Reinforcement Learning with Cartesian Commands and Sim to Real Transfer for Peg in Hole Tasks

Manuel Kaspar, Jürgen Bock*

***Abstract*—We show how to learn robotic peg in hole tasks with reinforcement learning. Using the Operational Space Control framework enables us to learn contact rich tasks with adjustable degrees of freedom in cartesian space. We perform system identification with an CMA-ES optimizer for aligning a simulation environment with the dynamics of a real robot. By randomizing the dynamics during learning we can directly transfer a policy for a peg in hole task to a real KUKA LBR iiwa.[1]**

I. Introduction

Most of today's Reinforcement Learning (RL) research with robots is still dealing with toy tasks, that do not reach the requirements of industrial problems. This is partly due to the fact that training on real robots is very time-consuming. Moreover, it is not trivial to setup a system where the robot can learn a task, but does not damage itself or any task relevant items. Therefore, the idea of sim to real transfer [1] was introduced. While this idea seems convincing in the first place, bridging the reality gap is a major difficulty, especially when contact dynamics, soft bodies etc. are involved, where dynamics are difficult to simulate. This paper investigates possibilities for sim to real transfer while trying to make the task to learn as easy as possible by using the Operational Space Control framework (OSC) [2]. For some problems like redundancy resolution or inverse kinematics calculation good analytical solutions exist, so the framework takes care about those parts. Furthermore, we can decide to only learn actions in certain cartesian DOF which reduces the task dimensionality. Our current setup tries to perform a peg in hole task as shown in Fig. 1, where we currently fix the rotations as we know the needed final rotation and just learn the necessary translation for a successful insertion. In every timestep our action contains an offset $x_{des}, y_{des}, z_{des}$ to the current position $x_{cur}, y_{cur}, z_{cur}$. The target position is implicitly encoded into the observation vector as e.g. $x_{obs} = x_{cur} - x_{target}$ and the current setup has an insertion tolerance of 3mm. The OSC framework can map cartesian tasks into joint-torque commands and spans a spring, when a force prohibits it from reaching the desired position. This enables the robot to learn tasks, that require robot-environment contacts, because no hard position controller is used. We believe those are tasks where RL can bring benefits compared to traditional techniques.

*All authors are with *Corporate Research KUKA Deutschland GmbH* Augsburg, Germany. E-Mail: {manuel.kaspar, juergen.bock}@kuka.com

[1]This work has been supported by the German Federal Ministry of Education and Research (BMBF) in the project TransLearn (01DQ19007B).

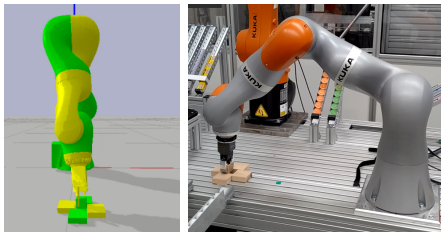

Fig. 1: Simulated and real setting

II. Related Work

Over the past years an increasing number of works tried to use sim to real transfer for learning robotic control: Progressive Nets [3] were proposed for giving the neural network a flexible way of using or not using past experience which has been collected in simulation, when fine tuning on a real system. Successful sim to real transfer for robots was demonstrated by [4] where in-hand manipulation of a cube is learned. [1] learn a policy to move an object to a specific position on a table and also introduce and analyze the idea of dynamics randomization in simulation.

III. Sim to Real Transfer

In this work we used the Soft-Actor-Critic (SAC) algorithm explained in [5]. In SAC not only a reward r is maximized, but also the entropy of the actor. The usage of this maximum entropy framework leads to robust policies, that do not collapse into a single successful trajectory but explore the complete range of successful trajectories. The objective in the maximum entropy framework is:
$\pi = arg\max_\pi \sum_t \mathbb{E}_{(s_t,a_t)\sim p_\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))]$,
where α is an automatically adjusted temperature parameter that determines the importance of the entropy term [6]. We stack n past observations and actions into the observation vector thereby trying to recover the Markov-condition [7] and giving the network the possibility to figure out the dynamics of the system. As reward function we used

$$C_{pos} = \alpha \cdot \|x_{dist}\|_2 + \beta \cdot \|x_{dist}\|_1 \tag{1}$$

$$C_{action} = \gamma \cdot \sum_{i=0}^{nr\_actions} a_i^2 \tag{2}$$

$$C_{bonus} = 50 \text{ if peg reached hole bottom} \tag{3}$$

$$C_{total} = -C_{pos} - C_{action} + C_{bonus} \tag{4}$$

where $x_{dist} = x_{target} - x_{current}$ and $\alpha = 0.1$, $\beta = 0.9$, $\gamma = 0.0005$.

