2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Reinforcement Learning with Multi-Fidelity Simulators

Mark Cutler, Thomas J. Walsh, Jonathan P. How

***Abstract*— We present a framework for reinforcement learning (RL) in a scenario where multiple simulators are available with decreasing amounts of fidelity to the real-world learning scenario. Our framework is designed to limit the number of samples used in each successively higher-fidelity/cost simulator by allowing the agent to choose to run trajectories at the lowest level that will still provide it with information. The approach transfers state-action Q-values from lower-fidelity models as heuristics for the "Knows What It Knows" family of RL algorithms, which is applicable over a wide range of possible dynamics and reward representations. 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 allows RL algorithms to find near-optimal policies for the real world with fewer expensive real-world samples than previous transfer approaches or learning without simulators.**

I. INTRODUCTION

Simulators play a key role as testbeds for robotics control algorithms, but deciding when to use them versus collecting real-world data is often treated as an art. For instance, several *reinforcement learning* (RL) algorithms use simulators to augment real robot data [1]–[3], but none of those agents actively decide when to sample simulated or real-world data. Likewise, the *transfer learning* (TL) community [4] has sought to more seamlessly transfer information from simulators to the real world [5], but TL methods for RL also do not prescribe theoretically justified rules for *when* an agent should move from simulation to the real world (e.g. [5]).

Deciding to sample from different simulators or the real world is important considering the relative cost of experience at various levels of fidelity. For example, an ODE simulation can take more time than a simple Newtonian inverse kinematics model, but it is still typically much less costly (and less risky) than running trials in the real world. Thus an efficient learning agent could greatly benefit from actively choosing to collect samples in less costly, low fidelity, simulators.

This paper introduces, analyzes, and empirically demonstrates a new framework, *Multi-Fidelity Reinforcement Learning* (MFRL), depicted in Figure 1, for performing reinforcement learning with a heterogeneous set of simulators (including the real world itself). The framework combines ideas from both multi-fidelity optimization [6] and advances in model-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.

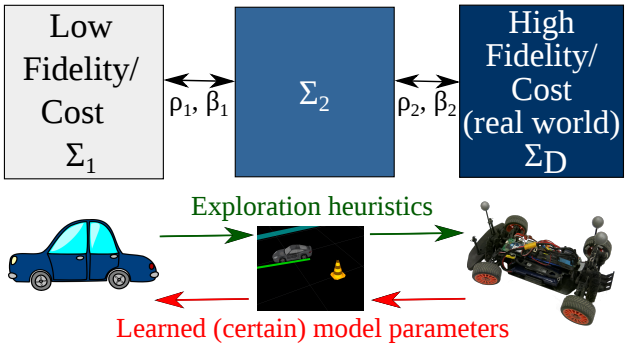


Fig. 1. MFRL architecture: a multi-fidelity chain of simulators and learning agents. Agents send exploration heuristics to higher-fidelity agents and learned model parameters to lower fidelity agents. The environments are related by state mappings ρ_i and optimism bounds β_i. Control switches between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

Unlike *unidirectional* methods that transfer heuristics only once to the real-world agent [5], the MFRL framework also specifies rules for when the agent should move up to a higher fidelity simulator, as well as moving down in fidelity 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 levels that have been proven suboptimal below, (2) minimizes (under certain conditions) the number of samples used in the real world and (3) limits the total number of samples used in all simulators. In addition, MFRL without resets provably uses no more (worst case) samples at the highest fidelity level than unidirectional transfer approaches.

We showcase MFRL in both bandit learning and multi-state RL. In addition, our theoretical results hold for a large class of representations such as linear and Gaussian-noise dynamics covered by the KWIK learning framework [7]. We test our algorithms in benchmark domains and real-world remote controlled (RC) car control problems. Our RC car experiments show that near-optimal driving policies can be found with fewer samples from the real car than unidirectional transfer methods or without using simulators.

Our main contributions are (1) introducing the MFRL

Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA, USA {cutlerm,twalsh,jhow}@mit.edu

framework for learning with multiple simulators, (2) a theoretical analysis of the framework's sample complexity and (3) several demonstrations of efficient learning on an RC car with fewer real-world data points than unidirectional transfer or learning without simulators. These results demonstrate MFRL is an efficient manager of the low and high quality simulators often available for robotics tasks.

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. [2]). However, such systems require practitioners to decide when to run policies in the simulator/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 [1], [3]. However, these approaches are specific to policy search algorithms and again do not provide exploration guarantees.

Multi-fidelity models have been considered for a single agent that learns different policies for varying observability conditions [8]. Instead, our algorithm has control over what level of fidelity it runs trajectories in and targets a single policy for an observable real-world task. Multi-fidelity models have been used in the multi-agent context to combine data from people performing tasks in different simulators [9]. However, these policies were learned from traces through supervised learning (not RL).

