Completeness in Robot Motion Planning

Ken Goldberg, *University of Southern California, Los Angeles, CA, USA*

In robot motion planning, we say that an algorithm is complete *for a problem if it is guaranteed, for all instances of the problem, to find a solution when one exists and to return failure otherwise. Completeness is a desirable property. It provides a guarantee that the algorithm will work as expected for all inputs and thus can be dependably included in a larger system. This is especially important when algorithms are incorporated into industrial applications, where delays and failures can be extremely costly.*

In this paper we consider the completeness of several algorithms in robot motion planning. The two conditions for completeness, finding a solution if it exists, and terminating otherwise, suggest a distinction between algorithms that satisfy the former but not the latter: we refer to these as "exact" algorithms and give examples.

We then ask the stronger question: does a solution exist for all instances of a problem? We propose the term "solution-complete" to describe a property of problems and describe two recent results: a proof that a part feeding problem is solution-complete, and a proof that a fixturing problem is not solution-complete.

1 Introduction

Although robot motion planning is a general topic, we confine our discussion to the class of problems that can be formulated computationally, for example navigation, assembly, grasping, and the design of parts feeders. In all cases, we represent physical entities with mathematical models. Each problem defines a class of allowable inputs and a class of allowable solutions.

For example, the Piano Movers' problem asks if a robot can be moved from one configuration to another without colliding with obstacles. There are many versions of this problem depending on the class of allowable inputs (obstacles, robots, movements), and the class of solutions (shortest or safest paths, etc.). We might consider a version of the Piano Movers' problem when inputs are restricted to 2D polygonal obstacles and a 2D polygonal robot, and solutions are restricted to shortest paths consisting of pure translations of the robot.

Latombe's text [17] gives a comprehensive review of developments in Robot Motion Planning. Latombe presents the concept of "configuration space" (C-space) as a unifying theme. In the Piano Movers' problem, the initial and desired final configurations of the robot can be specified as points in the continuous manifold of C-space. A problem instance defines surfaces in the C-space; solutions can be specified as free paths through this space. C-space can also be used to formulate problems involving contact mechanics and friction, for example the space of grasp configurations for three point contacts on a 2D object could be specified with a 3D C-space where friction defines subsets of the space that provide frictional form closure of the object.

As demonstrated in [20], problems that include uncertainty in the initial configuration can also be formulated in this framework, where all possible configurations of the robot is represented as a subset of C-space and commanded actions map between sets. Actions that allow compliant contact with a known boundary can be chained together to collapse a subset of C-space to a desired point.

In robot motion planning, a solution does not always exist. For example, it is easy to construct instances of the Piano Movers' problem such that there is no free path from initial to final configurations. Similarly with problems that include uncertainty in the initial configuration: it is easy to construct cases where there is no sequence of commanded actions that allow the initial subset to be collapsed to the desired point. For other problems, such as grasping, it may be difficult to construct instances where a solution does not exist;

for some problems it may be possible to prove that a solution exists for all instances.

An *algorithm*, $\mathcal{A}$, for problem $\mathcal{P}$, accepts as input (a representation of) an instance of $\mathcal{P}$ and should return a solution. Some algorithms may fail to find a solution when one exists, or may fail to terminate. Examples will be given below. By convention, we say that $\mathcal{A}$ is *complete* for $\mathcal{P}$ if it is guaranteed to find a solution when one exists and to return failure otherwise [17, p.18][1].

Completeness is closely related to the way a problem is defined: algorithm $\mathcal{A}$ may not be complete for problem $\mathcal{P}$; however by adding additional assumptions limiting the class of inputs, it may be the case that $\mathcal{A}$ is complete for problem $\mathcal{P}'$.

Completeness is a desirable property. It provides a guarantee that the algorithm will work as expected for all inputs and thus can be dependably included in a larger system [8]. This is especially important when algorithms are incorporated into industrial applications, where a delay on the assembly line can cost thousands of dollars per minute. Completeness as specified above is related to the notion of *correctness* in program verification, where the behavior of a procedure is mathematically proved so that it can be reliably included in a large system.

As in program verification, the practice of proving completeness in robot motion planning is more often observed in the breach. Algorithms are often reported only with examples illustrating instances where they successfully find a solution. Completeness focuses attention on cases where algorithms may fail. There is a natural bias against thinking about such cases, and it can be extremely difficult to precisely characterize the class of inputs for which an algorithm is complete. Yet it is important to address this issue if robot motion planning algorithms are to be adopted outside of the laboratory.

In this paper we consider the completeness of several algorithms in robot motion planning. The two conditions for completeness, finding a solution if it exists, and terminating otherwise, suggest a distinction between algorithms that satisfy the former but not the latter; we will use the term "exact" to describe the former property.

