Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm

Kamran H. Sedighi, Kaveh Ashenayi, Theodore W. Manikas,
Roger L. Wainwright, Heng-Ming Tai
Electrical Engineering and Computer Science Dept.
University of Tulsa
Tulsa, OK 74104
Email: kash@utulsa.edu

***Abstract*- This paper presents results of our work in development of a genetic algorithm based path-planning algorithm for local obstacle avoidance (local feasible path) of a mobile robot in a given search space. The method tries to find not only a valid path but also an optimal one. The objectives are to minimize the length of the path and the number of turns. The proposed path-planning method allows a free movement of the robot in any direction so that the path-planner can handle complicated search spaces.**

I. Introduction

During the last century, automation has become an extremely fast growing phenomenon impacting almost all facets of everyday life. Recently, robots have become a major part of this trend. Therefore, autonomously navigating robots have become increasingly important [1,3]. Motion planning [2] is one of the important tasks in intelligent control of an autonomous mobile robot [7-9]. The work presented here is part of a larger project to build an autonomous path-planning robot. This research is motivated by earlier work in this field of interest [4-6] by the same research team. This paper presents the research and simulation results of a genetic algorithm based path-planning software. The algorithm uses an improved, modified version of previous encoding techniques [4-6].

II. Robot Path Planning

The path-planning problem is usually defined as follows [2]: "Given a robot and a description of an environment, plan a path between two specific locations. The path must be collision-free (feasible) and satisfy certain optimization criteria." In other words, path planning is generating a collision-free path in an environment with obstacles and optimizing it with respect to some criterion.

The research presented in this paper is part of a project to build an autonomous mobile robot, which can be used as a platform for various applications. This project is divided into three major areas: visual detection of the environment, path planning, and control of the robot. The path-planning component is again divided in two sections: global path planning and local path planning. Running simulations on both global and local path planning in different environments will determine which approach yields the best performance.

A. Global vs. Local Path-Planning

Global path planning requires the environment to be completely known and the terrain should be static. In this approach the algorithm generates a complete path from the start point to the destination point before the robot starts its motion. On the other hand, local path planning means that path planning is done while the robot is moving; in other words, the algorithm is capable of producing a new path in response to environmental changes. Assuming that there are no obstacles in the navigation area, the shortest path between the start point and the end point is a straight line. The robot will proceed along this path until an obstacle is detected. At this point, our path-planning algorithm is utilized to find a feasible path around the obstacle. After avoiding the obstacle, the robot continues to navigate toward the end-point along a straight line (in our system the robot moves in a vertical or horizontal direction, not diagonally; hence, it will try to approximate a straight line) until (1) the robot detects another obstacle or (2) the desired position is reached. An example of local path planning is shown in Figure 1.

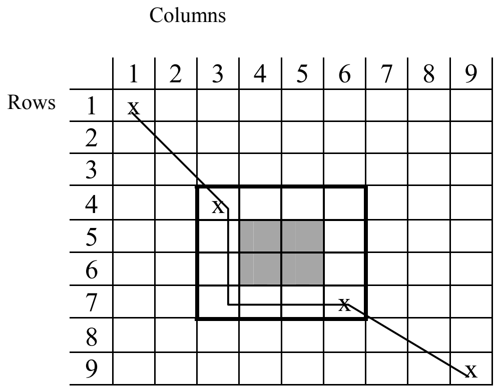


Figure 1. Path-planning example for local obstacle avoidance, applied on a subsection of the search space.

B. Genetic Algorithm Technique for Robot Path Planning

Robot path planning is part of a larger class of problems pertaining to scheduling and routing, and is known to be NP-hard (NP-complete) [10]. Thus, a heuristic optimization approach is recommended as shown by Hwang [11]. One of these approaches is the use of genetic algorithms. A genetic algorithm (GA) is an evolutionary problem solving method, where the solution to a problem evolves after a number of iterations. A proposed solution with the GA method to the path-planning problem is the best feasible path among the pool of all possible solutions.

