# Robot Gaits Evolved by Combining Genetic Algorithms and Binary Hill Climbing

Lena Mariann Garder
Department of Informatics
University of Oslo
N-0316 Oslo, Norway
lenaga@ifi.uio.no

Mats Erling Høvin
Department of Informatics
University of Oslo
N-0316 Oslo, Norway
matsh@ifi.uio.no

## ABSTRACT

In this paper an evolutionary algorithm is used for evolving gaits in a walking biped robot controller. The focus is fast learning in a real-time environment. An incremental approach combining a genetic algorithm (GA) with hill climbing is proposed. This combination interacts in an efficient way to generate precise walking patterns in less than 15 generations. Our proposal is compared to various versions of GA and stochastic search, and finally tested on a pneumatic biped walking robot.

## Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics—*Propelling mechanisms*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

## General Terms

Algorithms

## Keywords

Evolutionary robotics, Genetic algorithms, Machine learning

## 1. INTRODUCTION

Evolutionary algorithms has often been proposed as a method for designing systems for real-world applications [6]. Developing effective gaits for bipedal robots is a difficult task that requires optimization of many parameters in a highly irregular, multidimensional seach space. In recent years biologically inspired computation methods, and particularly genetic algorithms (GA), have been employed by several authors. For instance, Hornby et al. used GA to generate robust gaits on the Aibo quadruped robot [7]. GA applied to bipedal locomotion was also proposed by Arakawa and Fukuda [1] who made a GA based on energy optimization in order to generate a natural, human-like bipedal gait. One of the main objections to applying GA's in the seach for gaits is the time-consuming characteristic of these techniques due to the large fitness search space that is normally present. For this reason most approaches have been based on offline and simulator based searches. To reduce the time spent searching large search spaces with GA, various techniques for speeding up the algorithm have been presented.

With the increased complexity evolution schema introduced by Torresen [11], Torresen has shown how to increase the search speed by using a divide and conquer approach, by dividing the problem into subtasks in a character recognition system. Haddow and Tufte have also done experiments with reducing the genotype representation [5]. Kalganova [9] has shown how to increase the search speed by evolving incrementally and bidirectional to achieve an overall complex behaviour both for the complex system to the sub-system and from sub-system to the complex system. For an exhaustive description of other approaches readers may refer to Cantu-Paz [2].

The robot presented in this paper is a two-legged biped with binary operated pneumatic cylinders. The search space in our experiments was set up to describe the forward speed of the robot given the different gaits, and the goal was to find the most efficient gait with respect to speed. To enable efficient gaits the search space needed to be quite large as the accuracy of the pause lengths between the different leg positions is outmost critical, especially for gaits dominated by jumping movements. The focus has not been on evolving a balancing system as there have been no other sensory feedback than the forward position of the robot.

The main goal for our work was to find a search algorithm fast enough to enable real-time gait generation/adaptation where the fitness is provided by the mechanical robot without the need for an offline simulator model.

In this paper we present a different approach to increase the search speed by combining GA and binary hill climbing (BH) in an algorithm that we will refer to as the GABH algorithm.

In chapter II we describe the robot hardware and in chapter III we describe how the different gaits are represented in the chromosome. In chapter IV we present the simulated results of different search algorithms compared to the new GABH algorithm, and in Chapter V we present measured results of the GABH algorithm applied to the hardware robot in real-time with no simulator model.


*GECCO'06,* July 8–12, 2006, Seattle, Washington, USA.

## 2. THE ROBOT HARDWARE

The robot skeleton is made of aluminium and is provided with two identical legs. The height is 40 cm. Each leg is composed of an upper part (i.e. the thigh) connected through a cylindrical joint to the lower part (i.e. the calf). Pneumatic cylinders are attached to the thigh and the calf used for controlling the movements of the calf and the thigh separately. As shown in Fig. 1 the rear cylinder in each foot actuates the calf whereas the front cylinder actuates the thigh. The cylinders can either be fully compressed or fully extended (binary operation), and the pneumatic valves are located on top of the robot. The valves are electrically controlled by 4 power switches connected to a PC I/O card (National Instruments DAQ-pad) and the different searching algorithms are implemented in the programming language C++ on the PC.

