PLATO: Policy Learning using Adaptive Trajectory Optimization

Gregory Kahn[1], Tianhao Zhang[1], Sergey Levine[1], Pieter Abbeel[1,2,3]

Abstract—Policy search can in principle acquire complex strategies for control of robots and other autonomous systems. When the policy is trained to process raw sensory inputs, such as images and depth maps, it can also acquire a strategy that combines perception and control. However, effectively processing such complex inputs requires an expressive policy class, such as a large neural network. These high-dimensional policies are difficult to train, especially when learning to control safety-critical systems. We propose PLATO, a continuous, reset-free reinforcement learning algorithm that trains complex control policies with supervised learning, using model-predictive control (MPC) to generate the supervision, hence never in need of running a partially trained and potentially unsafe policy. PLATO uses an adaptive training method to modify the behavior of MPC to gradually match the learned policy in order to generate training samples at states that are likely to be visited by the learned policy. PLATO also maintains the MPC cost as an objective to avoid highly undesirable actions that would result from strictly following the learned policy before it has been fully trained. We prove that this type of adaptive MPC expert produces supervision that leads to good long-horizon performance of the resulting policy. We also empirically demonstrate that MPC can still avoid dangerous on-policy actions in unexpected situations during training. Our empirical results on a set of challenging simulated aerial vehicle tasks demonstrate that, compared to prior methods, PLATO learns faster, experiences substantially fewer catastrophic failures (crashes) during training, and often converges to a better policy.

I. INTRODUCTION

Policy search via optimization or reinforcement learning (RL) holds the promise of automating a wide range of decision making and control tasks, in domains ranging from robotic manipulation to self-driving vehicles. One particularly appealing prospect is to use policy search techniques to automatically acquire policies that subsume perception and control, thereby acquiring end-to-end perception-control systems that are adapted to the task.

However, representing policies that combine perception and control requires either a careful choice of features or the use of expressive function approximators. Recent results in perception domains, such as computer vision, natural language processing, and speech recognition, suggest that expressive function approximators, such as neural networks, can outperform hand-designed features when trained directly on raw input data while requiring substantially less manual engineering [1]. Recent years have seen considerable research on using deep networks for control [2], [3], [4], [5], [6], [7].

Unfortunately, training such large, high-dimensional policies on real physical systems is exceedingly challenging for

[1]Berkeley AI Research (BAIR), University of California, Berkeley
[2]OpenAI
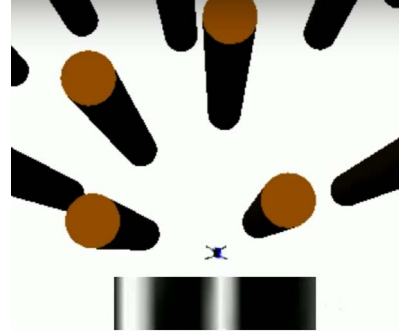[3]International Computer Science Institute (ICSI)

Fig. 1: **Policy Learning using Adaptive Trajectory Optimization**: A neural network control policy trained by PLATO navigates through a forest using camera images. During training, the adaptive MPC teacher policy chooses actions to achieve good long-horizon task performance while matching the learner policy distribution. The policy learned by PLATO converges with bounded cost.

two reasons. First, standard model-free reinforcement learning algorithms are difficult to apply to large non-linear function approximators [8]. Several recent methods demonstrate RL-based training of large neural networks [2], [9], [6], [7], but these approaches require a very large amount of experience, making them difficult to run on physical systems. In contrast, methods based on supervised learning, including DAgger [10], guided policy search [11], [5] and the work presented in this paper, are more sample-efficient, but require a viable source of supervision. The second obstacle to using RL in the real world is that, although a fully trained neural network controller can be very robust and reliable, a partially trained policy can perform unreasonable and even unsafe actions [10]. This can be a major problem when the agent is a mobile robot or autonomous vehicle and unsafe actions can cause the destruction of the robot or damage to its surroundings.

We propose PLATO (Policy Learning using Adaptive Trajectory Optimization), a reset-free method for training complex policies that combine perception and control by using a trajectory optimization teacher in the form of model-predictive control (MPC). At training time, MPC chooses actions that make a tradeoff between succeeding at the task and matching the behavior of the current policy. By gradually adapting to the policy, MPC ensures that the states visited during training will allow the policy to learn good long-horizon performance. MPC makes use of full state information, which could be obtained, for example, by instrumenting the environment at training time. The final policy, however, is trained to mimic the MPC actions using only the observations available to the robot, which makes it possible to run the resulting policy at test time without any instrumentation. The algorithm requires access to at least a rough model of the system dynamics in order to run MPC

during training, but does not require any knowledge of the observation model, making it feasible to use with complex, raw observation signals, such as images and depth scans. Since MPC is used to select all actions at training time, the algorithm never requires running a partially trained and potentially unsafe policy.

We prove that the policy learned by PLATO converges to a policy with bounded cost. Our empirical results further demonstrate that PLATO can learn complex policies for simulated quadrotor flight with laser rangefinder observations and camera observations in cluttered environments and at high speeds. We show that PLATO outperforms a number of previous approaches in terms of both the performance of the final neural network policy and the robustness to catastrophic failure during training. In comparisons with MPC-guided policy search [12], the DAgger algorithm [10], DAgger with coaching [13] and supervised learning, our approach experiences substantially fewer catastrophic failures both during training time and at test time.

II. RELATED WORK

Deep neural networks have emerged as powerful general-purpose models for processing complex sensory data. Recent years have seen an increasing amount of research on using neural networks to represent control policies for control tasks, including playing Atari games [2], robotic manipulation from camera images [5], and various other continuous control tasks [9], [6], [7]. Broadly speaking, these methods fall into two categories: methods based on reinforcement learning (RL), including Q-learning [2] and policy search [9], [6], and methods based on supervised learning, including DAgger [10] and guided policy search [11].

RL-based methods are typically more general, but require a very large amount of system experience, which limits their applicability to real physical systems [8]. Furthermore, the need to explore using partially learned policies, or worse, using random actions, causes these methods to exhibit potentially dangerous and unstable behavior during training. These limitations make it difficult to deploy RL-based algorithms directly on safety-critical systems.