There have been several contemporary applications of genetic algorithms to the robot navigation problem. One approach is to combine fuzzy logic with genetic algorithms [15, 16, 17]. In this approach, the genotype structure represents fuzzy rules that guide the robot navigation, so the genetic algorithm evolves the best set of rules. While this approach can produce a feasible path through an uncertain environment, the genotype structure becomes very complex, as it needs to represent a variety of fuzzy rules. A complex genotype structure can take a long time to process in a genetic algorithm, which affects the real-time performance of the robot during navigation.

Another approach is to use genotype structures that represent local distance and direction, as opposed to representing an entire path [18, 19, 8, 3]. While these are simple to process and allow for faster real-time performance, the local viewpoint of these methods may not allow the robot to reach its target. Some methods have relatively simple genotype structures that can represent feasible paths, but require complex decoders and fitness functions [20, 2, 9]. This can also affect real-time response.

Simplifying the models used to represent navigation paths will reduce the processing time of the genetic algorithm. Thus, our research has focused on improving the genetic algorithm performance by simplifying the genotype structure.

III. Genetic Algorithm-Based Local Path-Planner Structure

A. Genetic Algorithm

Genetic algorithms [10,12-14] are a class of adaptive methods that can be used to solve search and optimization problems involving large search spaces. The search is performed using the idea of simulated evolution (survival of the fittest). These algorithms maintain and manipulate "generations" of potential solutions or "populations". With each generation, the best solutions (as determined by a problem specific fitness function) are genetically manipulated to form the solution set for the following generation. As in nature, solutions are combined (via crossover) and/or undergo random mutation.

The following are general specifications for our GA-based local path-planning approach:

1) A map of the room in which the path planning takes place is known. The path planner will determine the length and the width of the search space and then apply a grid system to the room, similar to a chessboard. Thus, the room is divided into rows and columns. In our approach we assume the number of rows is equal to the number of columns. The locations of known obstacles are marked as "occupied cells" in the grid.

2) The row and column coordinates of the start-point and the end-point of the desired robot's movement are also known.

3) The robot is allowed to move on all "free" cells, where the center of the robot moves along an imaginary line from the center of one cell to the center of another cell.

B. Different Types of Robot Movement

Assume a robot is required to navigate from the upper-left corner of a room to the lower-right corner, as shown in Figure 1. In order for the robot to do this task, generally, there are two types of robot movements: *Row-Wise* and *Column-Wise*.

B.1) Row-Wised Movement: In a row-based movement, the robot starts moving row by row from the start-point to the end-point. In other words, any horizontal line in the search space will meet the path only once. Therefore, in this movement, the robot always has to go forward and it does not have the capability of going back (up) to the previous row.

B.2) Column-Wised Movement: In a column-based movement, the robot will start moving toward its destination column by column to the right. In other words, any vertical line in the search space will meet the path only once. Therefore, In this movement, the robot always has to move from left to right, and it does not have the capability of moving back to the left.

C. Encoding Technique

The chromosome structure must have sufficient information about the entire path from the start point to the end-point in order to be able to represent it. The previous genotype by a member of our research group, T. Geisler, [4,5] contained only two variables, which will be discussed later, *Path-Location* and *Path-Direction*. That encoding technique allowed only row-wise movements. Next, Aditia Hermanu, another member of our research group, [6] modified the genotype by introducing a new instruction flag for each path, called *Path*-Flag. This *Flag* instructs the next movement type for each step of the movement. Therefore, this genotype allowed the robot to plan either a row-wise or a column-wise movement according to the search space arrangements. But, neither of these two previous structures was able to combine both row-wise and column-wise paths while planning for a single path. This caused the robot to fail for complex environments that required the robot to move both row-wise and column-wise within those search spaces. Thus, the encoding that we have applied in this paper to address the path-planning problem consists of four variables: *Path-Flag*, *Path-Location*, *Path-Direction*, *and Path-Switch.* While the previous work required either a row-wise or column-wise movement, the new genotype is able to plan both row-wise and column-wise within a single search space. Hence, the path has more flexibility to switch between the two movement modes.

