On Efficient Safe Control based on Supervisory Control Theory and Deep Reinforcement Learning

Masahiro Konishi†, Tomotake Sasaki‡, Kai Cai†

† Osaka City University, Osaka, Japan. ‡ Fujitsu Limited, Kawasaki, Japan.

E-mail: konishi@c.info.eng.osaka-cu.ac.jp, tomotake.sasaki@fujitsu.com, kai.cai@eng.osaka-cu.ac.jp

Abstract: Safe control has recently attracted much attention due to its applications in safety-critical cyber-physical systems. Supervisory control theory (SCT) is a formal control method that provides correct-by-construction safety certificates, but is computationally inefficient when the number of system components is large. On the other hand, deep reinforcement learning (DRL) provides a toolbox of efficient algorithms to compute control decisions even for very large state space, but does not always guarantee safety. In this paper, we propose to synergize SCT and DRL into a new efficient safe control approach. Specifically, we first employ DRL algorithms to efficiently compute sub-optimal solutions which may be unsafe; then we convert the obtained solutions into a standard supervisory control problem with an automaton (plant model) and a set of unsafe states (safety specification); finally we use SCT to synthesize a supervisor with a safety certificate. A case study of multi-robot warehouse logistic automation is conducted to demonstrate the efficiency of this proposed approach.

Keywords: Safe Control, Supervisory Control, Deep Reinforcement Learning

1 Introduction

In recent years, *safe control* has received increasing attention in the field of systems and control engineering, because it has many applications in *safety-critical cyber-physical systems* such as self-driving cars, traffic networks, and multi-robot systems. One important safe control method is based on *control barrier functions* [1], which is effective for avoiding collisions with static or dynamic objects (obstacles or other system components). Another main approach is to first create a *(bi)simulation based discrete abstraction* (aka. symbolic model) for a continuous dynamic system, and then employ *formal methods* from computer science to satisfy safety properties specified using *(linear) temporal logics* [2].

A control native formal method is *supervisory control theory (SCT) for discrete-event system (DES)* [3, 4]. Given a plant modeled by an automaton (equivalently formal language), and a safe control specification in terms of either an automaton or a logical formula, SCT automatically synthesizes a safe (satisfying the specification), deadlock-free (trajectory never halts at an undesired state), and maximally permissive supervisor (allowing as much system behavior as possible). In [5]–[9], SCT has been applied to multi-robot warehouse automation, where item pickup and delivery tasks are automated using a network of mobile robots. For this application, SCT provides safe (collision-free among robots and environment obstacles) and deadlock-free certificates (all robots always complete their tasks without halting in the middle).

However, a shortcoming of SCT when applied to multi-agent systems is *computational scalability*. For an environment displayed in Fig. 1 (a 9×10 grid), standard SCT can compute a valid supervisor for at most 4 robots [5, 8, 9]. In fact, the supervisor synthesis problem is known to be *NP-hard* in that the total state space grows exponentially in the number of component agents [3]. This so-called *state explosion problem* also exists in all core problems addressed by formal methods, which limits their applicability to large-scale systems in practice.

On the other hand, the problem of multi-agent path planning in an environment like Fig. 1 may be formulated as a (discrete) reinforcement learning problem [10], and there exist a number of well-tested *deep reinforcement learning* (DRL) algorithms, e.g. deep Q networks (DQN) and proximal policy optimization (PPO), that may efficiently find *sub-optimal* solutions even if the total state space is large. However, solutions being sub-optimal here mean that there may still exist collision or deadlock states. Consequently such sub-optimal solutions cannot be directly used as safe controllers. On the other hand (as we will show from our experiment results in Section 3 below), running DRL algorithms until we obtain perfect solutions (no collisions or deadlocks) is possible but takes exceedingly long time, which defeats the efficiency expected from such algorithms.

