FPGA-Accelerated Sim-to-Real Control Policy Learning for Robotic Arms

Ce Guo and Wayne Luk
Imperial College London, UK
{c.guo, w.luk}@ic.ac.uk

***Abstract*—Sim-to-real robot learning has been used in various applications, but its implementation in software may not provide the best performance. This tutorial describes how hardware acceleration based on Field-Programmable Gate Array (FPGA) technology for deep reinforcement learning can improve sim-to-real robot control policy learning. A novel architecture for the Deep Deterministic Policy Gradient (DDPG) algorithm is developed for a full-stack sim-to-real development platform to learn control policies for robotic arms. The capability of our development platform is illustrated by transferring learned policies encoded as fixed-point numbers from our implementation to a miniature robotic arm.**

***Index Terms*—FPGA, acceleration, robotic arms, sim-to-real, robot learning, reinforcement learning**

I. Introduction

The traditional approach to robot development involves manually programming predefined action policies, suitable for straightforward environments. However, robots today often face complex and unpredictable settings, where manual programming becomes labor-intensive and error-prone. Enhancing programming quality and efficiency can be achieved by training robots using reward-based reinforcement learning [1]. Yet, applying reinforcement learning directly to physical robots presents drawbacks: there are risks of harm to humans, the environment, and the robots themselves, especially during the early stages of policy development. Furthermore, training with physical robots is time-consuming, as reinforcement learning algorithms require extensive interaction history to develop effective policies.

A viable alternative is sim-to-real robot learning [2], [3], where the reinforcement learning algorithm initially trains using virtual robots in simulated environments before applying the learned policy to physical robots. Simulations effectively reduce the risks associated with physical damage and allow for rapid data collection in a digital realm. However, the computational demands of policy learning can still be a bottleneck, even with efficient data collection. Our research employs Field-Programmable Gate Arrays (FPGAs) to accelerate the reinforcement learning process for robotic arms. FPGAs are adept at accelerating both training and inference algorithms in machine learning, particularly useful given the typically smaller neural networks used in robot policy learning compared to those in supervised learning. This allows FPGA-accelerated training to exploit fine-grained parallelism in compact models efficiently. Moreover, FPGA accelerators, being more power-efficient than CPUs and GPUs of equivalent performance, are particularly suited for battery-powered devices, facilitating on-site retraining to quickly adapt to changing environments [4].

This tutorial paper builds upon previous work [5], featuring an improved physical robotic arm, simulator, software reference, and policy quality tests. While focusing primarily on a 4-DoF (4 Degrees of Freedom) robotic arm for a simplified demonstration, the sim-to-real method and FPGA-based accelerator are applicable to a wider range of robotic systems. For more information, please refer to [2] and [6].

The following sections of this tutorial are organized as follows: Section II provides relevant background information. Section III introduces the policy learning platform. Section IV details the FPGA accelerator for policy learning. Section V evaluates training time and policy quality in an object manipulation task. Finally, Section VI offers concluding remarks and future research directions.

II. Background

Sim-to-real policy learning in robotics involves two stages: **training in simulations** and **policy transfer**. Initially, robots are trained in simulated environments, where they safely learn diverse and complex policies without real-world risks. This phase is crucial for developing adaptable and robust action strategies. The second stage transfers these policies to real settings, ensuring their effectiveness and practicality. Key applications, as detailed in [7], [8], and [9], range from touching to grasping. Sim-to-real learning effectively merges simulated learning's safety with the realism of physical deployment, significantly enhancing robotic training and deployment efficiency.

The sim-to-real learning system presented in this paper is built upon the **Deep Deterministic Policy Gradient (DDPG)** approach [10]. DDPG stands as a robust algorithm in the field of reinforcement learning, building upon the capabilities of Deep-Q-Networks (DQNs) [11], [12]. This method utilizes an actor-critic architecture, combining elements from both value-based and policy-based approaches. Within this framework, the actor network determines the optimal policy—specifying actions in given states—while the critic network assesses this policy by estimating the expected cumulative reward.

