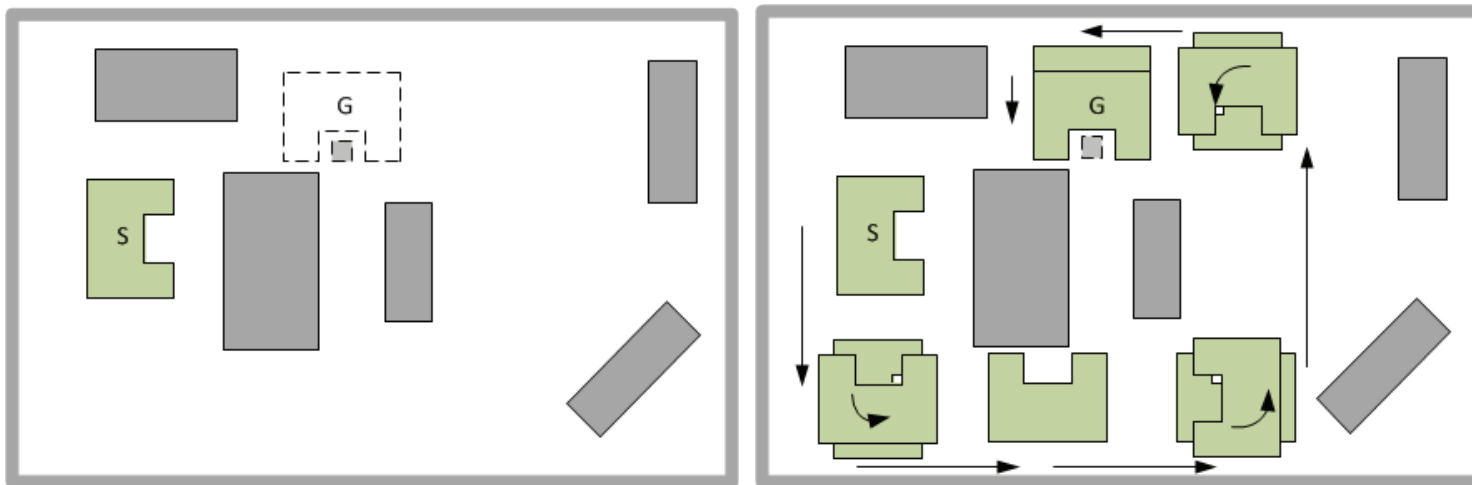


## Introduction to Motion Planning

A fundamental need in robotics is to have algorithms that convert high-level specifications of tasks from humans into low-level descriptions of how to move: motion planning and trajectory planning.

A classical version of motion planning is sometimes referred to as the Piano Mover's Problem.



Robot motion planning usually ignores dynamics and other differential constraints and focuses primarily on the translations and rotations required to move the piano.

Recent work, however, does consider other aspects, such as uncertainties, differential constraints, modeling errors, and optimality.

Trajectory planning usually refers to the problem of taking the solution from a robot motion planning algorithm and determining how to move along the solution in a way that respects the mechanical limitations of the robot.

In artificial intelligence, the terms planning and AI planning take on a more discrete flavor. Instead of moving a piano through a continuous space, as in the robot motion planning problem, the task might be to solve a puzzle, such as the Rubik's cube or a sliding-tile puzzle, or to achieve a task that is modeled discretely, such as building a stack of blocks. Although such problems could be modeled with continuous spaces, it seems natural to define a finite set of actions that can be applied to a discrete set of states and to construct a solution by giving the appropriate sequence of actions.

|   |    |    |    |
|---|----|----|----|
| 1 | 13 | 8  | 3  |
| 9 | 6  | 11 | 15 |
| 4 | 14 | 2  | 10 |
| 7 | 12 | 5  |    |

Natural questions at this point are, What is a plan? How is a plan represented? How is it computed? What is it supposed to achieve? How is its quality evaluated? Who or what is going to use it?

Regarding the user of the plan, it clearly depends on the application. In most applications, an algorithm executes the plan; however, the user could even be a human. Imagine, for example, that the planning algorithm provides you with an investment strategy.

Robot (Robotics), agent (AI), controller (Control Theory), decision-maker or player (game theory) Planning algorithms: find a strategy for one or more decision makers

## Motivational Examples and Applications

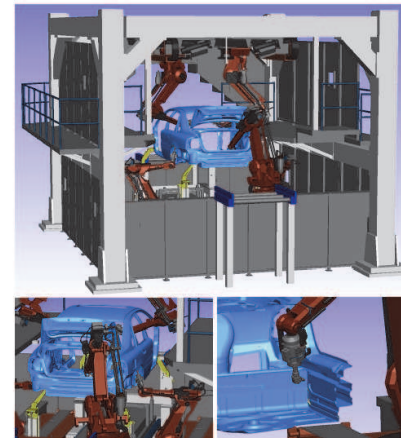
- ▶ Discrete puzzles, operations, and scheduling .
- ▶ An automotive assembly puzzle
- ▶ Sealing cracks in automotive assembly
- ▶ Making smart video game characters
- ▶ Virtual humans and humanoid robots
- ▶ Parking cars and trailers
- ▶ Flying Through the Air or in Space
- ▶ Designing better drugs



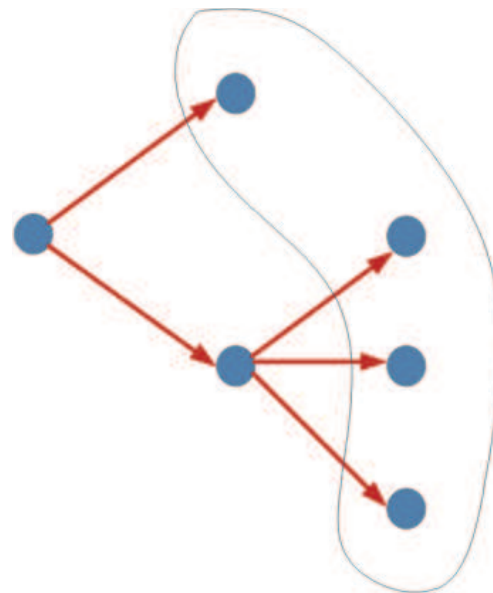
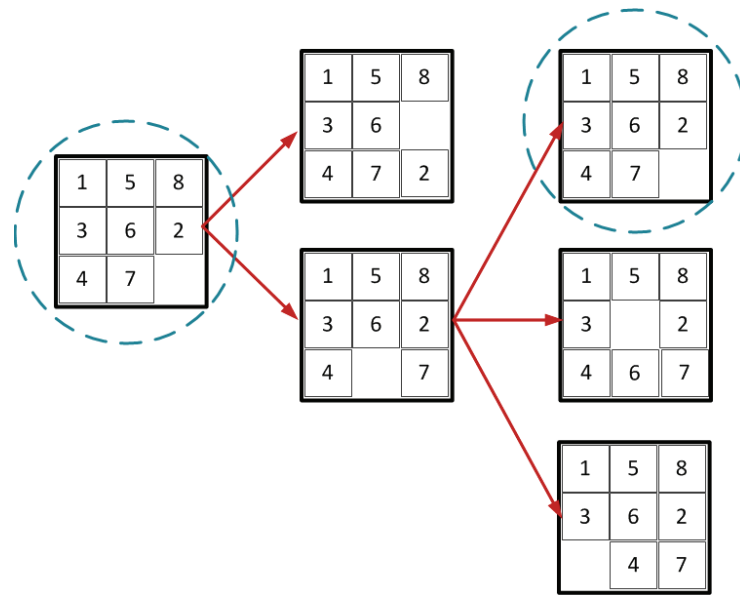
A motion computed by a planning algorithm, for a digital actor to reach into a refrigerator



A planning algorithm computes the motions of 100 digital actors moving across terrain with obstacles



An application of motion planning to the sealing process in automotive manufacturing



## Geometric Models

The world generally contains two kinds of entities:

- ▶ **Obstacles:** Portions of the world that are “permanently” occupied, for example, as in the walls of a building.
- ▶ **Robots:** Bodies that are modeled geometrically and are controllable via a motion plan.

Both obstacles and robots will be considered as (closed) subsets of  $\mathcal{W}$ . Let the obstacle region  $\mathcal{O}$  denote the set of all points in  $\mathcal{W}$  that lie in one or more obstacles; hence,  $\mathcal{O} \subset \mathcal{W}$ . The next step is to define a systematic way of representing  $\mathcal{O}$  that has great expressive power while being computationally efficient.



## Canonical Planning Problem

The assumptions in the canonical planning problem are

- ▶ The robot is the unique object that is moving (this excludes moving obstacles and other possible robots).
- ▶ The robot is a point moving in the space (this does not consider possible nonholonomy constraints).
- ▶ Obstacles positions and orientations are known (not true in unstructured environment).
- ▶ Obstacles must not be “touched” (this excludes robot-environment interaction, manipulation)

Such assumptions lead to a purely geometrical problem.

Representation issues:

- ▶ Can it be obtained automatically or with little processing?
- ▶ What is the complexity of the representation?
- ▶ Can collision queries be efficiently resolved?
- ▶ Can a solid or surface be easily inferred?

Idea: systematically constructing representations of obstacles and robots using a collection of primitives.

Both obstacles and robots will be considered as (closed) subsets of  $\mathcal{W}$ . Let the obstacle region  $\mathcal{O}$  denote the set of all points in  $\mathcal{W}$  that lie in one or more obstacles; hence,  $\mathcal{O} \subset \mathcal{W}$ . The next step is to define a systematic way of representing  $\mathcal{O}$  that has great expressive power while being computationally efficient.

Regarding obstacles, consider primitives of the form:

$$H = \{(x, y, z) \in \mathcal{W} \mid f(x, y, z) \leq 0\}$$

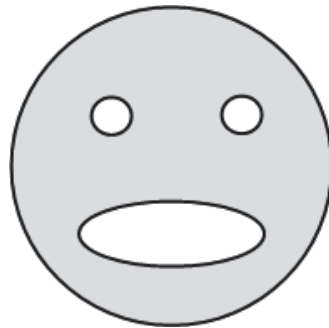
which is a half-space if  $f$  is linear.

Now let  $f$  be any polynomial, such as  $f(x, y) = x^2 + y^2 - 1$ . Obstacles can be formed from finite intersections or unions:

$$\mathcal{O} = H_1 \cap H_2 \cap \dots \cap H_n, \quad \mathcal{O} = H_1 \cup H_2 \cup \dots \cup H_n.$$

$\mathcal{O}$  could then become any semi-algebraic set.

Notions of inside and outside are clear and furthermore collision checking is performed in time that is linear in the number of primitives.

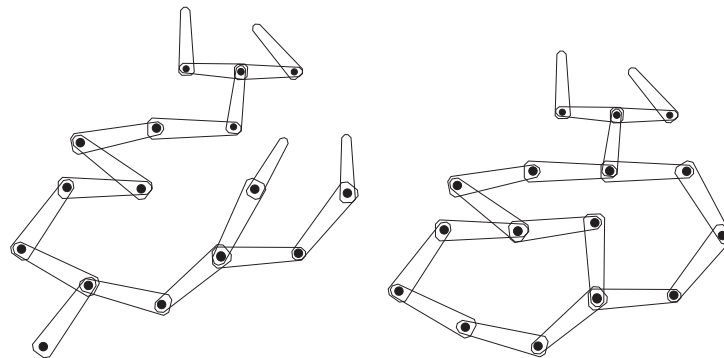


Regarding the robot if it is a rigid body, the robot transformations are 2D Rigid-body transformations:

$$T = \begin{pmatrix} \cos \theta & -\sin \theta & x_t \\ \sin \theta & \cos \theta & y_t \\ 0 & 0 & 1 \end{pmatrix}$$

3D Rigid-body transformations: RPY and translation

$$T = \begin{pmatrix} \cos \alpha \cos \beta & \cos \alpha \sin \beta \sin \gamma - \sin \alpha \cos \gamma & \cos \alpha \sin \beta \cos \gamma + \sin \alpha \sin \gamma & x_t \\ \sin \alpha \cos \beta & \sin \alpha \sin \beta \sin \gamma + \cos \alpha \cos \gamma & \sin \alpha \sin \beta \cos \gamma - \cos \alpha \sin \gamma & y_t \\ -\sin \beta & \cos \beta \sin \gamma & \cos \beta \cos \gamma & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



## The C-Space

The configurations space is the set of all possible transformations of the robot. It is an important abstraction that allows to use the same motion planning algorithm to problem that differ in geometry and kinematics.

- ▶ Path planning becomes a search on a space of transformations
- ▶ What does this space look like?
- ▶ How should it be represented?
- ▶ What alternative representations are allowed and how do they affect performance?

Three views of the configuration space:

- ▶ As a topological manifold
- ▶ As a metric space
- ▶ As a differentiable manifold

Number 3 is too complicated! There is no calculus in basic path planning

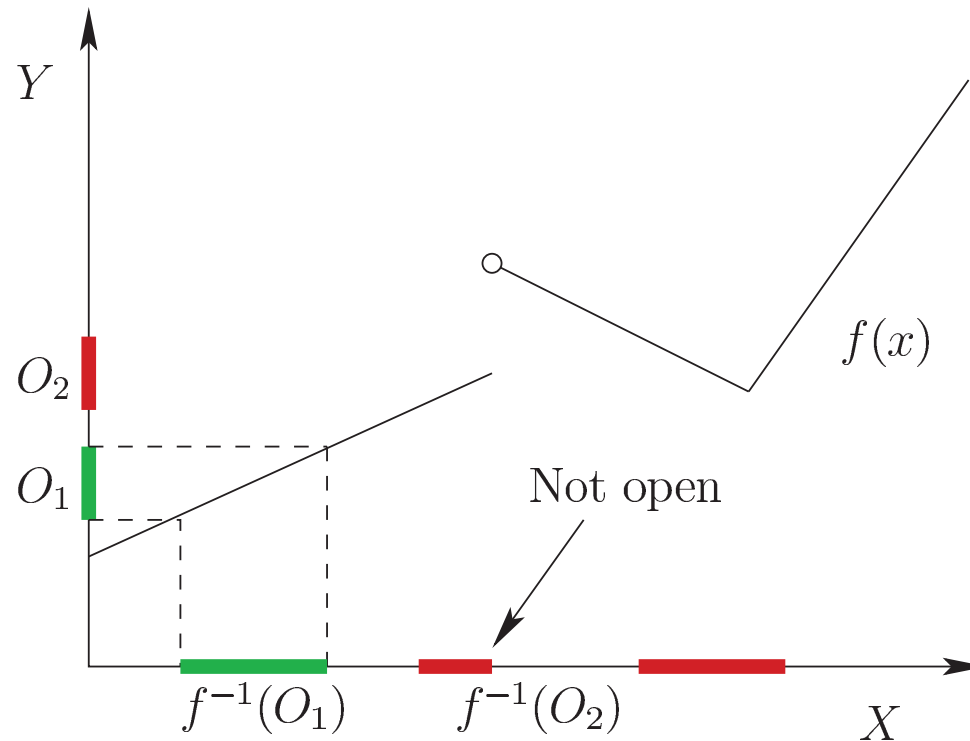
$\mathcal{X}$  topological set if

1. The union of any number of open sets is an open set.
2. The intersection of a finite number of open sets is an open set.
3. Both  $\mathcal{X}$  and  $\emptyset$  are open sets.

A set  $\mathcal{C} \subseteq \mathcal{X}$  is closed if and only if  $\mathcal{X} \setminus \mathcal{C}$  is open. Many subsets of  $\mathcal{X}$  could be neither open nor closed.

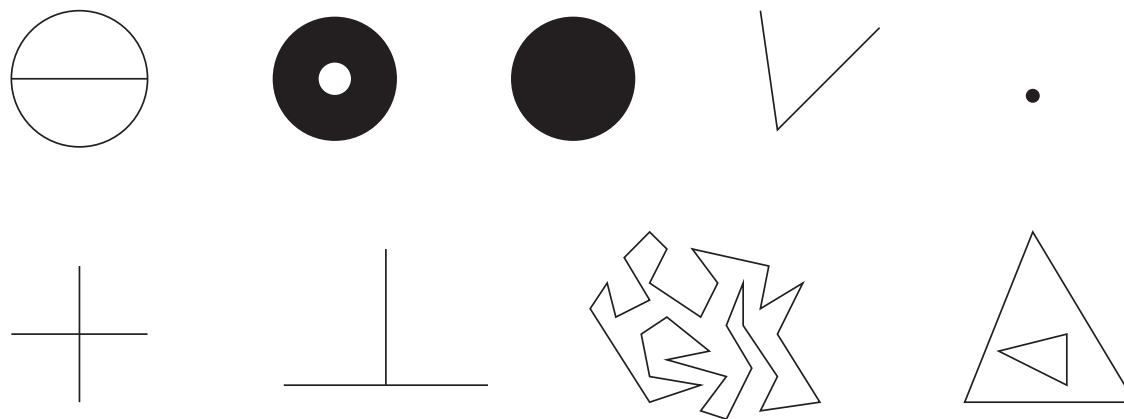
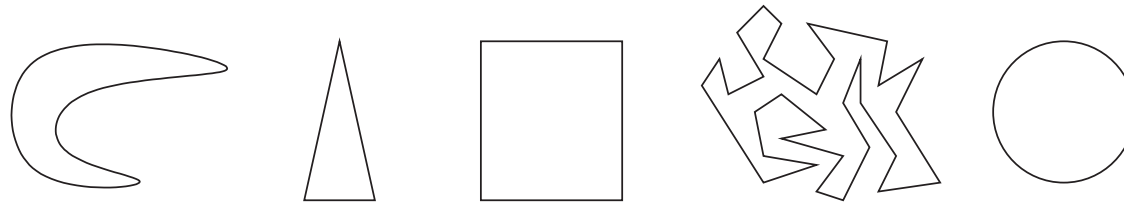
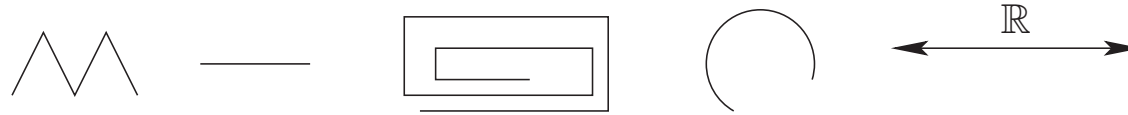
We will only consider spaces of the form  $\mathcal{C} \subseteq \mathbb{R}^n$ .  $\mathbb{R}^n$  comes equipped with standard open sets: A set  $\mathcal{O}$  is open if every  $x \in \mathcal{O}$  is contained in a ball that is contained in  $\mathcal{O}$ .

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be any topological spaces. A function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  is called continuous if for any open set  $O \subseteq \mathcal{Y}$ , the preimage  $f^{-1}(O) \subseteq \mathcal{X}$  is an open set.