Our MFRL framework can be viewed as a type of *transfer learning* [4]. In TL, values or model parameters are typically used to *bootstrap* learning in the next task (e.g. [4]). By contrast, we use values from lower-fidelity models as heuristics guiding exploration, and our agent (rather than nature) controls which simulator it is using, including the capability to return to a lower-fidelity simulator. We also differ from transfer learning between environments with different action sets [10] because our simulators can have different dynamics or rewards, rather than just different available actions.

A similar approach to ours is Transferred Delayed Q-Learning (TDQL) [5]. Like us, TDQL transfers the value function *unidirectionally* from a source learning task as a heuristic initialization for the target learning task. Our approach is different because our algorithm moves not only up, but also *down* the multi-fidelity simulator chain. Also, TDQL is designed only for tabular representations while we tie our approach to a much larger class (KWIK) of base learners. We show that, when an agent is not reset when returning to a simulator, our method provably uses no more (worst case) samples at the highest fidelity level than unidirectional transfer approaches like TDQL. Our experiments also verify that MFRL uses significantly fewer samples in the real world compared to unidirectional transfer.

Our work extends techniques in multi-fidelity optimization (MFO) [6] to sequential decision making problems. In MFO, an optimization problem, such as setting design parameters of an aircraft [11], is solved using multiple models. Techniques in MFO include learning model disparities [11] and constraining search based on results from lower fidelity models [12]. However, MFO does not consider sequential decision making (RL) tasks. MFRL borrows lessons from MFO by updating models with higher-fidelity data and performing constrained exploration based on lower-fidelity results.

III. BACKGROUND AND ASSUMPTIONS

In this section we provide background on RL and the KWIK framework. We also describe the assumptions made about our multi-fidelity simulators.

A. Reinforcement Learning

We assume that each simulator can be represented by a Markov Decision Process (MDP) [13], $M = \langle S, A, R, T, \gamma \rangle$ with states S, actions A, reward function $R(s,a) \mapsto \Re$ and transition function $T(s,a,s') = Pr(s'|s,a)$. The optimal *value function* $Q^*(s,a) = R(s,a) + \gamma \sum_s' T(s,a,s') V^*(s')$ where $V^*(s) = \max_a Q^*(s,a)$ is the expected sum of discounted rewards when taking action a in state s and then acting optimally thereafter. A deterministic policy $\pi : S \mapsto A$ is said to be optimal when $\pi^*(s) = \text{argmax}_a Q^*(s,a)$.

In reinforcement learning (RL) [13], an agent knows S,A, and γ but not T and R, which it learns from interaction with the environment. This leads to an inherent tension between *exploration*, where an agent seeks out new experiences to update its model, and *exploitation* of its current knowledge.

To judge exploration efficiency, we follow previous definitions [7] of *sample complexity* for an RL agent as a bound (with probability $1-\delta$) on the number of suboptimal steps (where $V^{\pi_t}(s) < V^*(s) - \epsilon$). The KWIK framework [7] standardizes sample complexity analysis for model-based RL by measuring the number of times the learners of T and R are uncertain in making a prediction. Specifically, a KWIK supervised learner is given an input (i.e. state/action pair) and asked to make a prediction $\hat{y}$ (e.g. a reward prediction) of the true output y. If the agent is certain of its prediction ($||\hat{y} - y|| < \epsilon$ with high probability) it predicts $\hat{y}$. Otherwise, it must state "I don't know" (denoted $\perp$) and will view the true label y. A hypothesis class H is said to be *KWIK learnable* if an agent can guarantee (with high probability) it will only predict $\perp$ a polynomial (in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $|H|$) number of times.

In the KWIK-Rmax RL algorithm [7], an approximate MDP is built with $\hat{T}$ and $\hat{R}$ based on the predictions of KWIK learners for each parameter. If a learner predicts $\perp$, the uncertainty is replaced using the "optimism in the face of uncertainty" heuristic, specifically setting the value of the corresponding uncertain states to $\frac{R_{\max}}{1-\gamma}$. This interpretation encourages exploration of unknown areas, but not at the expense of already uncovered dominant policies. It also guarantees polynomial sample complexity of the resulting RL agent [7]. The sample complexity of KWIK-Rmax may be smaller than $|S|$ if T and R can be represented compactly (e.g. as linear functions with only n parameters for an infinite $|S|$).

