24

An Embedded Evolutionary Controller to Navigate a Population of Autonomous Robots

Eduardo do Valle Simões
University of São Paulo – Department of Computer Systems
Brazil

1. Introduction

This chapter studies evolutionary computation applied to the development of embedded controllers to navigate a team of six mobile robots. It describes a genetic system where the population exists in a real environment, where they exchange genetic material and reconfigure themselves as new individuals to form the next generations, providing the means of running genetic evolutions in a real physical platform. The chapter presents the techniques that could be adapted from the literature as well as the novel techniques developed to allow the design of the hardware and software necessary to embedding the distributed evolutionary system. It also describes the environment where the experiments are carried out in real time. These experiments test the influence of different parameters, such as different partner selection and reproduction strategies. This chapter proposes and implements a fully embedded distributed evolutionary system that is able to achieve collision free-navigation in a few hundreds of trials. Evolution can manipulate some morphology aspects of the robot: the configuration of the sensors and the motor speed levels. It also proposes some new strategies that can improve the performance of evolutionary systems in general.
Ever more frequently, multi-robot systems have been shown in literature as a more efficient approach to industrial applications in relation to single robot solutions. They are usually more flexible, robust and fault-tolerant solutions (Baldassarre et al., 2003). Nevertheless, they still present state-of-the-art challenges to designers that have difficulties to understand the complexity of robot-to-robot interaction and task sharing in such parallel systems (Barker & Tyrrell, 2005). Often, designers are not able to predict all the situations that the robots are going to face and the resulting solutions are not able to adapt to variations in the working environment. Therefore, new techniques for the automated synthesis of robotic embedded controllers that are able to deal with bottom-up design strategies are being investigated. In this context bioinspired strategies such as Evolutionary Computation are becoming attractive alternatives to traditional design, since it can naturally deal with decentralized distributed solutions, and are more robust to noise and the uncertainty of real world applications (Thakoor et al., 2004).
Evolutionary robotics is a promising methodology to automatically design robot control circuits (Nelson et al., 2004a). It is been applied to the design of single robot navigation circuits with some success, where it is able to achieve efficient solutions for simple tasks,

Source: Frontiers in Evolutionary Robotics, Book edited by: Hitoshi Iba, ISBN 978-3-902613-19-6, pp. 596, April 2008, I-Tech Education and Publishing, Vienna, Austria

such as collision avoidance or foraging. Recently, evolutionary computation has being employed to look for solutions in multi-robot systems. In such systems, every robot can be treated as an individual that competes with the others to become the best solution to a given task (Liu et al., 2004). In doing so, the robot will have more chances to be selected to combine its parameters to produce new solutions that inherit its characteristics (i.e., spreading its genes and producing offspring, in biological terms).
Multi-robot evolutionary systems present many new challenges to robot designers, but have the advantage of a great degree of parallelism (Parker & Touzet, 2000). Therefore, the produced solutions that have to be tested one by one in a single robot system can be evaluated in parallel by every individual of the multi-robot system (Nelson et al., 2004b). In doing so, the addition of new robots to the system usually results in an increase in the performance of the evolutionary strategy, for more possibilities in the search space can be tested in parallel (Bekey, 2005).
Even though multi-robot evolutionary systems can test more solutions in the same time, the overall performance does not necessarily improve (Baldassarre et al., 2003). This is due to new factors intrinsic to multi-robot systems, such as robot-to-robot interaction. This may produce so much stochastic noise from the interactions of real physical systems that it may be impossible to the evolutionary strategy to distinguish among good solutions, which is the best one. In that context, the best solutions can suffer from the interaction with poorly trained individuals and receive lower scores, diminishing their chances to be selected to mate and spread good genes (Terrile et al., 2005).
When evolutionary systems are built in simulation, it is normally possible to exhaustively test most of the possible situations that the environment can present to an individual solution, resulting in a fitness score that better represents "how good a solution is" (Michel, 2004). With a real environment, it is very time consuming to evaluate a robot, which has to move around and react to different environment configurations. Usually, the faster the generation time, the poorest the evaluation will be. And for longer generations in the real world, the overall delay of the experiment will become prohibitive.
Additionally, the implementation of a fully embedded distributed evolutionary system often means that evolution is forced to deal with small robot populations, due to the high cost of robotic platforms (Parker, 2003). In that case, evolutionary algorithms that where designed to work in simulation with hundreds of individuals will eventually have to be redesigned to cope with these new challenges. Therefore evolutionary functions like crossover, mutation and selection will also have to be reconsidered. In such context, this work intends to present a series of experiments that investigates the effects of evolving small real robot populations, proposing novel evolutionary strategies that are able to work in such noisy environments.

2. The Implemented Evolutionary System

This session presents the strategies chosen to implement the individual controller of each robot and the evolutionary system that controls the robot team. It also shows an overview of the complete system and an introduction to the robot architecture. Although the strategies described can be applied, in theory, to control any number of robots, in this work the global idea was adapted to control a group of six robots. Even though the suggested system was proven to work with such a small population, a larger population of robots would give greater diversity to evolution, improving the performance of the system (Ficici. & Pollack,

2000). Thus, more individuals provide more genetic combinations and increase the chances of finding a good solution to the problem.
The goal of the implemented evolutionary system is to automatically train a team of six autonomous mobile robots to interact with an unforeseen environment in real time. The system is also able to continuously refine the generated solution during the whole working life of the robots, coping with modifications of the environment or in the robots (Burke et al., 2004). Although implemented into a specific group of 2-wheel differential-drive robots for a specific task, the evolutionary system can be adapted to control other kinds of robots performing different tasks. Therefore, this section is intended to be general enough to be used as guidelines to help the conversion of the system to other mobile or static platforms.
To test whether randomly initialised robots could really be trained by evolution to do something practical, a very simple task was chosen: exploration with obstacle avoidance. Such a simple task, that is also known as collision-free navigation, facilitates the implementation of the system and allows its development in relatively low-cost robots. Therefore, more robots could be built and evolution can benefit from more diversity in the population. The main issue considering functional specification in an evolutionary system is to tell evolution *what* the robots have to do, without telling it *how* they are going to achieve that (Mondada & Nolfi, 2001). In our case, the robots are encouraged to explore the environment, going as fast as possible without colliding into the obstacles or each other. Because the workspace contains various robots, the environment also includes some robot-to-robot interference (Seth, 1997) (e.g., collisions between robots and reflection of the infrared signals by approaching robots). The experiments will show how, based on a reward-punishment scheme, evolution can find unique, unexpected solutions for that problem.

2.1 The Embedded Evolutionary Controller

The robot architecture can conceptually be seen as a central control module interfacing all other functional modules, which either supply or demand data required for autonomous processing (see Figure 1). The modules were implemented using a combination of dedicated hardware and software executed by the robot microprocessor. The robot architecture is configured by a set of parameters, a certain number of bits stored in RAM memory. In evolutionary terms, this set of parameters is called the robot chromosome (Baldassarre et al., 2003). The Sensor Module is configured by a subset of the chromosome that indicates the number of sensors used and their position in the robot periphery. The motor drive module is configured by another subset of the chromosome that configures the speed levels of the robot.
The Motor Drive module receives and translates commands from the central control module and controls the direction of travel and speed of the two robot motors. The proximity of obstacles is obtained by the sensor module that decides which proximity sensors are connected to the central control module according to the parameters stored in the robot chromosome.
The Central Control Module (see Figure 1(b)) is divided into three others: the Evolutionary Control; the Supervisor Algorithm; and the Navigation Control. Connected together via the communication module, the Evolutionary Control circuits of all robots control the complete evolutionary process. They process the data stored in the chromosome and send the configuration parameters to the Navigation Control and the other modules. The

evolutionary control systems of all robots use communication to combine and form a global decentralised evolutionary system (Liu et al., 2004). This global system controls the evolution of the robot population from generation to generation. It is responsible for selecting the fittest robots (the best-adapted to interact with the environment), mating them with the others by exchanging and crossing over their chromosomes, and finally reconfiguring the robots with the resultant data (the offspring) (Tomassini, 1995).

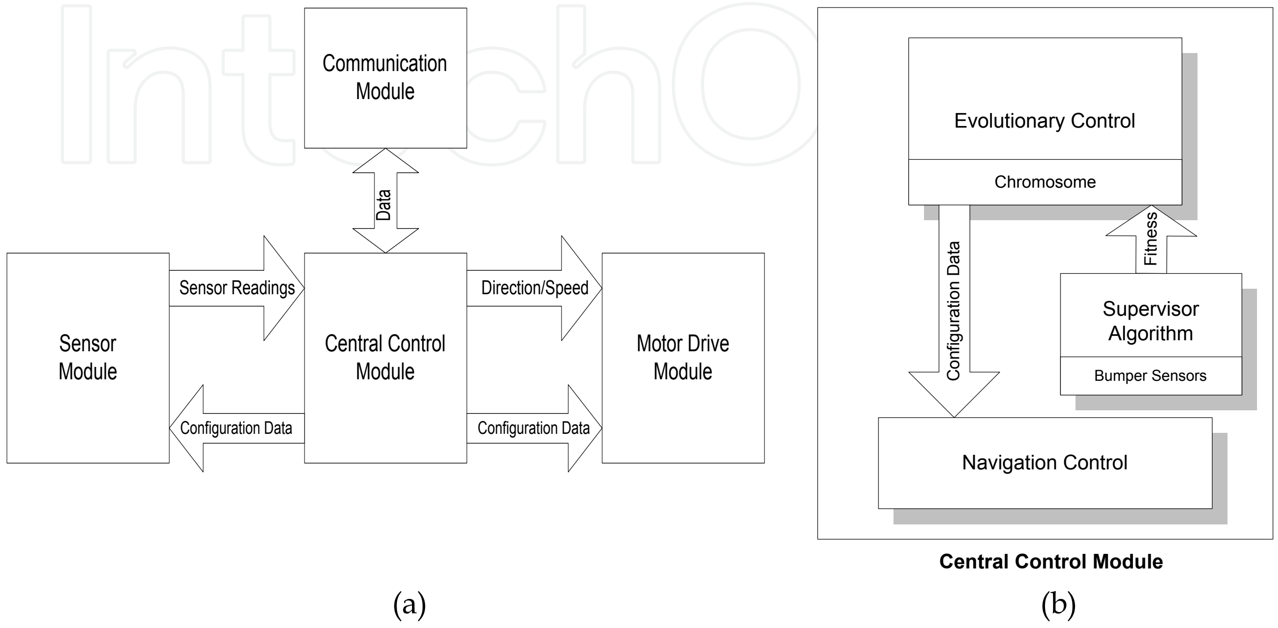

Figure 1. Architecture of the robot control system (a): the Sensor Module and Motor Drive Module are configured by the Central Control Module (b), which processes data from the sensors and commands the motor drive module in how to drive the robot

The robot performance is monitored by the Supervisor Algorithm, which informs the evolutionary control how well-adapted it is to the environment. According to events and tasks performed by the robot, perceived internally by special sensors, a score or fitness value is calculated and used by the global evolutionary system to select the best-adapted individuals to breed. The supervisor algorithm is responsible for activating a rescue routine, a built-in behaviour that is able to manoeuvre automatically the robot away from a dangerous situation once it is detected by the sensors. Contact sensors in the bumpers determine the occurrence and position of a collision. When activated, the rescue routine will take control of the robot until it is safely recovered. It can communicate directly to the motor drive module, by-passing the navigation control. When the rescue manoeuvre is completed, the supervisor algorithm allows the motor drive module to accept once more the commands of the navigation control circuit and the robot resumes on its way.
It is the Navigation Control, configured by the evolutionary control, that commands the motor drive module according to the information provided by the sensor module. It processes the information of the sensors and decides what the robot has to do. Then, it sends a command to the motor drive module, which will control the speed of the motors to make the robot manoeuvre accordingly. The navigation control is the centre of the autonomous navigation of the robot. Configured by the parameters stored in the chromosome, it drives the robot independently. Evolution is responsible for adjusting these parameters so that the robot performs well in the environment.

