World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:2, 2009

Design and Simulation of a New Self-Learning Expert System for Mobile Robot

Rabi W. Yousif, and Mohd Asri Hj Mansor

***Abstract*—**In this paper, we present a novel technique called Self-Learning Expert System (SLES). Unlike Expert System, where there is a need for an expert to impart experiences and knowledge to create the knowledge base, this technique tries to acquire the experience and knowledge automatically. To display this technique at work, a simulation of a mobile robot navigating through an environment with obstacles is employed using visual basic. The mobile robot will move through this area without colliding with any obstacle and save the path that it took. If the mobile robot has to go through a similar environment again, then it will apply this experience to help it move through quicker without having to check for collision.

***Keywords*—**Expert system, knowledge base, mobile robot, visual basic.

I. Introduction

EXPERT systems are being already used in almost all aspects of our life, from space travel to agriculture, Internet to underwater devices [1]. Knowledge-based systems techniques and applications will be one of the key technologies of the new economy [2].

The expert system receives facts from the user and provides expertise in return. The main components of the expert system are knowledge base and the inference engine. The inference engine may infer solutions from the knowledge base, based of the facts supplied by the user.

There are two main approaches to expert system design [3], [4]. In the first approach, knowledge representation in a conventional expert system is based on rules. This means that a human expert is needed to extract regularities from his experiences and to express them in the comprehensible, explicit, form of rules. Even though the system has perfect explanation abilities, due to the explicitness of the knowledge, the building of such a consistent knowledge base is a difficult process. The second approach is by utilizing artificial neural network (ANN) to generate or construct the knowledge base [5], [6]. Building such a system takes a shorter time because there's no need to disseminate the knowledge as required in the previous approach. Unfortunately, there is no general way to identify a purpose to single neurons in ANN because of the implicit knowledge representation.

Rabi W. Yousif is with the School of Engineering and Science, Curtin University of Technology Sarawak, Miri, CDT 250, 98009 Malaysia (phone: +60 85 443964; fax: +60 85 443837; e-mail: rabi.habash@curtin.edu.my).

Mohd Asri Hj Mansor is with the Faculty of Electrical Engineering, Universiti Teknologi Mara, Shah Alam, 40450 Malaysia.

In this paper, a self-learning expert system (SLES) for mobile robot is considered. SLES is an expert system that utilizes data collected or acquired from previous actions or operations. This technique is designed to automatically generate and store experiential data when it is applied. It then generates its own knowledge base.

In order to realize its implementation a simple application utilizing SLES is constructed. This application will take a form of a simulated mobile robot moving through an environment with obstacles from a specified START to GOAL positions. The concept is similar to a human taking a journey through an unknown terrain, like a town. The first step is to move slowly from the initial position and map its path in memory of the journey to its goal. Then, every time there is an expedition from roughly the same start position to the goal, the human or mobile robot will use this knowledge that was acquired to make the journey quicker, safer and easier.

The problem statement of this study is to develop a technique that will allow the generation of knowledge base with or without the full need of a human expert. The application making use of this technique will be designed to make use of *experiential data*. This experiential data could be considered to be similar to the training data of ANN, and it is accumulated during the running of the application. The advantage of this method is that it is still able to maintain the explanation abilities albeit not as comprehensible as conventional expert system. The generation or construction of the knowledge base will also be quicker as it does not need the full knowledge and experience of the human expert.

We use simple application of a simulation of a mobile robot navigating through an environment with obstacles to demonstrate the SLES at work.

II. System Description

A. User Interface

Visual Basic (VB) was used to program the graphical user interface for the simulation. VB is a Rapid Application Development (RAD) [7] programming language just as Borland Delphi. Visual Basic was chosen due to its capability for Windows programming. It allows easy programming of windows-based programs with forms, buttons and the likes. Fig. 1 shows the graphical user interface for the simulation of the mobile robot using VB.

