

ornl

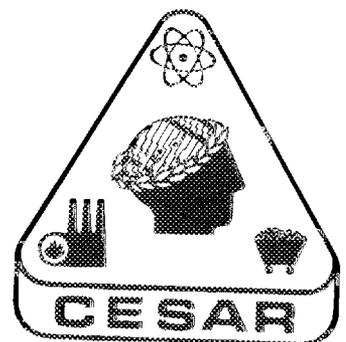
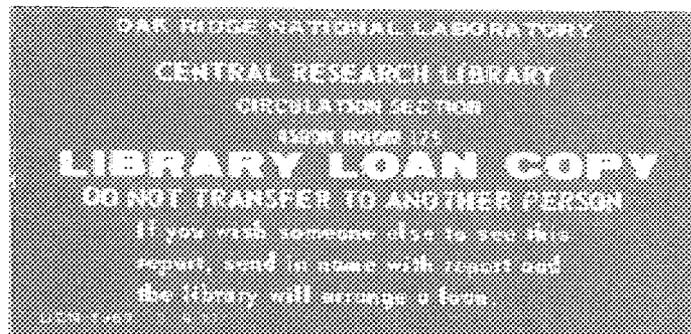
ORNL/TM-10726
(CESAR-88/09)

**OAK RIDGE
NATIONAL
LABORATORY**

MARTIN MARIETTA

Path-Planning in a Known Environment with Unexpected Obstacles: Potential Application to an Automatic Alarm Testing Robot

N. Sreenath



OPERATED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

Printed in the United States of America. Available from
National Technical Information Service
U.S. Department of Commerce
5263 Port Royal Road, Springfield, Virginia 22161
NTIS price codes—Printed Copy: A03; Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM-10726
CESAR-88/09

Engineering Physics and Mathematics Division

**PATH-PLANNING IN A KNOWN ENVIRONMENT
WITH UNEXPECTED OBSTACLES: POTENTIAL
APPLICATION TO AN AUTOMATIC ALARM
TESTING ROBOT**

N. Sreenath

Center for Engineering Systems Advanced Research
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, TN 37831-6364

Date Published: May 1988

Prepared for the
Energy Research Program
Office of Basic Energy Sciences
U.S. Department of Energy

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
operated by
MARTIN MARIETTA ENERGY SYSTEMS, I
for the
U.S. DEPARTMENT OF ENERGY
under contract No. DE-AC05-84OR21400



3 4456 0273678 8

TABLE OF CONTENTS

	Page
ABSTRACT	ix
I. INTRODUCTION	1
II. PROGRAM FEATURES	3
III. ALGORITHM	5
IV. PROGRAM DESCRIPTION	7
IV.1. Input File	7
IV.2. Data Structures (Partial List)	9
IV.3. Mission Mode	10
IV.4. 'Singleroute' Routine	11
IV.5. Routines and Functions	13
IV.6. Output File	15
V. SIMULATION	19
VI. SUGGESTED IMPROVEMENTS	21
VII. CONCLUSION	23
ACKNOWLEDGEMENTS	25
APPENDIX A	A1
APPENDIX B	B1

LIST OF FIGURES

	Page
Fig. 1. A sample floor map with 30 nodes	8
Fig. 2. Programming flow chart	17
Fig. 3. Floor plan for demo	20

LIST OF TABLES

	Page
Table 1. Program Support Files and Associated Function Routines	16

ABSTRACT

The feasibility of using an expanded version of a path planning algorithm in the development of an Automatic Alarm Testing Robot (AATR) was investigated. Use of an AATR for security tasks implies the requirement that communication exists between the robot and the security systems or personnel except in the event of a site emergency or mechanical failure aboard the robot. Furthermore, the execution of the actual alarm testing needs to be carried out in narrow, pre-defined time windows. Thus the path planning algorithm for such a robot should be capable of not only taking the robot from one point to another in a known environment with unexpected (a priori unknown) obstacles in real time, but also time-synchronization with the security systems. Here we propose such an algorithm which uses a breadth-first search technique and discuss its implementation onboard the HERMIES IIB robot. A demonstration of the algorithm in a hard real-time environment is presented and some future desirable enhancements of the algorithm are discussed.

I. INTRODUCTION

In recent years, an increasing number of studies have been undertaken to investigate the use of autonomous robots in performing security related functions. This paper relates to the development of an Automatic Alarm Testing Robot (AATR) designed to test alarm systems by physical intrusion. Within this framework, it is assumed that an AATR robot will be fully autonomous during the execution of its mission. There will be no communication between the robot and other security systems or personnel except in the event of a site emergency or if the robot is unable to return to the charging station either because of mechanical failure aboard the robot or because of obstacles in the work site. The robot will be briefed and debriefed at the charging station via a secure direct link to the central security computer system.

