Improved Exploration through Latent Trajectory Optimization in Deep Deterministic Policy Gradient

Kevin Sebastian Luck[1], Mel Vecerik[2], Simon Stepputtis[1], Heni Ben Amor[1] and Jonathan Scholz[2]

Abstract— Model-free reinforcement learning algorithms such as Deep Deterministic Policy Gradient (DDPG) often require additional exploration strategies, especially if the actor is of deterministic nature. This work evaluates the use of model-based trajectory optimization methods used for exploration in Deep Deterministic Policy Gradient when trained on a latent image embedding. In addition, an extension of DDPG is derived using a value function as critic, making use of a learned deep dynamics model to compute the policy gradient. This approach leads to a symbiotic relationship between the deep reinforcement learning algorithm and the latent trajectory optimizer. The trajectory optimizer benefits from the critic learned by the RL algorithm and the latter from the enhanced exploration generated by the planner. The developed methods are evaluated on two continuous control tasks, one in simulation and one in the real world. In particular, a Baxter robot is trained to perform an insertion task, while only receiving sparse rewards and images as observations from the environment.
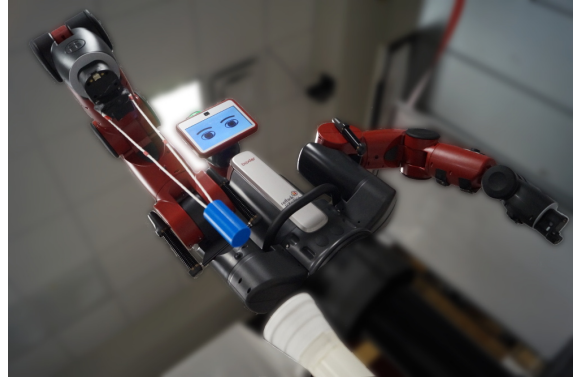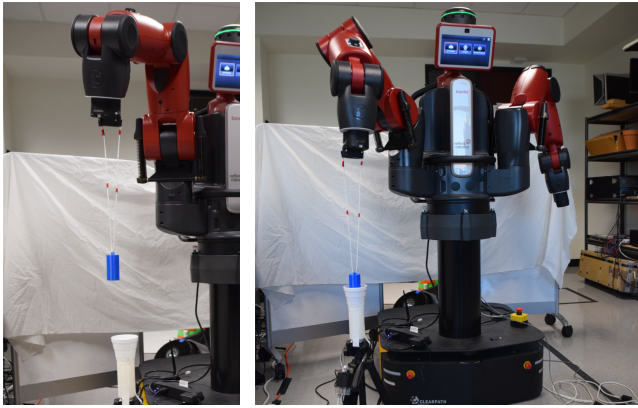
Fig. 1: A Baxter robot learning a visuo-motor policy for an insertion task using efficient exploration in latent spaces. The peg is suspended from a string.

I. INTRODUCTION

Reinforcement learning (RL) methods enabled the development of autonomous systems that can autonomously learn and master a task when provided with an objective function. RL has been successfully applied to a wide range of tasks including flying [24], [17], manipulation [26], [9], [12], [3], [1], locomotion [10], [13], and even autonomous driving [6], [7]. The vast majority of RL algorithms can be classified into the two categories of (a) inherently stochastic or (b) deterministic methods. While inherently stochastic methods have their exploration typically built-in [4], [18], their deterministic counterparts require an, often independent, exploration strategy for the acquisition of new experiences within the task domain [11], [8]. In deep reinforcement learning, simple exploration strategies such as Gaussian noise or Ornstein-Uhlenbeck (OU) processes [25], which model Brownian motion, are standard practice and have been found to be effective [11]. However, research has shown that advanced exploration strategies can lead to a higher sample-efficiency and performance of the underlying RL algorithm [14]. In practice, there are two ways to incorporate advanced exploration strategies into deterministic policy search methods. Where possible, one can reformulate the deterministic approach within a stochastic framework, such as by modeling the actions to be sampled as a distribution. Parameters of the distribution can then be trained and are tightly interconnected with the learning framework. One example for this methodology, is the transformation of Policy Search with Weighted

Returns (PoWER) [8] into Policy Search with Probabilistic Principal Component Exploration (PePPEr) [14]. Instead of using a fixed Gaussian distribution for exploration, the noise generating process in PePPEr is based on Probabilistic Principal Component Analysis (PPCA) and generates samples along the latent space of high-reward actions. Generating explorative noise from PPCA and sampling along the latent space was shown to outperform the previously fixed Gaussian exploration. Alternatively, one can choose to optimize the exploration strategy itself. Examples of this methodology are count-based exploration strategies [22], novelty search [21] or curiosity-driven approaches [16] which can be transferred with ease to other algorithms or frameworks. Typically, when incorporating these techniques into reinforcement learning, they are limited to local exploration cues based on the current state. This paper aims to combine the model-free deep deterministic policy gradient method with a model-based exploration technique for increased sample-efficiency in real world task domains. The proposed method generates exploratory noise by optimizing a (latent) trajectory from the current state to ideal future states, based on value functions learned by an RL algorithm. This experience is, in turn, used by the RL algorithm to optimize policy and value functions in an off-policy fashion, providing an improved objective function for the trajectory optimizer. We investigate whether this strategy of formulating exploration as a latent trajectory optimization problem leads to an improved learning process both in simulation, as well as in a robotic insertion task executed solely in the real world. In particular, we apply our approach to a challenging, flexible insertion task as seen in Fig. 1.

