Robot Algorithms

Konstantinos Tsianos, Rice University
Dan Halperin, Tel-Aviv University
Lydia Kavraki, Rice University
Jean-Claude Latombe, Stanford University

0.1 Introduction

People tend to have very different perceptions of what a robot is. For some people a robot is an intelligent and sophisticated machine. A robot must have the capability to move autonomously and make decisions on how to accomplish its task without any human intervention. From this perspective, examples would include mobile robots used in space exploration, or looking a bit into the future, autonomous intelligent cars. Other people think of robots more in the context of industrial automation. In this context, a robot is typically a robotic arm used in assembly lines, for example in car manufacturing. Those robots tend to perform the same repetitive motions in their highly controlled and predictable environment. Precision and accuracy are paramount for this kind of robots. Finally, it is not uncommon to treat a part or even a molecule as a robot. A major issue in production lines is how to process parts that arrive at random orientations and have to be oriented in a certain way before they are used for assembly. During the last few years, it has become a very important area of research to identify how big molecules - such as proteins - move, by adapting ideas that were initially developed for robots. This chapter covers some of the fundamental principles and presents several examples of algorithms that are currently used in robotics.

Robot algorithms differ in significant ways from traditional computer algorithms. The latter have full control over, and perfect access to the data they use, letting aside, for example, problems related to floating-point arithmetic. In contrast, robot algorithms eventually apply to physical objects in the real world, which they attempt to control despite the fact that these objects are subject to the independent and imperfectly modeled laws of nature. Data acquisition through sensing is also local and noisy. Robot algorithms hence raise controllability (or reachability) and observability (or recognizability) issues that are classical in control theory, but not present in computer algorithms.

On the other hand, control theory often deals with well-defined processes in strictly confined environments. In contrast, robot tasks tend to be underspecified, which requires addressing combinatorial issues ignored in control theory. For instance, to reach a goal among **obstacles** that are not represented in the input model, but are sensed during execution, a robot must search “on the fly” for a collision-free

path, a notoriously hard computational problem.

This blend of control and computational issues is perhaps the main characteristic of robot algorithms. It is presented at greater length in Section 0.2, along with other features of these algorithms. Section 0.3 then surveys specific areas of robotics (e.g., part manipulation, assembly sequencing, motion planning, sensing) and presents algorithmic techniques that have been developed in those areas.

0.2 Underlying Principles

0.2.1 Robot Algorithms Control

The primary goal of a robot algorithm is to describe a procedure for controlling a subset of the real world—the **workspace**—in order to achieve a given goal, say, a spatial arrangement of several physical objects. The real world, which is subject to the laws of nature (such as gravity, inertia, friction), can be regarded as performing its own actions, for instance, applying forces. These actions are not arbitrary, and to some extent, they can be modeled, predicted, and controlled. Thus, a robot algorithm should specify robot's operations whose combination with the (re-)actions of the real world will result in achieving the goal. Note that the robot is itself an important object in the workspace; for example, it should not collide with obstacles. Therefore, the algorithm should also control the relation between the robot and the workspace. The robot's internal controller, which drives the actuators and preprocesses sensory data, defines the primitive operations that can be used to build robot algorithms.

The design of a robot algorithm requires identifying a set of relevant states of the workspace (one being the goal) and selecting operations to take the workspace through a sequence of states ending at the goal. But, due to various inaccuracies (one is in modeling physical laws), an operation may transform a state into one among several possible states. The algorithm can then use sensing to refine its knowledge during execution. In turn, because workspace sensing is imperfect, a state may not be directly recognizable, meaning that no combination of sensors may be capable to return the state's identity. As a result, the three subproblems—choosing pertinent states, selecting operations to transit among these states toward the goal, and constructing state-recognition functions—are strongly interdependent and cannot be solved sequentially.

To illustrate part of the above discussion, consider the task of orienting a convex polygonal part P on a table using a robot arm equipped with a parallel-jaw gripper, to a desired goal orientation θ_g. This is a typical problem in industrial part feeding ("Part Feeding"). If an overhead vision system is available

Figure 1: Orienting a convex polygonal part. Starting from four different initial positions, two squeezing operations can bring the part to the correct orientation [20]

to measure P's orientation, we can use the following (simplified) algorithm:

ORIENT(P, θ_g)

1 Measure P's initial orientation θ_i

2 Move the gripper to the grasp position of P

3 Close the gripper

4 Rotate the gripper by $\theta_g - \theta_i$

5 Open the gripper

6 Move the gripper to a resting position

The states of interest are defined by the orientations of P, the position of the gripper relative to P, and whether the gripper holds P or does not. (Only the initial and goal orientations, θ_i and θ_g, are explicitly considered.) Step 1 acquires the initial state. Step 4 achieves the goal state. Steps 2 and 3 produce intermediate states. Steps 5 and 6 achieve a second goal not mentioned above, that the robot be away from P at the end of the orientation operation.

A very different algorithm for this same part-orienting task consists of squeezing P several times between the gripper's jaws, at appropriately selected orientations of the gripper (see Fig. 1). This algorithm, which requires no workspace sensing, is based on the following principle. Let P be at an arbitrary initial orientation. Any squeezing operation will achieve a new orientation that belongs to a set of $2n$ (n being the number of sides of P) possible orientations determined by the geometry of P and the orientation of the jaws. If P is released and squeezed again with another orientation of the gripper, the set of possible orientations of P can be reduced further. For any n-sided convex polygon P, there is a sequence of $2n - 1$ squeezes that achieves a single orientation of P (up to symmetries), for an arbitrary

(a) (b)

Figure 2: Goal recognition in mobile robot navigation

initial orientation of P [20].

The states considered by the second algorithm are individual orientations of P and sets of orientations. The state achieved by each squeeze is determined by the jaws' orientation and the previous state. Its prediction is based on understanding the simple mechanics of the operation. The fact that any convex polygon admits a finite sequence of squeezes ending at a unique orientation guarantees that any goal is reachable from any state. However, when the number of parameters that the robot can directly control is smaller than the number of parameters defining a state, the question of whether the goal state is reachable is more problematic (see "Motion Planning").

State recognition can also be difficult. To illustrate, consider a mobile robot navigating in an office environment. Its controller uses dead-reckoning techniques to control the motion of the wheels. But these techniques yield cumulative errors in the robot's position with respect to a fixed coordinate system. For better localization, the robot algorithm may sense environmental features (e.g., a wall, a door). However, because sensing is imperfect, a feature may be confused with a similar feature at a different place; this may occasionally cause a major localization mistake. Thus, the robot algorithm must be designed so that enough environmental features will be sensed to make each successive state reliably recognizable.

To be more specific, consider the workspace of Fig. 2. Obstacles are shown in bold lines. The robot is modeled as a point with perfect touch sensing. It can be commanded to move along any direction $\phi \in [0, 2\pi)$ in the plane, but imperfect control makes it move within a cone $\phi \pm \theta$, where the angle θ models directional uncertainty. The robot's goal is to move into G, the goal state, which is a subset of the wall W (for instance, G is an outlet for recharging batteries). The robot's initial location is not precisely known: it is anywhere in the disk I, the initial state. One candidate algorithm (illustrated in Fig. 2a) first commands the robot to move perpendicularly to W until it touches it. Despite directional uncertainty, the robot is guaranteed to eventually touch W, somewhere in the region denoted by H. From

state H, it can slide along the wall (using touch to maintain contact) toward G. The robot is guaranteed to eventually reach G. But can it reliably recognize this achievement? The answer depends on the growth of the dead-reckoning localization error as the robot moves along W. Clearly, if this error grows by more than half the difference in the size of G and H, G is not reliably recognizable.

An alternative algorithm is possible, using the wall W' (Fig. 2b). It commands the robot to first move toward W' until it touches it and then slide along it toward W. At the end of W', it continues heading toward W, but with directional uncertainty. The robot is nevertheless guaranteed to be in G when it senses that it has touched W.

0.2.2 Robot Algorithms Plan

Consider the following variant of the part-orienting task. Parts are successively fed with arbitrary orientations on a table by an independent machine. They have different and arbitrary convex polygonal shape, but whenever a part arrives, the feeding machine provides a geometric model of the part to the robot, along with its goal orientation. In the absence of a vision sensor, the multi-squeeze approach can still be used, but now the robot algorithm must include a planner to compute automatically the successive orientations of the gripper.

As another example, consider the pick-and-place task which requires a robot arm to transfer an object from one position to another. If the obstacles in the workspace are not known in advance, the robot algorithm needs sensing to localize them, as well as a planner to generate a collision-free path. If all obstacles cannot be sensed at once, the algorithm may have to interweave sensing, planning, and acting.

The point is that, for most tasks, the set of states that may have to be considered at execution time is too large to be explicitly anticipated. The robot algorithm must incorporate a planner. In first approximation, the planner can be seen as a separate algorithm that automatically generates a control algorithm for achieving a given task. The robot algorithm is the combination of the planning and the control algorithms. More generally, however, it is not sufficient to invoke the planner once, and planning and control are interleaved. The effect of planning is to dynamically change the control portion of the robot algorithm, by changing the set of states of the workspace that are explicitly considered.

Planning, which often requires exploring large search spaces, raises critical complexity issues. For example, finding a collision-free path for a three-dimensional **linkage** among polyhedral obstacles is PSPACE-hard [47], and the proof of this result provides strong evidence that any complete algorithm will require exponential time in the **number of degrees of freedom**. Planning the motion of a point robot among polyhedral obstacles, with bounded uncertainty in control and sensing, is NEXPTIME-

hard [10].

The computational complexity of planning leads to looking for efficient solutions to restricted problems. For example, for part orienting, there exists a complete planning algorithm that computes a sequence of squeezes achieving a single orientation (up to symmetries) of a given convex polygonal part in quadratic time in the number of sides of the part [20]. Another way of dealing with complexity is to trade off completeness against time, by accepting weaker variants of completeness. A **complete planner** is guaranteed to find a solution whenever one exists, and to notify that there exists none otherwise. A weaker variant is probabilistic completeness: if there exists a solution, the planner will find one only with high probability. This variant can be very useful if one can show that the probability of finding a solution (when one exists) tends rapidly toward 1 as the running time increases. In Section 0.3 we will present planning algorithms that embed similar approaches.

The complexity of a robot algorithm has also some interesting relations with the reachability and recognizability issues introduced in the previous subsection. We will mention several such relations in Section 0.3 (in particular, in "Motion Planning").

The potentially high cost of planning and the fact that it may often have to be done on-line raise an additional issue. A robot algorithm must carefully allocate time between computations aimed at planning and computations aimed at controlling and sensing the workspace. If the workspace is changing,(say, under the influence of other agents), spending too much time on planning may result in obsolete control algorithms; on the other hand, not enough planning may yield irreversible failures [76]. The problem of allocating time between planning and control remains poorly understood, though several promising ideas have been proposed. For example, it has been suggested to develop planners that return a plan in whatever amount of time is allocated to them and can be called back later to incrementally improve the previous plan if more time is allocated to planning [7]. Deliberative techniques have been proposed to decide what amount of time should be given to planning and control and update this decision as more information is collected [43].

