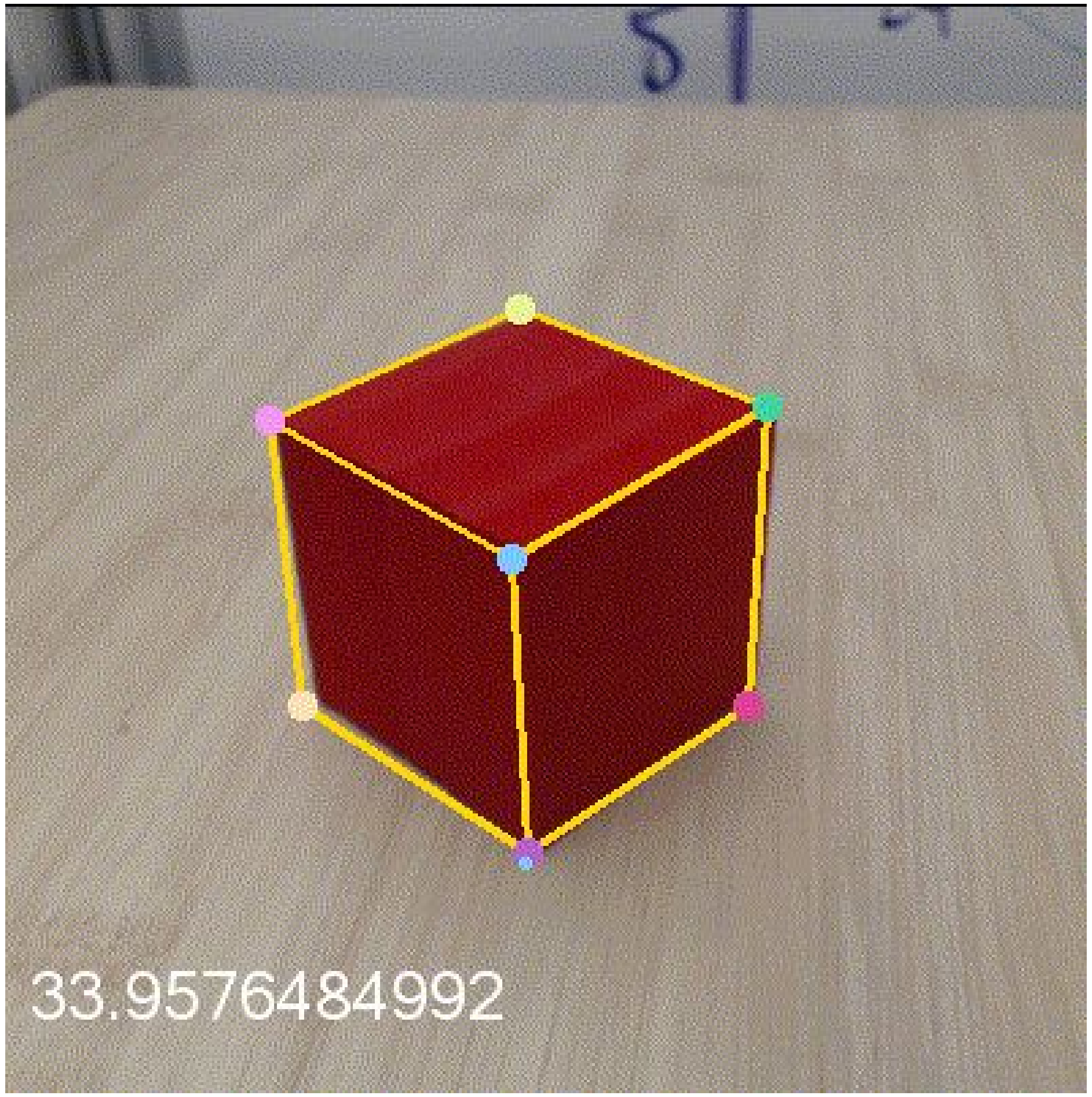
3. **Training object detectors**
   - Mask R-CNN



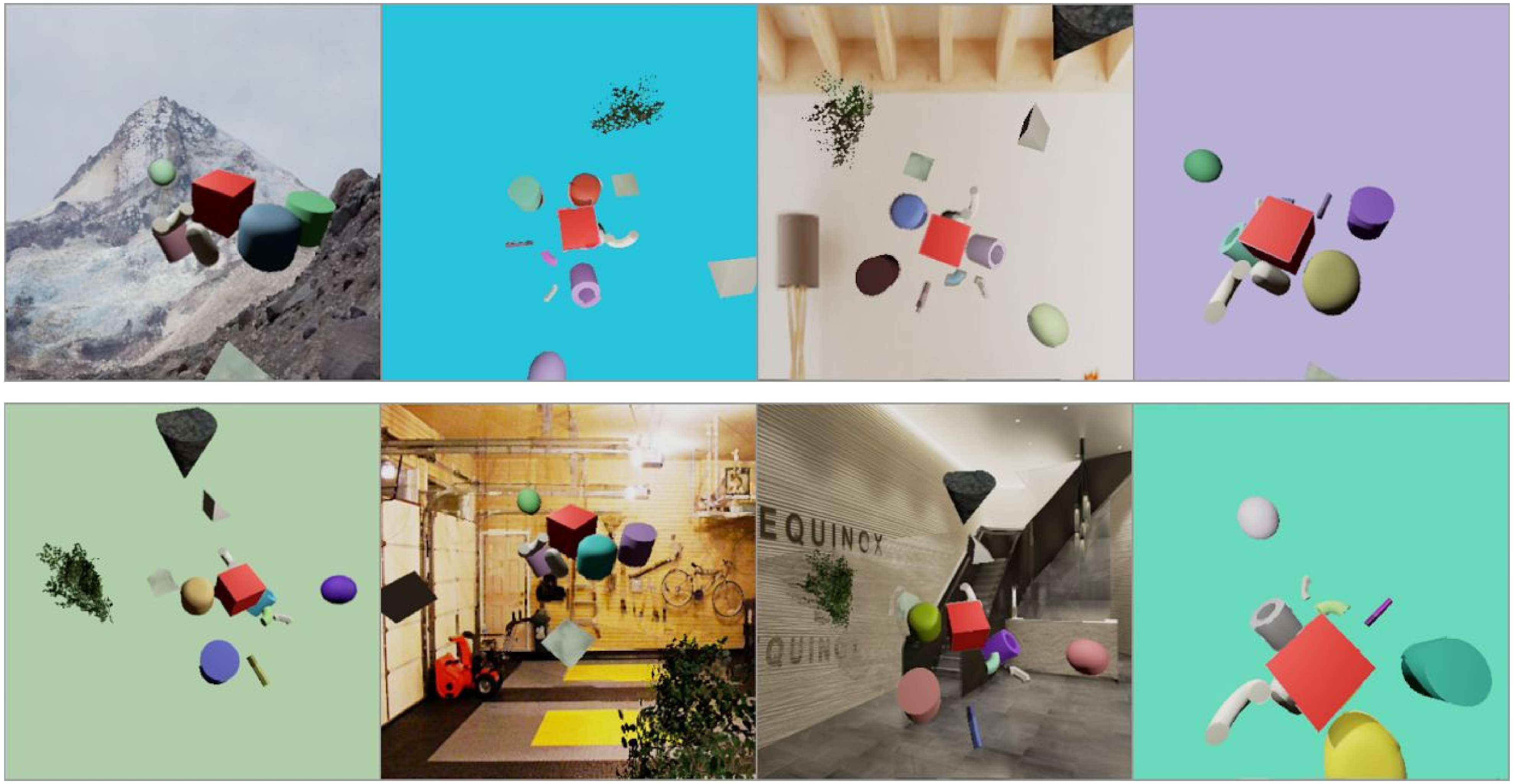
Data Dreaming for Object Detection: Learning Object-Centric State Representations for Visual Imitation, Sieb et al.

