Using Data-Driven Domain Randomization to Transfer Robust Control Policies to Mobile Robots

Matthew Sheckells[1], Gowtham Garimella[1], Subhransu Mishra[1], and Marin Kobilarov[1]

***Abstract*— This work develops a technique for using robot motion trajectories to create a high quality stochastic dynamics model that is then leveraged in simulation to train control policies with associated performance guarantees. We demonstrate the idea by collecting dynamics data from a 1/5 scale agile ground vehicle, fitting a stochastic dynamics model, and training a policy in simulation to drive around an oval track at up to 6.5 m/s while avoiding obstacles. We show that the control policy can be transferred back to the real vehicle with little loss in predicted performance. We compare this to an approach that uses a simple analytic car model to train a policy in simulation and show that using a model with stochasticity learned from data leads to higher performance in terms of trajectory tracking accuracy and collision probability. Furthermore, we show empirically that simulation-derived performance guarantees transfer to the actual vehicle when executing a policy optimized using a deep stochastic dynamics model fit to vehicle data.**

I. Introduction

Training control policies for a robotic system in simulation is attractive since data can be generated quickly and the safety of the robot is not a concern. However, policies trained in simulation often do not perform as well when transferred to the real world since the simulator may not completely match reality. Recent research suggests that domain randomization is a promising approach to generating policies that are robust to modelling errors in the simulator [1], [2], [3], [4], [5]. This technique injects noise into the model parameters or dynamics of the simulated system in order to make the generated control policy reliable under a variety of conditions. Many procedures, however, choose the uncertainty of the simulator in an ad-hoc, hand-tuned manner, often with great results [2], [3].

In this work, we propose to learn the uncertainty of the simulator in a data-driven fashion. We first fit a deep stochastic dynamics model to data generated from the real system. Then, we use this model to train a control policy using the stochastic dynamics. We train the policy using PROPS, a reinforcement learning algorithm that searches in policy space and which uses the training data to generate a statistical performance guarantee for future executions of the policy [6]. We show empirically that when the policy is transferred to the real robot that the performance guarantees computed in simulation, such as not exceeding a given probability of collision, still hold. An important consequence of this is that we can approximately quantify and improve the expected performance and safety of a policy in simulation before executing it on the actual physical system. We demonstrate this approach by computing an obstacle avoiding trajectory tracking control policy for an agile Unmanned Ground Vehicle (UGV) travelling up to 6.5 m/s. We show that this approach outperforms a relatively simple procedure which trains a policy in simulation using a standard kinematic car model injected with noise from a hand-tuned distribution.

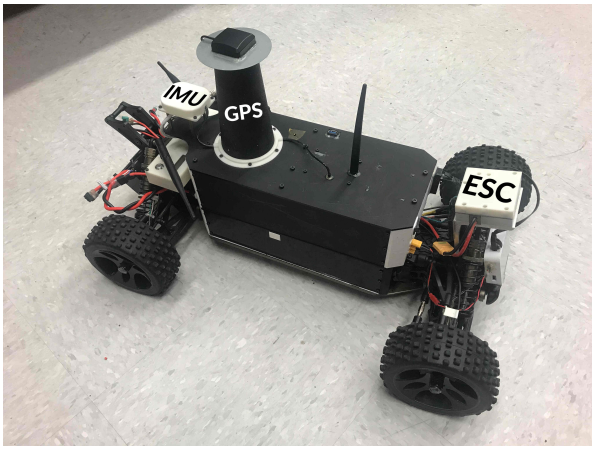

Fig. 1. The JHU all-terrain mobile robot used for navigation experiments.

A. Related Work

This work is closely related to recent research in using domain randomization in reinforcement learning as well as older developments in nonlinear system identification and model-based policy search.

1) Domain Randomization: Using domain randomization to make control policies robust to the transfer from simulation to reality, also known as "sim-to-real", is a relatively new avenue of research that has seen a lot of activity over the last two years. Tobin et al. first used domain randomization in a rendering engine to train an object detector completely in simulation and then use it in the real world to perform robotic grasping [1]. Researchers at OpenAI similarly trained an object pose estimation model in simulation using a randomized renderer while also using reinforcement learning to optimize a control strategy for in-hand manipulation in a randomized simulation. They were able to transfer both the pose estimator and control strategy to a real robotic hand with little performance degradation [2]. Other work trained control policies in simulation with randomized dynamics and successfully deployed the policies on real robots for manipulating objects [4], [7] and quadruped control [3].