Alarm testing missions will be scheduled automatically by the master robot control system in conjunction with the security computer so that the alarm testing will be synchronized with narrow time windows during which the security computer is aware that a particular alarm is to be tested and should be recorded as 'functional' or 'no response.' At the debriefing time the robot's mission log will be compared to the security computer alarm log to confirm that all alarm firings occurred as a result of robot activities and determine which of the 'no response' alarms correspond to alarms not triggered by the robot or to alarms requiring repair service. Because of the unexpected nature of the environment it would be highly inconvenient to enforce well defined robot paths clear of all obstacles between each alarm location. Therefore, the robots must have the capacity for detecting obstacles, for collision avoidance, for path planning and for 'time synchronization.'

We give here a program that is intended as a conceptual design alternative for high-level on-board path planning for such an AATR system. A brief description of the program features, the implemented algorithm, description of the important

data structures used in the program and finally the program implementation on the HERMIES IIB robot are given here along with some suggested enhancements.

II. PROGRAM FEATURES

The program has the following features:

- * has a priori knowledge of the floor plan of the environment – defined in the form of connected points or *nodes*,
- * finds feasible paths and generates primitives (actual instructions) for the robot to go from each point (*node*) to the next, in the order specified in the mission file (it does not try to solve the “traveling salesman” problem of optimizing the mission distance),
- * if a particular goal on the mission is inaccessible, for example, if all paths to the goal point (called the *Goal_node* in the program) from the start point (called the *Start_node* in the program) are blocked, the program skips that part of the mission and plans the rest,
- * if an *unexpected obstacle* (defined as an obstacle unknown to the robot a priori) is encountered when the robot is executing a particular portion of the path, (i.e., detected using sonar or other sensors) the robot returns to the nearest node it just passed, and replans another feasible path (if it exists) from this node to the *Goal_node*; it also updates the world map to indicate that portion of the path where the obstacle was encountered as being blocked,
- * if the collision avoidance sensor detects an obstacle, the program checks whether the path to the next node is clear (i.e., the obstacle is actually beyond the next node in the planned path), if so, the program moves the robot to the next node,
- * continuous motion of the robot until it reaches a turn (as opposed to stop-and-go at each *node*),

- * the program has a sense of *time synchronization*, i.e, if a particular portion of the mission is not completed in a specified time (known as the mission time) the program will skip the present *Goal_node* and proceed to execute the rest of the mission,
- * written in 'C,' the program runs in a real time environment on board the HERMIES IIB robot, on the NCUBE (host) computer.

The algorithm uses a breadth-first technique of searching a network. By this, we mean that it explores the “maze” of the network according to an algorithm that has the effect of searching the “trunk” and all main branches before it searches the leaves. (A “depth-first” algorithm would go all the way to the end, or “leaf” of one of the possible paths before restarting near the “trunk” to follow an alternative path). The breadth-first algorithm tends to minimize the distance required to find a feasible solution (sub-optimal solution) at the expense of:

1. Keeping a relatively large number of partial paths in memory.
2. Finding only a single path to the destination.

Remark 2.1: The algorithm gives a shortest path (optimal path) if the floor plan can be divided into a set of rectilinear symmetrical grids.

For the application we have in mind, this seems to be a workable method. The format for the data base is as follows: the floor area of the building is divided into a rectilinear grid of nodes labeled with increasing integers starting with '1.' Each node is connected to four adjacent nodes (except at the edges). A path between nodes is however subject to being “blocked.” The task of this program is to find a feasible route from a starting location and orientation to a goal. This information is then converted to a set of dead-reckoning directions compatible with the set of HERMIES primitives (control commands) and executed on the robot.

III. ALGORITHM

The algorithm, in detail, can be thought of as sending “scouts” out to explore a maze. The rules are:

1. One scout is examined at a time. The scout that is examined is the one with the smallest cumulative travel distance so far.
2. The scout checks to see if it is currently at the overall destination; if so, the program declares success and quits.
3. If not, new scouts are “created” for each node connected to the current node and moved to those nodes. The scout at the current node is removed.
4. The scouts at the new nodes are removed if the nodes prove to have had an earlier visitor with a smaller cumulative distance. That is, if any other scout had arrived at that location via a shorter route, then its route to the goal will be shorter than the route currently being processed.
5. If there are no more scouts left, then no legal route to the destination exists, and the program declares failure and quits. Otherwise, processing reverts to the first step.

