We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

11

Learning to Walk with Model Assisted Evolution Strategies

Matthias Hebbel and Walter Nisticò
Technische Universität Dortmund
Germany

1. Introduction

Many algorithms in robotics contain parameterized models. The setting of the parameters in general has a strong impact on the quality of the model. Finding a parameter set which optimizes the quality of the model typically is a challenging task, especially if the structure of the problem is unknown and can not be specified mathematically, i.e. the only way to get the function value for a certain parameter set is to test them on the real problem. Controlling the robot's legs in order to walk is an example of such kind of *black box* approach. The walk is usually defined by three-dimensional trajectories for the movement of the legs and feet. The parameters (e.g. step height, step length, timings etc.) of these trajectories affect the stability and speed of the walking gait. Finding a parameter set which creates an optimal (in this case fast) walk can not be done by hand, since the direct effect of a parameter change is unclear. This chapter describes how Evolution Strategies can be used for the gait optimization of a four-legged robot whose walk is defined by 31 parameters and presents how Model Assisted Evolution Strategies lead to a faster convergence of the optimization of the problem.

At first an introduction to the standard Evolution Strategy with self-adaptation of the mutation strengths is given. The individuals of the Evolution Strategy here define a set of walk parameters. To evaluate the fitness of each individual, the robot has to walk for a certain time with this parameter setting and measure the resulting walk speed. This is not only time consuming but also causes a lot of wear out of the robot – especially parameters resulting in stumbling or tumbling can damage the robot. Consequently, the goal is to minimize the amount of real fitness evaluations on the robot. For that reason Model Assisted Evolution Strategies will be introduced. These strategies use the fitness measurements as obtained samples from the search space to predict the fitness of the individuals and – based on the model – sort out the least promising individuals, resulting in a faster convergence of the optimization process.

Since the quality of the model used for fitness prediction has a big effect on the convergence rate of the Model Assisted Evolution Strategy, mechanisms for controlling the impact of the model will be explained. The chapter closes with the results and comparisons of the walk learning process of the standard Evolution Strategy and its model assisted modifications.

Source: Frontiers in Evolutionary Robotics, Book edited by: Hitoshi Iba, ISBN 978-3-902613-19-6, pp. 596, April 2008, I-Tech Education and Publishing, Vienna, Austria

2. The Standard Evolution Strategy

The Evolution Strategy (ES) has been developed by Schwefel (Schwefel, 1975) and Rechenberg (Rechenberg, 1973) for the optimization of technical problems. It is inspired by the biological evolution and is working with a population of individuals. Each individual represents a point $\vec{x}$ in the real-valued search space; the point $\vec{x}$ is encoded in the object parameters $x_1,\ldots,x_n$ of the individual and represents its genes. In terms of Evolution Strategies each individual can be assigned a function value $f(\vec{x})$ (in this case scalar) representing the quality of the solution, the so called fitness which has to be optimized. The process of the evolution is shown in Fig. 1. An amount of μ individuals constitute the parent population and create a new offspring population of the next generation with λ individuals. An offspring individual is created by recombining the real-valued genes of ρ randomly selected parents followed by a random mutation of each object parameter such that the offsprings differ in their genes from their parents. The fitness of each individual in the offspring generation will be measured and the selection operation chooses the μ best individuals to establish the new parent generation.

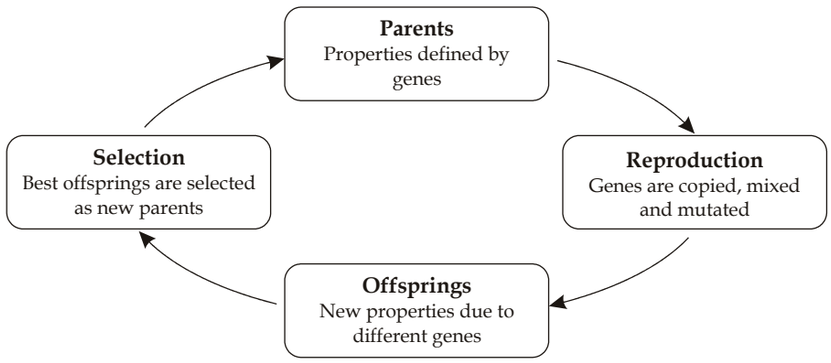


Figure 1. Optimization cycle of the standard Evolution Strategy

Offspring individuals are generated by means of recombination from the parent individuals. They can be recombined in a discrete or intermediate way. In case of the discrete recombination each object parameter x_i is selected randomly from the according object parameter p_i from one out of the ρ parents. The intermediate recombination on the other hand creates the components x_i out of the mean value of all p_i of the ρ parents. In both recombination variants the ρ parents are chosen randomly from the parent population, their fitness does not affect the probability to be chosen.

