Motion Planning for Robot Manipulators

Manuel Bonilla

PhD Student

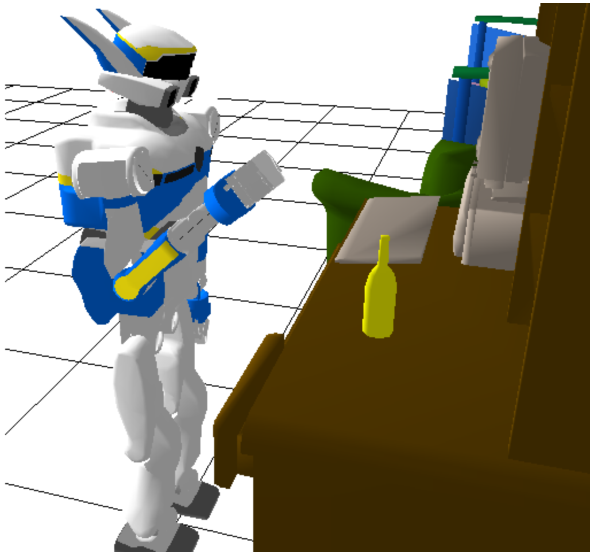
manuel.bonilla@centropiaggio.unipi.it

http://manuel-bonilla.com

1. What is the interesting part of Motion planning in Robotics

Ok, lets first talk about problem faced in robotics doing motion planning. Robotics includes:

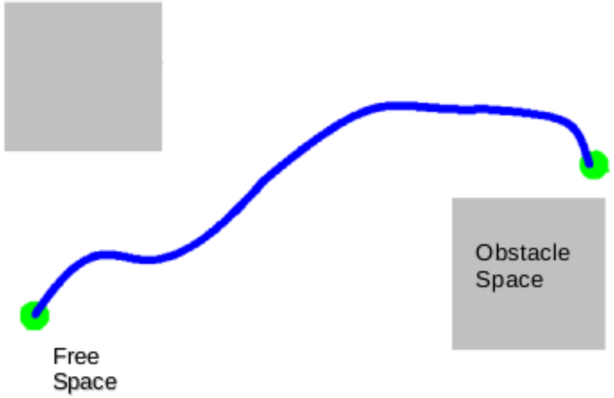
- **Perception**: Get information about the environment using sensors (Cameras, Laser…)
- **Planning**: Decide the steps to follow in order to perform a task.
- **Execution**: Control the robot to perform this action (Prof Bicchi talked about this: Computed torque, Back steeping, Arimoto, Feedforward control, Sliding modes … OMG)



The Motion Planning Problem

- Consider a Configuration Space (CS) as a compact set of $q_i \in R^d$ elements called configurations. Defining the obstacle region as $CS_{obs} \in CS$, $CS_{free} := CS \setminus CS_{obs}$ is the free region. Thus, we need to find a continuous path

$$\sigma : [0,1] \rightarrow CS_{free} \mid \{\sigma(0) = q_{ini}, \sigma(1) = q_{final}\}$$



… just recalling

- The three main philosophies for addressing the motion planning problem are:

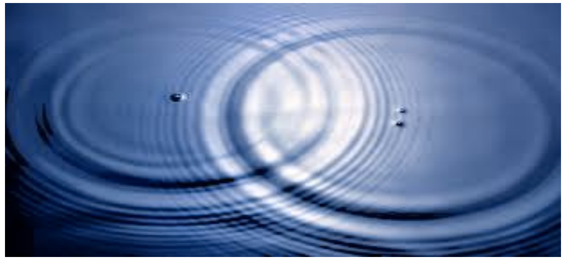
  - Combinatorial planning (exact Planning)
  - Sampling-based planning
  - Artificial potential fields methods

State of the art in combinatorial planning (cell decompositions)

- Fast Marching Method (FMM)
  - It is based on the solution of Eikonal equations over a grid

$$\max\left(\frac{T-T_1}{\Delta x},0\right)^2+\max\left(\frac{T-T_2}{\Delta y},0\right)^2=\frac{1}{F_{I,J}^2}$$

$$\left(\frac{T-T_1}{\Delta x}\right)^2+\left(\frac{T-T_2}{\Delta y}\right)^2=\frac{1}{F_{I,J}^2}$$

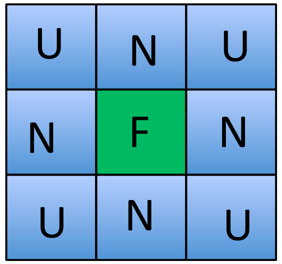


FMM: Algorithm

input : A grid map G of size $m \times n$
input : The set of cells Ori where the wave is originated
output: The grid map G with the T value set for all cells

Initialization

```
foreach g_ij ∈ Ori do
    g_ij.T ← 0;
    g_ij.state ← FROZEN;
    foreach g_kl ∈ g_ij.neighbours do
        if g_kl = FROZEN then skip; else
            g_kl.T ← solveEikonal(g_kl);
            if g_kl.state = NARROW BAND then
            narrow_band.update_position(g_kl);
            if g_kl.state = UNKNOWN then
                g_kl.state ← NARROW BAND;
                narrow_band.insert_in_position(g_kl);
            end
        end
end
```

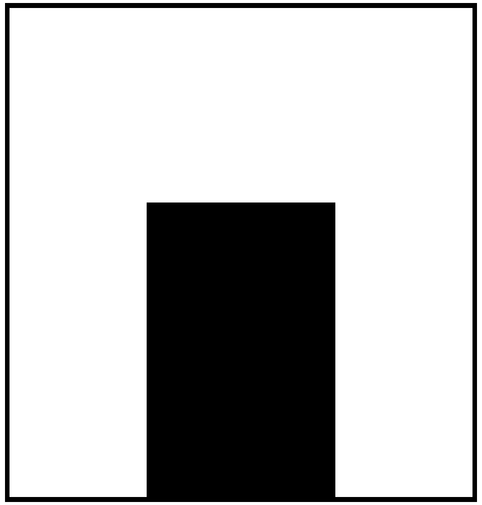


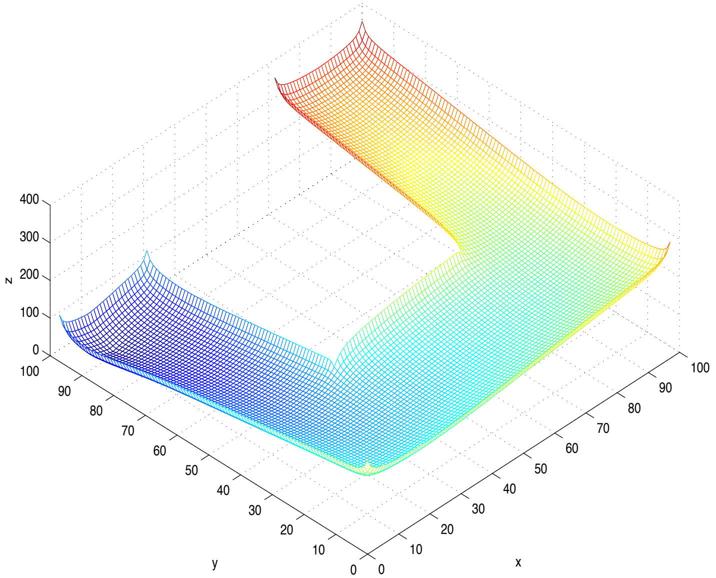
FMM: Algorithm - II

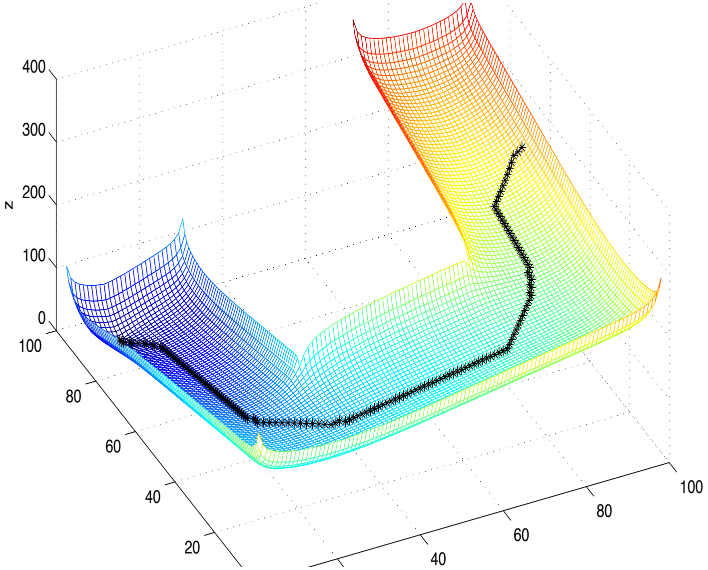
Iterations

```
while narrow_band NOT EMPTY do
    g_ij ← narrow_band.pop_first();
    foreach g_kl ∈ g_ij.neighbours do
        if g_kl = FROZEN then skip; else
            g_kl.T ← solveEikonal(g_kl);
            if g_kl.state = NARROW BAND then
            narrow_band.update_position(g_kl);
            if g_kl.state = UNKNOWN then
                g_kl.state ← NARROW BAND;
                narrow_band.insert_in_position(g_kl);
            end
        end
    end
end
end
```

State of the art in combinatorial planning (cell decompositions)







Advantages:

Optimality is guaranteed

Can be applied over manifolds

Disadvantages

Resolution complete

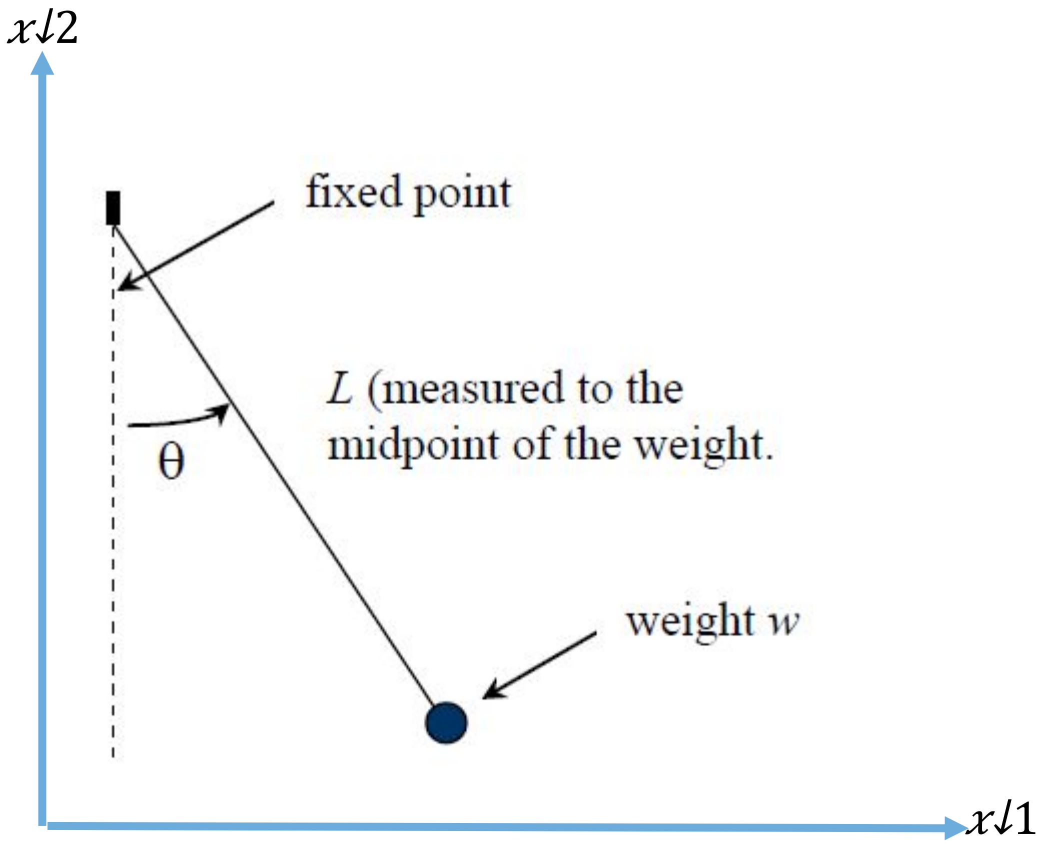
Slow in high dimensional spaces

… just recalling

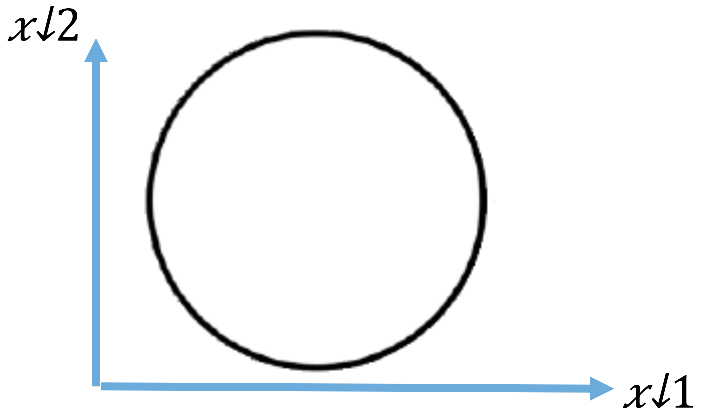
- The three main philosophies for addressing the motion planning problem are:

  - Combinatorial planning (exact Planning)
  - Sampling-based planning
  - Artificial potential fields methods

The Configuration Space

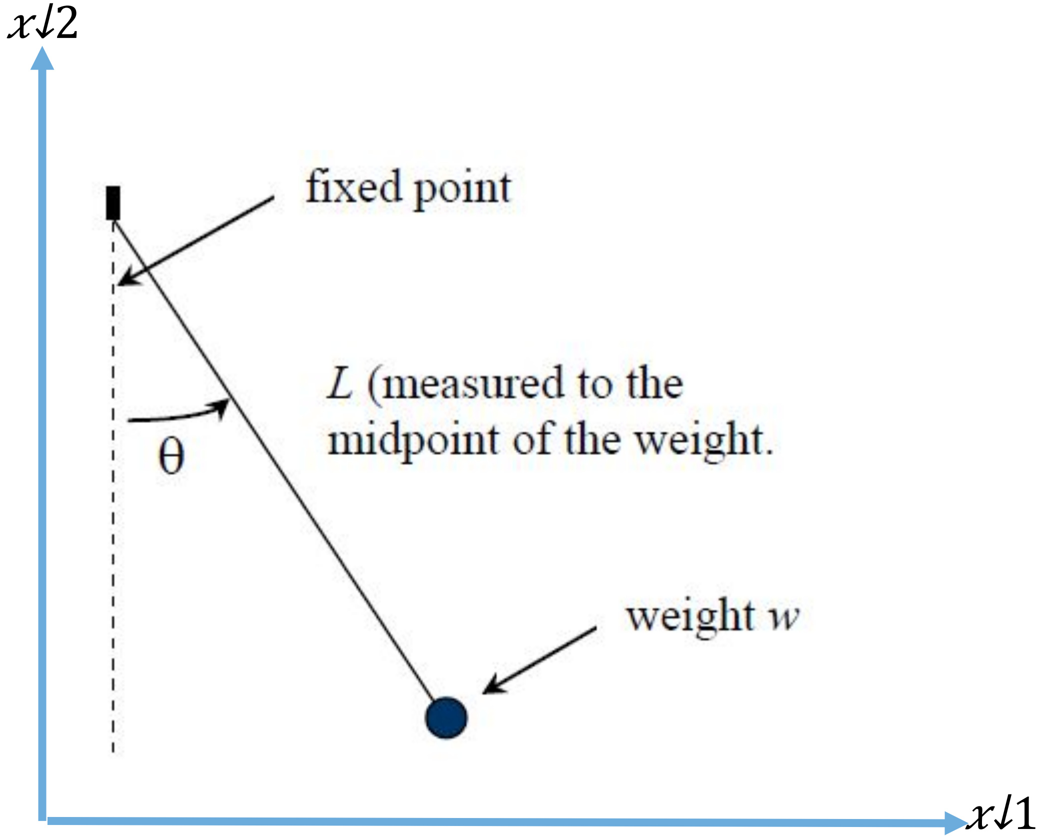


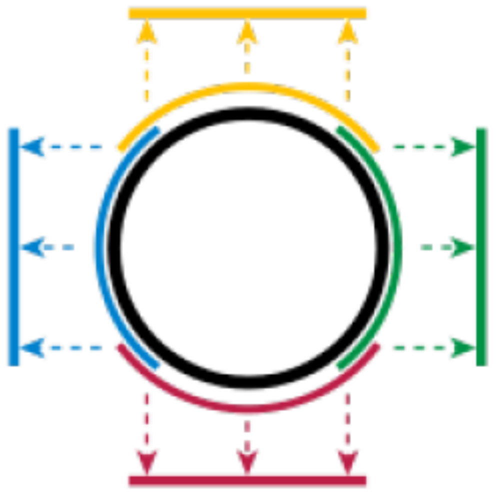
Possible configurations lies on
$q=(x_1, x_2) \in R^2 \mid x_1^2 + x_2^2 = L$



In this case the topological space S^1

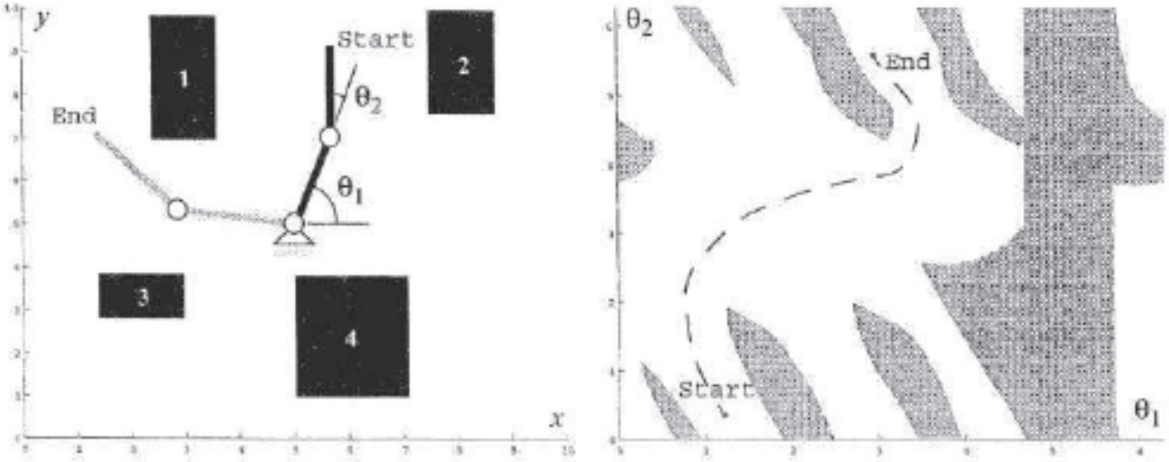
The Configuration Space



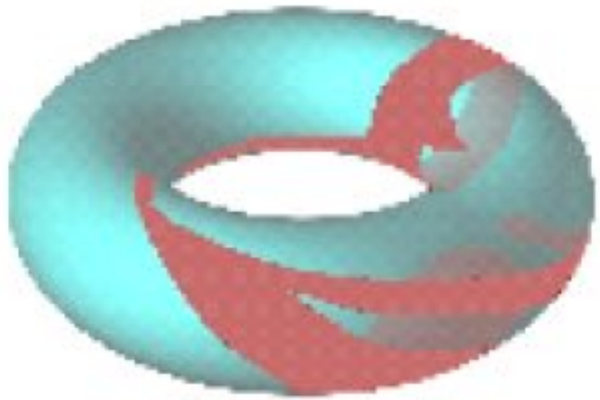


$$U_1 = \{x \in \mathbb{S}^1 | x_2 > 0\}, \phi_1(x) = x_1$$
$$U_2 = \{x \in \mathbb{S}^1 | x_2 < 0\}, \phi_2(x) = x_1$$
$$U_3 = \{x \in \mathbb{S}^1 | x_1 > 0\}, \phi_3(x) = x_2$$
$$U_4 = \{x \in \mathbb{S}^1 | x_1 < 0\}, \phi_4(x) = x_2$$

The Configuration Space



Topological Space
CS = $S^1 \times S^1 = T^2$



The Configuration Space – Typical Examples

- $S^1 \times S^1 \times \ldots \times S^1$ (n times) = $T^n \neq S^n$
- $S^1 \times S^1 \times S^1 \neq SO(3)$
- $SE(2) \neq R^3$
- $SE(3) \neq R^6$

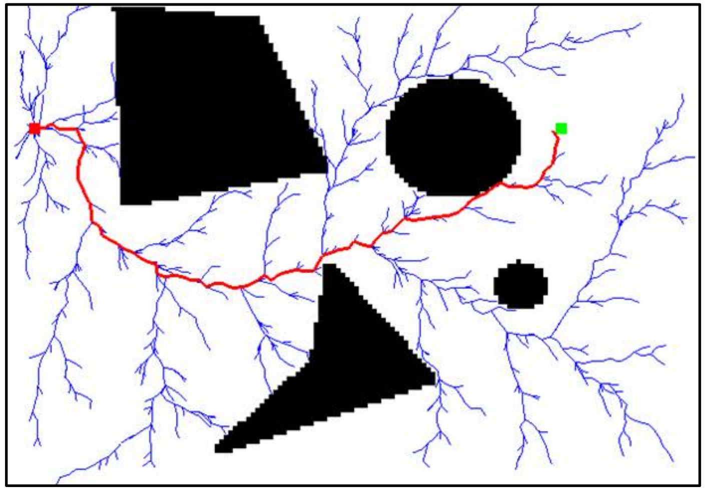
R

- R^1 and $SO(2)$ are 1-dimension manifolds
- R^2 , S^2 and T^2 are 2-dimension manifolds
- R^3 and $SE(2)$ $SO(3)$ and are 3-dimension manifolds
- R^6 , $SE(3)$ and T^6 are 6-dimension manifolds

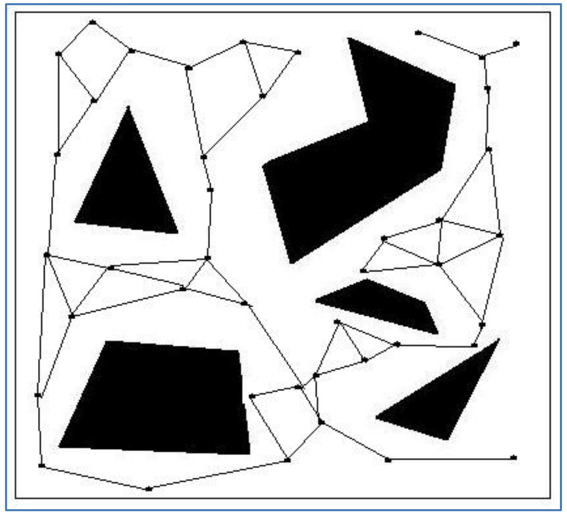
The Configuration Space – Typical Examples

Robot Type	**CS Representation**
Robot movil with planar translation	R^2
Robot movil with planar translation and rotation	$SE(2)$ or $R^2 \times S^1$
Rigid object with traslations in 3D	R^3
Free Rigid Object in 3D	$SE(3)$ or $R^3 \times SO(3)$
Robot arm with n articulations	T^n
Planar mobile robot with an Robot arm with n articulations mounted on it	$SE(2) \times T^n$

Random Sampling-based Methods



- There exist two main approaches
  - RRT (Rapidly-Exploring Random Trees)
  - PRM (Probabilistic Roadmap Method)

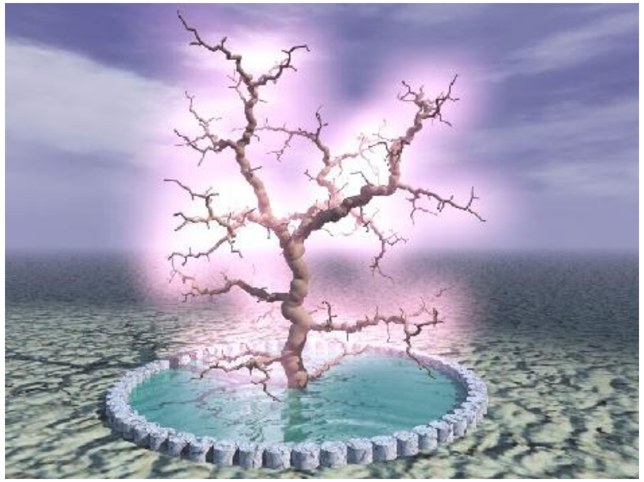


What do I need to apply this algorithms?

