Sample-Efficient Policy Search with a Trajectory Autoencoder

Alexander Fabisch[*]
Robotics Innovation Center, DFKI GmbH

Frank Kirchner
Robotics Innovation Center, DFKI GmbH
Robotics Research Group, University of Bremen

Abstract

We introduce a trajectory generator that can be used to perform sample-efficient policy search with Bayesian optimization (BO). BO is a sample-efficient approach to direct policy search that usually does not scale well with the number of parameters. Our trajectory generator is able to map a compact representation of trajectories to a high-dimensional trajectory space so that BO can search in the low-dimensional space. The trajectory generator will be trained as part of a variational autoencoder on demonstrations from an expert. The trajectory generator contains a trajectory layer, which is a new building block for neural networks that enforces smoothness on generated trajectories. We evaluate our approach with grasping on a real robot.

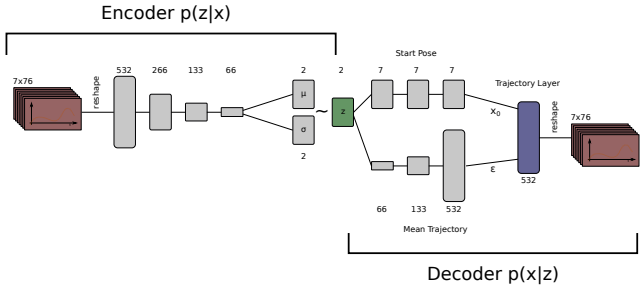
1 Introduction

When planning or control are not applicable because of uncertainty and noise or unavailability of an accurate model, implementing new behaviors for robots can be difficult. We could then use machine learning — reinforcement learning (RL, [24]) or imitation learning (IL, [20]). These methods come with drawbacks if applied to robots in the real world. Despite recent breakthroughs in RL [25, 22, 19] it is often unstable and not sample efficient enough to learn directly on real robots, which is amplified by the curse of dimensionality caused by increasing complexity of sensor data and kinematics. We discuss a setting in which we have one sensor measurement from which we predict a sequence of commands. Similar problems like playing table tennis [12] have often been handled with policy search and domain-specific policy representations such as dynamical movement primitives (DMPs, [7]). These methods mostly use linear or non-parametric models and are stable and sample efficient. Bayesian optimization for contextual policy search [18] is a prime example of a sample-efficient policy search algorithm that generalizes over task parameters (contexts), however, it requires policies with only a few parameters, which we obtain through expert demonstration and manifold learning.

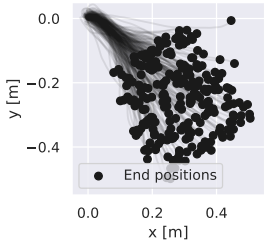
Tailored policy representations such as movement primitives are popular in robotics [13]. The most prominent examples are dynamical movement primitives (DMPs, [7]), which are suited for imitation and RL and generate state-space trajectories $\boldsymbol{x}_{t+1} = \pi_{\boldsymbol{v},\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$, with $\boldsymbol{v} = (\boldsymbol{x}_0, \boldsymbol{g}, \tau)$, where $\boldsymbol{x}_t$ is the state (position, velocity, and acceleration) at time t, $\boldsymbol{\theta}$ are the parameters and $\boldsymbol{v}$ are the metaparameters ($\boldsymbol{x}_0$: initial state; $\boldsymbol{g}$: final state; τ: duration of the movement). Trajectory-based policy representations like DMPs only generalize over their metaparameters. However, we can parameterize them based on the context, i. e., we can learn a mapping from context $\boldsymbol{s}$ to policy parameters $\boldsymbol{\theta}$. In contextual policy search (CPS) we seek to optimize $\arg\max_{\boldsymbol{\omega}} \int_{\boldsymbol{s}} p(\boldsymbol{s}) \int_{\boldsymbol{\theta}} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s}) \mathbb{E}\left[R(\boldsymbol{\theta}, \boldsymbol{s})\right] d\boldsymbol{\theta} d\boldsymbol{s}$, where $\boldsymbol{s} \in S$ is a context, $\pi_{\boldsymbol{\omega}}$ is a stochastic upper-level policy parameterized by $\boldsymbol{\omega}$ that defines a distribution of policy parameters for given contexts [3]. Context can also be interpreted as a parameter of the task. The return R is extended to take into account the context. During the learning process, we optimize $\boldsymbol{\omega}$, observe the current context $\boldsymbol{s}$, and select $\boldsymbol{\theta}_i \sim \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$. CPS algorithms are often based on policy search [15, 1, 18]. We are particularly interested in BO-CPS [18], since Gutzeit et al. [4] show

