Combining Model-Based and Model-Free Updates for Deep Reinforcement Learning

Yevgen Chebotar*†, Karol Hausman*†, Marvin Zhang*‡, Gaurav Sukhatme†, Stefan Schaal†, Sergey Levine‡
†Department of Computer Science, University of Southern California, Los Angeles, CA, USA
‡Department of Electrical Engineering and Computer Science, University of California Berkeley, Berkeley, CA, USA

I. Introduction

The ability to learn motor skills autonomously is one of the main requirements for deploying robots in unstructured real-world environments. The goal of reinforcement learning (RL) is to learn such skills through trial and error, thus avoiding tedious manual engineering. However, real-world applications of RL have to contend with two often opposing requirements: data-efficient learning and the ability to handle complex, unknown dynamical systems that might be difficult to model explicitly. Real-world physical systems, such as robots, are typically costly and time consuming to run, making it highly desirable to learn using the lowest possible number of real-world trials. Model-based methods tend to excel at this [5], but suffer from significant bias, since complex unknown dynamics cannot always be modeled accurately enough to produce effective policies. Model-free methods have the advantage of handling arbitrary dynamical systems with minimal bias, but tend to be substantially less sample-efficient [9, 17]. Can we combine the efficiency of model-based algorithms with the final performance of model-free algorithms in a method that we can practically use on real-world physical systems?

Many prior methods that combine model-free and model-based techniques achieve only modest gains in efficiency or performance [6, 7]. In this work, we aim to develop a method in the context of a specific policy representation: time-varying linear-Gaussian controllers. The structure of these policies provides us with an effective option for model-based updates via iterative linear-Gaussian dynamics fitting [10], as well as a simple option for model-free updates via the path integral policy improvement (PI^2) algorithm [19].

Although time-varying linear-Gaussian (TVLG) policies are not as powerful as representations such as deep neural networks [13, 14] or RBF networks [4], they can represent arbitrary trajectories in continuous state-action spaces. Furthermore, prior work on guided policy search (GPS) has shown that TVLG policies can be used to train general-purpose parameterized policies, including deep neural network policies, for tasks involving complex sensory inputs such as vision [10, 12]. This yields a general-purpose RL procedure with favorable stability and sample complexity compared to fully model-free deep RL methods [16].

The main contribution of this paper is a procedure for optimizing TVLG policies that integrates both fast model-based updates via iterative linear-Gaussian model fitting and corrective model-free updates via the PI^2 framework. The resulting algorithm, which we call PILQR, combines the efficiency of model-based learning with the generality of model-free updates and can solve complex continuous control tasks that are infeasible for either linear-Gaussian models or PI^2 by itself, while remaining orders of magnitude more efficient than standard model-free RL. We demonstrate that this approach can be integrated into GPS to train deep neural network policies and present empirical results both in simulation and on a real robotic platform. Our real-world results demonstrate that our method can learn complex tasks, such as hockey and power plug plugging (see Figure 1), each with less than an hour of experience and no user-provided demonstrations.

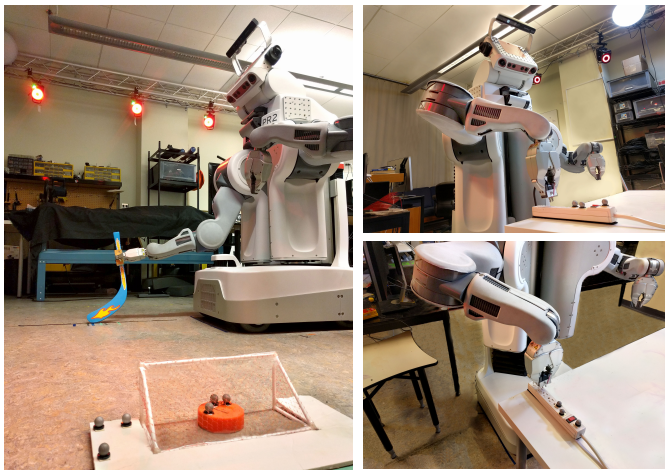

Fig. 1. Real robot tasks used to evaluate our method. Left: The hockey task which involves discontinuous dynamics. Right: The power plug task which requires high level of precision. Both of these tasks are learned from scratch without demonstrations.

