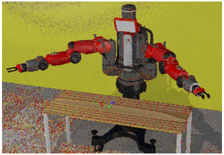
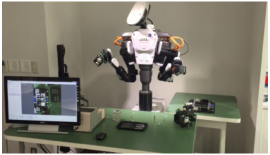
Manipulation motion planning

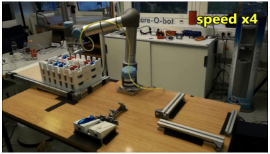
Florent Lamiraux

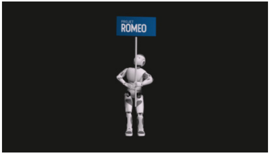
CNRS-LAAS, Toulouse, France

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.

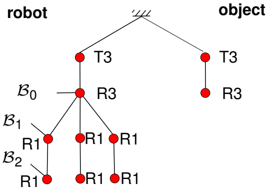
Definitions

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



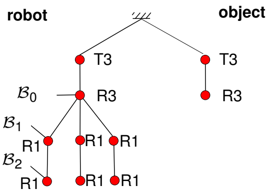
$\mathbf{q} = (q_0, \cdots, q_{n_r}, q_{n_r+1}, \cdots, q_{n_r+n_o})$

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

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

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.

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

Numerical constraints

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

- Numerical constraints:

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

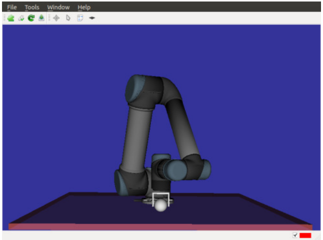
  - `setConstantRightHandSide(True)`

- Parameterizable numerical constraints:

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

  - `setConstantRightHandSide(False)`

Example: robot manipulating a ball



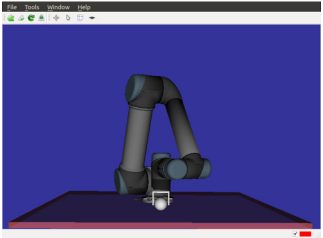
$$\mathcal{C} = [-\pi, \pi]^6 \times \mathbb{R}^3 \qquad (1)$$

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

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

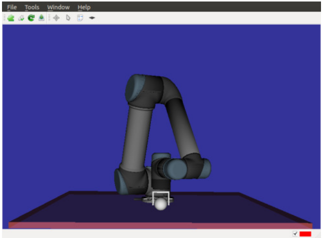
$$z_b = 0$$

- 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



Each state is defined by a numerical constraint

- `placement`

$$z_b = 0$$

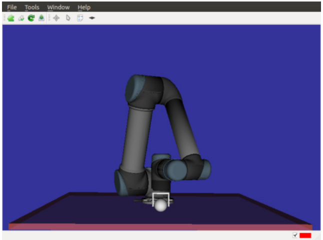
- `grasp`

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

Each state is a sub-manifold of the configuration space

Example: robot manipulating a ball

Motion constraints

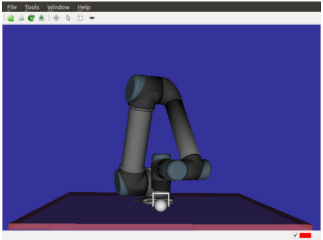


Two *types of motion*:

- `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`

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

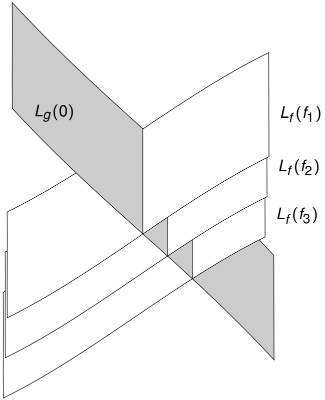
$$\left. z_b = 0 \right\} \text{ simple}$$

- `transfer`

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

Foliation

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



- f: position of the ball

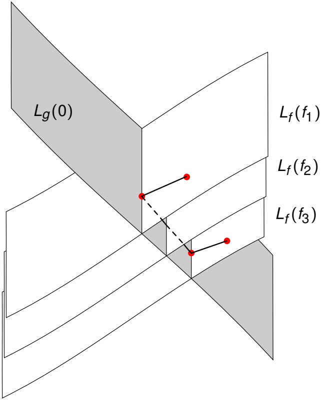
$$L_f(f_1) = \{\mathbf{q} \in \mathcal{C}, f(\mathbf{q}) = f_1\}$$

- 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` ∪ `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.

