Motion Planning – Sampling-Based Approaches (Contents)

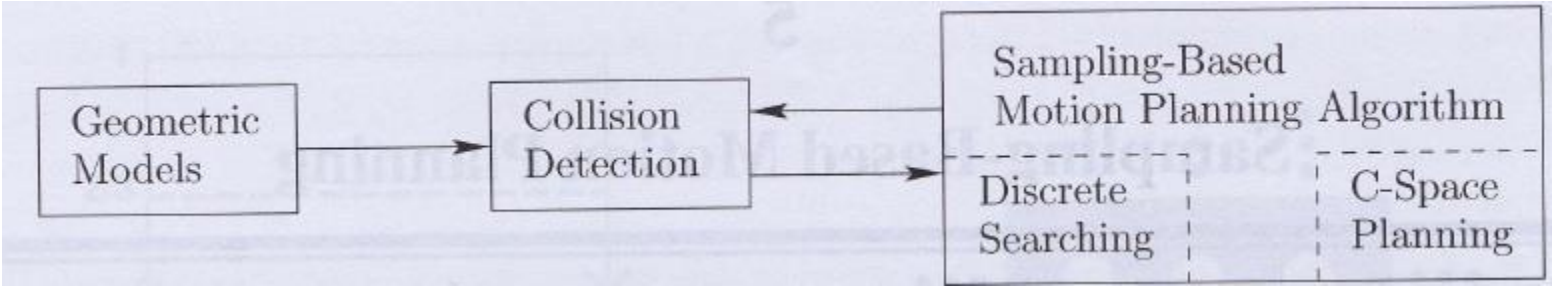
- Planning? Why sampling-based? The philosophy.
- Major properties of sampling-based approaches, comparison to systemic combinatorial approaches, relation to the configuration space approaches (*C-space*).
- Distance, metrics, pseudometrics and density of the sampled search space
- Kinds of sampling, evaluation of the random sampling quality
- Collision detection, hierarchical and incremental methods
- Stepwise/incremental planning
- Relation of dense and discrete planning approaches

- Sampling-based pathway planning
- Random walk, randomized potential field approach
- Rapidly growing random trees, the principle and clones (*RDT a RRT*).

- References

Why sampling-based planning?

- The major motivation is to overcome the necessity to recover the shapes of existing obstacles in the workspace (which is normally represented by $\mathcal{C}_{obst}$). This is substituted by a suitable sampling method, which delivers the necessary and local information to drive the planning process.
- The sampling-based approaches do not necessarily require a complete world model (but may be more efficient, if the model is known)
- The art of sampling the configuration space (*C-space*) in fact fulfills the functionality of „collision avoidance“ which conveys the information need for decision-making of the planning process.



Sampling-based approaches use the collision avoidance as a „black-box“ which separates the planning mechanism from the geometric and kinematic model of the case. The gathered/derived samples form the world model serve to accomplish a discrete planning (decision-making process).

Sampling-based planning, core properties I

- All the sampling-based planning algorithms are *incomplete,* i.e. do not ensure, that an existing solution to the planning task will be found in finite (contrary to that, this is the major feature of combinatorial approaches)
- Nevertheless, an acceptable result can always be obtained via finding of an *approximate solution*. This is enabled by the property of „density of samples“ (denseness) obtained through the sampling process. The major featuring of the „denseness“ herein is, that the samples are not limited to describe any admissible configuration, if the count of the samples $\rightarrow\infty$.
- From a deterministic point of view (a systematic search) and while the sampling is sufficiently dense (densely complete) the consequent alternatives appear as:
  1. If an acceptable solution exists, it‘ll be found in finite time.
  2. If a solution doesn‘t exist, and even the sampling is densely complete (sufficiently dense), the algorithm will not end ever.
- The preceding implies so called *probalistically complete* algorithms, which means that the probability of finding an acceptable solution (if there is any) goes to $\rightarrow 1$, if sufficiently high number of samples is performed
- Nevertheless, the speed of convergence remains a problem (its‘ assurance is a hard problem: the better/smarter way the samples are taken, the convergence of the algorithm appears faster and less samples need to be processed to obtain the same admissible solution.

Sampling-based planning, core properties II

- Sampling-based algorithms can be configured for single- or multi-query operation.
  - Single-query mode is represented by a pair (*start, goal)* request, while the algorithm is expected to workt until the goal is achieved (or reports an error)
  - The multi-query approaches, i.e. capable of delivery of more solutions for more pairs (*start, goal)* are much more computationally intensive in a phase of pre-processing, i.e. while creating suitable data structures to allow to deliver multiple solutions at once later on (on condition, the structure and shape the obstacles mainly) of the world do not vary meanwhile.

