Planning Approximate Exploration Trajectories for Model-Free Reinforcement Learning in Contact-Rich Manipulation

Sabrina Hoppe[1,2] and Zhongyu Lou[1] and Daniel Hennes[2] and Marc Toussaint[2]

***Abstract*—Recent progress in deep reinforcement learning has enabled simulated agents to learn complex behavior policies from scratch, but their data complexity often prohibits real-world applications. The learning process can be sped up by expert demonstrations but those can be costly to acquire. We demonstrate that it is possible to employ model-free deep reinforcement learning combined with planning to quickly generate informative data for a manipulation task. In particular, we use an approximate trajectory optimization approach for global exploration based on an upper confidence bound of the advantage function. The advantage is approximated by a network for Q-learning with separately updated streams for state-value and advantage that allows ensembles to approximate model uncertainty for one stream only. We evaluate our method on new extensions to the classical peg-in-hole task, one of which is only solvable by active usage of contacts between peg tips and holes. The experimental evaluation suggests that our method explores more relevant areas of the environment and finds exemplar solutions faster – both on a real robot and in simulation. Combining our exploration with learning from demonstration outperforms state of the art model-free reinforcement learning in terms of convergence speed for contact-rich manipulation tasks.**

***Index Terms*—Deep Learning in Robotics and Automation, Dexterous Manipulation, Learning and Adaptive Systems**

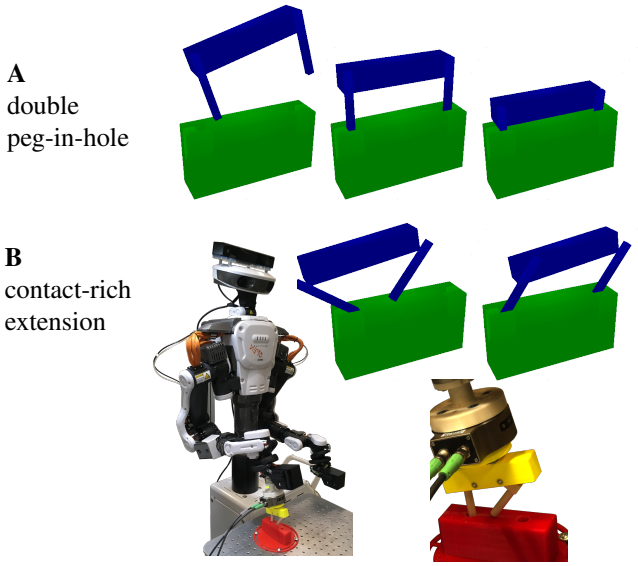

Fig. 1. Double peg-in-hole: With stiff joints (first row), the task can be solved by avoiding contact. With dangling pegs (e.g. enforced by off-axis joints) sustained contact for each successful solution is required (second row). Also 3d-printed for a real robot.

I. Introduction

THE field of deep reinforcement learning has seen rapid progress in recent years, with deep Q-networks reaching human-level performance in various games [1] as well as simulated dynamical control tasks [2]. Most of these systems can be trained in simulation using millions of samples and often autonomously without human assistance.

For applications in robotics this is often barely feasible as real-world experience can be slow, costly and might need human supervision throughout. When enough data is available, e.g. by collecting experience from several robots over the span of multiple months, deep reinforcement learning has been proven to outperform many conventional approaches [3].

In model-free reinforcement learning, exploration is notoriously hard since only dithering strategies or local exploration can be used in general. The problem of exploration can be circumvented by expert demonstrations [4]. If those are not available, model-based solutions can also be used instead [5]. However, contact-rich manipulation is one of those domains where accurate models are hard to obtain and purely data-driven methods have been shown to outperform model-based approaches [6].