C.1) Gene Structure: Path-Flag: The proposed genotype consists of a 1-bit flag for each chromosome. The main responsibility of this bit is to tell the robot whether the next step of the movement is row-wise or column-wise. The value stored at this variable is of type Boolean (0 or 1). If the value is 0, then the decoder considers it as a row-wise instruction; however, if it is 1, then the next step will be considered as a column-wise movement. At the very beginning, this bit tells the robot to start off the first step toward the destination on a row-based, or a column-based movement. During the entire robot movement, the decoder will check this instruction bit before each movement step. The next movement type will be based on the information provided by this flag.

C.2) Gene Structure: Path-Location: The encoding technique uses the information of a gene's position, as well as the value stored at that position as a y (x)- and an x (y)-coordinate, depending on the instruction flag (0 = row-wise or 1 = column-wise). These coordinates define the location of the robot within the search space. For example, if the robot is required to go row-wise (Path-Flag = 0), a gene's position within a chromosome corresponds to a row-number (y- coordinate). The value, stored in a gene, in a variable called *path-location*, corresponds to a column-number (x-coordinate).

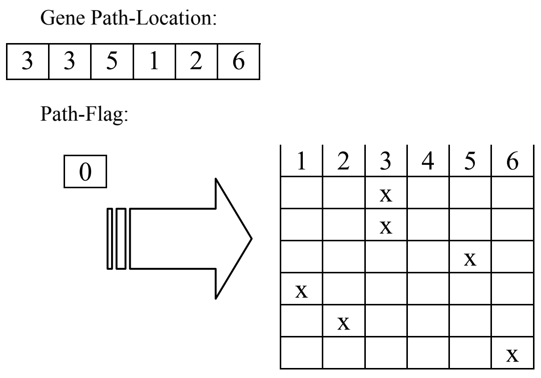


Figure 2. Example of Path-Location for row-wise movement

On the other hand, for the column-wise movement (Path-Flag = 1), a gene's position within a chromosome corresponds to a column-number (x-coordinate). Then, the value stored in that gene corresponds to a row-number (y-coordinate). Therefore, path-location is an integer variable whose value is in the range of 1 to the total number of rows or columns of the scanned environment. The length of all chromosomes is therefore also known and fixed, since the total number of rows or columns directly corresponds to the total number of genes within a chromosome. Since these numbers are always equal, the chromosome's length stays the same independent of the nature of the next step, row- wise or column-wise. Using this approach, we can avoid using varying-length chromosomes, which is more difficult to implement. For instance, for an n x n grid system, the chromosome's length would be n, and the value stored in each gene would be an integer number between 1 and n. A complete chromosome therefore represents a 'path' from a cell in the first row to a cell in the last row (for a row-wise movement) or a 'path' from a cell in the first column to a cell in the last column (for a column-wise movement), as shown in Figures 2 and 3

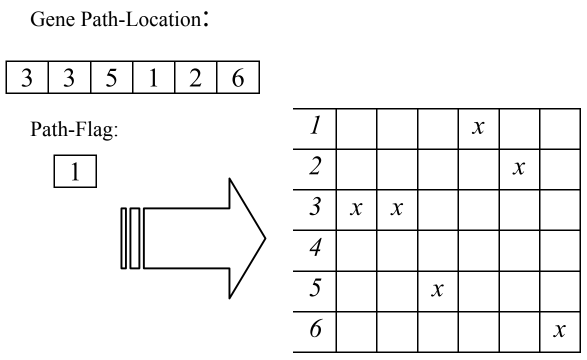


Figure 3. Example of Path-Location for column-wise movement

C.3) Gene Structure: Path-Direction: The gene structure described so far only represents vertices ('corner points' or 'intermediate steps') of a path. To send a robot on a straight line directly from a center of one vertex to the center of the next vertex would mean that the robot moves on a diagonal line across many adjacent cells. This could cause problems if not all adjacent cells that the robot is to traverse going from one cell to the next are free of obstacles, as shown in Figure 4.

