WORKSHOP
robocontrol'08
3rd APPLIED ROBOTICS AND
COLLABORATIVE SYSTEMS ENGINEERING
with emphasis in
INDUSTRIAL APPLICATIONS AND EDUCATIONAL ENVIRONMENTS

CNPq

AUTONOMOUS ROBOT NAVIGATION ON A FUZZY MAPPED ENVIRONMENT THROUGH NEURAL NETWORKS, WITH EVOLUTIONARY PATH PLANNING

João A. Fabro, Heitor S. Lopes, Lúcia Valéria R. Arruda

Federal University of Technology Paraná – UTFPR
Av. Sete de Setembro, 3165 – 80230-901 Curitiba (PR), Brazil
fabro@utfpr.edu.br, hslopes@utfpr.edu.br, arruda@cpgei.cefetpr.br

Abstract: This work describes a system for path planning and navigation of autonomous robots. The system uses a fuzzy model of the environment, in which path planning is done by a genetic algorithm, and the navigation is accomplished by means of a reactive subsystem controlled by neural networks. The fuzzy mapping technique proposed here has the advantage of modeling the imprecision of the obstacles, and also gives enough information about the distribution of obstacles so as to improve the genetic search for good paths. The neural controller is responsible for carry out the planned trajectory, and avoid new obstacles that are not present in the map, and cope with interactions with other dynamic obstacles (such as other mobile robots, or people) that crosses its path. The proposed architecture is hybrid, integrating map-based and sensor-based navigation. Some experiments using the Khepera simulator are reported for environments with different complexity. The promising results suggest the continuity of the work.

Keywords: Fuzzy mapping, trajectory planning, genetic algorithms.

1. INTRODUCTION

Autonomous navigation is accomplished by path generation and control of its execution. In general terms the control system must execute a given task, such as reaching a target, while avoiding obstacles. Thus, it is expected that an autonomous navigation system be able to find a path from the starting position to the target position, navigating through this path, avoiding collisions with other objects.

The intrinsic difficulties of the autonomous navigation problem have interested many artificial intelligence researchers who have found it to be of a considerable challenge.

Navigation control of autonomous mobile vehicles is a research area that can be roughly divided into two main approaches: model based path planning, and sensor based navigation. Model based path planning is based on environmental knowledge, and many approaches, ranging from mathematical analysis and path calculations [1], to symbol manipulation on a knowledge base about the environment [2] are available. These methods can solve the path planning problems for environments completely known, with off-line simulations. When facing real-time situations and unknown environments, or with dynamically changing environments, these methods cannot be used. To overcome these difficulties, methods considering real-time environment information from sensors must be considered. Based on sensor readings, the mobile vehicle should be able to perform local path planning and take appropriate control actions. Borenstein and Koren [3] introduced the virtual force field method to solve this problem. However, his method has problems in finding force coefficients in cluttered environments which cannot be described as a mathematical model. Brooks [4] presented a behavioral-based approach, called the subsumption architecture, which is based on pre-specified behavior encoded in task-achieving modules. This architecture has succeeded in navigating in unknown environments, but depends highly on the pre-defined knowledge structures implemented by each module. The success of this approach depends upon how complete the behavior can be described beforehand.

Many other approaches have been developed, mainly using fuzzy sets and neural networks. The fuzzy set approach has the advantage of treating uncertainty and imprecision using simple rule bases [5]. The knowledge must also be provided in the form of fuzzy IF-THEN rules. However, even after rule definition and refinement, it is generally difficult to treat all possible cases with specific rules. To overcome these difficulties, neural networks began to be used. The main advantage of the neural network approach is that there is no need for knowledge programming. For instance, using error back propagation neural networks and a set of training patterns [6], is possible to train a vehicle to navigate in several environments [7]. However, when there are contradictory situations, training is difficult. The system may not be prepared for certain changes in the environmental conditions.

In an attempt to unify the best of path planning, sensor based navigation, fuzzy logic and neural networks, Beom and Cho [8] introduced a control system based on a reinforcement learning scheme to tune a fuzzy rule base, and to obtain adaptive behavior during interaction with the environment. There is no knowledge pre-programming of actions in this approach, and reinforcement training should be performed to tune the control system properly.

Model based and sensor based approaches can be combined in hybrid approaches. In this case, the path planning is based on an incomplete model of the environment, and the execution of the trajectory is achieved by a sensor based controller, which is able to go around unknown (or mobile) obstacles.

In this work, we propose a hybrid model that uses synergically three techniques: genetic algorithms, fuzzy systems and neural networks. To make an imprecise model of the environment, it is proposed the use of fuzzy techniques [9]. The concept of "*fuzzy obstacle*" is introduced to represent the obstacles in the environment. The representation form of these fuzzy obstacles provides information that is used by a genetic algorithm to plan the path. After finding an optimized path, a neural network-based controller manages to move the robot through the planned path, using feed-forward networks [10]. These networks follow the planned path, but are also able to cope with unknown obstacles (static or mobile ones). There are two networks: one for path execution, and other for obstacle avoidance. If an unknown obstacle is detected by the distance sensors, the mobile robot tries to contour this obstacle, and return to the planned path as soon as possible. The Khepera Simulator software [11], explained below, was used for the experiments.