In this work we investigate if model-free deep reinforcement learning combined with trajectory optimization can perform efficient exploration. Using this exploration data analogously to demonstrations, we show that state of the art learning from demonstration can outperform model-free reinforcement learning from scratch. For efficient exploration, we use global exploration in state space via *trajectory optimization* based on a Bayesian-like Upper Confidence Bound (UCB) on the advantage function. To approximate the UCB efficiently, we extend networks that consist of separate streams for state-value and advantage, allowing to estimate model uncertainty from ensembles on the advantage stream only.

The approach is evaluated on two new extensions of the peg-in-hole paradigm. We introduce *double peg-in-hole* – two pegs connected by a bar need to be inserted into two holes.

Manuscript received: February 02 2019; Revised May 22 2019; Accepted June 06 2019.

This paper was recommended for publication by Editor Tamim Asfour upon evaluation of the Associate Editor and Reviewers' comments.

[1]S. Hoppe and Z. Lou are with Bosch Corporate Research first.last@de.bosch.com

[2]S. Hoppe, D. Hennes and M. Toussaint were with the Machine Learning and Robotics Lab when the work was conducted. University of Stuttgart, Germany. first.last@ipvs.uni-stuttgart.de

Digital Object Identifier (DOI): see top of this page.

With stiff connections task A is equivalent to traditional peg-in-hole: given the right pose, it is possible to insert the peg while mostly avoiding contact. With dangling pegs and off-axis joints (see Figure 1), sustained contact is needed to solve task B. The task was also 3D printed for evaluation on a real robot (see https://youtu.be/JTfeHhWSb0Y).

II. BACKGROUND

We consider a standard reinforcement learning setup where an agent interacts with its environment over discrete time steps $t = 1, \dots, T$. At each time t, the agent can observe its state s_t and take an action a_t which determines the next state s_{t+1} and an associated reward r_t.

A policy π maps from states to actions and induces a trajectory $\tau_\pi = (s_0, a_0, s_1, \dots s_T)$. The sum of all (discounted) rewards collected along a trajectory τ_π is called return: $R_t^\pi = \sum_{i=t}^{T} \gamma^{i-t} r_i$. The value of a state is then defined as the expected future return when following policy π: $V^\pi(s_t) = \mathbb{E}\left[R_t^\pi\right]$. The expected future return for executing an arbitrary action a_t and then following the policy is called $Q^\pi(s_t, a_t) = \mathbb{E}\left[r_t + \gamma \cdot R_{t+1}^\pi\right]$. The (dis)advantage the agent can expect from choosing a_t rather than following π is $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$. The agent's goal is to find an optimal policy π^* such that the expected future return is maximized from any state.

Nowadays one of the most powerful function approximators are neural networks and therefore they have also been used to fit Q-functions [1]. Most techniques from reinforcement learning with (conventional) function approximation transfer to deep reinforcement learning, e.g. double Q-learning [7]. Additionally, there is a large body of work presenting network-specific innovations (for an overview see e.g. [8]).

In actor-critic settings, the policy is approximated as one function (the so-called actor), while a second function approximates the quality of a state-action pair - e.g. using the Q-value. In the context of deep learning, the critic estimates can then be used as a training signal for an actor network, e.g. using deep deterministic policy gradients (DDPG) [9].

III. RELATED WORK

Deep Learning and in particular non-stationary iterative Deep Reinforcement Learning are infamously sample-inefficient.

If prior knowledge about the task is available, model-based reinforcement learning can leverage this information and enable efficient exploration [10]. In the presence of contacts however, models are notoriously hard to obtain and are often outperformed by model-free methods [6]. Even when models are theoretically available, robotics practitioners might struggle with limited knowledge about the materials at hand or the availability of suitable environment representations.

Without imposing models or structure on the functions to learn, exploration is the main bottleneck. There are different dithering strategies ranging from ϵ-greedy [11] and Gaussian noise to more sophisticated processes like the temporally correlated Ornstein-Uhlenbeck [9] or adaptive parameter space noise [12]. However, all dithering strategies mainly explore locally around states that the current policy tends to visit and it can take long until the relevant part of the state space is discovered – even more so, if some area is only reachable after a specific series of action.

