A Hybrid GA-ANN Approach for Autonomous Robots Topological Navigation

ABSTRACT

This paper proposes a hybrid approach using Genetic Algorithm and Artificial Neural Networks for autonomous path planning and motion control for mobile robots. A Topological Navigation approach is adopted, using the environment mapped as a graph. A Genetic Algorithm is used to generate and evolve a set of feasible actions, aiming to lead the robot to the goal considering the shortest path. Each action is a different reactive behavior designed for a specific environment feature such as corridors, turns or intersections. Then, an Artificial Neural Network is trained to recognize the different environment features, and the next behavior is activated every time the ANN detects a transition. Experiments were performed in Player/Stage robotics simulator and obtained results showed this approach as a promising way to plan and execute a path.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*; I.2 [**Artificial Intelligence**]: Robotics

General Terms

Algorithms, experimentation

Keywords

Genetic Algorithm, Robotics, Autonomous Navigation, Artificial Neural Networks

1. INTRODUCTION

Path planning and execution are very important tasks for autonomous mobile robots. In this process, firstly it is necessary to define the path from a source point to a destination point, and then execute the control and navigation tasks detailed by a set of intermediate goals. The navigation task can be described by a set/sequence of behaviors, which indicate the main actions to be performed in order to achieve the final destination.

According to [1], designing a mathematical model for an optimal or near-optimal planning and navigation control strategy of a robot is a very complex task. Methods based in heuristic optimization (e.g. A*, D*, visibility graph [2]) can provide good solutions to the robot path planning, but algorithms like these require a precise localization of the robot during all the path execution, a previously provided and well defined map, and also need a selection of an efficient method for controlling the robot's navigation (control movements along the planned path). Machine learning methods have been used for this purpose as well as evolutionary algorithms.

The present paper proposes the use of a genetic algorithm (GA) to solve the path planning problem in mobile robotics. Genetic algorithms are metaheuristics inspired by the fittest survival principle that mimic processes of biological evolution to solve problems [3]. This principle states that individuals more suited to the environment have a greater probability to survive as well to reproduce themselves. [4]. Thus, each individual is represented by a sequence of behaviors which allows the robot to reach a goal point at the environment, and then the GA algorithm objective is to evolve a set of individuals in order to find the best feasible sequence of actions.

The proposed system is based on a topological navigation approach [5], in which the environment is mapped with a graph representation. Each node is associated to one or more reactive behaviors, and each reactive behavior is responsible to make the robot safely reach the next node in this graph. Thus, the GA generates a suitable sequence of reactive behaviors, setting an adequate behavior for each decision point in this graph as, for example, intersections and turns.

There are two main advantages of this approach. First, the robot does not have prior knowledge about the environment (internal representations or maps), discovering the best path through the evolutionary process. Second, this evolutionary process is carried on the graph representation proposed that does not require a very detailed map, but only a sketch with the main elements of the environment. A precise localization is also not required, since the system knows its approximate position according to the current topological node, and the reactive behaviors are able to make the robot safely reach the next node. Sensorial data is used, so the robot is able to properly react to dynamic elements and

SAC'14 March 24-28, 2014, Gyeongju, Korea.

avoid obstacles.

This paper is organized as follows: Section 2 describes some related works, and section 3 describes the environment representation adopted. In Section 4, the developed genetic algorithm is explained and the experiments presented in Section 5. The conclusion and future works are presented in Section 6.

2. RELATED WORKS

Some previous works have already applied the genetic algorithm to the path planning problem. The domain knowledge was aggregated to a GA for path planning in static and dynamic environments of a mobile robot in [6]. The environment was represented by orderly numbered grids. The chromosome represented a path, composed by a sequence of grid numbers starting from the source and ending at the target position. The domain knowledge was used by specialized operators, combining a local search technique to improve a path. The evaluation considered feasible (collision free path) and infeasible paths. The fitness function value was defined as the sum of distances of each line segments and, if an infeasible path was generated, a penalty was added. The algorithm was also used in dynamic environments. In this case, if an obstacle was added to the map, a new path should be found by the genetic algorithm. However, this work is unclear if sensors were used to perceive and avoid obstacles, and also if they used any specific localization and navigation control methods. The proposed method was evaluated in simulation and demonstrated effectiveness and efficiency.

