Actor-Critic PAC Robust Policy Search

Matthew Sheckells[1] and Marin Kobilarov[1]

Abstract— **This work studies an approach for computing provably robust control laws for robotic systems operating in uncertain environments. We develop an actor-critic style policy search algorithm based on the idea of minimizing an upper confidence bound on the negative expected advantage of a control policy at each policy update iteration. This new algorithm is a reformulation of Probably-Approximately-Correct Robust Policy Search (PROPS) and, unlike PROPS, allows for both step-based evaluation and step-based sampling strategies in policy parameter space, enabled by the use of Generalized Advantage Estimation and Generalized Exploration. As a result, the new algorithm is more data efficient and is expected to compute higher quality policies faster. We empirically evaluate the algorithm in simulation on a challenging robot navigation task using a high-fidelity deep stochastic model of an agile ground vehicle and compare its performance to the original trajectory-based PROPS.**

I. Introduction

Policy search is a method for computing the optimal control parameters of a robotic system operating in an unknown, uncertain environment. A policy search algorithm can be factored into three components, consisting of the exploration, policy evaluation, and policy update strategies. A brief taxonomy of these components is given next (see [1] for a more detailed taxonomy of policy search algorithms).

a) Exploration Strategy: An algorithm can explore in policy parameter space or in action space. Popular algorithms like Trust Region Policy Optimization (TRPO) [2], A3C [3], and Proximal Policy Optimization (PPO) [4] explore in action space by forming a stochastic policy which outputs a distribution over controls conditioned on the current state. The algorithm that we develop in this work, along with CMA-ES [5], [6], Relative Entropy Policy Search (REPS) [7], and Parameter-exploring Policy Gradients (PEPG) [8] explore in parameter space. When exploring in policy parameter space, a policy can be sampled at the beginning of a trajectory and then held constant or it can be re-sampled at each time-step. *Episode-based* sampling strategies result in smoother, more realistic control trajectories but can only evaluate one policy per trajectory, while *step-based* sampling strategies can evaluate multiple policies per trajectory and may be less vulnerable to getting stuck in local minima. One can use a generalized exploration approach to achieve a more balanced trade-off between the advantages of step-based and episode-based policy sampling [9].

b) Policy Evaluation Strategy: Policy search algorithms can evaluate trajectories in either an episode-based or step-based manner. That is, an algorithm may estimate the quality of whole trajectories or the quality of single actions when computing a policy update. Episode-based evaluation strategies could result in undesirable high variance of the estimated cost of a trajectory due to the compounding uncertainty of rolling out a trajectory with stochastic dynamics. It is possible to combine step-based evaluation strategies with episode-based sampling strategies as in the PoWER [10] and PI^2 [11] algorithms.

c) Policy Update Strategy: Policy gradient methods like PEPG, DDPG [12], and REINFORCE [13] estimate the gradient of the expected reward with respect to the policy parameters and then use it in a gradient descent algorithm. Gradient-free methods, such as Reward-weighted Regression (RwR) [14] and CMA-ES treat the expected reward as a black-box function to be optimized over the policy parameters. Information-theoretic methods like REPS and TRPO update the policy parameters while limiting the deviation of the trajectory distribution (in terms of KL divergence) to a user-specified trust region. Actor-critic methods use a learned estimate of the value function (the critic) in order to update a separate policy (the actor). This class of methods includes, e.g. A3C, TRPO, and [15], [16], [17]. Methods like High Confidence Policy Improvement [18] and Probably-Approximately-Correct Robust Policy Search (PROPS) [19] minimize an upper confidence bound on the expected cost of trajectories. This update strategy is particularly useful because it provides a bound on the expected performance of the policy which it computes, and it does not require performance-sensitive user-tunable parameters like a descent step-size or trust region constraint multiplier.

This work presents a novel policy search algorithm based off of PROPS. Unlike PROPS, this algorithm uses a step-based policy evaluation strategy which relies on the estimation of the advantage function for each step in a trajectory. The algorithm estimates these unknown advantages using Generalized Advantage Estimation (GAE) [20]. This approach results in an advantage estimator that has a tunable bias-variance trade-off. We refer to this new algorithm as Actor-Critic PROPS (AC-PROPS) since the advantage estimation makes use of a learned value function in order to update the policy. AC-PROPS uses a generalized exploration strategy [9] that allows the user to interpolate between step-based and episode-based exploration. §II develops some preliminary theory while §III introduces the AC-PROPS algorithm. §IV evaluates AC-PROPS on a challenging robot navigation task and compares its performance to that of

[1]M. Sheckells and M. Kobilarov are with the Department of Computer Science and the Department of Mechanical Engineering, Johns Hopkins University, 3400 N Charles Str, Baltimore, MD 21218, USA `msheckells|marin@jhu.edu`

PROPS.