B. 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 size of their corresponding T and R. Since S may differ between Σ_i and a higher fidelity Σ_j (some variables may be absent from Σ_i), we follow prior work in TL [4] and assume a *transfer mapping* $\rho_i : S_i \mapsto S_j$ exists. Specifically, we assume that $S_i \subseteq S_j$ 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 our 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 define *fidelity* based on how much Σ_i *overvalues* the state-actions of Σ_j. Specifically, the fidelity of Σ_i to Σ_j (with associated ρ_i and tolerance β_i) is

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

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. Otherwise, Σ_i is considered to have no fidelity to Σ_j. While this may seem restrictive, this relationship is fairly common in real-life simulators. For instance, in our car simulators, the lowest fidelity Σ 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 1: An optimistic *multi-fidelity simulator chain* is a series of D simulators ordered $\Sigma_1 ... \Sigma_D$, 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.

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

Assumption 1: A single step from simulator Σ_i has the same *cost* as a polynomial (in $|\Sigma_i|$) number of samples 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 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 (Σ_D) and in certain simulators (e.g. most commercial video games).

Given such simulators, our objectives are the following:

1) Minimize the number of suboptimal steps (learning samples) 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 $|\Sigma|$ and D) number of switches between simulators.

Objective 1 skews the sampling burden to lower fidelity simulators while objectives 2 and 3 limit the sample complexity of the algorithm as a whole.

IV. MULTI FIDELITY BANDIT OPTIMIZATION

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

A. A MF-Reward Learning Algorithm

Consider a chain of bandit simulators: at each level $d \in \{1...D\}$ 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 $\hat{R}_d$ with parameter m based on accuracy parameters ϵ and δ. $\hat{R}_d$ predicts $\hat{\mu}(a)$, the empirical mean payout for arm a, if the number of samples seen for that arm is greater than m, and otherwise predicts $\perp$, translating into $R_{\max}$ as a loose upper bound.

Algorithm 1 presents the Multi-Fidelity Bandit Framework (MF-Bandit) for accomplishing objectives 1-3 using a KWIK learner that keeps track of reward means and upper bounds $\hat{U}_{da}$. MF-Bandit also tracks the *informed* upper bound U_{da}, which is the minimum of $\hat{U}_{da}$ 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_{da}), 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 8). Before executing the action, it checks to make sure the action has been tried sufficiently at the simulator below (lines 9). If not, values from d that are converged are transferred back to level $d-1$ (lines 12-15) and control is passed there. Otherwise, if learning at d is not finished, the action is taken and μ and $\hat{U}$ are updated (lines 17-22). Once the optimal action has been identified, the algorithm moves up to the level $d+1$ (lines 23-27).

Our algorithm differs sharply from unidirectional (only up) heuristic transfer [5] because it can backtrack to a lower fidelity simulator when a previously identified optimal action performs badly above. Effectively, backtracking asks the lower learner to find a new optimal policy given new knowledge from higher-fidelity simulators.

Algorithm 1 Multi-Fidelity Bandit Framework

Input: A bandit simulator chain $\langle \Sigma, \beta \rangle$, Actions A, $R_{\max}$, Accuracy requirements ϵ and δ
Initialize: con_{da}, $change_d \coloneqq false, \forall a, d$
Initialize: KWIK learners $\hat{R}_d(a, \bar{\epsilon}, \bar{\delta})$
$d \coloneqq 1$
$\hat{U}_{da}$, $U_{1a} \coloneqq R_{\max} \forall a$
for each timestep **do**
  Select $a^* \coloneqq \text{argmax}_a \, U_{da}$
  $closed_d \coloneqq con_{da^*} \vee a^*$ is definitely near optimal
  if $d > 1 \wedge \neg con_{d-1,a^*} \wedge change_d$ **then**
    {send values back to level $d-1$}
    $change_{d-1} \coloneqq false$
    for $a \in A$ **do**
      if con_{da} **then**
        $\hat{R}_{d-1} \coloneqq \hat{R}_d$ {copy μ_d and $\hat{U}_d$ down}
        $con_{d-1,a}$, $change_{d-1} \coloneqq true$
    $d \coloneqq d - 1$
  else if $\neg closed_d$ **then**
    Execute a^*, Observe r.
    Update $\hat{R}_d(a^*)$ {Update μ_{da*} and $\hat{U}_{da*}$}
    $U_{da^*} \coloneqq \min(U_{d,a^*}, \hat{U}_{da^*})$
    if $\hat{R}(a^*)$ switched from $\perp$ to "known" **then**
      con_{da^*}, $change_d \coloneqq true$
  else if $d < n \wedge closed_d$ **then**
    {chosen action already converged, go up}
    Where $\neg con_{d+1,a}$, $U_{d+1,a} = U_{d,a} + \beta_{d-1}$
    $change_{d+1} \coloneqq false$
    $d \coloneqq d + 1$

An example where this behavior is beneficial on our RC car is when an optimal configuration of parameters in the simulator generates a path with tight turns, but data in the real world proves such settings cause the car to spin out. In such a scenario there is still information to be gleaned from the lower-level simulator by exploring policies *given* the knowledge of spinning out from above, which is exactly what transferring the mean values down accomplishes.