The leading part in the genetic modification of individuals is played by the mutation operator. The mutation operator is executed after the recombination. Each object parameter x_i is modified by adding a normally distributed random value $N\left(0,\sigma_i^2\right)$ with expected value 0 and standard deviation σ_i:

$$\tilde{x}_i = x_i + N\left(0,\sigma_i^2\right). \tag{1}$$

The standard deviation conforms to the mutation strength: the bigger the standard deviation, the higher is the probability of a greater mutation and vice versa. Each object parameter x_i has its own mutation strength σ_i. The values of the mutation strengths σ_i have a big effect on the progress speed of the evolution process. With a mutation strength chosen

too big, the chance to overshoot the optimum rises, while small mutation strengths increase the probability to get stuck in a local optimum and furthermore the speed to approach the optimum is unnecessary small. Hence it is very important to adapt the mutations strengths to the current situation. For that reason Rechenberg developed the $1/5$ rule for an Evolution Strategy with 1 parent and 1 offspring. He proved for two models that the mutation strength is optimal if 1 out of 5 generated offsprings is fitter than the previous parent (Rechenberg, 1973). In case of a bigger success rate the mutation strength is too small and has to be increased, otherwise it is too big and has to be decreased.

Schwefel improved the control of the mutation strengths and extended it to Evolution Strategies with more than 1 offspring (Schwefel, 1975). Each object parameter carries its own mutation strength (strategy parameter). These strategy parameters are also included in each individual and are selected and inherited together with the individual's assignments for the object parameters. Thus, they have a higher probability to survive when they encode object parameter variations that produce fitter individuals. This adaptation of the mutation strength is called **self-adaptation** (Beyer, 1996). The recombination of the strategy parameters can be discrete or intermediate. The mutation of the strategy parameters is executed before the mutation of the object parameters with

$$\tilde{\sigma}_i = \sigma_i \cdot \exp\left(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)\right). \tag{2}$$

The mutation consists of the random number $\tau' \cdot N(0,1)$ which is equal for all variances σ_i of the individual while the random number $\tau \cdot N_i(0,1)$ is varied for each σ_i. Schwefel recommends $\tau' \sim 1/\sqrt{2n}$ and $\tau' \sim 1/\sqrt{2\sqrt{n}}$ for an n-dimensional search space.

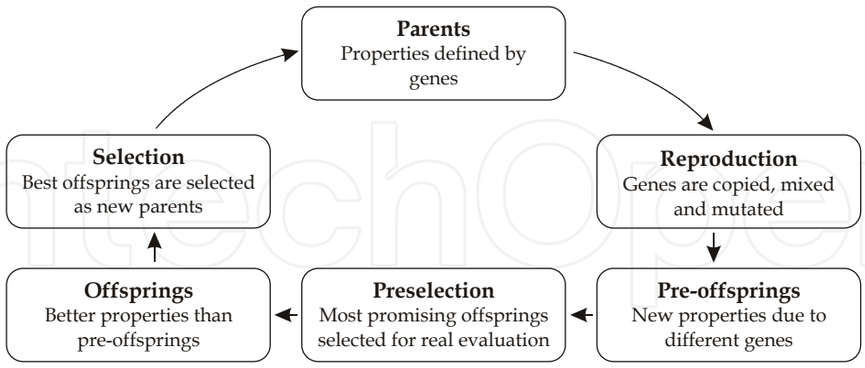
After the fitness values of the offspring individuals have been assigned, the selection operator selects the parents for the next generation. In case of the so called "+" strategy it chooses the μ best individuals out of the offspring **and** the parent population, while in case of the "," strategy only the best μ individuals out of the offspring population are selected to become the parents of the next generation. Additionally the lifetime of an individual can be limited to κ generations in case of the + selection, creating a mixture of the + and , selection.

The evolution loop can be stopped after a satisfying solution has been found, a number of generations have passed or the fitness does not improve over a number of generations, indicating a convergence.

3. Model Assisted Evolution Strategies

The basic idea of a Model Assisted Evolution Strategy (MAES) is to generate a higher amount of offsprings λ_p with $\lambda_p > \lambda$ and to preselect the most promising candidates. Out of the λ_p offsprings the λ best ones will be preselected based on their predicted fitness. The fitness is predicted from a model which takes the previously measured fitness values into account. The λ preselected individuals will afterwards be evaluated and the μ best chosen as parents for the next generation like in the standard Evolution Strategy, Fig. 2 visualizes this extension to the standard Evolution Strategy. Notice that the fitness of the offsprings which are available to become parents is always evaluated on the real problem and therefore the individuals are always assigned the true fitness. This way a convergence to a wrong optimum caused by wrong fitness predictions by the model is excluded.

If the model manages to sort out the worst candidates and to preselect the best ones, the convergence of the Evolution Strategy is accelerated, even though the number of real fitness



evaluations is the same as in the case of the standard Evolution Strategy. The following section will explain the three models used in this work for fitness approximation.

