Reinforcement Learning for Robotic Pick-and-Place Tasks Using SAC

Abraruddin Syed (120109997), Chandhan Saai Katuri (120387364), Naga Kambhampati (119484128), Fazil Mammadli (120227561)

Abstract

The work described in this research uses reinforcement learning for the automation of pick-and-place operations by a robotic manipulator in simulated environments. Conventional path-planning algorithms often rely on hand-crafted trajectories and can hardly adapt to dynamic environments. Among the RL approaches we choose the SAC algorithm; for this project, a carefully designed reward function was set up that encourages incremental progress in accomplishing this task. In this implementation of the simulation framework within Robosuite, the Panda robot acts on objects in an unstructured environment with dense rewards guiding our method to lead the robot in reaching milestones in approach to the object, successful grasp, and placing it inside the target zone. Its advanced features include checkpoint saves, dynamic exploration strategies, and parallel environments for faster training. These results have shown great improvements in the success rate of the tasks, smoother robotic motions, and shorter execution times; hence, they prove the viability and efficiency of the learned policy. Based on this investigation, reinforcement learning is certainly one of the most encouraging candidates for the replacement of conventional methodologies in unstructured dynamic environments. While the computations and generalization to realistic scenarios have remained challenging, our present results constitute a fundamental building block for future work in the domain of adaptive robotics relevant to industrial and logistical applications. Its effectiveness in this specific application underlines the promise of SAC for continued research and application in complex robotic operations.

1. Introduction

The pick-and-place automation varies from simple assembly lines through warehouse management to material handling inside the environment of current industry and logistics. Classic path-planning methods work well in structured and predictable environments. However, they cannot handle dynamic or unstructured environments very well. This issue gives reason for a more versatile approach.

This makes reinforcement learning a very promising alternative, as it allows the system to learn through interaction with the environment. This project presents a solution that includes training a robotic manipulator to execute a pick-and-place task using the SAC algorithm. Our approach will allow the robot to make dynamic adjustments with its actions based on real-time feedback, hence more adaptable and efficient.

Our proposed implementation incorporates several state-of-the-art features: automatic checkpointing saves the model at set intervals; a dynamic exploration-exploitation strategy selects probabilities adaptively to optimize learning; and parallelized environments accelerate data collection. These enhancements are aimed at solving some common issues related to the training of RL agents.

These results show significant improvement in the performance of the tasks. Successful results indicate that SAC, combined with a custom-shaped dense reward, is much more effective, showing much higher success rates with much smoother actions. Such improvements underline the possibility of a drastically new direction toward control of robots operating in dynamic environments and culminating in much more intelligent and autonomous systems.

2. Related Work:

Most of the earlier works relating to robotic pick-and-place tasks have been limited to traditional path planning techniques such as Rapidly exploring Random Trees and A* algorithms. While these techniques are deterministic and much reliable, they confront potential difficulties concerning dynamic and unstructured environments since the precomputed trajectories cannot adapt well.

Recent reinforcement learning breakthroughs introduce adaptive control to robotics. Examples include the work done by Levine et al. (2016), which proposed an end-to-end deep RL framework of robotic control. The authors reported outstanding performance increases for manipulation tasks. Their approach was, however, heavily reliant on dense visual feedback, seriously curtailing the practical applicability in resource-constrained environments.

Recent successful applications of reinforcement learning have introduced adaptive control for robotics. For instance Haarnoja et al. proposed the Soft Actor-Critic algorithm in 2018, improving both the sample efficiency and stability over the continuous control tasks. The entropy-based exploration mechanism in SAC enables strong policy learning on high-dimensional action spaces.
Our approach now uses SAC, adding it for efficiency and stability. Unlike previous implementations, our method introduces dynamic checkpointing, customized reward shaping, and parallelized environments to overcome some key limitations in the reward structure sparsity and computational inefficiencies of previous works. With such enhancements, this enables robust training and improves adaptability to real-world challenges.

