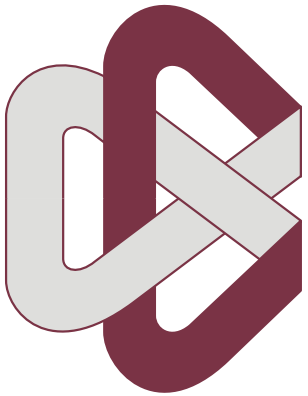
Visibility-based probabilistic roadmaps for motion planning



CIMAT

Miguel Vargas

Material taken form: T. Siméon, J.-P. Laumond, C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. Advanced Robotics, Volume 14, Number 6, 2000, pp. 477-493(17).

Introduction

Characteristics:

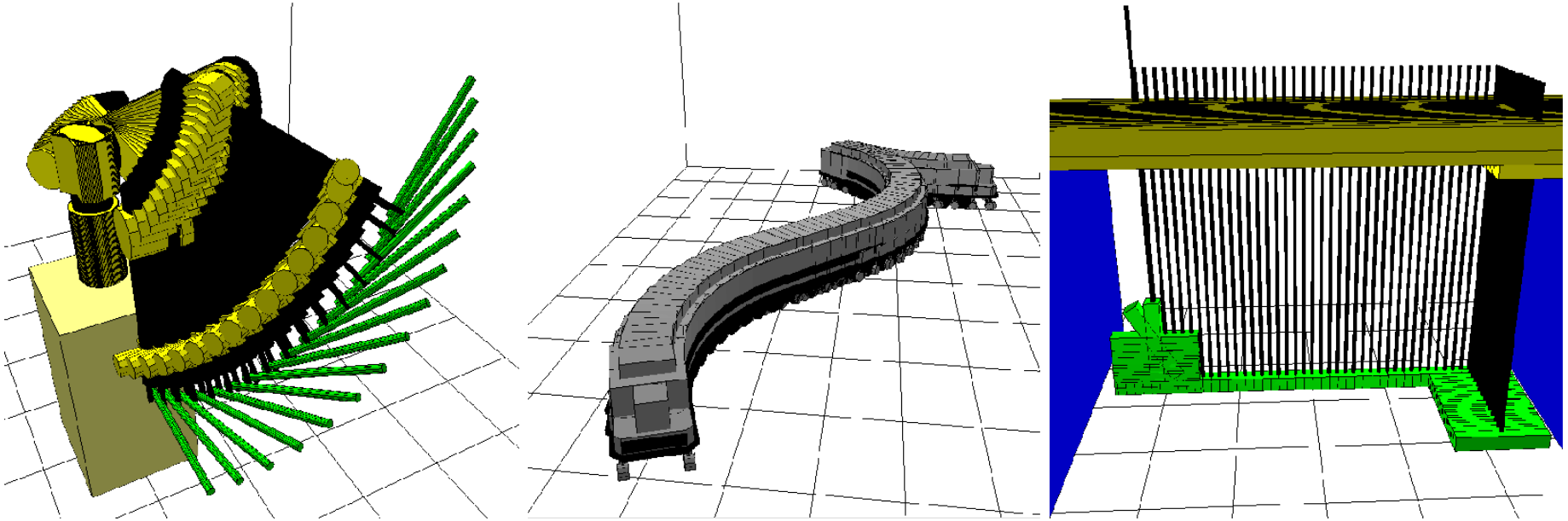
- It is a variant of the Probabilistic RoadMap (PRM) algorithm.
- Uses a visibility notion.
- The goal is to produce roadmaps that covers the configuration space using less nodes.

Some notation:

- CS configuration space.
- CS_{free} free space.
- q a configuration (also a node).
- $L(q, q')$ a path.
- $\mathbf{R}$ roadmap, represented as a graph.

Definitions

Local methods. Their goal is to find $L(q, q')$ taking in account kinematic constraints.



