QuadraTot

Final Report

Diana Hidalgo, B.S. CS '12
Sarah Nguyen, B.A. CS '13
Jason Yosinski, M.S. CS '11
{djh283,smn64,jy495}@cornell.edu

December 6, 2010

Abstract

This paper presents an array of approaches to optimizing a quadrupedal gait for forward speed. We implement, test, and compare different learning strategies including uniform and Gaussian random hill climbing, policy gradient reinforcement learning[3], Nelder-Mead simplex[4], new predictive methods based on linear and support vector regression, and an evolved neural network (HyperNEAT)[2]. We compare results to a baseline random search method. Many of the methods resulted in walks significantly faster than previously hand-tuned gaits.

Keywords: Learning Control, Walking Robots, Gait Learning, Quadruped

1 Introduction

Applications of walking robots often call for the ability to walk as quickly, efficiently, or with as little power as possible. Often these optimizations are done manually by an expert who designs and tweaks a gait specifically for a given objective. Other groups have used learning methods to generate gaits optimized for some metric. Approaches differ in their starting assumptions, some essentially tweaking the parameters of a hand-tuned model [1], others exploring a reasonably compact parameter space [3], and still others beginning with few assumptions besides periodicity [6].

We aimed to strike a middle ground between these approaches. Our motion generator did not rely on hand-tweaked gaits, but it did use parameterized gaits which, by their nature, impose some assumptions on the answers produced. We then used machine learning to design gaits for a quadruped robot with these models. This paper presents a comparison of the different learning methods implemented. Most methods created walks that are several

times faster than the original hand-tuned gait. We invite readers with short attention spans to view a video of some of our results online here:

`http://www.youtube.com/watch?v=ODoiOj9DdGg`

2 Problem definition

We are testing several different learning methods to design a parametrized gait for a quadruped robot from the Cornell Computational Synthesis Lab.

The output each of the learning algorithms is a function of time, $f(t)$, that outputs a vector of commanded motor positions. This function is generated using a parametrized motion model, described in Section 5.

The robot executes these commands and measures its change in location using the tracking system described in Section 5. The input to the learning algorithms is this measured displacement, which the algorithms attempt to maximize. This displacement is measured for each gait over a constant length run, usually 12 seconds.

A comparison and evaluation of the many different methods available for optimizing the gait of legged robots will be useful for future work on this challenging multidimensional control problem.

3 Method

We use several parameterized motion models that command motors to positions based on a sine wave, creating a periodic pattern. While we investigated several models, for the bulk of our experiments, we used a model whose five parameters are: amplitude, wavelength, scale inner vs outer motors, scale left vs right motors, scale back vs front motors. Each strategy below attempts to choose the best possible parameters for this motion model.

We implemented and tested 8 different learning strategies. All strategies except for the HyperNEAT method[2] were constrained to pick parameters from within predetermined ranges.

- *Random*: This method randomly generates parameter vectors in the allowable range. This strategy is used only as baseline.
- *Uniform random hill climbing*: This method begins by selecting a single random parameter vector. Subsequent iterations generate a neighbor by randomly choosing one parameter to adjust and replacing it with a new value chosen with uniform probability in the allowable range for that parameter. The neighbor is evaluated by running the robot with the newly chosen parameters. If this neighbor results in a longer distance walked than the previous best gait, it is saved as the new best gait. The process is then repeated, always starting with the best gait.

- *Gaussian random hill climbing*: This method works similarly to Uniform random hill climbing, except neighbors are generated by adding random Gaussian noise to the current best gait. This results in all parameters being changed at once, but the resulting vector is always fairly close to the previous best gait. We used independently selected noise in each dimension, scaled such that the standard deviation of the noise was 5% of the range of that dimension.

- *N-dimensional policy gradient descent*: As opposed to the previous methods, this method explicitly estimates the gradient for the objective function. It does this by first evaluating t randomly generated parameter vectors near the initial vector, each dimension of these vectors being perturbed by either $-\epsilon$, 0, or ϵ. Then, for each dimension, it groups vectors into three groups: $-\epsilon$, 0, and ϵ. The gradient along this dimension is then estimated as the average score for the ϵ group minus the average score for the $-\epsilon$ group. Finally, the method creates a new vector by changing all parameters by a fixed-size step in the direction of the gradient.

- *Nelder-Mead simplex method*[4]: The Nelder-Mead simplex method creates an initial simplex with 6 vertices. The initial parameter vector is stored as the first vertex and the other five vertices are created by adding to one dimension at a time one tenth of the allowable range for that parameter. It then tests the fitness of each vertex and based on these fitnesses, it reflects the worst point over the centroid in an attempt to improve it. However, to prevent cycles and becoming stuck in local minima, several other rules are used. In general, the worst vertex is reflected over the centroid. If the reflected point is better than the second worst point and worse than the best point, then the reflected point replaces the worst. If the reflected point is better than the best point, the simplex is expanded in the direction of the reflected point. The better of the reflected and the expanded point replaces the worst point. If the reflected point is worse than the second worst point, then the simplex is contracted away from the reflected point. If the contracted point is better than the reflected point, the contracted point replaces the worst point. If the contracted point is worse than the reflected point, the entire simplex is shrunk [4].

