Autonomous Drifting using Simulation-Aided Reinforcement Learning

Mark Cutler and Jonathan P. How

***Abstract*— We introduce a framework that combines simple and complex continuous state-action simulators with a real-world robot to efficiently find good control policies, while minimizing the number of samples needed from the physical robot. The framework combines the strengths of various simulation levels by first finding optimal policies in a simple model, and then using that solution to initialize a gradient-based learner in a more complex simulation. The policy and transition dynamics from the complex simulation are in turn used to guide the learning in the physical world. A method is developed for transferring information gathered in the physical world back to the learning agent in the simulation. The new information is used to re-evaluate whether the original simulated policy is still optimal given the updated knowledge from the real-world. This reverse transfer is critical to minimizing samples from the physical world. The new framework is demonstrated on a robotic car learning to perform controlled drifting maneuvers. A video of the car's performance can be found at `https://youtu.be/opsmd5yuBF0`.**

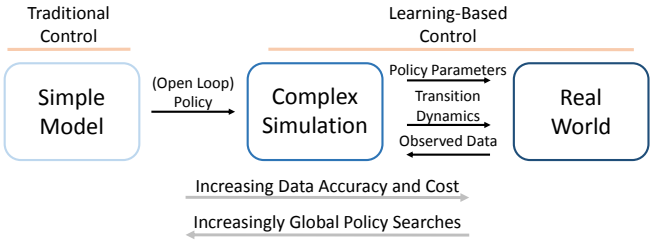


Fig. 1: Many robotic systems utilize models of varying fidelity for design, validation, and control. Simple models are good for finding policies using traditional control, while more complex simulations (including the real world) can use learning-based control methods to refine initial policies. By sending observed data from the real world back to the complex simulation, better policies are found in the real world with fewer samples.

I. Introduction

Simple models are needed in the control design process as many traditional methods such as classical, adaptive, and optimal control rely on models with explicit equations of motion to develop the required controllers. These simple models are often deterministic and use closed-form, expressible equations of motion. Particularly in the case of optimal control, resulting policies are sometimes open-loop and depend entirely on the model used to compute them. These restrictions on simple models used for control design lead to models that sometimes neglect parts of the true system, either because they are non-linear or because they are just not well-enough understood to be expressed in equations.

More complex (and hopefully more accurate) simulations are then used to verify and validate the controllers developed using the simple models [1], [2]. These complex simulations are typically stochastic, either through an attempt to model the stochasticity inherent in the physical system, or as a way to compensate for and capture unmodeled dynamics [3]. Complex simulators can also be "black-box" in the sense that the equations of motion can not be easily written down or that the simulation engine is proprietary or otherwise unavailable (e.g. a commercial video game).

Learning-based methods such as reinforcement learning (RL) can learn control policies directly from interacting with the environment or a simulator, but usually at the expense of requiring many samples before discovering useful information [4]. In robotics, where domains are often high-dimensional and continuous, significant progress has been made recently by utilizing policy search methods that search locally for improvements to the current policy. These methods scale well as the problem dimensionality increases, but, due to the local nature of the search, typically lack any formal guarantees as to global optimality and are subject to local solutions.

As summarized in Figure 1, we introduce a new framework for efficiently learning useful policies on real robots. We use a series of simulators [5]–[7] and model-based policy search RL to bootstrap real-world learning, improve policies already found, and avoid local minima. The framework is tested on a robotic car that learns to autonomously drift, or drive sideways, using very little data. The main contributions of this paper are (1) a way to incorporate principles of optimal control for initializing policy parameters, (2) a derivation of mixing real and simulated data together in order to re-plan in a simulated environment, and (3) simulated and hardware results empirically showing the benefits of using prior information in the learning process.

II. Background Material

Gaussian processes (GPs) [8] are a popular regression tool for modeling observed data while accounting for uncertainty in the predictions. Formally, a GP is a collection of random variables, of which any finite subset are Gaussian distributed. A GP can be thought of as a distribution over possible functions $f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathcal{X}$ such that $f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$, where $m(\boldsymbol{x})$ is the mean function and $k(\boldsymbol{x}, \boldsymbol{x}')$ is the covariance function.