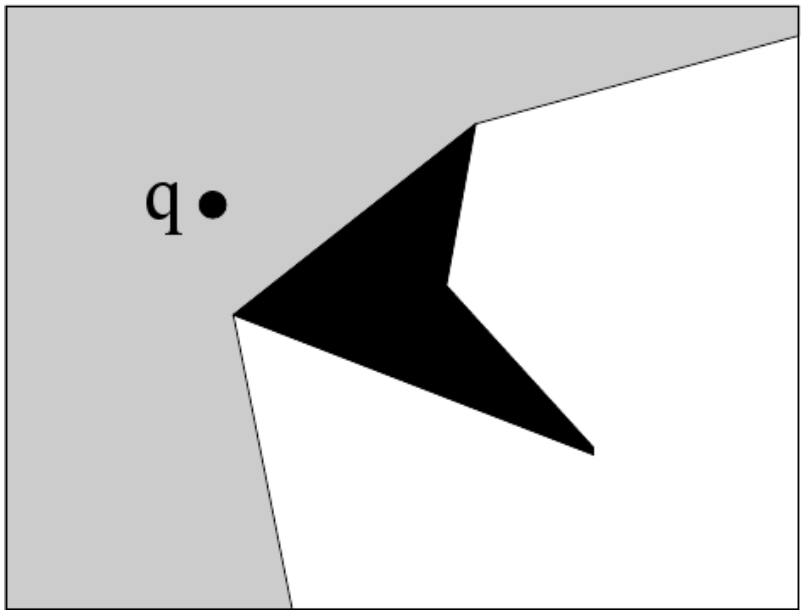
Roadmap. It is a graph whose nodes are collision-free configurations. Two nodes q and q' are adjacent if $L(q, q')$ (computed by the local method) lies in CS_{free}.

Query procedure. Given q_{init} and q_{goal}, if it is possible to connect both of them to the **R**, the procedure search for a path in this extended roadmap. The solution (if exists) is a sequence of connected subpaths.

Visibility domain. For a given local method L, the visibility domain of a configuration q is

$$Vis_L(q) = \{ q' \in CS_{free} \mid L(q, q') \subset CS_{free} \}.$$

Configuration q is said to be the guard of $Vis_L(q)$.

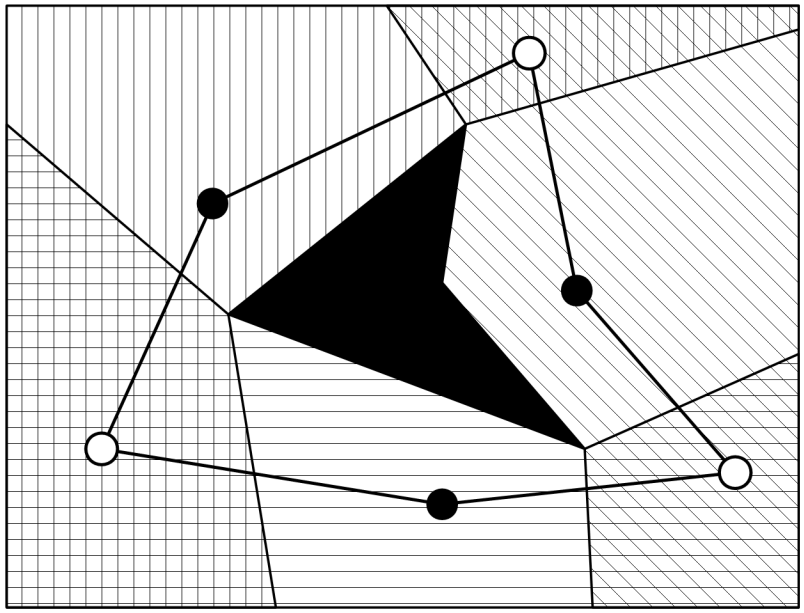


In this example $L(q, q')$ are straight-line segments.