[1]M. Sheckells, G. Garimella, S. Mishra, 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|ggarime1|smishra9|marin@jhu.edu

Sadeghi et al. trained a visual servoing policy in simulation while randomizing properties such as camera viewpoint and lighting conditions, showing that the transferred policy generalized to unseen objects in the real world [5].

2) Stochastic Dynamics Modelling: Neural networks are a powerful tool for modelling nonlinear functions, so researchers have used them to model nonlinear system dynamics [8], [9], [10] and stochastic dynamics [11] for the past two decades. Recent methods have used ensembles of deep models to account for both inherent model uncertainty and uncertainty in model parameters induced by the learning process (e.g. the data distribution) [12]. A different approach is to use Gaussian Process (GP) regression [13] or Bayesian Neural Networks (BNN) [14], [15] to model the dynamics and covariance as a continuous function of the state and control input.

3) Model-based Policy Search: This work falls in the realm of model-based policy search since we fit a model of the system using data from the vehicle and leverage that for stochastic policy optimization. There are several related works: PILCO learns a stochastic dynamics model using a GP and explicitly incorporates uncertainty into planning and control while using approximate inference to evaluate the policy and an analytic policy gradient to make policy updates [16], [17]. Deep PILCO maintains the same framework as PILCO but replaces the GP with a BNN in order to scale better to high dimensional observation spaces [18]. Guided policy search fits a series of linear-Gaussian models to local system dynamics, while using a Gaussian Mixture Model to capture a rough global estimate of the dynamics. These models are used to optimize local policies which are then used as target distributions for fitting a global policy [19]. More recent work used deep probabilistic models to perform uncertainty-aware model-predictive control [20], [12] and stochastic policy optimization using Back-Propagation Through Time (BPTT) [21] and Trust Region Policy Optimization (TRPO) [22]. Rajeswaran et al. developed the EPOpt algorithm which uses adversarial training and an ensemble of simulated source domains to train a policy that is robust to different target domains and unmodelled disturbances [23]. The proposed approach is similar to related work in that it finds a maximum likelihood estimate of the target system and then uses it for policy optimization in simulation. The key difference here is that we show empirically that performance guarantees generated by the PROPS policy search algorithm transfer to the real system when using a learned stochastic model.

II. Data-Driven Domain Randomization

Policy transfer from simulation to a real robot often leads to sub-optimal or even unsafe results if the policy is trained in a single, deterministic environment. This is because small differences between the simulated and real environment can lead to drastic differences in policy behavior. Recent research has shown that randomizing some aspects of the environment robustifies the policy and leads to better policy transfer. This technique is called domain randomization. Here, we introduce a procedure for modelling the uncertainty in the real environment in order to inject uncertainty into the simulation. Previous techniques use an ad-hoc procedure to pick which parameters of the simulation to randomize. This usually leads to a tuning procedure in which there is a trade-off between policy robustness and how conservative the policy is, where more noise leads to a more robust yet more conservative policy. In contrast to this, we attempt to learn the dynamics distribution from robot-generated data so that the policy is not overly conservative. Next, we describe the model and how we use it for generating random system trajectories.

A. Stochastic Dynamics Model

The state of the dynamic system is denoted by $x \in \mathbb{R}^n$ with control inputs $u \in \mathbb{R}^m$. We assume that the dynamics take the form of a stochastic ODE

$$\dot{x} = f(x,u) + g(x,u)w,$$

where $w(t)$ is a random variable taking values in $\mathbb{R}^\ell$ sampled from a standard Gaussian $\mathcal{N}(0, I_\ell)$, uncorrelated in time. The functions f and g are the mean and the Cholesky form of the covariance of the dynamics, respectively. More formally:

$$\mathbb{E}[\dot{x}(t)] = f(x(t),u(t)),$$
$$\mathbb{E}[\dot{x}(t)\dot{x}(\tau)^T] = g(x(t),u(t))g(x(t),u(t))^\top \delta(t-\tau),$$

where $\delta(t-\tau)$ is the Dirac delta for given times t and τ. While ℓ can be chosen high enough to capture the noise complexity, we set $\ell = n$ for the rest of this work as it works well in practice without making the model too complex. The dynamics can be approximated by some parameterized model $\hat{f}_\theta$ and $\hat{g}_\gamma$. Next, we describe the loss function for fitting such a model to dynamics data.

