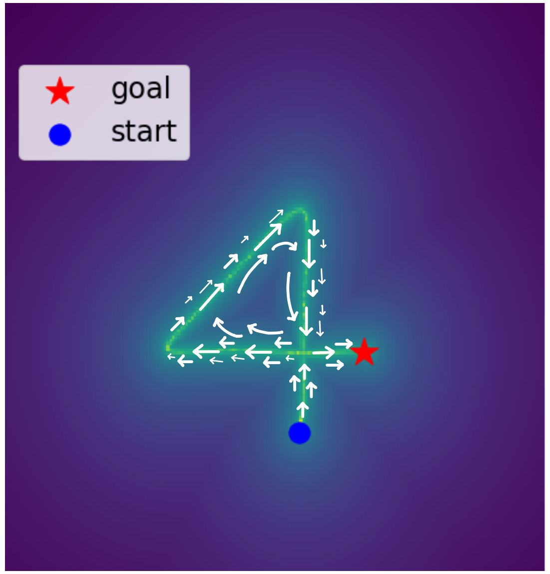
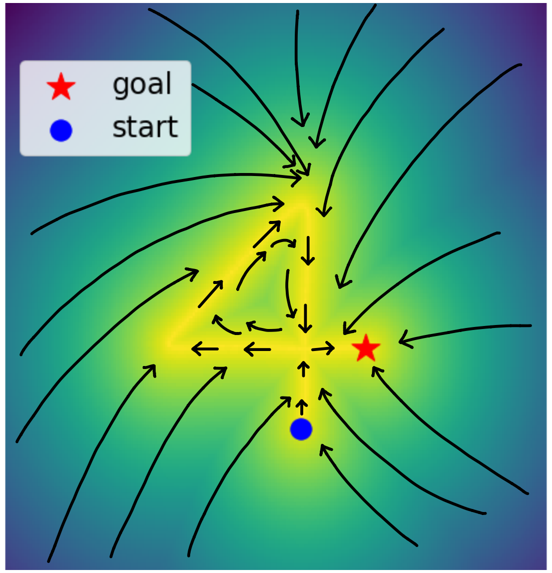
Neural Dynamic Policies for End-to-End Sensorimotor Learning



Vanilla Policy



NDP (Ours)

Shikhar Bahl, Mustafa Mukadam, Abhinav Gupta, Deepak Pathak


KINOVA

✓ Needs to reason in trajectory space

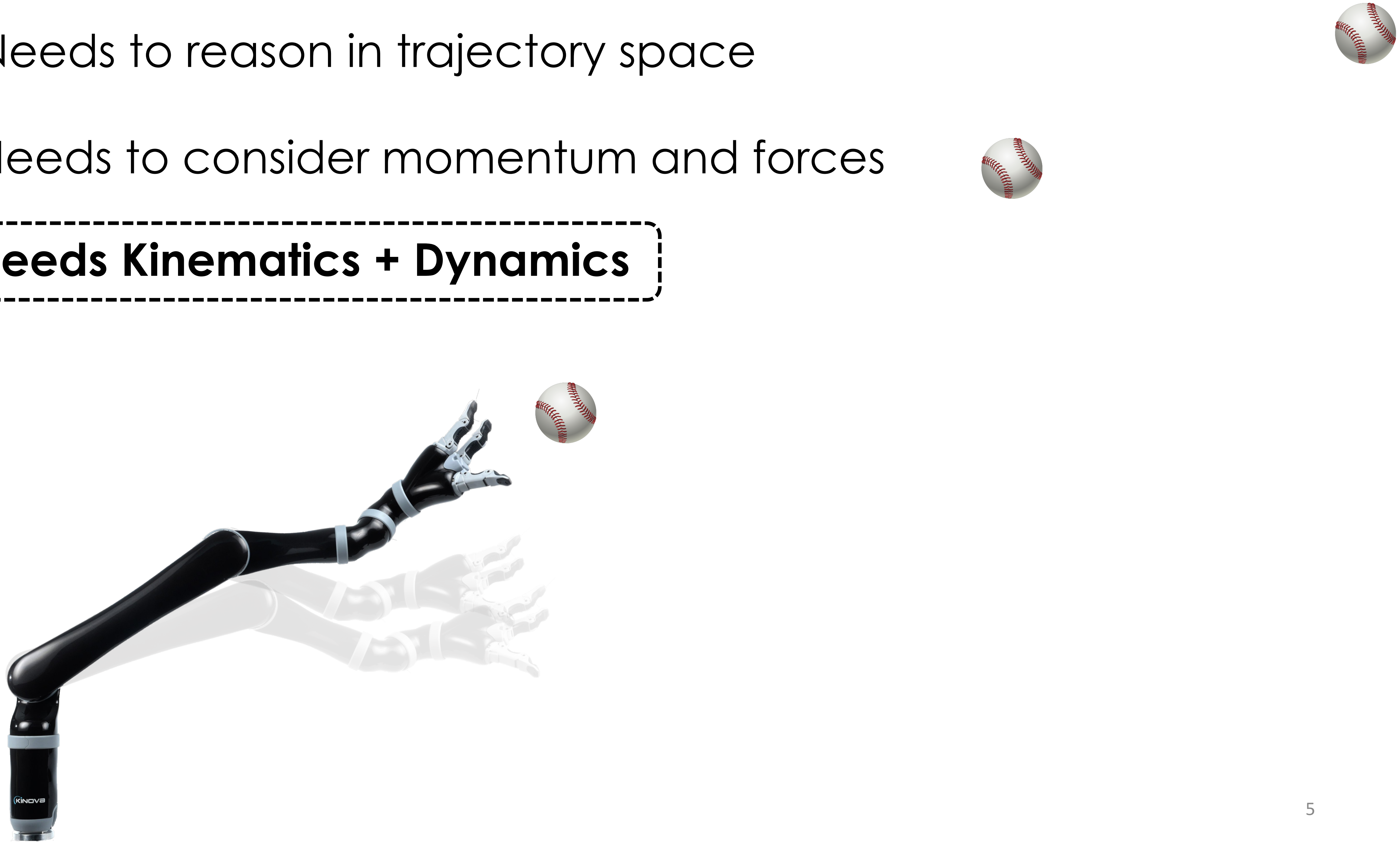
✓ Needs to reason in trajectory space

✓ Needs to consider momentum and forces

✓ Needs to reason in trajectory space

✓ Needs to consider momentum and forces

Needs Kinematics + Dynamics



✓ Needs to reason in trajectory space

✓ Needs to consider momentum and forces

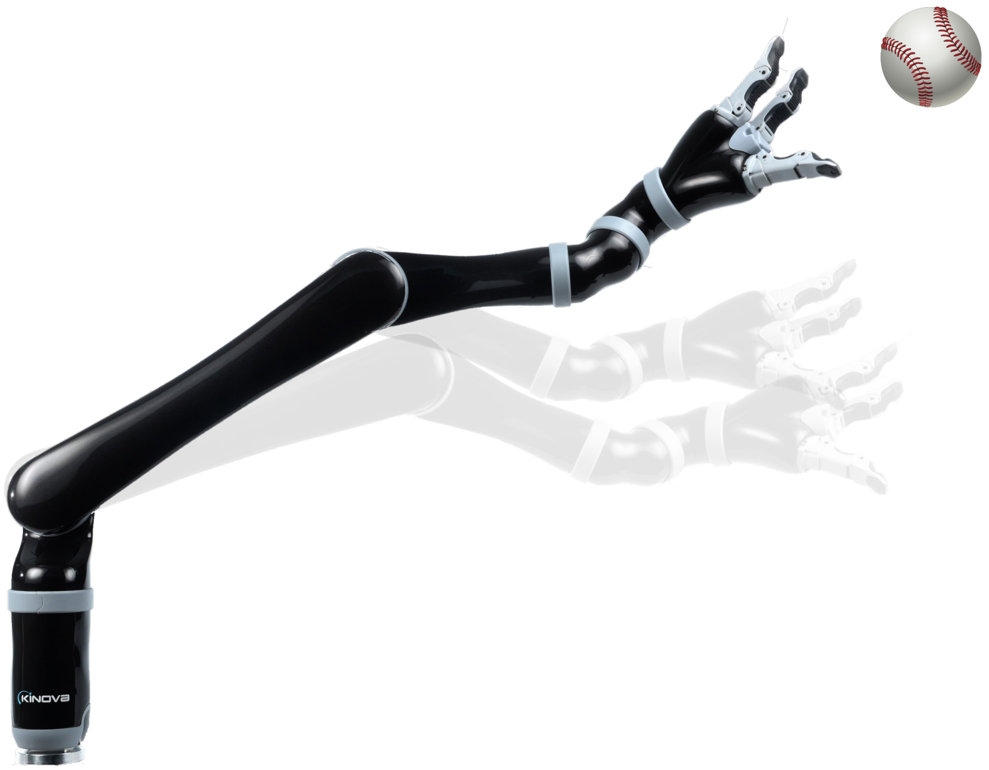
Needs Kinematics + Dynamics

✗ Deep robot learning methods only reason at each timestep

✓ Needs to reason in trajectory space

✓ Needs to consider momentum and forces

Needs Kinematics + Dynamics



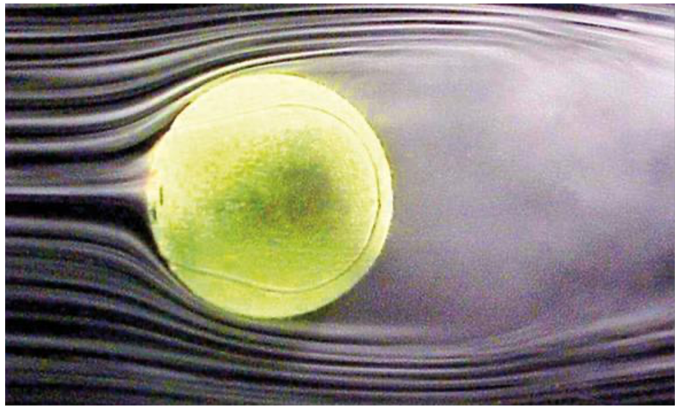
✗ Deep robot learning methods only reason at each timestep

✗ Only operate in raw action space (torque, joint angles, etc)

Can we build policies that reason directly in trajectory space?

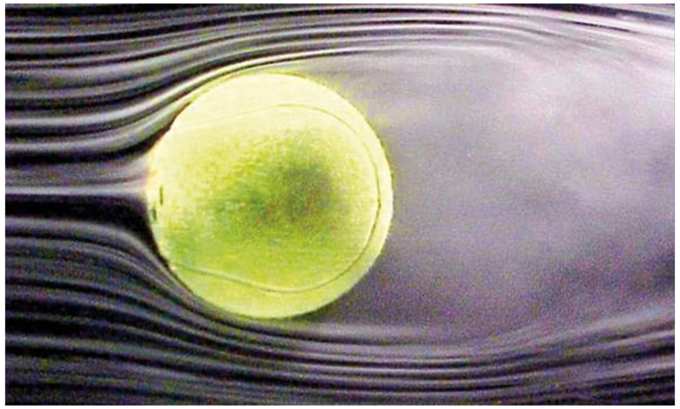
Many natural phenomena are dynamical systems which are described by ***differential equations***, i.e. $\ddot{y} = m^{-1} f(y, \dot{y})$

Many natural phenomena are dynamical systems which are described by ***differential equations***, i.e. $\ddot{y} = m^{-1} f(y, \dot{y})$

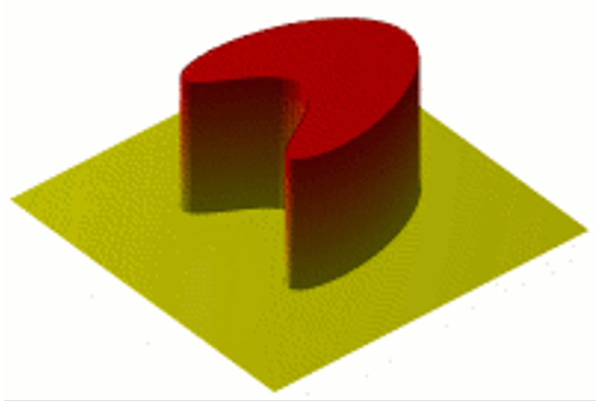


Fluids

Many natural phenomena are dynamical systems which are described by ***differential equations***, i.e. $\ddot{y} = m^{-1} f(y, \dot{y})$

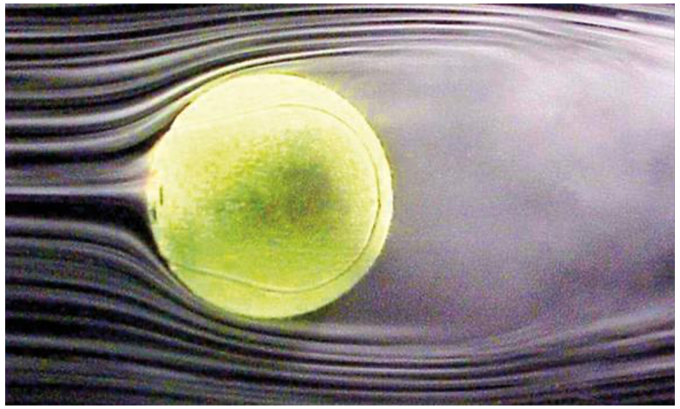


Fluids

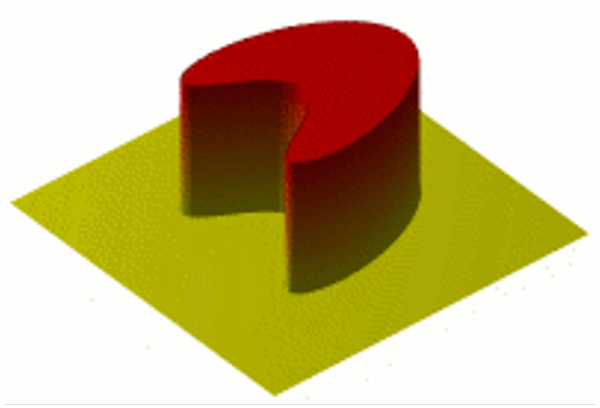


Heat

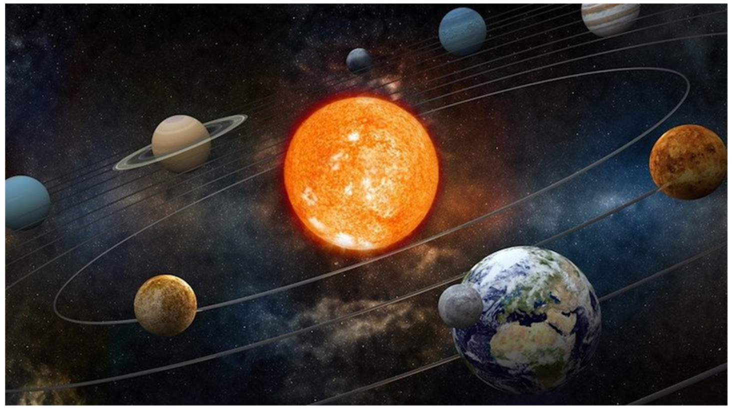
Many natural phenomena are dynamical systems which are described by ***differential equations***, i.e. $\ddot{y} = m^{-1} f(y, \dot{y})$



Fluids

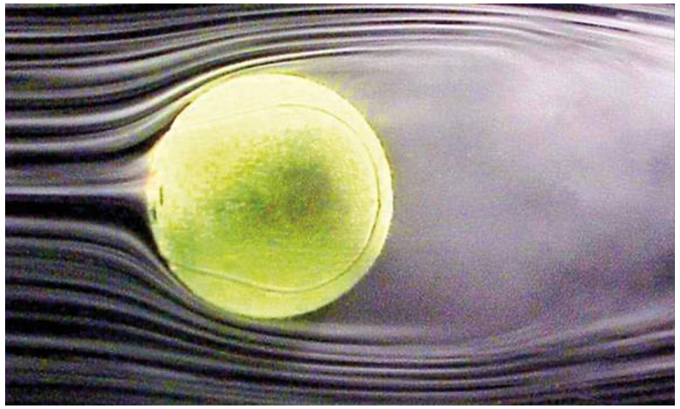


Heat

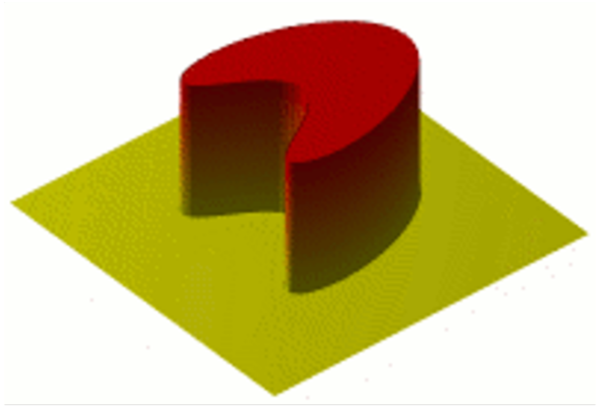


Gravity

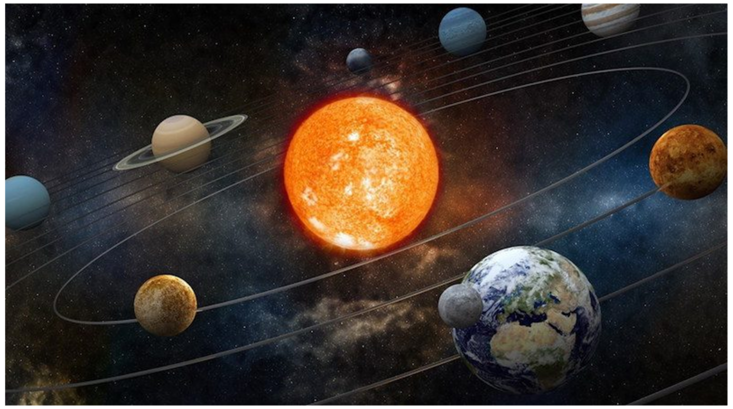
Many natural phenomena are dynamical systems which are described by ***differential equations***, i.e. $\ddot{y} = m^{-1} f(y, \dot{y})$



Fluids



Heat



Gravity

Can we use the same formulation be used to describe the motion of a robot?

Popular in classical robotics: ***Dynamic Movement Primitives (DMPs)*** [Schaal, 2002]

Popular in classical robotics: ***Dynamic Movement Primitives (DMPs)*** [Schaal, 2002]

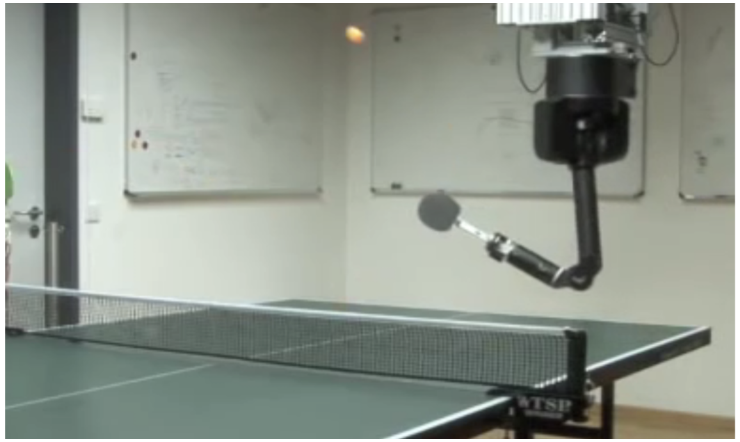
Popular in classical robotics: ***Dynamic Movement Primitives (DMPs)*** [Schaal, 2002]

Second order dynamical system

Popular in classical robotics: ***Dynamic Movement Primitives (DMPs)*** [Schaal, 2002]

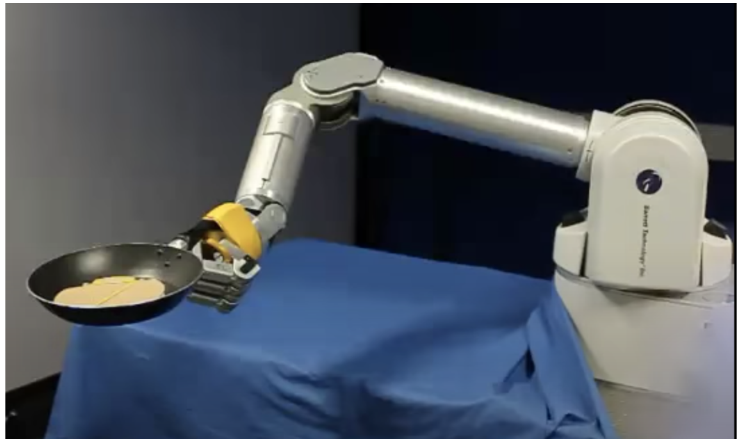
Second order dynamical system

Table Tennis



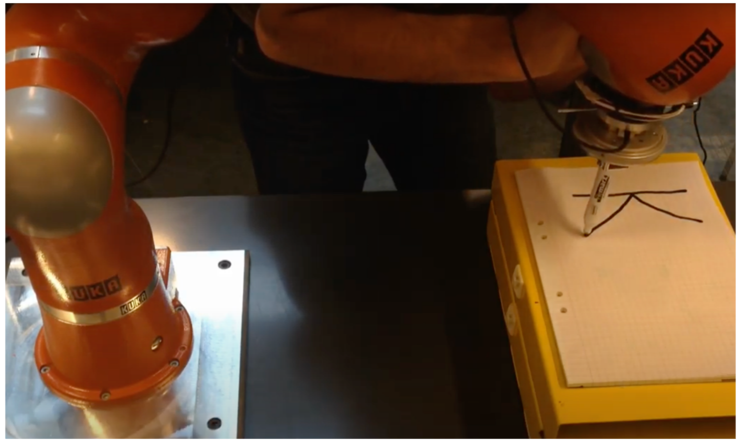
[Muelling et. al, 2013]

Pancake Flipping



[Kormushev et. al, 2010]

Letter Writing



[Steinmetz, 2014]

DMP-based Methods

Handle dynamic tasks well

DMP-based Methods

Handle dynamic tasks well

Can reason in trajectory space

DMP-based Methods

Handle dynamic tasks well

Can reason in trajectory space

Require dense supervision (i.e. demonstrations) + sensitive to parameter tuning

DMP-based Methods

Handle dynamic tasks well

Can reason in trajectory space

Require dense supervision (i.e. demonstrations) + sensitive to parameter tuning

Do not scale to high dimensional inputs

DMP-based Methods

Handle dynamic tasks well

Can reason in trajectory space

Require dense supervision (i.e. demonstrations) + sensitive to parameter tuning

Do not scale to high dimensional inputs

Deep RL

Can handle high-dimensional input (i.e. images)

DMP-based Methods

Handle dynamic tasks well

Can reason in trajectory space

Require dense supervision (i.e. demonstrations) + sensitive to parameter tuning

Do not scale to high dimensional inputs

Deep RL

Can handle high-dimensional input (i.e. images)

Can learn from weak supervision (i.e. rewards)

DMP-based Methods	Deep RL
Handle dynamic tasks well	Can handle high-dimensional input (i.e. images)
Can reason in trajectory space	Can learn from weak supervision (i.e. rewards)
Require dense supervision (i.e. demonstrations) + sensitive to parameter tuning	Reason at each timestep
Do not scale to high dimensional inputs	

DMP-based Methods	Deep RL
Handle dynamic tasks well	Can handle high-dimensional input (i.e. images)
Can reason in trajectory space	Can learn from weak supervision (i.e. rewards)
Require dense supervision (i.e. demonstrations) + sensitive to parameter tuning	Reason at each timestep
Do not scale to high dimensional inputs	Difficulty with dynamic tasks

How can we bridge the gap between these two paradigms?

