Value Function Learning for AutoRally Racing

Nolan Wagener[1] Panagiotis Tsiotras[2] Byron Boots[3]

Abstract— **We present a framework for value function learning that propagates value information over a given horizon instead of the usual single time step. By starting with a standard reinforcement learning problem, we relax the problem into another reinforcement learning problem where each time step corresponds to multiple time steps in the original problem. We then apply this framework to a simulated racing task and show that learning a value function over the horizon can improve performance of the car when using a relatively short planning horizon.**

I. Introduction

Autonomous racing has recently gathered more research interest, including how to automatically learn how to race. Autonomous racing techniques typically rely on model predictive control (MPC) and usually follow one of two approaches: either a two-level approach that first optimizes a reference trajectory and then constructs a controller to track the trajectory [1]–[4], or a one-level approach that directly solves the minimum-time problem [3], [5]. While the former approach splits the problem into two easier subproblems, the trajectory optimization phase tends to be purely geometric [1], [3] or uses linearized dynamics [2], [4]. Thus, the trajectory optimization phase may produce an infeasible or suboptimal trajectory. The latter approach directly accounts for the dynamics, but convex relaxations are made to allow for the use of optimization libraries like FORCES [6]. This may, for instance, limit the parts of the track the car is allowed to consider. A fundamental downside of both approaches is their reliance on the MPC principle, which means that they cannot globally optimize the minimum-time task. Indeed, prior works typically plan for only about a second ahead, which is not long enough to yield optimal behavior.

Dynamic programming techniques, on the other hand, can systematically find optimal behaviors by propagating the value information over time (e.g., through value iteration or Q-learning). Exact dynamic programming techniques are intractable to use in large (or continuous) spaces, which gives rise to approximate dynamic programming (ADP) techniques like fitted value iteration or fitted Q-iteration. ADP techniques have proven to be quite useful, yielding state-of-the-art results in several games [7]–[9] and impressive results in robotics [10]–[12]. For backgammon, Atari games, and the cited robotics papers, a trained Q-function is sufficient to make decisions based on a combination of ease of the task and representational power of the neural networks used to represent the Q-function. For the game of Go, though, a combination of a fitted value function and a randomized planning algorithm was necessary to surpass human-level performance [9], [13], owing to the combinatorial size of the state space and the difficulty of learning an accurate value estimate. In general, errors in a fitted value function with respect to the true value function can incur performance deficiencies proportional to the value error and length of the problem, something which can be mitigated significantly when the value function is used as the terminal cost in a planning algorithm [14].

The idea of combining planning with value functions can be extended to control, as MPC approaches can be viewed as solving a truncated optimal control problem [15], which can then be converted to an infinite-horizon problem by using the value function as a terminal cost. One recent approach that relies on this idea of combining MPC and ADP is learning MPC (LMPC) [16], which maintains an estimate of the value function based on interactions with the system of interest. LMPC forms a value function estimate by building a safe set from the visited states over an episode and recording the observed value from each visited state. Within the convex hull of the safe set, the value is estimated through a convex interpolation. This approach guarantees asymptotic stability and non-decreasing performance over time. LMPC has recently been applied to racing [17], [18], where the authors use practical approximations by only relying on recent data to form the value estimate. The authors then demonstrate consistently improving performance on a 1:10 scale car from an initially conservative controller, ultimately reaching the limits of the handling capability of the car. We observe, however, that despite the fact that value estimates come from the control system (and are therefore unbiased estimates), noisiness in the control system can induce pessimistic estimates of the value. For example, if the car drives over a rock on a turn that causes the car to not make the turn cleanly, this can give the misimpression that the control strategy was faulty.

Autonomous racing with the AutoRally platform [19] has made great strides over the past few years, owing in large part to the model predictive path integral (MPPI) control algorithm [20]. MPPI is a stochastic optimization-based MPC algorithm which operates by sampling about a nominal control sequence and updating the sequence based on the goodness of the sampled trajectories. An advantage

[1]Nolan Wagener is with the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA, USA 30332 nolan.wagener@gatech.edu

[2]Panagiotis Tsiotras is with the Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA, USA 30332 tsiotras@gatech.edu

[3]Byron Boots is with the Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA, USA 98195 bboots@cs.washington.edu

of this method compared to other MPC algorithms is that it can operate with general dynamics and cost functions, even discontinuous ones. This allows for high-level cost functions such as maximizing speed and remaining within the track. However, this method still lags in lap time compared to human experts [21]. One reason is due to its short planning horizon. Typically, MPPI plans for 2 to 2.5 seconds ahead, whereas human experts devise maneuvers between upcoming sections of a track, effectively planning over a longer horizon.

While one can merely increase MPPI's planning horizon to simulate what human experts do, the resulting performance actually worsens. This arises from a different shortcoming of the algorithm: Since MPPI is sampling-based, the variance of the optimization increases unfavorably with the planning horizon [22]. Furthermore, errors in the dynamics model used by MPPI can give rise to a spurious plan if the planning horizon is too long.

One can overcome the aforementioned issue while still considering a long horizon by including the value function as a terminal cost to MPPI, something already considered in [14], [23]. In these papers, the authors interleave executing an MPC algorithm on a control system with the use of the same MPC algorithm to generate value targets to train a value network. They validate their approach on simulated tasks like in-hand manipulation and humanoid pushing tasks where a learned value function is indeed necessary for MPC to solve the tasks. The authors also observe that using an MPC algorithm to generate the value targets helps to accelerate and stabilize the learning of the value function, an observation also made in [13] when solving Go.