Methods based on supervised learning are dramatically more sample-efficient, but require a viable source of supervision. In the case of guided policy search, this supervision comes from a simpler RL algorithm that does not directly optimize a neural network policy, but a much simpler trajectory-centric controller [14]. This approach typically requires the ability to deterministically reset the environment, which is not always feasible when learning in the real world. In the case of DAgger, supervision can come from a human expert [15] or a computational expert, such as Monte Carlo tree search [16]. However, this expert does not adapt to the learned neural network policy, and successful application of DAgger assumes that the learned policy can mimic the expert up to a small bounded error [10]. This assumption is not always realistic [17]. Furthermore, DAgger requires executing the learned policy at training time to acquire samples from its own state distribution. When learning is performed online

in non-stationary environments, this can expose the agent to dangerous situations for which the learned policy has not yet been fully trained.

In this paper, we propose PLATO, an algorithm that trains neural network control policies with supervised learning, using model-predictive control (MPC) to generate the supervision. In contrast to DAgger, PLATO adapts the computational "expert" (the MPC algorithm) to the learned policy, but does not actually execute the learned policy in the real world until training is completed. We show that this still enforces a bound on the difference between the state distribution of the learned policy and that of the MPC expert, but has the benefit of not exposing the agent to dangerous situations since MPC can still avoid dangerous on-policy actions in novel situations.

Furthermore, we demonstrate that PLATO can be used to train a policy that uses raw perceptual input, while the MPC teacher uses the true state, which allows us to train the policy without access to a model of the sensors, similarly to recent work on guided policy search [5], [12]. However, PLATO lifts a major limitation of guided policy search, which is the requirement to reset the environment between episodes – in fact, PLATO does not even assume an episodic formulation of the task; a practical training scenario might consist, for example, of a robot continuously and autonomously exploring its environment with MPC for the duration of the training period. Since resetting the environment for episodic tasks can be complex, time-consuming or even impossible in the real world, not requiring such resets is a major advantage.

III. PRELIMINARIES AND OVERVIEW

We address the problem of learning control policies for dynamical systems, such as robots and autonomous vehicles. The system is defined by states $\mathbf{x}$ and actions $\mathbf{u}$. The policy must control the system from observations $\mathbf{o}$, which are in general insufficient for determining the full state $\mathbf{x}$. The policy is a conditional distribution over actions $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$, parametrized by θ. At test time, the agent chooses actions according to $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ at each time step t, and experiences a loss $c(\mathbf{x}_t, \mathbf{u}_t)$. We assume without loss of generality that $c(\mathbf{x}_t, \mathbf{u}_t)$ is in the interval $[0, 1]$. The next state is distributed according to the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. The goal is to learn a policy $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ that minimizes the total cost $J(\pi) = E_\pi \left[\sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \right]$. We will use $J_t(\pi|\mathbf{x}_t) = E_\pi \left[\sum_{t'=t}^{T} c(\mathbf{x}_{t'}, \mathbf{u}_{t'})|\mathbf{x}_t \right]$ as shorthand for the expected cost from state $\mathbf{x}_t$ at time t, such that $J(\pi) = E_{\mathbf{x}_1 \sim p(\mathbf{x}_1)} \left[J_1(\pi|\mathbf{x}_1) \right]$.

In this work, we further assume that during training, our algorithm has access to the true underlying states $\mathbf{x}$. This additional assumption allows us to use simple and efficient model-predictive control (MPC) methods to generate training actions. We do not require knowing the true states $\mathbf{x}$ at test time, since the learned policy $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ only requires observations. This training setup could be implemented in various ways in practice, including instrumenting the training environment (e.g. using motion capture to track a mobile robot) or using more effective hardware at training time (such as a more accurate GPS system), while only having

access to cheaper and more practical hardware at test time. While this assumption does introduce some restrictions, we will show that it enables very efficient and relatively safe training, making it an appealing option for safety-critical systems.

We will train the policy $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ by mimicking a computational "teacher," rather than attempting to learn the policy directly with reinforcement learning. There are three key advantages to this approach: first, the teacher can exploit the true state $\mathbf{x}$, while the final policy π_θ is only trained on the observations $\mathbf{o}$; second, we can choose a teacher that will remain safe and stable, avoiding dangerous actions during training; third, we can train the final policy π_θ using standard, robust supervised learning algorithms, which will allow us to construct a simple and highly data-efficient algorithm that scales easily to complex, high-dimensional policy parametrization. Specifically, we will use MPC as the teacher. MPC uses the true state $\mathbf{x}$ and a model of the system dynamics (which we assume to be known in advance, but which in general could also be learned from experience). MPC plans locally optimal trajectories with respect to the dynamics, and by replanning every time step, is able to achieve considerable robustness to unexpected perturbations and model errors [18], making it an excellent choice for sample-efficient learning.

IV. Policy Learning using Adaptive Trajectory Optimization

One naïve approach to learn a policy from a computational teacher such as MPC would be to generate a training set with MPC, and then train the policy with supervised learning to maximize the log-likelihood of this dataset. The teacher can safely choose robust, near-optimal trajectories. However, this type of supervision ignores the fact that the state distribution for the teacher and that of the learner are different [10]. Formally, the distribution of states at test time will not match the distribution at training time, and we therefore cannot expect good long-horizon performance from the learned policy.

In order to overcome this challenge, PLATO uses an adaptive MPC teacher that modifies its actions in order to bring the state distribution in the training data closer to that of the learned policy, while still producing robust trajectories and reacting intelligently to unexpected perturbations that cannot be handled by a partially trained policy. To that end, the teacher generates actions at each time step t from a controller obtained by optimizing the following objective:

$$\pi_\lambda^t(\mathbf{u}|\mathbf{x}_t, \theta) \leftarrow \arg\min_\pi J_t(\pi|\mathbf{x}_t)$$
$$+ \lambda D_{\mathrm{KL}}\big(\pi(\mathbf{u}|\mathbf{x}_t)||\pi_\theta(\mathbf{u}|\mathbf{o}_t)\big), \quad (1)$$

where λ determines the relative importance of matching the learner π_θ versus optimizing the expected return $J(\cdot)$. Since the teacher uses an MPC algorithm, this objective is reoptimized at each time step to obtain a locally optimal controller for the current state. The only difference from a standard MPC algorithm is the inclusion of the KL-divergence

term. The particular MPC algorithm we use is based on iterative LQG (iLQG) [19], using a maximum entropy variant that produces linear-Gaussian stochastic controllers of the form $\pi_\lambda(\mathbf{u}|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \Sigma_t)$ [11]. The details of this maximum entropy variant of iLQG may be found in prior work [19], [20], [14]. We describe the details of PLATO and its relation to prior methods in Sec. IV-A and show that PLATO produces a good learned policy in Sec. V.

