Robot Learning and Control #5

Multi-agent Reinforcement Learning

Kazuki Shibata

Robot Learning Lab

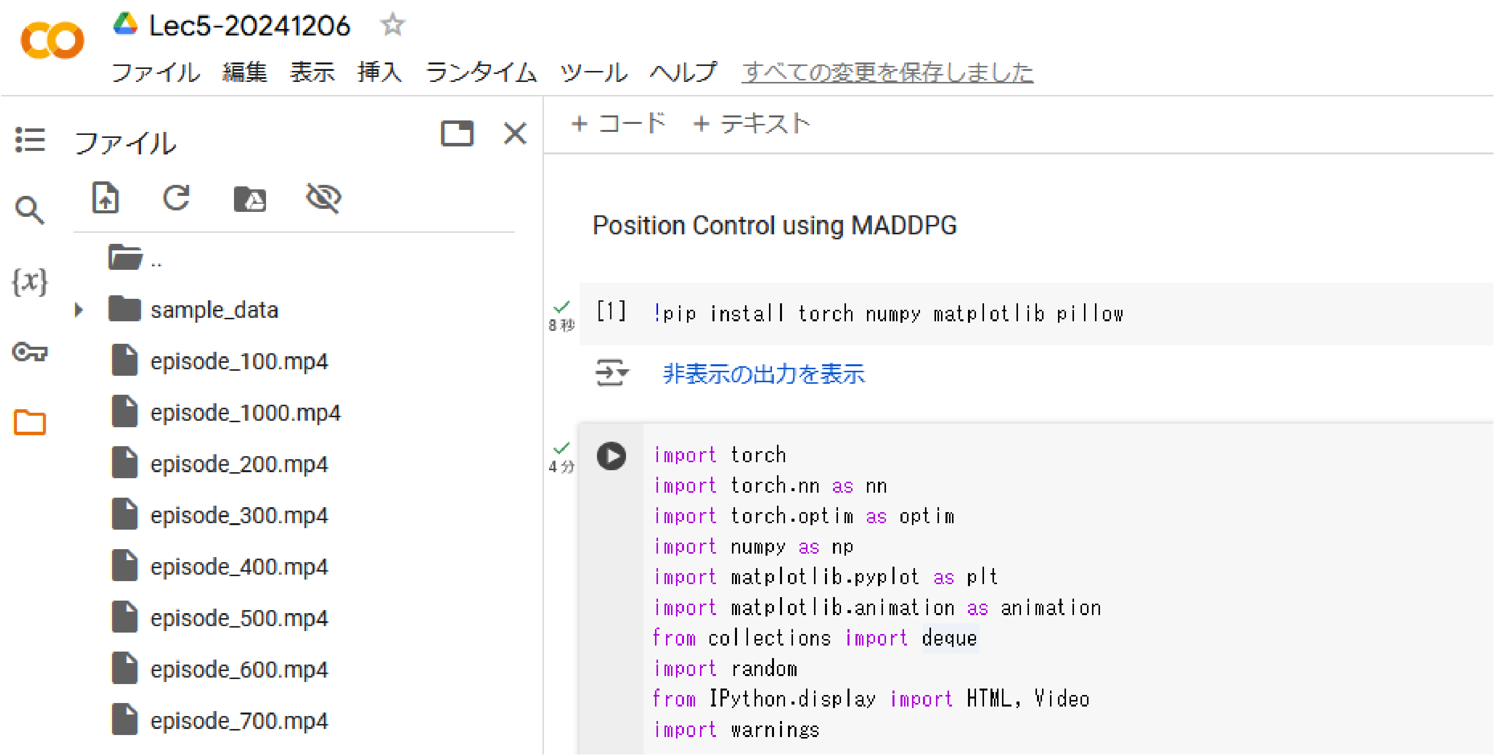
Dec. 6, 2024

Today’s Lecture

Baker+, ICLR2020

Today´s Notebook

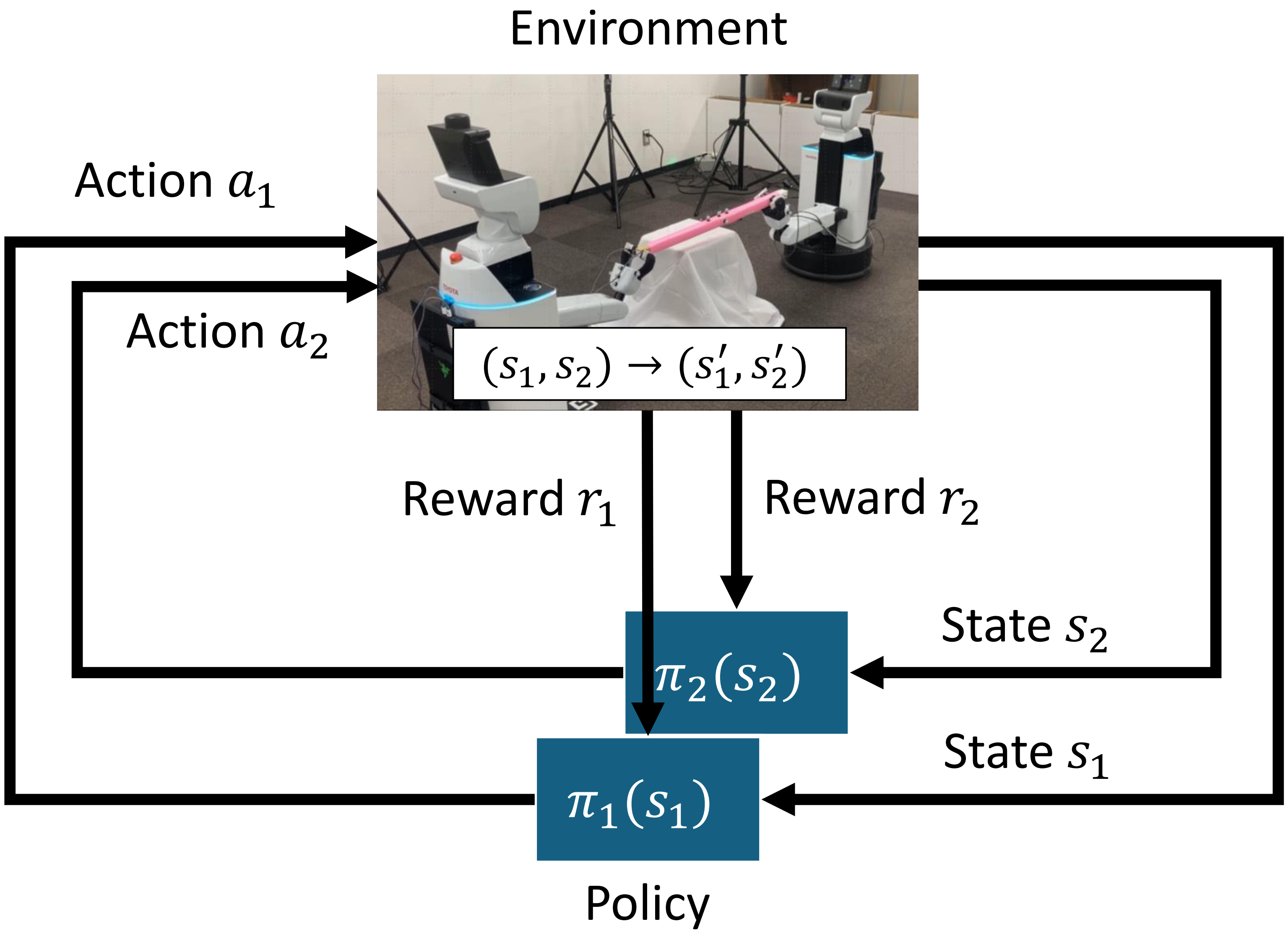
This lecture utilizes Google Colab to run example codes developed by Shibata.



URL: https://colab.research.google.com/drive/1JnRMb3haAvYW9eTZAwOCcDVwKQGlVc-0?usp=sharing

Table of Contents

Multi-agent Reinforcement Learning (MARL)



MARL Problem

The MARL problem involves finding policies for multiple robots to maximize the expected future rewards.

