11 Mobile Robot Path, Motion, and Task Planning

11.1 Introduction

Robot planning is concerned with the general problem of figuring out how to move to get from one place to another and how to perform a desired task. As a whole, it is a wide research field in itself. Actually, the term planning means different things to different scientific communities. The three categories of planning in robotics are [1−32]:

1. Path planning
2. Motion planning
3. Task planning

Planning represents a class of *problem solving* which is an interdisciplinary area of system theory and artificial intelligence (AI) [12,19]. A *general problem solver* is basically a search program in which the problem to be solved is defined in terms of a given initial state and a desired goal state. This program guides the search by evaluating the current state and operator (rule) ordering, that is, by applying only the operators that promise the best movement in the search space. The principal problem-solving method is the *means-ends analysis*, which consists in repeated reduction of the difference between the goal state and the current state. This needs a special feedback control strategy. An example of the use of means-ends analysis in AI problem solving is the well-known *Towers of Hanoi* puzzle problem. The general problem-solving methodology has limited value for specific problems that can be solved only by skilled experts such as fault diagnosis, decision support, and robot planning.

The objectives of this chapter are as follows:

- To present the general conceptual definition of robot path planning, motion planning, and task planning
- To investigate the path planning problem of mobile robots, including the basic operations and classification of methods
- To study in some detail the model-based mobile robot path planning, including configuration space and road map planning methods
- To discuss mobile robot motion planning presenting the vector fields method and the analytical parameterized method

Introduction to Mobile Robot Control. DOI: http://dx.doi.org/10.1016/B978-0-12-417049-0.00011-0

- To show how global and local path planning can be integrated such that to achieve the desired features of path smoothness and trapping avoidance
- To outline the basic issues of fixed and mobile robot task planning, including plan representation and generation, and the three phases of task planning, namely, world modeling, task specification, and robot program synthesis.

11.2 General Concepts

Path planning is a general capability embedded in all kinds of robots (robotic manipulators, mobile robots/manipulators, humanoid robots, etc.). In a broad sense, robot path planning is concerned with the determination of how a robot will move and maneuver in a workspace or environment in order to achieve its goals. The path planning problem involves computing a collision-free path between a start position and a goal position. Very often, besides the obstacle avoidance, the robot must also satisfy some other requirements or optimize certain performance criteria. Path planning is distinguished according to the knowledge available about the environment (i.e., fully known/structured environment, partially known environment, and fully unknown/unstructured environment). In most practical cases, the environment is only *partially known*, where the robot, prior to path planning and navigation, has already knowledge of some areas within the workspace (i.e., areas likely to pose local minima problems). The nature of an obstacle is described via its configuration which may be *convex* shaped or *concave* shaped or both. The status of an obstacle may be *static* (when its position and orientation relative to a known fixed coordinate frame is invariant in time), or *dynamic* (when either its position), or orientation or both change relative to the fixed coordinate frame change.

Path planning may be either *local* or *global*. Local path planning is performed while the robot is moving, taking data from local sensors. In this case, the robot has the ability to generate a new path in response to the changes of the environment. Global path planning can be performed only if the environment (obstacles, etc.) is static and perfectly known to the robot. In this case, the path planning algorithm produces a complete path from the start point to the goal point before the robot starts its motion.

Motion planning is the process of selecting a motion and the corresponding inputs such that to assure that all constraints (obstacle avoidance, risk avoidance, etc.) are satisfied. Motion planning can be considered as a set of computations which provide subgoals or set points for the control of the robot. These computations and the resulting plans are based on a suitable model of the robot and the environment in which it is moved. The process by which the robot executes (follows) the planned motion is the control process studied in Chapters 5−10.

We recall that the motion of a robot can be described in three different spaces:

1. Task or Cartesian space
2. Joints' (motors') space
3. Actuators' space

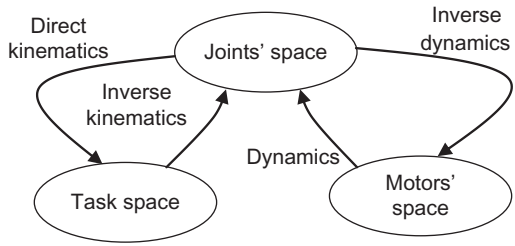


Figure 11.1 Robot motion spaces and their relationships.

The relation of these spaces is pictorially shown in Figure 11.1.

The description of a motion in the task space, that is, the specification of a reference point (e.g., the tip of the end effector) is typically made in the Cartesian world coordinate frame.

The specification of the end effector or mobile robot position is not always sufficient to determine the positions of all the links. For this reason, we use the *joints' space* which is the Cartesian product of the allowable ranges of all the degrees of freedom (usually a subset of n). Finally, for each robot motion that is compatible with the kinematic and dynamic constraints, there must exist at least one set of forces/torques that produce that motion. The forces/torques of the actuators (motors) that generate all the allowable motions define the *motors' space*. A basic issue in robot motion planning is the presence of obstacles and the resulting requirement to find an *obstacle-free* path. This is the *path planning* or *path finding* problem. The need for motion planning comes from the fact that there is a very large number (theoretically, an infinite number) of motions via which the robot can go to a goal pose and execute a desired task. Also, for a given motion, there may be more than one inputs (forces/torques) of the motors that produce the desired motion.

Task planning involves three phases [8]:

1. World modeling
2. Task specification
3. Robot program synthesis

World modeling: A world model must involve a geometric description of robots and objects in the environment (which is usually embodied in a CAD system), a physical description of objects (e.g., mass and inertia of parts), a kinematic description of robot body and linkages, and a description of robot features (e.g., joint limits, maximum possible or allowed acceleration, and sensor features).

Task specification: The task planner usually receives the tasks as a sequence of models of the world state at several steps of the task's execution. Actually, a task specification is a model of the world accompanied by a sequence of changes in the positions of the model components.

Robot program synthesis: This is the most important phase for the task planner to be successful. The program synthesized must include grasp commands, motion properties, sensor commands, and error tests. This implies that the program must be in the robot-level programming language of the robot.

11.3 Path Planning of Mobile Robots

11.3.1 Basic Operations of Robot Navigation

Path planning of a mobile robot is one of the basic operations needed to implement the navigation of the robot. These operations are as follows:

- Self-localization
- Path planning
- Map building and map interpretation

Robot localization provides the answer to the question "where am I?" The *path planning* operation provides the answer to the question "how should I get to where I' am going?" Finally, the *map building/interpretation* operation provides the geometric representation of the robot's environment in notations suitable for describing locations in the robot's reference frame. Vision-based navigation employs optical sensors including laser-based range finders and CCD cameras by which the visual features needed for the localization in the robot's environment are extracted.

Actually, to date, there is no generic method for mobile robot positioning (localization). The specific techniques that exist are divided into two categories:

1. Relative localization methods
2. Absolute localization methods

Because no single, globally good localization method is available, designers of autonomous guided vehicles (AGVs) and autonomous mobile robots (AMRs) usually employ some combination of methods, one from each category.

Relative localization is performed by odometry or inertial navigation. The first uses encoders to measure wheel rotation and/or steering angle. Inertial navigation employs gyroscopes (or accelerometers in some cases) to measure the rate of rotation and the angular acceleration.

Absolute localization uses the following:

- *Active beacons*, where the absolute position of the mobile robot is computed by measuring the direction of incidence of three or more transmitted beacons. The transmitters use light or radio frequencies and are placed at known positions in the environment.
- *Recognition of artificial landmarks*, which are placed at known locations in the environment and are designed so as to provide maximal detectability even under bad environmental conditions.
- *Recognition of natural landmarks*, that is, distinctive features of the environment, which must be known in advance. This method has lower reliability than the artificial landmarks method.
- *Model matching*, that is, comparison of the information received from on-board sensors and a map of the environment. The absolute location of the robot can be estimated if the sensor-based features match the world model map. The robot navigation maps are

distinguished in *geometric maps* and *topological maps*. The first category represents the world in a global coordinate frame, whereas the second category represents the world as a network of arcs and nodes.

11.3.2 Classification of Path Planning Methods

Path planning is a robotics field on its own. Its solution gives a feasible collision-free path for going from one place to another. Humans do path planning without thinking how it is done. If there is an obstacle ahead that has not been there before, humans just pass it. Very often, the human needs to change his/her pose in order to go through a narrow passage. This is a simple type of the so-called *piano-mover's problem*. If the obstacle blocks the way completely, humans just use another way. To perform all the above operations, a robot must be equipped with suitable high-level intelligence capabilities.

A very broad classification of free (obstacle-avoiding) path planning involves three categories, which include six distinct strategies. These are the following:

1. *Reactive control* ("Wander" routine, circumnavigation, potential fields, motor schemas)
2. *Representational world modeling* (certainty grids)
3. *Combinations of both* (vector field histogram)

In many cases, the above techniques do not assure that a path is found that passed obstacles although it exits, and so they need a higher level algorithm to assure that the mobile robot does not end up in the same position over and over again. In practice, it may be sufficient that the robot detects that it is "stuck" despite the fact that a feasible path way exists, and calls for help. In indoor applications, a maneuver for avoiding an obstacle is a good action. Outdoor situations are more complex, and more advanced perception techniques are needed (e.g., for distinguishing a small tree from an iron pole).

A research topic receiving much attention over the years is the *piano-mover's problem*, which is well known to most people that tried a couch or big table through a narrow door. The object has to be tilted and moved around through the narrow door. One of the first research works on this problem is described in Latombe [1].

On the basis of the way the information about the robot's environment is obtained, most of the path planning methods can be classified into two categories:

1. Model-based approach
2. Model-free approach

In the first category, all the information about the robot's workspace are prelearned, and the user specifies the geometric models of objects and a description of them in terms of these models. In the model-free approach, some of the information about the robot's environment is obtained via sensors (e.g., vision, range, touch sensors). The user has to specify all the robotic motions needed to accomplish a task.

11.4 Model-Based Robot Path Planning

The obstacles that may exist in a robotic work environment are distinguished into *static obstacles* and *moving obstacles*. Therefore, two types of path finding problems have to be solved, namely:

- Path planning among stationary obstacles
- Path planning among moving obstacles

The path planning methodology for stationary obstacles is based on the configuration space concept, and is implemented by the so-called *road map planning methods* discussed below.

The path planning problem for the case of moving obstacles is decomposed into two subproblems:

1. Plan a path to avoid collision with static obstacles.
2. Plan the velocity along the path to avoid collision with moving obstacles.

This combination constitutes the *robot motion planning* [2−5].

11.4.1 Configuration Space

Configuration space is a representation in which path planning for both manipulator robots and (most of) mobile robots is performed. For example, if the mobile system is a free-flying rigid body (i.e., a body that can move freely in space in any direction without any kinematics constraint), then six configuration parameters are required to determine fully its position, namely, x, y, z and three directional (Euler) angles. Path planning finds a path in this six-dimensional space. But, actually, a robot is not a free-flying robot. Its possible motion depends on its kinematic structure, for example, a unicycle-like robot has three configuration parameters: x, y, and ϕ. As we have seen, very often, these parameters are not independent, for example, the robot may or may not be able to turn on the spot (change ϕ while keeping x and y fixed), or be able to move sideway. A robotic arm which has n rotational joints needs n configuration parameters to specify its configuration in space, in addition to constraints such as the minimum or maximum values of each angular join. For example, a typical car-like robotic manipulator has 10 configuration parameters (4 for the mobile platform with the trailer, and 6 for the arm), whereas a certain humanoid robot such ASIMO or HRP may have 52 configuration parameters (2 for the head, 7 for each arm, 6 for each leg, and 12 for each hand that involves 4 fingers with 3 articulations each).

Now, given a robot with n configuration parameters moving in a certain environment, we define the following:

- The configuration $\mathbf{q}$ of the robot, that is, an n-tuple of real numbers that specifies the n parameters needed to determine the position of the robot in physical space.
- The configuration space CS of the robot, that is, the set of values that its configuration $\mathbf{q}$ may take.

- The free configuration space CS_{free}, that is, the subset of CS of configurations that are not in collision with the obstacles existing in the robot's environment.

It is noted that the degrees of freedom of a mobile robot are its control variables (a robot arm or a humanoid robot has as many degrees of freedom as configuration parameters, but a differential drive robot has three configuration parameters and only two degrees of freedom).

From the above definitions, it follows that path planning is the problem of finding a path in the free configuration space CS_{free}, between an initial and a final configuration. Thus, if CS_{free} could be determined explicitly, then path planning is reduced to a search for a path in this n-dimensional continuous space. Actually, the explicit definition of CS_{free} is a computationally difficult problem (its computational complexity increases exponentially with the dimension of CS), but there are available efficient probabilistic methods that solve this path planning problem in reasonable time [11]. As we have seen before, these techniques must involve two operations:

1. *Collision checking* (i.e., check whether a configuration $\mathbf{q}$ or a path between two configurations lies entirely in CS_{free})
2. *Kinematic steering* (i.e., find a path between two configurations $\mathbf{q}_0$ and $\mathbf{q}_f$ in CS that satisfies the kinematic constraints, without taking into account obstacles)

Example 11.1 Two-link planar manipulator

Consider the two-link planar manipulator of Figure 11.2A which has two configuration parameters θ_1 and θ_2 (i.e., its CS is two-dimensional).

An obstacle in an n-dimensional configuration space CS is represented by a *slice projection* which is defined by a range of values for one of the defining parameters of CS and an $(n-1)$-dimensional volume. The approximation of the full obstacle is built as the union of a number of $(n-1)$-dimensional slice projections, each for a different range of values of the same joint parameter [8]. In the present case (Figure 11.2A), the obstacles are approximated by a set of θ_2 ranges (which are shown by dark lines) for a set of values of θ_1. A sample path of a two-link robot in another environment with obstacles is shown in Figure 11.2B.

11.4.2 Road Map Path Planning Methods

The robot navigation maps which are used to represent the environment can be a continuous geometric description or a decomposition-based geometric map or a topological map. These maps must be converted to discrete maps appropriate for the path algorithm under implementation. This conversion (or decomposition) can be done by four general methodologies, namely [1,21]:

1. Road maps
2. Cell decomposition
3. Potential fields
4. Vector field histograms

In the following sections, a short description of these methodologies is given.

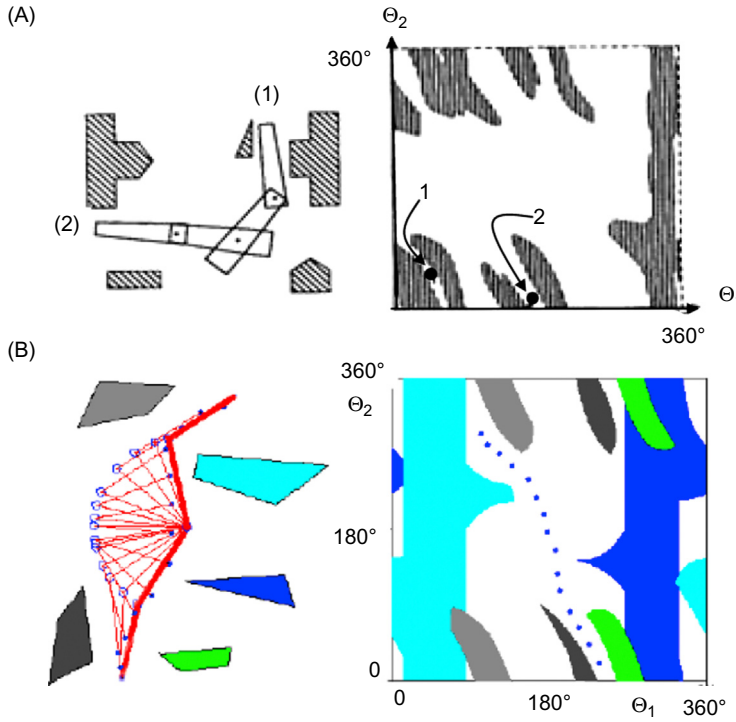


Figure 11.2 Configuration space of a two-link planar robot. (A) The manipulator with the obstacles, and the CS with obstacles approximated by a set of one-dimensional slice projections (shown in dark). (B) Another two-link manipulator in an environment with obstacles, and a possible path in CS_{free}.
Source: http://robotics.stanford.edu/∼latombe/cs326/2009/class3/class3.htm; http://www.cs.cmu.edu/∼motionplanning/lecture/Chap3-Config-Space_howie.pdf.

