Leonardo and Discipulus Simplex:

An Autonomous, Evolvable Six-Legged Walking Robot

Gilles Ritter, Jean-Michel Puiatti, and Eduardo Sanchez

Logic Systems Laboratory, Swiss Federal Institute of Technology,
CH – 1015 Lausanne, Switzerland
E-mail: {name.surname}@di.epfl.ch

Abstract

Evolutionary systems based on genetic algorithms (GAs) are common nowadays. One of the recent uses of such systems is in the burgeoning field of evolvable hardware which involves, among others, the use of FPGAs as a platform on which evolution takes place. In this paper, we describe the implementation of Discipulus Simplex, an autonomous evolvable system that controls the walking of our six-legged robot Leonardo. Discipulus Simplex is based on GAs and logic system reconfiguration, and is implemented into a single FPGA. As a result, we obtained a fully autonomous robot which is able to learn to walk with the aid of on-line evolvable hardware without any processors or off-line computation.

1 Introduction

The idea of applying the biological principle of natural evolution to artificial systems, introduced more than four decades ago, has seen impressive growth in the past few years. Usually grouped under the term *evolutionary algorithms* or *evolutionary computation*, we find the domains of genetic algorithms, evolution strategies, evolutionary programming, and genetic programming [1]. As a generic example of artificial evolution, we consider genetic algorithms.

A genetic algorithm is an iterative procedure that involves a (usually) constant-size population of individuals, each represented by a finite string of symbols, *the genome*, encoding a possible solution in a given problem space. This space, referred to as the *search space*, comprises all possible solutions to the problem at hand. The algorithm sets out with an initial population of individuals that is generated at random or heuristically. In every evolutionary step, known as a *generation*, the individuals in the current population are *decoded* and *evaluated* according to some predefined quality criterion, referred to as the *fitness*, or *fitness function*. To form a new population (the next generation), individuals are *selected* according to their fitness, and then transformed via genetically-inspired operators, of which the most well known are *crossover* ("mixing" two or more genomes to form novel offspring) and *mutation* (randomly flipping bits in the genomes). Iterating this evaluation-selection-crossover-mutation procedure, the

genetic algorithm may eventually find an acceptable solution, i.e., one with high fitness.

One of the recent uses of evolutionary algorithms is in the burgeoning field of *evolvable hardware*, which involves, among others, the use of FPGAs as a platform on which evolution takes place [2, 3]. In this paper, taking advantage of this new methodology, we use an autonomous evolvable system to control the walking of a six-legged robot.

The learning of walking behavior in legged robots is a well-known problem in the autonomous robotic field, because it is non-trivial to solve due to the complexity of the movement synchronization. Several approaches have been used to solve this problem, such as the use of reinforcement learning [4] or subsumption [5]. The proposed solutions are generally bio-inspired and are implemented with one or more processors which can be embedded in the robot or off-line. We believe it is interesting to have a circuit that controls the motions of the robot and can modify its functionality in order to find the right behavior. Thompson *and al.* use such an approach [2] (pp. 156–162), exploiting an evolvable system in order to control the motion of a two-wheeled robot. The resulting system (robot and evolvable hardware) was quite large and for practical reasons they had to simulate the world in which the robot evolved.

In our approach we want to avoid the use of processors and of off-line computations generally needed to solve the walk problem. We use a small autonomous 6-legged robot – *Leonardo* [6] – as a platform, and we develop *Discipulus Simplex*, an autonomous evolvable hardware walking control. Discipulus Simplex controls the motion of Leonardo's legs, and by way of evolution changes its behavior in order to obtain correct walking behavior. The entire hardware fits within a single FPGA. The result is small autonomous 6-legged robot (24cm x 20cm, weighting 1 kg) controlled by an autonomous evolving system implemented in an FPGA. The most important point is that this system evolves by itself with no processor or external aid.

The remainder of this paper is organized as follows. Section 2 describes briefly the mechanics and electronics of the robot Leonardo. Section 3 describes Discipulus Simplex, the evolvable system that realizes the learning of the walking behavior. Finally, in Section 4 we present some conclusion.

2 *Leonardo:* An Autonomous Six-Legged Robot

Leonardo [6] is an autonomous six-legged robot developed as an experimental platform for bio-inspired algorithms. The main features of this robot are its low cost and its small size. These goals were attained by the use of efficient and simple mechanic and electronic parts.