The objective is to derive a policy parameter $\theta_i (i = 1, \cdots, N)$ that maximizes the expected cumulative reward for robot i :

$$J(\theta_i) = E\left[\sum_{k=0}^{\infty} \gamma^k r_i(t+k)\right] \tag{1}$$

where r_i is immediate reward, and γ $(0 < \gamma \leq 1)$ is a discount factor that determines how much future rewards are discounted compared to the immediate reward.

Value Function

State-action value function $Q(s,a)$: The expected cumulative reward obtained by taking a specific action a in state s

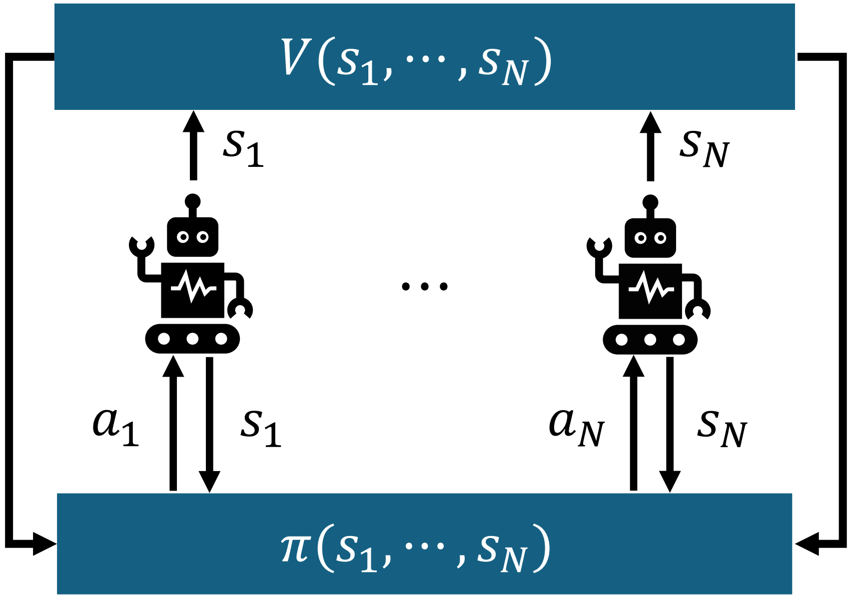
$$Q(s,a) = E\left[\sum_{k=0}^{\infty} \gamma^k r(t+k) | s, a\right] \tag{2}$$

State value function $V(s)$: The expected cumulative reward in state s

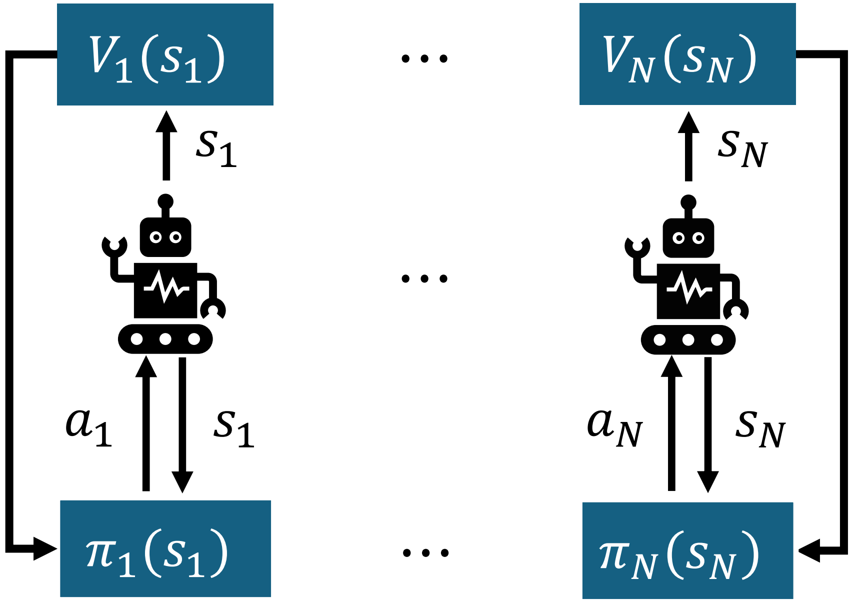
$$V(s) = E\left[\sum_{k=0}^{\infty} \gamma^k r(t+k) | s\right] \tag{3}$$

Both value functions are utilized to evaluate and improve the policy.

MARL Approaches



Central policy in Server



Policy 1 Policy N

<u>Centralized Training</u>

✓ Stable training owing to full observation

✗ Unapplicable if the central policy stops

<u>Decentralized Training</u>

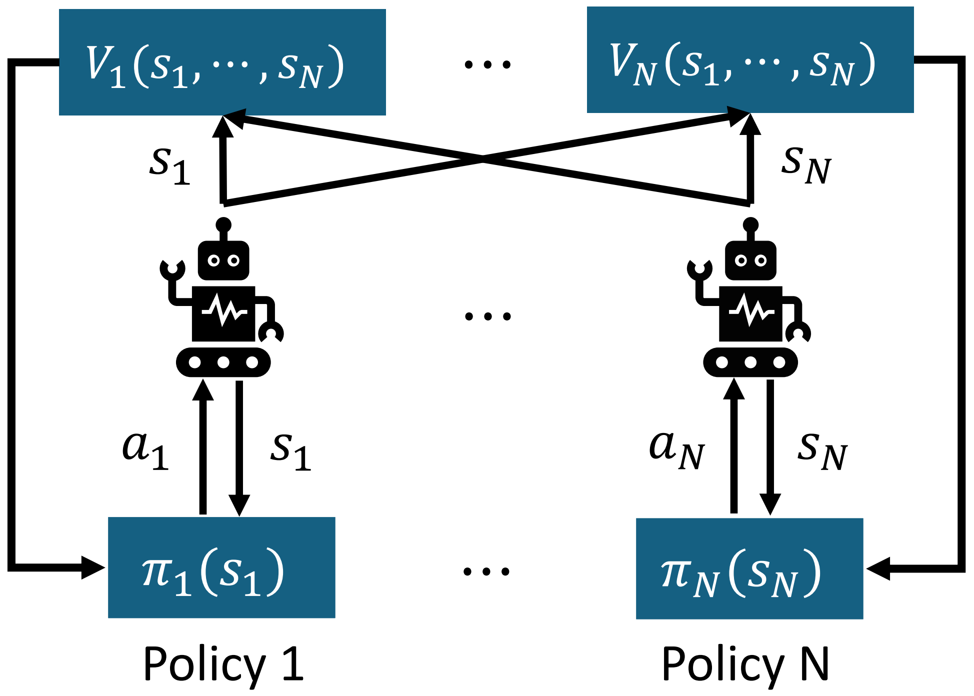
✓ Applicable without the central policy

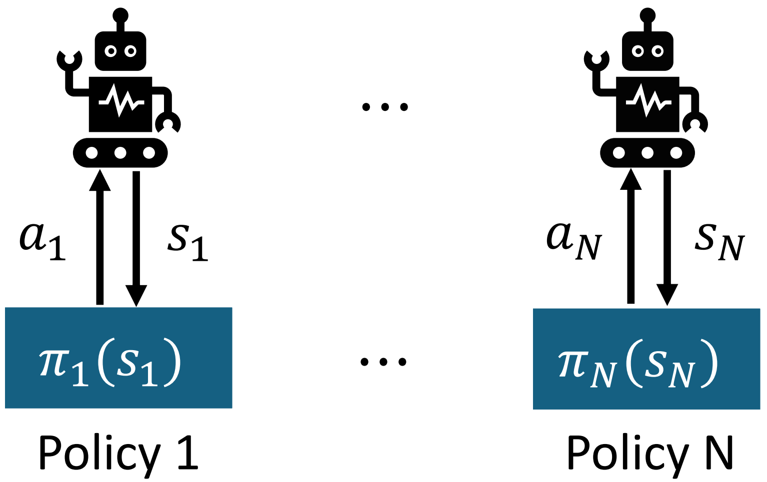
✗ Unstable training due to partial observation

How can we utilize the advantages of both approaches?