DMP-based Methods

Handle dynamic task well

Are sensitive to parameter tuning and require dense supervision

Do not scale to high dimensional inputs

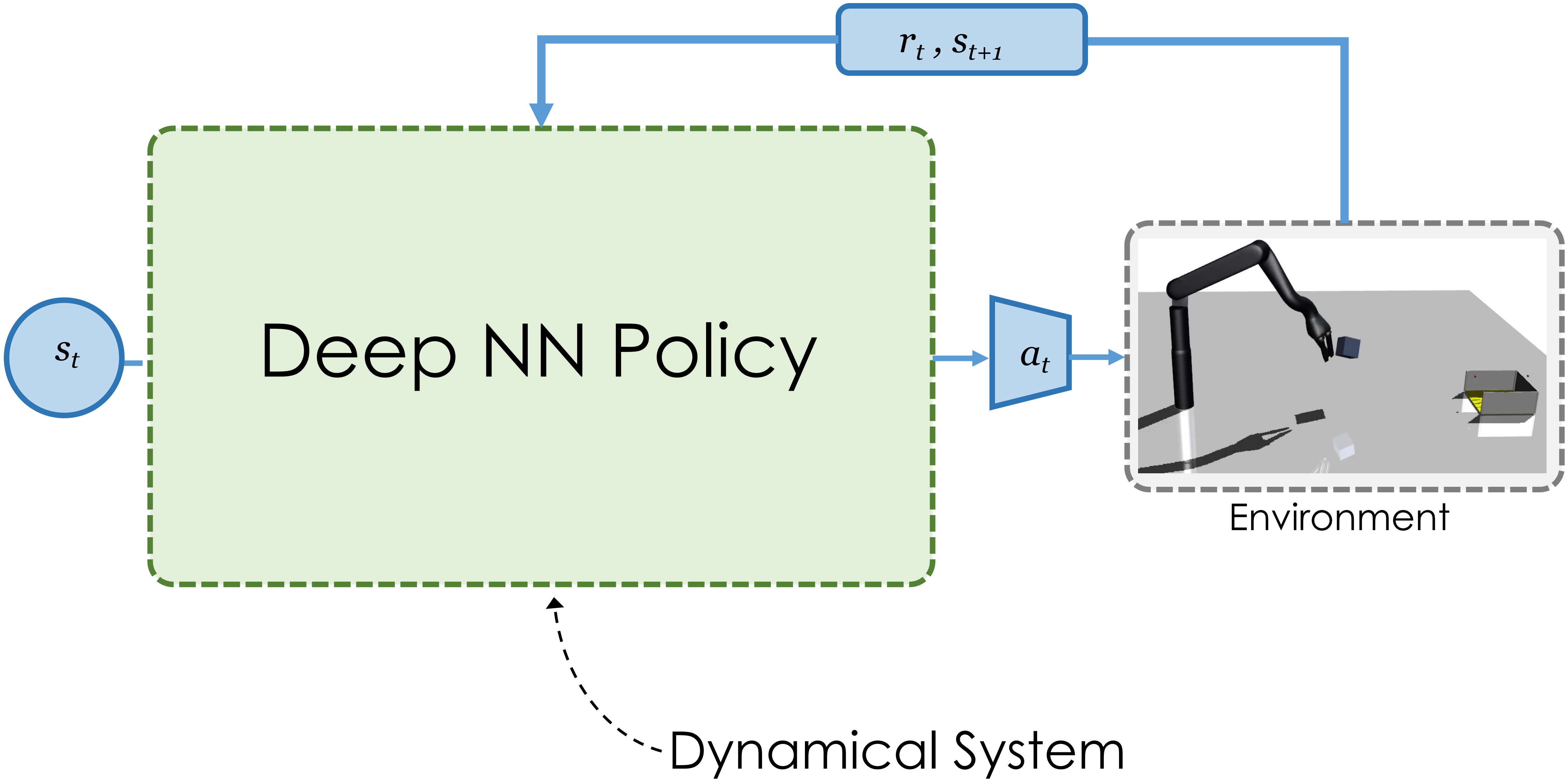
Deep RL

Can handle high-dimensional input (i.e. images)

supervision (i.e. rewards)

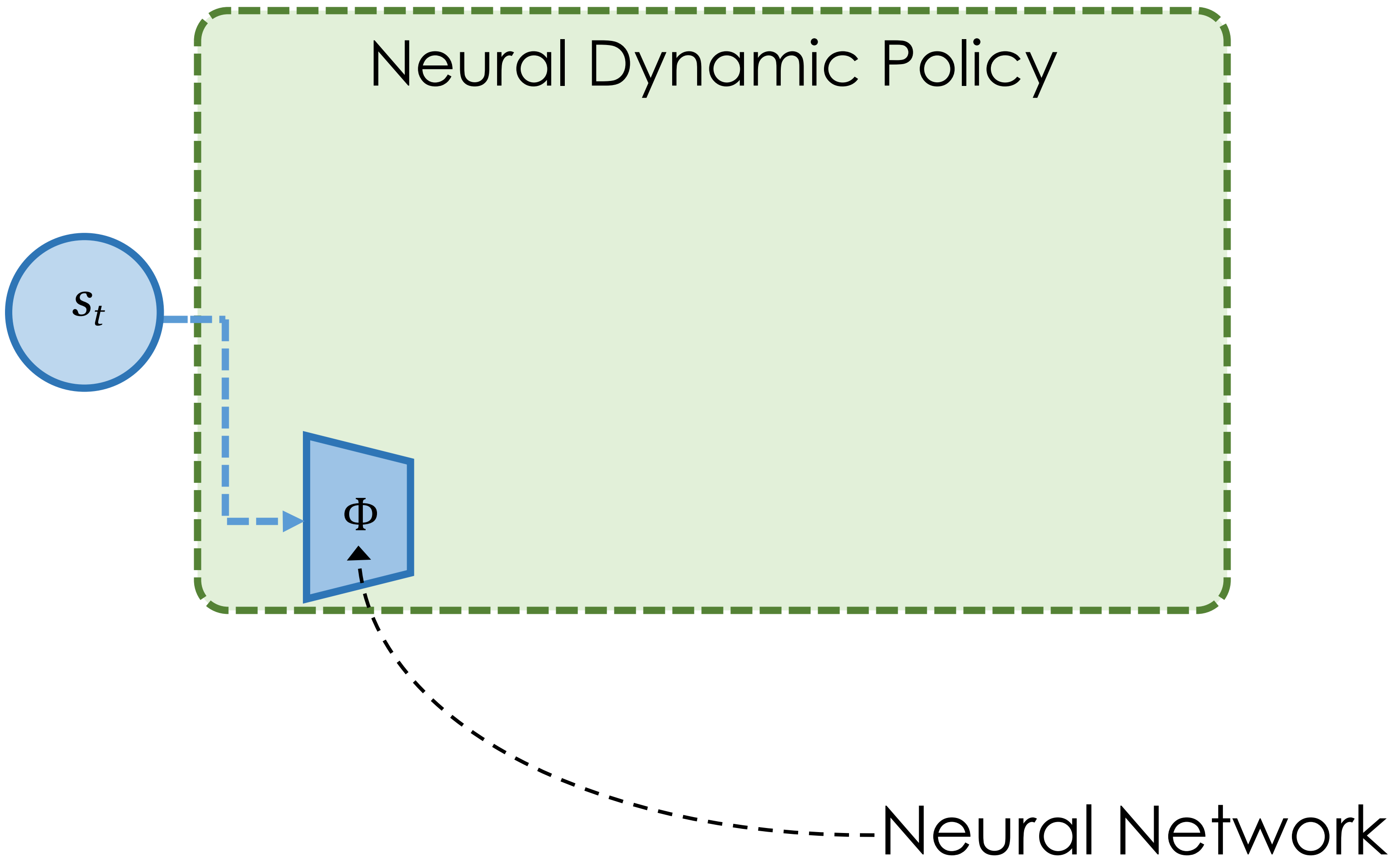
Reason at each timestep

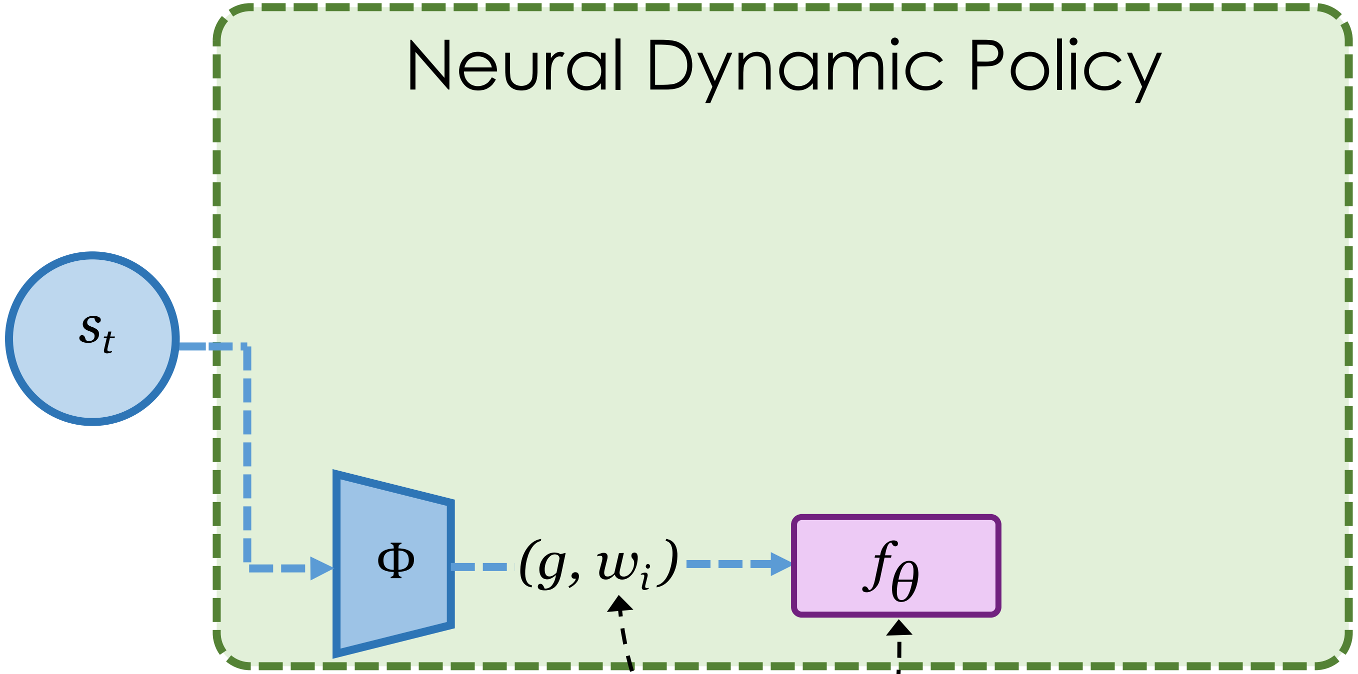
Difficulty with dynamic tasks



r_t , s_{t+1}
s_t
Deep NN Policy
a_t
Environment
Dynamical System

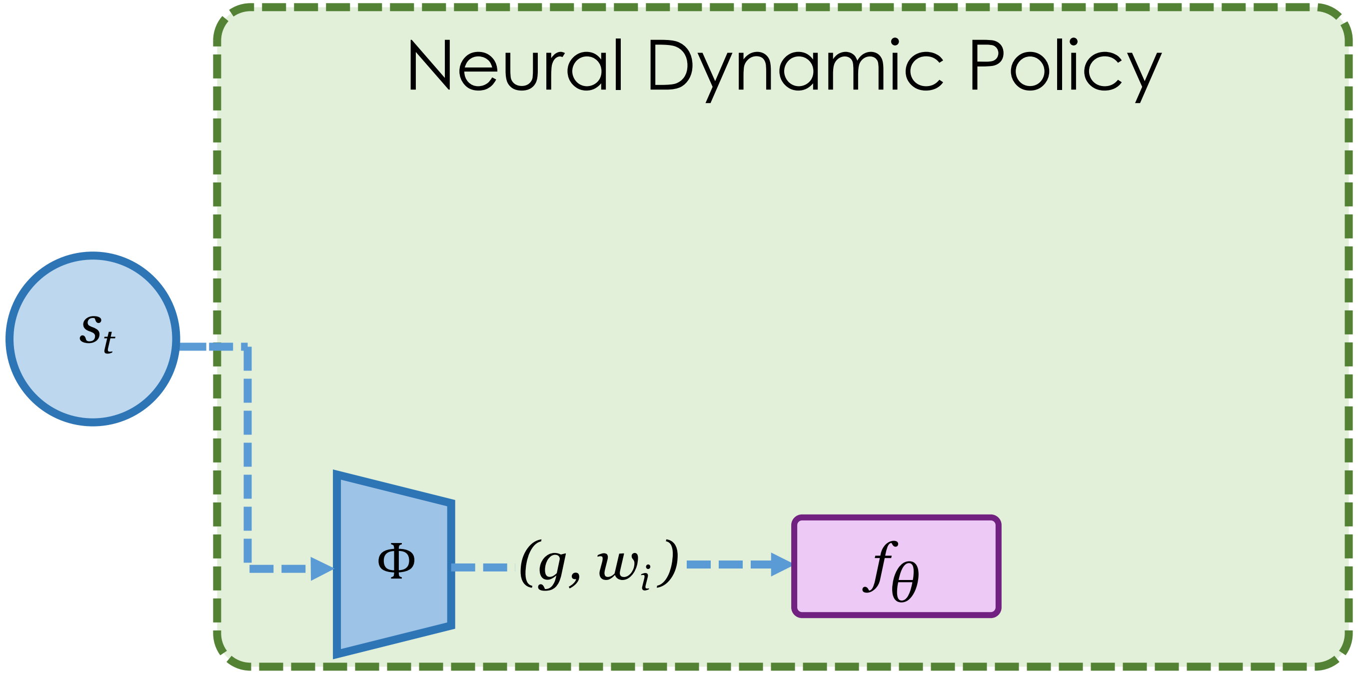
Neural Dynamic Policy


Neural Dynamic Policy
s_t
Φ
Neural Network



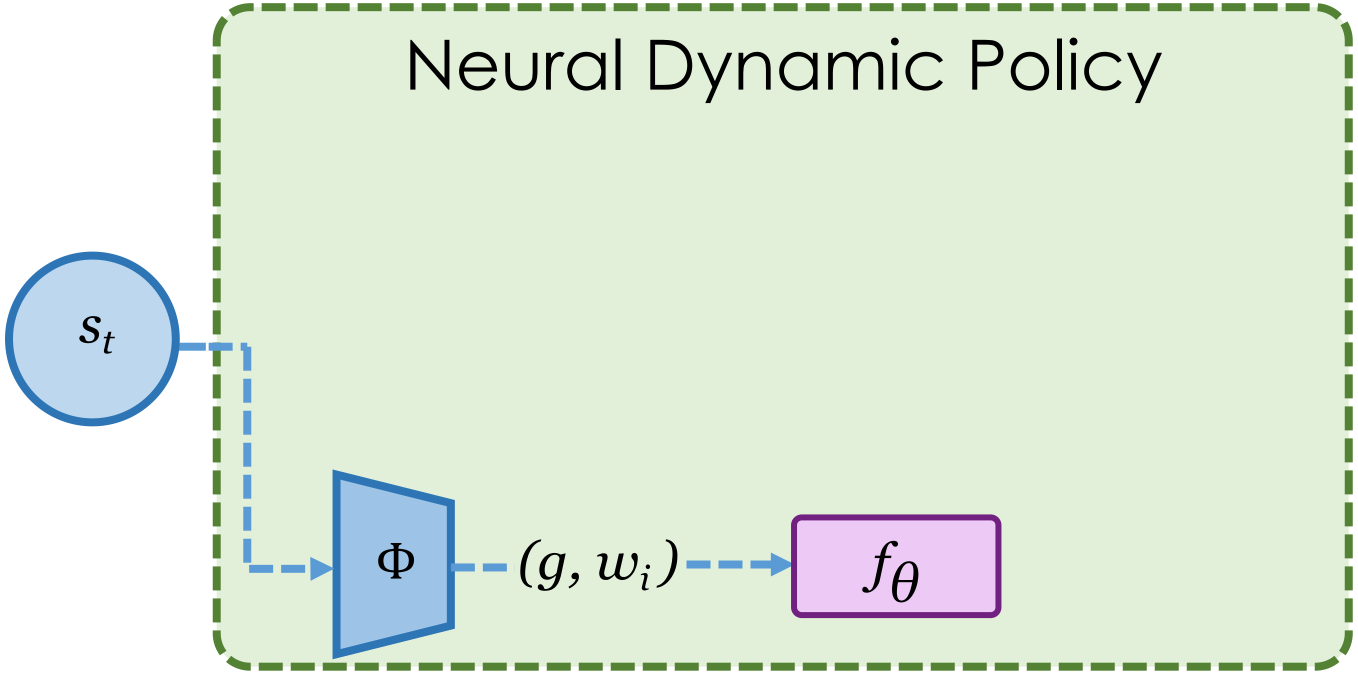
$$\mathbf{a} = -\alpha(\mathbf{y} - \boxed{g}) - \beta\dot{\mathbf{y}} + \boxed{f}$$

DMP Parameters



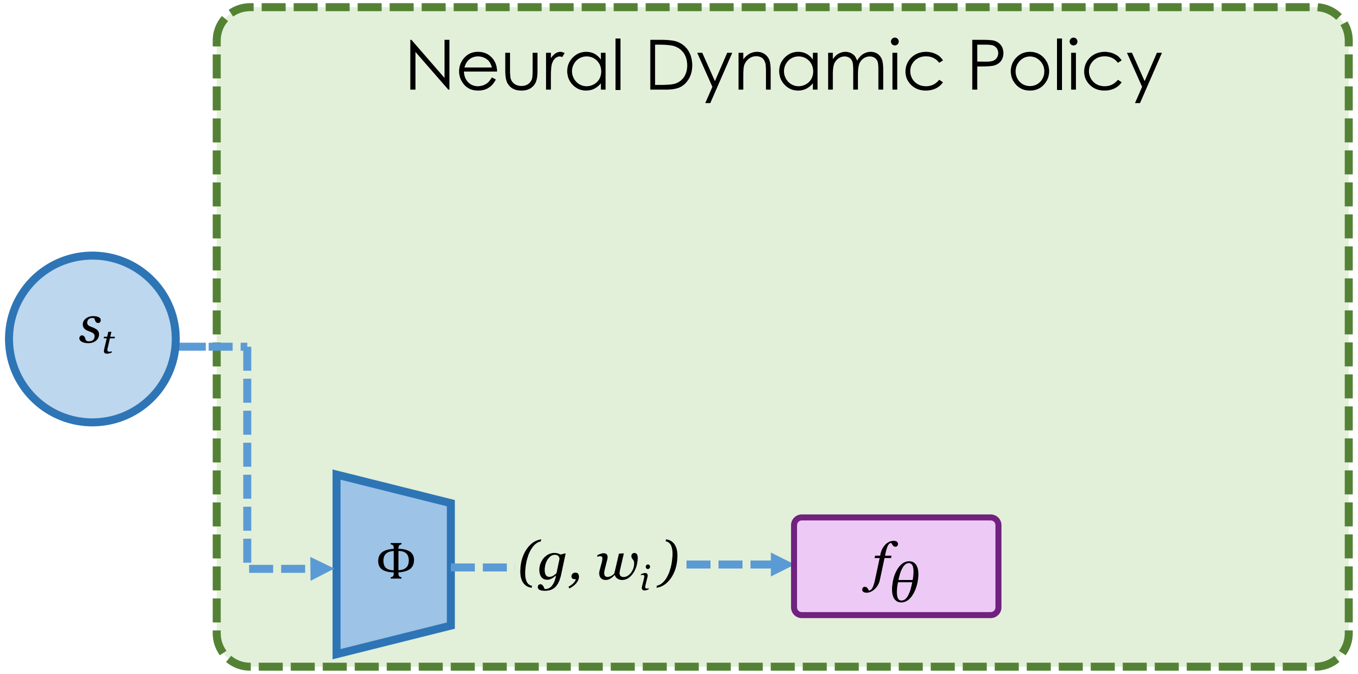
$$\mathbf{a} = -\alpha(\boxed{\mathbf{y}} - g) - \beta\dot{\mathbf{y}} + f$$

