Learning to Control End Effector Poses with PPO and SAC

Haomiao Zhang
Department of Mechanical Engineering
The University of Brisith Columbia
Vancouver, Canada
haomiaoz@student.ubc.ca

***Abstract*—End effector pose control is crucial in the industries for various tasks. To control end effector poses, inverse kinematics usually would be applied to decompose poses to joint positions for joint level control. While inverse kinematics is effective when the joint dimension low, it becomes difficult and time consuming with the increase of joint dimension. Fortunately, deep reinforcement learning (DRL) has demonstrated its capability in performing motion control with high joint dimension. In this paper, controllers are learned using PPO and SAC to control the end effector poses throuhgh inverse kinematics and joint position control. Training and evaluation are performed on a IIWA robot arm in PyBullet simulation. With the given training time and hyperparameters, the DRL controllers perform statistically better than PD controller when reaching the same target, but the performance decreases when trying to reach different targets within the work envelope.**

***Index Terms*—robotics, deep reinforcement learning, motion control, inverse kinematics**

I. Introduction

Robotic arms are widely used in the manufacturing industry to perform various tasks such as pick and place, assembly, welding and painting [1]. Most of the tasks would require precise pose (position and orientation) control of end effectors.To generate precise poses, robotic arms are typically controlled through classical approaches after inverse kinematics and dynamics with additional task dependent constraints or optimizations.

Deep reinforcement learning (DRL) has been a growing interest among the robotics community due to its ability to solve problems that are difficult to be solved through classical approaches. Some of those problems are robot manipulation [2], quadruped robot motion control [3], aerial robot motion control [4] and soft robot motion control [5]. Those problems are difficult to be solved through classical approaches commonly due to high degrees of freedom (DOF), complex environment or nonlinear dynamics. However, for problems that have been solved through classical approaches such as the control of end effector poses, DRL is not a commonly applied approach.

In this paper, two different DRL approaches are taken to design robot end effector pose controllers.The controllers take in the state of the robot arm and the target pose of the end effector to generate target joint positions for joint level control. A 7 DOF KUKA LBR IIWA 14 R820 (IIWA) robot is modeled in PyBullet simulation for training and evaluation. The trained controllers are compared with a PD position controller as the baseline. DRL controllers performed better than the PD controller in some environment but worse on the other. The reasons of the performance differences are discussed, which leads to future directions for improvement.

Project webpage:
https://haomiaoz.github.io/CPSC533V-Project-IIWA-RL-Control/

II. Background

A. Classical approach of Robot Position Control

Inverse kinematics is the method of obtaining the joint positions based on the specified position and orientation at the end of the kinematic chain given the configuration of the chain [6]. The difficulties of the closed form inverse kinematic solution depend on the configuration of the robot, including the robot DOF. With 6 DOF robotic arms, a close form solution could typically be obtained. [6]. With the advancement of human-robot interaction, anthropomorphic robotic arms emerge. Those arms mimic the configuration of the human arm, and thus have 7 DOF [7]. Even with the specification of both orientation and position of end effector, there is 1 DOF redundancy left for optimization or task augmentation. Closed form solutions are not available without extra constraints. Generated solutions depend on the approaches and could become more complicated with the increase of DOF, which is difficult to generalize.

B. Reinforcement Learning in Robot Position Control

Robotic arm control has been researched with DRL mostly for manipulation. In robot manipulation study like [2], the robot end effector needs to be moved to a specific pose to manipulate objects. There is some implicit coding of inverse kinematics in the model; however, the pose problem is not explicitly solved, since the main target usually is to successfully grasp the object rather than grasp the object with a specific pose. There are environments has low fidelity robot arm simulation such as reacher in DeepMind Control Suite [8], The lack of fidelity in the model would be difficult to transfer the learned controller policy safely onto a real robot. [9] trained a IIWA robot to investigate the sim to real transfer with 4 DOF specified by the target pose. [10] utilized a DQN network to control a Baxter arm while the end

effector position is limited in a plane. [11] trained a controller to reach different positions in the 3D space. However, no orientation constraint is applied, and the robotic arm is 6 DOF.

To the best of the author's knowledge, there is no study that explicitly control a 7 DOF robotic arm reaching different 6 DOF poses in high-fidelity simulation.

III. Control Algorithms

A. PD control

PD controller is a traditional approach to control the position of the robot. Based on the target position q_t and current position q_c , the error and the derivative of the error between the target and current position are used to formulate the controller in a form of:

$$u = K_p e + K_d \dot{e} \tag{1}$$

where

$$e = q_t - q_c \tag{2}$$

K_p and K_d are the proportional and derivative gain of the PD controller.

