A SELF-EVOLVING CONTROLLER FOR A PHYSICAL ROBOT: A NEW INTRODUCED AVOIDING ALGORITHM

Dan Marius Dobrea
Adriana Sirbu
Monica Claudia Dobrea
Faculty of Electronics, Telecommunications and Information Technologies
"Gheorghe Asachi" Technical University
Bd. Carol I, no. 11, Iaşi, România, 700506
E-mail: mdobrea@etti.tuiasi.ro

KEYWORDS
Genetic algorithms, controller, robot, avoidance algorithm, autonomous behavior

ABSTRACT

One of the major problems when building robots capable to operate in real-world environments is the ability of the robots to deal with a continuous stream of unexpected real-time events. In this paper, we present a real reactive robot able to avoid obstacles. The obstacle avoidance behavior is tested in real environments and it is mainly based on the movement knowledge database extracted previously in a simulation-based learning process. Exactly, a new obstacle-avoiding genetic algorithm implemented in a configurable simulation environment for differential wheeled robots is used to obtain the database of movements for the real robot. The extracted knowledge database comprises the main set of rules that directly maps the sensor information into the engine commands. In real environments, however, for those singular cases in which none of the simulation-based extracted rules applies (i.e. this is the case of the situations never seen before), new rules solving properly the imminent collisions are obtained online and with the same GA algorithm.

INTRODUCTION

In the last 20 years, due to the advances of the chipset industry that mainly boosted the processors market (e.g. higher working frequencies, new architectures, multiples cores etc.) the soft computing techniques received a powerful support. The soft computing techniques that deal mainly with imprecise and uncertain data are very well fitted with real environments (which have the same characteristics).

Fuzzy-logic (FL), neural networks (NNs) and genetic algorithms (GA) are all soft computing techniques largely used in real-environment applications such as real-time robot control applications. Genetic algorithms, for example, are extensively used to develop controllers for robots – especially the hybrid ones, like GA-fuzzy or GA-neural controllers. In these hybrid approaches, the GA adjusts the parameters of the robot controller. The different values of these parameters, obtained during an evolutionary process, generate different behaviors of the robot. In the genetic evolution, the robots with the best performances have more chances to spread their characteristics to the offsprings. At the end, after several iterations, a robot with good performances is obtained. There is a large number of successful reported applications in which the GA tunes the fuzzy logic controllers (Tan et al, 2011), (Martíneza et al, 2009). In the designs of the fuzzy controllers devised for obstacle avoidance, other type of algorithms like the ant colony enhances the used GA techniques (Chiou et al, 2010).

In robotics, one of the main applications of the GA algorithms consists in finding the optimal path that should be followed by a robotic system in order to reach a particular goal (Hosseinzadeh and Izadkhah, 2010), (Repoussis et al. 2009). In order to have a reliable navigation algorithm for an autonomous robot, the latter must be able to: (a) localize its current position, (b) execute (independently from a human operator) a local collision-free motion within its environment and (c) find out a global optimal path to its final goal. Out of all these last three objectives our work, presented in this paper, is dedicated only to develop a robot-improved ability to execute a local collision-free motion trajectory within real environments. For this, we take advantage of a simulation environment for robotic systems in which a genuine GA is used to extract the core movement knowledge-database. Finally, the effectiveness of the obtained controller-evolved strategies is tested on a real robot. The main concept of the new self-evolving controller is similar to that of the FL approach (e.g. the existence of a set of rules). Unlike the FL approach, in our case the rules are extracted using a GA not using the knowledge of some human experts.

The main application of our GA controller is in the field of the remote controlled robots. In such applications, the GA controller adjusts autonomously the remote command whenever the user command put the robot in the imminent danger of colliding.

OBSTACLE AVOIDANCE

The uncertainty of the real environments is the biggest challenge of each autonomous robot. The autonomous robotic controllers must deal with a large number of factors

like the robotic system mechanical and electrical characteristics and the environment complexity (e.g. the variation of spatial and temporal parameters, the nonlinearities and uncertainty manifested through chaotic and random dynamics of the environment objects etc.). Consequently, each autonomous robot must be able to explore autonomously its environment, recover from failures, solve new avoidance problems and, most of all – all these tasks should be done in real-time.

