Real-World Reinforcement Learning via Multi-Fidelity Simulators

Mark Cutler, Thomas J. Walsh, Jonathan P. How, *Senior Member, IEEE*

***Abstract*—Reinforcement learning (RL) can be a tool for designing policies and controllers for robotic systems. However, the cost of real-world samples remains prohibitive as many RL algorithms require a large number of samples before learning useful policies. Simulators are one way to decrease the number of required real-world samples, but imperfect models make deciding when and how to trust samples from a simulator difficult. We present a framework for efficient RL in a scenario where multiple simulators of a target task are available, each with varying levels of fidelity. The framework is designed to limit the number of samples used in each successively higher-fidelity/cost simulator by allowing a learning agent to choose to run trajectories at the lowest level simulator that will still provide it with useful information. Theoretical proofs of the framework's sample complexity are given and empirical results are demonstrated on a remote controlled car with multiple simulators. The approach enables RL algorithms to find near-optimal policies in a physical robot domain with fewer expensive real-world samples than previous transfer approaches or learning without simulators.**

***Index Terms*—Learning and Adaptive Systems, Reinforcement Learning, Autonomous Agents, Animation and Simulation**

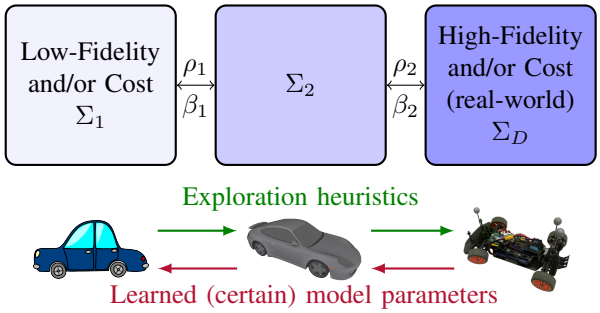


Fig. 1: The MFRL architecture: a multi-fidelity chain of simulators and learning agents. Agents send exploration heuristics to higher-fidelity agents and learned model parameters to lower-fidelity agents. The environments are related by state mappings ρ_i and optimism bounds β_i. Control switches between learning agents, going to high-fidelity levels when an optimal policy is found, and to lower-fidelity levels when unexplored regions are encountered.

I. INTRODUCTION

DESIGNING controllers and decision-making policies for autonomous agents such as robots is often difficult and tedious. Rather than designing custom controllers for each agent in every new situation, *reinforcement learning* (RL) [1] can be used to allow robots to autonomously discover novel policies in new situations. Unfortunately, as noted in Ref. [2], naïve application of RL algorithms to robotic domains often yields poor performance. The typical "curse of dimensionality" encountered in RL problems becomes especially difficult when the needed samples come from the real world, as robots are expensive, run in real-time, often require direct human supervision, and are subject to hardware degradation.

One tactic for decreasing the number of real-world samples needed in robotic RL is for the learning agent to generate samples through querying a simulation of the robot. Simulated samples are typically much less costly (financially and temporally) to obtain than samples from a real robot. In addition, there are often multiple simulations of a particular robot available, or multiple methods for implementing simulators, each with a different cost for generating samples associated with it. For instance, a car can be modeled using simple equations of motion or it can be modeled by complex simulations that account for aerodynamic drag, suspension, tire shear, etc. The more complex model will usually take more time to evaluate (is more costly), but remains less expensive than running the real car.

Unfortunately, even very complex and costly simulations of robots rarely (if ever) perfectly model the real world, and transferring policies learned in an imperfect simulation directly to a robot can yield poor performance [3]. However, simulations typically capture some aspects of an environment accurately and can provide valuable information to augment real-world data [4], [5].

In this paper we consider the problem of efficient real-world RL when multiple simulations are available, each with varying degrees of fidelity to the real world. Lower-fidelity simulators are assumed to be less expensive (typically less time to evaluate), but also less accurate with respect to the actual robot. Lower-fidelity simulators are also assumed to be optimistic with respect to the real world. While this assumption may not be valid in all domains, it is consistent with our observations of many simulators.

We introduce, analyze, and empirically demonstrate a new framework, *Multi-Fidelity Reinforcement Learning* (MFRL), for performing RL with a heterogeneous set of simulators (including the real world). MFRL, depicted in Fig. 1, not only chooses actions for an agent to execute, but also chooses which simulator to perform them in. The framework combines ideas from multi-fidelity optimization [6] and advances in model-

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

Thomas Walsh is with Kronos Inc., 297 Billerica Rd., Chelmsford, MA. All work was done while he was with the Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA, USA. thomasjwalsh@gmail.com

based RL that have yielded efficient solutions to the exploration/exploitation dilemma. More specifically, heuristics from lower-fidelity simulators and adjustments from high-fidelity data (common techniques in multi-fidelity optimization) are instantiated in MFRL using the successful "optimism in the face of uncertainty" heuristic and the "Knows What It Knows" (KWIK) model-learning framework from RL [7]. The result is an agent that both

- uses information from lower-fidelity simulators to perform limited exploration in its current simulator, and
- updates the learned models of lower-fidelity agents with higher-fidelity data.

Unlike *unidirectional* methods that transfer heuristics only once to the real-world agent [8], the MFRL framework specifies rules for when the agent should move to a higher-fidelity simulator, as well as moving to a lower-fidelity level before over-exploring in a more expensive simulation. We show that these rules and the transfer of values and data provide theoretical guarantees on convergence and sample efficiency. Specifically, the framework (1) does not run actions at high-fidelity levels that have been proven to be suboptimal at lower-fidelity levels, (2) minimizes (under certain conditions) the number of samples used in the real world and (3) polynomially limits the total number of samples used in all simulators. In addition, in the worst case, MFRL provably uses no more real-world samples than unidirectional transfer approaches.

The MFRL algorithm implements single-environment, model-based learners from the KWIK-Rmax family of algorithms at each level of simulation. Theoretical results in this paper are tied to the general KWIK framework and so apply not only to tabular models of environments, but also to a large class of representations such as linear and Gaussian-noise dynamics covered by the KWIK learning framework [7]. We illustrate this theoretical link by using a Dynamic Bayesian Network (DBN) representation [9] for model learning in two separate domains.

Our main contributions are (1) introducing the MFRL framework for learning with multiple simulators, which is the first framework that dynamically chooses which simulator to perform actions in, (2) a theoretical analysis of the framework's sample complexity, and (3) several demonstrations of efficient learning on a remote-controlled (RC) car with fewer real-world data points than unidirectional transfer or learning without simulators. These results demonstrate MFRL is a provably and practically efficient manager of the low and high quality simulators often available for robotics tasks.

MFRL is most applicable in situations where simulators are already available or where the cost of coding the simulator is outweighed by the increased learning speed. In our experience, most robotics domains construct simulators for reasons other than learning, such as during the initial hardware and software develop phases, and so simulators are often readily available [10], [11].

This paper builds upon our initial results in Ref. [12]. Here we provide new theoretical and experimental results demonstrating the MFRL algorithm using non-tabular representations of the reward and transition dynamics. We also investigate the use of generative simulators, show the sensitivity of MFRL to changes in parameter choices, and give more rigorous proofs.

II. RELATED WORK

In RL, simulators are often used to train learning agents, with real world experience used later to update the simulator or the agent's policy, e.g. [4]. However, such systems do not guarantee an efficient collection of samples from different models, require practitioners to decide when to run policies in the simulator or real world, and do not guarantee efficient exploration. Another approach is to always execute actions in the real world but use a low-fidelity simulator to help compute policy search gradients [4], [5]. However, these approaches are specific to policy search algorithms and, again, do not provide exploration guarantees.

Significant recent advances in RL for robotics have also come from model-based policy search methods [13]. Real-world data are used to construct a model of the world, from which trajectory rollouts can be used to update policy parameters [14]. Methods such as trajectory optimization for exploration [15] and skill generalization [16] have shown impressive results in simulated and real robot domains, albeit often without convergence or sample complexity guarantees. By contrast, MFRL uses data not just from a target domain but also from available simulators to build *multiple* models. Furthermore, to make provably efficient use of data, MFRL uses KWIK-Rmax-style learning and planning rather than policy search techniques.

Multi-fidelity models have been considered for a single agent that learns different policies for varying observability conditions [17]. MFRL uses similar multi-fidelity models but controls the current level of fidelity to find a single policy for an observable real world task. Multi-fidelity models have been used in the multi-agent context to combine data from tasks performed in different simulators [18], but these policies were learned from traces through supervised learning, not RL.

In *transfer learning* (TL), values or model parameters are typically used to *bootstrap* learning in the next task, e.g. [19]. By contrast, MFRL uses values from lower-fidelity models as heuristics guiding exploration, and the agent, rather than nature, controls which simulator it is using, including the capability to return to a lower-fidelity simulator. MFRL also differs from TL between environments with different action sets [20] in that the simulators can have different dynamics or rewards, rather than just different available actions. Finally, recent work in TL has considered sample complexity bounds in a series of environments under both bandit [21] and full MDP conditions [22]. However, both of these works assume environments are chosen i.i.d., not by the agent itself. By contrast, we provide sample complexity results for MFRL, which actively chooses which environment to sample next.

The field of *evolutionary robotics* commonly uses a combination of simulated and real data to learn new controllers [23]. Genetic algorithms are used to evaluate candidate controllers, with most of the evaluations occurring in simulation. Several papers have investigated the "reality gap" that occurs when controllers that work well in simulated environments perform

poorly on the physical hardware, and develop robust ways of making simulators that can be used in conjunction with the physical hardware [24]–[26]. While the ideas are similar to the current work, we are focused on the RL problem where we can guarantee efficient exploration.

Similar to MFRL, Transferred Delayed Q-Learning (TDQL) [8] transfers the value function from a source learning task as a heuristic initialization for the target learning task. Because TDQL uses Delayed Q-learning [27] as its base learner, it can only be used with tabular representations, whereas our use of the KWIK framework covers a number of more general structured representations that are explored in Section VI. However, the larger difference between TDQL and MFRL is that MFRL also sends learned model parameters from higher-fidelity simulators *back*, returning to lower-fidelity simulators when useful information might be gained. We demonstrate experimentally and prove theoretically that, in the worst case, MFRL uses no more samples at the highest-fidelity level than unidirectional transfer approaches like TDQL.

Finally, MFRL extends techniques in multi-fidelity optimization (MFO) [6] to sequential decision making problems. In MFO, an optimization problem is solved using multiple models. Techniques in MFO include learning model disparities [28] and constraining search based on results from lower-fidelity models [29]. However, MFO does not consider sequential decision making tasks. MFRL borrows lessons from MFO by updating models with higher-fidelity data and performing constrained exploration based on lower-fidelity results, bringing the core intuitions of MFO to sequential decision making problems.

III. BACKGROUND AND ASSUMPTIONS

A. Reinforcement Learning

We assume that each simulator can be represented by a Markov Decision Process (MDP) [30], $M = \langle S, A, R, T, \gamma \rangle$ with states S and actions A. The bounded reward function is defined as $R(s,a) \mapsto [R_{\min}, R_{\max}]$ where $R_{\min}$ and $R_{\max}$ are real numbers. The transition function T encodes the probability of reaching some next state given the current state and action. Specifically, $T(s,a,s') = Pr(s'|s,a)$.

The optimal *value function* is specified as

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s')$$

where $V^*(s) = \max_a Q^*(s,a)$. Intuitively, $Q^*(s,a)$ is the expected sum of discounted rewards when taking action a in s and then acting optimally thereafter. A deterministic policy $\pi : S \mapsto A$ is said to be optimal when $\pi^*(s) = \text{argmax}_a Q(s,a)$. Given an MDP (including T and R), planning methods such as value iteration [30] can be used to calculate the optimal value function and optimal policy.

In RL, an agent knows S, A, and γ but not T and R, which it learns from interaction with the environment. Many classes of algorithms exist for learning in this setting including classical algorithms such as Q-learning [31], which do not build explicit representations of T and R. In this work, however, we concentrate on *model-based* RL solutions, like classical Rmax [32] that explicitly build estimates $\hat{T}$ and $\hat{R}$ from data and then use a planner, such as value iteration, to determine the optimal policy. We make this choice for two reasons. First, model-based learners are generally more sample efficient than model-free approaches. Second, and more importantly for MFRL, using a model-based approach allows us to potentially transfer parameters learned in one simulator to another, a trait we make use of when visiting a lower-fidelity simulator after collecting data in a higher-fidelity environment.

