2 Map Building

2.1 Introduction

The phrase *map building* [Patnaik et al., 1998] refers to the construction of a map of the work space autonomously by the robot, which enables the robot to plan the optimal path to the goal. Map building helps the mobile robot to become conversant with the world around it. The information about the neighborhood world of the robot is thus required to be encoded in the form of a knowledge base. For the purpose of navigational planning, a mobile robot must acquire knowledge about its environment. This chapter demonstrates the scope of map building of a mobile robot of its workspace.

It is evident from the discussion in the last chapter that the *acquisition of knowledge* is a pertinent factor in the process of building perception. Human beings can acquire knowledge from their environment through a process of automated learning. Machines too can acquire knowledge by sensing and integrating consistent sensory information. A number of techniques are prevalent for automated acquisition of knowledge. The most common among them are unsupervised and reinforcement learning techniques. In an unsupervised learning scheme, the system updates its parameter by the analyzing consistency of the incoming sensory information. Reinforcement learning, on the other hand, employs a recursive learning rule that adopts the parameters of the systems, until convergence occurs, following which the parameters become time invariant. The chapter includes a technique for constructing a 2D world map by a point mass robot with its program written in C++.

2.2 Constructing a 2D World Map

It is assumed that the height of the robot is less than that of the obstacles within the workspace. In fact, most of the navigational problems for robots are confined to two-dimensional environments. The algorithm for map building in a 2D environment is given below.

There exist two different types of algorithms for automated map building. The first one refers to landmark-based map building [Taylor et al., 1998]and the second one is metric-based map building [Asada, 1990; Elfes, 1987; Pagac et al., 1998]. Offline map building is discussed here utilizing a metric-based approach [Patnaik et al., 1998]. Further, to maintain the order in the traversal of the robot around the obstacles, a directed search is preferred here. The *depth-first search,* which is a directed search technique, is being utilized here.

2.2.1 Data Structure for Map Building

Let us consider a circular mobile robot (shown in Fig. 2.1) that can orient itself in any of the following eight directions: north (N), north-east (NE), east (E), south-east (SE), south (S), south-west (SW), west (W) and north-west (NW).

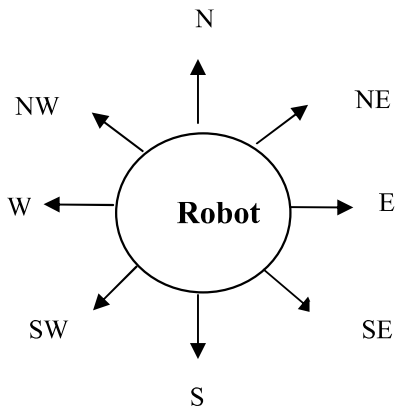


Fig. 2.1. The representation of a circular robot with eight ultrasonic sensors around it in eight geographical directions

In the *depth-first* algorithm, the following strategy is used for traversal within the workspace. The detail explanation is given in the next section:

If there is an obstacle in N
    Then move to the nearest obstacle in N
If there is an obstacle in NE
    Then move to the nearest obstacle in NE
If there is an obstacle in E
    Then move to the nearest obstacle in E
……
……
If there is an obstacle in NW
    Then move to the nearest obstacle in NW

If the above steps are executed recursively, then the robot would have a tendency to move to the north so long as there is an obstacle in the north, else it moves north-east. The process is thus continued until all the obstacles are visited. Another point needs to be noted here, that after moving to an obstacle, the robot should move around it to identify the boundary of the obstacles. Thus when all the obstacles are visited a map representing the boundary of all obstacles will be created. This map is hereafter referred to as the *2D world map* of the robot. Two procedures are given below, i.e. `Map Building` and `Traverse Boundary`. In the procedure `Map Building` a linked list structure is used with four fields. The first two fields, x_i, y_i denote the coordinate of the point visited by the robot. The third field points to the structures containing the obstacle to be visited next, and the fourth field denotes the pointer to the next point to be visited on the same obstacle. A schematic diagram depicting the data structure is presented in Fig. 2.2. Another data structure is used in procedure `Map Building` for acquiring the boundary points visited around an obstacle. This structure has three fields, the first two correspond to the x_i, y_i coordinate of one visited point on the obstacle i, while the third field is a pointer which corresponds to the next point on obstacle i. A schematic diagram for this pointer definition is presented in Fig. 2.3.

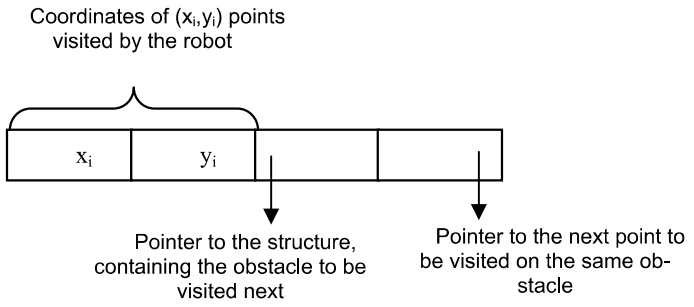


Fig. 2.2. Definition of one structure with two pointers, used for acquiring the list of visited obstacles

