In *The Nineteenth National Conference on Artificial Intelligence* (AAAI 2004), pp. 611-616, San Francisco, CA, July 2004.

Machine Learning for Fast Quadrupedal Locomotion

Nate Kohl and **Peter Stone**
Department of Computer Sciences, The University of Texas at Austin
1 University Station C0500, Austin, Texas 78712–1188
{nate,pstone}@cs.utexas.edu
http://www.cs.utexas.edu/~{nate,pstone}

Abstract

For a robot, the ability to get from one place to another is one of the most basic skills. However, locomotion on legged robots is a challenging multidimensional control problem. This paper presents a machine learning approach to legged locomotion, with all training done on the physical robots. The main contributions are a specification of our fully automated learning environment and a detailed empirical comparison of four different machine learning algorithms for learning quadrupedal locomotion. The resulting learned walk is considerably faster than all previously reported hand-coded walks for the same robot platform.

Introduction

The ability to deploy a fully autonomous robot in an unstructured, dynamic environment (the proverbial *real world*) over an extended period of time remains an open challenge in the field of robotics. Considerable progress is being made towards many components of this task including physical agility, power management, and on-board sensor technology. One such component that has drawn considerable interest recently is the ability for a robot to autonomously *learn* to improve its own performance (Ng *et al.* 2004; Bagnell & Schneider 2001; Zhang & Vadakkepat 2003). Despite this interest, considerable work remains due to the difficulties associated with machine learning in the real world.

Compared to other machine learning scenarios such as classification or action learning in simulation, learning on physical robots presents several formidable challenges, including the following.

Sparse Training Data: It is often prohibitively difficult to generate large amounts of data due to the maintenance required on robots, such as battery changes, hardware repairs, and, usually, constant human supervision. Thus, learning methods designed for physical robots must be effective with small amounts of data.

Dynamical Complexity: The dynamics of many robotic control tasks are too complex for faithful simulation to be possible. Furthermore, robots are inherently situated in an unstructured environment with unpredictable sensor and actuator noise, namely the real world. Thus, even when off-line simulation is possible, it can never be fully reflective of the target environment.

In this paper, we overcome these challenges for one concrete complex robot task, namely legged locomotion. Using a commercially available quadruped robot, we fully automate the training process (other than battery changes) and employ machine learning algorithms that are sufficiently data efficient to enable productive learning on physical robots in a matter of hours. The resulting learned walk is considerably faster than all previously reported hand-coded walks for the same robot platform.

This paper contributes both a specification of our fully automated learning environment and a detailed empirical comparison of four different machine learning algorithms for learning quadrupedal locomotion. The remainder of the paper is organized as follows. First, we introduce the parameterized walk which our learning process seeks to optimize. We then specify our four learning approaches, and follow with detailed empirical results. We close with a discussion of their implications and possible avenues for future work.

A Parameterized Walk

The Sony Aibo ERS-210A is a commercially available robot that is equipped with a color CMOS camera and an optional ethernet card that can be used for wireless communication. The Aibo is a quadruped robot, and has three degrees of freedom in each of its four legs (Sony 2004).

At the lowest level, the Aibo's gait is determined by a series of joint positions for the three joints in each of its legs. An early attempt to develop a gait by Hornby et al. (1999) involved using a genetic algorithm to learn a set of low-level parameters that described joint velocities and body position.[1] More recent attempts to develop gaits for the Aibo have involved adopting a higher-level representation that deals with the trajectories of the Aibo's four feet through three-dimensional space. An inverse kinematics calculation is then used to convert these trajectories into joint angles.

Among higher-level approaches, most of the differences between gaits that have been developed for the Aibo stem from the shape of the loci through which the feet pass and the exact parameterizations of those loci. For example, a team from the University of New South Wales achieved the fastest known hand-tuned gait using the high-level approach described above with trapezoidal loci. They subsequently generated an even faster walk via learning (Kim & Uther 2003). A team from Germany created a flexible gait implementation that allows them to use a variety of different shapes of loci (Rofer *et al.* 2003), and the team from the University of Newcastle was able to generate high-velocity gaits using a genetic algorithm and loci of arbitrary shape (Quinlan, Chalup, & Middleton 2003).