The pneumatic air pressure was set to 8 bar and provided by a stationary compressor. The robot was attached to a balancing rod at the top (Fig. 1 right and Fig. 2) making the robot able to move in two dimensions. The other end of the rod was attached to a rotating clamp on a hub. The robot walks around the hub with a radius of 2 meter. In addition to being a balancing aid, the rod supplies the robot with air pressure and control signals from the DAQ-pad. The hub has a built in optical sensor representing the rod angle in 13 bit Gray code.

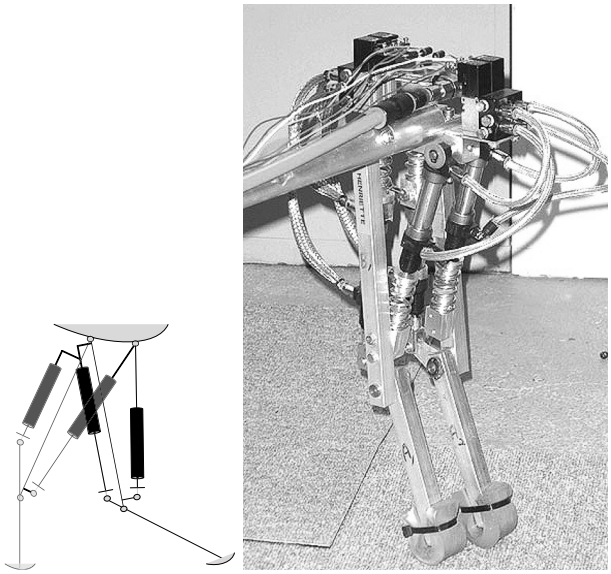

**Figure 1: Illustration (left) and photo (right) of the robot. Proper walking direction is left to right (bird construction).**

## 3. GENETIC ALGORITHM

### 3.1 Simple GA

A genetic algorithm is based on representing a solution to the problem as a genome (or chromosome). The genetic algorithm then creates a population of solutions and applies genetic operators to evolve the solutions in order to find the best one(s). In the simple GA approach [4], [12] the

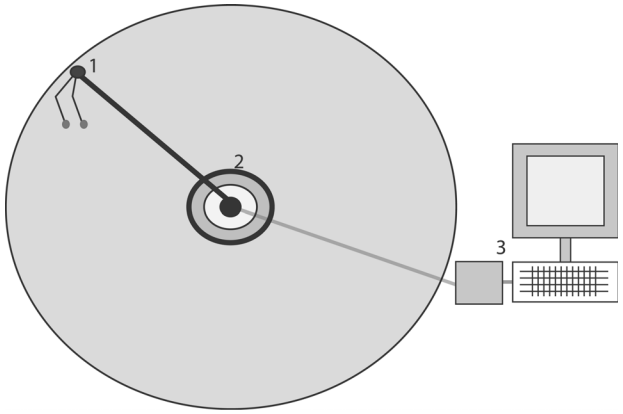


**Figure 2: The fitness measurement and balancing rod system (top view).**

chromosomes are randomly initiated and the only genetic operators used are mutation and crossover. The selection process is done by roulette wheel selection.

### 3.2 The chromosome coding

In our experiments each gait is coded by a 30 bit chromosome. The chromosome represents three body positions each followed by a variable pause. A body position is composed of the positions of the 2 legs (4 cylinders) and represented by four bits (Fig. 3) each describing the status of the corresponding cylinder (compressed or extracted). A complete gait is then created by executing 3 body positions with 3 appropriate pauses in between. Each pause length is represented by 6 bits. The pause length is represented as a binary number corresponding to pauses from 50ms to 300ms. Various simulations have shown no GA search speed improvement by representing the pauses in Gray code.

Two cylinders can move a single leg to 4 different positions. Two legs with four cylinders can hold 16 different positions, and three following positions with 6 bits pauses in between make a search space of

$$2^{30} = 1073741824 \quad (1)$$

different gaits.

Although the search space can be made slightly smaller by representing each gait by a cyclic coding [10] our experiments have shown no noticeable difference in search speed for cyclic/non cyclic coding for this robot. The size of this search space clearly requires a more efficient search algorithm than simple GA in order to enable real-time gait development in hardware.