Let’s try a more fine grained task

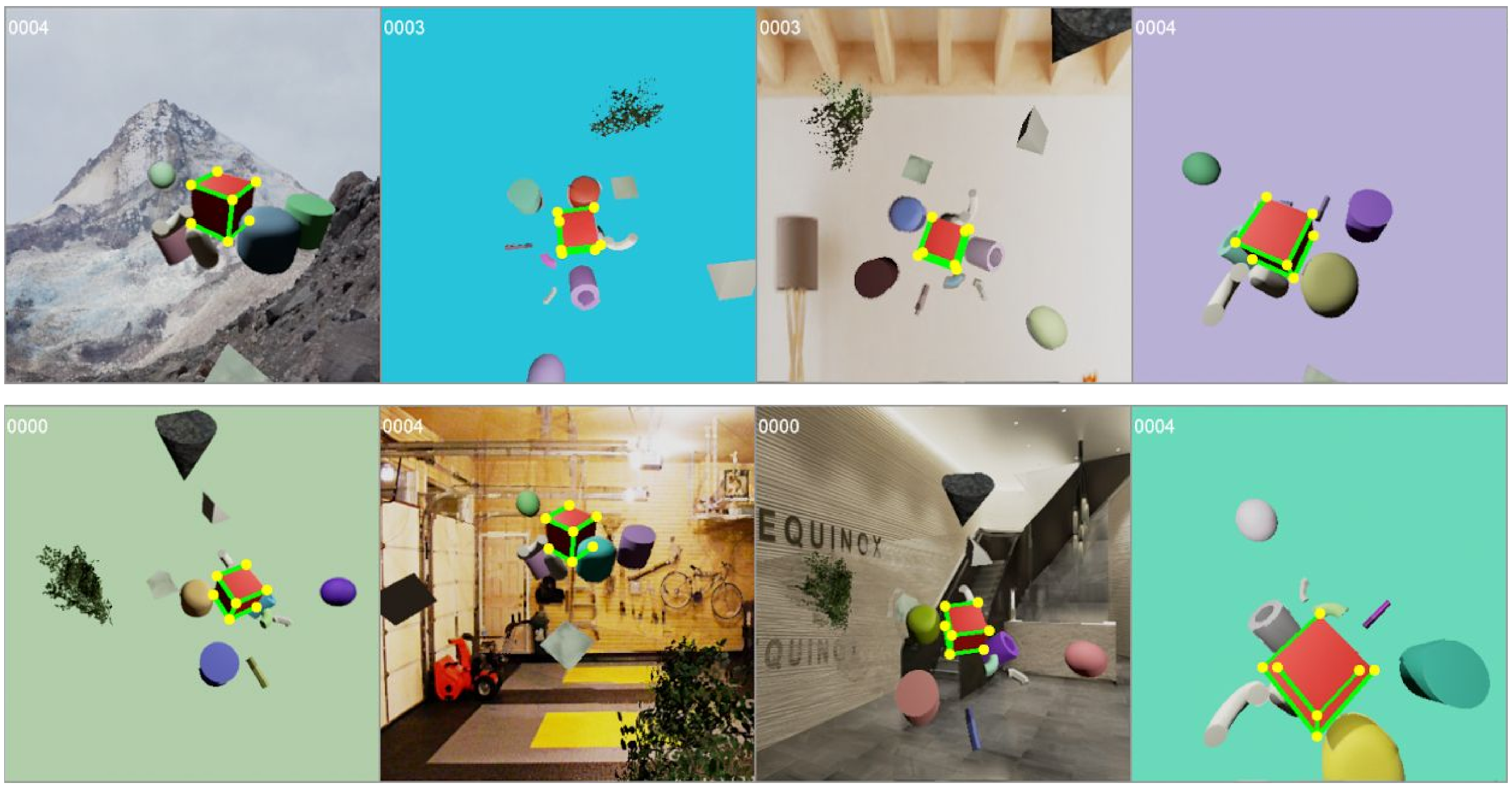
Cuboid Pose Estimation



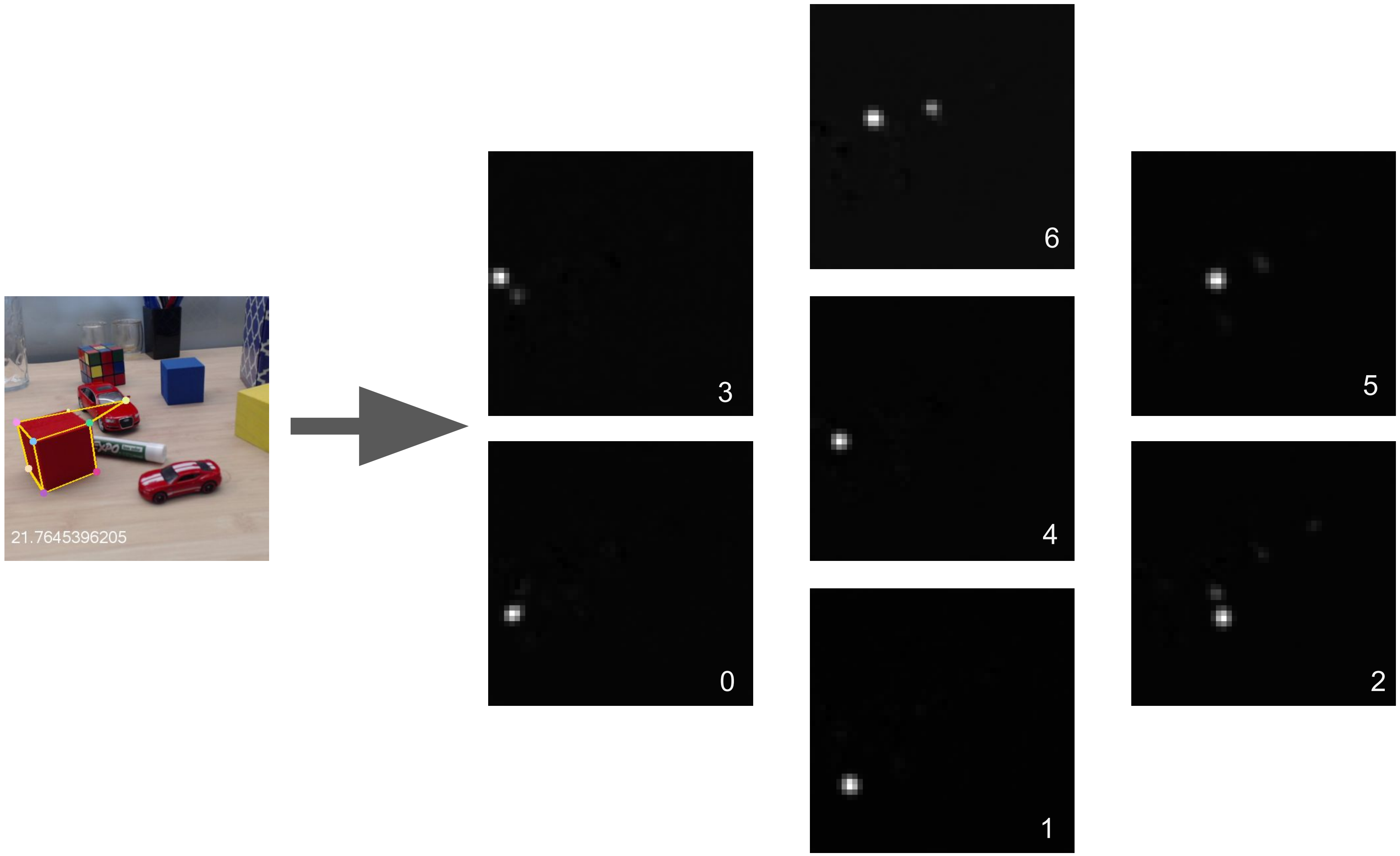
Synthetic data generation



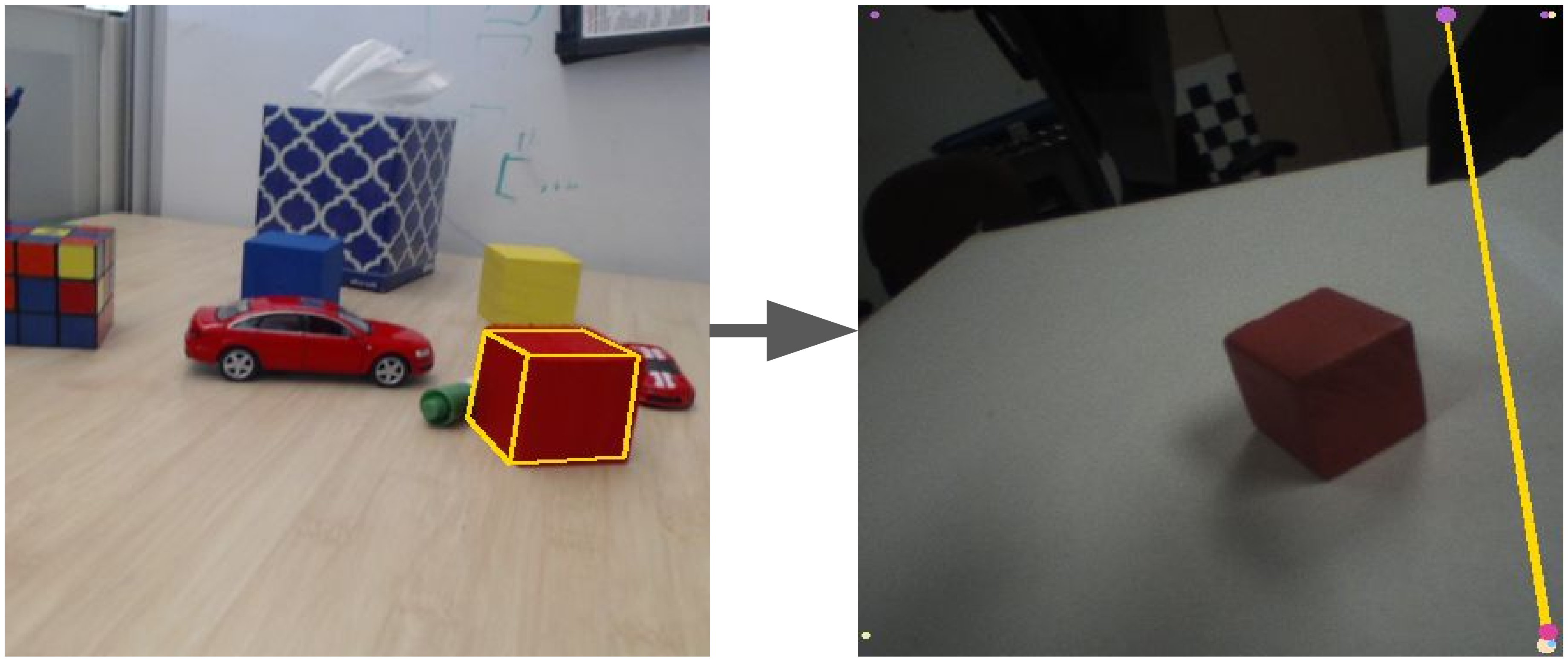
Synthetic data generation



Predicting vertex heatmaps



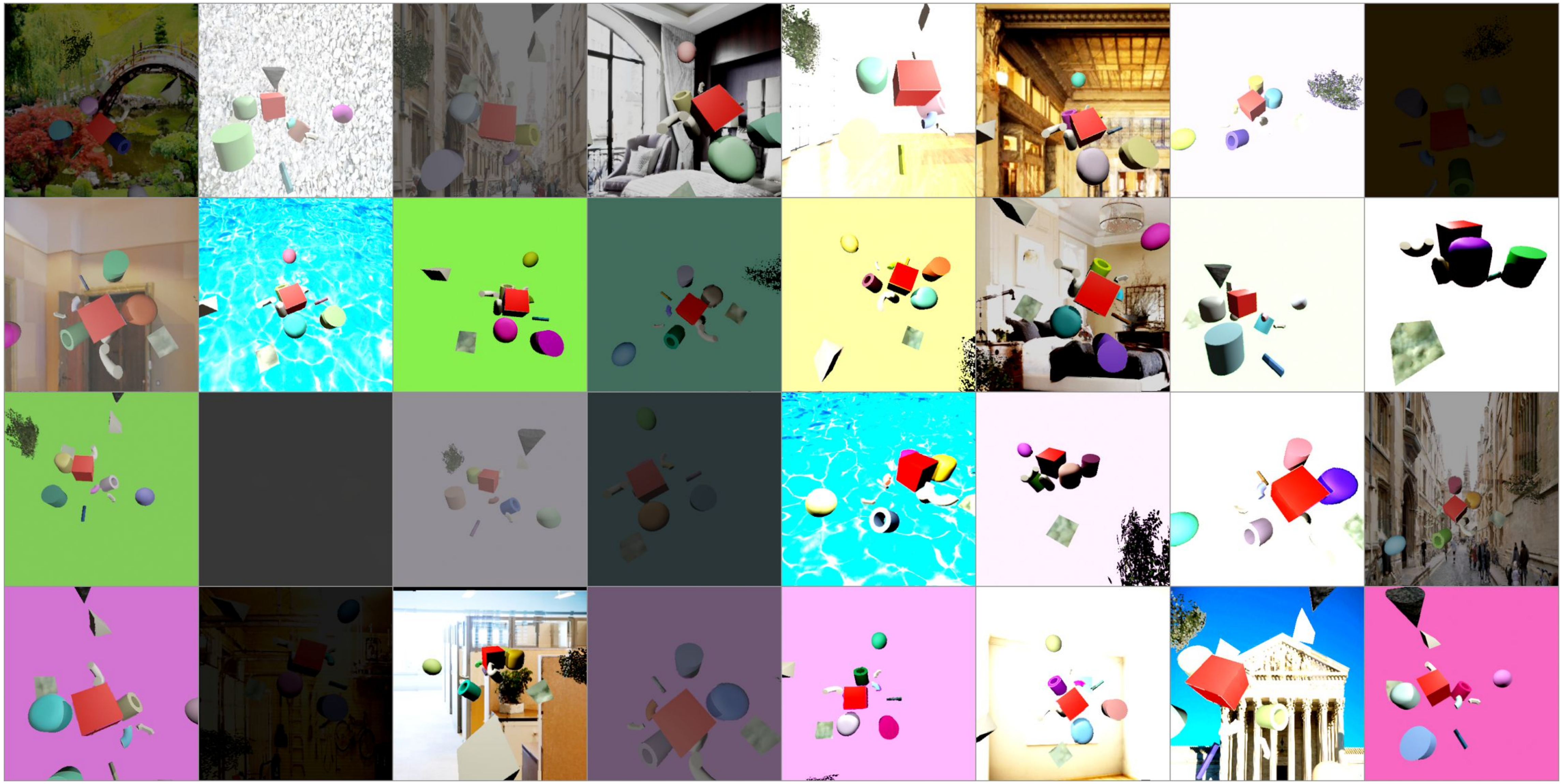
SIM2REAL



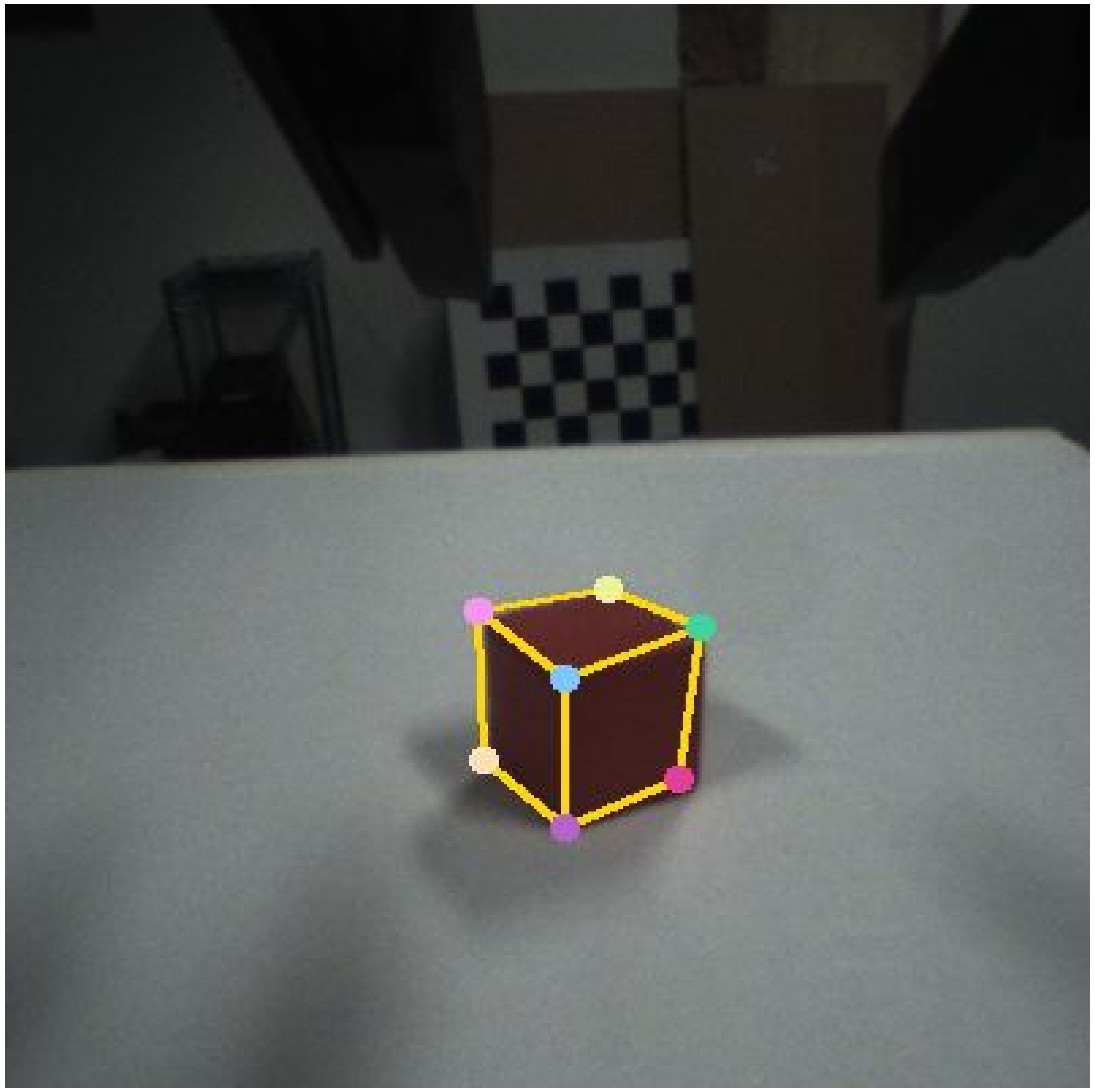
- Pose detector fails when the brightness of the image changes. Solution?
- Randomize also the brightness

Synthetic data generation

Data - Contrast and Brightness



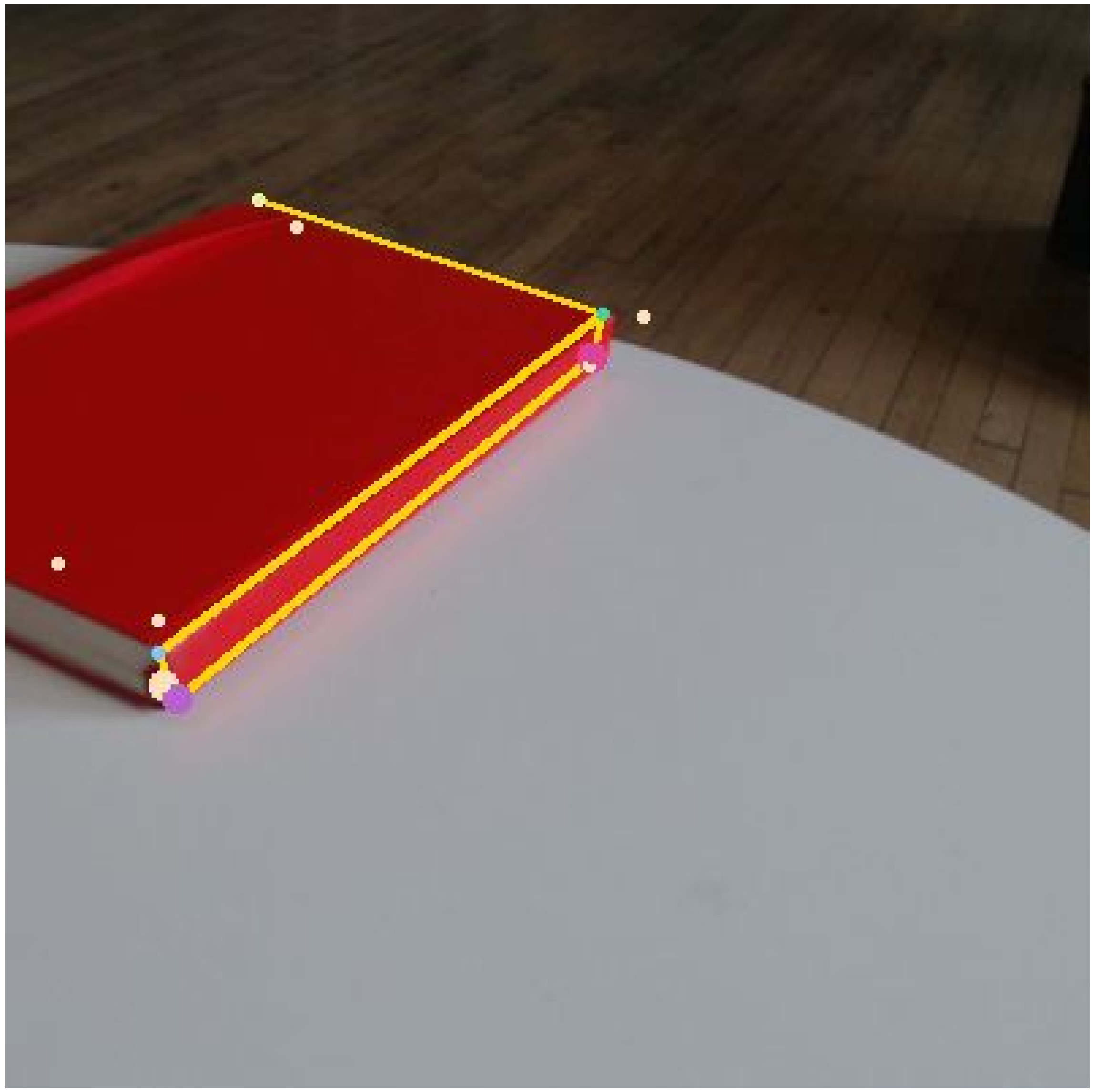
SIM2REAL



- Now it works..

SIM2REAL

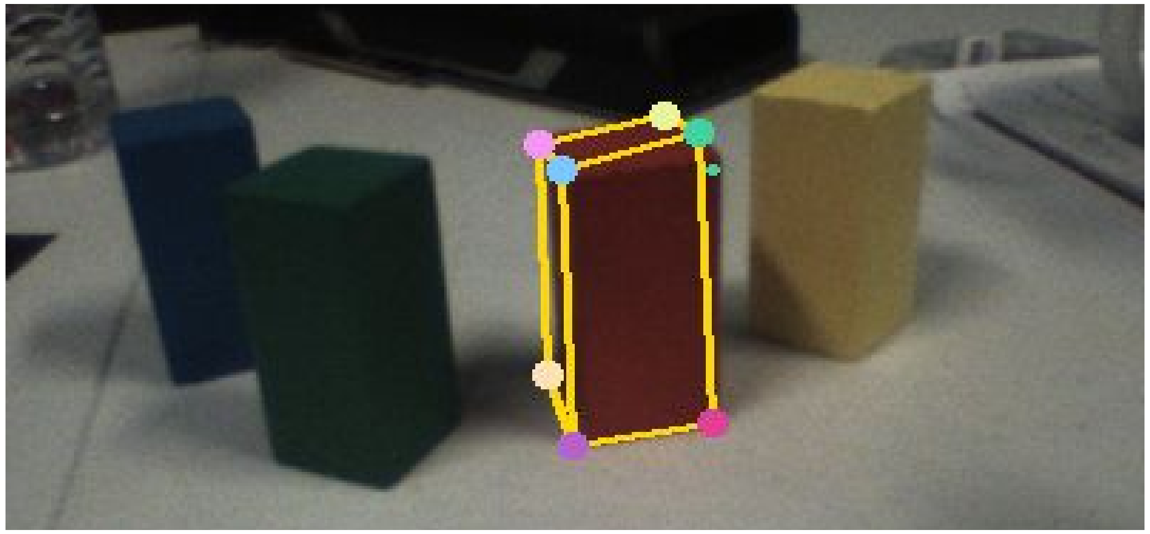
Surprising Result

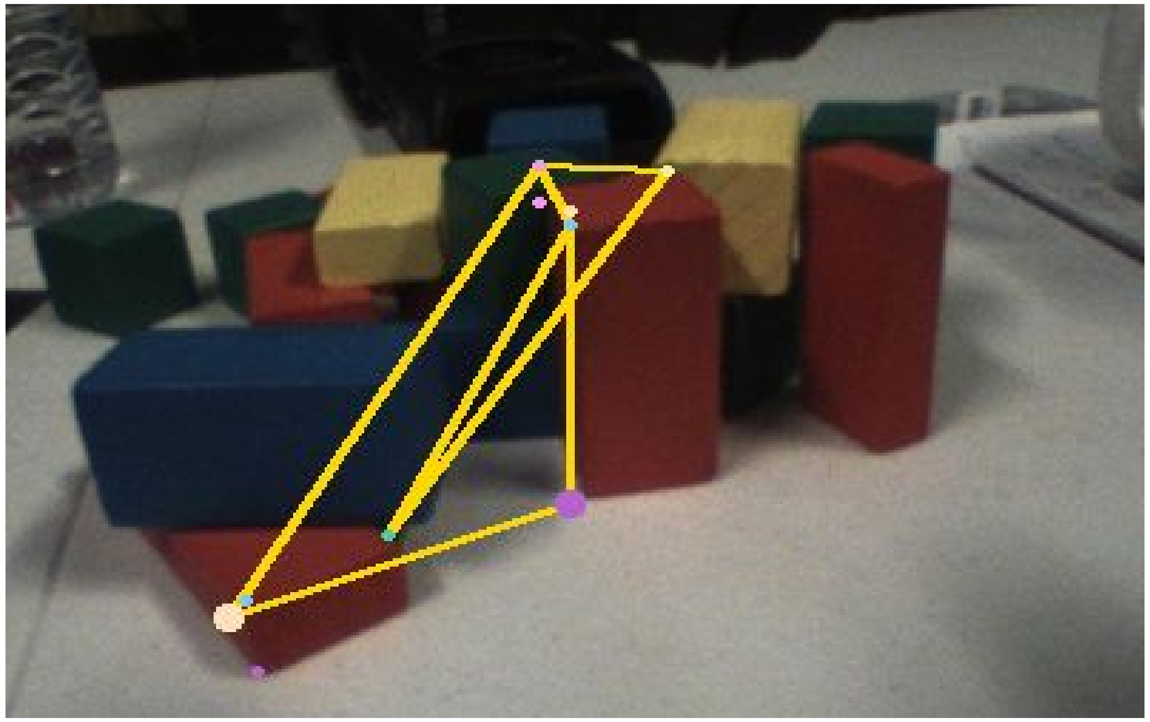


- Even for non cube objects sometimes

SIM2REAL

Baxter’s camera





- It can fail under clutter.
- Solution: use an architecture from computer vision research: combine object detection with vertex heatmap prediction, do not predict vertex heatmaps with the whole image as input

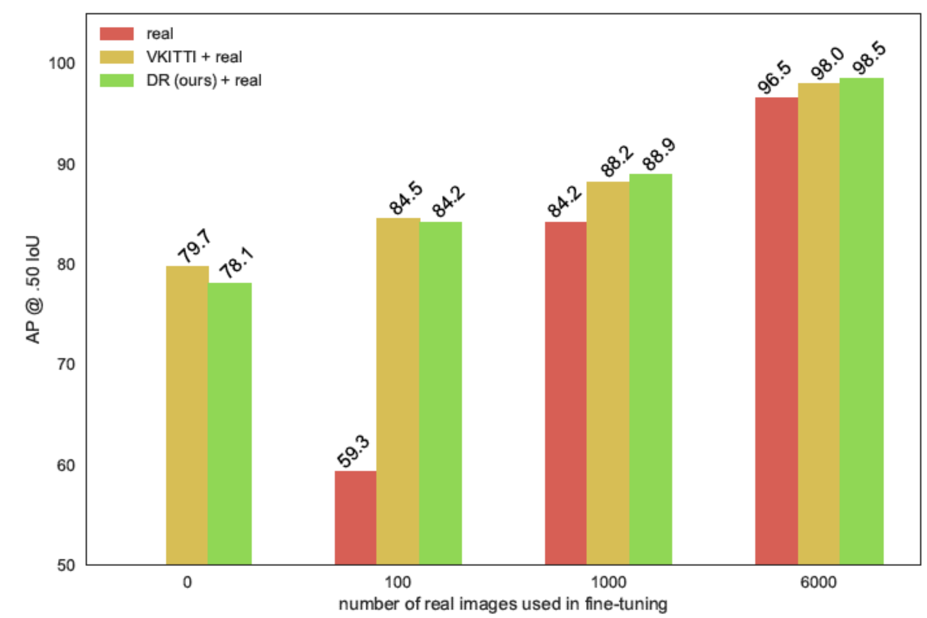
Car detection

VKITTI (Virtual KITTI): a carefully designed simulation dataset to mimic real driving conditions (large engineering effort)

DR: an automatically created simulation dataset with non-realistic visuals and content (small engineering effort)