2.2 The Navigation Control Circuit

A RAM neural network (Ludermir et al., 1999) was chosen to implement the navigation control circuit basically because they have unique features that facilitate their evolution by the system, simplify the implementation in the robot hardware, and allow small modifications to be carried out with minimum effort. They provide a robust architecture, with good stability to mutation and crossover. Most neural networks, like the chosen one, present redundancy between the genotype and the phenotype (Shipman et al., 2000). In other words, a small change in the bits of the chromosome (the genotype) will not produce a radical change in the behaviour of the network (the phenotype). Therefore, the selected neural network is stable enough to allow evolution to gradually refine the configuration parameters of the navigation control circuit, seeking a better performance. Its good neutrality makes it suited to be evolved by the system since a small mutation on a fit individual should, on the average, produce an individual of approximately the same fitness.

The RAM model does not have weighted connections between neuron nodes, and works with binary inputs and outputs. The neuron functions are stored in look-up tables that can be implemented in software or using Random Access Memories (RAMs). The learning phase consists of directly changing the neuron contents in the look-up tables, instead of adjusting the weights between nodes.

In relation to robot implementations, the RAM model, or RAM node is very attractive, since it provides great flexibility, modularity, parallel implementation, and high speed of learning, what leads to less complex architectures that can easily be implemented with simple commercial circuits. The RAM node is a random access memory addressed by its inputs. The connectivity of the neuron (N), or the number of inputs, defines the size of the memory: 2^N. The inputs are binary signals that compose the N-bit vector of the address that can access only one of the memory contents.

The RAM neural network simplicity and its implementation as elementary logic functions are responsible for its fast performance. The mapping of the RAM neural network into simple ALU logic functions and their direct execution in the microprocessor ALU can reduce even more the total memory required by the control algorithm. The faster speed provided by these simple implementations allows a faster controller, which can improve the decision rate in low-cost microprocessors.

Figure 2 shows the sensor module processing the information of the sensors and feeding the neural network inside the navigation control circuit. The output of the neural network is a command that tells the motor drive module how to control the motors. The evolutionary control reads the information contained in the chromosome and sends the parameters to configure the sensor and motor drive modules. It also reads the contents of the neurons from the chromosome and transfers them to the neural network in the navigation control circuit (Korenek & Sekanina, 2005). The motor drive module intercepts the command and activates the corresponding routine that generates the signals for the motors.

Figure 3 shows more details on how the navigation control circuit interfaces the sensor and motor drive modules. The neurons are connected in groups (discriminators) that correspond to one of the possible classes of commands (*C1*, *C2*, … *Cn*) the neural network can choose. The groups are connected to an Output Adder (*O1*, *O2*, … *On*) that counts the number of active neurons in the group. The Winner-takes-all block receives these counting from the output adders, chooses the group with more active neurons, and sends the corresponding

Command to the motor drive module. The sensor module converts the analogue readings of the infrared proximity sensors into 2-bit signals that can be connected to the neuron inputs.
In the selected approach, the inputs of the RAM neurons are connected to the sensor outputs provided by the sensor module. All neurons of the network have the same number of inputs, although that number may vary according to the application (Ludermir et al., 1999). In the implemented network, all neurons in the same position in the groups are connected to the same inputs (i.e., the first neuron of the first group will have the same inputs as the first neuron of the second group and so on…).

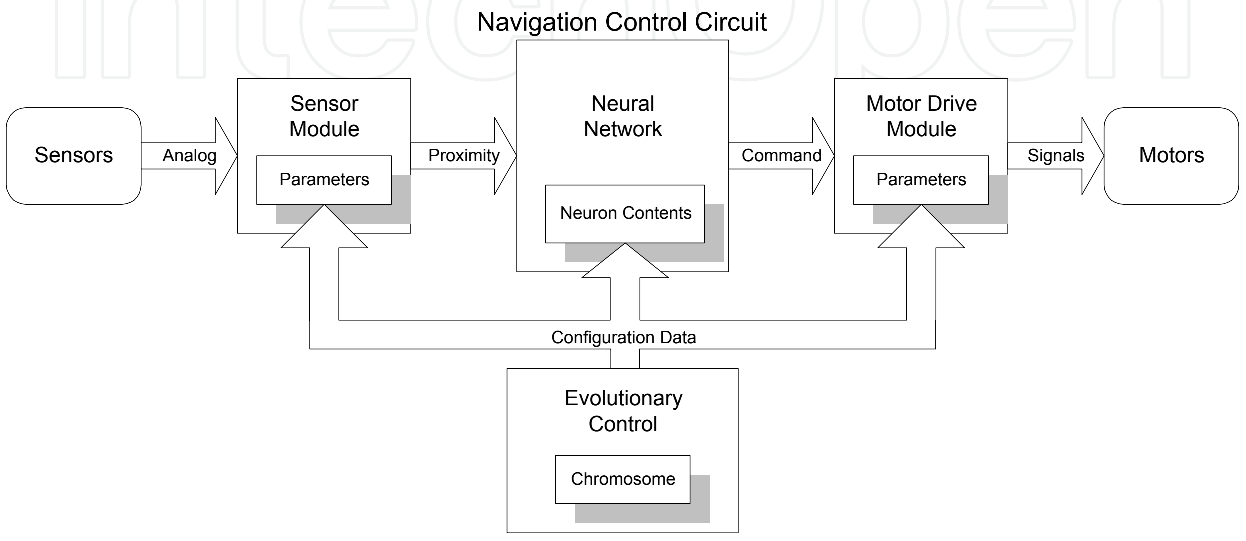

Figure 2. The navigation control circuit interfacing the sensor and motor drive modules, and the evolutionary control

The connectivity between the input lines and the sensor outputs is controlled by a Connectivity Matrix that defines which sensor outputs are connected to *Li*, *Lj*, and *Lk*. The connectivity matrix is randomly initialised at the beginning of a new evolutionary experiment. One other advantage of RAM neural networks is their modularity. This characteristic simplifies the modification of the architecture. The number of neuron inputs can be modified by rearranging the connectivity to the sensors alone. Sensors can also be added or removed in this way. New commands are easily included by inserting more neuron groups.
The RAM neural network can be evolved by simply storing sequentially the neuron contents into the robot chromosome and allowing the evolutionary algorithm to manipulate these bits. Basically, the neural network must have enough inputs to cover all the sensors, although some of the sensors may be connected to more than one input line. To avoid saturation, enough neurons must be placed in the groups so that the network can learn all the different input configurations that correspond to the correct output commands. If the network is having difficulty learning a different situation, more neurons should be added. Different architectures were implemented and simulated in software until the developed solution was obtained. Figure 4 shows an example of neural network that works with four commands to control the motor drive module: *Front Fast* (*FF*); *Turn Left Short1* (*TLS1*); *Turn Right Short1* (*TRS1*); and *Turn Right Short2* (*TRS2*).

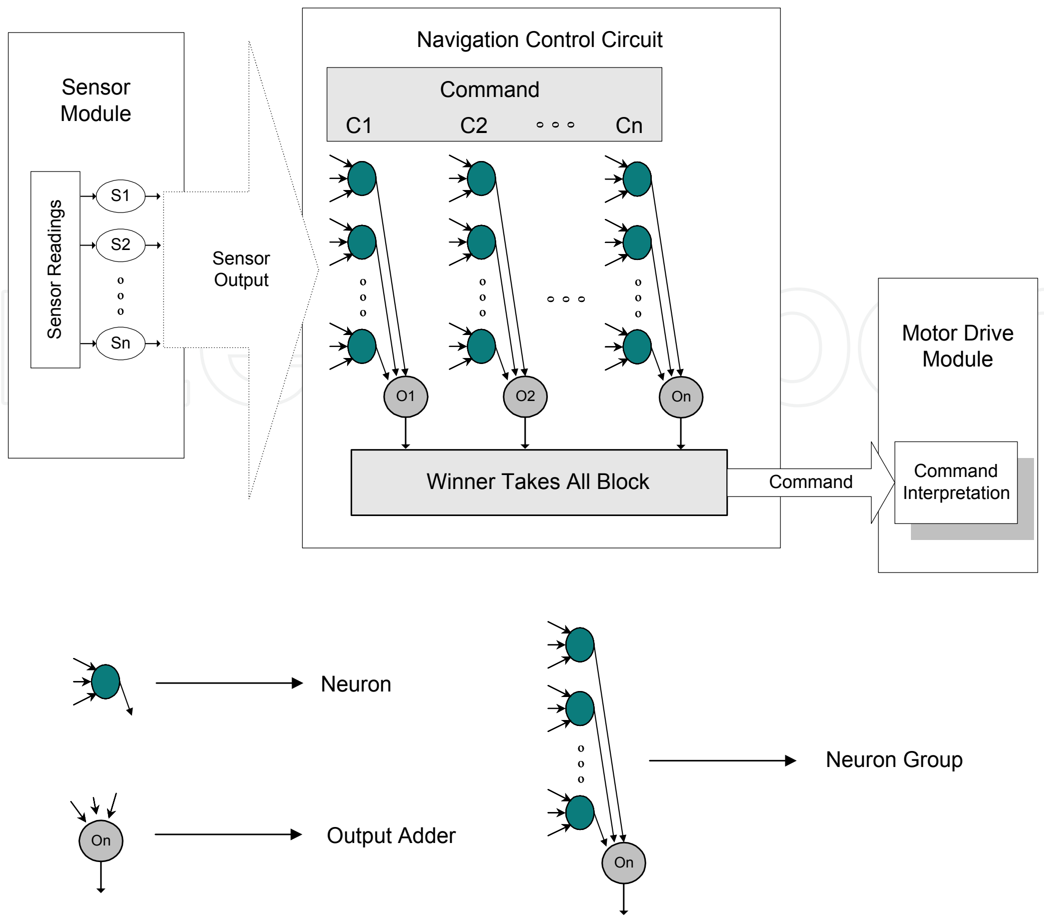

Figure 3. The neural network in the navigation control circuit. *S1* to *Sn* are the binary sensor readings. The Output Adders (*O1* to *On*) count the number of active neurons in the group. *C1* to *Cn* are the classes of commands to the motor drive module

2.3 The Evolutionary Control System

It is the evolutionary control system, located inside the central control module of the robots (see Figure 1), that performs the evolutionary processes of evaluation, selection, and reproduction (Tomassini, 1995). All robots are linked by radio, forming a decentralised evolutionary system. The evolutionary algorithm is distributed among and embedded within the robot population. Figure 5 exemplifies a cyclic evolutionary process where the individuals are evaluated according to their capacity to perform the tasks in the environment. If they perform well, it can be said that they are well-adapted to the environment. The robots are assigned a score, or fitness value, that tells how fit they are. When the evaluation period is over, the individuals select a partner to mate with according to their fitness value. The best individuals have more chance of being selected to breed. Next, they exchange their chromosomes, crossing over their genes to form the new combinations. The resultant chromosomes are then used to reconfigure the old individuals, originating new ones, or the offspring. Then, a new evaluation phase starts again. Assuming that new robots cannot really be created spontaneously, the offspring must be implemented by reconfiguring selected old individuals.

