A Two-Phase Navigation System for Mobile Robots in Dynamic Environments

Tsai-Yu Chang, Szu-Wen Kuo and Jane Yung-jen Hsu

Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 106, R.O.C.
yjhsu@csie.ntu.edu.tw

Abstract

This paper presents an implemented navigation system for mobile robots in dynamic environments. In order to take advantage of existing knowledge of the world and to deal with unknown obstacles in real time, our system divides motion planning into global path planning and local reactive navigation. The former uses genetic algorithm methods to find a collision-free path; the latter is implemented using neural network techniques to track the path generated by the global planner while avoiding unknown obstacles on the way. As a result, the system can adapt to dynamic environmental changes. Our experiments, both in simulation and on a real robot, showed that the system can find a reasonably good free path in a fraction of the time necessary to find an optimal free path, and it can effectively achieve its goal configurations without collision.

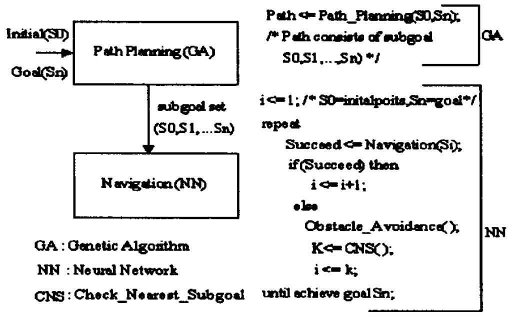
1 Introduction

A mobile robot accomplishes tasks by moving in the real world. *Motion planning*, namely, deciding what motions to perform in order to achieve goal arrangements of physical objects, is one of the most important capabilities for a mobile robot. Although it may seem like a relatively simple job for humans, motion planning requires sophisticated integration of reasoning, perception and control. Many techniques for performing path planning and navigation have been developed over the years. On the one hand, most planning methods assume complete knowledge of the environment, and they emphasize on finding the optimal free paths [14]. On the other hand, navigation algorithms often assume that the world is completely unknown, and they focus on guaranteeing goal achievement and obstacle avoidance [15; 10]. There are several existing navigation algorithms such as [15] that are complete, i.e. they guarantee to find a collision-free path in unknown environments if such paths exist. However, the resulting paths usually involve much extra tracking of the obstacles since prior knowledge about the obstacles were not taken into consideration.

In reality, a mobile robot operates in a world with both static and dynamic properties. In order to take advantage of exiting knowledge of the world and to deal with unknown obstacles in real time, our system divides motion planning into ***global path planning*** and ***local reactive navigation.*** The former finds a collision free path; the latter tracks the path generated by the global planner while avoiding unknown obstacles on the way. Due to the dynamic nature of the world, it is meaningless to spend too much time finding an optimal path that will be abandoned whenever any unknown obstacle is encountered. The global planner should search for a feasible path efficiently, and then incrementally refines and optimizes the path if time permits. Genetic algorithm was chosen for the global path planner, due to its ability to generate a reasonably good path in a fraction of the time necessary to find an optimal path based on the visibility graph approach. The reactive navigation module was implemented as a neural network that was trained using examples from manually guiding a simulated robot and from running a hand-coded navigation and collision-avoidance algorithm. As a result, the system can effectively adapt to dynamic environmental changes.

2 System Architecture

The overall architecture of the two-phase navigation system is shown in Figure 1. Given an arbitrary initial point and any goal point, the system will start by calling the genetic path planner. The initial collision-free path, consisting of several subgoal points, is then passed to the navigation module that controls the robot to move along the planned path. If there is any unknown obstacle in the way while executing the task, the navigation controller will react to the environment based on its sensor readings in order to avoid collision. When the obstruction is cleared, the system will continue to achieve the task by moving to the nearest subgoal point on the originally planned path.



```
Path <= Path_Planning(S0,Sn);
/* Path consists of subgoal
   S0,S1,...,Sn) */

i <= 1; /* S0=initalpoits,Sn=goal*/
repeat
     Succeed <= Navigation(Si);
     if(Succeed) then
         i <= i+1;
      else
         Obstacle_Avoidence();
          K <= CNS();
          i <= k;
until achieve goal Sn;
```

