Motion planning

Florent Lamiraux

CNRS-LAAS, Toulouse, France

CM 1

22.09.22

Robot = set of bodies B0, …, Bm linked to one another by joints.

Definitions:

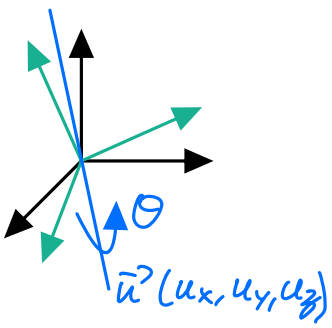
- SO(3) : group of rotation matrices R
- SE(3) : group of rigid body transformations t (or T)
- We denote T = T(R, t) rotation with transformation

Quaternion has 4 parameters and its norm is always equal to 1.

Newton's idea was to expand the idea of complex/imaginary numbers (2D) to the 3D domain.
i2=j2=k2=ink=-1
=> ij=k, …
Vector is represented by its 3 imaginary components.

Potential problem : q and -q represent the same rotation.

To represent a rotation, we can always find an axis which allows us to obtain the same result with a single rotation as the rotation matrix.



$$\cos\frac{\theta}{2} + \sin\frac{\theta}{2}(u_x \cdot i + u_y \cdot j + u_z \cdot k)$$

Context

Industrial robots

aerial robots

autonomous vehicles

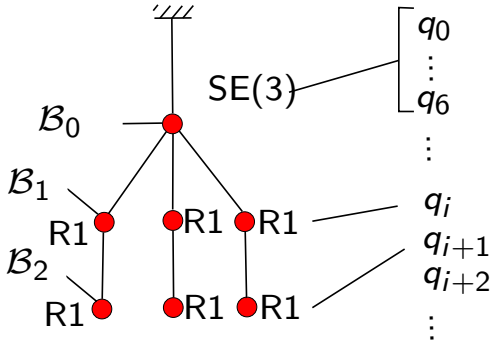
Mobile autonomous system

- moving in an environment cluttered with obstacles
- subject to kinematic or dynamic constraints

Motion planning : automatically computing a feasible trajectory between two configurations.

Robot

Set of rigid bodies $\mathcal{B}_0, \cdots \mathcal{B}_m$, linked to one another by *joints*.



Joint : parameterized rigid-body transformation between two frames (in SE(3)).

Rigid body transformation

Definitions

- SO(3) : group of 3 by 3 rotation matrices.

$$R \in SO(3) \Leftrightarrow R^T R = I_3 \text{ and } \det(R) = 1$$

- SE(3) : group of rigid body tranformations

$$T \in SE(3) \Leftrightarrow \quad \exists t \in \mathbb{R}^3, \exists R \in SO(3)$$
$$\forall x \in \mathbb{R}^3 \; T(x) = Rx + t$$

We denote $T = T_{(R,t)}$.

Joint

A joint is represented by a mapping from a sub-manifold of $\mathbb{R}^p$ in SE(3), where $p \geq 1$ is an integer.
Examples :

- Translation T1 :

$$\begin{array}{rcl} \mathbb{R} & \rightarrow & SE(3) \\ t & \rightarrow & T_{(I_3,(t\ 0\ 0))} \end{array} \qquad \text{translation along x}$$

Joint

A joint is represented by a mapping from a sub-manifold of $\mathbb{R}^p$ in SE(3), where $p \geq 1$ is an integer.
Examples :

- Translation T3 :

$$\begin{array}{rcl} \mathbb{R}^3 & \rightarrow & SE(3) \\ t & \rightarrow & T_{(I_3,t)} \end{array} \qquad \text{translation}$$

Joint

A joint is represented by a mapping from a sub-manifold of $\mathbb{R}^p$ in SE(3), where $p \geq 1$ is an integer.
Examples :

- Rotation R1 :

$$\begin{array}{rcl} \mathbb{R} & \rightarrow & SE(3) \\ t & \rightarrow & T_{(R,0)} \end{array} \qquad R = \begin{pmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Joint

A joint is represented by a mapping from a sub-manifold of $\mathbb{R}^p$ in SE(3), where $p \geq 1$ is an integer.

Examples :

- Rotation R3 :

$$\begin{array}{rcl} \mathbb{R}^4 & \rightarrow & SE(3) \\ t & \rightarrow & T_{(R,0)} \end{array}$$

$$\|t\| = 1$$

$$R = \begin{pmatrix} 1-2(t_2^2+t_3^2) & 2t_2t_1-2t_3t_0 & 2t_3t_1+2t_2t_0 \\ 2t_2t_1+2t_3t_0 & 1-2(t_1^2+t_3^2) & 2t_3t_2-2t_1t_0 \\ 2t_3t_1-2t_2t_0 & 2t_3t_2+2t_1t_0 & 1-2(t_1^2+t_2^2) \end{pmatrix}$$

$t_0 + t_1 i + t_2 j + t_3 k$ is a quaternion.

Quaternions

Non-commutative field isomorphic to $\mathbb{R}^4$, spanned by three elements i, j, k that satisfy the following relations :

$$i^2 = j^2 = k^2 = ijk = -1$$

from which we immediately deduce

$$ij = k,\ jk = i,\ ki = j$$

Hamilton (1843)

Unit Quaternions and rotations

Let $q = q_0 + q_1 i + q_2 j + q_3 k$ be a unit quaternion :

$$q_0^2 + q_3^2 + q_2^2 + q_3^2 = 1$$

$\forall x = (x_0, x_1, x_2) \in \mathbb{R}^3$, let $u = x_0 i + x_1 j + x_2 k$

$$q \,.\, u \,.\, q^* = y_0 i + y_1 j + y_2 k$$

where $q^* = q_0 - q_1 i - q_2 j - q_3 k$ is the conjugate of q.
$y = (y_0, y_1, y_2)$ is the image of x by the rotation of matrix

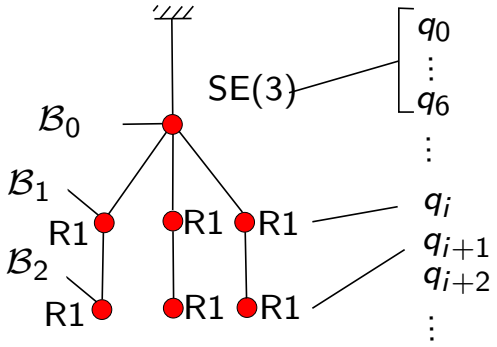
$$\begin{pmatrix} 1 - 2(q_2^2 + q_3^2) & 2q_2q_1 - 2q_3q_0 & 2q_3q_1 + 2q_2q_0 \\ 2q_2q_1 + 2q_3q_0 & 1 - 2(q_1^2 + q_3^2) & 2q_3q_2 - 2q_1q_0 \\ 2q_3q_1 - 2q_2q_0 & 2q_3q_2 + 2q_1q_0 & 1 - 2(q_1^2 + q_2^2) \end{pmatrix}$$

Unit Quaternions and rotations

- Notice that q and $-q$ represent the same rotation
- $SO(3)$ is isomorphic to $Sp(1)/\{\pm 1\}$, the half-sphere of $\mathbb{R}^4$.

Configuration of a robot

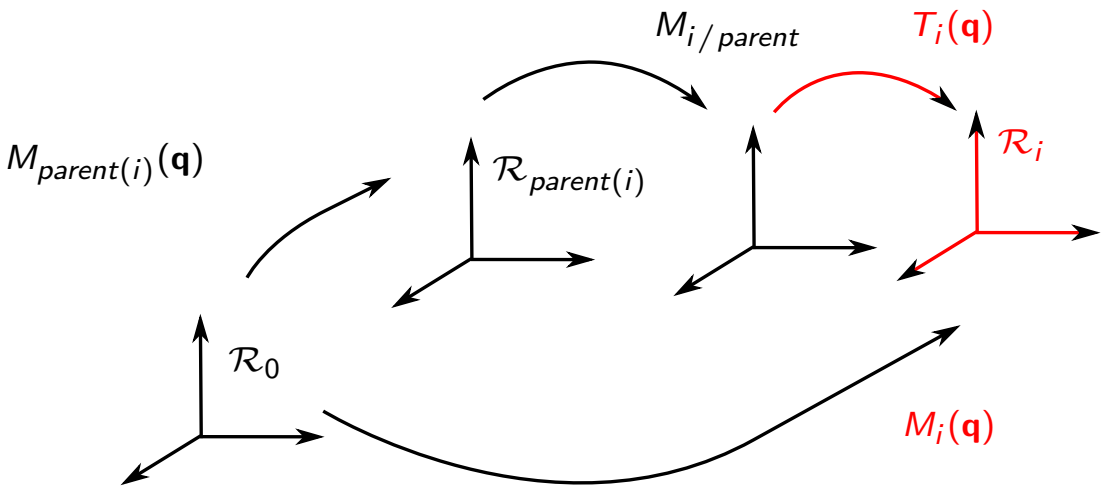
The configuration $\mathbf{q}$ of a robot is represented by the concatenation of the parameters of each joint.



Forward kinematics

Computation of the position of each joint in the global frame

$$M_i(\mathbf{q}) = M_{parent(i)}(\mathbf{q})\ M_{i/parent}\ T_i(\mathbf{q})$$



Definitions

- Workspace : $\mathcal{W} = \mathbb{R}^2$ or $\mathbb{R}^3$: space in which the robot evolves
- Obstacle in workspace : compact subset of $\mathcal{W}$, denoted by $\mathcal{O}$.
- Configuration space : $\mathcal{C}$.
- Position in configuration $\mathbf{q}$ of a point $M \in \mathcal{B}_i$: $\mathbf{x}_i(M, \mathbf{q})$.
- Obstacle in the configuration space :

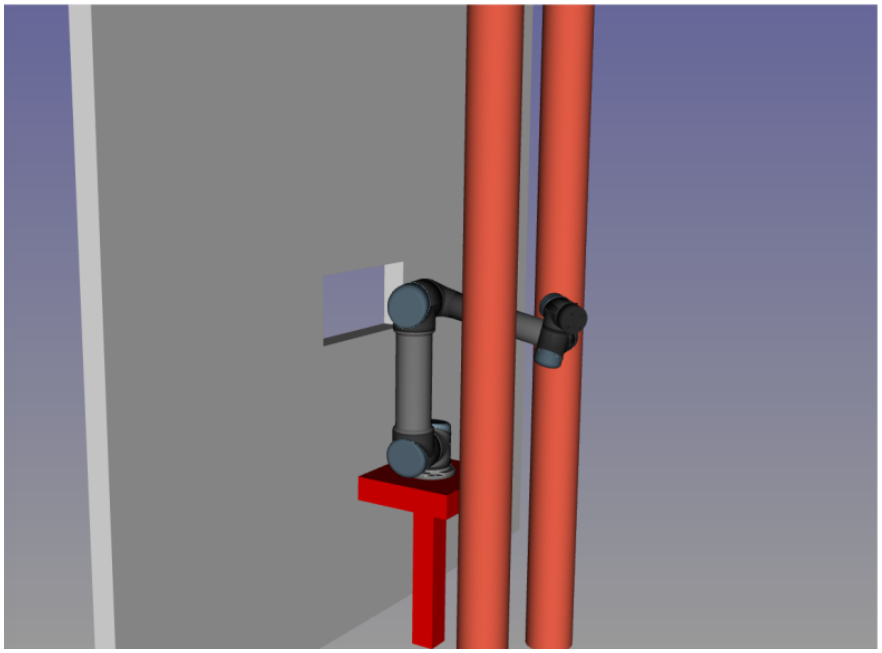
$$\begin{aligned} \mathcal{C}_{obst} = \{ & \mathbf{q} \in \mathcal{C}, \quad \exists i \in \{1, \cdots, m\}, \ \exists M \in \mathcal{B}_i, \ \mathbf{x}_i(M, \mathbf{q}) \in \mathcal{O} \text{ or} \\ & \exists i, j \in \{1, \cdots, m\}, \ \exists M_i \in \mathcal{B}_i, \ \exists M_j \in \mathcal{B}_j, \\ & \mathbf{x}_i(M_i, \mathbf{q}) = x_j(M_j, \mathbf{q}) \} \end{aligned}$$

- Free configuration space : $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obst}$.

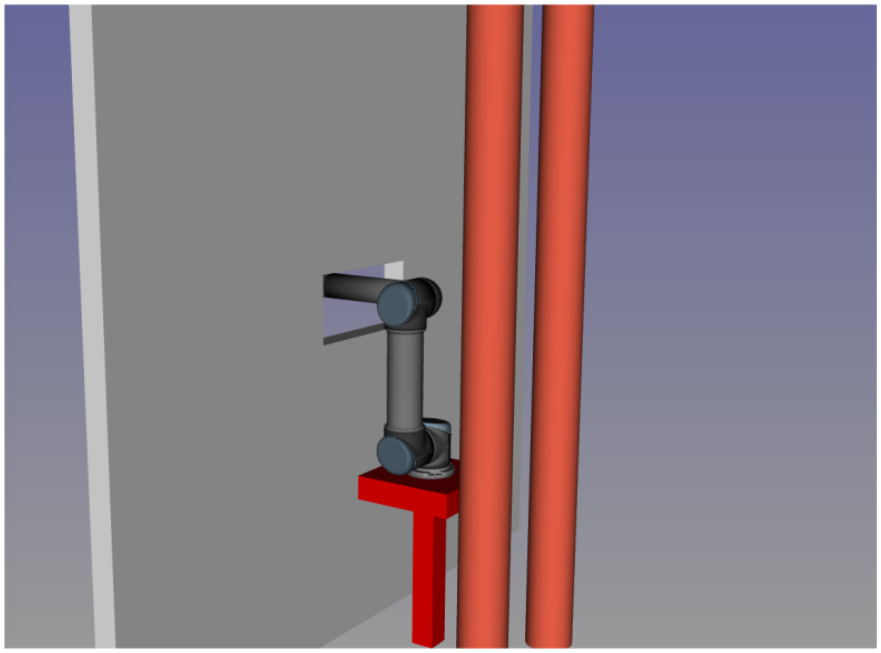
Motion

- Configuration space :
  - differential manifold
- Motion :
  - continuous function from $[0, 1]$ to $\mathcal{C}$.
- Collision-free motion :
  - continuous function from $[0, 1]$ to $\mathcal{C}_{free}$.

Motion planning problem



initial configuration



goal configuration

$$\mathcal{C} = \mathbb{R}^6$$

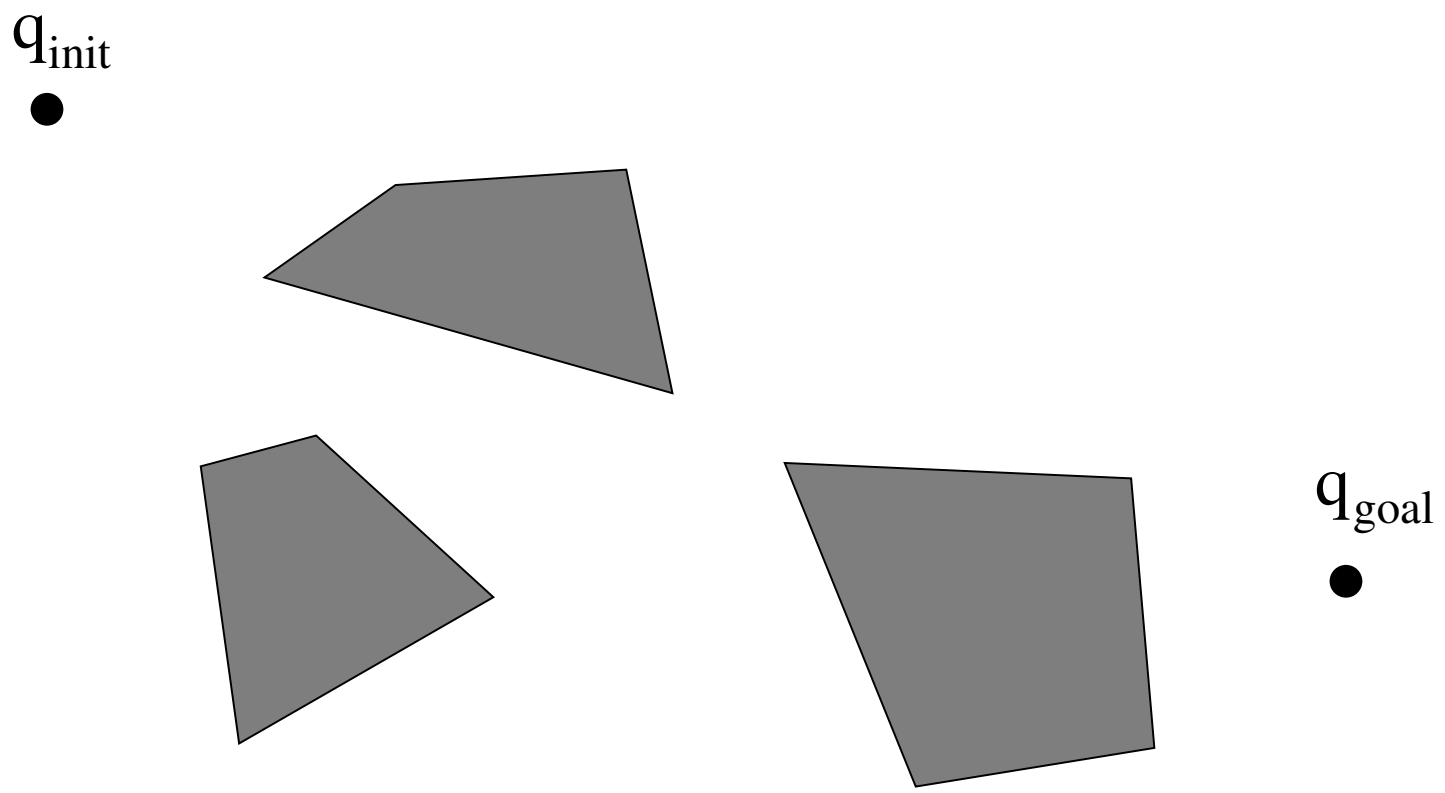
History

- Before the 1990's : mainly a mathematical problem
  - real algebraic geometry,
  - decidability : Schwartz and Sharir 1982,
    - Tarski theorem, Collins decomposition,
- from the 1990's : an algorithmic problem
  - random sampling (1993),
  - asymptotically optimal random sampling (2011).

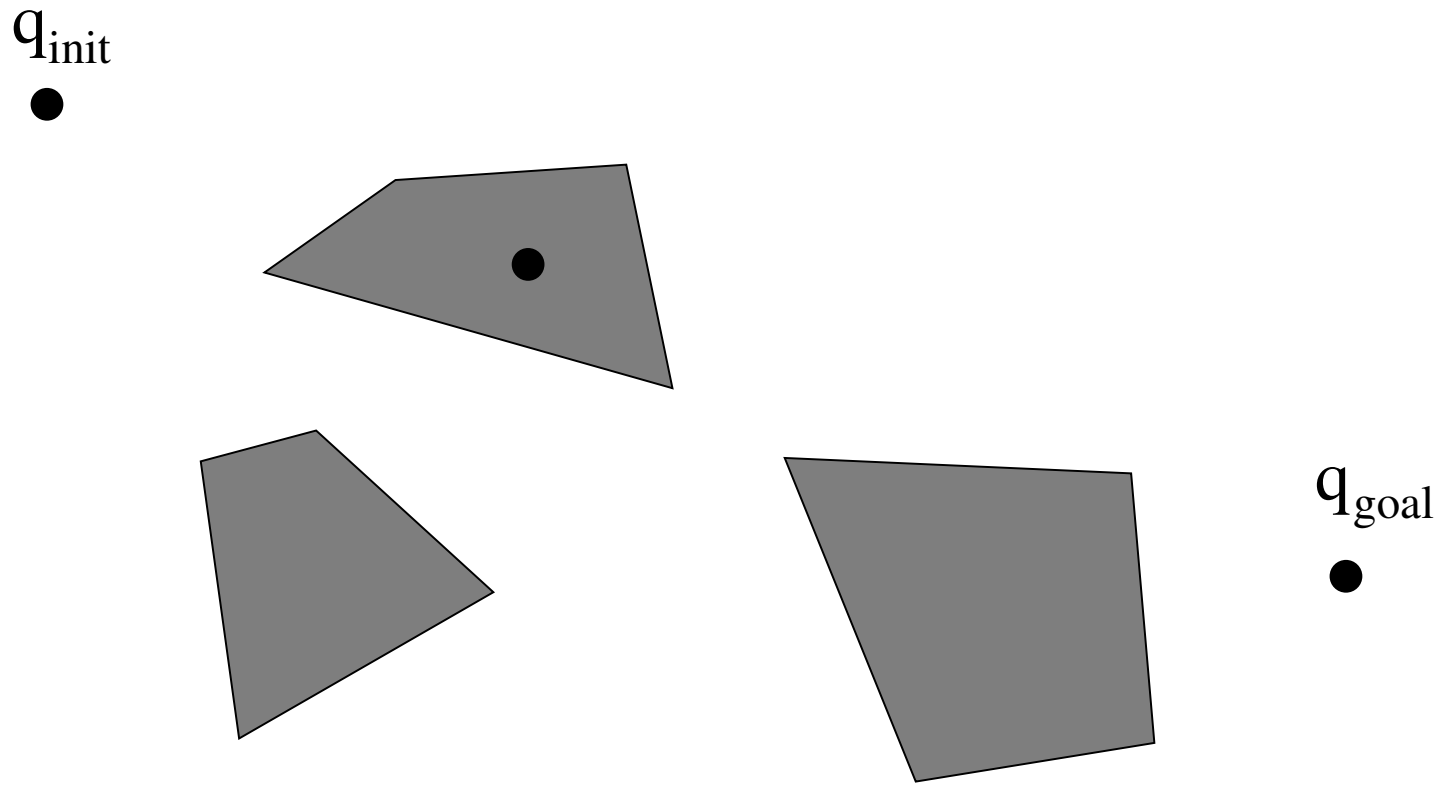
Random methods

- In the early 1990's, random methods started being developed
- Principle
  - shoot random configurations
  - test whether they are in collision
  - build a graph (roadmap) the nodes of which are free configurations
  - and the edges of which are collision-free linear interpolations