Table of Contents

Centralized Training and Decentralized Execution (CTDE)





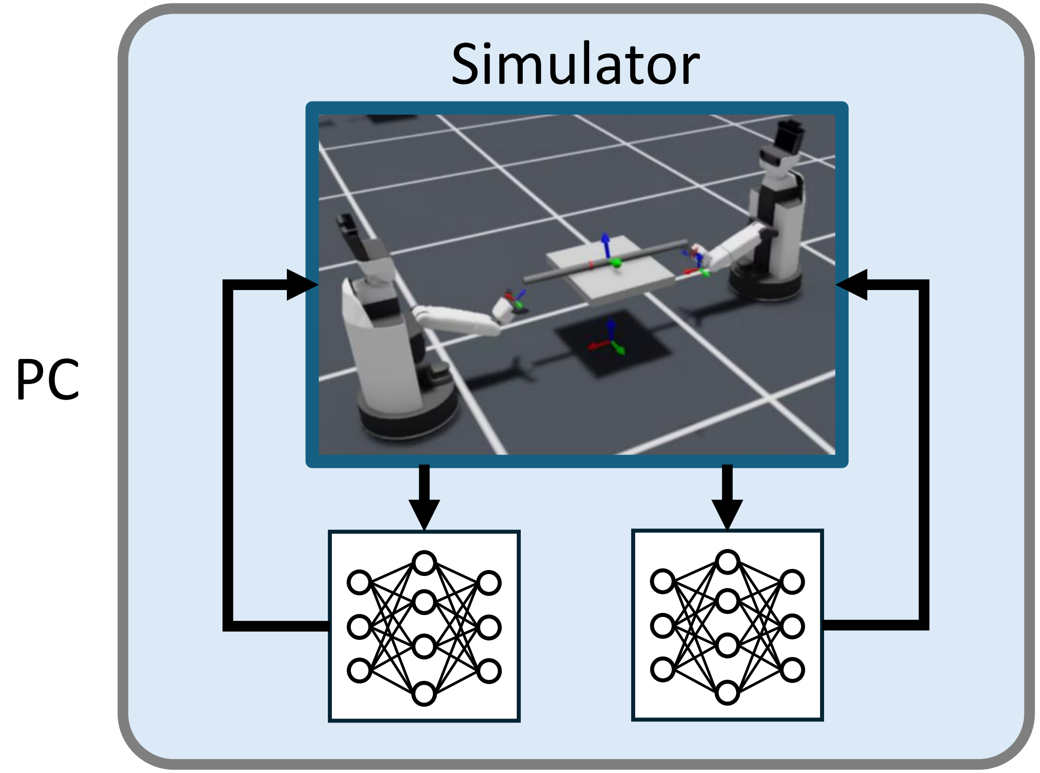
Centralized Training

- To utilize the states and actions of **all** robots during training phase

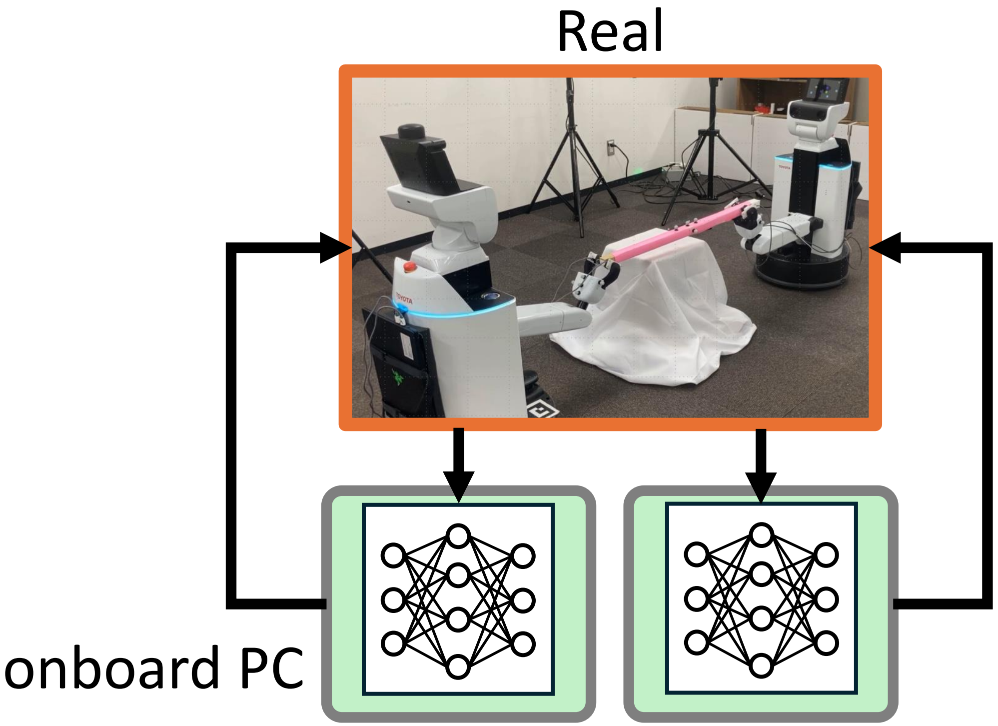
Decentralized Execution

- To utilize the state of **each** robot during execution phase

CTDE and Robot Learning



Training



Execution

If we conduct training in a simulator, we can utilize information about other robots.
In the real experiment, the robot computes action using its own observation.

Exercise 5-1: MARL to Robot Application

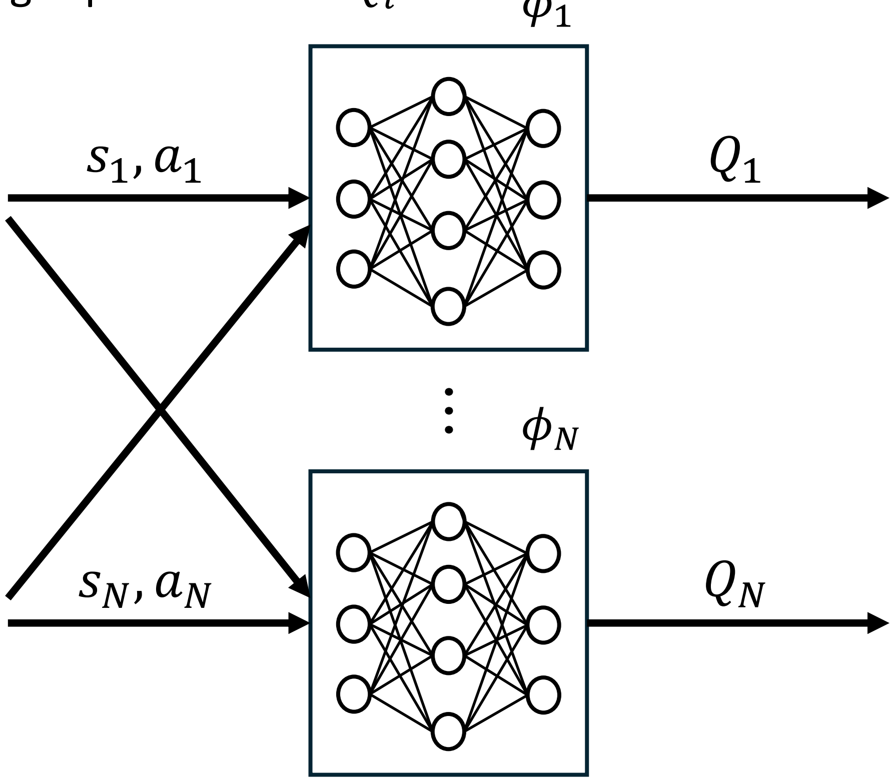
1. Consider issues when applying MARL to real robot tasks.
2. Consider ideas for solving above issues.