In this work, we propose using a value learning scheme similar to [14] and [23] where we use MPC to generate value function targets. We first show how incorporating planning into value learning can be formalized as a new reinforcement learning problem, one where each decision of the agent is a control sequence and each time step of the new problem corresponds to a number of time steps in the old problem. This allows us to update the value function over longer time scales and accelerate convergence of the value function. We then validate this approach on a simulated AutoRally racing task, showing that, by incorporating a value function into MPPI, one can improve performance of the controller. We also show that using planning to train the value function results in better performance than doing repeated one-step backups in the original RL problem.

II. Approach

A. Preliminaries

We consider a Markov decision process (MDP) defined by the tuple $\mathcal{M} = (\mathcal{X}, \mathcal{U}, p, c, \gamma)$, where $\mathcal{X} \subseteq \mathbb{R}^n$ is the state space, $\mathcal{U} \subseteq \mathbb{R}^m$ is the control space, $p : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to \mathbb{R}_+$ is the (modeled) stochastic dynamics, $c : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathbb{R}$ is the cost function, and $\gamma \in [0, 1)$ is the discount factor. At some time t, we define $\boldsymbol{x}_t \in \mathcal{X}$ as the state and $\boldsymbol{u}_t \in \mathcal{U}$ as the controls. We seek a stochastic policy $\pi(\boldsymbol{u}_t|\boldsymbol{x}_t)$ that accumulates low cost over time while being noisy in the form of having high entropy[1]. Encouraging the policy to be noisy has been shown to improve robustness of the policy [24]. Given a policy π, we define its value $V^\pi(\boldsymbol{x})$ at some state $\boldsymbol{x}$ as the expected sum of discounted costs and entropies when starting at $\boldsymbol{x}$:

$$V^\pi(\boldsymbol{x}) = \mathbb{E}_{\pi,p}\left[\sum_{t=0}^{\infty} \gamma^t (c(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) - \lambda \mathcal{H}(\pi(\boldsymbol{u}_t|\boldsymbol{x}_t))) \,\middle|\, \boldsymbol{x}_0 = \boldsymbol{x}\right]$$

where $\lambda > 0$ is a temperature parameter that encourages the policy to have higher entropy.

Following [23], [25], [26], we seek a stochastic policy that optimizes the average value over some initial state distribution $p_0(\boldsymbol{x})$:

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{p_0(\boldsymbol{x})}[V^\pi(\boldsymbol{x})] \tag{1}$$

We denote the resulting value function as V^* and refer to it as the optimal value function. It can be shown [25] that the optimal value function and the corresponding state-control value Q^* satisfy the following mutual recurrence for any $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{u} \in \mathcal{U}$:

$$V^*(\boldsymbol{x}) = -\lambda \log \int_{\mathcal{U}} \exp\left(-\frac{1}{\lambda} Q^*(\boldsymbol{x}, \boldsymbol{u})\right) \mathrm{d}\boldsymbol{u} \tag{2}$$

$$Q^*(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{E}_{p(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})}[c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') + \gamma V^*(\boldsymbol{x}')] \tag{3}$$

It can also be shown that the optimal policy has the form:

$$\pi^*(\boldsymbol{u}|\boldsymbol{x}) = \exp\left(-\frac{1}{\lambda}(Q^*(\boldsymbol{x}, \boldsymbol{u}) - V^*(\boldsymbol{x}))\right) \tag{4}$$

Intuitively, controls that yield lower accumulated cost are exponentially more probable.

B. Relaxing the Time Scale

Finding the optimal policy π^* through dynamic programming algorithms like soft value iteration [25] has appealing convergence rates. In practice, however, we must rely on approximate techniques with parametric functions like neural networks to represent V^*. Thus, costs may need to be propagated across many time steps through bootstrapped training of a neural network, which can result in divergence due to compounding errors [27], [28]. Recent work [14], [23] has shown that incorporating planning into the generation of value function targets can accelerate and stabilize the value function training since the costs are being more directly propagated over a longer horizon.

We thus propose to restrict ourselves to stochastic policies $\tilde{\pi}(\boldsymbol{u}_t, \ldots, \boldsymbol{u}_{t+H-1}|\boldsymbol{x}_t)$ over a *horizon* H. We also modify the problem in (1) as follows: Instead of querying the policy π every time step, we sample an *H-step control sequence every H time steps* from the policy $\tilde{\pi}$ and apply it open-loop. In doing so, we relax (1) to yield a reinforcement learning problem over $\tilde{\pi}$. Abbreviating $c(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1})$ with c_t, we observe the following:

[1]For some distribution $p(x)$, the entropy of the distribution is defined as $\mathcal{H}(p(x)) = \mathbb{E}_{p(x)}[-\log p(x)]$.

$$
\begin{aligned}
&\min_{\pi} \mathbb{E}_{\pi,p,p_0}\left[\sum_{t=0}^{\infty}\gamma^t(c_t - \lambda\mathcal{H}(\pi(\boldsymbol{u}_t|\boldsymbol{x}_t)))\right] \\
&\leq \min_{\tilde{\pi}} \mathbb{E}_{\tilde{\pi},p,p_0}\left[\sum_{k=0}^{\infty}\sum_{h=0}^{H-1}\gamma^{kH+h}(c_{kH+h} - \lambda\mathcal{H}(\tilde{\pi}(\boldsymbol{u}_{kH+h}|\boldsymbol{x}_{kH})))\right] \\
&\leq \min_{\tilde{\pi}} \mathbb{E}_{\tilde{\pi},p,p_0}\left[\sum_{k=0}^{\infty}\gamma^{kH}\sum_{h=0}^{H-1}\Big(\gamma^h c_{kH+h} - \gamma^{H-1}\lambda\mathcal{H}(\tilde{\pi}(\boldsymbol{u}_{kH+h}|\boldsymbol{x}_{kH}))\Big)\right] \\
&\leq \min_{\tilde{\pi}} \mathbb{E}_{\tilde{\pi},p,p_0}\left[\sum_{k=0}^{\infty}\gamma^{kH}\left\{\sum_{h=0}^{H-1}\gamma^h c_{kH+h}\right.\right. \\
&\qquad\qquad \left.\left. - \gamma^{H-1}\lambda\mathcal{H}\big(\tilde{\pi}(\boldsymbol{u}_{kH+(0:H-1)}|\boldsymbol{x}_{kH})\big)\right\}\right] \qquad (5)
\end{aligned}
$$