Reward shaping can help to lure the agent into relevant parts of the state space but it is typically manual effort and easily leads to unintended local minima [13].

Count-based exploration schemes aim to explore the full state space uniformly by keeping track of which area was visited how frequently [14], [15]. This behavior is often seen as an analogy to human *intrinsic motivation* [16], but in many cases it is unrealistic to cover the full state space. If an agent *knows what it knows* [17], [18] however, it can deliberately explore areas it is unsure about. Model uncertainties from neural networks can formally be obtained from Bayesian Neural Networks that consider distributions over weights. Since those are often intractable, there are approximations such as network ensembles or multiple instances of dropout at test time [19], [20]. Using information-theoretic criteria like the upper confidence bound (UCB), these approximations can be used to speed up learning [21]. Still, existing strategies only explore greedily for a current state. In our method we combine UCB-based exploration with global state space exploration via trajectory optimization to also explore far-away states if they seem promising in terms of their UCB.

Model-based and model-free methods have also been combined in the past: For instance, guided policy search turns reinforcement learning into iterative supervised learning from model-based solutions [5]. This approach is limited to tasks where model-based solutions are known or easy to obtain though.

To perform our global exploration, we require a coarse *steering function*: that is, we assume that some function can return actions that will guide the agent approximately towards some goal state. In contrast to model-based learning, this steering function is only used for exploration while the learned policy remains model-free. Therefore the method is quite robust to perturbations in this function. Importantly, our exploration scheme even generates informative data if it does not solve the task, as we will show in our evaluations.

Supervised learning from fixed input-output pairs is generally much more sample efficient than iterative reinforcement learning where changing optimization targets introduce some non-stationarity. Instead of model-based solutions also expert demonstrations can help speed up learning [4] but they are often costly to acquire. When available, already single demonstrations can be enough for robots to learn [22]. In this work, we follow the idea that learning from demonstration could also be used on top of automatically generated demonstrations. When exploration is separated from policy learning, the modalities can change between the two stages (e.g. [5]) and, crucially, exploration can be much faster when the value function can change quickly without hindering convergence of the actor network.

Convergence can be additionally sped up by using the value function as a control variate to reduce variance in the policy gradient [23]. This idea has been transfered to deep learning in [24] but their method 'loses the original semantics of V

and A'. We introduce novel update rules that enable separate training of both streams, preserving the semantics of both streams and thus allowing to build policies on the actual advantage.

IV. APPROACH

We split the problem of reinforcement learning into two stages: exploration and learning from demonstration. For the second stage, all samples from the first phase are slowly added to a new replay memory, similar to [25]. For the first stage we combine the following components to efficiently guide exploration for DDPG-like Q-learning [9], each of which will be discussed in greater detail in one of the following paragraphs.

1) We structurally encode the decomposition of Q-values into state-value and advantage in a network architecture with two streams similar to [24] thereby reducing the gradient variance which leads to faster and more stable training. To preserve the semantics of both streams, we introduce novel training updates for both streams.
2) Exploiting this network structure, we introduce an ensemble of networks in the advantage stream to approximate model uncertainty. We argue that only this portion of uncertainty in the Q-function is relevant for exploration and deriving a policy.
3) Using trajectory optimization under coarsely approximated dynamics, we globally explore areas of high model uncertainty by choosing trajectories such that the sum over the upper confidence bound (UCB) on the advantage for a set of way points is maximized.

