Creation of a Learning, Flying Robot by Means of Evolution

Peter Augustsson **Krister Wolff** **Peter Nordin**

Department of Physical Resource Theory, Complex Systems Group
Chalmers University of Technology SE-412 96 Göteborg, Sweden
E-mail: wolff, nordin@fy.chalmers.se

Abstract

We demonstrate the first instance of a real on-line robot learning to develop feasible flying (flapping) behavior, using evolution. Here we present the experiments and results of the first use of evolutionary methods for a flying robot. With nature's own method, evolution, we address the highly non-linear fluid dynamics of flying. The flying robot is constrained in a test bench where timing and movement of wing flapping is evolved to give maximal lifting force. The robot is assembled with standard off-the-shelf R/C servomotors as actuators. The implementation is a conventional steady-state linear evolutionary algorithm.

1 INTRODUCTION

As a novel application of EA, we set out to replicate flapping behavior in an evolutionary robot [Nolfi and Floreano, 2000]. There is a great interest in constructing small flying machines, and the way to do that might be artificial ornithopters. The continuum mechanics of insect flight is still not fully understood, at least not about controlling balance and motion. According to what was known about continuum equations a couple of years ago, the bumblebee could not fly. The way around this problem could be to give up understanding and let the machines learn for themselves and thereby create good flying machines [Langdon and Nordin, 2001] and [Karlsson et al, 2000]. We propose for several reasons the concept of evolutionary algorithms for control programming of so-called bio-inspired robots [Dittrich et al, 1998] as e.g. an artificial ornithopter. The traditional geometric approach to robot control, based on modelling of the robot and derivation of limb trajectories, is computationally expensive and requires fine-tuning of several parameters in the equations describing the inverse kinematics [Wolff and Nordin, 2001] and [Nordin et al, 1998]. The more general question is of course if machines, to complicated to program with conventional approaches, can develop their own skills in close interaction with the environment without human intervention [Nordin and Banzhaf, 1997], [Olmer et al, 1995] and [Banzhaf et al, 1997]. Since the only way nature has succeeded in flying is with flapping wings, we just treat artificial ornithopters. There have been many attempts to build such flying machines over the past 150 years[1]. Gustave Trouve's 1870 ornithopter was the first to fly. Powered by bourdon tube fueled with gunpowder, it flew 70 meters for French Academy of Sciences[2]. The challenge

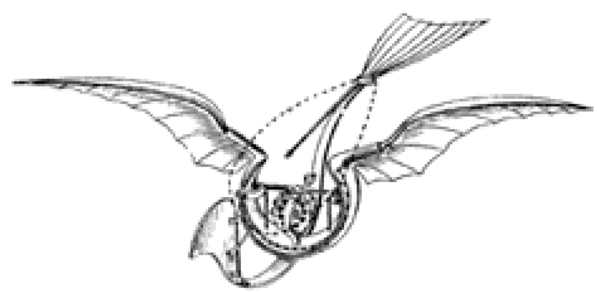

Figure 1: Drawing of Gustave Trouve's 1870 ornithopter.

of building ornithopters attracts an accelerating interest in our days as well and there are many different types of ornithopters. Both manned and unmanned machines, powered by rubber band, compressed air and combustion engines as well as electric engine exist[3]. All of these projects are still dependent of a for-

[1] www.ctie.monash.edu.au/hargrave/timeline2.html
[2] indev.hypermart.net/engine.html
[3] indev.hypermart.net/photos.html

ward motion trough the air and we are not aware of any machine with flapping wings that has succeeded in hovering like a dragonfly or a mosquito. Kazuho Kawai, e.g. has during several years tried to make a manpowered ornithopter fly, but has not succeeded yet[4]. Charles Ellington has created another insect-imitating robot[5]. The purpose of this robot is to learn about the aerodynamics of insect flight, by means of scaling up the insect and thereby get lower velocity and easier measurements. After nine months of design and construction, the flapper was born at a cost of £40 000. Although its slow motion ensured that the flapper would never get airborne, it was perfect for visualizing the detailed airflow over the wings. Conventional aero-