11.4.2.1 Road Maps

The basic idea is to capture the connectivity of CS_{free} with a road map (graph or network) of one-dimensional curves. After its construction, a road map is used as a network of path (road) segment for the planning of the robot motion. The main goal here is to construct a set of roads that as a whole enable the robot to reach any position in its CS_{free}. This is actually a hard problem.

The path planning problem is stated as follows:

Given input configurations $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$ and the set B of obstacles.
Find a path in CS_{free} connecting $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$.

The basic steps of the path planning algorithm are as follows:

Step 1: Build a road map in CS_{free} (the road map nodes are free or semifree configurations; two nodes are connected by an edge if the robot can easily move between them).

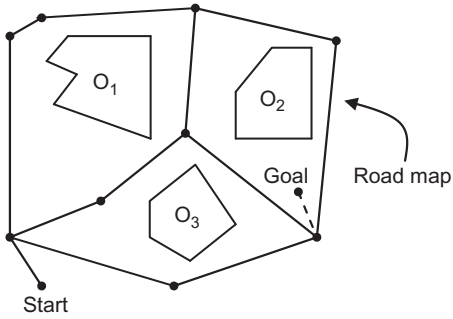


Figure 11.3 A road map example in an environment with obstacles O_1, O_2, O_3.

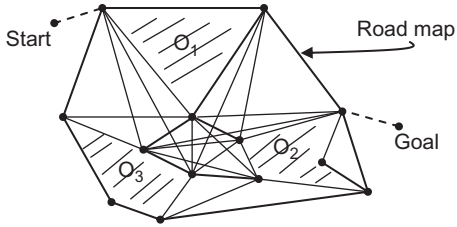


Figure 11.4 A visibility graph example.

Step 2: Connect $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$ to road map nodes υ_{start} and υ_{goal}.
Step 3: Find a path in the road map between υ_{start} and υ_{goal}, which gives directly a path in CS_{free}.

A road map example is shown in Figure 11.3.
The methods for building road maps are distinguished into:

- Traditional deterministic methods (they are suitable only for low-dimensional CSs, they build CS_{free}, and they are complete).
- Modern probabilistic methods (they do not build CS_{free}, they apply to both low- and high-dimensional CSs, but they are not complete).

Visibility graph of CS: A visibility graph for a polygonal CS is an undirected graph G where the nodes in G correspond to vertices of the polygonal obstacles, under the condition that the nodes can be connected by a straight line that lies fully in CS_{free} or by the edges of the obstacles (i.e., under the condition that the nodes can see each other including the initial and goal positions as vertices as well). An example of visibility graph is shown in Figure 11.4.

Mobile path planning based on visibility graphs is popular because of its simplicity. There are efficient algorithms with complexity $O(n^2)$, where n is the number of vertices of the objects or $O(E + n \log n)$, where E is the number of edges in G [17,22]. Visibility graphs are really suitable only for two-dimensional CS. It is noted that they are also methods for constructing "reduced" visibility graphs where not all edges are needed. A visibility graph and a reduced visibility graph (corresponding to it) are shown in Figure 11.5.

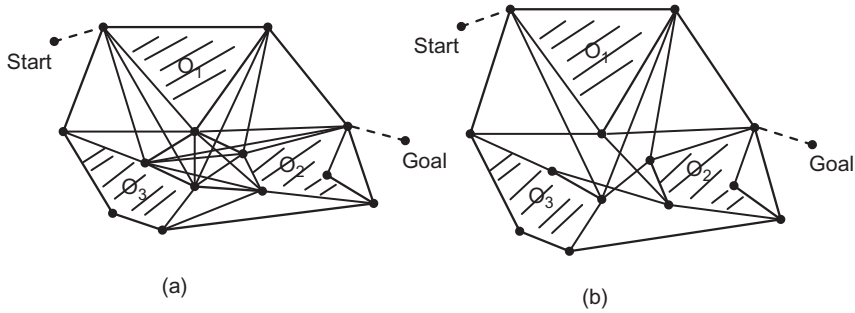


Figure 11.5 (A) A visibility graph. (B) An associated reduced visibility graph.

Voronoi diagram: A Voronoi diagram, after the name of the German mathematician who coined it in 1908, ensures a maximum distance between the robot and the obstacles in the map. The construction steps of a Voronoi diagram are as follows:

Step 1: For each point in the free space CS_{free}, compute its distance to the nearest obstacle.
Step 2: Plot that distance as a vertical height. The height increases as the point is moving away from the obstacle.
Step 3: At equidistant points from two or more obstacles, this distance plot has sharp ridges.
Step 4: Construct the Voronoi diagram by the union of the edges formed by these sharp ridges.

Mathematically, the Voronoi diagram V is a polygonal region defined as:

$$V = \{\mathbf{q} \in CS_{free} : |\text{near}(\mathbf{q})| > 1, \;\; CS = R^2\}$$

where:

$\text{near}(\mathbf{q}) = \{\mathbf{p} \in \delta : ||\mathbf{q} - \mathbf{p}|| = \text{clearance}(\mathbf{q})\}$
$\text{clearance}(\mathbf{q}) = \min\{||\mathbf{q} - \mathbf{p}|| : \mathbf{p} \in \delta\}$ for $\mathbf{q} \in CS_{free}\}$
$\delta = \partial(CS_{free})$, the boundary of CS_{free}

We see that, actually, near($\mathbf{q}$) is the set of boundary points of CS_{free} that minimize the distance to $\mathbf{q}$. The Voronoi diagram V consists of all points in CS_{free} with at least two nearest neighbors in the CS_{free} boundary δ. The Voronoi diagram is a finite collection of straight line segments and parabolic segments (called arcs), where:

- *Straight arcs* are defined by two vertices or two edges of the set of obstacles (i.e., the set of points equally close to two points or two line segments is a line).
- *Parabolic arcs* are defined by one vertex and one edge of the obstacle set (i.e., the set of points equally close to a point (focus) and a line (directrics) is a parabolic segment).

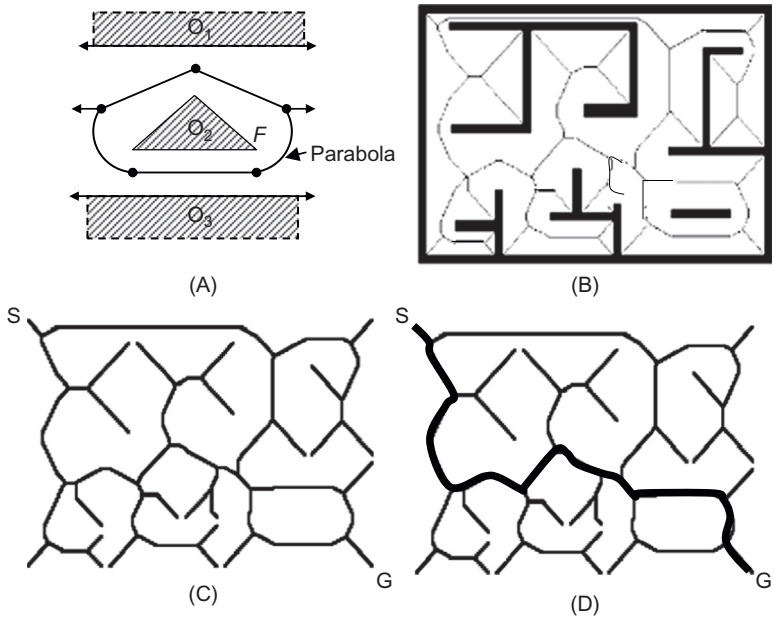


Figure 11.6 (A) An example of Voronoi diagram. (B) Environment (map) of an office floor. (C) Voronoi diagram of the floor. (D) A safe path found by the fast marching method. *Source*: Reprinted from Ref. [33] with the courtesy of S. Garrido and permission from CSC Press.

A construction procedure of V based on the above definition involves the following basic steps:

Step 1: Compute all arcs (for all vertex-vertex, edge-edge, vertex-edge pairs).
Step 2: Compute all intersection points (dividing arcs into segments).
Step 3: Keep the segments which are closest only to the vertices/edges that defined them.

A simple Voronoi diagram example for a terrain with two parallel walls and a triangular object between them is shown in Figure 11.6A. The Voronoi diagram of an office floor that has the map of Figure 11.6B is shown in Figure 11.6C, and a safe path from S to G found by the *fast marching* path planning method is depicted in Figure 11.6D [33−36].

11.4.2.2 Cell Decomposition

Cell decomposition is concerned with the discrimination between cells (i.e., connected geometric areas) in CS that are free (i.e., belong to CS_{free}) or are occupied by objects. The cells have a predefined resolution. After the step of determining the cells, the path planning procedure proceeds as follows:

- Determine the open cells that are adjacent and construct a connectivity graph G.
- Determine the cells that contain the starting and goal configurations and search for a path in G that joins the start and goal cell.

- In the sequence of cells that joins the starting and goal cell, find a path within each cell (e.g., moving via the midpoints of the cell boundaries or moving along the walls).

Cell decomposition is distinguished into:

- Exact cell decomposition
- Approximate cell decomposition

In the first, the boundaries are placed as a function of the environment's structure, and so the decomposition is lossless. In approximate cell decomposition, we obtain some approximation of the actual map. An example of exact cell decomposition is shown in Figure 11.7.

In Figure 11.7A, the boundaries of the cells are found by taking into account the geometric criticality. The path planning is complete because each cell is either completely free or completely occupied. It is clear that the robot can move from each free cell to adjacent free cells.

In approximate cell decomposition, the resulting cells may be free, completely occupied or mixed, or have reached an arbitrary resolution threshold. One way to apply approximate decomposition is to use the *occupancy grid* method. A possible occupancy grid can be obtained by assigning to each of the cells a value that relates to the probability of this cell's occupation. The decomposition threshold in this case defines the minimum required probability for a cell in the occupancy grid to be deemed occupied. Actually, the idea of "approximate" is to fuse neighboring

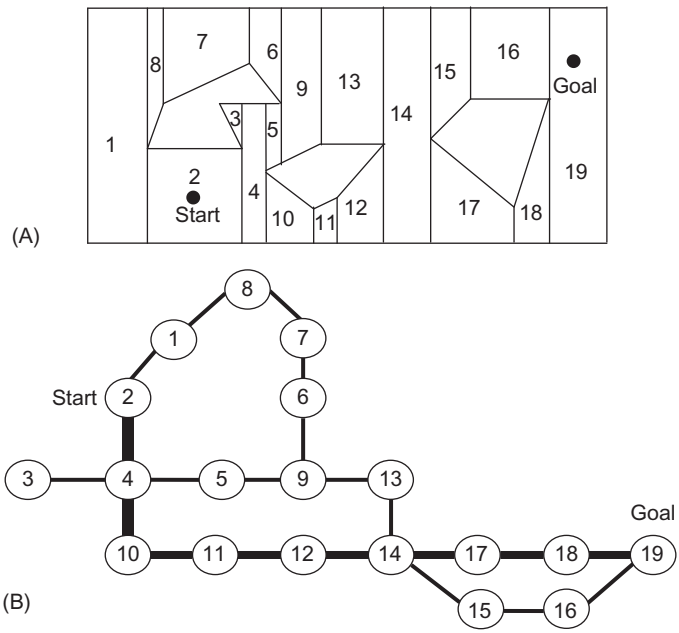


Figure 11.7 (A) Cell decomposition. (B) Network representation and a possible path from the start node to the goal node.

free cells into larger cells to allow a fast path determination. In two-dimensional workspaces, the approximate cell decomposition operation consists in recursively subdividing each cell which contains a mix of free and obstructed space into cells. Because of this property, this method is known as *quadtree method*. The recursion is ended when each cell is found to contain entirely free or obstructed space, or when the maximum desired resolution is reached. The height of the decomposition is the maximum allowable level of recursion, and specifies the resolution of the decomposition.

Figure 11.8 shows an application example of the approximate cell decomposition method. The workspace contains three obstacles and the robot has to move from S to G.

The result of approximate decomposition is a drastic reduction of the number of cells to be considered. In an example, a high resolution map that contains 250,000 cells, with a crude decomposition of height 4 was reduced to just 109 cells [37]. A problem that has to be faced here is to determine *cell adjacency* (i.e., to find which cells share a common border or edge with another one). Actually, many techniques are available to solve the adjacency problem.

One of them is to use *tesseral* (or *quad tesseral*) *addressing* which has the ability to map every part of a 2D (or *n*D) spatial domain into 1D sequence, and when stored with attributes in a database, each address can perform as a single key to data [37,38]. To generate tesseral addresses, the positive quadrant of 2D-Cartesian space is quartered to give parent tiles with labeling as shown in Figure 11.9A. This process can be continued with new tesseral addresses generated by always appending to the right of the parent addresses, until a desired depth is reached.

Another solution to global path planning, via decomposition, is to use local node refinement, path nodes refinement, and curve parametric interpolation [39].

A quad tesseral address can be stored in a quadtree structure. For example, the address of Figure 11.9A (right) can be stored by the quadtree structure of

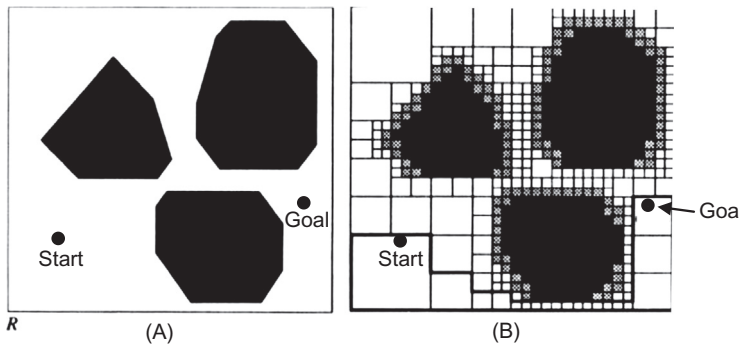


Figure 11.8 (A) A simple three-obstacle path planning problem. (B) An obstacle-free path based on approximate cell decomposition.
Source: http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/1998-99/robotics/basicmotion.html.

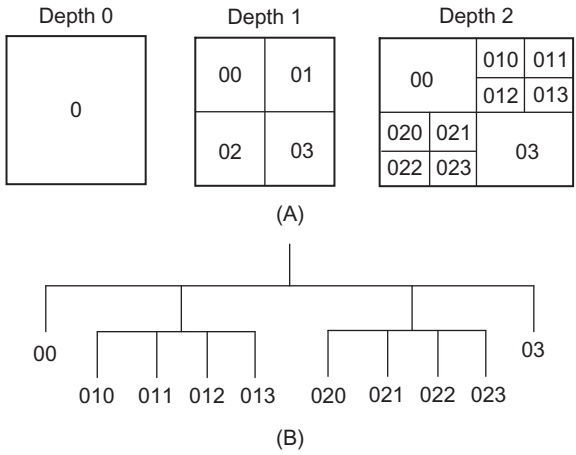


Figure 11.9 (A) 2D tesseral decomposition (addressing). (B) Quadtree structure of the above depth-2 address.

