Automatic Generation of Control Programs for Walking Robots Using Genetic Programming

Jens Busch, Jens Ziegler, Christian Aue, Andree Ross, Daniel Sawitzki, and Wolfgang Banzhaf

University of Dortmund,
Department of Computer Science, Chair of Systems Analysis (LS XI)
D-44221 Dortmund, Germany
{banzhaf, busch, ziegler, sigel}@ls11.cs.uni-dortmund.de
http://ls11-www.cs.uni-dortmund.de/~sigel

Abstract. We present the system *SIGEL* that combines the simulation and visualization of robots with a Genetic Programming system for the automated evolution of walking. It is designed to automatically generate control programs for arbitrary robots without depending on detailed analytical information of the robots' kinematic structure. Different fitness functions as well as a variety of parameters allow the easy and interactive configuration and adaptation of the evolution process and the simulations.

1 Introduction

Autonomous mobile robots are becoming more and more important, because they are expected to solve tasks that humans are not able to cope with or that humans ought not to cope with [3,4]. This requires extensively autonomous robots, because with growing complexity of the problems it will be no longer possible for the programmer to take all eventualities into account from the outset. A special form of mobile robots are walking robots. This term includes all robots that locomote without wheels, caterpillars or similar devices on firm ground. The evolution of robot control programs has been the topic of recent publications. A general introduction into the concept of Genetic Programming can be found in [2,10]. Several applications of Genetic Programming (or, more generally, Evolutionary Algorithms) to the task of controlling autonomous robots are given in, e.g., [8,13]. The evolution of crawling or walking robots can be found e.g. in [11, 15,6]. For biological inspiration, gait patterns of stick insects have been analyzed to gain more detailed information on natural gait coordination algorithms [5], that in turn has influences on the design of robust and fast walking gait patterns. A good overview over the evolution of neural network controllers can be found in [12].

To evolve gait patterns or walking agents, we use simulated robots. Simulating walking robots allows more flexible architectures and rapid prototyping, which is, compared to experiments with real hardware, less expensive. Additionally, simulation is fast and does not strain the hardware. The evolution of movements of virtual agents in a simulated environment has been successfully demonstrated [14,9,16].

The approach presented here does not imply that the evolved controllers depend on specific information on the robot morphology such as certain lengths or distances. On the contrary, it was one of the main goals of this work to make the evolution of robot controllers as independent as possible from morphology specific information. So morphology-related information, although available, will not be used for evaluating individuals.

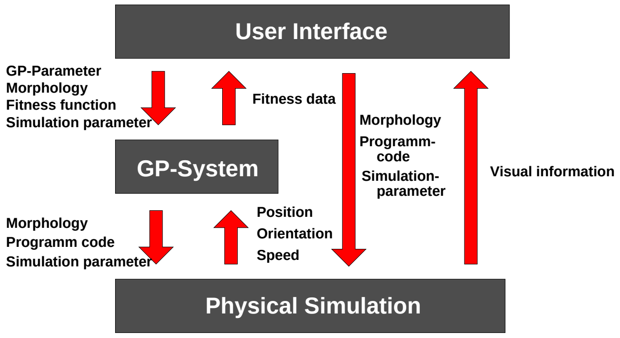

Fig. 1. Schematic view of the main components of *SIGEL*. Explanations see text.

The system *SIGEL* [1] is composed of three main components: The Genetic Programming module (described in Section 2), the physical robot simulation (Sect. 3) and the user interface (see Fig.1). The latter consists of the graphical interface as well as the underlying mechanisms to manage the experiments. An experiment mainly consists of GP and simulation parameters and the robot morphology. The following two sections are devoted to the description of the tested robot models and the results of the experiments.