IV. PROGRAM DESCRIPTION

The description of the program is best done using a flow chart, however, we first describe the various files, data structures, routines, functions and other components associated with the program and finally give the flow chart.

IV.1. Input File

The input file consists of the floor map of the building, divided in the form of a rectilinear grid of nodes with the nodes labeled from '1' onwards in an increasing order of consecutive integers. We also refer to the node label as the 'ID#' of the node. Figure 1 displays a sample floor map with 30 nodes.

The format of the data in the input file is as follows:

```

Number of nodes in the data (integer to a maximum of 99)
number of neighbors of node 1 — in direction — neighbor ID# — at
distance — in direction ...
number of neighbors of node 2 — in direction — neighbor ID# — at
distance — in direction ...
      :                               :                               :
      :                               :                               :
number of neighbors of node i — in direction — neighbor ID# — at
distance — in direction ...
      :                               :                               :
      :                               :                               :

```

Here the character '—' has been used just to indicate a blank space (this character should be replaced by a blank in the data file).

The 'in direction' of the neighbor can be N, S, E or W.

The path between neighbors is considered blocked if the distance between them is '9999.'

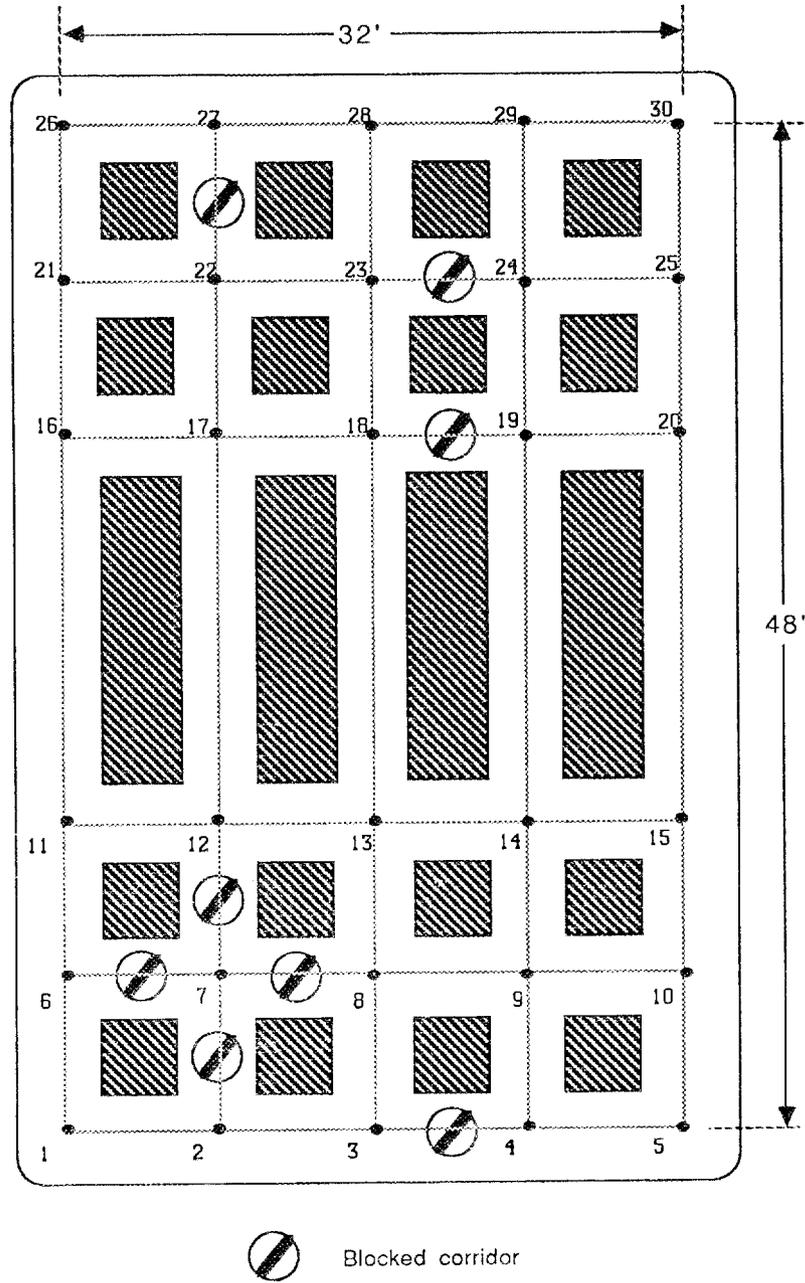


Figure 1 : A sample floor map with 30 nodes.

