Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 55-37 (2022) 193–198

Deep Reinforcement Learning for Robotic Control with Multi-Fidelity Models

David Felipe Leguizamo * **Hsin-Jung Yang** * **Xian Yeow Lee** * **Soumik Sarkar** *

* *Iowa State University, Ames, IA 50010 USA*
e-mail: {dfl, hjy, xylee, soumiks}@ iastate.edu

Abstract:
Deep reinforcement learning (DRL) can be used for the development of robotic controllers. Complicated kinematic relationships can be learned by a DRL agent, which will result in a control policy that takes actions based on an observed state. However, a DRL agent typically goes through much trial and error before beginning to take appropriate actions. Therefore, it is often useful to leverage simulated robotic manipulators before performing any training or testing on actual hardware. There are several options for such simulation, ranging from simple kinematic models to more complex models seeking to accurately simulate the effects of gravity, inertia, and friction. The latter models can provide excellent representations of a robotic plant, but typically with a noticeably increased computational expense. Reducing the expense of simulating the robotic plant (while still maintaining a reasonable degree of accuracy) can accelerate an already expensive DRL training loop. In this work, we present a methodology for using a lower-fidelity model (based on Denavit-Hartenberg parameters) to initialize the training of a DRL agent for control of a Sawyer robotic arm. We show that the trained DRL policy can then be fine-tuned in a higher-fidelity simulation provided by the robot's manufacturer. We demonstrate the accuracy of the fully trained policy by transferring it to the actual hardware, demonstrating the power of DRL to learn complicated robotic tasks entirely in simulation. Finally, we benchmark the time required to train a policy using each level of fidelity.

Keywords:
Robotic Systems, Self-learning Models, Real-time Artificial Intelligence, Reinforcement learning control, Engineering Applications of Artificial Intelligence, Optimization and Control, Multi-Fidelity Modeling

1. INTRODUCTION

Deep reinforcement learning (DRL), a class of algorithms typically used for sequential decision-making, has gained popularity in recent years. The rise in popularity of DRL is mainly driven by the representation of the policy using deep neural networks, powerful function approximators that enable the mapping of a state to optimal decisions. As such, DRL has been successfully applied in domains such as design (Yonekura and Hattori, 2019; Mirhoseini et al., 2020) and engineering (Lewis et al., 2012; Lee et al., 2022; Tan et al., 2019). One promising domain where the application of DRL is heavily studied is in control for robotics (Kober et al., 2013; Khan et al., 2020) where there has been a focus on functional and robust (Tan et al., 2020) controllers. In contrast to traditional controllers or hard-coded instructions, a DRL-based controller promises a control policy that is adaptable to environment changes when trained properly. Despite the potential benefits of DRL for control in robotics, it is still a major area of active research due to the bottleneck of poor sample complexity. The training of DRL controllers is notoriously expensive as they often require randomly sampling thousands to millions of trajectories in the environment to learn a useful policy. Hence, training DRL policies on actual robots is often cost-prohibitive. A common approach to training these controllers is to leverage physics simulators, where the simulations can be potentially accelerated and parallelized with modern computing hardware. Nevertheless, the quality of the simulators used poses another challenge as well. A simple, low-fidelity simulation can accelerate the speed of training but may result in an inaccurate control policy when transferred to the actual hardware. On the other hand, using a high-fidelity simulation that properly models the physics and movement of a robotic manipulator may result in a more transferrable policy but is also often computationally expensive, though more feasible than training on actual hardware, as safety considerations and degradation of mechanical parts can be avoided.

This work proposes a multi-fidelity training scheme that fuses the efficiency of one simulation modality and the accuracy of another. Specifically, our contributions are: **1)**We demonstrate that Denavit–Hartenberg parameters can be used as a low-fidelity simulation model to train a DRL control policy in an efficient manner and further fine-tune the policy on a higher-fidelity Gazebo simulation model, **2)**We further show that the fine-tuned policy can then be directly transferred to an actual Sawyer robot without any significant performance degradation, **3)**Finally, we

10.1016/j.ifacol.2022.11.183

evaluate the efficiency of this transfer learning as opposed to initially training in a higher-fidelity environment.

