Special and common methods for planning and problem complexity

- Why the *C-space* use is efficient?
- Complexity of a path planning and its’ relation to the *C-space*
- Complete and incomplete approaches
- Potential field-based planning
- Space decomposition approach, examples
- Using roadmaps for planning

- References

Why the *C-space* based planning stands efficient?

- Reduces complexity of the planning approach/solution for robots with physical dimensions (many constraints) in Euclidean space → substitute of complex constrains/cases by multi-dimensional space (*C-space*) and point-like robot (simplifies implementation of the planning approach, reduces number of the planning constraints by additional *C-space* dimensions, that stand for these constraints) process.
- *C-space* stands for unified framework good for comparison and evaluation of various planning algorithms

Major drawback(s):

- The motion and path planning is *continuous* from principle (given by the *C-space* definition)

Which can be resolved via:

  - Making the planning space discrete (the *C-space*)
  - Making the trajectory discrete
  - Discretizing both the previous items

Complexity of the path-planning

- Complete kind of path planning (rare) is computationally intensive. A „complete planner“ either: (a) finds an admissible solution (path), or (b) reports, that a solution doesn‘t exist

  ...which needs to search through the whole state space.

- More common approaches rely on incomplete methods (approximate methods), that: (a) fetch at least „some“ solution (mainly not very optimal) but being delivered in much shorter time (or in a given time, „any time algorithms“)

  or

  (b) do not find any solution at all (nevertheless, any confidence, that there is no solution does not still exists in such cases)

- Essentially, the complete methods exhibit a computational complexity of:

  –Exponential order with the *C-space* dimension (corresponds to degrees of freedom)

  –Polynomial order with the complexity of C_{obst} in the *C-space* (obstacles, number of their borders, the order of their algebraic description)

2+1 basic (and complete) ways to resolve a planning problem

(1) A complete decomposition of the workspace (an exact cell-decomposition)

- Double-exponential complexity $\sim 2^{2^d}$, where *d* stands for space dimension
- Based on the principle of decomposing the C_{free} into simple unique regions (= elementary cells, pixels or other primitives) and related connectivity relation between these (i.e. a graph of neighborhood)

(2) A method of roadmaps

- Simple-exponential complexity 2^d, where *d* denotes the space dimension
- Relies on computation of a „silhouette“ of the C_{free} space. Represents connectivity in C_{free} by a graph in a form of a network of 1D curves (transitions inbetween nodes, or roads)

The previous holds for a <u>complete planning </u>(which is not very practical), so simplification makes the task easier to compute:

- <u>Simplifying geometry/shape of the robot/obstacles via their approximation</u>
- <u>Limiting of number of DoFs $\longrightarrow$ constraining the dimension of the workspace</u>
- <u>Simplifying of road(s) description(s), decreasing the order of trajcetories, i.e to linear segments, etc.</u>

Therefore the planning is typically performed as a 2-step procedure as:

(1) Determination of the connectivity of the free workspace C_{free} and representing it as a graph (or as a function)

(2) Search for the final path in the graph (or search along the function values)

Following the afore aspects, the other possible method for planning enables to approach the problem as an (objective) function optimization problem - a „potential field“ approach

The potential field approach

- Takes the advantage of a potential field kind of functions (harmonic potentials), continuos and smooth functions that satisfy the additional Laplace condition:

$$\nabla^2 f\left(\overset{\rho}{x}\right) = 0$$

denoting the $f\left(\overset{\rho}{x}\right)$ as a conservative field function, which is differentiable at any point and exhibits monotonic and steady sinking (or rising) behavior and has only a single and global extreme at the loci of the target configuration (position).