Our robot learns, adapts and generalizes the knowledge related to both itself and environment with the help of a genetic algorithm and thus becomes able to execute a collision-free motion. The learning and adaptation processes are similar to those in a human being that reacts and learns through experience.

The robot

The robot has three degrees of freedom (3DOF), being able to execute two basic movements: rotation and translation. In the rear part of the robot there are placed two motor-driven wheels that provide locomotion through differential drive mechanism, see Figure 1. The two direct current engines that drive the wheels are, in turn, controlled by a microcontroller system. An unpowered wheel, placed in the frontal-central part of the robot, ensures stability. The robot has an average top speed of 0.3 m/s.

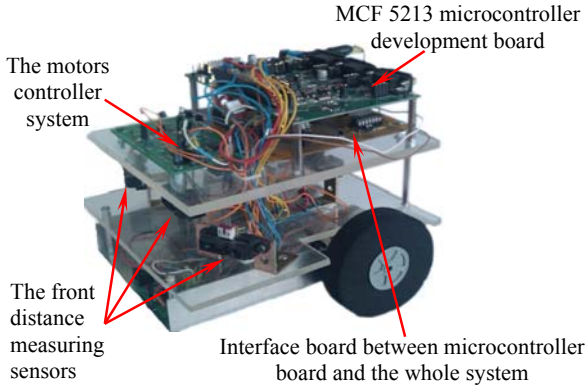

Figure 1: A Picture of the Robot Hardware Implementation

The robotic system has four infrared, IR, proximity sensors – GP2D120XJ00F. The IR sensors have an active distance measuring range from 4 up to 30 cm. Three of the IR sensors are placed in front of the robot, supervising the left, the central and the right part of the frontal environment, and one is placed in the central-rear position of the robot, see Figure 1. The sensors return a voltage value proportional to the distance to the detected obstacles. One of the main IR sensor problems is that the distance characteristic function is a non-linear one. As a result, the sensor characteristic was linearized based on a regression algorithm (Dobrea and Dobrea, 2010a).

The robot sustains the GA auto-organizing controller that is built on a powerful 32-bit MCF5213 microcontroller. The microcontroller belongs to ColdFire™ family and it is based on a RISC architecture comprising a large number of peripheral equipments like eight PWM channels, four 32-bit timers with DMA request capability, eight channels ADC, 3 UARTs, 1 CAN etc.

The robot is designed to learn, in an adaptive manner, based on the information supplied by the sensors. The speed of the wheels is updated at 3.3 Hz, through two PWM channels that command two H-bridges. The program is developed in C language, using the CodeWarrier IDE. More, we use the microcontroller interrupt sub-system in order to reduce the computational burden.

Genetic algorithm

Genetic algorithms (GA) are global search and optimization techniques inspired from the natural selection mechanism existing in the nature.

A GA is based on a population of candidate solutions, called chromosomes. Each chromosome is evaluated and ranked by a fitness evaluation function. The fitness function provides information of how good each chromosome is. The evolution of the GA from a generation to the next one involves three steps: fitness evaluation, chromosome selection and building the next generation (mainly based on GA operators like reproduction and mutation). The next generation is created with the only goal of improving the population fitness.

The avoidance algorithm

The avoidance algorithm has a GA as a main engine. In one chromosome we code left and right engine commands, using for this two signed chars variables; thus, c_j = {*left_eng*$_j$, *right_eng*$_j$} for the j^{th} chromosome. As a result, each chromosome is represented on 16 bits. Both chromosome sections, of 8 bits each, representing the left and the right engine commands, can take values only within the interval [–128, 127]. The values for the engine commands have the following significance: a value of **+127** denotes full forward power engine, **–128** represents full back power engine and **0** corresponds to a stop command for the engine.

