Learning to Move a Robot with Random Morphology

Peter Dittrich[1] , Andreas Bürgel[1] and Wolfgang Banzhaf[1 2]

[1] Dept. of Computer Science, University of Dortmund,44221 Dortmund, Germany
http://ls11-www.informatik.uni-dortmund.de
dittrich | buergel | banzhaf@LS11.informatik.uni-dortmund.de
[2] Presently at: International Computer Science Institute, Berkeley, CA, 94708

Abstract. Complex robots inspired by biological systems usually consist of many dependent actuators and are difficult to control. If no model is available automatic learning and adaptation methods have to be applied. The aim of this contribution is twofold: (1) To present an easy to maintain and cheap test platform, which fulfils the requirements of a complex control problem. (2) To discuss the application of Genetic Programming for evolution of control programs in real time. An extensive number of experiments with two real robots has been carried out.
Keywords genetic programming, real-time robotics, random morphology robot, hardware evolution

1 Complex Bio-Inspired Robots

Conventional industrial robots are designed in such a way that a model can be derived easily and the inverse kinematic can be calculated. In operation, the inverse kinematics is used to compute the trajectory for movement between given points in the working area of the robot. Connections between actuators are made as sticky as possible to yield (near) linear behaviour [7]. Perception relies on sense-model-plan-act cycle, where for planning a mostly predefined model of the system is required.

For the development of robots which are inspired by biological systems [1] "controllability" is not a primary design principal. Thus, their actuators are mostly dependent. A model usually does not exist, is very hard to derive or too complex so that a model-based calculation of motor commands requires too much time for reactive tasks. There is no obvious optimal control strategy for a desired action (e.g. movement) because of the complex interdependencies of all the actuators and a non-linear feedback from the environment. Examples of (at least partially) bio-inspired robots are modular robots like the robot snake [2, 3] and robot fishes like the robot tuna build at MIT Ocean Engineering [4].

But even if a model exists, a robot can get into a situation where this model is not valid anymore, e.g. through malfunctioning of parts. If the control strategy is based on the model and an unexpected error occurs (e.g. the breaking of a

[1] Shortly called: bio-inspired robot.

joint between two actuators) the model breaks down and the control strategy is likely to fail [2].

In this case a learning mechanism would be very useful that is able to generate a new control program adapted to the new situation.

In other words, every robot can turn into what we call a random morphology ("RM-") robot, where "RM-robot" refers to a robot with an arbitrary, complex architecture. In this paper, we study a 6-servo robot (see Figure 1) as an RM-robot. The term RM-robot does, however, *not* imply non-deterministic behaviour.

If the robot is on its own, an adaptation mechanism is needed which is able to cope with an unexpected architecture and which makes as few assumptions as possible about the hardware. In the following we will (1) present an easy to maintain real robot platform to test such mechanisms and (2) discuss Genetic Programming (GP) as a mechanism to cope with a RM robot.

It should be noted that the RM robot is also inspired by Sims's work on evolving morphologies [1]. From this point of view the RM robot can be seen as a step towards a physical instantiation of Sims's virtual box creatures.

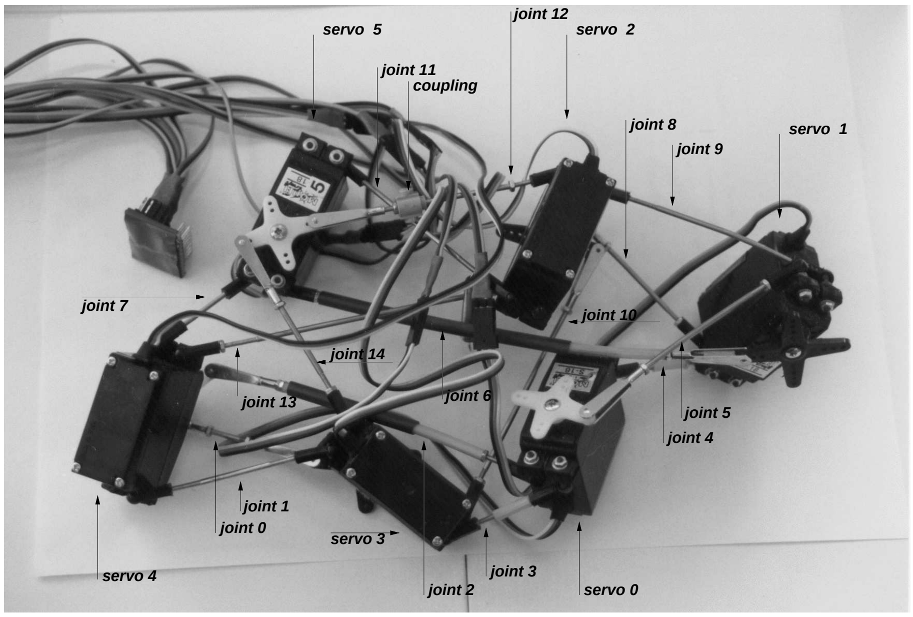

Fig. 1. The mechanics of the random morphology (RM-) robot. It consists of six conventional servos coupled arbitrarily by thin metal joints.