B. Model Loss Function

Given trajectory data from a real system, we can find functions $\hat{f}_\theta$ and $\hat{g}_\gamma$ which maximize the likelihood of the data and therefore provide a good probabilistic model of the system.

Assume that we have M samples of the dynamics $X = \{x_1 \ldots, x_M\}, U = \{u_1, \ldots, u_M\}, \dot{X} = \{\dot{x}_1 \ldots, \dot{x}_M\}$. To simplify notation, we denote $\hat{f}_i \triangleq \hat{f}_\theta(x_i, u_i)$ and $\hat{g}_i \triangleq \hat{g}_\gamma(x_i, u_i)$. The likelihood of the data is given by the joint probability density

$$p(\dot{X}|\theta,\gamma,X,U) = \prod_{i=1}^{M} \frac{e^{-\frac{1}{2}(\dot{x}_i-\hat{f}_i)^\top \hat{g}_i^{-1^\top} \hat{g}_i^{-1}(\dot{x}_i-\hat{f}_i)}}{\sqrt{(2\pi)^n |\hat{g}_i \hat{g}_i^\top|}}.$$

As is typical, we can instead maximize the simpler log likelihood

$$\ln p(\dot{X}|\theta,\gamma,X,U) = \sum_{i=1}^{M} -\frac{1}{2}(\dot{x}_i-\hat{f}_i)^\top \hat{g}_i^{-1^\top} \hat{g}_i^{-1}(\dot{x}_i-\hat{f}_i) - \frac{1}{2}\ln|\hat{g}_i\hat{g}_i^\top| - \frac{n}{2}\ln 2\pi.$$

Since g is lower triangular, the last two terms above can be simplified to $-\sum_j \ln \hat{g}_{ijj} - \frac{n}{2}\ln 2\pi$. Furthermore, $\hat{g}_i^{-1}(\dot{x}_i - \hat{f}_i)$ can be solved efficiently using forward substitution since

$\hat{g}_i$ is lower triangular. Thus, we can fit a probabilistic model of the dynamics by minimizing the loss

$$\mathcal{L}(\theta, \gamma) = \sum_{i=1}^{M} \frac{1}{2} (\dot{x}_i - \hat{f}_i)^\top \hat{g}_i^{-1^\top} \hat{g}_i^{-1} (\dot{x}_i - \hat{f}_i) + \sum_j \ln \hat{g}_{ijj}.$$

C. Model Architecture

In this work, we choose $\hat{f}_\theta$ and $\hat{g}_\gamma$ to be simple neural networks. Both consist of an input layer, several fully-connected hidden layers with ReLU activation functions, and an output layer. The $\hat{f}_\theta$ network simply outputs an n-dimensional vector. The $\hat{g}_\gamma$ network outputs $n(n+1)/2$ numbers that are re-shaped into a lower triangular matrix. The diagonal of the matrix is constrained to a positive number by exponentiating the diagonal output from $\hat{g}_\gamma$. We then add a small regularizing constant to the diagonal to ensure invertibility of the matrix. We whiten the data before passing it to the input layer to normalize the data and avoid saturating activation functions. Each hidden layers uses batch normalization also to avoid activation saturation.

D. Sampling Trajectories from Stochastic Model

Starting from an initial state x_0, we can generate a random trajectory sample using the stochastic model with an Euler integration scheme

$$x_{i+1} = x_i + \tilde{\dot{x}}\ dt,$$

where dt is the simulation time step and $\tilde{\dot{x}}$ is sampled from $\mathcal{N}(\hat{f}_\theta(x_i, u_i), \hat{g}_\gamma(x_i, u_i)\hat{g}_\gamma(x_i, u_i)^\top)$. Thus, we can use the learned stochastic model to randomize the environment while performing stochastic policy optimization.

III. Stochastic Policy Optimization

With a good probabilistic model of the dynamics in hand, the goal of policy search is to find an optimal policy for the model. Also of primary importance is providing a formal guarantee on the future performance of the system under the policy. In this section, we review a previously developed policy search method that directly minimizes a bound on future performance, called PAC Robust Policy Search (PROPS) [6], with PAC meaning "probably-approximately-correct".

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 is a cost function encoding the desired behavior, ξ is a vector of decision variables defining the control policy, τ is the system response governed by the density $p(\cdot|\xi)$, and ν parameterizes a surrogate distribution over the decision variables. 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.