where the first inequality comes from restricting our policy to $\tilde{\pi}$, the second inequality from $\gamma^h \geq \gamma^{H-1}$ for $0 \leq h \leq H-1$, and the third inequality from the fact that the joint entropy is at most the sum of the marginal entropies.

The structure of the original reinforcement learning problem is still present, and indeed we can reduce (5) to a reinforcement learning problem by properly defining a new MDP. We define our new time index as k, the new state space as $\tilde{\mathcal{X}} = \mathcal{X}$, and the new control space as $\tilde{\mathcal{U}} = \mathcal{U}^H$. Accordingly, we define the state and controls at time k as $\tilde{\boldsymbol{x}}_k$ and $\tilde{\boldsymbol{u}}_k$, respectively. The new state and controls are related to the old ones as follows:

$$
\begin{aligned}
\tilde{\boldsymbol{x}}_k &= \boldsymbol{x}_{kH} \\
\tilde{\boldsymbol{u}}_k &= (\boldsymbol{u}_{kH}, \ldots, \boldsymbol{u}_{kH+H-1})
\end{aligned}
$$

We define the new stochastic dynamics $\tilde{p}(\tilde{\boldsymbol{x}}_{k+1}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k)$ as the rolled-out dynamics of the original MDP marginalized over the intermediate states:

$$
\begin{aligned}
&\tilde{p}(\tilde{\boldsymbol{x}}_{k+1}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k) \\
&= p(\boldsymbol{x}_{kH+H}|\boldsymbol{x}_{kH}, \boldsymbol{u}_{kH+(0:H-1)}) \\
&= \int_{\mathcal{X}^{H-1}} p(\boldsymbol{x}_{kH+(1:H)}|\boldsymbol{x}_{kH}, \boldsymbol{u}_{kH+(0:H-1)})\,\mathrm{d}\boldsymbol{x}_{kH+(1:H-1)} \\
&= \int_{\mathcal{X}^{H-1}} \prod_{h=0}^{H-1} p(\boldsymbol{x}_{kH+h+1}|\boldsymbol{x}_{kH+h}, \boldsymbol{u}_{kH+h})\,\mathrm{d}\boldsymbol{x}_{kH+(1:H-1)}
\end{aligned}
$$

We define the new cost function $\tilde{c}(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k, \tilde{\boldsymbol{x}}_{k+1})$ as the expected sum of discounted costs over the intermediate states:

$$
\tilde{c}(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k, \tilde{\boldsymbol{x}}_{k+1}) = \mathbb{E}_{\boldsymbol{x}_{kH+(1:H-1)}|\boldsymbol{x}_{kH}, \boldsymbol{u}_{kH}, \boldsymbol{x}_{kH+H}}\left[\sum_{h=0}^{H-1}\gamma^h c_{kH+h}\right]
$$

Finally, we define the new discount factor $\tilde{\gamma}$ as $\tilde{\gamma} = \gamma^H$. We define the new MDP $\tilde{\mathcal{M}}$ as $\tilde{\mathcal{M}} = (\tilde{\mathcal{X}}, \tilde{\mathcal{U}}, \tilde{p}, \tilde{c}, \tilde{\gamma})$ with temperature $\tilde{\lambda} = \gamma^{H-1}\lambda$.

We now state and prove the main contribution of the paper.

Proposition 1: The reinforcement learning problem

$$
\min_{\tilde{\pi}} \mathbb{E}_{\tilde{\pi},\tilde{p},p_0}\left[\sum_{k=0}^{\infty}\tilde{\gamma}^k\Big(\tilde{c}(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k, \tilde{\boldsymbol{x}}_{k+1}) - \tilde{\lambda}\mathcal{H}(\tilde{\pi}(\tilde{\boldsymbol{u}}_k|\tilde{\boldsymbol{x}}_k))\Big)\right] \quad (6)
$$

is identical to (5).

Proof: We show that the objective functions are the same. We first substitute $\tilde{\gamma}$ and $\tilde{\lambda}$ into (5) and move the outer summation through the expectation so that we start with:

$$
\sum_{k=0}^{\infty}\tilde{\gamma}^k\mathbb{E}_{\tilde{\pi},p,p_0}\left[\sum_{h=0}^{H-1}\gamma^h c_{kH+h} - \tilde{\lambda}\mathcal{H}\big(\tilde{\pi}(\boldsymbol{u}_{kH+(0:H-1)}|\boldsymbol{x}_{kH})\big)\right]
$$

