Motion Planning: Recent Developments

Héctor H. González-Baños[1], *David Hsu*[2], *Jean-Claude Latombe*[3]

[1]*Honda Research Institute USA Inc., USA*

hgonzalez@hra.com

[2]*Department of Computer Science*

National University of Singapore, Singapore

dyhsu@comp.nus.edu.sg

[3]*Department of Computer Science*

Stanford University, USA

latombe@cs.stanford.edu

1 Introduction

A key trait of an autonomous robot is the ability to plan its own motion in order to accomplish specified tasks. Often, the objective of motion planning is to change the state of the world by computing a sequence of admissible motions for the robot. For example, in the path planning problem, we compute a collision-free path for a robot to go from an initial position to a goal position among static obstacles. This is the simplest type of motion planning problems; yet it is

provably hard computationally [64]. Sometimes, instead of changing the state of the world, our objective is to maintain a set of constraints on the state of the world (*e.g.*, following a target and keeping it in view), or to achieve a certain state of knowledge about the world (*e.g.*, exploring and mapping an unknown environment).

Ideally the robot achieves its objectives despite the many possible motion constraints internal or external to the robot. Traditionally, motion planning emphasizes a single *external* constraint: physical obstacles in the environment. This is actually the only constraint considered in path planning. However, real robots have inherent mechanical limitations, such as the non-holonomic constraints that prevent wheeled robots from moving sideways. Robots may also be constrained by sensor limitations, such as obstacles blocking the views of cameras. These internal constraints are important, but taking them into account further complicates motion planning.

In recent years, random sampling has emerged as a powerful approach for motion planning. It is computationally efficient and relatively simple to implement. Its development was originally driven by the need to plan motions for robots with many degrees of freedom (dofs), such as co-operating manipulator arms. However, we will downplay this aspect in this chapter. Instead, our main goal is to show how random sampling, combined with geometric and physical insights, can effectively handle motion constraints resulting from robots' mechanical and sensor limitations.

We start with an overview of path planning and proceed to the random-sampling approach to path planning (Section 2). Next, we focus on motion planning under two types of *internal* constraints: kinematic, dynamic constraints (Section 3) and visibility constraints (Section 4). We also briefly touch on the effect of uncertainty on motion planning (Section 5).

2 Path Planning

In path planning, we are given a complete description of the geometry of a robot and a static environment populated with obstacles. Our goal is to find a collision-free path for the robot to move from an initial position and orientation to a goal position and orientation.

Although path planning algorithms differ greatly in details, most of them follow a common framework (Figure 1). The first step is to map a robot, which may have complex geometric shape, to a *point* in a new, abstract space, called the *configuration space* [53], This mapping transforms the original problem to that of path planning for a moving point. Next we discretize the continuous configuration space and construct a graph that represents the connectivity of the space. Finally we search this graph to find a path for the robot to reach the goal. If no path is found, sometimes we may repeat the process by refining the discretization and searching for the path again.

An important consideration for path planning algorithms is *completeness*. A path planning algorithm is *complete*, if it finds a path whenever one exists and reports none exists otherwise. However, achieving completeness is often computationally intractable. In practice, we have to trade-off some amount of completeness for increased computational efficiency.

In this section, we first present the concept of configuration space (Section 2.1). Next, we briefly describe some early approaches to path planning (Section 2.2), before focusing on how the random-sampling approach works in this relatively simple setting (Section 2.3).

2.1 Configuration Space

The *configuration* of a robot is a set of parameters that uniquely determine the position of every point in the robot. For example, the configuration of a mobile robot is usually its position (x, y) and orientation θ for $\theta \in [-\pi, \pi)$. The configuration of an articulated robot manipulator is usually a list of joint angles $(\theta_1, \theta_2, \ldots)$.

Suppose that the configuration of a robot consists of d parameters. It can be then regarded as a point in a d-dimensional space $\mathcal{C}$, called the configuration space. A configuration q is *free*, if the robot placed at q does not collide with the obstacles or with itself. We define the *free space* $\mathcal{F}$ to be the subset of all free configurations in $\mathcal{C}$, and define the obstacle space $\mathcal{B}$ to be the complement of $\mathcal{F}$: $\mathcal{B} = \mathcal{C} \backslash \mathcal{F}$. See Figure 2*b* for an illustration.

For a robot that only translates in the plane, we can construct $\mathcal{C}$ explicitly by computing the

Minkowski difference of the robot and the obstacles. Intuitively, we can think of the computation as "growing" the obstacles by the shape of the robot and shrinking the robot to a point (Figure 2). In general, a mobile robot not only translates, but also rotates. In this case, we compute slices of $\mathcal{C}$ with the robot in various fixed orientations and then stack and stitch these slices together. Computing $\mathcal{C}$ exactly is also possible, though somewhat more complicated [2].

For high-dimensional configuration spaces, explicitly constructing $\mathcal{C}$ is difficult. Instead, we represent $\mathcal{C}$ implicitly by a function CLEARANCE : $\mathcal{C} \mapsto$ R, which maps a configuration $q \in \mathcal{C}$ to the distance between a robot at q and the obstacles. If CLEARANCE(q) returns 0, then q is in collision. An efficient implementation of this function can be achieved with hierarchical collision detection or distance computation algorithms [51].

Whether represented explicitly or implicitly, the configuration space encodes the key information of whether a robot at a particular configuration is in collision with obstacles or not. We can thus state the path planning problem formally in the configuration space as follows.

Problem 2.1 (path planning) *Given an initial configuration q_{init} and a goal configuration q_{goal}, find a path in the free space $\mathcal{F}$ between q_{init} and q_{goal}.*

In essence, the robot becomes a point in $\mathcal{C}$, and the path planning problem for the robot becomes that of finding a path for a moving point in $\mathcal{F}$. This transformation does not change the problem in any way, but it is often easier to think about the motion of a point than that of a robot with complex geometric shape. It also makes the problem formulation cleaner mathematically, especially when other constraints, in addition to physical obstacles, are considered (see Section 3).

2.2 Early Approaches

Path planning is fundamentally a question about the connectivity of $\mathcal{F}$: is there a path in $\mathcal{F}$ that connects two given configurations q_{init} and q_{goal}? To answer this question, a path planning algorithm usually discretizes $\mathcal{F}$ and computes a graph that represents its connectivity. It then searches this graph for a suitable path. The first step, constructing the connectivity graph, is the key and

is where algorithms differ. The second step, graph search, is accomplished with standard graph-search techniques, such as the Dijkstra's algorithm or the A* algorithm.

There are three general approaches for path planning: roadmap, cell decomposition, and potential field. They differ in the connectivity graphs constructed and their representations. These differences were important a decade ago, when computers were much slower and the differences could affect the computational cost greatly even for path planning in simple 2-D configuration spaces. With the advances in computer hardware, these differences are much less important today. All three approaches can solve path planning problems in 2-D configuration spaces in a fraction of a second on a modern PC. What is relevant today is whether an approach scales up for configuration spaces of high dimensions (six or more). Unfortunately none of them really does in their original forms. In the following, we give selected examples of the three approaches in 2-D configuration spaces, for the purpose of comparison with the random-sampling approach to be presented in the next sub-section. See [42] for a complete survey of these approaches.

Roadmap. The roadmap approach captures the connectivity of $\mathcal{F}$ in a network G of 1-D curves, called the *roadmap*. Once G is constructed, the robot is restricted to move along the curves in G. It appears that such a restriction may affect the robot's ability to find a collision-free path to the goal. However, a good roadmap has the property that there is a collision-free path in $\mathcal{C}$ between two configurations if and only if there is a collision-free path using only the curves represented in G. Algorithms that produce such roadmaps are clearly complete.

A classic example of the roadmap approach is the visibility graph algorithm [58], which applies mainly to 2-D configuration spaces with polygonal obstacles. It captures the connectivity of $\mathcal{C}$ in a visibility graph $G_{\rm vis}$ (Figure 3). The nodes of $G_{\rm vis}$ are the vertices of polygonal obstacles in $\mathcal{C}$, plus $q_{\rm init}$ and $q_{\rm goal}$. There is an edge between two nodes in $G_{\rm vis}$ if the straight-line path between the two nodes does not intersect the interior of the obstacles. The visibility graph can be computed in $O(n^2 \lg n)$ time using a simple rotational sweep-line algorithm [15], where n is the total number of vertices in the polygonal obstacles. After constructing $G_{\rm vis}$, we can find the shortest path between

q_{init} and q_{goal} by applying the Dijkstra's algorithm to G_{vis}. Furthermore, one can prove that the shortest path in G_{vis} is also the shortest among all possible paths in $\mathcal{F}$ between q_{init} and q_{goal}. This is the main strength of the visibility graph algorithm. However, it produces paths that graze the obstacles and thus bring the robot dangerously close to the obstacles, which is undesirable in practice.

An alternative is the Voronoi diagram algorithm, which captures the connectivity of $\mathcal{F}$ in the Voronoi diagram of $\mathcal{F}$ [59]. By following the curves in the Voronoi diagram, a robot stays as far away from the obstacles as possible, a clear advantage over the visibility graph algorithm. The Voronoi diagram can be computed in $O(n \lg n)$ time, which is also more efficient.

In 2-D polygonal configuration spaces, both the visibility graph and the Voronoi diagram capture the connectivity of the space exactly: there is a collision-free path in $\mathcal{C}$ between two given configurations if and only if there is such a path in the corresponding graphs. So both algorithms are complete for 2-D polygonal configuration spaces.

Cell decomposition. The cell decomposition approach first divides a robot's free space into simple, canonical regions called *cells*. Cells are usually convex so that it takes constant time to compute a path between any two configurations within a cell. We then construct a graph G_{cell} to capture the connectivity of $\mathcal{F}$, just as the roadmap algorithms do. The nodes of G_{cell} are the cells. There is an edge between two nodes if the corresponding cells are adjacent to each other.

The simplest cell decomposition is a grid with a fixed resolution (Figure 4*a*). To find a path between q_{init} and q_{goal}, we locate the two cells containing q_{init} and q_{goal}, respectively, and search for a path in G_{cell} between the two corresponding nodes. The result is a sequence of adjacent free cells that form a channel of free space between q_{init} and q_{goal}. A main advantage of this algorithm is the ease of implementation, giving rise to its great popularity in motion planning of mobile robots. However, its guarantee of completeness is weaker: it finds a path when one exists, only if the resolution of the grid is fine enough. Thus we say that the algorithm is only *resolution-complete*.

A more severe disadvantage of this algorithm is the grid size. If each dimension of a d-

dimensional configuration space is discretized into n intervals, we end up with $O(n^d)$ cells in total. This becomes prohibitively expensive to store and process, as d grows. To reduce the total number of cells, one possibility is to start with a coarse grid and refine the grid locally when necessary. This leads to a data structure similar to quad-tree or oct-tree (see [42] for more details). Another possibility is to analyze the input data carefully and use critical geometric features—such as the vertices or edges of polygonal obstacles—as a basis for discretizing the space, in order to avoid creating unnecessarily small cells. As an example, consider the triangulation algorithm [15], which divides the free space into triangles using the vertices of polygonal obstacles (Figure 4*b*). When there are a small number of simple obstacles, a triangulation contains much fewer cells than a grid with a reasonable resolution.