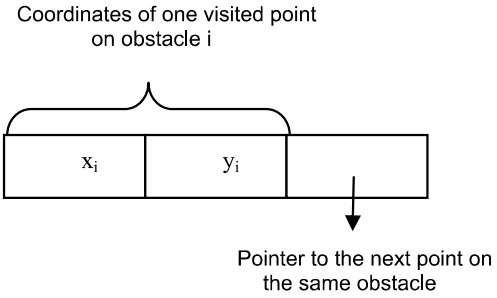


Fig. 2.3. Definition of another structure with one pointer for acquiring the boundary points visited around an obstacle

Procedure Traverse Boundary (current-coordinates)
Begin
Initial-coordinate = current-coordinate;
Boundary-coordinates:= Null;
Repeat
Move-to (current-coordinate) and mark the path of traversal;
Boundary-coordinates: = Boundary-coordinates ∪ {current-coordinate};
For (all possible unmarked set of point P)
Select the next point p ε P, such that
The perpendicular distance from the next point p to
Obstacle boundary is minimum;
Endfor
current-coordinate := next-coordinate;

Until current-coordinate = initial-coordinate
Return Boundary-coordinates;
End.

The above algorithm is self-explanatory and thus needs no elaboration. An algorithm for map building is presented below, where the procedure `Traversal Boundary` has been utilized.

Procedure Map Building (current-coordinate)
Begin
Move-to(current-coordinate);
Check-north-direction( );
If (new obstacle found) **Then do**
Begin
Current-obstacle = Traverse-boundary(new-obstacle-coordinate);
Add-obstacle-list (current-obstacle); //adds current obstacle to list//
Current-position = find-best-point (current-obstacle) // finding the best take off point from the current obstacle//
Call Map-building (current-position);
End
Else do Begin
Check-north-east-direction ( );
If (new obstacle found) **Then do**
Begin
Current-obstacle = Traverse-boundary(new-obstacle-coordinate);
Add-obstacle-list (current-obstacle);
Current-position = find-best-point (current-obstacle);
Call Map-building (current-position);
End;
Else do Begin
Check east direction( );
//Likewise in all remaining directions//
End
Else backtrack to the last takeoff point on the obstacle (or the starting point);
End.

Procedure `Map Building` is a recursive algorithm that moves from an obstacle to the next following the depth-first traversal criteria. The order of preference of visiting the next obstacle comes from the prioritization of the

directional movements in a given order. The algorithm terminates by backtracking from the last obstacle to the previous one and finally to the starting point.

2.2.2 Explanation of the Algorithm

The procedure `Map Building` gradually builds up a tree, the nodes of which denote the obstacles/boundary visited. Each node in the tree keeps a record of the boundary pixel coordinates. The procedure expands the node satisfying the well-known depth-first strategy. For instance, let n_i be the current root node. The algorithm first checks whether there exists any obstacle in the north direction. In case it exists, the algorithm allows the robot to move to an obstacle to the north of n_i, say at point n_j. The boundary of the obstacle is next visited, by the robot, until it reaches the point n_j. The algorithm then explores the possibility of another obstacle to the north of n_j, and continues so until no obstacle is found to the north of a point, say n_k on an obstacle. Under this circumstance only, the algorithm checks for possible obstacle to the north-east of point n_k. It is thus clear how depth-first search has been incorporated in the proposed algorithm. It needs to be pointed out that once the robot visits the boundary of an obstacle, the corresponding pixel-wise boundary descriptors are saved in an array.

The algorithm terminates when it reaches an obstacle at a point n_i and moves around its boundary but couldn't trace any obstacle in any of the possible eight direction around n_i. It then backtracks to the nodes, from where it visited node n_i. Let n_j be that parent node of n_i in the tree. Again if there exist no obstacles in any of the possible eight directions around n_j, then it backtracks to the parent of n_j. The process of backtracking thus continues until it returns to the starting point. The following properties envisage that the proposed algorithm is complete and sound.

Property 1: The procedure `Map Building` is complete.

Proof: By completeness of the algorithm, we mean that it will visit all obstacles and the floor boundary before termination.

Let us prove the property by the method of contradiction, i.e. there remains one or more obstacles before the termination of the algorithm. Now, since the algorithm has been terminated, there must be several backtrackings from n_m to n_k, n_k to n_j, n_j to n_i and so until the root node of the entire tree is reached. This can only happen if no points on an

unvisited obstacle is along any of the eight possible directions of the points on all visited obstacles. The last statement further implies that the unvisited obstacle, say O_X, is completely surrounded by other unvisited obstacles O_K such that $\cup\ O_k$ covers the entirety of O_K. Now, this too can only happen, if each of O_K is surrounded fully by other obstacles O_j. If the process continues this way, then the entire floor space will ultimately be covered by all unvisited obstacles, which however is a contradiction over the initial premise. So the initial premise is wrong, and hence the property follows.

Property 2: The procedure `Map Building` is sound.

Proof: The algorithm never generates a node corresponding to a point in a free space, as the algorithm visits the boundaries of the obstacles only, but not any free space. Hence the proof is obvious.