In this work, aiming to take advantage of both SCT (safety guarantee) and DRL (computational efficiency), we propose to combine the two in the following manner to provide efficient safe control solutions. First, we use a DRL algorithm to efficiently compute a sub-optimal solution which may contain unsafe states. Then, we convert the sub-optimal solution into an automaton (plant model) and identify the unsafe states therein (safety specification). This automaton generally has much fewer states than the initial total state number before the DRL computation. Finally, we use SCT to synthesize a provably safe supervisor, which effectively prune the sub-optimal solution by removing all unsafe states. We apply this new safe control

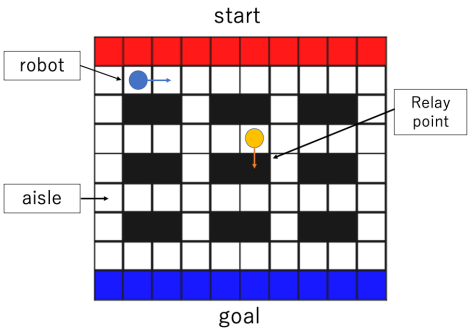

Fig. 1: Warehouse Environment

approach to the same multi-robot warehouse automation in Fig. 1, and report the improved efficiency of computing safe controllers.

The rest of this paper is organized as follows. In Section 2 we present our proposed safe control method by synergizing SCT and DRL. In Section 3 we apply the new method to multi-robot warehouse automation and present derived results. Finally in Section 4 we state our conclusions.

2 Problem Formulation and Proposed Method

Consider a multi-agent DES modeled by automata:

$$\mathbf{G}_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i}), \quad i = 1, \ldots, N.$$

Here Q_i is a finite set of states of agent i, Σ_i is a finite set of events (or actions), $\delta_i : Q_i \times \Sigma_i \to Q_i$ is a (partial) state transition function, $q_{0,i} \in Q_i$ is the initial state, and $Q_{m,i} \subseteq Q$ is the set of marker states (or goal states).

The global DES model is formed by taking the *synchronous product*, denoted by $||$, of all N component automata [3]:

$$\mathbf{G} = \mathbf{G}_1 || \cdots || \mathbf{G}_N = (Q, \Sigma, \delta, q_0, Q_m). \qquad (1)$$

Here the state set $Q = Q_1 \times \cdots \times Q_N$, namely the cartesian product of the N component state sets. Thus the size $|Q| = \prod_{i=1}^{N} |Q_i|$, i.e. it is exponential in N. The event set $\Sigma = \Sigma_1 \cup \cdots \cup \Sigma_N$, the initial state $q_0 = (q_{0,1}, \cdots, q_{0,N})$, and the marker state set $Q_m = Q_{m,1} \times \cdots \times Q_{m,N}$. Finally the state transition function $\delta : Q \times \Sigma \to Q$ is defined based on the component functions δ_i; for the precise definition, the reader is referred to [3].

Now let $Q_{us} \subseteq Q$ be a set of *unsafe states*, which denote those states corresponding to collision or deadlocks. Consider a *(supervisory) control* $U : Q \to 2^{\Sigma}$, which specifies for each state $q \in Q$ a subset of events $\Gamma \in 2^{\Sigma}$ to be *disabled*. Then the *safe control problem* we consider is stated as follows:

Given $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ as in (1) and a (proper) subset $Q_{us} \subseteq Q$ of unsafe states, design a control $U : Q \to 2^{\Sigma}$ such that

(i) Q_{us} is never reached (safety)

(ii) Q_m is always reachable (nonblocking).

To solve this problem, the well-established SCT [3] is effective in principle, but often computationally infeasible since the state size $|Q|$ grows exponentially in the number of agents. To tackle the large state space, we propose to exploit and leverage the efficiency of state-of-the-art DRL algorithms. Specifically, we propose a new safe control approach consisting of the following three steps.

Step 1. Convert the safe control problem into a reinforcement learning problem, and apply DRL algorithms to compute sub-optimal solutions.

First we construct a *Markov decision process* (MDP) $M = (S, A, T, s_0, R)$ as follows. From $\mathbf{G}$ in (1), let the finite state set $S = Q$, the finite action set $A = \Sigma$, the initial state $s_0 = q_0$, and specify the elements of the transition probability matrix T by assigning probabilities to transitions defined by δ. Moreover, design the reward function $R : S \times A \times S \to \mathbb{R}$ based on the set of marker states Q_m (with positive rewards) and the set of unsafe states Q_{us} (with negative rewards).

With the MDP M constructed, we convert the safe control problem into a reinforcement learning problem which finds a *policy* to maximize reward. By the design of R, maximizing reward corresponds tendency of visiting Q_m while avoiding Q_{us}. Let $\pi : S \times A \to [0, 1]$ be a stochastic (or nondeterministic) policy. Then the reinforcement learning problem is to design π such that the following function (state value function) is maximized [10]:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t'=0}^{\infty} \gamma^{t'} r_{t+t'+1} \,\middle|\, s_t = s \right]. \qquad (2)$$

Here $\mathbb{E}_\pi$ denotes the expectation under the transition probability matrix T and a policy π, γ is a discount factor satisfying $0 \leq \gamma < 1$, and $r_{t+t'+1} \in \mathbb{R}$ is the reward obtained at time $t + t' + 1$ according to the reward function R.