[2] Strictly speaking. the breaking of a joint is not an unexpected error, because we have already expected it. Thus, its very hard, maybe impossible, to model unexpected errors.

2 The RM-Robot

Actuators:

The RM robot is composed of a couple of servos which are connected arbitrarily (randomly). The servos are conventional cheap RC (remote control) servo motors available for hobby air planes and cars. These devices possess a complete servo system including: motor, gear box, feedback device, servo control circuitry, and drive circuit. The connections are made by brass poles also available for hobby modelling. They can be easily connected to the servos, thus one can set up or change an architecture quickly, which should be useful for evolutionary experiments in hardware. The complexity of the mechanics can be increased by connecting poles and servos with springs.

Sensors:

Movement of the RM-robot is measured by a computer mouse device, mechanically connected to the robot. This device allows precise measurement of motion in the 2-D plane. There are also light detecting sensors which are, however, not considered in this contribution.

Control:

The servos are controlled by a pulse signal that occurs at about 50 Hz. The width of the pulse determines the position of the servo motors. To generate this signal we use a simple micro-controller, connected to the host computer by a serial RS232 interface. The host is a PC running LINUX which is fast enough even without a real-time LINUX kernel. A piece of interface software was written to control the servos via the serial RS232 line and to measure analog voltage input via a A/D PC card. Figure 2 shows the overall system architecture.

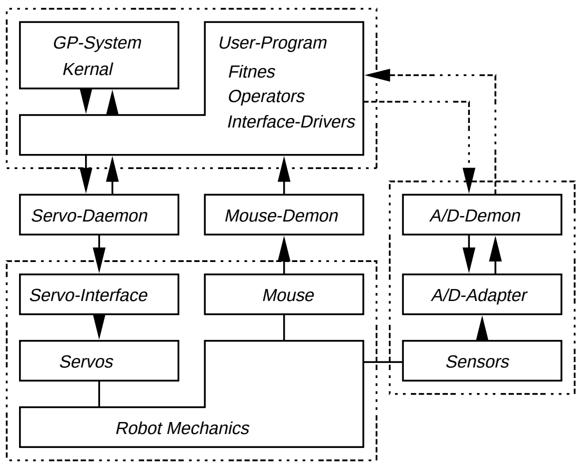

Fig. 2. Overview of the system architecture.

Discussion:

The system is composed of conventional and cheap parts. It is easy to maintain and to construct. It provides an interesting and very flexible test platform for adaptive and learning algorithms that have to cope with complex, unexpected architectures. Its main limitation is the external control by a desktop computer, which is not a problem from a scientific point of view. A wireless, fully autonomous version would make experiments and demonstrations simpler. At the moment the robot has to be watched constantly by an experimentator because it may interact with its wire which would bias the results.

3 Evolving Control Programs

In this section we will show how control programs can be generated by an evolutionary process using Genetic Programming [10, 5]. There are various ways how Evolutionary Algorithms (EA) can be applied to generate or optimize robot controllers [17, 6, 16]:

1. The fitness evaluation can be performed by a simulator, as in [11]. The advantages of this method are: Different controllers can be tested under exactly the same environmental conditions. A simulation is usually cheaper and faster than a real robot. Stability, robustness and correction correctness of solutions be proven – although not in general – which is important for an industrial application. On the other hand, problems with this approach are: A model must be available and this model or simulation might have artifacts, e.g. deadlocks. The individuals might exploit simplification or artifacts in the model [1].[3] As a result, the evolved controller might not work (reliably) on the real robot.
2. The fitness can be evaluated using the real robot [15, 12, 13]. The major qualitative differences to the simulator-based evaluation are: Fitness evaluation is now a stochastic process and real-world time plays an important role. In addition to the robot learning system, a changing environment must be taken into account where time flow is not synchronised with the learning system. For instance, changes of the environment might be slowed-down or stopped in a simulation whereas this is not possible in the real world. An advantage is that one may encounter "unexpected errors" which do not appear in simulations.
3. A combination of (1) and (2) can be used [17]. A controller generated by a simulation is fine tuned on the real robot. To evolve complex control systems the task is usually divided into subtasks (e.g. behaviours) which are independently evolved either by simulation or using the real robot. The behaviours are combined by an action selection mechanism which can be evolved, too [9].