II. Preliminaries

A. Markov Decision Process (MDP)

Consider a finite horizon Markov Decision Process (MDP) defined by a state $x \in \mathbb{R}^n$, control inputs $u \in \mathbb{R}^m$, reward function $r(x, u)$, transition dynamics $p(x'|x, u)$, and trajectory length N. Consider a policy $u = \pi(x; \xi)$ parameterized by a vector ξ which maps a state x to controls u. Trajectories $\tau \triangleq (x_0, u_0, \dots, u_{N-1}, x_N)$ are generated by sampling an initial state $x_0 \sim p_0(\cdot)$, using the policy π to generate controls u_t, and following the transition dynamics $p(x_{t+1}|x_t, u_t)$ to generate the next state along with its associated reward $r(x_t, u_t)$, for $t = 0, \dots, N-1$. Our goal is to choose a set of policy parameters ξ which maximize the expected total reward

$$\mathbb{E}_{(x_0,u_0,\dots)\sim p(\tau|\xi)}\left[\sum_{t=0}^{N} r(x_t, u_t)\right],$$

with $p(\tau|\xi) = p_0(x_0)\prod_{t=0}^{N} p(x_{t+1}|x_t, u_t)$ where $u_t = \pi(x_t; \xi)$. One approach for maximizing this objective is Iterative Stochastic Policy Optimization (ISPO), which is described in the following section.

B. Iterative Stochastic Policy Optimization (ISPO)

ISPO is a standard approach for policy search which optimizes the expected reward of a policy by performing the search in policy parameter space. This method forms a surrogate stochastic model $\rho(\xi|\nu)$ over policy parameters ξ parameterized by a vector ν and uses this distribution to explore in policy space. The policies themselves are deterministic.

A common way to define policy search in policy parameter space is through the optimization

$$\nu^* = \arg\min_{\nu} \mathbb{E}_{\tau,\xi\sim p(\cdot|\nu)}[J(\tau)],$$

where $J = \sum_{t=0}^{N} -r(x_t, u_t)$ is a cost function encoding the desired behavior. The surrogate stochastic model induces a joint density $p(\tau, \xi|\nu) = p(\tau|\xi)\pi(\xi|\nu)$ which contains the natural stochasticity of the system $p(\tau|\xi)$ and artificial control-exploration stochasticity $\pi(\xi|\nu)$ due to the surrogate model.

Here, we employ a cost function (i.e. a negative of the total reward) since the proposed method can be more easily understood in terms of *minimizing* a confidence bound. Algorithm 1 shows the general framework for solving the problem when using an episode-based evaluation strategy while Algorithm 2 shows the framework when using a step-based evaluation strategy.

A key step in ISPO is computing the new policy based on the observed costs of previously executed policies. The specific implementation of the update step (e.g. step 5 of Algorithm 1) corresponds to different policy search algorithms such as RwR, REPS, or PROPS. AC-PROPS follows the framework described in Algorithm 2 and is formulated in detail in §III.

Algorithm 1 ISPO with Episode-based Policy Evaluation

```
1: Initialize hyper-distribution $\nu_0$, $i \leftarrow 0$
2: while Bound on expected cost greater than threshold do
3:     for $j = 1, \dots, M$ do
4:         Sample trajectory $(\xi_j, \tau_j) \sim p(\cdot|\nu_i)$
5:     Compute a new policy $\nu_{i+1}$ using observed trajectory
       costs $\{J(\tau_1), \dots, J(\tau_M)\}$
6:     Set $i = i + 1$
```

C. Objective in Terms of Advantages

Now, we will formulate the learning objective in terms of step-wise advantages. We will use modified definitions of the value function V^ν, state-action value function Q^ν, and advantage function A^ν which reflect that a trajectory is generated by a deterministic policy that is sampled from a distribution $\rho(\cdot|\nu)$ as opposed to the traditional definition which assumes a fixed, stochastic policy. Note that a policy ξ can either be sampled in a step-based or episode-based manner. That is, either a new policy can be sampled from $\rho(\cdot|\nu)$ at each time step or a new policy can be sampled from $\rho(\cdot|\nu)$ once at the beginning of an episode and then held fixed until termination of the episode.

So, the expected reward of a policy distribution is

$$R(\nu) = \mathbb{E}_{(x_0,u_0,\xi_0,\dots)\sim p(\cdot|\nu)}\left[\sum_{t=0}^{N} r_t(x_t, u_t)\right]. \tag{1}$$

The expected total reward starting at state x and following the policy distribution $\rho(\cdot|\nu)$ is denoted by the value function $V^\nu(x)$. That is

$$V^\nu(x_t) = \mathbb{E}_{(u_t,\xi_t,x_{t+1}\dots)\sim p(\cdot|\nu)}\left[\sum_{k=t}^{N} r_k(x_k, u_k)\right].$$

The state-action value function is given by

$$Q^\nu(x_t, u_t) = \mathbb{E}_{(x_{t+1},u_{t+1},\xi_{t+1},\dots)\sim p(\cdot|\nu)}\left[\sum_{k=t}^{N} r_k(x_k, u_k)\right].$$

