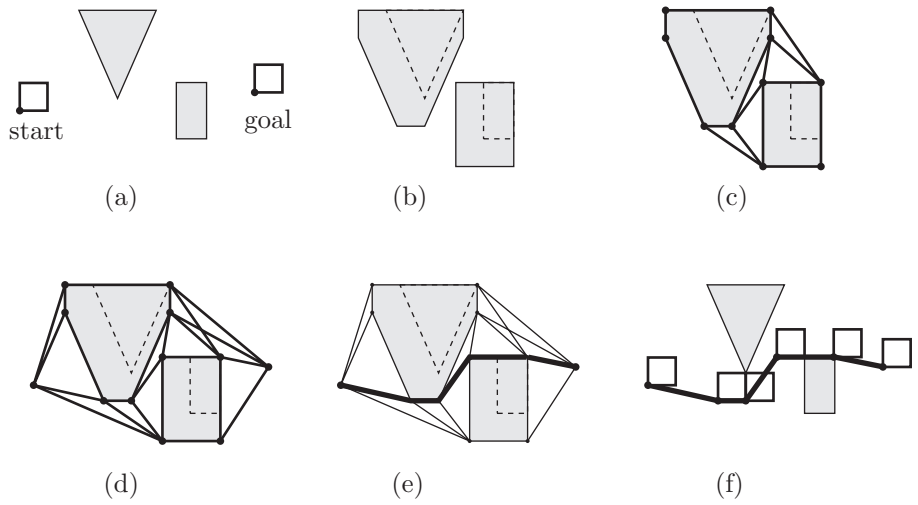

Figure 10.9: (a) The start and goal configurations for a square mobile robot (reference point shown) in an environment with a triangular and a rectangular obstacle. (b) The grown C-obstacles. (c) The visibility graph roadmap R of $\mathcal{C}_{\text{free}}$. (d) The full graph consists of R plus nodes at q_{start} and q_{goal}, along with the links connecting these nodes to the visible nodes of R. (e) Searching the graph results in the shortest path, shown in bold. (f) The robot is shown traversing the path.

path requires the robot to graze the obstacles, so we implicitly treat $\mathcal{C}_{\text{free}}$ as including its boundary.

10.4 Grid Methods

A search algorithm like A^* requires a discretization of the search space. The simplest discretization of C-space is a grid. For example, if the configuration space is n-dimensional and we desire k grid points along each dimension, the C-space is represented by k^n grid points.

The A^* algorithm can be used as a path planner for a C-space grid, with the following minor modifications:

- The definition of a "neighbor" of a grid point must be chosen: is the robot constrained to move in axis-aligned directions in configuration space or can it move in multiple dimensions simultaneously? For example, for a two-dimensional C-space, neighbors could be 4-connected (on the cardinal points of a compass: north, south, east, and west) or 8-connected

(diagonals allowed), as shown in Figure 10.10(a). If diagonal motions are allowed, the cost to diagonal neighbors should be penalized appropriately. For example, the cost to a north, south, east or west neighbor could be 1, while the cost to a diagonal neighbor could be $\sqrt{2}$. If integers are desired, for efficiency of the implementation, the approximate costs 5 and 7 could be used.

- If only axis-aligned motions are used, the heuristic cost-to-go should be based on the **Manhattan distance**, not the Euclidean distance. The Manhattan distance counts the number of "city blocks" that must be traveled, with the rule that diagonals through a block are not possible (Figure 10.10(b)).

- A node `nbr` is added to `OPEN` only if the step from `current` to `nbr` is collision-free. (The step may be considered collision-free if a grown version of the robot at `nbr` does not intersect any obstacles.)

- Other optimizations are possible, owing to the known regular structure of the grid.

An A^* grid-based path planner is resolution-complete: it will find a solution if one exists at the level of discretization of the C-space. The path will be a shortest path subject to the allowed motions.