2 The Genetic Programming System

The objective of a GP run is to evolve a robot control program that enables a simulated robot to walk. Our system uses linear genomes as a representation. Thus each program of a GP population is a sequence of robot instructions. The following instructions are element of the function set: ADD, SUB, MUL, DIV and MOD for arithmetic operations, COPY and LOAD for register manipulation, CMP, JMP, DELAY and NOP for execution control, and the SENSE and MOVE command as instructions that are directly connected to the robot. Each instruction is chosen with a predefined probability and needs one or two registers or constants as parameters. The program length is not fixed but a minimum and maximum length can be defined. By default, programs are initialized randomly, but it is possible to import manually defined code or programs of a previous experiment as well. A user interface provides easy access to the population for import and export of GP individuals and parameter adjustment. Other important GP parameters like population size, number of generations and variation probabilities may vary considerably between experiments and are explained in detail in Sect. 5.

Genetic Operators

Crossover, mutation and reproduction are applied with probabilities p_c,p_m and p_r, respectively. The crossover operator uses one-point crossover and thus swaps linear genome sequences between the mating partners. The crossover point is selected randomly. The genome of the offspring is cut or expanded to fulfill length restrictions. The mutation operator randomly changes different parts of the genome. (i) It deletes an arbitrary instruction. (ii) It inserts a new randomly initialized instruction at an arbitrary position. (iii) It replaces an arbitrary instruction by a new randomly initialized instruction. (iv) It modifies an arbitrary instruction. Either the function or its operand(s) are altered. The reproduction operator simply copies an individual. The genetic code will not be changed.

Selection and Parallelized Fitness Computation

The evaluation of control programs for walking robots in a physical simulation is computationally expensive. To save execution time of an evolutionary cycle we implemented a parallelized fitness evaluation. A number of tournaments T is scheduled and topologically sorted with respect to a partial order $\prec$ which is defined as follows:

$$T_x \prec T_y :\Leftrightarrow (\exists \text{ individual } i : (i \text{ is participant of } T_x \text{ and of } T_y \text{ and } x < y))$$

This method ensures a reproducible series of tournaments so that the experiment is independent of system parameters such as network traffic or workstation load. The resulting minimal set of tournaments is distributed over a heterogenous network of workstations using PVM.

3 Dynamic Simulation of Robots

In our system we simulate robots in a three-dimensional physical environment by using the DynaMechs software package (see [7]). This C++ programming library takes a model description of the robot and simulates its dynamics. The robot is given by a set of rigid bodies called links and their kinematic structure. The links are described by their physical attributes like inertia properties, mass, center of mass and their geometry in a polygonal representation. The robot's kinematic structure is defined by connecting links with joints. Additional parameters are global gravity and maximum and minimum joint forces and angles. Further forces result from the collision of the robot links with the floor. Collision between robot links is not taken into account in the actual implementation. To circumvent the intersection of links the appropriate allowed joint angles must be restricted manually. The implemented fitness function evaluates a robot and its program by measuring the distance it has moved during the simulation. The value is given in $[\frac{m}{s}]$. Additionally, it examines if the distance between the robot torso and the ground is larger than 0.5 length units. If this condition cannot be kept, so that the robot seems to break down, the robot program will receive a penalty value of zero. We supposed this function to be advantageous for a development of a *nice* movement, so we called this function *nice walking* fitness function. In the following experiments, the movement of the torso link of the robot is measured from the starting point to the point it has reached when the simulation stops. The faster and more linear an individual moves, the better is its fitness. The simulation is divided into discrete simulation steps translating the simulated system from time t to time $t + h$ where h is the granularity of the simulation. A small granularity leads to a more accurate simulation process. In each step the control program may effect the simulation in two aspects: (i) the program may apply forces at prismatic joints or torques at rotational joints with a move command. (ii) The program may read the actual angle of joints and store the values in the registers of the virtual machine with a sense command. These values can be used for future computations of forces and torques. Each simulation step consists of two substeps. First, the interpreter executes the next n commands of the control program with n being the maximum number of commands that can be executed in time h. Second, the system dynamics is calculated to update the simulation parameters consisting of the bodies' positions, orientations, velocities and accelerations. Given the set of forces affecting the system the DynaMechs simulation library calculates the motions (accelerations) of all links. Once the accelerations are known, numerical integration is used to determine the positions, orientations and velocities of the links at time $t + h$.