- *Linear regression*: To initialize, this method chooses and evaluates five random parameter vectors. It then fits a linear model from parameter vector to fitness. In a loop, the method chooses and evaluates a new parameter vector generated by taking a fixed-size step in the direction of the gradient for each parameter, and fits a new linear model to all vectors evaluated so far, choosing the model to minimize the sum of squared errors.

- *SVM regression*: Similarly to linear regression, this model starts with several random vectors, but this time they are chosen in a small neighborhood about some initial ran-

dom vector. These vectors (generally 8) are evaluated, and a support vector regression model is fit to the observed fitnesses. To choose the next vector for evaluation, we randomly generate some number (typically 100) of vectors in the neighborhood of the best observed gait, and select for evaluation the vector with the best predicted performance. We suspected that if we always chose the best predicted point out of 100, we may end up progressing along a narrow subspace, prohibiting learning of the true local fitness function. Put another way, we would always choose exploitation of knowledge vs. exploration of the space. To address this concern, we added a parameter dubbed `bumpBy` that added noise to the final selected point before it was evaluated.

Such a method naturally has many tunable parameters, and we endeavored to select these parameters by tuning the method in simulation. To estimate the performance of the algorithm, we ran it against a simulation with a known optimum. The simulated function was in the same five dimensional parameter space, and simply returned a fitness determined as the height of a Gaussian with a random mean. The width of the Gaussian in each dimension was 20% of the range of each dimension, and the maximum value at the peak was 100. Figure 1 shows the learning results on this simulated model using the ultimately selected SVM parameters. Interestingly, a non-zero value of `bumpBy` resulted in better learning than noise free (exploration free) learning.

Ultimately, however, the version of SVM tuned for simulation still did not show competitive performance on the real robot. We tried tuning some parameters on the real robot, but after some amount of tuning, the method still exhibited too little exploration and easily became stuck in local minima.

- *Evolutionary Neural Network (HyperNEAT)[2]*: Near the end of term, we hacked together an interface between HyperNEAT – an implementation of a method for evolving neural networks – and the robot, requiring a slightly modified strategy interface. Preliminary HyperNEAT runs were promising and resulted in several interesting gaits. Unfortunately, the gaits generated by HyperNEAT also tended to stress the robot more than typical gaits had before, and the servos would often overheat and malfunction, requiring restarts. We think these issues may be addressed by adding a small layer between the HyperNEAT strategy and the robot that disallows quickly shifting commanded positions, and we hope to be able to test these methods further once this filter is in place.

4 Related work

Various maching learning techniques have proven to be useful in finding control policies for a wide variety of robots. Kohl and Stone[3] presented a policy gradient reinforcement learning

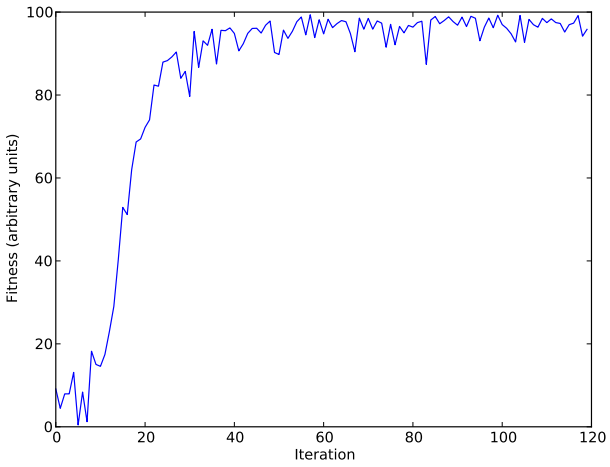

Figure 1: Results for the SVM regression strategy in simulation. This simulation was used to tune the SVM strategy's parameters before trying it on the physical robot. The strategy quickly approaches the maximum simulated fitness of 100.

approach for generating a fast walk on legged robots. We experimented with this method to create a walk for our robot (Policy Gradient Descent). Chernova and Velosa[1] took an evolutionary approach to this problem which we did not implement. Zykov, Bongard, and Lipson[6] describe the evolution of dynamic gaits on a physical robot requiring no prior assumptions about the locomotion pattern beyond the fact that it should be rhythmic.

5 System Architecture and Implementation

The quadruped robot has an on-board computer running Linux. The lower level drivers are in C and the system is implemented in Python. Feedback about distance travelled is provided via an infrared LED mounted on the robot and a Wii remote fixed to the ceiling.