Robot position



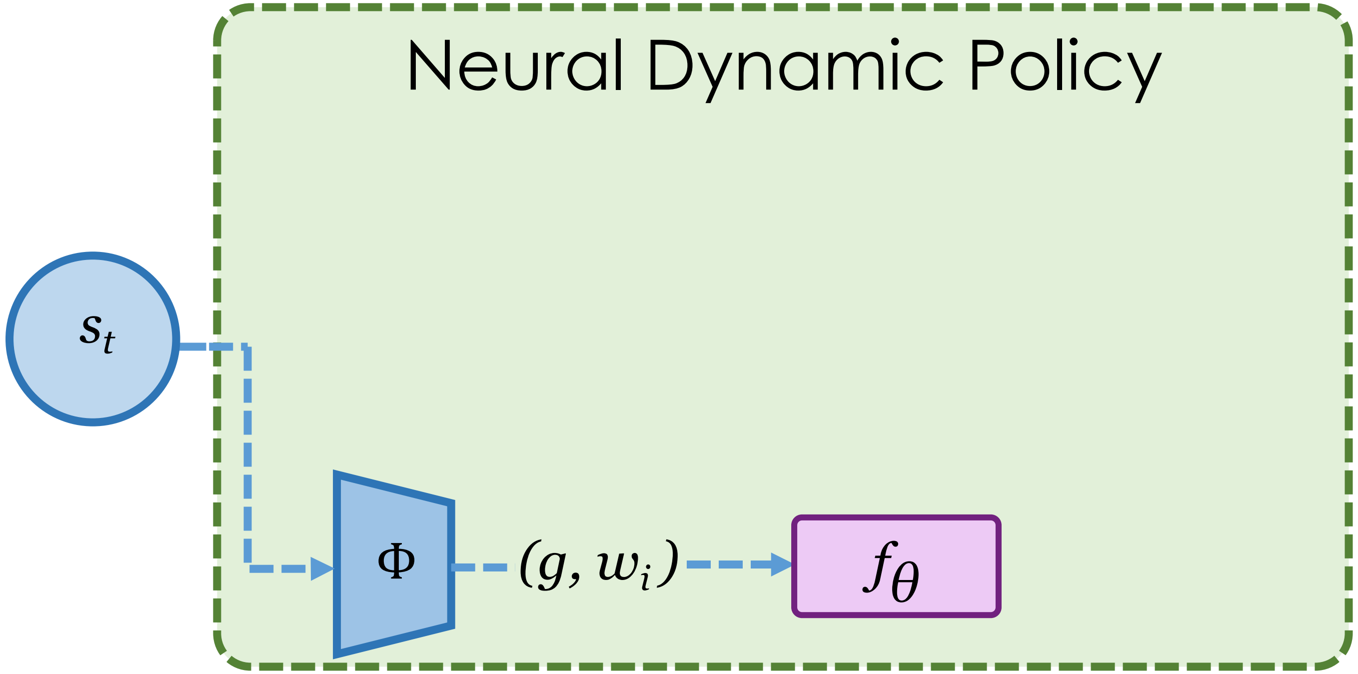
$$\mathbf{a} = -\alpha(\mathbf{y} - \boxed{g}) - \beta\dot{\mathbf{y}} + f$$

Goal position



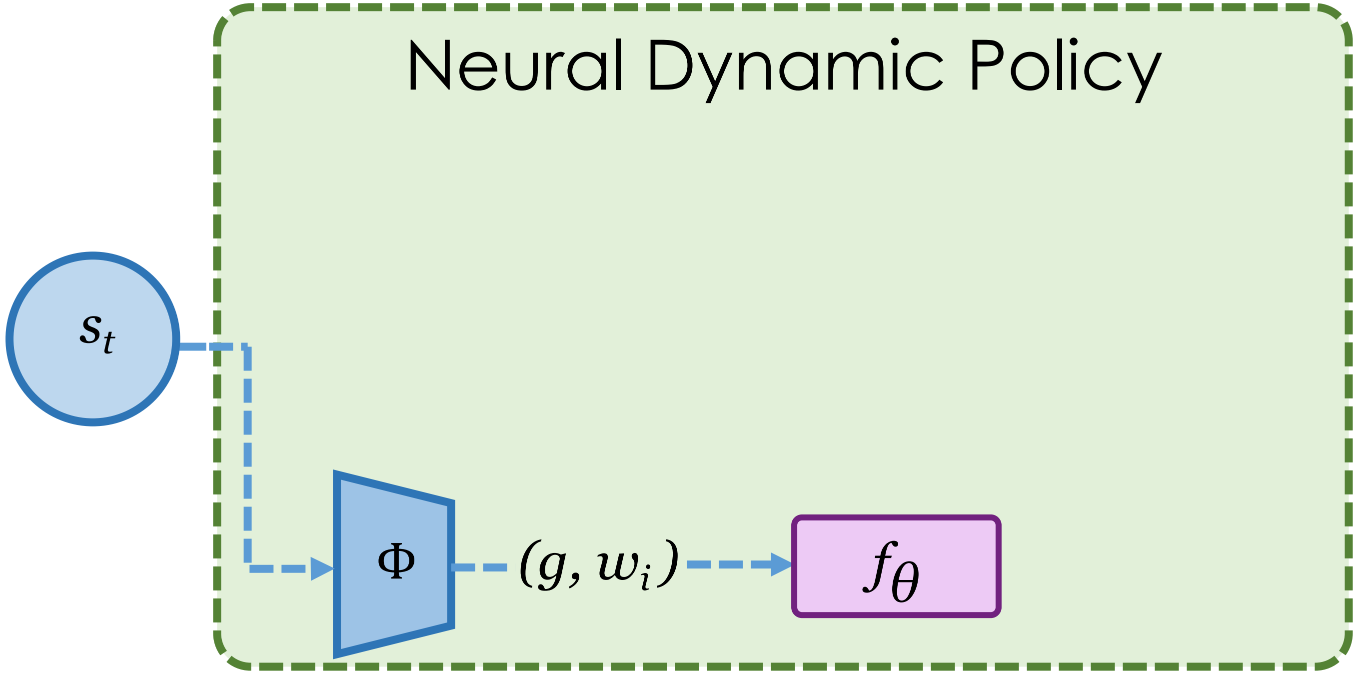
$$\boxed{\mathbf{a} = -\alpha(\mathbf{y} - g) - \beta\dot{\mathbf{y}} + f}$$

2nd Order Differential Equation in y

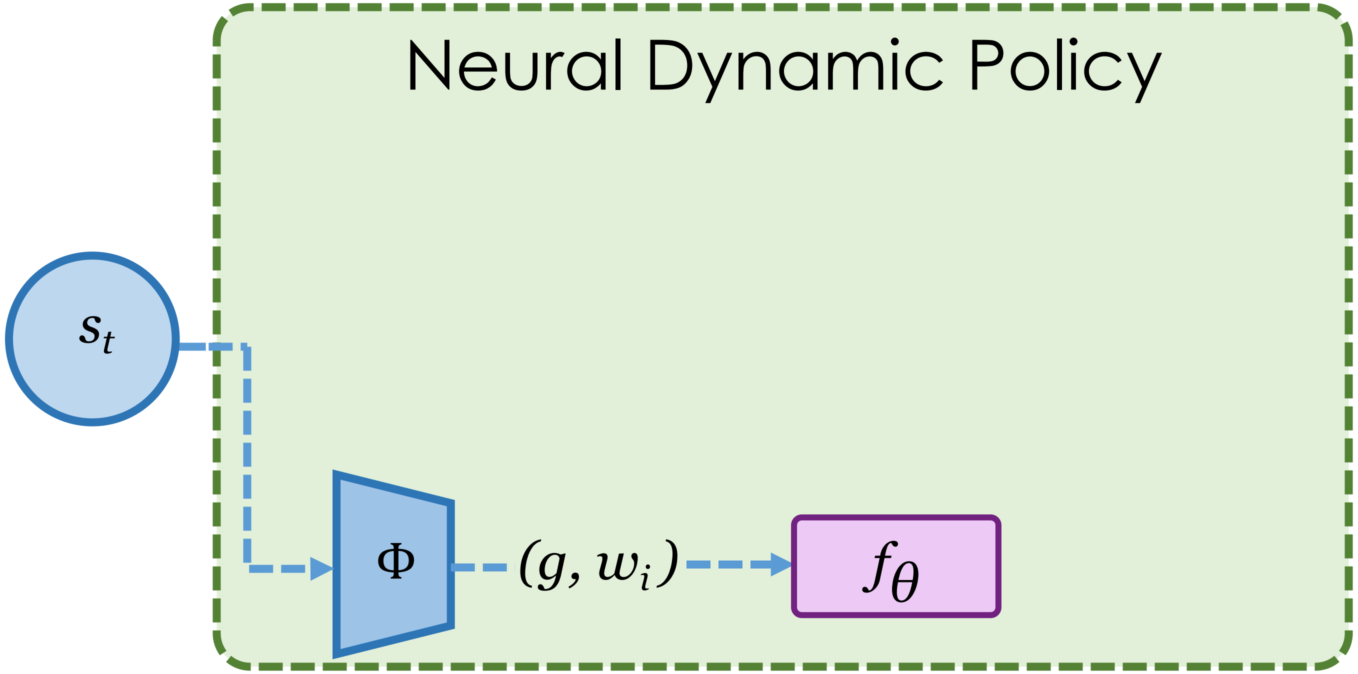


$$\mathbf{a} = -\alpha(\mathbf{y} - g) - \beta\dot{\mathbf{y}} + \boxed{f}$$

Forcing function

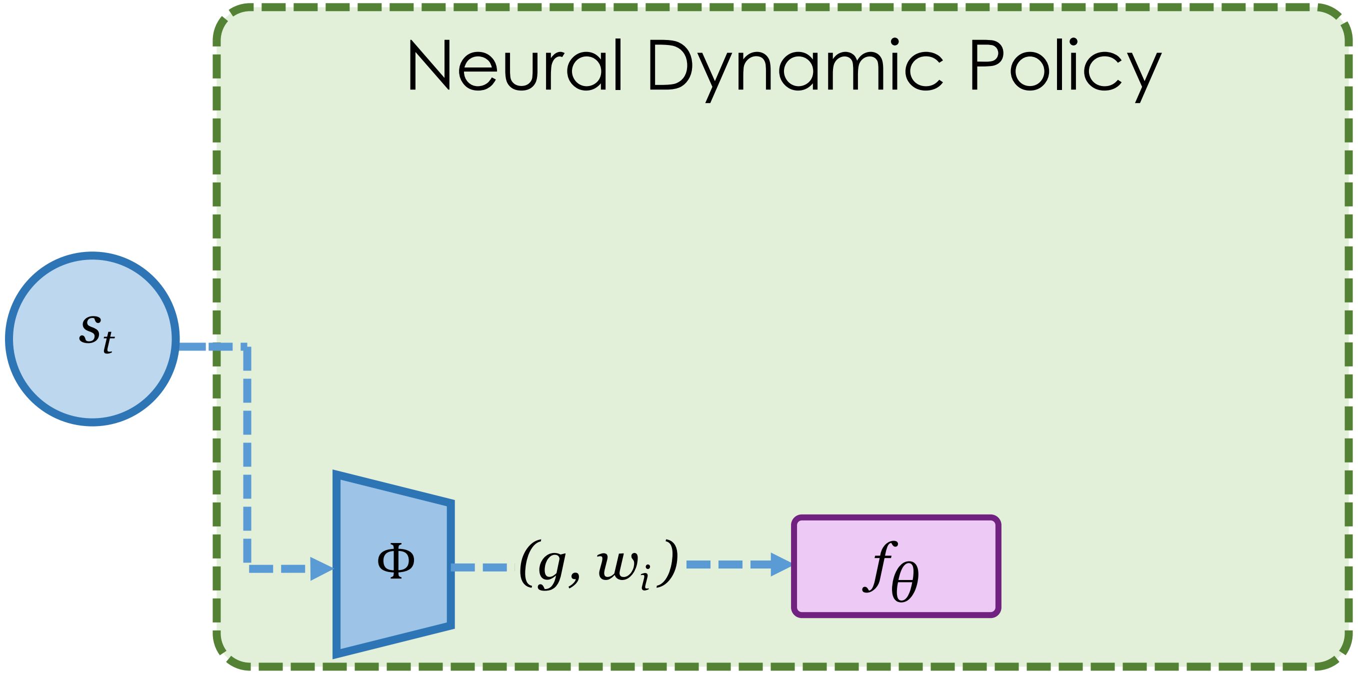


$$f(x) = \frac{\sum_{i=1}^{N} \Psi_i(x) w_i}{\sum_{i=1}^{N} \Psi_i(x)} x(g - y_0)$$



$$f(x) = \frac{\sum_{i=1}^{N} \boxed{\Psi_i}(x) w_i}{\sum_{i=1}^{N} \Psi_i(x)} x(g - y_0)$$

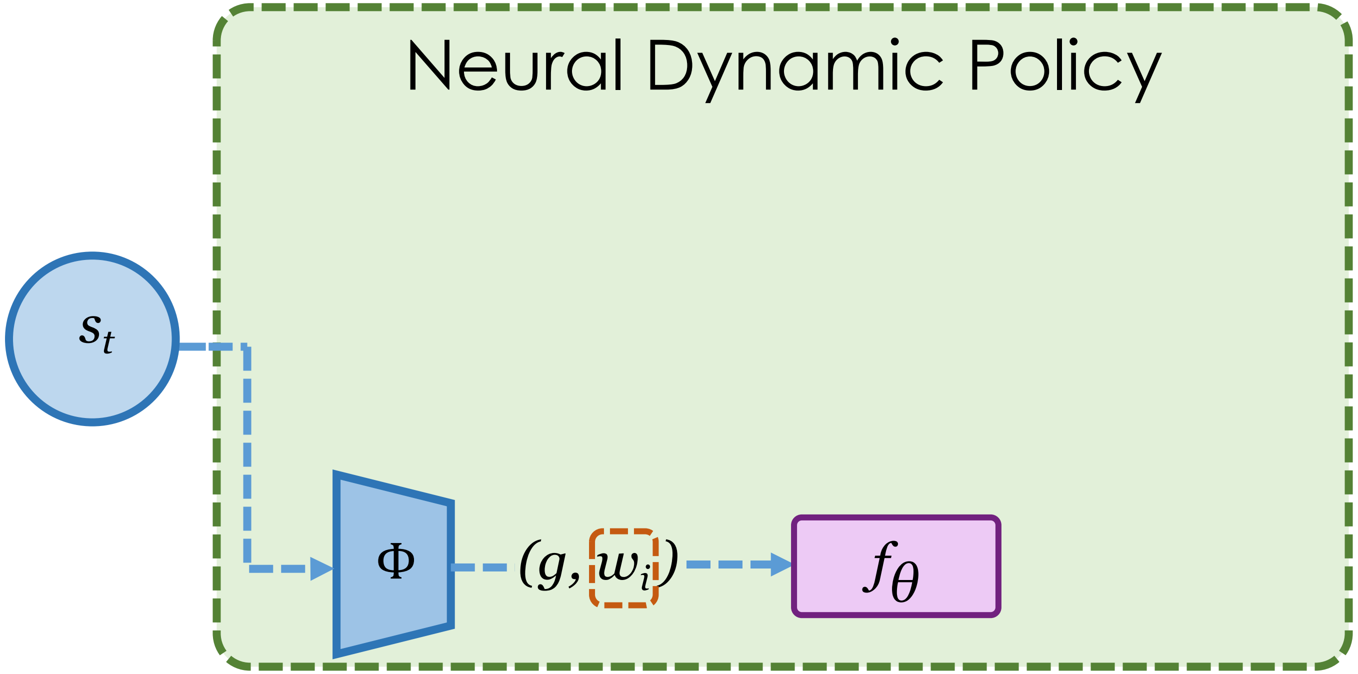
Radial Basis Function



$$f(x) = \frac{\sum_{i=1}^{N} \boxed{\Psi_i}(x) w_i}{\sum_{i=1}^{N} \Psi_i(x)} x(g - y_0)$$

$$\boxed{\Psi_i(x) = \exp\left(-\frac{1}{2\sigma_i^2}(x - c_i)^2\right),}$$

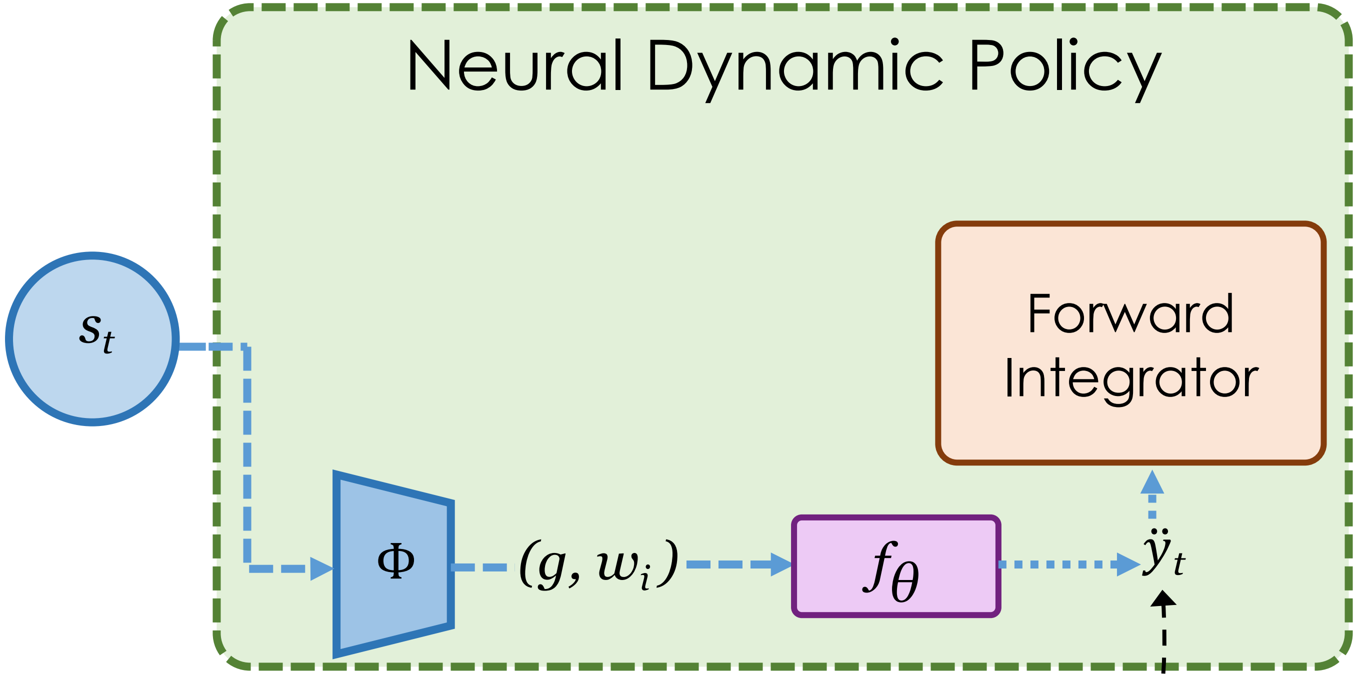
Radial Basis Function



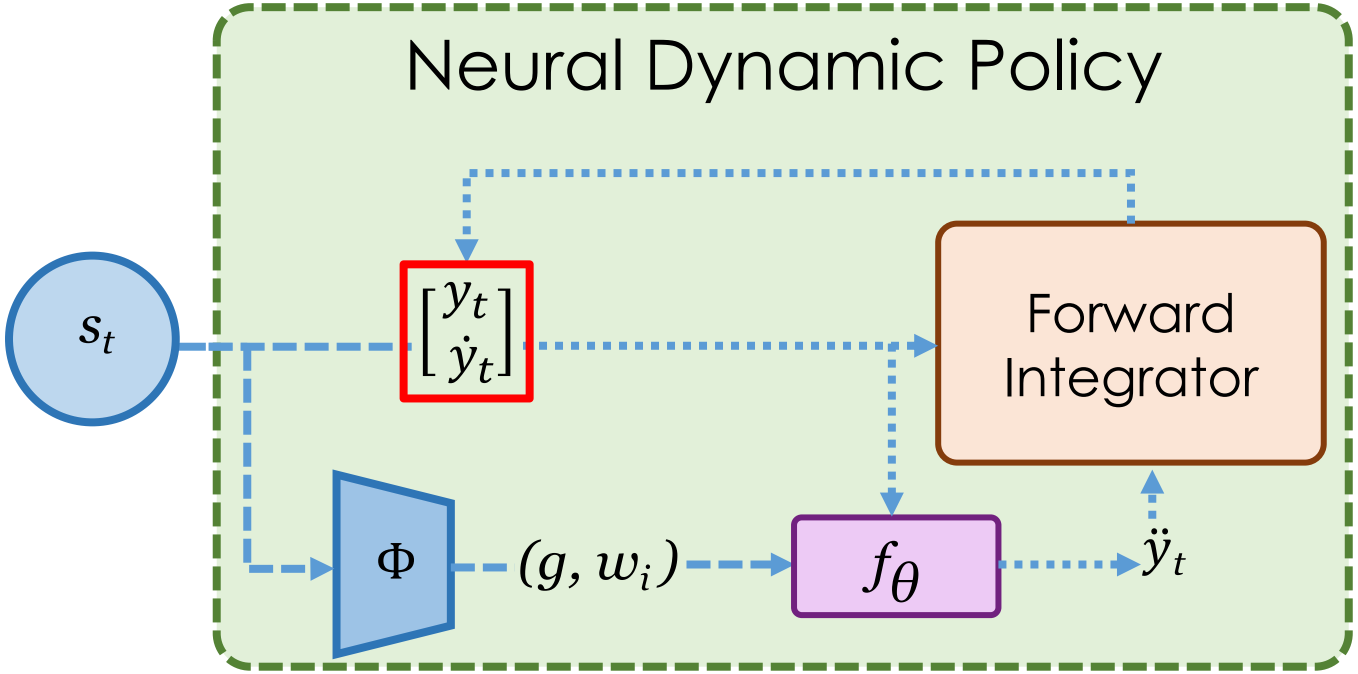
$$f(x) = \frac{\sum_{i=1}^{N} \Psi_i(x) \boxed{w_i}}{\sum_{i=1}^{N} \Psi_i(x)} x(g - y_0)$$