VKITTI

DR



The fewer the real labelled data, the larger the gain from synthetic data

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

Solving Rubik's Cube with a Robot Hand

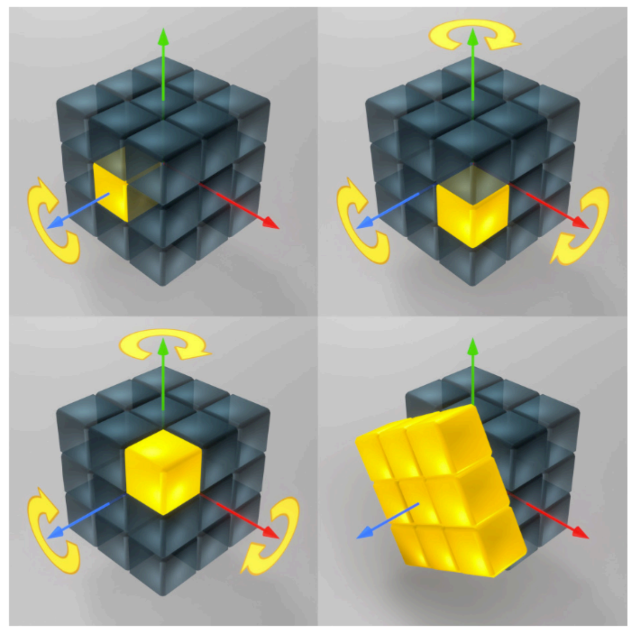
A Preprint

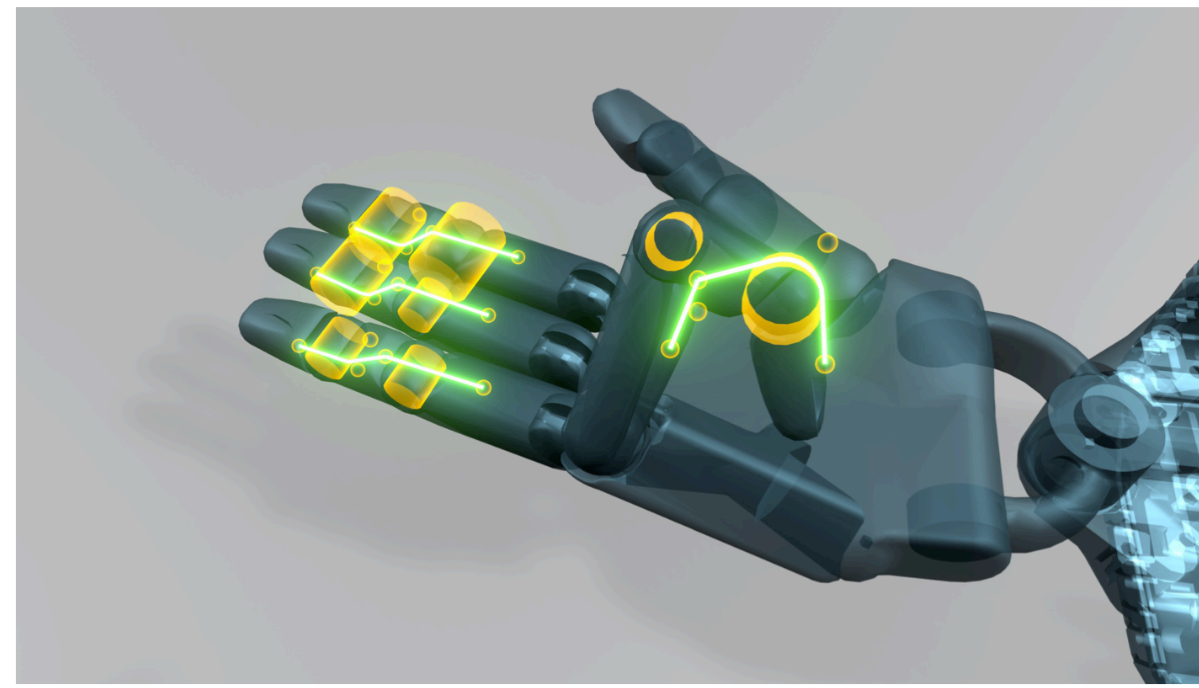
OpenAI
Ilge Akkaya,* Marcin Andrychowicz,* Maciek Chociej,* Mateusz Litwin,* Bob McGrew,* Arthur Petron,*
Alex Paino,* Matthias Plappert,* Glenn Powell,* Raphael Ribas,* Jonas Schneider,* Nikolas Tezak,*
Jerry Tworek,* Peter Welinder,* Lilian Weng,* Qiming Yuan,* Wojciech Zaremba,* Lei Zhang*

Main ideas:

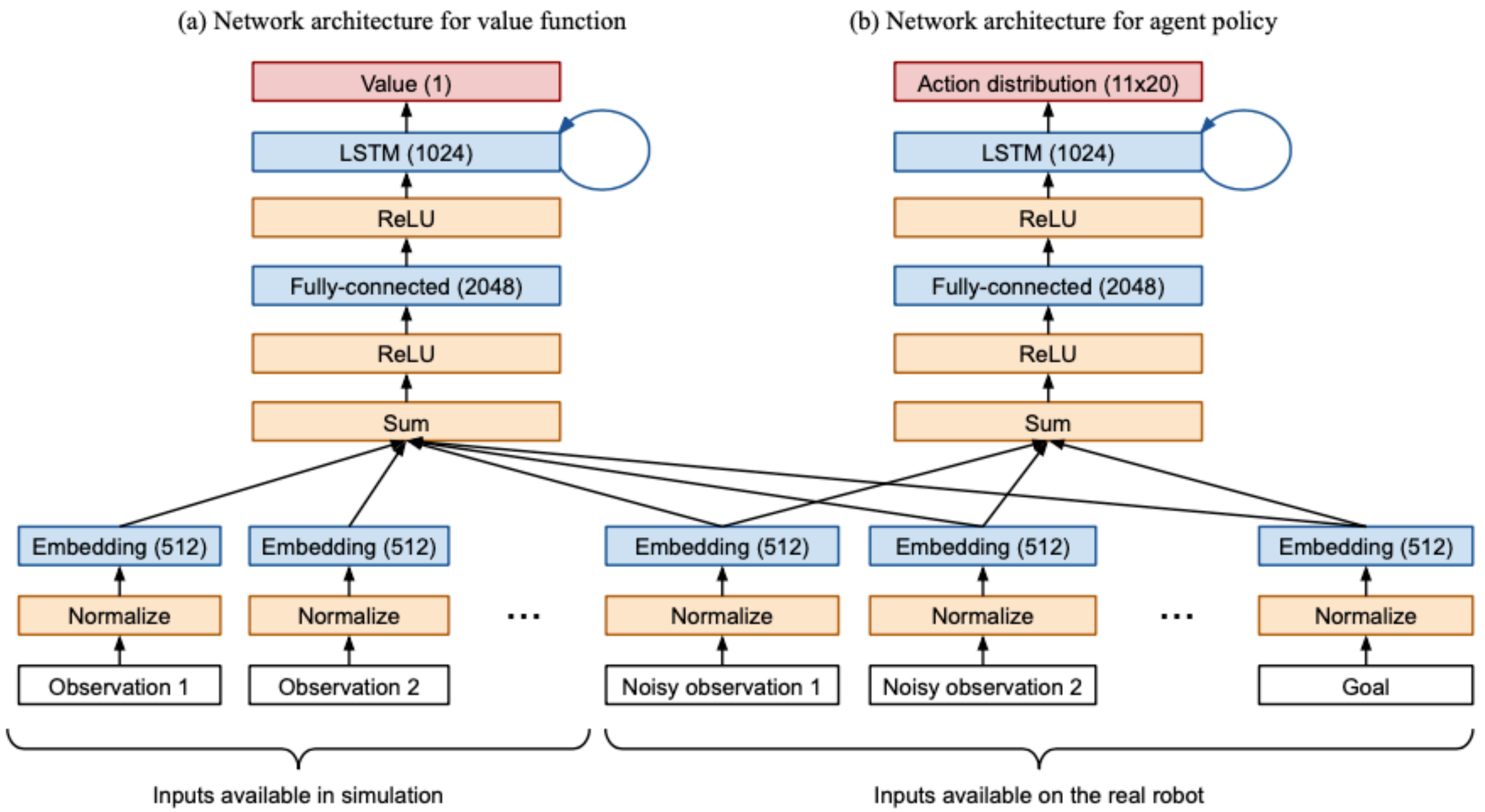
- Trained solely in simulation
- Automatic domain randomization for training:
  - Control Policies
  - State estimators from images
- LSTM policy as opposed to feedforward net

Models for the cube and the hand in Mujoco





Policy with memory



Discrete actions: 11 bins per each of 20 actuated joints

Rewards:

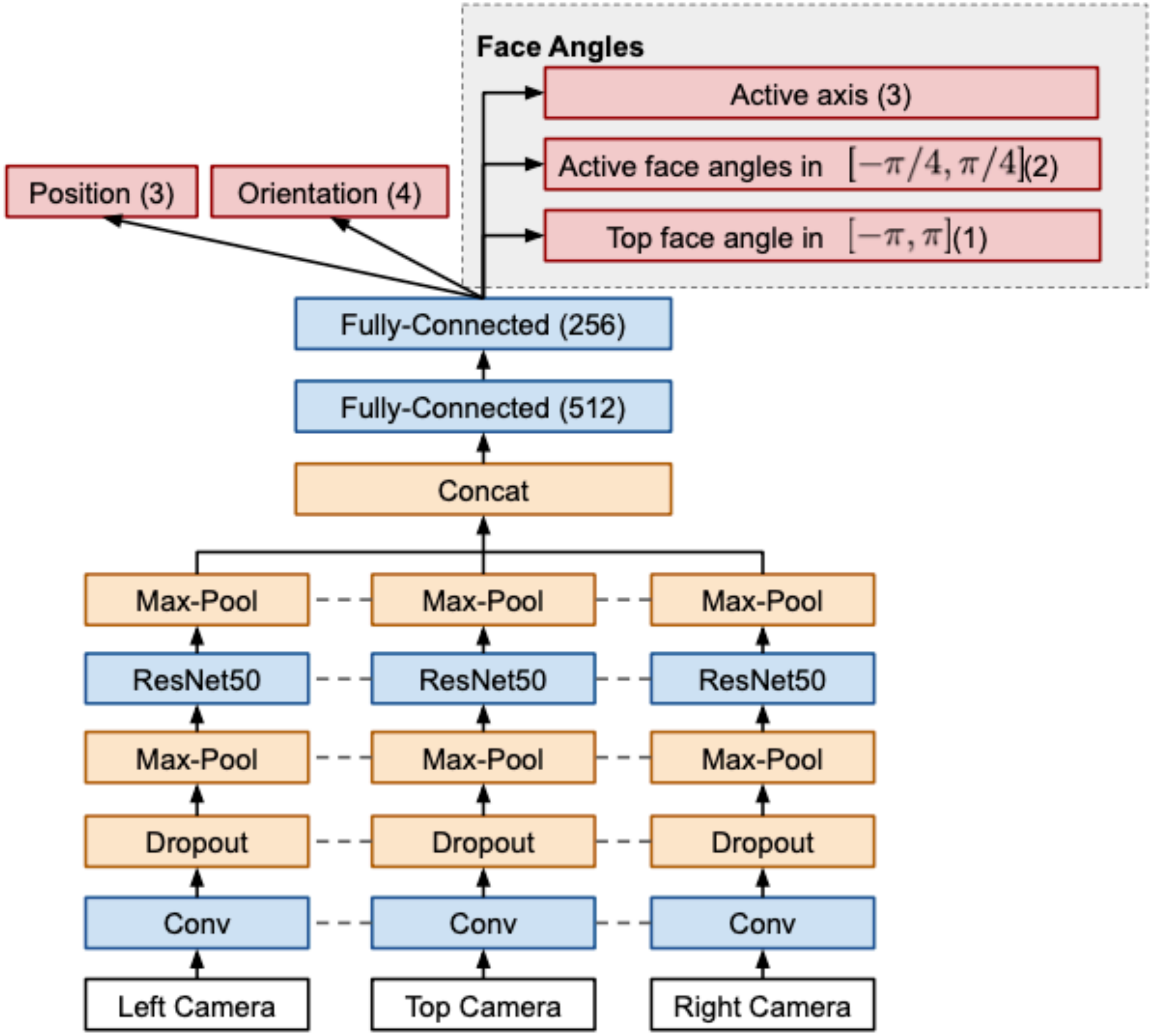
- The difference between the previous and the current distance of the system state from the goal state
- an additional reward of 5 whenever a goal is achieved
- a penalty of –20 whenever a cube/block is dropped

Trained with PPO

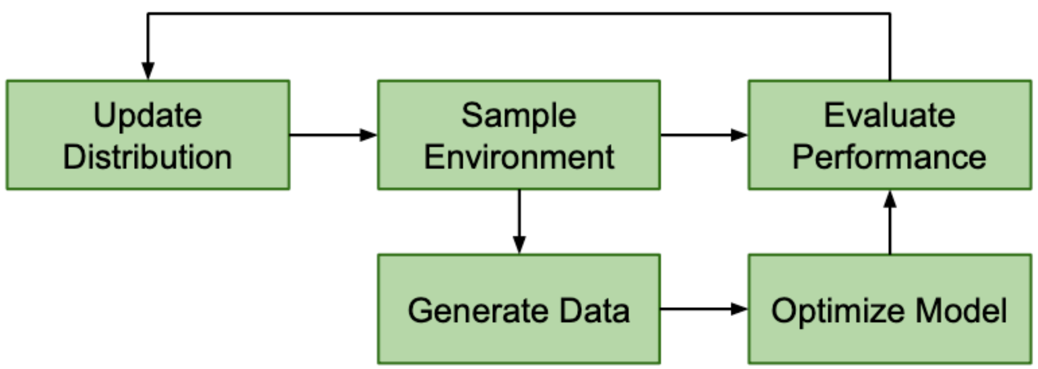
State estimation

(b) An assembled Giiker cube while charging.

GT



Adaptive Domain Randomization



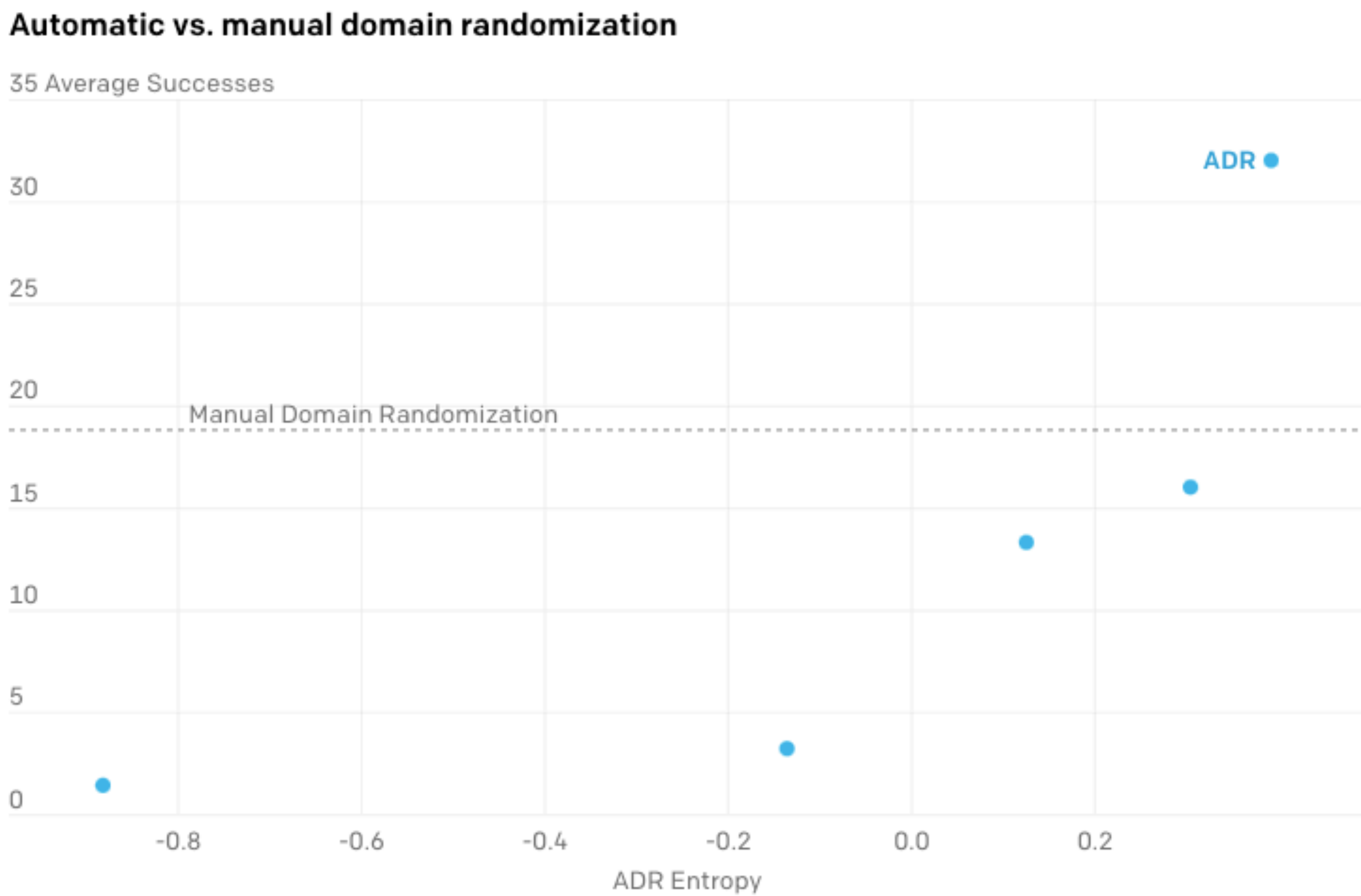
Algorithm 1 ADR

Require: ϕ^0 ▷ Initial parameter values
Require: $\{D_i^L, D_i^H\}_{i=1}^d$ ▷ Performance data buffers
Require: m, t_L, t_H, where $t_L < t_H$ ▷ Thresholds
Require: Δ ▷ Update step size

```
φ ← φ⁰
repeat
    λ ~ P_φ
    i ~ U{1, ..., d}, x ~ U(0, 1)
    if x < 0.5 then
        D_i ← D_i^L, λ_i ← φ_i^L            ▷ Select the lower bound in "boundary sampling"
    else
        D_i ← D_i^H, λ_i ← φ_i^H            ▷ Select the higher bound in "boundary sampling"
    end if
    p ← EVALUATEPERFORMANCE(λ)              ▷ Collect model performance on environment parameterized by λ
    D_i ← D_i ∪ {p}                         ▷ Add performance to buffer for λ_i, which was boundary sampled
    if LENGTH(D_i) ≥ m then
        p̄ ← AVERAGE(D_i)
        CLEAR(D_i)
        if p̄ ≥ t_H then
            φ_i ← φ_i + Δ
        else if p̄ ≤ t_L then
            φ_i ← φ_i − Δ
        end if
    end if
until training is complete
```