Potential field. The potential field approach [37] appears of a somewhat different nature from the previous two. It does not build a connectivity graph explicitly. Instead, it constructs an artificial potential function over $\mathcal{F}$ to guide a robot towards the goal. The potential function $U(q)$, which depends on the current configuration q of the robot, consists of an attractive component and a repulsive component: $U(q) = U_{\rm a}(q)+U_{\rm r}(q)$. The attractive potential $U_{\rm a}(q)$ pulls the robot towards the goal. The repulsive potential $U_{\rm r}(q)$ pushes the robot away from obstacles. The robot moves towards the goal, which is usually the global minimum of $U(q)$, by following the negated gradient of $U(q)$. One important advantage of this approach is that it computes not just a single path, but a feedback control strategy. The potential function $U(q)$ specifies the motion of the robot at any arbitrary configuration $q \in \mathcal{C}$. So the approach is more robust against control and sensing errors. It is also quite efficient. However, the potential field approach, which is based on steepest-descent optimization, suffers from the local minima problem: the robot may be trapped in a local minimum of $U(q)$ without reaching the global minimum, *i.e.*, the goal. The problem cannot be eliminated in general, but can be alleviated by constructing better potential functions with few local minima or executing random moves to help the robot escape from the local minima [42].

In some implementations, the potential function is represented on a grid. Such a potential field

algorithm is closely related to cell decomposition with a fixed-resolution grid. We can think of the potential function as a heuristic function for graph search on a grid.

2.3 Random Sampling

Even for a mobile robot, the dimensionality of its configuration space, $\dim(\mathcal{C})$, sometimes becomes quite high. The position and orientation of a mobile robot operating in the plane can typically be specified by three parameters (x, y, θ), but many mobile robots are wheeled differential-drive systems subject to non-holonomic or dynamic constraints. To represent these constraints, we may need to consider the velocities $(\dot{x}, \dot{y}, \dot{\theta})$ in addition to (x, y, θ), resulting in a 6-D space. If there are multiple robots cooperating in the same environment, $\dim(\mathcal{C})$ becomes even higher. As one expects, path planning becomes increasingly difficult as $\dim(\mathcal{C})$ grows.

During the past decade, random sampling has emerged as a powerful tool for path planning in high-dimensional configuration spaces. Algorithms based on random sampling, *e.g.*, the probabilistic roadmap (PRM) planners, are both efficient and simple to implement. They have solved path planning problems for multiple robots with dozens of dofs [65]. Although these algorithms are originally intended for robot manipulators with many dofs, the configuration space framework allows us to use them for mobile robots equally well.

As the name suggests, a PRM planner uses the roadmap approach. It tries to build a network of 1-D curves that captures the connectivity of $\mathcal{F}$. Compared with the classic roadmap algorithms presented in the previous sub-section, the main difference is that the nodes of a probabilistic roadmap are free configurations, sampled randomly according to a suitable probability distribution.

There are two main classes of random-sampling algorithms. The first class pre-computes a roadmap so that multiple planning queries in the same static environment can then be processed quickly. The second class performs no pre-computation and builds a small roadmap on the fly in order to process a single query as fast as possible. The latter scenario occurs if environments change frequently and pre-computation is not feasible. We refer to the first class as *multi-query*

planning, and the second class as *single-query* planning.

Multi-query planning. In multi-query planning, we proceed in two stages. The first stage is pre-computation, whose objective is to compute a roadmap G that captures the connectivity of $\mathcal{F}$ as accurately as possible in a reasonable amount of time. We sample $\mathcal{C}$ at random according to a suitable probability distribution π and retain the free configurations, called *milestones*, as nodes in G. Let LINK(q, q') denote a function that returns true if two milestones q and q' can be connected by a collision-free, straight-line path. We insert an edge in G between two milestones q and q' if LINK(q, q') returns true. Algorithm 1 below shows the main steps of this stage. The second stage is query processing. Each query asks for a collision-free path connecting q_{init} and q_{goal}. We first find two milestones q'_{init} and q'_{goal} in G such that q_{init} (q_{goal}, respectively) and q'_{init} (q'_{goal}, respectively) can be connected by a collision-free path. We then search for a path in G between q'_{init} and q'_{goal}.

Algorithm 1 Roadmap construction for multi-query PRM planning.

```
1: loop
2:    Pick q from C at random with probability π(q).
3:    if CLEARANCE(q) > 0 then
4:       Insert q into the roadmap G as a milestone.
5:       for every milestone q' ∈ G such that q' ≠ q do
6:          if LINK(q, q') returns TRUE then
7:             Insert an edge into G between q and q'.
```

The key issue in constructing probabilistic roadmaps is the sampling distribution for generating milestones. The first PRM planner uses a straightforward uniform distribution, followed by an enhancement step to increase sampling density in critical regions [36]. See Figure 5 for an example. The success of the first PRM planner led to intensive research. Many different sampling strategies for PRM planning have been proposed [1, 8, 22, 26, 27, 28, 50, 70, 77]. See [11, Chapter 7] for a survey. Most of them try to increase the sampling density inside narrow passages, which are small regions critical for capturing the connectivity of $\mathcal{F}$ well.

Another important issue for PRM planners is the representation of $\mathcal{C}$. The configuration space $\mathcal{C}$ is generally represented implicitly in PRM planning. In Algorithm 1, CLEARANCE(q) determines

whether q is collision-free, and LINK(q, q') determines whether there is a collision-free, straight-line path between q and q'. Both can be implemented efficiently using hierarchical bounding volume representation [51, 66].

Single-query planning. In contrast to multi-query planning, there is no pre-computation in the single-query setting. Instead, we construct a small roadmap on the fly to answer a single query. We sample only the connected components of $\mathcal{F}$ that contain either q_{init} or q_{goal} [30, 46]. The reason is that although $\mathcal{F}$ may contain several connected components, at most *two* of them, which contain q_{init} or q_{goal}, are relevant to the query being processed. It is clearly undesirable to construct a roadmap for the entire space. The roadmap for the single-query setting typically consists of two trees rooted at q_{init} and q_{goal} respectively (Figure 6). We expand the two trees by sampling new milestones at random from $\mathcal{C}$ and inserting them into the trees as milestones, until the two trees "meet", *i.e.*, a milestone in one tree is connected to a milestone in the other.

The two trees are expanded in an identical way. To add a new milestone to a tree T, we pick at random an existing milestone q in T with probability $\pi_T(q)$ and sample a new free configuration q' at random from the neighborhood of q with probability $\pi_q(q')$. If there is a straight-line path between q and q', then q' is inserted into T as a milestone along with an edge between q and q'. In contrast to Algorithm 1, a new configuration is inserted into T only if it can be connected to some existing milestone in T. So by construction, there is a path between the root of T and every milestone in T. The pseudocode in Algorithm 2 sketches out the algorithm for building a tree rooted at a given configuration.

Algorithm 2 Building a tree T rooted at configuration q_0.

```
1: Insert q_0 into T.
2: loop
3:    Pick an existing milestone q from T with probability π_T(q).
4:    Sample a new configuration q' at random from the neighborhood of q with probability π_q(q').
5:    if CLEARANCE(q) > 0 and LINK(q, q') returns TRUE then
6:       Insert q into T along with an edge between q and q'.
```

In Algorithm 2, we must avoid oversampling any region of $\mathcal{F}$, especially around q_{init} and q_{goal}.

Ideally we would like the milestones to eventually distribute rather uniformly over the connected components containing q_{init} or q_{goal}. Two common ways to achieve this are the *expansive space tree* (EST) [30] and the *rapidly-exploring random tree* (RRT) [46]. EST assigns every milestone q in T a weight that measures how densely the neighborhood of q has already been sampled. We then pick an existing milestone q with a suitable distribution $\pi_T(q)$ (line 3) so that low-density neighborhoods are more likely to be sampled. RRT uses a target distribution, *e.g.*, the uniform distribution, and pick q so that the final distribution of milestones are close to the target distribution.

Another interesting idea for single-query planning is to delay executing LINK, an expensive operation, until it becomes necessary [7, 65].

Probabilistic completeness. In general, path planning algorithms based on random sampling cannot detect that no path exists. We must explicitly set the maximum number of milestones to be sampled. We may also try to estimate how well $\mathcal{C}$ has been sampled and terminate the algorithm if $\mathcal{C}$ has been sampled adequately and no path has been found. Because of this, these algorithms are not complete. Instead they can only guarantee *probabilistic completeness*: a path planning algorithm is *probabilistically complete* if it finds a path with high probability when one exists. Probabilistic completeness provides a guarantee of performance only if a solution path exists. No assurance is implied, if there is no path. It can be shown that under reasonable geometric assumptions on the configuration space, both the multi-query and the single-query algorithms with suitable sampling distributions are probabilistically complete with exponentially fast convergence rate [30, 35, 39, 73].

Advantages of random sampling. The success of random sampling in path planning results from several factors.

- It can handle high-dimensional configuration spaces efficiently.
- It is easy to implement, partly due to the availability of good programming libraries for collision checking and pseudo-random number generation.

- It benefits from a probabilistic framework, which provides powerful tools for designing new sampling strategies and analysis techniques.
- It is difficult for an adversary to construct worst-case input, because of the random decisions made by the algorithm, thus improving the robustness of the algorithm on the average.

3 Motion Planning under Kinematic and Dynamic Constraints

Path planning is a purely geometric problem. It ignores some key aspects of real robots: inherent limits on mechanical systems restrict the range of possible motion. For example, a car cannot move sidewise. These limits cause certain configurations to be invalid, even if a robot does not collide with obstacles at those configurations. In this section, we consider two important classes of constraints, kinematic constraints and dynamic constraints, together referred to as *kinodynamic constraints*. Unlike the physical obstacles, kinodynamic constraints cannot always be represented in the configuration space. They involve not only the configuration, but also the velocity and possibly the acceleration of the robot.

To address this issue, we use *state space*, a straightforward generalization of configuration space. Every point in the state space contains information on both the configuration and the velocity of a robot. Our objective is to find, in the state space, an *admissible* path that is both collision-free and satisfies kinodynamic constraints. This class of problems is called *kinodynamic motion planning* [16].

3.1 Kinematic and Dynamic Constraints

Kinematic constraints impose a relationship between the configuration q of a robot and its velocity $\dot{q}$. They can be written mathematically as

$$F(q, \dot{q}) = 0. \tag{1}$$

Kinematic constraints can be further classified into holonomic and non-holonomic ones.

Holonomic constraints do not involve the velocity of a robot; they have the special form $F(q) = 0$. A set of holonomic constraints can be used to eliminate some of the configuration parameters and reduce the dimensionality of $\mathcal{C}$. By choosing a suitable parameterization of $\mathcal{C}$, we may be able to convert a problem with holonomic constraints into one with no constraints and apply the algorithms from Section 2.

Non-holonomic constraints are fundamentally different. They are not integrable, meaning that we cannot eliminate $\dot{q}$ via integration and convert them to the form $F(q) = 0$. A classic example is the constraints on the motion of car-like mobile robots (Figure 7). Let (x, y) be the position of the midpoint *R* between the rear wheels of the robot and θ be the orientation of the rear wheels with respect to the x-axis. Assuming that the wheels do not skid, the robot cannot move sidewise. This constraint can be written as $\tan\theta = \dot{y}/\dot{x}$, which clearly has the form $F(q, \dot{q}) = 0$. What is less obvious is that the constraint is not integrable. We will not get into the details here. It suffices to say that the mathematical conditions for integrability is known, but for a given set of constraints, checking these conditions is a non-trivial task. See [42, pages 403–451] for details.