From the target end effector poses, inverse kinematics needs to be applied to obtain target joint positions. Then joint level PD controllers could be used to control the end effector to reach the target poses.

B. Proximal Policy Optimization (PPO)

PPO is an on-policy reinforcement learning algorithm for continuous action spaces. Is it a type of actor critic algorithm based on policy gradient. The idea is similar to Trust Region Policy Optimization (TRPO), which has a clipped objective during update to improve stability, with simpler implementation than TRPO [12].The main objective of PPO is:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{3}$$

$r_t(\theta)$ is the ratio between the current policy and the old policy, parameterized by θ, and $\hat{A}_t$ is the estimation of advantage function at time t [12].

PPO is selected since it is a popular choice used for robotics reinforcement learning [13]. The implementation based on the course assignment is first utilized. However, later the implementation from Stable-Baseline3 [14] is used to provide better training efficiency and result.

C. Soft Actor Critic(SAC)

SAC is an off-policy reinforcement learning algorithm for continuous action spaces. It is also a type of actor critic algorithm, although the critic is a action-value (Q) function rather than value (V) function [15]. SAC modify the typical Q function with an additional entropy term as follow:

$$Q(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}[V(\mathbf{s}_{t+1})] \tag{4}$$

where

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi}[Q(\mathbf{s}_t, \mathbf{a}_t) - log\pi(\mathbf{a}_t|\mathbf{s}_t)] \tag{5}$$

SAC is selected due to suggestions from the course staff. Additionally, it is also a popular algorithm applied to robotic arm control [13]. The implementation of SAC from Stable-Baseline3 [14] is used for better training efficiency and result.

IV. Simulation Environment

IIWA robot is selected as the robot arm for simulation due to IIWA's wide application in different tasks for different purposes. Also, the author has access to a physical IIWA robot for future development such as sim2real transfer. The physics engine is initially decided between PyBullet [16] and MuJoCo [17]. PyBullet is eventually selected since IIWA has been simulated in pybullet beforehand [13], and it is easier to calculate the kinematics and inverse kinematics between links for evaluation and reward formulation compared with MuJoCo. The joint limits of IIWA are considered in modeling and listed in Table I. However, other limitations such velocity and torque limitations are not considered since those are not implemented in the control method in PyBullet.The information of joint position, velocity and torque limits are based on the values in [18]. The step frequency of the simulated environment is set to 30Hz, which is sufficient for position control with low to mid range velocities.

TABLE I: Joint Limits of IIWA

Joint	Joint Limits [deg]
q_1	± 170
q_2	± 120
q_3	± 170
q_4	± 120
q_5	± 170
q_6	± 120
q_7	± 175

A. Target Type

The base of the robot is set to the origin of the simulated environment. Three target types are considered when constructing the simulated environment:

- *Point*: the target pose is fixed for each episode.
- *Box*: the pose is randomly selected within the box region given the selected pose is reachable for each episode.
- *Random*: Any reachable pose is randomly selected for each episode.

The reachability is validated through inverse kinematics when selecting target poses.

For the point target, the position P_p and orientation P_o selected are:

$$P_p = [0.2, 0.3, 0.7], P_o = [0, 0, 0, 1] \tag{6}$$

The position is in Cartesian coordinate, and the orientation is represented with quaternion. The goal pose has been

visualized as a green capsule shape in the simulation in Fig. 1a. The pose is selected to avoid IIWA being close to the joint limits. This target configuration is the simplest environment to be solved.

For box shape, the box is defined as:

$$Box = [0.6 \pm 0.075, 0 \pm 0.2, 0.4 \pm 0.2] \tag{7}$$

The box is selected to maximize the size of the box in the work envelope of the robot. The robot work envelope information is obtained from [18]. Valid pose is randomly selected within the box for each episode. This configuration is similar to [11] setting, with additional constraints on the orientation. This target configuration is considered the intermediate step toward the most difficult configuration.

The random target type would select any valid target pose in the workspace. It is considered the most difficult configuration.

Three different target types are shown in Fig. 1. The green capsule shape visual marker visualizes the target pose of the episode, and the red visual marker represents the target pose selection bounds.

