Sampling-Based Algorithms

⇒ All the planners discussed so far relies on an explicit representation of the geometry of Q_{free}.

↳ Because of this, as the dimension of the Configuration space grows, these planners become impractical.

⇒ Sampling-based methods employ a variety of strategies for generating samples and for connecting the samples with Path to obtain solution to path-planning Problem

(samples) → {Collision-free Configuration of the robot}

★ The Development of Sampling-Based Planners

⇒ Sampling-based planners were developed at a time when several Complexity results on the path-planning Problem were known.

↳ The implementation of the geometric algorithms is very difficult and not practical.

⇒ The development of methods that approximated the free configuration space, heuristic planners, potential-field methods, and the early sampling-based planners.

⇒ The probabilistic RoadMap planner (PRM) demonstrated the tremendous potential of sampling-based methods.

↳ PRM fully exploits the fact that it is cheap to check if a single robot configuration is in $\mathcal{Q}_{free}$ or not.

⇒ PRM uses

↳ Coarse sampling to obtain the nodes of the roadmap.

↳ Very fine sampling to obtain the roadmap edges.

↳ free paths between node configurations.

⇒ Initially node sampling in PRM was done using a uniform random distribution.

↳ This planner is called basic PRM.

⇒ It was observed that random sampling worked very well for a wide variety of problems and ensured the probabilistic completeness of the planner.

⇒ Today sampling schemes range from importance sampling in areas that during the course of calculations are found difficult to explore, to deterministic sampling such as quasi-random sampling and sampling on a grid.

⇒ PRM was considered as multiple-query planner.

⇒ When PRM is used to answer a single query, some modifications are made:

- The initial k good configurations are added to the roadmap nodes.
- The construction of the roadmap is done incrementally.
- It is stopped when the query at hand can be answered.

⇒ PRM may not be the fastest planner to use for single queries.

⇒ More effective single-query planners include:

- Expansive-Space Tree planner (EST)
- Rapidly-exploring Random tree (RRT)

⇒ Sampling-Based Roadmap of Trees (SRT) planner constructs a PRM-style roadmap of single-query-planner trees.

⇒ Despite their simplicity, sampling based planners are capable of dealing with robots with many degree of freedom and with many different constraints.

- Kinematic & dynamic constraints
- Closed-loop kinematics
- Stability constraints
- reconfigurable robots
- Energy constraints
- Contact constraints
- Visibility constraints.

★ Characteristics of Sampling-Based Planners

⇒ An important characteristic of the Sampling-based Planner is that they do not attempt to explicitly construct the boundaries of the configuration space obstacles or represent cells of $\mathcal{Q}_{free}$.

- They rely on a procedure that can decide whether a given configuration of the robot is in collision with the obstacles or not.

⇒ Recent advancement in collision detection algorithms have contributed heavily to the success of Sampling based planners.

⇒ Another important characteristic of sampling-based planners is that they can achieve some form of (Completeness).

- Completeness requiring that the planner always answer a path-planning query correctly, in asymptotically bounded time.
- A weaker, but still interesting form of completeness in the following:
  ↳ If a solution path exists, the planner will eventually find it.

⇒ If the Sampling of the sampling-based planner is random, then this form of completeness is called probabilistic Completeness.

⇒ If the Sampling is deterministic, including quasirandom or sampling on a grid, this form of completeness is called resolution completeness.

★ Probabilistic Roadmaps (PRM)

⇒ PRM divides planning into two phases:

1. The Learning phase
   ↳ During this roadmap in Q_{free} is built.
2. The Query phase
   ↳ During this user-defined query configurations are connected with the precomputed roadmap.

⇒ The nodes of the roadmap are configurations in Q_{free} and the edges of the roadmap corresponds to free path computed by a local planner.

⇒ The basic PRM algorithm has been shown to be probabilistically complete.

★ Basic PRM

⇒ The basic PRM algorithm first constructs a roadmap in a probabilistic way for a given workspace.

↳ The roadmap is represented by an undirected graph $G = (V, E)$.