Closely related to the completeness of algorithms is the stronger question: does a solution exist for all instances of a problem? For example, can we specify conditions on inputs and outputs such that a solution exists for all instances of some version of Piano Movers' problem? In some cases it is possible to prove completeness; more commonly, it is sufficient to demonstrate a single counterexample. We introduce the term "solution-complete" to characterize this property of problems. We review two recent results: a proof that a part feeding problem is solution-complete, and a proof that a fixturing problem is not solution-complete.

2 Incomplete Algorithms

Consider a navigation algorithm that uses a naive potential field to trace a path from the initial pose toward a desired pose using gradient descent. It is well known that greedy algorithms such as this can lead to local minima which may cause the algorithm to terminate without finding a path when one exists. Thus this algorithm is incomplete: it is not guaranteed to find a solution even when one exists[2].

Algorithms that use a uniform grid (lattice) to discretely sample a continuous solution space generally sacrifice completeness since a solution may be hidden in the interstices. Consider the problem, introduced by Peshkin and Sanderson [26], of finding a sequence of fence angles that will orient a given polygonal part on a conveyor belt as it brushes past the sequence of fences. Consider an algorithm that samples the range of fence angles at 10° increments and enumerates all arrangements of length 1, 2, and so on, until a solution is found. Since this algorithm may overlook solutions that contain fences at odd angles, it is not guaranteed to find a solution when one exists.

Furthermore, this algorithm considers longer and longer sequences of fences until a predetermined length

[1] The term "complete" has other meanings. In logic, a theory (set of sentences closed under logical implication) is *complete* if every sentence or its negation can be proved. In complexity analysis, a problem to which all problems in a class can be reduced is *complete* for this class (*e. g.* NP-Complete).

[2] Rimon and Koditscheck recently gave conditions under which it is possible to construct a potential field with a unique minimum. These conditions define a class of problems that are solution-complete. For these problems the potential field algorithm is complete [28].

limit is used to terminate the algorithm. Thus there are two ways in which this algorithm may fail to find a solution: it may overlook a short solution that falls between lattice points, or it may terminate before considering a longer solution that lies on the lattice.

It is important to note that floating-point approximations can cause an algorithm to overlook solutions, as this is equivalent to imposing a uniform (yet fine!) grid on the set of available solutions. An otherwise complete algorithm may fail to find a solution if it is buried beneath the resolution of its arithmetic. In computational geometry, the *real-RAM* model of computation is generally assumed for the purposes of proof, although it is notoriously difficult to implement [16]. For computing with angular quantities, one approach is to use rational, or "exact", arithmetic [4, 23].

3 Exact Algorithms

Suppose an algorithm satisfies only the first the condition for completeness: it is guaranteed to find a solution when one exists. Such algorithms are thorough in that they will not overlook potential solutions; yet they are not necessarily complete. As suggested by John Reif, we use the term "exact" to describe such algorithms.

Consider a navigation algorithm that uses quad-tree decomposition to subdivide the configuration space. At each iteration the space is further divided in search of a collision-free path from start to goal. If a path exists, this algorithm will eventually find it. However, if a path does not exist, the algorithm may continue searching with finer and finer decompositions and may not terminate. This algorithm is exact but not complete[3].

One characteristic of exact robot motion planning algorithms is that they often partition a continuous C-space space into equivalence classes (cells) based on the geometry of the environment. For example, part vertices might define critical angles in a partition of C-space. Partitions induced by a collection of hyperplanes are called "arrangements" in Computational Geometry, which seeks precise bounds on the size of the partition [15].

[3]Suppose we terminate the decomposition at a prespecified resolution. Latombe uses the term "resolution-complete" for the property that an algorithm is guaranteed to find a solution when a solution exists at that resolution.

4 Complete Algorithms

An algorithm is complete if it is exact and guaranteed to terminate with a negative report if a solution does not exist. A good example is the Visibility Graph algorithm for navigation, which searches a graph that contains a link between all pairs nodes that are "visible" in the configuration space. The algorithm is guaranteed to find the shortest path or to report that no path exists.

Other examples are Erdmann and Mason's tray tilting planner [10] and a search-based algorithm for planning sequences of grasp motions to orient polygonal parts [13]. Both of these algorithms partition the angular space of actions into a finite number of equivalence classes based on part shape and then enumerate all combinations of elements in this partition. Note that we can terminate a search path when we reach a state that has already been encountered since there is never any advantage to looping. Since there are a finite number of possible actions, and a finite number of possible state sets (the power set), these search based planners are guaranteed to find a plan if one exists and to eventually terminate otherwise.

Another example where the problem can be converted to a graph search is Erdmann, Mason, and Vanecek's algorithm for orienting three-dimensional parts [9]. Given an n-sided polyhedral part resting on a planar table, the objective is to find a sequence of tilting angles for the table that will bring a particular part face into contact with the table (thereby eliminating all but one degree of rotational freedom). The authors gave a graph-based algorithm that is guaranteed to find a plan if one exists and to report failure otherwise.