Figure 11.9B, which if traversed from left to right produces a linearization. This important property of tesseral addressing facilitates the storage, comparison, and translation of groups of addresses.

In global path planning, the environment is assumed to be *a priori* known, in which case, a *distance optimal path* from the robot's current position to the goal can be found. This path consists of a sequence of waypoints whose proximity to one another is specified by the decomposition resolution and the configuration of the obstacles in the environment. In a changing environment, one must combine the above global path planning technique with a waypoint driven local path planning method. A distance-optimal global path planning method which uses tesseral addressing is presented in Ref. [37]. In this method, use is made of the connectivity graph to produce *nodes* that represent physical locations in the environment and *arcs* that represent the ability to avoid the obstacles during the motion along the path. The distance-optimal path is found by minimizing the following heuristic cost function using a suitable graph search method such as the A* algorithm [9,11,19]:

$$L_0(N) = L_s(N) + L_g(N)$$

$$L_s(N) = \sum_{i=s+1}^{N} a_i$$

$$L_g(N) = \sqrt{(x_g - x_n)^2 + (y_g - y_n)^2}$$

where a_i is the length of the arc (n_{i-1}, n_i) between two adjacent nodes $i-1$ and i in a sequence $(n_s, n_{s+1}, \ldots, n_{N-1}, n_N)$ of nodes that connect the node s to node N, and $L_g(N)$ is the distance from node N to the goal node G. Clearly, $L_g(N)$ is an underestimate of the optimal cost from N to G.

To carry out the minimization, the so-called *tesseral*[1] *arithmetic* is used. For example, in this arithmetic, addition or subtraction is *tile translation* to the *right* (addition of tesseral 1, binary 01) or to the *left* (subtraction of tesseral 1). Correspondingly, translation *down* and *up* is equivalent to the addition or subtraction of tesseral 2 (binary 10), respectively. It is remarked that actually the resulting optimal path depends on the connectivity graph employed between the initial robot's position and the goal, that is, it is not globally optimal. The computation time of the tesseral arithmetic algorithms is of order $O(4n)$. A critical review of spatial reasoning using quad tesseral representation is provided in Ref. [38]. This representation allows to carry out spatial reasoning (i.e., manipulation of 2D and 3D objects) as if in only one dimension.

11.4.2.3 Potential Fields

The potential field method for path planning is very attractive because of its simplicity and elegance. The idea of potential field is borrowed from nature, for example, a charge particle navigating a magnetic field, or a small ball rolling on a hill. We know that depending on the strength of the field, or the slope of the hill, the particle, or the ball, will go to the source of the field (the magnet), or the valley of this example. In robotics, we can emulate the same phenomenon by developing an artificial potential field that will attract the robot to the goal. In conventional path planning, we calculate the relative position of the robot to the goal, and then apply the necessary forces that will drive the robot to the goal.

If the robot's environment is free of obstacles, we simply create an *attractive field* that goes to the goal. The potential field is defined over the entire free space, and at each time increment, we compute the potential field at the robot position, and then calculate the induced force by this field. The result is that the robot will move according to this force. If the robot's space has an obstacle, we make it generate a *repulsive field* in the surrounding space, which when the robot approaches the obstacle, pushes the robot away from it. If we want to make the robot going to the goal and avoiding the obstacle, then we superimpose an attractive field around the goal and a repulsive field around each obstacle in the robot's space. The above concepts are illustrated in Figure 11.10A–D, where the path of the robot from a certain starting point to the goal is also shown (Figure 11.10E).

In the simplest situation, a mobile robot can be considered to be a *point robot*, in which case the robot's orientation ϕ is not included in the calculations, the potential field is two-dimensional, and the robot's position is simply $\mathbf{q} = [x, y]^{\mathrm{T}}$. In general, a mobile robot has $\mathbf{q} = [x, y, \phi]^{\mathrm{T}}$, and a robotic manipulator has $\mathbf{q} = [q_1, q_2, \ldots, q_n]^{\mathrm{T}}$. Mathematically, the potential field method develops as follows [1,5] (see also [24,25] for more formulations). Define:

- $U(\mathbf{q})$ the total potential field, which is equal to the sum of the attractive potential field $U_{\mathrm{att}}(\mathbf{q})$ and the repulsive potential field $U_{\mathrm{rep}}(\mathbf{q})$.

[1] The term *tesseral* comes from the Greek word $\tau\acute{\varepsilon}\sigma\sigma\varepsilon\rho\alpha$ (tessera=four).

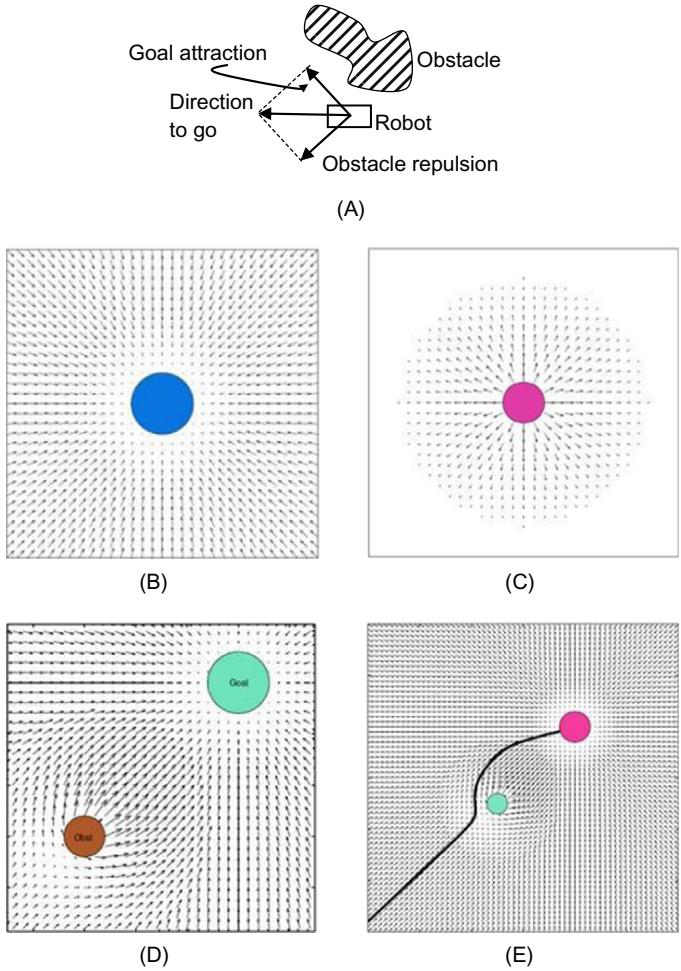


Figure 11.10 Illustration of the potential field path planning method. (A) Goal attraction, obstacle repulsion, and direction to go. (B) Attractive field to the goal. (C) Repulsive field around an obstacle. (D) Combination of the above two fields. (E) A possible robot path. *Source*: http://www.cs.mcgill.ca/∼hsafad/robotics/index.html.

- $\mathbf{F}_{att}(\mathbf{q})$ the attractive force, $\mathbf{F}_{rep}(\mathbf{q})$ the repulsive force, and $\mathbf{F}(\mathbf{q})$ the total force applied to the robot.

  Then, we have:

$$U(\mathbf{q}) = U_{att}(\mathbf{q}) + U_{rep}(\mathbf{q})$$

$$\mathbf{F}(\mathbf{q}) = \mathbf{F}_{att}(\mathbf{q}) + \mathbf{F}_{rep}(\mathbf{q})$$

where:

$$\mathbf{F}(\mathbf{q}) = -\nabla U(\mathbf{q}), \mathbf{F}_{\text{att}}(\mathbf{q}) = -\nabla U_{\text{att}}(\mathbf{q}), \mathbf{F}_{rep}(\mathbf{q}) = -\nabla U_{\text{rep}}(\mathbf{q})$$

with $\nabla[\cdot] = [\partial/\partial q_1, \partial/\partial q_2, \ldots, \partial/\partial q_n]^{\text{T}}$ being the gradient operator.

Now, suppose that $U_{\text{att}}(\mathbf{q})$ is quadratic:

$$U_{\text{att}}(\mathbf{q}) = \frac{1}{2}||\mathbf{q} - \mathbf{q}_{\text{goal}}||^2 = \frac{1}{2}\sum_{i=1}^{n}(q_i - q_{i,\text{goal}})^2$$

In this case, we have:

$$\begin{aligned}\nabla U_{\text{att}}(\mathbf{q}) &= \frac{1}{2}(2||\mathbf{q} - \mathbf{q}_{\text{goal}}||\nabla||\mathbf{q} - \mathbf{q}_{\text{goal}}||)\\ &= ||\mathbf{q} - \mathbf{q}_{\text{goal}}||\frac{(\mathbf{q} - \mathbf{q}_{\text{goal}})}{||\mathbf{q} - \mathbf{q}_{\text{goal}}||}\\ &= \mathbf{q} - \mathbf{q}_{\text{goal}}\end{aligned}$$

Therefore:

$$\mathbf{F}_{\text{att}}(\mathbf{q}) = -(\mathbf{q} - \mathbf{q}_{\text{goal}})$$

which shows that $F_{\text{att}}(\mathbf{q})$ tends linearly to 0 with distance to goal. This is good for stability but tends to infinity when the distance goes far from goal.

If we define $U_{\text{att}}(\mathbf{q})$ as the norm of $\mathbf{q} - \mathbf{q}_{\text{goal}}$, that is:

$$U_{\text{att}}(\mathbf{q}) = ||\mathbf{q} - \mathbf{q}_{\text{goal}}||$$

then:

$$\begin{aligned}\nabla_{\text{att}} U(\mathbf{q}) &= \nabla\left\{\sum_{i=1}^{n}(q_i - q_{i,\text{goal}})^2\right\}^{1/2}\\ &= \frac{1}{2}\left\{\sum_{i=1}^{n}(q_i - q_{i,\text{goal}})^2\right\}^{-1/2}\nabla\left[\sum_{i=1}^{n}(q_i - q_{i,\text{goal}})^2\right]\\ &= (\mathbf{q} - \mathbf{q}_{\text{goal}})/\left[\sum_{i=1}^{n}(q_i - q_{i,\text{goal}})^2\right]^{1/2}\\ &= \frac{(\mathbf{q} - \mathbf{q}_{\text{goal}})}{||\mathbf{q} - \mathbf{q}_{\text{goal}}||}\end{aligned}$$

In this case:

$$\mathbf{F}_{\text{att}}(\mathbf{q}) = -(\mathbf{q} - \mathbf{q}_{\text{goal}})/||\mathbf{q} - \mathbf{q}_{\text{goal}}||$$

which is singular (unstable) at the goal, and tends to 1 far from the goal.

Very often, in practice, we use a composite attractive potential field of the form:

$$U_{\text{att}}(\mathbf{q}) = \begin{cases} \frac{1}{2}\lambda\|\mathbf{q} - \mathbf{q}_{\text{goal}}\|^2 & \text{if} \quad \|\mathbf{q} - \mathbf{q}_{\text{goal}}\| < \varepsilon \\ \mu\|\mathbf{q} - \mathbf{q}_{\text{goal}}\| & \text{if} \quad \|\mathbf{q} - \mathbf{q}_{\text{goal}}\| \geq \varepsilon \end{cases}$$

where λ and μ are scaling factors and ε is a selected distance from the goal. The repulsive potential field is typically selected as:

$$U_{\text{rep}}(\mathbf{q}) = 1/||\mathbf{q} - \mathbf{q}^*||$$

where $\mathbf{q}^*$ is the closest point to the obstacle.

Therefore:

$$\mathbf{F}_{\text{rep}}(\mathbf{q}) = -\nabla U_{\text{rep}}(\mathbf{q}) = (\mathbf{q} - \mathbf{q}^*)/||\mathbf{q} - \mathbf{q}^*||^2$$

For each additional obstacle, we add a corresponding repulsive potential field. If an obstacle is non-convex, we triangulate it into multiple convex obstacle and weigh the separate fields of the object in case the summed repulsive fields are greater than that of the original obstacle. The main problem with potential-field-based path planning is that the robot may be trapped to a local minimum of the field. Actually, several methods exist for avoiding this trapping. Specifically, when the robot goes into a local minimum position, we can correct this situation in one of the following ways [27]:

- Backtrack from the local minimum and use an alternative strategy to avoid the local minimum.
- Perform certain random movements, hoping that they will help escaping the local minimum.
- Use more complex potential fields that are local minimum free (e.g., harmonic potential fields).
- Apply an extra force $\mathbf{F}_{\text{vfs}}$ which is called *virtual free space force*.

Of course, all the above methods assume that the robot can detect that it is trapped, which is also a difficult problem on its own. The virtual free space force is proportional to the amount of free space around the robot [28], and helps to pull the robot away from the local minimum region. In other words, the virtual force drags the robot outside the local minimum, and so the robot can begin again using the potential field planner. Fortunately, it is unlikely the robot to be trapped again

to the same local minimum. The above virtual force concept and its effect is illustrated pictorially in Figure 11.11A–C, where the robot is moved by the total force:

$$\mathbf{F} = \mathbf{F}_{att} + \mathbf{F}_{rep} + \mathbf{F}_{vfs}$$

A hybrid virtual force field method that integrates the virtual force field concept with the virtual obstacle and virtual goal concept is presented in Ref. [40].

One way to detect that the robot is trapped is to employ an open-loop position estimator which estimates the current position of the robot. If the current position does not change for a predefined period of time (time threshold), then the virtual force is generated and applied to the robot for pulling it out from the local minimum. Obviously, due to the repulsive force, the robot velocity is decreased as it approaches an obstacle. A Java source code of a path planning algorithm that uses $\mathbf{F}_{vfs}$ is provided in Ref. [27].

Three other problems that may be encountered in potential-field-based path planning are the following [29]:

1. The robot cannot pass between closely spaced obstacles.
2. Oscillations occur in the presence of obstacle disturbances.
3. Oscillations occur in narrow passages.

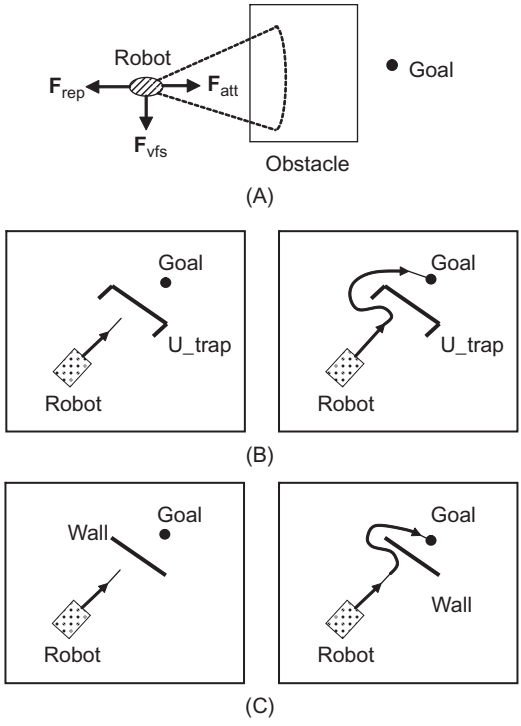


Figure 11.11 (A) The virtual free space force $\mathbf{F}_{vfs}$ drags the robot outside the local minimum area in CS_{free}. (B) $\mathbf{F}_{vfs}$ untraps the robot from a U-trap. (C) Untrapping from a wall trap.