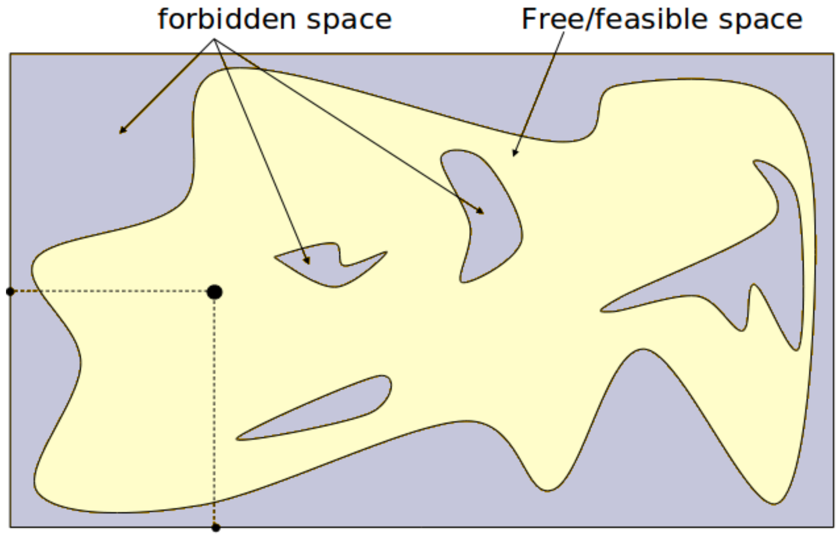
- Local motion planning policy (gradient-based, interpolation, optimal control)
- Collision detector (PQP, SOLID, V-Clip, Rapid, V-Collide). Example
- Cost function (Metric in topological spaces: Euclidean norm, Manhattan norm, Lp-norm)
- Sampling Method (Uniform, normal/biased sampling, manipulability)
- Geometric information about robot and environment (Mesh of links and obstacles)
- Smoothing function

What is the Idea?

- *The method:* Instead of Exploring exhaustively all possibilities, why not exploring randomly a sub-set of these possibilities but maintaining progress in exploration.
- *The Cost:* Relax Completeness and optimality of the solution
- ***The aim:* Trade-off among quality of the solution and computational time**



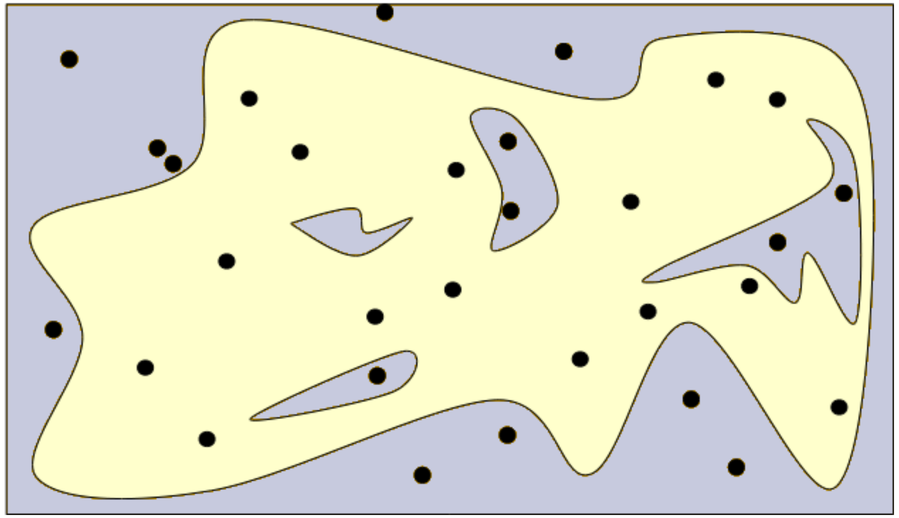
PRM – The method



- The objects are already augmented using Minkowsky sum, thus the robot can be considered as a point.
- Cost Function: Euclidean Norm
- Local planning: Interpolation

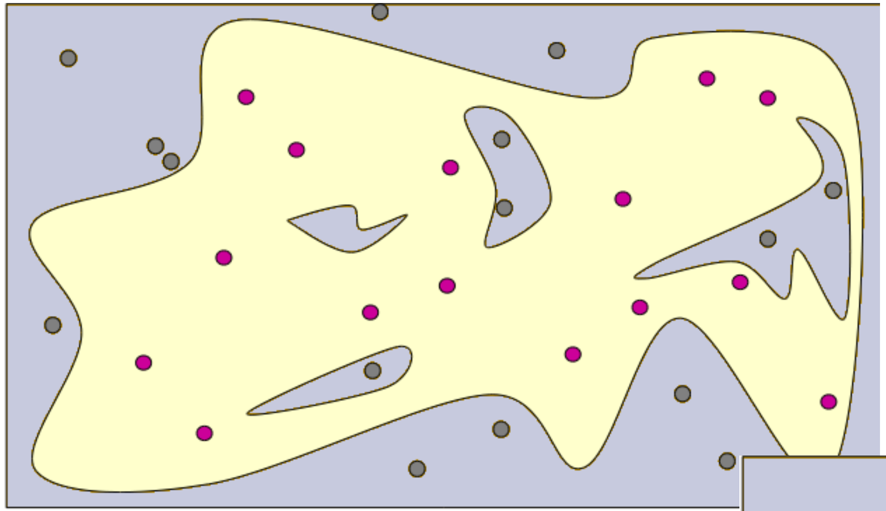
Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; Overmars, M. H. (1996), "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Transactions on Robotics and Automation* **12** (4): 566–580

PRM – The method

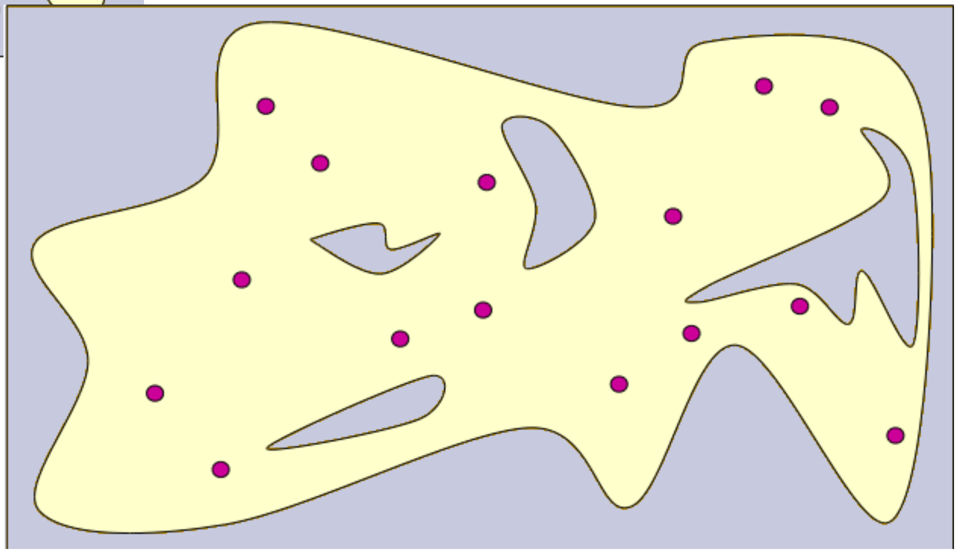


- Explore CS_{free} using a finite number of randomly sampled configuration

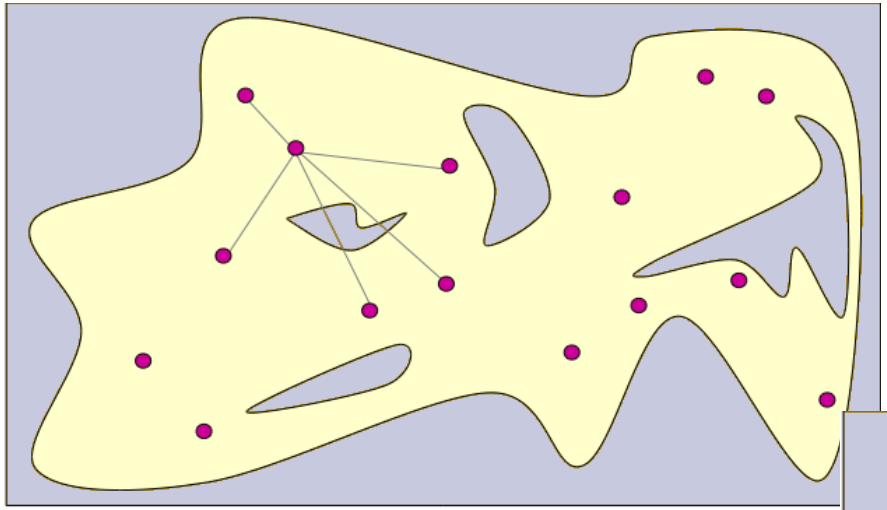
PRM – The method



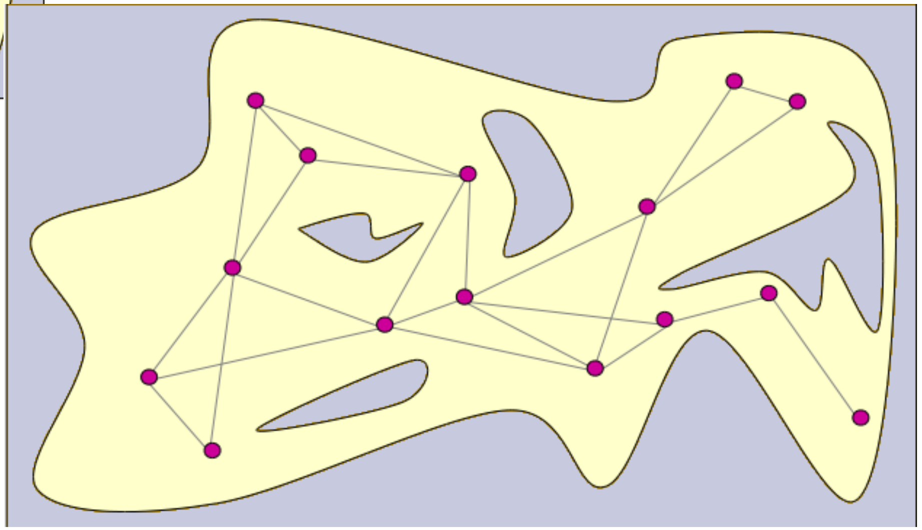
- Reject all configurations in CS_{free}



