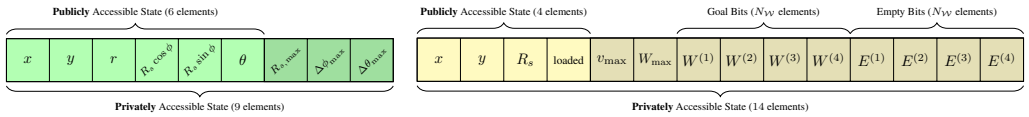
MATE: Benchmarking Multi-Agent Reinforcement Learning in Distributed Target Coverage Control

Supplementary Materials

A Camera and Target Observations - Detailed Formulation



(a) Overview on Camera States (b) Overview on Target States

Figure A.1: Schematic diagram of Agent States: the state of an agent contains two types of information, its status, and its ability, among which only the public information can be obtained by other agents.

The camera observation formulates as:

$$\boldsymbol{o}_c = \mathbb{C}\left(\left[\boldsymbol{s}^{\text{psrv}}, s_c^{\text{pvt}}, \mathbb{F}\left(\begin{bmatrix} \boldsymbol{s}_{t_1}^{\text{pub}} & 1 \\ \vdots & \vdots \\ \boldsymbol{s}_{t_{N_\mathcal{T}}}^{\text{pub}} & 1 \end{bmatrix} \otimes \begin{bmatrix} \text{mask} \\ \vdots \\ \text{mask} \end{bmatrix}\right), \mathbb{F}\left(\begin{bmatrix} \boldsymbol{s}_{o_1} & 1 \\ \vdots & \vdots \\ \boldsymbol{s}_{o_{N_\mathcal{O}}} & 1 \end{bmatrix} \otimes \begin{bmatrix} \text{mask} \\ \vdots \\ \text{mask} \end{bmatrix}\right), \mathbb{F}\left(\begin{bmatrix} \boldsymbol{s}_{c_1}^{\text{pub}} & 1 \\ \vdots & \vdots \\ \boldsymbol{s}_{c_{N_\mathcal{C}}}^{\text{pub}} & 1 \end{bmatrix} \otimes \begin{bmatrix} \text{mask} \\ \vdots \\ \text{mask} \end{bmatrix}\right)\right]\right), \tag{A.1}$$

where $\mathbb{C}$ means concatenation, $\mathbb{F}$ is the flatten operation, and $\otimes$ means element-wise multiplication. The mask in each row, it's a binary variable that indicates whether the entity is observable by the current camera. The mask variable becomes 1 as the camera can perceive that corresponding target or obstacle at the same row, otherwise, it's 0. In the observation, each publicly accessible state will be followed with an additional binary flag, which indicates whether the values of the state are valid.

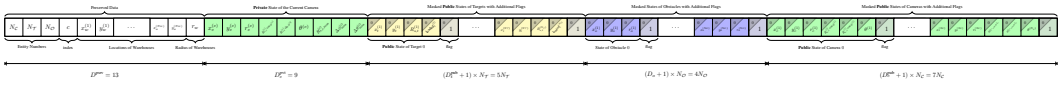


Figure A.2: Overview on Camera Observations

The target observation is defined by the following:

$$\boldsymbol{o}_t = \mathbb{C}\left(\left[\boldsymbol{s}^{\text{psrv}}, s_t^{\text{pvt}}, \mathbb{F}\left(\begin{bmatrix} \boldsymbol{s}_{c_1}^{\text{pub}} & 1 \\ \vdots & \vdots \\ \boldsymbol{s}_{c_{N_\mathcal{C}}}^{\text{pub}} & 1 \end{bmatrix} \otimes \begin{bmatrix} \text{mask} \\ \vdots \\ \text{mask} \end{bmatrix}\right), \mathbb{F}\left(\begin{bmatrix} \boldsymbol{s}_{o_1} & 1 \\ \vdots & \vdots \\ \boldsymbol{s}_{o_{N_\mathcal{O}}} & 1 \end{bmatrix} \otimes \begin{bmatrix} \text{mask} \\ \vdots \\ \text{mask} \end{bmatrix}\right), \mathbb{F}\left(\begin{bmatrix} \boldsymbol{s}_{t_1}^{\text{pub}} & 1 \\ \vdots & \vdots \\ \boldsymbol{s}_{t_{N_\mathcal{T}}}^{\text{pub}} & 1 \end{bmatrix} \otimes \begin{bmatrix} \text{mask} \\ \vdots \\ \text{mask} \end{bmatrix}\right)\right]\right). \tag{A.2}$$

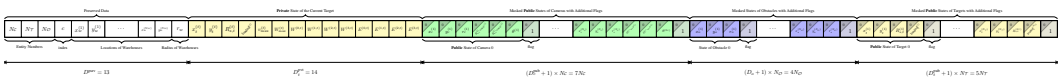


Figure A.3: Overview on Target Observations

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

B Built-in Wrappers and Agents

MATE ships with a variety of custom wrappers and built-in rule-based agents.