In [7], it was developed a GA to path planning for local obstacle avoidance of a mobile robot in static environments. The objective was to minimize the length of the path and the number of turns. The environment was modeled as a grid sliced into rows and columns. The chromosome structure was composed by four variables: *path-flag, path-location, path-direction and path-switch.* The robot had two possible movements: row-wise and column-rise. The path-flag variable indicated the possible movements; the path-location variable referred to robot position; the path-direction variable indicated the direction the robot should turn to proceed to the next node; and the path-switch variable allowed the robot to switch back and forth between a row-wise and a column-wise movement in a single path. The fitness value considered the path length, the number of turns and the collisions. According to the authors, the experiments showed that the proposed algorithm works better than other algorithms, but they did not explain how the robot navigates through the environment when dealing with real world imprecision and uncertainty.

In [8], an improved genetic algorithm for optimum path planning for mobile robots navigation was proposed. To generate the initial population, an obstacle avoidance algorithm was used to generate the first population and another algorithm was used to distinguish if the path was feasible or infeasible during the genetic evolution. The environment was represented by a numered grid model, in which each numbered grid unit is a node. So, the path was a sequence of nodes that is the individual representation. Single-point random crossover operator was used to generate the new individuals. Mutation operation was applied to a random node of the chromosome. After the application of each genetic operator, the distinguish algorithm was used to correct the infeasible paths. A refinement operator was used to reduce the path distance based on the triangle principle. If three nodes form a right triangle, the node with 90° to the others is deleted. In the delete operator, if two nodes can be connected without crossing any obstacles, the intermediate nodes between them can be deleted. The value fitness function was the inverse of the total distance of the path. The effectiveness of the proposed genetic algorithm was demonstrated in simulation and with a real robot. In the real experiment, the navigation control relies on sensor readings. Thus, the success of the navigation depends of the precision of the sensor readings.

A genetic algorithm was also applied to the path planning problem in dynamic environments by [9]. The environment was represented by a ordely numbered grid model. A chromosome was represented by a sequence of nodes for robot navigation, with each node representing a numbered grid cell in the environment. The initial population was generated randomly, but it was repaired if the information encoded in a chromosome allowed intersecting an obstacle. Single-point crossover was used in two parents to generate two new individuals. In random mutation operator, a random bit-wise binary complement operation or a random small change in a gene was performed. The fitness function was defined as the sum of distances between each node in a path. If there was an obstacle in the direction of the robot, a penalty should be added to the objective function value. The novelty of the work in [9] was the mutation operator. Instead of randomly choose one node to change, all the free nodes which do not intersect an obstacle close to mutation node were evaluated. The node was selected according to the fitness value of total path. The authors said the algorithm was tested in dynamic environments (environments which constantly change after the robot start to move). However, it is not clear how the robot can cope with these situations, without using the Genetic Algorithm to recalculate the path. According to the authors, the experiments showed that the proposed method converges faster than other methods in literature.

For motion planning with multiple mobile robots, [10] proposed a solution using master and slave evolutionary algorithm in maze-like map. The slave algorithm evolutionary was responsible to find the optimal path to each robot. The master algorithm was concerned about overall optimal paths. In the slave algorithm, the chromosome was represented as a sequence of integer values. Each integer represented a robot movement which could be *left, right, up, down.* Scattered crossover and uniform mutation were the operators used in the algorithm. The fitness was calculated as the total length of the path. If the robot did not reach the goal, a penalty was added proportionally to the distance between the last point reached by the robot and the current goal location. A factor which referred to the collisions among robots was added to the fitness function aiming to achieve the global solution. In the master algorithm, the individual was a set of pointers that points to some individual of the slave algorithm. Similar to the slave algorithm, scattered crossover was used. In the mutation, an existing pointer was deleted and the new ones pointed to some other individual in the slave population. The fitness value was defined by the average traveling time of each robot. A penalty was added if the robot did not reach the goal. The proposed method was experimented in several scenarios, which in each of them the algorithm found a collision free path for all robots from the source to the goal. The simulation was done using Java

Applet based GUI, but the authors do not explain how the robot can deal with imprecision or use sensors.