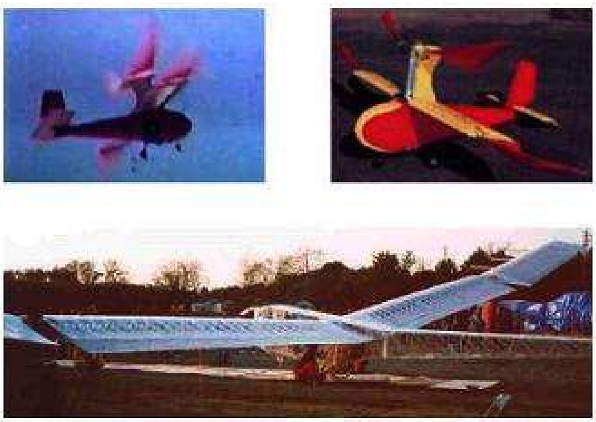

Figure 2: Pictures of some modern ornithopters. First known R/C ornithopter, P.H. Spencer's Orniplane, which took off in 1961 (top). Kazuho Kawai's project Kamura (bottom).

dynamics used in the design of aircraft and helicopters rely on "steady-state" situations such as a fixed wing moving at a constant speed, or a propeller rotating at a constant rate. By contrast, the motion of insect wings is a complex behavior in 3D-space. Within this theory rooted in steady-state situations, it has not been clearly understood why this special motion could generate any unusual sources of lift to explain the insect flight. This picture left out some obvious differences between insects and aircraft. First of all, insects are small. On this smaller scale, viscosity of air becomes more important so that, for the average insect, flying through air is like swimming through treacle. Because of this, the classic airfoil shape that generates an aircraft's lift doesn't work, and insects have evolved entirely different forms of wing structures. At the University of California at Berkeley, a research team at the Robotics and Intelligent Machines Laboratory came to the same conclusion as Ellington, i.e. that the problem is scale dependent[6]. They are now developing a micromechanical flying insect (MFI), which is a 10-25 mm (wingtip-to-wingtip) device, eventually capable of sustained autonomous flight. The goal of the MFI project is to use biomimetic principles to capture some of the exceptional flight performance achieved by true flies. The project is divided into four stages:

1. Feasibility analysis
2. Structural fabrication
3. Aerodynamics and wing control
4. Flight control and integration

Their design analysis shows that piezoelectric actuators and flexible thorax structures can provide the needed power density and wing stroke, and that adequate power can be supplied by solar cells. In the first year of this MURI grant, research has concentrated on understanding fly flight aerodynamics and on analysis, design and fabrication of MFI structures. The Berkeley project does not try to improve nature's way of flying but are more concerned with the actual construction of the robot. There are projects with learning control systems for model helicopters known, but there has not been any project involving learning flying machines with flapping wings reported.

2 METHOD

2.1 EA CONTROL

Simulated evolution is a powerful way of finding solutions for analytically hard problems [Banzhaf et al, 1998]. An ordinary mosquito has a very small computational power (a couple of 100.000 neurons at a very low clock frequency) and yet it is able solves problem that are rather complex. The problem is not only how to move its wings to fly, the mosquito also computes its visual impression and controls its flight path to avoid obstacles and compensates for disturbances like side wind. Beside this performance, it also handles its life as a caterpillar, finds food and a partner to mate with. This is an example of a control system created by nature. Developing such a control system would be impossible for a single programmer and possibly even for an organization. This is why we propose for evolutionary algorithms as a solution candidate for control of such complex systems.

[4]web.kyoto-inet.or.jp/people/kazuho/

[5]www.catskill.net/evolution/flight/home.html

[6]robotics.eecs.berkeley.edu/∼ronf/mfi.html

2.2 IMPLEMENTATION

The implementation is a simple linear evolutionary system. All source code is written in MSVC. We have used some MS Windows-specific functions, meaning that the program is not possible to run from any other platform without some changes in the code. The MS Windows components that are used in the EA class are first; writing to the serial port and second; a function returning the mouse cursor coordinates.