Table B.1: Built-in wrappers in MATE

Wrapper		Description
observation	`EnhancedObservation`	Enhance the agent's observation, which sets all observation mask to True. The empty bits for targets are set to truth values.
	`SharedFieldOfView`	Share field of view among agents in the same team.
	`MoreTrainingInformation`	Add more information to the `info` field of `step()`, enabling full observability of the environment.
	`RescaledObservation`	Rescale all entity states in the observation to $[-1, +1]$.
	`RelativeCoordinates`	Convert all locations of other entities in the observation to relative coordinates.
action	`DiscreteCamera`	Allow cameras to use discrete actions.
	`DiscreteTarget`	Allow targets to use discrete actions.
reward	`AuxiliaryCameraRewards`	Add auxiliary rewards for each individual camera.
	`AuxiliaryTargetRewards`	Add auxiliary rewards for each individual target.
single-team	`MultiCamera`	Wrap into a single-team multi-agent environment.
	`MultiTarget`	
	`SingleCamera`	Wrap into a single-team single-agent environment.
	`SingleTarget`	
communication	`MessageFilter`	Filter messages from agents of intra-team communications.
	`RandomMessageDropout`	Randomly drop messages in communication channels.
	`RestrictedCommunicationRange`	Add a restricted communication range to channels.
	`NoCommunication`	Disable intra-team communications, i.e., filter out all messages.
	`ExtraCommunicationDelays`	Add extra message delays to communication channels.
miscellaneous	`RepeatedRewardIndividualDone`	Repeat the `reward` field and assign individual `done` field of `step()`, which is similar to Multi-Agent Particle Environment (MPE) [1].

Table B.2: Built-in rule-based agents in MATE

Rule-based Agent		Description
random	camera	Takes random action.
	target	Takes random action.
naive	camera	Rotates anti-clockwise with the maximum viewing angle.
	target	Visits all warehouses in turn.
greedy	camera	Arbitrarily tracks the nearest target.
	target	Arbitrarily runs towards the destination (the desired warehouse).
heuristic	camera	Greedily maximizes the heuristic scores as much as possible. All camera agents send their observations to the centralized controller (agent 0). Then the central controller sends the goal state (camera pose) to each agent.
	target	Greedy target agent with an additional drift speed, which escapes away from the center of the cameras' field of view.

We represent the sample code at the following that wraps the environment into a single-team multi-agent setting (targets against cameras with given policies) with discrete action spaces and individual dense rewards.

```python
import mate

env = mate.make('MultiAgentTracking-v0')
env = mate.DiscreteTarget(env, levels=5)
env = mate.MultiTarget(env, camera_agent=mate.GreedyCameraAgent(seed=0))
env = mate.RepeatedRewardIndividualDone(env)
env = mate.AuxiliaryTargetRewards(env,
                                  coefficients={'raw_reward': 1.0,
                                                'real_coverage_rate': -1.0,
                                                'normalized_goal_distance': -1.0,
                                                'sparse_delivery': 100.0,
                                                'soft_coverage_score': -1.0},
                                  reduction='none')  # non-shared individual reward

dones = [False] * env.num_teammates
joint_observation = env.reset()
while not all(dones):
    joint_action = env.action_space.sample()
    joint_observation, rewards, dones, infos = env.step(joint_action)
```

C Configuration Generator

MATE ships with a generator to generate configurations with reasonable camera placement that automatically adjusts based on the given number of entities. The configuration generator solves the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \max_{\boldsymbol{x}} \min_{\boldsymbol{c}_i} \|\boldsymbol{x} - \boldsymbol{c}_i\|_2^2, \\ \text{subject to} \quad & -1 \preceq \boldsymbol{x} \preceq +1, \\ & -1 \preceq \boldsymbol{c}_i \preceq +1, \\ & \boldsymbol{x}, \boldsymbol{c}_i \in \mathbb{R}^2,\ i = 1, \ldots, n \end{aligned} \tag{C.1}$$

where $\boldsymbol{c}_i \in [-1, +1] \times [-1, +1]$ is the location of camera i. The intuition is to cover the terrain with a given number of circles and optimize the radius to be as small as possible, i.e., keep the overlap area is as small as possible. A example result of the generator is shown in Fig C.1 and Fig C.2.