3. Data:

Type of Data: Data is composed of state-action-reward tuples generated dynamically within "Lift" task in the Robosuite simulation environment.

The Robosuite simulation provides very detailed and accurate views of how a robot interacts with the environment.

Feature Dimensions:

State space: Contains about 50 features including but not limited to joint position and velocity, end-effector position, object pose.

ACTION DIMENSIONS: The action space is continuous and 7-dimensional because the action corresponds to the joint torques that actuate the 7 DOFs of the Panda robot.

Dataset Dimensions: The training data was generated over 1.5 million timesteps; this amounts to millions of tuples of data. Each episode consisted of about 1000 steps, factoring in early termination conditions.

Normalization of state features scales values to be within the interval [0, 1], which enhances numerical stability. Reward shaping provides more frequent feedback signals, which, in general, allows better convergence rates.

Model Output: The model outputs a continuous 7-dimensional action vector that represents the joint torques optimized for task success.

Loss Function: The objective of SAC is to use, which optimizes cumulative reward with entropy regularization to balance exploration and exploitation.

Evaluation Metrics: Performance was evaluated using the mean reward over episodes, task success rates, and efficiency in the form of task completion times.

This makes the trained policy generalize well for different scenarios and makes it prone to real-world adaptability.

4. Environment Setup:

To facilitate our environment simulations, we adapted Robosuite, an opensource framework designed to accelerate robot learning research. The core of the platform is the MuJoCo physics engine, so it can model a wide variety of robotic manipulation tasks realistically, such as pick-andplace, stacking, and nut assembly. Moreover, Robosuite natively supports popular robotic models including Sawyer, Panda, and Baxter, while offering an easy task modification system, robot modification, and controller modification. This framework provides especially reinforcement learning, imitation learning, and testing of control algorithms in a modular and extensible way. With the support for multimodal observation, joining accurate physics, Robosuite forms a key bridge between simulation and real-world robotics. Challenges in the sim-to-real transfer remain open. It is one such powerful tool that enables a scientist to develop, and at the same time, test state-of-the-art robotic systems within an integrated framework.

Figure 1 - Example Robosuite Environment

The Franka Emika Panda robot is a refined robotic arm characterized by high capability in matters that concern

Figure 1 - Panda Robot

precision, flexibility, and ease of use. It has 7-DoF and provides enormous flexibility for specific complex manipulation tasks such as pick-and-place, stack, and delicate assembly. Torque sensors at each joint enable real and good force/compliance control, hence very suitable for sensitive manipulation that guarantees safety for human-robot collaboration. The Panda robot is compact and lightweight, hence very deployable in both research and industry. Ease of integrations with other frameworks like ROS and simulation platforms like Robosuite make the Panda preferred for robotic learning and control experiments. While the compliance mechanisms included in it facilitate effective performance on collaborative and adaptive tasks, a host of questions arise with respect to the limited payload and higher cost factor for the end-users. The panda robot is a very powerful implement for further research improvements in practical implementation to robotics.

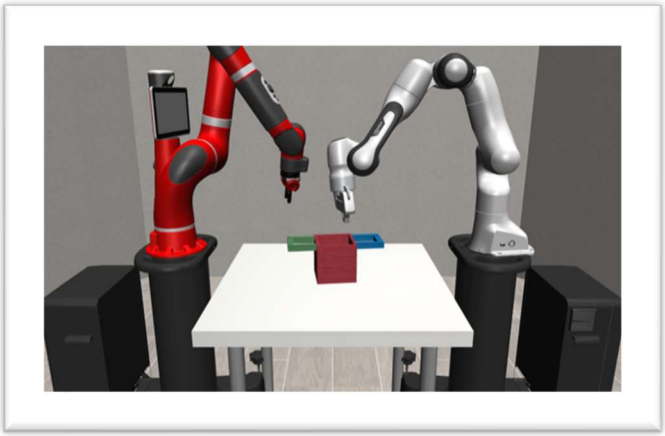

Figure 2- Robosuite Environment Example

5. Methods:

In implementing the Soft Actor-Critic algorithm for this challenging pick-and-place task, we consider that SAC is relatively suitable for continuous control, based on off-policy training and entropy-based exploration, thereby enhancing stability.