Fitness evaluation can also be characterised by a time scale:

[3] This effect can be put to use when testing models or simulators.

1. Global or goal-oriented fitness evaluation: The robot is run for a long time during which it is able to reach the desired goal once, or even many times starting from the same or different positions. Fitness can be easily derived from a measure how well the goal has been reached. This method is usually used in simulations.
2. Local fitness evaluation: The robot is run only for a very short time (in [12] only a fraction of a second). The fitness evaluation will be much faster, but defining the fitness function on local actions such that the global goal will be reached is more difficult than in the previous case.

Of course, it cannot be stated in general that one method is better than the other. It is very probable that in most applications a combination of different methods is favourable. Here, we shall concentrate on one method in Genetic Programming which we will use with only a few small modifications.

The motivation for using GP is: (1) We would like to examine whether or not GP is able to learn complex robot movements in real-time and uncover the benefits and limitations of this approach. (2) GP produces automatically programs. A program is a very flexible and most commonly used representation of a computable function.

3.1 The Genetic Programming System

The learning method is a conventional steady-state tree-based GP algorithm using a local fitness evaluation on a real robot with the following settings.

We use local fitness evaluation with a steady-state algorithm. An individual will control the robot only for a short while. For this it is executed n_e times. Its performance is measured as the robots advance in the desired direction. Backward movement will be punished by a factor of two and stagnation will be equal to the worst backward movement so far. The Fitness is evaluated at creation time not during the selection process. In tournaments, an individual is selected by drawing an number of individuals uniformly distributed from the population and choosing the individual with the best (or worst) fitness value. As in in conventional GP one subtree in each parent is selected randomly and exchanged. In mutation, each node is mutated by a probability $p_{nodeMutation}$. For this the node is replaced by a node randomly selected from the set of nodes with the same arity. So, a terminal is always replaced by another terminal. The arity of a node will never be changed by mutation.

The set of functions in Tab. 1 is explained in more detail in Tab. 3. The terminal set is explained in Tab. 3.

3.2 Algorithm

Because in real-time evolutionary learning it is important exactly when and how the fitness is evaluated, the algorithm is given in more detail below. Implementation is based on *gpquick* [14].

Objective	Find a program that moves the robot straight on as far as possible
Raw fitness	The sum of pixels the mouse pointer travels in a desired direction minus the sum of pixels the mouse pointer travels in the opposite direction. (See text.)
Fitness	Equal to raw fitness, except in the case when the raw fitness is zero the fitness is equal to the worst fitness so far encountered.
Executions per fitness evaluation	$n_{repetitions} = 4$
Terminal set	GETSERVO0, GETSERVO1, GETSERVO2, GETSERVO3, GETSERVO4, GETSERVO5, GETSERVO6, CONST
Function set	ADD, SUB, MUL, DIV, SINE, DELAY, SETSERVO0, SETSERVO1, SETSERVO2, SETSERVO3, SETSERVO4, SETSERVO5, SETSERVO6, IF, IFLTE, SEQUENCE2, PROG4
Population size	$M = 50, 100$
Maximal number of nodes	$l_{max} = 100, 200$ nodes
Probability of mutation	$p_m = 0.13$
Probability of node mutation	$p_{nodeMutation} = 0.99, 0.15$
Probability of crossover	$p_c = 0.86$
Probability of reproduction	$p_r = 0.01$
Tournament size for genetic operators	$T_r = 4$
Tournament size for replacement	$T_k = 2$
Termination criteria	running time excess or decision by experimenter

Table 1. Koza tableau of the evolution of motion control programs for the RM-robot.

ADD(a,b), SUB(a,b), MUL(a,b), SINE(a)	normal arithmetic operation
DIV(a,b)	protected division, returns 1 if $b = 0$
DELAY(t)	delays the execution of the program for t time steps
SETSERVO0(a), SETSERVO1(a), ..., SETSERVO5(a)	Commands servo to position a. No delay is executed. Returns current position of the servo. Values $a > 127$ and $a < -127$ result in a maximal left or right turn, respectively.
IF(a,b,c)	if $a > 0$, returns b, else c
IFLTE(a,b,c,d)	if $a \leq b$, returns c, else d.
SEQUENCE2(a,b)	evaluates a, then b, returns result of b
PROG4(a,b,c,d)	evaluates a, then b, then c, then d, returns result of d

Table 2. Function set of the GP system.

GETSERVO0, GETSERVO1, ..., GETSERVO5	returns current position of servo n
CONST	a fixed random constant out of [-127, 127]

Table 3. Terminal set of the GP system.

