

# Expert Learning for State Machine Selection on an Autonomous Robot

Brenna Argall  
The Robotics Institute  
Carnegie Mellon University  
Pittsburgh, PA 15224  
bargall@cs.cmu.edu

Brett Browning  
The Robotics Institute  
Carnegie Mellon University  
Pittsburgh, PA 15224  
brettb@cs.cmu.edu

Manuela Veloso  
Computer Science Department  
Carnegie Mellon University  
Pittsburgh, PA 15224  
mmv@cs.cmu.edu

**Abstract**—Hierarchical state machines have proven to be a powerful tool for controlling autonomous robots due to their flexibility and modularity. For most real robot implementations, however, much if not all of the control hierarchy is hand-coded. As a result, the development process is often time intensive and error prone. In this paper, we explore the use of an experts learning approach, based on Auer and colleagues Exp3 [1], to help overcome these limitations. In particular, we develop a modified learning algorithm, which we call rExp3, that exploits the structure provided by a control hierarchy by treating each state machine as an 'expert'. Our experiments validate the performance of rExp3 on a real robot performing a task, and demonstrate that rExp3 is able to quickly learn to select the best state machine expert to execute. Through our investigations in these environments, we identify a need for faster learning recovery when expert performances undergo a discrete switch, such as in response to a discrete environment change. We introduce a modified learning rule to improve the recovery rate in these situations and demonstrate through simulation experiments that rExp3 performs as well or better than Exp3 under such conditions.

## I. INTRODUCTION

There are a number of established approaches for developing a control architecture for generating autonomous mobile robot behavior. A common and powerful approach is to use hierarchies of finite state machines [2], [5], [6], whereby control consists of finite state machines that operate in parallel or that use other state machines to generate complex actions. Although many frameworks exist, nearly all require extensive hand-coding when applied to real robot control problems. That is, the designer is responsible for creating the state machine hierarchy, and the control policies and parameters encoded in each control state. This is a time consuming and error prone process and one that does not scale well with increasing complexity of the robot or tasks.

We seek practical learning techniques that can aid, or replace, this process, thereby enabling more complex robots and tasks. In this paper, we make two key contributions. First, we present an experts-based algorithm, based on Exp3 [1], for learning to select between state machines that can be integrated with a working robot control hierarchy. Our second contribution extends Exp3 further, which we here introduce as rExp3, to enable a more rapid response to discrete switches in expert performance. To evaluate our approach, we explore the performance of the algorithm in both simulation, and on a real robot platform. The real

robot implementation uses the Segway RMP robots [12] performing a task integral to robot soccer and makes use of the skill control hierarchy [6].

Our paper is structured in the following way. In the ensuing section, we describe hierarchical state machines, as applied to robot control, that will form the basis for this paper. We then describe the Exp3 algorithm, and its extension to integrate with a robot control architecture. We additionally outline our modified algorithm, called rExp3, for enhanced responsiveness to environment switches. Based on this, we present the evaluations of the resulting approach in simulation and on the real robot platform. We then close with our conclusions and future goals.

## II. STATE MACHINES FOR ROBOT CONTROL

In this section, we describe our use of hierarchical state machines for autonomous robot control. Alternate implementations of state-based control are described in *cite*, just other state-based control, or control in general<sup>??\*</sup>. We define a state machine for control as consisting of a set of control states  $cs_i \in CS$ . Each control state encodes a control policy  $\pi_i$ , which is a function of the robot's internal state and its beliefs about the world (i.e. is a result of its perception system). This policy  $\pi_i$  determines which action  $ac_i \in AC$  to take, when in control state  $cs_i$ .

Transitions between different control states occur as a function of the robot's beliefs. Additionally, this state machine may terminate (ie. enter an absorbing state) with success or failure. A state machine may therefore be viewed as providing deliberative goal-driven behavior, where it either succeeds and achieves the goal or fails. For ease of reference, we will refer to such a state machine as a *skill*. Thus, the term skill in this paper is equivalent to a goal-directed state machine for control, and the two terms will be used interchangeably.