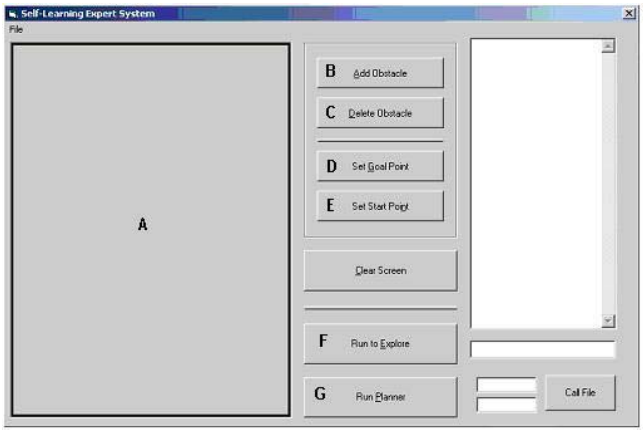


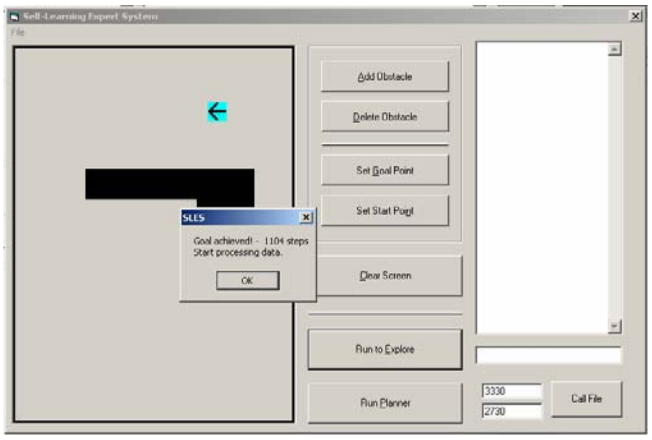
Fig. 1 The graphical user interface for the simulation of the mobile robot

The user clicks once on button B, 'Add Obstacle', to add obstacles. The button B will be depressed, indicating that the user can now click anywhere within the screen area A to add obstacle within it. The obstacle will be a solid black rectangle with its top left vertex positioned at the same point as the mouse arrow was at the time of clicking. More than one obstacle can be added. They can also be combined in such a way that the resulting obstacle have a different shape rather than the just the solid black rectangle. The obstacle can also be deleted by first clicking on the button C, 'Delete Obstacle', which will depress the button. The user can then click on the obstacle that needs to be deleted. This will erase just that obstacle.

Once the obstacles are set, the user can then place the START and GOAL positions for the mobile robot. This can be done by clicking buttons D, 'Set Goal Point', and E, 'Set Start Point', to place the GOAL and START points, respectively. If for any reasons, the placing of the obstacles and the START and GOAL positions are not satisfactory, the user can click on the button titled 'Clear Screen' to clear the whole screen area.

Once everything is in place, the button B, 'Run to Explore', can be clicked. This will run or execute the *explorer* program. This explorer program is used to navigate its path through the environment. While it is moving from its START point towards the GOAL, the mobile robot, in conjunction with the explorer program, will plot an obstacle-free path and storing them in memory. This usually will take some time, as the mobile robot has to check its every step. The mobile robot will stop once it reaches its destination as shown in Fig. 2.

After a new START point has been set in the screen area A within the same environment, the 'Run Planner' button G is clicked to move the mobile robot again. Now the mobile robot will make use of the knowledge that it acquired in its initial move through the environment, or its *experience*. If the new path takes it near the old path, then rather than going through the checking for obstacle for every step, it will use the old path to get nearer to the new destination. This will shorten the time taken, as it no longer has to check for obstacle after every step as in the initial passage.



Goal achieved! - 1104 steps
Start processing data.

Fig. 2 End of the trip for the mobile robot with the number of steps shown

B. Algorithm

The program that was written for the simulation is quite complex and long. The program has to be able to plan a path through the environment with obstacles from its given START point to a given GOAL location. It must also check that its path is free from obstacles. Every steps of the correct path, i.e. it is clear from obstacles, is stored or memorized so that it can be reused whenever needed. A flowchart of part of the algorithm, the explorer, is shown in Fig. 3.

It is not possible to give the standard flowchart, as VB is an event-based programming language as opposed to a sequential-based programming language like BASIC, C or C++. This means that the program runs whenever an event is activated, like clicking on a button, pressing a button on the mouse or even moving the mouse over a certain part of the window. Thus, the program acts in a random manner depending on which event it needs to service.