The DDPG algorithm functions as follows: It begins with the **data collection** phase, where interaction data between the simulated robot and its environment are gathered and stored in a replay buffer. This data forms the training set for

the neural networks. Following data collection, the algorithm undergoes an iterative process. In each iteration, the **gradient computation** step calculates gradients for both the actor and critic networks. Subsequently, the **model update** step occurs, where an optimizer adjusts the weights of these networks based on the calculated gradients. This process of weight adjustment and network refinement continues across multiple cycles. Detailed information on the algorithm's specific steps can be found in [10] and [13]. Applications of DDPG in sim-to-real learning have been demonstrated in tasks such as object pushing [14] and the manipulation of deformable objects [15].

The gradient computation step in the DDPG algorithm poses a significant computational challenge. To address this, we propose the use of Field-Programmable Gate Arrays (FPGAs) to accelerate the process. Leveraging FPGA-accelerated machine learning offers remarkable advantages in speed and energy efficiency [16], making it a compelling option in robotics applications. Recent advancements in design automation techniques [17], [18] further enhance the feasibility and attractiveness of developing FPGA-based machine learning accelerators.

The body of research on FPGA-based reinforcement learning is diverse, covering accelerators designed for various algorithms. This includes deep Q-Learning [19], [20], Asynchronous Advantage Actor-Critic (A3C) [21], Trust Region Policy Optimization (TRPO) [22] along with its advancements in 2018 [23], and Proximal Policy Optimization (PPO) [24]. Additionally, our publication [5] is pioneering in two significant aspects. Firstly, it represents the first instance of applying FPGA acceleration to the DDPG algorithm. Secondly, it is notable for being the initial study to successfully implement policies learned through FPGA-accelerated learning in the control of a physical robotic arm. This breakthrough enables the orchestration of object movements within a three-dimensional space through continuous actions.

III. FPGA-ACCELERATED SIM-TO-REAL POLICY LEARNING PLATFORM

This section introduces a sophisticated policy learning platform tailored for robotic arms. Our platform distinguishes itself from those discussed in [23] and [5] in terms of kinematics and action space.

The platform, illustrated in Fig. 1, encompasses six main components. The **robot control software**, operating on a microcontroller, handles the actions of the **physical robotic arm**. In parallel, another computer runs dedicated **control software for simulation** that controls a **simulated robotic arm**. This control software leverages a policy from the policy learning software, which orchestrates control signals for the simulated robot and collects feedback. A pivotal component is the **policy learning software**, tasked with gathering interaction data from the simulation control software. This data is converted into training material for the reinforcement learning algorithm. A key innovation in the platform is the **learning accelerator**, built on an FPGA platform. This accelerator is designed for rapid gradient computation within the reinforcement learning model's networks. These gradients are critical for the reinforcement learning algorithm to improve the policy.

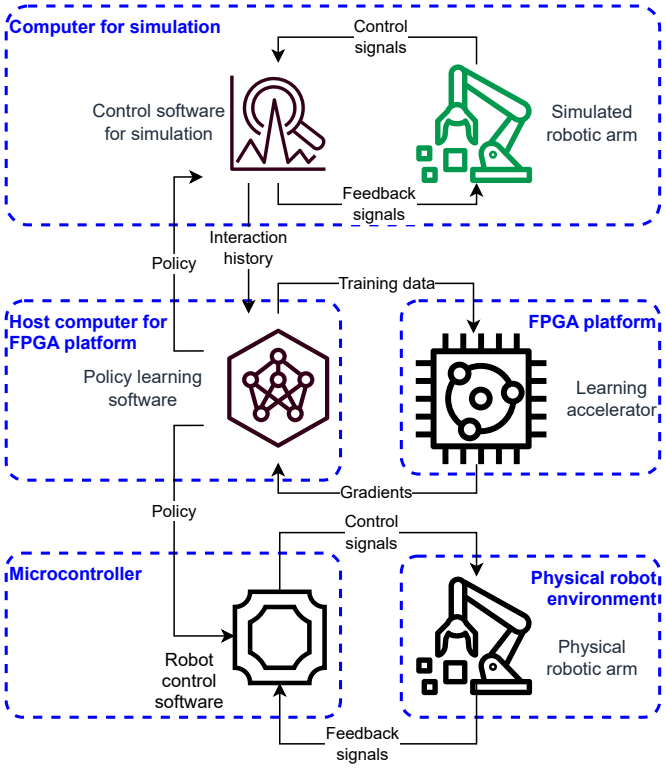

Fig. 1. Design of policy learning platform including a physical robotic arm and its simulated counterpart.