2.2.3 An Illustration of Procedure `Traverse Boundary`

This example illustrates how a robot moves around an obstacle by utilizing the procedure `Traverse Boundary`. Consider the rectangular obstacle and robot (encircled R) on the north side of the obstacle (shown in Fig. 2.4).

Let the robot's initial coordinates be $x = 75$, $y = 48$ (measured in 2D screen pixels). The sensor information from the robot's location in all directions is stored in an array as

N	NE	E	SE	S	SW	W	NW
(75,43)	(80,43)	(80,48)	(80,50)	(75,50)	(70,50)	(70,48)	(70,43)

The shaded portions are obstacle regions. So we choose the location (70, 48) (in the west direction) as the next location to move, since it is an obstacle-free point and it is next to an obstacle region. So the robot will move to a step ahead in the west direction.

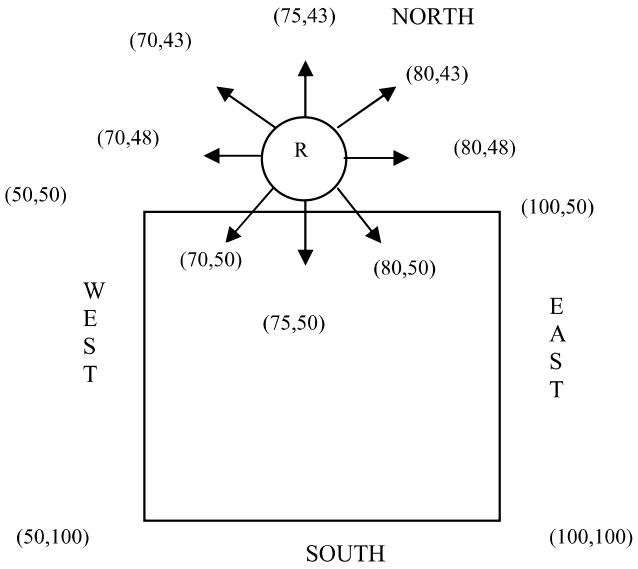


Fig. 2.4. The robot near a rectangular obstacle showing the sensory information in the eight specified directions

Let us consider the robot at another position (48, 48) shown in Fig. 2.5. The sensory information of the robot at this new location is given as follows.

N	NE	E	SE	S	SW	W	NW
(48,43)	(53,43)	(53,48)	(50,50)	(48,53)	(43,53)	(43,48)	(43,43)

From the above table, it is clear that the location (48, 53) is in the south direction, which is obstacle-free and it is next to the obstacle location (53, 53). So the next point to move is (48, 53) in the south direction. Likewise we repeat this process until we reach the initial coordinates (75, 48). The coordinates of the boundary of the obstacle are stored in a linked list, which is maintained in a general structure. Now the next task is to build the total map with the help of this boundary traversing algorithm, which is illustrated below.

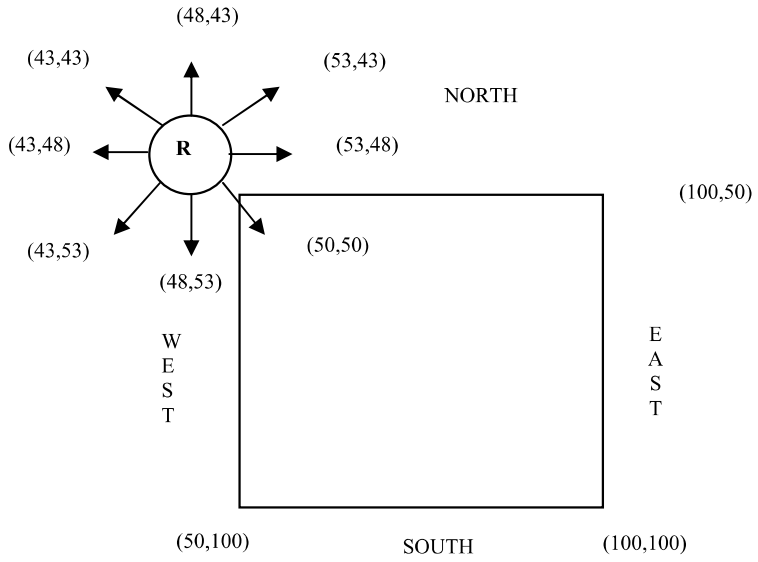


Fig. 2.5. The robot near a square-type obstacle at another location and the sensor information in the simulation

2.2.4 An Illustration of Procedure `Map Building`

An example is given here for creation of a linked list to record the visited obstacles and their boundaries of an environment shown in Fig. 2.6.

The searching process is started from the north direction of robot. If any obstacle is found, the robot will move to that obstacle and record the boundary coordinate information of that obstacle. Again it will start searching in the north direction from the recently visited obstacle. In this way it will go as deep as possible in the north direction only. If no new obstacle is available in the north direction, the robot will look for other directions, in order, for new obstacles. If any new one is found, the robot will visit it and move as deep as possible in the newly found direction. If in any case it cannot find any new obstacle, it will back-track to its parent obstacle and start looking in other directions, as shown in Fig. 2.7.

