Layered Learning on a Physical Robot

Peggy Fidelman and Peter Stone
Department of Computer Sciences, The University of Texas at Austin
1 University Station C0500, Austin, Texas 78712-0233
{peggy,pstone}@cs.utexas.edu
http://www.cs.utexas.edu/~{peggy,pstone}

***Abstract*— Layered learning is a general hierarchical machine learning paradigm that leverages a given task decomposition to learn complex tasks efficiently. Though it has been validated previously in simulation, this paper presents the first application of layered learning on a physical robot. In particular, it enables the learning of a high-level grasping behavior that relies on a gait which itself must be learned. All learning is done autonomously onboard a commercially available Sony Aibo robot, with no human intervention other than battery changes. We demonstrate that our approach makes it possible to quickly learn both a fast gait and a reliable grasping behavior which, in combination, significantly outperform our best hand-tuned solution.**

Keywords: learning/adaptive systems, legged robots, agents and agent based systems

I. Introduction

In order for robots to be useful for many real-world applications, they must be able to adapt to novel and changing environments. Ideally, a robot should be able to respond to a change in its surroundings by adapting both its low-level skills, such as its walking style, and the higher-level behaviors which depend on these skills. This adaptation should occur as autonomously as possible, because hand-coding is time-consuming and often leads to brittle solutions. Machine learning promises a way to generate solutions with little human interaction, so that when the environment changes the solution can be revised with no more than a few hours of machine time. It is also possible for machine learning to lead to better solutions than hand-tuning, because humans are often biased toward exploring a small part of the space of possible solutions, whereas machine learning explores the space in a systematic way.

Current learning methods typically need a large amount of training data to be effective. Thus, an appealing approach to creating learning *robots* is to train behaviors first in simulation before implementing them in the real world. However, especially when concerned with complex perception or manipulation tasks, we cannot assume an adequate simulator will always exist for a given robot. With no simulator, each trial requires interaction with the physical world in real time. This means it is not possible to offset the costs of an inefficient learning algorithm with a faster processor. The learning algorithm must make efficient use of the information gained from each trial.

Layered learning [11] is a hierarchical machine learning paradigm that leverages a given task decomposition to learn complex tasks efficiently. A key feature is that the learning of each subtask directly facilitates the learning of the next-higher subtask layer. Layered learning has been used previously to generate complex, multi-layer behaviors in *simulated* environments [11], [3], [2], [12]. This paper presents the first implementation of layered learning on a *physical robot*, with all training performed in the real world.

The two learned layers enable i) fast locomotion, and ii) robust object "grasping" (for manipulation) given this locomotion. The first learned layer, reported previously, generated a fast walk directly on a robot via a policy gradient search [6]. This paper builds on that work to learn a higher-level, more goal-oriented behavior using a similar approach. As this new behavior relies on the learned walk, the overall learned behavior can be characterized as an instance of layered learning.

The main contributions of this paper are i) the first instantiation of layered learning on a real robot, and ii) the fully implemented and tested learned grasping behavior.

The remainder of this paper is organized as follows. Section II describes the background and motivation for this work. Section III specifies the tasks to be learned and how the layered learning paradigm can be used to relate them, as well as how the training scenario is set up for each task. Section IV describes the primary machine learning algorithm used in the work. Section V details the results of the training, and Section VI discusses the contributions of this work, as well as possible directions for the future.

II. Background

This section summarizes the layered learning formalism (Section II-A). It also describes the robot hardware used in all experiments and introduces the target task towards which it is trained. (Section II-B).

A. Layered learning

The main principles of the layered learning paradigm (summarized in Table I) are:[1]

TABLE I

The key principles of layered learning.

1) Learning a mapping directly from inputs to outputs is not tractable.
2) A bottom-up, hierarchical task decomposition is given.
3) Machine learning exploits data to train and/or adapt. Learning occurs separately at each level.
4) The output of learning in one layer feeds into the next layer.

[1]This section is adapted from [11].