Distance and metrics I

- All the sampling-based approaches require distance definition to properly measure distance between two points in space → **metric space, metrics**

Def.: Metric space (X, *ρ)* with metrics (distance measure) *ρ: X* x *X* → *R that satisfies for any a,b,c* ∈ *X* the following axioms:

1. *ρ(a,b)* ≥ *0* (distance is non-negative)
2. *ρ(a,b)* = 0 only and and only if *a=b* (distance to itself equals to zero, reflexivity)
3. *ρ(a,b)* = *ρ(b,a)* (symmetry)
4. *ρ(a,b)+ ρ(b,c)* ≥ *ρ(a,c)* (triangle inequality)

- The most common are so called L_P-metrics in R^n that satisfy: $$\rho(a,b) = \left(\sum_{i=1}^{n} |a_i - b_i|^p \right)^{1/p}$$

In particular:

1. L_1 metrics –so called Manhattan, or orthogonal metrics (distance along rectangular axes, sum of differences of coordinates)

1. L_2 metrics – Euklidean metrics (Euklidean distance in R^n)

2. L_∞ metrics – Max metrics $$L_\infty(a,b) = \max_{1 \le i \le n} \{ |a_i - b_i| \}$$

Distance and metrics II

- Other metrics for robotics and motion planning:

  1. Metrics of complex mumbers is suitable for any C-space which is a subset of R^2, that satisfies:

  $$\{(a,b) \in R^2; a^2 + b^2 = 1\} \quad \rho(a_1,b_1,a_2,b_2) = \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}$$

  2. The preceding metrics doesn't account distance along circles, what may be desirable → *metrics of angles*, can be built by considering angles Θ_1 a Θ_2 (respecting the reachability of Θ_2 z Θ_1 via two possible directions, clockwise or anticlockwise):

  $$\rho(\Theta_1,\Theta_2) = \min\{|\Theta_1 - \Theta_2|, 2\Pi - |\Theta_1 - \Theta_2|\}$$

  ...or in complex numbers space (a point is denoted as: *a+jb*):

  $$\rho(a_1,b_1,a_2,b_2) = \cos^{-1}(a_1 a_2 + b_1 b_2)$$

- **Pseudometrics:** Construction of the „distance measure" does not satisfies all the aforementioned axioms 1. to 4., but still being capable of giviving the correct direction of the robot to the goal

  Examples:
  - Objective function for plannning based on used power need not to satisfy the symmetry axiom (i.e. kinematic constrains may not allow backward motion)
  - Use of potenial field for planning – the repulsive components may give rise to local extrems and break the triangle inequauality axiom.

Space sampling I, density of sampling

- If random sampling/sequences are „probabilistically dense“, it makes the sampling usable for the planning, i.e.:
  - It there is an interval of length *e* for random sampling the whole workspace by *k* samples, the probability of not hitting this interval is $(1-e)^k$.
  - On condition $k \to \infty$, the value of $(1-e)^k \to 0$, which for the given conditions means that the interval *e* contains probably at least one sample. I.e. Such the infinite sequence of random and independent samples stands „dense with probability =1“ (What has not the meaning of a guarantee; even the probability of choice of particular samples equals to zero, their choice remains still possible!)
- Random sampling – simplest possible way to go, very suitable for C-space cases while the samples can be taken stepwise, respecting coordinate(s) system - i.e. in an independent way for each dimension of the C-space.
- The previous still allows equalized random coverage of the workspace (to cover Cartesian space with deterministic methods is a much more complex case)
- Generation of (pseudo)random numbers
  - Standard algorithms are not truly random!
  - Classic linear congruence generator; for *a, M* being relatively prime and *c* such, that: $0 \le c < M$, where M is possibly $\gg 1$

$$y_{i+1} = a y_i + c \mod M$$

Where pseudo-random numbers x_i from range <0,1> can be obtained as: $x_i = y_i / M$ (and may be periodic).

Space sampling II, how to test/qualify random sampling?

- Due to previous reasons a computer-made sampling remains still somehow deterministic, not fully random → a need to evaluate uniformity/randomness of the taken samples

- Straightforward way is application of *Chi-squared* test, which verifies how far is the computed statistics from the expected one.

  Example:

  Assume a workspace being split into *n* x *n* pixels. Let *P* represent a set of *k* randomly chosen samples. Intuitively, each pixel (area) shall contain k/n^2 samples., what allows to define an error function representing the level of confidence, that the aforementioned assumption is correct:

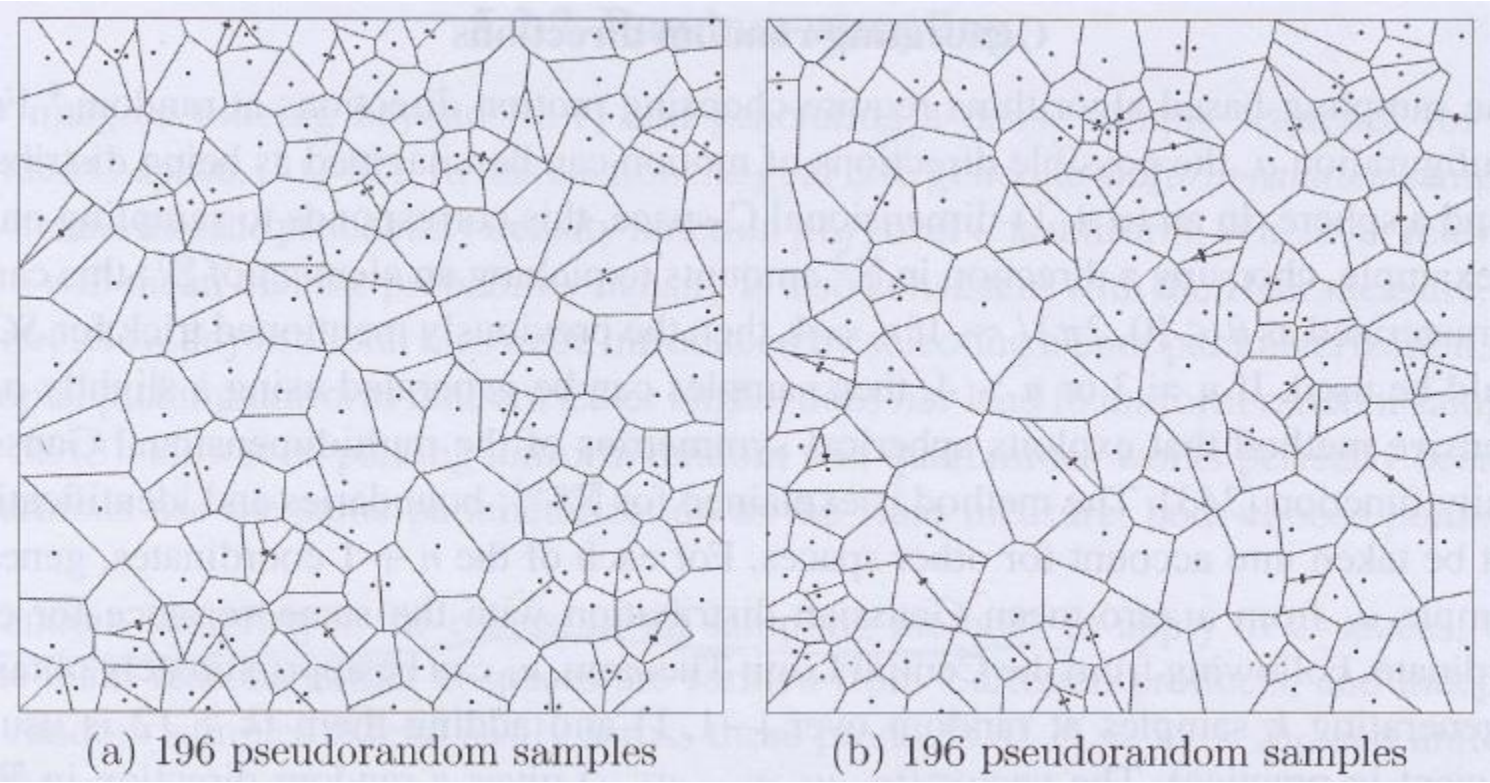
$$e(P) = \sum_{i=1}^{n^2} \left(b_i - k / n^2 \right)^2$$

  Where *b* stands for number of samples in pixel *i* (in fact, the error function *e(P)* denotes Chi-squared distribution of a random variable). *Value e(P) = 0* means best-possible uniformity of samples distribution, i.e. These are not random in that case.

  Then, generally said, the density shall tend be *e(P) » 0* for the expected case.

- Making-use of Voronoi diagrams – each sample has assigned a Voronoi region *Vor(x).* For each point $y \in Vor(x)$, for which *x* is the closest sample to *y,* while using Euclidean metrics. Diversity of size/area and shape of the Voronoi regions is being evaluated for the criterion then.

Space sampling III



An example of 196 pseudo-random samples (for better visibility of randomness, the corresponding Voronoi regions are superimposed.

Space sampling IV, pseudo-random sampling

- Sampling with Van der Corput sequence
  Based on principle of binary coding of interval lenght followed by interval/binary representation splitting and swap the LSB and MSB parts. The sequence exhibits a property that every open subset of the interval (in terms of the interval binary quatization) contains, at least, one sample.

i	Naive Sequence	Binary	Reverse Binary	Van der Corput	Points in $[0,1]/\sim$
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	

Detection of collisions I

Detection of collisions stands for a key step to drive a planning process using sampling.

- Each placement of a sample shall be decided in terms of possible collision with the environmental obstacles, i.e. Whether it can be used for building the plan? (a necessary condition)
- Collision detection is often built as a „back box“ and is completely sufficient to provide correct evaluation of collision cases only. It doesn‘t have direct binding to the planning process itself; nevertheless its‘ computational intensity may be very high, so it can slow down the used planner (optimization method) performance.
- There are many diverse approaches for collision detection (exact, heuristic...)
- The most common are methods based on verification of a suitable condition (i.e. a description of an obstacle; a test of presence of the particular sample in the model of the obstacle(s)) with respect to the $\mathcal{C}_{obst}$... of the configuration space. Decision, whether the sample (configuration) stands for a collision case or not is considered for the collision detection outcome (a binary value).
- In the case of 2D world with convex obstacles and robot is computational complexity of the algorithm linear. Nevertheless, it is always easiest to determine if the configuration is a collision or not without a need for complete recovery of the $\mathcal{C}_{obst}$ (model of obstacles/world).

Detection of collisions II

- Determination of collisions relies on computation of distance *d* in between two sets *d:*$\mathcal{C} \rightarrow$*<0,∞], which corresponds to least distance* within the existing point pairs *e* and *f* from given point sets E and F*:*

$$\rho(E,F) = \min_{e \in E} \{ \min_{f \in F} \{ \| e - f \| \} \}$$

where $\| \cdot \|$ denotes Euclidean norm and simultaneously is satisfied: $E \cap F = \varnothing \Rightarrow \rho(E,F) = 0$