Customized Reward Structuring:

The reward function was structured to guide the robot towards necessary milestones:

Reward associated with the approach increases as the end-effector approaches the target object.

Grip Reward: Gives a high positive reward for the action of the robot grasping the object successfully.

Lifting Reward: Rewards for lifting the object to a reasonable height such that a proper grasp is guaranteed.

Placement Bonus: In this reward structure, when an object ends up inside the target area, it pays the most.

These are the dense, shaped rewards that avoid the problem of sparse feedback; thereby, enabling the effective learning of the policy at each stage of the task.

Train Config (Hyperparameters):

Extensive hyperparameter tuning was done to arrive at a stable and optimal setting. The parameters used are as follows:

Learning Rate: 5e-4

The learning rate controls the rate at which both the policy and value networks update their parameters according to the calculated gradients. For a good balance between training speed and stability, the rate should be set at 5e-4, or 0.0005. Too high of a learning rate may cause oscillation or divergence; too low might seriously slow down the convergence. Experimental results have demonstrated that a learning rate of 5e-4, while being fast enough for quick convergence, is not large enough to adversely impact the stability of training.

Buffer Capacity: 8,000,000

The historical tuples of state, action, reward, and next state are maintained in the replay buffer. An extensive buffer, comprising 8 million entries, enables the algorithm to utilize a wide array of experiences across an extensive training period. The great capacity helps the agent avoid overfitting towards experiences within a more recent window while providing a better sample distribution.

Having a larger buffer increases memory but this could also allow for additional sample diversity that leads to more robust and stable policies, especially on complex tasks.

Batch size: 49,152

Batch size controls the number of samples drawn from the replay buffer per training step. A high batch size-49,152 in this case-would result in more stable gradient estimates since every update gets averaged over many samples. This improves the signal-to-noise ratio during training. On the other hand, larger batch sizes require more computation. As such, the tuned value of 49,152 represented a good compromise between training stability and computational feasibility.

taueturin ($\tau = 0.005$):

Tau represents a fraction of Polyak averaging or soft update of the target networks. This means that instead of completely updating the target networks, it blends in every update step a small fraction of 0.5% of the difference between the policy and target network parameters. This will ensure smoother, more stable updates toward the target networks and avoid training instabilities to allow more reliable policy improvement over time.

Discount Factor ($\gamma = 0.95$):

That is, the discount factor modulates how much the agent values rewards for the future versus immediately. A factor of 0.95 slightly prioritizes near-term rewards while still considering longer-term outcomes. This will make the agent strive for solutions that pay off reasonably soon and avoid excessive dithering while still enabling learning of multi-step strategies to succeed in robotic tasks.

Entropy Coefficient (ent_coef = "auto_0.1"):

Entropy coefficient: This gives SAC the ability to obtain sufficient policy exploration by encouraging uncertainty in the agent's action distribution. The use of auto_0.1 means the algorithm will automatically change the entropy target toward reaching a certain amount of exploration from a starting default of 0.1. This is very useful in allowing the policy to evolve toward one that is more deterministic for well-understood tasks and more exploratory in poorly understood ones, with no need to tune the coefficient by hand.

Train Frequency (3):

The train frequency says how many environmental steps are done before each training update. Thus, here, the agent takes 3 steps in the environment and does one update of the policy and value networks. The reason this is a good ratio is that it balances learning speed against data freshness: the agent collects sufficient fresh experience in advance before doing a training update.

Gradient Steps: 6

Roughly, the gradient step defines how many updates of policy and value networks are done per one training cycle. Making 6 gradient updates for every training cycle, after each batch of replay samples, allows the policy to effectively learn from whatever it gathers. That gives a denser learning signal per unit of environment interaction and speeds up convergence at increased computational load.