$$\Psi_i(x) = \exp\left(-\frac{1}{2\sigma_i^2}(x - c_i)^2\right),$$

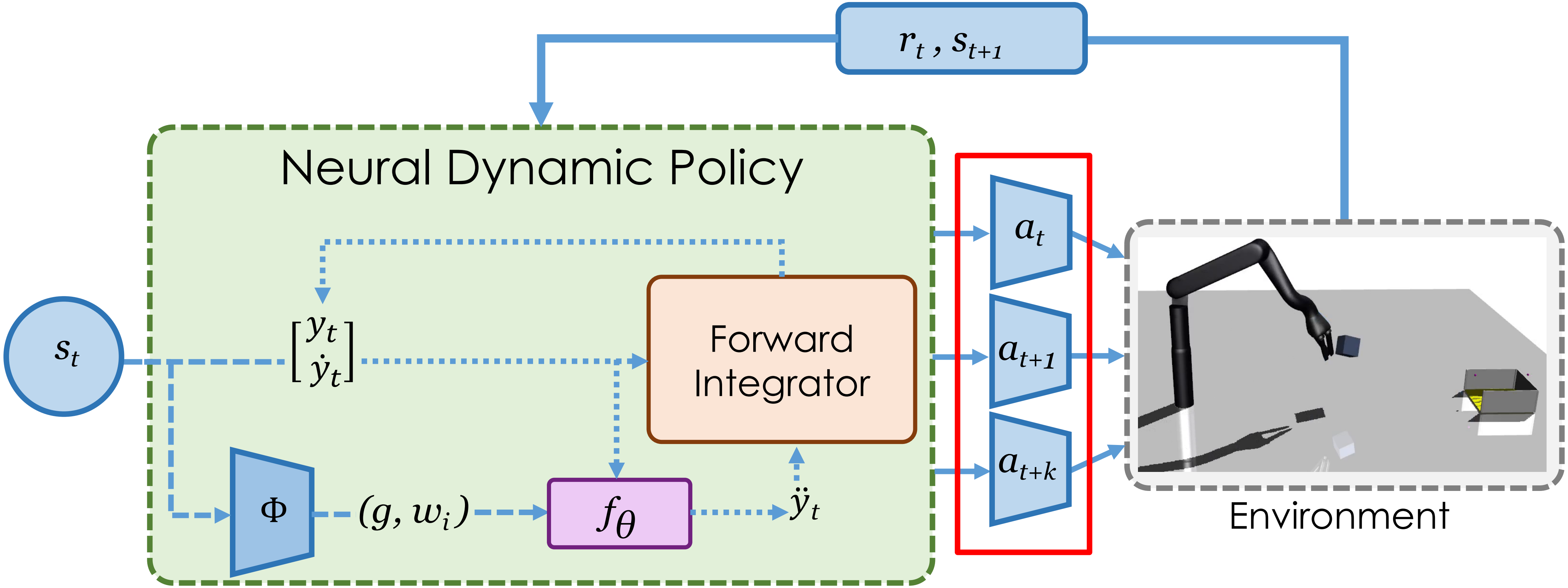
Weight of RBF



$$\boxed{\mathbf{a} = -\alpha(\mathbf{y} - g) - \beta\dot{\mathbf{y}} + f}$$

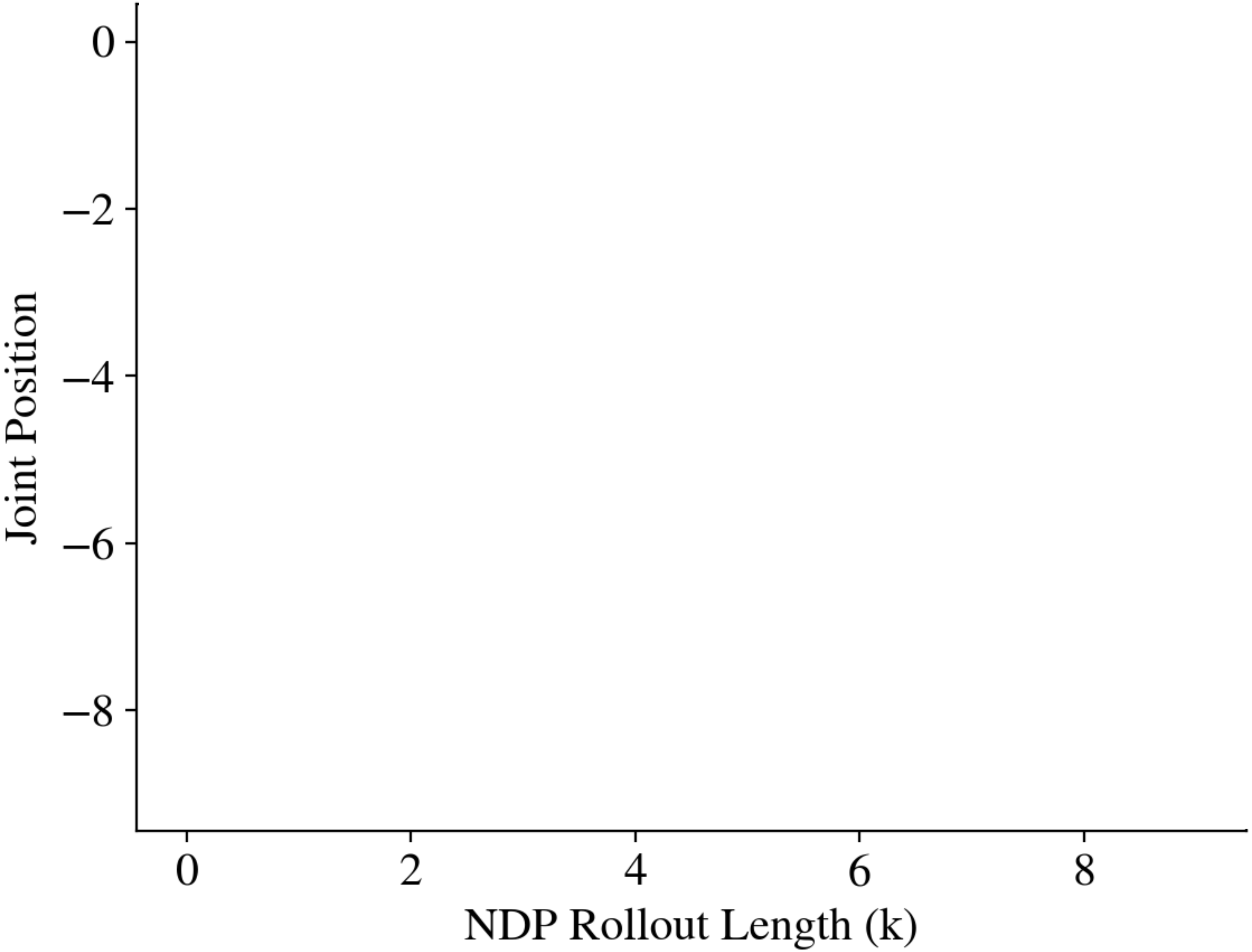


$$\mathbf{a} = -\alpha(\mathbf{y} - g) - \beta\dot{\mathbf{y}} + f$$

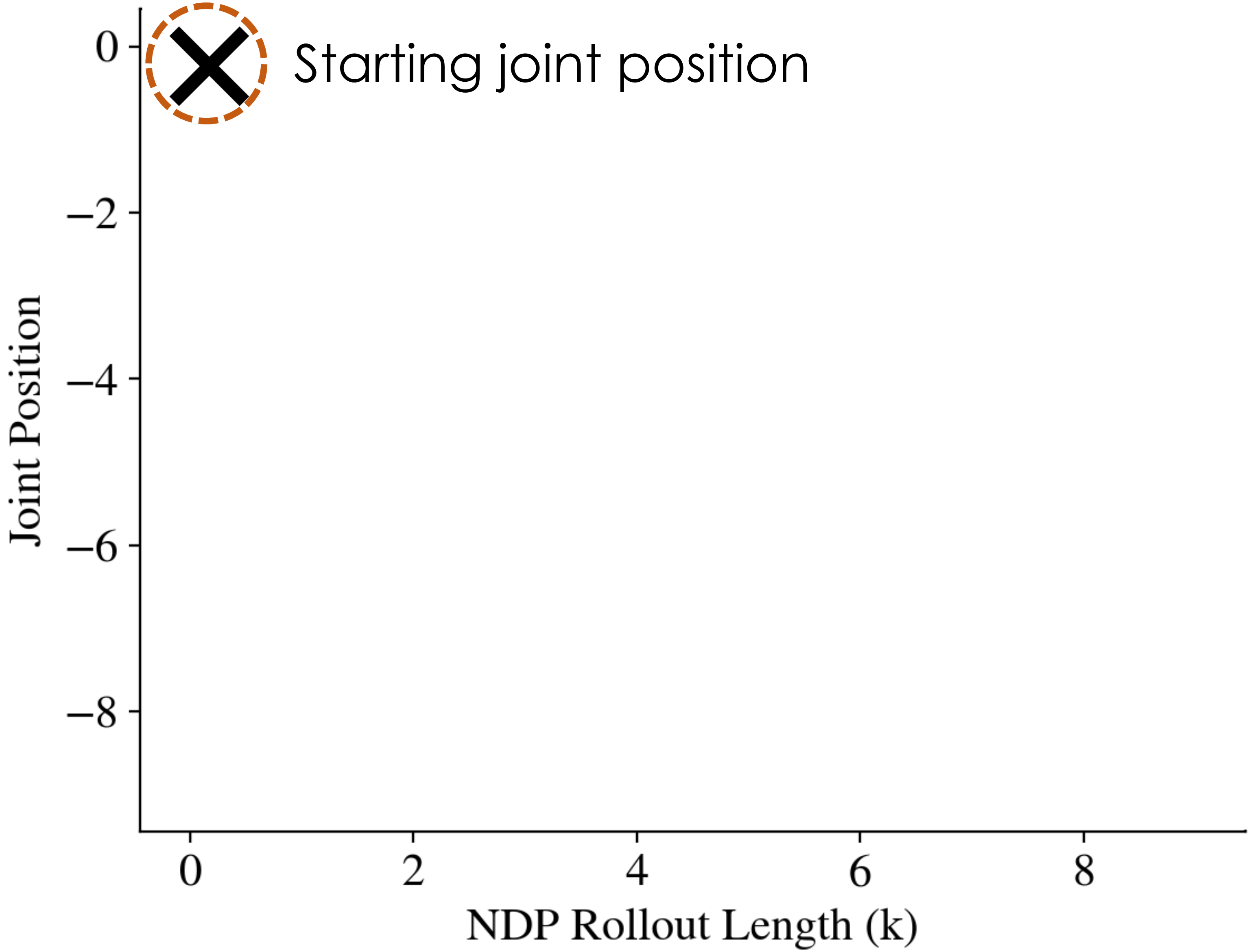


$$\mathbf{a} = -\alpha(\mathbf{y} - g) - \beta\dot{\mathbf{y}} + f$$

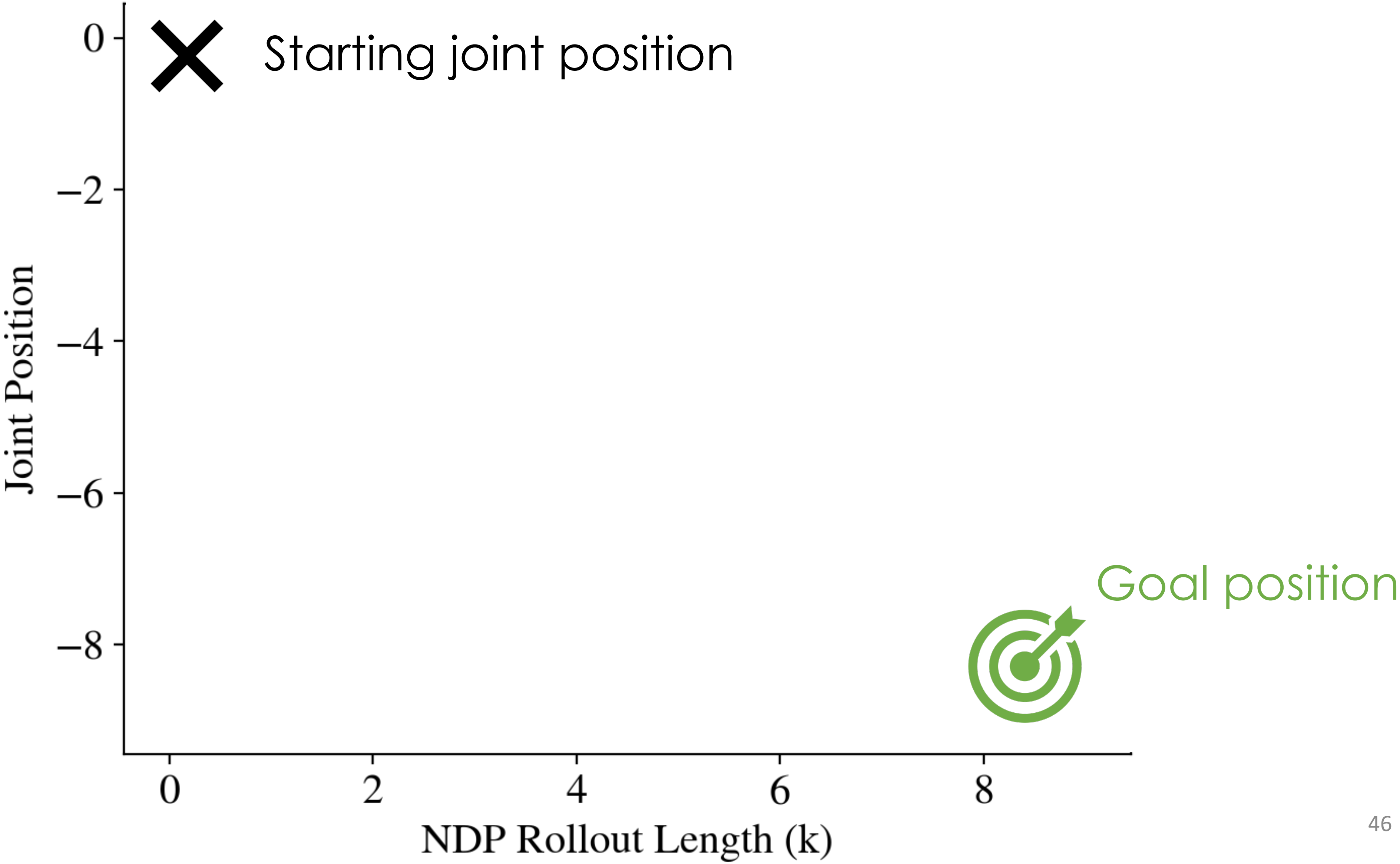
NDPs Intuition



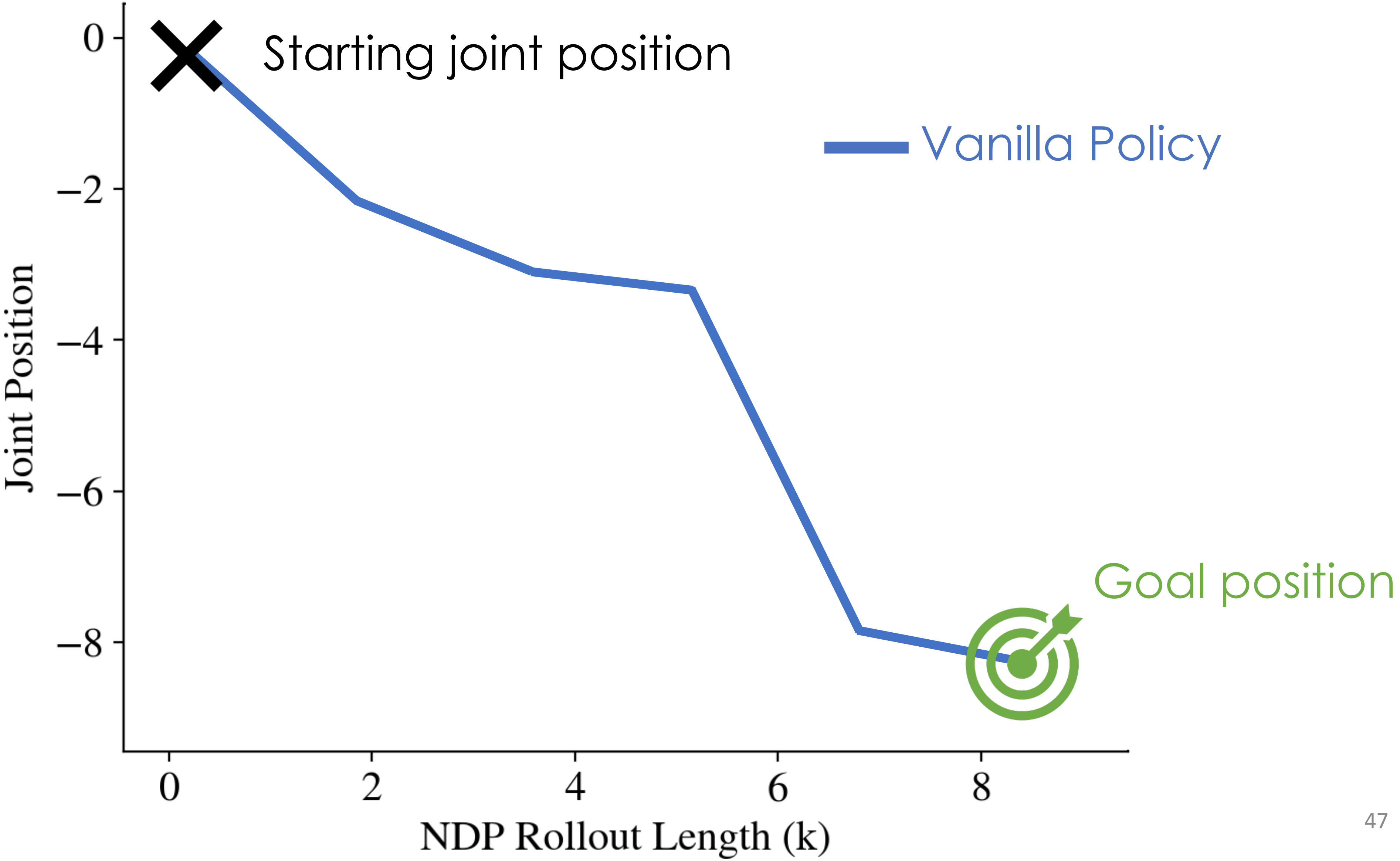
NDPs Intuition



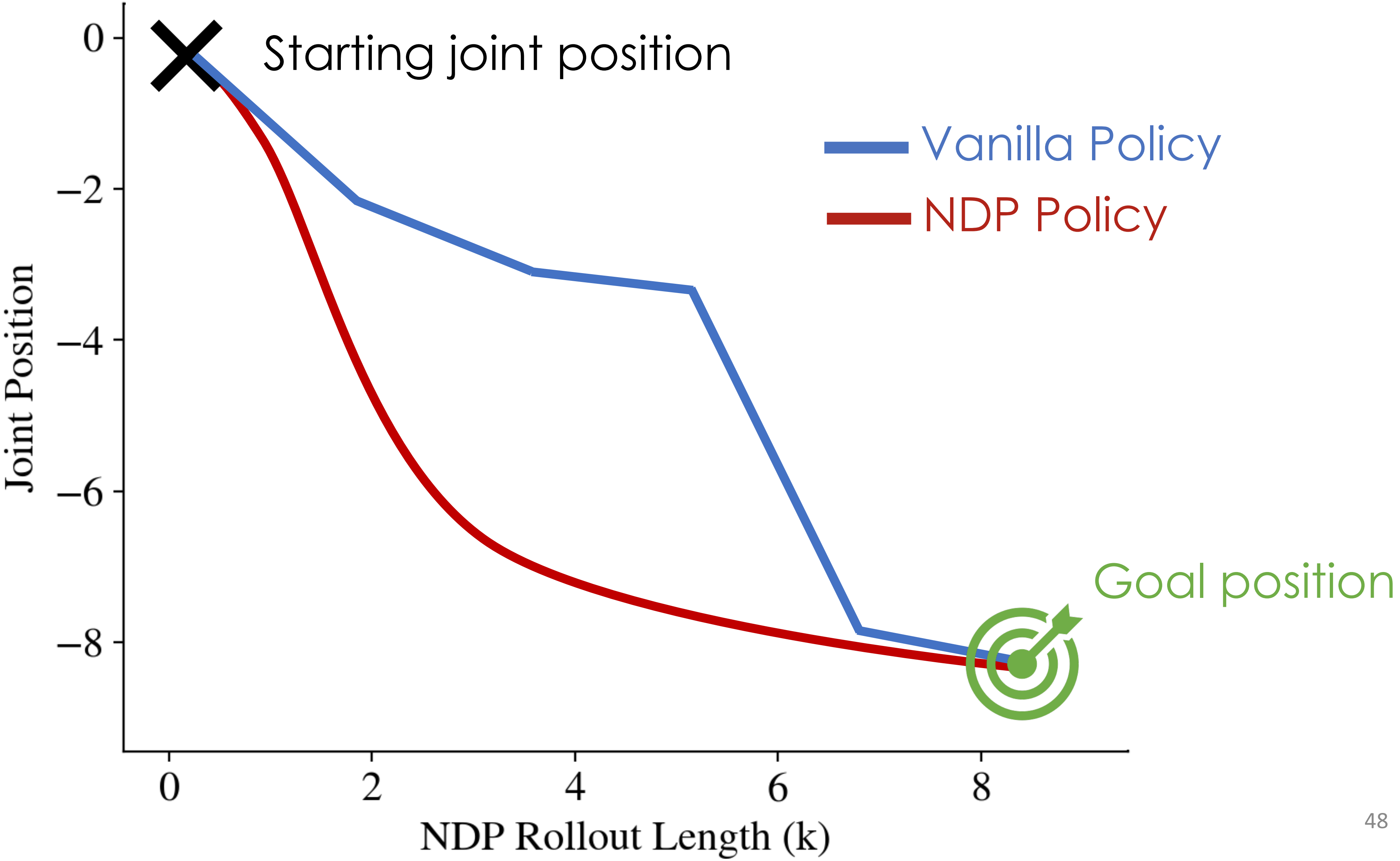
NDPs Intuition



NDPs Intuition



NDPs Intuition



NDPs Intuition

NDPs show a much smoother convergence to the goal

RL Algorithm

Algorithm 1 Training NDPs for RL

Require: Policy π, k NDP rollout length, g low-level inverse controller

for $1, 2, \ldots$ episodes **do**

  for $t = 0, k, \ldots,$ until end of episode **do**

    $w, g = \Phi(s_t)$

    Robot y_t, $\dot{y}_t$ from s_t (pos, vel)