An overview of the code follows.

- `Robot` class: Class wrapper for commanding motion of the robot. The `Robot` class takes care of the robot initialization, communication with the servos, and timing of the runs. In addition, it prevents the servos from ever being commanded to a point outside their normal range (0 - 1023) as well as beyond points where limbs would collide with parts of the robot body. The main class function, `run`, accepts a motion model (any function that takes a time argument and outputs a 9 dimensional position) and will

run the robot using this motion model, including, if desired, smooth interpolation over time for the beginning and end of the run.

- `RunManager` class: Deals with all the details of running the robot, including running the robot multiple times, tracking distance walked via a `WiiTrackClient` member object, and writing results to the log file. Also includes the `explore_dimensions` method to generate plots by varying each parameter independently.

- `Strategy` class: The user has a choice between eight different learning strategies: Random search, uniform random hill climbing, Gaussian random hill climbing, N-dimensional policy gradient descent, Nelder-Mead simplex, linear regression/prediction, SVM regression/prediction, and HyperNEAT evolution. Each strategy must derive from the base `Strategy` class and must implement two methods: `getNext`, for getting the next parameter vector to try, and `updateResults`, for communicating results of a run back to the strategy.

- `MotionModel` class: We implemented several motion models, the main being the `SineModel5` class. All motion models take as input a parameter vector and produce as output a motion model, which is simply a function mapping from time to nine motor positions. The `SineModel5` model commands positions based on a sine wave, creating a periodic pattern. The parameters are: amplitude, wavelength, a multiplier for the inner vs outer motors, multiplier for left vs right motors, and multiplier for back vs front motors. Other similar models were tested, including a seven parameter model which allowed sine waves shifted in time, but these were not used as extensively in our experiments as `SineModel5`.

- `WiiTrackClient` and `WiiTrackServer` classes and hardware: A Wii remote tracks the location of the robot through an infrared LED mounted on top of the robot. A `WiiTrackServer` is run on the robot and continuously tracks its position using the CWiid library[5] to interface with the remote via bluetooth. The `RunManager` then makes a `WiiTrackClient`, which connects via socket to the tracking server and requests position updates periodically. `RunManager` currently gets the robot's position at the beginning of each run and then again at the end and uses this to calculate the net change in position.

6 Experimental Evaluation

6.1 Methodology

The metric for evaluation of the designed gait is speed. We stop each run after plateauing results (no improvement for one third of the policies seen so far). The standard length of

a run designates that it should be stopped after there is no improvement for one half of the policies seen so far, but since all runs took place on the actual robot, without use of a simulator, certain time limitations were imposed on the learning process.

We controlled our experiments from a computer that was connected via a wireless ethernet to the robot. The robot collected data about distance walked automatically on its own. If it walked outside of the Wii remote's viewable area, it informed the user, so the only human intervention required during an experiment was to move the robot back inside the viewable area and to resume the run, which did not interrupt the learning process or result in the loss of data.

We evaluated the efficacy of a set of parameters by sending these parameters to the robot and instructing it to walk for a certain length of time. The robot always began from the same position and returned to the starting position at the end of the run in order to measure true displacement without giving credit for ending in a leaned position. More efficient parameters resulted in a faster gait, which translated into a longer distance walked and a better score. After completing an evaluation, the robot sent the resulting distance walked back to the host computer and prepared itself for a new set of parameters to evaluate.

Each algorithm was run on 3 different initial parameter vectors on the physical robot. We decided to evaluate all methods starting at the same three vector in order to allow for the fair comparison of each algorithm. We evaluate each method based on the amount of improvement seen from the initial parameter vectors, and on the fastest speeds achieved during runs.

The resulting gaits from our algorithms quickly outperformed the original hand-coded walk designed for this robot. The fastest walk, for example, was 4 times faster.

6.2 Results

We have done complete runs of 3 different initial parameter vectors for random search, uniform random hill climbing, Gaussian random hill climbing, policy gradient descent, Nelder-Mead simplex, and linear regression. We also evaluated the SVM regression and HyperNEAT methods, but these methods were more experimental, and thus we do not yet have the same volume of data for these runs. We developed several gaits that were about 4 times faster than the original hand-coded gait. Results are shown in Table 1.

- For vector A, shown in Figure 2, linear regression worked significantly better than the other algorithms, resulting in a gait (27.58 body lengths/minute) that walked over twice as fast as the next best gait – uniform random hill climbing at 11.37 body lengths/minute – and 5.3 times better than the previous hand-coded gait.

