Principles of Robot Motion: Theory, Algorithms, and Implementation ERRATA!!!! [1]

Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor,
Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun

August 23, 2010

Chapter 1

Introduction

No bugs to report, yet!

Chapter 2

Bug Algorithms

Even a simple planner can present interesting and difficult issues. The Bug1 and Bug2 algorithms [289] are among the earliest and simplest sensor-based planners with provable guarantees. These algorithms assume the robot is a point operating in the plane with a contact sensor or a zero range sensor to detect obstacles. When the robot has a finite range (non-zero range) sensor, then the Tangent Bug algorithm [208] is a Bug derivative that can use that sensor information to find shorter paths to the goal. The Bug and Bug-like algorithms are straightforward to implement; moreover, a simple analysis shows that their success is guaranteed, when possible. These algorithms require two behaviors: move on a straight line and follow a boundary. To handle boundary-following, we introduce a curve-tracing technique based on the implicit function theorem at the end of this chapter. This technique is general to following any path, but we focus on following a boundary at a fixed distance.

2.1 Bug1 and Bug2

Perhaps the most straight forward path planning approach is to move toward the goal, unless an obstacle is encountered, in which case, circumnavigate the obstacle until motion toward the goal is once again allowable. Essentially, the Bug1 algorithm formalizes the "common sense" idea of moving toward the goal and going around obstacles. The robot is assumed to be a point with perfect positioning (no positioning error) with a contact sensor that can detect an obstacle boundary if the point robot "touches" it. The robot can also measure the distance $d(x, y)$ between any two points x and y. Finally, assume that the workspace is *bounded*. Let $B_r(x)$ denote a ball of radius

r centered on x, i.e., $B_r(x) = \{y \in \mathbb{R}^2 \,|\, d(x,y) < r\}$. The fact that the workspace is bounded implies that for all $x \in \mathcal{W}$, there exists an r such that $\mathcal{W} \subset B_r(x)$.

The start and goal are labeled q_{start} and q_{goal}, respectively. Let $q_0^L = q_{\text{start}}$ and the m-line be the line segment that connects q_i^L to q_{goal}. Initially, $i = 0$. The Bug1 algorithm exhibits two behaviors: motion-to-goal and boundary-following. During motion-to-goal, the robot moves along the m-line toward q_{goal} until it either encounters the goal or an obstacle. If the robot encounters an obstacle, let q_1^H be the point where the robot first encounters an obstacle and call this point a *hit point*. The robot then circumnavigates the obstacle until it returns to q_1^H. Then, the robot determines the closest point to the goal on the perimeter of the obstacle and traverses to this point. This point is called a *leave point* and is labeled q_1^L. From q_1^L, the robot heads straight toward the goal again, i.e., it reinvokes the motion-to-goal behavior. If the line that connects q_1^L and the goal intersects the current obstacle, then there is no path to the goal; note that this intersection would occur immediately "after" leaving q_1^L. Otherwise, the index i is incremented and this procedure is then repeated for q_i^L and q_i^H until the goal is reached or the planner determines that the robot cannot reach the goal (figures 2.1, 2.2). Finally, if the line to the goal "grazes" an obstacle, the robot need not invoke a boundary following behavior, but rather continues onward toward the goal. See algorithm 1 for a description of the Bug1 approach.

Like its Bug1 sibling, the Bug2 algorithm exhibits two behaviors: motion-to-goal and boundary-following. During motion-to-goal, the robot moves toward the goal on the m-line; however, in Bug2 the m-line connects q_{start} and q_{goal}, and thus remains fixed. The boundary-following behavior is invoked if the robot encounters an obstacle, but this behavior is different from that of Bug1. For Bug2, the robot circumnavigates the obstacle until it reaches a new point on the m-line closer to the goal than the initial point of contact with the obstacle. At this time, the robot proceeds toward the goal, repeating this process if it encounters an object. If the robot re-encounters the original departure point from the m-line, then the robot concludes there is no path to the goal (figures 2.3, 2.4).

Let $x \in \mathcal{W}_{\text{free}} \subset \mathbb{R}^2$ be the current position of the robot, $i = 1$, and q_0^L be the start location. See algorithm 2 for a description of the Bug2 approach.

At first glance, it seems that Bug2 is a more effective algorithm than Bug1 because the robot does not have to entirely circumnavigate the obstacles; however, this is not always the case. This can be seen by comparing the lengths of the paths found by the two algorithms. For Bug1, when

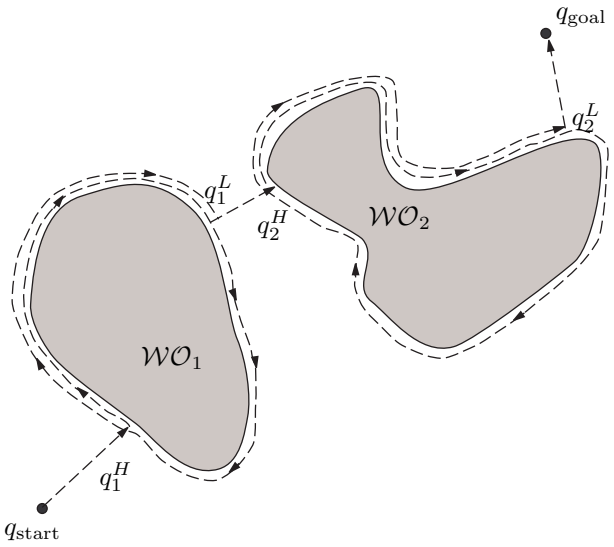

FIGURE 2.1. The Bug1 algorithm successfully finds the goal.

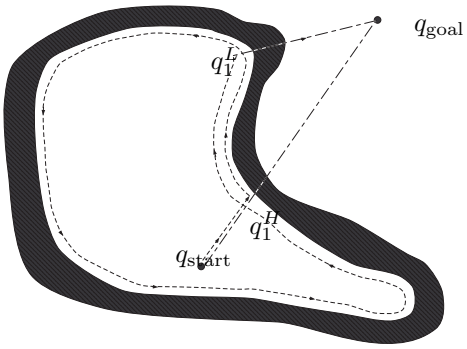

FIGURE 2.2. The Bug1 algorithm reports the goal is unreachable.

the ith obstacle is encountered, the robot completely circumnavigates the boundary, and then returns to the leave point. In the worst case, the robot must traverse half the perimeter, p_i, of the obstacle to reach this leave point. Moreover, in the worst case, the robot encounters all n obstacles. If there

Algorithm 1 Bug1 Algorithm

Input: A point robot with a tactile sensor

Output: A path to the q_{goal} or a conclusion no such path exists

```
1:  while Forever do
2:    repeat
3:      From q_{i-1}^L, move toward q_goal.
4:    until q_goal is reached or an obstacle is encountered at q_i^H.
5:    if Goal is reached then
6:      Exit.
7:    end if
8:    repeat
9:      Follow the obstacle boundary.
10:   until q_goal is reached or q_i^H is re-encountered.
11:   Determine the point q_i^L on the perimeter that has the shortest distance
      to the goal.
12:   Go to q_i^L.
13:   if the robot were not able to move toward the goal then
14:     Conclude q_goal is not reachable and exit.
15:   end if
16: end while
```

are no obstacles, the robot must traverse a distance of length $d(q_{\text{start}}, q_{\text{goal}})$. Thus, we obtain

$$L_{\text{Bug1}} \leq d(q_{\text{start}}, q_{\text{goal}}) + 1.5 \sum_{i=1}^{n} p_i. \tag{2.1}$$

For Bug2, the path length is a bit more complicated. Suppose that the line through q_{start} and q_{goal} intersects the ith obstacle n_i times. Then, there are at most n_i leave points for this obstacle, since the robot may only leave the obstacle when it returns to a point on this line. It is easy to see that half of these intersection points are not valid leave points because they lie on the "wrong side" of the obstacle, i.e., moving toward the goal would cause a collision. In the worst case, the robot will traverse nearly the entire perimeter of the obstacle for each leave point. Thus, we obtain

$$L_{\text{Bug2}} \leq d(q_{\text{start}}, q_{\text{goal}}) + \frac{1}{2} \sum_{i=1}^{n} n_i p_i. \tag{2.2}$$

Naturally, (2.2) is an upper-bound because the summation is over all of the

Algorithm 2 Bug2 Algorithm

Input: A point robot with a tactile sensor

Output: A path to q_{goal} or a conclusion no such path exists

while True **do**
  repeat
    From q_{i-1}^L, move toward q_{goal} along m-line.
  until
  q_{goal} is reached **or**
  an obstacle is encountered at *hit point* q_i^H.
  Turn left (or right).
  repeat
    Follow boundary
  until
    q_{goal} is reached **or**
    q_i^H is re-encountered **or**
    m-line is re-encountered at a point m such that
      $m \neq q_i^H$ (robot did not reach the hit point),
      $d(m, q_{\text{goal}}) < d(m, q_i^H)$ (robot is closer), and
      if robot moves toward goal, it would not hit the obstacle
  if Goal is reached **then**
    Exit.
  end if
  if q_i^H is re-encountered **then**
    Conclude goal is unreachable
  end if
  Let $q_{i+1}^L = m$
  Increment i
end while

obstacles as opposed to over the set of obstacles that are encountered by the robot.

A casual examination of (2.1) and (2.2) shows that L_{Bug2} can be arbitrarily longer than L_{Bug1}. This can be achieved by constructing an obstacle whose boundary has many intersections with the m-line. Thus, as the "complexity" of the obstacle increases, it becomes increasingly likely that Bug1 could outperform Bug2 (figure 2.4).

In fact, Bug1 and Bug2 illustrate two basic approaches to search problems. For each obstacle that it encounters, Bug1 performs an *exhaustive search* to find the optimal leave point. This requires that Bug1 traverse

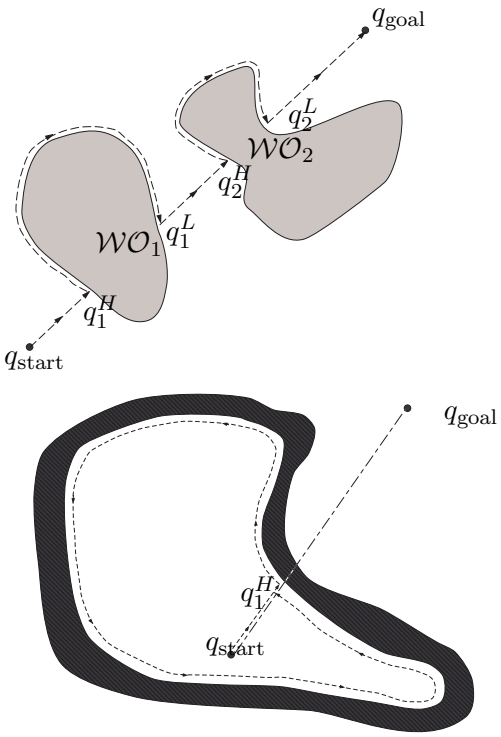

FIGURE 2.3. (Top) The Bug2 algorithm finds a path to the goal. (Bottom) The Bug2 algorithm reports failure.

the entire perimeter of the obstacle, but having done so, it is certain to have found the optimal leave point. In contrast, Bug2 uses an *opportunistic* approach. When Bug2 finds a leave point that is better than any it has seen before, it commits to that leave point. Such an algorithm is also called *greedy*, since it opts for the first promising option that is found. When the obstacles are simple, the greedy approach of Bug2 gives a quick payoff, but when the obstacles are complex, the more conservative approach of Bug1 often yields better performance.

2.2 Tangent Bug

Tangent Bug [207] serves as an improvement to the Bug2 algorithm in that it

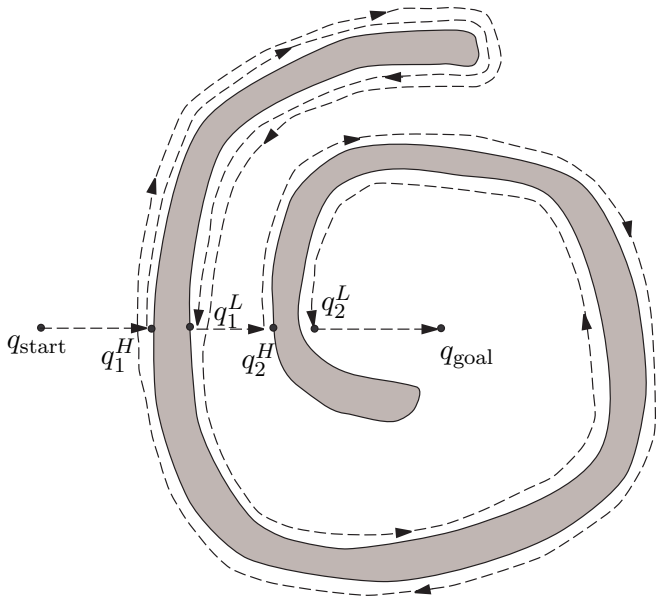

FIGURE 2.4. Bug2 Algorithm.

determines a shorter path to the goal using a range sensor with a 360 degree infinite orientation resolution. Sometimes orientation is called *azimuth.* We model this range sensor with the *raw distance function* $\rho\colon \mathbb{R}^2 \times S^1 \to \mathbb{R}$. Consider a point robot situated at $x \in \mathbb{R}^2$ with rays radially emanating from it. For each $\theta \in S^1$, the value $\rho(x, \theta)$ is the distance to the closest obstacle along the ray from x at an angle θ. More formally,

$$\rho(x,\theta) = \min_{\lambda \in [0,\infty]} d(x, x + \lambda[\cos\theta, \sin\theta]^T), \quad \text{such that} \quad x + \lambda[\cos\theta, \sin\theta]^T \in \bigcup_i \mathcal{WO}_i. \quad (2.3)$$

Note that there are infinitely many $\theta \in S^1$ and hence the infinite resolution. This assumption is approximated with a finite number of range sensors situated along the circumference of a circular mobile robot which we have modeled as a point.

Since real sensors have limited range, we define the *saturated raw distance function* , denoted $\rho_R\colon \mathbb{R}^2 \times S^1 \to \mathbb{R}$, which takes on the same values as ρ when the obstacle is within sensing range, and has a value of infinity when

the ray lengths are greater than the sensing range, R, meaning that the obstacles are outside the sensing range. More formally,

$$\rho_R(x,\theta) = \begin{cases} \rho(x,\theta), & \text{if } \rho(x,\theta) < R \\ \infty, & \text{otherwise.} \end{cases}$$

The set of points within sensing range of the robot can be denoted by

$$V_R(x) = \{y \in \mathcal{Q}_{\text{free}} | d(x,y) < R \text{and} \lambda x + (1-\lambda)y \in \mathcal{W}_{\text{free}} \text{for all} \lambda \in [0,1]\}$$

The Tangent Bug planner assumes that the robot can detect discontinuities in ρ_R as depicted in figure **??**. For a fixed $x \in \mathbb{R}^2$, an *interval of continuity* is defined to be a connected set of points on $\partial V(x)$ where ρ_R varies continuously and is finite. These are the points $x + \rho(x,\theta)[\cos\theta, \sin\theta]^T$ on the boundary of the free space where $\rho_R(x,\theta)$ is finite and continuous with respect to θ.

The endpoints of these intervals occur where $\rho_R(x,\theta)$ loses continuity, either as a result of one obstacle blocking another or the sensor reaching its range limit. The endpoints are denoted O_i. Figure 2.5 contains an example where ρ_R loses continuity. The points $O_1, O_2, O_3, O_5, O_6, O_7$, and O_8 correspond to losses of continuity associated with obstacles blocking other portions of $\mathcal{W}_{\text{free}}$; note the rays are tangent to the obstacles here. The point O_4 is a discontinuity because the obstacle boundary falls out of range of the sensor. The sets of points on the boundary of the free space between O_1 and O_2, O_3 and O_4, O_5 and O_6, O_7 and O_8 are the intervals of continuity.

Just like the other Bugs, Tangent Bug (algorithm 3) iterates between two behaviors: motion-to-goal and boundary-following. However, these behaviors are different than in the Bug1 and Bug2 approaches. Although motion-to-goal directs the robot to the goal, this behavior may have a phase where the robot follows the boundary. Likewise, the boundary-following behavior may have a phase where the robot does not follow the boundary.

The robot initially invokes the motion-to-goal behavior, which itself has two parts. First, the robot attempts to move in a straight line toward the goal until it senses an obstacle R units away and directly between it and the goal. This means that a line segment connecting the robot and goal must intersect an interval of continuity. For example, in figure 2.6, $\mathcal{WO}_2$ is within sensing range, but does not block the goal, but $\mathcal{WO}_1$ does. When the robot initially senses an obstacle, the circle of radius R becomes tangent to the obstacle. Immediately after, this tangent point splits into two O_i's, which are the endpoints of the interval. If the obstacle is in front of the robot, then this interval intersects the segment connecting the robot and the goal.

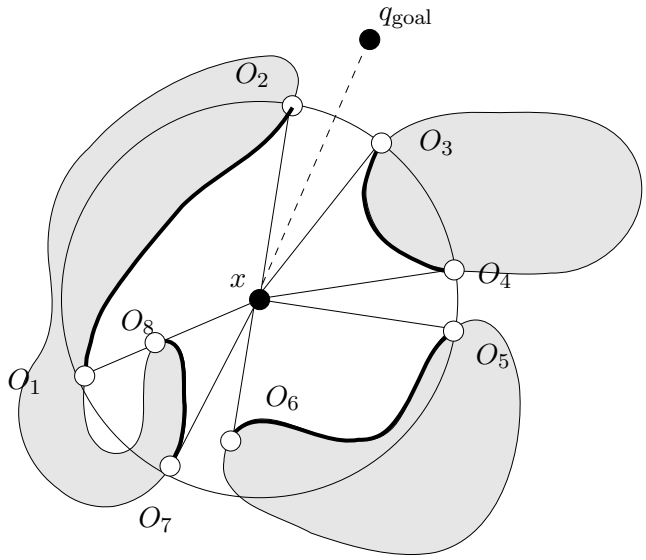

FIGURE 2.5. The points of discontinuity of $\rho_R(x, \theta)$ correspond to points O_i on the obstacles. The thick solid curves represent connected components of the range of $\rho_R(x, \theta)$, i.e., the intervals of continuity. In this example, the robot, to the best of its sensing range, believes there is a straight-line path to the goal.

Consider the O_i where $d(O_i, q_{\text{goal}}) < d(x, q_{\text{goal}})$. The robot then moves toward one of these O_i that maximally decreases a heuristic distance to the goal. An example of a heuristic distance is the sum $d(x, O_i) + d(O_i, q_{\text{goal}})$. (The heuristic distance can be more complicated when factoring in available information with regard to the obstacles.) In figure 2.7 (left), the robot sees $\mathcal{WO}_1$ and drives to O_2 because $i = 2$ minimizes $d(x, O_i)+d(O_i, q_{\text{goal}})$. When the robot is located at x, it cannot know that $\mathcal{WO}_2$ blocks the path from O_2 to the goal. In figure 2.7(right), when the robot is located at x but the goal is different, it has enough sensor information to conclude that $\mathcal{WO}_2$ indeed blocks a path from O_2 to the goal, and therefore drives toward O_4. So, even though driving toward O_2 may initially minimize $d(x, O_i)+d(O_i, q_{\text{goal}})$ more than driving toward O_4, the planner effectively assigns an infinite cost to $d(O_2, q_{\text{goal}})$ because it has enough information to conclude that any path through O_2 will be suboptimal.

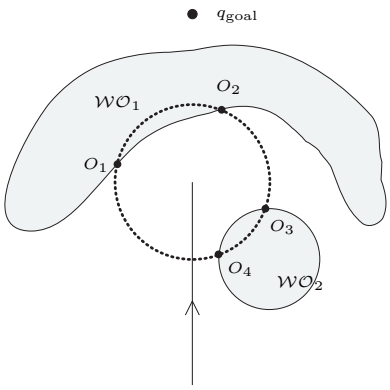

FIGURE 2.6. The vertical represents the path of the robot and the dotted circle its sensing range. Currently, the robot is located at the "top" of the line segment. The points O_i represent the points of discontinuity of the saturated raw distance function. Note that the robot passes by $\mathcal{WO}_2$.

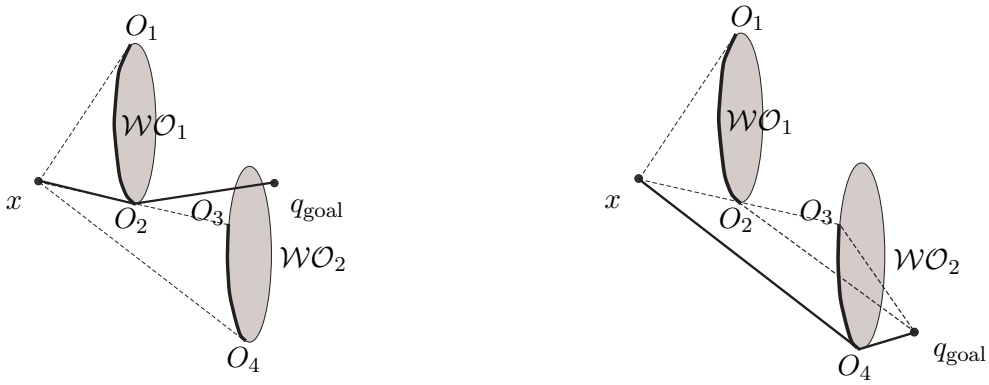

FIGURE 2.7. (Left) The planner selects O_2 as a subgoal for the robot. (Right) The planner selects O_4 as a subgoal for the robot. Note the line segment between O_4 and q_{goal} cuts through the obstacle.

The set $\{O_i\}$ is continuously updated as the robot moves toward a particular O_i, which can be seen in figure 2.8. At $t = 1$, the robot has not

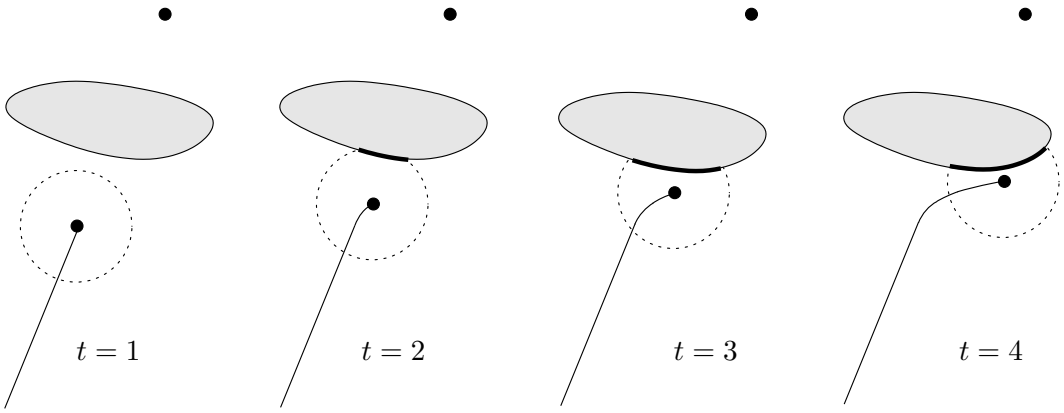

FIGURE 2.8. Demonstration of motion-to-goal behavior for a robot with a finite sensor range moving toward a goal which is "above" the light gray obstacle.

sensed the obstacle, hence the robot moves toward the goal. At $t = 2$, the robot initially senses the obstacle, depicted by a thick solid curve. The robot continues to move toward the goal, but off to the side of the obstacle heading toward the discontinuity in ρ. For $t = 3$ and $t = 4$, the robot senses more of the obstacle and continues to decrease distance toward the goal while hugging the boundary.

The robot undergoes motion-to-goal until it can no longer decrease the heuristic distance to the goal following the rules of motion-to-goal. Put differently, it finds a point that is like a local minimum of $d(\cdot, O_i)+d(O_i, q_{\text{goal}})$ restricted to the path that motion-to-goal dictates.

When the robot switches to boundary-following, it determines the point M on the currently sensed portion of the blocking obstacle that has the shortest distance to the goal. The robot then moves in the same direction as if it were in the motion - to - goal behavior. It continuously moves toward the O_i on the blocking obstacle in the chosen direction (figure 2.9). While undergoing this motion, the planner also updates two values: d_{min} and d_{leave}. The value d_{min} is the shortest distance between the goal and any point on the blocking obstacle boundary that has been sensed thus far. The value d_{leave} is the shortest distance between the goal and any point in the currently sensed environment at that robot location, i.e., $\partial V_R(x)$. Let $d_{\text{leave}}(x) = min_{y \in V(x)} d(x, y)$. When $d_{\text{leave}}(x) < d_{\text{min}}$, the robot terminates the boundary-following behavior.

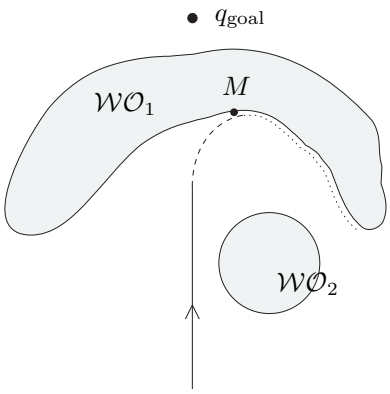

FIGURE 2.9. The workspace is the same as in figure 2.6. The solid and dashed segments represent the path generated by motion-to-goal and the dotted path represents the boundary-following path. Note that M is the "local minimum" point.

Note that in many cases, when the robot terminates the boundary-following behavior, it will drive toward the goal. In such a case, there is no blocking obstacle; define T to be the point where a circle, centered at x of radius R, intersects the segment that connects x and q_{goal}. This is the point on the periphery of the sensing range that is closest to the goal when the robot is located at x. When there is no blocking obstacle, $d_{\text{leave}} = d(T, q_{\text{goal}})$. Otherwise, T is undefined and d_{leave} is constantly updated to be the shortest distance between the goal and any point on an obstacle boundary that is viewable within the robot's current sensor range.

Figure 2.10 contains a path for a robot with zero sensor range. Here the robot invokes a motion-to-goal behavior until it encounters the first obstacle at hit point H_1. Unlike Bug1 and Bug2, encountering a hit point does not change the behavior mode for the robot. The robot continues with the motion-to-goal behavior by turning right and following the boundary of the first obstacle. The robot turned right because that direction minimized its heuristic distance to the goal. The robot departs this boundary at a *depart point* D_1. The robot continues with the motion-to-goal behavior, maneuvering around a second obstacle, until it encounters the third obstacle at H_3. The robot turns left and continues to invoke the motion-to-goal behavior until it reaches M_3, a minimum point. Now, the planner invokes

Algorithm 3 Tangent Bug Algorithm

Input: A point robot with a range sensor

Output: A path to the q_{goal} or a conclusion no such path exists

1: **while** True **do**
2: **repeat**
3: Continuously move toward the point $n \in \{T, O_i\}$ which minimizes $d(x, n) + d(n, q_{\text{goal}})$ where $d(n, q_{\text{goal}}) < d(x, q_{\text{goal}})$
4: **until**

- the goal is encountered **or**
- The direction that minimizes $d(x, n) + d(n, q_{\text{goal}})$ begins to increase $d(x, q_{\text{goal}})$, i.e., the robot detects a "local minimum" of $d(\cdot, q_{\text{goal}})$.

5: Choose a boundary following direction which continues in the same direction as the most recent motion-to-goal direction.
6: **repeat**
7: Continuously update d_{leave}, $d_{\min}$, and $\{O_i\}$.
8: Continuously moves toward $n \in \{O_i\}$ that is in the chosen boundary direction.
9: **until**

- The goal is reached.
- The robot completes a cycle around the obstacle in which case the goal cannot be achieved.
- $d_{\text{leave}} < d_{\min}$

10: **end while**

the boundary-following behavior until the robot reaches L_3. Note that since we have zero sensing range, d_{leave} is the distance between the robot and the goal. The procedure continues until the robot reaches the goal. Only at M_i and L_i does the robot switch between behaviors.