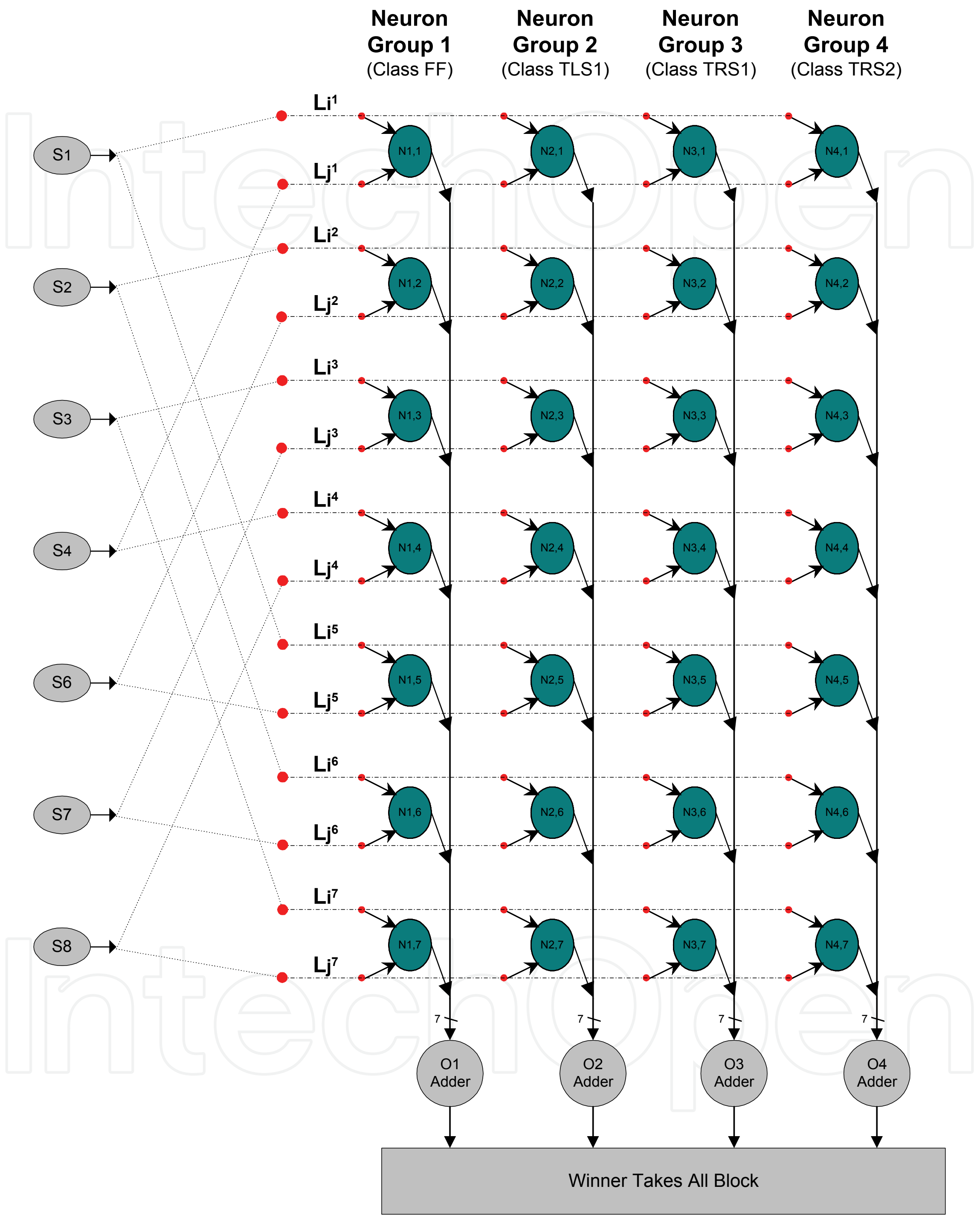
Neuron Group 1 (Class FF)
Neuron Group 2 (Class TLS1)
Neuron Group 3 (Class TRS1)
Neuron Group 4 (Class TRS2)
S1
S2
S3
S4
S6
S7
S8
Li¹ Lj¹ Li² Lj² Li³ Lj³ Li⁴ Lj⁴ Li⁵ Lj⁵ Li⁶ Lj⁶ Li⁷ Lj⁷
N1,1 N1,2 N1,3 N1,4 N1,5 N1,6 N1,7
N2,1 N2,2 N2,3 N2,4 N2,5 N2,6 N2,7
N3,1 N3,2 N3,3 N3,4 N3,5 N3,6 N3,7
N4,1 N4,2 N4,3 N4,4 N4,5 N4,6 N4,7
7
O1 Adder
O2 Adder
O3 Adder
O4 Adder
Winner Takes All Block

Figure 4. Configuration of the neural net with four groups of seven neurons

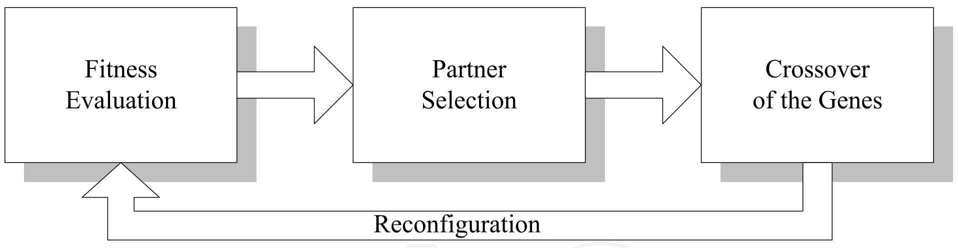

Figure 5. An evolutionary process of evaluation, selection, and reproduction (or crossover)

An evolutionary process, in the context of this work, is the procedure necessary for the development of suitable controllers for the population of robots. The process can stop when the average fitness value of the population reaches a specified threshold or continue indefinitely while the robots execute a certain task. In the developed evolutionary system, the robots work in a cyclic procedure, differently from a traditional design technique, where the controller is designed or trained at first and then transferred to the robot that is put to work. This cyclic procedure is inspired by the natural world where animals, like some birds for example, have a working or foraging season and a mating season, where they concentrate their attention in finding a mate and reproducing (Tomassini, 1995).

The cyclic procedure of the robots, a *generation* in evolutionary computation terms, is exemplified in Figure 5. The robots do not pursue reproductive activities concurrently with their task behaviour. Instead, they perform a working season, where they execute the selected task in the environment (or working domain) and are evaluated according to their performance. The internal timer of Robot 1 indicates the beginning of the mating season. It is important to observe that the evolutionary scheme is decentralised and distributed amongst all six robots. Robot 1 is by no means dominant in this process. The internal timer of Robot 1 is just used to signal the others, indicating the beginning of the mating season. It was necessary to avoid synchronisation problems, since it was impossible to guarantee that all robots would begin the mating season at the same time. In the mating season, the robots communicate to let the others know their fitness value. They start emitting a "mating call", where they "shout" their identification, their fitness values, and chromosomes. The best robots survive to the next generation, breeding to become the "parents" of the new individuals. The less well-adapted robots recombine their chromosomes with the better-adapted ones, reconfiguring their parameters as a new robot before starting a new generation.

The robot recurring procedure for one generation, shown in Figure 5, works according to the following algorithm:

Working Season:

1. Avoid obstacles;
2. Count collisions;
3. Internal timer of Robot 1 indicates the beginning of the mating season;

Mating Season:

1. Robot 1 orders all robots to stop;
2. Robot 1 sends a mating call via the radio (containing its identification, fitness value, and chromosome);
3. Robot 2 then sends its mating call and so do all other robots, one after the other, until the last one;

4. All robots listen for mating calls, receiving every fitness value, comparing with the others, and then selecting the partner to mate with (if own fitness is the highest, the robot does not breed);
5. When all genes are received and partners chosen, start Crossover;
6. Begin reconfiguration with the resultant chromosome and wait until…
7. Robot 1 announces the end of the mating season and orders all robots to start another cycle.

The process begins with the random initialisation of the robot chromosomes. Then, the first generation starts with all robots performing their tasks in a working season. For the case of obstacle avoidance, they will navigate and have their fitness value calculated according to a function similar to the one presented in Figure 6. Robot 1 has control over the duration of the working season and uses the radio to stop the other robots when its internal timer reaches the end of the working season or the "lifetime" of the robots. That is important to synchronise the cycle and make sure that all robots will stop working at the same time. Starting with Robot 1, each robot transmits, one by one, a mating call via the radio, containing its identification, fitness value, and chromosome. When they are not transmitting, the robots listen for other mating calls, receiving the fitness value from the call, comparing it with the others, and then selecting the optimal partner with which to mate. If own fitness is the highest, the robot does not breed and "survives" to the next generation. The cycle will be completed when all robots find a partner to mate with and combine their genes in the crossover phase. The mating season lasts until all the six robots signal Robot 1 that they have found a partner, have mated, and have reconfigured themselves with the resultant chromosome. Robot 1 then orders them to restart another cycle (once more Robot 1 is used only to synchronise the next phase). In other words, the best-adapted robots "survive" to the next generation, while the others "die" after mating, to lend their bodies to their offspring.

2.4 Fitness Evaluation

A simple obstacle avoidance task was chosen. The limited complexity of this task allows a good evaluation of the fitness in a short period. A complex task requires more time so that the robots can be subjected to more challenges in order to show that they can perform well in more than specific situations. A smaller generation time means a faster evolution, because more combinations of solutions will be tried (Pollack et al., 2000).

A reward-punishment scheme is applied during the fitness evaluation process, executed by the supervisor algorithm. Each robot is evaluated during the "*working season*", where its fitness function is calculated by penalising collisions and lack of movement (reducing the fitness value), encouraging the exploration of the environment (rewarding by increasing the fitness value for every second of movement). A major issue that must be addressed is how to detect a good (fit) robot. This question may be highly complex in nature, but in the context of evolutionary systems, it can be simply defined by the programmer, in accordance with the particular problem at hand. Furthermore, writing a fitness function depends on the targeted behaviour and the characteristics of the robot, and the necessary insights are gained through incremental augmentation over many trials in the environment.