These algorithms test each cell in a finite partition of solution space. If a solution exists at any point in a cell, then all points in that cell are solutions. Thus the algorithm is exact and is guaranteed to terminate. Such partitions also characterize several algorithms based on what Donald calls a "non-directional backprojection". Donald originally proposed this partition for planning compliant motions in the plane [7]. It has since been applied to planning navigation with landmarks [19] and to Assembly Planning [30]. All of these algorithms are complete.

A related problem is finding grasps for polygonal parts. Here the solution can be characterized as 4

points along the continuous boundary of the part that achieves form closure (the contacts can resist any applied forces and torques). Nguyen [25] partitioned the part boundary into equivalence classes and gave an algorithm that finds a form closure grasp, if one exists, in time $O(n^4)$.

A variation of grasp planning restricts contacts to lie on a regular lattice, as is the case for modular fixtures. Modular fixtures are gaining wide use for flexible manufacturing and job shop machining. A modular fixture is an arrangement of fixture elements (fixels) that will locate and securely hold a given part. Recently, polynomial-time algorithms have been reported that are guaranteed to find a form closure fixture if one exists [3, 29].

5 Solution-Complete Problems

For algorithms that are complete, it is natural to ask: When does a solution exist?. In path planning, it is easy to construct cases where a solution does not exist. This is also true for motion planning with uncertainty by making the initial set of possible configurations sufficiently large. As stated above, it is sometimes possible to prove that a solution exists for all instances of a problem. In such cases we say that a problem is solution-complete. In other cases, we can construct a counterexample to prove the converse.

There are several examples in robot motion planning. One is Akella and Mason's proof that in the absence of obstacles, it is always possible to position and orient a polygonal part by pushing [1]. They prove this by defining a general linear program for the problem and using linear algebra to show that a positive solution must exist.

Another is Barraquand and Latombe's proof for a mobile robot, subject to (nonholonomic) rolling constraints, whose steering angle can take on only two distinct angles. The authors showed that in the absence of obstacles, a path always exists between any two planar configurations [2].

For a one-dimensional version of tray tilting, [9] showed that a plan always exists as long as the polygonal part has a unique angle in its transition diagram.

And although lower bounds on the number of contacts required to achieve form closure were known since the turn of the century, [24, 22] recently proved upper bounds. In particular, that for any piecewise-smooth compact connected planar body, excluding surfaces of revolution, a form-closure grasp with 4 contacts always exists.

Solution completeness is similar to the notion of *controllability* in Control Theory: a linear system is controllable if, for any state of the system, there exists a solution (a control function) which will drive the system to the zero state in finite time [6, 18]. For example, it is known that in the absence of obstacles, a wheeled robot can reach any configuration in the plane. We might say that from any initial configuration we can always drive the robot to the zero configuration. Lynch and Mason recently used results from non-linear control theory to give conditions under which a plan exists to push a passive part to the zero configuration using a planar fence [21]. In such cases we say that the part is "controllable". It would be interesting to characterize a class of obstacles such that a plan always exists to reach the zero configuration. However it is not clear that results from Control Theory can be applied to this type of problem.

In the next section we describe a problem in Robot Motion Planning with Uncertainty and show that the problem is solution-complete.

6 Proving Solution-Completeness

Consider a planning algorithm that finds plans to orient a given part using a parallel-jaw gripper: given a list of n vertices describing a polygonal part whose initial orientation is unknown, find the shortest sequence of gripper actions that is guaranteed to orient the part up to symmetry in its convex hull. We prove that a plan exists for all polygonal parts and thus that the problem is solution-complete. The proof is based on an algorithm described in greater detail in [12].

When a polygonal part is grasped with the frictionless gripper, it assumes one of a finite number of "stable" configurations where at least one edge of the part's convex hull is in contact with a jaw. The outcome can be predicted with the **squeeze function**,$s : S^1 \rightarrow S^1$, such that if θ is the initial orientation of the part with respect to the gripper, $s(\theta)$ is the orientation of the part with respect to the gripper after the squeeze action is completed. For a polygonal part, the piecewise-constant squeeze function is derived as follows.

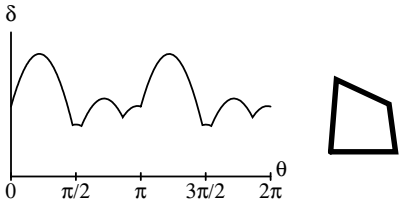

Figure 1: *A part and its width function.*

The **width function** for a $2D$ part is the distance between parallel supporting lines at angle θ as shown in Figure 1. All orientations that lie between a pair of adjacent local maxima in the width function will map to the orientation corresponding to the enclosed local minimum, *i. e.* the squeeze function is constant over this interval of orientations. The result is a step function.