PROPS works within the framework of Iterative Stochastic Policy Optimization (ISPO). The goal of ISPO is to generate an optimal control policy which minimizes the cost function $\mathcal{J}(\nu) \triangleq \mathbb{E}_{\tau, \xi \sim p(\cdot|\nu)}[J(\tau)]$. We perform the search directly in the parameter space of the policy and learn a distribution over the policy parameters $\pi(\cdot|\nu)$. ISPO iteratively builds a surrogate stochastic model $\pi(\xi|\nu)$ from which a policy ξ can be sampled. The goal is to then minimize the expected cost $\mathcal{J}(\nu)$ iteratively until convergence. This usually corresponds to $\pi(\xi|\nu)$ shrinking to a tight peak around ξ^* or around several peaks if the distribution is multi-modal. The general framework for solving the problem is described in Algorithm 1.

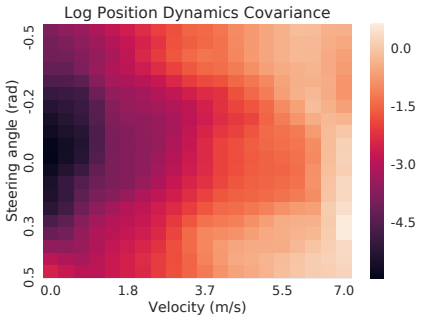

Fig. 2. Log position dynamics covariance (computed as $\log(\sigma^2_{\dot{p}_x} + \sigma^2_{\dot{p}_y})$) histogram over steering angle and velocity for the learned stochastic dynamics. The model learns that $\dot{p}$ is noisier at higher steering angles and velocities. Here, the orientation input to the covariance model is $\theta = 0$, but we see similar results for all $\theta \in [-\pi, \pi]$. Furthermore, to generate this histogram we set $v_c = v, \delta_c = \delta_s$.

Algorithm 1 Iterative Stochastic Policy Optimization (ISPO)

1: Initialize hyper-distribution ν_0, $i \leftarrow 0$
2: **while** Bound on expected cost greater than threshold **do**
3: **for** $j = 1, \ldots, M$ **do**
4: Sample trajectory $(\xi_j, \tau_j) \sim p(\cdot|\nu_i)$
5: Compute a new policy ν_{i+1} using observed costs $\{J(\tau_1), \ldots, J(\tau_M)\}$, set $i = i + 1$

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 (Step 5) corresponds to different policy search algorithms such as Reward-weighted Regression (RwR) [24] or Relative Entropy Policy Search (REPS) [25]. This work uses a recently introduced algorithm called PAC Robust Policy Search (PROPS) for updating the policy based on minimizing an upper confidence bound on its expected future cost. PROPS performs an optimization of the form

$$\min_{\nu, \alpha} \hat{\mathcal{J}}_\alpha(\nu) + \alpha d(\nu, \nu_0) + \phi(\alpha, N, \delta), \tag{1}$$

where $\hat{\mathcal{J}}_\alpha$ is a robust empirical estimate of $\mathcal{J}$, $d(\cdot, \cdot)$ denotes a distance between policy distributions, N is the number of samples, and ϕ is a concentration-of-measure term which reflects the discrepancy between the empirical cost $\hat{\mathcal{J}}_\alpha$ and the true mean cost $\mathcal{J}$. The expression in (1) (denoted $\mathcal{J}^+$) is in fact a high-confidence bound on the expected cost, i.e. with probability $1 - \delta$ it hold that $\mathcal{J} \leq \mathcal{J}^+$. PROPS is explained in more detail in previous work [6].

Fig. 3. The robot using an optimized control policy to follow a 22 m × 14 m oval track at 6.5 m/s while avoiding virtual obstacles.

IV. Training Obstacle Avoidance Policy for an Agile Mobile Robot

Using the procedure outlined in §II, we learn a deep stochastic dynamic model of a 1/5 scale off-road UGV and use it to train a control policy in simulation using the algorithm described in §III. Furthermore, we train another control policy on a simple kinematic car model (with and without hand-tuned noise) using the same policy search algorithm and compare the performance of each policy when evaluated on the real vehicle. The goal of each policy 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. Figure 3 illustrates the task.

A. Robot

Our UGV is a heavily modified 1/5 scale Redcat Racing Rampage XB-E equipped with an onboard Gigabyte BRIX computer with an i7 processor. A LORD Microstrain 3DM-GX4-25 IMU measures inertial data while a u-blox C94-M8P RTK GPS computes global position. A hall effect sensor encoder measures the rotation rate of the drive shaft, which is converted to the body velocity of the vehicle after calibration.

