Journal of Machine Learning Research 19 (2018) 1-25 Submitted 6/17; Revised 4/18; Published 8/18

Model-Free Trajectory-based Policy Optimization with Monotonic Improvement

Riad Akrour[1] RIAD@ROBOT-LEARNING.DE
Abbas Abdolmaleki[2] AABDOLMALEKI@GOOGLE.COM
Hany Abdulsamad[1] HANY@ROBOT-LEARNING.DE
Jan Peters[1,3] JAN@ROBOT-LEARNING.DE
Gerhard Neumann[1,4] GERI@ROBOT-LEARNING.DE
[1] *CLAS/IAS, Technische Universitäte Darmstadt, Hochschulstr. 10, D-64289 Darmstadt, Germany*
[2] *DeepMind, London N1C 4AG, UK*
[3] *Max Planck Institute for Intelligent Systems, Max-Planck-Ring 4, Tübingen, Germany*
[4] *L-CAS, University of Lincoln, Lincoln LN6 7TS, UK*

Editor: George Konidaris

Abstract

Many of the recent trajectory optimization algorithms alternate between linear approximation of the system dynamics around the mean trajectory and conservative policy update. One way of constraining the policy change is by bounding the Kullback-Leibler (KL) divergence between successive policies. These approaches already demonstrated great experimental success in challenging problems such as end-to-end control of physical systems. However, the linear approximation of the system dynamics can introduce a bias in the policy update and prevent convergence to the optimal policy. In this article, we propose a new model-free trajectory-based policy optimization algorithm with guaranteed monotonic improvement. The algorithm backpropagates a local, quadratic and time-dependent Q-Function learned from trajectory data instead of a model of the system dynamics. Our policy update ensures exact KL-constraint satisfaction without simplifying assumptions on the system dynamics. We experimentally demonstrate on highly non-linear control tasks the improvement in performance of our algorithm in comparison to approaches linearizing the system dynamics. In order to show the monotonic improvement of our algorithm, we additionally conduct a theoretical analysis of our policy update scheme to derive a lower bound of the change in policy return between successive iterations.

Keywords: Reinforcement Learning, Policy Optimization, Trajectory Optimization, Robotics

1. Introduction

Trajectory Optimization methods based on stochastic optimal control (Todorov, 2006; Theodorou et al., 2009; Todorov and Tassa, 2009) have been very successful in learning high dimensional controls in complex settings such as end-to-end control of physical systems (Levine and Abbeel, 2014). These methods are based on a time-dependent linearization of the dynamics model around the mean trajectory in order to obtain a closed form update of the policy as a Linear-Quadratic Regulator (LQR). This linearization is then repeated locally for the new policy at every iteration. However, this iterative process does

not offer convergence guarantees as the linearization of the dynamics might introduce a bias and impede the algorithm from converging to the optimal policy. To circumvent this limitation, we propose in this paper a novel model-free trajectory-based policy optimization algorithm (MOTO) couched in the approximate policy iteration framework. At each iteration, a Q-Function is estimated locally around the current trajectory distribution using a time-dependent quadratic function. Afterwards, the policy is updated according to a new information-theoretic trust region that bounds the KL-divergence between successive policies in closed form.

MOTO is well suited for high dimensional continuous state and action spaces control problems. The policy is represented by a time-dependent stochastic linear-feedback controller which is updated by a Q-Function propagated backward in time. We extend the work of (Abdolmaleki et al., 2015), which was proposed in the domain of stochastic search (having no notion of state space nor that of sequential decisions), to that of sequential decision making and show that our policy class can be updated under a KL-constraint in closed form, when the learned Q-Function is a quadratic function of the state and action space. In order to maximize sample efficiency, we rely on importance sampling to reuse transition samples from policies of all time-steps and all previous iterations in a principled way. MOTO is able to solve complex control problems despite the simplicity of the Q-Function thanks to two key properties: i) the learned Q-Function is fitted to samples of the current policy, which ensures that the function is *valid locally* and ii) the closed form update of the policy ensures that the KL-constraint is satisfied exactly irrespective of the number of samples or the non-linearity of the dynamics, which ensures that the Q-Function is *used locally*.

The experimental section demonstrates that on tasks with highly non-linear dynamics MOTO outperforms similar methods that rely on a linearization of these dynamics. Additionally, it is shown on a simulated Robot Table Tennis Task that MOTO is able to scale to high dimensional tasks while keeping the sample complexity relatively low; amenable to a direct application to a physical system.

Compared to Akrour et al. (2016), we report new experimental results comparing MOTO to TRPO (Schulman et al., 2015), a state-of-the-art reinforcement learning algorithm. These results showcase settings in which the time-dependent linear-Gaussian policies used by MOTO are a suitable alternative to neural networks. We also conduct a theoretical analysis of the policy update (Sec. 5) and lower bound the increase in policy return between successive iterations of the algorithm. The resulting lower bound validates the use of an expected KL-constraint (Sec. 3.1) in a trajectory-based policy optimization setting for ensuring a monotonic improvement of the policy return. Prior theoretical studies reported similar results when the *maximum* (over the state space) KL is upper bounded which is hard to enforce in practice (Schulman et al., 2015). Leveraging standard trajectory optimization assumptions, we extend prior analysis of the policy update to the specific setting of MOTO when only the *expected* policy KL is bounded.

2. Notation

Consider an undiscounted finite-horizon Markov Decision Process (MDP) of horizon T with state space $\mathcal{S} = \mathbb{R}^{\mathrm{d}_s}$ and action space $\mathcal{A} = \mathbb{R}^{\mathrm{d}_a}$. The transition function $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$,

which gives the probability (density) of transitioning to state $\boldsymbol{s}_{t+1}$ upon the execution of action $\boldsymbol{a}_t$ in $\boldsymbol{s}_t$, is assumed to be time-independent; while there are T time-dependent reward functions $r_t : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. A policy π is defined by a set of time-dependent density functions π_t, where $\pi_t(\boldsymbol{a}|\boldsymbol{s})$ is the probability of executing action $\boldsymbol{a}$ in state $\boldsymbol{s}$ at time-step t. The goal is to find the optimal policy $\pi^* = \{\pi_1^*, \dots, \pi_T^*\}$ maximizing the policy return $J(\pi) = \mathbb{E}_{\boldsymbol{s}_1, \boldsymbol{a}_1, \dots}\left[\sum_{t=1}^{T} r_t(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$, where the expectation is taken w.r.t. all the random variables $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ such that $\boldsymbol{s}_1 \sim \rho_1$ follows the distribution of the initial state, $\boldsymbol{a}_t \sim \pi_t(.|\boldsymbol{s}_t)$ and $\boldsymbol{s}_{t+1} \sim p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$.

As is common in Policy Search (Deisenroth et al., 2013), our algorithm operates on a restricted class of parameterized policies $\pi_{\boldsymbol{\theta}}, \boldsymbol{\theta} \in \mathbb{R}^{\mathrm{d}_\theta}$ and is an iterative algorithm comprising two main steps, *policy evaluation* and *policy update*. Throughout this article, we will assume that each time-dependent policy is parameterized by $\boldsymbol{\theta}_t = \{K_t, \boldsymbol{k}_t, \Sigma_t\}$ such that $\pi_{\boldsymbol{\theta}_t}$ is of linear-Gaussian form $\pi_{\boldsymbol{\theta}_t}(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(K_t s + \boldsymbol{k}_t, \Sigma_t)$, where the gain matrix K_t is a $d_a \times d_s$ matrix, the bias term $\boldsymbol{k}_t$ is a d_a dimensional column vector and the covariance matrix Σ_t, which controls the exploration of the policy, is of dimension $d_a \times d_a$; yielding a total number of parameters across all time-steps of $d_\theta = T(d_a d_s + \frac{1}{2} d_a(d_a + 3))$.

The policy at iteration i of the algorithm is denoted by π^i and following standard definitions, the Q-Function of π^i at time-step t is given by $Q_t^i(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, \dots}\left[\sum_{t'=t}^{T} r_{t'}(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'})\right]$ with $(\boldsymbol{s}_t, \boldsymbol{a}_t) = (\boldsymbol{s}, \boldsymbol{a})$ and $\boldsymbol{a}_{t'} \sim \pi_{t'}^i(.|\boldsymbol{s}_{t'}), \forall t' > t$. While the V-Function is given by $V_t^i(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a} \sim \pi_t(.|\boldsymbol{s})}\left[Q_t^\pi(\boldsymbol{s}, \boldsymbol{a})\right]$ and the Advantage Function by $A_t^i(\boldsymbol{s}, \boldsymbol{a}) = Q_t^i(\boldsymbol{s}, \boldsymbol{a}) - V_t^i(\boldsymbol{s})$. Furthermore the state distribution at time-step t, related to policy π^i, is denoted by $\rho_t^i(\boldsymbol{s})$. In order to keep the notations uncluttered, the time-step or the iteration number is occasionally dropped when a definition applies similarly for all time-steps or iteration number.

3. Model-free Policy Update for Trajectory-based Policy Optimization

MOTO alternates between policy evaluation and policy update. At each iteration i, the policy evaluation step generates a set of M rollouts[1] from the policy π^i in order to estimate a (quadratic) Q-Function $\tilde{Q}^i$ (Sec. 4.1) and a (Gaussian) state distribution $\tilde{\rho}^i$ (Sec. 4.3). Using these quantities, an information-theoretic policy update is derived at each time-step that uses a KL-bound as a trust region to obtain the policy π^{i+1} of the next iteration.