Initialisation:

1. Generate a random population P of size M.
2. For each individual in P, evaluate its fitness.

The GP execution cycle:

```
1. Delete one individual from $P$, which is selected by a tournament of size $T_k$
   (worst individual from $T_k$ randomly drawn individuals).
2. Draw a random number $x$ out of $[0, 1]$.
3. If $x < p_c$:
   Select $I_1$ from $P$ by a tournament of size $T_r$.
   Select $I_2$ from $P$ by a tournament of size $T_r$.
   Create offspring $I_o$ via crossover of $I_1, I_2$.
4. If $p_c < x < p_c + p_m$:
   Select $I$ from $P$ by a tournament of size $T_r$.
   Create offspring $I_o$ via mutation of $I$.
5. If $p_c + p_m < x < p_c + p_m + p_r$:
   Select $I$ from $P$ by a tournament of size $T_r$.
   Create offspring $I_o$ with $I_o = I$.
6. Evaluate fitness for offspring $I_o$.
7. Add offspring $I_o$ to population $P$.
```

A tournament of size T_r means that T_r individuals are randomly drawn from the population and the best according to its fitness is selected. Note that the fitness of each individual has been evaluated before, either during initialisation or in step (6). Although this is an efficient procedure, it might become a significant problem, because fitness values can become obsolete. The above algorithm will fail if the environment changes very quickly, which is not the case, however, in the experiments described here. The problem could be overcome by evaluating the fitness again for each individual during a tournament, like in [12].

Evaluation of fitness f of individual I:

```
1. Clear mouse event queue.
2. Execute $n_{repetitions}$ times Individual $I$. This creates a mouse event queue of
   size $n_q$.
3. For each mouse event $i$ set $E_i$ to the distance the pointer moved in the
   desired direction represented by this event. Backward movement results in
   a negative $E_i$.
4. $f \leftarrow \sum_{i=1}^{n_q} E_i$   (Variant B: $f \leftarrow (\sum_{i=1,E_i>0}^{n_q} E_i)^2 - 2(\sum_{i=1,E_i<0}^{n_q} E_i)^2$ )
5. $f_{min} \leftarrow \min(f_{min}, f)$.
6. if $f = 0$, set $f \leftarrow f_{min}$.
```

3.3 Results

Before describing the results with the 6-servo RM robot, preliminary experiments with a 3-servo robot will be shortly summarised.

3.4 Preliminary Experiments

A series of 55 experiments has been performed with the preliminary 3-servo robot shown in Figure 3. An experiment is a run of the robot with the algorithm described above. The preliminary robot consists of two servos able to retard the front wheels and the rear wheels, respectively. The third servo can bend the robot in the middle. In these experiments, we were able to evolve programs for movement in the plane and even on a gradient. The gradient is much more difficult because the two servos functioning as breaks must be synchronised very accurately. Furthermore, the experiments showed that

- punishment of zero-movement decreases the average number of non-moving individuals and increases convergence speed.
- learning how to delay is important. However, increasing the DELAY concentration only (no. of DELAY operations in the population) is not enough. The arguments of DELAY have to be large. This is achieved by multiplication of large numbers or (more seldom) by division by very small numbers.
- a MUL operation as an argument of a SETSERVOn operation usually creates an overflow (a value greater 127) that results in a maximum turn of the servo.

It could not been convincingly shown, that the overflow behaviour is exploited for protection against crossover and mutation.

3.5 Experiments with the 6-Servo Random Morphology Robot

A series of 32 experiments has been performed with the 6-servo RM-robot. As in the previous section an experiment is a run of the GP algorithm described above. A single run lasts several hours. Details can be found in Tab. 4 which gives an overview of all 32 experiments. There are 16 experiments with a population size $M = 50$ and max. individual size $I_{max} = 50$ and 16 experiments with $M = 100$ and $I_{max} = 100(500)$. In addition the operator set has been varied in order to explore the parameter space and to test the robustness of the GP system according to its setting. In each experiment the operator set consists at least of a core set $F_2 = \{SETSERVO0, SETSERVO1, \ldots, SETSERVO5, DELAY\}$ and an arithmetic function.

From these experiments, it becomes clear that the same system which learns to move the 3-servo preliminary robot is also able to move the 6-servo RM-robot. Although it is obvious that no significant conclusion can be made about which parameter setting is best, the following tendencies should be noted:

- Fitness evaluation using variant A (sum over E_i) together with a population size of $M = 100$ is better than using variant B and $M = 50$.