0.2.3 Robot Algorithms Reason About Geometry

Imagine a robot whose task is to maintain a botanic garden. To set and update its goal agenda, this robot needs knowledge in domains like botany and fertilization. The algorithms using this knowledge can barely be considered parts of a robot algorithm. But, on the other hand, all robots, including gardener robots, accomplish tasks by eventually moving objects (including themselves) in the real world. Hence, at some point, all robots must reason about the geometry of their workspace. Actually, geometry is not

enough, since objects have mass inducing gravitational and inertial forces, while contacts between objects generate frictional forces. All robots must therefore reason with classical mechanics. However, Newtonian concepts of mechanics translate into geometric constructs (e.g., forces are represented as vectors), so that most of the reasoning of a robot eventually involves dealing with geometry.

Computing with continuous geometric models raises discretization issues. Several planners computing a robot's path, discretize the robot's free space in order to build a connectivity graph to which well-known search algorithms can be applied.

Consider for example the **configuration space**. A formal definition of this very important concept is given in Section 0.3.3, but for now just think of it as the set of all collision-free poses (**configurations**) of the robot. One discretization approach is to place a fine regular grid across configuration space and search that grid for a sequence of adjacent points in free space. The grid is just a computational tool and has no physical meaning. Its resolution is arbitrarily chosen despite its critical role in the computation: if it is too coarse, planning is likely to fail; if it is too fine, planning will take too much time. Instead, criticality-driven discretizations have been proposed, whose underlying principle is widely applicable. They consist of partitioning the continuous space of interest into cells, such that some pertinent property remains invariant over each cell and changes when the boundary separating two cells is crossed. The second part-orienting algorithm in "Robot Algorithms Control" is based on such a discretization. The set of all possible orientations of the part is represented as the unit circle (the cyclic interval $[0, 2\pi)$). For a given orientation of the gripper, this circle can be partitioned into arcs such that, for all initial orientations of the part in the same arc, the part's final orientation will be the same after the gripper has closed its jaws. The final orientation is the invariant associated with the cell. From this decomposition, it is a relatively simple matter to plan a squeezing sequence.

Several such criticality-driven discretizations have been proposed for path planning, assembly sequence planning, motion planning with uncertainty, robot localization, object recognition, and so on, as will be described in Section 0.3. Several of them use ideas and tools originally developed in computational geometry, for instance: plane sweep, constructing arrangements, constructing Voronoi diagrams.

Robot algorithms often require dealing with high-dimensional geometric spaces. Although criticality-based discretization methods apply to such spaces in theory (for instance, see [49]), their computational complexity is then overwhelming. This has led the development of randomized techniques that efficiently approximate the topology and geometry of such spaces by random discretization. Such techniques have been particularly successful for building probabilistically complete planners ("Sampling-Based Algorithms").

0.2.4 Robot Algorithms Have Physical Complexity

Just as the complexity of a computation characterizes the amount of time and memory this computation requires, we can define the physical complexity of a robot algorithm by the amount of physical resources it takes, e.g., the number of "hands," the number of motions, or the number of beacons. Some resources, like the number of motions, relate to the time spent executing the algorithm. Others, like the number of beacons, relate to the engineering cost induced by the algorithm. For example, one complexity measure of the multi-squeeze algorithm to orient a convex polygon ("Robot Algorithms Control") is the maximal number of squeeze operations this algorithm performs. As another example, consider an assembly operation merging several parts into a subassembly. The number of subsets of parts moving relative to each other (each subset moving as a single rigid body) measures the number of hands necessary to hold the parts during the operation. The number of hands required by an assembly sequence is the maximal number of hands needed by an operation, over all the operations in the sequence ("Assembly Sequencing"). The number of fingers to safely grasp or fixture an object is another complexity measure ("Grasping").

Though there is a strong conceptual analogy between computational and physical complexities, there are also major differences between the two notions. Physical complexity must be measured along many more dimensions than computational complexity. Moreover, while computational complexity typically measures an asymptotic trend, a tighter evaluation is usually needed for physical complexity since robot tasks involve relatively few objects.

One may also consider the inherent physical complexity of a task, a notion analogous to the inherent complexity of a computational problem. For example, to orient a convex polygon with n sides, $2n-1$ squeezes may be needed in the worst case; no correct algorithm can perform better in all cases. By generating all feasible assembly sequences of a product, one could determine the number of hands needed by each sequence and return the smallest number. This number is a measure of the inherent complexity of assembling the product. No robot algorithm to assemble this product can require fewer hands.

Evaluating the inherent physical complexity of a task may lead to redefining the task, if it turns out to be too complex. For example, it has been shown that a product made of n parts may need up to n hands for its assembly ("Grasping"), thus requiring the delicate coordination of $n-1$ motions. Perhaps a product whose assembly requires several hands could be redesigned so that two hands are sufficient, as is the case for most industrial products. Indeed, designers strive to reduce physical complexity along various dimensions. For instance, many mass-produced devices are designed to be assembled with translations only, along very few directions (possibly a single one). The inherent physical complexity of a robot task is not a recent concept, but its formal application to task analysis is [54].

An interesting issue is how the computational and physical complexities of a robot algorithm relate to each other. For example, planning for mobile robot navigation with uncertainty is a provably hard computational problem ("Dealing with Uncertainties in Motion and Sensing"). On the other hand, burying wires in the ground or placing enough infrared beacons allows robots to navigate reliably at small computational cost. But isn't it too much? Perhaps the intractability of motion planning with uncertainty can be eliminated with less costly engineering.

0.3 State of the Art and Best Practices

Robotics is a broad domain of research. In this subsection we study a number of specific areas: part manipulation, assembly sequencing and motion planning. For each area, we introduce problems and survey key algorithmic results.

Although we present the current research according to problem domain, there are several techniques that cross over many domains. One of the most frequently applied methods in robotics is the criticality-based discretization mentioned in "Robot Algorithms Reason About Geometry." This technique allows us to discretize a continuous space without giving up the completeness or exactness of the solution. It is closely related to the study of arrangements in computational geometry [23]. When criticality-based discretization is done in a space representing all possible motions, it yields the so-called "nondirectional" data structures, which is another prevailing concept in robot algorithms and is exemplified in detail in "Monotone Two-Handed Assembly Sequence."

Randomization is another important paradigm in robotics. Randomized techniques have made it possible to cope practically with robot motion planning with many degrees of freedom (Section on "Sampling-Based Algorithms"). Also, randomized algorithms are often simpler than their deterministic counterparts and hence better candidates for efficient implementation. Randomization has recently been applied to solving problems in grasping as well as in many other areas that involve geometric reasoning.

Throughout this section we interchangeably use the terms "body," "physical object," and "part" to designate a rigid physical object modeled as a compact manifold with boundary $B \subset \mathbb{R}^k$ ($k = 2$ or 3). B's boundary is also assumed piecewise-smooth.

0.3.1 Part Manipulation

Part manipulation is one of the most frequently performed operations in industrial robotics: parts are grasped from conveyor belts, they are oriented prior to feeding assembly workcells, and they are immo-

bilized for machining operations.

Grasping

Part grasping has motivated various kinds of research, including the design of versatile mechanical hands, as well as simple, low-cost grippers. From an algorithmic point of view, the main goal is to compute "safe" grasps for an object whose model is given as input.

Force-Closure Grasp Informally, a grasp specifies the positions of "fingers" on a body B. A more formal definition uses the notion of a wrench, a pair $[\boldsymbol{f}, \boldsymbol{p} \times \boldsymbol{f}]$, where $\boldsymbol{p}$ denotes a point on the boundary ∂B of B, represented by its coordinate vector in a frame attached to B, $\boldsymbol{f}$ designates a force applied to B at $\boldsymbol{p}$, and $\times$ is the vector cross-product. If $\boldsymbol{f}$ is a unit vector, the wrench is said to be a unit wrench. A finger is any tool that can apply a wrench.

A grasp of B is a set of unit wrenches $\boldsymbol{w}_i = [\boldsymbol{f}_i, \boldsymbol{p}_i \times \boldsymbol{f}_i]$, $i = 1, \ldots, p$, defined on B. For each $\boldsymbol{w}_i$, if the contact is frictionless, $\boldsymbol{f}_i$ is normal to ∂B at $\boldsymbol{p}_i$; otherwise, it can span a friction cone (Coulomb law of friction).

The notion of a safe grasp is captured by force closure. A force-closure grasp $\{\boldsymbol{w}_i\}_{i=1,\ldots,p}$ on B is such that, for any arbitrary wrench $\boldsymbol{w}$, there exists a set of real values $\{f_1, \ldots, f_p\}$ achieving $\Sigma_{i=1}^{p} f_i \boldsymbol{w}_i = -\boldsymbol{w}$. In other words, a force-closure grasp can resist any external wrench applied to B. If contacts are nonsticky, we require that $f_i \geq 0$, for all $i = 1, \ldots, p$, and the grasp is called positive. Here, we only consider positive grasps. A form-closure grasp is a positive force-closure grasp when all finger-body contacts are frictionless.

Size of a Form/Force-Closure Grasp The following results characterize the physical complexity of achieving a safe grasp [40]:

- Bodies with rotational symmetry (e.g., discs in 2-space, spheres and cylinders in 3-space) admit no form-closure grasps.
- All other bodies admit a form-closure grasp with at most 4 fingers in 2-space and 12 fingers in 3-space.
- All polyhedral bodies have a form-closure grasp with 7 fingers.
- With frictional finger-body contacts, all bodies admit a force-closure grasp that consists of 3 fingers in 2-space and 4 fingers in 3-space.

Testing Force Closure A necessary and sufficient condition for a grasp $\{\boldsymbol{w}_i\}_{i=1,\ldots,p}$ to achieve force closure in 2-space (respectively, 3-space) is that the finger wrenches $\boldsymbol{w}_i$ span a space F of dimension 3 (respectively, 6) and that a strictly positive linear combination of them be zero. In other words, the origin of F (null wrench) should lie in the interior of the convex hull of the finger wrenches [40]. This condition provides an effective test for deciding in constant time whether a given grasp achieves force closure.

Computing Form/Force Closure Grasps Most research has concentrated on computing grasps with 2 to 4 nonsticky fingers. Algorithms that compute a single force-closure grasp of a polygonal/polyhedral part in time linear in the part's complexity have been derived in [40].

Finding the maximal regions on a body where fingers can be positioned independently while achieving force closure makes it possible to accommodate errors in finger placement. Geometric algorithms for constructing such regions are proposed in [42] for grasping polygons with two fingers (with friction) and four fingers (without friction), and for grasping polyhedra with three fingers (with frictional contact capable of generating torques) and seven fingers (without friction). Grasping of curved obstacles is addressed in [45].