4 The Robots

We tested the system on several different robot architectures including simple "hoppers" consisting of only two links and one rotational joint (Fig. 2 a). This simple architecture was used for general experiments to gain experience with the dynamic simulation. We used more complex architectures in our experiments:

The caterpillar robot. This caterpillar robot consists of four links and three rotational joints (Fig. 2 b). Its head is shaped like a cube, in difference to the tail, which ends in a half cylinder. The head and the tail are connected to the inner links with horizontal rotational joints. The two identical inner links are connected together with a vertical rotational joint.

The two-legged robot. This robot has two symmetrical legs hinged to the torso link (Fig. 2 c). Each leg consists of three links and three rotational joints.

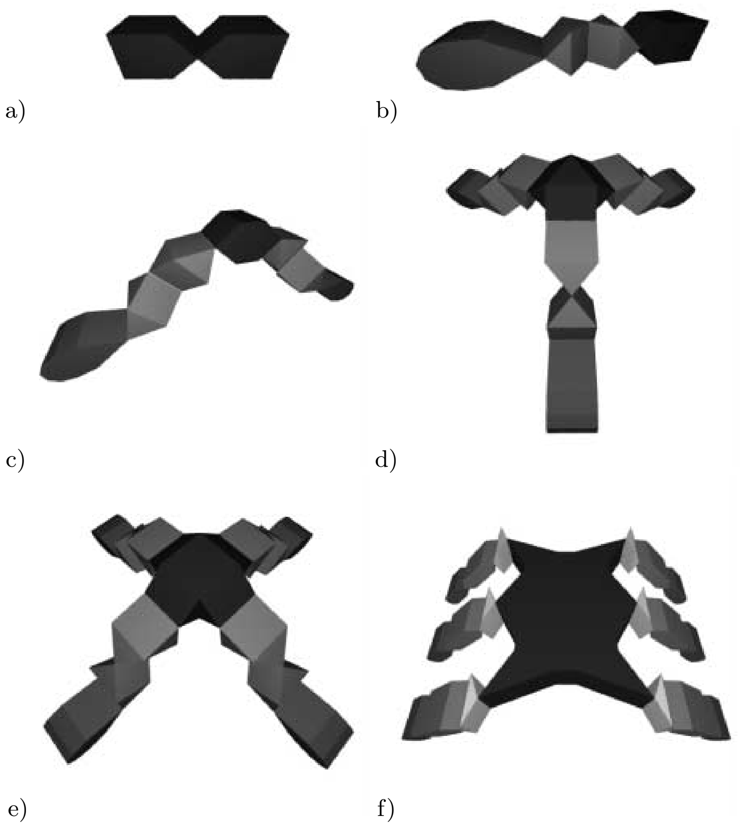

Fig. 2. The robots. From top left to bottom right: The hopper, the caterpillar, the two-legged and three-legged robot, the robot with four legs and the six-legged robot.

The three-legged robot. This robot has three symmetrical legs hinged around the torso link with a 120 degree angle between them (Fig. 2 d). Each leg consists of three links and three rotational joints. The legs are the same as the legs of the two-legged robot.

The four-legged robot. This robot has four symmetrical legs hinged around the torso link with a 90 degree angle between them (Fig. 2 e). This is the only difference between the architectures of the four-legged and the three-legged robot.

The six-legged robot. This robot consists of nineteen links and eighteen rotational joints arranged in a symmetrical architecture (Fig. 2 f). Its torso is carried by six identical legs, three on the left and three on the right side. Each of the six legs consists of three links. The shoulder-joint of each leg allows horizontal movement, the other two joints, for knee and ankle, have vertical axes of rotation.

5 Experiments and Results

The aim of our experiments was the development of control programs which let the robots walk straight-line and fast. Additionally, we wanted the robots to walk with the torso lifted to a specified level. In a series of experiments the effects of changing GP parameter settings have been investigated, whereas a second series analyzed the effects of changing kinematic structures. Each experiment includes several evolutions, which are not interdependent (with the exception of experiment 5, which takes some individuals of experiment 1).