Figure 10.10(c) illustrates grid-based path planning for the 2R robot example of Figure 10.2. The C-space is represented as a grid with $k = 32$, i.e., there is a resolution of $360°/32 = 11.25°$ for each joint. This yields a total of $32^2 = 1024$ grid points.

The grid-based planner, as described, is a single-query planner: it solves each path planning query from scratch. However, if the same q_{goal} will be used in the same environment for multiple path planning queries, it may be worth preprocessing the entire grid to enable fast path planning. This is the **wavefront** planner, illustrated in Figure 10.11.

Although grid-based path planning is easy to implement, it is only appropriate for low-dimensional C-spaces. The reason is that the number of grid points, and hence the computational complexity of the path planner, increases exponentially with the number of dimensions n. For instance, a resolution $k = 100$ in a C-space with $n = 3$ dimensions leads to $k^n = 1$ million grid nodes, while $n = 5$ leads to 10 billion grid nodes and $n = 7$ leads to 100 trillion nodes. An alternative is to reduce the resolution k along each dimension, but this leads to a coarse representation of C-space that may miss free paths.

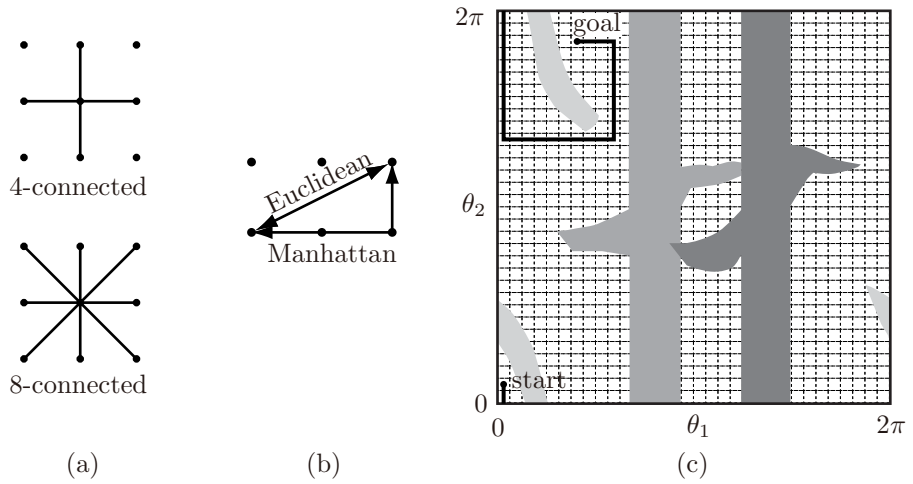

Figure 10.10: (a) A 4-connected grid point and an 8-connected grid point for a space $n = 2$. (b) Grid points spaced at unit intervals. The Euclidean distance between the two points indicated is $\sqrt{5}$ while the Manhattan distance is 3. (c) A grid representation of the C-space and a minimum-length Manhattan-distance path for the problem of Figure 10.2.

10	9	8	7	6	5	4	3	4	5	6	7	8	9	10
11					4	3	2	3				7	8	9
12	13	14			3	2	1	2				6	7	8
13	12	13			2	1	0	1	2	3	4	5	6	7
12	11	12			3	2	1	2						8
11	10				4	3	2	3						9
10	9	8	7	6	5	4	3	4						10

Figure 10.11: A wavefront planner on a two-dimensional grid. The goal configuration is given a score of 0. Then all collision-free 4-neighbors are given a score of 1. The process continues, breadth-first, with each free neighbor (that does not have a score already) assigned the score of its parent plus 1. Once every grid cell in the connected component of the goal configuration is assigned a score, planning from any location in the connected component is trivial: at every step, the robot simply moves "downhill" to a neighbor with a lower score. Grid points in collision receive a high score.

