2011 50th IEEE Conference on Decision and Control and
European Control Conference (CDC-ECC)
Orlando, FL, USA, December 12-15, 2011

PRISCA: A Policy Search Method for Extreme Trajectory Following

Tak Kit Lau

Abstract— **Consider slide parking, given a desired demonstration, how to repeat it accurately? Many robotics tasks, such as slide parking, can be formulated in trajectory following, but not many dynamics of which can be easily modeled to facilitate a solving by the optimal control. Although an emerging stream in robotics is to learn the dynamics and policy from demonstrations, multiple, if not numerous, demonstrations are required. Therefore, learning a policy from scarce experience remains a difficult problem. In this paper, we proposed an online algorithm to learn a policy for control using only a desired demonstration and our intuitive knowledge of the dynamics system. Our approach is found on this observation: For trajectory following, even on a highly nonlinear and coupled dynamical system, so long as the state deviation is initially small, a policy can be updated online to keep the robot on track according to a very obvious and coarse model information (e.g., for driving, this information is simply: steer left to turn left). Our policy search is then devised as a function minimization problem, and is solved by gradient descent using the techniques of optimal baseline, least-state-deviation error, smoothing and in an inverse depreciation as a cost intensifier. Apart from guarantees on performance and convergence, we also demonstrated its performance in two simulations, and an *extreme* trajectory-following scenario – four-wheel-drive slide parking experiment. To our best knowledge, it is the state-of-the-art autonomous precision slide parking of a 4x4 brakeless RC car.**

I. INTRODUCTION

Given sufficient expert's demonstrations, an optimal policy can be exactly known in the Markov Decision Process (MDP) formalism. These learn-from-demonstration algorithms in reinforcement learning typically assume a stochastic or a deterministic policy, and to generate a policy either by an explorative drawing of control inputs from a probabilistic distribution, or by an exploitation of some proper system knowledge. Although this formalism could be useful for many robotic tasks, such as trajectory following, we believe that even obtaining a number of expert's demonstrations of the same task for learning is redundant. Consider driving, given no prior policy and dynamic model but only a very rough model information (e.g., steer left to turn left) and a desired trajectory, a learning driver can gradually improve the policy if he/she continuously has access to the last few state-action pairs – i.e., when a deviation to the left is observed, this learning driver will gradually turn right to correct it, based on his/her rough understanding on the actuation and dynamics.

This feedback-and-learn mechanism is more vital than finding the underlying expert's reward/cost that tells the hidden desiderata. In practice, many robotics tasks can be formulated in trajectory following. Then, when a desired trajectory is given, the fundamental concern is to directly learn a policy that keeps the agent on track, rather to retrieve some subtle rewards/costs that answer "How to drive well?". Therefore, when such a learning-to-control problem is formulated without retrieving a reward/cost function but in a feedback-and-learn mechanism, we can achieve not only a model-free learning, but also an *undelayed* learning for control because the only prerequisites are a single desired demonstration and an intuitive understanding of the system.

In this paper, we proposed PRISCA (for Policy Reasoning In State-and-Cost Adaptation), which is a policy search method that makes use of a single desired demonstration and our intuitive understanding on a dynamical system for trajectory following. Our approach assumes that the policy is a linear combination of some policy parameters and state features. The objective is to minimize a H-step cost that is composed primarily of the square of a state-driven tracking error. By leveraging our intuitive knowledge on the system and a gradient descent update rule with the techniques of optimal baseline, least-state-deviation error, smoothing and an inverse depreciating function as a cost intensifier, we boosted the notoriously slow and noise-susceptible gradient update to become a fast online learning algorithm. Apart from theoretical guarantees on performance and convergence, we also demonstrated our approach in two simulations and an experiment. In a simulated acrobot, our approach was compared against two other approaches in learning control, and achieved the best results in terms of learning speed. In a simulated aerial robot, our approach outperformed a carefully tuned PD controller, and yielded an accurate trajectory following. In the experiment, we applied our approach in a slide parking task of a brakeless four-wheel-drive (4x4) RC car, which possess an under-actuated, highly coupled and nonlinear dynamics. By making use of a single demonstration and a brief understanding of driving (i.e., increase throttle to speed up, and steer left/right to turn left/right), our approach can successfully park into the designated lot with an accuracy of $5\text{cm} \pm 2\text{cm}$. To our best knowledge, it is one of the state of the arts in robotics maneuver.

The paper is organized as follows: Section II describes the related work. Section III establishes our algorithm, including the coarse model, and the design of cost function. Section IV gives our theoretical results. Section V discusses the simulations and experiment. Section VI closes with a conclusion.

This work was partially sponsored by the Hong Kong RGC under the grants 414707, 415110 and 415011. T.K. Lau is with the Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong. E-mail: `tklau@mae.cuhk.edu.hk`.

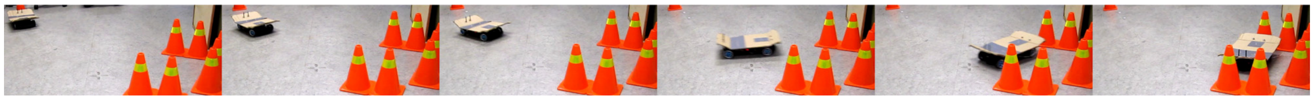


Fig. 1. **(From left to right)** These are snapshots of a four-wheel-drive (4x4) slide parking by PRISCA. The RC car started at a position shown on the top-left corner in the image, and parked into a desired parking lot shown on the right in the image by sliding. The control inputs are generated by a desktop PC running with a quad-core 3GHz processor in real-time using the live video feed from an infrared-sensitive CCD camera mounted on the ceiling. The control inputs are transmitted to the RC car by a typical RC transmitter. The videos of the parking experiments are available at: `http://www.mae.cuhk.edu.hk/~tklau/rccar`

II. Related Work

When the optimal baseline in (15) in our algorithm is set to zero, and the cost intensifier in (10) is set to one, the discounting factor is fixed to some values between zero and one, and the tracking error is driven by time instead of the proximity of state deviation, then our algorithm is similar to PGSD [1]. In fact, PGSD can be considered as a special case of our algorithm. As shown in Theorem 4.3, that kind of formulation in the cost function never outperforms our algorithm in terms of learning speed. Moreover, although PGSD and our algorithm both use a *coarse* model for control, our model is approximated by making an efficient use of the demonstration in a formulation of regression.

