The QOOL Algorithm for fast Online Optimization of Multiple Degree of Freedom Robot Locomotion

Daniel Marbach
January 31th, 2005

Swiss Federal Institute of Technology at Lausanne
Daniel.Marbach@epfl.ch
http://birg.epfl.ch/page32031.html

1. Introduction

This paper concludes the research of my Master Thesis. It presents the QOOL (**Q**uick **O**n-line **O**ptimization of **L**ocomotion) algorithm for fast online optimization of multiple degree of freedom robot locomotion. Previous research uses heuristic search algorithms such as simulated annealing and genetic algorithms to optimize robot locomotion. These algorithms are generally well suited for offline optimization but they are too slow for efficient online adaptation of locomotion. In contrast, QOOL is based on Powell and Brent's algorithms, which are not heuristic. These algorithms guarantee quadratic convergence (i.e. the number of significant figures is *doubled* at each iteration) to a local optimum. For a more detailed introduction and a discussion of previous research and the potential merits of different types of search methods, refer to Chapter 5 of the report [1].

Note that even though I use the terminology of modular robotics, QOOL is well suited to optimize locomotion of any multiple degree of freedom robot. As I argue in the report, the performance of a deterministic optimization algorithm depends heavily on the topology of the fitness landscape. If the search space is full of small local optima, the probability of finding a good locomotion gait is small. In the next section I discuss fitness evaluation and present some preliminary experiments that encouraged me to use Powell's method for optimization of locomotion. In section 3 I briefly present Brent's method for one-dimensional function optimization. Brent's method is a sub algorithm of Powell's multidimensional optimization method, which is outlined in section 4. Finally I shall discuss the QOOL algorithm for online optimization of robot locomotion and present the results.

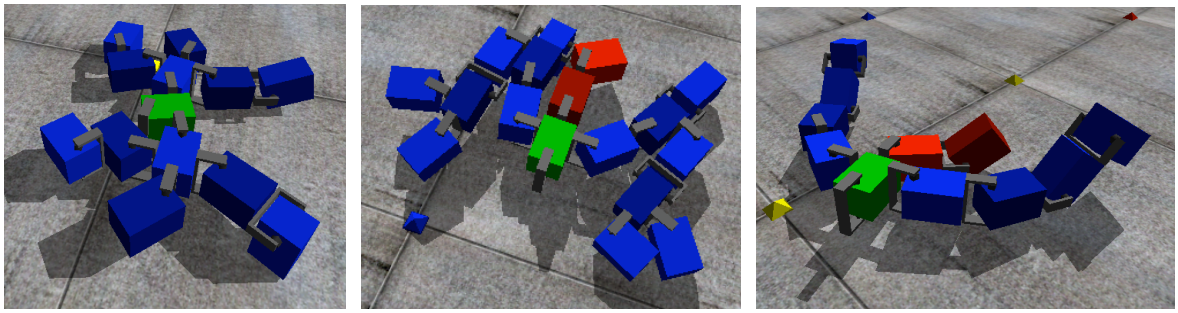

Figure 1: Three YaMoR robots that have been evolved with the co-evolutionary algorithm: A salamander like configuration, a walker and a crawler (from left to right). Head modules are green, spine modules red and limbs blue. Rigid modules are dark, powered modules light. Nonlinear oscillators with unidirectional couplings are used for locomotion control. The parameter space of the first two configurations is well suited for online optimization; the search space of the crawler is highly irregular (refer to Figures 2-4).

2. The fitness function and the topology of the search space

Effective fitness evaluation is essential for online optimization. It is much more complicated than one might think. When changing the locomotion gait of a robot online from gait A to gait B, there is a chaotic transition period. Clearly, this period should not be included in the fitness evaluation of gait B. By analyzing the trajectories of the nonlinear oscillators, one can detect when the CPG converges to a stable limit cycle and start measuring the fitness at that point. However, I did not get satisfactory results with this approach. The problem is, that the mechanical dynamics take some time to adapt to the new gait even after convergence of the CPG (e.g. the robot might still be accelerating when switching from a slow to a fast gait).