1.1. Related Work

Some other approaches using genetic algorithms (GA) for path planning were proposed on the bibliography. For instance, Grefenstette [12] has proposed the use of GA to evolve rule bases able to control the reactive (sensor based) behavior of mobile robots. Koza [13] has proposed the use of genetic programming to do the same. Michalewicz and Xiao [14] [15][16], proposed the use of GA for both global path planning (model based) and local navigation (sensor based). Through the use of many specific operators, and using closed polygons to model the obstacles, their proposal is able to cope with the complexities of the problem. Other approaches include the use of genetic algorithms for controller design [17], and multi-objective path planning [18], but always using standard occupancy grids as map representations.

2. THE KHEPERA SIMULATOR

The Khepera Simulator is a public-domain software that allows the development of control algorithms for the real Khepera mobile robot, using C or C++. It operates on Unix or Linux workstations using the X11 graphics library.

This simulator is divided in two parts: the "world", that simulates an environment of 1 m^2, occupying the left part of the screen, and the "robot", that simulates the real Khepera robot that has 5 cm of diameter, at the right part of the screen (Figure 1). In the world part the behavior of the robot can be observed while it is navigating in the environment, and in the robot's part it is possible to visualize the robot's sensors readings, and other related information.

Each maze is composed by objects, such as bricks, and lamps that can be positioned arbitrarily in the world section, using the simulator interface.

2.1 Description of the world

The Khepera Simulator has several world models, which can be loaded through the button *load* that is found in the left part of the screen. Besides the available worlds, the simulator allows the creation of new worlds, through the buttons *new* and *save*. The world is composed by objects, as bricks and lamps, that can be selected by the buttons + and - and added or removed of the world with the *add* and *remove* buttons. The bricks can also be rotated on its own axis by pressing the *turn* button. For the robot recognizing the objects in the environment it is necessary to press the button *scan* and to know that the robot is noticing it is enough to press the *!* button.

2.2 The robot's description

The Khepera robot is composed by 8 infrared sensors that, by reflection, detect the proximity of objects; 8 light sensors that measure the level of light in the environment; and two step motors that permit the robot's navigation in the environment.

The robot's position in the world can be specified through the button *set robot* that allows locating the robot in any place of the environment. It is also possible to guide the robot's direction by pressing the button *command* and typing in the line the command *set angle* and the desired angle, for example: *set angle 45*.

3 HYBRID CONTROL ARCHITECTURE

The control system proposed in this paper is hybrid in two senses. First, it proposes the use of both sensor-based navigation (for collision avoidance and navigation through the environment) and map-based path planning. Second, by the use of several techniques from Computational Intelligence ([19] together: Neural Networks are used in to implement the reactive (sensor-based) navigation subsystem; Fuzzy logic concepts [9] are used in the mapping of the environment; and Genetic Algorithms are used to find an optimized path from the starting point to an objective point (target) in the environment. This hybridization provides a synergy among techniques. Sensor based navigation is the first level, responsible for executing the planned trajectory, avoiding unknown (or moving) obstacles that can appear if the environment, since it is dynamic (there are other mobile robots and/or obstacles can be moved around). The use of fuzzy concepts in the mapping allows the modeling of imprecision when maintaining the map, mainly if it needs on-line update based on the sensor readings. The planning of paths is achieved by the use of genetic algorithms that can find (sub)optimal paths in any sort of environment, without the drawbacks of other search procedures.

In figure 2 a block diagram of the entire system is presented. From the Simulator, the system gets information from the robot and target positions (1), and the list of known obstacles (2) to build the "Fuzzy Map". The Genetic Algorithm block then searches a viable (and, hopefully, optimized) path through the environment and its known obstacles, minimizing the intersections between the segments of the path and the fuzzy obstacles (3). After that, a list of intermediate points is passed to the neural network reactive subsystem (4) that guides the robot following the planned path, acting on the motor controls (5). If the robot senses unknown obstacles (6), it takes avoiding actions, and then returns to its planned path as soon as the obstacle is avoided. The simulation ends when the robot arrives to its desired target position.