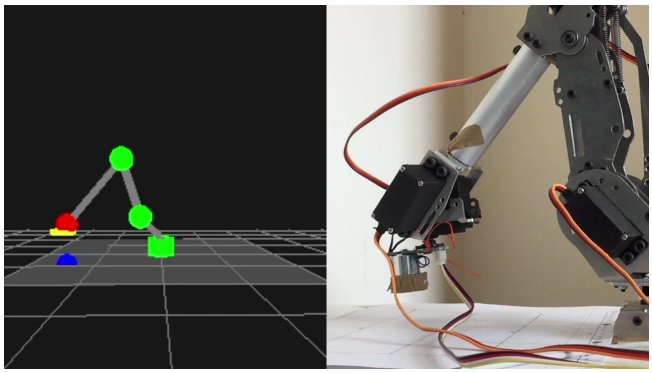

Fig. 2. A snapshot of the experiment when the simulated robot and its physical counterpart are in the same state while executing an identical policy.

The simulator, periodically updated with the refined policy from the policy learning software, continues to accumulate interaction records. Once the policy reaches an adequate level of development, it is implemented into the robot control software. Consequently, the microcontroller can utilize the policy to govern the physical robotic arm's movements.

Our platform utilizes a 4-DoF (Degrees of Freedom) physical robotic arm, as depicted in Fig. 2 alongside its simulated counterpart. This robotic arm comprises four solid links connected by three joints, with an electromagnet serving as the end effector. In our experiments, this electromagnet is designed to lift an iron cylinder. The simulator replicates the behavior of the robotic arm in discrete time steps. At each time step's start, the arm selects an action from a predefined action space,

which the simulator then executes in the virtual environment, providing feedback on the resulting state at the time step's end.

The action space of the robotic arm includes five signals. Four signals correspond to the rotation rate of each motor, where the sign of these signals indicates the rotation direction. The fifth signal is dedicated to the electromagnet's activation status.

IV. FPGA-accelerated DDPG design for policy Learning

A crucial element of the policy learning platform is the FPGA-based learning accelerator. Working in tandem with the policy learning software on the CPU host, this accelerator significantly enhances the efficiency of policy learning. In this setup, the CPU sends training data to the FPGA for the computation of network parameter gradients. These gradients are then relayed back to the CPU. On the CPU platform, an optimization algorithm utilizes these gradients to update the network parameters.

The actor and critic networks in the DDPG model, considering the defined action and state spaces, both utilize fully-connected layers. This structure primarily involves numerical calculations for gradient computation, which include the propagation of activation and error signals. With the ReLU activation function employed in both networks, the signal propagation can effectively be described as a series of dot product operations.

We organize gradient calculations into two distinct computational patterns: the **distributive pattern** and the **collective pattern**. For each pattern, we design specialized hardware processing elements (PEs) for each pattern.

- The **distributive pattern** is used when all N input signals of a layer are accessible at the start of computation. In this scenario, a specific PE, called a **distributive PE**, conducts computations across multiple cycles. Specifically, it works by initializing the dot product calculation for K pairs of M-dimensional vectors during each cycle. In other words, within one cycle, the PE takes information from all M signals and distributes it to produce K output signals. After the pipeline overhead of dot product calculation, K output signals are available in each cycle. Therefore, emitting all the M output signals requires $\lceil M/K \rceil$ consecutive cycles.
- The **collective pattern** emerges when the N input signals of the layer become available gradually over multiple cycles. A different type of hardware, the **collective PE**, accumulates partial results for all output signals using the information of K newly available input signals in each cycle. In other words, this PE collects information from K input signals during every cycle. The M accumulated results cannot be treated as output signals during the accumulation process since they have incomplete information from the input signals. However, these results become valid at the same time after $\lceil N/K \rceil$ cycles when the PE finishes collecting information from all N input signals, resulting in complete output signals.

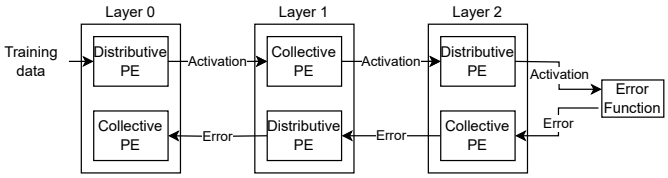

Fig. 3. A chain of distributive and collective PE to accelerate gradient computation for a 3-layer actor network. Each layer of the network maps to two PEs controlling the signal propagation in two directions.