Leonardo's mechanics combine originality, simplicity, and efficiency. The robot has 13 degrees of freedom: 2 degrees of freedom (elevation and propulsion) in each of the 6 legs, and 1 degree of freedom in the body. The latter is the most original mechanical part of the robot and allows the robot to make efficient turns. The body articulation is shown in Figure 1(a). Figure 1(b) shows a front view

of one of Leonardo's legs. Each leg has two servo-motors that control the elevation and the propulsion respectively; furthermore, lateral motions (i.e., a third pseudo-degree of freedom) are allowed by the introduction of an elastic joint. The sensorial part is composed of two simple contacts that indicate whether or not a leg is touching the ground or an obstacle. New sensors can be easily incorporated through the extension ports which are provided in the electronic part of the robot.

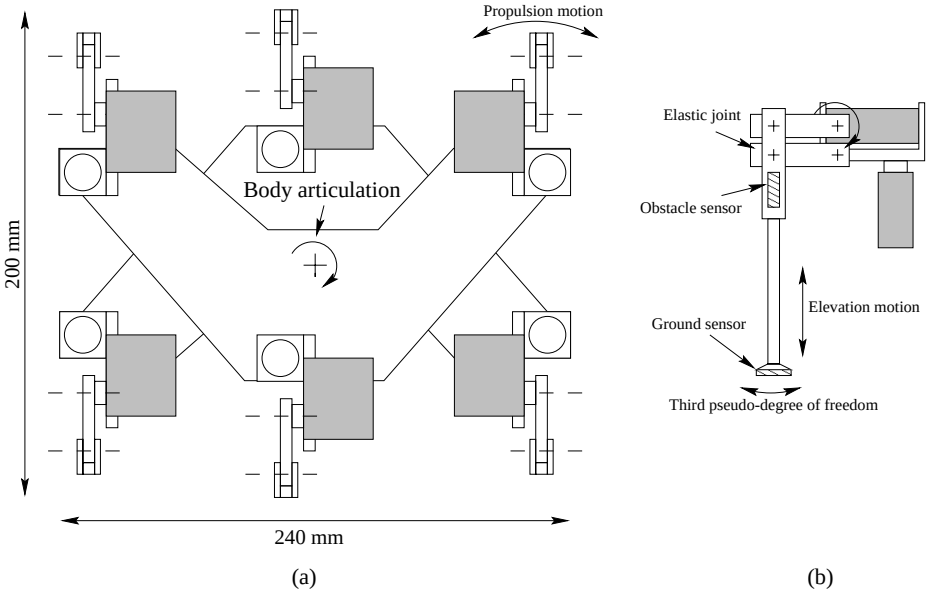

Fig. 1. Leonardo's mechanics: top view of the body (a), front view of a leg (b)

The robot is controlled through a main board, currently existing in two different versions: (1) a processor-based card; and (2) an FPGA-based board which is used for this paper. The processor-based controller is derived from the Khepera robot hardware [7].

The FPGA-based board has been developed specially for this experiment and is composed only of an FPGA (Xilinx XC4036EX), configuration ROM memory, a stabilized power supply that can be driven by an accumulator or an external source of power, and a clock. The FPGA directly controls the servo-motors and the sensors, replacing the micro-controller and the co-processor of the preceding board. Figure 2 depicts Leonardo with the FPGA-based board.

3 *Discipulus Simplex:* An Evolvable Logic System used for Walking Control

The objective for this evolvable hardware system (Discipulus Simplex) is to allow the robot to learn how to walk by itself. To implement this task we use one single