Our team (UT Austin Villa, Stone *et al.* 2004) first approached the gait optimization problem by hand-tuning

[1] Developed on an earlier version of the Aibo.

a gait described by half-elliptical loci. This gait performed comparably to those of other teams participating in RoboCup 2003. The work reported in this paper uses the hand-tuned UT Austin Villa walk as a starting point for learning. Figure 1 compares the reported speeds of the gaits mentioned above, both hand-tuned and learned, including that of our starting point, the UT Austin Villa walk. The latter walk is described fully in a team technical report (Stone *et al.* 2004). The remainder of this section describes those details of the UT Austin Villa walk that are important to understand for the purposes of this paper.

Hand-tuned gaits			Learned gaits		
CMU (2002)	Austin Villa (2003)	UNSW (2003)	Hornby (1999)	UNSW (2003)	NUBots (2003)
200	**245**	**254**	**170**	**270**	**296**

Figure 1: Maximum forward velocities of the best gaits (in mm/s) for different teams, both learned and hand-tuned.

The half-elliptical locus used by our team is shown in Figure 2. By instructing each foot to move through a locus of this shape, with each pair of diagonally opposite legs in phase with each other and perfectly out of phase with the other two (a gait known as a trot), we enable the Aibo to walk. Four parameters define this elliptical locus:

1. The length of the ellipse;
2. The height of the ellipse;
3. The position of the ellipse on the x axis; and
4. The position of the ellipse on the y axis.

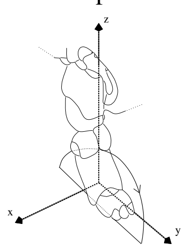

Figure 2: The elliptical locus of one the Aibo's feet. The half-ellipse is defined by length, height, and position in the x-y plane.

Since the Aibo is roughly symmetric, the same parameters can be used to describe loci on both the left and right side of the body. To ensure a relatively straight gait, the length of the ellipse is the same for all four loci. Separate values for the elliptical height, x, and y positions are used for the front and back legs. An additional parameter which governs the turning rate of the Aibo is used to determine the skew of all four ellipses in the x-y plane, a technique introduced by the UNSW team (Hengst *et al.* 2001).[2] The amount of skew is determined by the product of the angle at which the Aibo wishes to move and this skew multiplier parameter.

The following set of 12 parameters define the Aibo's gait:

- The front locus (3 parameters: height, x-pos., y-pos.)
- The rear locus (3 parameters)
- Locus length
- Locus skew multiplier in the x-y plane (for turning)
- The height of the front of the body
- The height of the rear of the body
- The time each foot takes to move through its locus
- The fraction of time each foot spends on the ground

[2]Even when walking directly forward, noise in an Aibo's motions occasionally requires that the four ellipses be skewed to allow the Aibo to execute small turns in order to stay on course.

During the American Open tournament in May of 2003,[3] UT Austin Villa used a simplified version of the parameterization described above that did not allow the front and rear heights of the robot to differ. Hand-tuning these parameters generated a gait with a velocity of 140 mm/s. After allowing the front and rear height to differ, the Aibo was tuned to walk at 245 mm/s in the RoboCup 2003 competition.[4] Applying machine learning to this parameter optimization process, however, allowed us to significantly improve the speed of the Aibos, as described in the following section.

Learning the Walk

Given the parameterization of the walk defined in the previous section, our task amounts to a parameter optimization problem in a continuous 12-dimensional space. For the purposes of this paper, we adopt forward speed as the sole objective function. That is, as long as the robot does not actually fall over, we do not optimize for any form of stability (for instance in the face of external forces from other robots).

This optimization is performed from a computer that is connected via wireless ethernet to the Aibos. All of the policy evaluations take place on actual robots, without the use of a simulator. Previous attempts at learning Aibo gaits involved running each experiment directly on the Aibo, which imposed certain time limitations on the learning process (Kim & Uther 2003). A more decentralized approach allows us to distribute the learning process over multiple Aibos and prevents the loss of data due to battery swaps and mechanical failure. The only human intervention required during an experiment is replacing discharged batteries, an event which occurs about once an hour. We use three simultaneously walking Aibos for our experiments, but our approach scales naturally to arbitrary numbers of robots.