- **Tractability:** Layered learning is designed for domains that are too complex for learning a mapping directly from the input to the output representation. Instead, the problem is broken down into several task layers. At each layer, a concept needs to be acquired, and any appropriate machine learning (ML) algorithm can be used for this purpose.
- **Decomposition:** Layered learning uses a bottom-up incremental approach to hierarchical task decomposition. Starting with low-level subtasks, the process of creating new ML subtasks continues until the high-level tasks, that deal with the full domain complexity, are reached. The appropriate learning granularity and subtasks to be learned are determined as a function of the specific domain. The task decomposition in layered learning is not automated. Instead, the layers are defined by the ML opportunities in the domain.
- **Learning:** Machine learning is used as a central part of layered learning to exploit data in order to *train* and/or *adapt* the overall system. ML is useful for training functions that are difficult to fine-tune manually. It is useful for adaptation when the task details are not completely known in advance or when they may change dynamically. Like the task decomposition itself, the choice of machine learning method depends on the subtask.
- **Interactions:** The key defining characteristic of layered learning is that each learned layer directly affects the learning at the next layer. A learned subtask can affect the subsequent layer by:
  - constructing the set of training examples;
  - providing the features used for learning; and/or
  - pruning the output set.

Layered learning is formally defined as follows. Consider the learning task of identifying a hypothesis h from among a class of hypotheses H which map a set of state feature variables S to a set of outputs O such that, based on a set of training examples, h is most likely (of the hypotheses in H) to represent unseen examples.

When using the layered learning paradigm, the complete learning task is decomposed into hierarchical subtask layers $\{L_1, L_2, \ldots, L_n\}$ with each layer defined as

$$L_i = (\vec{F_i}, O_i, T_i, M_i, h_i)$$

where:

$\vec{\boldsymbol{F_i}}$ is the input vector of state features relevant for learning subtask L_i. $\vec{F_i} = <F_i^1, F_i^2, \ldots>$. $\forall j,\ F_1^j \in S$.

$\boldsymbol{O_i}$ is the set of outputs from among which to choose for subtask L_i. $O_n = O$.

$\boldsymbol{T_i}$ is the set of training examples used for learning subtask L_i. Each element of T_i consists of a correspondence between an input feature vector $\vec{f} \in \vec{F_i}$ and $o \in O_i$.

$\boldsymbol{M_i}$ is the ML algorithm used at layer L_i to select a hypothesis mapping $\vec{F_i} \mapsto O_i$ based on T_i.

$\boldsymbol{h_i}$ is the result of running M_i on T_i. h_i is a function from $\vec{F_i}$ to O_i.

Note that a layer describes more than a subtask; it also describes an approach to solving that subtask and the resulting solution.

As stated in the Decomposition principle of layered learning, the definitions of the layers L_i are given *a priori*. The Interaction principle is addressed via the following stipulation. $\forall i < n$, h_i directly affects L_{i+1} in at least one of three ways:

- h_i is used to construct one or more features F_{i+1}^k.
- h_i is used to construct elements of T_{i+1}; and/or
- h_i is used to prune the output set O_{i+1}.

It is noted above in the definition of $\vec{F_i}$ that $\forall j,\ F_1^j \in S$. Since $\vec{F_{i+1}}$ can consist of new features constructed using h_i, the more general version of the above special case is that $\forall i, j$, $F_i^j \in S \cup_{k=1}^{i-1} O_k$.

When training a particular component, layered learning freezes the components trained in previous layers, thereby adding additional constraints to the learning process. It also adds guidance, by training each layer in a special environment intended to prepare it well for the target domain.

The original implementation of the layered learning paradigm was on the full robot soccer task in the RoboCup soccer simulator [11]. First, a neural network was used to learn an interception behavior. This behavior was used to train a decision tree for pass evaluation, which was in turn used to generate the input representation for a reinforcement learning approach to pass selection.

A subsequent application of layered learning uses two learned layers, each learned via genetic programming, for a soccer keepaway task in a simplified abstraction of the TeamBots environment [3]. In the full TeamBots environment, four learned layers were used, also on a keepaway task [12]. To our knowledge, there has been no previous implementation of layered learning on a physical robot.

B. Ball acquisition

Acquiring an object is a prerequisite for many types of manipulations in the world [1], [4]. For example, in the case of a Sony Aibo robot playing soccer, one of our motivating testbed domains [9], [10], it is much easier to design effective ways for the robot to kick the ball if we may assume that the ball starts in a specific position relative to the robot. Furthermore, if the robot can *grasp* the ball securely enough, it can move the ball by turning with it until the ball reaches a field position from which the kick is likely to move it to the desired destination (such as in the opponent's goal). Thus, as a representative task, we consider the goal of having a robot walk up to a ball and gain control of it. For the purposes of this paper, we define *control* to mean that the robot holds the ball under its chin in a way that allows it to turn with the ball as shown in Figure 1.