3.1 Optimization Problem

The goal of the policy update is to return a new policy π^{i+1} that maximizes the Q-Function $\tilde{Q}^i$ in expectation under the state distribution $\tilde{p}_i$ of the previous policy π^i. In order to limit policy oscillation between iterations (Wagner, 2011), the KL w.r.t. π^i is upper bounded. The use of the KL divergence to define the step-size of the policy update has already been successfully applied in prior work (Peters et al., 2010; Levine and Abbeel, 2014; Schulman et al., 2015). Additionally, we lower bound the entropy of π^{i+1} in order to better control

1. A rollout is a Monte Carlo simulation of a trajectory according to ρ_1, π and p or the execution of π on a physical system.

the reduction of exploration yielding the following non-linear program:

$$\underset{\pi}{\text{maximize}} \quad \int\int \tilde{\rho}_t^i(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\tilde{Q}_t^i(\boldsymbol{s},\boldsymbol{a})\mathrm{d}a\mathrm{d}s, \tag{1}$$

$$\text{subject to} \quad \mathbb{E}_{s\sim\tilde{\rho}_t^i(\boldsymbol{s})}\left[\mathrm{KL}(\pi(.|\boldsymbol{s}) \parallel \pi_t^i(.|\boldsymbol{s}))\right] \leq \epsilon, \tag{2}$$

$$\mathbb{E}_{s\sim\tilde{\rho}_t^i(\boldsymbol{s})}\left[\mathcal{H}\left(\pi(.|\boldsymbol{s})\right)\right] \geq \beta. \tag{3}$$

The KL between two distributions p and q is given by $\mathrm{KL}(p \parallel q) = \int p(x)\log\frac{p(x)}{q(x)}\mathrm{d}x$ while the entropy $\mathcal{H}$ is given by $\mathcal{H} = -\int p(x)\log p(x)\mathrm{d}x$. The step-size ϵ is a hyper-parameter of the algorithm kept constant throughout the iterations while β is set according to the entropy of the current policy π_t^i, $\beta = \mathbb{E}_{s\sim\tilde{\rho}_t^i(\boldsymbol{s})}\left[\mathcal{H}\left(\pi_t^i(.|\boldsymbol{s})\right)\right] - \beta_0$ and β_0 is the entropy reduction hyper-parameter kept constant throughout the iterations.

Eq. (1) indicates that π_t^{i+1} maximizes $\tilde{Q}_t^i$ in expectation under its own action distribution and the state distribution of π_t^i. Eq. (2) bounds the average change in the policy to the step-size ϵ while Eq. (3) controls the exploration-exploitation trade-off and ensures that the exploration in the action space (which is directly linked to the entropy of the policy) is not reduced too quickly. A similar constraint was introduced in the stochastic search domain by (Abdolmaleki et al., 2015), and was shown to avoid premature convergence. This constraint is even more crucial in our setting because of the inherent *non-stationarity* of the objective function being optimized at each iteration. The cause for the non-stationarity of the objective optimized at time-step t in the policy update is twofold: i) updates of policies $\pi_{t'}$ with time-step $t' > t$ will modify in the next iteration of the algorithm $\tilde{Q}_t$ as a function of s and a and hence the optimization landscape as a function of the policy parameters, ii) updates of policies with time-step $t' < t$ will induce a change in the state distribution ρ_t. If the policy had unlimited expressiveness, the optimal solution of Eq. (1) would be to choose $\arg\max_a \tilde{Q}_t$ irrespective of ρ_t. However, due to the restricted class of the policy, any change in ρ_t will likely change the optimization landscape including the position of the optimal policy parameter. Hence, Eq. (3) ensures that exploration in action space is maintained as the optimization landscape evolves and avoids premature convergence.

3.2 Closed Form Update

Using the method of Lagrange multipliers, the solution of the optimization problem in section 3.1 is given by

$$\pi_t'(\boldsymbol{a}|\boldsymbol{s}) \propto \pi_t(\boldsymbol{a}|\boldsymbol{s})^{\eta^*/(\eta^*+\omega^*)}\exp\left(\frac{\tilde{Q}_t(\boldsymbol{s},\boldsymbol{a})}{\eta^*+\omega^*}\right), \tag{4}$$

with η^* and ω^* being the optimal Lagrange multipliers related to the KL and entropy constraints respectively. Assuming that $\tilde{Q}_t(\boldsymbol{s},\boldsymbol{a})$ is of quadratic form in a and s

$$\tilde{Q}_t(\boldsymbol{s},\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T Q_{aa}\boldsymbol{a} + \boldsymbol{a}^T Q_{as}\boldsymbol{s} + \boldsymbol{a}^T\boldsymbol{q}_a + q(\boldsymbol{s}), \tag{5}$$

with $q(\boldsymbol{s})$ grouping all terms of $\tilde{Q}_t(\boldsymbol{s},\boldsymbol{a})$ that do not depend[2] on a, then $\pi'_t(\boldsymbol{a}|\boldsymbol{s})$ is again of linear-Gaussian form

$$\pi'_t(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{a}|FL\boldsymbol{s} + F\boldsymbol{f}, F(\eta^* + \omega^*)),$$

such that the gain matrix, bias and covariance matrix of π'_t are function of matrices F and L and vector $\boldsymbol{f}$ where

$$F = (\eta^*\Sigma_t^{-1} - Q_{aa})^{-1}, \qquad L = \eta^*\Sigma_t^{-1}K_t + Q_{as}, \qquad \boldsymbol{f} = \eta^*\Sigma_t^{-1}\boldsymbol{k}_t + \boldsymbol{q}_a.$$

Note that $\eta\Sigma_t^{-1} - Q_{aa}$ needs to be invertible and positive semi-definite as it defines the new covariance matrix of the linear-Gaussian policy. For this to hold, either $Q_t(\boldsymbol{s},.)$ needs to be concave in a (i.e. Q_{aa} is negative semi-definite), or η needs to be large enough (and for any Q_{aa} such η always exists). A too large η is not desirable as it would barely yield a change to the current policy (too small KL divergence) and could negatively impact the convergence speed. Gradient based algorithms for learning model parameters with a specific semi-definite shape are available (Bhojanapalli et al., 2015) and could be used for learning a concave Q_t. However, we found in practice that the resulting η was always small enough (resulting in a maximally tolerated KL divergence of ϵ between successive policies) while F remains well defined, without requiring additional constraints on the nature of Q_{aa}.

3.3 Dual Minimization

The Lagrangian multipliers η and ω are obtained by minimizing the convex dual function

$$g_t(\eta,\omega) = \eta\epsilon - \omega\beta + (\eta+\omega)\int \tilde{\rho}_t(\boldsymbol{s}) \log\left(\int \pi_t(\boldsymbol{a}|\boldsymbol{s})^{\eta/(\eta+\omega)} \exp\left(\tilde{Q}_t(\boldsymbol{s},\boldsymbol{a})/(\eta+\omega)\right) \mathrm{d}\boldsymbol{a}\right) \mathrm{d}\boldsymbol{s}.$$

Exploiting the structure of the quadratic Q-Function $\tilde{Q}_t$ and the linear-Gaussian policy $\pi_t(\boldsymbol{a}|\boldsymbol{s})$, the inner integral over the action space can be evaluated in closed form and the dual simplifies to

$$g(\eta,\omega) = \eta\epsilon_t - \omega\beta_t + \int \rho_t(\boldsymbol{s})(\boldsymbol{s}^T M\boldsymbol{s} + \boldsymbol{s}^T\boldsymbol{m} + m_0)\mathrm{d}\boldsymbol{s}.$$

The dual function further simplifies, by additionally assuming normality of the state distribution $\tilde{\rho}_t(\boldsymbol{s}) = \mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu}_s, \Sigma_s)$, to the function

$$g_t(\eta,\omega) = \eta\epsilon - \omega\beta + \boldsymbol{\mu}_s^T M\boldsymbol{\mu}_s + \mathrm{tr}(\Sigma_s M) + \boldsymbol{\mu}_s^T\boldsymbol{m} + m_0,$$

which can be efficiently optimized by gradient descent to obtain η^* and ω^*. The full expression of the dual function, including the definition of M, $\boldsymbol{m}$ and m_0 in addition to the partial derivatives $\frac{\partial g_t(\eta,\omega)}{\partial\eta}$ and $\frac{\partial g_t(\eta,\omega)}{\partial\omega}$ are given in Appendix A.

2. Constant terms and terms depending on s but not a won't appear in the policy update. As such, and albeit we only refer in this article to $Q_t(\boldsymbol{s},\boldsymbol{a})$, the Advantage Function $A_t(\boldsymbol{s},\boldsymbol{a})$ can be used interchangeably in lieu of $Q_t(\boldsymbol{s},\boldsymbol{a})$ for updating the policy.

4. Sample Efficient Policy Evaluation

The KL constraint introduced in the policy update gives rise to a non-linear optimization problem. This problem can still be solved in closed form for the class of linear-Gaussian policies, if the learned function $\tilde{Q}_t^i$ is quadratic in $\boldsymbol{s}$ and $\boldsymbol{a}$. The first subsection introduces the main supervised learning problem solved during the policy evaluation for learning $\tilde{Q}_t^i$ while the remaining subsections discuss how to improve its sample efficiency.

4.1 The Q-Function Supervised Learning Problem

In the remainder of the section, we will be interested in finding the parameter $\boldsymbol{w}$ of a linear model $\tilde{Q}_t^i = \langle \boldsymbol{w}, \phi(\boldsymbol{s}, \boldsymbol{a}) \rangle$, where the feature function ϕ contains a bias and all the linear and quadratic terms of s and a, yielding $1 + (d_a + d_s)(d_a + d_s + 3)/2$ parameters. $\tilde{Q}_t^i$ can subsequently be written as in Eq. (5) by extracting Q_{aa}, Q_{as} and $\boldsymbol{q}_a$ from $\boldsymbol{w}$.

At each iteration i, M rollouts are performed following π^i. Let us initially assume that $\tilde{Q}_t^i$ is learned only from samples $\mathcal{D}_t^i = \{\boldsymbol{s}_t^{[k]}, \boldsymbol{a}_t^{[k]}, \boldsymbol{s}_{t+1}^{[k]}; k = 1..M\}$ gathered from the execution of the M rollouts. The parameter $\boldsymbol{w}$ of $\tilde{Q}_t^i$ is learned by regularized linear least square regression

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \frac{1}{M} \sum_{k=1}^{M} \big(\langle \boldsymbol{w}, \phi(\boldsymbol{s}_t^{[k]}, \boldsymbol{a}_t^{[k]}) \rangle - \hat{Q}_t^i(\boldsymbol{s}_t^{[k]}, \boldsymbol{a}_t^{[k]}) \big)^2 + \lambda \boldsymbol{w}^T \boldsymbol{w}, \tag{6}$$

where the target value $\hat{Q}_t^i(\boldsymbol{s}^{[k]}, \boldsymbol{a}^{[k]})$ is a noisy estimate of the true value $Q_t^i(\boldsymbol{s}_t^{[k]}, \boldsymbol{a}_t^{[k]})$. We will distinguish two cases for obtaining the estimate $\hat{Q}_t^i(\boldsymbol{s}_t^{[k]}, \boldsymbol{a}_t^{[k]})$.

4.1.1 Monte-Carlo Estimate

This estimate is obtained by summing the future rewards for each trajectory k, yielding $\hat{Q}_t^i(\boldsymbol{s}_t^{[k]}, \boldsymbol{a}_t^{[k]}) = \sum_{t'=t}^{T} r_{t'}(\boldsymbol{s}_{t'}^{[k]}, \boldsymbol{a}_{t'}^{[k]})$. This estimator is known to have no bias but high variance. The variance can be reduced by averaging over multiple rollouts, assuming we can reset to states $\boldsymbol{s}_t^{[k]}$. However, such an assumption would severely limit the applicability of the algorithm on physical systems.