The initiation of signal propagation commences from the input layer of the network, where the initial activation signals correspond to the training data. This known training data prompts the computation to follow the distributive pattern. Consequently, as the activation signals within the subsequent layers become incrementally available, the network transitions to the collective pattern. The output derived from the collective pattern then feeds into the successive layer, thereby reinstating the distributive pattern. Upon the activation signal's culmination at the network's terminus, the error signal can be computed in a pipelined manner, commencing error propagation that differs from the final activation signal pattern.

To accelerate signal propagation for gradient calculations, we design an architecture with a chain of PEs. The choice between the two types of PEs depends on parity of the layer's position in the sequence of signal propagation. By interconnecting these PEs in accordance with the layout of the actor and critic networks, gradient computation can be accelerated. For example, Fig. 3 illustrates a design example for an actor network comprising three layers.

V. Evaluation

This section presents an empirical evaluation to demonstrate the potential of the proposed policy learning platform based on the DDPG algorithm.

A. Experiment Setup

The primary goal of this experiment is to relocate an iron cylinder from its initial position to a predefined target position. This task represents a standard relocation challenge in robotic manipulation, as outlined in [25]. To guide the policy optimization process with greater accuracy, we have customized the reward function in line with the methodology described in [26]. The task is segmented into discrete components, with each component receiving a relative weight. These weights are fine-tuned through numerous iterations of simulated trials, ensuring optimal task performance. Formally, the reward function for a state S is represented as follows:

$$R(S) = -\sum_{i=1}^{4} w_i r_i(S) \tag{1}$$

where $w_1 \dots w_4$ are the weights of the reward components. $r_1(S)$ represents the Euclidean distance between the end effector and the iron cylinder. $r_2(S)$ is the inclination angle of the end effector. $r_3(S)$ equals 0 if the electromagnet is close to the iron cylinder; otherwise, $r_3(S)$ equals 1. Lastly, $r_4(S)$

TABLE I
POLICY QUALITY AND LEARNING EFFICIENCY

Task	Initial Position	Target Position	TRPO [23]	DDPG	Training Time PyTorch (sec)	Training Time C+FPGA (sec)	Speedup (times)	Gradient Time PyTorch (sec)	Gradient Time C+FPGA (sec)	Speedup (times)
A	(200,0)	(200,100)	Success	Success	1102.86	71.16	15	1062.05	30.11	35
B	(200,0)	(100,100)	Failure	Success	1114.83	78.86	14	1074.02	37.96	28
C	(100,0)	(100,50)	Success	Success	1213.11	77.75	16	1168.59	32.48	36
D	(100,50)	(100,0)	Success	Success	1016.77	70.36	14	979.67	31.43	31
E	(100,0)	(50,50)	Failure	Success	1539.11	95.02	16	1483.46	39.13	38
F	(50,50)	(100,0)	Failure	Success	1119.90	75.82	15	1079.09	34.67	31

TABLE II
RESOURCE USAGE ON INTEL STRATIX-V FPGA

Resource	Utilisation	Available	Percentage
Logic utilisation	143569	262400	54.71%
- Primary FFs	304025	524800	57.93%
- Secondary FFs	14221	524800	2.71%
Multipliers (18x18)	3769	3926	96.00%
DSP blocks	1913	1963	97.45%
Block memory (M20k)	1913	2567	74.52%

quantifies the Euclidean distance between the iron cylinder and the target position, serving also as the termination condition.

In the learning efficiency experiments, all training time measurements for the CPU platform are derived from a Python implementation of the identical DDPG model utilizing PyTorch 2.0. The 8-thread software operates on a workstation equipped with an Intel Core i7-6700 CPU (14nm, 4 cores, 3.4 GHz). We do not measure the CPU time for the simulator because it can be amortized. In particular, the simulation time to produce a data set is far shorter than the spent on DDPG training using the data set. Therefore, while the training procedure is consuming a data set, the simulator has sufficient time to collect another one.

The proposed learning accelerator is realized on the Maxeler MAX4 platform, housing an Intel Stratix-V FPGA (28nm, 200 MHz). Each network is composed of a solitary hidden layer featuring 108 ReLU nodes. The actor network has an output layer with 5 TanH nodes, while the critic network's output layer comprises only one linear node. The input batch size is set at 8. On the FPGA platform, all multiplications employ 32-bit fixed-point numbers with 8 integer bits and 24 fractional bits.