Fixturing

Most manufacturing operations require fixtures to hold parts. To avoid the custom design of fixtures for each part, modular reconfigurable fixtures are often used. A typical modular fixture consists of a workholding surface, usually a plane, that has a lattice of holes where locators, clamps, and edge fixtures can be placed. Locators are simple round pins, while clamps apply pressure on the part.

Contacts between fixture elements and parts are generally assumed frictionless. In modular fixturing, contact locations are restricted by the lattice of holes, and form closure cannot always be achieved. In particular, when three locators and one clamp are used on a workholding plane, there exist polygons of arbitrary size for which no form-closure fixture exists; but, if parts are restricted to be rectilinear with all edges longer than four lattice units, a form-closure fixture always exists [56].

When the fixturing kit consists of a latticed workholding plane, three locators, and one clamp, it is possible to find all possible placements of a given part on the workholding surface where form closure can be achieved, along with the corresponding positions of the locators and the clamp [9].

Algorithms for computing all placements of (frictionless) point fingers that put a polygonal part in form closure and all placements of point fingers that achieve "2nd-order immobility" [96] of a polygonal part are presented in [95]. Immobilizing hinged parts is discussed in [97].

Part Feeding

Part feeders account for a large fraction of the cost of a robotic assembly workcell. A typical feeder must bring parts at subsecond rates with high reliability. An example of a flexible feeder is given in [20] and described in "Robot Algorithms Control" above.

Part feeding often relies on nonprehensile manipulation, which exploits task mechanics to achieve a goal state without grasping and frequently allows accomplishing complex feeding tasks with simple mechanisms [2]. Pushing is one form of nonprehensile manipulation [90]. Work on pushing originated in [38] where a simple rule is established to qualitatively determine the motion of a pushed object. This rule makes use of the position of the center of friction of the object on the supporting surface. Related results include a planning algorithm for a robot that tilts a tray with a planar part of known shape to orient it to a desired orientation [16] and an algorithm that computes the sequence of motions of a single articulated fence on a conveyor belt to achieve a goal orientation of an object [2]. A variety of interesting results on part feeding appear in the thesis [98].

0.3.2 Assembly Sequencing

Most mechanical products consist of multiple parts. The goal of assembly sequencing is to compute both an order in which parts can be assembled and the corresponding required movements of the parts. Assembly sequencing can be used during design to verify that the product will be easy to manufacture and service. An assembly sequence is also a robot algorithm at a high level of abstraction since parts are assumed free-flying, massless geometric objects.

Notion of an Assembly Sequence

An assembly A is a collection of bodies in some given relative placements. Subassemblies are separated if they are arbitrarily far apart from one another. An assembly operation is a motion that merges s separated subassemblies ($s \geq 2$) into a new subassembly, with each subassembly moving as a single body. No overlapping between bodies is allowed during the operation. The parameter s is called the number of hands of the operation. (Hence, a hand is seen here as a grasping or fixturing tool that can hold an arbitrary number of bodies in fixed relative placements.) Assembly partitioning is the reverse of an assembly operation.

An assembly sequence is a total ordering on assembly operations that merges the separated parts composing an assembly into this assembly. The maximum, over all the operations in the sequence, of the

number of hands of an operation is the number of hands of the sequence.

A monotone assembly sequence contains no operation that brings a body to an intermediate placement (relative to other bodies), before another operation transfers it to its final placement. Therefore, the bodies in every subassembly produced by such a sequence are in the same relative placements as in the complete assembly. Note that a product may admit no monotone assembly sequence for a given number of hands, while it may admit such sequences if more hands are allowed.

Number of Hands in Assembly

The number of hands needed for various families of assemblies is a measure of the inherent physical complexity of an assembly task ("Robot Algorithms Have Physical Complexity"). It has been shown that an assembly of convex polygons in the plane has a two-handed assembly sequence of translations. In the worst case, s hands are necessary and sufficient for assemblies of s star-shaped polygons/polyhedra [41].

There exists an assembly of six tetrahedra without a two-handed assembly sequence of translations, but with a three-handed sequence of translations. Every assembly of five or fewer convex polyhedra admits a two-handed assembly sequence of translations. There exists an assembly of thirty convex polyhedra that cannot be assembled with two hands [51].

Complexity of Assembly Sequencing

When arbitrary sequences are allowed, assembly sequencing is PSPACE-hard. The problem remains PSPACE-hard even when the bodies are polygons, each with a constant maximal number of vertices [41]. When only two-handed monotone sequences are permitted and rigid motions are allowed, finding a partition of an assembly A into two subassemblies S and $A\backslash S$ is NP-complete. The problem remains NP-complete when both S and $A\backslash S$ are connected and motions are restricted to translations [26]. These latter results were obtained by reducing in polynomial time any instance of the 3-SAT problem to a mechanical assembly such that the partitioning of this assembly gives the solution of the 3-SAT problem instance.

Monotone Two-Handed Assembly Sequencing

A popular approach to assembly sequencing is disassembly sequencing. A sequence that separates an assembly to its individual components is first generated and next reversed. Most existing assembly sequencers can only generate two-handed monotone sequences. Such a sequence is computed by partitioning

the assembly and, recursively, the obtained subassemblies into two separated assemblies.

The nondirectional blocking graph (NDBG, for short) is proposed in [54] to represent all the blocking relations in an assembly. It is a subdivision of the space of all allowable motions of separation into a finite number of cells such that within each cell the set of blocking relations between all pairs of parts remain fixed. Within each cell this set is represented in the form of a directed graph, called the directional blocking graph (DBG). The NDBG is the collection of the DBGs over all the cells in the subdivision. The NDBG is one example of a data structure obtained by a criticality-driven discretization technique ("Robot Algorithms Reason About Geometry").

We illustrate this approach for polyhedral assemblies when the allowable motions are infinite translations. The partitioning of an assembly consisting of polyhedral parts into two subassemblies is done as follows. For an ordered pair of parts P_i, P_j, the 3-vector $\boldsymbol{d}$ is a blocking direction if translating P_i to infinity in direction $\boldsymbol{d}$ will cause P_i to collide with P_j. For each ordered pair of parts the set of blocking directions is constructed on the unit sphere $\mathcal{S}^2$ by drawing the boundary arcs of the union of the blocking directions (each arc is a portion of a great circle). The resulting collection of arcs partitions $\mathcal{S}^2$ into maximal regions such that the blocking relation among the parts is the same for any direction inside such a region.

Next, the blocking graph is computed for one such maximal region. The algorithm then moves to an adjacent region and updates the DBG by the blocking relations that change at the boundary between the regions, and so on. After each time the construction of a DBG is completed, this graph is checked for strong connectivity in time linear in the number its edges. The algorithm stops the first time it encounters a DBG that is not strongly connected and it outputs the two subassemblies of the partitioning. The overall sequencing algorithm continues recursively with the resulting subassemblies. If all the DBGs that are produced during a partitioning step are strongly connected, the algorithm notifies that the assembly does not admit a two-handed monotone assembly sequence with infinite translations.

Polynomial time algorithms are proposed in [54] to compute and exploit NDBGs for restricted families of motions. In particular, the case of partitioning a polyhedral assembly by a single translation to infinity, is analyzed in detail, and it is shown that partitioning an assembly of m polyhedra with a total of v vertices takes $O(m^2v^4)$ time. Another case is where the separating motions are infinitesimal rigid motions. Then partitioning the polyhedral assembly, can be carried out efficiently and practically [93]. With the above algorithms, every feasible disassembly sequence can be generated in polynomial time.

0.3.3 Motion Planning

Motion planning is central to robotics, as motion planning algorithms can provide robots with the capability of deciding automatically which motions to execute to reach their goal. In its simplest form, the problem is known as *path planning* because the question is to find a collision free path from an initial to a final position. A more challenging and general version is captured by the term *motion planning*. The distinctive difference is that it is not enough to come up with a collision free path. In addition the algorithm must compute the exact actions that the robot's actuators must perform to implement the computed path.

It is reasonable to expect that motion planning problems can present various difficulties depending on the type of robot at hand (e.g., planning for mobile robots, humanoids, reconfigurable robots, manipulators, etc.). Fortunately all those differences can be abstracted with the use of the configuration space that is described below. In the rest of this section several motion planning issues are discussed. For the case of just path planning, complete algorithms with a complexity analysis are presented together with more recent sampling-based approaches. Then the case of motion planning is considered under the prism of planning with differential constraints. The section ends with a discussion of several other motion planning variants that include planning in dynamic workspaces, planning with moving obstacles in the environment, multiple robots, movable objects, online planning, optimal planning and dealing with uncertainties.

Configuration Space

The configuration space has been informally described in Section 0.2.3. At first sight, planning for a car on the highway looks very different from planning for an industrial robotic arm. It is possible though, to define a powerful abstraction that hides the robot's specific details and transforms the problem into finding a solution for a point robot that has to move from one position to another in some new space, called the **configuration space**.

A **configuration** of a robot $\mathcal{A}$ is any mathematical specification of the position and orientation of every body composing $\mathcal{A}$, relative to a fixed coordinate system. The configuration of a single body is also called a placement or a pose.

The robot's **configuration space** is the set of all its configurations. Usually, it is a smooth manifold. We will always denote the configuration space of a robot by $\mathcal{C}$ and its dimension by m. Given a robot $\mathcal{A}$, we will let $\mathcal{A}(\boldsymbol{q})$ denote the subset of the workspace occupied by $\mathcal{A}$ at configuration $\boldsymbol{q}$.

The number of degrees of freedom of a robot is the dimension m of its configuration space. We abbreviate "degree of freedom" by dof.

Given an obstacle B_i in the workspace, the subset $CB_i \subseteq \mathcal{C}$ such that, for any $\boldsymbol{q} \in CB_i$, $\mathcal{A}(\boldsymbol{q})$ intersects B_i is called a **C-obstacle**. The union $CB = \cup_i CB_i$ plus the configurations that violate the mechanical limits of the robot's joints is called the **C-obstacle region**. The **free space** is the complement of the C-obstacle region in $\mathcal{C}$, that is, $\mathcal{C} \backslash CB$. In most practical cases, C-obstacles are represented as semialgebraic sets with piecewise smooth boundaries.

A robot's path is a continuous map $\tau : [0,1] \rightarrow \mathcal{C}$. A **free path** is a path that entirely lies in free space. A **semifree path** lies in the closure of free space.

After the configuration space has been defined, solving a path problem for some robot, becomes a question of computing a free or semifree path between two configurations. A **complete planner** is guaranteed to find a (semi)free path between two given configurations whenever such a path exists, and to report that no such path exists otherwise.

