Simulation-based Reinforcement Learning Approach towards Construction Machine Automation

K. Matsumoto[a], A. Yamaguchi[a], T. Oka[a], M. Yasumoto[a], S. Hara[b], M. Iida[b] and M. Teichmann[c]

[a]Araya inc., Japan
[b]Information Services International-Dentsu, Ltd., Japan
[c]CM Labs Simulations, Canada
E-mail: matsumoto_k@araya.org, atsusysw@araya.org, t.oka@araya.org, yasumoto@araya.org, shara@isid.co.jp, iida.michitaka@isid.co.jp, marek@cm-labs.com

Abstract –

Research towards automation of heavy construction machines for efficient and safe construction processes that are robust to various environments and disturbances has been conducted for many years. In this paper, we show two contributions towards this objective. Firstly, we explore the use of reinforcement learning to automate construction machines. Secondly, we evaluate the effectiveness of three methods to reduce learning time of reinforcement learning: designing reward function, pre-training with BC, and changing frame-skip rate. The reinforcement learning approach is expected to gain robustness against disturbances through learning. We run experiments on two different realistic tasks. The first task is to reduce sway of a load suspended from a mobile boom crane, behaving as a single pendulum. The second task is to load an excavator bucket with soil with a hydraulic excavator. We demonstrate the effectiveness of algorithms using the reinforcement learning approach on the commercial simulator, Vortex Studio developed by CM Labs Simulations.

Keywords –

Reinforcement Learning; Imitation Learning; Automation; Autonomous; Simulation

Figure 1. A construction machine that performs tasks in a simulator. Anti-sway crane (left). Load the soil with excavator (right).

1 Introduction

For many years, automation of heavy construction machines has been studied to improve safety, work productivity, and construction quality. However, it has been pointed out that even recent studies lacks consideration of model uncertainties [1], parameter variations, and disturbances [2].

Reinforcement learning combined with neural networks has been shown to be an effective framework for solving complex problems. In reinforcement learning, an agent interacts with the environment through trial and error and learns the optimal control method based on signals from the reward function. Various problems in robotics are naturally expressed as reinforcement learning problems [3]. Using reinforcement learning, robots can autonomously find optimal movements and gain robust movement by gaining experience in dealing with various disturbances.

Reinforcement learning has had a great success in the game field [4], but is more difficult to apply in robotics [5]. Since reinforcement learning improves the controlling policy in an incremental manner, an initial movement is almost like a random movement, which can be sometime dangerous in the real-world tasks. Furthermore, agents need to learn in a wide variety of environments in order to prevent over-fitting to the specific environment and improve the robustness against the environment changes. However, this type of learning requires a lot of time and cost to prepare such environment in the real world.

A simple way to solve the above problem is to use a simulator [6][7]. One can easily create a wide variety of environments and simulation is completely safe i.e. the virtual construction machine will never be damaged even if it accidentally crashes in the learning phase. We chose to use Vortex Studio as a simulator which is specialized in construction machine models and can perform high-precision physical simulations quickly.

The biggest problem we face when using

reinforcement learning is sample efficiency. Generally, more than millions trial and error iterations are required to achieve human-level performance. Although simulators can generally run tasks much faster than real time, it still needs a couple of weeks to complete learning which hinders the efficiency of experiments.

Various reinforcement learning algorithms have been proposed to improve sampling efficiency such as distributed learning [8][9][10], transforming or adding reward functions such as reward shaping[11] and intrinsically motivated function[12], mixing imitation learning and reinforcement learning [13] and changing frame-skip, the number of frames an action is repeated before a new action is selected [14].

In this paper, we define two tasks for the construction machines and apply reinforcement learning for these tasks. We also introduce and evaluate some of the techniques for improving sample efficiency. In order to implement these techniques, we made some modification to the Vortex Studio simulator, which was originally not intended for reinforcement learning use and convert it to a "Reinforcement learning ready" simulator. In this paper, our contributions are:

- We introduce reinforcement learning for the following two tasks as shown in Figure 1: The first task is to reduce sway of a load suspended from a mobile boom crane, behaving as a single pendulum. The second task is to load an excavator bucket with soil with a hydraulic excavator. We show that reinforcement learning is one of the effective methods that can perform as well as humans in the automation of construction machine.
- We evaluate the effectiveness of the three methods to improve sample efficiency for reinforcement learning: designing reward function, pre-training, and changing frame-skip rate.
- We provide examples of practical methods on how to parallelize learning when applying reinforcement learning algorithms to domain-specific simulators that are not for reinforcement learning.