The literature in reinforcement learning for control is too broad to survey. Here are a portion of work that related to online learning for control: [2], [3] use immediate reward reinforcement learning, and learn a partitioned local and forward model around some states for control. It is formulated as an online learning algorithm. Yet it is reported to take about 3 machine hours to learn a policy for a 3DOF (degree-of-freedom) robot. [4], [5] propose a learning method that can be formulated in a stochastic policy or a deterministic policy. For the stochastic policy, it needs to obtain a model-free policy gradient. But such gradient is constructed on the distribution of $log\pi_{\boldsymbol{\theta}}(a_k|s_k)$ among a number of samples, hence is not suitable for online learning with scarce samples or trials. [6] shows how an optimal baseline can be added to gradients. Nevertheless, our baseline is of a local one and is not based on the true expectation which would require numerous roll-outs. [7] gave a stability-checking method for online algorithms in learn-to-control domain. Rather than proposing an algorithm to learn a policy online, the authors emphasized the monitoring of the safety guarantees in this class of online learning method. [8], [9] attempted to repeat an expert's demonstration in order to build a model to control. [10] proposed to learn an optimal trajectory and a local model for control. [11] focused on retrieving the hard to be described reward function in the MDP formalism, and learn an optimal policy by value/policy iteration.

For *extreme* maneuvers[1] like the slide parking, [12] successfully performed such a maneuver by a multiple-model LQR approach. That approach first builds a driving model by 2 minutes of data for a "normal" driving, and mixes the control with recorded actions when the residual between actual and predicted states goes up. In our algorithm, no training is needed to build an accurate dynamics model for the mixing of control inputs. Moreover, their slide parking is performed by using throttle, steering and brakes. The availability of brakes on their platform facilitates a more precise control over velocity and position during the maneuver, but our platform is a brakeless four-wheel-drive car which only takes throttle and steering as inputs.

[1]In this paper, we describes the control tasks that can yet be solved by the optimal and robust control as the *extreme* maneuvers.

III. Algorithm

A. Policy Search via Gradient

A vector $\mathbf{s}(k) \in \mathbb{R}^n$ describes the states of an agent at a discretized time k, i.e. $k\Delta t = t$, where Δt is the sampling time, and $J_\tau(\mathbf{s})$ is a positive cost function $J : \mathbb{R}^n \mapsto \mathbb{R}_+$ where τ denotes a H-step time horizon, i.e. the scope of J covers the most immediate H items. Suppose a control input $\mathbf{u} \in \mathbb{R}^m$ is parameterized by a policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{p\times m}$ and a state feature $\boldsymbol{\phi}(\mathbf{s}_k) \in \mathbb{R}^p$, we write $\mathbf{u}_k = \boldsymbol{\theta}^T\boldsymbol{\phi}(\mathbf{s}_k) + \mathbf{u}_k^*$, where $\mathbf{u}_k^*$ is the action from a desired demonstration, if available. The optimal policy is parameterized by a set of ${}^{(\ell)}\theta^*$, for $\ell = 1, 2, \ldots, L$, such that,

$$ {}^{(\ell)}\theta^* = \underset{{}^{(\ell)}\theta}{\operatorname{argmin}}{}^{(\ell)}J_\tau(\mathbf{s}) \tag{1}$$

By the gradient descent, we define an update rule that iteratively updates the policy parameters in the direction that minimizes the cost function,

$$ {}^{(\ell)}\theta_{k+1} \leftarrow {}^{(\ell)}\theta_k - \alpha_k {}^{(\ell)}g_k, \tag{2}$$

where ${}^{(\ell)}g$ is the partial derivative of the cost function with respect to the ℓth policy parameter, such that ${}^{(\ell)}g_k = \partial^{(\ell)}J_\tau/\partial^{(\ell)}\theta_k$, and α_k is a discounting factor which is square summable but not summable, such that,

$$ \alpha_k = \beta/(k+\beta) \text{ for } \beta \in \mathbb{R}_+ \tag{3}$$

The summary of our algorithm is given in Algorithm 1.

B. Coarse Model

Putting ${}^{(\ell)}\theta$ for $\ell = 1, 2, \ldots, L$ into a matrix $\boldsymbol{\theta}$ where $\boldsymbol{\theta} \in \mathbb{R}^{p\times m=L}$, the gradient term in Equation (2) involves $\partial\mathbf{s}_{k+1}/\partial\boldsymbol{\theta}_k$. Assuming that the change of states depends on the change of control inputs hence the change of control parameters, we write,

$$ \frac{\partial\mathbf{s}_{k+1}}{\partial\boldsymbol{\theta}_k} = \frac{\partial\mathbf{s}_{k+1}}{\partial\mathbf{u}_k}\cdot\frac{\partial\mathbf{u}_k}{\partial\boldsymbol{\theta}_k} = \boldsymbol{\phi}(\mathbf{s}_{k+1})\left(\frac{\partial\mathbf{s}_{k+1}}{\partial\mathbf{u}_k}\right)^T \tag{4}$$

This step narrows the gradient down, and diverts us to solve for $\partial\mathbf{s}_{k+1}/\partial\mathbf{u}_k$. Assume that the dynamics can be written in

