Reinforcement Learning for Robotics

Erwin M. Bakker

LIACS Media Lab

Reinforcement Learning

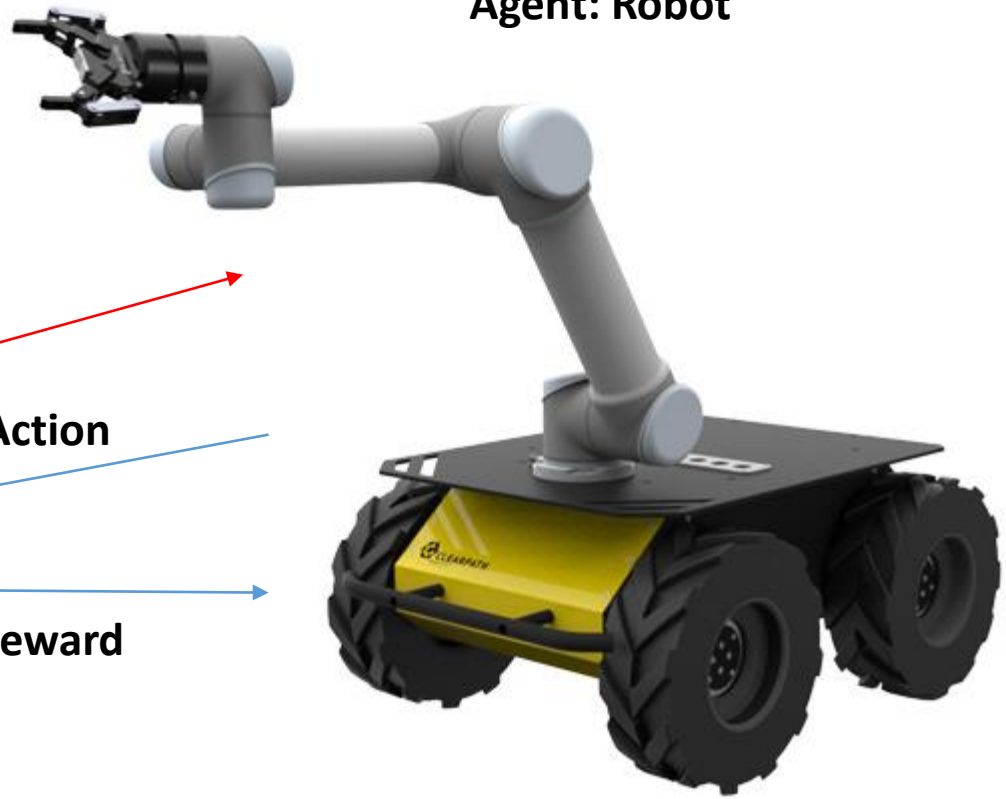
François Chollet, Deep Learning with Python, 2nd Edition. Manning, 2021.

R. Atienza, Advanced Deep Learning with Keras: Apply deep learning techniques, autoencoders, GANs, variational autoencoders, deep reinforcement learning, policy gradients, and more, 2018.

R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series) 2nd Edition, 2018.

W. Zhao, J.P. Queralta, T. Westerlund, Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey, 2020 IEEE Symposium Series on Computational Intelligence (SSCI), 2020.

Reinforcement Learning



Agent: e.g. a Robot

Environment: is in a certain state.

Actions of the Agent:

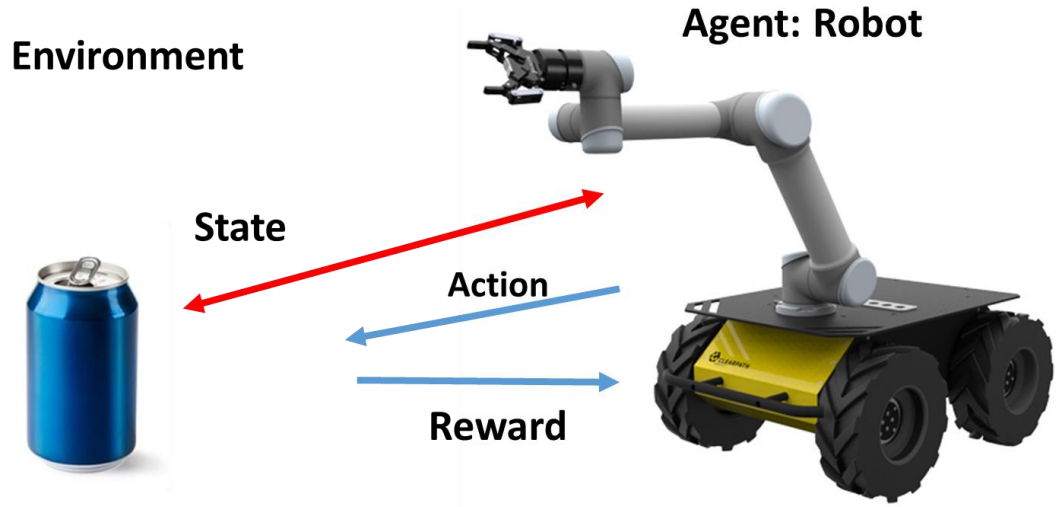
⇒ Environment transitions to a new state

⇒ Agent receives a reward

Policy: decides for each given state which action should be taken.

Goal: Learn a policy that maximizes the accumulated future rewards

Markov Decision Process



Environment

At time step t the environment is in state $s_t \in S$,
where S is the state space, s_0 is the start state, s_t is the current end state.

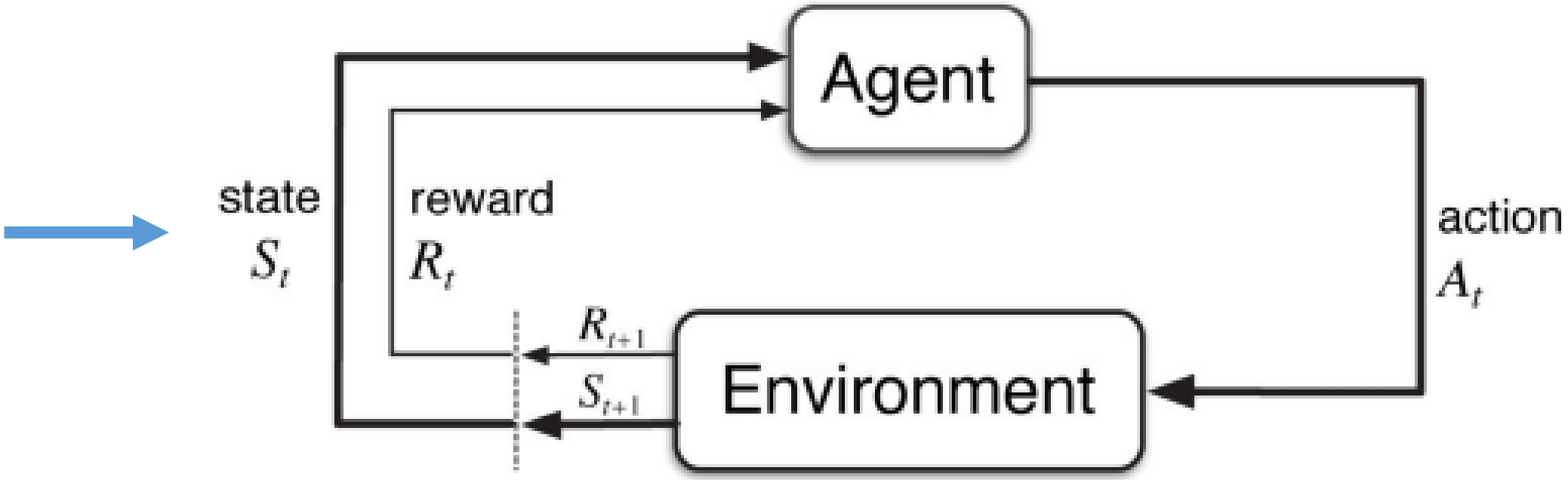
Actions

The agent takes actions from the action space A.
It follows a probabilistic policy $\pi(a_t|s_t)$
i.e., the probability that action a_t is taken given the environment is in state s_t.

Reinforcement Learning (RL) methods specify how
an agent changes its policy π_t as a result of its experience.

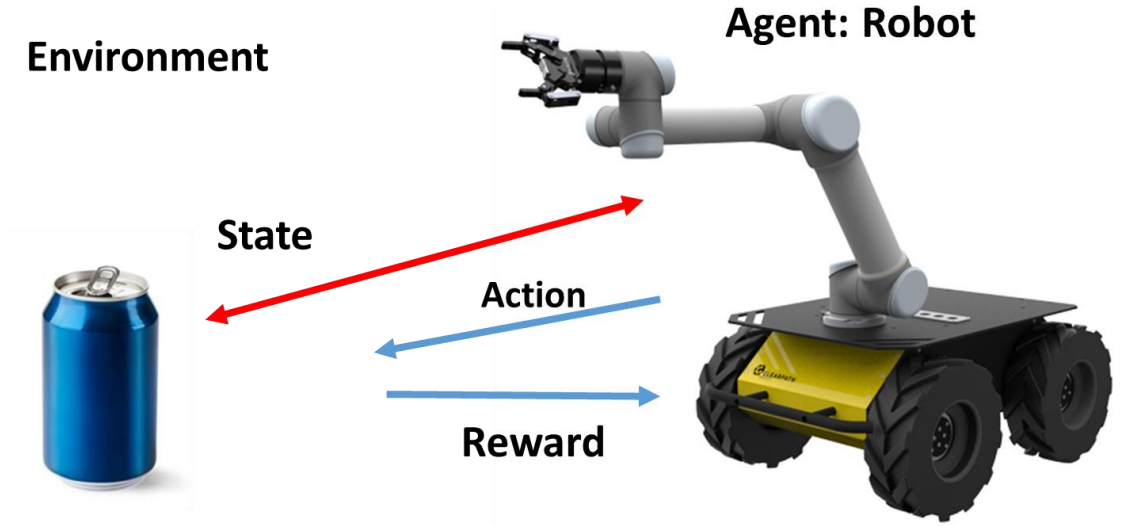
Environment: responds using the state transition $T(s_{t+1}|s_t, a_t)$.
Reward: The agent receives a reward $R_{t+1} = R(s_t, a_t)$

Agent-Environment Interaction



The Markov Decision Process and Agent give rise to a **trajectory**: **S_0**, A_0, R_1, **S_1**, A_1, R_2, **S_2**, A_2, R_3, **S_3**, …

Markov Decision Process (MDP)



Environment at time t in state $s_t \in S$.