Free-space coverage. A set of guards constitutes a coverage of CS_{free} if the union of their visibility domains covers the CS_{free}. Its existence depends on the shape of CS_{free} and on the local method.

Visibility roadmaps. These are constructed trying to reduce the number selecting non overlapping visibility domains and then choosing nodes to connect them.

Lets select s visibility domains such that the s guards do not "see" each mutually, i.e. for any two visibility domains $Vis_L(q_i)$ and $Vis_L(q_j)$ with guards $q_i, q_j \in s$, $L(q_i, q_j) \not\subset CS_{free}$.

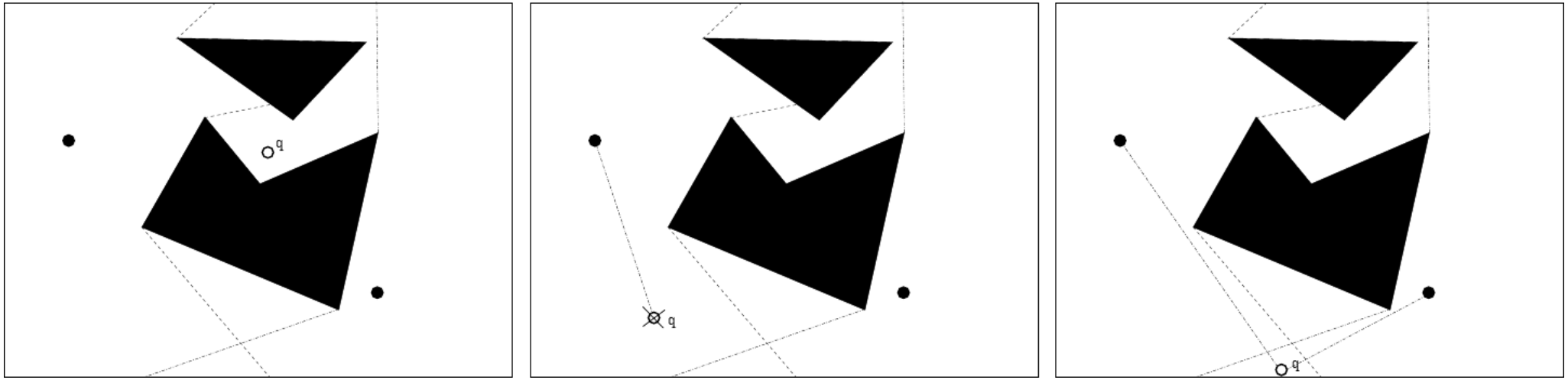


There are three guard nodes (black) and three connection nodes (white).

A graph **R** is building with nodes $\{q_i\}_{i=1}^{s}$. For any two intersecting visibility domains $Vis_L(q_i)$ and $Vis_L(q_j)$, we add a node q, called **connection node**, and two edges (q_i, q) and (q, q_j). The resulting graph **R** is called visibility roadmap. The number of guards is not optimal (in the sense of the art gallery problem, which is NP-hard).

Probabilistic algorithm

- The roadmap is constructed incrementally by randomly sampling the configuration space and attempting to connect some pairs of collition-free samples by the local method.
- The visibility roadmaps are build without any explicit computation of the visibility domains.



Three cases: node added as a guard; node rejected; connection node merging two connected components.

When a free sample is found, it is added to the roadmap in two cases:

- If it does not “see” another node already in **R**. This will be a new **guard**.
- If it is seen at leas by two nodes belonging to two distint connected components of **R**. This will be a **connection node**.