A better approach is to go to the side (horizontal) first, turn, and then go down (vertical), or vice versa. To indicate the first direction the robot will turn to proceed to the next vertex, a second variable called Path-Direction is added to the gene structure. Direction is a two-state variable (Boolean), which has either the value 1 or 0 for horizontal or vertical directions respectively. The length of the *direction* array is one less than the length of the *location* array, since there is no direction instruction for the last location.

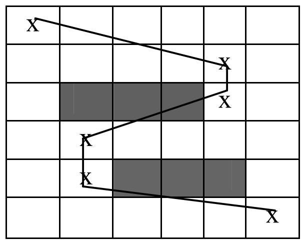


Figure 4. Problem with diagonal movement of the robot

Figure 5 shows the same search space as Figure 4. Now the connection, and therefore the path, from one vertex to the next one is not a diagonal line, but a combination of a horizontal / vertical movement. Since the first direction that the robot turns to can be either horizontal (solid line) or vertical (dotted line), there are two possible ways to get from one vertex to the next one for each step. The introduced variable *Path-direction* indicates which of the two ways the robot will use to go to the next vertex.

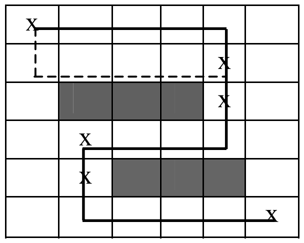


Figure 5. The path with horizontal / vertical instead of diagonal movement

It is obvious in Figure 5, one exception where the *path-direction* variable will not affect the robot movement direction is when the two consecutive movement steps are either in the same column (for the row-wise movement) or in the same row (for column-wise movement). In either case, there is only one way to go from one vertex to the next one, which is a straight horizontal or vertical line.

C.4) Gene Structure: Path-Switch: As discussed earlier, the previous work by a member of our group, T. Geisler [4,5], only allowed row-wise movements. Next, Aditia Hermanu [6] modified the genotype so that the robot was able to plan either a row-wise or a column-wise movement according to the search space arrangements. But, none of the previous research was able to combine both row-wise and column-wise paths while planning for a single path. This caused the robot to fail for complex environments that required the robot to move both row-wise and column-wise within those search spaces. For example, consider Figure 6 with two different search space environments.

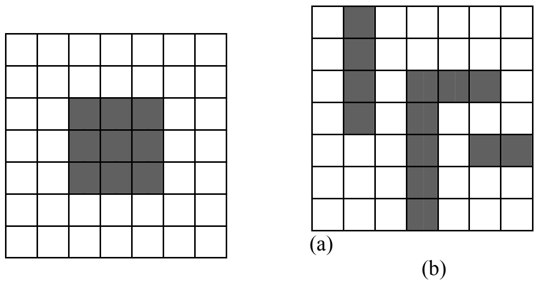


Figure 6. Examples of (a) an easy search space; (b) complex search space

Notice from the environment in Figure 6a, either row-wise or column-wise movement can address this problem since it does not have a complex obstacle arrangement. On the other hand, the environment in Figure 6b requires both row-wise and column-wise movement in order to be traversed. Therefore, it is considered a more complex search space compared to the search space shown in Figure 6a. Our previous research was not able to address this type of environment.

In order to overcome this movement restriction, we added the fourth variable, called *Path-Switch,* to the genotype. This variable enables the robot to switch back and forth between a row-wise (r.w.) and a column-wise (c.w.) movement in a single path. This array contains two switching numbers. Therefore, the robot can switch a maximum of two times from row-wise to column-wise and vice versa within a search space. The values that are stored in this array are integers and are in the range of 1 to the total length or width of the environment. The numbers stored in this array indicate the location where the robot has to switch from r.w. to c.w. movement or vice versa.

Following summarizes the points on the switching array:

This array always contains 2 switching numbers, which does not necessarily mean that we always have 2 switching points, as illustrated later.