PRM – The method

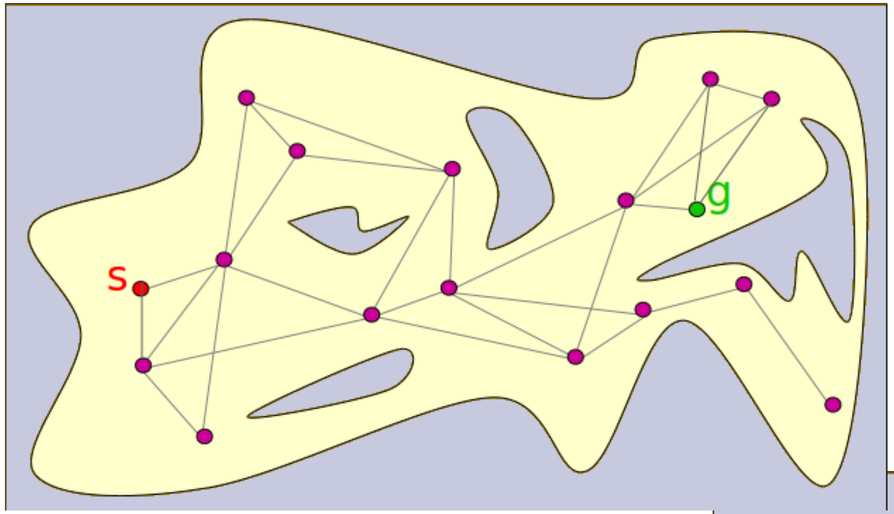


- Find all feasible connections in all remaining points in CS_{free}
- Also called learning phase

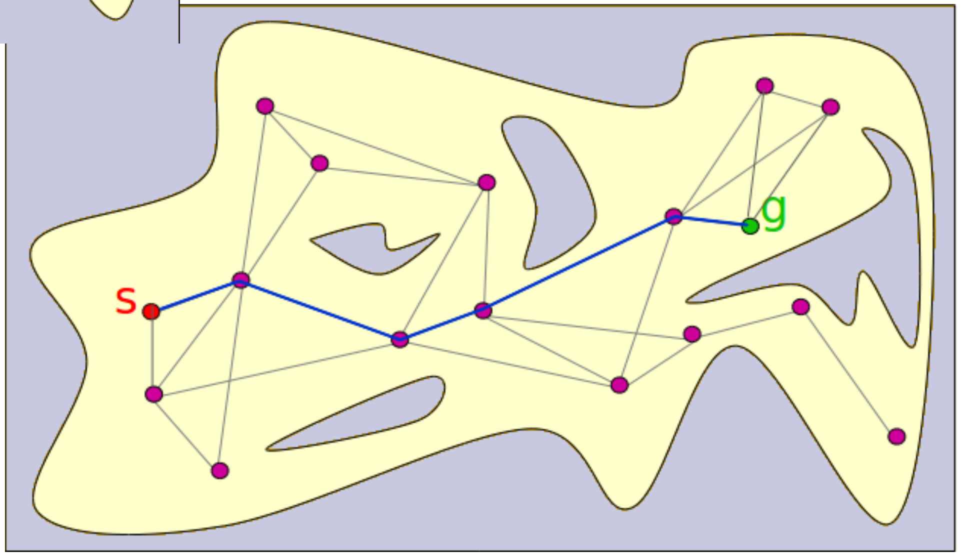


- Roadmap is done!!!
- Let’s make a query

PRM – The method

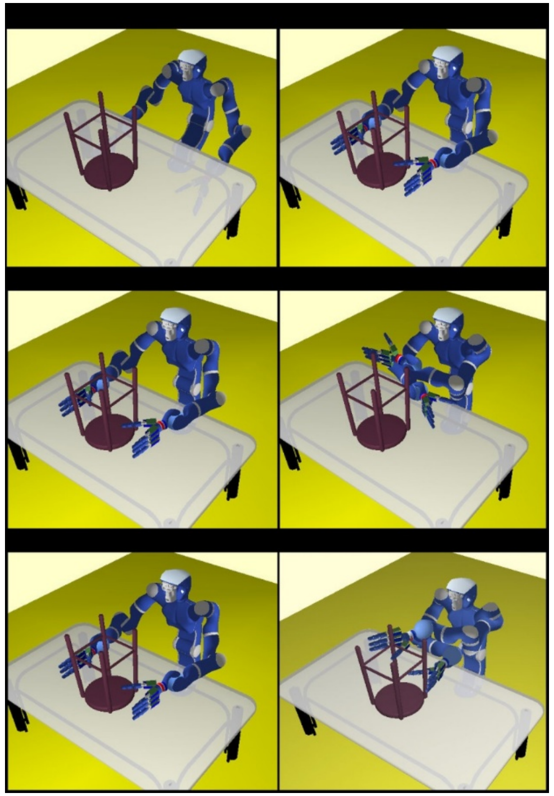


- Connect Initial and final configuration to the roadmap

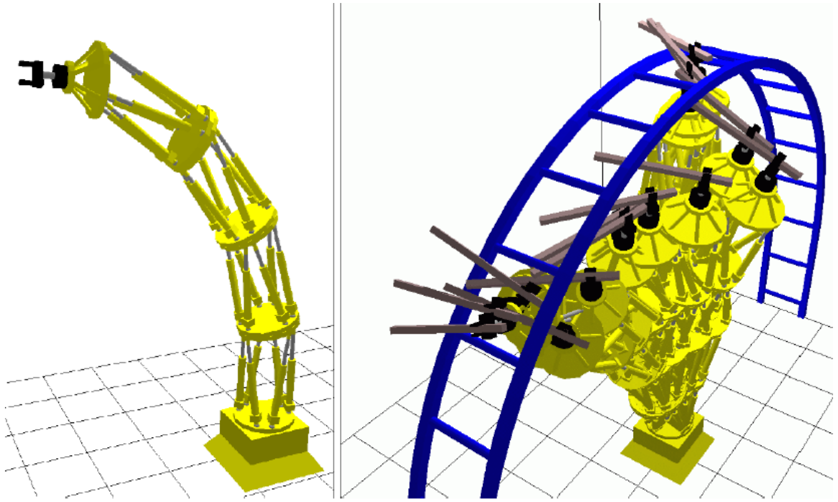


- Find a path!!!
  - Dijktra's, A*…

PRM – Application examples



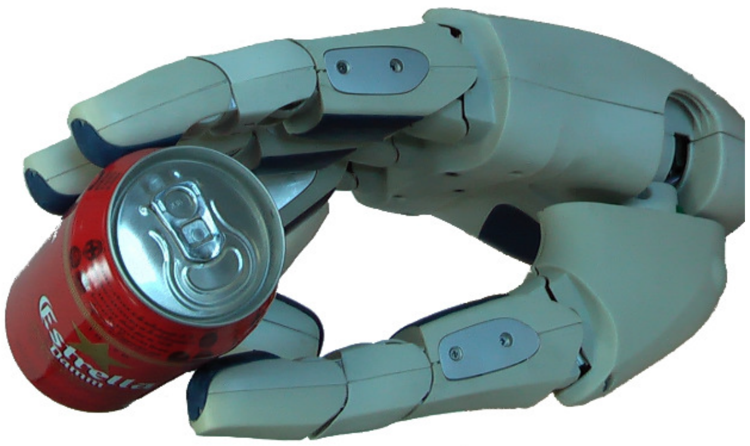
- Topology: $\mathcal{CS}=\mathrm{T}\uparrow 16$



Multiple PRM is used to connect the complete space going through singular configurations

PRM – Application examples

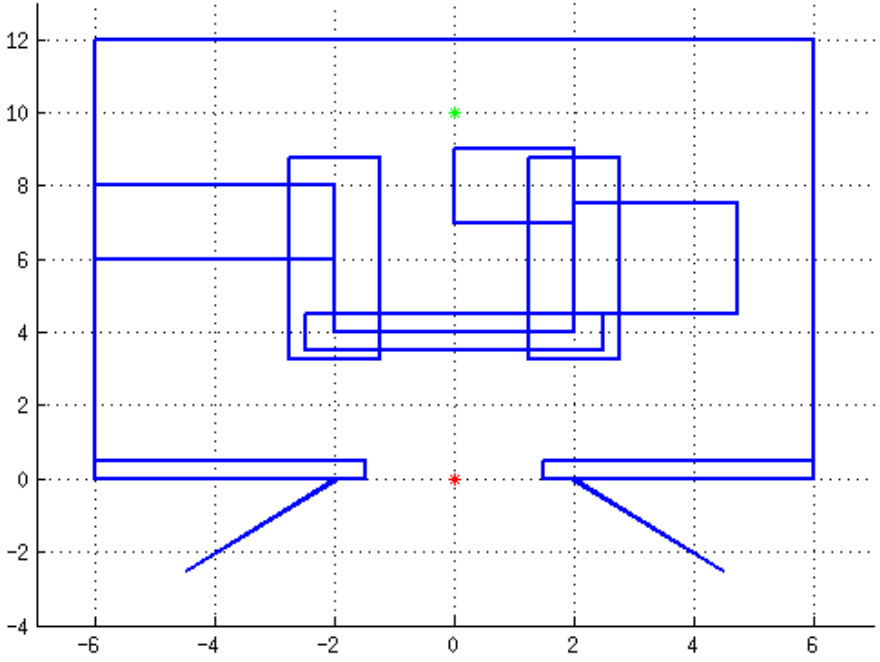
Finding better grasp configurations – Using continuation Methods



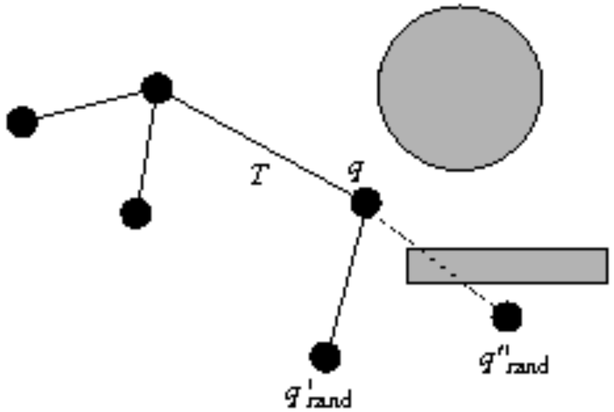
PRM is performed in the object

PRM VIDEOS

RRT – The method

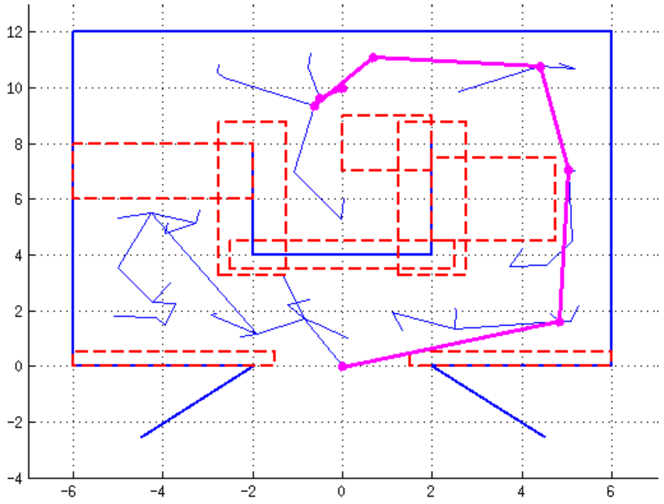


- Topology of the space : $CS = R^2$

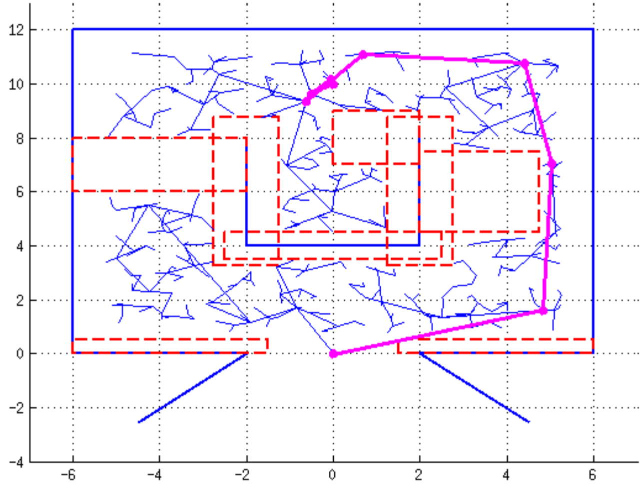


Lavalle, S.M. (1998). "Rapidly-exploring random trees: A new tool for path planning". *Computer Science Dept, Iowa State University, Tech. Rep. TR*: 98–11.