[1]Interactive Robotics Lab, Arizona State University, Tempe, AZ, USA {ksluck, sstepput, hbenamor} ät asu.edu
[2]Google DeepMind, London, UK. {vec, jscholz} ät google.com

(a) Rand. initial position (b) Insertion started

Fig. 2: The experimental setup in which a Baxter robot has to insert a blue cylinder into a white tube (b). The cylinder is with a string attached to the end-effector of the robot. Camera images are recorded with the integrated end-effector camera. The sensor detecting the state of insertion is integrated into the white tube. Experiments on this platform were run fully autonomously without human intervention or simulations.

II. RELATED WORK

The advancement of deep reinforcement learning in recent years has lead to the development of a number of methods combining model-free and model-based learning techniques, in particular to improve the sample complexity of deep reinforcement learning methods. Nagabandi et al. [15] present a model-based deep reinforcement learning approach which learns a deep dynamic function mapping a state and action pair $(\mathbf{s}_t, \mathbf{a}_t)$ to the next state $\mathbf{s}_{t+1}$. The dynamics function is used to unroll a trajectory and to create an objective function based on the cumulative reward along the trajectory. This objective function is, then, used to optimize the actions along the trajectory and thereafter the first action is executed. The procedure is repeated whenever the next state is reached. After a dataset of executed trajectories is collected by the planning process, the policy of a model-free reinforcement learning algorithm is initialized in a supervised fashion by training it to match the actions produced by the planner. This technique is different to our approach in that we do not force the actor to match the executed action, but rather see it as an exploration from which we generate off-policy updates. Furthermore, it is implicitly assumed in [15] that a reward function is available for each state during the planning process. This can be a rather strong assumption, especially when learning in the real world without access to a simulation of the task and only providing minimal human supervision. Using executions in the environment during the planning process would be too costly since each change in state would require a re-execution of the whole trajectory. Since our insertion task provides only sparse rewards during execution, the trajectory planning algorithm would fail when relying only on rewards due to flat regions with zero reward and require additional reward engineering. This leaves a large and mostly flat region in the state space with a reward of zero.

In [2], Chua et al. introduce the model-based *probabilistic ensembles with trajectory sampling* method. This work builds upon [15], but also makes use of a reward function. It makes use of a probabilistic formulation of the deep dynamics function by using an ensemble of bootstrapped models encoding distributions to improve the sample complexity and improves the properties of the trajectory planner. Both approaches do not explicitly train an actor or a critic network.

Similarly to us, Universal planning networks [20] introduced by Srinivas et al. use a latent, gradient-based trajectory optimization method. However, the planner requires a goal state for the trajectory optimization. In certain tasks such as walking or running, it might be hard to acquire such a goal state to use in place of a velocity-based reward function. It is mentioned in [20] that to achieve walking, it was necessary to re-render images or reformulate the objective function by including an accessible dense reward function.

In contrast to previous work, we focus explicitly on the impact of using trajectory optimization as an additional technique for exploration and its impact on the learning process when used by a deep reinforcement learning algorithm such as Deep Deterministic Policy Gradient. Furthermore, using an actor-critic architecture is a key element in our work to allow off-policy updates in a fast manner during the training process and to inform the trajectory optimization process initially.

III. METHOD

The following sections introduce the different components used to generate explorative actions via trajectory optimization. We first describe the image embedding used, then the training process of the dynamics function and Deep Deterministic Policy Gradient (DDPG) [11], as well as its extension for the use of a value function. The section ends with a description of our trajectory optimization based exploration for DDPG.

A. Image Embedding

All tasks used throughout this paper are setup such that they use only images as observations, which have to be projected into a latent image embedding. This serves two main purposes: First, the number of parameters is greatly reduced since the actor, critic, and the dynamics network can be trained directly in the low dimensional latent space. Second, it is desirable to enforce temporal constraints within the latent image embedding, namely that subsequent images are close to each other after being projected into the latent space. Therefore, we make use of the recently introduced approach of time-contrastive networks [19]: the loss function enforces that the distance between latent representations of two subsequent images are small but the distance between two randomly chosen images is above a chosen threshold α. Enforcing a temporal constraint in the latent space improves the learning process of a consistent deep dynamics function in the latent space [19]. Time-contrastive networks make use

of two losses. The first is defined on the output of the decoder network and the input image as found in most autoencoder implementations. The second loss, the triplet loss, takes the latent representation $\mathbf{z}_t$ and $\mathbf{z}_{t+1}$ of two temporally close images and the latent representation $\mathbf{z}_r$ of a randomly chosen image.

Thus, given two temporal images Im_t and Im_{t+1} and a randomly chosen image Im_r, the loss functions for each element in the batch is given by

$$L(\text{Im}_t, \text{Im}_{t+1}, \text{Im}_r) = L_{\text{ae}}(\text{Im}_t) + L_{\text{contr}}(\text{Im}_t, \text{Im}_{t+1}, \text{Im}_r). \tag{1}$$