The grid map representation was adopted in these presented works and refers to metric maps, demanding a precise localization and odometry systems. In order to reach the next node, the robot should follow a very accurate sequence of movements, implying that a motion planning is also necessary. However, motion control is not discussed in these papers, the path generation is obtained based on static maps (dynamic maps usually need to re-execute the path planning algorithm). Also, they can not prove robustness in avoiding dynamic elements since they did not present a sensorial system able to recognize these elements and react, with a local motion control or autonomously deciding to replan the path whilst navigating. So, the hybrid approach proposed in this paper aims to overcome these limitations, using a different environment representation and also a different individual representation from the presented related works, resulting in a complete system for both path planning and motion control.

The navigation process proposed in this paper is based on the Topological Navigation System proposed in [5]. In that system, the environment was represented by a topological map, a graph with each node related to a specific spot in the environment with particular structural features (corridors, turns, intersections). Any path in this map could be represented by a sequence of these nodes, resulting in a Finite State Machine [11]. Then, an Artificial Neural Network was trained to recognize the current node using the robot sensors. For each node, there was a set of possible local reactive behaviors which lead to the next expected node/state, also avoiding obstacles. Table 1 shows an example of states and the matching actions. It is also important to mention that in [5] path generation and the assignment of adequate behaviors for each state was not made automatically, being performed by a human operator. Thus, this work extends [5] work since the GA automatically generates a feasible sequence of actions. In this case, a genetic algorithm is introduced to set the best feasible sequence of actions found during the evolutionary process which can lead the robot from an initial to a goal position at the environment. The GA evolution happens over a graph representation of the environment that allows accelarating the evolutionary process.

Table 1: States and possible actions for each one

State	Related Actions
Straight	go forward
Right	turn right
Left	turn left
Intersection	go forward, turn right and turn left
Bifurcation	turn right and turn left

The main difference between the proposed approach and the related works which used GA for path planning, is that it does not generate a precise sequence of points to be visited, but a feasible sequence of local behaviors which lead to the goal. This way, accurate localization is not necessary and the robot does not need to know the environment map. When the ANN detects a state transition (context change), the next reactive behavior is activated, allowing the robot to safely navigate through this topological node and reach the next expected spot. This method can be easily extended to deal with dynamic obstacle avoidance, since reactive behaviors can be robust to dynamic elements which can possibly appear. The environment topological representation and navigation approach are described in the next section.

3. TOPOLOGICAL ENVIRONMENT REPRESENTATION AND NAVIGATION CONTROL

In the genetic algorithm, it is needed to evaluate all individuals during the evolutionary process. Considering this robotics navigation application, this evaluation means that the robot has to navigate on the map following the path and successfully reach the goal. Using a metric map in a robotics simulator (as Player/Stage [12]) could make the evolutionary process very slow, since a continuous search space would be used for optimization. Thus, the topological approach proposed in [5] was adopted, restricting the search space with the use of a graph to model the environment during the genetic evolution.

In [5] topological approach, the nodes are used to represent strategic points with particular structural features. One edge between two nodes indicates that these points are adjacent in the real environment, being possible to reach the next node with a reactive control. This way, a path between an initial point and the goal is merely a sequence of nodes, working as a Finite State Machine. An Artificial Neural Network is trained to recognize the different possible nodes, using robot's sensors data as input. Each node is related to an adequate and particular action (or a set of possible actions) which lead the robot to the next node. So, navigation step is performed using the ANN to detect the current node/state and performing an adequate reactive behavior according to current and next states. In [5], autonomous path planning and behavior selection was not implemented, motivating the study proposed in this work. Figures 1 and 2 illustrate a map and its respective graph representation.

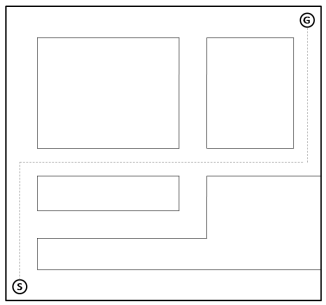

Figure 1: A map example. S is the start position of the robot and G is the goal.