The algorithm

$Guard \leftarrow \emptyset$; $Connection \leftarrow \emptyset$; $ntry \leftarrow 0$
while $(ntry < M)$
  Select a random free configuration q
  $g_{vis} \leftarrow \emptyset$; $G_{vis} \leftarrow \emptyset$
  for all $G_i \in Guard$ do
    $found \leftarrow false$
    for all $g \in G_i$ do
      if $q \in Vis(g)$ then
        $found \leftarrow true$
        if $g_{vis} = \emptyset$ then $g_{vis} \leftarrow g$; $G_{vis} \leftarrow G_i$
        else $Connection \leftarrow Connection \cup \{q\}$; Create (g, q) and (q, g_{vis}); Merge G_{vis} and G_i
    until $found = true$
    if $g_{vis} = \emptyset$ then $Guard \leftarrow Guard \cup \{q\}$; $ntry \leftarrow 0$
    else $ntry \leftarrow ntry + 1$
end

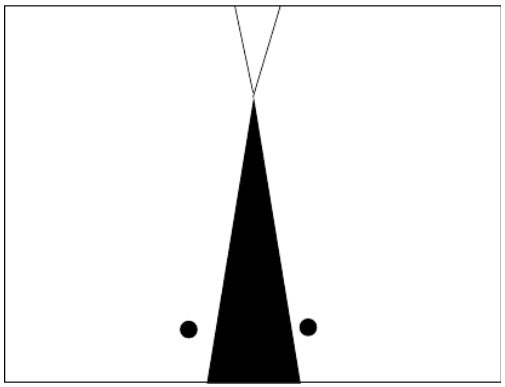
The parameter $ntry$ is the number of failures before the insertion of a new guard node. This parameter controls the stop of the algorithm. The algorithm stops when $ntry > M$.

Pros

- Each new guard increases the coverage of CS_{free}, therefore the probability of generating configurations in non covered regios keeps decreasing over the iterations.
- $1/ntry$ gives an estimation of the volumen not yet covered by visibility domains.
- The algorithm stops when $ntry > M$, which means that the volume of the free space covered by visibility domains becomes probably greated than $\left(1 - \frac{1}{M}\right)$.
- The size of the produced roadmaps, although not optimal, remains intrinsic to the complexity of CS_{free}.

Cons

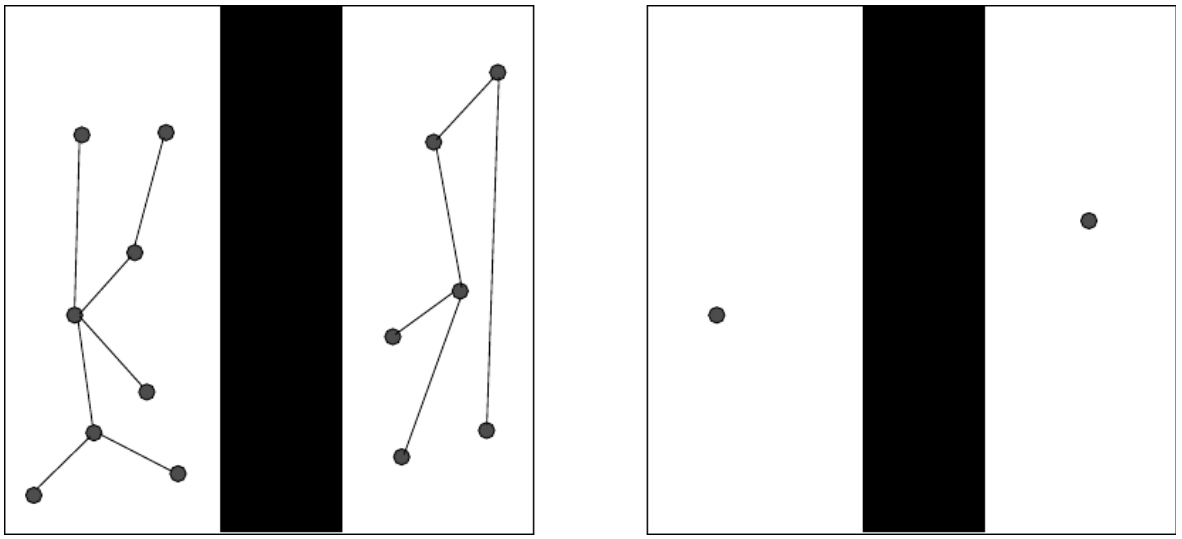
The random generation of the guards may produce in some cases guards that will be difficult to connect.