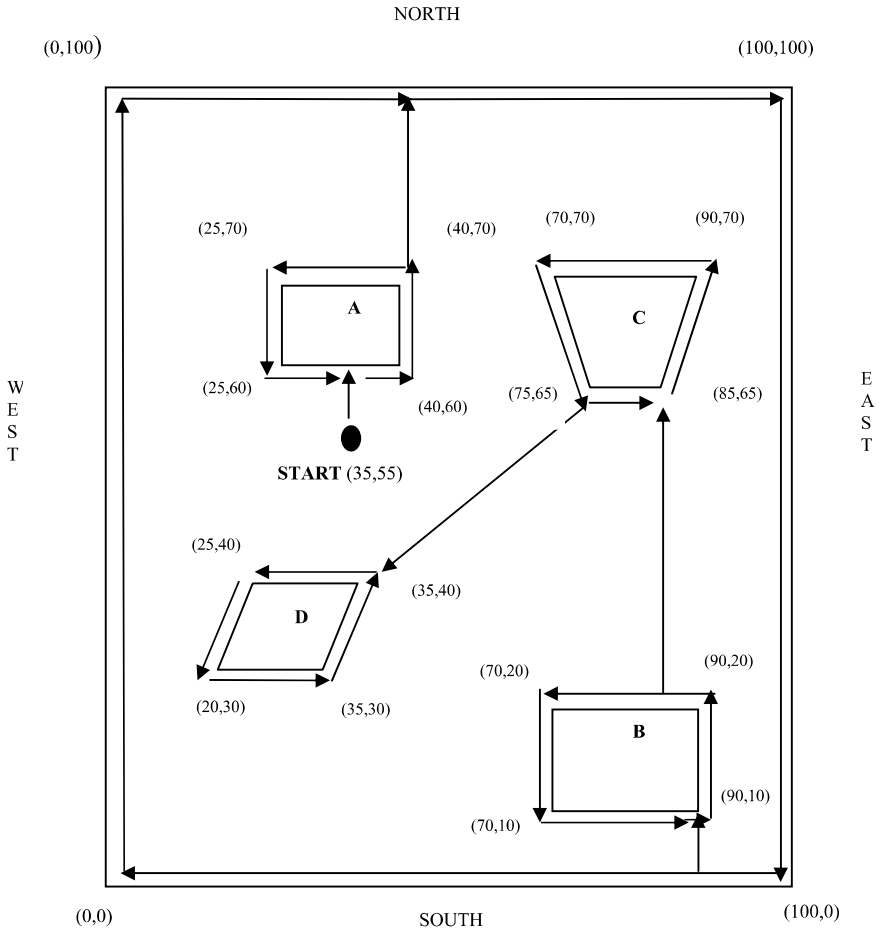


Fig. 2.6. The path traveled by the robot while building the 2D world map using depth-first traversal. The obstacles are represented with literals and the coordinates of the obstacles and the workspace is shown inside the braces

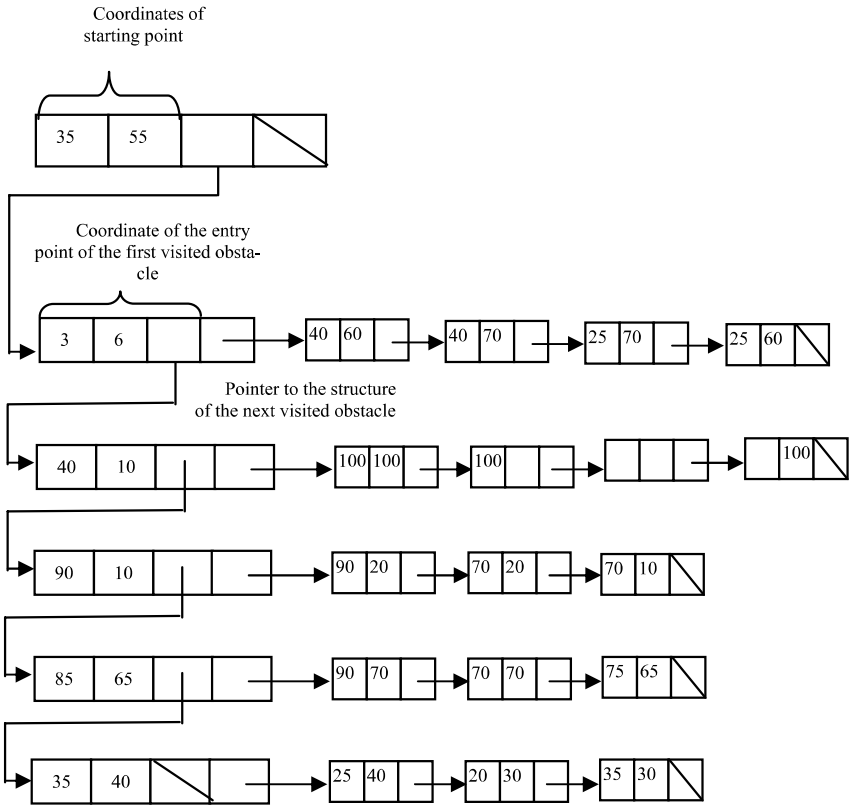


Fig. 2.7. The linked list created to record the visited obstacles and their boundary. Here all the points on the boundary visited by the robot have not been shown in the linked list in order to maintain clarity