The values, s_i, of the IR sensors are first acquired, linearized (Dobrea and Dobrea, 2010a) and, finally, normalized in [0, 0.9] range, with 0 denoting no obstacle and 0.9 denoting an imminent collision; more, due to the noise, each sensor value smaller than 0.02 is forced to take a 0 value. The analytic form of the fitness function, $f(\cdot)$, is presented in equation (1). The relation (1) complies with the fundamental paradigm that defines the fitness in the genetic algorithm field: the fitness takes the lowest value (zero in our situation) only when a chromosome successfully solves the problem. In other case, the fitness measures the ability of each chromosome to solve more or less the problem – being in a direct proportional relationship with the vicinity of an obstacle, as it is in our case.

$$f_j\left(c_j[n]\right) = \frac{1}{4}\sum_{i=1}^{4} s_i[n] \qquad (1)$$

After 300 ms of robot movement, based on the c_j chromosome containing the engine commands, the fitness is calculated. A collision-free motion, specified by a chromosome c_j, is characterized by values of 0 for all four sensors (this corresponds to the "no obstacle" case, in the vicinity of the robot); in such a situation the fitness function, $f(c_j[n])$, is also zero.

To obtain – based on a continue interaction of the robot with its environment, and without any human intervention – a set of adaptive movements rules, in a **first main stage** the robot has to go straightforward. When the robot comes close enough to an obstacle (i.e. the obstacle lies in the active range of its sensors), the GA starts to find the best solution in order to avoid the obstacle; this last case corresponds to the **second main stage** of the algorithm. In this last stage, the most important goal of the real-coded genetic algorithm is to find the best chromosome(s) that encapsulates those engine commands that minimize the fitness function. As a result, the GA finds the best way to avoid the obstacle.

The GA works with population of chromosomes. In other researches, each chromosome characterizes a single robot (Messom, 2002). In our case, we have only one robot that operates the GA, which can be considered a major practical limitation. To avoid this limitation, in a first step, the robot will move in one direction based on the engine commands encapsulated in the first chromosome from the population. After 300 ms the robot stops and the fitness value associated with the chromosome is determined. Then, the robot comes back into the initial position and the algorithm proceeds, in the same way, for the others chromosomes.

Table 1: Movement Knowledge Database – Real Recordings

Sensors				Engines	
Left	Center	Right	Back	Left	Right
⋮	⋮	⋮	⋮	⋮	⋮
0.54	0.19	0	0	105	-100
0	0	0.57	0	-109	117
0.49	0.37	0	0	103	-109
0	0	0.61	0	47	85
0.34	0	0	0	93	10
0	0	0.41	0	-104	56
0.43	0.62	0.31	0	-128	-72
0	0	0.40	0.32	75	91
⋮	⋮	⋮	⋮	⋮	⋮

After each generation, the fitness value(s) of the best chromosome(s) from the entire population improves. At the end, the GA obtains the best obstacle-avoiding solution(s) for that obstacle-encountered particular situation. Based on the best-obtained chromosome, the robot moves accordingly and it correctly navigates without any kind of collision. After it avoids successfully the obstacle, the robot switches back to the **first main stage** of the algorithm, moving only in a forward direction. However, this will happen only up to the moment when a new obstacle will come into the sight of the sensors range. In this last case, the robot will switch to the **second main stage** of the algorithm, in which the GA will have to solve the new avoidance problem. Several of these complete stages will compose the so-called **learning phase** of the robot (we will discuss this, in detail, later).

Each time when the GA solves an avoiding task, the obtained solution is saved into a table – the so-named **movement knowledge database**. Each row of this table contains the sensors information preceding the GA execution (four values – a value for each sensor) and the information stored into the best chromosome (two values – a value for each engine command), see Table 1.

In the **avoiding phase**, when an obstacle comes across, a Euclidean distance is computed between the actual sensor values of the robot and the sensor values stored in the robot movement knowledge database. If one of the computed distances is lower than a predefined threshold the associated engine commands are executed. Otherwise, the robot enters into the **learning phase** and a new movement rule is extracted and locally stored in the movement knowledge database. In this way, the self-evolving controller becomes endowed with the capacity to adjust itself to any new previously unseen situation encountered in real environments. More, from the cases presented above one can deduce that the avoiding phase can take place with or without the learning process, this fact being elected by the meeting of the predefined threshold criteria. The **learning phase** is composed of several GA instances. The number of the GA instances is equal with the number of the rules from the **movement knowledge database.**

RESULTS AND DISCUSSIONS

The first implementation of the self-evolving controller

In a first attempt, all the steps previously presented were implemented and the self-evolving controller of the real robot (see Figure 1) was tested. The method we chose for the chromosome selection was the stochastic universal sampling, which exhibits both no bias and a minimal spread (Baker, 1987). The crossover method was two points. Table 2 shows the list of the main parameters of the GA algorithm as they were implemented on the robot.