2. RELATED WORKS

DRL has been investigated from multiple angles, from the aspect of algorithmic efficiency such as learning from demonstration (Vecerik et al., 2017; Nair et al., 2018), environment definitions (Antonio Martin H. and de Lope, 2007; Weber and Schmidt, 2021), learning from high-dimensional state spaces (Joshi et al., 2020), and protection from adverserial attacks (Lee et al., 2020). However, training a DRL controller on the actual hardware or any high-fidelity physics simulator is often cost-prohibitive. Hence, the concept of utilizing low-fidelity models for the accelerated development of control approaches has been studied. One example of a low-fidelity model is the Denavit-Hartenberg (DH) model, which has been used to compute the inverse kinematics (Wang et al., 2020; Theofandidis et al., 2018). More closely related are the works of (Li et al., 2020), which uses the DH model as a low-fidelity simulation to train a DRL-controller in multiple stages of increasing difficulty, and the works of (Tang et al., 2019; Di Ianni, 2021), which compared the performance of a DRL controller trained with a DH-model with more traditional controllers. Collectively, these works demonstrated that low-fidelity models are extremely useful as a tool for the rapid development of control strategies.

3. BACKGROUND

3.1 Reinforcement Learning

Reinforcement learning (RL) studies how a machine learning agent learns to react optimally in an environment. A DRL problem is often formalized as a discrete-time stochastic process in which an agent, the decision-making entity, learns through trial and error by interacting with an environment. At each time step t and given environment state s_t, the agent makes an observation o_t and executes an action a_t. The environment then responds to the agent's action by returning a reward r_t, a measure of the quality of the state/action, and transitioning its state to s_{t+1}. Through this repeated interaction with the environment, the agent creates a sequence of actions and observations, also referred to as a trajectory τ $(s_t, a_t, s_{t+1}, a_{t+1}, ...)$. The goal of the DRL agent is to learn an optimal policy $\pi^*(a_t|s_t)$, a rule of the agent that maps states to actions.

Types of DRL Algorithms Two categories of DRL algorithms include value-based methods and policy-based methods.

Value-based methods are algorithms that build and rely on a value function, an intermediate quantity that satisfies the Bellman equation estimating how good a state or a state-action pair is, to subsequently define a policy. The Bellman equation decomposes the value function into the immediate reward and the future discounted reward when following a policy π, such that the value functions could be calculated iteratively through dynamic programming. In value-based DRL, the value function is often parameterized by a neural network θ.

Policy-based methods, on the contrary, do not rely on an intermediate quantity as the value-based methods do. Instead, they directly learn a trajectory-policy mapping that maximizes the expected return $J(\tau) = E[R(\tau)]$. Such algorithms achieve the goal by parameterizing the policy π with neural networks θ and optimized by gradient ascent to find the optimal policy π^*:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|\theta_k$$

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E[R(\tau)], a_t^* \sim \pi_\theta^*$$

Proximal Policy Optimization (PPO) PPO (Schulman et al., 2017) is an actor-critic algorithm designed with improving performance and stability in mind. Instead of using computationally expensive second-order derivatives, the PPO approximates with first-order derivatives with soft constraints such that it could maintain performance in a more stabilized manner. Empirically, PPO has been shown to scale well with continuous action spaces and is a benchmark algorithm readily available for various applications, which is why it is selected for our experiments.

Challenges with DRL Training DRL is computationally expensive. A large number of trials are necessary to properly expose the agent to the environment and enable it to develop an effective policy. In robotics, this expense is amplified further by environments with large observation and action spaces: a manipulator can reach an immense number of target positions. Considering that these degrees of freedom are associated with revolute or prismatic joints that move continuously, this provides an immense number of permutations for describing a robot's state. In addition, some robotic states can be kinematically redundant: a different set of joint angles and prismatic joint positions can cause the end effector to reach the same Cartesian position. All of these factors combined require the agent to take large numbers of actions to explore the robotic environment in order to learn a good control policy.

Applying DRL directly on hardware is often not feasible due to the time required for training, safety concerns about robots taking unsupervised action, mechanical wear, and the physical space required to allow a robot to freely explore its state space. In order to apply DRL, a model of the environment is beneficial. Fortunately, there are excellent simulation tools available for training of accurate models without a need for access to hardware. However, the accuracy of these simulations comes at a cost: since a large number of episodes are required for training, the expense of high-fidelity simulation can create a bottleneck for training.