A bijection  $f : \mathcal{X} \rightarrow \mathcal{Y}$  is called a homeomorphism if both  $f$  and  $f^{-1}$  are continuous. If such  $f$  exists, then  $\mathcal{X}$  and  $\mathcal{Y}$  are homeomorphic.

Homeomorphic and non homeomorphic subspaces of  $\mathbb{R}^2$ .





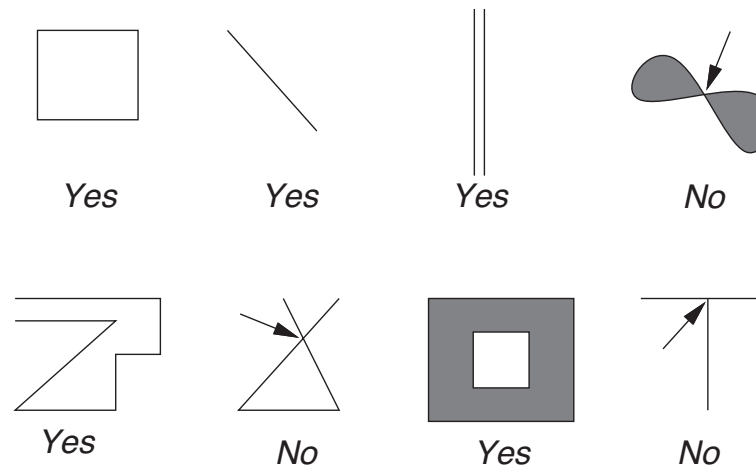
## Manifolds

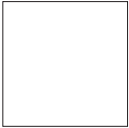
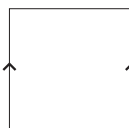
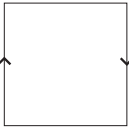
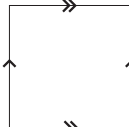
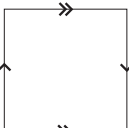
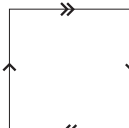
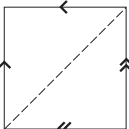
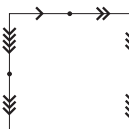
Let  $\mathcal{M} \subseteq \mathbb{R}^m$  be any set that becomes a topological space using the subset topology.

$\mathcal{M}$  is called a manifold if for every  $x \in \mathcal{M}$ , an open set  $O \subset \mathcal{M}$  exists such that

1.  $x \in O$ ;
2.  $O$  is homeomorphic to  $\mathbb{R}^n$ ;
3.  $n$  is fixed for all  $x \in \mathcal{M}$ .

It “feels like”  $\mathbb{R}^n$  around every  $x \in \mathcal{M}$ .



|   |   |
|---|---|
|  <p>Plane, <math>\mathbb{R}^2</math></p> |  <p>Cylinder, <math>\mathbb{R} \times S^1</math></p> |
|  <p>Möbius band</p>                      |  <p>Torus</p>  |
|  <p>Klein bottle</p>                     |  <p>Projective plane, <math>\mathbb{RP}^2</math></p> |
|  <p>Two-sphere, <math>S^2</math></p>   |  <p>Double torus</p>                               |

## Examples of configuration space

2D rigid body: Transformation is given by  $T_2 \in SE(2)$  hence in  $\mathbb{R}^9$ .

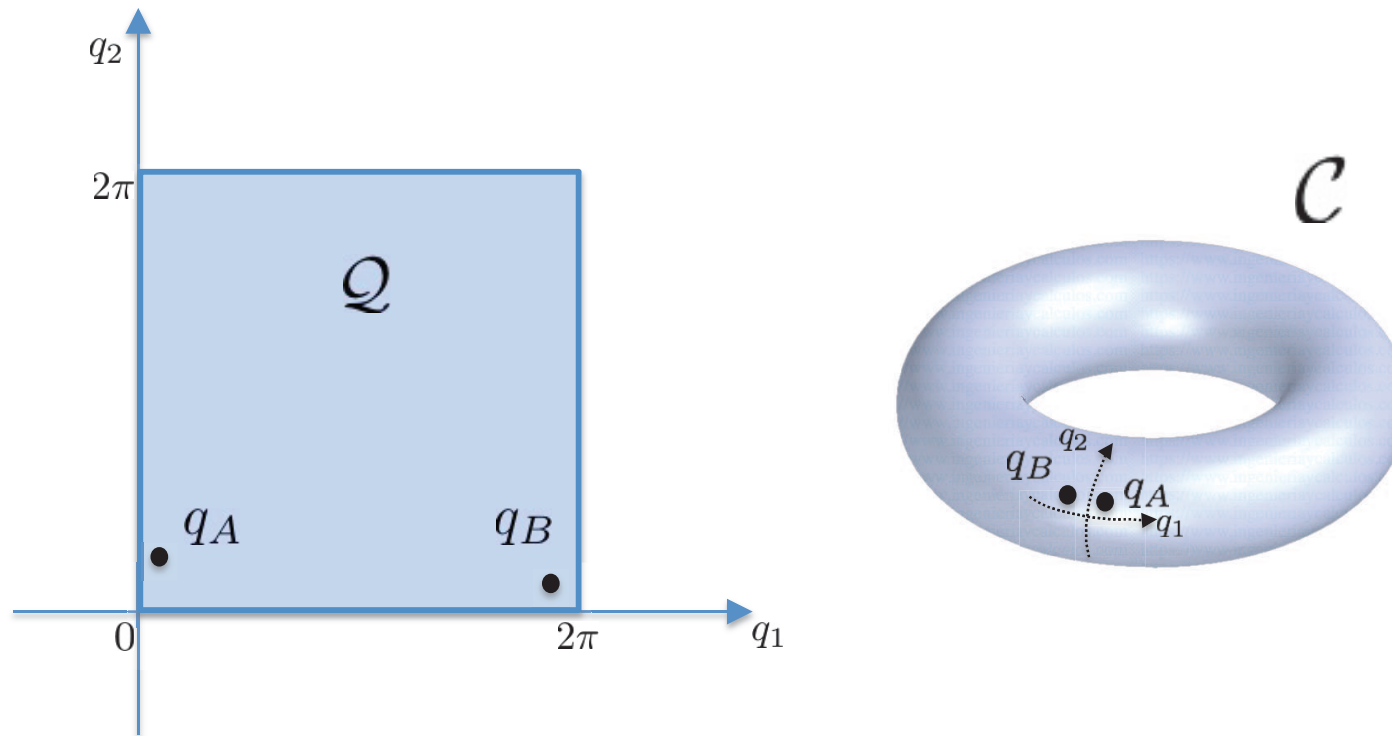
3D rigid body: Transformation is given by  $T_3 \in SE(3)$  hence in  $\mathbb{R}^{16}$ .

However rotations and translations can be chosen independently. Translations are in  $\mathbb{R}^n$  while rotation suffer of the periodicity of trigonometric functions, e.g. For 2D bodies  $\theta \in [0, 2\pi[$  and the set of rotations is  $\mathbb{S}^1$ .

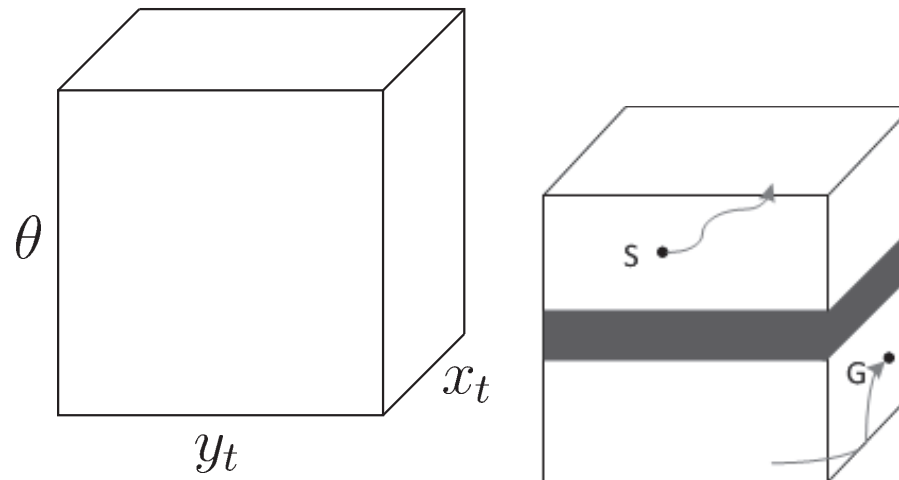
For a mobile polygonal robot in  $\mathcal{W} = \mathbb{R}^2$  the robot is described by the position of one of its points (e.g., COM, vertex) and the polygon orientation with respect to a fixed frame. Hence  $\mathcal{C} = \mathbb{R}^2 \times SO(2)$  that has dimension 3.

For a mobile polygonal robot in  $\mathcal{W} = \mathbb{R}^3$  the configuration space is  $\mathcal{C} = \mathbb{R}^3 \times SO(3)$  that has dimension 6.

For a planar manipulator with (fixed base) two links and two rotational joints the configuration space is a subset of  $(\mathbb{R}^2 \times SO(2))^2$ . The dimension of the configuration space is  $3n - 2n = n$  since each joint constrains the motion of the following link (2 constraints per joint). Configuration variables commonly used are  $q = (q_1, q_2)$  with  $q_1, q_2 \in [0, 2\pi)$ . However, such representation is valid only locally ( $q_1$  and  $q_2$  close in  $W$  far apart in  $\mathcal{Q}$ ). Hence  $\mathcal{Q}$  should be represented by  $SO(2) \times SO(2)$ .



C-space for a unicycle with configurations  $(x, y, \theta)$ .



## C-Space Obstacles

Given world  $\mathcal{W}$ , a closed obstacle region  $\mathcal{O} \subset \mathcal{W}$ , closed robot  $\mathcal{A}$ , and configuration space  $\mathcal{C}$ . Let  $\mathcal{A}(q) \subset \mathcal{W}$  denote the placement of the robot into configuration  $q$ . The obstacle region  $\mathcal{C}_{obs}$  in  $\mathcal{C}$  is

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\},$$

which is a closed set.

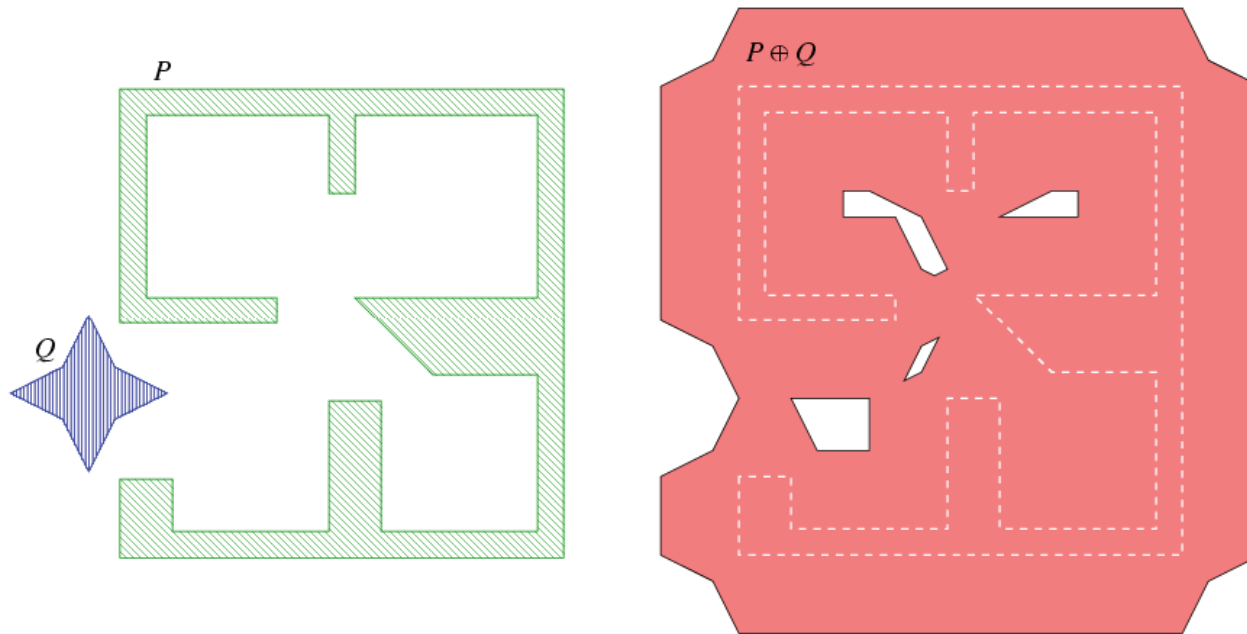
The free space  $\mathcal{C}_{free}$  is an open subset of  $\mathcal{C}$ :

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$$

We want to keep the configuration in  $\mathcal{C}_{free}$  at all times!

Consider  $\mathcal{C}_{obs}$  for the case of translation only. The Minkowski sum of two sets is defined as

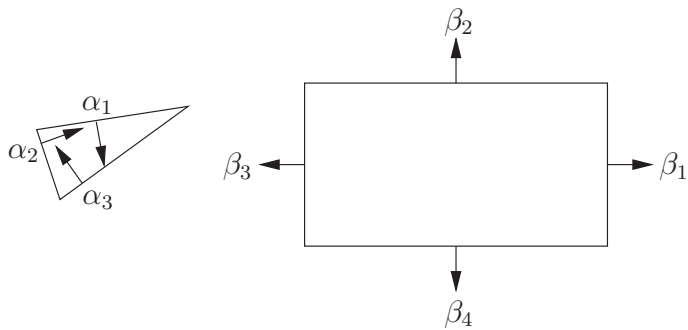
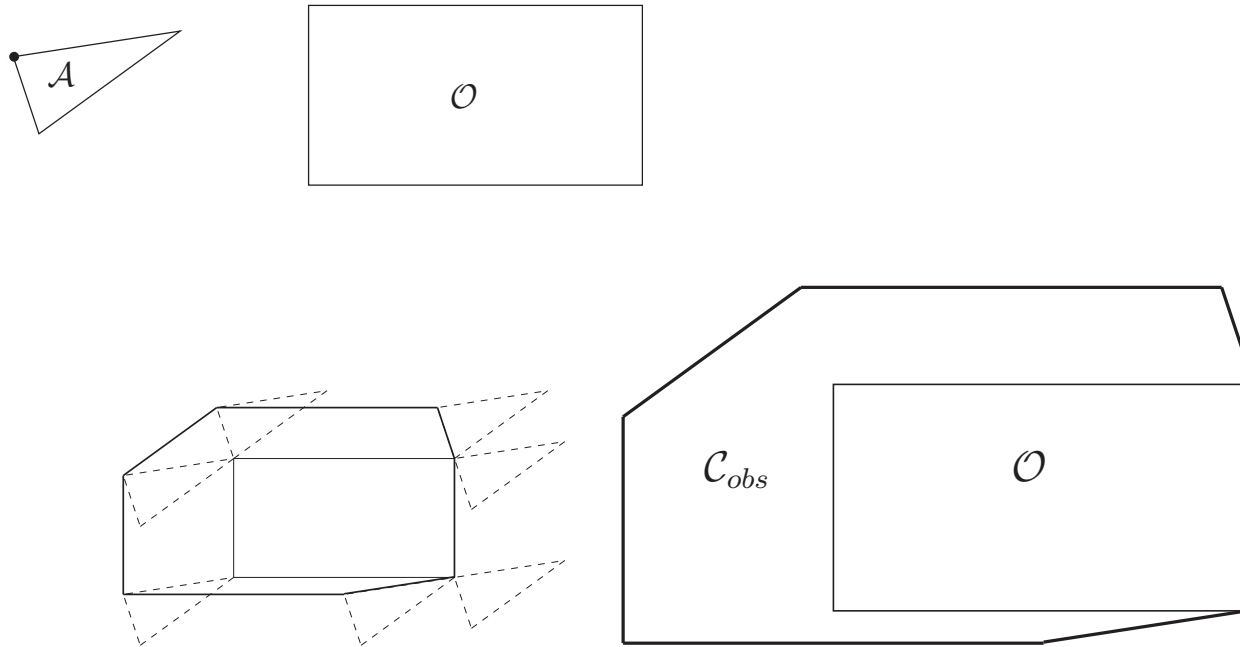
$$X \oplus Y = \{x + y \in \mathbb{R}^n \mid x \in X \text{ and } y \in Y\}$$



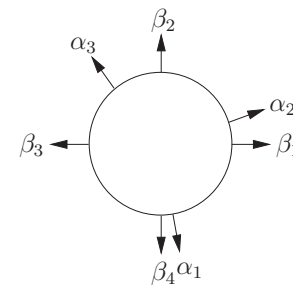
The Minkowski difference of two sets is defined as

$$X \ominus Y = \{x - y \in \mathbb{R}^n \mid x \in X \text{ and } y \in Y\}$$

2D translation only  $\mathcal{C}_{obs}$



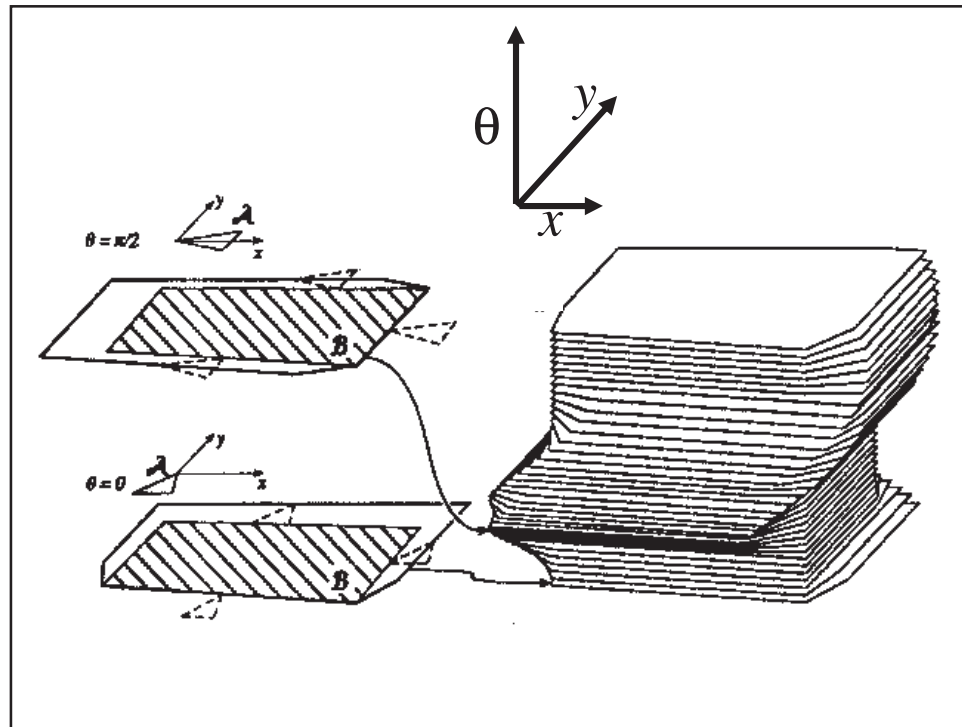
Inward and outward normals



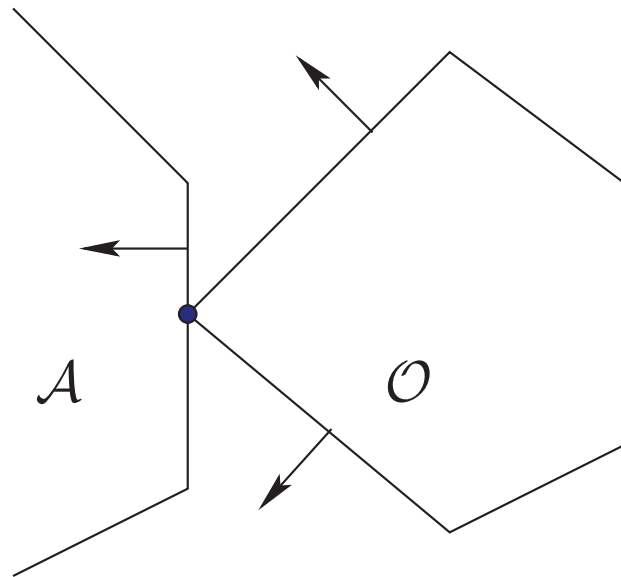
Sorted around  $S^1$



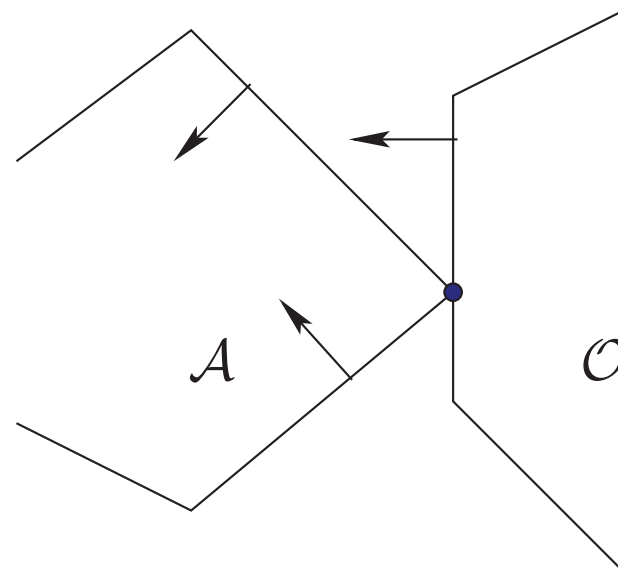
2D translation and rotation  $\mathcal{C}_{obs} \Rightarrow$  3D subset of  $\mathbb{R}^2 \times \mathbb{S}^1$ .