Comparison to basic PRM

The authors compared the visual probailistic roadmap method with a basic version of PRM.

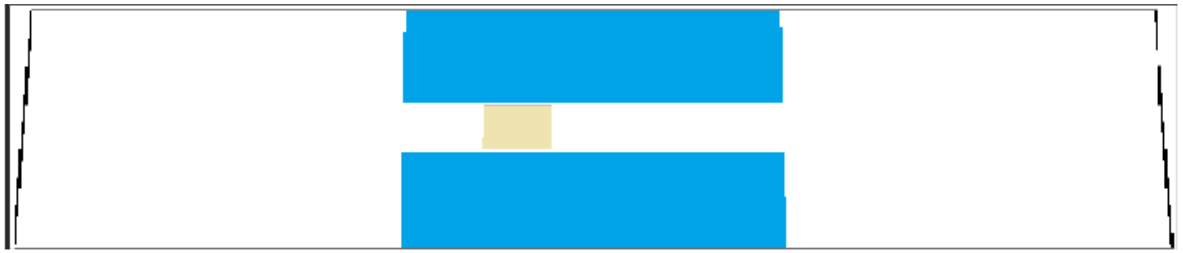
In PRM nodes are created first using uniform random distribution, then connected. This strategy requires dense roadmaps to capture the free space connectivity.



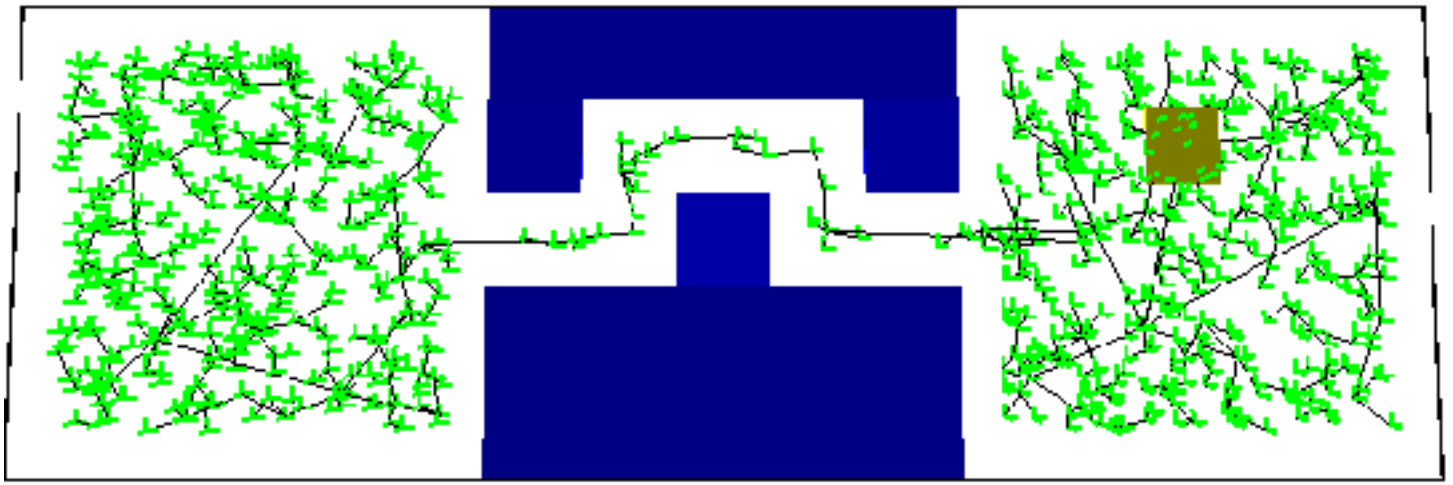
With a same number of n random collision-free configurations, the visibility roadmap algorithm will call the local method $O(n)$ times, while Basic-PRM will call it $O(n^2)$ times.