Complete Algorithms

Basic path planning for a three-dimensional linkage made of polyhedral links is PSPACE-hard [47]. The proof uses the robot's dofs to both encode the configuration of a polynomial space bounded Turing machine and design obstacles which force the robot's motions to simulate the computation of this machine. It provides strong evidence that any complete algorithm will require exponential time in the number of dofs. This result remains true in more specific cases, for instance when the robot is a planar arm in which all joints are revolute. However, it no longer holds in some very simple settings; for instance, planning the path of a planar arm within an empty circle is in P. For a collection of complexity results on motion planning see [30].

Most complete algorithms first capture the connectivity of the free space into a graph, either by partitioning the free space into a collection of cells (exact cell decomposition techniques), or by extracting a network of curves (roadmap techniques) [30]. General and specific complete planners have been proposed. The general ones apply to virtually any robot with an arbitrary number of dofs. The specific ones apply to a restricted family of robots usually having a fixed small number of dofs.

The general algorithm in [49] computes a cylindrical cell decomposition of the free space using the Collins method. It takes doubly exponential time in the number m of dofs of the robot. The roadmap algorithm in [10] computes a semifree path in time singly exponential in m. Both algorithms are polynomial in the number of polynomial constraints defining the free space and their maximal degree. Specific

algorithms have been developed mainly for robots with 2 or 3 dofs. For a k-sided polygonal robot moving freely in a polygonal workspace, the algorithm in [24] takes $O((kn)^{2+\epsilon})$ time, where n is the total number of edges of the workspace, for any $\epsilon > 0$.

Heuristic Algorithms

Several heuristic techniques have been proposed to speedup path planning. Some of them work well in practice, but they usually offer no performance guarantee.

Heuristic algorithms often search a regular grid defined over configuration space and generate a path as a sequence of adjacent grid points. The search can be guided by a potential field, a function over the free space that has a global minimum at the goal configuration. This function may be constructed as the sum of an attractive and a repulsive field [28]. The attractive field has a single minimum at the goal and grows to infinity as the distance to the goal increases. The repulsive field is zero at all configurations where the distance between the robot and the obstacles is greater than some predefined value, and grows to infinity as the robot gets closer to an obstacle. Ideally a robot could find its way to the goal by following a potential field that has only one global minimum at the goal (the potential field is then called a navigation function). Yet in general, the configuration space is usually a high dimensional and strangely shaped manifold, which makes it very hard to design potential fields that have no local minima where the robot may be trapped.

One may also construct grids at variable resolution. Hierarchical space decomposition techniques such as octrees and boxtrees have been used to that purpose [30].

Sampling-Based Algorithms

The complexity of path planning for robots with many dofs (more than 4 or 5) has led the development of computational schemes that trade off completeness against computational time. Those methods avoid computing an explicit representation of the free space. Instead their focus is on producing a graph that approximately captures the connectivity of the free space. As the title of this section suggests, those planners compute this graph by sampling the configuration space. The general idea is that a large number of random configurations are sampled from the configuration space. Then the configurations that correspond to collisions are filtered out and an attempt is made to connect pairs of collision free configurations to produce the edges of the connectivity graph.

One of the most popular such planners is the *Probabilistic RoadMap* (PRM) [4, 27]. The original algorithm consists of a learning and a querying phase. In the learning phase the roadmap is constructed.

A sampling strategy is used to generate a large number of configuration samples. The strategy can be just uniform random sampling, although many sophisticated strategies such as *bridge test, Gaussian, etc* have been proposed over the years [61, 62]. Then, an attempt is made to connect every free sample to its neighboring free samples. For this process a local planner is used that tries to interpolate between the two samples using a very simple strategy; for example a straight line in the configuration space. The intermediate configurations on the line must be checked for collisions [61, 62, 88] If no collisions are detected, the corresponding edge is added to the graph. After this phase is over, multiple planning queries can be solved using the same graph as a roadmap. The initial and goal configuration are connected to the roadmap by producing edges to one of their nearest neighbors on the roadmap. Then planning is reduced to a graph search problem which can be solved very efficiently.

PRM was initially intended as a multiquery planner where the same roadmap can be reused. In many cases though, it is required to solve only a single query as fast as possible and it is more important to explore the space towards some goal rather than trying to capture the connectivity of the whole space. Two very popular such planners are *Rapidly-Exploring Random Trees* (RRT) [68] and *Expansive-Spaces Trees* (EST) [1] [69] . Both end up building a tree T, rooted at the initial configuration. EST proceeds by extending new edges out the existing tree T. At each step a node q_1 on the tree is selected according to a probability distribution inverse to the local sampling density. Then a new random configuration q_2 is sampled in the neighborhood of q_1 and an attempt is made to create an edge between q_1 and q_2 using a local planner as before. Hopefully at some point there will be some configuration node in the tree that can be connected to the goal. RRT has a slightly different strategy. At each iteration, a new random configration q is produced and then an attempt is made to connect q to its nearest neighbor on the tree. A popular way to make this process more effective is to select the goal itself as q with some small probability. Several extentions of the above algorithms can be found in [61, 62].

There is a number of issues that can greatly affect the performance of sampling-based planners. The sampling strategy has already been discussed above, and is very important to produce useful samples. Moreover, sampling-based planners make extensive use of collision checking primitives [63]. This is a functionality that has to be available to provide the answer to the question of whether a configuration is in collision with an obstacle, or not. Finally, most of those planners need some functionality for nearest neighbors computations. This is a non-trivial task in general. The reason is that configuration spaces are usually strangely shaped manifolds where it is not always easy to define good distance metrics. This issue is discussed at the end of this chapter.

[1] The term EST was not used in the original paper. It was introduced later [62].

An interesting question is how many samples are required to solve a problem. In general this is related to the completeness properties of those algorithms. It is understood that if after some time the produced roadmap or tree is unable to solve the problem, it could just be that the sampler was "unlucky" and did not produce enough good samples yet. In principle sampling-based algorithms are not complete with respect the the definition given in previous sections. Instead most of them can be proven to be **probabilistically complete**. This is a weaker notion of completeness that guarantees that a solution will eventually be found if one exists. For this definition to be of practical importance, it is good to also demand that the convergence to the solution will be fast, i.e. exponential in the number of samples. Nevertheless, attempts have been made to estimate the quality of the roadmap and the number of samples that will be required. For example the results reported in [4] bound the number of samples generated by the algorithm in [27], under the assumption that the configuration space verifies some simple geometric property.

0.3.4 Motion Planning under Differential Constraints

Real robots are constrained by mechanics and the laws of physics. Moreover, it is commonly the case that a robot has more dofs than the number of actuators that can control those configuration space parameters (underactuation). This means that a planner needs to produce paths that respect those constraints and are thus implementable (feasible). Over the years many terms have been used in the literature to describe these classes of problems. Constraints in a robot's motion that cannot be converted into constraints that involve no derivatives, such as non-integrable velocity constraints in the configuration space, are usually referred to as **nonholonomic**. A more recent term is **kinodynamic** constraints. The latter describes second order constraints on both velocity and acceleration, and lately has been used to describe problems that involve dynamics in general. A generic term that captures all these constraints is **differential constraints** [61].

Below nonholonomic constraints are examined in some more depth to better understand the controllability issues introduced in "Robot Algorithms Control". This section closes with a description of the work on sampling-based planners that can be applied to problems with differential constraints.

Planning for Robots with Nonholonomic Constraints

The trajectories of a nonholonomic robot are constrained by $p \geq 1$ nonintegrable scalar equality constraints:

$$G(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) = \big(G^1(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)), \ldots, G^p(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))\big) = (0, \ldots, 0) \ ,$$

where $\dot{\boldsymbol{q}}(t) \in T_{\boldsymbol{q}(t)}(\mathcal{C})$ designates the velocity vector along the **trajectory** $\boldsymbol{q}(t)$. At every $\boldsymbol{q}$, the function $G_{\boldsymbol{q}} = G(\boldsymbol{q}, .)$ maps the tangent space[2] $T_{\boldsymbol{q}}(\mathcal{C})$ into $\mathbb{R}^p$. If $G_{\boldsymbol{q}}$ is smooth and its Jacobian has full rank (two conditions that are often satisfied), the constraint $G_{\boldsymbol{q}}(\dot{\boldsymbol{q}}) = (0, \ldots, 0)$ constrains $\dot{\boldsymbol{q}}$ to be in a linear subspace of $T_{\boldsymbol{q}}(\mathcal{C})$ of dimension $m - p$. The nonholonomic robot may also be subject to scalar inequality constraints of the form $H^j(\boldsymbol{q}, \dot{\boldsymbol{q}}) > 0$. The subset of $T_{\boldsymbol{q}}(\mathcal{C})$ that satisfies all the constraints on $\dot{\boldsymbol{q}}$ is called the set $\Omega(\boldsymbol{q})$ of controls at $\boldsymbol{q}$. A feasible path is a piecewise differentiable path whose tangent lies everywhere in the control set.

A car-like robot is a classical example of a nonholonomic robot. It is constrained by one equality constraint (the linear velocity must point along the car's axis so that the car is not allowed to move sideways). Limits on the steering angle impose two inequality constraints. Other nonholonomic robots include tractor-trailers, airplanes, and satellites.

A key question when dealing with a nonholonomic robot is: Despite the relatively small number of controls, can the robot span its configuration space? The study of this question requires introducing some controllability notions. Given an arbitrary subset $U \subset \mathcal{C}$, the configuration $\boldsymbol{q}_1 \in U$ is said to be U-accessible from $\boldsymbol{q}_0 \in U$ if there exists a piecewise constant control $\dot{\boldsymbol{q}}(t)$ in the control set whose integral is a trajectory joining $\boldsymbol{q}_0$ to $\boldsymbol{q}_1$ that fully lies in U. Let $A_U(\boldsymbol{q}_0)$ be the set of configurations U-accessible from $\boldsymbol{q}_0$. The robot is said to be locally controllable at $\boldsymbol{q}_0$ iff for every neighborhood U of $\boldsymbol{q}_0$, $A_U(\boldsymbol{q}_0)$ is also a neighborhood of $\boldsymbol{q}_0$. It is locally controllable iff this is true for all $\boldsymbol{q}_0 \in \mathcal{C}$. Car-like robots and tractor-trailers that can go forward and backward are locally controllable [5].