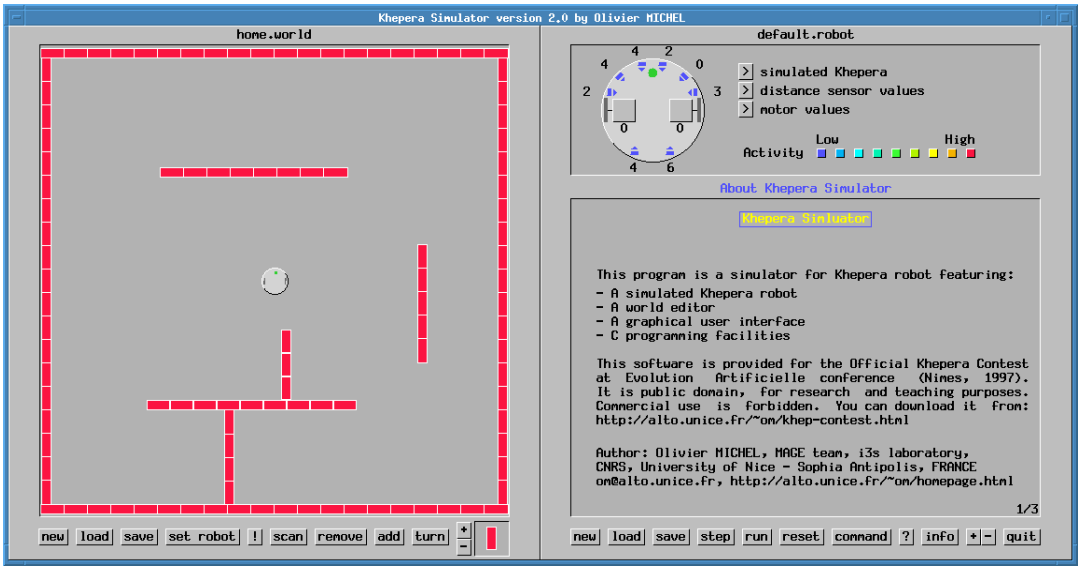


Figure1 - The Khepera simulator.

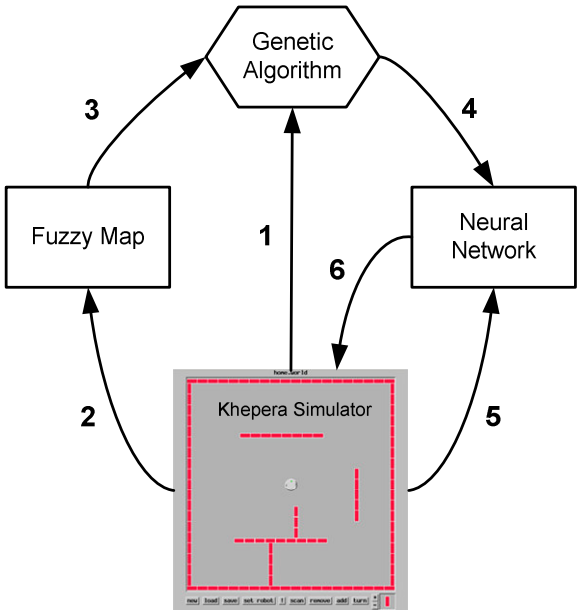


Figure 2: Block diagram.

The objective of the autonomous navigation is to allow the robot to arrive to a pre-determined point (objective point), avoiding collision. This is done by a sequence of sub-goals that is obtained by the trajectory planning algorithm. Therefore, the path is represented by a list of intermediate points (false lights) that the robot must find, in order to make the robot to arrive at the target point, avoiding collision with obstacles. Figure 3 presents an example of the robot's behavior with the map of the environment and the use of the false light points (indicated as "Sub-Goals").

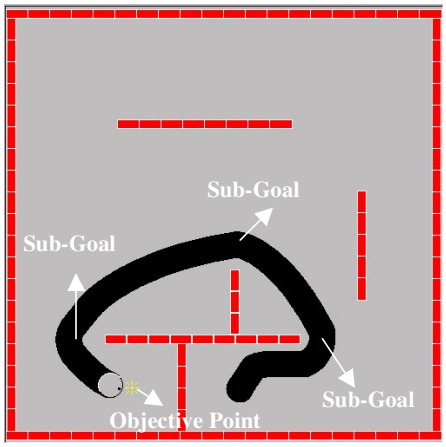


Figure 3 - Execution of the trajectory in a known environment

4 NEURAL NAVIGATION THROUGH THE PLANNED PATH

4.1 Reactive Control for the Autonomous Navigation with Artificial Neural Networks

In this work we used two neural networks for the control of robot navigation [20]. The use of neural networks for

sensor-based approaches facilitates the training of the navigation of the vehicle (robot), straight from the sensors' readout (each sensor reading represents an input for the neural network). Thus, it is possible to define a control strategy starting from navigation examples, not necessarily the definition of rigid rules, taking the generalization of the reactions of the vehicle in for several situations found in the environment. Besides, it facilitates a larger efficiency in the processing of the readings of the sensors, allowing interaction with the environment in real-time. After several experiments, the work resulted in two feedforward neural networks and supervised training for the robot's control. Each neural network has 9 inputs (8 sensors of the robot + one bias term), 27 neurons in the hidden layer and 2 in the output layer (the controls of the motors). One of the networks is responsible for avoiding the obstacles and the other for directing the robot towards the objective point established in the environment (represented in the simulator as a light source).

The two neural networks (for light following and collisions avoidance) are integrated by a simple program that prioritizes the collisions network. Therefore, when the robot is too close to an obstacle, the network for light following is disabled. Only when the obstacle has been avoided, it is reactivated. This control allows the robot to navigate throughout the environment without colliding with obstacles, while following its light sensors readings, and making the robot move towards the light source.

In the approach proposed in this paper, the light sensors are set to indicate the relative direction of the next intermediate point in the genetic planned path. When the robot gets closer enough to this point, the next point is targeted, and so on, until the robot reaches the real objective, i.e., the final point in the planned trajectory.

When there are no lights close enough to activate the light sensors, and there are no close obstacles, the robot goes forward. Therefore, there is no kind of global control of the robot´s trajectory. Consequently, it is possible that it do not reach the light source, because it identifies the light behind an obstacle, and then it contours the obstacle and the sensor will not find the light anymore. Figure 4 shows an example of this undesirable situation. The global path planning using genetic algorithms is the approach proposed in this paper to avoid such situations.