II. Preliminaries

The goal of policy search methods is to optimize the parameters θ of a policy $p(\mathbf{u}_t|\mathbf{x}_t)$, which defines a probability distribution over actions $\mathbf{u}_t$ conditioned on the system state $\mathbf{x}_t$ at each time step t of a task execution. Let $\tau = (\mathbf{x}_1, \mathbf{u}_1, \ldots, \mathbf{x}_T, \mathbf{u}_T)$ be a trajectory of states and actions. Given a cost function $c(\mathbf{x}_t, \mathbf{u}_t)$, we define the trajectory cost as $c(\tau) = \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t)$. The policy is optimized with respect to the expected cost of the policy

$$J(\theta) = \mathbb{E}_p\left[c(\tau)\right] = \int c(\tau)p(\tau)d\tau\,,$$

* equal contribution

where $p(\tau)$ is the policy trajectory distribution given the system dynamics $p\left(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t\right)$

$$p(\tau) = p(\mathbf{x}_1) \prod_{t=1}^{T} p\left(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t\right) p(\mathbf{u}_t|\mathbf{x}_t)\,.$$

A particular policy class that allows us to employ an efficient model-based update is the TVLG controller $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{\Sigma}_t)$. In this section, we present the model-based and model-free algorithms that form the constituent parts of our hybrid method. The model-based method is an extension of a KL-constrained LQR algorithm [10], which we shall refer to as LQR with fitted linear models (LQR-FLM). The model-free method is a PI^2 algorithm with per-time step KL-divergence constraints that is derived in previous work [2].

A. Model-Based Optimization of TVLG Policies

The model-based method we use is based on the iterative linear-quadratic regulator (iLQR) and builds on prior work [10, 18]. Hereby, we briefly outline the method.

We use samples to fit a TVLG dynamics model $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{f}_{\mathbf{x},t}\mathbf{x}_t + \mathbf{f}_{\mathbf{u},t}\mathbf{u}_t, \mathbf{F}_t)$ and assume a twice-differentiable cost function. Tassa et al. [18] showed that we can compute a second-order Taylor approximation of our Q-function and optimize this with respect to $\mathbf{u}_t$ to find the optimal action at each time step t. To deal with unknown dynamics, Levine & Abbeel [10] impose a KL-divergence constraint between the updated policy $p^{(i)}$ and previous policy $p^{(i-1)}$ to stay within the space of trajectories where the dynamics model is approximately correct. We similarly set up our optimization as

$$\min_{p^{(i)}} \; \mathbb{E}_{p^{(i)}}[Q(\mathbf{x}_t, \mathbf{u}_t)] \; s.t. \; \mathbb{E}_{p^{(i)}}\left[D_{\mathrm{KL}}(p^{(i)} \| p^{(i-1)})\right] \leq \epsilon_t\,. \quad (1)$$

The main difference from Levine & Abbeel [10] is that we enforce separate KL constraints for each linear-Gaussian policy rather than a single constraint on the induced trajectory distribution (i.e., compare Eq. (1) to the first equation in Section 3.1 of Levine & Abbeel [10]).

LQR-FLM has substantial efficiency benefits over model-free algorithms. However, as our experimental results in Section V show, the performance of LQR-FLM is highly dependent on being able to model the system dynamics accurately, causing it to fail for more challenging tasks.

B. Policy Improvement with Path Integrals

PI^2 is a model-free RL algorithm based on stochastic optimal control. A detailed derivation of this method can be found in [19].

Each iteration of PI^2 involves generating N trajectories by running the current policy. Let $S(\mathbf{x}_{i,t}, \mathbf{u}_{i,t}) = c(\mathbf{x}_{i,t}, \mathbf{u}_{i,t}) + \sum_{j=t+1}^{T} c(\mathbf{x}_{i,j}, \mathbf{u}_{i,j})$ be the cost-to-go of trajectory $i \in \{1, \ldots, N\}$ starting in state $\mathbf{x}_{i,t}$ by performing action $\mathbf{u}_{i,t}$ and following the policy $p(\mathbf{u}_t|\mathbf{x}_t)$ afterwards. Then, we can compute probabilities $P(\mathbf{x}_{i,j}, \mathbf{u}_{i,j})$ for each trajectory starting at time step t

$$P(\mathbf{x}_{i,t}, \mathbf{u}_{i,t}) = \frac{\exp\left(-\frac{1}{\eta_t} S(\mathbf{x}_{i,t}, \mathbf{u}_{i,t})\right)}{\int \exp\left(-\frac{1}{\eta_t} S(\mathbf{x}_{i,t}, \mathbf{u}_{i,t})\right) d\mathbf{u}_{i,t}}\,. \quad (2)$$

The probabilities follow from the Feynman-Kac theorem applied to stochastic optimal control [19]. The intuition is that the trajectories with lower costs receive higher probabilities, and the policy distribution shifts towards a lower cost trajectory region. The costs are scaled by η_t, which can be interpreted as the temperature of a soft-max distribution. This is similar to the dual variables η_t in LQR-FLM in that they control the KL step size, however they are derived and computed differently. After computing the new probabilities P, we update the policy distribution by reweighing each sampled control $\mathbf{u}_{i,t}$ by $P(\mathbf{x}_{i,t}, \mathbf{u}_{i,t})$ and updating the policy parameters by a maximum likelihood estimate [2].