We assume that all steps in the squeeze function are closed on the left and open on the right. Strictly speaking, the squeeze function has value $s(\theta) = \theta$ at its discontinuities, corresponding to an unstable equilibrium where the part is wedged between two exactly-aligned vertices. We could define a squeeze action at angle α to include closing and opening the gripper at angle α followed by closing the gripper at angle $\alpha + \epsilon$, rotating the gripper by $-\epsilon$, and then opening the gripper. In [14], we show how to find an appropriate ϵ for any polygonal part such that the combined action has a piecewise constant transfer function where each step is closed on the left and open on the right. In practice however, mechanical vibration in the gripping mechanism is sufficient to dislodge such wedged configurations, and after the first squeeze action causes the part to rotate into a stable configuration, the plan's margin for error (specified below) allows us to avoid actions that could produce a wedged configuration. For more on this issue, see [5].

In what follows, we use the term **interval** to refer to a connected subset of S^1. For an interval Θ, let $|\Theta|$ denote its Lebesgue measure. We define an **s-interval** to be a semiclosed interval of the form $[a, b)$ such that a, b are points of discontinuity in the domain of squeeze function s. For an s-interval Θ_X, let θ_X refer to its included bounding point. Since there are $O(n)$ discontinuities in the squeeze function, there are $O(n^2)$ unique s-intervals , each of which has non-zero measure. We define the **s-image** of a set, $s(\Theta)$, to be the smallest interval containing the following set: $\{s(\theta)|\theta \in \Theta\}$. Note that the s-image of any set will be a closed interval.

The algorithm begins with an s-interval whose image is a point. It continues, finding larger and larger s-intervals . When the algorithm terminates, the resulting sequence of s-intervals can be transformed into a sequence of squeeze actions that, in effect, "funnel" the largest s-interval into a unique final orientation. The algorithm is given below.

1. Compute the squeeze function.
2. Find the widest single step in the squeeze function and set Θ_1 equal to the corresponding s-interval . Let $i = 1$.
3. While there exists an s-interval Θ such that $|s(\Theta)| < |\Theta_i|$,
   - Set Θ_{i+1} equal to the widest such s-interval .
   - Increment i.
4. Return the list $(\Theta_1, \Theta_2, ..., \Theta_i)$.

We illustrate the algorithm using the squeeze function for the rectangular part as shown in Figure 2. Since this part has aspect ratio 1.5, let $a = \text{atan2}(3, 2)$.

In step 2 of the algorithm, the widest single step is found and Θ_1 is set to be the corresponding s-interval on the horizontal axis: $[\pi - a, \pi + a)$. Note that $s(\Theta_1)$ is the unique orientation at angle π.

In step 3 of the algorithm, we seek the widest s-interval whose s-image has smaller measure than Θ_1. As illustrated in Figure 3, this can be visualized by aligning the lower left corner of a box of dimension $|\Theta_1|$ with the leftmost point from each step in the squeeze function. If the squeeze function emerges from the right edge of the box, then the s-image of the corresponding s-interval has smaller measure than Θ_1. The largest such s-interval in this case is $\Theta_2 = [\pi - a, 2\pi - a)$. Note that $s(\Theta_2) = [\pi, 3\pi/2]$, $|s(\Theta_2)| = \pi/2 < |\Theta_i| = 2a$,

Continuing in this manner, wider and wider s-intervals are found until the loop terminates. This will occur when $|\Theta_i| = T$, a period of symmetry in the squeeze function. For the rectangular part, the algorithm terminates with $i = 2$ since $|\Theta_2| = \pi$.

Theorem 1 *For any polygonal part, we can always find a plan to orient the part up to symmetry.*

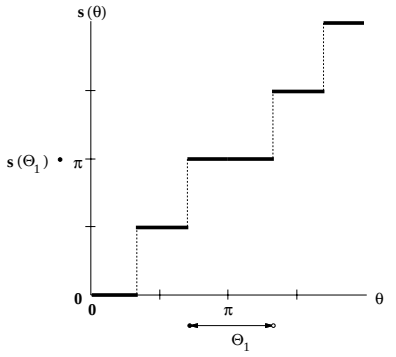

Figure 2: *In step 2, the widest single step in the squeeze function is identified. All the orientations in Θ_1 map into the single final orientation, $s(\Theta_1)$.*

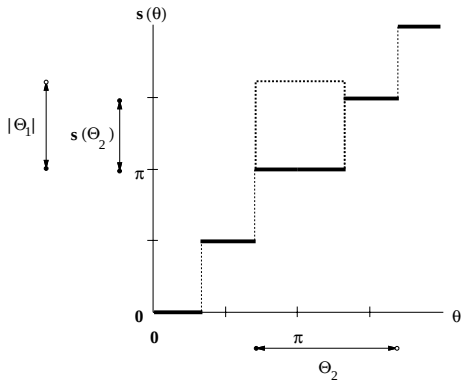

Figure 3: *Illustrating step 3 of the algorithm.*

Proof. Any polygonal part will generate a piecewise-constant squeeze function, $s : S^1 \rightarrow S^1$, where all s-intervals have non-zero measure and $s(\theta + T) = s(\theta) + T$, where T is a period of symmetry. To simplify the problem of wraparound at 0, in this section we extend s to a function on the real line, $s : \Re \rightarrow \Re$, that has exactly the same value as before on $[0, 2\pi)$. Elsewhere, it is specified as $s(\theta + T) = s(\theta) + T$.