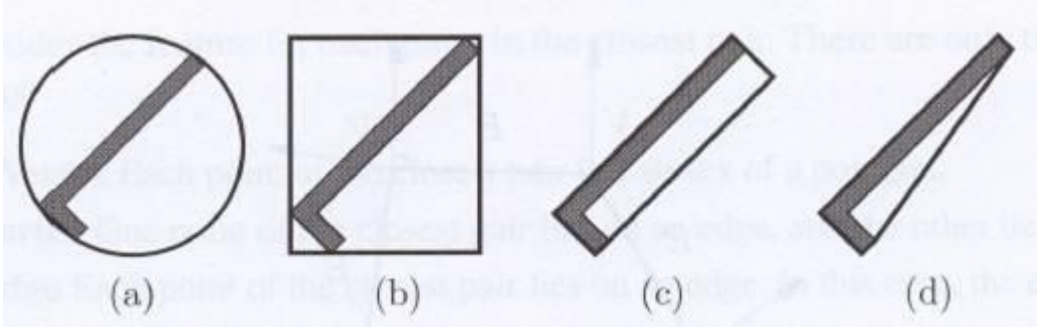
- To simplify the computation and save the time, the collision detection is efficient to be performed in two stages: rough detection (long-distance) and precise detection (in a close vicinity):
  - **Rough collision detection** – testing of relative position of (convex) hulls of objects/obstacles and the robot; extreme/corner points of surrounding frames, etc. Hashing may be used for reduction of number of possible collision mutual combinations
  - **Precise collision detection** – detailed execution on the level particular elements/points of obstacles and robot – hierarchical and incremental approaches possible, see bellow...

Hierarchical approaches I

- Recommendable for collision detection of „larger“ objects (an obstacle and a robot).
- The method performs decomposition of an each object into its‘ basic primitives (i.e. triangle decomposition) and orders these into a *tree-like structure*, for which:
  - Each vertex in the tree binds to a corresponding and limited component/region (a subset).
  - Root vertex represents the whole object

The art of the decomposition is guided by 2 contradictory requirements:

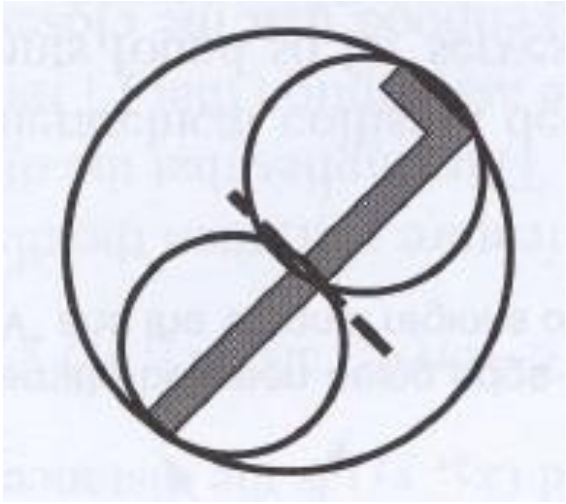
- Limiting component/region attaches the object in the tightest possible/closest way (is a hull)
- The method of testing for intersection of such pairs of regions needs to be simple (for the algorithm efficiency reasons)



Various kinds of limiting regions (hulls): (a) circular sphere, (b) limiting rectangle; co-linear with the coordinate system, (c) oriented rectagle, (d) konvex hull

Hierarchical approaches II

- The tree buildup goes top-down, i.e. the limiting region is always split into successor regions of similar size/area. If the object model is decomposed into primitives (i.e. triangles, circles, etc...) the splitting process is run unless a similar count of primitives in each of the successors is achieved
- The splitting process is executed unless decomposition into basic (primitive) regions and shapes is achieved. This is important for the ease of checking for mutual collisions.



Circular sphere denotes the vertex that describes the whole object. After further splitting (dashed line), 2 smaller circles represent description of both the halves of the original object (2 vertices at lower level)

Hierarchical approaches III

- Detection of a collision in the tree structure:
  1. Assume having objects E and F and the corresponding tree structures T_e and T_f, the possible collision of which is being investigate
  2. If the root vertices T_e and T_f do not collide, both the objects E and F are **not in a collision situation** → **end**.
  3. If the root vertices T_e and T_f collide, the limiting regions of all of successors of T_e are compared to the region.
  4. If non of the regions in step 3 do not collide, the limiting region for T_f is substitute by all the limiting regions of all its‘ direct successors (the successors on the next lower level).
  5. Recursive recall of step 2, unless the leaf vertices of the tree structure have not been achieved, otherways both the original **objects do collide** → **end**.

Remark 1.: If the decomposition has been done into complete primitives (i.e. Triangels), the testing for collision is performed inbetween these primitives.

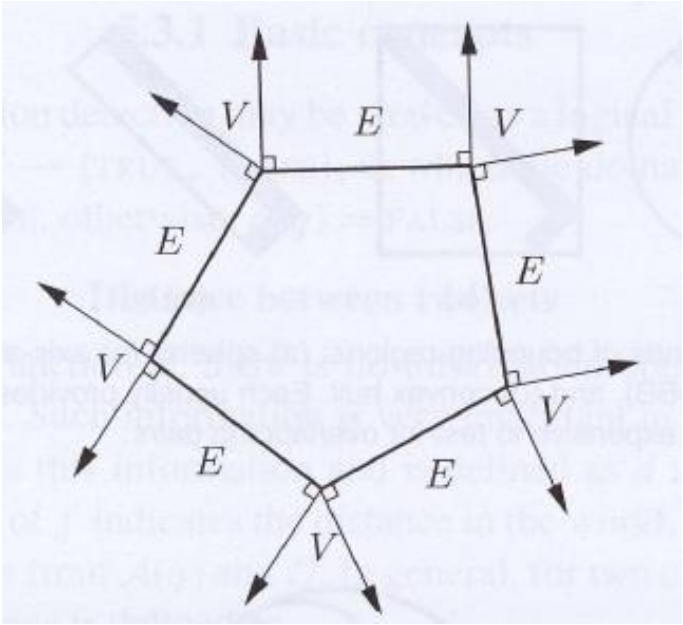
Remark 2.: Extension of the algorithm towards computation of a real distance of the limiting regions allows further cutting the tree → computation speedup.