2D contact types:



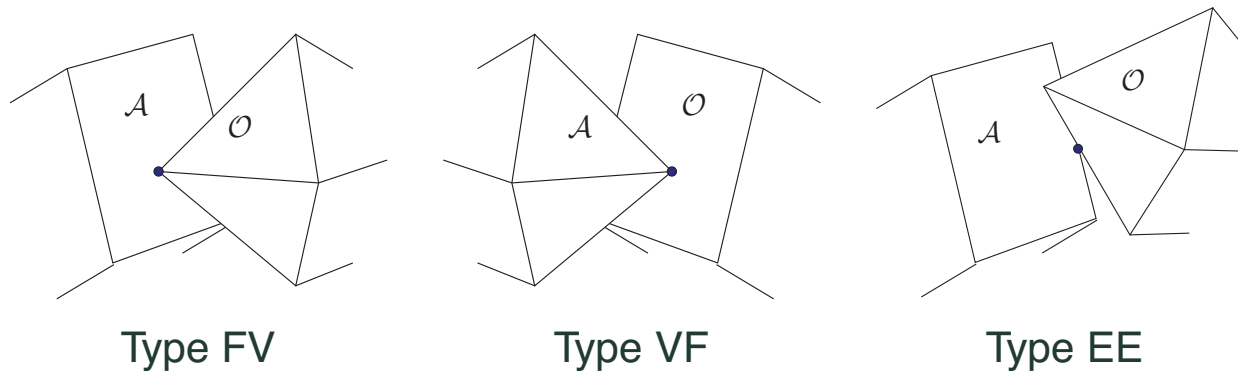
Type EV



Type VE

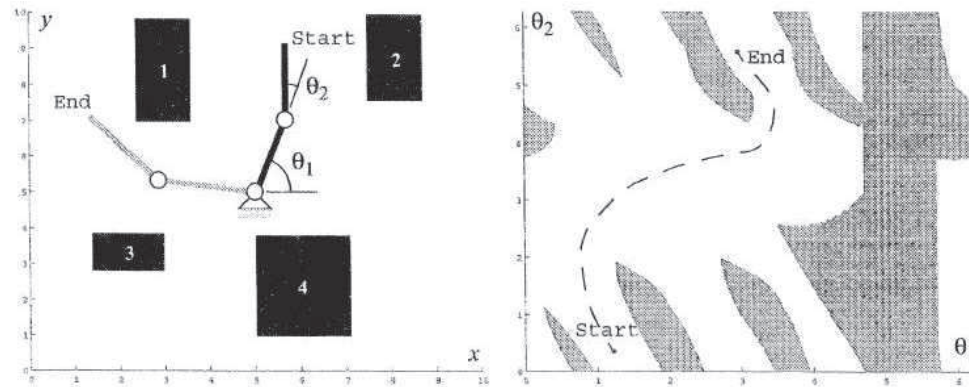
Equation polynomials in  $x_t, y_t, \cos \theta, \sin \theta$  arise.

3D contact types:



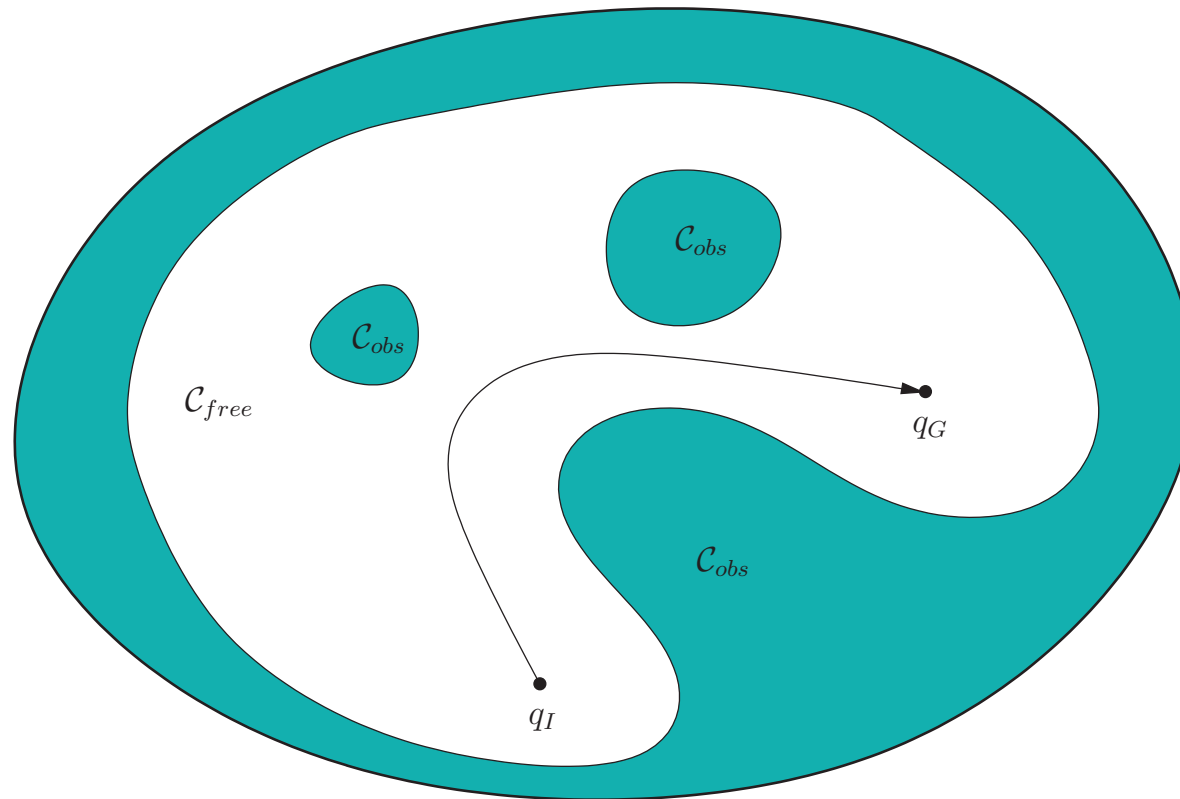
Equation polynomials in  $x_t, y_t, \cos \theta, \sin \theta$  arise.

For two-links  $C = \mathbb{S}^1 \times \mathbb{S}^1$



## Basic motion planning problem

Given robot  $\mathcal{A}$  and obstacle  $\mathcal{O}$  models, C-space  $\mathcal{C}$ , and  $q_I, q_G \in \mathcal{C}_{free}$ .



Automatically compute a path  $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$  so that  $\tau(0) = q_I$  and  $\tau(1) = q_G$ .

There are three main philosophies for addressing the motion planning problem:

- ▶ Combinatorial (or roadmap) planning (exact planning)
- ▶ Sampling-based planning (probabilistic planning, randomized planning)
- ▶ Artificial potential fields methods

All work in the configuration space and most of the methods require preliminary (expensive!) computation of  $\mathcal{C}_{obs}$  and  $\mathcal{C}_{free}$ . The computation of  $\mathcal{C}_{obs}$  can be **exact** (algebraic model needed) or **approximate** (grid decomposition).

A planning algorithm may be:

- ▶ **Complete:** If a solution exists, it finds one; otherwise, it reports failure.
- ▶ **Semi-complete:** If a solution exists, it finds one; otherwise, it may run forever.
- ▶ **Resolution complete:** If a solution exists, it finds one; otherwise, it terminates and reports that no solution within a specified resolution exists.
- ▶ **Probabilistically complete:** If a solution exists, the probability that it will be found tends to one as the number of iterations tends to infinity.

First for Combinatorial approach, the others for the sampling approach

## Combinatorial Motion Planning ( $\sim$ 1980)

There are generally two good reasons to study combinatorial approaches to motion planning:

1. For many special classes, elegant and efficient algorithms can be developed. These algorithms are complete, do not depend on approximation, and can offer much better performance than other planning methods.
2. It is both interesting and satisfying to know that there are complete algorithms for an extremely broad class of motion planning problems. Thus, even if the class of interest does not have some special limiting assumptions, there still exist general-purpose tools and algorithms that can solve it. These algorithms also provide theoretical upper bounds on the time needed to solve motion planning problems.

Combinatorial approaches construct a finite data structure (named **Roadmap**) that exactly encodes the planning problem.

Some of the algorithms first construct a cell decomposition of  $\mathcal{C}_{free}$  from which the roadmap is consequently derived (**cell decomposition methods**). Other methods directly construct a roadmap without the consideration of cells (**maximum clearance roadmap**).

Both methods typologies produce a **topological graph**  $G$ :

- Each vertex is a configuration  $q \in \mathcal{C}_{free}$
- Each edge is a path  $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$  for which  $\tau(0)$  and  $\tau(1)$  are vertices.



A roadmap is a topological graph  $G$  with two properties:

1. **Accessibility:** From anywhere in  $\mathcal{C}_{free}$  it is trivial to compute a path that reaches at least one point along any edge in  $G$ .
2. **Connectivity-preserving:** If there exists a path through  $\mathcal{C}_{free}$  from  $q_I$  to  $q_G$ , then there must also exist one that travels through  $G$ .

We consider polygonal obstacle regions where clever data structures can be used to encode vertices, edges, regions (e.g. Doubly connected edge list)

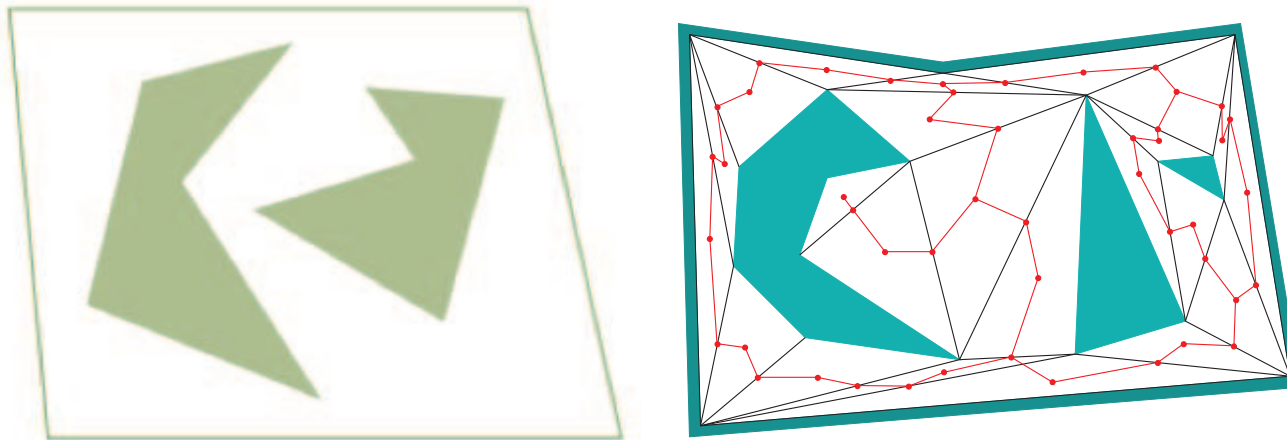


Figura 16: Left: Polygonal obstacles, Right: Example of roadmap

## Cell Decomposition Methods

Given  $\mathcal{C}_{free}$  cells are obtained with the following properties:

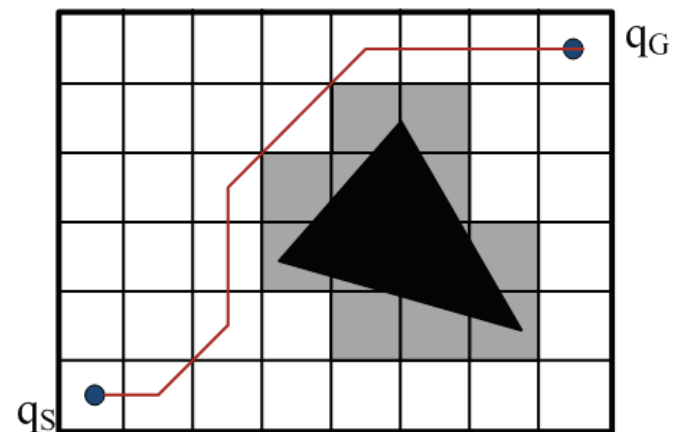
- ▶ Planning in a cell is trivial
- ▶ Adjacency information can be extracted easily to build the roadmap
- ▶ Exists an efficient way to determine the cells containing  $q_I$  and  $q_G$ .

Once the roadmap has been constructed a graph path computation is performed (see chapter on Graph Optimization).

Cell decomposition methods are divided in Approximate and Exact Cell Decomposition.

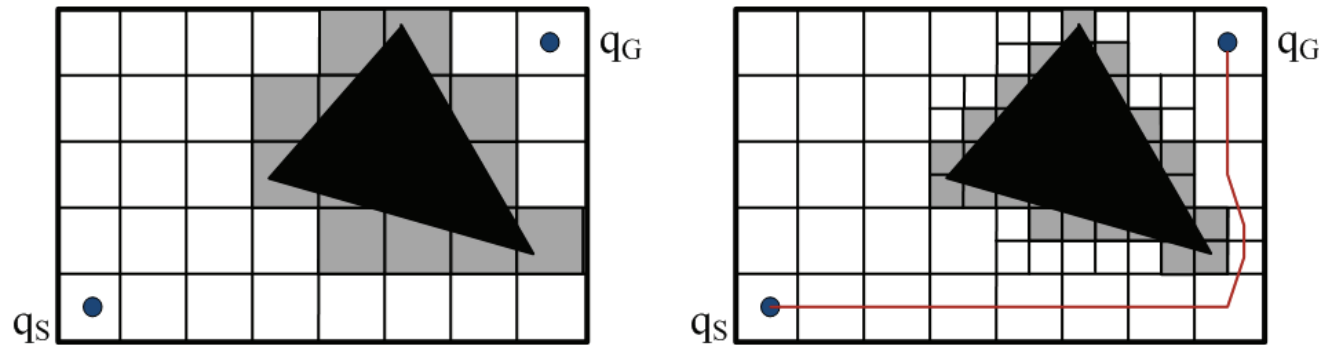
## Approximate Cell Decomposition

Idea: Decompose the space into cells with predefined shapes so that any path inside a cell is obstacle free. The union of such cells is a lower approximation of  $\mathcal{C}_{free}$ .



Define a discrete grid in C-Space and mark any cell of the grid that intersects  $\mathcal{C}_{obs}$  as blocked. Find a path through remaining cells by using algorithms for discrete optimization.

As described, the method can be incomplete:



Indeed, it cannot find a path, in the case in figure, even though one path exists.

A possible solution is to distinguish between

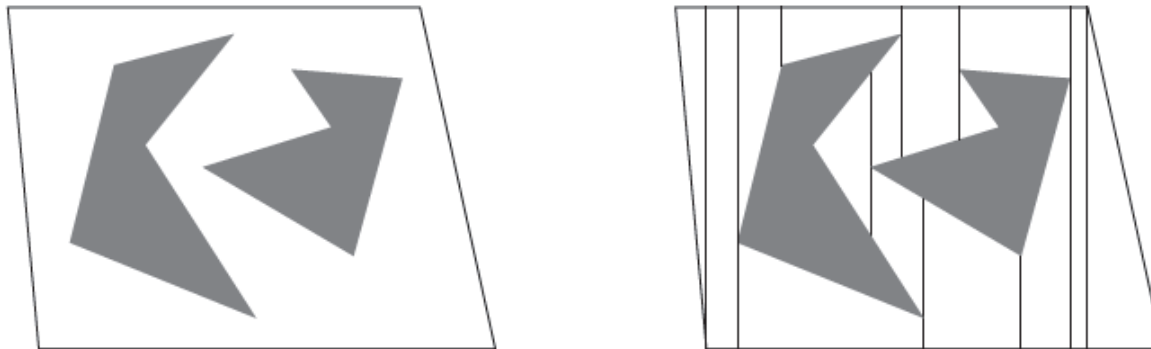
- Cells that are entirely contained in  $\mathcal{C}_{obs}$  (FULL) and
- Cells that partially intersect  $\mathcal{C}_{obs}$  (MIXED)

Try to find a path using the current set of cells. If no path is found subdivide the MIXED cells and try again with the new set of cells.

