Data-Efficient Multi-subtask Policy Learning for Robotic Manipulation

Edward Fouad [1] **Kelvin Christian** [1] **Chuanqi Chen** [1]

Abstract

The use of imitation learning or model-free reinforcement learning for a robotic manipulation task requires efficient data collection to offset the time and expense of operating physical robots. The learning process can be simplified by developing policies for shorter subtasks and then combining them into more complex behaviors. We compare the performance of behavioral cloning against two online methods, Proximal Policy Optimization (PPO) and Twin Delayed Deep Deterministic Policy Gradient (TD3), on a benchmark 2-brick stacking environment with a 7 degree of freedom Sawyer arm. We train two separate policies for the pick and place subtasks and the agent learns to switch from one to the other. Behavioral cloning was able to achieve near perfect performance on the pick subtask and 67% success on combined pick & place, whereas the best online method PPO achieved 70% success picking and was unsuccessful at placing. These preliminary results show that subtask decomposition may be an effective method in domains where expert data is obtainable for cloning.

1. Introduction and Motivation

Robotic manipulators have been widely used for decades to carry out repetitive, tedious, and complex manufacturing tasks with speed and precision. However, most existing robots are incapable of handling a new task or a dynamic environment without customized reprogramming and thus do not have a widespread, low-cost, and general-purpose usage.

Reinforcement learning (RL) is a promising alternative to traditional task-specific programming of robots. It enables the agent to conduct basic operations without programming, and thus to handle changes in the tasks without reprogramming. For example, the robotic agent is able to learn how to identify a target object from a cluttered location, pick up the object, and then place it at a designated location with desired pose. However, reinforcement learning with continuous state-action spaces, partially observable states, and high dimensional observations remains challenging.

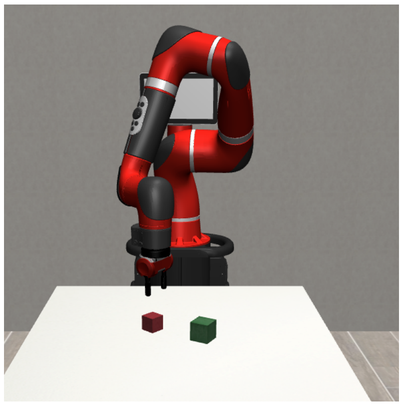


Figure 1. Sawyer robot setup in Robosuite. These colored bricks are placed randomly on the tables. The goal is for the robotic arm to learn an optimal policy to repeatedly stack the red brick on top of the green brick.

We considered two primitive robotic manipulation subtasks: pick and place. Our simulation environment in Figure 1 uses an extension of the Robosuite Stack setup with a Sawyer arm (Zhu et al., 2020). We created an expert policy that can generate data for each subtask and we use behavioral cloning to learn individual pick and place policies. The agent switches to the place policy once the pick policy has issued a close-gripper command for a sufficient period of time. We also contrast the cloned results with those of two model-free online methods: Proximal Policy Optimization (PPO) and Twin Delayed Deep Deterministic Policy Gradient (TD3).

2. Related Work

Model-based policy learning with deep dynamical models can be used to create low-dimensional representations of high-dimensional camera images. A robotic agent can

[1] Stanford University. Correspondence to:
Edward Fouad <efouad@stanford.edu>,
Kelvin Christian <kelv123@stanford.edu>,
Chuanqi Chen <cchuanqi@stanford.edu>,

Repository:
<https://github.com/chuanqichen/CS234_Final_Project>.

then use these representations to perform model predictive control (Wahlström et al., 2015). Many additional RL methods have had success in robotic grasping tasks (Quillen et al., 2018): supervised learning of pose-to-pose movements (Levine et al., 2016), deep deterministic policy gradient with off-policy updates (Gu et al., 2016), and imitation learning from expert demonstrations (Hua et al., 2021).