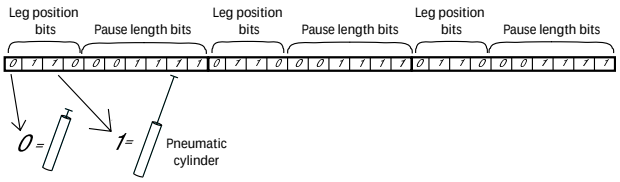


**Figure 3: The chromosome internal coding.**

## 3.3 Pauses

A gait is composed of leg positions and pauses. In our robot evolution we have found that the most efficient gaits with respect to forward speeds are gaits dominated by jumping movements. In a jumping movement the pause length between each leg kick is outmost critical as the robot may stumble if the timing of the leg kick is just slightly wrong. Measurements show that a pause length deviation in the magnitude of 10ms can make the difference between a relatively useless and a highly effective gait. It is however a trade-of between the desire to represent the pause lengths with a high number of bits and the exponential decrease in search speed for each extra bit used due to the increased size of the search space.

# 4. SIMULATED RESULTS

To compare the efficiency of the different search algorithms against each other the robot was first simulated in software.

## 4.1 The simulator

A simple mechanical chicken-robot simulator has been implemented in C++. This simulator models the robot with exact physical dimensions and a weight of 3kg. The centre of gravity is located at the hip joint. It was found very difficult to model the feet-to-floor friction force exactly as this force is heavily modulated by large vibrations in the robot body and supporting rod during walking/jumping. The feet-to-floor friction force is a very important factor for developing efficient jumping patterns and the lack of an exact model for this effect is assumed to be the main weakness of the simulator. The fitness of each chromosome (gait) is a function of the forward speed of the robot caused by the corresponding chromosome. Each gait is repeated 3 times in sequence to reduce the impact caused by the initial leg positions. A movement in the backward direction causes the fitness to be zero.

## 4.2 Search space topology

The optimal search algorithm for a given problem depends heavily on the topology of the search space. For the chromosome coding described in chapter 3.2 and the chosen software robot model we have tried to get an overview of this topology by separating the search space in two parts, one part generated by the pause bits and one part generated by the leg position bits.

Fig. 4 shows a plot of the fitness landscape for all possible leg positions in a single chromosome (gait) were all 3 pause lengths are fixed at 100ms. The size of this search space is $2^{4\cdot3} = 4096$ leg positions. This plot indicates that the part of the overall search space generated by the leg positions is very chaotic although there may be some repetitive phenomena. A similar topology has been found for other choices of constant pause lengths. The different leg positions are sorted by the Gray value of their corresponding bits to keep the bit difference between neighbouring chromosomes in the plot as low as possible, but even so the landscape is chaotic with many narrow peaks.

In Fig. 5 the fitness landscape is plotted for different pause lengths where the leg positions are kept constant. To make the fitness landscape visually informative one of the 3 pause lengths are also kept constant at 70ms resulting in a three dimensional plot. As this plot indicates the part of the overall fitness landscape generated by the pause lengths is smooth and will typically contain a few numbers of maxima. In this type of landscape a hill climbing search will normally be more efficient than a genetic algorithm.