We evaluate the efficacy of a set of parameters by sending those parameters to an Aibo and instructing it to time itself as it walks between two fixed landmarks (Figure 3). More efficient parameters result in a faster gait, which translates into a lower time and a better score. After completing a trial, the Aibo sends the resulting score back to the host computer and prepares itself for a new set of parameters to evaluate. Since there is significant noise in each trial, each set of parameters is evaluated three times. The resulting score for that set of parameters is computed by taking the average of the three trials.[5]

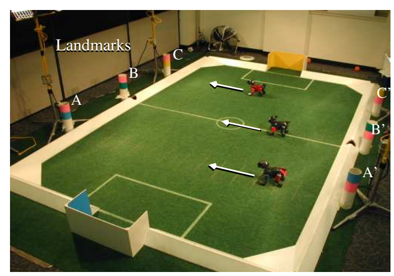

Figure 3: The training environment for our experiments. Each Aibo times itself as it moves back and forth between a pair of landmarks (A and A', B and B', or C and C').

[3]`http://www.cs.cmu.edu/~AmericanOpen03/`

[4]Thanks to Daniel Stronger for hand-tuning the walks to achieve these speeds.

[5]There is video of the training process at: `www.cs.utexas.edu/~AustinVilla/legged/learned-walk/`

The following sections describe four machine learning algorithms that were implemented to perform the learning for this parameter optimization problem: a hill climbing algorithm, the amoeba algorithm, a genetic algorithm, and a policy gradient algorithm. For ease of comparison, we use some common notation in their presentation. Policies, denoted as π, are vectors of 12 parameters $\{\theta_1, \ldots, \theta_{12}\}$. Each algorithm evaluates t policies per iteration of that algorithm, and will typically change the jth parameter of any policy by at most ϵ_j during any iteration.

Hill Climbing

A hill climbing algorithm is one of the simplest ways to approach a parameter optimization problem. The N-dimensional hill climbing algorithm implemented for this work starts from an initial parameter vector $\pi = \{\theta_1, \ldots, \theta_N\}$ (where $N = 12$ in this case) and generates t random policies in the vicinity of π. Each of these policies $\{R_1, R_2, \ldots, R_t\}$ is chosen such that each $R_i = \{\theta_1 + \Delta_1, \ldots, \theta_N + \Delta_N\}$ and each Δ_j is chosen randomly to be either $+\epsilon_j$, 0, or $-\epsilon_j$. After evaluating each R_i, the highest-scoring R_i is chosen as a new starting point and the process is repeated. Pseudocode for this algorithm is provided in Figure 4.

```
π ← InitialPolicy
while !done do
   {R_1, R_2, ..., R_t} = t random perturbations of π
   evaluate( {R_1, R_2, ..., R_t} )
   π ← getHighestScoring({R_1, R_2, ..., R_t})
end while
```

Figure 4: Pseudocode for the hill climbing algorithm. During each iteration of the main loop t policies are sampled near π. The highest-scoring policy is chosen as a new starting point, and the process is repeated.

Amoeba

The downhill simplex (or "amoeba") algorithm is a multidimensional optimization method that involves moving a simplex of points through a high-dimensional space via a series of geometrical transformations (Press 1988). A *simplex* is a geometrical figure consisting of $N + 1$ points in an N dimensional space. For example, in two dimensions a simplex is a triangle. The simplex of the amoeba algorithm is initially composed of $N + 1$ random policies in the vicinity of an initial policy π. These initial policies are generated in the same manner as the policies for the Hill Climbing algorithm, described above.

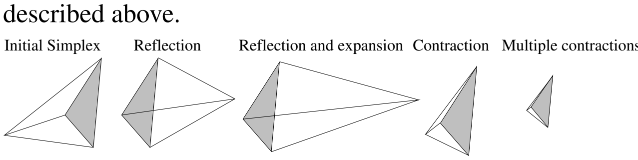

Figure 5: An example of the geometric transformations that would be used by the amoeba algorithm in three dimensions.