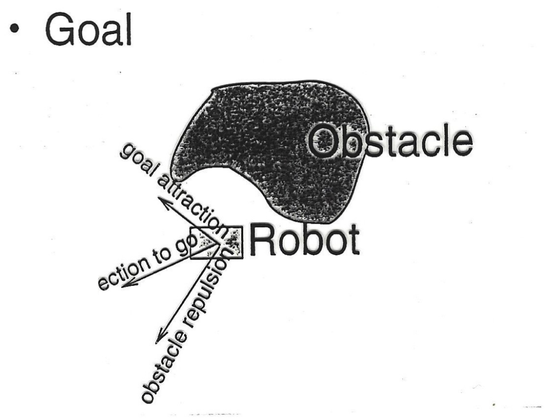
- As to the afore mentioned approach, the optimal path can be determined by performing a steepest descent (or ascent) search along the function values.
- Computational complexity of the solution is proportional (linear) to the path length (or number of transitions/steps if the case of discrete representation of the path)
- The idea of the potential field approach is bases on creating vector (gradient) field (i.e. a force-field of virtual forces) using the aforementioned potential (differentiable) function U

$$U\!: \; C_{free} \longrightarrow \Re, \text{so that} \qquad \overset{\Psi}{F}(q) = -\overset{\Psi}{\nabla} U(q)$$

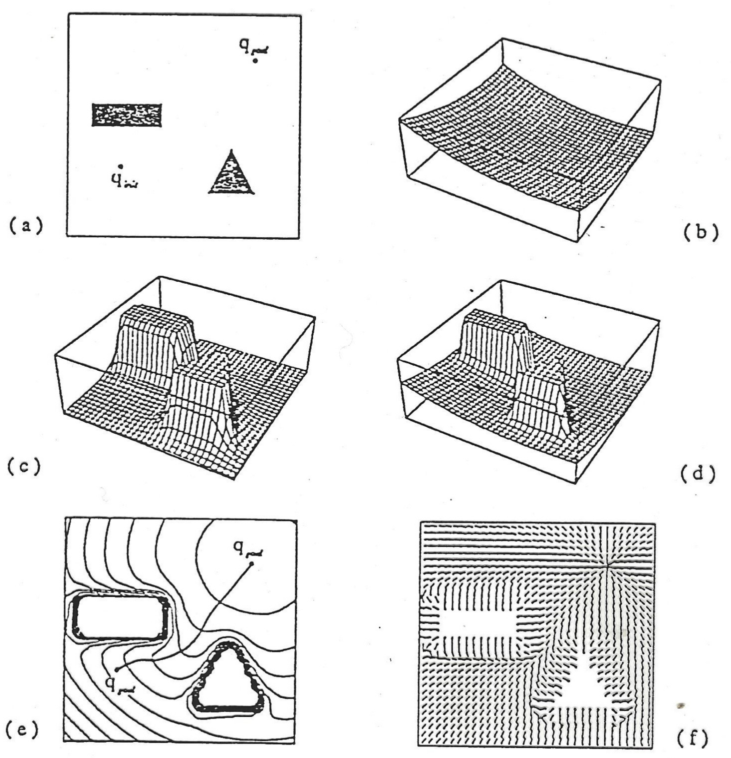
- The afore force field $\overset{\Psi}{F}(x)$ then attracts the robot to the goal position, whilst it does not guarantee, that the robot will move right along an obstacle border - a problem (!).

  Can be resolved by using yet another potential field, that repulses the robot from the obstacle border, so that:

$$\overset{\Psi}{F}(x) = \overset{\Psi}{F}_{att} + \overset{\Psi}{F}_{rep} = -\overset{\Psi}{\nabla} U_{att} - \overset{\Psi}{\nabla} U_{rep}$$



The basic situation for the otential field planning.



(a) The scene setup, (b) Attractive potential, (c) Repulsive potential, (d) Superposition of the repulsiveand attractive potentials, (e) Equivalent potentials lines, (f) Force vector field

Building the potentials

Example 1:

(a) The electric field in homogenous conductive environment (i.e. rezistive foil (2D), or liquid (3D)) The obstacles represented by insulated regions.

(b) A liquid flowwing through an environment. The obstacles represented physically by themselves.