## Exact Cell Decomposition

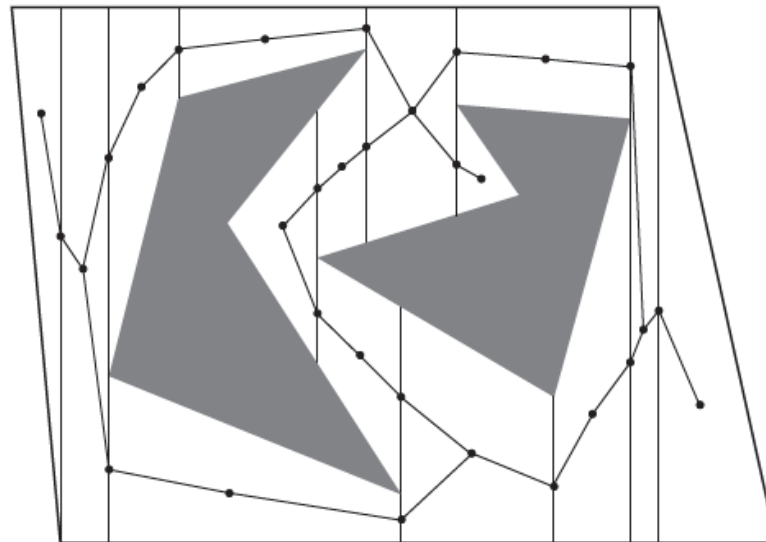
Idea: Any straight path within one cell is guaranteed to not intersect any obstacle. The difference with respect to approximate cell decomposition is that the union of cell is exactly  $\mathcal{C}_{free}$ .

### Trapezoidal decomposition



The construction is based on the *sweep vertical line* method that corresponds to a method to span the entire space with a vertical line and select lines that pass through vertex of obstacles or environment. Then sample point for each cell and for segments of the cells borders are considered as nodes of the graph, straight segments connecting nodes of cells with the nodes of the cells borders are the edges.

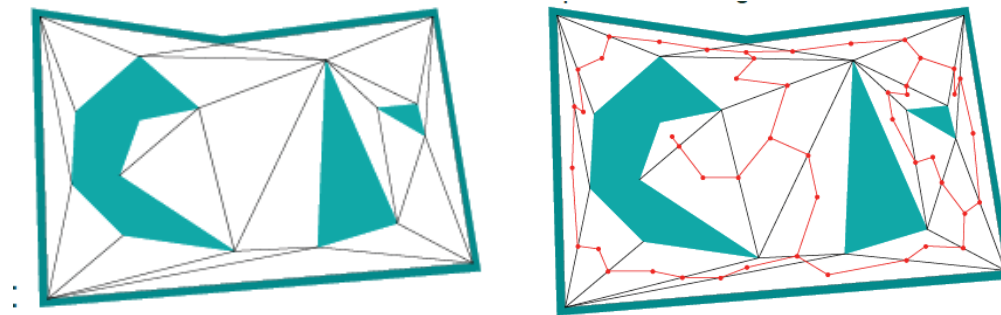
The resulting roadmap  $G$ :



## Triangulation decomposition

Another method is to triangulate  $\mathcal{C}_{free}$  by connecting vertices of obstacles with vertices of other obstacles or vertices of the environment that are in line of sight. Center of triangles and midpoints of triangle edges are the nodes of the roadmap. The straight arcs from the centers to the borders are the edges of the roadmap.

Compute triangulation:  $O(n^2)$  time naive,  $O(n)$  optimal,  $O(n \log n)$  a good tradeoff. Build easy roadmap from the triangulation:



More complex decompositions may be considered. Some are more suitable for generalization to higher dimensional spaces.

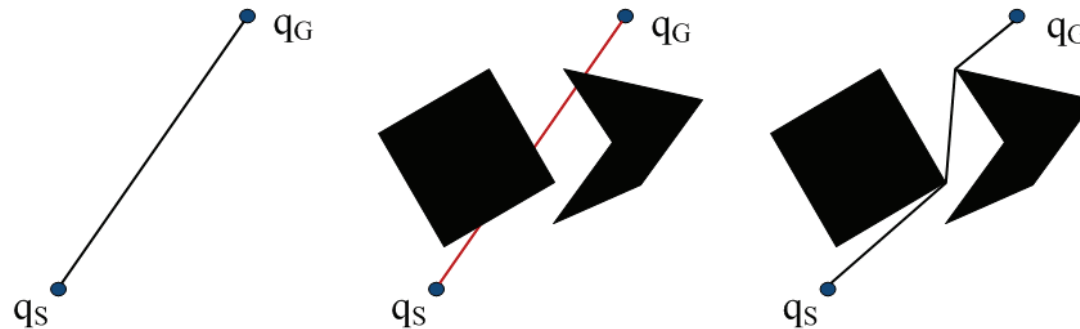
## Direct roadmap construction

In case of requests such as the computation of minimum length paths or of paths that maximize the distance from obstacles, the roadmap can be constructed without subdividing the configuration space into cells.

In case of shortest paths requirements the **shortest-path roadmaps** can be constructed based on the Visibility graph concept.

### Visibility Graphs

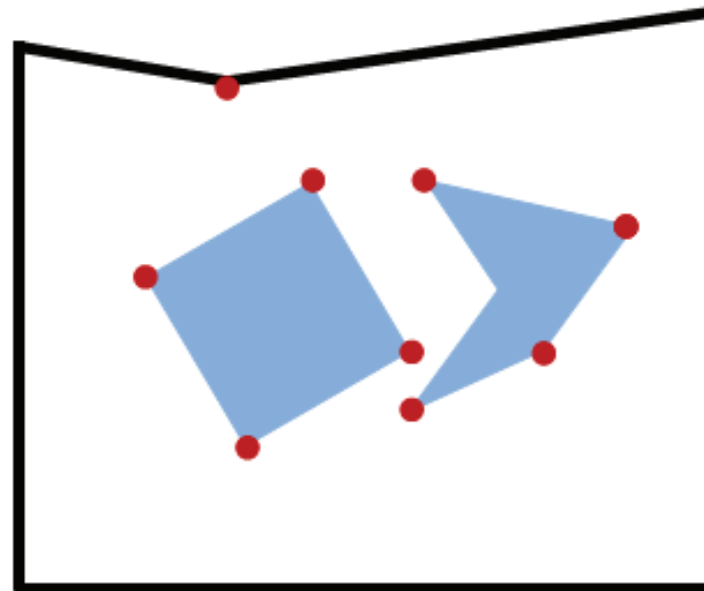
In case of obstacles absence, the shortest path from  $q_S$  to  $q_G$  is the straight line. On the other hand, in presence of obstacles, it is a sequence of straight lines.



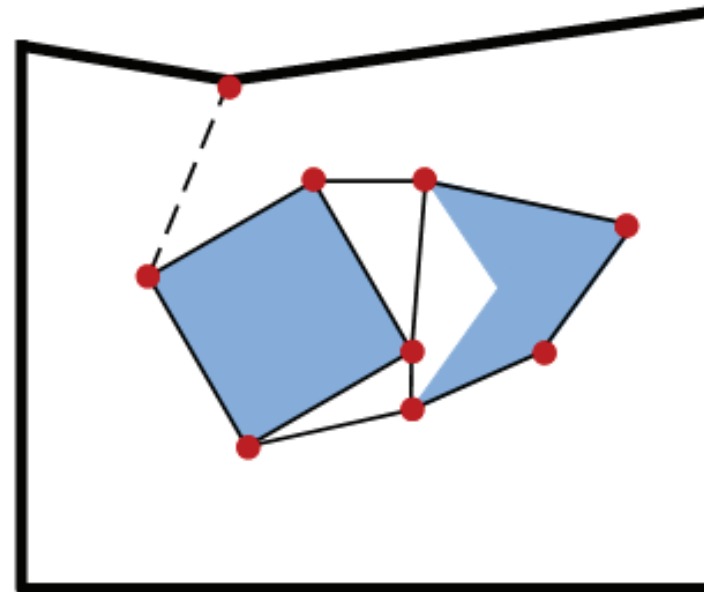


The path is obtained in the closure of  $\mathcal{C}_{free}$  and hence the robot can touch the obstacles.

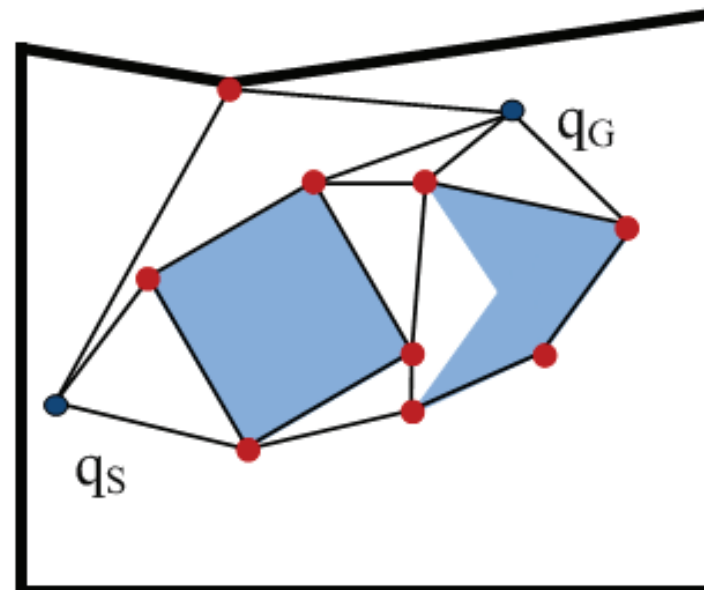
We consider as nodes in the roadmap the **reflex vertices** that are vertices of the polygons for which the interior angle (in  $\mathcal{C}_{free}$ ) is greater than  $\pi$



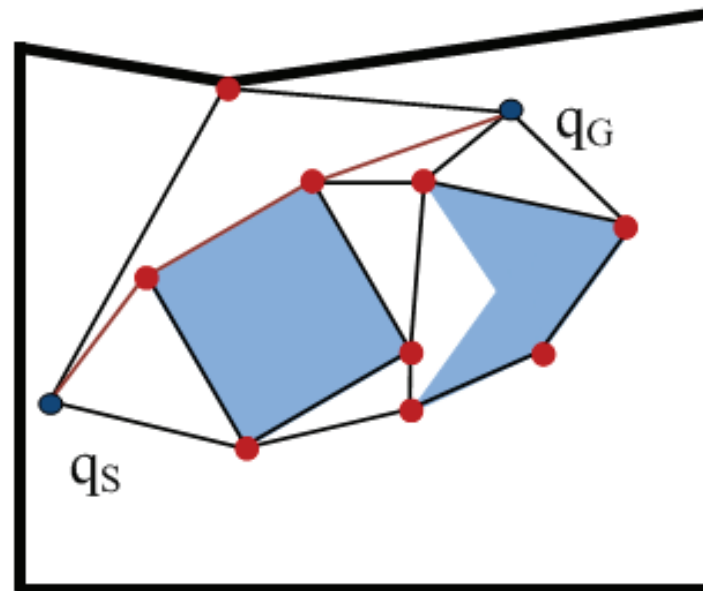
Edges of the roadmap are the arcs between two consecutive reflex vertices and the arcs corresponding to bitangent lines between (mutually visible) reflex vertices



To solve a query the initial and final configurations  $q_S$  and  $q_G$  are connected to visible vertices



The optimal path is then computed on the roadmap

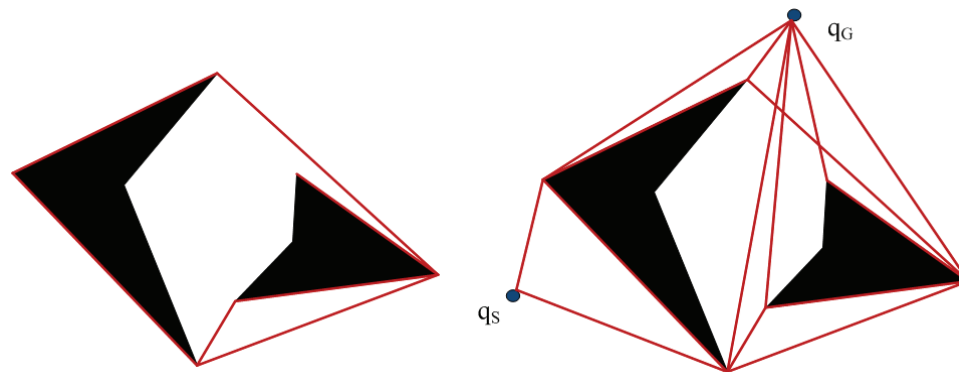


Known also as *reduced visibility graph*

The shortest-path roadmap,  $G$ , is constructed as follows: a reflex vertex is a polygon vertex for which the interior angle (in  $\mathcal{C}_{free}$ ) is greater than  $\pi$ . The vertices of  $G$  are the reflex vertices. Edges of  $G$  are formed from two different sources:

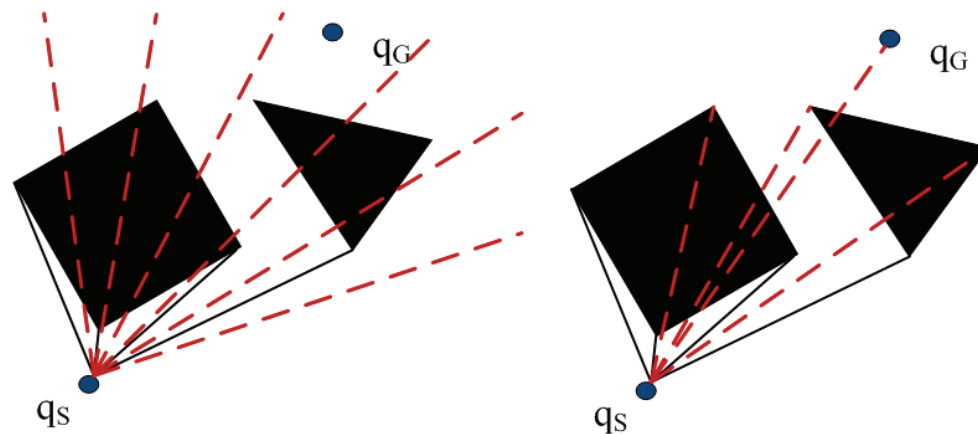
*Consecutive reflex vertices:* If two reflex vertices are the endpoints of an edge of  $\mathcal{C}_{obs}$ , then an edge between them is made in  $G$ .

*Bitangent edges:* If a bitangent line can be drawn through a pair of reflex vertices, then a corresponding edge is made in  $G$ . A bitangent line, is a line that is incident to two reflex vertices and does not poke into the interior of  $\mathcal{C}_{obs}$  at any of these vertices. Furthermore, these vertices must be mutually visible from each other.



We obtain shortest path but with this choice we try to stay as close as possible to obstacles. Hence, any error in the execution of the path may lead to a collision. Moreover, the approach is too complicated in higher dimensional spaces.

From the implementation point of view a *radial sweep* can be performed for the edges computation.



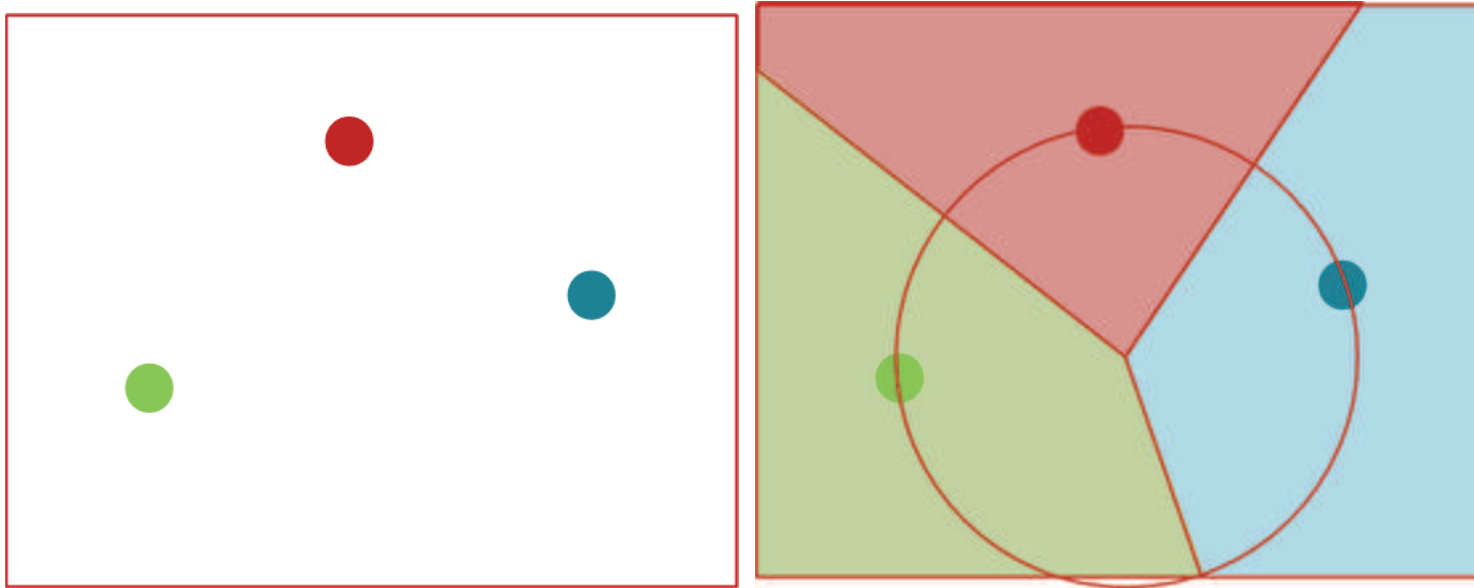
## Maximum-clearance roadmaps

We may be interested in moving along not-shortest paths but along paths that are far from obstacles. This can be done through the construction of the *Maximum-clearance roadmaps* that try to keep the robot as far as possible from obstacles and environment limits. Other names: *generalized Voronoi diagrams* or *retraction method*.

Peculiarity: each point of the roadmap (vertices and points on edges) are equidistant from two points of the boundary of the environment or of obstacles.

*Retraction*: the set  $S$  of points on the roadmap is the *deformation retract* of  $X = \mathcal{C}_{free}$  (in case of topological spaces): let  $h : X \times [0, 1] \rightarrow X$  such that

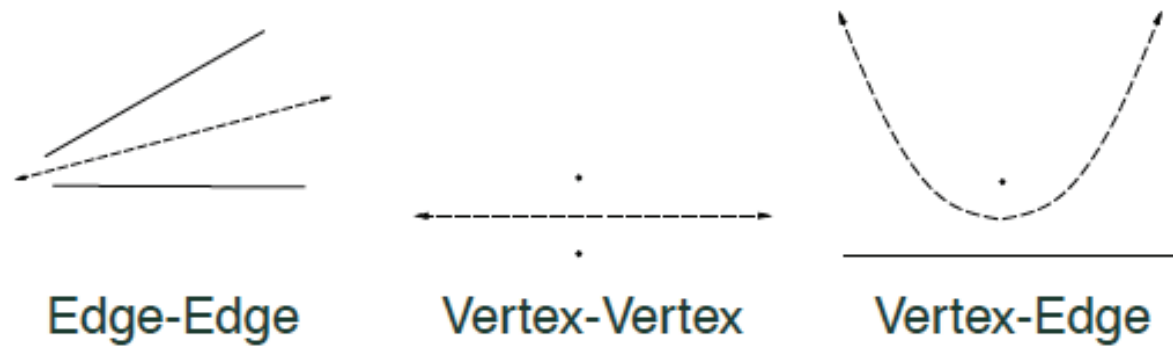
- ▶  $h(x, 0) = x \forall x \in X$ ,
- ▶  $h(x, 1) = g(x) \forall x \in X$  where  $g : X \rightarrow S$  is continuous,
- ▶  $h(s, t) = s \forall s \in S, t \in [0, 1]$ .