With a fixed mean function and data $\{X, \boldsymbol{y}\}$, where X and $\boldsymbol{y}$ are the input and output data, respectively, the predictive

Mark Cutler and Jonathan How are with the Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA, USA {cutlerm, jhow}@mit.edu

distribution for a deterministic input $\boldsymbol{x}_*$ is

$$
\begin{aligned}
f_* &\sim \mathcal{N}(\mu_*, \Sigma_*) \\
\mu_* &= m(\boldsymbol{x}_*) + k(\boldsymbol{x}_*, X)(K + \sigma_n^2 I)^{-1}(\boldsymbol{y} - m(X)) \\
&= m(\boldsymbol{x}_*) + k(\boldsymbol{x}_*, X)\boldsymbol{\beta} \\
\Sigma_* &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, X)(K + \sigma_n^2 I)^{-1} k(X, \boldsymbol{x}_*)
\end{aligned}
$$

where $\boldsymbol{\beta} = (K + \sigma_n^2 I)^{-1}(\boldsymbol{y} - m(X))$, $K = k(X, X)$, and σ_n^2 is the noise variance parameter.

In this paper the squared error kernel is used for its computational advantages. Thus, the kernel is

$$k(\boldsymbol{x}, \boldsymbol{x}') = \alpha^2 \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \Lambda^{-1}(\boldsymbol{x} - \boldsymbol{x}')),$$

where α^2 is the signal variance and Λ is a diagonal matrix containing the square of the length scales for each input dimension. The hyperparmeters (σ_n^2, α^2, and Λ) are learned via evidence maximization [8].

Probabilistic Inference for Learning COntrol (PILCO) is a recently developed model-based policy search RL algorithm [9]. One of the key advantages of PILCO is a careful handling of uncertainty in the learned model dynamics that helps prevent negative effects of model bias. By explicitly accounting for uncertainty, the algorithm is able to determine where in the state space it can accurately predict policy performance and where more data are needed to be certain of future outcomes.

Learning begins by randomly applying control signals and then using the observed data to build a probabilistic model of the transition dynamics using GPs. This model is then used to update the policy parameters by optimizing over long-term roll-outs of the learned model. Closed-form gradients of the optimization problem are available and so any gradient-based optimization algorithm can be applied. Once converged, the new policy parameters are executed on the system and the process repeats until satisfactory performance is obtained.

III. Policy Search Initialization

Because continuous state-action spaces are prohibitively large for performing global searches, many RL researchers use prior knowledge to initialize learning algorithms. This prior information often comes in the form of an initial policy from an expert demonstration [4]. Many examples of these initializations exist, such as learning helicopter aerobatics [10] and learning to play table tennis [11]. These expert initial policies give local search algorithms a good initial policy from which to start a policy search.

While initial policies from expert demonstrations can be extremely valuable, these demonstrations may not always be available. As an alternative to expert demonstrations, we generate initial policies from simple models using traditional control techniques which are used to initialize the learning in complex simulations. In this paper, Gauss Pseudospectral Optimization Software (GPOPS) [12] is used to compute these initial policies. In practice, any control solution method applied to a model of the system can be used as a policy initialization.

Note that additional learning beyond the resulting policy from GPOPS is only needed when discrepancies exist between reality and the simple model. As any model (especially a simple one) will rarely, if ever, perfectly match the physical system, applying the open-loop optimal control policy from GPOPS to a real robot will, in general, result in sub-optimal or even dangerous behavior [13]. Therefore, these policies are instead treated as an initialization for the learning algorithm in the complex simulation.

The policies learned in the complex simulation are parameterized, with the parameter values found by the learning algorithm. The primary focus in this paper is on policies that are represented using a regularized radial basis function (RBF) network (equivalent to a deterministic Gaussian process), where the centers, the associated targets, and the length-scale values of the RBFs are the parameters to be learned. The RBF network provides a general function approximation scheme for representing various policies.

To transfer the open-loop optimal control policy to the complex simulator, the policy is approximated as a closed-loop controller using an RBF network. Given a fixed budget size on the number of RBFs allowed (typically based on the computational constraints of the learning algorithm in the complex simulation), the centers and targets of the RBF network are found by applying the k-means algorithm [14] to a discretrized representation of the optimal control policy and corresponding optimal states from the simple model. The length scale parameters are then found using evidence maximization, just as the hyperparmeters of a Gaussian process are typically computed. In practice, this is a quick and easy way to train the RBF network using the optimal control policy.

IV. Reverse Transfer Using Gaussian Processes

The framework from Figure 1 is shown in Algorithm 1. The algorithm first uses optimal control methods to find a policy π_s^* in the simple model Σ_s (line 2). This policy is then approximated as an RBF network in line 3. Using these policy parameters, the PILCO algorithm is used to refine and update the initial policy using the more complex simulation Σ_c in line 4. Once converged, the framework passes these new policy parameters and learned transition dynamics from the complex simulation to an instance of PILCO running in the real world Σ_{rw} (line 6). Our previous work [7] describes in detail this forward transfer of policy parameters and transition dynamics. This paper primarily deals with transfer in the other direction as described below.