2.2.5 Robot Simulation

The algorithm for map building has been simulated and tested by a C++ program. An artificial workspace has been created with nine obstacles along with a closed room, which is shown in Fig. 2.8. The workspace dimension is fixed by four corner points having coordinates (80, 80), (400, 80), (400, 400) and (80, 400) in a (640, 480) resolution screen. The dimensions of the obstacles, described by their peripheral vertices, are as follows:
Obstacle 1: (140,120), (170,100), (185,120), (175,140)
Obstacle 2: (240,120), (270,140), (225,164), (210, 135)

Obstacle 3: (178,160), (280,180), (185,200), (170,180)
Obstacle 4: (245,175), (285,200), (258,204), (230,190)
Obstacle 5: (310,215), (360,240), (330,270), (298,250)
Obstacle 6: (110,245), (130,225), (180,240), (130,280)
Obstacle 7: (230,258), (270,250), (250,280), (220,280)
Obstacle 8: (220,320), (230,300), (250,330), (230,340)
Obstacle 9: (190,330), (210,350), (180,370), (170,350)

The source code is available in Listing 2.1 at the website of the book. The dimension of the soft mobile object is 10 pixels in diameter. The soft object starts at position (100, 380), and moves as per the map building algorithm, which is displayed in Fig. 2.9 and the simulation results are shown in Fig. 2.10.

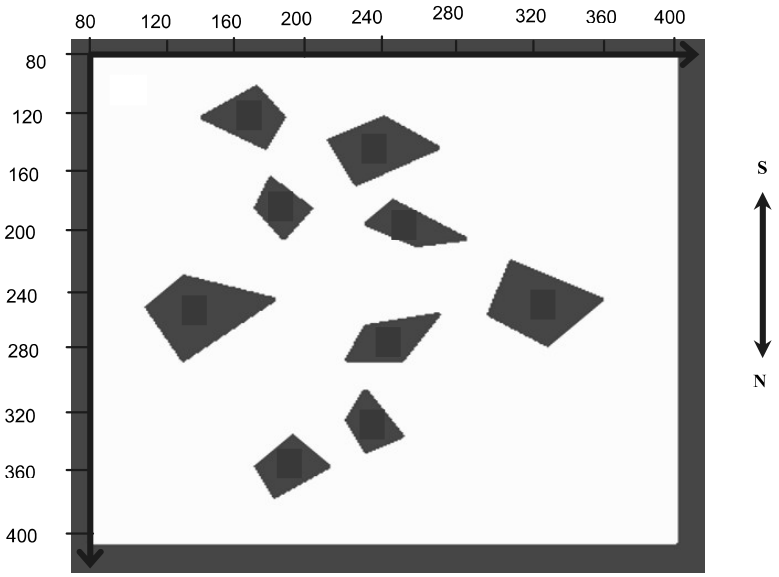


Fig. 2.8. A closed room workspace with nine convex obstacles

2.3 Execution of the Map Building Program

Following are the instructions to be followed for running this program.

```
Enter the starting X_position of Robot (80-400): 100
(enter)
Enter the starting Y_position of Robot (80-400): 380
(enter)
```

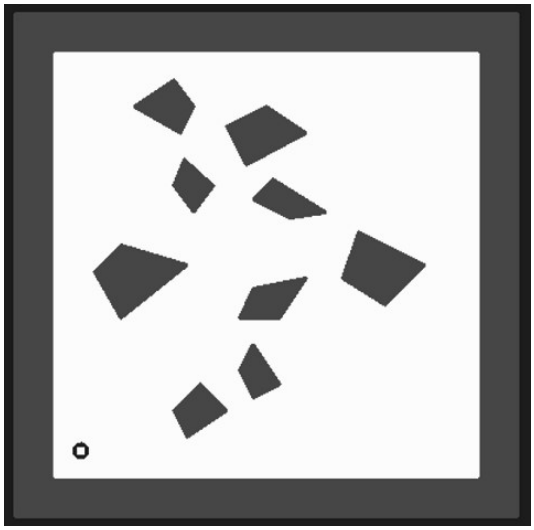
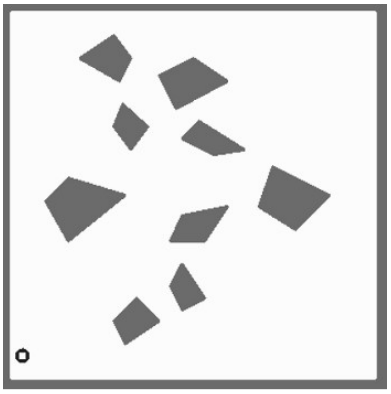
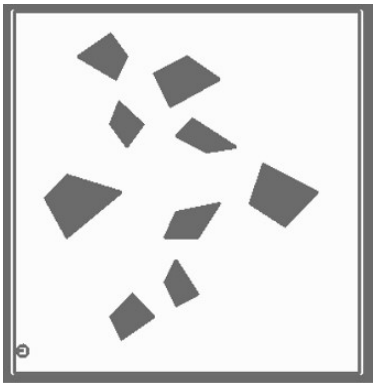


Fig. 2.9. The workspace along with the robot position

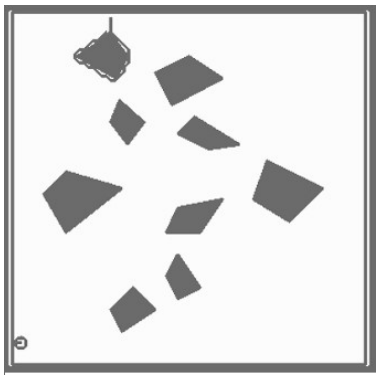
The simulation results given in Fig. 2.10 and the coordinates of the boundaries visited by the robot are given subsequently.