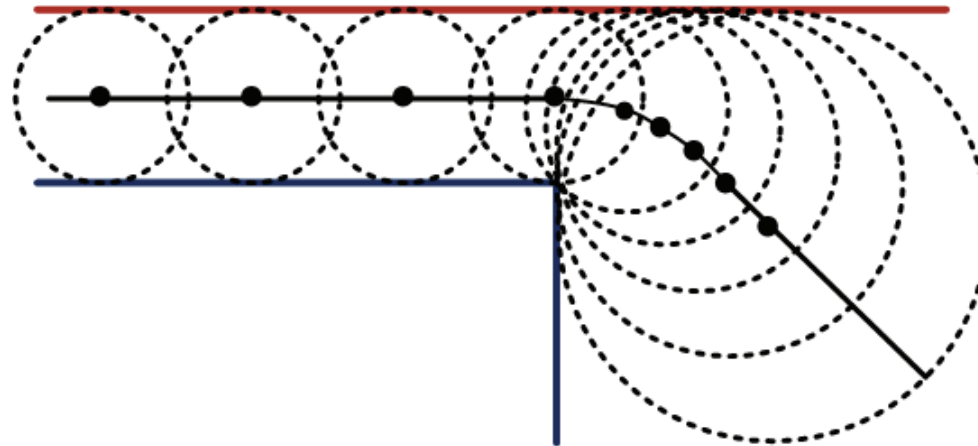
Voronoi Diagram: The set of line segments separating the regions corresponding to different colors.

- ▶ Line segment: points equidistant from 2 data points;
- ▶ Vertices: points equidistant from more than 2 data points.

<http://www.cs.cornell.edu/Info/People/chew/Delaunay.html>



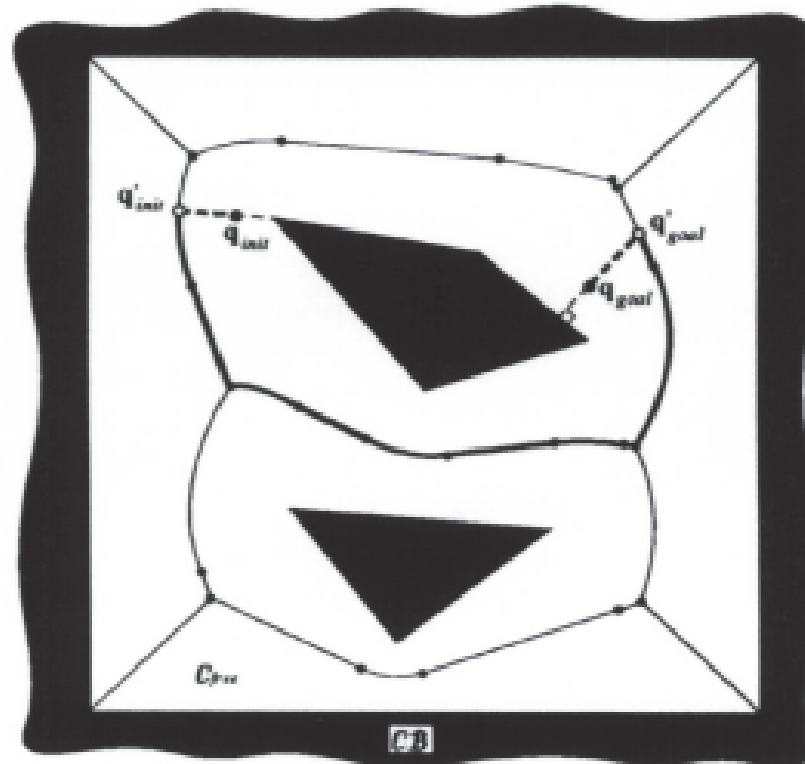
In case of polygonal  $C_{obs}$  and environment edges are straight line segments or arcs of quadratic curves.



Straight edges: Points equidistant from 2 lines

Curved edges: Points equidistant from one corner and one line





## Sample-Based Motion Planning

Idea: Avoid explicit construction of  $\mathcal{C}_{obs}$ , conduct a search that probes the  $\mathcal{C}$  space with a sampling scheme with a collision detection module as a “black box”.

With this approach planning algorithms are independent of the particular geometric models.

### Distances in $\mathcal{C}$ -space

$(\mathcal{X}, \rho)$  is a metric space if  $\mathcal{X}$  is a topological space and  $\rho : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  is such that  $\forall a, b, c \in \mathcal{X}$

1.  $\rho(a, b) \geq 0$  non-negativity,
2.  $\rho(a, b) = 0 \Leftrightarrow a = b$  reflexivity,
3.  $\rho(a, b) = \rho(b, a)$  symmetry,
4.  $\rho(a, b) + \rho(b, c) \geq \rho(a, c)$  triangle inequality

$L_p$  metrics in  $\mathbb{R}^n$ :  $\rho(x, y) = (\sum_{i=1}^n |x_i - y_i|^p)^{\frac{1}{p}}$ .

$L_2$  Euclidean metric,  $L_1$  Manhattan metric (in  $\mathbb{R}^2$  length of a path moving along axis-aligned grid),  $L_\infty(x, y) = \max_{i=1, \dots, n} |x_i - y_i|$ .

$L_p$  norms induced by the  $L_p$  metrics:

$$\|x\|_p = (\sum_{i=1}^n \|x_i\|^p)^{\frac{1}{p}}$$

With the norms:  $\rho(x, y) = \|x - y\|$  that is convenient for computations.

## Sampling Theory

The  $\mathcal{C}$ -space is uncountably infinite while there is a countable number of samples.

The performance of algorithms based on sampling depends on how the  $\mathcal{C}$ -space is sampled and on the sequence with which samples are chosen (the algorithm will end after a finite number of samples).

The gap between the infinite sampling sequence and the uncountable  $\mathcal{C}$ -space leads to the concept of denseness.

Given  $U, V \subset \mathcal{X}$  topological spaces,  $U$  is *dense in*  $V$  if the closure of  $U$  is  $V$ , i.e.  $cl(U) = U \cup \partial U = V$ .

*For example*  $(0, 1)$  is dense in  $[0, 1]$ ,  $\mathbb{Q}$  is dense in  $\mathbb{R}$ .

The goal is to have a dense sequence of samples in  $\mathcal{C}$ -space, but dense in probability!

For example, consider  $\mathcal{C} = [0, 1]$ , let  $I = [a, b] \subset \mathcal{C}$  with  $b - a = l$ . Consider a sequence of  $k$  independent random samples, the probability that no one of the samples falls into  $I$  is  $p = (1 - l)^k$ . When the number of samples tends to infinity the probability  $p$  tends to 0. Hence, the probability that any nonzero length interval contains no point converges to zero. In other words, the infinite sequence of samples is dense in  $\mathcal{C}$  with probability 1.

## Random Samples

The goal is to generate uniform random samples, i.e. to determine a uniform probability density function on the  $\mathcal{C}$ -space. Random sampling is the easiest of all sampling methods for the  $\mathcal{C}$ -space because it often consists of Cartesian product.

If a uniform sample is taken from  $X_1$  and  $X_2$  the uniformity is obtained also for  $X_1 \times X_2$ . Hence, for 5 robots with translational movements in  $[0, 1]^2$  we have  $\mathcal{C} = [0, 1]^{10}$ . Given 10 points uniformly at random from  $[0, 1]$  we may rearrange them in a 10D vector obtaining a uniform distribution over  $\mathcal{C}$ .

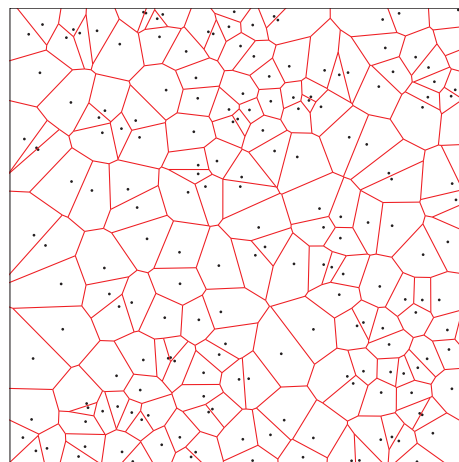
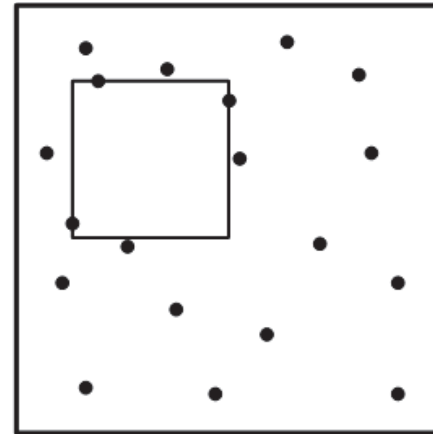
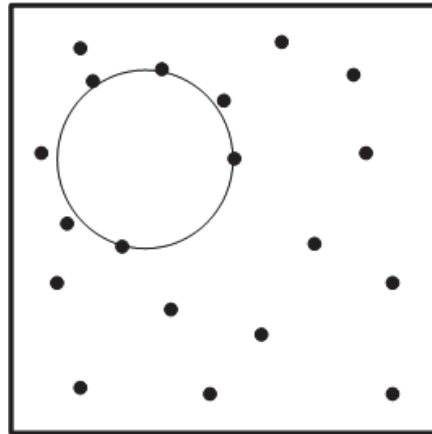
## Low-dispersion Sampling

In case of a grid, the resolution can be increased by decreasing the step size of each axis. A possible extension of this concept is the criterion of *dispersion*:

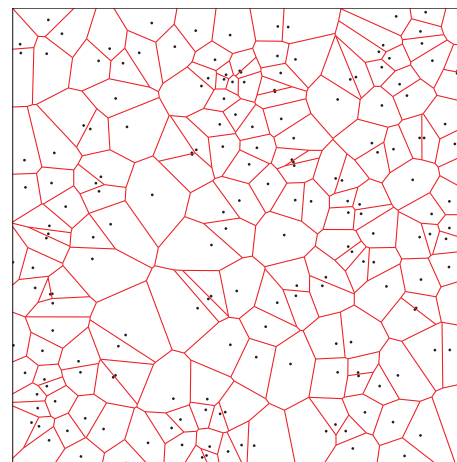
**Definition 1.** *In a metric space  $(\mathcal{X}, \rho)$  the dispersion of a finite set  $P$  of samples is*

$$\delta(P) = \sup_{x \in \mathcal{X}} \{ \min_{p \in P} \{ \rho(x, p) \} \}$$

$L_2$  and  $L_\infty$  dispersions:



(a) 196 pseudorandom samples



(b) 196 pseudorandom samples

## Collision Detection

Once samples have been obtained it is necessary to check if a configuration is in collision. Hence, a collision detection algorithm is crucial since it will also take the largest amount of time in the planning algorithm. Even though it is often treated as a black box, it is important to study its inner workings to understand the information it provides and its associated computational cost.

For 2D convex robots and convex obstacles linear-time collision detection algorithm can be determined.

Whenever we are able to determine a model of  $\mathcal{C}_{obs}$  we can consider a *logical predicate*  $\phi : \mathcal{C} \rightarrow T, F$  with  $T = true$  and  $F = false$ , where  $q \in \mathcal{C}_{obs} \Rightarrow \phi(q) = T$  and  $q \notin \mathcal{C}_{obs} \Rightarrow \phi(q) = F$ . The logical predicate may be easily implemented based on the available model. However, it is not sufficient in some cases, for example the logical predicate is a boolean function and it does not provide any information on how far the robot is from the obstacle.



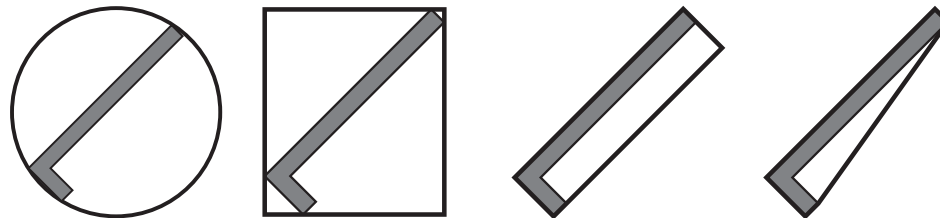
In this case, a *distance function* is preferred. Let  $d : \mathcal{C} \rightarrow [0, +\infty)$  be the distance between  $q$  and the closest point in  $\mathcal{O}$ .

For  $E, F$  closed sets of  $\mathbb{R}^n$   $\rho(E, F) = \min_{e \in E, f \in F} \|e - f\|$ .

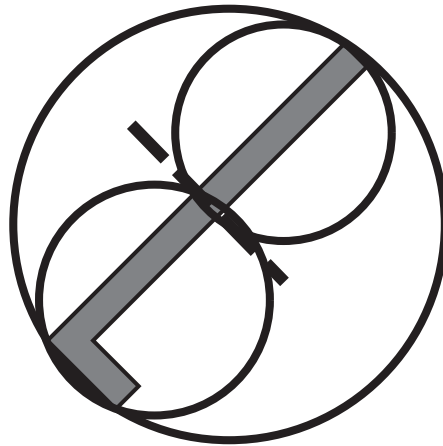
Consider the case of a robot with  $m$  links  $\mathcal{A}_1, \dots, \mathcal{A}_m$  and an obstacle set  $\mathcal{O}$  with  $k$  connected components. The detection of collisions is difficult and can be faced with a two-phase approach. The broad phase: the idea is to avoid computation for bodies that are far away from each other. A bounding-box can be placed around the objects and overlapping between bounding-boxes may be easily checked. The narrow phase: individual pairs of probably closer bodies are checked.

A hierarchical approach can be used:

consider two complicated non-convex sets  $E$ ,  $F$  to be checked for collisions. They can be part of the same robots or of a robot and an obstacles. They are subsets of  $\mathbb{R}^2$  or  $\mathbb{R}^3$ , defined using any kind of geometric primitives, such as triangles. The idea is to decompose a body into a set of bounding boxes. Such boxes may be as tight as possible around the part of the body or may be as simple as possible so that intersection test is easy.



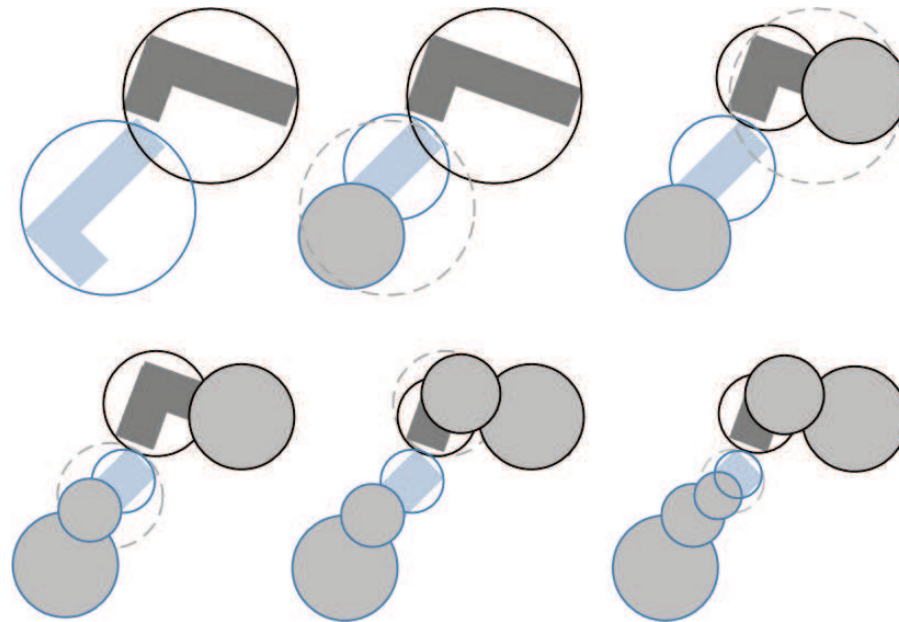
Consider a tree  $T_E$  and  $T_F$  for each set  $E$  and  $F$  a vertex corresponds to a region  $X$  of points of  $E$  (or  $F$ ) in a bounding-box. Two child-vertices are constructed by defining two smaller subset of  $X$  with associated bounding-boxes whose union cover  $X$ .



If root vertices  $e_0$  and  $f_0$ , of  $T_E$  and  $T_F$  respectively, do not overlap no collision occurs between  $E$  and  $F$ . If they overlap we consider the children  $e_{1,1}$  and  $e_{1,2}$  of the root  $e_0$  of  $T_E$  and we check the overlapping between their bounding-boxes and the bounding-box of the root  $f_0$  of  $T_F$ . In case of overlapping (e.g.  $e_{1,2}$  with  $f_0$ ) we test  $e_{1,2}$  with the children  $f_{1,1}$  and  $f_{1,2}$ . The procedure is iterated.

For each non overlapping node  $e_{i,j}$  or  $f_{i,j}$  the region is not further explored by

creating new children nodes and hence the branch is cut.



## Collision detection along paths

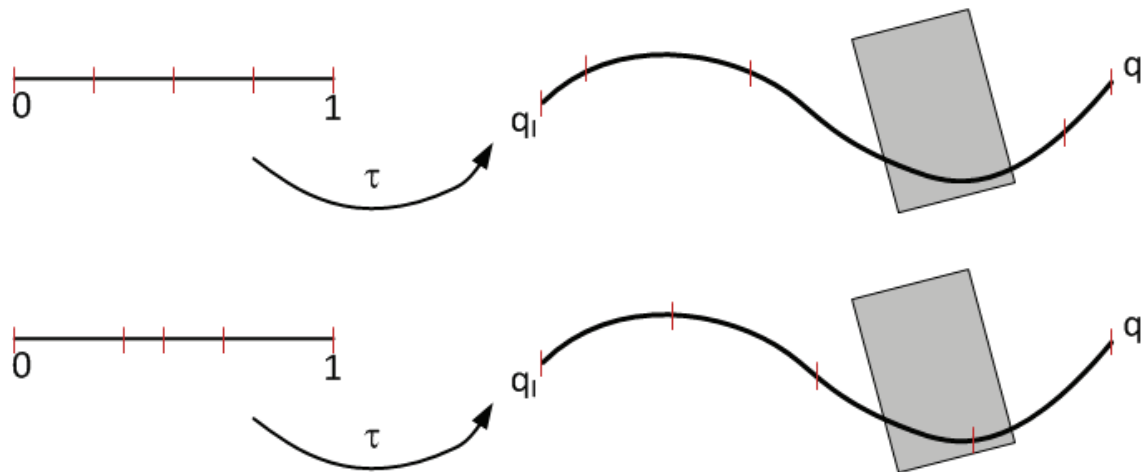
Motion planning algorithm will also require that an entire path is in  $\mathcal{C}_{free}$ .