After real data are observed, they can be passed back to the simulator and used to re-plan (line 7). Finally, in line 8, the algorithm exits when the re-planned policy is not sufficiently different from the policy previously found in the simulator, indicating that the new data provided by the real world will not cause new policy performance or new exploration in the simulator.

In practice, and where sufficient computational power exists, several instances of PILCO in the simulation environment should be run in parallel. One of the main advantages

Algorithm 1 Continuous State-Action Reinforcement Learning using Multi-Direction Information Transfer

Input: Simple, deterministic simulation Σ_s, more complex, stochastic simulator Σ_c, and real world Σ_{rw}
Use optimal control methods to find policy π_s^* in Σ_s
Use k-means to approximate π_s^* as π_c^{init} with an RBF network
Run PILCO in Σ_c with $\pi_c = \pi_c^{init}$ as initial policy
while 1 **do**
    Run PILCO in Σ_{rw} [7]
    Run PILCO in Σ_c to get π_c^{new}, combining GP predictions from Σ_{rw}
    if $\|\pi_c^{new} - \pi_c\| < \epsilon$ **then**
        break
    else
        $\pi_c = \pi_c^{new}$

of using a simulator is the low cost of obtaining samples. Thus, to increase the probability of converging to the global optimum, or of converging to a new local optimum, the simulation can be run in parallel with various random seeds and initial conditions. Then, the best performing or most different policy can be tried on the real system.

To re-plan using real-world data, the algorithm must determine if a given state-action pair is sufficiently well known. With discrete representations of the state-action space, this known-ness check is binary and depends only on whether or not the current state-action pair has been observed a sufficient number of times. With continuous states and actions, the number of times a specific state-action pair has been observed can not be counted as it is unlikely that the exact same data will be observed multiple times. Instead, the algorithm must determine if the current state-action pair is sufficiently "close" to states and actions about which there is little uncertainty. For this purpose, the covariance of the GP's representing the state-action space is used as a known-ness measure.

The covariance is a measure of how uncertain the GP is about the true function at the test input $\boldsymbol{x}_*$, minus the learned noise variance $\sigma_n^2 I$. Thus, Σ_* approaches zero as the true function is correctly modeled (or at least thought to be correctly modeled). The actual predictive covariance of $\boldsymbol{y}^*$ is $\Sigma_* + \sigma_n^2 I$ and so does not approach zero unless the data observations are noise free. Therefore, the value of Σ_* relative to the underlying noise in the system, σ_n^2, can be used as a measure of how much uncertainty remains about the function at a given test point. For the remainder of this section a subscript (rw) is used to indicate variables relating to data from the real world, and a subscript (sim) to denote variables relating to simulated data.

The general approach for using data from the real world when re-planning in the simulation will be to determine, for each sampled state-action pair when doing long-term predictions (when PILCO is sampling its model of the environment to see how well a given set of policy parameters will work), how much information is known about that state-action pair in the real world. When the real-world data are well known, those transition dynamics will be used, otherwise, the algorithm will default to the simulated values. Based on the ratio of Σ_* to $\sigma_n^2 I$, a scalar mixing value $p_{(rw)}$ is determined that dictates what percentage of the real-world data should be incorporated into the current prediction step. Thus, given $p_{(rw)}$, the predictive mean is a linear combination of the predicted mean from the simulated data and the real-world data as

$$\mu_* = \mu_{*_{(sim)}} + p_{(rw)}(\mu_{*_{(rw)}} - \mu_{*_{(sim)}}). \tag{1}$$

Similarly, using standard results of mixing normal distributions, the predictive covariance becomes

$$\begin{aligned}
\beta_{(sim)} &= (\mu_{*_{(sim)}} - \mu_*)(\mu_{*_{(sim)}} - \mu_*)^T + \Sigma_{*_{(sim)}} \\
\beta_{(rw)} &= (\mu_{*_{(rw)}} - \mu_*)(\mu_{*_{(rw)}} - \mu_*)^T + \Sigma_{*_{(rw)}} \\
\Sigma_* &= \beta_{(sim)} + p_{(rw)}(\beta_{(rw)} - \beta_{(sim)}),
\end{aligned} \tag{2}$$

where $\beta_{(sim)}$ and $\beta_{(rw)}$ are defined for notational convenience. The covariance between the input and output is similarly defined as

$$\Sigma_{\boldsymbol{x}_*, f_*} = \Sigma_{\boldsymbol{x}_*, f_{*_{(sim)}}} + p_{(rw)}(\Sigma_{\boldsymbol{x}_*, f_{*_{(rw)}}} - \Sigma_{\boldsymbol{x}_*, f_{*_{(sim)}}}). \tag{3}$$