Advantage Networks

Decomposing the state-action value Q into state-value V and the advantage is formally equivalent to using V as a control variate which leads to more stable learning if V correlates strongly with Q (but A does not) [23], [24]. If, for instance in our environment, there is a reward at each time step, then V and Q sum up (discounted) rewards over potentially long trajectories, while the advantage is the difference between V and Q. Effectively this means that V and Q might be in a very large range of absolute values, while A might be on a range of very small numbers. In those cases, V dominates gradients and uncertainty measures derived from Q. If a policy is derived from Q, its gradient is also dominated by V although actually only A matters: $\arg\max_a Q(s,a) = \arg\max_a A(s,a)$

We structurally encode the split into Q and A as two fully separate streams within our network that consists of four multilayer perceptrons (MLP): The inputs, i.e. state and action, are each processed by one MLP. The outputs of both MLPs are added element-wise and fed into a third MLP that predicts the advantage $A(s,a)$. Since advantage values associated with the optimal policy cannot be larger than zero, we use a ReLU activation on the advantage stream and change the sign of the output. The output of the state-MLP is also fed into the fourth MLP to predict $V(s)$ (see Figure 2 for illustration). In contrast to [24] the network branches are trained separately as described in the next paragraph.

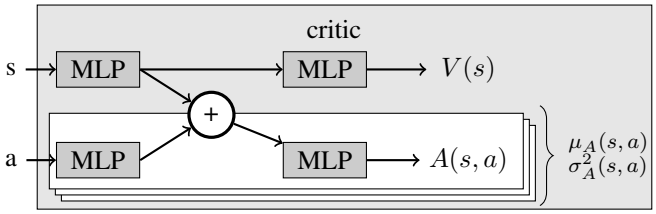

Fig. 2. Bootstrapped Advantage Network: Four multilayer Perceptrons (MLP), each consisting of 4 fully-connected layers - the first three using leaky ReLU activations [26], the last one using linear activations. The advantage stream of the network is bootstrapped, i.e. there are B copies of the lower part of the network.

Greedy Exploration using Bootstrapped Advantage Networks

A *greedy* policy only considers single actions from a given state s. Splitting Q into V and A, the greedy policy only depends on the advantage:

$$\pi(s) = \arg\max_{a\in\mathcal{A}}(A(s,a) + V(s)) = \arg\max_{a\in\mathcal{A}} A(s,a) \quad (1)$$

Bootstrapping uses ensembles of networks to get an uncertainty estimate [19] which we will use for exploration. In preliminary investigations, bootstrapping has outperformed dropout-based methods for our setting. We only add bootstrapping to two out of four MLPs, such that we obtain a distribution over advantage predictions, but still only predict a single state-value $V(s)$. If multiple state-value estimates were used as well, the direct comparison between advantages would become misleading. The full architecture as shown in Figure 2.

To determine the next action to take for exploration, acquisition functions inspired by Bayesian Blackbox Optimization can be used, e.g. the upper confidence bound (UCB) in a Q-learning setting [21]:

$$\begin{aligned}\pi(s) &= \arg\max_{a\in\mathcal{A}} \mathrm{UCB}_Q(s,a) \\ &= \arg\max_{a\in\mathcal{A}} \left(\mu_Q(s,a) + \kappa\cdot\sigma^2_Q(s,a)\right)\end{aligned}$$

where κ is a hyper-parameter that trades off variance and mean[1].

Each advantage network is updated independently with data from its separate replay memory. During each episode all data is written to one of these replay memories. To update the state-value network, samples are drawn from all replay memories.

All networks are trained using stochastic gradient descent to approximate iterative targets $\hat{A}$ and $\hat{V}$ which are computed from replay memory samples $(s_t, a_t, r_t, s_{t+1})_i$.

$$\begin{aligned}\hat{V}(s_t) &= r_t + (1 - \mathbb{1}_{s_{t+1}\text{ is terminal}})\cdot\gamma V(s_{t+1}) \\ \hat{A}_b(s_t,a_t) &= \hat{Q}_b(s_t,a_t) - V(s_t)\end{aligned}$$

where

$$\begin{aligned}\hat{Q}_b(s_t,a_t) &= r_t + (1 - \mathbb{1}_{s_{t+1}\text{ is terminal}})\cdot\gamma Q_{\hat{b}}(s_{t+1},\pi(s_{t+1})) \\ \pi(s_t) &= \arg\max_{a\in\mathcal{A}} \mu_A(s_t,a)\end{aligned}$$

where $\mathbb{1}$ is the indicator function and μ_A denotes the mean advantage for a given input. Since μ_A can in general be a

