Motion planning: sampling-based planners III basic modifications

Vojtěch Vonásek

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Lecture outline

- Issues of sampling-based planning
- Implementation details for sampling-based motion planning
  - Fast nearest-neighbor search
  - Fast collision detection
  - Metrics
- Physical simulations for motion planning
- Trajectory generation

Known issues of sampling-based planning

- One may consider sampling-based planning as a “magic” tool

  …but that’s not true at all!

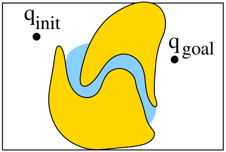
Sampling-based planners have many issues

- Narrow passage problem
  - Difficulty of sampling small region in $\mathcal{C}_{\text{free}}$ surrounded by $\mathcal{C}_{\text{obs}}$
  - Problematic if (all) solutions have to pass that region
- Sensitivity to metric & parameters
  - How to measure distance in $\mathcal{C}$?
  - Selecting a good metric is as difficult as motion planning!
  - Many methods have “too many” parameters
  - Some parameters are hidden (or not well described)
  - How to tune the parameters?
- Supporting functions
  - Collision detection & nearest-neighbor search
  - Fast and reliable implementation

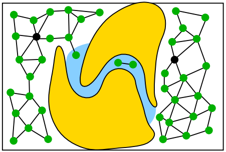
How do we recognize the issue? $\rightarrow$ performance measurement!

Narrow passage (NP)

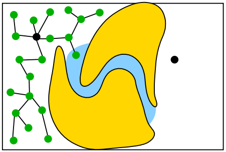
- A region $\mathcal{R} \subseteq \mathcal{C}_{\text{free}}$ with a small volume $vol(\mathcal{R}) < vol(\mathcal{C})$
- Probability that a random sample falls to $\mathcal{R}$ is $\sim vol(\mathcal{R})/vol(\mathcal{C})$
- NP are problematic if their removal changes connectivity of $\mathcal{C}_{\text{free}}$
- NP are regions in $\mathcal{C}$ $\rightarrow$ they are given implicitly
- Location/size/volume/shape of NPs is not known!



narrow passage (NP)



PRM & NP

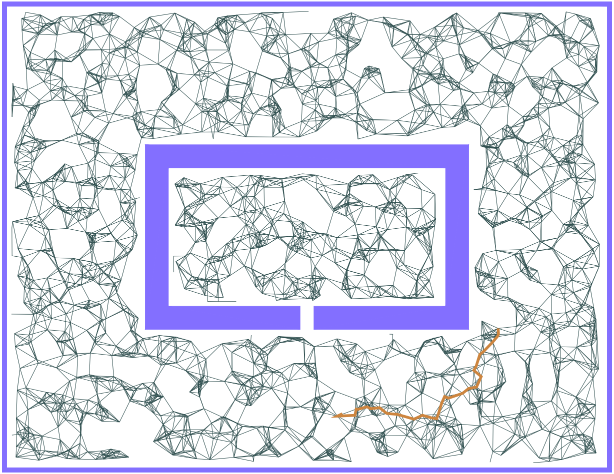


RRT/EST & NP

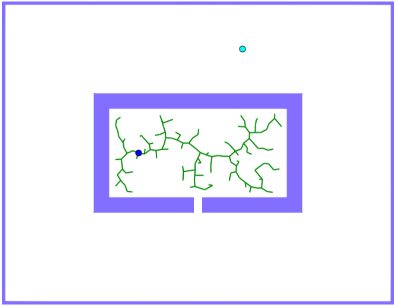
Consequences of having NP

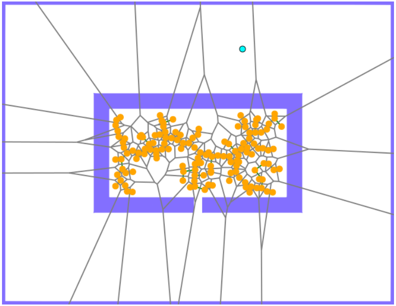
- PRM builds unconnected roadmaps $\rightarrow$ no solution
- RRT/EST cannot enter NP $\rightarrow$ no solution
- Number of samples must be significantly increased
- Runtime is increased

Narrow passage & PRM



Narrow passage & RRT



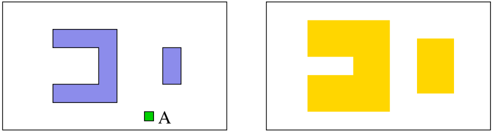


Narrow passage

- Narrow passages are in $\mathcal{C}$
- Sometimes, we cannot (easily) see/estimate them from workspace!
- What makes the narrow passage in the Alpha-puzzle benchmark?