In PILCO, given a multidimensional problem, each output dimension is modeled using a separate GP. With deterministic inputs, these separate GP's would be independent, but in PILCO uncertain inputs are passed through the GP's to get long-term cost and state predictions. Given these uncertain inputs, the outputs of the GP's covary and are no longer independent. Thus, for a given state-action pair, an individual $p_{(rw)}$ can not be determined for each output dimension, but a single scalar value is found based on the entire output covariance matrix.

For smoothness properties (derivatives of $p_{(rw)}$ are needed in the learning process) a generalized logistic function $f(x)$ defined as

$$\mathrm{f}(x) = \frac{1}{\left(1 + Q\,\mathrm{e}^{B(x - x_0)}\right)^{\frac{1}{Q}}} \tag{4}$$

is chosen to determine $p_{(rw)}$ based on the norm of the current covariance matrix and the norm of the learned noise variance parameters representing the noise in the observed data. The mixing probability $p_{(rw)}$ is therefore defined as

$$p_{(rw)} = \mathrm{f}\left(\frac{\|\Sigma_{*_{(rw)}}\|_F}{\|\left[\sigma_{n_1}^2, \ldots, \sigma_{n_E}^2\right]\|}\right) \tag{5}$$

TABLE I: Parameter values used for the generalized logistic function that dictates the extent to which state-action pairs are known in the real world.

	Q	B	x_0
Value	1.5	400	0.02

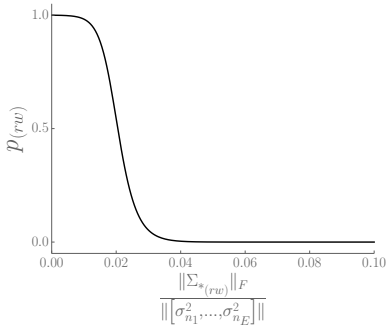


Fig. 2: Generalized logistic function for determining to what extent data should be used from the real world.

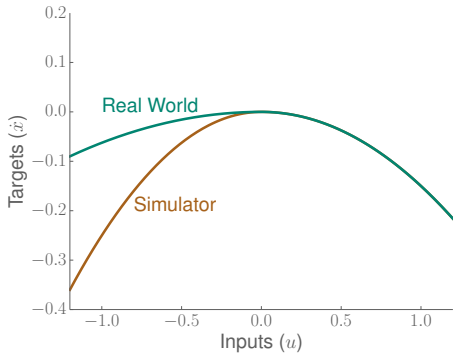


Fig. 3: Dynamics for the 1-D domain. In the simulator, the optimal solution involves negative u while in the real world the optimal solution uses postive u.

where $\|.\|_F$ is the Frobenius norm. A plot of $p_{(rw)}$ versus the covariance ratio is shown in Figure 2, where the parameters of the logistic function are given in Table I. With these parameters the real-world data are only used if there is very little uncertainty about the true function. Otherwise, the simulated data are used. The shape of the logistic is set based on how noisy the real world is believed to be. In all the experiments tried in this paper, the parameters in Table I worked well, without needing to tune them for different domains.

Using equations 1-3, the transfer of information from the real world back to the complex simulation is achieved by computing $p_{(rw)}$ at each state-action pair during long-term state predictions and using the combined output of the mean, covariance, and input-output covariance. To use these equations in PILCO the partial derivatives of these expressions with respect to the input mean and covariance are also needed. A full derivation of the required derivatives is given on the first author's website.[1]

V. RESULTS

Algorithm 1 is implemented in both simulated and real domains to show the applicability of using real-world data to re-plan in a continuous state-action simulator. In the simulated domain, the "real world" is a simulator that differs from the other simulation, demonstrating the performance of the algorithm in controlled environments.

