Algorithms and Theory of Computation Handbook, CRC Press, Boca Raton, NY, 1999

algorithms. It is presented at greater length in Section 2, along with other features of these algorithms. Section 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.

2 Underlying Principles

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 modelled, 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 modelling 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.

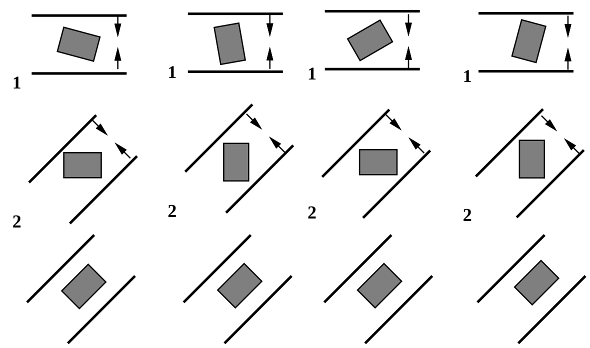

Figure 1: Orienting a convex polygonal part [Goldberg, 1993]

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. This is a typical problem in industrial part feeding (Subsection 3.1.3). If an overhead vision system is available 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 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.

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

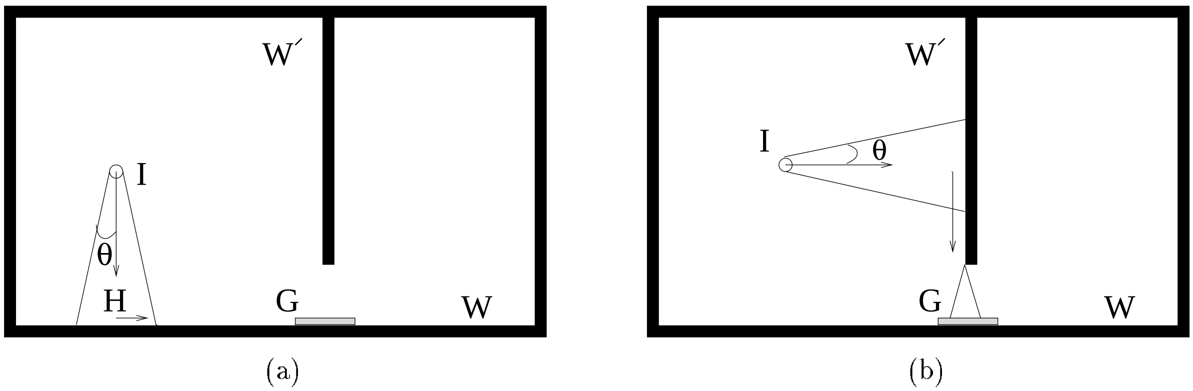

Figure 2: Goal recognition in mobile robot navigation

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 initial orientation of P [Goldberg, 1993].

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 Subsection 3.4 on nonholonomic robots).

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 modelled 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. 2.a) 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. 2.b). 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.

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 [Reif, 1979], 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 [Canny, 1988].

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 [Goldberg, 1993]. 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 not finding a solution (when one exists) tends rapidly toward 0 as the running time increases. In Section 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 3 (in particular, in Subsection 3.4 – nonholonomic robots – and in Subsection 3.5 – motion planning with uncertainty).

The potentially high cost of planning and the fact that it may 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 in an unpredictable way (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. 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 [Boddy and Dean, 1989]. 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 [Nourbakhsh, 1996].

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. For example, a typical planner computing a robot's path first discretizes the robot's free space (the set of collision-free **configurations** of the robot) in order to build a connectivity graph to which well-known search algorithms can be applied. 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 Subsection 2.1 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 3. Several of them use ideas and tools originally developed in Computational Geometry, for instance: plane sweep, topological sweep, computing arrangements.

Robot algorithms often require dealing with high-dimensional geometric spaces. Although criticality-based discretization methods apply to such spaces in theory (for instance, see [Schwartz and Sharir, 1983]), 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 path planners (Subsection 3.3.3).

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 (Subsection 2.1) 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 (Subsection 3.2). The number of fingers to safely grasp or fixture an object is another complexity measure (Subsection 3.1.1).

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 (Subsection 3.1.1), 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. Actually, 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 [Wilson and Latombe, 1995].

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 (Subsection 3.5). On the other hand, burying wires in the ground or placing enough infra-red 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.