2.2.1 Algorithm

The algorithm use tournament selection of the parents. In this algorithm, only four individuals are selected for the tournament. The fitness is compared in pairs and the winners breed to form new offspring that replaces the losers. This means that there are no well-defined generations but a successive change of the population [Banzhaf et al, 1998] and [Nordin, 1997].

2.2.2 Population storage and program interpreter

However individuals are of random length, there is a maximum length which the individuals are not allowed to exceed. The possible instructions are:

0: Do nothing.

1: Move wings forward to a given angle.

2: Move wings up to a given angle.

3: Twist wings to angle.

Because of these limited possibilities, this implementation cannot form individuals of any other kind even after a change in the fitness function. These evolved individuals are represented as a data structure and cannot be executed without the special interpreting program. This solution has the advantage of not having to wait for a compilation of the code. Of course, it is possible to create compilable code by generating a header, a series of strings from the information from the structure and finally the footer, but this have not yet been implemented. The program is not computationally efficient but there is no need for that. Since an evaluation of one individual can take up to 5 seconds, it does not matter if the rest of the computation takes ten milliseconds instead of one. The interpreter translates the information of the data structure to commands to the control card of the robot. The different stages are:

1. All the instructions of the program are sent to the robot without any delays. This sets the robots starting position at the same as its final. If we do not do that, the individual could benefit from a favorable position from the last program executed.

2. The interpreter waits for one second to let the robot reach the starting position and to let it come to a complete standstill.

3. The cursor is set to the middle of the screen.

4. The instructions are sent to the robot at a rate of 20 instructions per second. Meanwhile, the cursor position is registered every 5 millisecond. The cycle of the program is repeated three times to find out if the program is efficient in continuous run.

The resulting array of mouse cursor position is then passed to the fitness function for evaluation.

2.2.3 Fitness function

The fitness is calculated from the series of mouse cursor coordinates, from the interpretation of the program. Our first intention was to use the average of these coordinates as the only fitness function but we had to add some penalty for "cheating" behaviors.

3 HARDWARE

3.1 ACTUATORS

The robot is assembled with standard off-the-shelf R/C servomotors as actuators. This kind of servo has an integrated closed loop position control circuit, which detects the pulse-code modulated signal that emanates from the controller board for commanding the servo to a given position [Jones et al, 1999]. The signal consists of pulses ranging from 1 to 2 milliseconds long, repeated 60 times a second. The servo positions its output shaft in proportion to the width of that pulse. In this implementation, each servo is commanded to a given position by the robot control program by addressing it an integer value within the interval $\{0, 255\}$.

3.2 ROBOT

Five servomotors are used for the robot. They are arranged in such a way that each of the two wings has three degrees of freedom. One servo controls the two wings forward/backward motion. Two servos control up/down motion and two small servos control the twist of the wings. The robot can slide vertically on two steel rods. The wings are made of balsa wood and solar, which is a thin, light air proof film used for model aircrafts, to keep them lightweight. They are as large as the servos can handle, 900 mm.

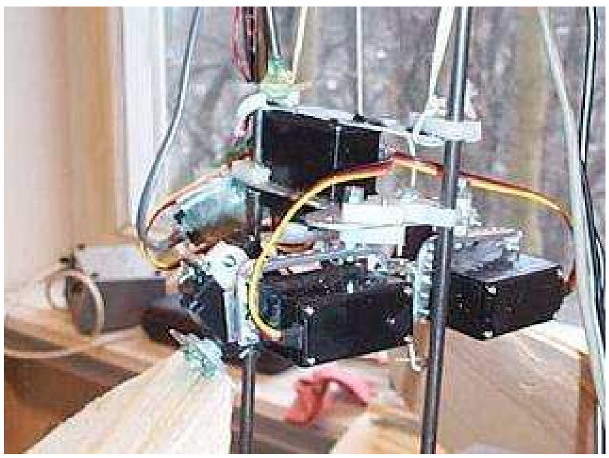

Figure 3: The robot mounted on its sliding rods.