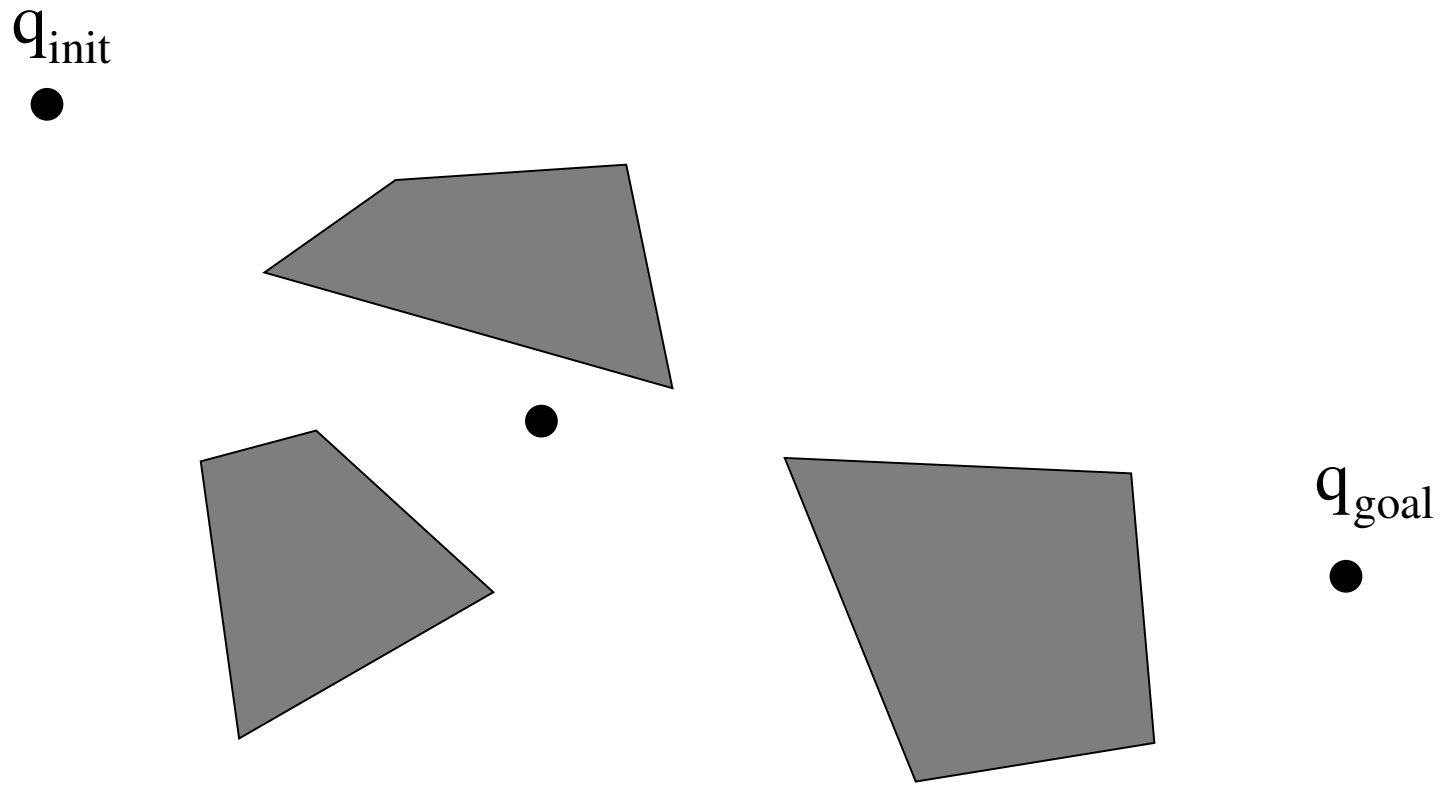
Probabilistic roadmap (PRM) 1994



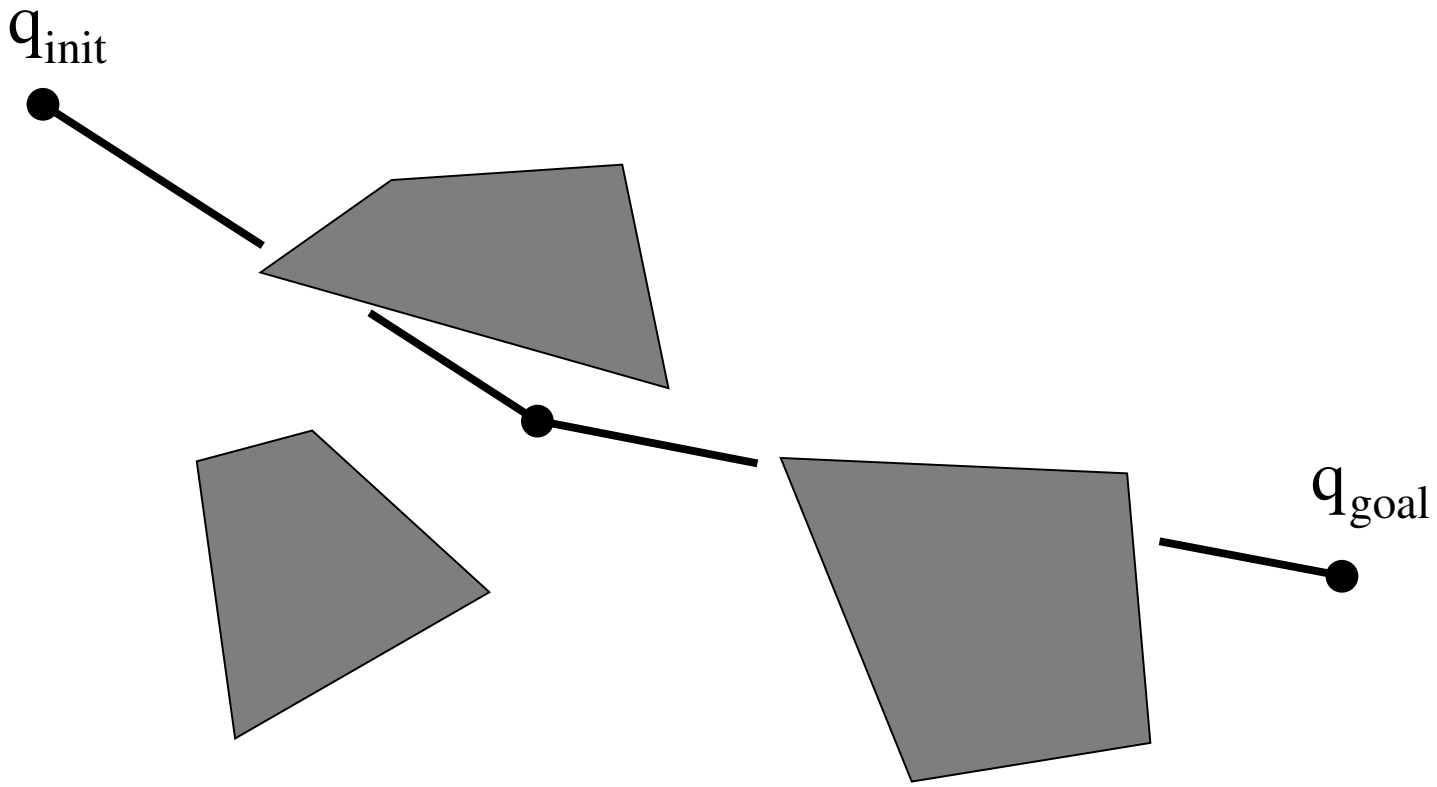
Probabilistic roadmap (PRM) 1994



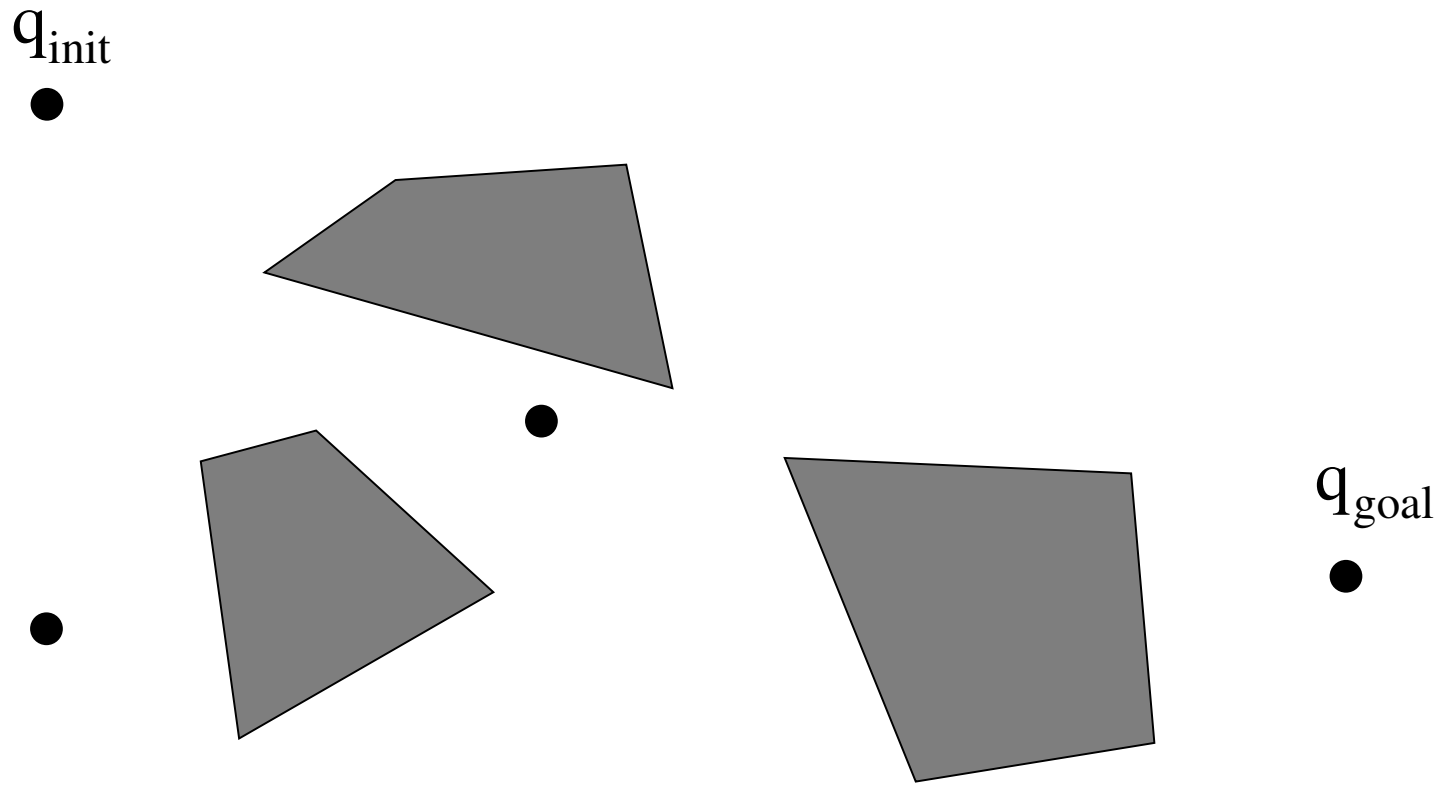
Probabilistic roadmap (PRM) 1994



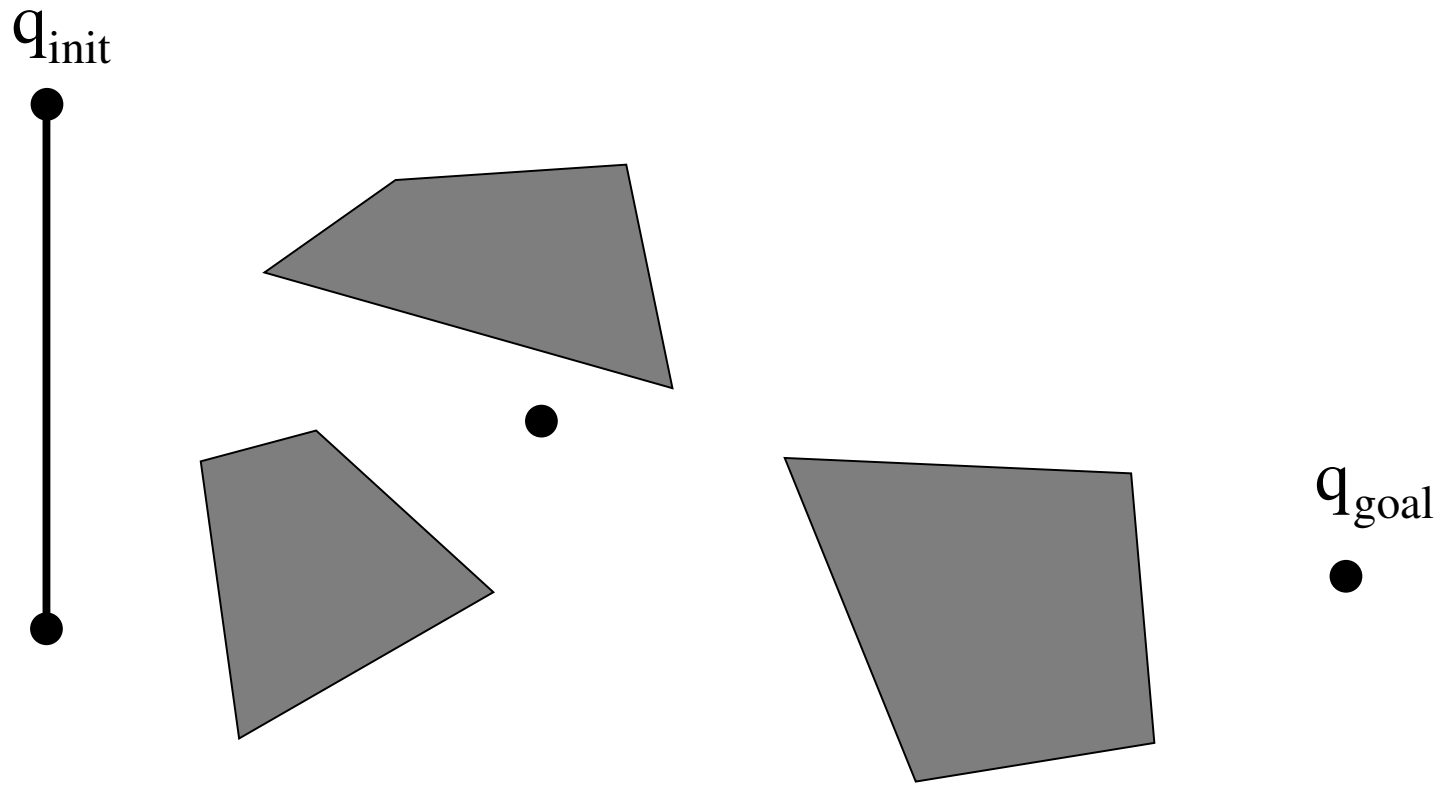
Probabilistic roadmap (PRM) 1994



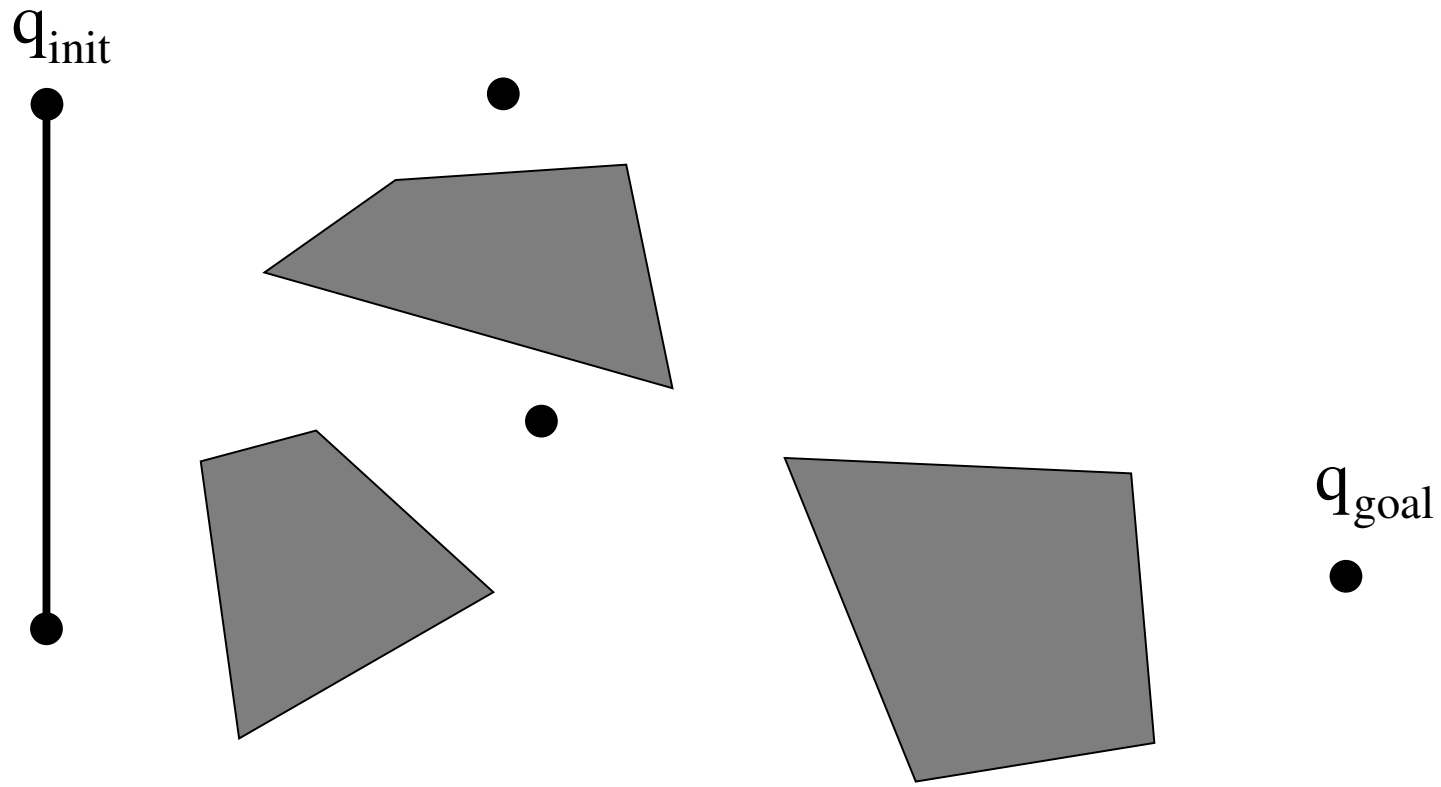
Probabilistic roadmap (PRM) 1994



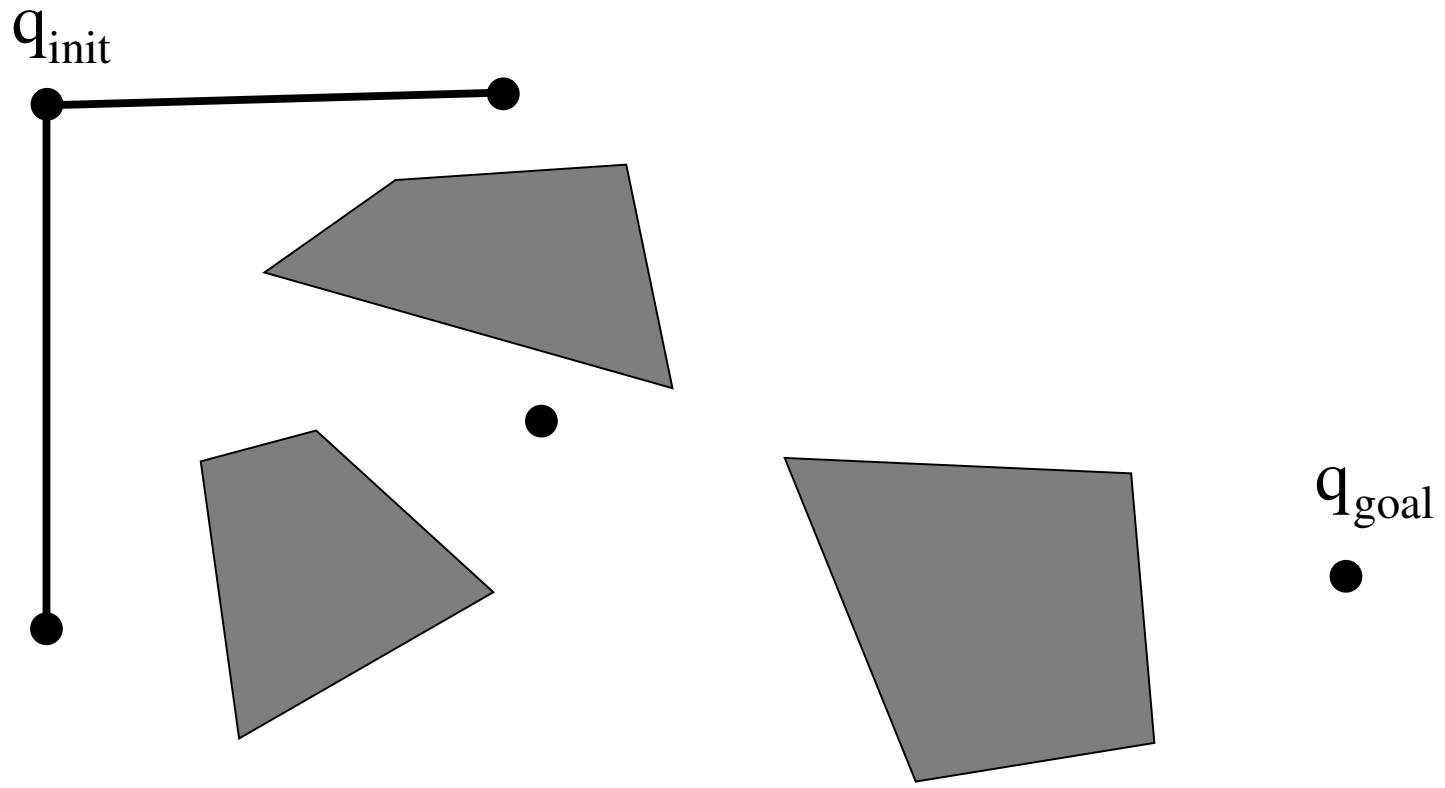
Probabilistic roadmap (PRM) 1994



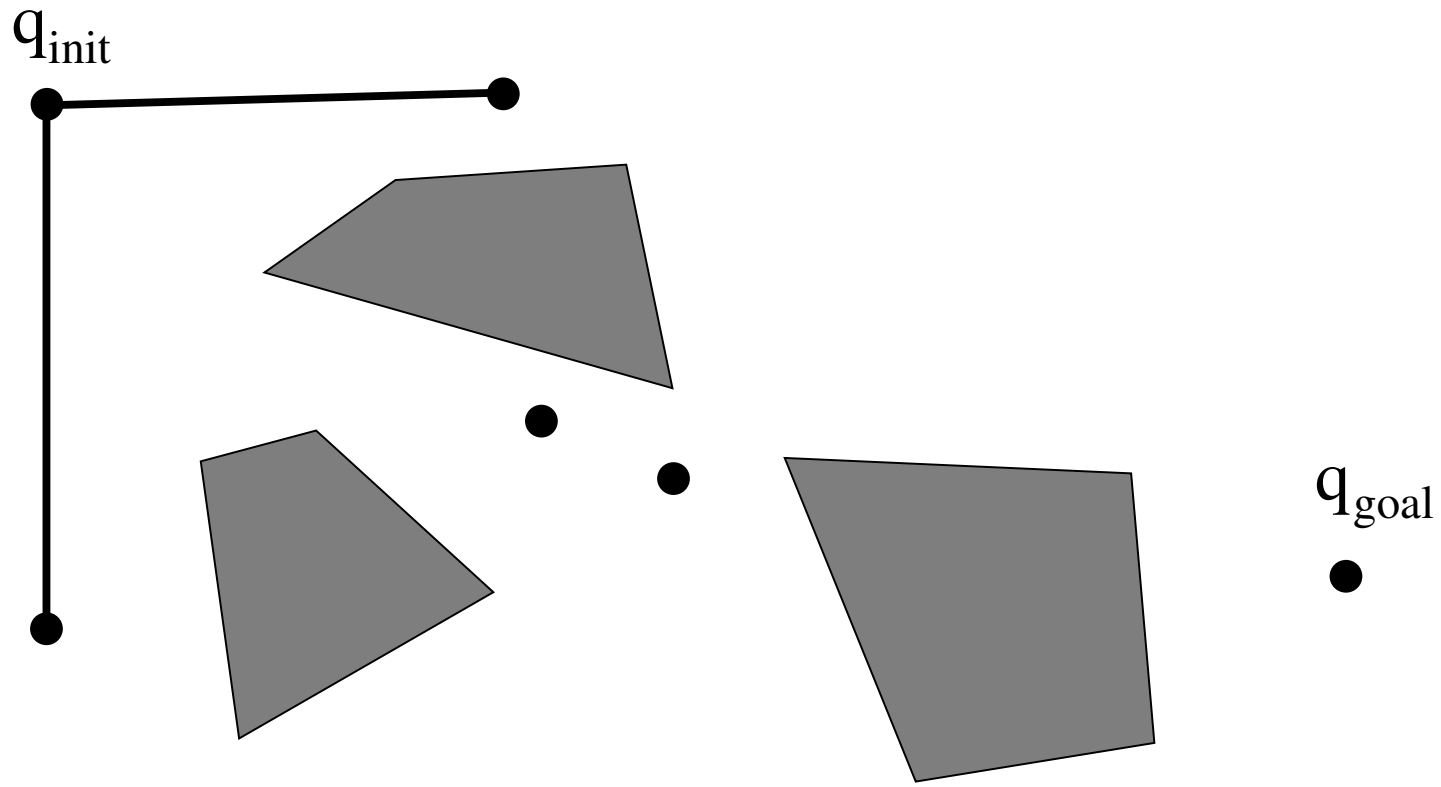
Probabilistic roadmap (PRM) 1994



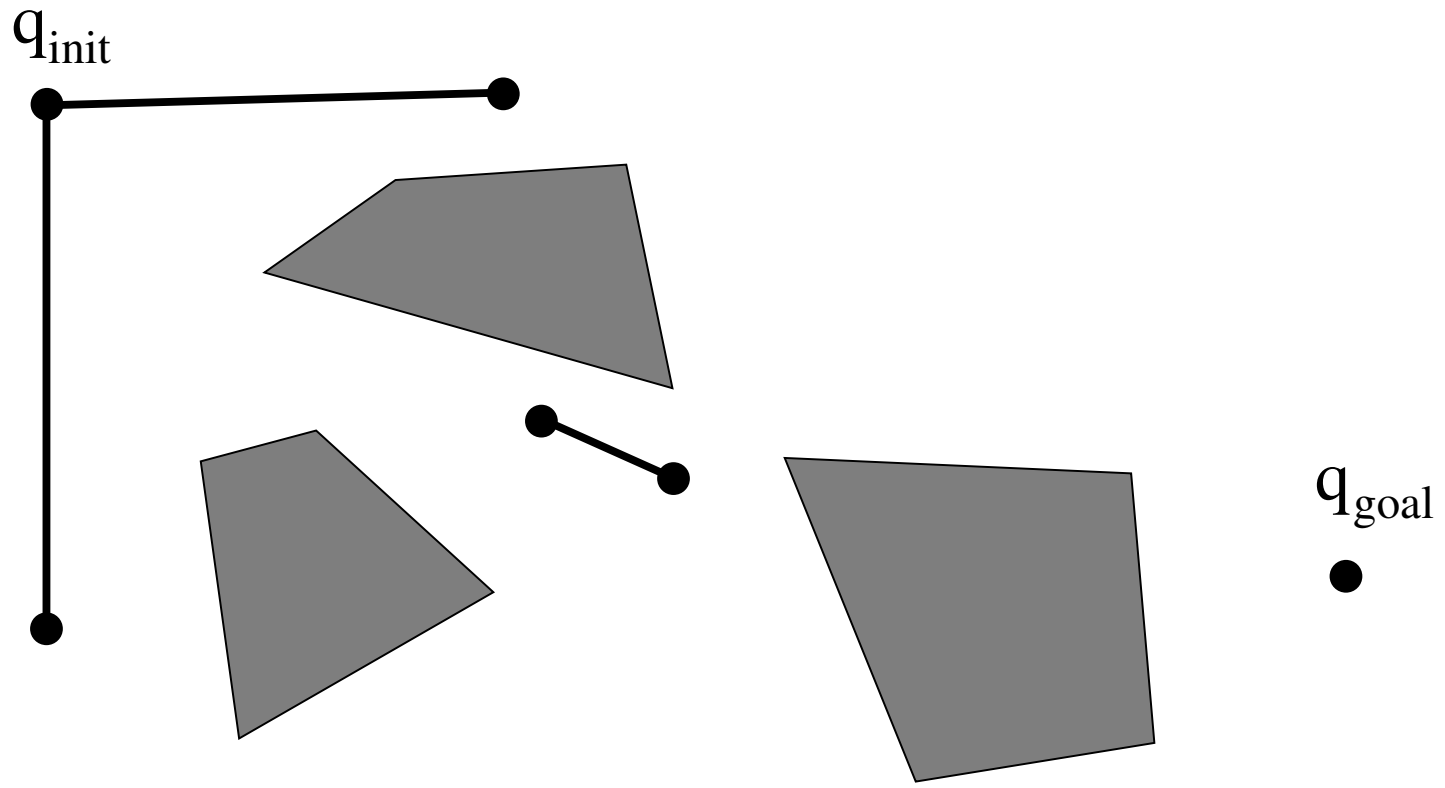
Probabilistic roadmap (PRM) 1994



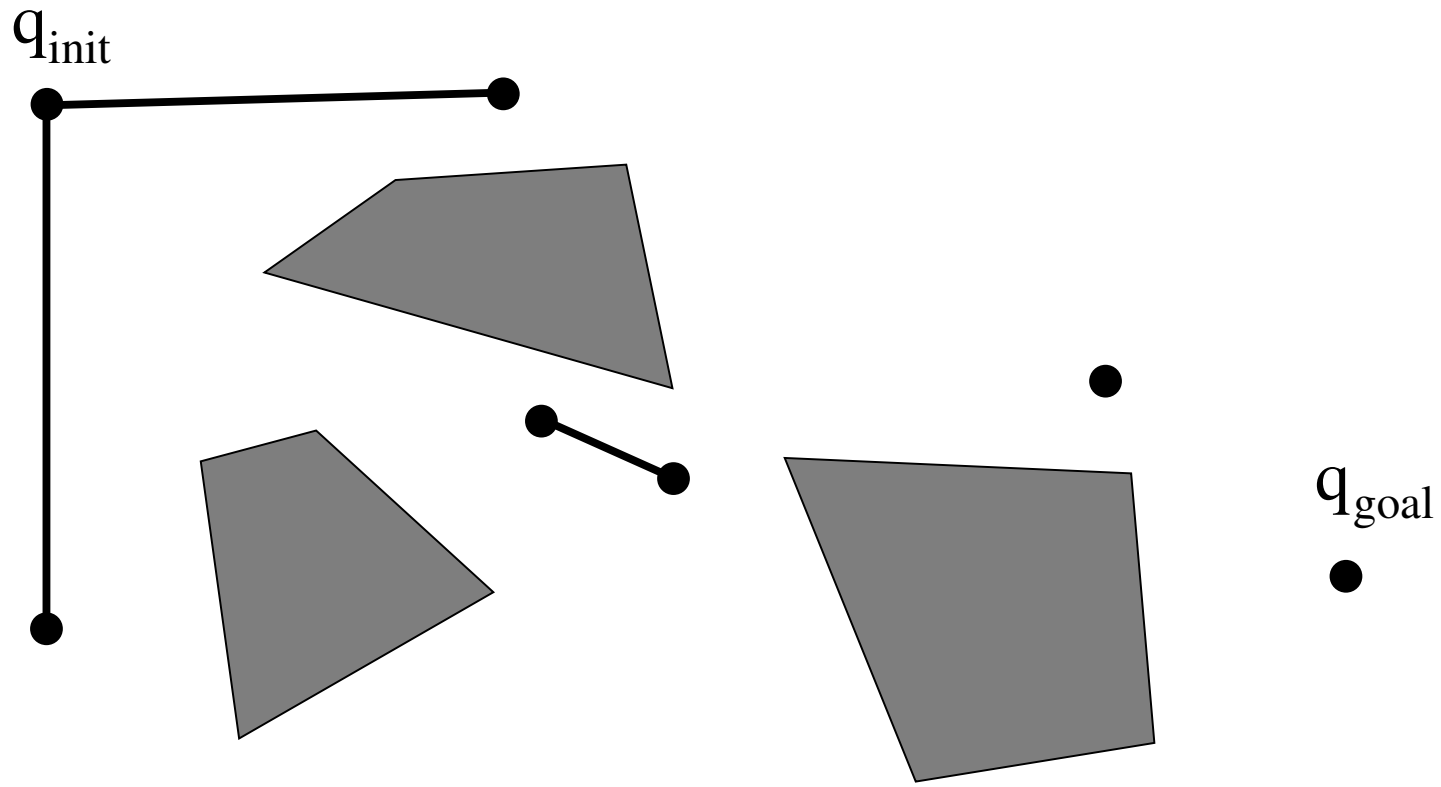
Probabilistic roadmap (PRM) 1994



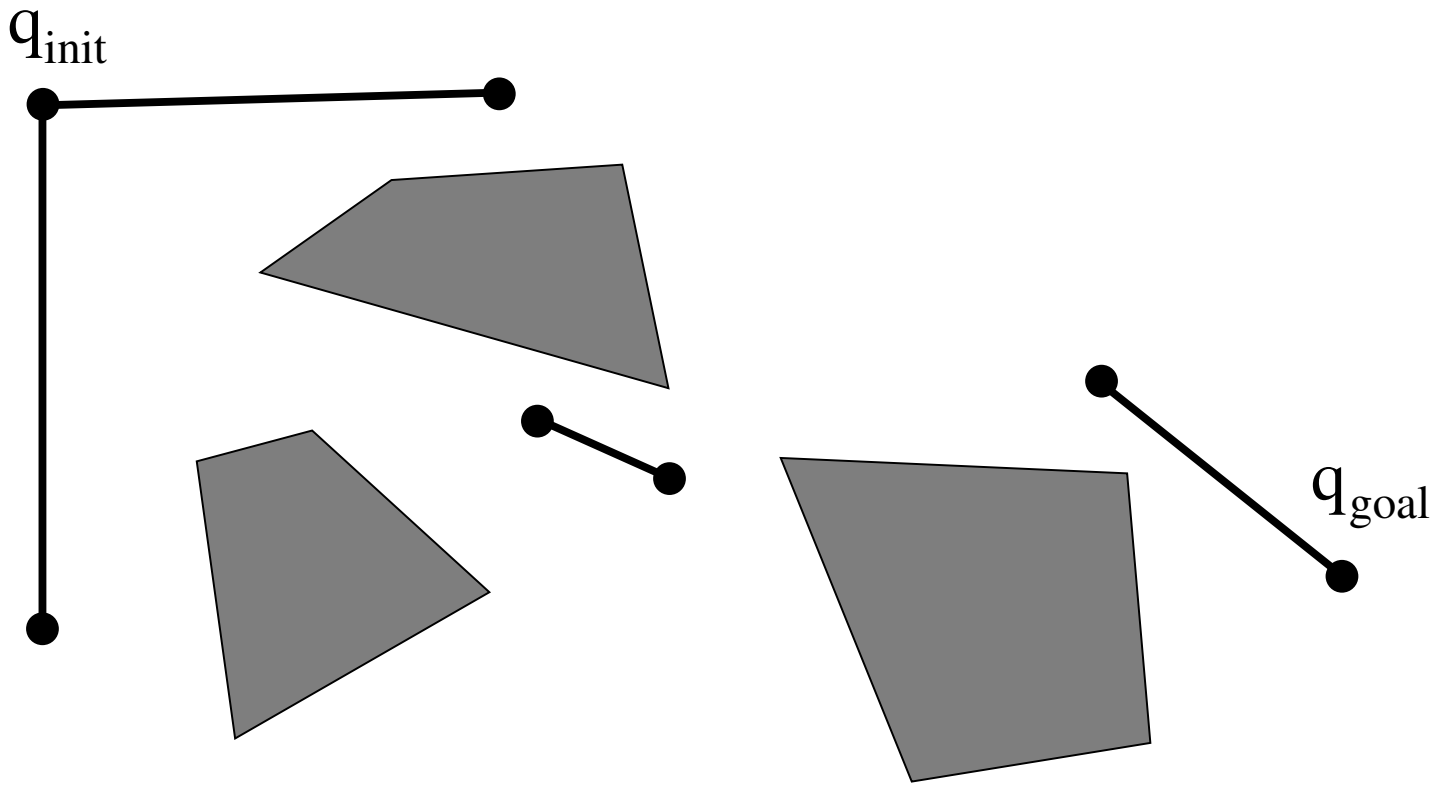
Probabilistic roadmap (PRM) 1994



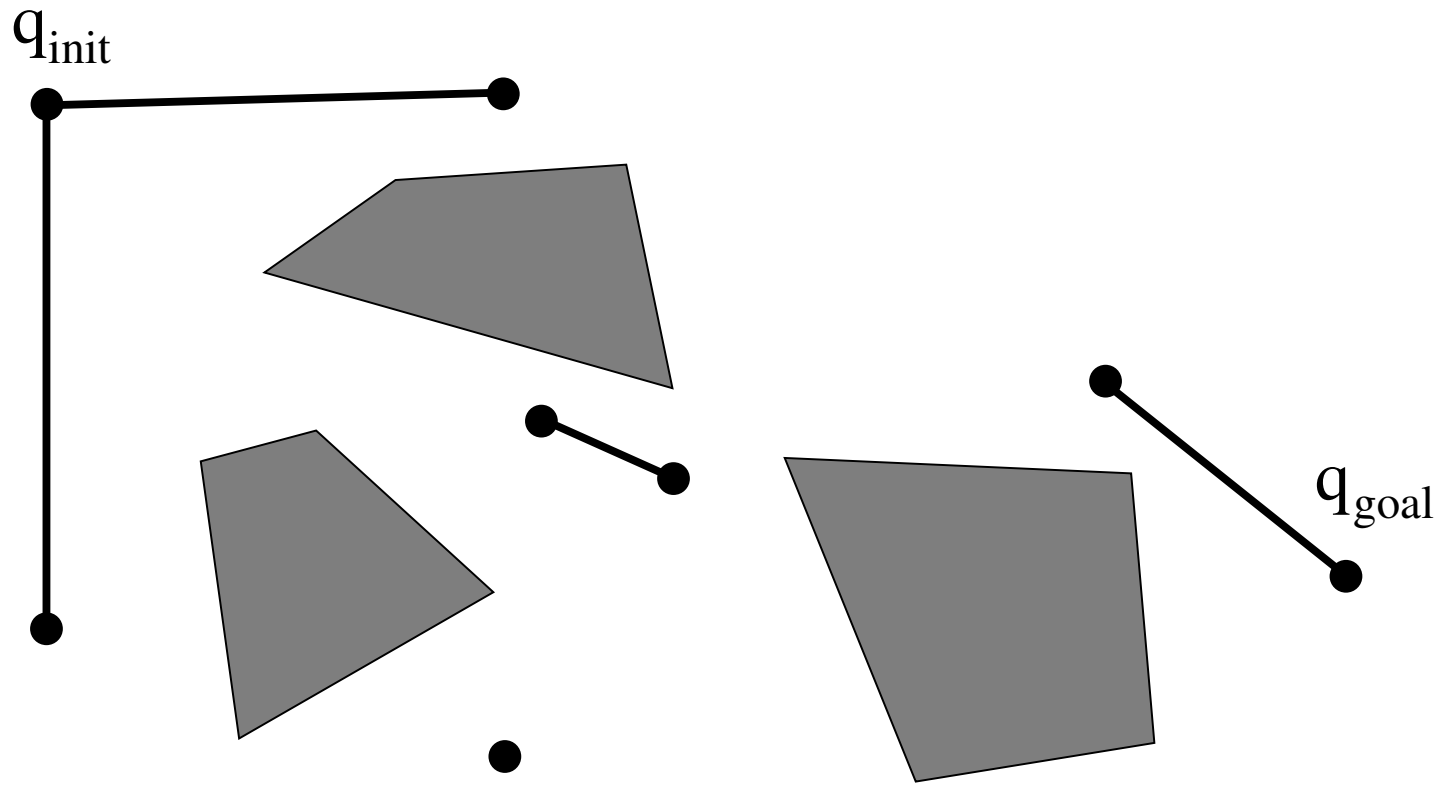
Probabilistic roadmap (PRM) 1994



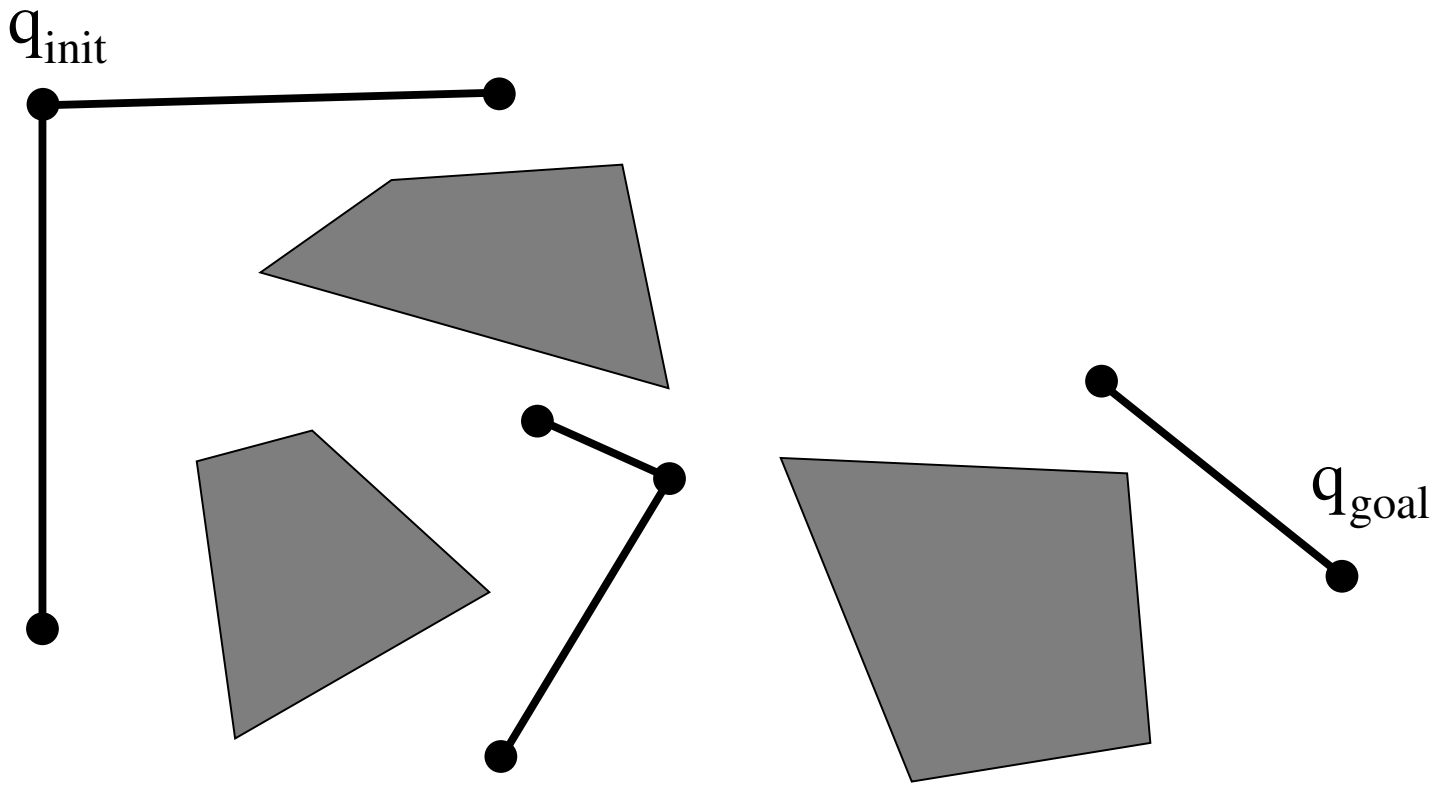