The first situation is illustrated in Figure 11.12A. The two obstacles exert the repulsive forces $\mathbf{F}_{rep,1}$ and $\mathbf{F}_{rep,2}$ giving a total (resultant) repulsive force $\mathbf{F}_{rep}$. Thus, the robot moves under the influence of the goal attractive force $\mathbf{F}_{att}$ and the total obstacles' repulsive force $\mathbf{F}_{rep}$. Their sum:

$$\mathbf{F}_{total} = \mathbf{F}_{att} + \mathbf{F}_{rep}$$

in this case gets the robot away from the passage that leads toward the robot. Of course, depending on the relative magnitude of $\mathbf{F}_{att}$ and $\mathbf{F}_{rep}$, the robot may be moved toward the goal passing through the opening between the obstacles.

In Figure 11.12B, a robot is forced to move alongside a wall which obstructed its path. At a certain point, the wall has a discontinuity which causes the robot to move in an oscillatory mode. Figure 11.12C (right) shows the oscillatory behavior caused when the robot moves to a narrow corridor. This is due to that the robot is subject to two repulsive forces from opposite sides, simultaneously. If the corridor

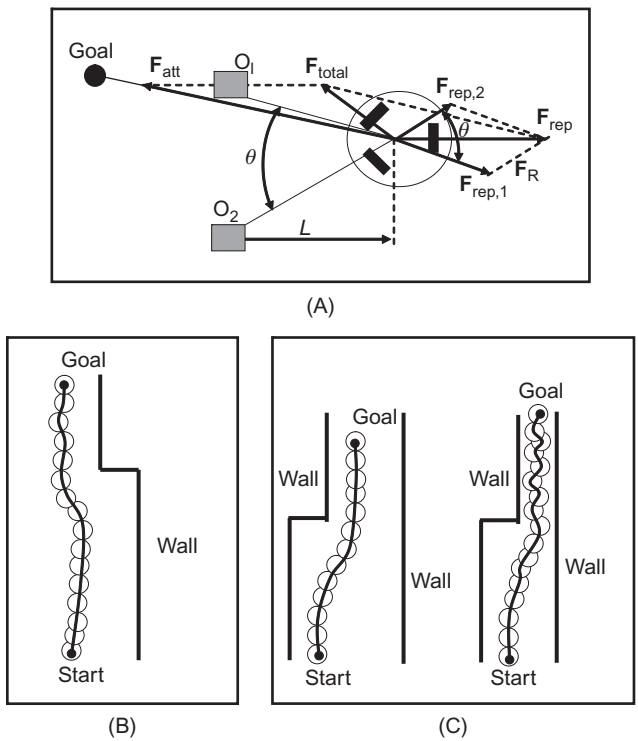


Figure 11.12 (A) A situation in which the robot does not pass through the opening between obstacles. (B) The robot motion enters an oscillatory mode when it encounters an obstacle disturbance. (C) In a wide corridor, the robot manages to move without oscillations (left), but if the corridor is very narrow, the robot exhibits oscillatory motion (right).

is sufficiently wide (Figure 11.12C, left), the robot may manage to get a stable (nonoscillatory) motion. All the above phenomena have been studied analytically by stability theory, and have been observed in practical or simulated experiments [29]. To overcome these limitations of the potential field approach, Koren and Borenstein developed the so-called *vector field histogram* (VFH) method [30], which was further improved in Ref. [31].

11.4.2.4 Vector Field Histograms

In this method, the obstacle-free path planning is performed with the aid of an intermediate data structure about the local obstacle distribution, called *polar histogram* which is an array of, say, 72 (5° wide) *angular sectors* (Figure 11.13). To take into account the robot changing position and the new sensor readings, the polar histogram is totally updated and rebuilt every, say, 30 ms (sampling period). The method involves two steps [30]:

1. The histogram grid is reduced to one-dimensional polar histogram which is built around the robot's instantaneous location (Figure 11.15A). Each sector in the polar histogram involves a value that represents the *polar obstacle density* (POD) in this direction.
2. The most suitable sector from among all polar histogram sectors with a low POD is selected, and the robot moves in that direction.

To implement these steps, a window *(called active window)* moves with the robot, overlying a square region of cells (e.g., 33×33 cells) in the histograms. All

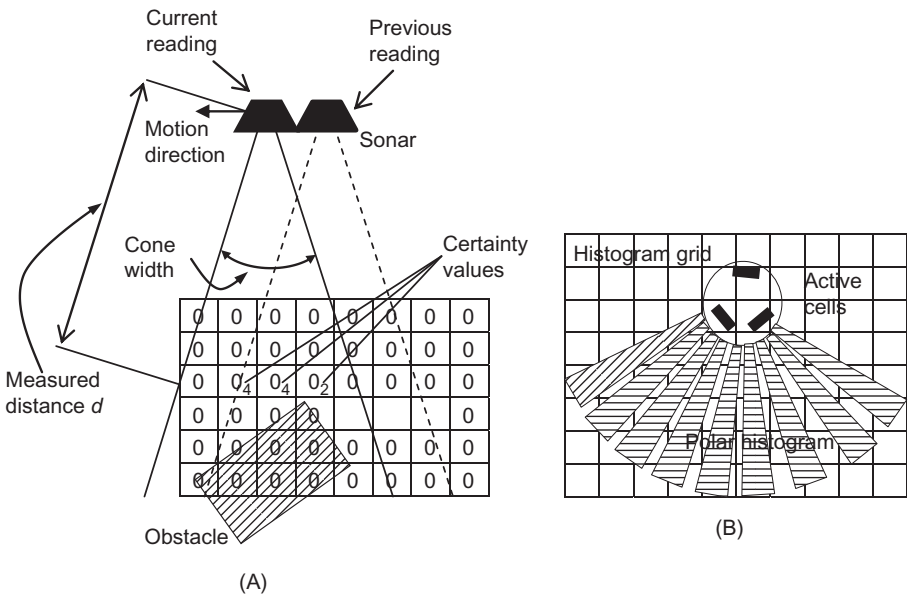


Figure 11.13 (A) Histogram grid. (B) The active cells are mapped onto the polar histogram.

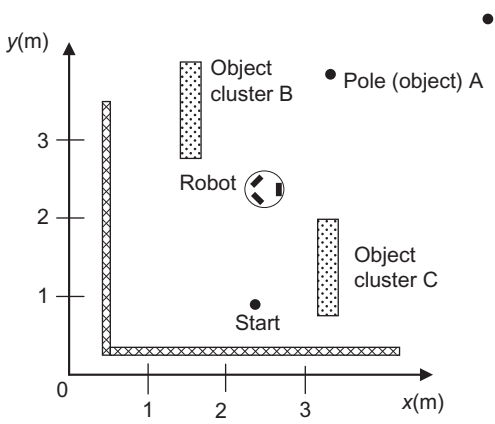


Figure 11.14 A mobile robot moving in a terrain with three obstacles (or clusters of obstacles) A, B, and C.

cells that lie on the moving window, each time, are called *active cells*. Actually, the cell that lies on the sonar axis and corresponds to the measured distance d found by each range reading is incremented and increases the *certainty value* (CV) of the cell (Figure 11.13A). The contents of each active cell in the histogram are mapped onto the corresponding polar histogram sector (say the kth sector), which gives a value H_k to this sector (Figure 11.13A). The value is higher if there are many cells with high CV in one sector. Obviously, this value can be regarded as the POD in the direction of sector k.

A typical terrain with three obstacles A, B, and C is shown Figure 11.14.

The polar histogram of the obstacle configuration of Figure 11.14 has the form of Figure 11.15B, and its counterclockwise pseudoprobability polar histogram (from A to C) on the H-k plane has the form of Figure 11.15A.

The peaks A, B, and C in Figure 11.15A result from the obstacle clusters A, B, and C in the histogram grid. To determine the safe directions of motion, a threshold T in POD is used. If POD $> T$, then we have unsafe (prohibited) direction of motion. If POD $< T$, we can select the most suitable direction in this sector. Figure 11.16A–C depicts three possible cases of robot motion alongside an obstacle. When the robot is too close to the obstacle, the steering angle ψ_{steer} points away from the obstacle. If the robot is away from the obstacle, ψ_{steer} points toward the obstacle. Finally, when the robot is at the proper distance from the obstacle, the robot moves alongside it.

11.4.3 Integration of Global and Local Path Planning

As discussed in Section 11.2, mobile robot path planning is distinguished in local and global path planning. In the first, which is performed while the robot is moving, the robot can find a new path according to the changes of the environment (obstacles, stairs, etc.). Global path planning can be performed only in a static environment which is known to the robot. In this case, the path planning algorithm produces a complete path from the start point to the goal before the robot starts its

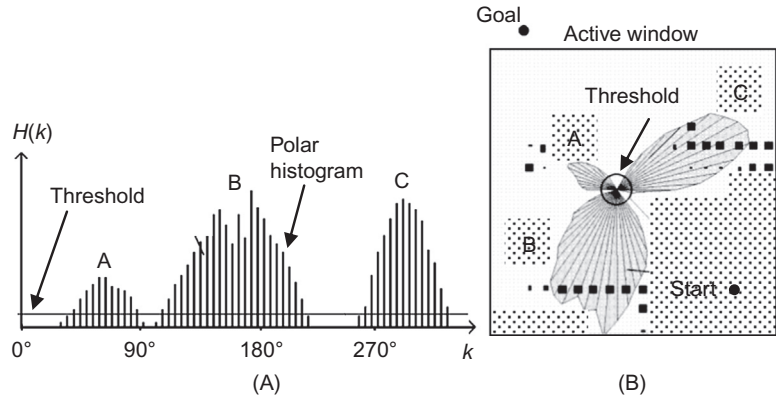


Figure 11.15 (A) The one-dimensional polar histogram (pseudoprobability distribution) corresponding to obstacles A, B, and C of Figure 11.14 in the counterclockwise direction from the *x*-axis. (B) The polar form of (A).
Source: Reprinted from Ref. [30], with permission from Institute of Electrical and Electronic Engineers.

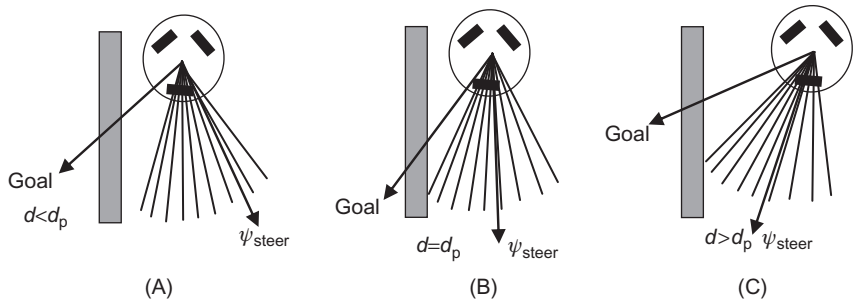


Figure 11.16 (A) Direction of motion when the robot obstacle distance d is smaller than the proper (desired) distance d_p. (B) Motion alongside the obstacle ($d = d_p$). (C) Direction of motion when $d > d_p$.

motion. A property that characterizes a path is its *smoothness* which is defined as the maximum curvature of the path measured over both its segments and the entire path. A motion planning method that takes into account the path curvature is presented in Section 11.5.3. The relative advantages and disadvantages of a typical global and local path planning can be seen in Figure 11.17.

The global path might be produced, for example, by the approximate cell decomposition method, but this could be done by including segments with high curvature values (e.g., the sharp turn shown in Figure 11.17A at the original motion steps of the navigation). This is a result of the nature of global path planning and is caused by the digitization of the space cells. But a global path method (e.g., cell decomposition) has an excellent capability in driving the robot out of an H-shaped obstacle which is known to be one of the most difficult cases (Figure 11.17A).

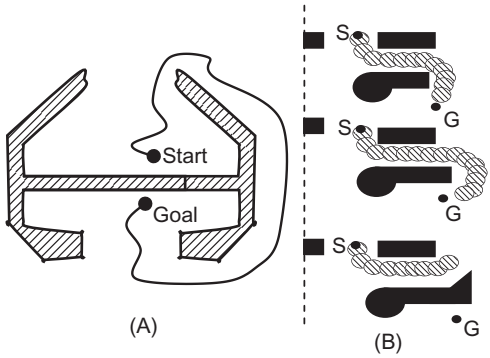


Figure 11.17 Typical cases of path planning: (A) global path planning and (B) local path planning.

As it is seen in Figure 11.17B, the local navigation strategies (e.g., active kinematic histograms [41]) present a series of simpler problems. The local planner manages to reach the goal in the first two cases, but in the last case, when the obstacle fully obstructs the way to the goal, the method fails to find a path although one exists, and the robot stays trapped in local minima in front of the obstacle. In this case, all the paths produced are sufficiently smooth, and the robot is traveling even adapting its speed according to obstacle configuration.

A global path planner is able to find a path from the start to the goal, if there exists such a path. But, in order to achieve smooth motion, the local planner gets the subgoals produced by the global planner and drives the robot toward them avoiding obstacles in the vicinity of the robot. Each time the local planner reaches a subgoal, the control returns back to the global path planner with the next subgoal. The above illustrates the necessity to properly integrate a global and a local path planner in order to warrant smooth and obstacle-free paths (without trapping at local minima). Actually, if a local minimum is detected at some point of the path, the global path planner is called to escape from the situation.

The above integration (collaboration) of local and global path planning is illustrated in pseudocode form as follows [42]:

Go to goal *(start position, final goal)*
While not at final goal

next subgoal = Global Path Planner (final goal)

Local path planner *(next subgoal)*
While not at *next subgoal* and *not blocked in local minima*

Drive the robot toward *next subgoal*

If blocked at local minima

Call recursively the go to goal algorithm with *start operant* the current position and *goal operant* the final goal.

A fuzzy algorithm which implements global and local navigation integration and uses potential fields is presented in Section 13.10. Another integrated (hybrid) path planner for indoor environments is presented in Ref. [32]. This hybrid planner

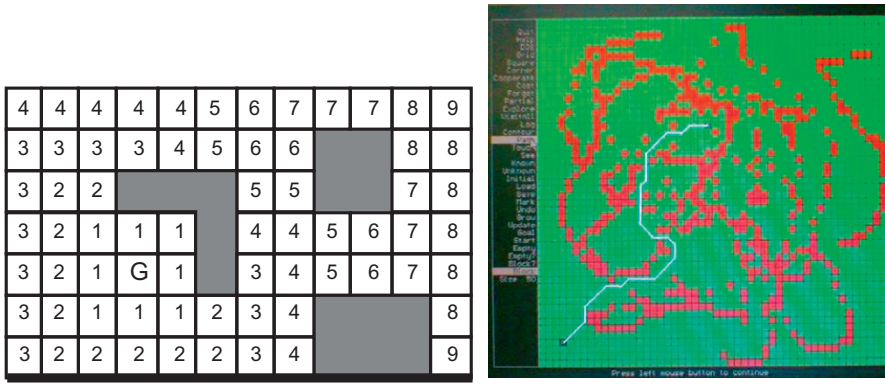


Figure 11.18 (A) The DT path planning scheme. (B) A typical minimal length path found by the steepest descend and DT.
Source: Reprinted from Ref. [43], with permission from International Journal of Computer Science and Applications.

combines the so-called *distance transform path planner* (DTPP) and the potential field planner. Unlike most planners, the distance transform (DT) planner treats the task of path planning by finding paths from the goal location back to the start location. This path planner propagates a distance wave front through all free space grid cells in the environment from the goal cell as shown in Figure 11.18A. Figure 11.18B shows a minimal length DT path [43].