[*]Correspondence to: alexander.fabisch@dfki.de

NeurIPS 2021 Workshop on Robot Learning: Self-Supervised and Lifelong Learning, Virtual, Virtual



(a) Architecture of the variational trajectory autoencoder (VTAE). The encoder reduces the dimensionality of trajectories. The decoder generates trajectories from a low-dimensional representation. The decoder has a special structure that splits the initial state from the shape of the trajectory. Both are combined in the trajectory layer.



(b) Demonstrated grasps projected to x-y plane. Lines indicate trajectories and circles mark end positions of these.

Figure 1: Trajectory autoencoder and corresponding trajectory dataset.

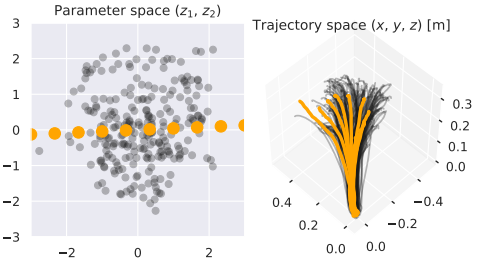
that BO-CPS can efficiently learn throwing on a real robot if the parameter space is reduced to 2D. However, the dimensions were selected manually and we would like to determine relevant parameters automatically. Hence, we apply manifold learning to figure out which latent variables are best to cover the distribution of expert demonstrations.

This work is inspired by Matsubara et al. [17] who introduce stylistic DMPs (SDMPs). SDMPs reduce the dimensionality of DMPs learned from multiple demonstrations via SVD to identify and represent multiple styles of similar motions. However, the shortcomings are that SDMPs are deeply coupled with DMPs and an SVD only allows linear transformations. Rueckert et al. [23] present a similar extension of ProMPs with hierarchical priors to learn a low number of control parameters from multiple demonstrations. Our work is similar as we learn a low-dimensional representation of trajectories from expert demonstrations, however, we do not need an indirect trajectory representation. As black-box optimization and policy search are closely connected, Bayesian optimization can be used for policy search but it is difficult to scale to many parameters. Therefore, parameter reduction has been used in high-dimensional spaces before: Wang et al. [26] use random embeddings to solve high-dimensional problems with low intrinsic dimensionality. Using fixed low-dimensional representations of high-dimensional policies has also been proposed for neuroevolution [14]. Our approach uses a low-dimensional representation learned from data instead.

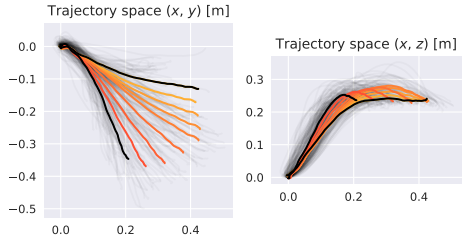
2 Variational Trajectory Autoencoder (VTAE)

We use a variational autoencoder (VAE, [11]) to learn a trajectory encoding. Figure 1(a) shows our proposed architecture. We are only interested in the generative model $p(\boldsymbol{X}|\boldsymbol{z})$ after training, where $\boldsymbol{X}$ is a trajectory and $\boldsymbol{z}$ a latent vector. The generator creates trajectories without indirection through movement primitives. For this purpose we develop a new layer that ensure smoothness. This layer does not contain learnable parameters and can be added at the end of a neural network that should output trajectories. Thereby, we integrate prior knowledge in the structure of the neural network. The trajectory layer is linear and does not restrict the capacity of the network.