The paper has been organized as follows. Section 2 describes the fundamental concept of reinforcement learning. Section 3 describes the methods applied to improve sample efficiency. In Section 4, we describe the detail setup of the experiments. In Sections 5, we mention some considerations and ideas for performing high-speed learning with a simulator that is not built for reinforcement learning. In Section 6, we describe the experimental results performed on crane or excavator tasks. Section 7 presents the summary and conclusions of this paper and discusses future work.

2 Preliminary

2.1 Reinforcement learning

In this section, we provide a definition used in reinforcement learning [15] focusing on the case the environment E as a finite-state Markov decision processes (MDP). An agent maximizes cumulative rewards by selecting an optimal action from all actions $A = \{a^1, \dots, a^k\}$ an agent can select, in discretized timesteps in some state $s \in S$ of an environment E,where k is the number of actions the agent has. Rewards are the criteria by which an agent learns good or bad behavior. At every timestep t, an agent takes an action $a_t \in A$ when in state s_t. After that, an environment transitions to next state s_{t+1} by transition function $s_{t+1} = T(s_t, a_t)$ and the agent gets a reward $r_t = R(s_t, s_{t+1})$ where T gives the state transition and R gives the reward. To evaluate how much the action performed in a certain time step has contributed to the total cumulative reward, we consider the estimated reward

$$G_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-t-1} r_{\mathrm{T}} = r_{t+1} + \gamma G_t \quad (1)$$

, where $\gamma \in (0, 1]$ is the discount factor, and T is the time when the episode ends. An agent learns a policy $\pi(s_t)$ that maximize cumulative expected rewards until the end of episode. In reinforcement learning, an optimal policy $\pi^*$is obtained by interacting with the environment until the action-value function

$$Q_\pi(s_t, a_t) = \sum_{s_{t+1}} T(s_t, a_t)\,(R(s_t, s_{t+1}) + \gamma G_{t+1}) \quad (2)$$

converges.

Generally, it is difficult to model real transition functions T. The use of model-free algorithms to approximate action-value function

$$Q_\pi(s_t, a_t) \approx R(s_t, s_{t+1}) + \gamma \max_{a_{t+1}} E[Q_\pi(s_{t+1}, a_{t+1})] \quad (3)$$

is now mainstream [16].

In this paper, we use the PPO algorithm[9], which is well known as the stable model-free algorithm and support distributed learning. Distributed learning is the way to learn effectively by replicating the agent and the environment, having the agents act in each environment and gaining a lot of experience. With distributed learning we can speed up the learning. The algorithms in [9] is composed of Actor-Critic network architecture. Actor-Critic is common methodology for Critic to encourage Actor to update the policy, and there are various implementations such as making the Actor and the Critic in different networks or sharing a part of the network.

2.2 Imitation learning

Imitation learning is another approach to create an agent. It learns an optimal policy from demonstrations of expert human by imitating. A simple approach to imitation learning is behavior cloning (BC) [17], which learns a policy from demonstrations of successful behavior through supervised learning.

Pure imitation learning methods cannot exceed the capabilities of the demonstrations. In addition, a huge number of demonstrations is required when applying it to actual tasks because it is not possible to respond to scenes that are not in the demonstration.

Imitation learning can also be used to boost the learning efficiency of reinforcement learning by using the learned policy of BC as an initial policy of reinforcement learning.

3 Method

In this section, we describe three methods to improve the sample efficiency of reinforcement learning in detail. All of these methods are ways to simplify the state space that the agent explores. Figure 2 shows how the state space is simplified by each method. Even if the state space is continuous, the reachable state space is discretized by time, so we represent the state space by grid world.