Algorithm 1 PRISCA for trajectory following

```
1: Input: ${\mathbf{S}^*, \mathbf{U}^*}$, a single demonstration
2:         $\mathbf{B}_v, \tilde{\mathbf{B}}$, estimated and intuitive action action mappings
3:         $\boldsymbol{\phi}$, state feature
4:         $\gamma$, rate of inverse depreciation
5:         $H$, size of time horizon
6:         $\beta$, rate of learning
7: Learning Dynamics:
8: $\mathbf{B}_v^* = \underset{\mathbf{B}_v}{\text{argmin}}\ \mathbf{E}_{i=0:T}\mathbf{E}_{j=i:T}||\delta\mathbf{s}_{k+1} - \mathbf{B}_v \odot \tilde{\mathbf{B}}\Delta t\ \delta\mathbf{u}_k||_2^2$
9: Learning Control:
10: for $k = 0$ to $K$ do
11:     Compute baseline,
12:     ${}^{(\ell)}b = \frac{2\mathbf{E}_{\tau'}[(\mathbf{E}_i\mathbf{E}_j\mathfrak{A}D)(\mathbf{E}_i\mathbf{E}_j\mathfrak{A})]}{\mathbf{E}_{\tau'}[(\mathbf{E}_i\mathbf{E}_j\mathfrak{A})^2]+[\mathbf{E}_{\tau'}(\mathbf{E}_i\mathbf{E}_j\mathfrak{A})]^2}$, for $\ell = 1, 2, \ldots, L$,
        where $\mathfrak{A} = {}^{(\ell)}\phi(\mathbf{s})(\mathbf{s} - \bar{\mathbf{s}}^*)^T({}^{(\ell)}\mathbf{B}_v^*)^T$, for $\ell = 1, 2, \ldots, L$
13:     Compute gradient,
14:     ${}^{(\ell)}g = \mathbf{E}_{i=k-H:k}\ \mathbf{E}_{j=i:k}{}^{(\ell)}\phi(\mathbf{s}_j)(\mathbf{s}_j - \bar{\mathbf{s}}_j^*)^T({}^{(\ell)}\mathbf{B}_v^*)^T(D_j - {}^{(\ell)}b)$, for $\ell = 1, 2, \ldots, L$
15:     Update parameter,
16:     ${}^{(\ell)}\theta := {}^{(\ell)}\theta - \alpha_k{}^{(\ell)}g$, for $\ell = 1, 2, \ldots, L$
17:     Compute control input,
18:     $\mathbf{u} := \boldsymbol{\theta}^T\boldsymbol{\phi}(\mathbf{s}_k) + \mathbf{u}_k^*$, ($\mathbf{u}_k^* = 0$ if unavailable)
19: end for
```

a system of differential equations, and the states are update by the time-step integration, then we have,

$$\frac{\partial \mathbf{s}_{k+1}}{\partial \mathbf{u}_k} = \frac{\partial}{\partial \mathbf{u}_k}\sum_{i=0}^{N-1}(\mathbf{A}\Delta t + \mathbf{I})^i(\mathbf{B}\Delta t)\mathbf{u}_{k-i} = \mathbf{B}\Delta t \quad (5)$$

where $\mathbf{A}, \mathbf{B}$ are the state transition mapping and action mapping, Δt is the sampling time.

Assuming that we have a rough knowledge of the actuation and dynamics, we decompose the action mapping $\mathbf{B}$ into two terms,

$$\mathbf{B} \cong \mathbf{B}_v \odot \tilde{\mathbf{B}}, \quad (6)$$

where $\odot$ is defined as an element-wise multiplication between two matrices of the same size[2]. $\mathbf{B}_v$ is a to-be-estimated action mapping. $\tilde{\mathbf{B}}$ is the intuitive action mapping.

For car driving, we intuitively know that if we increase the throttle, the car will speed up; and if we steer left, it will move left and rotate to the left. Therefore, we can have such a mapping,

	Lateral Movement	**Longitudinal Movement**	**Orientation Movement**
Throttle	1	0	0
Steering	0	1	1

Hence,

$$\mathbf{B}_v \odot \tilde{\mathbf{B}} = \begin{pmatrix} b_1 & 0 & 0 \\ 0 & b_2 & b_3 \end{pmatrix} \odot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}, \quad (7)$$

where $b_{1,2,3}$ are the terms to be estimated. For different dynamical systems, the arrangement of the 1's and 0's in $\tilde{\mathbf{B}}$ will vary, and is depended on the choices of states and control inputs.

By making use of a series of state-and-action pairs from a single demonstration, we can approximate the model by regression, such that,

$$\mathbf{B}_v^* = \underset{\mathbf{B}_v}{\text{argmin}}\ \mathbf{E}_{i=0:T}\mathbf{E}_{j=i:T}||\delta\mathbf{s}_{k+1} - \mathbf{B}_v \odot \tilde{\mathbf{B}}\Delta t\ \delta\mathbf{u}_k||_2^2 \quad (8)$$

where $\mathbf{E}(\cdot)$ is the expectation. $\delta\mathbf{s}_{k+1}$ is the change of states, i.e. $\delta\mathbf{s}_{k+1} = \mathbf{s}_{k+1} - \mathbf{s}_k$, and $\delta\mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k-1}$. Here, we assumed that the system is responsive. To account for latencies, an extra term can be added to (8). Also, the states should be carefully selected to reflect the deviations in the body frame. The advantage of this method is that, we make use of the intuitive, yet often ignored knowledge of a dynamics system, then efficiently guide the dynamics modeling in a setting of regression using few samples.

C. Cost Function

In this paper, we consider an algorithm that, given a desired demonstration $\mathbf{S}^* : \{\mathbf{s}^*(0), \mathbf{s}^*(\Delta t)\ldots\mathbf{s}^*(K\cdot\Delta t)\}$, to find a policy for an agent online so that the agent can automatically reproduce the desired demonstration. Straightforwardly, we define the cost function $J_\tau = J_{(k-H):k}$, where $k\cdot\Delta t = t$, as a quadratic tracking error,

$$ {}^{(\ell)}J_\tau = \mathbf{E}_{i=k-H:k}\ \mathbf{E}_{j=i:k}C_j \quad (9)$$

where C_j is the quadratic tracking error, i.e. $C_j = (\mathbf{s}_j - \bar{\mathbf{s}}_j)^T(\mathbf{s}_j - \bar{\mathbf{s}}_j)$, and $\bar{\mathbf{s}}_j$ is the desired state. $\mathbf{E}_{i=k-H:k}\ \mathbf{E}_{j=i:k}(\cdot)$ is a smoothing function which covers the most immediate H items. Apart from the penalty from the state deviations, we introduce an inverse depreciation,

$$D_i = H/(\gamma^i + 1), \text{ for } \gamma \in (0, 1) \quad (10)$$

which is a concave function to inversely depreciate the state deviation. This function intensifies the rate of learning in the progress of time. Therefore, the sooner the tracking error is minimized, the lower the cost that J_τ will incurred.