Figure 1: Two-Phase Navigation System Architecture

3 Global Genetic Path Planner

There are many path planning methods that can generate a collision-free path for a robot to move from its initial position to its goal position on a given map [7]. One of the earliest methods is ***visibility graph***, which has been widely used for mobile robots [14]. The principle idea is to construct a semi-free path composed of line segments connecting the initial point to the goal through vertices of obstacles. Constructing the visibility graph requires $O(n^3)$ time, where n is the total number of vertices of obstacles [9]. The standard A* algorithm, with the Euclidean distance as the heuristic evaluation function, can guarantee finding the shortest path if one exists. However, finding the optimal path may be unnecessary if a robot has to modify its path in order to avoid collisions with unknown obstacles. Insteading of spending a lot of time finding the shortest path that will be changed later, it is a better idea to find a reasonable path quickly in a dynamic environment.

Genetic algorithms are an effective parallel search technique [3], and it can be used to find a collision-free path on a visibility graph within a relatively short time. In addition, given more time, a genetic algorithm can refine the current solutions incrementally in search of the optimal solution. The proposed architecture uses a modified SGA (simple genetic algorithm) that can manipulate variable-length strings necessary to represent robot paths in the search process.

3.1 Genetic representation

Given a map containing all known objects in the environment, the planner assigns a number to each vertex of the obstacles, the initial point, and the goal point. A path consists of a sequence of vertices from the starting point to the end point. Each chromosome corresponds to a path represented by a sequence of numbers. Variable-length strings are used to represent chromosomes in order to deal with paths of arbitrary lengths. A population is then defined as a set of variable-length strings. For example, "p1: 1 8 6 5 15" denotes a chromosome representing a path from point 1 to point 15 through vertices 8, 6, and 5.

3.2 Reproduction operators

Two reproduction operators, crossover and mutation, are used by the genetic planner.

Crossover Given two parent strings from the current population, crossover is achieved by choosing a crossover point (denoted by ⊗) and then swapping the substrings from both parents about the chosen point. Since the lengths of any two parent strings may be different, the crossover point is chosen based on the shorter string. By swapping the substrings of numbers following the crossover point, two new strings are created. A number may appear more than once in a new string created this way, indicating a loop in the corresponding path. In this case, the redundant numbers are removed. For example, consider two parent strings A1 and A2:

```
parent A1:  1 8 6 ⊗ 5 7 15
parent A2:  1 4 3 ⊗ 6 15
child C1  : 1 8 6 15
child C2  : 1 4 3 5 7 15
```

Mutation Three mutation operators are defined: `replace`, `add` and `delete`. The replace operator updates a number with a new random number as defined in SGA. The add and delete operators are specially designed for this application. Add inserts the number corresponding to an arbitrary vertex into a randomly selected place; `delete` removes a randomly selected number except the first and last numbers. As a result, new individuals of variable length can be generated. The results of the mutation operators on a chromosome (the point at which mutations take place is indicated by $\star$) are shown below:

```
Current : 1 8 6 ⋆ 5 7 15
Replace: 1 8 6 2 7 15
Add     : 1 8 6 4 5 7 15
Delete : 1 8 6 7 15
```

3.3 Natural selection

The genetic planner uses a combination of the standard roulette wheel parent selection technique and the elitist strategy. The random selection technique dictates that an individual chromosome's chance of being selected is proportional to its fitness. On the other hand, the elitist strategy copies the fittest individual(s) into the new generation in order to avoid losing the best chromosome(s) of the old generation by accident.

3.4 Fitness function

There are two criteria for deciding the quality of a candidate path: the percentage of free segments and the overall path length. Given two vertices v_i and v_j, the path segment between them is free if and only if the straight line $\overline{v_i v_j}$ does not intersect with any obstacles, and the path length is defined as the Euclidean distance between v_i and v_j. The Euclidean distance of a chromosome is the summation over path length of every segment in the path.