RRT – The method



- RRT algorithm using 200 samples



- RRT algorithm using 600 samples

- **Is the same path !!!**

RRT and PRM Differences

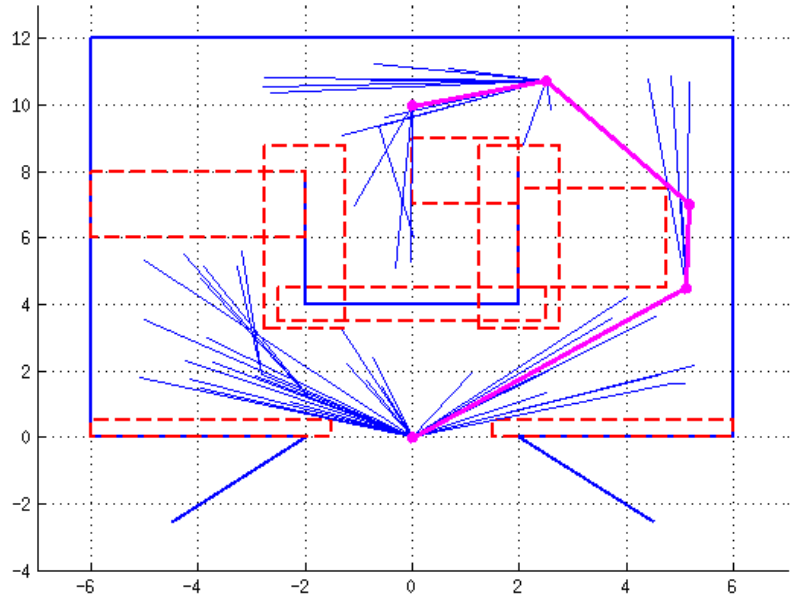
- RRT
  - Single query : you have to build a tree for each query
  - Fast initial solution
  - Better for dynamic environments
- PRM
  - Multiple query : If the environment does not change the roadmap is reusable.
  - Slow but the quality of the path is better
  - Better for static environments

– Applicable in real time

– This methods deliver a global solution, there are local minima (using uniform distribution)

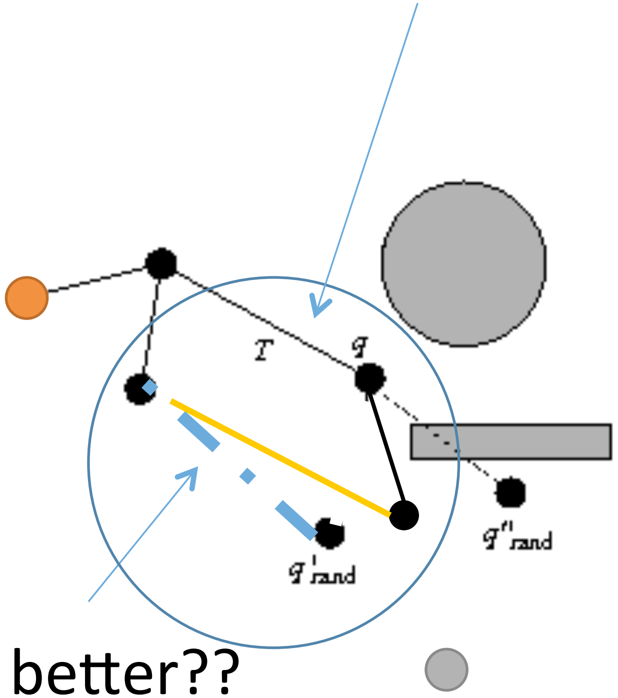
– What if we combine them? **It depends on your application!**

RRT*



RRT* 200 samples

Can we Rewire??

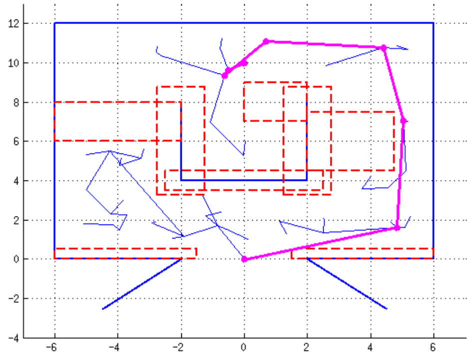


Is it better??

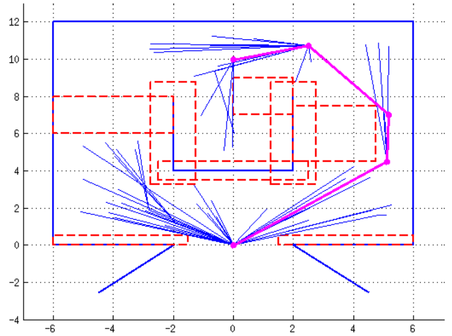
Basically you have to apply the A* algorithm in each iteration

Sertac Karaman and Emilio Frazzoli. *Incremental Sampling-based Algorithms for Optimal Motion Planning*, 2010.

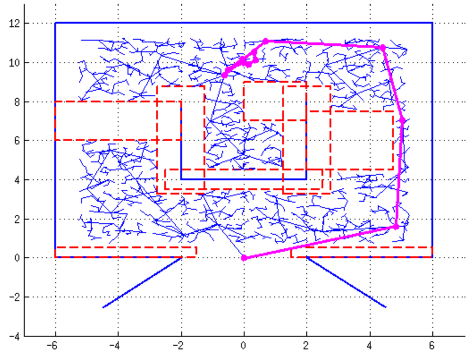
RRT vs. RRT*



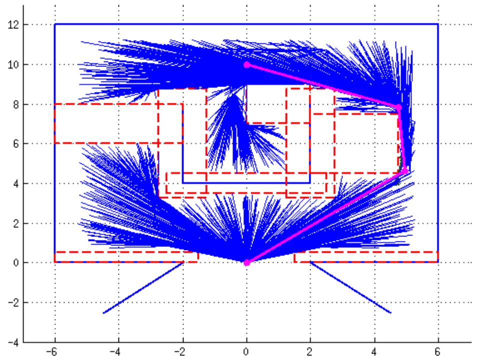
RRT 200 samples



RRT* 200 samples



RRT 2000 samples



RRT* 2000 samples

RRT vs. RRT* in T^{7}

RRT

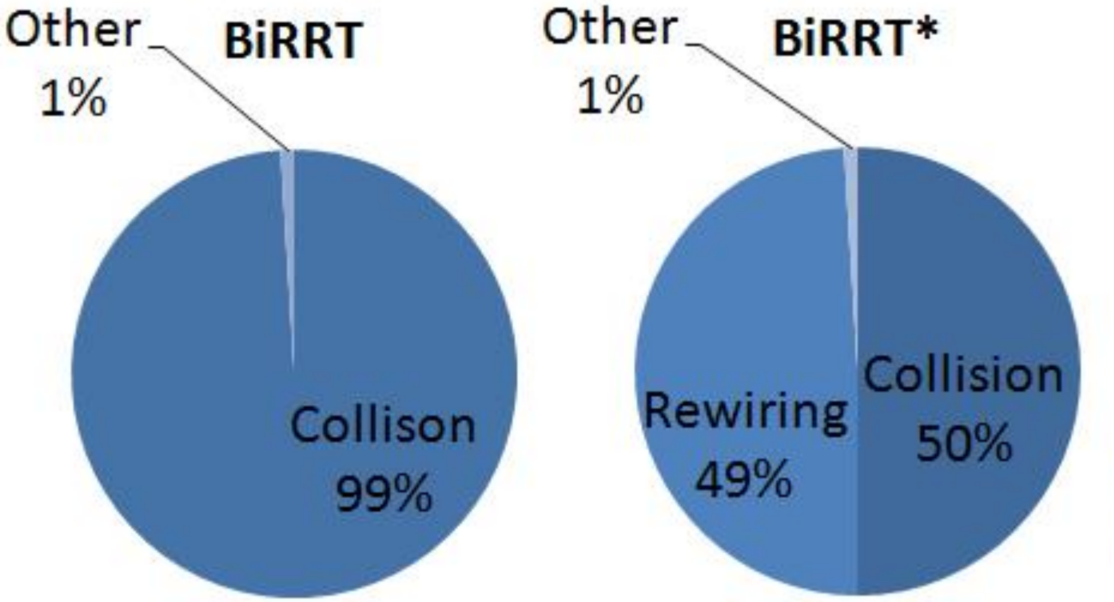
RRT*

Time RRT = ~.97sec
RRT*=~1.8sec

	RRT	RRT*
Cost Function	4.28	2.23
Samples	5000	5000

Collision Detector POP all this in C++ o_O

5. RRT vs. RRT*


Other
1%
BiRRT
Collison
99%
Other
1%
BiRRT*
Collision
50%
Rewiring
49%

Summarizing

- **+ Positive Issues**
  - Probabilistic Completeness but not deterministic
  - It is not necessary to build the Configuration Spaces.
  - Easy application in High-Dimensional Spaces
  - Fast queries

- **- Negative Issues**
  - Behavior is not good when narrow passages.
  - Connection among nodes is difficult when there exist additional constraints
  - It is difficult to guarantee completeness and optimality.