Experiments with the Four-Legged Robot

We studied varying genetic operator probabilities and a reduced function set. The robot architecture (the four-legged robot), fitness function (niceWalking), and the simulation time remained unchanged.

Experiment 1 – Evolution with high mutation rate. We have chosen a high mutation probability of 80% and a crossover probability of 5% for the first experiment. The reproduction probability was defined to be 15%. Each evolution took place in a population of 100 individuals, which have been initialized randomly with an arbitrarily chosen number of between 100 and 1000 program lines. Table 1 shows the chosen genetic parameter settings. The simulation time was set to one minute. Overall, the *best fitness* averages out at 0.19 m/s. The best fitness is reached after 270 generations on average (Fig. 3). As expected, the standard error of the *average fitness* is small. This allows to state that all experiments with small mutation rate have a similar performance. In sum, by using a dominant mutation operator the GP System was capable to evolve various robot programs that have moved the four-legged robot.

Experiment 2 – Evolution with a high crossover rate. By using a high crossover rate of 80% in combination with a low mutation rate of 5% we have chosen the most usual genetic parameter settings for the second experiment. The other parameter settings remained unchanged (Tab. 1). So the effect of a changing dominant genetic operator should be analyzed. Figure 3 shows the average development of the best fitness over the time. Overall, the *best fitness* averages out at 0.26 m/s. The best fitness is reached on average after 240 generations. Similar to the first experiment, the standard error of the *average fitness* is small. Again, the experiments show related performance. Compared to exp. 1, a high crossover rate seems to result in a better quality of the solutions.

Experiment 3 – Evolution with a reduced robot instruction set. While the first two experiments used all instructions available for the robot programs, we analyzed the effect of a *limited instruction set* (Tab. 1). As expected, the development of the *best fitness* is not as fast as in the first experiments, which can be explained with the reduced instruction set (Fig. 3). The best fitness of 0.08 m/s on average is reached after 290 generations. The standard error is small, pointing to the alike

Table 1. GP parameters

Parameter	**Exp. 1**	**Exp. 2**	**Exp. 3**	**Exp. 4**	**Exp. 5**
objective	fast linear movement				
terminal set	random constants				
function set	MOVE, SENSE, COPY, LOAD, ADD, SUB, MUL, DIV, MOD, DELAY, MAX, MIN		MOVE, SENSE, COPY, LOAD, ADD	see exp. 1	
crossover prob.	5%	80%	5%	80%	
mutation prob.	80%	5%	80%	5%	
robot model	four-legged			modified four-legged	
initialization	random				50% pre-evolved, 50% random
population size	100				
selection	tournament				
max. prg. length	100 lines				
min. prg. length	1000 lines				
termination	after 300 generations				
reprod. prob.	15%				

performance of all experiments. Although a strongly reduced instruction set was used, the GP System was capable to evolve different effective robot programs. This result underlines the assumption that the set of supported instructions can be subdivided into sets of *necessary* and *not necessary* instructions. In parallel, this result also proves the power of GP in finding good solutions in a limited solution space. The solutions of experiments 1-3 clearly show differences in quality. The high crossover rate is superior to the other two settings, while a reduced instruction set only allows poor walking speed.

Experiments with an Altered Four-Legged Robot

In a second series of experiments the kinematic structure of the four-legged robot was altered while the general control parameters remained unchanged. The initial GP population of a subsequent evolution cycle was augmented with previously evolved control programs. This reflects the situation in which a changing kinematic structure – maybe due to a technical defect – and/or a changing environment (e.g. change of terrain from flat to rugged) require an adequate response in robot control.