B. Bandit Examples

We now present examples to showcase various features of Algorithm 1. First, we show 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 simulator are $\Sigma_1 = \{0.8, 0.8, 0.8, 0.8, 0.1\}$, $\Sigma_2 = \{0.8, 0.8, 0.6, 0.6, 0.1\}$, and $\Sigma_3 = \{0.8, 0.6, 0.6, 0.6, 0.1\}$, all with uniform random noise up to 0.1. Table I (left) shows the results of running Algorithm 1 in this scenario with our KWIK bandit learner with $m = 20$ as well as results with only $\langle \Sigma_2, \Sigma_3 \rangle$ and only Σ_3. We see that the use of both simulators or just Σ_2 produces a significant reduction in samples from Σ_3 and that having Σ_1 helps limit samples needed from Σ_2.

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 (right) 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 (never going back to a lower fidelity simulator) [5]. 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 steps in the real world is minimized.

TABLE I
SAMPLES USED FROM SIMULATORS

Sims Used	$\boldsymbol{\Sigma_1}$	$\boldsymbol{\Sigma_2}$	$\boldsymbol{\Sigma_3}$
Σ_1,Σ_2,Σ_3	100	80	40
Σ_2, Σ_3		100	40
Σ_3			100

Sims Used	$\boldsymbol{\Sigma_1}$	$\boldsymbol{\Sigma_2}$	$\boldsymbol{\Sigma_{3'}}$
Σ_1,Σ_2,$\Sigma_{3'}$	100	80	60
Uni-directional	100	60	80
$\Sigma_{3'}$			100

C. Theoretical Analysis

We now formalize the intuitive lessons from above with theoretical guarantees for MFRL when the base learner is the KWIK bandit learner. We begin by focusing on objectives 2 and 3 from Section III-B: limiting the number of suboptimal actions at each level and the number of samples overall.

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

Proof: Given ϵ and δ, an application of Hoeffding's inequality yields a KWIK algorithm with $m = O(\frac{1}{\epsilon^2} \ln(\frac{1}{\delta}))$. Setting the base learners at each level with stricter accuracy requirements: $\bar{\epsilon} = \frac{\epsilon}{2D}$ and $\bar{\delta} = \frac{\delta}{A^2 D}$ yields the desired result in terms of the maximum number of samples. Each execution at a level must determine a new parameter before moving up or down, and once an arm's value is set from above it cannot be sampled at the current level. Therefore there can be at most AD level changes. Applying a Union bound across actions gives a probability of failure at a specific level d of $\frac{\delta}{AD}$ and applying another union bound across the number of changes yields the desired result. ■

Now we turn our attention to objective 1, minimizing the number of samples used in Σ_D. We begin with the following lemma, which is similar to Lemma 1 of [5], stating that no action is tried at a level beyond which it is dominated by the value of a^* in Σ_D.

Lemma 1: Consider action a at level d and let $\mu_d = \hat{R}_d(a)$ if a has been executed m times at level d, otherwise $\mu_d = U_d(a)$. 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.

Proof: Set $\bar{\epsilon}$ and $\bar{\delta}$ as above. At each level $d' \geq d$, by Definition 1 we have that the expectation on the reward of a^*_D will be $R_{d'}(a^*_D) \geq R_D(a^*_D) - \sum_{\bar{d}=d}^{D-1} \beta_{\bar{d}} - \epsilon > \mu_d$ based on the lemma's assumption and Definition 1. 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 within $\bar{\epsilon}$ of $R_{d'}(a^*_D)$ with high probability,

so there will also be no need to pull arm a at level d' after collecting these m samples. ■

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

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.

Proof: 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, 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 not near-optimal is to execute it in Σ_D. ■

A corollary of this theorem is that the MF-Bandit algorithm uses provably no more (wost case) samples in Σ_D than unidirectional transfer, which does not ensure that actions in Σ_D have been vetted in lower fidelity simulators.

V. MULTI-FIDELITY REINFORCEMENT LEARNING

We now instantiate the principles of generating heuristics from low-fidelity simulators and sending learned model data down from high fidelity simulators in the full (multi-state, cumulative reward) RL case.

A. The MFRL Algorithm

Algorithm 2 shows the MFRL framework, which takes as input a simulator chain, maximum reward $R_{\max}$, state-mapping between simulators $\rho_{1\ldots D-1}$, a planner P (e.g. Value Iteration [13]) and accuracy requirements ϵ , δ, and m_{known}, the latter of which determines when to move to a higher fidelity simulator. MFRL is similar to MF-Bandit but now the heuristic passed to higher fidelity simulators is the Q-function, and both the reward and transition functions are passed down to lower fidelity simulators.