A. Algorithm Description

Algorithm 1 outlines PLATO. We collect training trajectories by choosing actions $\mathbf{u}_t$ according to an adaptive teacher policy $\pi_\lambda^t(\mathbf{u}|\mathbf{x}_t, \theta)$, which is generated by optimizing the objective in Equation 1 at each time step via iLQG. We then update the learner policy $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ with supervised learning at the observations $\mathbf{o}_t$ corresponding to the visited states $\mathbf{x}_t$ to minimize the difference between $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ and the locally optimal policy

$$\pi^*(\mathbf{u}|\mathbf{x}_t) \leftarrow \arg\min_\pi J(\pi), \quad (2)$$

which is also obtained via MPC, but without considering the KL-divergence term. This approach ensures the teacher visits states that are similar to those that would be visited by the learner policy π_θ, while still providing supervision from a near-optimal policy. Note that the MPC policy is conditioned on the state of the system $\mathbf{x}_t$, while the learned policy $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ is only conditioned on the observations. MPC requires access to at least a rough model of the system dynamics, as well as the system state, in order to robustly choose near-optimal actions. However, by training π_θ on the corresponding observations, instead of the true states, π_θ can learn to process raw sensory inputs without requiring true state observations, making it possible to run the learned policy with only the raw observations at test time. In the rest of this section, we describe the MPC teacher and the supervised learning procedure in detail.

Adaptive MPC teacher: The teacher's policy π_λ^t must take reasonable, robust actions while visiting states that are similar to those that would be seen by the learner policy π_θ. However, we do not know the state distribution of π_θ in advance, since although we have some approximate knowledge of the system

---

Algorithm 1 PLATO algorithm
___
1: Initialize data $\mathcal{D} \leftarrow \emptyset$
2: **for** $i = 1$ **to** N **do**
3:     **for** $t = 1$ **to** T **do**
4:         Optimize π_λ^t with respect to Equation (1)
5:         Sample $\mathbf{u}_t \sim \pi_\lambda^t(\mathbf{u}|\mathbf{x}_t, \theta)$
6:         Optimize π^* with respect to Equation (2)
7:         Sample $\mathbf{u}_t^* \sim \pi^*(\mathbf{u}|\mathbf{x}_t)$
8:         Append $\big(\mathbf{o}_t, \mathbf{u}_t^*\big)$ to the dataset $\mathcal{D}$
9:         State evolves $\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$
10:     **end for**
11:     Train $\pi_{\theta_{i+1}}$ on $\mathcal{D}$
12: **end for**
___

dynamics, we do not assume a model of the observation function that produces observations $\mathbf{o}_t$ from states $\mathbf{x}_t$, making it impossible to simulate the policy π_θ into the future. Instead, we choose the actions at each time step according to an MPC policy π_λ^t that minimizes the expected long-term sum of costs $J_t(\pi_\lambda^t|\mathbf{x}_t)$, but only greedily minimizes the KL-divergence against π_θ at the current time step t, where the observation $\mathbf{o}_t$ is already available, resulting in the objective in Equation 1. Since MPC reoptimizes the local policy at each time step, this method produces a sequence of policies $\pi_\lambda^{1:T}$, each of which is optimized with respect to its long-horizon cost and immediate disagreement with π_θ.

As discussed previously, our iLQG-based MPC algorithm produces linear-Gaussian local controllers $\pi_\lambda^t(\mathbf{u}|\mathbf{x}_t) = \mathcal{N}(\mu_\lambda(\mathbf{x}_t), \Sigma_t)$ where $\mu_\lambda(\mathbf{x}_t) = \mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t$. We will further assume that our learner policy is conditionally Gaussian (but nonlinear), though other parametric distributions are also possible. The policy therefore has the form $\pi_\theta(\mathbf{u}|\mathbf{o}_t) = \mathcal{N}(\mu_\theta(\mathbf{o}_t), \Sigma_{\pi_\theta})$ where $\mu_\theta(\mathbf{o}_t)$ is the output of a nonlinear function, such as a neural network, and covariance Σ_{π_θ} can be either learned or deterministic. Then the MPC objective can be expressed in closed form:

$$\min_\pi \ J_t(\pi|\mathbf{x}_t) + \frac{1}{2}\lambda\bigg[\ln\bigg(\frac{|\Sigma_{\pi_\theta}|}{|\Sigma_t|}\bigg) + \text{tr}\big(\Sigma_{\pi_\theta}^{-1}\Sigma_t\big)$$
$$+ \big(\mu_\theta(\mathbf{o}_t) - \mu_\lambda(\mathbf{x}_t)\big)^\mathsf{T}\Sigma_{\pi_\theta}^{-1}\big(\mu_\theta(\mathbf{o}_t) - \mu_\lambda(\mathbf{x}_t)\big) + \text{const}\bigg].$$

The KL-divergence term in this objective is quadratic in $\mathbf{u}_t$ and linear in the covariance Σ_t, with an entropy maximization term $-\ln|\Sigma_t|$. This is precisely the objective that is optimized by the maximum entropy variant of iLQG [14], and optimization requires us only to expand the cost-to-go J_t to second order, which is a standard procedure in iLQG.