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.

General case

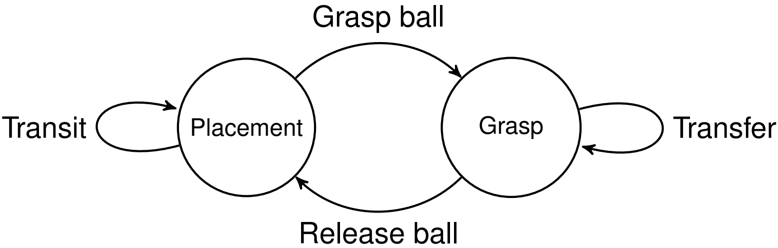
In a manipulation problem,

- 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

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.

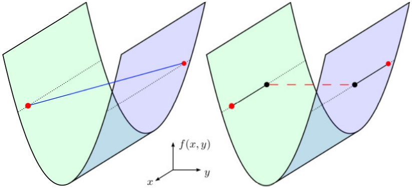
A constrained steering method $\mathcal{S}\bar{\mathcal{M}}$ of constraint f is a steering method such that

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

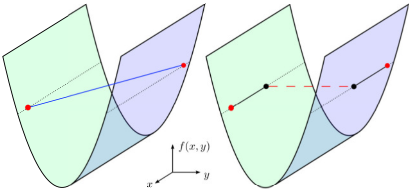
Projecting path on constraint

- *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



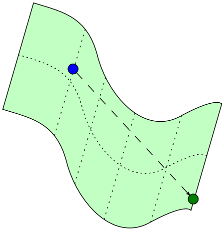
$$\mathcal{C} = \mathbb{R}^2, f(x,y) = y^2 - 1$$

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

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

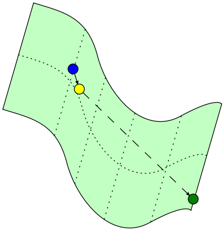
Testing projection continuity

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



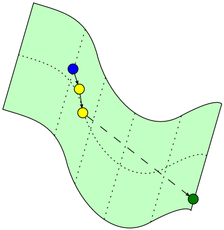
Testing projection continuity

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



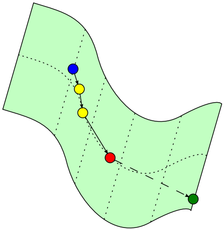
Testing projection continuity

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



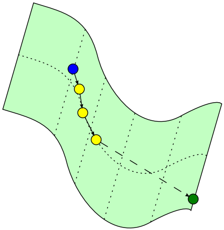
Testing projection continuity

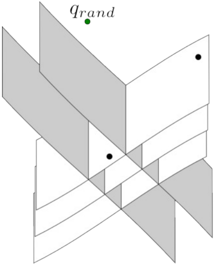
- The initial path is sampled and successive samples are projected,
- if 2 successive projections are too far away, an intermediate sample is selected.
-



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.