B. State and Action

The state of the environment includes the state of the robot, which are the joint positions ($q_c \in R^7$) and velocities ($\dot{q}_c \in R^7$). Other than the state of the robot arm, the target pose is also included as part of the state of the environment. The target position is represented in Cartesian coordinate ($P_p \in R^3$), while the orientation is represented as quaternion ($P_o \in R^4$). Using quaternion is beneficial since it prevents gimbal lock, and the same orientation will only have one representation. The state vector therefore is:

$$s = \begin{bmatrix} q_c & \dot{q}_c & P_p & P_o \end{bmatrix} \in R^{21} \tag{8}$$

The action are the target joint positions subjected to joint limits shown in table [insert table]. The action vector therefore is:

$$a = q \in R^7 \tag{9}$$

State and action are normalized to [-1,1] to provide equal flow of gradient during training.

C. Reward

The reward is selected to be a dense reward to speed up the training process. At each step, the difference between the target end effector pose ($[P_{p_t} \quad P_{o_t}]$) and the current end effector pose ($[P_{p_c} \quad P_{o_c}]$)is penalized to encourage the robot to move toward the target pose. Position error and orientation error are separated when calculating the Euclidean norm, and they are summed up as the reward (cost). When the pose error is smaller than 0.1 in total, a reward of 500 will be given to encourage the behaviour of reaching the specified pose. The threshold of 0.1 is selected since the maximum end effector position error is intended to be within 10 cm of the target end effector position when the orientation of the end effector is perfectly matched with the target. The reward function is:

$$r = \begin{cases} -e & if \quad e \geq 0.1 \\ 500 & if \quad e < 0.1 \end{cases} \tag{10}$$

where

$$e = \|P_{p_t} - P_{p_c}\| + \|P_{o_t} - P_{o_c}\| \tag{11}$$

D. Starting and Terminating Conditions

Initially, the robot arm would start at the position where each joint is randomly initialized between -5 °to 5 °. The episode then would start to step. There are two terminate conditions:

1) The pose error e is smaller than 0.1, which indicates that the end effector reaches the target. The episode terminates immediately.
2) The total step is 300, which corresponds to 10s of movement with 30Hz stepping frequency. Generally the end effector should be able to reach the target within 10 s. If not, then the robot is unlikely to reach the target, and the episode should be terminated.

V. Training Process

PD controller does not need training. For both PPO and SAC, the implementations from Stable-baseline3 [14] are utilized. The training is perform with an Intel i7 8th Gen CPU on a laptop. For both PPO and SAC, the actor and critic are operated on separate neural networks. The structure of the networks are the same: 2 hidden layers with 256 units on each layer. The activation function is ReLU on all layers. PPO is trained without parallelization, so the batch size equals the max episode length. Other hyperparameters used for PPO and SAC are the default hyperparameters implemented by Stable-baseline3.

VI. Experiments and Results

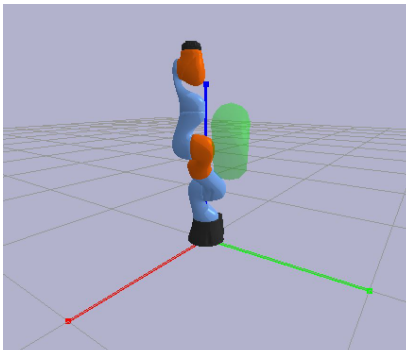
For *Point* environment, both PPO and SAC are trained for about 1 hour wall time each to achieve desired results. PPO is trained for about 4000 epochs, while SAC is trained for about 400 epochs.

For *Box* environment, both PPO and SAC are trained for about 16 hours wall time each to show some promising behaviours. PPO is trained for about 80000 epochs, while SAC is trained for about 8000 epochs.

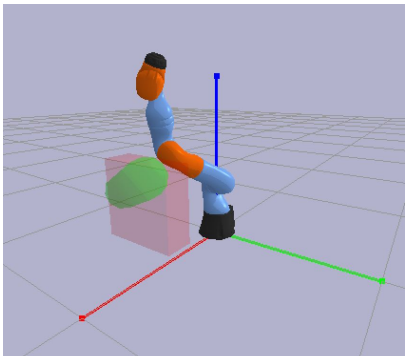
For *Random* environment, both PPO and SAC are trained for about 4 hours wall time each to show some promising behaviours. PPO is trained for about 20000 epochs, while SAC is trained for about 2000 epochs.

For testing, 1000 episodes are run for each type of control algorithm with all three types of the target. The average reward and episode lengths are calulated and shown in Table II and III.