Actually, the robot has to build the local map on the move, and at the same time constantly recompute the *distance map* and the path using locally available sensory data. DTPP employs an occupancy grid-based map of the workspace to compute its distance map. At the initialization stage, the goal cell and obstacle cells are assigned *values* that represent their distance from the goal. These distance values are propagated flowing around the obstacles. An algorithm similar to raster scanning iterates until the values of all cells are stabilized, in which case distance values are assigned to the rest of the cells. Obviously, the obstacle cells must be given very high values, and are passed over in the raster scan, and the raster scans are repeated until no further changes occur. The free configuration space cells should be treated in the same way. It was shown that the resulting DT is independent of any start point and represents a potential field without local minima [32]. Thus, a globally minimum distance path from any start point in free space can be determined via the standard gradient/steepest descend technique (see section 7.2.1). For a nonpoint robot, the obstacles are grown as usual by the *maximum effective radius* of the robot to convert the path planning to that of a point (dimensionless) robot.

Omitting the details, the steps of DTPP are as follows [32]:

Step 1: Using DT, create an optimal global path from the start point to the final goal.
Step 2: If not at the final goal, create a circle centered at the robot current position with reasonable radius depending on the configuration of the environment.

Step 3: Select the next subgoal as the intersection of the circumference of the circle and the planned global path.
Step 4: Use the potential field planner to reach the subgoal. Since the subgoals are always on the obstacle-free global path, the potential field planner cannot be trapped at local minima.

If there are two intersections along the planned global path, the one with lower distance value is selected such that the subgoal will always move toward the final goal. If no intersections are found, the radius of the circle is increased until an intersection with the planned global path is found. Implementation examples of DTPP are provided in Ref. [32]. A useful modification of DTPP method is provided in Ref. [44]. This modification extends the method to multiple robots by setting the initial exploration directions.

11.4.4 Complete Coverage Path Planning

Complete coverage path planning produces a path in which the robot sweeps all areas of free space in an environment. It is needed very often in practice (e.g., in autonomous room vacuum, security robots, lawn mowers). In the complete coverage DTPP, the robot moves away from the goal keeping track of the cell it has visited. The robot moves to a cell that has a smaller distance from the goal, only if it has visited all the neighboring cells that lie further away from the goal. The typical DTPP algorithm for complete coverage is as follows) [45,46]:

Set the start cell to current cell
Set all cells to not visited
Loop
Find unvisited neighboring cell with highest DT
If no neighbor cell is found then
Mark as visited and stop at Goal
If Neighbor cell DT <= Current cell DT then
Mark as visited and stop at Goal
Set current cell to neighboring cell
End Loop

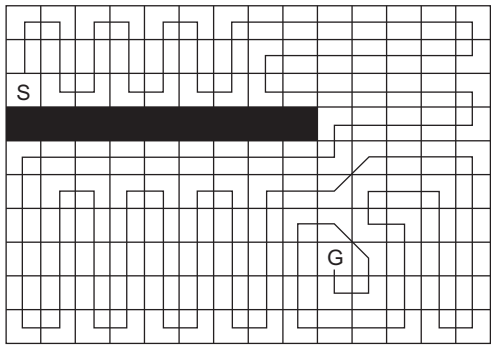
A one-obstacle environment with corresponding DT values is shown in Figure 11.19A, and a complete coverage path for this environment is depicted in Figure 11.19B.

A modified complete coverage DT path planning, called *path transform path planning* (PTPP) [45] propagates a weighted sum of the distance from the goal and a measure of the discomfort of moving too close to obstacles. Thus actually, a PTPP is a form of DTPP without the trapping to local minima possibility. The steps of the PTPP are the following:

Step 1: Invert the DT transform into an *obstacle transform* (OT), where the obstacle cells become the goals. This implies that for each free cell, the minimal distance from the center of the free space to the boundary of an obstacle cell is obtained.

13	12	11	10	9	8	7	7	7	7	7	7	7	7
13	12	11	10	9	8	7	6	6	6	6	6	6	6
S	12	11	10	9	8	7	6	5	5	5	5	5	5
									4	4	4	4	4
9	8	7	6	5	4	3	3	3	3	3	3	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	1	G	1	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4

(A)



(B)

Figure 11.19 (A) An environment with a single obstacle. (B) A complete coverage DT path from S to G. *Source*: Reprinted from Ref. [45], with permission from Japan Robot Association.

Step 2: Define a cost function transform, called the *path transform* (V_{PT}), of the form:

$$V_{\mathrm{PT}}(c) = \min_{p \in P} \left\{ L(p) + \sum_{c_i \in p} \lambda\, O(c_i) \right\}$$

where P is the set of all possible paths to the goal, c_i is the ith cell in P, and p is a single path in P. The function $L(p)$ is the length of the path p to the goal, and the function $O(c_i)$ is a cost function produced by using the values of OT. The constant $\lambda \geq 0$ is a weight that specifies how strongly the PT will avoid obstacles. The minimization of $L(p) + \sum \lambda\, O(c_i)$ is done by the standard steepest descent method, without the possibility to be locked at some minimum. This is because all costs of paths to the goal from each cell are computed.

Figure 11.20 depicts an environment with four obstacles and distances shown (A), a path found with the obstacle transform (B), and two paths found by path transform with weight constants $\lambda_1 < \lambda_2$ (C and D) [45,46].

Figure 11.21A shows the actual path generated by PTPP implemented on the AMROS simulator [47] in a 7 m × 6 m room with a 1.5 m × 1.0 m obstacle at the room's center.

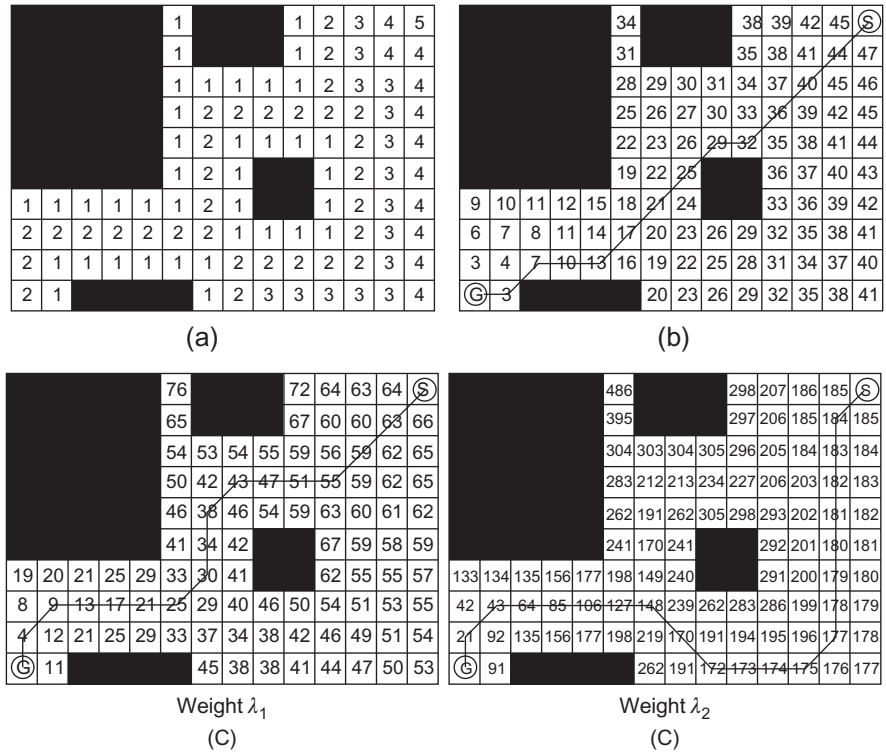


Figure 11.20 (A) DT for an environment with four obstacles. (B) Obstacle transform with the corresponding path. (C, D) Paths found with the PTPP for the same environment, where each path corresponds to the minimum propagated path cost to the goal. The path (C) corresponds to a lighter cost function than (D), that is, $\lambda_1 < \lambda_2$.
Source: Reprinted from Ref. [45], with permission from Japan Robot Association.

Another class of complete coverage path planning methods is based on the spiral algorithm as shown in Figure 11.21B [48].

If no obstacle on the right turn right
ELSE
If no obstacle in the front
Move forward
ELSE
If no obstacle on the left
Turn left
OTHERWISE
Terminate the algorithm

In this algorithm, previously covered cells and presently occupied cells are regarded as obstacles. The above basic spiral complete coverage path planning algorithm has been improved by many authors. For example, in Ref. [48], the complete coverage algorithm represents the environment as a union of robot-sized cells

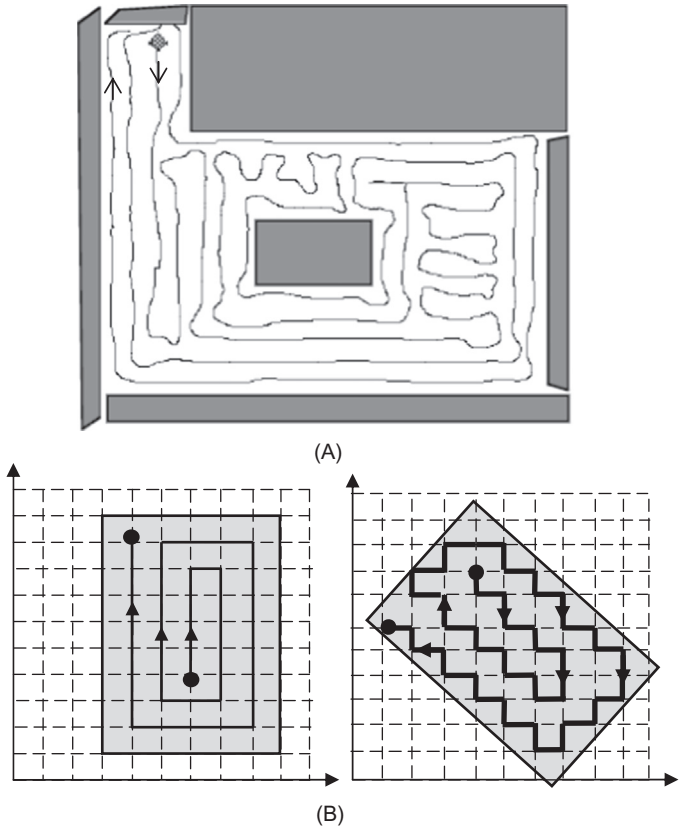


Figure 11.21 (A) A simulated complete coverage path for the Yamabico robot using PTPP with estimated position correction. (B) Spiral complete coverage path planning for initial robot orientation parallel (left) and nonparallel (right) to the wall.
Source: (A) Reprinted from Ref. [45], with permission from Japan Robot Association.

and then uses a spiral scheme. The overall path planner links the basic spiral paths using the constrained inverse DT.

11.5 Mobile Robot Motion Planning

11.5.1 General Online Method

The motion planning algorithms are distinguished into:

- Explicit
- Nonexplicit

according to the underlying path planning scheme.

Discrete explicit path planning schemes are the *road map* method and the *cell decomposition* method. The continuous explicit schemes are open-loop control algorithms. When a path compatible with the problem constraints is selected, this path and its derivatives constitute the feedforward component (reference trajectory) for the closed-loop system. Depending on the level at which the motion planning is made, we have *kinematic planning* and *dynamic planning*.

The *nonexplicit motion planning methods* (or as otherwise called, *online methods*) can be regarded as feedback control algorithms which are based on the displacement of the robot from the desired trajectory which is (possibly) generated by an explicit motion planning algorithm. The motion plan is an algorithm that determines how the robot should move, given its present state and present knowledge. A representative nonexplicit method uses an artificial potential field function in the configuration space which takes a minimum at the goal configuration.

Suppose that the configuration space is a subset of R^n. Let $\mathbf{q}$ be the n-dimensional vector of joint positions, and $\mathbf{q}_\mathrm{d}$ the vector of desired joint positions (that specify the goal configuration of the robot). Then, we define a potential function as:

$$V_0(\mathbf{q}) = (\mathbf{q}-\mathbf{q}_\mathrm{d})^\mathrm{T}\mathbf{K}(\mathbf{q}-\mathbf{q}_\mathrm{d}) \tag{11.1a}$$

where $\mathbf{K}$ is an $n \times n$ positive definite matrix ($\mathbf{K} > 0$). The simplest way to go to the goal is to use a velocity controller:

$$\dot{\mathbf{q}} = -\frac{\partial V_0(\mathbf{q})}{\partial \mathbf{q}} \tag{11.1b}$$

which leads the robot asymptotically to the goal configuration $\mathbf{q}_\mathrm{d}$. If there are obstacles, then the velocity $\dot{\mathbf{q}}$ should move the robot away from them. Let $d(\mathbf{q}, E_i)$ be a function that gives the distance of the robot's point that lies in a smaller distance from the obstacle E_i. We construct a new potential function $V_{E_i}(\mathbf{q})$ as:

$$V_{E_i}(\mathbf{q}) = -k_i/d(\mathbf{q}, E_i)^r \tag{11.2}$$

where r is a suitable integer and k_i a positive constant, and instead of Eq. (11.1a) we use a total potential function:

$$V(\mathbf{q}) = V_0(\mathbf{q}) + V_{E_i}(\mathbf{q})$$

Then, the controller:

$$\dot{\mathbf{q}} = -\frac{\partial V(\mathbf{q})}{\partial \mathbf{q}} = -\frac{\partial V_0(\mathbf{q})}{\partial \mathbf{q}} - \frac{\partial V_{E_i}(\mathbf{q})}{\partial \mathbf{q}} \tag{11.3}$$

leads the robot to the goal $\mathbf{q}_\mathrm{d}$ while simultaneously moves it away from the obstacle E_i. Clearly, for each obstacle $E_i (i = 1, 2, \ldots)$, we must add a new $V_{E_i}(\mathbf{q})$.

Example 11.2

The problem is to derive the local motion planning equations for a WMR using potential fields and polar representation.

Solution

For the purposes of local motion planning, we only need the local (relative) polar coordinates depicted in Figure 11.22 [26].

These coordinates are as follows:

- Distance d_{GR} and angle ϕ_{GR} of the goal relative to the robot
- Distance d_{OR} and angle ϕ_{OR} of the obstacle relative to the robot
- Distance d_{OG} and angle ϕ_{OG} of the obstacle relative to the goal
- Angle ϕ_γ between the robot axis direction and the direction in which the robot should go to the goal
- Angle ϕ_δ between the robot-goal and the obstacle-goal direction lines

From the geometry of Figure 11.22, we find the relations:

$$d_{OR} = (d_{GR}^2 + d_{OG}^2 - 2d_{GR}d_{OG}\cos\phi_\delta)^{1/2}$$
$$\phi_\delta = (\phi_{GR} - \phi_\gamma) - (\pi - \phi_{OG})$$
$$\phi_{OR} = \phi^* \operatorname{sgn}(\phi_\gamma) + \phi_{GR}$$

where:

$$\cos\phi^* = \left(\frac{d_{GR}^2 + d_{OR}^2 - d_{OG}^2}{2d_{GR}d_{OR}}\right)$$

The above relations give the distance d_{OR} and angle ϕ_{OR} of the obstacle relative to the robot in terms of d_{OG} and ϕ_{OG} which are independent of the robot position and direction of motion.