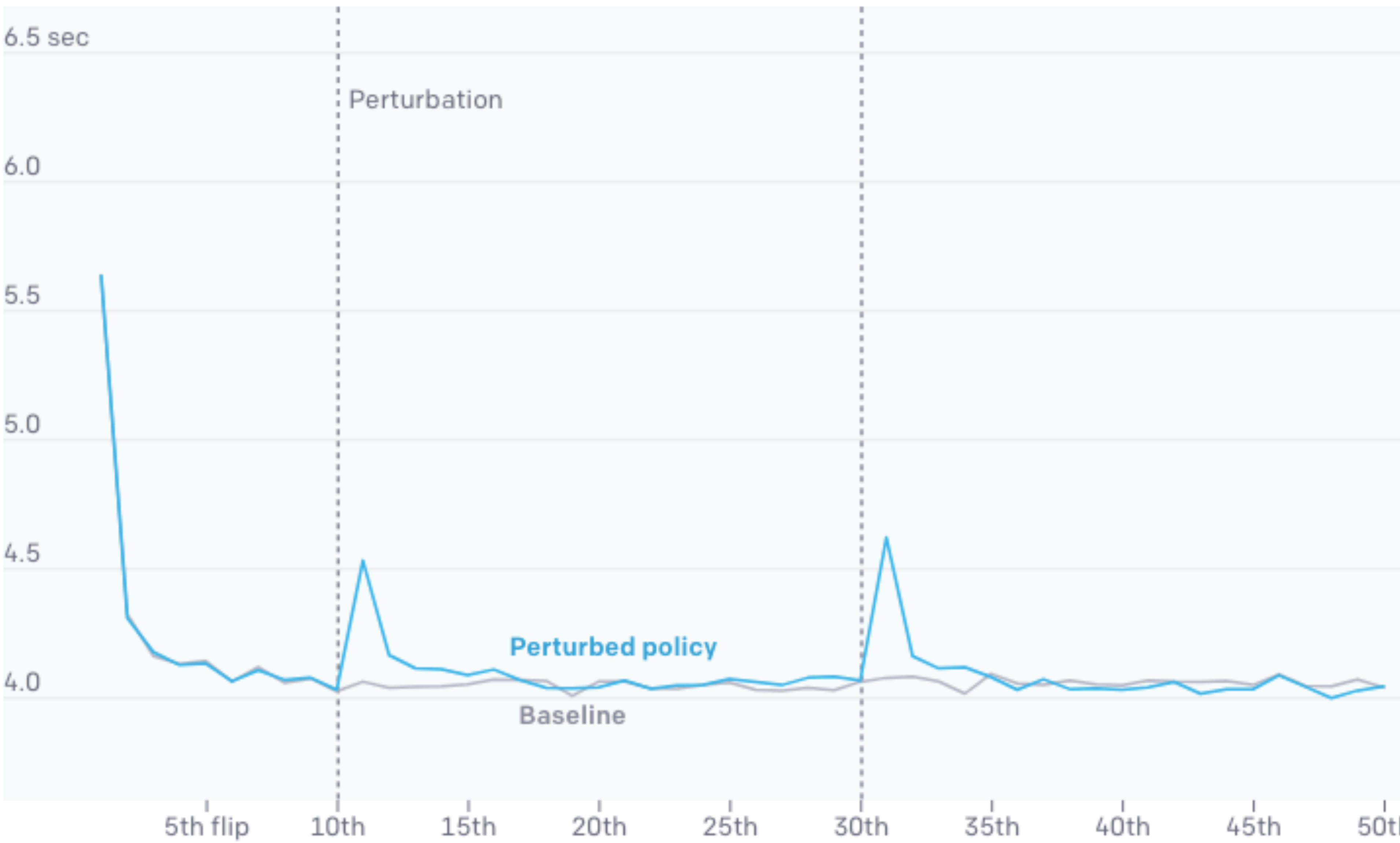
DR for state estimation

Automatic Domain Randomization (ADR)

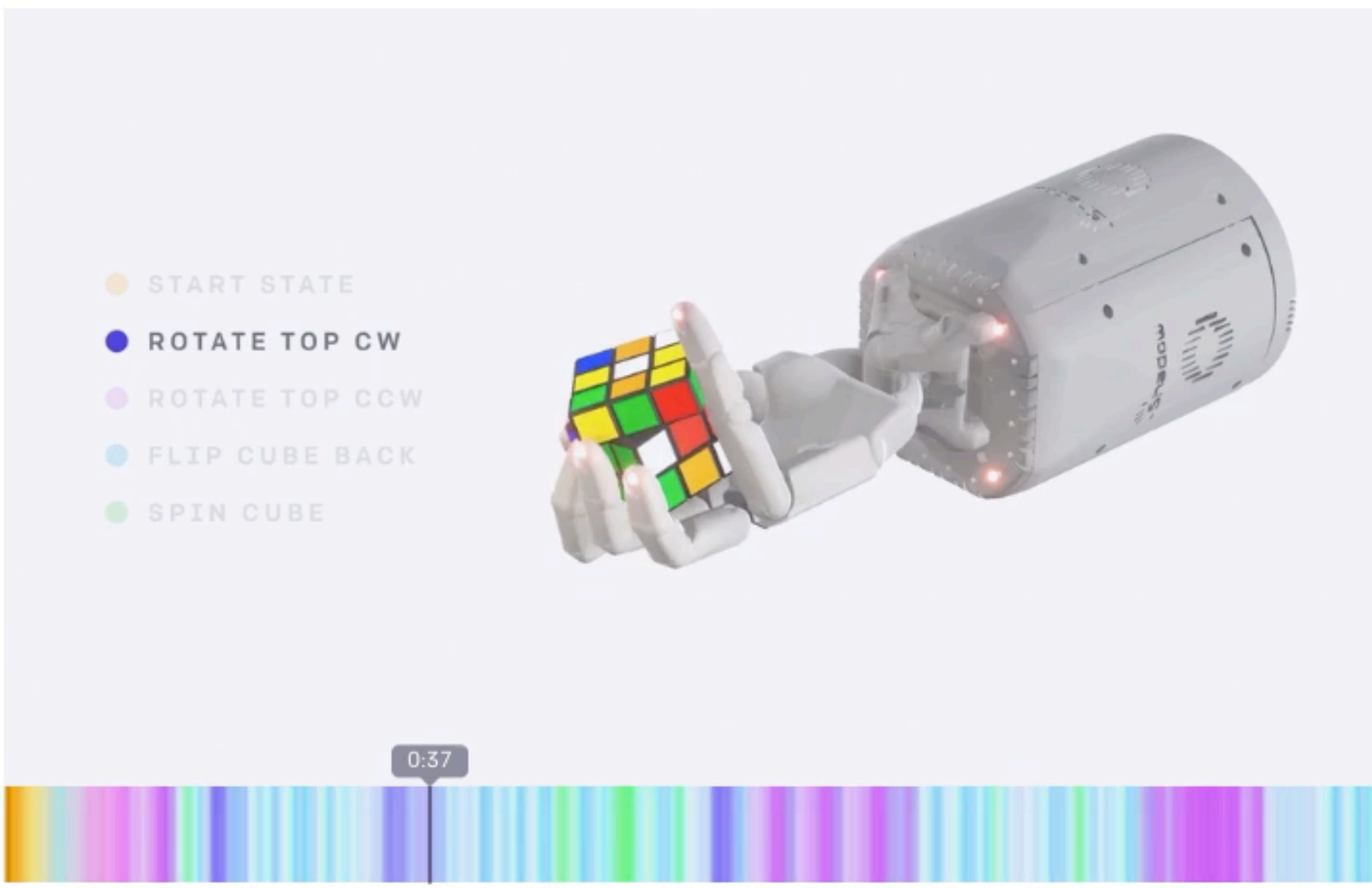


https://openai.com/blog/solving-rubiks-cube/

Time to success when the network's memory is erased



The LSTM state is interpretable



Driving Policy Transfer via Modularity and Abstraction

Matthias Müller
Visual Computing Center
KAUST, Saudi Arabia

Alexey Dosovitskiy
Intelligent Systems Lab
Intel Labs, Germany

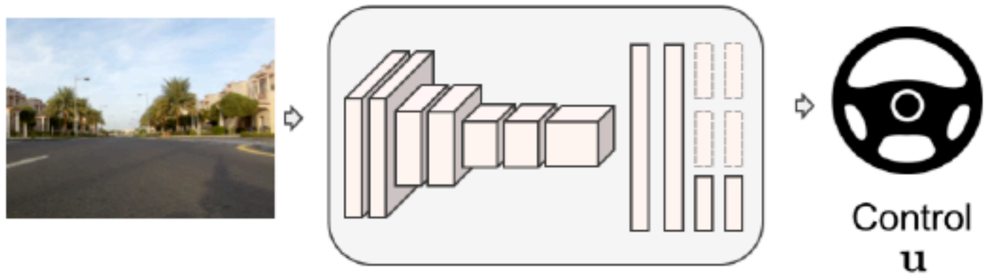
Bernard Ghanem
Visual Computing Center
KAUST, Saudi Arabia

Vladlen Koltun
Intelligent Systems Lab
Intel Labs, USA

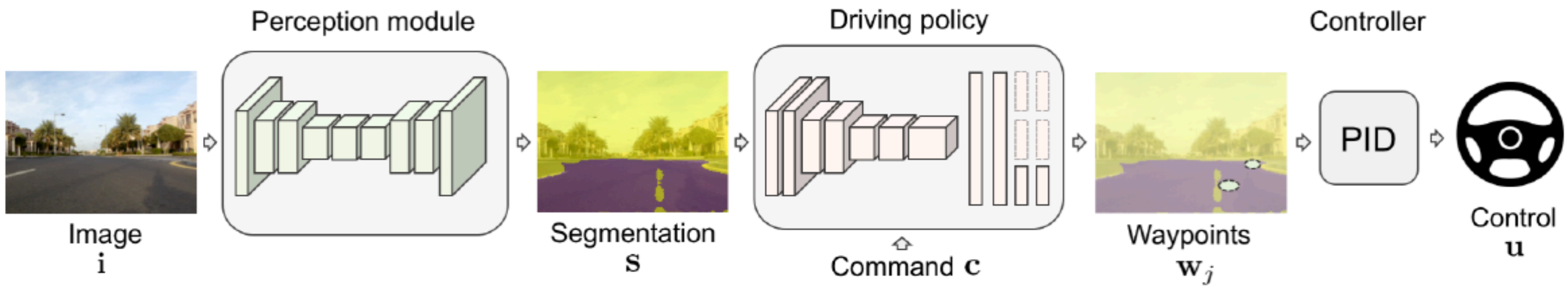
Idea: the driving policy is not directly exposed to raw perceptual input or low-level vehicle dynamics.

Main idea

Pixels to steering wheel mapping is not SIM2REAL transferable: image textures and car dynamics mismatch



Instead: label maps to waypoint mapping is better SIM2REAL transferable: label maps and waypoints are similar across SIM and REAL. A low-level controller will take the car from waypoint to waypoint in the real world



Results: Train/Test

We train policies via behaviour cloning (standard regression loss) in Town1/ Weather1 dataset, and evaluate them on all four.

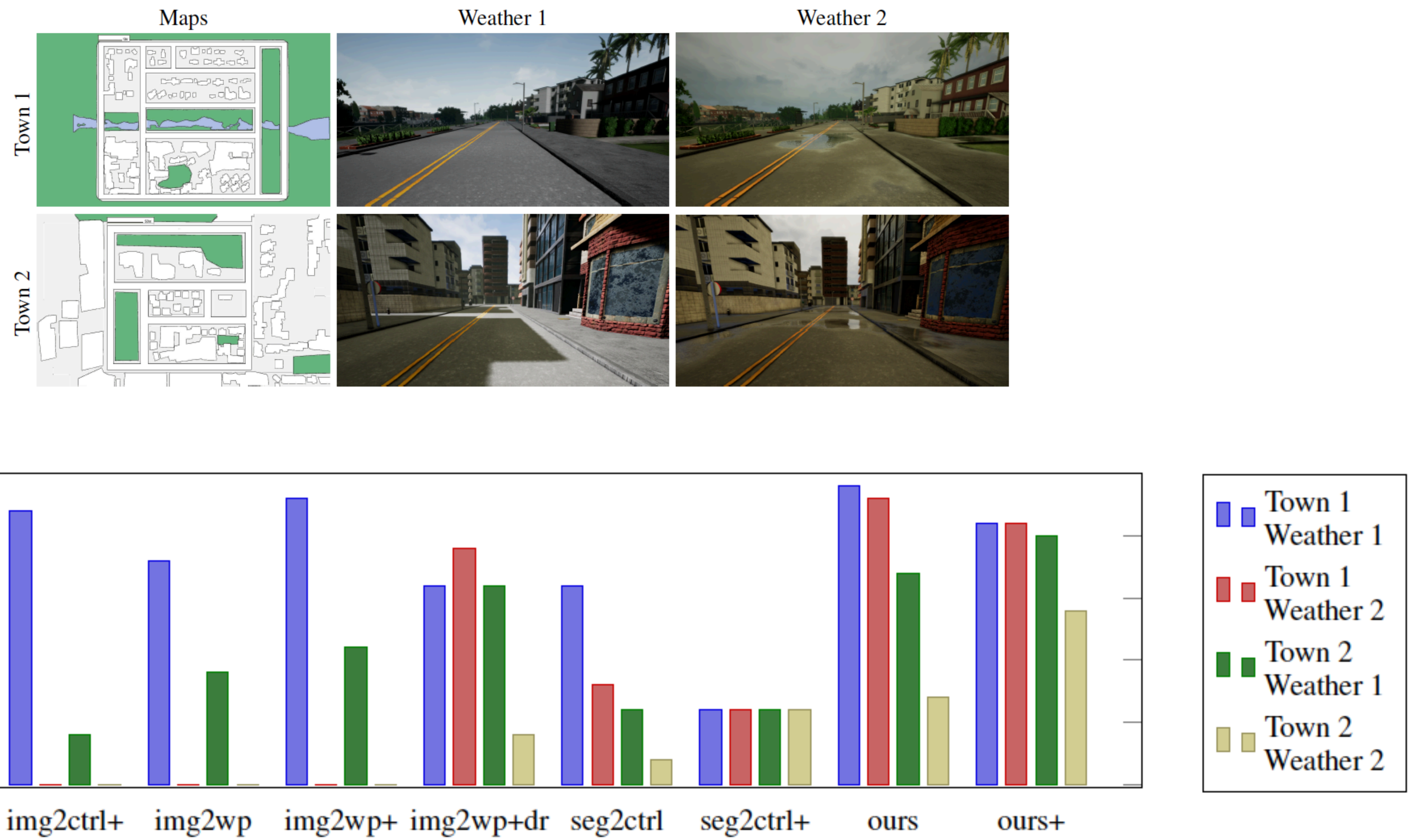


Figure 4: Quantitative evaluation of goal-directed navigation in simulation. We report the success rate over 25 navigation trials in four town-weather combinations. The models have been trained in Town 1 and Weather 1. The evaluated models are: *img2ctrl* – predicting low-level control from color images; *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.

TossingBot: Learning to Throw Arbitrary Objects with Residual Physics

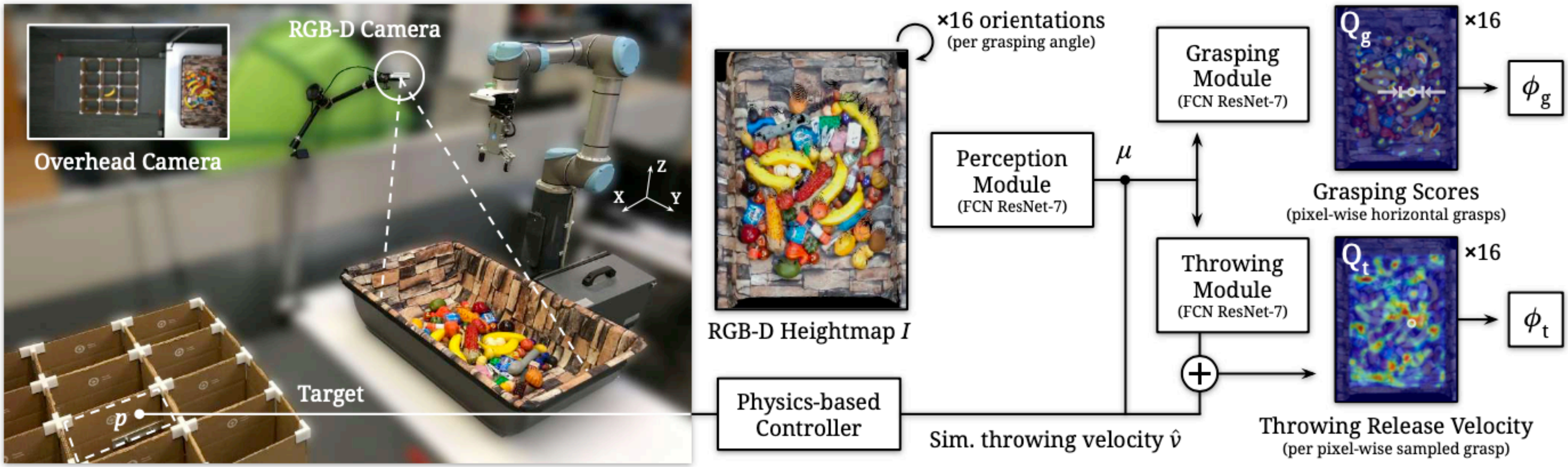
Andy Zeng[1,2], Shuran Song[1,2,3], Johnny Lee[2], Alberto Rodriguez[4], Thomas Funkhouser[1,2]
[1]Princeton University [2]Google [3]Columbia University [4]Massachusetts Institute of Technology
http://tossingbot.cs.princeton.edu

Tossing bot (this work)

Pik-n-place bot (slow)

Tossing: learning to grasp and throw objects into selected boxes outside the robot's natural range

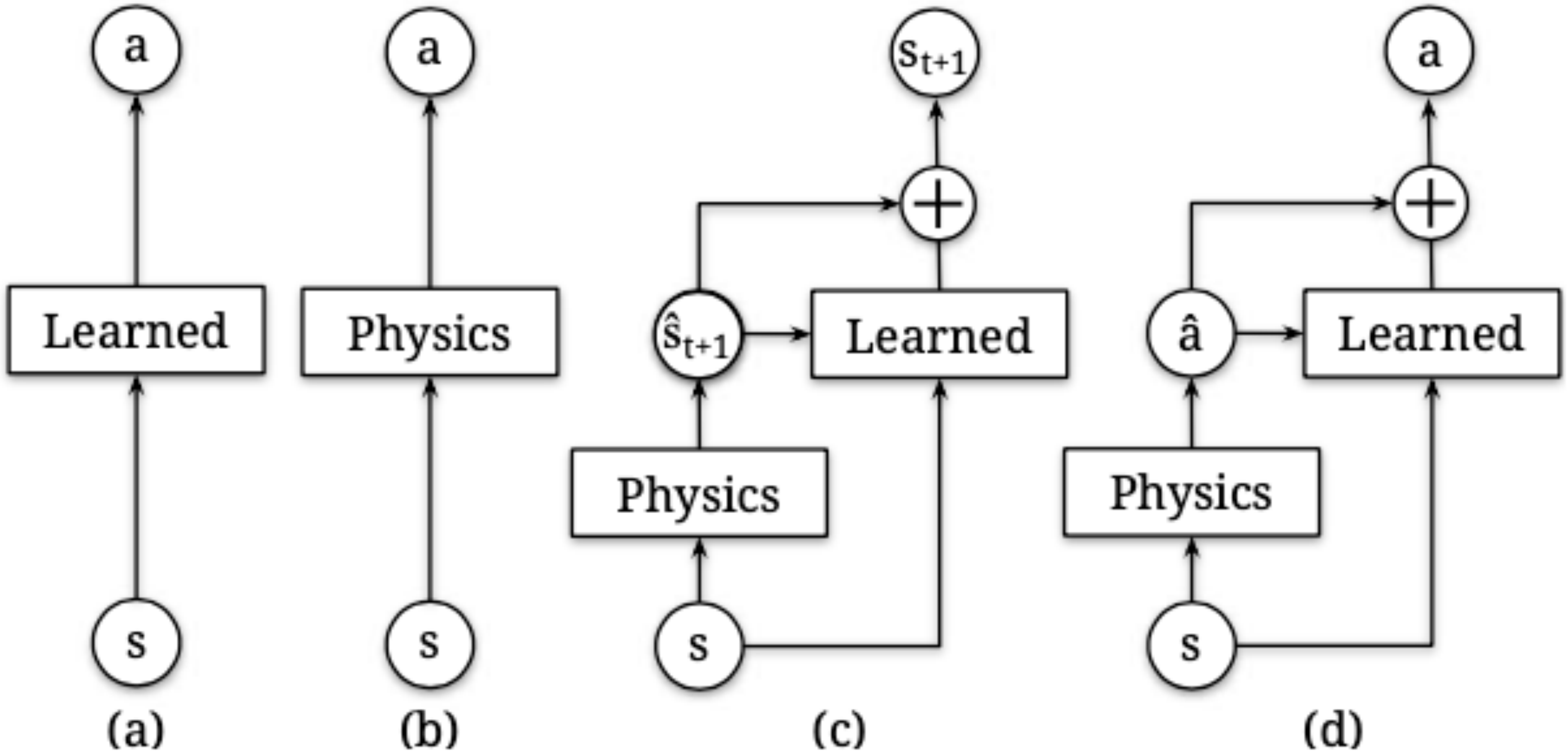
Tossing Bot



Jointly learns grasping and throwing by mapping visual observations (RGB-D images) to control parameters for motion primitives

- **Grasping primitive parameters**: 3D location of a top-down parallel jaw grasp (IK are used to execute the grasp). The output of the grasping net represents pixel wise grasping success. Rotate the input by 16 angles and output 16 such pixel wise probability maps, to allow any oriented planar grasp.
- **Throwing primitive parameters**: the release 3D position and velocity of an object leaving the robot hand. The throwing primitive takes as input parameters $\phi_t = (r,v)$ and executes an end effector trajectory such that the mid-point between the gripper fingertips reaches a desired release position $r = (r_x,r_y,r_z)$ and velocity $v = (v_x,v_y,v_z)$, at which point the gripper opens and releases the object.