Action: - a_t following $\pi(a_t|s_t)$

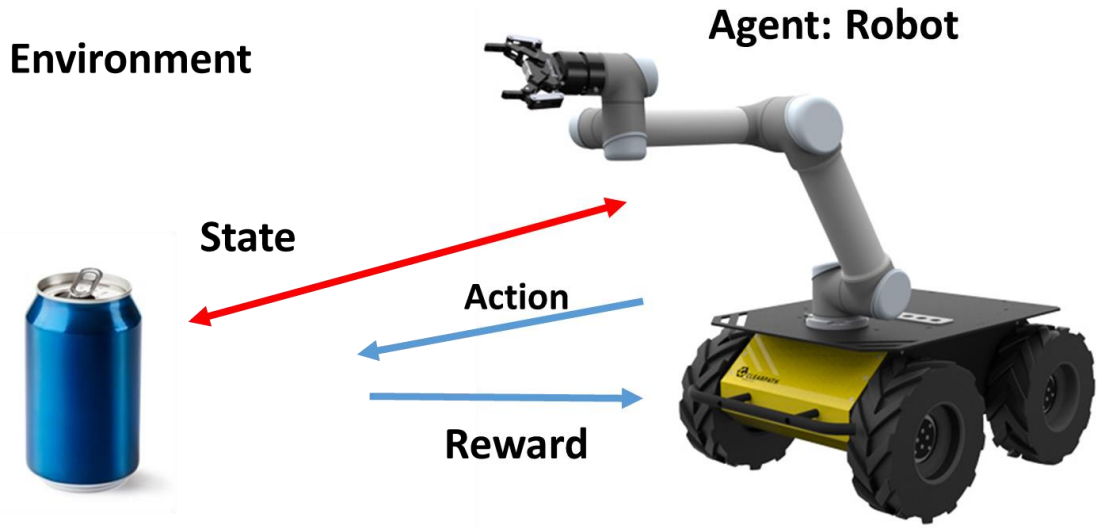
Result: - Environment state transition $T(s_{t+1}|s_t, a_t)$.

- Agent's reward $R_{t+1} = R(s_t, a_t)$

Note:

- Functions T and R may or may not be known to the agent.
- Future rewards can be discounted by γ^k, where $\gamma \in [0,1]$, and k a future time step.
- Process can have episodes => a horizon H is used, with T the number of time steps to complete one episode from s_0 to s_T, …, etc.

Reinforcement Learning (RL)



Environment

- Can be fully or Partially Observable (=> PO-MDP)

Note:

- The decision process sometimes takes past observations into account.
- Obeying the **Markov-property**: all information should be maintained in the current state.

Our robot agent:

- **State** can be a camera estimate of the 3D position of the soda can with respect to the gripper.
- **Reward**
  - +1, if the robot gets closer to the soda can.
  - -1, if the robot gets farther away from the soda can.
  - +100 when it successfully picks up the soda can.

Markov Decision Process (MDP) Framework

Time

- can be abstract, stages

Actions

- low-level: voltages applied to a motor in a robot arm, …
- high level: grab lunch, grab can, recharge, …
- abstract: internal actions

Environment and States

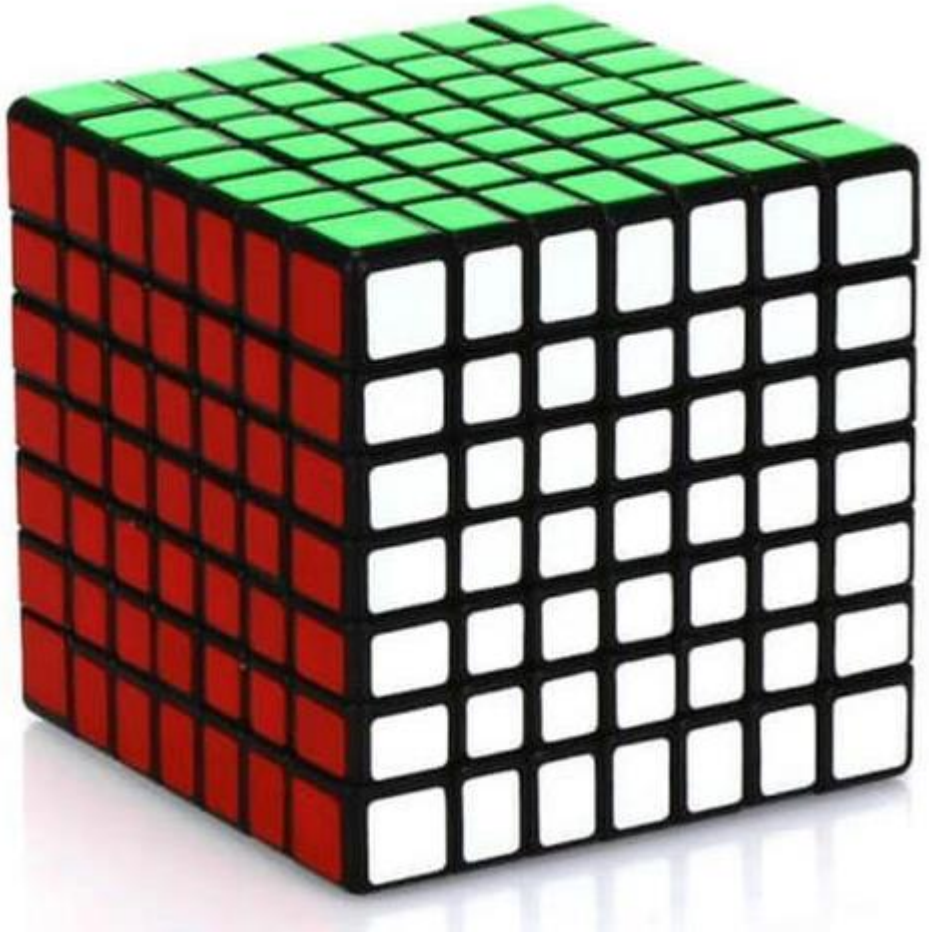
- low-level: sensor readings, …
- high level: symbolic descriptions of objects, …
- abstract: past sensations, subjective, etc.

Markov Decision Process (MDP) Framework

Boundary between Environment and Agent:

- motors, links, and sensors are part of environment
- Represents the limit of the agent's absolute control, not of it's knowledge

Note: An Agent may know everything about how it's environment works, but still it would be a challenging reinforcement learning task.



Example I: Pick and Place Robot

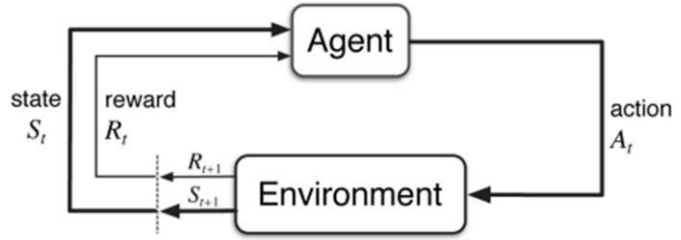
Task: control the motion of a robot arm in a repetitive pick and place task.
Goal: fast and smooth movements

Agent:
- Direct low level control of motors
- Low-latency information of position and velocities of mechanical links

Actions
- Voltage applied to each motor at each joint
- Readings of joint angles and velocities

Reward
- +1 for each object that is picked and placed
- Small negative reward as function of the jerkiness of the motion (per moment).

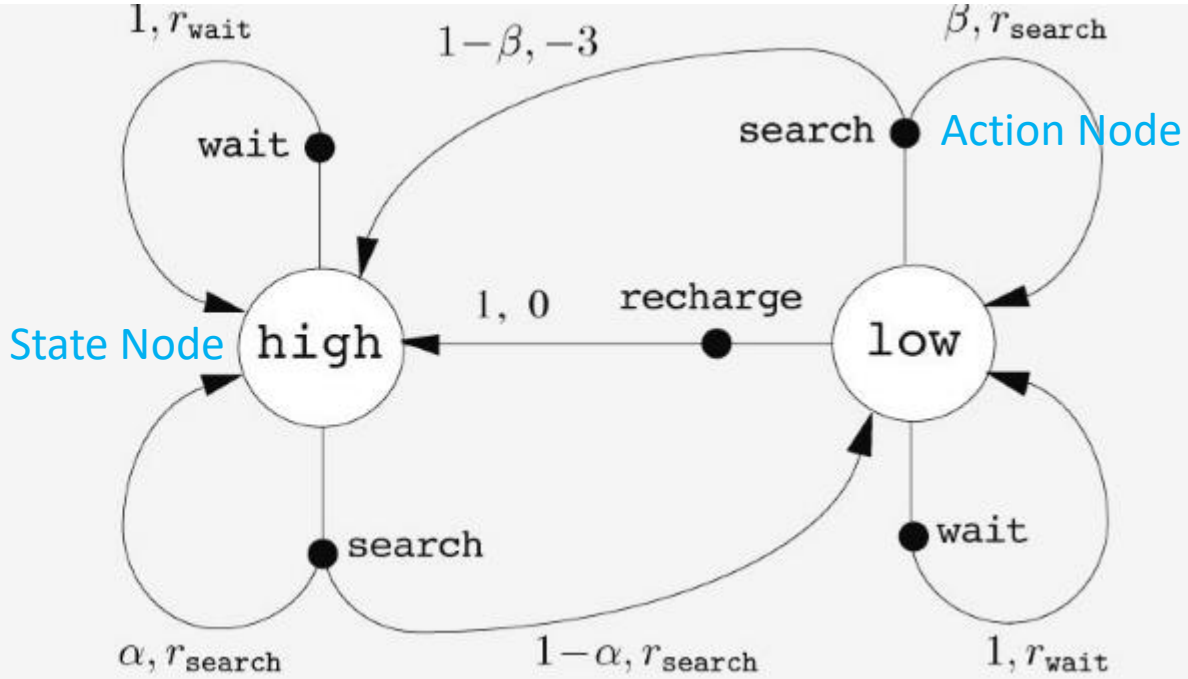
Example II: Recycling Robot



Transition

Transition prob.