NOTE: The program also refers to the directions N, E, S and W as the integers 1, 2, 3 and 4 respectively (clockwise notation). For the sake of convenience, the symbolic and numerical references to the directions are used interchangeably by the program.

IV.2. Data Structures (Partial List)

1. The floor map of the building is stored using an internal data structure 'cnode.' Each element of this list corresponds to an equivalent node on the floor map. Each node is assumed to have a maximum of four neighbors. The ID# of the neighbors is stored using the record '.neighbor,' which is a list by itself. We also define:

.neighbor[1] - North neighbor,
.neighbor[2] - East neighbor,
.neighbor[3] - South neighbor,
.neighbor[4] - West neighbor,
.neighbor[0] - unused.

Correspondingly we have another record '.length' which is also a list, and stores the distances to the respective neighbors. Again '.length[0]' is unused. If the distance between two neighboring nodes is equal to '9999' then the path between the nodes is considered blocked.

2. During the search for paths, the internal data structure 'cnode' contains information on whether the node is on an active route for potential paths, i.e.,

.status = FULL,

or not, i.e.,

.status = EMPTY.

If *.status* is *FULL* then the record for storing the ID# of the previous node to which it is connected to on the path,

.pnode,

and the record to store the cumulative distance so far on the route:

.cum,

are defined.

Thus from any such route segment, the path from the origin to that point can be reconstructed and the data stored compactly.

3. The list (array) of “active routes” defined by the internal data structure - ‘Active,’ consists of records, each of which contains the cumulative distance so far for that route. This list can be thought of as containing the locations of all of the “scouts” who are out exploring the network. The records used are as follows:

.cum - the cumulative distance up to that point,

.name - the ID# of the node.

The number of active routes is stored in ‘Nactive.’

IV.3. Mission Mode

The program works in “mission” mode where the user can specify a connected sequence of locations to be visited. This could represent a mission consisting of testing several alarms and returning to base. The mission is read from a data file, one location per line in the format:

Number of mission nodes (integer)

ID# of node --- direction --- mission type --- mission time

Here the character ‘—’ has been used just to indicate a blank space (this character should be replaced by a blank in the data file).

The first line specifies the number of mission nodes (including starting node). The second line specifies the starting location and direction (orientation). The direction can be 1, 2, 3 or 4 corresponding to N, E, S or W directions respectively. Each succeeding line is taken as the next goal (and desired final orientation at that goal). The mission type can be any of the following:

S *Snap a picture (to be later used for self location and correction)*

D *test the Door intrusion alarm,*

M *test the Motion detector alarm,*

H *test the Heat detector alarm,*

N *do Nothing.*

We can also specify the maximum 'mission time' allowed for the robot to get to the next mission node from the previous mission node and execute the desired mission type.

IV.4. 'Singleroute' Routine:

This routine finds a single sub-optimal route from any node to any other (or knows that it has failed) by making use of the algorithm described in Chapter III. The routine uses the following routines and functions implicitly or explicitly to find a feasible path and move along it:

spawn(),

not_laggard(),

Addroute(),

Remove(),

Traceback(),

Gen_primitives(),

Re_route().

Once the *singleroute* routine finds a feasible path from the *Start_node* to the *Goal_node* a flag 'Arrived' is set to TRUE (if there is no feasible path it is set to FALSE).

If Arrived = TRUE then the following sequence is executed.

The control is handed over to the 'Traceback' routine which traces the path from the *Start_node* to the *Goal_node* and stores the path using the internal data structure 'trace.' This structure has the following records:

- .name - ID# of present node,
- .dir - direction (orientation) at the present node; this is either 1, 2, 3 or 4, corresponding to N, E, S or W directions (clockwise labelling).
- .next_node
- .length - distance from the present node to the next node.

The control is now handed over to the 'Gen_primitives' routine which generates the HERMIES compatible primitives according to the path information stored in the data structure 'trace.'

The routine generates 'fmoves' commands which uses the sonar ID# 1 to detect any unknown obstacle in the path of the robot. If no obstacle is detected the command 'fmoves' returns '-1,' else it returns the distance traveled so far by the robot. This information is stored in 'Check_Obstacles' and is tested at the end of each 'fmoves' command.