To relate PI^2 updates to LQR-FLM optimization of a constrained objective, which is necessary for combining these methods, we can formulate the following theorem.

Theorem 1. *The PI^2 update corresponds to a KL-constrained minimization of the expected cost-to-go $S(\mathbf{x}_t, \mathbf{u}_t) = \sum_{j=t}^{T} c(\mathbf{x}_j, \mathbf{u}_j)$ at each time step t*

$$\min_{p^{(i)}} \; \mathbb{E}_{p^{(i)}}[S(\mathbf{x}_t, \mathbf{u}_t)] \; s.t. \; \mathbb{E}_{p^{(i-1)}}\left[D_{KL}\left(p^{(i)} \| \, p^{(i-1)}\right)\right] \leq \epsilon,$$

where ϵ is the maximum KL-divergence between the new policy $p^{(i)}\left(\mathbf{u}_t|\mathbf{x}_t\right)$ and the old policy $p^{(i-1)}\left(\mathbf{u}_t|\mathbf{x}_t\right)$.

Proof: The Lagrangian of this problem is given by

$$\mathscr{L}(p^{(i)}, \eta_t) = \mathbb{E}_{p^{(i)}}[S(\mathbf{x}_t, \mathbf{u}_t)] + \eta_t \mathbb{E}_{p^{(i-1)}}\left[D_{\mathrm{KL}}\left(p^{(i)} \| \, p^{(i-1)}\right) - \epsilon\right]$$

By minimizing the Lagrangian with respect to the new policy we can find its relationship to the old policy:

$$p^{(i)}(\mathbf{u}_t|\mathbf{x}_t) \propto p^{(i-1)}(\mathbf{u}_t|\mathbf{x}_t)\, \mathbb{E}_{p^{(i-1)}}\left[\exp\left(-\frac{1}{\eta_t} S(\mathbf{x}_t, \mathbf{u}_t)\right)\right] \quad (3)$$

This gives us an update rule for $p^{(i)}$ that corresponds exactly to reweighing the controls from the previous policy $p^{(i-1)}$ based on their probabilities $P(\mathbf{x}_t, \mathbf{u}_t)$ described earlier. The temperature η_t now corresponds to the dual variable of the KL-divergence constraint. ■

III. Integrating Model-Based Updates into PI^2

A. Two-Stage PI^2 update

To integrate a model-based optimization into PI^2, we can divide it into two steps. Given an approximation $\hat{c}(\mathbf{x}_t, \mathbf{u}_t)$ of the real cost $c(\mathbf{x}_t, \mathbf{u}_t)$ and the residual cost $\tilde{c}(\mathbf{x}_t, \mathbf{u}_t) = c(\mathbf{x}_t, \mathbf{u}_t) - \hat{c}(\mathbf{x}_t, \mathbf{u}_t)$, let $\hat{S}_t = \hat{S}(\mathbf{x}_t, \mathbf{u}_t)$ be the approximated cost-to-go of a trajectory starting with state $\mathbf{x}_t$ and action $\mathbf{u}_t$, and $\tilde{S}_t = \tilde{S}(\mathbf{x}_t, \mathbf{u}_t)$ be the residual of the real cost-to-go $S(\mathbf{x}_t, \mathbf{u}_t)$ after approximation. We can rewrite the PI^2 policy update rule from Eq. (3) as

$$\begin{aligned} p^{(i)}\left(\mathbf{u}_t|\mathbf{x}_t\right) & \\ & \propto p^{(i-1)}\left(\mathbf{u}_t|\mathbf{x}_t\right) \mathbb{E}_{p^{(i-1)}}\left[\exp\left(-\frac{1}{\eta_t}\left(\hat{S}_t + \tilde{S}_t\right)\right)\right] \\ & \propto \hat{p}\left(\mathbf{u}_t|\mathbf{x}_t\right) \mathbb{E}_{p^{(i-1)}}\left[\exp\left(-\frac{1}{\eta_t}\tilde{S}_t\right)\right], \end{aligned} \quad (4)$$

where $\hat{p}(\mathbf{u}_t|\mathbf{x}_t)$ is given by

$$\hat{p}(\mathbf{u}_t|\mathbf{x}_t) \propto p^{(i-1)}(\mathbf{u}_t|\mathbf{x}_t)\, \mathbb{E}_{p^{(i-1)}}\left[\exp\left(-\frac{1}{\eta_t}\hat{S}_t\right)\right] \quad (5)$$