[1]For all experiments, we fixed κ to 1.96 which corresponds to the 97.5 percentile under a Gaussian.

highly non-convex function, the argmax to compute π was approximated by evaluating 100 sampled actions. Empirically, a higher sample size did not improve performance in our evaluation setting. Moreover, our experiment on the real robot has confirmed that the additional time for this update is negligible compared to the time needed for the robot rollouts. $\hat{b} \neq b$ is a uniformly sampled replay memory index that was introduced in analogy to double Q learning [27], [7]. It reduces bias in estimating Q which is caused by the over-estimation of the typical Q update using $Q(s, \arg\max_a Q(s,a))$. This bias occurs analogously if A is estimated based on $\arg\max_a A(s,a)$ and its prevention turned out crucial for performance.

Global Exploration using Trajectory Optimization

Just using the UCB over the advantage is already enough to derive a greedy exploration policy. This will explore locally but often it is necessary to go through well-explored areas for several steps before reaching a point of high uncertainty. To incorporate global state space exploration into the UCB policy, we use Trajectory Optimization to compute a series of h waypoints[2] that approximately maximize the sum over UCBs along a number of waypoints using the Powell method [28]: $\sum_{i \in 0..h} \max_a \text{UCB}_A(s_i, a)$. Thus, the resulting exploration is referred to as *global*, because the full state space is considered in general.

The agent is supposed to move towards each waypoint successively. This could in principle be achieved by exploiting a dynamics model, but contacts and friction are hard to model and some of the physical properties of relevant materials might be unknown. Instead we rely on general domain knowledge for robotic manipulation tasks in form of a *steering function*: It uses straight-line velocity control to reach any robot configuration. This is only a coarse approximation of the dynamics and will fail whenever obstacles or contacts come into play. In section V, we demonstrate the robustness of exploration under noise and miscalibration of the steering function.

Thus, the agent will subsequently try to reach the waypoints by moving straight (in configuration space) towards it, which leads to piecewise linear motion in configuration space. Since an action in most reinforcement learning settings is restricted, the agent might effectively take several small steps towards the waypoint. If the waypoint is reached, the optimal action $\arg\max_a \text{UCB}_A(s,a)$ is executed. If the agent does not get any closer to a waypoint, e.g. because of obstacles, the agent moves on to the next waypoint. After the last waypoint was reached or discarded, we use the greedy exploration policy as a fallback for the rest of the episode.

Empirically, we found it beneficial to start with a small number of episodes using a random policy and introduce some random steps after each waypoint was reached to further explore the area of high uncertainty[3] Starting from its current position, the steering function will drive the agent towards the next waypoint.

[2]we fixed $h = 5$ waypoints per rollout of 200 steps.

[3]we used 5 random steps after each waypoint and 10 random episodes in the beginning of each experiment.

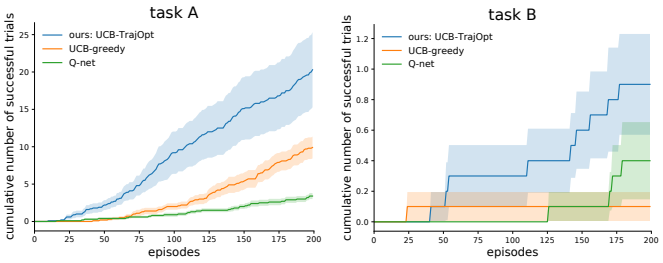

Fig. 3. Ablation study of off-policy exploration performance: All algorithms were tested on a grid of hyperparameters with ten random seeds per hyperparameter configuration. The best performance is shown as the mean over random seeds as a solid line, the shaded area covers one standard deviation of the mean estimator.