Moreover, although the method of gradient descent theoretically converges, it is susceptible to noises in practice. To learn a policy only from one demonstration, we do not have enough samples to construct a value function that tells us a probabilistically sound direction to update the policy parameters. Therefore, we introduce a variance reducer, which is often known as *baseline*, in the cost function, hence the gradient can update the policy towards convergence amidst noises. And unlike the baselines commonly used in the gradient methods, ours is of a *local* one, i.e. it is generated from the most immediate samples covered in a horizon, but not the true expectation over numerous rollouts. Furthermore, we augment the quadratic cost function with a least-state-deviation term to drive the tracking error by the proximity of the nearest desired state but not the time, such that,

$$C_j = (\mathbf{s}_j - \bar{\mathbf{s}}_j^*)^T(\mathbf{s}_j - \bar{\mathbf{s}}_j^*), \quad (11)$$

where,

$$\mathbf{s}_j^* = \underset{\mathbf{s}^*}{\text{argmin}}\mathbf{E}_\mathbf{s}|s_j - s^*| \quad (12)$$

[2]Same spirit of the $.*$ operator in MATLAB.

We re-write the cost function in (9),

$$^{(\ell)}J_\tau = \mathbf{E}_{i=k-H:k}\ \mathbf{E}_{j=i:k} C_j(D_j - {}^{(\ell)}b) \tag{13}$$

Hence,

$$^{(\ell)}g = \mathbf{E}_{i=k-H:k}\ \mathbf{E}_{j=i:k}{}^{(\ell)}\phi(\mathbf{s}_j)(\mathbf{s}_j - \bar{\mathbf{s}}^*_j)^T({}^{(\ell)}\frac{\partial \mathbf{s}}{\partial \mathbf{u}})^T(D_j - {}^{(\ell)}b) \tag{14}$$

where $^{(\ell)}(\cdot)$ denote the terms that are related to the ℓ^{th} control parameter, i.e. $^{(\ell)}\theta$, for $\ell = 1, 2, \ldots, L$.

The purpose of b is to suppress and eventually reduce the effect of variance to the gradient during the learning of a policy, hence the optimal variance reducer of a H-step horizon is formulated as,

$$\begin{aligned} ^{(\ell)}b^* &= \underset{^{(\ell)}b}{\operatorname{argmin}}\, Var({}^{(\ell)}g) \\ &= \underset{^{(\ell)}b}{\operatorname{argmin}} \mathbf{E}_\tau[({}^{(\ell)}g - \mathbf{E}_\tau[{}^{(\ell)}g])^2] \end{aligned}$$

By setting,

$$\partial\, Var({}^{(\ell)}g)/\partial^{(\ell)}b = 0$$

then,

$$^{(\ell)}b = \frac{2\mathbf{E}_{\tau'}[(\mathbf{E}_i\mathbf{E}_j\mathfrak{A}D)(\mathbf{E}_i\mathbf{E}_j\mathfrak{A})]}{\mathbf{E}_{\tau'}[(\mathbf{E}_i\mathbf{E}_j\mathfrak{A})^2] + [\mathbf{E}_{\tau'}(\mathbf{E}_i\mathbf{E}_j\mathfrak{A})]^2} \tag{15}$$

where $\mathfrak{A} = {}^{(\ell)}\phi(\mathbf{s})(\mathbf{s} - \bar{\mathbf{s}}^*)^T({}^{(\ell)}\frac{\partial \mathbf{s}}{\partial \mathbf{u}})^T$. $\mathbf{E}_{\tau'}(\cdot)$ is an expectation over a time horizon τ'. $\mathbf{E}_i(\cdot)$ is denoted as $\mathbf{E}_{i=k-H:k}(\cdot)$, and $\mathbf{E}_j(\cdot)$ is denoted as $\mathbf{E}_{j=i:k}(\cdot)$. This τ' is picked as 2τ in experiments. Unlike the baselines that are commonly used in the gradient method, our baseline is of a *local* one, i.e. it does not depend on a true expectation of the states but on a range of states covered by τ'.

IV. Theoretical Results

The formulation of the cost function in the previous section is predicted on the assumption that the inverse depreciation and variance reducer together yield a better policy search than the typical policy gradient method (PGM) during the learning. In this section, we prove that our formulations of the cost function and update rule always yield a policy that gives better results when comparing with PGM, and in probability that our algorithm will converge to a local optimal policy.

The following lemmas establish the proofs of the performance and convergence.