Owing to the large size of the state set S, we employ efficient (model-free) DRL algorithms to compute sub-optimal solutions. The basic idea of DRL is to approximate the function $V_\pi(s)$ or a relevant variant (e.g. Q function) by a deep neural network, and use well-tested heuristics to speedup computation. Well-known baseline DRL algorithms include deep Q networks (DQN)[11], advantage actor critic (A2C)[12], and proximal policy optimization (PPO)[13].

Step 2. Convert the computed sub-optimal solutions into a supervisory control problem

Once we obtain a sub-optimal policy $\pi : S \times A \to [0, 1]$, convert π to a deterministic one by greedily selecting the action with the highest probability.

Namely define $\pi_g : S \to A$ according to

$$\pi_g(s) := \text{argmin}_{a \in A} \pi(s, a), \quad s \in S.$$

Then this deterministic policy π_g gives rise to the following trajectory:

$$s_0 \xrightarrow{\pi_g(s_0)} s_1 \longrightarrow \cdots \xrightarrow{\pi_g(s_{k-1})} s_k.$$

Note, in general, that the states in the trajectory need not be distinct; hence loops may exist. If this trajectory happens to contain no unsafe states and the final state $s_k \in Q_m$, then we have found a solution to our safe control problem. In general, however, a trajectory generated from a sub-optimal solution by DRL algorithm does not provide such a certificate. Therefore, for a provably safe solution, we resort to SCT [3]. Based on the above trajectory, we define a supervisory control problem by defining a plant automaton $\mathbf{P} = (X, \Pi, \xi, x_0, X_m)$ and a set of unsafe states $X_{us} \subseteq X$. For $\mathbf{P}$, let

$$\begin{aligned} X &= \{s_0, \ldots, s_k\}, \\ \Pi &= \{\pi_g(s_0), \ldots, \pi_g(s_{k-1})\}, \\ \xi &: X \times \Pi \to X : (s_i, \pi_g(s_i)) \mapsto s_{i+1}, \\ x_0 &= s_0, \\ X_m &= X \cap Q_m. \end{aligned}$$

Finally define the unsafe state set $X_{us} := X \cap Q_{us}$.

Step 3. Apply SCT to solve the supervisory control problem to provide safety certificate.

With the plant automaton $\mathbf{P} = (X, \Pi, \xi, x_0, X_m)$ and unsafe state set X_{us} constructed, standard SCT [3] is applied to synthesize a (supervisory) control $U : X \to 2^{\Pi}$ that guarantees avoiding all states in X_{us} and always reaching X_m.

Now embed $\mathbf{P}$ (as a subautomaton) in the global DES model $\mathbf{G}$. Since the control U keeps trajectory inside X while avoiding $X_{us} \subseteq Q_{us}$, no state in Q_{us} will be visited under U. This provides safety certificate. Moreover, since U keeps X_m reachable and $X_m \subseteq Q_m$, U also ensures nonblocking. Therefore, we have shown that the derived control U solves the safe control problem.

3 Case Study: Multi-Robot Warehouse Automation

In this section, we apply the proposed safe control method to a case study of warehouse logistic automation. Consider the warehouse environment as displayed in Fig. 1 with a grid-type layout. The item pickup and delivery logistics are automated by a team of mobile robots. The top area is where the robots are waiting for tasks, the black-rectangle areas are where items to be picked up are stored, and the bottom area is the item-delivery destination.

A *task* assigned to a robot (at a start location) consists of an item location and a goal location. Once being assigned a task, a robot should travel first to the item location (shown as 'relay point' in Fig. 1), pick up the item, and then deliver the item to the goal location. Treating the positions (denoted by 2D integer coordinates) of the robot as its states, and the movements between neighboring cells as events, an automaton model of the robot may be defined.

When multiple robots (modeled by multiple automata) concurrently serving tasks in the same environment, the safe control problem in this case study requires (i) the robots never collide with each other or with the walls/shelves (safety); (ii) all robots finish their tasks at designated goal locations (nonblocking or deadlock-free).