Figures 2.11 and 2.12 contain examples where the robot has finite and infinite sensing ranges, respectively. Note that in these examples, since the robot has a non-zero sensor range, it does not reach an M_i but rather reaches an sw_i where it detects its corresponding M_i. The sw_i are sometimes called *switch points.*

Figures 2.13 demonstrates a situation in which the robot invokes bound-

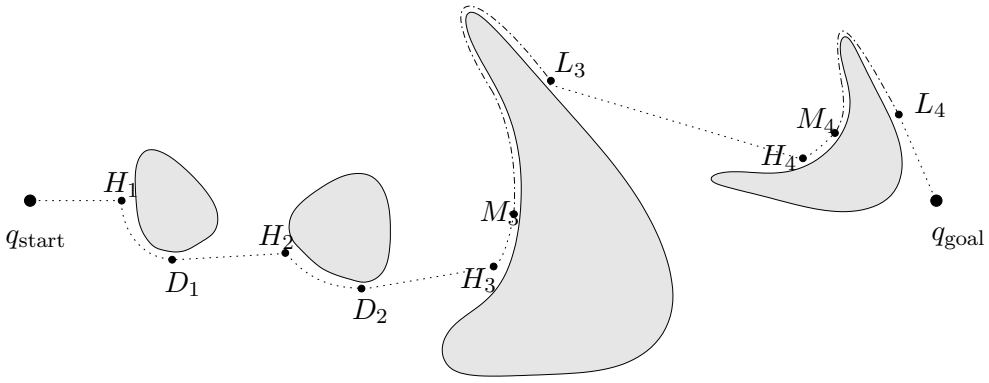

FIGURE 2.10. The path generated by Tangent Bug with zero sensor range. The dashed lines correspond to the motion - to - goal behavior and the dotted lines correspond to boundary-following.

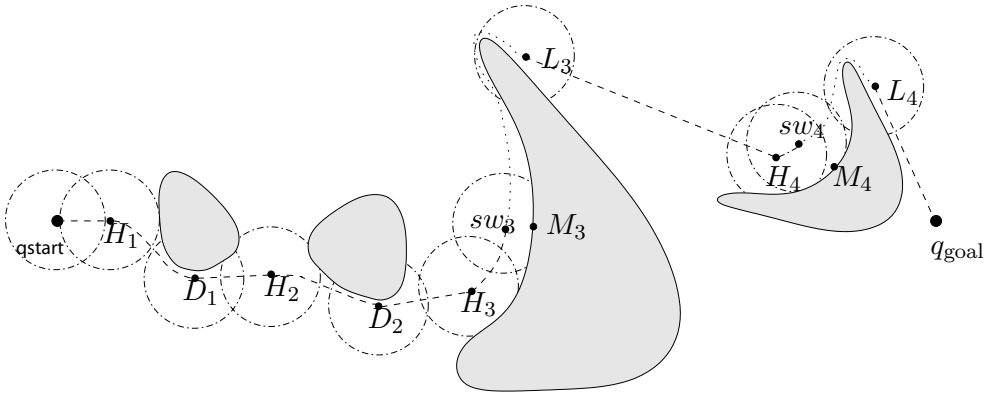

FIGURE 2.11. Path generated by Tangent Bug with finite sensor range. The dashed lines correspond to the motion - to - goal behavior and the dotted lines correspond to boundary-following. The dashed-dotted circles correspond to the sensor range of the robot.

ary following at a minimum point M_1 but must update the value for $d_{\min}$ when it encounters another minimum M_2. The boundary following behavior in this case continues until the robot reaches L_2.

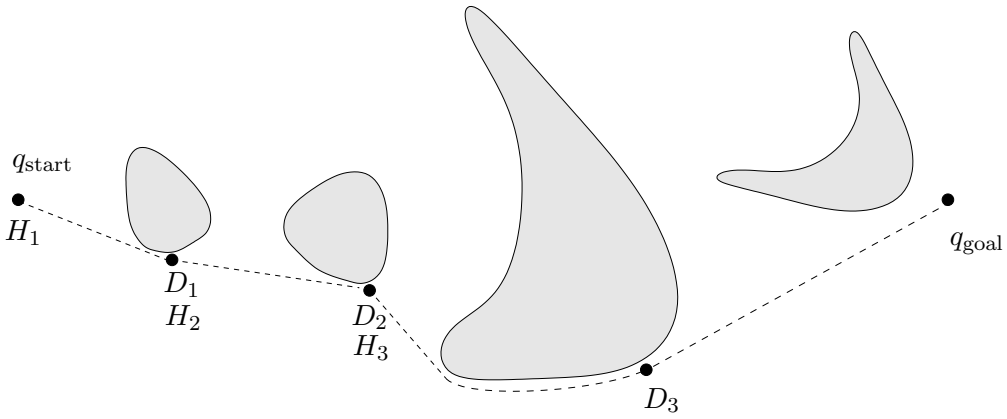

FIGURE 2.12. Path generated by Tangent Bug with infinite sensor range. The dashed-lines correspond to the motion - to - goal behavior and there is no boundary-following.

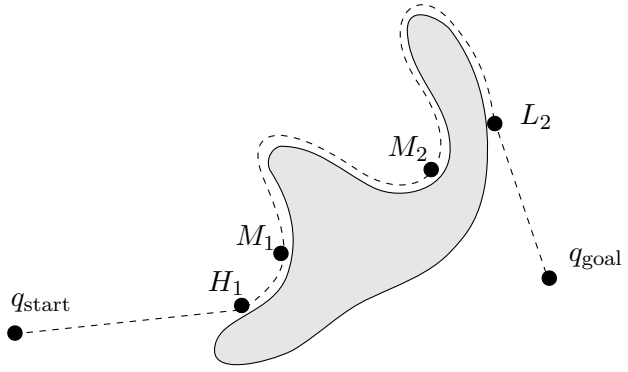

FIGURE 2.13. Path generated by Tangent Bug with zero sensor range and an update to $d_{\min}$ due to two minima in the blocking obstacle.

2.3 Implementation

Essentially, the bug algorithms have two behaviors: drive toward a point and follow an obstacle. The first behavior is simply a form of gradient descent of $d(\cdot, n)$ where n is either q_{goal} or an O_i. The second behavior, boundary-following, presents a challenge because the obstacle boundary is not known *a priori.* Therefore, the robot planner must rely on sensor information

to determine the path. However, we must concede that the full path will not be determined from one sensor reading: the sensing range of the robot may be limited and the robot may not be able to "see" the entire world from one vantage point. So, the robot planner has to be incremental. We must determine first what information the robot requires and then where the robot should move to acquire more information. This is indeed the challenge of sensor-based planning. Ideally, we would like this approach to be reactive with sensory information feeding into a simple algorithm that outputs translational and rotational velocity for the robot.

There are three questions: What information does the robot require to circumnavigate the obstacle? How does the robot infer this information from its sensor data? How does the robot use this information to determine (locally) a path?

2.3.1 What Information: The Tangent Line

If the obstacle were flat, such as a long wall in a corridor, then following the obstacle is trivial: simply move parallel to the obstacle. This is readily implemented using a sensing system that can determine the obstacle's surface normal $n(x)$, and hence a direction parallel to its surface. However, the world is not necessarily populated with flat obstacles; many have non-zero curvature. However, the robot can follow a path that is consistently orthogonal to the surface normal; this direction can be written as $n(x)^{\perp}$ and the resulting path satisfies $\dot{c}(t) = v$ where v is a basis vector in $(n(c(t)))^{\perp}$. The sign of v is based on the "previous" direction of $\dot{c}$.

Consistently determining the surface normal can be quite challenging and therefore for implementation, we can assume that obstacles are "locally flat." This means the sensing system determines the surface normal, the robot moves orthogonal to this normal for a short distance, and then the process repeats. In a sense, the robot determines the sequence of short straight-line segments to follow based on sensor information.

This flat line, loosely speaking, is the tangent (figure 2.14). It is a linear approximation of the curve at the point where the tangent intersects the curve. The tangent can also be viewed as a first-order approximation to the function that describes the curve. Let $c\colon [0,1] \to \mathcal{Q}_{\text{free}}$ be the function that defines a path. Let $x = c(s_0)$ for a $s_0 \in [0,1]$. The tangent at x is $\frac{dc}{ds}\big|_{s=s_0}$. The tangent space can be viewed as a line whose basis vector is $\frac{dc}{ds}\big|_{s=s_0}$, i.e., $\left\{\alpha \frac{dc}{ds}\big|_{s=s_0} \mid \alpha \in \mathbb{R}\right\}$.

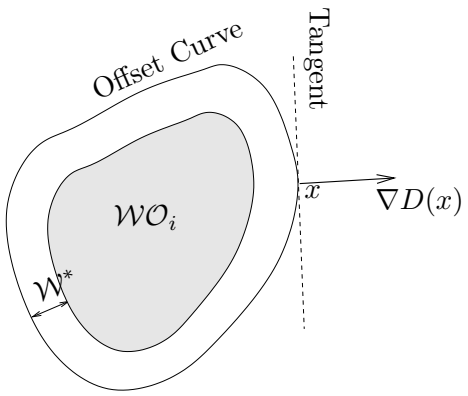

FIGURE 2.14. The solid curve is the offset curve. The dashed line represents the tangent to the offset curve at x.

2.3.2 How to Infer Information with Sensors: Distance and Gradient

The next step is to infer the tangent from sensor data. Instead of thinking of the robot as a point in the plane, let's think of it as a circular base which has a fine array of tactile sensors radially distributed along its circumference (figure 2.15). When the robot contacts an obstacle, the direction from the contacted sensor to the robot's center approximates the surface normal. With this information, the robot can determine a sequence of tangents to follow the obstacle.

Unfortunately, using a tactile sensor to prescribe a path requires the robot to collide with obstacles, which endangers the obstacles and the robot. Instead, the robot should follow a path at a safe distance $\mathcal{W}^* \in \mathbb{R}$ from the nearest obstacle. Such a path is called an *offset curve* [360]. Let $D(x)$ be the distance from x to the closest obstacle, i.e.,

$$D(x) \quad = \min_{c \in \bigcup_i \mathcal{WO}_i} d(x, c). \tag{2.4}$$

To measure this distance with a mobile robot equipped with an onboard range sensing ring, we use the raw distance function again. However, instead of looking for discontinuities, we look for the global minimum. In other words, $D(x) = \min_s \rho(x, s)$ (figure 2.16).

We will need to use the gradient of distance. In general, the gradient is a vector that points in the direction that maximally increases the value of

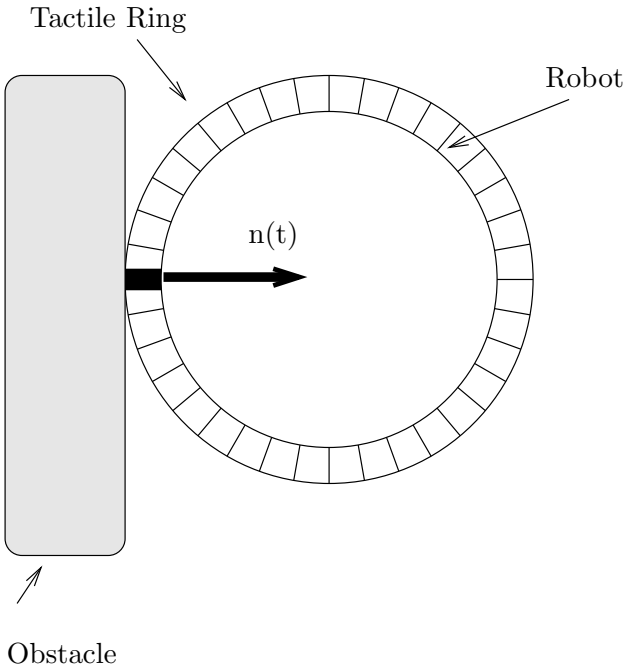

FIGURE 2.15. A fine-resolution tactile sensor.

a function. See appendix **??** for more details. Typically, the ith component of the gradient vector is the partial derivative of the function with respect to its ith coordinate. In the plane, $\nabla D(x) = [\frac{\partial D(x)}{\partial x_1} \quad \frac{\partial D(x)}{\partial x_2}]^T$ which points in the direction that increases distance the most. Finally, the gradient is the unit direction associated with the smallest value of the raw distance function. Since the raw distance function seemingly approximates a sensing system with individual range sensing elements radially distributed around the perimeter of the robot, an algorithm defined in terms of D can often be implemented using realistic sensors.

There are many choices for range sensors; here, we investigate the use of ultrasonic sensors (figure 2.17), which are commonly found on mobile robots. Conventional ultrasonic sensors measure distance using time of flight. When the speed of sound in air is constant, the time that the ultrasound requires to leave the transducer, strike an object, and return is proportional to the distance to the point of reflection on the object [106]. This object, however, can be located anywhere along the angular spread of the sonar sensor's beam pattern (figure 2.18). Therefore, the distance information that sonars

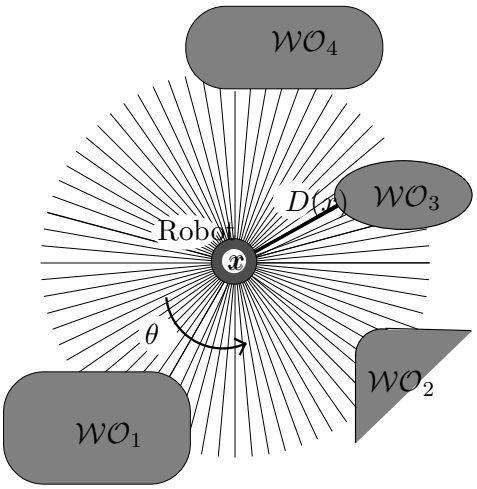

FIGURE 2.16. The global minimum of the rays determines the distance to the closest obstacle; the gradient points in a direction away from the obstacle along the ray.

provide is fairly accurate in depth, but not in azimuth. The beam pattern can be approximated with a cone (figure 2.19). For the commonly used Polaroid transducer, the arcbase is 22.5 degrees. When the reading of the sensor is d, the point of reflection can be anywhere along the arc base of length $\frac{2\pi d 22.5}{360}$.

Initially, assume that the echo originates from the center of the sonar cone. We acknowledge that this is a naive model, hence we term this the *centerline model* (figure 2.19). The ultrasonic sensor with the smallest reading approximates the global minimum of the raw distance function, and hence $D(x)$. The direction that this sensor is facing approximates the negated gradient $-\nabla D(x)$ because this sensor faces the closest obstacle. The tangent is then the line orthogonal to the direction associated with the smallest sensor reading.

2.3.3 How to Process Sensor Information: Continuation Methods

The tangent to the offset curve is $(\nabla D(x))^{\perp}$, the line orthogonal to $\nabla D(x)$ (figure 2.14). The vector $\nabla D(x)$ points in the direction that maximally increases distance; likewise, the vector $-\nabla D(x)$ points in the direction that

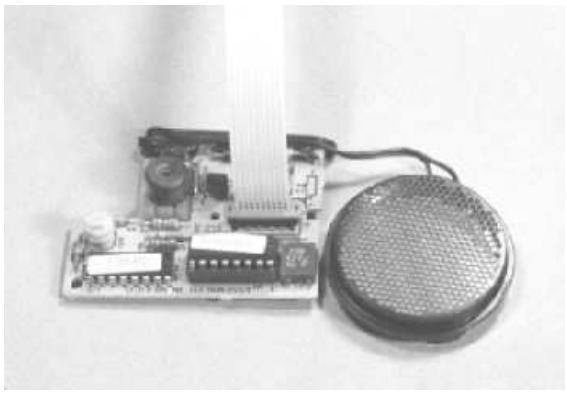

FIGURE 2.17. The disk on the right is the standard Polaroid ultrasonic transducer found on many mobile robots; the circuitry on the left drives the transducer.

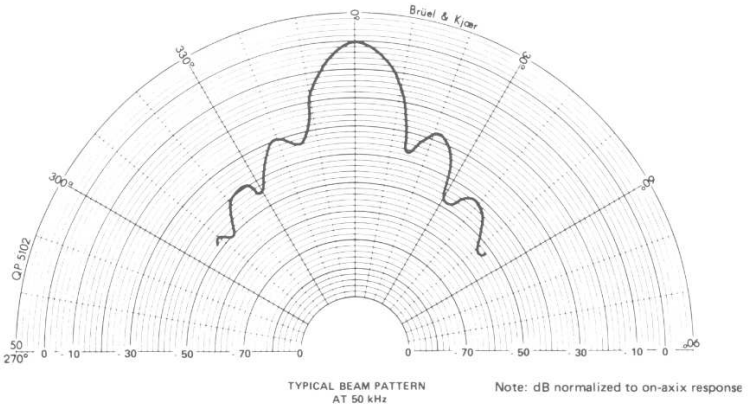

FIGURE 2.18. Beam pattern for the Polaroid transducer.

maximally decreases distance; they both point along the same line, but in opposite directions. Therefore, the vector $(\nabla D(x))^{\perp}$ points in the direction that locally maintains distance; it is perpendicular to both $\nabla D(x)$ and $-\nabla D(x)$. This would be the tangent of the offset curve which maintains distance to the nearby obstacle.

Another way to see why $(\nabla D(x))^{\perp}$ is the tangent is to look at the def-

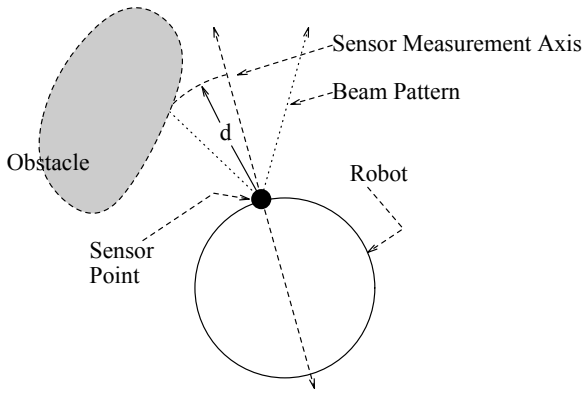

FIGURE 2.19. Centerline model.

inition of the offset curve. For a safety distance $\mathcal{W}^*$, we can define the offset curve implicitly as the set of points where $G(x) = D(x) - \mathcal{W}^*$ maps to zero. The set of nonzero points (or vectors) that map to zero is called the *null space* of a map. For a curve implicitly defined by G, the tangent space at a point x is the null space of $DG(x)$, the Jacobian of G [388]. In general, the i, jth component of the Jacobian matrix is the partial derivative of the ith component function with respect to the jth coordinate and thus the Jacobian is a mapping between tangent spaces. Since in this case, G is a real-valued function (i = 1), the Jacobian is just a row vector $DD(x)$. Here, we are reusing the symbol D. The reader is forced to use context to determine if D means distance or differential.

In Euclidean spaces, the ith component of a single-row Jacobian equals the ith component of the gradient and thus $\nabla D(x) = (DD(x))^T$. Therefore, since the tangent space is the null space of $DD(x)$, the tangent for boundary-following in the plane is the line orthogonal to $\nabla D(x)$, i.e., $(\nabla D(x))^\perp$, and can be derived from sensor information.

Using distance information, the robot can determine the tangent direction to the offset curve. If the obstacles are flat, then the offset curve is also flat, and simply following the tangent is sufficient to follow the boundary of an unknown obstacle. Consider, instead, an obstacle with curvature. We can, however, assume that the obstacle is locally flat. The robot can then move along the tangent for a short distance, but since the obstacle has curvature, the robot will not follow the offset curve, i.e., it will "fall off" of the offset curve. To reaccess the offset curve, the robot moves either toward or away from the obstacle until it reaches the safety distance $\mathcal{W}^*$. In doing so,

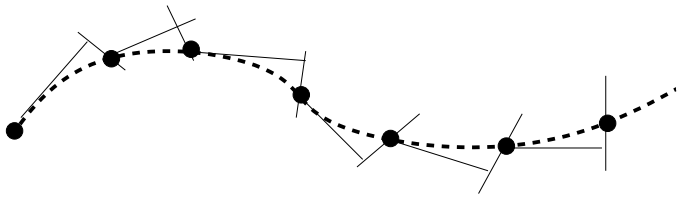

FIGURE 2.20. The dashed line is the actual path, but the robot follows the thin black lines, predicting and correcting along the path. The black circles are samples along the path.

the robot is moving along a line defined by $\nabla D(x)$, which can be derived from sensor information.

Essentially, the robot is performing a numerical procedure of prediction and correction. The robot uses the tangent to locally predict the shape of the offset curve and then invokes a correction procedure once the tangent approximation is not valid. Note that the robot does not explicitly trace the path but instead "hovers" around it, resulting in a sampling of the path, not the path itself (figure 2.20).

A numerical tracing procedure can be posed as one which traces the roots of the expression $G(x) = 0$, where in this case $G(x) = D(x) - \mathcal{W}^*$. Numerical curve-tracing techniques rest on the *implicit function theorem* [6, 222, 294] which locally defines a curve that is implicitly defined by a map $G\colon Y \times \mathbb{R} \to Y$. Specifically, the roots of G locally define a curve parameterized by $\lambda \in \mathbb{R}$. See appendix **??** for a formal definition.

For boundary following at a safety distance $\mathcal{W}^*$, the function $G(y, \lambda) = D(y, \lambda) - \mathcal{W}^*$ implicitly defines the offset curve. Note that the λ-coordinate corresponds to a tangent direction and the y-coordinates to the line or hyperplane orthogonal to the tangent. Let Y denote this hyperplane and $D_Y G$ be the matrix formed by taking the derivative of $G(x) = D(x) - \mathcal{W}^* = 0$ with respect to the y-coordinates. It takes the form $D_Y G(x) = D_Y D(x)$ where D_Y denotes the gradient with respect to the y-coordinates. If $D_Y G(y, \lambda)$ is surjective at $x = (\lambda, y)^T$, then the implicit function theorem states that the roots of $G(y, \lambda)$ locally define a curve that follows the boundary at a distance $\mathcal{W}^*$ as λ is varied, i.e., $y(\lambda)$.

By numerically tracing the roots of G, we can locally construct a path. While there are a number of curve tracing techniques [222], let us consider an adaptation of a common predictor-corrector scheme. Assume that the

robot is located at a point x which is a fixed distance $\mathcal{W}^*$ away from the boundary. The robot takes a "small" step, $\Delta\lambda$, in the λ-direction (i.e., the tangent to the local path). In general, this *prediction step* takes the robot off the offset path. Next, a *correction method* is used to bring the robot back onto the offset path. If $\Delta\lambda$ is small, then the local path will intersect a *correcting plane,* which is a plane orthogonal to the λ-direction at a distance $\Delta\lambda$ away from the origin.

The correction step finds the location where the offset path intersects the correcting plane and is an application of the Newton convergence theorem [222]. See appendix **??** for a more formal definition of this theorem. The Newton convergence theorem also requires that $D_Y G(y, \lambda)$ be full rank at every (y, λ) in a neighborhood of the offset path. This is true because for $G(x) = D(x) - \mathcal{W}^*$, $[0 \quad D_Y G(y, \lambda)]^T = DG(y, \lambda)$. Since $DG(y, \lambda)$ is full rank, so must be $D_Y G(y, \lambda)$ on the offset curve. Since the set of nonsingular matrices is an open set, we know there is a neighborhood around each (y, λ) in the offset path where $DG(y, \lambda)$ is full rank and hence we can use the iterative Newton method to implement the corrector step. If y^h and λ^h are the hth estimates of y and λ, the $h + 1$st iteration is defined as

$$y^{h+1} = y^h - (D_Y G)^{-1} G(y^h, \lambda^h), \tag{2.5}$$

where $D_Y G$ is evaluated at (y^h, λ^h). Note that since we are working in a Euclidean space, we can determine $D_Y G$ solely from distance gradient, and hence, sensor information.

Problems

1. Prove that $D(x)$ is the global minimum of $\rho(x, s)$ with respect to s.
2. What are the tradeoffs between the Bug1 and Bug2 algorithms?
3. Extend the Bug1 and Bug2 algorithms to a two-link manipulator.
4. What is the difference between the Tangent Bug algorithm with zero range detector and Bug2? Draw examples.
5. What are the differences between the path in figure 2.10 and the paths that Bug1 and Bug2 would have generated?
6. The Bug algorithms also assume the planner knows the location of the goal and the robot has perfect positioning. Redesign one of the Bug algorithms to relax the assumption of perfect positioning. Feel free

to introduce a new type of "reasonable" sensor (not a high-resolution Global Positioning System).

7. In the Bug1 algorithm, prove or disprove that the robot does not encounter any obstacle that does not intersect the disk of radius $d(q_{\text{start}}, q_{\text{goal}})$ centered at q_{goal}.

8. What assumptions do the Bug1, Bug2, and Tangent Bug algorithms make on robot localization, both in position and orientation?

9. Prove the completeness of the Tangent Bug algorithm.

10. Adapt the Tangent Bug algorithm so that it has a limited field of view sensor, i.e., it does not have a 360 degree field of view range sensor.

11. Write out $D_Y G$ for boundary following in the planar case.

12. Let $G_1(x) = D(x) + 1$ and let $G_2(x) = D(x) + 2$. Why are their Jacobians the same?

13. Let $G(x, y) = y^3 + y - x^2$. Write out a y as a function of x in an interval about the origin for the curve defined by $G(x, y) = 0$.

Chapter 3

Configuration Space

- Page 56, in the paragraph

  As an example, consider the one-dimensional manifold $S^1 = \{x = (x_1, x_2) \in \mathbb{R}^2 \mid x_1^2 + x_2^2 = 1\}$. For any point $x \in S^1$ we can define a neighborhood that is diffeomorphic to $\mathbb{R}$. For example, consider the upper portion of the circle, $U_1 = \{x \in S^1 \mid x_2 > 0\}$. The chart $\phi_1 : U_1 \to (0,1)$ is given by $\phi_1(x) = x_1$, and thus x_1 can be used to define a local coordinate system for the upper semicircle. In the other direction, the upper portion of the circle can be parameterized by $z \in (0,1) \subset \mathbb{R}$, with $\phi_1^{-1}(z) = (z, (1-z)^{\frac{1}{2}})$, which maps the open unit interval to the upper semicircle. But S^1 is not globally diffeomorphic to $\mathbb{R}^1$; we cannot find a single chart whose domain includes all of S^1.