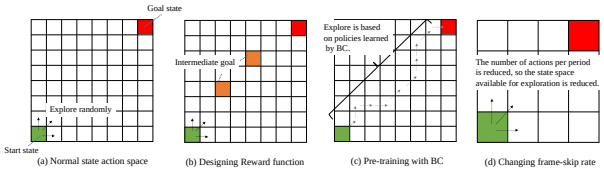

Figure 2. State space is how simplified by each method.

3.1 Designing Reward function

The simplest form of reward is a binary reward, that is, an agent gets a positive reward when an agent achieves the desired state at the end of an episode, otherwise a negative reward. This form of reward leads an agent to get desired policy. The problem with this reward is that the agent is essentially exploring by random actions at first, and therefore cannot learn good behavior at all until the agent is able to earn the reward. This is especially problematic when the state space or action space is large, as it takes longer to explore and the possibility that the agent gets a reward is lower.

To avoid this problem, there is a way to give a reward at each action or state that will lead to the final goal. This type of reward is called as a dense reward. While this can improve learning speed, you need to design dense reward function carefully otherwise the learned behavior may be different from what we actually want.

In this paper, we presented two results for each task: sparse reward and dense reward.

3.2 Pre-training with BC

In general, RL uses random behavior as an initial policy, which is one of the reasons for low sample efficiency of the RL algorithms. As mentioned in 2.2, exploration can be performed efficiently by properly initializing the policy using imitation learning[13].

In this paper, we combine BC [17] with the PPO algorithms[9]. Unlike [13], we use the model trained in BC as the initial value of the model for reinforcement learning.

3.3 Changing frame-skip rate

Another way to speed up learning is to change the frame-skip rate[14]. Frame skipping is the technique to repeat the same action for several frames. Increasing the frame-skip rate has the effect of coarsening the granularity of action. The coarsening of the granularity of the action reduces the state that can be reached within a certain time step. As a result, the combination of state and action spaces is reduced, which simplifies the task.

Increasing the frame-skip rate will also be very important when applying reinforcement learning to real construction machines because it allows the granularity of a single action choice to be changed and the overall action to be smoother. Increasing the frame-skip rate reduce not only productivity but also fuel efficiency and machine fouling and wear. However, simply increasing the frame-skipping rate does not allow for fine-grained action, and thus does not explore the state space that makes the task successful and may prevent agents from achieving their goals. Trade-offs between fineness of action and learning time should be considered, depending on the task and state in order to decide the appropriate number of frame skips. In this paper we tried two patterns about frame-skip rate, no frame skip and 19 frame skips.

4 Experiments

In our experiments we aim at answering the following questions:

1. Does reinforcement learning work to solve realistic construction machine tasks?
2. Which is the most effective method to reduce learning time?

We evaluate our approach on two different realistic construction tasks, anti-sway crane and loading the soil with an excavator. We used the stable baselines framework [18] as the basis for our implementation in

this experiment. The network architecture of our system is depicted in Figure 3. Both policy network and value network have 3 fully connected layers, and each network are independent.

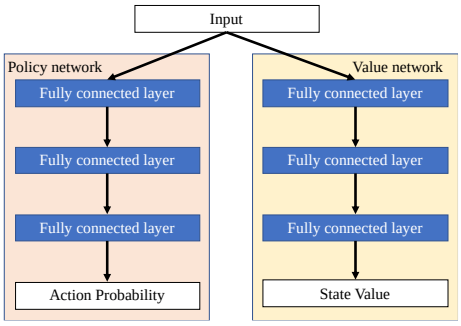

Figure 3. The network architecture of our system.

The Learning is performed on a single machine with an Intel Core i7-8700 and GeForce GTX 1070.

The hyperparameters we used, are same as the default values defined in [18].