Current state s	Action a	Next state s'	$p(s'\|s,a)$	Transition reward $r(s,a,s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	r_{wait}
low	wait	high	0	r_{wait}
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	0



High level agent decides to search, wait or recharge:

- Environment State space: two charge levels: high, low
- Robot Action set: state low -> {search, wait, recharge}; state high -> {search, wait}

Environment responds with state s' and reward r(s,a,s')

Goals and Rewards

- Agent receives after each time step t a reward R_{t+1}
- Goal is to maximize the total amount of received rewards.

The maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

More formally (but still a simplification):

Sequence of rewards after time step t: $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$

T final time step, sum of rewards $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$

Reinforcement Learning (RL)

Goal:

- Maximize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{T+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \qquad \gamma \in [0,1]$$

Note:

- $\gamma \in [0,1]$ the discount rate.
- $\gamma = 0$, if only the immediate reward matters
- $\gamma = 1$, if future rewards weigh the same as the immediate reward

Reinforcement Learning (RL)

Goal:

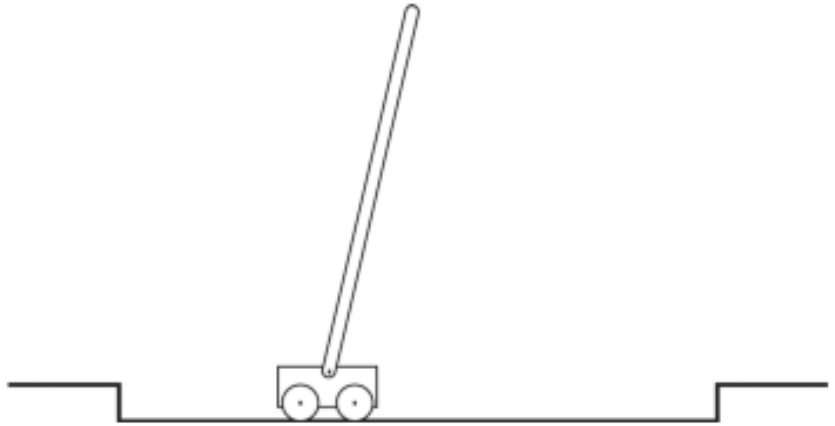
- Maximize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{T+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \qquad \gamma \in [0,1]$$

Note:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{T+3} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{T+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Example III: Pole-Balancing



Objective: Apply forces to the cart such that pole does not fall over.
Failure: If pole falls, or cart runs off the track.

Task of pole-balancing seen as repeated attempts, episodes, during which it is balanced:

Reward: +1 for every time step without failure

$\Rightarrow$ expected return -> ∞ if successful balancing for ever.

Pole-balancing seen as a continuous task:

Reward: -1 on each failure, 0 otherwise.

=> discounted return related to $-\gamma^K$ ($\gamma \in [0,1]$), where K is the number of time steps before failure.

Policies and Estimations: Value Functions

Try to estimate value-functions (of states, or state-action pairs) that estimate for an agent:

1. how good it is to be in a state or
2. how good it is to perform a given action in a given state

(1) The **value function** of a state s under a policy π is defined as:

$$v_\pi(s) = \boldsymbol{E_\pi}[\boldsymbol{G_t}|\boldsymbol{S_t} = \boldsymbol{s}] \quad = \mathrm{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s], \text{ for all } s \in S$$

(2) The **expected return** starting from s, taking action a and further on following policy π is defined as:

$$q_\pi(s,a) = \boldsymbol{E_\pi}[\boldsymbol{G_t}|\boldsymbol{S_t} = \boldsymbol{s}, \boldsymbol{A_t} = \boldsymbol{a}] \quad = \mathrm{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

Reinforcement Learning (RL)

Goal:

- Learn an optimal policy π^*, where

$$\pi^* = argmax_\pi G_t, \qquad \text{where } G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}, \qquad \gamma \in [0,1],$$

$$\text{and} \quad R_{t+1} = R(s_t, a_t)$$

Methods:

- Brute Force, Tabular Methods, Monte Carlo Methods, DNN for RL, Adversarial RL, Sim-to Real Transfer DRL, etc.

[1] L. Pinto, J. Davidson, R. Sukthankar, A. Gupta, Robust Adversarial Reinforcement Learning, March 2017.

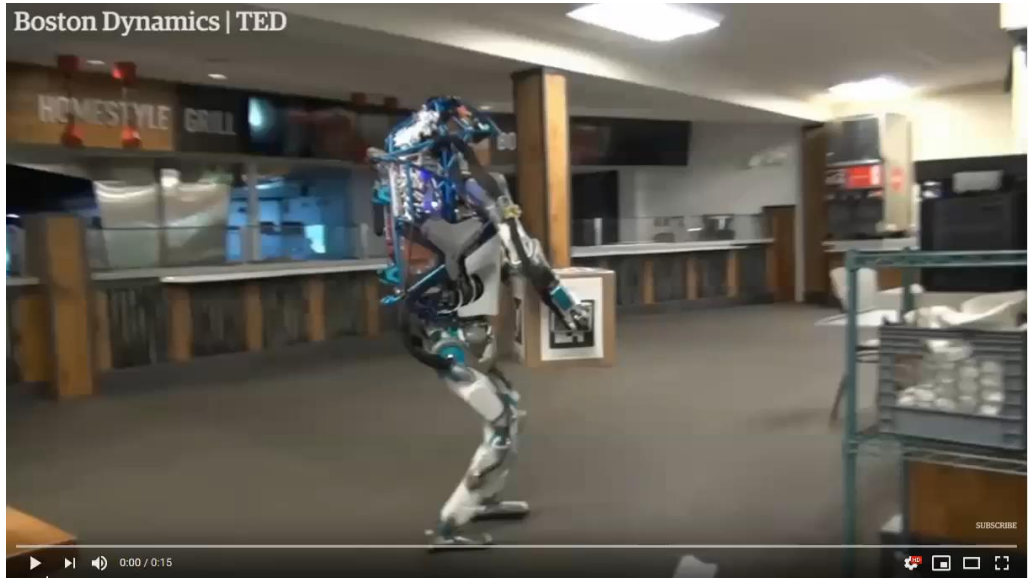
Deep neural networks successes in the field of Reinforcement Learning:

- Fast computations
- Fast Simulations
- Improved networks

But, most RL-based approaches **fail to generalize**, because:

1. gap between simulation and real world
2. policy learning in real world is hampered by data scarcity

RL Challenges for Real-world Policy Learning



The training of the agent's policy
in the real-world:

- too expensive
- dangerous
- time-intensive

$\Rightarrow$ scarcity of data.

$\Rightarrow$ training often restricted to a limited set of scenarios, causing overfitting.

$\Rightarrow$ If the test scenario is different (e.g., different friction coefficient, different mass), the learned policy fails to generalize.

But a learned policy should be robust and generalize well for different scenarios.

RL in the Real World: use more robots

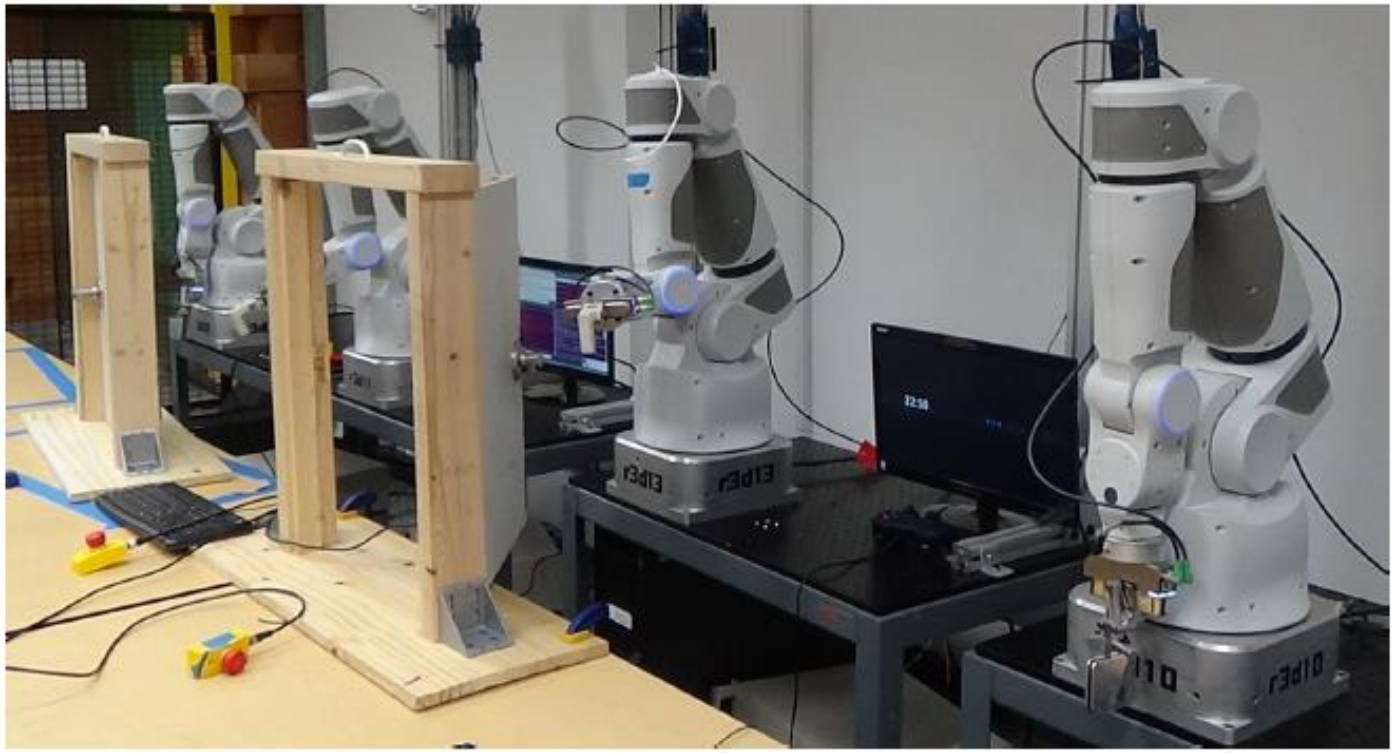


Fig. 1: Two robots learning a door opening task. We present a method that allows multiple robots to cooperatively learn a single policy with deep reinforcement learning.

From [2] Gu et al. , Nov. 2016.

Reinforcement Learning in simulation:

Facing the **data scarcity** in the real-world by

- Learning a policy in a simulator
- Transfer learned policy to the real world

But:

environment and physics of the simulator are not the same as the real world.