Let X and Y be two smooth vector fields on $\mathcal{C}$. The Lie bracket of X and Y, denoted by $[X, Y]$, is the smooth vector field on $\mathcal{C}$ defined by $[X, Y] = dY \cdot X - dX \cdot Y$, where dX and dY, respectively, denote the $m \times m$ matrices of the partial derivatives of the components of X and Y w.r.t. the configuration coordinates in a chart placed on $\mathcal{C}$. The Control Lie Algebra associated with the control set Ω, denoted by $L(\Omega)$, is the space of all linear combinations of vector fields in Ω closed by the Lie bracket operation. The following result derives from the Controllability Rank Condition Theorem [5]: *A robot is locally controllable if, for every $\boldsymbol{q} \in \mathcal{C}$, $\Omega(\boldsymbol{q})$ is symmetric with respect to the origin of $T_{\boldsymbol{q}}(\mathcal{C})$ and the set $\{X(\boldsymbol{q}) \mid X(\boldsymbol{q}) \in L(\Omega(\boldsymbol{q}))\}$ has dimension m.*

The minimal number of Lie brackets sufficient to express any vector in $L(\Omega)$ using vectors in Ω is called the degree of nonholonomy of the robot. The degree of nonholonomy of a car-like robot is 2. Except at some singular configurations, the degree of nonholonomy of a tractor towing a chain of s trailers is

[2]The tangent space $T_p(M)$ at a point p of a smooth manifold M is the vector space of all tangent vectors to curves contained in M and passing through p. It has the same dimension as M.

$2 + s$. Intuitively, the higher the degree of nonholonomy the more complex (and the slower) the robot's maneuvers to perform some motions.

Let $\mathcal{A}$ be a locally controllable nonholonomic robot. A necessary and sufficient condition for the existence of a feasible free path of $\mathcal{A}$ between two given configurations is that they lie in the same connected component of the open free space. Indeed, local controllability guarantees that a possibly nonfeasible path can be decomposed into a finite number of subpaths, each short enough to be replaced by a feasible free subpath [31]. Hence, deciding if there exists a free path for a locally controllable nonholonomic robot has the same complexity as deciding if there exists a free path for the holonomic robot having the same geometry. Transforming a nonfeasible free path τ into a feasible one can be done by recursively decomposing τ into subpaths. The recursion halts at every subpath that can be replaced by a feasible free subpath. Specific substitution rules (e.g., Reeds and Shepp curves) have been defined for car-like robots [31]. The complexity of transforming a nonfeasible free path τ into a feasible one is of the form $O(\epsilon^d)$, where ϵ is the smallest clearance between the robot and the obstacles along τ and d is the degree of nonholonomy of the robot. The algorithm in [5] directly constructs a nonholonomic path for a car-like or a tractor-trailer robot by searching a tree obtained by concatenating short feasible paths, starting at the robot's initial configuration. The planner is guaranteed to find a path if one exists, provided that the length of the short feasible paths is small enough. It can also find paths that minimize the number of cusps (changes of sign of the linear velocity).

Path planning for nonholonomic robots that are not locally controllable is much less understood. Research has almost exclusively focused on car-like robots that can only move forward. The algorithm in [17] decides whether there exists such a path between two configurations, but it runs in time exponential in obstacle complexity. The algorithm in [1] computes a path in polynomial time under the assumptions that all obstacles are convex and their boundaries have a curvature radius greater than the minimum turning radius of the point (so called "moderate obstacles"). Other polynomial algorithms [5] require some sort of discretization.

Planning for Robots with Differential Constraints

A robot's motion can generally be describe by a set of non-linear equations of motion $\dot{x} = f(x, u)$, together with some constraints $g(x, \dot{x}) \leq 0$. x is the robot's state vector and u is a vector of control inputs. The robot can be abstracted into a point that moves in a **state space**. The state space is a superset of the aforementioned configuration space. For example a car moving in a plane can be modelled using a 5-dimensional state space where $x = (x_r, y_r, \theta, v, s)$. x_r and y_r provide the position of a reference point on

the car and together with orientation θ, they constitute the car's configuration. For the state description, we need the components for the linear velocity v and angle of the steering s. In this model the car is controlled by two inputs: $u = (a, b)$. a is the car's linear acceleration and b is the car's steering velocity. Notice that this system can have bounds in its acceleration which is a second order constraint.

Sampling-based techniques are becoming increasingly popular for tackling problems with kinodynamic and in general differential constraints [61, 62, 72]. For this reason, this section will not extend beyond sampling-based planners. For planning under differential constraints, the planner is now called to search the state space described above. Since motion is constrained by non-linear equations, collision checking is generalized to also check for invalid states (e.g., due to constraint violations). Moreover, the sampling process is modified. The planner typically produces a random set of controls that are applied at a given state for some time in attempt to move the systems towards a newly sampled state. For this process, the planner requires a function that can integrate the equations of motion forward in time. Notice that all the produced trajectories are feasible by construction. This is an advantage, since the planner avoids dealing with controllability issues. The disadvantage is that it becomes hard to drive the system to an exact goal state. To overcome this difficulty sometimes it is easier to define a whole region of states that are acceptably close to the goal.

Adaptations of tree-based planners such as RRT and EST proved very successful in kinodynamic problems (for example see [73, 74]). In addition, many newer planners are specifically designed for problems with differential constraints [70, 75] although they can apply to simpler problems as well. An interesting observation is that the planner only needs the integrator function as a black box. For this reason it is possible to use a physical simulator to do the integration of motion [92]. In this way, it is possible to model more realistic effects and constraints such as inertia, gravity and friction. Moreover, it becomes possible to do planning for systems for which the exact equations are not explicitly written.

0.3.5 Extentions to Motion Planning

Most of the topic discussed so far, focused on difficulties presented by the nature of a robot itself. In practice there are many interesting problems that extend beyond planning for a robot in a static known environment. In the following paragraphs an attempt is made to cover the most important ones. For more detailed descriptions the reader is referred to [30, 60, 61, 62].

Dynamic Workspace

In the presence of moving obstacles, one can no longer plan a robot's motion as a mere geometric path. The path must be indexed by time and is then called a trajectory. The simplest scenario is when the trajectories of the moving obstacles are known in advance or can be accurately predicted. In that case, planning can be done in the configuration×time space ($\mathcal{C} \times [0, +\infty)$) of the robot. All workspace obstacles map to static forbidden regions in that space. A free trajectory is a free path in that space whose tangent at every point points positively along the time axis (or within a more restricted cone, if the robot's velocity modulus is bounded).

Computing a free trajectory for a rigid object in 3-space among arbitrarily moving obstacles (with known trajectories) is PSPACE-hard if the robot's velocity is bounded, and NP-hard otherwise [48]. The problem remains NP-hard for a point robot moving with bounded velocity in the plane among convex polygonal obstacles translating at constant linear velocities [10]. A complete planning algorithm is given in [48] for a polygonal robot that translates in the plane among polygonal obstacles. The obstacles translate at fixed velocities. This algorithm takes time exponential in the number of moving obstacles and polynomial in the total number of edges of the robot and the obstacles.

In realistic scenarios, exact information about moving obstacles trajectories is not available. In those, cases, a robot has to make conservative assumptions about where the obstacles will be in the future. E.g., for problems on a plane, moving obstacles with bounded velocity can be anywhere within a disk whose radius grows in time. As long as the robot's path does not enter any of those disk at any moment it time, the robot's path will be collision free. This idea is described in [65] which also shows how to find time-optimal paths. Another approach that tries to address these issues is by employing an adaptive online replanning strategy [67, 70]. In such approaches, the planner operates in a closed loop and is interleaved with sensing operations that try to keep track of the motions of the obstacles. A partial plan for some small time horizon is computed and implemented. Then using the latest available information the planner is called again to compute a new plan. Such replanning techniques are also known as online planning and are discussed in a bit more detail in the relevant section below. For further reference, [94] is closely related to the topics discussed in this subsection

Planning for Multiple Robots

The case of multiple robots can be trivially addressed by considering them as the components of a single robot, that is, by planning a path in the cross product of their configuration spaces. This product is called the composite configuration space of the robots and the approach is referred to as centralized planning.

This approach can be used to take advantage of the completeness properties that single robot algorithms have. Due to the increased complexity, powerful planners need to be used (e.g., [78]).

One may try to reduce complexity by separately computing a path for each robot, before tuning the robots' velocities along their respective paths to avoid inter-robot collision (decoupled planning) [64]. This approach can be helped by assigning priorities to robots according to some measure of importace. Then, each robot has to respect the higher priorities robots when planning. Although inherently incomplete, decoupled planning may work well in some practical applications.

Planning for multiple robots can become really challenging if it has to be done distributedly, with each robot restricted to its own configuration space. This a reasonable situation in practice where each robot may have its own limitations and goals and can interact only with the robots that are close to it. In [64] the robots move towards their independent goals. During the process, when robots come close, they form dynamic networks. The robots within a network solve a centralized motion planning problem to avoid collisions between each other using all available information within the network. Whenever robots get out of some communication range, networks are dissolved. For this kind of problems, planning can greatly benefit from utilizing the communication capabilities that robots typically have. In [76, 77] a coordination framework for mulitrobot planning is described. In this work each robot plans in its own state space, and uses only local information obtained through communication with neighboring robots. The approach can deal with robots that have non-trivial kinodynamic constraints. Moreover, it is possible to guarantee collision avoidance between robots and static obstacles while the robots can move as a connected network. Coordination is achieved with a distributed message passing scheme.

Planning with Movable Objects

Many robot tasks require from the robot to interact with its environment by moving an object. Such objects, are called *movable objects*, and cannot move by themselves; they must be moved by a robot. These problems fall into the category of *manipulation planning*.

In [53] the robot $\mathcal{A}$ and the movable object M are both convex polygons in a polygonal workspace. The goal is to bring $\mathcal{A}$ and M to specified positions. $\mathcal{A}$ can only translate. To grasp M, $\mathcal{A}$ must have one of its edges that exactly coincides with an edge of M. While $\mathcal{A}$ grasps M, they move together as one rigid object. An exact cell decomposition algorithm is given that runs in $O(n^2)$ time after $O(n^3 \log^2 n)$ preprocessing, where n is the total number of edges in the workspace, the robot, and the movable object. An extension of this problem allowing an infinite set of grasps is solved by an exact cell decomposition algorithm in [3].

Heuristic algorithms have also been proposed. The planner in [29] first plans the path of the movable object M. During that phase, it only verifies that for every configuration taken by M there exists at least one collision-free configuration of the robot where it can hold M. In the second phase, the planner determines the points along the path of M where the robot must change grasps. It then computes the paths where the robot moves alone to (re)grasp M. The paths of the robot when it carries M are obtained through inverse kinematics. This planner is not complete, but it has solved complex tasks in practice.

Another scenario is that of a robot that needs to move inside a building where the doors are blocked by obstacles that have to be moved out of the way. Questions of interest are, how many obstacles must be moved, in which order must these obstacles be moved and where should the robot put them so as not to block future motions. [66, 91] presents a planner that can solve problems in class LP. An $LP_k \subseteq LP$ problem is one where a series of $k-1$ obstacles must be moved before an obstacle can be reached and moved, so that a room becomes accessible through a blocked door.

On-Line Planning and Exploration

On-line planning addresses the case where the workspace is initially unknown or partially unknown. As the robot moves, it acquires new partial information about the workspace through sensing. A motion plan is generated using the partial information that is available and updated as new information is acquired. With this planning strategy, a robot exhibits a "reactive or adaptive behavior" and can also move in the the presence of moving obstacles or other robots.