The main programs are related to the two buttons shown in Fig. 4: Run to Explore and Run Planner. These two programs form the bulk of the whole program as they are responsible for the tasks described earlier. To make the programs more manageable, they are split into procedures. The procedures are:

1) *CloseTo()* and *fnZero()* are procedures of type functions for the management of data.
2) *fnNew_rOrient()* is a function responsible for the new orientation of the mobile robot.
3) *fnObstacle()* is the part of the program that checks whether the mobile robot has collided with an obstacle.
4) *subMove()* moves the mobile robot forward.
5) *fnGoalInFront()* checks whether the goal has been achieved or not. This is just so that that part of the program to explore can be stopped to await the next command.
6) *fnTurn()* will turn the mobile robot towards a prescribed direction.

7) *subDisplayRobot()* will run the commands to print or display the mobile robot within the screen area. The mobile robot can be seen moving due to this program.
8) *subPathExplorer()* is the next program executed once either Run to Explore or Run Planner is clicked. This part is responsible for most of the workings of the program, and it takes the longest time to realize and code. This code will directly or indirectly calls almost all the procedures described. It is shown as a flowchart in Fig. 3.
9) *subProcessData()* rearranges the data collected from the *exploring* before saving to a file.
10) *fnPosit()* is similar to modulus in mathematics, which is to turn a number into a positive value. This was written as the author could not find an equivalent command.
11) *subSaveData()* will save data to a file after it has been *processed.*

C. Obstacle Avoidance

When the Run to Explore button is clicked, after having set the START and GOAL points and the obstacles, this program will be executed. Firstly, it will check whether the goal has been achieved. If yes, then it will stop. Otherwise, the program will check for forward obstacle. If there is none, it will take a step forward. If there is an obstacle, it will turn accordingly. After the turn, it will check for forward obstacle again. If there is none, it will proceed forward; otherwise it will check for obstacle again. This will be repeated again until it reaches its GOAL point. This is just a brief description of the algorithm. As can be obviously seen from the flowchart, the actual algorithm is a bit more elaborate than just described.

III. Results and Discussion

There are two parts to the simulation. The first part will run the mobile robot through the environment with the explorer mode on. Moving through the workspace, the mobile robot will attempt to avoid collision with the obstacles and mapping this collision-free path from START to GOAL.

This path is then processed to remove any unnecessary or irrelevant path, so that the final path is optimized. This will be made clearer from the result of the investigations. It will then be saved to a file. The second phase will be to assess its success by running the mobile robot through the same environment with either a similar or different start point to the same goal point or near its vicinity.

Three runs of the simulation were actually done, each in a different environment with different obstacles and start positions. However, on this technical report only the first run is described. Fig. 4 shows the graphical user interface of the workspace the mobile robot had to navigate through for the first run. The obstacle is placed as shown in the diagram, and the start and goal points are marked as an ‘arrow’ and a ‘crossed-out G’ respectively.