The amoeba algorithm then takes a series of steps, most of which simply reflect the lowest-scoring point of the simplex through the opposite face of the simplex. Most of these reflections preserve the volume of the simplex, which helps prevent the search process from stagnating around one policy. When possible, the amoeba algorithm expands the simplex. When the algorithm is faced with a narrow valley through which it must pass, all of the points of the simplex are contracted around its best point. After it has been contracted past a certain point, the simplex is re-initialized around the best point in a manner similar to how it is initialized at the start of the algorithm. An example of the geometric manipulations that would be performed on the simplex in three dimensions is given in Figure 5, and pseudocode for the amoeba algorithm is shown in Figure 6.

```
π ← InitialPolicy
Simplex ← N + 1 random perturbations of π
while !done do
   bestPoint ← determineBestPoint(Simplex)
   secondWorst ← determineSecondWorst(Simplex)
   score ← reflect(Simplex)
   if score > bestPoint.score                // good reflection, extend
   then
      reflectAndExpand(Simplex)
   else if score < secondWorst.score         // bad reflection, contract
   then
      score ← contract(Simplex)
      if score < secondWorst.score           // still bad, contract
      then
         if numContractions < MAX then
            contractAll(Simplex)
            numContractions++
         else
            Simplex = N + 1 perturbations of bestPoint //or reset
            numContractions = 0
         end if
      end if
   end if
end while
```

Figure 6: Pseudocode for the amoeba algorithm. A simplex undergoes various transformations in the search for an optimal policy.

Genetic Algorithm

The standard genetic algorithm searches for an optimal policy by applying genetic operators to a population of policies. Policies that perform well are rewarded and proliferate through the population, whereas policies that perform poorly are removed from the population (Mitchell 1996). Our genetic algorithm maintains a population of t policies, and uses the genetic operators of mutation and multi-point crossover to manipulate the policies in the population.

The population of policies is divided into a fixed number of species, where each species contains policies that are similar to each other. This technique, called *speciation*, allows the genetic algorithm to perform *explicit fitness sharing* (Goldberg & Richardson 1987). Explicit fitness sharing forces similar members of the same species to "share" one representative score, thereby penalizing species with a large number of policies. This allows new species to form even if they do not perform as well as other, larger, species.

The mutation operator acts on a policy π by randomly altering each parameter of that policy θ_j with some probability ρ. The actual amount by which any parameter is altered is determined by drawing a number from a Gaussian distribution cut off after three standard deviations. The distribution is scaled to generate values between $-\epsilon_j$ and $+\epsilon_j$.

The crossover operator generates a new policy from two "parent" policies by performing multi-point recombination on the parameter strings of the two parents. An example of the crossover process is given in Figure 7.

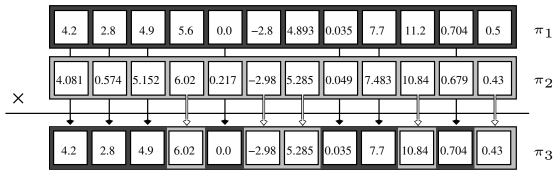

Figure 7: An example of the crossover operator for the genetic algorithm. A new policy (π_3) is created through multi-point recombination of two parent policies (π_1 and π_2).

Policy Gradient Algorithm

Our implementation of the N-dimensional policy gradient algorithm builds on the hill climbing algorithm described above. Initially, a collection of t policies is generated in the vicinity of an initial policy, π. Instead of focusing the search on the best policy at each step, however, the policy gradient algorithm estimates the gradient of the objective function in the parameter space and follows it towards a local optimum.

Since we do not know anything about the true functional form of the policy, we cannot calculate the gradient exactly. Furthermore, empirically estimating the gradient by sampling can be computationally expensive if done naively, given the large size of the search space and the temporal cost of each evaluation. Given the lack of accurate simulators for the Aibo, we are forced to perform the learning entirely on real robots, which makes efficiency a prime concern.