The classical autoencoder loss L_{ae} and the contrastive loss L_{contr} are here defined as

$$
\begin{aligned}
L_{\text{ae}} &= \parallel \text{Im}_t - \text{D}(E(\text{Im}_t)) \parallel, \\
L_{\text{contr}} (\text{Im}_t, \text{Im}_{t+1}, \text{Im}_r) &= \parallel E(\text{Im}_t) - E(\text{Im}_{t+1}) \parallel \\
&\quad + \max(\alpha - \parallel E(\text{Im}_t) - E(\text{Im}_r) \parallel, 0),
\end{aligned} \tag{2}
$$

with E being the encoder and D being the decoder network. The scalar value α defines the desired minimum distance between two random images in the latent embedding. Thus the classic autoencoder loss L_{ae} trains both the encoder and decoder network to learn a reconstructable image embedding. The contrastive loss L_{contr}, on the other hand, generates only a learning signal for the encoder network and places a temporal constraint on the image embedding. The encoder and decoder consist of three convolutional networks with a kernel shape of $(3, 3)$ and a stride of $(2, 2)$, followed by a linear layer of size 20 and an l2-normalized embedding which projects the states on a unit sphere [19]. All activation functions are rectified linear units (ReLU).
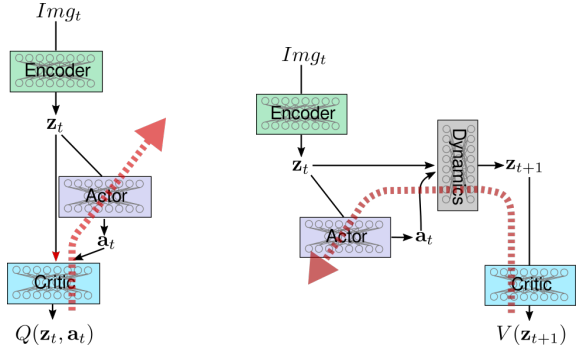
B. Latent Dynamics

Using a trajectory optimization algorithm in latent space requires a dynamics function which maps a latent state $\mathbf{z}_t$ and an action $\mathbf{a}_t$ to a subsequent latent state $\mathbf{z}_{t+1}$. This allows us to unroll trajectories into the future. In the case of a single image with $\mathbf{z}_t = E(\text{Im}_t)$, we learn a dynamics mapping of $\Psi(\mathbf{z}_t, \mathbf{a}_t) = \tilde{\mathbf{z}}_{t+1}$. In the other case, when our latent state is derived from several stacked images, then we project each image into the latent space, for example by

$$
\begin{bmatrix} E(\text{Im}_{t-2}) \\ E(\text{Im}_{t-1}) \\ E(\text{Im}_t) \end{bmatrix} = \begin{bmatrix} \mathbf{z}_t^{t-2} \\ \mathbf{z}_t^{t-1} \\ \mathbf{z}_t^t \end{bmatrix} = \mathbf{z}_t. \tag{3}
$$

To predict the next latent state, the dynamics function simply has to rotate the state and only predict the third latent sub-state. This function can be described with

$$
\mathbf{z}_t = \begin{bmatrix} \mathbf{z}_t^{t-2} \\ \mathbf{z}_t^{t-1} \\ \mathbf{z}_t^t \end{bmatrix} \mapsto \begin{bmatrix} \mathbf{z}_t^{t-1} \\ \mathbf{z}_t^t \\ \overline{\Psi}(\mathbf{z}_t, \mathbf{a}_t) \end{bmatrix} = \tilde{\mathbf{z}}_{t+1}, \tag{4}
$$

where $\overline{\Psi}$ is the output of the neural network while we will use the notation $\Psi(\mathbf{z}_t, \mathbf{a}_t) = \tilde{\mathbf{z}}_{t+1}$ for the whole operation, and $\tilde{\mathbf{z}}_{i+1}$ is the predicted next latent state. The loss function for the dynamics network is then simply the difference

(a) Q-Value based actor update   (b) Value based actor update

Fig. 3: The original DDPG algorithm (a) can be reformulated such that a value function (b) is used. In the case of a value function the policy gradient (red arrow) is computed via a neural dynamics function.

between the predicted latent state and the actual latent state. Therefore, the loss is given as

$$L_{\text{dyn}}(\text{Im}_{t-2:t}, \mathbf{a}_t, \text{Im}_{t+1}) = \parallel \overline{\Psi}(\mathbf{z}_t, \mathbf{a}_t) - E(\text{Im}_{t+1}) \parallel, \tag{5}$$

for each state-action-state triple $(\text{Im}_{t-2:t}, \mathbf{a}_t, \text{Im}_{t-1:t+1})$ observed during execution. The dynamics networks is constructed out of 3 fully connected layers of size 400, 400 and 20 with ReLUs as nonlinear activation functions.

C. Deep Reinforcement Learning

We make use of the Deep Deterministic Policy Gradient (DDPG) algorithm since action and state/latent space are continuous. DDPG is based on the actor-critic model which is characterized by the idea to generate a training signal for the actor (network) from the critic (network). In turn, the critic utilizes the actor to achieve an off-policy update and models usually a Q-value function. In DDPG, the actor is a network mapping (latent) states to an action with the goal of choosing optimal actions under a reward function. Hence, the loss function for the actor is given by

$$L_{\text{actor}}(\mathbf{z}_t) = -\text{Q}(\mathbf{z}_t, \pi(\mathbf{z}_t)), \tag{6}$$

where only the parameters of the actor $\pi(\mathbf{z}_t)$ are optimized (see Eq. 6 in [11]). In the case of classical DDPG, the critic is a Q-function network, which maps state and action pairs to a Q-value: $Q(\mathbf{z}_t, \mathbf{a}_t) = r(\mathbf{z}_t, \mathbf{a}_t) + \gamma Q(\mathbf{z}_{t+1}, \pi(\mathbf{z}_{t+1}))$. The scalar $gamma$ is a discount factor and $r(\mathbf{z}_t, \mathbf{a}_t)$ is the reward. The loss function of the critic network is based on the Bellman equation:

$$
\begin{aligned}
L_{\text{critic}}(\mathbf{z}_t, \mathbf{a}_t, r_{t+1}, \mathbf{z}_{t+1}) = \parallel \text{Q}(\mathbf{z}_t, \mathbf{a}_t) - \\
(r_{t+1} + \gamma \text{Q}'(\mathbf{z}_{t+1}, \pi'(\mathbf{z}_{t+1}))) \parallel,
\end{aligned} \tag{7}
$$

where Q' and π' are target networks. For more details on DDPG we refer the interested reader to [11]. It is worth noting that DDPG can be reformulated such that the critic resembles a value function instead of a Q-value function (Fig.
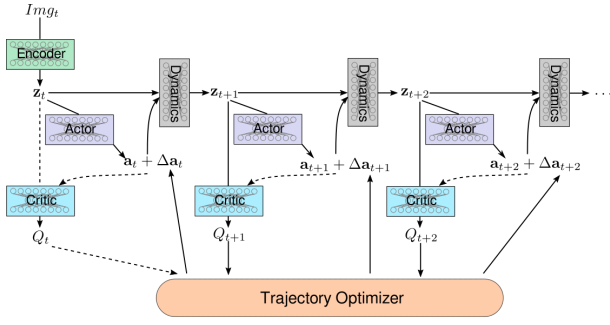
Fig. 4: The proposed exploration strategy unrolls the trajectory in the latent space and uses the Value/Q-Value to optimize the actions of the trajectory. Dotted connections might not be used when using a Value function as critic.

3, see also [5]). A naive reformulation of the loss function given above is

$$L_{\text{critic}}(\mathbf{z}_t, \mathbf{a}_t, r_t, \mathbf{z}_{t+1}) = \| V(\mathbf{z}_t) - (r_{t+1} + \gamma V'(\mathbf{z}_{t+1})) \|, \tag{8}$$

given an experience $(\mathbf{z}_t, \mathbf{a}_t, r_{t+1}, \mathbf{z}_{t+1})$. But this reformulation updates only on-policy and lacks the off-policy update ability of classical DDPG. Even worse, we would fail to use such a critic to update the actor since no action gradient can be computed due to the sole dependency on the state. However, since we have access to a dynamics function we reformulate for our extension of DDPG the loss function and incorporate off-policy updates with

$$\begin{aligned} L_{\text{critic}}(\mathbf{z}_t, \mathbf{a}_t, r_t, \mathbf{z}_{t+1}) = \| V(\mathbf{z}_t) \\ - (r_t + \gamma V'(\Psi(\mathbf{z}_t, \pi'(\mathbf{z}_t)))) \| . \end{aligned} \tag{9}$$

This formulation allows for off-policy updates given the experience $(\mathbf{z}_t, \mathbf{a}_t, r_t, \mathbf{z}_{t+1})$, for which we assume that the reward $r(\mathbf{z})$ is only state-dependent. While this might appear to be a strong assumption at first, it holds true for most tasks in robotics. The insertion task presented in the remainder of this paper is such a case in which the reward is fully described by the current position of both end-effector and the object to be inserted.

The loss function for the actor is then given with

$$L_{\text{actor}}(\mathbf{z}_t) = -V(\Psi(\mathbf{z}_t, \pi(\mathbf{z}_t))), \tag{10}$$

which is fully differentiable and, again, only used to optimize the parameters of the actor network. We use for both actor and critic two fully connected hidden layers of size 400 and 300 with ReLUs as nonlinear activation functions.

D. Optimized Exploration

Due to the deterministic nature of the actor network in DDPG and similar algorithms, the standard approach for exploration is to add random noise to actions. Random noise is usually generated from an Ornstein-Uhlenbeck process or a Gaussian distribution with fixed parameters. Such parameters, like the variance for a Gaussian distribution, are

usually chosen by intuition or have to be optimized as hyperparameter, for example with grid-search. In preliminary experiments we found Ornstein-Uhlenbeck processes with $\sigma = 0.5$ and $\theta = 0.15$ most effective on the chosen simulated task. In the presented approach we make use of the fact that we can access a dynamics function and therefore unroll trajectories throughout the latent space. The basic idea is to first unroll a trajectory using the actor network a number of steps into the future from the current point in time. We then optimize the actions $\mathbf{a}_t, \cdots, \mathbf{a}_{t+n}$ such that we maximize the Q-values/rewards along the latent trajectory. We characterize a latent trajectory, given a start state $\mathbf{z}_t = E(\text{Im}_t)$, as a sequence of state-action pairs $(\mathbf{z}_t, \mathbf{a}_t, \cdots, \mathbf{z}_{t+H}, \mathbf{a}_{t+H}, \mathbf{z}_{t+H+1})$. We can then formulate a scalar function to be maximized by the trajectory optimizer based on the Q-value or reward-functions available. This process is visualized in Fig. 4. The Q-function in the following equations can be substituted with a learned value function. An intuitive objective function to optimize is to simply sum up all Q-values for each state-action pair of the trajectory

$$f_Q(\mathbf{a}_{t:t+H}, \mathbf{z}_t) = w_0 Q(\mathbf{z}_t, \mathbf{a}_t) + \sum_{j=1}^{H} w_j Q(\mathbf{z}_{t+j}, \mathbf{a}_{t+j}), \tag{11}$$

with $\mathbf{z}_{t+j} = \Psi(\mathbf{z}_{t+j-1}, \mathbf{a}_{t+j-1})$ and $\mathbf{z}_t$ being the current state from which we start unrolling the trajectory. The time-dependent weight w_i determines how much actions are going to be impacted by future states and their values and can be uniform, linearly increasing or exponential. We consider in our experiments the special case of $w_i = \frac{1}{H}$. Alternatively, if one has access to a rewards function or learns a state-to-reward mapping simultaneously, then an objective function can be used which accumulates all rewards along the latent trajectory and adds only the final q-value:

$$f_{r+Q}(\mathbf{a}_{t:t+H}, \mathbf{z}_t) = \sum_{j=1}^{H-1} w_j r(\mathbf{z}_{t+j}) + w_H Q(\mathbf{z}_{t+H}, \mathbf{a}_{t+H}). \tag{12}$$