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, motion planning, and (briefly) sensing. 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 Subsection 2.3. 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 [Halperin, 1997]. When criticality-based discretization is done in a space representing all possible motions, it yields the so-called "non-directional" data structures, which is another prevailing concept in robot algorithms and is exemplified in detail in Subsection 3.2.4.

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 (Subsection 3.3.3). 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 modelled as a compact manifold with boundary $B \subset \mathbb{R}^k$ ($k = 2$ or 3). B's boundary is also assumed piecewise-smooth.

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 immobilized for machining operations.

3.1.1 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 in 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 non-sticky, 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 [Mishra, Schwartz, and Sharir, 1987]:

- 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 (resp. 3-space) is that the finger wrenches $\boldsymbol{w}_i$ span a space F of dimension 3 (resp. 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 H of the finger wrenches [Mishra, Schwartz, and Sharir, 1987]. 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 non-sticky 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 [Mishra, Schwartz, and Sharir, 1987].

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 [Nguyen, 1988] 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 [Ponce et al., 1995].

3.1.2 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 [Zhuang, Goldberg, and Wong, 1994].

When the fixturing kit consists of a latticed workholding plane, three locators, and one clamp, [Brost and Goldberg, 1996] 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.

3.1.3 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 [Goldberg, 1993] and described in Section 2.1 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 [Akella et al., 1996]. Pushing is one form of nonprehensile manipulation. Work on pushing originated in [Mason, 1986] 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 [Erdmann and Mason, 1988] 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 [Akella et al., 1996].

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.

3.2.1 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.

3.2.2 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 (Subsection 2.4). 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 [Natarajan, 1988].

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 [Snoeyink and Stolfi, 1994].

3.2.3 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 [Natarajan, 1988]. 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 [Kavraki and Kolountzakis, 1995]. 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.

3.2.4 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 non-directional blocking graph (NDBG, for short) is proposed in [Wilson and Latombe, 1995] 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 (Subsection 2.3).

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 $\vec{d}$ is a blocking direction if translating P_i to infinity in direction $\vec{d}$ 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 its number of 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 [Wilson and Latombe, 1995] 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 takes $O(mc^5)$ time, where m is the number of pairs of parts in contact and c is the number of independent point-plane contact constraints. With the above algorithms, every feasible disassembly sequence can be generated in polynomial time.

3.3 Basic Path Planning

Motion planning is aimed at providing robots with the capability of deciding automatically which motions to execute. It arises in a variety of forms. The simplest —**basic path planning**— requires finding a geometric collision-free path for a single robot in a known static workspace. The path is represented by a curve segment connecting two points in the robot's configuration space [Lozano-Pérez, 1983]. This curve must not intersect a forbidden region, the C-obstacle region, which is the image of the workspace obstacles. Other motion planning problems require dealing with moving obstacles, multiple robots, movable objects, uncertainty, etc.

In this subsection we consider basic path planning. In the next three subsections we will review other motion planning problems and issues.

3.3.1 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 semi-algebraic sets with piecewise smooth boundaries.

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

The basic path planning problem is to compute a free or semifree path between two input configurations. A complete motion planner is 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.

3.3.2 Complete algorithms

Basic path planning for a three-dimensional linkage made of polyhedral links is PSPACE-hard [Reif, 1979]. 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 [Latombe, 1991].

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) [Latombe, 1991]. 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 [Schwartz and Sharir, 1983] 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 [Canny, 1988] 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 [Halperin and Sharir, 1996] takes $O((kn)^{2+\epsilon})$ time, where n is the total number of edges of the workspace, for any $\epsilon > 0$.

3.3.3 Probabilistic 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 time. One such scheme, probabilistic planning [Barraquand et al., 1997], avoids computing an explicit geometric representation of the free space. Instead, it uses an efficient procedure to compute distances between bodies in the workspace. It samples the configuration space by selecting a large number of configurations at random and retaining only the free configurations as milestones. It then checks if each pair of milestones can be connected by a collision-free straight path in configuration space. This computation yields the graph (V, E), called a probabilistic roadmap, where V is the set of milestones and E is the set of pairs of milestones that have been connected.

Various strategies can be applied to sample the configuration space. The strategy in [Kavraki et al., 1996] proceeds as sketched above. Once a roadmap has been computed, it is used to process an arbitrary number of path planning queries.