We then note that we can marginalize out all variables in the expectation except for the relevant subtrajectories $\boldsymbol{x}_{kH+(0:H)}$ and $\boldsymbol{u}_{kH+(0:H-1)}$. We define $\rho_t^{\tilde{\pi}}(\boldsymbol{x}_t)$ as the marginal distribution of the state $\boldsymbol{x}_t$ at time step t when we run $\tilde{\pi}$ under dynamics p and initial state distribution p_0. Thus, the expectation in the above expression can be written as

$$
\mathbb{E}_{\rho_{kH}^{\tilde{\pi}}(\tilde{\boldsymbol{x}}_k)}\left[\mathbb{E}_{\tilde{\pi}(\tilde{\boldsymbol{u}}_k|\tilde{\boldsymbol{x}}_k)}\mathbb{E}_{\boldsymbol{x}_{kH+(1:H)}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k}\left[\sum_{h=0}^{H-1}\gamma^h c_{kH+h}\right] - \tilde{\lambda}\mathcal{H}(\tilde{\pi}(\tilde{\boldsymbol{u}}_k|\tilde{\boldsymbol{x}}_k))\right]
$$

We then observe that the innermost expectation is equal to

$$
\begin{aligned}
&\mathbb{E}_{\boldsymbol{x}_{kH+(1:H-1)}, \boldsymbol{x}_{(k+1)H}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k}\left[\sum_{h=0}^{H-1}\gamma^h c_{kH+h}\right] \\
&= \mathbb{E}_{\tilde{\boldsymbol{x}}_{k+1}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k}\mathbb{E}_{\boldsymbol{x}_{kH+(1:H-1)}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k, \tilde{\boldsymbol{x}}_{k+1}}\left[\sum_{h=0}^{H-1}\gamma^h c_{kH+h}\right] \\
&= \mathbb{E}_{\tilde{p}(\tilde{\boldsymbol{x}}_{k+1}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k)}[\tilde{c}(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k, \tilde{\boldsymbol{x}}_{k+1})]
\end{aligned}
$$

so that we have

$$
\mathbb{E}_{\rho_{kH}^{\tilde{\pi}}(\tilde{\boldsymbol{x}}_k)}\left[\mathbb{E}_{\tilde{\pi}(\tilde{\boldsymbol{u}}_k|\tilde{\boldsymbol{x}}_k)}\mathbb{E}_{\tilde{p}(\tilde{\boldsymbol{x}}_{k+1}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k)}[\tilde{c}(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k, \tilde{\boldsymbol{x}}_{k+1})] - \tilde{\lambda}\mathcal{H}(\tilde{\pi}(\tilde{\boldsymbol{u}}_k|\tilde{\boldsymbol{x}}_k))\right]
$$

Defining $\tilde{\rho}_k^{\tilde{\pi}}(\tilde{\boldsymbol{x}}_k)$ as the marginal distribution of $\tilde{\boldsymbol{x}}_k$ at time step k when we run $\tilde{\pi}$ under dynamics $\tilde{p}$ and initial state distribution p_0 and noting that $\tilde{\rho}_k^{\tilde{\pi}}(\tilde{\boldsymbol{x}}_k) = \rho_{kH}^{\tilde{\pi}}(\tilde{\boldsymbol{x}}_k)$, we observe the above expression is equal to

$$
\mathbb{E}_{\tilde{\rho}_k^{\tilde{\pi}}(\tilde{\boldsymbol{x}}_k)}\left[\mathbb{E}_{\tilde{\pi}(\tilde{\boldsymbol{u}}_k|\tilde{\boldsymbol{x}}_k)}\mathbb{E}_{\tilde{p}(\tilde{\boldsymbol{x}}_{k+1}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k)}[\tilde{c}(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k, \tilde{\boldsymbol{x}}_{k+1})] - \tilde{\lambda}\mathcal{H}(\tilde{\pi}(\tilde{\boldsymbol{u}}_k|\tilde{\boldsymbol{x}}_k))\right]
$$

Substituting this expression into the expression at the beginning of the proof yields

$$
\sum_{k=0}^{\infty}\tilde{\gamma}^k\mathbb{E}_{\tilde{\rho}_k^{\tilde{\pi}}(\tilde{\boldsymbol{x}}_k)}\left[\mathbb{E}_{\tilde{\pi}(\tilde{\boldsymbol{u}}_k|\tilde{\boldsymbol{x}}_k)}\mathbb{E}_{\tilde{p}(\tilde{\boldsymbol{x}}_{k+1}|\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k)}[\tilde{c}_k] - \tilde{\lambda}\mathcal{H}(\tilde{\pi}(\tilde{\boldsymbol{u}}_k|\tilde{\boldsymbol{x}}_k))\right]
$$

where $\tilde{c}_k$ is shorthand for $\tilde{c}(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k, \tilde{\boldsymbol{x}}_{k+1})$. Grouping the three expectations into one expectation (over $\tilde{\boldsymbol{x}}_k$, $\tilde{\boldsymbol{u}}_k$, and $\tilde{\boldsymbol{x}}_{k+1}$), re-introducing all marginalized states and controls into the expectation, and moving the expectation through the summation, we get the objective in (6). ■

We denote the optimal value function, state-control value function, and policy corresponding to (6) as $\tilde{V}^*$, $\tilde{Q}^*$, and $\tilde{\pi}^*$, respectively, and find that they take an expectedly similar form to (2)–(4):

$$
\tilde{V}^*(\tilde{\boldsymbol{x}}) = -\tilde{\lambda}\log\int_{\tilde{\mathcal{U}}}\exp\bigg(-\frac{1}{\tilde{\lambda}}\tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}})\bigg)\,\mathrm{d}\tilde{\boldsymbol{u}} \quad (7)
$$

$$
\tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) = \mathbb{E}_{\tilde{p}(\tilde{\boldsymbol{x}}'|\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}})}\Big[\tilde{c}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{x}}') + \tilde{\gamma}\tilde{V}^*(\tilde{\boldsymbol{x}}')\Big] \quad (8)
$$

$$
\tilde{\pi}^*(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}}) = \exp\bigg(-\frac{1}{\tilde{\lambda}}\Big(\tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) - \tilde{V}^*(\tilde{\boldsymbol{x}})\Big)\bigg) \quad (9)
$$