=> Reality Gap

This reality gap often results in an unsuccessful transfer, if the learned policy isn’t robust to modeling errors (Christiano et al., 2016; Rusu et al., 2016).

Robust Adversarial Reinforcement Learning (RARL)

Training of an agent in the presence of a **destabilizing adversary**

- Adversary can employ disturbances to the system
- Adversary is trained at the same time as the agent
- Adversary is reinforced: it learns an optimal destabilization policy.

Here policy learning can be formulated as
a zero-sum, minimax objective function.

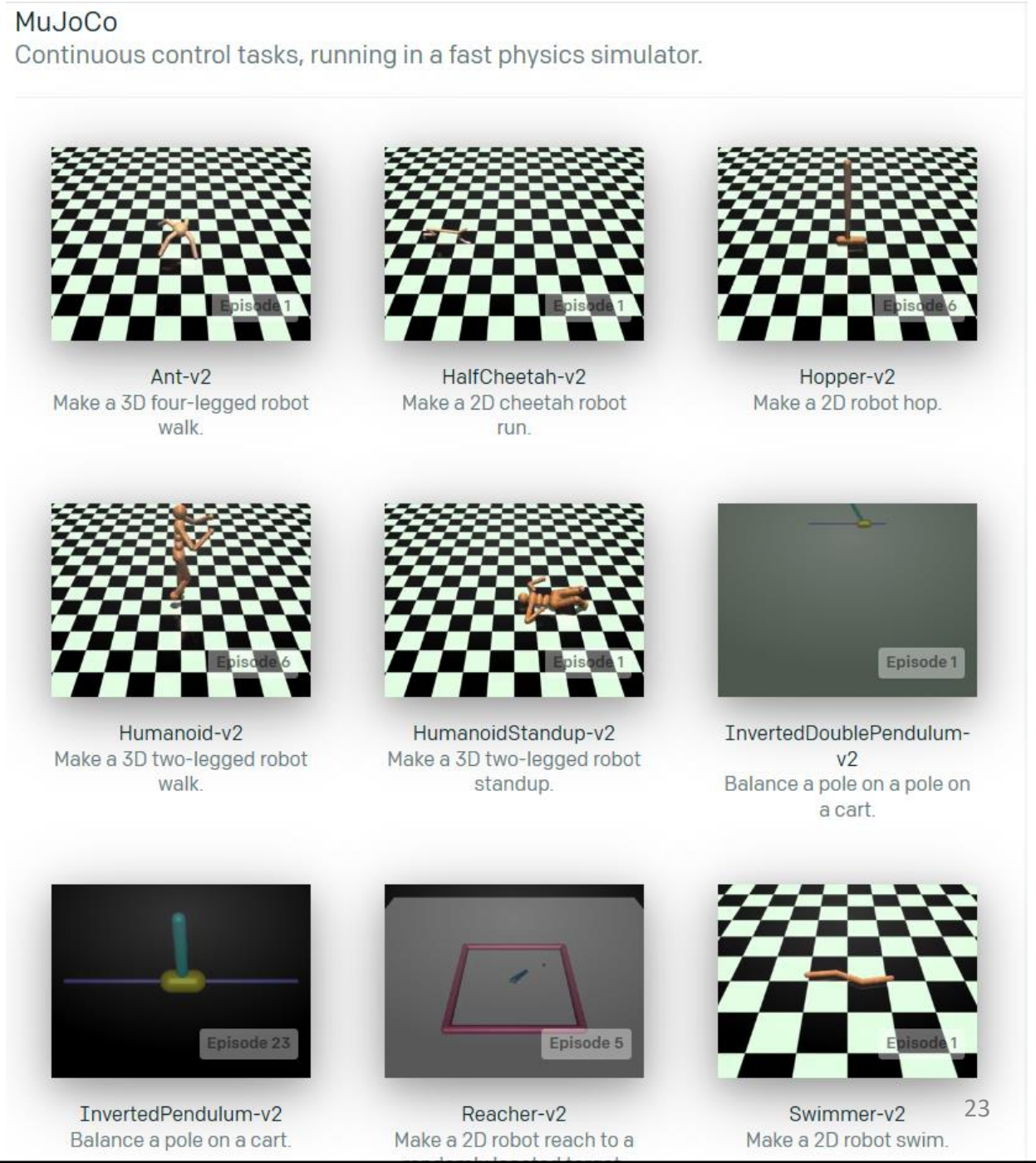
Minimax in zero-sum games: minimizing the opponent's maximum payoff.
Here a zero-sum game is identical to:
- minimizing one's own maximum loss, and to
- maximizing one's own minimum gain

Zero-sum game: gain and loss cancel each other out.

Experimental Environments

- InvertedPendulum
- HalfCheetah
- Swimmer
- Hopper
- Walker2d

https://gym.openai.com/



Unconstrained Scenarios: Challenges

In unconstrained scenarios:

- the space of possible disturbances could be larger than the space of possible actions

=> sampled trajectories for learning etc. become even sparser

Challenges of unconstrained scenarios

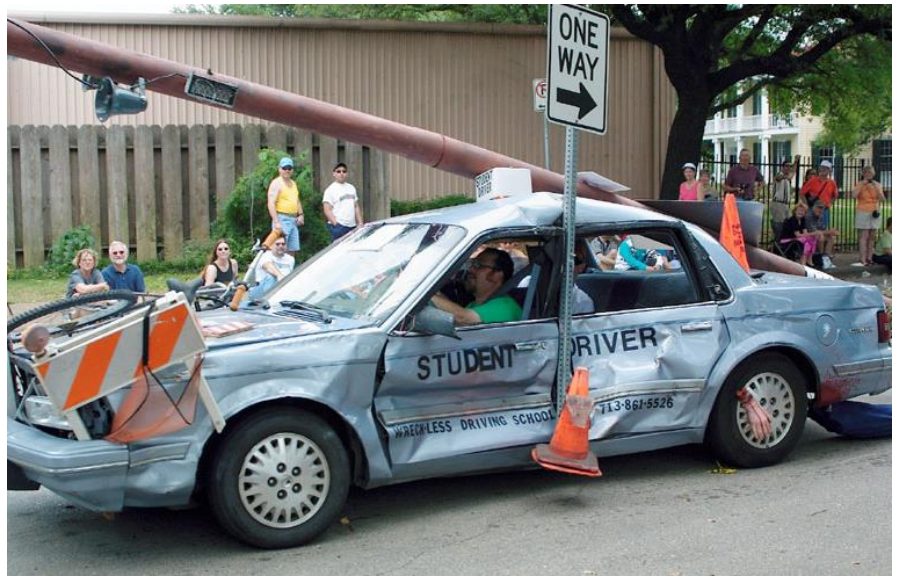
Use adversaries for modeling disturbances:

- we do not want to and can not sample all possible disturbances
- we jointly train a second agent (**the adversary**)
- goal of adversary is to impede the original agent (**the protagonist**)
  - by applying destabilizing forces.
  - rewarded only for the failure of the protagonist

=> the adversary learns to sample hard examples, disturbances that make original agent fail

=> the protagonist learns a policy that is robust to any disturbances created by the adversary.

Challenges of unconstrained scenarios



Use adversaries that incorporate domain knowledge:

- Naïve: **give adversary the same action space as the protagonist**
  - Like a driving student and driving instructor fighting for control of a dual-controlled car.

Proposal paper:

- exploit **domain knowledge**
- focus on the protagonist's weak points;
- give the adversary "super-powers"

=> it can affect the robot or environment in ways the protagonist cannot

e.g. sudden changes in frictional coefficient, mass, etc.

Adversary with Domain Knowledge

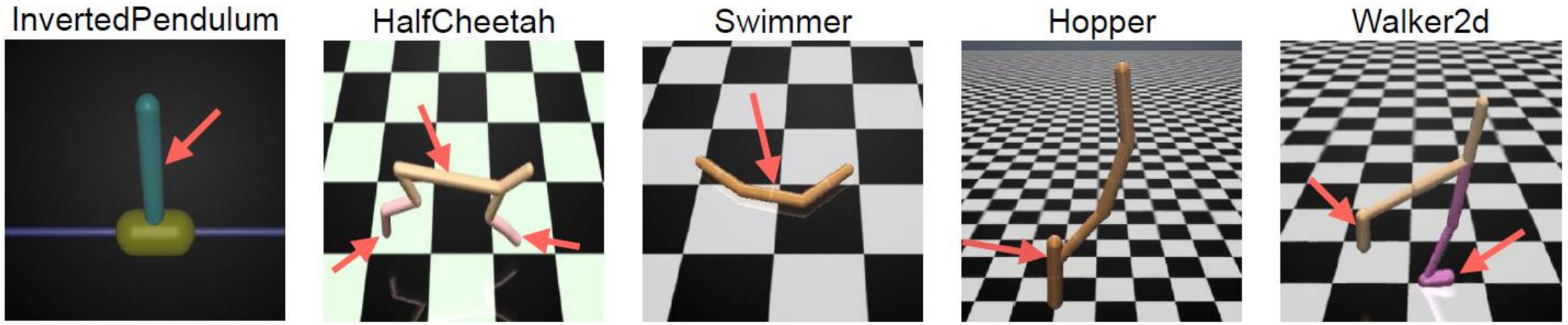


Figure 1. We evaluate RARL on a variety of OpenAI gym problems. The adversary learns to apply destabilizing forces on specific points (denoted by red arrows) on the system, encouraging the protagonist to learn a robust control policy. These policies also transfer better to new test environments, with different environmental conditions and where the adversary may or may not be present.

Figure from [1].