These concerns prompted us to develop an efficient method of estimating the policy gradient. This method can be considered a degenerate form of standard policy gradient reinforcement learning techniques (Sutton *et al.* 2000; Baxter & Bartlett 2001) in that the control policy is purely open loop and not a function of the robot's sensory input. Like these more general techniques, our approach will only converge towards a local optimum. In contrast, some action-value reinforcement learning algorithms, such as Q-learning provably converge to the globally optimal policy (Watkins 1989). However, Q-learning, which is designed for Markov decision processes, is not directly applicable to our problem, which features open-loop control and no notion of "state".

Our approach starts from an initial parameter vector $\pi = \{\theta_1, \ldots, \theta_N\}$ (where $N = 12$ in our case) and proceeds to estimate the partial derivative of π's objective function with respect to each parameter. We do so by first evaluating t randomly generated policies $\{R_1, R_2, \ldots, R_t\}$ near π, such that each $R_i = \{\theta_1 + \Delta_1, \ldots, \theta_N + \Delta_N\}$ and each Δ_j is chosen randomly to be either $+\epsilon_j$, 0, or $-\epsilon_j$. Each ϵ_j is a fixed value that is small relative to θ_j.

After evaluating the speed of each R_i, we estimate the partial derivative in each of the N dimensions. This is accomplished by grouping each R_i into one of three sets for each dimension n:

$$R_i \in \begin{cases} S_{+\epsilon,n} & \text{if the } n\text{th parameter of } R_i \text{ is } \theta_n + \epsilon_n \\ S_{+0,n} & \text{if the } n\text{th parameter of } R_i \text{ is } \theta_n + 0 \\ S_{-\epsilon,n} & \text{if the } n\text{th parameter of } R_i \text{ is } \theta_n - \epsilon_n \end{cases}$$

We then compute an average score $Avg_{+\epsilon,n}$, $Avg_{+0,n}$, and $Avg_{-\epsilon,n}$ for $S_{+\epsilon,n}$, $S_{+0,n}$, and $S_{-\epsilon,n}$, respectively. These three averages give us an estimate of the benefit of altering the nth parameter by $+\epsilon_n$, 0, or $-\epsilon_n$. Note that in expectation, there will be $t/3$ of the t policies with each of the three possible parameter settings, though there will be some random variation. We use these scores to construct an adjustment vector A of size N, where

$$A_n = \begin{cases} 0 & \text{if } Avg_{+0,n} > Avg_{+\epsilon,n} \text{ and} \\ & \quad Avg_{+0,n} > Avg_{-\epsilon,n}, \\ Avg_{+\epsilon,n} - Avg_{-\epsilon,n} & \text{otherwise} \end{cases}$$

We normalize A and multiply it by a scalar step-size η (we used a value of $\eta = 2$ to offset the relatively small values of each ϵ_j) so that our adjustment will remain a fixed size each iteration. Finally, we add A to π, and begin the next iteration. Pseudocode for this policy gradient algorithm is shown in Figure 8.

$\pi \leftarrow InitialPolicy$
while !done **do**
  $\{R_1, R_2, \ldots, R_t\} = t$ random perturbations of π
  evaluate($\{R_1, R_2, \ldots, R_t\}$)
  for $n = 1$ to N **do**
    $Avg_{+\epsilon,n} \leftarrow$ average score for all R_i that have a positive perturbation in dimension n
    $Avg_{+0,n} \leftarrow$ average score for all R_i that have a zero perturbation in dimension n
    $Avg_{-\epsilon,n} \leftarrow$ average score for all R_i that have a negative perturbation in dimension n
    if $Avg_{+0,n} > Avg_{+\epsilon,n}$ and $Avg_{+0,n} > Avg_{-\epsilon,n}$ **then**
      $A_n \leftarrow 0$
    else
      $A_n \leftarrow Avg_{+\epsilon,n} - Avg_{-\epsilon,n}$
    end if
  end for
  $A \leftarrow \frac{A}{|A|} * \eta$
  $\pi \leftarrow \pi + A$
end while

Figure 8: Pseudocode for the N-dimensional policy gradient algorithm. During each iteration of the main loop t policies are sampled near π to estimate the gradient around π, then π is moved by an amount of η in the most favorable direction.