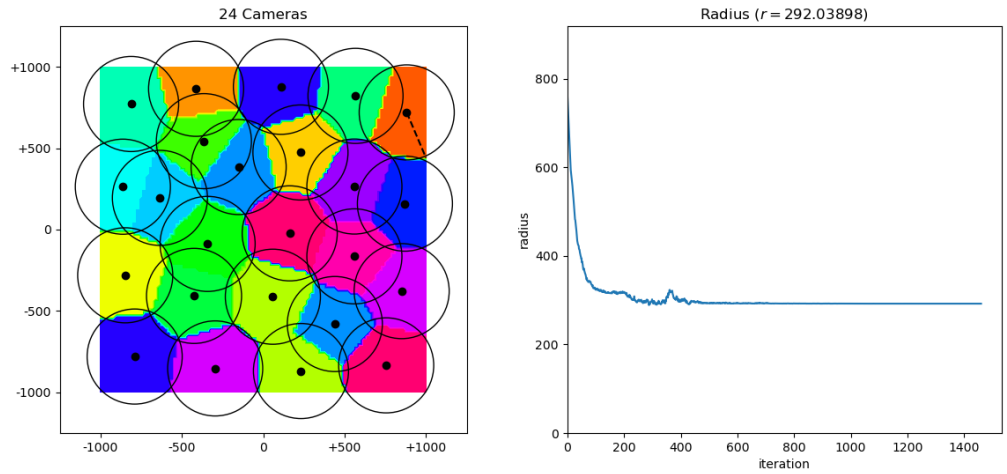


Figure C.1: An example camera placement of 24 cameras (the black dots).

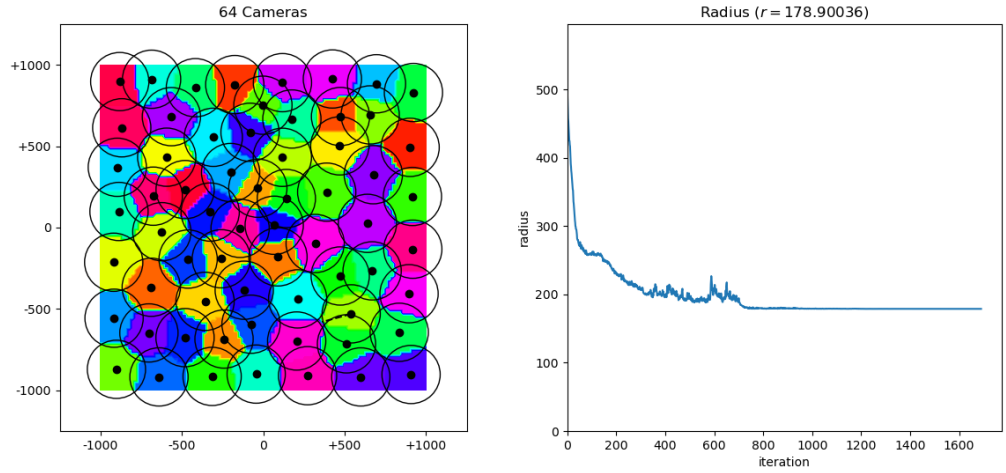


Figure C.2: An example camera placement of 64 cameras (the black dots).

D Scalability Benchmarking in Different Scales

We use a single-thread program to collect rollout samples on an Intel 12-core Core i7-8700 CPU @ 3.20GHz (only one core is used by the sampling program). The sampling performance is tested by the Python built-in module `timeit`. For each test, we sample 1000 steps with random actions. The result is averaged over 10 tests with separated seeds. The snippet to perform our test is shown below:

```
from timeit import timeit

MAX_STEPS = 1000
NUMBER = 10
CONFIG = 'MATE-4v8-0.yaml'

setup = f"""
import mate

env = mate.MultiAgentTracking('{CONFIG}')
env.seed(None)
env.reset()
"""

stmt = f"""
for i in range({MAX_STEPS}):
    env.step(env.action_space.sample())
"""

fps = MAX_STEPS * NUMBER / timeit(stmt, setup, number=NUMBER)
print(f'FPS = {fps}')
```

In our default configuration (4 cameras, 8 targets, 9 obstacles), the sampling worker can collect 372 steps per second on average on our test machine.