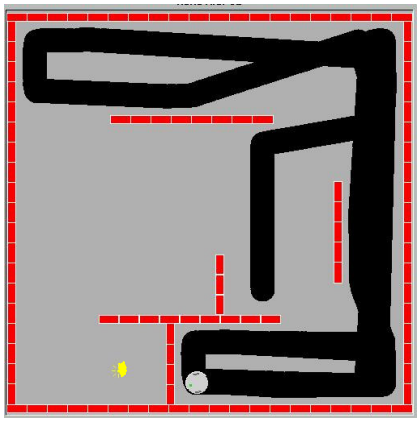


Figure 4–Example of execution in which the robot cannot reach the objective.

5 MAPPING THE ENVIRONMENT WITH FUZZY OBSTACLES

Planning a path through an environment that has obstacles requires a model of these obstacles, so as to apply an algorithm to find an obstacle-free path [20]. This is usually done by representation of the obstacles with polygons [1], or logic-based representations [2]. In Marchi and Fabro [20], the data structure used for the map was a 1000x1000 bits matrix, representing the simulated world of 1 m^2. Every position in the matrix represented an obstacle (a bit 1) or the absence of obstacles (a bit 0). An example of this data structure is presented in Figure 5. All known obstacles (the bricks present at the environment) were mapped into this matrix, providing an exact (crisp) model of the environment.

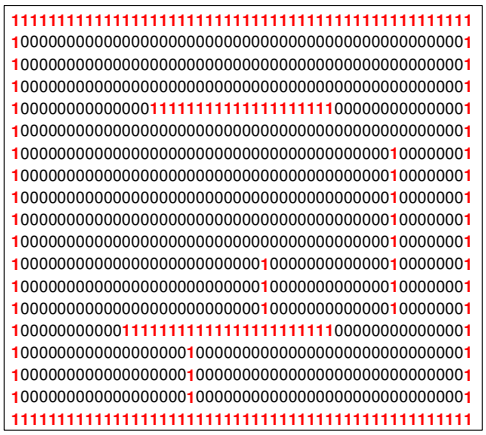


Figure 5–Example of non-fuzzy map of the environment.

In this paper, a new approach for constructing the map of the environment is proposed, based on the drawbacks of the "binary map" approach, and previously mentioned. Instead of representing the "presence" or "absence" of obstacles by binary digits, it is proposed another continuous representation, using "fuzzy obstacles". Each position of the environment is represented still in a 1000x1000 matrix, but instead of discrete binary values, each position has a "fuzzy" representation of the obstacles. A (membership) value of 1.0 represents that a given position is the "center of an obstacle", thus indicating that this position is as far as possible of the borders of a certain obstacle. A membership value of 0.0 represents the absence of obstacle in that position, as in the binary map. But intermediate values (between 1.0 and 0.0) indicate how far a point is from being a center of obstacle and being a border of an obstacle. Thus, this coding has the great advantage of indicating the direction that the robot must follow in order to contour the obstacle. This information can be used by the planning algorithm to find the borders of obstacles, and thus find free paths among obstacles much easier than in the previous coding. In order to avoid problems such as the exponential growth of processing time of the search algorithm, and to find (sub) optimal path in any given environment, the search algorithm was performed by a genetic algorithm, that is presented in section 7.

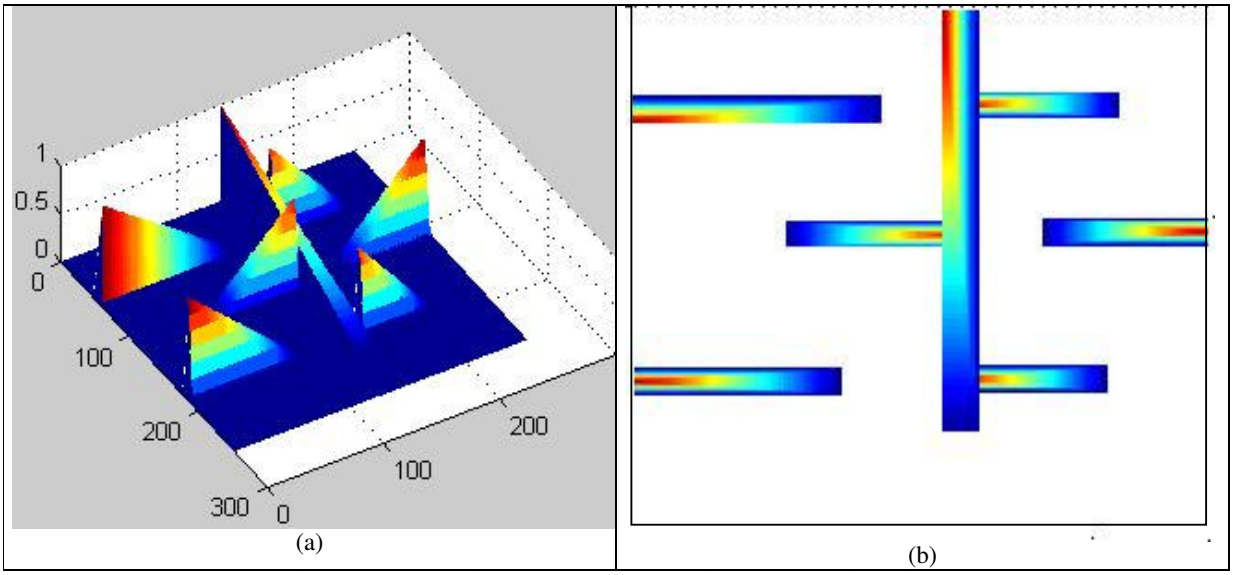


Figure 6–A Fuzzy map of an environment: 3D vision (a) and vision from the top (b).