#	1997	f	M	l_{max}	Operator set	Tests	Bst	Wst	Avg
56	11.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV}	8000	69	-25	40
57	14.8	B	50	50	$F_2\cup$ {ADD}	4350	20	-17	9
58	16.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV}	5150	52	-23	29
59	19.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV}	5000	54	-42	22
60	21.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV}	8000	45	-25	14
61	22.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV}	7000	35	-34	14
62	23.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV}	4400	40	-20	7
63	24.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV, IF, IFLTE, GETSERVOX}	7000	53	-29	23
64	25.8	B	50	50	$F_2\cup$ {SUB, MUL}	5600	23	-25	9
65	25.8	B	50	50	$F_2\cup$ {SUB, MUL}	1500	16	-22	3
66	25.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV, IF, IFLTE}	7000	32	-29	9
67	26.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV, IF, IFLTE}	2600	26	-29	10
68	27.8	B	50	50	$F_2\cup$ {ADD, SUB, MUL, DIV, IF, IFLTE}	9100	33	-29	16
71	27.8	B	50	50	$F_2\cup$ {ADD, MUL}	2000	22	-31	4
72	28.8	A	50	50	$F_2\cup$ {ADD, MUL}	7850	22	-22	10
73	1.9	A	50	100	$F_2\cup$ {ADD, SUB, MUL, DIV}	2850	20	-25	5
74	3.9	A	100	100	$F_2\cup$ {ADD, SUB, MUL, DIV}	3800	136	-64	70
75	4.9	A	100	100	$F_2\cup$ {ADD, SUB, MUL, DIV}	4000	22	-18	8
76	5.9	A	100	100	$F_2\cup$ {ADD, MUL}	2500	180	-60	148
77	8.9	A	100	100	$F_2\cup$ {ADD, MUL}	2000	115	-31	82
78	9.9	A	100	100	$F_2\cup$ {ADD, MUL}	4000	20	-14	4
79	10.9	A	100	100	$F_2\cup$ {ADD, MUL}	1800	135	-44	120
80	11.9	A	100	100	$F_2\cup$ {ADD, SUB, MUL, DIV, IF, IFLTE}	3100	26	-23	14
81	13.9	A	100	100	$F_2\cup$ {ADD, SUB, MUL, DIV, IF, IFLTE}	1700	19	-17	9
82	14.9	A	100	500	$F_2\cup$ {ADD, SUB, MUL, DIV, IF, IFLTE}	4100	22	-21	13
83	15.9	A	100	500	$F_2\cup$ {ADD, SUB, MUL, DIV}	3800	148	-24	127
84	17.9	A	100	500	$F_2\cup$ {ADD, MUL}	1400	147	-33	108
85	18.9	A	100	100	$F_2\cup$ {ADD, SUB, MUL, DIV}	1600	177	-14	168
86	19.9	A	100	100	$F_2\cup$ {ADD, SUB, MUL, DIV, IFLTE}	3000	31	-25	13
87	19.9	A	100	100	$F_2\cup$ {ADD, MUL, IFLTE}	2500	77	-22	68
88	22.9	A	100	100	$F_2\cup$ {MUL, IFLTE}	3000	36		26

Table 4. Overview of the experiments with the 6-servo RM-robot. The table shows from left to right: experiment number, day of experiments realization, variant of fitness calculation, population size M, max. number of nodes per individual l_{max}, operator set, total number of fitness evaluation, best/worst/average fitness in final population. $F_2 = \{SETSERVO0, SETSERVO1\ldots, SETSERVO5, DELAY\}$

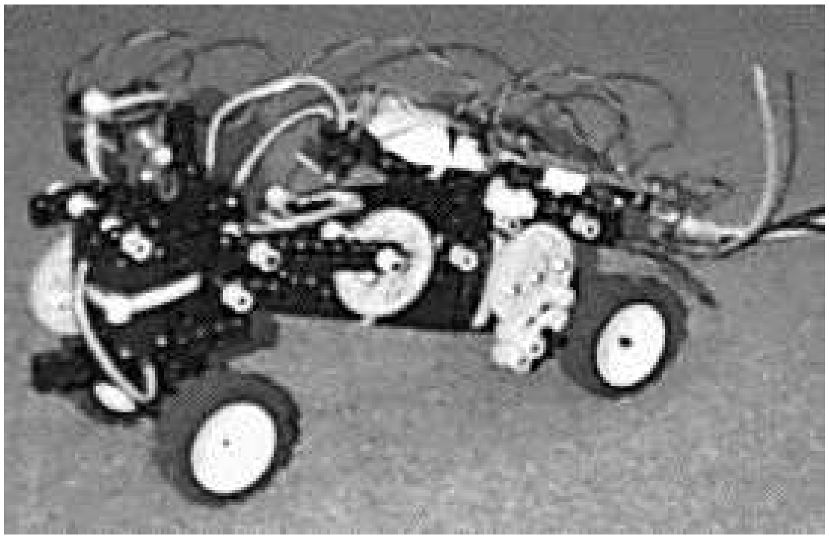

Fig. 3. The preliminary robot with 3 servos. Two servos are used as brakes for the front and rear wheels, respectively. The third servo is able to bend the robot in the middle.