For autonomous pick and place tasks, it is helpful to decompose the maneuver into multiple steps: approach, manipulation, and retraction. This approach has previously been executed using three separate cycles of Deep Deterministic Policy Gradient algorithm (DDPG) (Lillicrap et al., 2015), which are then combined into a single operation (Marzari et al., 2021). This decomposition also makes it easier to automate the generation of expert data using individual controllers for use in behavioral cloning (Liu et al., 2020). The Twin Delayed Deep Deterministic policy gradient algorithm (TD3) (Fujimoto et al., 2018) further improves both the learning speed and performance of DDPG (Lillicrap et al., 2015) in a number of challenging continuous control settings such as the robotic manipulation tasks we are considering. Alternatively, inverse reinforcement learning can allow the agent to learn a policy directly from expert demonstrations in the absence of an explicitly defined reward function (Abbeel & Ng, 2004).

3. Methodology

We decomposed the stacking problem into three independent low-level subtasks: Moving the robot to the destination, picking up a brick, and stacking the brick. This approach is similar to the approach, manipulate, and retract of (Marzari et al., 2021), except when stacking the brick, the robotic arm needs to move to the destination before placing the brick on top of another. This section will discuss the methods to tackle the task sequencing.

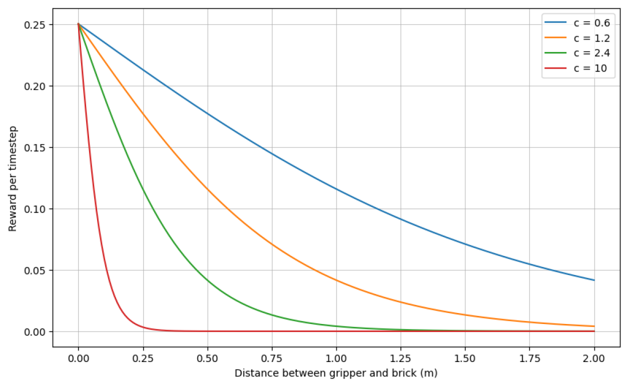


Figure 2. Reward shaping function: $\mathbf{r}_{tasks} = r_i(1 - \tanh(c_i x))$. A higher value c allows the reward to be felt when further from the brick, but with a smaller gradient.

3.1. Simulation Environment

We set up a Robosuite simulation environment (Zhu et al., 2020) created with the Mujoco physics simulator containing a Sawyer robotic arm and two bricks (Figure 1). The Sawyer is a 7 degree of freedom (DoF) single-arm robot with 8th joint for swiveling its display monitor and a parallel-jaw gripper with long fingers capable of grasping a variety of objects.

3.1.1. Reward engineering

The design of a good reward function so as to train the agent with desired behaviors is very important and also very difficult. A sparse reward can be very challenging to learn in a setting with a large, continuous action space. Instead, we shaped a dense reward function based on current progress in the task sequence with four different contributors:

$$\mathbf{reward} = \max(\mathbf{r}_{reach}, \mathbf{r}_{lift}, \mathbf{r}_{stack}, \mathbf{r}_{dist})$$

$$\mathbf{r}_{reach} = \begin{cases} 0.25(1 - \tanh(c_1 x)) + 0.25 & \text{if grasps the brick} \\ 0.25(1 - \tanh(c_1 x)) & \text{otherwise} \end{cases}$$

$$\mathbf{r}_{lift} = \begin{cases} 1 & \text{if brick is lifted} \\ 1 + 0.5(1 - \tanh(c_2 d)) & \text{lifted and aligned} \end{cases}$$

$$(d\text{: distance between two bricks})$$

$$\mathbf{r}_{dist} = 0.01(1 - \tanh(c_3 d_1)) + 0.005(1 - \tanh(c_3 d_2))$$

$$(d_1, d_2\text{: distance from gripper to these two bricks})$$

- $\mathbf{r}_{reach}$: reward for reaching and grasping, add 0.25 if grasping sub-task succeeds. Refer to Figure 2, we choose $\mathbf{r}_{reach}$ with $c_1 = 1.2$ to achieve the best agent behavior to reach closer to the brick.
- $\mathbf{r}_{lift}$: reward for lifting and aligning, $c_2 = 1$
- $\mathbf{r}_{stack}$: reward of 2 for successful stacking
- $\mathbf{r}_{dist}$: reward for getting close to these two bricks, $c_3 = 10$

3.1.2. Simulation configuration

We have a variety of simulation configurations:

- Fixed initial placement vs. random initial placement of these bricks
- Clutter environments with more bricks on the table
- Configure the agent to start in any arbitrary sub-tasks

Our training and testing results presented were from a 2-brick environment with random brick placements. For training the place subtask policy, the setup could be configured to start the training from the final step of the pick task.