Training the learner's policy: We want the learner's policy π_θ to approach the optimal policy $\pi^*(\mathbf{u}|\mathbf{x}_t)$. We can estimate a (locally) optimal policy π^* at each state $\mathbf{x}_t$ with iLQG, simply by repeating the optimization at each time step but excluding the KL-divergence term. During the supervised learning phase, we minimize the KL-divergence between the learner π_θ and the precomputed near-optimal policies π^* at the observations stored in the dataset $\mathcal{D}$:

$$\theta \leftarrow \arg\min_\theta \sum_{(\mathbf{x}_t, \mathbf{o}_t) \in \mathcal{D}} D_{\text{KL}}\big(\pi_\theta(\mathbf{u}|\mathbf{o}_t) || \pi^*(\mathbf{u}|\mathbf{x}_t)\big). \quad (3)$$

Since both π_θ and π^* are conditionally Gaussian, the KL-divergence can be expressed in closed-form:

$$\min_\theta \frac{1}{2} \sum_{(\mathbf{x}_t, \mathbf{o}_t) \in \mathcal{D}} \big(\mu^*(\mathbf{x}_t) - \mu_\theta(\mathbf{o}_t)\big)^\mathsf{T}\Sigma_{\pi^*}^{-1}\big(\mu^*(\mathbf{x}_t) - \mu_\theta(\mathbf{o}_t)\big)$$
$$+ \text{tr}\big(\Sigma_{\pi^*}^{-1}\Sigma_{\pi_\theta}\big) + \ln\bigg(\frac{|\Sigma_{\pi^*}|}{|\Sigma_{\pi_\theta}|}\bigg) + \text{const}.$$

Ignoring the terms that do not involve the learner policy mean $\mu_\theta(\mathbf{o}_t)$, the objective function can be rewritten in the form of a weighted Euclidean loss:

$$\min_\theta \sum_{(\mathbf{x}_t, \mathbf{o}_t) \in \mathcal{D}} ||\mu^*(\mathbf{x}_t) - \mu_\theta(\mathbf{o}_t)||^2_{\Sigma_{\pi^*}^{-1/2}}.$$

approach	teacher policy	supervision policy
supervised learning	π^*	π^*
DAgger	π_{MIX}	π^*
DAgger + coaching	π_{MIX}	π_{COACH}
PLATO	π_λ	π^*

TABLE I: **Overview of teacher-based policy optimization methods**: For PLATO and each prior approach, we list which teacher policy is used for sampling trajectories and which supervision policy is used for generating training actions from the sampled trajectories. Note that the prior methods execute the mixture policy π_{MIX}, which requires running the learned policy π_θ, potentially executing dangerous actions when π_θ is not fully trained.

This optimization can then be solved using standard regression methods. In our experiments, μ_θ is represented by a neural network, and the above optimization problem corresponds to standard neural network regression, solvable by stochastic gradient descent. The covariance of π_θ can be solved for in closed form, and corresponds to the inverse of the average precisions of π^* at the training points [11].

B. Relationship to previous work

The motivation behind PLATO is most similar to the MPC variant of guided policy search (MPC-GPS) [12]. However, PLATO lifts a major limitation of MPC-GPS. MPC-GPS requires the ability to deterministically reset the environment into one of a small set of initial states. MPC-GPS requires deterministic resets because the KL-divergence term is evaluated using a linearization around each rollout. Deterministic episodic resets can be complex, time-consuming, or even impossible in the real world. For example, imagine a robot learning to navigate a human crowd; deterministic resets would require having the crowd walk through the same paths in each episode. Not requiring such resets is a major advantage. Furthermore, even when deterministic resets are feasible, PLATO empirically outperforms MPC-GPS (Section VI).

Formally, PLATO can also be viewed as a generalization of the Dataset Aggregation (DAgger) algorithm [10], which samples trajectories according to the mixture policy $\pi_{\text{MIX}i} = \beta_i\pi^* + (1 - \beta_i)\pi_{\theta i}$. The training data is generated from the observations sampled by executing $\pi_{\text{MIX}i}$ but labelled with actions from π^*. DAgger converges if $\frac{1}{N}\sum_{i=1}^N \beta_i \to 0$ as $N \to \infty$. Coaching [13], a related extension to DAgger, modifies the supervision policy π^* to adapt to the learned policy π_θ by labelling the training data with a coach policy π_{COACH} that encourages the action training labels to be similar to the actions $\pi_{\theta i}$ would choose. Our empirical evaluation shows that PLATO outperforms coaching.

Another distinction of PLATO is the use of an adaptive MPC policy $\pi_\lambda^{1:T}$ to select the actions at each time step, rather than the mixture policy π_{MIX} used in the prior methods. As demonstrated in our evaluation, this adaptive MPC policy allows PLATO to robustly avoid catastrophic failure during training, which is particularly important in safety-critical domains. Our experiments also demonstrate that policies trained using PLATO empirically outperform policies trained by either DAgger or coaching. Table I summarizes the teacher and supervision policies used by PLATO and prior work.

V. Theoretical Analysis

In this section, we present a proof that the policy π_θ learned by PLATO converges to a policy with bounded cost. This proof extends the result by Ross et al. [10], which only admits mixture policies, to our adaptive MPC policy $\pi_\lambda^{1:T}$.

Given a policy π, we denote d_π^t as the state distribution at time t when executing policy π from time 1 to $t-1$. Define the cost function $c(\mathbf{x}_t, \mathbf{u}_t)$ as a function of state $\mathbf{x}_t$ and control $\mathbf{u}_t$, with $c(\mathbf{x}_t, \mathbf{u}_t) \in [0, 1]$ without loss of generality. We wish to learn a policy $\pi_\theta(\mathbf{u}|\mathbf{o}_t)$ that minimizes the total expected cost over time horizon T:

$$J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}_t \sim d_\pi^t}[\mathbb{E}_{\mathbf{u}_t \sim \pi_\theta(\mathbf{u}|\mathbf{o}_t)}[c(\mathbf{x}_t, \mathbf{u}_t)|\mathbf{x}_t]].$$

Let $J_t(\pi, \tilde{\pi})$ denote the expected cost of executing π for t time steps, and then executing $\tilde{\pi}$ for the remaining $T-t$ time steps, and let $Q_t(\mathbf{x}, \pi, \tilde{\pi})$ denote the cost of executing π for one time step starting from initial state $\mathbf{x}$, and then executing $\tilde{\pi}$ for the remaining $t-1$ time steps. We assume the cost-to-go difference between the learned policy and the optimal policy is bounded: $Q_t(\mathbf{x}, \pi_\theta, \pi^*) - Q_t(\mathbf{x}, \pi^*, \pi^*) \leq \delta$. In the worst case, δ is $O(T)$ and PLATO (as well as similar methods such as DAgger) will not outperform supervised learning. However, if π^* is able to quickly recover from mistakes made by π_θ, δ will be $O(1)$ [10].