As the robot platform for this research, we use the commercially available Sony Aibo ERS-7, a quadruped robot [7]. The ERS-7 has four legs with three degrees of freedom in each, a head with three degrees of freedom, and a CMOS camera in the head. It has several pressure sensors and two infrared range sensors, as well as position sensors in each of

Fig. 1. An Aibo with control of a ball. Achieving this position without knocking the ball away in the process is a challenge; layered learning allows the Aibo to learn to do this more reliably without sacrificing walking speed.

its joints. The robot is able to capture frames from the camera at a rate of 30 Hz. From these images, our software recognizes objects such as the orange ball based on color segmentation and aggregation [8]. This variety of sensors allows us to rely on local sensing alone. In addition, the 576 MHz 64 bit RISC processor allows all necessary processing to be done onboard. In this work, we use a system for vision processing, walking, and kicking that was developed as part of our larger robot soccer project [9], [10].

III. Layered Learning on a Physical Robot

The process of approaching a ball and then gaining control of it relies on the gait that allows the robot to move toward the ball. Thus, we have a layered learning hierarchy consisting of two layers, with previously learned gait as the bottom layer (L_1) and the novel acquisition behavior as the top layer (L_2).

A. Learning a gait

By the method of Kohl and Stone [6], the gait is defined by a set of 12 continuous parameters specifying, among other things, the shape of the trajectory through which each leg moves as well as the target heights of the front and rear of the body. Thus, gait learning is framed as a parameter optimization problem, with forward speed as the objective function. The learning is accomplished via the policy gradient algorithm described in detail in Section IV.

The fitness of a policy, or set of values for the 12 parameters, is obtained by having one or more Aibos time themselves as they walk a fixed, known distance indicated by a pair of landmarks. To reduce the effect of noise, this evaluation process is performed three times for each policy, and the resulting times are averaged to get the fitness of the policy. Figure 2 depicts the setup of the training process.

In the notation of layered learning, the gait layer (L_1) can thus be defined as follows:

$\vec{F}_1$: $\emptyset$;

O_1: values for the 12 parameters defining a gait, plus the speed of the resulting gait;

Fig. 2. The training paradigm used in gait learning. Each Aibo times itself as it moves between a pair of landmarks (A and A', B and B', or C and C'). Reproduced with permission from Kohl and Stone [6].

T_1: the set of training examples obtained by recording the time it takes to walk back and forth across a fixed distance;

M_1: the policy gradient algorithm described in Section IV;

h_1: the parameters of the fastest discovered gait, and its speed.

The details of the learned walk, including detailed empirical analyses, were reported previously [6], [5]. With three robots continually walking across the field more than 1000 total times for approximately 3 hours, they were able to almost double the speed of their walk, achieving the fastest known walk on the Aibo at the time: 291 mm/sec.[2] Notably, the robots learned without any human intervention other than battery changes approximately once an hour.

The formulation of the walking task within the framework of layered learning is novel to this paper. Section III-B introduces a novel learned behavior in full detail within the layered learning framework. The new behavior uses the learned walk (h_1) as a part of its training scenario.

B. Learning to acquire the ball

The task of learning to capture a ball under the robot's chin is motivated by our ongoing development of the UT Austin Villa four-legged robot soccer team [9], [10]. The robot is only able to kick in certain directions, so it is useful to be able to capture the ball and turn with it before kicking. Our team adopted the following strategy for getting the ball into this position: when the Aibo is walking to a ball with the intent of kicking it and gets close enough, it first slows down to allow for more precise positioning, and then it lowers its head to capture the ball under its chin (the *capturing motion*).

Executing the capturing motion without knocking the ball away is a challenge: if the head is lowered when the ball is too far away, the head may knock the ball away; but if it is not lowered in time, the body of the robot may bump the ball away. Furthermore, certain aspects of the acquisition motion

[2]That speed was achieved on the ERS-210A model. It was subsequently retrained on the ERS-7 model used in this paper and achieved an even faster walk of 335–370 mm/sec (depending on the surface on which it is evaluated).

interact, such as the perceived ball distance at which the head should be lowered and the amount that the robot slows down when close to the ball. Parameters like these must therefore be tuned simultaneously. This entire process is time-consuming to perform by hand.