4.1 Simulators

CM Labs' Vortex Studio provides a virtual environment for real-time simulation of complex multibody systems. In Vortex, each rigid body is formulated using six degrees of freedom, and these bodies can be connected with several constraint types with varying numbers of linear and/or angular degrees of freedom. The toolkit supports equality as well as inequality constraints, e.g., contacts, which can be holonomic or nonholonomic. Vortex employs the method of Lagrange multipliers together with a direct solver and a semi-implicit integration scheme for stable and fast multi-body simulation. With its optimized core for fast and real-time simulation and its advanced graphical capabilities, Vortex has various applications including operator training, mission planning and design. Furthermore, there are some modules provided in Vortex for particular applications. We used the Cable system module and the Earthwork system module for this study which differentiates Vortex Studio from other real-time simulations [19].

4.1.1 Vortex Cable system simulation

Vortex Studio's Cable Systems provides a realistic simulation of heavy-equipment cables. These extended capabilities ensure real-time behavior by allowing cables to adjust to bends while distributing mass and forces correctly. It can predict cable behavior with minimum number of segments to simulate, which the number of segments changes depending on the curvature of the cable using Adaptive Cable method. This allows Vortex to consider all bending/axial/torsion stiffness and damping to simulate the real cable behavior without falling behind the time step as required by real-time simulation.

4.1.2 Vortex Earthworks system simulation

The Vortex Earthwork Systems module is tailored to the needs of high-fidelity, real-time earth-moving simulations, and employs physically based soil deformation algorithms. Terrain deformations, caused by earth-moving tools such as buckets, and the corresponding terrain reaction forces, are captured in real-time and full two-way force coupling with other simulation entities is modelled. The interactions between machine and soil are fully simulated, allowing soil to be cut, compressed and spilled, all inside an interactive environment by using a hybrid particle-based and mesh-based soil simulation method [20].

4.2 Task 1: Crane

In this task, the goal is to control the relative position shift and speed of the suspended load (the rotation of the load is not taken into account in this experiment) as shown in Figure 4, so this crane is regarded as a single pendulum.

Figure 4. At first, the load suspended by the crane is intentionally shaken a certain amount. After a certain time, the operational agent reduces sway of a load suspended from a mobile boom crane.

4.2.1 Experimental setup

We model the components of the crane task as shown in Figure 5. In this figure, x_{diff} and y_{diff} are distance between boom tip coordinate to load center coordinate, $|v|$ is absolute value of speed of the load, $|a|$ is acceleration of the load. We stack these 4-dimensional data for 3 timesteps and use them as input for reinforcement learning model.

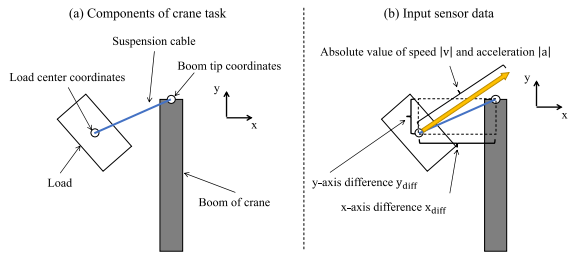

Figure 5. (a) Components of crane task (b) Input

sensor data.

We define the agent actions as elevation and boom as shown in the Figure 6. While PPO [9] can handle both continuous and discrete action spaces, we use the discretized elevation action out of 3 and the discretized boom action out of 5. The reason to use the discretized action is to speed up the learning.

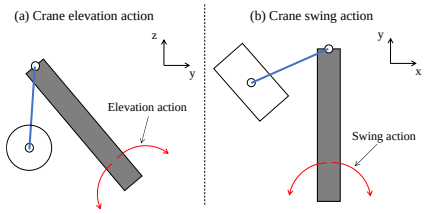

Figure 6. (a) Crane elevation action (b) Crane swing action.

We set one episode to consist of a maximum of 600 steps to use the framework of reinforcement learning. We define the episode success condition as maintaining $|x_{diff}|$ and $|y_{diff}|$ below 1.5 m and speed $|v|$ below 1.5 m/s for 60 steps. We define the episode failure conditions as follows:

- The acceleration of the suspended load exceeds a certain threshold
- The suspended load collides with another object (ground or crane body)
- 600 timesteps elapse without reaching episode success condition

4.2.2 Method detail

Details of each method to be compared in the crane task are presented below.