- Vector B, depicted in Figure 3, resulted in similarly performing gaits with all the algorithms. The random method got lucky and produced the best gait (17.26 body lengths/minute) and uniform random hill climbing produced the worst (9.44 body

	A	B	C	Average
Previous hand-coded gait	–	–	–	5.16
Random search	6.04	17.26	4.90	9.40
Uniform Random Hill Climbing	11.37	9.44	2.69	7.83
Gaussian Random Hill Climbing	3.10	13.59	13.40	10.03
Policy Gradient Descent	0.68	14.69	3.60	6.32
Nelder-Mead simplex	8.51	13.62	14.83	12.32
Linear Regression	27.58	12.51	1.95	14.01

Table 1: The best gaits found for each starting vector and algorithm, in body lengths per minute.

lengths/minute). The remaining algorithms each produced a gait around 13 body lengths/minute.

- For vector C, shown in Figure 4, simplex and Gaussian random hill climbing each produced a gait that substantially outperformed the other algorithms. Simplex resulted in a gait of nearly 15 body lengths/minute and Gaussian random hill climbing produced a gait of just over 13 body lengths/minute, whereas the other algorithms returned gaits of less than 5 body lengths/minute.

Based on the three trials, as shown in Figure 5, linear regression worked the best, followed by simplex. Gaussian random hill climbing performed about as well as random, but it is unclear whether Gaussian would continue to improve and become much better than random search if allowed to run for more iterations. Uniform random hill climbing and policy gradient descent performed similarly to each other on average, but differed greatly on individual runs.

In addition to running strategies to optimize parameterized gaits, we also wanted to investigate the space of possible gaits. To accomplish this, we selected a parameter vector that resulted in motion, but not an exceptional gait, and plotted performance along each dimension individually. In addition, we duplicated each measurement to be able to estimate the measurement noise at each point. Results for this exploration are shown in Figure 6 through Figure 10. As is shown in the figures, some dimensions are smoother than others, and some measurements are fairly noisy.

6.3 Discussion

Because only three trials were tested with each algorithm and no algorithm consistently outperformed the others, there is a large standard error for each algorithm, as show in

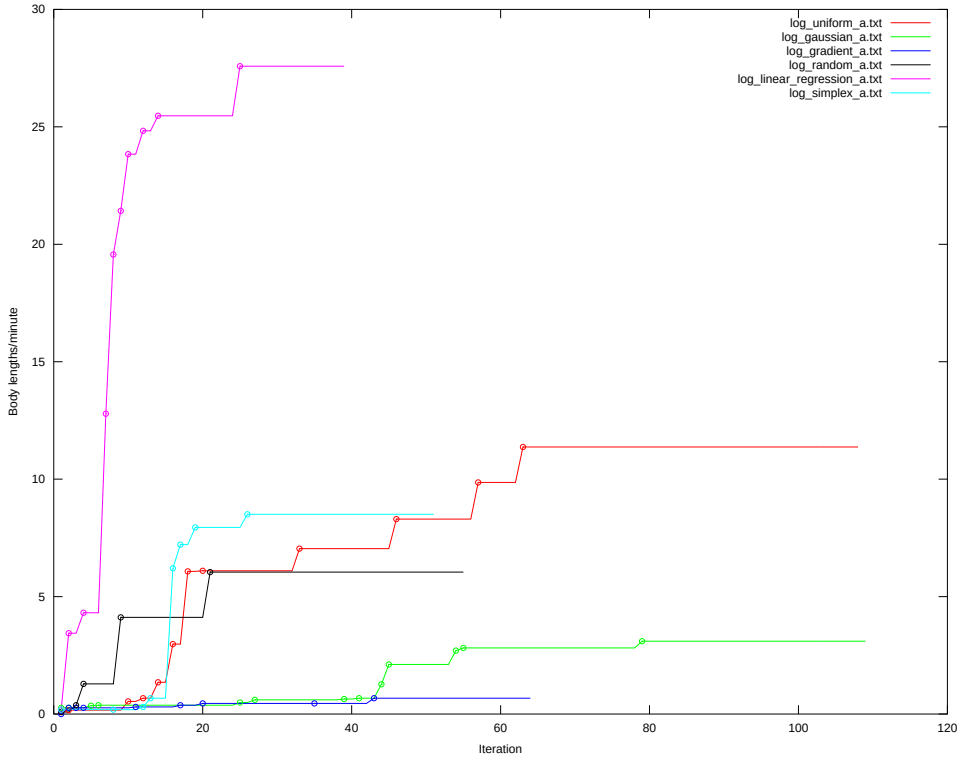

Figure 2: Results for runs beginning with vector A. Linear regression worked significantly better than the other algorithms, resulting in a gait (27.58 body lengths/minute) that walked over twice as fast as the next best gait – uniform random hill climbing at 11.37 body lengths/minute – and 5.3 times better than the previous hand-coded gait.