Thus, for online fitness evaluation each module should constantly estimate it's average speed during the last period (the period of the oscillations). The current fitness of the robot is defined as the average of the module speeds over the last period. Note that this fitness could easily be computed online by the modular robot provided that the modules are equipped with adequate sensors. The fitness evaluation of a gait is only done when the average speed of each module stabilizes at a constant value over several periods. This indicates that the mechanical dynamics converged. However, even with these precautions the fitness evaluation is quite 'noisy' as you can see on Figures 2-4.

In order to test the fitness evaluation and to visualize the topologies of the fitness landscapes I explored two-dimensional hyper planes within the N-dimensional search space. In other words, all parameters of an evolved locomotion gait are fixed except for two free parameters. For each combination of these two parameters (with small increments) the online fitness is then evaluated. I found that for some robots the fitness landscapes seem to be quite regular. In this case, a heuristic optimization algorithm is clearly not appropriate and Powell's method should find the optimum with few fitness evaluations. However, the fitness landscape of other robots is more irregular, containing local optima. For these configurations, Powell's algorithm might converge to a small local optimum. Refer to Figures 2-4.

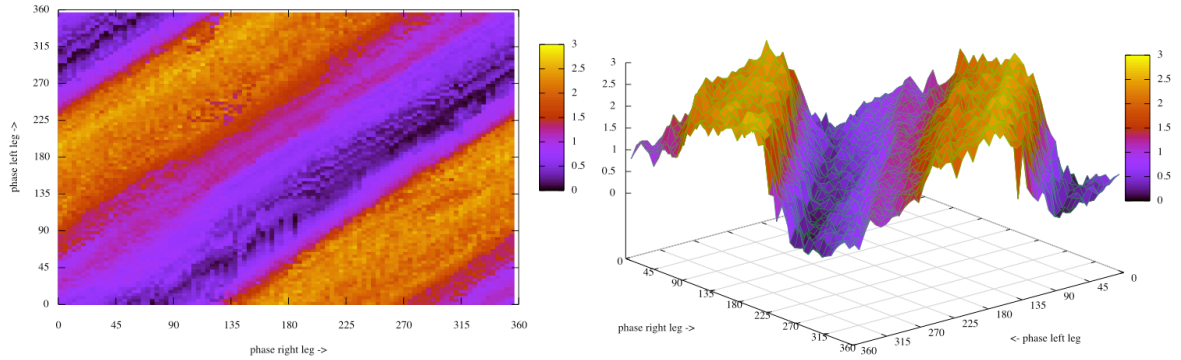

Figure 2: Fitness landscape of the walker (Figure 1). The two free parameters are the phase differences of the two limbs. Both plots represent the same data. Using Brent's method for optimization in one dimension, an optimum is found with only about four fitness evaluations thanks to parabolic interpolation (see Chapter 3).

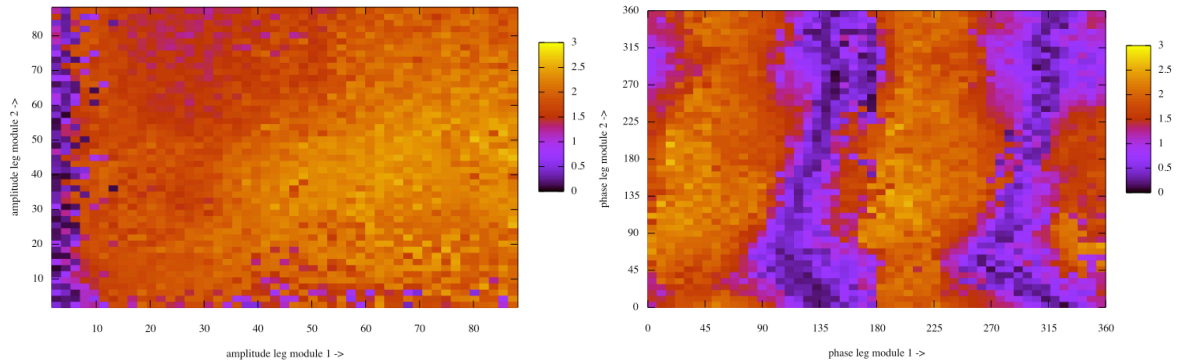

Figure 3: Fitness landscape of the walker (Figure 1). On the left, the two free parameters are the amplitudes of two modules within a limb. On the right, the free parameters are the phases of the same limb modules. Again, the search space seems to be well suited for a quadratically converging optimization algorithm.