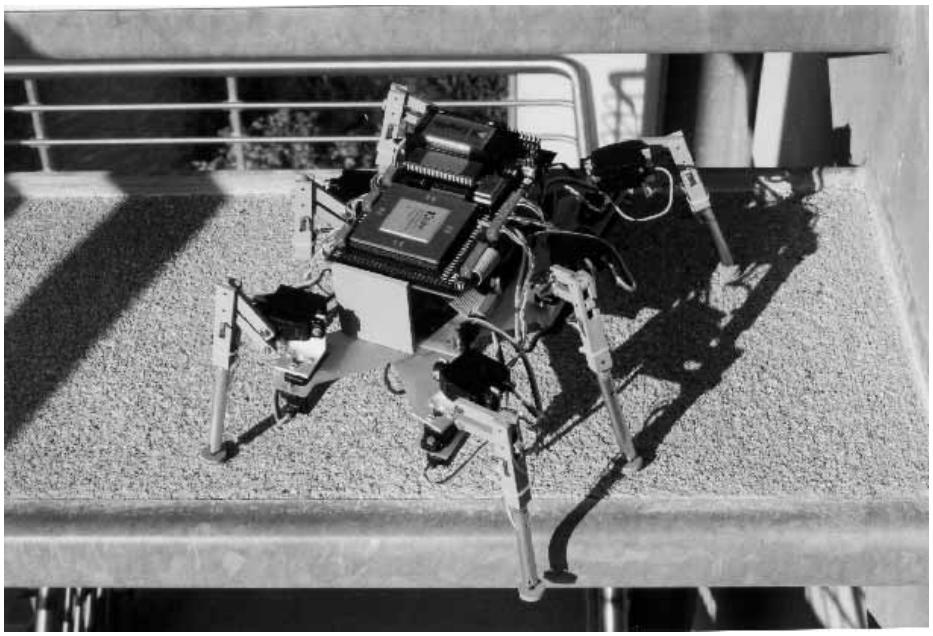

Fig. 2. Leonardo (six-legged robot) and Discipulus Simplex (evolvable hardware walking control)

FPGA which drives each leg of the robot and also performs the learning process. This constitutes an autonomous hardware system which can evolve and drive a robot without any help from processors, while remaining very simple (only a single FPGA chip is needed).

The learning process is realized with a genetic algorithm. The principle is to hold a population of individuals stored in the memory system which encodes a specific walk for the robot. The individual encodes the walk for the robot through a reconfigurable state machine which generates the sequence of movements for the steps. Once executed, the genetic algorithm generates a new population of individuals with average fitness usually better than the previous generation. This continues until a good individual is found for the walking behavior.

Three main modules are thus needed in our system. The first is the reconfigurable logic system for the walking control: it has to change its behavior during the learning process, in accordance with the walking performance. To configure this evolvable state machine we use a genome (individual), encoded by a bit-stream, so as to generate the sequence of movements. The genome with the greater fitness in the current population is provided to the evolvable state machine by the genetic algorithm. The second module performs the learning process: it is the genetic algorithm processor (GAP) responsible for the genetic operations (selection, crossover and mutation) on the population of genomes. Finally, the third module is the fitness module, responsible for the evaluation

of an individual's performance. Figure 3 shows the complete evolvable hardware system, implemented in the FPGA, with the three main modules.

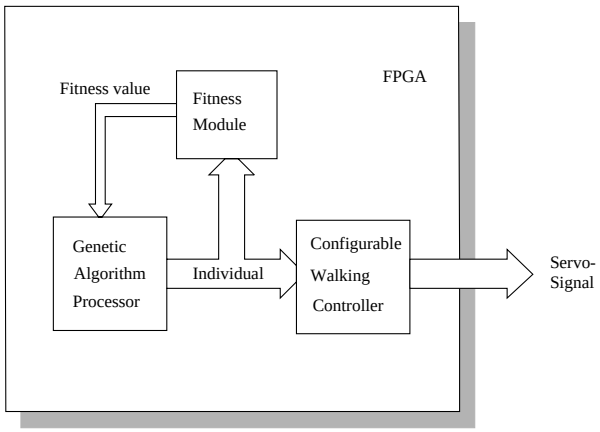

Fig. 3. Global view of Discipulus Simplex: the evolvable hardware system used for evolving walking behavior.

3.1 Reconfigurable state machine as walking control

The walk of the robot is controlled by a state machine which is able to modify its behavior through reconfiguration. This is the part of the system that evolves during the learning process. The behavior of the state machine is encoded in a bit-stream (i.e,, genome) which is provided by the learning process.

A genome encodes two steps of the walk. In each step there are six subparts, one for each leg. This means that the six parts are used and decoded at the same time during the walk. Finally, inside the six parts there are three bits which encode the movement of the leg during the step. The first bit codes whether the leg first goes up or down. The second bit codes whether the leg goes forward or backward. The last bit codes whether the leg goes up or down after the horizontal move. In all, one individual is composed of 36 bits, giving rise to a search space of size $2^{36} = 68$ billion possibilities.

The evolvable walking controller, including all these functionalities, is separated into two distinct modules. The main module is the reconfigurable state machine which is configured by the individual and generates the sequence of movements. The second module generates the signals for the servo-motor of each leg. Figure 4 shows the evolvable walking controller module and its submodules.

There are two servo-controls for each leg which generate PWM (Pulse Width Modulation) signals for the servo-motors from the position given by the parameterizable state machine.