Trajectories executed by humans are usually smooth and so should the robot's trajectories be. We use a trick presented by Kalakrishnan et al. [9] to generate smooth and dense trajectories of end-effector poses with a layer of a neural network. The trajectory layer implements a function $\boldsymbol{X} = g(\boldsymbol{\epsilon}, \boldsymbol{x}_0)$, where $\boldsymbol{\epsilon} \in \mathbb{R}^{T \times D}$ and $\boldsymbol{x}_0 \in \mathbb{R}^D$ are inputs from previous layers. $\boldsymbol{x}_0$ is the initial state of the trajectory. $\boldsymbol{X} \in \mathbb{R}^{T \times D}$ is a trajectory of T steps in D dimensions. In practice, we often predict multiple trajectories at the same time, e. g., when we compute the gradient of a batch during training. The layer performs a matrix multiplication $\boldsymbol{L}\boldsymbol{\epsilon}_{\bullet d}$ for each column $\boldsymbol{\epsilon}_{\bullet d}$ of $\boldsymbol{\epsilon}$ to compute trajectory offsets in each dimension $d \in \{1, \ldots, D\}$ that will be added to the initial state $\boldsymbol{x}_{0d}$ of that dimension. Hence, the layer performs only linear operations. We will now define $\boldsymbol{L}$. Second order backward differences for a sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)^T$ and a temporal difference of Δt are computed as $\ddot{x}_t \approx [x_t - 2x_{t-1} + x_{t-2}] / \Delta t^2$, which can be written as matrix multiplication $\ddot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$, where



(a) Latent space interpolation along feature axis 1. Training set is shown in the background.



(b) Latent space interpolation between samples. Projection on x-y plane (left) and x-z plane (right).

Figure 2: Interpolations in latent space.

$\boldsymbol{A} \in \mathbb{R}^{(n+2)\times n}$, to obtain the sequence of accelerations $\ddot{\boldsymbol{x}}$ from $\boldsymbol{x}$. We want to find a covariance matrix of a multivariate Gaussian that generates trajectories with low acceleration according to the quadratic cost $c(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{x} = \ddot{\boldsymbol{x}}^T \ddot{\boldsymbol{x}}$. Kalakrishnan et al. [9] propose $(\boldsymbol{A}^T \boldsymbol{A})^{-1}$ as a covariance matrix $\mathcal{N}(\boldsymbol{x}|\boldsymbol{0}, (\boldsymbol{A}^T \boldsymbol{A})^{-1})$ in the context of motion planning for manipulation. Sampled trajectories start at zero and end at zero. To generate a distribution that diverges from zero in the end we can use the upper left quarter of this matrix. We can reparameterize a standard normal distribution $\boldsymbol{y} \sim \mathcal{N}(0, \boldsymbol{I})$ to output these trajectories by $\boldsymbol{x} = \boldsymbol{L}\boldsymbol{y}$. $\boldsymbol{L}$ is obtained by Cholesky decomposition $\boldsymbol{L}\boldsymbol{L}^T = (\boldsymbol{A}^T \boldsymbol{A})^{-1}$. We apply this trick to the output of a neural network that transforms a Gaussian distribution and we extend it to multi-dimensional space.