When optimizing Equation 1 to obtain the teacher policy π_λ, we choose λ such that $D_{\mathrm{KL}}(\pi_\lambda(\mathbf{u}|\mathbf{x})||\pi_\theta(\mathbf{u}|\mathbf{o})) \leq \epsilon_{\lambda\theta}$ for all state-observation pairs $(\mathbf{x}, \mathbf{o})$. We can always guarantee this bound when optimizing Equation 1 because $D_{\mathrm{KL}}(\pi_\lambda(\mathbf{u}|\mathbf{x})||\pi_\theta(\mathbf{u}|\mathbf{o})) \to 0$ as $\lambda \to \infty$.

When optimizing the supervised learning objective in Equation 3 to obtain the learner policy π_θ, we assume the supervised learning objective function error is bounded by a constant $D_{\mathrm{KL}}(\pi_\theta(\mathbf{u}|\mathbf{o})||\pi^*(\mathbf{u}|\mathbf{x})) \leq \epsilon_{\theta*}$ for all states $\mathbf{x}$ (and corresponding observations $\mathbf{o}$) in the dataset, which were sampled from the teacher policy distribution d_{π_λ}. Since the policy π_θ is trained with supervised learning precisely on these states $\mathbf{x} \sim d_{\pi_\lambda}$, this bound $\epsilon_{\theta*}$ corresponds to assuming that the learner policy π_θ attains bounded training error.

Let $l(\mathbf{x}, \pi_\theta, \pi^*)$ denote the expected 0-1 loss of π_θ with respect to π^* in state $\mathbf{x}$: $\mathbb{E}_{\mathbf{u}_\theta \sim \pi_\theta(\mathbf{u}|\mathbf{o}), \mathbf{u}^* \sim \pi^*(\mathbf{u}|\mathbf{x})}[\mathbf{1}[\mathbf{u}_\theta \neq \mathbf{u}^*]]$. We note that the total variation divergence is an upper bound on the 0-1 loss [21] and the KL-divergence is an upper bound on the total variation divergence [22]. Therefore for all states $\mathbf{x} \sim d_{\pi_\lambda}$ in the dataset used for supervised learning, the 0-1 loss can be upper bounded:

$$\begin{aligned}
l(\mathbf{x}, \pi_\theta, \pi^*) &= \mathbb{E}_{\mathbf{u}_\theta \sim \pi_\theta(\mathbf{u}|\mathbf{o}), \mathbf{u}^* \sim \pi^*(\mathbf{u}|\mathbf{x})}[\mathbf{1}[\mathbf{u}_\theta \neq \mathbf{u}^*]] \\
&\leq D_{\mathrm{TV}}(\pi_\theta(\mathbf{u}|\mathbf{o})||\pi^*(\mathbf{u}|\mathbf{x})) \\
&\leq \sqrt{D_{\mathrm{KL}}(\pi_\theta(\mathbf{u}|\mathbf{o})||\pi^*(\mathbf{u}|\mathbf{x}))} \\
&\leq \sqrt{\epsilon_{\theta*}}.
\end{aligned}$$

We also note the state distribution bound $||d_\pi^t - d_{\tilde{\pi}}^t||_1 \leq 2t\sqrt{D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi})}$ proven in [9]. This lemma implies that for an arbitrary function $f(\mathbf{x})$, $E_{\mathbf{x} \sim d_\pi^t}[f(\mathbf{x})] \leq E_{\mathbf{x} \sim d_{\tilde{\pi}}^t}[f(\mathbf{x})] + 2f^{\max}t\sqrt{D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi})}$

We can then prove the following theorem:

Theorem 5.1: Let the cost-to-go $Q_t(\mathbf{x}, \pi_\theta, \pi^*) - Q_t(\mathbf{x}, \pi^*, \pi^*) \leq \delta$ for all $t \in \{1, ..., T\}$. Then for PLATO, $J(\pi_\theta) \leq J(\pi^*) + \delta\sqrt{\epsilon_{\theta*}}O(T) + O(1)$.

Proof:

$$\begin{aligned}
J(\pi_\theta) &= J(\pi^*) + \sum_{t=0}^{T-1} J_{t+1}(\pi_\theta, \pi^*) - J_t(\pi_\theta, \pi^*) \\
&= J(\pi^*) + \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x} \sim d_{\pi_\theta}^t}[Q_t(\mathbf{x}, \pi_\theta, \pi^*) - Q_t(\mathbf{x}, \pi^*, \pi^*)] \\
&\leq J(\pi^*) + \delta \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x} \sim d_{\pi_\theta}^t}[l(\mathbf{x}, \pi_\theta, \pi^*)] \quad (4a) \\
&\leq J(\pi^*) + \delta \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x} \sim d_{\pi_\lambda}^t}[l(\mathbf{x}, \pi_\theta, \pi^*)] + 2l^{\max}t\sqrt{\epsilon_{\lambda\theta}} \quad (4b) \\
&\leq J(\pi^*) + \delta \sum_{t=1}^{T} \sqrt{\epsilon_{\theta*}} + 2t\sqrt{\epsilon_{\theta*}}\sqrt{\epsilon_{\lambda\theta}} \quad (4c) \\
&= J(\pi^*) + \delta T\sqrt{\epsilon_{\theta*}} + \delta T(T+1)\sqrt{\epsilon_{\theta*}}\sqrt{\epsilon_{\lambda\theta}}
\end{aligned}$$

Equation 4a follows from the fact that the expected 0-1 loss of π_θ with respect to π^* is the probability that π_θ and π^* pick different actions in $\mathbf{x}$; when they choose different actions, the cost-to-go increases by $\leq \delta$. Equation 4b follows from the state distribution bound proven in [9]. Equation 4c follows from the upper bound on the 0-1 loss.

Although we do not get to choose $\epsilon_{\theta*}$ because that is a property of the supervised learning algorithm and the data, we are able to choose $\epsilon_{\lambda\theta}$ by varying parameter λ. If we choose λ such that $\epsilon_{\lambda\theta} = O(\frac{1}{T^2})$. We therefore have

$$J(\pi_\theta) \leq J(\pi^*) + \delta\sqrt{\epsilon_{\theta*}}O(T) + O(1) \quad . \qquad \square$$

As with DAgger, in the worst case $\delta = O(T)$. However, in many cases $\delta = O(1)$ or is sub-linear in T, for instance if π^* is able to quickly recover from mistakes made by π_θ. We also note that this bound, $O(T)$, is the same as the bound obtained by DAgger, but without actually needing to directly execute π_θ at training time. Compared to supervised learning with bound $O(T^2)$ [10], PLATO trains the policy at states closer to those induced under its own distribution.