For the obstacle-avoidance problem, a simple rule can be applied: a robot will increase its fitness each time it comes across an obstacle and successfully avoids it. Each time it collides, the fitness will be decreased. Figure 6 shows an example of a fitness function where the robot fitness is increased by one for every second the robot is in movement, encouraging exploration. It is punished by decreasing its fitness by ten when it collides. The fitness is also decreased by 100 to punish the robot for turning for more than five seconds. Therefore, this sub-function prevents a particular efficient solution that kept the robot spinning in a small circle within an obstacle-free area.

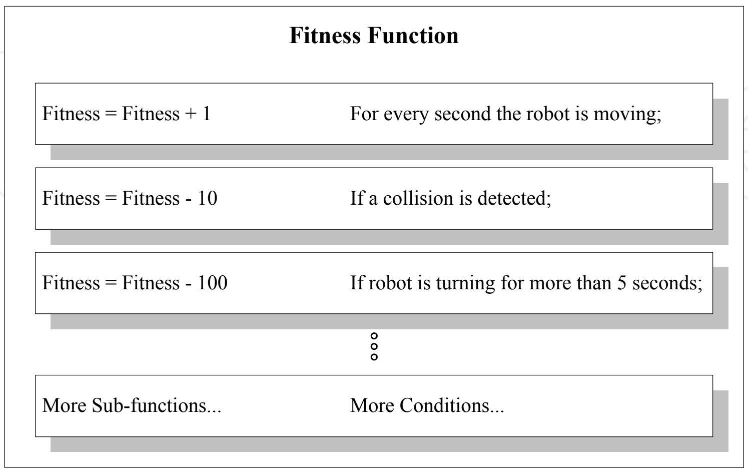

Figure 6. An example of how a fitness function can be constructed

In a situation where an obstacle is close to the robot, but the proximity sensor readings are not interpreted correctly by the navigation control or are not enabled by the sensor module, a collision may occur and the fitness variable will be decreased. The bumper sensors will then be analysed by the supervisor algorithm to calculate where the collision took place in a total of 12 sectors with 30 degrees each. Once the place of collision is detected, a rescue routine will drive the robot away from the obstacle, returning the navigation control to the neural network. When the robot is moving forward without colliding with obstacles, its fitness will be increased every second.

2.5 Partner Selection

The procedure for partner selection is based on the robot fitness value or score. They can be very simple, like choosing the best robot to mate with all other ones, or more complex, such as the roulette-wheel technique (Tomassini, 1995). In this work, the simple approaches are preferred because of the restrictions of an embedded controller (i.e., low processor speed and small memory size). As the population is very small, a simple technique can deal with the robot selection without problems. Therefore, some simple, but efficient selection techniques were developed. These techniques are:

1. Select the robot with the highest fitness value in the generation to breed with all other robots and survive to the next generation. This tries to make sure that in the next generation the best fitness will be at least similar to the present one.
2. An "*Inheritance*" scheme was developed: the score used to select the robot is the average of the robot fitness in the last five generations (i.e., inheriting the scores of its previous generations). The robot with the best average survives, but only breeds with the robots with the fitness in the present generation lower than its own fitness. This approach protects new robots that are actually better than the one with the highest average, but need more generations to be selected by their average.

3. Another very simple strategy that was effective in the experiments is to select the fittest robot, allow it to survive, and reconfigure all the others with a small variation (mutation) of its chromosome. This is a form of "asexual reproduction", where the robots do not cross over their chromosomes. All robots in the next generation will be a copy of the best one, but will suffer random changes (mutation) in a few genes.

All techniques suggested above are *elitist*. Elitism requires that the current fittest member (or members) of the population is never deleted and survives to the next generation (Tomassini, 1995). The developed inheritance scheme prevents a robot from being deleted even if it is not the fittest, but has the biggest accumulated average fitness value. It is the only selection technique where the fittest member of the population does not have the same number of offspring (or the same probability to have offspring) whether it is far better than the rest, or only slightly better. In the other ones, it will always have the same probability to have offspring; and in most of them, will breed with all other robots. This approach is often too severe in restricting exploration by the less fit robots.

A common problem with these techniques is the possible appearance of a super fit individual that can get many copies and rapidly come to dominate the population, causing premature convergence to a local optima (Mondada & Nolfi, 2001). This can be avoided by suitably scaling the evaluation function, or by choosing a selection method that does not allocate trials proportionally to fitness, such as tournament selection. This work, however, does not experiment with these methods.

2.6 Reproduction Strategy

The crossover is the phase in the evolutionary algorithm where the chromosomes of both parents are combined to produce the offspring (Tomassini, 1995). Many techniques are proposed in the literature to implement the crossover phase. Nevertheless, this work uses a very simple strategy, because of the restricted resources of the embedded controller.

In the developed evolutionary system, both morphological features and the controller circuit are evolved to respond to changes in the environment. The robots constantly adapt to changes in the surroundings by modifying their features and the contents of the RAM neural controller. The term "morphology" is defined as the physical, embodied characteristics of the robot, such as its mechanics and sensor organisation. In the performed experiments, the morphological features modified by evolution are the number and position of sensors, as well as the speed levels of the drive motors. Therefore, the genetic material specifies the configuration of the robot control device and morphological features. Eight pairs of genes in the chromosome (*B1*, *B2* to *B15*, *B16*) are used to configure the sensor module; ten genes (*B17* to *B26*) configure the motor drive module; and the remaining genes (*B27* to *Bn*) configure the navigation control module (neuron size × number of neurons for a RAM neural network). The control device is implemented within the robot microprocessor (a neural network for navigation control) and two programmable modules control the robot features, which are the sensor module and the motor drive module (refer to Figure 1).

For the selection of the robot features controlled by the sensor module, a more complex "dominance approach" was implemented to combine the eight pair of genes. Each sensor in the sensor module is configured by two genes in the chromosome: *i)* two genes will determine the presence of a feature ("enable the sensor"); *ii)* one gene comes from each parent; and *iii)* all features are recessive. The two genes are coded using bits in such a way that the combinations "1,1", "0,1", and "1,0" disable the sensor, and "0,0" enables it.

The motor drive module has ten bits associated with it in the chromosome. The features selected by the module are the three different speed levels for the motors. Figure 7 shows that the three speed levels, *Fast*, *Medium*, and *Slow* are controlled by these ten bits. The contents of these ten bits (B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, B_{22}, B_{23}, B_{24}, B_{25}, and B_{26}) are added together to form a decimal number that indicates the speed level *Fast*. The result is a number between zero and ten that indicates the level of the *Fast* speed. The contents of the first six bits (B_{17}, B_{18}, B_{19}, B_{20}, B_{21}, and B_{22}) are added together to form a decimal number that indicates the speed level *Medium*. The result is a number between zero and six. In the same way, the contents of the first three bits (B_{17}, B_{18}, and B_{19}) are added together to form the decimal number that indicates the speed level *Slow*. The result is a number between zero and three. Therefore, as they use the same bits, this guarantees that the level *Fast* is always greater than or equal to *Medium*, which is always greater than or equal to *Slow*.

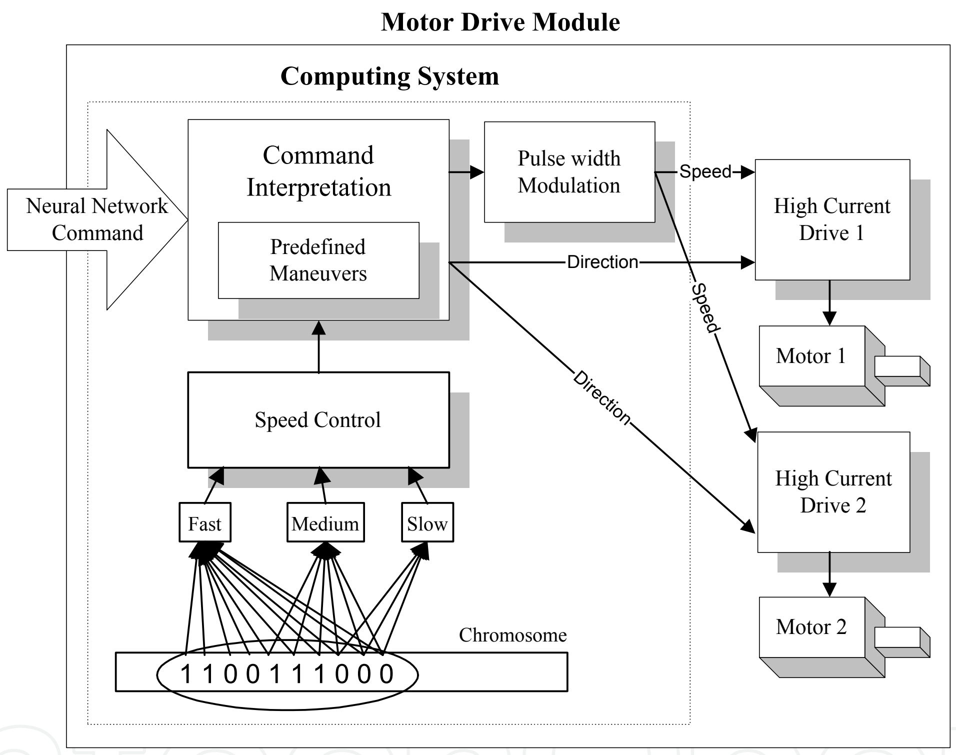

Figure 7. The motor drive module. The speed levels *Fast*, *Medium*, and *Slow* are specified by the chromosome. The command interpretation block has a set of predefined routines to drive both motors. The microprocessor implements most of this module, including the pulse-width modulation signals that control the motors

The resultant speed for the three levels is converted by the motor drive module to a value (an internal parameter) between 1 and 32, because the robot motor can have 32 speed levels. This conversion is not linear, because of the way the motors are controlled by pulse modulation, and the corresponding values are shown in Table 1. This approach permits co-adaptation where the chromosome integrates specifications for both controller and morphological features. Evolution can select not only the number of sensors to use, but, if the number of sensors is fixed, it can select which ones to pick (i.e., the sensor position on the robot).

Speed Level:	0	1	2	3	4	5	6	7	8	9	10
Internal parameter:	1	7	9	11	14	17	20	23	26	29	32
Velocity (m/s):	0	0.02	0.05	0.08	0.1	0.13	0.15	0.17	0.2	0.23	0.26

Table 1. Conversion of the speed levels of the robot

For the genes that control the neural network and the motor drive module, a random exchange of the genes from the parents is used to form the resultant chromosome. This strategy is called *uniform crossover* (Tomassini, 1995), although here only one offspring is produced. Therefore, a gene is selected from the father or the mother to occupy the corresponding position in the offspring chromosome. Thus, a random exchange of genes from both parents occurs after a dominance selection of the 16 bits that enable the sensors.
After the crossover is completed, a mutation phase starts. Applying mutation to a chromosome means that a small number of copying errors may occur when copying the genes from the parent chromosomes to the offspring. In this work, a mutation rate of *M*% means that each gene in the chromosome has a probability of *M*% of being selected and binary inverted (e.g., new gene = *NOT*(gene)). Small mutation rates, usually between 1 and 3%, are the ones that produced the best results in the experiments. Higher mutation is only useful in the beginning of the evolutionary experiment. A high mutation rate does not help to evolve faster, and did not prove to be a good strategy.