State machines may arrange into *hierarchies* which pass control between them. Two actions types are available for selection by a control policy: complex or primitive. A complex action consists of calling another control policy, and thus passing control to another state machine This builds a hierarchy of state machines, or equivalently a hierarchy of skills. By contrast, primitive actions command the robot using its available control primitives (e.g. velocity control). Note, that state machines may also execute in parallel,

with or without synchronization coupling their execution. However, this is beyond the scope of this paper and will not be considered further.

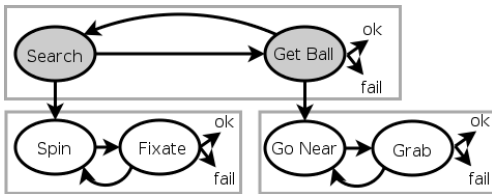


Fig. 1. An example soccer-related skill hierarchy. The robot will search for the ball, and when it finds it will run up and grab it. Shown transitions are unlabelled, but are conditioned on the robot’s perceptual beliefs.

Figure 1 shows a schematic of a hierarchical state machine, or skill hierarchy, for controlling a robot performing a simple soccer skill. The robot’s task is to grab the nearest ball it can find. It must therefore first search for the ball, and run up to grab it once found. The skill hierarchy enables a larger task to be easily decomposed into sub-tasks which can be solved as independent problems (namely, *search* and *get ball* within this example). Moreover, each of the resulting skills may be reused for other similar problems. It is this ‘divide and conquer’ ability that makes state machines so useful for robot control. The drawback to this approach is that for a real robot, the control architecture is often hand coded. Typically, it is not task decomposition which is difficult. Instead, most designer effort focuses on developing the control policies which call primitive actions (for example, those for the states *go near* or *grab* in Figure 1). Moreover, the performance of skills using such policies is highly dependent on the robot hardware and in many cases the environment (\*\*cite Aibo hierarchy\*\*).

In this paper, we aim to provide adaptability to the robot through the use of an experts learning algorithm. The key idea is as follows. Let us first view each state machine, or skill, as an expert that ‘recommends’ a control policy to achieve a particular goal. Suppose also, that we have multiple skills available for achieving the same goal. If these skills vary in their use of perception or more generally in their control policy, their performance characteristics (ie. success/failure rates) will be different. Thus, we can use expert learning to determine which ‘expert’ skill should be executed to maximize performance. In summary, we can free up the designer to create a range of skills that have varying strengths and weaknesses, and let the expert learning automatically determine which skill should be run to achieve maximum performance (see Figure 2).

### III. LEARNING WITH EXPERT ADVICE

Expert learning (or the k-armed bandits problem, originally proposed by [13]) addresses the issue of choosing between multiple action recommenders (experts) at each time step. The performance of an expert is reflected in its probability of being selected, where good performance pairs with high selection probability.

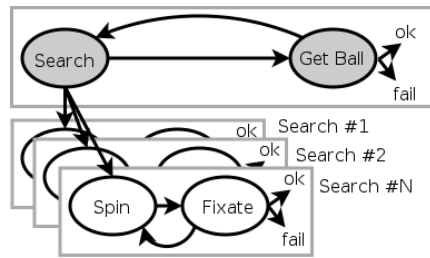


Fig. 2. Experts learning applied to state machines enables multiple skills to be developed, and the best one learned for a given environment.

At each decision cycle,  $k$ , each of the  $n$  experts makes a recommendation. The algorithm selects a single expert and executes the corresponding action, resulting in a payoff of  $r^k \in \mathbb{R}$ . After  $d$  decision cycles, a sequence of  $r^1, r^2, \dots, r^d$  payoffs have been rewarded. The aim of expert learning is to select the best expert over all decision cycles. We formulate this learning objective in terms of regret, where the regret at decision cycle  $k$  is the difference between the observed reward  $r^k$  of the selected action, and the reward  $r_b^k$  we would have received from the best expert. Summed over all decision cycles,

$$Regret = \sum_{k=1}^d r_b^k - \sum_{k=1}^d r^k. \quad (1)$$

The goal of expert learning is to minimize this total regret.