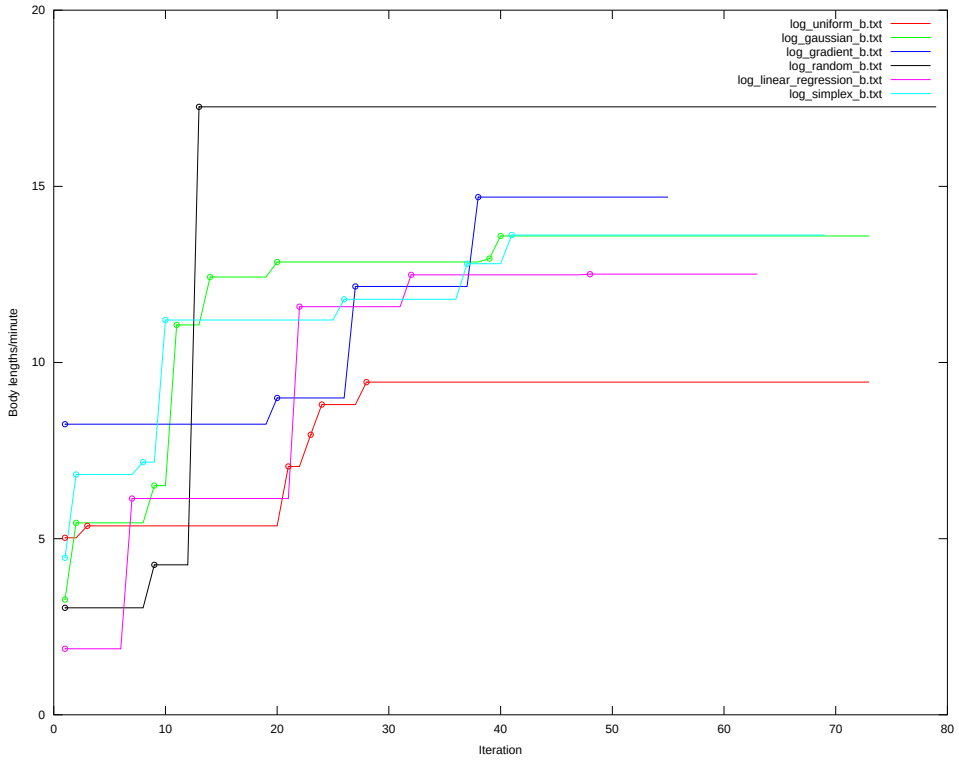

Figure 3: Results for run starting with vector B. All algorithms performed similarly. The random method actually got lucky this time and produced the best gait (17.26 body lengths/minute) and uniform random hill climbing produced the worst (9.44 body lengths/minute). The remaining algorithms each produced a gait around 13 body lengths/minute.

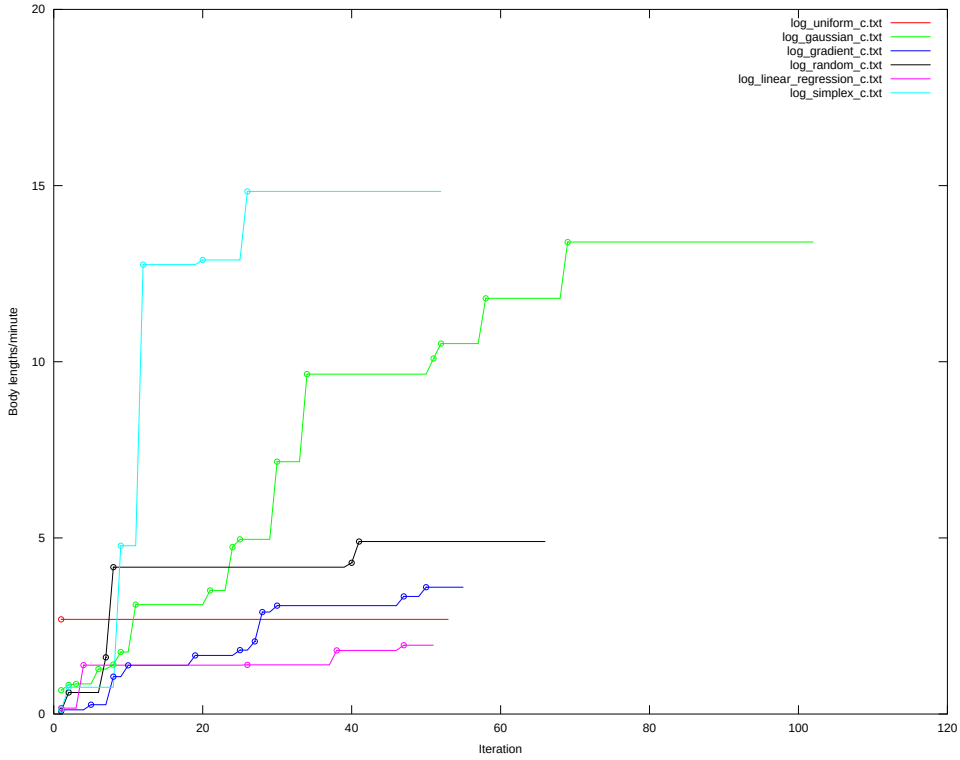

Figure 4: Results for runs starting with vector C. Simplex and Gaussian random hill climbing each produced a gait that substantially outperformed the other algorithms, with around 15 body lengths/minute and 13 body lengths/minute, respectively, whereas the other algorithms returned gaits of less than 5 body lengths/minute.