    for $m = 1, \ldots, M$ (integration steps) **do**

      Estimate $\dot{x}_m$ via (2) and update x_m

      Estimate $\ddot{y}_{t+m}, \dot{y}_{t+m}, y_{t+m}$ via (4), (5)

      $a_{t+n} = g(y_{t+m}, y_{t+m-1})$

      Apply action a_{t+n} to get s_{t+n+1}

      Store transition (s, a, s', r)

    end for

    Compute Policy gradient ∇_θ

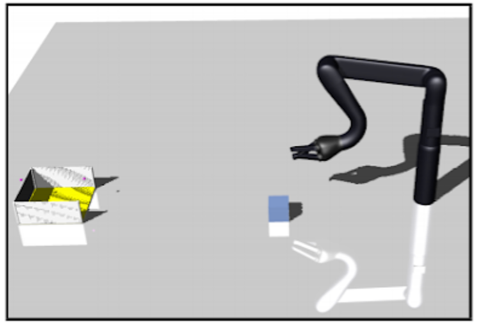
    $\theta \leftarrow \theta + \eta \nabla_\theta J$

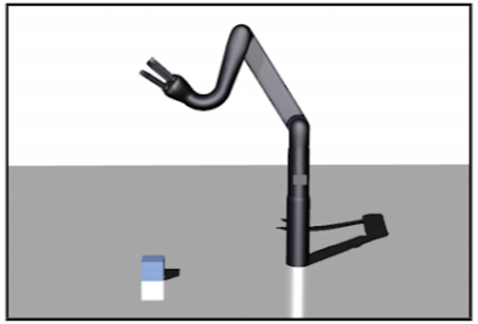
  end for

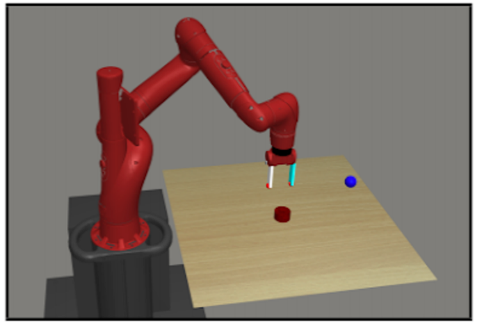
end for

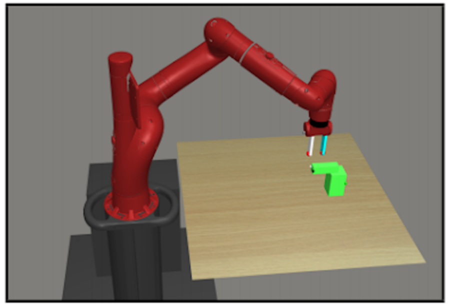
Results

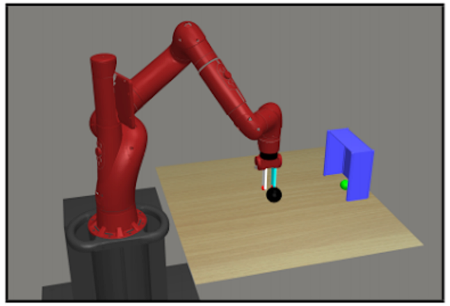
Reinforcement Learning

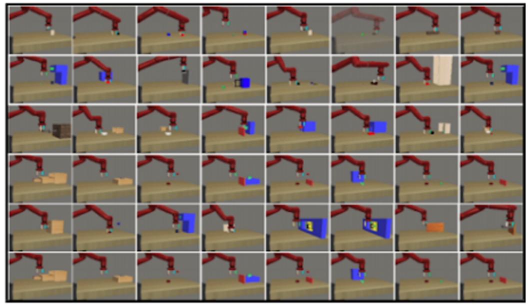

(a) Throwing


(b) Picking

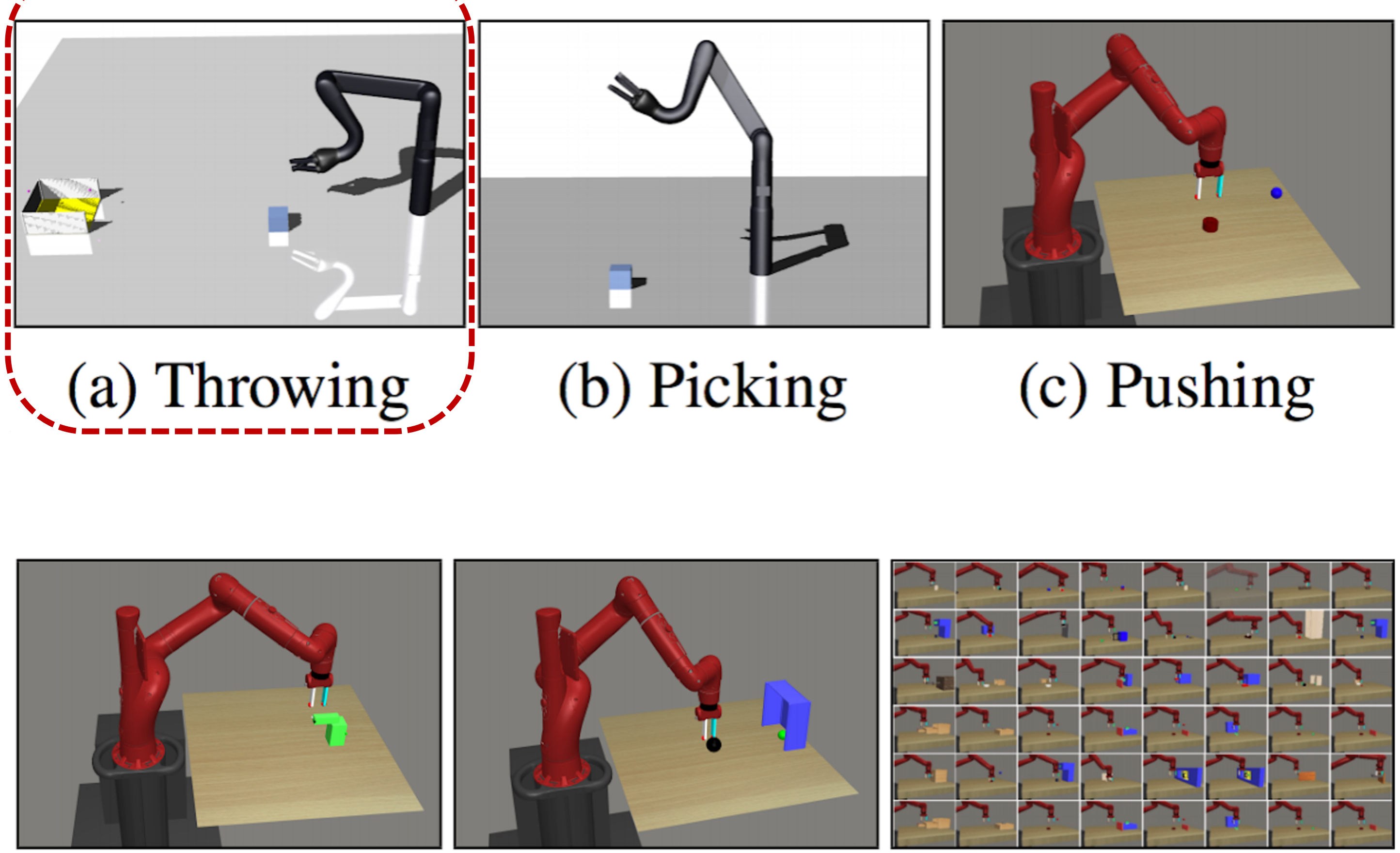

(c) Pushing


(d) Faucet Open


(e) Soccer


(f) 50 Tasks

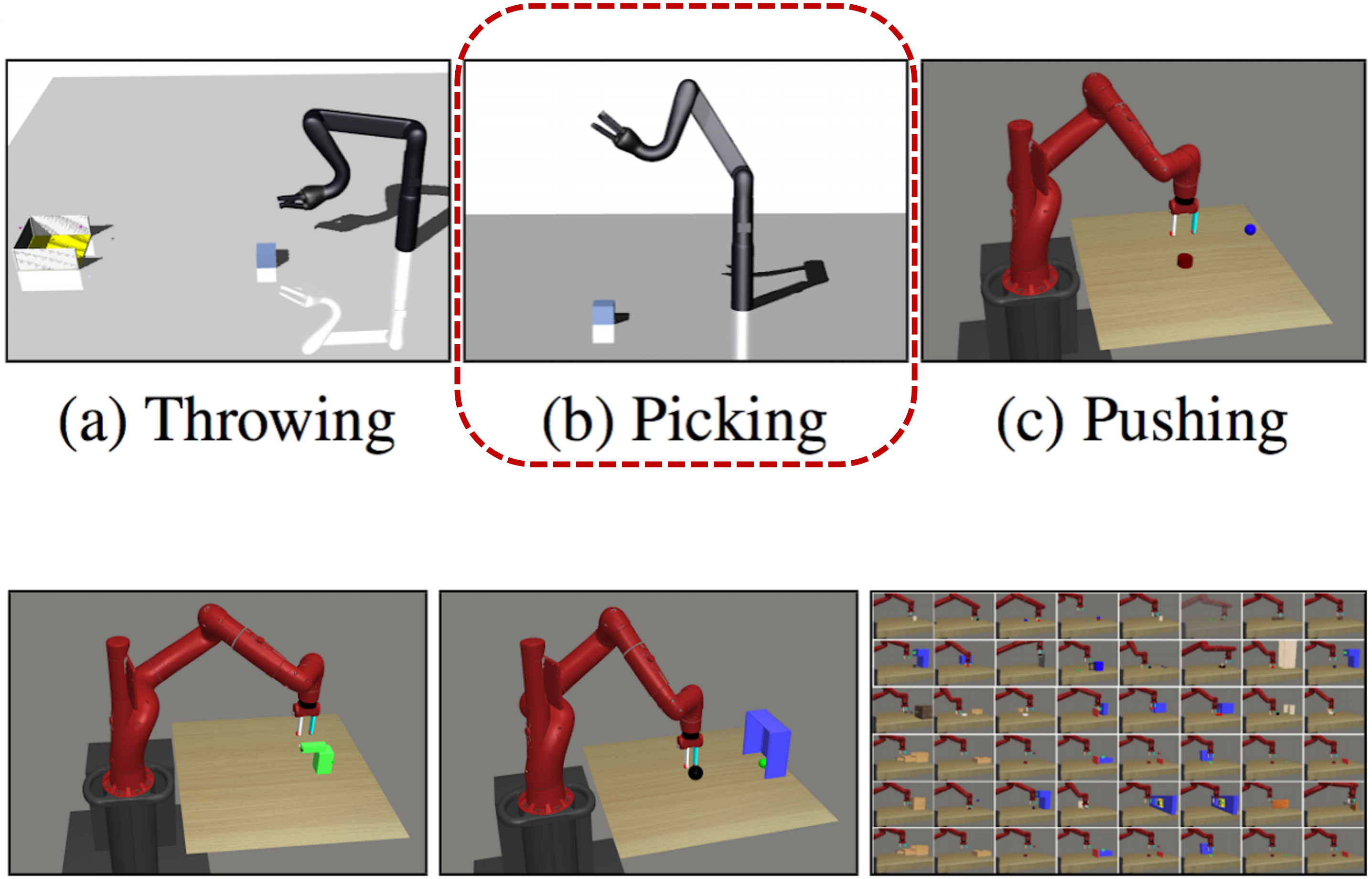
Reinforcement Learning



(a) Throwing (b) Picking (c) Pushing

(d) Faucet Open (e) Soccer (f) 50 Tasks