Given a chromosome x in the population, its fitness evaluation function is defined as the weighted sum of these two criteria as follows:

$$f(x) = w \cdot \frac{p(x)}{P_{max}} + (1 - w) \cdot \frac{D_{max} - d(x)}{D_{max}}$$

where $p(x)$ is the percentage of free segments in x, P_{max} is $\max_i p(x_i)$ in the current population, $d(x)$ is the Euclidean distance of the chromosome x, and D_{max} is $\max_i d(x_i)$ in the current population. Maximizing fitness therefore entails finding individuals with freest path and shortest path length. Tuning the relative weight of these two criteria plays an important role in finding a good quality path within a short time. At the beginning, the weight is set to be high in order to gather more feasible paths in the population. Once a free path has been found, the weight is decreased gradually so that the search will focus more on finding the shortest paths. The strategy is based on two observations. First, the path between two vertices that are closest together is not always collision-free. Secondly, once a population contains enough free paths, future generations will continue to be mostly free. The process terminates when the relative error of two generations is below a threshold value. If the fitness of the resulting population is not satisfactory, i.e. a local maximum, the system should increase the mutation rate and continue the process.

3.5 Tabling

In general, the cost of path planning is dominated by the cost of constructing the visibility graph, which requires $O(n^3)$ in a straightforward implementation [9]. Given the assumption that the world may change dynamically, the genetic planner is only interested in finding a good path rather than the optimal path. It is therefore not necessary to construct the complete visibility graph in advance. In the proposed architecture, the visibility graph is generated incrementally at run time. The freeness and distance between any pair of vertices are recorded in two separate tables in order to reduce repeated computation.

3.6 Performance analysis

The performance of the genetic path planner (GA) was compared with that of the visibility graph using A* search method (VG). Two criteria were evaluated: the *time* to find a path and the *distance* of the resulting path. The results are shown in Table 1. Each column of the table records the average performance of VG and GA over ten randomly generated maps, each with n numbers of obstacles. The experiment was repeated for n=10,12,14,16,18,20. In comparison with GA, the time required by GV grows rapidly as the number of obstacles increases, while the resulting distance is only slightly shorter than that of GA. As a result, GA can better cope with real-time requirements when the number of obstacles is large.

Table 1: Performance Comparison

* Time is measured in 0.01 second
* Distance is measured in 0.1 inch

	n	10	12	14	16	18	20
VG	Time	102.20	148.20	242.00	329.40	469.00	609.80
	Distance	1871.50	1878.20	2223.20	1444.40	1558.60	1940.60
GA	Time	68.63	85.61	139.86	160.08	167.58	212.05
	Distance	1955.77	1946.53	2302.23	1521.29	1612.29	2073.72

4 Local NN Navigation System

The local navigation module extracts features from raw sensor data. The processed data are then used as input patterns to the neural network controller that selects an appropriate action for the robot to perform in response to the current sensor readings. Figure 2 shows the overall operation cycle.

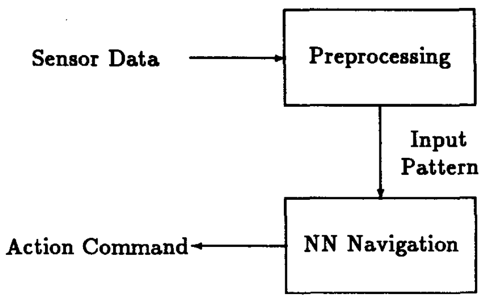


Figure 2: Operation cycle of the NN Controller

4.1 Robot hardware configuration

The Nomad 200 mobile robot used in our experiments is shown in Figure 3. It is composed of a circular omni-directional three-wheeled base and four sensory modules including infrared, ultrasonic, and tactile sensors, as well as a 2D laser ranger.

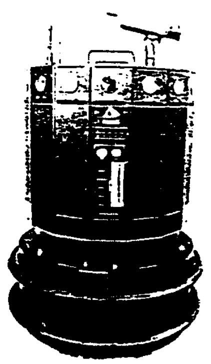


Figure 3: The Nomad 200 mobile robot.