A. Simple 1-D Problem

The first test domain is a single state, single action domain. This toy domain is used to illustrate how the reverse transfer in line 7 of Algorithm 1 is accomplished. The domain consists of a state x and action u. A saturating cost function is $c(x) = 1 - \mathrm{e}^{-\frac{x^2}{2\sigma^2}}$, with the dynamics being

$$\dot{x} = \begin{cases} -au^2, & u \geq 0 \\ -bu^2, & u < 0, \end{cases} \tag{6}$$

where $|u| \leq 1$. In Σ_c and Σ_{rw} $\sigma = 0.25$ and $a = 3$, while in Σ_c $b = 5$ and in Σ_{rw} $b = 1.25$. Figure 3 shows the mean of the dynamics for the simulator and the real world, with a sampling rate of 20 Hz. Each episode starts with $x_0 = 3$ and, according to the cost function, tries to drive x to zero. Both positive and negative values of u result in negative velocities, but with varying magnitudes. In Σ_c, the optimal solution is to start with negative u, while in Σ_{rw} the optimal policy starts with positive u. With this simple domain, the optimal control portion of Algorithm 1 is omitted and π_c^{init} is instead randomly initialized. The parameterized policy in this domain is linear ($u = Ax + b$) rather than using an RBF network as in the other domains.

Figure 4 shows the performance of Algorithm 1 on this simple problem. Each subplot shows the current representation of the GP modeling the transition dynamics (Equation 6). In Figure 4(a), the initial random policy explores the state space. After this initial exploration, the first iteration of the policy update phase finds the optimal policy as seen by the low cost in Figure 4(b) and the new data all on the negative side of u.

The policy and transition dynamics from Σ_c are passed to Σ_{rw} as explained in [7]. Despite the dynamics being different from the simulator, the gradient-based policy update scheme does not explore policy parameters that would involve positive u values since the initial policy is in a local minimum. As seen in Figures 4(c) and 4(d), the policy update phase improves the initial policy, but stays on the negative side of u.

Following Algorithm 1, the data observed from Σ_{rw} are passed back to Σ_c. The GP in Figures 4(e) and 4(f) is a combination of the data from Σ_{rw} where the variance was low (negative u values) and Σ_c elsewhere, shown by the embedded plots of $p_{(rw)}$. With this updated hybrid model of the dynamics, the algorithm converges to a policy that favors positive u first. While this new policy is not optimal in Σ_c, is it optimal *given* the data observed from the real world. This new policy is then passed back to Σ_{rw} whereupon the policy improvement phase largely keeps the same policy, converging to the true optimum in Σ_{rw}.

This simple example is clearly contrived to show the performance of Algorithm 1; however, it illustrates an important