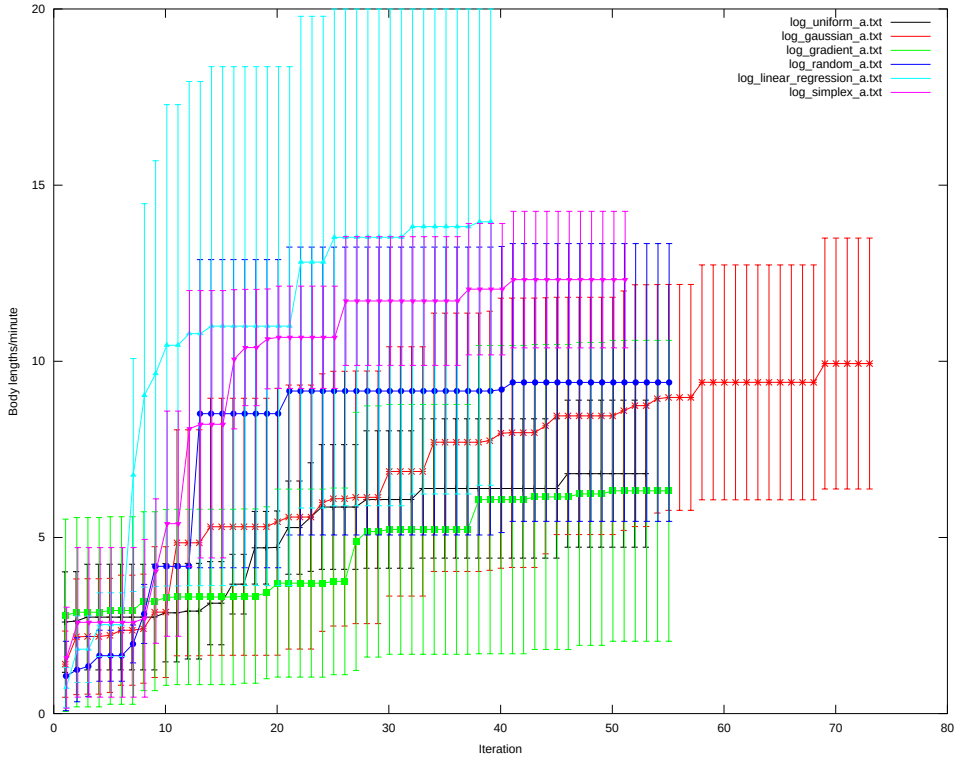

Figure 5: Average results for each method starting with parameter vectors A, B, and C. Error bars are plotted showing the standard error. Linear regression outperformed other methods, followed by simplex. Gaussian random hill climbing performed about as well as random, but it is unclear whether Gaussian would continue to improve and become much better than random search if allowed to run for more iterations. Uniform random hill climbing and policy gradient descent performed similarly to each other on average, but differed greatly on individual runs.

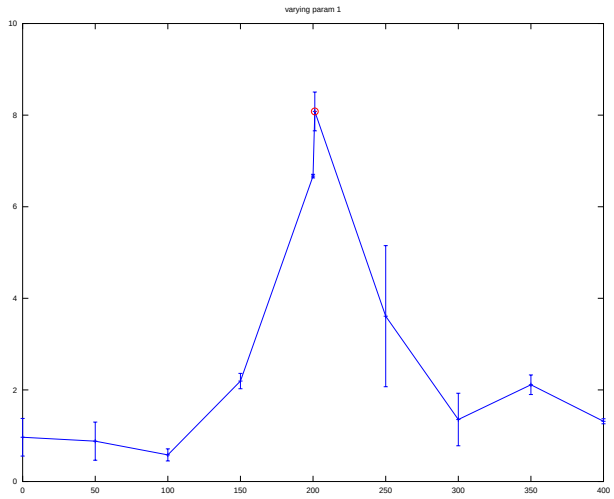

Figure 6: Fitness mean and standard deviation vs. dimension 1. The circle is a common point in Figure 6 through Figure 10