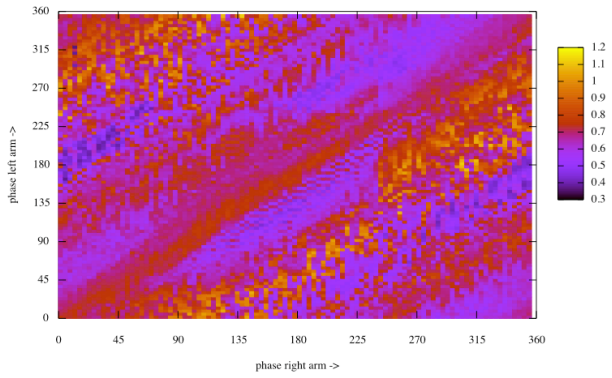

Figure 4: Fitness landscape of the crawler (Figure 1). The free parameters correspond to the phases of the two limbs. The search space is very noisy and unstructured. Depending on the starting point the optimization might converge to a local optimum (i.e. one of the red islands).

3. Brent's method for one-dimensional function optimization

The goal of function optimization is to find x such that $f(x)$ is the highest or lowest value in a finite neighborhood. From now on I just consider the problem of function minimization. Note that function maximization is trivially related because one can minimize $-f$.

The main idea of one-dimensional function optimization is to bracket the minimum with three points $a < b < c$ such that $f(b)$ is less than both $f(a)$ and $f(c)$. In this case and if f is nonsingular, f must have a minimum between a and c. Now suppose that a new point x is chosen between b and c. If $f(b) < f(x)$, the minimum is bracketed by the triplet (a, b, x). In the other case if $f(x) < f(b)$, the new bracketing points are (b, x, c). In both cases, the bracketing interval decreases and the function value of the middle point is the minimum found so far. Bracketing continues until the distance between the two outer points is tolerably small.

One can show that the best strategy for choosing the new point x given a bracketing interval (a, b, c) is the point that is a fraction of 0.38 into the larger of the two sub intervals [2]. The consequence of this strategy is, that the midpoint always lies in a fraction of 0.61 from one end and 0.38 from the other end of the bracketing interval. These fractions are the so-called *golden mean* or *golden section*, hence the name *golden section search.*

"A golden section search is designed to handle, in effect, the worst possible case of function minimization, with the uncooperative minimum hunted down and cornered like a scared rabbit [2]". Most functions are parabolic near the minimum. Thus, instead of choosing x with the golden section strategy, we can fit a parabola through the points $f(a)$, $f(b)$ and $f(c)$ and choose the new point x at the minimum of this parabola. Generally one can get very close to the function minimum in a single leap with this strategy.

However no optimization method is likely to succeed following strictly this approach. Brent's method [2,3] is a clever combination of golden section search and parabolic interpolation. Brent uses golden section search when the function is not cooperative but switches to parabolic interpolation when the function allows it. Obviously, detecting whether the function is cooperative or not is far from trivial. A presentation of the detection scheme goes beyond the scope of this paper; refer to [2,3] for details.

Initially bracketing the minimum is actually at least as complex as finding the minimum with Brent's method. From an initial point a, I choose a second point $b = a + \delta$, where δ is a random number from a Gaussian distribution with mean zero and standard deviation 0.75. Now suppose $f(a) > f(b)$ (rename the points in the other case). The idea is to step downhill with increasing step sizes until bracketing the minimum. In analogy to Brent's minimization method, I use a combination of two strategies: Increasing the step size by the golden ratio and parabolic *extrapolation*. Refer also to the bracketing method described in [2].

Initial bracketing and subsequent minimization can be observed very nicely in the figures of the last chapter.

4. Powell's method in multidimensions

Consider a line defined by a starting point $\mathbf{P}$ and a direction $\mathbf{n}$ in N-dimensional space. It is possible to find the minimum of a multidimensional function f on this line using a one-dimensional optimization algorithm such as Brent's method. In order to do so, one can minimize the one-dimensional function g with Brent's method as described in the previous section:

$$g(\lambda) = f(\mathbf{P}+\lambda\mathbf{n})$$

The minimum λ_{min} of function g corresponds to the minimum $\mathbf{P}+\lambda_{min}\mathbf{n}$ of the multidimensional function f on the line with direction $\mathbf{n}$ through $\mathbf{P}$.