B. Sample Complexity and the KWIK Framework

To judge the exploration efficiency of our algorithm, we follow previous definitions [27] of sample complexity for an RL agent. Sample complexity analysis is used to bound the worst-case number of experiences needed by an agent to reach near-optimal behavior with high probability. Specifically, we use the following definition.

Definition 1. *The* sample complexity *of a reinforcement learning algorithm $\mathcal{A}$ that selects action a_t at state s_t on each timestep t is, with probability $1-\delta$, the maximum number of timesteps where $V^{\mathcal{A}_t}(s_t) < V^*(s_t) - \epsilon$.*

The KWIK framework [7] standardizes sample complexity analysis for model-based RL agents by measuring the number of times the learners of T and R are uncertain in making a prediction. Because samples for T and R have labels (the noisy sampled transition and the reward signal itself), these learners can be analyzed in the supervised learning setting. The KWIK framework defines sample complexity for supervised learners that initially only know the intended hypothesis class $H : X \mapsto Y$ and accuracy/confidence parameters ϵ and δ. Learning then follows the following protocol:

1) At each timestep t, the learner is given an input $x_t \in X$, potentially chosen adversarially. No distributional assumptions are made on the choice or order of inputs.
2) If the agent is certain of its prediction ($||\hat{y}_t - y_t|| < \epsilon$ with high probability), it predicts $\hat{y}_t$.
3) Otherwise, it must state "I don't know" (denoted $\perp$) and will view a noisy output z_t, where $E[z_t] = y_t$.

The framework forbids (with high probability) the agent from making an inaccurate prediction of y_t. It must explicitly admit areas of the input space where it does not yet have enough data to make accurate predictions of T and/or R. A state becomes known once it has been observed sufficiently many times for the learner to be ϵ-confident of the outcomes. The *KWIK sample complexity* for such a supervised learner is just the number of times, with probability $1-\delta$, where it predicts $\perp$. A hypothesis class H is said to be *KWIK learnable* if an agent can guarantee, with probability $1-\delta$, it will only predict $\perp$ a polynomial (in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $|H|$) number of times.

The KWIK learnability of T and R can be used to induce a polynomially sample efficient RL algorithm, through the KWIK-Rmax RL framework [7]. KWIK-Rmax maintains KWIK learners L_T and L_R with appropriate accuracy parameters and queries these learners to create an approximate MDP with transition and reward functions $\hat{T}$ and $\hat{R}$.

Because of its reliance on KWIK learners, the algorithm is explicitly aware of which areas of the state space are "known" (where L_R and L_T can make accurate predictions) or "unknown" (L_T or L_R predict $\perp$). In the unknown areas, $\perp$ predictions from the learners are replaced using the "optimism in the face of uncertainty" heuristic when constructing $\hat{T}$ and $\hat{R}$. Specifically, when L_R predicts $\perp$ for $\langle s, a\rangle$, $\hat{R}(s, a) = R_{\max}$. When L_T predicts $\perp$, a special transition to a state with a value of $V_{max} = \frac{R_{\max}}{1-\gamma}$ is inserted. This interpretation encourages exploration of unknown areas but not at the expense of already uncovered dominant policies. It has been shown that proving T and R are KWIK learnable and using them in the KWIK-Rmax framework guarantees polynomial sample complexity of the resulting RL agent [7].

In MFRL, the KWIK-Rmax framework is used at each level of simulation. Unknown and known areas at each level are cataloged by KWIK learners as specified above. However, the uninformed $R_{\max}$ heuristic is instead replaced by value functions from lower level simulators, filling in areas where learners at the current level predict $\perp$. We also show how to share learned parameters of $\hat{T}$ and $\hat{R}$ from higher-fidelity levels to increase the accuracy of lower-fidelity simulations.

C. Simulator Assumptions and Objectives

In this work, we define a *simulator* Σ as any environment that can be modeled as an MDP. We follow [7] by defining the complexity of such domains, $|\Sigma|$, as the number of parameters of their corresponding T and R representations, which may be far smaller than the number of parameters need to represent S. Also, since S may differ between Σ_i and a higher-fidelity Σ_j (some variables may be absent in Σ_i), we follow prior work in TL [19] and assume a *transfer mapping* $\rho_i : S_i \mapsto S_j$ exists. Specifically, we assume that $S_i \subseteq S_j$, that is, that the states available in lower simulators are a subset of those available at a higher level, and that ρ_i maps states in S_i to states in S_j, setting data uniformly across variables that exist in S_j, but not in S_i. For instance, in the RC car simulations, the lowest-fidelity simulator (Σ_1) does not model rotational rate $\dot{\psi}$, so states in S_1 map to all states in S_2 with the same variable values except for $\dot{\psi}$. The reverse mapping ρ_i^{-1} only applies to states in S_j with a single *default* value of the missing variable ($\dot{\psi} = 0$ for the car). We further assume that that the set of available actions is the same in all simulators.

We define *fidelity* based on how much Σ_i *overvalues* the state/actions of Σ_j. Specifically, the fidelity f of Σ_i to Σ_j, with associated mapping ρ_i and tolerance β_i, is

$$f(\Sigma_i, \Sigma_j, \rho_i, \beta_i) = \begin{cases} -\max_{s,a} |Q^*_{\Sigma_i}(s, a) - Q^*_{\Sigma_j}(\rho_i(s), a)|, \\ \qquad \text{if } \forall s, a, [Q^*_{\Sigma_j}(s, a) - Q^*_{\Sigma_i}(\rho_i(s), a) \leq \beta_i] \\ -\infty, \quad \text{otherwise} \end{cases}$$

where $s \in S_i$. Intuitively, the fidelity of Σ_i to Σ_j is inversely proportional to the maximum error in the optimal value function, given that Σ_i never undervalues a state/action pair by more than β_i. Otherwise, Σ_i is considered to have no fidelity to Σ_j. While there are many other possible definitions of fidelity (for instance based on T and R), this definition fits natural chains of simulators and facilitates efficient learning through the MFRL architecture. We note that this definition is not a distance metric (for instance, it is not symmetric), but rather describes the relationship between simulators based on optimistic differences in their value functions.

While it may seem restrictive, this relationship is fairly common in real-life simulators. For instance, in our car simulators, the lowest-fidelity simulator assumes that actions will have perfect outcomes, so aggressive maneuvers achieve their desired results. In higher-fidelity simulators, and eventually the real world, these optimistic values are replaced with more realistic outcomes/values. Hence, the simulators form an *optimistic chain*, formally defined as follows:

Definition 2. *An optimistic* multi-fidelity simulator chain *is a series of D simulators ordered $\Sigma_1, \ldots, \Sigma_D$, with Σ_D being the target task (real-world model) and $f(\Sigma_i, \Sigma_{i+1}, \rho_i, \beta_i) \neq -\infty$ for specified ρ_i and β_i.*

Intuitively, each simulator overvalues the optimal value function of the next higher simulator, with β compensating for any undervalues.

We also make the following assumptions about the cost and accessibility of the simulators.

Assumption 1. *A single step from simulator Σ_i has a larger (but polynomially bounded in $|\Sigma_{i-1}|$)* cost *than a sample from simulator Σ_{i-1}.*

Assumption 2. *Access to each simulator may be limited to running contiguous* trajectories *rather than having random access to a generative model or the model parameters.*

The first assumption states that each successively higher-fidelity simulator costs more to run per step than the one below it, but it is potentially not worth sampling every $\langle s, a\rangle$ at the lower level. The polynomial relationship enforces the fact that we still want to limit samples at the lower level. The second restriction states that we may not have access to the simulator parameters or the ability to sample state/action outcomes generatively. This is the case in the real world and in certain simulators (e.g. most commercial video games). In Section VII we provide an algorithm for MFRL without Assumption 2 restricting generative access to the simulators, but first the algorithm is presented and efficiency results are shown in the more restricted trajectory-only case.

Given such simulators, our objectives are the following:

1) Minimize the number of suboptimal learning samples (with respect to Q^*) taken in Σ_D.
2) Ensure that, for any run of the agent with simulator Σ_i, only a polynomial number of steps (in $|\Sigma_i|$) are taken before near-optimal behavior (given constraints from higher-fidelity simulators) is achieved or control is passed to a lower-fidelity simulator.
3) Guarantee that there are only a polynomial (in $|\Sigma_{1,\ldots,D}|$ and D) number of switches between simulators.

Objective 1 skews the sampling burden to lower-fidelity simulators while objective 2 limits the sample complexity of the algorithm as a whole. Objective 3 is included to prevent

excessive switching between the real-world robot and a simulator, as there may be significant start-up costs associated with the robot. This is particularly pertinent with dynamic robots that can not pause their actions to wait for a learning agent to update a model or policy based on simulated outcomes.

IV. MULTI-FIDELITY BANDIT OPTIMIZATION

One of the simplest RL settings where exploration is studied is the *k-armed bandit* case, an episodic MDP with a single state, k actions (called *arms*), and $\gamma = 0$. A learner must choose actions to explore the rewards, eventually settling on the best arm, which it then exploits. We now present a Multi-Fidelity RL algorithm for the bandit setting, which has several features of the full MFRL algorithm presented later.

A. A MF-Reward Learning Algorithm

Consider a chain of bandit simulators: at each level $d \in \{1, \ldots, D\}$ there are $|A|$ actions with expected rewards $R_d(a) \leq R_{d-1}(a) + \beta_d$. For a single simulator, we can utilize a *base learner* that can update the estimates of each arm's reward. Here, we use KWIK reward learners $L_{R,d,a}$ with parameter m based on the required accuracy parameters for a single-arm learner: $\bar{\epsilon}$ and $\bar{\delta}$. These quantities are related to the overall accuracy parameters in the learning scenario and are defined later in Theorem 1. Specifically, a KWIK learner for a single bandit arm a at a single level d can be created by keeping track of the number of times $c_{d,a}$ that arm has been pulled. The algorithm then predicts

$$\begin{cases} \hat{R}_d(a) & \text{if } c_{d,a} \geq m = \frac{1}{2\bar{\epsilon}^2} \log(\frac{2}{\bar{\delta}}) \\ \perp & \text{otherwise} \end{cases} \quad (1)$$

where the value of m is set based on Hoeffding's inequality [33], assuming that the rewards are bounded on $[0, 1]$.[1] When $\perp$ is predicted, a loose upper bound for the possible payout of the action is $R_{\max}$, which will be used in our algorithm below.

Algorithm 1 presents the Multi-Fidelity Bandit Framework (MF-Bandit) for accomplishing objectives 1-3 using a KWIK learner that keeps track of empirical reward means $\hat{R}_d(a)$, number of pulls $c_{d,a}$, and upper bounds on the rewards $\hat{U}_{d,a}$ at fidelity level d. MF-Bandit also tracks the *informed* upper bound $U_{d,a}$, which is the minimum of $\hat{U}_{d,a}$ and the heuristic from the lower level: $U_{d-1,a} + \beta_{d-1}$ (lines 20 and 25). The algorithm also keeps track of whether the value of each action has converged ($con_{d,a}$), whether an optimal action has been identified ($closed_d$), and if the learned model has changed ($change_d$) at simulator level d.

Starting in Σ_1, the algorithm selects an action a^* greedily based on U_d and checks if learning at the current level is complete (line 7). Before executing the action, it checks to make sure the action has been tried sufficiently many times at the simulator below (line 8). If not, control is returned to level $d-1$ where actions are selected using values from d that are converged (lines 11-15). Otherwise, if learning at d is not finished, the action is taken and L_R and $\hat{U}$ are updated (lines 17-22). Once the optimal action has been identified, the algorithm moves up to level $d+1$ (lines 23-26).

[1] We make this assumption for notational convenience. An extra term is added if the range is larger than 1.