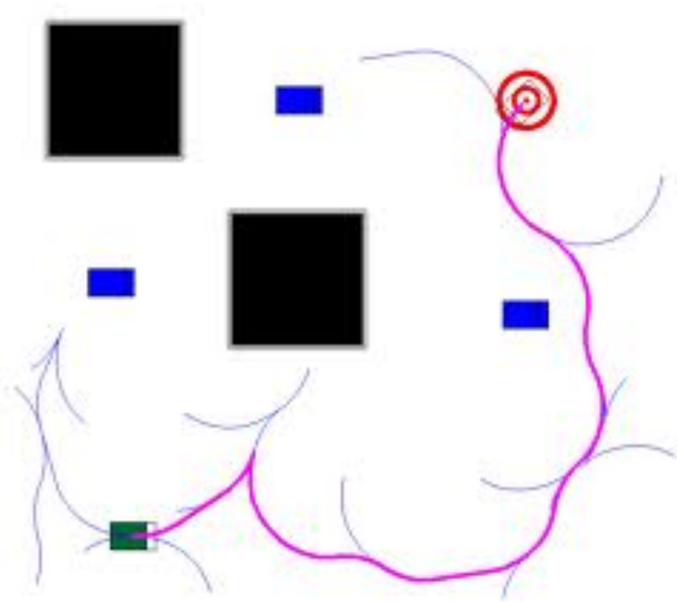
Remember to use an smoother to apply paths in real robots

Videos

State of the Art

- Constrained Motion Planning (Projection, rejection, direct sampling)
  - Kavraki 20001, Cortes 2005, Stilman 2010, Stilman 2013, Berenson 2013
- Acceleration level constraints (Kinodynamic Planning)
  - Lavalle 2001, Masoud 2010
- Introduce Compliant (Can we relax constraints through compliance?)
  - Relaxing Constraints
  - Reactive Planning

Considering non-holonomic constraints



The difference resides in the local planning policy

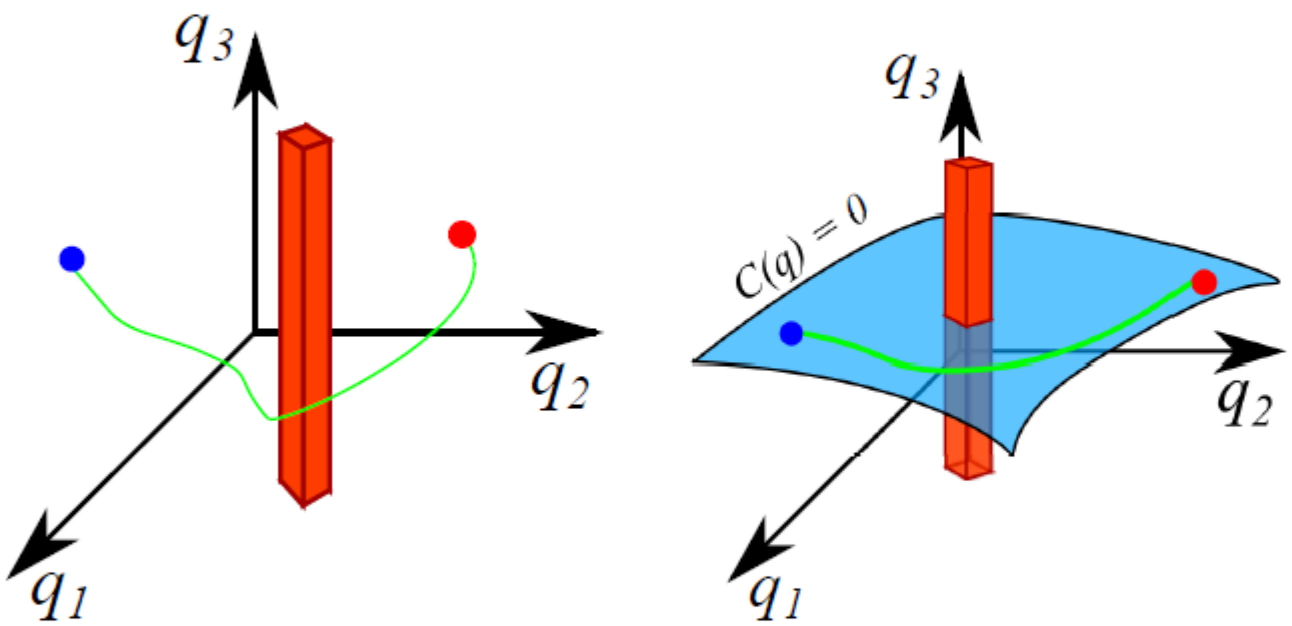
The Motion Planning Problem with constraints

- Consider a Configuration Space $(CS) \in R^d$ as a compact set of q_i elements called configurations. Defining the obstacle region as $CS_{obs} \in CS$, $CS_{free} := CS \setminus CS_{obs}$ is the free region. Then a constrained subspace is defined by $CS_{con} := \{q_i \mid F(q_i) = 0\}$. Thus, we need to find a continuous path

$$\sigma : [0,1] \rightarrow CS_{valid} \mid \{\sigma(0) = q_{ini}, \sigma(1) = q_{final}\}$$

Where $CS_{valid} = CS_{free} \cap CS_{con}$

The Motion Planning Problem with constraints



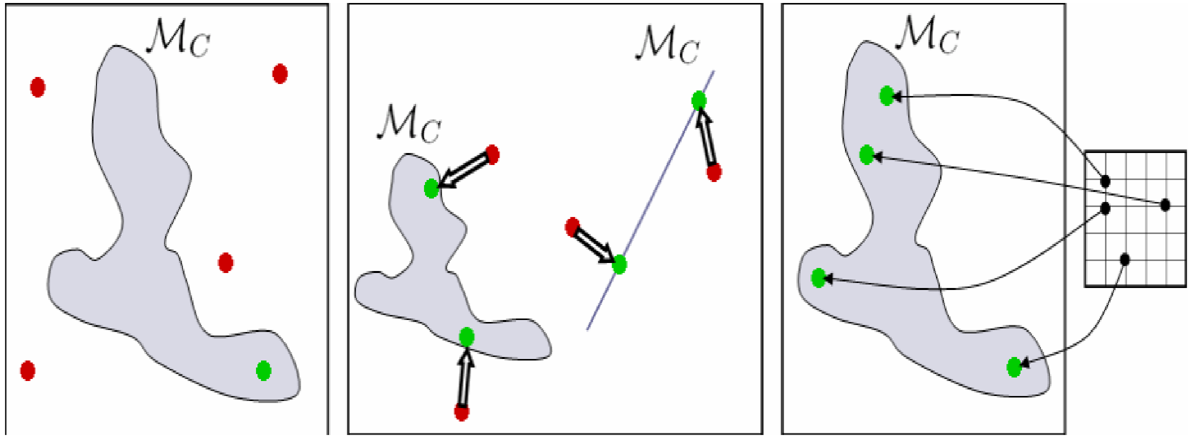
What is the main problem

- First CS_{con} is defined by in an implicit function $F(q_i)$ describing, thus impossible use direct sampling
- CS_{con} has lower dimension than CS
- The probability of sampling a point in CS_{con} is defined by

  $\rho = vol(CS_{con}) / vol(CS)$, this is zero!!!

State of The art

- Random sampling approaches for constrained systems can deal with
  - Nonholonomic Constraints (Lavalle 2000)
  - Closed kinematic Chains (Cortes 2005)
  - Task Constraints (Stillman 2007)
  - Dynamic Constraints (Masoud 2010)
- The main methods are:



Rejection

Projection

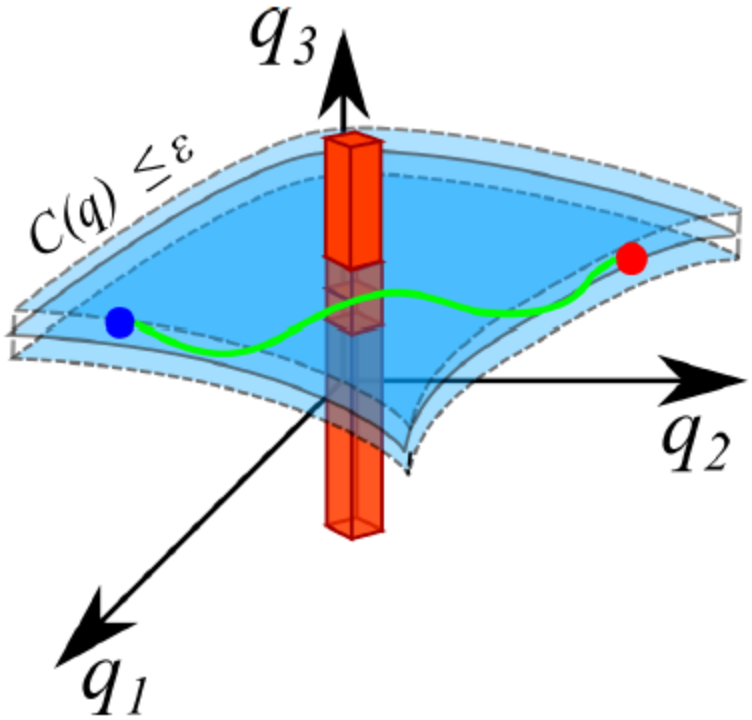
Direct Sampling (It requires a splicit parameterization of CS_{valid}

State fo the Art

- **What are the problems with this approaches?**
  - Most (practically all) samples are rejected in method one
  - Projections take long computational time
  - PRM and RRT are far from optimality (unnecessary complex motions typically appear)
  - There is no provision for planning or controlling interaction forces
- **Good News !!!**
  - Most of the times, motions need not to exactly match the plan
  - Indeed, execution will not coincide with plans, and environments are not as modeled, so some built-in robustness is mandatory
  - Systems are not rigid – indeed, modern robots are rather soft, or even have variable stiffness

So what can we do?

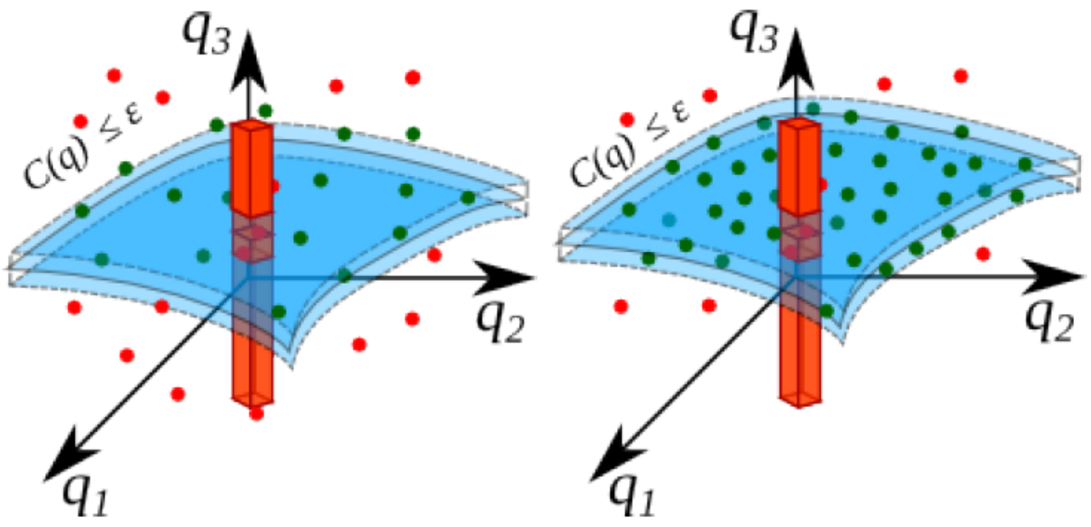
- Relax!!!
  - Instead of planning on $F(q_i) = 0$ we can plan in $F(q_i) \leq |\epsilon|$



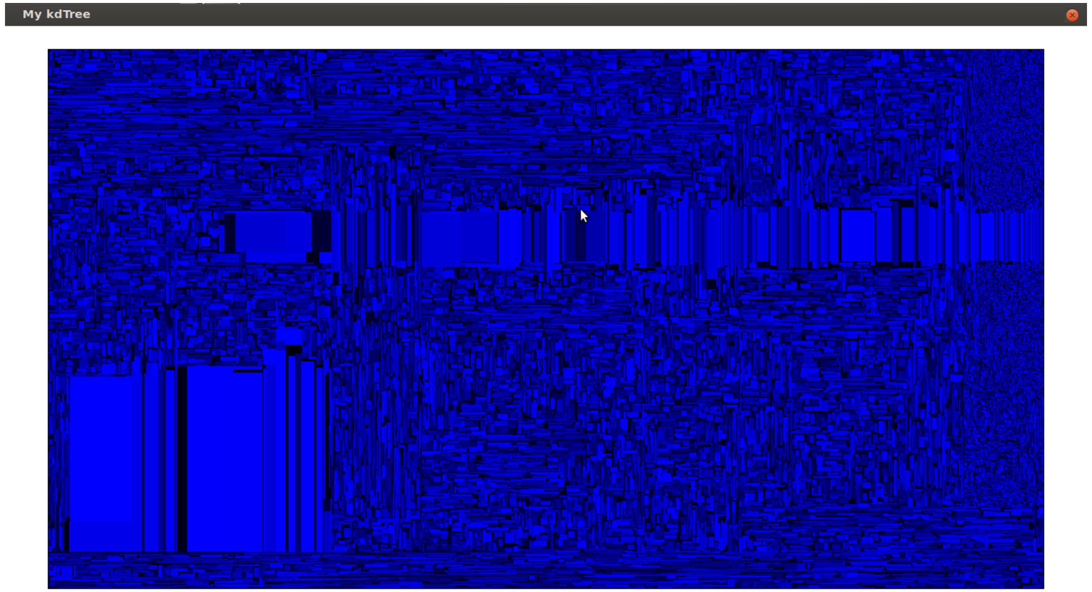
- Now there is a narrow but fully dimensional boundary layer. Thus, there is probability of picking a point on the $C_{S_{valid}}$

So what can we do?