Now, applying the general potential-field-based motion planning method of Section 11.5.1, we define a potential function:

$$V(\mathbf{q}) = V_{att}(\mathbf{q}) + V_{rep}(\mathbf{q})$$

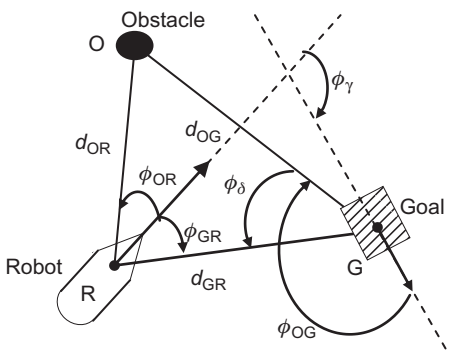


Figure 11.22 Geometry of local WMR motion planning in the presence of an obstacle.

which involves the goal-attraction term:

$$V_{\text{att}}(\mathbf{q}) = \begin{cases} \frac{1}{2}\lambda||\mathbf{q}-\mathbf{q}_{\text{goal}}||^2 = \frac{1}{2}\lambda d_{\text{GR}}^2 & \text{when} \quad d_{\text{GR}} < \varepsilon \\ \mu||\mathbf{q}-\mathbf{q}_{\text{goal}}|| = \mu d_{\text{GR}} & \text{when} \quad d_{\text{GR}} > \varepsilon \end{cases}$$

and an obstacle-repulsion term:

$$V_{\text{rep}}(\mathbf{q}) = 1/||\mathbf{q}-\mathbf{q}^*_{\text{obstacle}}|| = 1/d_{\text{OR}}$$

with λ, μ being scaling factors, $\varepsilon > 0$ being a distance from the goal, and $\mathbf{q}^*_{\text{obstacle}}$ the closest point to the obstacle. For each additional obstacle O_i, a corresponding repulsive potential field $V_{\text{rep},i}(\mathbf{q})(i = 1, 2\ldots, M)$ should be added, where M is the number of obstacles. In this case, the total repulsive potential is equal to:

$$V_{\text{rep}}(\mathbf{q}) = \sum_{i=1}^{M} V_{\text{rep},i}(\mathbf{q})$$

The attractive and repulsive forces applied to the robot are:

$$F_{\text{att}}(\mathbf{q}) = -\partial V_{\text{att}}(\mathbf{q})/\partial\mathbf{q}, \quad F_{\text{rep}}(\mathbf{q}) = -\partial V_{\text{rep}}(\mathbf{q})/\partial\mathbf{q}$$

And the total force is:

$$F(\mathbf{q}) = F_{\text{att}}(\mathbf{q}) + F_{\text{rep}}(\mathbf{q})$$

Clearly, since here we consider only local motion planning, we can assume that $d_{\text{GR}} < \varepsilon$ for some appropriate value of the distance ε that specifies the local area around the robot. Therefore:

$$\begin{aligned} V_{\text{att}}(d_{\text{GR}}) &= (1/2)\lambda d_{\text{GR}}^2 \\ V_{\text{rep}}(d_{\text{OR}}) &= (d_{\text{GR}}^2 + d_{\text{OG}}^2 - 2d_{\text{GR}}d_{\text{OG}}\cos\phi_\delta)^{-1/2} \\ \phi_\delta &= \phi_{\text{GR}} - (\phi_\gamma + \pi - \phi_{\text{OG}}) \end{aligned}$$

and

$$\begin{aligned} \partial V_{\text{att}}(d_{GR})/\partial d_{\text{GR}} &= \lambda d_{\text{GR}} \\ \partial V_{\text{rep}}/\partial d_{\text{GR}} &= -(V_{\text{rep}})^3(d_{\text{GR}} - d_{\text{OG}}\cos\phi_\delta) \\ \partial V_{\text{rep}}/\partial \phi_{\text{GR}} &= -(V_{\text{rep}})^3 d_{\text{GR}}d_{\text{OG}}\sin\phi_\delta \end{aligned}$$

since $\partial\phi_\delta/\partial\phi_{\text{GR}} = 1$.

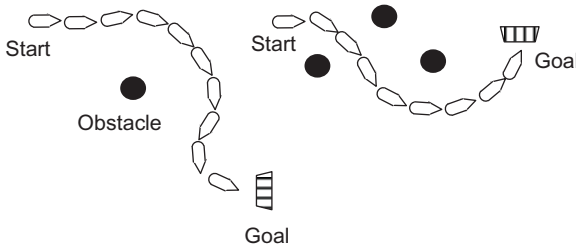


Figure 11.23 Robot paths with the above potential-based local motion planning.

These gradients are now used in the robot motion (velocity) planning controller (11.3). The result is:

$$v_{GR} = \dot{d}_{GR} = -\frac{\partial V_{att}}{\partial d_{GR}} - \frac{\partial V_{rep}}{\partial d_{GR}}$$

$$= -\lambda d_{GR} + (V_{rep})^3 (d_{GR} - d_{OG} \cos \phi_\delta)$$

$$\omega_{GR} = \dot{\phi}_{GR} = (V_{rep})^3 d_{GR} d_{OG} \sin \phi_\delta$$

Two examples of the potential field path followed by a robot in a terrain with one or three obstacles are shown in Figure 11.23.

The potential field motion planning method enhanced with neural network learning was studied in Ref. [26].

11.5.2 Motion Planning Using Vector Fields

This method is particularly useful for the motion planning of nonholonomic WMRs. Consider the differential drive WMR of Figure 2.7 which is described by the affine kinematic model (2.37a):

$$\begin{bmatrix} \dot{x}_Q \\ \dot{y}_Q \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} (r/2)\cos \phi \\ (r/2)\sin \phi \\ r/2a \end{bmatrix} u_1 + \begin{bmatrix} (r/2)\cos \phi \\ (r/2)\sin \phi \\ -r/2a \end{bmatrix} u_2 \tag{11.4}$$

where $u_1 = \dot{\theta}_r$ and $u_2 = \dot{\theta}_l$. As we already know, although using the two inputs u_1 and u_2 in Eq. (11.4), the robot can go to any desired point of the terrain, and due to the nonholonomicity, the vehicle cannot follow all the possible trajectories on the plane.

Any motion plan (program) should include the constraint Eq. (11.4). A solution to this is as follows [23].

Let a robot be described by:

$$\dot{\mathbf{x}} = \mathbf{f}_1(\mathbf{x})u_1 + \mathbf{f}_2(\mathbf{x})u_2 + \cdots + \mathbf{f}_m(\mathbf{x})u_m \tag{11.5}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$ is the n-dimensional state vector and $\mathbf{u} = [u_1, u_2, \ldots, u_m]^{\mathrm{T}}$ the m-dimensional input vector. The initial and final configurations are, respectively:

$$\mathbf{x}(t_0) = \mathbf{x}_0, \quad \mathbf{x}(t_{\mathrm{f}}) = \mathbf{x}_{\mathrm{f}} \tag{11.6}$$

where t_0 is the initial time and t_{f} is the final time.

The *motion planning* problem is to find a piecewise continuous and bounded input vector $\mathbf{u}(t)$ such that to satisfy the second relation in Eq. (11.6). This is actually an *open-loop final state (pose) control problem*. Here, we will examine it using the vector-field and Lie bracket theory. The Lie bracket of the vector fields $\mathbf{f}_i$ and $\mathbf{f}_j (i, j = 1, 2, \ldots, m)$ in the configuration space CS is defined as in Eq. (6.12), that is:

$$\left[\mathbf{f}_i, \mathbf{f}_j\right](\mathbf{x}) = \frac{\partial \mathbf{f}_j}{\partial \mathbf{x}} \mathbf{f}_i - \frac{\partial \mathbf{f}_i}{\partial \mathbf{x}} \mathbf{f}_j \tag{11.7}$$

It is an antisymmetric operator that returns a vector field and provides a measure of how the flows $\partial \mathbf{f} / \partial \mathbf{x}$ that correspond to the vector fields $\mathbf{f}_i$ and $\mathbf{f}_j$ are interchanged. Actually, the Lie bracket represents the infinitesimal motions that result when we have a forward flow by $\mathbf{f}_i$ and then by $\mathbf{f}_j$ and after that a backward flow by $-\mathbf{f}_i$ and then by $-\mathbf{f}_j$. Clearly, if this input sequence is applied to a linear system, it will lead to a zero net motion. But in nonlinear systems, we can generate new motion directions by simply transposing vector fields through the Lie bracket (see Figure 6.3).

This can be done by applying the Lie bracket (operator) to each of the new motion directions and each of the directions that has been used in their generation.

Let $\boldsymbol{\Delta}_0$ be the *span* distribution of the initial input vector fields $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_m$. We denote the sequential application of the Lie bracket to the input vector fields as:

$$\boldsymbol{\Delta}_i = \boldsymbol{\Delta}_{i-1} + \mathrm{span}\{[\boldsymbol{\alpha}, \boldsymbol{\beta}],\ \boldsymbol{\alpha} \in \boldsymbol{\Delta}_0,\ \boldsymbol{\beta} \in \boldsymbol{\Delta}_{i-1}\}$$

The criterion for checking controllability is the following (see also Theorem 6.9):

"The system is controllable if and only if we have $\boldsymbol{\Delta}_k = R^n$ for some k." Applying this criterion to our differential drive WMR Eq. (11.4), we have:

$$\mathbf{f}_1 = \begin{bmatrix} (r/2)\cos\phi \\ (r/2)\sin\phi \\ r/2a \end{bmatrix}, \quad \mathbf{f}_2 = \begin{bmatrix} (r/2)\cos\phi \\ (r/2)\sin\phi \\ -r/2a \end{bmatrix} \tag{11.8}$$

The Lie bracket of $\mathbf{f}_1$ and $\mathbf{f}_2$ is:

$$\mathbf{f}_3 = [\mathbf{f}_1, \mathbf{f}_2] = \begin{bmatrix} -(r^2/2a)\sin\phi \\ (r^2/2a)\cos\phi \\ 0 \end{bmatrix} \tag{11.9}$$

Because these vector fields cover all the allowable motion directions on the plane, the system is *controllable*. This criterion can be used for the generation of desired motions (i.e., of successfully controlling robots with nonholonomic constraints along a motion plan). Indeed, suppose that we want to find a motion for a Lie bracket (system) $[\mathbf{f}_i, \mathbf{f}_j]$ of first order (Eq. (11.9)). This can be done by using high-frequency sinusoid inputs of the form [23,49].

$$u_i = \xi_i(t) + \sqrt{\omega}\xi_{ij}(t)\sin \omega t \tag{11.10a}$$

$$u_j = \xi_j(t) + \sqrt{\omega}\xi_{ji}(t)\cos \omega t \tag{11.10b}$$

In the limit $\omega \to \infty$, we get the motion:

$$\dot{\mathbf{x}} = \mathbf{f}_i(\mathbf{x})\xi_i(t) + \mathbf{f}_j(\mathbf{x})\xi_j(t) + \frac{1}{2}\xi_{ij}(t)\xi_{ji}(t)\left[\mathbf{f}_i, \mathbf{f}_j\right](\mathbf{x}) \tag{11.11}$$

This means that using the above u_i and u_j allows the robot to follow the direction of the Lie bracket $[\mathbf{f}_i, \mathbf{f}_j]$ as if it was one of the initial controlled directions.

11.5.3 Analytic Motion Planning

The analytic obstacle avoidance motion planning approach consists in describing the WMR trajectory parametrically and determining the optimal values of the parameters used [13]. Here, we will use sinusoid parameterization. Let $\mathbf{x}_0$ be an initial configuration of the mobile robot M that moves on the configuration space $CS = R^2$, which involves a number of stationary obstacles $B_i (i = 1, 2, \ldots, n)$. The problem is to find a trajectory that determines a sequence of configurations of M from $\mathbf{x}_0$ to a desired configuration (pose) $\mathbf{x}_d$ such that the robot avoids collisions with the obstacles B_i.

To solve this problem, we express the plane trajectory by the Cartesian parametric equations $x = x(t)$ and $y = y(t)$ which represent the position of the robot M as a function of the parameters from $t = 0$ to $t = t_f = 1$ at the end of the trajectory. Such a curve must be sufficiently smooth to assure the continuity of the curvature $\kappa(t)$, which is defined as:

$$\kappa(t) = \frac{\mathrm{d}\phi}{\mathrm{d}s} = \frac{\mathrm{d}\phi/dt}{\mathrm{d}s/dt} = \frac{\dot{\phi}(t)}{\sqrt{\dot{x}^2 + \dot{y}^2}} \tag{11.12a}$$

where ϕ is the tangential angle and s is the curve length. The derivative $\dot{\phi}(t)$ can be found using the relation:

$$tg\ \phi(t) = \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y/dt}{\mathrm{d}x/dt} = \frac{\dot{y}(t)}{\dot{x}(t)} \tag{11.12b}$$

From Eq. (11.12b), we find:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t} tg\ \phi(t) &= \frac{1}{\cos^2\phi(t)}\dot{\phi}(t)\\ &= (1 + tg^2\ \phi(t))\dot{\phi}(t)\\ &= (1 + \dot{y}^2/\dot{x}^2)\dot{\phi}(t)\end{aligned}$$

that is:

$$\begin{aligned}\dot{\phi}(t) &= \left(\frac{\dot{x}^2}{\dot{x}^2 + \dot{y}^2}\right)\frac{\mathrm{d}}{\mathrm{d}t} tg\ \phi(t)\\ &= \left(\frac{\dot{x}^2}{\dot{x}^2 + \dot{y}^2}\right)\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\dot{y}(t)}{\dot{x}(t)}\right)\\ &= \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2}\end{aligned} \tag{11.13}$$

Therefore, Eq. (11.12a) gives:

$$\kappa(t) = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{[\dot{x}^2(t) + \dot{y}^2(t)]^{3/2}} \tag{11.14}$$

It follows that for $\kappa(t)$ to be continuous, the functions $x(t)$ and $y(t)$ must be differentiable at least up to order 3.

Some typical forms of parametric curves $\{x(t), y(t)\}$ used for path/motion planning are as follows:

Cartesian polynomials

$$x(t) = \sum_{i=0}^{n} a_i t^i, \quad y(t) = \sum_{i=0}^{n} b_i t^i \quad (a_i, b_i \in R)$$

Cubic spirals

$$\dot{\phi}(t) = \frac{1}{2}At^2 + Bt + C = \kappa(t)$$

$$\phi(t) = \frac{1}{6}At^3 + \frac{1}{2}Bt^2 + Ct + D$$

This class minimizes the integral of the square of the curvature's derivative.

Finite sums of sinusoids

$$x(t) = \sum_{i=1}^{n} [A_i \cos(i\omega t) + B_i \sin(i\omega t)] \tag{11.15a}$$

$$y(t) = \sum_{i=1}^{n} [C_i \cos(i\omega t) + D_i \sin(i\omega t)] \tag{11.15b}$$

For $n = \infty$, these functions can approximate any continuous function.

The optimization criterion is typically the length of the curve $c(t) = c(x(t), y(t))$ which is given by:

$$J = \int_0^1 [\dot{x}^2(t) + \dot{y}^2(t)]dt \tag{11.16}$$

The *optimal trajectory* minimizes J. Any trajectory that satisfies the inequality $J < J_{\max}$, where $J_{\max}$ is a predetermined (allowed) upper bound, is defined as *sub-optimal trajectory*.

In the following, we will assume a given number n of sinusoids and a given frequency ω. Thus, the parameter vector $\boldsymbol{\theta}$ to be selected has dimensionality $4n$:

$$\boldsymbol{\theta} = [A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2, A_3, B_3, C_3, D_3, \ldots, A_n, B_n, C_n, D_n]^{\mathrm{T}} \tag{11.17}$$

Differentiating Eqs. (11.15a) and (11.15b), we get:

$$\dot{x}(t) = \sum_{i=1}^{n} [-(i\omega)A_i \sin(i\omega t) + (i\omega) B_i \cos(i\omega t)] \tag{11.18a}$$

$$\dot{y}(t) = \sum_{i=1}^{n} [-(i\omega)C_i \sin(i\omega t) + (i\omega)D_i \cos(i\omega t)] \tag{11.18b}$$

Therefore, the above quadratic function J is a square function of the components of $\boldsymbol{\theta}$, independent of the parameter t.