Figure 2. Evolution loop of a Model Assisted Evolution Strategy

3.1 Modelling the Search Space

During the evolution process the fitness measurements give an indication for the structure of the search space, they represent known scalar function values. Based on these known function values the model has to be able to make predictions of function values of unobserved points. Mathematically speaking the model must approximate the function $f(\vec{x}): \mathbb{R}^n \to \mathbb{R}$ based on d previously observed data points which are stored (with the according measured fitness values) in the data base D for any query point $\vec{q} \notin D$. The estimated function value will be called $\hat{f}(\vec{q})$ and should be close to the real (unknown) function value $f(\vec{q})$.

Principally the model can be global or local. Global models are valid for the whole search space and can be used for estimation of any query point $\vec{q}$. They have only to be calculated when new points are integrated in the underlying data base D, i.e. for Evolution Strategies after each generation. The biggest problem for global models is the typically small number of points in D relative to the dimension of the usually multimodal search space, leading to a poor general approximation. Local models on the other hand are only valid for a small region around a query point $\vec{q}$ and have to be recalculated for each approximation of $\hat{f}(\vec{q})$. Local models normally only take a subset of points in D closest to $\vec{q}$ into account since these points carry the most information about $\hat{f}(\vec{q})$. Notice that the data base D grows during the evolution process: each individual whose true fitness is evaluated is added with its measured fitness value.

The simplest model is the **Nearest Neighbour** approximation. It estimates the function value of a query point $\vec{q}$ with the function value of its nearest neighbour among the previously measured points in the data base D. The distance Δ between two points $\vec{x}$ and $\vec{q}$ can be the Euclidian distance

$$\Delta(\vec{x},\vec{q}) = \sqrt{\sum_{i=1}^{n}(x_i - q_i)^2} \tag{3}$$

which is appropriate if each dimension has the same scaling and a comparable gradient. If the dimensions have different scales, they can be standardized to have a standard deviation of 1 to make them comparable. For distance calculation each dimension i then has to be scaled by the reciprocal of the standard deviation $\sigma_i = \sqrt{\mathrm{var}\left(x_i^{(l)}\right)}, 1 \le l \le d$ with the weight $w_i = 1/\sigma_i$

$$\Delta(\vec{x},\vec{q}) = \sqrt{\sum_{i=1}^{n}\left(w_i\left(x_i - q_i\right)\right)^2}\ . \tag{4}$$

In case of this Nearest Neighbour model the approximated function value $\hat{f}(\vec{q})$ is given by

$$\hat{f}(\vec{q}) = f(\vec{x}_{\min}),\ \vec{x}_{\min} = \underset{l=1,\ldots,d}{\arg\min}\left\{\Delta\left(\vec{x}^{(l)},\vec{q}\right)\right\} \tag{5}$$

This model is computationally very efficient; however it has the disadvantage that it does not interpolate between measured points which could lead to the problem that several offsprings will be assigned the same fitness value and the preselection operation can not distinguish their quality.

The **polynomial model** uses linear regression to fit the measured points in D to a polynomial function to overcome this problem. At first the standard linear regression will be explained and then extended to the weighted linear regression.

An n-dimensional polynomial of second-order is specified by

$$\hat{f}(\vec{x}) = a_0 + \sum_{1\le i\le n} a_i x_i + \sum_{1\le i\le j\le n} a_{n-1+i+j} x_i x_j \tag{6}$$

with the coefficients a_i. The problem in regression is now to determine the coefficients a_i such that $\hat{f}(\vec{x})$ approximates $f(\vec{x})$ in all d points of the data set D as good as possible. In mathematical notation the squared error function

$$\varepsilon(\vec{a}) = \sum_{i=1}^{d}\left(f\left(\vec{x}^{(i)}\right) - \hat{f}\left(\vec{x}^{(i)}\right)\right)^2 \tag{7}$$

has to be minimized. Written as a matrix equation with the measured function values $y^{(1)},\ldots,y^{(d)}$ in D

$$\vec{y} = \left[\, y^{(1)},\ldots,y^{(d)}\,\right]^T \tag{8}$$

and the matrix X representing the second-order polynomial

$$\mathrm{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & \left(x_n^{(1)}\right)^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(d)} & x_2^{(d)} & \cdots & \left(x_n^{(d)}\right)^2 \end{bmatrix} \tag{9}$$

the equation

$$\vec{y} = \mathrm{X}\cdot\vec{a} \tag{10}$$

holds for an exact approximation.