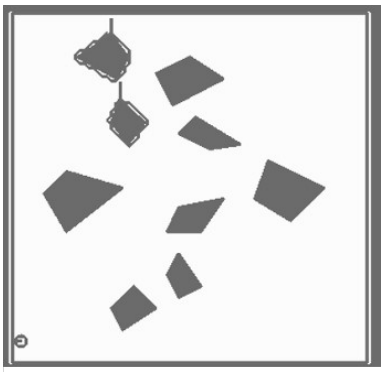
(i): Showing starting position of Robot



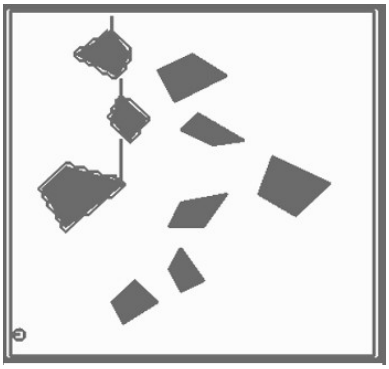
(ii): After visiting First Obstacle.



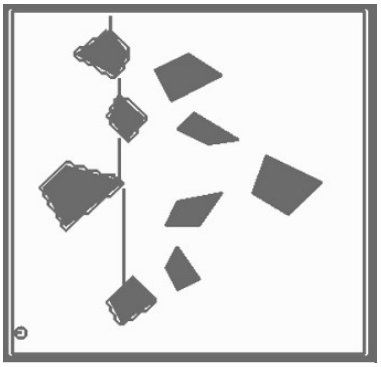
(iii): After visiting 2nd Obstacle



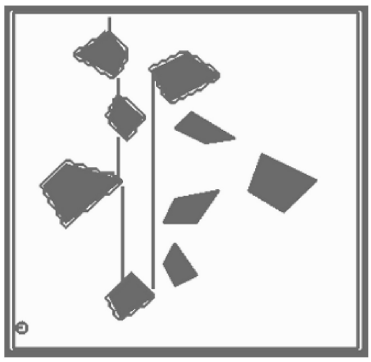
(iv): After visiting 3rd Obstacle



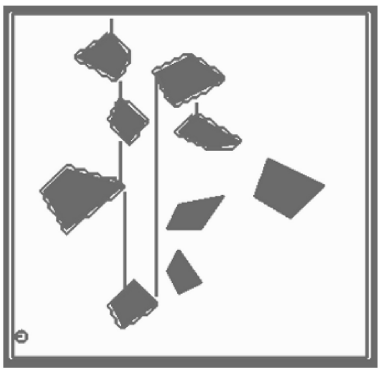
(v): After visiting 4th Obstacle



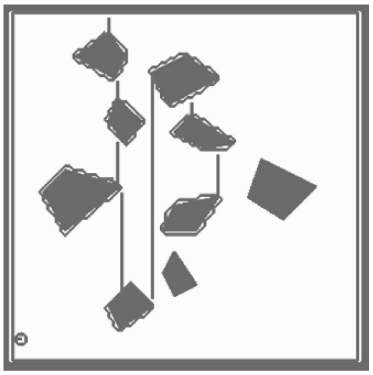
(vi): After visiting 5th Obstacle



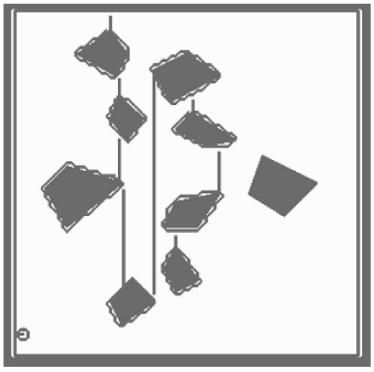
(vii): After visiting 6th Obstacle



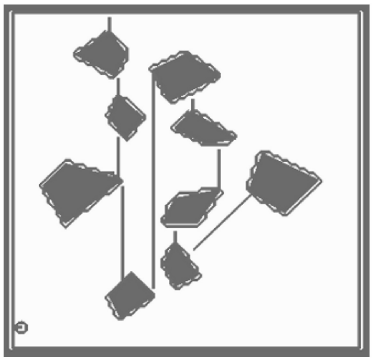
(viii): After visiting 7th Obstacle



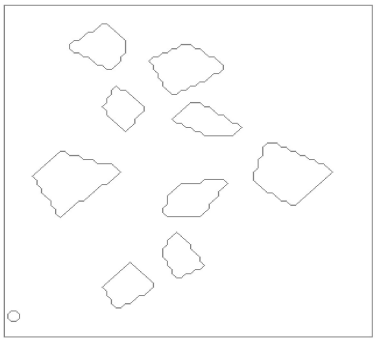
(ix): After visiting 8th Obstacle



(x): After visiting 9th Obstacle



(xi): After visiting 10th Obstacle



(xii): Silhouette of the workspace

Fig. 2.10. Experimental results in a closed workspace containing nine convex obstacles