The switching numbers could be any number from 1 to the total number of the search space rows or columns.

The switching numbers are integer type.

The number stored in each switching point indicates the location of the gene in which the robot has to switch. For instance, switching numbers 2,5 means the robot is switching two times: first at gene locations 2, then at location 5.

For n x n environment, path-switch i,j translates into:

- ➢ No switching points if i=j=n
- ➢ One switching point if i j=n

- One switching point if i=j n
- Two switching points if i j n

IV. GA Elements

A. Fitness Evaluation

The population of paths is evaluated during each generation. The evaluation is based on the paths' fitness, which depends on how suitable the solution (path) is according to the problem. In preliminary evaluation, the values for the path length, the number of turns, and the number of infeasible steps (collisions) are determined for each path in the population. The reason for considering the number of turns is that the total number of turns has a direct impact on the overall time needed for the robot to travel from the starting point to the end-point. If the path has numerous turns, the robot has to slow down for each turn; therefore, the total movement time will increase. These values are set in relation to the entire population and therefore are represented as fractional values from 0 to 1, where 1 indicates the optimal fitness value. The shortest path length in the population corresponds to length-fitness $f_{Length} = 1.0$; the longest path in the population corresponds to $f_{Length} = 0$. The greatest number of infeasible steps (collisions) corresponds to $f_{collisions} = 0$; the least number of infeasible steps corresponds to $f_{collisions} = 1.0$ and same for $f_{numberofTurns}$. f_{Length} is the fitness value associated to the path length, $f_{collisions}$ is the fitness value associated with the number of infeasible steps (collisions), and $f_{numberofTurns}$ is the fitness value associated with the number of turns in the path.

An attempt was made to weigh each part of a path's fitness (length, number of infeasible steps, and number of turns) according to its importance to the algorithm objective. A path's most important feature is the number of infeasible steps, and it should therefore have the biggest influence on the path's fitness. Thus, the other two features will be multiplied by this fitness value. Recall that all three values are fractions between 0 and 1. The other two parts of the path's fitness are multiplied with a weighing factor (the value of these factors are determined through experimentation) and added. This sum will be divided by the sum of the weighing factors in order to end up with an over all fitness value between 0 and 1. Finally, the entire fitness function will be multiplied by 100 to end up with fitness values between 0 and 100, which is necessary for the selection process.

$$f_{path} = f_{collisions} \,.\, [L.\, f_{Length} + T.\, f_{numberofTurns}] . \frac{100}{L + T} \qquad (1)$$

f_{path} is the fitness value for the entire path, L is the weighing factor for f_{Length} and T is the weighing factor for $f_{numberofTurns}$. For our GA, weighing factor L was set to 1and weighing factor T was set to 2 to emphasize the number of turns over the path length. Thus by penalizing the turn fitness more than the length fitness, we reduce the total number of turns by taking the risk of having a longer path. This decision underlies the assumption that a turn is more time intensive than some extra steps, since a robot needs to slow down and stop for a turn, and accelerate again afterwards.

A penalty has also been applied for a solution that contains any infeasible steps to emphasize the importance of all steps needing to be feasible for a usable result path. Based on these criteria, the fitness function is modified as follows for those solutions that contain infeasible steps (collisions) and will be applied in the proposed path-planning algorithm:

$$f_{path} = 0.1 \times \frac{fpath}{(number\ of\ Collisions)^2} \qquad (2)$$

After the fitness value for all chromosomes in the population have been computed, Rank Selection [10] is used to determine the parent chromosomes that will be used for reproduction.

B. Crossover Process

For our GA, two parent chromosomes are combined applying a single-cross-point, value encoding crossover [10]. The crossover operator has been modified to produce two offspring chromosomes with each crossover operation. This is achieved by using the gene information, which were not used to build offspring one, in order to build a second chromosome. Furthermore, the crossover operation acts only on the location and the direction (with the instruction flag included) parts of the chromosomes, as it does not make sense to have single point crossover operator acting on the two switching points. The function of the crossover operator is illustrated in Figure 7.