Although most of the work on non-holonomic motion planning focuses on car-like or tractor-trailer robots, many results are applicable to other problems, including object pushing [54] and dextrous manipulation [31].

Dynamic constraints are closely related to non-holonomic constraints, but they involve not only the configuration and the velocity of a robot, but also the acceleration. Consider the Lagrange's equations of motion, which have the form $G(q, \dot{q}, \ddot{q}) = 0$, where q, $\dot{q}$, and $\ddot{q}$ are the robot's configuration, velocity, and acceleration. Defining $s = (q, \dot{q})$, we can rewrite the equation as $F(s, \dot{s}) = 0$ which is the same as (1).

The motion of a robot may also be constrained by inequalities of the form $F(q, \dot{q}) \leq 0$ or $G(q, \dot{q}, \ddot{q}) \leq 0$. Such constraints restrict the set of admissible states to a subset of the state space.

The presence of kinodynamic constraints implies that not all collision-free path are admissible,

because they may violate the constraints. For some robots, we can represent motion constraints explicitly by constructing a class Γ of admissible path segments. Ideally Γ has the property that if there is an admissible path between two states, then one can construct another admissible path as a sequence of segments from Γ. This property is necessary for algorithms using Γ to be complete. Examples of such path segments include jump curves [33] or Reeds and Shepp curves [63] for car-like robots. In general, one can prove such a class of path segments can be constructed for any locally controllable system using tools from non-linear control theory [4, 44, 49]. Unfortunately, the path segments generated by the proof are often inefficient in practice, because they may contain many unnecessary maneuvers.

An alternative representation of motion constraints is a *control system*

$$\dot{s} = f(s, u), \tag{2}$$

which constitutes the robot's equations of motion under suitable control. In the above equation, $s \in \mathcal{S}$ is the robot's state, which encodes the robot's configuration and optionally velocity as well; $\dot{s}$ is the derivative of s with respect to time; $u \in \Omega$ is the control input. The set $\mathcal{S}$ and Ω are called the *state space* and *control space*, respectively. We assume that $\mathcal{S}$ and Ω are bounded manifolds of dimensions n and m, with $m \leq n$. By defining appropriate charts on these manifolds, we can treat $\mathcal{S}$ as a subset of R^n and Ω, a subset of R^m.

Eq. (2) can represent both kinematic and dynamic constraints described earlier. Suppose that we have ℓ kinodynamic constraints $G_i(s, \dot{s}) = 0$ for $i = 1, 2, \ldots, \ell$. We can solve these ℓ equations for $\dot{s}$. In general, if ℓ is less than n, the solution is not unique, but we can parameterize the set of solutions by $u \in \mathrm{R}^{n-\ell}$ and write them down, at least formally, as $\dot{s} = f(s, u)$ for some suitable function f. More precisely, it can be shown that under suitable conditions, the set of constraints $G_i(s, \dot{s}) = 0$ for $i = 1, 2, \ldots, \ell$ is equivalent to (2), in which u is a point in $\mathrm{R}^m = \mathrm{R}^{n-\ell}$ [4].

To deal with inequality constraints of the form $G(s, \dot{s}) \leq 0$, we typically restrict the state space $\mathcal{S}$ and control space Ω to suitable subsets of R^n and R^m, respectively.

Let us now look at an example to illustrate the above notions.

Example 3.1 (simplified non-holonomic car navigation) Consider the car example in Figure 7. The state of the car is specified by $(x, y, \theta) \in \mathrm{R}^3$. The non-holonomic constraint $\tan\theta = \dot{y}/\dot{x}$ is equivalent to the system

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= (v/L)\tan\phi. \end{aligned}$$

This reformulation corresponds to defining the car's state to be its configuration (x, y, θ) and choosing the control input to be the vector (v, ϕ), where v and ϕ are the car's speed and steering angle, respectively. Bounds on (x, y, θ) and (v, ϕ) can be used to restrict $\mathcal{S}$ and Ω to subsets of R^3 and R^2, respectively. For instance, if the maximum speed of the car is 1, we require $|v| \leq 1$.

3.2 General Approaches

Sometimes the path planning approaches described in Section 2 can be applied to kinodynamic motion planning after some modifications. To construct a roadmap for car-like robots, we may discretize the boundaries of polygonal obstacles and connect pairs of points on the boundaries with jump curves composed of circular and straight-line segments [33]. To apply this idea to other robots would require a suitable class of admissible path segments to be constructed. Alternatively, we may consider the cell decomposition approach by placing a regular grid over the state space [4, 16]. We represent the motion constraints as a control system and search for an admissible path in the discretized state space. As we have mentioned before, the cell-decomposition approach works only for robot with few dofs, because the grid size increases exponentially with $\dim(\mathcal{C})$. We may also use the potential field approach by projecting the potential forces onto the surface defined by the motion constraints and applying the projected forces on the robot.

One approach unique to kinodynamic motion planning is path transformation. It proceeds in three steps [43]. First we generate a collision-free path γ that disregards the motion constraints. We then discretize γ into a sequence of short path segments and replace each segment with one from a class Γ of admissible path segments, thus transforming γ into an admissible path γ'. Finally we smooth γ' to remove the unnecessary maneuvers and obtain a more efficient admissible path. This algorithm can be extended in various ways, which are all based on the idea of successive path transformation, but differ in what transformations to use and how to perform the transformations [6, 21, 68]. A natural question to ask about these path-transformation algorithms is whether it is always possible to transform a collision-free path into an admissible path that obeys the motion constraints. In theory, the answer is yes, if the robot is locally controllable [43], *e.g.*, car-like robots. However, the approach is only practical for robots for which a class Γ of efficient admissible path segments can be easily constructed. It is not applicable to robots that are not locally controllable, *e.g.*, car-like robots that can only go forward.

3.3 Random Sampling

Random sampling has also been successful for kinodynamic motion planning, including robots that are *not* locally controllable. In this section, we give two representative examples.

The first one follows the multi-query approach [73], described in Section 2.3. It applies to car-like robots and assumes the existence of a class Γ of admissible path segments. It proceeds in almost the same way as Algorithm 1, with one major difference. When connecting two milestones in the roadmap, the algorithm uses path segments from Γ instead of straight-line paths. Thus every path in the roadmap is not only collision-free, but also admissible.

The second example follows the single-query approach. It represents the motion constraints as a control system. The main steps of the algorithm are similar to Algorithm 2. The difference occurs in lines 3 and 5. In Algorithm 2, we sample a new configuration and connect it to an existing milestone with a straight-line path. However, straight-line paths often violate the motion

constraints. So instead, we choose a random control function and integrate the robot's equations of motion forward under this control function for a small period of time. The motion constraints are enforced automatically during the integration. If the resulting path is admissible, we then insert the endpoint of the path into the tree being constructed as a new milestone. Intuitively, we map a random sample in the control space Ω to a random sample in the state space $\mathcal{S}$ by integrating the equations of motion. Of course, we must still avoid oversampling. We can use the same methods described in Section 2.3, but they work less effectively here, because the motion constraints skew the density estimate and the target distribution.

It may appear somewhat surprising that, in the random-sampling approach, algorithms for path planning and kinodynamic motion planning are very similar. This is in fact one major advantage of the approach: it applies to a wide class of problems with relatively small, local changes related to the specifics of robots. This greatly eases implementation.

3.4 Case Studies on Real Robotic Systems

Having seen a number of motion planning algorithms, we now look into some important practical issues in the context of two real robotic systems.

Motion planning of trailer-trucks for transporting Airbus A380 components [41]. Airbus A380 is the largest commercial aircraft that has ever been built. The main components—wings, fuselage sections, and the tail plane—are produced in different European cities and transported by trailer-trucks to a central location for assembly. The transport itinerary must go through small towns and villages with sometimes very narrow roads. The enormous size of the cargo, the length of the itinerary, and the narrow roads along the way pose unique challenges. It is highly desirable to have an automated system to help validate the itinerary in advance and guide the truck driver.

Trailer-trucks have been studied extensively in non-holonomic motion planning. In this case, a path transformation algorithm is used for motion planning [40]. An initial admissible path is computed and then iteratively improved to make it more efficient. Obstacle avoidance is achieved

with a potential field method.

The automated system is used to validate the itinerary for the trailer-trucks and determine which parts of the itinerary must be adapted to fit the vehicle size. The system also optimizes the trajectories to maximize the distance between the truck and the surrounding obstacles, such as buildings and trees. The validated trajectory is then fed into a computer-aided driving system to help the driver follow the trajectory.

A space robotics test-bed [29]. A variant of the single-query random-sampling planner described in Section 3.3 has been implemented on a real robot in an environment with moving obstacles. The robot system was developed in the Stanford Aerospace Robotics Laboratory for testing space robotics technology. The air-cushioned robot moves frictionlessly on a flat granite table (Figure 9). It has eight air thrusters providing omni-directional motion capability, but the force is small compared to the robot's mass, resulting in tight acceleration limits.

We model the robot as a disc in the plane for planning purposes. To deal with moving obstacles, the planner augments the state space with a time axis and computes a trajectory for the robot in the state-time space instead of the usual state space. An overhead vision system estimates the motion of moving obstacles in the environment and sends the information to the planner, which runs on an off-board computer. The planner is then allocated a short, pre-defined amount of time to compute a trajectory, as required by the real-time nature of the system,

The success of random sampling for motion planning in real-time system indicates its effectiveness despite many adversarial conditions, including (i) severe dynamic constraints on the robot's motion, (ii) moving obstacles, and (iii) various time delays and uncertainties inherent to an integrated system operating in a physical (as opposed to a simulated) environment.

4 Motion Planning under Visibility Constraints

Often, we picture robots as intelligent machines maneuvering autonomously through a cluttered environment, transporting parts or assembling products. These tasks fall strictly within the domain of classic motion planning. However, acquiring information about environments through *sensing* is another important task: surveillance and mapping unknown environments are all examples of tasks in which observing the world is the main objective. It may not be immediately obvious, but motion planning plays a key role in these problems.

The goal of sensing is to extract an understanding of the world from sensor data. The basic act of sensing is passive. It becomes active when an algorithm directs the robot to move in order to make sensing more effective. The motion may help the robot keep a target within the sensor range or gain new information about an unknown environment. More generally, motion is executed to maintain a set of constraints on the state of the world or achieve a certain state of knowledge about the world. Here, the term "state" reflects not only the robot's physical configuration, as in the previous sections, but also the robot's observations and knowledge. The admissible paths for the robot are constrained not only by the robot's geometry and mechanics, but also by a set of *visibility constraints* due to the robot's sensors.

To understand the role of visibility constraints, consider the example of a robot following a target. Suppose that at its initial location, the robot has the target in view. As the target moves, it may get out of the robot's sensor range. The robot must move to a new location to keep the target in view. The path that the robot takes must, of course, be collision-free. In addition, at every point along the path, the robot must maintain target visibility. The visibility constraints reduce the set of admissible paths available to the robot, just as the kinodynamic constraints do. To deal with visibility constraints effectively, we must now leave the realm of classic motion planning and enter the realm of *motion planning under visibility constraints*.