Example 2: The „harmonic“ potential function is denoted by the additional condition $\nabla^2 f(x) = 0$ (a conservative field) does not exhibit any local extremes and assures finding of the solution always. The harmonic field is far more costly to be computed. Electric or gravitation field are conservative and generate harmonic potentials. Magnetic field is not conservative and denotes simple potential field.

The potential filed for path planning usage is normally generated in an arificial way, an example of possible buildup:

The attractive field: $U_{att}(q) = \frac{1}{2}\xi \left\| q - q_{goal} \right\|^2$, where ξ denotes scale $\xi \in \Re^+$

and the term $\|\|$ stands for Euclidean distance

Besides, always $U_{att}(q) \geq 0$ and $U_{att}(q_{goal}) = 0$ as well as U_{att} is continuously differentiable $\forall q \in C_{free}$, so that always exists: $F_{att}(q) = -\nabla U_{att}(q) = -\xi(q - q_{goal})$

The repulsive field:

- Creates a barrier in a vicinity of the obstacle to prevent the robot to get too close to, and collide with the obstacle
- Frequent common requirement is, that a robot in sufficiently large distance is not influsenced by the repulsive field at all:

$$U_{rep}(q) = \frac{1}{2}\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2, \forall\rho(q) \le \rho(q_o)$$
$$U_{rep}(q) = 0, \text{ othewise}$$

Wherereis ρ_0 stands for the distance of influence and the squared term above denotes the inverted distance between the robot and the obstacle such that: $\rho(q) = \min_{q \in obstacle} \left\| q - q' \right\|$

and featuring: $U_{rep}(q) = 0, \forall\rho(q) \ge \rho_0$ as for the distance of influence

and $U_{rep}(q) \to \infty; \forall\rho(q) \to 0$ as for the obstacle

Since the boundary of the obstacle is at least piecewise continuously differentiable, the repulsive force stands:

$$\vec{F}_{rep}(q) = -\vec{\nabla}U_{rep}(q) = \eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2(q)}\vec{\nabla}\rho(q); \forall\rho(q) \le \rho_0$$
$$\vec{F}_{rep}(q) = -\vec{\nabla}U_{rep}(q) = 0, \text{ otherwise}$$

Computation of the robot path using the potential field approach

The main steps:

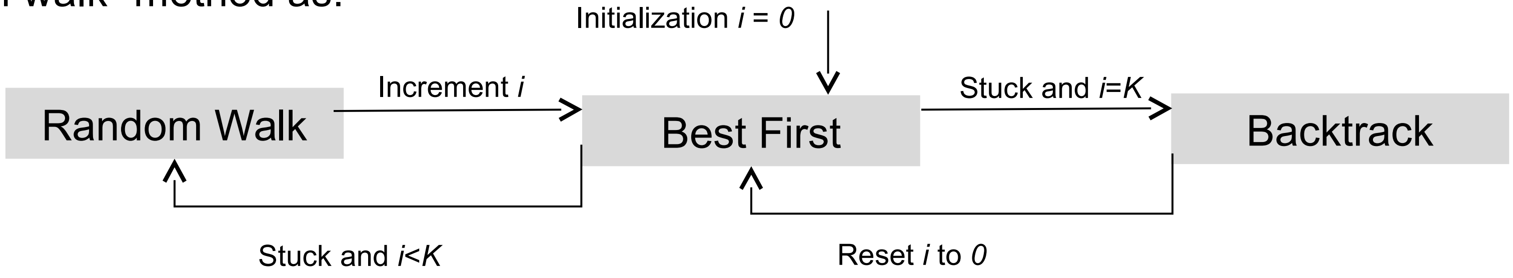
(1) Discretization of the robot workspace C_{free}

(2) Computation of the potentila function over the robot workspace with the minimal value posed at q_{goal}

(3) Search for the optimal (steepest descent) path from teh current standpoint to the goal q_{goal} (gradient driven optimization, best-first search, greedy approach, etc.)