- ↳ Nodes in V are a set of robot configurations chosen by some method over Q_{free}.
- ↳ The edge in E correspond to paths
  - ↳ An edge (q_1, q_2) corresponds to a collision free path connecting configurations q_1 & q_2.

⇒ These paths which are referred to as local path, are computed by a local planner.

↳ In its simplest form, the local planner connects two configurations by the straight line in Q_{free}.

⇒ In the query phase, the roadmap is used to solve individual path-planning problems.

⇒ Given an initial configuration q_{init} and a goal configuration q_{goal}, the method first tries to connect q_{init} & q_{goal} to two nodes q' & q'', respectively in V.

⇒ If successful, the planner then searches the graph G for a sequence of edges in E connecting q' to q''.

⇒ Finally, it transforms this sequence into a feasible path for the robot by recomputing the corresponding local paths and connecting them.

⇒ The roadmap can be reused & further augmented to capture the connectivity of Q_{free}.

↳ It is reasonable to spend a considerable amount of time in the learning phase as the roadmap will be used to solve many queries.

★ Roadmap Construction

- Let Δ be the local planner that on input $(q, q') \in Q_{free} \times Q_{free}$ returns either a collision free path from q to q' or NILL if it cannot find such path.
  - ↳ For Now let Δ is symmetric & deterministic.
- dist be a function $Q \times Q \to \mathbb{R}^+ \cup \{0\}$, called the distance function, usually a metric on Q.

Roadmap Construction Algorithm

Input

n: number of nodes to put in the roadmap
K: number of closest neighbors to examine for each configuration.

Output

A roadmap $G = (V, E)$

---

1. $V \leftarrow \emptyset$
2. $E \leftarrow \emptyset$
3. while $|V| < n$ do
4. repeat
5. $q \leftarrow$ a random configuration in Q
6. until q is collision free
7. $V \leftarrow V \cup \{q\}$
8. end while
9. for all $q \in V$ do
10. $N_q \leftarrow$ the k closest neighbors of q chosen from V according to dist.

```
11.  for all q' ∈ N_q do
12.      if (q, q') ∈ E and Δ(q, q') ≠ NIL then
13.          E ← E ∪ {(q, q')}
14.      end if
15.  end for
16. endfor
```

★ Query Phase

⇒ During the query phase, paths are found between arbitrary input configurations q_{init} and q_{goal} using the roadmap constructed in the learning phase.

Query Algorithm

Input

q_{init} : the initial configuration
q_{goal} : the goal configuration
K: the number of closest neighbors to examine for each configuration.
$G = (V, E)$: The roadmap

Output

A path from q_{init} to q_{goal} or failure.

---

1. $N_{q_{init}} \leftarrow$ the K closest neighbors of q_{init} from V according to dist.
2. $N_{q_{goal}} \leftarrow$ the K closest neighbors of q_{goal} from V according to dist.

3. $V \leftarrow \{q_{init}\} \cup \{q_{goal}\} \cup V$
4. Set q' to be the closest neighbor of q_{init} in $N_{q_{init}}$
5. repeat
6. if $\Delta(q_{init}, q') \neq NIL$ then
7. $E \leftarrow (q_{init}, q') \cup E$
8. else
9. Set q' to be the next closest neighbor of q_{init} in $N_{q_{init}}$
10. end if
11. Until a connection was succesful or the set $N_{q_{init}}$ is empty.
12. Set q' to be the closest neighbor of q_{goal} in $N_{q_{goal}}$
13. repeat
14. if $\Delta(q_{goal}, q') \neq NIL$ then
15. $E \leftarrow (q_{goal}, q') \cup E$
16. else
17. Set q' to be the next closest neighbor of q_{goal} in $N_{q_{goal}}$
18. end if
19. Until a connection was succesful or the set $N_{q_{goal}}$ is empty
20. $P \leftarrow$ Shortest path (q_{init}, q_{goal}, G)
21. if P is not empty then
22. return P
23. else
24. return failure
25. end if

⇒ If one wishes, this path may be improved by running a Smoothing post processing algorithm.