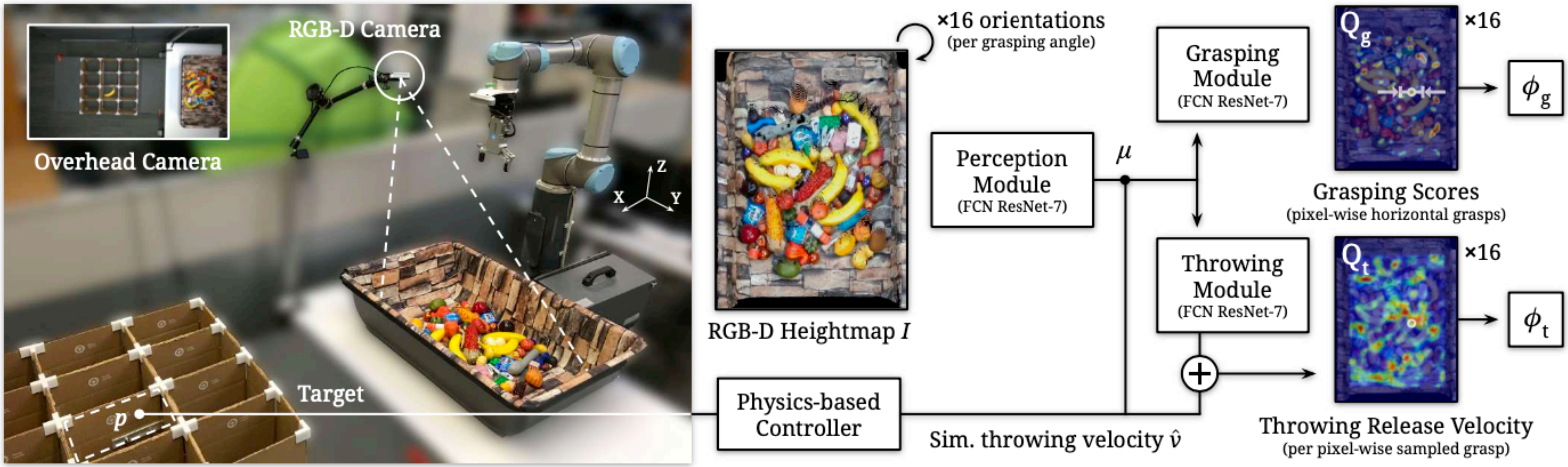
Residual Physics



A. Learned state-to-action mapping
B. Infer action per state with analytic physics models
C. Residual physics for predicting next state
D. Residual physics for predicting action (parameters for controller)—> this work

Use projectile ballistics to provide an estimate for the release object velocity that is needed to get an object to land at a target location

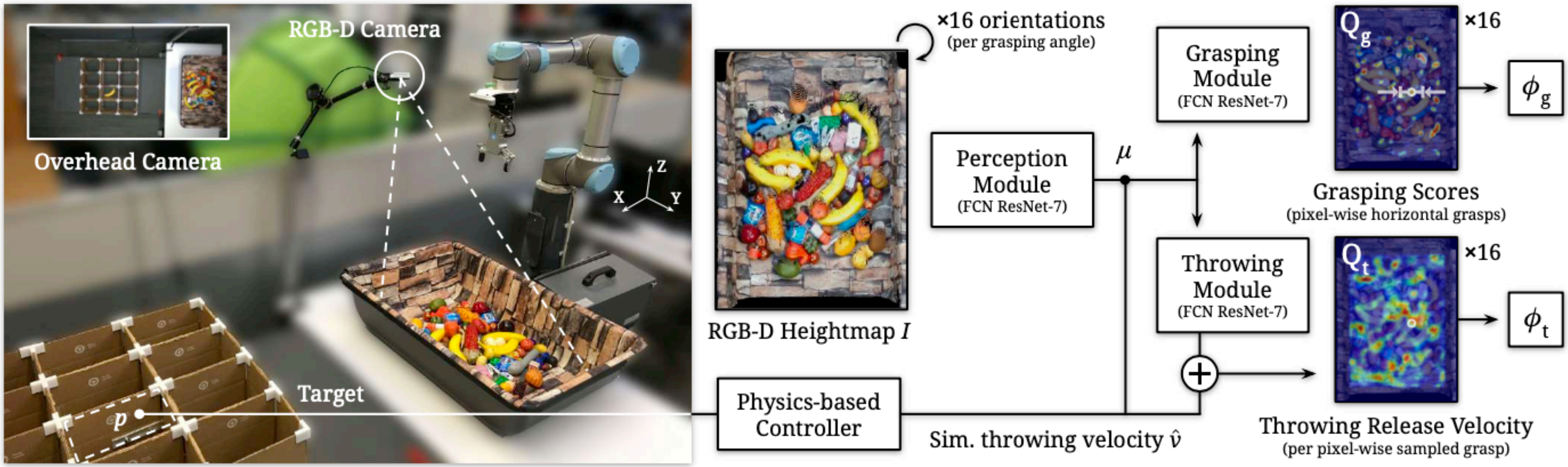
Tossing Bot



Jointly learns grasping and throwing by mapping visual observations (RGB-D images) to control parameters for motion primitives

- **Grasping primitive parameters**: 3D location of a top-down parallel jaw grasp (IK are used to execute the grasp). The output of the grasping net represents pixel wise grasping success. Rotate the input by 16 angles and output 16 such pixel wise probability maps, to allow any oriented planar grasp.
- **Throwing primitive parameters**: the release 3D position and velocity of an object leaving the robot hand. The throwing primitive takes as input parameters $\phi_t = (r,v)$ and executes an end effector trajectory such that the mid-point between the gripper fingertips reaches a desired release position $r = (r_x,r_y,r_z)$ and velocity $v = (v_x,v_y,v_z)$, at which point the gripper opens and releases the object.

Training by trial-and-error



The system predicts grasp and throw parameters.

Records grasp success and actual landing location.

Trains the grasping net as a pixel-wise classification and the throwing net as a location-conditioned regression for the residual velocity.

Semantics emerge

Results

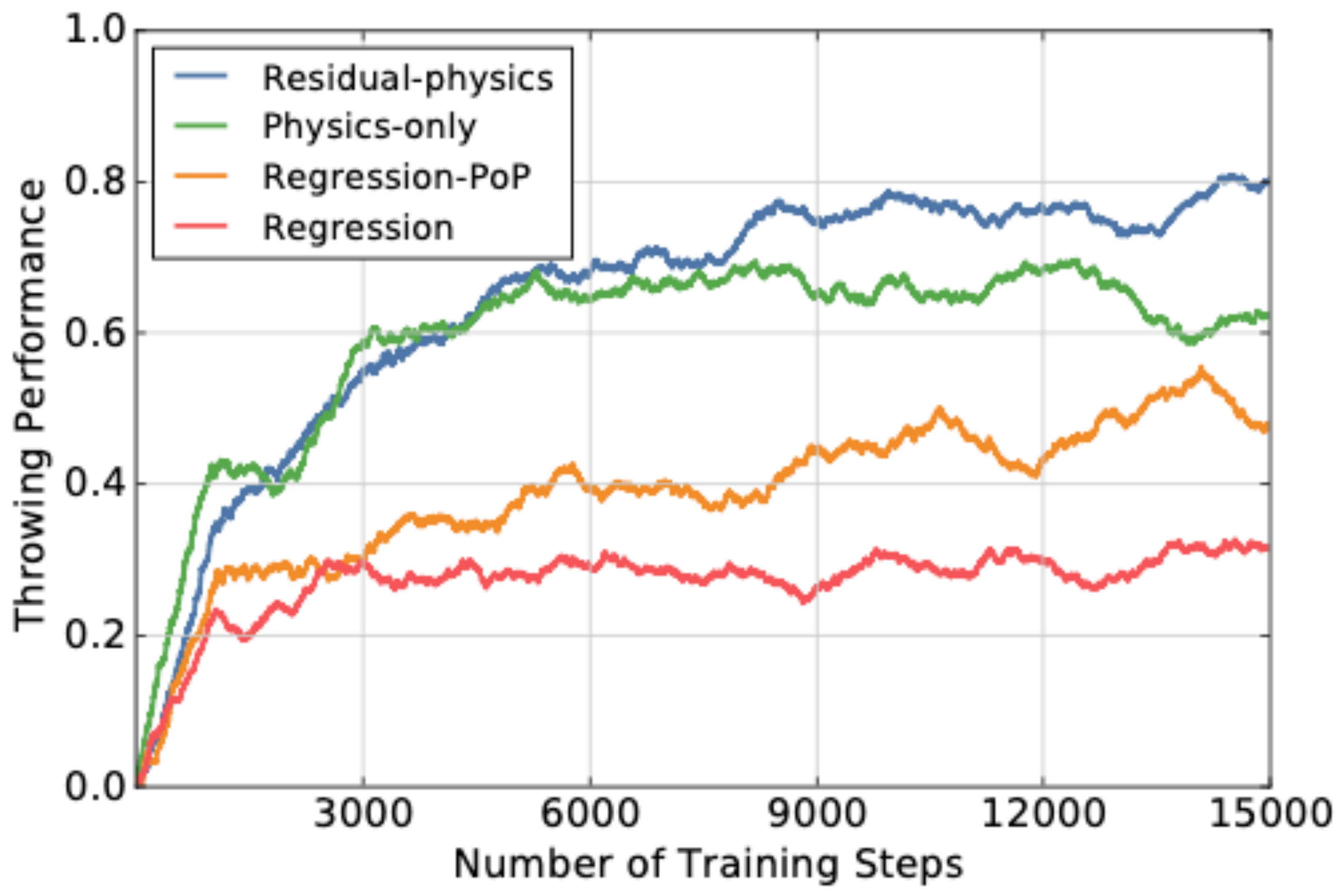
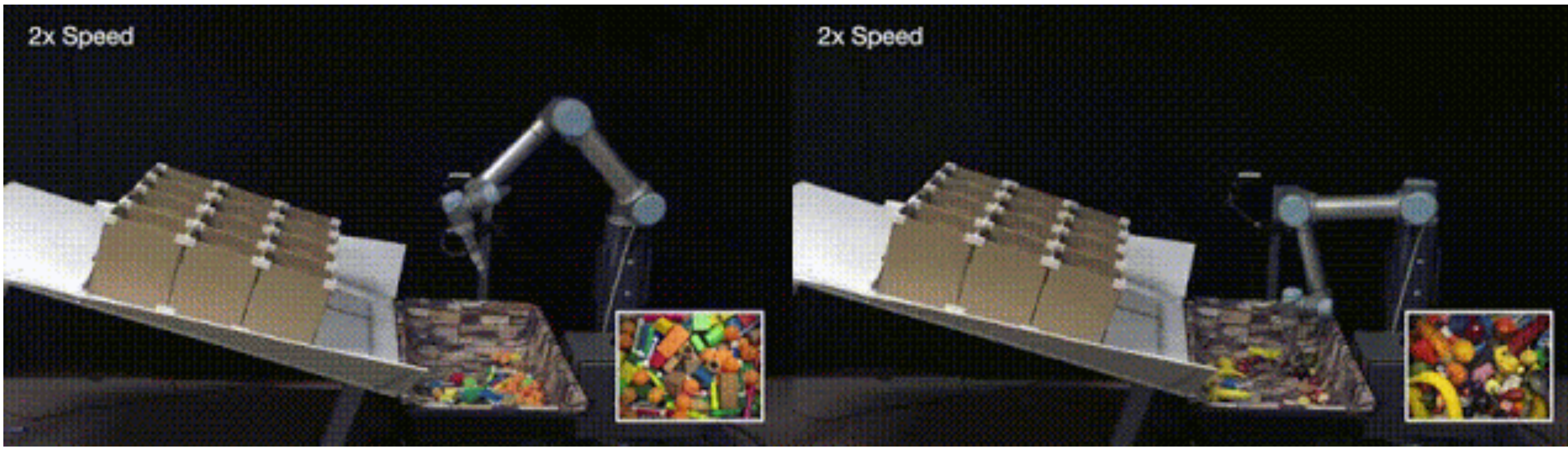


Fig. 7. Our method (Residual-physics) outperforms baseline alternatives in terms of throwing success rates in simulation on the Hammers object set.

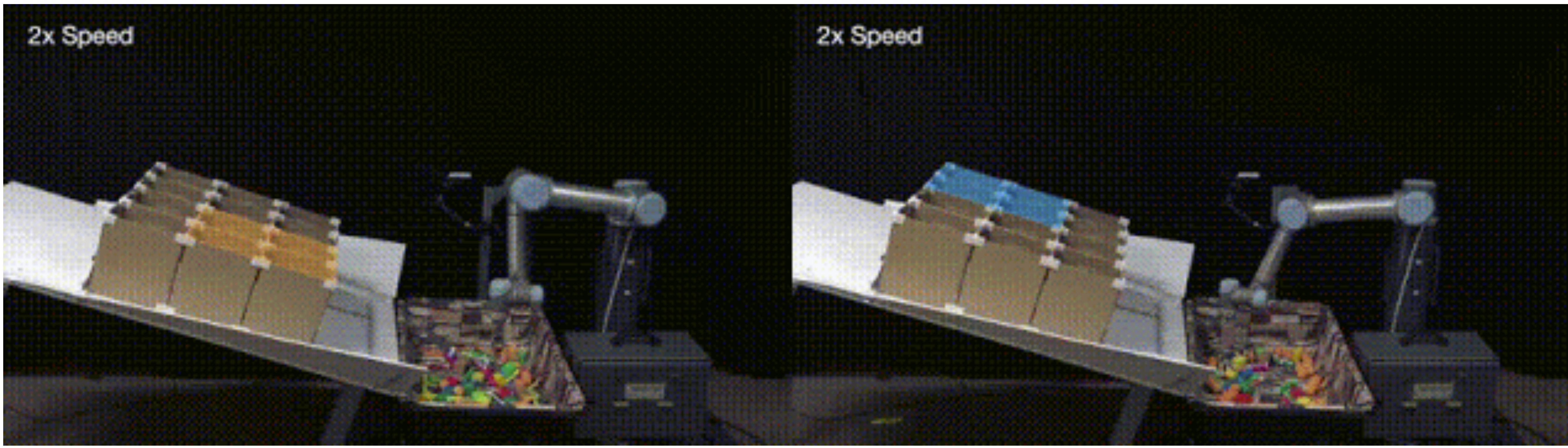
Generalization/Adaptation to novel objects



On new objects, TossingBot starts out with lower performance, but quickly adapts within a few hundred training steps (i.e., an hour or two) to achieve similar performance as with training objects.

What makes learning so sample efficient?

Generalization/Adaptation to novel target locations

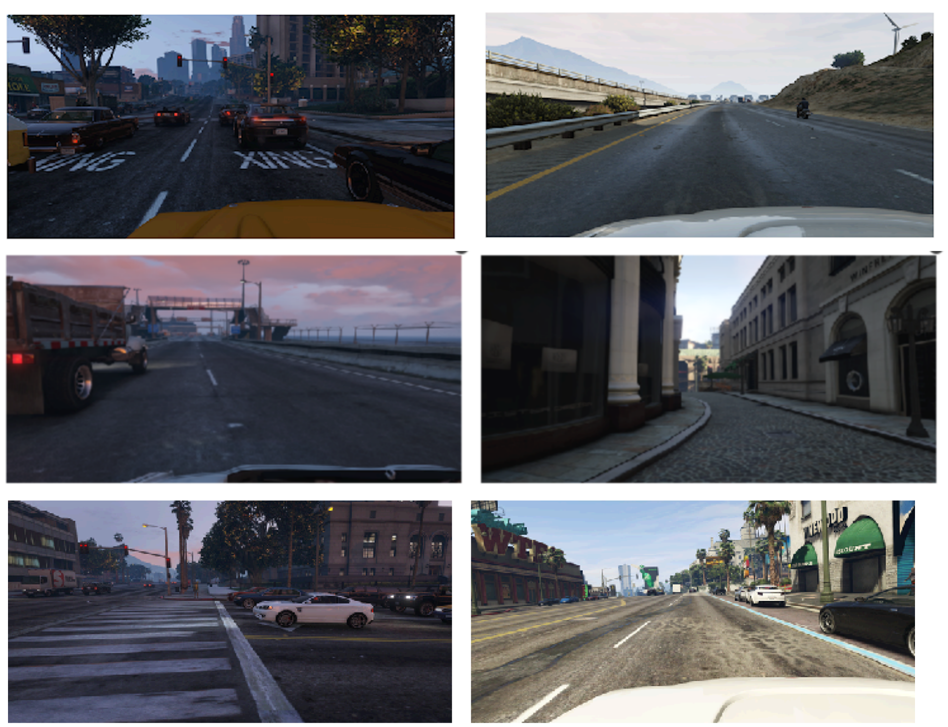


- Residual Physics helps generalization.
- The initial estimates of throwing velocities from projectile ballistics easily generalize to new target locations, while the residuals help make adjustments on top of those estimates to compensate for varying object properties in the real world.
- Deep learning without physics can only handle target locations seen during training.

Domain adaptation for visual observations

GTA: synthetic data of urban scenes from a camera mounted on a car

Cityscapes: real data of urban scenes from a camera mounted on a car



source

target

19 object classes to be detected: people, cars, stop signs, poles, etc.