- You now have about 8 minutes to write up your report
- Write your name and student number in your report

Table of Contents

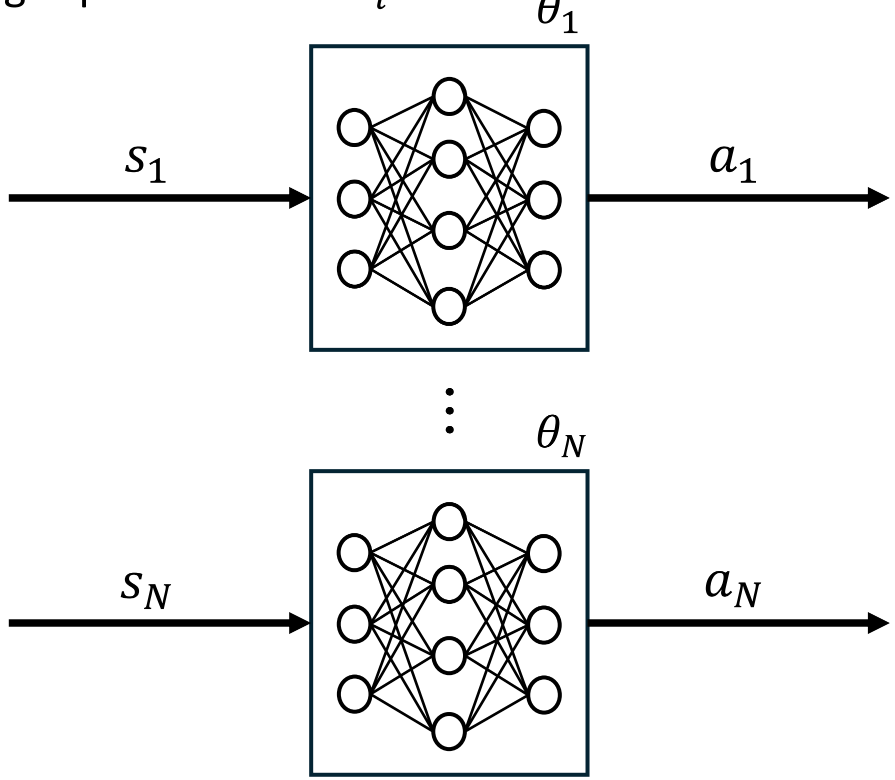
Multi-agent Deterministic Policy Gradient (MADDPG)

ϕ_i: weight parameter of Q_i



Critic Network (= Value Function)

θ_i: weight parameter of π_i



Actor Network (= Policy)

Let's confirm how to update these networks.

Policy Update

The objective of MADDPG is to derive a policy that maximizes the following objective function:

$$J(\theta_i) = E[Q_i(s, a)] \tag{4}$$

where $s \coloneqq [s_1, \cdots, s_N]$ is joint state of N robots, $a \coloneqq [a_1, \cdots, a_N]$ is joint action of N robots.

By taking the gradient w.r.t θ_i, we obtain:

$$\nabla_{\theta_i} J(\theta_i) = E\left[\nabla_{\theta_i} Q_i(s, a)\right] \tag{5}$$

Using the chain rule, this can be further expanded as:

$$\nabla_{\theta_i} J(\theta_i) = E\left[\nabla_{\theta_i} \pi_{\theta_i}(s_i) \nabla_{a_i} Q_i(s, a)\right] \tag{6}$$

This decomposition allows the gradients of the policy and the value function to be computed separately, making the gradient calculations simpler.

con’t

Next, we consider updating the weight parameter of the actor network θ_i.

Using the gradient $\nabla_{\theta_i} J(\theta_i)$, θ_i is updated as follows:

$$\theta_i \leftarrow \theta_i + \alpha \nabla_{\theta_i} J(\theta_i) \tag{7}$$

where $\alpha > 0$ is the learning rate.

To ensure stable learning, MADDPG utilizes the target network to update the network gradually.

The target network parameters are updated as follows:

$$\theta_i' \leftarrow \tau\theta_i + (1 - \tau)\theta_i' \tag{8}$$

where θ_i' is the weight parameter of target network, and $\tau (0 < \tau \ll 1)$ controls the update rate.

The use of the target network stabilizes training by introducing a first-order time-delay system.

Value Function Update

Next, we consider updating the value function.

State-action value function is written in recursive form:

$$Q_i(s,a) = E\left[r_i(t) + \gamma \sum_{k=0}^{\infty} \gamma^k r_i(t+k+1) | s, a\right] \tag{9}$$

Using the law of total expectation $E[X] = E[E[X|Y]]$, the Bellman equation for the state-action value function can be derived as:

$$Q_i(s,a) = E\left[r_i(t) + \gamma E\left[\sum_{k=0}^{\infty} \gamma^k r_i(t+k+1) \middle| s', a'\right] \middle| s, a\right] \tag{10}$$

$$= E[r_i(t) + \gamma Q_i(s', a')] \tag{11}$$

con't

Define a target value function y_i as follows:

$$y_i = r_i(t) + \gamma Q_i(s', a') \tag{12}$$

The objective is to train the critic network to bring the predicted $Q_i(s, a)$ closer to y_i.

This can be achieved by minimizing the following loss function:

$$L(\phi_i) = E\left[\left(y_i - Q_i(s, a)\right)^2\right] \tag{13}$$

The weight parameter of the critic network ϕ_i is updated as follows:

$$\phi_i \leftarrow \phi_i - \beta \nabla_{\phi_i} L(\phi_i) \tag{14}$$

Similarly, the critic network is updated using a target network to ensure stable learning:

$$\phi_i' \leftarrow \tau \phi_i + (1 - \tau)\phi_i' \tag{15}$$

Experience Replay: How to utilize the experience data?

MADDPG implements experience replay, which utilizes past experiences stored in a replay buffer:

$$D = \{(s, a, s', r)\} \tag{16}$$

To train efficiently and stably from past experiences, a mini-batch B, a small random subset of experiences, is sampled from D.

Using the mini-batch, the policy gradient is computed as:

$$\nabla_{\theta_i} J(\theta_i) \approx \frac{1}{|B|} \sum_{(s,a,s',r)\sim B} \nabla_{\theta_i} \pi_{\theta_i}(s_i) \nabla_{a_i} Q_i(s, a) \tag{17}$$

Similarly, the critic loss function is computed as:

$$L(\phi_i) \approx \frac{1}{|B|} \sum_{(s,a,s',r)\sim B} \left(y_i - Q_i(s, a)\right)^2 \tag{18}$$