Incremental approaches I

- *Incremental distance computation* – assumes that in between two subsequent requests for collision detection no substantial variation in structure and shape of the scene appears (the objects move, or vary, in a negligible way only).
- ***Advantage:*** The preceding assumption allows to attain nearly constant computational time/complexity for objects of a convex polyhedron type. Non-convex polyhedrons can always be decomposed into convex ones.
- ***Drawback:*** Models of objects and robots need to be coherent, i.e. all of their primitives must be aligned to each other and the the shapes need to be closed. .. Contrary to that, this does not allow existence of isolated objects or their segments/walls, or objects with missing side (Models of the admissible objects must not be as a simple set of primitives, which implies computationally intensive pre-processing due to performing a recognition step in fact..)

Incremental approaches II

- Collision detection can also apply the principle of investigation of mutual relations of object features (herein a 2D case)
- Each object, a polyhedron, having *n* vertices can be described using *2n* features (vertices and edges), which correspond to Voronoi regions (see the figure bellow).



- Each pair of objects, that exhibits potential danger of mutual collision can be classified into the following cases:
  - *Vertex-to-vertex; whereas both the pair elements are vertex points from each of the polygons*
  - *Edge-to-vertex; one of the closest points from one polygon is located on its' edge, since the other one stands for one of the vertices of the other polygon*
  - *Edge-to-edge; both the closest points ale on respective edges of the polygons (note, the edges are parallel in this case)*

  ...for which the distance of these sets (polyhedrons) can easily be computed.

Incremental sampling and plan building I

- Single query algorithms are given a single pair of a *Start* and a *Goal*, whereas: (q_I, q_g) for each robot and a set of obstacles (the world model) are given.

  → no need to perform any preliminary computations of needed structures as for the cases of multi-query setups

  → the motion planning can be understood as a task of a *state-space search*, with the following adjustments:

  - An „action“ execution is substituted by generation of a „segment of a path“ (note the step 3 of the algorithm bellow)
  - The searched graph is not oriented; the edges correspond to pathways in between locations/vertices. (contrary to an oriented graph with edges representing actions)

The basic approach – a single query path planning algorithm

Initilalization

- Let‘s have *G(V,E)* representing non-oriented graph, that consists of at least one verticle *V,* while *E* may contain no edge at all. Typically, *V* represents the *start* and/or the *goal* and optionally also some other points of the free space $\mathcal{C}_{free}$.

  1. **Vertex selection.** (Vertex Selection Method, VSM) Select the vertex q_{cur} *for expansion.*
  2. **Local planning.** (Local Planning Method, LPM) For a suitable and new q_{new}, that need not to be from the set of existing vertices *V,* try to find a path from q_{cur} to q_{new}, such, that does not exhibit a collision. If a search for collision-free path fails, continue with step 2. If not, continue.

Incremental sampling and plan building II

The basic approach – a single query path planning algorithm (continuation)

4. **Edge insertion into the graph.** *Since q_{new} does not belong E,* the previously found path (or its‘ part, respectively) is to be inserted into the *E as the edge q_{cur} to q_{new}.*
5. **Has the solution been achieved?** Verify, if the graph *G* already represent the final requested path, the solution? (This step is easy in discrete cases, if a unique search tree exists; in other cases the decision may be very complex and computationally expensive)
6. **Return to step 2.** Iterates the preceeding unless the target solution is achieved, or some other ending condition is satisfied (as the algorithm may operate in unlimited way under certain specific circumstances)

Remark. The afore used graph *G* is a topological graph, or sometimes labeled also as a „roadmap“

Incremental sampling and plan building III

- In cases of certain configuration of obstacles, a possibility of bug-trapping the method exists. The bug-trap situations do not allow simple and straightforward placement of samples at particular typical locations and → causing slow-down or even compete failure of the method.
- This can be resolved making-use of i.e. combinations of diverse search methods (LMPs) as:
  - Single direction search – the search tree expansion from the *start* to the *goal* position only.
  - Double direction search– simultaneous growing of 2 search trees one towards the other, from the *start* and from the *goal* position (mainly resolves the non-symmetric bug-trap situations)
  - Multi-directional search – growing multiple search trees from multiple origins in the scene (mainly resolves double bug-trap cases, random vs. systemic choice of the root location(s)...)