In this example, the possible actions to be performed from each current node are *turn right, turn left, go forward, turn around* and *go straight*. The action *turn around* is activated when the robot is going forward and an obstacle blocks the way. The difference between actions *go forward* and *go straight* is that the first is used in a cross and the second is used in a corridor.

Each node has a list of four adjacent nodes: north, south,

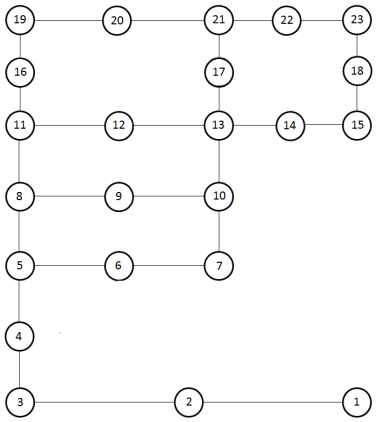

Figure 2: The correspondent graph from Figure 1.

east and west. Accordingly to the direction assumed by the robot and the current action, the position is changed to one of the adjacent nodes. For example, in Figure 2, if the current position is node 13 and the robot came from west direction, the neighbours nodes are: north (node 14), south (node 12), east (node 10) and west (node 17). Furthemore, each node is related to an approximate discrete position in the map that is used to calculate the fitness value. The fitness function and other GA parameters are explained in the next section.

4. GENETIC ALGORITHM

The proposed genetic algorithm (AG) generates a sequence of actions that allows the robot to accomplish the navigation to its destination target point. The objective is to automatically provide the sequence of reactive behaviors to follow a path using topological navigation approach. In most of works found in the literature, the chromosome is modeled as a sequence of exact points that the robot has to visit. In this paper, the chromosome is defined as a sequence of integers, each integer representing a reactive behavior that can be performed by the robot. An example of the chromosome is shown in Figure 3. The possible actions to be performed will be enumerate as go forward (0), turn left (1), turn right (2), turn around (3) and go straight (4).

For example, let's assume the initial and final positions indicated in Figure 1, the robot coming from west and executing the actions defined by Figure 3. According to this chromosome, the robot should perform the actions: *turn left, go straight, go foward, go straight, turn right, go straight, go forward, go straight, turn left and go straight.* In the correspondent graph illustrated by Figure 2, the following nodes will be visited: 3, 4, 5, 8, 11, 12, 13, 14, 15, 18 and 23. It can be observed that the chromosome holds 20 possible actions to be applied, but the goal was reached with only 10 actions. In this case, the remaining genes are not considered.

1	4	0	4	2	4	0	4	1	4	3	4	1	4	0	4	0	4	2	4

Figure 3: An example of the individual representation. Each gene represents a code for an action.

The initial population is generated filling randomly the genes of each chromosome with possible integer values for the related actions. Each chromosome is evaluated performing a robot navigation through the graph representation. At this step, a repair operator is executed. If an action indicated by the gene is not possible to be applied during the graph navigation, the repair operator replaces it by a new one, according to the possible actions to be applied at this node in the graph. For example, if the robot is coming from south to node 5, in Figure 2, and the next gene action to be applied is 1, the turn left action should be executed. However, this action is not possible for this node and the action (integer value) in this gene must be replaced. There are two possible actions for this node, 0 or 2, which is randomly chosen by the repair operator and fixed as new gene value.

The fitness function is based on the length of the path executed by the robot and on the distance from its final position to the goal, according to the graph representation. The distance travelled is the sum of euclidian distances between each pair of nodes, V_i and V_j, from the set of nodes included in the path (Equation 1). When the action *turn around (3)* is chosen, the distance between the nodes is summed twice, once that the robot entried in a blocked corridor. If the goal is reached, this sum is divided by a constant α. If the goal is not reached, the euclidian distance between the current node and the goal node is added to the sum, multiplied by a constant β. Equation 2 is the fitness function explained, where f is the fitness value, α and β are constant values, V_a is the current node, V_g is the goal node and $d(V_a, V_g)$ is the euclidian distance between V_a and V_g nodes.