  - All of the $(0,1)$'s should be $(-1,1)$.
  - The $\phi_1^{-1}(z) = (z, (1-z)^{\frac{1}{2}})$ should read $\phi_1^{-1}(z) = (z, (1-z^2)^{\frac{1}{2}})$

- Page 57, the

$$\begin{aligned} \phi_1^{-1}(z) &= (z, 1-z^2) \\ \phi_2^{-1}(z) &= (z, z^2-1) \\ \phi_3^{-1}(z) &= (1-z^2, z) \\ \phi_4^{-1}(z) &= (z^2-1, z). \end{aligned}$$

should read

$$\begin{aligned}
\phi_1^{-1}(z) &= (z, (1-z^2)^{\frac{1}{2}}) \\
\phi_2^{-1}(z) &= (z, (z^2-1)^{\frac{1}{2}}) \\
\phi_3^{-1}(z) &= ((1-z^2)^{\frac{1}{2}}, z) \\
\phi_4^{-1}(z) &= ((z^2-1)^{\frac{1}{2}}, z).
\end{aligned}$$

- Page 58, the $1 - z^2$ should read $(1 - z^2)^{\frac{1}{2}}$
- Page 71, in the Jacobian matrix, element $(2, 3)$ lower right-hand corner, the element should not be $\frac{d\phi_3}{dq_2}$, but instead it should be $\frac{d\phi_3}{dq_2}$, so the Jacobian at the end of the chapter (before the problems) which currently reads

$$J(q) = \frac{\partial \phi}{\partial q} = \begin{bmatrix} \frac{\partial \phi_1}{\partial q_1} & \frac{\partial \phi_1}{\partial q_2} & \frac{\partial \phi_1}{\partial q_3} \\ \frac{\partial \phi_2}{\partial q_1} & \frac{\partial \phi_2}{\partial q_2} & \frac{\partial \phi_3}{\partial q_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -r_1 \sin q_3 - r_2 \cos q_3 \\ 0 & 1 & r_1 \cos q_3 - r_2 \sin q_3 \end{bmatrix}.$$

should read

$$J(q) = \frac{\partial \phi}{\partial q} = \begin{bmatrix} \frac{\partial \phi_1}{\partial q_1} & \frac{\partial \phi_1}{\partial q_2} & \frac{\partial \phi_1}{\partial q_3} \\ \frac{\partial \phi_2}{\partial q_1} & \frac{\partial \phi_2}{\partial q_2} & \frac{\partial \phi_2}{\partial q_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -r_1 \sin q_3 - r_2 \cos q_3 \\ 0 & 1 & r_1 \cos q_3 - r_2 \sin q_3 \end{bmatrix}.$$

Chapter 4

Potential Functions

- On Pg 78, the Hessian matrix has the indices in the lower left corner incorrect. The $\frac{\partial^2 U}{\partial q_1 \partial q_n}$ should read $\frac{\partial^2 U}{\partial q_n \partial q_1}$, so the matrix

$$\begin{bmatrix} \frac{\partial^2 U}{\partial q_1^2} & \ddots & \frac{\partial^2 U}{\partial q_1 \partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 U}{\partial q_1 \partial q_n} & \cdots & \frac{\partial^2 U}{\partial q_n^2} \end{bmatrix}.$$

  should read

$$\begin{bmatrix} \frac{\partial^2 U}{\partial q_1^2} & \ddots & \frac{\partial^2 U}{\partial q_1 \partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 U}{\partial q_n \partial q_1} & \cdots & \frac{\partial^2 U}{\partial q_n^2} \end{bmatrix}.$$

- On page 103 eq. 4.26(there are two equations) are wrong. The u's should be upper case and the double summation notation should be consistent.

$$\begin{aligned} u(q) &= \sum_j u_{\text{att}j} + \sum_{ij} u_{\text{rep}ij} \\ &= \sum_j J_j^T(q) \nabla u_{\text{att}ij}(q) + \sum_i \sum_j J_j^T(q) \nabla u_{\text{rep}ij} \end{aligned} \tag{4.1}$$

  should read

$$\begin{aligned} U(q) &= \sum_j U_{\text{att}\,j} + \sum_i \sum_j U_{\text{rep}ij} \\ &= \sum_j J_j^T(q) \nabla U_{\text{att}\,j}(q) + \sum_i \sum_j J_j^T(q) \nabla U_{\text{rep}ij} \end{aligned} \tag{4.2}$$

Chapter 5

Roadmaps

- Chap 5, pg 145, an z should be replaced by a q, so

  $$\mathcal{Q}_\lambda = \{x \in \mathcal{Q} | \pi_1(q) = \lambda\}$$

  should read

  $$\mathcal{Q}_\lambda = \{q \in \mathcal{Q} | \pi_1(q) = \lambda\}$$

Chapter 6

Cell Decompositions

Bugs coming soon!

Chapter 7

Sampling-Based Algorithms

Bugs coming soon!

Chapter 8

Kalman Filtering

- Apparently, there is an AppendixJ with no space but I cannot find it again
- Page 276, 7 lines from the bottom, the line $H(k+1)^T K(y(k+1) - H(k+1)x(k+1|k))$. should be replaced with $H(k+1)^T K(y(k+1) - H(k+1)\hat{x}(k+1|k))$.

  There was missing hat on the $x(k+1|k)$.
- Page 277, the $\Delta x = Kv$ below eq. 8.7 should read $\Delta x = H^T Kv$
- Page 285, eq. 8.30 has an extra right parenthesis
- Page 286, second line of eq. 8.34 $\triangleq Gx(k) + w(k)$ should be replaced with $\triangleq Hx(k) + w(k)$
- Page 287, fig 8.5 - bold circle is an ellipse but looks like a circle because the axes were scaled incorrectly.
- Page 291, the phrase

  the process model for this robot nonlinear, i.e.,

  is missing "is".
- Page 297, eq. 8.48 $y(k)_i$ should be replaced with $y_i(k)$.
- on page 293, ninth line from the top $\chi^2_{ij} = \nu_{ij} S^{-1}_{ij} \nu^T_{ij}$ should be replaced with $\chi_{ij} 2 = \nu^T_{ij} S^{-1}_{ij} \nu_{ij}$.

Chapter 9

Bayesian Methods

- on page 308, in the first sentence of Section 9.1.1, $P(x \mid u(0:k-1), y(1:k))$ should have an $x(k)$ not an x so should read
  $P(x(k) \mid u(0:k-1), y(1:k))$
- on page 319, eq. 9.18

$$x^i = x^i + \begin{bmatrix} x_r^i + d' \cos(\theta_r^i + \alpha') \\ y_r^i + d' \sin(\theta_r^i + \alpha') \\ \theta_r^i + \alpha' + \beta' \end{bmatrix}$$

  should be replaced with

$$x^i = x^i + \begin{bmatrix} d' \cos(\theta_r^i + \alpha') \\ d' \sin(\theta_r^i + \alpha') \\ \alpha' + \beta' \end{bmatrix}$$

- on page 325, the h_{ij} below eq. (9.23) should real $h_{i,j}$.

Chapter 10

Robot Dynamics

- page 362, section 10.4.1 Planar Rotation, sentence 3. There is a repeated “to.”

Chapter 11

Trajectory Planning

Bugs coming soon!

Chapter 12

Nonholonomic and Underactuated Systems

Bugs coming soon!

Appendix A

Mathematical Notation

No bugs to report, yet!

Appendix B

Basic Set Definitions

No bugs to report, yet!

Appendix C

Topology and Metric Spaces

No bugs to report, yet!

Appendix D

Mathematical Notation

No bugs to report, yet!

Appendix E

Representations of Orientation

No bugs to report, yet!

Appendix F

Polyhedra Robots in Polyhedra Worlds

No bugs to report, yet!

Appendix G

Analysis of Algorithms and Complexity Classes

No bugs to report, yet!

I believe that there were so many mistakes in the graph search appendix, that we just rewrote the whole thing. This files needs to replace the MIT Press official file.

Appendix H

Graph Representation and Basic Search

H.1 Graphs

Thus far, we have been operating on grids without taking full advantage of the neighboring relationships among the cells. A grid, as well as other maps, can be represented by a *graph* which encodes these neighboring relationships. A graph is a collection of *nodes* and *edges,* i.e., $G = (V, E)$. Sometimes, another term for a node is *vertex,* and this chapter uses the two terms interchangeably. We use G for graph, V for vertex (or node), and E for edge. Typically in motion planning, a node represents a salient location and an edge connects two nodes that correspond to locations that have an important relationship. This relationship could be that the nodes are mutually accessible from each other, two nodes are within line of sight of each other, two cells are next to each other in a grid, etc. This relationship does not have to be mutual: if the robot can traverse from nodes V_1 to V_2, but not from V_2 to V_1, we say that the edge E_{12} connecting V_1 and V_2 is directed. Such a collection of nodes and edges is called a *directed graph*. If the robot can travel from V_1 to V_2 and vica versa, then we connect V_1 and V_2 with two directed edges E_{12} and E_{21}. If for each vertex V_i that is connected to V_j, both E_{ij} and E_{ji} exist, then instead of connecting V_i and V_j with two directed edges, we connect them with a single undirected edge. Such a graph is called an *undirected graph.* Sometimes, edges are annotated with a non-negative numerical value reflective of the costs of traversing this edge. Such values are called *weights*.

A *path* or *walk* in a graph is a sequence of nodes $\{V_i\}$ such that for

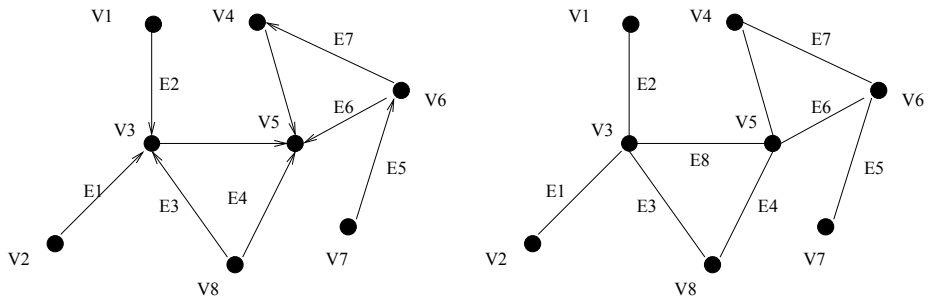

FIGURE H.1. A graph is a collection of nodes and edges. Edges are either directed (left) or undirected (right).

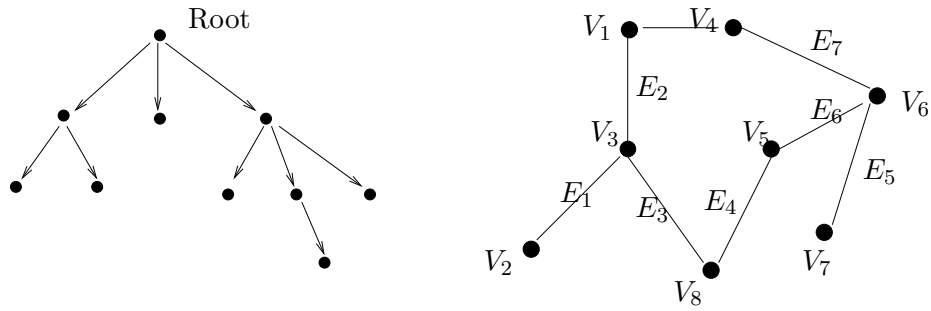

FIGURE H.2. A tree is a type of directed acyclic graph with a special node called the *root*. A cycle in a graph is a path through the graph that starts and ends at the same node.

adjacent nodes V_i and V_{i+1}, $E_{i\ i+1}$ exists (and thus connects V_i and V_{i+1}). A graph is *connected* if for all nodes V_i and V_j in the graph, there exists a path connecting V_i and V_j. A *cycle* is a path of n vertices such that first and last nodes are the same, i.e., $V_1 = V_n$ (figure H.2). Note that the "direction" of the cycle is ambiguous for undirected graphs, which in many situations is sufficient. For example, a graph embedded in the plane can have an undirected cycle which could be both clockwise and counterclockwise, whereas a directed cycle can have one orientation.

A *tree* is a connected directed graph without any cycles (figure H.2). The tree has a special node called the *root*, which is the only node that possesses no incoming arc. Using a parent-child analogy, a parent node has nodes below it called children; the root is a parent node but cannot be a child node. A node with no children is called a *leaf*. The removal of any

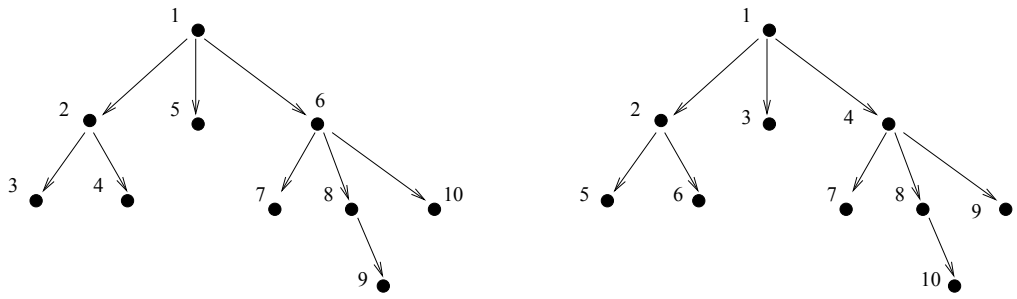

FIGURE H.3. depth-first search vs. breadth-first search. The numbers on each node reflect the order in which nodes are expanded in the search.

non-leaf node breaks the connectivity of the graph.

Typically, one searches a tree for a node with some desired properties such as the goal location for the robot. A *depth-first search* starts at the root, chooses a child, then that node's child, and so on until finding either the desired node or a leaf. If the search encounters a leaf, the search then backs up a level and then searches through an unvisited child until finding the desired node or a leaf, repeating this process until the desired node is found or all nodes are visited in the graph (figure H.3).

Breadth-first search is the opposite; the search starts at the root and then visits all of the children of the root first. Next, the search then visits all of the grandchildren, and so forth. The belief here is that the target node is near the root, so this search would require less time (figure H.3).

A grid induces a graph where each node corresponds to a cell and an edge connects nodes of cells that neighbor each other. Four-point connectivity will only have edges to the north, south, east, and west, whereas eight-point connectivity will have edges to all cells surrounding the current cell. See figure H.4.

As can be seen, the graph that represents the grid is not a tree. However, the breadth-first and depth-first search techniques still apply. Let the *link length* be the number of edges in a path of a graph. Sometimes, this is referred to as edge depth. Link length differs from path length in that the weights of the edges are ignored; only the number of edges count. For a general graph, breadth-first search considers each of the nodes that are the same link length from the start node before going onto child nodes. In contrast, depth-first search considers a child first and then continues

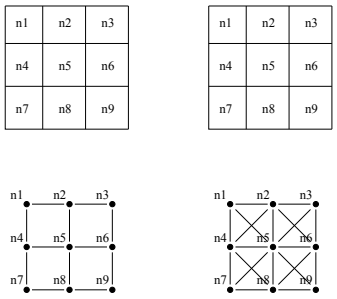

FIGURE H.4. Four-point connectivity assumes only four neighbors, whereas eight-point connectivity has eight.

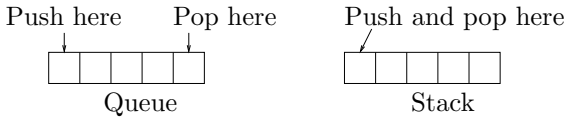

FIGURE H.5. Queue vs. stack.

through the children successively considering nodes of increasing link length away from the start node until it reaches a childless or already visited node (i.e., a cycle). In other words, termination of one iteration of the depth-first search occurs when a node has no unvisited children.

The wave-front planner (chapter **??**, section **??**) is a breadth-first search. Breadth-first search, in general, is implemented with a list where the children of the current node are placed into the list in a first-in, first-out FIFO (manner). This construction is commonly called a *queue* and forces all nodes of the same linklength from the start to be visited first (figure H.5). The breadth-first search starts with placing the start node in the queue. This node is then *expanded* by it being popped off (i.e., removed from the front) the queue and all of its children being placed onto it. This procedure is then repeated until the goal node is found or until there are no more nodes to expand, at which time the queue is empty. Here, we expand all nodes of the same level (i.e., link length for the start) first before expanding more deeply into the graph.

Figure H.6 displays the resulting path of breadth-first search. Note that all paths produced by breadth-first search in a grid with eight-point connectivity are optimal with respect to the the "eight-point connectivity metric." Figure H.7 displays the link lengths for all shortest paths between each cell

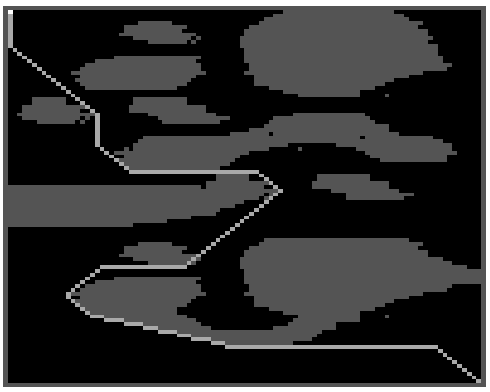

FIGURE H.6. White cells denote the path that was determined with breadth-first search.

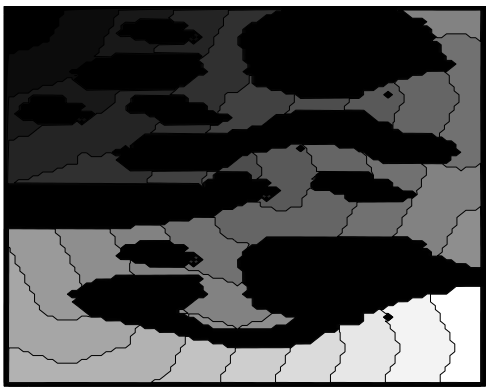

FIGURE H.7. A plot of link length values from the start (upper - left corner) node where colored cells correspond to link length (where the lighter the cell the greater the link length in the graph) and black cells correspond to obstacles.

and the start cell in the free space in Figure H.6. A path can then be determined using this information via a gradient descent of link length from the goal cell to the start through the graph as similarly done with the wavefront algorithm.

Depth-first search contrasts breadth-first search in that nodes are placed in a list in a last-in, first-out LIFO (manner). This construction is commonly

FIGURE H.8. White cells denote the path that was determined with depth-first search.

called a *stack* and forces nodes that are of greater and greater link length from the start node to be visited first. Now the expansion procedure sounds the same but is a little bit different; here, we pop the stack and push all of its children onto the stack, except popping and pushing occur on the same side of the list (figure H.5). Again, this procedure is repeated until the goal node is found or there are no more nodes to expand. Here, we expand nodes in a path as deep as possible before going onto a different path.

Figure H.8 displays the resulting path of depth-first search. In this example, depth-first search did not return an optimal path but it afforded a more efficient search in that the goal was found more quickly than breadth-first search. Figure H.9 is similar to figure H.7, except the link lengths here do *not* correspond to the shortest path to the start; instead, the link lengths correspond to the paths derived by the depth-first search. Again, we can use a depth-first search algorithm to fill up such a map and then determine a path via gradient descent from the goal cell to the start.

Another common search is called a *greedy search* which expands nodes that are closest to the goal. Here, the data structure is called a *priority queue* in that nodes are placed into a sorted list based on a priority value. This priority value is a heuristic that measures distance to the goal node.

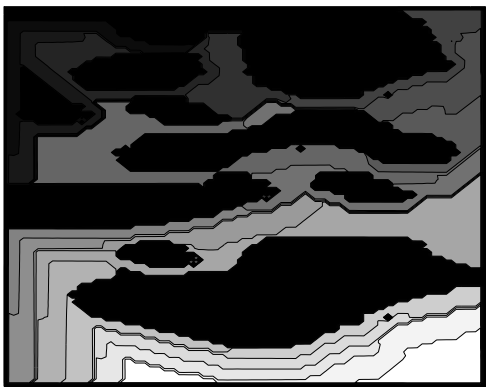

FIGURE H.9. A plot of linklength values from the start (upper - left corner) node where colored cells correspond to link lengths of paths defined by the depth-first search. The lighter the cell the greater the linklengths in the graph; black cells correspond to obstacles.

H.2 A^* Algorithm

Breadth-first search produces the shortest path to the start node in terms of link lengths. Since the wave-front planner is a breadth-first search, a four-point connectivity wave-front algorithm produces the shortest path with respect to the Manhattan distance function. This is because it implicitly has an underlying graph where each node corresponds to a cell and neighboring cells have an edge length of one. However, shortest-path length is not the only metric we may want to optimize. We can tune our graph search to find optimal paths with respect to other metrics such as energy, time, traversability, safety, etc., as well as combinations of them.

When speaking of graph search, there is another opportunity for optimization: minimize the number of nodes that have to be visited to locate the goal node subject to our path-optimality criteria. To distinguish between these forms of optimality, let us reserve the term *optimality* to measure the path and *efficiency* to measure the search, i.e., the number of nodes visited to determine the path. There is no reason to expect depth-first and breadth-first search to be efficient, even though breadth-first search can produce an optimal path.

Depth-first and breadth-first search in a sense are uninformed, in that the search just moves through the graph without any preference for or influence

on where the goal node is located. For example, if the coordinates of the goal node are known, then a graph search can use this information to help decide which nodes in the graph to visit (i.e., expand) to locate the goal node.

Alas, although we may have some information about the goal node, the best we can do is define a *heuristic* which hypothesizes an expected, but not necessarily actual, cost to the goal node. For example, a graph search may choose as its next node to explore one that has the shortest Euclidean distance to the goal because such a node has highest possibility, based on local information, of getting closest to the goal. However, there is no guarantee that this node will lead to the (globally) shortest path in the graph to the goal. This is just a good guess. However, these good guesses are based on the best information available to the search.

The A^* algorithm searches a graph efficiently, with respect to a chosen heuristic. If the heuristic is "good," then the search is efficient; if the heuristic is "bad," although a path will be found, its search will take more time than probably required and possibly return a suboptimal path. A^* will produce an optimal path if its heuristic is *optimistic*. An optimistic, or admissible, heuristic always returns a value less than or equal to the cost of the shortest path from the current node to the goal node within the graph. For example, if a graph represented a grid, an optimistic heuristic could be the Euclidean distance to the goal because the L^2 distance is always less than or equal to the L^1 distance in the plane (figure H.10).

First, we will explain the A^* search via example and then formally introduce the algorithm. See figure H.11 for a sample graph. The A^* search has a *priority queue* which contains a list of nodes sorted by priority. This priority is determined by the sum of the distance from the start node to the current node and the heuristic at the current node.

The first node to be put into the priority queue is naturally the start node. Next, we *expand* the start node by popping the start node and putting all adjacent nodes to the start node into the priority queue sorted by their corresponding priorities. Since node B has the highest priority, it is expanded next, i.e., it is popped from the queue and its neighbors are added (figure H.12). Note that only unvisited nodes are added to the priority queue, i.e., do not re-add the start node.

Now, we expand node H because it has the highest priority. It is popped off of the queue and all of its neighbors are added. However, H has no neighbors, so nothing is added to the queue. Since no new nodes are added, no more action or expansion will be associated with node H (figure H.12). Next, we pop off the node with highest priority, i.e., node A, and expand it,

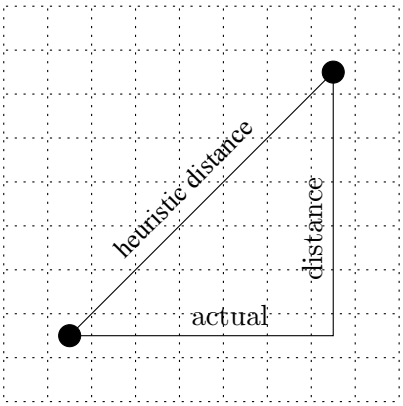

FIGURE H.10. The heuristic between two nodes is the Euclidean distance, which is less than the actual path length in the grid, making this heuristic optimistic.

adding all of its adjacent neighbors to the priority queue (figure H.12).

Next, node E is expanded which gives us a path to the goal of cost 5. Note that this cost is the real cost, i.e., the sum of the edge costs to the goal. At this point, there are nodes in the priority queue which have a priority value greater than the cost to the goal. Since these priority values are lower bounds on path cost to the goal, all paths through these nodes will have a higher cost than the cost of the path already found. Therefore, these nodes can be discarded (figure H.12).

The explicit path through the graph is represented by a series of *back pointers.* A back pointer represents the immediate history of the expansion process. So, the back pointers from nodes A, B, and C all point to the start. Likewise, the back pointers to D, E, and F point to A. Finally, the back pointer of goal points to E. Therefore, the path defined with the back pointers is start, A, E, and goal. The arrows in figure H.12 point in the reverse direction of the back pointers.

Even though a path to the goal has been determined, A^* is not finished because there could be a better path. A^* knows this is possible because

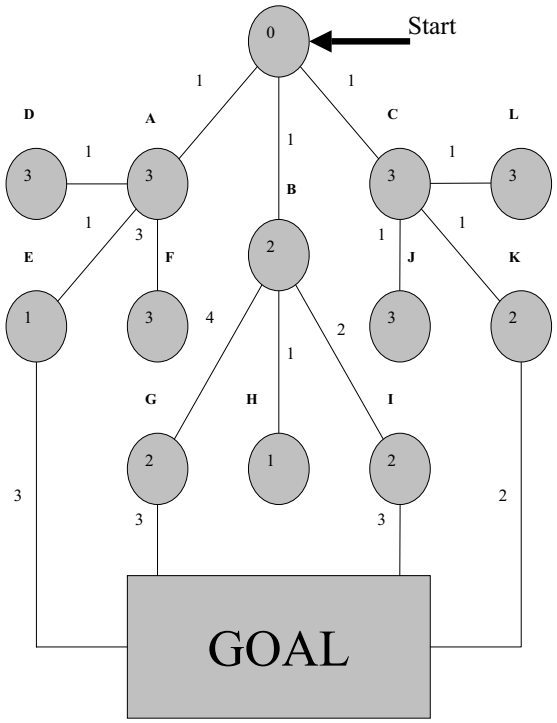

FIGURE H.11. Sample graph where each node is labeled by a letter and has an associated heuristic value which is contained inside the node icon. Edge costs are represented by numbers adjacent to the edges and the start and goal nodes are labeled. We label the start node with a zero to emphasize that it has the highest priority at first.

the priority queue still contains nodes whose values are smaller than that of the goal state. The priority queue at this point just contains node C and is then expanded adding nodes J, K, and L to the priority queue. We can immediately remove J and L because their priority values are greater than or equal the cost of the shortest path found thus far. Node K is then expanded finding the goal with a path cost shorter than the previously found path through node E. This path becomes the current best path. Since at this point the priority queue does not possess any elements whose values are smaller than that of the goal node, this path results in the best path

FIGURE H.12. (Left) Priority queue after the start is expanded. (Middle) Priority queue after the second node, B, is expanded. (Right) Three iterations of the priority queue are displayed. Each arrow points from the expanded node to the nodes that were added in each step. Since node H had no unvisited adjacent cells, its arrow points to nothing. The middle queue corresponds to two actions. Node E points to the goal which provides the first candidate path to the goal. Note that nodes D, I, F, and G are shaded out because they were discarded.

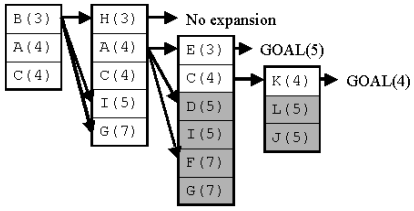

FIGURE H.13. Four displayed iterations of the priority queue with arrows representing the history of individual expansions. Here, the path to the goal is start, C, K, goal.

(figure H.13).

H.2.1 Basic Notation and Assumptions

Now, we can more formally define the A^* algorithm. The input for A^* is the graph itself. These nodes can naturally be embedded into the robot's free space and thus can have coordinates. Edges correspond to adjacent nodes and have values corresponding to the cost required to traverse between the adjacent nodes. The output of the A^* algorithm is a back-pointer path, which is a sequence of nodes starting from the goal and going back to the

start.

We will use two additional data structures, an open set O and a closed set C. The open set O is the priority queue and the closed set C contains all processed nodes. Other notation includes

- Star(n) represents the set of nodes which are adjacent to n.
- $c(n_1, n_2)$ is the length of edge connecting n_1 and n_2.
- $g(n)$ is the total length of a backpointer path from n to q_{start}.
- $h(n)$ is the heuristic cost function, which returns the estimated cost of shortest path from n to q_{goal}.
- $f(n) = g(n) + h(n)$ is the estimated cost of shortest path from q_{start} to q_{goal} via n.

The algorithm can be found in algorithm 4.