Now we apply the proposed method in Section 2 step-by-step.

Step 1. Convert the safe control problem into a reinforcement learning problem, and apply DRL algorithms to compute sub-optimal solutions.

Owing to large state size, we employ model-free DRL algorithms. Namely, we do not explicitly write down the MDP model, but only specify the rewards for different scenarios where we want to impose incentives or penalties. For this case study, with trial-and-errors we use the following reward for every robot:

$$r = \begin{cases} 1 & \text{(Reaching goal or relay point)} \\ -1 & \text{(Colliding with another robot)} \\ -0.5 & \text{(Colliding with wall/shelf)} \\ -0.2 & \text{(At start point)} \\ -0.01 \times d & \text{(Moving along aisle)} \end{cases},$$

where d is the Manhattan distance between the robot and the coordinate of the goal or relay point. The first three scenarios are straightforward: encourage the behavior of reaching goal or relay point, and discouraging collisions. The fourth scenario is to discourage robots to stay at the starting points, and the last scenario is to encourage the robots to take as short path towards goal or relay point as possible.

To experiment with DRL algorithms for this case study, the warehouse environment is created in Python with an OpenAI Gym [14] compatible API. Among well-tested DRL algorithms, after trial-and-errors we adopt PPO which is emprically best-suited for this case study. Accordingly we use Ray RLLib [15], and the parameters set in RLLib are shown in Table 1. The parameters that have significant impact on the results are *learning rate* and *clip size*. Learning rate is a parameter that adjusts how fast the learning is progressed. Clip size is a parameter that indicates the tolerance of the difference accepted when updating from an old policy to a new one. The smaller values of both parameters are, the more stable the learning becomes, but the longer the learning time becomes.

The neural network model used in PPO is a feed-forward neural network (multi-layer perceptron), and

Table 1: Hyperparameters used in PPO

parameter	value
GAE parameter(λ)	0.9
SGD batch size	4096
SGD minibatch size	64
SGD epoch	15
discount rate(γ)	0.99
learning rate(α)	5×10^{-5}
entropy coefficient(β)	0.01
clip size(ϵ)	0.1

Table 2: Neural network model used in PPO

input layer	current position, target position, current positions other robots
hidden layer	2 layers 256 parameters
output layer	probability of movement in 5 directions (up, right, down, left, stop)
activation func	tanh

the settings are shown in Table 2.

The experiment results are given in Tables 3 and 4. Each table presents 17 results for 4 to 20 robots, with computation time and training steps. "NG" means the algorithm did not finish within 2000 trials of learning. A trial means collecting data for the number of steps of the *SGD batch size* and training of the model using it.

Table 3 shows the case where the algorithm is terminated until a solution with no collision is found. Such solutions are already valid solutions for the safe control problem. However, we point out that it is very time consuming to compute these safe solutions by PPO, and there are 4 "NG" cases in this experiment. On the other hand, Table 4 shows the case where the algorithm is terminated when a solution with no more than 5 collision states is computed. This way notably reduces computation time and "NG" cases, but the solutions are not yet safe.

Considering the efficiency obtained by terminating DRL algorithms when a reasonable sub-optimal solution is obtained, we adopt the results in Table 4 and resort to SCT to further provide safety certificate.

Step 2. Convert the computed sub-optimal solutions into a supervisory control problem

To determine if a sub-optimal solution is 'reasonable' and thereby set a criterion to terminate the DRL algorithms is case-dependent. For this case study, we consider using the criterion of the number of collision occurrences contained in sub-optimal solutions, and conduct tests of algorithm execution times as displayed in Fig. 2. We use bar graphs to compare the execution times for collision numbers 0, 5, 10, 15, 20, as well as for robot numbers from 4 to 18.