Direction-set methods for multidimensional function minimization consist of sequences of such line minimizations. The methods differ by the strategies in choosing a new direction for the next line minimization at each stage. Some methods require gradient calculations for the choice of successive directions. Even though these methods are more powerful than those that do without the gradient, numerically estimating the gradient of the fitness function would take too much time because it involves many fitness evaluations. For this reason I chose Powell's algorithm [2,3], which is a direction-set method that does not rely on the gradient.

Powell's method starts with the unit vectors $\mathbf{e_1}$, $\mathbf{e_2}$, … $\mathbf{e_N}$ of the N-dimensional search space as a set of directions. One iteration of the algorithm does N line minimizations along the N directions in the set. Using the unit vectors as direction set works actually fine for most functions. However, in some cases convergence might be very slow. Consider a function with a 'valley', for example:

$$f(\mathbf{x}) = |\mathbf{x}| + \sum_{i=1}^{N} (x_i - x_{i-1})^2 \qquad (1)$$

This function is plotted for N = 2 (two dimensions) in Figure 5. The first term of f implicates that the function value increases with the distance from the origin. The consequence of the second term is a 'valley' along x = y that descends towards the origin. The global minimum is the origin.

In order to test my implementation of Powell's method I optimized f using different numbers of dimensions. The two-dimensional case is nice to illustrate the algorithm, see

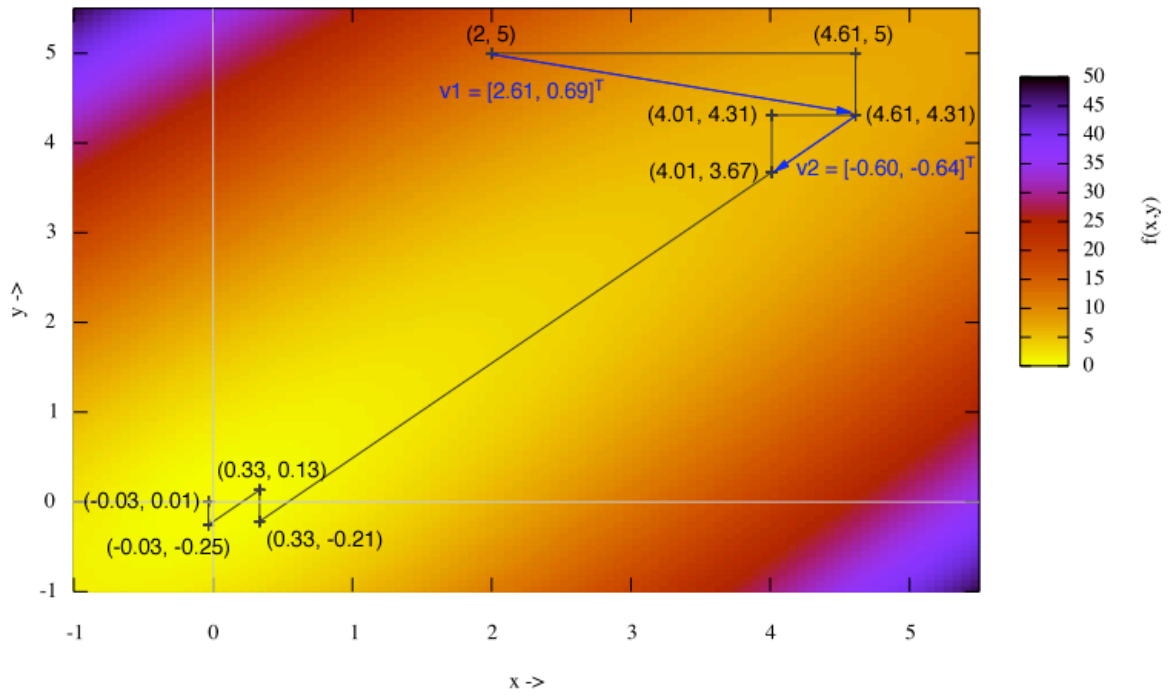

Figure 5: The color corresponds to the function value of f given in Equation 1 (two-dimensional case). Succeeding line minimizations by Powell's method starting at point (2, 5) are plotted in black. For details, refer to the main text.