If 'Check_Obstacles' is not equal to '-1' the program tests to see whether the next node is still accessible, i.e., the obstacle is not along the path to the next node but beyond it, even though the sonar detected an obstacle. If next node is still accessible the program moves the robot to the next node using an 'fmove' command. Otherwise, if an obstacle is found on the way the robot returns to the nearest node and declares that node as the *Start_node* and proceeds to find a 'singleroute' from this node to the *Goal_node*. To achieve this it calls the 'Singleroute' routine again.

We define:

$$Goal_time = mission\ time - wait_time,$$

where *wait_time* is the maximum time required to execute any mission type.

Every time an 'fmoves' command is to be executed the system clock is called using the system routine 'n_clock,' and the time checked to find whether or not the Goal_time has been exceeded. If the Goal_time has been exceeded then the program declares that it is not possible to reach the desired *Goal_node* in the specified mission time, declares the present node as the *Start_node*, skips the present *Goal_node* and continues with the next mission node.

If Arrived = FALSE then the program skips to the next mission node.

IV.5. Routines and Functions

The following is a comprehensive list of routines and functions used by the program, along with a brief description. They are listed in the order in which they are called.

init_route: Initializes all the arrays used to find a feasible route i.e,

Read_data: Reads in the floor map data in the format described in Section IV.1 from the input data file.

set_neighbors: Sets the length (distance) data for the node neighbors.

Read_mission: Reads in the mission data from the mission file.

Singleroute: See Section IV.4.

spawn: Spawns new scouts according to the rules defined in the algorithm (see Chapter III). Calls Not_Laggard.

Not_Laggard: Scouts at the new nodes are removed if their nodes prove to have an earlier visitor with a smaller cumulative distance. That is, if any other scout had arrived at that location via a shorter route, then its route to the goal will be shorter than the route currently being processed. In such cases Not_Laggard returns 'FALSE,' otherwise it returns "TRUE."

Remove: Removes the scout at a given node from the 'Active' route list. May be called to delete a 'father' node which has spawned or to delete a scout which is laggard.

Gen_primitives: Generates primitives like 'fmoves' for the actual movement of the robot. Checks whether the collision avoidance sensor detected an obstacle. If so, checks whether the path to the next node is still clear – using 'Check_node_reachable' routine. If the path is clear the robot is moved to the next node, if not, the robot is moved back to the node it most recently passed and the control is handed over to the Re_route routine. The routine also checks whether the 'mission time' is exceeded or not. If it is exceeded it declares the node where the robot is presently stationed as the *Start_node*, skips the present *Goal_node*, declares the next mission node as the *Goal_node*, and proceeds with the program.

Check_node_reachable: Checks to see whether the path to the next node is clear when the collision avoidance sensor detects an obstacle. In effect checks using the range data from the collision avoidance sensor whether the obstacle lies along the path to the next node or further beyond it.

make_turn: Used to make the robot turn in the relevant direction, by multiples of 90 degrees about its axis, whenever desired.

same_dir: If the robot has to move in the same direction passing many nodes then these piece movements are integrated into a single movement using this routine.

Re_route: Called if an obstacle is encountered by the robot. Calls update_map routine to update the floor map. Finds a new feasible path from the present node to the *Goal_node* by calling Singleroute routine.

update_map: Updates floor map (stored in the internal data structure 'cnode') on encountering an obstacle. Marks the path where the obstacle was encountered as blocked.

Traceback: Traces back from the *Start_node* to the *Goal_node* after a feasible path has been generated by the Singleroute routine.

donne: Executes the mission type after robot reaches *Goal_node*. If all paths from the *Start_node* to the *Goal_node* have been blocked then a sequence of commands are executed to indicate so.

not_donne: Executed in case the robot cannot reach the *Goal_node* in the specified mission time. This is not executed when there is no feasible path from the *Start_node* to the *Goal_node*.

idle_robot: Idles the robot for synchronization when the *Goal_node* has been reached before mission time.

The working of the program is best understood by studying the *flow chart* given in Fig. 2.

Table 1 gives a list of files and of the routines and functions which they contain.

IV.6. Output File

An output file for monitoring the execution of the program is generated. Important steps of the execution are recorded so that the file will be useful for debugging. This is the precursor to the mission log which would be compared with the central security computer alarm firing data to identify faulty alarms (see Chapter I).

Table 1. Program Support Files and Associated Function Routines.