Probabilistic roadmap (PRM) 1994



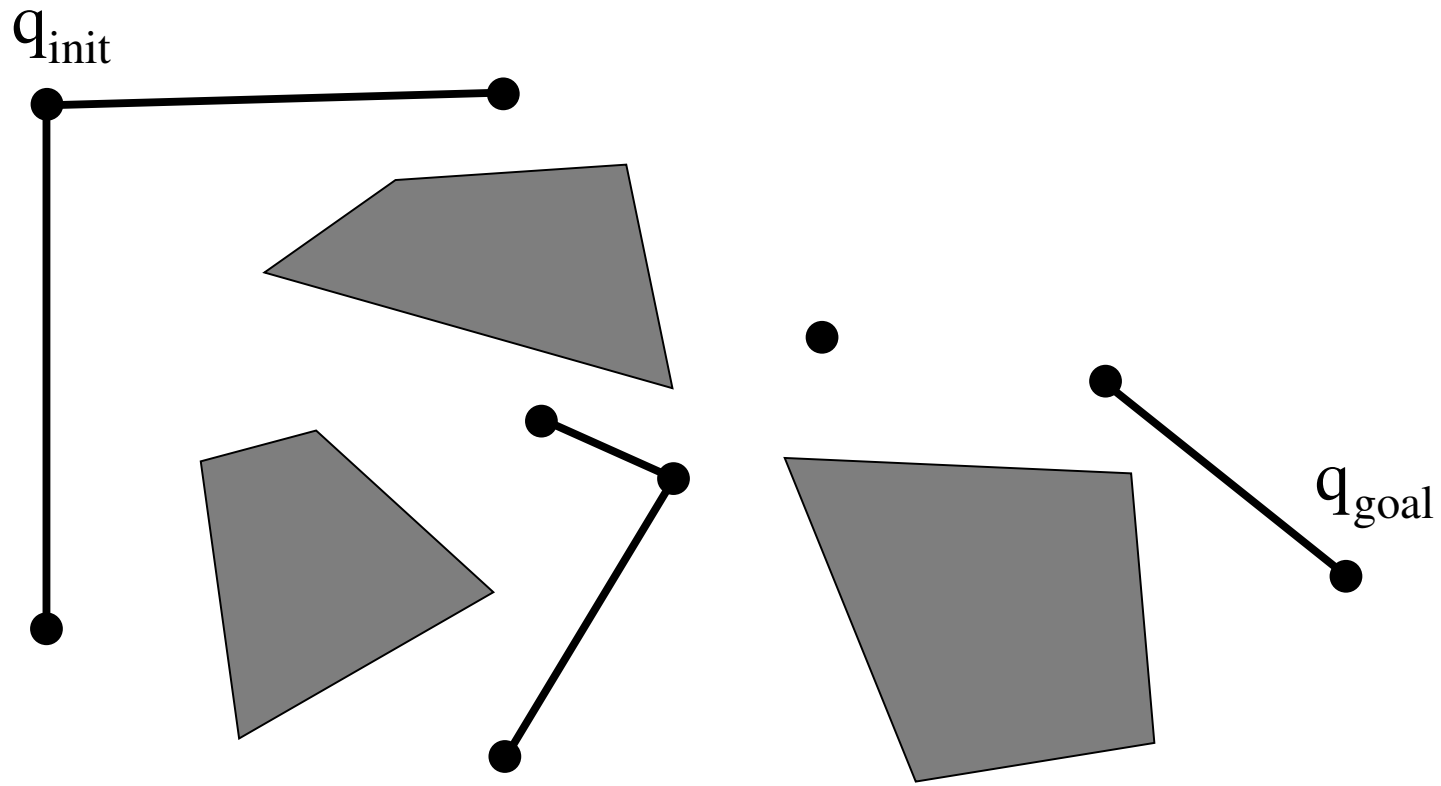
Probabilistic roadmap (PRM) 1994



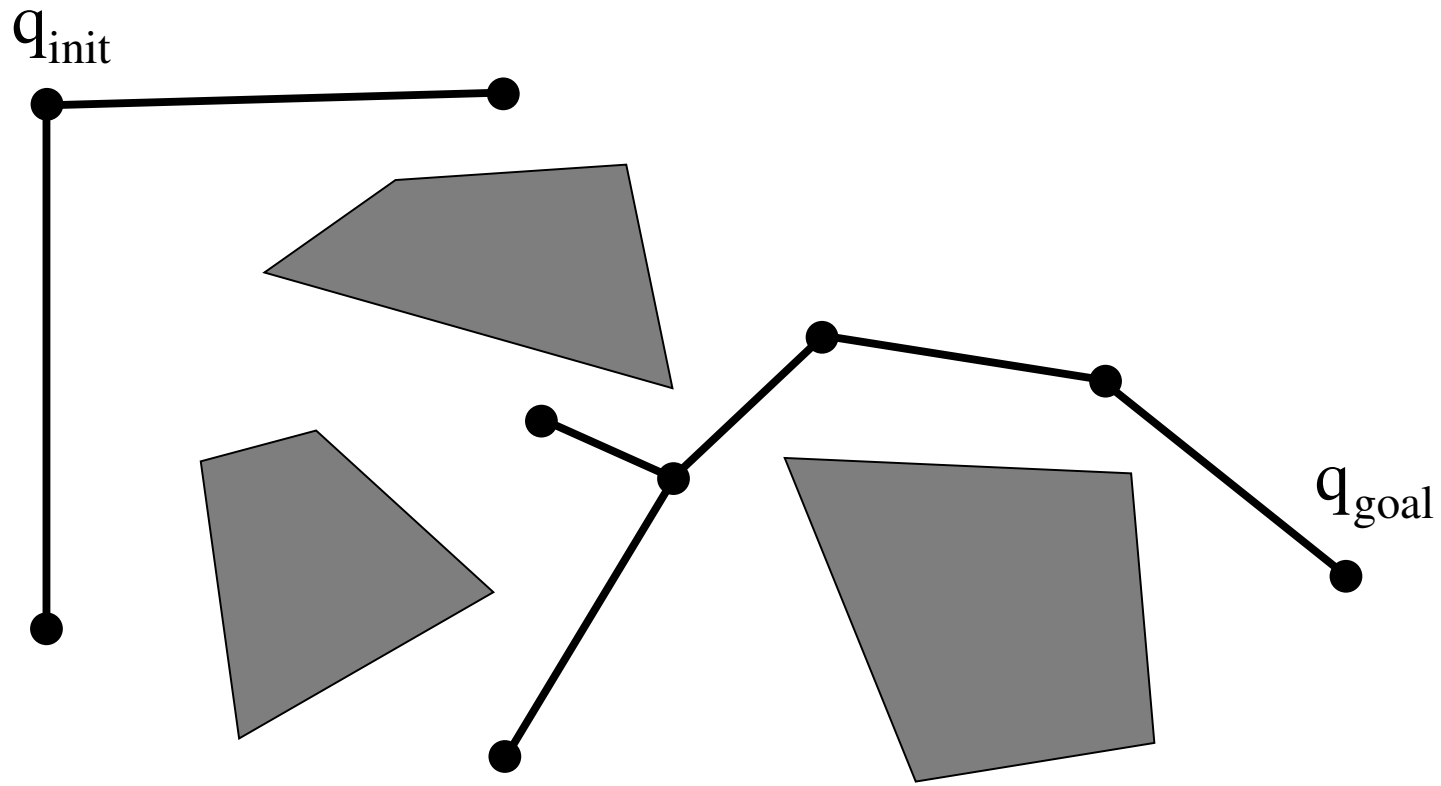
Probabilistic roadmap (PRM) 1994



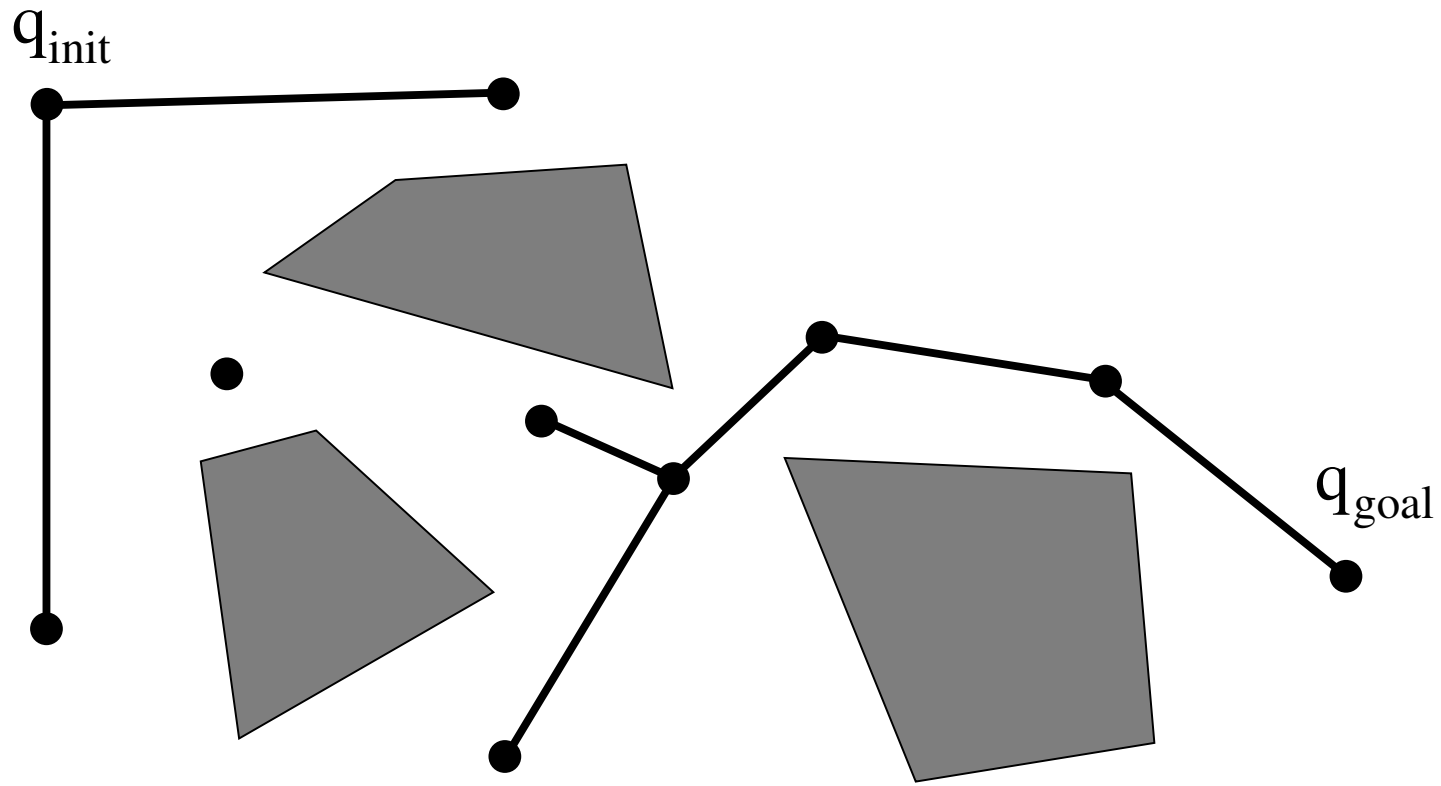
Probabilistic roadmap (PRM) 1994



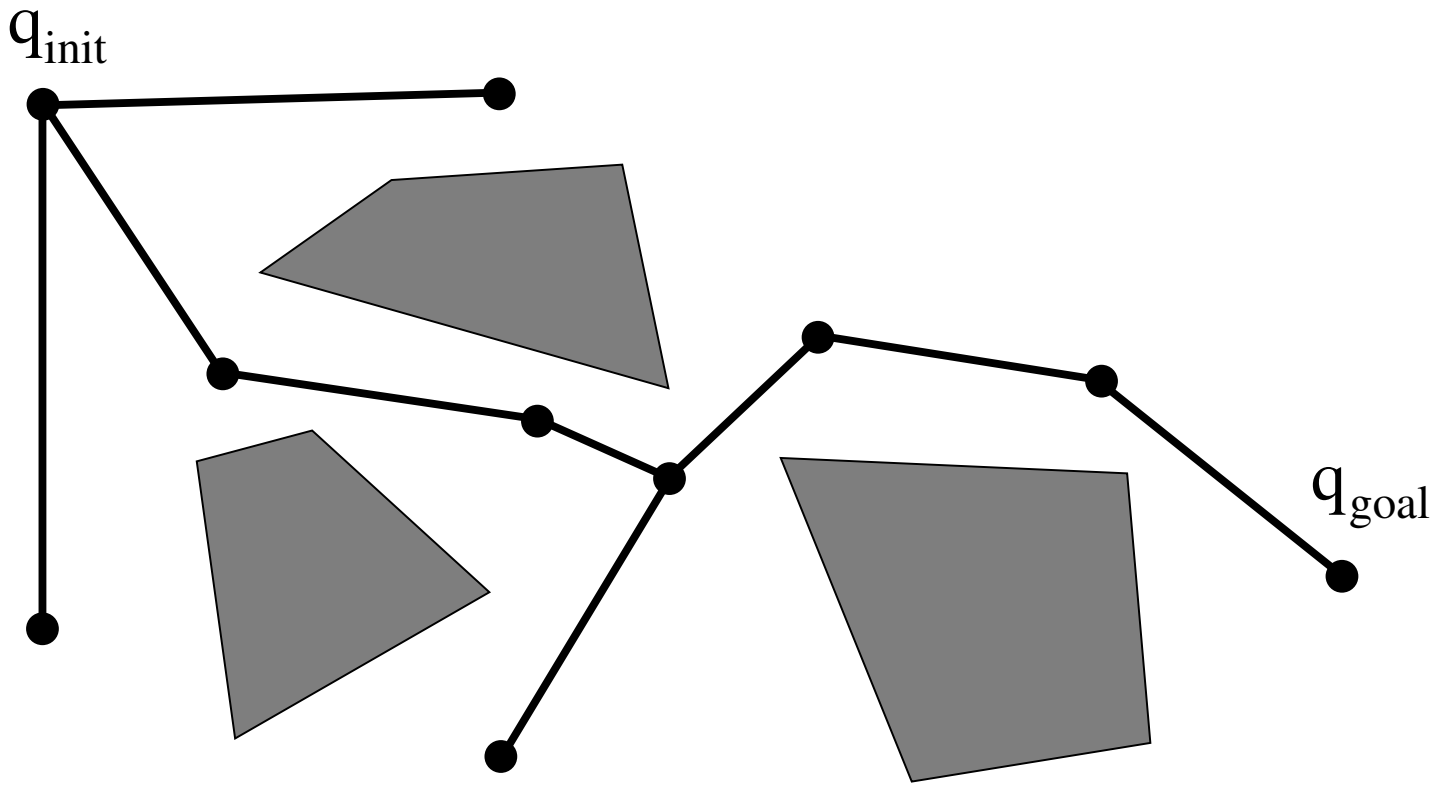
Probabilistic roadmap (PRM) 1994



Probabilistic roadmap (PRM) 1994



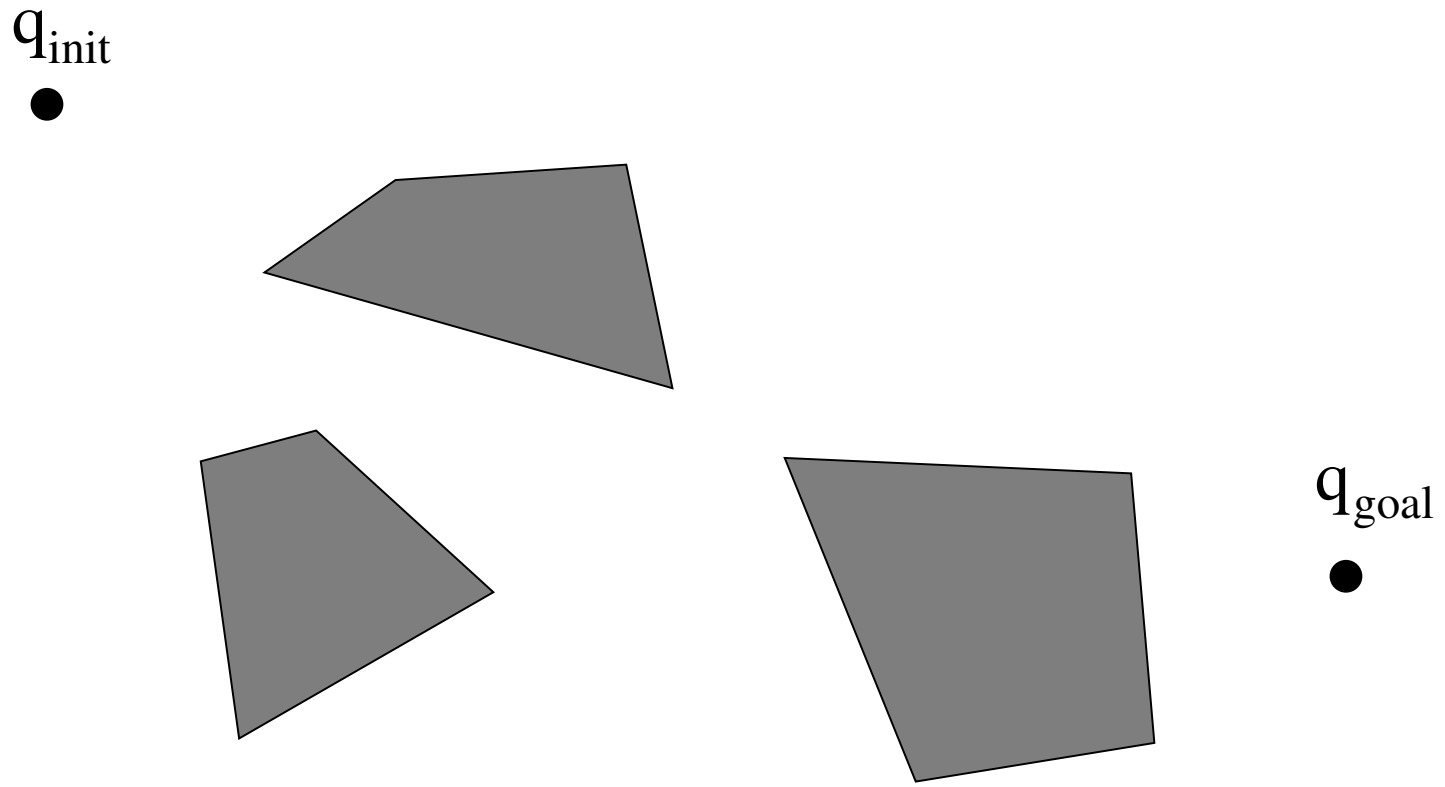
Probabilistic roadmap (PRM) 1994



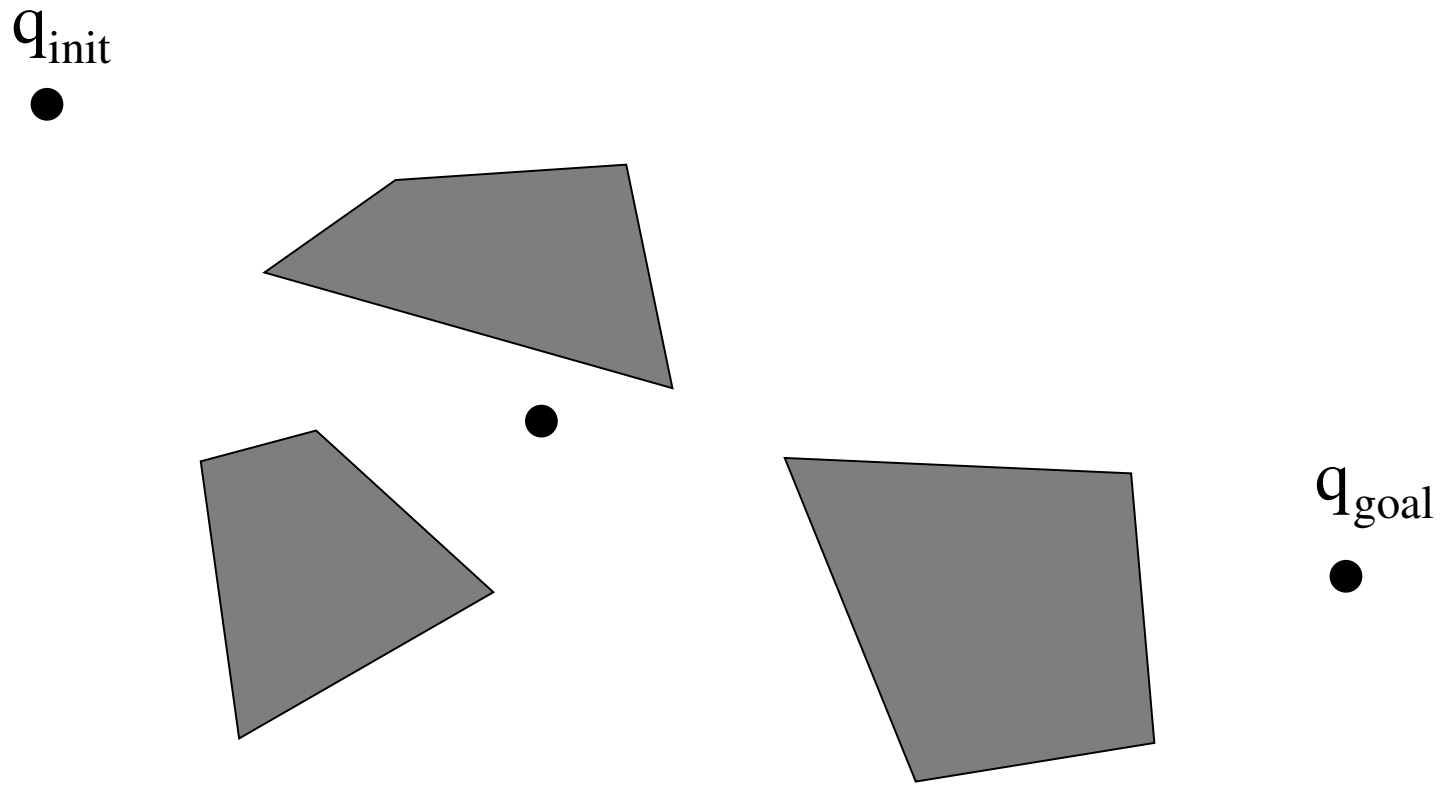
Probabilistic roadmap (PRM)

- A lot of useless nodes are created,
  - this increases the cost to connect new nodes to the existing roadmap
- Improvement : visibility-based PRM
  - Only *interesting* nodes are kept.

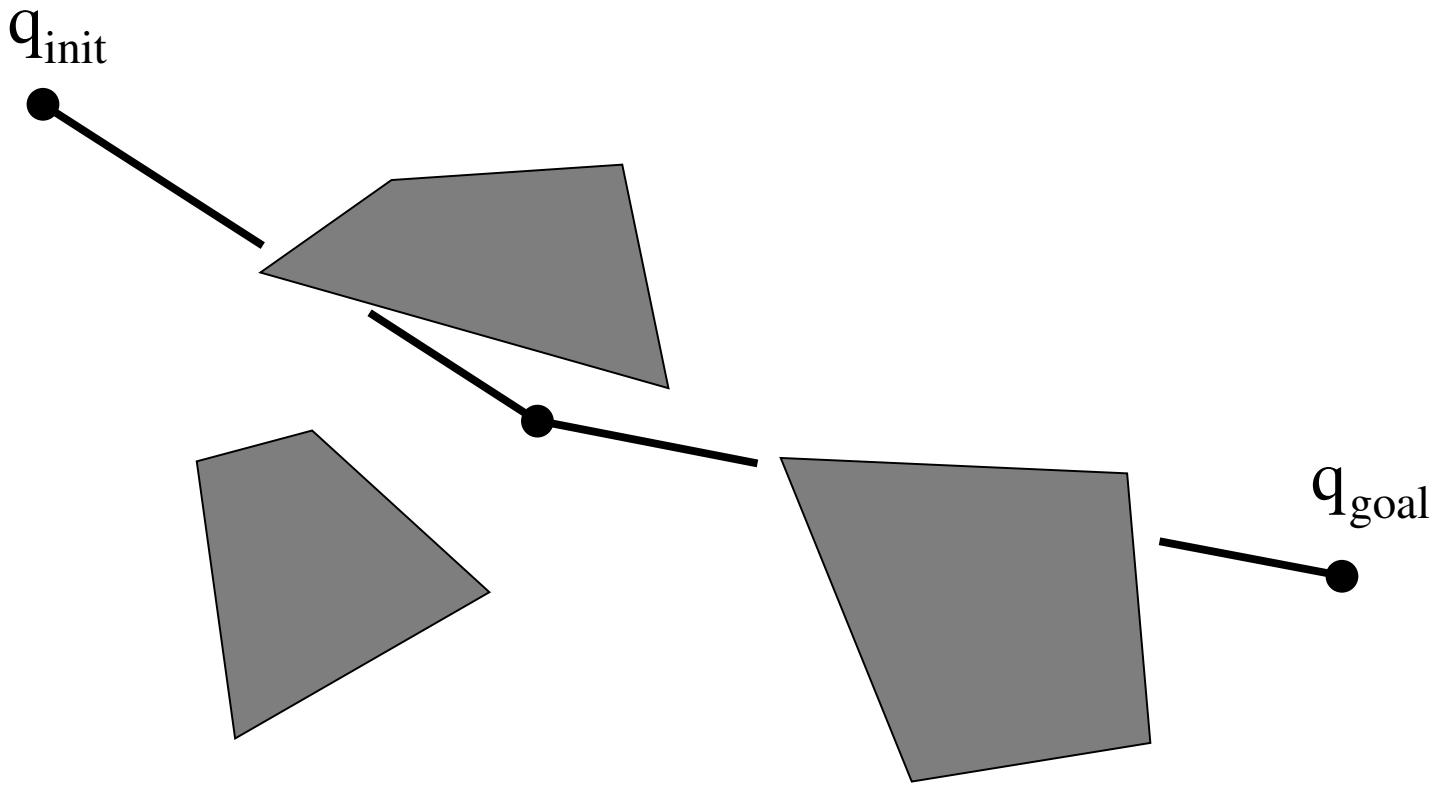
Visibility-based probabilistic roadmap (Visi-PRM) 1999



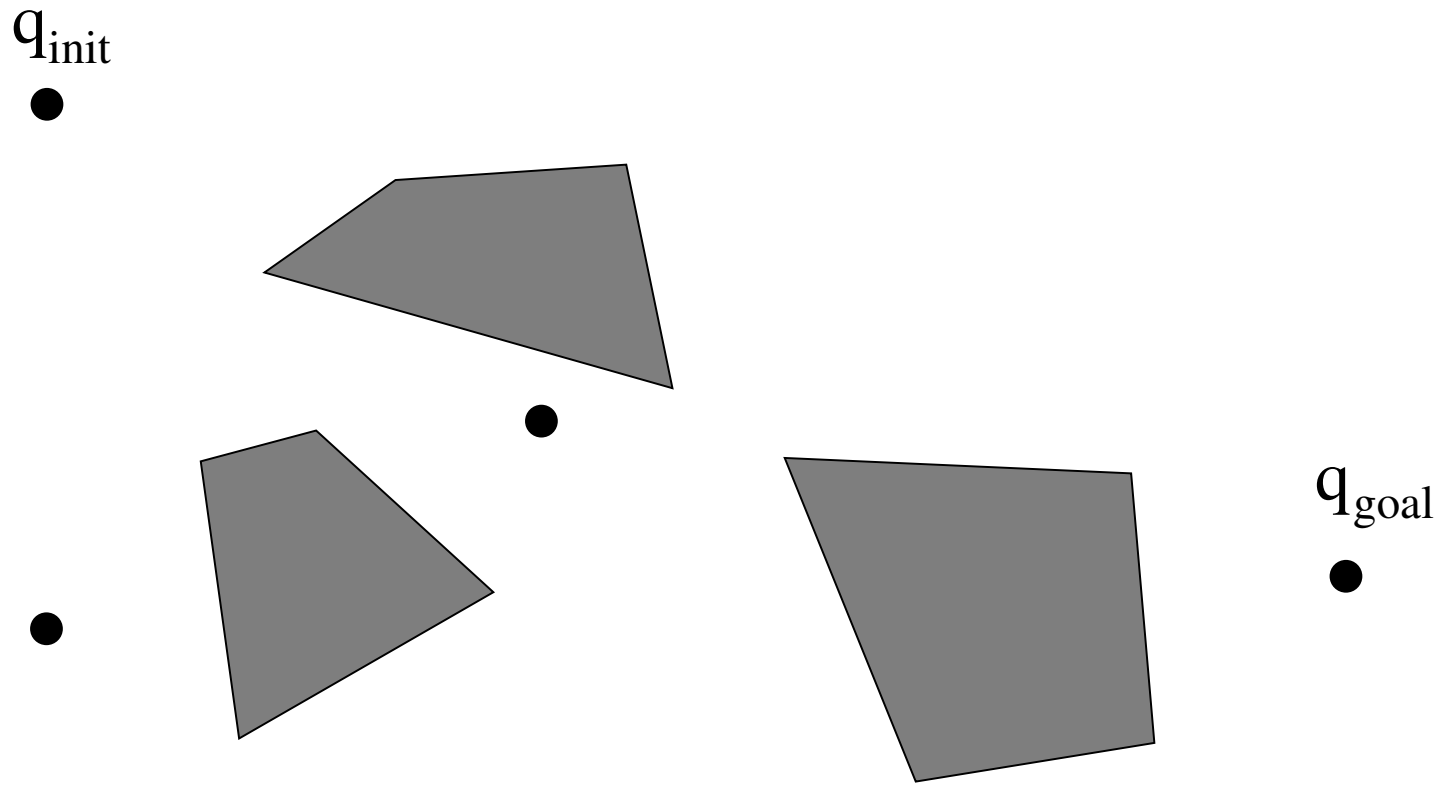
Visibility-based probabilistic roadmap (Visi-PRM) 1999