The advantage function is then defined by

$$A^\nu(x_t, u_t) = Q^\nu(x_t, u_t) - V^\nu(x_t)$$

Algorithm 2 ISPO with Step-based Policy Evaluation

```
1: Initialize hyper-distribution $\nu_0$, $i \leftarrow 0$
2: while Bound on expected cost greater than threshold do
3:     for $j = 1, \dots, M$ do
4:         Sample $x_{0,j} \sim p_0(\cdot)$
5:         for $t = 1, \dots, N$ do
6:             Sample policy $\xi_{t,j} \sim p(\cdot|\nu_i)$
7:             Sample state $x_{t+1,j} \sim p(\cdot|x_{t,j}, \pi(x_{t,j}; \xi_{t,j}))$
8:     Compute a new policy $\nu_{i+1}$ using observed step-wise
       costs $\{-r(x_{1,1}, u_{1,1}), \dots, -r(x_{N,M}, u_{N,M})\}$
9:     Set $i = i + 1$
```

where $x_0 \sim p_0(\cdot), u_t = \pi(x_t;\xi_t), x_{t+1} \sim p(\cdot|x_t,u_t)$, and $\xi_t \sim \rho(\cdot|\nu)$.

Using these definitions, (1) can be reformulated in terms of an expectation of step-wise advantages over the stationary state distribution. The expected return of a policy distribution $\rho(\cdot|\nu)$ can be expressed in terms of its advantage over a policy distribution $\rho(\cdot|\nu_0)$ as

$$R(\nu) = R(\nu_0) + \mathbb{E}_{(x_0,u_0,\xi_0\dots)\sim p(\cdot|\nu)}\left[\sum_{t=0}^{N} A^{\nu_0}(x_t,u_t)\right], \quad (2)$$

where $x_0 \sim p_0(\cdot), u_t = \pi(x_t;\xi_t), x_{t+1} \sim p(\cdot|x_t,u_t)$, $\xi_t \sim \rho(\cdot|\nu)$ (see Theorem 2 in Appendix, adapted from [2]).

Thus, maximizing $R(\nu)$ is equivalent to maximizing the expected advantage from policy ν_0 under the trajectory distribution induced by policy ν. Let $p(x|\xi)$ be the unnormalized state visitation distribution for a given fixed policy $p(x|\xi) \triangleq \sum_{t=0}^{N} p(x_t = x|\xi)$. Then, we can re-write the last term in (2) as an expectation over states instead of a sum over time-steps as

$$\begin{aligned}
&\mathbb{E}_{(x_0,u_0,\xi_0\dots)\sim p(\cdot|\nu)}\left[\sum_{t=0}^{N} A^{\nu_0}(x_t,u_t)\right] \\
&= \int \sum_{t=0}^{N}\left[\int p(x_t = x|\xi) A^{\nu_0}(x,u)dx\right]\rho(\xi|\nu)d\xi \\
&= \int\int \sum_{t=0}^{N}\left[p(x_t = x|\xi)A^{\nu_0}(x,u)\right]\rho(\xi|\nu)dx\,d\xi \\
&= \int\int p(x|\xi)A^{\nu_0}(x,u)\rho(\xi|\nu)dx\,d\xi \\
&= \mathbb{E}_{x,\xi\sim p(x|\xi)\rho(\xi|\nu)}\left[A^{\nu_0}(x,u)\right].
\end{aligned}$$

Now, one can see that maximizing $\mathbb{E}_{x,\xi\sim p(x|\xi)\rho(\xi|\nu)}\left[A^{\nu_0}(x,u)\right]$ is equivalent to maximizing the expected reward under policy distribution ν.

Finally, we can then further reformulate the objective with respect to a different density $\rho(\xi|\nu_0)$ as:

$$\begin{aligned}
&\mathbb{E}_{x,\xi\sim p(x|\xi)\rho(\xi|\nu)}\left[A^{\nu_0}(x,u)\right] \\
&= \mathbb{E}_{x,\xi\sim p(x|\xi)\rho(\xi|\nu_0)}\left[A^{\nu_0}(x,u)\frac{\rho(\xi|\nu)}{\rho(\xi|\nu_0)}\right],
\end{aligned}$$

which now includes a likelihood ratio between ν_0 and ν. This formulation will be employed to develop an algorithm using data sampled from the current policy ν_0 to estimate the expected advantage (and thus reward) of an untested policy ν. AC-PROPS follows this strategy and is described next.

III. Actor-Critic PAC Robust Policy Search (AC-PROPS)

The goal of AC-PROPS is to compute a new policy ν_{i+1} with minimal expected cost using estimates of step-wise advantages $\widehat{A}^{\nu_i}(x_t,u_t)$ corresponding to policy parameters ξ_t sampled from ν_i, where i denotes the iteration index. That is, it seeks to solve the following program

$$\nu_{i+1} = \arg\min_{\nu} \mathbb{E}_{x,\xi\sim p(x|\xi)\rho(\xi|\nu_0)}\left[-A^{\nu_0}(x,u)\frac{\rho(\xi|\nu)}{\rho(\xi|\nu_0)}\right].$$