4.1.2 Dynamic Programming

In order to reduce the variance, this estimate exploits the V-Function to reduce the noise of the expected rewards of time-steps $t' > t$ through the following identity

$$\hat{Q}_t^i(\boldsymbol{s}_t^{[k]}, \boldsymbol{a}_t^{[k]}) = r_t(\boldsymbol{s}_t^{[k]}, \boldsymbol{a}_t^{[k]}) + \hat{V}_{t+1}^i(\boldsymbol{s}_{t+1}^{[k]}), \tag{7}$$

which is unbiased if $\hat{V}_{t+1}^i$ is. However, we will use for $\hat{V}_{t+1}^i$ an approximate V-Function $\tilde{V}_{t+1}^i$ learned recursively in time. This approximation might introduce a bias which will accumulate as t goes to 1. Fortunately, $\tilde{V}$ is not restricted by our algorithm to be of a particular class as it does not appear in the policy update. Hence, the bias can be reduced by increasing the complexity of the function approximator class. Nonetheless, in this article, a quadratic function will also be used for the V-Function which worked well in our experiments.

The V-Function is learned by first assuming that $\tilde{V}^i_{T+1}$ is the zero function.[3] Subsequently and recursively in time, the function $\tilde{V}^i_{t+1}$ and the transition samples in $\mathcal{D}^i_t$ are used to fit the parametric function $\tilde{V}^i_t$ by minimizing the loss $\sum_{k=1}^{M} \left(\hat{Q}^i_t(\boldsymbol{s}^{[k]}_t, \boldsymbol{a}^{[k]}_t) - \tilde{V}^i_t(\boldsymbol{s}^{[k]}_t)\right)^2$.

In addition to reducing the variance of the estimate $\hat{Q}^i_t(\boldsymbol{s}^{[k]}_t, \boldsymbol{a}^{[k]}_t)$, the choice of learning a V-Function is further justified by the increased possibility of reusing sample transitions from all time-steps and previous iterations.

4.2 Sample Reuse

In order to improve the sample efficiency of our approach, we will reuse samples from different time-steps and iterations using importance sampling. Let the expected loss which $\tilde{Q}^i_t$ minimizes under the assumption of an infinite number of samples be

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \mathbb{E}[\ell^i_t(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}'; \boldsymbol{w})],$$

where the loss ℓ^i_t is the inner term within the sum in Eq. (6); the estimate $\hat{Q}^i_t(\boldsymbol{s}^{[k]}_t, \boldsymbol{a}^{[k]}_t)$ is taken as in Eq. (7) and the expectation is with respect to the current state $\boldsymbol{s} \sim \rho^i_t$, the action $\boldsymbol{a} \sim \pi^i_t(.|\boldsymbol{s})$ and the next state $\boldsymbol{s}' \sim p(.|\boldsymbol{s}, \boldsymbol{a})$.

4.2.1 Reusing samples from different time-steps

To use transition samples from all time-steps when learning $\tilde{Q}^i_t$, we rely on importance sampling, where the importance weight (IW) is given by the ratio between the state-action probability of the current time-step $z^i_t(\boldsymbol{s}, \boldsymbol{a}) = \rho^i_t(\boldsymbol{s})\pi^i_t(\boldsymbol{a}|\boldsymbol{s})$ divided by the time-independent state-action probability of π^i given by $z^i(\boldsymbol{s}, \boldsymbol{a}) = \frac{1}{T}\sum_{t=1}^{T} z^i_t(\boldsymbol{s}, \boldsymbol{a})$. The expected loss minimized by $\tilde{Q}^i_t$ becomes

$$\min_{\boldsymbol{w}} \mathbb{E}\left[\frac{z^i_t(\boldsymbol{s}, \boldsymbol{a})}{z^i(\boldsymbol{s}, \boldsymbol{a})} \ell^i_t(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}'; \boldsymbol{w}) \mid (\boldsymbol{s}, \boldsymbol{a}) \sim z^i(\boldsymbol{s}, \boldsymbol{a})\right]. \tag{8}$$

Since the transition probabilities are not time-dependent they cancel out from the IW. Upon the computation of the IW, weighted least square regression is used to minimize an empirical estimate of (8) for the data set $\mathcal{D}^i = \cup_{t=1}^{T} \mathcal{D}^i_t$. Note that the (numerator of the) IW needs to be recomputed at every time-step for all samples $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D}^i$. Additionally, if the rewards are time-dependent, the estimate $\hat{Q}^i_t(\boldsymbol{s}^{[k]}_t, \boldsymbol{a}^{[k]}_t)$ in Eq. (7) needs to be recomputed with the current time-dependent reward, assuming the reward function is known.

4.2.2 Reusing samples from previous iterations

Following a similar reasoning, at a given time-step t, samples from previous iterations can be reused for learning $\tilde{Q}^i_t$. In this case, we have access to the samples of the state-action distribution $z^{1:i}_t(\boldsymbol{s}, \boldsymbol{a}) \propto \sum_{j=1}^{i} z^j_t(\boldsymbol{s}, \boldsymbol{a})$. The computation of $z^{1:i}_t$ requires the storage of all previous policies and state distributions. Thus, we will in practice limit ourselves to the K last iterations.

3. Alternatively one could assume the presence of a final reward $R_{T+1}(\boldsymbol{s}_{t+1})$, as is usually formulated in control tasks (Bertsekas, 1995), to which V^i_{T+1} could be initialized to.

Finally, both forms of sample reuse will be combined for learning $\tilde{Q}_t^i$ under the complete data set up to iteration i, $\mathcal{D}^{1:i} = \cup_{j=1}^{i} \mathcal{D}^j$ using weighted least square regression where the IW are given by $z_t^i(\boldsymbol{s}, \boldsymbol{a})/z^{1:i}(\boldsymbol{s}, \boldsymbol{a})$ with $z^{1:i}(\boldsymbol{s}, \boldsymbol{a}) \propto \sum_{t=1}^{T} z_t^{1:i}(\boldsymbol{s}, \boldsymbol{a})$.

4.3 Estimating the State Distribution

To compute the IW, the state distribution at every time-step ρ_t^i needs to be estimated. Since M rollouts are sampled for every policy π^i only M state samples are available for the estimation of ρ_t^i, necessitating again the reuse of previous samples to cope with higher dimensional control tasks.

4.3.1 Forward propagation of the state distribution

The first investigated solution for the estimation of the state distribution is the propagation of the estimate $\tilde{\rho}_t^i$ forward in time. Starting from $\tilde{\rho}_1^i$ which is identical for all iterations, importance sampling is used to learn $\tilde{\rho}_{t+1}^i$ with $t > 1$ from samples $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \in \mathcal{D}_t^{1:i}$ by weighted maximum-likelihood; where each sample $\boldsymbol{s}_{t+1}$ is weighted by $z_t^i(\boldsymbol{s}_t, \boldsymbol{a}_t)/z_t^{1:i}(\boldsymbol{s}_t, \boldsymbol{a}_t)$. And the computation of this IW only depends on the previously estimated state distribution $\tilde{\rho}_t^i$. In practice however, the estimate $\tilde{\rho}_t^i$ might entail errors despite the use of all samples from past iterations, which are propagated forward leading to a degeneracy of the number of effective samples in latter time-steps.

4.3.2 State distribution of a mixture policy

The second considered solution for the estimation of $\tilde{\rho}_t^i$ is heuristic but behaved better in practice. It is based on the intuition that the KL constraint of the policy update will yield state distributions that are close to each other (see Sec. 5 for a theoretical justification of the closeness in state distributions) and state samples from previous iterations can be reused in a simpler manner. Specifically, $\tilde{\rho}_t^i$ will be learned from samples of the mixture policy $\pi^{1:i} \propto \sum_{j=1}^{i} \gamma^{i-j} \pi^j$ which selects a policy from previous iterations with an exponentially decaying (w.r.t. the iteration number) probability and executes it for a whole rollout. In practice, the decay factor γ is selected according to the dimensionality of the problem, the number of samples per iterations M and the KL upper bound ϵ (intuitively, a smaller ϵ yields closer policies and henceforth more reusable samples). The estimated state distribution $\tilde{\rho}_t^i$ is learned as a Gaussian distribution by weighted maximum likelihood from samples of $\mathcal{D}_t^{1:i}$ where a sample of iteration j is weighted by γ^{i-j}.

4.4 The MOTO Algorithm

MOTO is summarized in Alg. 1. The innermost loop is split between policy evaluation (Sec. 4) and policy update (Sec. 3). For every time-step t, once the state distribution $\tilde{\rho}_t^i$ is estimated, the IWs of all the transition samples are computed and used to learn the Q-Function (and the V-Function using the same IWs, if dynamic programming is used when estimating the Q-Function), concluding the policy evaluation part. Subsequently, the components of the quadratic model $\tilde{Q}_t^i$ that depend on the action are extracted and used to find the optimal dual parameters η^* and ω^* that are respectively related to the KL and the entropy constraint, by minimizing the convex dual function g_t^i using gradient descent.

Algorithm 1 Model-Free Trajectory-based Policy Optimization (MOTO)

Input: Initial policy π^0, number of trajectories per iteration M, step-size ϵ and entropy reduction rate β_0
Output: Policy π^N
for $i = 0$ **to** $N - 1$ **do**
  Sample M trajectories from π^i
  for $t = T$ **to** 1 **do**
    Estimate state distribution $\tilde{\rho}_t^i$ (Sec. 4.3)
    Compute IW for all $(\boldsymbol{s}, a, s') \in \mathcal{D}^{1:i}$ (Sec. 4.2)
    Estimate the Q-Function $\tilde{Q}_t^i$ (Sec. 4.1)
    Optimize: $(\eta^*, \omega^*) = \arg\min g_t^i(\eta, \omega)$ (Sec. 3.3)
    Update π_t^{i+1} using η^*, ω^*,$\tilde{\rho}_t^i$ and $\tilde{Q}_t^i$ (Sec. 3.2)
  end for
end for

The policy update then uses η^* and ω^* to return the new policy π_{t+1} and the process is iterated.