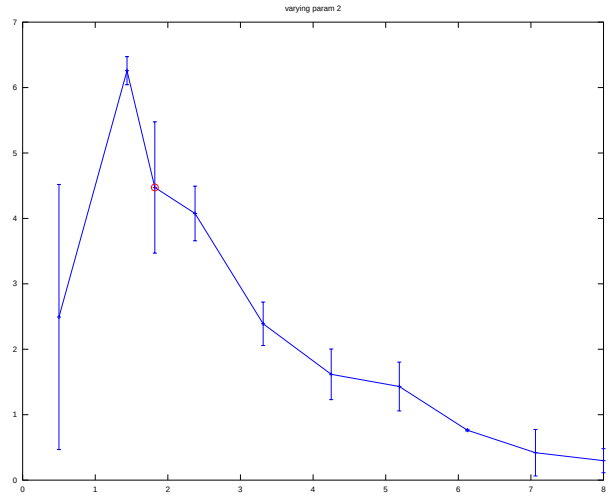

Figure 7: Fitness mean and standard deviation vs. dimension 2. The circle is a common point in Figure 6 through Figure 10

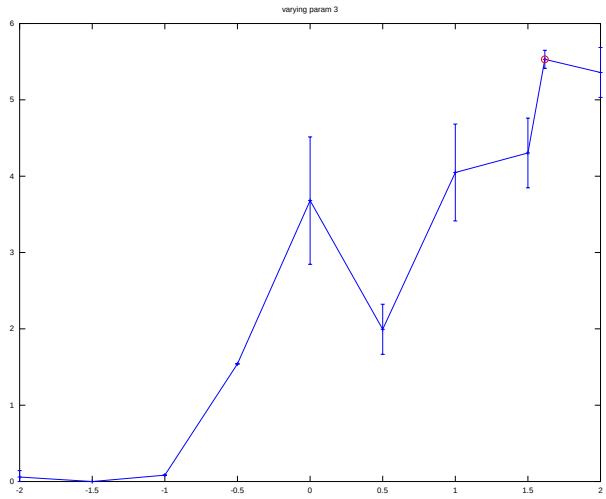

Figure 8: Fitness mean and standard deviation vs. dimension 3. The circle is a common point in Figure 6 through Figure 10

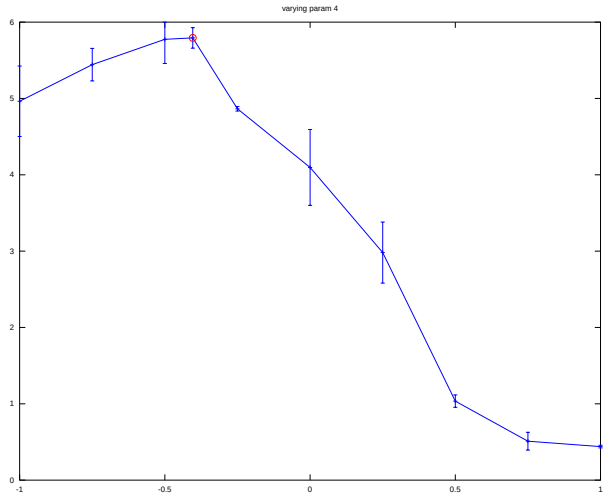

Figure 9: Fitness mean and standard deviation vs. dimension 4. The circle is a common point in Figure 6 through Figure 10

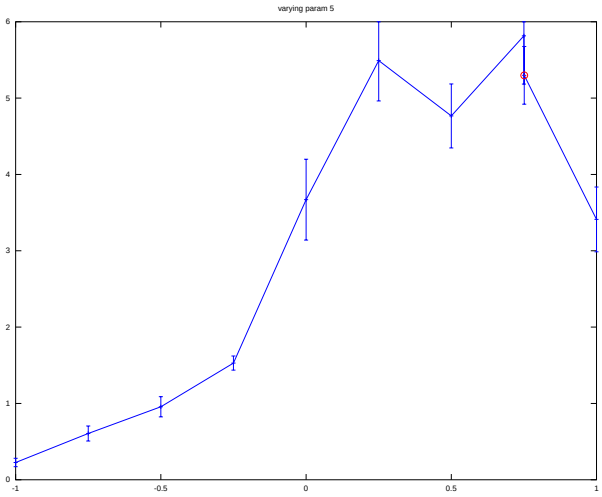

Figure 10: Fitness mean and standard deviation vs. dimension 5. The circle is a common point in Figure 6 through Figure 10

Figure 5. For this reason, it is unclear if any algorithm is superior to another in this application. More trials with each algorithm would be necessary to reach a definitive ranking of the performance of the algorithms.

However, each algorithm discovered at least one gait of over 10 body lengths/minute, including random. For this reason, we speculate that the motion model is at least as critical as the learning algorithm used.