FILES	FUNCTIONS
addroute.h	Addroute
gen_primitives.h	Gen_primitives Check_node_reachable make_turn same_dir Initialize_Robot
global.h	
mission.c	main idle_robot
not_laggard.h	Not_Laggard Remove
re_route.h	Re_route update_map
read_data.h	Read_data set_neighbor Read_mission
singleroute.h	Singleroute
spawn.h	Spawn
theatricals.h	donne not_donne
traceback.h	Traceback

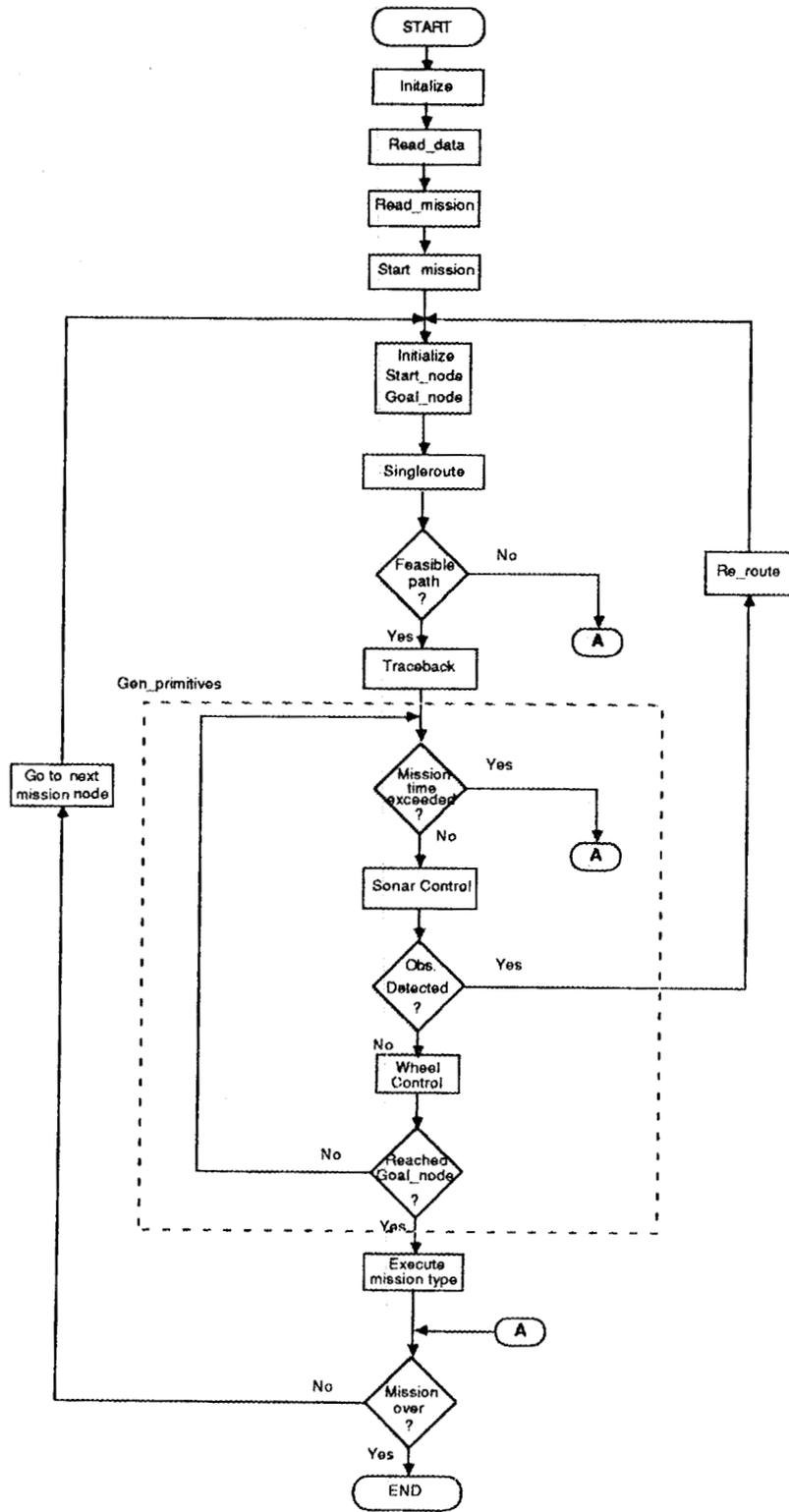


Figure 2 : Program flow chart .