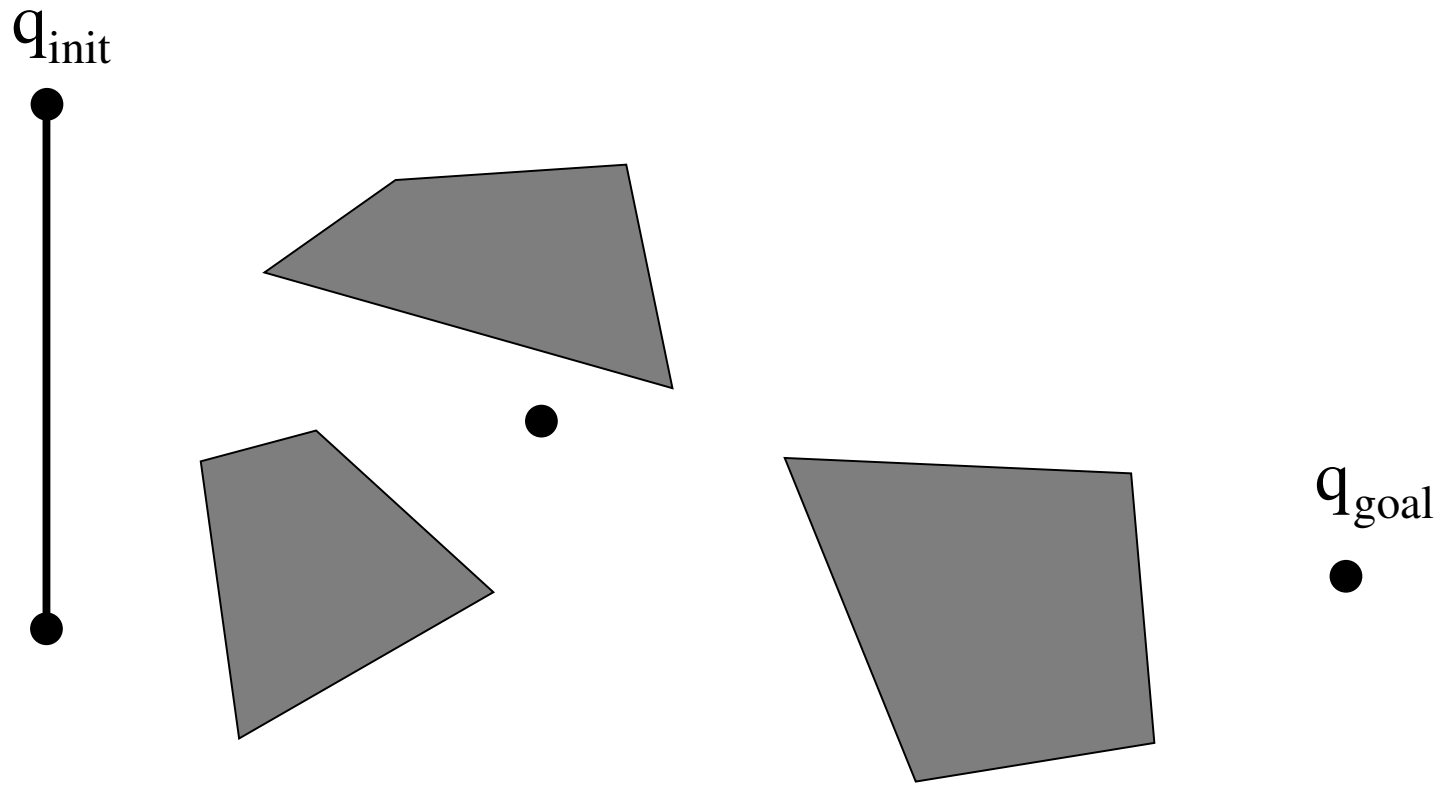
Visibility-based probabilistic roadmap (Visi-PRM) 1999



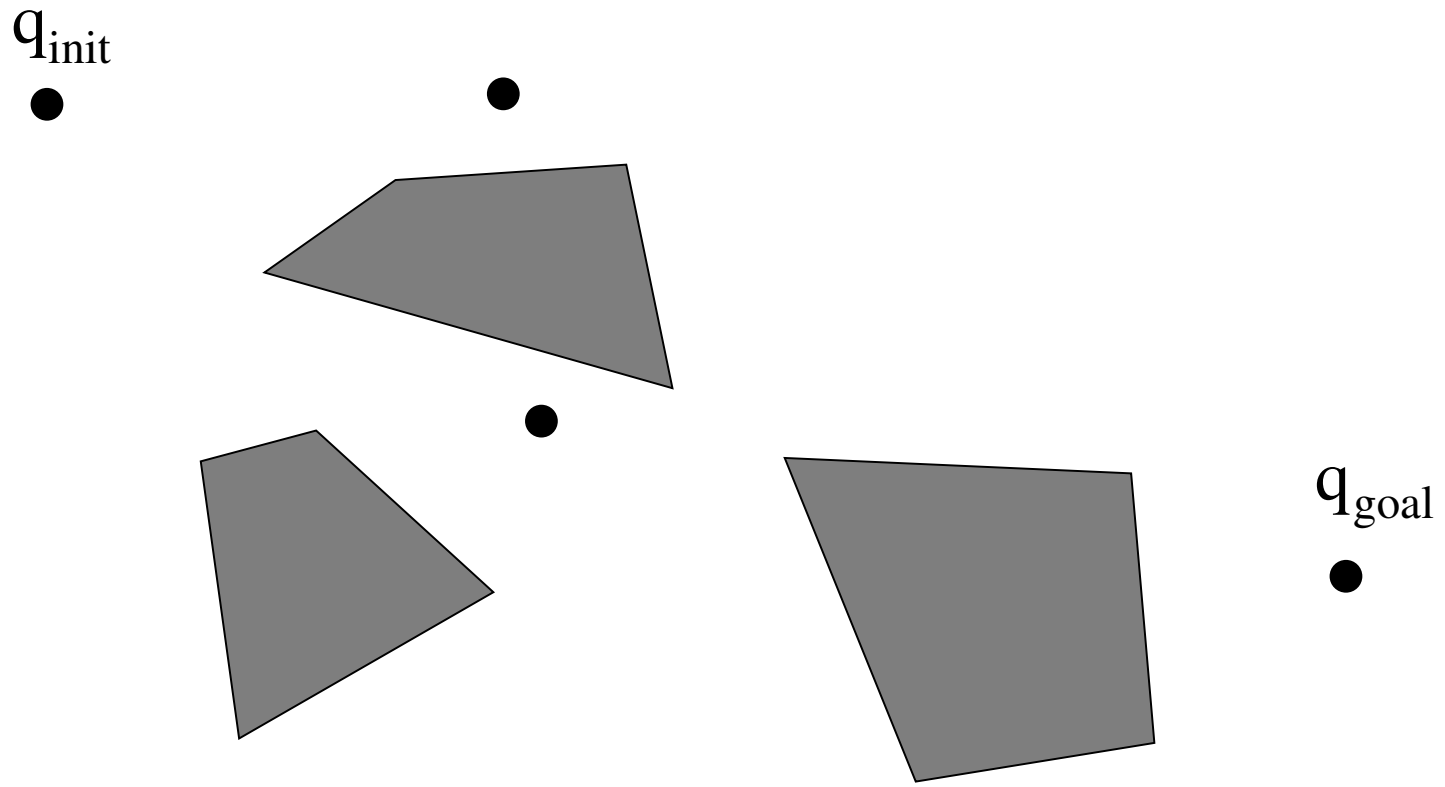
Visibility-based probabilistic roadmap (Visi-PRM) 1999



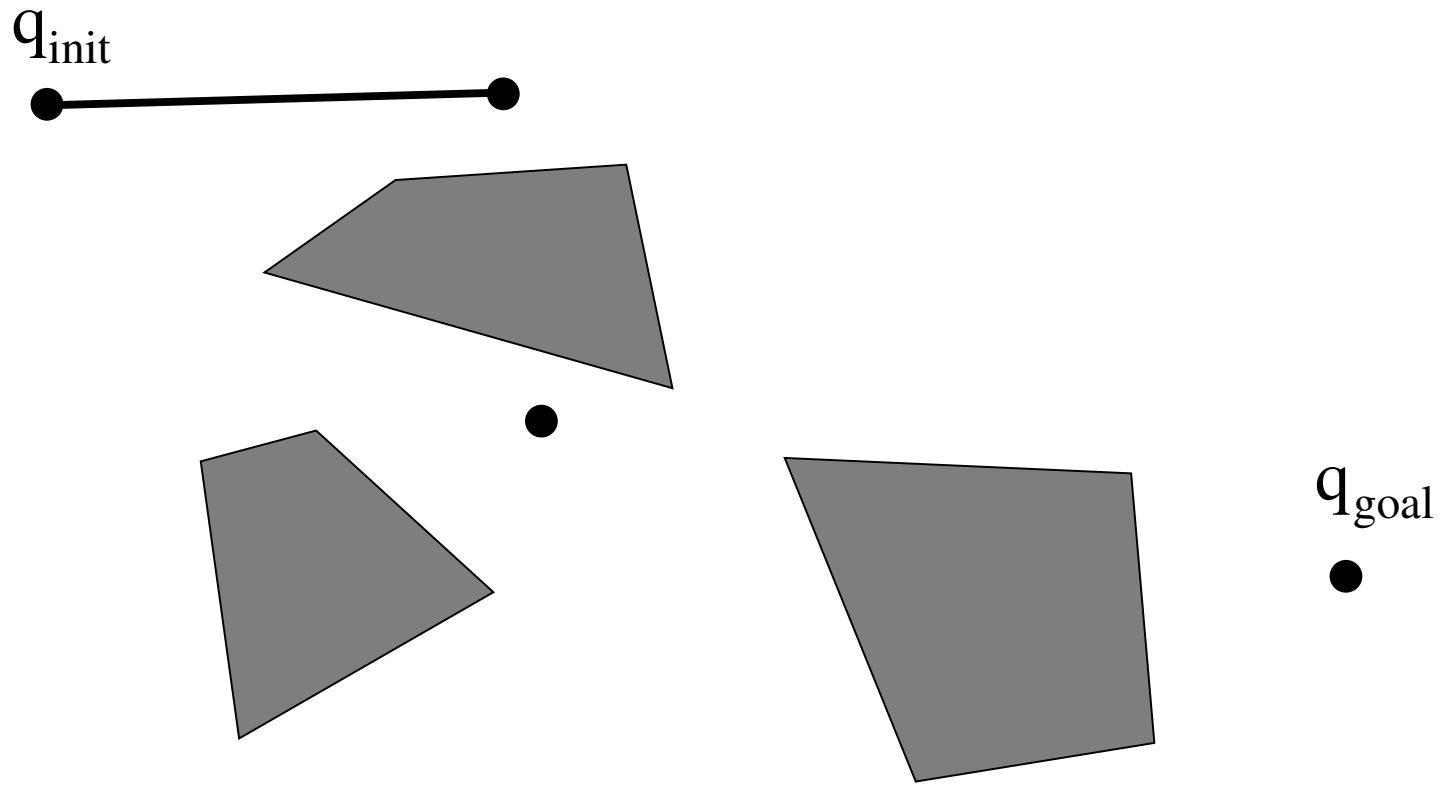
Visibility-based probabilistic roadmap (Visi-PRM) 1999



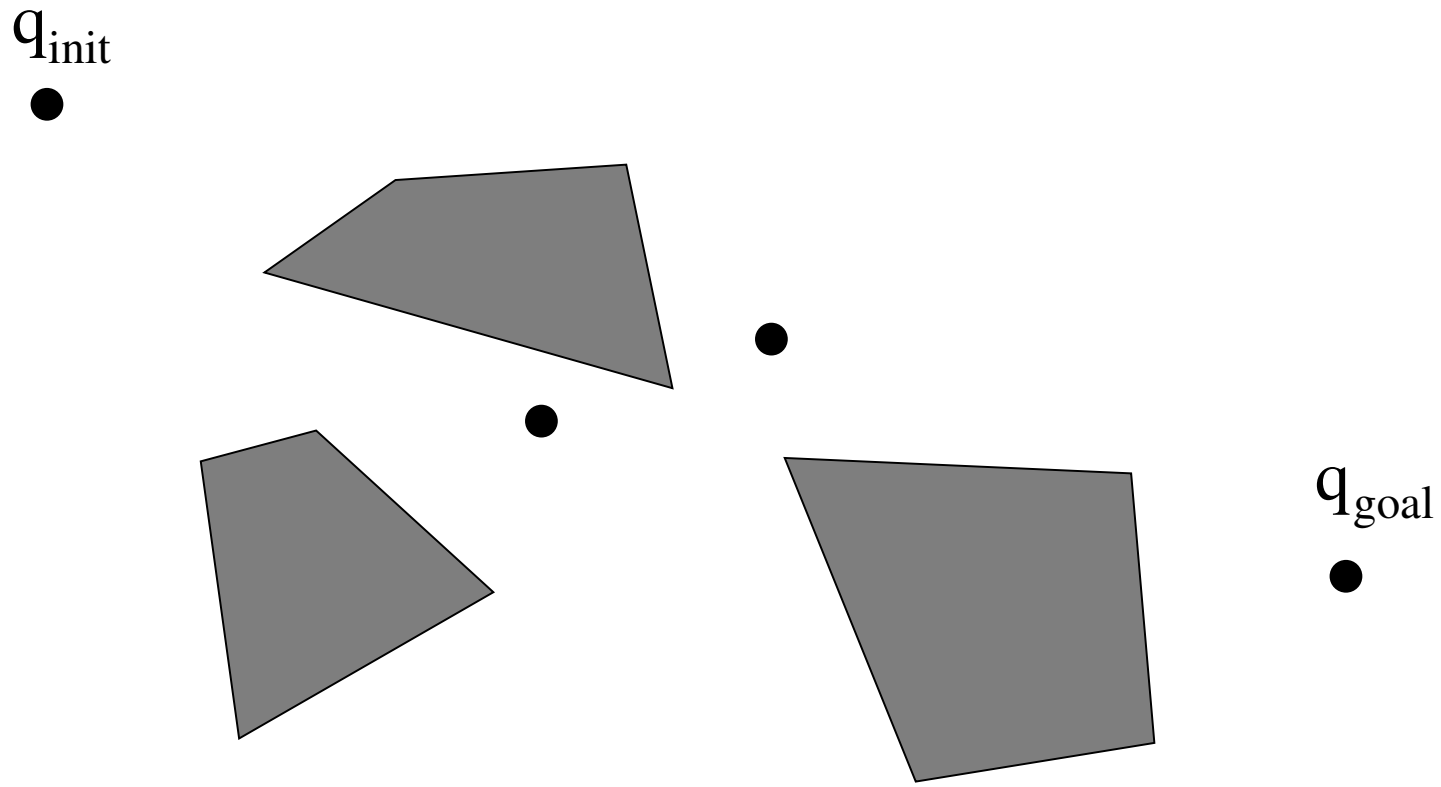
Visibility-based probabilistic roadmap (Visi-PRM) 1999



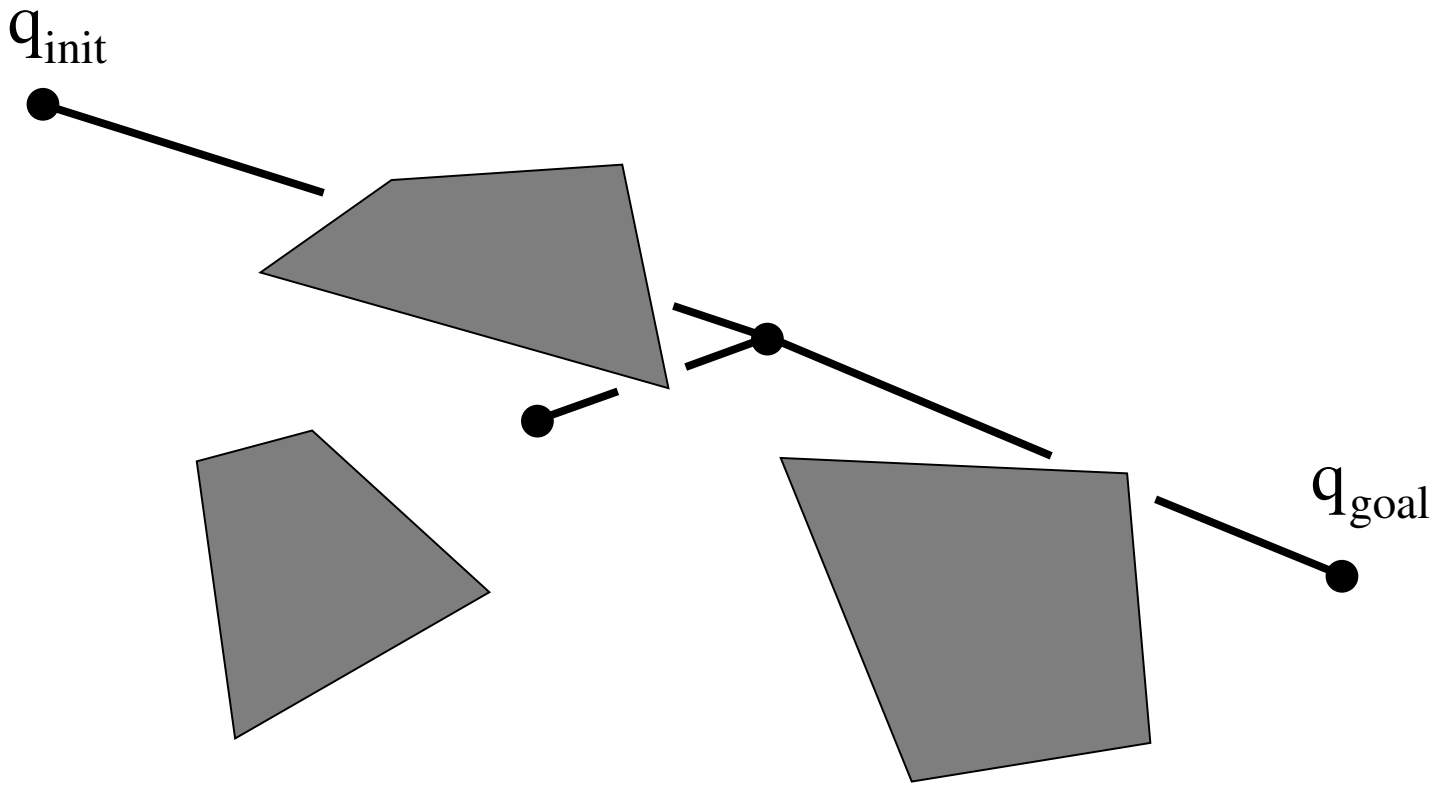
Visibility-based probabilistic roadmap (Visi-PRM) 1999



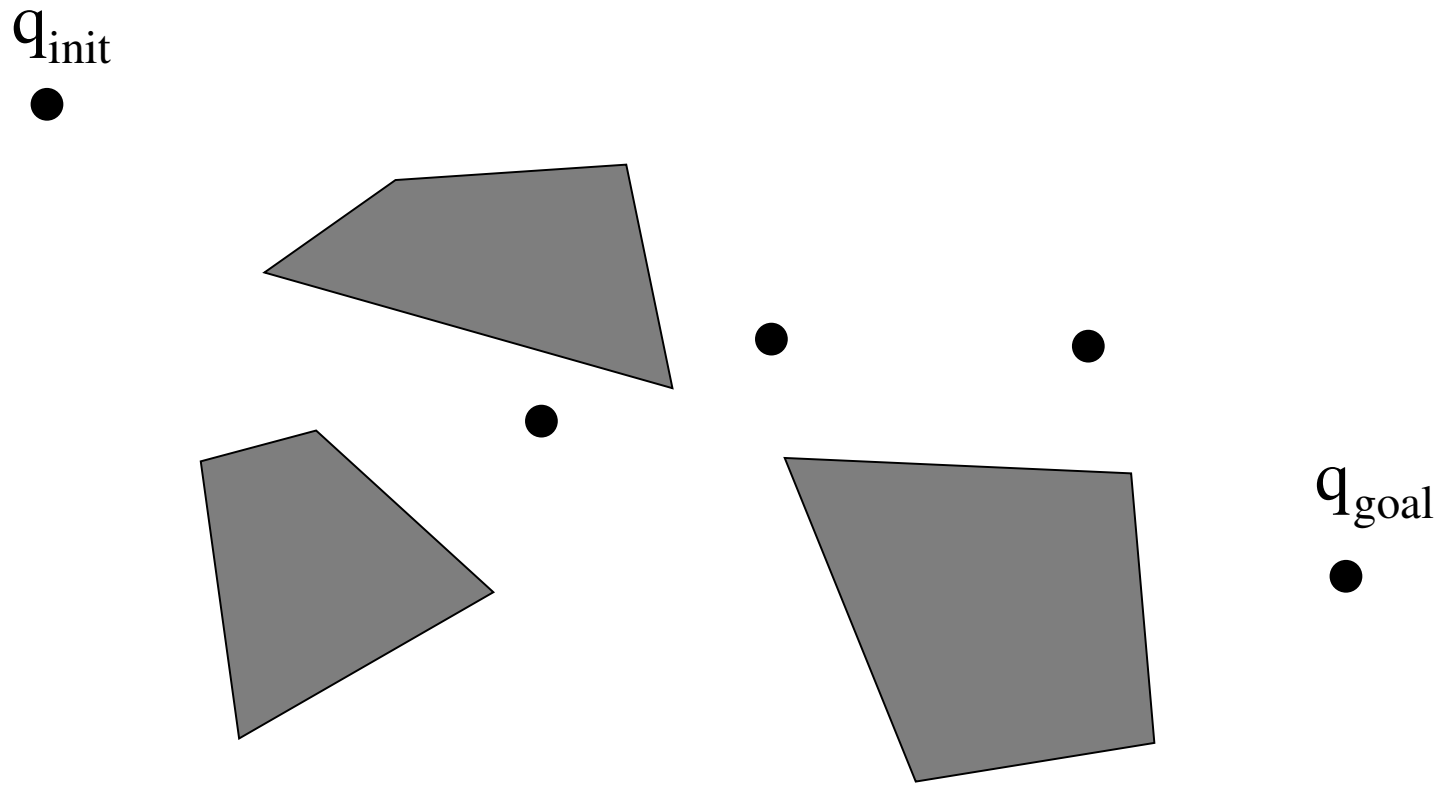
Visibility-based probabilistic roadmap (Visi-PRM) 1999



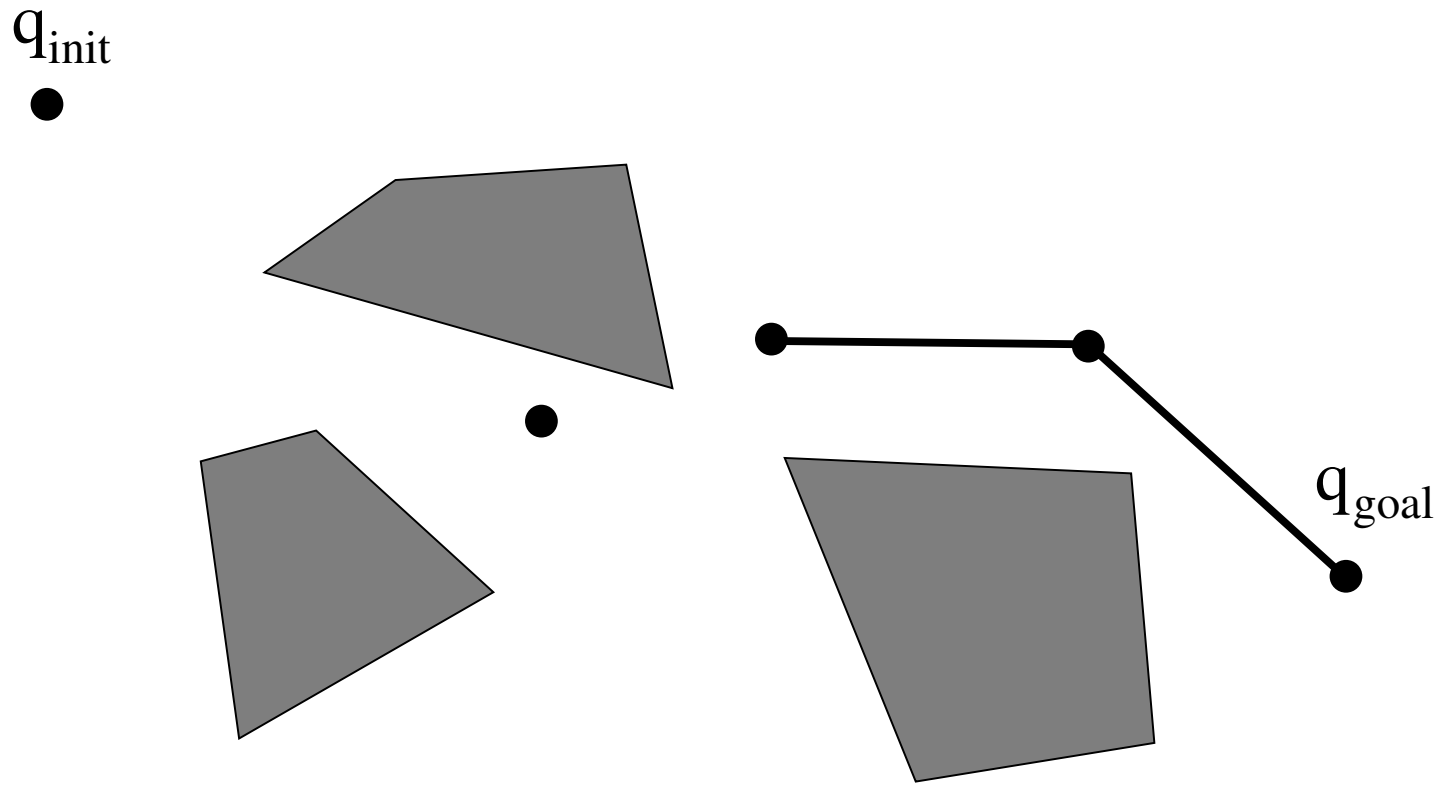
Visibility-based probabilistic roadmap (Visi-PRM) 1999



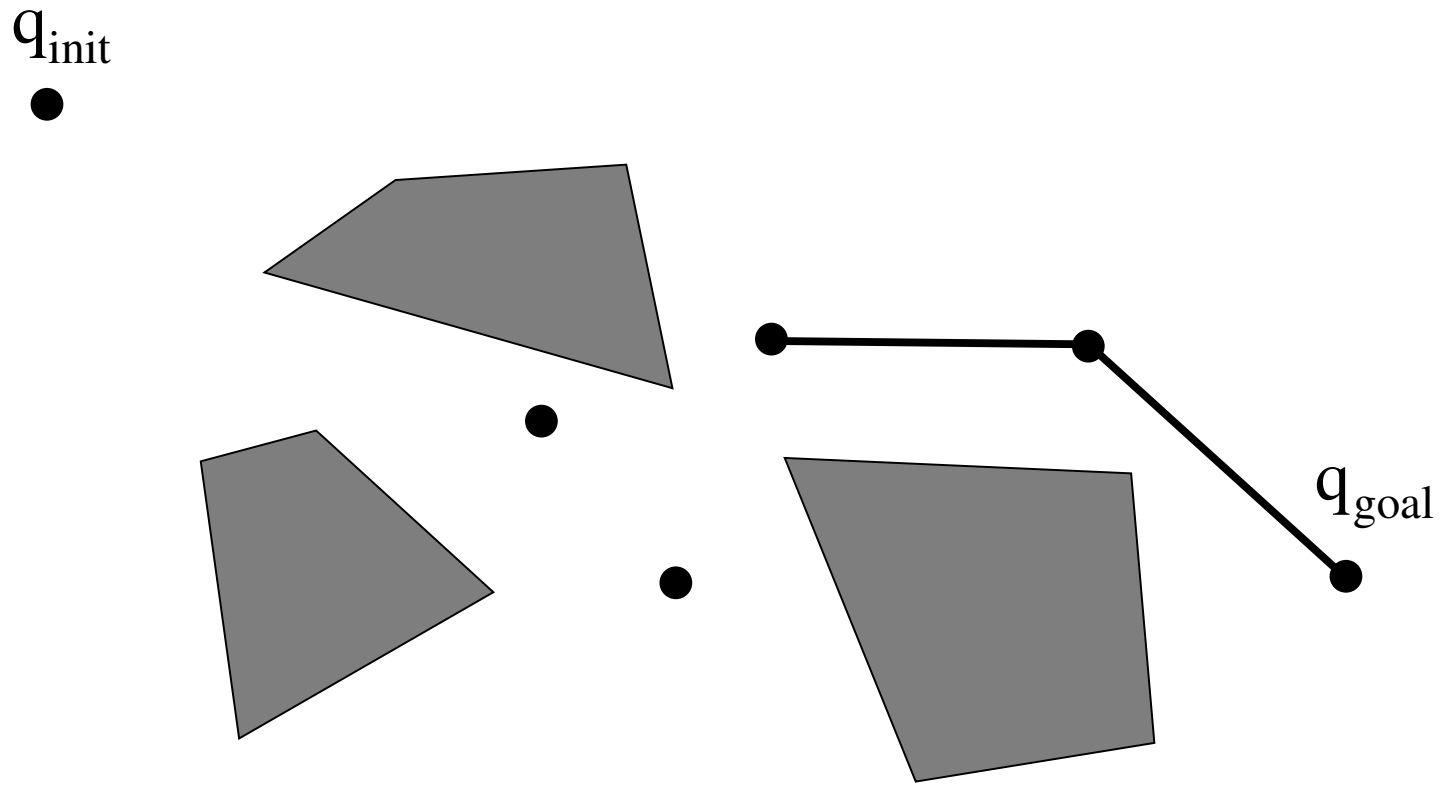
Visibility-based probabilistic roadmap (Visi-PRM) 1999



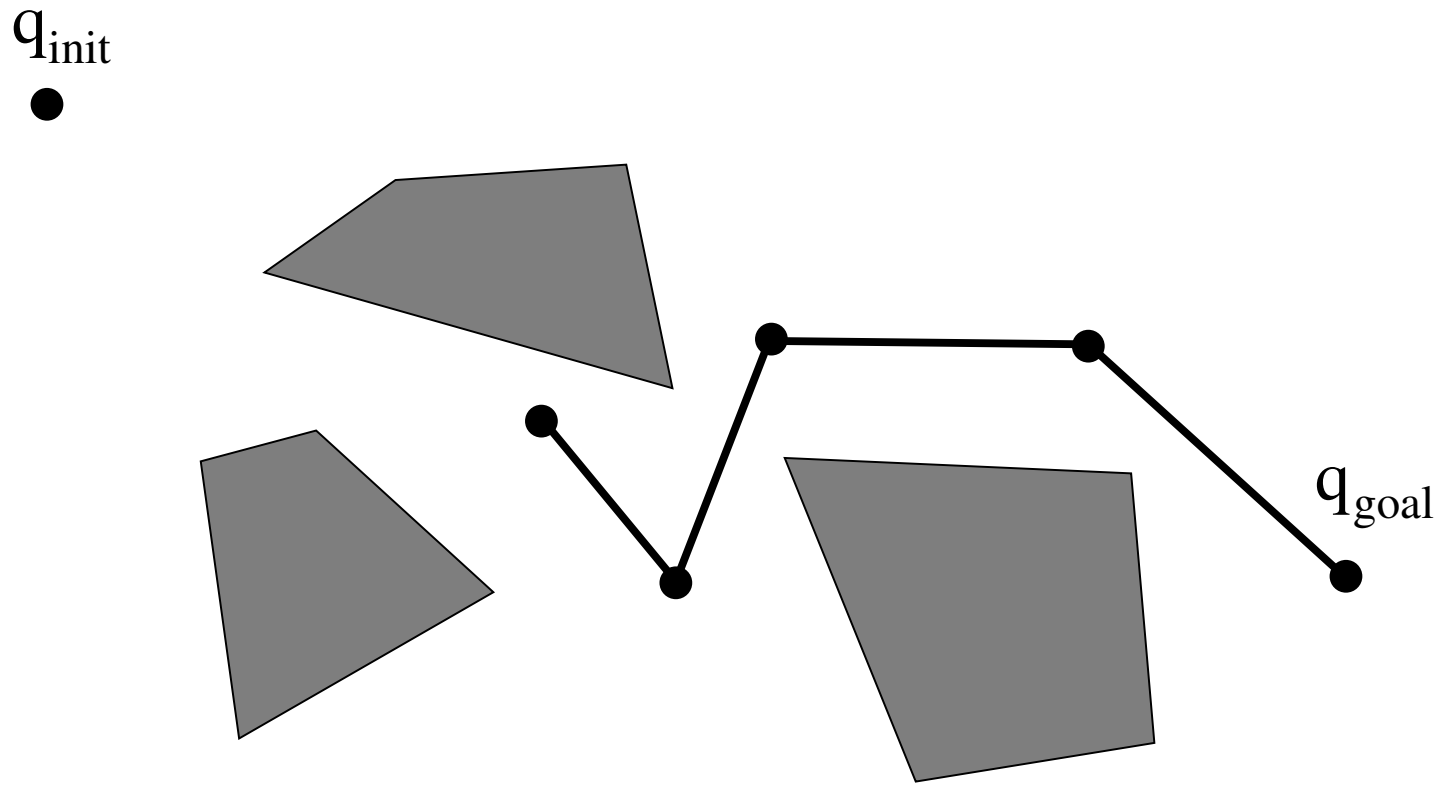
Visibility-based probabilistic roadmap (Visi-PRM) 1999