Table 2: The Main Parameters of the Genetic Algorithm

Parameter	Value
Population Size	10
Generations	20
Generation gap	0.9
Probability of Mutation	0.0437
Crossover Points	2
Crossover Probability	0.7

As a result, we obtained finally a functional prototype. One big disadvantage of the self-evolving GA implemented controller was the significant computation time spent by the robot in the learning phase. In its turn, this large learning time generated other additional problems like the wear and tear of the mechanical parts of the robot as well as the necessity of a large number of rechargeable cycles for the batteries. To reduce this time we made use of some practical observations. For illustration, not every time the GA needed

20 generations to converge. In many situations after fewer generations (in several cases even in the first generation) the fitness function took a zero value. Based on this observation we imposed the GA to stop whenever the fitness on a single chromosome took a zero value. In this mode, the learning time was considerable diminished.

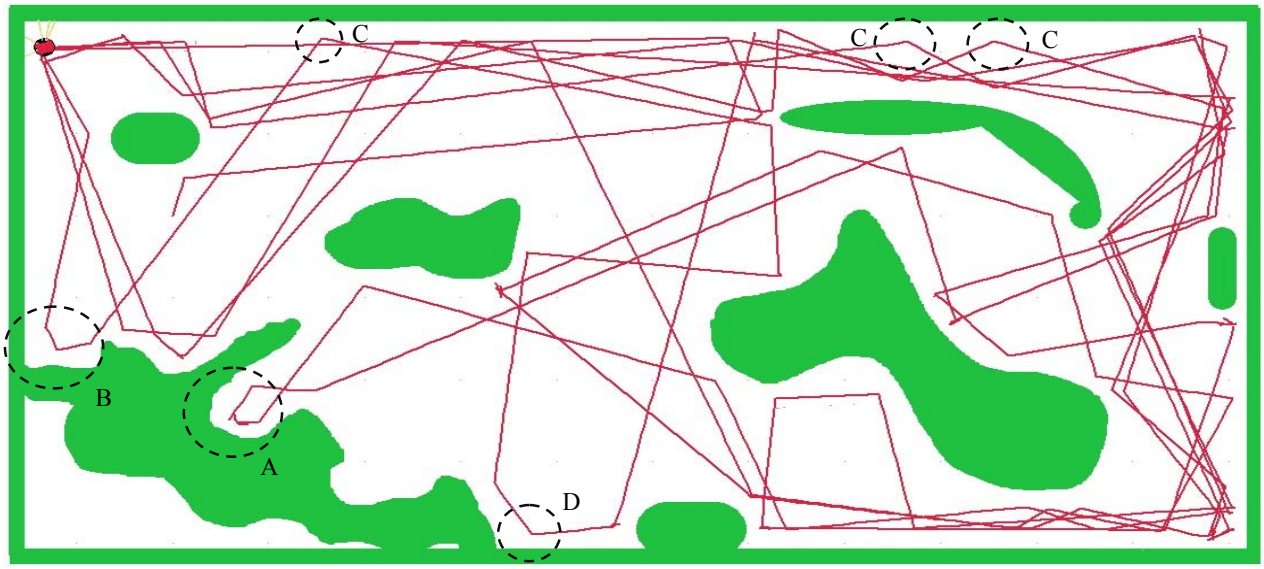

Figure 2: The **Learning Phase** of the Robot

However, even after all the software enhancements the time elapsed in the **learning phase** remained high for our real implemented robotic system. Moreover, a robotic system tested in real environments usually raises some difficulties mainly related to the possibility to make debug on the software modules. These difficulties cumulated with the long time required for each practical testing made our first solution to be quite impractical. Therefore, an alternative approach had to be considered.

In order to overcome all the above-presented disadvantages and thus, to further be able to confirm the main concepts of the new-introduced self-evolving GA controller, we chose the following complementary approach that speeds up the development process of the robotic system: (1) **the (**main part of the) **learning phase** is done in a simulation environment that finally provides the extracted movement knowledge database, and (2) **the avoiding phase** (we mean by this the testing of the controller) is done in a real environment, using for this the previous extracted movement knowledge database.