Further remarks

- In the case of using regular (non-harmonic) potential function the search procedure may stuck in local extreme of the force-field. This can be resolved in multiple ways: restart of the search with modified initial conditions, simulated anealing, etc.
- Application of a „randomized potential – a combination of a „potential-based“ method and a „random walk“ method as:



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:C→<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: $E \cap F = \emptyset \;\Rightarrow\; \rho(E,F) = 0$

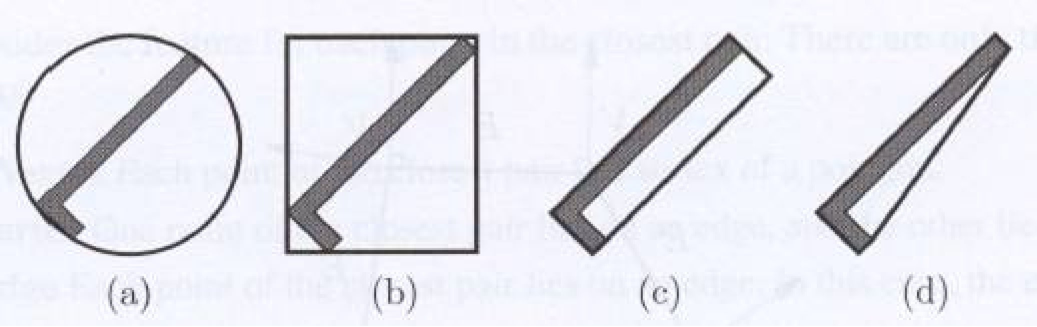
- To simplify the computation and save the time, the collision detection is efficient to be performed in two stages: rough detection (long-distance) and precise detection (in a close vicinity):
  - **Rough collision detection** – testing of relative position of (convex) hulls of objects/obstacles and 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:

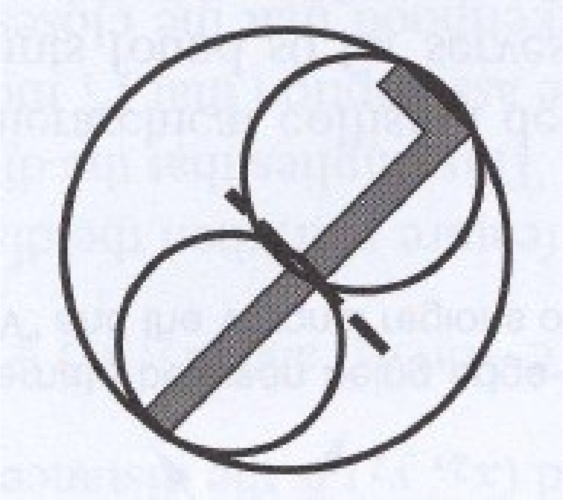
- Limiting component/region attaches the object in the tightest possible/closest way (is a hull)
- The method of testing for intersection of such pairs of regions needs to be simple (for the algorithm efficiency reasons)



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.

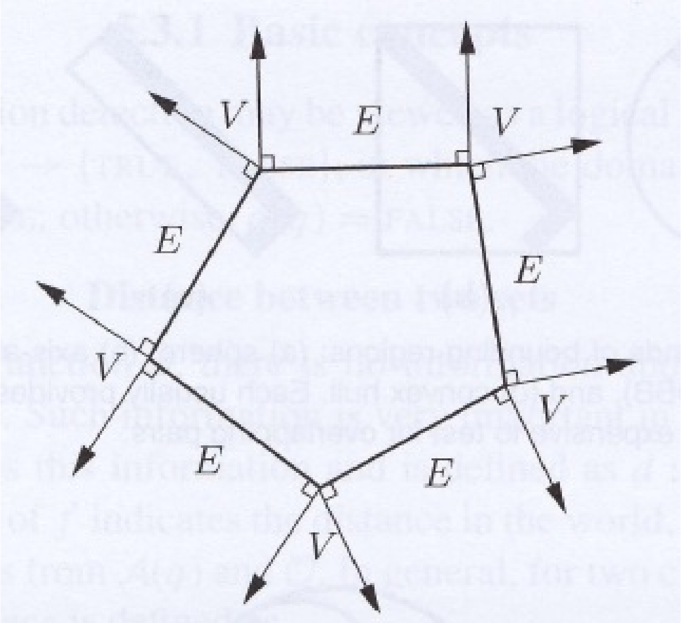
Remark 2.: Extension of the algorithm towards computation of a real distance of the limiting regions allows further cutting the tree → computation speedup.