B. Results and Discussion

We apply the policies acquired within the simulated environment to our physical robotic arm to assess their quality. Following the completion of training within the simulation, the parameters of the actor network are saved locally. These parameters, extracted from different episodes of training, are preserved every 100 episodes thereafter, reaching up to 2000 episodes. This experiment encompasses the evaluation of six distinct tasks. In each task, the robotic arm is tasked with relocating an object from a designated initial position to a specified target position. For a task to be deemed successful, the robotic arm must place the iron cylinder within a range of 10mm from the target position.

The results on policy quality and learning efficiency are shown in Table I. The resource utilization of the FPGA design is shown in Table II. The comparative analysis encompasses the conventional DDPG method in PyTorch, the FPGA-accelerated design of the DDPG algorithm, and the FPGA-based design for Temporal Difference Policy Optimization (TRPO) methodology developed in [23].

Columns 1–3 in Table I show the task settings. Columns 4 and 5 show whether the TRPO design in [23] and the proposed DDPG design can successfully finish each task. Given the disparate algorithmic underpinnings, a direct comparison between TRPO and DDPG in terms of training time is unfair. Accordingly, we focus on whether TRPO can successfully accomplish each task. Columns 6 and 7 present the overall training time, covering gradient computation, gradient transmission, and model update. Less training time means faster convergence within the policy search process. Column 8 shows the speedup of the FPGA-accelerated DDPG design over the PyTorch design. We can see that the DDPG designs are able to find usable policies in all six tasks, while the TRPO design can only succeed in tasks A, C, and D. Regarding the two DDPG designs, it is anticipated that the PyTorch-based design attains convergence with marginally fewer training episodes than its FPGA-accelerated counterpart, primarily due to its higher numeric precision. Nonetheless, since the FPGA-accelerated design spends less time on each training episode than the PyTorch design, the FPGA-accelerated design still demonstrates 14–16 times speedup.

We notice that the FPGA does not demonstrate its full potential in the learning procedure for two reasons. First, gradients computed on the FPGA need to go back to the CPU software, and there is an IO bandwidth bottleneck in the PCIe interface for this data transmission operation. Second, the model update operation in the software is not accelerated on the FPGA since it does not constitute a performance bottleneck. To estimate the full potential of the FPGA chip, we record the execution time spent on gradient computation in Column 9 and 10 for the two DDPG designs. Column 11 shows the corresponding speedup. The isolation of data transmission and model update enables us to gauge the speedup afforded by the computation kernel. The execution time under this setting corresponds to the FPGA chip's theoretical maximum acceleration, assuming infinite bandwidth with the host computer and zero execution time for model updates. The results show that the maximum speedup of the FPGA-accelerated design can reach 28–38 times, which is higher than the speedup for the whole training procedure. As the model size increases with the complexity of the robotic system, gradient computation tends to consume a larger proportion of

the learning time. In such cases, the speedup of the entire training process is likely to approach this theoretical upper bound.

Besides the FPGA, our evaluation involves the use of an NVIDIA GeForce RTX 4090 GPU with PyTorch CUDA support. To maintain brevity, we omit the results, as the training acceleration with the GPU was consistently slower than that achieved with the FPGA in all tested scenarios. This observation aligns with findings from a previous empirical study [27].

VI. Conclusions and future work

This tutorial paper presents a reinforcement learning framework uniquely designed to execute the Deep Deterministic Policy Gradient (DDPG) algorithm on Field-Programmable Gate Arrays (FPGAs). Our platform integrates a physical robotic arm and its corresponding virtual simulator. The FPGA-accelerated DDPG architecture is specifically developed to facilitate efficient policy learning through the use of the virtual robotic arm simulation. Despite facing input-output (IO) bandwidth limitations between the CPU host and the FPGA, our approach achieves significant acceleration while maintaining the integrity of the learned policies. These policies, represented in fixed-point numerals, effectively direct the physical arm's maneuvers within a three-dimensional environment.

Looking ahead, our study will focus on two main areas of future work. Firstly, we aim to explore the potential of FPGAs in enhancing the learning process by leveraging data from low-precision simulations. Secondly, we plan to conduct a thorough analysis and optimization of energy consumption during both the simulation and training phases.