Algorithm 4 A^* Algorithm

Input: A graph
Output: A path between start and goal nodes

1: **repeat**
2: Pick n_{best} from O such that $f(n_{best}) \leq f(n), \forall n \in O$.
3: Remove n_{best} from O and add to C.
4: If $n_{best} = q_{goal}$, EXIT.
5: Expand n_{best}: for all $x \in$ Star(n_{best}) that are not in C.
6: **if** $x \notin O$ **then**
7: add x to O.
8: **else if** $g(n_{best}) + c(n_{best}, x) < g(x)$ **then**
9: update x's backpointer to point to n_{best}
10: **end if**
11: **until** O is empty

The A^* algorithm searches for a path from the start to the goal. In such a case, the g function is sometimes called the *cost-to-come* or *cost-from-start* function. If the search were to occur in reverse, from goal to start, then the g function is called the *cost-to-go* function which measures the path cost to the goal. Likewise, the heuristic then becomes an estimated cost of the shortest path from the current node to the start. The objective function f is still the sum of g and h.

H.2.2 Discussion: Completeness, Efficiency, and Optimality

Here is an informal proof of completeness for A^*. A^* generates a search tree, which by definition, has no cycles. Furthermore, there are a finite number of acyclic paths in the tree, assuming a bounded world. Since A^* uses a tree, it only considers acyclic paths. Since the number of acyclic paths is finite, the most work that can be done, searching all acyclic paths, is also finite. Therefore A^* will always terminate, ensuring completeness.

This is not to say A^* will always search all acyclic paths since it can terminate as soon as it explores all paths with greater cost than the minimum goal cost found. Thanks to the priority queue, A^* explores paths likely to reach the goal quickly first. By doing so, it is efficient. If A^* does search every acyclic path and does not find the goal, the algorithm still terminates and simply returns that a path does not exist. Of course, this also makes sense if every possible path is searched.

Now, there is no guarantee that the first path to the goal found is the cheapest/best path. So, in the quest for optimality (once again, with respect to the defined metric), all branches must be explored to the extent that a branch's terminating node cost (sum of edge costs) is greater than the lowest goal cost. Effectively, all paths with overall cost lower than the goal must be explored to guarantee that an even shorter one does not exist. Therefore, A^* is also optimal (with respect to the chosen metric).

H.2.3 Greedy-Search and Dijkstra's Algorithm

There are variations or special cases of A^*. When $f(n) = h(n)$, then the search becomes a *greedy* search because the search is only considering what it "believes" is the best path to the goal from the current node. When $f(n) = g(n)$, the planner is not using any heuristic information but rather growing a path that is shortest from the start until it encounters the goal. This is a classic search called *Dijkstra's algorithm.* Figure H.14 contains a graph which demonstrates Dijkstra's Algorithm. In this example, we also show backpointers being updated (which can also occur with A^*). The following lists the open and closed sets for the Dijkstra search.

1. $O = \{S\}$
2. $O = \{1, 2, 4, 5\}$; $C = \{S\}$ ($1, 2, 4, 5$ all point back to S)
3. $O = \{1, 4, 5\}$; $C = \{S, 2\}$ (there are no adjacent nodes not in C)

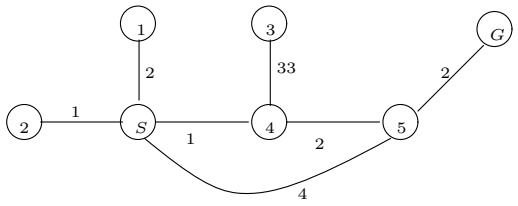

FIGURE H.14. Dijkstra Graph Search Example

4. $O = \{1, 5, 3\}; C = \{S, 2, 4\}$ ($1, 2, 4$ point to S; 5 points to 4; $C = \{S, 2, 4, 1\}$
5. $O = \{3, G\}; C = \{S, 2, 4, 1\}$ (goal points to 5 which points to 4 which points to S)

H.2.4 Example of A^* on a Grid

Figure H.15 contains an example of a grid world with a start and goal identified accordingly. We will assume that the free space uses eight-point connectivity, and thus cell $(3, 2)$ is adjacent to cell $(4, 3)$, i.e., the robot can travel from $(3, 2)$ to $(4, 3)$. Each of the cells also has its heuristic distance to the goal where we use a modified metric which is not the Manhattan or the Euclidean distance. Instead, between free space cells, a vertical or horizontal step has length 1 and a diagonal has length 1.4 (our approximation of $\sqrt{2}$). The cost of traveling from a free space cell to an obstacle cell is made to be arbitrarily high; we chose 10000. So one cell step from a free space to an obstacle cell along a vertical or horizontal direction costs 10000 and one cell step along a diagonal direction costs 10000.4. Here, we are assuming that our graph connects *all* cells in the grid, not just the free space, and the prohibitively high cost of moving into an obstacle will prevent the robot from collision (figure H.16).

Note that this metric, in the free space, does not induce a true Euclidean metric because two cells sideways and one cell up is 2.4, not $\sqrt{5}$. However, this metric is quite representative of path length within the grid. This heuristic is optimistic because the actual cost to current cell to the goal will always be greater than or equal to the heuristic. Thus far, in figure H.15 the back pointers and priorities have not been set.

The start cell is put on the priority queue with a priority equal to its heuristic. See figure H.17. Next, the start node is expanded and the pri-

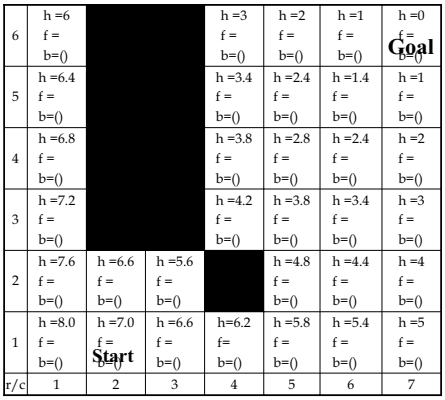

FIGURE H.15. Heuristic values are set, but backpointers and priorities are not.

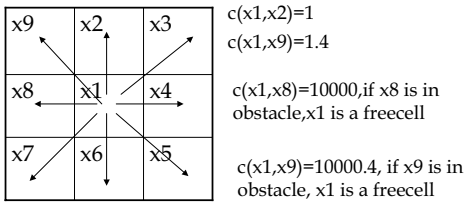

FIGURE H.16. Eight-point connectivity and possible cost values.

ority values for each of the start's neighbors are determined. They are all put on the priority queue sorted in ascending order by priority. See figure H.18(left). Cell $(3, 2)$ is expanded next, as depicted in figure H.18(right). Here, cells $(4, 1)$, $(4, 2)$, $(4, 3)$, $(3, 3)$, and $(2, 3)$ are added onto the priority queue because our graph representation of the grid includes both free space and obstacle cells. However, cells $(4, 2)$, $(3, 3)$, and $(2, 3)$ correspond to obstacles and thus have a high cost. If a path exists in the free space or the

longest path in the free space has a traversal cost less than our arbitrarily high number chosen for obstacles (figure H.16), then these cells will never be expanded. Therefore, in the figures below, we did not display them on the priority queue.

Eventually, the goal cell is reached (figure H.19 (left)). Since the priority value of the goal is less than the priorities of all other cells in the priority queue, the resulting path is optimal and A^* terminates. A^* traces the backpointers to find the optimal path from start to goal (figure H.19 (right)).

H.2.5 Nonoptimistic Example

Figure H.20 contains an example of a graph whose heuristic values are nonoptimistic and thus force A^* to produce a nonoptimal path. A^* puts node S on the priority queue and then expands it. Next, A^* expands node A because its priority value is 7. The goal node is then reached with priority value 8, which is still less than node B's priority value of 13. At this point, node B will be eliminated from the priority queue because its value is greater than the goal's priority value. However, the optimal path passes through B, not A. Here, the heuristic is not optimistic because from B to G, $h = 10$ when the actual edge length was 2.

H.3 D^* Algorithm

So far we have only considered *static* environments where only the robot experiences motion. However, we can see that many worlds have moving obstacles, which could be other robots themselves. We term such environments *dynamic*. We can address dynamic environments by initially invoking the A^* algorithm to determine a path from start to goal, follow that path until an unexpected change occurs (see $(4, 3)$ in figure H.21(left)) and then reinvoke the A^* algorithm to determine a new path. This, however, can become quite inefficient if many cells are changing from obstacle to free space and vice versa. The D^* algorithm was devised to "locally repair" the graph allowing for an efficient updated searching in dynamic environments, hence the term D^* [376].

The D^* algorithm (algorithm 5) uses the notation found in table H.1. Just like A^*, D^* has a data structure called the open list O. However, D^* may process states after they are removed from the open list, so $t(X)$ is used to classify states as NEW, $OPEN$, and $CLOSED$ to mean X has

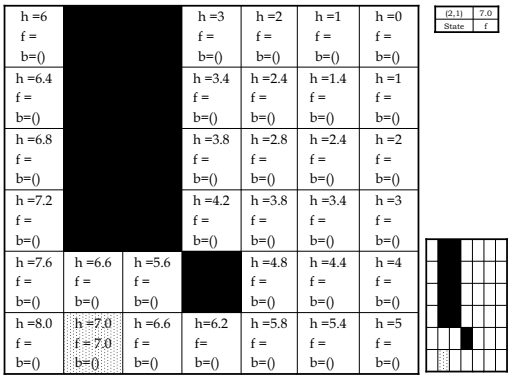

FIGURE H.17. Start node is put on priority queue, displayed in upper right.

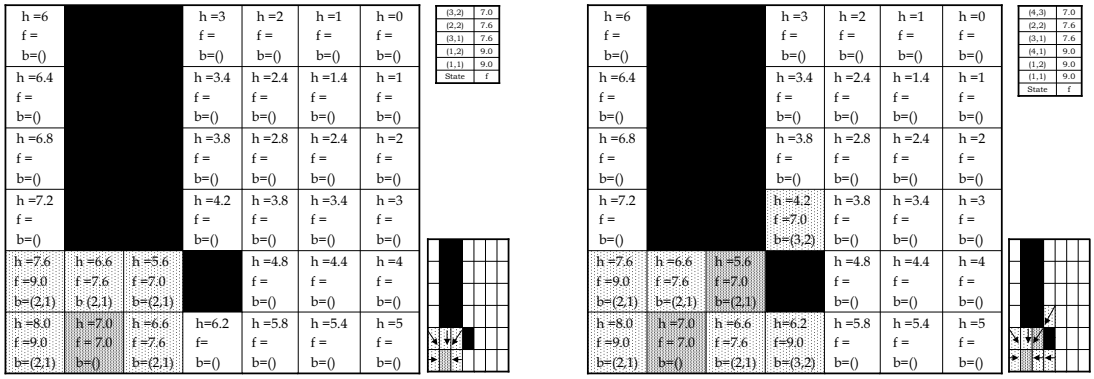

FIGURE H.18. (Left) The start node is expanded, the priority queue is updated, and the backpointers are set, which are represented by the right bottom icon. $b = (i, j)$ points to cell (i, j). (Right) Cell $(3, 2)$ was expanded. Note that cells $(3, 3)$, $(2, 3)$, and $(4, 2)$ are not displayed in the priority queue because they correspond to obstacles.

never been in O, X is currently in 0, and X was in O but currently is not, respectively.

The function $h(X)$ measures path cost from the goal to X; this is some-

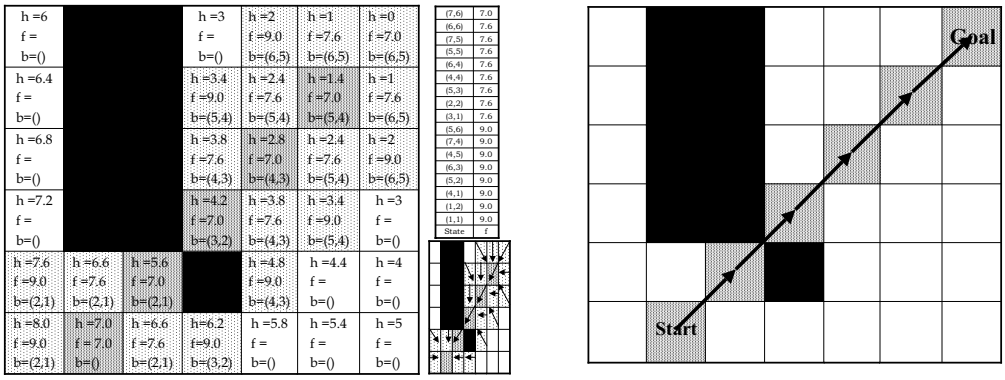

FIGURE H.19. (Left) The goal state is expanded. (Right) Resulting path.

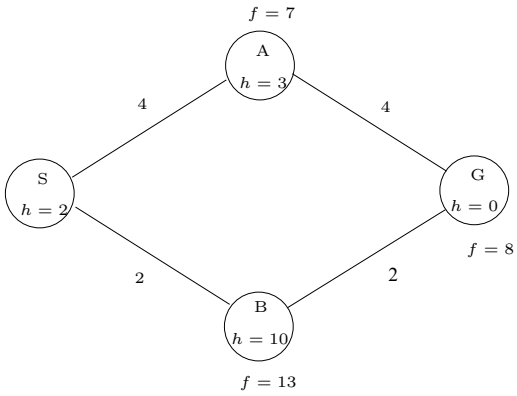

FIGURE H.20. A nonoptimistic heuristic leads to a nonoptimal path with A^*.

times called cost-to-go. Ultimately, the D^* planner uses h to determine the path in the graph; when sequencing through the path to the goal, the values of h decrease. Note that some notation may get confusing here because A^* uses h as the heuristic function whereas D^* uses h as the cost-to-go function.

The D^* algorithm uses the k values to determine the priority of the nodes in the open list. The k value for a particular node X is the smallest h value for that node since it was most recently inserted onto the open list.

State		Functions	
State	X	Stored cost from Y to X	$c(X,Y)$
Current State	X_c	Cost from Y to X based on sensors	$r(X,Y)$
Neighbor State	X_n	Cost from X to goal	$h(X)$
Start	S	Minimum cost from X to goal	$k(X)$
Goal	G	Tag	$t(X)$
List of all States	L	Parent state (predecessor) of X	$b(X)$
Open List	O		

TABLE H.1. Common notation used in D^*

This means if X were inserted into the open list, if its h value changes, then the k value is the smallest of the h values while X remains in the open list. However, once X is removed from the open list, if it is re-inserted, $k(X)$ starts off with the new h value.

The significance of $k(X)$ is that it distinguishes *raise* states, those where $k(X) > h(X)$ from *lower* states, those where $k(X) = h(X)$. It is possible for obstacle nodes to be placed on the open list, but they will have high k values meaning they will probably never be processed. After the initial search, the k-values are identical to the h values.

We once again consider a directed graph where a node corresponds to a robot state and an edge connects the nodes of adjacent states. Note that two states may not physically lie side-by-side, so an edge really corresponds to two nodes for which a robot can traverse from one state X to another state Y for some edge cost $c(Y,X)$. It is possible that $c(X,Y) \neq c(Y,X)$.

After some initialization, D^* performs a modified Dijkstra's search using $INIT-PLAN$ (algorithm 6), which calls $PROCESS-STATE$ (algorithm 14) to expand nodes. The initial search starts by placing the goal node onto the open list and ends when the start node is labeled as closed. Once the search is complete, the robot begins to follow the optimal path to the goal. This happens in $PREPARE-REPAIR$ (algorithm 7) and $REPAIR-REPLAN$ (algorithm 8). Effectively, $PREPARE-REPAIR$ looks for changes in the environment that *affects* the costs among nodes within sensor range and if such a changes exist, $REPAIR-REPLAN$ locally repairs the graph by readjusting the backpointers, so a new optimal path can be found. If no change exists, $REPAIR-REPLAN$ simply directs the robot along the current optimal path.

The actual repairing of the backpointers and directed motion of the

Algorithm 5 D^* Algorithm

Input: List of all states L

Output: The goal state, if it is reachable, and the list of states L are updated so that the backpointer list describes a path from the start to the goal. If the goal state is not reachable, return NULL.

1: **for each** $X \in L$ **do**
2: t(X) = NEW
3: **end for**
4: $h(G) = 0$
5: $INSERT(O, G, h(G))$
6: $X_c = S$
7: $P = INIT - PLAN(O, L, X_c, G)$ (algorithm 6)
8: **if** P = NULL **then**
9: Return (NULL)
10: **end if**
11: **while** $X_c \neq G$ **do**
12: $PREPARE - REPAIR(O, L, X_c)$ (algorithm 7)
13: $P = REPAIR - REPLAN(O, L, X_c, G)$ (algorithm 8)
14: **if** P = NULL **then**
15: Return (NULL)
16: **end if**
17: X_c = the second element of P {Move to the next state in P}.
18: **end while**
19: Return (X_c)

Algorithm 6 $INIT - PLAN(O, L, X_c, G)$

Input: Open list O, List of all states L, Current Position X_c, Goal G

Output: A list of states to goal as described by back pointers in the list of states L; Open List O is modified

1: **repeat**
2: $k_{min} = PROCESS - STATE(O, L)$
3: **until** ($k_{min} = -1$) or ($t(X_c) = CLOSED$)
4: $P = GET - BACKPOINTER - LIST(L, X_c, G)$ (algorithm 9)
5: Return (P)

robot occurs in $REPAIR - REPLAN$. Notice that $INIT - PLAN$ and $REPAIR - REPLAN$ look quite similar except the "until" terminating condition is different. This is because $INIT - PLAN$ only accesses one

part of $PROCESS - STATE$ since $t(X) = NEW$ whereas $REPAIR - REPLAN$ uses all parts of $PROCESS - STATE$. The repairing process terminates when k value of any node in the open list is greater than or equal to $h(X_c)$. In other words, the process terminates when the minimum path cost from any node in the open list to the goal is greater than or equal to the the path cost from the current robot position X_c to the goal. This is the terminating condition in $REPAIR - REPLAN$ which is different from the terminating condition in $INIT - PLAN$.

There is one piece of terminology which should be made clear at this point. In actuality, D^* is not considering *actual* path costs but rather the *perceived* path costs which are derived from the robot's current understanding of the graph being searched. If the edges of the graph are incorrect, say due to a change in the environment, then the D^* algorithm does not use the updated information until the robot discovers a change in the environment, i.e., until the robot discovers a change in an edge cost. The D^* literature hence uses the term *estimated* path cost to reflect this, but such terminology could be confusing because a heuristic also estimates path cost and there are many other ways to estimate path cost. Therefore, this description of D^* avoids the use of this term but the reader should be aware of it when reading the literature. Later on, the focused D^* algorithm was developed to include a heuristic function for guiding, or focusing, the repairing process. Typically for D^*, g is used as the heuristic function switching the g and h notation convention from A^*. Often, when people speak of D^*, they really mean focused D^*, but this section focuses, no pun intended, on the original D^* algorithm.

At this point instead of directly explaining the details of $PROCESS - STATE$, we give an example of the entire D^* algorithm. Consider the grid environment in figure H.21(left) which is identical to the one in figure H.15, except cell $(4, 3)$ is a gate which can either be a free-space cell or an obstacle cell. Assume it starts as a free-space cell. Finally, a node in the search graph corresponds to a cell, regardless if it is obstacle or free space, and since this graph is directed, a pair of nodes corresponding to adjacent cells have two directed edges.

To achieve the initial Dijkstra-like search from the goal back to the start, the goal node is first placed on the open list (figure H.21(right)) with $h = 0$. It is then expanded (figure H.22(left)), adding $(6, 6)$, $(6, 5)$, and $(7, 5)$ onto the queue (algorithm 14, lines 17-19). D^* increments the k and h values according to the metric described in figure H.16. Unless stated otherwise, when a node is expanded, it is automatically put on the closed list, so the goal is expanded and then put on the closed list.

Algorithm 7 $PREPARE - REPAIR(O, L, X_c)$

Input: Open list O, List of all states L, Current Position X_c
Output: Open List O is modified

```
1: for each state X ∈ L within sensor range of X_c and X_c do
2:    for each neighbor Y of X do
3:       if r(Y,X) ≠ c(Y,X) then
4:          MODIFY − COST(O, Y, X, r(Y,X))
5:       end if
6:    end for
7:    for each neighbor Y of X do
8:       if r(X,Y) ≠ c(X,Y) then
9:          MODIFY − COST(O, X, Y, r(X,Y))
10:      end if
11:   end for
12: end for
```

Algorithm 8 $REPAIR - REPLAN(O, L, X_c, G)$

Input: Open list O, List of all states L, Current Position X_c, Goal G
Output: A list of states to goal as described by back pointers in the list of states L; Open List O is modified

```
1: repeat
2:    k_min = PROCESS − STATE(O, L)
3: until (k_min ≥ h(X_c)) or (k_min = −1)
4: P = GET − BACKPOINTER − LIST(L, X_c, G)
5: Return (P)
```

Next, node $(6, 6)$ is expanded adding nodes $(5, 6)$ and $(5, 5)$ onto the open list (figure H.22(right)). The node $(7, 5)$ is then expanded adding nodes $(6, 4)$ and $(7, 4)$ into the open list. (figure H.23(left)). More nodes are expanded until we arrive at node $(4, 6)$ (figure H.23(right)). When $(4, 6)$ is expanded, nodes $(3, 6)$ and $(3, 5)$, which are obstacle nodes, are placed onto the open list, but with high k and h values figure H.23(right). Since the h values of the expanded obstacle nodes are high, they will most likely never be expanded, which makes sense because they are obstacle nodes.

The Dijkstra-like search is complete when the start node $(2, 1)$ is expanded (figure H.24(left)). Note that some cells may not have been considered by the D^* algorithm. The optimal path from start to goal (assuming that the gate cell $(4, 3)$ is not an obstacle) is found by traversing the back-

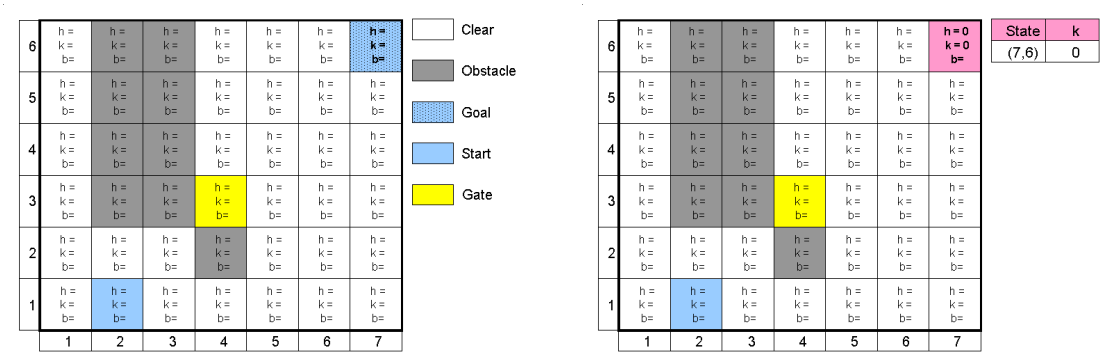

FIGURE H.21. (Left) A cell world similar to figure H.15, except it has a gate, h values and k values. (Right) Put goal node on open list.

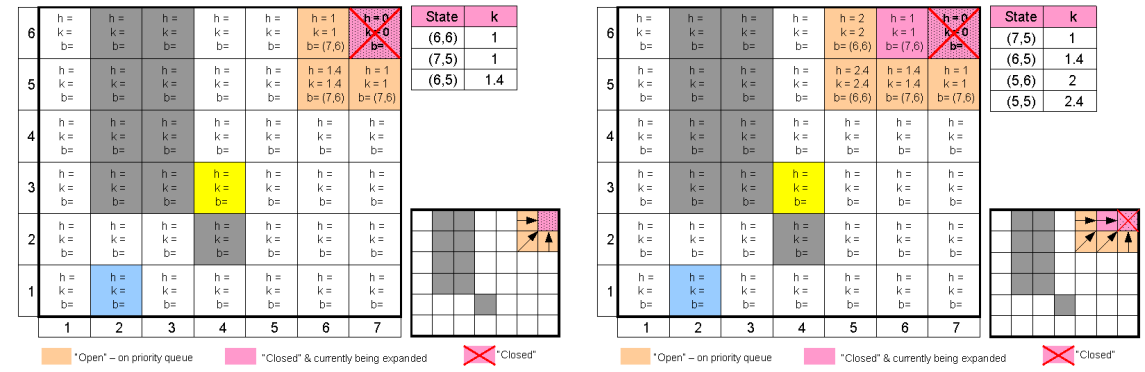

FIGURE H.22. (Left) Expand the Goal Node (Right) Node $(6,6)$ is expanded.

pointers starting from the start node to the goal node (figure H.24(right)). The optimal path is $(2,1) \longrightarrow (3,2) \longrightarrow (4,3) \longrightarrow (5,4) \longrightarrow (6,5) \longrightarrow (7,6)$. Note that nodes $(1,1)$, $(1,2)$, $(1,3)$, $(2,3)$, $(3,3)$, $(3,4)$, $(3,5)$, $(3,6)$ and $(4,2)$ are still on the open list.

The robot then starts tracing the optimal path from the start node to the goal node. In figure H.25(left), the robot moves from node $(2,1)$ to $(3,2)$. When the robot tries to move from node $(3,2)$ to $(4,3)$, it finds that the

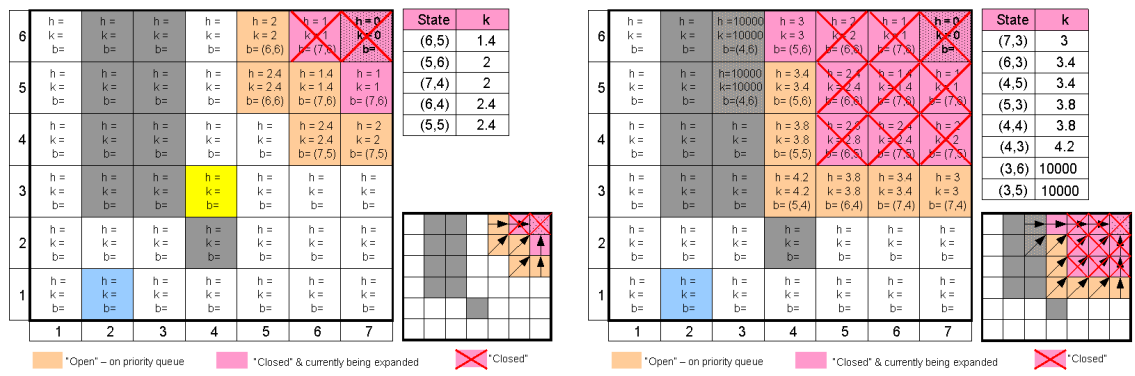

FIGURE H.23. (Left) Expand $(7, 5)$. (Right) Expand $(4, 6)$.

gate node $(4, 3)$ is an obstacle (figure H.25(left)). In the initial search for an optimal path, we had assumed that the gate node was a free space node, and hence the current path is not feasible. At this stage, instead of re-planning for an optimal path from the current node $(3, 2)$ to goal node, D^* tries to make local repairs to the graph until a new optimal path is found.

There is one subtlety about the sensor range which should be noted. When we assume the robot has a sensor range of one cell, does this mean that the robot can see all neighboring cells, the current cell, and the edge costs among all such cells, or does this mean that it can see all such cells and all of the edges costs associated with these cells. If the latter, then some edges may be considered "out of sensor range" because such edge costs are between a cell within sensor range and a cell outside of sensor range. However, it is reasonable to expect such edge costs to change because the cell within sensor range has changed. In this example, we assume that the robot infers that the peripheral edge costs change as well. Either assumption is fine, as long as the implementation is consistent.