Base condition: We define the base method as follows:

- $+1.0$ reward when episode reaches success condition, and -1.0 reward when episode reaches failure condition (sparse reward)
- Learning starts from scratch (random actions)
- No frame-skip

Designing reward function: We design the dense reward function as follows:

- For every step, reward is given according to the following function

$$r_t = \frac{1.0}{\max\left(1, \sqrt{x_{diff}^2 + y_{diff}^2}\right)} \quad (3)$$

- $+(960 - current\ step)$ reward when episode success condition
- -1.0 reward when episode failure condition

Pre-training with BC: We do not attempt pre-training with BC because the algorithm in [18] doesn't support BC with multi-discrete actions.

Changing frame-skip rate: We change frame-skip rate to 19.

4.3 Task 2: Excavator

In this task, the goal is to load more soil in a single excavation with a hydraulic excavator as shown in Figure 7.

Figure 7.The agent operates only the bucket part. The agent scoops as much soil as possible in accordance with the movement of the excavator that follows a certain trajectory.

4.3.1 Experimental setup

We model the components of the excavator task as shown in Figure 8. In this figure, y_{diff} and z_{diff} are distance between the excavator body to the bucket, v_y and v_z are value of speed of the bucket, n is an actuator force of the bucket cylinder, θ is a bucket angle, w is the Weight of the soil in the bucket. We stack this 7-dimensional data for 3 timesteps and use them as input for reinforcement learning model.

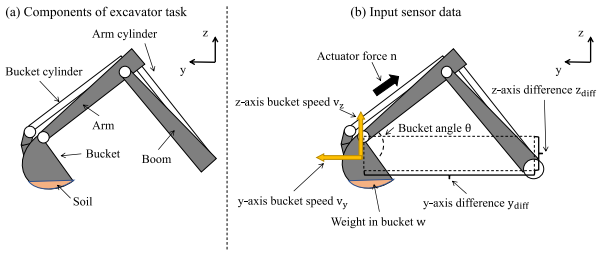

Figure 8. (a) Components of excavator task (b) Input sensor data.

We define the agent bucket action as shown in Figure 9. We discretize the bucket action into 11 values.

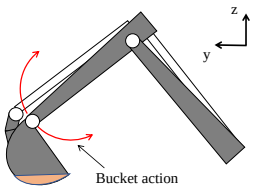

Figure 9. Excavator action.

We set one episode to consist of up to 460 steps to use the framework of reinforcement learning. We define the episode success condition to be $w \geq 1800$ after 460 steps have elapsed. We define the episode failure condition to be the lack of satisfaction of the success condition after 460 steps have elapsed.

4.3.2 Method detail

Details of each method to be compared in the excavator task are presented below.
Base condition: We define base method as follows:

- $+w/1800$ reward when episode reaches success condition, and -1.0 reward when episode reaches failure condition (sparse reward)
- Learning starts from scratch (random actions)
- No frame-skip

Designing reward function: We design the dense reward function as follows:

- Every step, $+1.0$ reward is given when $w \geq 1800$
- $+w/1800$ reward when episode success condition is satisfied
- -1.0 reward when episode failure condition is satisfied

Pre-training with BC: To perform pre-training with BC, we collected 120 expert demonstration data on the simulator.
Changing frame-skip rate: We change frame-skip rate to 19.

5 Implementation

In order to implement distributed learning with Vortex simulator, we consider three options, as shown in the Figure 10.

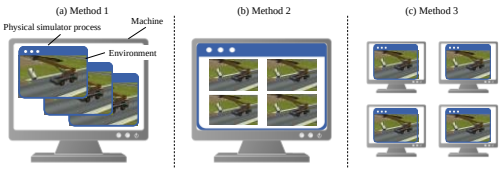

Figure 10. (a) Launching multiple processes on a single machine. (b) Creating multiple environments on a single physical simulator process. (c) Provide multiple machines on which the physical simulator process can run.

There are cases that GUI operation is the only way to advance the time of the simulator. In this case, Method 1 take much time due to the interaction with the GUI, but this can be resolved by using the Vortex python language scripting API to run the simulation. However, each process of the multiple simulators needs memory and CPU which requires a high-performance computer.