When the chosen action executes on a real robot, however, only the reward for the recommending expert will be observed. Fortunately, the algorithm Exp3 (“Exponential-weight algorithm for Exploration and Exploitation”), introduced by Auer and colleagues in [1], addresses exactly this issue. Exp3 handles what they refer to as a partial information game, where at each trial only the reward gained by the chosen expert, and not by every expert, is observed. The algorithm is a modification of the HEDGE experts learning algorithm [1], [10], which is founded upon the weighted majority algorithm of [11] and aggregating strategies of [14].

In Exp3, the reward earned at decision cycle  $k$  is scaled inversely with the expert’s selection probability  $Pr(x^k = i)$  for that decision cycle, to compensate the fact that in a partial information game, experts with low probability are infrequently chosen and therefore have fewer observed rewards. Thus the reward earned at trial  $k$  by expert  $i$  is

$$\hat{r}_i^k = \frac{r_i^k}{Pr(x^k = i)} \quad (2)$$

for the selected expert, and zero for all other experts. [1] proved that this regret approaches zero in the limit, assuming opponents with static policies.

More concretely, the probability of choosing expert  $i$  at time step  $k$  is governed by the normalizing equation

$$Pr(x^k = i) = \frac{w_i^{k-1}}{\sum_j w_j^{k-1}}. \quad (3)$$

where  $w_i^{k-1} \equiv e^{\sum_{t=1}^{k-1} \hat{r}_i^t}$  is the weight of expert  $i$  at time step  $k-1$ . Weights are initialized to be equal across experts,

and the weight of an executed expert  $i$  is updated according to the following rule

$$w_i^k = w_i^{k-1} (m_i^k)^{\frac{1}{Pr(x^k=i)}}. \quad (4)$$

Here the multiplier  $m_i^k \equiv e^{\hat{r}_i^k}$  follows the notation of [3] for update simplicity. Note that the product of the weight at decision cycle  $k-1$  and the multiplier at cycle  $k$  is the same as adding  $\hat{r}_i^k$  to  $\sum_k \hat{r}_i^{k-1}$ , and thus represents the cumulative reward received by expert  $i$  up to trial  $k$ .

#### IV. EXPERTS LEARNING FOR SKILL SELECTION

We now examine how to utilize expert learning in a hierarchical state machine (or skill hierarchy) for robot control. Concretely, we address the problem of learning to select the best skill, or that of *skill selection*. That is, from within our current execution of a skill, and given its current control state, which is the best state machine to call next (see Figure 2). The idea behind this approach is that if a designer creates multiple skills to accomplish the same task, the robot will then be able to automatically determine the best skill to use given the current conditions.

We define the higher-level skill as the parent skill, and the state machine choices which it may call as the child skills. At each decision cycle, the parent skill chooses a new child skill to execute. Upon termination of the child skill, a reward is assigned to that skill. Its execution is evaluated based on success or failure; that is, the two absorbing states are assigned a payoff. Furthermore, each binary reward may be modified to incorporate a measure of execution quality. In our particular implementation, a quality measure of execution time was added; and so faster executions earned more reward than slower executions. This modification was applied the binary success reward exclusively, since the failure absorbing state was only entered after a timeout and thus all failed executions performed equally by this measure.

We thus have a framework amenable to the experts learning approach; namely, each child skill is an expert that we select with some probability. Based on the payoff from the selected expert/skill, we modify its future selection probability.