V. EXPERIMENTAL RESULTS

Our exploration scheme is supposed to generate useful data such that in a second step, these samples can give a head start to train a policy, similar to learning from demonstration.

We have evaluated how fast our exploration scheme is able to find successful solutions, how robust the scheme is to a perturbed or mis-calibrated steering function, the feasibility of our exploration strategy on a real robot, and if the combination of exploration and learning from demonstration is more sample-efficient than direct reinforcement learning from scratch.

Simulation Tasks

All results are based on the double peg-in-hole tasks illustrated in Figure 1 where two pegs need to be inserted into an object with two holes. Bot state and action spaces are continuous. The blue object can move in a plane, i.e. there are two positional degrees of freedom and one rotational ($\left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$). The state is defined by the raw coordinates for these three dimensions. The state space was manually restricted to some box area around the green object and scaled linearly to $[-1, +1]$ in all dimensions. The initial state of the blue object was uniformly chosen from all states at the upper end of the box and combined with any possible rotation.

The agent uses velocity control with an action being the 3D offset of the next state from the current state. The simulation is designed to wait until the agent including both pegs has reached a stable position before the next action is applied. The terminal states were manually defined such that only states with both pegs correctly inserted were included. The reward is based on the euclidean distance in state space between the current state s and the goal g: $r_t = \exp\left(-\frac{||s-g||_2}{\sigma}\right) - 1$. [4]

Since the state representation does not include the peg rotation, task B is technically partially observable. However, due to the nature of the task, locally the unobserved degrees of freedom (DoF) are often a function of the observed DoFs.

A. *Exploration Speed*

We first evaluate the speed of exploration, measured by the number of successful trials generated during exploration, i.e. in

[4]σ was manually fixed to 0.3 for all experiments

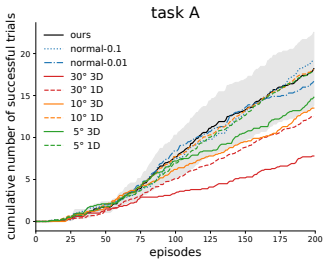

Fig. 4. The experiment from Figure 3 re-run with different perturbations to the steering function: Gaussian noise ('normal-STD') and systematic mis-calibrations of 5°, 10° and 30° around one ('X° 1D') or all three action dimensions ('X° 3D'). Each line represents the mean over results from 10 different random seeds, the gray shade represents the standard deviation around the mean estimator without disturbance.

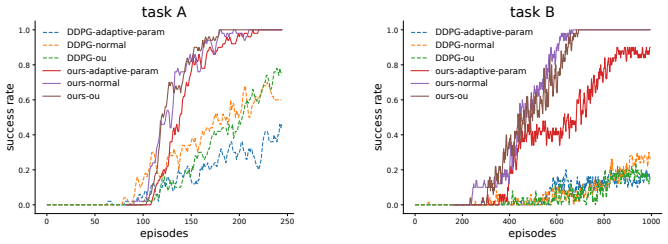

Fig. 5. Sample efficiency in policy learning: Policies learned from scratch ('DDPG', dashed lines) are compared to policies that started with data from our exploration scheme ('ours', solid lines starting after exploration episodes). Each line represents the ratio of successful policies during on-policy evaluation from ten different random seeds.

off-policy behavior. In an ablation study, we compare the full system (ours) to two baselines: (1) a system with bootstrapped Q networks (Q-net), i.e. without the split into state-value and advantage. We changed the number of states in the last MLP to match the total number of parameters used. (2) a system with advantage networks but with greedy UCB exploration instead of trajectory optimization (UCB-greedy). The approach in [21] is a combination of these two baselines, where a greedy exploration is chosen based on Q networks. Each method was tuned with a hyper parameters grid search[5] and ten random seeds per parameter set. Only the set with best performance on average was chosen for evaluation. Figure 3 illustrates the mean cumulative number of episodes that ended in a terminal state for different random seeds. Our full exploration system achieves more successful episodes than the same algorithm based on Q networks or a greedy UCB policy.