The simulator

One problem with the new approach used to extract the movement knowledge database resides in the possibility of the self-evolving controller to learn an avoiding behavior that is adjusted to the particular features of the simulated robotic system – features that does not necessary has an identical correspondent in the real world. To avoid this problem we had to model both the robot (its mechanical, electrical and electronically characteristics) and the environment, as realistic as possible. To achieve this goal we chose the MobotSim simulator, version 1.0.03. The MobotSim is a configurable 2D simulator of differential drive mobile robots. The core language for MobotSim is the Sax Basic language – a Visual Basic for Applications™ compatible language. In the MobotSim environment we configured for the robot the following parameters: the platform diameter, the distance between wheels, the wheels diameter, the number of the sensors, the angle between the sensors, the sensor ring radius, the radiation cone, the sensors range and the percentage of misreading. All of these parameters were set accordingly with the real robot mechanical and electronic features.

The second implementation of the self-evolving controller

In the second implementation of the robot we split, as previously presented, the learning phase and the avoiding phase. For the learning phase, we used a simulator environment while the avoiding phase was tested in real environments.

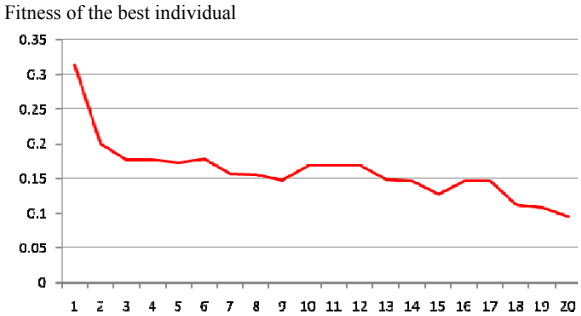

Figure 3: The Evolution of the Fitness for the Best Chromosome in Each Generation – Case D in Figure 2

Figure 2 presents the results obtained in the **learning phase**, on more than 200 GA instances. The time needed to obtain this figure on an Intel(R) Core(TM) i7 CPU at 2.8 GHz, with 8 GB of RAM was 12 hours and 24 minute. Figure 3 presents the evolutions of the GA (i.e. the evolution of the fitness of the best chromosome on each generation) that correspond to the case D displayed in Figure 2. Out of this figure, one can notice the ability of the GA algorithm to improve the avoidance performances and to solve each of the particular indicated obstacle-avoidance tasks.
One of the main parameters of the GA (with an important impact on both the convergence time to a solution and the quality of that solution) is the population size. This is even more critical in our case because when a new, previously unseen situation is encountered by the robot a local GA have to solve the new avoiding task; in this last case and, especially when speaking about real applications – a fast convergence time and a good avoiding solution provided by the GA algorithm proves to be of great value in critical situations. After a large number of tests, a population of 10 chromosomes seemed to comply with the both previously presented constrains.

In Figure 2 one can observe that the GA self-evolving controller is able to solve all the collision situations, like the ones when the robot is placed in corners or even in the hard situations similar with the ones marked with A and B. The ability of the real robot to solve real situations is given by the diversity of the situations encapsulated in the **movement knowledge database**. From this reason, the **learning phase** has to be stopped not when a time value is elapsed or when a number of GA instances is reached. The learning phase will be considered completed only when the simulated robot had been already solved a large number of different types of obstacle avoidance cases. Otherwise, the **movement knowledge database** will comprise only solutions for the most common encountered avoiding situations – as the one marked with C in Figure 2.

The learning environment for the robot consisted in a delimited zone (of around 2 m x 2 m) having one obstacle randomly placed within. This learning zone was build from a number of boxes, of different dimensions, placed as to obtain a close perimeter. From the entire movement knowledge database obtained in the learning process presented in Figure 2, in the real robot only the first 50 instances of the GA were used. The first 50 rules includes A and B avoidance cases, presented in Figure 2.