B. Deep Stochastic UGV Model

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 §II, using 3 hidden layers of 64 units for both the f and g models. 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.

The learned mean dynamics of the car behaved similarly to a simple kinematic car model at lower velocities, while the learned covariance model indicated higher noise in $\dot{p}$ at higher velocities and sharper steering angles as illustrated in Figure 2.

C. Simple Stochastic UGV Model

The UGV can be modelled using a kinematic bicycle model, where the state includes the position $p = (p_x, p_y) \in \mathbb{R}^2$ and orientation θ, and the control inputs include the body-x velocity v and the steering angle δ_s. The state is then $x = (p, \theta)$ with dynamics

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} R(\theta) & 0_{2\times 1} \end{bmatrix} \begin{bmatrix} v + w_x \\ w_y \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ v(\frac{\tan \delta_s}{L} + w_\theta) \end{bmatrix}, \tag{2}$$

where $R(\theta) \in SO(2)$, w_x, w_y, w_θ are zero-mean normally distributed random variables with respective covariances $\sigma_x, \sigma_y, \sigma_\theta$, and L is the UGV wheelbase. The model also incorporates a tunable control delay. We train policies both for the model without noise (i.e. $\sigma_x, \sigma_y, \sigma_\theta = 0$) and with noise of various amplitudes, while showing results for the policy that performed best on the real UGV.

D. Control Policy

Using the models discussed in the previous section, we optimize control policies for the UGV in simulation using PROPS. We use a policy based on feedback controllers for achieving desired lateral offset, speed, and obstacle avoidance, with relatively few learnable parameters. Next, we discuss the representation of the car that we use for the policy and then explain the details of the policy itself.

1) Curvilinear Car Dynamics: During motion planning, it is useful to express the state of an autonomous vehicle with respect to some curvilinear coordinate system. For example, the reference curvature of the coordinate system can follow the center line of a road or path. We define the state of the vehicle as $x_{curv} = (s, e_r, e_\theta, v, a, \delta_s) \in \mathbb{R}^6$, where s is the arc length along the reference path, e_r is the lateral offset from the path, e_θ is the angular offset from the path tangent at s, and v is the forward body-velocity, and a is the forward body-acceleration. The control inputs to the system consist of the jerk $u_1 \in \mathbb{R}$ and steering angle rate $u_2 \in \mathbb{R}$. Typical bicycle dynamics expressed using path coordinates are given as

$$\begin{bmatrix} \dot{s} \\ \dot{e_r} \\ \dot{e_\theta} \\ \dot{v} \\ \dot{a} \\ \dot{\delta_s} \end{bmatrix} = \begin{bmatrix} \frac{v \cos e_\theta}{1-\kappa(s)e_r} \\ v \sin e_\theta \\ v\frac{\tan \delta_s}{L} - \kappa(s)\dot{s} \\ a \\ u_1 \\ u_2 \end{bmatrix}, \tag{3}$$

where $\kappa(s)$ is the curvature of the path at s and L is the wheelbase of the vehicle [26].

2) Controller Design: While there exists previous work which can perform obstacle avoidance while guiding the system to a static goal (e.g. [27], [28], [29]), to the author's knowledge no such control design methodology exists to simultaneously track a trajectory and avoid obstacles while providing convergence guarantees.

Here, we design a Lyapunov stable controller that achieves a desired track offset and velocity in a de-coupled manner. A higher level planner commands a track offset to the lateral controller that is computed to avoid any detected obstacles.

Lateral controller: The lateral controller guides the vehicle to the center of the track lane, i.e. $e_r = 0$. We derive such a control law using backstepping on the lateral offset. We start with the Lyapunov candidate $V_0 = \frac{1}{2}k_{r_p}e_r^2 + \frac{1}{2}\dot{e}_r^2$, which has the time derivative

$$\begin{aligned}\dot{V}_0 &= \dot{e}_r[k_{r_p}e_r + \ddot{e}_r] \\ &= \dot{e}_r\left[k_{r_p}e_r + a\sin e_\theta + v\cos e_\theta\left(\dot{\theta} - v\frac{\kappa(s)\cos e_\theta}{1-\kappa(s)e_r}\right)\right]\end{aligned}$$

with $\dot{\theta} \triangleq v\frac{\tan\delta_s}{L}$ and $k_{r_p} > 0$. Setting $\dot{\theta}$ to the desired value