6 FUZZY OBSTACLE FUSION

If any two obstacles are close, and relatively aligned (that is, they have the same orientation, and are close enough such that the robot cannot pass between then) we can transform these two obstacles in a single one. The reason to do this is because is possible to extract very important information from the fuzzy obstacles: one can see how far some point is from the border of the obstacle, just looking at the membership of this point to the obstacle. Therefore, if we manage to merge a row of obstacles into a single one, we will be able to find more easily a path to circumvent this obstacle: just get closer and closer to its border. This kind of information makes the genetic algorithm (see next session) work easier, achieving a faster convergence towards paths that do not cross obstacles.

To further improve the information provided by fuzzy obstacles, a post-processing step was done to include information about the interactions of two close obstacles that could not be merged (for example, if they were orthogonal). In this case, it is possible to include into the fuzzy obstacle the information that one side, which is closer to another obstacle, is therefore far from the border of this obstacle. We can do this by simply moving the center of the obstacle to the border that is closer to another obstacle. In this way, the connections between obstacles are represented, and the genetic algorithm can use this information to find paths that are as far as possible from this intersection points between two fuzzy obstacles.

These ideas lead to a *Fuzzy Map* of the environment, which is presented in Figure 6. This map includes all the known obstacles present in the environment, which are obtained directly from the simulator, and manipulated to form the map depicted. In the figure a tri-dimensional view is provided where it can be observed the impact of the fuzzy map in the visual interpretation of the obstacles in the environment. It is shown, also, in the top view, that the use of the fuzzy map of the environment does not affect the obstacles and their positioning themselves, when compared with the discrete (binary) map of the environment.

7 OFF-LINE PATH PLANNING USING GENETIC ALGORITHMS

A solution for the problem of navigation is a path through which the robot can reach the target. Since the number of steps of this path is unknown, a possible solution was coded by a variable size list. That is, as a double-chained chained list, where each element is a coordinate $(x,y$ point) of the environment. Since a single path represents a solution to the problem, each individual of the population of the genetic algorithm will have a single chromosome. Therefore, in this work, chromosome and individual will have the same meaning.

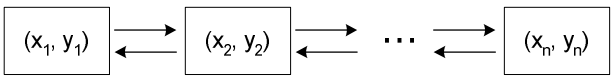


Figure 7: A double-chained list representing a chromosome (a path)

7.1 Fitness function

To evaluate a path, an objective function was defined, based on the distance to be covered by this path. This objective function will lead to the minimization of the length of the path. Therefore, the best path will be the shortest path as possible. However, if this path crosses some obstacle, it cannot be executed. Therefore, it is necessary to include a penalty in the evaluation of the planned path, using the information about the obstacles in the environment (that is, the fuzzy map).

To get this penalty, the intersection between the planned path and the obstacles that are represented in the map is calculated. This is done by covering all the intermediate steps of the path that the robot should follow, and adding the membership of each point where an obstacle is found. The number of steps is also used as a penalty factor, so that the algorithm looks for solutions with the minimum number of intermediate steps. The equation to evaluate a given path is:

$$fitness\ (t) = ctte - (t\_length + p*t\_obs + t\_seg) \quad (1)$$

where *t_length* is the length of the complete path (sum of the lengths of all segments); *t_obs* is the sum of the memberships of obstacles in the path; *t_seg* is the number of segments (steps); *p* is the penalty weight; and *ctte* is an arbitrary constant to transform the function into a maximization. In the experiments, the constant used was 1000, and the penalty weight was 20.

7.2 Genetic operators

Three knowledge-based genetic operators were especially developed so as to include heuristic information about the problem.

The first operator is the "mutation by insertion" that inserts a random node of position (*x,y*) into the path. However, this is not a blind operator, since the insertion always occurs after a non-feasible node. A non-feasible node of the chained list is a node where begins a segment of the path that passes through an obstacle. If another random node is inserted, there is a chance that the segment leaving this node becomes feasible (see example in Figure 8).

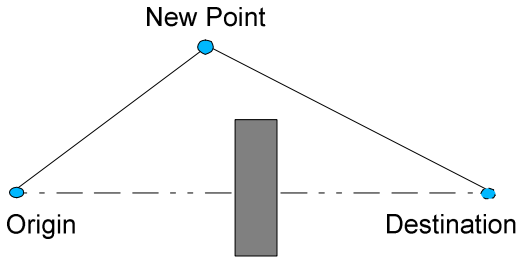


Figure 8: Example of the mutation by insertion operator.