Computing the expectation exactly is intractable. It can, however, be bound from above using a relation developed in [21]. The leads to a program of the following form

$$\nu_{i+1} = \min_{\nu,\alpha} \hat{\mathcal{J}}_\alpha(\nu) + \alpha d(\nu,\nu_0) + \phi(\alpha,K,\delta), \quad (3)$$

where $\hat{\mathcal{J}}_\alpha$ is a robust empirical estimate of $\mathbb{E}_{x,\xi\sim p(x|\xi)\rho(\xi|\nu_0)}\left[-A^{\nu_0}(x,u)\frac{\rho(\xi|\nu)}{\rho(\xi|\nu_0)}\right]$, $d(\cdot,\cdot)$ denotes a distance between policy distributions, K is the number of samples, and ϕ is a concentration-of-measure term which reflects the discrepancy between the empirical advantage $\hat{\mathcal{J}}_\alpha$ and the true mean advantage. The expression in (3) (denoted $\mathcal{J}^+$) is in fact a high-confidence bound on the expected advantage, i.e. with probability $1-\delta$ it hold that $\mathbb{E}_{x,\xi\sim p(x|\xi)\rho(\xi|\nu_0)}\left[-A^{\nu_0}(x,u)\frac{\rho(\xi|\nu)}{\rho(\xi|\nu_0)}\right] \le \mathcal{J}^+$. This is an alternative, step-based formulation of the PROPS update law, which is explained in more detail in §III-A (see [19] for the original formulation).

The advantages that are used in this formulation are unknown and, thus, need to be estimated. The specifics of the advantage estimation and policy sampling procedures are described in §III-B and §III-C, respectively. Algorithm 3 describes the entire AC-PROPS procedure.

Algorithm 3 AC-PROPS

```
1: Initialize hyper-distribution $\nu_0$, $i \leftarrow 0$
2: while Bound on expected cost greater than threshold do
3:     for $j = 1,\dots,M$ do
4:         Sample $x_{0,j} \sim p_0(\cdot)$
5:         for $t = 1,\dots,N$ do
6:             Sample policy $\xi_{t,j}$ according to (6)
7:             Sample state $x_{t+1,j} \sim p(\cdot|x_{t,j},\pi(x_{t,j};\xi_{t,j}))$
8:             Compute $\widehat{A}^{\nu_i}(x_{t,j},u_{t,j})$ using GAE
9:     Compute $\nu_{i+1}$ using $\{(\xi_{i,j},\widehat{A}^{\nu_i}(x_{t,j},u_{t,j}))\}$ in (3)
10:    Update value function parameters $\phi$ using (5).
11:    Set $i = i + 1$
```

A. Upper Confidence Bound

This section defines the terms that compose the upper confidence bound (3). Define the expectation over negative advantages as

$$J(\nu) \triangleq \mathbb{E}_{x,\xi\sim p(x|\xi)\rho(\xi|\nu_0)}\left[-A^{\nu_0}(x,u)\frac{\rho(\xi|\nu)}{\rho(\xi|\nu_0)}\right].$$

$\mathcal{J}(\nu)$ can be approximated empirically using samples $\xi_{t,j} \sim \rho(\xi|\nu_0)$ and $x_{t,j} \sim p(x|\xi_{t,j})$, i.e. $\mathcal{J}(\nu) \approx \frac{1}{MN}\sum_{j=1}^{M}\sum_{t=0}^{N}\left[-A^{\nu_0}(x_{t,j},u_{t,j})\frac{\rho(\xi_{t,j}|\nu)}{\rho(\xi_{t,j}|\nu_0)}\right]$, where j is an index over trajectory samples. As noted in [22], the change-of-measure likelihood ratio $\frac{\rho(\xi_{t,j}|\nu)}{\rho(\xi_{t,j}|\nu_0)}$ can be unbounded, so a standard Hoeffding or Bernstein bound becomes impractical to apply. The bound derived in [21] employs a recent robust estimation technique [23] to deal with the unboundedness of the policy adaptation. Instead of estimating the expectation $m = \mathbb{E}[X]$ of a random variable $X \in [0,\infty)$ using its empirical mean $\widehat{m} = \frac{1}{M}\sum_{j=1}^{M} X_j$, a more robust estimate can be obtained by truncating its higher moments, i.e. using $\widehat{m}_\alpha \triangleq \frac{1}{\alpha M}\sum_{j=1}^{M}\psi(\alpha X_j)$ for some $\alpha > 0$ where $\psi(x) = \log(1+x+\frac{1}{2}x^2)$. As a result, as long as x has finite variance it is possible to obtain practical bounds even if x itself is unbounded.