- Conditional operators (IF, IFLTE) are not necessary to evolve good programs and do not increase the quality significantly. On the contrary, table 4 suggests that they inhibit the evolutionary process.
- There is no increase in quality of the best individual when going from $M = 100$ to $M = 500$ (only 3 experiments with $M = 500$ have been performed).
- In successful experiments (e.g. 83 and 84), the program length increases (Figure 4).

Figure 5 shows the average forward movement of the robot during a very interesting run. In this run all six servos are involved into the movement until at time step 3300, an important joint (the clevis of joint 0, Fig. 1) between two servos broke accidently. Joint 0 connects servo 3 with servo 4 and is the only connection to the rudder horn of servo 3. The forward movement decreased drastically. However, after a short while (about 200 fitness tests), the GP system was able to compensate the error.

As an example, Figure 6 gives an impression of the behaviour of the following high-fitness individual that uses all 6 servos for movement:

(DIV (DIV (DELAY (SETSERVO4 (SETSERVO1 (ADD (MUL (SETSERVO5 (SETSERVO1 -103)) (DELAY (DELAY (SUB -56 (SETSERVO0 (SETSERVO5 (SETSERVO2 83))))))) (DIV (DELAY (SETSERVO4 (SETSERVO1 (ADD (MUL (SETSERVO5 (DELAY -40)) (DELAY -38)) (ADD (MUL -79 -8) (ADD (MUL (SETSERVO5 (SETSERVO1 -103)) (DELAY (DELAY (SUB 49 (DELAY (SETSERVO4 (SUB 49 (SETSERVO3 (SETSERVO5 (SETSERVO2 -36)))))))))) (DIV (DELAY (SETSERVO4 (SETSERVO1 (ADD (SETSERVO5 (SETSERVO1 -104)) (DELAY (MUL (SETSERVO5 (DELAY -40)) (DELAY -38))))))) (SETSERVO3 (DELAY (SETSERVO3 -38)))))))))) (SETSERVO3 (SETSERVO3 (SETSERVO0 (SETSERVO4 (DELAY (DIV (DELAY (SETSERVO4 (SETSERVO1 (ADD (MUL (SETSERVO5 (SETSERVO1 -103)) (DELAY (DELAY (SUB 49 (SETSERVO0 (SETSERVO5 (SETSERVO1 -100))))))) (DIV (DELAY (SETSERVO4 (SETSERVO1 (ADD (MUL (SETSERVO5 (SETSERVO1 -104)) (DELAY -38)) (DELAY (DELAY (SETSERVO4 -37))))))) (SETSERVO3 (DELAY (SETSERVO3 -38)))))))) (SETSERVO3 (DELAY (SETSERVO3 -40)))))))))))))) (ADD (MUL (DELAY (SETSERVO4 -37)) (DELAY (DELAY (SUB 49 (SETSERVO0 (SETSERVO0 (SETSERVO5 (DELAY (DELAY (SETSERVO1 -104))))))))))) (DIV (DELAY (SETSERVO5 (SETSERVO1 (ADD (DIV (SETSERVO5 (SETSERVO3 -104)) (DELAY -38)) (DELAY (SUB 105 126)))))) (SETSERVO3 (DELAY (SETSERVO3 -38)))))) (SETSERVO3 (DELAY (SETSERVO3 -40))))

Best individual of experiment 85 in praefix notation (ID: 1320/85)