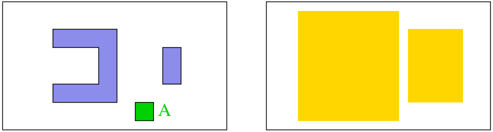
How does $\mathcal{C}_{\text{obs}}$ appears?

- Can we guess shape of $\mathcal{C}_{\text{obs}}$ based on workspace?
- $\textit{vol}(\mathcal{A}) \ll \textit{vol}(\mathcal{O})$



Workspace | Configuration space

- $\textit{vol}(\mathcal{A}) < \textit{vol}(\mathcal{O})$



Workspace | Configuration space

- When obstacles $\mathcal{O}$ dominate, they mostly influence the shape of $\mathcal{C}_{\text{obs}}$

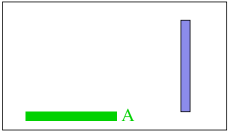
How does $\mathcal{C}_{\text{obs}}$ appear?

- Let $X, Y \subset R^n$, X and Y are nonempty
- Brunn-Minkowski theorem:

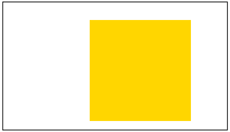
$$vol(X \oplus Y) \geq \left(vol(X)^{\frac{1}{n}} + vol(Y)^{\frac{1}{n}}\right)^n$$

- $vol(\mathcal{C}_{\text{obs}})$ is larger than $\min(vol(\mathcal{A}), vol(\mathcal{O}))$
- $vol(\mathcal{C}_{\text{obs}})$ can be much larger!

Example: $vol(\mathcal{A}) = vol(\mathcal{O})$



Workspace



Configuration space

Improvements

Why improvements of PRM/RRT/EST?

- To cope with the narrow passage problem, improve path quality, speed-up planning, to enable planning in specific cases

Main tricks

- Control distribution of random samples
- Dedicated metrics
- Improved nearest-neighbor search
- Use suitable local planners
- Improve collision-detection

initialize tree $\mathcal{T}$ with q_{init}
for $i = 1, \ldots, I_{max}$ **do**
    q_{rand} = generate randomly in $\mathcal{C}$
    q_{near} = find nearest node in $\mathcal{T}$ towards q_{rand}
    q_{new} = localPlanner from q_{near} towards q_{rand}
    if *canConnect*$(q_{\text{near}}, q_{\text{new}})$ **then**
        $\mathcal{T}$.addNode(q_{new})
        $\mathcal{T}$.addEdge($q_{\text{near}}, q_{\text{new}}$)
        if $\varrho(q_{\text{new}}, q_{\text{goal}}) < d_{goal}$ **then**
            return path from q_{init} to q_{new}

- Many existing modifications, look at survey by Elbanhawi
- Next slides present the basic principle of improvements

☛Elbanhawi, M., & Simic, M. (2014). Sampling-based robot motion planning: A review. IEEE access, 2, 56-77.