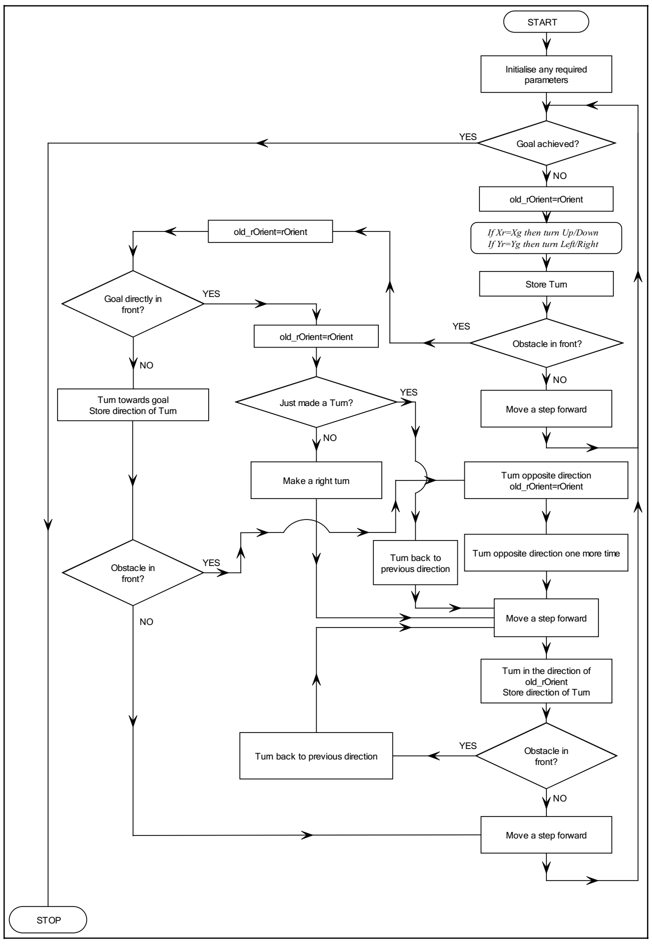

START
Initialise any required parameters
Goal achieved?
YES
NO
old_rOrient=rOrient
If Xr=Xg then turn Up/Down
If Yr=Yg then turn Left/Right
Store Turn
Obstacle in front?
Move a step forward
Turn opposite direction old_rOrient=rOrient
Turn opposite direction one more time
Move a step forward
Turn in the direction of old_rOrient Store direction of Turn
Obstacle in front?
Move a step forward
Turn back to previous direction
old_rOrient=rOrient
Goal directly in front?
old_rOrient=rOrient
Just made a Turn?
Turn towards goal Store direction of Turn
Make a right turn
Turn back to previous direction
Obstacle in front?
STOP

Fig. 3 A flowchart of the obstacle avoidance program

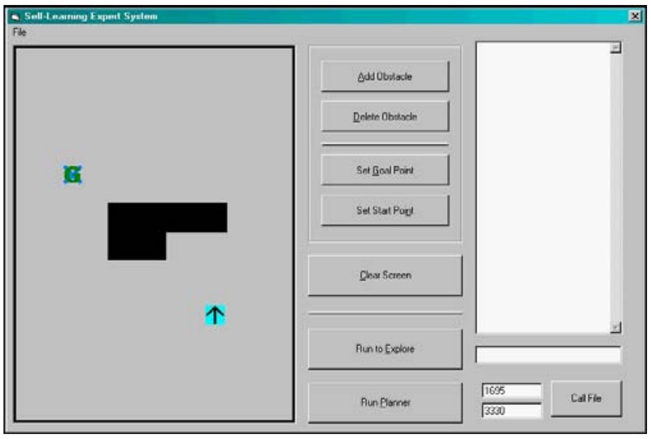


Fig. 4 A picture of the workspace with obstacles, and START and GOAL points

Fig. 5 shows the actual path taken by the mobile robot while navigating through the environment for the first time. As can be seen, it follows the wall of the obstacle quite closely once its path towards its goal is interrupted. The direction of movement is still towards the goal, similar to what a human being will undertake. This is to make certain that the GOAL is never lost sight of.

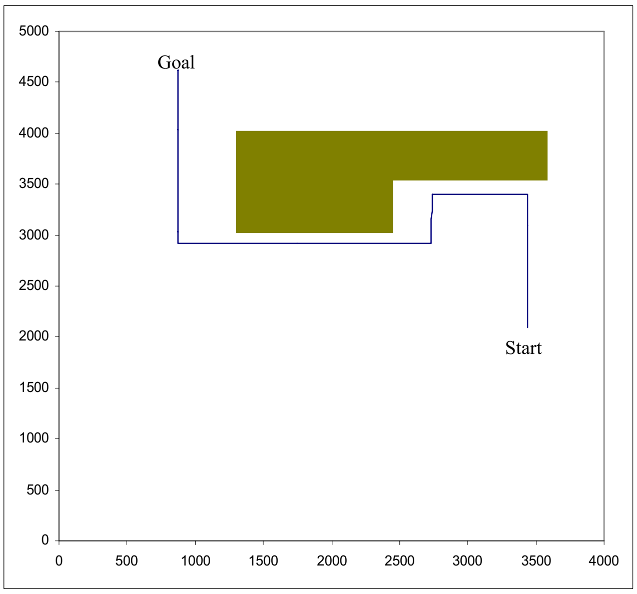
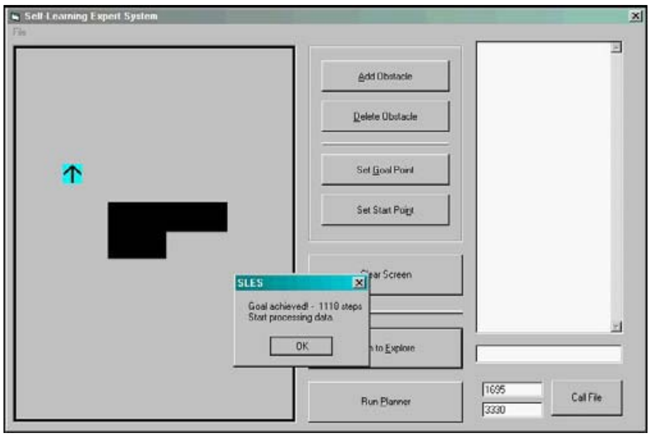