The algorithm begins by initializing the variables d, m_k and $change_d$ and the base KWIK learners $\hat{T}$ and $\hat{R}$, with parameters $\bar{\epsilon}$ and $\bar{\delta}$ described in Section V-C. We use the shorthand $\hat{\Sigma}$ to denote the MDP induced by $\hat{T}$ and $\hat{R}$ and replacing all $\perp$ predictions with a heuristic (in this case corresponding lower fidelity values). Q-values for the lowest fidelity simulator are set optimistically using $\frac{R_{\max}}{1-\gamma}$, and the agent begins choosing actions greedily at that level.

If the selected state/action pair still has uncertainty at level $d-1$ (according to the KWIK model learners there), and a change has been made at the current level, the algorithm backtracks one layer of fidelity (lines 9-13).[1] Otherwise, the action is executed and $\hat{T}$ and $\hat{R}$ are updated. If the update changes a parameter from unknown to known, these learned parameters are transferred to all lower simulators (line 23). This has the effect of forcing the corresponding agent in the lower level simulator (if we return there) to explore policies that might be optimal given the dynamics of the higher fidelity simulator. If the model parameters change, the $change_d$ flag is also set and the planner recalculates the Q values. However, unlike standard (no-transfer) KWIK-Rmax, the algorithm uses heuristic values from the lower fidelity simulator (Q_{d-1}) to fill in the Q-values for state/actions where the KWIK learners are uncertain during planning.

Finally, we have the convergence check (line 24) to see 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. We provide theoretical guidelines for setting m_{known} in Theorem 3.

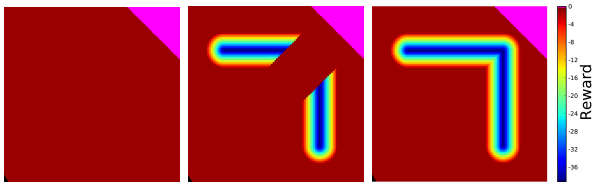
Algorithm 2 MFRL (MF-KWIK-Rmax)

1: **Input:** A simulator chain $\langle \Sigma, \beta, \rho \rangle$, $R_{\max}$, Planner P, accuracy parameters $\langle \epsilon, \delta, m_{known} \rangle$
2: $d \coloneqq 1$ and $m_k \coloneqq 0$
3: **Initialize:** $change_d \coloneqq$ false, $\forall d$
4: **Initialize:** $2D$ KWIK learners $\hat{R}(\bar{\epsilon}, \bar{\delta})$ and $\hat{T}_d(\bar{\epsilon}, \bar{\delta})$
5: **Initialize:** $Q_0 \coloneqq \frac{R_{\max}}{1-\gamma}$
6: **Initialize:** $\hat{Q}_1(s,a) \coloneqq P(\langle S_1, A, \hat{R}_1, \hat{T}_1, \gamma \rangle, Q_0)$
7: **for** each timestep and state s **do**
8: Select $a^* \coloneqq \text{argmax}_a \hat{Q}_d(s,a)$
9: **if** $d > 1 \wedge change_d \wedge (\hat{T}_{d-1}(\rho^{-1}_{d-1}(s), a^*) = \perp \vee \hat{R}(\rho^{-1}_{d-1}(s), a^*) = \perp)$ **then**
10: {send values back to level $d-1$}
11: $\hat{Q}_{d-1}(s,a) \coloneqq P(\hat{\Sigma}_{d-1}, Q_{d-2} + \beta_{d-2})$
12: $m_k \coloneqq 0$
13: $d \coloneqq d-1$
14: **else**
15: Execute a^*, Observe r, s'.
16: **if** $\hat{R}_d(s,a^*) = \perp \vee \hat{T}_d(s,a^*) = \perp$ **then**
17: $m_k \coloneqq 0$
18: Update $\hat{R}_d$ and/or $\hat{T}_d$ that predict $\perp$
19: **else**
20: $m_k \coloneqq m_k + 1$
21: **if** $\hat{R}_d(s,a^*)$ or $\hat{T}_d(s,a^*)$ went from $\perp$ to *known* **then**
22: $\hat{Q}_d(s,a) \coloneqq P(\langle S_d, A, \hat{R}_d, \hat{T}_d, \gamma \rangle, Q_{d-1} + \beta_{d-1})$
23: Set $\langle \hat{R}, \hat{T} \rangle_{d'}(\rho_d^{-1}(s), a^*)$ for $d' \leq d$ based on the new values. Set $change_{d'} \coloneqq true$
24: **if** $d < D \wedge m_k = m_{known}$ **then**
25: {Go up to level $d+1$}
26: $\hat{Q}_{d+1}(s,a) \coloneqq P(\hat{\Sigma}_{d+1}, Q_d + \beta_d)$
27: $m_k \coloneqq 0$, $change_d \coloneqq$ false
28: $d \coloneqq d+1$