Our goal: Train detectors and pixel labelers on GTA that generalize to Cityscapes

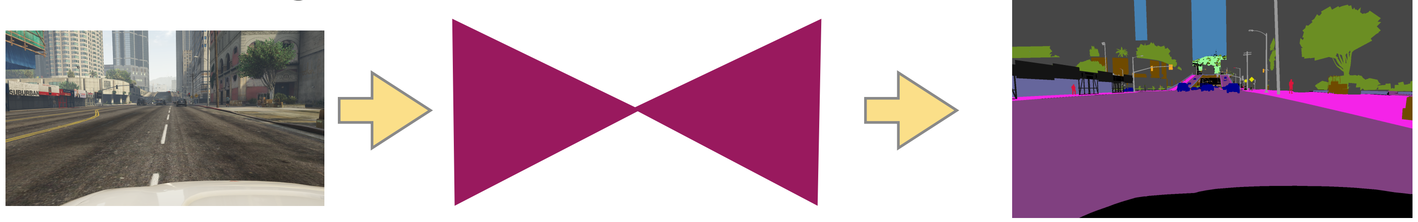
Baseline

Train a classifier on source and test it on the target, and hope it generalizes

1. Pick a network architecture, e.g. ResNet101 or VGG
2. Download a pretained neural network, e.g., trained for image classification on Imagenet dataset
3. Finetune it on the source domain (GTA)
4. Apply it on the target domain (Cityscapes)

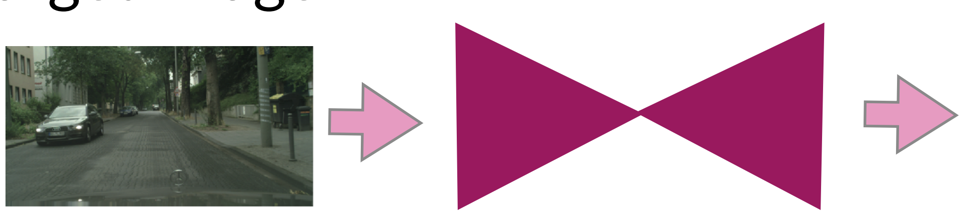
Training

source image **Labeller NN** label mask



Testing

target image **Labeller NN**



?

Image classification in Imagenet

Baseline

Train a classifier on source and test it on the target, and hope it generalizes

1. Download a pretained neural network, e.g., trained for image classification on Imagenet dataset
2. Finetune it on the source domain (GTA)
3. Apply it on the target domain (Cityscapes)

Pretrain on Imagenet -> finetune in GTA->test in GTA: 53% meanIoU
Pretrain on Imagenet -> finetune in GTA->test in Cityscapes: 28% meanIoU

Pretrain on PASCAL -> finetune in GTA->test in GTA: 58.84% meanIoU
Pretrain on PASCAL -> finetune in GTA->test in Cityscapes: 32% meanIoU

Pretrain on PASCAL -> cotrain in GTA/PASCAL->test in Cityscapes: 39% meanIoU

Baseline

Train a classifier on source and test it on the target, and hope it generalizes

1. Download a pretained neural network, e.g., trained for image classification on

Catastrophic forgetting:

- During fine-tuning, the network forgets the general and nicely transferable PASCAL features!
- Finetuning a neural net on a very limited domain is a bad idea for transfer

Pretrain on PASCAL -> finetune in GTA->test in GTA: 58.84% meanIoU
Pretrain on PASCAL -> finetune in GTA->test in Cityscapes: 22% meanIoU

Other solutions for catastrophic forgetting?

Baseline

Train a classifier on source and test it on the target, and hope it generalizes

1. Download a pretained neural network, e.g., trained for image classification on Imagenet dataset
2. Finetune it on the source domain (GTA):
   1. Adapt only the top layers and keep the earlier frozen.
   2. Cotrain it using both the old task and the new task in the smaller dataset
3. Apply it on the target domain (Cityscapes)

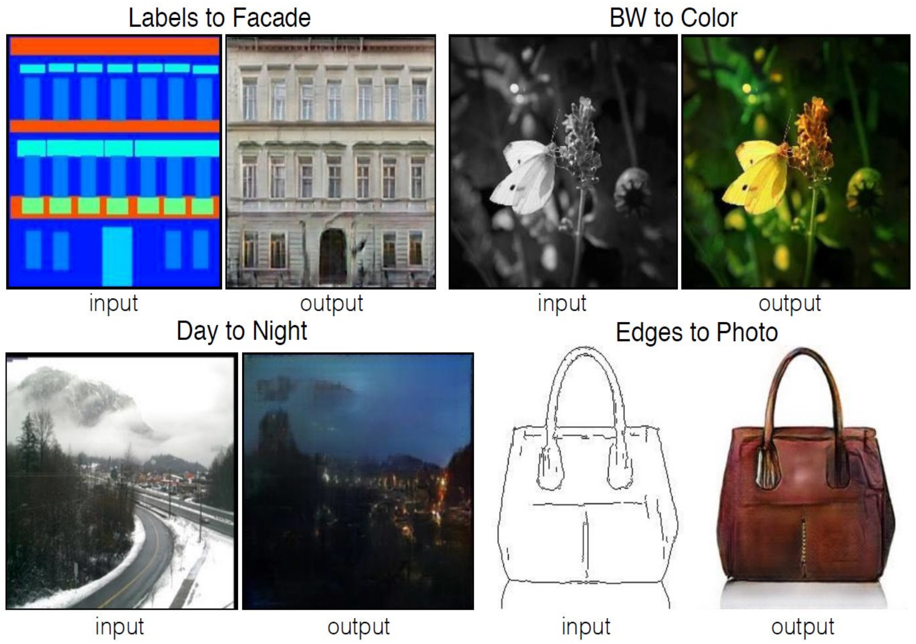
Pretrain on Imagenet -> finetune in GTA->test in GTA: 53% meanIoU
Pretrain on Imagenet -> finetune in GTA->test in Cityscapes: 28% meanIoU

Pretrain on PASCAL -> finetune in GTA->test in GTA: 58.84% meanIoU
Pretrain on PASCAL -> finetune in GTA->test in Cityscapes: 32% meanIoU

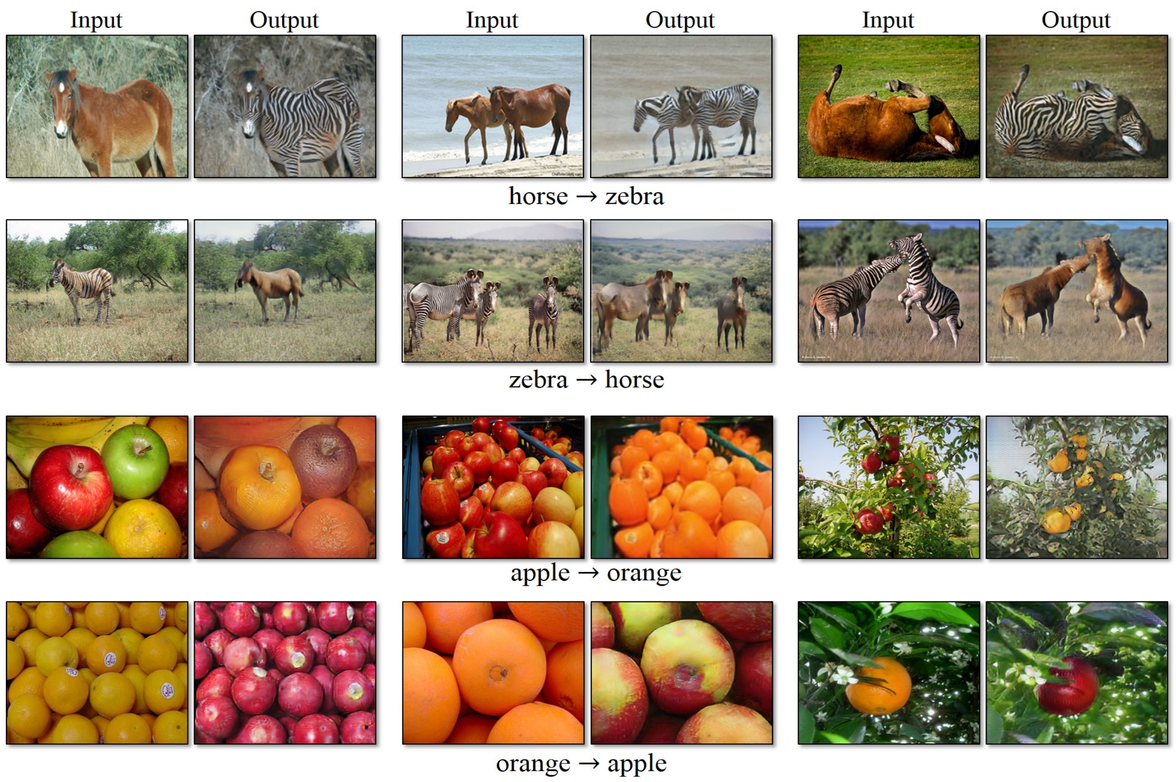
Pretrain on PASCAL -> cotrain in GTA/PASCAL->test in Cityscapes: 39% meanIoU

Learning to translate images across domains (sim and real)

- Paired

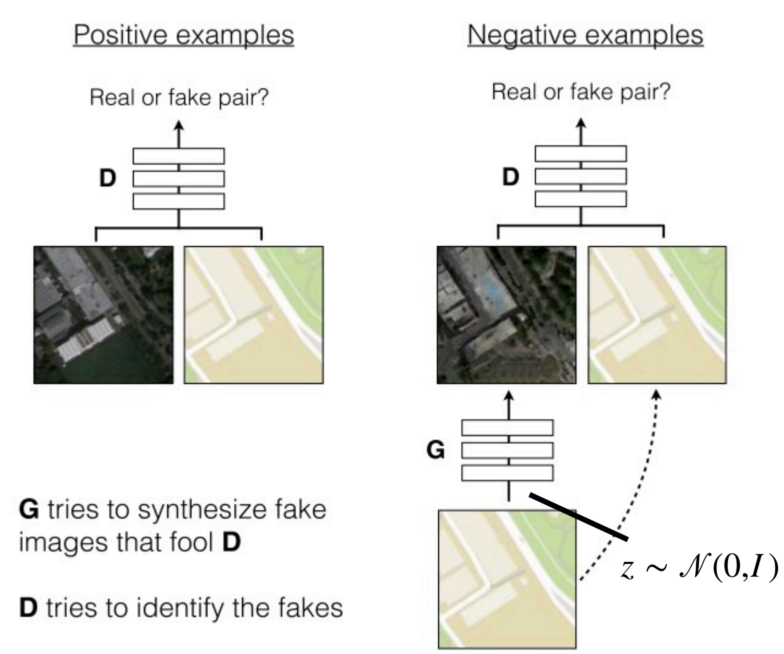


- Unpaired (this is our case)

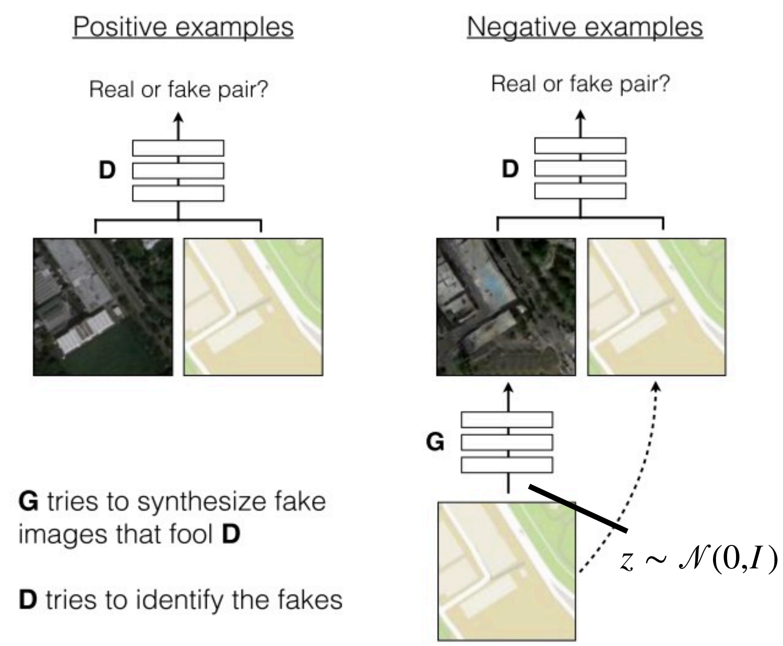


Paired case

- The generator takes the (source) image as input and tries to output the corresponding target image
- Minimize
  - image reconstruction loss
  - adversarial loss: pairs of source-target images as input to discriminator



Paired case

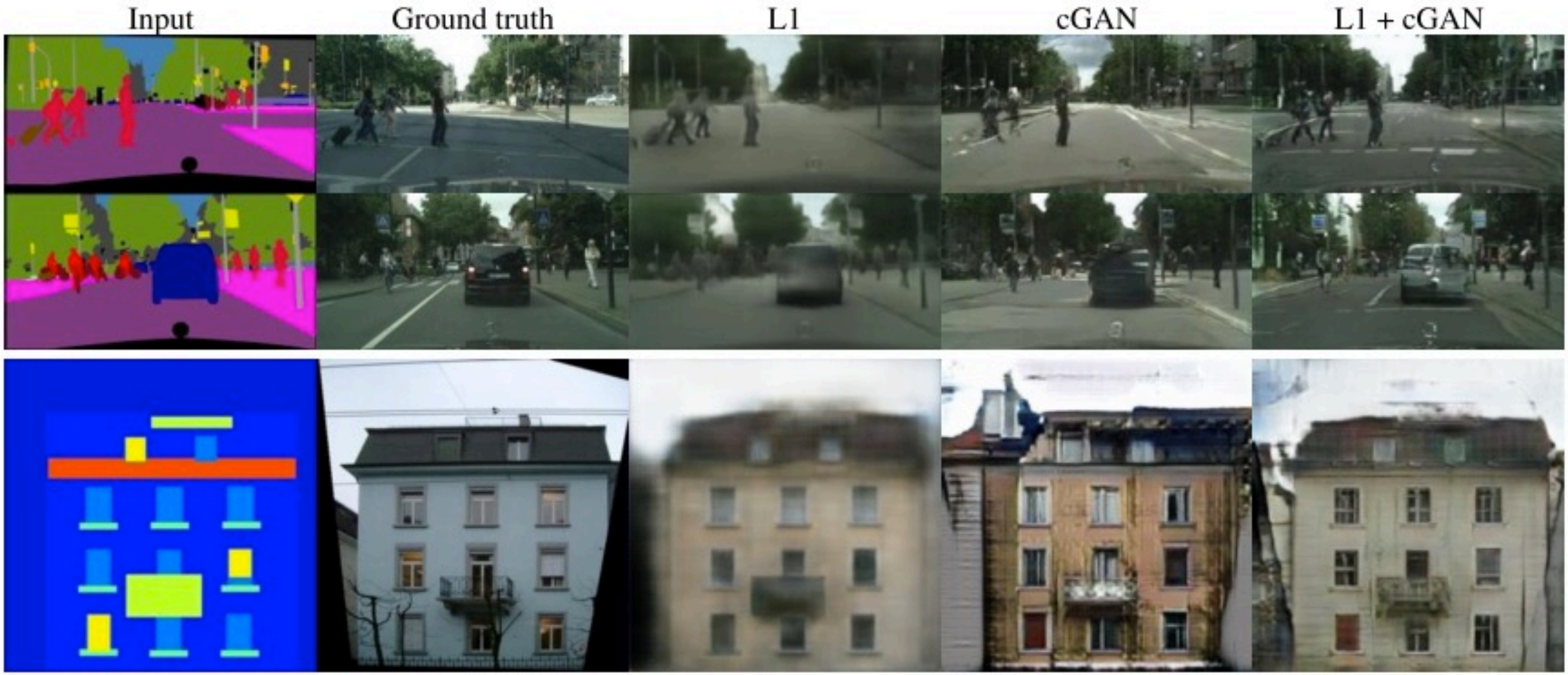


x: source image, y:target image, z: noise

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y \sim p_{\text{data}_a}(x,y)}[\log D(x, y)] + \mathbb{E}_{x \sim p_{\text{data}}(x), z \sim p_z(z)}[\log(1 - D(x, G(x, z)))]$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y \sim p_{\text{data}}(x,y), z \sim p_z(z)}\left[\|y - G(x, z)\|_1\right]$$