Standard Reinforcement Learning (RL)

RL for continuous space Markov Decision Processes

$(S, A, P, r, \gamma, s_0)$, where

S the set of continuous states
A the set of continuous actions
$P: S \times A \times S \rightarrow \mathbb{R}$ the transition probability
$r: A \rightarrow \mathbb{R}$ the reward function
γ the discount factor
s_0 the initial state distribution

Standard Reinforcement Learning (RL)

- RL for continuous space Markov Decision Processes

  $(S, A, P, r, \gamma, s_0)$, where

  S the set of continuous states

  A the set of continuous actions

  $P: S \times A \times S \rightarrow \mathbb{R}$ the transition probability

  $r: S \times A \rightarrow \mathbb{R}$ the reward function

  γ the discount factor

  s_0 the initial state distribution

Batch policy algorithms [Williams 1992, Kakade 2002, Shulman 2015]:

Learning a stochastic policy:

$\pi_\theta: S \times A \rightarrow \mathbb{R}$ which maximizes $\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)$ the cumulative discounted reward

- Θ the parameters of the policy π.
- Policy π: probability taking action a_t given state s_t at time t

2 Player γ discounted zero-sum Markov Game
(Litman 1994, Perolat 2015)

- **2 Player continuous space Markov Decision Processes**
  $(S, A_1, A_2, P, r, \gamma, s_0)$, where

  S the set of continuous states
  A_1 the set of continuous actions of Player 1
  A_2 the set of continuous actions of Player 2
  $P: S \times A_1 \times A_2 \times S \rightarrow \mathbb{R}$ the transition probability
  $r: S \times A_1 \times A_2 \rightarrow \mathbb{R}$ the reward function of both players
  γ the discount factor
  s_0 the initial state distribution

If Player 1 use strategy μ and Player 2 use strategy ϑ , then the reward function $r_{\mu,\vartheta}$ is given by:

$$r_{\mu,\vartheta} = E_{a^1 \sim \mu(.|s),\, a^2 \sim \vartheta(.|s)}[r(s, a^1, a^2)]$$

Player 1 tries maximizing while Player 2 minimizes the exp. cummulative γ discounted reward R^1

(=> Zero Sum 2 player game)

$$R^{1*} = \min_{\nu} \max_{\mu} R^1(\mu, \nu) = \max_{\mu} \min_{\nu} R^1(\mu, \nu)$$

Robust Adversarial RL Algorithm

The initial parameters for both players' policies are sampled from a random distribution.

Two phases

1. Learn the protagonist's policy while holding the adversary's policy fixed.
2. Learn the adversary's policy while protagonist's policy is held fixed.

Repeat until convergence.

In each phase a *roll*-function is used sampling the N_{traj} trajectories in environment $\mathcal{E}$.

$\mathcal{E}$ contains the transition function P and reward functions r^1 and r^2

Algorithm 1 RARL (proposed algorithm)

Input: Environment $\mathcal{E}$; Stochastic policies μ and ν (= ϑ in our notation)

Initialize: Learnable parameters θ_0^μ for μ and θ_0^ν for ν

for i=1,2,..N_{iter} **do**

  $\theta_i^\mu \leftarrow \theta_{i-1}^\mu$

  for j=1,2,..N_μ **do**

    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_{i-1}^\nu}, N_{\text{traj}})$

    $\theta_i^\mu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{1i}, r_t^{1i})\}, \mu, \theta_i^\mu)$

  end for

  $\theta_i^\nu \leftarrow \theta_{i-1}^\nu$

  for j=1,2,..N_ν **do**

    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_i^\nu}, N_{\text{traj}})$

    $\theta_i^\nu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{2i}, r_t^{2i})\}, \nu, \theta_i^\nu)$

  end for

end for

Return: $\theta_{N_{\text{iter}}}^\mu, \theta_{N_{\text{iter}}}^\nu$

Experimental Setup

- Environments built using OpenAI gym's (Brockman et al., 2016).
- Control of environments with the MuJoCo physics simulator (Todorov et al., 2012) .

RARL is built on top of rllab (Duan et al., 2016)

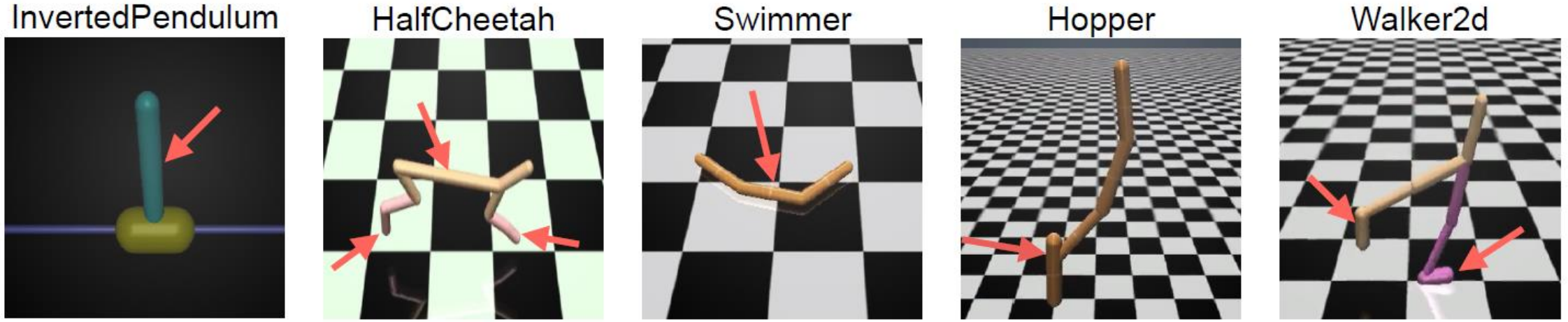
Baseline: Trust Region Policy Optimization (TRPO) (Schulman et al., 2015)

For all the tasks and for both the protagonist and adversary,
a policy network with two hidden layers with 64 neurons per layer is used.

Robust Adversarial RL and the baseline are trained with

- 100 iterations on InvertedPendulum
- 500 iterations on the other environments

Hyper-parameters of Trust Region Police Optimization (TRPO) are selected by grid search.



InvertedPendulum

- State space 4D: position, velocity
- Protagonist: 1D forces; Adversary: 2D forces on center of pendulum

HalfCheetah

- State space 17D: joint angles and joint velocities, …
- Adversary: 6D actions with 2D forces

Swimmer

- State space 8D: joint angles and joint velocities, …
- Adversary: 3D forces to center of swimmer

Hopper

- State space 11D: joint angles and joint velocities, …
- Adversary: 2D force on foot

Walker2d

- State space 17D: joint angles and joint velocities, …
- Adversary: 4D actions with 2D forces on both feet

Actions of Adversary

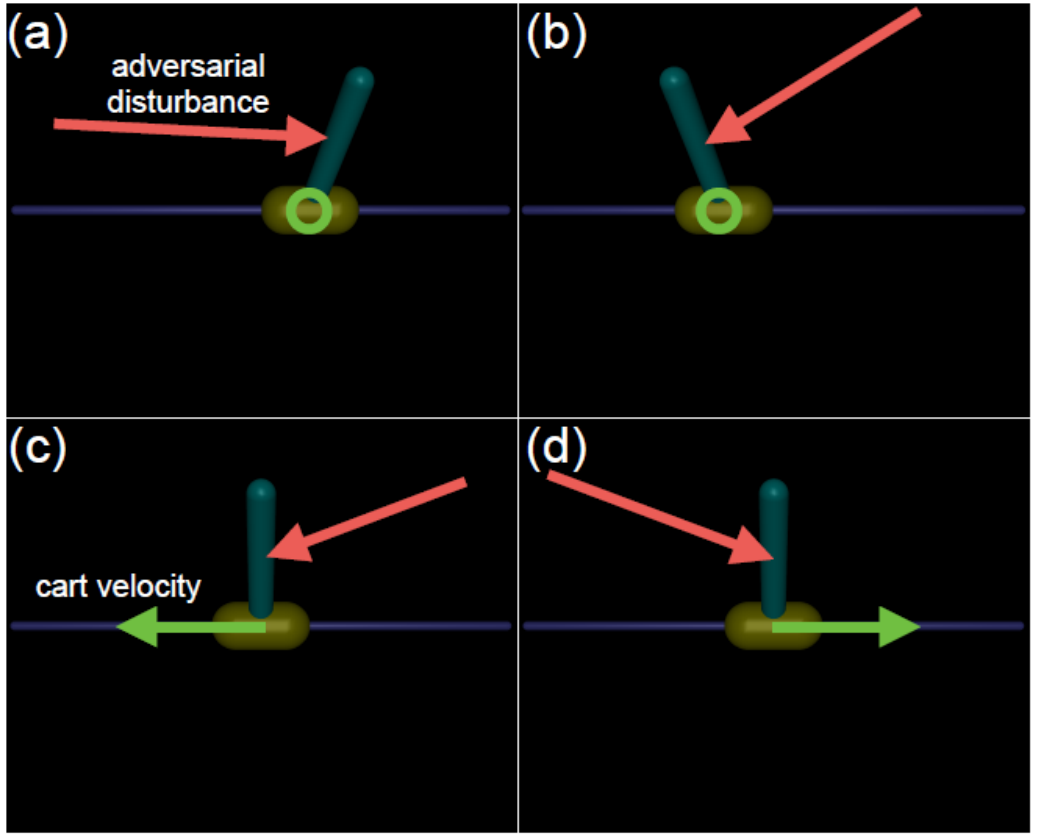


Figure 8. Visualization of forces applied by the adversary on InvertedPendulum. In (a) and (b) the cart is stationary, while in (c) and (d) the cart is moving with a vertical pendulum.

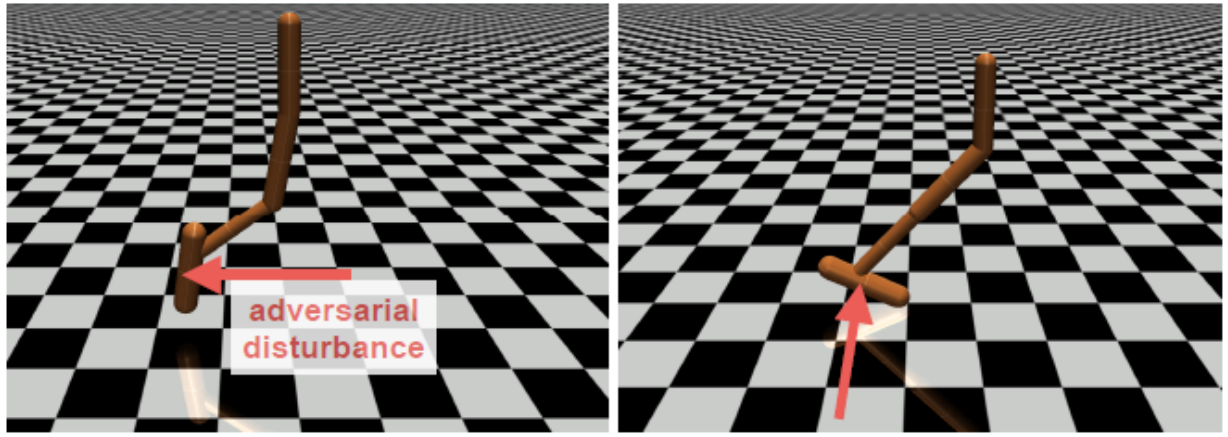


Figure 9. Visualization of forces applied by the adversary on Hopper. On the left, the Hopper's foot is in the air while on the right the foot is interacting with the ground.