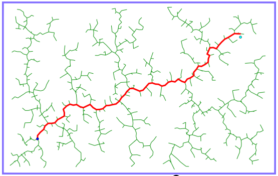
RRT improvement I: goal bias

Observation

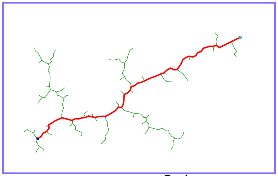
- RRT tree grows towards random samples
- If we samples some region more dense, the tree is “attracted” to grow there

Goal-bias

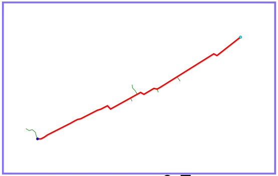
- Random sample q_{rand} is generated in $\mathcal{C}$ with probability $(1 - p_{goal})$, otherwise it is set to $q_{\text{rand}} = q_{\text{goal}}$
- The rest of RRT algorithm is the same
- Improves the performance if the tree can directly reach the goal
- Decreases the performance if the tree is hindered by obstacles



$p_{goal} = 0$



$p_{goal} = 0.1$



$p_{goal} = 0.7$

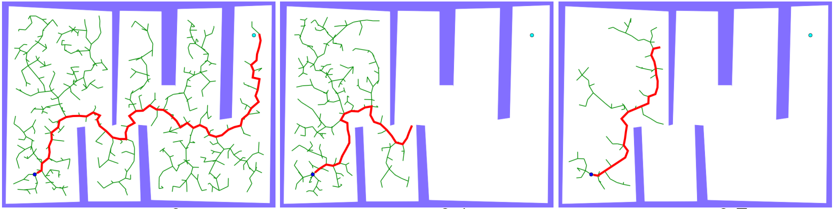
RRT improvement I: goal bias

Observation

- RRT tree grows towards random samples
- If we samples some region more dense, the tree is "attracted" to grow there

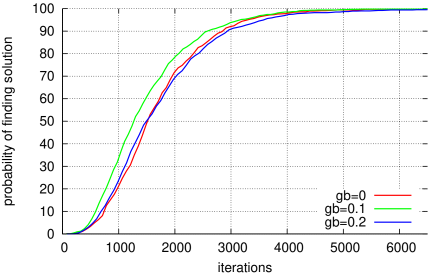
Goal-bias

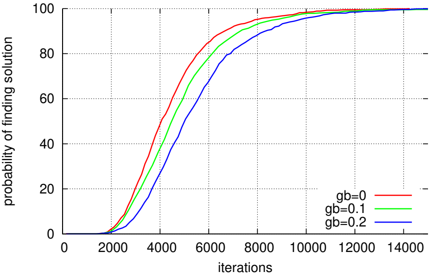
- Random sample q_{rand} is generated in $\mathcal{C}$ with probability $(1 - p_{goal})$, otherwise it is set to $q_{\mathrm{rand}} = q_{\mathrm{goal}}$
- The rest of RRT algorithm is the same
- Improves the performance if the tree can directly reach the goal
- Decreases the performance if the tree is hindered by obstacles

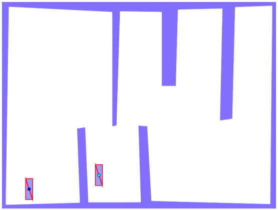


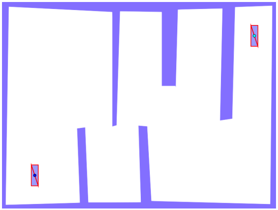
$p_{goal} = 0$ $p_{goal} = 0.1$ $p_{goal} = 0.7$

- Goal-bias may improve or even worse the performance!



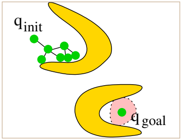






Observation

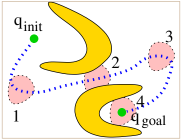
- Goal-bias attracts the tree towards q_{goal}, but the tree may be blocked by obstacles
- Generalization: we can attract the tree toward any region $\mathcal{R} \subseteq \mathcal{C}$ if we sample $\mathcal{R}$ densely



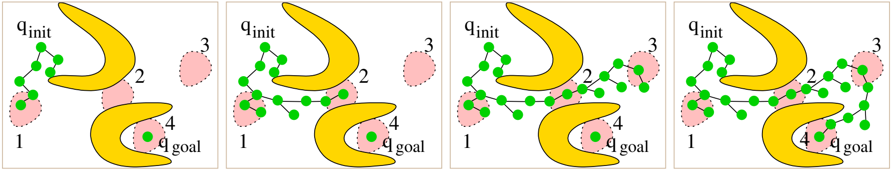
RRT + goal-bias

Guided-based sampling

- Estimate a path that can "guide" the tree in the $\mathcal{C}$-space
- Generate q_{rand} around the path-waypoints (starting from first waypoint) until the tree reaches the waypoint
- Then generate q_{rand} around the next waypoint