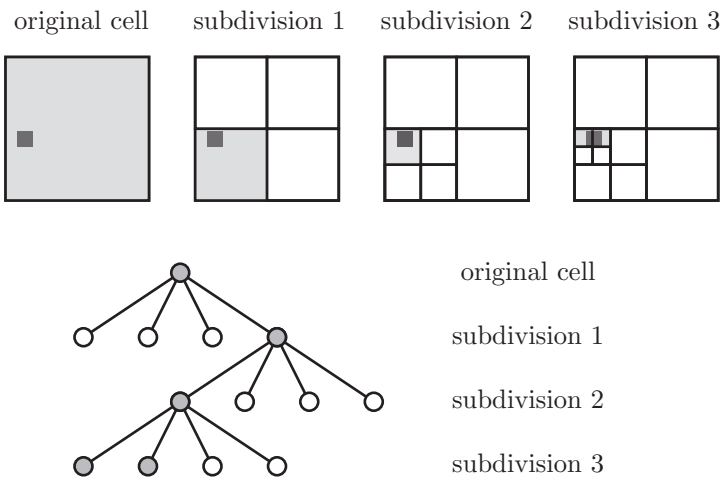

Figure 10.12: At the original C-space cell resolution, a small obstacle (indicated by the dark square) causes the whole cell to be labeled an obstacle. Subdividing the cell once shows that at least three-quarters of the cell is actually free. Three levels of subdivision results in a representation using ten total cells: four at subdivision level 3, three at subdivision level 2, and three at subdivision level 1. The cells shaded light gray are the obstacle cells in the final representation. The subdivision of the original cell is shown in the lower panel as a tree, specifically a quadtree, where the leaves of the tree are the final cells in the representation.

10.4.1 Multi-Resolution Grid Representation

One way to reduce the computational complexity of a grid-based planner is to use a multi-resolution grid representation of $\mathcal{C}_{\text{free}}$. Conceptually, a grid point is considered an obstacle if any part of the rectilinear cell centered on the grid point touches a C-obstacle. To refine the representation of the obstacle, an obstacle cell can be subdivided into smaller cells. Each dimension of the original cell is split in half, resulting in 2^n subcells for an n-dimensional space. Any cells that are still in contact with a C-obstacle are then subdivided further, up to a specified maximum resolution.

The advantage of this representation is that only the portions of C-space near obstacles are refined to high resolution, while those away from obstacles are represented by a coarse resolution. This allows the planner to find paths using short steps through cluttered spaces while taking large steps through wide open space. The idea is illustrated in Figure 10.12, which uses only 10 cells to represent an obstacle at the same resolution as a fixed grid that uses 64 cells.

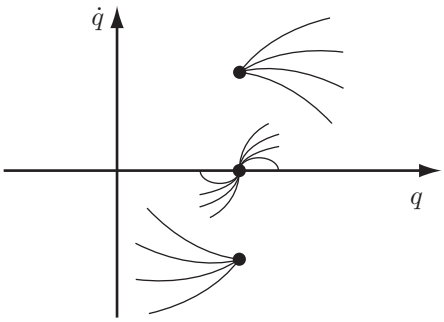

Figure 10.13: Sample trajectories emanating from three initial states in the phase space of a dynamic system with $q \in \mathbb{R}$. If the initial state has $\dot{q} > 0$, the trajectory cannot move to the left (corresponding to negative motion in q) instantaneously. Similarly, if the initial state has $\dot{q} < 0$, the trajectory cannot move to the right instantaneously.

For $n = 2$, this multi-resolution representation is called a **quadtree**, as each obstacle cell subdivides into $2^n = 4$ cells. For $n = 3$, each obstacle cell subdivides into $2^n = 8$ cells, and the representation is called an **octree**.

The multi-resolution representation of $\mathcal{C}_{\text{free}}$ can be built in advance of the search or incrementally as the search is being performed. In the latter case, if the step from `current` to `nbr` is found to be in collision, the step size can be halved until the step is free or the minimum step size is reached.