B. Robustness to Corrupted Steering Functions

To perform global state-space exploration, our method uses a so-called steering function to perform straight-line velocity control towards any configuration space state. Since the steering function is only used for exploration, we hypothesized that our approach would be robust to perturbations. To test this, we re-ran the experiment from the previous section on task A with different kinds of disturbances: Additive Gaussian noise ('normal-X') that might cancel out over longer trajectories as well as systematic mis-calibrations of 5°, 10° and 30° around one ('X° 1D') or all three action dimensions ('X° 3D'). This experiment was conducted on the best set of hyperparameters from the previous section. The results in Figure 4 show that indeed the system is robust to most disturbances: Gaussian noise does not affect performance on average, presumably because it cancels out over multiple steps. Similarly, rotations of 5° and 10° around one axis can be tolerated. The performance decreases for stronger disturbances. Except for the 30° rotation around all dimensions, all exploration schemes based on noisy steering functions outperform the best baseline from Figure 3 on average.

[5]learning rates $\{10^{-2}, 10^{-3}, 10^{-4}\}$, batch sizes $\{32, 64\}$, noise scales $\{0.1, 0.2, 0.3\}$. 50 update steps were performed by SGD after each rollout.

C. Feasibility on a Real Robot

We have replicated the experiment from subsection V-A, i.e. the exploration phase, using a real KAWADA Nextage Open and a 3D printed version of the harder double peg-in-hole (task B) as illustrated in Figure 1. States and actions were defined exactly as in simulation. The object was approximately 4.5cm wide, the state space covers 8cm from right to left, 5cm from the lowest to the highest point in z direction and a rotation of up to 30°. One action could move the robot by up to 0.5cm. To avoid harmful collisions, a force-torque sensor was installed at the end effector. Whenever the force or torque exceeded a manually set threshold, the current action was aborted and the end effector moved upwards until the pressure was relieved. Since evaluation on real hardware is too time-consuming to perform statistically sound analyses with multiple hyperparameters and random seeds, we used those hyperparameters that performed best in simulation. To save time, we only executed 2 random episodes in the beginning. As the video illustrates, visually plausible exploration behavior emerges and both tangling pegs are inserted after less than 1200 steps.

D. Sample Efficiency for Policy Learning

Exploration should at some point discover successful behavior but since the main purpose in our case is to facilitate learning, we here examine how fast a full policy can be trained utilizing the exploration data. This means we consider on-policy data in the following.

A DDPG actor-critic setup[6] serves as a baseline for learning from scratch with Gaussian noise, Ornstein-Uhlenbeck [9] and adaptive parameter noise [12]. Additional parameters for these noise processes were included in the hyper parameter grid search. For learning from demonstration, we use the same setup but slowly add experience from our exploration scheme to the replay memory similar to [25].

We used the data for the best performing hyperparameter set from our ablation study as exploration samples. Since all configurations were run 10 times, we chose the random seed whose performance was closest to average to keep the data as representative as possible. Empirically, we found it beneficial

[6]as implemented in the openAI baselines repository

to discard the initial random episodes and add samples from 10 exploration episodes at each training episode.

Training an actor from the exploration data in a supervised setting without any further environment interaction failed completely – most likely due to compound errors known in behavioral cloning. Also training an actor network while exploring with our strategy off-policy was empirically unstable unless a very small learning rate was used (which in turn counteracts the sample efficiency).

Figure 5 compares the performance of a vanilla DDPG trained from scratch to our DDPGfD (DDPG from demonstration) using 80 episodes of exploration data for the simpler task and 160 for the harder one. There were no successful samples in the demonstration data before episode 80 or 160 respectively, but it can be observed that injecting the data from our exploration scheme speeds up learning anyway. Interestingly, the improvement of the full policy trained on our data over training from scratch is larger for the harder manipulation task B, although the steering function is clearly less accurate in task B. Together this indicates that the data created is generally informative.