Figure 5. Starting at the initial point (2, 5), the first line minimization along the direction given by the unit vector $[1, 0]^T$ takes us to the point (4.61, 5). From this point the second line minimization along $[0, 1]^T$ completes the first iteration. As you can see on Figure 5, repeated line minimizations along the unit vectors would involve many iterations because the minimum would be approached in small steps such as the one of iteration 2 from (4.61, 4.31) to (4.01, 3.67).

After each iteration, Powell's method checks if it is beneficial to replace one of the directions in the set by $\mathbf{v} = \mathbf{P}_0 - \mathbf{P}_N$ where $\mathbf{P}_0$ is the starting point of the iteration and $\mathbf{P}_N$ the new point after the N line minimizations. In the example of Figure 5, $\mathbf{v}_2$ replaces $[1, 0]^T$ in the second iteration. The algorithm correctly decides not to include new directions in all other iterations as this would actually slow down convergence.

Deciding whether or not to include the new direction **v** after each iteration and which direction in the set should be replaced is quite complicated. For details and a mathematical discussion of conjugate direction sets and the resulting property of quadratic convergence, see [2].

5. The QOOL algorithm

QOOL is an implementation of Powell's method in the context of multiple degree of freedom robot locomotion. The robot constantly evaluates its fitness as described in Chapter 2. If the QOOL online adaptation is active, the robot updates it with the current fitness as soon as the mechanical dynamics are stable. QOOL then returns a new set of parameters that should be tested by the robot.

In Chapter 5 of the report [1] I discuss two applications for online optimization of locomotion in the context of modular robotics: a) When self-assembling modules build random configurations, they could learn to locomote efficiently (optimization from scratch); b) The locomotion gait of a robot may be adapted online to the environment (adaptation of a specific gait). QOOL may be used for both scenarios. The only difference is, that the standard deviation for the Gaussian distribution in the bracketing procedure should be much smaller in the second scenario. This implicates that the robot continuously explores the parameter space in a *small* neighborhood. Thus, locomotion remains efficient by continuously adapting to changing environmental constraints. Contrary to optimization of mathematical functions, it is important for QOOL to re-evaluate the current optimum at each iteration because the environment and consequently also the fitness landscape are dynamic.

One can show that Powell's method finds the minimum of a function in the order of N^2 iterations, where N is the number of dimensions [2]. Each iteration consists of N line minimizations. A one-dimensional optimization usually involves about five to ten fitness evaluations. Thus, I estimated that QOOL online optimization from scratch would involve approximately $10*N^3$ fitness evaluations. To my surprise, the algorithm turned out to be much faster (see next chapter) because it converges to an efficient locomotion gait already in the very first iterations of Powell's method.

The key to fast online optimization is also a reduction of the parameter space. A powered module is controlled with three parameters as described in [2]: The amplitude, the phase and the initial angle. The coupling weights of the nonlinear oscillators are set using phase prediction. Corresponding modules in symmetric limbs share the same control parameters; only the very first modules of symmetric limbs have distinct phase differences.

6. Results

I tested QOOL with robots that have been evolved with the co-evolutionary algorithm [2]. For online optimization from scratch, I reset all control parameters to random values. Online adaptation is tested by simulating evolved robots and trying to further optimize locomotion.

The following figure shows QOOL online optimization from scratch (randomly initialized control parameters) of the salamander from Figure 1. It has six powered modules, but since they are all limb modules the locomotion controller has only seven parameters (three phases, three amplitudes plus the additional phase of the mirrored limb). Already the second line minimization, which optimizes the phase of the right limb (in red) leads to an efficient locomotion gait after exactly two minutes of simulated time (i.e. real time, *not* computation time of the computer). The first iteration of Powell's method is finished after 6.5 minutes and the new direction is tested (the point where all parameters change at the same time). However, the new direction is not included in the set. In contrast, the new direction of the second iteration (after 14.5 minutes) is included in the set. The subsequent line minimization along this direction gets very close to the optimum in a single leap.