★ Adding to the Roadmap

⇒ If path-planning queries fails frequently, the roadmap may not adequately Capture the Connectivity of Q_{free}.

⇒ When this occurs, the current roadmap can be extended by resuming the Construction Step algorithm

★ Directed Roadmaps and Roadmaps that Store Local paths

⇒ So far, it has been assumed that Δ is Symmetric and deterministic.

↳ It is also possible to use a local planner Δ that is neither Symmetric nor deterministic.

⇒ If the local planner takes the robot from q to q' and the robot can also execute the path in reverse to go from q' to q the roadmap is an Undirected graph.

⇒ If local paths cannot be reversed, a directed roadmap must be constructed.

⇒ A deterministic local planner will always return the same path between two configurations. ~~in the corresponding edge.~~

- The roadmap does not have to store the local path between the two configurations in the corresponding edge.
- The path can be recomputed if needed to answer a query.

⇒ If a nondeterministic local planner is used, the roadmap will have to associate with edge the local path computed by Δ.

★ A Practical Implementation of Basic PRM

⇒ One of the advantages of the basic PRM algorithm presented ~~[illegible]~~ is that it is easy to implement & performs well for a variety of problems.

★ Sampling Strategy: Uniform Distribution

⇒ To obtain a configuration, each translational dof can be drawn from the interval of allowed values of the corresponding dof using the uniform probability distribution over this interval.

- The same principle applies to rotational dof but care should be taken not to favor specific orientation because of the representation used.

⇒ The sampled configuration is checked for collision.

↳ Can be done using a variety of existing general techniques.

★ Connection Strategy: Selecting Closest Neighbors

⇒ Another important choice to be made is that of selecting the set N_q of closest neighbors to a configuration q.

⇒ Many data structure has been proposed to efficiently calculate the closest neighbors to a point in a d-dimensional space.

↳ A relatively efficient method both in terms of space & time is the Kd-tree
{K-Dimensional tree}

⇒ A d-dimentional Kd-tree uses as input a set S of n points in d-dimensions and construct a binary tree.

↳ That decomposes space into cells such that no cell contains too many points.

⇒ A Kd-tree is build recursively by splitting S by a plane into two subsets of roughly equal size:

↳ S_1 which includes points of S that lie to the left of the plane.

↳ S_{gr}, which includes the remaining points of S.

⇒ The plane is stored at the root, and the left and right child are recursively constructed with input S_l & S_{gr} respectively.

⇒ A Kd-tree for a set of n points in n dimensions uses $O(dn)$ storage and can be built in $O(dn\log n)$ time.

⇒ A rectangular range query takes $O(n^{1-\frac{1}{d}}+m)$ time, where m is the number of reported neighbors.

⇒ The rectangular range query time can be reduced considerably by introducing a small approximation error.

↳ Modified approach is called Approximate Nearest Neighbor queries (ANN) and is becoming increasingly popular.

★ Distance function and Embeddings

⇒ Function dist is used to resolve the K closest neighbors query.

↳ dist(q', q'') reflects the likelihood that the local planner will fail to compute a collision free path between these configuration.

⇒ One possibility is to define dist(q', q'') as some measure of the workspace region swept by the robot.

- Such as the area or the volume, when it moves in the absence of obstacles along the path Δ(q', q'').
- Minimizing the swept volume, will minimize the chance of collision with the obstacles.

⇒ An exact computation of swept area or volume is notoriously difficult, which is way heuristic metrics generally attempts to approximate the swept-volume metric.

⇒ An approximate and inexpensive measure of the swept-region can be constructed as follows.

- The robot configuration q' and q'' can be mapped to points in a Euclidian space, emb(q') and emb(q'') respectively.

So $dist(q', q'') = \| emb(q') - emb(q'') \|$

⇒ A practical choice for the embedding function is to select $p > 0$ points on the robot, concatenate them, and create vector whose dimension is p multiplied by the dimension of the workspace of the robot.

⇒ The embedding does not take into account obstacles.