We use a VAE [11] to train the trajectory generator. Its architecture is displayed in Figure 1(a). We use dense layers with nonlinear activation functions. We found that a leaky ReLU gives better results than tanh or ReLU (refer to [16] for definitions). Our decoder has a special structure that includes the trajectory layer, which needs two inputs $\boldsymbol{\epsilon}$ and $\boldsymbol{x}_0$. These will be generated by two separate paths in the decoder from the latent vector $\boldsymbol{z}$. We use a modified loss that is similar to the β-VAE [5]. Although Higgins et al. [5] suggest to use $\beta > 1$ to learn a better disentanglement of the data, we choose $0 < \beta < 1$ to generate trajectories that are closer to the demonstrated trajectories. This configuration has been investigated before [6].

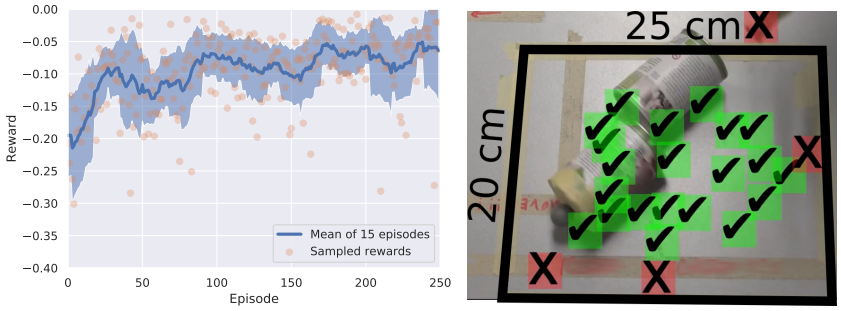
3 Experiments

We investigate these questions: (1) Can a VTAE learn a manifold, in which we can smoothly interpolate between trajectories that are similar to demonstrations? (2) Can we use BO-CPS to learn a mapping from contexts to parameters in the latent space? (3) Can we do this on a real robot?

Dataset: We recorded 249 pick and place movements from one person with XSens MVN Awinda, which is a motion capture system based on inertial measurement units. The person had to pick a small cylindrical object from various positions on a table. We are only interested in grasping motions and manually extracted those. Each trajectory consists of 76 steps at a frequency of 60 Hz. We only use the end-effector pose since the positions of the fingers are completely different from the positions of the fingers on the robot. Figure 1(b) displays all demonstrated trajectories.

Interpolation: We explore the learned projection from the latent space to trajectory space. Figure 2(a) shows interpolation along an axis indicated by the orange dots. In Figure 2(b) we select two samples from the training set that are clearly different in trajectory space, project them to latent space, interpolate in latent space and project the interpolated latent variables back to trajectory space. We can see that we can identify axes in latent space that have different effect on the shape of trajectories. We can also smoothly interpolate between trajectories.

CPS on Real Robot: After confirming the sample efficiency of BO-CPS in a simulated reaching task (see Appendix A.3), we use a UR5 robot arm and a Robotiq 2F-140 gripper to perform RL experiments. BO-CPS will generalize grasping to a predefined area (see Figure 3(b)) by generating an upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$, where $\boldsymbol{\theta}$ are latent vectors that will be projected to trajectories by the decoder of the VTAE. The object that will be grasped is a can. We will see that the generated trajectories are easily executable by a robot arm as they are smooth. We perform 250 episodes, which



(a) Learning curve of CPS on UR5. (b) Successful and failed grasp attempts.

Figure 3: Results of CPS on UR5.

is a good compromise between minimizing episodes on the real robot on the one hand and reaching a good performance on the other hand. With a marker-based motion capture system we measure the pose of the gripper and learn to reach points. The robot does not receive any information other than the target location (context) and the reward, which is the negative distance between the gripper's center and the target. The context is sampled uniformly from an area of the size $0.2m \times 0.25m$ during training. The learning curve is displayed in Figure 3(a). Each reward corresponds to a different context, hence, we filter the learning curve by plotting mean and standard deviation of 15 episodes. Figure 3(b) shows the performance of the final policy. We see that can locations close to the border or outside of the context area result in failures. However, it is possible to successfully grasp objects in the middle of the target area. It is often failing to grasp the can just by a centimeter at the border of the target area.