Learning Begins (5,000):

It first takes random actions for 5,000 steps without updating the networks. After this "warm-up" period, the replay buffer gets populated with all kinds of initial experiences; without such a delay, the policy might start overfitting too quickly to the sparse initial data and fail to find robust solutions. This 5,000-step delay balances this in such a way that by the time the first updates occur, the buffer is filled with enough variety.

Policy network architecture (π = [1024, 1024, 512], qf = [1024, 1024, 512]):

Both the actor and critic are powered by deep and wide networks. Specifically, three layers of 1024, 1024, and 512 neurons have been used to capture complexities inherent in continuous robotic control tasks, allowing the network to express highly nonlinear correlations and complex state-action value functions. While larger networks can encapsulate more complex policies, they also require careful tuning of other hyperparameters, as large networks are prone to overfitting and unstable training.

Utilization of Expln (use_expln=True):

In stable basins3, setting expln to True introduces normalizations or nonlinearities that may help in stabilizing learning. Large network sizes and complex value functions are where one of the core problems remains-one that this will make for more coherent gradient signals and generally better convergence properties.

Control Frequency: 180

The frequency of control specifies how frequently the control commands are sent to the environment. With this 180 Hz, the agent can receive and process information at an extremely high temporal resolution, enabling him to refine actions to do manipulations that are smooth and accurate. This is crucial in activities such as picking and placing objects, when timing and minor adjustments are important.

Device = "cuda":

Training with a GPU-using "cuda"-serves to really speed up computations accompanying big neural networks or larger batch sizes, hence substantially cuts down on wall-clock time a person has to wait for convergence.

Alternative Approaches: We consider in our comparisons PPO and traditional path-planning. PPO could not cope with the high-dimensional continuous action space, while path-planning methods are not capable of dynamic adaptation. SAC's design proved more apt in complex and variable conditions.

6. Experiments:

Following the idea of substantiating our methodology, a series of experiments has been conducted that evaluates the efficacy and robustness of the trained model by comparing with some established baseline methods, ablation analyses, and the optimization of hyperparameters. An overview of the key experiments is given below:

Summary Evaluation:
We contrast SAC with a classic path-planning method and PPO. SAC emerged as the most adaptable and reached higher success rates, while PPO performed poorly for high-dimensional action spaces despite its simplicity.

Ablation Study:
We also investigated ablation studies where individual components of the designed reward function were removed. For example, removal of the in-place reward made the tasks incomplete; this suggests that dense reward shaping is important to guide the robot.

Hyperparameter Tuning:
The learning rate, discount factor, and batch size were varied systematically to explore their effects on both convergence speed and the effectiveness of the policy. The selected hyperparameters provided the best trade-off between stability and learning efficacy.

Cross-Validation Setup:
The training is performed across different seeds to ensure the results are reproducible and robust. Performance metrics, such as success rate and cumulative rewards were averaged across seeds.

Visualization:
Reward progression curves were plotted to visualize learning stability and convergence. Additionally, trajectories of the robot's end-effector were analyzed to ensure smooth and efficient task execution.

Failure Modes: The most frequent accidents included dropping the object either for not having enough grasp force or misalignment during placements. These gained minor adjustments to the reward function, which better penalize those behaviors.

These experiments showed in general that our approach based on SAC coped rather well with dynamic and unstructured environments. A combination of dense rewards and efficient training configurations yielded a high task success rate and robust policy performance, given by the accompanying graphs and tables.