Hence, given a parametrization of the path  $\tau : [0, 1] \rightarrow \mathcal{C}$  we must be able to check whether  $\tau([0, 1]) \subset \mathcal{C}_{free}$ .

A possible solution is to determine a resolution  $\Delta_q$  in  $\mathcal{C}_{free}$  that induce a step size  $t_2 - t_1$  where  $t_1, t_2 \in [0, 1]$  and  $\rho(\tau(t_1), \tau(t_2)) \leq \Delta_q$  where  $\rho$  is a metric on  $\mathcal{C}$ .

If  $\Delta_q$  is too small we incur in high computational times. On the other hand, if it is too large we may miss collisions.

Choosing a resolution in  $[0, 1]$  may lead to a non efficient resolution in  $\mathcal{C}$  and to a collision missing.



## Incremental Sampling and Searching

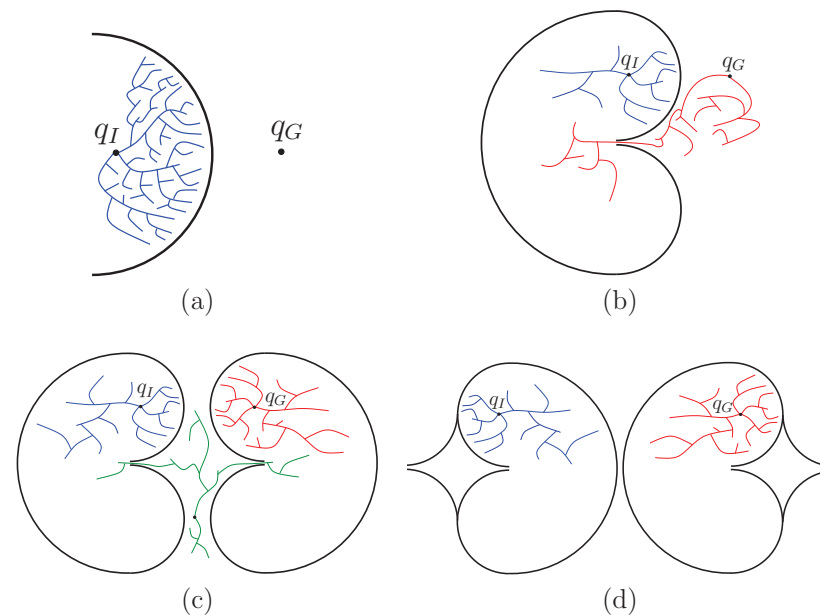
Such methods are similar to those shown in the chapter on discrete optimization. The main difference is that in those algorithm edges were representing control action while incremental sampling and searching algorithm construct a topological graph where edges are path segments.

The main idea behind such algorithm may be synthetized as follows:

1. **Initialization:** consider a graph  $\mathcal{G}(V, E)$  with  $E = \emptyset$  and  $V$  contains at least  $q_I$  and  $q_G$  (and possibly other points in  $\mathcal{C}_{free}$ ).
2. **Vertex Selection Method (VSM):** Choose vertex  $q_{cur} \in V$  to expand the graph.
3. **Local Planning Method (LPM):** For some  $q_{new} \in \mathcal{C}_{free}$  try to construct a path  $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$  with  $\tau(0) = q_{cur}$  and  $\tau(1) = q_{new}$ . If a collision occurs along  $\tau$  go to step 2.
4. **Insert edges and nodes to the graph:** Insert  $\tau$  in  $E$  and if not already in  $V$  insert  $q_{new}$  in  $V$ .
5. **Check for a solution:** Check if in  $\mathcal{G}$  there is the desired path.
6. **Iterate:** Iterate until a solution is found or termination conditions are met.

Algorithms usually differ on implementations of VSM (similar to priority queue) and LPM.

Similarly to algorithms for graph explorations, unidirectional (single-tree), bidirectional (two-trees) or multi-directional (more than two trees) methods can be used. Bidirectional and multi-directional methods are useful in case of complex spaces with “traps” but are more difficult to manage.



For bidirectional trees we can alternate between trees when selecting vertices

while, in case of more than two trees, heuristics on which pair of trees should be selected for connection must be used.

### Re-adaptation of grid search algorithms

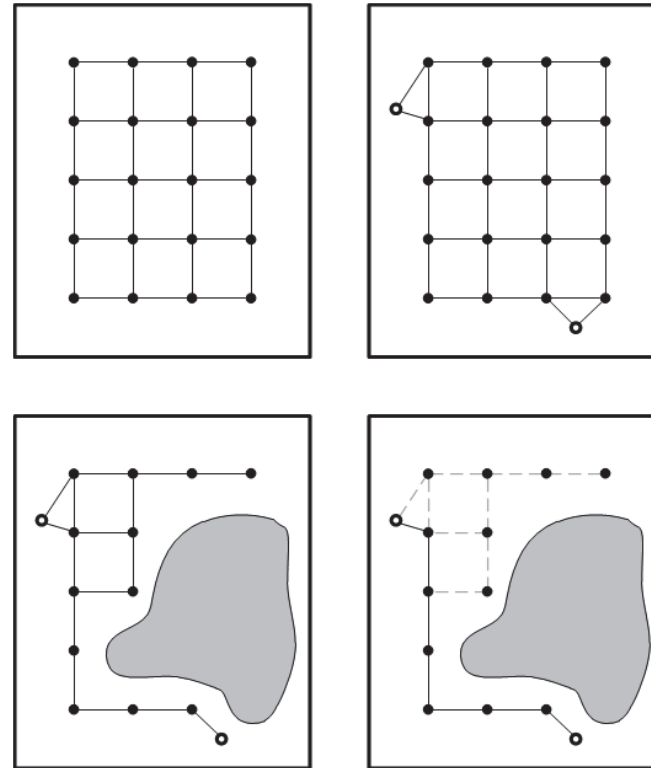
Grid search algorithm can be used with the proposed sample and searching scheme. The basic idea is to discretize each dimension of the  $\mathcal{C}$ -space obtaining  $k$ -neighbourhood with  $k \geq 1$ . The algorithm start searching the closest 1-neighbourhood (or  $k$ -neighbourhood) that are in  $\mathcal{C}_{free}$ .

For example:

$$N_2(q) = \{q \pm \Delta q_i \pm \Delta q_j | 1 \leq i, j \leq n, i \neq j\} \cup N_1(q).$$

A candidate vertex is extracted from the  $k$ -neighbourhood of  $q_{cur}$ , it is checked for collision. Then a path on the grid from  $q_{cur}$  is computed and tested for collision. The grid graph is computed on the fly.





If a solution is not found there are two possible alternatives: increase resolution by tuning parameters (how? how much? how often interleave between search and sampling?) renounce to the grid and work on the continuous space.

## Rapidly Exploring Dense Tree

Idea: no parameter tuning but incrementally construct a search tree that gradually improves the resolution. The tree will densely cover the space in the limit (nodes and edges arbitrarily close to any configuration of the space).

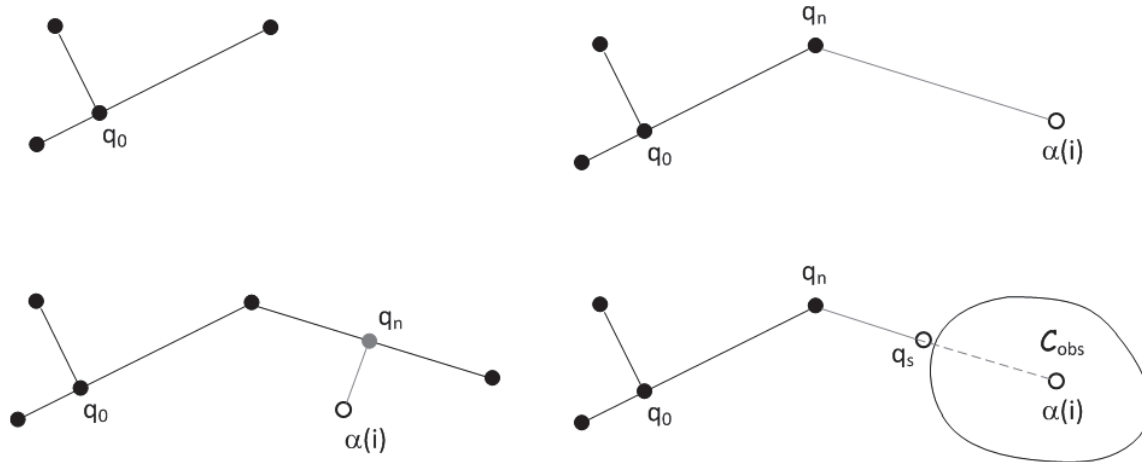
At the base of such algorithm there is a dense sequence of samples  $\alpha(i)$ . If the sequence is random the tree is called *Rapidly exploring Random Tree (RRT)* while in general is called *Rapidly exploring Dense Tree (RDT)*.

Consider first only the exploration of the tree in a obstacle free  $\mathcal{C}$ -space. An RTD is a topological graph  $\mathcal{G} = (V, E)$ . With  $S \subset \mathcal{C}_{free}$  we denote the set of all points reached by  $\mathcal{G}$  (*swath of the graph*):

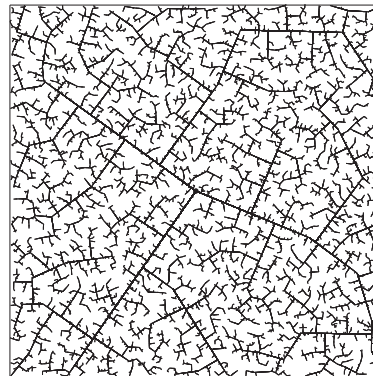
$$S = \bigcup_{e \in E} e([0, 1])$$

**Scheme of RDT Algorithm with no obstacle ,**

- 1  $\mathcal{G}.init(q_0)$
- 2 **for**  $i = 1$  **to**  $k$  **do**
- 3      $\mathcal{G}.add\_vertex(\alpha(i));$
- 4      $q_n \leftarrow \text{NEAREST}(S(\mathcal{G}), \alpha(i));$
- 5      $\mathcal{G}.add\_edge(q_n, \alpha(i));$



45 iterations



2345 iterations

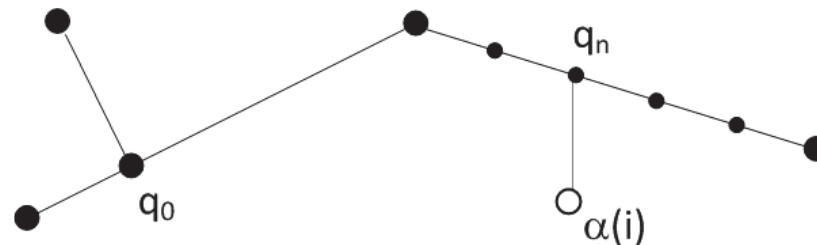
### Scheme of RDT Algorithm ,

```

1  $\mathcal{G}.\text{init}(q_0)$ 
2 for  $i = 1$  to  $k$  do
3      $q_n \leftarrow \text{NEAREST}(S(\mathcal{G}), \alpha(i));$ 
4      $q_s \leftarrow \text{STOPPING-CONFIGURATION}(q_n, \alpha(i));$ 
5     if  $q_s \neq q_n$  then
6          $\mathcal{G}.\text{add\_vertex}(q_s);$ 
7          $\mathcal{G}.\text{add\_edge}(q_n, q_s);$ 

```

Two possible approaches for NEAREST computation: exact and approximation. Exact approaches may be very difficult in case on paths that are not segments (e.g. in  $SO(3)$ ), while approximate approaches require a resolution parameter.



## Planning on a RDT

For a single-tree search the idea is to grow a tree from  $q_I$  and periodically ( $k$  iterations) check if  $q_G$  is reachable in  $\mathcal{C}_{free}$ . This can be done by considering  $q_G$  as the sampled node.

In case of a bidirectional search two RDT are grown from  $q_I$  and  $q_G$  and the growth should be balanced between the two. After a given number of iteration a node added in a tree is used as a random sample for the other one. When the cardinality of a tree is larger than the other the procedure is swapped between the two trees. The cardinality of a tree can be computed in terms of number of nodes or total length of all arcs.

More formally, we first introduce the following concepts

Let  $r \in \mathbb{R}_+$ ,  $n, d \in \mathbb{N}$ , the Random  $r$ -Disc Graph in  $d$  dimensions  $G^{disc}(n, r)$  is a graph whose  $n$  vertices  $\{X_1, \dots, X_n\}$  are independent and uniformly distributed random variables in  $(0, 1)^d$  and such that  $(X_i, X_j)$  with  $i, j \in \{1, \dots, n\}$ ,  $i \neq j$  is an edge if and only if  $\|X_i - X_j\| < r$ .

**Theorem (Penrose 2003): Connectivity of random  $r$ -Graphs**

Given a random  $r$ -disc Graph in  $d$  dimensions,  $G^{disc}(n, r)$ , it holds

$$\lim_{n \rightarrow +\infty} P(\{G^{disc}(n, r) \text{ is connected}\}) = \begin{cases} 1 & \text{if } \zeta_d r^d > \log(n)/n \\ 0 & \text{if } \zeta_d r^d < \log(n)/n \end{cases}$$

where  $\zeta_d$  is the volume of the unit ball in  $d$  dimensions.

Let  $n, d, k \in \mathbb{N}$ , the Random  $k$ -Nearest Neighbour Graph in  $d$  dimension  $G^{near}(n, k)$  is a graph whose  $n$  vertices  $\{X_1, \dots, X_n\}$  are independent and uniformly distributed random variables in  $(0, 1)^d$  and such that  $(X_i, X_j)$  with  $i \neq j$  is an edge if and only if  $X_j$  is among the  $k$  nearest neighbours of  $X_i$  or viceversa.

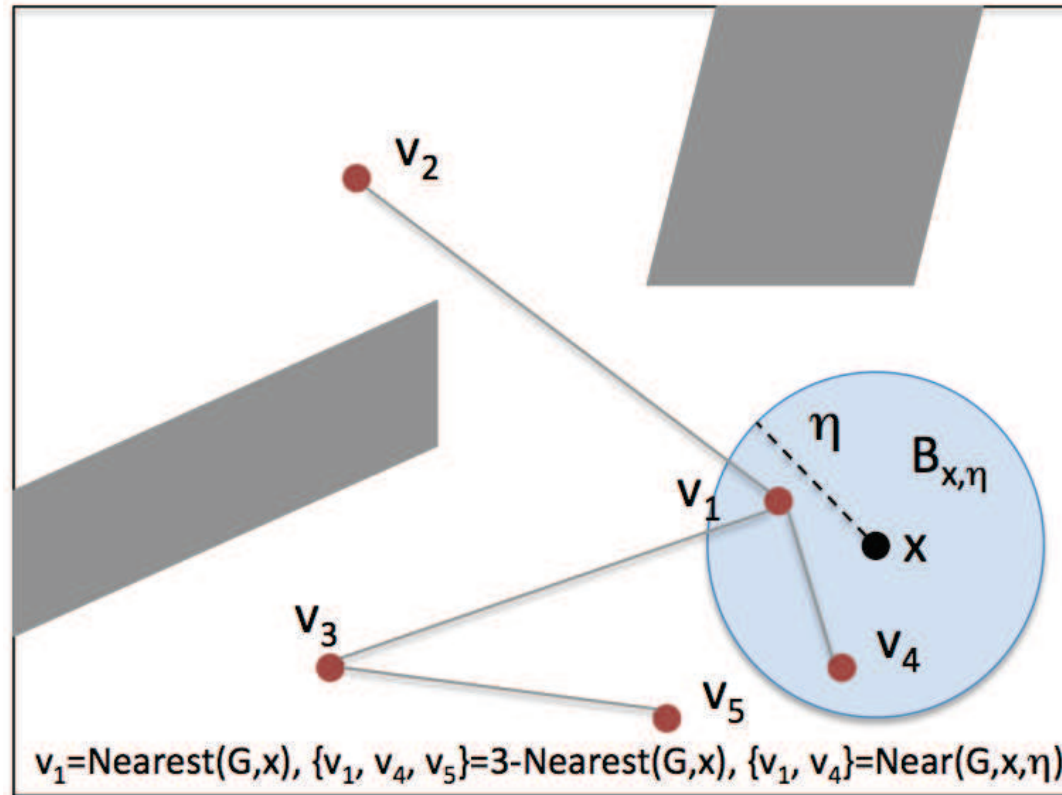
**SAMPLING:** A sample procedure is a map that provides a sequence of points in  $\mathcal{C}$  such that the samples are independent and identically distributed. It may be convenient to consider a map (SampleFree in the following) that provides a subsequence of points in  $\mathcal{C}_{free}$ .

**NEAREST NEIGHBOUR:** is a map that given graph  $G = (V, E)$  and a point  $x \in \mathcal{C}$  provides the closest vertex  $v = Nearest(G, x)$  to  $x$  for a given distance function. For example,  $v = argmin_{\bar{v} \in V} \|x - \bar{v}\|$ . The function  $k - Nearest(G, x, k) = \{v_1, \dots, v_k\}$  provides the  $k$  nearest vertex of  $V$  to  $x$ .

**NEAREST VERTICES:** is a map that given graph  $G = (V, E)$  a point  $x \in \mathcal{C}$  and a parameter  $r \in \mathbb{R}$  provides the subset of vertices in  $V$  that are contained in a ball of radius  $r$  centered in  $x$ :

$$Near(G, x, r) = \{v \in V | v \in B_{x,r}\}$$

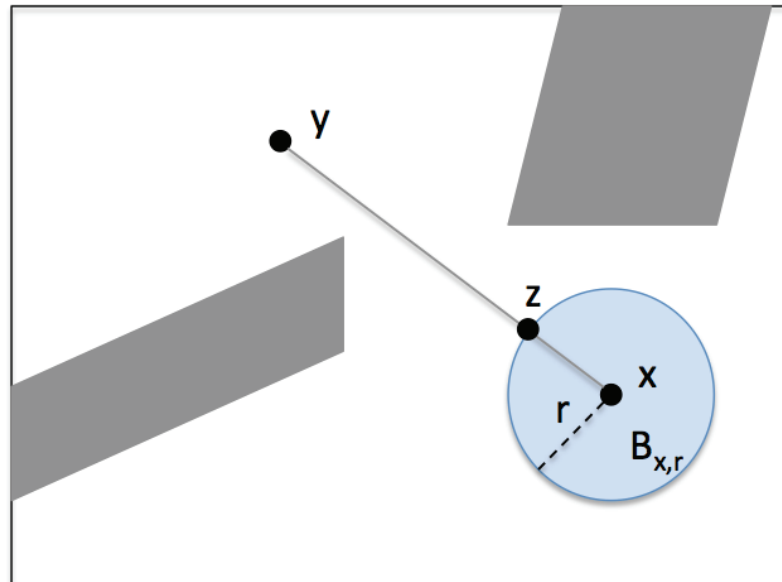
where the same distance of Nearest function is used.