Results

RARL achieves better mean than Baseline.

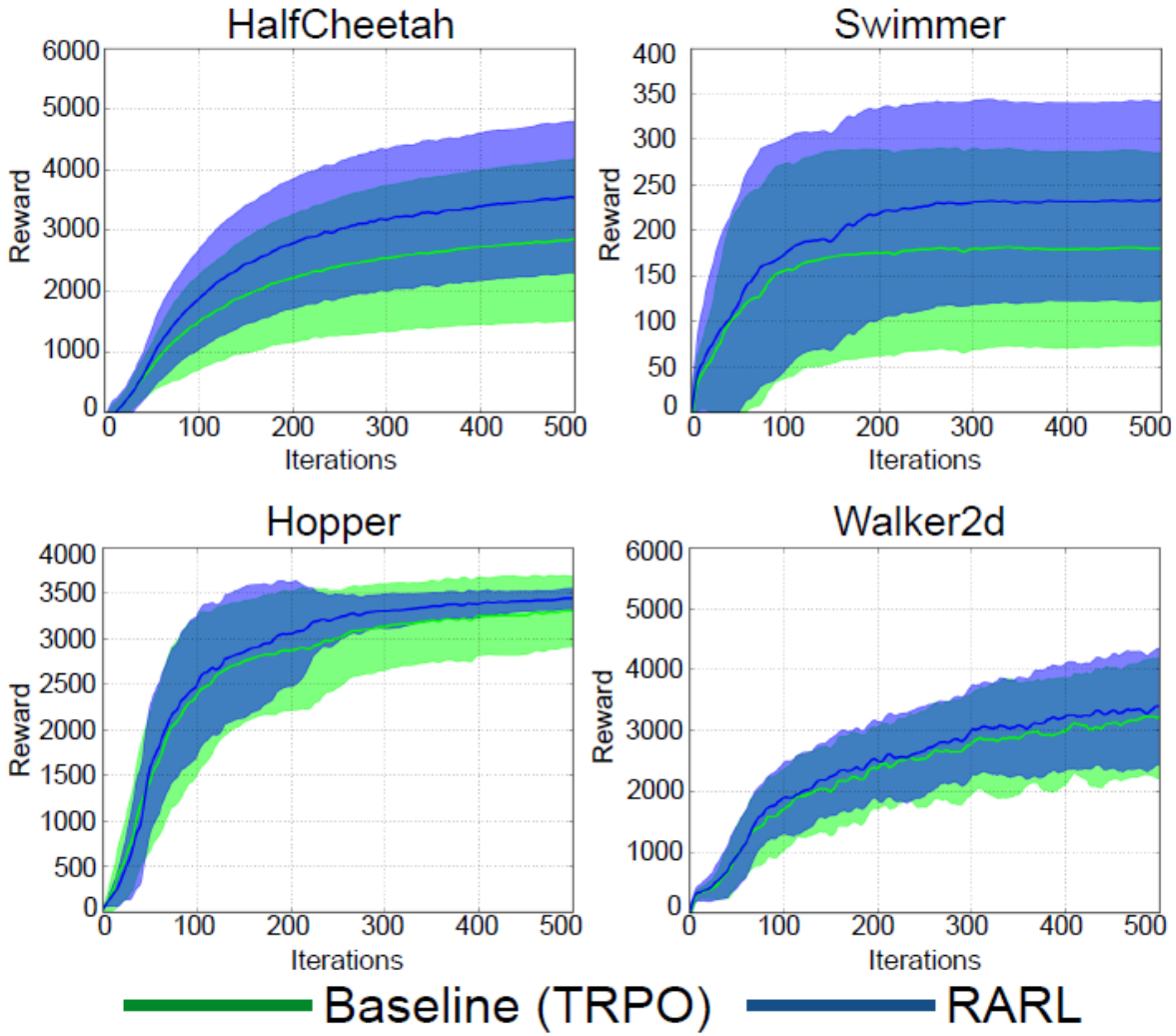


Figure 2. Cumulative reward curves for RARL trained policies versus the baseline (TRPO) when tested without any disturbance. For all the tasks, RARL achieves a better mean than the baseline. For tasks like Hopper, we also see a significant reduction of variance across runs.

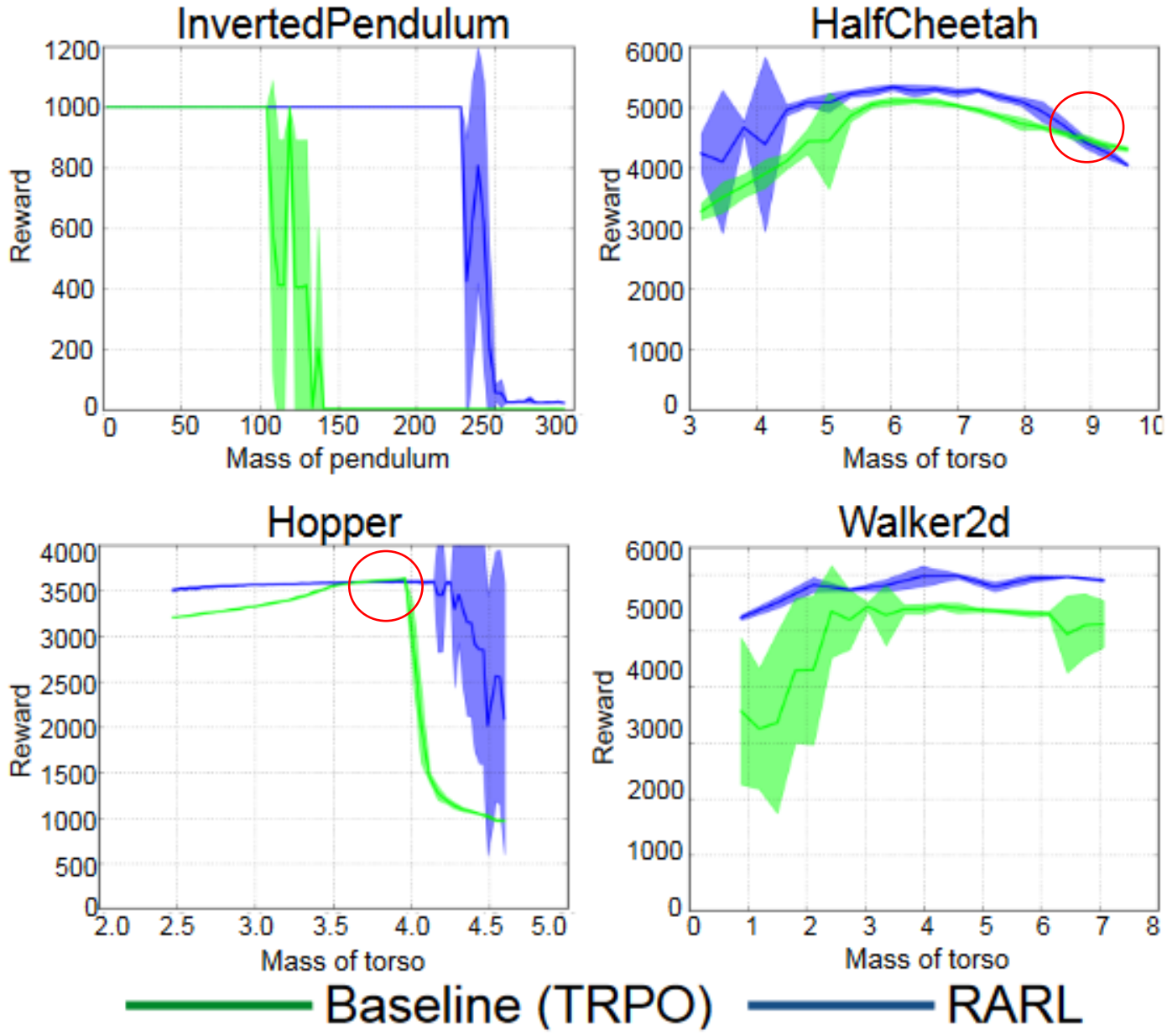
Table 1. Comparison of the best policy learned by RARL and the baseline (mean±one standard deviation)

	InvertedPendulum	HalfCheetah	Swimmer	Hopper	Walker2d
Baseline	1000 ± 0.0	5093 ± 44	358 ± 2.4	3614 ± 2.16	5418 ± 87
RARL	1000 ± 0.0	5444 ± 97	354 ± 1.5	3590 ± 7.4	5854 ± 159

Results
Robustness to Changing Mass

Inverted Pendulum:
- mass of pendulum changed.

For others:
- mass of torso changed.