Simulation: Position Control using MADDPG

Objective: To control the positions of two agents to the goal positions

- State s_i

$$s_i = p_i \tag{19}$$

p_i: Position of agent i

- Action a_i

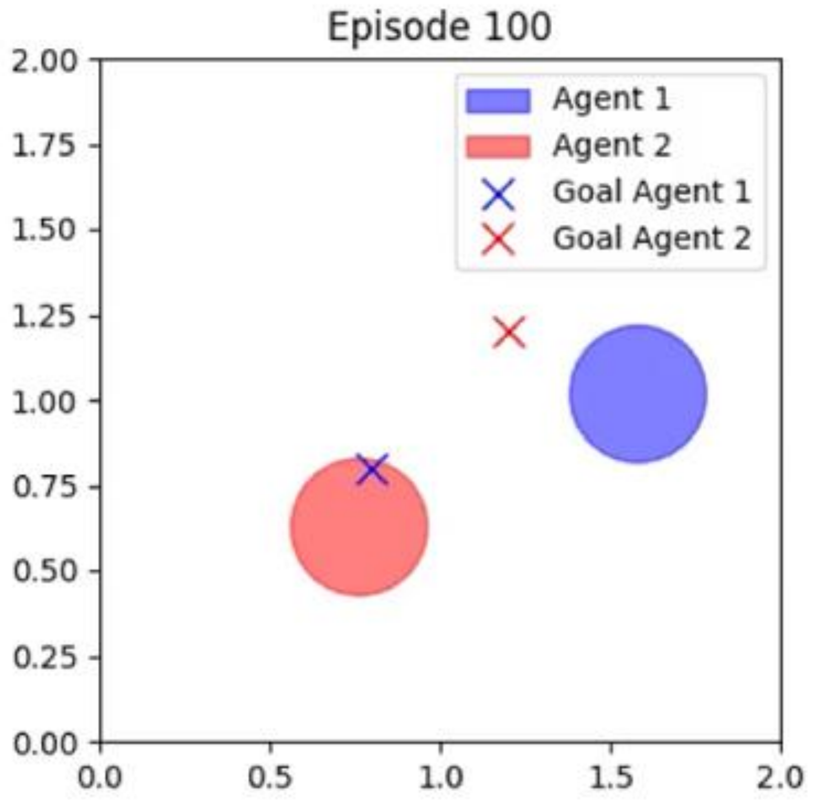
$$a_i = u_i \tag{20}$$

u_i: Velocity input of agent i

- Reward r_i

$$r_i = -|g_i - p_i| \tag{21}$$

g_i: Goal position of agent i

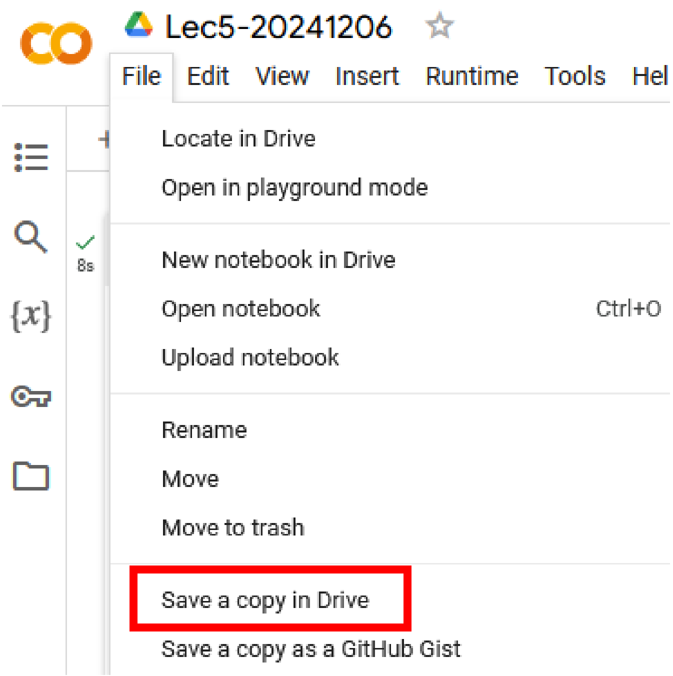


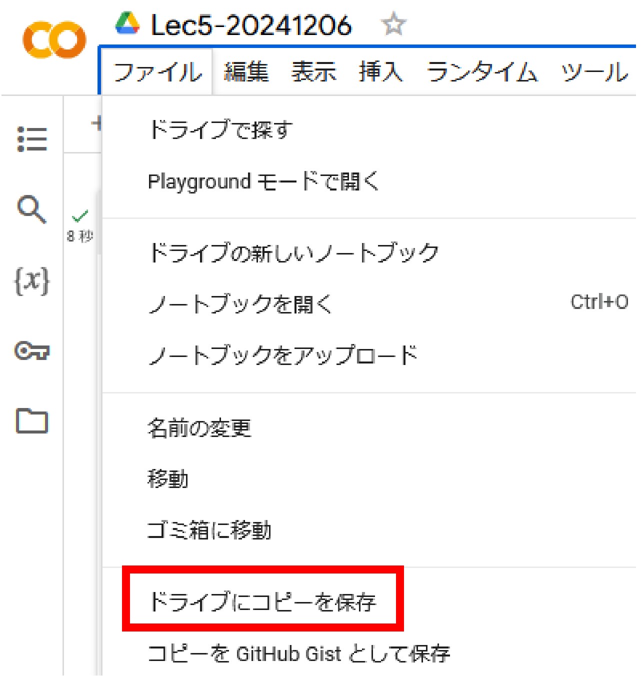
How to use the Google Colab

1. Access Google Colab through the URL below

https://colab.research.google.com/drive/1JnRMb3haAvYW9eTZAwOCcDVwKQGlVc-0?usp=sharing

2. Save to your Google Drive





Exercise 5-2: MADDPG

Please design the observation and reward so that each agent can reach the goal position while reducing the number of collisions.

Please try following steps:

1. Add the collision penalty

```
collision_penalty = 0 if distance_between_agents < 2 * agent_radius else 0
```

2. Add the state of the other agent

- You have 20 min to implement your code
- If you have any trouble, please feel free to ask.

Exercise 5-2: Answer: MADDPG

Let's confirm Google Colab through the URL below.

https://colab.research.google.com/drive/1i_6_4e-OdH48f8MzpzYLz_uv2k8BfvaW?usp=sharing

Code. 1: Partial Observation without Collision Penalty

$$s_i = p_i$$

$$r_i = -|g_i - p_i|$$

Code. 2: Partial Observation with Collision Penalty

$$s_i = p_i$$

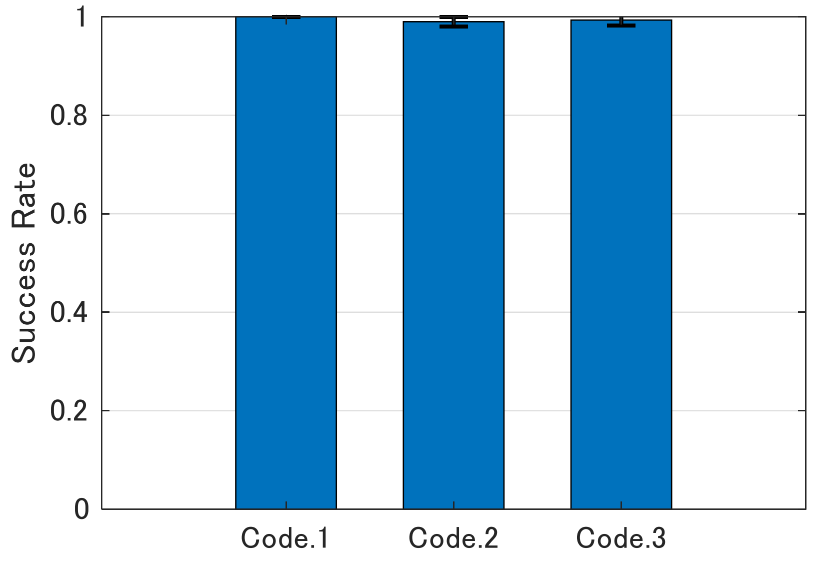
$$r_i = -|g_i - p_i| - \mathrm{col}$$

Code. 3: Full Observation with Collision Penalty

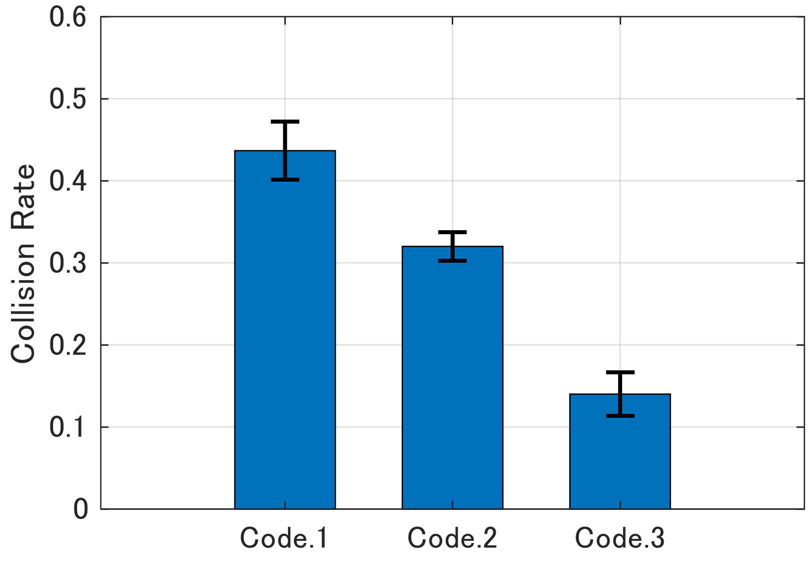
$$r_i = -|g_i - p_i| - \mathrm{col}$$

$$\mathrm{s}_i = [p_1, p_2]$$

Exercise 5-2: Answer: MADDPG



Success Rate (3 trainings)