We use the configuration generator to generate multiple configurations with different numbers of agents. All these configurations are generated with 9 obstacles. As shown in Fig. D.1, in the case of frames per second (FPS) is not less than 10, the MATE environment supports up to hundreds of agents to interact simultaneously. With multi-process concurrent sampling, the throughput of the MATE environment can be multiplied.

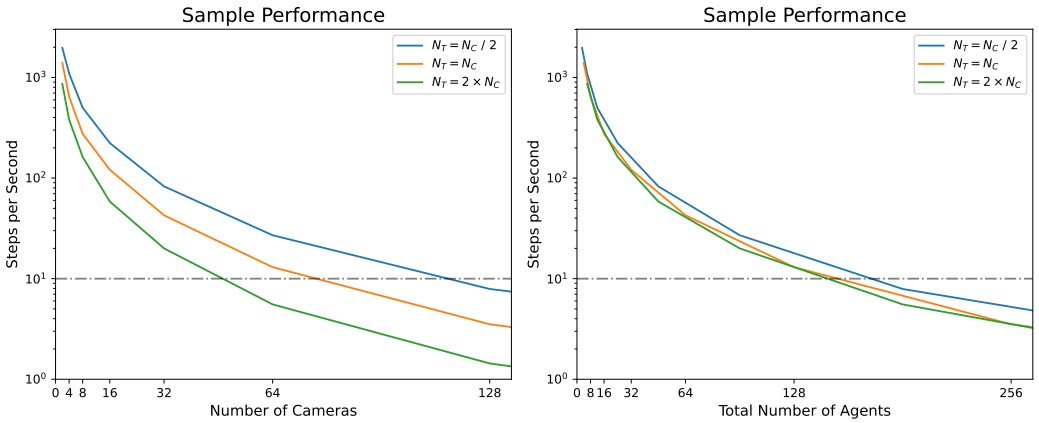


Figure D.1: Sample performance in different scales.

E Ablation of Communication on Rule-based Agents

In the ablation experiment of the communication network, we test our built-in rule-based greedy agents with and without communication in 100 cameras vs. 20 targets setting. In our rule-based greedy camera agent, the camera broadcasts its location to teammates at the first environment step. The recipients (other cameras) will use these messages to determine who their neighbors are (distance less than the twice maximum sight range). At other times, the camera only sends the detected target locations to its neighbors. For a camera agent, there is a random interval between two communications (up to 40 timesteps). Our rule-based cameras achieved a mean coverage of 69% without communication. After introducing the communication mechanism, the mean coverage rate of the camera network rose to 78%, gaining a performance improvement of nearly 10%.

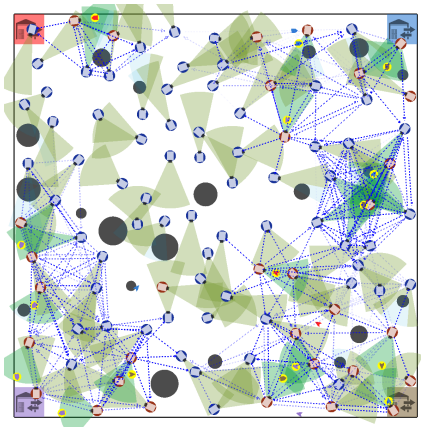


Figure E.1: A typical scene of communications in MATE.