Manipulation RRT

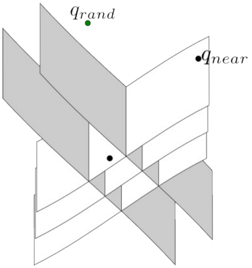
$\mathbf{q}_{rand}$ = shoot_random_config()

for each connected component:

- $\mathbf{q}_{near}$ = nearest_neighbour($\mathbf{q}_{rand}$, *roadmap*)
- e = select_transition($\mathbf{q}_{near}$)
- $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, e)
- $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, edge)
- *roadmap*.insert_node($\mathbf{q}_{new}$)
- *roadmap*.insert_edge(e, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
- new_nodes.append ($\mathbf{q}_{new}$)

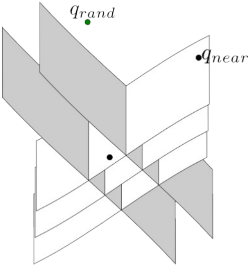
for $\mathbf{q} \in (\mathbf{q}^1_{new}, \dots, \mathbf{q}^{n_{cc}}_{new})$:

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



Manipulation RRT

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



Manipulation RRT

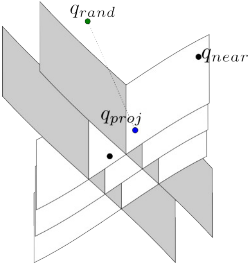
$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:

  $\mathbf{q}_{near}$ = nearest_neighbour($\mathbf{q}_{rand}$, *roadmap*)
  e = select_transition($\mathbf{q}_{near}$)
  $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, e)
  $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, edge)
  roadmap.insert_node($\mathbf{q}_{new}$)
  roadmap.insert_edge(e, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
  new_nodes.append ($\mathbf{q}_{new}$)

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

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

Algorithm
Manipulation RRT



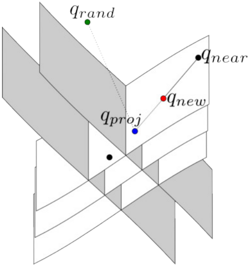
Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:

- $\mathbf{q}_{near}$ = nearest_neighbour($\mathbf{q}_{rand}$, *roadmap*)
- *e* = select_transition($\mathbf{q}_{near}$)
- $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, *e*)
- $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, edge)
- *roadmap*.insert_node($\mathbf{q}_{new}$)
- *roadmap*.insert_edge(e, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
- new_nodes.append ($\mathbf{q}_{new}$)

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

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



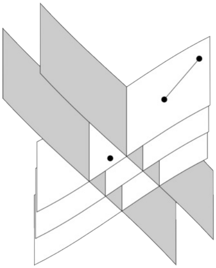
Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:

$\mathbf{q}_{near}$ = nearest_neighbour($\mathbf{q}_{rand}$, *roadmap*)
e = select_transition($\mathbf{q}_{near}$)
$\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, *e*)
$\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, edge)
roadmap.insert_node($\mathbf{q}_{new}$)
roadmap.insert_edge(e, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
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:

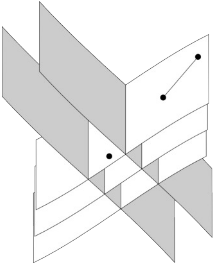
- $\mathbf{q}_{near}$ = nearest_neighbour($\mathbf{q}_{rand}$, *roadmap*)
- e = select_transition($\mathbf{q}_{near}$)
- $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, e)
- $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, edge)
- *roadmap*.insert_node($\mathbf{q}_{new}$)
- *roadmap*.insert_edge(e, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
- new_nodes.append ($\mathbf{q}_{new}$)

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

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

Algorithm

Manipulation RRT



Manipulation RRT

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

for each connected component:

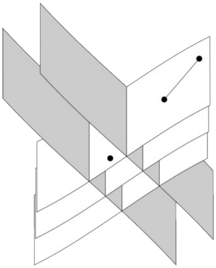
> $\mathbf{q}_{near}$ = nearest_neighbour($\mathbf{q}_{rand}$, *roadmap*)
> *e* = select_transition($\mathbf{q}_{near}$)
> $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, *e*)
> $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, edge)
> *roadmap*.insert_node($\mathbf{q}_{new}$)
> *roadmap*.insert_edge(e, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
> 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