$$sum = \sum d(V_i, V_j) \tag{1}$$

$$f = \begin{cases} sum/\alpha & \text{if the goal is reached} \\ sum + \beta * d(V_a, V_g) & \text{if the goal is not reached} \end{cases} \tag{2}$$

The population at each generation t is formed by $p\%$ of the best individuals from the population at generation $t-1$. The other $(1-p)\%$ are determined by crossover and mutation. The tournment between two individuals selects two parents from population at generation $t-1$. Thus, the uniform crossover operator is applied over these parents generating a new individual. Mutation can be applied to the new individual following the mutation rate. If the mutation rate is satisfied for one individual, one gene randomly selected is changed, that means select randomly another action (integer value) to be performed. Finally, the fitness values of the new individual is determined. The new individual is inserted in the population, except if its fitness value is equal than one of its parents. In this case, the reproduction process is repeated. The evolutionary process evolves until one of the two stop criteria has been satisfied: if the best individual found so far is not improved by a number of generations or if the maximum number of generation is reached.

After the evolution happens over the graph representation, the best solution found is executed using the simulator Player/Stage. At this point, the ANN proposed in [5] is then used to detect changes in the current state, activating actions encoded in the GA best solution. In the next section is presented the experimental evaluation and results analysis.

5. EXPERIMENTS AND RESULTS

To evaluate the performance of the GA, experiments were carried out in the map represented in Figure 4, wich is similar to map used in [10]. In theses experiments, the start and goal nodes are indicated in the figure by S and $G1$, respectively. The values $\alpha = 2$ and $\beta = 2$ were adopted for the fitness function and the individual length was set as 60 genes. The stopping criteria are 10 generations without update the best fitness or to reach a maximum number of 50 generations. These values were empirically determined based on previous tests.

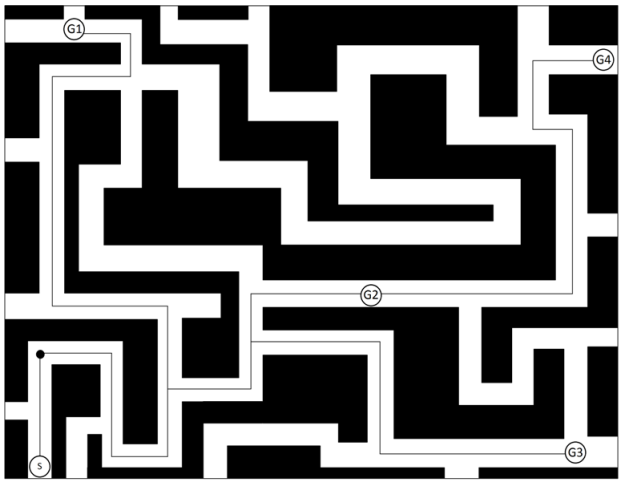

Figure 4: The map used in the experiments. S is the start position of the robot and G1, G2, G3 and G4 are the goals.

However, the results found evaluating some other parameters of the proposed GA are reported next. The first one is related to the population size as describe in Table 2. The columns represent the population size, the number of generations spent, the fitness value of the best individual and the run time in miliseconds to convergence, respectively. These results are average values followed by standard deviations found after 100 runs of the GA. In these experiments, mutation rate was fixed as 0.1 and a elistism strategy named $E_1/2$ was adopted. This elitism means that 50% of the best individuals in the population at generation t will be in generation $t + 1$.

The population size evaluated are 25, 50, 100, 150, 300, 500, 1000. In Table 2, it can be noted that the number the generations decreases and the best fitness gets better as the population size increases, although the run time increases. The algorithm found the same best final solution in all executions with 300, 500 and 1000 individuals. However, the population size of 300 individuals presents the best compromise between solution value and computational time. Thus, this population size was set to evaluate different mutation rates.

The different values applied to evaluate mutation were 0.05, 0.1, 0.15 and 0.2 (Table 3). It is possible to observe that the mutation rate had a little interference in the results. The lower value of 0.05% has the best results with faster convergence.