C. Estimating the Value Function

We therefore have an MDP that runs at a coarser time scale (one time step in $\tilde{\mathcal{M}}$ corresponds to H time steps in $\mathcal{M}$). One consequence is that dynamic programming algorithms like soft value iteration can converge much more quickly (at a rate of $\tilde{\gamma} = \gamma^H$ in $\tilde{\mathcal{M}}$ versus γ in $\mathcal{M}$) without increasing the dimensionality of the state. Though the dimensionality of the controls has increased to Hm, the value function in (7) involves integration (or expectation if importance sampling is used) instead of a minimization operation encountered in standard reinforcement learning. Thus, for moderate values of the temperature $\tilde{\lambda}$, we can build a low variance estimator of the value function.

As previously alluded, the value function in (7) can be recursively expressed with importance sampling. That is, for some $\tilde{\pi}$ with full support over $\tilde{\mathcal{U}}$, we have

$$\tilde{V}^*(\tilde{\boldsymbol{x}}) = -\tilde{\lambda} \log \mathbb{E}_{\tilde{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \left[\frac{1}{\tilde{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \exp\left(-\frac{1}{\tilde{\lambda}} \tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) \right) \right]$$

The importance sampler that induces the minimum variance for the estimator turns out to be the optimal policy $\tilde{\pi}^*$, which, unfortunately, is intractable to sample from. Instead, we use some Gaussian distribution $\hat{\pi}$ to perform the importance sampling. To mitigate the sampling variance, we bring $\hat{\pi}$ as close as possible to the optimal policy $\tilde{\pi}^*$. In particular, for some given state $\tilde{\boldsymbol{x}}$, we seek a $\hat{\pi}$ which minimizes the KL divergence[2] to the optimal distribution:

$$\hat{\pi}^*(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}}) = \arg\min_{\hat{\pi}} \mathrm{KL}(\tilde{\pi}^*(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}}) \,\|\, \hat{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}}))$$

This is equivalent to matching the mean and covariance of $\tilde{\pi}^*$, which we denote as $\tilde{\boldsymbol{\mu}}^*(\tilde{\boldsymbol{x}})$ and $\tilde{\Sigma}^*(\tilde{\boldsymbol{x}})$, respectively. We can find the mean and covariance through importance sampling:

$$\begin{aligned} \tilde{\boldsymbol{\mu}}^*(\tilde{\boldsymbol{x}}) &= \mathbb{E}_{\tilde{\pi}^*(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})}[\tilde{\boldsymbol{u}}] \\ &= \frac{\int_{\tilde{\mathcal{U}}} \exp\left(-\frac{1}{\tilde{\lambda}} \tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) \right) \tilde{\boldsymbol{u}} \, \mathrm{d}\tilde{\boldsymbol{u}}}{\int_{\tilde{\mathcal{U}}} \exp\left(-\frac{1}{\tilde{\lambda}} \tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) \right) \mathrm{d}\tilde{\boldsymbol{u}}} \\ &= \frac{\mathbb{E}_{\hat{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \left[\frac{1}{\hat{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \exp\left(-\frac{1}{\tilde{\lambda}} \tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) \right) \tilde{\boldsymbol{u}} \right]}{\mathbb{E}_{\hat{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \left[\frac{1}{\hat{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \exp\left(-\frac{1}{\tilde{\lambda}} \tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) \right) \right]} \end{aligned} \tag{10}$$

where we overload $\hat{\pi}$ to mean some setting of our Gaussian distribution. Similarly, for the covariance:

$$\begin{aligned} \tilde{\Sigma}^*(\tilde{\boldsymbol{x}}) &= \mathbb{E}_{\tilde{\pi}^*(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \left[(\tilde{\boldsymbol{u}} - \tilde{\boldsymbol{\mu}}^*(\tilde{\boldsymbol{x}}))(\tilde{\boldsymbol{u}} - \tilde{\boldsymbol{\mu}}^*(\tilde{\boldsymbol{x}}))^\mathsf{T} \right] \\ &= \frac{\mathbb{E}_{\hat{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \left[\frac{1}{\hat{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \exp\left(-\frac{1}{\tilde{\lambda}} \tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) \right) (\tilde{\boldsymbol{u}} - \tilde{\boldsymbol{\mu}}^*(\tilde{\boldsymbol{x}}))(\tilde{\boldsymbol{u}} - \tilde{\boldsymbol{\mu}}^*(\tilde{\boldsymbol{x}}))^\mathsf{T} \right]}{\mathbb{E}_{\hat{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \left[\frac{1}{\hat{\pi}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})} \exp\left(-\frac{1}{\tilde{\lambda}} \tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) \right) \right]} \end{aligned} \tag{11}$$

From the consistency condition of the value function and the form of the importance sampler, we can now use a practical algorithm to estimate the value function. We first define the Bellman operator $\tilde{\mathcal{B}}$ as

$$\tilde{\mathcal{B}}\tilde{V}(\tilde{\boldsymbol{x}}) = -\tilde{\lambda} \log \int_{\tilde{\mathcal{U}}} \exp\left(-\frac{1}{\tilde{\lambda}} \mathbb{E}_{\tilde{p}(\tilde{\boldsymbol{x}}'|\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}})} \left[\tilde{c}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{x}}') + \tilde{\gamma} \tilde{V}(\tilde{\boldsymbol{x}}') \right] \right) \mathrm{d}\tilde{\boldsymbol{u}} \tag{12}$$

For some given $\tilde{\boldsymbol{x}}$ and value function estimator $\tilde{V}$, we compute (12) by first finding the Gaussian distribution $\hat{\pi}^*(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}})$ which minimizes the KL divergence to the optimal distribution induced by $\tilde{V}$. In this case, the Gaussian is defined by Equations (10) and (11) but with $\mathbb{E}_{\tilde{p}(\tilde{\boldsymbol{x}}'|\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}})} \left[\tilde{c}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{x}}') + \tilde{\gamma} \tilde{V}(\tilde{\boldsymbol{x}}') \right]$ in place of $\tilde{Q}^*(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}})$. From there, we use importance sampling with $\hat{\pi}^*$ in (12) to estimate the Bellman operator at $\tilde{\boldsymbol{x}}$.