3.2 Genetic algorithm processor (GAP)

To perform the learning process we use a genetic algorithm. The algorithm generates a population of individuals (genomes) and its goal is to continually improve

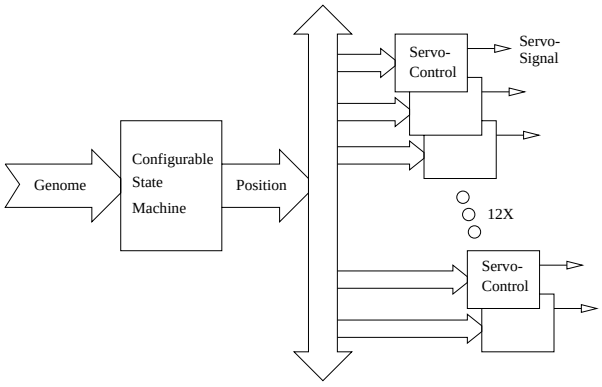

Fig. 4. Evolvable walking controller: module breakdown

the population's walking behavior. During the genetic process, we have to evaluate the performance of each individual. This performance measure is called the *fitness value*.

To evaluate fitness, the first idea is to get information on how the robot walks directly from the robot. For example, with a measure of the distance traveled in a given unit of time and a measure of the equilibrium of the robot with the help of sensors on his legs. But the problem is that the robot has a dynamic constraint and needs to try a genome for about five seconds to execute the walk. This time is too long to be used in our case. Therefore, we had to define a fitness function only in terms of logic computations. We could choose to define fitness by comparing an individual with a known solution; this would, however, diminish our work's interest. We chose to define the fitness with some high-level rules about the dynamics of the robot. After tests and simulations, we retained three rules which give good results, without knowledge of the solution:

1. The first physical rule involves the equilibrium of the robot. For example, if the robot has three legs raised on the same side, it will stumble and fall, resulting in a bad fitness value.
2. The second rule tests the symmetry between the two steps. If a leg goes forward in the first step, it should go backward in the next step. This can be deduced from observation of the walk of animals.
3. The last rule concerns the coherence of the movement of a leg. For example, it is not good for a leg to be down before it goes forward, the robot will go backward. The leg has to be up before going forward. On the other hand, the leg has to be down before doing a propulsion movement (going backward).

These rules are interesting in that they do not include knowledge of the solution genome. They arise from physical considerations, so they can be applied to other robots with other leg configurations.

The genetic algorithm is implemented in a main module called GAP (Genetic Algorithm Processor), which contains different modules that perform each a part

of the genetic algorithm process. The GAP includes the four principal operators for the genetic algorithm: fitness, selection, crossover, and mutation. Each of these operators is implemented in one module, so they can work independently. The GAP also includes a random number generator, an initialisation unit, and a population of individuals. Figure 5 shows the GAP modules with their different components.

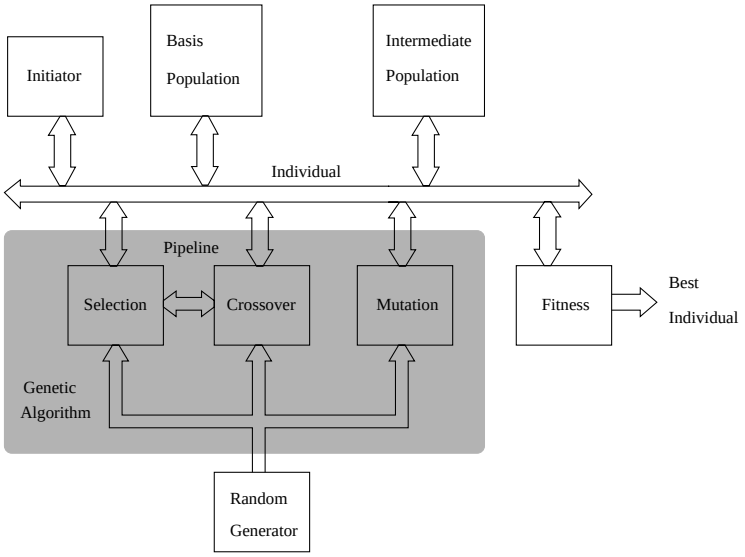

Fig. 5. Genetic Algorithm Processor.

When the GAP starts to work, the initialization module generates a new population of individuals using pseudo-random numbers. Once done, it starts the genetic algorithm cycle.

The four principal operators run in a fixed order. From the initial population the fitness operator is applied, then selection, then crossover, and finally mutation. To decrease computation time by a factor of about two, we ran the selection and crossover operators in a pipeline. During these operations, however, the selection operator needs to read in the population and the crossover operator needs to write the new individuals in an intermediate population. This is why we used two populations of individuals.