We prove the claim by showing that for any such squeeze function, we can always find a sequence of s-intervals , $(\Theta_1, \Theta_2, ..., \Theta_i)$, of increasing measure with the condition that Θ_j has larger measure than the s-image of Θ_{j+1}. In other words, we must show that for any piecewise-constant monotonic squeeze function and any s-interval , we can always find a larger s-interval unless it corresponds to a period of symmetry in the squeeze function.

Let h be the measure of some s-interval . Either we can find a larger s-interval whose s-image is smaller than h,

$$\exists\theta, s(\theta + h) - s(\theta) < h, \tag{1}$$

Or h is a period of symmetry in the squeeze function:

$$\forall\theta, s(\theta + h) = s(\theta) + h, \tag{2}$$

where the quantifiers range over the interval $[0, T)$.

To understand formula 1, consider that we've reached a point in the algorithm where the current s-interval is $\Theta_j = [\theta_j, \theta_j + h)$. Formula 1 says that there is some closed interval, $\Theta = [\theta, \theta + h]$, whose s-image is smaller than Θ_j. We can expand Θ (without increasing its s-image) by extending it to the right until we reach a discontinuity in the squeeze function. This yields an s-interval whose s-image is smaller that Θ_j. The difference between this s-interval and Θ is an open interval and hence has nonzero measure. Thus this s-interval will have larger measure than Θ_j.

We can also interpret formula 1 with reference to figure 3. The formula states that we can always find a position for the lower left hand corner of the box such that the squeeze function enters on the left edge of the box and exits on the right edge.

To show that for any such squeeze function and any h, either formula 1 or formula 2 must hold, consider the integral of the function $s(\theta + h) - s(\theta) - h$ over the interval $[0, T)$.

$$\begin{aligned}
& \int_0^T [s(\theta + h) - s(\theta) - h]d\theta \\
= & \int_h^{T+h} s(\theta)d\theta - \int_0^T s(\theta)d\theta - hT & (3)\\
= & -\int_0^h s(\theta)d\theta + \int_T^{T+h} s(\theta)d\theta - hT & (4)\\
= & -\int_0^h s(\theta)d\theta + \int_0^h [s(\theta) + T]d\theta - hT & (5)\\
= & -\int_0^h s(\theta)d\theta + \int_0^h s(\theta)d\theta + hT - hT & (6)\\
= & \ 0. & (7)
\end{aligned}$$

Since this integral is zero, either there is some point where the function is less than zero (formula 1 is true),

or the function is uniformly zero (formula 2 is true, i.e. $h = T$).

Hence we can always continue to find larger s-intervals until we reach a period of symmetry in the squeeze function. We have shown earlier that we can transform this sequence of s-intervals into a plan to orient the part up to symmetry. Thus we've shown that a plan exists for all polygonal parts and hence that the problem is solution-complete. □

Rao and Goldberg recently extended this result to the class of algebraic parts [27]. We note that a proof of solution-completeness for problem $\mathcal{P}$ has consequences for its algorithms: if $\mathcal{A}$ is an exact algorithm for problem $\mathcal{P}$ and $\mathcal{P}$ is solution-complete, then $\mathcal{A}$ is complete. For example, the result given in this section implies that the exact algorithm in [13] is complete.

7 Disproving Solution-completeness

Brost and Goldberg [3] recently gave a complete algorithm that is guaranteed to find a form closure fixture for a given polygonal part in polynomial time and to terminate with a negative report otherwise. For all parts that we tested, the algorithm found dozens of solutions. This leads us to speculate that the problem is solution-complete. In this section we show that it is not.

The algorithm considers a class of fixtures using 4 frictionless point contacts: 3 locators and a clamp (3L/1C) such that each are attached to a square lattice of point holes. In particular, is any convex polygonal part fixturable in this fashion? Clearly if the part is very small with respect to the lattice spacing it may fall between the holes and thus not be fixturable. However, we might conjecture that for parts of sufficient "width" the method is complete: it is always possible to find a 3L/1C fixture to hold the part. Below we construct a counterexample; details can be found in [31].