Method 2 needs only one process, so it is efficient in terms of memory and CPU. However, if the simulator is unable to reset each separate environment in the process to its initial state individually (synchronous parallel environment) as in Vortex, one needs to develop a new function module to summarize the environment state in the process because it is a requirement of the OpenAI gym interface, which is de facto interface and is assumed by many reinforcement learning frameworks. Moreover, as shown in the Figure 10, in the case of synchronous parallel environments, unlike asynchronous parallel environments, all environments need to reach the end of the episode. This may lead to a deterioration of the sample efficiency.

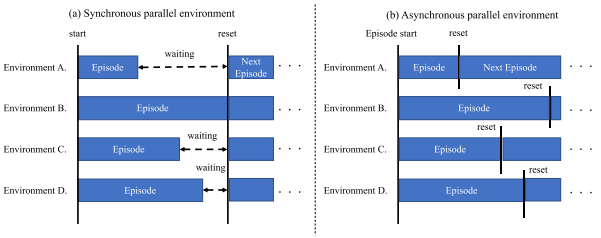

Figure 11. Progress of episodes in synchronous and asynchronous parallel environments

Method 3 is very simple but requires multiple computers.

Method 1 or 2 and Method 3 can be used together, and faster learning is expected. In our experiment, we adopt Method 2, which requires the lowest machine cost and is expected to achieve better performance than Method 1.

Note that Vortex simulator plan to develop an asynchronous parallel environment, so it will resolve the problem of sample efficiency in the future.

6 Results

In this section, we evaluate how each method contributes to each task success rate and learning speed. The results are showed in the Figure 12. The success rate is measured through 100 trials. We also show the task

success rate using the best performance model after learning. Table 1 shows the best success rate for each method on the simulator.

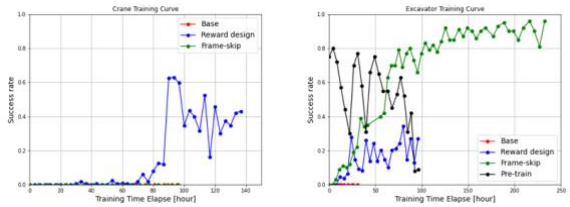

Figure 12. Learning efficiency for each method (left) crane (right) excavator

Table 1. Task success rate

Task	Success rate by methods, %			
	Base	Designing Reward function	Pre-training with BC	Changing Frame-skip
Crane	0	63	-	0
Excavator	0	34	80	96

6.1 Base condition

In both tasks, the base condition had no success, while other methods are starting to succeed at the same point in learning time. This indicates that the base condition needs more learning time and the other methods make learning more efficient.

6.2 Designing the reward function

In the crane task (see Figure 12 left), this method was the only one that worked. It reached about 60% success rate, and then the success rate dropped off. And also, in the excavator task (see Figure 12 right), this method had about 30% success rate. This indicates that the designed reward function is not directly related to the success of the task.

6.3 Pre-training with BC

Pre-training with BC took almost 1 hour wall-clock time to converge. In the early stages of learning, BC gave a high performance, but as the learning progressed, the performance gradually decreased. We believe that it is because the form of reward is different between BC and PPO.

6.4 Changing Frame-skip

The frame skip method achieved 96% success rate in the excavator task, whereas the crane task did not succeed at all. This indicates that the exploration for a good frame skip rate for each task or scene will yield good results.

7 Conclusion & Future work

In this paper, we demonstrated through simulations that reinforcement learning works effectively for two tasks: reducing sway of a load suspended from a mobile boom crane and loading an excavator bucket with soil with a hydraulic excavator. In addition, we confirmed the effectiveness of three methods to reduce learning time of reinforcement learning: designing the reward function, pre-training, and changing frame-skip rate. Finally, we provided examples of practical methods on how to parallelize learning when applying reinforcement learning algorithms to domain-specific simulators that were not originally designed for reinforcement learning.

In future work, we intend to compare our reinforcement learning algorithms with an existing control method on simulation. We also consider that the current learning on simulation is not tolerant to transfer to the real world. In recent years, methods to close the reality gap has been widely studied [21][22][23]. It is necessary to apply these methods to smoothly transfer learning result of simulation to real world operation.