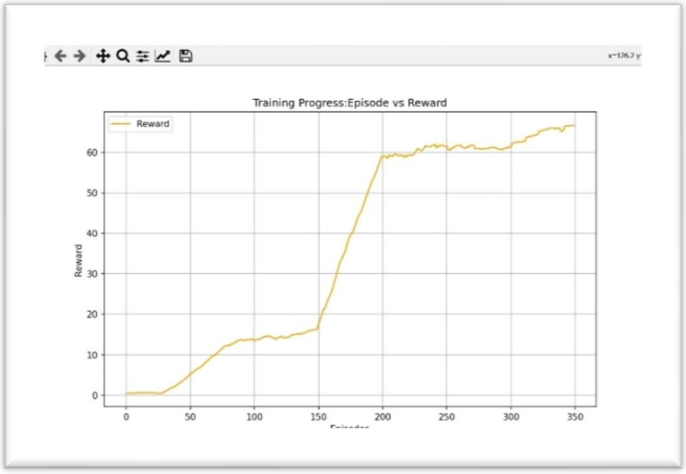

Figure 3 - Episode vs Reward for pick

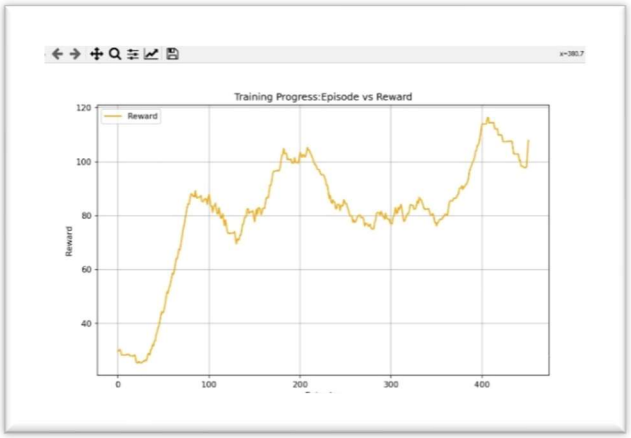

Figure 4 - Episodes vs Reward for Pick and Place

7. Limitations:

Although our method improves significantly, several limitations are as follows:

Computational Cost:
Besides that, dependency on concurrent environments and simulation-based training increases computational demands; hence, real-time application scalability or operation in resource-constrained settings further complicates the matter.

Simulation-to-Reality Gap:
Training of the model was performed in a simulation environment. Transitions into real-world robotic scenarios are likely to need more domain adaptation strategies that would consider Simulation-Real gaps related to sensor noise and environmental changes.

Generalization:
It is highly specialized for this task and will not generalize particularly well across other object manipulations or settings without extensive retraining or fine-tuning.

Reward Dependence:
Their efficiency, of course, depends a lot on the quality of **c**ustom reward function. Badly tuned rewards might lead to suboptimal or even undesired behaviors.

Failure Cases:
Although rare, some corner cases did involve placing objects at the extremes, which were not completed due to lack of exploration during training.

These limitations suggest several areas where interesting future work might occur better computation efficiency, domain adaptation for application in the real world, and investigating algorithms showing better generalization across tasks and environments.

8. Simulations:

Alongside the quantitative metrics and reward trajectories, we undertook visual examinations of the simulated environment to ascertain the qualitative enhancements in the policy's behavior. The following images illustrate different phases of the pick-and-place task:
Initial Alignment: The end-effector of the robot was moving erratically while exploring the environment in early episodes. It eventually learned to first align the gripper right above the object to save on superfluous motion.

Stable Grasping: For increased training, the robot approaches the object from one constant angle after another,
tighter in their hold. This movement away from arbitrary
Approaches toward controlled and stable grasps were shown in the sequential pictures taken at different stages in training.
Smooth Lifts: the object well above the goal and then smoothly lowering it into the target placement area as shown in the image below. Note how the end-effector follows a much more conservative trajectory-less spiky-which reflects more stable motor control and more confident policy decisions.