In addition to the simplification of the policy update, the rationale behind the use of a local quadratic approximation for Q_t^i is twofold: i) since Q_t^i is only optimized locally (because of the KL constraint), a quadratic model would potentially contain as much information as a Hessian matrix in a second order gradient descent setting ii) If $\tilde{Q}_t$ in Eq. (4) is an arbitrarily complex model then it is common that π_t', of linear-Gaussian form, is fit by weighted maximum-likelihood (Deisenroth et al., 2013); it is clear though from Eq. (4) that however complex $\tilde{Q}_t(\boldsymbol{s}, \boldsymbol{a})$ is, if both π_t and π_t' are of linear-Gaussian form then there exist a quadratic model that would result in the same policy update. Additionally, note that $\tilde{Q}_t$ is not used when learning $\tilde{Q}_{t-1}$ (sec. 4.1) and hence the bias introduced by $\tilde{Q}_t$ will not propagate back. For these reasons, we think that choosing a more complex class for $\tilde{Q}_t$ than that of quadratic functions might not necessarily lead to an improvement of the resulting policy, for the class of linear-Gaussian policies.

5. Monotonic Improvement of the Policy Update

We analyze in this section the properties of the constrained optimization problem solved during our policy update. Kakade and Langford (2002) showed that in the approximate policy iteration setting, a monotonic improvement of the policy return can be obtained if the successive policies are close enough. While in our algorithm the optimization problem defined in Sec. 3.1 bounds the expected policy KL under the state distribution of the current iteration i, it does not tell us how similar the policies are under the new state distribution and a more careful analysis needs to be conducted.

The analysis we present builds on the results of Kakade and Langford (2002) to lower-bound the change in policy return $J(\pi^{i+1}) - J(\pi^i)$ between the new policy π^{i+1} (solution of the optimization problem defined in Sec. 3.1) and the current policy π^i. Unlike Kakade and Langford (2002), we enforce closeness between successive policies with a KL constraint instead of by mixing π^{i+1} and π^i. Related results were obtained when a KL constraint is

used in Schulman et al. (2015). Our main contribution is to extend these results to the trajectory optimization setting with continuous states and actions and where the expected KL between the policies is bounded instead of the maximum KL over the state space (which is hard to achieve in practice).

In what follows, p and q denote the next policy π^{i+1} and the current policy π^i respectively. We will denote the state distribution and policy at time-step t by p_t and $p_t(.|\boldsymbol{s})$ respectively (and similarly for q). First, we start by writing the difference between policy returns in term of advantage functions.

Lemma 1 *For any two policies p and q, and where A^q_t denotes the advantage function at time-step t of policy q, the difference in policy return is given by*

$$J(p) - J(q) = \sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{s}\sim p_t, \boldsymbol{a}\sim p_t(.|\boldsymbol{s})} \left[A^q_t(\boldsymbol{s},\boldsymbol{a})\right].$$

The proof of Lemma 1 is given by the proof of Lemma 5.2.1 in (Kakade, 2003). Note that Lemma 1 expresses the change in policy return in term of expected advantage under the current state distribution while we optimize the advantage function under the state distribution of policy q, which is made apparent in Lemma 2.

Lemma 2 *Let $\epsilon_t = \mathrm{KL}(p_t \parallel q_t)$ be the KL divergence between state distributions $p_t(.)$ and $q_t(.)$ and let $\delta_t = \max_s |\mathbb{E}_{\boldsymbol{a}\sim p_t(.|\boldsymbol{s})}[A^q_t(\boldsymbol{s},\boldsymbol{a})]|$, then for any two policies p and q we have*

$$J(p) - J(q) \geq \sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{s}\sim q_t, \boldsymbol{a}\sim p_t(.|\boldsymbol{s})} \left[A^q_t(\boldsymbol{s},\boldsymbol{a})\right] - 2\sum_{t=1}^{T} \delta_t \sqrt{\frac{\epsilon_t}{2}}.$$

Proof

$$\begin{aligned}
\mathbb{E}_{s\sim p_t, a\sim p_t(.|\boldsymbol{s})} \left[A^q_t(\boldsymbol{s}_t,\boldsymbol{a}_t)\right] &= \int p_t(\boldsymbol{s}) \int p_t(\boldsymbol{a}_t|\boldsymbol{s}_t) A^q_t(\boldsymbol{s}_t,\boldsymbol{a}_t), \\
&= \int q_t(\boldsymbol{s}) \int p_t(\boldsymbol{a}_t|\boldsymbol{s}_t) A^q_t(\boldsymbol{s}_t,\boldsymbol{a}_t) \\
&\quad + \int (p_t(\boldsymbol{s}) - q_t(\boldsymbol{s})) \int p_t(\boldsymbol{a}_t|\boldsymbol{s}_t) A^q_t(\boldsymbol{s}_t,\boldsymbol{a}_t), \\
&\geq \mathbb{E}_{s\sim q_t, a\sim p_t(.|\boldsymbol{s})} \left[A^q_t(\boldsymbol{s},\boldsymbol{a})\right] - \delta_t \int (p_t(\boldsymbol{s}) - q_t(\boldsymbol{s})), \\
&\geq \mathbb{E}_{s\sim q_t, a\sim p_t(.|\boldsymbol{s})} \left[A^q_t(\boldsymbol{s},\boldsymbol{a})\right] - 2\delta_t \frac{1}{2} \int |p_t(\boldsymbol{s}) - q_t(\boldsymbol{s})|, \\
&\geq \mathbb{E}_{s\sim q_t, a\sim p_t(.|\boldsymbol{s})} \left[A^q_t(\boldsymbol{s},\boldsymbol{a})\right] - 2\delta_t \sqrt{\frac{1}{2}\mathrm{KL}(p_t \parallel q_t)}. \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(Pinsker's inequality)}
\end{aligned}$$

Summing over the time-steps and using Lemma 1 completes the proof. ■

Lemma 2 lower-bounds the change in policy return by the advantage term optimized during the policy update and a negative change that quantifies the change in state distributions between successive policies. The core of our contribution is given by Lemma 3 which

relates the change in state distribution to the expected KL constraint between policies of our policy update.

Lemma 3 *If for every time-step, the state distributions p_t and q_t are Gaussian and the policies $p_t(.|\boldsymbol{s}_t)$ and $q_t(.|\boldsymbol{s}_t)$ are linear-Gaussian and if $\mathbb{E}_{s\sim q_t}\left[\mathrm{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))\right] \leq \epsilon$ for every time-step then $\mathrm{KL}(p_t \parallel q_t) = \mathcal{O}(\epsilon)$ as $\epsilon \to 0$ for every time-step.*

Proof We will demonstrate the lemma by induction noting that for $t = 1$ the state distributions are identical and hence their KL is zero. Assuming $\epsilon_t = \mathrm{KL}(p_t \parallel q_t) = \mathcal{O}(\epsilon)$ as $\epsilon \to 0$, let us compute the KL between state distributions for $t+1$

$$\begin{aligned}
\mathrm{KL}(p_{t+1} \parallel q_{t+1}) &= \int p_{t+1}(\boldsymbol{s}') \log \frac{p_{t+1}(\boldsymbol{s}')}{q_{t+1}(\boldsymbol{s}')}, \\
&\leq \iiint p_t(\boldsymbol{s},\boldsymbol{a}) p(\boldsymbol{s}'|\boldsymbol{a},\boldsymbol{s}) \log \frac{p_t(\boldsymbol{s},\boldsymbol{a}) p(\boldsymbol{s}'|\boldsymbol{a},\boldsymbol{s})}{q_t(\boldsymbol{s},\boldsymbol{a}) p(\boldsymbol{s}'|\boldsymbol{a},\boldsymbol{s})}, \qquad \text{(log sum inequality)} \\
&= \int p_t(\boldsymbol{s}') \int p_t(\boldsymbol{a}|\boldsymbol{s}') \log \frac{p_t(\boldsymbol{s}') p_t(\boldsymbol{a}|\boldsymbol{s}')}{q_t(\boldsymbol{s}') q_t(\boldsymbol{a}|\boldsymbol{s}')}, \\
&= \epsilon_t + \mathbb{E}_{s\sim p_t}[\mathrm{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))]. \qquad (9)
\end{aligned}$$

Hence we have bounded the KL between state distributions at $t+1$ by the KL between state distributions and the expected KL between policies of the previous time-step t. Now we will express the KL between policies under the new state distributions, given by $\mathbb{E}_{s\sim p_t}[\mathrm{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))]$, in terms of KL between policies under the previous state distribution, $\mathbb{E}_{s\sim q_t}\left[\mathrm{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))\right]$ which is bounded during policy update by ϵ, and $\mathrm{KL}(p_t \parallel q_t)$. To do so, we will use the assumption that the state distribution and the policy are Gaussian and linear-Gaussian. The complete demonstration is given in Appendix B, and we only report the following result

$$\mathbb{E}_{s\sim p_t}[\mathrm{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))] \leq 2\epsilon \left(3\epsilon_t + d_s + 1\right). \qquad (10)$$

It is now easy to see that the combination of (9) and (10) together with the induction hypothesis yields $\mathrm{KL}(p_{t+1} \parallel q_{t+1}) = \mathcal{O}(\epsilon)$ as $\epsilon \to 0$. ■

Finally, the combination of Lemma 2 and Lemma 3 results in the following theorem, lower-bounding the change in policy return.

Theorem 4 *If for every time-step the state distributions p_t and q_t are Gaussian and the policies $p_t(.|\boldsymbol{s}_t)$ and $q_t(.|\boldsymbol{s}_t)$ are linear-Gaussian and if $\mathbb{E}_{s\sim q_t}\left[\mathrm{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))\right] \leq \epsilon$ for every time-step then*

$$J(p) - J(q) \geq \sum_{t=1}^{T} \mathbb{E}_{s\sim q_t, a\sim p_t(.|\boldsymbol{s})}\left[A_t^q(\boldsymbol{s},\boldsymbol{a})\right] - \sum_{t=1}^{T} \delta_t \mathcal{O}(\sqrt{\epsilon}).$$

Theorem 4 shows that we are able to obtain similar bounds than those derived in (Schulman et al., 2015) for our continuous state-action trajectory optimization setting with a bounded KL policy update in expectation under the previous state distribution. While, it

is not easy to apply Theorem 4 in practice to choose an appropriate step-size ϵ since $A_t^q(\boldsymbol{s}, \boldsymbol{a})$ is generally only known approximately, Theorem 4 still shows that our constrained policy update will result in small changes in the overall behavior of the policy between successive iterations which is crucial in the approximate RL setting.