4 Discussion

We use manifold learning to train a low-dimensional representation of grasps and demonstrate that this can be combined with CPS algorithms to efficiently learn grasping on a real robotic system. The approach, however, is not limited to this learning paradigm and we would also like to build a bridge to deep RL, which learns nonlinear, parametric models but often assumes a tight sensor-actuator coupling. The decoder of our VTAE could also be used with policy gradient algorithms.

Acknowledgments and Disclosure of Funding

We thank Marc Tabie and Manuel Meder for their feedback. This work was supported by the German Federal Ministry for Economic Affairs and Energy (BMWi, FKZ 50RA1703 and 50RA1701) and a grant from the European Commission (870142).

References

[1] A. Abdolmaleki, B. Price, N. Lau, L.P. Reis, and G. Neumann. Contextual covariance matrix adaptation evolutionary strategies. In *IJCAI*, pages 1378–1385, 2017.

[2] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited-memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16:1190–1208, 1995.

[3] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1–2):328–373, 2013.

[4] Lisa Gutzeit, Alexander Fabisch, Marc Otto, Jan Hendrik Metzen, Jonas Hansen, Frank Kirchner, and Elsa Andrea Kirchner. The besman learning platform for automated robot skill learning. *Frontiers in Robotics and AI*, 5:43, 2018.

[5] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[6] Matt Hoffman, Carlos Riquelme, and Matthew Johnson. The β-vae's implicit prior. In *Bayesian Deep Learning, NeurIPS Workshop*, 2017.

[7] A. Ijspeert, J. Nakanishi, P Pastor, H. Hoffmann, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Comput.*, 25(2):328–373, 2013.

[8] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

[9] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *ICRA*, pages 4569–4574, 2011.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[11] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2014.

[12] J. Kober, K. Mülling, O. Krömer, C. H. Lampert, B. Schölkopf, and J. Peters. Movement templates for learning of hitting and batting. In *ICRA*, pages 853–858, 2010.

[13] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 2013.

[14] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Conference on Genetic and Evolutionary Computation*, GECCO '13, page 1061–1068, New York, NY, USA, 2013. Association for Computing Machinery.

[15] Andras G. Kupcsik, Marc Peter Deisenroth, Jan Peters, and Gerhard Neumann. Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2013.

[16] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[17] T. Matsubara, S. Hyon, and J. Morimoto. Learning stylistic dynamic movement primitives from multiple demonstrations. In *IROS*, pages 1277–1283, 2010.

[18] Jan Hendrik Metzen, Alexander Fabisch, and Jonas Hansen. Bayesian optimization for contextual policy search. In *IROS Workshop: Machine Learning in Planning and Control of Robot Motion (MLPC)*, 2015.

[19] OpenAI. Learning dexterous in-hand manipulation. *International Journal of Robotics Research*, 39(1):3–20, 2020.

[20] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 7(1-2):1–179, 2018.

[21] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

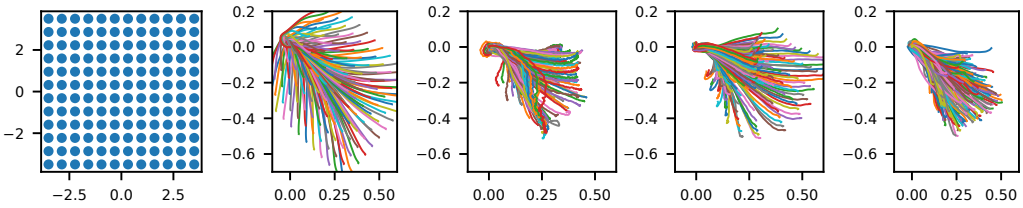
[22] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4), 2018.

[23] E. Rueckert, J. Mundo, A. Paraschos, J. Peters, and G. Neumann. Extracting low-dimensional control variables for movement primitives. In *ICRA*, pages 1511–1518, 2015.