10.4.2 Grid Methods with Motion Constraints

The above grid-based planners operate under the assumption that the robot can go from one cell to any neighboring cell in a regular C-space grid. This may not be possible for some robots. For example, a car cannot reach, in one step, a "neighbor" cell that is to the side of it. Also, motions for a fast-moving robot arm should be planned in the state space, not just C-space, to take the arm dynamics into account. In the state space, the robot arm cannot move in certain directions (Figure 10.13).

Grid-based planners must be adapted to account for the motion constraints of the particular robot. In particular, the constraints may result in a directed graph. One approach is to discretize the robot controls while still making use of a grid on the C-space or state space, as appropriate. Details for a wheeled mobile robot and a dynamic robot arm are described next.

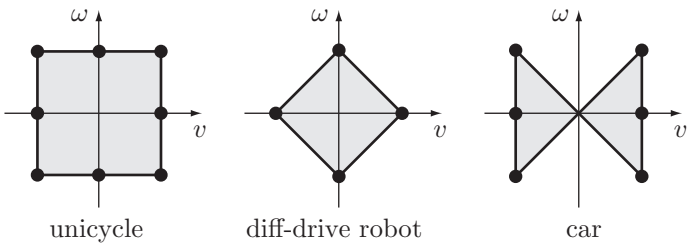

Figure 10.14: Discretizations of the control sets for unicycle, diff-drive, and car-like robots.

10.4.2.1 Grid-Based Path Planning for a Wheeled Mobile Robot

As described in Section 13.3, the controls for simplified models of unicycle, diff-drive, and car-like robots are (v, ω), i.e., the forward–backward linear velocity and the angular velocity. The control sets for these mobile robots are shown in Figure 10.14. Also shown are proposed discretizations of the controls, as dots. Other discretizations could be chosen.

Using the control discretization, we can use a variant of Dijkstra's algorithm to find short paths (Algorithm 10.2).

The search expands from q_{start} by integrating forward each control for a time Δt, creating new nodes for the paths that are collision-free. Each node keeps track of the control used to reach the node as well as the cost of the path to the node. The cost of the path to a new node is the sum of the cost of the previous node, `current`, plus the cost of the action.

Integration of the controls does not move the mobile robot to exact grid points. Instead, the C-space grid comes into play in lines 9 and 10. When a node is expanded, the grid cell it sits in is marked "occupied." Subsequently, any node in this occupied cell will be pruned from the search. This prevents the search from expanding nodes that are close by nodes reached with a lower cost.

No more than `MAXCOUNT` nodes, where `MAXCOUNT` is a value chosen by the user, are considered during the search.

The time Δt should be chosen so that each motion step is "small." The size of the grid cells should be chosen as large as possible while ensuring that integration of any control for a time Δt will move the mobile robot outside its current grid cell.

The planner terminates when `current` lies inside the goal region, or when there are no more nodes left to expand (perhaps because of obstacles), or when `MAXCOUNT` nodes have been considered. Any path found is optimal for the choice of cost function and other parameters to the problem. The planner actually

Algorithm 10.2 Grid-based Dijkstra planner for a wheeled mobile robot.

```
OPEN ← {q_start}
past_cost[q_start] ← 0
counter ← 1
while OPEN is not empty and counter < MAXCOUNT do
  current ← first node in OPEN, remove from OPEN
  if current is in the goal set then
    return SUCCESS and the path to current
  end if
  if current is not in a previously occupied C-space grid cell then
    mark grid cell occupied
    counter ← counter + 1
    for each control in the discrete control set do
      integrate control forward a short time Δt from current to q_new
      if the path to q_new is collision-free then
        compute cost of the path to q_new
        place q_new in OPEN, sorted by cost
        parent[q_new] ← current
      end if
    end for
  end if
end while
return FAILURE
```

runs faster in somewhat cluttered spaces, as the obstacles help to guide the exploration.

Some examples of motion plans for a car are shown in Figure 10.15.