Algorithm 1 Multi-Fidelity Bandit Framework

1: **Input:** A bandit simulator chain $\langle \Sigma, \beta \rangle$, Actions A, $R_{\max}$, Accuracy requirements ϵ and δ
2: **Initialize:** $\text{con}_{d,a}$, $\text{change}_d \coloneqq \text{false}, \forall a, d$
3: **Initialize:** KWIK learners $L_{R,d}(a, \bar{\epsilon}, \bar{\delta})$
4: **Initialize:** $d \coloneqq 1$, $\hat{U}_{d,a}$, $U_{1,a} \coloneqq R_{\max} \forall a$
5: **for** each timestep **do**
6: Select $a^* \coloneqq \text{argmax}_a \, U_{d,a}$
7: $\text{closed}_d \coloneqq \text{con}_{d,a^*} \vee a^*$ is definitely near optimal
8: **if** $d > 1 \wedge \neg\text{con}_{d-1,a^*} \wedge \text{change}_d$ **then**
9: // Return to level $d-1$
10: $\text{change}_{d-1} \coloneqq \text{false}$
11: **for** $a \in A$ **do**
12: **if** $\text{con}_{d,a}$ **then**
13: // Updated learner using $\hat{R}_d$ and $\hat{U}_d$
14: Set $L_{R,d-1,a}$ based on $L_{R,d,a}$
15: $\text{con}_{d-1,a}, \text{change}_{d-1} \coloneqq \text{true}$
16: $d \coloneqq d - 1$
17: **else if** $\neg\text{closed}_d$ **then**
18: Execute a^* in Σ_d, Observe r.
19: Update L_{R,d,a^*} // Update $\hat{R}_d(a^*)$, $\hat{U}_{d,a^*}$
20: $U_{d,a^*} \coloneqq \min(U_{d,a^*}, \hat{U}_{d,a^*})$
21: **if** L_{R,d,a^*} switched from $\perp$ to "known" **then**
22: con_{d,a^*}, $\text{change}_d \coloneqq \text{true}$
23: **else if** $d < D \wedge \text{closed}_d$ **then**
24: // Chosen action already converged, go up
25: Where $\neg\text{con}_{d+1,a}$: $U_{d+1,a} \coloneqq U_{d,a} + \beta_d$
26: $\text{change}_{d+1} \coloneqq \text{false}$, $d \coloneqq d + 1$

Algorithm 1 differs from unidirectional heuristic transfer [8] because it can backtrack to a lower-fidelity simulator when a previously identified optimal action performs poorly. In unidirectional transfer, information is only transferred from lower- to higher-fidelity levels, with no option for the learning agent to return to lower-fidelity levels to continue exploration. Effectively, backtracking asks the lower-fidelity learner to find a new optimal policy given additional knowledge from higher-fidelity simulators.

B. Bandit Examples

We now present examples to showcase various features of Algorithm 1. First, we show that MF-Bandit can find an optimal policy for Σ_D with far fewer samples in Σ_D than an algorithm without multiple simulators. Consider a bandit problem with $|A| = 5$ arms and $D = 3$ simulators with bounded reward $[0, 1]$. The rewards for each of the 5 actions in each simulator are

$$\Sigma_1 = \{0.8, 0.8, 0.8, 0.8, 0.1\}$$
$$\Sigma_2 = \{0.8, 0.8, 0.6, 0.6, 0.1\}$$
$$\Sigma_3 = \{0.8, 0.6, 0.6, 0.6, 0.1\},$$

all with uniform random noise up to 0.1. Table I(a) shows the results of running Algorithm 1 in this scenario with our KWIK bandit learner with $m = 20$. Results with only $\langle \Sigma_2, \Sigma_3 \rangle$ and only Σ_3 are also shown. Using both simulators or just Σ_2 produces a significant reduction in samples from Σ_3, and having Σ_1 helps limit the samples needed from Σ_2.

TABLE I: Samples used from the simulators

(a) Same optimal action in all levels

Sims Used	Σ_1	Σ_2	Σ_3
Σ_1, Σ_2, Σ_3	100	80	40
Σ_2, Σ_3	–	100	40
Σ_3	–	–	100

(b) Optimal action in Σ_3 is not optimal in Σ_2

Sims Used	Σ_1	Σ_2	$\Sigma_{3'}$
Σ_1, Σ_2, $\Sigma_{3'}$	100	80	60
UNI	100	60	80
$\Sigma_{3'}$	–	–	100

In the scenario above, the algorithm could potentially avoid backtracking because one of the optimal actions always remained the same at each level. But, consider the same scenario except with an alternate top level, $\Sigma_{3'} = \{0.4, 0.4, 0.6, 0.6, 0.1\}$. Now, neither of the optimal actions in Σ_2 are optimal in $\Sigma_{3'}$. Table I(b) shows the results of Algorithm 1 in this case along with a version that does no transfer and a version that only performs unidirectional transfer (UNI) [8]. We see here that, by allowing the algorithm to return to lower-fidelity simulators once the previously considered optimal action has been disproved at a higher level, valuable exploration steps in Σ_D are saved and the cost in terms of samples from the highest level is minimized.

C. Theoretical Analysis

We now formalize the intuition gained from the examples above in theoretical guarantees for the bandit case. Throughout these theoretical results we assume, without loss of generality, that the rewards of each arm are bounded on $[0, 1]$. We also assume the base learner is the KWIK bandit learner as described earlier that predicts $\perp$ as the output of an action where it does not have m samples and otherwise predicts the empirical mean $\hat{R}(a)$. Multi-state versions of most of these guarantees are presented in later sections.

We begin by focusing on objectives 2 and 3 from Section III-C: limiting the number of suboptimal actions at each level and the number of samples overall. The following theorem provides these sample complexity results as well as guidelines for setting $\bar{\epsilon}$ and $\bar{\delta}$ in (1).

Theorem 1. *Algorithm 1 uses only a polynomial number of samples over all the levels, specifically using only* $O(\frac{|A|D^2}{\epsilon^2} \log(\frac{|A|^2 D}{\delta}))$ *samples per run at level* d *and only changing* d *a maximum of* $|A|D$ *times.*

A proof of Theorem 1, as well as the rest of the theorems in the paper, is contained in the Appendix.

Now we turn our attention to objective 1 from Section III-C, minimizing the number of samples used in Σ_D. We begin this investigation with the following lemma, which is similar to Lemma 1 of [8], stating that no action is tried at a level beyond which it is dominated by the value of a^* in Σ_D.

Lemma 1. *In the bandit setting described above with actions* $a \in A$ *and levels* $1, \ldots, D$*, consider action* a *at level* d*. If* a *has been executed* m *times at level* d*, let* $\mu_d = \hat{R}_d(a)$*. Otherwise, set* $\mu_d = U_d(a)$*. If* $\mu_d < R_D(a^*_D) - \sum_{\bar{d}=d}^{D-1} \beta_{\bar{d}} - \epsilon$*, where* a^*_D *is the optimal action in* Σ_D*, then, with probability* $1 - \delta$*,* a *will not be attempted at or above level* d*.*

Algorithm 2 MFRL (MF-KWIK-Rmax)

Input: A simulator chain $\langle \Sigma, \beta, \rho \rangle$, $R_{\max}$, Planner P, accuracy parameters $\langle \epsilon, \delta, m_{known} \rangle$
Initialize: $\text{change}_d \coloneqq \text{false}, \forall d$
Initialize: $2D$ KWIK learners $L_{R,d}(\bar{\epsilon}, \bar{\delta})$ and $L_{T,d}(\bar{\epsilon}, \bar{\delta})$
Initialize: $\hat{Q}_0 \coloneqq \frac{R_{\max}}{1-\gamma}$, $\hat{Q}_1(s, a) \coloneqq \text{PLAN}(1)$
Initialize: $d \coloneqq 1$, $m_k \coloneqq 0$
for each timestep and state s **do**
  Select $a^* \coloneqq \text{argmax}_a \hat{Q}_d(s, a)$
  if $d > 1 \wedge \text{change}_d \wedge (L_{T,d-1}(\rho_{d-1}^{-1}(s), a^*) = \perp \vee L_{R,d-1}(\rho_{d-1}^{-1}(s), a^*) = \perp)$ **then**
    // Return to level $d - 1$
    $\hat{Q}_{d-1} \coloneqq \text{PLAN}(d-1)$
    $m_k \coloneqq 0$, $d \coloneqq d - 1$
  else
    Execute a^* in Σ_d, Observe r, s'.
    if $L_{R,d}(s, a^*) = \perp \vee L_{T,d}(s, a^*) = \perp$ **then**
      $m_k \coloneqq 0$
      Update $L_{R,d}$ and/or $L_{T,d}$ that predict $\perp$
    else
      $m_k \coloneqq m_k + 1$
    if $L_{R,d}(s, a^*)$ or $L_{T,d}(s, a^*)$ is now *known* **then**
      $\hat{Q}_d \coloneqq \text{PLAN}(d)$, $\text{change}_d \coloneqq \text{true}$
  if $d < D \wedge m_k = m_{known}$ **then**
    // Go up to level $d + 1$
    $\hat{Q}_{d+1}(s, a) \coloneqq \text{PLAN}(d+1)$
    $m_k \coloneqq 0$, $\text{change}_d \coloneqq \text{false}$, $d \coloneqq d + 1$
procedure PLAN(d)
  // Call planner P using highest-fidelity data available
  For any (s, a) let $d^*(s, a)$ be largest d such that $L_{R,d^*}(s, a) \neq \perp \wedge L_{T,d^*}(s, a) \neq \perp \wedge d^* \geq d$
  if d^* does not exist **then**
    $d^* \coloneqq d$
  $\hat{Q}_d \coloneqq P(\langle S_d, A, L_{R,d^*}.\hat{R}, L_{T,d^*}.\hat{T}, \gamma \rangle, Q_{d-1} + \beta_{d-1})$
end procedure

Now we show that only actions that *must* be tested in Σ_D are used there (objective 1 from Section III-C).

Theorem 2. *With probability* $1 - \delta$*, any action* a *attempted in simulator* Σ_D *(the real world) by Algorithm 1 is either near optimal (within* ϵ *of* $R_D(a^*)$*) or could only be shown to be suboptimal in* Σ_D*.*

Thus, Algorithm 1 never tries actions that can be pruned by lower-fidelity simulators. A corollary of this theorem is that, in the *worst case*, MF-Bandit uses no more samples in Σ_D than unidirectional transfer methods use.

V. MULTI-FIDELITY REINFORCEMENT LEARNING

We now instantiate the principles of generating heuristics from lower-fidelity simulators and sending learned model

data down from higher-fidelity simulators in the multi-state, cumulative discounted reward RL case.

A. The MFRL Algorithm

Algorithm 2 shows the MFRL framework, which takes as input a simulator chain, the maximum reward $R_{\max}$, a state-mapping between simulators $\rho_1, \ldots, \rho_{D-1}$, a planner P, and accuracy requirements ϵ, δ, and m_{known}. The algorithm begins by initializing the variables d, m_k and $change_d$ and the base KWIK learners $L_{T,d}$ and $L_{R,d}$, parametrized by $\bar{\epsilon}$ and $\bar{\delta}$. These KWIK learners are proxies for the learned transition and reward functions at level d, which we denote $L_{T,d}.\hat{T}$ and $L_{R,d}.\hat{R}$, respectively. More specifically, whenever a state/action pair is "known" (neither L_R nor L_T predict $\perp$), $\hat{T}$ and $\hat{R}$ model the predictions of the KWIK learners. If one of the learners predicts $\perp$, the Q-values from the previous level will be inserted as a heuristic to encourage exploration. The Q-values for the lowest-fidelity simulator are set optimistically using $\frac{R_{\max}}{1-\gamma}$, and the agent begins choosing actions at that level.

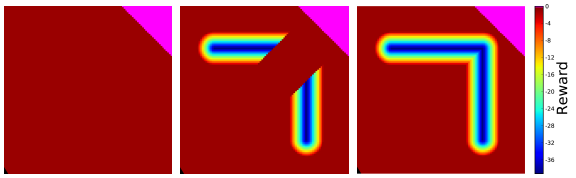
The agent chooses actions for the current state greedily. If, according to the KWIK model learners, the selected state/action pair is not known at level $d-1$, and a change has been made at the current level, the algorithm backtracks one layer of fidelity (lines 8-11). Otherwise, the action is executed and L_T and L_R at the current level are updated. Note that while backtracking after seeing a single state/action pair that is not known at level $d-1$ is theoretically correct, in our experiments we typically wait until $m_{unknown}$ such "unknown" state/action pairs are encountered, which helps control sampling at lower-fidelity simulators.

If the model parameters change, the $change_d$ flag is also set and the planner recalculates the Q-values using the PLAN subroutine (lines 25-31). Information from the highest-fidelity level that does not predict $\perp$ is used in the planner. If no such level exists, then the planner uses the current heuristic value (passed from a lower-fidelity level).

Finally, the convergence check (line 21) determines if MFRL should move to a higher-fidelity simulator. In the multi-state case, simply encountering a known state does not indicate convergence, as states that are driving exploration may be multiple steps away. Instead, Algorithm 2 checks if the last m_{known} states encountered at the current level were known according to the base learners. For theoretical purposes, we can set m_{known} to the following quantity, which is the number of steps needed to show that an MDP comprised only of the "known" states sufficiently models the environment (see Theorem 4 of [34]).

$$m_{known} = \frac{1}{1-\gamma} \ln \left(\frac{4\left(R_{\max} - R_{\min}\right)}{\bar{\epsilon}(1-\gamma)} \right) \qquad (2)$$

This quantity guarantees that, if the true value function is significantly different from the value of the current policy in the "known" MDP, with high probability an unknown state will be encountered during the run of m_{known} states. Further details about the theoretical properties are given in Theorem 3 below. In practice, a smaller value is usually adequate to check for convergence and move to a higher-fidelity simulator, but (2) can be used to ensure theoretical correctness.