The results reported in [Barraquand et al., 1997] bound the number of milestones generated by the algorithm in [Kavraki et al., 1996], under the assumption that the configuration space verifies some simple geometric property. One such property is ϵ-goodness: a set S of volume μ is said to be ϵ-good, if every point in S sees a subset of S of volume at least $\epsilon \times \mu$. Under such an assumption, the number of milestones needed to correctly answer path planning queries with probability $1 - \alpha$ is proportional to $(1/\epsilon)(\log(1/\epsilon) + \log(1/\alpha))$.

3.3.4 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 [Khatib, 1986]. 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.

One may also construct grids at variable resolution. Hierarchical space decomposition techniques such as octrees and boxtrees have been used to that purpose [Latombe, 1991].

3.4 Path Planning for Nonholonomic Robots

In some cases, the configuration parameters of a robot are not directly controllable. This is in particular the case with nonholonomic robots. Informally, for such robots, the number of parameters that can be controlled is smaller than the number of parameters defining a configuration, which raises controllability issues introduced in Subsection 2.1.

3.4.1 Mathematical background

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

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

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[1] $T_{\boldsymbol{q}}(\mathcal{C})$ 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 points along the car's axis). 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 robots 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 [Barraquand and Latombe, 1993].

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

[1]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.

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 [Barraquand and Latombe, 1993]: *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 $2 + s$. Intuitively, the higher the degree of nonholonomy the more complex (and the slower) the robot's maneuvers to perform some motions.

3.4.2 Planning for controllable robots

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 non-feasible path can be decomposed into a finite number of subpaths, each short enough to be replaced by a feasible free subpath [Laumond et al., 1994]. 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 non-feasible 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 [Laumond et al., 1994]. The complexity of transforming a non-feasible 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 [Barraquand and Latombe, 1993] 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).

3.4.3 Planning for non-controllable robots

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 [Fortune and Wilfong, 1988] decides whether there exists such a path between two configurations, but it runs in time exponential in obstacle complexity. The algorithm in [Agarwal, Raghavan, and Tamaki, 1995] 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 [Barraquand and Latombe, 1993] require some sort of discretization.

3.5 Motion Planning with Uncertainty

In practice, robots deviate from planned paths due to errors in control and position sensing. Such errors raise recognizability issues which make planning more complex.

3.5.1 Problem formulation

The inputs to a motion planning problem with uncertainty are the initial region $I \subset \mathcal{C}$, in which the robot is known to be prior to moving, the goal region $G \subset \mathcal{C}$, in which it should terminate its motion, and the uncertainty in control and sensing. Uncertainty is specified in the form of regions. For instance, the uncertainty in position sensing is the set of possible actual robot's configurations given the sensor readings. The output is a series of motion commands, if one exists, whose execution enables the robot to reach G from I. Each command is described by a velocity vector $\boldsymbol{v}$ and a termination condition T. The vector $\boldsymbol{v}$ specifies the desired behavior of the robot over time (with or without compliance). The condition T is a boolean function of the sensor readings and time which causes the motion to stop as soon as it becomes true. A plan may contain conditional branchings. In [Canny, 1988] this problem is shown NEXPTIME-hard for a point robot moving in 3-space among polyhedral obstacles.

3.5.2 Preimage of a goal

Given a goal G and a command $(\boldsymbol{v}, T)$, a preimage of G is any region $P \subset \mathcal{C}$ such that executing the command from anywhere in P makes the robot reach and stop in G [Lozano-Pérez, Mason, and Taylor, 1984]. One way to compute a (non-maximal) preimage is to restrict the termination condition so that it recognizes G independently of the region from which the motion started [Erdmann, 1986]. For example, one may shrink G to a subset K, called the kernel of G, such that whenever the robot is in K, all robot configurations consistent with the current sensor readings are in G. A preimage is then computed as the region from which the robot commanded along $\boldsymbol{v}$ is guaranteed to reach K. This region is called the backprojection of K for $\boldsymbol{v}$. This preimage computation approach has been well studied in a polygonal configuration space when G is a polygon [Latombe, 1991].