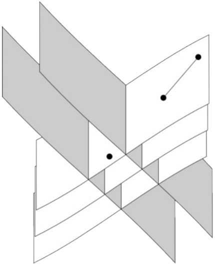
Manipulation RRT



Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:
- $\mathbf{q}_{near}$ = nearest_neighbour($\mathbf{q}_{rand}$, *roadmap*)
- *e* = select_transition($\mathbf{q}_{near}$)
- $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}$, $\mathbf{q}_{rand}$, *e*)
- $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, edge)
- *roadmap*.insert_node($\mathbf{q}_{new}$)
- *roadmap*.insert_edge(e, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
- new_nodes.append ($\mathbf{q}_{new}$)

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



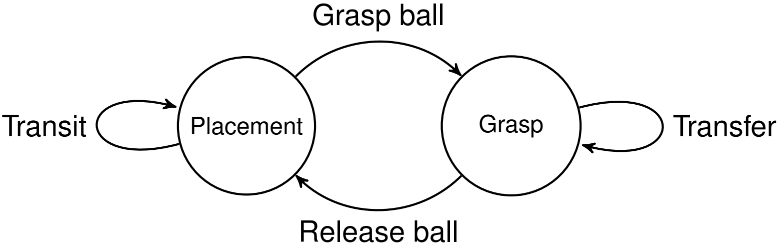
Manipulation RRT

$\mathbf{q}_{rand}$ = shoot_random_config()
for each connected component:
  $\mathbf{q}_{near}$ = nearest_neighbour($\mathbf{q}_{rand}$, *roadmap*)
  e = select_transition($\mathbf{q}_{near}$)
  $\mathbf{q}_{proj}$ = generate_target_config($\mathbf{q}_{near}, \mathbf{q}_{rand}$, e)
  $\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, edge)
  roadmap.insert_node($\mathbf{q}_{new}$)
  roadmap.insert_edge(e, $\mathbf{q}_{near}$, $\mathbf{q}_{new}$)
  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

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

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



Generate target configuration

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

Once edge e has been selected, $\mathbf{q}_{rand}$ is *projected* onto the destination node n_{dest} in a configuration reachable by $\mathbf{q}_{near}$.

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

Extend

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

Project straight path $[\mathbf{q}_{near}, \mathbf{q}_{proj}]$ on edge 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.

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

Extend

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

Project straight path $[\mathbf{q}_{near}, \mathbf{q}_{proj}]$ on edge 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.

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

Extend

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

Project straight path $[\mathbf{q}_{near}, \mathbf{q}_{proj}]$ on edge 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.

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

Connect

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

for each connected component *cc* not containing $\mathbf{q}$:

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

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

Connect

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

s_0 = state ($\mathbf{q}_0$)

s_1 = state ($\mathbf{q}_1$)

e = transition (n_0, n_1)

if e and $f_e(\mathbf{q}_0) == f_e(\mathbf{q}_1)$:

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

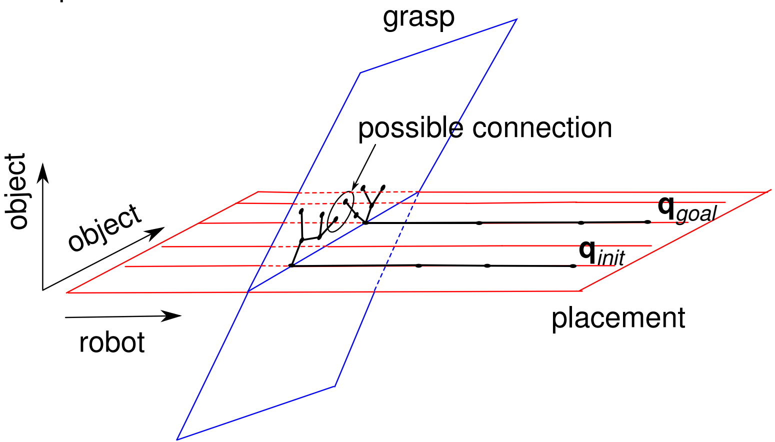
roadmap.insert_edge (e, $\mathbf{q}_0$,$\mathbf{q}_1$)