In this research, only the first three modules were used. The infrared module consists of a ring of 16 sensors each of which returns a value from 0 to 30 inches. The ultrason module consists of a ring of 16 sensors each of which returns a value from 17 to 255 inches. The sensor data is returned through an array variable – `State`. The size of `State` is 43 with the infrared data being stored in `State[1]` to `State[16]` and the sonar data being stored in `State[17]` to `State[32]`. Figure 4 illustrates the organization of the infrared and sonar regions on the robot. In addition, the current position coordinates x, y can be

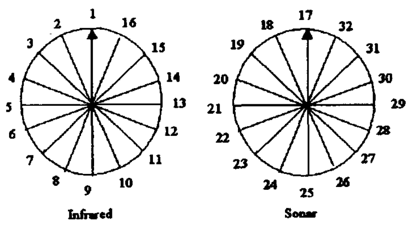


Figure 4: Sensor organization

found in `State[34]` and `State[35]`, the steering angle of the robot is in `State[36]`, and the bumper state is in `State[33]` (`State[33]` $\neq 0$ indicates that the robot has collided with some objects.).

4.2 Preprocessing of sensor data

Due to the large amount of sensor data, it is necessary to preprocess the data in order to minimize network topology as well as training time. The mobile robot uses infrared for short-range object detection and sonar for long-range object detection, which complement each other. Based on the circular arrangement of the sensors, obstacle avoidance and boundary tracing were implemented by segmenting the space around the robot into relevant sensory regions (see Figure 5). The area in front of the robot was divided

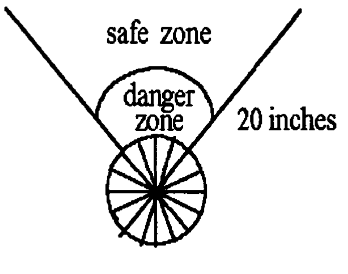


Figure 5: Classification of sensor regions

into two regions: the *danger* zone and the *safety* zone. The zone dividing threshold of 20 inches was derived from the robot's velocity and the minimum sonar range of 17 inches.

It guarantees that at least a subset of sensor data will be available before reaching an obstacle, thus decreasing the probability of any collision.

There are a total of 10 inputs to the neural network controller. Each input indicates a specific feature in the environment. Each input can be defined by a combination of sonar, infrared and bumper data in the following way.

Front_safe_by_sonar_detection: This input feature indicates if the front is clear based on sonar sensors. That is,

$$I_1 = \begin{cases} 0 & \text{if } \min_{i=17,18,19,31,32}(\texttt{State}[i]) < 20 \\ 1 & \text{otherwise} \end{cases}$$

Front_safe_by_infrared_detection: This input feature indicates that the front is clear based on infrared sensors. That is,

$$I_2 = \begin{cases} 0 & \text{if } \min_{i=1,2,3,15,16}(\texttt{State}[i]) < 20 \text{ or} \\ & \quad \min_{i=4,14}(\texttt{State}[i]) \leq 6 \\ 1 & \text{otherwise} \end{cases}$$

The second condition for the "0" case is necessary because the robot is not a point robot, and it may bump into obstacles on the side even if the front sensors do not detect any obstacle.

Obstacle_in_right_danger_zone
Obstacle_in_left_danger_zone: Intuitively, the mobile robot should move toward areas with fewer obstacles. When obstacle are detected by the front sensors, we must check whether they fall more within the left side or the right side of the danger zone. We define two scores to estimate the distance to obstacle(s) on the left and right respectively:

$$L = \min_{19,20,21}(\texttt{State}[i]) + \min_{22,23,24}(\texttt{State}[i]);$$
$$R = \min_{29,30,31}(\texttt{State}[i]) + \min_{26,27,28}(\texttt{State}[i]).$$

In addition, goal attraction can be taken into consideration. If the goal is toward the right side of the robot, then increment R; otherwise increment L. The inputs are then defined as:

$$I_3 = \begin{cases} 0 & \text{if } R \geq L; \\ 1 & \text{otherwise} \end{cases}$$

$$I_4 = \begin{cases} 0 & \text{if } L > R; \\ 1 & \text{otherwise} \end{cases}$$

Parallel_to_right_obstacle
Parallel_to_left_obstacle: These two features describe if the robot is parallel to some obstacle on either side. Either input is on if the robot is in potential danger of moving into an obstacle, as is shown in Figure 6. The input features can be defined as