$$\dot{\theta}_d \triangleq v\frac{\kappa(s)\cos e_\theta}{1-\kappa(s)e_r} + \frac{1}{v\cos e_\theta}(-k_{r_p}e_r - a\sin e_\theta - k_{r_d}\dot{e}_r)$$

with $k_{r_d} > 0$ makes $\dot{V}_0 \leq 0$ and renders the system stable, but we cannot directly set $\dot{\theta}$ to $\dot{\theta}_d$. Thus, we create a second storage function V_1 which drives the error $z_\theta \triangleq \dot{\theta} - \dot{\theta}_d$ to zero

$$V_1 = V_0 + \frac{1}{2}z_\theta^2.$$

Noting that $\dot{V}_0 = \dot{e}_r[v\cos e_\theta(\dot{\theta} - \dot{\theta}_d)] - k_{r_d}\dot{e}_r^2$, we have

$$\begin{aligned}\dot{V}_1 &= z_\theta[\dot{e}_r v\cos e_\theta + \dot{z}_\theta] - k_{r_d}\dot{e}_r^2 \\ &= z_\theta\left[\dot{e}_r v\cos e_\theta + \frac{a\tan\delta_s}{L} + \frac{v\sec^2\delta_s}{L}\dot{\delta}_s - \ddot{\theta}_d\right] - k_{r_d}\dot{e}_r^2.\end{aligned}$$

Setting

$$u_2 = \dot{\delta}_s = \frac{L\cos^2\delta_s}{v}\left[-\frac{a\tan\delta_s}{L} + \ddot{\theta}_d - \dot{e}_r v\cos e_\theta - k_{z_\theta}z_\theta\right]$$

makes $\dot{V}_1 \leq 0$, thereby stabilizing the system. Thus, $\dot{\delta}_s$ gives a control law for tracking a reference line and has three tunable parameters $k_{r_p}, k_{r_d}, k_{z_\theta}$.

Speed Control: The speed control law is a simple PD controller on the forward-jerk u_1 of the vehicle $u_1 = k_{v_p}(v_d - v) - k_{v_d}a$ to achieve a desired speed v_d. The desired speed is set to the goal speed v_{goal} while the vehicle is driving straight; however, it is often important for the vehicle to slow down while turning to avoid obstacles. So, we introduce a tunable lateral acceleration constraint a_{latmax}, which limits v_d to be smaller at larger steering angles, i.e. $v_d \leq v_{max}(\delta_s) \triangleq \sqrt{a_{latmax}L/|\tan\delta_s|}$. If $v_{goal} < v_{max}$, we set $v_d = v_{goal}$.

Obstacle Avoidance: A high level control strategy chooses the desired track offset for the vehicle based on the position of detected obstacles. If an obstacle is detected within some radius of the vehicle, denoted k_{det}, then the desired track offset is shifted by a distance k_{shift} away from the obstacle in the direction that the vehicle is pointing relative to the obstacle. So, if the robot is pointing to the left of the obstacle, then the desired track offset is shifted to the left. If the track itself is already far enough away from the detected obstacle, then no shift occurs. Both k_{det} and k_{shift} are tunable parameters in the navigation system.

Thus, the whole navigation policy has 8 tunable parameters: lateral gains $k_{r_p}, k_{r_d}, k_{z_\theta}$, velocity control parameters $k_{v_p}, k_{v_d}, a_{latmax}$, and obstacle avoidance parameters k_{det}, k_{shift}. These compose the vector $\xi \in \mathbb{R}^8$ (see Sec. III).

E. 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, $R, Q_r, Q_v > 0$ 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 $\times$ 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 $[-4\text{ m}, 4\text{ m}]$ and a radius uniformly distributed in the range $[0.3\text{ m}, 1.0\text{ m}]$. 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 train each policy for 260 iterations using PROPS, collecting 50 episodes (i.e. trajectory roll-outs) in each iteration and using a sliding window of 20 batches for each policy update. For the PROPS algorithm, we set the bound confidence $1 - \delta = .95$ indicating that the computed performance bound should hold with 95% probability. PROPS is not sensitive to this user-selected parameter and has no other tunable parameters.