Observe that the times needed to achieve 0 collision

Table 3: PPO learning results (0 collisions)

No. of robots	Learning time
4 robot	5.4[min]
5 robot	8.0[min]
6 robot	21.1[min]
7 robot	59.1[min]
8 robot	NG
9 robot	NG
10 robot	46.5[min]
11 robot	22.5[min]
12 robot	149.7[min]
13 robot	118.7[min]
14 robot	86.7[min]
15 robot	NG
16 robot	145.3[min]
17 robot	136.4[min]
18 robot	79.9[min]
19 robot	NG
20 robot	145.8[min]

Table 4: PPO learning results (≤ 5 collisions)

No. of robots	Learning time
4 robot	4.6[min]
5 robot	5.5[min]
6 robot	8.2[min]
7 robot	9.8[min]
8 robot	11.6[min]
9 robot	13.4[min]
10 robot	16.0[min]
11 robot	19.8[min]
12 robot	21.2[min]
13 robot	27.0[min]
14 robot	28.8[min]
15 robot	NG
16 robot	118.8[min]
17 robot	52.0[min]
18 robot	40.7[min]
19 robot	54.0[min]
20 robot	54.1[min]

are notably longer (from 6 robots); this shows that to use only DRL algorithms to achieve perfectly safe solutions is inefficient. Further, the rest cases of collision numbers 5, 10, 15, 20 do not exhibit significant differences in execution times, though the tendency is the more collisions allowed the shorter times needed (which matches intuition).

Based on the observation from Fig. 2, we choose to set the criterion of (maximum) 5 collisions to terminate the algorithm execution. The results, as already stated above, are displayed in Table 4 for up to 20 robots.

On each termination, a sub-optimal stochastic policy π is obtained, which is generally unsafe. Namely using π, robots may collide with one another or with wall/shelf up to 5 times. To provide safety guarantees,

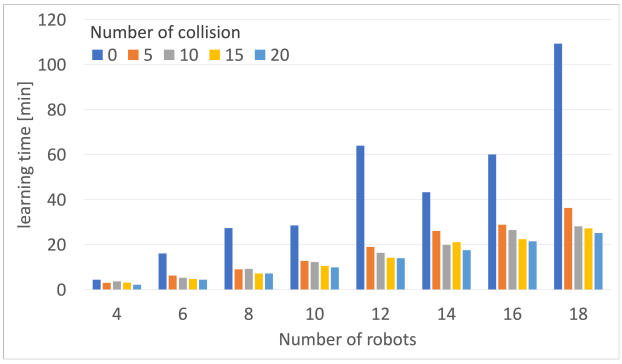

Fig. 2: Comparisons of PPO execution times for different numbers of collisions and robots

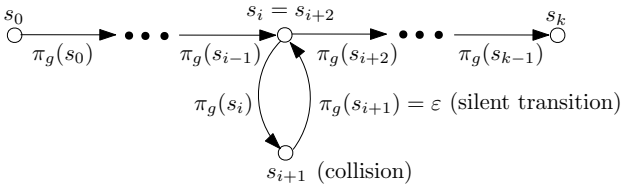

Fig. 3: Plant automaton constructed from policy

we resort to SCT. To that end, we define a supervisory control problem based on π.

Following the procedure outlined in Section 2, we first convert π to a deterministic policy π_g by a greedy heuristic: at each state selecting the action with the highest probability. Under this π_g, we construct a trajectory

$$s_0 \xrightarrow{\pi_g(s_0)} s_1 \longrightarrow \cdots \xrightarrow{\pi_g(s_{k-1})} s_k.$$

For this case study, the initial state s_0 is where all robots are at their starting locations, while the final state s_k is where all robots are at their goal locations. Some states in the middle may represent collisions, where at least two robots have the same coordinates, or a robot has the same coordinate as wall/shelf. In our simulation environment, if a collision occurs due to a robot's action, the robot does not actually execute that action and its coordinate does not change. To reflect this in the trajectory, we add a dummy collision state and a dummy silent transition to form a 'virtual selfloop'.

An illustration is displayed in Fig. 3. Consider that a collision occurs at state s_i when action $\pi_g(s_i)$ is taken. We add a collision state s_{i+1} (even though in the simulation the robot does not actually move), and then add a silent transition denoted by ε from s_{i+1} back to s_i but now relabeled to s_{i+2} (as a successor state of s_{i+1}). Thus we create a dummy loop with an explicit collision state.

With the trajectory constructed above, we maps it to the plant automaton $\mathbf{P}$, and define the set of unsafe states X_{us} to include all the dummy collision states (like s_{i+1} in Fig. 3).

Step 3. Apply SCT to solve the supervisory control problem to provide safety certificate.