VI. Experiments

We evaluate PLATO on a series of simulated quadrotor navigation tasks. MPC is a standard choice for quadrotor control [23] because approximate models are typically known in advance from standard rigid body physics and the vehicle specifications. However, effective use of MPC requires explicit state estimation and can be computationally intensive. It is therefore very appealing to be able to train an entirely feedforward, reactive policy to control a quadrotor performing navigation in obstacle-rich environments, directly in response to raw sensor inputs. During training, the vehicle might be placed in a known, instrumented training environment to collect data using MPC, while at test time, the learned feedforward policy could control the aircraft directly from raw observations. This makes simulated quadrotor navigation an ideal domain in which to compare PLATO to prior work.
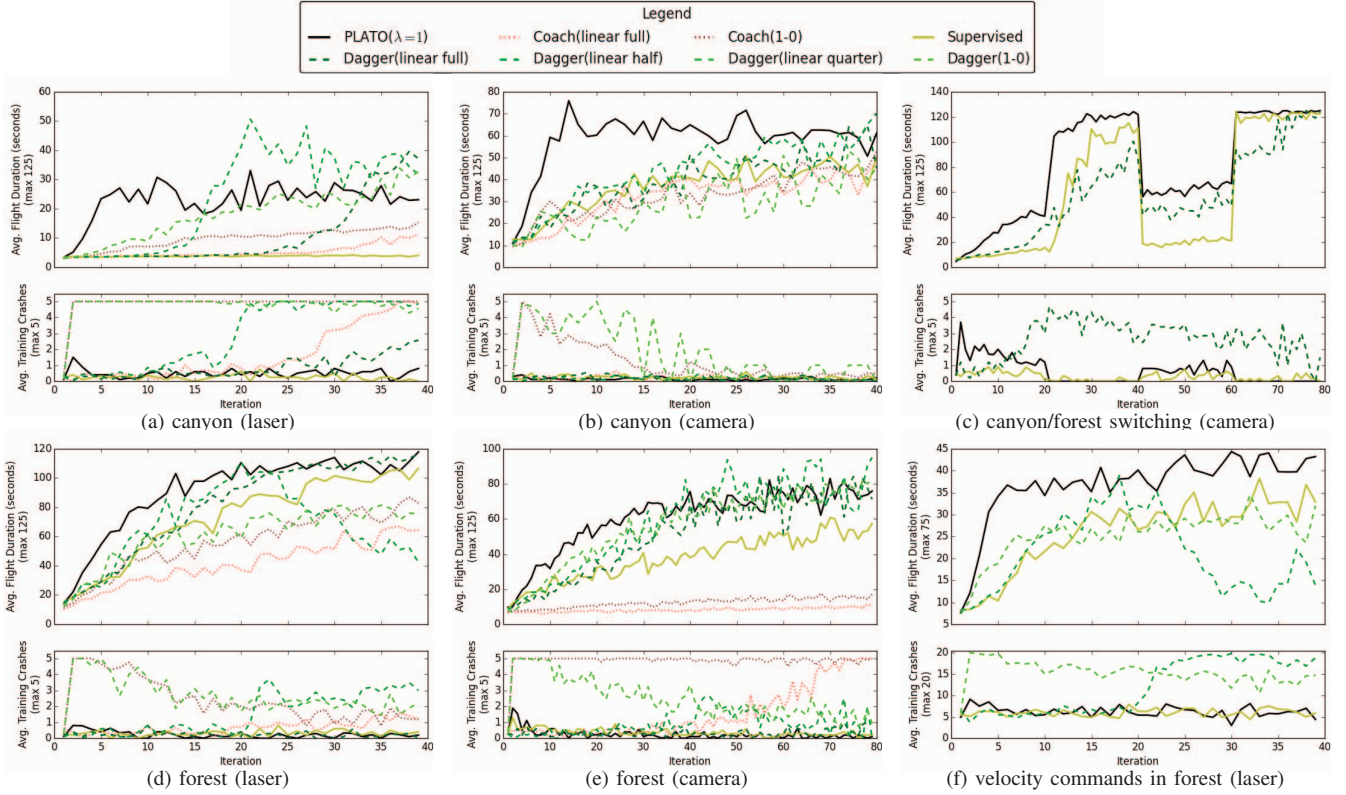
Fig. 2: **Experiments**: We compare PLATO to baseline methods in a winding canyon, a dense forest, and an alternating canyon/forest. For each scenario and learning method, we trained 10 different policies using different random seeds. Each iteration required 2 minutes of flight time. Then for each policy, we evaluated the neural network policy trained at each iteration by flying through the scenario 20 times. Therefore each datapoint corresponds to 200 samples.

Prior methods and baselines: We compare PLATO to four methods. The first method is DAgger, which, as discussed in Section IV-B, executes a mixture of the learned policy and teacher policy, which in this case is MPC (without a KL-divergence term). DAgger has previously been used for learning quadrotor control policies from human demonstrations [15]. While DAgger carries the same convergence guarantees as PLATO, successful use of DAgger requires the learned policy to be executed at training time, before the policy has converged to a near-optimal behavior. The second method is the coaching algorithm of [13] which, like DAgger, executes a mixture of the learned and teacher policies, but supervises the learner using the adapted policy. In these experiments, we chose the coaching policy π_{COACH} to be the teacher policy π_λ from PLATO. For both DAgger and coaching, we must choose the mixing parameter β_i at each iteration i. Since the performance of these algorithms is quite sensitive to the schedule of the β_i parameter, we include four schedules for comparison: three linear schedules that interpolate β_i from 1 at the first iteration to 0 at the last iteration ("linear full"), the halfway iteration ("linear half"), and the quarter-way iteration ("linear quarter"), as well as the more standard "1-0" schedule that sets $\beta_i = 1[i = 1]$. The third method is MPC-GPS [12], which, unlike PLATO, DAgger and coaching, requires deterministic resets during training (Figure 4b). In addition to these prior methods, we also compare our approach to a standard supervised learning baseline, which always executes the MPC policy without any adaptation. For all experiments, we assume additive Gaussian noise is applied to both controls and observations.