To obtain tight bounds, it is useful to use samples created during previous iterations of AC-PROPS, say from L previous policies $\nu_0, \nu_1, \ldots, \nu_{L-1}$ from iterations $i = 0, \ldots, L-1$. Let $z = (x, u, \xi)$ and define $\ell_i(z, \nu) \triangleq -A^{\nu_i}(x, u)\frac{\rho(\xi|\nu)}{\rho(\xi|\nu_i)}$. The expected cost based on multiple iterations can now be expressed as

$$\mathcal{J}(\nu) \equiv \frac{1}{L}\sum_{i=0}^{L-1}\mathbb{E}_{z\sim p(\cdot|\nu_i)}\ell_i(z, \nu).$$

This, again, can be approximated by the empirical mean $\mathcal{J}(\nu) \approx \frac{1}{NML}\sum_{i=0}^{L-1}\sum_{j=1}^{M}\sum_{t=0}^{N}[\ell_i(z_{tij}, \nu)]$. The more robust estimate is then given by

$$\widehat{\mathcal{J}}_\alpha(\nu) \triangleq \frac{1}{\alpha NLM}\sum_{i=0}^{L-1}\sum_{j=1}^{M}\sum_{t=1}^{N}\psi\left(\alpha\ell_i(z_{tij}, \nu)\right).$$

The main result obtained in [21] can now be stated as follows:

Theorem 1: With probability $1 - \delta$ the expected cost of executing a stochastic policy with parameters $\xi \sim \rho(\cdot|\nu)$ is bounded according to

$$\mathcal{J}(\nu) \le \inf_{\alpha>0}\left\{\widehat{\mathcal{J}}_\alpha(\nu) + \frac{\alpha}{2L}\sum_{i=0}^{L-1}b_i^2 e^{D_2(\rho(\cdot|\nu)||\rho(\cdot|\nu_i))} + \frac{1}{\alpha LM}\log\frac{1}{\delta}\right\}, \tag{4}$$

computed after L iterations, with NM samples $z_{ti1}, \ldots, z_{tiM} \sim p(\cdot|\nu_i)$ obtained at iterations $i = 0, \ldots, L-1$, where $D_\eta(p||q)$ denotes the η-order Renyii divergence between p and q. The constants b_i are such that $0 \le -A^{\nu_i}(x_t, u_t) \le b_i$ at each iteration. Note that AC-PROPS requires all negative advantages to be greater than 0, so an offset equal to the smallest sampled advantage estimate is subtracted from all advantage value to ensure that this property holds.

B. Generalized Advantage Estimation

AC-PROPS requires the step-wise advantages $A^\nu(x_t, u_t)$ in its update law, but they are unknown. One solution is to model V^ν using a parameterized function approximator $V(x;\phi)$ (e.g. a neural network) and to estimate the advantages using one-step rewards as $\widehat{A}^\nu(x_t, u_t) = r(x_t, u_t) + \widehat{V}(x_{t+1};\phi) - \widehat{V}(x_t;\phi)$. However, estimation error in $V(x, \phi)$ can lead to excessive bias in the advantage estimator $\widehat{A}^\nu(x_t, u_t)$ which will result in an unstable policy update. On the other hand, we could reduce this bias in the estimator by using an N-step rollout of the rewards as $\widehat{A}^\nu(x_t, u_t) = \sum_{k=0}^{N-t} r(x_{t+k}, u_{t+k}) - \widehat{V}(x_t;\phi)$, but the sum over rewards leads to excessively high variance. Generalized Advantage Estimation (GAE) introduces an advantage estimator which has parameters for tuning this bias-variance trade-off [20]. We use GAE in this work whenever computing sample estimates $\widehat{A}^\nu(x_t, u_t)$. For such an estimator, AC-PROPS needs an approximate representation of the value function. The procedure for learning the value function follows.

1) Value Function Learning: After each policy update, AC-PROPS uses the data generated from the most recent iteration to update the value function parameters ϕ using a trust-region method (as in [20]). The trust region helps to avoid overfitting to the most recent batch of data. We first compute sample estimates of the value function as $\widehat{V}_t = \sum_{k=0}^{N-t}\gamma^k r(x_{t+k}, u_{t+k})$, where $\gamma \in [0, 1]$ is a tunable discount factor. Next, let $\sigma^2 = \frac{1}{M}\sum_{i=0}^{M}\|V(x_i;\phi) - \widehat{V}_i\|^2$. Then, the value function optimization can be formulated as

$$\begin{aligned}
&\underset{\phi}{\text{minimize}} && \sum_{i=0}^{M}\|V(x_i;\phi) - \widehat{V}_i\|^2\\
&\text{subject to} && \frac{1}{M}\sum_{i=0}^{M}\frac{\|V(x_i;\phi) - V(x_i;\phi_{old})\|^2}{2\sigma^2} < \epsilon,
\end{aligned} \tag{5}$$

where ϕ_{old} represents the value function parameters from the previous iteration and ϵ defines the size of the trust region. This objective is optimized using the conjugate gradient algorithm. We refer readers to [20] for more details.

C. Generalized Policy Exploration

AC-PROPS uses generalized policy exploration so that it can trade-off between the advantages of step-based and episode-based exploration strategies. Generalized policy exploration introduces a parameter $\beta \in [0, 1]$ that determines how smoothly the policy parameters can change across time-steps. In this work, we choose the policy distribution to be Gaussian. That is, $\xi \sim \mathcal{N}(\mu_\nu, \Sigma_\nu)$, where μ_ν and Σ_ν are the mean and covariance encoded by ν. Given this form, policy sampling can be smoothed across time-steps as

$$\xi_{t+1} = \beta\tilde{\xi} + (1-\beta)\xi_t,\ \tilde{\xi} \sim \mathcal{N}\left(\mu_\nu, \left(\frac{2}{\beta} - 1\right)\Sigma_\nu\right), \tag{6}$$

where $\beta = 1$ corresponds to a purely step-based exploration and $\beta = 0$ corresponds to episode-based exploration. The covariance of the policy distribution is modified by a function of β to preserve detailed balance as explained in [9].