Clearly, this objective function is especially useful in the context of tasks with dense rewards. Both objective functions will be evaluated on the simulated cheetah task, which provides such dense rewards. While executing policies in the real world, we unroll a planning trajectory from the current state for n steps into the future. Then, the actions $\mathbf{a}_{t:t+H}$ are optimized under one of the introduced objectives from above with a gradient-based optimization method such as L-BFGS [27]. After a number of iterations of trajectory optimization, here 20, the first action of the trajectory, namely $\mathbf{a}_t$, is executed in the real world (Alg. 1).

IV. EXPERIMENTS

We compare in our experiments the classical approach of exploration in DDPG with an optimized Ornstein-Uhlenbeck process against the introduced approach of exploration through optimization. First, an experiment in simulation was conducted using the DeepMind Control Suite [23]. The

Algorithm 1 Exploration through trajectory optimization in DDPG

Require: Horizon H, Encoder network
  for number of episodes **do**
    while end of episode not reached **do**
      Compute latent state $\mathbf{z}_t$ from images with encoder
      Initialize action with $\mathbf{a}_t = \pi(\mathbf{z}_t)$
      if training **then**
        for $k = t+1 : t + H$ **do**
          Initialize action with $\mathbf{a}_k = \pi(\mathbf{z}_k)$
          Predict latent state $\mathbf{z}_{k+1} = \Psi(\mathbf{z}_k, \mathbf{a}_k)$
        end for
        Optimize $\max_{\mathbf{a}_{t:t+H}} f(\mathbf{a}_{t:t+H}, \mathbf{z}_t)$
      end if
      Execute step in environment with action $\mathbf{a}_t$
      Store $(\mathbf{z}_t, \mathbf{a}_t, r_t, \mathbf{z}_{t+1})$ in replay buffer
    end while
    Optimize dynamics network
    Optimize actor network
    Optimize critic network
    Update target networks
  end for

---

cheetah task, in which a two-dimensional bipedal agent has to learn to walk, is especially interesting because it involves contacts with the environment that makes the dynamics hard to model. In the second experiment, we evaluate the algorithms directly on a robot and aim to solve an insertion task in the real world.

A. Evaluation in Simulation on the Cheetah Task

The cheetah environment of the DeepMind control suite [23] has six degrees-of-freedom in its joints and we only use camera images as state information. The actions are limited to the range of $[-1, 1]$ and camera images are of the size $320 \times 240\ px$ in RGB and were resized to $64 \times 64\ px$. Each episode consists of 420 time steps and actions are repeated two times per time step. First, a dataset of 50 representative episodes was collected through the use of DDPG on the original state space of joint positions, joint velocities, relative body pose and body velocity of cheetah. This dataset was used to train the time-contrastive autoencoder as described above. The same parameters for the neural encoder were use for all exploration strategies. This was done to allow the sole evaluation of the exploration strategies independently of the used embedding. Since cheetah is a quite dynamic task and rewards depend on the forward velocity, this velocity must be inferable from each state. Hence, we project three subsequent images $(\mathrm{Im}_{t-2}, \mathrm{Im}_{t-1}, \mathrm{Im}_t)$ down by using the encoder network and define the current state $\mathbf{z}_t$ as the three stacked latent states $\mathbf{z}_t = [\mathbf{z}_{t-2}, \mathbf{z}_{t-1}, \mathbf{z}_t]^T$. For each of the presented evaluations 25 experiments were executed and the mean and standard deviations of the episodic cumulative rewards are shown in Figures 5-8.

1) Comparison between Ornstein-Uhlenbeck and optimized exploration: As a first step we optimized the hyperpa-

TABLE I: The average success rate of insertion for policies trained by DDPG with standard Ornstein-Uhlenbeck exploration or trajectory optimization with varying planning horizons. The individual success rates for each experiment were computed over a window of 50 subsequent episodes of 500 executions total. The average success rates and standard deviations were then computed with the highest success rate achieved in each experiment. A total of five experiments were executed for each method.

Method	Avg. Success rate ($\pm std$)
Ornstein-Uhlenbeck Exploration	75.2% ($\pm 11.7\%$)
1 Step Planning Horizon	**93.2%** ($\pm$**5.2%**)
3 Steps Planning Horizon	91.6% ($\pm$**1.5%**)
5 Steps Planning Horizon	84.0% ($\pm 14.1\%$)
15 Steps Planning Horizon	84.4% ($\pm 9\%$)

rameter σ of DDPG and found that an Ornstein-Uhlenbeck process with $\sigma = 0.5$ and $\theta = 0.15$ achieve a better result for DDPG on this task than the variance of $\sigma = 0.2$ proposed in [11], especially in the early stages of the training process. A planning horizon of ten steps was used to generate the optimized noise. We make comparisons between the training process, in which we use the exploration strategies, and the test case, in which we execute the deterministic actions produced by the actor without noise. Throughout the training process we evaluate the current policy of the actor after each episode. The results are presented in Fig. 5.