To address the fact that $(4, 3)$ is now an obstacle, D^* increases by a large number the transition cost to and from $(4, 3)$ for all nodes adjacent to $(4, 3)$. Actually, in our example, we simply set the transition cost to a high number, say $10,000$. Next, all nodes affected by the increased transition costs (all eight neighbors and $(4, 3)$) are put on the open list (algorithm 7). Recall that D^* is repairing a directed graph, so $MODIFY - COST$ is called 16 times, once for each neighbor and eight times on $(4, 3)$ but $(4, 3)$ is put onto the open list only once. See figure H.25 (right). Note that some neighbors

of (4,3), and (4,3) itself have lower k values than most elements on the open list already. Therefore, these nodes will be popped first.

The node $(5, 4)$ is now popped because its k value is the smallest. Since its k and h are the same, consider each neighbor of $(5, 4)$. One such neighbor is $(4, 3)$. The node $(4, 3)$ has a back pointer which points to $(5, 4)$ but its original h value is no longer the sum of the h value of $(5, 4)$ and the transition cost, which was just raised due to the obstacle (algorithm 14, line 17). Therefore, $(4, 3)$ is put on the open list but with a high h value (algorithm 14, line 19). Note that since $(4, 3)$ is already on the open list, its k value remains the same and hence the node $(4, 3)$ is a raise state because $h > k$. See figure H.26 (left). Next, $(5, 3)$ is popped but this will not affect anything because none of the surrounding nodes are new, and the h values of the surrounding nodes are correct. A similar non-action happens for $(4, 4)$. See figure H.26 (right).

Now, the node $(4, 3)$ is popped off the open list and since $k < h$, the objective is to try to decrease the h value (algorithm 14, line 7). This is akin to finding a better path via $(4, 3)$ to the goal, but this is not possible because $(4, 3)$ is an obstacle. Looking more carefully at the algorithm, consider node $(5, 3)$, which is a neighbor of $(4, 3)$ and has an h value which is less than the k value of $(4, 3)$ but h value of $(4, 3)$ "equals" the sum of the h value of $(5, 3)$ and the transition cost. This means that nothing is improved coming from $(4, 3)$ to $(5, 3)$. This is also true for $(5, 4)$ and $(4, 4)$. See figure H.27 (left). Note that our notion of equality is not precise in that any two "large" numbers are equal; so for example, $10000 = 10001.4$.

So, we cannot find a path through any of the neighbors of $(4, 3)$ to reduce h. Therefore, the node $(4, 3)$ is expanded next, which places all nodes whose back pointers point to $(4, 3)$, which in this case is only $(3, 2)$, onto the open list with a high h value (algorithm 14, line 17). Now, $(3, 2)$ is also a raise state. Note that the k value of $(3, 2)$ is set to the minimum of its old and new h values (this setting happens in the insert function). Next, we pop $(5, 2)$ but this will not affect anything because none of the surrounding nodes are new, and the h values of the surrounding nodes are correct. See figure H.27 (left).

Now, node $(3, 2)$ is popped off the open list. Since $k < h$, D^* looks for a neighbor whose h value is less than the k value of $(3, 2)$ (algorithm 14, line 9). If such a neighbor exists, then D^* would redirect the backpointer through this neighbor. However, no such neighbor exists.

Next, D^* looks for neighboring nodes whose back pointers point to $(3, 2)$ and have an "incorrect" h value, i.e., all neighboring nodes with h values not equal to the h value of $(3, 2)$ plus its associated transition cost. Such

nodes are also placed onto the open list with a high h value, making them raise states (algorithm 14, line 24). These are $(3,1)$, $(2,1)$, and $(2,2)$. Note that the k values of these nodes are set to the minimum of the new h value and the old h value.

Also, D^* looks for neighboring nodes whose back pointer does not point to $(3,2)$, whose h value plus the transition cost is less than the h value of $(3,2)$, which is on the closed list, and whose h value is greater than the k value of $(3,2)$ (algorithm 14, line 29). The only such neighbor is $(4,1)$. This could potentially lead to a lower cost path. So, the neighbor $(4,1)$ is put on the open list with its current h value because it could potentially reduce the h value of $(3,2)$. It is called a lower state because $h = k$. See figure H.27 (right).

Continuing with the lowest k value node, the node $(4,1)$ is popped off the open list and expanded. Since the h and k values of $(4,1)$ are the same, D^* considers the neighbors whose pack pointers do not point to $(4,1)$ to see if passing through $(4,1)$ reduces any of the neighbors h values (algorithm 14, line 17). This redirects the backpointers of $(3,2)$ and $(3,1)$ to pass through $(4,1)$; moreover, these nodes are then put onto the open list. However, since $(3,2)$ was "closed," its new k value is the smaller of its old and new h values, making it a lower state (since $k = h$). Similarly, since $(3,1)$ was "open" (already on the open list), its new k value is the smaller of its old k value and its new h value. See figure H.28 (left).

Next, the node $(3,1)$ is popped off the open list. Since its k value 6.6 is less than its h value 7.2, D^* looks for a neighbor whose h value is less than the k value of $(3,1)$ (algorithm 14, line 9). The only such neighbor is $(4,1)$. This gives us hope that there is a lower cost path through $(4,1)$. However, since the sum of the transition cost to $(4,1)$ and the h value of $(4,1)$ is greater than the h value of $(3,1)$, no such improved path exist and nothing happens. However, the node $(3,1)$ can be used to form a reduced cost path for its neighbors, so $(3,1)$ is put back on the open list but with a k value set to the minimum of its old h value and new h value. Thus, it now also becomes a lower state. See figure H.28 (right).

The node $(2,2)$ is then popped off the open list and expanded. This increases the h values of the nodes that pass through $(2,2)$ and puts them back on the open list. When the nodes $(1,1)$, $(1,2)$ and $(1,3)$ are put back onto the open list, their k values are unaffected, hence their position in the open list remains the same, but their h values are increased making them raise states (algorithm 14, line 24). See figure H.29 (left). Next the node $(2,1)$ is popped off the queue and since $k < h$ and it cannot reduce the cost to any of its neighbors, so this has no effect. See figure H.29 (right).

Now, the node $(3, 1)$ is popped off the open list and expanded. This has the effect of redirecting the back pointers of $(2, 2)$ and $(2, 1)$ through $(3, 1)$ and putting them back on the open list with a k value equal to the minimum of the old and new h values (algorithm 14, line 17). Because k equals h, they are now lower states. See figure H.30 (left). Now, $k_{min} = h(X_c)$ which is the h-value of the current robot position, the terminating condition of $REPAIR - REPLAN$ (algorithm 8). Note that X_c is still on the open list and this should not be a concern because even if X_c were popped off of the open list, no improvement can be made because the current path cost h is already optimal. Finally, the new path is determined via gradient descent of the h values (figure H.30 (right)), and then the robot follows the path to the goal (figure H.31).

Algorithm 9 $GET - BACKPOINTER - LIST(L, S, G)$

Input: A list of states L and two states (start and goal)
Output: A list of states from start to goal as described by the back-pointers in the list of states L

```
1: if path exists then
2:    Return (The list of states)
3: else
4:    Return (NULL)
5: end if
```

Algorithm 10 $INSERT(O, X, h_{new})$

Input: Open list, a state, and an h-value
Output: Open list is modified

```
1: if t(X) = NEW then
2:    k(X) = h_new
3: else if t(X) = OPEN then
4:    k(X) = min(k(X), h_new)
5: else if t(X) = CLOSED then
6:    k(X) = min(h(X), h_new)
7: end if
8: h(X) = h_new
9: t(X) = OPEN
10: Sort O based on increasing k values
```

Algorithm 11 $MODIFY - COST(O, X, Y, cval)$

Input: The open list, two states and a value
Output: A k-value and the open list gets updated

1: $c(X, Y) = cval$
2: **if** $t(X) = CLOSED$ **then**
3: $INSERT(O, X, h(X))$
4: **end if**
5: Return $GET - KMIN(O)$ (algorithm 13)

Algorithm 12 $MIN - STATE(O)$

Input: The open list O
Output: The state with minimum k value in the list related values

1: **if** $O = \emptyset$ **then**
2: Return (-1)
3: **else**
4: Return $(\text{argmin}_{Y \in O} k(Y))$
5: **end if**

Algorithm 13 $GET - KMIN(O)$

Input: The open list O
Output: Lowest k-value of all states in the open list

1: **if** $O = \emptyset$ **then**
2: Return (-1)
3: **else**
4: Return $(\min_{Y \in O} k(Y))$
5: **end if**

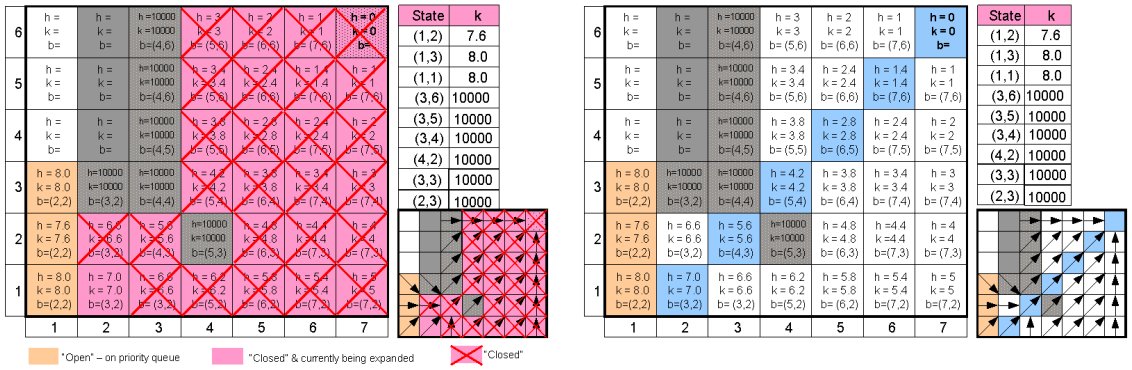

FIGURE H.24. (Left) Termination of initial search phase: start cell is expanded. (Right) Tracing backpointers yields the optimal path, or is it?

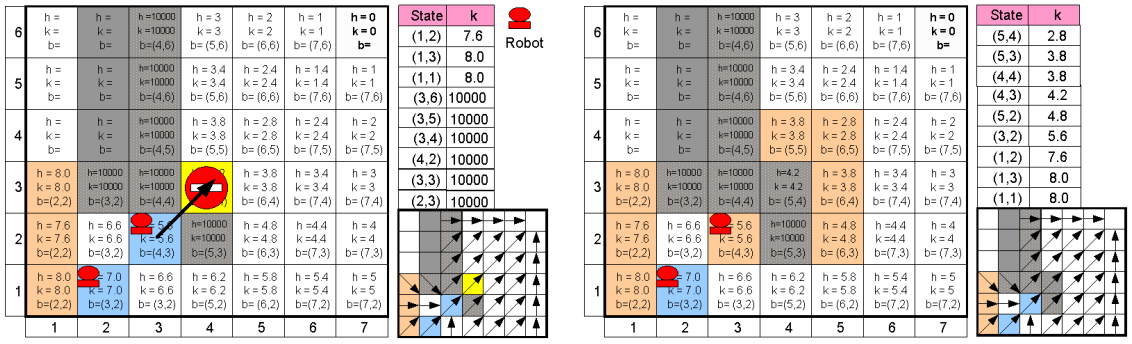

FIGURE H.25. (Left) The robot physically starts tracing the optimal path. (Right) The robot cannot trace the assumed optimal path: gate $(4, 3)$ prevents passed. All nodes surrounding $(4, 3)$ are put on the open list.

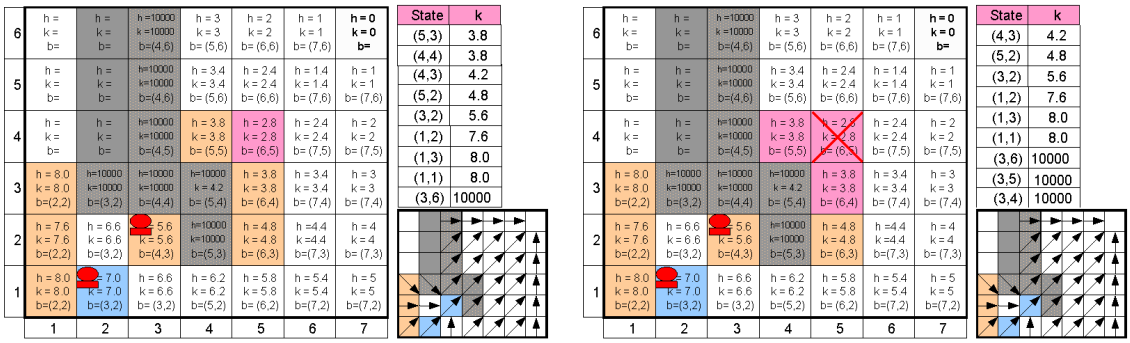

FIGURE H.26. (Left) Pop $(5,4)$ off of the open list and expand; node $(4,3)$ becomes a raise state. (Right) Pop $(5,3)$ off of open list but this has not effect.

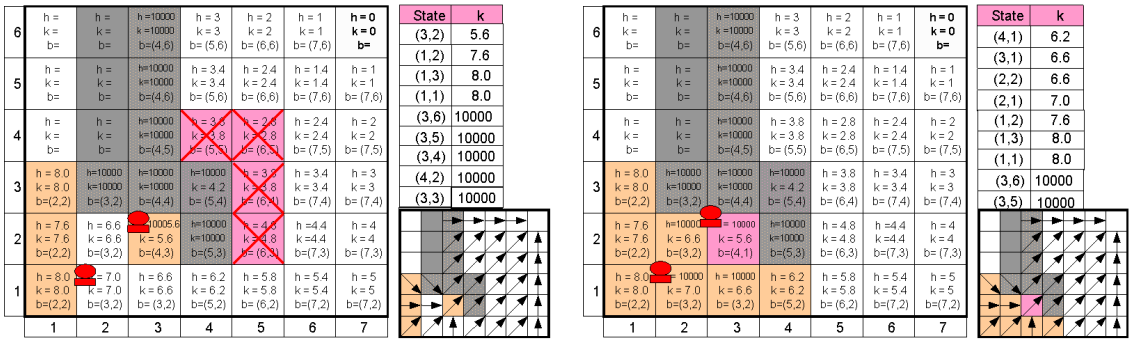

FIGURE H.27. (Left) Pop $(4,3)$ off of open list, and try to find a better path through it; none exist. Eventually $(3,2)$ is put on the open list as a raise state. (Right) Pop $(3,2)$ off of open list.

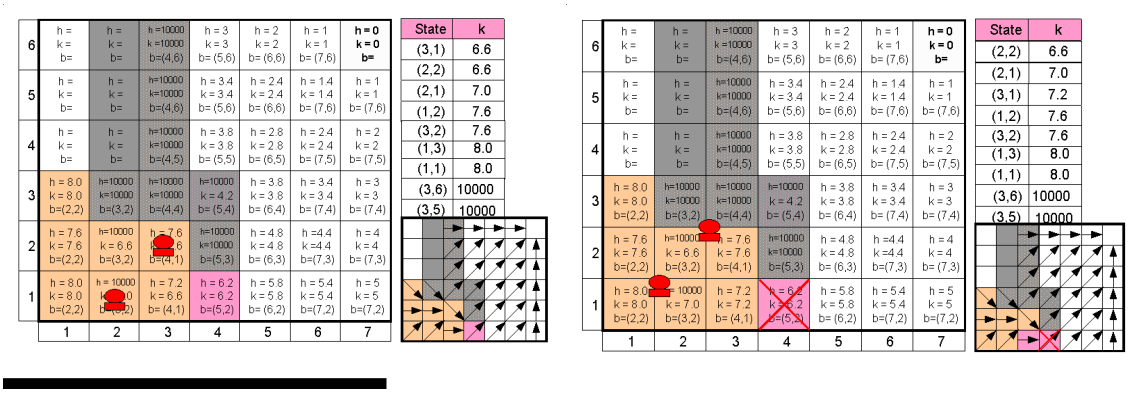

FIGURE H.28. (Left) Pop $(4,1)$ off of open list (Right) Pop $(3,1)$ off of open list.

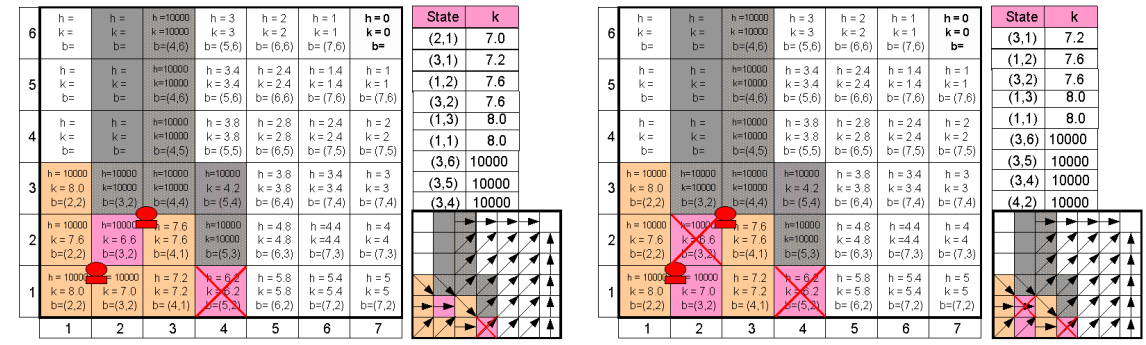

FIGURE H.29. (Left) Pop (2,2) off the queue and expand it (Right) Pop $(2,1)$ off of open list.

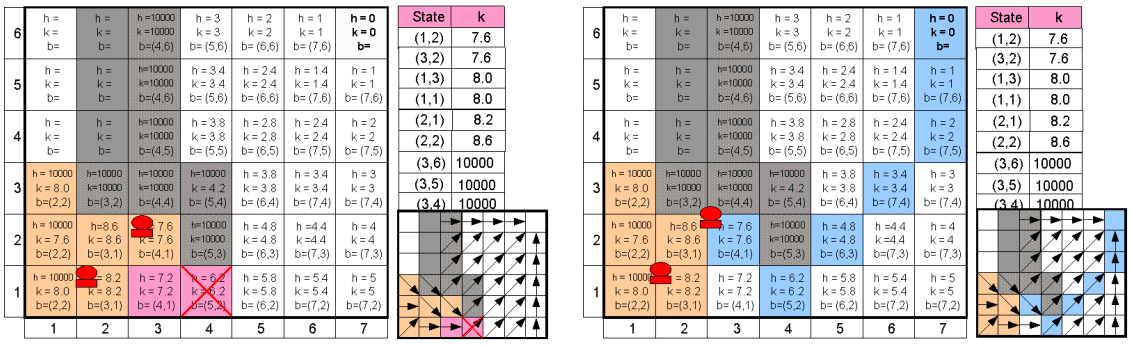

FIGURE H.30. (Left) Pop (3,1) off the queue and expand it. (Right) Determine optimal path from the current location to the goal by following gradient of h values.

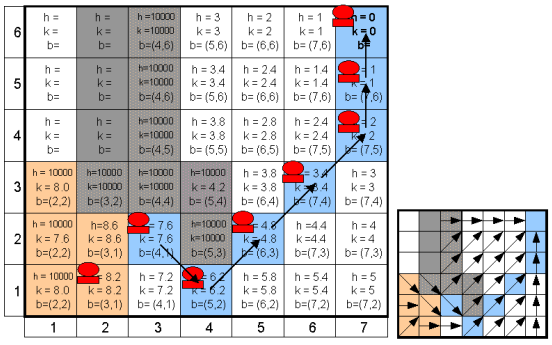

FIGURE H.31. Robot moves to goal from its current location.

Algorithm 14 $PROCESS - STATE$

Input: List of all states L and the list of all states that are open O
Output: A k_{min}, an updated list of all states, and an updated open list

1: $X = MIN - STATE(O)$ (algorithm 12)
2: **if** $X = NULL$ **then**
3: Return (-1)
4: **end if**
5: $k_{old} = GET - KMIN(O)$ (algorithm 13)
6: $DELETE(X)$
7: **if** $k_{old} < h(X)$ **then**
8: **for each** neighbor $Y \in L$ of X **do**
9: **if** $t(Y) \neq NEW$ and $h(Y) \leq k_{old}$ and $h(X) > h(Y) + c(Y, X)$ **then**
10: $b(X) = Y$
11: $h(X) = h(Y) + c(Y, X)$;
12: **end if**
13: **end for**
14: **end if**
15: **if** $k_{old} = h(X)$ **then**
16: **for each** neighbor $Y \in L$ of X **do**
17: **if** $(t(Y) = NEW)$ or $(b(Y) = X$ and $h(Y) \neq h(X) + c(X, Y))$ or $(b(Y) \neq X$ and $h(Y) > h(X) + c(X, Y))$ **then**
18: $b(Y) = X$
19: $INSERT(O, Y, h(X) + c(X, Y))$ (algorithm 10)
20: **end if**
21: **end for**
22: **else**
23: **for each** neighbor $Y \in L$ of X **do**
24: **if** $(t(Y) = NEW)$ or $(b(Y) = X$ and $h(Y) \neq h(X) + c(X, Y))$ **then**
25: $b(Y) = X$
26: $INSERT(O, Y, h(X) + c(X, Y))$
27: **else if** $b(Y) \neq X$ and $h(Y) > h(X) + c(X, Y)$ **then**
28: $INSERT(O, X, h(X))$
29: **else if** $(b(Y) \neq X$ and $h(X) > h(Y) + c(Y, X))$ and $(t(Y) = CLOSED)$ and $(h(Y) > k_{old})$ **then**
30: $INSERT(O, Y, h(Y))$
31: **end if**
32: **end for**
33: **end if**
34: Return $GET - KMIN(O)$ (algorithm 13)

H.4 D^* Lite

The D^* Lite algorithm is perhaps an easier-to-understand advancement over the D^* approach and is therefore more often used. One need not know the details of D^* to understand D^* Lite, however we make some comparisons here for the sake of explanation. Just like D^*, D^* Lite has the effect of locally repairing the graph when a change occurs. However, D^* Lite does not have any back pointers to determine a path; instead each D^* Lite node contains additional values: an objective function g and a "look ahead" function rhs. In general, g is a type of cost-to-goal function.

Nodes are called *consistent* if their g and rhs values are the same and likewise are *inconsistent* if their g and rhs functions differ. If $g > rhs$, then a node is *over-consistent* and if $g < rhs$, the a node is *under-consistent*. This notion of consistency is analogous to the raise and lower states of D^*. Finally, there is a heuristic function h, which has the same meaning as h from A^* and is therefore different from the h in D^*.

The graph being search is assumed to be a directed graph where $c(u, v)$ is the cost to traverse a directed edge from the source node u to the destination v (D^* defined $c(u, v)$ to be the cost from v to u). Hence, the $Succ(u)$ and the $Pred(u)$ are the successors and predecessors, respectively, of the node u. With these terms in-hand, the rhs function is defined as

$$rhs(u) = \min_{s' \in Succ(u)} (c(u, s') + g(s'))$$

There is an open list (again sometimes called a priority queue) U whose key k is now a two-vector, as opposed to the real k values as before. This key is

$$k(s) = \begin{bmatrix} \min(g(s), rhs(s)) + h(s_{start}, s) \\ \min(g(s), rhs(s)) \end{bmatrix}$$

Sometimes, the first and second components of the key are called the primary key and secondary key, respectively. If the primary key of u is less than the primary key of v, then $k(u) < k(v)$. If the primary keys are equal, then the secondary keys are used as a tie breaker.

The D^* Lite algorithm (algorithm 15) takes as input a directed graph, the edge costs (which can be modified), a start state and a goal state. In general, after an initial search, D^* Lite determines a path by performing gradient descent on the sum of the edge costs and objective function g, i.e., chose the next node whose g value in addition to the cost to get there is smallest. This procedure terminates either when the robot reaches the goal or detects a change in edge cost. At this point, the edge costs are repaired

and the rhs and g values are updated to reflect these changes. Once all of the changes are computed, D^* Lite continues with gradient descent to follow the optimal path. Ultimately, algorithm 15 does not output anything, per say, but the search graph is updated so that the rhs and g values are assigned appropriately.

Now, let us take a closer look at algorithm 15. Assume without loss of generality that the start and goal nodes initially differ, so the objective is to move from the start to the goal. First, algorithm 16 initializes the open list to empty, sets all of the rhs and g values to infinity, assigns the rhs value of the goal to zero, and places the goal on the open list. This makes the goal inconsistent, specifically over-consistent. Therefore, when $ComputeShortestPath$ (algorithm 17) is called, the highest priority element, i.e., the lowest key value, of the open list is the goal. Naturally, the goal's key value is less than the key value of the start. $ComputeShortestPath$ then makes the goal consistent by setting the g value equal to its rhs value and for all nodes with outgoing edges that terminate at the goal, each node has its rhs value updated. This process repeats until the start node is consistent and the top key on the open list is not less than the key of the start node. At this point, the loop terminates and an optimal path can be determined.

While the current and goal nodes are not the same, the robot moves from the current node toward the goal. At each step, the planner directs the robot to the successor node whose g value summed with cost to traverse the edge to that successor node is minimal over all successor nodes, i.e., from u, chose the next node such that $c(u, s') + g(s')$ is minimized over all $s' \in Succ(u)$.

In the process of following the optimal path to the goal, if there are any changes in the graph, or more specifically if there are any changes in the graph within sensor range of the robot, D^* Lite first updates these edge costs, updates the source nodes of the affected edges, updates the keys of the appropriate nodes in the open list, and then calls $ComputeShortestPath$ again to make the appropriate nodes consistent. This last action has the ultimate effect of locally repairing the optimal path by altering the g and rhs values. This entire process continues until the current and goal states are the same.

The $ComputeShortestPath$ (algorithm 17) does nothing unless the start node is inconsistent or the lowest priority node in the open list has a key value less than the start's key value. If this is the case, the lowest priority state u is popped off the open list. If it is over-consistent, $ComputeShortestPath$ makes u consistent and updates all of the nodes with edges terminating at u. If u is under-consistent, then $ComputeShortestPath$ makes u over-

Algorithm 15 $D^*Lite(S, s_{start}, s_{goal})$

Input: A graph of nodes S and two nodes (start and goal)
Output: A modified graph of nodes S with their k and rhs values properly set.

$s_{current} = s_{start}$
$Initialize()$ (algorithm 16)
$ComputeShortestPath()$ (algorithm 17)
while $s_{current} \neq s_{goal}$ **do**
  if $g(s_{current}) = \infty$ **then**
    Break (No path exists)
  end if
  $s_{current} = \text{argmin}_{s' \in Succ(s_{current})}(c(s_{current}, s') + g(s'))$
  Move to $s_{current}$
  Scan graph for any changed costs (within sensor limits)
  if any edge cost changed **then**
    for each directed edge (u, v) with changed cost **do**
      Update edge cost $c(u, v)$
      $UpdateVertex(u)$
    end for
    for each $s \in U$ **do**
      $Update(U, s, CalculateKey(s)$
    end for
    $ComputShortestPath()$
  end if
end while

Algorithm 16 $Initialize()$

Input: The start s_{start}, $s_{current}$, goal s_{goal}, open list U and graph of states S are global variables.
Output: A modified open list U and modified graph of states S with updated rhs and g values.

$U = \emptyset$
for each $s \in S$ **do**
  $rhs(s) = g(s) = \infty$
end for
$rhs(s_{goal}) = 0$
$Insert(U, s_{goal}, CalculateKey(s_{goal}))$

Algorithm 17 *ComputeShortestPath*()

Input: The start s_{start}, $s_{current}$, goal s_{goal}, open list U and graph of states S are global variables.