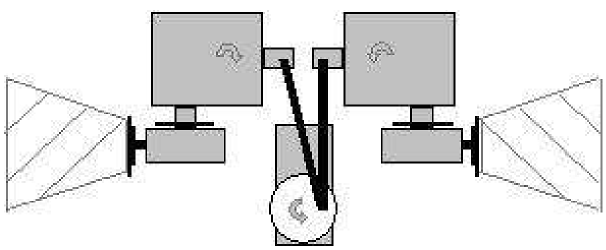

Figure 4: Schematic orientation of the actuators.

3.3 CONTROLLER BOARD

A controller board called Mini SSC II (Serial Servo Controller) from Scott Edwards Electronics Inc. was used as the interface between the robot and the PC workstation, via serial communication. It is possible to connect up to eight servomotors to a single Mini SSC II controller board. The communication protocol consists of three bytes: first byte is a synchronization byte, the second is the number of a servo (0-7), and the last is a position-proportional byte (0-255). The controller board maintains the actuators position according to the last given command, as long as there are no new commands sent.

The robot is placed on two rigid steel rods and is free to slide vertically. In vertical direction, the robot is supported by an elastic rubber band and a string connects the robot to the mouse, which is used to detect fitness during the evolutionary experiments.

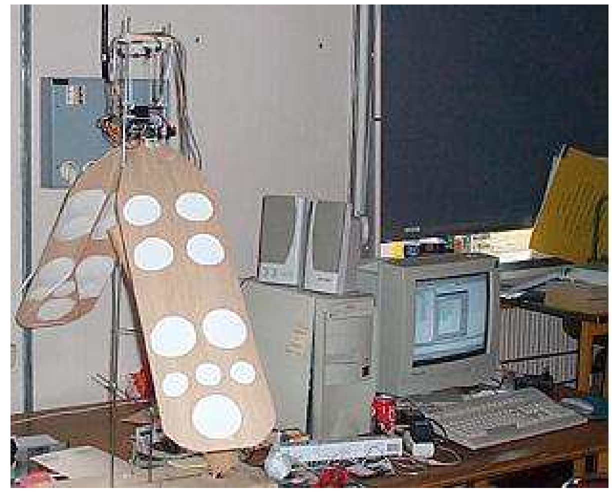

Figure 5: Picture of the robot and the experimental environment.

3.4 FEEDBACK

An ordinary computer mouse gives feedback to the computer, which is a measure of the vertical position of the robot. The mouse takes care about the conversion from the analogue outside world to the digital computer. The mouse is placed above the robot, which is connected to the Y-direction detection wheel of the mouse via a thin string. When the robot moves in vertical direction, the mouse detects the changes in position and the program simply reads the cursor position on the screen [Nordin and Banzhaf, 1995] and [Andersson et al, 2000].

4 RESULTS

The robot was very fast to find ways to cheat. First, it noticed that if it made a large jerk, it could make the thread between the mouse and the robot slide and therefore make the mouse unable to detect its real position. Once, the robot managed to make use of some objects that had, by mistake, been left on the desk underneath the robot. After a few hours the robot was completely still and had twisted one wing so it touched the objects and thereby pressed himself up. It also invented motions that kept it hanging on the rods without sliding down.

Initially, all individuals consist of completely random movements. The fitness of an individual is the average cursor position during the run. At the beginning of the run, the cursor is set to 384, half the height of the screen. A fitness of 200 or lower means that the robot is producing enough lift to carry its weight; i.e. to fly.

Figure 6: Series of pictures showing the best individual of Experiment 1.

4.1 EXPERIMENT 1, VERTICALLY

Immediately the individuals achieve lower fitness. After a couple of minutes all individuals has a better result then just hanging still, which would give the fitness 384. For a couple of hours, the average fitness continues to decrease but finally the result does not improve any more. The lengths of the individuals increase to a certain program length, where they remain fixed. At the random construction of the code, the length is set to somewhere between one and half the maximum program length.