↳ So even when two configurations are close to one another, connecting them may be impossible due to obstacles.

⇒ For the case of rigid body motion, an alternative solution is to split dist into two components, one that expresses the distance between two configurations due to translation and one due to orientation.

↳ Example, if X and R represent the translation and rotation components of the configuration $q = (X, R) \in SE(3)$ then,

$$\boxed{dist(q', q'') = w_t \| X' - X'' \| + w_r f(R', R'')}$$

⇒ The rotation distance is scaled relative to the translation distance via the weight w_t & w_r.

⇒ One of the difficulties with this method is deciding proper weight values.

↳ Furthermore, the extension to articulated body is not straightforward.

★ Local Planner

⇒ An important design decision is related to how fast the local planner should be.

⇒ There is clearly a tradeoff between the time spent in each individual call of this planner and the number of calls.

⇒ One popular planner, applicable to all holonomic robots, connects any two given configurations by a straight-line segment in Q and checks this line segment for collision.

⇒ There are two commonly-used choices for collision checking:

- Incremental collision-checking algorithm.
- Subdivision collision-checking algorithm.

⇒ In both the cases:

- Path generated by the local planner between configurations q' and q'', is discretized into a number of configurations $(q_1, \ldots, q_k)$ where $q' = q_1$ & $q'' = q_k$.
- The distance between any two consecutive configurations q_i and q_{i+1} is less than some small constant step-size.

↳ Again sampling is used to determine if a local path is collision free.

⇒ In case of incremental collision checking

- The robot is positioned at q' and moved at each step by step-size along the straight line in Q between q' and q''.
- A collision check is performed at the end of each step.
- The algorithm terminates as soon as a collision is detected or when q'' is reached.

⇒ In the case of the subdivision collision checking:

- The middle point q_m of the straight line in Q between q' and q'' is first checked for collision.
- Then the algorithm proceeds on the straight lines between (q', q_m) & (q_m, q'').
- The procession halts when a collision is found or the length of the line segment is less than step-size.

★ Post processing Queries

⇒ A postprocessing step may be applied to the path connecting q_{init} to q_{goal} to improve its quality according to some criteria.

{Example Shortness, Smoothness etc...}

⇒ From a given path, a shorter path could be obtained by checking whether non-adjacent configurations q_1 & q_2 along the path ~~could be obtained~~ can be connected with the local planner.

⇒ Another alternative would be a greedy approach:

- Start from q_{init} & try to connect directly to the target q_{goal}.
- If this step fails, start from the configuration after q_{init} & try again.
- Repeate until a connection can be made to q_{goal}, say from the point q_0.
- Now set the target to q_0 and begin again.
- This procedure can also be applied toward the opposit direction.

⇒ Instead of shortening the path, a different object may be to get a path with smooth curvature.

↳ A possible approach to this is to use interpolating curves, such as splines & use the configurations that have been computed by PRM as the interpolation points for the curves.

↳ In this case, collision checking is performed along the curves until curves that satisfy both the smoothness properties and the collision avoidance criteria are found.

⇒ Post processing steps can improve the quality of the path, but can also impose a significant overhead on the time that it takes to report the results of a query.

★ PRM Sampling Strategies

⇒ Several node-sampling strategies have been developed over the years of PRM.

⇒ Increasing the density of sampling in some areas of the free space is referred to as Importance Sampling.

⇒ The uniform random sampling used in early work in PRM is the easiest sampling scheme to implement.

⇒ There exists cases where uniform random sampling has poor performance.
  ↳ Often this is the result of the so-called narrow passage Problem.

⇒ A number of different sampling methods have been designed with the narrow passage problem in mind.
  ↳ The narrow passage problem still remains a challenge for PRM planners and is an active area of research.

★ Sampling near the Obstacles

⇒ The motivation behind this kind of sampling is that narrow passage can be considered as thin corridors in Cfree surrounded by Obstacles.

⇒ OBPRM is one of the first and very successful representatives of obstacle-based sampling methods.