(a) No puddle or noise (Σ_1) (b) Some puddle and noise (Σ_2) (c) Full puddle and noise (Σ_3)

Fig. 2: $\Sigma_1, \ldots, \Sigma_3$ for the puddle world. Σ_1 has no puddle. Σ_2 has most of the puddle, but the optimal policy in Σ_1 can bypass these puddle portions. Thus, the optimal policy in Σ_1 is still optimal in Σ_2, but not in Σ_3.

TABLE II: Parameters used for the puddle world results.

Sim	$\mathbf{m_{L_R}}$	$\mathbf{m_{L_T}}$	$\boldsymbol{\sigma}$	$\mathbf{m_{known}}$	$\mathbf{m_{unknown}}$	$\boldsymbol{\beta}$
Σ_1	1	3	0	75	–	0
Σ_2	1	4	0.01	75	20	0
Σ_3	1	5	0.02	75	20	–

B. Puddle World with MFRL

We illustrate the behavior of MFRL in a variant of the puddle world domain [35] with multi-fidelity simulators, shown in Fig. 2. A puddle world agent moves in one of four diagonal directions with a step cost of -1 (0 at the goal) and high negative rewards in the puddle. We implemented the puddle world with diagonal actions and $\gamma = 0.95$, so the optimal policy in Σ_3 is generally to skirt along the outer edges of the puddle, while the optimal policy in Σ_2 is to move diagonally between the puddles. The puddle world is 1 unit square and each step moves 0.1 units plus some zero mean Gaussian noise. For learning and policy evaluation, the world is discretized in the two dimensions into a 10 by 10 grid array.

We tested MFRL in the presence of two lower-fidelity simulators with respect to the "real" puddle world. The base level Σ_1 contains no puddle and has deterministic actions. The middle level Σ_2 contains some of the puddle and has noisy actions. This creates a scenario where an optimal policy in the low-fidelity simulator is poor in Σ_3 but still contains significant useful information, such as the puddle portions in Σ_2 and the goal location. The top level Σ_3 contains the full puddle and the full action noise.

Fig. 3 shows learning curves from this experiment with Table II showing the parameters used during the experiments. Here, m_{L_T} and m_{L_R} denote the number of times a state/action pair must be observed before the transition and reward functions, respectively, are known. MFRL is compared with unidirectional transfer (UNI), no-transfer (RMAX) and prioritized sweeping (PS) [1]. The m_{L_T} and m_{L_R} parameters were the same in each of the three Rmax-based algorithms, ensuring a consistent comparison. In PS, the agent explores using an ϵ-greedy policy ($\epsilon = 0.1$) and optimistic initialization while evaluation occurs greedily.

MFRL performed the best, with some negative transfer at the beginning from the "shortcut" in Σ_2. As the learning agent encounters the real puddle in Σ_3, it starts exploring areas of the state-action space that where not explored in Σ_2. This exploration results in several level changes where

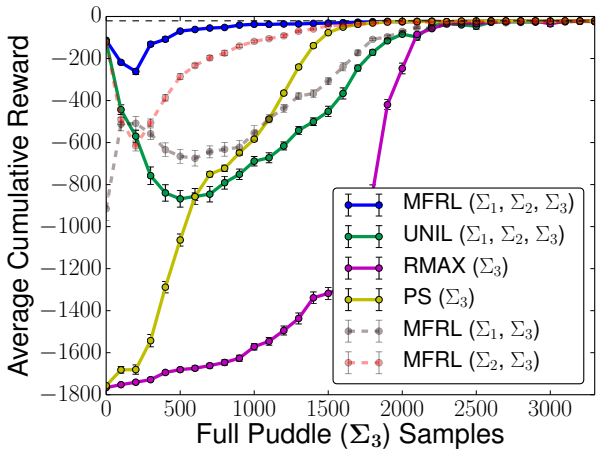
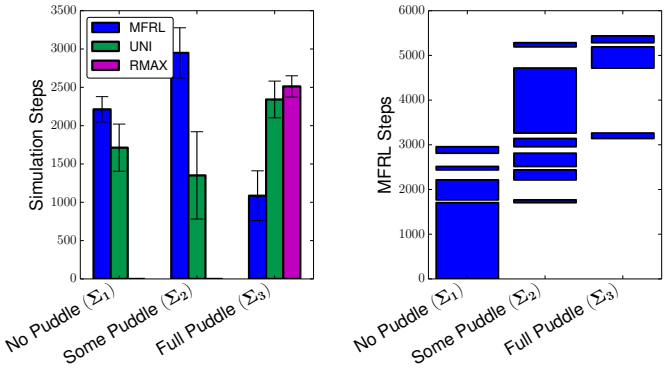


Fig. 3: During learning, MFRL consistently outperforms unidirectional transfer (UNI), no-transfer Rmax (RMAX), and prioritized sweeping (PS) at Σ_3. Note that these are only the samples from Σ_3. Each point is an average of 1000 learning runs (standard errors shown). Greedy policies are evaluated 60 times, each capped at 600 steps.



(a) Average Samples

(b) Example MFRL algorithm run

Fig. 4: (a) At the top level, MFRL requires fewer than half the samples needed by both unidirectional transfer (UNI) and no-transfer Rmax (RMAX). This is accomplished by transferring some learning burden to the other simulators. Each bar represents an average of 1000 learning runs with standard deviations shown. (b) An example run of the MFRL algorithm. After initially exploring the top level, the algorithm spends a significant number of samples exploring the rest of the middle level, thus decreasing the required samples at the top level.

the learning agent in Σ_2 plans using information from Σ_3, causing that lower-fidelity agent to find a way around the puddle. The result is a consistent and significant improvement over unidirectional transfer throughout learning. Note that backtracking from higher- to lower-fidelity levels occurs only when there is still uncertainty at the lower-fidelity levels. If no uncertainty exists at the lower-fidelity levels, then returning to those levels will not yield better Q-values to guide exploration at the higher-fidelity levels.

MFRL also outperforms naïve model-based RL algorithms such as PS despite PS performing better than Rmax alone. Fig. 3 also shows that even in the presence of only one simulator (MFRL (Σ_1, Σ_3) and MFRL (Σ_2, Σ_3)), MFRL still outperforms standard Rmax.

If Σ_2 and Σ_3 were switched so that Σ_2 is the "real" world, the optimistic chain assumption would be violated. In this case, the learning agent would explore all of the puddle in Σ_3, converging to a policy that moves around the puddle towards the goal. When the agent transitioned to Σ_2, it would not explore and find the "shortcut" between the puddles, but instead continue with the optimal policy from Σ_3. Despite not converging to the true optimum, a good policy in Σ_2 would be found with very few steps needed from that level.

The improvement of MFRL over unidirectional transfer is accomplished while using more than 50% fewer samples in the top level as seen in Fig. 4(a). In fact, unidirectional transfer takes almost as long as the no-transfer case to consistently find the optimal policy, primarily because even with unidirectional transfer, the learning agent still needs to explore the majority of the state/action space in Σ_3 before finding the path around the puddle. Fig. 4(b) shows a single run of MFRL with bars showing samples in each of the 3 simulators. The MFRL agent relied heavily on Σ_1 initially and swapped back to Σ_2 several times while learning at Σ_3, gathering crucial information through this lower cost simulator.

C. Theoretical Analysis

We now formally quantify the behavior of Algorithm 2 by extending the theoretical results from the bandit case. Throughout this section we make the standard assumption, without loss of generality, that the rewards of the MDP are in the range $[0, R_{\max}]$. We also set m_{known} by (2) and instantiate the accuracy parameters of the KWIK MDP learners as

$$\bar{\epsilon} = \frac{\epsilon}{4(D+1)}, \qquad \bar{\delta} = \frac{\delta}{4(D+2D|\Sigma|)}. \tag{3}$$

where $|\Sigma|$ is the maximum number of parameters used to describe a simulator in the optimistic chain. Typically, $|\Sigma|$ is the number of parameters representing the transition and reward functions in the real-world MDP. Furthermore, we assume throughout this section that the variables in each Σ_i are the same; that is, ρ_i is the identity mapping. We return to this assumption at the end of the section and discuss how β_i may need to be increased to preserve the stated properties when some simulators have missing variables.

It should be noted that, like all algorithms in the Rmax family, the sample complexity bounds presented here are meant to show the scalability of the algorithm with respect to problem size, not for actually setting the known-ness parameters in practice. That is, the analysis shows that as problems grow larger (with respect to $|S|$, $|A|$, ϵ, or δ), the number of samples needed increases only polynomially in those terms. However, because of the loose bounds used, practitioners almost always choose far lower values for known-ness based on the noisiness and risk in their environments. It is likely that setting known-ness parameters lower in this way

invalidates the theoretical worst-case sample complexity, but in most problems these algorithms have proven remarkably robust to these lower values. Recent theoretical results [36] indicate that tighter bounds can be achieved, supporting these more aggressive known-ness parameters.

We begin by analyzing the sample complexity of a run at each level of simulation and the total number of level changes in the MFRL algorithm. As with Theorem 1, this Theorem covers objectives 2 and 3 from Section III-C.

Theorem 3. *Algorithm 2 uses only a polynomial number of samples in* $\langle |\Sigma|, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma} \rangle$ *per run at level* d *and only changes* d *a maximum of* $(D + 2D|\Sigma|)$ *times.*

While Theorem 3 covers the convergence and sample complexity of MFRL, the worst case bounds in the theorem above may require an agent to do the same amount of learning in Σ_D as an agent without any simulators. This is necessary because in the worst case the simulators provide no useful information to the real world. To understand the more general theoretical benefits of MFRL and its relationship to the unidirectional transfer method, we expand Lemma 1 and Theorem 2 to the multi-state case, covering objective 1: limiting steps in Σ_D.

The following extension of Lemma 1 shows that, for a given state s, if an action's Q-value is definitely dominated by the optimal action at s, with high probability the action will not be attempted at the current or higher-fidelity levels.

Lemma 2. *Consider state* s *and action* a *at level* d *and let* $\mu_d = \hat{Q}_d(s,a)$ *if* $\hat{T}(s,a)$ *and* $\hat{R}(s,a)$ *are known. Otherwise,* $\mu_d = Q_{d-1}(s,a)$*. If* $\mu_d < Q_D(s, a^*_D) - \sum_{\bar{d}=d'}^{D-1} \beta_{\bar{d}} - \epsilon$ *where* a^*_D *is the optimal action for* s *in* Σ_D*, then, with probability* $1-\delta$*,* a *will not be attempted in* s *at or above level* d*.*

Unlike the bandit case, the lemma above does not translate directly into a guarantee on the *necessity* of an action in Σ_D because the KWIK-Rmax algorithm is not guaranteed to be the most efficient exploration algorithm possible. It does, however, guarantee that every suboptimal step is, with high probability, leading towards a learning experience (see Lemma 13 of [7] and Theorem 4 of [34]). Therefore, we can state the following property of Algorithm 2 based on the lemma above and this guarantee of a future learning experience.

Property 1. *With probability* $1-\delta$*, any action* a *attempted in state* s *of simulator* Σ_D *(the real environment) by Algorithm 2 is either near optimal (within* ϵ *of* $V^*_D(s)$*) or will eventually lead to an unknown state that is either not learned about in levels below or that needs to be learned about in* Σ_D*.*

Thus, Property 1 means that MFRL tries actions in the real world that either lead to needed exploration or backtracking to a lower-fidelity level. Property 1 also means that MFRL will, with high probability, enter no more unknown states in Σ_D than a unidirectional transfer method with the same base learner and architecture. Both approaches will be drawn to areas that fit the two cases above. However, by returning to Σ_{D-1}, MFRL can potentially learn about areas that were not visited earlier in Σ_{D-1} and thereby prune actions as in Lemma 2. By contrast, the unidirectional case can only learn about such areas in Σ_D. Because of the optimistic chain assumption, obtaining data from the lower-fidelity simulator can only strengthen the heuristic and prune more actions. Therefore, in the case where MFRL returns to Σ_D in the exact state it was in before it decided to use the simulators, MFRL will make no more (worst case) sub-optimal steps than the unidirectional approach with the same base learners.