For the correct functioning of this operator, however, the first element of the stack must always be a reachable point from the initial position of the robot. To assure this, a specific initialization process of the population was developed. The initialization process generates a path with variable number of intermediate points, between 0 and 15. This operation guarantees that the first node is always reachable directly from the initial position of the robot. All the remaining points of the path are randomly generated. The list contains only intermediate points of the path, and can be an empty list as well, indicating that the path is a straight line from the initial position towards the target.

Another operator developed in this work is the "mutation by deletion of a node" operator. This operator removes one node that is just after a non-feasible point in the chained list, trying again to create a feasible path. An example of its function is presented in Figure 9.

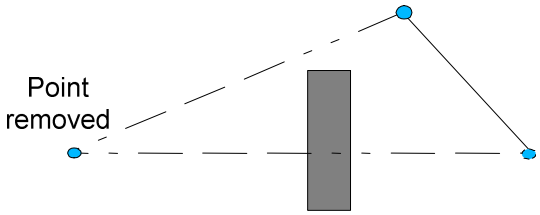


Figure 9: Example of the mutation by deletion operator.

A specific crossover operator was also developed to substitute the traditional one-point crossover. This new operator is illustrated in Figure 10 and the following steps are done to generate offsprings:

a) Selection of two individuals from population, parent1 and parent2;
b) Analysis of the two individuals, computing the number of non-feasible points in each one;
c) Cut parent1 in the last non-feasible point, that is, the non-feasible point closest to the end of the chromosome;
d) Cut parent2 in the first non-feasible point, that is, the point closest to the beginning of the chromosome;
e) Concatenate initial part of parent1 with the final part of parent2, generating the offspring;

With the application of this operator one new individual is created. This descendant will have no more than one non-feasible segment in its path, exactly at the concatenation point. Figure 10 illustrates the special crossover operator: at the top there are two unfeasible paths (parent #1 and #2), below there is a feasible offspring after crossover and insertion mutation.

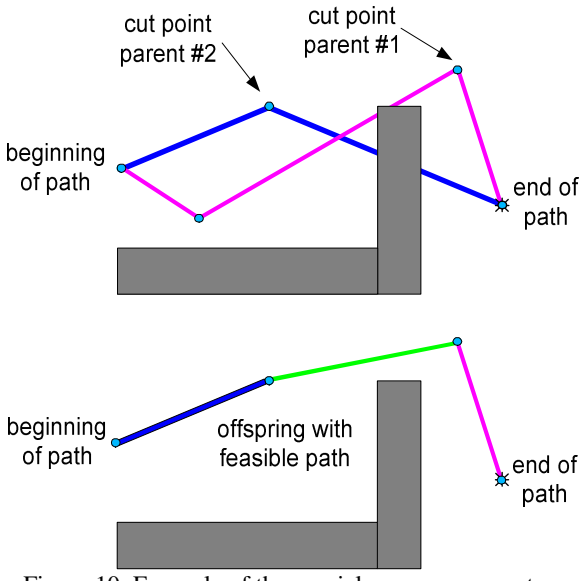


Figure 10: Example of the special crossover operator.

7.3 Running parameters

The population used for the execution of the genetic algorithm was 100 individuals. The probabilities of application of mutation operators were empirically adjusted to 15%, and 80% for the crossover operator. Looking for a solution to this problem, we used a crowding factor. The library used [21] does not implement this technique directly, but it has a variation of the Genetic Algorithm called Deterministic Crowding that works as follows:

a) Random selection of two individuals from the population;
b) Application of the special crossover to these individuals, generating an offspring;
c) One of the two individuals selected is replaced by the new individual: this substitution is established by the similarity degree between the two individuals – the most similar to the descendant will be replaced.

This variation of the algorithm presented a slow convergence rate, but an improved ability to find feasible solutions, even in more complex environments, as those presented in Figures 11 and 12. However, due to its slow convergence rate, the processing time was considerably large.

The stop criterion of the algorithm needed to be modified to consider the broad range of situations and the convergence of the algorithm. The search ended 20 generations after the first feasible path was found. However, when using the Deterministic Crowding, the selection of the individuals is done randomly by the library used here. Therefore, the algorithm did not achieved good results in the optimization of the feasible path.

To solve this problem, a post-processing phase was included at the end of the execution of the algorithm. This phase is responsible only for the local improvement of the path. This is carried through successive applications of the mutation operator that modifies each point to some another point in the neighborhood (random value between 0 and 10). This operator is applied several times, while it is capable to improve the path. After being applied 10 times without improvement in the path, the post-processing phase ends and the resulting path can then be executed by the robot.