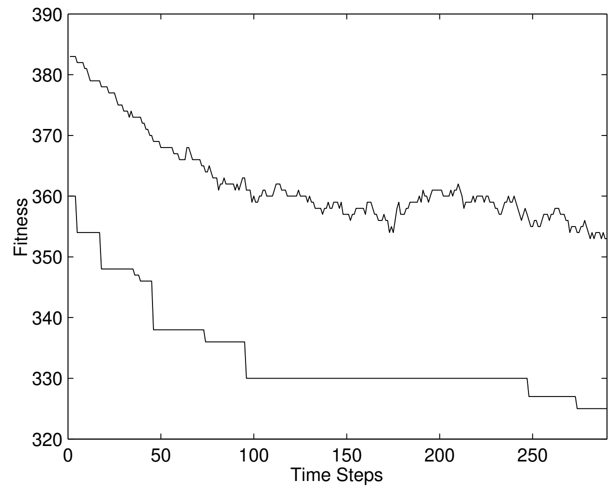

Figure 7: Fitness values from a representative run. Average fitness (top) and best individual fitness (bottom). The time scale is 2.5 hours.

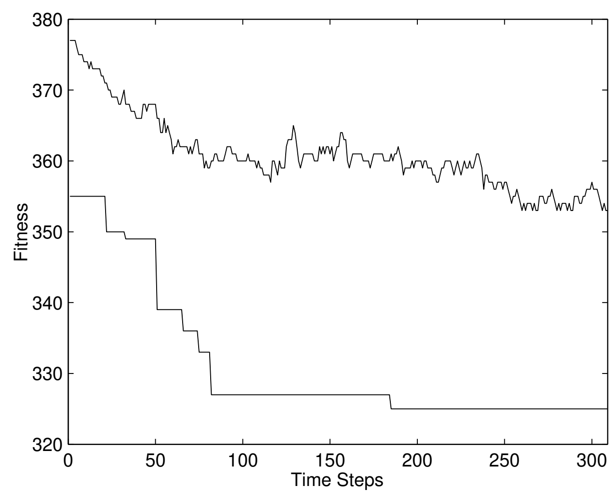

Figure 8: Fitness values of a longer run of 5 hours. Average fitness (top) and best individual fitness (bottom).

The resulting individuals did come up with the result one should have expected. A down stroke with no twist angle of the wings and an up stroke with wings maximally twisted to minimize the resulting downward force.

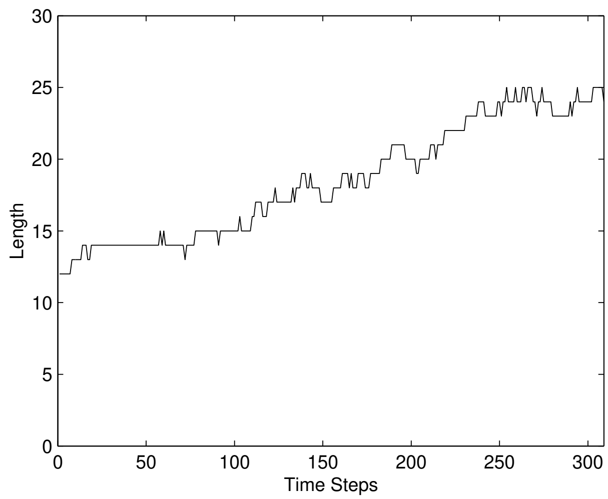

Figure 9: Average program length of the same run as shown in figure 8.

4.2 EXPERIMENT 2, HORIZONTALLY

The second experiment aimed at exploring the possibilities to have the robot to fly in horizontal direction. The experimental set-up is shown in figure 9. The robot is attached to a sleigh, which is free to move along a horizontal rod.