Hence, by decomposing the cost into its approximation and the residual approximation error, the PI^2 update can be split into two steps: (1) update using the approximated costs $\hat{c}(\mathbf{x}_t, \mathbf{u}_t)$ and samples from the old policy $p^{(i-1)}(\mathbf{u}_t|\mathbf{x}_t)$ to get $\hat{p}(\mathbf{u}_t|\mathbf{x}_t)$; (2) update $p^{(i)}(\mathbf{u}_t|\mathbf{x}_t)$ using the residual costs $\tilde{c}(\mathbf{x}_t, \mathbf{u}_t)$ and samples from $\hat{p}(\mathbf{u}_t|\mathbf{x}_t)$.

B. Model-Based Substitution with LQR-FLM

We can use Theorem (1) to rewrite Eq. (5) as a constrained optimization problem

$$\min_{\hat{p}} \quad \mathbb{E}_{\hat{p}}\left[\hat{S}(\mathbf{x}_t, \mathbf{u}_t)\right] s.t. \quad \mathbb{E}_{p^{(i-1)}}\left[D_{\mathrm{KL}}\left(\hat{p} \| \, p^{(i-1)}\right)\right] \leq \epsilon\,.$$

Thus, the policy $\hat{p}(\mathbf{u}_t|\mathbf{x}_t)$ can be updated using any algorithm that can solve this optimization problem. By choosing a model-based approach for this, we can speed up the learning process significantly. Model-based methods are typically constrained to some particular cost approximation, however, PI^2 can accommodate any form of $\tilde{c}(\mathbf{x}_t, \mathbf{u}_t)$ and thus will handle arbitrary cost residuals.

LQR-FLM solves the type of constrained optimization problem in Eq. (1), which matches the optimization problem needed to obtain $\hat{p}$, where the cost-to-go $\hat{S}$ is approximated with a quadratic cost and a linear-Gaussian dynamics model.[1] We can thus use LQR-FLM to perform our first update, which enables greater efficiency but is susceptible to modeling errors when the fitted local dynamics are not accurate, such as in discontinuous systems. We can use a PI^2 optimization on the residuals to correct for this bias.

C. Optimizing Cost Residuals with PI^2

In order to perform a PI^2 update on the residual costs-to-go $\tilde{S}$, we need to know what $\hat{S}$ is for each sampled trajectory. That is, what is the cost-to-go that is actually used by LQR-FLM to make its update? The structure of the algorithm implies a specific cost-to-go formulation for a given trajectory – namely, the sum of quadratic costs obtained by running the same policy under the TVLG dynamics used by LQR-FLM. A given trajectory can be viewed as being generated by a deterministic policy conditioned on a particular noise realization $\xi_{i,1}, \ldots, \xi_{i,T}$, with actions given by

$$\mathbf{u}_{i,t} = \mathbf{K}_t \mathbf{x}_{i,t} + \mathbf{k}_t + \sqrt{\mathbf{\Sigma}_t}\xi_{i,t}\,, \quad (6)$$

where $\mathbf{K}_t$, $\mathbf{k}_t$, and $\mathbf{\Sigma_t}$ are the parameters of $p^{(i-1)}$. We can therefore evaluate $\hat{S}(\mathbf{x}_t, \mathbf{u}_t)$ by simulating this deterministic controller from $(\mathbf{x}_t, \mathbf{u}_t)$ under the fitted TVLG dynamics and evaluating its time-varying quadratic cost, and then plugging these values into the residual cost.

[1]In practice, we make a small modification to the problem in Eq. (1) so that the expectation in the constraint is evaluated with respect to the new distribution $\hat{p}(\mathbf{x}_t)$ rather than the previous one $p^{(i-1)}(\mathbf{x}_t)$. This modification is heuristic and no longer aligns with Theorem (1), but works better in practice.

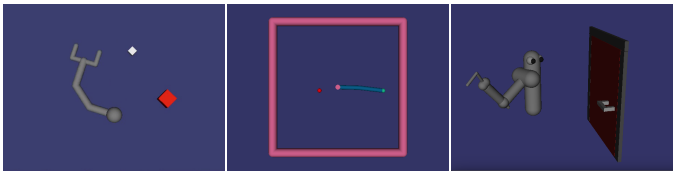

Fig. 2. We evaluate on a set of simulated robotic manipulation tasks with varying difficulty. Left to right, the tasks involve pushing a block, reaching for a target, and opening a door in 3D.