If the polynomial can not approximate the function values exactly, an error term $\vec{v}$ has to be added:

$$\vec{y} = \mathrm{X} \cdot \vec{a} + \vec{v} \,. \tag{11}$$

As mentioned before, the error term $\varepsilon(\vec{a})$ has to be minimized and equation (7) can be written in matrix notation as

$$\varepsilon(\vec{a}) = \sum_{i=1}^{d} \left(f\left(\vec{x}^{(i)}\right) - \hat{f}\left(\vec{x}^{(i)}\right) \right)^2 = \vec{v}^T \cdot \vec{v} = \left(\vec{y} - \mathrm{X} \cdot \vec{a}\right)^T \left(\vec{y} - \mathrm{X} \cdot \vec{a}\right). \tag{12}$$

For the minimization of $\varepsilon(\vec{a})$ it is sufficient to postulate that $\nabla\left(\varepsilon(\vec{a})\right) = 0$, resulting in the solution for the coefficients $\vec{a}$

$$\vec{a} = \left(\mathrm{X}^T \mathrm{X}\right)^{-1} \mathrm{X}^T \vec{y} \,. \tag{13}$$

This regression method can be used to build a global model if all points in *D* are used to estimate the coefficients $\vec{a}$. The disadvantage is that an underlying multimodal search space requires a polynomial of high order to be approximated closely. Unfortunately the number of coefficients rises disproportionately with the order of the polynomial; a polynomial of second order already has $(n+1)(n+2)/2$ coefficients (Jin, 2003) and still can only approximate the function with a unimodal polynomial. This least square method requires at least as many samples in the data base *D* as coefficients exist.

This problem can be overcome with a local model. The model becomes local if the distances to the query point are weighted and represent the fact that the close points carry more information about the query point $\vec{q}$ than points which are further away. All points in *D* can be taken or, since far points would get a weight close to 0, only a subset of the *k* closest points. The weight w_l is a function of the distance between the query point $\vec{q}$ and the *l*-th measured point $\vec{x}^{(l)}$ in the data base. As suggested in (Hoffmann & Hölemann, 2006) the weight is determined by a Gaussian kernel function

$$w_l\left(\Delta\left(\vec{x}^{(l)}, \vec{q}\right)\right) = \exp\left(-\Delta\left(\vec{x}^{(l)}, \vec{q}\right)^2\right). \tag{14}$$

The distance is measured as suggested in equation (4).

For the local model a polynomial of first order will be used which is given for an *n*-dimensional space by

$$\hat{f}\left(\vec{x}\right) = a_0 + \sum_{l=1}^{n} a_l x_l \tag{15}$$

with the coefficients $a_0, \ldots, a_{n+1}$. The weights for the *k* points are stored in the diagonal matrix

$$\mathrm{W} = \begin{pmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & w_k \end{pmatrix} \tag{16}$$

and the error function (12) results in the weighted case to

$$\varepsilon(\vec{a}) = (\vec{y} - \mathrm{X} \cdot \vec{a})^T \, \mathrm{W} (\vec{y} - \mathrm{X} \cdot \vec{a}) \,. \tag{17}$$

The optimal coefficients then can be calculated with

$$\vec{a} = \left(\mathrm{X}^T \mathrm{W} \mathrm{X}\right)^{-1} \mathrm{X}^T \mathrm{W} \vec{y} \,. \tag{18}$$

Another used model is the approximation with a **Gaussian Process**. The Gaussian Process is a probabilistic model which can calculate besides the predicted value also a confidence value representing the certainty of the prediction. In this work the implementation of Rasmussen with automatic relevance determination has been used. Explaining prediction with a Gaussian Process in detail would go beyond the scope of this chapter, very good detailed descriptions of Gaussian Processes are given in (Gibbs & MacKay, 1997; Rasmussen & Williams, 2006).

3.2 Controlling the Model Impact

Theoretically the Model Assisted Evolution Strategy already converges faster than the standard Evolution Strategy when the model performs better than a random preselection of the individuals would perform (Ulmer et al., 2004). However for a model which performs worse than a random selection, the progress speed can be reduced or optima might even not be reached because the individuals leading towards the optimum always get sorted out by the preselection. For that reason it makes sense to control the impact of the model based on the capability of the model to preselect the individuals.

For the preselection process the capability of the model to select the most promising individuals is of importance, while the ability to estimate the exact fitness value is of little interest. Thus the quality of the model is defined according to (Ulmer et al., 2004; Jin et al., 2003), as follows: Since only the real fitness values of the λ offsprings are known, the quality will be evaluated based on the comparison of the selected μ individuals based on the real fitness measurement and the best μ individuals that would have been selected based on the predicted fitness. The λ offsprings are sorted according to their true fitness and a weight $(\lambda - i)$ with i representing their position is assigned to each individual. Thus the individual with the best fitness would get a weight $(\lambda - 1)$, the second best $(\lambda - 2)$ etc. Now the weights for the individuals that would have also been selected by the fitness estimated by the model are summed up:

$$Q = \sum_{i=1}^{\mu} g_i (\lambda - i) \tag{19}$$

with g_i =1 if the individual would have been selected and g_i =0 otherwise. This quality measure takes the true rank of the individuals into account and reflects the fact that a wrongly classified best individual is punished more than for example a missed second best. The maximum quality as a result is given by

$$Q^{max} = \sum_{i=1}^{\mu} (\lambda - i) = \mu\lambda - \frac{\mu(\mu+1)}{2} \tag{20}$$

and the quality of a random preselection is defined by

$$Q^{rand} = \sum_{i=1}^{\mu} i \frac{\binom{\mu}{i}\binom{\lambda-\mu}{\mu-i}}{\binom{\lambda}{\mu}} \cdot \frac{1}{\mu} \sum_{i=1}^{\mu} (\lambda - i) = \frac{\mu^2}{\lambda} \frac{2\lambda - \mu - 1}{2}. \tag{21}$$

Two approaches based on the current quality of the preselection to influence the impact of the model will be used in this work.