- Recently Recenlty Frazzoli in 2013 propose a method to bias new samples to free space using an weighted kd-tree.
- We can use it to bias new samples to the manifold.



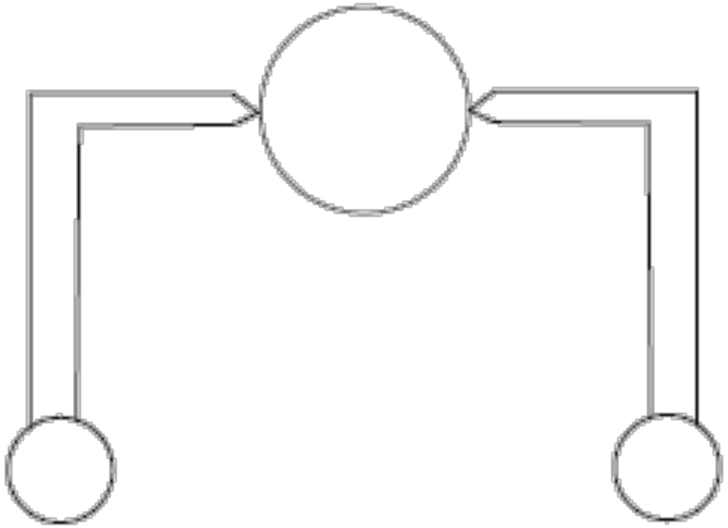
Adaptive *kd*-tree

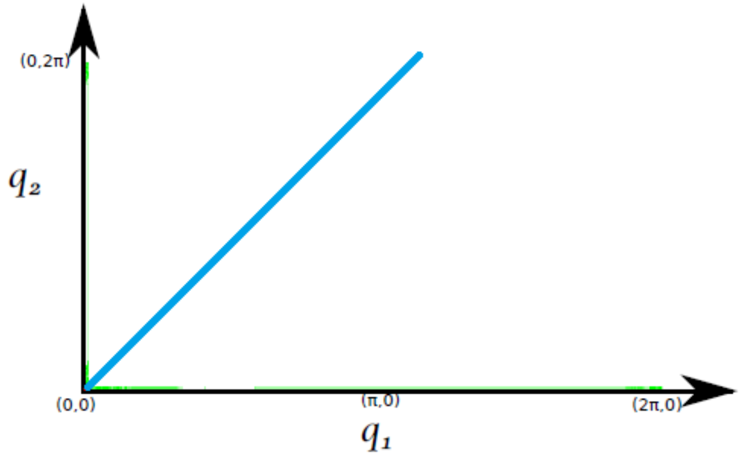


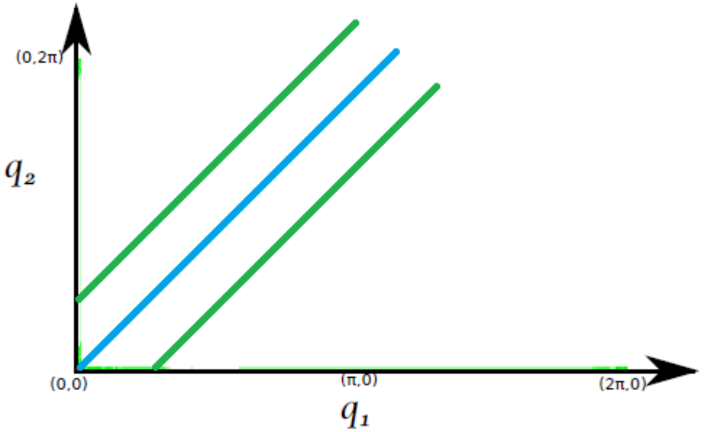
Algorithm 1: GenerateSample (H, v)

if $v.c[0] = v.c[1] = \emptyset$ **then**
  $x \leftarrow$ SampleUniform (H) ;
  $v.T \leftarrow v.T + 1$;
  $r =$ Collision-free (x) ;
  if r **then**
    $v.x \leftarrow x$;
    $v.F \leftarrow v.F + 1$;
    $(v.c[0], v.c[1]) \leftarrow$ Split (v, x) ;
    for $i = \{0, 1\}$ **do**
      $v.c[i].P \leftarrow v$;
      $v.c[i].j \leftarrow (v.j + 1) \bmod d$;
      $w \leftarrow$ Measure $(v.c[i])$ /Measure (v) ;
      $v.c[i].T \leftarrow w \cdot v.T$;
      $v.c[i].F \leftarrow w \cdot v.F$;
      $v.c[i].M = \left(\frac{v.c[i].F}{v.c[i].T}\right)$ Measure $(v.c[i])$;
else
  $u \leftarrow$ SampleUniform $([0, v.M])$;
  if $u \leq v.c[0].M$ **then**
    $(x, r) \leftarrow$ GenerateSample $(v.c[0])$;
  else
    $(x, r) \leftarrow$ GenerateSample $(v.c[1])$;
  $v.M = v.c[0].M + v.c[1].M$;
return (x, r)

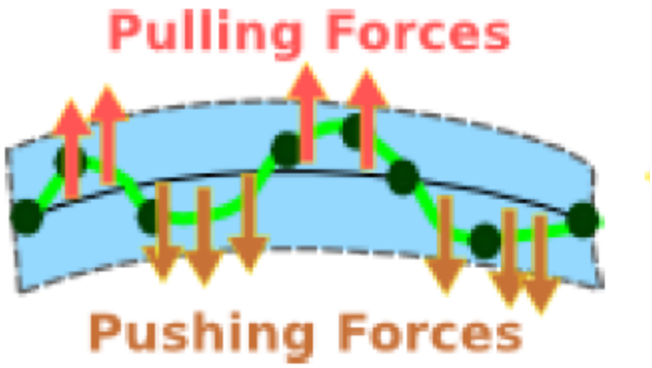
Example





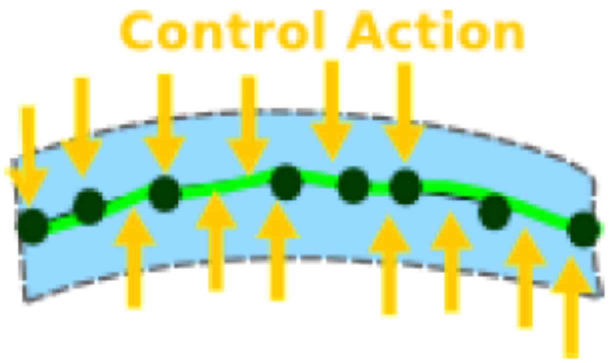


What does relaxation means in real robots?



So we need a Controller

- Controller task
  - Follow the planned path
  - Project back the configurations to the constraint
  - Maintain desired interaction forces



- We can do this with Force/Position Controllers
  - There are a lot of approaches to do that (linear, nonlinear, robust, adaptive, Force-position subspaces, etc.)

Linear Controller

- Linearized dynamic equations of the system

$$\dot{x} = Ax + B_\tau \tau' + B_\omega \omega$$

$$A = \left[\begin{array}{c|c} 0 & I \\ \hline -L_k & -L_b \end{array}\right], \; B_\tau = \left[\begin{array}{c} 0 \\ 0 \\ \hline M_h^{-1} \\ 0 \end{array}\right]; B_\omega = \left[\begin{array}{c} 0 \\ 0 \\ \hline 0 \\ M_o^{-1} \end{array}\right]$$

$$x = \left[q_{eq}^T \; u_{eq}^T \; 0^T \; 0^T\right]^T$$
$$\tau' = \tau - J^T t_{eq}$$
$$\omega = G t_{eq}.$$

$$L_k = M^{-1} P_k; \; L_b = M^{-1} P_b,$$

$$M = \begin{bmatrix} M_h & 0 \\ 0 & M_o \end{bmatrix}$$
$$P_k = \begin{bmatrix} J^T \\ -G \end{bmatrix} K \begin{bmatrix} J & -G^T \end{bmatrix}$$
$$P_b = \begin{bmatrix} J^T \\ -G \end{bmatrix} B_q \begin{bmatrix} J & -G^T \end{bmatrix}$$

Linear Controller

- Linearized dynamic equations of the system

$$\dot{x} = Ax + B_\tau \tau' + B_\omega \omega$$

$$x = \left[q_{eq}^T \; u_{eq}^T \; 0^T \; 0^T\right]^T$$
$$\tau' = \tau - J^T t_{eq}$$
$$\omega = G t_{eq}.$$

$$C_u \doteq [0 \; I \; 0 \; 0]$$

$$C_t = \left[KJ \;\; -KG^T \;\; B_q J \;\; -BG^T\right]$$

Rigid Body motions

$$C = \begin{bmatrix} \Gamma_u^+ C_u \\ E^+ C_t \end{bmatrix}$$

Internal forces

Linear Controller

- Linearized dynamic equations of the system

$$\dot{x} = Ax + B_\tau \tau' + B_\omega \omega$$

$$x = \left[q_{eq}^T \; u_{eq}^T \; 0^T \; 0^T\right]^T$$
$$\tau' = \tau - J^T t_{eq}$$
$$\omega = G t_{eq}.$$

Internal forces

$$\begin{aligned} \dot{y} &= C_t A x + C_t B \tau_t^* \\ &= C_t A x + E^+ B_q J M_h^{-1} \tau_t^* \end{aligned}$$

Rigid Body motions

$$\begin{aligned} \dot{y} &= C_u A x + C_u B_\tau \tau_u^* \\ &= C_u A x + 0 \tau_u^* \\ \ddot{y} &= C_u A^2 x + C_u A B_\tau \tau_u^* \\ &= C_u A^2 x + 0 \tau_u^* \\ \dddot{y} &= C_u A^3 x + C_u A^2 B_\tau \tau_u^* \\ &= C_u A^2 x + \Gamma_u^+ M_o^{-1} G B_q J M_h^{-1} \tau_u^* \end{aligned}$$

Linear Controller

- Linearized dynamic equations of the system

$$\dot{x} = Ax + B_\tau \tau' + B_\omega \omega$$

$$x = \left[q_{eq}^T \ u_{eq}^T \ 0^T \ 0^T\right]^T$$
$$\tau' = \tau - J^T t_{eq}$$
$$\omega = G t_{eq}.$$