Policy representation: For all of the methods, we represent π_θ as a conditional Gaussian policy, with a constant covariance and a mean given by a neural network function of the observation $\mathbf{o}_t$. The network has two fully connected hidden layers of size 40 with ReLU activations [24]. The loss function is the weighted euclidean loss (see Section IV-A). We used the Caffe [25] framework and the ADAM solver [26]. Each iteration was trained using the final weights from the previous iteration.

Experimental domains: The comparisons are conducted on two test environments: a winding canyon with randomized turns, and a dense forest of cylindrical trees with randomized positions. An example environment is shown in Figure 1. The canyon changes direction up to $\frac{\pi}{4}$ radians every 0.5m. The forest is composed of 0.5m radius cylinders with an average spacing of 2.5m. The target velocity is 6m/s in the canyon and 2m/s in the forest.

The dynamical system is a quadrotor with dynamics described by [27]. The state of the vehicle $\mathbf{x} \in \mathbb{R}^{13}$ consists of the position and orientation, as well as their time derivatives, and the control $\mathbf{u} \in \mathbb{R}^4$ consists of motor velocities. The observations $\mathbf{o}$ consist of orientation, linear velocity, angular velocity and either (i) a set of 30 equally spaced 1-d laser depth scanners arranged in 180 degree fan in front of the vehicle ($\mathbf{o} \in \mathbb{R}^{40}$) or (ii) a 5×20 grayscale camera image ($\mathbf{o} \in \mathbb{R}^{110}$). Learning neural network policies with these

observations forces the policies to perform both perception and control, since success on each of the domains requires avoiding obstacles using only raw sensory input.

The cost function for the MPC teacher encourages the quadrotor to fly at a specific linear velocity and orientation while minimizing control effort and avoiding collisions:

$$L(\mathbf{x}, \mathbf{u}) = 10^3 ||\mathbf{x}_{\text{LINVEL}} - \mathbf{x}_{\text{LINVEL}}^*||_2^2 + 10^3 ||\mathbf{x}_{\text{HEIGHT}} - \mathbf{x}_{\text{HEIGHT}}^*||_2^2 +$$
$$10^4 ||\mathbf{x}_{\text{QUAT}} - \mathbf{x}_{\text{QUAT}}^*||_2^2 + 250 ||\mathbf{x}_{\text{ANGVEL}}||_2^2 +$$
$$5^{-3} ||\mathbf{u} - \mathbf{u}_{\text{HOVER}}||_2^2 +$$
$$10^3 \max(d_{\text{SAFE}} - \text{signed-distance}(\mathbf{x}), 0),$$

where $\mathbf{x}_{\text{LINVEL}}, \mathbf{x}_{\text{HEIGHT}}, \mathbf{x}_{\text{QUAT}}, \mathbf{x}_{\text{ANGVEL}}$ are the linear velocity, height, orientation, and angular velocity of the state $\mathbf{x}$, respectively; $\mathbf{x}_{\text{LINVEL}}^*, \mathbf{x}_{\text{HEIGHT}}^*, \mathbf{x}_{\text{QUAT}}^*$ are the target linear velocity, height, and orientation, respectively; and $\mathbf{u}_{\text{HOVER}}$ is the rotor velocity when the quadrotor is hovering. The final term is a hinge loss on the distance of the quadrotor to the nearest obstacle; there is no penalty if the nearest obstacle is further than d_{SAFE}.

Performance of learned policies: In Figures 2a, 2b, 2d, and 2e, we present the mean time to failure (MTTF) of the learned policy π_θ on the canyon and forest environments using the laser or camera sensors. The graphs show the MTTF of each policy at each iteration of the learning process, averaged over 10 training runs of each method with 20 repetitions each. Failure occurs when the quadrotor crashes into an obstacle, with the maximum flight time for each domain listed on the graphs. The results indicate that the PLATO algorithm is able to learn effective policies faster, and converges to a solution that is better than or comparable to the baseline methods. For some choices of β schedule and supervision scheme, some DAgger variants achieve similar final MTTFs, but always at a slower rate and, as discussed next, with significantly more training crashes.

Robustness during training: In Figures 2a, 2b, 2d, and 2e, we show the number of crashes experienced during training at each iteration. PLATO on average experiences less than one crash per iteration, comparable in performance to the baseline MPC method (supervised learning), indicating that mimicking the learner with a KL-divergence penalty does not substantially degrade the robustness of MPC. In contrast, both DAgger and coaching begin to experience a substantial number of failures when the mixing constant β drops. By carefully selecting the schedule for β, the number of crashes can be reduced.

However, even with a carefully chosen schedule, the prior methods are vulnerable to non-stationary training environments, as illustrated in Figure 2c. In this experiment, the vehicle switches from the canyon to the forest halfway through training, and then switches back to the forest. Prior methods that directly execute π_θ during training experience many crashes because a policy trained only on the canyon cannot succeed on the forest without additional training. However, PLATO experiences on average less than one crash per episode because PLATO is able to automatically switch to more off-policy behavior when encountering novel scenarios. While this example might appear pathological, it is in fact

a plausible training setup for a real quadrotor exploring a varied environment, such as different floors of a building. If the walls on one floor are painted, e.g., a different color than the rest, the learned policy could easily experience a catastrophic failure when entering the floor for the first time, even if it was consistently successful on preceding floors.

Policies with user velocity commands: Figure 2f shows the performance of PLATO when learning policies that take an additional input to simulate high-level user control in the form of the desired velocity of the quadrotor. These policies are useful because instead of training multiple policies for different target velocities, we can train one generalizable policy. This input modifies the cost function used by MPC, producing command-aware supervision. During training, the commands vary in the range of ± 1 m/s sideways and 1 to 2.5 m/s forward. At test time, we sample velocity commands uniformly at random; the velocity commands are re-sampled whenever the quadrotor reaches the current sampled velocity. The results indicate that PLATO can successfully learn such policies, outperforming prior methods and again minimizing the number of crashes during training.

Sensitivity to KL-divergence weight: Recall that λ determines the degree to which MPC prioritizes following the learner π_θ versus performing the desired task. As $\lambda \to 0$, PLATO approaches standard supervised learning and is thus safe, while as $\lambda \to \infty$, PLATO approaches DAgger 0-1. In practice, to choose lambda, we start with $\lambda = 0$, and then increase λ until the cost of the behavior starts to increase. Figure 3 compares different non-limiting settings of λ, while we refer to Figure 2 for limiting cases for $\lambda = 0$ and $\lambda = \infty$. The results suggest that a relatively broad range of λ values produces successful policies.
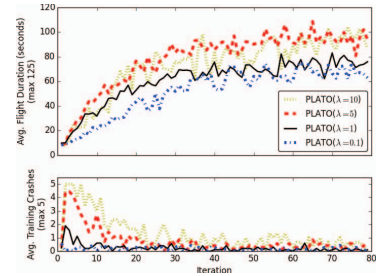

Fig. 3: Effect of KL-divergence weight λ.