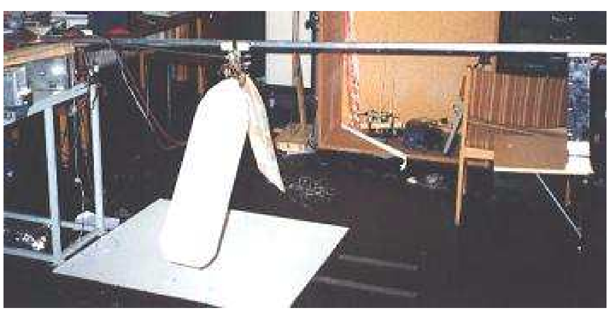

Figure 10: Horizontally flying experiment set-up.

The fitness function in this experiment was set to the average horizontal velocity of the sledge/robot. This function did cause some trouble since the population could not get an over all increasing fitness due to the fact that the track was not infinitely long. The wires to the robot were left hanging to generate an increasing backward force as the robot travels further away. Therefore the fitness of an individual was dependent on the result of the last evaluated individual. As seen in figure 10, the fitness only increases for one hour but the resulting individuals were still getting better. The position of the robot was recorded every tenth second for two minutes and, as shown in figure 11, the evolution is still in progress even after that the fitness has come to a standstill.

This experiment resulted in two different flying behaviors, both present in the small population of 50 individuals after two hours. The two types were one "flying" and one "rowing" program.

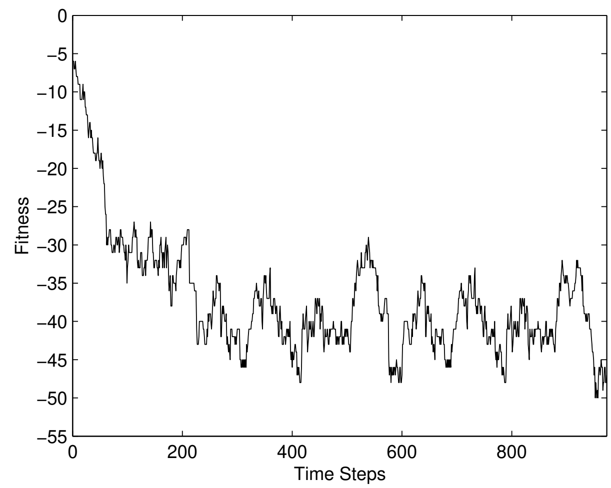

Figure 11: Average fitness of experiment 2, horizontally. The time scale is 2.5 hours.

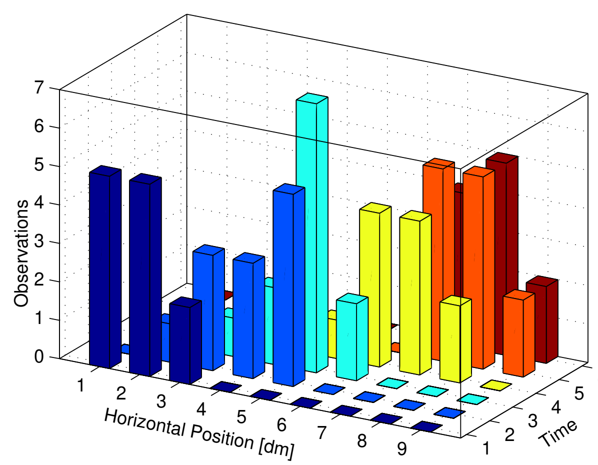

Figure 12: Diagram showing the position of the robot. The position is the distance in horizontal direction from the starting point.

5 SUMMARY AND CONCLUSIONS

We have demonstrated the first instance of a real online robot learning to develop feasible flying (flapping) behavior, using evolution. Using evolutionary robotics in the difficult field of ornithopters could be a fruitful way forward.

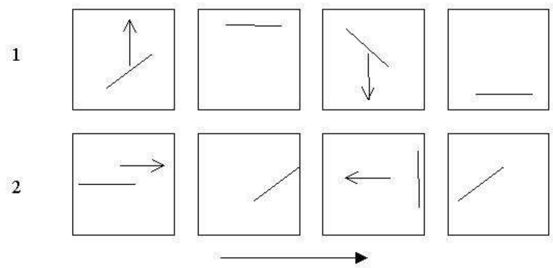

Figure 13: Behavior of the two resulting individuals in experiment 2, horizontally. "Flying" behavior (top) and "rowing" behavior (bottom).