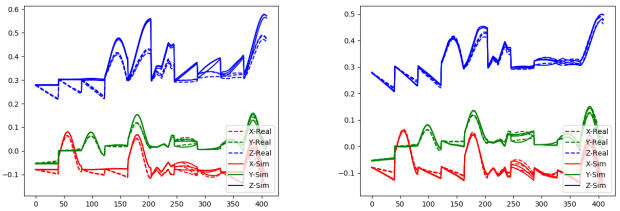

Fig. 2: Real and simulated trajectories at the beginning of the optimization process (left) and afterwards (right). Every sub-trajectory consists of 50 steps (x-axis). So, the figure shows 9 sub-trajectories behind each other.

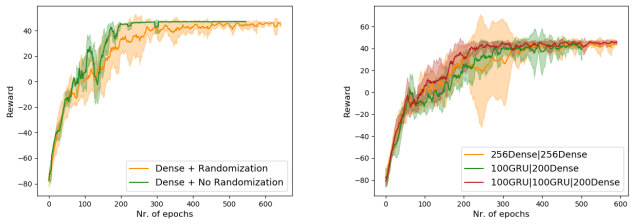

Fig. 3: Effect of network architecture and randomization

A. Simulation environment

We use the pybullet simulation, where we load an KUKA LBR iiwa 14kg with appropriate dynamics values and an attached Schunk MEG50 gripper. We directly command torques to the joints of the robot.

B. Dynamics and Environment Randomization

As introduced in [4] and [1] we performed dynamics and environment randomization for transferring our policy from simulation to the real world. We randomize link masses, surface friction, joint damping, gravity, goal position (x, y) and goal imprecision (x, y). Goal imprecision means an offset between the real hole center and the position we tell the agent, what emulates a noisy vision system. Indeed, this randomization led to an emergence of a more subtle search strategy.

C. System Identification

In our first trials for using a policy, which was learned in simulation and transferred to the real robot, we found, that it worked pretty poorly. The dynamics of the real robot were too different from the dynamics of the simulated one. Dynamics randomization is important for getting robust against (slightly) different dynamics. Nevertheless, in our experiments it showed to be important that simulated dynamics are close to the real ones to enable a successful sim to real transfer. Therefore, we performed a special type of system identification, where we run actions on the real robot and tried to change parameters in simulation to make trajectories as similar to the real trajectories as possible. We used the CMA-ES [8] algorithm to change the gravity, link masses and joint damping simulation parameters and let them optimize to minimize the 2-norm $(\sum_{i=1}^{n}(v_i)^2)^{\frac{1}{2}}$ where v are three end effector positions. Fig. 2 shows the real and simulated trajectory before the system identification and afterwards. The trajectory after the identification is much closer to the real trajectory.

IV. Evaluation and Future Work

We evaluated the possibility of learning a peg in hole task in simulation and transferring it to a real robot. Simulation and OSC run in 5ms steps, while the learning module runs with 160ms. Training in simulation works very well in our experiments with a final performance of nearly 100%. As we give a bonus of 50 after a successful insertion, we can see the performance rising in Fig. 3 around epoch 150, when the robot starts succeeding in most trials. Also, the effect of environment and parameter randomization can be observed. The randomization makes the task more difficult and also finally a bit more unstable. Compared to [4] we can't observe a positive effect when using a recurrent architecture. When transferring the policy to the real robot without further fine-tuning we got a performance of 20 successful out of 20 tried insertions. The policy was also robust to manual perturbations of the robot like pushing it away from the target. Compared to the simulation, the real-world trajectory was slower with inserting, because it searches quite some time for the hole. In our future work we plan to investigate the possibilities of using sim to real transfer on industrial peg in hole tasks where the insertion tolerance is in the sub-millimeter range. In our view tasks that involve contact are the most interesting class of problems for applying RL. With today's industrial robots, force sensitive tasks require a large amount of expert knowledge to program and a big amount of time for fine tuning it to specific applications. Nevertheless, very often those tasks with friction, soft objects or snap-in events are also inherently difficult to simulate and we therefore want to see if solving them is possible with sophisticated simulation environments and what parts need to be fine-tuned on the real robot.

V. References

[1] X. B. Peng, M. Andrychowicz, W. Zaremba, *et al.*, "Sim-to-real transfer of robotic control with dynamics randomization," *CoRR*, vol. abs/1710.06537, 2017.

[2] O. Khatib, "A unified approach for motion and force control of robotic manipulators: The operational space formulation," *IEEE Journal of Robotics and Automation*, 1987.

[3] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, *et al.*, "Progressive neural networks," *CoRR*, vol. abs/1606.04671, 2016.

[4] OpenAI, M. Andrychowicz, B. Baker, *et al.*, "Learning dexterous in-hand manipulation," *CoRR*, vol. abs/1808.00177, 2018.

[5] T. Haarnoja, A. Zhou, P. Abbeel, *et al.*, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018.

[6] T. Haarnoja, A. Zhou, K. Hartikainen, *et al.*, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018.
[7] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, vol. abs/1701.07274, 2017.
[8] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.