This section presents three motion planning problems under visibility constraints: sensor placement (Section 4.1), indoor exploration (Section 4.2), and target tracking (Section 4.3). In the first

problem, we compute a set of robot sensing locations to build a model of an environment effectively. This is the simplest scenario, because we ignore the cost of robot motion. The second problem, often called the *next best view*, is an extension of the first, when the environment is not known in advance. Motion planning becomes important, because the robot may inadvertently collide with unknown obstacles in the environment. The last problem is that of computing the motion of a robot observer following a target. This is probably the most complex problem of the three, because it involves both visibility and kinodynamic constraints. Moreover, the robot is sometimes expected to track an unpredictable target in real time.

4.1 Sensor Placement

Nowadays robots equipped with laser range sensors are often used to build 3-D models of the environment [12, 55, 61, 76]. Acquiring high-quality 3-D information is a costly operation, and it is desirable to minimize the number of sensing operations. To do this, we use an initial 2-D map of the environment and compute a set of locations from which a range sensor (*e.g.*, laser) scans the environment. We call this problem *sensor placement*.

Sensor placement is related to the classic art-gallery problem [69], which asks for the minimum number of guards whose joint visibility region covers the interior of an art gallery. In its simplest form, the problem considers the art gallery to be a polygonal environment. It also assumes a simple line-of-sight visibility model, where two points are visible to each other if the line segment between them is unobstructed. The problem seems deceptively simple, but finding the minimum number of guards is actually NP-hard. In robotics, the visibility model is rarely as clean as that assumed in the art-gallery problem. So the art-gallery results are usually not directly applicable.

To derive a practical sensor placement algorithm, the visibility model must take into account the limitations of laser range sensors. The visibility definition below lists three constraints, which, we believe, are most relevant (Figure 10).

Definition 4.1 (constrained visibility) *Let the bounded and open set $\mathcal{W} \subset \mathrm{R}^2$ denote the robot's*

free space, and $\partial\mathcal{W}$ denote the boundary of $\mathcal{W}$. A point $w \in \partial\mathcal{W}$ is visible *from a point $q \in \mathcal{W}$ if the following conditions hold:*

- Line-of-sight constraint: *The open segment $S(q, w)$ joining q and w does not intersect $\partial\mathcal{W}$.*
- Range constraint: $d_{\min} \leq d(q, w) \leq d_{\max}$*, where $d(q, w)$ is the Euclidean distance between q and w, and $d_{\min} \geq 0$ and $d_{\max} > d_{\min}$ are constants.*
- Incidence constraint: $\angle(\boldsymbol{n}, \boldsymbol{v}) \leq \tau$*, where $\boldsymbol{n}$ is the vector perpendicular to $\partial\mathcal{W}$ at w, $\boldsymbol{v}$ is the vector oriented from w to q, and $\tau \in [0, \pi/2]$ is a constant.*

We are interested in finding a minimal set of sensor locations that cover $\partial\mathcal{W}$:

Problem 4.1 (sensor placement) *Given a bounded, open set $\mathcal{W} \subset \mathrm{R}^2$, compute the minimal set of sensor locations $\mathcal{G}$ in $\mathcal{W}$, such that every point $w \in \partial\mathcal{W}$ is visible from at least one point in $\mathcal{G}$ under the visibility model given in* Definition 4.1.

Like the art-gallery problem, Problem 4.1 is NP-Hard, and we have to settle for an approximate solution, one that covers most of, but not the entire boundary $\partial\mathcal{W}$. We use random sampling to transform the sensor placement problem into a *set cover problem* [23].

Sampling. Sample at random a set of m points from $\mathcal{W}$. Denote the set by $\mathcal{G}_{\mathrm{sam}}$. For every edge $e \in \partial\mathcal{W}$, compute the fraction seen by each point in $\mathcal{G}_{\mathrm{sam}}$. The arrangement of all covered fractions decomposes each edge into cells such that all points within the same cell are visible to the same subset of $\mathcal{G}_{\mathrm{sam}}$ (see Figure 11*a* for an example). Now enumerate all the cells in the decomposition of $\partial\mathcal{W}$ and group them under the ground set $X = \{1, 2, \ldots, l\}$, where l is the number of cells. This ground set represents the decomposition of $\partial\mathcal{W}$.

Let R_i be the subset of X that is visible to a sample point $g_i \in \mathcal{G}_{\mathrm{sam}}$. The set family $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$ is thus a collection of subsets of X. The set system $\Sigma = (X, \mathcal{R})$ can be regarded as an encoding of the sampled or discretized version of Problem 4.1, and the original problem is

reduced to that of computing the optimal set cover of the set system Σ: find the smallest sub-collection $\hat{\mathcal{R}} \subseteq \mathcal{R}$, such that the union of all the R_i's in $\hat{\mathcal{R}}$ equals X.

The sampled problem is clearly not the same as the original. Finding the optimal set cover of Σ may not lead to an optimal set of sensor locations: $\mathcal{G}_{\text{sam}}$ may contain incorrectly distributed points, or $\mathcal{W}$ admits no finite solution due to its geometry. Sampling, however, often produces a satisfactory solution at a small cost, because the probability that $\mathcal{G}_{\text{sam}}$ contains the optimal set of guards quickly approaches 1 in most practical scenarios. Even when no finite solution exists, sampling produces reasonable solutions, as Σ encodes a "portion" of the original problem that actually admits a finite solution for realistic sensor models.

Near-optimal set covers After sampling, we ask the question: has the problem become easier? Unfortunately the set cover problem is also NP-hard. However, finding optimal set covers is a well-studied problem, and efficient algorithms that produce near-optimal solutions are available. More interestingly, the set cover problem has a dual, the *hitting set problem*.

Every set system has a dual. Consider $\Sigma = (X, \mathcal{R})$. Its *dual* $\Sigma' = (X', \mathcal{R}')$ is defined by $X' = \mathcal{R}$ and $\mathcal{R}' = \{\mathcal{R}_x | x \in X\}$, where $\mathcal{R}_x$ consists of all the sets $R \in \mathcal{R}$ that contain x. Figure 11*b* illustrates the dual set system for our sensor placement problem. Note that the set of candidate sensor locations now becomes the ground set X'. A *hitting set* for $\Sigma' = (X', \mathcal{R}')$ is a subset $H' \subseteq X'$ such that $H' \cap R' \neq \emptyset$ for every set R' in $\mathcal{R}'$. In other words, the hitting set H' contains members from all the sets in $\mathcal{R}'$. The problem of finding the smallest set cover for Σ is *equivalent* to that of finding the smallest hitting set for Σ'. For a set system with finite VC-dimensions[1], an efficient algorithm exists for finding near-optimal hitting sets [9].

Assume that $\mathcal{W}$ is represented as a polygon with holes caused by obstacles. The VC-dimension of the set system for the sampled version of Problem 4.1 is then bounded by $\mathrm{O}(\log(n+h))$, where n is the number of vertices describing $\partial\mathcal{W}$ and h is the number of holes [23]. Using the algorithm in [9], we can find a set of sensor locations that is within a factor $\mathrm{O}(\log(n+h) \cdot \log(c \log(n+h)))$

[1]VC-dimension stands for the Vapnik-Červonenkis dimension. It is a measure of the complexity of a set system.

of the optimal size c. In other words, we can compute a near-optimal set of sensor locations within a logarithmic factor of the optimal.

Sensor placement is a set cover problem in nature, and the same is true for art-gallery problems in general. A key development in recent years is to transform such problems into set systems, which may have finite VC-dimensions and lead to efficient approximation algorithms. For example, it has been shown that for a polygon with h holes, the VC-dimension of the set system for the classic art-gallery problem is $O(h)$ [75] under the simple line-of-sight visibility model. This fact is exploited to produce a polynomial-time algorithm that finds a solution within a factor $O(\log(h) \cdot \log(c \log(h)))$ of the optimal size c [18].

Extensions A straightforward extension of the sensor placement problem is to generate routes instead of locations for sensing tasks involving mobile robots. If the cost of sensing is very expensive compared to that of motion, then motion costs can be ignored. The problem remains the same as that defined in Problem 4.1. If the converse is true, then the cost of sensing can be ignored, and motion incurs the dominant cost. The problem becomes the *watchman route problem* [69]: find the shortest closed path from which the entire environment is visible. Developing sampling techniques to compute watchman routes is an interesting topic for future research.

A more difficult problem requires both the cost of sensing and the cost of motion to be considered. This topic remains largely unexplored, but some limited work exists [14, 20].

4.2 Indoor Exploration

Automatic map building is an important problem in robotics. Research in this area has traditionally focused on developing techniques to extract environmental features, such as edges and corners, from sensor data and integrating these features into a consistent map. The former is a computer vision problem, and the latter is the *simultaneous localization and mapping* (SLAM) problem [74].

SLAM algorithms seek the best way to integrate sensor data acquired by a robot during navigation. It, however, does not answer the following question: Given the map known so far, where

should the robot move next to observe the unexplored regions? From the point of view of motion planning, this is the most interesting question in automatic map building. It involves the computation of successive sensing locations by iteratively solving the *next best view* (NBV) problem. At each location, the robot must not only observe large unexplored areas of the environment, but also a portion of the known environment to allow for image registration [62]. NBV is complementary to SLAM [74]. A SLAM algorithm builds a map by making the best use of the available sensor data, whereas an NBV algorithm guides the robot through locations that provide the best possible sensor data. In addition to robotics and computer vision, NBV arises in computer graphics [13] and many other areas.

NBV is an *on-line* version of the sensor placement problem, where the 2-D map of the environment is unknown initially and only revealed incrementally as new sensor data are acquired.

Constraints on the next best view. In mobile robotics, two important constraints must be considered by NBV algorithms. First, a mapping robot must not collide with obstacles, whether they are known or unknown in advance. The second constraint results from imperfect robot localization. Due to errors in inertial navigation (*e.g.*, wheel slippage), a mobile robot must constantly re-localize itself as the map is built. New laser scan images must be aligned with the current map, a problem called *image registration*. Image registration requires an overlap between each new image and previously seen portions of the environment. An NBV algorithm must take this requirement into account.

NBV can thus be viewed as an optimization problem where the best sensing position is computed subject to safe-navigation and image-registration constraints. As is often the case in optimization, the problem can be solved more effectively if the search domain is characterized explicitly. In motion planning terms, the next best view is a position in the free space, where the free space is collision-free with respect to both the known and unknown obstacles. Is it possible to characterize this free space explicitly?

It seems odd to define a free space that depends on obstacles yet to be discovered, for if they

are not discovered, how can we use them to build the free space? The key is to view free space from the sensor's perspective, and not from the environment's perspective. That is, construct the largest region guaranteed to be free of obstacles, mapped or not, given the history of sensor data. Such a region is called the *safe region* to distinguish it from the usual notion of free space.