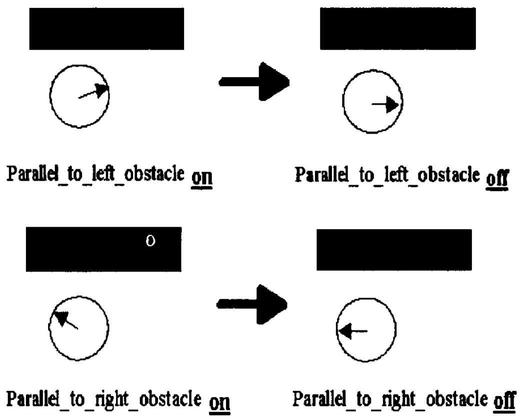

Parallel_to_left_obstacle on
Parallel_to_left_obstacle off
0
Parallel_to_right_obstacle on
Parallel_to_right_obstacle off

Figure 6: Parallel to obstacle

$$I_5 = \begin{cases} 1 & \text{if } \min_{i=12,13,14}(\texttt{State}[i]) < 20 \text{ and} \\ & \quad \min_{i=9,10,11}(\texttt{State}[i]) \geq 20 \\ 0 & \text{otherwise} \end{cases}$$

$$I_6 = \begin{cases} 1 & \text{if } \min_{i=4,5,6}(\texttt{State}[i]) < 20 \text{ and} \\ & \quad \min_{i=7,8,9}(\texttt{State}[i]) \geq 20 \\ 0 & \text{otherwise} \end{cases}$$

Away_from_right_obstacle
Away_from_left_obstacle: These two input features are used to signal that the obstacle on the right (left) side is no longer blocking the way. The situation is shown in Figure 7. After getting away from the obstacles, the system should check if the distance to goal has increased since obstacle avoidance last started. If so, the robot should stroll along the obstacles until the distance to goal is shorter than or equal to the original distance. Input 7 references sonar 19, 20, 21 and infrared 6, 7, 8; input 8 references sonar 29, 30, 31 and infrared 10, 11, 12.

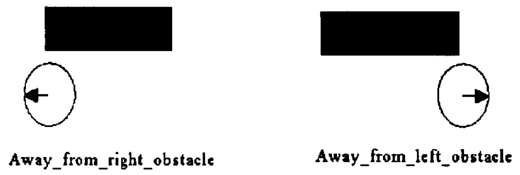


Figure 7: Away from obstacle

Bumper: This input, which references the tactile sensor data in `State[33]`, is set to be on whenever the robot has bumped into any obstacles. Otherwise, it is off.

Direction: When the robot has rotated to the direction of the goal, this input becomes on. Otherwise, it remains off.

4.3 Network topology

The local navigation controller was implemented as a three-layer neural network with one hidden layer. The topology is shown in Figure 8. Based on the input definitions in the previous section, there are 10 input nodes, 17 hidden nodes and 6 output nodes. Each node of the output layer dictates a specific action for the robot to perform. Multiple output nodes may be on at the same time. The six output actions of the navigation controller are explained below:

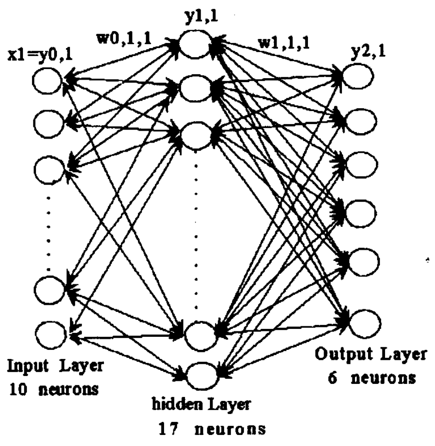


Figure 8: The topology of navigation controller

1. Orient_to_goal:
   Rotate to the direction of the goal.
2. Move_forward:
   Move in the current direction for a fixed distance.
3. Move_backward:
   Move opposite the current direction for a fixed distance.
4. Turn_Left:
   Rotate counter-clockwise for a fixed angle.
5. Turn_Right:
   Rotate clockwise for a fixed angle.
6. Stop: Terminate all actions.