→ Initially generate many configurations at random from a uniform distribution.

→ For each configuration q_{in} found in collision, it generates a random direction V, and the planner finds a free configuration q_{out} in the direction V.

→ Finally, it performs a simple binary search to find the closest free configuration q to the surface of the obstacle

→ Configuration q is added to the roadmap, while q_{in} & q_{out} are discarded.

⇒ The Gaussian Sampler addresses the narrow passage problem by sampling from a gaussian distribution that is biased near the obstacles.

→ It is obtained by first generating a configuration q_1 randomly from uniform distribution.

→ Then a distance step is chosen according to a normal distribution to generate a configuration q_2 at random at distance step from q_1

→ Both configurations are discarded if both are in collision or if both are collision free.

↳ A sample is added to the roadmap if it is collision-free & the other sample is in collision.

⇒ Samples generated in a dilated Q_{free}:

↳ The dilation of Q_{free} widens narrow passages, making it easier for the planner to capture the connectivity of the space.

↳ During a second stage, all samples that do not lie in Q_{free} are pushed into Q_{free} by performing local resampling operations.

★ Sampling Inside Narrow Passages

⇒ The bridge planner uses a bridge test to sample configurations inside a narrow passages.

↳ In a bridge test, two configurations q' & q'' are sampled randomly from a uniform distribution in Q.

↳ These configurations are considered for addition to the roadmap

↳ If they are both in collision, then the point q_m halfway between them is added to the roadmap if it is collision free.

↳ This is called a bridge test because the line segment between q' & q'' resembles a bridge with q' & q'' inside obstacles
↳ acting as piers and the midpoint q_m hovering over Q_{free}.

⇒ Geometry of narrow passage makes the construction of short bridges easy, while in open space construction of short bridge is difficult.
↳ This allows the bridge planner to sample points inside narrow passage by favoring the construction of short bridges.

⇒ An efficient solution to the narrow passage problem would generate samples that are inside narrow passage but as far away as possible from the obstacles.
→ The GVD has exactly this property.
→ Although exact computation of the GVD is impractical for high-dimensional configuration space
→ But it is possible to find samples on the GVD without computing it explicitly.

→ This can be done by a retraction scheme.
→ Retraction is achieved by a bisection method that moves each sample configuration until it is equidistant from two points on the boundary of Q_{free}.

⇒ A simpler approach is to compute the GVD of the workspace and generate samples that somehow conforms to this GVD.

→ The disadvantage of workspace-GVD sampling is that it is in general difficult to generate configurations of the robot close to the GVD.

→ Advantage: It captures narrow passages well.

★ Visibility based Sampling

⇒ The goal of the Visibility based PRM is to produce visibility roadmaps with a small number of nodes by structuring the configuration space into visibility domain.

⇒ The visibility domain of a configuration q includes all configurations that can be connected to q by the local planner.

⇒ This planner adds a configuration q to the roadmap only it satisfy one of the two criteria:

① → q cannot be connected to any existing node.
(i.e. q is a new component)

② → q connects at least two existing components.

★ Manipulability based Sampling

⇒ It is an importance-sampling approach that exploits the manipulability measures associated with the manipulator Jacobian.

⇒ The motivation of using manipulability as a bias for sampling is as follows:

→ In regions of the configuration space where manipulability is high, the robot has great dexterity, and therefore relatively fewer samples should be required in these areas.

→ Regions of the configuration space where manipulability is low tend to be near singular configurations of the arm.

↳ Near singularities, the range of possible motion is reduced, and therefore such regions should be sampled more densely.

⇒ Let $J(q)$ denote the manipulator Jacobian matrix.

⇒ For a redundant arm the manipulability in configuration q is given by

$$\omega(q) = \sqrt{\det J(q) J^T(q)}$$

⇒ For bias sampling, the CDF of ω is created.

⇒ Samples are then drawn from ~~Ex~~ Uniform density on the configuration space, and rejected with probability proportional to the associated CDF value of their manipulability value.