6. Related Work

In the Approximate Policy Iteration scheme (Szepesvari, 2010), policy updates can potentially decrease the expected reward, leading to policy oscillations (Wagner, 2011), unless the updated policy is 'close' enough to the previous one (Kakade and Langford, 2002). Bounding the change between π^i and π^{i+1} during the policy update step is thus a well studied idea in the Approximate Policy Iteration literature. Already in 2002, Kakade and Langford proposed the Conservative Policy Iteration (CPI) algorithm where the new policy π^{i+1} is obtained as a mixture of π^i and the greedy policy w.r.t. Q^i. The mixture parameter is chosen such that a lower bound of $J(\pi^{i+1}) - J(\pi^i)$ is positive and improvement is guaranteed. However, convergence was only asymptotic and in practice a single policy update would require as many samples as other algorithms would need to find the optimal solution (Pirotta et al., 2013b). Pirotta et al. (2013b) refined the lower bound of CPI by adding an additional term capturing the closeness between policies (defined as the matrix norm of the difference between the two policies), resulting in a more aggressive updates and better experimental results. However, both approaches only considered discrete action spaces. Pirotta et al. (2013a) provide an extension to continuous domains but only for single dimensional actions.

When the action space is continuous, which is typical in e.g. robotic applications, using a stochastic policy and updating it under a KL constraint to ensure 'closeness' of successive policies has shown several empirical successes (Daniel et al., 2012; Levine and Koltun, 2014; Schulman et al., 2015). However, only an empirical sample estimate of the objective function is generally optimized (Peters et al., 2010; Schulman et al., 2015), which typically requires a high number of samples and precludes it from a direct application to physical systems. The sample complexity can be reduced when a model of the dynamics is available (Levine and Koltun, 2014) or learned (Levine and Abbeel, 2014). In the latter work, empirical evidence suggests that good policies can be learned on high dimensional continuous state-action spaces with only a few hundred episodes. The counter part being that time-dependent dynamics are assumed to be linear, which is a simplifying assumption in many cases. Learning more sophisticated models using for example Gaussian Processes was experimented by Deisenroth and Rasmussen (2011) and Pan and Theodorou (2014) in the Policy Search and Trajectory Optimization context, but it is still considered to be a challenging task, see Deisenroth et al. (2013), chapter 3.

The policy update in Eq. (4) resembles that of (Peters et al., 2010; Daniel et al., 2012) with three main differences. First, without the assumption of a quadratic Q-Function, an additional weighted maximum likelihood step is required for fitting π^{i+1} to weighted samples as in the r.h.s of Eq. (4), since this policy might not be of the same policy class. As a result, the KL between π^i and π^{i+1} is no longer respected. Secondly, we added an entropy constraint in order to cope with the inherent non-stationary objective function maximized by the policy (Eq. 1) and to ensure that exploration is sustained, resulting in better quality

policies. Thirdly, their sample based optimization algorithm requires the introduction of a number of dual variables typically scaling at least linearly with the dimension of the state space, while we only have to optimize over two dual variables irrespective of the state space.

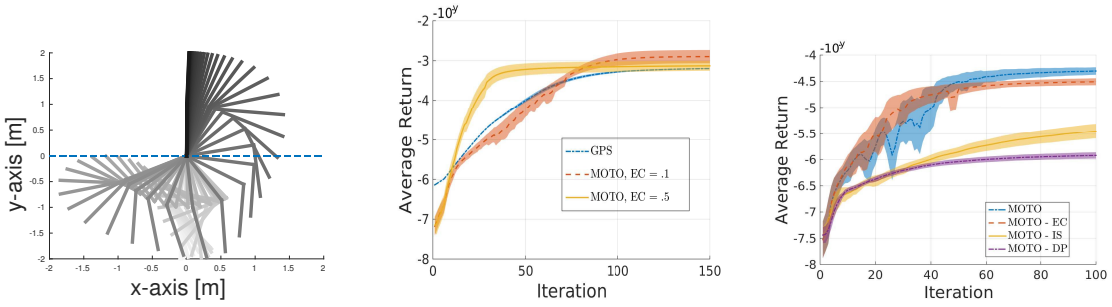
Most trajectory optimization methods are based on stochastic optimal control. These methods linearize the system dynamics and update the policy in closed form as a LQR. Instances of such algorithms are for example iLQG (Todorov, 2006), DDP (Theodorou et al., 2010), AICO (Toussaint, 2009) and its more robust variant (Rückert et al., 2014) and the trajectory optimization algorithm used in the GPS algorithm (Levine and Abbeel, 2014). These methods share the same assumptions as MOTO for ρ_t^i and π_t^i respectively considered to be of Gaussian and linear-Gaussian form. These methods face issues in maintaining the stability of the policy update and, similarly to MOTO, introduce additional constraints and regularizers to their update step. DDP, iLQG and AICO regularize the update by introducing a damping term in the matrix inversion step, while GPS uses a KL bound on successive trajectory distributions. However, as demonstrated in Sec. 7, the quadratic approximation of the Q-Function performed by MOTO seems to be empirically less detrimental to the quality of the policy update than the linearization of the system dynamics around the mean trajectory performed by related approaches.

7. Experimental Validation

MOTO is experimentally validated on a set of multi-link swing-up tasks and on a robot table tennis task. The experimental section aims at analyzing the proposed algorithm from four different angles: i) the quality of the returned policy comparatively to state-of-the-art trajectory optimization algorithms, ii) the effectiveness of the proposed variance reduction and sample reuse schemes, iii) the contribution of the added entropy constraint during policy updates in finding better local optima and iv) the ability of the algorithm to scale to higher dimensional problems. The experimental section concludes with a comparison to TRPO (Schulman et al., 2015), a state-of-the-art reinforcement learning algorithm that bounds the KL between successive policies; showcasing settings in which the time-dependent linear-Gaussian policies used by MOTO are a suitable alternative to neural networks.

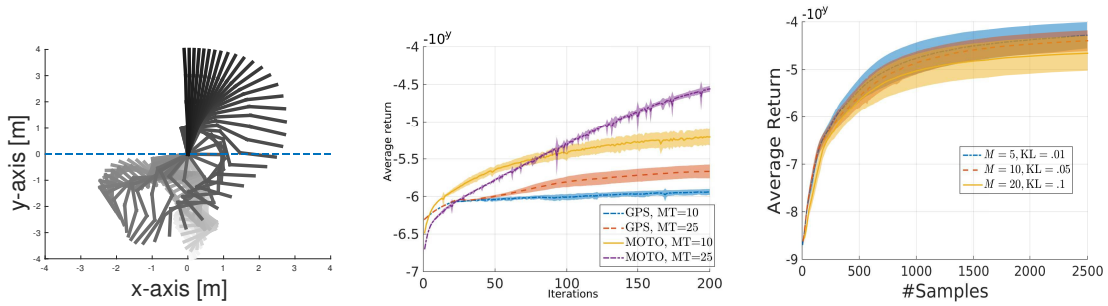
7.1 Multi-link Swing-up Tasks

A set of swing-up tasks involving a multi-link pole with respectively two and four joints (Fig. 1.a and 2.a) is considered in this section. The set of tasks includes several variants with different torque and joint limits, introducing additional non-linearities in the dynamics and resulting in more challenging control problems for trajectory optimization algorithms based on linearizing the dynamics. The state space consists of the joint positions and joint velocities while the control actions are the motor torques. In all the tasks, the reward function is split between an action cost and a state cost. The action cost is constant throughout the time-steps while the state cost is time-dependent and is equal to zero for all but the 20 last time-steps. During this period, a quadratic cost penalizes the state for not being the null vector, i.e. having zero velocity and reaching the upright position. Examples of successful swing-ups learned with MOTO are depicted in Fig. 1.a and 2.a.



(a) (b) (c)

Figure 1: a) Double link swing-up policy found by MOTO. b) Comparison between GPS and MOTO on the double link swing-up task (different torque limits and state costs are applied compared to c) and f). c) MOTO and its variants on the double link swing-up task: MOTO without the entropy constraint (EC), importance sampling (IS) or dynamic programming (DP). All plots are averaged over 15 runs.



(a) (b) (c)

Figure 2: a) Quad link swing-up policy found by MOTO. b) Comparison between GPS and MOTO on the quad link swing-up task with restricted joint limits and two different torque limits. c) MOTO on the double link swing-up task for varying number of rollouts per episode and step-sizes. All plots are averaged over 15 runs.

MOTO is compared to the trajectory optimization algorithm proposed in Levine and Abbeel (2014), that we will refer to as GPS.[4] We chose to compare MOTO and GPS as both use a KL constraint to bound the change in policy. As such, the choice of approximating the Q-Function with time-dependent quadratic models (as done in MOTO) in order to solve the policy update instead of linearizing the system dynamics around the mean trajectory (as done in most trajectory optimization algorithms) is better isolated. GPS and MOTO both use a time-dependent linear-Gaussian policy. In order to learn the linear

4. This is a slight abuse of notation as the GPS algorithm of (Levine and Abbeel, 2014) additionally feeds the optimized trajectory to an upper level policy. However, in this article, we are only interested in the trajectory optimization part.

model of the system dynamics, GPS reuses samples from different time-steps by learning a Gaussian mixture model on all the samples and uses this model as a prior to learn a joint Gaussian distribution $p(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ for every time-step. To single out the choice of linearizing the dynamics model or lack thereof from the different approaches to sample reuse, we give to both algorithm a high number of samples (200 and 400 rollouts per iteration for the double and quad link respectively) and bypass any form of sample reuse for both algorithms.

Fig. 1.b compares GPS to two configurations of MOTO on the double-link swing up task. The same initial policy and step-size ϵ are used by both algorithm. However, we found that GPS performs better with a smaller initial variance, as otherwise actions quickly hit the torque limits making the dynamics modeling harder. Fig. 1.b shows that even if the dynamics of the system are not linear, GPS manages to improve the policy return, and eventually finds a swing-up policy. The two configurations of MOTO have an entropy reduction constant β_0 of .1 and .5. The effect of the entropy constraint is similar to the one observed in the stochastic search domain by (Abdolmaleki et al., 2015). Specifically, a smaller entropy reduction constant β_0 results in an initially slower convergence but ultimately leads to higher quality policies. In this particular task, MOTO with $\beta_0 = .1$ manages to slightly outperform GPS.

Next, GPS and MOTO are compared on the quad link swing-up task. We found this task to be significantly more challenging than the double link and to increase the difficulty further, soft joint limits are introduced on the three last joints in the following way: whenever a joint angle exceeds in absolute value the threshold $\frac{2}{3}\pi$, the desired torque of the policy is ignored in favor of a linear-feedback controller that aims at pushing back the joint angle within the constrained range. As a result, Fig. 2.b shows that GPS can barely improve its average return (with the torque limits set to 25, as in the double link task.) while MOTO performs significantly better. Finally, the torque limits are reduced even further but MOTO still manages to find a swing-up policy as demonstrated by Fig. 2.a.