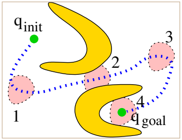
Guiding path

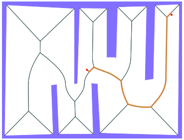


Sampling at 1 | Sampling at 2 | Sampling at 3 | Sampling at 4

How to compute the guiding path?

- Generally, the guiding path has to be located in $\mathcal{C}$!!
- Finding a good guiding path has the same complexity as the original planning problem!
- (i.e., guiding sampling is 'planning solved by planning')
- Practically, we have two options





Guiding path in $\mathcal{W}$
$q = (x, y, \varphi)$
(x, y) from the path
φ randomly

Guiding path in $\mathcal{W}$

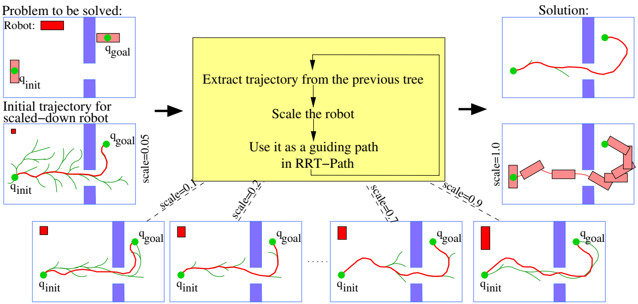
- Path is computed in workspace — geometric planning (Voronoi diagram, Visibility graph, etc.)
- Suitable for low-dimensional problems
- The remaining dimensions are sampled uniformly

Guiding path in $\mathcal{C}$

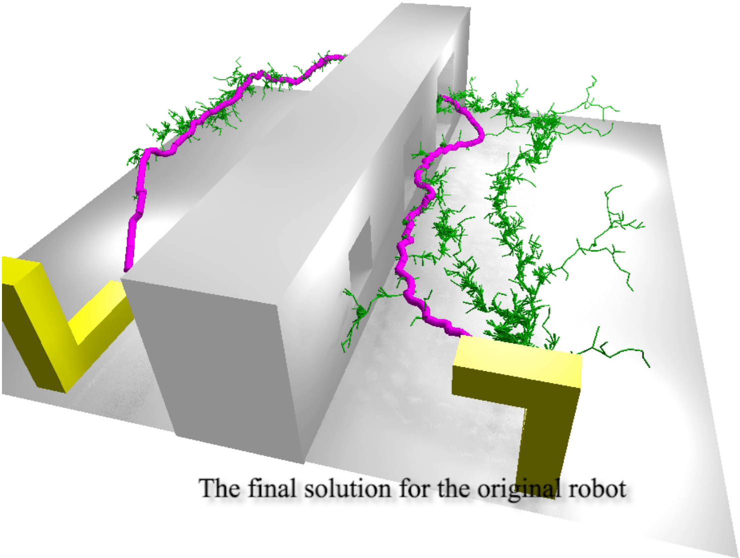
- Path is computed in $\mathcal{C}$ by a simplified search