Using the population size of 300 and mutation rate of 0.05, the last analysis of the GA parameters is about the proportion of individuals maintained to the next generation during the evolutionary process. The elitism strategies evaluated were named as E_0, E_1, $E_{1/3}$ and $E_{1/2}$. The strategy E_0 means to update the whole population with new individuals, while E_1 keeps only the best individual from the previous population. The strategies $E_{1/2}$and $E_{1/3}$ keep, respectively, one half and one third of the best individuals from previous population. Table 4 shows the results achieved.

Table 2: Analysis of population size. Here, the mutation rate is 0.1.

Population size	Generations to converge	Best fitness	Time (miliseconds)
25	15.91 ± 5.28	53.86 ± 7.25	3700 ± 4852.36
50	15.59 ± 5.44	50.20 ± 2.45	7800 ± 5427.20
100	15.42 ± 4.92	49.10 ± 1.57	15200 ± 6273.24
150	14.77 ± 5.13	48.63 ± 0.76	22400 ± 8302.37
200	13.75 ± 3.70	48.68 ± 0.83	27200 ± 7664.69
250	12.88 ± 3.07	48.50 ± 0.25	32100 ± 8444.11
300	12.53 ± 2.50	48.48	37700 ± 8391.31
500	11.54 ± 2.23	48.48	58100 ± 12366.05
1000	10.38 ± 0.80	48.48	112500 ± 10187.63

Table 3: Analysis of mutation rate.

Mutation rate	Generations to converge	Best fitness	Time (miliseconds)
0.05	12.44 ± 2.72	48.48	37100 ± 8444.11
0.1	12.53 ± 2.5	48.48	37700 ± 8391.31
0.15	12.88 ± 2.87	48.50 ± 0.25	38300 ± 9107.11
0.2	12.64 ± 2.79	48.52 ± 0.39	38200 ± 9574.80

Table 4: Analysis of elitism proportion rate.

Elitism proportion rate	Generations to converge	Best fitness	Time (miliseconds)
E_0	47.49 ± 7.44	55.67 ± 4.77	247100 ± 38828.68
E_1	10.98 ± 1.34	48.48	62300 ± 7895.15
$E_{1/3}$	11.75 ± 1.92	48.48	47500 ± 8918.82
$E_{1/2}$	12.44 ± 2.72	48.48	37100 ± 8444.11

When the best solution was discarded from a generation to the next (E_0), the convergence became slow and the maximum number of generations was almost reached. If elitism is introduced, the best solution was found in all tests without relevant difference among the strategies E_0, $E_{1/3}$, and $E_{1/2}$. In this case, strategy E_1 returns the best solution faster than others.

As mentioned in Section4, a repair operator was proposed to be applied when the individual is evaluted. In order to verify the impact of this operator, the genetic algorithm was executed also without this operator. In this experiment, the genetic algorithm was set with population size of 300 individuals, mutation rate value 0.05 and elitism strategy E_1. The results are presented in Table 5. The method returned the same value in all executions, spending more time than using the repair operator. Furthemore, the method with repair operator returned a better solution value. The best solution found is the one illustrated in Figure 4, where the black circle indicates the end of the path.

Table 6 presents the results found for all goals ilustrated in Figure 4. The proposed method takes almost the same computational time and number of generations for the different goals and always finds the optimal solution. This re-

Table 5: Analysis of the impact of repair operator.

Repair operator	Generations to converge	Best fitness	Time (miliseconds)
No	30.58 ± 0.50	63.64	275200 ± 46176.65
Yes	10.98 ± 1.34	48.48	62300 ± 7895.15

sults shows the effectiveness and efficiency of the proposed approach to solve this problem.

Table 6: Results of the experiments with different goals.

Goal	Generations to converge	Best fitness	Time (miliseconds)
1	10.98 ± 1.34	48.48	62300 ± 7895.15
2	10	31.72	53600 ± 4824.18
3	10.01 ± 0.1	44.35	52900 ± 4776.84
4	11.18 ± 1.35	57.65	62800 ± 8297.50

A full navigation test was performed applying the proposed method to the maze shaped map represented in Figure 4, where the start and goal are indicated in the figure by S and $G1$, respectively. This experiment was performed in Player/Stage simulator, using a simulated Pioneer P3-AT robot with a 180° SICK Lidar sensor. The ANN proposed in [5] was trained to recognize seven possible states: *corridor, crossing, left turn, right turn, right fork, left fork* and *T-shaped bifurcation*. The ANN training database was collected in a different map with similar structural features in order to evaluate the ANN robustness and generalization capability. Five reactive behaviors were modeled according to actions defined enumerated from 0 to 4 in Section 4. In corridors, a simple wall-following behavior was implemented; in crossings, bifurcations and turns a wall-following behavior was used to align the robot, and then applied a constant speed and steering angle according to the action.