In the last set of comparisons, the importance of each of the components of MOTO is assessed on the double link experiment. The number of rollouts per iteration is reduced to $M = 20$. Fig. 1.c shows that: i) the entropy constraint provides an improvement on the quality of the policy in the last iterations in exchange of a slower initial progress, ii) importance sampling greatly helps in speeding-up the convergence and iii) the Monte-Carlo estimate of $\hat{Q}_i^t$ is not adequate for the smaller number of rollouts per iterations, which is further exacerbated by the fact that sample reuse of transitions from different time-steps is not possible with the Monte-Carlo estimate.

Finally, we explore on the double-link swing-up task several values of M, trying to find the balance between performing a small number of rollouts per iterations with a small step-size ϵ versus having a large number of rollouts for the policy evaluation that would allow to take larger update steps. To do so, we start with an initial $M = 20$ and successively divide this number by two until $M = 5$. In each case, the entropy reduction constant is set such that, for a similar number of rollouts, the entropy is reduced by the same amount, while we choose γ', the discount of the state sample weights as $\gamma' = \gamma^{M/M'}$ to yield again

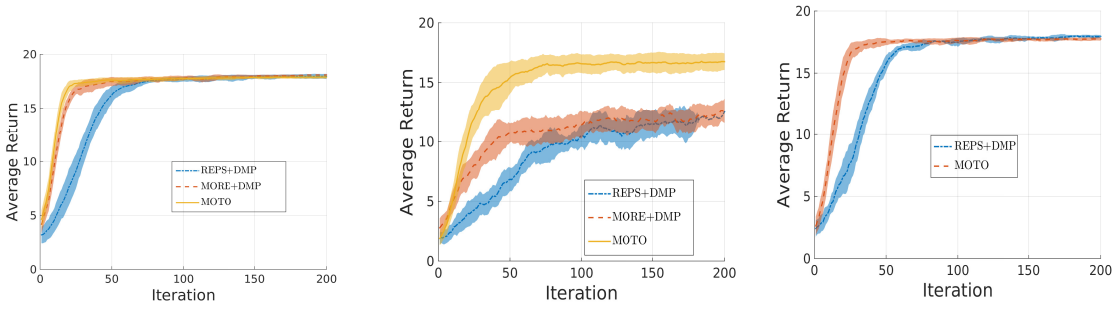


Figure 3: a) Comparison on the robot table tennis task with no noise on the initial velocity of the ball. b) Comparison on the robot table tennis task with Gaussian noise during the ball bounce on the table. c) Comparison on the robot table tennis task with initial velocity sampled uniformly in a 15cm range.

a similar sample decay after the same number of rollouts have been performed. Tuning ϵ was, however, more complicated and we tested several values on non-overlapping ranges for each M and selected the best one. Fig. 2.c shows that, on the double link swing-up task, a better sample efficiency is achieved with a smaller M. However, the improvement becomes negligible from $M = 10$ to $M = 5$. We also noticed a sharp decrease in the number of effective samples when M tends to 1. In this limit case, the complexity of the mixture policy $z^{1:i}$ in the denominator of the importance ratio increases with the decrease of M and might become a poor representation of the data set. Fitting a simpler state-action distribution that is more representative of the data can be the subject of future work in order to further improve the sample efficiency of the algorithm, which is crucial for applications on physical systems.

7.2 Robot Table Tennis

The considered robot table tennis task consists of a simulated robotic arm mounted on a floating base, having a racket on the end effector. The task of the robot is to return incoming balls using a forehand strike to the opposite side of the table (Fig. 4). The arm has 9 degrees of freedom comprising the six joints of the arm and the three linear joints of the base allowing (small) 3D movement. Together with the joint velocities and the 3D position of the incoming ball, the resulting state space is of dimension $d_s = 21$ and the action space is of dimension $d_a = 9$ and consists of direct torque commands.

We use the analytical player of Mülling et al. (2011) to generate a single forehand stroke, which is subsequently used to learn from demonstration the initial policy π^1. The analytica player comprises a waiting phase (keeping the arm still), a preparation phase, a hitting phase and a return phase, which resets the arm to the waiting position of the arm. Only the preparation and the hitting phase are replaced by a learned policy. The total control time for the two learned phases is of 300 time-steps at 500hz, although for the MOTO

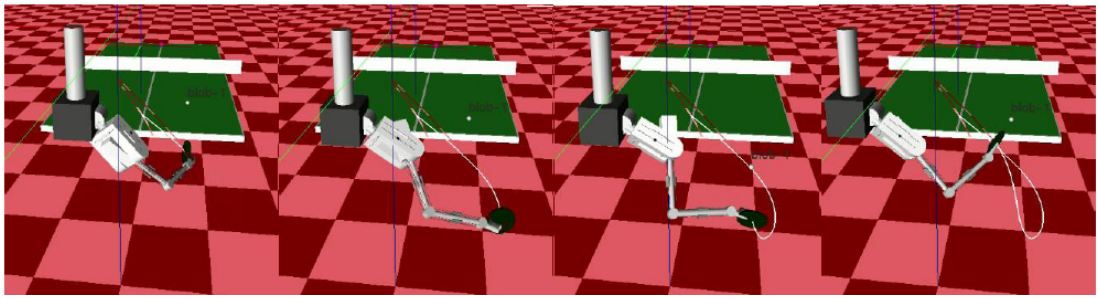


Figure 4: Robot table tennis setting and a forehand stroke learned by MOTO upon a spinning ball.

algorithm we subsequently divide the control frequency by a factor of 10 resulting in a time-dependent policy of 30 linear-Gaussian controllers.

The learning from demonstration step is straightforward and only consists in averaging the torque commands of every 10 time-steps and using these quantities as the initial bias for each of the 30 controllers. Although this captures the basic template of the forehand strike, no correlation between the action and the state (e.g. the ball position) is learned from demonstration as the initial gain matrix K for all the time-steps is set to the null matrix. Similarly, the exploration in action space is uninformed and initially set to the identity matrix.

Three settings of the task are considered, a noiseless case where the ball is launched with the same initial velocity, a varying context setting where the initial velocity is sampled uniformly within a fixed range and the noisy bounce setting where a Gaussian noise is added to both the x and y velocities of the ball upon bouncing on the table, to simulate the effect of a spin.

We compare MOTO to the policy search algorithm REPS (Kupcsik et al., 2013) and the stochastic search algorithm MORE (Abdolmaleki et al., 2015) that shares a related information-theoretic update. Both algorithms will optimize the parameters of a Dynamical Movement Primitive (DMP) (Ijspeert and Schaal, 2003). A DMP is a non-linear attractor system commonly used in robotics. The DMP is initialized from the same single trajectory and the two algorithm will optimize the goal joint positions and velocities of the attractor system. Note that the DMP generates a trajectory of states, which will be tracked by a linear controller using the inverse dynamics. While MOTO will directly output the torque commands and does not rely on this model.

Fig. 3.a and 3.c show that our algorithm converges faster than REPS and to a smaller extent than MORE in both the noiseless and the varying context setting. This is somewhat surprising since MOTO with its time-dependent linear policy have a much higher number of parameters to optimize than the 18 parameters of the DMP's attractor. However, the resulting policy in both cases is slightly less good than that of MORE and REPS. Note that for the varying context setting, we used a contextual variant of REPS that learns a mapping from the initial ball velocity to the DMP's parameters. MORE, on the other hand couldn't be compared in this setting. Finally, Fig. 3.b shows that our policy is successfully

capable of adapting to noise at ball bounce, while the other methods fail to do so since the trajectory of the DMP is not updated once generated.

7.3 Comparison to Neural Network Policies

Recent advances in reinforcement learning using neural network policies and supported by the ability of generating and processing large amounts of data allowed impressive achievements such as playing Atari at human level (Mnih et al., 2015) or mastering the game of Go (Silver et al., 2016). On continuous control tasks, success was found by combining trajectory optimization and supervised learning of a neural network policy (Levine and Abbeel, 2014), or by directly optimizing the policy's neural network using reinforcement learning (Lillicrap et al., 2015; Schulman et al., 2015). The latter work, side-stepping trajectory optimization to directly optimize a neural network policy raises the question as to whether the linear-Gaussian policies used in MOTO and related algorithms provide any benefit compared to neural network policies.

To this end, we propose to compare on the multi-link swing-up tasks of Sec. 7.1, MOTO learning a time-dependent linear-Gaussian policy to TRPO (Schulman et al., 2015) learning a neural network policy. We chose TRPO as our reinforcement learning baseline for its state-of-the-art performance and because of its similar policy update than that of MOTO (both bound the KL between successive policies). Three variants of TRPO are considered while for MOTO, we refrain from using importance sampling (Sec. 4.2) since similar techniques such as off-policy policy evaluation can be used for TRPO.

First, MOTO is compared to a default version of TRPO using OpenAI's baselines implementation (Dhariwal et al., 2017) where TRPO optimizes a neural network for both learning the policy and the V-Function. Default parameters are used except for the KL divergence constraint where we set $\epsilon = .1$ for TRPO to match MOTO's setting. Note that because the rewards are time-dependent (distance to the upright position penalized only for the last 20 steps, see Sec. 7.1) we add time as an additional entry to the state description. Time entry is in the interval $[0, 1]$ (current time-step divided by horizon T) and is fed to both the policy and V-Function neural networks. This first variant of TRPO answers the question: is there any benefit for using MOTO with its time-dependent linear-Gaussian policy instead of a state-of-the-art deep RL implementation with a neural network policy.

The second considered baseline uses the same base TRPO algorithm but replaces the policy evaluation using a neural network V-Function with the same policy evaluation used by MOTO (Sec. 4), back-propagating a quadratic V-Function. In this variant of TRPO the time-entry is dropped for the V-Function. This second baseline better isolates the policy update, which is the core of both algorithms, from the learning of the V-Function which could be interchanged.

Finally, we consider a third variant of TRPO that uses both the quadratic V-Function and a time-dependent linear-Gaussian policy with diagonal covariance matrix (standard formulation and implementation of TRPO does not support full covariance exploration noise). The time entry is dropped for both the V-Function and the policy in this third baseline. While both algorithms bound the KL divergence between successive policies, there are still a few differentiating factors between this third baseline and MOTO. First, TRPO bounds the KL of the whole policy while MOTO solves a policy update for each time-step independently

(but still results in a well-founded approximate policy iteration algorithm as discussed in Sec. 5). In practice the KL divergence upon update for every time-step for MOTO is often equal to ϵ and hence both MOTO and TRPO result in the same KL divergence of the overall policy (in expectation of the state distribution) while the KL divergence of the sub-policies (w.r.t. the time-step) may vary. Secondly, MOTO performs a quadratic approximation of the Q-Function and solves the policy update exactly while TRPO performs a quadratic approximation of the KL constraint and solves the policy update using conjugate gradient descent. TRPO does not solve the policy update in closed form because it would require a matrix inversion and the matrix to invert has the dimensionality of the number of policy parameters. In contrast, MOTO can afford the closed form solution because the matrix to invert has the dimensionality of the action space which is generally significantly smaller than the number of policy parameters.