3.2 Denavit-Hartenberg notation

Robotic manipulators are made of links and joints. Each joint can be described with a reference frame: for revolute joints, the unit vector $\hat{Z}$ of this frame is defined in the direction of rotation, while for prismatic joints, $\hat{Z}$ points in the direction of sliding motion. The unit vector $\hat{X}$ of a reference frame is used to point in the direction of the next frame, which describes the next link in the manipulator. For link i, $\hat{X}_i$ is along the mutual perpendicular line of $\hat{Z}_i$ and $\hat{Z_{i+1}}$, which is assigned to the frame describing

joint $i+1$. With this convention, it is possible to describe the geometric state of a robot with a chain of frames that individually describe the robot's joints.

The geometric relationship between individual frames depends on four parameters: the link length, the link twist, the link offset, and the joint angle. This definition of four parameters for configuration of frames describing a robotic manipulator is called Denavit-Hartenberg (DH) notation (Craig, 2004).

The link length $\hat{a_i}$ describes the distance along $\hat{X}_i$ between frame i and frame $i+1$. This is the length of the perpendicular between $\hat{Z}_i$ and $\hat{Z_{i+1}}$. For a multi-joint robot, this is the shortest distance between sequential joints. The link twist α_i is defined as the angle between $\hat{Z}_i$ and $\hat{Z_{i+1}}$.

The link offset d_{i+1} is the distance from the common perpendicular to the origin of frame $i+1$, measured along $\hat{Z_{i+1}}$. Finally, the joint angle θ_i is the rotation of frame i about $\hat{Z}_i$. This DH notation allows for the creation of a transformation matrix between frames [1]:

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i & a_i c\theta_i \\ s\theta_i & c\alpha_i c\theta_i & -s\alpha_i c\theta_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This transformation matrix describes the rotation and translation between two sequential frames, therefore the transformation matrix between a universal (fixed) frame and the frame attached to the end effector of a manipulator can be found by multiplying transformation matrices together. Assuming a 7-DOF manipulator:

$$ {}^{0}_{7}T = {}^{0}_{1}T\ {}^{1}_{2}T\ {}^{2}_{3}T\ {}^{3}_{4}T\ {}^{4}_{5}T\ {}^{5}_{6}T\ {}^{6}_{7}T $$

Notice that the translation between matrices ${}^{i-1}_{i}P$ can be found from the transformation matrix:

$$ {}^{i-1}_{i}P = \begin{bmatrix} a_i c\theta_i \\ a_i s\theta_i \\ d_i \end{bmatrix} = \begin{bmatrix} {}^{i-1}_{i}T_{14} \\ {}^{i-1}_{i}T_{24} \\ {}^{i-1}_{i}T_{34} \end{bmatrix} $$

For the transformation matrix ${}^{0}_{7}T$ derived previously, the corresponding ${}^{0}_{7}P$ describes the Cartesian position of the end-effector relative to the fixed origin frame.

4. METHODS

We seek to use DRL to develop a policy that will bring Sawyer's end-effector to a specified Cartesian endpoint. Given its relative efficiency and stability, we use the PPO algorithm, implemented with Stable Baselines 3 (OpenAI, 2022b). First, it is necessary to define the environment that the agent will use to develop its policy. This was implemented as an OpenAI Gym environment (OpenAI, 2022a).

4.1 Environment

The robot's state is defined by the angular position of the seven joints, as well as the Cartesian coordinates of the target. Therefore, observations have ten dimensions:

$$ s = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, x_t, y_t, z_t] \quad (1) $$

Both the joint positions and the Cartesian coordinates are normalized to have zero mean and exist within the range [-1, 1]. The Cartesian coordinates are normalized and mapped to the range [0.5, 0.8] in the x, y, and z directions. Normalizing inputs is a common practice in DRL: optimization algorithms often combine features, meaning that features with different scales could be given different and disproportionate weights. Combining unnormalized Cartesian coordinates with normalized joint angles could result in disproportionate weight given to either the joint angles or the target position when developing the policy. The bounds of the joints' range are mapped to the maximum and minimum angles (in radians) that can be commanded for each joint. The DRL agent brings the robot to a neutral position for each episode, and can take actions by manipulating the joint angles. These actions are seven dimensional (for each joint angle) and in the range [-1, 1]. Actions are multiplied by the maximum joint motion allowed per timestep (currently set at 0.02 radians), and added to the existing joint states. Consequently, the agent will move a joint continuously up to 0.02 radians in the positive or negative direction.