In addition to the residual costs $\tilde{S}$ for each trajectory, the PI^2 update also requires control samples from the updated LQR-FLM policy $\hat{p}(\mathbf{u}_t|\mathbf{x}_t)$. Although we have the updated LQR-FLM policy, we only have samples from the old policy $p^{(i-1)}(\mathbf{u}_t|\mathbf{x}_t)$. However, we can apply a form of the re-parametrization trick [8] and again use the stored noise realization of each trajectory $\xi_{t,i}$ to evaluate what the control would have been for that sample under the LQR-FLM policy $\hat{p}$. The expectation of the residual cost-to-go in Eq. (4) is taken with respect to the old policy distribution $p^{(i-1)}$. Hence, we can reuse the states $\mathbf{x}_{i,t}$ and their corresponding noise $\xi_{i,t}$ that was sampled while rolling out the previous policy $p^{(i-1)}$ and evaluate the new controls according to $\hat{\mathbf{u}}_{i,t} = \hat{\mathbf{K}}_t \mathbf{x}_{i,t} + \hat{\mathbf{k}}_t + \sqrt{\hat{\mathbf{\Sigma}}_t}\xi_{i,t}$. This linear transformation on the sampled control provides unbiased samples from $\hat{p}(\mathbf{u}_t|\mathbf{x}_t)$. After transforming the control samples, they are reweighed according to their residual costs and plugged into the PI^2 update in Eq. (2).

IV. Training Parametric Policies with GPS

PILQR offers an approach to perform trajectory optimization of TVLG policies. In this work, we employ mirror descent guided policy search (MDGPS) [15] in order to use PILQR to train parametric policies, such as neural networks. Instead of directly learning the parameters of a high-dimensional parametric or "global policy" with RL, we first learn simple TVLG policies, which we refer to as "local policies" $p(\mathbf{u}_t|\mathbf{x}_t)$ for various initial conditions of the task. After optimizing the local policies, the optimized controls from these policies are used to create a training set for learning the global policy π_θ in a supervised manner. Hence, the final global policy generalizes across multiple local policies.

Using the TVLG representation of the local policies makes it straightforward to incorporate PILQR into the MDGPS framework. Instead of constraining against the old local TVLG policy as in Theorem (1), each instance of the local policy is now constrained against the old global policy

$$\min_{p^{(i)}} \quad \mathbb{E}_{p^{(i)}}[S(\mathbf{x}_t, \mathbf{u}_t)] s.t. \quad \mathbb{E}_{p^{(i-1)}}\left[D_{\mathrm{KL}}\left(p^{(i)} \| \, \pi_\theta^{(i-1)}\right)\right] \leq \epsilon\,.$$

The two-stage update prfoceeds as described in Section III-A, with the change that the LQR-FLM policy is now constrained against the old global policy $\pi_\theta^{(i-1)}$.

V. Experimental Evaluation

Our experiments aim to answer the following questions: (1) How does our method compare to other trajectory-centric

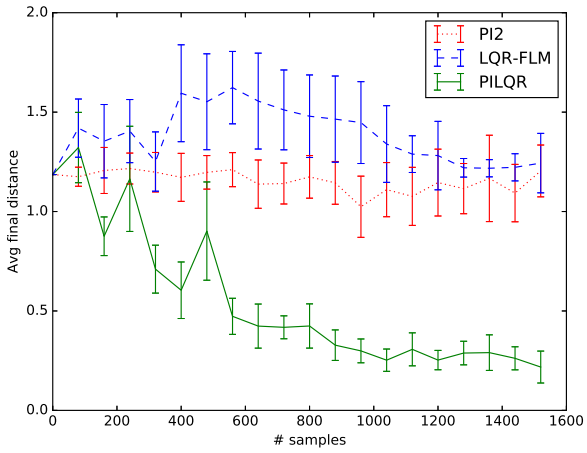

Fig. 3. Average final distance from the block to the goal on one condition of the gripper pusher task. This condition is difficult due to the block being initialized far away from the gripper and the goal area, and only PILQR is able to succeed in reaching the block and pushing it toward the goal.

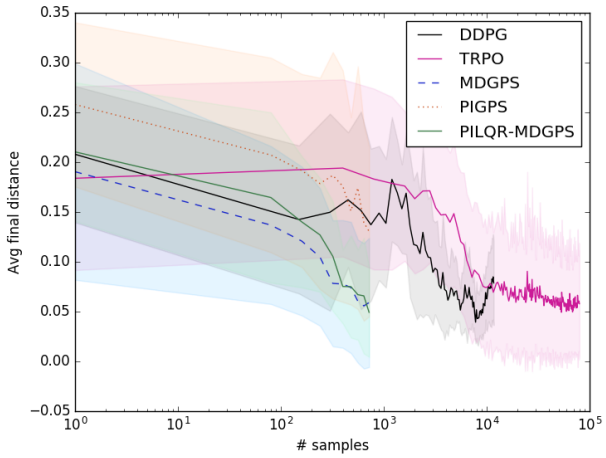

Fig. 4. Final distance from the reacher end effector to the target averaged across 300 random test conditions per iteration. MDGPS with LQR-FLM, MDGPS with PILQR, TRPO, and DDPG all perform competitively, however, as the log scale for the x axis shows, TRPO and DDPG requires orders of magnitude more samples. MDGPS with PI^2 performs noticeably worse.