Guiding path in $\mathcal{C}$

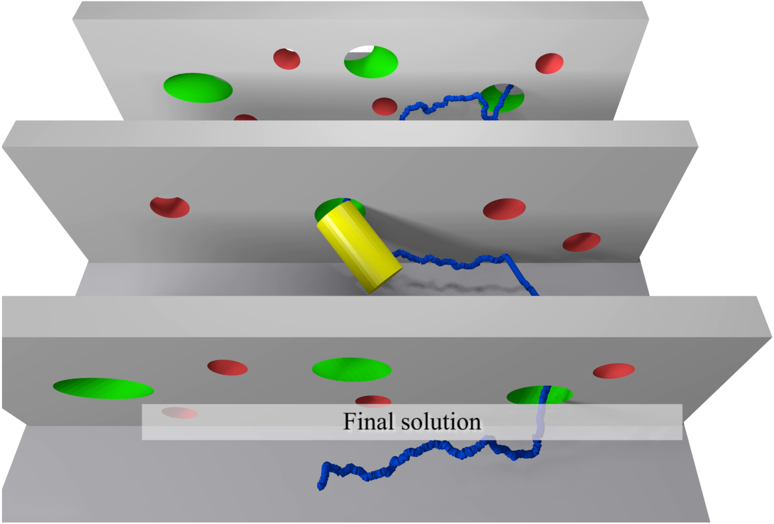
- Problem is simplified — relaxation of constraints
- For example, robot is scaled-down
- Solve simplified planning problem
- Use the solution to generate random samples along it
- The process can be iterative

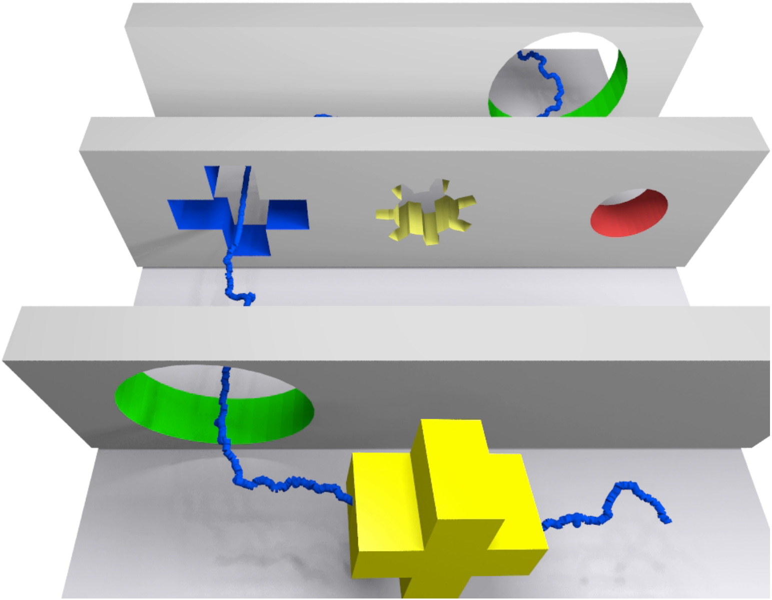


Computing guiding path in $\mathcal{C}$



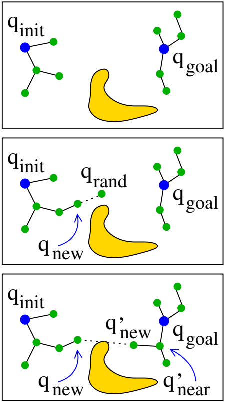
Computing guiding path in $\mathcal{C}$





RRT improvement III: bidirectional search

- Use two trees: $\mathcal{T}_i$ rooted at q_{init}, $\mathcal{T}_g$ rooted q_{goal}
- One tree expands towards q_{rand}, second tree expands towards q_{new} of the first tree

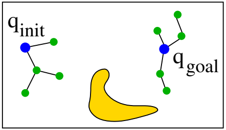


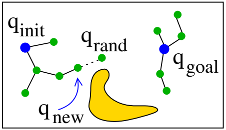
$\mathcal{T}_i$.addNode(q_{init})
$\mathcal{T}_g$.addNode(q_{goal})
for $i = 1, \ldots, I_{max}$ **do**
    q_{rand} = generate randomly in $\mathcal{C}$
    q_{near} = find nearest node in $\mathcal{T}_i$ towards q_{rand}
    q_{new} = localPlanner from q_{near} towards q_{rand}
    if *canConnect*($q_{\mathrm{near}}, q_{\mathrm{new}}$) **then**
        $\mathcal{T}_i$.addNode(q_{new})
        $\mathcal{T}_i$.addEdge($q_{\mathrm{near}}, q_{\mathrm{new}}$)
        q'_{near} = find nearest node in $\mathcal{T}_g$ towards q_{new}
        q'_{new} = localPlanner from q_{near} towards q_{rand}
        if *canConnect*($q'_{\mathrm{near}}, q'_{\mathrm{new}}$) **then**
            $\mathcal{T}_g$.addNode(q_{new})
            $\mathcal{T}_g$.addEdge($q_{\mathrm{near}}, q_{\mathrm{new}}$)
            if *canConnect*($q'_{\mathrm{new}}, q_{\mathrm{new}}$) **then**
                joint trees
                return path from q_{init} to q_{goal}
    $\mathcal{T}_i, \mathcal{T}_g = \mathcal{T}_g, \mathcal{T}_i$ `// swap trees`