3. System Overview

The mobile autonomous robots form a decentralised embedded evolutionary system where no host computer is required. Nevertheless, an IBM PC is used to monitor all data exchanged via the radio link, producing a complete record of the chromosomes, parameters, and variables for every generation. The robots can communicate with each other and the monitor computer via an asynchronous serial data link using their communication module. The computer monitors the robot internal variables without interfering in the system, but has the capacity to start or stop an evolutionary experiment.

Figure 8. The monitor computer connected to the radio

A radio board is connected to the monitor computer, which logs data on the evolutionary experiment. It is a multi-channel driver/receiver interfacing the IBM PC via its serial port. A software interface permits the downloading of software, data, and commands between the robots and the computer. Programming and low-level debugging are provided by the computer. When programming or debugging, bi-directional data between the robot processor and the computer can operate either via a wired link or via radio. Figure 8 shows the monitor computer connected to the radio board that is used to communicate with the robots and monitor the evolutionary experiment.

The experiments were performed within a 2.50m × 2.50m working domain, containing walls and obstacles of varied sizes, where the robots can explore the environment, avoiding collisions. Many movable obstacles and internal walls of different sizes are available to change the scope of the workspace where the robots navigate. The workspace can be modified into different configurations by rearranging the obstacles and walls. This flexibility is necessary to allow different degrees of complexity of the environment during the experiments. Figure 9 shows different configurations of the workspace, representing a simple (a) and a complex (b) environment.

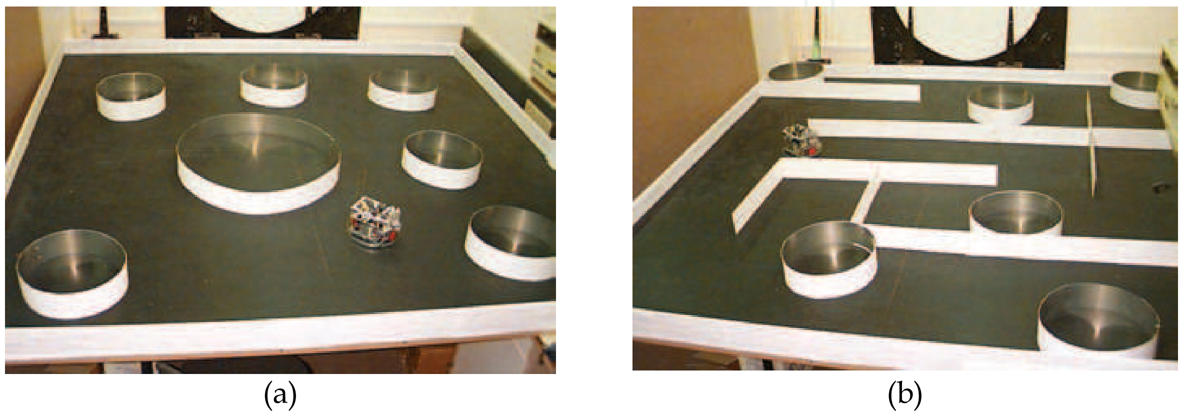
Figure 9. An example on how simple (a) and complex (b) environments can be produced by rearranging the obstacles and walls

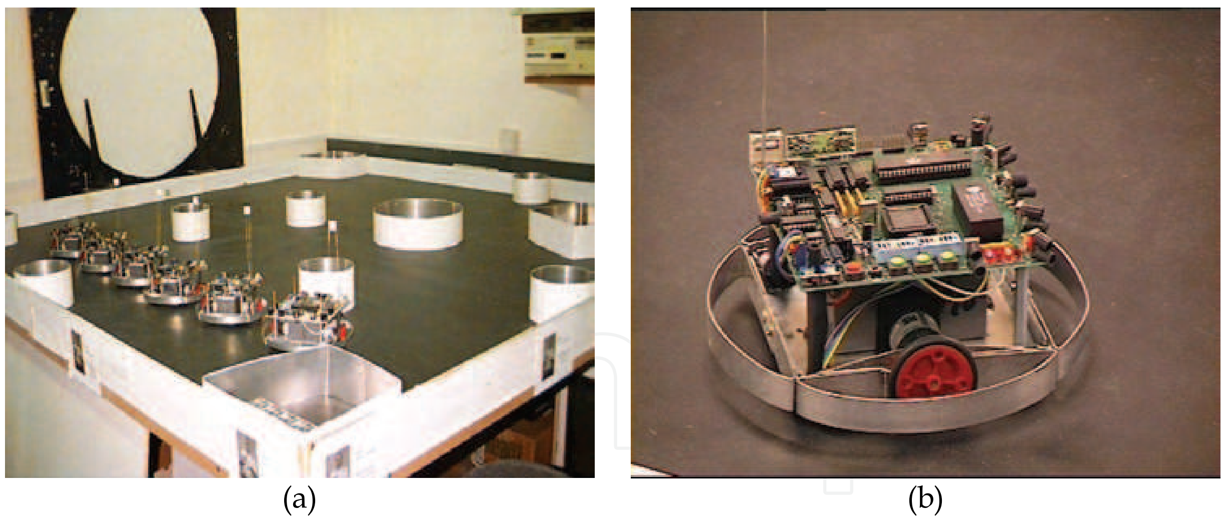
Figure 10. A view of the robot team (a) and top-view of Robot 2 (b), showing the position of the infrared sensors and bumpers

The robot architecture consists of a two-wheel differential-drive platform (20cm diameter), containing a Motorola 68HC11 - 2MHz, 64Kb of RAM, bumpers with eight collision sensors, and eight peripheral active infrared proximity sensors. It exchanges information with the other robots at 1.2Kbps by a 418MHz AM radio. Both robots and workspace were specially built for the experiments. All eight proximity sensors are connected to the sensor module, which is configured by the chromosome. The module can individually enable or disable the

sensors, changing the number of active sensors and, consequently, their position in each generation. Therefore, the physical, embodied characteristics of the robot can be modified. The wheels are placed in the middle of the robot, allowing it to turn around its central axis. The robot has four round bumpers on its front, back, left, and right, which are attached to the base by eight contact sensors that permit the supervisor algorithm to pinpoint the location of a collision.
Figure 10 (a) shows the team of robots parading in their workspace. Figure 10 (b) shows a top-view of Robot 2, displaying its infrared proximity sensors and round bumpers. The three green keys on the right form a small keyboard used to enter commands to the robots manually.

4. The Experiments

4.1 Experiment 1: Evolving with a Simple Fitness Function

In this experiment, the embedded evolutionary system attempts to evolve the control circuits of the group of six robots. The eight sensors were permanently enabled and the speed of the motors was fixed at maximum speed. Therefore, the navigation control circuit was the only one under evolutionary control. All sensors were set to operate at *medium* range, detecting obstacles closer than 15cm. The sensor positions around the robot are presented in Figure 11.

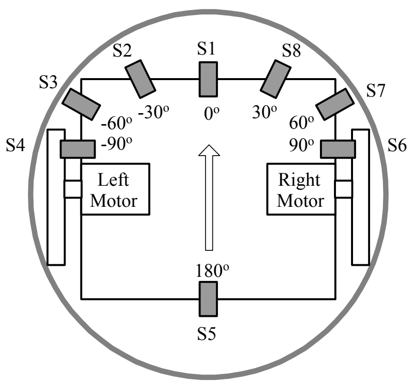

Figure 11. Position of the sensors on the robot and their angle in relation to the central line of the robot

The neural network can take seven commands to control the motor drive module: *Front Fast* (*FF*); *Turn Left Short1* (*TLS1*); *Turn Right Short1* (*TRS1*); *Turn Right Short2* (*TRS2*); *Turn Left Short2* (*TLS2*); *Turns Right Long* (*TRL*); and *Turn Left Long* (*TLL*). *Front Fast* means "move forward with maximum speed". To turn left/right short, the robot moves with reverse direction in one of its motors (with both motors at maximum speed), causing a spin around its own axis. The difference between *TRS1* and *TRS2* is that in the later, the robot keeps turning for 200ms, while the duration of the other three commands is just one iteration (10ms more or less). In *TRL* and *TLL,* the robot turns right/left by breaking one wheel and turning around it with the other one in maximum speed in a wide arch of one wheel span of diameter. The speed of both motors was fixed at the maximum setting, enabling the robots the move at 0.26m/s. As the selection of the sensors and the speed levels were fixed, the only feature that varied was the controller configuration.

Using only seven output commands means that the neural network needs only three bits to encode them and seven classes of discriminators to work. The neural network architecture shown in Figure 4 is preserved and three new groups of seven neurons where added too represent the new commands. Therefore, the architecture has now seven groups of 7 neurons (m=7 and n=7) with two inputs each (the neuron size is four bits). The interconnections between sensors and neurons are randomly chosen. Each neuron has four bits of memory to store its contents. The neural network has 49 neurons with a total memory size of 196 bits. This would also be the size of the corresponding bit string in the chromosome, if the neural network was to be evolved. The winner-takes-all block chooses the command that has more active neurons and encodes it with three bits, before sending it to the motor drive module. Therefore, if the size of the genotype of the robots is 196 bits, then the current search space is considerably large: $2^{196} = 1.004\times10^{59}$.

Aim of the Experiment:

This experiment aimed to test if the embedded evolutionary controller was able to evolve the navigation control circuit until the collision-free navigation behaviour emerges, while having the robot morphology (the sensor configuration and motor speeds) fixed. To achieve this, the sensor configuration was fixed with all sensors enabled and the velocity of both motors was fixed at maximum speed. The navigation control circuit is configured by only 196 bytes in the robot RAM memory and evolution is allowed to manipulate any bit of these bytes. The robots were allowed to reproduce and suffer mutation, but only with the bits in the chromosome that correspond to the navigation control circuit.

Experimental Setting:

The population of robots with different configurations was evaluated in 60-second generations. The robots were allowed to reproduce and mutate with mutation rate equal to 3%. The chosen selection strategy for this evolutionary experiment was:

- *Select the robot with the highest fitness value in the generation to breed with all other robots and survive to the next generation.*

This strategy means that the robot with the highest fitness is selected, and will send its chromosome to the other five robots. Each one of the remaining five robots then combines its own chromosome with the one received from the best robot to produce a resultant chromosome. Then, the remaining five robots reconfigure themselves with the mutated chromosomes and the robots continue in the next generation.