Lemma 4.1: Let there be a step size $H \in [1, 2 \ldots 50]$ and a time-discounting factor $\gamma \in [0, 1]$ in a concave function that $D_i = H/(\gamma^i + 1)$, then there always exist a combination of H and γ that

$$\mathbf{E}_{\tau'}(D^2) - \mathbf{E}_{\tau'}(D)^2 - 1 < 0$$

Proof: The existence of such a combination can be best understood by visualization. We plotted the $H - \gamma$ graph in Figure 2. ■

Lemma 4.2: Let there be a Prisca gradient $\mathbf{g} \in \mathbb{R}^{p\times m}$ and a typical PGM gradient $\tilde{g} \in \mathbb{R}^{p\times m}$ that is formulated without the inverse depreciation and variance reducer, such that $\tilde{J} = C$. The variance of Prisca gradient is smaller than the variance of the typical PGM gradient. We then have that,

$$Var_{\tau'}({}^{(\ell)}g) < Var_{\tau'}({}^{(\ell)}\tilde{g})$$

Proof: Consider that,

$$Var_{\tau'}({}^{(\ell)}g) - Var_{\tau'}({}^{(\ell)}\tilde{g}) < [\mathbf{E}_{\tau'}(D^2) - \mathbf{E}_{\tau'}(D)^2 - 1]\mathbf{E}_{\tau'}({}^{(\ell)}\tilde{g}^2) \tag{16}$$

Using Lemma 4.1, there exist a combination of γ and H that would satisfy,

$$\mathbf{E}_{\tau'}(D^2) - \mathbf{E}_{\tau'}(D)^2 - 1 < 0$$

Then we immediately have,

$$Var({}^{(\ell)}g) - Var({}^{(\ell)}\tilde{g}) < 0$$

■

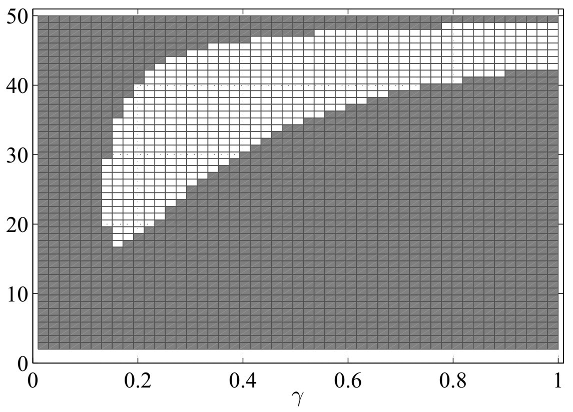


Fig. 2. The gray grids represent the combinations of the corresponding H and γ satisfies $\mathbf{E}_{\tau'}(D^2) - \mathbf{E}_{\tau'}(D)^2 - 1 < 0$.

Theorem 4.3: Let there be a matrix $\boldsymbol{\theta} \in \mathbb{R}^{p\times m}$ which consists of a set of $^{(\ell)}\theta$ for $i = 1, 2, \ldots, L$, and it iterates by setting $^{(\ell)}\theta_{k+1} \leftarrow {}^{(\ell)}\theta_k - \alpha_k{}^{(\ell)}g_k$ where a gradient, $^{(\ell)}g_k$, is the partial derivative of a cost function, such that $^{(\ell)}g_k = \nabla_{^{(\ell)}\theta}{}^{(\ell)}J_\tau$. Assume that up to an arbitrary k time-step, Prisca and the typical PGM yield the same output, then, Prisca would always yield a $^{(\ell)}\theta_{k+1}|_g$ that is closer to a local optimum than a typical PGM gradient does, i.e. for $\ell = 1, 2, \ldots, L$,

$$\mathbf{E}[({}^{(\ell)}\theta_{k+1}|_g - {}^{(\ell)}\theta^*)^2; {}^{(\ell)}\theta_k] < \mathbf{E}[({}^{(\ell)}\theta_{k+1}|_{\tilde{g}} - {}^{(\ell)}\theta^*)^2; {}^{(\ell)}\theta_k] \tag{17}$$

Proof: Using the typical PGM, the ℓ^2-norm of the squared difference between an instance of the policy parameters $^{(\ell)}\theta_{k+1}$ and the optimum is $({}^{(\ell)}\theta_{k+1} - {}^{(\ell)}\theta^*)^2$. We denote the parameter obtained by our method as $^{(\ell)}\theta_{k+1}|_g$, and the one by the typical PGM is $^{(\ell)}\theta_{k+1}|_{\tilde{g}}$.

Assume that the gradient in our method is $^{(\ell)}g$ and it is composed of a true gradient $^{(\ell)}\bar{g}$ and variance v_g, i.e. $^{(\ell)}g = {}^{(\ell)}\bar{g} + v_g$. Similarly, the gradient in typical PGM is denoted as $^{(\ell)}\tilde{g}$, i.e., $^{(\ell)}\tilde{g} = {}^{(\ell)}\bar{g} + v_{\tilde{g}}$. As variances are non-negative, and by Lemma 4.2, we have $v_g^2 < v_{\tilde{g}}^2$. Also, we assume that a true gradient $^{(\ell)}\bar{g}$ would lead to a local optimum $^{(\ell)}\bar{\theta}_{k+1}$, i.e. $^{(\ell)}\theta^* = {}^{(\ell)}\bar{\theta}_{k+1} = {}^{(\ell)}\theta_k - \alpha^{(\ell)}\bar{g}$.

Taking an expectation to observe the bound of the squared difference between the iterated parameter and the local optimum, when given the same previous parameter ${}^{(\ell)}\theta_k$,

$$\mathbf{E}[({}^{(\ell)}\theta_{k+1}|_g - {}^{(\ell)}\theta^*)^2; {}^{(\ell)}\theta_k] <$$
$$\mathbf{E}(-2\alpha v_g {}^{(\ell)}\theta_{k+1} + \alpha^2 v_g^2 - 2{}^{(\ell)}\theta^* {}^{(\ell)}\bar{\theta}_{k+1} + 2\alpha v_g {}^{(\ell)}\theta^*; {}^{(\ell)}\theta_k) \quad (18)$$

Similarly, we can write a similar form for $\mathbf{E}[({}^{(\ell)}\theta_{k+1}|_{\tilde{g}} - {}^{(\ell)}\theta^*)^2; {}^{(\ell)}\theta_k]$. Then, by subtracting these two expectations together, we have,

$$\mathbf{E}[({}^{(\ell)}\theta_{k+1}|_g - {}^{(\ell)}\theta^*)^2; {}^{(\ell)}\theta_k] - \mathbf{E}[({}^{(\ell)}\theta_{k+1}|_{\tilde{g}} - {}^{(\ell)}\theta^*)^2; {}^{(\ell)}\theta_k]$$
$$< \mathbf{E}[\alpha^2(v_g^2 - v_{\tilde{g}}^2); {}^{(\ell)}\theta_k] < 0, \text{ for } \ell = 1, 2, \ldots, L \quad (19)$$

Remark Theorem 4.3 shows a that PRISCA always yields a better control parameter in each step in terms of the closeness to the local optimum. In other words, it always learns faster than typical PGM.