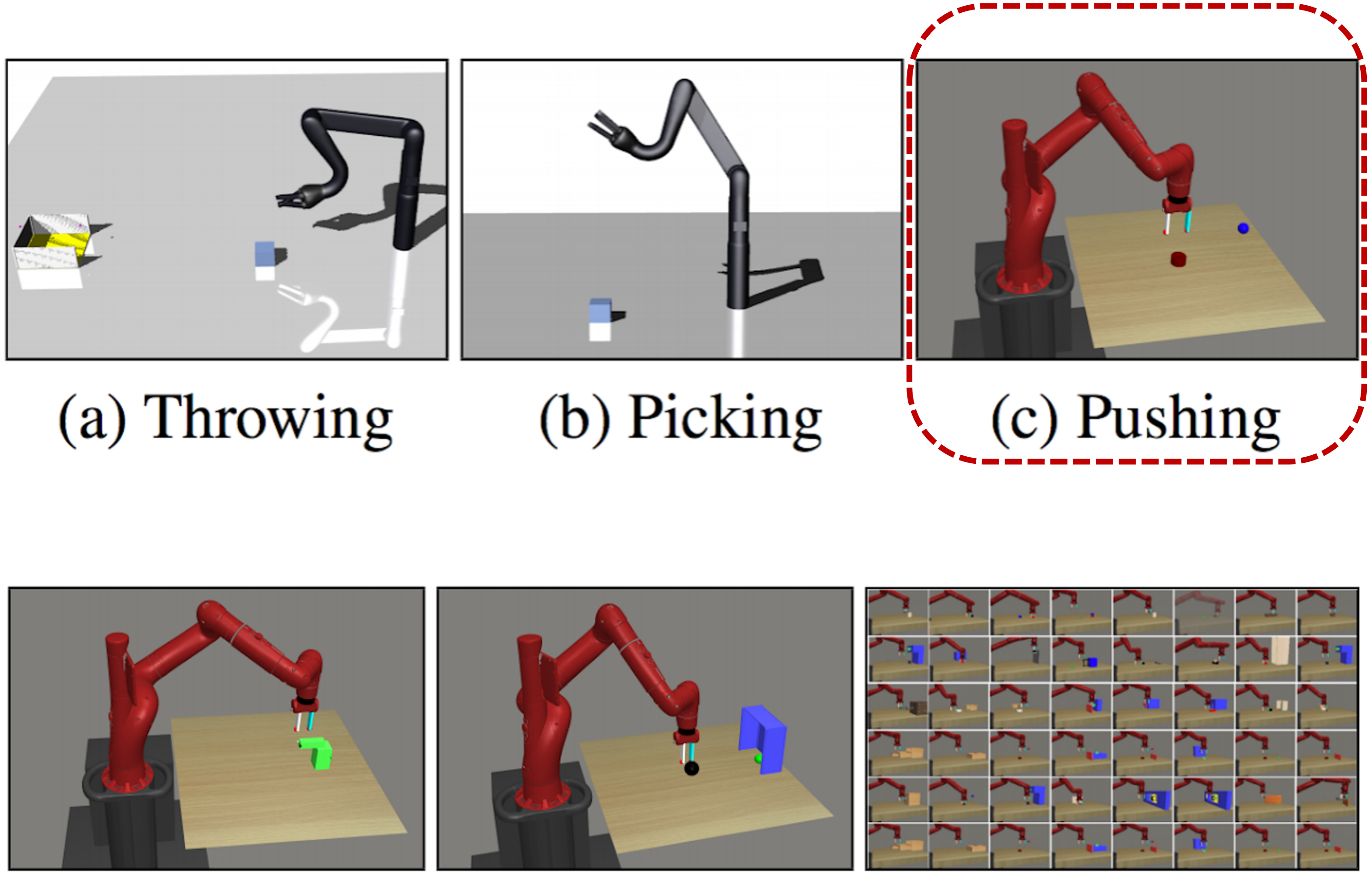
Reinforcement Learning



(a) Throwing (b) Picking (c) Pushing

(d) Faucet Open (e) Soccer (f) 50 Tasks

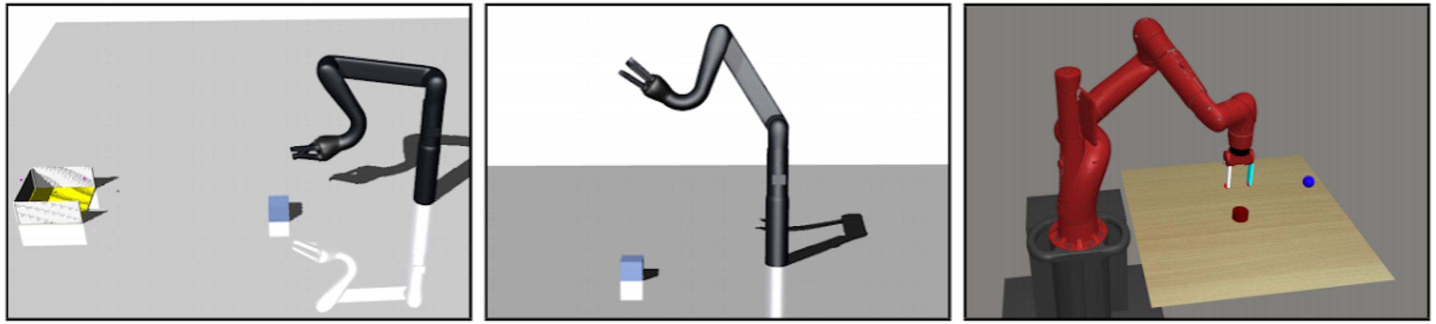
Reinforcement Learning



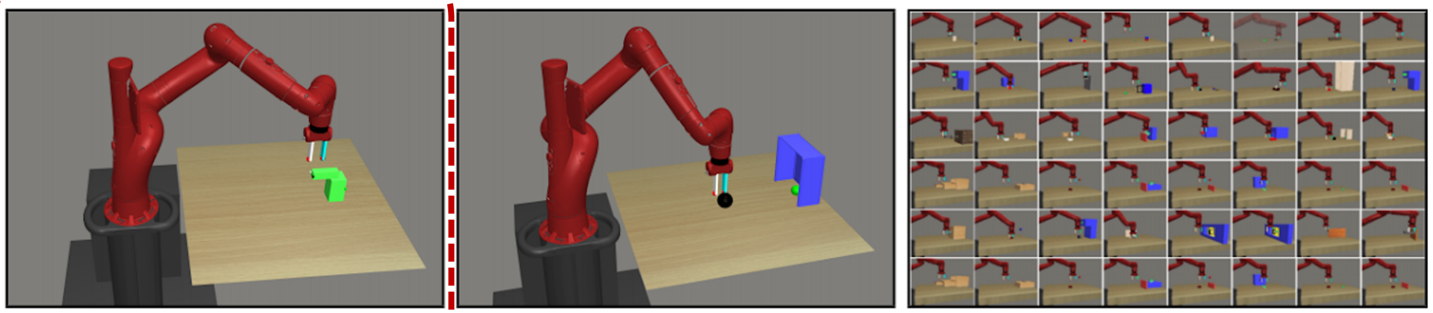
(a) Throwing (b) Picking (c) Pushing

(d) Faucet Open (e) Soccer (f) 50 Tasks

Reinforcement Learning

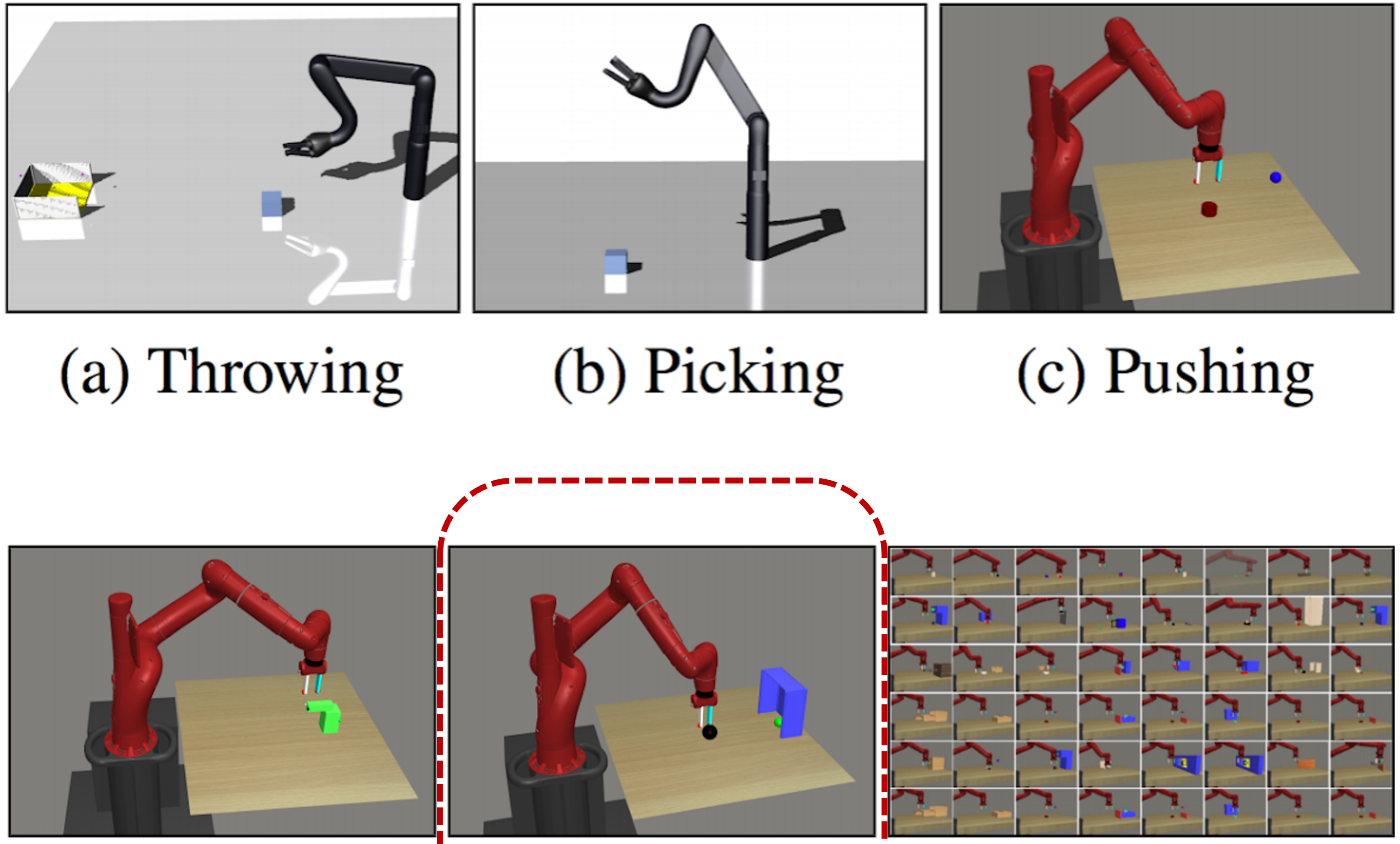


(a) Throwing (b) Picking (c) Pushing



(d) Faucet Open (e) Soccer (f) 50 Tasks

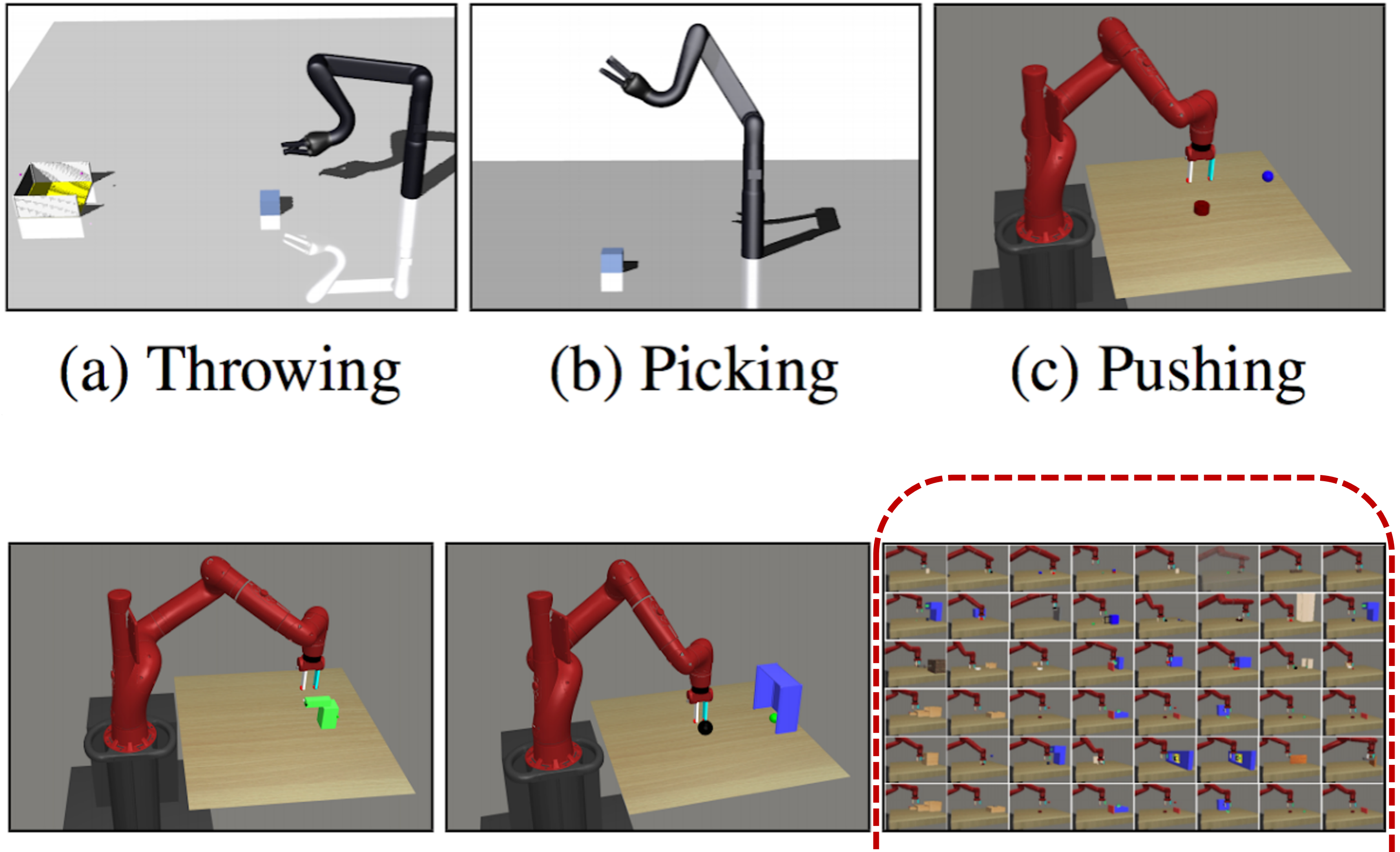
Reinforcement Learning



(a) Throwing (b) Picking (c) Pushing

(d) Faucet Open (e) Soccer (f) 50 Tasks

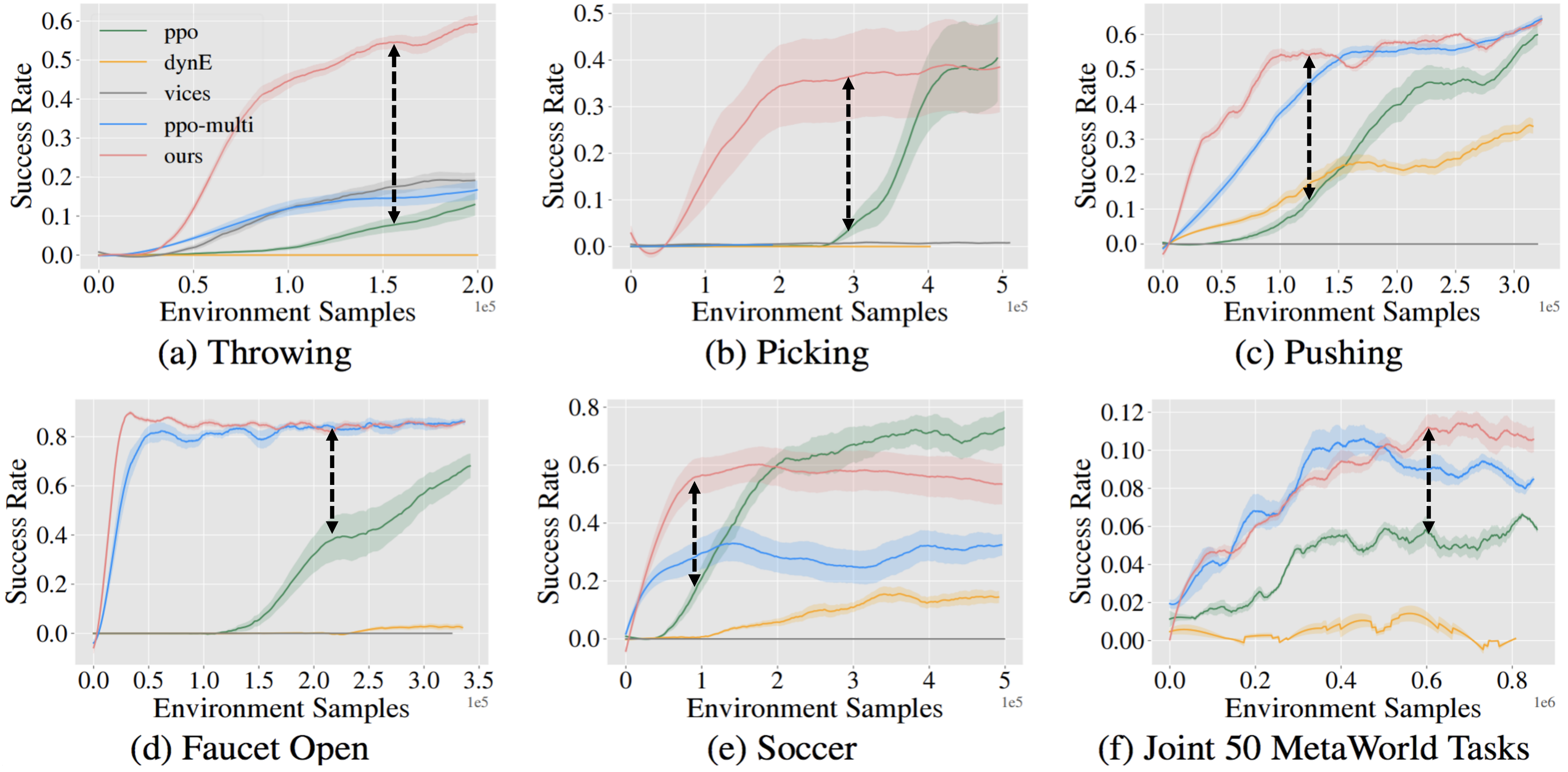
Reinforcement Learning



(a) Throwing (b) Picking (c) Pushing

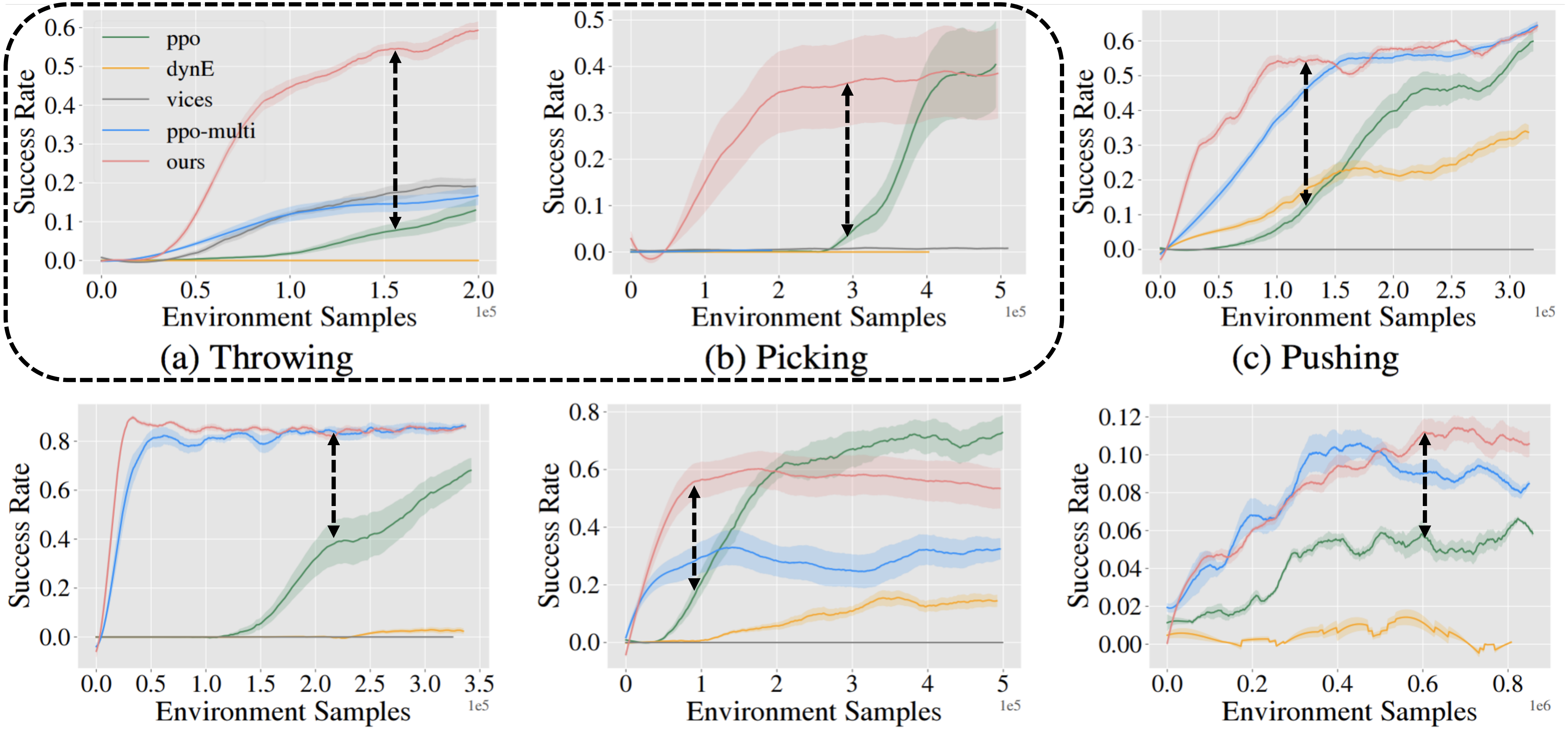
(d) Faucet Open (e) Soccer (f) 50 Tasks

RL: Results



ppo
dynE
vices
ppo-multi
ours
Success Rate
Environment Samples
1e5
1e6
(a) Throwing
(b) Picking
(c) Pushing
(d) Faucet Open
(e) Soccer
(f) Joint 50 MetaWorld Tasks