Output: A modified open list U and modified graph of states S with updated rhs and g values.

```
1: while (TopKey(U) < CalculateKey(s_current)) or (rhs(s_current) ≠ g(s_current)) do
2:   u = Pop(U)
3:   if g(u) > rhs(u) then
4:     g(u) = rhs(u)
5:     for each s ∈ Pred(u) do
6:       UpdateVertex(s)
7:     end for
8:   else
9:     g(u) = ∞
10:    for each s ∈ Pred(u) ⋃ {u} do
11:      UpdateVertex(s)
12:    end for
13:  end if
14: end while
```

consistent and updates u, as well as all nodes with edges terminating at u. The nodes are updated in algorithm 18 and the keys are calculated in algorithm 19.

Algorithm 18 *UpdateVertex*(u)

Input: A node u, and the start s_{start}, $s_{current}$, goal s_{goal}, and open list U are global variables.

Output: A modified node u and a modified open list U.

```
if u ≠ s_goal then
  rhs(u) = min_{s'∈Succ(u)}(c(u, s') + g(s'))
end if
if u ∈ U then
  Remove(U, u)
end if
if g(u) ≠ rhs(u) then
  Insert(U, u, CalculateKey(u))
end if
```

Algorithm 19 $CalculateKey(s)$

Input: A node s.
Output: A key for s.

Return $(\min(g(s), rhs(s)) + h(s_{current}, s); \min(g(s), rhs(s)))$

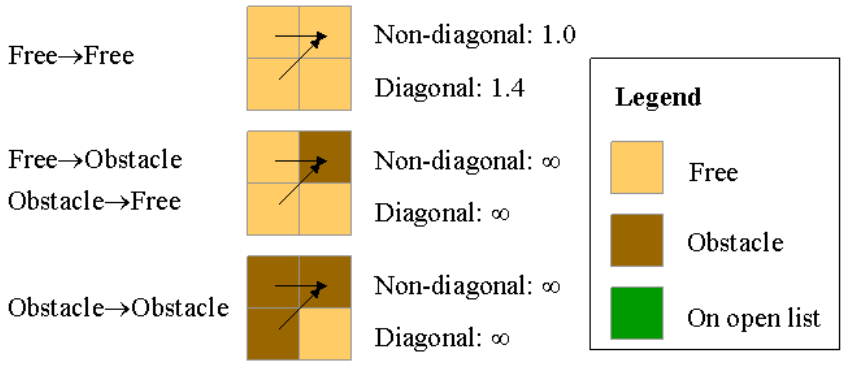

FIGURE H.32. Eight connected grids with pairwise directed edges between neighboring cells.

Let's consider the example where D^* Lite searches a grid of cells. Each node in the graph corresponds to a cell and each pair of neighboring nodes u and v has a pair of directed edges: one from u to v and visa versa. The cost to travel from one free cell to a neighboring free cell is 1 if it is an up, down, left or right motion, and is 1.4 if it is a diagonal motion. The cost of travel either from an obstacle cell or to an obstacle cell infinite, as depicted in figure H.32.

Initially, all of the nodes' rhs and g values are set to infinity, except for the goal whose rhs value is set to zero and its g value is set to infinity. Since the goal is now inconsistent, it is put on the open list. See figure H.33.

D^* Lite then calls $ComputeShortestPath$ which immediately pops the goal off of the open list, and since it is over-consistent, makes it consistent with rhs and g values of zero. Now, $ComputeShortestPath$ expands the popped node by calling $UpdateVertex$ on all of its predecessors. This computes rhs values for the predecessors and puts them on the open list, but only if they become inconsistent. Node $(1, 1)$ is a predecessor but not put on the open list because it remained consistent. See figure H.34 where the small arrows indicate which node is used to compute the rhs value, e.g., the

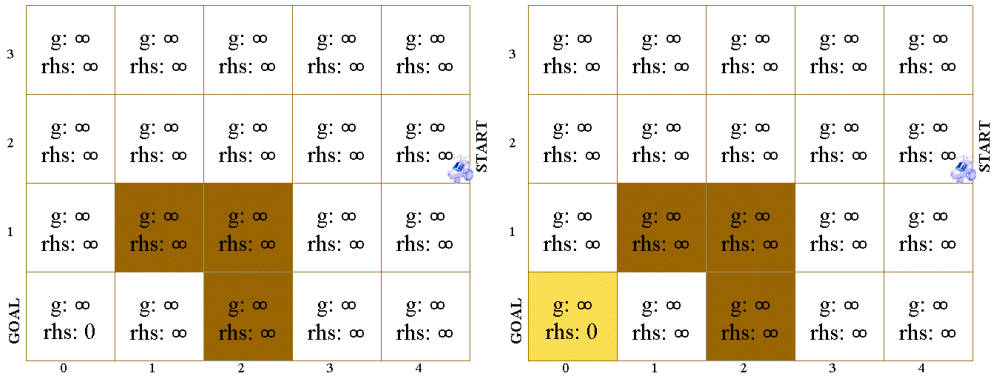

FIGURE H.33. (Left) Goal *rhs* value set to zero and all other *rhs* and *g* values to infinity (Right) Goal is put on open list. Arrows are *not* back pointers but rather represent gradient directions.

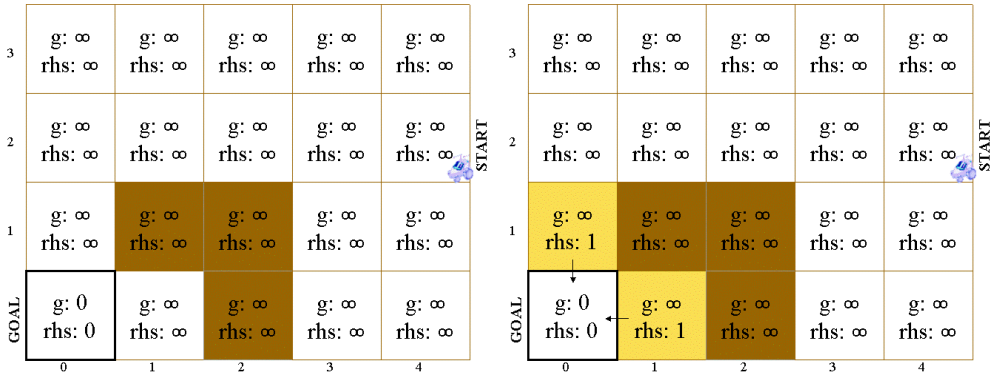

FIGURE H.34. (Left) The goal is popped of the open list. (Right) The goal is expanded and the resulting inconsistent nodes are put the on open list.

rhs value of $(0, 1)$ is computed using the *g* value of $(0, 0)$ and the transition cost from $(1, 0)$ to $(0, 0)$, i.e., $1 = 0 + 1$.

Continue in *ComputeShortestPath* by popping $(0, 1)$, which is tied with $(1, 0)$ for the minimum node in the open list. Here, *UpdateVertex* is called on all of the predecessors of the popped node. When *UpdateVertex* is called

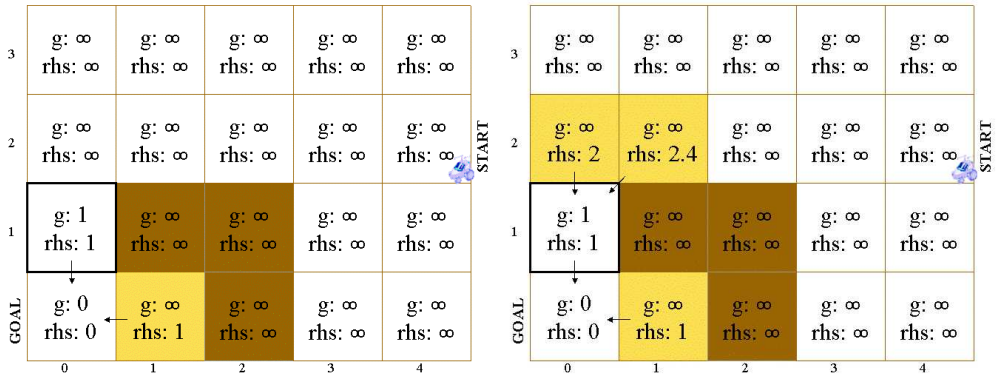

FIGURE H.35. (Left) Pop minimum node off of list but do not expand any neighbors on open list. (Right) Pop minimum node off of open list and put inconsistent neighboring nodes on the open list.

on each predecessor node, the rhs value of each predecessor is updated by examining the g-values of each of the predecessor's successors in the graph. Since, $(0, 1)$ is over-consistent, it is made consistent and all predecessors of $(0, 1)$ have their rhs values updated via $UpdateVertex$. Two of its predecessors become inconsistent and are put on the open list. Again, the rhs values of the predecessors $(0, 0)$ and $(1, 1)$ did not change, and as such, did not become inconsistent and are not put on the open list. The rhs value of $(1, 0)$ did not also did not change but was already inconsistent and on the open list. See figure H.35. Now, $(1, 0)$ is expanded but no predecessors of $(1, 0)$ are put on the open list because they remained consistent after calling $UpdateVertex$.

This Dijkstra-like search continues until the start node is effectively expanded and made consistent. In figure H.36, $(3, 1)$ is popped and expanded, and all of its predecessors, which become inconsistent, are put on the open list. Note that the start was already on the open list. At this point, the start has the lowest key value, so is popped off the open list and made consistent. In this case, none of the predecessors of the start become inconsistent, so are not put on the open list, although some were already on the open list. At this point, since the start node is consistent and the top key on the open list is not less than the key of the start node, an optimal path exists. This allows the $ComputeShortestPath$ loop to terminate. Finally, note that some

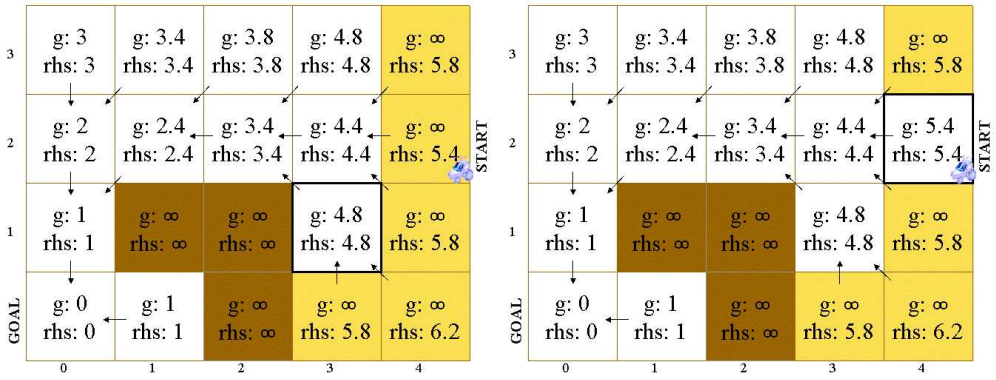

FIGURE H.36. (Left) Pop minimum node off of open list and put inconsistent neighboring nodes on the open list. (Right) Pop start off of the open list, make it consistent, but no nodes are put on the open list.

nodes remain on the open list and for examples with larger graphs, some nodes may not have been considered at all.

The robot then follows the negated gradient of g from the start to the goal until the robot detects an obstacle at $(2, 2)$. See figure H.37. The algorithm dictates that for all directed edges (u, v) with changed edge costs, $UpdateVertex(u)$ is called. Since the edges are directed and in this example all neighboring cells u and v have two edges, one from u to v and visa versa, $(2, 2)$ has 16 affected edges. See figure H.38.

Let's consider the outgoing and incoming edges to $(2, 2)$ separately. For each of the outgoing edges, $UpdateVertex$ is called on $(2, 2)$. First, the outgoing edge to $(2, 3)$ is called. Since the edge cost is now infinite, the rhs value of $(2, 2)$ is raised to infinity making it inconsistent and hence $(2, 2)$ is put on the open list. Now, when $UpdateVertex$ is called for the rest of the outgoing edges of $(2, 2)$, nothing happens because $(2, 2)$ remains inconsistent.

Now, consider the incoming edges to $(2, 2)$. One of the predecessors of the incoming edge to $(2, 2)$ is $(3, 3)$, so $UpdateVertex$ is called on $(3, 3)$. The minimum possible rhs value of $(3, 3)$ is still 4.8, but this value is based on the g value of $(2, 3)$, not $(2, 2)$. The node $(3, 3)$ is still consistent, so it is not put on the open list. Another incoming edge to $(2, 2)$ comes from $(3, 2)$, so $UpdateVertex$ is called on this node. Since the transition cost to $(2, 2)$ increased, the minimum possible rhs value of $(3, 2)$ is now 5.2, computed

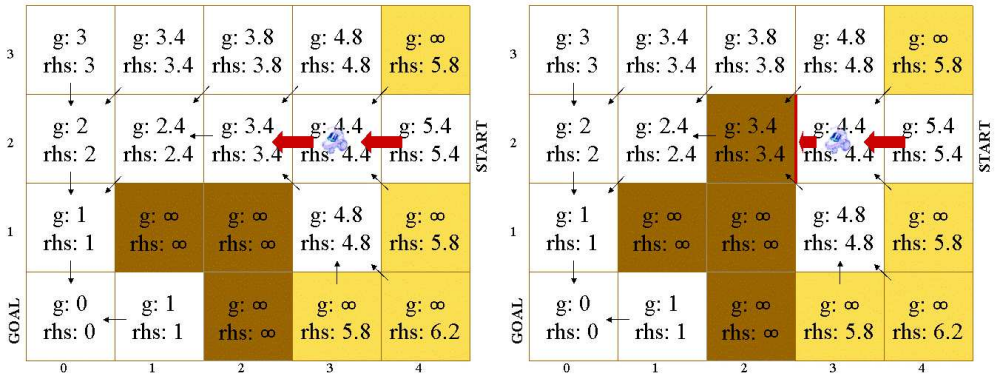

FIGURE H.37. (Left) Follow optimal path via gradient descent of g. (Right) The robot discovers that $(2, 2)$ is an obstacle.

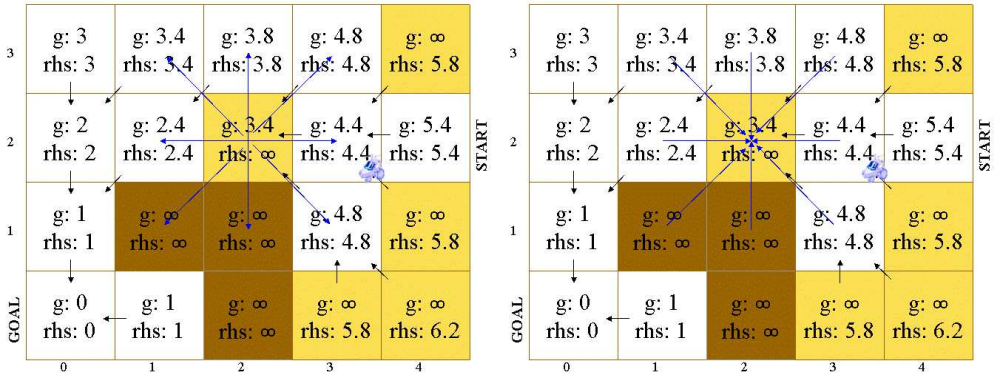

FIGURE H.38. (Left) Outgoing edges to $(2, 2)$. (Right) Incoming edges to $(2, 2)$.

based on the g value of $(2, 3)$ where $5.2 = 3.8 + 1.4$. See figure H.39.

Another incoming edge to $(2, 2)$ comes from $(3, 1)$. The minimum possible rhs value of $(3, 1)$ is now 5.4, computed based on the g value of $(3, 2)$. Again, note that the rhs value of a node is always computed using the g, not a rhs, values of its successors. The remaining five nodes – $(1, 1)$, $(1, 2)$, $(1, 3)$, $(2, 3)$ and $(2, 1)$ – remain consistent and hence are not put on the open list. See figure H.40.

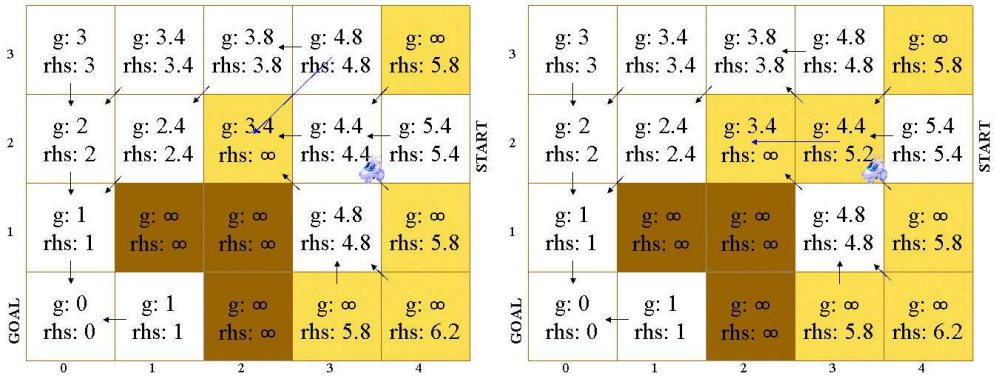

FIGURE H.39. Incoming edges to $(2,2)$ (Left) Node $(3,3)$ is considered. (Right) Node $(3,2)$ is considered

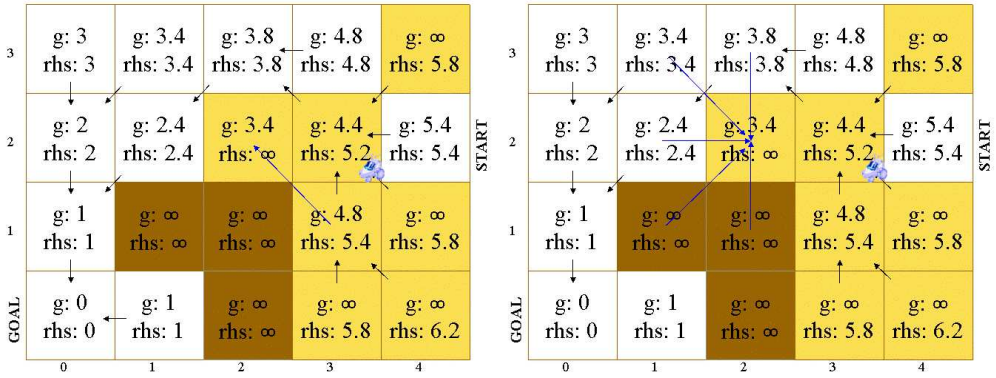

FIGURE H.40. Incoming edges to $(2,2)$ (Left) Node $(3,1)$ is considered. (Right) The remaining nodes are considered

Note that the processing order of $(3,2)$ versus $(3,1)$ does not matter because when $(3,2)$ is processed, its *rhs*-value, not its g-value, is updated. Then, when $(3,1)$ is updated, its *rhs*-value is updated and is based on the g-value of $(3,2)$, which has not changed, and not its rhs value. As such, we will get the same effect whether we process $(3,2)$ before $(3,1)$ or vice versa.

Now, D^* Lite goes back to *CompteShortestPath* until a new optimal path is found. Note that the current robot position is inconsistent and does not have the smallest key value in the open list. This indicates an optimal

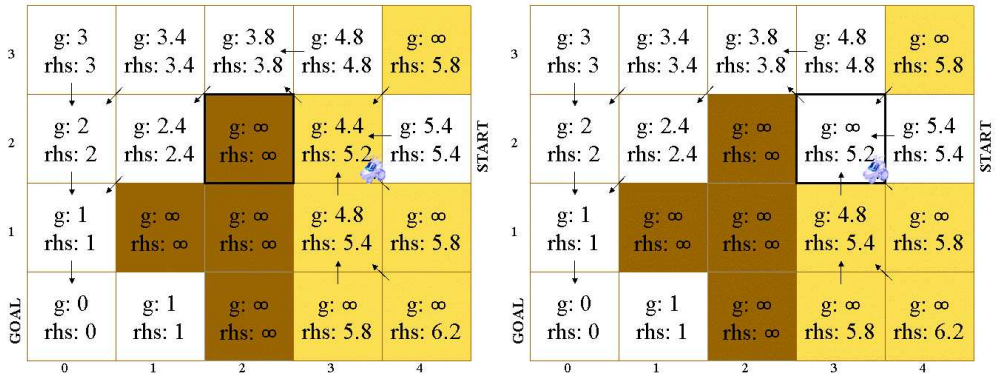

FIGURE H.41. (Left) Node $(2, 2)$ is considered. (Right) Node $(3, 2)$ is considered.

path, based on all available information, has not been determined. The node with the minimum key is $(2, 2)$. It is under-consistent, so its g value is set to infinity and $UpdateVertex$ is called on all of its predecessors. This does not make any of the predecessors inconsistent, so none are put on the open list. See figure H.41.

Next, $(3, 2)$ is popped and it is under-consistent, so its g is set to infinity. Next, its predecessors are updated: $(4, 2)$ becomes inconsistent, $(3, 1)$ is updated but remains inconsistent, $(4, 1)$ remains inconsistent but its rhs value does not change and is now computed from the g value of $(3, 1)$. Also, $(3, 2)$ is updated, remains inconsistent and is put back on the open list. See figure H.42.

Still in the $ComputeShortestPath$ procedure, $(3, 1)$ is popped off the open list and since it is under-consistent, its g value is made infinite and its predecessors are updated: $(4, 1)$ is updated and remains inconsistent, while $(3, 0)$ and $(4, 0)$ are updated but are now consistent since both g and rhs are infinite. See figure H.43.

Also, since $(3, 1)$ is under-consistent, $ComputeShortest$ path calls $UpdateVertex$ on $(3, 1)$, which results in putting $(3, 1)$ back on the open list since it remains inconsistent. Now, $(3, 2)$ has the smallest key value, so it is popped off of the open list and since it is over-consistent, its g value is set to its rhs value. See figure H.44. When its predecessors are updated, $(3, 1)$ is modified but still remains inconsistent, so it stays on the open list. See figure H.45 (left).

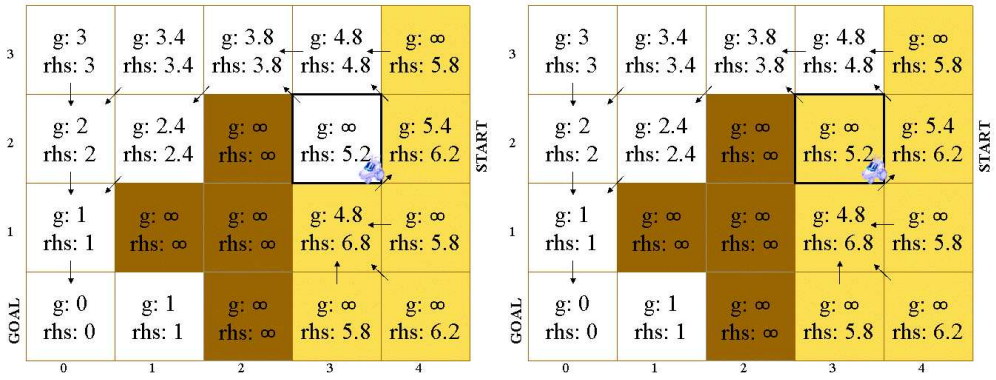

FIGURE H.42. Expand $(3, 2)$ and update its predecessors and it.

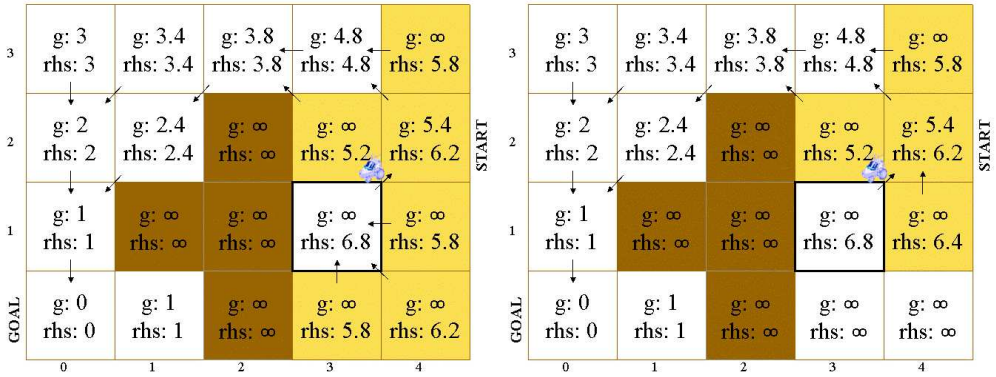

FIGURE H.43. (Left) Node $(3, 1)$ is popped. (Right) The predecessor nodes of $(3, 1)$ are updated.

Once again, the node corresponding to the robots current position is consistent *and* the top key on the open list is not less than the key of current position. Therefore, a new optimal path has been found and *ComputeShortestPath* breaks out of its loop. Once again, the optimal path is determined by following the gradient of g. See figure H.45.

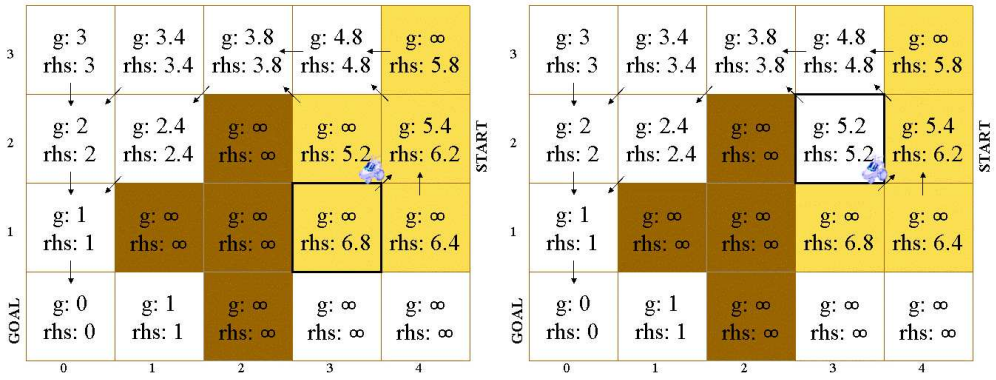

FIGURE H.44. (Left) Node $(3, 1)$ remains on the open list (Right) Node $(3, 2)$ is expanded.

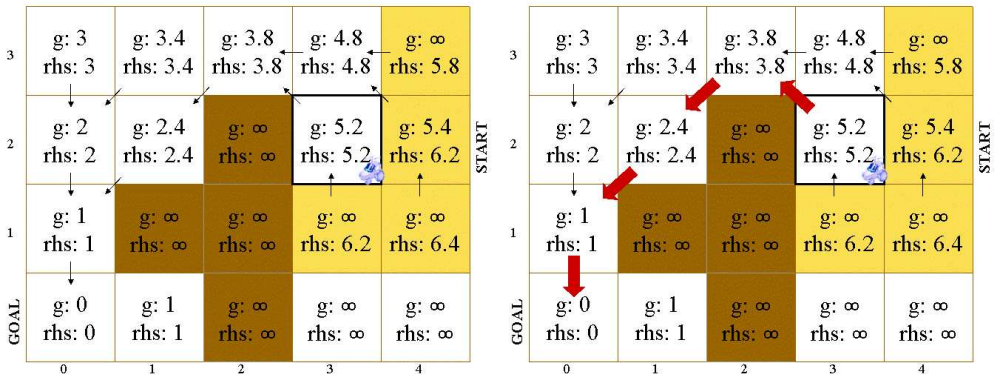

FIGURE H.45. (Left) Update the predecessors of $(3, 2)$ (Right) A new optimal path has been found.