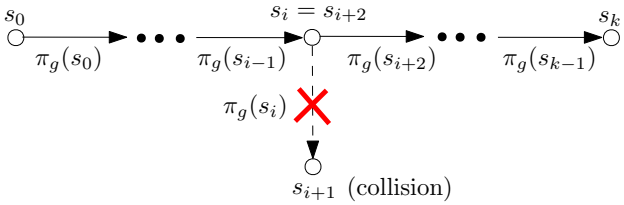

Fig. 4: Supervisory control that disables actions to unsafe states

Finally we apply the standard SCT to provide safety certificate. With the plant automaton $\mathbf{P}$ and define the set of unsafe states X_{us} as inputs, SCT disables all actions that potentially lead to unsafe states. For the illustration given in Fig. 3, the corresponding supervisory control is to disable action $\pi_g(s_i)$ which leads to collision (see Fig. 4). In this way, all unsafe states will never be visited.

In addition, since the final state s_k is the desired marker state (as it represents all robots at their goal locations), the supervisory control also ensures nonblocking. Therefore, we have obtained valid solutions to the safe control problem for this case study.

Regarding efficiency, as compared to previous work on the same case study [5, 8, 9] where at most 4 robots are computationally feasible, we have successfully obtained solutions for 20 robots in reasonable time thanks to DRL.

4 Conclusion

In this paper, we have proposed a new safe control approach by combining the correct-by-construction SCT and the computationally efficient DRL algorithms. Safety certificate of this approach is established, and a case study on multi-robot warehouse automation is conducted to show notable improvement of efficiency as compared to previous work.

In our ongoing work, we continue to experiment with larger numbers of robots than reported in this paper. Another intriguing direction is to directly explore the stochastic policies obtained by DRL (in step 2 of our approach). This is because DRL algorithms only generate sub-optimal solutions, non-greedy selections of actions may possibly lead to better solutions. They may lead also to worse solutions, with more collisions and deadlocks, but these can all be correctly removed by SCT at the final step. Lastly we will aim to apply this approach to other safety-critical cyber-physical systems such as connected autonomous vehicles and traffic networks.

References

[1] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath and P. Tabuada: Control Barrier Functions: Theory and Applications;

Proc. 18th European Control Conference (ECC), pp. 3420-3431 (2019)

[2] C. Belta, B. Yordanov, and E. A. Gol: Formal methods for discrete-time dynamical systems, Springer (2017)

[3] W. M. Wonham and K. Cai: *Supervisory Control of Discrete-Event Systems*, Springer (2019)

[4] K. Cai and W. M. Wonham: *Encyclopedia of Systems and Control, 2nd ed.*, pp.1-9, Springer (2020)

[5] Y. Tatsumoto, M. Shiraishi, K. Cai: Application of supervisory control theory with warehouse automation case study; *Transactions of the Institute of Systems, Control and Information Engineers (ISCIE)*, Special Issue on Event-Driven Approach to System Design – Application and Development, vol. 62, no. 6, pp. 203-208 (2018)

[6] Y. Tatsumoto, M. Shiraishi, K. Cai, and Z. Lin: Application of online supervisory control of discrete-event systems to multi-robot warehouse automation; *Control Engineering Practice*, vol. 81, pp. 97-104 (2018)

[7] K. Cai: Warehouse automation by logistic robotic networks – a cyber-physical control approach; *Frontiers of Information Technology & Electronic Engineering*, vol. 21, pp. 693-704 (2020)

[8] M. Kasahara and K. Cai: Online multi-agent supervisory control for warehouse automation: heterogeneous payloads; *Proc. 29th Mediterranean Conference on Control and Automation*, (2021)

[9] M. Kasahara and K. Cai: Online multi-agent supervisory control for warehouse automation: prioritized tasks, *Proc. 17th International Conference on Automation Science and Engineering*, (2021)

[10] R. S. Sutton and A. G. Barto: *Reinforcement Learning, 2nd ed.*, The MIT Press (2018)

[11] V. Mnih, K. Kavukcuoglu, D. Silver, et al: Human-level control through deep reinforcement learning; *Nature*, vol. 518, pp. 529–533 (2015)

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu: Asynchronous methods for deep reinforcement learning; *Proc. 33rd International Conference on Machine Learning (ICML)*, pp. 1928-1937 (2016)

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov: Proximal policy optimization algorithms; *arXiv preprint*, arXiv:1707.06347 (2017)

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba: OpenAI Gym; *arXiv preprint*, arXiv:1606.01540 (2016)

[15] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica: Rllib: Abstractions for distributed reinforcement learning, *arXiv preprint*, arXiv:1712.09381 (2018)