[1] http://markjcutler.com/papers/Cutler16_ICRA_additional.pdf

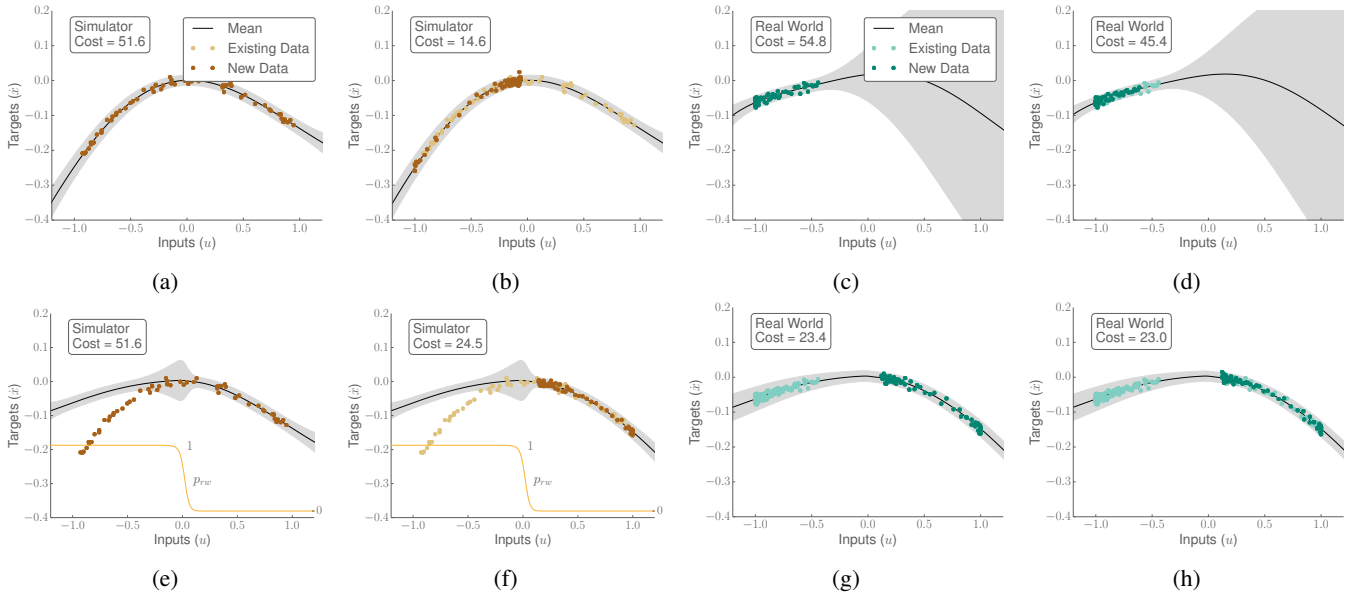


Fig. 4: Algorithm 1 running on a simple 1-D domain. Each subplot shows the current representation of the GP modeling the transition dynamics (Equation 6). In Figures (a) and (b), an initial random policy explores the state space and then quickly converges to the optimal policy, as seen by the low cost in (b) and the new data all in the negative u range. Policy and transition dynamics from Σ_c are passed to Σ_{rw} and used in (c) and (d). The real-world policy is improved, but is stuck in a local minimum based on the initialization. In (e) and (f), model data is passed back to Σ_c which uses data from Σ_{rw} where possible to find a policy involving positive u. This optimal policy (with respect to Σ_{rw}) is passed to Σ_{rw} in (g) and (h) and the algorithm terminates.

point of any gradient-based policy improvement method: convergence to local solutions is sensitive to the initial policy. Algorithm 1 attempts to formally provide a method to account for initial policies that come from inaccurate simulators. Note that the policy found in Figure 4(d) is not necessarily bad—it solves the problem of driving x to zero. The reverse transfer framework provides a way, though, to revisit the simulator and verify, given the updated information from the real world, whether or not a better policy exists for the real world.

B. Drifting Car

The final example in this paper shows Algorithm 1 applied to a small robotic car learning to drift, or drive sideways. The car is outfitted with slick plastic tires, making it quite difficult to control on our hard lab floor.

While numerous other works have demonstrated aggressive control with cars (e.g. [15]–[17]), this work focuses on utilizing prior information in the learning process.

The target task in this section is to control the car in a steady-state drift, where the car maintains a constant sideways velocity. Recent work demonstrated an LQR and backstepping control approach to stabilizing a car under steady-state drift conditions [18]. The controller calculates steady-state cornering conditions which rely on exact knowledge of tire and road forces and conditions. In reality, these values will not be known and inaccuracies in these parameters will yield steady-state errors in the control law. Also, the controller developed is only valid around an equilibrium and

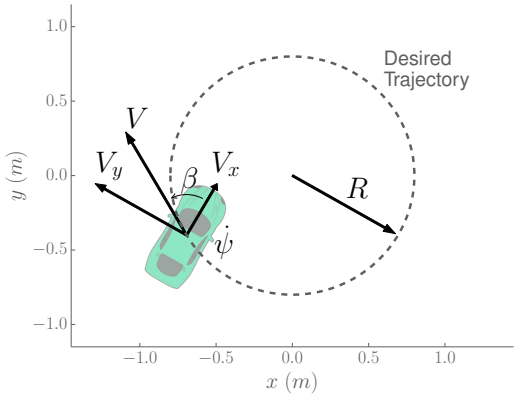
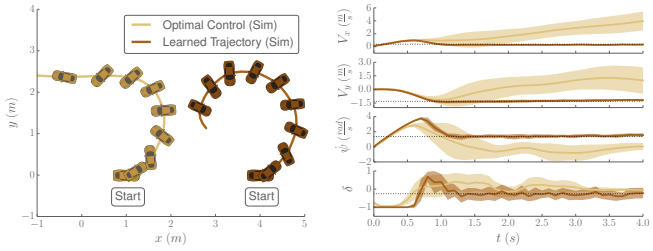


Fig. 5: State variables used for the steady-state drifting domain. The body-frame forward and side velocities are V_x and V_y, respectively, with V being the total velocity and $\dot{\psi}$ the body turn rate. The slip (or drift) angle is β while R is the radius of curvature of the desired trajectory.

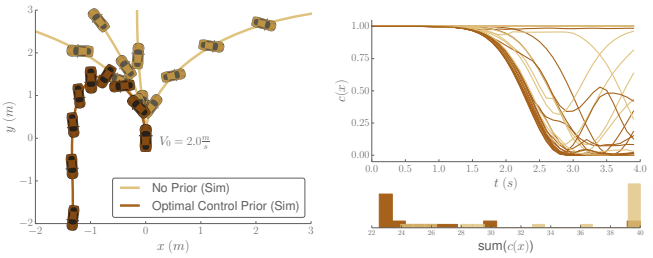
so open-loop commands based on expert driver data are needed to get the vehicle close to the desired state before the controller can be engaged.