Based on these rules the robot was able to navigate correctly in the real environment, without any kind of collision. A very interesting characteristic consists in the fact that the robot (with only 50 movement knowledge rules) never triggered the GA in the real environment. More, comparing to other existing approaches, one of the most important features of this robotic self-evolving controller remains its ability to navigate in real environment without any kind of collision which still exists in the learning phase of other reported solutions based on ANNs (Dobrea and Dobrea, 2010a), (Dobrea and Dobrea, 2010b), (Tan et al. 2008).

CONCLUSIONS

This paper reports the development of a new intelligent self-evolving controller for a robotic system able to auto-evolve to a behavior that allows to sense, reason and act – all these in order to avoid all obstacles from a real world environment.

As we mentioned previously, a single evaluation of the entire population, even for a small population of 10 chromosomes, requires significant computation time. From this reason, the main set of movement knowledge database was extracted inside a **simulation environment**. This set of rules supports, on the first stage, the robot dynamics. In the second stage, if an unseen situation occurs, a genetic algorithm, similar with the one implemented in the simulation environment, will solve the problem and the new extracted rule will be used in order to avoid in the future similar situations.

In classical approaches, transferring controllers evolved in simulation environments to physical robots is a very difficult task (Messom, 2002). In our approach, to transfer the **movement knowledge database** obtained inside a simulation environment to the real robot was a success and it represents a new way to solve this problem. The approach presented in this paper has another big advantage: it is very easy to implement it in real robotic systems as it has already been proven.

ACKNOWLEDGMENT

This work was supported by the CNCSIS – UEFISCSU project number PN II – IDEI 1552/2008.

REFERENCES

Tan, S.; Yang, S.X.; and A.M. Zhu. 2011. “A Novel GA-Based Fuzzy Controller for Mobile Robots on Dynamic Environments with Moving Obstacles.”, International Journal of Robotics & Automation, Vol. 26, Issue 2, 212-228

Chiou, J.S.; Wang, C.J.; Wang K.Y.; Hu, Y.C.; Cheng, S.W.; and C.H. Chen. 2010. “Hybrid Algorithm of FLC Design for Robot Soccer.”, International Journal of Nonlinear Sciences and Numerical Simulation, Vol. 11, 119-122

Martíneza, R.; Castilloa, O.; and L.T. Aguilarb. 2009. “Optimization of interval type-2 fuzzy logic controllers for a perturbed autonomous wheeled mobile robot using genetic algorithms.”, Information Sciences, Vol. 179, Issue 13, 2158-2174

Hosseinzadeh, A. and H. Izadkhah. 2010. “Evolutionary Approach for Mobile Robot Path Planning in Complex Environment.”, International Journal of Computer Science Issues, Vol. 7, Issue 4, No 8, 1-9.

Repoussis, P.P.; Tarantilis, C.D.; and G. Ioannou. 2009. “Arc-Guided Evolutionary Algorithm for the Vehicle Routing Problem With Time Windows.”, IEEE Transactions on Evolutionary Computation, Vol. 13, Issue 3 (June), 624 - 647

Dobrea D.M. and M.C. Dobrea. 2010a. “An auto-organization bio-inspired robotic system.”, *In Proceedings of the International Conference on Future Information Technology* (Changsha, Ch, Dec. 14-15). IEEE, Picataway, N.J., 354-358.

Baker J.E., 1987. “Reducing bias and inefficiency in the selection algorithm.”, *In Proceedings of the Second International Conference on Genetic Algorithms* (Cambridge, USA). L. Erlbaum Associates Inc. Hillsdale, NJ, pp. 14-21, July 1987

Tan A.H.; Lu N.; and D. Xiao. 2008. “Integrating Temporal Difference Methods and Self-Organizing Neural Networks for Reinforcement Learning With Delayed Evalutive Feedback.”, IEEE Transaction on Neural Network, Vol. 19, 230-244

Messom C. 2002. “Genetic Algorithms for Auto-tuning Mobile Robot Motion Control.”, Research Letters in the Information and Mathematical Sciences, Vol. 3, 129-134

Dobrea D.M. and M.C. Dobrea. 2010b. “An Autonomous Robotic System”, *In Proceedings of the 9th International Symposium on Electronics and Telecommunications* (Timişoara, Ro, Nov. 11-12) . IEEE, Picataway, N.J., 107-110