Experiment 4 – Four-legged robot with stiff leg. To test the capabilities of our system to cope with changes of the robot's kinematic structure we altered the four-legged robot by immobilizing the joints of one leg. This was achieved by modifying the joint limits. First we evolved control programs with the same set of

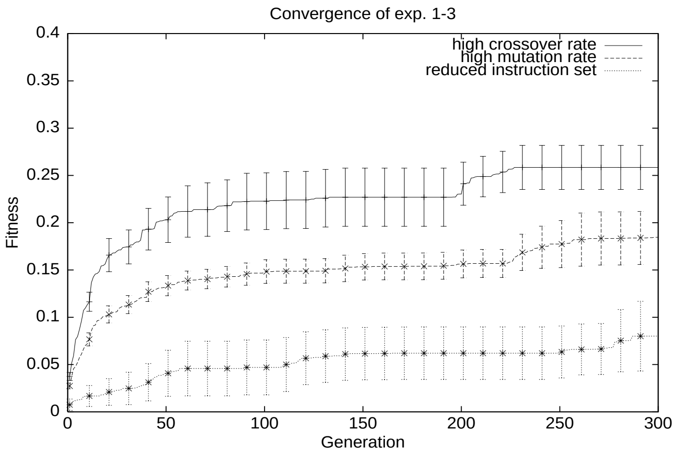

Fig. 3. Mean fitness and the standard error of exp. 1-3. Top: Evolution with high crossover rate. Middle: High mutation rate. Bottom: Reduced instruction set.

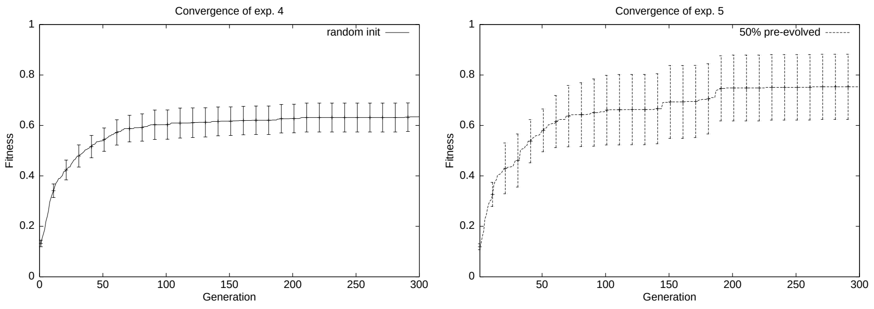

Fig. 4. Mean fitness and standard error of exp. 4 and 5. Left: New evolution of four-legged robot with stiff leg. Right: Evolution using pre-evolved individuals.

parameters as in experiment 1 (parameters see Tab. 1). The resulting programs differed from the programs of exp. 1–3 in that they successfully compensate the reduced degrees of freedom. It is interesting to note that the quality of movement is visibly better than in all earlier experiments (Fig. 4, left).

Experiment 5 – Four-legged robot with stiff leg, individuals from previous exp. We repeated the evolution with only fifty individuals created randomly and another fifty individuals taken from the evolved population of experiment 1. The best fitness of all runs seems to be about 30% higher (i.e. the robot moves faster) than in exp. 4, but the standard error is greater, indicating highly varying runs. In general, it was not the case that using individuals from earlier experiments results in faster evolution. Pre-evolved programs seem to be highly adapted to their architecture and thus may prevent better progress and quality when re-used for other morphologies. However, it can be advantageous to import individuals from other experiments. To make more detailed statements, further experiments with different handicaps and different ratios of randomly initialized and imported individuals need to be analyzed. Interestingly, this intuitive approach does not lead to as big improvements as expected.

Achieved Walking Strategies

In fact nearly all evolutionary runs showed good results by means of regularly walking with speeds up to 0.85 m/s. It seems that our system evolves more successful programs for the altered version of the four-legged robot. We observed mainly four different kinds of moving. (i) Some runs led to walking strategies which only kept the avoided fast and sudden movements. As a consequence this behavior limited the speed of walking. (ii) These strategies used just one leg to generate an impulse in the walking direction. The other legs are only used for stability. (iii) In these cases one leg pushes the robot forward while the other legs are again stabilizing it. (iv) Another kind of walking was observed only for the altered four-legged robot. In contrast to the other mentioned strategies all three flexible legs of the robot are used. This results in a continuous and fast natural like movement.