Rewards are calculated at each commanded step, and based on negative reinforcement to encourage the agent to move the end effector to the target position as efficiently as possible. This method allows negative rewards to accumulate with increasing magnitude if the end effector spends too many steps at larger distances from the target. Given the Euclidean distance d between the end effector and the target position, the reward function is $r_t = -50d^2$.

Squaring the distance in the reward function allows for stronger discouragement of positions that are nowhere near the target. At each step, a check is performed to ensure that the agent's selected action will not cause the robot's joints to exceed the limits of motion set by the manufacturer. If the commanded action violates these limits, a reward of -5 is given. If the end effector does not reach the target within 100 steps, the episode ends, and a reward of -20 is given. Otherwise, successfully reaching the point within a threshold of one centimeter yields a reward of 200. These values were determined after identifying roughly how much negative reward was accumulated during 100 steps, and finding suitable values that would sway the cumulative episode reward in an appropriate positive or negative direction.

4.2 Sawyer Platform

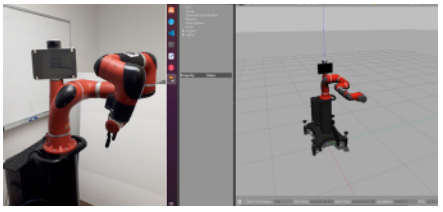

Fig. 1. Sawyer: a 7-DOF robotic manipulator, in hardware and in Gazebo simulation

Sawyer is a robotic manipulator developed by Rethink Robotics, who developed an SDK that allows for com-

[1] For brevity, $\cos\beta$ and $\sin\beta$ can be written as $c\beta$ and $s\beta$, respectively.

manding of a desired angle to all seven joints on the manipulator via a Python interface. This in turn commands the manipulator via the Robotic Operating System (ROS).

4.3 Gazebo: a high-fidelity model

Gazebo is a simulation tool commonly used in robotic simulation. While Gazebo provides an excellent simulation environment, it comes with a severe computational cost. First, it runs in parallel to ROS, hence the backend processes associated with ROS add computational overhead and often are not required for training. Second, the sophisticated physics engine provides realistic simulation, but it adds expense at every timestep when training a DRL model. Even with the physics engine accelerated to the maximum speed supported, this creates a bottleneck for training.

4.4 The Denavit-Hartenberg model: a lower-fidelity model

Others have previously found DH parameters for Sawyer (Theofandidis et al., 2018). These parameters were taken and cross checked with the frame origins for each joint as described in Gazebo:

Table 1. Denavit-Hartenberg parameters for Sawyer

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	0.317	θ_1
2	0.081	$-90°$	0.192501	θ_2
3	0	$90°$	0.4	θ_3
4	0	$-90°$	-0.1683	θ_4
5	0	$90°$	0.4	θ_5
6	0	$-90°$	0.1360	θ_6
7	0	$90°$	0.2701	θ_7

This model (which relies on multiplying the transformation matrices) can then be utilized in the same training loop as Gazebo or the actual Sawyer robot. The DRL agent selects an action, which will either move the joint in a positive or negative direction, or perhaps not at all. In the DH model, this consists of updating the DH table with new joint angles and recalculating the position of the end effector.

4.5 Inaccuracies with Denavit-Hartenberg model

Denavit-Hartenberg parameters are susceptible to errors: even with slight deviation from the true values, it is possible for error to accumulate in the forward kinematic solution (Roth et al., 1987). To demonstrate this, 100 sets of joint angles were randomly generated. The DH model was then utilized to calculate the end-effector position associated with these joint angles. The difference in x, y, and z end-effector position between the DH model and Gazebo was uniformly distributed around zero, but with errors ranging up to 0.13 meters. Due to these errors, the DH model can be utilized as a low-fidelity simulation, but Gazebo is better suited for tuning of control strategies that will transfer to the actual hardware.

4.6 Experiments

The Sawyer manipulator is intended for automating manufacturing processes. With a 7-DOF manipulator, this opens up an immense range of possible tasks. In this paper, we simplified the task to moving the end-effector to a target position as a proof of concept. Five Cartesian points were selected at random and are used for experiments. The x, y, and z coordinates were sampled from the range [0.5, 0.8]. This range for each dimension creates a box in Cartesian space that is well within Sawyer's action space.

5. RESULTS

5.1 Sawyer hardware: training results

The physical Sawyer robot was utilized to train for 250,000 timesteps. The final results were suggestive of a successful policy: rewards converged just below 200, indicating successful manipulation of the hardware to reach the target. However, this was extremely expensive, with trainings for the full number of timesteps lasting up to 13 hours. Owing to this expense, only three points were trained on the hardware.