■

Theorem 4.4: Let there be a policy parameter which is updated according to PRISCA. Then, this policy parameter will converge to a local optimum in probability, such that,

$$Pr(\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|_2^2 \geq \epsilon) \leq 0, \text{ where } \epsilon > 0$$

Proof: Recall that the time-discounting factor in (3) is chosen as $\alpha_k = \beta/(k+\beta)$, which is square summable but not summable, the proof of convergence of PRISCA is equivalent to the proof of convergence of subgradient. See [13] for the details. ■

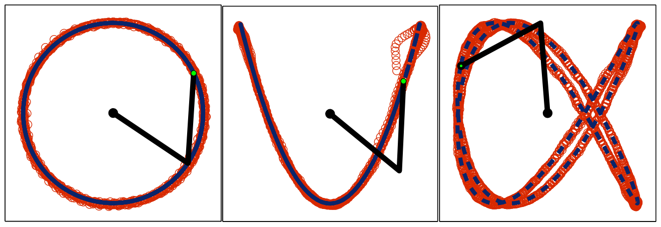
V. EXPERIMENTS

A. Acrobot

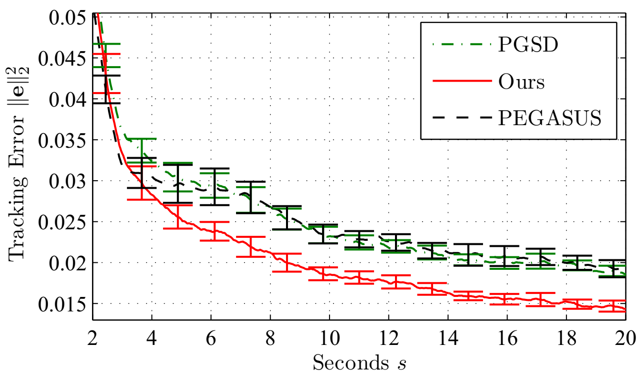
Our first simulation is the trajectory following of an acrobot. The mass of each link is assumed to be a point mass centered at the middle of each link. Each joint can exert a limited torque. The tracking error is the deviation of the desired joint angles. Given a desired trajectory, we generate the series of desired joint angles at each time-step. The state feature is defined as the deviation of tracking angle and the rate of this deviation.

Two other methods are included for comparison. PGSD [1] and PEGASUS [14]. PGSD uses a coarse model and policy gradient to refine the policy online. See Section II for more. PEGASUS searches for a policy by the finite difference method.

All methods are carefully hand-tuned to yield their best performance. Figure 3 shows the result of these algorithms.



(a) Trajectory following of an acrobot



(b) Comparison of algorithms

Fig. 3. (a) Trajectory following of an acrobot: Given a desired trajectory for an acrobot, we evaluated two other reinforcement learning algorithms along with our approach in a MATLAB simulation. (b) Performance evaluation: After a careful calibrations of the step size on each algorithm, this figure shows the best results of each algorithm. The simulations ran for 20 times. The 95% confidence interval is plotted. The tracking error is the square of ℓ^2-norm of the joint angle deviation. This results show that PRISCA learnt faster and in a more focused manner than PGSD and PEGASUS.

B. Aerial Robot

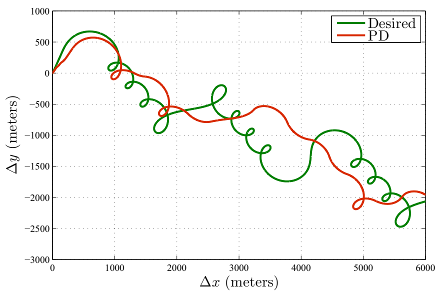
In our second simulation, we evaluated our algorithm on a fixed-wing aerial robot in a trajectory following task. We ran our algorithm on an open source flight simulation called FlightGear with a flight dynamic model from JSBSim. We first defined a desired trajectory, then activated our algorithm in the middle of flight to follow this trajectory. The desired trajectory is localized to the state of the robot at which the algorithm is activated. The state feature is defined as the longitudinal position deviation with respect to the desired body frame, and the heading deviation. The definition of coordinate frame is described in Figure 4. A PD controller is implemented as a comparison. To show the enormous improvement that this learning algorithm can achieve, we inherited all PD channels except the rudder and throttle in our evaluation. It suggests that our learning algorithm can improve a PD controller drastically.

C. Slide Parking of a Four-Wheel-Drive RC Car

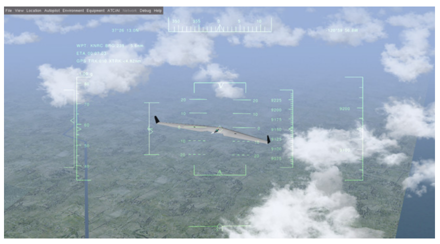
Unlike full-sized cars that can make use of handbrake and brake to perform slide parking, the maneuverability of the RC car in this experiment is much more demanding because this car takes only two inputs: throttle and steering, for control. To slide this RC car into a desired parking lot, this RC car must be accelerated to a speed and steer properly at the right time with a right magnitude and rate to achieve a precise parking. While the dynamics of the traction and actuation in the regime of sliding is hard to be described, the typical optimal control, such as LQR, cannot be trivially applied to this robotics maneuver.

For the evaluation, we as well compared our algorithm with the variants of PD controllers, and the results are shown in Table I. We denote the PD controller with the H-step horizon (13) as *PD+H*, and with the least-state-deviation (12) as *PD+LSD*. *PD+H+LSD* is denoted as having both techniques.