##### A. Adapting to Discrete Performance Changes

The algorithm Exp3 has an additional feature of promoting adaptability within non-stationary environments. In particular, we consider situations where the probability distribution describing the performance of each expert switches in a discrete, non-continuous way. That is, some change in the world causes the performance of each expert to change such that their relative selection probability ranking reorders. For example, suppose expert A recommends a wheel velocity for the robot which is appropriate for smooth ground, while expert B recommends one which is appropriate for rough terrain. Operating indoors on a flat surface would rank the probability of selecting A above that of selecting B. If the robot then goes out of doors and onto the bumpy grass, a change in the world has occurred which alters the performance of each expert. Namely, expert B now begins to

outperform expert A, and as a result its selection probability will come to outrank that of A.

Within Exp3, experts which have performed poorly in the past are rewarded more strongly for good performance than experts which have already performed well in the past. This occurs because of the ratio, within the weight update, which scales reward inversely with selection probability. For an expert with low selection probability (indicating poor past performance), this ratio is larger than that of an expert with high selection probability (indicating good past performance). An expert with a previously low probability of being selected, but which now performs well, may therefore have its weight increased quickly in this manner.

This adaptability idea, however, is fully captured by the Exp3 algorithm only when an expert succeeds. For example, when a high probability expert fails, it is no longer performing as it had in the past. To follow the reasoning above, its weight should decrease quickly, as this weight no longer represents its performance under the current world conditions. However, in this case the scaling ratio decreases the effect of the reward on the weight update, and the weight will actually change only minimally.

We generalize this adaptability idea by saying that *expected* expert performances should result in *small* weight changes, while *unexpected* expert performances should result in *large* weight changes. Concretely, we categorize expert performances  $s^k$  as successful (1) or failed (0), and selection probabilities as high  $Pr(x^k = i) > \delta_h$  or low  $Pr(x^k = i) < \delta_l$ ,  $0 < \delta_l \leq \delta_h < 1$ . We then define expert performance classifications according to Table 1.

Expected	Unexpected
$Pr(x^k = i) > \delta_h \wedge s^k = 1$	$Pr(x^k = i) > \delta_h \wedge s^k = 0$
$Pr(x^k = i) < \delta_l \wedge s^k = 0$	$Pr(x^k = i) < \delta_l \wedge s^k = 1$

TABLE I. Classification of Expert Performance

Note the similarity between this idea and the Win or Learn Fast (WoLF) approach presented by [4] for learning agents participating in zero sum competitive games. In WoLF, agents that are losing (ie. not performing as expected) learn at a faster rate, while those that are winning (ie. performing as expected) learn at a slower rate.

##### B. The rExp3 Algorithm

To further enhance adaptability, we want expert weights to update based upon whether they reflect the expected performance of their recommendation. We introduce the algorithm rExp3 (*responsive* Exp3) to encourage swift reactions to unexpected expert performances. In particular, our goal is to strengthen the link between selection probability and expected performance. To accomplish this, rExp3 modifies Exp3 to depend the weight updates explicitly upon expert failure or success.

Key to our algorithm modification is to introduce two distinct formulations for the weight update multiplier, one each for whether an expert succeeds or fails. This distinction occurs in two places within the weight update; the exponent

on the multiplier  $m_i^k$  and the reward earned by the expert performance.

The exponent within the weight update of a chosen expert  $i$  at trial  $k$  is determined according to the following update rule,

$$w_i^k = w_i^{k-1} (m_i^k)^{\frac{1}{g}} \quad (5)$$

$$g = \begin{cases} Pr(x^k = i) & \text{if success} \\ 1 - Pr(x^k = i) & \text{if failure} \end{cases}$$

Furthermore, we incorporate two distinct reward metrics, allowing their specific formulation to be task-determined, but requiring that a failure case reward be strictly less than 0, and a success case reward be strictly greater than 0. In this manner a failed execution always reduces the weight of a given expert, while a successful execution always increases the weight of a given expert. Recalling that  $m_i^k \equiv e^{r_i^k}$ , the strength of the exponent on this multiplier therefore determines the strength of the reward’s ability to reduce or increase expert weight.