However, in cases where the agent "resets" to the start state of Σ_D upon returning from the lower-fidelity simulators, it is possible for MFRL to make more suboptimal steps than the unidirectional algorithm because it needs to retake potentially suboptimal steps to reach the state it was in when it decided to return to the lower-fidelity simulators. However, this increase in suboptimal steps is at most a multiple of the polynomial number of possible entries into the simulator (covered in Theorem 3) and will usually be offset by better information gained in the lower simulators.

D. Properties of MFRL with Missing Variables

We now return to the assumption, made in Section V-C, that each Σ_i contains the same set of variables. In cases where a variable v exists in Σ_i but is missing in Σ_{i-1}, one value of the variable is designated as the *default* value and only parameters learned in states with this value can be used by the planner in Σ_{i-1}. For instance, in our RC car, if a simulator is missing the "wheel slip" variable, only non-slip dynamics should be used by the planner in that simulator. However, because ρ_i is potentially one-to-many from Σ_{i-1} to Σ_i, the Q-values passed *up* to Σ_i could cause an undervaluation of some states in Σ_i.

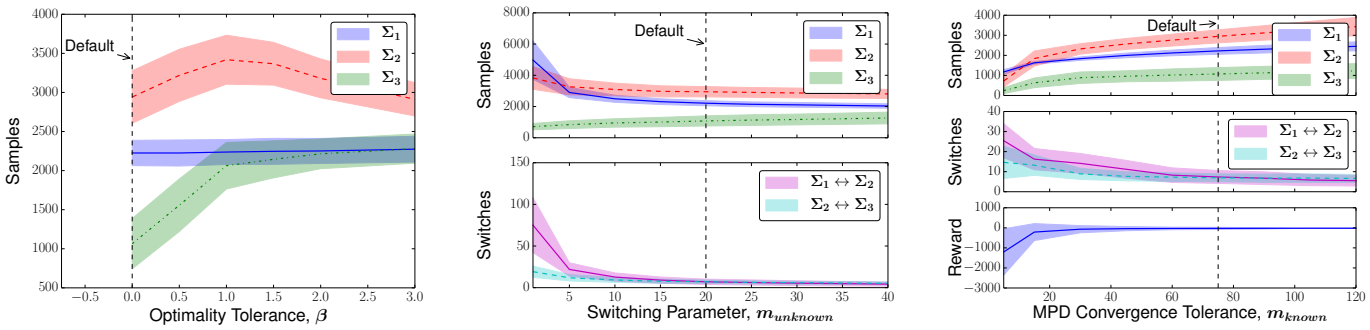
Consider the wheel-slip example with states s_0 and s_1 in Σ_i where s_0 has no slip and s_1 has slip. Suppose $V^*(s_1) = 1$ and $V^*(s_0) = 0$; that is, the slipping state is more valuable. If the agent experiences s_0 first and then returns to Σ_{i-1}, s_0 in Σ_{i-1} may set $V^* = 0$, the value in the simulator above. Now, when the agent returns to Σ_i, 0 will be used as the heuristic value for s_1. Unfortunately, this is an *underestimate* of the value function in the slippery state, invalidating the KWIK-Rmax assumptions.

However, MFRL has machinery to compensate for such undervaluation. Specifically, β_{i-1} can be used to increase the transferred heuristics in this case. All of the properties described above hold in the case of missing variables as long as β_{i-i} is set high enough at each level to guarantee both Definition 2 *and* that $Q^*_i(s_1, a) + \beta_{i-1} > Q^*_{i-1}(s_0, a)$ where s_1 and s_0 are states in Σ_i that were aliased in Σ_{i-1}.

In summary, missing variables in some of the simulators add complications to the algorithm's exploration strategy, but they can be overcome by utilizing the existing β_i parameters already built into the framework. Future work will investigate the effects of a non-unity ρ_i mapping in greater detail.

E. Sensitivity of MFRL Parameters

In this section we demonstrate how the input parameters β_i, m_{known}, and $m_{unknown}$ affect the performance of Algorithm 2. Fig. 5 shows the performance of the algorithm in the puddle world domain as each of the three parameters is varied from its nominal value. The nominal values used in the domain are $\beta = \beta_1 = \beta_2 = 0$, $m_{known} = 75$, and $m_{unknown} = 20$.



(a) Varying optimality tolerance (b) Varying switching parameter (c) Varying convergence parameter

Fig. 5: The effects of varying the parameters of the MFRL algorithm as demonstrated in the puddle world domain. Bold lines denote means and shaded areas show one standard deviation. Please refer to the text for details.

Each data point in the graphs is an average of 1000 runs, with the shaded regions denoting one standard deviation.

In Fig. 5(a), the simulator tolerance β is increased significantly from the true value of $\beta = 0$. The algorithm converges to the optimal policy for all values of β; however, as the tolerance is increased, the number of simulator steps at Σ_D increases since the algorithm is unable to prune as many suboptimal state/action pairs. Eventually, Σ_3 no longer gains any information from the simulators below it and the number of samples needed to converge at Σ_3 quickly approaches the number needed to converge at Σ_1. However, even with an inaccurate β value, the MFRL algorithm converges to the correct policy using no more top-level samples than had learning been performed at the top level alone.

Fig. 5(b) shows how the number of samples used at the top level increases with increasing $m_{unknown}$. As with changing β, the MFRL algorithm converged to the optimal policy for all values of $m_{unknown}$. For the theoretically correct value of $m_{unknown} = 1$ (see Section V-A), the number of samples used at Σ_3 is minimized, but at the expense of a high number of simulator switches and a large number of samples used at the other simulator levels. Setting $m_{unknown}$ determines a trade-off between minimizing the samples at Σ_3 and minimizing the number of simulator switches. For many robotic scenarios it may be advantageous to set $m_{unknown} > 1$ if the start-up costs of running the robot are significant. Also, setting $m_{unknown}$ low for a particular expensive simulator can help limit the number of samples needed from that level.

Finally, Fig. 5(c) shows the algorithm performance as the convergence parameter m_{known} is varied. Notice that setting m_{known} too low causes the algorithm to converge prematurely to a suboptimal policy, while setting the value too high wastes unnecessary samples converging to the same policy.

Empirically, we see that the MFRL algorithm is robust to variations in the tuning parameters of β, $m_{unknown}$, and m_{known}. The algorithm converges to the optimal policy at the top level for all values of the tried parameters, except for when m_{known} is set artificially too low. While no firm guidelines exist yet for determining these parameters for new domains, in general, β should be set based on approximately how optimistic two adjacent simulators are, m_{known} based on how many successive "known" samples should be observed before being confident that further exploration is not needed, and $m_{unknown}$ based on how costly it is to backtrack to a lower-fidelity simulator.

VI. More General Representations through the KWIK Framework

Thus far, while our analysis of MFRL's sample complexity has been done through the general KWIK framework, we have focused mostly on tabular representations of T and R. However, the KWIK framework allows model-based RL agents to employ a number of more general representations that scale polynomially efficient learning to larger environments.

The mechanism for performing such scaling is to use a representation of T and R with far fewer parameters than $|S|$. Many such representation classes have been analyzed within the KWIK framework [7], including linear systems [37], "typed" Gaussian-offset dynamics [38], Dynamic Bayesian Networks (DBNs) [7], and Gaussian processes [39]. Complex robotic motion may also benefit from stronger relational representations such as Object Oriented MDPs [40]. Since our theoretical analysis was done for any KWIK learnable representation, the algorithm and efficiency guarantees hold for these representations as well. When continuous representations are used, Q is approximated with a function approximator. The approximator parameters are passed from lower- to higher-fidelity levels to construct optimistic values. Note that not all systems are KWIK learnable [7] (e.g., the conjunction of n terms).

In the following sections, we describe one of these general representations, a DBN, in more depth and illustrate how it can be used in the MFRL framework in the puddle world simulators from Section V-B.

A. Dynamic Bayesian Networks

A Dynamic Bayesian Network [9], [41] represents the evolution of a set of factors F from one step to another. A DBN can be viewed as a graphical model containing $2|F|$ nodes representing the value of each factor at time t and time $t+1$. Each factor f at level $t+1$ is connected to a set of *parent* nodes $\Pi(f)$ at level t.[2] The assumption leveraged in

[2]Cross-Edges within the $t+1$ level are also allowed as long as they do not create cyclic dependencies.

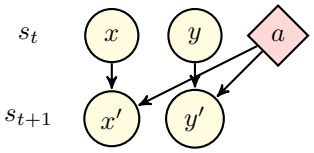
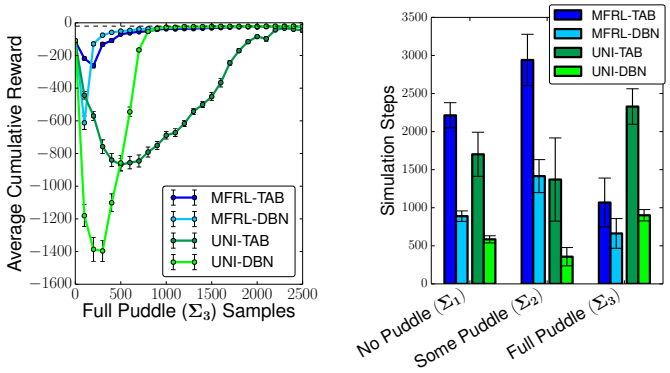


Fig. 6: A Dynamic Bayesian Network representation of the transition dynamics for the puddle world domain introduced in Section V-B. Each state consists of an x and a y grid location. These dimensions are independent given the current action, thus reducing the fully connected tabular representation to one with 2 factors.



(a) Learning curves (standard errors shown)

(b) Samples used (standard deviation shown)

Fig. 7: Using the Dynamic Bayes Network (DBN) shown in Fig. 6 to represent the transition function for the puddle world decreases the top-level convergence time and overall samples used. (a) However, the more generalized representation function also transfers more negative information than the tabular representation (TAB) as seen by the initial dips in average reward. (b) The total samples used across the three simulator levels is decreased by an average of 52% and 66% for the MFRL and unidirectional transfer (UNI) cases, respectively.

a DBN is that the probability distribution over f's value is independent of the probability of any other factor value given $\Pi(f)$. Thus, the probability of f taking on a given value can be encoded in a small probability table that grows exponentially only in the number of parents. In this work we use tabular representations of the conditional probability distributions, but any non-tabular, KWIK compatible representation could also be used.

DBNs can naturally be used to represent an MDP transition function for a factored state space by representing the factors of s and a in the top level and the factors of s' in the bottom level (with dependencies on the variables of s and a). Then, based on the independence assumption of the DBN we have $T(s,a,s') = \prod_{f \in F} Pr(f(s')|\Pi(f|s,a))$.

A DBN over n factors, each with v values, contains only $n(v-1)v^{|\Pi|}$ parameters, where $|\Pi|$ is the maximum number of parents of any node. Thus, the DBN is KWIK learnable given the structure Π using the KWIK-Rmax framework.[3] Specifically, a parameter $m = 2v/\epsilon^2 \ln(2v/\delta)$ is set, and, for a given $\langle s,a \rangle$, if any factor induces a $\Pi(f|s,a)$ with fewer than m experiences, $L_T(s,a) = \perp$, otherwise the maximum likelihood distribution given the current data is used for T. Combining this partitioning of known and unknown factor combinations with the $R_{\max}$ heuristic yields agents that efficiently target factor combinations that are unknown but may lead to better behavior, without exploring the entire ground state space.

In MFRL, we use the generic Algorithm 2 with the base KWIK-DBN learners described above, but instead of the $R_{\max}$ heuristic, we set $\hat{Q}_i(s,a)$ for any s that induces an unknown parent configuration (that is, $L_T(s,a) = \perp$) to $\hat{Q}_{i-1}(s,a)$.

B. DBNs for the Puddle World

States in the puddle world domain described in Section V-B consist of the x and y coordinates of the grid containing the current location of the agent. When the agent takes a step, the x coordinate of the new grid depends only on the previous x grid coordinate, and not on the previous y grid coordinate. Similarly, transitions between y grid values depend only on the current y values. Therefore, the transition dynamics of the puddle world agent can be captured by the DBN model shown in Fig. 6, where the next x and y state values depend only on the previous action and previous x or y state value, respectively. It should be noted that this DBN representation is not the most efficient way to represent or solve the puddle world domain, but is merely an example of how DBN's can be used to increase the speed of learning.

Fig. 7 shows average learning curves at the top level and average samples at each of the three levels using Algorithm 2 with the DBN extensions described in Section VI-A. Improvement over the tabular transition function representation is evident as more than 50% fewer samples are needed in the three levels when using the DBN.