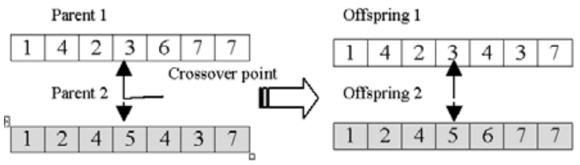


Figure 7. Single-cross-point, value-encoding crossover

C. Mutation

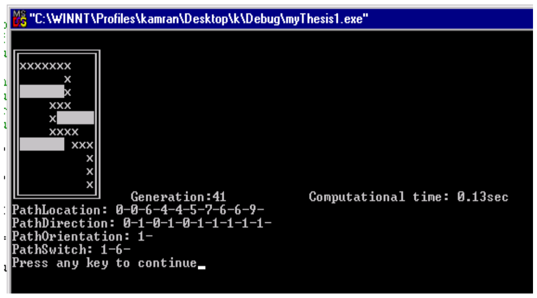
For mutation [10], almost every operation that changes the order of genes within a chromosome or that changes a gene's value (such as *location* or *direction)* is a valid mutation operator.

The mutation operator used here checks with the mutation probability for every single gene and decides whether it should be mutated or not. If a gene is to be mutated, a random number between 1 and the total number of rows or columns in the search space is assigned to *location,* and a random direction, either 1 or 0, is assigned to *direction* and to the *instruction flag.* Unlike the crossover operator that does not act on the switching points, mutation operator will affect these points. Using this mutation method, theoretically, every gene within a chromosome could be mutated and thus the chromosome totally changed.

Elitism [10] was also used in order to keep the best individual (path) within a generation. If elitism is applied, the fittest chromosome path is copied to the offspring population without any changes.

V. Simulation Results

After considerable testing, we determined the best operators and parameters for our GA. Path-planning simulations were conducted on different sized search spaces with different obstacle configurations.



```
"C:\WINNT\Profiles\kamran\Desktop\k\Debug\myThesis1.exe"

xxxxxxx
      x
      x
    xxx
    x
    xxxx
       xxx
         x
         x
         x
                Generation:41           Computational time: 0.13sec
PathLocation: 0-0-6-4-4-5-7-6-6-9-
PathDirection: 0-1-0-1-0-1-1-1-1-1-
PathOrientation: 1-
PathSwitch: 1-6-
Press any key to continue_
```

Figure 8. Example search space (SPSet01) for path-planning simulations

Figure 8 shows an example search space. It is obvious this search space is very easy and can be navigated by previous methods [4-6] via a row–wise movement as well.

The proposed path-planning GA was tested on eight different search spaces (SPSet01 ~ SPSet08) and the results were compared against those yielded by the Geisler and Hermanu's path-planning GA [4-6]. For each set of tests, the GAs were run 15 times and the average success rates were calculated and are shown in Table 1.

Search spaces 07 and 08 were the two that required both row-wise and column-wise movement of the robot in order for the robot to get around the obstacles. As shown in Table 1, only the genotype developed in this research can address these search spaces as previous genotypes failed to handle these environments. This is because the new genotype allows the robot to switch back and forth from row-wise to column-wise movement and vice versa, which means a free movement of the robot in any direction. Figure 9 shows two search spaces and compares the paths evolved by the proposed GA in this paper with the one developed by Hermanu's.

TABLE 1. SUCCESS RATES COMPARISON BETWEEN THE PROPOSED GA AND THE PREVIOUS METHODS

Search space	Success Rate (%)		
	Geisler's GA	Hermanu's GA	Proposed GA
SPSet01	100	93.3	93.3
SPSet02	0.00	86.7	100
SPSet03	73	86.7	100
SPSet04	53	80	100
SPSet05	0.00	100	100
SPSet06	0.00	20	100
SPSet07	0.00	0.00	86.7
SPSet08	0.00	0.00	73.3