Results Robustness to Changing Friction

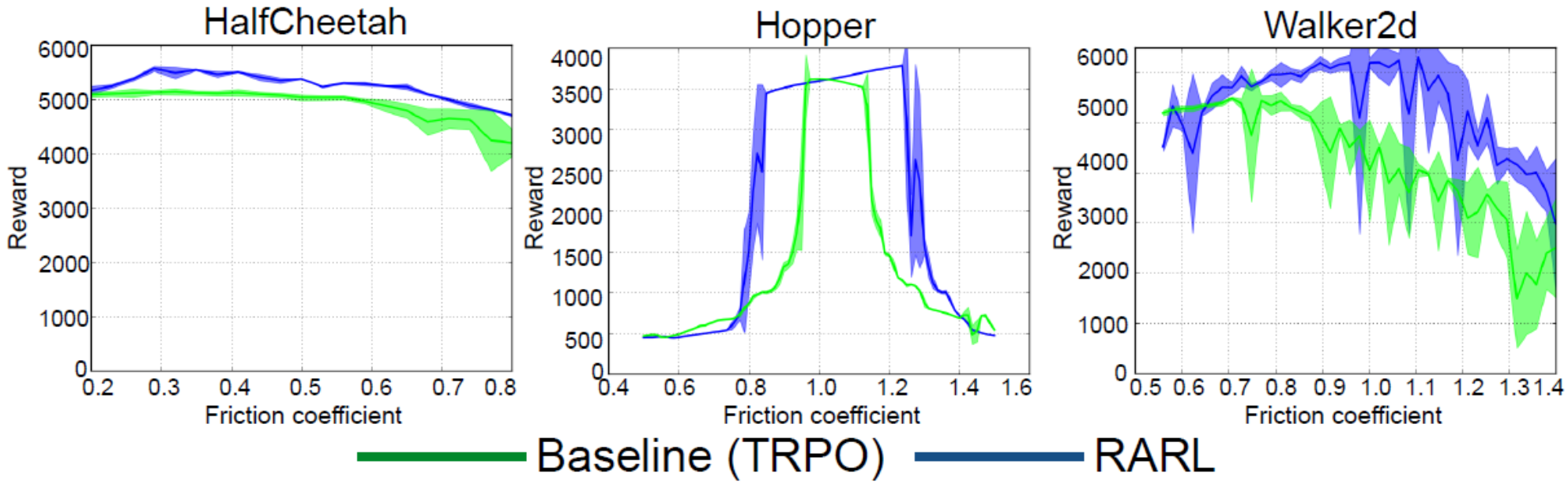


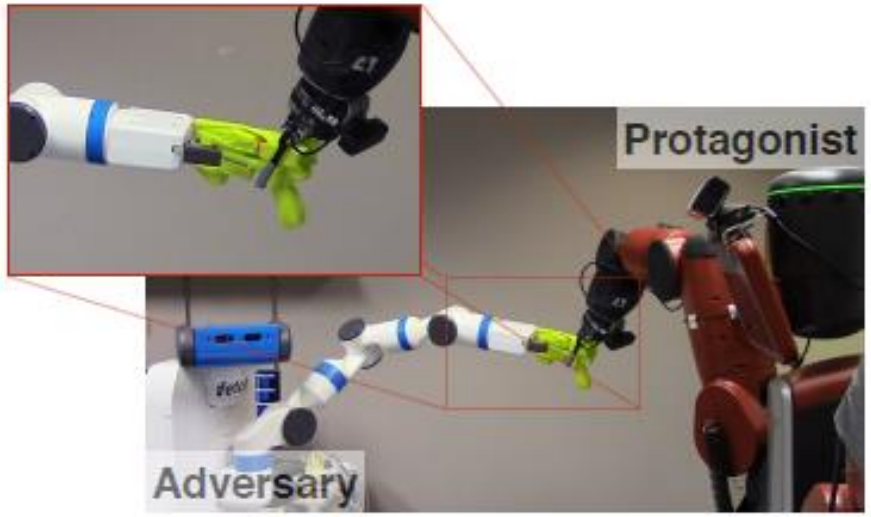
Figure 6. The graphs show robustness of RARL policies to changing friction between training and testing. Note that we exclude the results of InvertedPendulum and the Swimmer because friction is not relevant to those tasks.

Conclusions Experiment Results

1. improves training stability
2. is robust to differences in training/test conditions
3. outperform the baseline even in the absence of the adversary

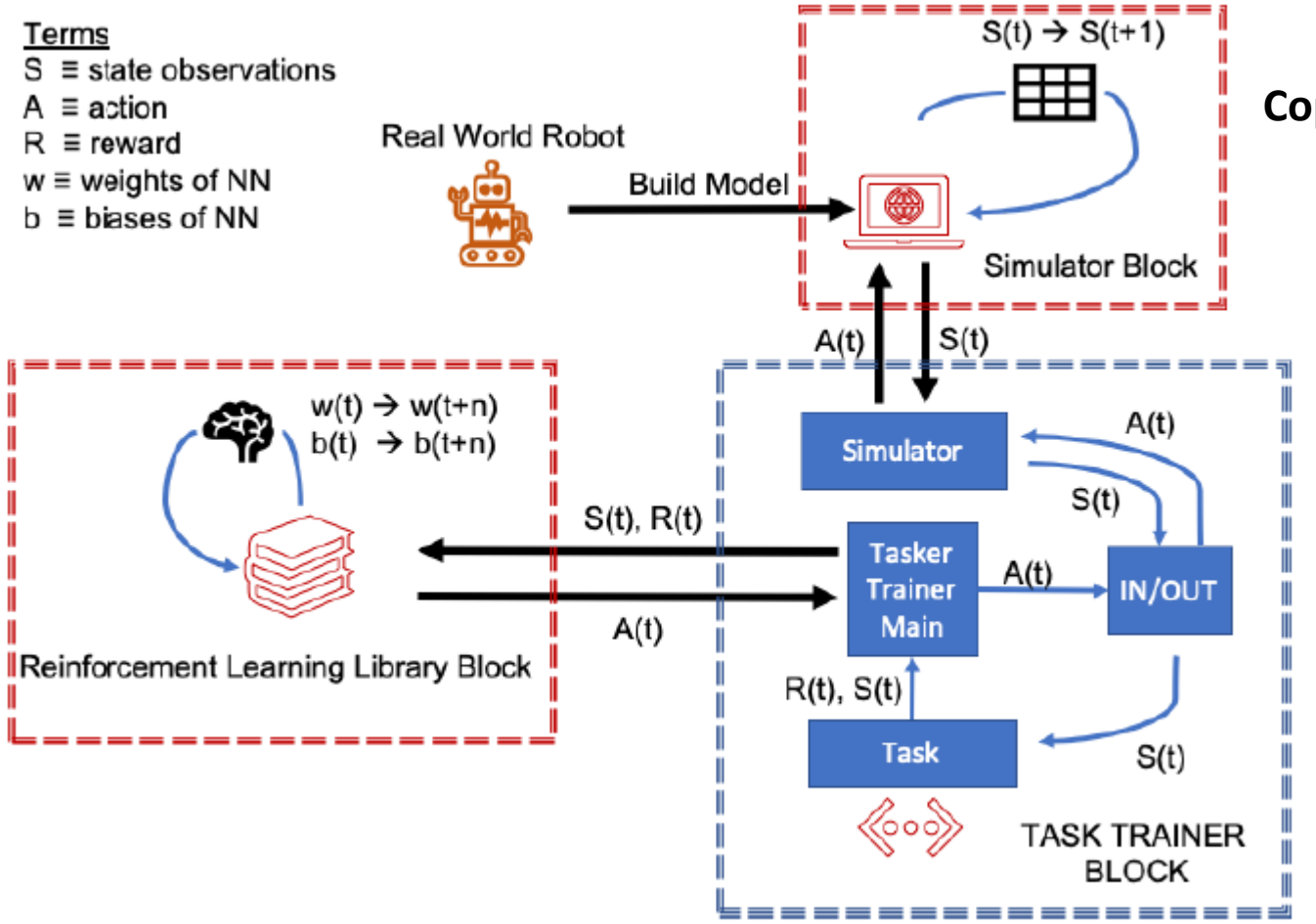
Discussion

- Results for completely simulated environments: how does it translate to the real world?
- Adversary can be very easily too powerful. How do you incorporate/ formulate the adversary's powers in your RARL model?
- Can you think of a good hybrid setup: part simulator, part the real thing. Have the adversary coming from/to the real world into the simulation…
- …

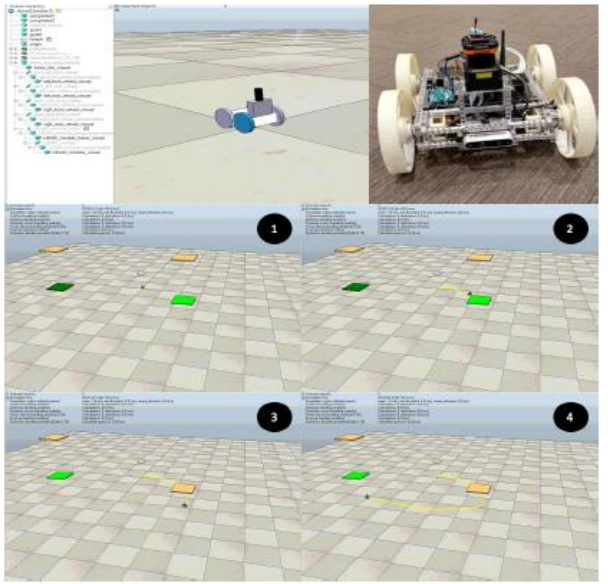


From [4] Pinto et al., 2016.

T. Blum et al. **RL STaR Platform: Reinforcement Learning for Simulation based Training of Robots, i-SAIRAS2020, Oct. 2020.**



CoppeliaSim (used in alternative platforms with Pyrep)



OpenAI's **Baselines**; **Stable Baselines**, **Tenserflow RL Agents**

https://github.com/Space-Robotics-Laboratory/rlstar ; Others: https://github.com/chauby/CoppeliaSimRL ; https://github.com/stepjam/PyRep; most used https://gym.openai.com/

W. Zhao, et al., Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey, IEEE SSCI, 2020.

Zero-shot Transfer

- build a realistic simulator; use extensive simulator time => direct transfer to real-world

System Identification

- Use mathematical models and calibrate them to the real environment

Domain Randomization

- Visual randomization, and/or dynamics randomization of simulator

Domain Adaptation

- Use data from source domain to further train the model.

Learning with Disturbances

- Noisy rewards; environmental perturbations (when learning in parallel); etc.

Simulation Environments

- Gazebo (with ROS), Unity3D, PyBullet, MuJuCo (last 2 good integration with DL and RL)

See the original paper by Zhao et al. [8] for a more careful look at this table.

TABLE I: Classification of the most relevant publications in Sim2Real Transfer.