4.4 The training process

A set of 60 training examples were collected automatically by operating the mobile robot in the simulator using a hand-coded collision avoidance program. The standard backpropagation method is used to train the navigation controller. The learning algorithm from [12; 4; 8] was used:

$$W(t+1) = W(t) - \epsilon\frac{\partial E}{\partial W(t)} + \alpha(W(t) - W(t-1))$$

where

$W(t+1)$	: weight at time $t+1$;
$W(t)$	: weight at time t;
$W(t-1)$	: weight at time $t-1$;
E	: sum of square error;
ϵ	: learning rate;
α	: momentum term.

The learning rate and the momentum term were fixed in the entire training process. The number of hidden layers and nodes were determined through a series of experiments. Once the topology was designed, the training process took approximately 12 hours on a PC 386. The weights converged to a mean square error of less than 0.011 after about training 15000 cycles (60 patterns for each cycle). Figure 9 shows how the mean square error converged during the training process.

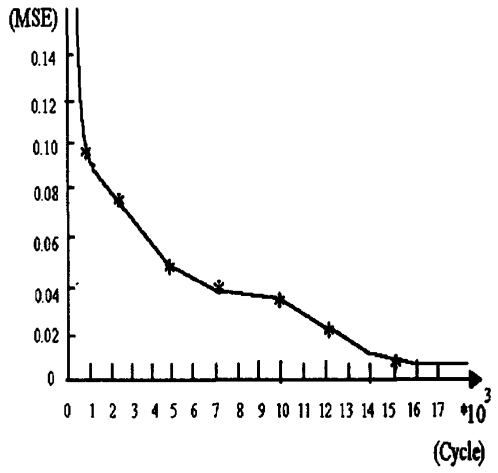


Figure 9: The learning curve in the training phase

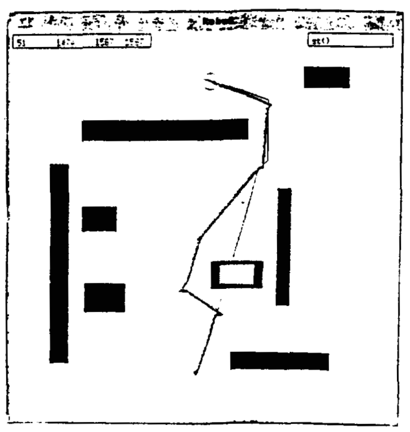


Figure 10: Simulation result 1

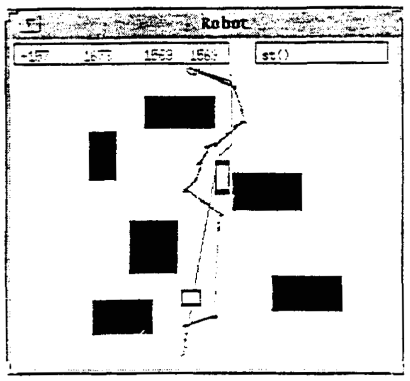


Figure 11: Simulation result 2

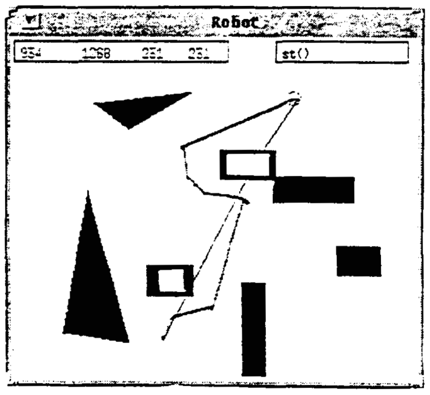


Figure 12: Simulation result 3

5 Simulation Results

The trained neural network was tested on the simulator using randomly generated environments. Our experiments showed that given an arbitrary map with unknown obstacles, the robot can get to its destination successfully for about 90% of the time. The success rate can be improved if the clearance (as defined by the safety zone) on the side is reduced. Figures 10, 11 and 12 show the robot traces from several simulation runs. In these figures, the filled objects are assumed to be known in advance and the outlined objects represent unknown obstacles. The thin line indicates the path generated by the genetic path planner and the thick line is the actual robot trajectory. In addition, the performance of the hand coded collision avoidance program and the neural network controller are compared. As shown in Figure 13, our system outperformed the hand-coded program in most situations.