7 Future work

There are several directions in which we could continue our work. First, the error margins for our runs were large, so reducing them by running more trials would lead to more conclusive data. It would also be insightful to run our algorithms on different motion models. We suspect that our choice of motion model influenced the results greatly, as even random choices in the space produced gaits that moved a significant fraction of the time (¿ 5%). It would be interesting to see how learning methods would perform using a model that included a much higher percentage of unproductive gaits. We also intend to experiment further with SVM regression and evolutionary algorithms/HyperNEAT. Some parameter vectors resulted in the robot turning, as opposed to it walking long distances. This could

be an interesting learning goal in future projects. To these ends, we propose the following additions and enhancements:

- More runs and/or longer runs
- Different motion representations
- Better tuning of SVM regression
- Evolutionary algorithms/HyperNEAT[2]
- Learning how to turn

8 Conclusion

We have presented an array of approaches to optimizing a quadrupedal gait for forward speed. We have implemented and tested different learning strategies, including uniform and Gaussian random hill climbing, policy gradient reinforcement learning, Nelder-Mead simplex, several new predictive methods based on linear and support vector regression, and an evolved neural network (HyperNEAT). We have also compared these approaches to random search as a baseline. Many of the methods resulted in walks significantly faster than previously hand-tuned gaits.

Because only three trials were tested with each algorithm and no algorithm consistently outperformed the others, there was a large standard error for each method, as shown in Figure 5. Thus it was unclear if any algorithm was superior to another in this application. More trials with each algorithm would be necessary to reach a definitive ranking. Because each algorithm discovered at least one gait of over 10 body lengths/minute, including random search, we also conjectured that the motion representation for the robot is more integral to forward speed than the learning algorithm. How to learn the motion representation, in addition to its parameters, remains an open problem.

9 Acknowledgments

- Hod Lipson, Cornell Computational Synthesis Lab: adviser.
- Jim Tørreson, University of Oslo: adviser.
- Juan Zagal, University of Chile: designed and printed robot and provided code for hand tuned gait.
- Jeff Clune, CCSL Red Couch: collaborated on HyperNEAT implementation/testing.

- Cooper Bills, Cornell University: assisted with Wii tracker development.
- Anshumali Srivastava, Cornell University: Teaching Assistant.

10 The Team

- Diana Hidalgo:
  - Simplex (Nelder-Mead)
  - `WiiTrackClient` and `WiiTrackServer` classes and hardware
  - Plotting and evaluating results
  - Investigation into using Optitrack IR system for tracking robot
- Sarah Nguyen
  - Random search
  - Uniform random hill climbing
  - Gaussian random hill climbing
  - N-dimensional policy gradient descent
  - Linear regression and prediction
  - `RunManager` and `Strategy` classes
  - Video editing
- Jason Yosinski
  - SVM regression and prediction
  - Evolutionary Neural Network (HyperNEAT)
  - `WiiTrackClient` and `WiiTrackServer` classes and hardware
  - `Robot`, `RunManager`, and `Strategy` classes
  - `Parameterized Motion Model` abstract base class
  - `SineModel` classes
  - `explore_dimensions`
  - Plotting and evaluating results

11 Appendix

A brief description of the code uploaded to the CMS follows:

- `optimize.py`: Main program, determines a strategy to try and runs the robot with that strategy.
- `RunManager.py`: Deals with all the details of running the robot, including choosing an initial parameter, running the robot multiple times, tracking distance walked, and writing to the log file. Also includes the `explore_dimensions` method.
- `Strategy.py`: Contains all the different possible strategies, which will be passed as objects in `optimize.py`.
- `Robot.py`: Implements the `Robot` class, described in Section 5.
- `SineModel.py`: Implements a sine based motion model, described in Section 5.
- `Motion.py`: Motion helper functions.
- `WiiTrackServer.py`: Broadcasts the position of the infrared LED.
- `WiiTrackClient.py`: Connects to the WiiTrackServer to get the current position information.

References

[1] S Chernova and M Veloso. An evolutionary approach to gait learning for four-legged robots. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings*, pages 2562–2567.

[2] Clune J, Beckmann BE, Ofria C, and Pennock RT. Evolving coordinated quadruped gaits with the hyperneat generative encoding. *Proceedings of the IEEE Congress on Evolutionary Computing*, 2764-2771, 2009.

[3] N Kohl and P Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. *IEEE International Conference on Robotics and Automation*, 3:2619–2624, 2004.

[4] Sasa Singer and John Nelder. Nelder-mead algorithm. `http://www.scholarpedia.org/article/Nelder-Mead_algorithm`, 2009.

[5] Donnie Smith. Cwiid. `http://abstrakraft.org/cwiid/`, 2010.

[6] V Zykov, J Bongard, and H Lipson. Evolving dynamic gaits on a physical robot. *Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO*, 4, 2004.