H.5 A Comment on Reverse Searching

The search starts at the goal, and works backward, for good reason. After the initial search, the back pointers form a search tree which is rooted at the goal. Bearing in mind that the robot detects changes in edge costs near its current position, one could easily see that if the root of the search tree were the start node, then the search tree would have be drastically recomputed. With the goal node at the root of the tree, when an edge cost changes,

only a subtree that contains the current robot position is repaired where part of that subtree will be redirected to nodes in a neighboring subtree. In fact, in general, the entire subtree is not repaired; the repairing process terminates when an optimal path from the current robot position to the goal is determined.

A change in the environment is just one reason why a robot may need to replan its path. Another has to do with the stochastic nature of the robot's motion. For example, error in control or unforeseen slippage may cause the robot to fall off its intended path. The benefit of performing the reverse search is that for small perturbations, an optimal path to the goal for nearby nodes was already computed during the initial Dijkstra-like search. In fact, one can determine the best action for *all* nodes in the graph, not just the ones along the shortest path.

A mapping from nodes to actions is called a *universal plan*, or *policy*. Techniques for finding optimal policies are known as universal planners and can be computationally more involved than the shortest path techniques surveyed here. One simple way to attain a universal plan to a goal is to run Dijkstra's algorithm backward (as in D^*): After completion, we know for each node in the graph the length of an optimal path to the goal, along with the appropriate action. Generalizations of this approach are commonly used in stochastic domains, where the outcome of actions is modeled by a probability distribution over nodes in the graph.

Appendix I

Statistics Primer

- On Pg 548, the $\wedge$ should be a $\cap$. So
  $\Pr(E_1 \cap E_2) = \Pr(E_1) \cdot \Pr(E_2)$
  should read
  $\Pr(E_1 \cap E_2) = \Pr(E_1) \cdot \Pr(E_2)$
- On pg 549, cumulative is misspelled, twice.

Appendix J

Linear Systems and Control

- The dot over $x(k+1)$ should not be there in eq (J.10). So the $\dot{x}(k+1)$ should read $x(k+1)$.

Bibliography

[1] http://www.aemdesign.com.

[2] http://www.sbsi-sol-optimize.com/NPSOL.htm.

[3] http://www.vni.com.

[4] http://www.nag.com.

[5] *Webster's Ninth New Collegiate Dictionary*. Merriam-Webster, Inc., Springfield, MA, 1990.

[6] R. Abraham, J. Marsden, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications*. Springer-Verlag, New York, 2 edition, 1988.

[7] R. Abraham and J. E. Marsden. *Foundations of Mechanics*. Addison-Wesley, 1985.

[8] E. U. Acar and H. Choset. Sensor-based coverage of unknown environments: Incremental construction of Morse decompositions. *International Journal of Robotics Research*, 21:345–366, April 2002.

[9] E. U. Acar, H. Choset, A. A. Rizzi, P. Atkar, and D. Hull. Morse decompositions for coverage tasks. *International Journal of Robotics Research*, 21:331–344, April 2002.

[10] M. Akinc, K. E. Bekris, B. Chen, A. Ladd, E. Plaku, and L. E. Kavraki. Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps. In *International Symposium on Robotics Research*, 2003. Book to appear.

[11] R. Alami, J. Laumond, and T. Siméon. Two manipulation planning algorithms. In K. Goldberg, D. Halperin, J. C. Latombe, and R. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 109–125. A.K. Peters, 1995.

[12] R. Alami, T. Siméon, and J. P. Laumond. A geometrical approach to planning manipulation tasks. In *International Symposium on Robotics Research*, pages 113–119, 1989.

[13] P. Allen and I. Stamos. Integration of range and image sensing for photorealistic 3D modeling. In *IEEE International Conference on Robotics and Automation*, 2000.

[14] N. M. Amato, B. Bayazit, L. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3d workspaces. In P. Agarwal, L. E. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 156–168. AK Peters, 1998.

[15] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *IEEE International Conference on Robotics and Automation*, pages 630–637, 1998.

[16] N. M. Amato, K. Dill, and G. Song. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. In *International Conference on Research in Computational Molecular Biology*, pages 2–11, April 2002.

[17] N. M. Amato and G. Song. Using motion planning to study protein folding pathways. In *International Conference on Research in Computational Molecular Biology*, pages 287–296, 2001.

[18] E. Anshelevich, S. Owens, F. Lamiraux, and L. E. Kavraki. Deformable volumes in path planning applications. In *IEEE International Conference on Robotics and Automation*, pages 2290–2295, 2000.

[19] M. Apaydin, D. Brutlag, C. Guestrin, D. Hsu, J. Latombe, and C. Varm. Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion. *Journal of Computational Biology*, 10:257–281, 2003.

[20] M. Apaydin, C. Guestrin, C. Varma, D. Brutlag, and J. Latombe. Studying protein-ligand interactions with stochastic roadmap simulation. *Bioinformatics*, 18(2):18–26, 2002.

[21] M. S. Apaydin, D. L. Brutlag, C. Guestrin, D. Hsu, and J. C. Latombe. Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion. In *International Conference*

on Research in Computational Molecular Biology, pages 12–21, April 2002.

[22] V. I. Arnold. *Mathematical Methods of Classical Mechanics*. Springer-Verlag, 1989.

[23] K. Arras, N. Tomatis, B. Jensen, and R. Siegwart. Multisensor on-the-fly localization: Precision and reliability for applications. *Robotics and Autonomous Systems*, 34(2-3):131–143, 2001.

[24] K. Arras and S. Vestli. Hybrid, high-precision localization for the mail distributing mobile robot system MOPS. In *IEEE International Conference on Robotics and Automation*, 1998.

[25] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[26] F. Aurenhammer. Voronoi diagrams — A survey of a fundamental geometric structure. *ACM Computing Surveys*, 23:345–405, 1991.

[27] D. Avots, E. Lim, R. Thibaux, and S. Thrun. A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[28] B. Baginski. Local motion planning for manipulators based on shrinking and growing geometry models. In *IEEE International Conference on Robotics and Automation*, pages 3303 –3308, 1996.

[29] J. Baillieul and B. Lehman. Open-loop control using oscillatory inputs. In *CRC Control Handbook*, pages 967–980. CRC Press, Boca Raton, FL, 1996.

[30] D. J. Balkcom and M. T. Mason. Time optimal trajectories for differential drive vehicles. *International Journal of Robotics Research*, 21(3):199–217, Mar. 2002.

[31] J. Barraquand and P. Ferbach. A penalty function method for constrained motion planning. In *IEEE International Conference on Robotics and Automation*, pages 1235–1242, 1994.

[32] J. Barraquand, L. E. Kavraki, J. C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for robot path planning. *International Journal of Robotics Research*, 16(6):759–774, 1997.

[33] J. Barraquand, B. Langlois, and J. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Man and Cybernetics*, 22(2):224–241, Mar/Apr 1992.

[34] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. Technical Report STAN-CS-89-1257, Stanford University, Stanford CA, 1989.

[35] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, Dec. 1991.

[36] J. Barraquand and J. C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.

[37] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer-Verlag, 2003.

[38] K. E. Bekris, B. Chen, A. Ladd, E. Plaku, and L. Kavraki. Multiple query motion planning using single query primitives. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 656–661, 2003.

[39] J. Bentley. Multidimensional divide and conquer. *Communications of the ACM*, 23(4), 1980.

[40] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, second edition, 1999.

[41] P. Besl and N. McKay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(14):239–256, 1992.

[42] P. Bessiere, E. Mazer, and J.-M. Ahuactzin. Planning in continuous space with forbidden regions: The ariadne's clew algorithm. In K. Goldberg, K. Goldberg, R. Wilson, and D. Halperin, editors, *Algorithmic Foundations of Robotics (WAFR)*, pages 39–47. A.K. Peters, Wellsley MA, 1995.

[43] P. Bessiere, E. Mazer, and J.-M. Ahuactzin. The ariadne's clew algorithm. *Journal of Artificial Intelligence Research (JAIR)*, 9:295–316, 1998.

[44] J. T. Betts. Survey of numerical methods for trajectory optimization. *AIAA Journal of Guidance, Control, and Dynamics*, 21(2):193–207, March-April 1998.

[45] A. M. Bloch. *Nonholonomic Mechanics and Control*. Springer, New York, 2003.

[46] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *International Journal of Robotics Research*, 4(3):3–17, Fall 1985.

[47] R. Bohlin. Path planning in practice: Lazy evaluation on a multi-resolution grid. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[48] R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. In *IEEE International Conference on Robotics and Automation*, pages 521–528, 2000.

[49] R. Bohlin and L. E. Kavraki. A randomized algorithm for robot path planning based on lazy evaluation. In P. Pardalos, S. Rajasekaran, and J. Rolim, editors, *Handbook on Randomized Computing*, pages 221–249. Kluwer Academic Publishers, 2001.

[50] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. John Wiley and Sons Inc., New York, NY, 2000.

[51] B. Bonnard. Contrôlabilité des systèmes nonlinéaires. *C. R. Acad. Sci. Paris*, 292:535–537, 1981.

[52] V. Boor, N. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, pages 1018–1023, 1999.

[53] W. M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Academic Press, 1986.

[54] J. Borenstein, B. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques*. A.K. Peters, Ltd., Wellesley, MA, 1996.

[55] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *IEEE International Conference on Robotics and Automation*, pages 1481–1487, 2001.

[56] G. E. Bredon. *Topology and Geometry*. Springer-Verlag, New York, NY, 1993.

[57] T. Bretl, J. C. Latombe, and S. Rock. Toward autonomous free climbing robots. In *International Symposium on Robotics Research*, 2003. Book to appear.

[58] R. W. Brockett. Nonlinear systems and differential geometry. *Proceedings of the IEEE*, 64(1):61–72, Jan. 1976.

[59] R. W. Brockett. Control theory and singular Riemannian geometry. In P. J. Hilton and G. S. Young, editors, *New Directions in Applied Mathematics*, pages 11–27. Springer-Verlag, 1982.

[60] R. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions Systems, Man, and Cybernetics*, 15:224–233, 1985.

[61] R. A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3):190–197, 1983.

[62] R. C. Brost. *Analysis and Planning of Planar Manipulation Tasks*. PhD thesis, Carnegie Mellon University, Jan. 1991. Available as Technical Report CMU-CS-91-149.

[63] R. C. Brost. Computing the possible rest configurations of two interacting polygons. In *IEEE International Conference on Robotics and Automation*, pages 686–693, Apr. 1991.

[64] A. E. Bryson. *Dynamic Optimization*. Addison-Wesley, 1998.

[65] A. E. Bryson and Y. C. Ho. *Applied Optimal Control*. Hemisphere Publishing, New York, 1975.

[66] J. Buhmann, W. Burgard, A. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot RHINO. *AI Magazine*, 16(2):31–38, Summer 1995.

[67] F. Bullo. Series expansions for the evolution of mechanical control systems. *SIAM Journal on Control and Optimization*, 40(1):166–190, 2001.

[68] F. Bullo. Averaging and vibrational control of mechanical systems. *SIAM Journal on Control and Optimization*, 41:542–562, 2002.

[69] F. Bullo, N. E. Leonard, and A. D. Lewis. Controllability and motion algorithms for underactuated Lagrangian systems on Lie groups. *IEEE Transactions on Automatic Control*, 45(8):1437–1454, 2000.

[70] F. Bullo and A. D. Lewis. *Geometric Control of Mechanical Systems*. Springer, 2004.

[71] F. Bullo, A. D. Lewis, and K. M. Lynch. Controllable kinematic reductions for mechanical systems: Concepts, computational tools, and examples. In *2002 International Symposium on the Mathematical Theory of Networks and Systems*, Aug. 2002.

[72] F. Bullo and K. M. Lynch. Kinematic controllability for decoupled trajectory planning of underactuated mechanical systems. *IEEE Transactions on Robotics and Automation*, 17(4):402–412, Aug. 2001.

[73] F. Bullo and M. Žefran. On mechanical control systems with nonholonomic constraints and symmetries. *Systems and Control Letters*, 45(2):133–143, Jan. 2002.

[74] W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2), 2000.

[75] W. Burgard, A. Derr, D. Fox, and A. Cremers. Integrating global position estimation and position tracking for mobile robots: the dynamic Markov localization approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1998.

[76] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1996.

[77] W. Burgard, D. Fox, H. Jans, C. Matenar, and S. Thrun. Sonar-based mapping of large-scale mobile robot environments using EM. In *Proc. of the International Conference on Machine Learning (ICML)*, 1999.

[78] L. Bushnell, D. Tilbury, and S. Sastry. Steering three-input nonholonomic systems: The fire-truck example. *International Journal of Robotics Research*, 14(4):366–381, 1995.

[79] Z. J. Butler, A. A. Rizzi, and R. L. Hollis. Contact sensor-based coverage of rectilinear environments. In *Proc. of IEEE Int'l Symposium on Intelligent Control*, Sept. 1999.

[80] P. E. Caines and E. S. Lemch. On the global controllability of Hamiltonian and other nonlinear systems: Fountains and recurrence. In *IEEE International Conference on Decision and Control*, pages 3575–3580, 1998.

[81] S. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Automation*, pages 291–302, 1990.

[82] S. Cameron. Enhancing GJK: Computing minimum distance and penetration distanses between convex polyhedra. In *IEEE International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[83] J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[84] J. Canny. Constructing roadmaps of semi-algebraic sets I: Completeness. *Artificial Intelligence*, 37:203–222, 1988.

[85] J. Canny. Computing roadmaps of general semi-algebraic sets. *The Computer Journal*, 35(5):504–514, 1993.

[86] J. Canny and M. Lin. An opportunistic global path planner. *Algorithmica*, 10:102–120, 1993.

[87] J. Canny, J. Reif, B. Donald, and P. Xavier. On the complexity of kinodynamic planning. In *IEEE Symposium on the Foundations of Computer Science*, pages 306–316, White Plains, NY, 1988.

[88] J. F. Canny. Some algebraic and geometric computations in pspace. In *Proc. 20th ACM Symposium on the Theory of Computing*, pages 460–469, 1998.

[89] Z. L. Cao, Y. Huang, and E. Hall. Region filling operations with random obstacle avoidance for mobile robots. *Journal of Robotic systems*, pages 87–102, February 1988.

[90] J. Carpenter, P. Clifford, and P. Fernhead. An improved particle filter for non-linear problems. *IEE Proceedings on Radar and Sonar Navigation*, 146(2-7), 1999.

[91] A. Casal. *Reconfiguration Planning for Modular Self-Reconfigurable Robots*. PhD thesis, Stanford University, Stanford, CA, 2002.

[92] J. Castellanos, J. Montiel, J. Neira, and J. Tardós. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15(5):948–953, 1999.

[93] J. Castellanos and J. Tardós. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Kluwer Academic Publishers, Boston, MA, 2000.

[94] P. C. Chen and Y. K. Hwang. SANDROS: A motion planner with performance proportional to task difficulty. *IEEE International Conference on Robotics and Automation*, pages 2346–2353, 1992.

[95] P. C. Chen and Y. K. Hwang. SANDROS:a dynamic graph search algorithm for motion planning. *IEEE Transactions on Robotics and Automation*, 14(3):390–403, June 1998.

[96] P. Cheng and S. M. LaValle. Reducing metric sensitivity in randomized trajectory design. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 43–48, 2001.

[97] H. Choset. Nonsmooth analysis, convex analysis, and their applications to motion planning. *Special Issue of the Int. Jour. of Comp. Geom. and Apps.*, 1998.

[98] H. Choset and J. Burdick. Sensor based motion planning: Incremental construction of the hierarchical generalized Voronoi graph. *International Journal of Robotics Research*, 19(2):126–148, February 2000.

[99] H. Choset and J. Burdick. Sensor based motion planning: The hierarchical generalized Voronoi graph. *International Journal of Robotics Research*, 19(2):96–125, February 2000.

[100] H. Choset and J. Y. Lee. Sensor-based construction of a retract-like structure for a planar rod robot. *IEEE Transaction of Robotics and Automation*, 17, 2001.

[101] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (T-SLAM). *IEEE Transactions on Robotics Automation*, 17, April 2001.

[102] H. Choset, K. Nagatani, and A. Rizzi. Sensor based planning: Using a honing strategy and local map method to implement the generalized Voronoi graph. In *SPIE Conference on Systems and Manufacturing*, Pittsburgh, PA, 1997.

[103] H. Choset and P. Pignon. Coverage path planning: The boustrophedon decomposition. In *Proceedings of the International Conference on Field and Service Robotics*, Canberra, Australia, December 1997.

[104] P. Choudhury and K. M. Lynch. Trajectory planning for second-order underactuated mechanical systems in the presence of obstacles. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, pages 559–575. Springer-Verlag, 2002.

[105] W.-L. Chow. Uber systemen von linearen partiellen differentialgleichungen erster ordnung. *Math. Ann.*, 117:98–105, 1939.

[106] S. Ciarcia. An ultrasonic ranging system. *Byte Magazine*, pages 113–123, October 1984.

[107] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Society of Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[108] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Symposium on Interactive 3D Graphics*, pages 189–196, 218, 1995.

[109] J. Colegrave and A. Branch. A case study of autonomous household vacuum cleaner. In *AIAA/NASA CIRFFSS*, 1994.

[110] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Lecture Notes in Computer Science*, volume 33, pages 134–183. Springer-Verlag, 1975.

[111] H. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[112] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2002.

[113] J. Cortes, S. Martinez, J. P. Ostrowski, and H. Zhang. Simple mechanical control systems with constraints and symmetry. *SIAM Journal on Control and Optimization*, 41(3):851–874, 2002.

[114] J. Cortés, T. Simeon, and J.-P. Laumond. A random loop generator for planning the motions of closed kinematic chains. In *IEEE International Conference on Robotics and Automation*, pages 2141–2146, 2002.

[115] J. Crowley. World modeling and position estimation for a mobile robot using ultrasound ranging. In *IEEE International Conference on Robotics and Automation*, 1989.

[116] T. Danner and L. E. Kavraki. Randomized planning for short inspection paths. In *IEEE International Conference on Robotics and Automation*, pages 971–976, San Fransisco, CA, April 2000. IEEE Press.

[117] M. de Berg, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 1997.

[118] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization for mobile robots. In *IEEE International Conference on Robotics and Automation*, 1999.

[119] F. Dellaert, S. Seitz, C. Thorpe, and S. Thrun. Structure from motion without correspondence. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2000.

[120] A. O. Dempster, A. N. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[121] G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 2001.

[122] A. W. Divelbiss and J. Wen. Nonholonomic path planning with inequality constraints. In *IEEE International Conference on Decision and Control*, pages 2712–2717, 1993.

[123] A. W. Divelbiss and J.-T. Wen. A path space approach to nonholonomic motion planning in the presence of obstacles. *IEEE Transactions on Robotics and Automation*, 13(3):443–451, 1997.

[124] M. P. do Carmo. *Riemannian Geometry*. Birkhäuser, Boston, MA, 1992.

[125] B. Donald. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence*, 31:295–353, 1987.

[126] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the Association for Computing Machinery*, 40(5):1048–1066, Nov. 1993.

[127] B. R. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open chain manipulators. *Algorithmica*, 4(6):480–530, 1995.

[128] B. R. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: robots with decoupled dynamics bounds. *Algorithmica*, 4(6):443–479, 1995.

[129] A. Doucet. On sequential simulation-based methods for Bayesian filtering. Technical report, Department of Engeneering, University of Cambridge, 1998.

[130] A. Doucet, J. de Freitas, K. Murphy, and S. Russel. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

[131] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.

[132] D. Duff, M. Yim, and K. Roufas. Evolution of polybot: A modular reconfigurable robot. In *Proc. of the Harmonic Drive Intl. Symposium*, Nagano, Japan, 2001.

[133] S. Ehmann and M. C. Lin. Swift: Accelerated distance computation between convex polyhedra by multi-level Voronoi marching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[134] S. A. Ehmann and M. C. Lin. Geometric algorithims: Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum - Proc. of Eurographics*, 20:500–510, 2001.

[135] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3:249–265, June 1987.

[136] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Percepti on and Navigation.* PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

[137] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, pages 46–57, 1989.

[138] S. Engelson. *Passive Map Learning and Visual Place Recognition.* PhD thesis, Department of Computer Science, Yale University, 1994.

[139] C. Fernandes, L. Gurvits, and Z. Li. Optimal nonholonomic motion planning for a falling cat. In Z. Li and J. Canny, editors, *Nonholonomic Motion Planning.* Kluwer Academic, 1993.

[140] C. Fernandes, L. Gurvits, and Z. Li. Near-optimal nonholonomic motion planning for a system of coupled rigid bodies. *IEEE Transactions on Automatic Control*, 30(3):450–463, Mar. 1994.

[141] R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[142] S. Fleury, P. Souères, J.-P. Laumond, and R. Chatila. Primitives for smoothing paths of mobile robots. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 832–839, 1993.

[143] S. Fleury, P. Souères, J.-P. Laumond, and R. Chatila. Primitives for smoothing mobile robot trajectories. *IEEE Transactions on Robotics and Automation*, 11:441–448, 1995.

[144] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. On differentially flat nonlinear systems. In *IFAC Symposium NOLCOS*, pages 408–412, 1992.

[145] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. Flatness and defect of nonlinear systems: Introductory theory and examples. *International Journal of Control*, 61(6):1327–1361, 1995.

[146] A. T. Fomenko and T. L. Kunii. *Topological Modeling for Visualization.* Springer-Verlag, Tokyo, 1997.

[147] M. Foskey, M. Garber, M. Lin, and D. Manocha. A voronoi-based hybrid motion planner. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[148] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo localization: Efficient position estimation for mobile robots. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1999.

[149] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8(3), 2000.

[150] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research (JAIR)*, 11:391–427, 1999.

[151] T. Fraichard and J.-M. Ahuactzin. Smooth path planning for cars. In *IEEE International Conference on Robotics and Automation*, pages 3722–3727, Seoul, Korea, 2001.

[152] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics*, 25(1):116–129, 2002.

[153] C. Früh and A. Zakhor. 3D model generation for cities using aerial photographs and ground level laser scans. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[154] R. Geraerts and M. Overmars. A comparative study of probabilistic roadmap planners. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, pages 43–58. Springer-Verlag, 2003.

[155] E. Gilbert, D. Johnson, and S. Keerthi. A fast procedure for computing distance between complex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation*, 4:193–203, 1988.

[156] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, New York, 1981.

[157] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *IEEE International Conference on Robotics and Automation*, pages 1718–1723, 1990.

[158] N. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Procedings F*, 140(2):107–113, 1993.

[159] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.

[160] P. Grandjean and A. Robert de Saint Vincent. 3-D modeling of indoor scenes by fusion of noisy range and stereo data. In *IEEE International Conference on Robotics and Automation*, 1989.

[161] F. Gravoit, S. Cambon, and R. Alami. asymov: a planner that deals with intricate symbolic and geometric problems. In *International Symposium on Robotics Research*, 2003. Book to appear.

[162] L. J. Guibas, C. Holleman, and L. E. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis- based sampling approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 254–260, 1999.

[163] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry and Applications*, 9(4/5):471–512, August/October 1999.

[164] V. Guillemin and A. Pollack, editors. *Differential Topology*. Prentice-Hall, Inc., New Jersey, 1974.

[165] K. Gupta and Z. Guo. Motion planning with many degrees of freedom: sequential search with backtracking. *IEEE Transactions on Robotics and Automation*, 6(11):897–906, 1995.

[166] L. Gurvits. Averaging approach to nonholonomic motion planning. In *IEEE International Conference on Robotics and Automation*, pages 2541–2546, 1992.

[167] J. Gutmann and D. Fox. An experimental comparison of localization methods continued. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[168] J.-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1998.

[169] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the IEEE Int. Symp. on Computational Intelligence in Robotics and Automation (CIRA)*, 1999.

[170] J.-S. Gutmann and C. Schlegel. AMOS: Comparison of scan matching approaches for self-localization in indoor environments. In *Proc. of the 1st Euromicro Workshop on Advanced Mobile Robots*. IEEE Computer Society Press, 1996.

[171] J.-S. Gutmann, T. Weigel, and B. Nebel. A fast, accurate, and robust method for self-localization in polygonal environments using laser-range-finders. *Advanced Robotics Journal*, 14(8):651–668, 2001.

[172] D. Hähnel, W. Burgard, D. Fox, and S. Thrun. A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. Submitted for publication.

[173] D. Hähnel, D. Schulz, and W. Burgard. Map building with mobile robots in populated environments. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[174] D. Halperin and M. Sharir. A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. *Discrete Computational Geometry*, 16:121–134, 1996.

[175] L. Han and N. M. Amato. A kinematics-based probabilistic roadmap for closed chain systems. In B. R. Donald, K. Lynch, and D. Rus, editors, *New Directions in Algorithmic and Computational Robotics*, pages 233–246. AK Peters, 2001.

[176] G. Heinzinger, P. Jacobs, J. Canny, and B. Paden. Time-optimal trajectories for a robot manipulator: A provably good approximation algorithm. In *IEEE International Conference on Robotics and Automation*, pages 150–156, 1989.

[177] G. Heinzinger and B. Paden. Bounds on robot dynamics. In *IEEE International Conference on Robotics and Automation*, pages 1227–1232, Scottsdale, Arizona, 1989.

[178] S. Hert, S. Tiwari, and V. Lumelsky. A Terrain-Covering Algorithm for an AUV. *Autonomous Robots*, 3:91–119, 1996.

[179] J. Hertzberg and F. Kirchner. Landmark-based autonomous navigation in sewerage pipes. In *Proc. of the First Euromicro Workshop on Advanced Mobile Robots*, 1996.

[180] H. Hirukawa, B. Mourrain, and Y. Papegay. A symbolic-numeric silhouette algorithm. In *Intelligent Robots and Systems*, pages 2358 – 2365, Nov 2000.

[181] C. Hofner and G. Schmidt. Path planning and guidance techniques for an autonomous mobile cleaning robot. *Robotics and Autonomous Systems*, 14:199–212, 1995.

[182] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *IEEE International Conference on Robotics and Automation*, pages 1408–1413, 2000.

[183] D. Hsu. *Randomized Single-Query Motion Planning In Expansive Spaces*. PhD thesis, Department of Computer Science, Stanford University, 2000.

[184] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, 2003.

[185] D. Hsu, L. E. Kavraki, J. C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In e. a. P. Agarwal, editor, *Robotics: The Algorithmic Perspective*, pages 141–154. A.K. Peters, Wellesley, MA, 1998.

[186] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21(3):233–255, 2002.

[187] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE International Conference on Robotics and Automation*, pages 2719–2726, 1997.

[188] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9(4/5):495–512, 1998.

[189] Y. Y. Huang, Z. L. Cao, and E. Hall. Region filling operations for mobile robot using computer graphics. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1607–1614, 1986.

[190] T. C. Hudson, M. C. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-COLLIDE: Accelerated collision detection for VRML. In R. Carey

and P. Strauss, editors, *VRML 97: Second Symposium on the Virtual Reality Modeling Language*, pages 119–125, New York City, NY, 1997. ACM Press.

[191] S. Iannitti and K. M. Lynch. Exact minimum control switch motion planning for the snakeboard. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[192] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1), 1998.

[193] A. Isidori. *Nonlinear Control Systems: An Introduction*. Springer-Verlag, 1985.

[194] P. Isto. A two-level search algorithm for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 2025–2031, 1997.

[195] P. Isto. Constructing probabilistic roadmaps with powerful local planning and path optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2323–2328, 2002.