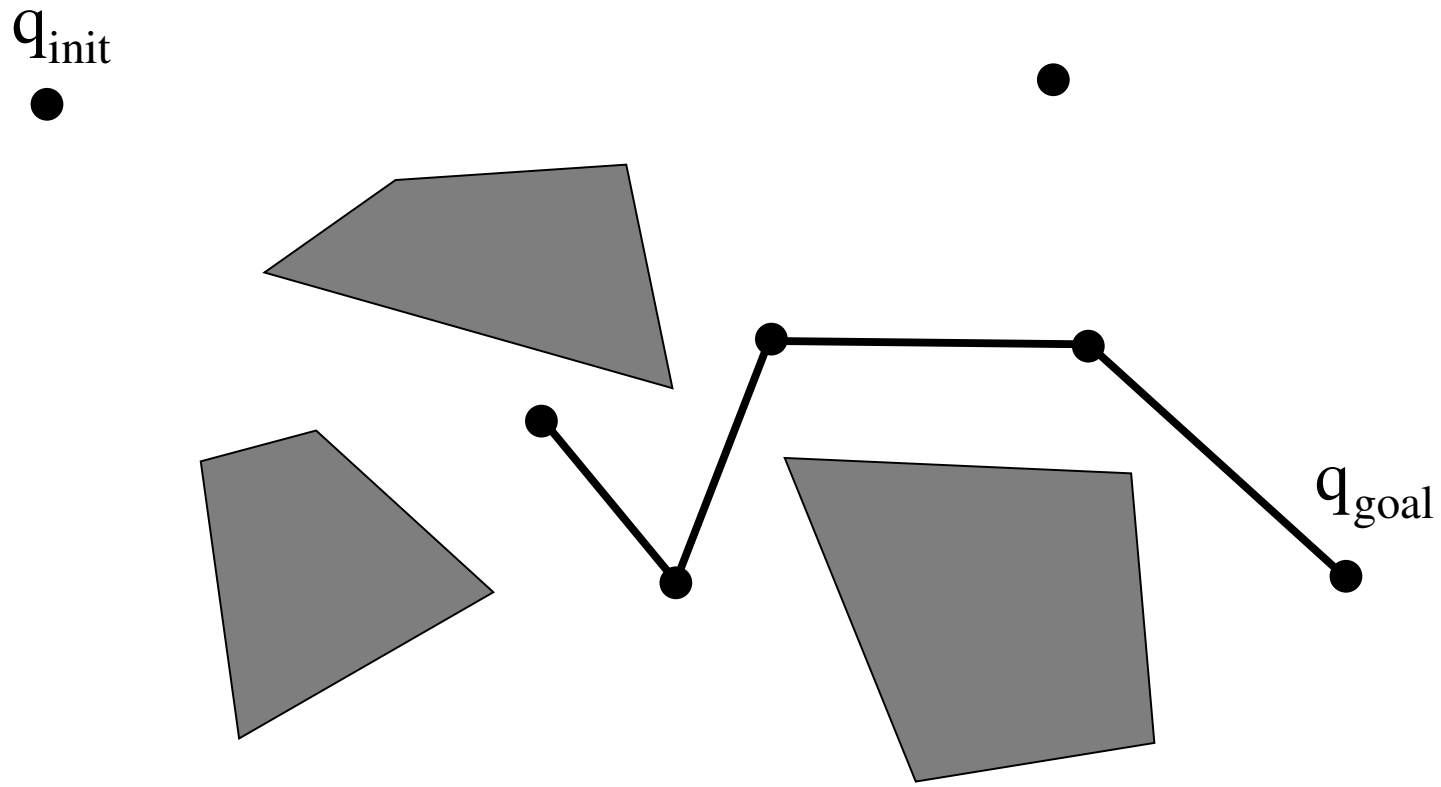
Visibility-based probabilistic roadmap (Visi-PRM) 1999



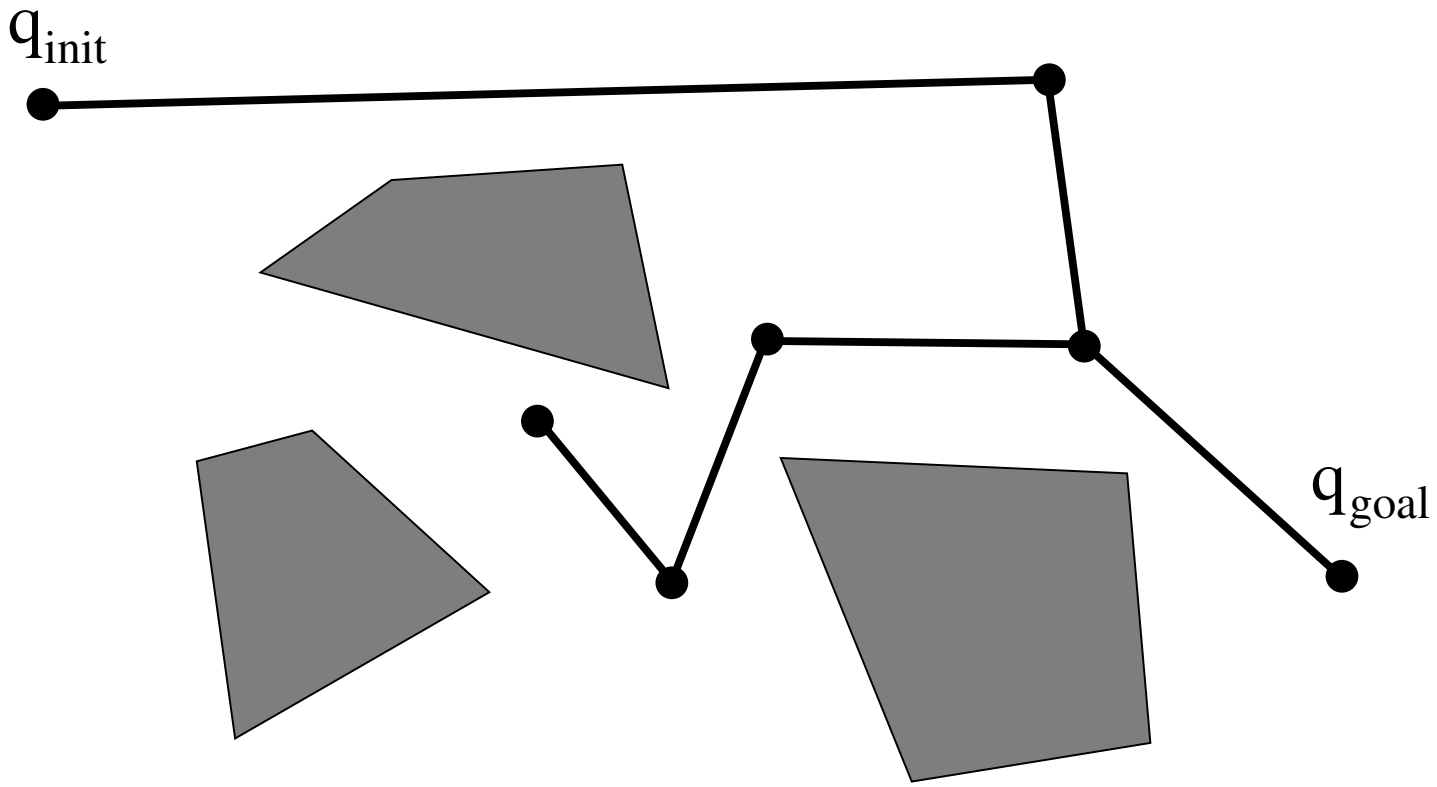
Visibility-based probabilistic roadmap (Visi-PRM) 1999



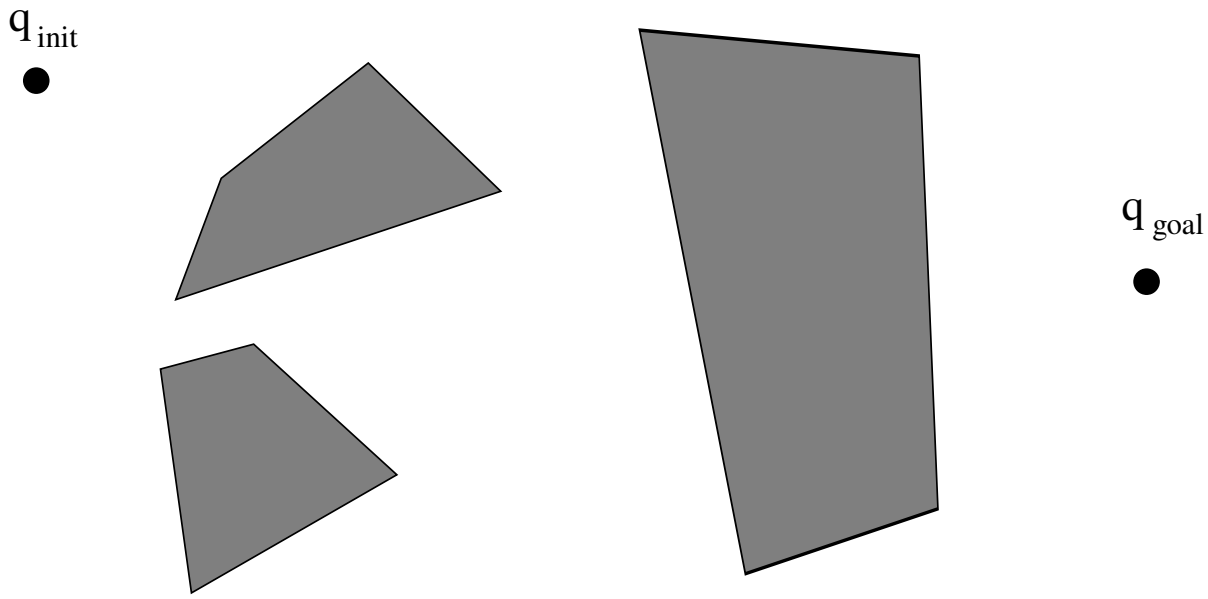
Visibility-based probabilistic roadmap (Visi-PRM) 1999



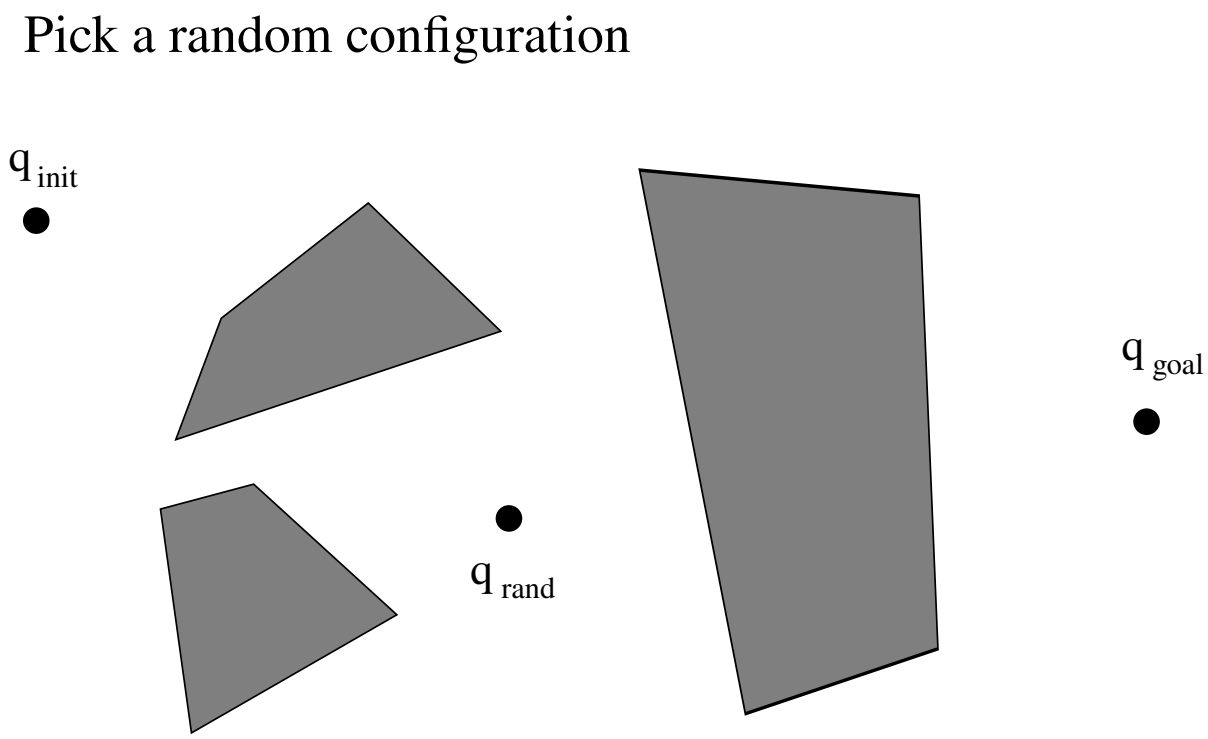
Visibility-based probabilistic roadmap (Visi-PRM) 1999



Rapidly exploring Random Tree (RRT) 2000

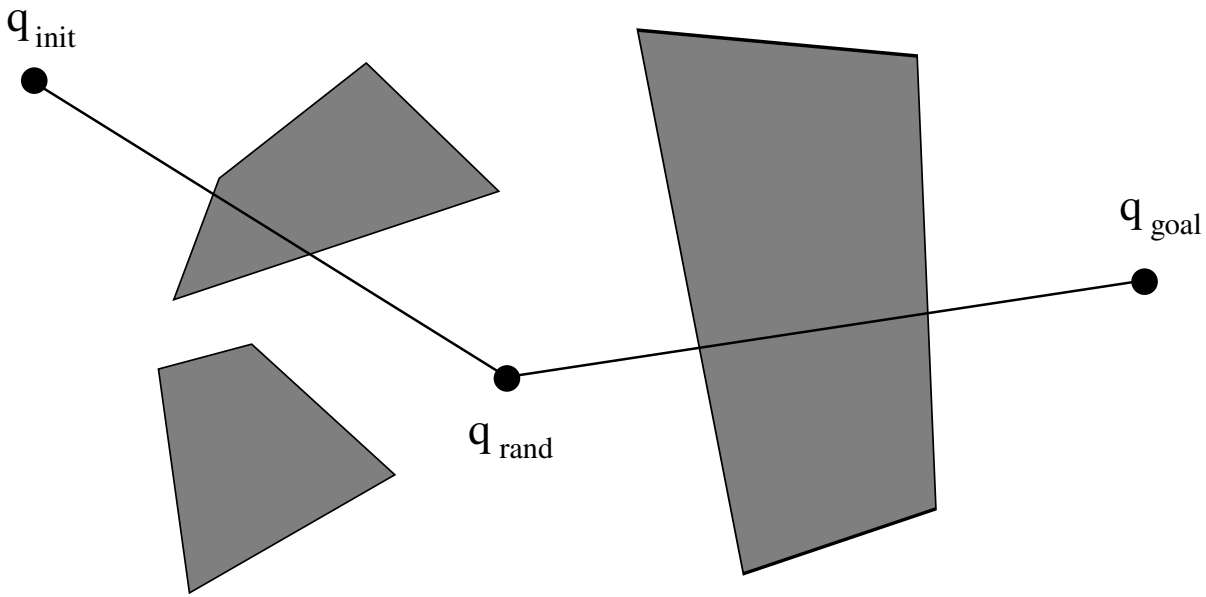


Rapidly exploring Random Tree (RRT) 2000



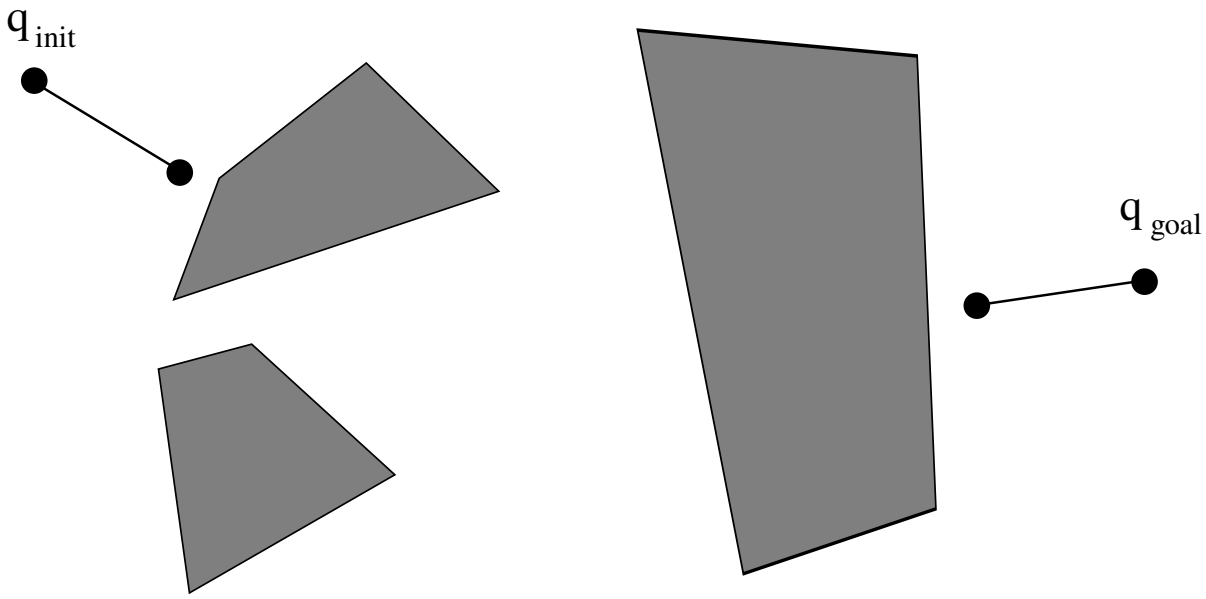
Rapidly exploring Random Tree (RRT) 2000

Try to connect it to the nearest nodes of each connected component



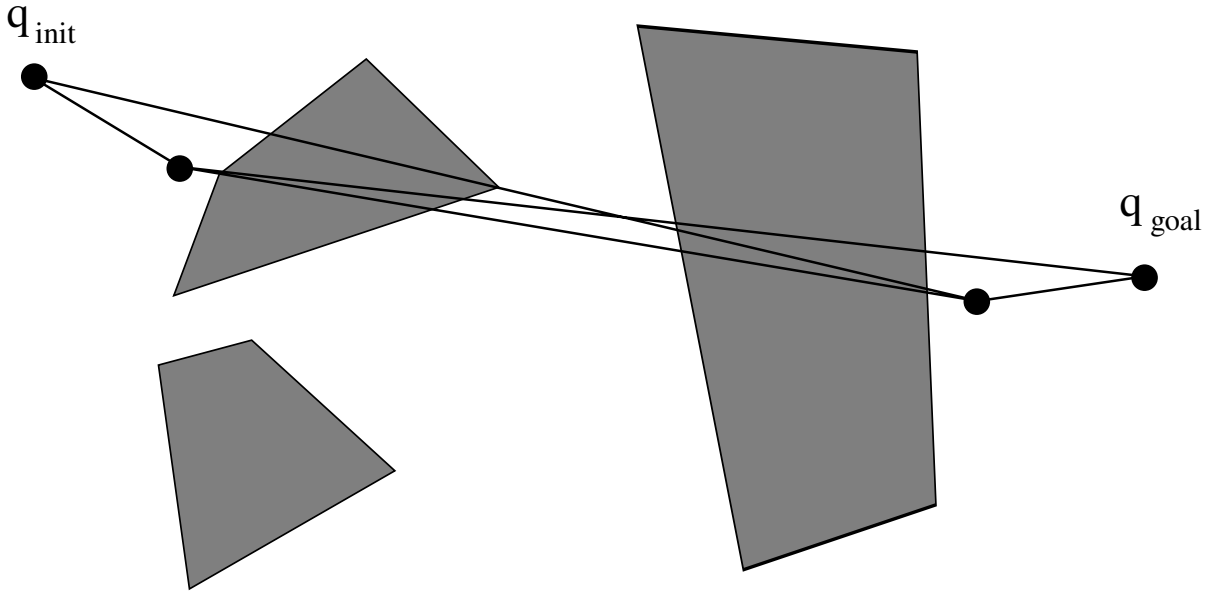
Rapidly exploring Random Tree (RRT) 2000

Keep collision-free parts of paths

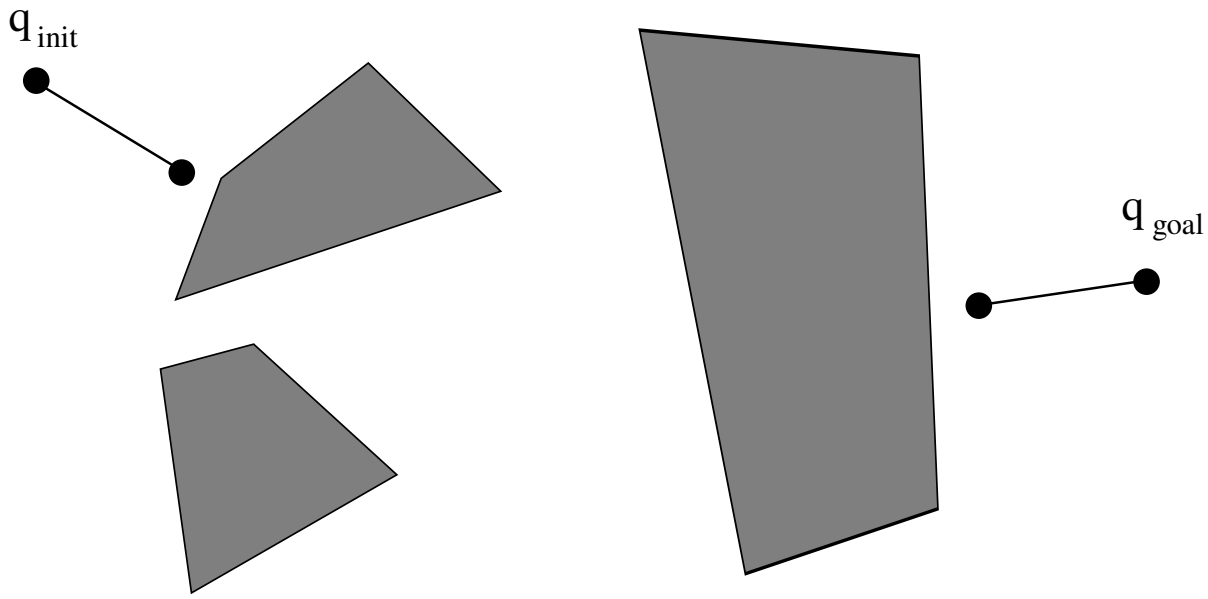


Rapidly exploring Random Tree (RRT) 2000

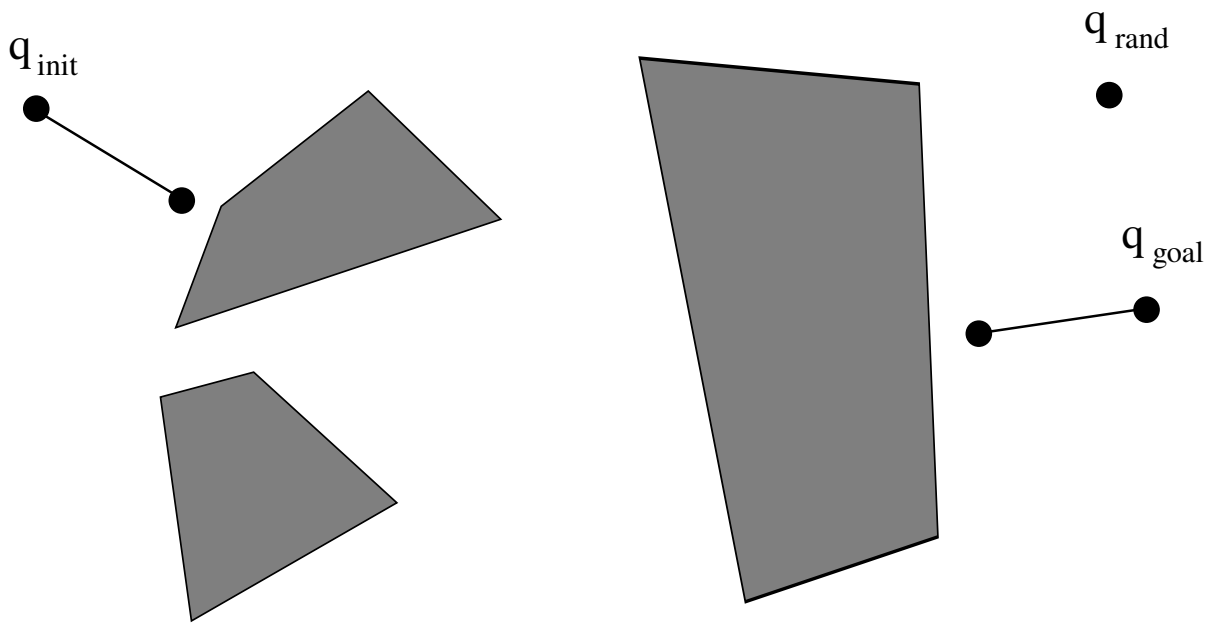
Try to connect new nodes to nearest nodes of other connected components



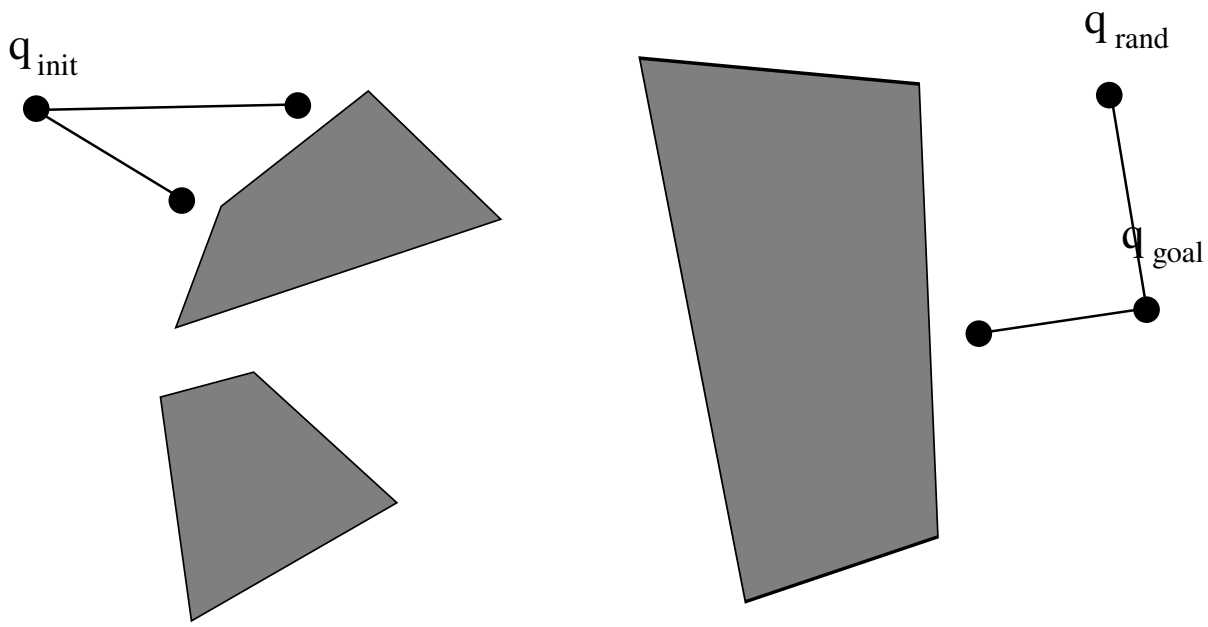
Rapidly exploring Random Tree (RRT) 2000



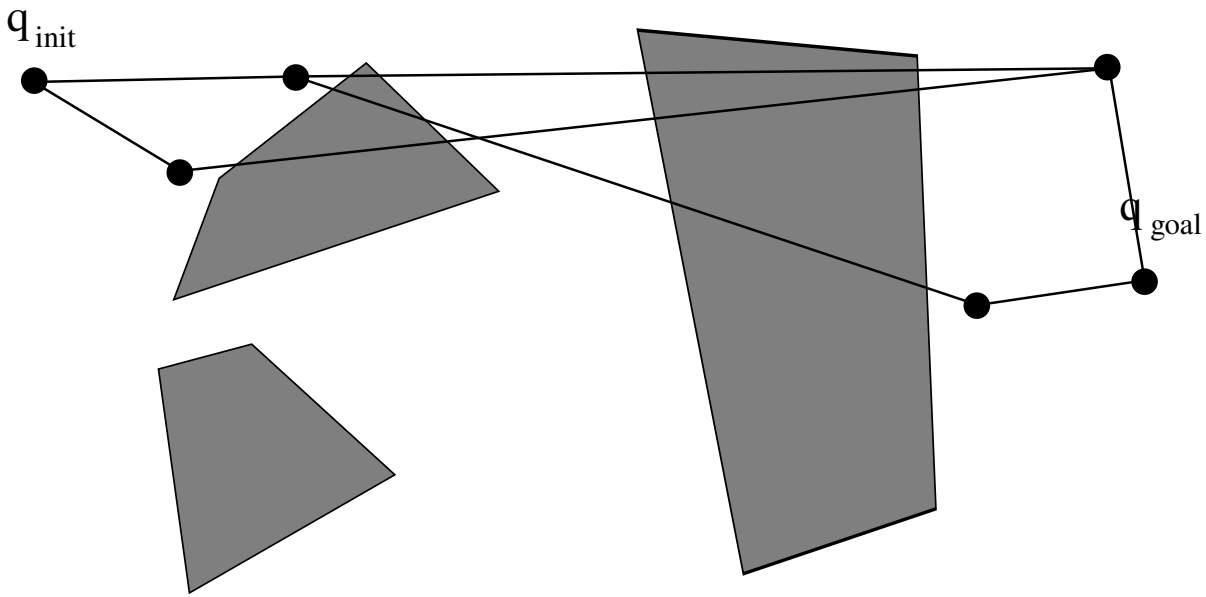
Rapidly exploring Random Tree (RRT) 2000



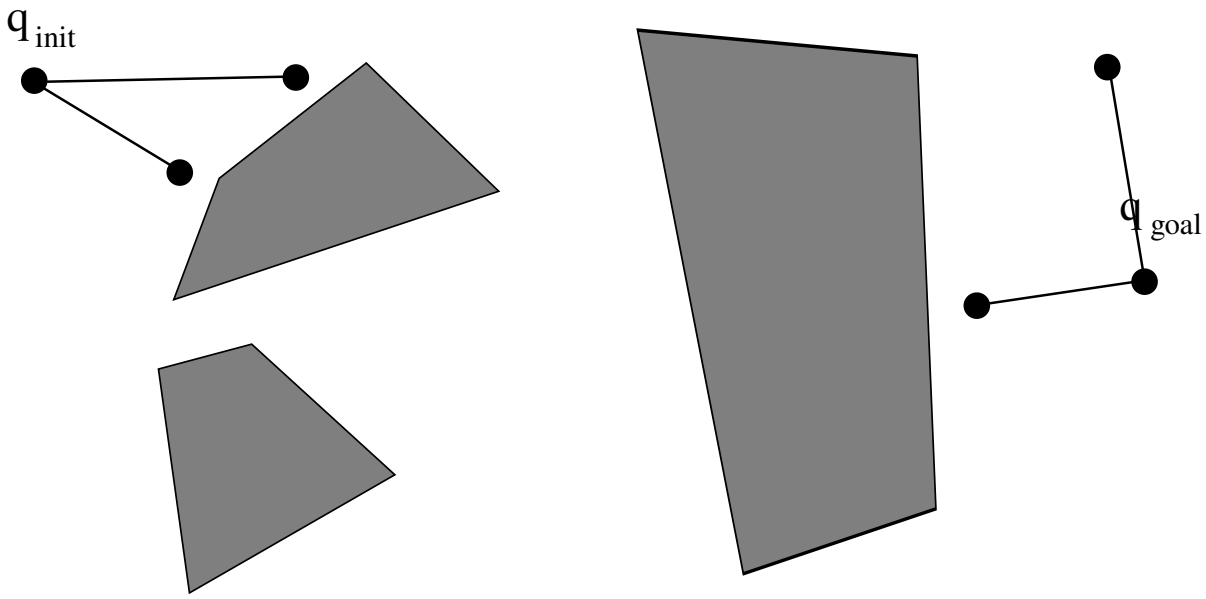
Rapidly exploring Random Tree (RRT) 2000