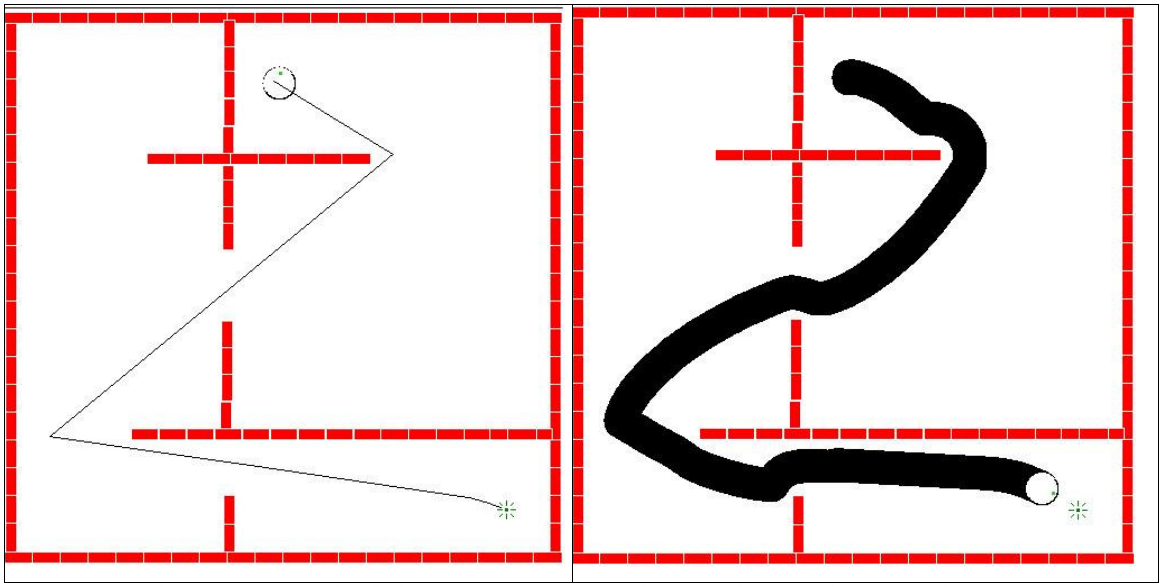
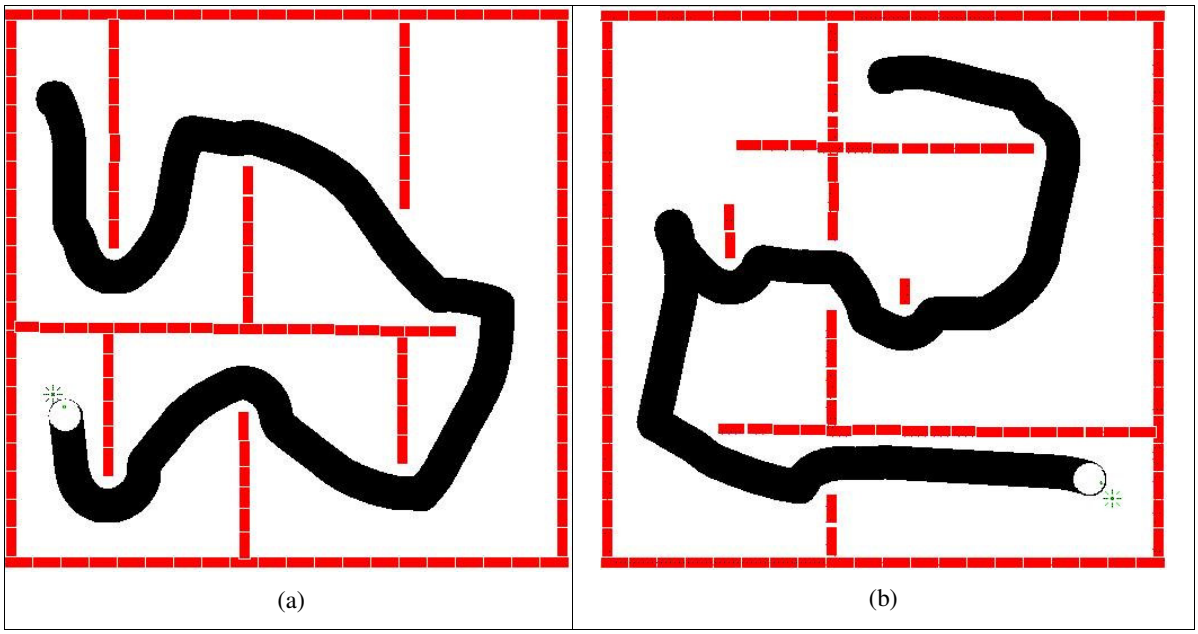


Figure 11 - Example of planning and execution of the path in an environment of great complexity.



(a) (b)

Figure 12 -Execution of the planned path in a complex environment (a), and with unknown obstacles (b).

8 EXPERIMENTS AND RESULTS

Experiments where done with the environments presented in Figures 11 and 12, aiming at evaluating the performance of the genetic algorithm to find feasible solutions. The solutions found, as well as the paths executed by the neural navigator, are presented together. Some differences between the planned path and the executed one are due to the neural network-based execution, because the robot avoids getting too close to obstacles during the execution of the planned path. The subsystem of neural navigation also has the capability to locate and avoid obstacles not present (moving objects) in the fuzzy map, as shown in figure 12(b).

The evolution of the genetic algorithm had a convergence time proportional to the complexity of the path to be found. The executions varied from the cases where a feasible path do existed at the initial population, to cases where it was necessary up to 500 generations to find a feasible path (environment in figure 12a). Anyhow the genetic algorithm showed its efficiency for path planning in complex environments.

9 CONCLUSIONS

With the development of this project, the difficulty of the path planning problem was confirmed. The lack of knowledge of the different kinds of environments in which the robot will navigate makes it difficult to include enough heuristic information for planning efficient paths, and leads the system to look for more complex solutions. However, even with little heuristic information available, it is possible to the genetic algorithm to find feasible solutions in different environments. According to our experiments, the quality of such solutions could be considered human-competitive.