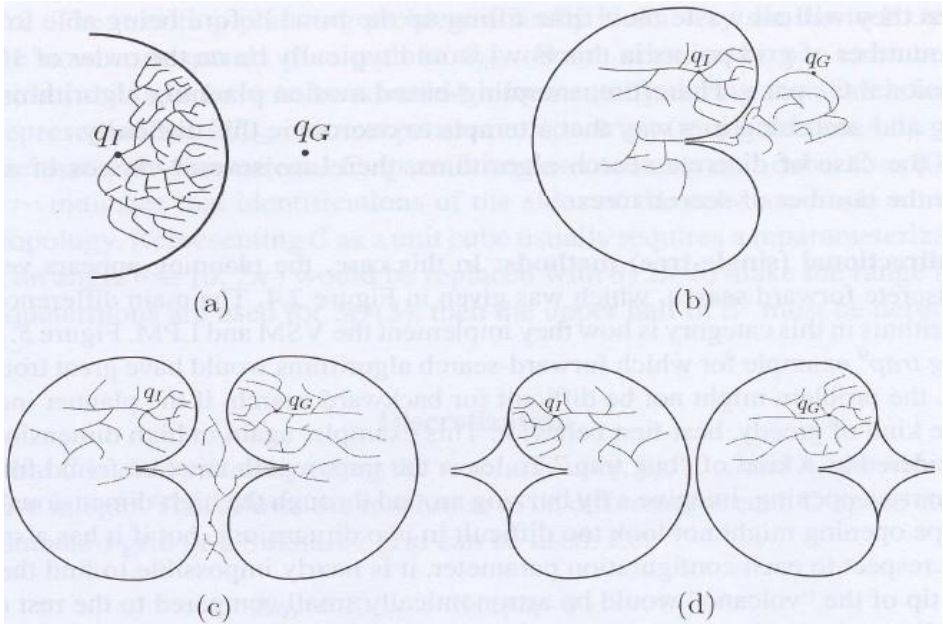
- Hard configurations for sampling-based planning algorithms can serve for benchmarking:

  (a) preferred search through the shielded area,

  (b) bug-trap configuration with a hard escape out of the limited area,

  (c) double bug-trap case desires multi-directional search with root points in, and even out of the limited regions,

  (d) hard-resolvable situation...

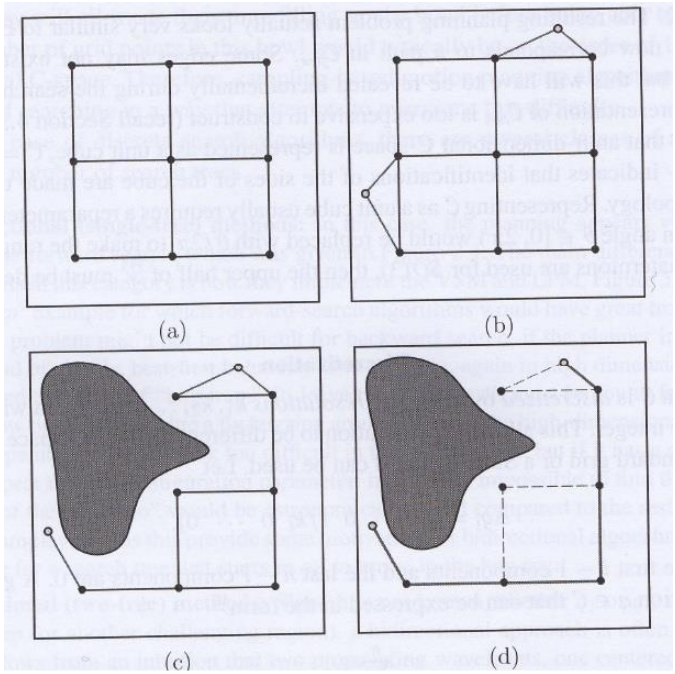


Relation to discrete planning

- Sampling-based planning is possible to be combined with methods for discrete planning approaches.
  - → Brings substantial speed-up of the plan computation via imposing a priori constrains on the search space, i.e. by setting a grid over the search space (configuration *C-space*), or over the corresponding topological graph (roadmap)
  - → There nodes (roots) is necessary to choose respecting the search space coverage
  - → A sampling-based algorithm for planning can be launched on such a reduced space representation. This may lead to a solution for the plan in very few steps.

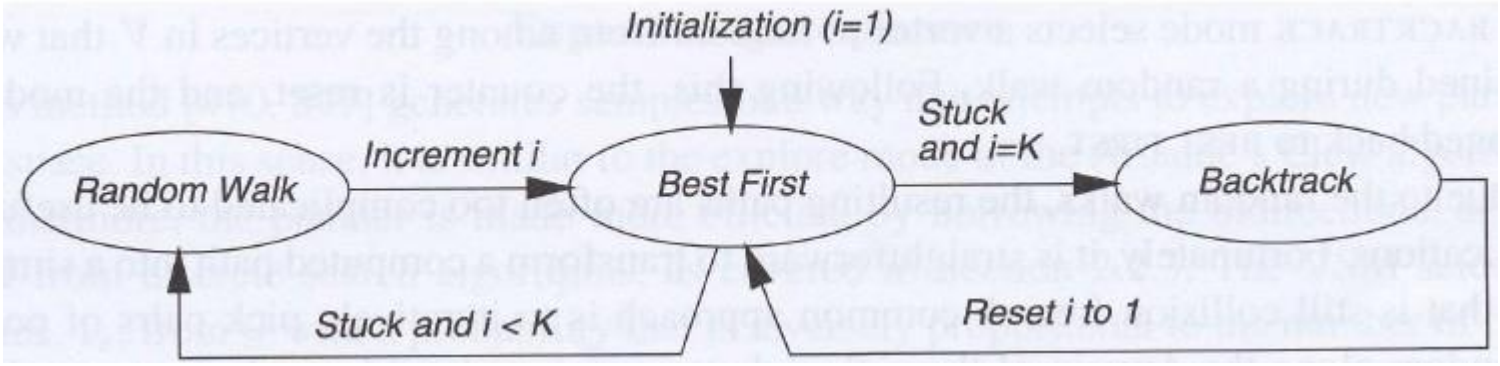
A topological graph (a roadmap) may be built during the space search.