V. SIMULATION

The feasibility of the above program was demonstrated on the HERMIES IIB platform.² A semi-realistic floor plan with corridors and alarms was created. Pieces of (2 feet \times 2 feet) foam slabs were used to simulate the corridors. Some corridors were blocked. A layout of the *demo* is given in Fig. 3. The floor plan simulated had 3 feet wide corridors and the adjacent nodes were 3 feet apart. The *input data file* of the floor plan is given in Appendix A.

The mission was to start from *node 23* and go to *node 1*, do *self-location* by 'snapping a picture' of the icon (Fig. 3). The next part of the mission consisted of testing a 'heat detector' alarm located at *node 6* followed by another *self-location* at *node 29*. Then the robot was expected to go to *node 50* to test a 'doorway intrusion alarm,' another *self-location* at *node 48*, test a 'motion detector' at *node 45* and finally return to *node 23*. Appendix B contains the *mission file*.

The path that would have been followed by the robot in the absence of any *unexpected obstacle* is given by the 'dotted' line. However when the robot was traversing the path between *node 33* and *node 40* an *unexpected obstacle* (in the form of a human being) was placed in the path. On detecting the obstacle, the robot back tracked to the node it just passed i.e., *node 33*; planned another path to the *Goal_node* (in this case *node 50*) and proceeded along the new path. It also marked the path between *node 33* and *node 40* as being blocked in its memory.

Again an *unexpected obstacle* was placed along the new path in the corridor between *nodes 32* and *38*. The robot promptly replanned another path to the *Goal_node 50* and marked the above corridor as being blocked. The rest of the mission was executed without any incident.

The program which is implemented in the 'C' language runs in real time. It is our contention that the program will work in a realistic environment with simulated or real corridors with little or no modification.

²The *demo* has been rendered immortal on a VHS video tape. If interested to view the tape please contact Dr. F. G. Pin at the Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge TN 37831.

VI. SUGGESTED IMPROVEMENTS

- * Presently 'wait_time' (which is defined as the maximum time for testing any alarm) is fixed. This should be varied according to the mission type.
- * Only N, S, E and W directions have been implemented currently. Other directions may be needed depending on the floor plan. Also in case the floor plan contains curved corridors the robot's primitives should be changed accordingly, along with the enhancement of the program.
- * In the above case the feasible path found was not required to be the shortest, in which case the program needs to be enhanced to find the shortest path.
- * A way of recognizing whether an obstacle encountered is *blocking* the path *permanently* or *temporarily* should be defined.
- * In some instances the desired path will be the one which takes the least time rather than the shortest distance. The program could be expanded to include criteria for estimating mission time and accordingly choose a minimum time path.
- * Currently only one Sonar is being used in detecting unexpected obstacles; concurrent usage of other Sonar sensors needs to be explored, along with other types of sensors.
- * Self location and correction should be implemented.
- * Reflex can be implemented (i.e., when an object is moving towards the robot, the robot should get out of the way of the moving obstacle).
- * Read_data routine could be modified to input data (floor plan) in an intelligent way. If we declare a node as a neighbor to another, the program should automatically declare the former as a neighbor of the latter. Presently this data is being read in two times.

VII. CONCLUSION

We have implemented a path-planning program capable of running in real time on board the HERMIES IIB robot, in a structured environment with a-priori unknown obstacles. On encountering such an obstacle the robot replans another path to the goal in real time and promptly executes it. A *demo* of the same was successfully completed on November 10, 1987. At that time 'time-synchronizatiion' was not implemented. Presently it is. This program resides in

/usr/sreenath/y12/t_demo

directory. The program without 'time-synchronization' is in the directory

/usr/sreenath/y12/a_demo.

Both programs reside on the 'host' of the NCUBE machine on board HERMIES IIB.

ACKNOWLEDGEMENTS

We are thankful to C. R. Weisbin and F. G. Pin for introducing us to the project, without their continued encouragement this project could not have been realized.

We wish to acknowledge the contribution of L. D. Trowbridge who originally proposed the algorithm. We thank B. L. Burks for his help in suggesting corrections and in making the *demo* a success. The help of S. M. Killough, M. R. Kedi and others in the CESAR Lab for the implementation of the program onboard HERMIES IIB is appreciated. Finally we thank W. W. Manges, D. B. Reister and other members of the AATR team.