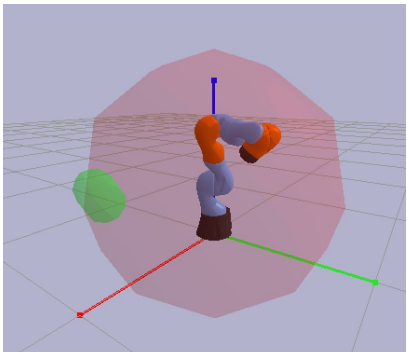
For *Point* environment, all three control algorithms are able to solve the environment. Also, PPO and SAC are statistically significantly better than PD since both algorithms at least have



(a) Point Target Type



(b) Box Target Type



(c) Random Target Type

Fig. 1: Simulation Environment with Different Target Type

TABLE II: Average Reward of 1000 Episodes

Controller	Target Type		
	Point	***Box***	***Random***
PD	492.0 ± 0.3	466.3 ± 73.7	238.8 ± 308.1
PPO	493.3 ± 0.2	-378.0 ± 103.5	-491.5 ± 103.6
SAC	492.5 ± 0.2	-288 ± 100.0	-404.9 ± 140.9

TABLE III: Average Episode Length of 1000 Episodes

Controller	Target Type		
	Point	***Box***	***Random***
PD	23.7 ± 0.5	42.2 ± 38.5	138.7 ± 126.9
PPO	21.1 ± 0.3	300.0 ± 0.0	300 ± 0.0
SAC	20.0 ± 0.1	300 ± 0.0	300 ± 0.0

statistically significant higher reward or shorter episode length compared with PD. With sufficient training, PPO and SAC perform better than PD in *Point* environment.

For *Box* environment, PD could solve the environment almost every time. which yields a positive reward on average. Reaching the target in *Box* environment takes longer than the point environment. Therefore, the episode length is larger. The reward is smaller than *Point* environment due to the increase of total penalization with longer episode length. For PPO and SAC, the end effector does move toward the target pose but never reach the target pose. Therefore, the episode length is always 300.

For *Random* environment, PD could still solve more than half of the environment without trajectory generation, although the average reward drops significantly. In *Random* environment, PPO and SAC have similar performance as *Box* environment. The end effector does move toward the target pose, but never reaches it.

VII. Discussion and Insights Gained

The performance of PPO and SAC in *Point* environment demonstrate the capability of the algorithms. However they are not performed as well as PD in *Box* and *Random* environment. There are several reasons that might cause the drop in performance.

- **Insufficient Training:** Complex environments need more time in training for exploration to reach the optimal policy that can solve the environment. *Box* and *Random* environment are significantly more difficult than *Point* environment. Therefore more training time is expected. There is a possibility that the training time is insufficient that the optimal policy has not been found yet, given non of the episodes reach the target with the given pose error threshold. Training more might improve the performance of PPO and SAC in *Box* and *Random* environment.
- **Tight Reaching Threshold:** Tight reaching threshold could increase the difficulties of the end effector reaching the target pose. Without the final reward, the optimal policy that could be learned from actions is to reduce the penalization at each step. Therefore, only the motion of moving toward the target pose is learned, but not reaching the target pose itself. Loosen the threshold might improve the performance of PPO and SAC in *Box* and *Random* environment.
- **Untuned Reward Function:** The reward function only contain the term of penalization of current pose and target pose with separation of position and orientation. One observation during testing is that the algorithms tend to minimize the orientation error first. The reason might be that when the end effector position is within proximity of the target position, orientation error is the major factor in penalization. Therefore, the algorithms learned to prioritize the matching of orientation over position when the end effector is close to target position. Adjusting the factors in the reward function might improve the performance of PPO and SAC in *Box* and *Random* environment.
- **Untuned Hyperparameters:** Stable-baseline3 default hyperparameters are used for both PPO and SAC. Non-default hyperparameters might suit better for *Box* and *Random* environment. Tuning the hyperparameters might improve the performance of PPO and SAC in *Box* and *Random* environment.