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 $\mathcal{C}_{free}$.
  - **Vertex selection.** (Vertex Selection Method, VSM) Select the vertex q_{cur} *for expansion.*
  - **Local planning.** (Local Planning Method, LPM) For a <u>suitable and new</u> 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)...)

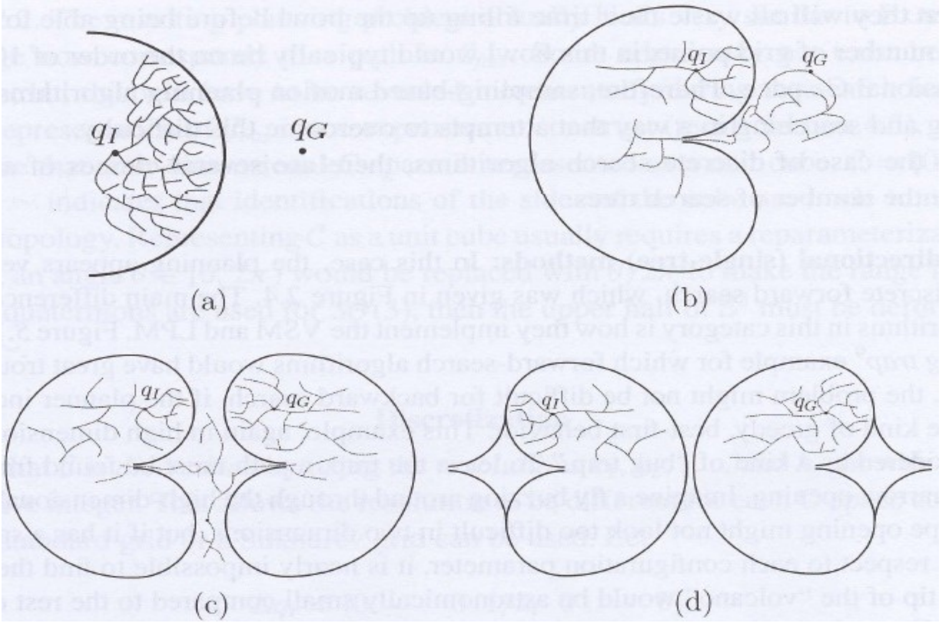
- Hard configurations for sampling-based planning algorithms can serve for benchmarking:

(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...

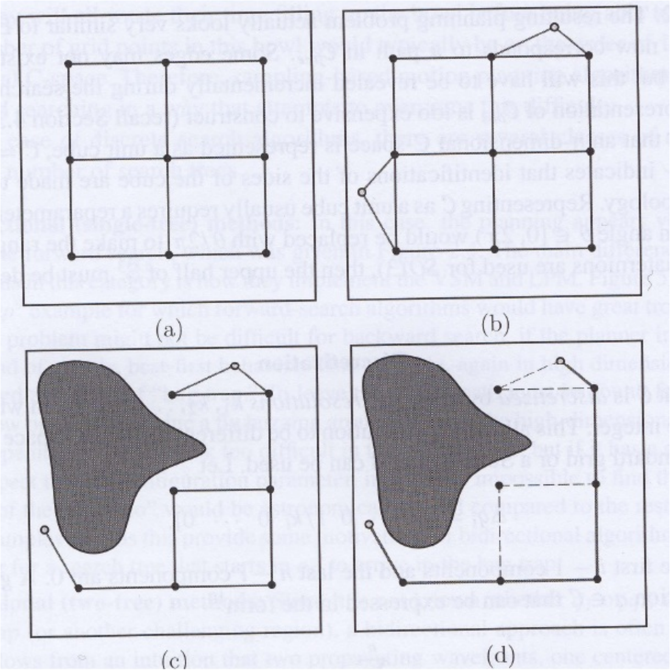


Relation to discrete planning

- 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.