3.1.3. Action Space

The 7-DoF arm is controlled using an operational space controller (OSC) with relative position inputs. At each timestep, the robot policy returns a 4-element action command containing a desired $(\Delta x, \Delta y, \Delta z)$ end effector position target and a fourth value whose sign indicates whether the gripper should open or close. The fixed rotation target has the end effector oriented downwards and parallel to the near edge of the table. The OSC computes and outputs the required joint torques based on its tuned internal control gains at a frequency 20 times greater than the policy input.

3.1.4. Observation Space

The simulation environment provides observations of the positions and velocities of the end effector and arm joints, RGB images from a camera on the robot base, and the positions of each brick. In our policy training, we found the most success using observations of the relative position vectors from the end effector to each brick, as well as the open-to-closed positions of the gripper prongs.

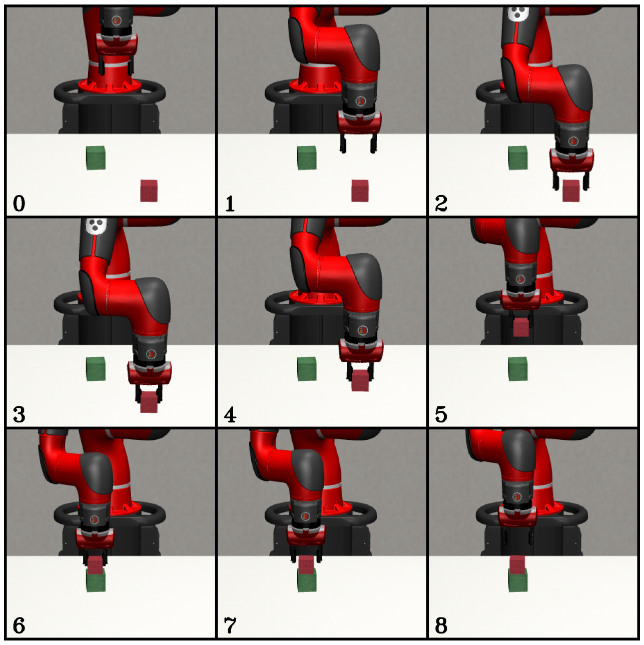


Figure 3. Expert subtask movement decomposition. The expert model defines four movements for the pick subtask (1. prime, 2. descend, 3. grasp, 4. ascend) and four movements for the place subtask (5. prime, 6. descend, 7. ungrasp, 8. ascend). The expert is used to generate random trajectories for behavioral cloning.

3.2. Behavioral Cloning

We used behavioral cloning to generate a pick policy and a place policy by using supervised learning on expert trajectories. The robot switches from the pick policy to the place policy after it has issued a command to close the gripper for 70 consecutive timesteps, after which it assumes the brick has been successfully grasped.

3.2.1. Expert Policy

The "expert" policy can find, acquire, and stack bricks in the Robosuite environment. The expert policy directly accesses the true brick positions and generates a motion path consisting of the nine stacking sequence movements shown in Figure 3. We then add random noise to the generated motion plan and use an Operational Space Controller (OSC) to convert the desired Cartesian space poses into robot joint torques. Our trajectory generator can simulate multiple full stacking episodes using the expert policy. At each step, it records (1) the current movement number, (2) the actions vector produced by the expert policy, and (3) the observation state of the robot. Any unsuccessful episodes are removed from the dataset. A total of 2400 expert trajectories were generated with random initial brick positions, providing a combined 1.4M timesteps.