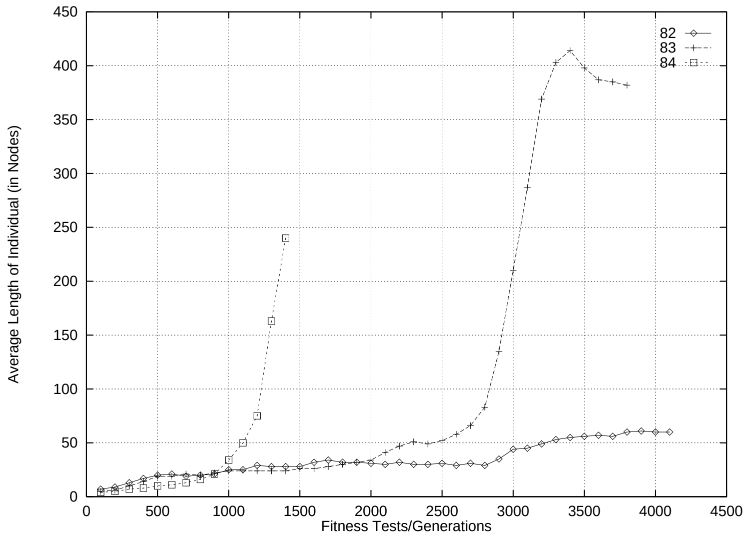

Fig. 4. Average individual length over time of three typical experiments. The successful experiments (83, 84) show a drastic increase of program length compared to unsuccessful run 82.

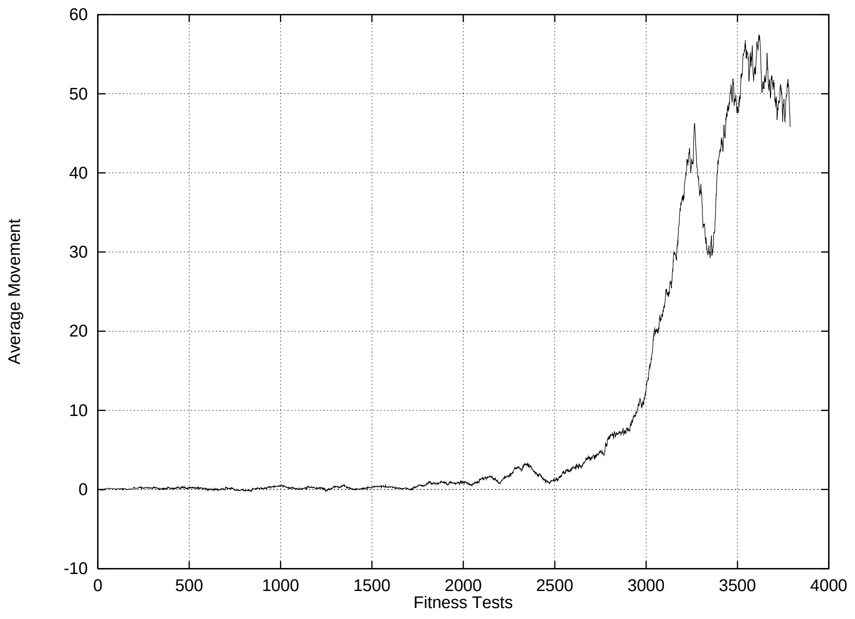

Fig. 5. Adaptation to an unexpected error (experiment 83). The average moved distance (running average over 100 fitness evaluations) is shown over time. The sudden decrease at time 3300 results from a break of an important joint connecting to servos. The increase shows that the system is able to adapt to the new situation.

4 Summary and Conclusion

An easy-to-maintain and cheap robot architecture has been presented, which has been shown to be useful as a platform for testing and demonstrating learning techniques for bio-inspired robots. It has also been shown that Genetic Programming can be used to evolve control programs in real time for an architecture for that no model exists.

The algorithm presented here needs only a very low amount of computational resources because the fitness evaluation that is performed by the robot takes most of the time. The huge amount of remaining processor time can be used to speed up the learning process, e.g. by learning from the past sensory data.

Another future direction will be to include sensor information. In the experiments presented here only the moved distance is measured and used as an input to the learning GP system. By adding sensors, two aspects will become important. The robot has to cope with sensor information (e.g., in order to follow a wall), so the operator set of the GP system has to be extended to process sensory data. Secondly, sensors may provide information about internal states of the robot, which may result in a faster learning process and a robust control strategy.