and deep RL algorithms in terms of final performance and sample efficiency? (2) Can we utilize linear-Gaussian policies trained using PILQR to obtain robust neural network policies using MDGPS? (3) Is our proposed algorithm capable of learning complex manipulation skills on a real robotic platform? We study these questions through a set of simulated comparisons against prior methods, as well as real-world tasks using a PR2 robot. The performance of each method can be seen in our supplementary video.[2] Our focus in this work is specifically on robotics tasks that involve manipulation of objects, since such tasks often exhibit elements of continuous and discontinuous dynamics and require sample-efficient methods, making them challenging for both model-based and model-free methods.

A. Simulation Experiments

We evaluate our method on three simulated robotic manipulation tasks, depicted in Figure 2 and discussed below:

Gripper pusher. This task involves controlling a 4 DoF arm with a gripper to push a white block to a red goal area. The cost function is a weighted combination of the distance from the gripper to the block and from the block to the goal.

Reacher. The reacher task from OpenAI gym [1] requires moving the end of a 2 DoF arm to a target position. This task is included to provide comparisons against prior methods. The cost function is the distance from the end effector to the target. We modify the cost function slightly, since the original task from OpenAI Gym uses an ℓ_2 norm, while we use a differentiable Huber-style loss on the distance, which is more typical for LQR-based methods [18].

Door opening. This task requires opening a door with a 6 DoF 3D arm. The arm must grasp the handle and pull the door to a target angle, which can be particularly challenging for model-based methods due to the complex contacts between the hand and the handle, and the fact that a contact must be established before the door can be opened. The cost function is a weighted combination of the distance of the end effector to the door handle and the angle of the door.

We first compare PILQR to LQR-FLM and PI^2 on the gripper pusher and door opening tasks. Figure 3 details performance of each method on the most difficult condition for the gripper pusher task. Both LQR-FLM and PI^2 perform significantly worse on the two more difficult conditions of this task. While PI^2 improves in performance as we provide more samples, LQR-FLM is bounded by its ability to model the dynamics, and thus predict the costs, at the moment when the gripper makes contact with the block. Our method solves all four conditions with 400 total episodes per condition and, as shown in the supplementary video, is able to learn a diverse set of successful behaviors including flicking, guiding, and hitting the block. On the door opening task, PILQR trains TVLG policies that succeed at opening the door from each of four initial robot positions. While the policies trained with LQR-FLM are able to reach the handle, they fail to open the door.

Next we evaluate neural network policies on the reacher task. Figure 4 shows results for MDGPS with each local policy method, as well as two prior deep RL methods that directly learn neural network policies: trust region policy optimization (TRPO) [17] and deep deterministic policy gradient (DDPG) [13]. MDGPS with LQR-FLM and MDGPS with PILQR perform competitively in terms of the final distance from the end effector to the target, which is unsurprising given the simplicity of the task, whereas MDGPS with PI^2 is again not able to make much progress. On the reacher task, DDPG and TRPO use 25 and 150 times more samples, respectively, to achieve approximately the same performance as MDGPS

[2]https://sites.google.com/site/icml17pilqr

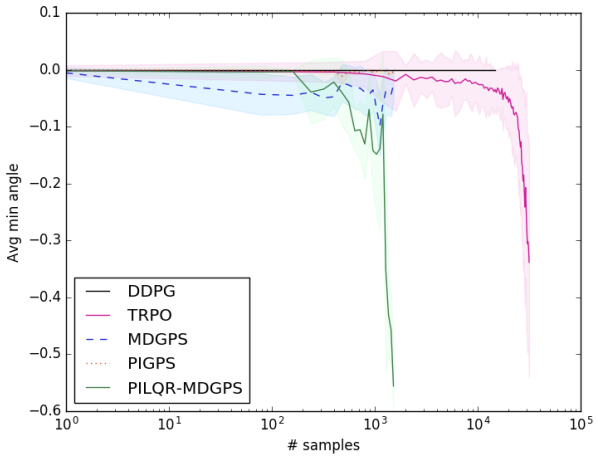

Fig. 5. Minimum angle in radians of the door hinge (lower is better) averaged across 100 random test conditions per iteration. MDGPS with PILQR outperforms all other methods we compare against, with orders of magnitude fewer samples than DDPG and TRPO, which is the only other successful algorithm.