return

Connecting trees

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

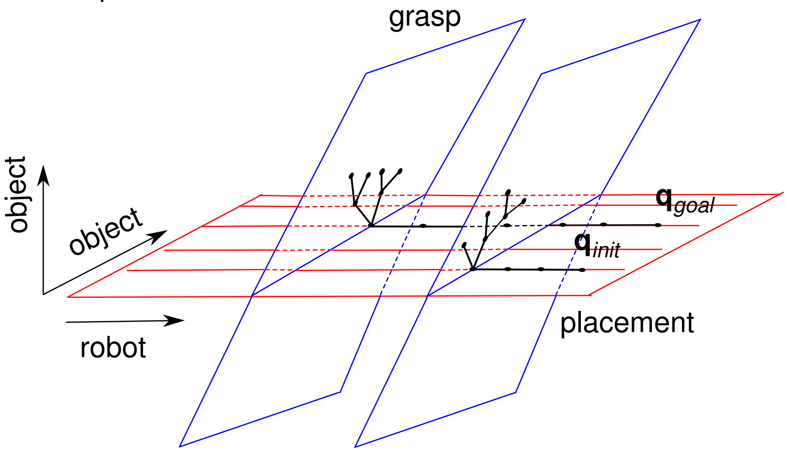
- 2 connected components.
- possible connection.



Connecting trees: general case

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

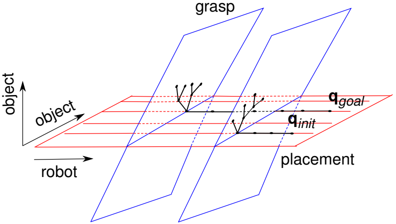
- 2 connected components,
- no possible connection.

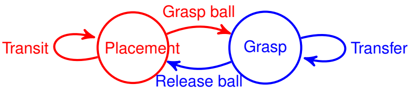


Connecting trees: general case

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

- 2 connected components,
- no possible connection.





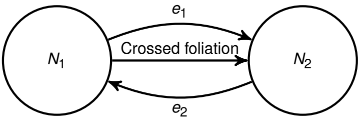
Crossed foliation transition: generate target configuration

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

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

- in node N_1,
- not in same connected component as $\mathbf{q}_{near}$

$$f_{e_1}(\mathbf{q}_{proj}) = f_{e_1}(\mathbf{q}_{near})$$
$$f_{e_2}(\mathbf{q}_{proj}) = f_{e_2}(\mathbf{q}_1)$$
$$f_{N_2}(\mathbf{q}_{proj}) = 0$$



Crossed foliation transition: extend

$\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, e_1)

Project straight path $[\mathbf{q}_{near}, \mathbf{q}_{proj}]$ on e_1 constraint:

- if projection successful and projected path collision free

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

$$f_{e_2}(\mathbf{q}_2) = f_{e_2}(\mathbf{q}_1)$$
$$f_{N_2}(\mathbf{q}_2) = 0$$

- $\mathbf{q}_2$ is connectable to $\mathbf{q}_1$ via e_2.

Crossed foliation transition: extend

$\mathbf{q}_{new}$ = extend($\mathbf{q}_{near}$, $\mathbf{q}_{proj}$, e_1)

Project straight path $[\mathbf{q}_{near}, \mathbf{q}_{proj}]$ on e_1 constraint:

- if projection successful and projected path collision free

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

$$f_{e_2}(\mathbf{q}_2) = f_{e_2}(\mathbf{q}_1)$$
$$f_{N_2}(\mathbf{q}_2) = 0$$

- $\mathbf{q}_2$ is connectable to $\mathbf{q}_1$ via e_2.

Relative positions as numerical constraints

Let $T_1 = T_{(R_1,t_1)}$ and $T_2 = T_{(R_2,t_2)}$ be two rigid-body transformations. The relative transformation $T_{2/1} = T_1^{-1} \circ T_2$ can be represented by a vector of dimension 6:

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

where

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

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

A few words about the BE