The parameters that control the transition from walking to capturing the ball are as follows:

- `slowdown_dist`: the ball distance (in millimeters) at which slowing down begins;
- `slowdown_factor`: the (multiplicative) factor, in the range [0,1], by which the gait slows down at this point;
- `capture_angle`: the maximum ball angle (in degrees) at which the capturing motion may begin (see Figure 3);
- `capture_dist`: the ball distance (in millimeters) at which the capturing motion begins (if the ball is within the specified angle);
- `turn_cutoff`: the minimum ball angle (in degrees) at which the robot will not move directly toward the ball at all, but instead will turn in place to face the ball more directly. This parameter controls how straight the final part of the robot's approach will be.

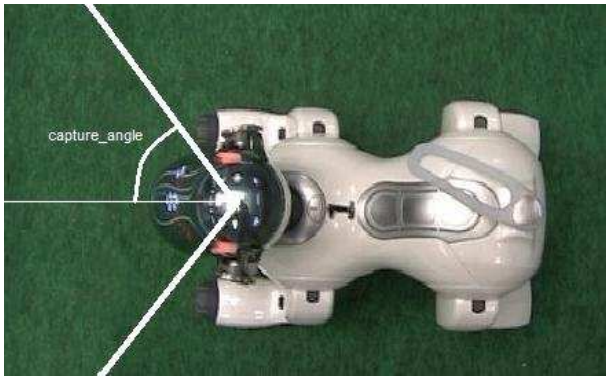

Fig. 3. Illustration of `capture_angle`. If the Aibo believes that the center of the ball is to the right of the thick white lines, then it will continue to turn toward the ball rather than beginning the capturing motion, even if the ball distance is believed to be less than `capture_dist`.

Given this parameterization, we are faced with a parameter optimization problem in five dimensions. Because our policies can be expressed in this way, and because our domain has the same efficiency constraints as that of learning fast locomotion for the Aibo, the policy gradient learning algorithm used to learn the gait (see Section IV) is again a natural choice.

However, one new challenge in learning ball acquisition is defining an appropriate reward signal. The policy gradient algorithm relies on the magnitude of the fitness difference between policies. This magnitude is readily available in the learned walking scenario, because speed provides a natural and continuous measure of fitness. But in the case of ball acquisition, there is no straightforward way to rate a particular policy with regard to "how well" it captures the ball: it either does or it does not.

Therefore, we use a binary reinforcement signal: if the robot captures the ball, it receives a reward of 1; if not, it receives a reward of 0. The Aibo can determine autonomously whether it has captured the ball by trying to put its chin all the way down to its chest and then taking note of the value of the position sensor in its head tilt joint; if the ball is indeed under its chin, the head tilt motor will stop moving before getting to the requested position. During training, the score for a given policy is determined by running a fixed number of trials (12) with that policy and averaging the reinforcement signal over those trials (thus producing a discrete reinforcement signal).

Each trial consists of the robot approaching the ball from a random location on the standard field used in RoboCup, which is surrounded by a short wall designed to keep the ball from leaving the field (see Figure 2). The training procedure is summarized in pseudocode in Figure 4.

```
1:  totalscore ← 0
2:  for j ∈ [1, n] do
3:    locate ball
4:    while ball farther than slowdown_dist do
5:      walk to ball at maxspeed
6:    end while
7:    while ball farther than capture_dist and outside of
      capture_angle do
8:      walk to ball at maxspeed*slowdown_factor
9:    end while
10:   lower head over ball
11:   if head tilt position sensor senses ball then
12:     totalscore ← totalscore + 1
13:     if center of field to robot's left then
14:       kick to left
15:     else
16:       kick to right
17:     end if
18:   end if
19:   turn 180°
20: end for
21: policy_score ← totalscore/n
```

Fig. 4. Method for evaluating policies while learning to approach the ball. n is the number of trials per policy; in our experiments, we used $n = 12$.