Coordinates of the boundaries visited by the robot:

The coordinates of the boundaries of the obstacles and the room traversed by the soft object (robot) has been recorded in a file `c:\coord.dat` by the program. The contents of the file are given below.

```
(100,398)(104,398)(108,398)(112,398)(116,398)(120,398)(
124,398)(128,398)(132,398)(136,398)(140,398)(144,398)(1
48,398)(152,398)(156,398)(160,398)(164,398)(168,398)(17
2,398)(176,398)(180,398)(184,398)(188,398)(192,398)(196
,398)(200,398)(204,398)(208,398)(212,398)(216,398)(220,
398)(224,398)(228,398)(232,398)(236,398)(240,398)(244,3
98)(248,398)(252,398)(256,398)(260,398)(264,398)(268,39
8)(272,398)(276,398)(280,398)(284,398)(288,398)(292,398
)(296,398)(300,398)(304,398)(308,398)(312,398)(316,398)
(320,398)(324,398)(328,398)(332,398)(336,398)(340,398)(
344,398)(348,398)(352,398)(356,398)(360,398)(364,398)(3
68,398)(372,398)(376,398)(380,398)(384,398)(388,398)(39
2,398)(396,398)(400,398)(400,394)(400,390)(400,386)(400
,382)(400,378)(400,374)(400,370)(400,366)(400,362)(400,
358)(400,354)(400,350)(400,346)(400,342)(400,338)(400,3
34)(400,330)(400,326)(400,322)(400,318)(400,314)(400,31
0)(400,306)(400,302)(400,298)(400,294)(400,290)(400,286
)(400,282)(400,278)(400,274)(400,270)(400,266)(400,262)
(400,258)(400,254)(400,250)(400,246)(400,242)(400,238)(
400,234)(400,230)(400,226)(400,222)(400,218)(400,214)(4
00,210)(400,206)(400,202)(400,198)(400,194)(400,190)(40
0,186)(400,182)(400,178)(400,174)(400,170)(400,166)(400
,162)(400,158)(400,154)(400,150)(400,146)(400,142)(400,
138)(400,134)(400,130)(400,126)(400,122)(400,118)(400,1
14)(400,110)(400,106)(400,102)(400,98)(400,94)(400,90)(
400,86)(400,82)(396,82)(392,82)(388,82)(384,82)(380,82)
(376,82)(372,82)(368,82)(364,82)(360,82)(356,82)(352,82
)(348,82)(344,82)(340,82)(336,82)(332,82)(328,82)(324,8
2)(320,82)(316,82)(312,82)(308,82)(304,82)(300,82)(296,
82)(292,82)(288,82)(284,82)(280,82)(276,82)(272,82)(268
,82)(264,82)(260,82)(256,82)(252,82)(248,82)(244,82)(24
0,82)(236,82)(232,82)(228,82)(224,82)(220,82)(216,82)(2
12,82)(208,82)(204,82)(200,82)(196,82)(192,82)(188,82)(
184,82)(180,82)(176,82)(172,82)(168,82)(164,82)(160,82)
(156,82)(152,82)(148,82)(144,82)(140,82)(136,82)(132,82
)(128,82)(124,82)(120,82)(116,82)(112,82)(108,82)(104,8
2)(100,82)(96,82)(92,82)(88,82)(84,82)(80,82)(80,86)(80
,90)(80,94)(80,98)(80,102)(80,106)(80,110)(80,114)(80,1
18)(80,122)(80,126)(80,130)(80,134)(80,138)(80,142)(80,
146)(80,150)(80,154)(80,158)(80,162)(80,166)(80,170)(80
```