Collision Rate (3 trainings)

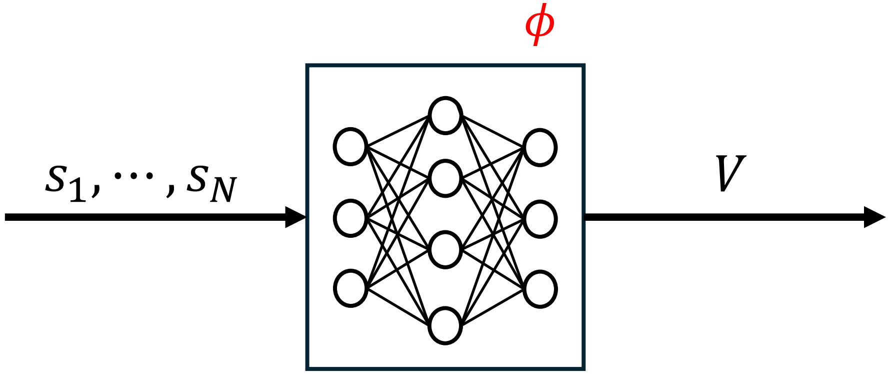
The collision rate in Code. 2 is higher than in Code. 3 because the agent is unaware of the other robot's position. This scenario corresponds to "**partial observation**" in MARL.

Table of Contents

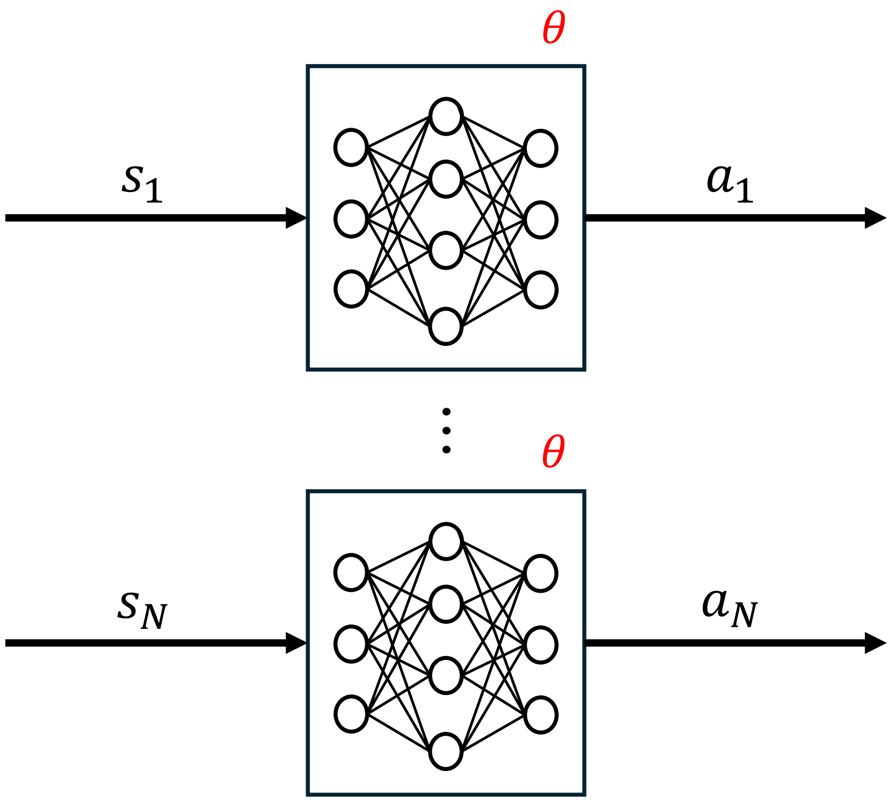
Multi-agent Proximal Policy Optimization (MAPPO)

<u>Parameter sharing</u>

Sharing the same parameter for all agents can improve the learning efficiency.



Critic Network (= Value Function)



Actor Network (= Policy)

Let's confirm how to update these networks.

Advantage Function

The advantage function $A(s,a)$ measures how much a specific action a outperforms the average action in state s. It is defined as:

$$A(s,a) \coloneqq Q(s,a) - V(s) \tag{22}$$

Let's consider following cases.

- $A(s,a) > 0$: Action a is better than the average action in state s.
- $A(s,a) < 0$: Action a is worse than the average action in state s.

Maximizing the advantage function $A(s,a)$ ensures that the policy selects action that outperforms the average action, thereby improving the policy.

Policy Update

MAPPO aims to maximize the following objective function:

$$J(\theta) = \sum_{i=1}^{N} E[A(s_i, a_i)] = \sum_{i=1}^{N} \int \pi_\theta(a_i|s_i) A(s_i, a_i) dsda \tag{23}$$

Take the derivative of $J(\theta)$ w.r.t θ, we obtain:

$$\nabla_\theta J(\theta) = \sum_{i=1}^{N} \int \nabla_\theta \pi_\theta(a_i|s_i) A(s_i, a_i) dsda \tag{24}$$

In MAPPO, the policy ratio is used to measure how much the current policy outperforms the previous policy as follows:

$$\nabla_\theta J(\theta) = \sum_{i=1}^{N} \int \nabla_\theta \pi_{\theta_{\mathrm{old}}}(a_i|s_i) \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\mathrm{old}}}(a_i|s_i)} A(s_i, a_i) dsda \tag{25}$$

con't

MAPPO maximizes the following objective function:

$$L(\theta) = \sum_{i=1}^{N} E\left[\frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\text{old}}}(a_i|s_i)} A(s_i, a_i)\right] \tag{26}$$

To ensure the stability of training, MAPPO employs a clipped objective:

$$L^{\text{CLIP}}(\theta) = \sum_{i=1}^{N} E\left[\min\left(\frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\text{old}}}(a_i|s_i)} A(s_i, a_i), \text{clip}\left(\frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\text{old}}}(a_i|s_i)}, 1-\epsilon, 1+\epsilon\right) A(s_i, a_i)\right)\right] \tag{27}$$

The clip function is defined as:

$$\text{clip}(x, 1-\epsilon, 1+\epsilon) = \begin{cases} x, & \text{if } 1-\epsilon < x < 1+\epsilon \\ 1-\epsilon, & \text{if } x < 1-\epsilon \\ 1-\epsilon, & \text{if } x > 1+\epsilon \end{cases} \tag{28}$$

The weight parameter of the actor network is updated as follows:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L^{\text{CLIP}}(\theta) \tag{29}$$

Value Function Update

Next, we consider updating the state value function $V(s_i)$.

State value function is written in recursive form:

$$V(s_i) = E\left[r_i(t) + \gamma \sum_{k=0}^{\infty} \gamma^k r_i(t+k+1) \middle| s_i\right] \tag{30}$$

Using the law of total expectation $E[X] = E[E[X|Y]]$, the Bellman equation for the state value function can be derived as:

$$V(s_i) = E\left[r_i(t) + \gamma E\left[\sum_{k=0}^{\infty} \gamma^k r_i(t+k+1) \middle| s_i'\right] \middle| s_i\right] \tag{31}$$

$$= E[r_i(t) + \gamma V(s_i')] \tag{32}$$

con't

Define a target value function y_i as follows:

$$y_i = r_i + \gamma V(s_i') \tag{33}$$

The objective is to learn the critic network so that the predicted value can get closer to y_i.