Failure-case expert outcomes are now properly represented within the weight update, such that unexpected performances have larger changes in weight than expected ones. Returning to our example of the unexpected failure of an expert with high probability, within its weight update the multiplier exponent ( $\frac{1}{g} = [1 - Pr(x^k = i)]^{-1}$ ) is large. Since  $0 < m_i^k < 1$  (due to negative reward), a larger exponent on this multiplier means a larger reduction in weight. Thus, the unexpected performance results in a significant weight reduction, as we intend.

## V. ALGORITHM VALIDATION ON REAL ROBOTS

We next present results from the integration of our approach into the control architecture of a robot doing a task drawn from robot soccer. To clarify, we refer to a single execution of an expert as a *trial*, and to a series of trials over which learning is performed as a *run*.

### A. Robot Implementation

The robot learning data was collected on a Segway RMP [12]. The Segway RMP is a dynamically balancing robot which in this case was outfitted to play soccer [8], [7]. Included in our augmentations are two cameras for world sensing (Fig. 3) [7], [9], which identify the ball for our example skill.

To illustrate the effectiveness of rExp3 on a robotic system, we applied the algorithm to the example task of learning the best ball searching routine. Seven experts were implemented, which were distinguished by the cameras from which they drew perception information to identify the ball.

Data were collected under two world conditions. In the *soccer scenario* the ball was passed around the robot by human riding a Segway HT, to relate learning back to the Robocup robot soccer domain. The ball therefore appeared at varying distances and states of occlusion, possibly stopped or in motion. In the *occlusion scenario* a controlled switch in

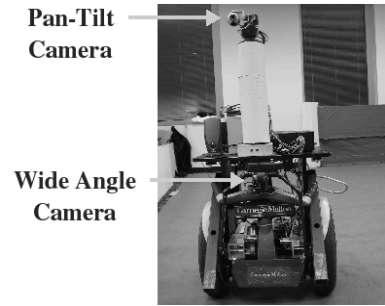


Fig. 3. Our Segway RMP soccer playing robot.

camera occlusion was performed, to examine appropriate and quick adaptability to an environment switch. By physically blocking the camera lens, the ball was visible to exclusively one camera for the first portion of a learning run, and exclusively the other for the remainder.

The reward metric for our implementation was dependent upon both expert success and execution speed. Within the context of our example skill, this represents the importance of not only finding the ball, but finding it quickly. The initial determination of success or failure was binary. The success case reward was then further subject to discounting by  $\lambda$  for the  $t$  times steps of execution,

$$m_i^k = \begin{cases} 1 + \lambda^t d_s & \text{if success} \\ d_f & \text{if failure.} \end{cases}$$

To guarantee that successful executions strictly increase, and failed executions strictly decrease, expert weight, we require  $d_s > 0$  and  $0 < d_f < 1$  (here  $\lambda = 0.98$ ,  $d_s = 0.3$ ,  $d_f = 0.7$ ). For implementation simplicity, we dealt directly with the multiplier  $m_i^k \equiv e^{r_i^k}$  present in our weight update, rather than the actual reward  $r_i^k$ .

### B. Robot Results

The algorithm rExp3 was able to properly and flexibly learn which ball search routines to execute.

Within the soccer scenario, the experts preferred by the learning algorithm agreed with those which performed fastest, as determined by baseline data collected without learning. From this baseline data we qualitatively classify the experts into three performance categories, where Experts 0-2 perform best, Experts 3-4 perform less well, and Experts 5-6 perform poorest. Note that these classifications apply only to soccer scenario environments, and will not necessarily hold when cameras are obstructed within occlusion scenario environments. Figure 4.A presents a single example learning run, where the learned expert (Expert 1) is ‘good’ by baseline data classifications.