3.2.2. Policy Learning

The pick and place networks each consist of a small dense network with a 5-neuron input layer containing the relative (x, y, z) displacement to bricks 1 and 2 respectively, as well as the open-to-closed positions of the two gripper prongs. This is followed by a 32-neuron hidden layer and a 4-neuron output layer containing the $\mathbf{\Delta x} = [\Delta x, \Delta y, \Delta z]^T$ relative position target and the signed gripper open/close command g. A ReLU activation function is used after the first hidden layer and a Tanh activation function is used on the gripper command only after the output layer. The pick model is trained using expert data of movements (1, 2, 3, 4), and the place model is trained with movements (5, 6, 7, 8).

Careful tuning of the training loss function was required to achieve correct timing of the gripper open/close command, particularly on the place policy. For a batch of n expert commands in the form $y = (\mathbf{\Delta x}, g)$ and network output $\hat{y} = (\mathbf{\Delta \hat{x}}, \hat{g})$, the loss is defined as

$$\ell(\hat{y}, y) = \frac{1}{N} \sum_{n=1}^{N} \ell_{mse}(\mathbf{\Delta \hat{x}_n}, \mathbf{\Delta x_n}) + \lambda\, \ell_{bce}^{\beta}(\hat{g}_n, g_n)$$

Where ℓ_{mse} is the mean square error loss and ℓ_{bce}^{β} is a modified binary cross-entropy loss in which false positive grasp commands (i.e. the robot fails to open the gripper to place the cube) are penalized more heavily by a weighting factor $\beta \geq 1$. The second weighting factor λ sets the relative importance of these two loss contributors.

3.3. PPO (Schulman et al., 2017)

Proximal Policy Optimization (PPO) is a policy gradient method. It learns the policy parameters (actor) and the value function (critic). Compared to the other policy gradient methods, PPO uses the past batch of datasets for further training, which makes it more data efficient. The idea of re-using the data was first introduced by Trust Region Policy Optimization (TRPO). PPO was mainly made to tackle the complexity of TRPO by introducing the clipped loss function:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon), \hat{A}_t)]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ and $\hat{A}_t$ is an advantage value.

The min operator on the loss function prevents PPO from changing its policy parameters by a large amount. This could make the training more stable.

We make use of the PPO implementation provided in Stable Baselines3 (Raffin et al., 2021).

3.4. Twin Delayed Deep Deterministic policy gradient algorithm(TD3) (Fujimoto et al., 2018)

Deep Deterministic Policy Gradient algorithm (DDPG) (Lillicrap et al., 2015) is an actor-critic method using a learned value estimation to train a deterministic policy from off-policy data. There exists consistent overestimation bias because Q-learning applies the maximization of noisy value estimation. Twin Delayed Deep Deterministic policy gradient algorithm(TD3)(Fujimoto et al., 2018) builds upon DDPG by concurrently training two Q-networks and taking the minimum of the two so as to avoid over-estimation, as outlined in Fig. 4. TD3 further improves performance by updating the policy network less frequently than the Q networks to reduce per-update error. Finally, it adds noise to the target action to create a smoother target policy.

We make use of the TD3 implementation provided in Stable Baselines3 (Raffin et al., 2021).

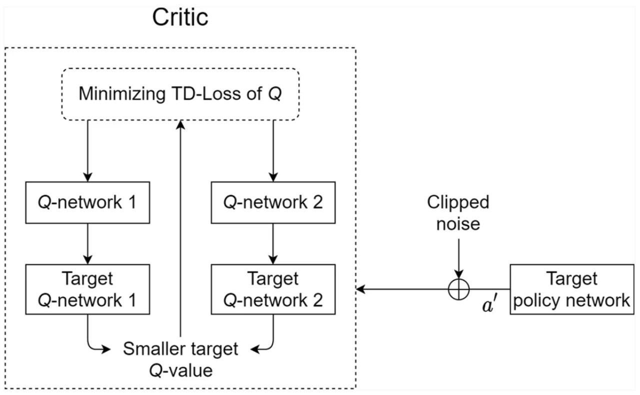


Figure 4. TD3 (Dong et al., 2022)

4. Experiment Results

4.1. Expert Policy

The scripted expert policy completes the brick stacking stack with 99.01% success when evaluated against 10,000 random environments. Despite the gripper being at a fixed orientation, it is very successful at either straightening out the bricks as it grasps them or pinching the corners of the brick. The failures occur when the bricks are initially positioned so closely that the gripper cannot pick the first brick without contacting the second brick.