We use a neural network $\tilde{V}_{\boldsymbol{\theta}}$ parameterized by some vector $\boldsymbol{\theta}$ to approximate the value function $\tilde{V}^*$. Over some distribution $\rho(\tilde{\boldsymbol{x}})$ over states, we seek a setting of $\boldsymbol{\theta}$ which minimizes the regression error

$$\boldsymbol{\theta}_{\text{new}} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\rho(\tilde{\boldsymbol{x}})} \left[\left(\tilde{V}_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}) - \tilde{\mathcal{B}}\tilde{V}_{\boldsymbol{\theta}_{\text{old}}}(\tilde{\boldsymbol{x}}) \right)^2 \right] \tag{13}$$

where $\boldsymbol{\theta}_{\text{old}}$ is the parameter setting when we perform the regression and $\boldsymbol{\theta}_{\text{new}}$ is the parameter setting that then overrides $\boldsymbol{\theta}_{\text{old}}$ in our neural network. This bootstrapped regression can be done repeatedly to have $\tilde{V}_{\boldsymbol{\theta}}$ more directly approximate the infinite horizon value function $\tilde{V}^*$.

D. Racing Task

Fig. 1: The race track of interest. Image taken from [20].

Fig. 2: The AutoRally car. Image taken from [29].

For the AutoRally task, we seek to minimize the lap time on a given dirt track (Fig. 1) when driving a 1:5-scale rally car (Fig. 2). We respectively define the state $\boldsymbol{x}$ and controls $\boldsymbol{u}$ as $\boldsymbol{x} = (s, e_y, e_\psi, \varphi, v_x, v_y, \dot{\psi})$ and $\boldsymbol{u} = (a, \delta)$. Here, the position of the car is represented in a curvilinear reference frame (Fig. 3) so that s is the longitudinal position along the track, e_y is the lateral deviation from the center of the track, and e_ψ is the deviation of the car's heading from the centerline heading. We define φ as the roll of the car body, v_x as the longitudinal velocity, v_y as the lateral velocity, and $\dot{\psi}$ as the heading rate. For the controls, a is the car's throttle, and δ is the steering angle of the front wheels.

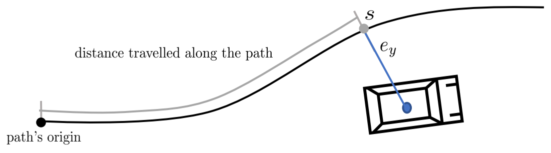


Fig. 3: Curvilinear coordinate frame. Image taken from [18].

[2]For two distributions $p(x)$ and $q(x)$ such that $q(x) = 0 \implies p(x) = 0$ for any x, the KL divergence is defined as $\mathrm{KL}(p(x) \,\|\, q(x)) = \mathbb{E}_{p(x)} \left[\log \frac{p(x)}{q(x)} \right]$.

We define the step cost c to maximize progress along the track while penalizing for any violated constraints:

$$c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') = 1000(s - s') + 10M(e_y) + 10^5 \mathbf{1}_{\text{violation}}(\boldsymbol{x}, \boldsymbol{x}')$$

Here, $M(e_y)$ is the positional cost of the car, with lowest cost at the center of the track (i.e., $e_y = 0$) and increasing cost as we move towards either side of the track. The indicator function $\mathbf{1}_{\text{violation}}(\boldsymbol{x}, \boldsymbol{x}')$ is one if either state is outside the track boundaries or if the car's heading is more than 60 degrees off from the current track heading (i.e., $|e_\psi| \geq 60°$). Otherwise, the value is zero. If such a violation occurs, the car is treated as reaching some terminal "crash" state where we accrue a (discounted) cost of 10^5 for each remaining time step in the planning horizon.

We model the dynamics of the car as a combination of known kinematics (since the car is a rigid body) and a neural network to model the acceleration. The neural network is fitted to acceleration data collected from a human driving the car through various maneuvers. Since both the kinematics and neural network are deterministic, the resulting dynamics $p(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{u})$ correspond to a Dirac delta function (i.e., a deterministic mapping).

To control the car, we employ the MPPI algorithm, which operates by similarly finding a Gaussian distribution which minimizes the KL divergence to the optimal distribution in (9). The main difference with the previous section is that we control in receding horizon fashion (i.e., plan H steps into the future, but re-plan every time step instead of every H time steps) and keep the covariance fixed. The former improves robustness of the control system, while the latter has been found to be more reliable than optimizing the covariance in a continual task like racing, mainly to prevent covariance collapse [30]. For MPPI, we define the optimal distribution to be the one induced by our neural network $\tilde{V}_{\boldsymbol{\theta}}$:

$$\tilde{\pi}^*_{\boldsymbol{\theta}}(\tilde{\boldsymbol{u}}|\tilde{\boldsymbol{x}}) = \exp\left(-\frac{1}{\lambda}\left(\mathbb{E}_{\tilde{p}(\tilde{\boldsymbol{x}}'|\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}})}\left[\tilde{c}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{x}}') + \tilde{\gamma}\tilde{V}_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}')\right] - \tilde{V}_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})\right)\right)$$

III. Experiments and Results

For all our experiments, we use the dynamics model discussed in the previous section as our simulator. We set the discount factor and temperature in $\mathcal{M}$ to be $\gamma = 0.995$ and $\lambda = 0.3$, respectively. We represent the value function estimator with a neural network with two hidden layers (32 neurons per hidden layer with a $\tanh$ activation) and use the exponential function as the output activation. The exponential function constrains the value to be positive[3] and allows us to learn a function with a large output range.