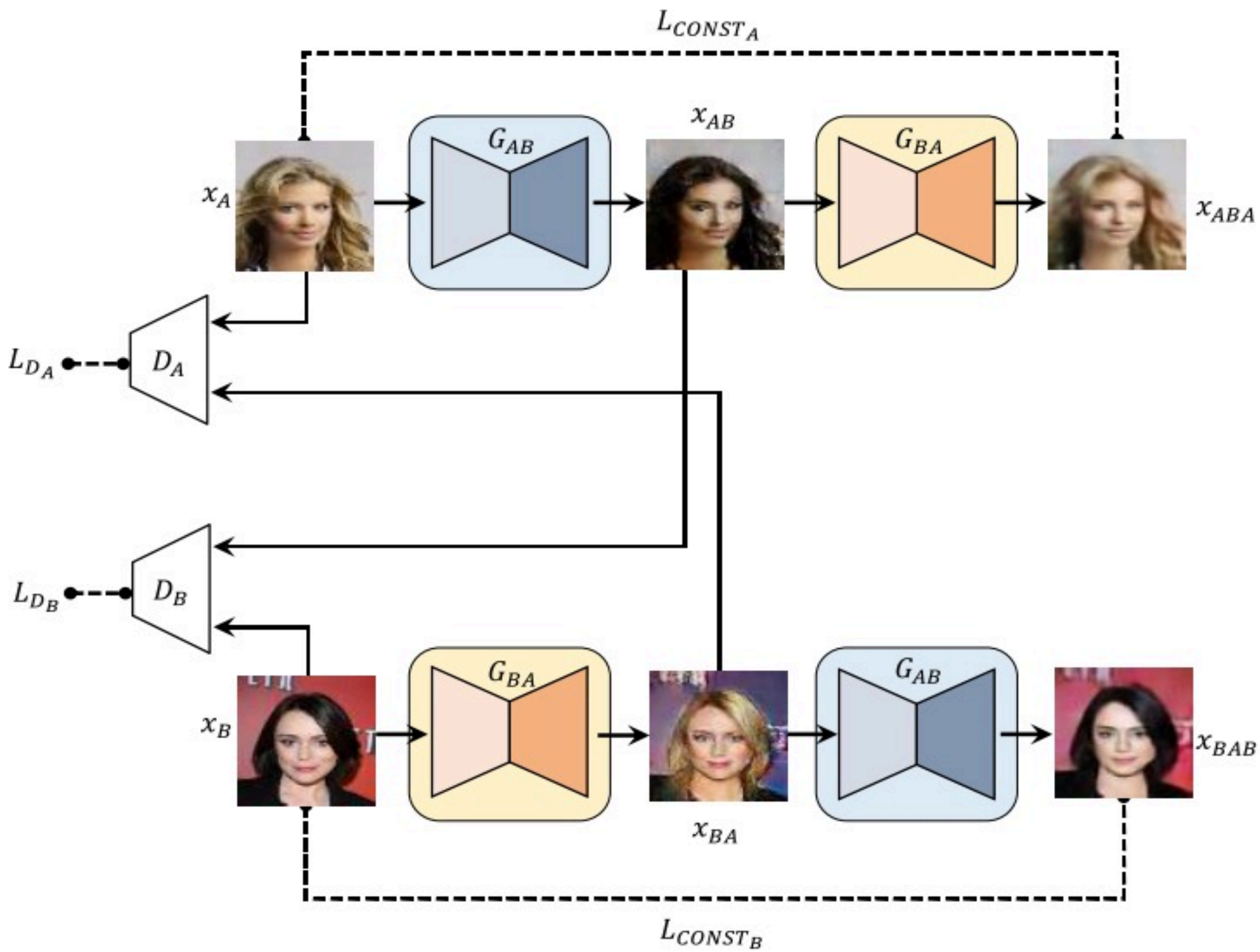
Paired case



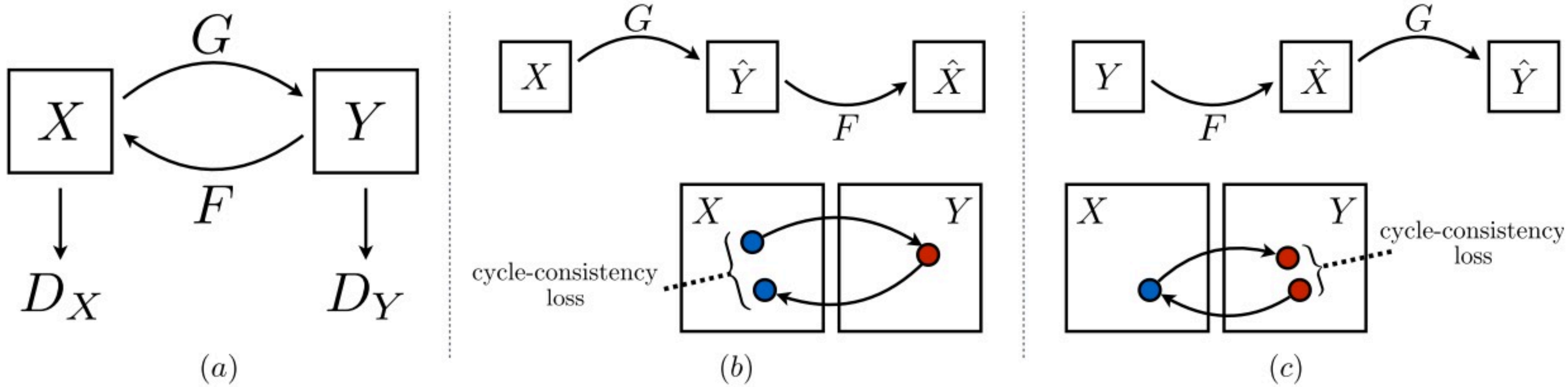
Paired case

Paired case

Unpaired case



Unpaired case: Cycle GAN / DISCO GAN



Cycle GAN / DISCO GAN

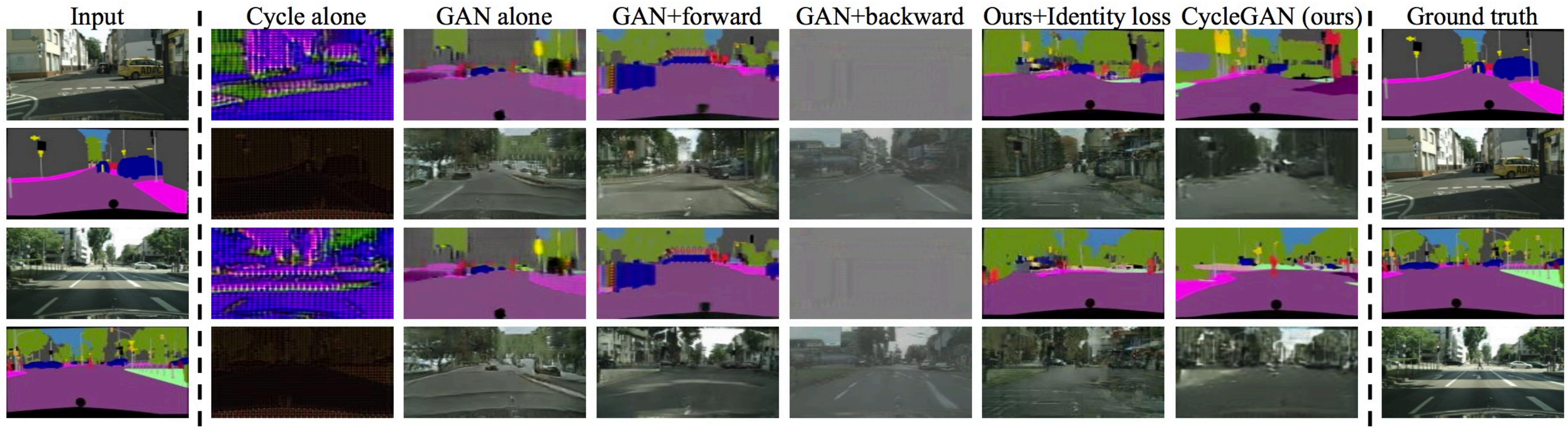
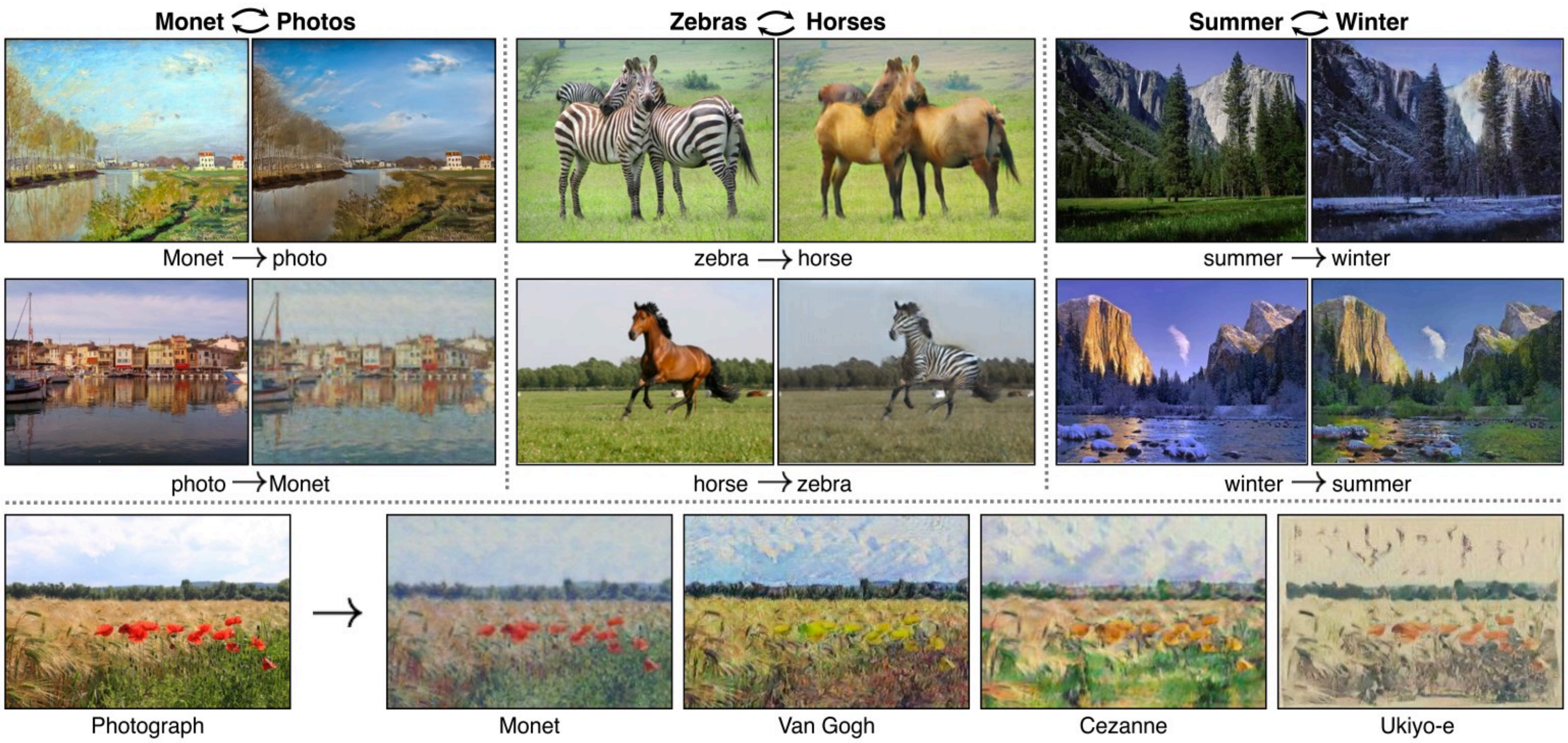
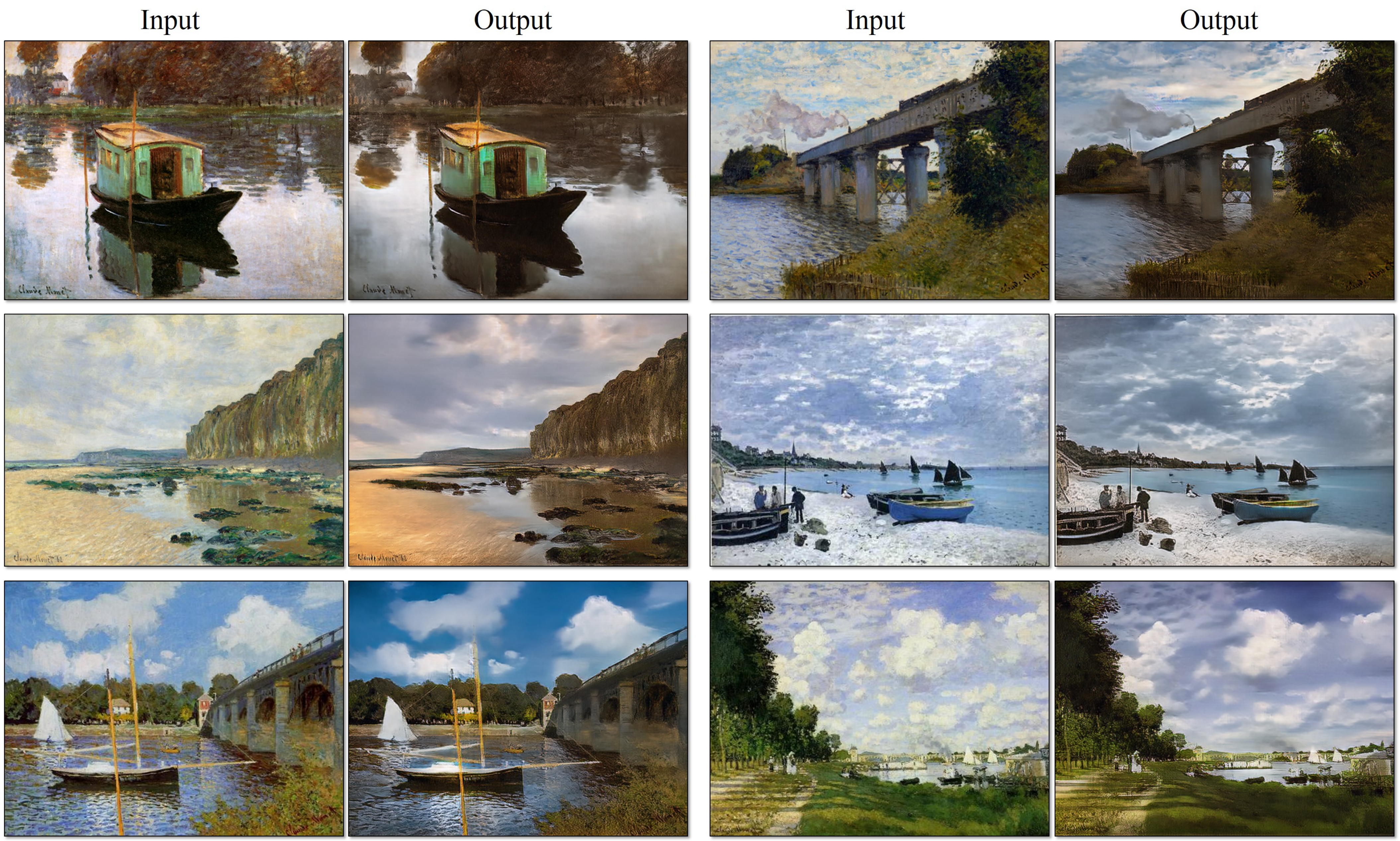


Figure 7: Different variants of our method for mapping labels↔photos trained on cityscapes. From left to right: input, cycle-consistency loss alone, adversarial loss alone, GAN + forward cycle-consistency loss ($F(G(x)) \approx x$), GAN + backward cycle-consistency loss ($G(F(y)) \approx y$), CycleGAN (our full method), and ground truth. Both *Cycle alone* and *GAN + backward* fail to produce images similar to the target domain. *GAN alone* and *GAN + forward* suffer from mode collapse, producing identical label maps regardless of the input photo.

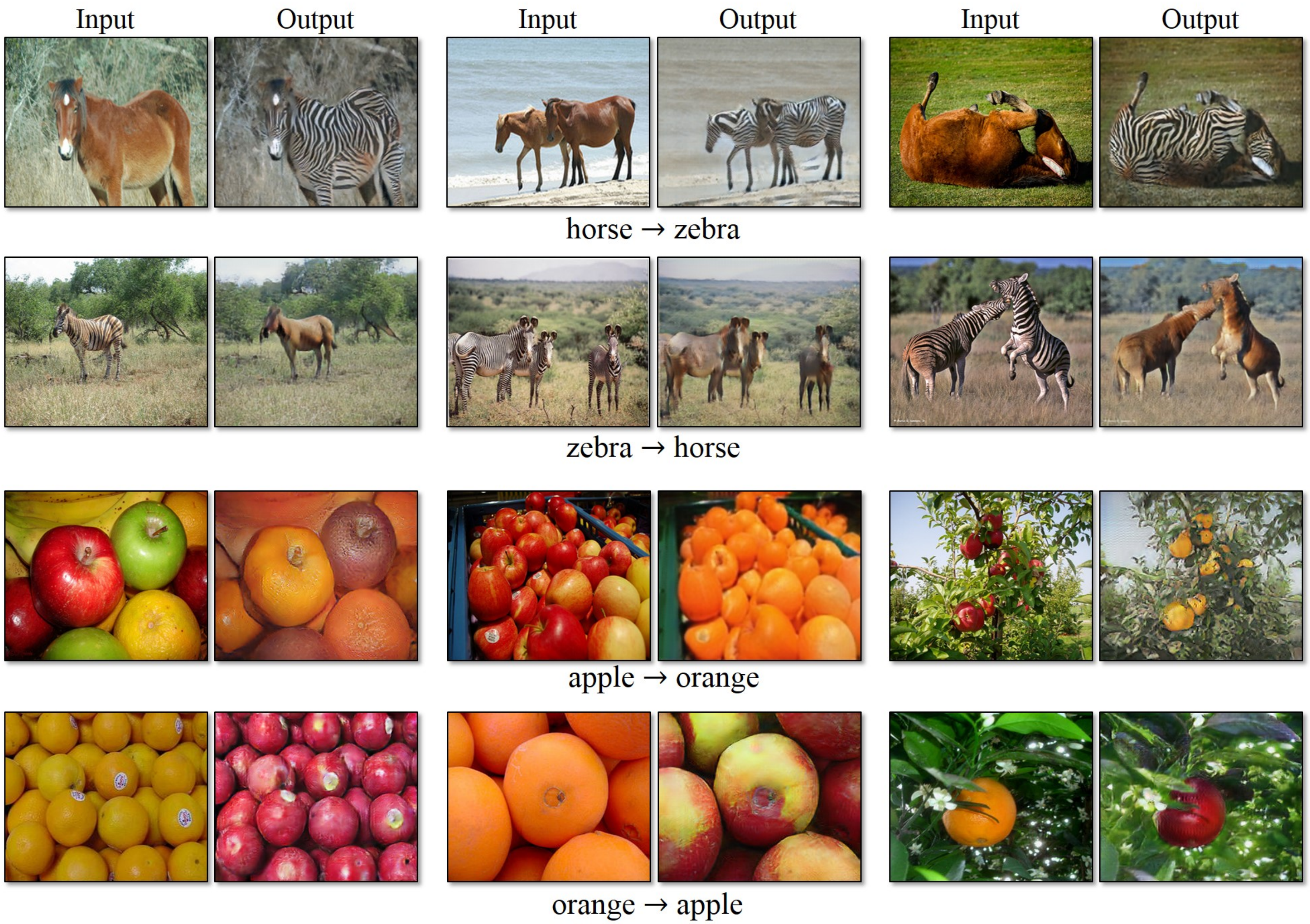
Unpaired case



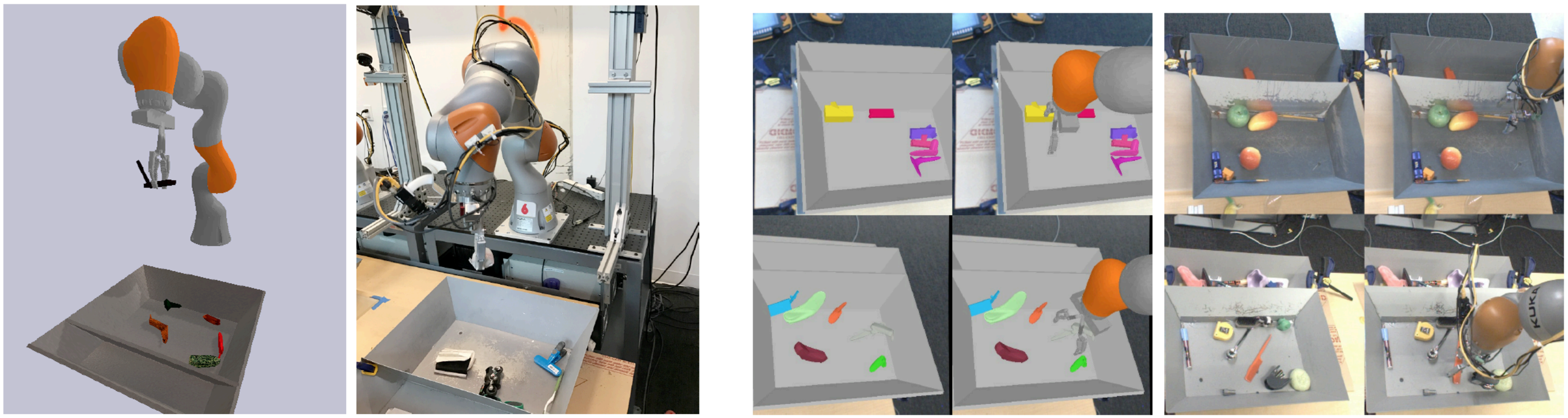
Unpaired case



Unpaired case



Sim2real for learning to grasp



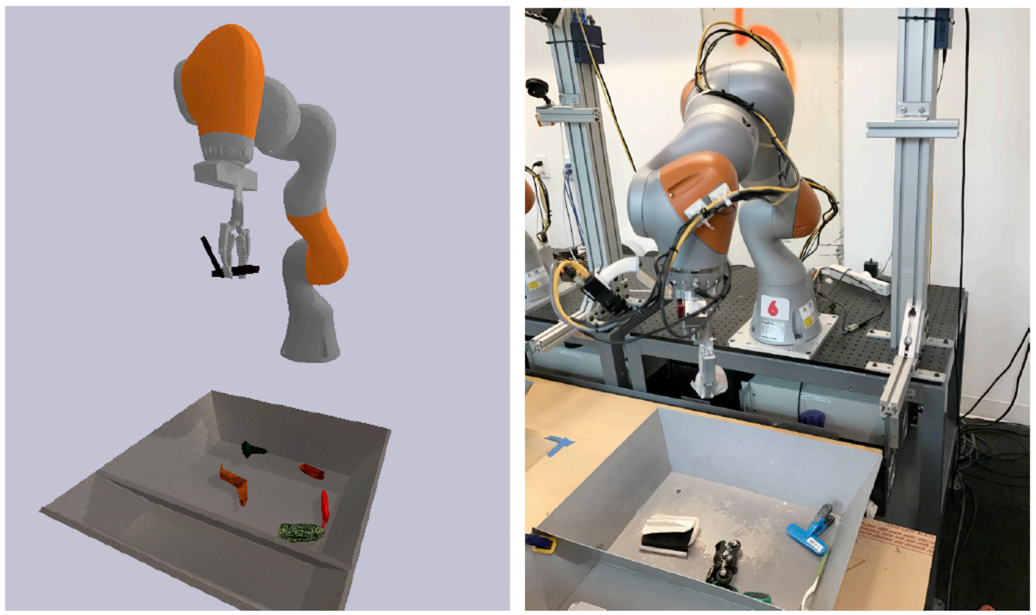
(a) Simulated World (b) Real World (c) Simulated Samples (d) Real Samples

- I want to learn the function $\mathrm{Grasp}(I, v; \theta)$: given image I and end-effector motion v, will I successfully grasp the object?