The **Controlled Model Assisted Evolution Strategy** (C-MAES) adapts the influence of the model by varying the amount of offsprings λ_p which will be evaluated by the model. A good quality of the model results in a bigger trust in the model and λ_p can be increased, while for a poor model quality λ_p has to be decreased. After each generation t the selection quality $Q^{(t)}$ gets re-evaluated and $\lambda_p^{(t+1)}$ adapted according to the control law suggested by (Ulmer et al., 2004). If the quality is better than the random quality, the amount of offsprings is increased by

$$\lambda_p^{(t+1)} = \lambda_p^{(t)} + \frac{Q^{max} - Q^{(t)}}{Q^{max} - Q^{rand}} \cdot \delta_{\lambda_p}, \tag{22}$$

if the quality on the other hand is worse than the random quality, the model impact is decreased by

$$\lambda_p^{(t+1)} = \lambda_p^{(t)} - \frac{Q^{max} - Q^{(t)}}{Q^{rand}} \cdot \delta_{\lambda_p}. \tag{23}$$

The factor δ_{λ_p} denotes the adaptation rate.

Hoffmann and Hölemann suggest the adaptation of the size of the offspring generation λ in order to have a better control over the model impact (Hoffmann & Hölemann, 2006). This variant is called **λ-Controlled Model Assisted Evolution Strategy** (λ-CMAES). Here the amount of offsprings λ_p that is undergoing the preselection is kept constant. The idea is that in case of a good quality of the preselection only a smaller amount of offsprings λ has to be evaluated on the real problem, since in case of a good model already the best individuals from the λ_p have been preselected. The amount of offsprings $\lambda^{(t+1)}$ for the next generation $t+1$ is updated according to the following control laws: if the current model quality is better than a random model, i.e. $Q^{(t)} > Q^{rand}$, the $\lambda^{(t+1)}$ is decreased, giving more trust in the model by

$$\lambda^{(t+1)} = \max\left(\lambda^{(t)} - \frac{Q^{rand} - Q^{(t)}}{Q^{rand}} \cdot \delta_{\lambda}, \mu \right). \tag{24}$$

If the model actual quality is worse than the quality of the random model, the trust in the model is decreased by increasing $\lambda^{(t+1)}$ according to

$$\lambda^{(t+1)} = \min\left(\lambda^{(t)} + \frac{Q^{max} - Q^{(t)}}{Q^{max} - Q^{rand}} \cdot \delta_{\lambda}, \lambda_p \right). \tag{25}$$

A particularity to be mentioned is that the previously explained quality calculation becomes unstable for $\lambda = \mu$ since all individuals would always be selected correctly. To ensure a sufficient statistical basis, μ used in the quality calculation is replaced by $\tilde{\mu} = \lambda/2$ if λ drops below 2μ (Hoffmann & Hölemann, 2006).

4. Learning to Walk with Evolution Strategies

The performance of the different Evolution Strategies will be evaluated for a robot learning to walk. The problem of learning to walk is reduced to optimizing parameters for a given parameterized model which generates walk movements for a robot.
The used robot platform for the presented research is a simulated model of the formerly commercially available Sony Aibo ERS-7 shown in Fig. 3. The Aibo is a quadruped robot with three degrees of freedom in each leg, i.e. a hip abduction, a hip flexion and a knee flexion joint. SimRobot (Laue et al., 2006) has been used as the simulator. It is a general robot simulator using the OpenDynamicsEngine for the physical simulation. The physical properties of the simulated robot like maximum force and maximum speed of the motors in the joints have been specially trained to behave like the joints of the real robot.