10.4.2.2 Grid-Based Motion Planning for a Robot Arm

One method for planning the motion for a robot arm is to decouple the problem into a path planning problem followed by a time scaling of the path:

(a) Apply a grid-based or other path planner to find an obstacle-free path in C-space.

(b) Time scale the path to find the fastest trajectory that respects the robot's dynamics, as described in Section 9.4, or use any less aggressive time scaling.

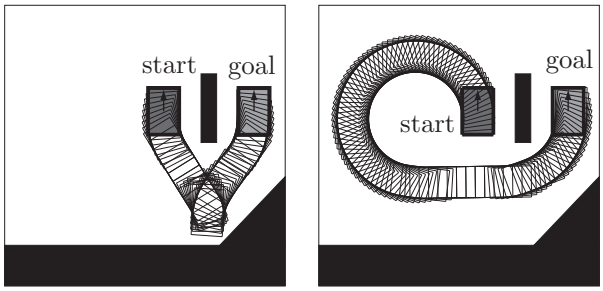

Figure 10.15: (Left) A minimum-cost path for a car-like robot where each action has identical cost, favoring a short path. (Right) A minimum-cost path where reversals are penalized. Penalizing reversals requires a modification to Algorithm 10.2.

Since the motion planning problem is broken into two steps (path planning followed by time scaling), the resultant motion will not be time-optimal in general.

Another approach is to plan directly in the state space. Given a state $(q, \dot{q})$ of the robot arm, let $\mathcal{A}(q, \dot{q})$ represent the set of accelerations that are feasible on the basis of the limited joint torques. To discretize the controls, the set $\mathcal{A}(q, \dot{q})$ is intersected with a grid of points of the form

$$\sum_{i=1}^{n} c a_i \hat{e}_i,$$

where c is an integer, $a_i > 0$ is the acceleration step size in the $\ddot{q}_i$-direction, and $\hat{e}_i$ is a unit vector in the ith direction (Figure 10.16).

As the robot moves, the acceleration set $\mathcal{A}(q, \dot{q})$ changes but the grid remains fixed. Because of this, and assuming a fixed integration time Δt at each "step" in a motion plan, the reachable states of the robot (after any integral number of steps) are confined to a grid in state space. To see this, consider a single joint angle of the robot, q_1, and assume for simplicity zero initial velocity, $\dot{q}_1(0) = 0$. The velocity at timestep k takes the form

$$\dot{q}_1(k) = \dot{q}_1(k-1) + c(k) a_1 \Delta t,$$

where $c(k)$ is any value in a finite set of integers. By induction, the velocity at any timestep must be of the form $a_1 k_v \Delta t$, where k_v is an integer. The position at timestep k takes the form

$$q_1(k) = q_1(k-1) + \dot{q}_1(k-1)\Delta t + \frac{1}{2} c(k) a_1 (\Delta t)^2.$$

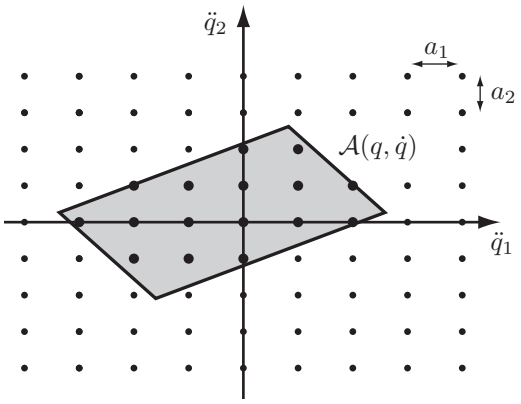

Figure 10.16: The instantaneously available acceleration set $\mathcal{A}(q, \dot{q})$ for a two-joint robot, intersected with a grid spaced at a_1 in $\ddot{q}_1$ and a_2 in $\ddot{q}_2$, gives the discretized control actions (shown as larger dots).

Substituting the velocity from the previous equation, we find that the position at any timestep must be of the form $a_1 k_p (\Delta t)^2/2 + q_1(0)$, where k_p is an integer.