D. Implementation Details

The AC-PROPS update equation (3) is implemented in Tensorflow so that gradient computation can be parallelized on a GPU, which is crucial for efficiently optimizing over large numbers of parameters. AC-PROPS uses a truncated Newton (also known as Newton Conjugate-Gradient) method to minimize (3) using the computed gradients. See Section 6.2 of [24] for more details on truncated Newton methods.

The trajectory sample generation is parallelized across multiple workers with different random seeds to take full advantage of the power offered by multi-core CPUs.

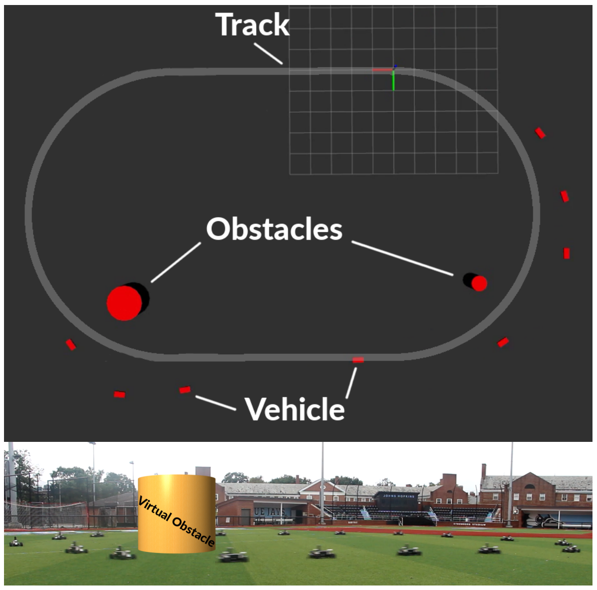

Fig. 1. A simulated mobile robot (top) and a real robot (bottom) using an optimized control policy to follow a 22 m × 14 m oval track at 6.5 m/s while avoiding randomly sampled virtual obstacles in the vehicle's path. This work only trains a policy in simulation, but recent work suggests the policy can be safely transferred directly to a real robot [25]

IV. Training Obstacle Avoidance Policy for an Agile Mobile Robot

We leverage a deep stochastic dynamic model of a 1/5 scale off-road UGV and use it to train a control policy in simulation using Algorithm 3. The goal is to track an oval trajectory at a speed of 6.5 m/s while avoiding randomly generated virtual obstacles in the path of the vehicle. Although this work only trains a policy in simulation, recent work suggests the policy can be safely transferred directly to a real robot [25]. Figure 1 illustrates the task.

A. Deep Stochastic UGV Model

The UGV that we model is a heavily modified 1/5 scale Redcat Racing Rampage XB-E. We collected about 30 minutes of dynamics data, including position, orientation, wheel velocity, steering angle, and steering and velocity commands, while manually driving the car on an astroturf field and took care to make sure the data distribution evenly spanned the state-action space of the vehicle expected for the task. We built a stochastic dynamics model using the technique described in [25]. The model inputs include the car orientation θ, body-x velocity v, steering angle δ_s, velocity command v_c, and commanded steering angle δ_c. The model outputs $\dot{x}$, with $x = (p, \theta, v, \delta_s)$, where $p = (p_x, p_y) \in \mathbb{R}^2$ is the position of the vehicle.

B. Control Policy

Using the dynamics model discussed in the previous section, we optimize control policies for the UGV in simulation using both PROPS and AC-PROPS. We use a policy based on feedback controllers for achieving desired lateral offset, speed, and obstacle avoidance, with relatively few learnable parameters. The policy formulation is described in more detail in [25].

C. Policy Optimization

Navigation Cost: The policy search cost function that we attempt to minimize takes the form

$$J(\tau) = \sum_{t=0}^{t_f/dt} [Ra_t^2 + Q_r e_{r\,t}^2 + Q_v(v_t/v_{goal} - 1)^2 + |v_t| O(d_t)]dt,$$

where $(\cdot)_t$ indicates the state at a discrete time index t, a is the vehicle acceleration, e_r is the lateral offset of the vehicle relative to the track, v_{goal} is the goal velocity, $R, Q_r, Q_v > 0$ are tuning weights, dt is the time step, and $O(d)$ is a cost that encourages obstacle avoidance and is defined as

$$O(d) = \begin{cases} O(o_{low}) + C_{low}(o_{low} - d)^2, & d < o_{low} \\ C_{high}(o_{high} - d)^2, & o_{low} < d < o_{high} \\ 0, & \text{otherwise,} \end{cases}$$

where d is the distance from the car to the closest obstacle with distance measured from the edge of the car to the edge of the obstacle. The variables o_{high} and o_{low} are distance thresholds that determine when the car incurs a small penalty or a large penalty for being close to the obstacle, respectively. $O(d)$ is multiplied by the car velocity to encourage the vehicle to stay away from the obstacle rather than allowing it to quickly speed close by the obstacle without incurring a large penalty.