Figure 3. Sony Aibo robot ERS-7

The most significant achievement concerning the walk on this robotic platform has been presented by Hengst et al. with the so-called *PWalk* (Hengst et al., 2002) which achieves a very good stability using a particular posture of the front legs, where the contact point with the ground is located on the forearm instead of the paw. As shown in Fig. 4., this approach moves the robot's legs on predefined loci. The required joint angles for the movement of the legs are then calculated using inverse kinematics. Following a trot gait, the two diagonal opposite legs are always moved at the same time, i.e. two legs are in the air while the other two legs remain in contact with the ground. The loci defining the foot movement are represented in a parameterized form (Hebbel et al., 2006). Each locus is defined by four 3-dimensional points $P_1,\ldots,P_4$ both for the front and hind legs (see Fig. 4.). The time parameter t_{step} defines the total time for a complete step. Additionally 3 timing parameters t_1,t_2,t_3 are used for each locus to specify the relative amount of time needed for the foot to travel from vertex P_i to P_{i+1} for $1 \le i < 4$. These relative times sum up to 1, consequently the time t_4 to travel from P_4 to P_0 is defined by $t_4 = 1 - t_1 - t_2 - t_3$. Thus, each locus is well-defined by the $3 \cdot 4$ parameters $P_1,\ldots,P_4$, the 3 timing parameters t_1,t_2,t_3 and one global timing parameter t_{step}, summing up to 16 parameters for one locus. The loci of the front and hind legs are specified separately, but the global time parameter t_{step} is shared, hence the total amount of parameters is 31.
Optimizing these 31 parameters by hand is not feasible and it has become state-of-the-art to use machine learning algorithms to optimize the walk parameters (Kohl & Stone, 2004; Röfer 2004; Hebbel et al., 2006). The criterion to be optimized is the walk speed of the robot. In this case only the straight forward walk is optimized. An individual defines with its 31 object parameters the walk movement and its fitness is evaluated by letting the (simulated)

robot walk for an amount of time and measure its speed. Individuals which result in a walk which is either not straight or lets the robot fall on its side are assigned the worst fitness value, zero. The offsprings have been evaluated in parallel on a compute cluster with 60 nodes, allowing an acceptable simulation time.

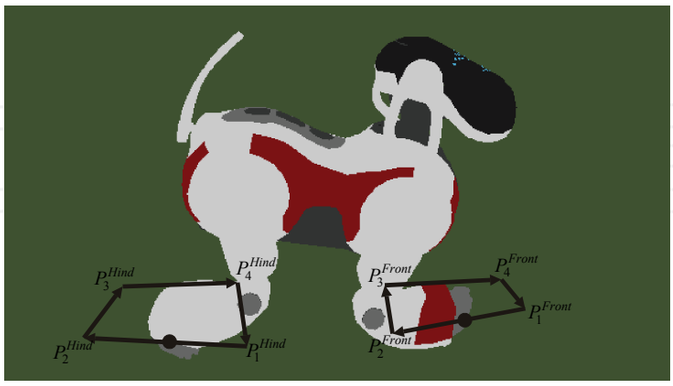


Figure 4. Schematic representation of the leg movement

5. Results

For all experiments the basic settings of the $(5,30)$ Evolution Strategy have been used: the population size of the parents was 5, the size of the offspring generation 30. The selection was a comma selection which ensures a good performance of the self-adaptation of the mutation strengths. The recombination was intermediate and the first parent generation was randomly initialised. Each strategy has been run until 7000 fitness evaluations have been exceeded and repeated with exactly the same settings at least 15 times. The resulting fitness curves of the runs have been averaged to suppress random fluctuations.

At first the performance of the not controlled Model Assisted Evolution Strategies has been compared to the standard Evolution Strategy. The size of λ_p has in this experiment been constantly set to $\lambda_p = 2\lambda = 60$. In case of the polynomial model the 150 closest neighbours to the query point $\vec{q}$ have been used to build the local polynomial model. The Gaussian Process was trained with the 2λ most recently measured fitness evaluations (Ulmer et al., 2003).

Fig. 5 shows the results: all model assisted strategies converge much faster towards an optimum fitness of approximately 66 than the standard Evolution Strategy. All model assisted strategies have found the optimum at the latest after 5000 fitness evaluations. The standard Evolution Strategy however has not reached this optimum after 7000 fitness evaluations. Apparently the capability to model the search space is reflected in the convergence speed. The Gaussian Process model converges fastest closely followed by the polynomial model and the nearest neighbour model.

In the next experiment the effect of the Controlled Model Assisted Evolution Strategy and the λ-Controlled Model Assisted Evolution Strategy was analyzed. For this experiment the polynomial model has been used even though it is slightly inferior to the Gaussian Process model but is a lot faster to be calculated. The adaptation parameters δ_{λ_p} in equation (22) and (23) is set to $\delta_{\lambda_p} = 10$ and δ_λ from equation (24) and (25) is set to $\delta_\lambda = \mu/2 = 2.5$. The general strategy parameters are set as before. Fig. 6 shows the results of the controlled MAES compared to the not controlled MAES, notice that the ordinate in the graph has a different range than in Fig. 5. Both the C-MAES and the λ-CMAES show a slightly faster convergence

than the uncontrolled MAES. This happens because already after 500 fitness evaluations the model in both cases is good enough to preselect the most promising individuals and thus the impact of the model is increased. A meaningful difference between the C-MAES and the λ-CMAES is not observable for this problem.