3.5.3 One-step planning

In a polygonal configuration space, the kernel of a polygonal goal is either independent of the selected $\boldsymbol{v}$ or changes at a number of critical orientations of $\boldsymbol{v}$ that is linear in the workspace complexity [Latombe, 1991]. Moreover, the backprojection of a polygonal region K, when the orientation of $\boldsymbol{v}$ varies, changes topology only at a quadratic number of critical directions. Its intersection with a polygonal initial region I of constant complexity also changes qualitatively at few directions of $\boldsymbol{v}$. Checking the containment of I by the backprojection at each such direction yields a one-step motion plan, if one exists, in amortized time $O(n^2 \log n)$, where n is the number of edges in $\mathcal{C}$ [Briggs, 1995].

3.5.4 Multi-step planning

For multi-step planning, algebraic approaches that check the satisfiability of a first-order semi-algebraic formula have been proposed. In [Canny, 1989] it is assumed that all possible trajectories have an algebraic description. The approach there is based on a two-player-game interpretation of planning, where the robot is one player and nature the other. Each step of a plan contributes three quantifiers: one existential quantifier applies to the direction of motion, and corresponds to choosing this direction; another existential quantifier applies to time, and corresponds to choosing when to terminate the motion; one universal quantifier applies to the sensor readings and represents the unknown action of nature. The formula representing an r-step plan thus contains r quantifier alternations; checking its satisfiability takes double-exponential time in r, which is itself polynomial in the total complexity of the robot and the workspace.

3.5.5 Landmark-based planning

Often a workspace contains features that can be reliably sensed and used to precisely localize the robot. Each such landmark feature induces a region in configuration space called the landmark area from which the robot can sense the corresponding feature.

The planner in [Lazanas and Latombe, 1995] considers a point robot among n circular obstacles and $O(n)$ circular landmark areas. It assumes perfect position sensing and motion control in landmark areas. Outside these areas, it assumes that the robot has no position sensing and that directional errors in control are bounded by the angle θ. Given circular initial and goal regions I and G (with G intersecting at least one landmark area), the planner constructs a motion plan that enables the robot to move from landmark areas to landmark areas, until it reaches the goal. It proceeds backward by computing P_1 – the preimage of the landmark regions intersecting G. Then it computes the preimage P_2 of the landmark regions intersected by P_1, and so on, until a preimage contains I. The planner runs in $O(n^4 \log n)$ time; it is complete and generates plans that minimize the number of steps to be executed in the worst case.

3.6 Other Motion Planning Issues

There are many other useful extensions of the basic path planning problem [Latombe, 1991]. Below we briefly present some of them.

3.6.1 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. It can be represented 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 [Reif and Sharir, 1994]. 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 [Canny, 1988]. A complete planning algorithm is given in [Reif and Sharir, 1994] for a polygonal robot that translates in the plane among polygonal obstacles translating 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.

3.6.2 Coordination of 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.

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) [Kant and Zucker, 1986]. Although inherently incomplete, decoupled planning may work well in some practical applications.

3.6.3 Manipulation planning

Many robot tasks consist of achieving arrangements of physical objects. Such objects, called movable objects, cannot move autonomously; they must be moved by a robot. Planning with movable objects is called manipulation planning.

In [Wilfong, 1991] 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 [Alami, Laumond, and Siméon, 1995].

Heuristic algorithms have also been proposed. The planner in [Koga et al., 1994] 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.

3.6.4 Optimal planning

There has been considerable research in Computational Geometry on finding shortest Euclidean paths, 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 parametrization $s(t)$ 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 [Craig, 1986]. 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.

Minimal-time trajectory planning: Finding a minimal-time trajectory is called kinodynamic motion planning. One 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. Kinodynamic planning is NP-hard for a point robot under Newtonian mechanics in 3-space. The approximation algorithm in [Donald et al., 1993] computes a trajectory ϵ-close to optimal in time polynomial in both $1/\epsilon$ and the workspace complexity.

3.6.5 Discovery and on-line planning

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.

Early examples of on-line planners are reviewed in [Lumelsky, 1991]. Most of these planners apply to a point robot in 2-space that must go from a start position s to a goal position g

among unknown obstacles, each bounded by a Jordan curve of measurable length. The robot is equipped with perfect position and touch sensors. The planners select a mix of motions following either the line segment connecting s to g or boundaries of hit obstacles. The main consideration is the length of the generated path, expressed as a function of the distance between s and g, the number of obstacles, and their perimeters.