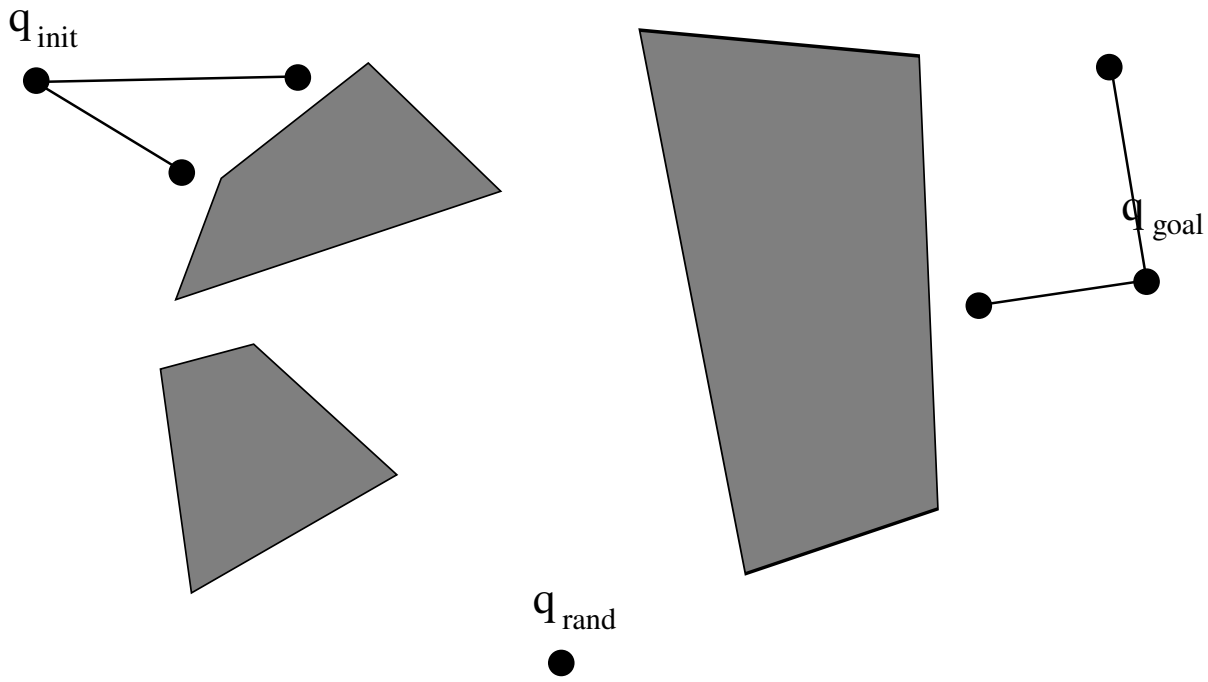
Rapidly exploring Random Tree (RRT) 2000



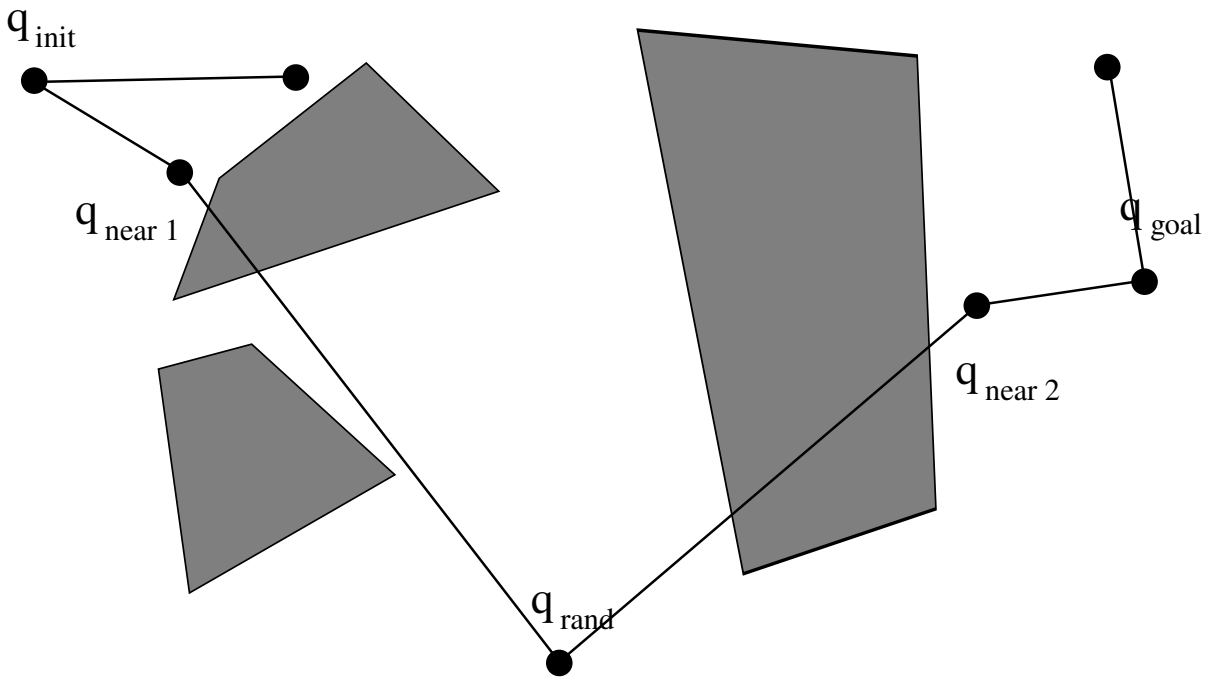
Rapidly exploring Random Tree (RRT) 2000



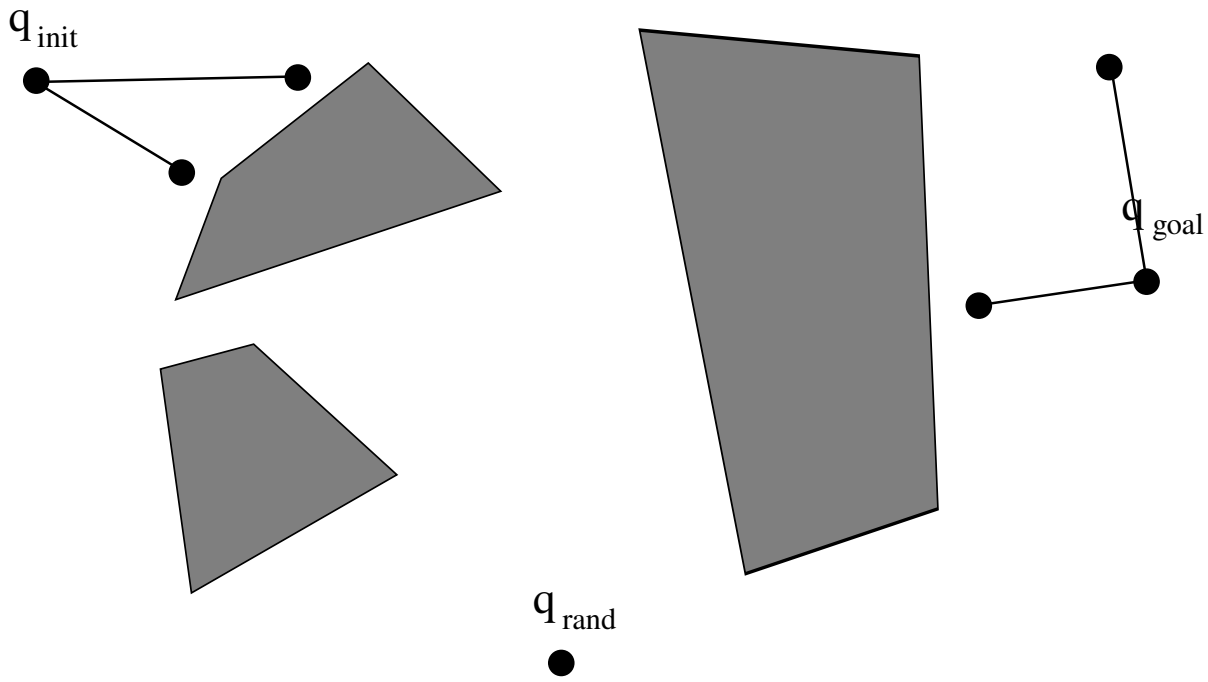
Rapidly exploring Random Tree (RRT) 2000



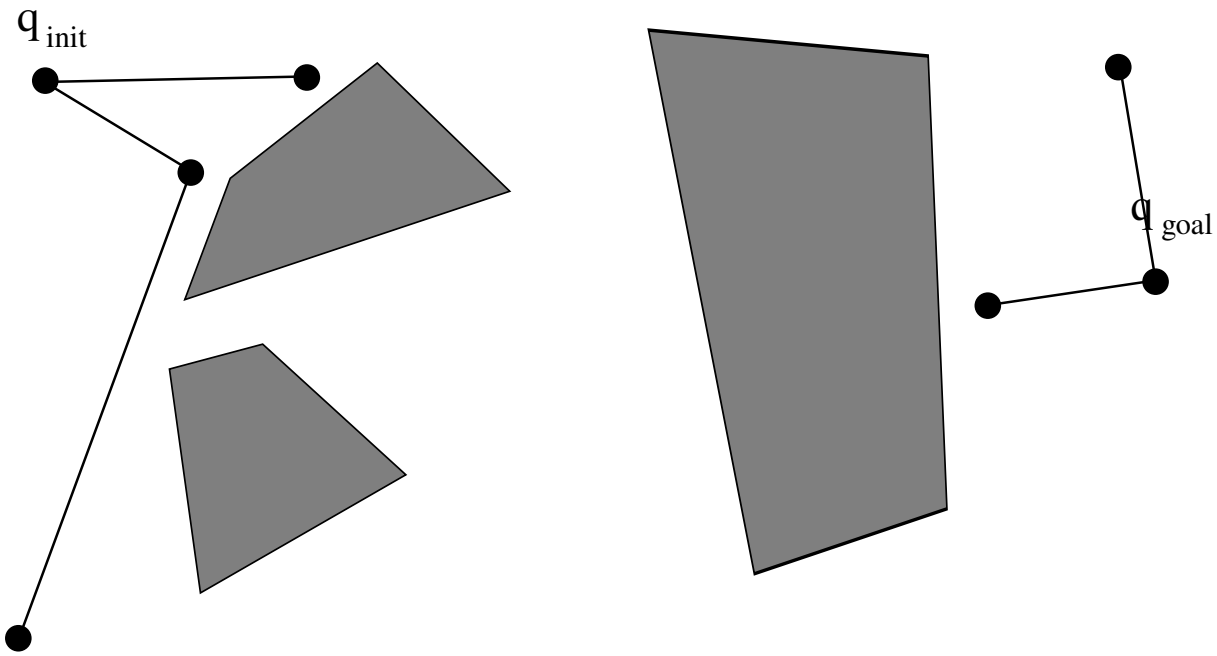
Rapidly exploring Random Tree (RRT) 2000



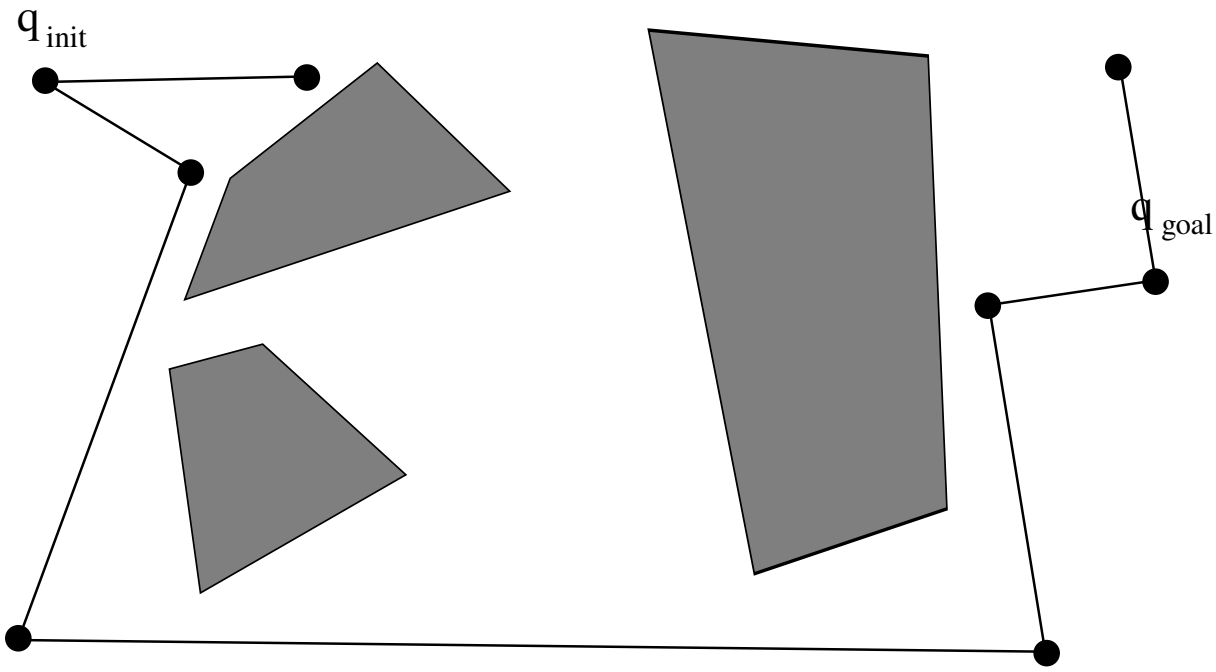
Rapidly exploring Random Tree (RRT) 2000



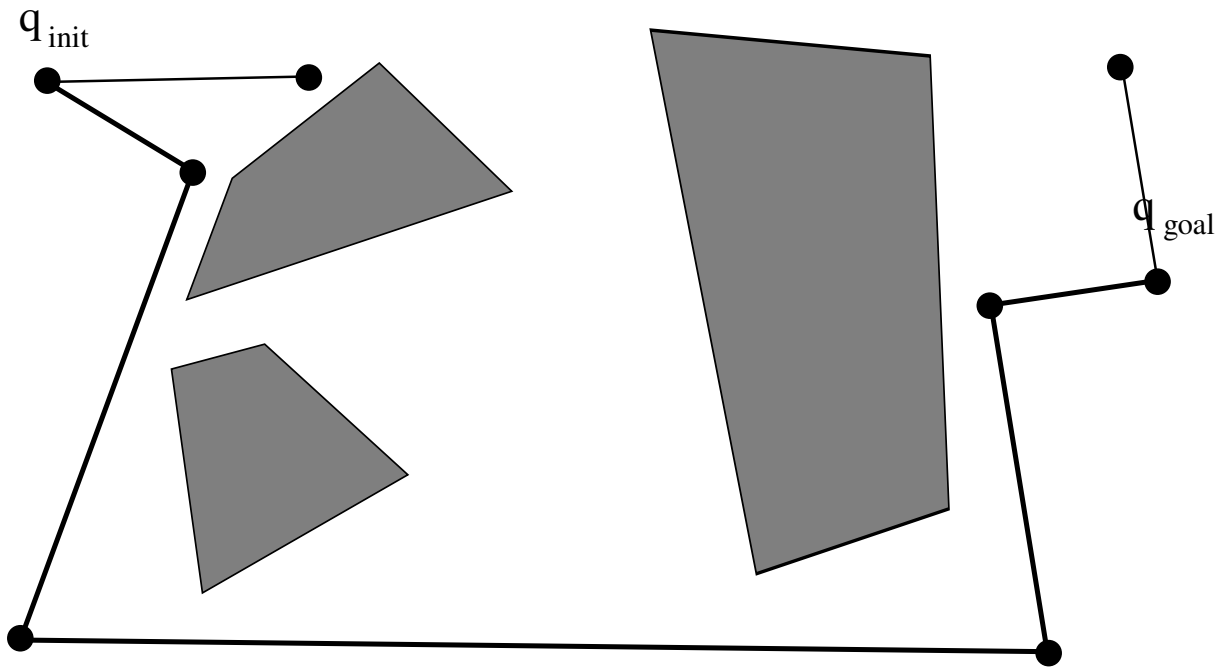
Rapidly exploring Random Tree (RRT) 2000



Rapidly exploring Random Tree (RRT) 2000



Rapidly exploring Random Tree (RRT) 2000



Random methods

- Pros :
  - no explicit computation of the free configuration space,
  - easy to implement,
  - robust.
- Cons :
  - no completeness property, only probabilistic completeness,
  - difficult to find narrow passages.
- Requested operators :
  - Collision tests
    - for configurations (static),
    - for paths (dynamic)

Asymptotically optimal random methods

Asymptotically optimal variants of PRM and RRT exist :

- when the number of nodes tends to infinity,
- the solution computed by the algorithm tends to the optimal collision-free path.

PRM*

PRM

$V \leftarrow \emptyset$, $E \leftarrow \emptyset$
for $i \in \{1, \cdots, n\}$ **do**
  $\mathbf{q}_{rand} \leftarrow \text{SampleFree}_i$
  $U \leftarrow G.near(\mathbf{q}_{rand}, r)$
  for all $\mathbf{q} \in U$ in order of increasing $\|\mathbf{q} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ and $\mathbf{q}$ in different connected components **then**
      $\text{TryConnect}(\mathbf{q}_{rand}, \mathbf{q})$
    end if
  end for
end for

PRM*

$V \leftarrow \text{SampleFree}_{i=1,\cdots,n}$, $E \leftarrow \emptyset$
for all $\mathbf{q} \in V$ **do**
  $U \leftarrow G.near(\mathbf{q}, r^*) \setminus \mathbf{q}$
  for all $\mathbf{q}' \in U$ **do**
    $\text{TryConnect}(\mathbf{q}, \mathbf{q}')$
  end for
end for

$r^* = \gamma_{PRM}(\log(n)/n)^{\frac{1}{d}}$

kPRM*

kPRM

$V \leftarrow \emptyset$, $E \leftarrow \emptyset$
for $i \in \{1, \cdots, n\}$ **do**
  $\mathbf{q}_{rand} \leftarrow \text{SampleFree}_i$
  $U \leftarrow G.nearest(\mathbf{q}_{rand}, k)$
  for all $\mathbf{q} \in U$ in order of increasing $\|\mathbf{q} - \mathbf{q}_{rand}\|$ **do**
    if $\mathbf{q}_{rand}$ and $\mathbf{q}$ in different connected components **then**
      TryConnect$(\mathbf{q}_{rand}, \mathbf{q})$
    end if
  end for
end for

kPRM*

$V \leftarrow \text{SampleFree}_{i=1,\cdots,n}$, $E \leftarrow \emptyset$
for all $\mathbf{q} \in V$ **do**
  $U \leftarrow G.nearest(\mathbf{q}, k^*) \setminus \mathbf{q}$
  for all $\mathbf{q}' \in U$ **do**
    TryConnect$(\mathbf{q}, \mathbf{q}')$
  end for
end for

$$k^* = k_{PRM}(\log(n)),$$
$$k_{PRM} > e(1 + \frac{1}{d})$$

PRM*,kPRM*

Note that

- PRM* and kPRM* are not iterative anymore,
- making them iterative is not trivial.

Inconvenience: we have to choose
“n” before starting the algorithm

There exists also asymptotically optimal variants of RRT

- RRG, RRT*

but they are specific to a given problem $(\mathbf{q}_{init}, \mathbf{q}_{goal})$.

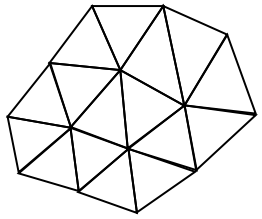
Collision tests

- for configurations
  - problem : given
    - two rigid sets of triangles,
    - the relative position of one set with respect to the other set,

    determine whether the intersection between the sets is empty, or compute the distance between the sets.

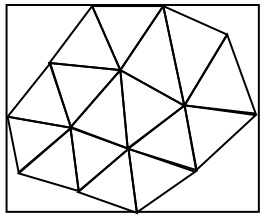
Hierarchy of bounding volumes

- Binary trees of bounding volumes such that
  - each node contains two children,
  - leaves are the triangles



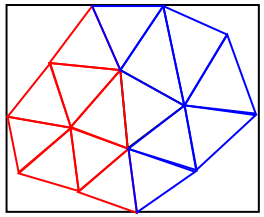
Hierarchy of bounding volumes

- Binary trees of bounding volumes such that
  - each node contains two children,
  - leaves are the triangles



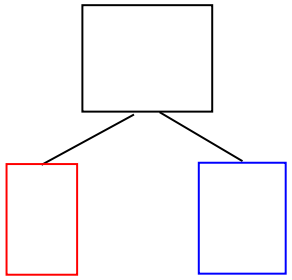
Hierarchy of bounding volumes

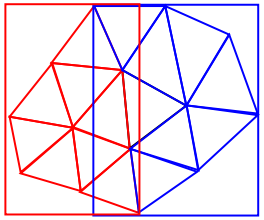
- Binary trees of bounding volumes such that
  - each node contains two children,
  - leaves are the triangles



Hierarchy of bounding volumes

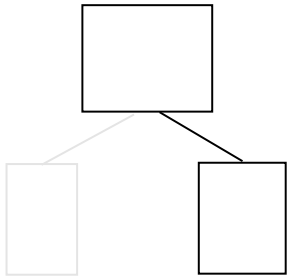
- Binary trees of bounding volumes such that
  - each node contains two children,
  - leaves are the triangles

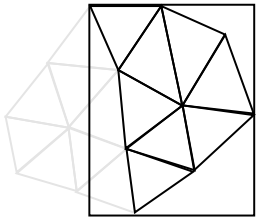




Hierarchy of bounding volumes

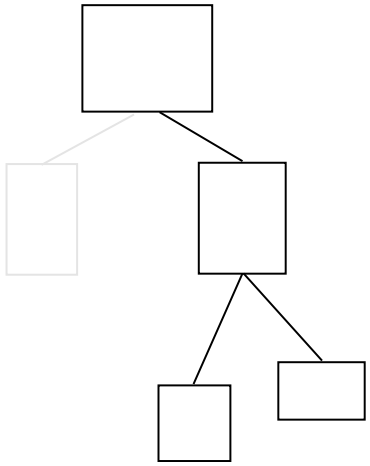
- Binary trees of bounding volumes such that
  - each node contains two children,
  - leaves are the triangles

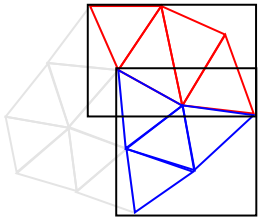




Hierarchy of bounding volumes

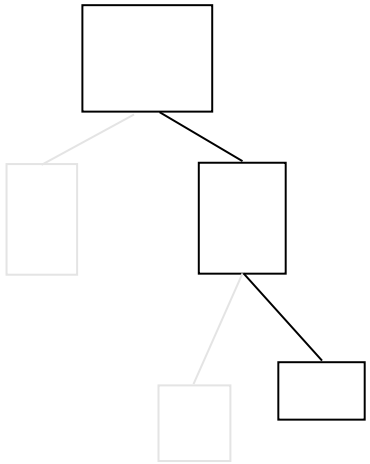
- Binary trees of bounding volumes such that
  - each node contains two children,
  - leaves are the triangles

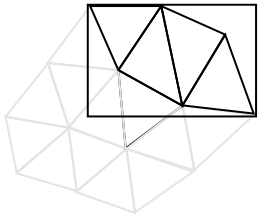




Hierarchy of bounding volumes

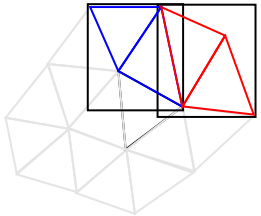
- Binary trees of bounding volumes such that
  - each node contains two children,
  - leaves are the triangles

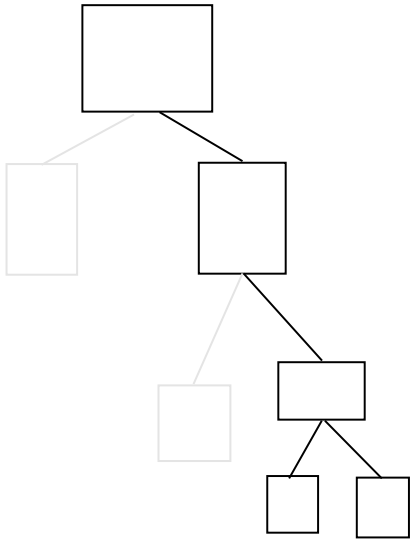




Hierarchy of bounding volumes

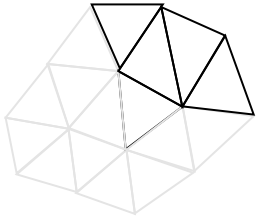
- Binary trees of bounding volumes such that
  - each node contains two children,
  - leaves are the triangles

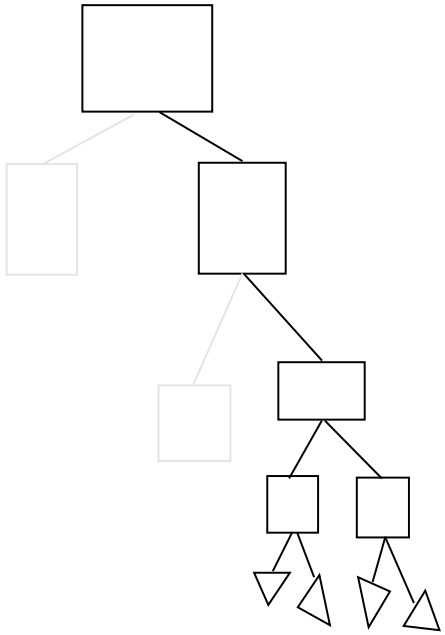




Hierarchy of bounding volumes

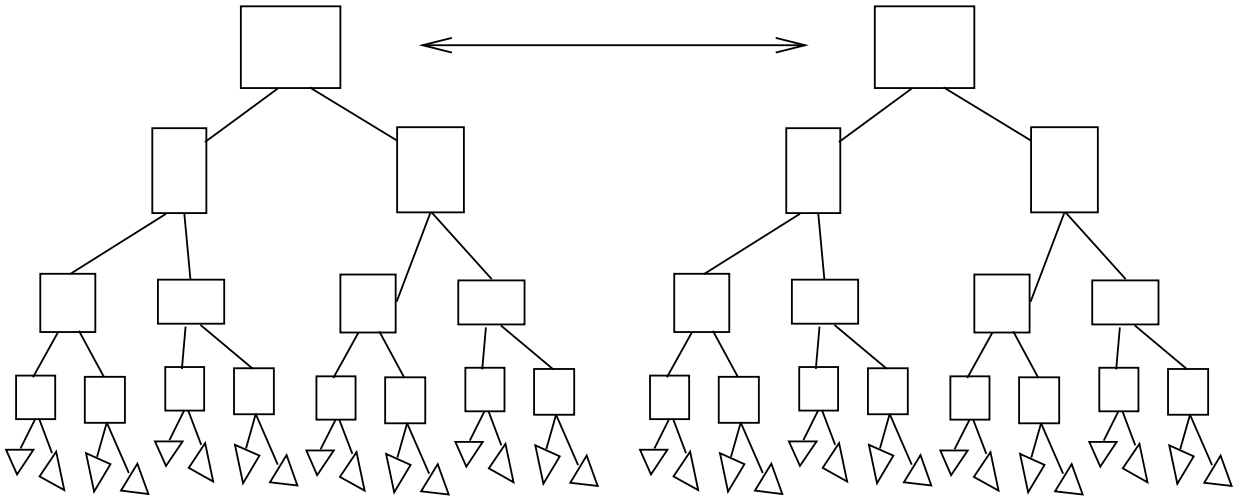
- Binary trees of bounding volumes such that
  - each node contains two children,
  - leaves are the triangles





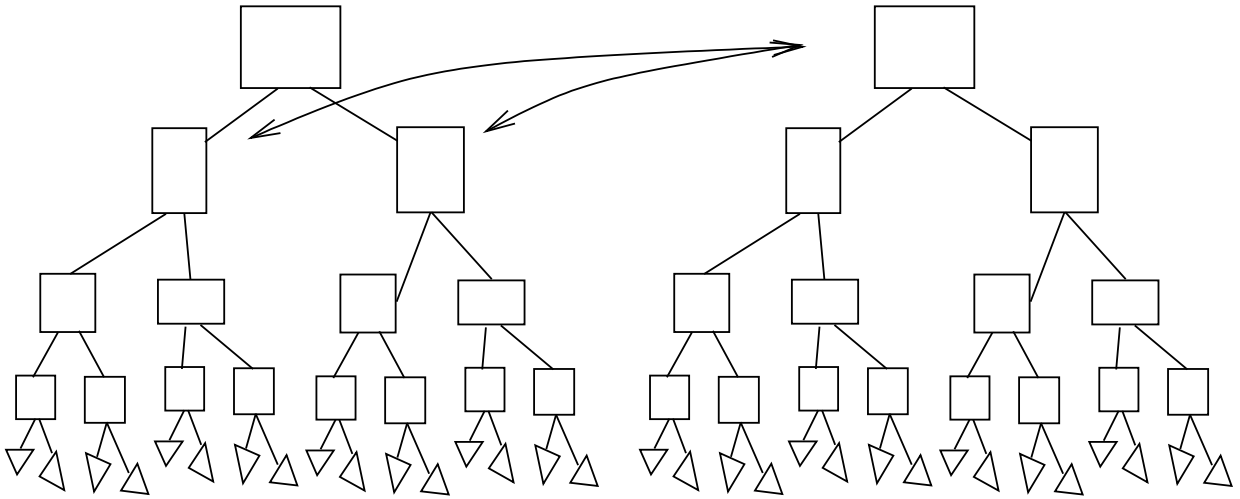
Collision tests for configurations

- Algorithm
  - test root nodes of each tree against one another
  - if two nodes are in collision, test one with the children of the other node



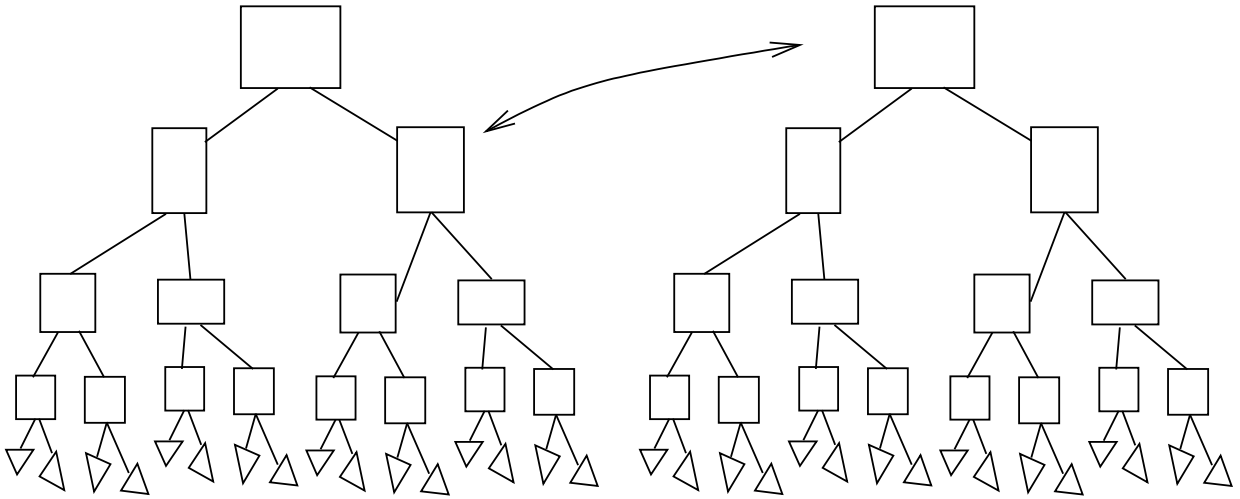
Collision tests for configurations

- Algorithm
  - test root nodes of each tree against one another
  - if two nodes are in collision, test one with the children of the other node



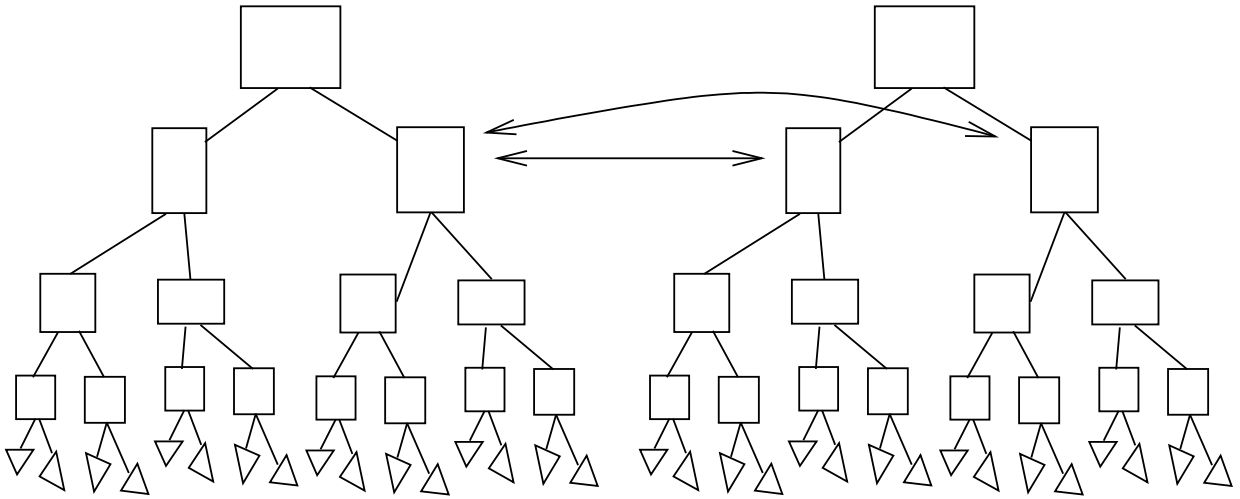
Collision tests for configurations

- Algorithm
  - test root nodes of each tree against one another
  - if two nodes are in collision, test one with the children of the other node



Collision tests for configurations

- Algorithm
  - test root nodes of each tree against one another
  - if two nodes are in collision, test one with the children of the other node

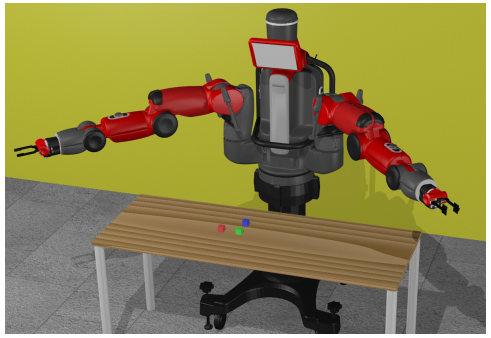


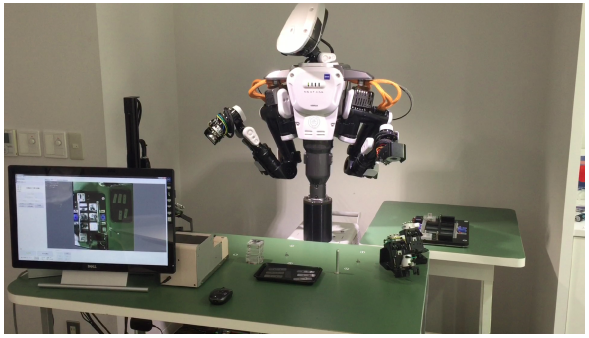
Manipulation motion planning

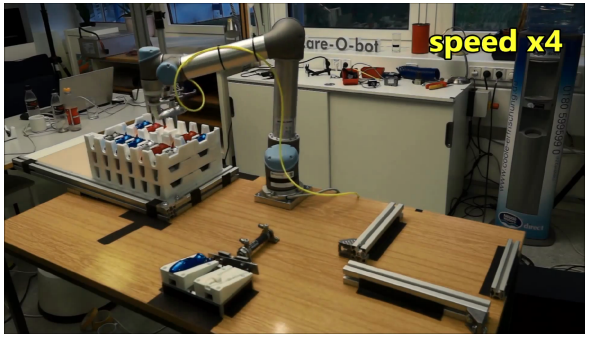
Florent Lamiraux

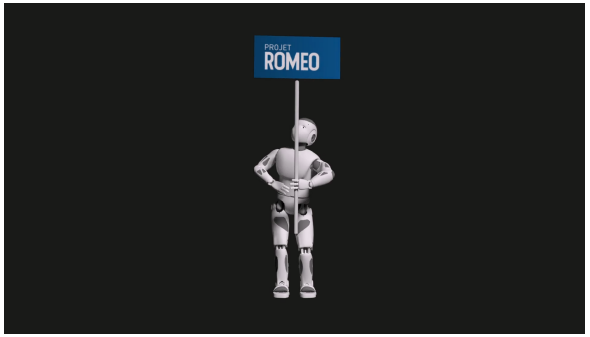
CNRS-LAAS, Toulouse, France

A few examples









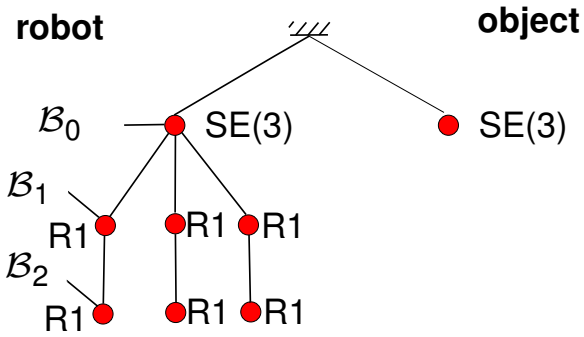
Definitions

A manipulation motion

- is the motion of
  - one or several robots and of
  - one or several objects
- such that each object
  - either is in a stable position, or
  - is moved by one or several robots.