Through this project, I learned a lot about implementing a simulation environment and training DRL algorithms on it. To

name a few lessons learned:

- Start with a simple environment. Especially when the ultimate goal is a complex environment, start with simple environment to train for parameter tuning and debugging.
- Making DRL algorithms work needs a lot of tuning. One might need to tune reward function, termination conditions, stepping frequency, hyperparameters and so forth to make algorithms work.
- Look for existing implementations of algorithms and environments. It is likely that someone else has a better implementation of the algorithms or environments. Take a look before implement it by yourself.
- Get a GPU or cloud computing resource for training. Training on CPU with a laptop is not fun when the environment become complex.
- Have a better understanding of metrics such as KL and entropy. With better understanding of those metrics, I might be able to tune the hyperparameters in the direction of improving the performance.
- Have checkpoints to save models during training in case the program crashes after a few hours, or more training is needed after that round.
- It takes some time to figure out the simulation software and how to write a customized environment

VIII. Future Work

Here is a list of future work that I intend to do when time permits:

- **Tune Environment:**
  1) Adjust the reward function to include more terms, or adjust the ratio between position error and orientation error.
  2) Loosen the pose threshold to simplify the environment
  3) Use curriculum learning to learn the workspace starting from simpler target poses.
  4) Create target type with only 3D positional constraints to simplify the environment, which has been done in [11].
  5) Use 6 DOF robot arm to simplify inverse kinematics that need to be modeled.
- **Tune Algorithms:**
  1) More training (with GPU)
  2) Tune hyperparameters
- **Other:**
  1) Train a joint torque controller with DRL
  2) Apply the trained model to the physical robot

IX. Conclusion

Two DRL approaches, PPO and SAC, are investigated to create joint controllers to perform inverse kinematics and position control. An IIWA robot is simulated in PyBullet environment for training and evaluation. While both DRL algorithms perform better in *Point* environment, they cannot fully solve the *Box* and *Random* environment with the given training time and hyperparameters. With more training and tuning, DRL could have improved performance and learn inverse kinematics for all end effector poses in the work envelope.

X. Acknowledgment

I really appreciate all the help from the instructor and the course staff. Thank you for the feedback on the proposal. Thanks for suggestions on using existing implementation SB3, SAC, normalization of the observation and action, and network architectures. Thank you for all the help throughout the course.

References

[1] B. Siciliano, O. Khatib, M. Hägele, K. Nilsson, R. Bischoff, and J. N. Pires, "Ch 54. Industrial Robotics," in Springer Handbook of Robotics, Springer International Publishing, 2016, pp. 1385–1422.

[2] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017.

[3] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for Legged Robots," Science Robotics, vol. 4, no. 26, 2019.

[4] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous Drone Racing with Deep Reinforcement Learning," 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021.

[5] S. Bhagat, H. Banerjee, Z. Ho Tse, and H. Ren, "Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges," Robotics, vol. 8, no. 1, p. 4, 2019.

[6] B. Siciliano, O. Khatib, K. J. Waldron, and J. Schmiedeler, "Ch2. Kinematics," in Springer Handbook of Robotics, Berlin: Springer, 2016, pp. 11–36.

[7] L. Luthsamy, H.F. AL-Qrimli, S.Shazzana Wan Taha, and N.A. Raj, "Design and control of an anthropomorphic robotic arm," Journal of Industrial Engineering Research, vol. 2, no. 1, pp.1–8, 2016.

[8] Y. Tassa et al., "DeepMind Control Suite", CoRR, vol abs/1801.00690, 2018.

[9] M. Kaspar, J. D. Munoz Osorio, and J. Bock, "Sim2Real transfer for Reinforcement Learning Without Dynamics Randomization," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.

[10] F. Zhang, J. Leitner, M. Milford, B. Upcroft, en P. I. Corke, "Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control", CoRR, vol abs/1511.03791, 2015.

[11] A. Franceschetti, E. Tosello, N. Castaman, S. Ghidoni. Robotic Arm Control and Task Training through Deep Reinforcement Learning. The 16th International Conference on Intelligent Autonomous Systems (IAS-16); Proceedings of. 2021.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, en O. Klimov, "Proximal Policy Optimization Algorithms", CoRR, vol abs/1707.06347, 2017.

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

[14] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, en N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations", Journal of Machine Learning Research, vol 22, no 268, bll 1–8, 2021.

[15] T. Haarnoja, A. Zhou, P. Abbeel, en S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", CoRR, vol abs/1801.01290, 2018.

[16] E. Coumans en Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning", 2016–2021. [Online]. Available at: http://pybullet.org.

[17] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.

[18] “LBR iiwa 7 R800, LBR iiwa 14 R820 Specification,” LBR iiwa - Spez_LBR_iiwa_en.pdf. [Online]. Available: https://www.oir.caltech.edu/twiki_oir/pub/Palomar/ZTF/KUKARoboticArmMaterial/Spez_LBR_iiwa_en.pdf. [Accessed: 12-Dec-2021].