VI. CONCLUSION

To overcome the infamous sample inefficiency of exploration for model-free reinforcement learning from scratch, we have proposed to use global state space exploration via trajectory optimization to generate informative data in a first step and then use this data in analogy to expert demonstrations when training a full model-free policy in a second step.

The approximate trajectory optimization aims to maximize the upper confidence bound on advantages at a set of way points that the agent visits sequentially using some coarse steering function. We have also demonstrated the robustness of the approach to noise and systematic mis-calibration of the steering function. We have shown that our approach rapidly finds successful sample trajectories; and experience collected during this exploration can significantly speed up policy learning even if the exploration data does not contain successful samples. These results indicate that data from our exploration scheme is generally more informative than existing strategies. Visual inspection of the agent's exploratory behavior on a real robot shows that our method identifies areas of high uncertainty and relevance to the task (e.g. the holes of the object rather than its contour in general).

Separating exploration from policy learning in general also allows to change modalities in between. We leave it to future work to investigate to which degree our exploration can also speed up learning in high-dimensional state spaces, e.g. image space – or whether low-dimensional spaces with a specific kind of dynamics can be automatically inferred from high-dimensional input data.

We argue that this work represents one successful way of combining planning and model-free reinforcement learning, and will explore further combinations of classical robotics techniques and model-free learning in the future.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016, pp. 1928–1937.

[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," *CoRR*, vol. abs/1603.02199, 2016.

[4] S. Schaal, "Learning from demonstration," in *NeurIPS*, 1996.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, vol. 17, no. 1, pp. 1334–1373, 2016.

[6] N. Fazeli, S. Zapolsky, E. Drumwright, and A. Rodriguez, "Learning data-efficient rigid-body contact models: Case study of planar impact," in *CoRL*, 2017, pp. 388–397.

[7] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI*, 2016.

[8] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

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

[10] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," in *NeurIPS*, 2016, pp. 1109–1117.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[12] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," *arXiv preprint arXiv:1706.01905*, 2017.

[13] K. Lowrey, A. Rajeswaran, S. M. Kakade, E. Todorov, and I. Mordatch, "Plan online, learn offline: Efficient learning and exploration via model-based control," *CoRR*, vol. abs/1811.01848, 2018.

[14] R. I. Brafman and M. Tennenholtz, "R-max-a general polynomial time algorithm for near-optimal reinforcement learning," *JMLR*, vol. 3, no. Oct, pp. 213–231, 2002.

[15] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," in *NeurIPS*, 2017, pp. 2750–2759.

[16] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *NeurIPS*, 2005, pp. 1281–1288.

[17] L. Li, M. L. Littman, and T. J. Walsh, "Knows what it knows: a framework for self-aware learning," in *ICML*. ACM, 2008, pp. 568–575.

[18] M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer, "Exploration in model-based reinforcement learning by empirically estimating learning progress," in *NeurIPS*, 2012, pp. 206–214.

[19] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," in *NeurIPS*, 2016, pp. 4026–4034.

[20] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *ICML*, 2016.

[21] R. Y. Chen, S. Sidor, P. Abbeel, and J. Schulman, "Ucb and infogain exploration via q-ensembles," *CoRR*, vol. abs/1706.01502, 2017.

[22] P. Englert and M. Toussaint, "Learning manipulation skills from a single demonstration," *IJRR*, vol. 37, no. 1, pp. 137–154, 2018.

[23] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *JMLR*, vol. 5, pp. 1471–1530, 2001.

[24] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *ICML*, 2016, pp. 1995–2003.

[25] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *CoRR*, vol. abs/1707.08817, 2017.

[26] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML*. Citeseer, 2013.

[27] H. van Hasselt, "Double q-learning," in *NIPS*, 2010.

[28] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The computer journal*, vol. 7, no. 2, pp. 155–162, 1964.