Safe regions. Consider a 2-D range sensor that obeys the visibility model in Definition 4.1, with $d_{\min} = 0$. Figure 12*a* shows a sample sensor reading. Here, the sensor detects the obstacle contour shown in bold black. From this reading, we want to construct a closed region that is obstacle-free. One possibility is to join the detected contour to the range limit of the sensor using radial line segments. This region is shown in light color in Figure 12*b*. Unfortunately, such a region is guaranteed to be free of obstacles only in the absence of incidence constraints. Consider Figure 12*c*, which shows the actual environment. Notice how the region from Figure 12*b* overlaps with walls oriented at a grazing angle (roughly 70 degrees) with respect to the sensor position. In contrast, the region in Figure 12*d*, which takes into account the incidence constraint, is indeed safe.

Assume that the sensor output is an ordered list Π of piecewise continuous curves. The *local safe region* $s_l(q)$ is the largest closed region guaranteed to be free of obstacles given an observation $\Pi(q)$ made at location q. Such a region is bounded by the curves in $\Pi(q)$, representing the visible sections of the free space boundary $\partial\mathcal{W}$, plus additional curves joining the disjoint visible sections and calculated from the information in $\Pi(q)$ [24] (see Figure 12*d* for an example). The safe region $s_l(q)$ is topologically equivalent to a classic visibility region. In fact, when the visibility constraints in Definition 4.1 are relaxed, the safe region becomes exactly the visibility region. Several properties and algorithms that apply to visibility regions also apply to safe regions. For example, $s_l(q)$ is a *star-shaped* set, a set that is entirely visible from at least one interior point.

A global safe region is constructed iteratively from local safe regions. First, a local safe region $s_l(q_0)$ is constructed from the sensor reading $\Pi(q_0)$ made at the robot's initial position q_0. The global safe region $S_g(q_0)$ is initially equal to $s_l(q_0)$. Next, the robot moves to a position q_1 and gets a new sensor reading $\Pi(q_1)$, yielding a new local safe region $s_l(q_1)$. Now, $S_g(q_1) = S_g(q_0) \cup s_l(q_1)$.

The robot again moves, now to q_2. A new reading $\Pi(q_2)$ is made, yielding $s_l(q_2)$, and $S_g(q_2) = S_g(q_1) \cup s_l(q_2)$, and so on. The region $S_g(q_t)$ represents both a map of the environment at time t and the search domain for computing the next best view for $t+1$.

Image registration. Robots cannot localize with perfect precision. An algorithm ALIGN is used to compute the transform T that aligns $s_l(q_{t+1})$ with $S_g(q_t)$ before the union operation. Image registration has been studied widely, and many techniques exist [61]. The details of ALIGN are inconsequential to the NBV computation, but it is important to note that most image registration algorithms are based on feature matching. It is thus essential that the next best view for $t+1$ ensures a minimum overlap between the current $S_g(q_t)$ and the anticipated $s_l(q_{t+1})$.

Evaluating next views. Suppose that at time t, the robot is positioned at q_t and the global safe region is $S_g(q_t)$. The goal is to compute the future position of the robot, given $S_g(q_t)$. The unexplored areas of the environment can only be revealed through the free boundary of $S_g(q_t)$, *i.e.*, the portions of $S_g(q_t)$ not blocked by obstacles. Therefore, a potential candidate q is good, if it sees large unexplored areas outside of $S_g(q_t)$ through the free boundary of $S_g(q_t)$. We say that such q has high *potential visibility gain*, measured by a function $V_g(q,t)$.

Several definitions of $V_g(q,t)$ are possible. One way is to first compute the visibility region from q assuming that the free boundary is transparent, and intersect this region with the complement of $S_g(q_t)$ [24]. The gain $V_g(q,t)$ is the area of the resulting intersection. This definition works well for office environments, even in cluttered conditions. As an alternative, the next view can be chosen to maximize entropy reduction, and the gain $V_g(q,t)$ becomes a measure of the expected entropy reduction at position q [78].

The computation of the next best view must also factor in the cost of motion, which is weighed against the potential visibility gain. Again, this can be done in several ways. One way is to define the overall merit of q, factoring in both visibility gain and motion cost, as

$$g(q,t) \;=\; V_g(q,t)\exp(-\lambda L(q,t)), \tag{3}$$

where $L(q,t)$ is the length of the collision-free path computed by a path planner between position q and the current robot position at time t. The constant $\lambda \geq 0$ is used to weigh the cost of motion against the visibility gain. A small λ gives priority to the gain of information. Conversely, a large λ gives priority to motion economy, favoring locations near q_t that potentially produce marginal information gain.

Computing the next best view. At this point, the only remaining issue is to search for the next best view. This is simple, as the global safe region $S_g(q_t)$ completely characterizes the search domain. Following a random-sampling approach akin to those described in Section 2.3, a set $\mathcal{N}$ of NBV candidate positions is generated along the free boundary of $S_g(q_t)$. This set is processed in three steps. First, for each $q \in \mathcal{N}$, we determine the extent to which $s_l(q)$ and $S_g(q_t)$ overlap. The overlap $\zeta(q)$ is measured by the length of the visible part of $S_g(q_t)$'s boundary abutting obstacles. If $\zeta(q)$ is smaller than a threshold imposed by ALIGN, then q is removed from $\mathcal{N}$. Second, a path planner computes a collision-free path between q_t and each remaining candidate q in $\mathcal{N}$. Those candidates that yield no feasible paths are removed from $\mathcal{N}$. Finally, the merit of each remaining candidate in $\mathcal{N}$ is evaluated according to (3), and the best candidate is selected.

Figure 13 shows a sample map constructed using the NBV algorithm in [24]. The figure shows the partial map of a wing of the Computer Science Building at Stanford University after 14 iterations. Note the final mismatch after the robot completed a circuit around the lab (about 40 meters). The discrepancy appears, because every image alignment transform was computed locally. To reduce the discrepancy, the NBV algorithm should be combined with a SLAM algorithm.

Extensions. We have so far ignored any error-recovery capabilities in the NBV computation. Any serious errors in sensing or image registration lead to unacceptable maps. An experimental system must be designed conservatively to avoid this, perhaps forcing the robot to take more measurements or travel longer paths to produce the final map. A better solution is to combine the NBV computation with SLAM algorithms and exploit their complementary strengths.

Another extension is to have multiple robots building a map cooperatively. Centralized ap-

proaches are acceptable, if the relative positions of all the robots are known. A single map can be generated from all the sensor readings, and a centralized NBV algorithm then computes the aggregate next best view for the entire team. The problem becomes far more difficult, if the relative positions of the robots are not known. In this case, the robots act independently, perhaps communicating their positions and findings only sporadically. A distributed approach is then needed.

4.3 Target Tracking

Tracking in the sense of detecting targets in images is studied widely in computer vision. In contrast, target tracking in motion planning is concerned with computing the motion of a robotized camera in order to keep a target in view [5]. Variations of this problem arise in different applications, *e.g.*, visual servoing [32, 60] and computer assisted surgery [47]. Target tracking is also called *target following* to distinguish it from the tracking problem in computer vision.

Target tracking is a motion planning problem that combines visibility constraints with kinodynamic constraints. It takes into account the actions of an external agent—the target—acting as a potential opponent. Thus target tracking can be treated as a problem in game theory [45]. The game-theoretic view provides a clean mathematical formulation of the problem.

State transition equations. Suppose that both the robot observer and the target are rigid bodies moving in the plane. The free configuration space for the observer is a subset of R^2 and denoted by $\mathcal{F}^o$, while that for the target is denoted by $\mathcal{F}^t$. Define $s^o(t)$ as the observer's *state* at time t. Suppose that the state transition equation for the observer is given by $\dot{s}^o = f^o(s^o, u)$, where $u(t)$ is the action selected from an action set $\mathcal{U}$ at time t. The function f^o models the observer's dynamics and may encode non-holonomic constraints or other types of kinodynamic constraints. Similarly, the transition equation for the target is given by $\dot{s}^t = f^t(s^t, \theta)$, with the action $\theta(t)$ selected from a target action set Θ. The state of the observer-target system is given by $s = (s^o, s^t)$. Let $\mathcal{X}$ be the joint state space, which is the Cartesian product of the individual state spaces of both the observer and the target. A state may encode both the configuration of a robot and its velocity. So, in general,

$\mathcal{X}$ is *not* equal to $\mathcal{F}^o \times \mathcal{F}^t$, the Cartesian product of individual configuration spaces.

Visibility constraints. The distinction between state space and configuration space is important. The state space, along with the associated transition equation, focuses on the kinodynamic constraints. The configuration space, on the other hand, focuses on where the robot observer can see the target. Now let us identify those configurations where the target is visible.

Let $\mathcal{V}(q^o)$ be the visibility region at the observer position q^o, *i.e.*, the set of all locations from which the target is visible to an observer located at q^o. Usually, the target is said to be visible if the line of sight to the observer is unobstructed, but this model can be extended. For example, the field of view can be restricted to some fixed visibility cone or limited by lower- and upper-bounds on the distance range. Incidence constraints such as those in Definition 4.1 can also be added.

Tracking algorithms usually compute the visibility region from a synthetic model or reconstruct it from sensor data. In the former case, a sweep-line algorithm can be used [15]. In the latter case, laser range sensors or similar sensors are installed on the robot observer (see Figure 14*a*), but some sensors cannot provide reliable measurements, thus complicating the reconstruction of the visibility region. For example, stereo vision systems often produce unreliable range measurements if the object surface is textureless.

An important concept in target tracking is that of the *visibility sweeping line* $\ell(t)$, defined as the line passing through the target position at time t and a reflex vertex of the free space (Figure 15). At any time t, the observer must stay on the side of $\ell(t)$ which allows it to see the the target. The observer's path is influenced by the behavior of these sweeping lines, and some tracking algorithms exploit them explicitly [17, 25].

Tracking strategies. Target tracking consists of computing a function $u(t)$, called a *strategy*, so that the target remains in view for all $t \in [0, T]$, where T is the target's *stopping time*, also known as the *horizon* of the problem. It may also be important to optimize secondary criteria such as the total distance traversed by the observer and the final distance to the target. Various tracking strategies are known, and they can be compared from different angles:

Predictable vs. unpredictable targets. The target is predictable if the target action $\theta(t) \in \Theta$ is known in advance for all $t \leq T$. Thus the location of the target is known for all t, and its state transition equation simplifies to $\dot{s}^t = f^t(s^t)$. The target is unpredictable if its actions are not known in advance, though the action set Θ may be known.

Off-line vs. on-line. Off-line tracking strategies have access to future states, while on-line strategies do not. In other words, on-line algorithms are *causal*, whereas off-line ones are *non-causal*. Causality is a characteristic of the algorithm, not a logical requirement of target predictability. Obviously, an off-line strategy that relies on the target's future positions implies that the target is predictable, but an algorithm can be non-causal for other reasons. Also note that on-line strategies may or may not run in closed loop (see below).

Critical vs. average tracking. Sometimes it is impossible to track the target for all $t \leq T$. Thus some strategies maximize the target's *escape time* t_{esc}, the time when the observer first loses the target. An alternative is to maximize the *exposure*, the total time that the target remains visible. The former choice, critical tracking, implies that losing the target effectively ends the task, whereas the latter choice, average tracking, implies that the observer can possibly re-acquire the target after losing it.