4.2. Behavioral Cloning

Several cloned policies were trained using different hyperparameter values for the training loss function. The results are compiled in Table 1. While the pick policy has a success nearly on par with the expert policy, the poorer performance of the place policy results in a maximum full sequence success rate of 0.67.

The setting of loss hyperparameters affected the qualitative results significantly. With β too small, the robot does not release the first brick and simply holds it in place above the second brick. With β too large, the robot may drop the brick before it is properly positioned.

The arm switches from the pick policy to the place policy after the pick policy has outputted a close gripper command for 70 successive timesteps. This heuristic worked perfectly: at no point did the robot switch to the place policy while not holding a brick.

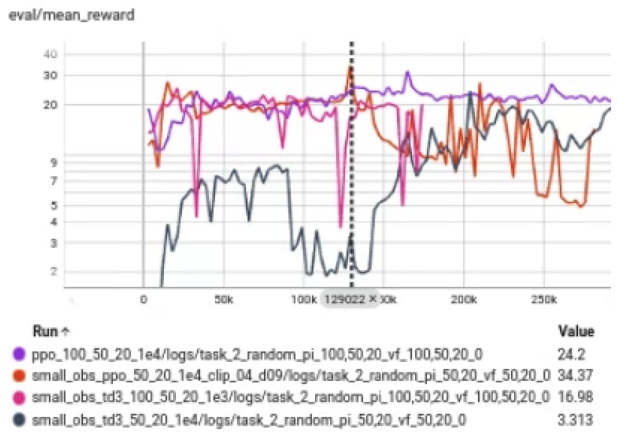
β	$\lambda = 1$	$\lambda = 5$	$\lambda = 20$
1	**0.05**	**0.12**	**0.19**
5	**0.52**	**0.56**	**0.26**
50	**0.23**	**0.67***	**0.58**
100	**0.46**	**0.21**	**0.62**

Table 1. Behavioral cloning success rates on 100 full pick-and-place sequences for each pair of training loss hyperparameters. A larger λ represents a greater emphasis on reducing gripper command error than position command error. A larger β encourages the gripper command to err on the side of open.

4.3. PPO and TD3 model training and evaluation

We used both PPO and TD3 methods to train the policies in online simulation. The agent was evaluated after every 3000 timesteps and performance measured over 10 episodes. The mean rewards from these test evaluations are shown in Fig. 5. We then saved the best-performing model across all evaluations over a 1M timestep training duration.

We iterated through numerous model hyperparameters, simulation configurations, policy / value model architectures, and learning rates. The best performing model is PPO with the policy model architecture $\{50, 20\}$, value function model architecture $\{50, 20\}$, linear decaying learning rate starting at $1e-4$, clipping setting 0.4, replay buffer size 4096, batch size 256, discount factor 0.98, and control frequency 40.



Run ↑	Value
● ppo_100_50_20_1e4/logs/task_2_random_pi_100,50,20_vf_100,50,20_0	24.2
● small_obs_ppo_50_20_1e4_clip_04_d09/logs/task_2_random_pi_50,20_vf_50,20_0	34.37
● small_obs_td3_100_50_20_1e3/logs/task_2_random_pi_100,50,20_vf_100,50,20_0	16.98
● small_obs_td3_50_20_1e4/logs/task_2_random_pi_50,20_vf_50,20_0	3.313

Figure 5. Training evaluation mean reward, PPO and TD3. We evaluate the agent after every 3000 training timesteps for 10 episodes and calculate the mean reward. Intensive architecture search and hyper parameters tuning result in the best performed model of PPO with policy model architecture $\{50, 20\}$ and value function model architecture $\{50, 20\}$.