This can be achieved by minimizing the following loss function:

$$L(\phi) = E\left[\left(y_i - V(s_i)\right)^2\right] \tag{34}$$

Similarly, clipping is used to prevent the state value function from changing abruptly.

$$L^{\text{CLIP}}(\phi) = \sum_{i=1}^{N} E[\max([(V(s_i) - y_i)^2], (\text{clip}(V(s_i), V_{\text{old}}(s_i) - \epsilon, V_{\text{old}}(s_i) + \epsilon) - y_i)^2)\,] \tag{35}$$

The weight parameter of the critic network is updated as follows:

$$\phi \leftarrow \phi - \beta \nabla_\phi L^{\text{CLIP}}(\phi) \tag{36}$$

Generalized Advantage Estimator (GAE)

The advantage function can be computed using the Bellman equation as:

$$A(s_i, a_i) = E[r_i(t) + \gamma V(s_i')] - V(s_i) \tag{37}$$

Define the TD Error

$$\delta_i(t) = r_i(t) + \gamma V(s_i') - V(s_i) \tag{38}$$

Using $\delta_i(t)$, the advantage function can be written as:

$$A(s_i, a_i) = E\left[\sum_{l=0}^{\infty} \gamma^l \delta_i(t+l)\right] \tag{39}$$

The GAE introduces a weight parameter λ $(0 < \lambda \leq 1)$

$$A(s_i, a_i) = E\left[\sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_i(t+l)\right] \tag{40}$$

where λ is used to reduce the influence of future TD errors, which stabilizes the training.

Exercise 5-3: GAE

Derive

$$A(s_i, a_i) = E\left[\sum_{l=0}^{\infty} \gamma^l \delta_i(t+l)\right]$$

using the following formula:

$$A(s_i, a_i) = E[r_i(t) + \gamma V(s_i')] - V(s_i)$$

$$\delta_i(t) = r_i(t) + \gamma V(s_i') - V(s_i)$$

- You now have about 8 minutes to write up your report

Exercise 5-3: Answer: GAE

Using the Bellman equation, we obtain:

$$A(s_i, a_i) = E[r_i(t) + \gamma V(s_i')] - V(s_i) \tag{41}$$

$$= E\left[r_i(t) + \gamma E[r_i(t+1) + \gamma V(s_i'')]\right] - V(s_i) \tag{42}$$

$$= E[r_i(t) + \gamma r_i(t+1) + \gamma^2 V(s_i'')] - V(s_i) \tag{43}$$

By repeating this process, we obtain:

$$A(s_i, a_i) = E\left[r_i(t) + \gamma r_i(t+1) + \cdots + \gamma^l r_i(t+l) + \gamma^{l+1} V\left(s_i^{l+1}\right)\right] - V(s_i) \tag{44}$$

where s_i^l represents the state l steps ahead of the current step.

Exercise 5-3: Answer: GAE

Using the TD error, we obtain:

$$\delta(t) + \gamma\delta(t+1) = r_i(t) + \gamma V(s_i') - V(s_i) + \gamma r_i(t+1) + \gamma^2 V(s_i'') - \gamma V(s_i') \quad (45)$$

$$= r_i(t) + \gamma r_i(t+1) + \gamma^2 V(s_i'') - V(s_i) \quad (46)$$

By repeating this process, we obtain:

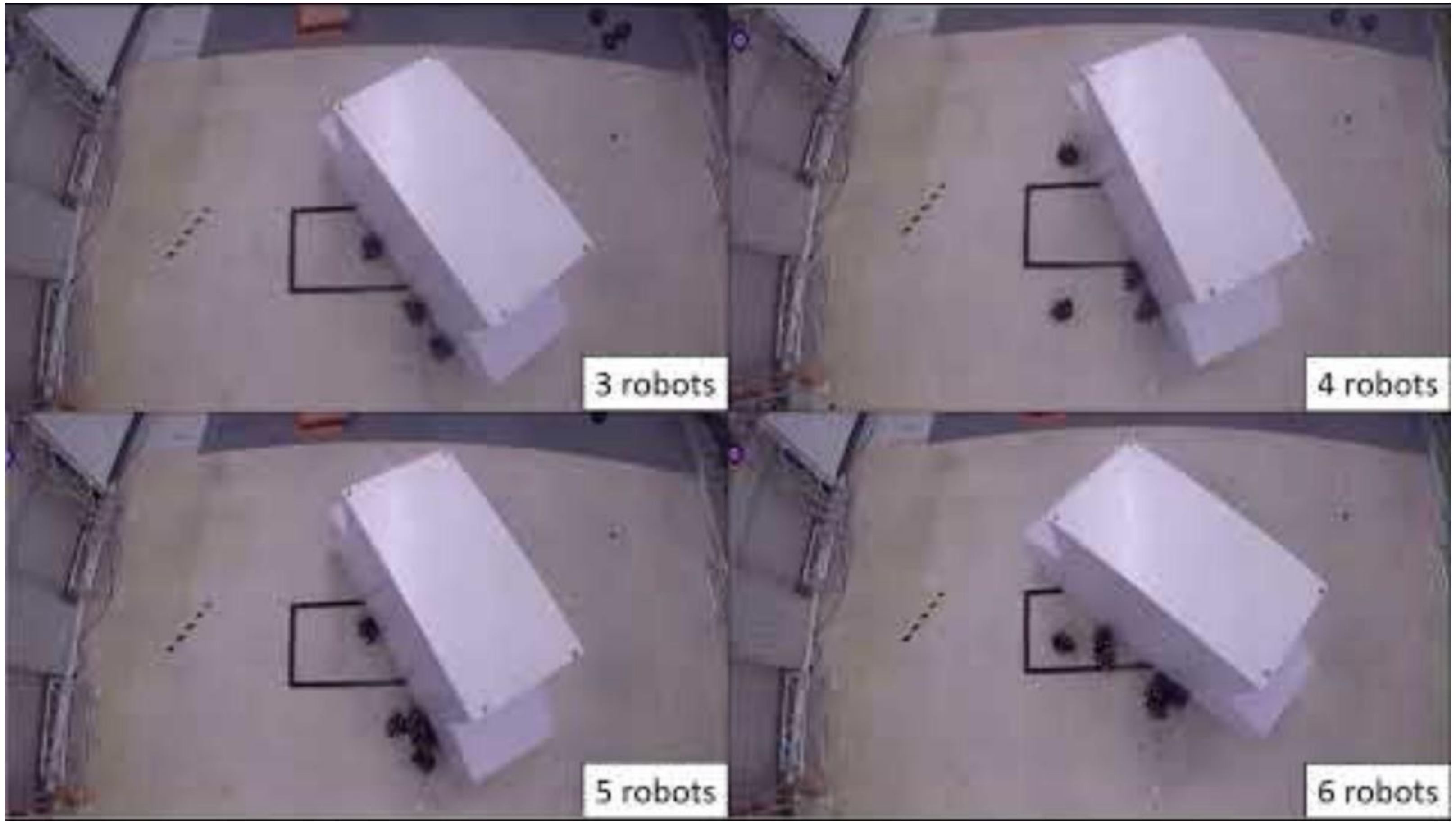
$$\delta(t) + \gamma\delta(t+1) + \cdots \gamma^l \delta(t+l)$$

$$= r_i(t) + \gamma r_i(t+1) + \cdots + \gamma^l r_i(t+l) + \gamma^{l+1} V\left(s_i^{l+1}\right) - V(s_i) \quad (47)$$

From (44) and (47), we obtain:

$$A(s_i, a_i) = E\left[\sum_{l=0}^{\infty} \gamma^l \delta_i(t+l)\right] \quad (48)$$