,174)(80,178)(80,182)(80,186)(80,190)(80,194)(80,198)(8
0,202)(80,206)(80,210)(80,214)(80,218)(80,222)(80,226)(
80,230)(80,234)(80,238)(80,242)(80,246)(80,250)(80,254)
(80,258)(80,262)(80,266)(80,270)(80,274)(80,278)(80,282
)(80,286)(80,290)(80,294)(80,298)(80,302)(80,306)(80,31
0)(80,314)(80,318)(80,322)(80,326)(80,330)(80,334)(80,3
38)(80,342)(80,346)(80,350)(80,354)(80,358)(80,362)(80,
366)(80,370)(80,374)(80,378)(80,382)(80,386)(80,390)(80
,394)(80,398)(84,398)(88,398)(92,398)(96,398)(100,398)
Obstacle Boundary
(168,99)(172,99)(176,103)(180,107)(180,111)(184,115)(18
8,119)(188,123)(184,127)(184,131)(180,135)(180,139)(176
,143)(172,143)(168,139)(164,139)(160,135)(156,131)(152,
131)(148,127)(144,127)(140,123)(136,119)(140,115)(144,1
15)(148,111)(152,107)(156,107)(160,103)(164,99)(168,99)
Obstacle Boundary
(176,161)(180,157)(184,161)(188,165)(192,169)(196,173)(
200,177)(204,181)(200,185)(196,189)(192,193)(192,197)(1
88,201)(184,201)(180,197)(176,193)(172,189)(172,185)(16
8,181)(168,177)(168,173)(172,169)(172,165)(176,161)
Obstacle Boundary
(180,238)(180,242)(176,246)(172,250)(168,254)(164,258)(
160,258)(156,262)(152,266)(148,270)(144,274)(140,274)(1
36,278)(132,282)(128,282)(124,278)(124,274)(120,270)(12
0,266)(116,262)(112,258)(112,254)(108,250)(108,246)(108
,242)(112,238)(116,234)(120,230)(124,226)(128,222)(132,
222)(136,222)(140,226)(144,226)(148,226)(152,230)(156,2
30)(160,230)(164,230)(168,234)(172,234)(176,234)(180,23
8)
Obstacle Boundary
(180,338)(184,334)(188,330)(192,330)(196,334)(200,338)(
204,342)(208,346)(212,350)(208,354)(204,358)(200,362)(1
96,362)(192,366)(188,370)(184,370)(180,374)(176,370)(17
6,366)(172,362)(172,358)(168,354)(168,350)(172,346)(176
,342)(180,338)
Obstacle Boundary
(212,142)(208,138)(208,134)(212,130)(216,130)(220,126)(
224,126)(228,122)(232,122)(236,118)(240,118)(244,118)(2
48,122)(252,126)(256,126)(260,130)(264,134)(268,134)(27
2,138)(272,142)(268,146)(264,146)(260,150)(256,150)(252
,154)(248,154)(244,158)(240,158)(236,162)(232,162)(228,
166)(224,166)(220,162)(220,158)(216,154)(216,150)(212,1
46)(212,142)
Obstacle Boundary
(244,174)(248,174)(252,174)(256,178)(260,182)(264,182)(
268,186)(272,190)(276,190)(280,194)(284,194)(288,198)(2
84,202)(280,206)(276,206)(272,206)(268,206)(264,206)(26

```
0,206)(256,206)(252,206)(248,202)(244,202)(240,198)(236
,198)(232,194)(228,190)(232,186)(236,182)(240,178)(244,
174)
Obstacle Boundary
(268,248)(272,252)(268,256)(268,260)(264,264)(260,268)(
260,272)(256,276)(252,280)(248,284)(244,284)(240,284)(2
36,284)(232,284)(228,284)(224,284)(220,284)(216,280)(22
0,276)(220,272)(220,268)(224,264)(224,260)(228,256)(232
,256)(236,252)(240,252)(244,252)(248,252)(252,252)(256,
248)(260,248)(264,248)(268,248)
Obstacle Boundary
(232,300)(236,304)(240,308)(240,312)(244,316)(248,320)(
248,324)(252,328)(252,332)(248,336)(244,336)(240,340)(2
36,340)(232,344)(228,340)(224,336)(224,332)(220,328)(22
0,324)(216,320)(220,316)(220,312)(224,308)(224,304)(228
,300)(232,300)
Obstacle Boundary
(297,251)(297,247)(297,243)(297,239)(301,235)(301,231)(
301,227)(305,223)(305,219)(305,215)(309,211)(313,215)(3
17,215)(321,219)(325,219)(329,223)(333,223)(337,227)(34
1,227)(345,231)(349,231)(353,235)(357,235)(361,239)(361
,243)(357,247)(353,251)(349,255)(345,259)(341,263)(337,
267)(333,271)(329,271)(325,271)(321,267)(317,267)(313,2
63)(309,259)(305,259)(301,255)(297,251)
```

2.4 Summary

The chapter presents a tool for the representation of the 2D environment of a mobile robot along with a simulation employing a depth-first search strategy.

3 Path Planning

3.1 Introduction

Path planning of mobile robots means to generate an optimal path from a starting position to a goal position within its environment. Depending on its nature, it is classified into offline and online planning. Offline planning determines the trajectories when the obstacles are stationary and may be classified again into various types such as obstacle avoidance; path traversal optimization; time traversal optimization. The obstacle avoidance problems deal with identification of obstacle free trajectories between the starting point and goal point. The path traversal optimization problem is concerned with identification of the paths having the shortest distance between the starting and the goal point. The time traversal optimization problem deals with searching a path between the starting and the goal point that requires minimum time for traversal. Another variant which handles path planning and navigation simultaneously in an environment accommodating dynamic obstacles is referred to as online navigational planning, which we will cover in the next chapter. Depending on the type of planning, the robot's environment is represented by a tree, graph, partitioned blocks, etc. Let us discuss the path planning problem, with suitable structures and representations.

3.2 Representation of the Robot's Environment

Let us first discuss the generalized Voronoi diagram (GVD) representation to find a path from a starting node to a goal node. The GVD describes the free space for the robot's movement in its environment. There exist various approaches to construct the GVD namely the potential field method [Rimon, 1992], two-dimensional cellular automata [Tzionas et al., 1997], and piecewise linear approximation [Takahashi, 1989].