One unanticipated effect of using the more general representation is the increased negative transfer from Σ_1 and Σ_2 to Σ_3, as demonstrated by the more substantial initial decrease in average reward during learning shown in Fig. 7(a). The increased negative transfer is due to the increased generalization of the transition dynamics. When exploring Σ_3, the agent is less likely to initially reach states unobserved at Σ_2 because the DBN generalizes each experience to many more states than a tabular representation would. Thus, the DBN agent will explore more of the puddle in Σ_3 before moving back to Σ_2 and Σ_1. A detailed analysis of the effects of transition and reward dynamics generalization on the amount of negative transfer in MFRL is outside the scope of this paper and will be saved for future work.

VII. Generative Simulators

Thus far, we have conducted our analysis and experiments under Assumption 2, which states samples can only be obtained by executing trajectories. While this is certainly true in the real world and may be true in many high-level simulators with little API access (such as a commercial video

[3] KWIK algorithms also exist for learning this structure. See [7].

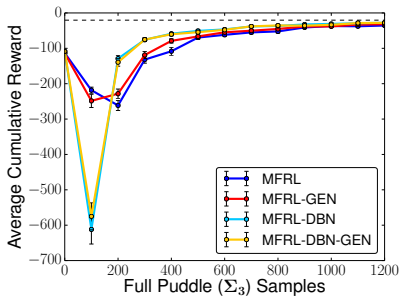


Fig. 8: Using generative access in the puddle world domain doesn't significantly improve learning performance. Other simulation environments could benefit more from generative access to the simulation environments. Learning curves are the average of 1000 runs with standard errors shown.

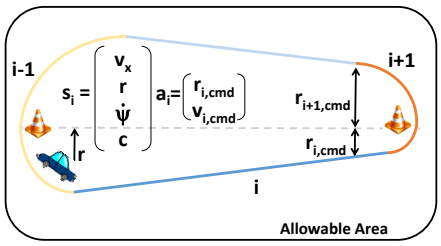


Fig. 9: The RC car task consists of driving around a fixed-length track. Each lap is divided into four segments and the car must choose both the commanded velocity and commanded turn radius of each segment upon completion of the current segment. These decisions are made based on the state values at the end of each segment, namely the car forward velocity, current radius, rotational velocity, and segment type.

game or complex simulator), often this assumption may be too restrictive. Many simulators provide *generative* access to dynamics and reward information, where arbitrary state/action pairs can be queried.

We propose a simple extension to the MFRL algorithm when one or more simulators provide generative access. When moving to a lower-fidelity simulator based on a set $\bar{S}$ of states unknown at level $d-1$, if Σ_{d-1} is generative, the algorithm runs greedy trajectories starting directly at the states in $\bar{S}$ to gather samples for its model of Σ_{d-1}. Once all of the states in $\bar{S}$ are known, it performs the normal greedy trajectories in Σ_{d-1} as specified in the original MFRL algorithm.

Fig. 8 shows the result of applying Algorithm 2 to the puddle world domain described in Section V-B, where generative access is assumed at all levels of the domain. Generative access does little to improve the convergence of the algorithm in the real puddle world or with the RC car in the next section. However, we believe other domains might benefit more from generative simulator access.

VIII. RC CAR RESULTS

A. Experimental Setup

The MFRL algorithms are experimentally verified using an RC car driving on an indoor track. The car is an off-the-shelf

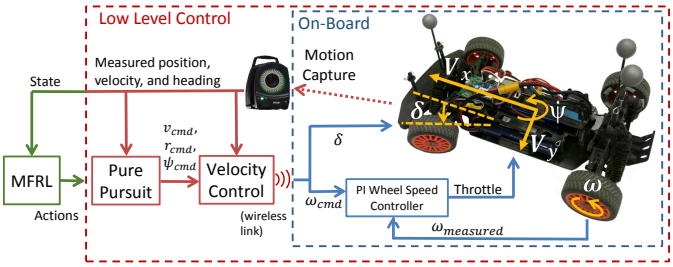


Fig. 10: One of the benefits of the MFRL framework is the ability to wrap the learning around fixed lower-level controllers. The car is controlled using a series of closed-loop controllers that perform high-rate velocity and position tracking, while MFRL learns which high-level commands to issue at a much slower rate. The RC car and associated variables are also shown. The linear velocities V_x and V_y are in the body-frame. The steering angle is δ, the wheel velocity is ω, and the heading rate is $\dot{\psi}$.

rally car shown in Fig. 10. The position, velocity, and heading angle of the vehicle are measured using an external motion capture system. The wheel velocity is measured and filtered using an optical encoder read by a microcontroller.

Fig. 9 shows the car task, which consists of selecting different radii and velocities to minimize lap-times on a track of fixed length. In this scenario, the track consists of straight and curved segments, each with an associated distance and velocity parameter, both of which remain constant during the length of each segment. As shown in the diagram, the virtual "cones" are fixed and denote the length of the path. As the car finishes each segment, the commanded radius $r_{i,cmd}$ and velocity $v_{i,cmd}$ values for the next segment are chosen. The reward returned for each segment is $-t$ where t is the elapsed time required to drive that segment. If the car drives, or often slips, out of a virtual "drivable" area around the track (denoted by the large black box in Fig. 9), the car resets to a fixed initial condition and is given a large negative reward. The state variables in s_i (Fig. 9) are the body frame forward velocity V_x, rotational rate $\dot{\psi}$, distance from track center r, and the current segment type c (straight or curved).

Choosing the next radius and velocity fully defines the desired path for the next segment (note that straight and curved track segments are forced to alternate). The car follows this path using a pure pursuit controller where the look ahead control distance is a function of the commanded velocity [42]. Running at 50 Hz, the pure pursuit controller computes the desired forward velocity, rotational rate and heading angle required to keep the car on the specified trajectory. Steering angle commands and desired wheel velocities are computed using the closed-loop controllers from [43], where the cross track error term in the steering angle control law is omitted, as the cross track error is reduced by the pure pursuit algorithm. The C_y parameter in this control law (see Equation 8 in [43]) is found by matching measured vehicle data to input commands.

The steering angle δ and commanded wheel speed ω_{cmd} are sent to the car's microcontroller. Steering commands are sent directly to the servo. Commands to the motor come from

a simple closed-loop controller around the commanded and measured wheel speed. This proportional-integral wheel speed controller is used to lessen the effects of the time-varying battery voltage on the velocity dynamics.

An overview of the control loops used to control the vehicle is shown in Fig. 10. One of the strengths of the MFRL framework is that it can be used around existing closed-loop controllers. These closed-loop controllers need not be the same at the different levels of simulation.

The simulation environments for the RC car consist of a naïve simulator (Σ_1) and a dynamics-based simulator (Σ_2). The naïve simulator ignores the dynamic model of the car and returns ideal segment times assuming the car followed the requested trajectory exactly. As mentioned in Section III-C, this simulator does not model rotational rate $\dot{\psi}$ and so the state-space mapping is not 1-to-1 when going from Σ_1 to Σ_2. The higher-fidelity simulator models the basic dynamics of the car, including wheel slip, where model parameters such as the "Magic Tyre Parameters" [44] are estimated using test data collected on the car. This simulator captures much of the dynamic behavior of the car, although discrepancies in real-world data and simulator data become more significant at higher velocities (above about 2.0 m/s) and when the wheels slip significantly. Therefore, learning needs to be performed not only in the simulators, but also on the physical car.

Both car simulators (Σ_1 and Σ_2) run many times faster than real time. Therefore, the total learning time in the experiments below is dominated by the time spent collecting samples from the real car. As mentioned earlier, the known-ness (m and m_{known}) parameters for the base learners in these experiments are set lower than the loose worst-case bounds from the theoretical section, which is standard practice with the Rmax family of algorithms.

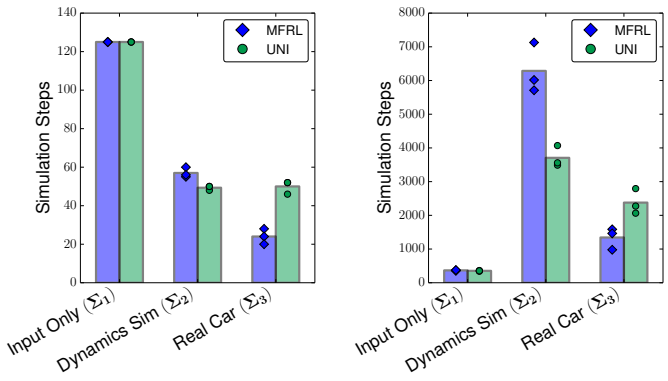
B. Experiments for the Bandit Setting

The first RC car experiment takes place in the bandit setting: choosing a single radii and two velocities (one for curves and one for straightaways) at the beginning of a 3-lap run. We allowed 5 values for radii (between 0.5 and 1.2 m) and 5 values for velocities (between 2.0 and 3.5 m/s), yielding 125 actions/arms in the bandit scenario. We evaluated Algorithm 1 in this setting as well as unidirectional transfer of Q-value heuristics, that is, the transfer mechanism of [8] applied with the KWIK-Rmax framework.

The simulators are deterministic (Σ_2 has only a small amount of added artificial noise) and so we set $m = 1$ at Σ_1 and Σ_2. To account for real-world noise, we set $m = 2$ at Σ_3, meaning 2 tries with a given parameter setting were needed to determine its value. Both MFRL and unidirectional transfer found a near-optimal policy within 60 steps on the real car. We did not compare to a no-transfer algorithm since it would need at least 250 trials to identify the optimal policy[4].

Fig. 11(a) depicts the samples used in each simulator by the algorithms. MF-Bandit on average uses fewer than half as many samples in the real-world when compared to the unidirectional learner. Both MF-Bandit and unidirectional transfer converged to policies with lap times of about 3.7 seconds per lap and learned to use higher velocities on the straightaways than the curves. These lap times are similar to the ones found in the state-based setting described in the next section, although the state-based policy is more robust to disturbances and noise.

[4]While there are only 125 actions, each action must be tried $m = 2$ times in the real world before it is known.



(a) Bandit (b) State-based

Fig. 11: Samples used by MFRL and unidirectional transfer (UNI) at each level in the (a) bandit case and the (b) state-based case. In both experiments, the MFRL algorithm converges to an equally fast policy when compared to unidirectional transfer; however, MFRL uses over 50% fewer real-world samples in the bandit case and over 30% fewer in the state-based case. Each algorithm is tested 3 times with bars showing the average.

TABLE III: State and action discretizations for the state-based car MFRL results.

Value	Min	Max	Disc. Small	Disc. Large
c (segment)	–	–	2	2
$\dot{\psi}$ (rad/s)	-1.0	3.5	3	3
r (m)	0.5	1.2	3	4
v_x (m/s)	2.0	3.2	3	6
r_{cmd} (m)	0.5	1.2	3	4
$v_{x,cmd}$ (m/s)	2.0	3.2	3	6

TABLE IV: Parameters for the state-based car MFRL results.

Parameter	Σ_1	Σ_2	Σ_3
m_{L_R}	1	1	1
m_{L_T}	1	3	3
m_{known}	100	100	50
$m_{unknown}$	–	10	10
β	0	0	–

C. Experiments for the State-Based Setting

In the multi-state case, we used the state space described in Section VIII-A and Fig. 9 and allowed the car to pick a radius and velocity at the beginning of every segment (4 per lap). Because we are making closed-loop, state-based decisions (i.e. changing velocities and radii in short segments), we can reduce the action-space from the bandit setting, since $|\Pi| = O(|A|^{|S|})$. Here we used 3 radii and 3 velocities ($|A| = 9$). The discretization of the state and action values is shown in

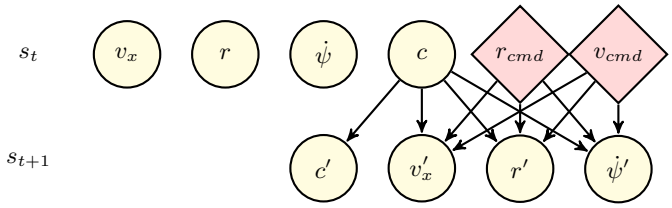
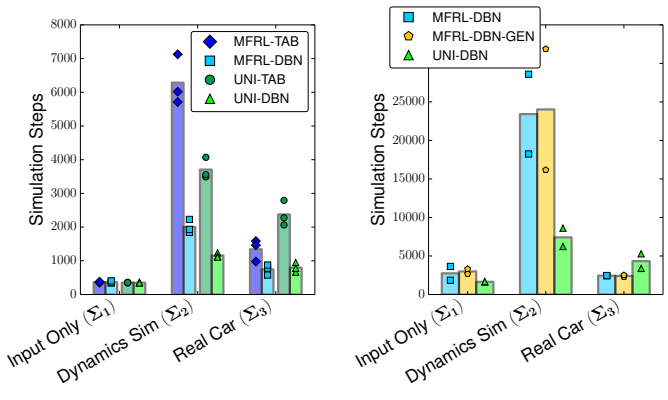


Fig. 12: A Dynamic Bayesian Network representation of the transition dynamics for the RC car. The next radius, forward velocity, and rotational rate depend primarily on the commanded radius, commanded velocity and the current segment. The next segment depends only on the current segment as the car is forced to go from straight to curved to straight and so forth. Note that this transition model decreases the maximum factor order from 6 to 3.