Results

Figure 9 shows the progress of the four algorithms during training. Each curve is an average over 2 runs, and represents the best policy found at each instant in time. The genetic algorithm and amoeba algorithm were able to accomplish some learning and improve on the initial hand-tuned gait. The hill climbing and policy gradient algorithms yielded much better results, and both of these methods were able to offer better performance than our best hand-tuned walk. After 350 evaluations the policy gradient algorithm generated a gait that moved at 291 mm/s, faster than all current hand-tuned gaits and among the fastest learned gaits (in parallel with our work, a learned gait has been reported at 296 mm/s by Quinlan, Chalup, & Middleton, 2003).

Both the genetic algorithm and the amoeba algorithm performed fairly poorly, despite the fact that by many standards they would be considered to be more sophisticated algorithms than the hill climbing and policy gradient algorithms. To investigate this surprising result, we analyzed each algorithm in terms of the amount of the search space that each al-

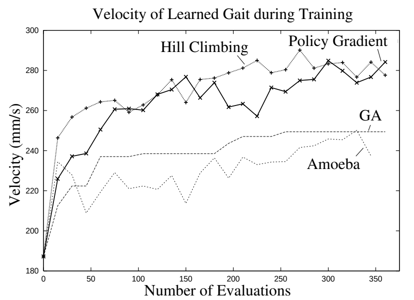

Figure 9: The velocity of the best gait from each iteration during training for the four methods. The policy gradient method was able to generate a gait that is faster than current hand-coded gaits and among the fastest learned gaits.

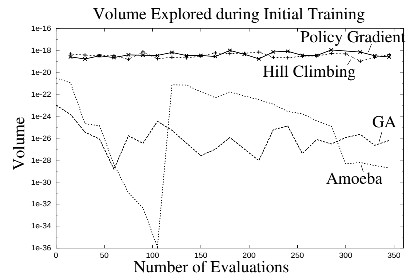

Figure 10: The amount of volume of the search space that each method explored during training. The genetic algorithm and the amoeba algorithm encompass much less volume than the hill climbing and policy gradient algorithms.

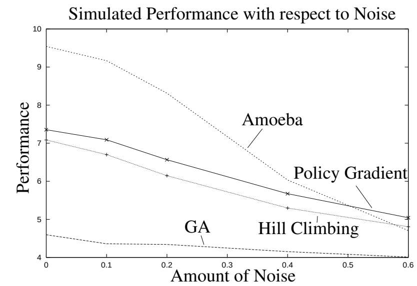

Figure 11: The performance of the four learning algorithms in a simulated domain. The performance of the amoeba algorithm is high when there is little noise in the objective function, but decreases rapidly as noise increases.

gorithm encompassed at each instant in time during training. This value was measured for each algorithm by computing the volume of a convex hull around all of the policies at a given point in time. The results of this analysis, which are shown in Figure 10, indicate that both the genetic algorithm and the amoeba algorithm suffer from rather low coverage rates, compared to the more successful hill climbing and policy gradient algorithms. In an effort to increase coverage for these two algorithms, we increased the mutation rate of the genetic algorithm and increased the rate at which the simplex was re-centered around the best policy for the amoeba algorithm. After these modifications, the performance of the genetic algorithm increased slightly and the performance of the amoeba algorithm improved to a level comparable to that of the hill climbing and policy gradient algorithms.

Despite these improvements, our results indicate that simpler algorithms perform as well as, or better than, more complicated algorithms in this domain. To further understand this phenomenon, we compared the four algorithms in an abstract simulation. We replaced our original objective function of maximizing forward velocity with an average over a set of ten mathematical functions with a variable amount of noise. Each of these functions had the same dimensionality as the original objective function (12 inputs and 1 output) and consisted of an arbitrarily chosen mix of constant, linear, trigonometric, and sigmoidal components. This set of functions was meant to represent a variety of different optimization problems, with the hope that lessons learned from the relative performance of algorithms on this simulated set of functions would translate to our target domain of velocity maximization for the Aibo. The results of this comparison, shown in Figure 11, show that while the amoeba algorithm performs well when there is little noise present, its performance drops sharply as the level of noise increases. This analysis suggests that our domain may be too noisy for the amoeba algorithm to perform at its best.