Expected vs. worst-case analysis. A tracking strategy may maximize either worst-case or expected performance. In the first case, a tracking strategy maximizes the minimum escape time given all the adversarial choices for $\theta(t) \in \Theta$ during the problem's horizon. This approach is suitable for tracking antagonistic targets. In the second case, the expected escape time is maximized given a probability distribution over the target's actions. In both cases, the problem is intractable, and we have to settle for approximate solutions. A typical one is to solve the problem for a time horizon much smaller than the target's stopping time T.

Open vs. closed loop. A strategy operates in closed loop, if the strategy u is computed as a function of the state $s(t)$. Otherwise, the strategy runs in open loop, and u depends

explicitly on t. Closed-loop strategies are preferred over open-loop ones even when the target is predictable, unless it is guaranteed that the state transition models and observations are exact, *e.g.*, the case in [48]. Open-loop strategies are often used in theoretical studies, but they rarely work well in practice.

Backchaining and dynamic programming. One way to compute an observer trajectory for a predictable target is through *pre-image backchaining*. Suppose that both the observer and the target are modeled as points in the plane. Let $\bar{\mathcal{V}}(q^t) \subset \mathcal{F}^o$ be the set of observer configurations from which a target at q^t is visible. Let $\mathcal{A}(t) \subset \mathcal{F}^o$ be the set of all configurations at time t from which the observer could move into $\bar{\mathcal{V}}(q^t(t+1))$ at time $t+1$. Since the observer must see the target at time t and move to a configuration that sees the target at $t+1$, its configuration at t must be contained in $\bar{\mathcal{V}}(q^t(t)) \cap \mathcal{A}(t)$, which can often be computed easily for the 2-D case. Thus, the observer's trajectory can be obtained by backchaining from the final stage, guaranteeing visibility at each step, until a set of possible initial states is obtained or the problem is shown to have no solution.

Backchaining can be generalized into higher dimensions using dynamic programming (DP) [45]. Kinodynamic constraints and secondary optimization criteria can also be added. However, DP is computationally intensive. A brute-force implementation of DP leads to a grid whose size grows exponentially with the dimensionality of the state space. Random sampling may ease the computational burden, but to achieve real-time performance, approximate local strategies are needed.

Escape-time approximations. The time horizon is often reduced in practice to handle unpredictable targets. In the extreme case, only one step into the future is considered. If there are no kinodynamic constraints, maximizing the minimum escape time is equivalent to maximizing the shortest distance to escape. The observer's action for the next step can be selected to maximize this distance. This is sometimes achieved through randomized techniques [56]. The shortest distance to escape is easy to compute, but it could be a poor approximation of the escape time for longer time horizons or under kinodynamic constraints.

Alternatively, the escape time can be approximated with a quantity called the *escape risk* [25]. The negative gradient of the escape risk is composed of a *reactive* component and a *look-ahead* component. The reactive component drives the observer to swing around corners as a target is about to be occluded, while the look-ahead component drives the observer towards a corner in order to make future tracking easier. The algorithm relies on an *escape-path tree*, a data structure encoding all the locally worst-case paths that a target may use to escape the observer's visibility region (Figure 16). This data structure can be computed in $O(n)$ time for 2-D environments, where n is the number of polygon vertices describing the observer's visibility region.

Robot localization. If a tracking strategy uses a global map of the environment to determine the observer's actions, tracking is tied to robot localization. This connection potentially leads to a conflict between the goals of tracking and localization. Suppose, for example, that the observer re-localizes whenever a ceiling landmark is visible. The target may force an observer trajectory without any landmarks, resulting in the localization error becoming so large that tracking fails.

A simple solution of this problem is to increase the number of landmarks, or to use more robust localization techniques based on (hopefully) abundant natural features. A better solution is to explicitly add the re-localization constraint into the tracking problem. For example, the observer actions maximize the sum of two utility functions: one based on the probability of observing the target and the other based on localization precision [19].

An entirely different approach is to abandon the global map and avoid the localization problem altogether. For example, the observer's actions could depend only on the gradient of the escape risk, which can be computed from purely local sensor information [25].

Other results and extensions. We often ignore the kinodynamic constraints on the observer and the target in order to simplify the tracking problem. However, it is important to assume bounded target velocity; otherwise, the target's escape time may become zero. The effect of velocity bounds on tracking has been studied [57]. Assuming bounded target velocity, an optimal strategy can be computed efficiently for polygonal environments and predictable targets [17].

An interesting extension of the tracking problem is that of *stealth tracking*: the observer tracks the target while remaining hidden from it. The work in [3] extends the linear-time algorithm in [25] to account for the additional stealth constraint. This involves computing the subset of the target's visibility region contained inside the observer's visibility region. The computation can be done efficiently so that the total cost of the strategy remains linear per step.

A more difficult problem is to track multiple targets with multiple observers. If a centralized strategy is used, the problem is not fundamentally different from tracking a single target with a single observer. However, the dimensionality of the state space gets higher, and visibility regions may become disconnected [56]. Distributed strategies, on the other hand, require a coordination scheme among observers.

5 Other Important Issues

Uncertainty is animportant issue in motion planning, but we will only touch on it very briefly here. See Chapter 13 for more details. Except for Sections 3.4 and 4.2, we have mostly assumed that a planning algorithm knows exactly the geometry of the robot, the shapes and locations of obstacles in the environment, and when and how the environment changes. We have also assumed that the robot can exactly execute the path computed by a planning algorithm. These assumptions are satisfied to various degrees in real robotic systems.

Depending on the degree of uncertainty present and the amount of prior knowledge available, there are different ways to deal with uncertainty. If the uncertainties are small, we can largely ignore them during planning and use closed-loop control during path execution to reduce its effects (*e.g.*, the air-cushioned robot in Section 3.4). If uncertainty is bounded or modeled by a probability distribution, we can incorporate it into planning using methods such as pre-image backchaining [52] or partially observable Markov decision processes (POMDP) [34]. In this case, path planning and execution together form a closed-loop process. However, the computational cost of incorporating uncertainty into planning is often high and sometimes intractable. Also, uncertainty

is difficult to model effectively for lack of prior knowledge, and we must rely on a worst-case analysis of various possible scenarios (*e.g.*, in the target tracking problem of Section 4.3). In the extreme case, no prior knowledge of the environment is available. Planning is then of little use, and the robot must rely on sensor-based reaction.

Another important topic is multi-robot motion planning. Conceptually, we can take the cross product of the state spaces of all the robots involved and plan in this composite space. This is called centralized planning, which is computationally expensive due to the high dimensionality of the composite space. Alternatively, we may plan the motion for each individual robot separately and then coordinate their motion afterwards. This is called distributed planning, which is computationally more efficient, but sacrifices completeness and optimality. Chapter 11 provides a more in-depth discussion of this topic.

In recent years, bi-pedal humanoid robots become more prevalent, *e.g.*, Honda's ASIMO and Sony's QRIO. A bipedal robot has the ability to navigate on uneven surfaces and step over obstacles along its path, but efficient footstep planning algorithms that take into account the robot's dynamics are needed to realize this potential [38]. Motion planning for humanoid robots is an important area of research, but is outside the scope of this chapter.

6 Conclusion

Motion planning has moved far beyond its original form of computing a collision-free path for a mobile robot to move from an initial to a final goal position. We have seen in this chapter how kinodynamic constraints and visibility constraints come into play. Nowadays motion planners compute footsteps for humanoid robots [38], paths for inserting a probe into an airplane engine with hundred of parts [10], and motion trajectories for minimal-invasive procedures in robot-assisted surgery [67]. Motion planning also continues to grow into unexpected domains, *e.g.*, exploring molecular energy landscapes [71, 72]. In all these disparate problems, our objective remains the same: find a sequence of admissible motions, to transform the world from an initial to a final state

or to maintain a set of constraints on the state. The notion of what constitutes a state has certainly expanded to cover an increasing number of applications; yet, motion remains the crux of the problem.

In recent years, we have also witnessed a trend towards the unification of principles. In essence, motion planning is a collection of common principles for analyzing motion combinatorially. "Motion" refers to the continuous process of state changes, and "combinatorial" refers to the partition of the continuous process into discrete elements. Motion planning studies those problems where the rearrangement of these elements is the result of motion—problems that cannot be reduced to pure instances of computational geometry or control theory. As we have seen in this chapter, random sampling plays a critical role in solving these problems and has shown great success.

References

[1] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In P.K. Agarwal et al., editors, *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 155–168. A. K. Peters, Wellesley, MA, 1998.

[2] F. Avnaim and J.-D. Boissonnat. Polygon placement under translation and rotation. In *Proc. Annual Symposium on Theoretical Aspects of Computer Science*, volume 294 of *LNCS*, pages 322–333. Springer-Verlag, 1988.

[3] T. Bandyopadhyay, Y.P. Li, M.H. Ang Jr., and D. Hsu. Stealth tracking of an unpredictable target among obstacles. In *Proc. The Sixth Int. Workshop on the Algorithmic Foundations of Robotics*. Springer-Verlag, 2004.

[4] J. Barraquand and J.C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-4):121–155, 1993.

[5] C. Becker, H.H. González-Baños, J.-C. Latombe, and C. Tomasi. An intelligent observer. In *Proc. Int. Symp. on Experimental Robotics*, pages 153–160, 1995.

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

[7] R. Bohlin and L.E. Kavraki. Path planning using lazy PRM. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 521–528, 2000.

[8] V. Boor, M.H. Overmars, and F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1018–1023, 1999.

[9] H. Brönnimann and M.T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete and Computational Geometry*, 14:463–479, 1995.

[10] H. Chang and T.-Y. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1012–1019, 1995.

[11] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion : Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[12] C. I. Conolly. The determination of next best views. In *Porc. IEEE Int. Conf. on Robotics & Automation*, pages 432–435, 1985.

[13] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *SIGGRAPH 96 Conference Proceedings*, pages 303–312, 1996.

[14] T. Danner and L.E. Kavraki. Randomized planning for short inspection paths. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 971–976, 2000.

[15] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 2nd edition, 2000.

[16] B.R. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *J. ACM*, 40(5):1048–1066, 1993.

[17] A. Efrat, H.H. González-Baños, S. Kobourov, and L. Palaniappan. Optimal strategies to track and capture a predictable target. In *Proc. Int. Conf. on Robotics & Automation*, pages 3789–3796, 2003.

[18] A. Efrat and S. Har-Peled. Locating guards in art galleries. In *Proc. IFIP Int. Conf. on Theoretical Computer Science*, pages 181–192, 2002.

[19] P. Fabiani and J.C. Latombe. Dealing with geometric constraints in game-theoretic planning. In *Proc. Int. Joint Conf. on Artif. Intell.*, pages 942–947, 1999.

[20] S.P. Fekete, R. Klein, and A. Nüchter. Online searching with an autonomous robot. In *Proc. The Sixth Int. Workshop on the Algorithmic Foundations of Robotics*, 2004.

[21] P. Ferbach. A method of progressive constraints for nonholonomic motion planning. *IEEE Trans. Robotics & Automation*, 14(1):172–179, 1998.

[22] M. Foskey, M. Garber, M.C. Lin, and D. Manocha. A Voronoi-based hybrid motion planner. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 55–60, 2001.

[23] H.H. González-Baños and J.C. Latombe. A randomized art-gallery algorithm for sensor placement. In *Proc. ACM Symp. on Computational Geometry*, pages 232–240, 2001.