Acknowledgments

The support of UK EPSRC (grant number EP/V028251/1, EP/L016796/1 and EP/N031768/1), SRC and Intel is gratefully acknowledged.

References

[1] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.

[2] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[3] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning," *IEEE Access*, vol. 9, pp. 153 171–153 187, 2021.

[4] W. Jiang, H. Yu, and Y. Ha, "A high-throughput full-dataflow mobilenetv2 accelerator on edge FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[5] C. Guo, W. Luk, S. Q. S. Loh, A. Warren, and J. Levine, "Customisable control policy learning for robotics," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160-052X, 2019, pp. 91–98.

[6] C. F. Joventino, R. de Abreu Alves e Silva., J. H. M. Pereira, J. M. S. C. Yabarrena, and A. S. de Oliveira, "A sim-to-real practical approach to teach robotics into k-12: A case study of simulators, educational and diy robotics in competition-based learning," *Journal of Intelligent & Robotic Systems*, vol. 107, no. 1, p. 14, Jan 2023. [Online]. Available: https://doi.org/10.1007/s10846-022-01790-2

[7] C. Yuan, Y. Shi, Q. Feng, C. Chang, M. Liu, Z. Chen, A. C. Knoll, and J. Zhang, "Sim-to-real transfer of robotic assembly with visual inputs using cyclegan and force control," in *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2022, pp. 1426–1432.

[8] Y. Lin, J. Lloyd, A. Church, and N. F. Lepora, "Tactile gym 2.0: Sim-to-real deep reinforcement learning for comparing low-cost high-resolution robot touch," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 754–10 761, 2022.

[9] Y. Liu, H. Xu, D. Liu, and L. Wang, "A digital twin-based sim-to-real transfer for deep reinforcement learning-enabled industrial robot grasping," *Robotics and Computer-Integrated Manufacturing*, vol. 78, p. 102365, 2022.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[13] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[14] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[15] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning*. PMLR, 2018, pp. 734–743.

[16] S. Liu, H. Fan, M. Ferianc, X. Niu, H. Shi, and W. Luk, "Toward full-stack acceleration of deep convolutional neural networks on FPGAs," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 8, pp. 3974–3987, 2022.

[17] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "FPGA HLS today: Successes, challenges, and opportunities," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 4, pp. 1–42, 2022.

[18] J. Vandebon, J. G. F. Coutinho, W. Luk, and E. Nurvitadhi, "Enhancing high-level synthesis using a meta-programming approach," *IEEE Transactions on Computers*, vol. 70, no. 12, pp. 2043–2055, 2021.

[19] J. Su, J. Liu, D. B. Thomas, and P. Y. Cheung, "Neural network based reinforcement learning acceleration on FPGA platforms," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 4, pp. 68–73, 2017.

[20] L. M. Da Silva, M. F. Torquato, and M. A. Fernandes, "Parallel implementation of reinforcement learning q-learning technique for fpga," *IEEE Access*, vol. 7, pp. 2782–2798, 2018.

[21] H. Cho, P. Oh, J. Park, W. Jung, and J. Lee, "FA3C: FPGA-accelerated deep reinforcement learning," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 499–513.

[22] S. Shao and W. Luk, "Customised pearlmutter propagation: A hardware architecture for trust region policy optimisation," in *International Conference on Field Programmable Logic and Applications*. IEEE, 2017, pp. 1–6.

[23] S. Shao, J. Tsai, M. Mysior, W. Luk, T. Chau, A. Warren, and B. Jeppesen, "Towards hardware accelerated reinforcement learning for application-specific robotic control," in *International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2018, pp. 1–8.

[24] Y. Meng, S. Kuppannagari, and V. Prasanna, "Accelerating proximal policy optimization on CPU-FPGA heterogeneous platforms," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020, pp. 19–27.

[25] A. Billard and D. Kragic, "Trends and challenges in robot manipulation," *Science*, vol. 364, no. 6446, p. eaat8414, 2019.

[26] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation," *arXiv preprint arXiv:1704.03073*, 2017.

[27] Y. Meng, C. Zhang, and V. Prasanna, "FPGA acceleration of deep reinforcement learning using on-chip replay management," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, 2022, pp. 40–48.