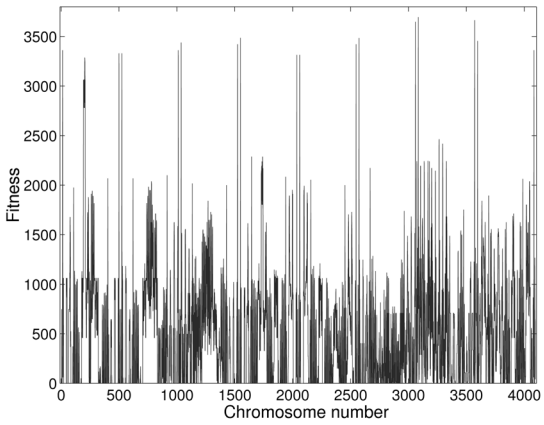


**Figure 4: Fitness search space for different leg positions (fixed pauses at 100ms).**

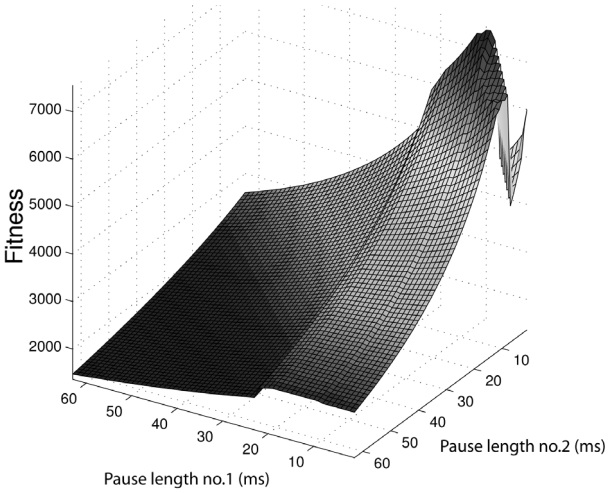


**Figure 5: Fitness search space for pause no. 1 and no. 2. All leg positions and pause no. 3 are fixed.**

## 4.3 Simple GA simulations

The focus for this real-time application has been to find a search algorithm capable of finding an optimal gait in less than 20 generations. The first search approach was to perform a search for an optimal chromosome (gait) in the global search space consisting of $2^{30}$ different chromosome values. Simple and more advanced genetic algorithms were tested against different evolutionary strategies (ES) [4]. ES's showed to be less effective for this particular application and a genetic algorithm was therefore chosen.

In all our simulations 5% noise is added to the fitness function to model practical effect such as variable foot friction,

vibrations, variable air pressure and pause length deviations caused by non-ideal real-time behaviour of the XP operating system.

A simple genetic algorithm with roulette wheel selection, elitism, a population size of 10 chromosomes, no crossover but with as high as 0.2% mutation probability for each bit was found to be the most effective. The high mutation probability indicates that GA is struggling with the topology in this global search space. This result is not surprising as the global search space is assumed to be dominated by the chaotic and complex phenomena shown in the partial search space shown in Fig. 4. In Fig. 7 we see that GA produces slightly less than twice as effective gaits compared to a stochastic search after 15 generations. In all plots each graph shows the mean result from 1000 simulations with randomly initiated populations. 5 different graphs are shown to illustrate the consistency of the simulations.

### 4.3.1 An incremental GA approach

The next approach was to evolve the partial search spaces shown in Fig. 4 and Fig. 5 separately by an incremental genetic algorithm. Incremental GA differs from simple GA because the search space is divided into smaller parts and evolved separately [11] [8]. By gradually evolving each task in series increased complexity can be achieved [3] [1]. The first incremental approach was to first evaluate the leg position bits, with fixed pause lengths. After obtaining gaits with sufficient fitness the leg position bits are fixed and the pause bits are evolved separately. From Fig. 6 we se that this approach is not successful as the fitness is never found to be higher than the fitness provided by simple GA. Leg position bits are evolved up to generation 11 and pause bits are evolved from generation 12.

The next incremental approach was to divide the search in to 7 increments. First the leg position bits were evolved, then the most significant pause bits were evolved, then the next most significant pause bits were evolved until the least significant pause bits were evolved in the last increment. Even this approach was not found to provide better results than simple GA.

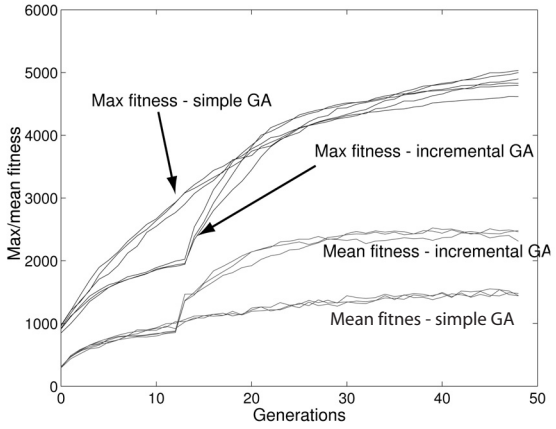


**Figure 6: Incremental GA versus simple GA. Leg position bits are evolved up to generation 11 and pause bits are evolved from generation 12.**

## 4.4 The GABH algorithm

The third and more successful incremental approach was to combine GA and binary hill climbing in the GABH algorithm. From Fig. 5 we notice that the typical pause length fitness landscape is smooth with few maxima. In a practical application disturbances will be added to this landscape due to variable foot friction, vibrations, variable air pressure and pause length deviations caused by non-ideal real-time behaviour of the operating system. However, the main characteristic of this landscape indicates that a hill climbing algorithm may be more efficient than a GA based search.