F. Results and Discussion

Figure 4 shows the convergence of the navigation policy parameters over time for the deep stochastic model and simple noisy car model, respectively. The policy trained on the simple model learns to track the reference much

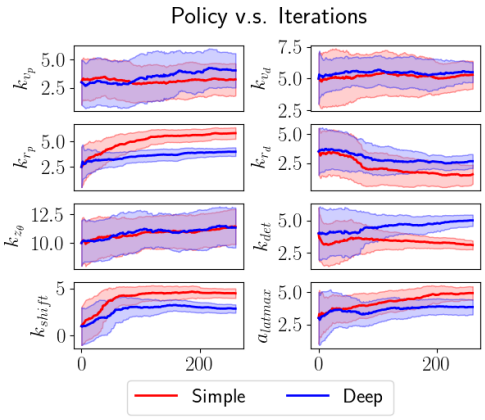

Fig. 4. Comparison of the convergence of policies trained using a simple ad-hoc stochastic car model (red) and deep stochastic model (blue).

more aggressively with a larger k_{r_p} and smaller k_{r_d}. It also computes a larger obstacle avoidance distance k_{shift} and smaller detection radius k_{det} compared to the policy trained on the deep stochastic model.

Table I compares the average absolute track offset $|e_r|$ and average velocity tracking error $|v - v_{goal}|$ of the policy trained on the noiseless simple car model with $\sigma_x = \sigma_y = \sigma_\theta = 0$, the stochastic simple model with $\sigma_x = 1.5, \sigma_y = 0.15, \sigma_\theta = 0.11$ (chosen based on hyper-parameter search using real UGV performance), and the deep stochastic model when executed on the real UGV. The policy trained on the stochastic model outperforms those trained on both the simple models in terms of following the oval track and achieving the goal velocity of 6.5 m/s. Qualitatively, Figure 6 shows a representative trajectory from each policy, where the policy trained on the simple model aggressively overshoots the track while the policy trained on the deep model does not. Note that each policy exhibits some degree of error in following the track due to the fact that the vehicle must move away from the path to avoid obstacles. Furthermore, slowing down to satisfy the learned lateral acceleration constraint leads to some velocity tracking error. The policy trained on dynamics closer to the real vehicle is able to find optimal parameters that minimize these tracking error metrics while still safely avoiding obstacles.

PROPS provides a PAC performance guarantee for each policy based on performance of previously executed policies. These guarantees are only expected to transfer from simulation to the real vehicle if the simulation environment distribution (i.e. the dynamics) matches that of the real vehicle. Figure 5 compares the simulation-derived performance bound to the actual mean performance of each policy when executed on the real vehicle for 200 episodes. The real world cost and collision probability of the policy optimized using the deep stochastic dynamics is upper bounded by the associated guarantees while the policies optimized using a simple dynamic model with user-tuned noise are not. Training in simulation using learned stochastic dynamics provides guarantees that actually transfer to the real world, indicating that it is a powerful tool to estimate the performance of a policy before executing it in the reality. Furthermore, the policy

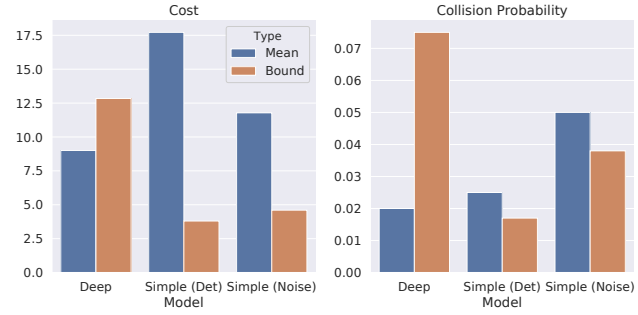

Fig. 5. Mean cost and collision probability for policies executed on the real vehicle compared to simulation-derived performance guarantees. The policy trained using a deep stochastic model both satisfies the performance bounds computed in simulation and outperforms the policies trained on a simple deterministic (Det) and noisy (Noise) kinematic car model.

trained on the deep stochastic model outperforms the policy trained on the simple model in terms of mean trajectory cost while maintaining a similar collision probability.