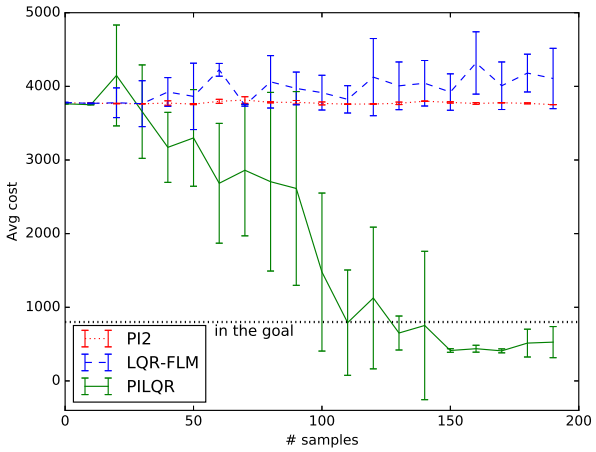

Fig. 6. Single instance comparison of the hockey task performed on the real robot. Costs lower than the dotted line correspond to the puck entering the goal.

with LQR-FLM and PILQR. For comparison, amongst previous deep RL algorithms that combined model-based and model-free methods, SVG and NAF with imagination rollouts reported using approximately up to five times fewer samples than DDPG on a similar reacher task [6, 7]. Thus we can expect that MDGPS with our method is about one order of magnitude more sample-efficient than SVG and NAF. While this is a rough approximation, it demonstrates a significant improvement in efficiency.

Finally, we compare the same methods for training neural network policies on the door opening task, shown in Figure 5. TRPO requires 20 times more samples than MDGPS with PILQR to learn a successful neural network policy. The other three methods were unable to learn a policy that opens the door despite extensive hyperparameter tuning.

B. Real Robot Experiments

To evaluate our method on a real robotic platform, we use a PR2 robot (see Figure 1) to learn the following tasks:

Hockey. The hockey task requires using a stick to hit a puck into a goal $1.4\,\mathrm{m}$ away. The cost function consists of two parts: the distance between the current position of the stick and a target pose that is close to the puck, and the distance between the position of the puck and the goal. The puck is tracked using a motion capture system. Although the cost provides some shaping, this task presents a significant challenge due to the difference in outcomes based on whether or not the robot actually strikes the puck, making it challenging for prior methods, as we show below.

Power plug plugging. In this task, the robot must plug a power plug into an outlet. The cost function is the distance between the plug and a target location inside the outlet. This task requires fine manipulation to fully insert the plug.

Both of these tasks have difficult, discontinuous dynamics at the contacts between the objects, and both require a high degree of precision to succeed. In contrast to prior works [3] that use kinesthetic teaching to initialize a policy that is then finetuned with model-free methods, our method does not require any human demonstrations. In all of the real robot experiments, policies are updated every 10 rollouts and the final policy is obtained after 20-25 iterations, which corresponds to mastering the skill with less than one hour of experience.

In the first set of experiments, we aim to learn a policy that is able to hit the puck into the goal for a single position of the goal and the puck. The results of this experiment are shown in Figure 6. In the case of the prior PI^2 method [19], the robot was not able to hit the puck. Since the puck position has the largest influence on the cost, the resulting learning curve shows little change in the cost over the course of training. The policy to move the arm towards the recorded arm position that enables hitting the puck turned out to be too challenging for PI^2 in the limited number of trials used for this experiment. In the case of LQR-FLM, the robot was able to occasionally hit the puck in different directions. However, the resulting policy could not capture the complex dynamics of the sliding puck or the discrete transition, and was unable to hit the puck toward the goal. The PILQR method was able to learn a robust policy that consistently hits the puck into the goal. Using the step adjustment rule, the algorithm would shift towards model-free updates from the PI^2 method as the TVLG approximation of the dynamics became less accurate. Using our method, the robot was able to get to the final position of the arm using fast model-based updates from LQR-FLM and learn the puck-hitting policy, which is difficult to model, by automatically shifting towards model-free PI^2 updates.