B. Puddle World with MFRL

We illustrate the behavior of MFRL in a variant of puddle world (Figure 2) [13] with multi-fidelity simulators. A puddle

[1] While this "one unknown" backtracking is theoretically correct, in our experiments we wait until $m_{unknown}$ such states are encountered, which helps control sampling at lower level simulators.



(a) No Puddle (b) Some Puddle (c) Full Puddle

Fig. 2. $\Sigma_1 \ldots \Sigma_3$ for puddle world. Σ_1 has no puddle, Σ_2 has most of the puddle but the optimal policy can bypass it.

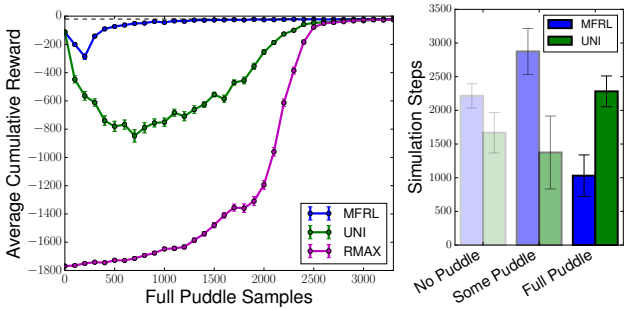


Fig. 3. Left: During learning, MFRL consistently outperforms unidirectional transfer and no-transfer Rmax at the Full Puddle level. Each point is an average of 1000 learning runs (standard errors shown). Greedy policies are evaluated 60 times, each capped at 600 steps. Right: At the top level, MFRL requires fewer than half the samples needed by unidirectional transfer. This is accomplished by transfering some learning burden to the other simulators. Average of 1000 learning runs (standard deviations shown).

world agent moves in one of four directions with Gaussian noise and a step cost of -1 (0 at the goal) and high negative rewards in the puddle (increasing with depth). We implemented our puddle world with diagonal actions (to illustrate the behavior of the algorithm in Σ_2) and $\gamma = 0.95$ so the optimal policy in Σ_3 is generally to skirt along the outer edges of the puddle.

We tested our algorithm in the presence of two lower fidelity simulators with respect to the "real" puddle world (Figure 2). Σ_1 is the same environment without the puddle. Σ_2 contains a large portion of the puddle, but has an opening in the worst reward region from Σ_D. This again creates a scenario where an optimal policy in the low-fidelity simulator supports a policy that is poor in Σ_D but still contains significant useful information (in this case the puddle portions in Σ_2).

Figure 3 (left) shows learning curves from this experiment. MFRL performed the best, with some negative transfer at the beginning from the "shortcut" in Σ_2, but as it encounters the real puddle it passes this information back to the learner in Σ_2 (through several level changes) and forces that lower-fidelity agent to find a way around the puddle. The result is a consistent and significant improvement over unidirectional transfer throughout learning. In fact, unidirectional transfer takes almost as long as the no-transfer case to consistently find the optimal policy since it must explore essentially the whole puddle at the top level. Figure 3 (right) shows an average of 1000 runs of MFRL with bars showing the average number samples in each of the 3 simulators. The MFRL agent relied heavily on Σ_1 and Σ_2, gathering crucial information through these lower cost simulators to decrease learning time at the top level.

C. Theoretical Analysis

Most of the theoretical results for multi-state MFRL are similar to the bandit theorems, so here we note places where changes to the bandit proofs are made. We assume here that the variables in each Σ_i are the same, that is ρ is the identity mapping. The properties can still be made to hold with missing variables with increases to β where transfer based on default values may cause under-estimation.

Theorem 3: MFRL with $m_{known} = \frac{1}{1-\gamma} \ln\left(\frac{4R_{\max}}{\bar{\epsilon}(1-\gamma)}\right)$ and $\bar{\epsilon} = \frac{\epsilon}{4(D+1)}$ and $\bar{\delta} = \frac{\delta}{4(D+2D|\Sigma|)}$ has the following properties with probability $1-\delta$. (1) Actions that have been proven to be suboptimal with respect to Q^*_D at level d will not be tried above. (2) Actions taken in Σ_D will either be near-optimal or lead to an unknown state not learned about below or that needs to be learned about in Σ_D. (3) The total number of level changes is no more than $(D+2D|\Sigma|)$ and the number of total samples is polynomially bounded in $\langle |\Sigma|, D, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma} \rangle$.