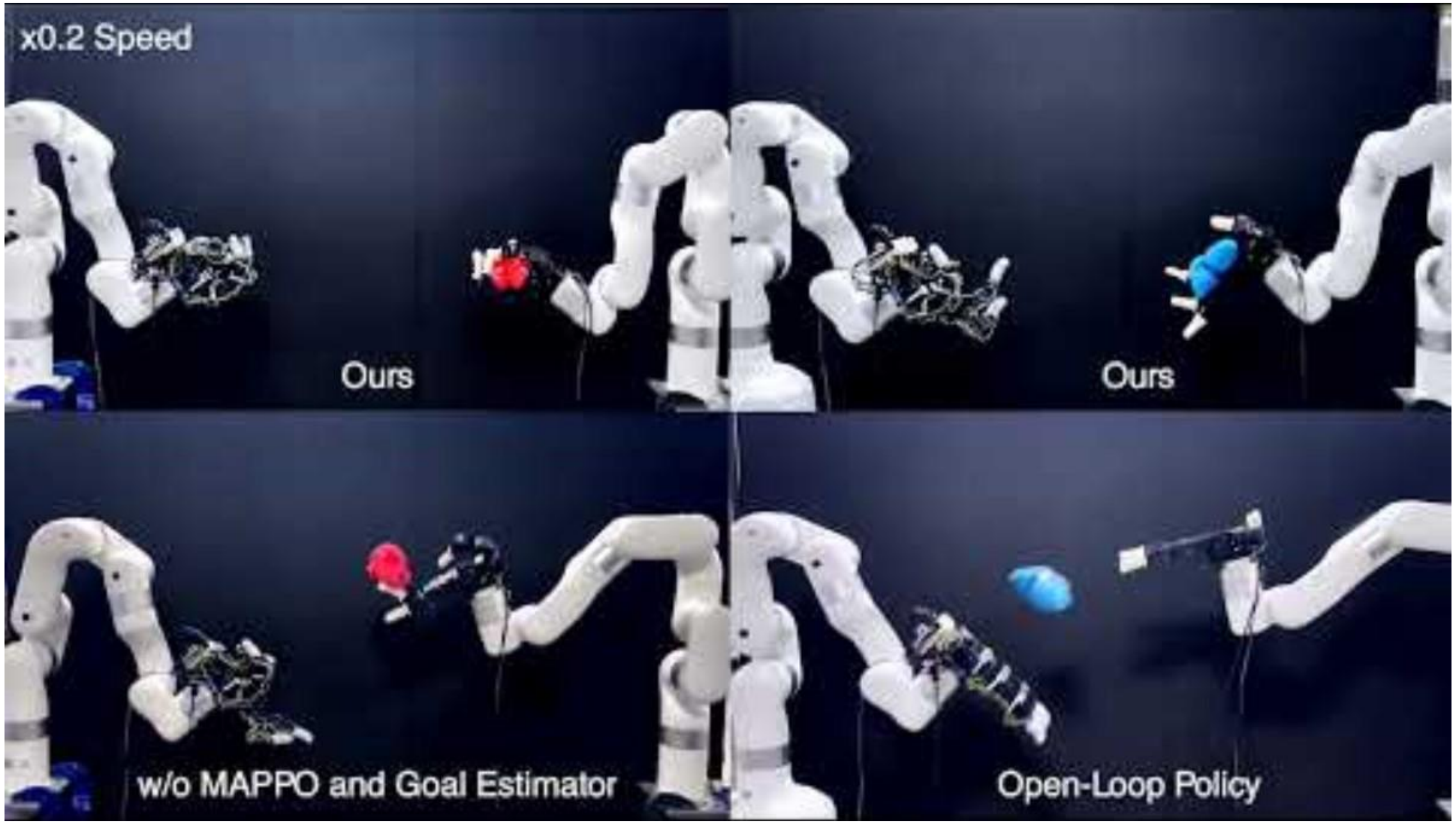
Robot Application: Communication and Control



Let's see youtube video

Shibata+, RAS2023

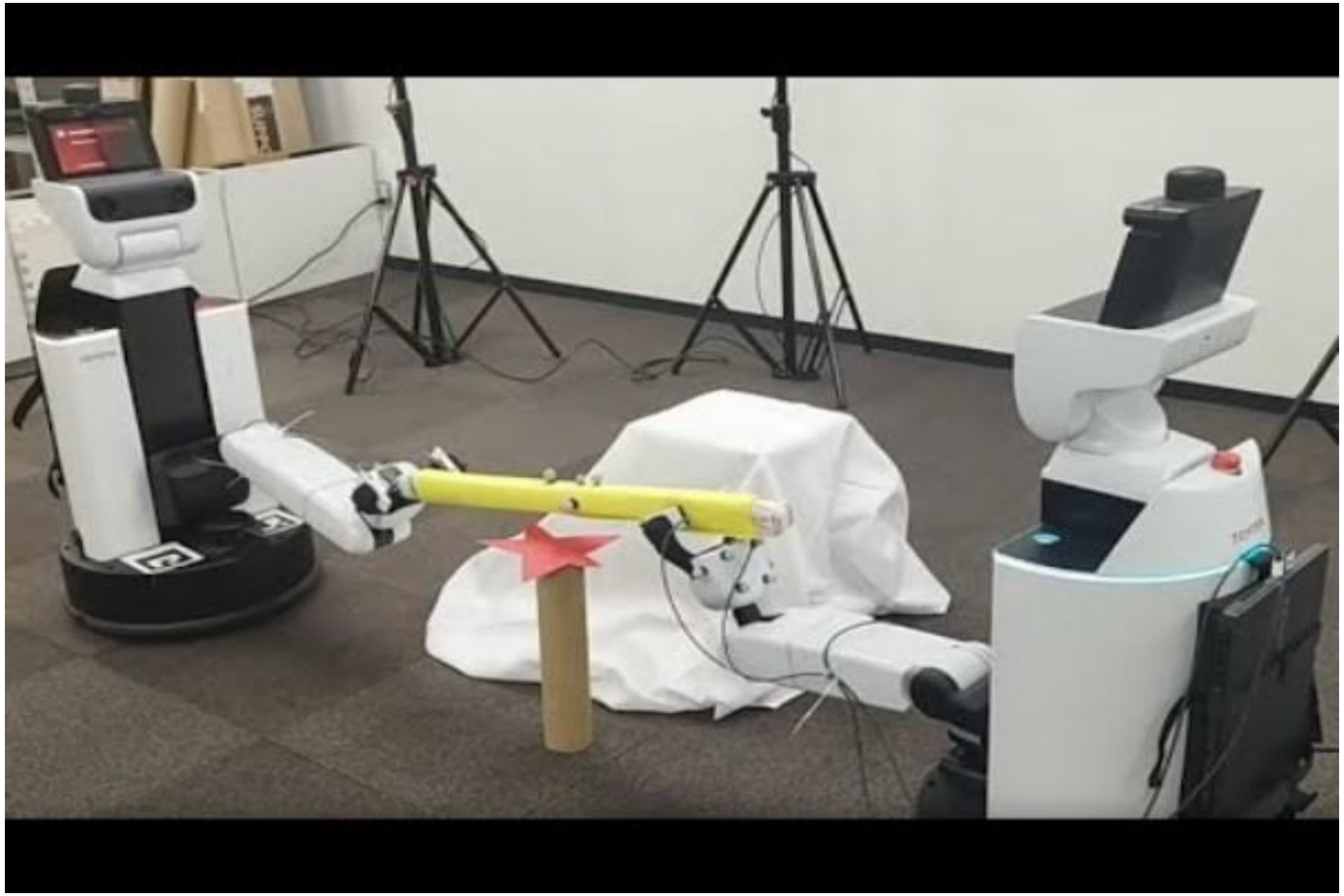
Robot Application: Sim2Real



Let's see youtube video

Huang+, CoRL2023

Robot Application: Sim2Real



Let’s see youtube video

Bernard+, SII2025

References

[1] B. Baker et al., “Emergent tool use from multi-agent autocurricula”, ICLR, 2020.

[2] R. Lowe et al., “Multi-agent actor-critic for mixed cooperative-competitive environments”, NeurIPS, pp. 6379-6390, 2017.

[3] C. Yu et al., “The surprising effectiveness of PPO in cooperative multi-agent games,” NeurIPS, pp. 24611-24624, 2022.

[4] K. Shibata et al., “Deep reinforcement learning of event-triggered communication and consensus-based control for distributed cooperative transport”, RAS, 159, 104307, 2023.

[5] B. Huang et al., "Dynamic handover: throw and catch with bimanual hands", CoRL, 2023.

[6] I. S. B. Tiong et al., : Cooperative grasping and transportation using multi-agent reinforcement learning with ternary force representation, IEEE/SICE SII, 2025.