(a) DBN (3 runs) (b) DBN Large (2 runs)

Fig. 13: (a) Using the Dynamic Bayes Network (DBN) shown in Fig. 12 to represent the transition function of the car decreases the total samples used across the three levels of learning by an average of 63% for both the MFRL and unidirectional transfer cases. (b) With the larger domain specified in Table III, the MFRL algorithm decreases the samples used on the real car by an average of 40% when compared to a unidirectional transfer case. Note that the size of the state/action space here is over 3400, but the MFRL algorithm converges in fewer than 2500 samples on the real car.

Table III (small problem). Since the discretization in the state space makes the simulation results potentially noisier (from state aliasing), we used $m_{L_T} = 3$ in Σ_2 and the real car. The MFRL parameters used for these results are shown in Table IV.

In the experiment, both MFRL and unidirectional transfer converged to a policy with lap times of just under 3.7 seconds, slightly faster than the bandit results from the previous section. Fig. 11(b) shows the samples used in each level by MFRL and unidirectional transfer. The MFRL algorithm converges using an average of 35% fewer samples in the real world when compared to unidirectional transfer.

The converged policy in the state-based experiments is different from the bandit case, due to the versatility of state-based control. Instead of an oval shape, MFRL chose different values for the radius entering a curve versus the radius exiting the curve. This led to initially wide turns towards the cones followed by a sharp turn towards the straightaway, maximizing the time the car could drive fast down the straight section. Reaching this fairly complicated policy with a reasonable number of real-world samples was made possible by MFRL's efficient use of samples from the previous levels. Particularly, experience in Σ_1 pruned policies that were too slow, while experience in Σ_2 pruned policies that were too fast on the curves. Finally, experience with the real car refined the policy under the actual, but noisier, conditions in the real environment.

Videos showing the MFRL algorithm in the puddle world and on the RC car are available at http://goo.gl/ZOX4hX and http://ieeexplore.ieee.org.

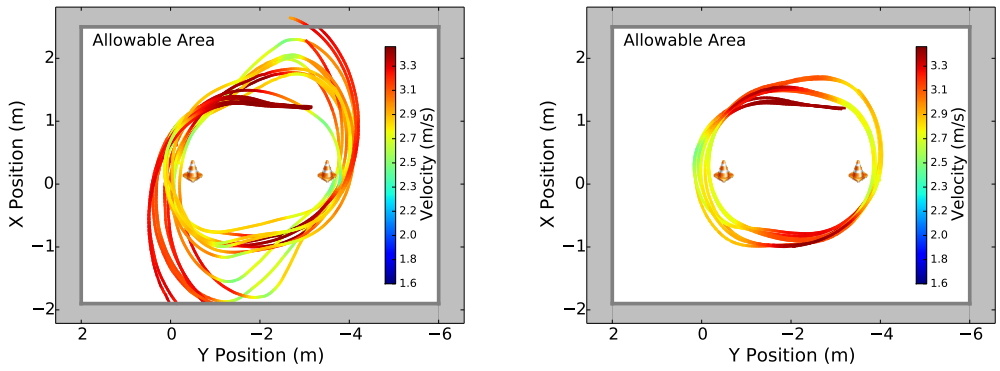
D. Experiments using a DBN Representation and Generative Access

In this section we use the notion of a generalized transition function via a DBN introduced in Section VI-A to decrease the required real-world samples needed to solve the track problem. The independence assumptions made are shown in Fig. 12. Because the closed-loop controllers guiding the car are fast relative to the length of the track segments, we assume that the state values at the end of a segment depend only on the segment type and the actions chosen for that segment, and not on the state values when the segment was started. There are cases when this assumption is violated, such as when the car starts sliding out of control in one segment before transferring to the next segment; however, for most of the state/action space and our testing, this assumption holds true.

Fig. 13(a) shows the samples used to solve the track problem using the small discretization values shown in Table III. The DBN transition model significantly decreases the samples needed to converge at all levels of the learning, with both the MFRL and the unidirectional algorithms using more than 60% fewer samples than when using the nominal tabular transition function (results from Fig. 11(b)).

Comparing the MFRL algorithm using a DBN transition model (MFRL-DBN) to unidirectional transfer with a DBN (UNI-DBN), we see that MFRL-DBN uses slightly (though not significantly) fewer samples in Σ_3 while using more samples in Σ_2. This trade-off is by design, as MFRL assumes (under Assumption 1) that using more samples in a lower-fidelity simulator is worth using fewer samples in the real world, since the real world samples are far more costly. Indeed, this is true in the RC car experiments where actually running the car in the lab takes significantly more time and effort than the computer simulation. The combination of a DBN and the MFRL transfer framework significantly decreases the number of real world samples compared to the other representations/framework combinations studied here.

This decreased number of required samples now allows us to solve the track problem using a larger state/action space. Table III shows state and action discretization values for the larger problem solved using MFRL with a DBN transition function representation. Note that, in this case, the state/action size is nearly an order of magnitude larger than the state/action size for the smaller problem.



(a) Initial real-world policy

(b) Converged real-world policy

Fig. 14: Actual trajectories driving by the car during one of the learning runs. (a) As the MFRL algorithm transitions to the real car for the first time, the attempted trajectories are too aggressive for the actual hardware, leading to the car frequently exiting the allowable region. (b) By the end of the learning process, the car consistently drives fast along the straight segments and slower around the turns, yielding fast lap times.

Fig. 13(b) shows that with this larger state/action space, the MFRL algorithm uses on average more than 40% fewer samples in the real world than the unidirectional transfer algorithm. As in the puddle world examples, giving the algorithm generative access to the simulators does little to improve the convergence rate of the algorithm. In all of the tested cases, the algorithms converged to policies very similar to those found in the smaller state/action space problem described in Section VIII-C. In these experiments, the car converged to this policy with fewer real-world samples than the size of the state/action space.

An example of the types of policies that the MFRL algorithm converges to in these experiments is shown in Fig. 14. Plots of the actual trajectory driven by the car for the first several and last several laps during the learning process are shown in Figs. 14(a) and 14(b), respectively. When the MFRL algorithm first transitions to the real world, the optimism assumption in the simulators is evident by the fact that the car often attempts to drive quickly around the turns, a strategy that works well in simulation but causes the car to drive out of the feasible region in practice. Eventually, the algorithm discovers that driving quickly on the straight-aways and slower around the cones yields a safer and faster policy.

IX. CONCLUSIONS

We have introduced MFRL, which extends lessons from the multi-fidelity optimization community to sequential decision making problems. MFRL transfers heuristics from lower- to higher-fidelity simulators to guide exploration. Unlike previous transfer learning techniques, our framework also allows agents in lower-fidelity simulators to plan using higher-fidelity learned model parameters, a tactic we have shown is crucial for minimizing sub-optimal steps in the real world. Many robotics domains already use simulators of varying fidelity during the initial stages of hardware and software development. MFRL can leverage these existing simulators to decrease the dependence of RL algorithms on the physical hardware.

Throughout MFRL, the learning agents retain theoretical sample efficiency guarantees over the entire learning process because of our integration of the KWIK-Rmax framework, and our empirical results show these algorithms are also efficient in practice. Experiments with an RC car show that, not only is the framework theoretically sound, but it is also a practical technique for scaling RL algorithms to real-world decision making problems.

APPENDIX

Proof of Theorem 1: First, we set $\bar{\epsilon}$ and $\bar{\delta}$, the accuracy parameters for a learner of an individual action's mean reward at a particular level d. The following settings will be sufficient for the theorem:

$$\bar{\epsilon} = \frac{\epsilon}{2D}, \qquad \bar{\delta} = \frac{\delta}{|A|^2 D}. \tag{4}$$

Instantiating m based on (1) with these values and applying Hoeffding's inequality [33] produces a sample complexity bound of $O(\frac{D^2}{\epsilon^2}\log(\frac{|A|^2 D}{\delta}))$. There are $|A|$ arms at the given level, thus yielding the bound given in the theorem statement. This bound guarantees that, at each level d, if an arm is pulled m times, the rewards are learned with $\bar{\epsilon}$ accuracy with probability $1-\bar{\delta}$. Applying a Union bound across actions gives a probability of failure at a specific level d of $\frac{\delta}{|A|D}$.

Each execution at a level must learn a new action's reward before moving up or down, and once an arm's value is set from above it cannot be sampled at the current level. Therefore, by the Pigeonhole principle, there can be at most $|A|D$ level changes (changes to d). Applying a Union bound across these $|A|D$ possible runs, each with failure probability $\frac{\delta}{|A|D}$, the total probability of failure in the algorithm is δ, satisfying the overall failure bound. Since each level achieves accuracy of $\frac{\epsilon}{2D}$, we can add the potential errors across levels to determine the total error due to pruning out actions based on rewards from below. This quantity is $2D * \frac{\epsilon}{2D} = \epsilon$, so the accuracy requirements are fulfilled with the sample efficiency described in the theorem statement. ■

Proof of Lemma 1: Set $\bar{\delta}$ and $\bar{\epsilon}$ based on (4) and again instantiate each individual arm's KWIK Learner with m based on (1). As above, by Hoeffding's inequality we are ensured that, with probability $1-\delta$, the learners will introduce no more than ϵ error throughout the process.

Now consider action a as above. At each level $d' \geq d$, by Definition 2, we have that the expectation on the reward of a^*_D will satisfy

$$R_{d'}(a^*_D) \geq R_D(a^*_D) - \sum_{\bar{d}=d}^{D-1} \beta_{\bar{d}}. \tag{5}$$

From the lemma's assumption, we also have:

$$\mu_d < R_D(a^*_D) - \sum_{\bar{d}=d}^{D-1} \beta_{\bar{d}} - \epsilon. \tag{6}$$

Combining these two inequalities shows that at level d' we will have $R_{d'}(a^*_D) > \mu_d$. Since the upper bound $U_{d'a^*} \geq R_{d'}(a^*)$ with high probability, we also have $U_{d'a^*} > \mu_d$. This means whenever we enter level d', action a^*_D will be used before a. By Hoeffding's inequality, pulling arm a^*_D m times will give us a mean reward estimate $\hat{R}_{d'}(a^*_D) \geq R_{d'}(a^*_D) - \bar{\epsilon}$ with high probability, so there will also be no need to pull arm a at level d' after collecting these m samples. ■

Proof of Theorem 2: Consider any action a executed in Σ_D and its associated μ_{D-1} values at the next lower-fidelity simulator as defined in Lemma 1. From Lemma 1, we have $\mu_{D-1} \geq R_D(a^*_D) - \beta_{D-1} - \bar{\epsilon}$. Otherwise, with high probability, a would have been pruned and not executed in Σ_D. If a is near optimal we are done. If not, then by line 8 of Algorithm 1 the algorithm must have taken action a at level $D-1$ and, with high probability, found $U_D(a) = \hat{R}_{D-1}(a) \geq R_D(a^*_D) - \beta_{D-1} - \bar{\epsilon}$. Therefore, the only way to determine that a is sub-optimal is to execute it in Σ_D. ■

Proof of Theorem 3: For the first portion, we begin by noting that, when entering a level, the KWIK-Rmax algorithm at each instantiation makes only a polynomial number of suboptimal steps (where $V^\pi(s) < V^*(s) - \epsilon$). This bound on the number of suboptimal steps is a known property of the KWIK-Rmax algorithm [7], [34]. We also note that the derivation of this property shows that "unknown" states are only encountered a polynomial number of times. In fact, this is exactly $B(\epsilon, \delta)$ where B is the KWIK bound for learning the transition and reward functions.

Since we wish to limit the number of samples used at each level and not just the number of suboptimal steps, we need to also bound the number of *optimal* steps taken at level d. For this, we utilize m_{known} as described in (2). There can be at most $m_{known} - 1$ steps between encounters with "unknown" states, and, since the latter can only occur $B(\bar{\epsilon}, \bar{\delta})$ times, this results in at most $B(\bar{\epsilon}, \bar{\delta})(m_{known} - 1)$ samples.