In the occlusion scenario, the algorithm rExp3 was able to recover from the switch in occluded camera ( $1.75 \pm 0.5$  trials to recover). Figure 4.B presents a single example learning run, where at first Expert 5 dominates. Expert 5 is an expert which depends upon the camera which will be occluded after the switch (vertical line). Following the switch, accordingly,

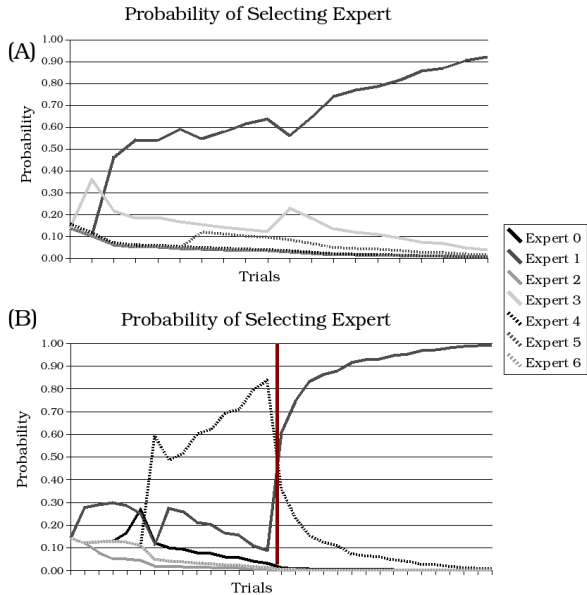


Fig. 4. Single rExp3 robot learning runs, soccer (A) and occlusion (B) scenarios. The vertical line in B indicates an environment switch.

an expert which does not depend upon this camera (Expert 1) comes to dominate.

## VI. ALGORITHM VALIDATION IN SIMULATION

To further test the performance of rExp3, both of the algorithms rExp3 and Exp3 were implemented in simulation. Our aim was that the modification introduced in rExp3 would compare to Exp3 in the following manner:

- Faster recovery (defined below), when the performance was *unexpected*.
- Lower regret, when the performance was *unexpected*.
- Similar regret, when the performance was *expected*.

### A. Simulation Implementation

In our implementation, experts recommend actions under distinct environment scenarios. Environments are constructed such that two scenario classifications are possible: an expert’s performance is expected, or it is unexpected.

Specifically, our simulation mimicked the ball searching task of the real robot implementation. There were again seven experts distinguished by the camera combinations from which they drew perception information. Each expert had an associated failure probability  $Pr(x^k = 0)$ , which was determined by the world and based upon whether a currently occluded camera was depended upon by the expert. All learning parameters were set as in the real robot implementation.

Our intent was to gather data at a simulated environment switch, but to do so in a controlled manner. We therefore began our data collection at this switch, and simulated prior learning by biasing the initial selection probabilities to favor a single expert. Each expert consequently had an associated initial selection probability  $Pr(x^0 = i)$  as well as its failure probability, and the combination of which determined the classification of expert performance as expected or unexpected according to Table 1.

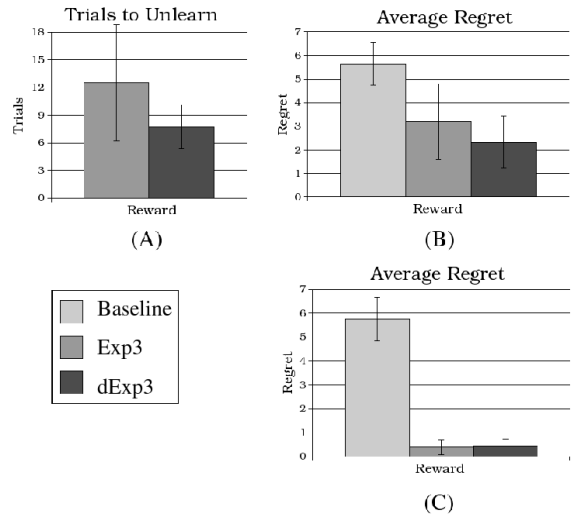


Fig. 5. Average results of baseline (no learning), Exp3 and rExp3 simulation learning data. *Unexpected performance*, trials to recovery (A) and average regret (B); *expected performance* average regret (C). Error bars for all show one standard deviation.