One goal of the training procedure is to generate as many trials as possible in the open field, rather than with the ball starting against the wall. The latter trials are somewhat less informative because capturing the ball along the wall is considerably harder; even a good policy will fail much more frequently along the wall, which can lead to a smaller spread of scores among policies. In order to keep the ball in the open field, if the Aibo successfully captures it, it kicks it in whichever direction it estimates is away from the wall (lines 12–15 in Figure 4). Before starting the next trial, the Aibo turns around approximately $180°$ in place in order to knock the ball away from it if it is still close (line 17). Once it has done this, it begins the next trial by searching for the ball and then approaching it with the parameters of the current policy (lines 2–8). Videos depicting the training process in action are available online.[3]

In the formal notation of layered learning, we thus have the following definition of the acquisition layer (L_2):

- $\vec{F_2}$: {BallAngle, BallDistance} $\in \{[-180, 180], [0, \infty)\}$. The five thresholds that comprise an acquisition policy (`slowdown_dist`, etc.) relate to these two sensor readings alone;
- O_2: whether or not to lower the head at the current time;
- T_2: evaluations of mappings from $\vec{F_i}$ to O_i, obtained by repeatedly trying to grasp the ball by the process described above and summarized in Figure 4. In particular, the learned walk (h_1) is used during training;
- M_2: the policy gradient algorithm described in Section IV;
- h_2: the final learned acquisition policy.

All learning is done on the Aibo itself, including all calculations necessary to execute the learning algorithm. Interruptions caused by dead batteries are of little consequence, since the learning algorithm we use has practically no state: if we resume from its last base policy, we will never lose as much as an entire iteration of the algorithm. With the algorithm parameters used in our experiments, a battery typically lasts for the amount of time necessary to complete two iterations, so on average a run requires about 4 battery changes.

IV. The Policy Gradient Algorithm

The learning algorithm common to both learned layers estimates the gradient of the policy's value function in the neighborhood of the current policy via efficient experimentation. It then takes a step in the direction of the estimated gradient and repeats the process. This algorithm is fully specified and compared against alternatives on the learned walk comprising our layer L_1 [5]. This section summarizes the algorithm in task-independent terms and points out some of its advantages for the purpose of ball acquisition.

Starting from a base policy $\{\theta_1, ..., \theta_N\}$, $t-1$ new policies are chosen by selecting one of $\{\theta_i - \epsilon_i, \theta_i, \theta_i + \epsilon_i\}$ randomly for each dimension i, where ϵ_i is a fixed increment particular to dimension i. These t policies (the base policy and the $t-1$ randomly selected policies) are then evaluated for their fitness. Their scores are used to estimate the partial derivative of fitness with respect to each of the N dimensions, which leads to a new base policy.

The estimation of partial derivatives works as follows. For each dimension i, the policies are divided into three sets according to the value of parameter i: if its value is $\theta_i - \epsilon_i$, the policy is in set $S_{-\epsilon,i}$; if it is θ_i, the policy is in set $S_{0,i}$; and if it is $\theta_i + \epsilon_i$, the policy is in set $S_{+\epsilon,i}$. Then the average score over all the policies in each set is computed and used to build an adjustment vector A of size N. For each i, if the average score over the set $S_{0,i}$ is greater than the average score over each of the other two sets, then $A_i = 0$; otherwise, A_i becomes the difference between the average scores over set $S_{+\epsilon,i}$ and set $S_{-\epsilon,i}$. A is then normalized and multiplied by a scalar step size η, so that the policy is adjusted by a fixed amount each time. The above process comprises one iteration of the algorithm. This algorithm is specified in pseudocode in [6]. For the parameters used in learning ball acquisition,

TABLE II

Parameters for the policy gradient algorithm in the ball acquisition learning task

Parameter	Value
Policies per iteration (t)	8
Increment for `slowdown_dist` (ϵ_1)	10mm
Increment for `slowdown_factor` (ϵ_2)	0.1
Increment for `capture_angle` (ϵ_3)	5°
Increment for `capture_dist` (ϵ_4)	10mm
Increment for `turn_cutoff` (ϵ_5)	10°
Scalar step size (η)	2

V. Results

The success of layer L_1 at producing a significantly faster forward gait has been demonstrated in previous work [6]. It remains to demonstrate that, in the layered learning paradigm we present here, L_2 can build upon the gait improvement conferred by L_1. In particular, we hypothesize the ability to learn a significantly improved ball-acquisition behavior to go with the significantly improved gait.

To test this hypothesis, we learn ball acquisition using three gaits learned by separate runs of layer L_1. All three of these learned gaits represent significant improvements in speed over the initial hand-tuned gait. The initial (before learning) ball acquisition policy was hand-tuned for the initial hand-tuned gait from which gait learning began. The ball-acquisition learning paradigm described by layer L_2 was then applied to each of these gaits, and significantly improved acquisition policies were discovered for all three.