After only 24 minutes of simulated time and three iterations of Powell's method, QOOL detects that the optimum has been reached. In this short time, a highly efficient walking gait of a robot with six degrees of freedom has been found. Notably, the gait is efficient already after the first five minutes of QOOL optimization! By turning on the graphics one can actu-

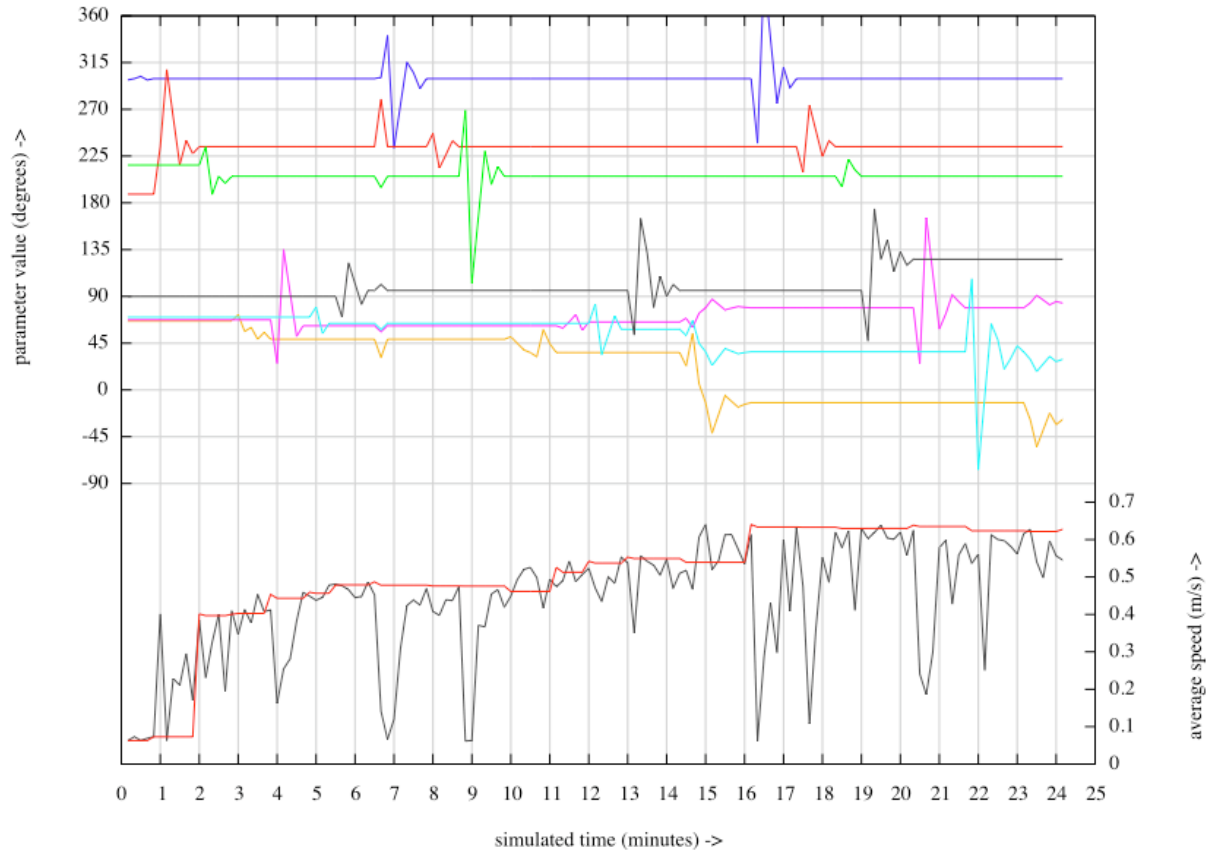

Figure 6: Online optimization of the salamander (see Figure 1) from scratch. The optimum is found after 24 simulated minutes, or 145 fitness evaluations (a fitness evaluation takes about 10 simulated seconds). The lines in the upper part of the figure correspond to the parameter values. The fitness of the optimal gait found so far (in red) and the gait that is currently being tested (in black) are plotted below. The maximum fitness decreases sometimes because it is being re-evaluated at each iteration.

ally watch how locomotion improves within few minutes. Note that in previous research, *offline* optimization of comparable gaits (for example with genetic algorithms) took hours of *computation* time, i.e. dozens of hours of simulated time. Movies are available at:

http://birg.epfl.ch/page32031.html

The results of the online optimization from scratch of the walker from Figure 1 are shown below. With eleven degrees of freedom this robot is quite complex. Nevertheless, a very efficient and elegant locomotion gait is found in less than 40 minutes of simulated time. The first iteration is completed after 12 minutes and the new direction is added to the set. The subsequent line optimization along this direction gets very close to the optimum. Note that the optimized gait is slightly fitter than the gait that had been evolved with the genetic algorithm in hours of computation time.