2) Comparison between different planning horizons: The main hyperparameter for optimized noise is the length of the planning horizon. If it is too short, actions are optimized greedily for immediate or apparent short-term success; if it is too long, the planning error becomes too large. Figure 7 shows the optimized exploration strategy with three different step-sizes: one step, ten steps and 20 steps into the future from the current state.

3) Comparison between different objectives: We introduced two potential objective functions, based on Q-values (Eq. 11) and a mix of reward- and Q-function (Eq. 12). We compare both of these against another objective where we only optimize for the q-value of the very last state-action pair of the unrolled trajectory (Fig. 8).

B. Insertion in the real world

Fast exploration is especially important when tasks have to be solved in a real world environment and training needs to be executed on the real robot. An insertion task was set up in which a Baxter robot had to insert a cylinder into a tube where both training and testing were performed in the real world environment, without the use of simulation (Fig. 2) [1]. Cylinder and tube were 3D-printed. The cylinder was attached to the right end-effector of the robot with a string. The position control mode was used because there is a variable delay in the observations. Image observation were acquired from the end-effector camera of the Baxter robot via ethernet. The six dimensional actions are in the range

---
[1]A video of the experiment can be found here: https://youtu.be/rfZcUWnut5I

(a) Deterministic Policy (Q-Value)

(b) Exploration (Q-Value)

(c) Deterministic Policy (Value)
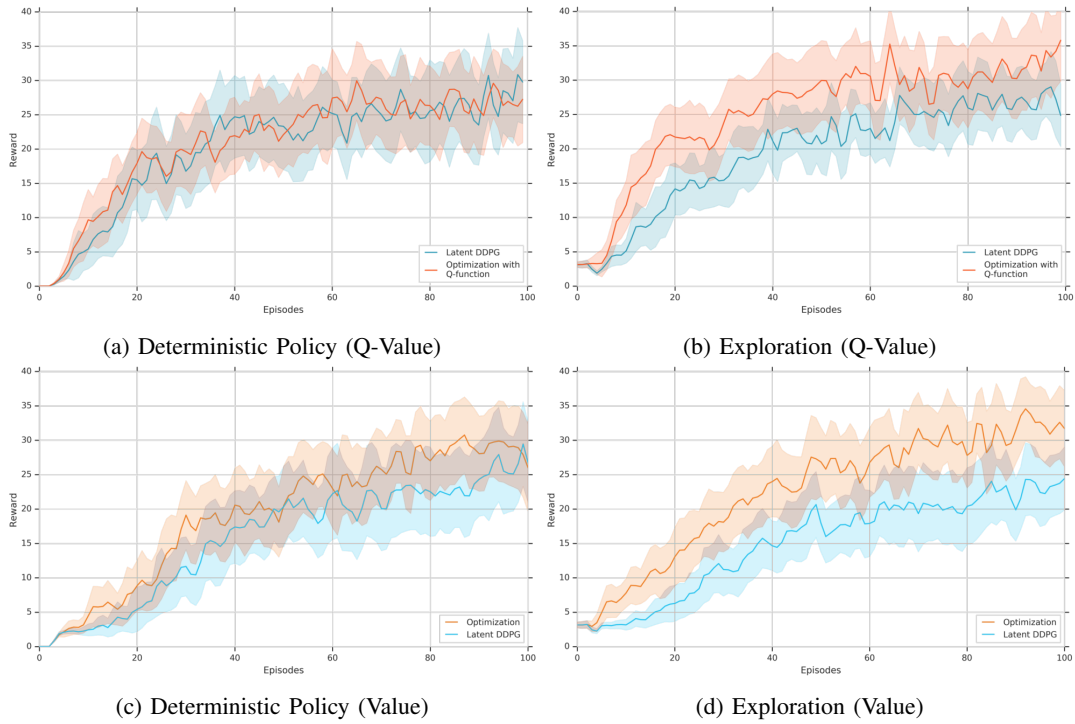

(d) Exploration (Value)

Fig. 5: Comparison between DDPG using exploration with optimization (orange) and classical exploration using an Ornstein-Uhlenbeck process (blue) on the simulated cheetah task. The exploitation graph shows the evaluation of actions produced by the deterministic actor while exploration strategies are applied during training.
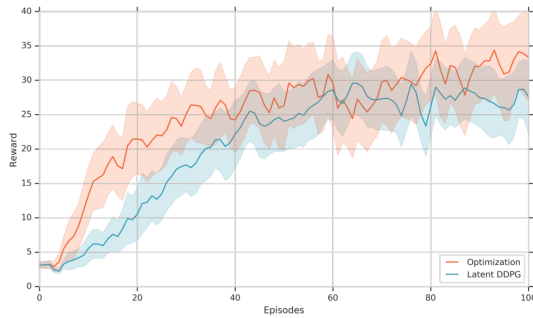

Fig. 6: Comparison between DDPG using exploration with optimization (orange) and classical exploration using an Ornstein-Uhlenbeck process (blue) on the simulated cheetah task while using a value function as critic. The number of training iterations per episode were raised from 1000 (Fig. 5-d) to 3000 for this evaluation.
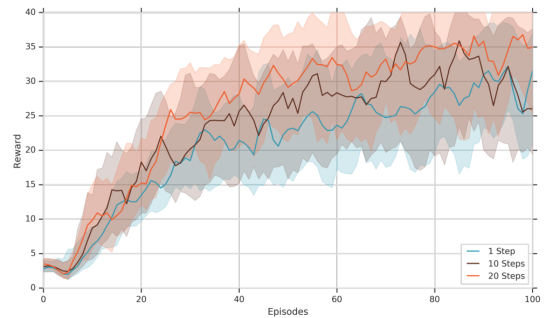