In the GABH algorithm the leg position bits are first evolved by simple GA up to generation 8. All pause length bits are fixed corresponding to pause lengths of 150ms. In generation 8 GA has normally found a decent leg position pattern. From generation 9 all leg position bits are fixed. In generation 9 all possible combinations of the most significant pause length bits are tested (coarse seach) where all other bits are kept fixed. With 3 pauses in a chromosome there are 8 possible combinations of the most significant pause bits to be tested. The chromosome with the highest fitness containing the most successful most significant pause bits is kept. 8 copies of this chromosome are then made forming generation 10. In generation 10 all combinations of the next most significant pause bits are tested keeping the other bits fixed. The chromosome with the highest fitness containing the most successful next most significant pause bits are then kept. 8 copies of this chromosome are then made forming generation 11 and so on until the least significant pause bits are found in generation 14. The search is then terminated. In this way the search space given by pause lengths is searched in a coarse to fine sequence.

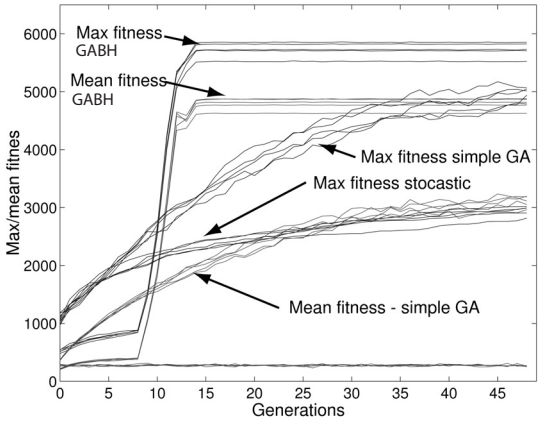


**Figure 7: Comparison between simple GA, GABH and stocastic seach.**

In Fig. 7 the GABH algorithm is compared to simple GA and stochastic search. As each graph represents the average fitness development over 1000 simulations, we see that the GABH algorithm is in average superior to the others in this application where the focus is fast learning in less than 20 generations. A possible objection to the proposed GABH algorithm is that heavy noise in the fitness calculations may cause the algorithm to derail and search in a non optimal region of the search space. To make the algorithm more ro-

bust an improvement could therefore be to let the algorithm run each increment over more than 1 generation and select the optimal chromosome based on fitness averaging.

### 4.5 Gaits obtained

The gaits obtained can be divided into three categories. Two suboptimal gaits and one optimal gait. In Fig.8-10 these gaits are illustrated. The optimal gaits were based on synchronous jumping where both legs are kicking at the same time. By kicking both feet at the same time the most power was available causing the longest jumps. Other suboptimal gaits were based on one-leg jumping or asymmetric jumping where one foot was slightly delayed with respect to the other.

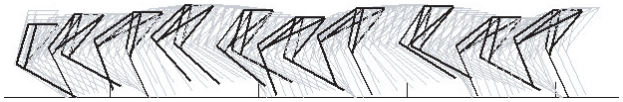

**Figure 8: Suboptimal gait based on asymmetric jumping.**

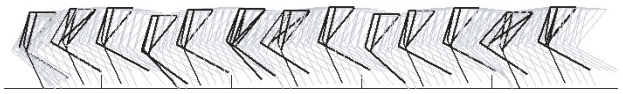

**Figure 9: Suboptimal gait based on every other one-leg jumping.**

**Figure 10: Optimal gaits based on synchronous jumping.**

## 5. MEASURED RESULTS

The GABH algorithm has been tested on the pneumatic robot in an attempt to verify the theory. It was found very difficult to verify the theory accurately due to various practical side effects. One major problem was time consumption and mechanical wear out, particularly of the sandpaper shoe sole which affected the system significantly. When the robot moved, the whole system was vibrating heavily due to the quick contraction/expansion movement of the pneumatic pistons. This vibration made the robot shoe soles occasionally slip during kick-off, and this made the system very unpredictable as the robot occasionally stumbled instead of jumped even for seemingly optimal jumping patterns.