	Description	Sim-to-real transfer and learning details	Multi-agent learning	Simulator / Engine	Knowledge Transfer	Learning Algorithm	Real Robot/Platform	Application
Balaji et al. [30]	DeepRacer: an educational autonomous racing platform.	Random colors and parallel domain randomization	✓(sim only) Distr. rollout	Gazebo RoboMaker	✗	PPO	DeepRacer 4WD 1:18 Car	Autonomous racing
Traore et al. [12]	Continual RL with policy distillation and sim-to-real transfer.	Continual learning with policy distillation.	✗	PyBullet	✓Multi-task Distillation	PPO2	Small mobile platform	Robotic navigation
Kaspar et al. [31]	Sim-to-real transfer for RL without Dynamics Randomization.	System identification and a high-quality robot model.	✗	PyBullet	✗	SAC	KUKA LBR iiwa +WSG50 gripper	Peg-in-Hole manipulation
Matas et al. [6]	Sim-to-real RL for deformable object manipulation.	Stochastic grasping and domain radomization.	✓(sim)	PyBullet	✗	DDPGfD	7DOF Kinova Mico Arm	Dexterous manipulation
Witman et al. [32]	Sim-to-real RL for thermal effects of an atmospheric pressure plasma jet.	Custom physics model and dynamics randomization	✗	Custom	✗	A3C	kHz-excited APPJ in He	Plasma jet control
Jeong et al. [33]	Modeling Generalized Forces with RL for Sim2Real Transfer	Modeling and learning state dependent generalized forces.	✗	MuJoCo	✗	MPO	Rethink Robotics Sawyer	Nonprehensile manipulation
Arndt et al. [11]	Meta Reinforcement Learning for Sim2Real Domain Adaptation	Domain random. and model-agnostic meta-learning.	✗	MuJoCo	✓Meta-training	PPO	Kuka LBR 4+ arm	Manipulation (hockey puck)
Breyer et al. [34]	Flexible robotic grasping with Sim2Real RL	Direct transfer. Elliptic mask to RGB-D images.	✗	PyBullet	✗	TRPO	ABB YuMi with parallel-jaw gripper	Robotic Grasping
Van Baar et al. [35]	Sim-to-real transfer with robustified policies for robot tasks.	Variation of appearance and/ or physics parameters.	✓(sim)	MuJoCo +Ogre 3D	✓	A3C (sim) +Off-policy	Mitsubishi Melfa RV-6SL	Marble Maze Manipulation
Bassani et al. [36]	Sim2Real RL for robotic soccer competitions.	Domain adaptation and custom simulator for transfer.	✗	VSSS-RL	✓	DDPG /DQN	VSSS Robot	Robotic Navigation
Qin et al. [37]	Sim2Real for six-legged robots with DRL and curriculum learning.	Curriculum learning with inverse kinematics.	✗	V-Rep	✓	PPO	Six-legged robot	Navigation and obstacle avoid.
Vacaro et al. [38]	Sim-to-real in reinforcement learning for everyone	Domain randomization (light + color + textures).	✓(sim)	Unity3D	✗	IMPALA	Sainsmart robot arm	Low-cost robot arm
Chaffre et al. [39]	Sim-to-Real Transfer with Incremental Environment Complexity	SAC training using incremental environment complexity.	✗	Gazebo	✗	DDPG /SAC	Wifibot Lab V4	Mapless navigation
Kaspar et al. [40]	Rl with Cartesian Commands for Peg in Hole Tasks.	Dynamics (CMA-ES) and environment randomization.	✗	PyBullet	✗	SAC	Kuka LBR iiwa	Peg-in-hole tasks
Hundt et al. [41]	Efficient RL for Multi-Step Visual Tasks via Reward Shaping.	Direct transfer with custom simulation framework.	✗	SPOT Framework	✗	SPOT-Q +PER	Universal Robot UR5	Long-term multi-step tasks
Pedersen et al. [42]	Sim-to-Real Transfer for Gripper Pose Estimation with GAN	CycleGANs for domain adaption and transfer.	✗	Unity	✗	PPO	Panda robot	Robotic Grippers
Ding et al. [43]	Sim-to-Real Transfer for Optical Tactile Sensing	Analysis of different amounts of randomization.	✗	PyBullet	✗	CNN	Sawyer robot +TacTip sensor	Tactile sensing
Muratore et al. [9]	Data-efficient Bayesian Domain Randomization for sim-to-real	Proposed bayesian randomization (BAYR).	✗	Custom/ BoTorch	✗	PPO / RF Classifier	Quanser Qube	swing-up/ balancing
Zhao et al. [8]	Towards closing the sim-to-real gap in collaborative DRL with perturbances	Domain randomization (custom perturbations)	✓(sim)	Pybullet	✗	PPO	Kuka (sim-only)	Robot arm reacher
Nachum et al. [44]	Multi-agent manipulation via locomotion	Hierarchichal sim-to-real, model-free, zero-shot transfer.	✓	MuJoCo	✗	Custom	D'Kitty robo (2x)	Multi-agent manipulation
Rajeswaran et al. [5]	Dexterous manipulation with DRL and demonstrators.	Imitation learning via demonstrators with VR.	✗	MuJoCo	✗	DAPG	ADROIT 24-DoF Hand	Multi-fingered robot hands

Very nice primer for RL to have a look at:

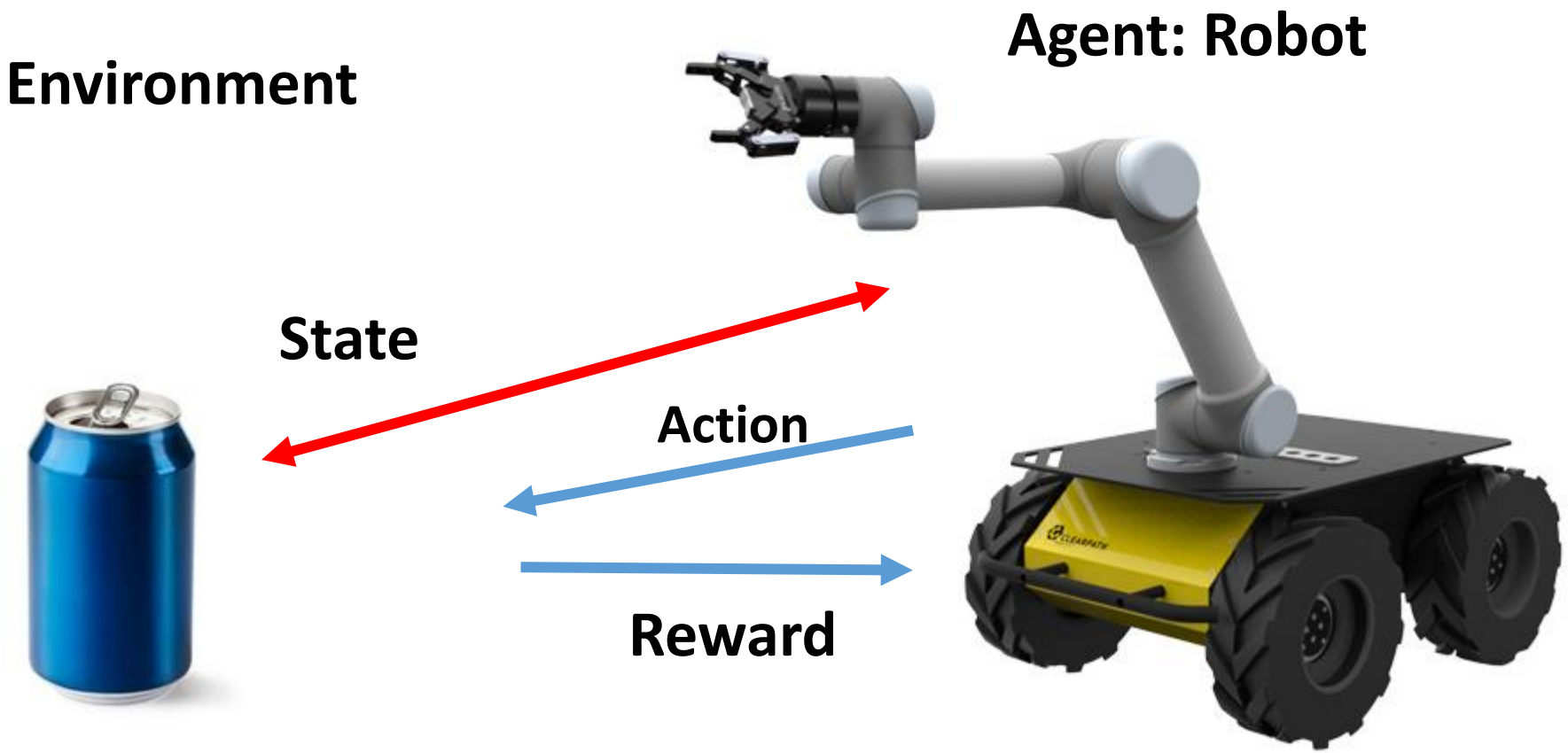
- https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

- MuJoCo is a proprietary software that requires a license,
- There is a free trial and above that it is free for students.

References

1. L. Pinto, J. Davidson, R. Sukthankar, A. Gupta, Robust Adversarial Reinforcement Learning, *Proceedings of the 34th International Conference on Machine Learning*, PMLR 70:2817-2826, 2017.
2. S. Gu, E. Holly, T. Lillicrap, S. Levine, Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates, arXiv:1610.00633v2 [cs.RO], October 2016.
3. C. Finn, S. Levine, Deep Visual Forsight for Planning Robot Motion, arXiv:1610.00696, ICRA 2017, October 2016.
4. L. Pinto, J. Davidson, A. Gupta, Supervision via Competition: Robot Adversaries for Learning Tasks, arXiv:1610.01685, ICRA 2017, October 2016.
5. K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, D. Krishnan, Unsupervised Pixel–Level Domain Adaptation with Generative Adversarial Networks, arXiv:1612.05424, CVPR 2017, December 2016.
6. A. Banino et al., Vector-based navigation using grid-like representations in artificial agents, https://doi.org/10.1038/s41586-018-0102-6, Research Letter, Nature, 2018.
7. R. Borst, Robust self-balancing robot mimicking, Bachelor Thesis, August 2017
8. W. Zhao, J.P. Queralta, T. Westerlund, Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey, 2020 IEEE Symposium Series on Computational Intelligence (SSCI), 2020.

Algorithm	Description	Model	Policy	Action Space	State Space	Operator
Monte Carlo	Every visit to Monte Carlo	Model-Free	Off-policy	Discrete	Discrete	Sample-means
Q-learning	State–action–reward–state	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA	State–action–reward–state–action	Model-Free	On-policy	Discrete	Discrete	Q-value
Q-learning - Lambda	State–action–reward–state with eligibility traces	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA - Lambda	State–action–reward–state–action with eligibility traces	Model-Free	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q Network	Model-Free	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	Model-Free	On-policy	Continuous	Continuous	Advantage
NAF	Q-Learning with Normalized Advantage Functions	Model-Free	Off-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor-Critic	Model-Free	Off-policy	Continuous	Continuous	Advantage


Environment
Agent: Robot
State
Action
Reward