Figure 5 shows the learning curve for one of these gaits, which we will refer to as gait A. For this gait, the initial ball acquisition policy acquires the ball roughly 26% of the time, whereas the best learned policy acquires the ball approximately 77% of the time. This improvement was reached in 8 iterations, which requires 768 attempted acquisitions (approximately 3 hours).

Gait A has a speed of approximately 315mm/sec, whereas the initial hand-tuned gait from which it was learned has a speed of approximately 245mm/sec. The gait training process also requires roughly 3 hours [6], [5]. Therefore, with 6 hours of training, our robot's walking speed increased 29% and its reliability at acquiring the ball more than doubled[4] in comparison with the original hand-coded solution.

Table III shows a summary of the ball acquisition policies learned for all three gaits. It also shows the success rate of each when tested on the gait with which it was learned. These success rates were obtained by running 100-trial evaluations of the policy (except for gait A, where the data is the result of all 500 trials run to establish statistical significance on the

[3] `http://www.cs.utexas.edu/~AustinVilla/legged/learned-acquisition/`

[4] The initial ball-acquisition behavior had a success rate of 36% with the initial gait, and was the result of extensive tuning involving the testing of dozens of parameter settings over the course of several days.

TABLE III

POLICY VALUES LEARNED FOR EACH GAIT, AND THE APPROXIMATE SUCCESS RATE OF EACH

Policy	slowdown_dist	slowdown_factor	capture_angle	capture_dist	turn_cutoff	Success rate
Initial	200	0.8	15	110	90	26%/14%/14%
Best: gait A	193	1	32	155	80	77%
Best: gait B	187	1	19	155	69	84%
Best: gait C	228	1	31	129	84	52%

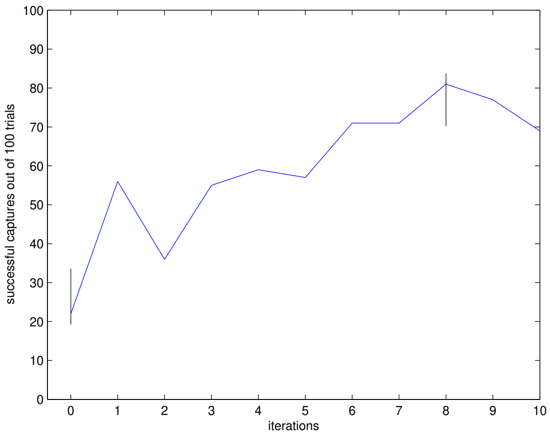

Fig. 5. Progress of acquisition learning on gait A. This learning curve was produced by running 100-trial evaluations on the base policy of each iteration. Error bars (showing the 95% confidence interval) are depicted for the initial and best learned policy; these error bars were obtained by running five 100-trial evaluations on each of these policies.

data in Figure 5). The success rate of the initial hand-coded policy is 26% for gait A, 14% for gait B, and 14% for gait C.

Note that in all cases, the method learned not to slow down at all (`slowdown_factor`=1). When `slowdown_factor` is 1, the parameter `slowdown_dist` has no effect on the robot's behavior, which is presumably why learning resulted in such a wide range of values for this parameter.

The fact that, in all cases, our method learns not to slow down is a demonstration of the advantage that machine learning can bestow because of its unbiased exploration of the space. In hand-tuning, we believed that slowing down would make the ball approach more reliable at the expense of speed, since the estimates of ball distance should change less rapidly if the robot is walking more slowly. However our system, which optimized only for reliability, found that slowing down is in fact a disadvantage: in all learning trials it actively increased the `slowdown_factor` parameter from its initial value of 0.8 to 1.0.

We originally hypothesized that different gaits would require different acquisition policies. This hypothesis was supported by the fact that the initial ball acquisition policy dropped in effectiveness from approximately 36% on the gait for which it was hand-tuned to 14–26% on the learned gaits. However, it turned out not to be the case with these learned gaits and their trained acquisition policies. Rather, upon testing the best acquisition policy learned with gait A on each of the other two learned gaits, there was no significant difference in performance — if anything, the acquisition policy learned on gait A performs better in each case, as shown in Table IV.

Nonetheless, the layered learning paradigm enabled the separation of the learning for the walk and ball acquisition into two distinct phases. Given that they learn most efficiently in different training environments, such a hierarchical approach is an essential component of our successful behavior learning.