Composite robot

Kinematic chain composed of each robot and of each object



$$\mathbf{q} = (q_0, \cdots, q_{n_r}, q_{n_r+1}, \cdots, q_{n_r+n_o})$$

The configuration space of a composite robot is the cartesian product of the configuration spaces of each robot and object.

$$\mathcal{C} = \mathcal{C}_{r1} \times \cdots \times \mathcal{C}_{r_{nb\ robots}} \times SE(3)^{nb\ objets}$$

Numerical constraints

Constraints to which manipulation motions are subject can be expressed numerically.

- Numerical constraints:

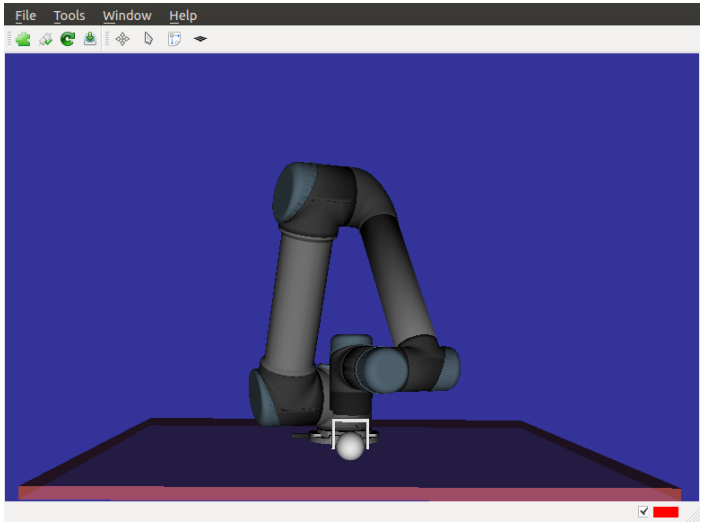
$$f(\mathbf{q}) = 0, \quad \begin{array}{l} m \in \mathbb{N}, \\ f \in \mathcal{C}^1(\mathcal{C}, \mathbb{R}^m) \end{array}$$

  - `setConstantRightHandSide(True)`
- Parameterizable numerical constraints:

$$f(\mathbf{q}) = f_0, \quad \begin{array}{l} m \in \mathbb{N}, \\ f \in \mathcal{C}^1(\mathcal{C}, \mathbb{R}^m) \\ f_0 \in \mathbb{R}^m \end{array}$$

  - `setConstantRightHandSide(False)`

Example: robot manipulating a ball



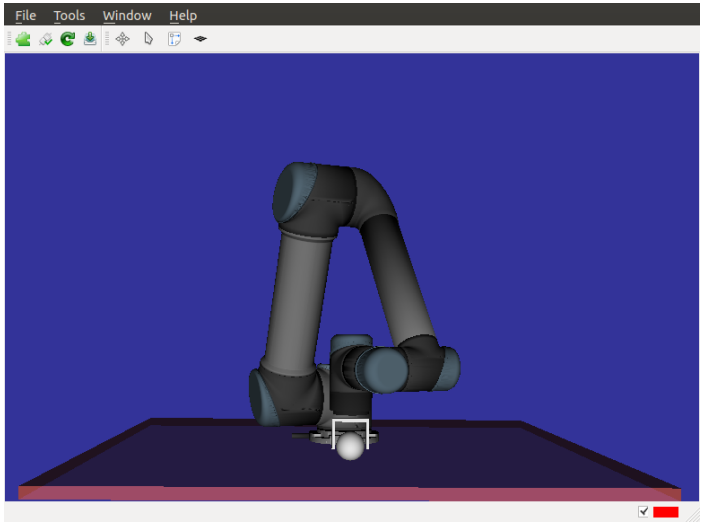
$$\mathcal{C} = [-\pi, \pi]^6 \times \mathbb{R}^3 \quad (1)$$

$$\mathbf{q} = (q_0, \cdots, q_5, x_b, y_b, z_b) \quad (2)$$

Two *states*:

- `placement`: the ball is lying on the table,
- `grasp`: the ball is hold by the end-effector.

Example: robot manipulating a ball



Each state is defined by a numerical constraint

- `placement`

$$z_b = 0$$

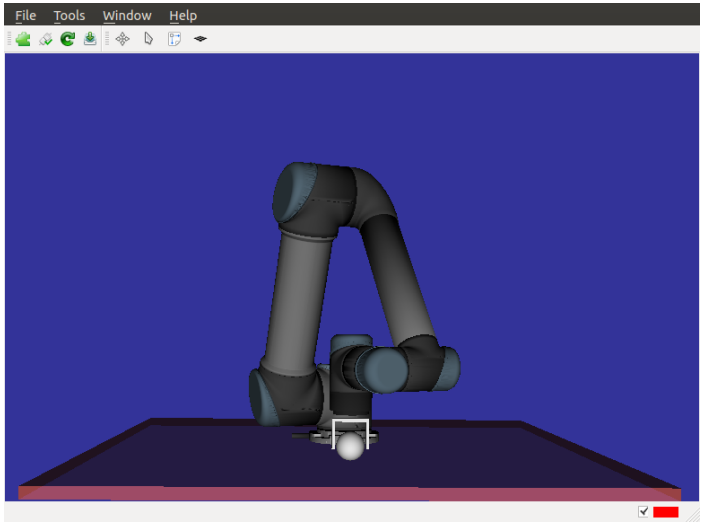
- `grasp`

$$\mathbf{x}_{gripper}(q_0, \cdots, q_5) - \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = 0$$

Each state is a sub-manifold of the configuration space

Example: robot manipulating a ball

Motion constraints

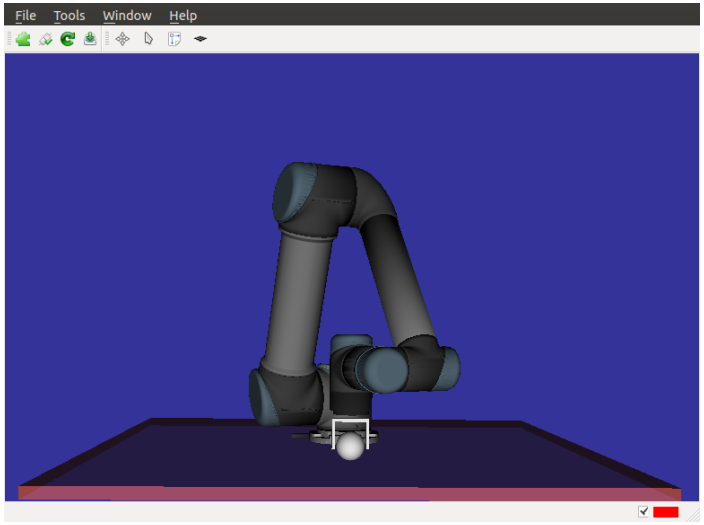


Two *types of motion*:

- `transit`: the ball is lying and **fixed** on the table,
- `transfer`: the ball moves with the end-effector.

Example: robot manipulating a ball

Motion constraints



- `transit`

$$\left.\begin{array}{rcl} x_b & = & x_0 \\ y_b & = & y_0 \end{array}\right\} \text{ parameterizable}$$

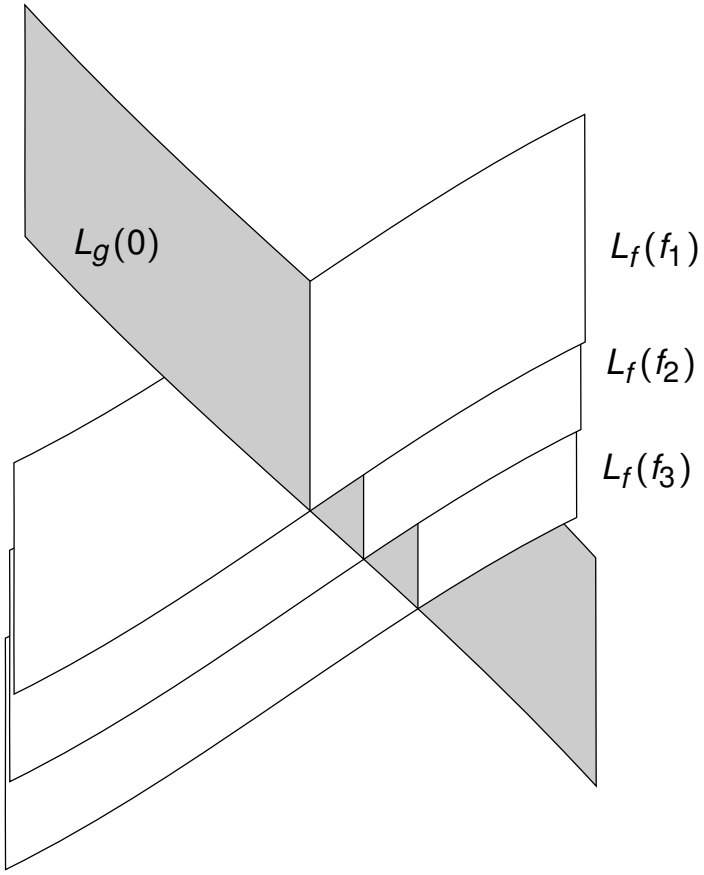
$$\left.\begin{array}{l} z_b = 0 \end{array}\right\} \text{ simple}$$

- `transfer`

$$\mathbf{x}_{gripper}(q_0, \cdots, q_5) - \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = 0$$

Foliation = feuilletage

Motion constraints define a foliation of the admissible configuration space (`grasp` $\cup$ `placement`).



- f: position of the ball

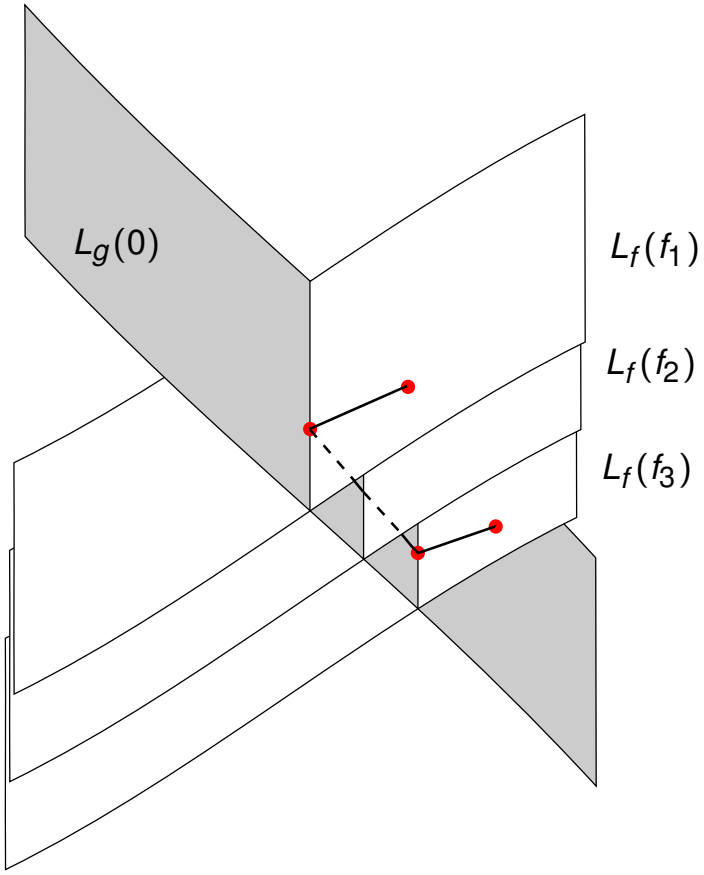
$$L_f(f_1) = \{\mathbf{q} \in \mathcal{C}, f(\mathbf{q}) = f_1\}$$

- g: grasp of the ball

$$L_g(0) = \{\mathbf{q} \in \mathcal{C}, g(\mathbf{q}) = 0\}$$

Foliation

Motion constraints define a foliation of the admissible configuration space (`grasp` $\cup$ `placement`).



Solution to a manipulation planning problem is a concatenation of *transit* and *transfer* paths.

General case

In a manipulation problem,

- the state of the system is subject to
  - numerical constraints
- trajectories of the system are subject to
  - numerical constraints
  - parameterizable numerical constraints.

General case

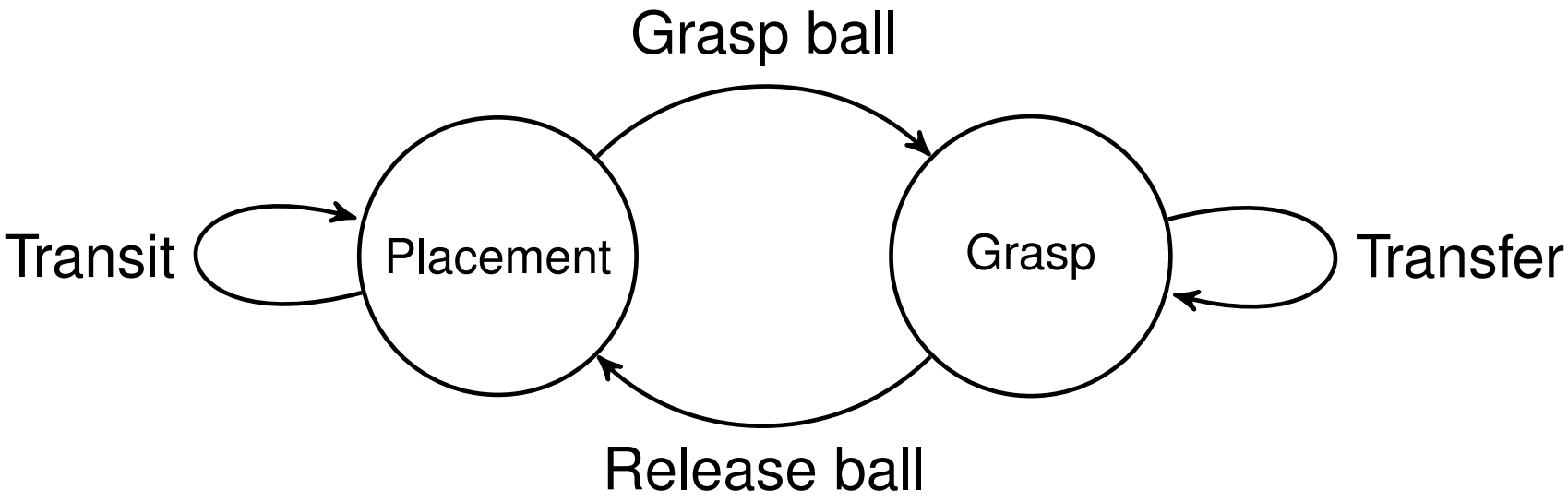
In a manipulation problem,

- the state of the system is subject to
  - numerical constraints
- trajectories of the system are subject to
  - ~~numerical constraints~~
  - parameterizable numerical constraints, the dimension of the parameter being possibly less than the dimension of the constraint.
  - parameter value is constant along trajectories.

Constraint graph

A manipulation planning problem can be represented by a *manipulation graph*.

- **Nodes** or *states* are numerical constraints.
- **Edges** or *transitions* are parameterizable numerical constraints.



Projecting configuration on constraint

Newton-Raphson algorithm

- $\mathbf{q}_0$ configuration,
- $f(\mathbf{q}) = 0$ non-linear constraint,
- ϵ numerical tolerance

Projection $(\mathbf{q}_0, f)$:

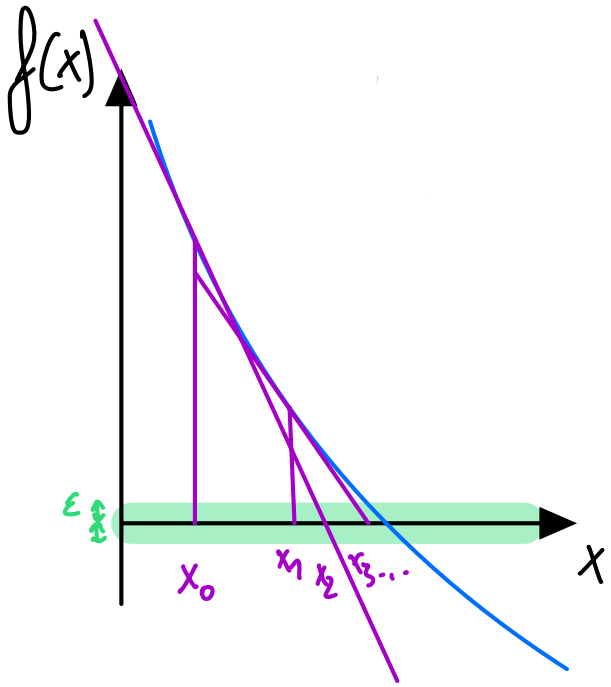
$\mathbf{q} = \mathbf{q}_0$; $\alpha = 0.95$

for i from 1 to max_iter:

$\mathbf{q} = \mathbf{q} - \alpha \left(\frac{\partial f}{\partial \mathbf{q}}(\mathbf{q}) \right)^{+} f(\mathbf{q})$

if $\|f(\mathbf{q})\| < \epsilon$: return $\mathbf{q}$

return failure



$$f(x_1) = f(x_0) + f'(x_0)(x_1 - x_0) + o(x_1 - x_0) = 0$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

→ dimension n

$$f'(x_0)(x_1 - x_0) = -f(x_0)$$

$$\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_p}{\partial x_1} & \cdots & \frac{\partial f_p}{\partial x_n} \end{pmatrix}$$

A X = Y

Steering method

Mapping $\mathcal{SM}$ from $\mathcal{C} \times \mathcal{C}$ to $C^1([0,1], \mathcal{C})$ such that

$$\mathcal{SM}(\mathbf{q}_0, \mathbf{q}_1)(0) = \mathbf{q}_0$$
$$\mathcal{SM}(\mathbf{q}_0, \mathbf{q}_1)(1) = \mathbf{q}_1$$

Constrained steering method

Let

- $\mathcal{SM}$ be a steering method
- $f \in C^1(\mathcal{C}, \mathbb{R}^m)$ be a numerical constraint.

A constrained steering method $\mathcal{S}\bar{\mathcal{M}}$ of constraint f is a steering method such that

$$\forall t \in [0,1], f(\mathcal{S}\bar{\mathcal{M}}(t)) = 0$$

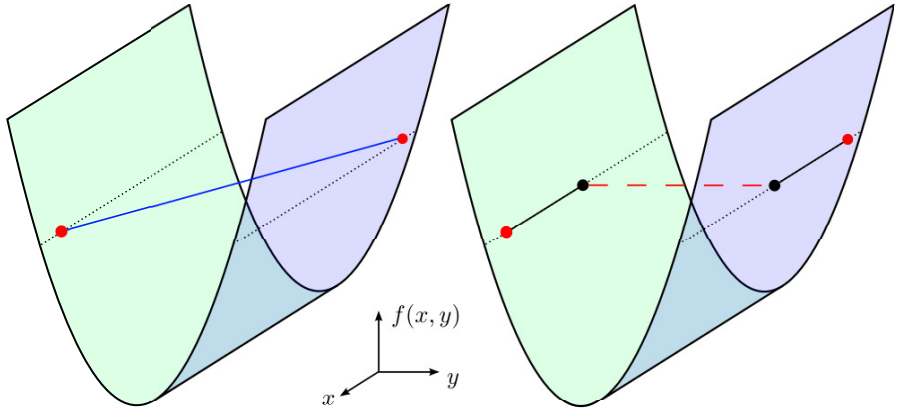
It can be defined by projection

$$\mathcal{S}\bar{\mathcal{M}} = proj \circ \mathcal{SM}$$

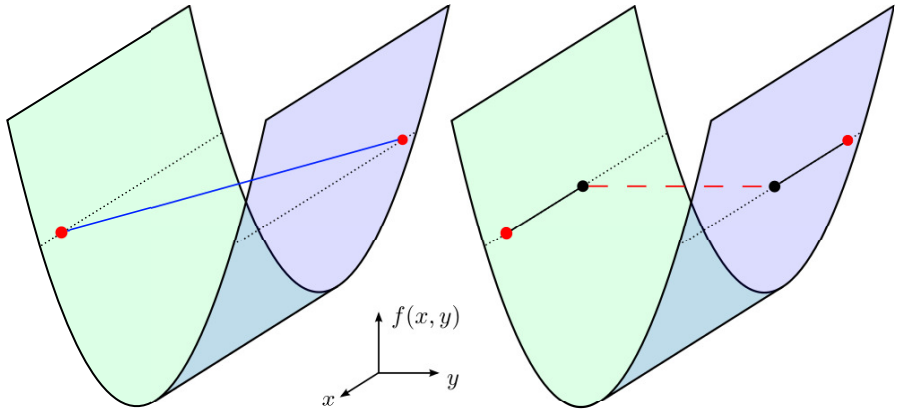
Projecting path on constraint

- *path*: mapping from $[0, 1]$ to $\mathcal{C}$
- $f(\mathbf{q}) = 0$ non-linear constraint,

Applying Newton Raphson at each point may result in a discontinuous path



Discontinuous Projection



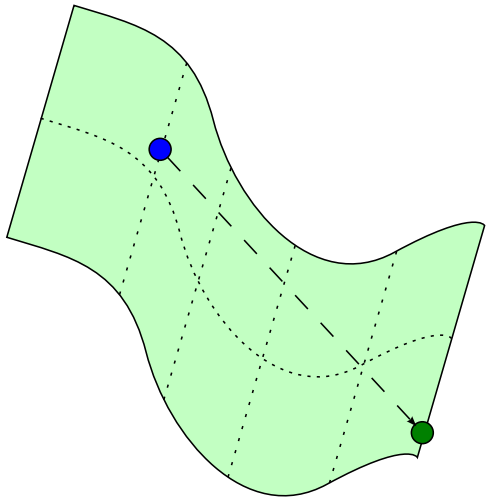
$$\mathcal{C} = \mathbb{R}^2, f(x,y) = y^2 - 1$$

$$\frac{\partial f}{\partial \mathbf{q}} = \begin{pmatrix} 0 & 2y \end{pmatrix}, \frac{\partial f}{\partial \mathbf{q}}^+ = \begin{pmatrix} 0 \\ \frac{1}{2y} \end{pmatrix} \text{ ou } \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$y_{i+1} = y_i + \frac{1 - y_i^2}{2y_i}$$

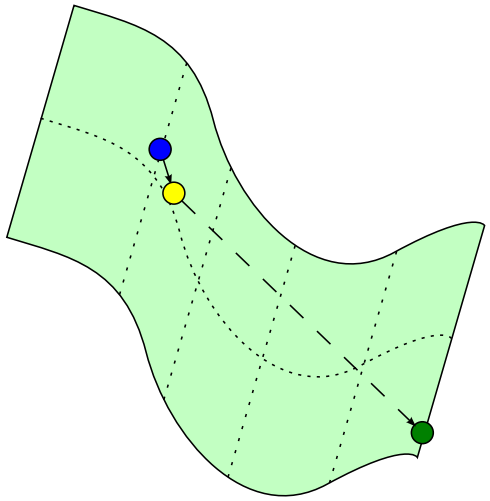
Testing projection continuity

- The initial path is sampled and successive samples are projected,
-
-



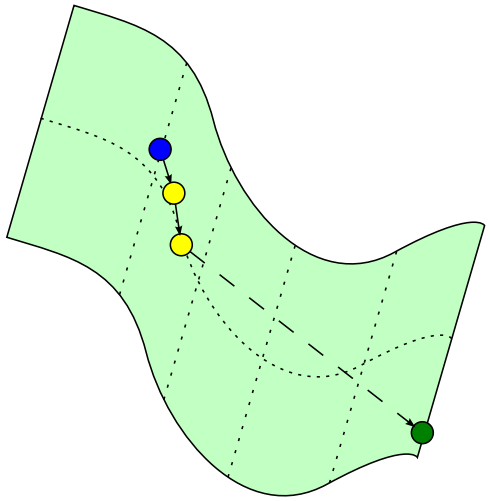
Testing projection continuity

- The initial path is sampled and successive samples are projected,
-
-



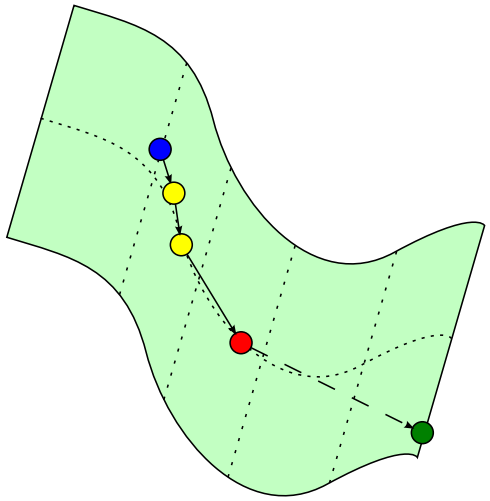
Testing projection continuity

- The initial path is sampled and successive samples are projected,
-
-



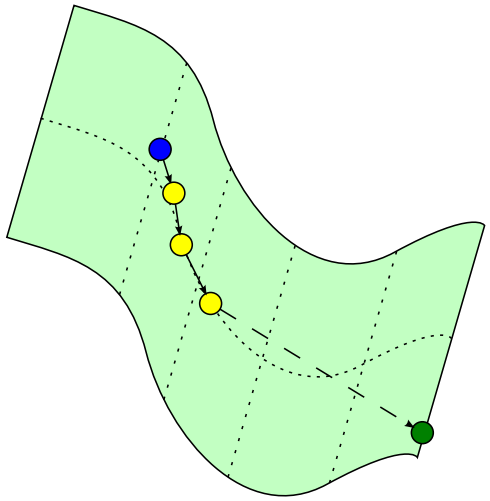
Testing projection continuity

- The initial path is sampled and successive samples are projected,
- if 2 successive projections are too far away, an intermediate sample is selected.
-



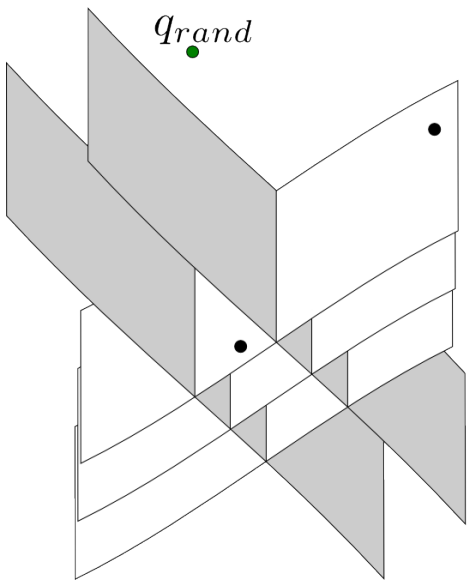
Testing projection continuity

- The initial path is sampled and successive samples are projected,
- if 2 successive projections are too far away, an intermediate sample is selected.
- Choosing appropriate sampling ensures us continuity of the projection.