**STEERING:** given two configurations  $x, y \in \mathcal{C}$  and a value  $\eta > 0$  the map provides the point closer to  $y$  than  $x$  is:

$$\text{Steer}(x, y) = \operatorname{argmin}_{z \in B_{x, \eta}} \|z - y\|$$



**COLLISION TEST:** given two configurations  $x, x' \in \mathcal{C}$  the function  $\text{ObstacleFree}(x, x')$  returns *True* if the line segment between  $x$  and  $x'$  lies in  $\mathcal{C}_{\text{free}}$ , *False* otherwise.

## Probabilistic RoadMaps (PRM)

Consider the case of multiple queries for the same robot and environment. In this case, it worth spending more time in sampling than searching in order to preprocess the models to better handling future queries. Hence, a *roadmap* is built based on *Probabilistic Roadmap Methods* (or *Sampling-based Roadmaps*).

The basic method consists in the *preprocessing phase* and the *query phase*.

The idea of the preprocessing phase is to build a graph by attempting connections among  $n$  randomly sampled points in  $\mathcal{C}$ . The graph should be easily accessible from any point in  $\mathcal{C}_{free}$

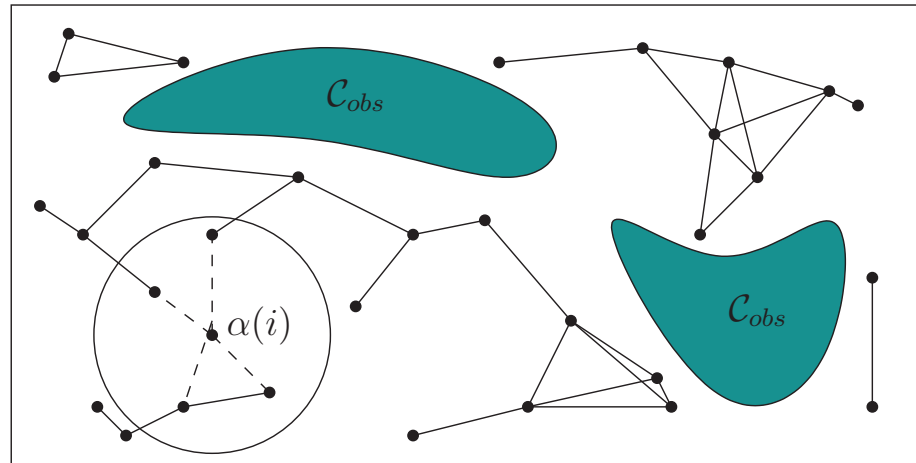
### Scheme of PRM Preprocessing ,

```

1  $V \leftarrow \emptyset; E \leftarrow \emptyset;$ 
2 for  $i = 0, \dots, n$  do
3      $q_{rand} \leftarrow \text{SampleFree}(\omega_i)$ 
4      $U \leftarrow \text{Near}(G, q_{rand}, r)$ 
5      $V \leftarrow V \cup \{q_{rand}\}$ 
6     for each  $u \in U$  (in order of increasing  $\|u - q_{rand}\|$ ) do
7         if  $q_{rand}$  and  $u$  are not in the same connected component of  $G$  then
8             if  $\text{ObstacleFree}(q_{rand}, u)$  then  $E \leftarrow E \cup \{(q_{rand}, u), (u, q_{rand})\}$ 
9 return  $G$ 

```

For construction the roadmap is a forest.

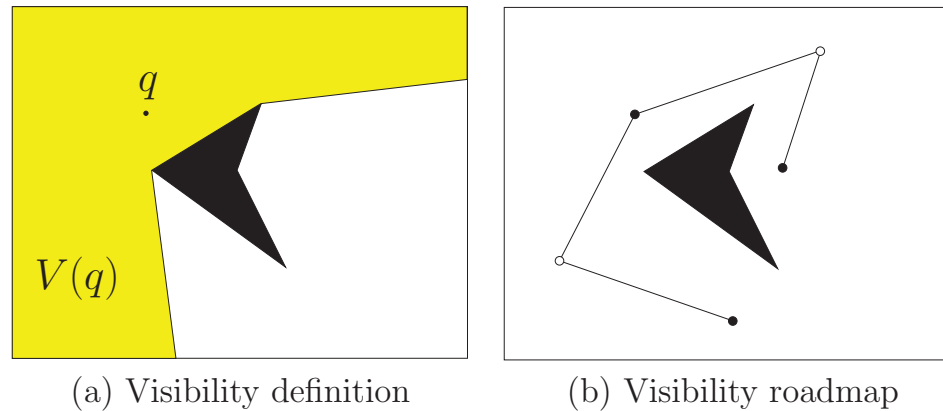


The query phase can be processed only when the tree is sufficiently complete to cover the whole space. Indeed, given  $q_I$  and  $q_G$  they can be used as samples and tried to be connected to  $\mathcal{G}$ .

If connection is found a classical graph path search is conducted. Otherwise, with the obtained resolution (given by the sampling dispersion) no solution is found.

## Visibility Roadmap

Another possible approach is to consider a smaller representation of the roadmap that is based on the visibility concept. This approach has larger computational costs but suits well for multi-query cases.



A vertex  $q \in V$  of graph  $\mathcal{G}$  is a *Guard* if no other guards lay on the visibility region  $V(q)$  of  $q$ , i.e.  $\exists \tilde{q}$  guard such that  $\tilde{q} \in V(q)$ .  $q \in V$  of graph  $\mathcal{G}$  is a *Connector* if there are at least two different guards in its visibility region, i.e.  $\exists \tilde{q}_1, \tilde{q}_2$  guards such that  $q \in V(\tilde{q}_1) \cap V(\tilde{q}_2)$ .

The neighbourhood function returns the entire set  $V$  and hence each sample  $\alpha(i)$  tries to connect with all vertices. However, if  $\alpha(i)$  cannot be connected to any guards (i.e.,  $\alpha(i) \notin V(q_G) \forall q_G$  guards),  $\alpha(i)$  becomes a guard and it is inserted in

$\mathcal{G}$ . If  $\alpha(i)$  can be connected to two guards of two different connected component of  $\mathcal{G}$ ,  $\alpha(i)$  and the two edges towards the guards are inserted in  $\mathcal{G}$ . Otherwise,  $\alpha(i)$  is discarded.

A positive aspect of this approach is that there is a dramatic reduction in the number of vertices thanks to the possible discard of samples. Notice that the algorithm is probabilistic complete if random samples are used and is resolution complete if sample are dense even though samples are discarded.

A drawback is that better guards can come up during tree construction but are not taken into account.

## Rapidly Exploring Random Tree (RRT)

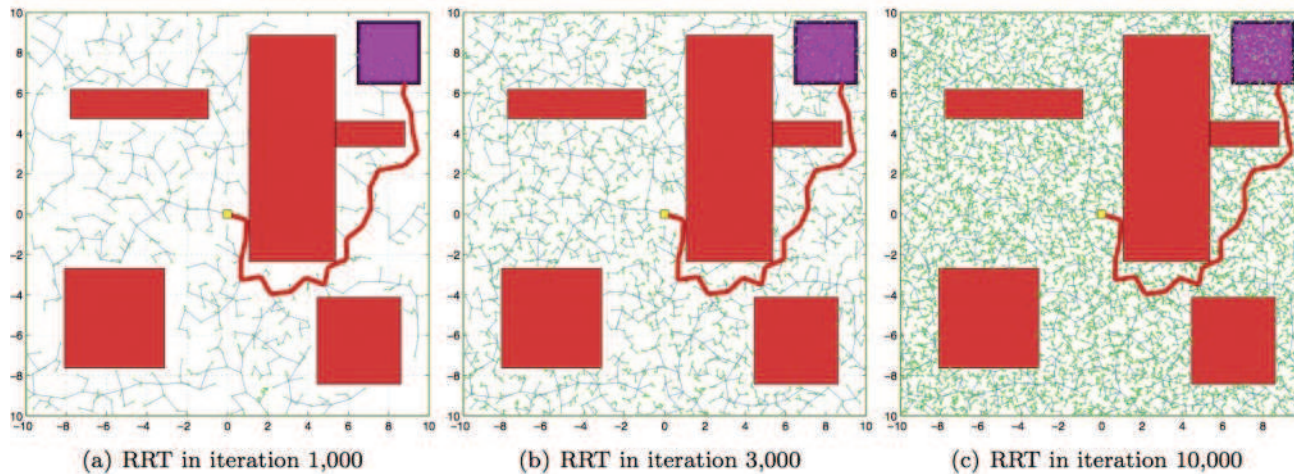
Consider the case of single query, the RRT scheme is as follows

**Scheme of RRT** ,

```

1  $V \leftarrow \{q_{init}\}; E \leftarrow \emptyset;$ 
2 for  $i = 0, \dots, n$  do
3      $q_{rand} \leftarrow SampleFree(\omega_i)$ 
4      $q_{nearest} \leftarrow Nearest(G, q_{rand})$ 
5      $q_{new} \leftarrow Steer(q_{nearest}, q_{rand})$ 
6     if  $ObstacleFree(q_{nearest}, q_{new})$  then
7          $V \leftarrow V \cup \{q_{new}\}; E \leftarrow E \cup \{(q_{nearest}, q_{new})\}$ 
8 return  $G$ 

```



Figures from Prof. Karaman website: <http://sertac.scripts.mit.edu/web/?p=502>

## Optimality in sample based methods

In case we are interest in finding optimal paths, a cost function must be introduced and the RRT and PRM methods can be modified accordingly to provide optimal paths. Since the methods are sample based the optimality is in probability (almost sure convergence to optimal solutions). One of the main differences between RRT\* (PRM\*) and RRT (PRM) is the use of a varying value of  $r$  in the function *Near*. The idea is to chose a value that decreases for increasing number of vertices in the graph guaranteeing that a neighbour vertex exists with high probability.

### Optimal Probabilistic RoadMap

#### Scheme of PRM\* Preprocessing ,

```

1  $V \leftarrow \{q_{init}\} \cup \{SampleFree(\omega_j) | j = 1, \dots, n\}; E \leftarrow \emptyset;$ 
2 for each  $v \in V$  do
3      $U \leftarrow Near(G, q_{rand}, r(n)) \setminus \{v\}$ 
4     for each  $u \in U$  do
5         if ObstacleFree( $v, u$ ) then  $E \leftarrow E \cup \{(v, u), (u, v)\}$ 
6 return  $G$ 

```

where  $r(n) = \gamma_{PRM} \left( \frac{\log(n)}{n} \right)^{1/d}$  e  $\gamma_{PRM} > \gamma_{PRM}^* = 2(1 + 1/d)^{1/d} (\mu(C_{free})/\zeta_d)^{1/d}$ ,  $\mu(C_{free})$  is the volume of  $C_{free}$ . The connection radius decay with  $n$  and the average number of connection attempted is proportional to  $\log(n)$ .

## RRT\*

The main concept of RRT\* is the rewiring of paths that are not optimal when a new node is inserted. Hence, arcs are added and removed from the tree based on the cost of the paths.

Let  $Line(q_1, q_2) : [0, s] \rightarrow \mathcal{C}$  be the straight line between the configurations  $q_1$  and  $q_2$ . Let denote with  $Parent(v)$  the unique vertex  $u$  such that  $(u, v) \in E$ .

$Cost : V \rightarrow \mathbb{R}_+$  maps a vertex  $v$  in the cost of the unique path from the root vertex  $v_0$  to  $v$ . We suppose  $Cost(v) = Cost(Parent(v)) + c(Line(Parent(v), v))$  and  $Cost(v_0) = 0$ .

Let  $r(|V|) = \min\{\gamma_{RRT^*}(\log(|V|)/|V|)^{1/d}, \eta\}$  with  $\gamma_{RRT^*} \geq 2(1 + 1/d)^{1/d}(\mu(\mathcal{C}_{free})/\zeta_d)^{1/d}$ ,  $\mu(\mathcal{C}_{free})$  is the volume of  $\mathcal{C}_{free}$ .

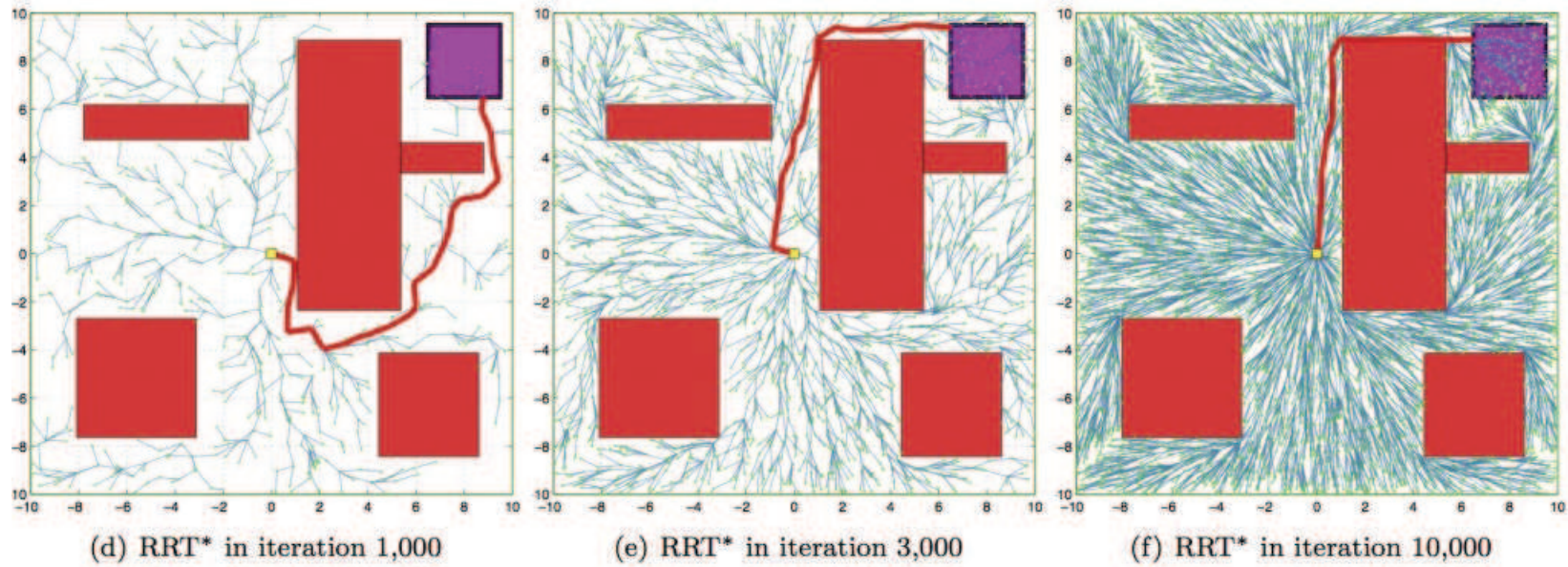


**Scheme of RRT\* ,**

```

1  $V \leftarrow \{q_{init}\}; E \leftarrow \emptyset;$ 
2 for  $i = 0, \dots, n$  do
3    $q_{rand} \leftarrow \text{SampleFree}(\omega_i)$ 
4    $q_{nearest} \leftarrow \text{Nearest}(G, q_{rand})$ 
5    $q_{new} \leftarrow \text{Steer}(q_{nearest}, q_{rand})$  (depends on value  $\eta$ )
6   if  $\text{ObstacleFree}(q_{nearest}, q_{new})$  then
7      $Q_{near} \leftarrow \text{Near}(G, q_{new}, r(|V|))$ 
8      $V \leftarrow V \cup \{q_{new}\}; E \leftarrow E \cup \{(q_{nearest}, q_{new})\}$ 
9      $q_{min} \leftarrow q_{nearest}; c_{min} \leftarrow \text{Cost}(q_{nearest}) + c(\text{Line}(q_{nearest}, q_{new}))$ 
10    for each  $q_{near} \in Q_{near}$  do
11      if  $\text{ObstacleFree}(q_{near}, q_{new}) \wedge \text{Cost}(q_{near}) + c(\text{Line}(q_{near}, q_{new})) < c_{min}$ 
12    then
13       $q_{min} \leftarrow q_{near}; c_{min} \leftarrow \text{Cost}(q_{near}) + c(\text{Line}(q_{near}, q_{new}))$ 
14       $E \leftarrow E \cup \{(q_{min}, q_{new})\}$ 
15      for each  $q_{near} \in Q_{near}$  do
16        if
17           $\text{ObstacleFree}(q_{new}, q_{near}) \wedge \text{Cost}(q_{new}) + c(\text{Line}(q_{new}, q_{near})) < \text{Cost}(q_{near})$  then
18             $q_{parent} \leftarrow \text{Parent}(q_{near});$ 
19             $E \leftarrow (E \setminus \{(q_{parent}, q_{near})\}) \cup \{(q_{new}, q_{near})\}$ 
20 return  $G$ 

```



Figures from Prof. Karaman website: <http://sertac.scripts.mit.edu/web/?p=502>

## Kynodynamic Planning

Consider the problem of determining a control law  $u$  to steer a dynamic system  $\dot{x} = f(x, u)$  from  $x(0) = x_0$  toward a reachable point  $x_G$  avoiding all obstacles.

Consider also the cost function  $J(x) = \int_0^T g(x(t))dt$  to be minimized.

Let  $dist : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$  be the optimal cost of a trajectory that connects the two points avoiding all obstacles. More formally

$$\begin{aligned} dist(z_1, z_2) &= \min_{T \in \mathbb{R}_+, u: [0, T] \rightarrow \mathcal{U}} J(x) \text{ subject to} \\ &\dot{x} = f(x, u), \forall t \in [0, T] \\ &x(0) = z_1 \\ &x(T) = z_2 \end{aligned}$$

Also in this case the Nearest function is given by

$$Nearest(G, z) = \operatorname{argmin}_{v \in V} dist(v, z).$$

In order to define the set of vertices near a sample we first have to consider small time controllable points. Let

$Reach(z, l) = \{z' \in \mathcal{X} : dist(z, z') \leq l \vee dist(z', z) \leq l\}$  where  $l(n)$  is chosen so that  $Reach(z, l(n))$  contains a ball of volume  $\gamma \log(n)/n$ . The set of near vertices

is hence  $Near(G, z, n) = V \cap Reach(z, l(n))$  with  $n = |V|$ .

The (local) steering function computes then an optimal path between two points  $z_1$  and  $z_2$  with  $\|z_1 - z_2\| \leq \epsilon$ . The function provides both the control and the time that are solution of the optimal control problem above.