### B. Simulation Results

In the comparison of rExp3 to Exp3, our simulation results show both faster recovery and lower regret for unexpected performances, as well as similar regret for expected performances. Data were collected over 100 runs of 50 trials for each of the expected and unexpected scenarios, as well as 50 runs of 50 trials with no learning, to provide a baseline against which each learning algorithm might be compared.

1) *Faster Recovery (Unexpected Performances)*: The number of time steps required to recover from having learned an expert which in the past had performed well but now begins to fail, was significantly fewer ( $t = 4.14$ , 99.9% confidence interval  $> 3.26$ ) for rExp3 than Exp3 (trials to recover for rExp3 = 7.74 vs. Exp3 = 12.52, Figure 5.A). Here recovery was defined very simply, as the time step at which a favored expert no longer has the highest probability of being selected.

2) *Lower Regret (Unexpected Performances)*: In the cases where experts perform unexpectedly, particularly when experts which previously performed well now begin to fail, rExp3 displayed significantly lower overall regret for its learning runs than Exp3 (regret on rExp3 = 2.33 vs. Exp3 = 3.21,  $t = 3.1$ , 99.5% confidence interval 2.68 – 3.26 Figure 5.B). Both learning algorithms perform significantly better than the baseline data which utilized no learning (rExp3  $t = 18.39$ , Exp3  $t = 9.24$ , 99.9% confidence interval  $> 3.26$ ).

3) *Similar Regret (Expected Performance)*: In the cases when experts exhibited expected performance, and therefore needed no recovery, the difference in regret displayed by algorithm rExp3 in comparison to Exp3 was not significant (regret on rExp3 = 0.44 vs. Exp3 = 0.39,  $t = 0.81$ , 75.0% confidence interval 0.68 – 1.30, Figure 5.C). By contrast, the difference in regret between each learning algorithm and the baseline data which used no learning was significant (rExp3

$t = 39.05$ , Exp3  $t = 38.82$ , 99.9% confidence interval  $> 3.26$ ).

## VII. DISCUSSION

The algorithm rExp3 modifies Exp3 by addressing expert failure or success explicitly within the weight update. In adopting two distinct reward scalings and exponent formulations, our intent is to strengthen the link between expert performance and selection probability.

Relating this modification back to the original inverse scaling motivation of Auer and colleagues, the failure case scaling for a high probability expert might be seen as a prediction on future reward frequency. An expert which begins to fail also begins a drop in selection probability. Scaling with  $\frac{1}{1-Pr(x^k=i)} > \frac{1}{Pr(x^k=i)}$ , in the high probability case, in a sense already begins to compensate for fewer future selections of, and therefore reward observations for, this expert. The same may be reasoned about the inverse scaling with  $Pr(x^k = i)$  in the low probability success case, present within both algorithms, when applied to a low probability expert.

Past work with Exp3 tested the algorithm in environments different to those presented here, namely against an all-knowing adversary [1]. If tested in similar environments to the past work of Exp3, we expect dExp3 would perform at least as well as Exp3. Our reasoning is twofold. First, in our simulation results for scenarios with no explicit unexpected performances, the algorithms responded similarly. Second, particularly within the context of an all knowing adversary, dExp3 would adapt quicker to those changes in the adversary's strategies which resulted in the poor performance of a previously successful expert. Worst case scenario would be if the adversary altered strategy at a rate such that the change in relative expert weights was strong within rExp3, but relatively stable within Exp3. Under such circumstances, however, we expect that the strategy of the adversary would be changing too quickly for any learning to prove useful, in which case we again would expect similar performances.