Comparison with training on full state: Figure 4a shows a comparison where the policy maps state to action using an oracle SLAM algorithm that provides perfect state information and a local 2D distance map of the obstacles. The observation-based policy substantially outperforms the policy that learns to map the state to action, even with an oracle SLAM algorithm. Although the state and obstacle map are sufficient to choose good actions, this mapping is much harder to learn. Of course, alternative full state representations that are
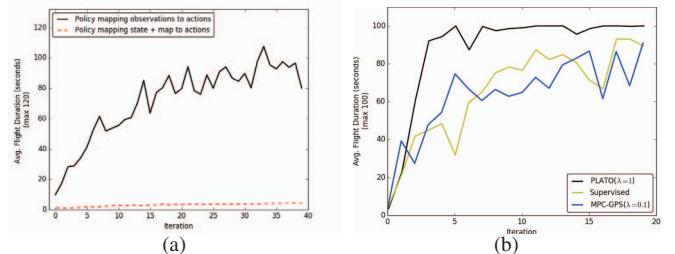

Fig. 4: Comparisons with (a) training on full state and (b) MPC-GPS.

carefully engineered to the task may perform better, but this experiment demonstrates that, at least in some cases, mapping observations directly to actions without going through full state estimation can lead to better performance.

Comparison with MPC-GPS: MPC-GPS [12] cannot directly be evaluated on the domains described above, because training must occur in episodes with deterministic resets (see Section IV-B). We constructed a fixed-length episodic variant of the forest task where MPC-GPS was allowed to use deterministic resets. Besides not requiring an episodic formulation or deterministic resets, the comparison in Figure 4b shows that PLATO substantially outperforms the policy learned by MPC-GPS in terms of MTTF.

Supplementary material, including a video, can be viewed online: `sites.google.com/site/platopolicy`.

VII. DISCUSSION

In this paper, we presented PLATO, a continuous, reset-free algorithm for learning complex, high-dimensional policies that combine perception and control into a single expressive function approximator, such as a deep neural network. PLATO uses a trajectory optimization teacher to provide supervision to a standard supervised learning algorithm, allowing for a simple and data-efficient learning method. The teacher adapts to the behavior of the neural network policy to ensure that the distribution over states and observations is sufficiently close to the learned policy, allowing for a bound on the long-term performance of the learned policy. Our empirical evaluation on a simulated quadrotor demonstrates that PLATO outperforms a number of previous methods, both in terms of the robustness and effectiveness of the final policy, and in terms of the safety of the training procedure.

PLATO has two key advantages that make it well-suited for learning control policies for real-world robotic systems. First, since the learned neural network policy does not need to be executed at training time, the method benefits from the robustness of model-predictive control (MPC), minimizing catastrophic failures at training time. This is particularly important when the training state distribution is non-stationary. Methods that execute the learned policy, such as DAgger, can suffer a catastrophic failure when the agent encounters novel observations. Mitigating these issues typically requires hand-designed safety mechanisms, while PLATO automatically switches to a more off-policy behavior.

The second advantage of PLATO is that the learned policy can use a different set of observations than MPC. Effective use of MPC requires observing or inferring the full state of the system, which might be accomplished, for instance, by instrumenting the environment with motion capture, or using a known map with relocalization [28]. The policy, however, can be trained directly on raw input from onboard sensors, forcing it to perform both perception and control. Once trained, such a policy can be used in uninstrumented natural environments.

One of the most appealing prospects of learning expressive neural network policies is the possibility of acquiring real-world policies that directly use rich sensory inputs. Because of this, one very interesting avenue for future work is to apply PLATO on real physical platforms.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and R. M., "Playing atari with deep reinforcement learning," in *Workshop on Deep Learning, NIPS*, 2013.

[3] A. Giusti, J. Guzzi, D. C. Ciresan, F. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Caro, D. Scaramuzza, and L. Gambardella, "A machine learning approach to visual perception of forest trails for mobile robots," in *IEEE Robotics and Automation Letters*, 2016.

[4] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: learning affordance for direct perception in autonomous driving," in *ICCV*, 2015.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, 2016.

[6] 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," in *ICLR*, 2016.

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

[8] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," in *Foundations and Trends in Robotics*, 2011.

[9] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," in *ICML*, 2015.

[10] S. Ross, G. Gordon, and J. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *AISTATS*, 2011.

[11] S. Levine and V. Koltun, "Guided policy search," in *ICML*, 2013.

[12] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *ICRA*, 2016.

[13] H. He, J. Eisner, and H. Daume, "Imitation learning by coaching," in *NIPS*, 2012.

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

[15] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *ICRA*, 2013.

[16] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning," in *NIPS*, 2014.

[17] S. Levine and V. Koltun, "Variational policy search via trajectory optimization," in *NIPS*, 2013.

[18] D. Mayne, M. M. Seron, and S. V. Rakovic, "Robust model predictive control of constrained linear systems with bounded disturbances," in *Automatica*, 2005.

[19] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference*, 2005.

[20] H. Kappen, V. Gomez, and M. Opper, "Optimal control as a graphical model inference problem," in *Machine Learning*, 2012.

[21] X. Nguyen, M. J. Wainwright, and M. I. Jordan, "Divergences, surrogate loss functions and experimental design," in *NIPS*, 2005.

[22] D. Pollard, "Asymptopia: an exposition of statistical asymptotic theory," 2000. [Online]. Available: stat.yale.edu/~pollard/Books/Asymptopia/

[23] M. Mueller and R. D'Andrea, "A model predictive controller for quadrotor state interception," in *ECC*, 2013.

[24] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[25] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[26] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[27] P. Martin and E. Salaun, "The true role of accelerometer feedback in quadrotor control," in *ICRA*, 2010.

[28] B. Williams, G. Klein, and I. Reid, "Real-time slam relocalisation," in *ICCV*, 2007.