In our second set of hockey experiments, we evaluate whether we can learn a neural network policy using the

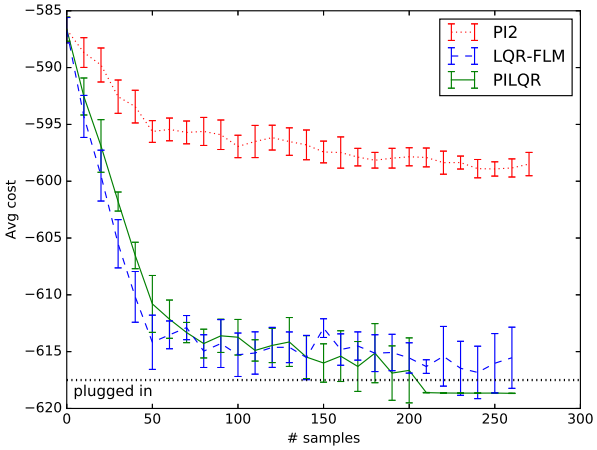

Fig. 7. Single instance comparison of the power plug task performed on the real robot. Note that costs above the dotted line correspond to executions that did not actually insert the plug into the socket. Only our method (PILQR) was able to consistently insert the plug all the way into the socket by the final iteration.

MDGPS-PILQR algorithm that can hit the puck into different goal locations. The goals were spaced $0.5\,\mathrm{m}$ apart. The strategies for hitting the puck into different goal positions differ substantially, since the robot must adjust the arm pose to approach the puck from the right direction and aim toward the target. This makes it quite challenging to learn a single policy for this task. We performed 30 rollouts for three different positions of the goal (10 rollouts each), two of which were used during training. The neural network policy was able to hit the puck into the goal in 90% of the cases. This shows that our method can learn high-dimensional neural network policies that generalize across various conditions.

The results of the plug experiment are shown in Figure 7. PI^2 alone was unable to reach the socket. The LQR-FLM algorithm succeeded only 60% of the time at convergence. In contrast to the peg insertion-style tasks evaluated in prior work that used LQR-FLM [11], this task requires very fine manipulation due to the small size of the plug. Our method was able to converge to a policy that plugged in the power plug on every rollout at convergence. The supplementary video illustrates the final behaviors of each method for both the hockey and power plug tasks.[3]

VI. Conclusion

We presented an algorithm that combines elements of model-free and model-based RL, with the aim of combining the sample efficiency of model-based methods with the ability of model-free methods to improve the policy even in situations where the model's structural assumptions are violated. We further demonstrate that, although this algorithm is specific to TVLG policies, it can be integrated into the GPS framework in order to train arbitrary parameterized policies, including deep neural networks.

[3]https://sites.google.com/site/icml17pilqr

References

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

[2] Chebotar, Y., Kalakrishnan, M., Yahya, A., Li, A., Schaal, S., and Levine, S. Path integral guided policy search. In *ICRA*, 2017.

[3] Daniel, Christian, Neumann, Gerhard, Kroemer, Oliver, and Peters, Jan. Learning sequential motor tasks. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 2626–2632. IEEE, 2013.

[4] Deisenroth, M., Rasmussen, C., and Fox, D. Learning to control a low-cost manipulator using data-efficient reinforcement learning. In *RSS*, 2011.

[5] Deisenroth, M., Neumann, G., and Peters, J. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2): 1–142, 2013.

[6] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. Continuous deep Q-learning with model-based acceleration. *CoRR*, abs/1603.00748, 2016.

[7] Heess, N., Wayne, G., Silver, D., Lillicrap, T., Tassa, Y., and Erez, T. Learning continuous control policies by stochastic value gradients. In *NIPS*, 2015.

[8] Kingma, D. and Welling, M. Auto-encoding variational Bayes. *CoRR*, abs/1312.6114, 2013.

[9] Kober, J., Bagnell, J., and Peters, J. Reinforcement learning in robotics: a survey. *International Journal of Robotic Research*, 32(11):1238–1274, 2013.

[10] Levine, S. and Abbeel, P. Learning neural network policies with guided policy search under unknown dynamics. In *NIPS*, 2014.

[11] Levine, S., Wagener, N., and Abbeel, P. Learning contact-rich manipulation skills with guided policy search. In *ICRA*, 2015.

[12] Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *JMLR*, 17(1), 2016.

[13] Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[14] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with deep reinforcement learning. In *NIPS Workshop on Deep Learning*, 2013.

[15] Montgomery, W. and Levine, S. Guided policy search via approximate mirror descent. In *NIPS*, pp. 4008–4016, 2016.

[16] Montgomery, W., Ajay, A., Finn, C., Abbeel, P., and Levine, S. Reset-free guided policy search: efficient deep reinforcement learning with stochastic initial states. In *ICRA*, 2017.

[17] Schulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. Trust region policy optimization. In *ICML*, 2015.

[18] Tassa, Y., Erez, T., and Todorov, E. Synthesis and stabilization of complex behaviors. In *IROS*, 2012.

[19] Theodorou, E., Buchli, J., and Schaal, S. A generalized path integral control approach to reinforcement learning. *JMLR*, 11, 2010.