F Additional Results in Zero-sum Fully-competitive Game

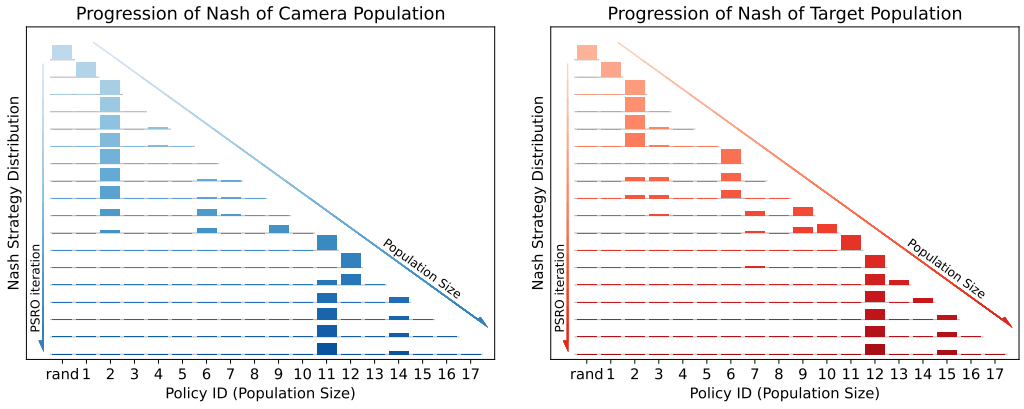


Figure F.1: The probability distribution of the meta-strategy for each PSRO [2] iteration for the policy population of both camera and target teams. Both teams adaptively change their policy according to the adversaries. The agents are trained under the 2C vs. 4T (OO) configuration.

G Additional Results in Training Camera Agents at Different Difficulty Levels

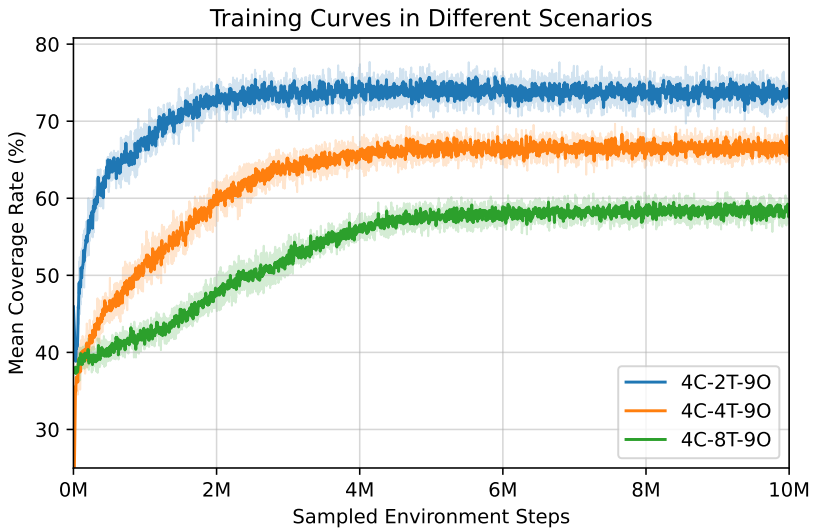


Figure G.1: A comparison between three different settings (easy (4C vs. 2T (0D)) to hard (4C vs. 8T (0D))) shows the significance of the number of opponents in convergence speed for the camera agents. The camera agents are controlled by the MAPPO [3] method with hierarchical control (MAPPO + HRL). The target agents are controlled by (greedy) rule-based agents.

H Comparison with the HiT-MAC [4] algorithm for Camera Agents

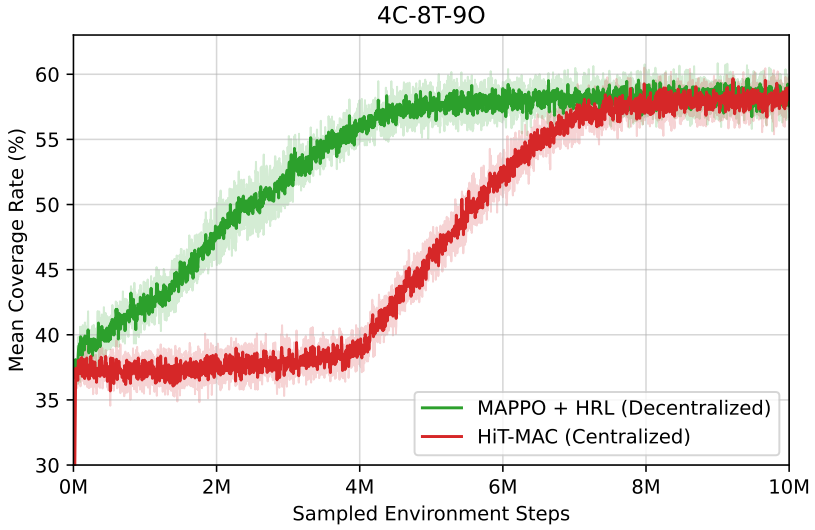


Figure H.1: A comparison between the HiT-MAC algorithm and MAPPO + HRL method for the camera agents in the 4C vs. 8T (0D)) scenario. The target agents are controlled by (greedy) rule-based agents.