In the crossover phase, where the two chromosomes are combined to form a resultant offspring, each bit is randomly chosen from the corresponding location in the chromosome of the parents. Then, in the mutation phase, a random number r is generated between zero and 100 for each bit in the chromosome, and the bit will be flipped each time r is smaller than the mutation rate.

This experiment used a very simple fitness function in an attempt to prevent biasing evolution to any preconceived idea of an ideal controller. The selected fitness function for this test was:

1- Start with 4096 points;

2- Reward: increase fitness by 10 points every 1 second without collision;

3- Punishment: decrease fitness by 30 points for every time command is not FF for more than 15 seconds;

4- Punishment: decrease fitness by 10 points for every collision if command = FF, TLL, or TRL.

Rule 2 constantly rewards the robot with five points for every second it is moving without colliding. Rule 3 was introduced to prevent evolution from producing a solution that keeps turning around itself and never collided. This rule punishes the robots that keep turning for more than 15 seconds, encouraging them to move forward. Rule 4 only punishes the robots that are not turning around themselves, since a collision in that case can only mean that they are being crashed by another robot. If a robot is executing any command but *FF*, *TLL*, or *TRL* its centre is not moving, so the collision cannot be its fault.

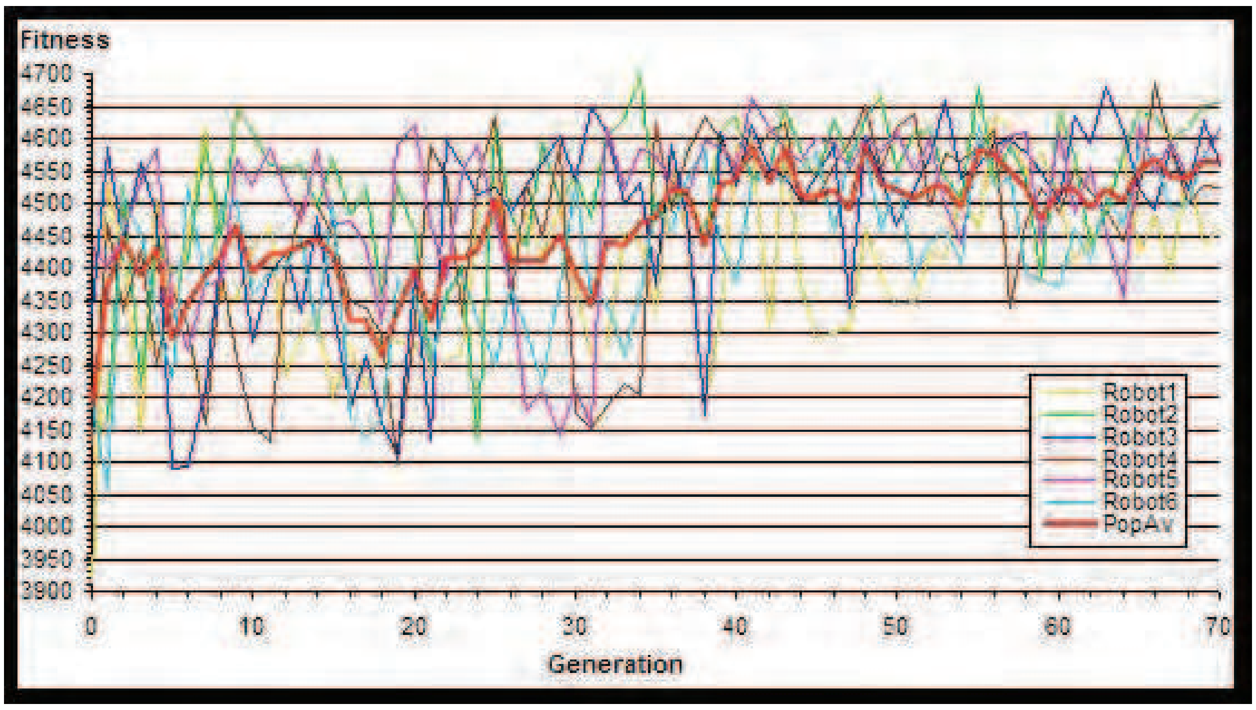

Figure 12. Evolution of the _neural network controller_ using _mutation rate of 3%_, and _generation time of 60s_. The population was randomly initialised. *Robotn* is the fitness of Robot *n* and *PopAv* is the average fitness of all robots in the generation

The experiment in Figure 13 was performed to investigate if the robot population, once initialised with a well-trained controller, could hold this configuration or would degenerate. The robots were initialised with a previously trained neural network, which was hand-designed to avoid obstacles in every situation the robots can face. The population was then allowed to mate and mutate with the same parameters of the previous evolutionary experiment to determine if it would ever get to such a good solution if the experiment was allowed to continue for more generations. The results presented in this figure can be compared to the ones obtained from the evolution of a randomly initialised population in Figure 12.

Discussion:

Figure 12 showed that the evolutionary system succeeded in evolving the population towards the expected behaviour, producing solutions that practically did not collide after 35 generations. It was observed in this experiment that one could not rely only on the fitness of a robot to know how well-adapted it is. A simple solution can be lucky enough to start its lifetime in a safe part of the environment, away from obstacles and other robots, which would produce a very high performance. This was observed in some robots during the evolutionary experiment (e.g., Robot 2 that scored 4644 in generation 10, or Robot 1 that scored 4621 in generation 8). Therefore, human judgement was the best way to evaluate the abilities of the robots and the performance of the evolutionary system.

In the experiment shown in Figure 12, all sensors were enabled from the beginning of the evolutionary experiment. The controller had information from all the sensors and needed to learn what to do with it. Based on observation, some robots learned how to use the frontal sensor, *S1*, which gave them an advantage early on in the process. This was the case for Robot 2 that could avoid obstacles detected by *S1* from generation 8. Robot 2 quickly became the best robot and throughout the crossover operation transferred to Robot 5 the ability of using *S1*. Robots 2 and 5 dominated the population until generation 23, where they were overtaken by Robot 3 and Robot 4. Robot 3 learned how to use *S1*, *S4*, and *S7* from generation 29, and was able to avoid most of the obstacles very well thereafter. From generation 47 on, all robots acquired the necessary skills to employ at least three sensors and the average performance was much better. From generation 50, Robot 2 was well-adapted to the environment, and could use *S1*, *S2*, *S4*, *S6*, and *S7* to avoid collisions with most of the obstacles and other robots.
The major observed problem was the instability of the system, where there was no guarantee that a good solution, such as Robot 2 in generation 50, would be selected as the best robot until it is outperformed by better robots. The population performance dropped when a robot with a poorly-adapted controller (Robot 1) was lucky enough to be selected as the best robot and spread its bad genes through the population. Robot 1 had a poorly-adapted controller that made it move forward each time *S6* detected an obstacle, independently of having something in its way. The average performance dropped, but after a few generations, the fitness of the population recovered.

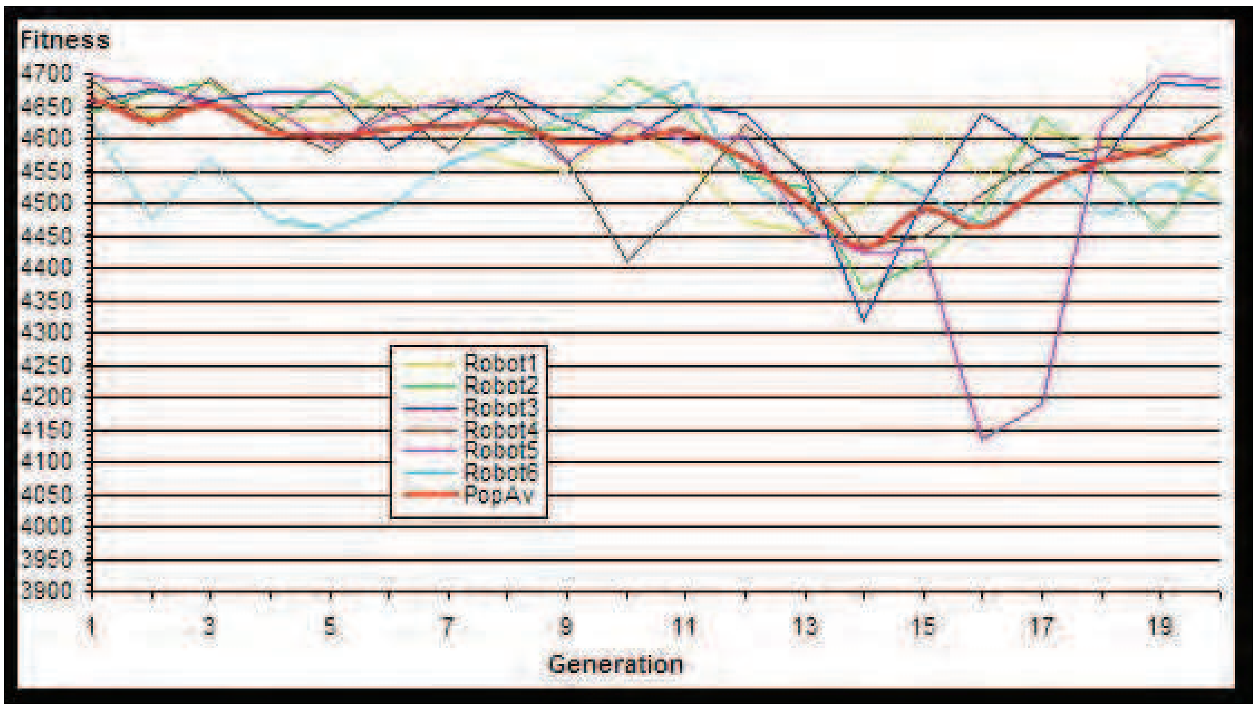

Figure 13. Evolution of the <u>*neural network controller*</u> using <u>mutation rate of 3%</u>, and <u>generation time of 60s</u>. The population was initialised as a hand-designed controller. *Robotn* is the fitness of Robot *n* and *PopAv* is the average fitness of all robots in the generation

The average of the population fitness is a good parameter to determine if the system is converging to the desired behaviour. The population performance oscillated, but kept improving through the evolutionary experiment. With such a large search space (1.004×10^{59}), a perfect robot that can deal with all sensors should take a very long time to

obtain with this evolutionary approach. Nevertheless, the system succeeded in producing an even population of robots that can successfully avoid the obstacles in some of the situations faced and produce a fitness near to the maximum performance (4696).
Figure 13 shows how the system behaves after the population has been initialised with a hand-designed controller. The aim of this test was to show that if such a good solution happen to be produced by evolution, it would be preserved in the process. If the evolutionary experiment cannot keep such a solution, it is unlikely that it will ever be produced by evolution under these circumstances. Unfortunately, the figure showed that the evolutionary system, as set in this experiment, could not keep a good solution for more than ten generations and the performance of the system degenerated. Although it was possible to recover and continue to improve performance after the loss of the hand-designed controller genes, this test indicated that it is unlikely that such a good solution will be produced by the process as it was configured for this experiment. Maybe a more biasing fitness function can produce a better result.