Appendix A

The *input data file* used in the *demo* is given here :

```
51          /* Total number of nodes on the floor map */
2 N 8 3 E 2 3 /* Data of node neighbors */
2 E 3 3 W 1 3
3 N 9 3 E 4 9999 W 2 3
2 E 5 9999 W 3 9999
3 N 10 3 E 6 3 W 4 9999
2 E 7 9999 W 5 3
2 N 11 9999 W 6 9999
2 N 12 3 S 1 3
2 N 14 3 S 3 3
2 N 16 3 S 5 3
2 N 18 9999 S 7 9999
3 N 19 3 S 8 3 E 13 3
2 E 14 3 W 12 3
4 N 20 3 S 9 3 E 15 3 W 13 3
2 E 16 3 W 14 3
4 N 21 3 S 10 3 E 17 3 W 15 3
2 E 18 3 W 16 3
3 N 22 3 S 11 9999 W 17 3
2 N 23 3 S 12 3
2 N 25 3 S 14 3
2 N 27 3 S 16 3
2 N 29 3 S 18 3
3 N 30 3 S 19 3 E 24 3
2 E 25 3 W 23 3
4 N 31 3 S 20 3 E 26 3 W 24 3
2 E 27 3 W 25 3
4 N 32 3 S 21 3 E 28 3 W 26 3
2 E 29 3 W 27 3
3 N 33 3 S 22 3 W 28 3
2 N 34 3 S 23 3
2 N 36 3 S 25 3
2 N 38 3 S 27 3
2 N 40 3 S 29 3
3 N 41 3 S 30 3 E 35 3
2 E 36 3 W 34 3
4 N 42 3 S 31 3 E 37 3 W 35 3
2 E 38 3 W 36 3
4 N 43 3 S 32 3 E 39 3 W 37 3
2 E 40 3 W 38 3
```

3 N 44 3 S 33 3 W 39 3
2 N 45 3 S 34 3
2 N 47 3 S 36 3
2 N 49 3 S 38 3
2 N 51 3 S 40 3
2 S 41 3 E 46 3
2 E 47 3 W 45 3
3 S 42 3 E 48 3 W 46 3
2 E 49 3 W 47 3
3 S 43 3 E 50 3 W 48 3
2 E 51 3 W 49 3
2 S 44 3 W 50 3

Appendix B

The *mission file* used in the *demo* :

```
8      /* Number of mission nodes */
23 2 N /* Node number, orientation, type of mission */
1 3 S
6 3 H
29 2 S
50 1 D
47 1 S
45 4 M
23 2 N
```


INTERNAL DISTRIBUTION

- | | | | |
|--------|-------------------|--------|----------------------------|
| 1. | B. R. Appleton | 21. | P. J. Otaduy |
| 2. | D. L. Barnett | 22. | L. E. Parker |
| 3. | M. Beckerman | 23-27. | F. G. Pin |
| 4. | B. L. Burks | 28. | D. B. Reister |
| 5. | G. de Saussure | 29. | J. T. Robinson |
| 6. | J. R. Einstein | 30. | P. F. Spelt |
| 7. | C. W. Glover | 31-35. | C. R. Weisbin |
| 8. | E. C. Halbert | 36. | B. A. Worley |
| 9. | W. R. Hamel | 37. | EPMD Reports Office |
| 10. | J. P. Jones | 38. | Central Research Library |
| 11. | S. M. Killough | 39. | ORNL Technical Library |
| 12-16. | F. C. Maienschein | | Document Reference Section |
| 17. | W. W. Manges | 40-41. | Laboratory Records |
| 18. | R. C. Mann | 42. | ORNL Patent Office |
| 19. | J. R. Merriman | 43. | Laboratory Records - RC |
| 20. | E. M. Oblow | | |

EXTERNAL DISTRIBUTION

44. Office of the Assistant Manager, Energy Research and Development, DOE-ORO, Oak Ridge, TN 37831
45. Dr. John J. Dorning, Department of Nuclear Engineering and Engineering Physics, Thornton Hall, University of Virginia, Charlottesville, VA 22901
46. Professor Gene H. Golub, UNIACS -- 2321, Computer Science Building, University of Maryland, College Park, MD 20742
47. Dr. Robert M. Haralick, Department of Electrical Engineering, University of Washington, Seattle, WA 98195
48. Dr. Oscar P. Manley, Division of Engineering, Mathematical, and Geosciences, Office of Basic Energy Sciences, ER-15, U.S. Department of Energy - Germantown, Washington, DC 20545
- 49-53. Dr. Narasingarao Sreenath, Systems Engineering Department, Crawford Hall, Case Western Reserve University, Cleveland, OH 44106
54. Dr. Don Steiner, Institute Professor, Department of Nuclear Engineering, Rensselaer Polytechnic Institute, Troy, NY 12181
- 55-64. Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37830