I have not yet tested the online adaptation of locomotion to changing environmental constraints but I expect it to perform well because it is actually the simpler case compared to optimization of a randomly initialized CPG (optimization from scratch). Evolved locomotion gaits could not be further improved by QOOL in a static environment because they are generally already highly optimal (see Figure 8).

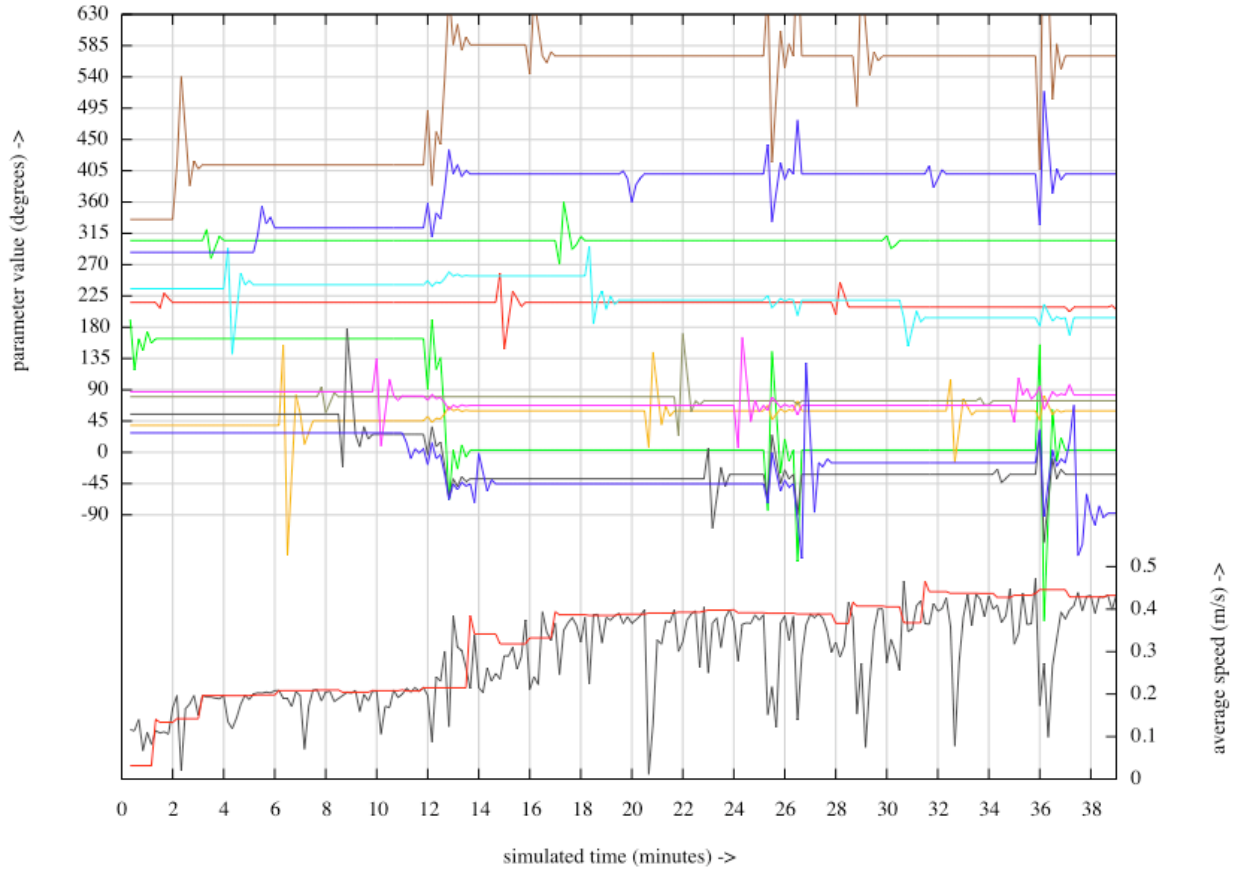

Figure 7: Online optimization of the walker (see Figure 1) from scratch. The lines in the upper part of the figure correspond to the values of the different parameters and the maximum and current fitness are plotted below.