Now, suppose that the robot is to go from the initial position $x(0) = 0$, $y(0) = 0$ with orientation ϕ_0, to the goal (final) position $x(1) = x_{\mathrm{f}}$, $y(1) = y_{\mathrm{f}}$ with orientation ϕ_{f}. The corresponding conditions for A_i, B_i, C_i, and D_i are found by replacing the above initial and final conditions in Eqs. (11.15a), (11.15b), (11.18a), and (11.18b), namely:

$$x(0) = 0 \text{ gives } \sum_{i=1}^{n} A_i = 0$$

$$y(0) = 0 \text{ gives } \sum_{i=1}^{n} C_i = 0$$

$$x(1) = x_f \text{ gives } \sum_{i=1}^{n} [A_i \cos(i\omega) + B_i \sin(i\omega)] = x_f$$

$$y(1) = y_f \text{ gives } \sum_{i=1}^{n} [C_i \cos(i\omega) + D_i \sin(i\omega)] = y_f$$

$$\frac{\dot{y}(0)}{\dot{x}(0)} = tg\ \phi_0 \text{ gives } \frac{\sum_{i=1}^{n} [i\omega D_i]}{\sum_{i=1}^{n} [i\omega B_i]} = tg\ \phi_0,$$

$$\frac{\dot{y}(1)}{\dot{x}(1)} = tg\ \phi_f \text{ gives } \frac{\sum_{i=1}^{n} [-i\omega C_i \sin(i\omega) + i\omega D_i \cos(i\omega)]}{\sum_{i=1}^{n} [-i\omega A_i \sin(i\omega) + i\omega B_i \cos(i\omega)]} = tg\ \phi_f$$

The obstacles that exist in the task space can also be represented by parametric curves. For example, an object with the shape of an ellipse is described by:

$$(x(t) - x_*)^2/a + (y(t) - y_*)^2/b = 1 \tag{11.19}$$

An edge of a polygonic object is described by a straight line segment:

$$y(t) = ax(t) + b$$

Therefore, one way to guarantee obstacle avoidance is to assure that the path and objects parametric curves do not intersect. This can be done if we equate the expression of $c(t)$ with any boundary of the obstacle (which is another parametric function with parameter s) and solve the resulting algebraic system with respect to t and s. The two curves intersect if:

$$0 \le t \le 1 \quad \text{and} \quad 0 \le s \le 1$$

Another method is to consider the constrained minimization of the function J with the constraint:

$$g(\boldsymbol{\theta}) \le 0 \tag{11.20}$$

that expresses the obstacle-free task space. For example, for the elliptic obstacle, this obstacle avoidance constraint is:

$$[x(t) - x_*]^2/a + [y(t) - y_*]^2/b - 1 \le 0 \tag{11.21}$$

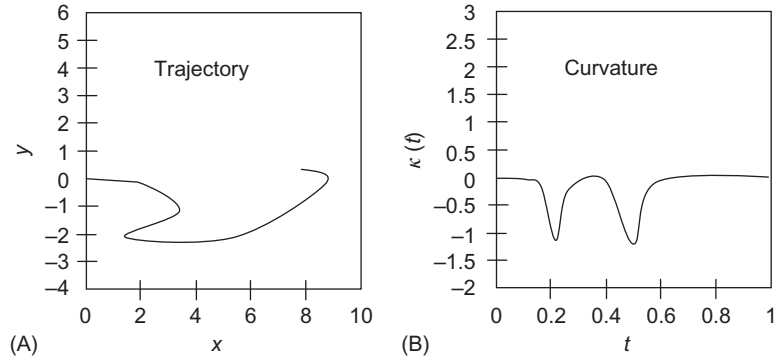


Figure 11.24 Obstacle-free case ($n = 2,\ \omega = \pi$). (A) Trajectory and (B) curvature.

The difficulty of this technique is that the constraints (11.20) and (11.21) depend on the parameter t, whereas the optimization problem requires their expression in terms of $\boldsymbol{\theta}$. One way to face this difficulty is to apply Eq. (11.21) for a certain number of values (e.g., equidistant values) of t. This is done in the simulation results that follow, where the optimization was performed with the aid of Matlab's optimization toolbox (function "constr"). The input data used are:

$$(x_0, y_0) = (0, 0), \quad \phi_0 = 0, \quad (x_f, y_f), \quad \phi_f, \quad \omega = \pi$$

Case 1 *Obstacle-free space with* $n = 2,\ \omega = \pi$

Initial parameter value $\boldsymbol{\theta}_0 = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$

Final position/orientation $x_f = 9,\ y_f = 1,\ \phi_f = 3\pi/4$

The optimal value of J obtained is $J^0 = 14.28$. The resulting trajectory $\{x(t), y(t)\}$ and curvature $\kappa(t)$ are shown in Figure 11.24 [13].

We see that the curvature does not take large values. Thus, this trajectory may be appropriate for a WMR with nonholonomic or hard constraints (e.g., constraints on the steering angle).

Case 2 *Obstacle nonfree space with* $n = 2,\ \omega = \pi$

Here:

$$(x_0, y_0) = (0, 0), \quad (x_f, y_f) = (8, 4), \quad \phi_0 = \phi_f = 0$$

The initial parameter value is again $\boldsymbol{\theta}_0 = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$.

The resulting trajectories for an elliptic and hexagonal obstacle are shown in Figure 11.25.

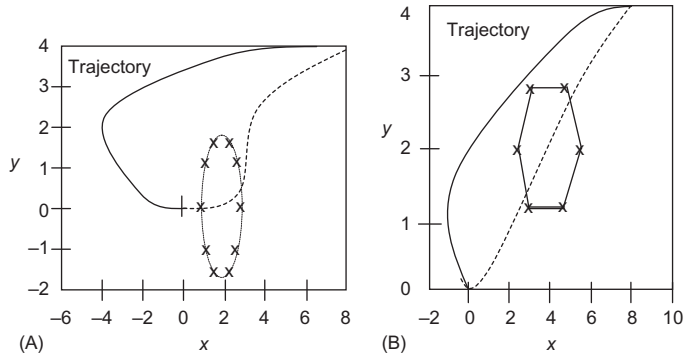


Figure 11.25 Path generation for obstacle avoidance. (A) Elliptic obstacle and (B) hexagonal obstacle.

The trajectory length for the elliptic obstacle is $L = 10.37$ and for the hexagonal $L = 8.75$. In both cases, the obstacle-avoiding path is shown by the continuous line, while the obstacle-free path is shown by the dotted line.

Example 11.3

Examine the motion planning problem in the joint space for a single joint of a fixed or mobile robot to go from an initial angular position θ_0 at time $t = 0$, to a final position θ_f at $t = t_f$. No obstacle is present in the robot's configuration space.

Solution

Each joint of the robot is driven (actuated) by a motor (see Example 5.1). The simplest way to plan the motion is to consider that the acceleration and deceleration periods of the motor are equal. We assume that at $t = 0$, the motor is stationing (zero velocity and zero acceleration). At time $t = 0^+$, we apply a constant acceleration $+a_0$, and we assume that the maximum velocity allowed is v_{max}. When reaching the maximum velocity at time t_1, the motor is moving with constant speed up to time $t_f - t_1$, and then decelerates with deceleration $-a_0$. During the acceleration period $[0, t_1 = 2T]$, we have:

$$\theta(t) = \frac{1}{2}a_0 t^2, \quad \theta_1 = \theta(t_1) = \frac{1}{2}a_0 t_1^2, \quad v_{max} = at_1$$

During the constant velocity period $[t_1, t_2 = t_f - t_1]$, we have:

$$\theta(t) = \theta_1 + v_{max}(t - t_1), \quad \theta_2 = \theta(t_2) = \theta_1 + v_{max}(t_2 - t_1)$$

Finally, during the deceleration period $[t_2, t_f]$ we have:

$$\theta(t) = \theta_2 + v_{max}(t - t_2) - \frac{1}{2}a_0(t - t_2)^2$$

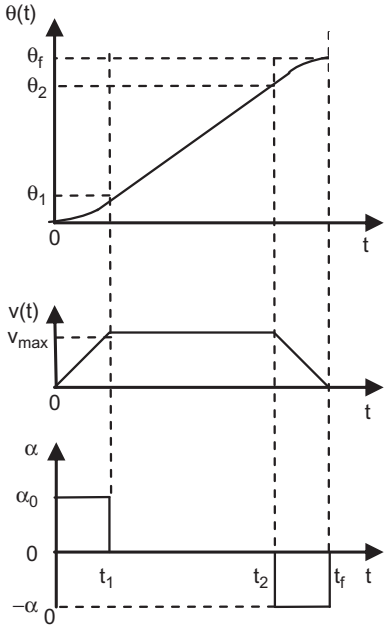


Figure 11.26 Typical plots for the point-to-point motion planning case.

The final position $\theta_f = \theta(t_f)$ is equal to:

$$\theta_f = \theta_2 + v_{max}(t_f - t_2) - \frac{1}{2}a_0(t_f - t_2)^2$$

$$= \theta_2 + v_{max}t_1 - \frac{1}{2}a_0 t_1^2 = \theta_2 - \theta_1 + v_{max}t_1$$

$$= v_{max}(t_2 - t_1) + v_{max}t_1 = v_{max}t_2$$

The time t_1 (switching from constant acceleration $+a_0$ to zero acceleration) and the time t_2 (switching from zero acceleration to deceleration $-a_0$) are given by:

$$t_1 = v_{max}/a_0 \quad \text{and} \quad t_2 = \theta_f/v_{max} \tag{11.22}$$

The total time of motion is equal to:

$$t_f = t_1 + t_2 = \frac{v_{max}}{a_0} + \frac{\theta_f}{v_{max}} \tag{11.23}$$

Given a_0, v_{max}, and θ_f, the switching times t_1 and t_2 can be automatically computed using the above formulas (11.22). Figure 11.26 shows the acceleration, velocity, and position plots for the constant acceleration/deceleration case.

11.6 Mobile Robot Task Planning

11.6.1 General Issues

Robot task planning belongs to the general planning problem of artificial intelligence, called *AI planning* [6−12]. AI planning is a process of generating possible multistage solutions to a specific problem. It is based on partitioning a problem in smaller, simpler, quasi-independent steps starting with an initial situation (state) and reaching a specified goal state by performing only the allowed movement steps along the solution path. This is equivalent to the movement through the solution space using the permitted state transformation operators. Thus, we can say that AI planning is a specific search process in the solution space. Basic concepts of AI planning are the following:

- *State space*: A space that involves all possible situations (states) that may occur. For example, a state could represent the position and orientation of a robot, the position and velocity of a car, and so on. The state space may be discrete (finite or countably infinite) or continuous (uncountably infinite).
- *Actions or operators*: A plan generates actions (operators) that manipulate the state. The terms actions and operators are used indistinguishably in AI, control, and robotics. The planning formulation must include a specification of how the state changes when actions (or controls) are applied. In discrete time, this may be defined as a state-valued function, and in continuous time as an ordinary differential equation. In some cases, actions could be selected by nature and are not under the control of the decision maker.
- *Initial and goal states*: A planning problem must involve some initial state, and specify the goal state in a set of goal states. The actions must be selected so as to drive the system from the initial state to the desired goal state.
- *Criterion*: This encodes the desired outcome of a plan in terms of the state and actions that are performed. The two basic criteria used in planning are *feasibility* (i.e., find a plan that causes the arrival at a goal state, regardless of its efficiency) and *optimality* (i.e., find a feasible plan that optimizes the system performance in some desired way, in addition to arriving at a goal state). Achieving optimality is much more challenging than simply assuring feasibility.
- *Plan*: A plan imposes a specific strategy or behavior on a decision maker. It may simply specify a sequence of actions to be taken, independently of how complicated this sequence is.
- *Algorithm*: This is difficult to be precisely and uniquely defined. In theoretical computer science, algorithm is a *Turing machine* (i.e., a finite-state machine with a special head that can read and write along an infinite piece of tape). Using the Turing machine as a model for algorithms implies that the physical world must be first properly modeled and written on tape before the algorithm can make decisions. But if changes take place in the world during the execution of the algorithm, it is not clear what may happen (e.g., in case a mobile robot is moving in a cluttered environment where humans are walking around). Thus, an online algorithm model is more appropriate in these cases (which relies on special sensory information).
- *Planner*: A planner simply produces a plan and may be a machine or a human. If the planner is a machine, it is generally considered to be a planning algorithm (a Turing machine or an online program). In many cases, the planners are humans who develop

plan(s) that can work in all situations. The three ways of employing a generated plan are: *execution* by a machine (e.g., a robot) or a simulator, (ii) *refinement* (i.e., improvement to a better plan), and (iii) *hierarchical inclusion* (i.e., packaging it as an action in a higher level plan).

Representative examples of robot task planning problems are the following:

- Automotive and other assembly planning
- Sealing cracks in automotive assembly
- Parking cars and trailers
- Mobile robot navigation
- Mobile manipulator service planning

Questions that have to be addressed in task planning are: What is a plan? How is a plan represented? How is it computed? What is expected to achieve? How is its quality evaluated? and Who or what is going to use it? Answers to these questions can be found in AI textbooks.

11.6.2 Plan Representation and Generation

11.6.2.1 Plan Representation

Plans can be represented and generated as:

- State-space problems
- Action-ordering problems

State-Space Representation

It is very familiar in systems theory and automata theory. When applied to *plan representation,* the state vectors represent the states of the application domain of the plan and the actions or operators. The state-space representation is the best way to recognize a solution of the problem at hand. In addition, state-space representations can be easily converted to equivalent computer programs, and are directly usable when searching algorithms are applied to problem solving. In this case, the solution to a problem in state space provides a path that connects the initial and goal state in a feasible way.

Clearly, the planning process in the state space is actually a search process, driven by the requirement of achieving a goal (i.e., it is essentially a goal-driven search process). Nodes of search trees can be viewed as possible states of the world presented as some possible partial plans, and the goal pursuing process, as a partial-plan search process.

Action-Ordering Representation

This representation involves a number of actions in the order in which they should be executed, along with the associated restrictions to be compiled. For example, it might be a simple list of actions under execution in the order they appear on the list. The action-ordering representation focuses on actions to be performed and not on conditions that could be affected by the individual actions. It is therefore a plan

representation very different than the state-space representation. Actually, no states outside the list are used for problem representation. Essentially, action-ordering representation is a behavioral representation, because it contains no explicit notion of state, although one can associate with each action-ordering plan a certain number of possible states to be generated by the actions. Two major advantages of action-ordering plan representation are as follows:

1. Capability to represent parallel activities, which is very important when the action-ordering list is only partially ordered leaving room for reordering actions.
2. Easy implementation in a computer (e.g., as a directed graph structure), where the nodes represent the actions and the arcs the corresponding ordering relations.

Partially ordered plans allow parallel actions, and are called *parallel plans*. Totally, ordered plans require sequential actions, and are called *serial plans*. Serial plans are equivalent to state-space transition diagrams with single-operator labeled arcs. Alternative methods for formulating robotic planning problems are *predicate calculus*, *tick lists*, and *triangle tables*.