Fig. 5 shows the learning performance of MOTO and three TRPO variants on the double link and quadruple link swing-up tasks (Sec. 7.1). In both tasks MOTO outperforms all three TRPO variants albeit when TRPO is combined with the quadratic V-Function (second variant), it initially outperforms MOTO on the double link swing-up task. The quadratic V-Function befits these two tasks in particular and the quadratic regulation setting more generally because the reward is a quadratic function of the state-action pair (here the negative squared distance to the upright position and a quadratic action cost). However, MOTO makes better use of the task's nature and largely outperforms the third variant of TRPO despite having a similar policy evaluation step and using the same policy class. In conclusion, while neural networks can be a general purpose policy class demonstrating success on a wide variety of tasks, on specific settings such as on quadratic regulator tasks, trajectory-based policy optimization is able to outperform deep RL algorithms. MOTO in particular, which does not rely on a linearization of the dynamics around the mean trajectory is able to handle quadratic reward problems with highly non-linear dynamics such as the quadruple link swing-up task and outperform state-of-the-art trajectory optimization algorithms (Sec. 7.1) as a result.

8. Conclusion

We proposed in this article MOTO, a new trajectory-based policy optimization algorithm that does not rely on a linearization of the dynamics. Yet, an efficient policy update could be derived in closed form by locally fitting a quadratic Q-Function. We additionally conducted a theoretical analysis of the constrained optimization problem solved during the policy update. We showed that the upper bound on the expected KL between successive policies leads to only a small drift in successive state distributions which is a key property in the approximate policy iteration scheme.

The use of a KL constraint is widely spread including in other trajectory optimization algorithms. The experiments demonstrate however that our algorithm has an increased robustness towards non-linearities of the system dynamics when compared to a closely related trajectory optimization algorithm. It appears as such that the simplification resulting from considering a local linear approximation of the dynamics is more detrimental to the overall convergence of the algorithm than a local quadratic approximation of the Q-Function.

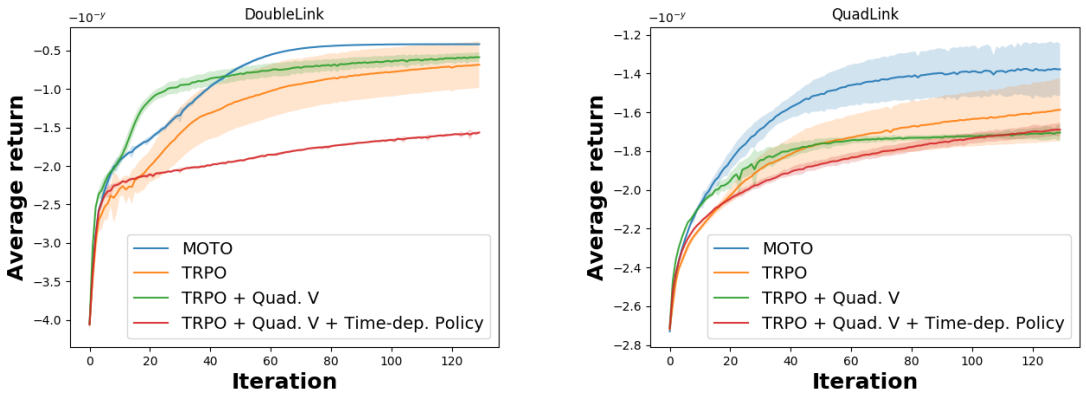


Figure 5: Comparisons on multi-link swing-up tasks between MOTO and TRPO. TRPO uses a neural network policy and V-Function (default) or a quadratic V-Function and a time-dependent linear-Gaussian policy as in MOTO. Quadratic V-Function is a good fit for such tasks and allows MOTO to outperform neural network policies on the double and quadruple link swing-up tasks. Rewards of the original task divided by 1e4 to accomodate with the neural network V-Function. Plots averaged over 11 independent runs.

On simulated robotics tasks, we demonstrated the merits of our approach compared to direct policy search algorithms that optimize commonly used low dimensional parameterized policies. The main strength of our approach is its ability to learn reactive policies capable of adapting to external perturbations in a sample efficient way. However, the exploration scheme of our algorithm based on adding Gaussian noise at every time-step is less structured than that of low dimensional parameterized policies and can be harmful to the robot. One of the main addition that would ease the transition from simulation to physical systems is thus to consider the safety of the exploration scheme of the algorithm. On a more technical note, and as the V-Function can be of any shape in our setting, the use of a more complex function approximator such as a deep network can be considered in future extensions to allow for a more refined bias-variance trade-off.

Acknowledgments

The research leading to these results has received funding from the DFG Project LearnRobotS under the SPP 1527 Autonomous Learning, from the Intel Corporation, and from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 640554 (SKILLS4ROBOTS). Computing time for the experiments was granted from Lichtenberg cluster.

Appendix A. Dual Function Derivations

Recall the quadratic form of the Q-Function $\tilde{Q}_t(\boldsymbol{s}, \boldsymbol{a})$ in the action a and state s

$$\tilde{Q}_t(\boldsymbol{s}, \boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T Q_{aa}\boldsymbol{a} + \boldsymbol{a}^T Q_{as}\boldsymbol{s} + \boldsymbol{a}^T \boldsymbol{q}_a + q(\boldsymbol{s}).$$

The new policy $\pi'_t(\boldsymbol{a}|\boldsymbol{s})$ solution of the constrained maximization problem is again of linear-Gaussian form and given by

$$\pi'_t(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{a}|FL\boldsymbol{s} + F\boldsymbol{f}, F(\eta^* + \omega^*)),$$

such that the gain matrix, bias and covariance matrix of π'_t are function of matrices F and L and vector $\boldsymbol{f}$ where

$$F = (\eta^*\Sigma_t^{-1} - Q_{aa})^{-1}, \qquad L = \eta^*\Sigma_t^{-1}K_t + Q_{as},$$
$$\boldsymbol{f} = \eta^*\Sigma_t^{-1}\boldsymbol{k}_t + \boldsymbol{q}_a,$$

with η^* and ω^* being the optimal Lagrange multipliers related to the KL and entropy constraints, obtained by minimizing the dual function

$$g_t(\eta, \omega) \;=\; \eta\epsilon - \omega\beta + (\eta + \omega)\int \tilde{\rho}_t(\boldsymbol{s}) \log\left(\int \pi(\boldsymbol{a}|\boldsymbol{s})^{\eta/(\eta+\omega)} \exp\left(\tilde{Q}_t(\boldsymbol{s}, \boldsymbol{a})/(\eta + \omega)\right)\right).$$

From the quadratic form of $\tilde{Q}_t(\boldsymbol{s}, \boldsymbol{a})$ and by additionally assuming that the state distribution is approximated by $\tilde{\rho}_t(\boldsymbol{s}) = \mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu}_s, \Sigma_s)$, the dual function simplifies to

$$g_t(\eta, \omega) = \eta\epsilon - \omega\beta + \boldsymbol{\mu}_s^T M \boldsymbol{\mu}_s + \mathrm{tr}(\Sigma_s M) + \boldsymbol{\mu}_s^T \boldsymbol{m} + m_0,$$

where M, $\boldsymbol{m}$ and m_0 are defined by

$$M = \frac{1}{2}\left(L^T F L - \eta K_t^T \Sigma_t^{-1} K_t\right), \; \boldsymbol{m} = L^T F \boldsymbol{f} - \eta K_t^T \Sigma_t^{-1} \boldsymbol{k}_t,$$
$$m_0 = \frac{1}{2}(\boldsymbol{f}^T F \boldsymbol{f} - \eta \boldsymbol{k}_t^T \Sigma_t^{-1} \boldsymbol{k}_t - \eta \log|2\pi\Sigma_t| + (\eta + \omega)\log|2\pi(\eta + \omega)F|).$$

The convex dual function g_t can be efficiently minimized by gradient descent and the policy update is performed upon the computation of η^* and ω^*. The gradient w.r.t. η and ω is given by[5]

$$\begin{aligned}
\frac{\partial g_t(\eta, \omega)}{\partial \eta} &= \mathrm{cst} + \mathrm{lin} + \mathrm{quad} \\
\mathrm{cst} &= \epsilon - \frac{1}{2}\left(\boldsymbol{k}_t - F\boldsymbol{f}\right)^T \Sigma_t^{-1}\left(\boldsymbol{k}_t - F\boldsymbol{f}\right) - \frac{1}{2}[\log|2\pi\Sigma_t| - \log|2\pi(\eta + \omega)F| \\
&\quad + (\eta + \omega)\mathrm{tr}(\Sigma_t^{-1}F) - d_a]. \\
\mathrm{lin} &= \left((K_t - FL)\boldsymbol{\mu}_s\right)^T \Sigma_t^{-1}(F\boldsymbol{f} - \boldsymbol{k}_t). \\
\mathrm{quad} &= \boldsymbol{\mu}_s^T (K_t + FL)^T \Sigma_t^{-1}(K_t + FL)\boldsymbol{\mu}_s + \mathrm{tr}(\Sigma_s(K_t + FL)^T\Sigma_t^{-1}(K_t + FL)) \\
\frac{\partial g_t(\eta, \omega)}{\partial \omega} &= -\beta + \frac{1}{2}(d_a + \log|2\pi(\eta + \omega)F|).
\end{aligned}$$

5. cst, lin, quad, F, L and $\boldsymbol{f}$ all depend on η and ω. We dropped the dependency from the notations for compactness. d_a is the dimensionality of the action.

Appendix B. Bounding the Expected Policy KL Under the Current State Distribution

Let the state distributions and policies be parameterized as following: $p_t(\boldsymbol{s}) = \mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu}_p, \Sigma_p)$, $q_t(\boldsymbol{s}) = \mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu}_q, \Sigma_q)$, $p_t(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{a}|K\boldsymbol{s} + \boldsymbol{b}, \Sigma)$ and $q_t(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{a}|K'\boldsymbol{s} + \boldsymbol{b}', \Sigma')$. The change of the state distribution in the expected KL constraint of our policy update, given by $\mathbb{E}_{\boldsymbol{s}\sim q_t}[\mathrm{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))]$ from state distribution q_t to p_t will only affect the part of the KL that depends on the state.