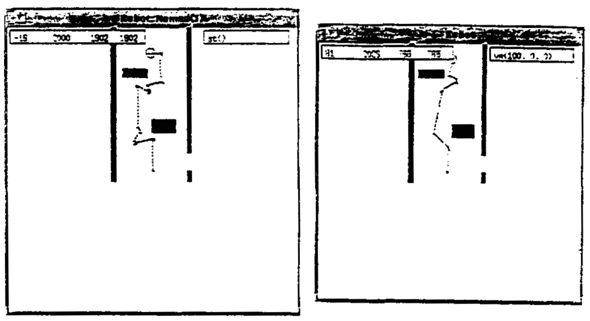


Figure 13: Sample comparison of hand-coded (left) and neural networks (right) collision avoidance navigation.

6 Conclusion

In this paper, we have presented an implemented navigation system that can effectively achieve its goal configurations in a partially known environment with unknown obstacles. The global path planner uses genetic algorithms to search for a collision-free path on the visibility graph constructed among known obstacles. The local reactive navigator tracks the path while avoiding unknown obstacles using a neural network that maps run-time sensory information into the appropriate action(s). Our empirical data showed that the GA path planner can find a near-optimal free path in less time than the standard A* search. The quality of the path is proportional to the amount of time available at the planning stage. The neural network controller reacts to its sensor data with minimal amount of detouring from the planned path. It also outperforms the hand-coded collision avoidance program in terms of the resulting path length and time. Due to its ability to learn, the two-phase system can successfully adapt to changes in the environment.

References

[1] Y. Davidor, "A Genetic algorithm Applied to Robot Trajectory Generation", PhD dissertation, Imperial College, London, 1989.

[2] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, Reading, NY, 1991.

[3] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[4] D.R. Hush and B.G. Horne, "Progress in supervised neural networks: What's new since Lippmann?", *IEEE Signal Processing Magazine*, pp. 8–39, 1993.

[5] S. Ishikwa, "A method of indoor mobile robot navigation by using fuzzy control", *Proc. of IEEE International Workshop on Intelligent Robots and Systems*, pp. 1013–1018, Osaka, 1991.

[6] T. Kimoto, D. Masumoto, H. Yamakawa, and S. Nagata, "Hierarchical sensory information processing model with neural networks", *Proc. of the 1993 IEEE International Conference on Robotics and Automation*, 1993.

[7] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[8] R. Lippmann, "An introduction to computing with neural networks", *IEEE ASSP Magazine*, 4:4–22, 1987.

[9] T. Lozano-Pérez and M.A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles", *Communications of the ACM*, 22(10):560–570, 1979.

[10] Y. Maeda, M. Tanabe, M. Yuta, and T. Takagi, Yoichiro Maeda, Minoru Tanabe, Morikazu Yuta, and Tomohiro Takagi, "Hierarchical control for autonomous mobile robots with behavior-decision fuzzy algorithm", *Proc. IEEE int. Conf. Robotics and Automation*, pp. 117–122, 1992.

[11] Maja J. Mataric, "Integration of representation into goal-driven behavior-based robots", *IEEE Trans. Automatic Control*, 8(3):304–312, 1992.

[12] J.L. McClelland and D.E. Rumelhart, *Parallel Distributed Processing*, Reading, MIT Press, 1986.

[13] S. Nagata, M. Sekiguchi, and K. Asakawa, "Mobile robot control by a structured hierarchical neural network", *IEEE Trans. Automatic Control*, pp. 69–76, 1990.

[14] N.J. Nilsson, "A Mobile Automaton: An Application of Artificial Intelligence Techniques", *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pp.509–520, 1969.

[15] A. Sankaranarayanan and M. Vidyasagar, "A new path planning algorithm for moving a point object amidst unknown obstacles in a plane : A new algorithm and a general theory for algorithm development.", *Proc. of 29th IEEE Conf. Robotics and Automation*, pp. 1930–1936, 1990.