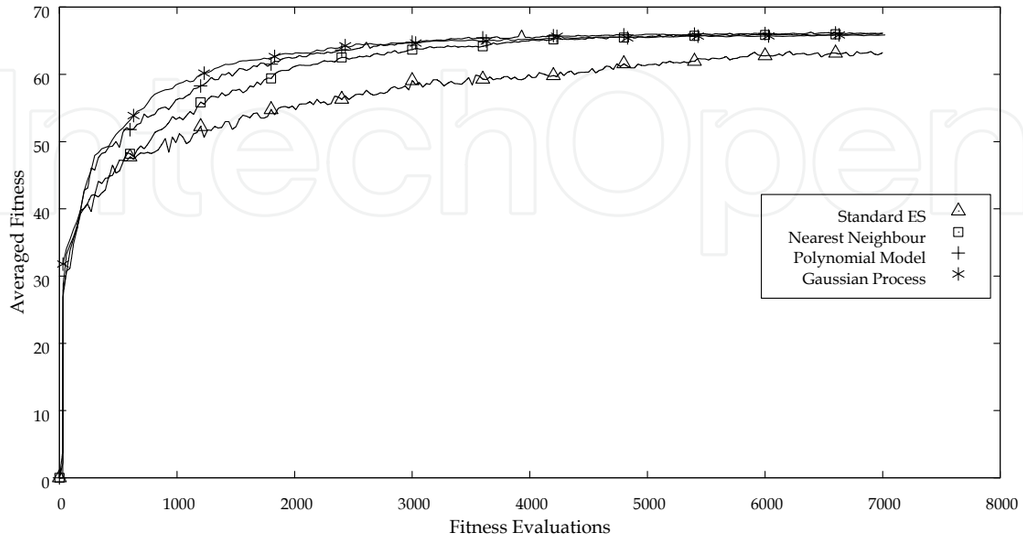


Figure 5. Comparison of the fitness curves of the Standard Evolution Strategy and the Model Assisted Evolution Strategies with different models

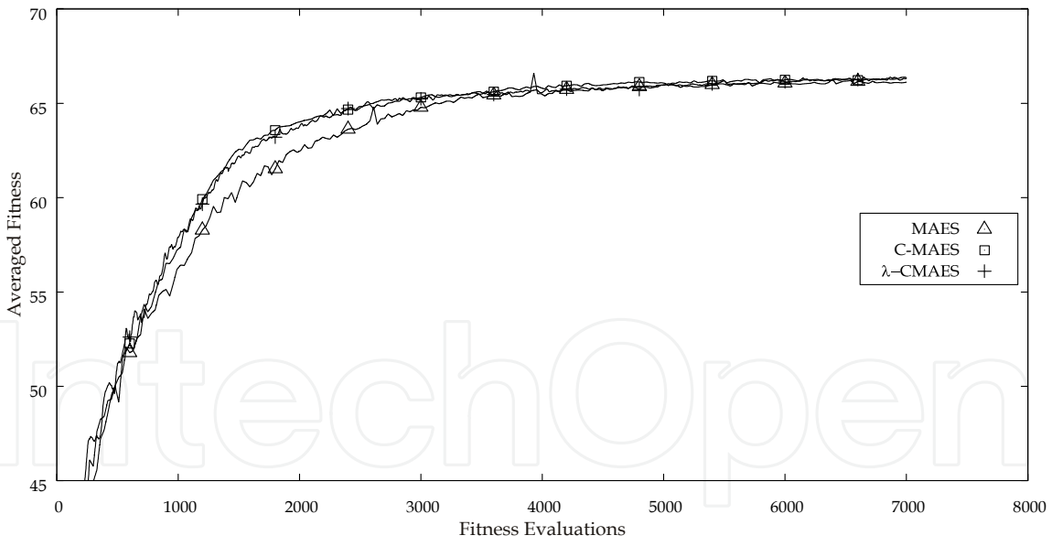


Figure 6. Comparison of the fitness curves of the uncontrolled Model Assisted Evolution Strategy and the controlled Model Assisted Evolution Strategies with different control mechanisms

6. Conclusion

Evolution Strategies are well suited to optimize the parameters of this practical problem. However the MAES outperform the standard ES by far and lead to a much faster convergence. The controlled MAES variants in this case can improve the convergence speed

compared to the uncontrolled strategies a bit more. But since they do not perform worse they should by preferred over the uncontrolled strategies making sure that in case of a bad model quality the model impact is reduced and the Evolution Strategy does not perform worse than a strategy without model assistance.