Example of an optimal planning for a Dubins car is reported in figure:

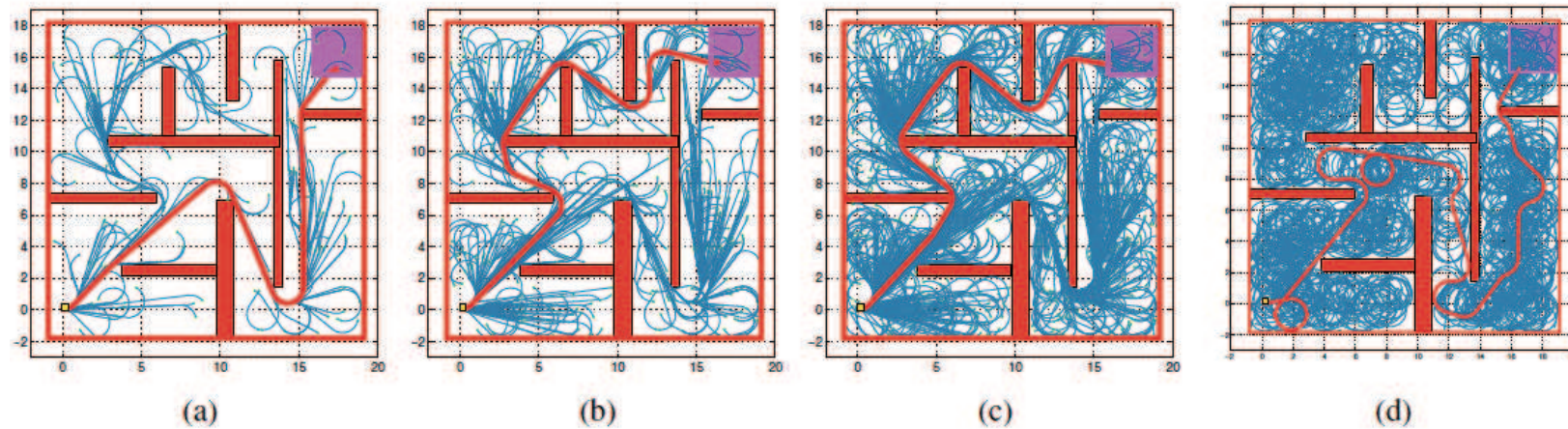


Figura 17: From a) to c) from 500 to 6500 samples for RRT\*, in d) RRT after 2000 samples. Picture from Karaman, Sertac, and Emilio Frazzoli. “Optimal Kinodynamic Motion Planning Using Incremental Sampling-based Methods.” 49th IEEE Conference on Decision and Control (CDC). Atlanta, GA, USA, 2010. 7681-7687.

## Artificial potential based planning

For online planning it can be useful to use the artificial potential field approach<sup>a</sup>. The idea is to let a point (representative point of the robot such as the center of mass) move under the action of a potential field  $U$ . The potential field is the combination of an attractive action toward the desired target and a repulsive action from  $\mathcal{C}_{obs}$ . The most promising direction of motion is given by the opposite of the gradient  $-\nabla U(q)$ .

### Attractive potential

Let  $q_g$  be the goal configuration we want to steer the system to.  $q_g$  is hence considered as a minimum of a potential, e.g.  $U_{a_1}(q) = \frac{1}{2}k_a e^T(q)e(q) = \frac{1}{2}k_a \|e\|^2$  with  $k_a > 0$  and  $e(q) = q - q_g$ .  $U_{a_1}$  is positive definite with a global minimum in  $q_g$ . The attractive force  $f_{a_1} = -\nabla U_{a_1}(q) = k_a e(q)$  has large module for configuration far from  $q_g$ .

Another possible choice is  $U_{a_2}(q) = k_a \|e(q)\|$  with a corresponding attractive force  $f_{a_2} = -\nabla U_{a_2}(q) = k_a \frac{e(q)}{\|e(q)\|}$  that is constant in module but it is not defined in  $q_g$ .

---

<sup>a</sup>Refer to the book: “Robotics: Modelling, Planning and Control” by Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G., Springer-Verlag London, 2009.

A possible trade-off between the two potentials is to glue them in  $\|e(q)\| = 1$  to have an attractive force that is limited in module and is defined everywhere:

$$U_a(q) = \begin{cases} U_{a_2} & \text{for } \|e(q)\| \geq 1 \\ U_{a_1} & \text{otherwise} \end{cases}$$

### Repulsive potential

To avoid collisions with  $\mathcal{C}_{obs}$  the idea is to superimpose a repulsive potential to the one that attracts the robot toward  $q_g$ . For any  $\mathcal{CO}_i$  convex component of  $\mathcal{C}_{obs}$  we define the following repulsive potential

$$U_{r_i}(q) = \begin{cases} \frac{k_{r,i}}{\gamma} \left( \frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}} \right)^\gamma & \text{for } \eta_i(q) \leq \eta_{0,i} \\ 0 & \text{otherwise} \end{cases}$$

where  $k_{r,i} > 0$  and  $\eta_i(q) = \min_{q' \in \mathcal{CO}_i} \|q - q'\|$  is the distance of  $q$  from  $\mathcal{CO}_i$ .  $\eta_{0,i}$  is the radius of influence of  $U_{r,i}$ . The constant  $\gamma$  is typically chosen equal to 2.

The repulsive force is

$$f_{r_i}(q) = -\nabla U_{r,i} = \begin{cases} \frac{k_{r,i}}{\eta_i^2(q)} \left( \frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}} \right)^{\gamma-1} \nabla \eta_i(q) & \text{for } \eta_i(q) \leq \eta_{0,i} \\ 0 & \text{otherwise} \end{cases}$$

From the convexity of any  $\mathcal{CO}_i$  we have that  $q_m = \operatorname{argmin}_{q' \in \mathcal{CO}_i} \|q - q'\|$  exists and it is unique.  $\nabla \eta_i(q)$  is orthogonal to the equipotential curve through  $q$  and it is directed as the oriented half-line from  $q_m$  through  $q$ .

The total repulsive potential is  $U_r(q) = \sum_{i=1}^p U_{r,i}(q)$ . We suppose that the goal configuration is out of the radius of influence of any  $\mathcal{CO}_i$ , i.e.

$$\eta(q_g) > \eta_{0,i}, \forall i = 1, \dots, p.$$

The total potential the robot is subject to is  $U_t(q) = U_a(q) + U_r(q)$  with a corresponding force  $f_t(q) = -\nabla U_t = f_a(q) + \sum_{i=1}^p f_{r,i}(q)$ .

The problem with such approach is that there exists a unique global minimum but there may be several local minima where the robot can steer to. Hence, as it is the method is not complete and techniques to avoid local minima must be considered.

## Planning Method

The potential fields can be used in different ways to steer a robot. In case of online planning, the force  $f_t$  generated by the potential field can be used as generalized forces acting on the dynamical system:  $\tau = f_t(q)$ . This approach will lead toward a more natural motion since the effect of  $f_t$  is filtered by the dynamic of the robot. Another possible approach is to consider the robot as a

unitary mass point moving under the effect of  $f_t$ :  $\ddot{q} = f_t(q)$ ; or the force can be seen as a reference speed:  $\dot{q} = f_t(q)$ . The latter provides faster reaction to the effects of  $f_t$  and is used as kinematic control where low level control references are provided. The former approach requires inverse kinematic resolution. The first two methods will steer the robot toward the final destination but with not null speed as the latter does. In such cases a damping term must be added to  $f_t$ .

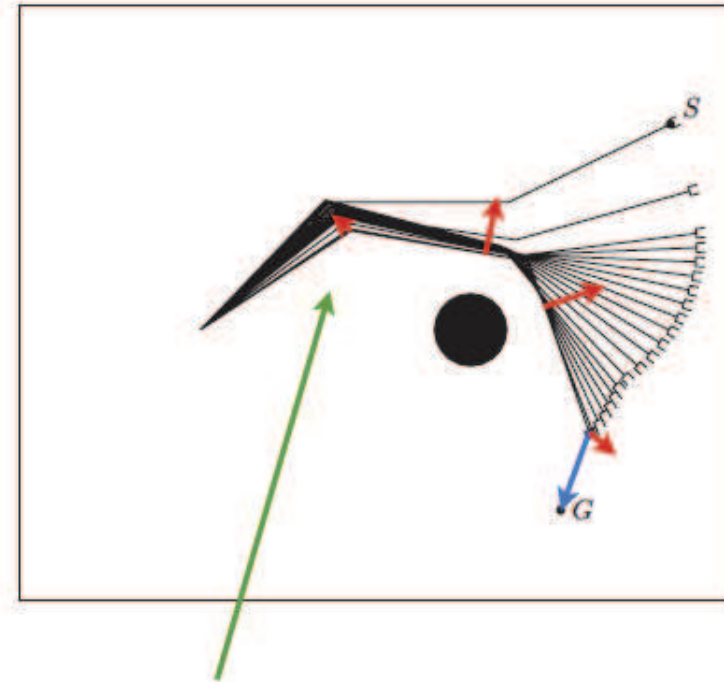
Potential planning methods can be used also in discrete (or discretized) state space where to each cell a value of the potential field is associated. The idea is still to move the robot toward potential minimizers.



success



failure



a force equilibrium  
between attractive and repulsive forces

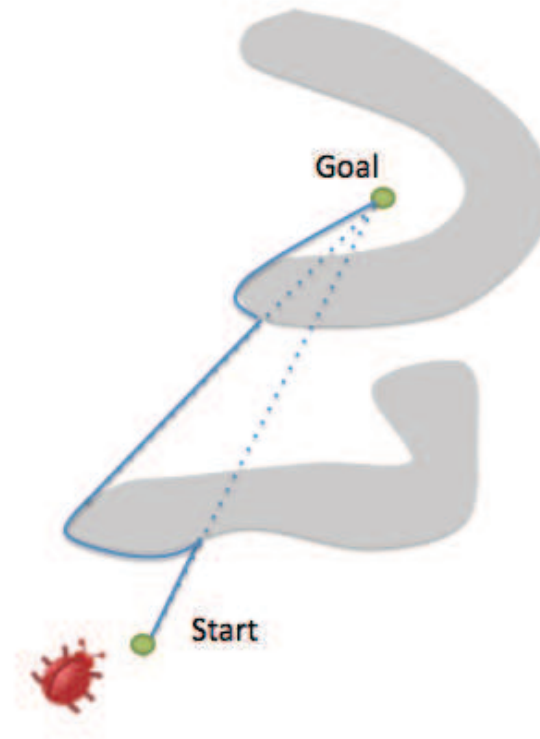
## Behaviour based Algorithms: Bug Algorithms

In case of robots with on-board sensors that provide only local knowledge of the environment, the classical planning algorithms can not be applied without an a priori information on the environment. The behaviour based planning algorithms are usually applied in case of such limited knowledge. The main idea is that robot reacts to what it detects in the environment by following pre-assigned rules.

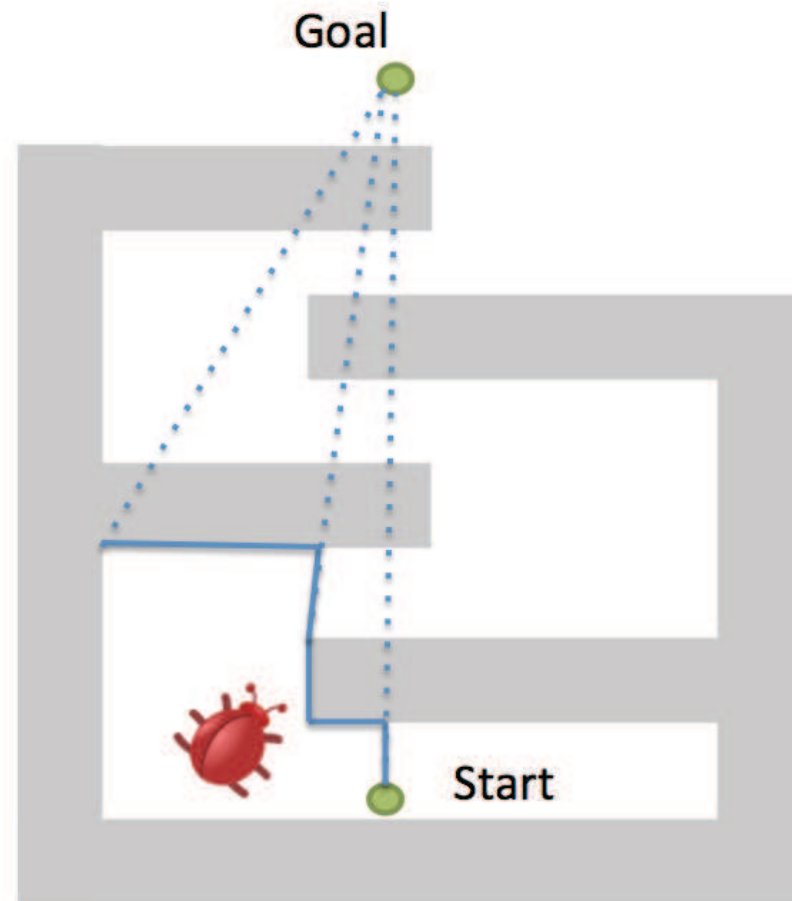
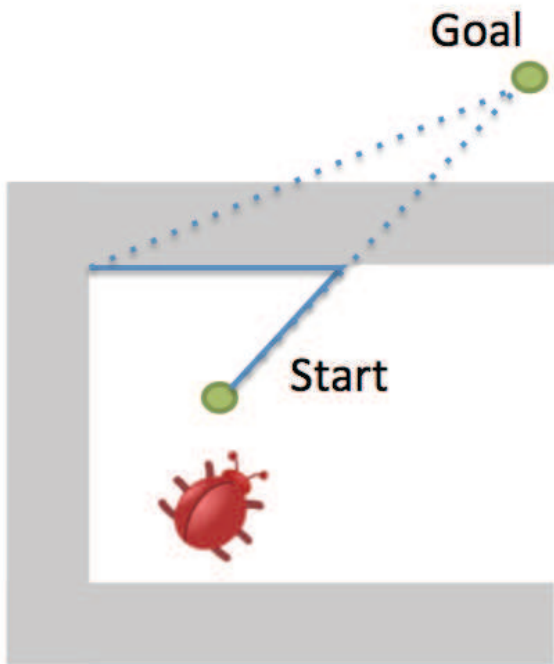
The bug algorithm is a very simple behaviour based planning method for mobile robots and is based on two behaviours (control laws): move straight to the goal, follow a wall. Few assumptions must be made: robots are points that know the direction toward the goal and can measure both the traveled distance and the distance between two points. The robots have on board (tactile) sensors that detect the contact with obstacles and are able to follow the obstacle contour. The world is supposed to be a bounded subset of  $\mathbb{R}^2$  with a finite number of obstacles.

### Scheme of Bug-0 Algorithm ,

- 1 Head toward the goal;
- 2 **if** the goal is achieved **then** stop **else**
- 3     **if** a contact with an obstacle is detected **then** follow the obstacle's boundary (on the left) until heading toward the goal is again possible



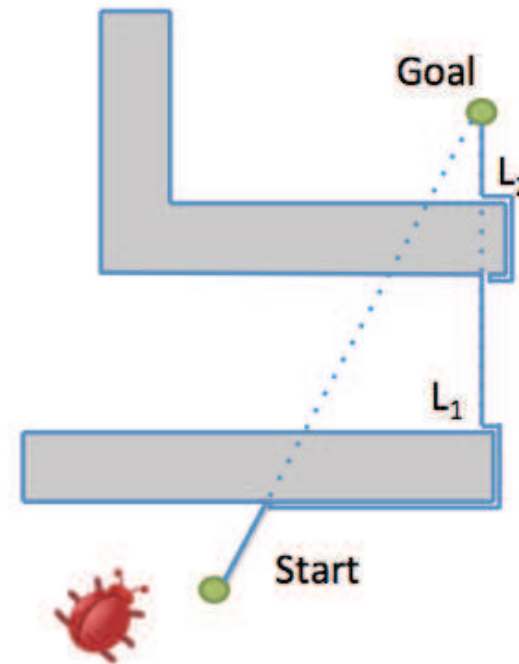
The Bug-0 Algorithm is not complete



Consider now a smarter bug that is able to compute the distance from the goal, to compute the traveled distance and to recognize a point previously crossed.

### Scheme of Bug-1 Algorithm ,

- 1 Head toward the goal;
- 2 **if** the goal is achieved **then** stop **else**
- 3     **if** a contact with an obstacle is detected **then** circumnavigate the obstacle (toward the left), identify the closest point  $L$  to the goal in the obstacle's boundary and return to this point along the shortest path along obstacle boundary.



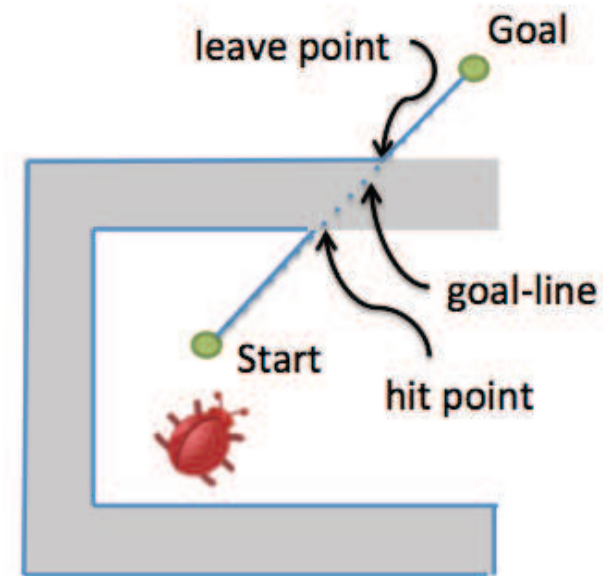
How the bug can recognize that the goal is not reachable? (e.g. from  $L$  the direction toward the goal points into an obstacle the goal is unreachable).

A lower bound of the distance  $T$  traveled by the bug is the distance  $D$  from start to goal. An upper bound is  $T \leq D + 3/2 \sum P_i$  with  $\sum P_i$  is the sum of the perimeters of all obstacles.

Consider another extension

### Scheme of Bug-2 Algorithm ,

- 1 Head toward the goal along the goal-line;
- 2 **if** the goal is achieved **then** stop **else**
- 3     **if** a hit point is reached follow the obstacle's boundary (toward the left) until the goal-line is crossed at a leave point closer to the goal than any previous hit point on the same side of the goal-line.



A lower bound of the distance  $T$  traveled by the bug is the distance  $D$  from start to goal. An upper bound is  $T \leq D + 1/2 \sum n_i P_i$  where the sum is done over all the obstacles intersected by the goal-line,  $P_i$  is the perimeter of the intersected obstacles and  $n_i$  is the number of times the goal-line intersects obstacle  $i$ .