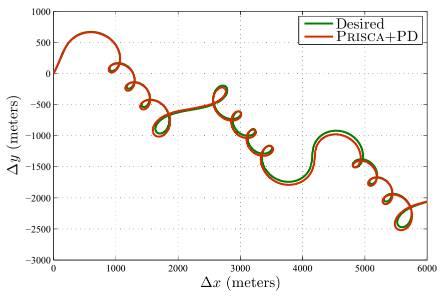
We found that the techniques of least-state-deviation and



(a)



(b)



(c)

Fig. 4. **(b)**. Using an open-source flight simulation called FlightGear (`http://www.flightgear.org/`) and a popular flight dynamic model by JSBSim (`http://www.jsbsim.org/`), we ran our learning algorithm on a desktop PC to control a fixed-wing UAV for a trajectory following task in real-time. **(a)** PD controller: *Green* line is the desired trajectory. *Red* line is the actual trajectory achieved by the controller. It is the best results after a careful calibration of the gains. Best view in colors. **(c)** PRISCA: Given the same desired trajectory, our algorithm performed an immediate online learning by using a coarse model (i.e. rudder left to turn "level-left", give a larger throttle to speed up) and the tracking error. Only the rudder and throttle are controlled by the learning algorithm, the rest of the channels are inherited from the same PD controller as shown in (a). The purpose of this mixture of control is to show that even with only two learn-to-control channels, the performance of a typical PD controller can be improved enormously. The learning algorithm worked immediately without much tuning on the step size H and the time-discounting factor γ. In this evaluation, we set $H = 10, \alpha = 1$. The control frequency and the measurement sampling rate are at 40Hz.

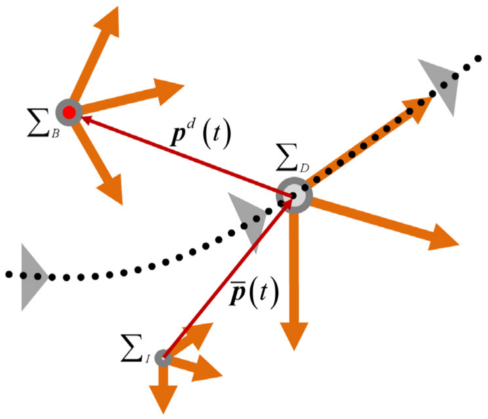


Fig. 5. $\bar{\mathbf{p}}(t)$ indicates the desired position of the robot at time t with respect to the inertial frame, $\mathbf{p}^d(t)$ is the position of the robot with respect to the desired position frame at time t. The desired position frame moves along the desired trajectory according to the time t, and the orientation of the frame is the desired orientation for the robot.

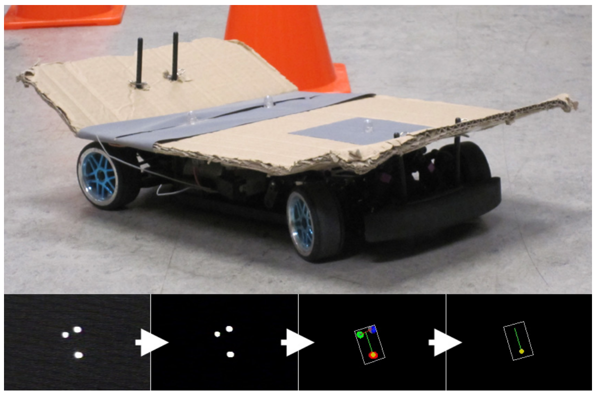


Fig. 6. The vehicle for experiments is a four-wheel-powered electric RC car. It takes only two control inputs, namely throttle δ_{thro} and steering δ_{steer}. Three high-power IrLEDs are placed on top of the car for localization. An infrared-sensitive CCD camera is mounted on ceiling to track the position and orientation of the RC car. At the bottom of this image, a sequence of snap-shots taken from each stage of the image processing is shown.

H-step horizon together improve the results of PD controller. But our algorithm yields the best results among all other methods, and is the only controller that can persistently park into the desired lot.

VI. CONCLUSION

We presented a policy search method, to learn a parameterized control policy from only a very coarse model information (e.g., for driving, steer left to turn left) and a desired demonstration for trajectory following. We showed that our algorithm holds the guarantees in performance and convergence, and demonstrated its performance and effectiveness not only in simulations of an acrobot and an aerial robot, but also in a slide parking experiment of a 4x4 RC car. Our algorithm is generally applicable to robotic tasks that can be formulated in trajectory following. We believe that our online learning algorithm – which requires neither

TABLE I

THE PERFORMANCE OF PRISCA IN 4-WHEEL SLIDE PARKING

	POSITION	
	MEAN-ABSOLUTE-ERROR	ERROR VARIANCE
OURS	$4.89cm \pm 2.17cm$	$4.73cm$
PD	$32.12cm \pm 28.19cm$	$794.59cm$
PD (SELECTED)	$22.35cm \pm 15.21cm$	$231.42cm$
PD+H	$13.44cm \pm 7.35cm$	$54.04cm$
PD+LSD	$22.63cm \pm 13.68cm$	$187.03cm$
PD+H+LSD	$12.96cm \pm 8.33cm$	$69.35cm$
ACTION PLAYBACK	$12.41cm \pm 5.34cm$	$28.55cm$
	ORIENTATION	
	MEAN-ABSOLUTE-ERROR	ERROR VARIANCE
OURS	$17.35^o \pm 7.49^o$	56.14^o
PD	$50.29^o \pm 68.61^o$	4707.59^o
PD (SELECTED)	$30.40^o \pm 15.50^o$	240.35^o
PD+H	$21.14^o \pm 9.52^o$	90.72^o
PD+LSD	$27.66^o \pm 10.42^o$	108.59^o
PD+H+LSD	$22.71^o \pm 10.83^o$	117.22^o
ACTION PLAYBACK	$11.61^o \pm 13.28^o$	176.28^o