The representation of the environment using fuzzy obstacles allowed the genetic algorithm to face the problem from a smoother fitness landscape, when compared with that generated by a crisp map of the environment. This fact leaded to an improved capability of comparison among the multiple evaluated paths, making the process of finding feasible paths faster and more efficient.

The use of neural networks to execute the planned trajectory allows the system to avoid collisions with unknown (and dynamic) obstacles.

Such integrated use of different technologies to solve a complex problem appears to be, by its synergic strengths, a remarkably consistent approach to the problem of navigation in mobile robotics. With techniques used where its features are more relevant, the global solution achieved combines the best of each of them: imprecision modeling with fuzzy obstacles, reactive control with neural networks, and unstructured search with genetic algorithms. This hybrid solution proposed here presents a high potential of application to real-world problems. The authors believe that other solutions similar to this approach will soon appear in real applications due to its potential to solve problems more complex than those that can be solved by the use of any technique alone. Results encourage the continuity of this work in several directions, such as the development of new knowledge-based operators for the genetic algorithm, and the inclusion of obstacles in the map during the navigation (i.e. unknown obstacles) in order to update the map during the interaction with the environment (map building).

ACKNOWLEDGMENTS

The first author would like to thank the support of CAPES.

REFERENCES

[1] T. Lozan-Pérez, and M.A. Wesley, "An algorithm for planning collision-free paths among the polyhedral obstacles", Communications of the ACM, vol 22, no. 10, pp. 560-570, 1979.

[2] R.E. Fikes, P.E. Hart, and N.J. Nilson, "Learning and executing generalized robot plans", Artificial Intelligence 3, pp. 251-288, 1972.

[3] J. Borenstein, and Y. Koren, "Real-time obstacle avoidance for fast mobile robot", IEEE Transactions on Systems, Man and Cybernetics, vol 19, no. 5, pp. 1179-1187, 1989.

[4] R.A. Brooks, "A robust layered control system for a mobile robot", IEEE Journal of Robotics Automation, vol. 2, no. 1, 14-23, 1986.

[5] S. Ishikawa, "A method of indoor mobile robot navigation by fuzzy control", in Proc. Int. Conf. Intelligent Robots and Systems, Osaka, Japan, Nov. 3-5, pp. 1013-1018, 1991.

[6] C. Kozakiewicz and M. Ejiri, "Neural network approach to path planning for two dimensional robot motion", in Proc. Int. Conf .Intelligent Robots and Systems, Osaka, Japan, Nov. 3-5, pp. 818-823, 1991.

[7] M. Sekiguchi, S. Nagata and K. Asakawa, "Mobile robot control by a structured hierarchical neural network", IEEE Control Systems Magazine, vol. 10, no. 3, pp. 69-76, 1990.

[8] H. R. Beom, and H.S. Cho, "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning", IEEE Transactions on Systems, Man and Cybernetics, vol. 25, no. 3, 1995.

[9] W. Pedrycz, and F. Gomide, "Fuzzy Systems Engineering". John Wiley & Sons, 2007.

[10] S. Haykin, "Neural Networks and Learning Machines", 3rd edition. Prentice-Hall, 2008.

[11] O. Michel, "Khepera Simulator version 2.0". Homepage: http://diwww.epfl.ch/lami/team/michel/khep-sim/,1996.

[12] J. Grefenstette, "Evolutionary Algorithms in Robotics", In: M. Janshidi, C. Nguyem, eds., Robotics and Manufacturing: Recent Trends in Research, Education and Applications, pp. 65-72, ASME Press, 1994.

[13] J. R. Koza, "Genetic Programming", MIT Press, Cambridge, MA, 1992.

[14] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", Springer-Verlag, New York, 1998.

[15] J. Xiao, Z. Michalewicz, L. Zhang and K. Trojanowski, "Adaptive Evolutionary Planner/Navigator for Mobile Robots", IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp 18-28, 1997.

[16] Xiao, J. and Michalewicz, Z., "An Evolutionary Computation Approach to Robot Planning and Navigation", in Hirota, K. and Fukuda, T. (eds.), Soft Computing in Mechatronics, Springer-Verlag, Heidelberg, Germany, 117 – 128, 2000.

[17] C. Thomaz, M. Pacheco, and M. Vellasco. "Mobile robot path planning using genetic algorithms", Lecture Notes in Computer Science, Volume 1606, pp. 671-679 1999.

[18] O. Castillo and L. Trujillo, "Multiple Objective Optimization Genetic Algorithms for Path Planning in Autonomous Mobile Robots", International Journal of Computers, Systems and Signals, Vol. 6, No. 1, 2005.

[19] L. Rutkowski, "Computational Intelligence – Methods and Techniques". Springer-Verlag, 2008

[20] J. Marchi, and J.A. Fabro, "SNNAP - Sistema Neural para Navegação em Ambientes Pré-Mapeados", Anais do IV Congresso Brasileiro de Redes Neurais, São José dos Campos ITA/CTA, pages 118-123, 1999. [In Portuguese]

[21] M. Wall, "GALIB Programming Library Version 2.4.5", HomePage: http://lance.mit.edu/ga, February 2008.