[24] H.H. González-Baños and Jean-Claude Latombe. Navigation strategies for exploring indoor environments. *Int. J. Robotics Research*, 21(10-11):829–848, 2002.

[25] H.H. González-Baños, C.-Y. Lee, and J.-C. Latombe. Real-time combinatorial tracking of a target moving unpredictably among obstacles. In *Proc. Int. Conf. on Robotics & Automation*, pages 1683–1690, 2002.

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

[27] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4420–4426, 2003.

[28] D. Hsu, L.E. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P.K. Agarwal et al., editors, *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 141–154. A. K. Peters, 1998.

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

[30] D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 2719–2726, 1997.

[31] P. Hsu, Z. Li, and S. Sastry. On grasping and coordinated manipulation by a multifingered robot hand. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 384–389, 1988.

[32] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Trans. Robotcis & Automation*, 12(5):651–670, October 1996.

[33] P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 2–7, 1989.

[34] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

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

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

[37] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robotics Research*, 5(1):90–98, 1986.

[38] J.J. Kuffner, K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue. Online footstep planning for humanoid robots. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2003.

[39] A.M. Ladd and L.E. Kavraki. Theoretic analysis of probabilistic path planning. *IEEE Trans. Robotics & Automation*, 20(2):229–242, 2004.

[40] F. Lamiraux, D. Bonnafous, and C. Van Geem. Path optimization for nonholonomic systems: Application to reactive obstacle avoidance and path planning. In *Control Problems in Robotics*. Springer, 2002.

[41] F. Lamiraux, J.-P. Laumond, C. Van Geem, D. Boutonnet, and G. Raust. Trailer-truck trajectory optimization for airbus A380 component transportation. *IEEE Robotics and Automation Magazine*, to appear.

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

[43] J.-P. Laumond. Feasible trajectories for mobile robots with kinematic and environmental constraints. In *Proc. Int. Conf. on Intelligent Autonomous Systems*, pages 346–354, 1986.

[44] J.-P. Laumond, S. Sekhavat, and F. Lamiraux. Guidelines in nonholonomic motion planning for mobile robots. In J.P. Laumond, editor, *Robot Motion Planning and Control*, Lectures Notes in Control and Information Sciences 229, pages 1–53. Springer, 1998.

[45] S.M. LaValle, H.H. González-Baños, C. Becker, and J.C. Latombe. Motion strategies for maintaining visibility of a moving target. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 731–736, 1997.

[46] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 473–479, 1999.

[47] S. M. Lavallée, J. Troccaz, L. Gaborit, A. L. Benabid P. Cinquin, and D. Hoffmann. Image guided operating robot: A clinical application in stereotactic neurosurgery. In R.H.Taylor et al., editors, *Computer Integrated Surgery: Technology and Clinical Applications*, pages 342–351. The MIT Press, 1995.

[48] T.Y. Li, J.M. Lien, S.Y. Chiu, and T.H. Yu. Automatically generating virtual guided tours. In *Proc. Computer Animation*, pages 99–106, 1999.

[49] Z. Li, J.F. Canny, and G. Heinzinger. Robot motion planning with nonholonomic constraints. In H. Miura et al., editors, *Robotics Research: The Fifth International Symposium*, pages 309–316. The MIT Press, Cambridge, MA, 1989.

[50] J.-M. Lien, S.L. Thomas, and N.M. Amato. A general framework for sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4439–4444, 2003.

[51] M. Lin and D. Manocha. Collision and proximity queries. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35. CRC Press, 2004.

[52] T. Lozano-Pérez, M.T. Mason, and R.H. Taylor. Automatic synthesis of find-motion strategies for robot. *Int. J. Robotics Research*, 3(1):3–24, 1984.

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

[54] K.M. Lynch and M.T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. J. Robotics Research*, 15(6):533–556, 1996.

[55] J. Maver and R. Bajcsy. Occlusions as a guide for planning the next view. *IEEE Trans. Pattern Analysis & Machine Intelligence*, 15(5):417–433, May 1993.

[56] R. Murrieta-Cid, H.H. González-Baños, and B. Tovar. A reactive motion planner to maintain visibility of unpredictable targets. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4242–4248, 2002.

[57] R. Murrieta-Cid, A. Sarmiento, S. Bhattacharya, and S. Hutchinson. Maintaining visibility of a moving target at a fixed distance: The case of observer bounded speed. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 479–484, 2004.

[58] N.J. Nilsson. A mobile automation: An application of intelligence techniques. In *Proc. Int. Jnt. Conf. on Artificial Intelligence*, pages 509–520, 1969.

[59] C. Ó'Dúnlaing and C.K. Yap. A retraction method for planning the motion of a disc. *J. Algorithms*, 6:104–111, 1982.

[60] N.P. Papanikolopoulos, P.K. Khosla, and T. Kanade. Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision. *IEEE Trans. Robotics & Automation*, 9(1):14–35, 1993.

[61] R. Pito. A solution to the next best view problem for automated CAD model acquisition of free-form objects using range cameras. Technical Report 95-23, GRASP Lab, U. of Pennsylvania, May 1995.

[62] R. Pito. A sensor based solution to the next best view problem. In *Proc. IEEE Int. Conf. on Pattern Recognition*, volume 1, pages 941–5, 1996.

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

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

[65] G. Sánchez and J.C. Latombe. On delaying collision checking in PRM planning—application to multi-robot coordination. *Int. J. Robotics Research*, 21(1):5–26, 2002.

[66] F. Schwarzer, M. Saha, and J.C. Latombe. Exact collision checking of robot paths. In J.D. Boissonnat et al., editors, *Algorithmic Foundations of Robotics V*, pages 25–41. Springer, 2002.

[67] A. Schweikard, J. R. Adler, and J.C. Latombe. Motion planning in stereotaxic radiosurgery. *IEEE Trans. Robotics & Automation*, 9(6):764–774, 1993.

[68] S. Sekhavat, P. Švestka, J.-P. Laumond, and M.H. Overmars. Multi-level path planning for non-holonomic robots using semi-holonomic subsystems. In J.-P. Laumond et al., editors, *Algorithms for Robotic Motion and Manipulation: 1996 Workshop on the Algorithmic Foundations of Robotics*, pages 79–96. A. K. Peters, Wellesley, MA, 1996.

[69] T. Shermer. Recent results in art galleries. *Proc. IEEE*, 80(9):1384–1399, 1992.

[70] T. Simeon, J.P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *J. Advanced Robotics*, 14(6):477–494, 2000.

[71] A.P. Singh, J.C. Latombe, and D.L. Brutlag. A motion planning approach to flexible ligand binding. In *Proc. Int. Conf. on Intelligent Systems for Molecular Biology*, pages 252–261, 1999.

[72] G. Song and N.M. Amato. Using motion planning to study protein folding pathways. In *Proc. ACM Int. Conf. on Computational Biology (RECOMB)*, pages 287–296, 2001.

[73] Švestka and M.H. Overmars. Motion planning for car-like robots, a probabilistic learning approach. *Int. J. Robotics Research*, 16:119–143, 1997.

[74] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[75] P. Valtr. Guarding galleries where no point sees a small area. *Israel J. Mathematics*, 104:1–16, 1998.

[76] L. Wixson. Viewpoint selection for visual search. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*, pages 800–805, 1994.

[77] Y. Yang and O. Brock. Adapting the sampling distribution in PRM planners based on an approximated medial axis. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2004.

[78] Y. Yu and K. Gupta. C-space entropy: A measure for view planning and exploration for general robot-sensor systems in unknown environments. *Int. J. Robotics Research*, 23(12):1197, 2004.

Héctor H. González-Baños: Héctor H. González-Baños is a Senior Research Scientist at Honda Research Institute USA Inc. He received his Ph.D. in electrical engineering from Stanford University in 2001. His research interests are robotics and computer vision.

David Hsu: David Hsu is the Sung Kah Kay Assistant Professor of Computer Science at the National University of Singapore and a fellow of the Singapore-MIT Alliance. He received B.Sc. in computer science & mathematics from the University of British Columbia in 1995 and Ph.D. in computer science from Stanford University in 2000. From 2000 to 2001, He was a Member of Research Staff at Compaq Computer Corporation's Cambridge Research Lab. He moved to Singapore in 2002. His main research interests include robotics, motion planning, computational biology, and geometric algorithms.

Jean-Claude Latombe: Jean-Claude Latombe is the Kumagai Professor of Computer Science at Stanford University. He received his PhD from the National Polytechnic Institute of Grenoble (INPG) in 1977. He was on the faculty of INPG from 1980 to 1984, when he joined Industry and Technology for Machine Intelligence (ITMI), a company that he had co-founded in 1982. He moved to Stanford in 1987. At Stanford, he served as the Chairman of the Computer Science Department from 1997 to 2001, and on the BioX Leadership Council from 2002 to 2004. His main research interests are in artificial intelligence, robotics, motion planning, computational biology, computer-aided surgery, and graphic animation.

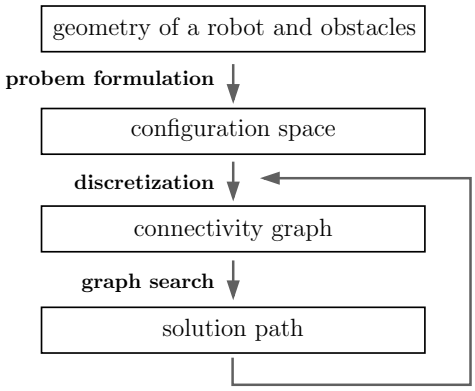


Figure 1. A common framework for path planning.

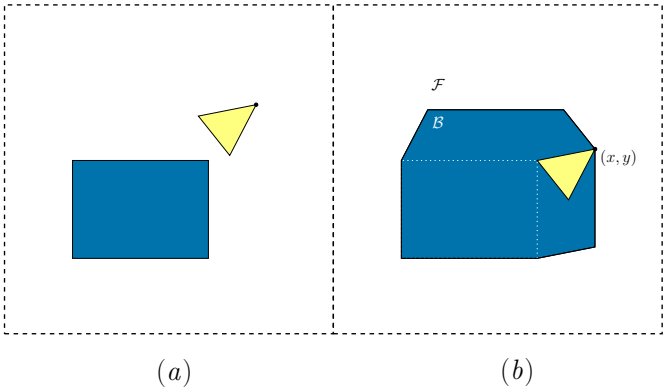


Figure 2. A robot translating in the plane. (*a*) The triangular robot moves in an environment with a single rectangular obstacle. (*b*) The configuration space of the robot. The configuration of the robot is represented by the position (x, y) of a reference point in the robot.

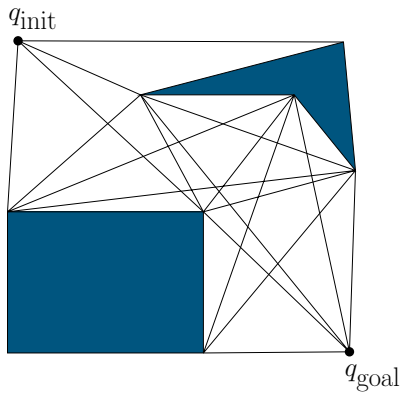


Figure 3. The visibility graph of a configuration space.