The maximum and minimum values of the width function are well defined; we denote them with $\bar{d}(S)$ and $\underline{d}(S)$, respectively. The minimum width of a part is a useful way to characterize its size.

To construct a polygonal part of arbitrary size that is unfixturable, we show that for any given positive number M, we can construct a disk of size $> M$ that can make contact with at most 2 lattice sites. We then show how to transform this disk into a regular polygon while preserving this property. Since this polygon can make contact with at most 2 locators, it cannot be fixtured under the 3L/1C model.

From geometry, we know that any three noncolinear points uniquely determine a circle. Thus, every non-trivial lattice site triplet, t, determines a disk which we denote by $s(t)$. If two triplets determine a disk of the same width, we say that these triplets are *equivalent*.

Let the *maximum width* of a triplet be the length of the longest side of the triangle it determines.

Lemma 1 *For any given width there exists a* **disk** *of greater width that can achieve contact with at most two lattice sites.*

Proof. For any given positive number M, let $S(M)$ be the set of disks with widths between M and $M+1$ defined by triplets of lattice sites. $S(M)$ is finite since any disk $s(t)$ has a width no less than the maximum width of t and the number of non-trivial triplets with maximum width less than $M+1$ is finite. Let us define the set of widths as

$$D(M) = \{d(s(t) \mid s(t) \in S(M)\}$$

Then $D(M)$ is a finite set. Let c be any width in the interval $(M, M+1]$ not in $D(M)$, the disk with c as its width can achieve contact with at most two fixel points. *(Figure 4)* □

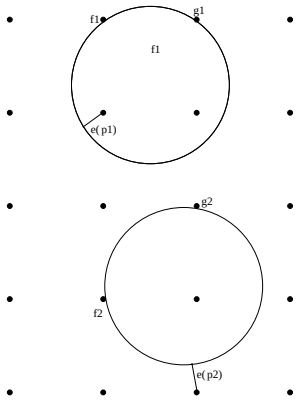

Figure 4: *The disk of indicated width can achieve contact with at most two lattice sites*

Based on this disk, we now construct a polygon that can achieve contact with at most two fixel contacts.

Theorem 2 *For any given width there exists a* **convex polygon** *of greater width that can achieve contact with at most two lattice sites.*

Proof. For any given positive number M, let S_c denote the disk with the width c, which is constructed in the lemma 1. By the proof of the lemma 1, S_c has at most two fixels on its boundary. Obviously for any fixel pair with a length less than or equal to c, we can always locate the disk S_c such that the two fixel points are on its boundary. Therefore we only need to consider the set of all non-identical pairs of sites with length less than or equal to c. Let us denote this set by Q. It is clear that Q is a finite set. Assume $|Q| = k, k > 0$,then we can represent Q as $Q = \{p_i \mid 1 \leq i \leq k\}$. We define $\epsilon(p)$ as the minimum distance from any fixels, other than the coordinates of p, to the boundary of the disk ∂S_c. Then we have $\epsilon(p_i) = \inf\{d_{\partial S_c}(f) \mid f \in F, f \notin p_i\} > 0$, for all $1 \leq i \leq k$, by the construction of the disk S_c. Let $\epsilon = \min\{\epsilon(p_i) \mid 1 \leq i \leq k\}$, then we have $\epsilon > 0$. There exists an inscribed regular polygon, P, of S_c, such that the length of its side is less than $\frac{1}{2}\epsilon$. *(Figure 5)* In order to achieve this, we only need to choose the number of sides of P, denoted by N, large enough, since the length of the side of P, denoted by L, satisfies the following

$$L = c \sin \frac{\pi}{N}.$$

Therefore we only need choose

$$N > \frac{\pi}{\sin^{-1} \frac{\epsilon}{2c}}.$$

Since $c > M$ (by the construction of S_c in the lemma 1), then we can select N to be large enough, such that $\underline{d}(P) > M$.

We denote the maximum distance between P and S_c by δ. Since $N \geq 3$, then

$$\delta = \frac{1}{2} L \tan \frac{\pi}{2N} < \frac{1}{2} L < L < \frac{1}{2}\epsilon.$$

Claim: Such a polygon, P, can achieve contact with at most two fixel points.

Proof of Claim: Let f, g and h be three fixel points, among which each pair has a length no greater than c. (If some pair has a length greater than c, then they cannot form a three-point contact, due to the fact that the maximum width of P is c.) Without loss of generality, we assume that P has f and g as its $two-point$ contact, and the contacting edges are e_f and e_g, respectively. Now we put P inside the disk S_c such that P is a inscribed polygon of S_c, then we position the disk (with P inscribed) on the lattice, such that f and g are on the arcs corresponding to e_f and e_g, respectively. (This can be always achieved.) By the construction of S_c, we know that $d_{\partial S_c}(h) \geq \epsilon$. Therefore, by the construction of the equilateral polygon P, the distance from h to the boundary of P is greater than $\frac{1}{2}\epsilon$, namely $d_{\partial P}(h) > \frac{1}{2}\epsilon$. In order to achieve $two-point$ contact on the edges e_f and e_g, we can always rotate P inside the disk S_c first, then translate it to achieve the contact. After the rotation, f and g will be still on the corresponding arcs. Therefore the distance needed for translation is less than $\frac{1}{2}\epsilon$, because the distance of the translation is bounded by the length of the side of P, which is less than $\frac{1}{2}\epsilon$. But this translation won't be enough to make h the third point contact, since h is distant from the edge of P more than $\frac{1}{2}\epsilon$.