[24] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[25] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[26] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando De Freitas. Bayesian optimization in a billion dimensions via random embeddings. *J. Artif. Int. Res.*, 55(1):361–387, 2016.



(a) Latent space grid. (b) PCA. (c) VAE. (d) VTAE. (e) Training set.

Figure 4: Projection of grid in latent space to trajectory space with 3 manifold learning algorithms: PCA, standard VAE, and VTAE. Trajectories are projected on the x-y plane. Each colored line corresponds to one trajectory that is projected from one point of the grid in latent space. For comparison, we also show all trajectories from the training set.

A Experiments

A.1 Training and Hyperparameters of Variational Trajectory Autoencoder

In this paper, we reduce the number of dimensions to two to restrict the search space for policy search and increase sample efficiency. We explored several architectures for the decoder. In the initial architecture we had an additional layer with 266 nodes as the third layer of the decoder. We found this made the mapping from latent space to trajectory space too complex so that interpolation between trajectories in latent space was not smooth enough for contextual policy search. The decoder has to be nonlinear to represent the demonstrated distribution, but we have to trade off between the capacity to reconstruct the original dataset and the smoothness of interpolation between trajectories. One way to do this is to tune β. Alternatively, we design the decoder's architecture carefully.

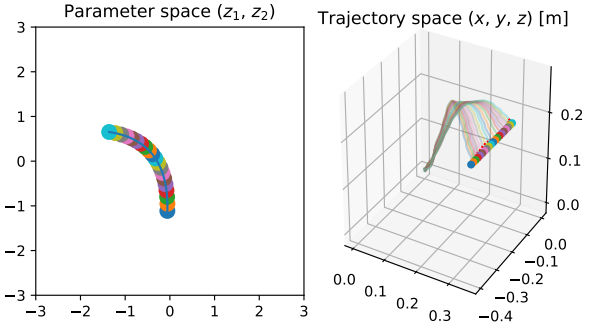
We set $\beta = 0.1$ as a compromise between fitting the dataset accurately and approximating a Gaussian distribution in the latent space. We tried $\beta \in [0, 0.01, 0.1, 1, 5]$. We train the VAE with a batch size of 16 for 500 epochs with Adam [10]. All samples from the training set are used 500 times. We also trained for 50,000 epochs but did not see relevant progress after 500 epochs. We explored smaller and larger batch sizes and found that a batch size of 16 achieves the lowest loss. Training was done either on an Intel i7-5960X CPU, or on an NVidia Titan X, which was twice as fast.

A.2 Comparison of Manifold Learning Approaches

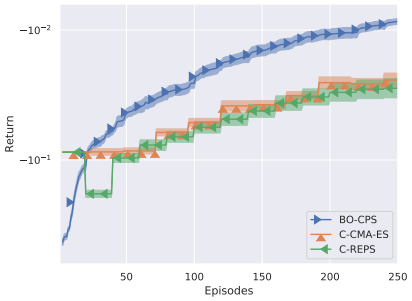
Figure 4 compares 3 manifold learning approaches to encode the grasp dataset. We generate a grid with 144 points in latent space and project it with a principal component analysis (PCA, [21]), a variational autoencoder (VAE), and a variational trajectory autoencoder (VTAE). PCA is a linear model and does not capture the distribution of the training set accurately as there are many trajectories that are not in the training set. This extrapolation might be desirable, for instance, in cases with only a few demonstrations but it is not in our case where we want to guide policy search through demonstrations. The VAE uses the same architecture as the VTAE for decoding $\boldsymbol{\epsilon}$, however, we can see that the trajectories are more shaky than for the other models. This is a property that is not desirable for trajectories that should be executed on a robot. The VTAE is a good compromise between capturing the distribution and smoothness.