The number of vertices which bind roads is less, i.e. it allows determination of a solution via selection from very few samples much faster.



Random walk and randomized potential field

- Comparison of the random and the systemic approaches:
  - Random approaches (a random walk) suffer on performance finding a passageway in the cases of problematic/hard structures appearance (see the benchmarking cases or other similar situations)
  - Systemic (non-informed) approaches to search of the state space tend to be computationally intensive (for example: a 10-dimensional space having 50 samples/dimension → creates 50^{10} alternatives)
  - Hard implementation of informed methods → usage of various pseudo-metrics (i.e. potential field) is computationally intensive as well and/or dos not assure non-existence of other local extremes, other than the goal(s).
- The afore given bottlenecks can be bridged through combination of informed search techniques and a random search.
- The criterion for the particular method switching is detection of a possible trapping in local extreme of the objective function, or the number of steps performed by each of the methods:

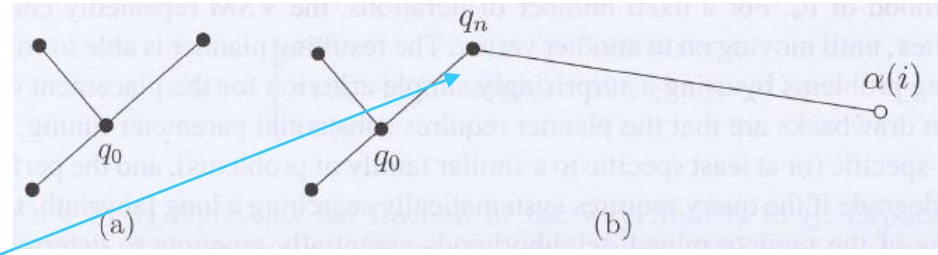


Rapidly Exploring Dense Trees (RDT) I

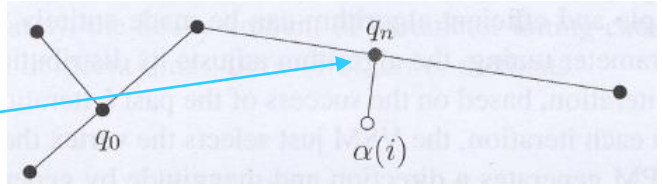
- Incremental sampling/planning approach, that delivers good performance without setup of any parameters, the idea:
  - Stepwise construction of a search tree, which improves resolution of the space description over time/number of steps (no need for parameter tweaking, that are related to the resolution at all). In result, it is capable to coverage of all the workspace in a sufficiently „dense“ way.
  - The built dense tree is deterministic or randomized (RDT), *Rapidly Exploring Random Trees (RRT)* stand for a special case.

Algorithm to construct a dense search tree:

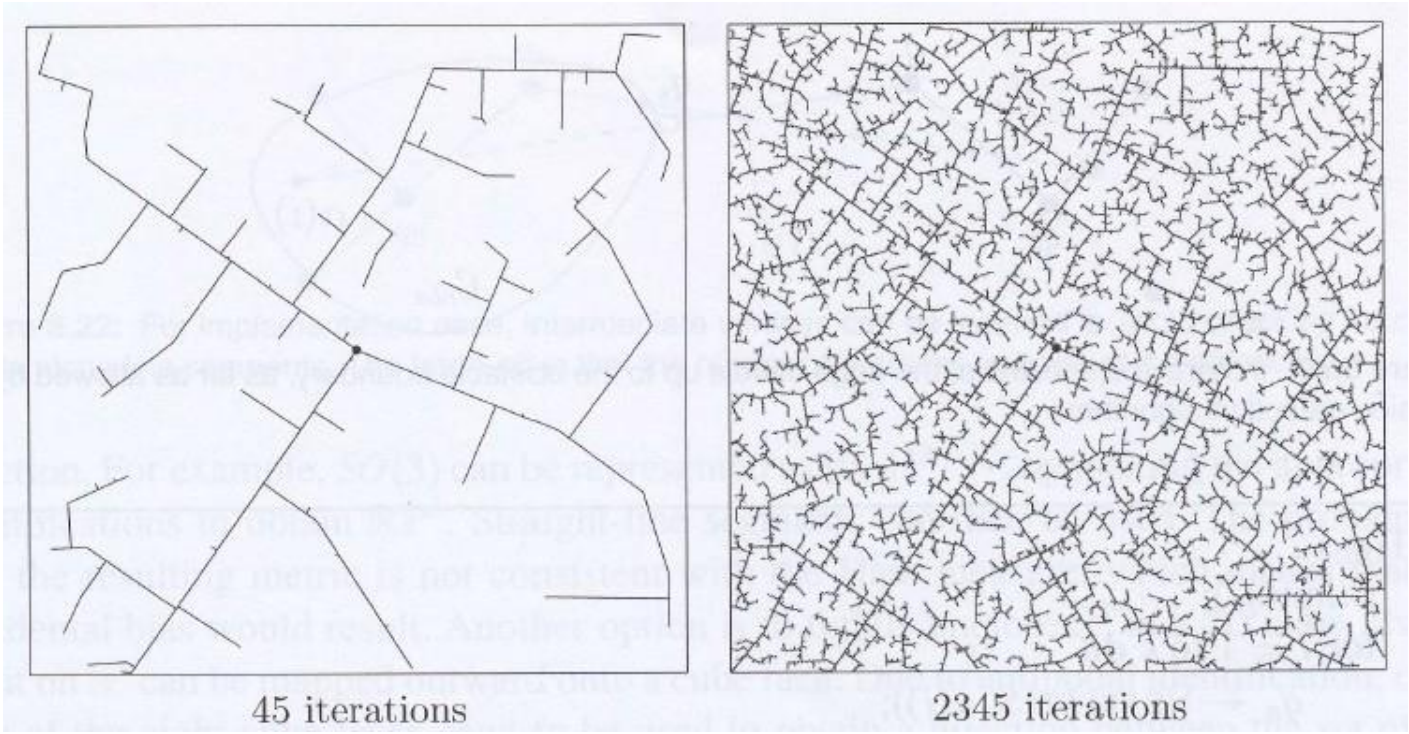
```
SIMPLE_RDT(q_0)
1  G.init(q_0);
2  for i = 1 to k do
3      G.add_vertex(α(i));
4      q_n ← NEAREST(S(G), α(i));
5      G.add_edge(q_n, α(i));
```