11.6.2.2 Plan Generation

As we saw before, planning is a problem of performing a search through a state space. At each planning stage, several computational steps are required before its execution. The computations provide a selection guide for the next move in the search space, that is, for the path construction that leads from the initial state to the goal state. Each plan node in the state space involves partial-plan information, and so the full set of nodes between the initial and the goal state represents the total plan. By searching through possible plan states, the planner *generates the plan*. A basic issue in planning is to select a suitable modeling method or knowledge representation describing the task domain. Some methods commonly used are as follows:

- Production rules (forward/backward)
- Predicate logic
- Procedural nets
- Object or schema-based methods
- AND/OR graphs
- Petri nets

Usually, if the problem is too complicated and the state space to be searched is very large, the total planning problem is decomposed into a number of simple subprocesses that can be performed separately. When solving a planning problem, the planning system must take into account the existing constraints which must be properly formulated, propagated, and satisfied.

Constraint formulation helps in specifying the interactions among individual solutions to different goals. *Constraint propagation* creates new constraints out of old ones and helps to refine the solution path in screening its nonpromising links. Finally, *constraint satisfaction* must be assured by the generated plan.

The least-commitment method helps to refine only those parts of the plan that will not be abandoned later.

The plan generation methods so far discussed produce plans that are linear sequences of complete subplans and are known as *linear planning*. On the contrary, *nonlinear planners* can generate plans whose individual subplans, needed to achieve the given conjunctive goals, are generated in parallel.

11.6.3 World Modeling, Task Specification, and Robot Program Synthesis

We now discuss a little more the three phases of robot task planning, namely, *world modeling*, *task specification*, and *program synthesis*.

11.6.3.1 World Modeling

World modeling is concerned with the geometric and physical description of the objects in the workspace (including the robot itself) and the representation of the assembly state of the objects. A geometric model provides the spatial information (dimensions, volume, shape, etc.) of the object in the workspace. The typical method used for modeling 3D objects is the *constructive solid geometry* (CSG) where the objects are defined as combinations or constructions via regularized set operations (union, intersection, etc.) of primitive objects (cube, parallelpiped, cylinder). The primitives can be represented in many ways, such as:

- A set of points and edges
- A set of surfaces
- Generalized cylinders
- Cell decomposition
- A procedure name that calls to other procedures representing other objects.

In the *AUTOPASS* assembly planner, the world state is represented by a graph, the nodes of which represent objects and edges represent relationships, such as [6]:

- *Attachment* (rigid, nonrigid, or conditional attachment of objects)
- *Constraints* (relationships that represent physical constraints, translational or rotational, between objects)
- *Assembly component* (which indicates that the subgraph linked by this edge is an assembly part that can be referenced as an object).

11.6.3.2 Task Specification

This can be done using a high-level specification language. For example, an assembly task can be described as a sequence of states of the world model. The states must be provided by the configurations of all the objects in the workspace (as it will be done later in Example 11.4). Another way of task description is to use a sequence of symbolic operations on the objects, including proper spatial constraints to eliminate possible ambiguities. Most robot-oriented languages use this type of

task specification. AUTOPASS uses this kind of specification, but it uses a more detailed syntax, which divides its assembly related statements into three groups:

1. *State change statement* (i.e., an assembly operation description)
2. *Tools statement* (i.e., description of the kind of tools that must be used)
3. *Fastener statement* (i.e., description of fastening operation)

11.6.3.3 Robot Program Synthesis

This is the most difficult and important phase of task planning. The major steps in this phase are as follows:

- Grasp planning
- Motion planning
- Plan checking

Grasp planning specifies how the way the object is grasped affects all subsequent operations. The way the robot can grasp an object depends on the geometry of the object being grasped and the other objects present in the workspace. To be useful, a grasping configuration must be feasible and stable. The robot must be able to reach the object without any collision with other objects in the workspace. Also, the object once grasped must be stable during subsequent operations of the robot.

Motion planning specifies how the robot must move the object to its destination and complete the operation. The steps for successful motion planning are as follows:

- *Guarded departure* from the current configuration
- *Collision-free motion* to the desired configuration

Plan checking is made to ensure that the plan accomplishes the desired sequence of tasks, and if not to attempt an alternative plan.

Example 11.4 ***(Blocks world problem)***

We are given three blocks A, B, and C in the initial configuration shown in Figure 11.27A. The problem is for the robotic manipulator to move the blocks to the final configuration shown in Figure 11.27B through the application of the following forward production rules (F-rules):

1. pick up (X)
2. put down (X)

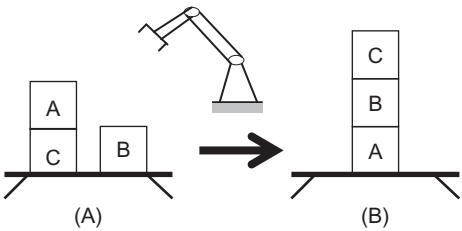


Figure 11.27 A "blocks world" problem. (A) Initial state and (B) final state.

3. stack (X,Y)
4. unstack (X,Y)

that represent possible robot actions.

Solution
The initial block configuration (arrangement) can be represented by the conjunction of the following statements:

clear (B)	Block B has a clear top
clear (A)	Block A has a clear top
on (A,C)	Block A is on block C
on table (B)	Block B is on the table
on table (C)	Block C is on the table
handempty	The robot hand is empty

Robot actions change one state, or arrangement, of the world to another. Given that the final state is to be achieved using the above four F-rules, the triangle table representation of the planning process is as shown in Figure 11.28.

Each column of the table describes the new situation after an operator (action) (e.g., *unstuck, put down*) has been carried out, while *handempty* and *handholding* mean that the

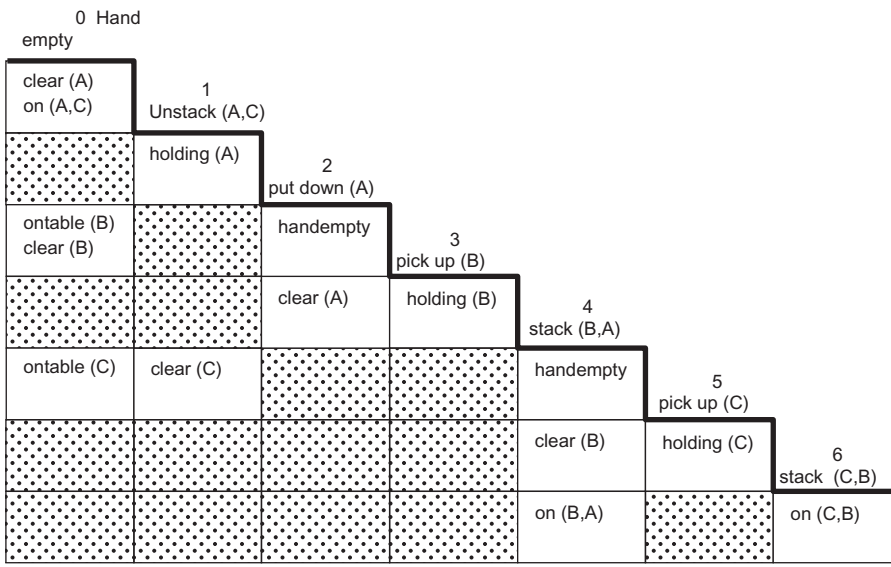


Figure 11.28 The triangle table representation of the planning process that solves the problem of Figure 11.15 (the numbers above the columns indicate the sequence of robot actions).

robot hand is empty or it is holding a block, respectively. Each column represents the preconditions of the next F-rule to be applied.

In our case, the *plan sequence* of F-rules is the following:

- *handempty*
- *unstack* (A,C)
- *put down* (A)
- *pick up* (B)
- *stack* (B,A)
- *pick up* (C)
- *stack* (C,B)

with the preconditions shown in the previous (left) column of each F-rule being applied.

References

[1] Latombe JC. Robot motion planning. Boston, MA: Kluwer; 1991.
[2] Lozano-Perez T. Spatial planning: a configuration space approach. IEEE Trans Comput 1983;32(2):108–20.
[3] Erdmann M, Lozano-Perez T. On multiple moving obstacles. Algorithmica 1987;2 (4):477–521.
[4] Fugimura K. Motion planning in dynamic environments. Berlin/Tokyo: Springer; 1991.
[5] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. Int J Robot Res 1986;5(1):90–8.
[6] Lieberman LL, Wesley M. AUTOPASS: an automatic programming system for computer controlled mechanical assembly. IBM J Res Dev 1977;321–33.
[7] Homemde Mello LS, Sanderson AC. A correct and complete algorithm for the generation of mechanical assembly sequences. IEEE Trans Robot Autom 1990;7 (2):228–40.
[8] Sheu PCY, Xue Q. Intelligent robotic systems. Singapore/London: World Scientific Publishers;1993.
[9] Pearl J. Heuristics: intelligent search strategies for computer problem solving. Reading, MA: Addison-Wesley;1984.
[10] Bonert M. Motion planning for multi-robot. Ottawa: National Library of Canada;1999.
[11] LaValle SM. Planning algorithms. Cambridge: Cambridge University Press;2006.
[12] Popovic D, Bhatkar VP. Methods and tools for applied artificial intelligence. New York, NY: Marcel Dekker;1994.
[13] Gallina P, Gasparetto A. A technique to analytically formulate and solve the 2-dimensional constrained trajectory planning for a mobile robot. J Intell Robot Syst 2000;27(3):237–62.
[14] Lozano-Pérez T, Jones JL, Mazers E, O'Donnel P. Task-level planning of pick-and-place robot motions. IEEE Comput 1989;March:21–9.
[15] Hatzivasiliou FV, Tzafestas SG. A path planning method for mobile robots in a structured environment. In: Tzafestas SG, editor. Robotic systems: advanced techniques and applications. Dordrecht/Boston: Kluwer;1992.
[16] Kant K, Zuckler S. Toward efficient trajectory planning: the path velocity decomposition. Int J Robot Res 1986;5:72–89.

[17] Canny J. The complexity of robot motion planning. Cambridge, MA: MIT Press;1988.
[18] Garcia E, De Santos PG. Mobile-robot navigation with complete coverage of unstructured environment. Robot Auton Syst 2004;46:195–204.
[19] Russel S, Norwig P. Artificial intelligence: a modern approach. Upper Saddle River, NJ: Prentice Hall;2003.
[20] Stentz A. Optimal and efficient path planning for partially known environments. Proceedings of IEEE conference on robotics and automation. San Diego, CA; May 1994. p. 3310–17.
[21] Amato N. Randomized motion planning, Part 1, roadmap methods, course notes. University of Padova;2004.
[22] Welzl E. Constructing the visibility graph for n line segments in $O(n^2)$ time. Inf Process Lett 1985;20:161–71.
[23] Kumar V, Zefran M, Ostrowski J. Motion planning and control of robots.In:Nof S. editor.Handbook of Industrial Robotics. New York: Wiley and Sons:1999, p.295–315.
[24] Wang Y, Chirikjian GS. A new potential field method for robot path planning. Proceedings of 2000 IEEE conference robotics and automation. San Francisco, CA; April 2000. p. 977–82.
[25] Pimenta LCA. Robot navigation based on electrostatic field computation. IEEE Trans Magn 2006;42(4):1459–62.
[26] Engedy I, Horvath G. Artificial neural network based local motion planning of a wheeled mobile robot. Proceedings of 11th international symposium on computational intelligence and informatics. Budapest, Hungary; November 2010. p. 213–18.
[27] Safadi H. Local path planning using virtual potential field. Report COMP 765: spatial representation and mobile robotics—project. School of Computer Science, McGill University, Canada; April 2007.
[28] Ding FG, Jiang P, Bian XQ, Wang HJ. AUV local path planning based on virtual potential field. Proceedings of IEEE international conference on mechatronics and automation. Niagara Falls, Canada; 2005. 4, p. 1711–16.
[29] Koren Y, Borenstein J. Potential field methods and their inherent limitations for mobile robot navigation. Proceedings of IEEE conference on robotics and automation. Sacramento, CA; April 2005. p. 1398–1404.
[30] Borenstein J, Koren Y. The vector field histogram: fast obstacle avoidance for mobile robots. IEEE J Robot Autom 1991;7(3):278–88.
[31] Ulrich I. Borenstein journal of VFH*: local obstacle avoidance with look-ahead verification. Proceedings of IEEE international conference on robotics and automation. San Francisco, CA; May 2000.
[32] Wang LC, Yong LS, Ang Jr MR. Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment. Proceedings of IEEE international symposium on intelligent control. October 2002. p. 821–26.
[33] Garrido S, Moreno L, Blanco D, Jurewicz P. Path planning for mobile robot navigation using Voronoi diagram and fast marching. Int J Robot Autom 2011;2(1):42–64.
[34] Sethian JA. Theory, algorithms, and applications of level set methods for propagating interfaces. Acta Numerica, Cambridge: Cambridge University Press; 1996. p. 309–95.
[35] Sethian JA. Level set methods. Cambridge: Cambridge University Press;1996.
[36] Garrido S, Moreno L, Blanco D. Exploration of a cluttered environment using Voronoi transform and fast marching. Robot Auton Syst 2008;56(12):1069–81.
[37] Arney T. An efficient solution to autonomous path planning by approximate cell decomposition. Proceedings of international conference on information and automation for sustainability (ICIAFS 07). Colombo, Sri Lanka; December 4–6, 2007. p. 88–93.

[38] Coenen FP, Beattle B, Shave MJR, Bench-Capon TGM, Diaz GM. Spatial reasoning using the quad tesseral representation. J Artif Intell Rev 1998;12(4):321–43.
[39] Katevas NI, Tzafestas SG, Pnevmatikatos CG. The approximate cell decomposition with local node refinement global path planning method: path nodes refinement and curve parametric interpolation. J. Intell Robot Syst 1998;22:289–314.
[40] Olunloyo VOS, Ayomoh MKO. Autonomous mobile robot navigation using hybrid virtual force field concept. Eur J Sci Res 2009;31(2):204–28.
[41] Katevas NI, Tzafestas SG. The active kinematic histogram method for path planning of non-point non-holonomically constrained mobile robots. Adv Robot 1998;12 (4):375–95.
[42] Katevas NI, Tzafestas SG, Matia F. Global and local strategies for mobile robot navigation. In: Katevas N, editor. Mobile robotics in healthcare. Amsterdam, the Netherlands: IOS Press;2001.
[43] Jarvis R. Intelligent robotics: past, present and future. Int J Comput Sci Appl 2008;5 (3):23–35.
[44] Taylor T, Geva S, Boles WW. Directed exploration using modified distance transform. Proceedings of international conference on digital imaging computing: techniques and applications. Cairus, Australia; December 2012. p. 208–16.
[45] Zelinsky A, Jarvis RA, Byrne JC, Yuta S. Planning paths of complete coverage of an unstructured environment by a mobile robot. Proceedings of international symposium on advanced robotics. Tokyo, Japan; November 1993.
[46] Zelinsky A, Yuta S. A unified approach to planning, sensing and navigation for mobile robots. Proceedings of international symposium on experimental robotics. Kyoto, Japan; October 1993.
[47] Kimoto K, Yuta S.A Simulator for programming the behavior of an autonomous sensor-based mobile robot. Proceedings of international conference on intelligent robots and systems (IROS '92). Raleigh, NC; July 1992.
[48] Choi Y-H, Lee T-K, Baek S-H, Oh S-Y. Online complete coverage path planning for mobile robots based on linked spiral paths using constrained inverse distance transform. Proceedings of IEEE/RSJ international conference on intelligent robots and systems. St. Louis, MO; 2009. p. 5688–5712.
[49] Murray RM, Sastry SS. Nonholonomic motion planning: steering using sinusoids. IEEE Transactions on Automatic Control 1993;38(5):700–715.