The HiT-MAC [4] algorithm also utilizes a two-level hierarchical framework to solve the target control problem. However, its high-level policy is a centralized superagent. That requires modeling the joint policy with larger dimensions of observation and action space, which results in a harder exploration. The HiT-MAC algorithm results in slower convergence due to the computational complexity of MARL problems.

I Experiment Hyperparameters

In this section, unless specified, we use the default hyperparameters in the RLlib [5] implementation.

I.1 QMIX [6] Hyperparameters with RLlib

Table I.1.1: Common hyperparameters for QMIX

QMIX	
base_policy	QMIX
fcnet_hiddens & activation	[512, 256] & "relu"
mixer	"qmix"
mixing_embed_dim	128
use_lstm	True
lstm_cell_size	256
max_seq_len	10000
rollout_fragment_length	0
horizon	500
gamma	0.99
num_workers	16
num_envs_per_worker	8
batch_mode	"complete_episodes"
buffer_size	2000 (episodes)
train_batch_size	1024
grad_norm_clipping	1E3
target_network_update_freq	500
lr	1E-4
seed	[0, 1, 2]

Table I.1.2: Hyperparameters for QMIX Camera Agents

QMIX	4C vs. 2T (90) / 4C vs. 8T (90)
base_policy	QMIX
reward	coverage_rate
action_space	Discrete(25)
discrete_levels	5 (25 discrete actions)
frame_skip	5

Table I.1.3: Hyperparameters for QMIX+HRL Camera Agents

QMIX	4C vs. 2T (90) / 4C vs. 8T (90)
base_policy	QMIX+HRL
reward	coverage_rate
action_space	Discrete(4) / Discrete(256)
frame_skip	5

Table I.1.4: Hyperparameters for QMIX Target Agents

QMIX	2C vs. 4T (0O) / 4C vs. 8T (0O)
base_policy	QMIX
reward	raw_reward
action_space	Discrete(25)
discrete_levels	5 (25 discrete actions)
frame_skip	10

I.2 MADDPG [1] Hyperparameters with RLlib

Table I.2.5: Common hyperparameters for MADDPG

QMIX	
base_policy	TD3
actor_hiddens & activation	[512, 256] & "relu"
critic_hiddens & activation	[512, 256] & "relu"
vf_share_layers	False
max_seq_len	25
rollout_fragment_length	25
horizon	500
gamma	0.99
twin_q	True
policy_delay	2
smooth_target_policy	True
target_noise	0.2
target_noise_clip	0.5
n_step	1
num_workers	16
num_envs_per_worker	8
batch_mode	"truncate_episodes"
buffer_capacity	5E6
train_batch_size	1024
grad_norm_clipping	1E3
target_network_update_freq	0
tau	0.01
user_huber	True
huber_threshold	10
actor_lr	1E-4
critic_lr	1E-4
seed	[0, 1, 2]

Table I.2.6: Hyperparameters for MADDPG Camera Agents

MADDPG	4C vs. 2T (9O) / 4C vs. 8T (9O)
base_policy	MADDPG
reward	coverage_rate
action_space	Box([-5. -2.5], [5. 2.5], (2,))
frame_skip	5

Table I.2.7: Hyperparameters for MADDPG Target Agents

MADDPG	2C vs. 4T (OO) / 4C vs. 8T (OO)
base_policy	MADDPG
reward	raw_reward
action_space	Box([-20. -20.], [20. 20.], (2,))
frame_skip	10

I.3 IPPO [7] Hyperparameters with RLlib

Table I.3.8: Common hyperparameters for IPPO

IPPO	
base_policy	PPO
fcnet_hiddens & activation	[512, 256] & "relu"
vf_share_layers	False
use_lstm	True
lstm_cell_size	256
max_seq_len	25
rollout_fragment_length	25
horizon	500
gamma	0.99
use_critic	True
use_gae	True
clip_param	0.3
vf_clip_param	1E4
num_workers	16
num_envs_per_worker	8
batch_mode	"truncate_episodes"
train_batch_size	3200
sgd_minibatch_size	256
grad_norm_clipping	None
lr	5E-4
lr_schedule	[[0, 5E-4], [4E6, 5E-4], [4E6, 1E-4], [8E6, 1E-4], [8E6, 5E-5]]
entropy_coeff	0.05
entropy_coeff_schedule	[[0, 0.05], [2E6, 0.01], [4E6, 0.001], [10E6, 0.0]]
seed	[0, 1, 2]