6 Conclusions

We presented *SIGEL*[1], an integrated software package that combines the simulation and visualization of robots with a Genetic Programming system for the parallelized evolution of walking. The capabilities of the system were investigated with six different robot morphologies. We could show that it is possible to get control programs for arbitrary walking robots without having insight into their architecture and kinematic structure. The advantages of this approach are two-fold: On the one hand, people without engineering background are enabled to control walking robots. On the other hand, engineers are less restricted in the design of robots since the system allows rapid prototyping and testing.

The successful automatic compensation of a technical defect by using preliminary evolved control programs saves time for reprogramming the walking algorithm. In conjunction with the detection of either unpredictable environmental conditions or malfunctioning of parts of the robot, our system increases the robustness of the robot control. The download of off-line evolved control programs to real robots is an important step in this direction and will be the topic of our future work.

[1] More information on the system *SIGEL*, including experiment files, screenshots and video files showing the walking robots can be found on the web (http://ls11-www.cs.uni-dortmund.de/people/sigel/).

References

[1] C. Aue, A. Benkacem, M. Gregorius, A. Ross, S. R. Abdallah, D. Sawitzki, V. Strunk, H. Türk, M. C. Varcol, J. Busch, and J. Ziegler. Simulator für GP-evolvierte Laufrobotersteuerungsprogramme - PG 368. Technical report, Department of Computer Science, University of Dortmund, Germany, 2001.

[2] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann, dpunkt.verlag, 1998.

[3] V. Braitenberg. *Vehicles: experiments in synthetic psychology.* MIT Press, 1984.

[4] R. A. Brooks. New approaches to robotics. *Science*, 253:1227 – 1232, 1991.

[5] H. Cruse. Coordination of leg movement in walking animals. In *From animals to animats. Intl. Conf. on Simulation of Adaptive Behavior*, pages 105–119, 1991.

[6] P. Dittrich, A. Bürgel, and W. Banzhaf. Learning to control a robot with random morphology. In *Proceedings Evo-Robot-98, P. Husbands and J.-A. Meyer, Eds.*, pages 165–178, 1998.

[7] S. McMillan et al. *DynaMechs (Dynamics of Mechanisms): A Multibody Dynamic Simulation Library.* Ohio State University
Internet Access (last access 22.08.2001) via http://dynamechs.sourceforge.net/.

[8] J. J. Grefenstette and A. C. Schultz. An evolutionary approach to learning in robots. In *Machine Learning Workshop on Robot Learning*, New Brunswick, NJ, 1994.

[9] M. Komosinski and S. Ulatowski. Framsticks - Artificial Life. In *ECML '98 Demonstration and Poster Papers, Chemnitzer Informatik Berichte*, pages 7–9, 1998.

[10] J. R. Koza. *Genetic Programming.* MIT Press, Cambridge, MA, 1992.

[11] M. A. Lewis, A. H. Fagg, and A. Solidum. Genetic programming approach to the construction of a neural network control of a walking robot. In *Proceedings of the 1992 IEEE InternationalConference on Robotics and Automation*, pages 2618–2623, Nice, France, May 1992.

[12] J.-A. Meyer. Evolutionary approaches to walking and higher-level behaviors in 6-legged animats. In Gomi, editor, *Evolutionary Robotics II: From Intelligent Robots to Artificial Life (ER'98).* AAAI Books, 1998.

[13] M. Olmer, W. Banzhaf, and P. Nordin. Evolving real-time behavior modules for a real robot with genetic programming. In *Proceedings of the international symposium on robotics and manufacturing*, Montpellier, France, May 1996.

[14] Karl Sims. Evolving virtual creatures. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 15–22. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[15] G. F. Spencer. Automatic generation of programs for crawling and walking. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, page 654, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[16] J. Ziegler and W. Banzhaf. Evolution of robot leg movements in a physical simulation. In K. Berns and R. Dillmann, editors, *Proceedings of the Fourth International Conference on Climbing and Walking Robots, CLAWAR*, pages 395–402, Bury St Edmunds, London, UK, 2001. Professional Engineering Publishing.