To find a trajectory from a start node to a goal set, a breadth-first search can be employed to create a search tree on the state space nodes. When exploration is made from a node $(q, \dot{q})$ in the state space, the set $\mathcal{A}(q, \dot{q})$ is evaluated to find the discrete set of control actions. New nodes are created by integrating the control actions for time Δt. A node is discarded if the path to it is in collision or if it has been reached previously (i.e., by a trajectory taking the same or less time).

Because the joint angles and angular velocities are bounded, the state space grid is finite and therefore it can be searched in finite time. The planner is resolution-complete and returns a time-optimal trajectory, subject to the resolution specified in the control grid and timestep Δt.

The control-grid step sizes a_i must be chosen small enough that $\mathcal{A}(q, \dot{q})$, for any feasible state $(q, \dot{q})$, contains a representative set of points of the control grid. Choosing a finer grid for the controls, or a smaller timestep Δt, creates a finer grid in the state space and a higher likelihood of finding a solution amidst obstacles. It also allows the choice of a smaller goal set while keeping points of the state space grid inside the set.

Finer discretization comes at a computational cost. If the resolution of the control discretization is increased by a factor r in each dimension (i.e., each a_i is reduced to a_i/r), and the timestep size is divided by a factor τ, the computation time spent growing the search tree for a given robot motion duration increases

by a factor $r^{n\tau}$, where n is the number of joints. For example, increasing the control-grid resolution by a factor $r = 2$ and decreasing the timestep by a factor $\tau = 4$ for a three-joint robot results in a search that is likely to take $2^{3\times 4} = 4096$ times longer to complete. The high computational complexity of the planner makes it impractical beyond a few degrees of freedom.

The description above ignores one important issue: the feasible control set $\mathcal{A}(q, \dot{q})$ changes during a timestep, so the control chosen at the beginning of the timestep may no longer be feasible by the end of the timestep. For that reason, a conservative approximation $\tilde{\mathcal{A}}(q, \dot{q}) \subset \mathcal{A}(q, \dot{q})$ should be used instead. This set should remain feasible over the duration of a timestep regardless of which control action is chosen. How to determine such a conservative approximation $\tilde{\mathcal{A}}(q, \dot{q})$ is beyond the scope of this chapter, but it has to do with bounds on how rapidly the arm's mass matrix $M(q)$ changes with q and how fast the robot is moving. At low speeds $\dot{q}$ and short durations Δt, the conservative set $\tilde{\mathcal{A}}(q, \dot{q})$ is very close to $\mathcal{A}(q, \dot{q})$.

10.5 Sampling Methods

Each grid-based method discussed above delivers optimal solutions subject to the chosen discretization. A drawback of these approaches, however, is their high computational complexity, making them unsuitable for systems having more than a few degrees of freedom.

A different class of planners, known as sampling methods, relies on a random or deterministic function to choose a sample from the C-space or state space: a function to evaluate whether a sample or motion is in $\mathcal{X}_{\text{free}}$; a function to determine nearby previous free-space samples; and a simple local planner to try to connect to, or move toward, the new sample. These functions are used to build up a graph or tree representing feasible motions of the robot.

Sampling methods generally give up on the resolution-optimal solutions of a grid search in exchange for the ability to find satisficing solutions quickly in high-dimensional state spaces. The samples are chosen to form a roadmap or search tree that quickly approximates the free space $\mathcal{X}_{\text{free}}$ using fewer samples than would typically be required by a fixed high-resolution grid, where the number of grid points increases exponentially with the dimension of the search space. Most sampling methods are probabilistically complete: the probability of finding a solution, when one exists, approaches 100% as the number of samples goes to infinity.

Two major classes of sampling methods are rapidly exploring random trees (RRTs) and probabilistic roadmaps (PRMs). The former use a tree representation for single-query planning in either C-space or state space, while PRMs