Table I.3.9: Hyperparameters for IPPO Camera Agents

IPPO	4C vs. 2T (90) / 4C vs. 8T (90)
base_policy	IPPO
reward	coverage_rate
action_space	Discrete(25)
discrete_levels	5 (25 discrete actions)
frame_skip	5

Table I.3.10: Hyperparameters for IPPO+HRL Camera Agents

IPPO	4C vs. 2T (90) / 4C vs. 8T (90)
base_policy	IPPO+HRL
reward	coverage_rate
action_space	MultiDiscrete([2 2]) / MultiDiscrete([2 2 2 2 2 2 2 2])
frame_skip	5

Table I.3.11: Hyperparameters for IPPO Target Agents

IPPO	2C vs. 4T (00) / 4C vs. 8T (00)
base_policy	IPPO
reward	raw_reward
action_space	Discrete(25)
discrete_levels	5 (25 discrete actions)
frame_skip	10

I.4 MAPPO [3] Hyperparameters with RLlib

Table I.4.12: Common hyperparameters for MAPPO

MAPPO	
base_policy	PPO
actor_hiddens & activation	[512, 256] & "relu"
critic_hiddens & activation	[512, 256] & "relu"
vf_share_layers	False
use_lstm	True
lstm_cell_size	256
max_seq_len	25
rollout_fragment_length	25
horizon	500
gamma	0.99
use_critic	True
use_gae	True
clip_param	0.3
vf_clip_param	1E4
num_workers	16
num_envs_per_worker	8
batch_mode	"truncate_episodes"
train_batch_size	3200
sgd_minibatch_size	256
grad_norm_clipping	None
lr	5E-4
lr_schedule	[[0, 5E-4], [4E6, 5E-4], [4E6, 1E-4], [8E6, 1E-4], [8E6, 5E-5]]
entropy_coeff	0.05
entropy_coeff_schedule	[[0, 0.05], [2E6, 0.01], [4E6, 0.001], [10E6, 0.0]]
seed	[0, 1, 2]

Table I.4.13: Hyperparameters for MAPPO Camera Agents

MAPPO	4C vs. 2T (90) / 4C vs. 8T (90)
base_policy	MAPPO
reward	coverage_rate
action_space	Discrete(25)
discrete_levels	5 (25 discrete actions)
frame_skip	5

Table I.4.14: Hyperparameters for MAPPO+HRL Camera Agents

MAPPO	4C vs. 2T (90) / 4C vs. 8T (90)
base_policy	MAPPO+HRL
reward	coverage_rate
action_space	MultiDiscrete([2 2]) / MultiDiscrete([2 2 2 2 2 2 2 2])
frame_skip	5

Table I.4.15: Hyperparameters for MAPPO Target Agents

MAPPO	2C vs. 4T (00) / 4C vs. 8T (00)
base_policy	MAPPO
reward	raw_reward
action_space	Discrete(25)
discrete_levels	5 (25 discrete actions)
frame_skip	10

I.5 TarMAC [8] Hyperparameters with RLlib

Table I.5.16: Additional hyperparameters for TarMAC aside with the base policy

TarMAC	
message_dim	64
message_key_dim	32
message_value_dim	32

I.6 I2C [9] Hyperparameters with RLlib

Table I.6.17: Additional hyperparameters for I2C aside with the base policy

I2C	
message_dim	64
policy_corr_reg_coeff	0.01
temperature	0.1
prior_buffer_size	1E5
prior_percentile	50

References

[1] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[2] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[3] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of PPO in cooperative multi-agent games. In *Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[4] Jing Xu, Fangwei Zhong, and Yizhou Wang. Learning multi-agent coordination for enhancing target coverage in directional sensor networks. *Advances in Neural Information Processing Systems*, 33:10053–10064, 2020.

[5] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, volume 80, pages 3053–3062. PMLR, 10–15 Jul 2018.

[6] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4295–4304. PMLR, 10–15 Jul 2018.

[7] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[8] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, pages 1538–1546. PMLR, 2019.

[9] Ziluo Ding, Tiejun Huang, and Zongqing Lu. Learning individually inferred communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems*, volume 33, pages 22069–22079, 2020.