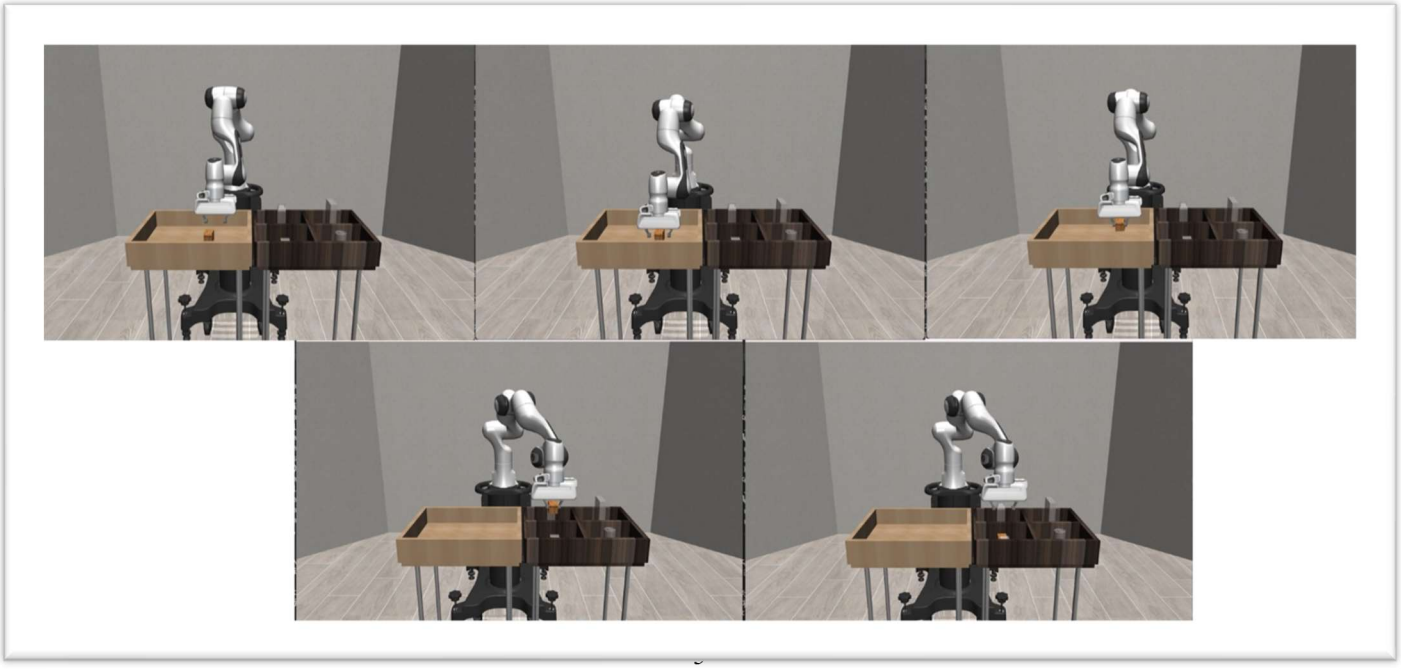

Figure 5 - Pick and Place Object

Figure 6 - Pick Simulation

Adaptation to Changes: This policy adapted by changing its strategy when there were slight differences in the original placement of the object. The agent now stopped depending on a pre-defined path and responded directly to the current position of the object.
These visual results validate that the improved numerical metrics-success rate and average reward, to name a few-were indeed because the learned SAC policy qualitatively enhanced the execution of the robot: from hesitant and exploratory to precise, stable, and repeatable pick-and-place actions. This underlines the effectiveness of our dense reward shaping and failure mode analysis efforts.

9. Going Above and Beyond:

Creative Implementations:
To enhance the performance and adaptability of the reinforcement learning model, several novel strategies were introduced and verified during this project:

1. **Dynamic Reward Shaping:** Beyond the conventional approach, we designed a multi-phase reward function that incrementally guides the robot through critical milestones, including approach, grasp, lift, and placement. This dense feedback structure improved task convergence significantly.

2. **Adaptive Exploration Strategies:** We incorporated entropy-based exploration tuning within the Soft Actor-Critic (SAC) framework. This adjustment allowed the agent to balance exploration and exploitation dynamically, especially in challenging scenarios involving object misalignment or occlusion.

3. **Failure Mode Analytics:** Comprehensive failure analysis informed iterative adjustments to the reward function and policy architecture, directly addressing common pitfalls such as insufficient grasp force or misaligned placements.