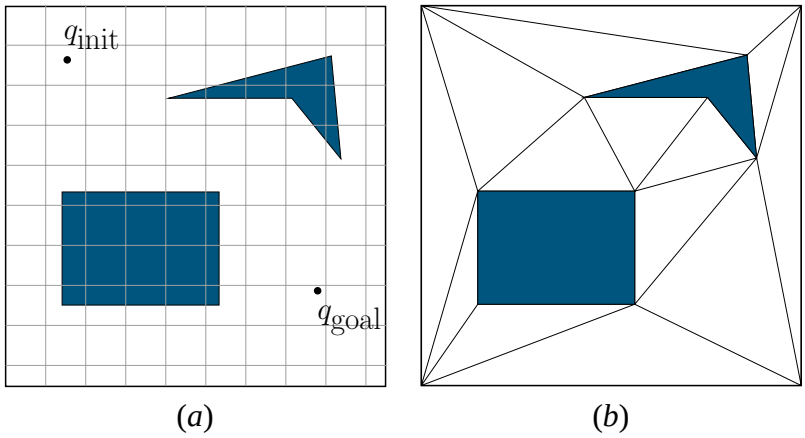


Figure 4. Cell decomposition with (*a*) a fixed-resolution grid and (*b*) a triangulation.

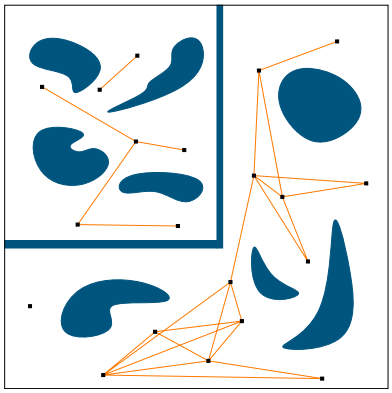


Figure 5. A probabilistic roadmap generated by the uniform sampling strategy for multi-query planning in a 2-D configuration space.

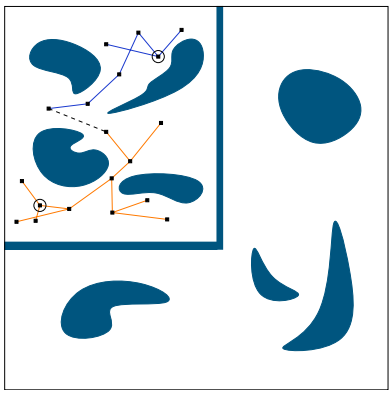


Figure 6. A roadmap for single-query planning in a 2-D configuration space. The two circles mark q_{init} and q_{goal}.

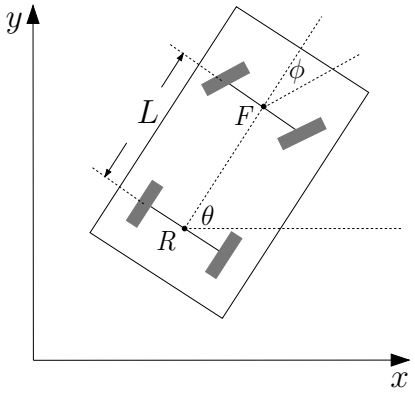


Figure 7. A simplified model for a car-like robot.

Figure 8. A trailer-truck carrying aircraft components on a narrow road with many obstacles nearby.

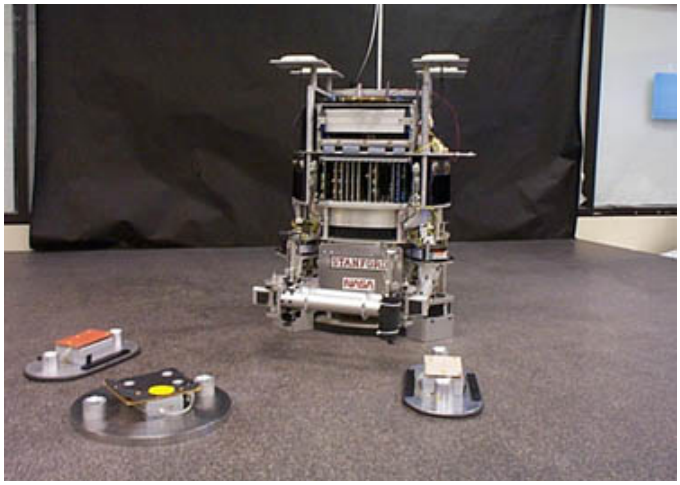


Figure 9. An air-cushioned robot among moving obstacles.

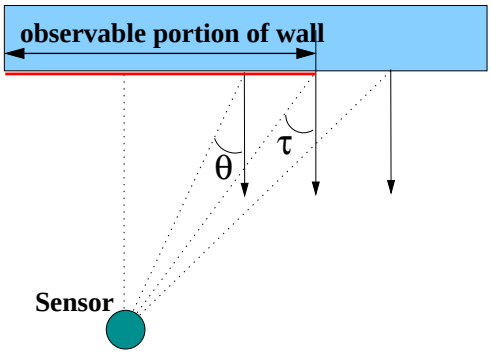
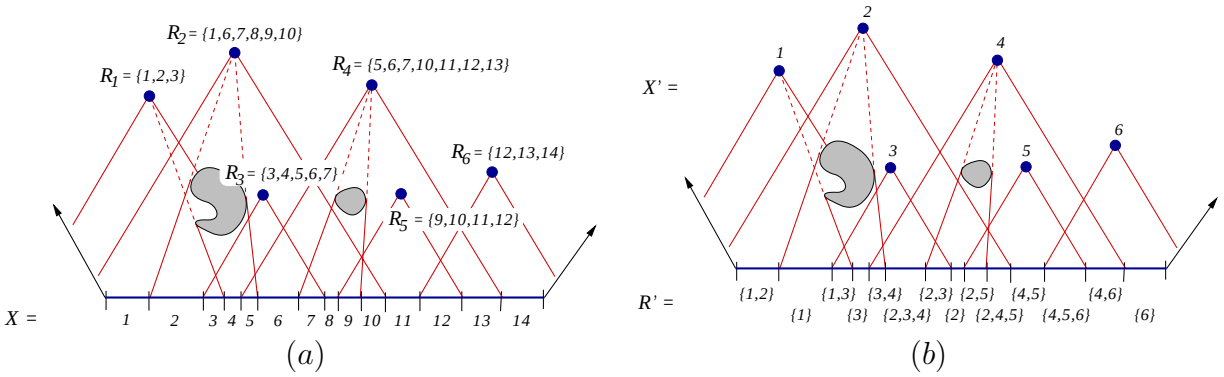
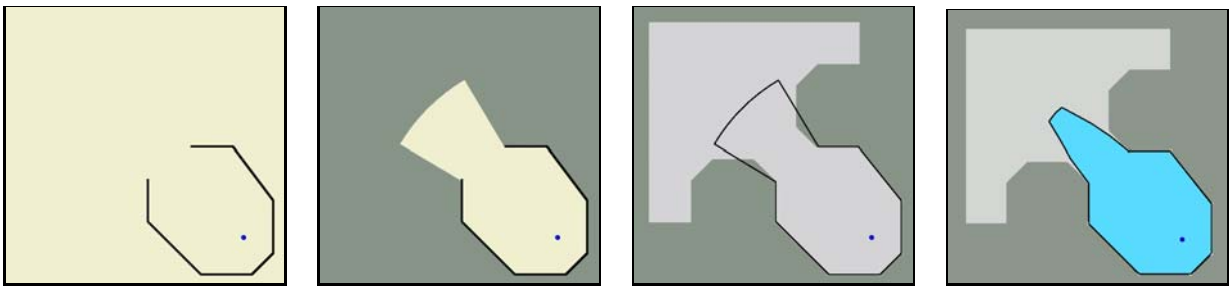


Figure 10. The incidence constraint of laser range sensors: wall sections are seen reliably, only if $|\ \theta\ | \leq \tau$.



(a) (b)

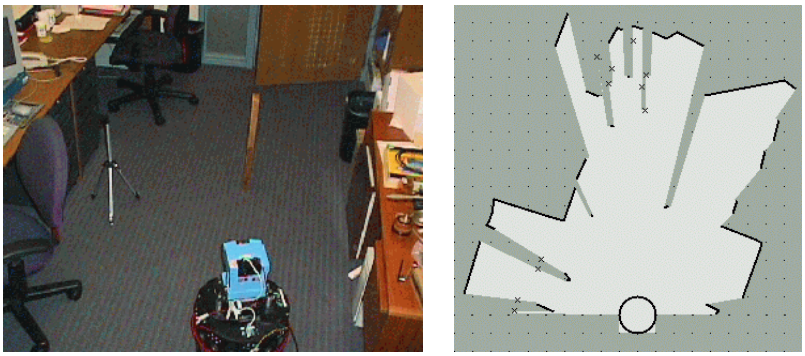
Figure 11. Sensor placement seen as a set system. (a) Each boundary edge is decomposed into cells. All points within the same cell are visible to the same subset of $\mathcal{G}_{\text{sam}}$. Each cell is then labeled with an integer and grouped under X. A subset $R_i \subseteq X$ is the set of cells visible from the sample point g_i. (b) In the *dual* representation, candidate sensor locations are grouped and labeled under X'. Each set $R'_i \in \mathcal{R}'$ is the set of locations covering cell i in the boundary decomposition.



(a) (b) (c) (d)

Figure 12. The effect of incidence constraints on safe regions.

Figure 13. A partial map of a wing of the Computer Science Building at Stanford University. The total length of the circuit is approximately 40 meters. The circled region corresponds to the last local measurement.



(a) (b)

Figure 14. Measuring the visibility region with a laser range sensor.

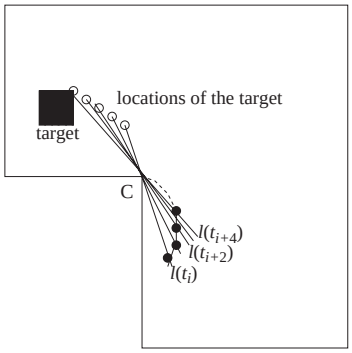


Figure 15. The visibility sweeping line $\ell(t)$ going through the target position at time t and the reflex vertex C. In this example, the observer must remain above $\ell(t)$ to keep the target visible.

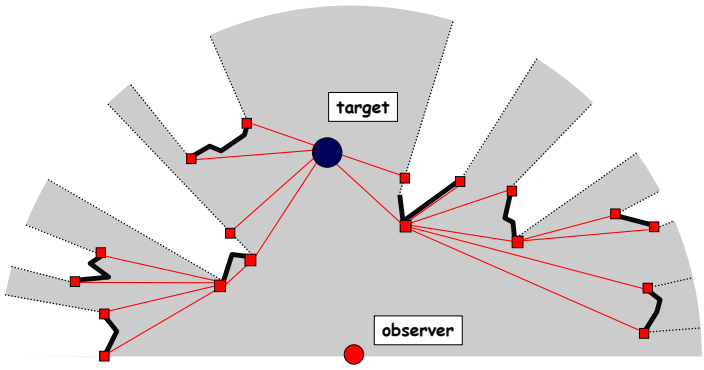


Figure 16. An example of the escape-path tree. The area in gray is the observer's visibility region, with obstacle boundaries shown in bold. The squares indicate the nodes of the tree.