Model	Avg Offset	Avg Vel Error	Avg Vel	Max Vel
Simple (Det)	1.8 m	2.8 m/s	3.7 m/s	5.8 m/s
Simple (Noise)	1.4 m	2.0 m/s	4.5 m/s	6.2 m/s
Deep Stochastic	1.4 m	1.8 m/s	4.7 m/s	6.5 m/s

TABLE I
POLICY TRACKING PERFORMANCE ON UGV

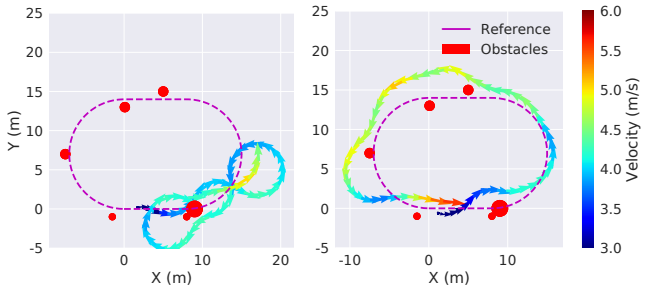

Fig. 6. Sample trajectories executed by the UGV using the policy trained on a simple stochastic car model (left) and deep stochastic model (right). The color of the arrow indicates the velocity of the UGV.

V. CONCLUSION

This work developed a technique for transferring control policies from simulation to reality while preserving performance guarantees. We achieved this by fitting a stochastic dynamic model to data generated from a robotic car and then used the model in simulation to optimize a control policy for a car to navigate an oval track while avoiding obstacles. We transferred the policy back to the real car and experiments showed that it maintained a similar level of performance as in simulation, whereas a policy trained using a simple kinematic car model did not. Furthermore, experiments showed that performance guarantees generated in simulation successfully transferred to the real vehicle.

Future research will apply this policy transfer technique to more general, unstructured control policies for more aggressive off-road driving. We will also look at theoretically bounding the difference in simulated and actual policy performance based on the discrepancy between the learned model and the dynamics training data.

REFERENCES

[1] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, IEEE, 2017.

[2] OpenAI, "Learning dexterous in-hand manipulation," 2018.

[3] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[4] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *arXiv preprint arXiv:1710.06537*, 2017.

[5] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2real viewpoint invariant visual servoing by recurrent control," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4691–4699, 2018.

[6] 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.

[7] R. Antonova, S. Cruciani, C. Smith, and D. Kragic, "Reinforcement learning for pivoting task," *arXiv preprint arXiv:1703.00472*, 2017.

[8] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on neural networks*, vol. 1, no. 1, pp. 4–27, 1990.

[9] S. Chen, S. Billings, and P. Grant, "Non-linear system identification using neural networks," *International journal of control*, vol. 51, no. 6, pp. 1191–1214, 1990.

[10] A. Draeger, S. Engell, and H. Ranke, "Model predictive control using neural networks," *IEEE Control systems*, vol. 15, no. 5, pp. 61–66, 1995.

[11] S. E. Vt and Y. C. Shin, "Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems," *IEEE Transactions on Neural Networks*, vol. 5, no. 4, pp. 594–603, 1994.

[12] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *arXiv preprint arXiv:1805.12114*, 2018.

[13] C. K. Williams and C. E. Rasmussen, "Gaussian processes for regression," in *Advances in neural information processing systems*, pp. 514–520, 1996.

[14] C. M. Bishop, "Bayesian neural networks," *Journal of the Brazilian Computer Society*, vol. 4, no. 1, 1997.

[15] R. M. Neal, "Bayesian training of backpropagation networks by the hybrid monte carlo method," tech. rep., Citeseer, 1992.

[16] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

[17] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," 2011.

[18] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving pilco with bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, ICML*, vol. 4, 2016.

[19] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, pp. 1071–1079, 2014.

[20] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in Neural Information Processing Systems*, pp. 6402–6413, 2017.

[21] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, "Learning and policy search in stochastic dynamical systems with bayesian neural networks," *arXiv preprint arXiv:1605.07127*, 2016.

[22] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *arXiv preprint arXiv:1802.10592*, 2018.

[23] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," *arXiv preprint arXiv:1610.01283*, 2016.

[24] 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.

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

[26] A. De Luca, G. Oriolo, and C. Samson, "Feedback control of a nonholonomic car-like robot," in *Robot motion planning and control*, pp. 171–253, Springer, 1998.

[27] G. Garimella, M. Sheckells, and M. Kobilarov, "A stabilizing gyroscopic obstacle avoidance controller for underactuated systems," in *IEEE 55th Conference on Decision and Control (CDC)*, pp. 5010–5016, IEEE, 2016.

[28] M. T. Wolf and J. W. Burdick, "Artificial potential functions for highway driving with collision avoidance," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3731–3736, IEEE, 2008.

[29] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 341–346, IEEE, 1999.