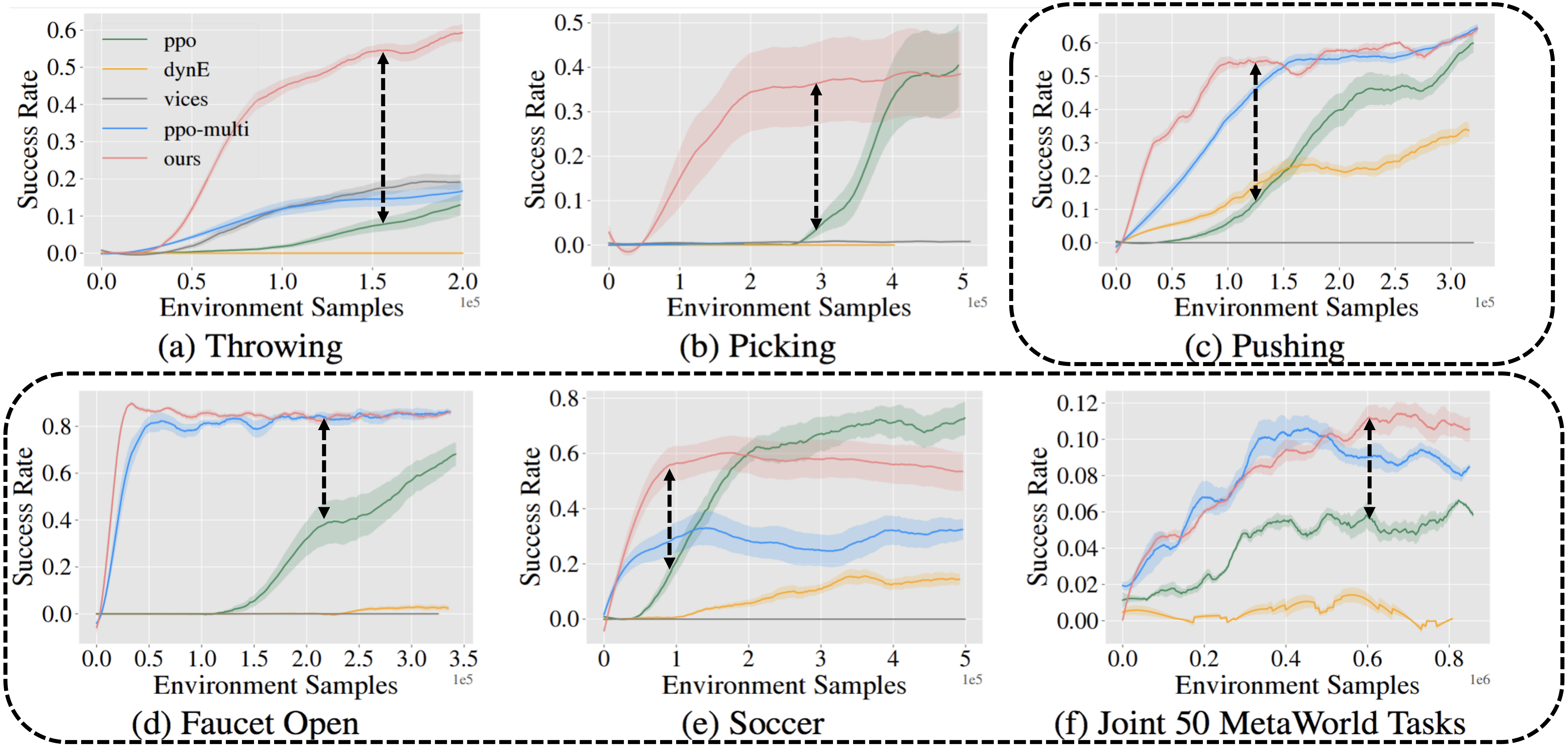
RL: Results



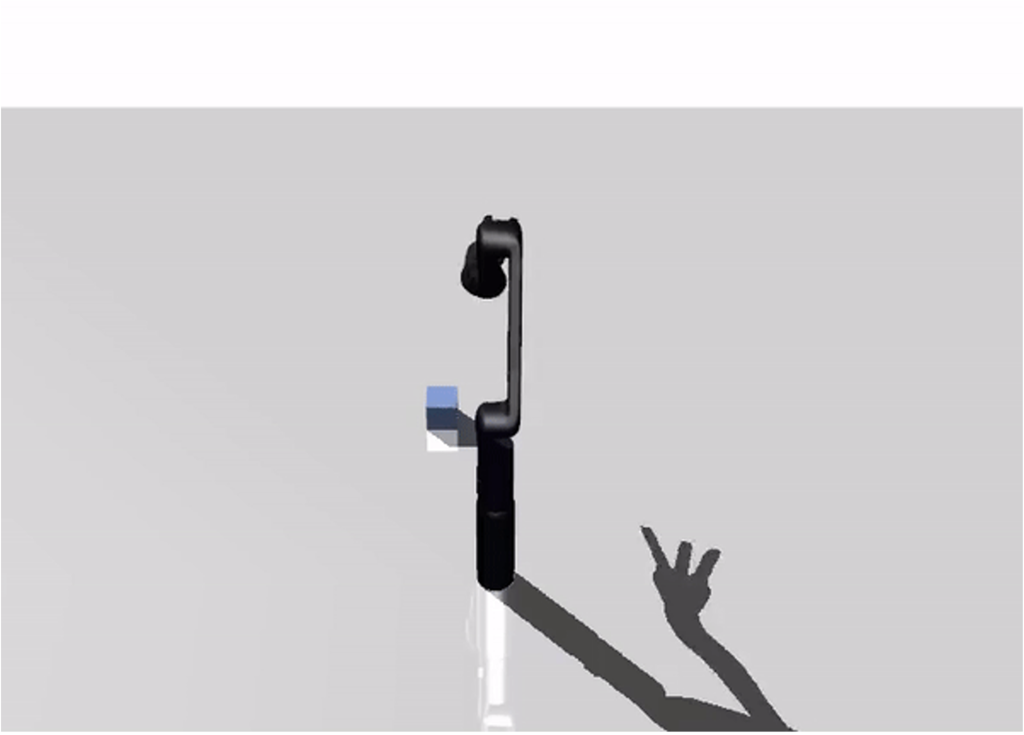
(a) Throwing (b) Picking (c) Pushing

(d) Faucet Open (e) Soccer (f) Joint 50 MetaWorld Tasks

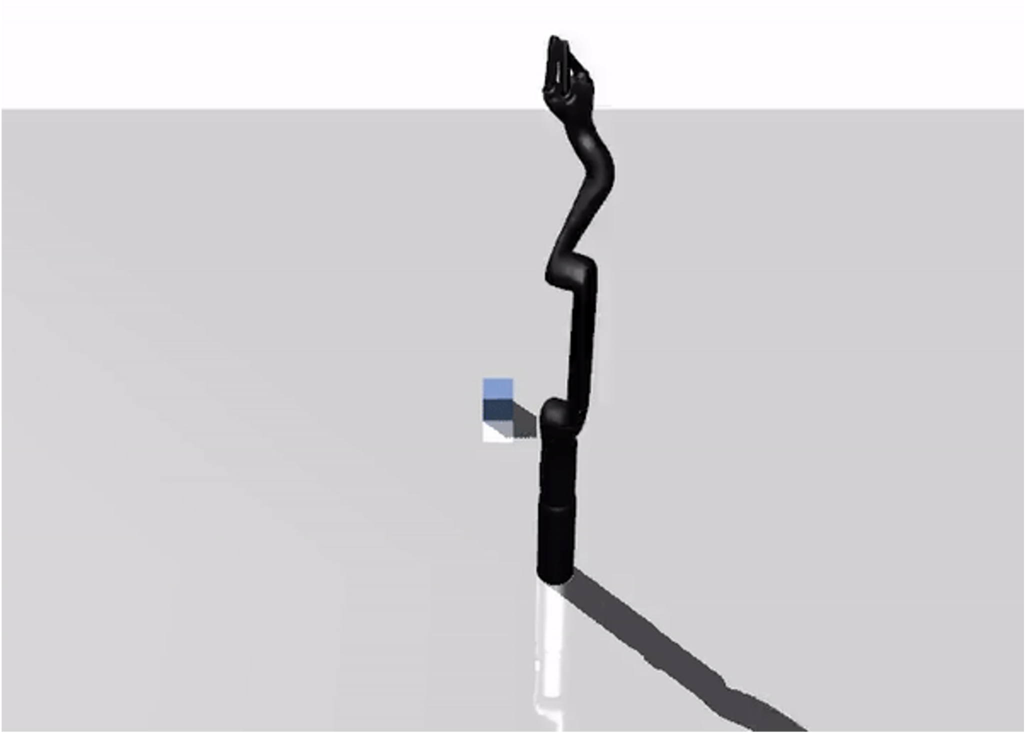
RL: Results


ppo
dynE
vices
ppo-multi
ours
Success Rate
Environment Samples
1e5
1e6
(a) Throwing
(b) Picking
(c) Pushing
(d) Faucet Open
(e) Soccer
(f) Joint 50 MetaWorld Tasks

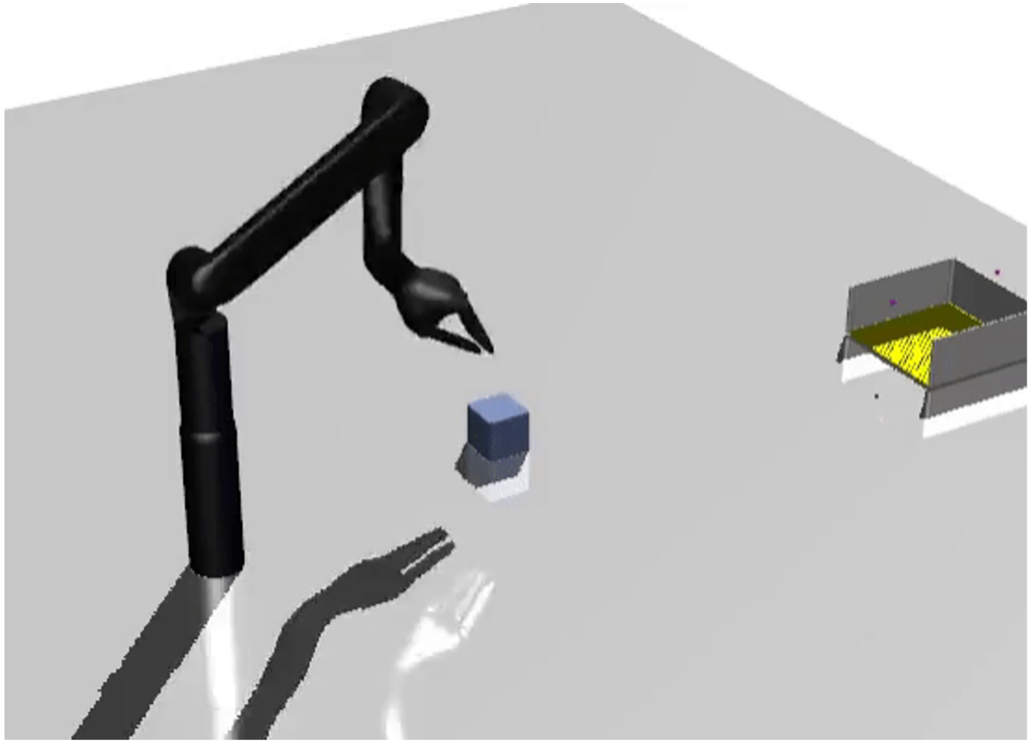
NDP (ours)



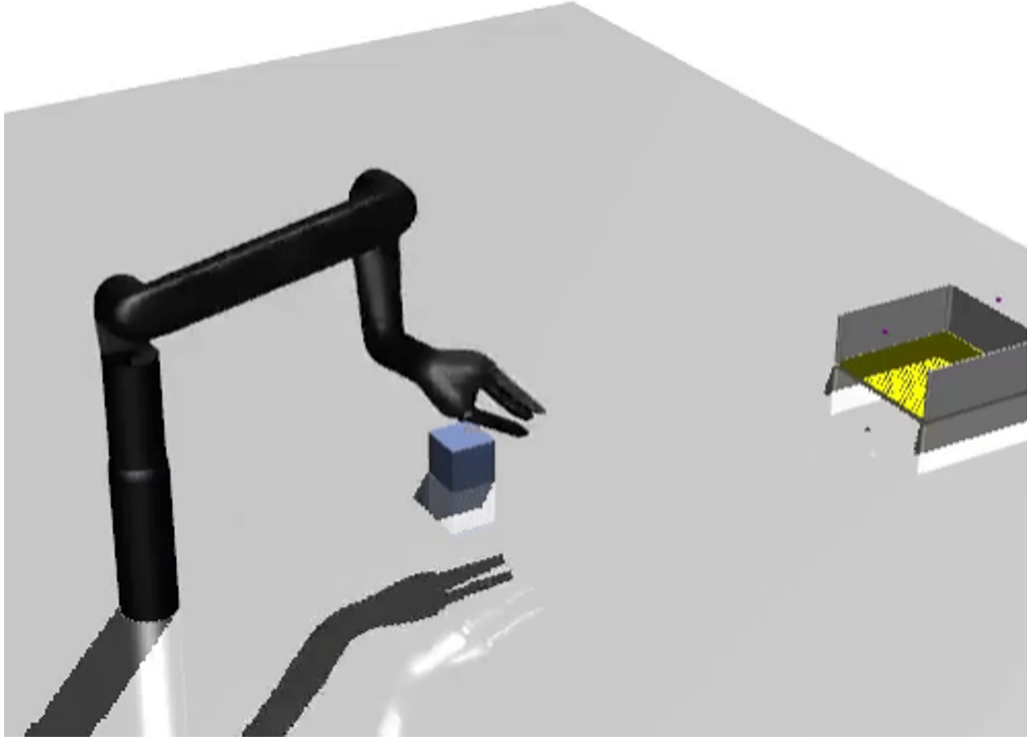
PPO-Multi (Baseline)



NDP (ours)

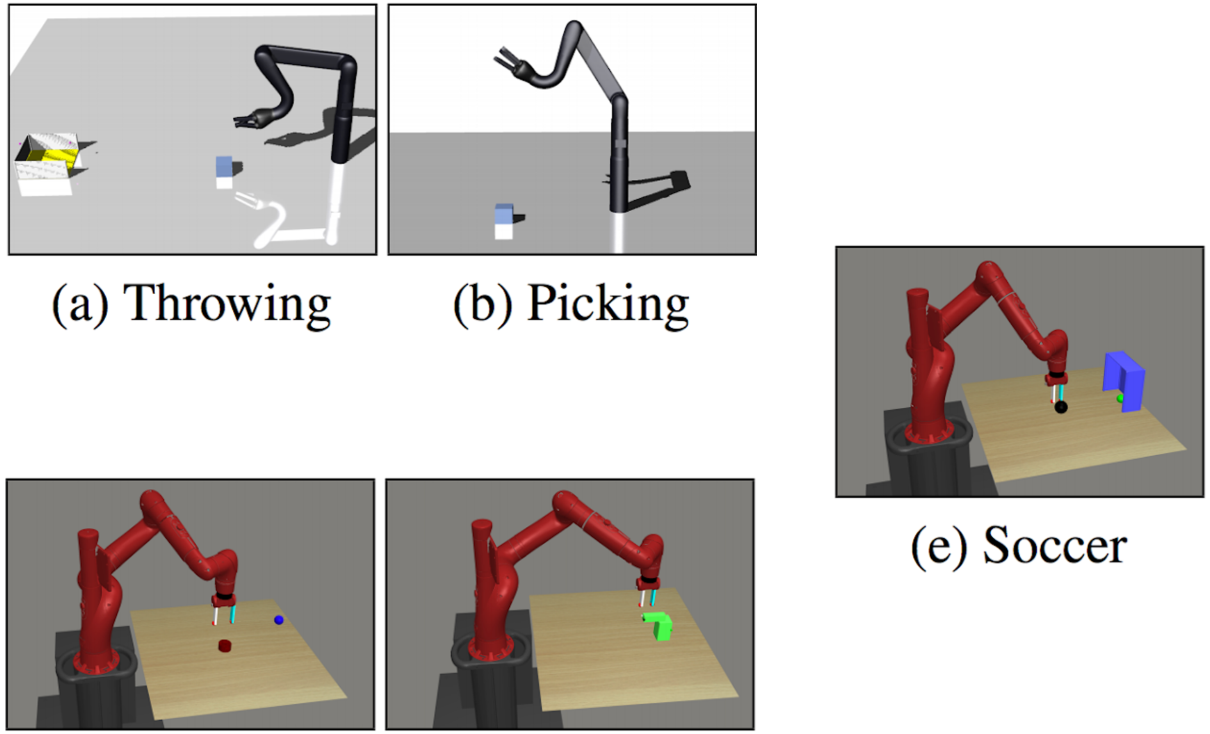


PPO-Multi (Baseline)



Imitation Learning

Method	NN	**NDP (ours)**
Throw	0.528 ± 0.262	$\mathbf{0.642 \pm 0.246}$
Pick	$\mathbf{0.672 \pm 0.074}$	0.408 ± 0.058
Push	0.002 ± 0.004	$\mathbf{0.208 \pm 0.049}$
Soccer	0.885 ± 0.016	$\mathbf{0.890 \pm 0.010}$
Faucet	0.532 ± 0.231	$\mathbf{0.790 \pm 0.059}$



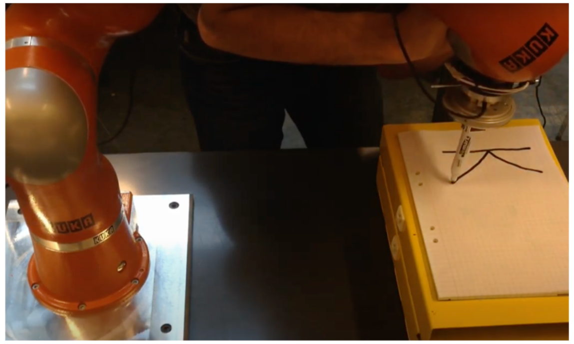
(a) Throwing (b) Picking (c) Pushing (d) Faucet Open (e) Soccer