Fig. 5 The actual path taken by the mobile robot navigating through the workspace

Once it reaches its GOAL position, the explorer mode program will end. The program will state the number of steps or actions taken to reach its goal (Fig. 6). Actions or steps could be a move or turn. In this instance, it was 1118 steps. The program will then process the path to optimize the final path before being saved to a file. This final path will rid of any unneeded path such as repeated path, near path and so on. The final path is as shown in Fig. 7. The final path consists of 880 steps.



Goal achieved! - 1110 steps
Start processing data.

1695
3330

Fig. 6 The journey of the mobile robot ends

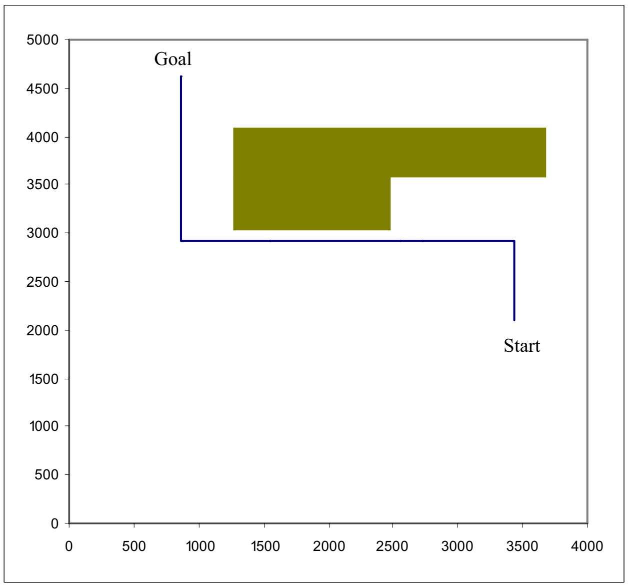
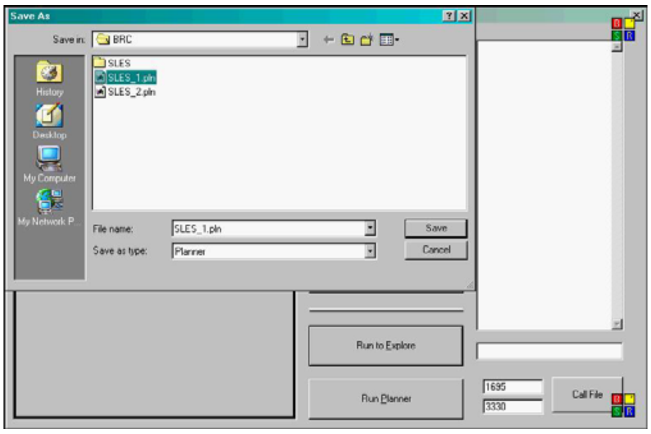


Fig. 7 The final optimized path of the mobile robot

Some part of the wall-following track was deleted for optimization of paths. Only a small part of the wall-following track was retained. This resulted in a shorter path, and thus a quicker route. This part of SLES technique.

A human being will also perform the *optimization* while navigating through the environment. As an example, while traversing, if the human being saw a path that he or she can cut across in order to save time and distance without getting lost of the GOAL, he or she will do so immediately. This was not implemented as the increased complexity in the programming involved at the time was too much time consuming for this work. Both the actual and optimized data will then be saved in memory as shown in Fig. 8.