Fig. 7: Exploration through optimization evaluated with different horizons for the planning trajectory on the simulated cheetah task.
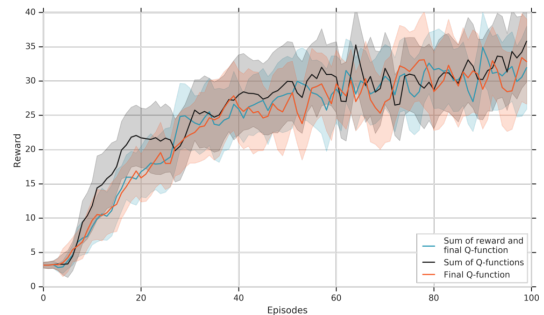

Fig. 8: Comparison between three different objective functions for optimized exploration on the simulated cheetah task.

of $[-0.05, 0.05]$ radians and represent the deviation for each joint of the arm at a point in time. This restriction ensures a strong correlation between subsequent camera images throughout the execution and allows the task to be solved in 20 steps. The initial position (radians) of the robot arm was randomized by sampling from a normal distribution with mean $\mu_{1:6} = (0.48, -1.23, -0.15, 1.42, 0.025, 1.35)$ and variances $\sigma_{1:6} = (0.05, 0.05, 0.05, 0.05, 0.05, 0.1)$, ensuring that the tube is in the image. As a simplification of the task, we excluded the last rotational wrist joint of the robot arm. Because of the adynamic nature of this task and the
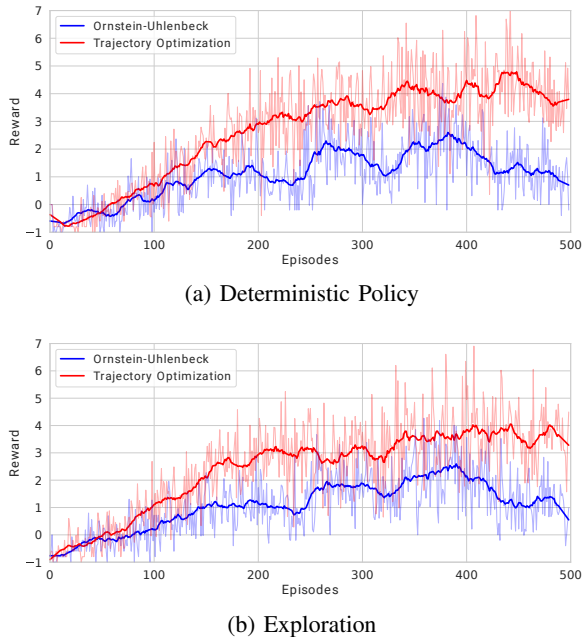
(a) Deterministic Policy

(b) Exploration

Fig. 9: Comparison between exploration with an Ornstein-Uhlenbeck (blue) and exploration through optimization (red) on the insertion task in the real world. The planning horizon is three steps. The figures show the cumulative rewards averaged over five experiments in light colours and in bold colours, for better interpretability due to the sparse reward, the mean smoothed with a Savitzky-Golay filter with window size 21 and 1st order polynomials.

necessity to use position control mode it is sufficient to use the latent representation of the current image versus a stack of images as in simulation. Larger movements of the cylinder appear as blur in the images. Each episode consists of 20 time steps and a sparse reward is used: For safety reasons, if the end-effector left the designated workspace area, the episode ended and a reward of -1 is assigned. When the cylinder is inserted into the tube, the extent of insertion is transformed into a reward from $[0, 1.0]$ and an episode stops if a reward of 0.9 or higher is assigned. The state of insertion is measured with a laser-based time-of-flight sensor (VL6180). The reward for all other possible states is zero. Five experiments were conducted on the robot: DDPG with a value function as critic and Ornstein-Uhlenbeck exploration, DDPG with exploration using trajectory optimization and a varying planning horizon (1, 3, 5 and 15 steps). We use a reduced planning horizon in this task due to the low number of time steps per episode. The comparison between Ornstein-Uhlenbeck exploration and optimized exploration with a horizon of three is shown in Fig. 9. Every episode which ends with a negative cumulative reward violated the workspace boundaries and episodes reaching a reward of 0.9 or more were successful insertions. Table I shows the comparison between exploration with Ornstein-Uhlenbeck noise and using planning horizons of different lengths in

terms of successful insertions. Each experiment was repeated five times and the cumulative reward for each episode is used to compute the mean shown in Figure 9. For better interpretability, the figures show, in bold lines, additionally a smoothed version of the mean where a Savitzky-Golay filter was applied with a window size of 21 and polynomials of order one. The autoencoder network as well as the dynamics network were trained with a demonstration dataset of 50 trajectories. Of these, 19 were positive demonstrations, in which the cylinder was successfully inserted. At the beginning of each training process, 5 of these 19 trajectories were added to the replay buffer to ensure convergence of the training process due to the difficulty of the task caused by using sparse reward.

V. DISCUSSION

We start with a discussion of the results from the simulated bipedal cheetah task which uses a dense reward function: The first insight is that both actors seem to perform equally well after 20 episodes, with the actor trained with optimized noise outperforming classical DDPG throughout the first 20 episodes (Fig. 5 (a)). However, during training the optimized exploration does not only perform better than exploration with an Ornstein-Uhlenbeck process (Fig. 5 (b)) but also performs better than the actions produced by both actors during test time (Fig. 5 (a)).