In [67, 70] the robot initially preprocesses the workspace and creates a roadmap as described in "Sampling-Based Algorithms". Then the robot finds a path to its goal and starts following it, until an unexpected change, such as a moving obstacle, is sensed. Then, the robot quickly revises its path online, and chooses a new path on the roadmap that avoids regions that are invalidated by the obstacle. Another useful application where online planning is needed is that of exploration of unknown environments [89]. In that case it in necessary that a robot plans online and revises its plan periodically after receiving new sensor information. Exploration can be done more efficiently when multiple robots are used. In [76] a team of robots explores an unknown environment. Each robot plans online and communicates with neighboring robots to coordinate their actions before computing a new plan. In this way, robots that have kinodynamic constraints, manage to complete their task while avoiding all collisions between robots or workspace obstacles.

The main difficulty in online planning, is that the planner has to run under a time budget, and must generally be very fast in producing a new plan. Moreover, since the overall motion is a concatenation of

small motions that use only partial information without a global view of the workspace, the quality of the paths can be bad, and it is very hard to to establish path optimality.

A way of evaluating an on-line planner is competitive analysis. The competitive ratio of an on-line planner is the maximal ratio (over all possible workspaces) between the length of the path generated by the on-line algorithm and the length of the shortest path [44]. Competitive analysis is not restricted to path length and can be applied to other measures of performance as well.

Optimal Planning

There has been considerable research in computational geometry on finding shortest Euclidean paths [63] (chapter 27), but minimal Euclidean length is usually not the most suitable criterion in robotics. Rather, one wishes to minimize execution time, which requires taking the robot's dynamics into account.

Optimal-Time Control Planning, the input is a geometric free path τ parameterized by $s \in [0, L]$, the distance travelled from the starting configuration. The problem is to find the time parameterization $s(t)$ that minimizes travel time along τ, while satisfying actuator limits.

The dynamic equation of motion of a robot arm with m dofs can be written as $M(\boldsymbol{q})\ddot{\boldsymbol{q}} + V(\dot{\boldsymbol{q}}, \boldsymbol{q}) + G(\boldsymbol{q}) = \Gamma$, where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$, respectively, denote the robot's configuration, velocity, and acceleration [12]. M is the $m \times m$ inertia matrix of the robot, V the m-vector (quadratic in $\dot{\boldsymbol{q}}$) of centrifugal and Coriolis forces, and G the m-vector of gravity forces. Γ is the m-vector of the torques applied by the joint actuators.

Using the fact that the robot follows τ, this equation can be rewritten in the form: $\boldsymbol{m}\ddot{s} + \boldsymbol{v}\dot{s}^2 + \boldsymbol{g} = \Gamma$, where $\boldsymbol{m}$, $\boldsymbol{v}$, and $\boldsymbol{g}$ are derived from M, V, and G, respectively. Minimum-time control planning becomes a two-point boundary value problem: find $s(t)$ that minimizes $t_f = \int_0^L ds/\dot{s}$, subject to $\Gamma = \boldsymbol{m}\ddot{s} + \boldsymbol{v}\dot{s}^2 + \boldsymbol{g}$, $\Gamma_{min} \leq \Gamma \leq \Gamma_{max}$, $s(0) = 0$, $s(t_f) = L$, and $\dot{s}(0) = \dot{s}(L) = 0$. Numerical techniques solve this problem by finely discretizing the path τ.

To find a minimal-time trajectory, a common approach is to first plan a geometric free path and then iteratively deform this path to reduce travel time. Each iteration requires checking the new path for collision and recomputing the optimal-time control. No bound has been established on the running time of this approach or the goodness of its outcome. The problem is NP-hard for a point robot under Newtonian mechanics in 3-space. The approximation algorithm in [13] computes a trajectory ϵ-close to optimal in time polynomial in both $1/\epsilon$ and the workspace complexity.

Dealing with Uncertainties in Motion and Sensing

In all of the topics discussed so far it has been implicitly assumed that the robot has accurate knowledge abouts its own state at the present and possibly future times after executing some action. Unfortunately this is not at all the case in practice. Real robots have a number of inherent mechanichal imperfections that affect the result of an action and introduce motion errors. For example, the actuators introduce errors when executing a motion command and there is usually discrepancy between how much the robot thinks it has moved as opposed to how much it has actually moved. The problem is more pronounced in mobile robots that have *odometry errors* produced by unaccounted for wheel slipage. The effect of such errors, is that the robot no longer has exact knowledge of its own state. Instead, it has to somehow infer its state from the available sensing information. To add to the overall problem, processing sensor information is computationally intensive and erroneous due to imperfect sensors.

All these issues amount to the very fundamental problem of dealing with uncertainty in motion and sensing. A very simple example was already shown in "Robot Algorithms Control", Fig. 2. A broad area of robotics algorithms focus on processing all the available information to reduce this uncertainty. Maybe the most common framework for handling uncertainty is based on the theory of Bayesian estimation [60]. The robot is assumed to be at some state x_t which is unknown. The robot maintains a probability distribution $bel(x)$ called belief, that represents how likely it is that the true state is x. For Bayesian estimation, some system specific characteristics must be modelled. In particular, a state transition function must be given, to describe how likely it is to move to some new state x_{t+1} when starting at some state x_t and performing an action/motion. Moreover, a measurement model must be available to describe how likely it is to take some specific measurement when being at some state. Given the belief at time t together with the latest action and sensor measurement, Bayesian estimation performs two steps in an attempt to compute the next belief $bel(x_{t+1})$. First, there is a prediction step where using $bel(x_t)$ and the state transition model, an estimate of $bel(x_{t+1})$ is computed. Then, this estimate is refined using the measurement model to validate the predictions against the latest sensor measurement. This technique is very general and powerful and has been successfully applied in many robotics estimation problems. Below, concrete examples of the most important estimation problems are briefly described. An extensive reference for such problems is [60].

Localization One of the most fundamental problem in robotics is that of robot localization. The goal is to maintain an accurate estimation of the robot's state at all times as the robot moves in a known environment for which a map is available. The easiest version is when the initial state is known with

certainty. Then the problem is called *position tracking* and the robot just needs to compensate for local uncertainty introduced everytime it attempts to move. A very successful technique for such problem is the Extended Kalman Filter (EKF) [60, 79]. A more challenging variant is called *global localization* [57, 80]. This is the problem of estimating the robot's state when the initial location is also unknown. This kind of problems are usually solved using Particle Filters [60].

Mapping The assumption that the robot's map is known does not always hold. In fact the mapping problem tries to address exactly this issue. Here, the robot is assumed to have an accurate mechanism for tracking is position (e.g., using a GPS). Its target is to use sensor information to build an accurate map of the environment. The map is usually described by the locations of all the features in the environment, called landmarks. A feature can be

In a sense this is the complementary problem to localization. In a mapping problem the state vector describes the map and $bel(x)$ is the distribution that describes how likely each version of a map is [82].

SLAM One of the most exciting and challenging class of estimation problems is that of Simultaneous Localization and Mapping or SLAM, where localization and mapping have to be tackled simultaneously. This is a hard problem since in most cases there is a demand to perform all the computations online as the robot moves. Moreover, this ends up being a "chicken and egg" problem where trusting a map would increase localization accuracy, while trusting localization can give better estimates about the position of the obstacles.

The typical difficulty for solving any SLAM problem, is in *data association*. The current estimated map is represented by the locations of certain landmarks. As the robot moves, the sensors extract new sensed features and the algorithm needs to decide whether a sensed feature represent a new landmark that has never been observed before or not. To solve this correspondence problem between the sensed features and the current list of observed landmarks, a standard way is to use maximum likelihood estimation techniques. Another problem that is also a benchmark for the effectiveness of SLAM algorithms, is that of *loop closure*. As the robot is moving it often happens that after spending a lot of time in some newly discovered part of the environment, the robot returns to an area it has visited before, and closes a loop. The ability to detect that indeed a loop was closed can be a challenging problem especially for online SLAM algorithms. Finally, for SLAM problems, the state that needs to be estimated tends to be very high dimensional and this leads to expensive computations for updating the state. There are a number of state of the art algorithms for SLAM [58, 81, 83, 84, 85] and the area is an active research field.

0.4 Distance Computation

The efficient computation of (minimum) distances between bodies in 2- and 3-space is a crucial element of many algorithms in robotics [63].

Algorithms have been proposed to efficiently compute distances between two convex bodies. In [14], an algorithm is given which computes the distance between two convex polygons P and Q (together with the points that realize it) in $O(\log p + \log q)$ time, where p and q denote the number of vertices of P and Q, respectively. This time is optimal in the worst case. The algorithm is based on the observation that the minimal distance is realized between two vertices or between a vertex and an edge. It represents P and Q as sequences of vertices and edges and performs a binary search that eliminates half of the edges in at least one sequence at each step. A widely tested numerical descent technique is described in [18] to compute the distance between two convex polyhedra; extensive experience indicates that it runs in approximately linear time in the total complexity of the polyhedra.

Most robotics applications, however, involve many bodies. Typically, one must compute the minimum distance between two sets of bodies, one representing the robot, the other the obstacles. Each body can be quite complex and the number of bodies forming the obstacles can be large. The cost of accurately computing the distance between every pair of bodies is often prohibitive. In that context, simple bounding volumes, such as parallelepipeds and spheres, have been extensively used to reduce computation time. They are often coupled with hierarchical decomposition techniques, such as octrees, boxtrees, or sphere trees (for an example, see [46]). These techniques make it possible to rapidly eliminate pairs of bodies that are too far apart to contribute the minimum distance.

When motion is involved, incremental distance computation has been suggested for tracking the closest points on a pair of convex polyhedra [34, 86, 87]. It takes advantage of the fact that the closest features (faces, edges, vertices) change infrequently as the polyhedra move along finely discretized paths.

0.5 Research Issues and Summary

In this chapter we have introduced robot algorithms as abstract descriptions of processes consisting of motions and sensing operations in the physical space. Robot algorithms send commands to actuators and sensors in order to control a subset of the real world, the workspace, despite the fact that the workspace is subject to the imperfectly modeled laws of nature. Robot algorithms uniquely blend controllability, observability, computational complexity, and physical complexity issues, as described in Section 0.2. Research on robot algorithms is broad and touches many different areas. In Section 0.3 we have surveyed

a number of selected areas in which research has been particularly active: part manipulation (grasping, fixturing, feeding), assembly sequencing, motion planning (including basic path planning, nonholonomic planning, dynamic workspaces, multiple robots, and optimal-time planning), and sensing.