The goal of the learning algorithm is to control the vehicle to constant forward, side, and turn rate velocities, resulting in a steady-state drifting motion as indicated by Figure 5. The vehicle state consists of body-frame component velocities V_x and V_y, a turn rate $\dot{\psi}$, and the current wheel speed ω. Turn rate and wheel speed are measured using on-board sensors



(a) (b)

Fig. 6: Policy roll-outs of the simulated car learning to drift. The light colored lines show the performance of the optimal policy from Σ_s when first applied to Σ_c. The dark lines show the improvement the policy search algorithm is able to make over the initial policy. A single representative roll-out of each type is shown in (a), while (b) shows the mean and one standard deviation of five learning runs, where each policy was executed five times. Note that only the velocity states are controlled here, and so the position starting locations in (a) are irrelevant.



(a) (b)

Fig. 7: The effect of a good prior on the performance of PILCO. The light colored lines show the results of applying PILCO to the car turning domain with randomly initialized policy parameters (showing converged policy). In almost every case, the algorithm is unable to find the correct control inputs to turn the car around, instead getting stuck in local optima. Figure (a) shows some representative samples of the two cases while (b) shows the instantaneous and cumulative costs for 20 learning runs for each case. This figure highlights the need for the optimal control prior for complicated domains.

while body-frame velocities are measured using an external motion capture system. Velocity information could be obtained using GPS outdoors. The action δ is the command sent to the on-board servo actuating the wheel steering. The cost function is

$$c(x) = 1 - \mathrm{e}^{\left[-\left((V_x - V_{x_{ss}})^2 + (V_y - V_{y_{ss}})^2 + (\dot{\psi} - \dot{\psi}_{ss})^2\right)/\left(2\sigma^2\right)\right]}, \tag{7}$$

where the subscript ss signifies the desired steady-state values for that variable. This cost function favors steady state drifting where the state variables are kept at a constant value, with a non-zero V_y component. The throttle setting was kept constant as a car can maintain a steady-state drift with just modulating the steering command. The car used a high-current switching voltage regulator to keep the battery voltage from affecting the performance of the learning algorithm.

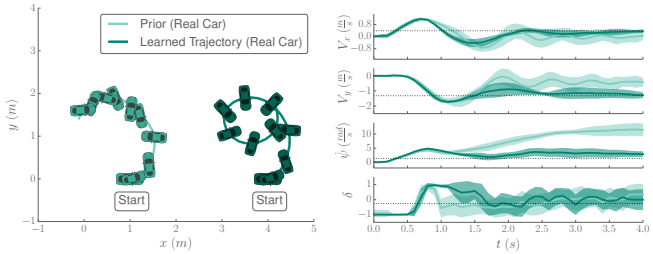
As in the previous domain, Σ_s is a deterministic version of Σ_c. GPOPS is used to solve for π_s^*, the optimal policy in the deterministic simulation. Figure 6 shows the performance of the initial policy from GPOPS in the noisy simulator compared to the policy from Σ_c after several policy improvement steps are made. The car is able to quickly initiate a drifting motion by turning into the drift, briefly counter steering, and then settling on a near constant steering angle.

Figure 7 shows the performance of the PILCO algorithm in Σ_c when the deterministic simulator is not used. In this example, the car starts with a forward velocity of 2 m/s and then tries to turn around in minimum time, maintaining a velocity of 3 m/s in the opposite direction. Here, both the steering and throttle commands are learned. In this domain, the problem is sufficiently complex with enough local minima that PILCO is unable to solve the problem without prior information. The problem was run several times with varying initial random policy parameters, but in almost every case PILCO was unable to find a policy that quickly reversed the car's direction. Figure 7(a) shows some representative samples of the performance of PILCO in Σ_c using an optimal control prior versus a randomly initialized policy. In Figure 7(b), immediate and cumulative costs of 20 learning runs for each case are displayed. As shown by the histogram, without prior information, PILCO was rarely able to solve the problem.

This behavior is perhaps not unexpected. As noted in [4], many, if not most, researchers doing RL in real robotics domains initialize the problem with problem-specific data, often through example demonstrations or hand-crafted policies. In this domain, no one in the lab was expert enough to drive the car in a steady-state drift. It was also not immediately clear how to hand-craft a policy that would lead to this behavior. Thus, the proposed algorithm uses principles of optimal control to take the place of these demonstrations and specific policies.