5.2 Gazebo alone: training results

To establish a baseline for simulated trainings, all points were trained entirely in Gazebo for 500,000 timesteps. The agent learned well: strong convergence on high rewards was observed as the simulated Sawyer robot was able to reach the target position consistently. Once again, the training was computationally expensive, with trainings for the full number of timesteps taking up to seven hours.

Gazebo simulations are typically run in real time to match the behavior of the hardware. They can be accelerated to the fastest speed allowed by the ROS Physics Engine. This acceleration was performed for all Gazebo trainings in this paper, yet the final simulation was only accelerated to three or four times faster than real time. This presents a significant bottleneck to those attempting to deploy DRL agents on Sawyer: prototyping of new reward functions, tweaking of observation spaces, and tuning of hyperparameters require hours of training before results become apparent. Therefore, we seek to understand whether a DH model can bridge the gap between high-fidelity simulation and efficient DRL training.

5.3 Denavit-Hartenberg model: training results

Training on the DH model was originally performed for 500,000 timesteps, with runtimes averaging around 1400 seconds. Training was then limited to 200,000 timesteps for the DH model. This was done in an attempt to accelerate the training process that would be completed in Gazebo. Furthermore, given that the DH model comes with its own set of inaccuracies, it does not make sense to train a highly refined policy on a flawed simulation before performing a transfer: there is a risk of the agent creating an overfitted policy and struggling to correct it in the higher-fidelity simulation.

5.4 DH Model + Gazebo: training results

After the policy had been trained for 200,000 timesteps, it was transfered to Gazebo. OpenAI Gym allows for