When training our value function, we maintain a dataset of states and corresponding value targets from (12). If we perform bootstrapping, the value targets get recomputed using the current value function estimator $\tilde{V}_{\boldsymbol{\theta}}$. After performing some number of episodes in the simulator, we append the corresponding states and value targets to our dataset. We then solve (13) using the Adam optimizer with a step size of 10^{-3} and 5000 epochs over the dataset.

We show that our value learning framework can be successfully applied to a simulated racing task. In particular, we show that incorporating a value function as a terminal cost can enhance the performance of MPPI, especially if the planning horizon is short. We also show that performing a Bellman backup over a horizon as in (12) can result in better performance than performing an equivalent number of 1-step Bellman backups.

A. Value Functions for Racing

We train a value function by doing a single batch of 25-step Bellman backups (corresponding to a horizon of 0.5 seconds) after MPPI drives the car 5 laps with a planning horizon of 50 steps (corresponding to a horizon of 1 second).

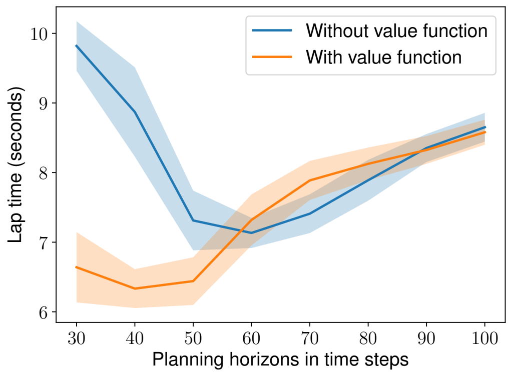
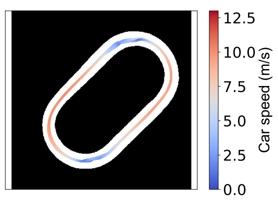


Fig. 4: Lap times with different planning horizons and whether a value function is incorporated. Shaded regions correspond to ± 1 standard deviation.

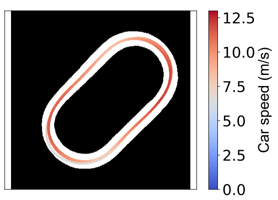
When using the value function as a terminal cost in MPPI, we find that lap times vastly improve when a shorter planning horizon is used (Fig. 4). As the planning horizons increases past 50 time steps, the value function serves a much lesser role owing to the heavy discount factor. Furthermore, the MPPI algorithm becomes noisier owing to the larger horizon, which is reflected in the lap times increasing with the planning horizon. With a reasonably accurate value function, a short planning horizon offers the best performance since the optimization is much less noisy, and the value function acts as a guide for MPPI. This is corroborated by the fact that in Fig. 4 shorter planning horizons with a terminal value function provide the shortest lap times.

We can also see the effect of the value function by comparing the paths taken with and without it (Fig. 5). With a short horizon and no value function, the car cannot drive around the turn without significantly losing momentum (Fig. 5a). On the other hand, the value function provides MPPI enough foresight to make the entire turn at high speeds (Fig. 5b).

[3]Accordingly, we add a small positive number to the cost function to ensure it is positive.



(a) Without value function



(b) With value function

Fig. 5: MPPI with planning horizon of 30 time steps.

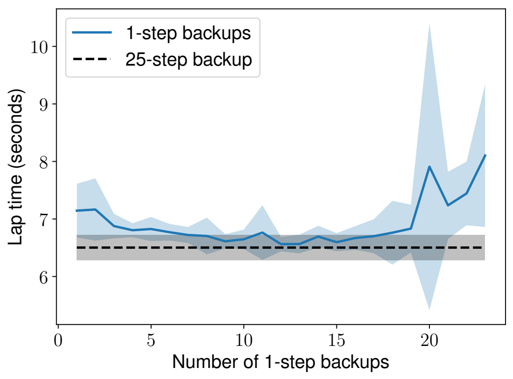


Fig. 6: Lap times when using 1-step Bellman backups. Shaded regions correspond to ± 1 standard deviation.

B. Comparison with One-Step Backups

While, in principle, we can instead perform multiple 1-step Bellman backups, we found that does not work well for the AutoRally task, as shown in Fig. 6. Here, we fix the MPPI planning horizon to 50 time steps. While there is improvement in lap times up to 12 backups, the performance deteriorates thereafter due to errors in the cost function propagating over time. This manifests in the car taking a turn more slowly than necessary, as seen in Fig. 7. Furthermore, none of the trained value function results in a lower lap time than the network trained on a single 25-step backup. This is likely because directly considering the right controls to take over a moderate horizon allows to more easily generate good value targets rather than relying on multiple single-step backups which may produce errors over time.

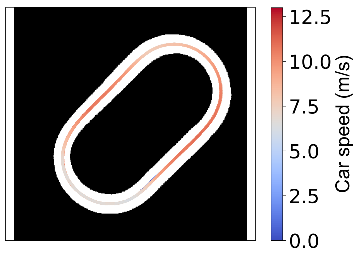


Fig. 7: Worsened performance when using 1-step bootstrapped value function.