After the training and evaluation to achieve the best model, we then test the agent in the testing environment. To independently test these primitive sub-tasks, we use the Operation Space Controller to move the robot to the desired starting location of the sub-task and then use the agent's policy to decide the action in the rollout to conduct the sub-task. The testing results of these sub-tasks in term of success rates are shown in Table 2.

Subtask	TD3	PPO
Pick	$\mathbf{0.185}$	$\mathbf{0.704}$
Place	**0**	**0**

Table 2. Success rates pick and place subtask policies trained with TD3 and PPO.

As discussed in section 3.1.1, we spent a lot of time engineering and tuning the reward functions to improve success on on the Pick sub-task. We need further fine tuning to improve the success rate of Place sub-task, as the agent often fails to release the brick.

5. Conclusion

Our goal for this project is to develop a data-efficient method to learn the robot policy by dividing the task into multi-sequential subtasks. We decomposed the task of stacking a brick on top of another into two independent tasks: Picking the first brick and stacking the brick. We have done many experiments using several techniques such as behavioral cloning, TD3, and PPO for each of the subtasks.

Our experiment results showed us that decomposing the tasks into multiple low-level subtasks can significantly improve robot performance. The combined timesteps of all subtasks that are required for convergence are significantly smaller than the convergence timesteps of training the full task. This is because the agent is trained in a simpler environment. The complexity of both action and state spaces has a huge influence on the convergence of the training process. On the other hand, separating the tasks allows us to redefine the reward according to the goal of each individual task. Having the reward modified makes the robot movement more align with human knowledge.

Decomposing tasks adds another complexity in combining the sequential sub-tasks. It can be done by changing the agent according to which subtasks the robot is currently in. Our crucial next step for the policy learning method is to use the same feature extractor network to be applied to different sub-tasks. This allows us to reduce the complexity of the model. On the other hand, we also planned to use a simple neural network to determine the current task which makes the transition smooth.

A data-efficient method is crucial to develop a general low-cost robot. We believe that our study in task decomposition will bring a significant impact on the field of robotics. Future studies might be done to expand the construction of the low-level subtasks to more complicated tasks.

6. Acknowledgements

We appreciate the teaching staff of Stanford CS 234 for their excellent teaching throughout the quarter. We especially thank our mentor Max Sobol Mark for his support and guidance throughout the project.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, pp. 1, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. doi: 10.1145/1015330.1015430. URL https://doi.org/10.1145/1015330.1015430.

Dong, L., Qian, Y., and Xing, Y. Dynamic spectrum access and sharing through actor-critic deep reinforcement learning. *EURASIP Journal on Wireless Communications and Networking*, 2022, 06 2022. doi: 10.1186/s13638-022-02124-4.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods, 2018. URL https://arxiv.org/abs/1802.09477.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, 2016. URL https://arxiv.org/abs/1610.00633.

Hua, J., Zeng, L., Li, G., and Ju, Z. Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning. *Sensors*, 21:1278, 02 2021. doi: 10.3390/s21041278.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection, 2016. URL https://arxiv.org/abs/1603.02199.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N. M. O., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

Liu, Y., Romeres, D., Jha, D. K., and Nikovski, D. Understanding multi-modal perception using behavioral cloning for peg-in-a-hole insertion tasks, 2020. URL https://arxiv.org/abs/2007.11646.

Marzari, L., Pore, A., Dall'Alba, D., Aragon-Camarasa, G., Farinelli, A., and Fiorini, P. Towards hierarchical task decomposition using deep reinforcement learning for pick and place subtasks. In *2021 20th International Conference on Advanced Robotics (ICAR)*, pp. 640–645. IEEE, 2021.

Quillen, D., Jang, E., Nachum, O., Finn, C., Ibarz, J., and Levine, S. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6284–6291, 2018. doi: 10.1109/ICRA.2018.8461039.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Wahlström, N., Schön, T. B., and Deisenroth, M. P. From pixels to torques: Policy learning with deep dynamical models, 2015. URL https://arxiv.org/abs/1502.02251.

Zhu, Y., Wong, J., Mandlekar, A., Martín-Martín, R., Joshi, A., Nasiriany, S., and Zhu, Y. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.