Function *NEAREST* determines a vertex of the so far existing tree *S(G)* which is closest to the newly generated vertex *α(i)*; there appears either 1 such a new object with creation of an each new vertex *(the closest is a vertex),* or 2 new edges *(an edge is the closest → splitting of the existing edge)*.



Rapidly Exploring Dense Trees (RDT) II



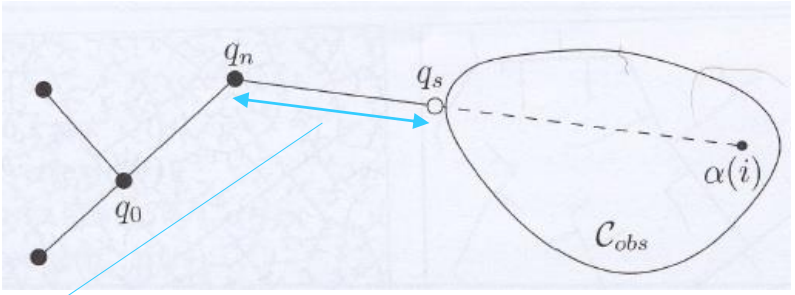
Behavior of a RDT: The initial iterations are very effcient/fast in approaching yet not visited areas. Subsequetly, the coverage is refined – in a limit case, the method assures full coverage with probability =1

Rapidly Exploring Dense Trees (RDT) III

- Representing C_{obst} v RDT may be resolved in the phase of the tree generation → via defining of an end-condition for approaching „the nearest point by a object border“ in the direction towards the generated vertex $\alpha(i)$. The closest point q_n is given in the same manner (without any respect to existence of an obstacle. The corresponding edge belongs to q_s, only, see the drawing bellow:

A RDT algorithm for the cases with obstacles:

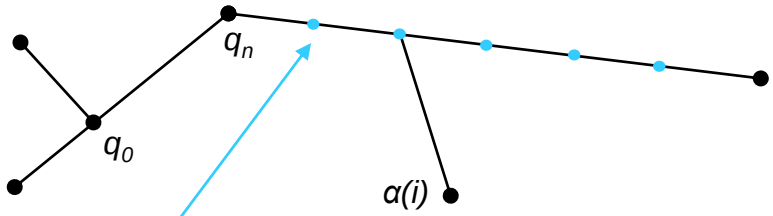
```
RDT(q0)
1  G.init(q0);
2  for i = 1 to k do
3      qn ← NEAREST(S, α(i));
4      qs ← STOPPING-CONFIGURATION(qn,α(i));
5      if qs ≠ qn then
6          G.add_vertex(qs);
7          G.add_edge(qn, qs);
```

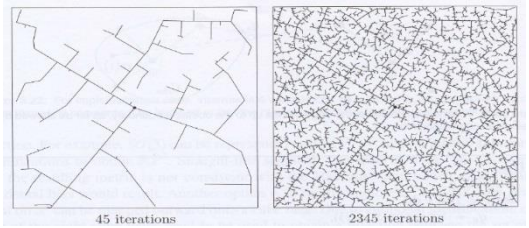


- The least admissible distance of the point q_s from the border of the obstacle – this is denoted by the applied algoritm for collision avoidnace; in some cases with small distances of q_n from the obstacle the edge $q_n q_s$ needs not to be created at all..

Rapidly Exploring Dense Trees (RDT) IV

- Buildup of the function *NEAREST* (a search for the “closest point in the tree”) offers two alternative approaches:
  - **Exact solution:** Applies a hierarchical approach. Computation of the distance of the new vertex *α(i)* towards branches of the search tree, which were obtained in the early steps of its‘ buildup (the early major branches). This delivers approximate information, which part of the tree has candidate points for the closest point (the computation is done at linear costs).
  - Consequetly, the *NEAREST* value is refined in an iterative way up to a desired resolution.





  - **Approximate solution:** Relies on re-sampling of the search tree. Each particular edge is inserted with additional vertices so, that two neighboring vertices distance is within a given threshold *Δq*. The other inner points on edges are omitted for further computation and the distance is computed for *α(i)* and each of the re-sampled tree vertices.
  - The final accuracy/resolution is denoted by the chosen distance *Δq* of the new vertices.

Reference:

- LaValle, S. M.: Planning Algorithms, Cambridge University Press, U.S.A, 2006, 826 pp. ISBN 0-521-86205-1