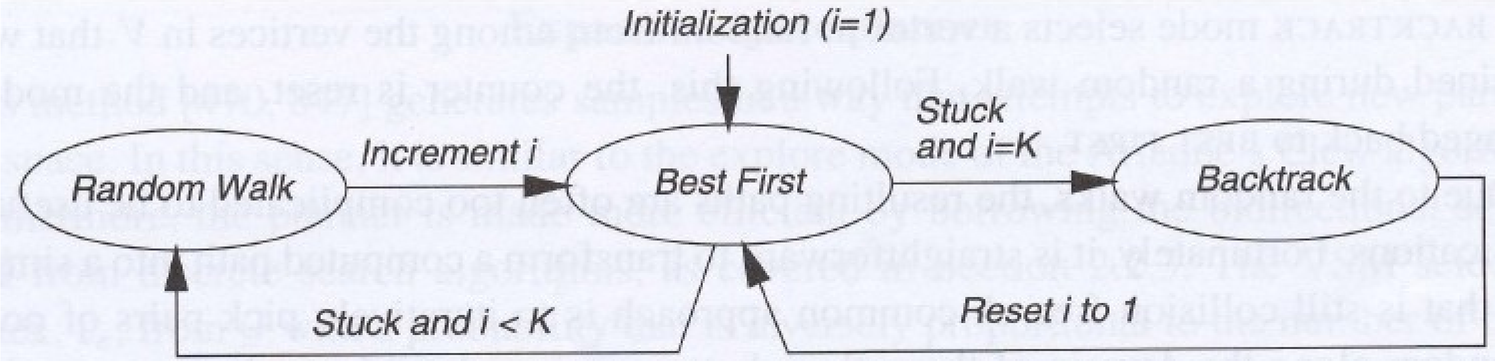
A topological graph (a roadmap) may be built during the space search.

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

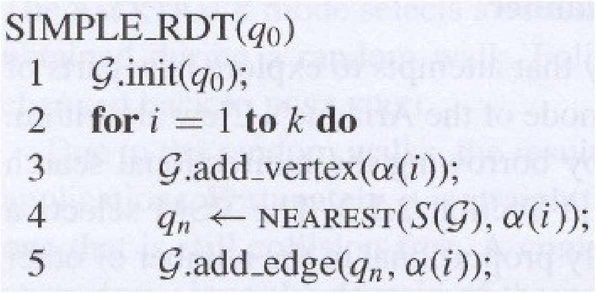
- Comparison of the random and the systemic approaches:
  - Random approaches (a random walk) suffer on performance finding a passageway in the cases of problematic/hard structures appearance (see the benchmarking cases or other similar situations)
  - Systemic (non-informed) approaches to search of 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:



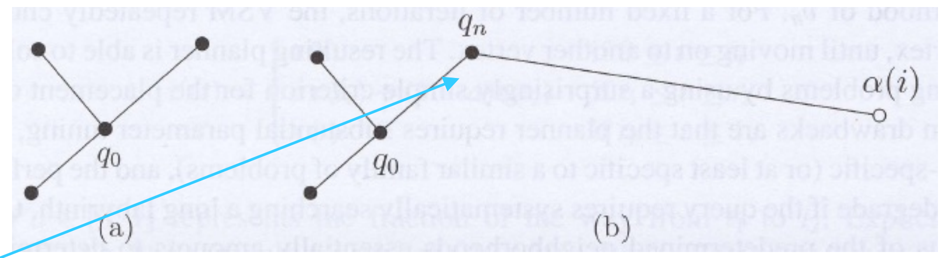
Rapidly Exploring Dense Trees (RDT) I

- 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.