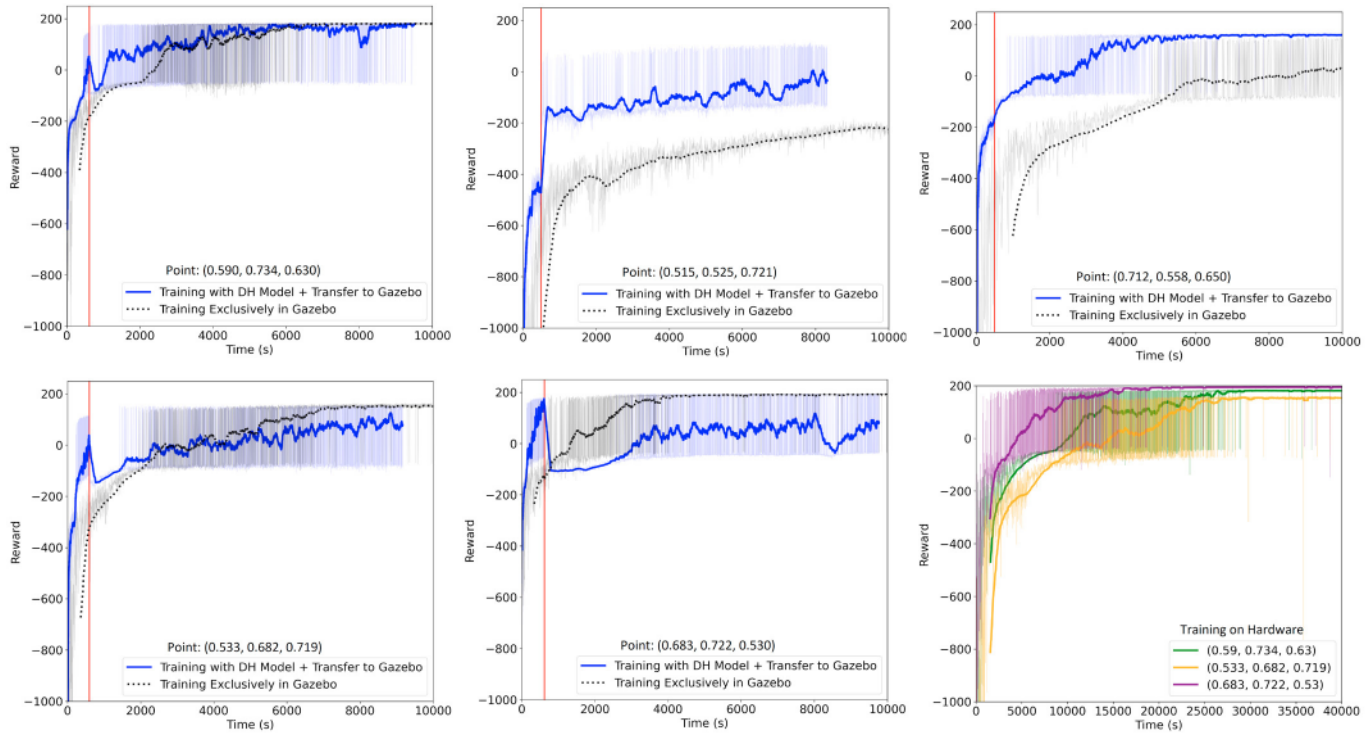

Fig. 2. Completion of training in Gazebo for four of the five targets. Time where policy was transferred to Gazebo is depicted with a red line. Bottom right: hardware trainings offered for runtime comparison.

saving and loading of trained models, so the training was continued for another 250,000 timesteps. Since this policy (represented by a neural network) takes as inputs the robot joint angles and normalized Cartesian coordinates of the target, this can be done just as easily with Gazebo. What will change after the transfer is the policy that has previously been learned: instead of being suited to guide the simulated end effector of the DH model, it will now learn on the more accurate Gazebo model. The neural network weights will update to do a better job of mapping the current robot state to desirable actions. We now seek to understand the efficiency of this transfer learning method when compared to training exclusively in Gazebo from the first timestep. To make this comparison, Figure 2 shows reward achieved with respect to time.

Transferring a partially trained model is considerably more efficient for some points. This is evident for point (0.515, 0.525, 0.721) and (0.712, 0.558, and 0.650) in Figure 2: the more efficient DH model allows us to accelerate the initial stages of training where the agent has little indication of the best actions to take. Note that training with Gazebo did not result in a convergence after almost three hours for point (0.515, 0.525, 0.721) in Figure 2, yet the transferred policy from the DH model was able to converge after slightly more than one hour.

Other trainings fared less well. The curve for point (0.683, 0.722, 0.530) in Figure 2 shows the DH model reaching higher rewards almost instantly, but the policy developed after roughly 1000 seconds struggled to continue training in Gazebo. Given the imperfections of the DH model, the agent learns a policy based on imperfect information. Therefore, the same joint motions that bring the end effector to the target do not achieve the same in Gazebo. This is expected, but the time that it takes to use the existing policy as a baseline and learn a better policy better suited for Gazebo varies from point to point.

Finally, some trainings performed approximately the same as if they had run on Gazebo in the first place. Points (0.590, 0.734, 0.630) and (0.533, 0.682, 0.719) in Figure 2 show the different training strategies as having roughly the same performance: both start to converge upon high rewards that indicate successfully reaching the target, yet one does not clearly outpace the other.

These results show that the training loop involving the transfer of a policy from the DH model to Gazebo can provide a time saving advantage in certain cases. For this advantage to extend to all cases, improvements to the DH model likely need to be made. The error between the DH model and Gazebo needs to be reduced as much as possible. Errors in the link lengths or link twists of the manipulator eventually translate into a larger error that needs to be overcome by the agent when training on higher-fidelity simulation.

5.5 Transferring Refined Policy (DH+Gazebo) to Hardware

Given that the DH model has flaws and Gazebo is still a simulation, it is necessary to verify that the policy at the end of the DH+Gazebo training process transfers to hardware. The trained policy for each point was saved and loaded on Sawyer. The policy then provided actions for the robot to take based on the current state of the seven joints and the target position. This process was repeated three times, and the distance between the target position and the end effector (as defined in Gazebo) was calculated.

Generally, the policy that was refined with the higher-fidelity Gazebo simulation performed well: in ten of fifteen cases, it successfully reach the target position within a one centimeter threshold. However, for one trial the policy was unsuccessful at reaching the target and experienced a 13 centimeter error. This highlights the sensitivity of the policy and the need for extended training or improvement to the DRL algorithm to consistently reach the target.