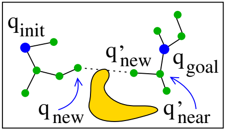
RRT improvement III: bidirectional search

- Use two trees: $\mathcal{T}_i$ rooted at q_{init}, $\mathcal{T}_g$ rooted q_{goal}
- One tree expands towards q_{rand}, second tree expands towards q_{new} of the first tree

- Helps to enter narrow passages (sometimes)
- Connection of two trees
  - Computationally intensive
  - To speed up, performs only if $\varrho(q_{\text{new}}, q'_{\text{new}})$ is small enough
  - Difficult if motion model/constraints have to be considered
- Balanced trees: swap trees if $|\mathcal{T}_i| > |\mathcal{T}_g|$







PRM variants I: sampling strategies

Original PRM/sPRM

- Uniform sampling $q \sim U(\mathcal{C})$

Gaussian sampling: two-samples

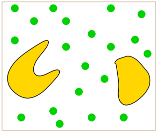
- Uniform sample $q_1 \sim U(\mathcal{C})$, then another sample $q_2 \sim N(q, \Sigma)$ (around q_1 from Gaussian distribution)
- Ignore if $q_1, q_2 \in \mathcal{C}_{\text{free}}$ or $q_1, q_2 \in \mathcal{C}_{\text{obs}}$, otherwise
- add the collision-free one to the roadmap
- Generates the random samples near $\mathcal{C}_{\text{obs}}$ only!

Gaussian + uniform

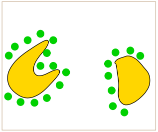
- Combination of two previous methods
- More dense sampling around $\mathcal{C}_{\text{obs}}$ than basic PRM

Bridge test

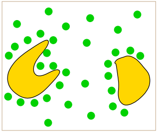
- Generate q_1 and q_2 using the Gaussian method
- Determine the midpoint q' on the line segment $|q_1, q_2|$
- Use q' if $q' \in \mathcal{C}_{\text{free}}$ and $q_1, q_2 \in \mathcal{C}_{\text{obs}}$



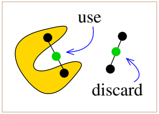
Uniform



Gaussian



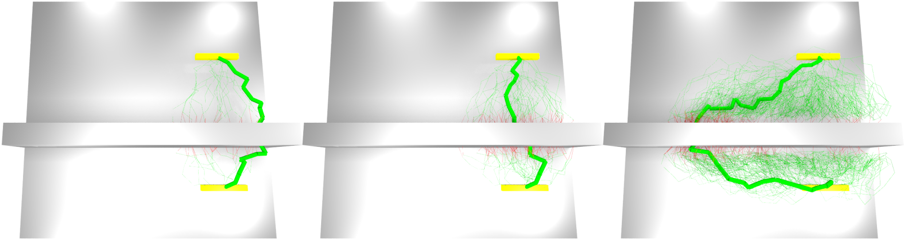
Gaussian + Uniform



Bridge-Test

PRM variants II: Lazy PRM

- Build PRM roadmap, but without collision detection of edges
- After a path is found, edges are checked for collision and the path is recalculated
- If no path is found, extend the roadmap by new samples/edges
- Otherwise, the path is collision-free



$\sim$200 nodes $\sim$500 nodes $\sim$2000 nodes

green: valid edges, red: invalid edges

- Faster planning in certain scenarios, but not always!

☛ R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," IEEE ICRA, 2000.

PRM variants II: Lazy PRM