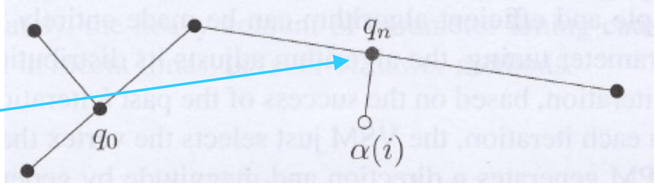
Algorithm to construct a dense search tree:



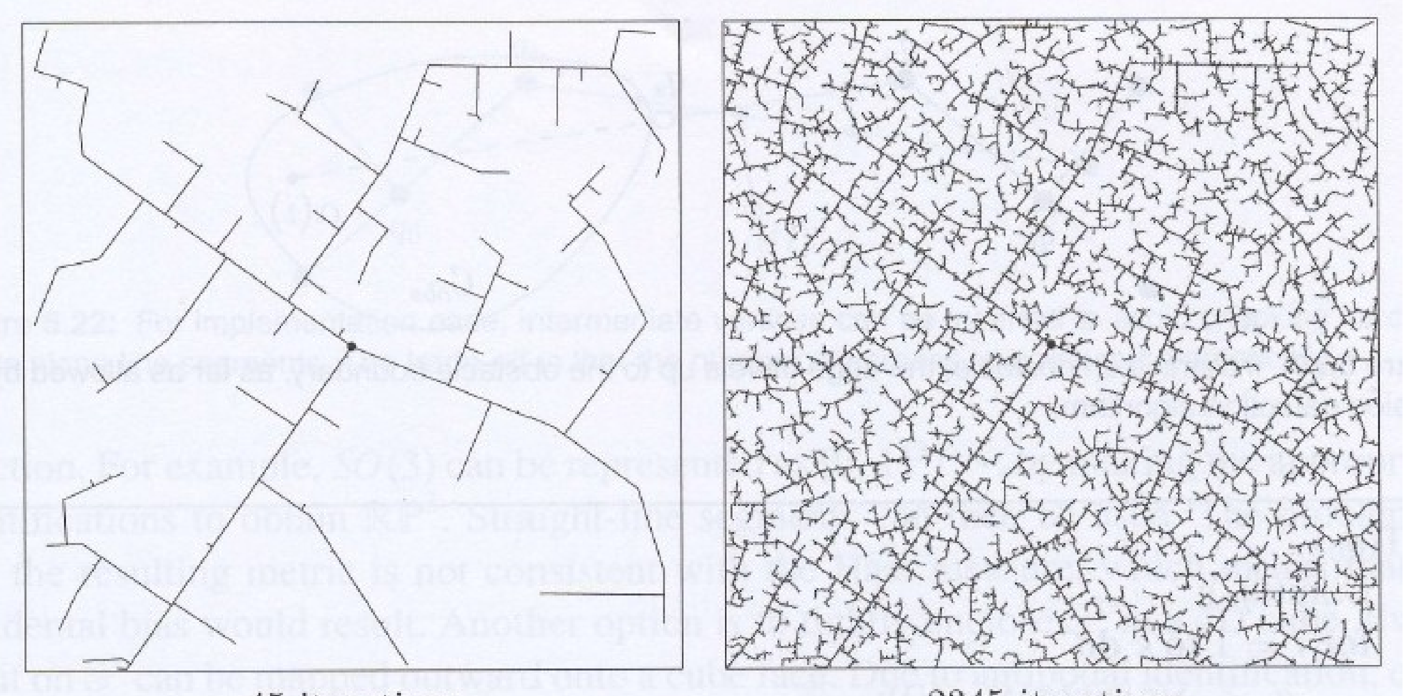
```
SIMPLE_RDT($q_0$)
    $\mathcal{G}$.init($q_0$);
    for $i = 1$ to $k$ do
        $\mathcal{G}$.add_vertex($\alpha(i)$);
        $q_n \leftarrow$ NEAREST($S(\mathcal{G}), \alpha(i)$);
        $\mathcal{G}$.add_edge($q_n, \alpha(i)$);
```