The most interesting future approach would be to supply the robot more power compared to its weight, which should give the robot a reasonable chance of flying. To achieve this, conventional R/C servomotors is not likely to be used as actuators, since they have a rather poor power-to-weight ratio.
Furthermore, providing the robot with a more sophisticated feedback system would give it the possibility to practice balancing in space. Another future development stage is to make the robot autonomous, e.g. carrying its energy source and control system.

References

S. Nolfi and D. Floreano (2000). Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. Massachusetts: The MIT Press.

W. B. Langdon and J. P. Nordin (2001). Evolving Hand-Eye Coordination for a Humanoid Robot with Machine Code Genetic Programming. In *Proceeding of EuroGP 2001,* (pp 313-324). Lake Como, Milan, Italy: Springer Verlag.

R. Karlsson, J. P. Nordin and M. G. Nordahl (2000). Sound Localization for a Humanoid Robot using Genetic Programming. In *Proceedings of Evoiasp2000.* Edinburgh, Scotland: Springer Verlag.

P. Dittrich, A. Burgel and W. Banzhaf (1998). Learning to move a robot with random morphology. Phil Husbands and Jean Arcady Meyer, editors, *First European Workshop on Evolutionary Robotics,* (pp. 165-178). Berlin: Springer-Verlag.

K. Wolff and J. P. Nordin (2001). Evolution of Efficient Gait with Humanoids Using Visual Feedback. In *Proceedings of the 2nd IEEE-RAS International Conference on Humanoid Robots, Humanoids 2001,* (pp. 99-106). Waseda University, Tokyo, Japan: Institute of Electrical and Electronics Engineers, Inc.

J. P. Nordin, W. Banzhaf and M. Brameier (1998). Evolution of World Model for a Miniature Robot using Genetic Programming. *International Journal of Robotics and Autonomous systems,* North-Holland, Amsterdam.

J. P. Nordin and W. Banzhaf (1997). An On-line Method to Evolve Behavior and to control a Miniature Robot in Real Time with Genetic Programming. *The International Journal of Adaptive Behavior,* (5) (pp. 107 - 140). USA: The MIT Press.

F. M. Olmer, J. P. Nordin and W. Banzhaf (1995). Evolving Real-Time Behavioral Modules for a Robot with Genetic Programming. In *Proceeding of ISRAM 1996.* Montpelier, France.

W. Banzhaf, J. P. Nordin and F. M. Olmer (1997). Generating Adaptive Behavior using Function Regression with Genetic Programming and a Real Robot. In *Proceedings of the Second International Conference on Genetic Programming.* Stanford University, USA.

W. Banzhaf, J. P. Nordin, R. E. Keller and F. D. Francone (1998). Genetic Programming An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. San Francisco: Morgan Kaufmann Publishers, Inc. Heidelberg: dpunkt verlag.

J. P. Nordin (1997). Evolutionary Program Induction of Binary Machine Code and its Applications. Muenster, Germany: Krehl Verlag.

J. L. Jones, A. M. Flynn and B. A. Sieger (1999). Mobile Robots: Inspiration to Implementation. Massachusetts: AK Peters.

J. P. Nordin and W. Banzhaf (1995). Controlling an Autonomous Robot with Genetic Programming. In *Proceedings of 1996 AAAI fall symposium on Genetic Programming.* Cambridge, USA.

B. Andersson, P. Svensson, J. P. Nordin and M. G. Nordahl (2000). On-line Evolution of Control for a Four-Legged Robot Using Genetic Programming. In Stefano Cagnoni, Riccardo Poli, George D. Smith, David Corne, Martin Oates, Emma Hart, Pier Luca Lanzi, Egbert Jan Willem, Yun Li, Ben Paechter and Terence C. Fogarty, editors, *Real-World Applications of Evolutionary Computing,* volume 1803 of LNCS, (pp. 319-326). Edinburgh, Scotland: Springer Verlag.