TABLE IV

SUCCESS RATES OF BEST NATIVELY LEARNED ACQUISITION POLICY AND BEST ACQUISITION POLICY LEARNED ON GAIT A

Gait	Natively learned	Best on gait A
B	84%	91%
C	52%	53%

VI. CONCLUSION

We have presented an efficient and effective means of learning a high-level behavior on a physical robot. This paper makes two main contributions: i) the first instantiation of layered learning on a physical robot, and ii) a significantly improved grasping behavior achieved via fully autonomous machine learning with all training and computation executed on-board the robot.

The layered learning approach to locomotion and ball acquisition learning that we describe here is very useful in practice. Compared to manually tuning these skills, this method saves time and can generate better policies. Indeed, we used the described automated training paradigms for both the gait and the acquisition in our competitive team development for the RoboCup 2004 robot soccer competitions, finishing in 3rd place (out of 8) at the U.S. Open and reaching the quarterfinals (out of 24) at the international event [10].

In our ongoing research, we aim to identify additional behaviors that can be learned in a similarly autonomous and efficient fashion. Several candidates for an L_3 that builds on the grasping behavior learned in L_2 exist. Currently, for example, all design and tuning of kicks for our RoboCup team is done by hand. If this process could be automated, it would likely save time and might also lead to improved solutions. However, since most kicks begin by grasping the ball, autonomous learning of kicks would be intractable without a good grasping behavior. Another possible candidate for an L_3 that builds on the learned grasping behavior is the tuning of walks that manipulate the ball, such as the one used in the turning-with-ball behavior which makes grasping so crucial in the first place (see Section II-B). Ultimately, we hope to characterize the full range of characteristics of tasks on a mobile robot that may be improved by these methods.

ACKNOWLEDGMENTS

Thanks to the members of the UT Austin Villa team for their efforts in developing the software used as a basis for the work reported in this paper. Special thanks to Nate Kohl for sharing his machine learning infrastructure used for Aibo locomotion. This research was supported in part by NSF CAREER award IIS-0237699 and ONR YIP award N00014-04-1-0545.

REFERENCES

[1] A. Bicchi and V. Kumar. Robotic grasping and contact: A review. In *Proceedings of the IEEE International Conference on Robotics and Automation*, April 2000.

[2] S. M. Gustafson. Layered learning for a cooperative robot soccer problem. Master's thesis, Kansas State University, 2000.

[3] W. H. Hsu and S. M. Gustafson. Genetic programming and multi-agent layered learning by reinforcements. In *Genetic and Evolutionary Computation Conference*, New York,NY, July 2002.

[4] I. Kamon, T. Flash, and S. Edelman. Learning to grasp using visual information. Technical report, The Weizmann Institute of Science, Revhovot, Israel, March 1994.

[5] N. Kohl and P. Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, July 2004.

[6] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2004.

[7] Sony. Aibo robot, 2004. `http://www.sony.net/Products/aibo`.

[8] P. Stone, K. Dresner, S. T. Erdoğan, P. Fidelman, N. K. Jong, N. Kohl, G. Kuhlmann, E. Lin, M. Sridharan, D. Stronger, and G. Hariharan. UT Austin Villa 2003: A new RoboCup four-legged team. Technical Report UT-AI-TR-03-304, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, 2003.

[9] P. Stone, K. Dresner, S. T. Erdoğan, P. Fidelman, N. K. Jong, N. Kohl, G. Kuhlmann, E. Lin, M. Sridharan, D. Stronger, and G. Hariharan. The UT Austin Villa 2003 four-legged team. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *RoboCup-2003: Robot Soccer World Cup VII*. Springer Verlag, Berlin, 2004.

[10] P. Stone, K. Dresner, P. Fidelman, N. K. Jong, N. Kohl, G. Kuhlmann, M. Sridharan, and D. Stronger. The UT Austin Villa 2004 RoboCup four-legged team: Coming of age. Technical Report UT-AI-TR-04-313, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, October 2004.

[11] P. Stone and M. Veloso. Layered learning. In R. L. de Mántaras and E. Plaza, editors, *Machine Learning: ECML 2000 (Proceedings of the Eleventh European Conference on Machine Learning)*, pages 369–381. Springer Verlag, Barcelona,Catalonia,Spain, May/June 2000.

[12] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone. Evolving keep-away soccer players through task decomposition. *Machine Learning*, 2005. To appear.