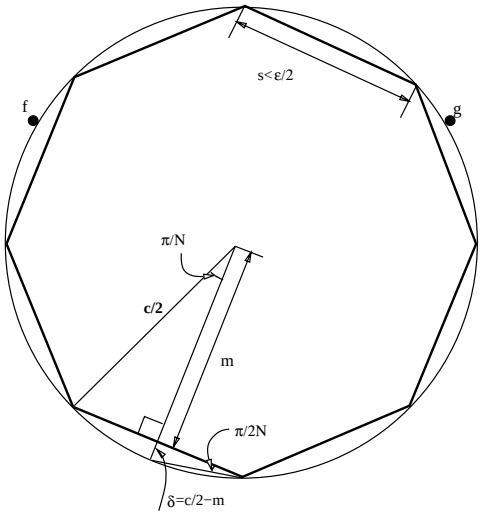

Figure 5: *The construction of the polygon that does not have a* 3L/1C *design.*

Hence P can achieve contact with at most two lattice sites.

□

Thus we have shown how to construct an infinite set of polygonal parts that cannot be fixtured using 3 point locators and a point clamp and so for any of these polygons the planning algorithm will terminate with no solutions. Thus the problem is not solution-complete.

Perhaps by suitably restricting the set of admissible parts we can show that a fixture always exists; this is currently an open problem.

8 Discussion

In this paper we focused on the issue of completeness in robot motion planning and specified three properties:

- An algorithm is *exact* if it is guaranteed to find a solution if one exists.
- An algorithm is *complete* if it is exact and guaranteed to terminate with a negative report if a solution does not exist.
- A problem is *solution-complete* if a solution exists for all instances.

There are many subtleties that are not addressed in this discussion, such as categories for probabilistic algorithms that terminate with probability one [11], and the relationship between completeness and complexity. Complexity analysis generally requires showing completeness: it is not clear how to define the asymptotic complexity of an algorithm that is not guaranteed to terminate. For solution-completeness, it is possible in some cases to prove that solutions always exist without providing a constructive algorithm for finding them. Similarly, the asymptotic complexity of the decision problem may be lower than the complexity of finding a solution.

As noted in the Introduction, algorithmic completeness bears some relation to the notion of correctness in program verification. In fact, Rimon and Koditschek use this language when they state that their results yield a "potential-function-based robot navigation algorithm that is provably correct" [28]. If these notions are identical, then why is "completeness" preferred in the motion planning literature? Etymology suggests that complete ("to make full") may be more appropriate than correct ("to make straight")!

We should note that all of our definitions are with respect to mathematical problems . The relationship between mathematical models and the physical world offers additional complications. For example, consider the navigation algorithm that uses the Visibility Graph. Although it is complete for the idealized mathematical model of the environment, this plan may perform extremely poorly when executed in the presence of disturbances in the physical environment.

Although the issue of completeness is familiar in Computational Geometry, it is often neglected in the Robotics literature. This may be due in part to the experimental nature of much robotics research, where empirical demonstrations are more common than mathematical proofs. Certainly it is often difficult to formally characterize complex problems in robotics; in many cases the abstractions used to simplify problems, such as assuming zero friction, may be of little relevance to practitioners. Yet empirical demonstrations of success without careful examination of failures leaves the impression that algorithms will work in all cases. This can lead to unpleasant surprises on the factory floor and hurt the credibility of future efforts.

As with asymptotic complexity, completeness focuses on worst-case scenarios. There is a natural bias against such "pessimism". Yet careful examination of such cases may lead to a proof of completeness or a counterexample, both of which are useful to practitioners. Completeness is also relevant to experimenters; although experiments cannot demonstrate completeness, they can be used to identify counterexamples. My hope is that focusing on these cases will ultimately lead to better algorithms.

Acknowledgements

This work was supported by the National Science Foundation under Award IRI-9123747 and by National Young Investigator Award IRI-9457523. I'm grateful to my students Yan Zhuang, Hadi Moradi, and Rick Wagner, and to the workshop participants, especially John Canny, Mike Erdmann, Danny Halperin, Dave Kriegman, Jean-Claude Latombe, Kevin Lynch, Anil Rao, John Reif, and Randy Wilson for their insightful feedback on early drafts of this paper.