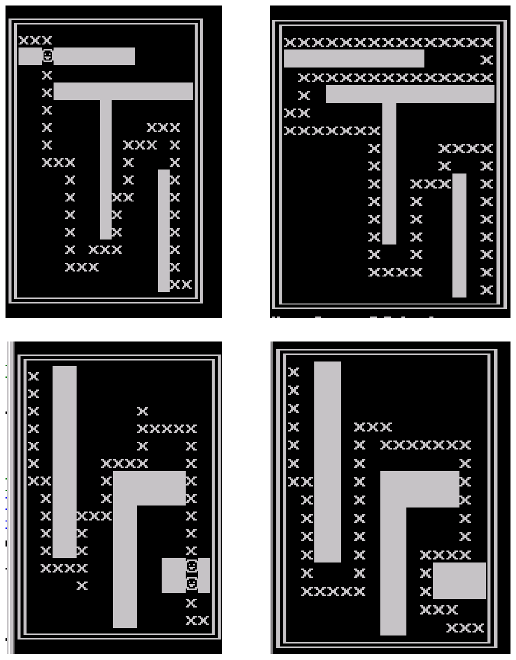
Search space SPSet01, as shown in Figure 8, is the search space with obstacle configuration that can be addressed only by a row-wise movement. As demonstrated in Table 1, the proposed GA is able to address this environment as well. The little difference in the success rates between the new genotype and the previous ones for this particular environment is because the previous genotypes were tuned specifically to address these types of search spaces. This differs from the new genotype that has been developed to address all types of environments with different obstacle arrangements. Figure 10 shows the resulted paths generated by our new path-planning GA on different search spaces.

VI. Conclusion

Our path-planning genetic algorithm yields the best performance on the search spaces that can be addressed with two switching points or less. This meets the needs of local path planning, which deals only with part of the room around obstacles. For global path planning, a genotype that is able to address the more complicated search spaces (entire room) should be considered.

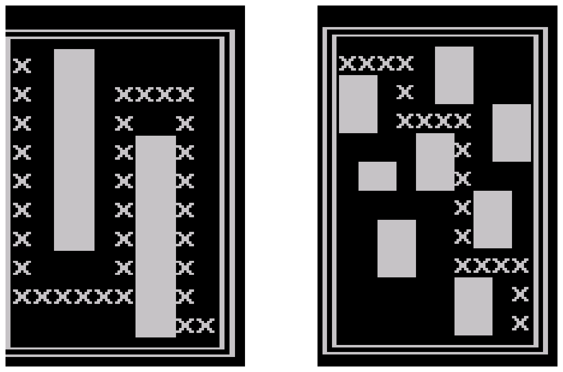
Future work includes developing a method to address global path planning, comparing the results against those presented in this paper for local path planning, and determining which method yields the best performance.

Finally, the path-planning software will be incorporated into the overall robot control software and the algorithm performance will be tested on an actual robot.

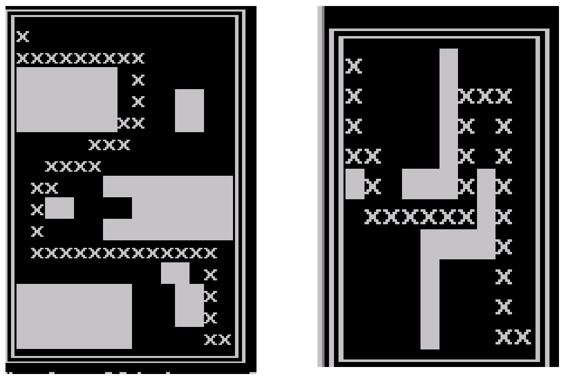


(a) (b)

Figure 9. Resulted paths for SPSet07 and SPSet08 (a) Proposed GA (b) Previous method.