6. CONCLUSION & FUTURE WORKS

We demonstrate that a model of kinematic motion for Sawyer using Denavit-Hartenberg parameters is plausible and allows for very fast training of a DRL policy. Despite this, limitations still exist in terms of the accuracy of the DH model, hence the need for higher-fidelity simulation in Gazebo. To accelerate training, it is possible to fuse these two modalities: the agent learns a nascent policy from the efficient DH model and refines it in the more accurate Gazebo model, with generally strong performance when transferred to hardware. Further work is needed to improve the DH model and reduce the error that the DRL agent needs to overcome when transferring between modalities. It is possible that a different reward function will allow for more efficient training. Furthermore, it is also necessary to evaluate how a policy could be learned that guides the robot arm to multiple points. While not the main focus of this work, such a policy could expand the use cases of the controller that is developed. Regardless, these results show that a DH model can be used as a tool for prototyping and efficient performance evaluation of a novel control strategy for a robotic manipulator.

REFERENCES

Antonio Martin H., J. and de Lope, J. (2007). A distributed reinforcement learning control architecture for multi-link robots.

Craig, J.J. (2004). *Introduction to Robotics: Mechanics and Control*, volume 3. Pearson.

Di Ianni, L. (2021). *Control System Design with Reinforcement Learning Algorithm for a Space Manipulator*. Ph.D. thesis, Politecnico di Torino.

Joshi, S., Kumra, S., and Sahin, F. (2020). Robotic grasping using deep reinforcement learning. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 1461–1466. IEEE.

Khan, M.A.M., Khan, M.R.J., Tooshil, A., Sikder, N., Mahmud, M.P., Kouzani, A.Z., and Nahid, A.A. (2020). A systematic review on reinforcement learning-based robotics within the last decade. *IEEE Access*, 8, 176598–176623.

Kober, J., Bagnell, J.A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11), 1238–1274.

Lee, X.Y., Ghadai, S., Tan, K.L., Hegde, C., and Sarkar, S. (2020). Spatiotemporally constrained action space attacks on deep reinforcement learning agents. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Lee, X.Y., Sarkar, S., and Wang, Y. (2022). A graph policy network approach for volt-var control in power distribution systems. *Applied Energy*.

Lewis, F.L., Vrabie, D., and Vamvoudakis, K.G. (2012). Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE Control Systems Magazine*, 32(6), 76–105.

Li, Z., Ma, H., Ding, Y., Wang, C., and Jin, Y. (2020). Motion planning of six-dof arm robot based on improved ddpg algorithm. In *2020 39th Chinese Control Conference (CCC)*, 3954–3959. doi:10.23919/CCC50068.2020.9188521.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E., Wang, S., Lee, Y.J., Johnson, E., Pathak, O., Bae, S., et al. (2020). Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, 6292–6299. IEEE.

OpenAI (2022a). Gym. URL `https://gym.openai.com/`.

OpenAI (2022b). Stable baselines 3. URL `https://stable-baselines3.readthedocs.io/en/master`.

Roth, Z., Mooring, B., and Ravani, B. (1987). An overview of robotic calibration. *IEEE Journal of Robotics and Automation*.

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

Tan, K.L., Esfandiari, Y., Lee, X.Y., and Sarkar, S. (2020). Robustifying reinforcement learning agents via action space adversarial training. *American Control Conference*.

Tan, K.L., Poddar, S., Sarkar, S., and Sharma, A. (2019). Deep reinforcement learning for adaptive traffic signal control. *Dynamic Systems and Control Conference*.

Tang, M., Yue, X., Zuo, Z., Huang, X., Liu, Y., and Qi, N. (2019). Coordinated motion planning of dual-arm space robot with deep reinforcement learning. In *2019 IEEE International Conference on Unmanned Systems (ICUS)*, 469–473. doi:10.1109/ICUS48101.2019.8996069.

Theofandidis, M., Sayed, S.I., Cloud, J., Brady, J., and Makedon, F. (2018). Kinematic estimation with neural networks for robotic manipulators. *Artificial Neural Networks and Machine Learning – ICANN 2018*.

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*.

Wang, X., Cao, J., Chen, L., and Hu, H. (2020). The optimized algorithm based on machine learning for inverse kinematics of two painting robots with non-spherical wrist. *PloS one*, 15(4), e0230790.

Weber, J. and Schmidt, M. (2021). An improved approach for inverse kinematics and motion planning of an industrial robot manipulator with reinforcement learning. In *2021 Fifth IEEE International Conference on Robotic Computing (IRC)*, 10–17. doi:10.1109/IRC52146.2021.00009.

Yonekura, K. and Hattori, H. (2019). Framework for design optimization using deep reinforcement learning. *Structural and Multidisciplinary Optimization*, 60(4), 1709–1713.