The sampling strategy used by VPRM is more expensive. However, thesting if a configuration is collision-free is *far less* expensive than checking the connections to the roadmap.

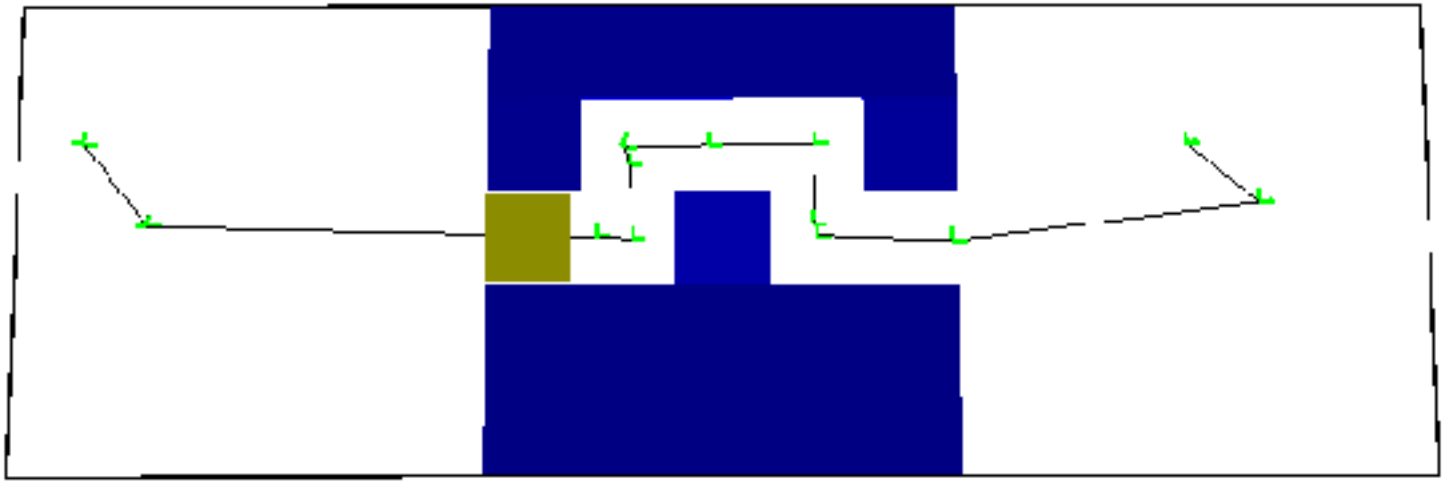
Narrow pasages



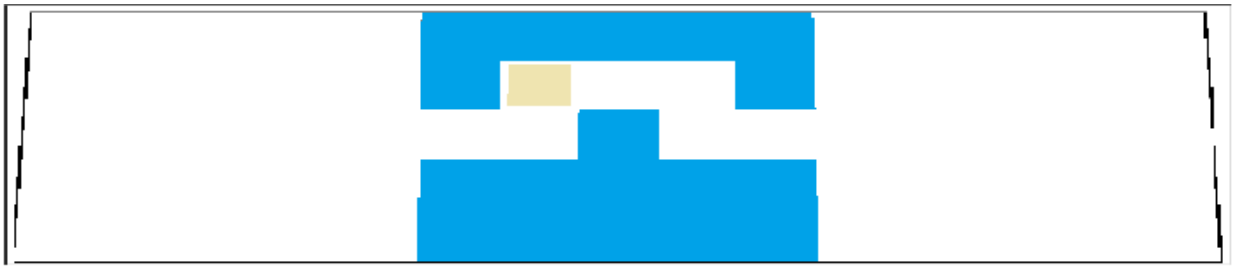
$\frac{1}{\epsilon}$	Basic-PRM $l_b(*1000)$	Basic-PRM $\#nodes$	Visib-PRM $l_v(*1000)$	Visib-PRM $\#nodes$	$gain$ $\frac{l_b}{l_v}$
100	33	363	14	5	2.3
1.000	2.508	3252	132	5	19
10.000	269.667	35987	1577	5	171