Proof: [sketch] Property (1) can be proven similarly to Lemma 1, with the learned Q values replacing the learned reward function. Property (2) is a property of the KWIK-Rmax algorithm (see Lemma 13 and the Theorem 4 of [7]) with an admissible heuristic. Our transferred Q-values are (w.h.p.) admissible by Definition 1 and the accuracy guarantees of the underlying KWIK learners.

For Property (3), $D + 2D|\Sigma|$ 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.

By instantiating the KWIK learners $\hat{T}$ and $\hat{R}$ with $\bar{\epsilon}$ and $\bar{\delta}$ we achieve a polynomial bound on the number of encounters with unknown states, and we know the algorithm can have at most $m_{known} - 1$ samples between these unknown states without changing d. Since m_{known} is also polynomial in the relevant quantities, we have the desired bound on the number of samples. Also, based on the analysis of KWIK-Rmax (Theorem 4 of [14]), if m_{known} known states in a row are encountered, then with high probability we have identified the optimal policy at the current level. ■

Property (2) of the theorem shows that, if the agent is not reset when returning to a simulator, MFRL will, with high probability, enter no more unknown states (where $\hat{T}$ or $\hat{R}$ predict $\perp$) in Σ_D than a unidirectional transfer method with the same base learner and architecture. Unlike the bandit case, this does not translate into such steps being *necessary* as the base KWIK-Rmax architecture only gives an upper (not lower) bound on the number of samples needed.

VI. RC CAR RESULTS

RC cars have been popular testbeds for RL algorithms [1], [3], [15], though none of these approaches chose which simulator (if any) to run trajectories in. We ran two experiments (one bandit, one multi-state) on an RC car with two simulators of the car's dynamics.

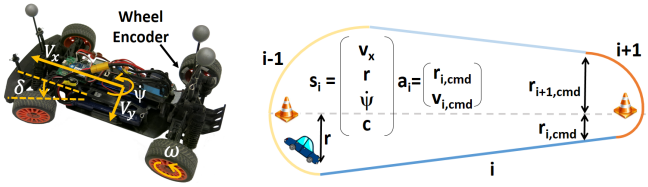


Fig. 4. Left: Our RC car and associated variables. Right: The track configuration and state/actions spaces for the experiments.

A. Experimental Setup

Our MFRL algorithms are experimentally verified using an RC car driving on an indoor track. The car is an off-the-shelf $1/16$ scale 4-wheel drive rally car shown in Figure 4. 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 16 bit microcontroller.

Figure 4 (right) shows the car task consisting 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. 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 for that segment. If the car drives (or often slips) out of a virtual "drivable" area around the track, the car resets to a fixed initial condition and is given a large negative reward. The state variables in s_i (Figure 4 (right)) 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, $r_{i,cmd}$, and velocity, $v_{i,cmd}$, fully defines the desired path for segment i (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 [16]. 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. A steering angle command and a desired wheel velocity is computed using the closed-loop controllers from [17], where the cross track error term in the steering angle control law is omitted, as the cross track error is minimized by the pure pursuit algorithm. The C_y parameter in this control law is found by matching measured vehicle data to input commands.

The steering angle, δ, and commanded wheel speed, ω_{cmd}, are broadcast to the car's microcontroller over a wireless serial connection. 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 effects of changing battery voltage on the velocity dynamics.

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. The higher fidelity simulator models the basic dynamics of the car, including wheel slip, where model parameters such as the "Magic Tyre Parameters" [18] 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 simulators, but also on the physical car.

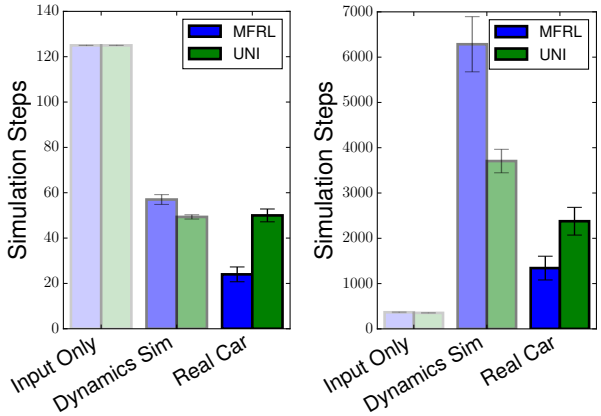


Fig. 5. Samples used by MFRL and unidirectional transfer at each level. The bandit case (left) and in the state-based case (right) are shown. In both cases, the MFRL algorithm uses significantly fewer samples in the real world, but converges to an identical policy. Each case is the average of 3 learning runs with standard deviations shown.