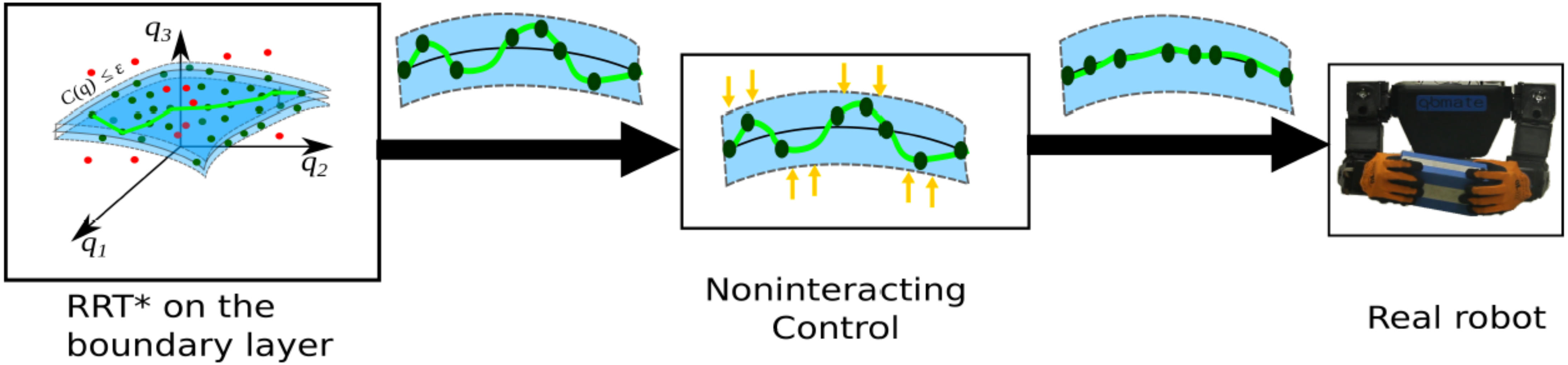
$$\hat{y} = Px + QB_\tau \tau^*$$

$$P = \begin{bmatrix} C_u A^3 \\ C_t A \end{bmatrix} \text{ and } Q = \begin{bmatrix} C_u A^2 \\ C_t \end{bmatrix}$$

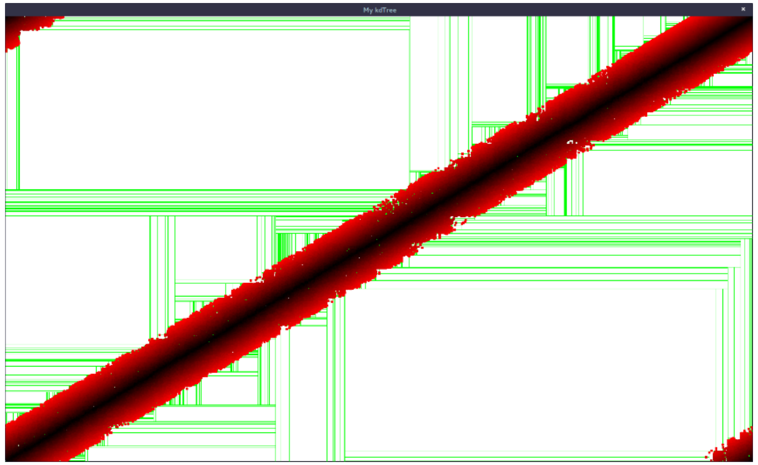
$$\tau = -Q^{-1}(Px + \tau^*)$$

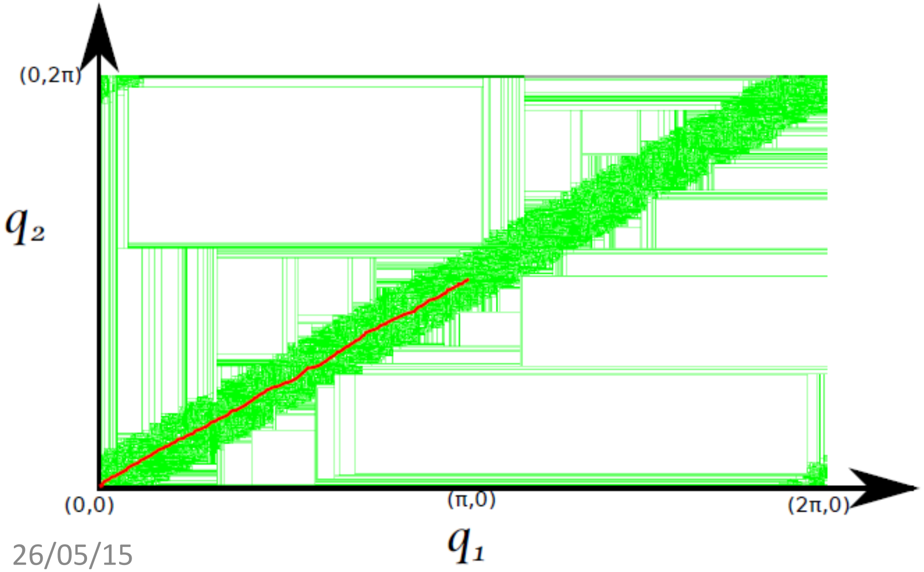
$$\hat{y} = \tau^* = [\tau_u \ \tau_t]^T = [\dddot{y}_u \ \dot{y}_t]^T$$

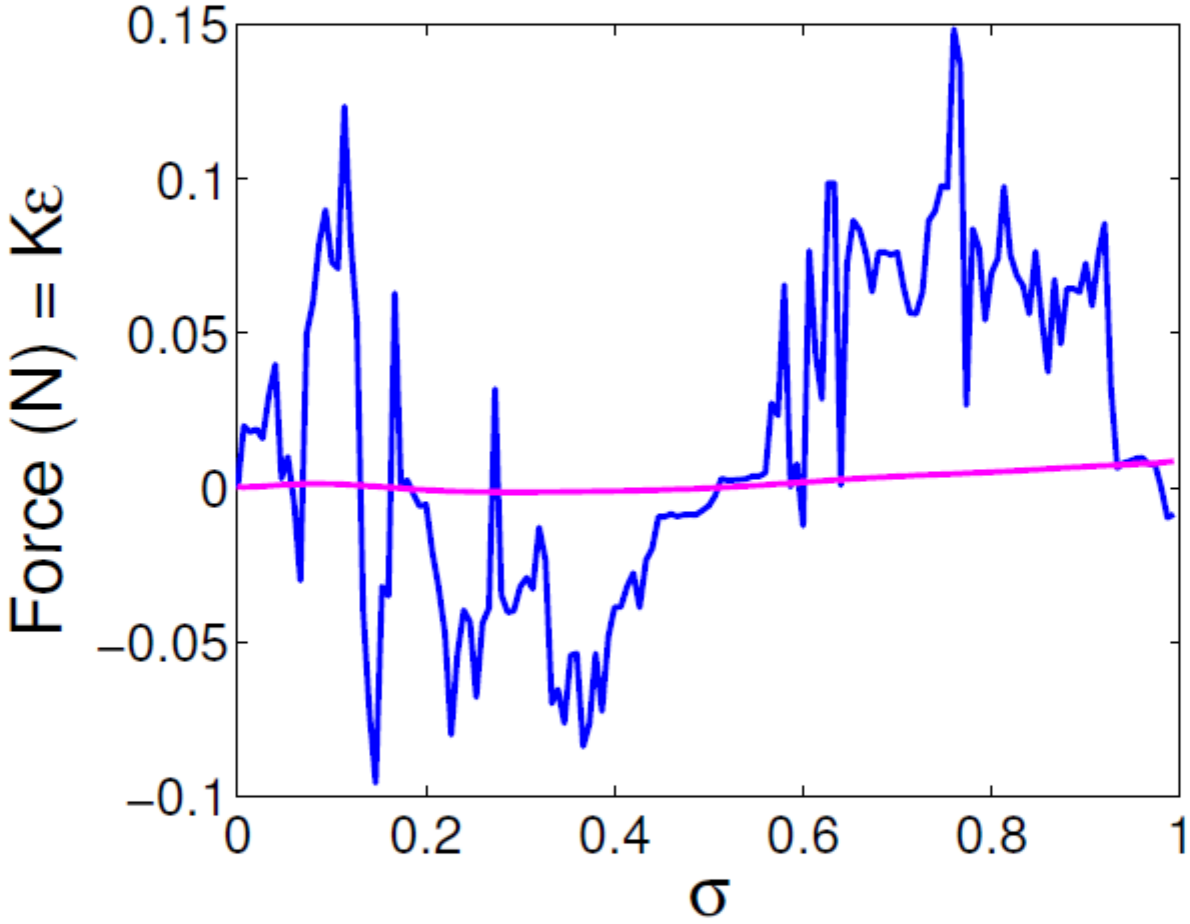
The method


q_3
$C(q) \leq \varepsilon$
q_2
q_1
RRT* on the
boundary layer
Noninteracting
Control
Real robot

Results







4 dof Example

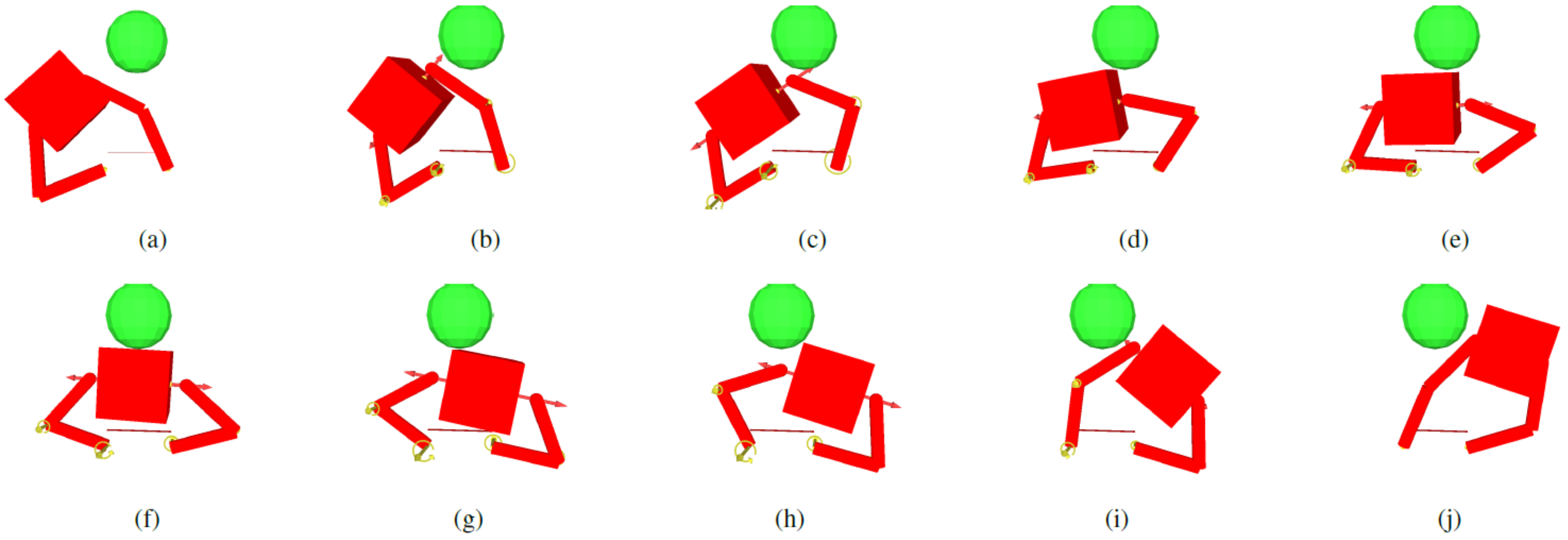


Fig. 7. Final path from the presented experiment. a) Initial position and j) final position.

Things to add to randomized planning

- Exploit Randomized planers (Not to expect just a yes/no answer)
  - Compliance (Bonilla)
  - Planning for contact points (Houser)
  - Minimal Constraint Removal (MCR) problem (Hauser)
  - Uncertaintity (Zito)
  - Bias for High Dimensional Spaces (Stilman)

Libraries for Motion Planning

- OMPL http://ompl.kavrakilab.org/
- Moveit (ROS Interface for OMPL) http://moveit.ros.org/
- Klampt http://motion.pratt.duke.edu/klampt/
- Openrave http://openrave.org/
- There is a repository on github with the kuka nimanual platform of centro piaggio. Ask for an account to use it. http://github.com/centroepiaggio
- Manue.bonilla@centropiaggio.unipi.it