Many of the core issues reviewed in Section 0.2 have been barely addressed in currently existing algorithms. There is much more to understand in how controllability, observability, and complexity interact in robot tasks. The interaction between controllability and complexity has been studied to some extent for nonholonomic robots. The interaction between observability (or recognizability) and complexity has been considered in motion planning with uncertainty. But, in both cases, much more remains to be done.

Concerning the areas studied in Section 0.3, several specific problems remain open. We list a few below (by no means is this list exhaustive):

- Given a workspace W, find the optimal design of a robot arm that can reach everywhere in W without collision. The three-dimensional case is largely open. An extension of this problem is to design the layout of the workspace so that a certain task can be completed efficiently.
- Given the geometry of the parts to be manipulated, predict feeders' throughputs to evaluate alternative feeder designs. In relation to this problem, simulation algorithms have been used to predict the pose of a part dropped on a flat surface [39].
- In assembly planning, the complexity of an NDBG grows exponentially with the number of parameters that control the allowable motions. Are there situations where only a small portion of the full NDBG need be constructed?
- Develop efficient sampling techniques for searching the configuration space of robots with many degrees of freedom in the context of the scheme given in [4].
- Establish a nontrivial lower bound on the complexity of planning for a nonholonomic robot that is not locally controllable.

0.6 Defining Terms

Basic path planning problem: Compute a free or semifree path between two input configurations for a robot moving in a known and static workspace.

C-Obstacle: Given an obstacle B_i, the subset CB_i of the configuration space $\mathcal{C}$ such that, for any

$\boldsymbol{q} \in CB_i$, $\mathcal{A}(\boldsymbol{q})$ intersects B_i. The union $CB = \cup_i CB_i$ plus the configurations that violate the mechanical limits of the robot's joints is called the **C-obstacle region**.

Complete motion planner: A planner guaranteed to find a (semi)free path between two given configurations whenever such a path exists, and to notify that no such path exists otherwise.

Configuration: Any mathematical specification of the position and orientation of every body composing a robot $\mathcal{A}$, relative to a fixed coordinate system. The configuration of a single body is also called a **placement** or a **pose**.

Configuration space: Set $\mathcal{C}$ of all configurations of a robot. For almost any robot, this set is a smooth manifold.

Differential constraint: A motion constraint that is both nonholonomic (non-integrable) and kinodynamic (at least second order).

Free path: A path in free space.

Free space: The complement of the C-obstacle region in $\mathcal{C}$, that is, $\mathcal{C} \backslash CB$.

Kinodynamic constraint: A second order constraint in the configuration space, i.e., for problems where both velocity and acceleration bounds are specified for the robot's motion.

Linkage: A collection of rigid objects, called links, in which some pairs of links are connected by joints (e.g., revolute and/or prismatic joints). Most industrial robot arms are serial linkages with actuated joints.

Nonholonomic constraint: Constraints in a robot's motion that cannot be converted into constraints that involve no derivatives. E.g. non-integrable velocity constraints in the Configuration space.

Number of degrees of freedom: The dimension m of $\mathcal{C}$.

Obstacle: The workspace W is often defined by a set of obstacles (bodies) B_i $(i = 1, \ldots, q)$ such that $W = \mathbb{R}^k \backslash \bigcup_1^q B_i$.

Path: A continuous map $\tau : [0, 1] \to \mathcal{C}$.

Probabilistically complete: An algorithm is probabilistically complete if given enough time, the probability of finding a solution goes to 1 when a solution exists

Semifree path: A path in the closure of free space.

State space: The set of all robot's states. A superset of the configuration space that captures the robot's dynamics as well.

Trajectory: Path indexed by time.

Workspace: A subset of the two- or three-dimensional physical space modeled by $W \subset \mathbb{R}^k$, $k = 2, 3$.

Workspace complexity: The total number of features (vertices, edges, faces, etc.) on the boundary of the obstacles.

References

Bibliography

[1] Agarwal, P.K., Raghavan, P., and Tamaki, H., Motion Planning for a Steering-Constrained Robot Through Moderate Obstacles. *Proc. 28th ACM STOC,* 343–352, 1995.

[2] Akella, S., Huang, W., Lynch, K., and Mason, M.T., Planar Manipulation on a Conveyor with a One Joint Robot. In *Robotics Research,* Giralt, G. and Hirzinger, G., Eds., Springer, 265–276, 1996.

[3] Alami, R., Laumond, J.P., and Siméon, T., Two Manipulation Algorithms. In *Algorithmic Foundations of Robotics,* Goldberg, K.Y., Wellesley, M.A., Eds., AK Peters, 109–125, 1995.

[4] Barraquand, J., Kavraki, L.E., Latombe, J.C., Li, T.Y., Motwani, R., and Raghavan, P., A Random Sampling Framework for Path Planning in Large-Dimensional Configuration Spaces. *Int. J. of Robotics Research,* 16(6), 759–774, 1997.

[5] Barraquand, J. and Latombe, J.C., Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles, *Algorithmica,* 10(2-3-4), 121–155, 1993.

[6] de Berg, M., van Kreveld, M., Overmars, M. and Schwarzkopf, O., *Computational Geometry: Algorithms and Applications.* Springer, New York, 2000.

[7] Boddy M. and Dean T.L., Solving Time-Dependent Planning Problems, *Proc. 11th Int. Joint Conf. on Artificial Intelligence,* 979–984, 1989.

[8] Briggs, A.J., *Efficient Geometric Algorithms for Robot Sensing and Control.* Report No. 95-1480, Dept. of Computer Science, Cornell University, Ithaca, NY, 1995.

[9] Brost, R.C. and Goldberg, K.Y., Complete Algorithm for Designing Planar Fixtures Using Modular Components. *IEEE Tr. on Systems, Man and Cybernetics,* 12, 31–46, 1996.

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

[11] Canny, J.F., On Computability of Fine Motion Plans, *Proc. IEEE Int. Conf. on Robotics and Automation,* Scottsdale, AZ, 177–182, 1989.

[12] Craig, J.J., *Introduction to Robotics. Mechanics and Control.* Addison-Wesley, Reading, MA, 2004.

[13] Donald, B.R., Xavier, P., Canny, J.F., and Reif, J.H., Kinodynamic Motion Planning. *J. of the ACM,* 40, 1048–1066, 1993.

[14] Edelsbrunner, H., Computing the Extreme Distances between Two Convex Polygons. *J. of Algorithms,* 6, 213–224, 1985.

[15] Erdmann, M., Using Backprojections for Fine Motion Planning with Uncertainty. *Int. J. of Robotics Research*, 5, 19–45, 1986.

[16] Erdmann, M. and Mason, M.T., An Exploration of Sensorless Manipulation. *IEEE Tr. on Robotics and Automation,* 4(4), 369–379, 1988.

[17] Fortune, S. and Wilfong, G.T., Planning Constrained Motions. In *Proc. ACM Symp. on Theory of Computing,* 445–459, 1988.

[18] Gilbert, E.G., Johnson, D.W., and Keerthi, S.S., A Fast Procedure for Computing Distance Between Complex Objects in Three-Dimensional Space. *IEEE Tr. on Robotics and Automation,* 4, 193–203, 1988.

[19] Giralt, G. and Hirzinger, G., Eds., *Robotics Research,* Springer, 1996.

[20] Goldberg, K.Y., Orienting Polygonal Parts without Sensors. *Algorithmica,* 10(2-3-4), 201–225, 1993.

[21] Goldberg, K.Y., Halperin, D., Latombe, J.C., and Wilson, R.H., Eds., *Algorithmic Foundations of Robotics,* AK Peters, Ltd., Wellesley, MA, 1995.

[22] Guibas, L., Motwani, R., and Raghavan, P., The Robot Localization Problem in Two Dimensions. *SIAM J. on Computing,* 26(4), 1121–1138, 1996.

[23] Halperin, D. Arrangements. In Goodman, J.E. and O'Rourke, J., Eds., *Handbook of Discrete and Computational Geometry,* CRC Press, Boca Raton, FL, 389–412, 1997.

[24] Halperin, D. and Sharir, M., Near-Quadratic Algorithm for Planning the Motion of a Polygon in a Polygonal Environment. *Discrete Computational Geometry,* 16, 121–134, 2004.

[25] Kant, K.G. and Zucker, S.W., Toward Efficient Trajectory Planning: Path Velocity Decomposition. *Int. J. of Robotics Research,* 5, 72–89, 1986.

[26] Kavraki, L.E. and Kolountzakis, M.N., Partitioning a Planar Assembly Into Two Connected Parts is NP-complete. *Information Processing Letters,* 55, 159–165, 1995.

[27] Kavraki, L.E., Švestka, P., Latombe, J.C., and Overmars, M., Probabilistic Roadmaps for Fast Path Planning in High Dimensional Configuration Spaces. *IEEE Tr. on Robotics and Automation,* 12, 566–580, 1996.

[28] Khatib, O. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *Int. J. of Robotics Research,* 5, 90–98, 1986.

[29] Koga, Y., Kondo, K., Kuffner, J., and Latombe, J.C., Planning Motions with Intentions. *Proc. ACM SIGGRAPH'94,* 395–408, 1994.

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

[31] Laumond, J.P., Jacobs, P., Taix, M., and Murray, R., A Motion Planner for Nonholonomic Mobile Robots. *IEEE Tr. on Robotics and Automation,* 10, 577–593, 1994.

[32] Laumond, J.P. and Overmars, M., Eds., *Algorithms for Robot Motion and Manipulation,* AK Peters, Wellesley, MA, 1997.

[33] Lazanas, A. and Latombe, J.C., Landmark-Based Robot Navigation. *Algorithmica,* 13, 472–501, 1995.

[34] Lin, M.C. and Canny, J.F., A Fast Algorithm for Incremental Distance Computation. *Proc. IEEE Int. Conf. on Robotics and Automation,* 1008–1014, 1991.

[35] Lozano-Pérez T., Spatial Planning: A Configuration Space Approach, *IEEE Tr. on Computers,* 32(2), 108–120, 1983.

[36] Lozano-Pérez T., Mason, M.T., and Taylor, R.H., Automatic Synthesis of Fine-Motion Strategies for Robots, *Int. J. of Robotics Research,* 3(1), 3–24, 1984.

[37] Lumelsky, V., A Comparative Study on the Path Length Performance of Maze-Searching and Robot Motion Planning Algorithms. *IEEE Tr. on Robotics and Automation,* 7, 57–66, 1991.

[38] Mason, M.T., Mechanics and Planning of Manipulator Pushing Operations, *Int. J. of Robotics Research,* 5(3), 53–71, 1986.

[39] Mirtich, B., Zhuang, Y., Goldberg, K., Craig, J.J., Zanutta, R., Carlisle, B., and Canny, J.F., Estimating Pose Statistics for Robotic Part Feeders. *Proc. IEEE Int. Conf. on Robotics and Automation,* 1140–1146, 1996.