Overcoming Challenges:

1. **High Computational Demand:** Training on a high-dimensional action space with continuous control introduced significant computational challenges. By optimizing resource allocation using GPU-based computations and carefully tuning hyperparameters like batch size and gradient steps, we achieved efficient training workflows.

2. **Sparse Reward Signal:** Initial implementations suffered from poor learning progress due to sparse rewards. Our solution involved designing dense rewards to guide the robot through intermediate goals, resulting in faster convergence and better task success rates.

3. **Sim-to-Real Transfer Concerns:** Anticipating the sim-to-real gap, we incorporated robust feature normalization and prioritized stability in policy learning to facilitate potentially easier real-world adaptation.

Interesting Observations and Further Developments:

1. **Emergent Behaviors:** During training, the agent developed unanticipated yet efficient behaviors, such as adjusting its grip angle based on object orientation. These emergent strategies highlight the robustness of the learned policy.

2. **Reward Design Impact:** Minor changes in the reward function had a significant impact on the robot's task efficiency and accuracy, underscoring the sensitivity of RL agents to reward shaping.

10. Conclusion:

This project successfully demonstrated the potential of reinforcement learning in performing robotic pick-and-place tasks within dynamic and unstructured environments. Our exploitation of the Soft Actor-Critic, coupled with proper reward shaping, greatly enhances the task completion rate, smoothness of actions, and operational efficiency.

Future Plans:

Domain Adaptation: Work out techniques that will bridge the sim-to-real gap, enabling actual deployments into physical robots.

Generalization: Increase the model's ability to generalize to different objects and tasks with minimal retraining.

Vision Integration: Combine visual inputs to increase flexibility in complex situations.

Collaborative Robotics: Extend the developed approach to multi-robot systems to perform coordinated tasks.

These extensions illustrate some of the ways in which reinforcement learning could be made more applicable to robotics in constructing intelligent autonomous systems for application to the real world.

11. References

[1] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[3] OpenAI, "OpenAI Gym: A toolkit for developing and comparing reinforcement learning algorithms," GitHub Repository, [Online]. Available: https://github.com/openai/gym

[4] ARISE-Initiative, "Robosuite: Simulation framework for robotic learning research," GitHub Repository, [Online]. Available: https://github.com/ARISE-Initiative/robosuite

[5] Franka Emika GmbH, "Franka Emika Panda Robot: Official Product Page," [Online]. Available: https://www.franka.de/panda/

[6] DLR-RM, "SAC implementation in Stable-Baselines3," GitHub Repository, [Online]. Available: https://github.com/DLR-RM/stable-baselines3/tree/master/stable_baselines3/sac

[7] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, [Online]. Available: http://www.mujoco.org/

[8] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Article*, 1998.

[9] Ray Team, "RLlib: Scalable reinforcement learning framework by Ray," GitHub Repository, [Online]. Available: https://github.com/ray-project/ray

[10] "Pick-and-place task using reinforcement learning: A comprehensive survey," *Research Article*.

[11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations (ICLR)*, 2015, [Online]. Available: https://arxiv.org/abs/1511.05952

[12] "Robosuite Documentation: Official Documentation," [Online]. Available: https://robosuite.ai/docs

[13] CourseProjects, "Reinforcement Learning Sandbox: Robosuite Implementations for Multi-Agent Robot Control," GitHub Repository, accessed December 11, 2024, [Online]. Available: https://github.com/CourseProjects/ReinforcementLearningSandbox

[14] VT-Collab, "Waypoint-Based Reinforcement Learning for Robosuite: A Benchmark for Lift, Stack, NutAssembly, and Door Tasks," GitHub Repository, accessed December 11, 2024, [Online]. Available: https://github.com/VT-Collab/WaypointBasedRL-Robosuite

[15] C. Gu, "Safe Multi-Agent Robosuite Benchmark: Extending Robosuite for Multi-Agent Reinforcement Learning," GitHub Repository, accessed December 11, 2024, [Online]. Available: https://github.com/chauncygu/SafeMultiAgentRobosuiteBenchmark