We similarly find bootstrapping the 25-step Bellman backups results in worse lap times (Fig. 8), again due to compounding errors. In this case, the resulting value function causes the car to take both turns more conservatively.

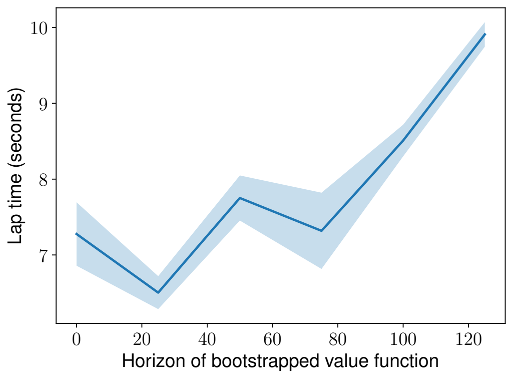


Fig. 8: Lap times when using 25-step bootstrapped value function. Shaded regions correspond to ± 1 standard deviation.

IV. Conclusions

We presented a planning-based method to train value function approximators. By starting with the reinforcement learning problem of interest, we relaxed it to a new reinforcement learning problem where each decision corresponds to a control sequence. Though we can no longer optimize the original problem, the new RL problem allows for use of efficient estimation methods and can propagate value information over a longer time scale.

We then validated our approach on a simulated racing task by showing that incorporating a value function can significantly decrease lap times when using a short planning horizon. We also showed that directly propagating costs over a horizon is more stable than performing some number of single-step backups.

Acknowledgments

We thank Sahit Chintalapudi and Matthieu Capuano for their assistance in coding the AutoRally task.

References

[1] E. Velenis and P. Tsiotras, "Minimum-time travel for a vehicle with acceleration limits: Theoretical analysis and receding-horizon implementation," *Journal of Optimization Theory and Applications*, vol. 138, no. 2, pp. 275–296, 2008.

[2] P. A. Theodosis and J. C. Gerdes, "Generating a racing line for an autonomous racecar using professional driving techniques," in *ASME 2011 Dynamic Systems and Control Conference and Bath/ASME Symposium on Fluid Power and Motion Control*. American Society of Mechanical Engineers, 2011, pp. 853–860.

[3] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[4] N. R. Kapania, J. Subosits, and J. C. Gerdes, "A sequential two-step algorithm for fast generation of vehicle racing trajectories," *Journal of Dynamic Systems, Measurement, and Control*, vol. 138, no. 9, p. 091005, 2016.

[5] R. Verschueren, S. De Bruyne, M. Zanon, J. V. Frasch, and M. Diehl, “Towards time-optimal race car driving using nonlinear MPC in real-time,” in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 2505–2510.

[6] A. Domahidi and J. Jerez, “FORCES Professional,” *embotech GmbH (*`https://embotech.com/FORCES-Pro`*)*, 2014.

[7] G. Tesauro, “Temporal difference learning and TD-Gammon,” *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, p. 484, 2016.

[10] R. Hafner and M. Riedmiller, “Neural reinforcement learning controllers for a real robot application,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 2098–2103.

[11] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[12] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, “Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation,” in *Conference on Robot Learning*, 2018, pp. 651–673.

[13] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, p. 354, 2017.

[14] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, “Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control,” in *International Conference on Learning Representations*, 2019.

[15] D. P. Bertsekas, “Dynamic programming and suboptimal control: A survey from ADP to MPC,” *European Journal of Control*, vol. 11, no. 4-5, pp. 310–334, 2005.

[16] U. Rosolia and F. Borrelli, “Learning model predictive control for iterative tasks. A data-driven control framework,” *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2017.

[17] U. Rosolia, A. Carvalho, and F. Borrelli, “Autonomous racing using learning model predictive control,” in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 5115–5120.

[18] U. Rosolia and F. Borrelli, “Learning How to Autonomously Race a Car: A Predictive Control Approach,” *IEEE Transactions on Control Systems Technology*, 2019.

[19] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg, “AutoRally: An Open Platform for Aggressive Autonomous Driving,” *IEEE Control Systems Magazine*, vol. 39, no. 1, pp. 26–55, 2019.

[20] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, “Information theoretic MPC for model-based reinforcement learning,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[21] B. Goldfain, “Autonomous Rally Racing with AutoRally and Model Predictive Control,” Ph.D. dissertation, Georgia Institute of Technology, 2019.

[22] A. Vemula, W. Sun, and J. Bagnell, “Contrasting Exploration in Parameter and Action Space: A Zeroth-Order Optimization Perspective,” in *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019, pp. 2926–2935.

[23] M. Bhardwaj, A. Handa, D. Fox, and B. Boots, “Information Theoretic Model Predictive Q-Learning,” in *Learning for Dynamics and Control*, 2020, pp. 840–850.

[24] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, “Composable deep reinforcement learning for robotic manipulation,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6244–6251.

[25] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, “Reinforcement learning with deep energy-based policies,” in *Proceedings of the 34th International Conference on Machine Learning*. JMLR. org, 2017, pp. 1352–1361.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International Conference on Machine Learning*, 2018, pp. 1856–1865.

[27] J. N. Tsitsiklis and B. Van Roy, “Analysis of temporal-diffference learning with function approximation,” in *Advances in Neural Information Processing Systems*, 1997, pp. 1075–1081.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, “Agile autonomous driving using end-to-end deep imitation learning,” in *Robotics: Science and Systems*, 2018.

[30] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, “Information-theoretic model predictive control: Theory and applications to autonomous driving,” *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, 2018.