Figure 8 shows the performance of Algorithm 1 applied to the real car. Figure 9 shows snapshots of the car as it begins a steady-state drift from rest. The car is able to initiate and maintain a steady-state drifting motion with a trajectory radius of approximately 0.8 m. Drifting with other trajectory radii was also successfully accomplished. The learned real-world policy looks very similar to the optimal policy from Σ_c, but with a longer period of counter-steering (see δ in Figure 8 from 0.7–1.5 seconds). This highlights the utility of incorporating simulated data into the learning framework. Rather than having to learn the entire policy from scratch, the learning agent just needed to learn those parts of the policy that were different in the real world when compared to the simulator.

Due to limitations in the size of the testing area, the drifting domain was unable to be solved without prior infor-



(a) (b)

Fig. 8: Policy roll-outs of the real car learning to drift. The light colored lines show the performance of the optimal policy from Σ_c when first applied to Σ_{rw}. The dark lines show the improvement the policy search algorithm is able to make over the initial policy. A single representative roll-out of each type is shown in (a), while (b) shows the mean and one standard deviation of three learning runs, where each policy was executed five times. The real car requires a more counter steering than the simulated car to slow the initial vehicle rotation. Note that only the velocity states are controlled here, and so the position starting locations in (a) are irrelevant.

Fig. 9: Results of Algorithm 1 applied to the real drifting car. Snapshots of the car every 0.25 seconds are shown. The vehicle starts from rest and quickly enters a controlled drifting motion with a 0.8 m radius.

mation from the simulators. Executing random actions (the typical initialization for PILCO) always resulted in the car quickly running into a wall. For this problem, the simulated data are necessary to solve the problem.

A video showing the performance of the drifting car can be seen at `https://youtu.be/opsmd5yuBF0`.

VI. Conclusion

This paper developed a framework for incorporating optimal control solutions into the learning process and for transferring data from real-world experiments back to simulators, allowing the simulator to re-plan using the updated information. The re-planning both validates policies from the real world and possibly leads to better real-world policies by exploring more of the state-action space in the simulator. The reverse transfer was combined together with forward propagation in PILCO [7]. The resulting framework was applied to a simple 1-D simulated domain, showing that the reverse transfer can be necessary to find the optimal solution in the target domain. Finally, the algorithm was applied to a robotic car learning to drift. The task was sufficiently complex that PILCO was unable to solve the problem without the use of prior simulated information.

Acknowledgment

The authors acknowledge Boeing Research & Technology for support of the facility in which the experiments were conducted.

References

[1] E. Gazi, W. D. Seider, and L. H. Ungar, "A non-parametric monte carlo technique for controller verification," *Automatica*, vol. 33, no. 5, pp. 901–906, 1997.

[2] A. M. Jadhav and K. Vadirajacharya, "Performance verification of pid controller in an interconnected power system using particle swarm optimization," *Energy Procedia*, vol. 14, pp. 2075–2080, 2012.

[3] P. Abbeel, M. Quigley, and A. Y. Ng, "Using inaccurate models in reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2006, pp. 1–8.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[5] M. Cutler, T. J. Walsh, and J. P. How, "Reinforcement learning with multi-fidelity simulators," in *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, 2014, pp. 3888–3895.

[6] ——, "Real-world reinforcement learning via multifidelity simulators," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 655–671, June 2015. [Online]. Available: http://markjcutler.com/papers/Cutler15_TRO.pdf

[7] M. Cutler and J. P. How, "Efficient reinforcement learning for robots using informative simulated priors," in *IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA: IEEE, May 2015.

[8] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

[9] M. Deisenroth, D. Fox, and C. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PP, no. 99, 2014.

[10] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[11] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.

[12] M. A. Patterson and A. V. Rao, "Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 1, p. 1, 2014.

[13] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *International Conference on Machine Learning (ICML)*, 1997, pp. 12–20.

[14] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Applied statistics*, pp. 100–108, 1979.

[15] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *American Control Conference (ACC)*, 2007, pp. 2296–2301.

[16] T. K. Lau and Y.-h. Liu, "Stunt driving via policy search," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 4699–4704.

[17] S. L. N. Keivan and G. Sibley, "A holistic framework for planning, real-time control and model learning for high-speed ground vehicle navigation over rough 3d terrain." IROS, 2012.

[18] E. Velenis, D. Katzourakis, E. Frazzoli, P. Tsiotras, and R. Happee, "Steady-state drifting stabilization of RWD vehicles," *Control Engineering Practice*, vol. 19, no. 11, pp. 1363 – 1376, 2011.