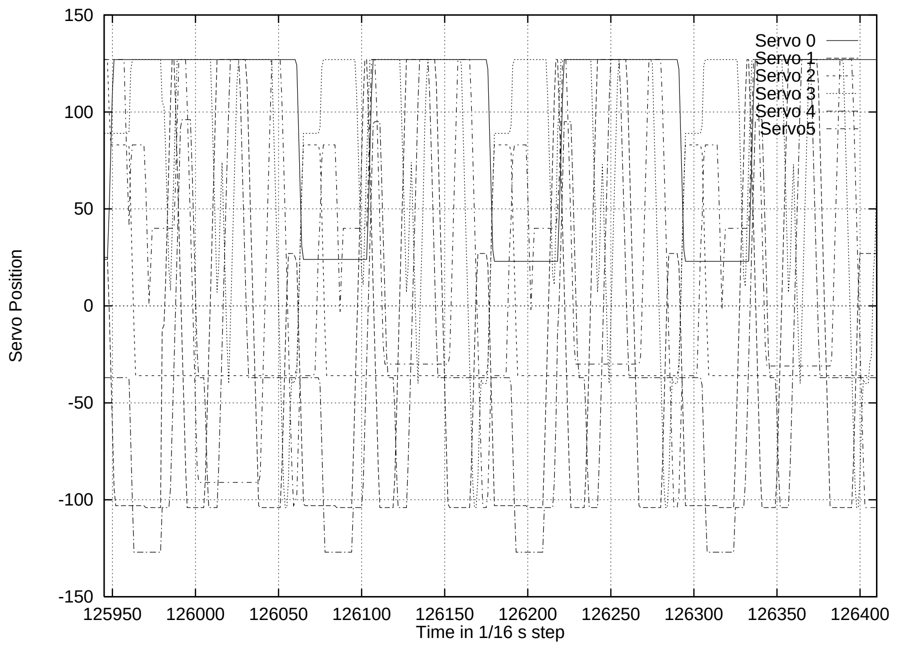

Fig. 6. Behaviour of the best individual of experiment 85. The figure should give an impression of the complex movement where all six servos are involved.

ACKNOWLEDGEMENT

Support has been provided by the DFG (Deutsche Forschungsgemeinschaft) under grant Ba 1042/2-2. W.B. acknowledges partial support by the International Computer Science Institute, UC Berkeley, CA.

References

1. Sims, K.: Evolving 3D Morphology and Behavior by Competition, Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems, Artificial Life IV, pp. 28-39, MIT Press, July (1994)
2. Paap, K. L., Dehlwisch, M., Klaassen, B.: GMD-Snake: A Semi-Autonomous Snake-like Robot, In: Distributed Autonomous Robotic Systems 2, Springer-Verlag, Tokio, (1996)
3. Ostrowski J. P., Burdick, J. W.: Gait Kinematics for a Serpentine Robot, Int. Conf. on Robotics and Automation (1996)
4. Triantafyllou, M. S., Triantafyllo, G. S.: An efficient swimming machine, Scientific American, 272, pp. 64-70 (1995), see also: http://web.mit.edu/towtank/www/projects.html
5. Banzhaf, W., Nordin P., Keller, R. E., Francone F.: Genetic Programming - an Introduction Morgan Kaufmann, (1997)

6. Mataric, M. J., Cliff. D.: Challenges in evolving controllers for physical robots. *Journal of Robotics and Autonomous Systems 19*(1), 67–83 (1996).
7. Davidor, Y.: *Genetic Algorithms and Robotics.* World Scientific, Singapore, 1990.
8. Dorigo, M., Schneph, U.: Genetics-based machine learning and behavior based robotics. *IEEE Transactions on Systems, Man and Cybernetics*, 23(1), 1993.
9. Koza. J. R.: Evolution of subsumption using genetic programming. In F. J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life. Towards a Practice of Autonomous Systems*, pages 110–119, Paris, France, 11-13 December 1992. MIT Press.
10. Koza, J. R.: : Genetic Programming, MIT Press, Cambridge MA, (1992)
11. Lee, W.-P. Hallam, J., Lund. H. H.: Learning complex robot behaviors by evolutionary approaches. In *6th European Workshop on Learning Robots, EWLR-6*, pages 42–51, Hotel Metropole, Brighton, UK, 1-2 August 1997.
12. Nordin, P., Banzhaf, W.: Genetic programming controlling a miniature robot. In E. V. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 61–67, MIT, Cambridge, MA, USA, 10–12 November 1995. AAAI.
13. Olmer, M., Banzhaf, W., Nordin. P.: 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. Singleton, A.: gpquick (Steady-state tree-based C++ GP-Sytem), ftp.cc.utexas.edu/pub/genetic-programming/code.
15. Salomon. R.: Scaling behavior of the evolution srategy when evolving neuronal control architectures for autonomous agents. In *Evolutionary Programming 6 6th International Conference, EP97*, pages 48–57, Indianapolis, Indiana, USA, apr 1997.
16. Steels, L.: Emergent functionality in robotic agents through on-line evolution. In Rodney A. Brooks and Pattie Maes, editors, *Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems ArtificialLifeIV*, pages 8–16, Cambridge, MA, USA, July 1994. MIT Press.
17. Nolfi, S., Floreano, D., Miglino, O., Mondada. F.: How to evolve autonomous robots: Different approaches in evolutionary robotics. In Rodney A. Brooks and Pattie Maes, editors, *Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems ArtificialLifeIV*, pages 190–197, Cambridge, MA, USA, July 1994. MIT Press.