SPSet02 SPSet04



SPSet05 SPSet06

Figure 10. Examples of the resulted paths on different search spaces

REFERENCES

[1] *Farritor, S. and Dubowsky, S., "A Genetic Algorithm Based Navigation and Planning Methodology for Planetary Robot Exploration*", Proceedings of the 7th American Nuclear Society Conference on Robotics and Remote Systms, Augusta, GA, 1997

[2] *Sugihara, K. and Smith, J., "Genetic Algorithms for Adaptive Motion Planning of an autonomous Mobile Robot*", Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation, Monterey, CA, pp. 138-146, 1997

[3] *Vadakkepat, P. and Chen, T.K., "Evolutionary Artificial Potential Fields and Their Application in Real Time Robot Path Planning*", Proceeding of the 2000 Congress on Evolutionary Computation, San Diego, CA, pp. 256-264, 2000

[4] *Geisler, T. and Manikas, T., "Autonomous Robot Navigation System Using a Novel Value Encoded Genetic Algorithm*", Proceeding of IEEE Midwest Symposium on Circuits and Systems, Tulsa, OK, 2002.

[5] *Geisler, T., "Autonomous Robot Navigation System Using A Genetic Algorithm with a Novel Value Encoding Technique*", Master's Thesis, The University of Tulsa, OK, 2002.

[6] *Hermanu, A., "Genetic Algorithm with Modified Novel Value Encoding Technique for Autonomous Robot Navigation*", Master's Thesis, The University of Tulsa, OK, 2002

[7] *Fogel, D.B., "What is evolutionary computation*?", IEEE Spectrum, pp. 26-32, February 2000

[8] *Gallardo, D. and Colomina, O., "A Genetic Algorithm for Robust Motion Planing", Eleventh* International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Castellon, Spain, June, pp. 115-121, 1998

[9] *Xiao, J. and Zhang, L., "Adaptive Evolutionary Planner/Navigator for Mobile Robots*", IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 18-28, April 1997

[10] *Obitko, M., "Genetic Algorithms*", Internet publication, 1998, http://cs.felk.cvut.cz/~xobitko/ga/main.html

[11] *Hwang, Y.K., Ahuja, N., "Gross Motion Planning – A Survey*", ACM Computing Surveys, volume 24, issue 3, pp. 219-291, September 1992

[12] *Trivedi, N., Lai, W. and Zhang, Z., "Optimizing Windows Layout by Applying a Genetic Algorithm*", Proceedings of the 2001 Congress on Evolutionary Computation, Seoul, Korea, pp. 431-435, 2001

[13] *Filho, J.L.R. and Treleaven, P.C., "Genetic Algorithm Programming Environment*", IEEE Computer, pp. 28-43, June 1994

[14] *Srinivas, M. and Patnaik, L.M., "Genetic Algorithms: A survey*", IEEE computer, pp. 17-26, June 1994

[15] *Arsene, C.T.C. and Zalzala, A.M.S., "Control of Autonomous Robots Using Fuzzy Logic Controllers Tuned by Genetic Algorithms*", Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), pp. 428-435, 1999

[16] *Kubota, N., Morioka, T., Kojima, F. and Fukuda, T., "Perception-Based Genetic Algorithm for a Mobile Robot with Fuzzy Controllers*", Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), pp. 397-404, 1999

[17] *Pratihar, D.K., Deb, K. and Ghosh, A. "Fuzzy-Genetic Algorithms and Mobile Robot Navigation among Static Obstacles*", Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), pp. 327-334, 1999

[18] *Cazangi, R.R. and Figuieredo, M., "Simultaneous Emergence of Conflicting Basic Behaviors and Their Coordination in an Evolutionary Autonomous Navigation System*", Proc. 2002 IEEE Conf. on Evol. Comp. (CEC '02), IEEE, 2002

[19] *Di Gesu, V., Lenzitti, B., Lo Bosco, G. and Tegolo, D., "A Distributed Architecture for Autonomous Navigation of Robots*", Proceedings Fifth IEEE International Workshop on Computer Architectures for Machine Perception, pp. 190 - 194, 2000.

[20] *Hocaoglu, C. and Sanderson, A.C., "Planning Multiple Paths with Evolutionary Speciation"*, IEEE Trans. Evolutionary Computation, vol. 5, no. 3, pp. 169-191, June 2001.