For our experiments we set $v_{goal} = 6.5$ m/s, $t_f = 7$ s, $dt = 0.02$ s, $R = 10^{-3}, Q_r = 0.25, Q_v = 4, C_{low} = 800, C_{high} = 80, o_{low} = 0.5$ m, and $o_{high} = 1.0$ m.

Stochastic Policies: The surrogate policy $p(\cdot|\nu)$ is a Gaussian with a diagonal covariance matrix. We initialize the surrogate policy to have a standard deviation of 2 in all dimensions.

Environment: The robot attempts to follow a 22 m × 14 m oval track at a goal velocity of 6.5 m/s. At the start of each episode, an obstacle is randomly generated 8 m in front of the vehicle with a track offset uniformly distributed in the range $[-4$ m$, 4$ m$]$ and a radius uniformly distributed in the range $[0.3$ m$, 1.0$ m$]$. An episode terminates either when the robot has hit an obstacle or when t_f seconds have elapsed. When the episode terminates, the obstacle is cleared and a new obstacle is generated at the beginning of the next episode. The robot state at the end of one episode is the same as its initial state at the beginning of the next episode, i.e. the robot remains in motion from one episode to the next.

Policy Search: We perform ten policy search trials each for PROPS and AC-PROPS. We train each policy for 400 iterations, collecting 50 episodes (i.e. trajectory roll-outs) in each iteration and using a sliding window of 20 batches for each policy update. For both algorithms, we set the bound confidence $1 - \delta = .95$ indicating that the computed performance bound should hold with 95% probability. For AC-PROPS, we pre-train the value function using 500 trajectory samples before policy updates begin and we use a value function update constraint of $\epsilon = 0.01$. The value function

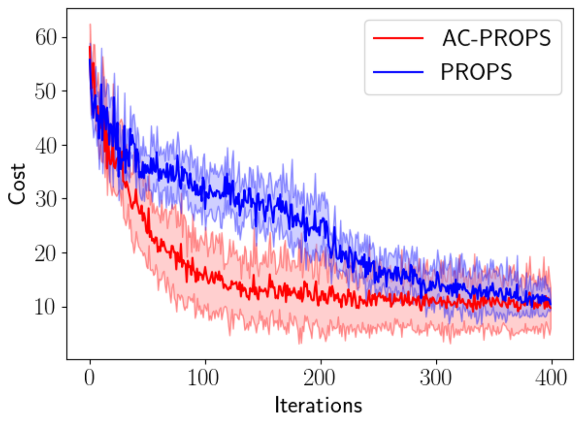

Fig. 2. Comparison of convergence of policy costs of PROPS (red) and AC-PROPS (blue). Shaded regions show standard deviation over 10 trials.

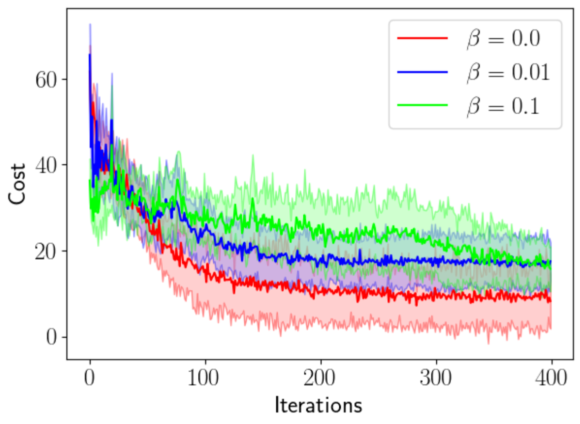

Fig. 3. Comparison of convergence of policy costs for AC-PROPS over different values of policy smoothing parameter β. Shaded regions show standard deviation over 10 trials.

approximator is a neural network with three hidden layers of sizes 32, 32, and 16. Each layer uses a Rectified Linear Unit (ReLU) activation function. For GAE used in AC-PROPS, we set $\lambda = 0.99$ and $\gamma = 0.999$. Furthermore, we sample policy parameters only at the beginning of each episode for both methods (i.e. $\beta = 0$).

D. Results and Discussion

Figure 2 compares the learning curves for the navigation policies trained with PROPS and AC-PROPS over ten trials. AC-PROPS converges at a faster rate than PROPS, suggesting that the reduced variance of the advantage estimates relative to that of the total trajectory cost used by PROPS leads to a more stable update strategy. However, note that the variance of the learning curve across trials is larger for AC-PROPS. The bias induced by by the advantage estimator due to the estimation error of the learned value function may result in premature convergence of the policy distribution.

It is important to note that the bias introduced by the advantage estimator is not properly accounted for in the performance bound used in the update law. Thus, the computed upper confidence bound based on estimated advantages may be violated more often than expected. This is not a major issue because (1) the data demonstrate that it still results in a reliable policy update and (2) a trajectory-based confidence bound can still be computed as in [19] and used to certify the resulting policy.