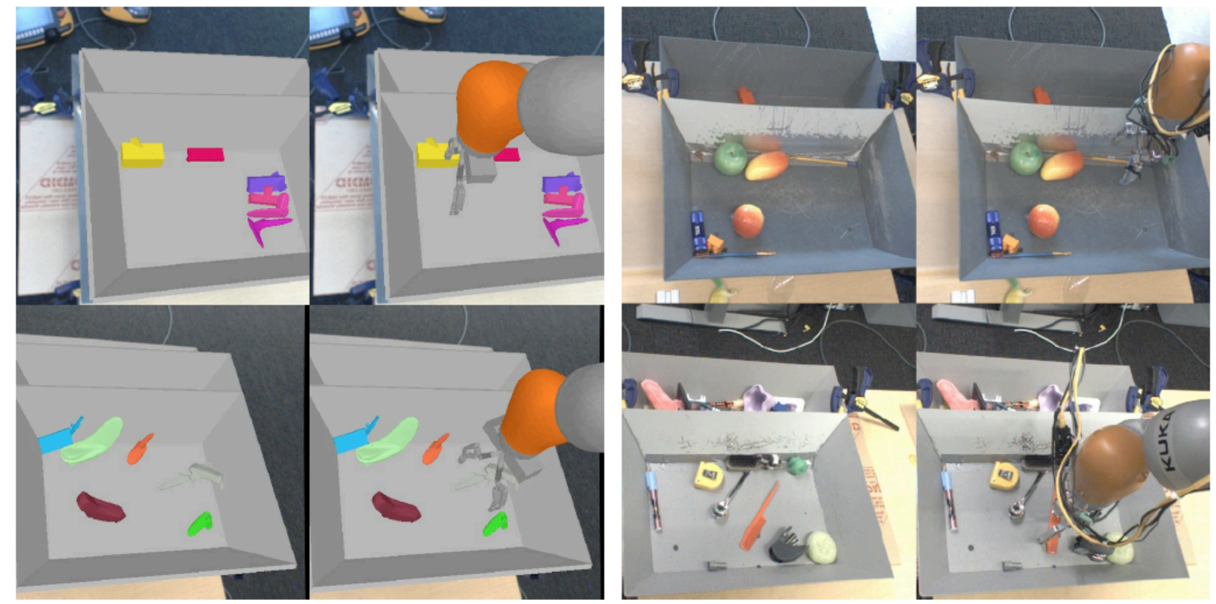
- $\mathrm{Grasp}(I, v; \theta)$ can be trained with supervised learning. I want to use a simulated environment to quickly collect lots of samples. I want it to generalize to the real world.

Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping, Google, CVPR 2017

Sim2real for learning to grasp



(a) Simulated World (b) Real World



(c) Simulated Samples (d) Real Samples

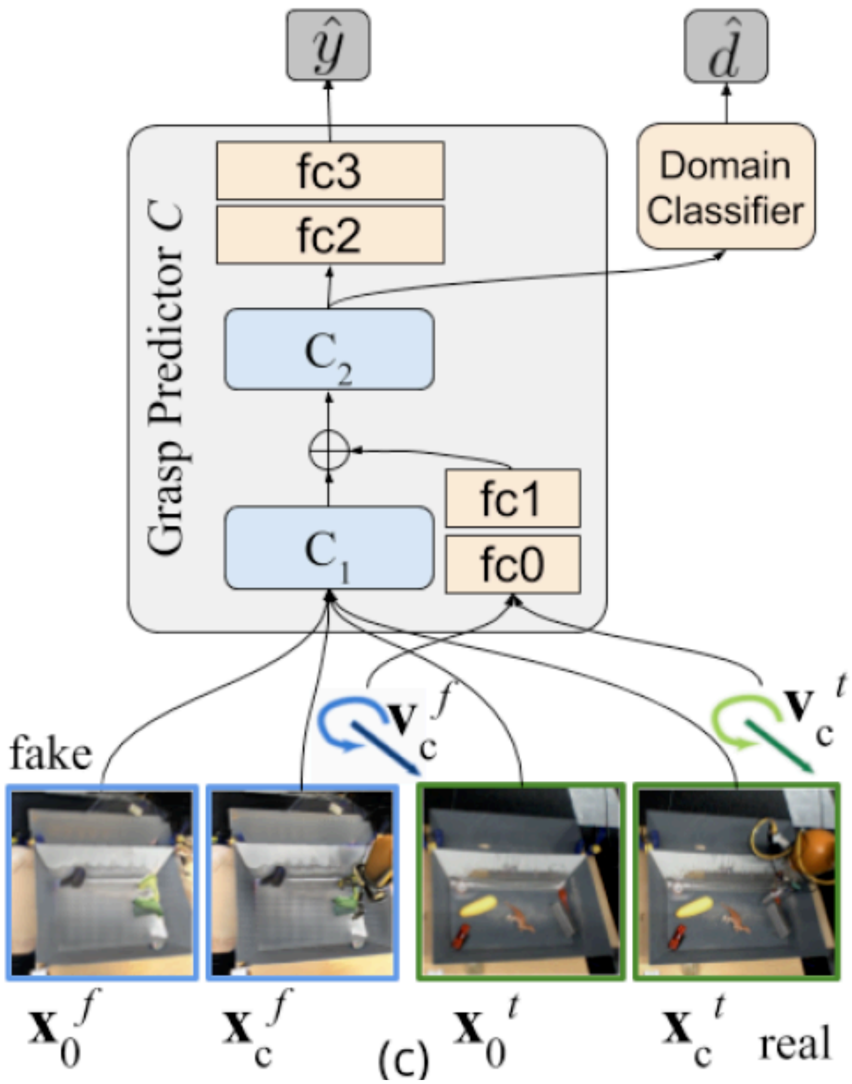
- Use Bullet simulator to emulate the Kuka hardware setup. Camera is mounted over the Kuka shoulder
- 51300 ShapeNet 3D models
- Use progressively better grasping models to collect data
- Randomization: both visuals and dynamics were randomized in simulation: the background image, object masses, textures, coefficients of friction.

Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping, Google, CVPR 2017

Feature adaptation

Two losses: domain confusion loss and grasping prediction loss

Grasping prediction (task loss)



We add a domain classifier, that attempts to classify the domain the features come from

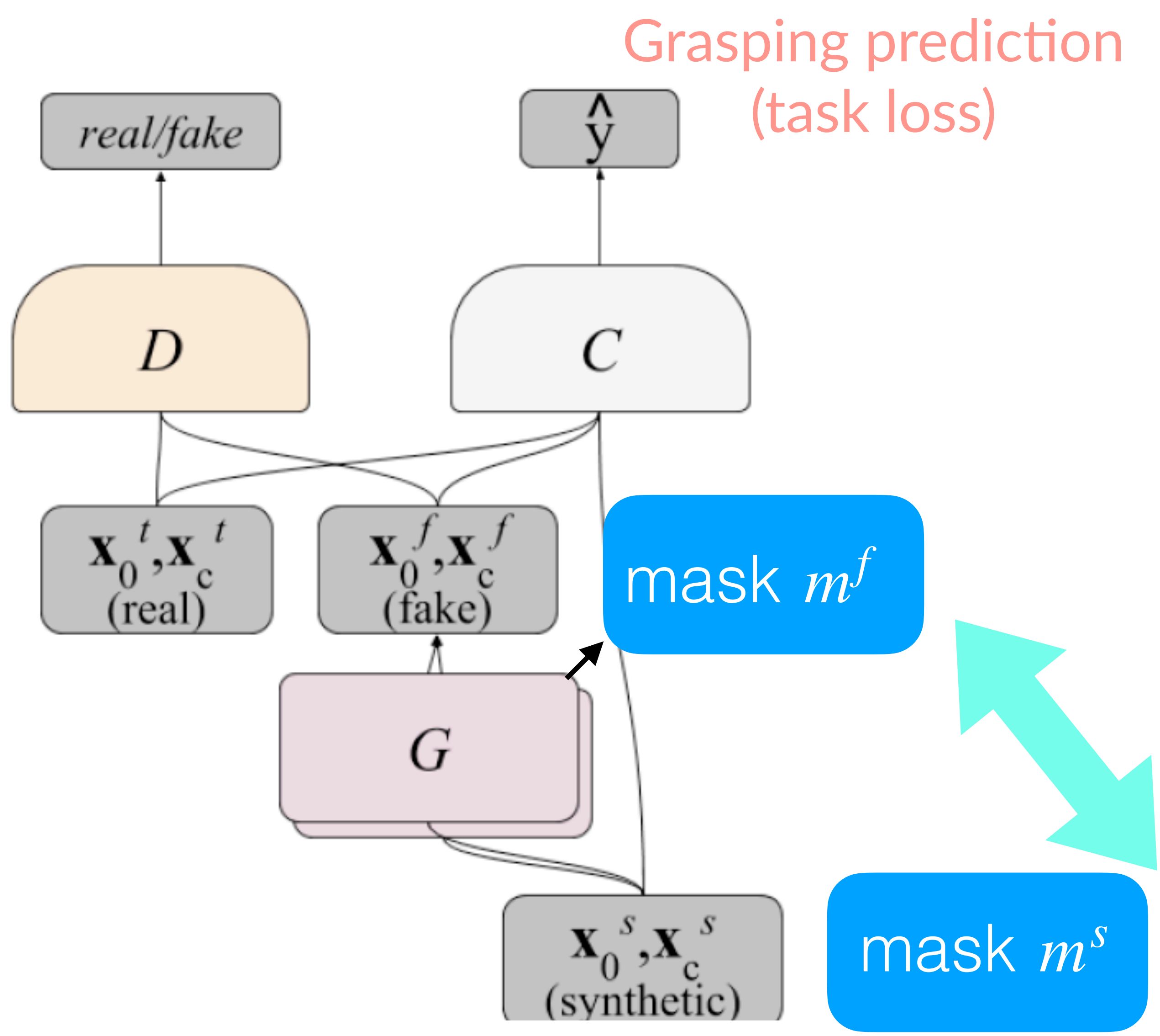
$$\mathscr{L}_{\text{DANN}} = \sum_{i=0}^{N_s+N_t} \{ d_i \log \hat{d}_i + (1-d_i) \log(1-\hat{d}_i) \}$$

The shared features C1, C2 attempt to confuse the domain classifier (maximize its loss), while the domain classifier features attempts to minimize its loss.

Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping, Google, CVPR 2017

Pixel Adaptation

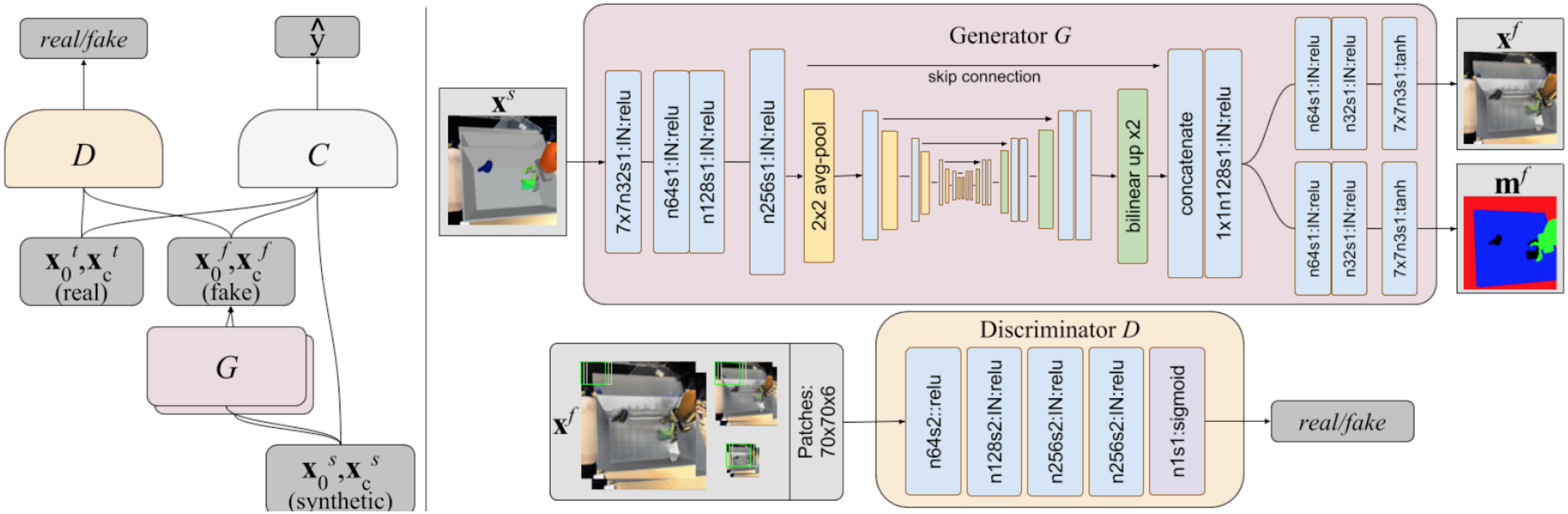
Three losses: grasping prediction loss, semantic labelling loss, adversarial loss



Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping, Google, CVPR 2017

Pixel Adaptation

Three losses: grasping prediction loss, semantic labelling loss, adversarial loss

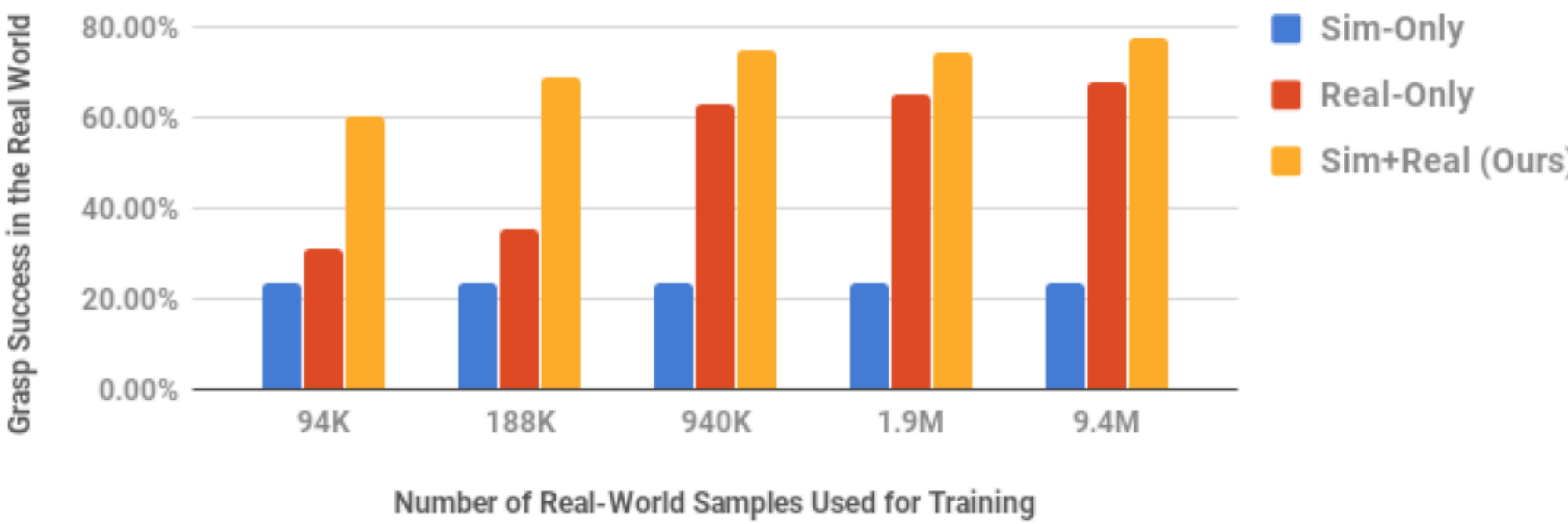


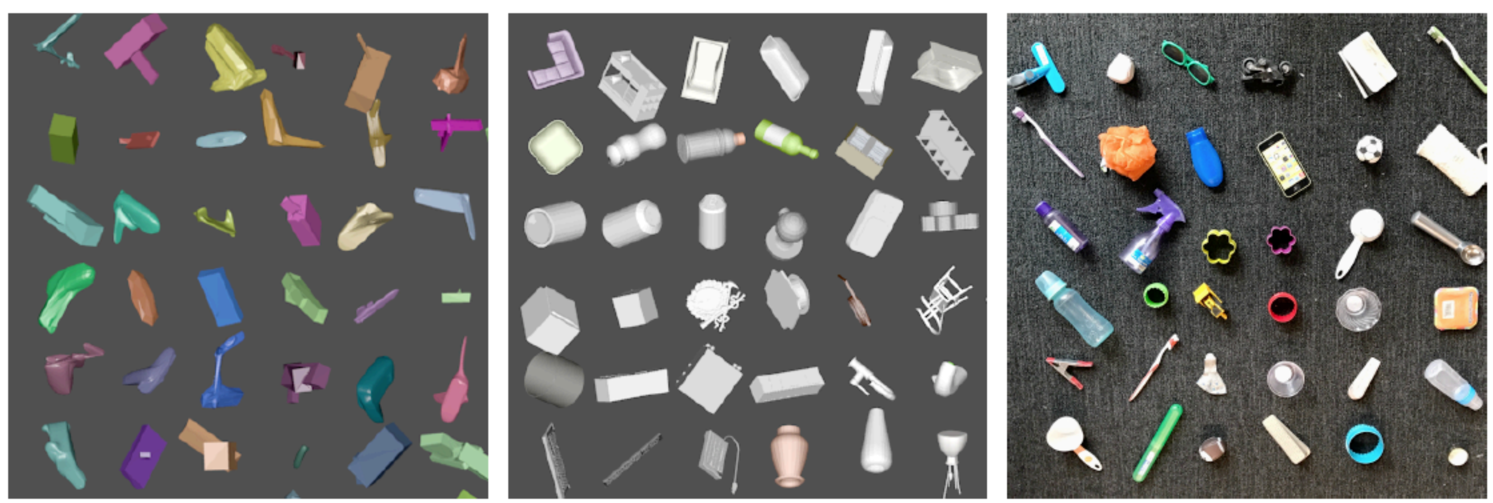
Goal: we want our generator to translate simulated images so that:

1. they do well in the task loss (grasping),
2. look real
3. retain the same semantics as their simulated counterparts

Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping, Google, CVPR 2017

Results





(a) Procedural (b) ShapeNet [31] (c) Real