4.2 Experiment 2: Evolving with Inheritance

This experiment developed a different selection strategy for evolutionary systems that was defined as *Inheritance Strategy*. From now on, the robots will be selected to breed based on not only their performance in the current generation, but also on their fitness in the previous generations. Therefore, the robots *inherit* part of the points scored by their predecessors. By considering the average fitness scored by the robot in the previous generations, this strategy attempts to reduce the effect of chance, which can produce bad performances if the robot is "unlucky". This experiment tested this new approach, and evaluated if it could solve the instability problem pointed out by Experiment 1.

Aim of the Experiment:

The aim of this experiment is to develop and test a novel selection strategy involving inherited scores. It evaluated whether this new strategy could solve the problems with instability presented by the previous selection strategy, where a lucky unfit robot can be selected as the best robot instead of better ones that by chance started in a more crowded area of the environment.

Experimental Setting:

Apart from a different selection strategy, this experiment was set exactly as Experiment 1. The chosen selection strategy for this evolutionary experiment was:

- *The score used to select the robot is the average of the robot fitness in the last three generations (i.e., inheriting the scores of its previous two generations). The robot with the best average survives, but only breeds with the robots with the fitness in the present generation lower than its own fitness.*

This approach favours the robots with the highest average, which are the ones that have performed well for the last three generations. If in the current generation one lucky unfit robot happens to achieve the highest score, it is still unlikely that it will have the best average score and will not be selected to mate. This was an attempt to improve the stability of the system and reduce the effect of noise and interactions among robots and obstacles.
The charts display the fitness of the six robots in the current generation as well as the average fitness value (*AvRn*) scored by each robot in the current and the previous two generations. For each robot:

$$AvRn = (FitnessRn_{G0} + FitnessRn_{G-1} + FitnessRn_{G-2})/3$$

In the equation above, n is the number of the robot; $FitnessRn_{G0}$ is the fitness of the Robot n in the current generation; $FitnessRn_{G-1}$ is the fitness scored by Robot n in the previous generation; and $FitnessRn_{G-2}$ is the fitness scored by Robot n two generations before.

Results:

This is a strategy that protects a fit robot in the current generation that scored more than the best robot (the one with the highest average) by not allowing it to mate. If the robot with the best average is allowed to breed with robots with fitness in the present generation lower than its own fitness, novel better solutions will not be selected because their average will be smaller than the one of the robot that won for the last three rounds. This can destroy the precious good genes of this robot and evolution will lose this better performance.

This approach protects new robots that are actually better than the one with the highest average, but need to be evaluated for more generations to be selected by their average. If one of these protected robots has a better configuration, it will probably repeat its good performance, and within one or two generations its average fitness may be the highest and the robot will be chosen to reproduce. If the protected robot is in fact a lucky unfit one, it may not repeat its good performance and, if so, will be allowed to breed in the next generation. Figure 14 shows a test with the updated selection strategy.

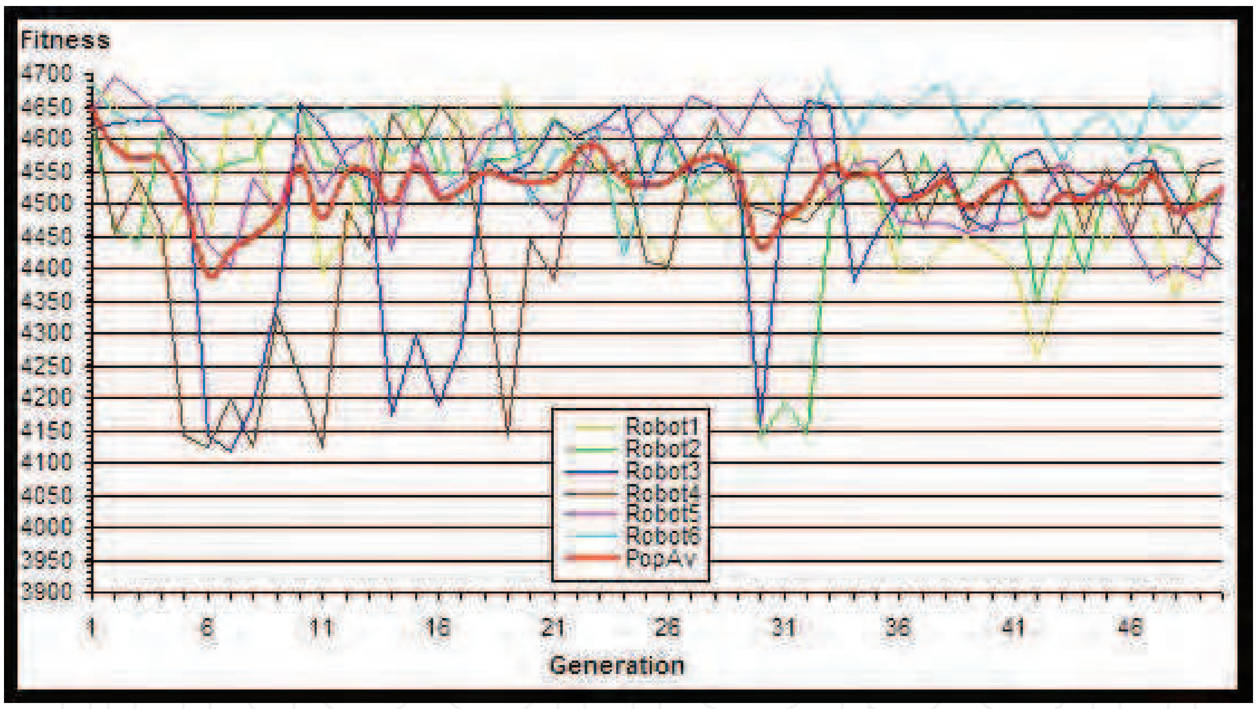

Figure 14. Evolution of the *neural network controller* using inheritance selection, mutation rate of 3%, and generation time of 60s. The population was initialised as a hand-designed controller. *Robotn* is the fitness of Robot *n* and *PopAv* is the average fitness of all robots in the generation

Discussion:

The new selection strategy presented in Figure 14 succeeded in improving system stability. The fitness of the population dispersed mostly because of mutation, but the performance of the best robot did not degrade as much as in the previous attempts. It now protects the potentially better configurations, as happened with some of the robots that outperformed the robot with the best average and were given the chance to repeat their better performance. When they managed to win again, their average became finally higher and they were selected as the best robot of the generation, breeding with all other robots. In the next experiment, presented in Figure 15, this new selection strategy is applied to evolve a randomly initialised population.

Figure 15 shows that even now the evolutionary system could not prevent the population performance from degrading due to mutation. This figure showed that in 50 generations, the population performance degraded to a level similar to the one where the experiment of Figure 14 stopped, showing that it is unlikely that it will improve the population much more than this level.

This experiment showed that the new inheritance selection helped to improve system performance, but did not completely solve the instability problem. The system does not seem to be able to produce an optimal solution to the collision-free navigation task. The experiments demonstrated that evolution still can take a considerable amount of time to achieve a solution as good as the hand-designed controller and may not even be able to get there.

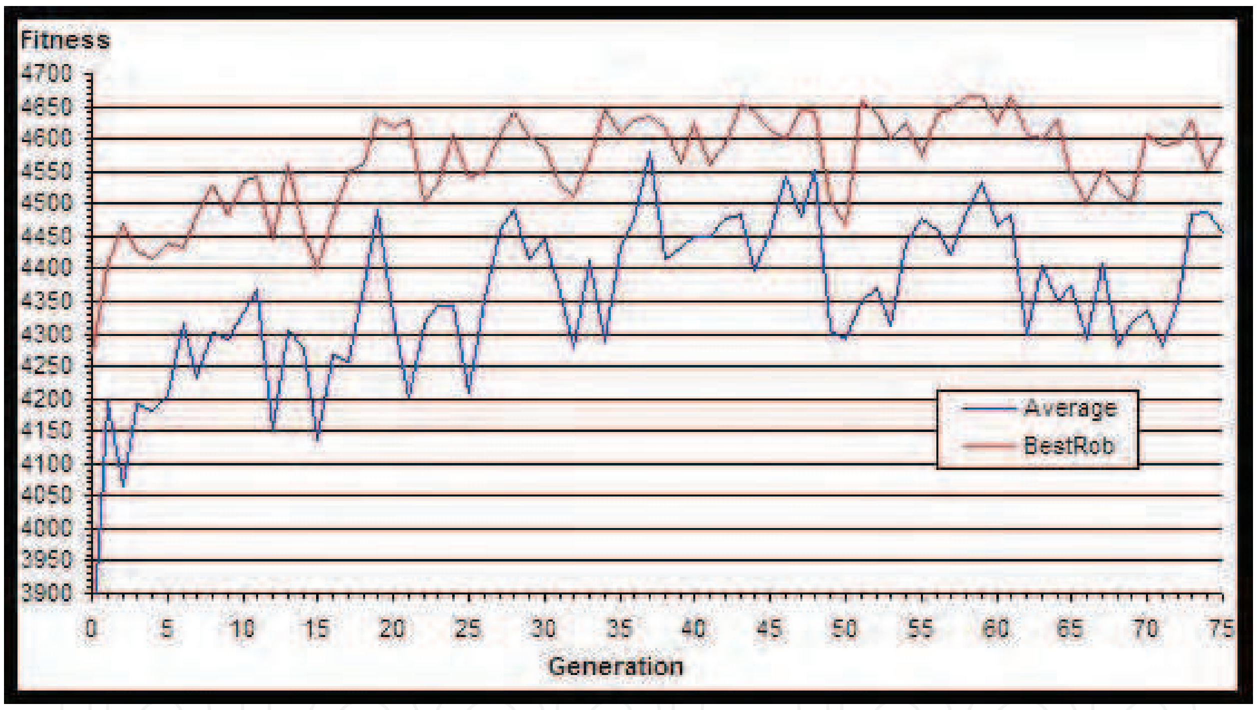

Figure 15. Evolution of the *neural network controller* using inheritance selection, mutation rate of 3%, and generation time of 60s. The population was randomly initialised. *Average* is the average fitness of all robots and *BestRob* is the fitness of the best robot in the generation

4.3 Experiment 3: Disabling Back Mutation

In the previous experiments, mutation was randomly choosing which bits in the chromosome to change. This meant that any bit in the chromosome could mutate to a different value and later mutate back to its original value. In the scope of this work, these events were called *back mutations* (Tomassini, 1995). This fact was causing evolution to waste time trying to find a good solution by testing some configurations that have been tested before. Consider a chromosome that has 99% of its bits set to the correct ones. If any bit can be changed by mutation, the chance of changing the bad ones is only 1%. This means that