B. Experiment Results for Bandit Setting

The first RC car experiment takes place in the (single-state) 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 the unidirectional transfer of Q-value heuristics [5]. Both of these found an optimal policy within 60 steps on the real car and so we did not compare to a no-transfer algorithm since it would need at least 250 trials to identify the optimal policy[2]. 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 $m = 2$ in the real world, meaning 2 tries with a given parameter setting were needed to determine its value.

Figure 5 (left) depicts the average number of samples used in each simulator by each algorithm. MF-Bandit uses fewer than half as many samples at 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

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

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.

C. Experiments for the State-Based Setting

In the multi-state case, we used the state space described earlier 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). Because of the discretization in the state space, which makes the simulation results potentially noisier (from state aliasing), we used $m = 3$ in Σ_2 and the real car.

In the experiment, both MFRL and unidirectional transfer converged to an optimal policy that just under 3.7 seconds around the track. Figure 5 (right) shows the average number of 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 a straightaway. 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 between cones. 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, Σ_1 pruned policies that were too slow while Σ_2 pruned policies that were too fast on the curves. This left Σ_3 (the real car) to refine the policies in a noisier environment.

VII. EXTENSIONS AND CONCLUSIONS

More powerful representations than our tabular T and R models, including linear dynamics and Gaussian-noise models, are polynomially KWIK learnable [7]. This extends Theorem 3 to certain continuous MDPs where the discretization used in our experiments would not be necessary and will be empirically investigated in future work.

Another extension is relaxing Assumption 2, which assumes samples can only be obtained by performing trajectories. However, allowing such generative access does not necessarily make choosing *where* to sample any easier. For instance, consider the case where an agent in Σ_d encounters a state that has not been visited in Σ_{d-1}. It is tempting to simply query Σ_{d-1} at that state. However, these samples might be ineffective. For example, in our first RC car experiment, optimistic transitions in Σ_{D-1} cause the car to attempt aggressive maneuvers that fail Σ_D. But this does not mean the system should query the simulator to learn what happens *during* an unrecoverable spin. The more prudent course is to replan in Σ_{D-1} given the transition dynamics actually encountered in Σ_D, which is what MFRL does. Thus, actively choosing samples in MFRL with a generative mode is a topic for future work.

We have introduced MFRL, which extends lessons from the multi-fidelity optimization community to sequential decision making. MFRL transfers heuristics to guide exploration in high fidelity (but higher cost) simulators. Unlike previous transfer learning techniques, our framework also allows the transfer of learned model parameters to agents in lower fidelity simulators, a tactic we have shown is crucial for minimizing sub-optimal steps in the real world. Throughout this process, our agents retain sample efficiency guarantees over the entire learning process because of our integration of the KWIK-Rmax framework. Our experiments with an RC car show that not only is the framework theoretically sound, but it is also a practical technique for scaling reinforcement learning algorithms to real-world decision making.

REFERENCES

[1] P. Abbeel, M. Quigley, and A. Y. Ng, "Using inaccurate models in reinforcement learning," in *ICML*, 2006.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *NIPS*, 2006.

[3] J. Z. Kolter and A. Y. Ng, "Policy search via the signed derivative," in *RSS*, 2009.

[4] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via intertask mappings for temporal difference learning," *Journal of Machine Learning Research*, vol. 8, no. 1, pp. 2125–2167, 2007.

[5] T. A. Mann and Y. Choe, "Directed exploration in reinforcement learning with transferred knowledge," in *European Workshop on Reinforcement Learning (EWRL)*, 2012.

[6] T. D. Robinson, K. E. Willcox, M. S. Eldred, and R. Haimes, "Multi-fidelity optimization for variable-complexity design," in *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2006.

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

[8] E. Winner and M. M. Veloso, "Multi-fidelity robotic behaviors: Acting with variable state information," in *AAAI*, 2000.

[9] 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 *UAI*, 2012.

[10] 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," *Robotics and Autonomous Systems*, vol. 61, no. 7, pp. 694–703, 2013.

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

[12] 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 Propulsion Conf.*, 2010.

[13] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998.

[14] L. Li, "A unifying framework for computational reinforcement learning theory," Ph.D. dissertation, Rutgers University, New Brunswick, NJ, 2009.

[15] T. K. Lau and Y.-h. Liu, "Stunt driving via policy search," in *ICRA*, 2012.

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

[17] 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 *ACC*, 2007.

[18] E. Velenis, E. Frazzoli, and P. Tsiotras, "Steady-state cornering equilibria and stabilisation for a vehicle during extreme operating conditions," *International Journal of Vehicle Autonomous Systems*, vol. 8, no. 2, pp. 217–241, 2010.