Another 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 [Papadimitriou and Yannakakis, 1991]. Competitive analysis is not restricted to path length and can be applied to other measures of performance as well.

3.7 Sensing

Sensing allows a robot to acquire information about its workspace and localize itself. Here we mention a few selected topics.

3.7.1 Model building

Consider a mobile robot in an unknown workspace W. A first task for this robot is likely to be the construction of a geometric model (also called a map) of W [Zhang and Faugeras,1996]. This requires the robot to perform a series of sensing operations at different locations. Each operation yields a partial model. The robot must patch together the successively obtained partial models to eventually form a complete map of the workspace. This problem is complicated by the fact that the robot has imperfect control and cannot accurately keep track of its position in a fixed coordinate system.

3.7.2 Robot localization

A robot often has to localize itself relative to its workspace W. A model of W is given and localization is done by matching sensory inputs against this model to infer the robot configuration. This problem usually arises for mobile robots. Other types of robots, such as robot arms, often have absolute references (e.g., mechanical stops) and internal sensors (e.g., joint encoders) that provide configurations more directly. Mobile robots have wheel encoders allowing dead-reckoning, but the absence of absolute reference on the one hand and slipping on the ground on the other hand usually require sensor-based localization. GPS (Global Positioning System) has recently become a widely available alternative, but it does not work in all environments.

Two kinds of sensor-based localization problems can be distinguished, static and dynamic. In the static problem, the robot is placed at an arbitrary unknown configuration and the problem is to compute this configuration. In the dynamic problem, the robot moves continuously and must regularly update its configuration. The second problem consists of refining an available estimate of the current configuration; but the computation must be done in real time. The static problem is usually more complex; but computation time is less restricted. A preprocessing approach to the static localization problem for a point robot equipped with a 360 degree range sensor is presented in [Guibas, Motwani, and Raghavan, 1996]. Practical techniques for localization are described in many papers [Talluri and Aggarwal, 1996].

3.7.3 Additional issues in sensing

Sensor placement is the problem of computing the set of placements from which a sensor can monitor a region within a given workspace [Briggs, 1995]. Another problem is to choose a minimal set of sensors and their placement so as to completely cover a given region. Additionally, there has been considerable interest in reconstructing shapes of objects using simple sensors, called probes. See [Skiena, 1997] for a review of problems and results in this area. Matching and aspect graphs are two related topics that have been well studied, mainly in computer vision.

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.

Algorithms have been proposed to efficiently compute distances between two convex bodies. In [Edelsbrunner, 1985], 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 [Gilbert et al., 1988] 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 example, see [Quinlan, 1994]. 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 [Lin and Canny, 1991]. It takes advantage of the fact that the closest features (faces, edges, vertices) change infrequently as the polyhedra move along finely discretized paths.

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 it is subject to the imperfectly modelled laws of nature. Robot algorithms uniquely blend controllability, observability, computational complexity, and physical complexity issues, as described in Section 2. Research on robot algorithms is broad and touches many different areas. In Section 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, planning with uncertainty, dynamic workspaces, and optimal-time planning), and sensing.

Many of the core issues reviewed in Section 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 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 [Mirtich et al., 1996].

- 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 [Barraquand et al, 1997].

- Establish a non-trivial lower bound on the complexity of planning for a nonholonomic robot that is not locally controllable.

6 Defining Terms

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.

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

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

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

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.

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

C-Obstacle: Given an obstacle B_i, the subset $CB_i \subseteq \mathcal{C}$ 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**.

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

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

Trajectory: Path indexed by time.

Free path: A path in free space.

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

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

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.

7 References

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

Akella, S., Huang, W., Lynch, K., and Mason, M.T. 1996. Planar Manipulation on a Conveyor with a One Joint Robot. In [Giralt and Hirzinger, 1996], p. 265-276.

Alami, R., Laumond, J.P., and Siméon, T. 1995. Two Manipulation Algorithms. In [Goldberg et al., 1995], p. 109-125.

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

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

de Berg, M., van Kreveld, M., Overmars, M. and Schwarzkopf, O. 1997. *Computational Geometry: Algorithms and Applications.* Springer, New York, NY.

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

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

Brost, R.C. and Goldberg, K.Y. 1996. Complete Algorithm for designing Planar Fixtures Using Modular Components. *IEEE Tr. on Systems, Man and Cybernetics*, 12:31-46.