Note in Figure 9 that we stopped training each method after 350 evaluations, which amounted to just over 1000 field traversals in about 3 hours. Subsequent evaluations showed no further improvement, suggesting that the learning had plateaued in all cases.

Discussion and Future Work

One of the useful aspects of automating the gait optimization process is that search algorithms often possess less bias than human engineers. For example, our gait was designed after a trot gait, where diagonally opposite legs strike the ground simultaneously. We assumed that the ideal trot gait would keep two feet on the ground at all times. Interestingly enough, the best learned gait defied our expectations by attempting to keep each foot on the ground only 43% of the time. By allowing the learning process to affect a large portion of the gait, we were able to discover parameter settings that we would not have likely found through hand-tuning.

On a similar note, recent work has suggested that due to mechanical limitations and environmental noise, the actual locus that each foot follows is significantly different than the requested half-elliptical locus (Stronger & Stone 2005). Given this discrepancy between the ideal locus and the real locus, a change in parameters describing the ideal locus may not have the intended effect on the real locus. This discrepancy could make hand-tuning difficult for humans, who expect a certain correspondence between parameters and their effects on the gait. Since the learning process is unaware of the semantics of the parameters, it might not suffer as much from discrepancies between the expected loci and the actual loci.

Another benefit of automated learning can arise in situations such that the robots are required to repeatedly switch surfaces. In RoboCup, the surfaces of different playing fields can vary widely in hardness and friction. Repeatedly tuning parameters by hand for each surface could consume a great deal of time from human engineers, whereas automatically learning parameters for various surfaces would require significantly less human effort.

One of the benefits of simple algorithms like the policy gradient algorithm and the hill climbing algorithm is that they could be considered a form of multi-robot learning, in that it would be relatively straightforward to implement these algorithms without the use of a central controller. Since the evaluations that each Aibo performs are generated randomly, the algorithm that each Aibo executes could be run almost completely independently of the other Aibos. The only requirement would be that each Aibo communicate the results of the evaluations that it performs to the other Aibos and that they have a common notion of the parameter t which governs the number of policies evaluated per iteration. After finding out the results of t evaluations, each robot could then independently perform the calculation to determine the next policy π and continue with the next iteration. In contrast, more complicated algorithms like the amoeba algorithm require much stricter control over which policies are evaluated. Thus the robots would need to explicitly coordinate which policies they are to evaluate, and find a way to re-do evaluations that are interrupted by battery changes.

It is important to note that our implementations of all of the algorithms mentioned above require reasonable starting policies. Since we are not explicitly trying to find a stable gait, starting from an unstable gait can lead to mechanical failures that can make it difficult for the Aibos to make much progress. As a result, our method is probably not yet directly applicable to the problem of finding an initial stable walk (e.g. for a bipedal robot).

The fact that the amoeba algorithm performs relatively well given an absence of noise in our simulated experiments suggests that we could improve its performance on the Aibos by finding a way to reduce the noise associated with each evaluation. Possible methods to decrease the noise level include using more sensory input to determine the Aibo's location and averaging over more trials.

Since we distributed the learning process over multiple robots, the policies that we discovered were not tuned to any particular robot. While the hardware that the Aibos are made up of is theoretically homogeneous, cumulative wear and tear over time can lead to significant differences between robots. It is therefore possible that we could achieve further increases in performance by training individual Aibos to fine-tune their parameters.

Another possible avenue for future work involves broadening our analysis to include more existing optimization algorithms. Of particular interest are algorithms like Q2 (Moore *et al.* 1998) and PB1 (Anderson, Moore, & Cohn 2000) which were designed to work well in domains with sparse and noisy data. It would also be interesting to optimize fast omni-directional gaits. While the work presented in this paper has focused on maximizing velocity in a forward direction, methods similar to those presented here could be used to optimize turning gaits or gaits for movement in other directions.