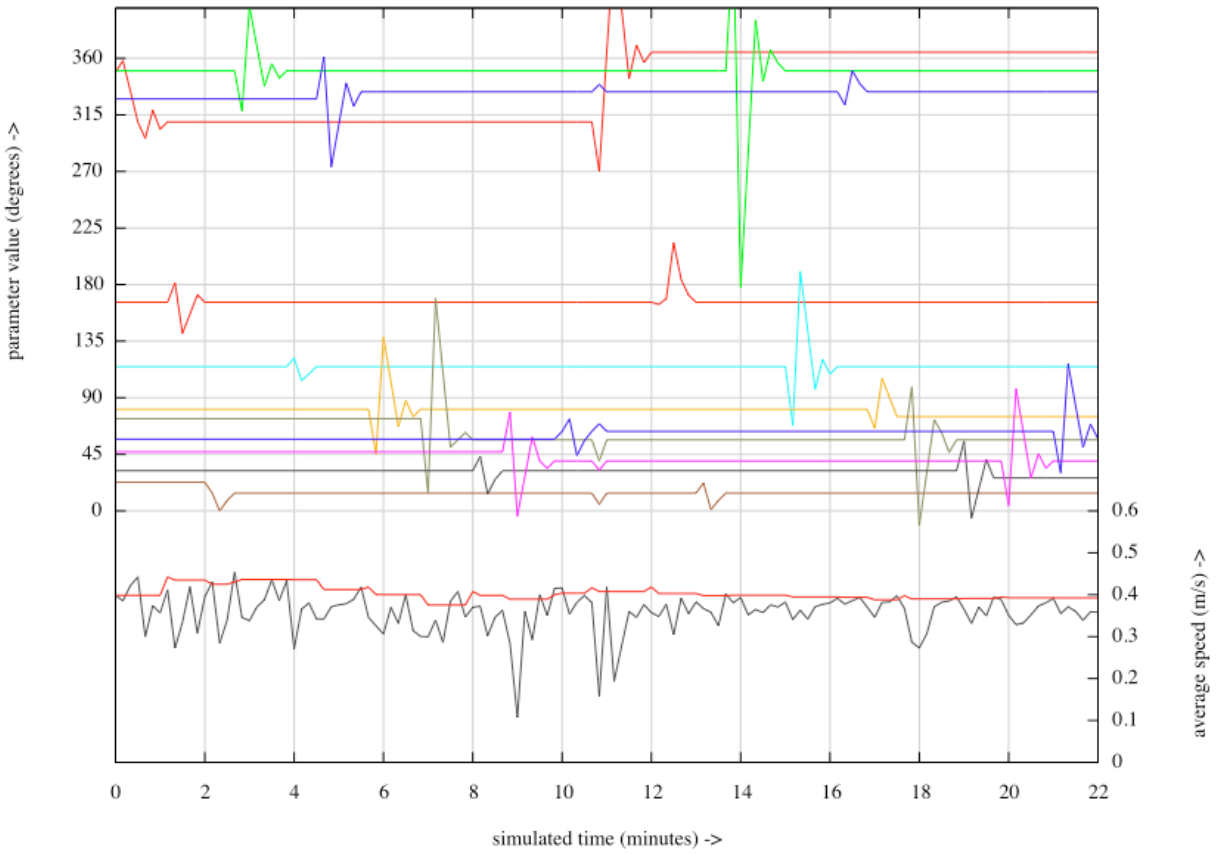

Figure 8: Adaptation of an evolved locomotion gait in a static environment (walker of Figure 1). Because the gait is already optimal for this environment, it can't be further improved. The maximum fitness fluctuates because it is constantly being re-evaluated. Note that most parameter values are left unchanged. The parameters that are modified by the optimization algorithm (e.g. the red line on the top) have little effect on the gait.

References

[1] D. Marbach. *Evolution and Online Optimization of Central Pattern Generators for Modular Robot Locomotion*. Unpublished Master Thesis, Swiss Federal Institute of Technology Lausanne, 2005. Available at: http://birg.epfl.ch/page32031.html

[2] W.H. Press, S. A. Teukolsky, W.T. Vetterling, B.P. Flannery. *Numerical Recepies in C: the art of scientific computing* (2nd ed.), Cambridge University Press, 1992. Available at: http://www.library.cornell.edu/nr/bookcpdf.html

[3] R.P. Brent. *Algorithms for Minimization without Derivatives.* Englewood Cliffs, NJ: Prentice-Hall, 1973.