There are several possibilities for implementing each genetic operator [8]. In our case, the choices were constrained by the logic system. It is difficult to implement complex functions in a logic system. So we have to choose an operator which can be implemented with low computational cost.

The first operator which runs every time is the random number generator. It generates a new pseudo-random number for all genetic operators at each clock cycle. It is implemented as a one-dimensional cellular machine (XOR system).

It does not depend on the execution of the genetic algorithm, in order to render the evolutionary process less data-dependent.

The implementation choice made for the selection module was that of tournament selection [8] because it does not use real numbers and divisions which are difficult to implement in logic systems. This operator randomly draws two individuals from the population. A threshold defines the probability that the better individual will be selected.

For the crossover operator, the single-point crossover method is used. This method takes two individuals in the population and randomly selects a crossover position in the genome. The two genomes are cut at the crossover point and the part after the point are swapped, creating two new genomes. A threshold define how many crossover operations are performed on the population.

The mutation operator is single-bit mutation. This method randomly flips a bit in an individual's genome.

The last operator to define is the fitness operator. This operator evaluates the performance of each individual, using the high-level rules described above.

3.3 Implementation and tests

The evolvable logic system was synthesized for a Xilinx XC4036ex FPGA, from a VHDL language description. The use of VHDL is interesting because it allows to define parameters such as selection threshold, crossover threshold, population size, etc. So, it is possible to parameterize the entire logic system and it is easy to modify it.

The GAP implements the operators described above: selection, crossover, mutation, and fitness evalutation. The different parameters used for the GAP are:

- Population size: 32 individuals.
- Genome size: 36 bits.
- Selection threshold: 0.8.
- Crossover threshold: 0.7.
- Number of mutations: 15 bits (over 1152 bits).
- Frequency: 1 MHz.

Using the three rules for fitness evaluation, the maximum fitness does not necessarily correspond to the best walk known for the robot. However, the walking behavior found with the maximum fitness respecting all these rules is nonetheless good. To evolve the maximum fitness it needs an average of about 2000 generations. This number is small, if the speed of the GAP is considered. For example, if we had to test all the 68 billion possibilities for the genome, we would need about 19 hours at 1 MHz to obtain the best genome. With this system, the avarage time needed is only about 10 minutes.

The complete system implemented in the XC4036ex FPGA uses 96 percent of the available CLBs, i.e. 1244 CLBs. It represents around 34500 logic gates.

4 Conclusion

In this paper, we have investigated whether evolvable hardware systems can be used for the learning of the walking behavior of a six-legged robot. Using our six-legged robot Leonardo as a platform, we built an evolvable hardware walking control – called Discipulus Simplex – based on GAs and reconfiguration. Discipulus Simplex is implemented into a single FPGA which evolves on-line and is able to learn to walk. As a result, we obtained a fully autonomous robot which is able to learn to walk with the aid of on-line evolvable hardware without any processors or off-line computation.

This realization validates the use of evolvable hardware into autonomous systems, but still is a first step in the direction of a complex autonomous evolvable system. In future work, we will take advantage of the computational power provided by the GAP, and use the same kind of evolvable system in order to solve problems which deal with bigger genomes (i.e., more complex reconfigurable systems) and where the final solution is not known.

5 Acknowledgments

The authors want to thank Moshe Sipper and Gianluca Tempesti for their helpful comments and contributions, and André Badertscher for photographs.

References

1. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Heidelberg: Springer-Verlag, third ed., 1996.
2. E. Sanchez and M. Tomassini, eds., *Towards Evolvable Hardware*, vol. 1062 of *Lecture Notes in Computer Science*. Heidelberg: Springer-Verlag, 1996.
3. M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 83–97, April 1997.
4. W. Ilg and K. Berns, "A learning architecture based on reinforcement learning for adaptive control of the walking machine lauron," *Robotics and Autonomous Systems*, vol. 15, pp. 321–334, 1995.
5. P. Maes and R. Brooks, "Learning to coordinate behaviors," in *Proceedings of the Eighth National Conference on Artificial Intelligence*, (Boston, MA), pp. 796–802, August 1990.
6. J.-M. Puiatti, "Robot 6 pattes," tech. rep., LAMI – Swiss Federal Institute of Technology at Lausanne, 1995.
7. F. Mondada, E. Franzi, and P. Ienne, "Mobile robot miniaturization: A tool for investigation in control algorithms," in *Proceedings of the Third International Symposium on Experimental Robotics* (T. Yoshikawa and F. Miyazaki, eds.), (Kyoto, Japan), pp. 501–513, 1993.
8. M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.