Figure 3 compares the learning curves of AC-PROPS for different values of β, the tunable parameter for generalized exploration. The data indicate that for this particular task, pure episode-based policy sampling results in the fastest convergence. Since van Hoof et. al. demonstrate that small values of β can lead to gains in sample complexity for learning policies for certain tasks [9], we suspect that the same may be true for AC-PROPS even if it does not help on this particular navigation task. Future work will evaluate the practicality of generalized exploration in AC-PROPS on a wider range of scenarios.

V. Conclusion

This work presented Actor-Critic PAC Robust Policy Search, a policy search algorithm based on minimizing an upper confidence bound on the expected step-wise negative advantage of a policy. We evaluated the algorithm on a vehicle navigation task and showed that it exhibited a faster learning rate than the original PROPS formulation. Since AC-PROPS is capable of step-wise policy sampling, we experimented with generalized policy exploration and found that episode-based sampling worked best on this particular task. Future work will leverage AC-PROPS to train more general policy representations with larger numbers of parameters (like neural networks) and will evaluate AC-PROPS on a wider range of tasks.

VI. Appendix

Theorem 2: The expected reward of a policy distribution $\rho(\cdot|\nu)$ can be expressed in terms of the expectation of advantages A^{ν_0} from some other policy distribution $\rho(\cdot|\nu_0)$ under the trajectory distribution induced by $\rho(\cdot|\nu)$ as

$$R(\nu) = R(\nu_0) + \mathbb{E}_{\tau,\xi\sim p(\cdot|\nu)}\left[\sum_{t=0}^{N} A^{\nu_0}(x_t,u_t)\right].$$

Proof: Note that $A^\nu(x,u)$ can be re-written as

$$A^\nu(x,u) = \mathbb{E}_{x'\sim p(x'|x,u)}[r(x,u) + V^\nu(x') - V^\nu(x)].$$

Then, we have

$$\begin{aligned}
&\mathbb{E}_{\tau,\xi\sim p(\cdot|\nu)}\left[\sum_{t=0}^{N} A^{\nu_0}(x_t,u_t)\right]\\
&= \mathbb{E}_{\tau,\xi\sim p(\cdot|\nu)}\left[\sum_{t=0}^{N}(r(x_t,u_t) + V^{\nu_0}(x_{t+1}) - V^{\nu_0}(x_t))\right]\\
&= \mathbb{E}_{\tau,\xi\sim p(\cdot|\nu)}\left[\sum_{t=0}^{N} r(x_t,u_t) - V^{\nu_0}(x_0)\right]\\
&= -\mathbb{E}_{x_0\sim p_0(x)[V^{\nu_0}(x_0)]} + \mathbb{E}_{\tau,\xi\sim p(\cdot|\nu)}\left[r(x_t,u_t)\right]\\
&= -R(\nu_0) + R(\nu)
\end{aligned}$$

from which Theorem 2 follows. ∎

References

[1] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[2] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1889–1897, 2015.

[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[6] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[7] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *AAAI*, pp. 1607–1612, Atlanta, 2010.

[8] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-exploring policy gradients," *Neural Networks*, vol. 23, no. 4, pp. 551–559, 2010.

[9] H. van Hoof, D. Tanneberg, and J. Peters, "Generalized exploration in policy search," *Machine Learning*, vol. 106, no. 9-10, pp. 1705–1724, 2017.

[10] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, pp. 849–856, 2009.

[11] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *journal of machine learning research*, vol. 11, no. Nov, pp. 3137–3181, 2010.

[12] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[13] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[14] 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*, pp. 745–750, ACM, 2007.

[15] P. WawrzyńSki and A. K. Tanwani, "Autonomous reinforcement learning with experience replay," *Neural Networks*, vol. 41, pp. 156–167, 2013.

[16] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Advances in Neural Information Processing Systems*, pp. 2944–2952, 2015.

[17] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[18] P. Thomas, G. Theocharous, and M. Ghavamzadeh, "High confidence policy improvement," in *International Conference on Machine Learning*, pp. 2380–2388, 2015.

[19] M. Sheckells, G. Garimella, and M. Kobilarov, "Robust policy search with applications to safe vehicle navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2343–2349, IEEE, 2017.

[20] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[21] M. Kobilarov, "Sample complexity bounds for iterative stochastic policy optimization," in *Advances in Neural Information Processing Systems*, pp. 3114–3122, 2015.

[22] C. Cortes, Y. Mansour, and M. Mohri, "Learning bounds for importance weighting," in *Advances in neural information processing systems*, pp. 442–450, 2010.

[23] O. Catoni *et al.*, "Challenging the empirical mean and empirical variance: a deviation study," in *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, vol. 48, pp. 1148–1185, Institut Henri Poincaré, 2012.

[24] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[25] M. Sheckells, G. Garimella, S. Mishra, and M. Kobilarov, "Using data-driven domain randomization to transfer robust control policies to mobile robots," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3224–3230, IEEE, 2019.