Algorithm

Manipulation RRT



Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()

for each connected component:

$\mathbf{q}_{near}$ = nearest_neighbor($\mathbf{q}_{rand}$, *roadmap*)

T = select_transition($\mathbf{q}_{near}$)

$\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, T)

$\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)

roadmap.insert_node($\mathbf{q}_{new}$)

roadmap.insert_edge(T, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)

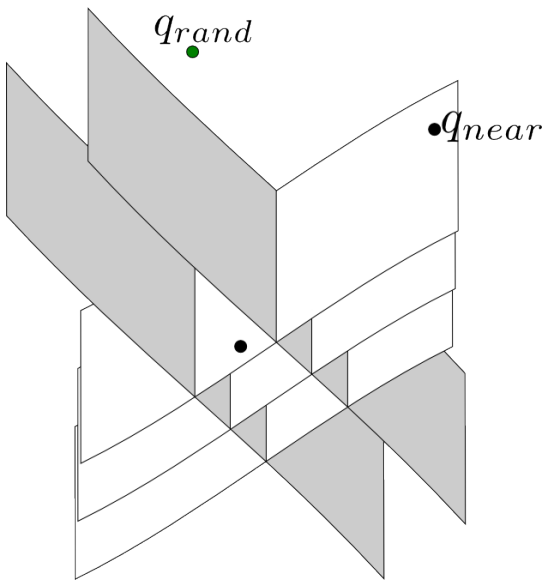
new_nodes.append ($\mathbf{q}_{new}$)

for $\mathbf{q} \in (\mathbf{q}^1_{new}, \ldots, \mathbf{q}^{n_{cc}}_{new})$:

connect ($\mathbf{q}$, roadmap)

Algorithm

Manipulation RRT

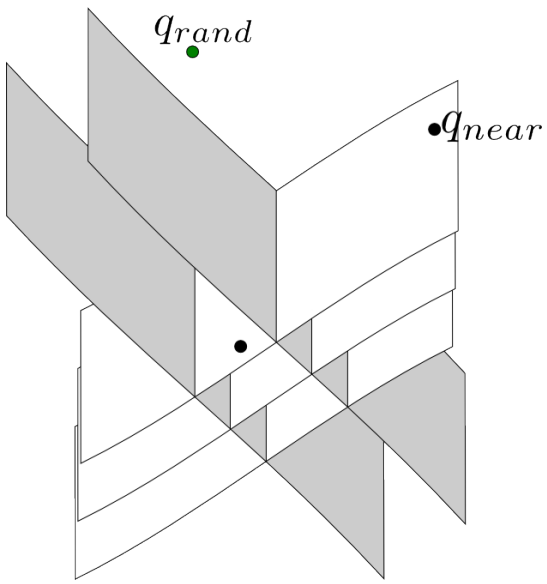


Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:
    $\mathbf{q}_{near}$ = nearest_neighbor($\mathbf{q}_{rand}$, *roadmap*)
    T = select_transition($\mathbf{q}_{near}$)
    $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, T)
    $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)
    roadmap.insert_node($\mathbf{q}_{new}$)
    roadmap.insert_edge(T, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
    new_nodes.append ($\mathbf{q}_{new}$)
for $\mathbf{q} \in (\mathbf{q}^1_{new}, \ldots, \mathbf{q}^{n_{cc}}_{new})$:
    connect ($\mathbf{q}$, roadmap)

Algorithm

Manipulation RRT



Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()

for each connected component:

$\mathbf{q}_{near}$ = nearest_neighbor($\mathbf{q}_{rand}$, *roadmap*)

T = select_transition($\mathbf{q}_{near}$)

$\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, T)

$\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)

roadmap.insert_node($\mathbf{q}_{new}$)

roadmap.insert_edge(T, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)

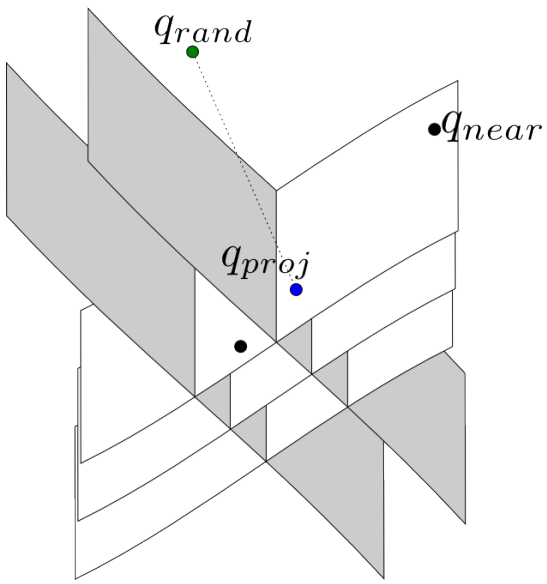
new_nodes.append ($\mathbf{q}_{new}$)

for $\mathbf{q} \in (\mathbf{q}^1_{new}, \ldots, \mathbf{q}^{n_{cc}}_{new})$:

connect ($\mathbf{q}$, roadmap)

Algorithm

Manipulation RRT

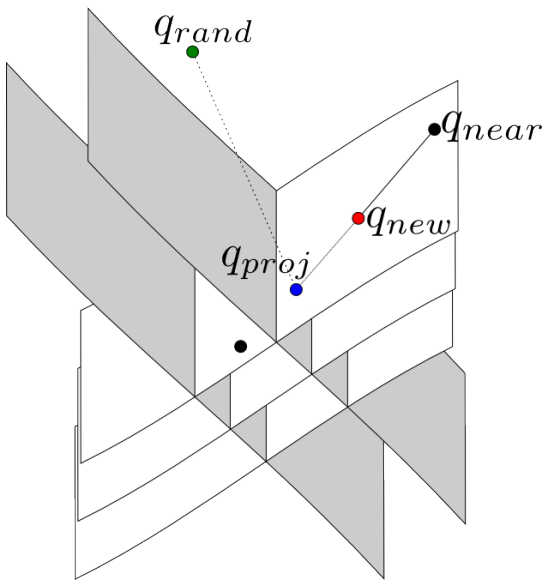


Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:
    $\mathbf{q}_{near}$ = nearest_neighbor($\mathbf{q}_{rand}$, *roadmap*)
    T = select_transition($\mathbf{q}_{near}$)
    $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, T)
    $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)
    roadmap.insert_node($\mathbf{q}_{new}$)
    roadmap.insert_edge(T, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
    new_nodes.append ($\mathbf{q}_{new}$)
for $\mathbf{q} \in (\mathbf{q}^1_{new}, \ldots, \mathbf{q}^{n_{cc}}_{new})$:
    connect ($\mathbf{q}$, roadmap)

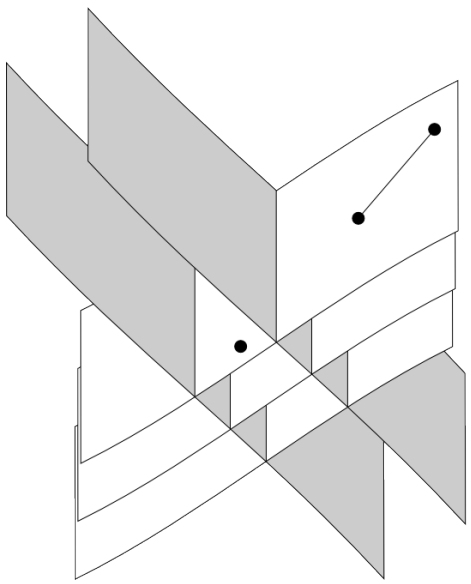
Algorithm

Manipulation RRT



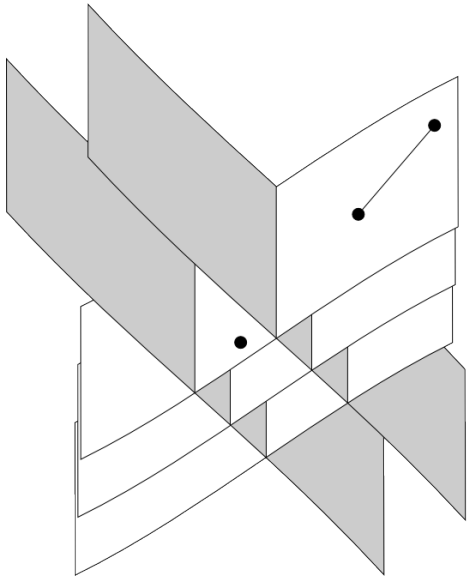
Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:
    $\mathbf{q}_{near}$ = nearest_neighbor($\mathbf{q}_{rand}$, *roadmap*)
    T = select_transition($\mathbf{q}_{near}$)
    $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, T)
    $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)
    roadmap.insert_node($\mathbf{q}_{new}$)
    roadmap.insert_edge(T, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
    new_nodes.append ($\mathbf{q}_{new}$)
for $\mathbf{q} \in (\mathbf{q}^1_{new}, \ldots, \mathbf{q}^{n_{cc}}_{new})$:
    connect ($\mathbf{q}$, roadmap)



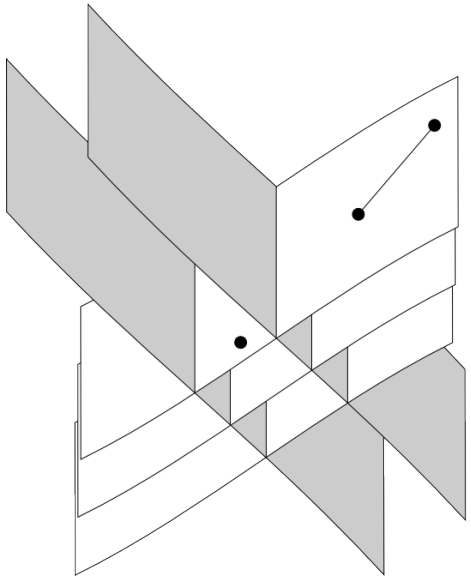
Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:
    $\mathbf{q}_{near}$ = nearest_neighbor($\mathbf{q}_{rand}$, *roadmap*)
    T = select_transition($\mathbf{q}_{near}$)
    $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, T)
    $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)
    roadmap.insert_node($\mathbf{q}_{new}$)
    roadmap.insert_edge(T, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
    new_nodes.append ($\mathbf{q}_{new}$)
for $\mathbf{q} \in (\mathbf{q}^1_{new}, \ldots, \mathbf{q}^{n_{cc}}_{new})$:
    connect ($\mathbf{q}$, roadmap)



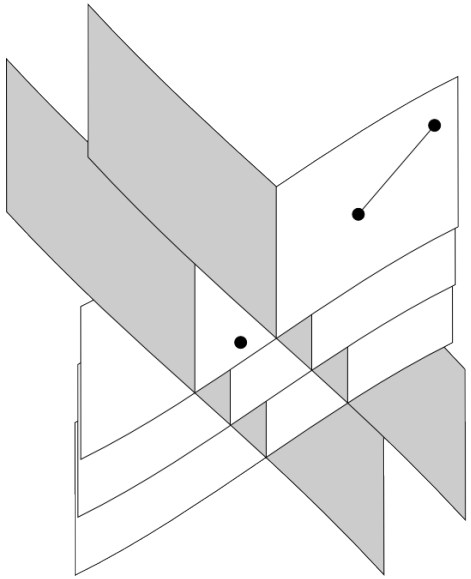
Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:
    $\mathbf{q}_{near}$ = nearest_neighbor($\mathbf{q}_{rand}$, *roadmap*)
    T = select_transition($\mathbf{q}_{near}$)
    $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, T)
    $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)
    roadmap.insert_node($\mathbf{q}_{new}$)
    roadmap.insert_edge(T, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
    new_nodes.append ($\mathbf{q}_{new}$)
for $\mathbf{q} \in (\mathbf{q}^1_{new}, \ldots, \mathbf{q}^{n_{cc}}_{new})$:
    connect ($\mathbf{q}$, roadmap)



Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:
    $\mathbf{q}_{near}$ = nearest_neighbor($\mathbf{q}_{rand}$, *roadmap*)
    T = select_transition($\mathbf{q}_{near}$)
    $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, T)
    $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)
    roadmap.insert_node($\mathbf{q}_{new}$)
    roadmap.insert_edge(T, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
    new_nodes.append ($\mathbf{q}_{new}$)
for $\mathbf{q} \in (\mathbf{q}^1_{new}, \ldots, \mathbf{q}^{n_{cc}}_{new})$:
    connect ($\mathbf{q}$, roadmap)



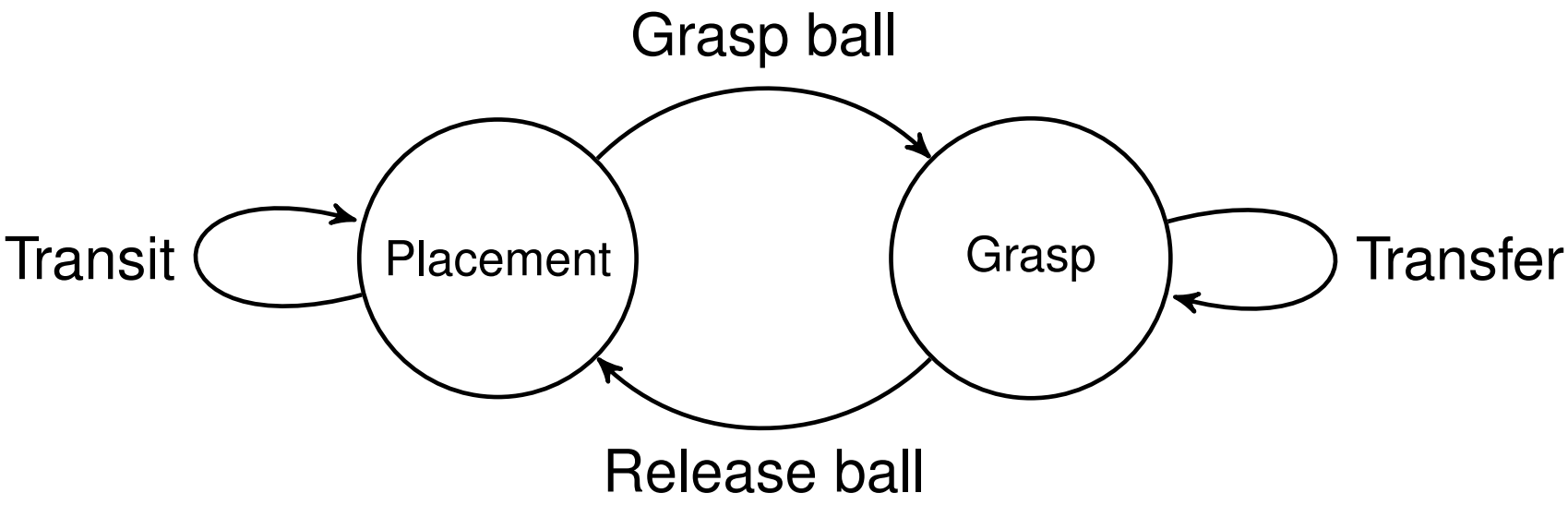
Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:
    $\mathbf{q}_{near}$ = nearest_neighbor($\mathbf{q}_{rand}$, *roadmap*)
    T = select_transition($\mathbf{q}_{near}$)
    $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, T)
    $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)
    roadmap.insert_node($\mathbf{q}_{new}$)
    roadmap.insert_edge(T, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
    new_nodes.append ($\mathbf{q}_{new}$)
for $\mathbf{q} \in (\mathbf{q}^1_{new}, \ldots, \mathbf{q}^{n_{cc}}_{new})$:
    connect ($\mathbf{q}$, roadmap)

Select transition

T = select_transition($\mathbf{q}_{near}$)

Outward transitions of each state are given a probability distribution. The transition from a state to another state is chosen by random sampling.



Generate target configuration

$\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}, \mathbf{q}_{rand}, T$)

Once transition T has been selected, $\mathbf{q}_{rand}$ is *projected* onto the destination state S_{dest} in a configuration reachable by $\mathbf{q}_{near}$.

$$f_T(\mathbf{q}_{proj}) = f_T(\mathbf{q}_{near})$$
$$f_{S_{dest}}(\mathbf{q}_{proj}) = 0$$

Extend

$\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, T)

Project straight path $[\mathbf{q}_{near}, \mathbf{q}_{proj}]$ on transition constraint:

- if projection successful and projected path collision free

$$\mathbf{q}_{new} \leftarrow \mathbf{q}_{proj}$$

- otherwise $(\mathbf{q}_{near}, \mathbf{q}_{new}) \leftarrow$ largest path interval tested as collision-free with successful projection.

$$\forall \mathbf{q} \in (\mathbf{q}_{near}, \mathbf{q}_{new}),\ f_T(\mathbf{q}) = f_T(\mathbf{q}_{near})$$

Connect

connect (**q**, roadmap)

for each connected component *cc* not containing **q**:

for all *n* closest config $\mathbf{q}_1$ to **q** in *cc*:

- connect (**q**, $\mathbf{q}_1$)

Connect

connect ($\mathbf{q}_0$, $\mathbf{q}_1$):

S_0 = state ($\mathbf{q}_0$)

S_1 = state ($\mathbf{q}_1$)

T = transition (S_0, S_1)

if T and $f_T(\mathbf{q}_0) == f_T(\mathbf{q}_1)$:

if p = projected_path (T, $\mathbf{q}_0$, $\mathbf{q}_1$) collision-free:

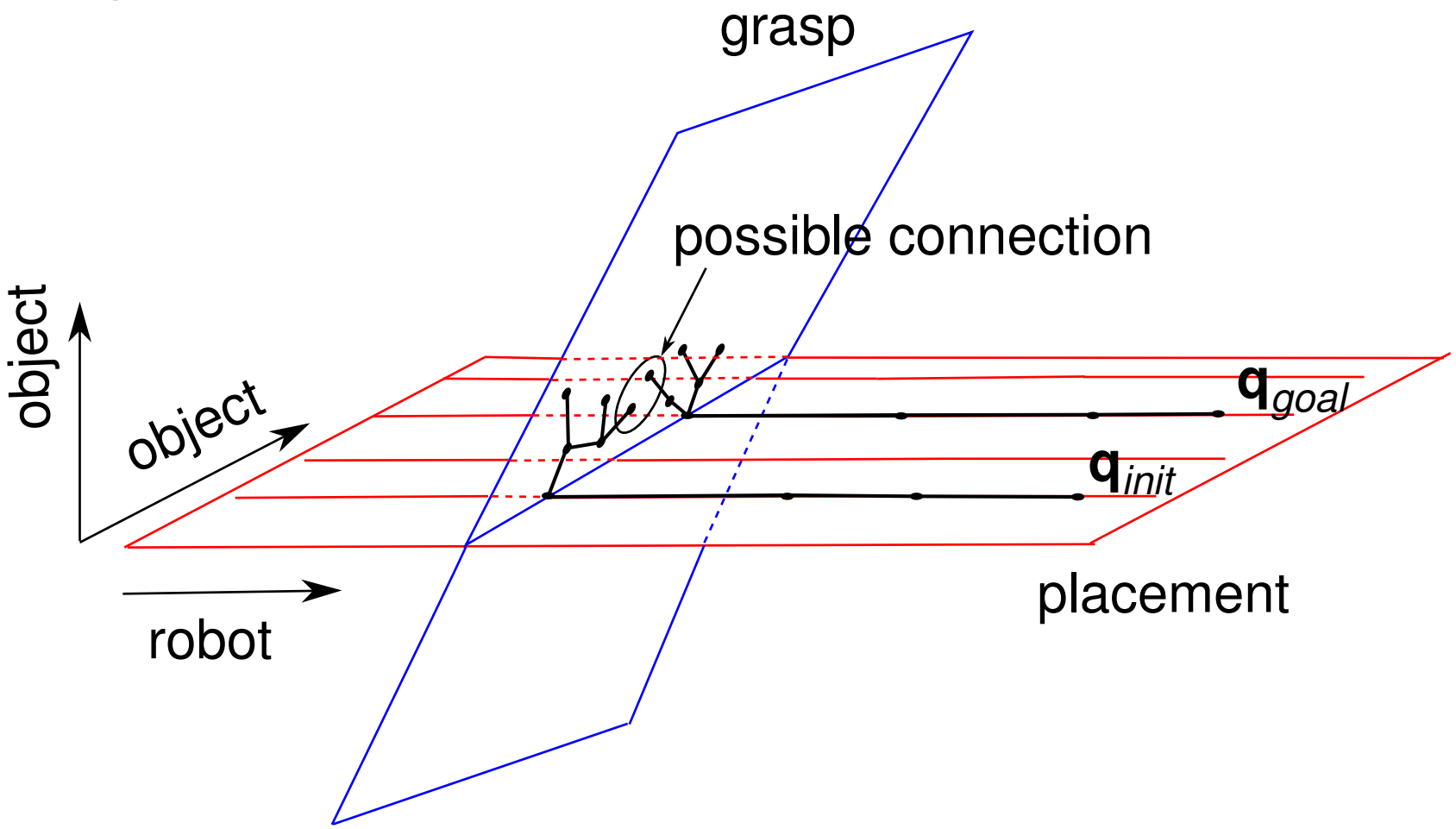
roadmap.insert_edge (T, $\mathbf{q}_0$,$\mathbf{q}_1$)

return

Connecting trees

Manipulation RRT is initialized with $\mathbf{q}_{init}$, $\mathbf{q}_{goal}$.

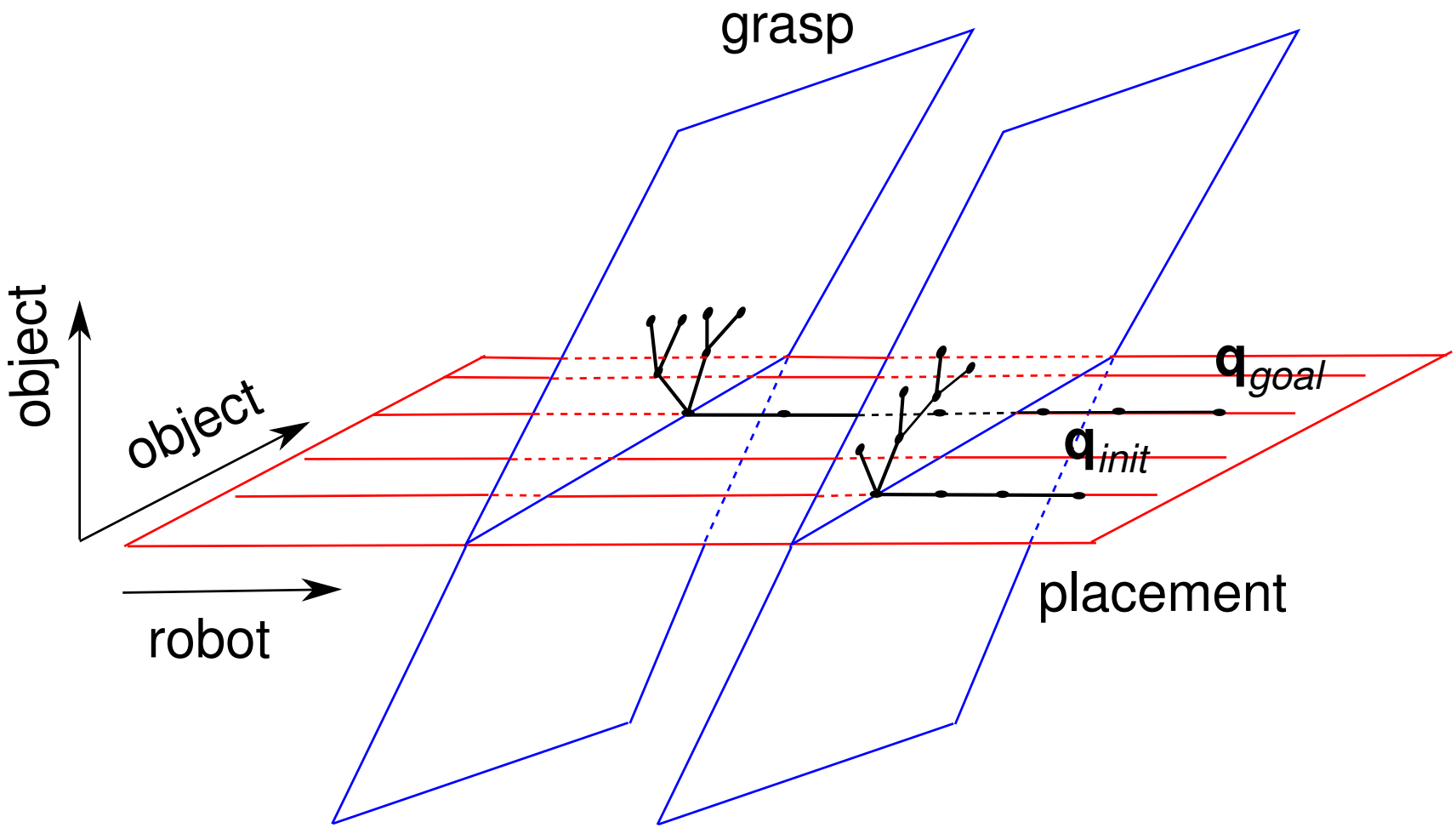
- 2 connected components.
- possible connection.



Connecting trees: general case

Manipulation RRT is initialized with $\mathbf{q}_{init}$, $\mathbf{q}_{goal}$.

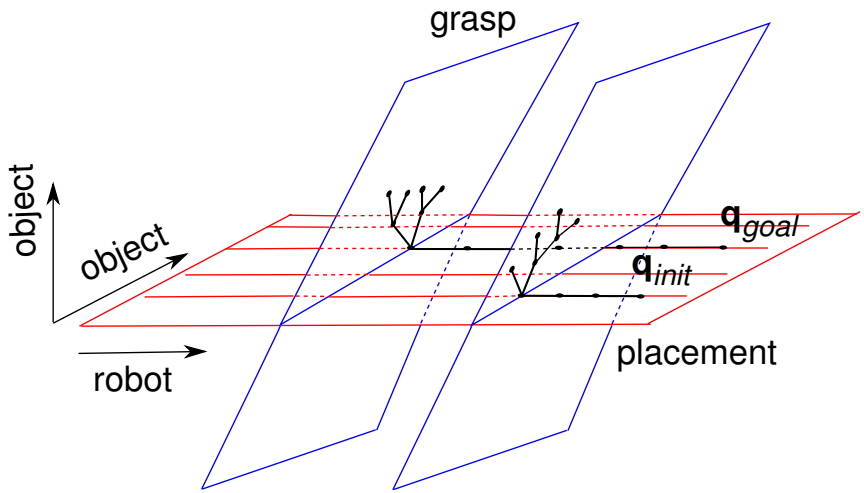
- 2 connected components,
- no possible connection.

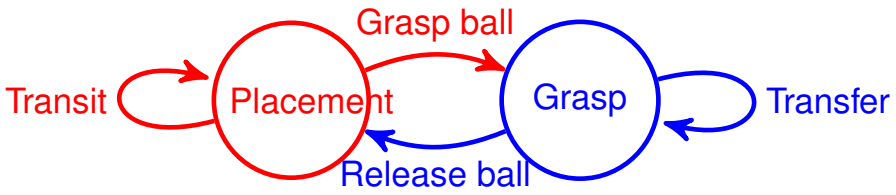


Connecting trees: general case

Manipulation RRT is initialized with $\mathbf{q}_{init}$, $\mathbf{q}_{goal}$.

- 2 connected components,
- no possible connection.





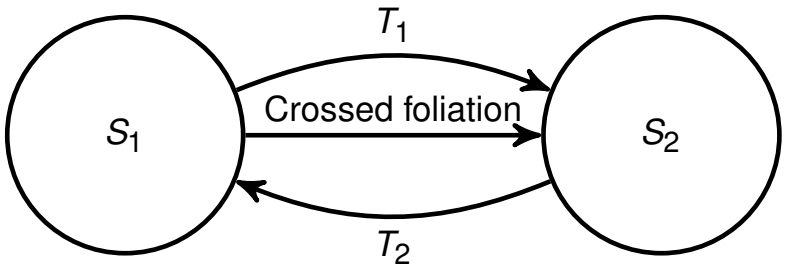
Crossed foliation transition: generate target configuration

$\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}, \mathbf{q}_{rand}, T$)

$\mathbf{q}_1 \leftarrow$ pick configuration

- in state S_1,
- not in same connected component as $\mathbf{q}_{near}$

$$f_{T_1}(\mathbf{q}_{proj}) = f_{T_1}(\mathbf{q}_{near})$$
$$f_{T_2}(\mathbf{q}_{proj}) = f_{T_2}(\mathbf{q}_1)$$
$$f_{S_2}(\mathbf{q}_{proj}) = 0$$



Crossed foliation transition: extend

$\mathbf{q}_{new} = \text{extend}(\mathbf{q}_{near}, \mathbf{q}_{proj}, T_1)$

Project straight path $[\mathbf{q}_{near}, \mathbf{q}_{proj}]$ on T_1 constraint:

- if projection successful and projected path collision free

$$\mathbf{q}_2 \leftarrow \mathbf{q}_{proj}$$

$$f_{T_2}(\mathbf{q}_2) = f_{T_2}(\mathbf{q}_1)$$
$$f_{S_2}(\mathbf{q}_2) = 0$$

- $\mathbf{q}_2$ is connectable to $\mathbf{q}_1$ via T_2.

Relative positions as numerical constraints

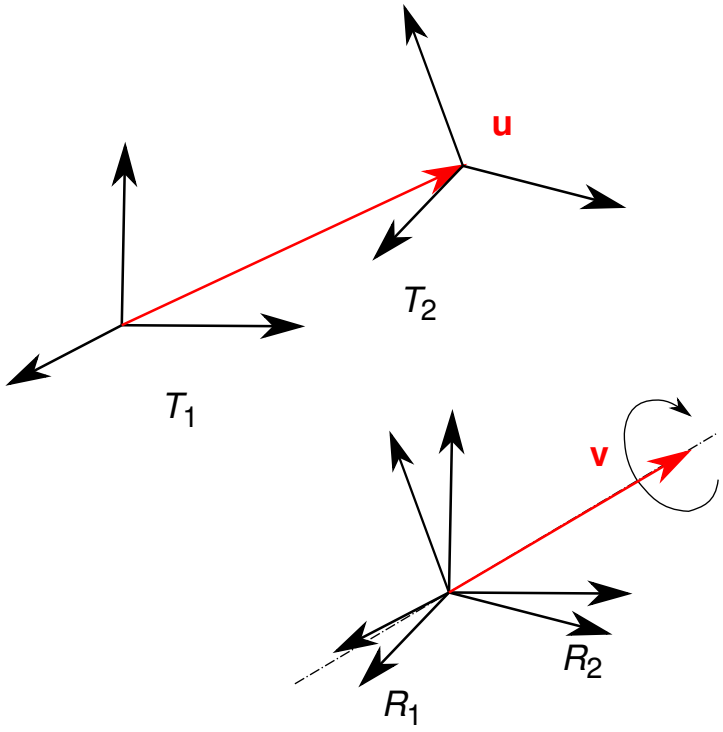
- $T_1 = T_{(R_1,t_1)} \in SE(3)$,
  $T_2 = T_{(R_2,t_2)} \in SE(3)$.
- $T_{2/1} = T_1^{-1} \circ T_2$ can be represented by a vector of dimension 6:

$$\begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}$$

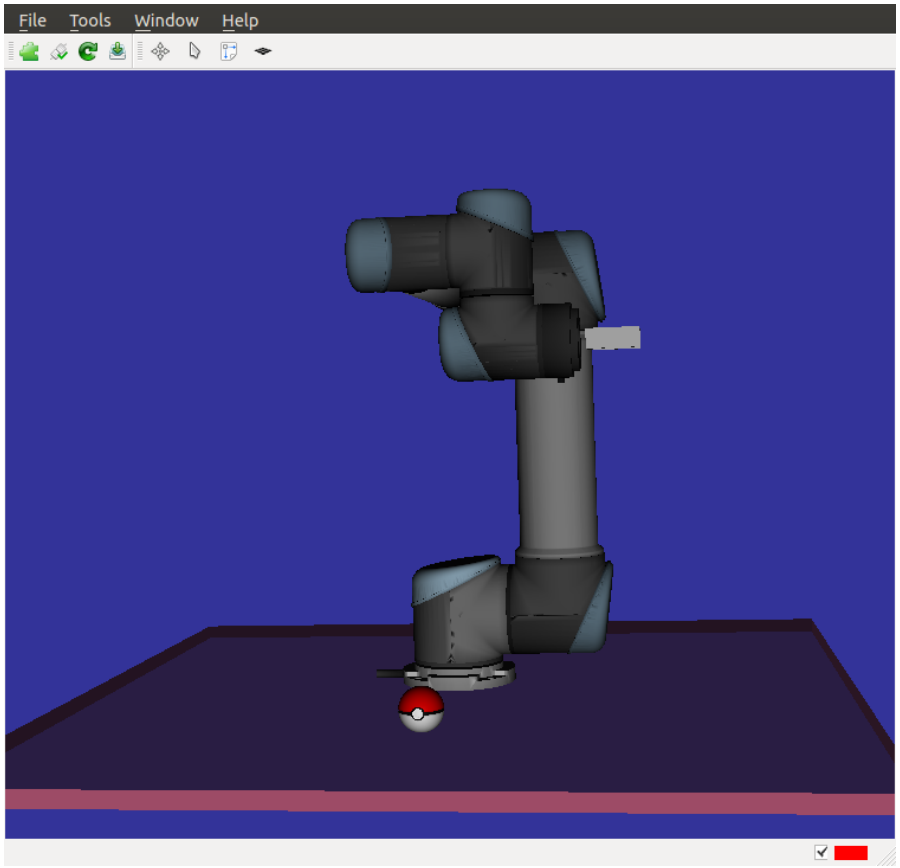
where

$$\mathbf{u} = R_1^T (t_2 - t_1)$$

$R_1^T R_2$ matrix of the rotation around axis $\mathbf{v}/\|\mathbf{v}\|$ and of angles $\|\mathbf{v}\|$.



A few words about the BE



- `script/grasp_ball.py`