Leveraging Deep Reinforcement Learning for Reaching Robotic Tasks

Kapil. Katyal   I-Jeng Wang   Philippe Burlina
Johns Hopkins University Applied Physics Laboratory and Dept. of Computer Science

Abstract

This work leverages Deep Reinforcement Learning (DRL) to make robotic control immune to changes in the robot manipulator or the environment and to perform reaching, collision avoidance and grasping without explicit, prior and fine knowledge of the human arm structure and kinematics, without careful hand-eye calibration, solely based on visual/retinal input, and in ways that are robust to environmental changes. We learn a manipulation policy which we show takes the first steps towards generalizing to changes in the environment and can scale to new manipulators. Experiments are aimed at a) comparing different DCNN network architectures b) assessing the reward prediction for two manipulators with varying kinematics and c) performing a sensitivity analysis comparing a classical visual servoing formulation of the reaching task with the proposed DRL method.

1. Introduction

There have been several efforts using reinforcement learning (RL) and DRL (since the seminal work by Google DeepMind in DRL [4]) in the context of robotic navigation or manipulation tasks [1, 3, 5, 8]. Reaching and grasping manipulation have been less studied using DRL [7]. This is a unique contribution made by this study along with these: we develop a general approach that is agnostic and can adapt to new manipulators via DRL. We demonstrate robust and direct mapping from image to action domain which exploits simulation for learning. We compare different network architectures and show that the proposed control policy addresses robotic control that adapts to environmental changes and does so robustly when compared to a classical approach.

2. Method

Conventional RL can be formalized as a Markov Decision Process (MDP) which consists of an *agent* interacting with the *environment*. The agent selects *actions* that result in a *reward* causing a change to the *environment* as observed as the next *state*. Q-Learning is a form of RL using a policy where the *agent* selects an *action* based on current *state* that maximizes expected *reward*. The Q-value correlates to the quality of choosing the action given the state and is iteratively updated. With Q-Learning, typically the assumption of a compact system state representation is made for computational efficiency. For manipulation, the state of the system could be modeled as the Denavit-Hartenberg (DH) parameters defining the robot kinematics, the current joint angles and the location of the target object. But doing so would make the state application specific and would not apply to other manipulators. A preferred route is to model the state using raw image pixels input. This would allow maximum generalization but lead to curse of dimensionality: assuming a state represented by 4 frames of an 84×84 pixel grayscale image, the state space is $\approx 10^{67970}$ dimensions. This motivates using DRL: Instead of using a table to represent Q-values, a deep neural network (DNN) is used to map raw image pixels to Q-values corresponding to potential actions. Our architecture consists of the DRL framework and the robot interface including a module able to effect manipulator joint angles discrete changes (+/- 5 degrees to 3 joints angles) and compute the Euclidean distance between the endpoint and the closest target object. The DRL Framework inputs raw image pixels from the robotics simulator and outputs one of six discrete joint biasing actions. It is only aware of the number of actions available and is making decisions solely based on the reward. We use and compare two different CNN architectures (Table 1). During training, the goal is to learn a Q-value for each action corresponding to the current state that represents a best estimate of the quality of selecting the given action. An ϵ-greedy approach is used for random exploration with $\epsilon = 0.1$. A discount reward factor, γ is set to 0.95. During the learning

Table 1. DRL DCNN Network Architectures

SMALLER NETWORK ARCHITECTURE						
Layer	**Input**	**Kernel Size**	**Stride**	**Num Filters**	**Activation**	**Output**
conv1	$4 \times 84 \times 84$	8×8	4	32	ReLU	$20 \times 20 \times 32$
conv2	$20 \times 20 \times 32$	4×4	2	64	ReLU	$9 \times 9 \times 64$
fc3	$9 \times 9 \times 64$	N/A	N/A	256	ReLU	256
fc4	256	N/A	N/A	18	ReLU	18

LARGER NETWORK ARCHITECTURE						
Layer	**Input**	**Kernel Size**	**Stride**	**Num Filters**	**Activation**	**Output**
conv1	$4 \times 84 \times 84$	8×8	4	32	ReLU	$20 \times 20 \times 32$
conv2	$20 \times 20 \times 32$	4×4	2	64	ReLU	$9 \times 9 \times 64$
conv3	$9 \times 9 \times 64$	3×3	1	64	ReLU	$7 \times 7 \times 64$
fc4	$7 \times 7 \times 64$	N/A	N/A	512	ReLU	512
fc5	512	N/A	N/A	18	ReLU	18

phase, the loss function is used to update the weights of the DQN network after every episode consisting of 75 training steps. The reward function is based on the Euclidean distance between the end effector position and the target object. If the distance decreases since the last timestep, the reward is incremented by one. Conversely, if the distance increases since the previous timestep, the reward is decremented by one. The DRL Framework and the corresponding neural network approximation of the Q-Value function are implemented using Caffe [2].