A.3 Simulated Reaching Task

We check whether CPS algorithms are able to exploit the latent space to do sample-efficient policy search and which algorithm works best. For this purpose we will use a simple reaching task: goal positions for the end effector are located on a line in 3D space. Goals $\boldsymbol{s} = (g_x, g_y)^T$ are varied along one axis, i. e., $g_x = 0.35$ and $g_y \in [-0.4, -0.15]$. We use them as context for CPS and to define the reward, which is the negative distance of the end effector's center after executing the trajectory to the goal. In this experiment, CPS algorithms learn an upper-level policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}|\boldsymbol{s})$, where $\boldsymbol{\theta}$ are latent vectors that will be projected to trajectories by the decoder of the VTAE. Figure 5(a) shows a solution to the problem that has been obtained by BO-CPS after 250 episodes. Note that we only optimize the position distance to the goal. We do not penalize accelerations or velocities. Nevertheless, all trajectories are smooth. That is even the case during the learning process. Figure 5(b) shows learning curves of the 3 algorithms. BO-CPS outperforms the others, however, its computational complexity depends on the number of samples that were explored. This makes it slow after more than the 250 episodes that were needed. Hence, it is not an optimal algorithm for learning in simulation. Nevertheless, for learning in reality this is perfectly fine because learning in the real world with a robot is a tedious task if it cannot be fully automated. Hence, we prefer sample-efficiency over low computational complexity.



(a) Example of final policy for the reaching problem. Goals are represented by circles in the right plot. Corresponding trajectories are indicated by lines. The corresponding parameters in latent space are displayed in the left plot in matching color. A quadratic mapping from goal positions to trajectories has been learned.



(b) Comparison of CPS algorithms in latent space. Mean and standard error over 20 experiments are displayed.

Figure 5: Simulated reaching problem.

A.4 Hyperparameters of CPS Algorithms

We compare BO-CPS, C-REPS, and C-CMA-ES. The search space for all algorithms is restricted to $[-3, 3] \times [-3, 3]$ in the latent space. The initial variance of C-CMA-ES and C-REPS is set to 5. We use a quadratic upper-level policy that maps from context $\boldsymbol{s}$ to trajectory parameters $\boldsymbol{z}$. Although BO-CPS is non-parametric, we learn a quadratic policy from all samples at test time to speed up querying parameters for given contexts. The quadratic policy is obtained in the same way as in C-REPS with all samples. After hyperparameter tuning we update the upper-level policy in C-REPS after 20 episodes with a history of 100 samples and use C-CMA-ES' default values for these parameters. We tried updates after $[5, 10, 20, 30, 50]$ samples and a training set size of $[5, 10, 20, 30, 50, 100, 200, 250]$ for both algorithms but did not find better configurations. For BO-CPS we use Gaussian process regression as surrogate model. We designed the kernel $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = c_1 M_\nu(\boldsymbol{x}_1, \boldsymbol{x}_2) + W(\boldsymbol{x}_1, \boldsymbol{x}_2) + c_2$, where M_ν is a Matérn kernel with an anisotropic length scale initialized at 1 and limited to $[0.01, 100]$ and the parameter ν is set to 1.5. c_1, c_2 are constant kernels that are initialized to 100 and are limited to $\left[10^{-3}, 10^5\right]$ and $\left[10^{-2}, 10^5\right]$ respectively. W is a white noise kernel initialized at a noise level of 1 and limited to $\left[10^{-5}, 10^5\right]$. We did not invest much time in tuning the kernel hyperparameters. We use upper confidence bound as acquisition function with the parameter κ to control exploration and exploitation. $\kappa = 2$ is just enough to avoid too much exploitation (we tried $1, 1.5, 2$). In every query of the acquisition function we run 500 iterations of the global optimizer DIRECT [8] and L-BFGS-B [2] to convergence for fine tuning.

B Code

An implementation of the variational trajectory autoencoder with a simple example application is available at `https://github.com/AlexanderFabisch/vtae`.