Save in: BRC
SLES
SLES_1.pln
SLES_2.pln
File name: SLES_1.pln
Save as type: Planner
1695
3330

Fig. 8 The dialog box to save both the actual and optimized data

To assess the effectiveness of the algorithm, two assessment runs were done. The first assessment run was through the same environment, but with a different START position (Fig. 9). Even though no timing was recorded, it was obvious that the run was a lot quicker due to 2 things; the path was now optimized, i.e. it is a shorter route from START to GOAL; and there were no more need for the mobile robot to do any checking for obstacles, and thus can move through quite quickly on some paths. The only checking was done from the *new* START to the *old* START position.

Fig. 9 shows the path that was taken by the mobile robot. The action taken by the mobile robot was described earlier.

For the second assessment run, an extra obstacle was added as shown in Fig. 10. Again, the run time is a lot quicker due to the reasons stated earlier.

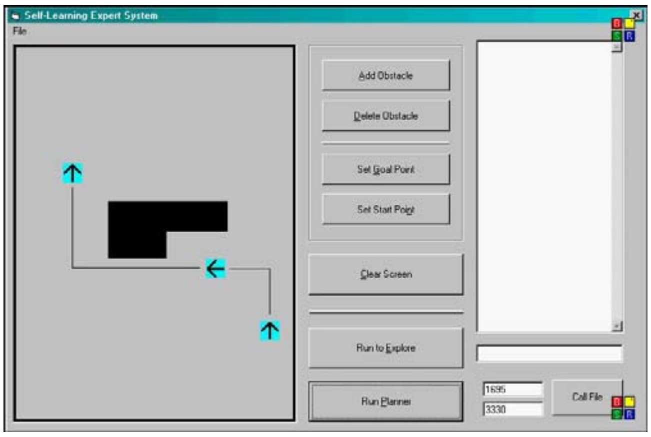


Fig. 9 The path that was taken from START to GOAL

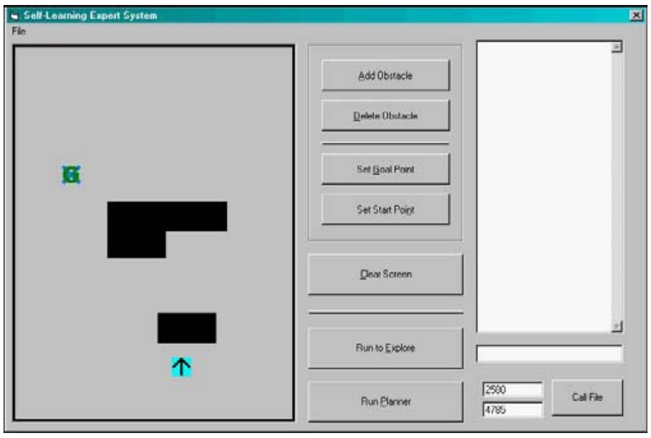


Fig. 10 The second test run after the path has been optimized with the newly added obstacle

Fig. 11 shows the mobile robot at the old START point after having navigated through the environment from the new START point and avoiding collision with any obstacle, in this instance, the newly added obstacle. Once it reaches the old START point, it will employ the optimized path stored in memory to move to its GOAL position much more quickly. This path can be traversed more quickly as it is guaranteed to be an obstacle-free path and thus no checking for obstacles was needed.

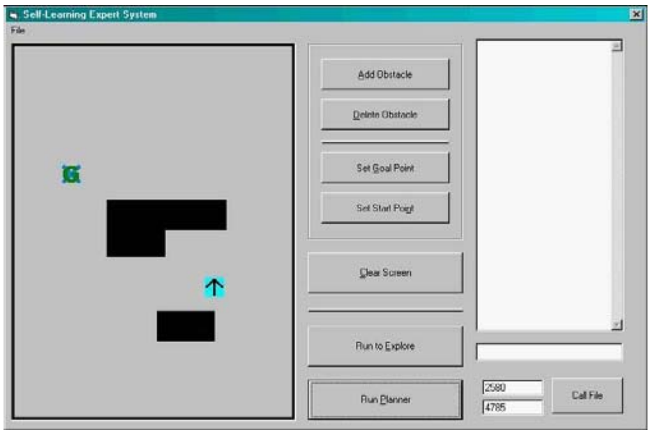


Fig. 11 The mobile robot at the old START point after navigating from the new START point