In Fig. 11 two typical fitness developments are shown for the GABH algorithm. In these examples the binary hill climbing starting point was set to the 7th generation. From the measurements we notice an improvement in fitness after this point. After the 13th generation the population was kept static, but even for repeated executions of the same chromosomes the fitness was found to vary significantly due to practical effects such as variable sole friction. However, the algorithm was found to produce proper gaits in less than 10 generations in almost all our experiments. From these few measurements it is difficult to conclude that the algorithm is working significantly better than simple GA. The only conclusion one can make so far from these measurements is that the algorithm itself is working quite well in this very noisy environment.

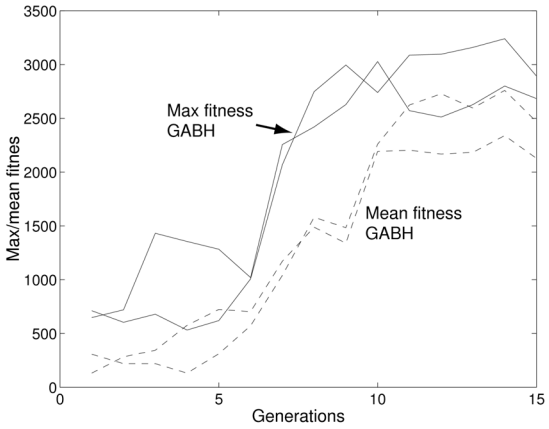


**Figure 11: Measured results.**

## 6. CONCLUSION

This paper has presented an incremental search algorithm combining GA and binary hill climbing. In various simulations this algorithm has shown to develop proper gaits significantly faster than standard GA/ES based algorithms. However, in a physical environment with practical side effects such as highly unpredictable shoe sole friction due to vibrations, varying pneumatic air pressure and wear out it has been difficult to prove in hardware that this algorithm is better than standard GA based algorithms. The algorithm itself, on the other hand was found to perform quite well in a very noisy environment.

## 7. REFERENCES


[1] T. Arakawa and T. Fukuda. Natural motion trajectory generation of biped locomotion robot using genetic algorithm through energy optimization. In *Proceedings of the 1996 IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1495–1500, 1996.

[2] E. Cantú-Paz. A survey of parallel genetic algorithms. In *Calculateurs Paralleles, Reseaux et Systems Repartis*, pages 141–171, Paris, 1998.

[3] D. Floreano and F. Mondada. Hardware solutions for evolutionary robotics. In *Proceedings of the First European Workshop on Evolutionary Robotics*, pages 137–151, London, UK, 1998. Springer-Verlag.

[4] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[5] P. C. Haddow and G. Tufte. An evolvable hardware FPGA for adaptive hardware. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 553–560, California, USA, 2000. IEEE Press.

[6] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu. Real-world applications of analog and digital evolvable hardware. *IEEE Transactions on Evolutionary Computation*, 3(3):220–235, 1999.

[7] G. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto, and M. Fujita. Evolving robust gaits with aibo. In *ICRA*, pages 3040–3045, 2000.

[8] K. De Jong and M. A. Potter. Evolving complex structures via cooperative coevolution. In *Proceedings on the Fourth Annual Conference on Evolutionary Programming*, pages 307–317, Cambridge, MA, 1995. MIT Press.

[9] T. Kalganova. Bidirectional incremental evolution in extrinsic evolvable hardware. In *EH '00: Proceedings of the 2nd NASA/DoD workshop on Evolvable Hardware*, pages 65–74, Washington, DC, USA, 2000. IEEE Computer Society.

[10] G.B. Parker. Evolving cyclic control for a hexapod robot performing area coverage. In *Proceedings of the 2001 IEEE Computational Intelligence in Robotics and Automation*, pages 555–560, Canada, 2001.

[11] J. Torresen. A divide-and-conquer approach to evolvable hardware. In *ICES '98: Proceedings of the Second International Conference on Evolvable Systems*, pages 57–65, London, UK, 1998. Springer-Verlag.

[12] J. Torresen. An evolvable hardware tutorial. In *Proceedings of the 14th International Conference on Field Programmable Logic and Applications (FPL'2004)*, pages 821–830, Belgium, 2004.