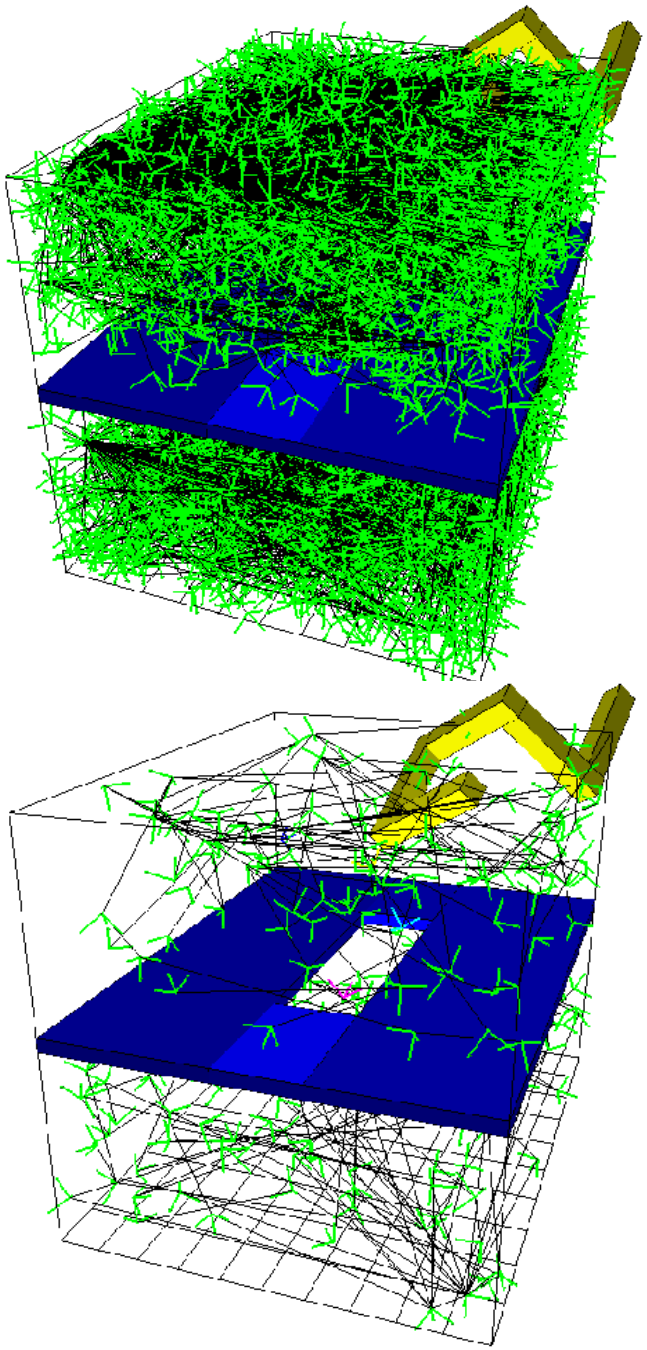
Basic-PRM



Visibility-PRM



$\frac{1}{\epsilon}$	Basic-PRM $l_b(*1000)$	Basic-PRM $\#nodes$	Visib-PRM $l_v(*1000)$	Visib-PRM $\#nodes$	$gain$ $\frac{l_b}{l_v}$
12	18	250	4	13	4.5
25	142	635	8	13	17
50	2.000	2.900	30	13	66
100	31.000	11.700	145	13	213



PRM	Basic	Visibility
Roadmap size	4723	103
CS_{free} coverage	99.9%	99.7%
Random confs	14169	14369
Free confs	4723	4753
Local method #calls	700610	57622
Col. checker #calls	8.725985	1.121790
CPU time	3367 sec	281 sec

Thanks!