References

[1] Srinivas Akella and Matthew T. Mason. An open-loop planner for posing polygonal objects in the plane by pushing. In *International Conference on Robotics and Automation*. IEEE, May 1992.

[2] Jerome Barraquand and Jean-Claude Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning the the presence of obstacles. *Algorithmica*, 10(2):121–155, August 1993. Special Issue on Computational Robotics.

[3] Randy Brost and Ken Goldberg. A complete algorithm for synthesizing modular fixtures for polygonal parts. Technical Report SAND 93-2028, Sandia National Laboratories, Oct 1993.

[4] John Canny, Bruce Donald, and Gene Ressler. A rational rotation method for robust geometric algorithms. December 1991.

[5] Yui-Bin Chen and Doug Ierardi. The complexity of non-adaptive plans for orienting polygonal parts. Technical Report USC-CS-92-502, University of Southern California, December 1991.

[6] Bruce Donald. Guest editor's forward: The geometric theory of manipulation, planning and control. *Algorithmica*, 10(2):91–101, August 1993. Special Issue on Computational Robotics.

[7] Bruce R. Donald. The complexity of planar compliant motion planning under uncertainty. In *Fourth ACM Symposium on Computational Geometry*, 1988.

[8] Michael Erdmann. Personal Communication, 1990.

[9] Michael Erdmann, Matthew T. Mason, and George Vanecek Jr. Mechanical parts orienting: The case of a polyhedron on a table. *Algorithmica*, 10(2), August 1993. Special Issue on Computational Robotics.

[10] Michael A. Erdmann and Matthew T. Mason. An exploration of sensorless manipulation. In *IEEE International Conference on Robotics and Automation*, 1986. Also appears in IEEE Journal of Robotics and Automation, V. 4.4, August 1988.

[11] Mike Erdmann. Randomization in robot tasks. *IJRR*, 11(5), Oct 1992.

[12] Ken Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2):201–225, August 1993. Special Issue on Computational Robotics.

[13] Ken Goldberg and Matthew T. Mason. Bayesian grasping. In *International Conference on Robotics and Automation*. IEEE, May 1990.

[14] Ken Goldberg and Anil Rao. Orienting planar parts. Technical Report IRIS-275, University of Southern California, September 1991.

[15] Dan Halperin and Micha Sharir. Arrangements and their applications in robotics: Recent developments. In *The First Workshop on the Algorithmic Foundations of Robotics*. A. K. Peters, Boston, MA, 1994.

[16] C. M. Hoffman, J. E. Hopcroft, and M. S. Karasick. Towards implementing robust geometric computations. In *4th Symp. on Computational Geometry*. ACM, 1988.

[17] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Press, 1991.

[18] Jean-Claude Latombe. Robot algorithms. In Takeo Kanade and Richard Paul, editors, *Robotics Research: The Sixth International Symposium*, 1993.

[19] Anthony Lazanas and Jean-Claude Latombe. Landmark-based robot navigation. Technical Report STAN-CS-92-1428, Stanford CS, May 1992.

[20] T. Lozano-Perez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, Spring 1984.

[21] Kevin Lynch and Matt Mason. Stable pushing: Mechanics, controllability, and planning. In *The First Workshop on the Algorithmic Foundations of Robotics*. A. K. Peters, Boston, MA, 1994.

[22] Xanthippi Markenscoff, Luqun Ni, and Christos H. Papadimitriou. The geometry of grasping. *IJRR*, 9(1), February 1990.

[23] Matthew T. Mason. exact angle arithmetic package in commonlisp. (Ported to C++ by Jeff Wiegley (USC) in 1993)., August 1988.

[24] Bud Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2(4):641–558, 1987.

[25] Van-Duc Nguyen. Constructing force-closure grasps. *International Journal of Robotics Research*, 7(3), 1988.

[26] Michael A. Peshkin and Art C. Sanderson. Planning robotic manipulation strategies for workpieces that slide. *IEEE Journal of Robotics and Automation*, 4(5), October 1988.

[27] Anil Rao and Ken Goldberg. Manipulating algebraic parts in the plane. Technical Report 318, IRIS, December 1993. (To appear in the IEEE Transactions on Robotics and Automation).

[28] Elon Rimon and Dan Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5), October 1992.

[29] Aaron Wallack and John Canny. Planning for modular and hybrid fixtures. In *International Conference on Robotics and Automation*. IEEE, May 1994.

[30] Randy Wilson and Jean-Claude Latombe. One the qualitative structure of a mechanical assembly. In *National Conference on Artificial Intelligence*, 1992.

[31] Yan Zhuang, Yin-Chung Wong, and Ken Goldberg. On the existence of modular fixtures. In *International Conference on Robotics and Automation*. IEEE, May 1994. Also available as USC Techreport IRIS-93-314.