The graph representation was made as explained in section 3, where the GA evolves with 300 of population size, 0.05 of mutation rate and elitism strategy E_1 as already explained. The resulting action sequence returned by the best individual was: *go straight, go straight, go forward, go straight, turn right, go straight, turn right, go straight, go forward, turn left, go straight, turn left, go straight, go forward, go straight, turn left, go straight, turn right, go straight, go forward, go straight, turn right, go straight, turn left, turn left and go straight*. The robot successfully performed this actions sequence using the topological navigation approach proposed in [5]. A video showing this navigation is available in `https://www.4shared.com/video/6PrFtgIQ/GA_ANN.html`.

6. CONCLUSIONS AND FUTURE WORKS

The hybrid approach proposed in this paper was successfully applied to autonomously generate and follow a path in a topological map, demonstrating the feasibility and efficiency of this method. The robot autonomously reached the goal with simple reactive behaviors, without knowing the environment map or its precise localization. Complex mapping and localization algorithms were not necessary, allowing the implementation of this approach even in systems with limited computational resources. It is also important to highlight that the robot no longer needs to know the path to be followed as proposed in [5], since it is only necessary to activate the next reactive behavior generated by the GA when a state transition is detected. Thus, the improvements to [5] work can be seen as some of the main contributions of this paper. The proposed system showed to be a promising approach for autonomous path planning and mobile robots motion control.

As future works, the proposed approach will be improved to deal with dynamic environments. For this, a method for building a dynamic graph will be proposed too. Furthemore, a chromosome with dynamic length will be incorpored.

7. ACKNOWLEDGMENTS

The authors would like to thank CNPq for their support.

8. REFERENCES

[1] A. L. Nelson, G. J. Barlow, and L. Doitsidis, "Fitness functions in evolutionary robotics: A survey and analysis," *Robot. Auton. Syst.*, vol. 57, pp. 345–370, Apr. 2009.

[2] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.

[3] M. Mitchell, "Genetic algorithms: An overview," *Complexity*, vol. 1, no. 1, pp. 31–39, 1995.

[4] W. Atmar, "On the rules and nature of simulate evolutionary programming," *Proc, of the First Ann. Conf. On Evolutionary Programming*, pp. 17–26, 1992.

[5] D. O. Sales, F. S. Osório, and D. F. Wolf, "Topological autonomous navigation for mobile robots in indoor environments using ann and fsm," in *I CBSEC: Conferência Brasileira em Sistemas Embarcados Críticos*, (São Carlos, Brasil), 2011.

[6] Y. Hu and S. X. Yang, "A knowledge based genetic algorithm for path planning of a mobile robot," in *ICRA'04*, pp. 4350–4355, 2004.

[7] K. Sedighi, K. Ashenayi, T. Manikas, R. Wainwright, and H.-M. Tai, "Autonomous local path planning for a mobile robot using a genetic algorithm," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2, pp. 1338–1345 Vol.2, 2004.

[8] S. C. Yun, V. Ganapathy, and L. O. Chong, "Improved genetic algorithms based optimum path planning for mobile robot," in *Control Automation Robotics Vision (ICARCV), 2010 11th International Conference on*, pp. 1565–1570, 2010.

[9] A. Tuncer and M. Yildirim, "Dynamic path planning of mobile robots with improved genetic algorithm," *Comput. Electr. Eng.*, vol. 38, pp. 1564–1572, Nov. 2012.

[10] R. Kala, "Multi-robot path planning using co-evolutionary genetic programming.," *Expert Syst. Appl.*, vol. 39, no. 3, pp. 3817–3831, 2012.

[11] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[12] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proc. of the Australasian Conf. on Robotics and Automation (ACRA)*, (Sydney, Australia), dec 2005.