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



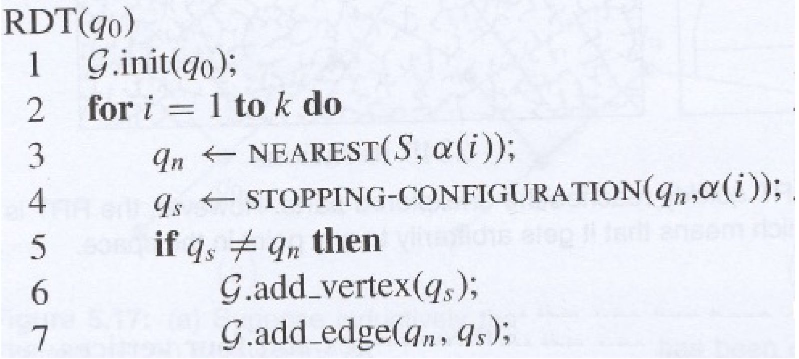
45 iterations 2345 iterations

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

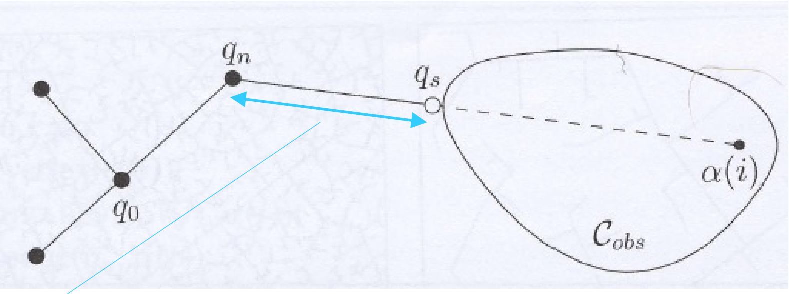
Rapidly Exploring Dense Trees (RDT) III

- 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 *α(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:



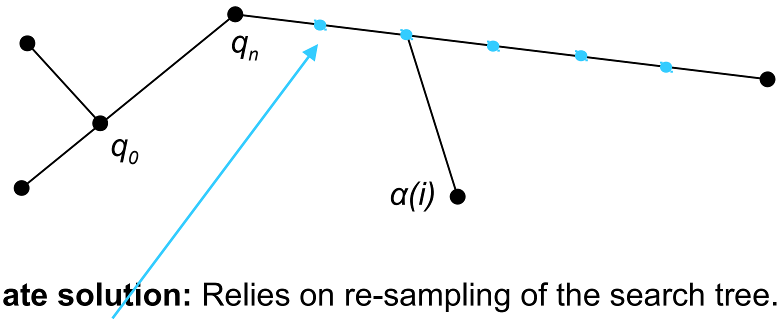
```
RDT($q_0$)
    $\mathcal{G}$.init($q_0$);
    for $i = 1$ to $k$ do
        $q_n \leftarrow$ NEAREST($S, \alpha(i)$);
        $q_s \leftarrow$ STOPPING-CONFIGURATION($q_n$,$\alpha(i)$);
        if $q_s \neq q_n$ then
            $\mathcal{G}$.add_vertex($q_s$);
            $\mathcal{G}$.add_edge($q_n, q_s$);
```

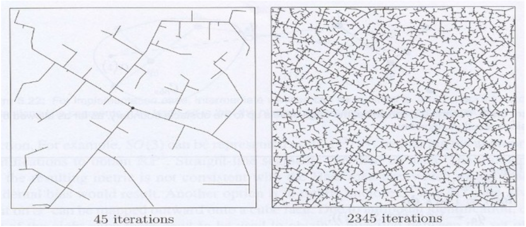


- The least admissible distance of the point q_s from the border of the obstacle – this is denoted by the applied algoritm for collision avoidnace; in some cases with small distances of q_n from the obstacle the edge $q_n q_s$ needs not to be created at all..

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:

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