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

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

Craig, J.J. 1986. *Introduction to Robotics. Mechanics and Control.* Addison-Wesley, Reading, MA.

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

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

Erdmann, M. 1986. Using Backprojections for Fine Motion Planning with Uncertainty. *Int. J. of Robotics Research*, 5:19-45.

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

Fortune, S. and Wilfong, G.T. 1988. Planning Constrained Motions. In *Proc. ACM Symp. on Theory of Computing*, p. 445-459.

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

Giralt, G. and Hirzinger, G. (editors) 1996. *Robotics Research*, Springer.

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

Goldberg, K.Y., Halperin, D., Latombe, J.C., and Wilson, R.H. (editors) 1995. *Algorithmic Foundations of Robotics*, A K Peters, Ltd., Wellesley, MA.

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

Halperin, D. 1997. Arrangements. In Goodman, J.E. and O'Rourke, J. (editors), *Handbook of Discrete and Computational Geometry*, CRC Press, Boca Raton, FL, p. 389-412.

Halperin, D. and Sharir, M. 1996. Near-Quadratic Algorithm for Planning the the Motion of a Polygon in a Polygonal Environment. *Discrete Computational Geometry*, 16:121-134.

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

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

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

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

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

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

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

Laumond, J.P. and Overmars, M. (editors) 1997. *Algorithms for Robot Motion and Manipulation* A K Peters, Ltd., Wellesley, MA.

Lazanas, A. and Latombe, J.C. 1995. Landmark-Based Robot Navigation. *Algorithmica*, 13:472-501.

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

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

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

Lumelsky, V. 1991. 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.

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

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

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

Natarajan, B.K. 1988. On Planning Assemblies. *Proc. 4th ACM Symp. on Computational Geometry*, p. 299-308.

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

Nourbakhsh, I.R. 1996. *Interleaving Planning and Execution.* PhD Thesis. Dept. of Computer Science, Stanford University, Stanford, CA.

Papadimitriou, C.H. and Yannakakis, M. 1981. Shortest Paths Without a Map. *Theoretical Computer Science*, 84:127-150.

Ponce, J., Sudsang, A., Sullivan, S., Faverjon, B., Boissonnat, J.D., and Merlet, J.P. 1995. Algorithms for Computing Force-Closure Grasps of Polyhedral Objects. In [Goldberg et al., 1995], p. 167-184.

Quinlan, S. 1994. Efficient Distance Computation between Non-Convex Objects. *Proc. IEEE*

Int. Conf. on Robotics and Automation, p. 3324-3329.

Reif J.H. 1979. Complexity of the Mover's Problem and Generalizations. *Proc. FOCS*, p. 421-427.

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

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

Skiena, S.S. 1997. Geometric reconstruction problems. In Goodman, J.E. and O'Rourke, J. (editors), *Handbook of Discrete and Computational Geometry*, CRC Press, Boca Raton, FL, p. 481-490.

Snoeyink, J. and Stolfi, J. 1994. Objects That Cannot Be Taken Apart with Two Hands. *Discrete Computational Geometry*, 12:367-384.

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

Wilfong, G.T. 1991. Motion Planning in the Presence of Movable Objects. *Annals of Mathematics and Artificial Intelligence*, 3:131-150.

Wilson, R.H and Latombe, J.C. 1995. Reasoning About Mechanical Assembly. *Artificial Intelligence*, 71:371-396.

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

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

8 Further Information

For an introduction to robot arm kinematics, dynamics and control, see [Craig, 1986]. Robot motion planning and its variants are discussed in [Latombe, 1991]. 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 [Giralt and Hirzinger, 1996]. The Workshop on Algorithmic Foundations of Robotics [Goldberg et al, 1995; Laumond and Overmars, 1997] emphasizes algorithmic issues in Robotics. Several Computational Geometry books contain sections on robotics or motion planning [de Berg et al., 1997].

Each mobile robot is built on an autonomous chassis and works using a Raspberry Pi computer with ROS. All the software is autonomous. A simple RGB-camera is used for receiving data from the environment. SLAM Constructor for ROS.Â In spite of the various algorithms which have already been proposed, an algorithm that robustly solves the problem in a general case and satisfies performance constraints is still a subject of research.