We found that using a critic network modelling the Q-function (Fig. 5 (b)) outperformed the formulation of DDPG using a value network when using optimized exploration (Fig. 5 (d)), while DDPG with Ornstein-Uhlenbeck noise performs slightly better with a Value network (Fig. 5 (a,d)). One could argue, that the effects of using optimized noise could vanish when increasing the number of trainings per episode, giving DDPG more time to find an optimal actor given the current training set. Following this line of thought we increased the number of training iterations per episode three times to 3000 (Fig. 6). The evaluation shows that while DDPG with OU noise improves in the later stages of the learning process, the trajectory optimization uncovers valuable training experience now much faster early on. This strongly indicates that the data distribution generated by the exploration strategy has an impact on the performance of DDPG. Evaluating the step-lengths we could find that trajectory optimization improved up to a planning horizon of 20 steps, although we opted for our experiments with a conservative planning horizon of 10 steps to reduce the overall training time. The evaluation of the three introduced objective functions show that the summation of Q-values along the planning trajectory yields better performance in the early training stages, up to episode 25, for the dense reward task (Fig. 8). This is an interesting result given that many other trajectory optimization approaches use a Bellman-inspired sum of weighted rewards [15], [2]. It is also worth to notice that the Q-Value is the more suitable objective function for optimizing actions in the presented real-world insertion task due to the reward function being zero for the majority of time steps.

The results showing the learning progress on the insertion task in the real world draw a clearer picture of the benefit of exploration through optimization (Fig. 9, Table I). Generally, after roughly 50 training episodes, the networks trained with optimized exploration outperformed DDPG with OU and also achieved higher rewards in later stages of the learning process (Fig. 9). An evaluation of the length of the planning horizon shows, as expected, that longer planning horizons lead to a decreases performance (Table I). This is very likely due to the accumulating error of predicted future states from the dynamics network. However, even with longer planning horizons the presented approach outperformed exploration using OU noise.

VI. CONCLUSION

This work investigated the possibility of combining an actor-critic reinforcement learning method with a model-based trajectory optimization method for exploration. By using trajectory optimization only to gain new experience, the ability of DDPG to learn an optimal policy is not affected and we can furthermore make use of DDPG's off-policy training ability. We were able to show that by using this strategy, a performance gain can be achieved, especially in the presented real world insertion task learned from images. It is worth noting that this performance gain can be mainly attributed to the change in exploration strategy since a fixed image embedding was used, reducing the possibility of performance differences caused by using different image embeddings. This work only considered using reward, Q-Value or value functions as objective functions for optimizing the latent trajectory. In future work we plan to investigate the possibility of using additional cost terms, eg. safety and state-novelty. Furthermore, another natural next step would be to use probabilistic dynamics networks and advanced trajectory optimization algorithms to evaluate their impact on deep reinforcement learning algorithms when used for exploration in this setup.

REFERENCES

[1] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *arXiv preprint arXiv:1810.05687*, 2018.

[2] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4759–4770, 2018.

[3] Adria Colome, Fabio Amadio, and Carme Torras. Exploiting symmetries in reinforcement learning of bimanual robotic tasks. *IEEE Robotics and Automation Letters*, 2019.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1856–1865, 2018.

[5] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[6] Maximilian Jaritz, Raoul De Charette, Marin Toromanoff, Etienne Perot, and Fawzi Nashashibi. End-to-end race driving with deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2070–2075. IEEE, 2018.

[7] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. *arXiv preprint arXiv:1807.00412*, 2018.

[8] Jens Kober and Jan R Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009.

[9] Fengming Li, Qi Jiang, Sisi Zhang, Meng Wei, and Rui Song. Robot skill acquisition in assembly process using deep reinforcement learning. *Neurocomputing*, 2019.

[10] Tianyu Li, Akshara Rai, Hartmut Geyer, and Christopher G Atkeson. Using deep reinforcement learning to learn high-level policies on the atrias biped. *arXiv preprint arXiv:1809.10811*, 2018.

[11] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[12] Kevin Sebastian Luck and Heni Ben Amor. Extracting bimanual synergies with reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4805–4812. IEEE, 2017.

[13] Kevin Sebastian Luck, Joseph Campbell, Michael Andrew Jansen, Daniel M. Aukes, and Heni Ben Amor. From the lab to the desert: Fast prototyping and learning of robot locomotion. In *Robotics: Science and Systems*, 2017.

[14] Kevin Sebastian Luck, Gerhard Neumann, Erik Berger, Jan Peters, and Heni Ben Amor. Latent space policy search for robotics. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1434–1440. IEEE, 2014.

[15] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[16] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[17] Gautam Reddy, Jerome Wong-Ng, Antonio Celani, Terrence J Sejnowski, and Massimo Vergassola. Glider soaring via reinforcement learning in the field. *Nature*, 562(7726):236, 2018.

[18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[19] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018.

[20] A Srinivas, A Jabri, P Abbeel, S Levine, and C Finn. Universal planning networks. In *International Conference on Machine Learning (ICML)*, 2018.

[21] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[22] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pages 2753–2762, 2017.

[23] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[24] Russ Tedrake, Zack Jackowski, Rick Cory, John William Roberts, and Warren Hoburg. Learning to fly like a bird. In *14th International Symposium on Robotics Research. Lucerne, Switzerland*. Citeseer, 2009.

[25] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[26] Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[27] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.