We also need to limit the number of steps before moving to another level. For moving down to level $d-1$, a state/action pair at $d-1$ can only be unknown if it is unknown at level d. Therefore, the bound on the number of steps before this might happen is the same as the number of steps before learning all the unknown states: $B(\bar{\epsilon}, \bar{\delta})(m_{known} - 1)$. For moving up to level $d+1$, a maximum of $B(\bar{\epsilon}, \bar{\delta})(m_{known} - 1) + m_{known}$ steps is needed to reach the conditions for going to the next higher level of fidelity.

Now we need to show that if m_{known} known states in a row are encountered, then, with high probability, we have identified the optimal policy at the current level. This property has been shown previously in Theorem 4 of [34], which we briefly recount here. First, we can define an escape probability $Pr(W)$ of encountering an unknown state. Then, we have $V^\pi(s_t, m_{known}) \geq V^\pi_{M_k}(s_t, m_{known}) - Pr(W)V_{\max}$, where $V^\pi(s_t, m_{known})$ is the value of running policy π from s for m_{known} steps. By using the closeness of this finite-horizon sum to the infinite discounted sum and several properties of the KWIK-Rmax algorithm, Li showed that $V^\pi(s_t, m_{known}) \geq V^*(s_t) - \frac{3\epsilon}{4} - Pr(W)V_{\max}$. If $Pr(W) < \frac{\epsilon}{4V_{\max}}$, then we have a very low probability of reaching an unknown state, and it can be shown that the current policy is near-optimal. Otherwise, $Pr(W) \geq \frac{\epsilon}{4V_{\max}}$, which means that the probability of reaching an unknown state is high, and, with high probability, an unknown state will be encountered before m_{known} known states are seen in a row.

For the number of possible changes to d, $D + 2D|\Sigma|$ is an upper bound on the number of level changes because each backtrack can only occur when at least one parameter is learned, and the number of parameters in the system is $|\Sigma|D$. The number of "up" entries can only be D more than he number of down entries, giving us $D+2D|\Sigma|$ level changes. ■

Proof of Lemma 2: The proof of this property is similar to the proof of Lemma 1 so here we outline the argument.

We set $\bar{\delta}$ and $\bar{\epsilon}$ based on (3) to ensure the accuracy of the known model parameters throughout the learning process. Through the simulation lemma (Lemma 4 of [45]) it is known that a learned MDP with ϵ-accurate parameters will model the value function with comparable loss (adding some additional, but still polynomial, terms).

Therefore, a chain of inequalities similar to those in Lemma 1 can be formed, but now replacing R with Q from the lower-fidelity simulators. The situation in each state becomes exactly the same as in the single-state bandit case, where an action will not be chosen if its optimistic value has fallen beyond the optimal action's Q-value as stated in the lemma. ■

ACKNOWLEDGMENT

This work was partially supported by the National Science Foundation Graduate Research Fellowship under grant No. 0645960 and by the Office of Naval Research Science of Autonomy program under contract No. N000140910625. The authors also acknowledge Boeing Research & Technology for support of the facility in which the experiments were conducted.

REFERENCES

[1] R. Sutton and A. Barto, *Reinforcement Learning, an Introduction*. Cambridge, MA: MIT Press, 1998.

[2] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Robot. Research*, vol. 32, no. 11, pp. 1238–1274, Aug. 2013.

[3] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Int. Conf. Mach. Learn.*, 1997, pp. 12–20.

[4] P. Abbeel, M. Quigley, and A. Y. Ng, "Using inaccurate models in reinforcement learning," in *Int. Conf. Mach. Learn.*, 2006, pp. 1–8.

[5] J. Z. Kolter and A. Y. Ng, "Policy search via the signed derivative," in *Robot.: Sci. and Syst.*, 2009, pp. 1–8.

[6] T. D. Robinson, K. E. Willcox, M. S. Eldred, and R. Haimes, "Multifidelity optimization for variable-complexity design," in *AIAA/ISSMO Multidiscip. Anal. and Opt. Conf.*, 2006, pp. 1–18.

[7] L. Li, M. L. Littman, T. J. Walsh, and A. L. Strehl, "Knows what it knows: a framework for self-aware learning," *Mach. Learn.*, vol. 82, no. 3, pp. 399–443, Mar. 2011.

[8] T. A. Mann and Y. Choe, "Directed exploration in reinforcement learning with transferred knowledge," in *European Workshop Reinforcement Learn.*, 2012, pp. 59–76.

[9] T. Dean and K. Kanazawa, "A model for reasoning about persistence and causation," *Comput. Intell.*, vol. 5, no. 2, pp. 142–150, Feb. 1989.

[10] S. Balakirsky and E. Messina, "A simulation framework for evaluating mobile robots," in *Proc. Performance Metrics for Intell. Syst. Workshop*, 2002, pp. 1–8.

[11] A. Staranowicz and G. L. Mariottini, "A survey and comparison of commercial and open-source robotic simulator software," in *Proc. 4th Int. Conf. Perv. Technol. Rel. to Assistive Env.*, 2011, pp. 56:1–56:8.

[12] M. Cutler, T. J. Walsh, and J. P. How, "Reinforcement learning with multi-fidelity simulators," in *IEEE Int. Conf. Robot. and Autom.*, 2014, pp. 3888–3895.

[13] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Found. and Trends in Robot.*, vol. 2, no. 12, pp. 1–142, Feb. 2013.

[14] M. Deisenroth, D. Fox, and C. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Trans. Pattern Anal. and Mach. Intell.*, vol. 37, no. 2, pp. 408–423, Feb. 2015.

[15] S. Levine and V. Koltun, "Variational policy search via trajectory optimization," in *Adv. in Neural Inf. Process. Syst.*, 2013, pp. 207–215.

[16] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-efficient generalization of robot skills with contextual policy search." in *Association for Adv. of Artificial Intell.*, 2013, pp. 1401–1407.

[17] E. Winner and M. M. Veloso, "Multi-fidelity robotic behaviors: Acting with variable state information," in *Association for Adv. of Artificial Intell.*, 2000, pp. 872–877.

[18] E. J. Schlicht, R. Lee, D. H. Wolpert, M. J. Kochenderfer, and B. Tracey, "Predicting the behavior of interacting humans by fusing data from multiple sources," in *Int. Conf. Unc. in Artificial Intell.*, 2012, pp. 756–764.

[19] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning," *J. Mach. Learn. Research*, vol. 8, no. 1, pp. 2125–2167, Dec. 2007.

[20] B. Fernández-Gauna, J. M. López-Guede, and M. Graña, "Transfer learning with partially constrained models: Application to reinforcement learning of linked multicomponent robot system control," *Robot. and Autonomous Syst.*, vol. 61, no. 7, pp. 694–703, July 2013.

[21] M. G. Azar, E. Brunskill, and A. Lazaric, "Sequential transfer in multi-armed bandits with finite set of models," in *Adv. in Neural Inf. Process. Syst.*, 2013, pp. 2220–2228.

[22] E. Brunskill and L. Li, "Sample complexity of multi-task reinforcement learning," in *Int. Conf. Unc. in Artificial Intell.*, 2013, pp. 122–131.

[23] N. Jakobi, "Evolutionary robotics and the radical envelope-of-noise hypothesis," *Adaptive Behavior*, vol. 6, no. 2, pp. 325–368, 1997.

[24] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Trans. Evol. Comput.*, vol. 17, no. 1, pp. 122–145, Feb. 2013.

[25] J. C. Zagal and J. Ruiz-Del-Solar, "Combining simulation and reality in evolutionary robotics," *J. Intell. and Robot. Syst.*, vol. 50, no. 1, pp. 19–39, Mar. 2007.

[26] R. Moeckel *et al.*, "Gait optimization for roombots modular robots-matching simulation and reality," in *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2013, pp. 3265–3272.

[27] A. L. Strehl, L. Li, and M. L. Littman, "Reinforcement learning in finite mdps: Pac analysis," *J. Mach. Learn. Research*, vol. 10, pp. 2413–2444, Dec. 2009.

[28] F. A. Viana, V. Steffen Jr., S. Butkewitsch, and M. de Freitas Leal, "Optimization of aircraft structural components by using nature-inspired algorithms and multi-fidelity approximations," *J. Global Opt.*, vol. 45, no. 3, pp. 427–449, Nov. 2009.

[29] A. Molina-Cristobal, P. R. Palmer, B. A. Skinner, and G. T. Parks, "Multi-fidelity simulation modelling in optimization of a submarine propulsion system," in *Vehicle Power and Prop. Conf.*, 2010, pp. 1–6.

[30] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons, Inc., 1994.

[31] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 279–292, May 1992.

[32] R. Brafman and M. Tennenholtz, "R-max - a general polynomial time algorithm for near-optimal reinforcement learning," *J. Mach. Learn. Research*, vol. 3, pp. 213–231, Oct. 2002.

[33] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. American Statistical Association*, vol. 58, no. 301, pp. 13 – 30, Mar. 1963.

[34] L. Li, "A unifying framework for computational reinforcement learning theory," Ph.D. dissertation, Computer Science Dept., Rutgers Univ., New Brunswick, NJ, 2009.

[35] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Neural Inf. Process. Syst.*, 1996, pp. 1038–1044.

[36] O.-A. Maillard, T. A. Mann, and S. Mannor, ""how hard is my mdp?" the distribution-norm to the rescue," in *Adv. in Neural Inf. Process. Syst.*, 2014, pp. 1835–1843.

[37] A. L. Strehl and M. L. Littman, "Online linear regression and its application to model-based reinforcement learning," in *Adv. in Neural Inf. Process. Syst.*, 2007, pp. 1417–1424.

[38] E. Brunskill, B. Leffler, L. Li, M. Littman, and N. Roy, "Provably efficient learning with typed parametric models," *J. Mach. Learn. Research*, vol. 10, pp. 1955–1988, Dec. 2009.

[39] R. C. Grande, T. J. Walsh, and J. P. How, "Sample efficient reinforcement learning with Gaussian processes," in *Int. Conf. Mach. Learn.*, 2014, pp. 1332–1340.

[40] C. Diuk, A. Cohen, and M. L. Littman, "An object-oriented representation for efficient reinforcement learning," in *Int. Conf. Mach. Learn.*, 2008, pp. 240–247.

[41] C. Boutilier, R. Dearden, and M. Goldszmidt, "Stochastic dynamic programming with factored representations," *Artificial Intell.*, vol. 121, no. 1-2, pp. 49–107, Aug. 2000.

[42] S. Park, J. Deyst, and J. P. How, "Performance and lyapunov stability of a nonlinear path following guidance method," *J. Guidance, Control, and Dynamics*, vol. 30, no. 6, pp. 1718–1728, Nov. 2007.

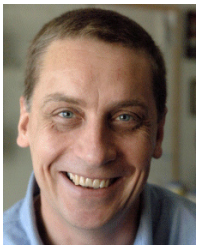
[43] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *American Control Conf.*, 2007, pp. 2296–2301.

[44] E. Velenis, E. Frazzoli, and P. Tsiotras, "Steady-state cornering equilibria and stabilisation for a vehicle during extreme operating conditions," *Int. J. Vehicle Autonomous Syst.*, vol. 8, no. 2, pp. 217–241, Oct. 2010.

[45] M. J. Kearns and S. P. Singh, "Near-optimal reinforcement learning in polynomial time," *Mach. Learn.*, vol. 49, no. 2-3, pp. 209–232, Nov. 2002.

Mark Cutler received a B.S. degree in Mechanical Engineering from Brigham Young University in 2010 and a S.M. degree in Aeronautics and Astronautics from MIT in 2012. He is currently a Ph.D. student at MIT. His research interests include aggressive flight control and learning algorithms for fast robotic systems.

Thomas J. Walsh is a research engineer with Kronos Incorporated. He received his Ph.D. in Computer Science at Rutgers University, and previously held research positions at MIT, the University of Kansas, and the University of Arizona. His research focuses on efficient learning in sequential decision making problems with rich structure.



Jonathan P. How is the Richard Maclaurin Professor of Aeronautics and Astronautics at MIT. He received a B.A.Sc. from the University of Toronto in 1987 and his S.M. and Ph.D. in Aeronautics and Astronautics from MIT in 1990 and 1993. Prior to joining MIT in 2000, he was an Assistant Professor at Stanford University. Research interests include robust coordination and control of autonomous vehicles. He is an Associate Fellow of AIAA, and a senior member of IEEE.