Fig. 12 shows the actual path that the mobile robot took in order to complete its assigned task. As can be seen in the diagram, the mobile robot navigated around the obstacle in order to get to the *old* START point before continuing towards its GOAL. The path taken by the mobile robot is, from the point of a human being, is not efficient. This is because, instead of turning left at the new START, the mobile robot turned right. This is not due to the Self-Learning Expert System, but rather due to the algorithm of the explorer program. It was designed to go to the old START position first before applying the optimized stored track. Further, the algorithm was not designed to do 'on-the-fly' planning as described earlier. Programming-wise, this can be implemented but since the time was not available, it was decided not to give it such a capability yet. This is in no way reflective of the Self-Learning Expert System, but rather the way the algorithm was planned and designed from the start.

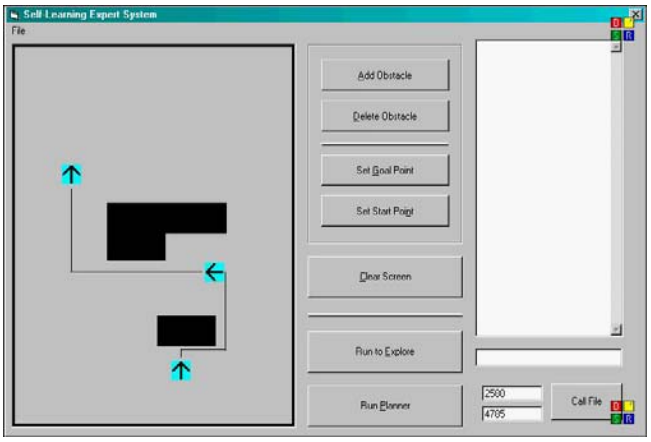


Fig. 12 The path of the mobile robot around the newly added

IV. Conclusion

A knowledge-based self-learning expert system (SLES) for mobile robot is presented. The system is designed to collect or acquire data from actions or operations and automatically generating, optimizing and storing that data. This effectively, is creating and storing the data as a knowledge base or experiential data.

Even though the work focuses on SLES, the outcome of this work also produces a simplistic mobile robot *path planner*. It was shown that the path planner was successful in navigating through the specified workspaces. It may not be an optimized path but the path was done without any collision with any of the obstacles.

It was also shown that the technique was successfully applied to the mobile robot. The data gathered after the successful path tracking through the workspaces were optimized before being stored for later usage. This is the experience of the mobile robot navigating through that area. Thus if the mobile robot has to go through the same area again, then it will access that part of the experiential data to help it navigate through. The assessment runs that were done proved that the technique can be applied effectively and successfully.

This work can be further enhanced by making the path planner more optimal. The techniques that can be applied are genetic programming, genetic algorithm, and potential field and others. For the mobile robot, at least, the path planner can be integrated with the SLES. That is, the 'on-line' or 'on-the-fly' planning using the experiential data can be incorporated so that the assessment runs will be more optimal.

REFERENCES

[1] Cornelius T. Leondes, *Knowledge-Based Systems Techniques and Applications*. San Diego, Calif.; Academic, 2000.
[2] Ovidiu S. Noran. (2003). The Evolution of Expert Systems. Available: http://www.cit.gu.edu.au/~noran
[3] Levine Robert et al, *AI and Expert Systems*, McGraw-Hill, 1990.
[4] Durkin John, *Expert Systems: Design and Development*, Prentice-Hall Int., 1994.
[5] Šíma Jiří & Neruda R., "Neural Expert Systems", *Proceedings of IJCNN*, Beijing, 1992.
[6] Šíma Jiří & Neruda R., *Neural Networks as Expert System*, Neural Network World, 2, 775-784, 1992.
[7] Halvorson Michael, *Microsoft Visual Basic 6.0 Professional Step by Step*, Microsoft Press, 2002.