Conclusion

In this paper, we presented a comparison of four algorithms for learning a fast walk on a quadruped robot, namely the Aibo ERS-210A. All of the algorithms were able to improve over initial hand-tuned gaits, and the policy gradient algorithm was able to generate one of the fastest known walks on the Aibo: 291 mm/s. These algorithms allow distributed, efficient policy evaluation in the real world, with all learning happening on the robots. Video of the initial results, training process, and final results are all available on-line.[6]

Acknowledgments

We would like to thank the members of the UT Austin Villa team, and Daniel Stronger in particular, for their efforts in developing the gaits and software mentioned in this paper. This research was supported in part by NSF CAREER award IIS-0237699.

[6] `www.cs.utexas.edu/~AustinVilla/legged/learned-walk/`

References

Anderson, B.; Moore, A.; and Cohn, D. 2000. A nonparametric approach to noisy and costly optimization. In *International Conference on Machine Learning*.

Bagnell, J. A., and Schneider, J. 2001. Autonomous helicopter control using reinforcement learning policy search methods. In *International Conference on Robotics and Automation*, 1615–1620. IEEE Press.

Baxter, J., and Bartlett, P. L. 2001. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research* 15:319–350.

Goldberg, D. E., and Richardson, J. 1987. Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J. J., ed., *Proceedings of the 2nd International Conference on Genetic Algorithms*, 148–154.

Hengst, B.; Ibbotson, D.; Pham, S. B.; and Sammut, C. 2001. Omnidirectional motion for quadruped robots. In Birk, A.; Coradeschi, S.; and Tadokoro, S., eds., *RoboCup International Symposium, Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence LNAI 2377*, 368. Springer.

Hornby, G. S.; Fujita, M.; Takamura, S.; Yamamoto, T.; and Hanagata, O. 1999. Autonomous evolution of gaits with the sony quadruped robot. In Banzhaf, W.; Daida, J.; Eiben, A. E.; Garzon, M. H.; Honavar, V.; Jakiela, M.; and Smith, R. E., eds., *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, 1297–1304. Orlando, Florida, USA: Morgan Kaufmann.

Kim, M. S., and Uther, W. 2003. Automatic gait optimisation for quadruped robots. In *Australasian Conference on Robotics and Automation*.

Mitchell, M. 1996. *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.

Moore, A.; Schneider, J.; Boyan, J.; and Lee, M. S. 1998. Q2: Memory-based active learning for optimizing noisy continuous functions. In Shavlik, J., ed., *Proceedings of the Fifteenth International Conference of Machine Learning*, 386–394. 340 Pine Street, 6th Fl., San Francisco, CA 94104: Morgan Kaufmann.

Ng, A. Y.; Kim, H. J.; Jordan, M. I.; and Sastry, S. 2004. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 17*. MIT Press. To Appear.

Press, W. H. 1988. *Numerical Recipes in C: the art of scientific computing*. Cambridge: Cambridge University Press.

Quinlan, M. J.; Chalup, S. K.; and Middleton, R. H. 2003. Techniques for improving vision and locomotion on the sony aibo robot. In *Proceedings of the 2003 Australasian Conference on Robotics and Automation*.

Rofer, T.; Burkhard, H.-D.; Duffert, U.; Hoffman, J.; Gohring, D.; Jungel, M.; Lotzach, M.; v. Stryk, O.; Brunn, R.; Kallnik, M.; Kunz, M.; Petters, S.; Risler, M.; Stelzer, M.; Dahm, I.; Wachter, M.; Engel, K.; Osterhues, A.; Schumann, C.; and Ziegler, J. 2003. Germanteam robocup 2003. Technical report.

Sony. 2004. Aibo robot. `www.sony.net/Products/aibo`.

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

Stronger, D., and Stone, P. 2005. A model-based approach to robot joint control. In Nardi, D.; Riedmiller, M.; and Sammut, C., eds., *RoboCup-2004: Robot Soccer World Cup VIII*. Berlin: Springer Verlag. 297–309.

Sutton, R.; McAllester, D.; Singh, S.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A.; Leen, T. K.; and Muller, K.-R., eds., *Advances in Neural Information Processing Systems*, volume 12, 1057–1063. The MIT Press.

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, King's College, Cambridge, UK.

Zhang, R., and Vadakkepat, P. 2003. An evolutionary algorithm for trajectory based gait generation of biped robot. In *Proceedings of the International Conference on Computational Intelligence, Robotics and Autonomous Systems*.