It is possible during execution that an expert might oscillate between failure and success. Within the weight update of the rExp3 formulation, this will then induce an oscillation in exponent formulation. One argument might favor such an oscillation, as it represents the true behavior of the expert. Indeed, as our intent for this algorithm is very domain specific, we value swift responses to our dynamic soccer environment. By contrast, many others attempt to minimize responses to domain instability **\*\*cite\*\***. Note, however, that an oscillation in exponent formulation will not necessarily cause an oscillation in the actual weight. In fact, such a weight oscillation will only occur if the selection probability of an expert lies at either extreme. From our empirical observations, such extreme oscillations rarely occurred.

## VIII. CONCLUSIONS

We have presented an adapted experts approach for learning the execution control loop on a robot system, and demonstrate its effectiveness on a real robot system. We introduce

a modified experts learning algorithm, which we call rExp3, based upon the Exp3 algorithm of Auer and colleagues [1], to enhance responsiveness to discrete environment switches. The effectiveness of this modification is presented with comparisons between rExp3 and Exp3 within simulation, in addition to the implementation of rExp3 on our Segway robots. When a learned good expert begins to fail, we have shown rExp3 to both accumulate smaller overall regret and 'unlearn' this expert faster, and to behave similarly to Exp3 otherwise. In future work, we intend to apply this algorithm to skill selection in other soccer scenarios, particularly when in response to strategy decisions of the opponent team. An interesting extension would incorporate more weight update formulations; for example to evaluate performance in a more continuous manner, such as "successful, but worse than expected" rather than just "successful" or "failed".

## IX. ACKNOWLEDGMENTS

This work was supported by United States Department of the Interior under Grant No. NBCH-1040007. The content of the information in this publication does not necessarily reflect the position or policy of the Defense Advanced Research Projects Agency (DARPA), US Department of Interior, US Government, and no official endorsement should be inferred.

## REFERENCES

- [1] Peter Auer, Nicol'o Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: The adversarial multiarm bandit problem. In *36th Annual Symposium on Foundations of Computer Science*, pages 322–331, Milwaukee, WI, 1995.
- [2] T. Balch, G. Boone, T. Collins, H. Forbes, D. MacKenzie, and J. Santamaria. Io, ganymede and callisto: A multiagent robot trash-collecting team. *AI Magazine*, 16(2):39–53, 1995.
- [3] Michael Bowling, Brett Browning, and Manuela Veloso. Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04)*, 2004. in press.
- [4] Michael Bowling and Manuela Veloso. Convergence of gradient dynamics with a variable learning rate. In *Proceedings of ICML-2001*, pages 27–34, Williams College, MA, June 2001.
- [5] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, RA-2(1), 1986.
- [6] B. Browning, J. Bruce, M. Bowling, and M. Veloso. Stp: Skills, tactics and plays for multi-robot control in adversarial environments. *IEEE Journal of Control and Systems Engineering*, 2004.
- [7] B. Browning, P. Rybski, J. Searock, and M. Veloso. Development of a soccer-playing dynamically-balancing mobile robot. In *Proceedings of International Conference on Robotics and Automation*, May 2004.
- [8] Brett Browning, Jeremy Searock, Paul E. Rybski, and Manuela Veloso. Turning segways into soccer robots. *Industrial Robot: An International Journal*, 32(2):149–156, 2005.
- [9] Brett Browning and Manuela Veloso. Real-time, adaptive color-based robot vision. In *Proceedings of IROS'05*, 2005.
- [10] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.
- [11] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.
- [12] H. G. Nguyen, J. Morrell, K. Mullens, A. Burmeister, S. Miles, K. Thomas, and D. W. Gage. Segway robotic mobility platform. In *SPIE Mobile Robots XVII*, October 2004.
- [13] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin American Mathematical Society*, 55:527–535, 1952.
- [14] Volodimir G. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383, 1990.