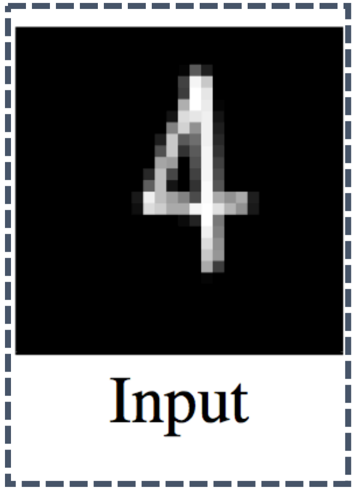
NDPs from Images

Digit Writing

Image of desired digit

Output: end-effector positions

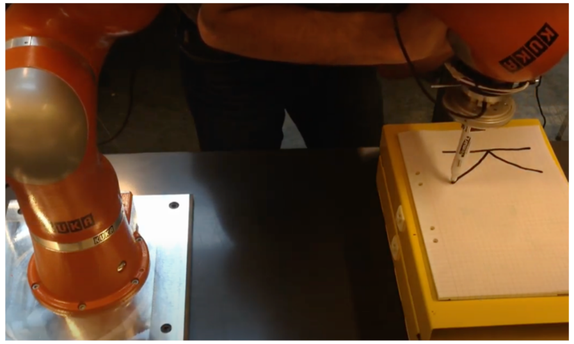


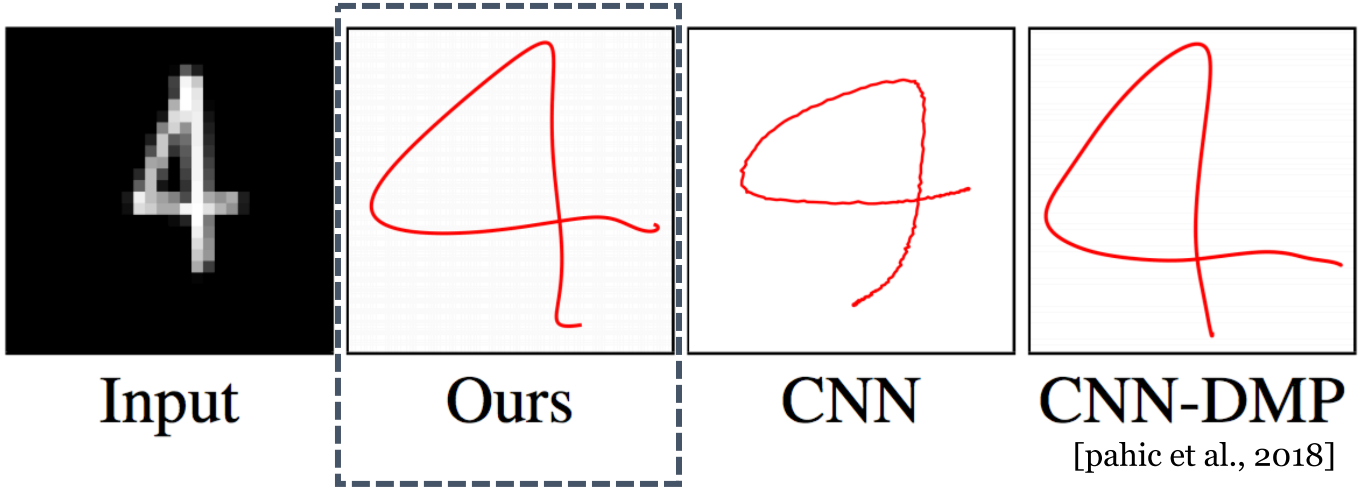


Digit Writing

Image of desired digit

Output: end-effector positions

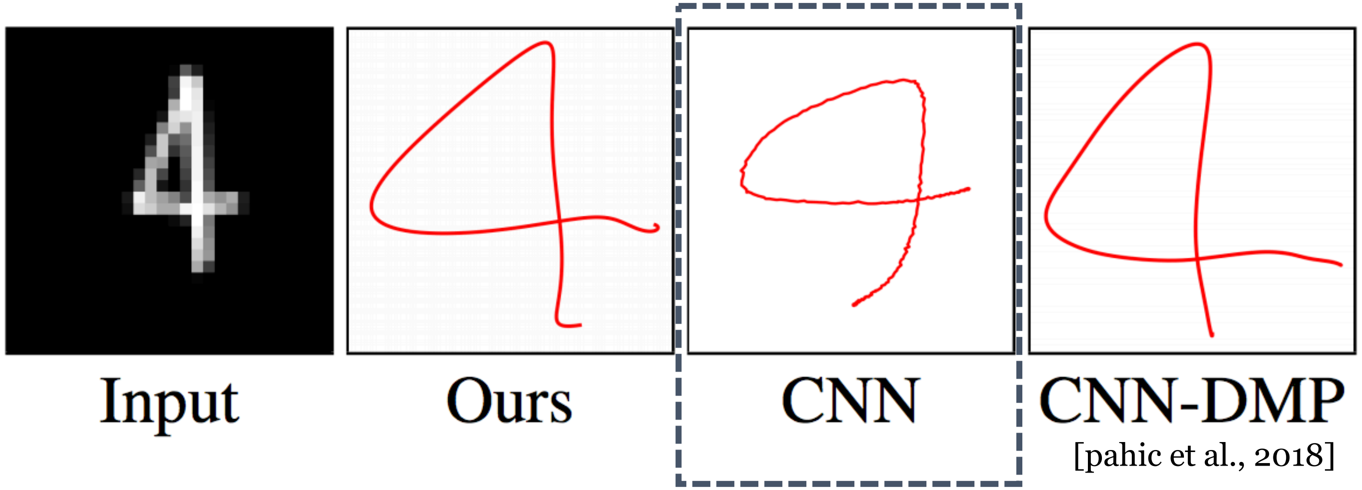


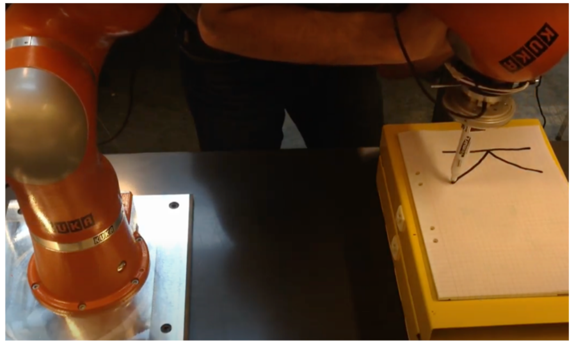


Digit Writing

Image of desired digit

Output: end-effector positions

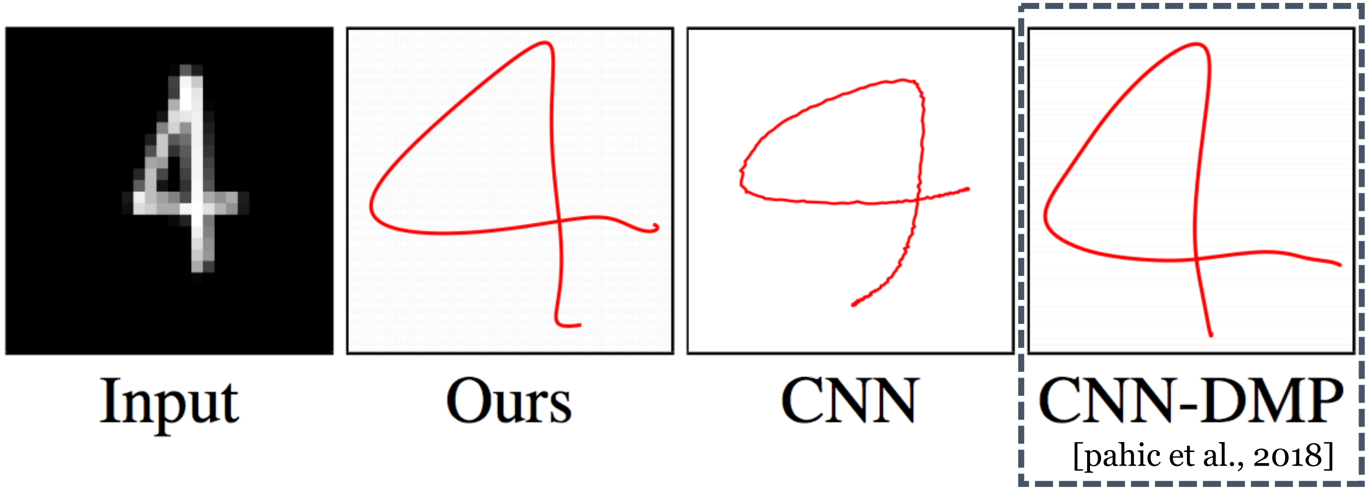


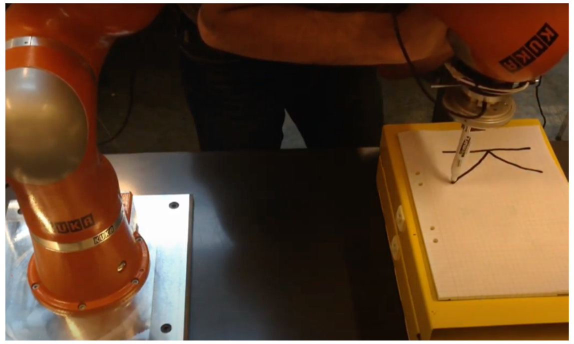


Digit Writing

Image of desired digit

Output: end-effector positions

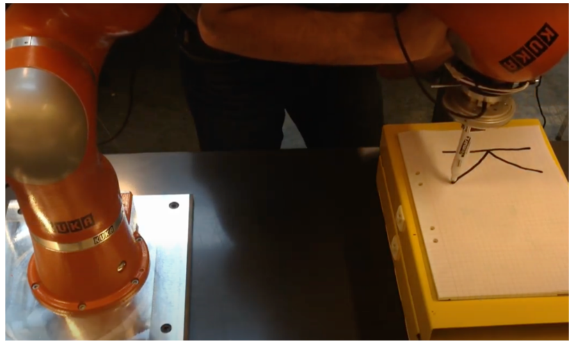


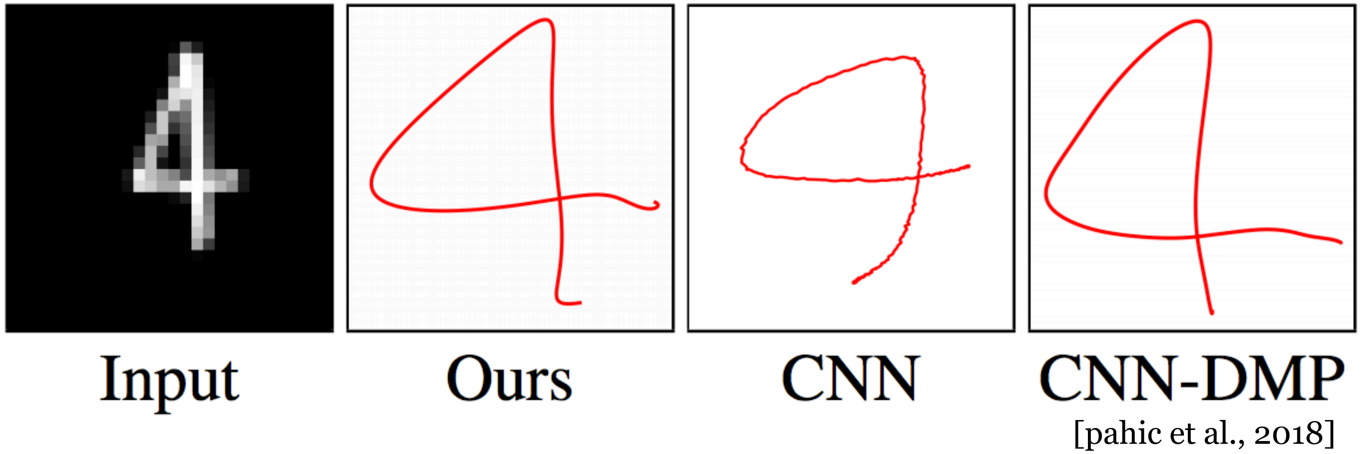


Digit Writing

Image of desired digit

Output: end-effector positions



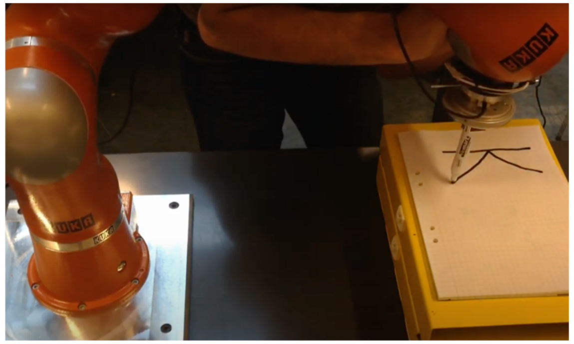


NDP has smoother and more accurate reconstruction

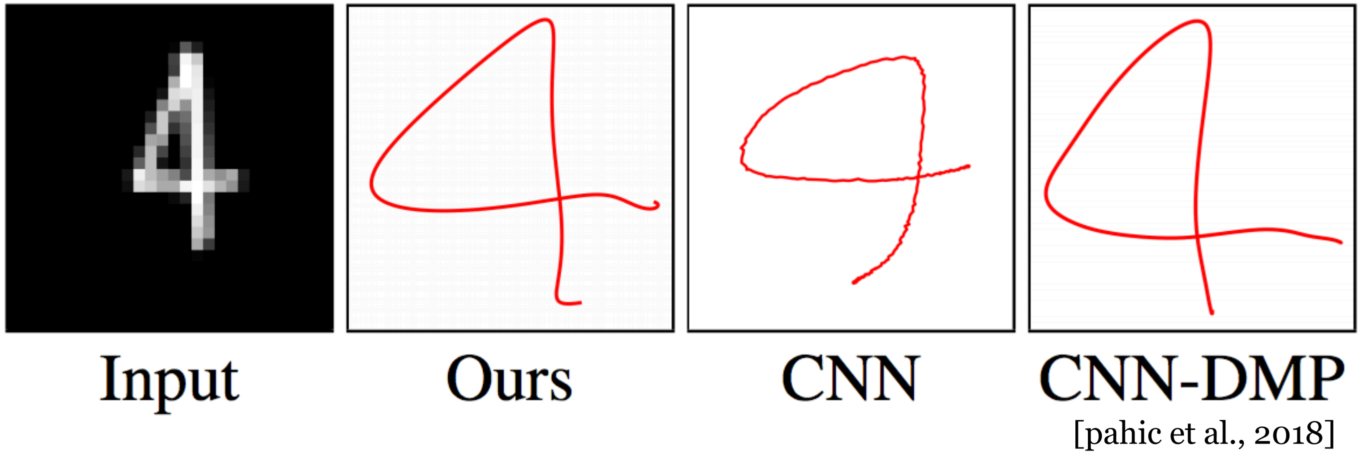
Digit Writing

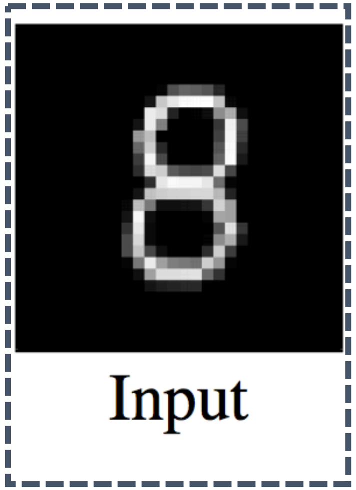
Image of desired digit

Output: end-effector positions



NDP has smoother and more accurate reconstruction

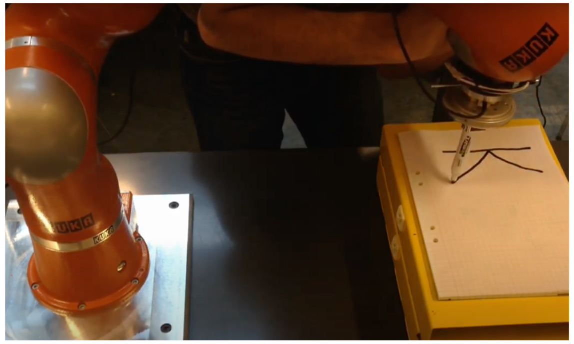




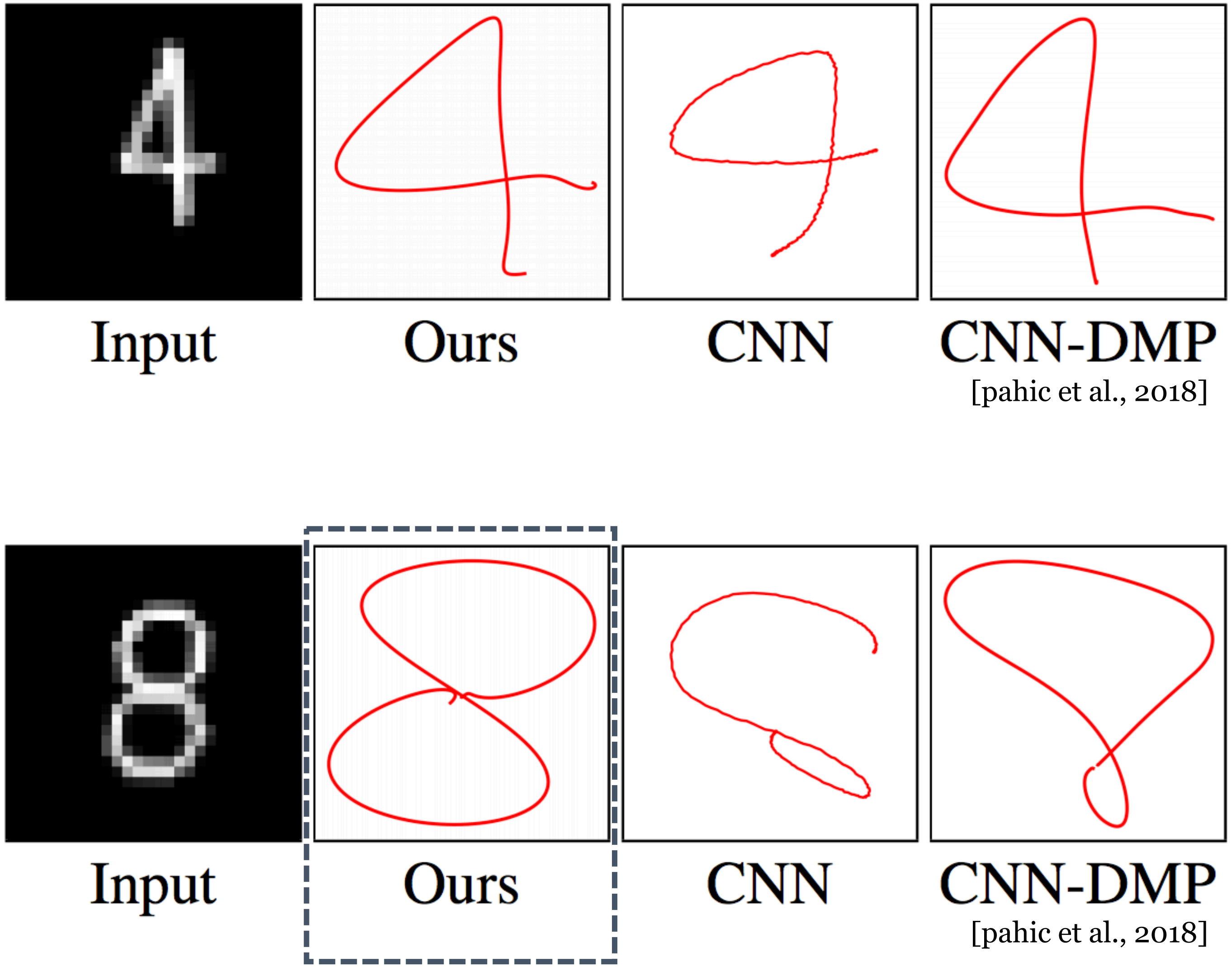
Digit Writing

Image of desired digit

Output: end-effector positions



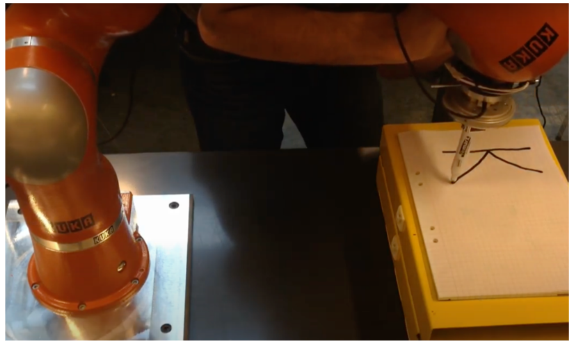
NDP has smoother and more accurate reconstruction



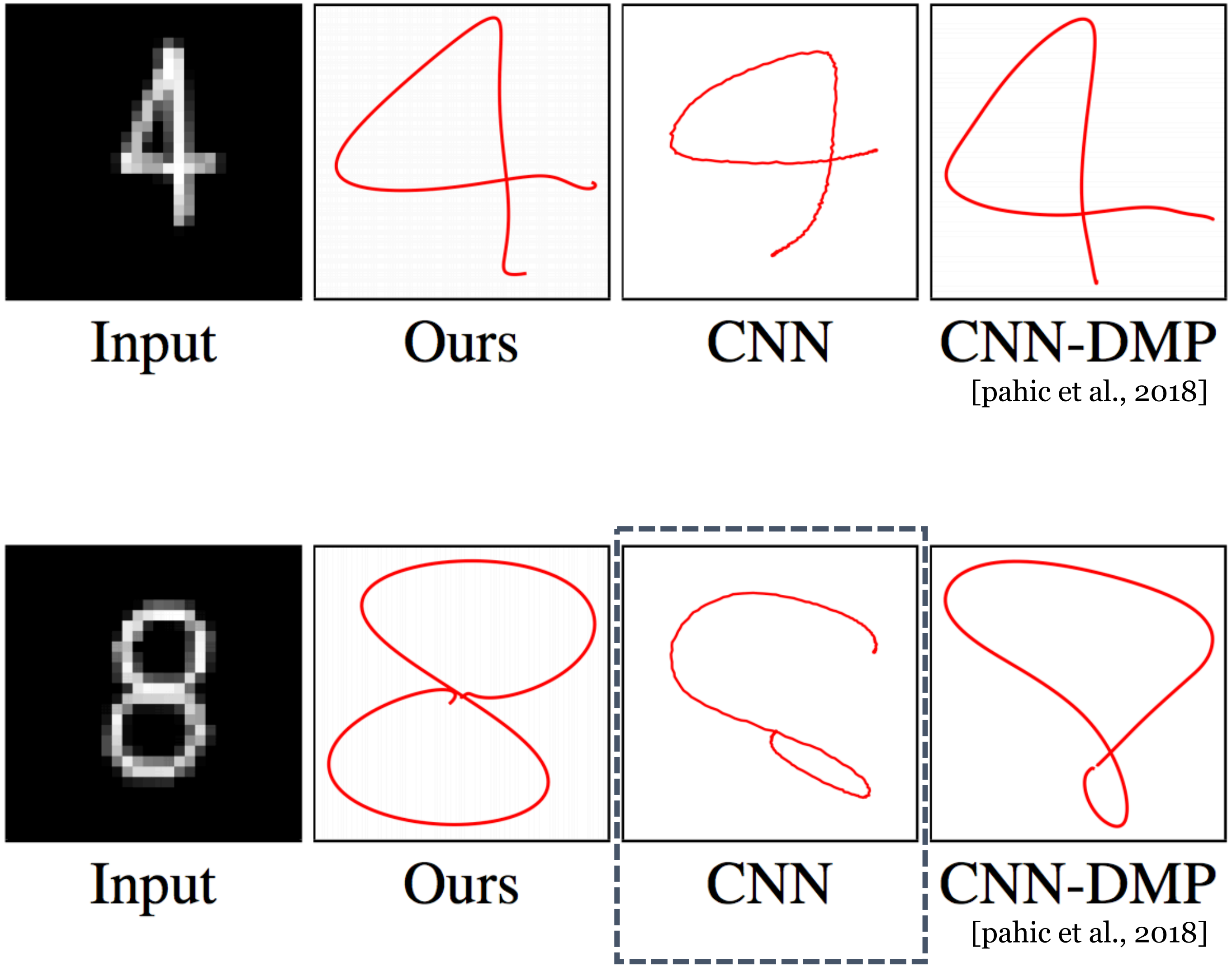
Digit Writing

Image of desired digit

Output: end-effector positions



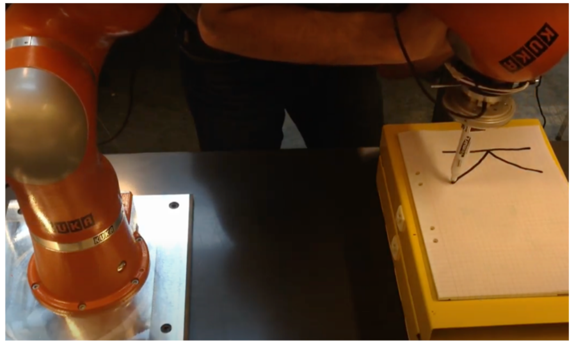
NDP has smoother and more accurate reconstruction



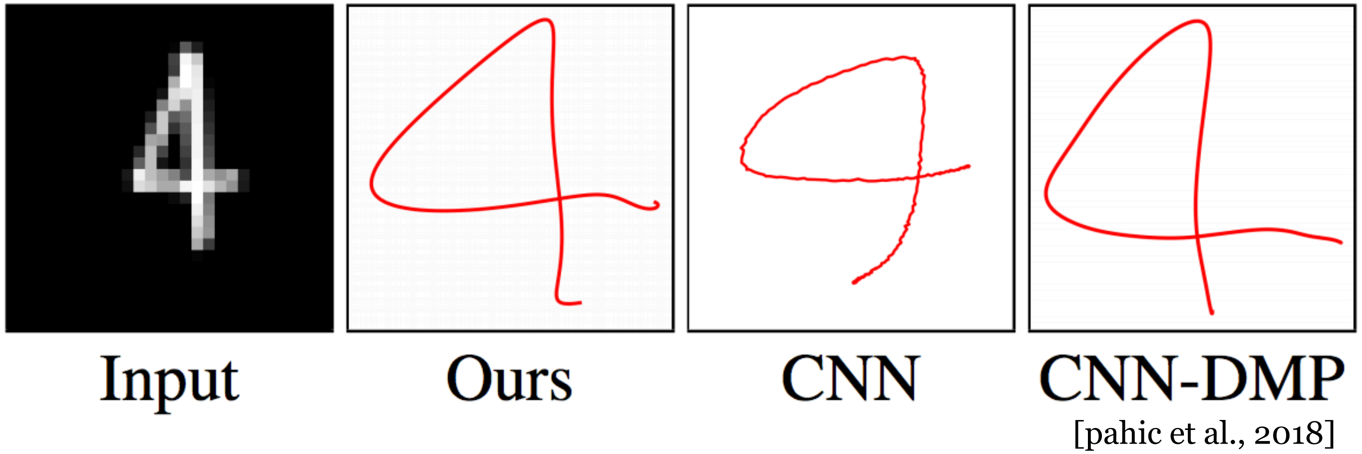
Digit Writing

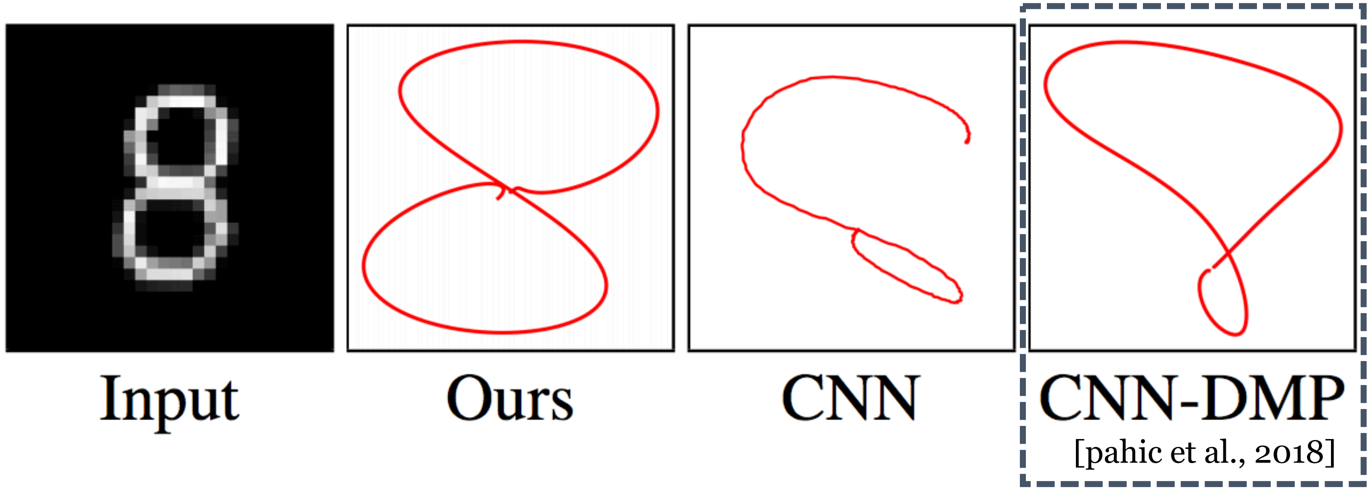
Image of desired digit

Output: end-effector positions



NDP has smoother and more accurate reconstruction





Summary

NDPs are modeled after dynamical systems in nature

Reason at a trajectory level + physically plausible paths

Keep advantages of deep learning (adaptability, learning from visual inputs, etc)

Can easily be integrated in end-to-end setups

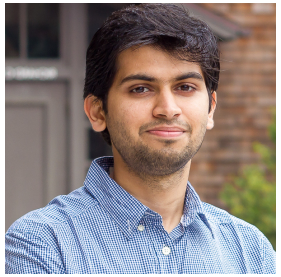
Strong performance in Reinforcement Learning and Imitation Learning, especially dynamic tasks

Thanks for Watching!

For paper and code:

shikharbahl.github.io/neural-dynamic-policy



Carnegie Mellon University
School of Computer Science