99% of the time, this strategy produces configurations that are potentially worse than the current chromosome, and many generations are wasted in evaluating them.
To prevent evolution from wasting time in evaluating configurations that have been tested before, a new strategy that prevents back mutations is developed in this experiment. It consists of marking each bit in the chromosome that suffered mutation and only allowing the bits that are not marked to mutate. To achieve this, a binary array was created in memory with the same size of the chromosome. It is initialised with zeros and when one bit in the chromosome is mutated, "1" is written to the corresponding position in the array. Only the bits in the chromosome that have zeros in the corresponding position in the array are allowed to mutate. Once all bits have mutated, the array will be full of "1s". Then, the evolutionary system resets the array to "0s" and every bit in the chromosome will be allowed to mutate again. This strategy was called *back-mutation prevention*. By preventing back mutations, this strategy forces evolution to evaluate the effect produced by each bit in the chromosome.
Aim of the Experiment:
The aim of this experiment is to apply a new strategy that prevents back mutation of the bits in the chromosome. It tested whether this strategy could be applied in sexual reproduction and provided data to analyse which is the best solution for evolving the robots in the long term.
Experimental Setting:
This experiment used the same settings of Experiment 2 where the evolutionary system is working with inheritance. It examined the new strategy of back-mutation prevention in the robot population.
Results:

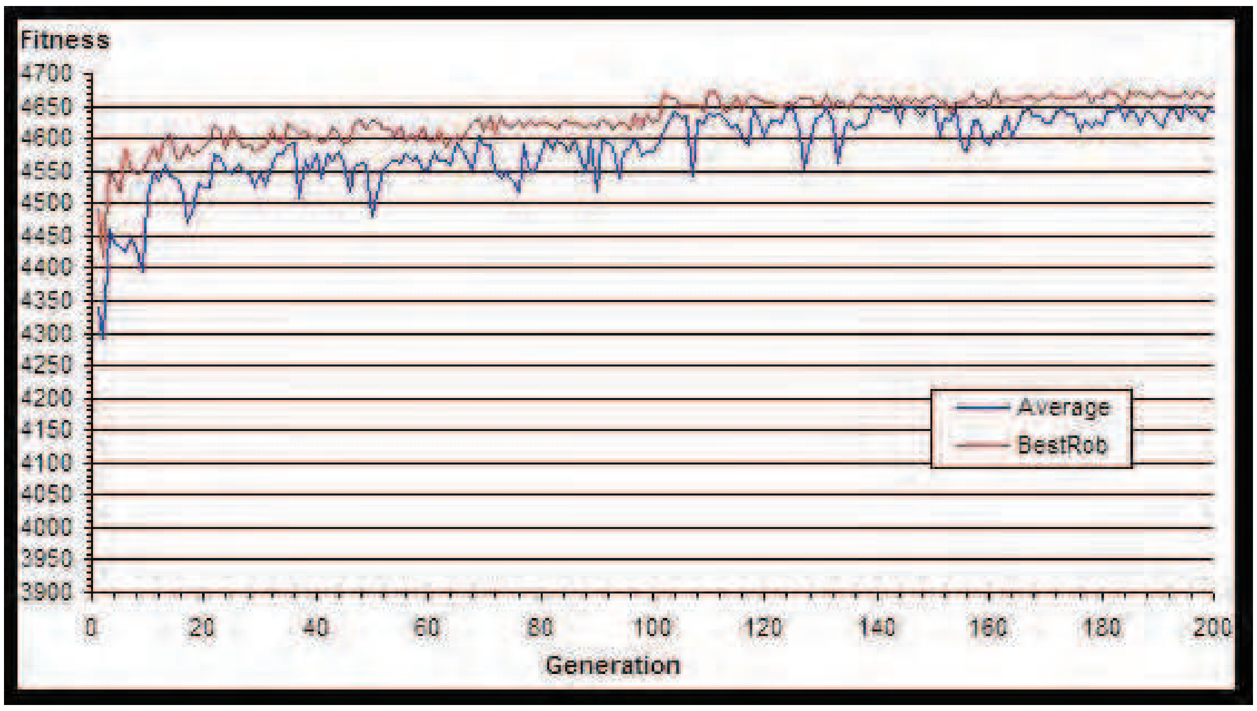

Figure 16. Evolution of the *neural network controller* using back mutation prevention, inheritance selection, mutation rate of 3%, and generation time of 60s. The population was randomly initialised. *BestRob* is the fitness of the best robot of the current generation and *Average* is the average fitness of all robots in the generation

Figure 16 shows a small increase in performance in comparison to experiment 2. The techniques developed so far succeeded in allowing the employment of the evolutionary system to control such a small population of robots.
Preventing back mutation was efficient in speeding up the process of finding a solution. Evolution was able to get close to maximum performance in less than 120 generations and was able to maintain a good result in the population thereafter. These results allow the evolution of a real system in 2 hours. The combined strategies of preventing back mutations and inheritance made viable the use of evolution to autonomously design robot controllers.

5. Conclusion

This work intended to provide understanding on the implementation of real evolutionary systems and inspiring insights that have potential of application in real time robotic control. When implementing an evolutionary system with real robots, the stochastic noise arising from the interactions of real physical systems makes very difficult to distinguish between global and local optima. The same robot can get different fitness values if evaluated again, even with the same configuration, due to noise on infrared signals, dust on the environment floor, etc. But the most important factor was the interaction between the robots. In this contest, good robots can get low fitness values if they are unlucky enough to spend most of the generation time navigating amongst poorly trained robots.
The suggested inheritance selection is a powerful strategy to prevent much of the chance factor from affecting the system. It succeeded in preventing most of the unfit individuals being mistaken as the best robot and protected the better robots from being erased by reproduction, giving them a chance to survive and repeat their better performance, until their average performance overcame the one of the current best robot.
The strategy of back-mutation prevention tested in the experiments was very efficient in looking for a solution in a considerably large search space. It demonstrated the power of the developed evolutionary system for many other applications of genetic algorithms, not only in robotics. The performance of the system was improved since the search space was reduced by using a neural network to implement a structured evolving controller.

6. References

Baldassarre, G., Nolfi, S., & Parisi, D. (2003). Evolution of Collective Behavior in a Team of Physically Linked Robots, *Proceedings of the Second European Workshop on Evolutionary Robotics*, pp. 581-592, ISBN 3-540-00976-0, Essex, UK, April 14-16, 2003, Springer Verlag, Berlin

Barker, w & Tyrrell. M. (2005). Hardware Fault-Tolerance Within the POEtic System, *Proceedeings of the 6th International Conference on Evolvable Systems*. LNCS 3637, pp. 25-36, ISBN: 3-540-28736-1, Barcelona, Spain, September, 2005, Springer-Verlag

Bekey, G. A. (2005). Autonomous Robots. From Biological Inspiration to Implementation and Control. MIT Press. ISBN: 0-262-02578-7

Burke, E., Gustafson, S., & Kendall, G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 1, 2004, pp. 47–62, ISSN: 1089-778X

Ficici, S. G. & Pollack, J. B. (2000) Effects of Finite Populations on Evolutionary Stable Strategies. *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, pp. 927-934, ISBN: 1-55860-708-0, Las Vegas, Nevada, USA, Morgan-Kaufmann, 2000.

Korenek, J. & Sekanina, L. (2005). Intrinsic evolution of sorting networks: a novel complete hardware implementation for FPGAs, *Proceedeings of the 6th International Conference on Evolvable Systems*. LNCS 3637, pp. 46-55, ISBN: 3-540-28736-1, Barcelona, Spain, September, 2005, Springer-Verlag

Liu, S., Tian, Y., & Liu, J. (2004). Multi robot path planning based on genetic algorithm. *Proceedings of 5th World Congress on Intelligent Control and Automation*, pp. 4706–4709, ISBN: 0-7803-8273-0, Hangzhou, China, June 2004

Ludermir, T., Carvalho, A., Brage, A., & Souto, M. (1999). Weightless Neural Models: a Review of Current and Past Works. *Neural Computing Surveys*, v. 2, pp. 41-61, ISBN:1093-7609

Michel, O. (2004). Webot: Professional mobile robot simulation. *Journal of Advanced Robotic Systems*, Vol. 1, No. 1, (March 2004), pp. 39–42, ISSN 1729-8806

Mondada, F., D., & Nolfi, S. (2001). Co-Evolution and Ontogenetic Change in Competing Robots. *In Advances in the Evolutionary Synthesis of Intelligent Agents*, Publisher: MIT Press, Cambridge, MA, USA, ISBN: 0-262-16201-6, 30p., 2001.

Nelson, A. L., Grant, E., Galeotti, J., & Rhody, S. (2004a). Maze exploration behaviors using an integrated evolutionary robotics environment. *Robotics and Autonomous Systems*, Vol. 46, N. 3, pp. 159–173, ISSN : 0921-8890

Nelson, A. L., Grant, E., & Henderson, T. C. (2004b). Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous Systems*, Vol. 46, N. 3, pp. 135–150, ISSN : 0921-8890

Parker, C. A., Zhang, H., & Kube, C. R. (2003). Blind Bulldozing: Multiple Robot Nest Construction. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2010-2015, ISBN-10: 0780378601, Las Vegas, Nevada, October 27-31, 2003, IEEE Press

Parker L. & Touzet. C. (2000). Multi-robot learning in a cooperative observation task. *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems, DARS 2000,* pp. 391—402, ISBN 4-431-70295-4, Knoxville, Tennessee, USA, October 4-6, 2000, Springer,.

Pollack, J. B., Lipson, H., Ficici, S. G., Funes, P., Hornby, G. S., & Watson, R. A. (2000). Evolutionary Techniques in Physical Robotics. *Proceedings of the Third International Conference on Evolvable Systems - ICES 2000: From Biology to Hardware (Lecture Notes in Computer Science; V. 1801),* pp. 175-186, ISBN-10: 3540673385, April 17-19, 2000, Edinburgh, UK, Springer

Seth, A. K. (1997). Interaction, Uncertainty, and the Evolution of Complexity. *Proceedings of the Fourth European Conference on Artificial Life*, Husbands, pp. 521-530, ISBN: 0262581574, Brighton UK, July 1997, P. and Harvey, I. (Eds.), MIT Press, 1997.

Shipman, R., Shackleton, M., Ebner, M., & Watson, R. A. (2000). Neutral Search Spaces for Artificial Evolution: A Lesson From Life. *Proceedings of the Seventh International Workshop on the Synthesis and Simulation of Living Systems - Artificial Life VII*, pp. 52-59, ISBN-10: 026252290X, Reed College, Portland, Oregon, USA, Aug. 1-2, 2000

Terrile, R. J., et al. (2005) Evolutionary Computation technologies for space systems. *Proceedings of the IEEE Aerospace Conference*, pp. 4284- 4295, ISBN: 0-7803-8870-4, Big Sky, March 2005.

Thakoor, S., Morookian, J. M., Chahl, J., Hine, B., & Zormetzer, S. (2004). Bees: Exploring mars with bioinspired technologies. *Computer*, Vol. 37, No. 9, (Sept. 2004) pp. 38–47, ISSN: 0018-9162

Tomassini, M. (1995). A Survey of Genetic Algorithms. *In Annual Reviews of Computational Physics*, Vol. III, 87-118, World Scientific, 1995, ISBN-10: 9810221762