The simulator used to demonstrate trajectory planning is based on the open source Robotics Library simulator [6]. The simulator also allows capturing of image pixels of the screen and provides the ability to detect distances to other objects in the scene. It was modified to take as input incoming commands to control the robot joints and to output the screen pixels and the Euclidean distance over a network interface.

3. Experiments

Our first goal was to assess the ability to perform learning for disparate manipulators. We then turned our attention to characterize the resulting error for both a classical and DRL based method when altering the manipulator characteristics (here the link dimension).

Generalizing to different manipulators We evaluated the proposed control framework with two different industrial robots: the Unimation Puma 560 and the Mitsubishi RV-6SL with varying kinematics. The objective was to determine if the same neural network and hyperparameters can be used to provide a motion plan trajectory for both arms even with varying kinematics. Plots in Figure 1 represent the average reward over time during the training process using the DRL framework. In both cases, the network and hyperparameters were held constant to demonstrate the generalizability of the framework. Both robot arms were able to satisfactorily reach the desired targets requiring 9 hours of training using a Titan X GPU.

Sensitivity analysis and robustness to manipulator alterations Our next set of experiments were focused on a sensitivity analysis where we evaluated whether the proposed DRL approach could accommodate changes in the robotic manipulator link lengths relative to a baseline kinematic model. To demonstrate this, we conducted a series of tests where the link lengths of the model in simulation were increased by a factor ranging from 5% and up to 20%, in increments of 5% using the robotic simulation framework. We then evaluated whether the DRL-based approach could still learn to reach the target object. In both scenarios, the target position remained fixed and collisions were not considered. In future efforts, our goal is to relax these constraints. We measured the final Euclidean distance between the manipulator and the target object after allowing

Table 2. Sensitivity Analysis

Model Error	**DRL Error (cm)**	**IK/FK Error (cm)**
5%	0.04	2.39
10%	0.09	6.29
15%	0.03	10.52
20%	0.10	13.95

the training to converge and compared it to the error associated to a control policy using a traditional inverse/forward kinematics which relies on a kinematic model. We used inverse kinematics to find a baseline set of joint angles that reach the desired object. We then used forward kinematics with the varying link lengths to compute the actual pose of the end effector. As the link lengths increase from the kinematic model, the error between the observed and estimated end effector position also increases whereas the DRL approach is able to learn the new kinematic model to prevent the error from increasing as presented in Table 2.

Comparisons between both network architectures Lastly, in Fig. 1 (left) we compared the average reward based on the reaching error between DRL1 and DRL2 network architectures. As can be seen the two network architectures perform about the same.

Videos and model Trained model weights and videos are made available under https://github.com/kdk132/drl_reach_task.

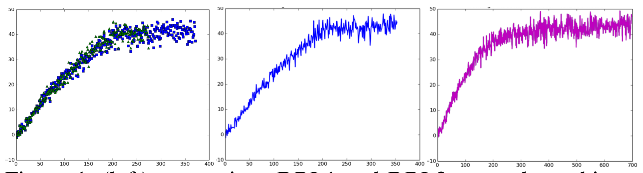

Figure 1. (left) comparison DRL1 and DRL2 networks architectures. (middle) Average Reward per Epoch (500 Steps per Epoch) for Puma Robot (right) and Mitsubishi Robot

4. Conclusion

This work describes a DRL-based manipulator control algorithm for a reaching task that takes the first steps towards generalizing to new manipulators and environmental changes. Our goal is to extend this to the physical robot domain and emphasize transfer learning from simulation to the the real robot arm.

References

[1] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*, 2016. 1

[2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 2

[3] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *arXiv preprint arXiv:1603.02199*, 2016. 1

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013. 1

[5] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *arXiv preprint arXiv:1509.06825*, 2015. 1

[6] M. Rickert. *Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems*. Dissertation, Technische Universität München, Munich, Germany, 2011. 2

[7] H. Yousef, M. Boukallel, and K. Althoefer. Tactile sensing for dexterous in-hand manipulation in roboticsa review. *Sensors and Actuators A: physical*, 167(2):171–187, 2011. 1

[8] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. I. Corke. Towards vision-based deep reinforcement learning for robotic motion control. *CoRR*, abs/1511.03791, 2015. 1