[40] Mishra B., Schwartz, J.T., and Sharir, M., On the Existence and Synthesis of Multifinger Positive Grips, *Algorithmica,* 2, 541–558, 1987.

[41] Natarajan, B.K., On Planning Assemblies. *Proc. 4th ACM Symp. on Computational Geometry,* 299–308, 1988.

[42] Nguyen, V.D., Constructing Force-Closure Grasps. *Int. J. of Robotics Research,* 7, 3–16, 1988.

[43] Nourbakhsh, I.R., *Interleaving Planning and Execution.* Ph.D. Thesis. Dept. of Computer Science, Stanford University, Stanford, CA, 1996.

[44] Papadimitriou, C.H. and Yannakakis, M., Shortest Paths Without a Map. *Theoretical Computer Science,* 84, 127–150, 1991.

[45] Ponce, J., Sudsang, A., Sullivan, S., Faverjon, B., Boissonnat, J.D., and Merlet, J.P., Algorithms for Computing Force-Closure Grasps of Polyhedral Objects. In *Algorithmic Foundations of Robotics,* Golberg, K.Y and Wellesley, M.A., Eds., AK Peters, 167–184, 1995.

[46] Quinlan, S., Efficient Distance Computation between Non-Convex Objects. *Proc. IEEE Int. Conf. on Robotics and Automation,* 3324–3329, 1994.

[47] Reif J.H., Complexity of the Mover's Problem and Generalizations. *Proc. FOCS,* 421–427, 1979.

[48] Reif, J.H. and Sharir, M., Motion Planning in the Presence of Moving Obstacles. *Journal of the ACM,* 41(4), 764–90, 1994.

[49] Schwartz, J.T. and Sharir, M., On the 'Piano Movers' Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds, *Advances in Applied Mathematics,* 4, 298–351, 1983.

[50] Skiena, S.S., Geometric reconstruction problems. In Goodman, J.E. and O'Rourke, J., Eds., *Handbook of Discrete and Computational Geometry*, CRC Press, Boca Raton, FL, 481–490, 2004.

[51] Snoeyink, J. and Stolfi, J., Objects That Cannot Be Taken Apart with Two Hands. *Discrete Computational Geometry,* 12, 367–384, 1994.

[52] Talluri, R. and Aggarwal, J.K., Mobile Robot Self-Location Using Model-Image Feature Correspondence. *IEEE Tr. on Robotics and Automation,* 12, 63–77, 1996.

[53] Wilfong, G.T., Motion Planning in the Presence of Movable Objects. *Annals of Mathematics and Artificial Intelligence,* 3, 131–150, 1991.

[54] Wilson, R.H. and Latombe, J.C., Reasoning About Mechanical Assembly. *Artificial Intelligence,* 71, 371–396, 1995.

[55] Zhang, Z. and Faugeras, O., A 3D World Model Builder with a Mobile Robot. *Int. J. of Robotics Research,* 11, 269–285, 1996.

[56] Zhuang, Y., Goldberg, K.Y., and Wong, Y., On the existence of modular fixtures. *Proc. IEEE Int. Conf. on Robotics and Automation,* 543–549.

[57] Thrun, S., Fox, D., Burgard, W. and Dellaert, F., Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence,* 128, 99–141, 2000.

[58] Montemerlo, M.,Thrun, S., Koller, D. and Wegbreit, B., FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem, *Proceedings of the AAAI National Conference on Artificial Intelligence,* 2002.

[59] Ferris, B., Hhnel, D. and Fox, D., Gaussian Processes for Signal Strength-Based Location Estimation. *Robotics: Science and Systems,* 2006.

[60] S. Thrun, W. Burgard, and D. Fox., Probabilistic Robotics, *MIT Press*, Cambridge, MA, 2005.

[61] LaValle, S. M. , Planning Algorithms, *Cambridge University Press,* 2006.

[62] Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, E.L. and Thrun, S. , Principles of Robot Motion, *MIT Press,* 2005.

[63] Lin, M. C.and Manocha, D., Collision and Proximity Queries, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, 787–807, 2004.

[64] Clark, C. M., Rock, S. M. and Latombe, J.-C., Motion Planning for Multiple Mobile Robots using Dynamic Networks, *Proceedings IEEE International Conference on Robotics & Automation*, vol 3, 4222–4227, 2003.

[65] Berg, J.P. van den and Overmars, M.H., Planning the shortest safe path amidst unpredictably moving obstacles, *Proc. Workshop on Algorithmic Foundations of Robotics,* 2006.

[66] Nieuwenhuisen, D., Stappen, A. F. van der and Overmars, M. H., An Effective Framework for Path Planning amidst Movable Obstacles *Proc. Workshop on the Algorithmic Foundations of Robotics,* 2006.

[67] Berg, J. van den and Ferguson, D. and Kuffner, J., Anytime Path Planning and Replanning in Dynamic Environments, *Proc. IEEE Int. Conf. on Robotics and Automation,* 2366–2371, 2006.

[68] Kuffner, J. and LaValle, S.M., RRT-connect: An efficient approach to single-query path planning, *Proc. IEEE Int. Conf. on Robotics and Automation,* 995-1001, 2000.

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

[70] Bekris, K.E. and Kavraki, L.K., Greedy but safe replanning under kinodynamic constraints, *Proc. IEEE Int. Conf. on Robotics and Automation,* 2007.

[71] Laumond, J.P., Robot Motion Planning and Control, *Springer-Verlag,* 1998.

[72] Tsianos, K.I., Sucan, I.A., Kavraki, L.E., Sampling-based robot motion planning: Towards realistic applications, *Computer Science Review,*, Vol 1, 2-11, August 2007.

[73] Kindel, R., Hsu, D., Latombe, J.C. and Rock, S., Kinodynamic motion planning amidst moving obstacles, *Proc. IEEE Int. Conf. on Robotics and Automation,* 537-543, 2000.

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

[75] Ladd, A. M. and Kavraki, L. E. . Motion Planning in the Presence of Drift, Underactuation and Discrete System Changes. *In Robotics: Science and Systems,* MIT, Boston, MA, June 2005.

[76] Bekris, K.E., Tsianos, K.I., and Kavraki, L.E., Distributed and Safe Real-Time Planning for Networks of Second-Order Vehicles, *Internationl Conference in Robot Communication and Coordination* , October 2007.

[77] Bekris, K.E., Tsianos, K.I., and Kavraki, L.E., A Decentralized Planner that Guarantees the Safety of Communicating Vehicles with Complex Dynamics that Replan Online, *International Conference on Intelligent Robots and Systems,* October 2007.

[78] Plaku, E., Bekris, K. E., Chen, B. Y., Ladd, A. M., and Kavraki, L. E.. Sampling-Based Roadmap of Trees for Parallel Motion Planning. IEEE Transactions on Robotics, vol. 21, n. 4, 597-608, August 2005.

[79] Smith, R. C., and Cheeseman, P.. On the Representation and Estimation of Spatial Uncertainty. IJRR, 5(4):5668, 1987.

[80] Kwok,C., Fox, D., and Meila, M., Real-time Particle Filters, *Proceedings of the IEEE,* Vol 92(2), 469- 484 2004.

[81] Davison, A. J. Real-time simultaneous localisation and mapping with a single camera. In 9th ICCV, volume 2, pages 14031410, October 13-16 2003.

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

[83] Dellaert, F. and Kaess, M. Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing, *The International Journal of Robotics Research,* Vol. 25, No. 12, 1181-1203, 2006.

[84] Durrant-Whyte, H. and Bailey, T. Simultaneous localization and mapping: part I *IEEE Robotics and Automation Magazine,* Vol 13(2),99- 110 June 2006.

[85] Bailey, T. and Durrant-Whyte, H. Simultaneous localization and mapping: part II *IEEE Robotics and Automation Magazine,* Vol 13(2),110- 117 June 2006.

[86] Larsen, E., Gottschalk, S., Lin, M.C and Manocha, D., Fast Distance Queries using Rectangular Swept Sphere Volumes *IEEE Int. Conference on Robotics and Automation,* 2000.

[87] Zhang,L., Kim, Y.J. and Manocha, D., C-DIST: Efficient Distance Computation for Rigid and Articulated Models in Configuration Space, *ACM Solid and Physical Modeling Symposium ,* 2007.

[88] Schwarzer, F., Saha, M. and Latombe, J.C., Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments, *IEEE Transactions on Robotics*, 21(3):338-353, June 2005.

[89] Thrun, S., Thayer, S., Whittaker, W., Baker, C., Burgard, W., Ferguson, D., Hhnel, D., Montemerlo, M., Morris, A., Omohundro, Z. and Reverte, C., Autonomous exploration and mapping of abandoned mines. *IEEE Robotics and Automation,* 11(4), 2005.

[90] Mason, M.T., Mechanics of Robotic Manipulation, *The MIT press*, 2001.

[91] Stilman, M. and Kuffner, J., Planning Among Movable Obstacles with Artificial Constraints, *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2006.

[92] Ladd, A., Motion planning for physical simulation, *PhD thesis,* Rice University, USA, 2007.

[93] Guibas, L. J., Halperin, D., Hirukawa, H., Latombe, J.C. and Wilson, R. H., Polyhedral Assembly Partitioning Using Maximally Covered Cells in Arrangements of Convex Polytopes, *Int. Journal of Computational Geometry and Applications,*vol 8:179–200, 1998.

[94] Berg, J. van den, Path Planning in Dynamic Environments, *PhD Thesis,* Utrecht University, The Netherlands, 2007.

[95] . Stappen, A.F. van der, Wentink, C., Overmars, M.H., Computing immobilizing grasps of polygonal parts, *Int. Journal of Robotics Research,* 19(5), pp. 467-479, 2000.

[96] Rimon, E. and Burdick, J., Mobility of Bodies in Contact–I: A 2nd Order Mobility Index for Multiple-Finger Grasps, *IEEE Trans. on Robotics and Automation,* vol. 14, no. 5, 1998.

[97] Cheong, J.-S., Stappen, A.F. van der, Goldberg, K., Overmars, M.H. and Rimon, E., Immobilizing hinged parts, *Int. Journal on Computational Geometry and Applications,* 17, pp. 45-69, 2007.

[98] Berretty, R-P. M., Geometric design of part feeders, *PhD Thesis,* Utrecht University, The Netherlands, 2000.

Further Information

For an introduction to robot arm kinematics, dynamics and control, see [12]. More on robotic manipulation can be found in [90] Robot motion planning and its variants are discussed in a number of books [30, 62, 61, 71]. Research in all aspects of robotics is published in the IEEE Transactions of Robotics and Automation and the International Journal of Robotics Research, as well as in the proceedings of the IEEE International Conference on Robotics and Automation and the International Symposium on Robotics Research [19]. The Workshop on Algorithmic Foundations of Robotics [21, 32] emphasizes algorithmic issues in robotics. Several computational geometry books contain sections on robotics or motion planning [6].