7. References

Beyer, H.-G. (1996). Toward a theory of evolution strategies: Self-adaptation, *Evolutionary Computation*, Vol. 3, No. 3, pp. 311-347

Gibbs, M. & MacKay, D. (1997). *Efficient implementation of gaussian processes*, Technical Report, Cavendish Laboratory, Cambridge, UK

Hebbel, M.; Kosse, R. & Nisticò, W. (2006). Modeling and learning walking gaits of biped robots, *Proceedings of the Workshop on Humanoid Soccer Robots of the IEEE-RAS International Conference on Humanoid Robots,* pp. 40-48, Genoa, December 2006

Hebbel, M., Nisticò, W. & Fisseler, D. (2006). Learning in a high dimensional space: fast omnidirectional quadrupedal locomotion. *RoboCup 2006: Robot Soccer World Cup X*, Lecture Notes in Artificial Intelligence, Springer

Hengst, B.; Ibbotson, D.; Pham, S.B. & Sammut, C. (2002). Omnidirectional locomotion for quadruped robots. *RoboCup 2001: Robot Soccer World Cup V*, pp. 368-373, Lecture Notes in Computer Science, Springer

Hoffmann, F & Hölemann, S. (2006). Controlled model assisted evolution strategy with adaptive preselection, *Proceedings of 2nd International Symposium on Evolving Fuzzy Systems*, pp. 182-187, Ambleside, September 2006

Jin, Y.; Hüsken, M. & Sendhoff, B. (2003). Quality measures for approximate models in evolutionary computation, *Proceedings of GECCO Workshops: Workshop on Adaptation, Learning and Approximation in Evolutionary Computation*, pp. 170-174

Jin, Y. (2003). A comprehensive survey of fitness approximation in evolutionary computation, *Soft Computing*, Vol. 9, No. 1, 2005, pp. 3-12

Kohl, N. & Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*

Laue, T.; Spiess, K. & Röfer, T. (2006). SimRobot - A general physical robot simulator and its application in RoboCup. *RoboCup 2005: Robot Soccer World Cup IX*, pp. 173–183, Lecture Notes in Artificial Intelligence, Springer

Rasmussen, C. E. & Williams, C. K. I. (2006). *Gaussian processes for machine learning*, MIT Press, ISBN 0-262-18253-X

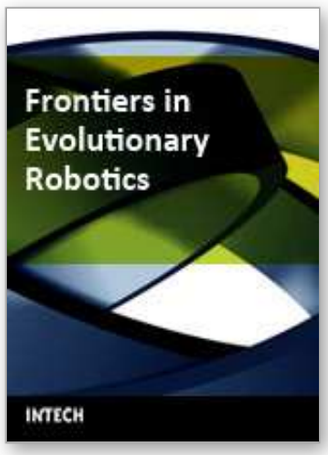
Rechenberg, I. (1973). *Evolutionsstrategie*. Friedrich Fromm Verlag, Stuttgart

Röfer, T. (2004). Evolutionary gait-optimization using a fitness function based on proprioception. *RoboCup 2004: Robot Soccer World Cup VIII, LNAI*, Springer

Schwefel, H.-P. (1975). *Evolutionsstrategie und numerische Optimierung*. Dr.-Ing. Dissertation, Technische Universität Berlin, Fachbereich Verfahrenstechnik

Ulmer, H.; Streichert F. & Zell, A. (2003). Evolution strategies assisted by gaussian processes with improved pre-selection criterion, *Proceedings of IEEE Congress on Evolutionary Computation (CEC'03)*, pp. 692-699, Canberra, Australia, December 2003

Ulmer, H.; Streichert, F. & Zell, A. (2004). Evolution Strategies with Controlled Model Assistance, *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pp. 1569-1576, Portland, Oregon, June 2004



Frontiers in Evolutionary Robotics

Edited by Hitoshi Iba

ISBN 978-3-902613-19-6

Hard cover, 596 pages

Publisher I-Tech Education and Publishing

Published online 01, April, 2008

Published in print edition April, 2008

This book presented techniques and experimental results which have been pursued for the purpose of evolutionary robotics. Evolutionary robotics is a new method for the automatic creation of autonomous robots. When executing tasks by autonomous robots, we can make the robot learn what to do so as to complete the task from interactions with its environment, but not manually pre-program for all situations. Many researchers have been studying the techniques for evolutionary robotics by using Evolutionary Computation (EC), such as Genetic Algorithms (GA) or Genetic Programming (GP). Their goal is to clarify the applicability of the evolutionary approach to the real-robot learning, especially, in view of the adaptive robot behavior as well as the robustness to noisy and dynamic environments. For this purpose, authors in this book explain a variety of real robots in different fields. For instance, in a multi-robot system, several robots simultaneously work to achieve a common goal via interaction; their behaviors can only emerge as a result of evolution and interaction. How to learn such behaviors is a central issue of Distributed Artificial Intelligence (DAI), which has recently attracted much attention. This book addresses the issue in the context of a multi-robot system, in which multiple robots are evolved using EC to solve a cooperative task. Since directly using EC to generate a program of complex behaviors is often very difficult, a number of extensions to basic EC are proposed in this book so as to solve these control problems of the robot.

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen

IntechOpen