[196] P. Jacobs, G. Heinzinger, J. Canny, and B. Paden. Planning guaranteed near-time-optimal trajectories for a manipulator in a cluttered workspace. In *International Workshop on Sensorial Integration for Industrial Robots: Architectures and Applications*, Zaragoza, Spain, 1989.

[197] P. Jacobs, G. Heinzinger, J. Canny, and B. Paden. Planning guaranteed near-time-optimal trajectories for a manipulator in a cluttered workspace. Technical Report RAMP 89-15, University of California, Berkeley, Engineering Systems Research Center, Sept. 1989.

[198] K. Janich. *Topology*. Spring-Verlag, New York, NY, 1984.

[199] R. Jarvis. Collision free trajectory planning using distance transforms. *Mech Eng Trans of the IE Aust*, ME10:197–191, 1985.

[200] P. Jensfelt and S. Kristensen. Active global localisation for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 17(5):748–760, Oct. 2001.

[201] X. Ji and J. Xiao. Planning motion compliant to complex contact states. *International Journal of Robotics Research*, 20(6):446–465, 2001.

[202] V. Jurdjevic. *Geometric Control Theory*. Cambridge University Press, 1997.

[203] V. Jurdjevic and H. J. Sussmann. Control systems on Lie groups. *Journal of Differential Equations*, 12:313–329, 1972.

[204] L. Kaelbling, A. Cassandra, and J. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[205] T. Kailath. *Linear Systems*. Prentice-Hall, 1980.

[206] R. Kalman. A new approach to linear filtering and prediction problems. *Trans. of the ASME, Journal of basic engineering*, 82:35–45, March 1960.

[207] I. Kamon, E. Rimon, and E. Rivlin. Tangentbug: A range-sensor based navigation algorithm. *Int. Journal of Robotics Research*, 17(9):934–953, 1998.

[208] I. Kamon, E. Rivlin, and E. Rimon. A new range-sensor based globally convergent navigation for mobile robots. In *IEEE Int'l. Conf. on Robotics and Automation*, Minneapolis, MN, April 1996.

[209] K. Kanazawa, D. Koller, and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *Proc. of the 11th Annual Conference on Uncertainty in AI (UAI)*, 1995.

[210] K. Kant and S. Zucker. Toward efficient trajectory planning: Path velocity decomposition. *International Journal of Robotics Research*, 5:72–89, 1986.

[211] L. E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford University, 1995.

[212] L. E. Kavraki, M. Kolountzakis, and J. C. Latombe. Analysis of probabilistic roadmaps for path planning. In *IEEE International Conference on Robotics and Automation*, pages 3020–3026, 1996.

[213] L. E. Kavraki, M. N. Kolountzakis, and J. C. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171, February 1998.

[214] L. E. Kavraki, F. Lamiraux, and C. Holleman. Towards planning for elastic objects. In P. Agrawal, L. E. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 313–325. A.K. Peters, 1998.

[215] L. E. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. Technical Report STAN-CS-93-1490, Dept. Comput. Sci., Stanford Univ., Stanford, CA, 1993.

[216] L. E. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for path planning. In *IEEE International Conference on Robotics and Automation*, pages 2138–2139, 1994.

[217] L. E. Kavraki and J. C. Latombe. Probabilistic roadmaps for robot path planning. In K. Gupta and A. P. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Challenges*, pages 33–53. John Wiley, West Sussex, England, 1998.

[218] L. E. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot motion planning. In *Proc. ACM Symp. on Theory of Computing*, pages 353–362, 1995.

[219] L. E. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. *Journal of Computer and System Sciences*, 57(1):50–60, August 1998.

[220] L. E. Kavraki, J. C. Latombe, and R. Wilson. On the complexity of assembly partitioning. *Information Processing Letters*, 48:229–235, 1993.

[221] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, June 1996.

[222] H. Keller. *Lectures on Numerical Methods in Bifurcation Problems*. Tata Institute of Fundamental Research, Bombay, India, 1987.

[223] S. D. Kelly and R. M. Murray. Geometric phases and robotic locomotion. *Journal of Robotic Systems*, 12(6):417–431, 1995.

[224] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5:90–98, 1986.

[225] R. Kindel, D. Hsu, J. C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *IEEE International Conference on Robotics and Automation*, pages 537–543, 2000.

[226] D. E. Kirk. *Optimal Control Theory*. Prentice-Hall Inc., 1970.

[227] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, O. E., and H. Matsubara. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, 1997.

[228] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[229] D. E. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11:412–442, 1990.

[230] S. Koenig and R. Simmons. A robot navigation architecture based on partially observable Markov decision process models. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, Cambridge, MA, 1998.

[231] Y. Koga, K. Kondo, J. Kuffner, and J. C. Latombe. Planning motions with intentions. *Computer Graphics (SIGGRAPH'94)*, pages 395–408, 1994.

[232] Y. Koga and J. C. Latombe. Experiments in dual-arm manipulation planning. In *IEEE International Conference on Robotics and Automation*, pages 2238–2245, 1992.

[233] Y. Koga and J. C. Latombe. On multi-arm manipulation planning. In *IEEE International Conference on Robotics and Automation*, pages 945–952, 1994.

[234] K. Kondo. Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration. *IEEE Transactions on Robotics and Automation*, 7:267–277, 1991.

[235] K. Konolige. Markov localization using correlation. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[236] J. Kuffner. Effective sampling and distance metrics for 3D rigid body path planning. In *IEEE International Conference on Robotics and Automation*, 2004.

[237] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots under obstacle and dynamic balance constraints. In *IEEE International Conference on Robotics and Automation*, pages 692–698, Seoul, Korea, May 2001.

[238] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots. In *International Symposium on Robotics Research*, 2003. Book to appear.

[239] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, pages 995–1001, 2000.

[240] B. Kuipers and Y. Byan. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[241] A. Ladd and L. E. Kavraki. Motion planning for knot untangling. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, pages 7–24. Springer-Verlag, 2002.

[242] A. M. Ladd and L. E. Kavraki. Measure theoretic analysis of probabilistic path planning. *IEEE Transactions on Robotics and Automation*, 20(2):229–242, 2004.

[243] G. Lafferriere and H. Sussmann. Motion planning for controllable systems without drift. In *IEEE International Conference on Robotics and Automation*, pages 1148–1153, Sacramento, CA, 1991.

[244] G. Lafferriere and H. J. Sussmann. A differential geometric approach to motion planning. In Z. Li and J. Canny, editors, *Nonholonomic Motion Planning*. Kluwer Academic, 1993.

[245] F. Lamiraux and L. Kavraki. Planning paths for elastic objects under manipulation constraints. *International Journal of Robotics Research*, 20(3):188–208, 2001.

[246] F. Lamiraux and J. P. Laumond. On the expected complexity of random path planning. In *IEEE International Conference on Robotics and Automation*, pages 3014–3019, 1996.

[247] F. Lamiraux and J.-P. Laumond. Smooth motion planning for car-like vehicles. *IEEE Transactions on Robotics and Automation*, 17(4):498–502, Aug. 2001.

[248] F. Lamiraux, S. Sekhavat, and J.-P. Laumond. Motion planning and control for Hilare pulling a trailer. *IEEE Transactions on Robotics and Automation*, 15(4):640–652, Aug. 1999.

[249] S. Land and H. Choset. Coverage path planning for landmine location. In *Third International Symposium on Technology and the Mine Problem*, Monterey, CA, April 1998.

[250] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of North Carolina at Chapel Hill, North Carolina, 1999.

[251] J. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[252] J.-C. Latombe. Personal communication.

[253] J. Laumond and R. Alami. A geometrical approach to planning manipulation tasks: The case of a circular robot and a movable circular object amidst polygonal obstacles. Report 88314, LAAS/CNRS, Toulouse, France, 1989.

[254] J.-P. Laumond. Controllability of a multibody mobile robot. *IEEE Transactions on Robotics and Automation*, 9(6):755–763, Dec. 1993.

[255] J.-P. Laumond. *Robot motion planning and control*. Springer, 1998.

[256] J.-P. Laumond, P. E. Jacobs, M. Taïx, and R. M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 10(5):577–593, Oct. 1994.

[257] S. LaValle, J. Yakey, and L. E. Kavraki. Randomized path planning for linkages with closed kinematics chains. *IEEE Transactions on Robotics and Automation*, 17(6):951–959, 2001.

[258] S. M. LaValle and M. S. Branicky. On the relationship between classical grid search and probabilistic roadmaps. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, pages 59–76. Springer-Verlag, 2002.

[259] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *IEEE International Conference on Robotics and Automation*, pages 473–479, 1999.

[260] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

[261] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. Lynch, and D. Rus, editors, *New Directions in Algorithmic and Computational Robotics*, pages 293–308. AK Peters, 2001.

[262] S. M. Lavalle, D. Lin, L. J. Guibas, J. C. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *IEEE International Conference on Robotics and Automation*, pages 1677–1682, 1997.

[263] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. *Computer Graphics*, 24(4):327–335, 1990.

[264] S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *IEEE International Conference on Robotics and Automation*, 2000.

[265] J. Leonard and H. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer Academic, Boston, MA, 1992.

[266] J. Leonard and H. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In J. Hollerbach and D. Koditschek, editors, *Proceedings of the Ninth International Symposium on Robotics Research*, Salt Lake City, Utah, 1999.

[267] J. J. Leonard and H. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems*, pages 1442–1447, May 1991.

[268] N. E. Leonard. Control synthesis and adaptation for an underactuated autonomous underwater vehicle. *IEEE Journal of Oceanic Engineering*, 20(3):211–220, July 1995.

[269] N. E. Leonard and P. S. Krishnaprasad. Motion control of drift-free, left-invariant systems on Lie groups. *IEEE Transactions on Automatic Control*, 40(9):1539–1554, Sept. 1995.

[270] P. Leven and S. Hutchinson. Real-time path planning in changing environments. *International Journal of Robotics Research*, 21(12):999–1030, Dec. 2002.

[271] P. Leven and S. Hutchinson. Using manipulability to bias sampling during the construction of probabilistic roadmaps. *IEEE Transactions on Robotics and Automation*, 19(6):1020–1026, Dec. 2003.

[272] A. D. Lewis. When is a mechanical control system kinematic? In *IEEE Conference on Decision and Control*, pages 1162–1167, Dec. 1999.

[273] A. D. Lewis. Simple mechanical control systems with constraints. *IEEE Transactions on Automatic Control*, 45(8):1420–1436, 2000.

[274] A. D. Lewis and R. M. Murray. Configuration controllability of simple mechanical control systems. *SIAM Journal on Control and Optimization*, 35(3):766–790, May 1997.

[275] A. D. Lewis and R. M. Murray. Configuration controllability of simple mechanical control systems. *SIAM Review*, 41(3):555–574, 1999.

[276] F. L. Lewis and V. L. Syrmos. *Optimal Control*. John Wiley and Sons, Inc., 1995.

[277] Z. Li and J. Canny. *Nonholonomic Motion Planning*. Kluwer Academic, 1993.

[278] K. Lian, L. Wang, and L. Fu. Controllability of spacecraft systems in a central gravitational field. *IEEE Transactions on Automatic Control*, 39(12):2426–2440, Dec. 1994.

[279] M. C. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision detection: Algorithms and applications. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 129–142. A K Peters, Wellesley, MA, 1997.

[280] S. R. Lindemann and S. M. LaValle. Incremental low-discrepancy lattice methods for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 2920–2927, 2003.

[281] G. Liu and Z. Li. A unified geometric approach to modeling and control of constrained mechanical systems. *IEEE Transactions on Robotics and Automation*, 18(4):574–587, Aug. 2002.

[282] Y. Liu and S. Arimoto. Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. *International Journal of Robotics Research*, 11(4):376–382, 1992.

[283] C. Lobry. Controllability of nonlinear systems on compact manifolds. *SIAM Journal on Control*, 12(1):1–4, 1974.

[284] I. Lotan, F. Schwarzer, D. Halperin, and J.-C. Latombe. Efficient maintenance and self-collision testing for kinematic chains. In *Proceedings of the 18th annual Symposium on Computational geometry*, pages 43–52. ACM Press, 2002.

[285] T. Lozano-Pérez. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, RA-3(3):224–238, 1987.

[286] T. Lozano-Perez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[287] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[288] V. Lumelsky, S. Mukhopadhyay, and K. Sun. Dynamic path planning in sensor-based terrain acquisition. *IEEE Transactions on Robotics and Automation*, 6(4):462–472, August 1990.

[289] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[290] K. M. Lynch. Controllability of a planar body with unilateral thrusters. *IEEE Transactions on Automatic Control*, 44(6):1206–1211, June 1999.

[291] K. M. Lynch and C. K. Black. Recurrence, controllability, and stabilization of juggling. *IEEE Transactions on Robotics and Automation*, 17(2):113–124, Apr. 2001.

[292] K. M. Lynch, N. Shiroma, H. Arai, and K. Tanie. Collision-free trajectory planning for a 3-DOF robot with a passive joint. *International Journal of Robotics Research*, 19(12):1171–1184, Dec. 2000.

[293] D. K. M. Ben-Or and J. Reif. The complexity of elementary algebra and geometry. *Journal of Computational Sciences*, 32:251–264, 1986.

[294] J. Marsden. *Elementary Classical Analysis*. W. H. Freeman and Company, New York, 1974.

[295] J. Marsden and T. Ratiu. *Introduction to Mechanics and Symmetry*. Springer-Verlag, New York, 1994.

[296] P. Martin, R. M. Murray, and P. Rouchon. Flat systems. In G. Bastin and M. Gevers, editors, *1997 European Control Conference Plenary Lectures and Mini-Courses*. 1997.

[297] S. Martinez, J. Cortés, and F. Bullo. A catalog of inverse-kinematics planners for underactuated systems on matrix Lie groups. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[298] M. T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, 2001.

[299] P. Maybeck. The Kalman filter: An introduction to concepts. In *Autonomous Robot Vehicles*. Springer verlag, 1990.

[300] M. B. Milam, K. Mushambi, and R. M. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *IEEE International Conference on Decision and Control*, 2000.

[301] J. Milnor. *Morse Theory*. Princeton University Press, Princeton, NJ, 1963.

[302] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, 1998.

[303] M. Moll and L. E. Kavraki. Path planning for minimal energy curves of constant length. In *IEEE International Conference on Robotics and Automation*, pages 2826–2831, 2004.

[304] M. Montemerlo and S. Thrun. Simultaneous localization and mapping problem with unknown data association using FastSLAM. In *IEEE International Conference on Robotics and Automation*, 2003.

[305] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2002.

[306] M. Montemerlo, S. Thrun, and W. Whittaker. Conditional particle filters for simultaneous mobile robot localization and people tracking. In *IEEE International Conference on Robotics and Automation*, 2002.

[307] M. Morales, S. Rodriguez, and N. M. Amato. Improving the connectivity of prm roadmaps. In *IEEE International Conference on Robotics and Automation*, pages 4427–4432, 2003.

[308] H. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.

[309] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *IEEE International Conference on Robotics and Automation*, 1985.

[310] J. J. Moré and S. J. Wright. *Optimization Software Guide*. SIAM, Philadelphia, PA, 1993.

[311] K. A. Morgansen. *Temporal patterns in learning and control*. PhD thesis, Harvard University, 1999.

[312] K. A. Morgansen, P. A. Vela, and J. W. Burdick. Trajectory stabilization for a planar carangiform robot fish. In *IEEE International Conference on Robotics and Automation*, 2002.

[313] K. Murphy. Bayesian map learning in dynamic environments. In *Neural Info. Proc. Systems (NIPS)*, 1999.

[314] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[315] R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. In *ASME Int Mech Eng Congress and Expo*, 1995.

[316] R. M. Murray and S. S. Sastry. Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, 38(5):700–716, 1993.

[317] Y. Nakamura, T. Suzuki, and M. Koinuma. Nonlinear behavior and control of a nonholonomic free-joint manipulator. *IEEE Transactions on Robotics and Automation*, 13(6):853–862, 1997.

[318] P. Newman, J. Leonard, J. Neira, and J. Tardós. Explore and return: Experimental validation of real time concurrent mapping and localization. In *IEEE International Conference on Robotics and Automation*, 2002.

[319] C. Nielsen and L. E. Kavraki. A two level fuzzy PRM for manipulation planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1716–1722, Japan, 2000.

[320] C. Nissoux, T. Simeon, and J. Laumond. Visibility based probabilistic roadmaps. *Advanced Robotics Journal*, 14(6), 2000.

[321] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Verlag, 1999.

[322] I. Nourbakhsh, R. Powers, and S. Birchfield. DERVISH an office-navigating robot. *AI Magazine*, 16(2), 1995.

[323] C. Ó'Dúnlaing and C. Yap. A "retraction" method for planning the motion of a disc. *Algorithmica*, 6:104–111, 1985.

[324] M. Ollis and A. Stentz. First results in vision-based crop line tracking. In *IEEE International Conference on Robotics and Automation*, 1996.

[325] J. P. Ostrowski and J. W. Burdick. The geometric mechanics of undulatory robotic locomotion. *International Journal of Robotics Research*, 17(7):683–701, July 1998.

[326] J. P. Ostrowski, J. P. Desai, and V. Kumar. Optimal gait selection for nonholonomic locomotion systems. *International Journal of Robotics Research*, 19(3):225–237, Mar. 2000.

[327] M. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, Oct. 1992.

[328] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In K. Goldberg, D. Halperin, J. C. Latombe, and R. Wilson, editors, *Algorithmic Foundations of Robotics (WAFR)*, pages 19–37. A. K. Peters, Ltd, 1995.

[329] R. Parr and A. Eliazar. DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[330] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1988.

[331] F. Pfeiffer and R. Johanni. A concept for manipulator trajectory planning. *IEEE Journal of Robotics and Automation*, RA-3(2):115–123, 1987.

[332] J. Phillips, L. Kavraki, and N. Bedrossian. Spacecraft rendezvous and docking with real-time, randomized optimization. In *AIAA Guidance, Navigation, and Control*, 2003.

[333] A. Piazza, M. Romano, and C. G. L. Bianco. G^3-splines for the path planning of wheeled mobile robots. In *European Control Conference*, 2003.

[334] C. Pisula, K. Hoff, M. Lin, and D. Manocha. Randomized path planning for a rigid body based on hardware accelarated Voronoi sampling. In B. R. Donald, K. Lynch, and D. Rus, editors, *New Directions in Algorithmic and Computational Robotics*. AK Peters, 2001.

[335] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The Mathematical Theory of Optimal Processes*. Interscience Publishers, 1962.

[336] C. Pradalier, J. Hermosillo, C. Koike, C. Braillon, P. P.Bessière, and C. Laugier. Safe and autonomous navigation for a car-like robot among pedestrian. In *IARP Int. Workshop on Service, Assistive and Personal Robots*, 2003.

[337] F. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985. p198-257.

[338] S. Quinlan. Efficient distance computation between nonconvex objects. In *IEEE International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[339] N. Rao, N. Stolzfus, and S. Iyengar. A retraction method for learned navigation in unknown terrains for a circular robot. *IEEE Transactions on Robotics and Automation*, 7:699–707, October 1991.

[340] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[341] J. Reif. Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[342] J. H. Reif and H. Wang. Nonuniform discretization for kinodynamic motion planning and its applications. *SIAM Journal of Computing*, 30(1):161–190, 2000.

[343] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, October 1992.

[344] T. Röfer. Using histogram correlation to create consistent laser scan maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[345] H. Rohnert. Shortest path in the plane with convex polygonal obstacles. *Information Processing Letters*, 23:71–76, 1986.

[346] G. Sánchez and J. C. Latombe. On delaying collision checking in prm planning : Application to multi-robot coor dination. *International Journal of Robotics Research*, 21(1):5–26, 2002.

[347] S. S. Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Springer-Verlag, New York, 1999.

[348] D. H. Sattinger and O. L. Weaver. *Lie Groups and Algebras with Applications to Physics, Geometry, and Mechanics*. Springer-Verlag, 1986.

[349] A. Scheuer and T. Fraichard. Collision-free and continuous-curvature path planning for car-like robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1304–1311, Osaka, Japan, 1997.

[350] B. Schiele and J. Crowley. A comparison of position estimation techniques using occupancy grids. In *IEEE International Conference on Robotics and Automation*, 1994.

[351] B. Schutz. *Geometrical methods of mathematical physics*. Cambridge University Press, 1980.

[352] J. T. Schwartz and M. Sharir. On the piano movers' problem: II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.

[353] J. T. Schwartz and M. Sharir. On the piano movers' problem: V. the case of a rod moving in three-dimensional space amidst polyhedral obstacles. *Communications on Pure and Applied Mathematics*, 37:815–848, 1984.

[354] J. T. Schwartz and M. Sharir. A survey of motion planning and related geometric algorithms. *Artificial Intelligence.*, 37:157–169, 1988.

[355] F. Schwarzer, M. Saha, and J. Latombe. Exact collision checking of robot paths. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, pages 25–42. Springer-Verlag, 2002.

[356] S. Sekhavat and J.-P. Laumond. Topological property of trajectories computed from sinusoidal inputs for nonholonomic chained form systems. In *IEEE International Conference on Robotics and Automation*, pages 3383–3388, 1996.

[357] S. Sekhavat and J.-P. Laumond. Topological property for collision-free nonholonomic motion planning: the case of sinusoidal inputs for chained form systems. *IEEE Transactions on Robotics and Automation*, 14(5):671–680, Oct. 1998.

[358] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. H. Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *International Journal of Robotics Research*, 17(8):840–857, Aug. 1998.

[359] J.-P. Serre. *Lie Algebras and Lie Groups*. W. A. Benjamin, New York, 1965.

[360] J. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, UK, 1999.

[361] H. Shatkay and L. Kaelbling. Learning topological maps with weak local odometric information. In *Proceedings of IJCAI-97*. IJCAI, Inc., 1997. 1997.

[362] Z. Shiller and S. Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics and Automation*, 7(6):785–797, Dec. 1991.

[363] Z. Shiller and H.-H. Lu. Computation of path constrained time optimal motions with dynamic singularities. *ASME Journal of Dynamic Systems, Measurement, and Control*, 114:34–40, Mar. 1992.

[364] K. G. Shin and N. D. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, 30(6):531–541, June 1985.

[365] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[366] A. Singh, J. C. Latombe, and D. Brutlag. A motion planning approach to flexible ligand binding. In *Intelligent Systems for Molecular Biology*, pages 252–261, 1999.

[367] J.-J. E. Slotine and H. S. Yang. Improving the efficiency of time-optimal path-following algorithms. *IEEE Transactions on Robotics and Automation*, 5(1):118–124, Feb. 1989.

[368] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.

[369] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*. Springer Verlag, 1990.

[370] E. Sontag. Gradient techniques for systems with no drift: A classical idea revisited. In *IEEE International Conference on Decision and Control*, pages 2706–2711, 1993.

[371] E. D. Sontag. Control of systems without drift via generic loops. *IEEE Transactions on Automatic Control*, 40(7):1210–1219, July 1995.

[372] O. J. Sørdalen. Conversion of a car with n trailers into a chained form. In *IEEE International Conference on Robotics and Automation*, pages 1382–1387, 1993.

[373] P. Souères and J.-D. Boissonnat. Optimal trajectories for nonholonomic mobile robots. In J.-P. Laumond, editor, *Robot Motion Planning and Control*. Springer, 1998.

[374] P. Souères and J.-P. Laumond. Shortest paths synthesis for a car-like robot. *IEEE Transactions on Automatic Control*, 41(5):672–688, May 1996.

[375] R. F. Stengel. *Optimal control and estimation*. Dover, New York, 1994.

[376] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. *International Journal of Robotics and Automation*, 10, 1995.

[377] G. Strang. *Linear Algebra and Its Applications*. Orlando: Academic Press, 1980.

[378] H. Sussmann. A continuation method for nonholonomic path-finding problems. In *IEEE International Conference on Decision and Control*, pages 2718–2723, 1993.

[379] H. J. Sussmann. A general theorem on local controllability. *SIAM Journal on Control and Optimization*, 25(1):158–194, Jan. 1987.

[380] H. J. Sussmann and W. Tang. Shortest paths for the Reeds-Shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control. Technical Report SYCON-91-10, Rutgers University, 1991.

[381] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal of Computing*, 21(5):863–888, October 1992.

[382] P. Švestka. A probabilistic approach to motion planning for car-like robots. Technical Report RUU-CS-93-18, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, 1993.

[383] P. Švestka and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *IEEE International Conference on Robotics and Automation*, pages 1631–1636, 1995.

[384] P. Švestka and J. Vleugels. Exact motion planning for tractor-trailer robots. In *IEEE International Conference on Robotics and Automation*, pages 2445–2450, 1995.

[385] K. R. Symon. *Mechanics*. Addison-Wesley, 1971.

[386] X. Tang, B. Kirkpatrick, S. Thomas, G. Song, and N. M. Amato. Using motion planning to study rna folding kinetics. In *International Conference on Research in Computational Molecular Biology*, 2004.

[387] M. Teodoro, G. N. Phillips, and L. E. Kavraki. Molecular docking: A problem with thousands of degrees of freedom. In *IEEE International Conference on Robotics and Automation*, pages 960–966, 2001.

[388] J. Thorpe. *Elementary Topics in Differential Geometry*. Springer-Verlag, 1985.

[389] S. Thrun. Exploration and model building in mobile robot domains. In *Proc. of the IEEE International Conference on Neural Networks*, 1993.

[390] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–363, 2001.

[391] S. Thrun. Learning occupancy grids with forward sensor models. *Autonomous Robots*, 2002.

[392] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A second generation mobile tour-guide robot. In *IEEE International Conference on Robotics and Automation*, 1999.

[393] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, to appear.

[394] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *IEEE International Conference on Robotics and Automation*, 2000.

[395] S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning and Autonomous Robots (joint issue)*, 31(1-3):29–53, 1998.

[396] S. Thrun, J.-S. Gutmann, D. Fox, W. Burgard, and B. Kuipers. Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1998.

[397] D. Tilbury, R. Murray, and S. Sastry. Trajectory generation for the n-trailer problem using Goursat normal form. In *IEEE International Conference on Decision and Control*, 1993.

[398] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools: JGT*, 2(4):1–14, 1997.

[399] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools: JGT*, 4(2):7–25, 1999.

[400] P. Vela and J. W. Burdick. Control of biomimetic locomotion via averaging theory. In *IEEE International Conference on Robotics and Automation*, 2003.

[401] P. A. Vela, K. A. Morgansen, and J. W. Burdick. Underwater locomotion from oscillatory shape deformations. In *IEEE International Conference on Decision and Control*, 2002.

[402] G. Weiß, C. Wetzler, and E. von Puttkamer. Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 595–601, 1994.

[403] J. T. Wen. Control of nonholonomic systems. In W. S. Levine, editor, *The Control Handbook*, pages 1359–1368. CRC Press, 1996.

[404] S. Wilmarth, N. M. Amato, and P. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of th e free space.

In *IEEE International Conference on Robotics and Automation*, pages 1024–1031, 1999.

[405] R. Wilson and J. C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71:371–396, 1995.

[406] R. H. Wilson, L. E. Kavraki, J. C. Latombe, and T. Lozano-Pérez. Two-handed assembly sequencing. *International Journal of Robotics Research*, 14:335–350, 1995.

[407] B. Yamauchi and P. Langley. Place recognition in dynamic environments. *Journal of Robotic Systems*, 14(2):107–120, 1997.

[408] T. Yoshikawa. Manipulability of robotic mechanisms. *International Journal of Robotics Research*, 4(2):3–9, Apr. 1985.

[409] M. Zhang and L. E. Kavraki. A new method for fast and accurate derivation of molecular conformations. *Journal of Chemical Information and Computer Sciences*, 42(1):64–70, 2002.