References

[1] S. Dadhich, U. Bodin, and U. Andersson, Key challenges in automation of earth-moving machines. Automation in Construction, 68:212-222, 2016.

[2] L. Ramli, Z.Mohamed, A. M. Abdullahi, H. I. Jaafar, and I. M. Lazio, Control strategies for crane systems: A comprehensive review. Mechanical Syst-ems and Signal Processing, 95:1-23, 2017.

[3] J. Kober, J. A. Bagnell, J. Peters, Reinforcement Learning in Robotics: A Survey, The International Journal of Robotics Research, 32.11: 1238-1274, 2013.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, L. Antonoglou, D. Wierstra, and M. Riedmiller, Playing Atari with Deep Reinforcement Learning, arX-iv preprint arXiv: 1312.5602, 2013.

[5] L. Tai, J. Zhang, M. Liu, J. Boedecker, and W. Burgard, A Survey of Deep Network Solutions for Learning Control in Robotics: From Reinforcement to Imitation, arXiv preprint arXiv: 1612.07139v4, 2018.

[6] J. García, and F. Fernández, A Comprehensive Survey on Safe Reinforcement Learning, Journal of Machine Learning Research 16.1: 1437-1480, 2015.

[7] D. Amodei, C. Olah, J. Schulman, J. Steinhardt, P. Christiano, D. Mané, Concrete Problems in AI

Safety. arXiv preprint arXiv:1606.06565v2, 2016.
[8] V. Mnih, A. P. Badia, M. Mirza, A.Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcauoglu, Asynchronous Methods for Deep Reinforcement Learning. International conference on machine learning:1928-1937, 2016.
[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347, 2017.
[10] S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan, and R. Munos. Recurrent experience replay in distributed reinforcement learning. In International Conference on Learning Representations, Louisiana, United States, 2019.
[11] A. Y. Ng, D. Harada, and S. Russell, Policy invariance under reward transformations: Theory and application to reward shaping. *ICML*. Vol. 99:278-287, 1999.
[12] S. P. Singh, A. G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In Advances in neural information processing systems: pages 1281–1288, 2005.
[13] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, L. Osband, J. Agapiou, J. Z. Leibo, A. Gruslys, Deep Q-learning from Demonstrations. arXiv preprint arXiv:1704.03732, 2018.
[14] A. Braylan, M. Hollenbeck, E. Meyerson and R. MiikkulainenFrame Skip Is a Powerful Parameter for Learning to Play Atari. Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
[15] R. S. Sutton, and G. B. Andrew, Introduction to reinforcement learning. Vol. 135. Cambridge: MIT press, 1998.
[16] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems:1057-1063, 2000.
[17] D. A. Pomerleau, Efficient training of artificial neural networks for autonomous navigation. Neural Computation, 3(1):88–97, 1991.
[18] H. Asheley, R. Antonin, E. Maximilian, G. Adam, K. Anssi, T. Rene, D. Prafulla, H. Christopher, K. Oleg, N. Alex, P. Matthias, R. Alec, S. John, S. Szymon, and W. Yuhuai, Stable Baselines. Online: https://github.com/hill-a/stable-baselines, Accessed: 04/03/2020.
[19] Daniel Holz, Ali Azimi, and Marek Teichmann,Virtual Reality Simulation of Vehicles and Tools Interacting with Deformable Terrain, CM-Labs Simulations Inc.,2012
[20] CM-Labs Simulations Inc., Vortex Studio 2018a Product Capability Specifications, Online: https://cmlabsnew.kamacom.com/vortexstudiodocumentation/Vortex_User_Documentation/Content/Resources/Vortex_Studio_2018a_Product_Specifications.pdf, 06/05/2020.
[21] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. arXiv preprint arXiv:1703.06907, 2017.
[22] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. arXiv preprint arXiv:1804.10332, 2018.
[23] I. Clavera, D. Held and P. Abbeel, Policy Transfer via Modularity. in IROS. IEEE, 2017.