We give as a reminder the general formula for the KL between two Gaussian distributions $l = \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and $l' = \mathcal{N}(\boldsymbol{\mu}', \Sigma')$

$$\mathrm{KL}(l \parallel l') = \frac{1}{2}\left(\mathrm{tr}(\Sigma'^{-1}\Sigma) + (\boldsymbol{\mu} - \boldsymbol{\mu}')^T\Sigma'^{-1}(\boldsymbol{\mu} - \boldsymbol{\mu}') - dim + \log\frac{|\Sigma'|}{|\Sigma|}\right).$$

For the linear-Gaussian policies, and since only the mean of the policies depend on the state, the change of state distribution in the expected KL will only affect the term

$$(K\boldsymbol{s} - K'\boldsymbol{s})^T\Sigma'(K\boldsymbol{s} - K'\boldsymbol{s}) = \boldsymbol{s}^T M\boldsymbol{s},$$

with p.s.d. matrix $M = (K - K')^T\Sigma'(K - K')$. Thus it suffices to bound the expectation $\int p_t(\boldsymbol{s})\boldsymbol{s}^T M\boldsymbol{s}$ since the rest of the KL terms are already bounded by ϵ, yielding

$$\begin{aligned}\mathbb{E}_{s\sim p_t}[\mathrm{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))] &\leq \epsilon + \frac{1}{2}\int p_t(\boldsymbol{s})\boldsymbol{s}^T M\boldsymbol{s}, \\ &= \epsilon + \frac{1}{2}\left(\boldsymbol{\mu}_p M\boldsymbol{\mu}_p + \mathrm{tr}(M\Sigma_p)\right),\end{aligned}$$

where we exploited the Gaussian nature of p_t in the second line of the equation. We will now bound both $\boldsymbol{\mu}_p M\boldsymbol{\mu}_p$ and $\mathrm{tr}(M\Sigma_p)$. First, note that for any two p.d. matrices Σ and Σ' we have

$$\mathrm{tr}(\Sigma'^{-1}\Sigma) + -d_s + \log\frac{|\Sigma'|}{|\Sigma|} \geq 0. \tag{11}$$

This immediately follows from the non-negativity of the KL. Since, if for some Σ and Σ', eq. (11) is negative then the KL for two Gaussian distributions having Σ and Σ' as covariance matrices and sharing the same mean would be negative which is not possible. Hence it also follows that

$$(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p)^T\Sigma_q^{-1}(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p) \leq 2\epsilon_t, \tag{12}$$

from the bounded KL induction hypothesis between p_t and q_t.

For the expected policy KL, since the part that does not depend on s is positive as in eq. (11), it can thus be dropped out yielding

$$\begin{aligned}\mathbb{E}_{s\sim q_t}[\mathrm{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))] \leq \epsilon &\Rightarrow \int q_t(\boldsymbol{s})\boldsymbol{s}^T M\boldsymbol{s} \leq 2\epsilon, \\ &\Rightarrow \boldsymbol{\mu}_q M\boldsymbol{\mu}_q + \mathrm{tr}(M\Sigma_q) \leq 2\epsilon.\end{aligned} \tag{13}$$

Also note that for any p.s.d. matrices A and B, $\text{tr}(AB) \geq 0$. Letting $\boldsymbol{x} = \boldsymbol{\mu}_p - \boldsymbol{\mu}_q$, we have

$$\begin{aligned}\boldsymbol{x}^T M \boldsymbol{x} &= \text{tr}(\boldsymbol{x}\boldsymbol{x}^T M), \\ &= \text{tr}(\Sigma_q^{-1}\boldsymbol{x}\boldsymbol{x}^T M \Sigma_q), \\ &\leq \text{tr}(\Sigma_q^{-1}\boldsymbol{x}\boldsymbol{x}^T)\text{tr}(M\Sigma_q), \\ &\leq 4\epsilon_t\epsilon.\end{aligned}$$

Third line is due to Cauchy-Schwarz inequality and positiveness of traces while the last one is from eq. (12) and (13). Finally, from the reverse triangular inequality, we have

$$\begin{aligned}\boldsymbol{\mu}_p M \boldsymbol{\mu}_p &\leq \boldsymbol{x}^T M \boldsymbol{x} + \boldsymbol{\mu}_q M \boldsymbol{\mu}_q, \\ &\leq 2\epsilon(1 + 2\epsilon_t),\end{aligned}$$

Which concludes the bounding of $\boldsymbol{\mu}_p M \boldsymbol{\mu}_p$.

To bound $\text{tr}(M\Sigma_p)$ we can write

$$\begin{aligned}\text{tr}(M\Sigma_p) &= \text{tr}(M\Sigma_q\Sigma_q^{-1}\Sigma_p), \\ &\leq \text{tr}(M\Sigma_q)\text{tr}(\Sigma_q^{-1}\Sigma_p).\end{aligned}$$

We know how to bound $\text{tr}(M\Sigma_q)$ from Eq. (13). While $\text{tr}(\Sigma_q^{-1}\Sigma_p)$ appears in the bounded KL between state distributions. Bounding $\text{tr}(\Sigma_q^{-1}\Sigma_p)$ is equivalent to solving $\max \sum \lambda_i$ under constraint $\sum \lambda_i - d_s - \sum \log \lambda_i \leq 2\epsilon_t$, where the $\{\lambda_i\}$ are the eigenvalues of $\Sigma_q^{-1}\Sigma_p$. For any solution $\{\lambda_i\}$, we can keep the same optimization objective using equal $\{\lambda_i'\}$ where for each i, $\lambda_i' = \bar{\lambda} = \sum \lambda_i / d_s$ is the average lambda. This transformation will at the same time reduce the value of the constraint since $-d_s \log \bar{\lambda} \leq -\sum \log \lambda_i$ from Jensen's inequality. Hence the optimum is reached when all the λ_i are equal, and the constraint is active (i.e. $d_s\bar{\lambda} - d_s - d_s \log \bar{\lambda} = 2\epsilon_t$). Finally, the constraint is at a minimum for $\bar{\lambda} = 1$, hence $\bar{\lambda} > 1$. The maximum is reached at

$$\begin{aligned}& d_s\bar{\lambda} - d_s - d_s \log \bar{\lambda} = 2\epsilon_t \\ \Leftrightarrow & \bar{\lambda} - \log \bar{\lambda} = \frac{2\epsilon_t}{d_s} + 1 \\ \Rightarrow & \bar{\lambda} \leq \left(\frac{2\epsilon_t}{d_s} + 1\right)\frac{e}{e-1} \\ \Rightarrow & \text{tr}(\Sigma_q^{-1}\Sigma_p) \leq 4\epsilon_t + 2d_s \qquad (14)\end{aligned}$$

The equation in the second line has a unique solution ($f(\lambda) = \lambda - \log \lambda$ is a strictly increasing function for $\lambda > 1$) for which no closed form expression exists. We thus lower bound f by $g(\lambda) = \frac{e-1}{e}\lambda$ and solve the equation for g which yields an upper bound of the original equation that is further simplified in the last inequality.
Eq. (13) and (14) yield $\text{tr}(M\Sigma_p) \leq 2\epsilon(4\epsilon_t + 2d_s)$ and grouping all the results yields

$$\mathbb{E}_{s\sim p_t}[\text{KL}(p_t(.|\boldsymbol{s}) \parallel q_t(.|\boldsymbol{s}))] \leq 2\epsilon\,(3\epsilon_t + d_s + 1)$$

References

A. Abdolmaleki, R. Lioutikov, J. Peters, N. Lau, L. Pualo Reis, and G. Neumann. Model-based relative entropy stochastic search. In *Advances in Neural Information Processing Systems (NIPS)*. 2015.

R. Akrour, A. Abdolmaleki, H. Abdulsamad, and G. Neumann. Model-free trajectory optimization for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.

D. P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, 1995.

S. Bhojanapalli, A. T. Kyrillidis, and S. Sanghavi. Dropping convexity for faster semi-definite optimization. *CoRR*, 2015.

C. Daniel, G. Neumann, and J. Peters. Hierarchical Relative Entropy Policy Search. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

M. Deisenroth and C. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *International Conference on Machine Learning (ICML)*, 2011.

M. P. Deisenroth, G. Neumann, and J. Peters. A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*, 2013.

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

A. Ijspeert and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems (NIPS)*. 2003.

S. Kakade. *On the Sample Complexity of Reinforcement Learning.* PhD thesis, University College London, 2003.

S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2002.

A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann. Data-efficient generalization of robot skills with contextual policy search. In *The Conference on Artificial Intelligence (AAAI)*, 2013.

S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NIPS)*. 2014.

S. Levine and V. Koltun. Learning complex neural network policies with trajectory optimization. In *International Conference on Machine Learning (ICML)*, 2014.

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*, 2015.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. 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*, 2015.

K. Mülling, J. Kober, and J. Peters. A biomimetic approach to robot table tennis. *Adaptive Behavior Journal*, 2011.

Y. Pan and E. Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*. 2014.

J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. In *Conference on Artificial Intelligence (AAAI)*, 2010.

M. Pirotta, M. Restelli, and L. Bascetta. Adaptive step-size for policy gradient methods. In *Advances in Neural Information Processing Systems (NIPS)*. 2013a.

M. Pirotta, M. Restelli, A. Pecorino, and D. Calandriello. Safe policy iteration. In *International Conference on Machine Learning (ICML)*, 2013b.

E. A. Rückert, M. Mindt, J. Peters, and G. Neumann. Robust policy updates for stochastic optimal control. In *International Conference on Humanoid Robots (Humanoids)*, 2014.

J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016.

C. Szepesvari. *Algorithms for Reinforcement Learning*. Morgan & Claypool, 2010.

E. Theodorou, J. Buchli, and S. Schaal. Path integral stochastic optimal control for rigid body dynamics. In *International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, 2009.

E. Theodorou, Y. Tassa, and E. Todorov. Stochastic differential dynamic programming. In *American Control Conference (ACC)*, 2010.

E. Todorov. Optimal control theory. *Bayesian Brain*, 2006.

E. Todorov and Y. Tassa. Iterative local dynamic programming. In *International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2009.

M. Toussaint. Robot trajectory optimization using approximate inference. In *International Conference on Machine Learning (ICML)*, 2009.

P. Wagner. A reinterpretation of the policy oscillation phenomenon in approximate policy iteration. In *Advances in Neural Information Processing Systems (NIPS)*. 2011.