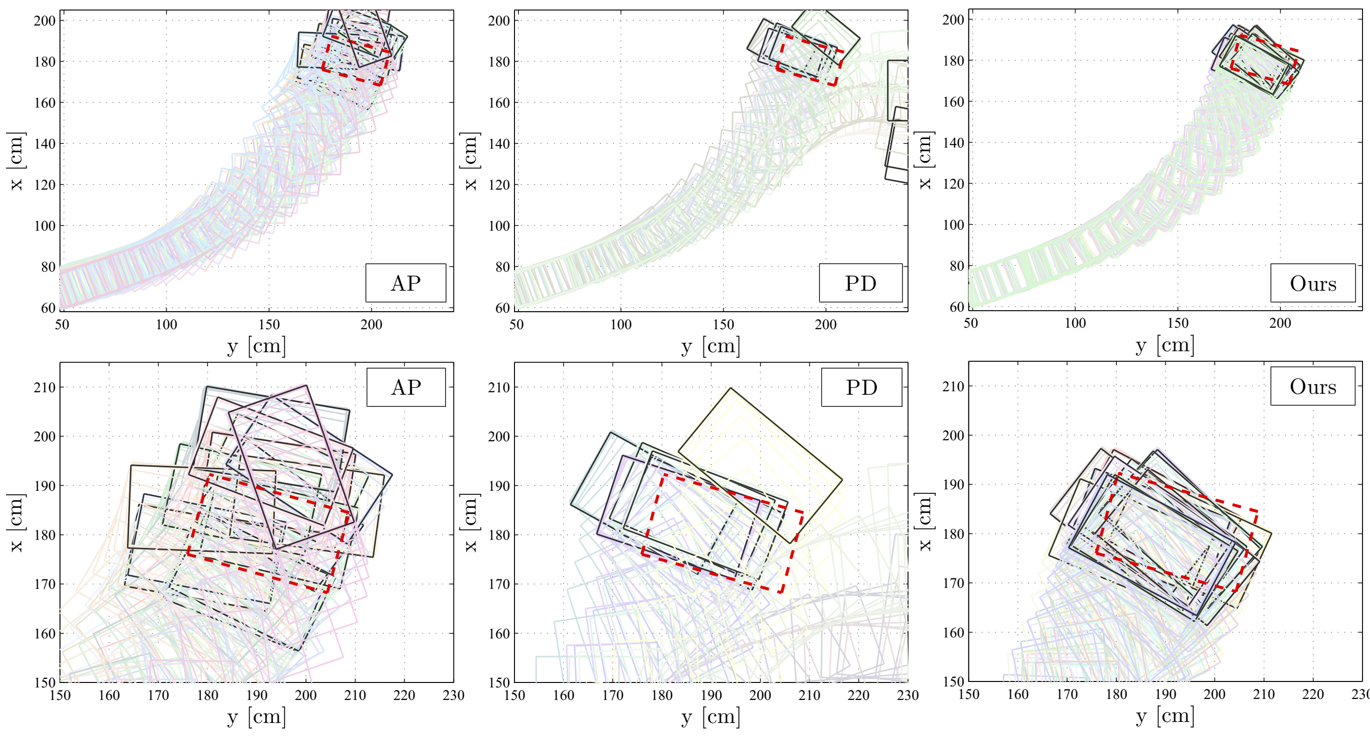


Fig. 7. **(Upper)** The whole slide parking trajectories performed by actions playback (AP), proportional-derivative (PD) control, and PRISCA. **(Lower)** A closer look around the region of the desired parking lot. The desired parking lot is in dotted red lines. The end postures of each parking attempt are in dark colored lines. The intermediate trajectories are in pale colored lines. **Left:** By actions playback (AP). **Middle:** By proportional-derivative (PD) control plus AP. **Right:** By PRISCA. As a whole, by AP or PD , the car could only park into the desired parking lot occasionally by luck. By PRISCA, the RC car can consistently park by making use of only one demonstration, and the intuitive knowledge of car driving. The axes are in centimeters $[cm]$. The performance is detailed in Table I. Best view in colors.

a dynamical model nor multiple demonstrations – holds promise for solutions to robotics maneuver in some extreme situations.

ACKNOWLEDGMENT

The author gives warm thanks to Yun-Hui Liu for his helpful discussion and support, and to J. Zico Kolter for his implementation of the dynamical simulation of acrobot in MATLAB, and to Kit-Hung Lee and Shun-Yi Lau for their help in the setup of experiments.

REFERENCES

[1] J. Kolter and A. Ng, "Policy search via the signed derivative," in *Proceedings of Robotics: Science and Systems*, 2009.
[2] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 745–750.
[3] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Advances in Neural Information Processing Systems*, vol. 22, 2009.
[4] P. Glynn, "Likelihood ratio gradient estimation for stochastic systems," *Communications of the ACM*, vol. 33, no. 10, 1990.
[5] J. Tang and P. Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," in *Advances in Neural Information Processing Systems*, 2010.
[6] E. Greensmith, P. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *The Journal of Machine Learning Research*, vol. 5, 2004.
[7] A. Ng and H. Kim, "Stable adaptive control with online learning," in *Advances in Neural Information Processing Systems*, 2004.
[8] C. Atkeson and S. Schaal, "Learning tasks from a single demonstration," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, 2002.
[9] ——, "Robot learning from demonstration," in *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997, pp. 12–20.
[10] A. Coates, P. Abbeel, and A. Ng, "Learning for control from multiple demonstrations," in *Proceedings of 25th International Conference on Machine Learning*, 2008, pp. 144–151.
[11] P. Abbeel and A. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the 21st International Conference on Machine Learning*, 2004, p. 1.
[12] J. Kolter, C. Plagemann, D. Jackson, A. Ng, and S. Thrun, "A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving," in *Robotics and Automation, 2010 IEEE International Conference on*, 2010, pp. 839–845.
[13] N. Shor, *Nondifferentiable optimization and polynomial problems*. Kluwer Academic Publishers, 1998.
[14] A. Ng and M. Jordan, "PEGASUS: A policy search method for large MDPs and POMDPs," in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, 2000.