★ Quasirandom Sampling

⇒ A number of deterministic (sometimes called quasirandom) alternatives to random sampling have been used.

↳ These alternatives were first introduced in the context of Monte Carlo integration and aim to optimize various properties of the distribution of the samples.

⇒ Let P be a set of points samples on some space X, and N be the number of points in P.

⇒ One way to evaluate the quality of the samples in P is to assess how uniformly the points in P cover X.

-> This is done with respect to a specific collection of subsets of X, called a range space denoted by R.

⇒ Let R be the set of all axis-aligned rectangular subsets of X, and defined μ to be the measure (or volume) of a set.

⇒ Since P contains N points, the difference between the relative volumes of R to X and the fraction of samples contained in $R \in \mathcal{R}$ is given by

$$\left| \frac{\mu(R)}{\mu(X)} - \frac{|P \cap R|}{N} \right|$$

⇒ If we take the supremum of this difference over all $R \in \mathcal{R}$ we obtain the concept of discrepancy.

Definition: The discrepancy of point set P with respect to a range space R over some space X is defined as

$$D(P, \mathcal{R}) = \sup_{R \in \mathcal{R}} \left| \frac{\mu(R)}{\mu(X)} - \frac{|P \cap R|}{N} \right|$$

⇒ It is not necessary to take R as the subset of axis-aligned rectangles, but this choice gives an intuitive understanding of discrepancy.

⇒ Another common choice is to take R as the set of d-balls.

↳ $\forall R \in \mathcal{R}$, $R = \{x' \mid \|x - x'\| < \epsilon\}$

⇒ While discrepancy provides a measure of how uniformly points are distributed over the space X, dispersion provides a measure of the largest portion of X that contains no points in P.

⇒ For a given metric f, the distance between a point $x \in X$ and a point $p \in P$ is given by $f(x, p)$

⇒ Then $\min_{p \in P} f(x, p)$ gives the distance from x to the nearest point in P

⇒ If we take f to be the Euclidean metric, this gives the largest empty ball centered on x.

⇒ If we then take the minimization over all points in X, we obtain the size of the largest empty ball in X.

↳ This is exactly the concept of dispersion.

Defination: The dispersion δ of point set P with respect to the metric f is given by

$$\delta(P, f) = \sup_{x \in X} \min_{p \in P} f(x, p)$$

⇒ An important result due to Sukharev gives a bound on the number of samples required to achieve a given dispersion.

→ Sukharev sampling criterion states that when f is taken as L_∞ norm, a set P of N samples on the d-dimensional unit cube will have

$$\delta(P, f) \geq \frac{1}{2\lfloor N^{\frac{1}{d}} \rfloor}$$

→ So to achieve a given dispersion value, say δ^*, since N must be an integer, we have

$$\delta^* \geq \frac{1}{2\lfloor N^{1/d} \rfloor} \Rightarrow \boxed{N \geq \left(\frac{1}{2\delta^*}\right)^d}$$

⇒ Now that we have quantitative measures of the quality of a set of samples, we describe some common ways to generate samples.

⇒ For the case of $X = [0, 1]$ the van der Corput sequence gives a set of samples that minimizes both dispersion and discrepancy.

⇒ The n^{th} sample in the sequence is generated as follows:

→ Let $a_i \in \{0, 1\}$ be the coefficients that define the binary representation of n

$$n = \sum_i a_i 2^i = a_0 + a_1 2 + a_2 2^2 + \cdots$$

→ The n^{th} element of the Van der Corput Sequence $\phi(n)$ is defined as :-

$$\phi(n) = \sum_i a_i 2^{-(i+1)} = \frac{a_0}{2} + \frac{a_1}{2^2} + \cdots$$

⇒ The Van der Corput sequence can only be used to sample the real line.

⇒ The Halton Sequence generalizes the Van der Corput sequence to d dimension.

→ Let $\{b_i\}$ define a set of d relatively Prime Integer.

{eg $b_1 = 2$ $b_2 = 3$, $b_3 = 5$ - - - -}

→ The Integer n has a representation in base b_j given by

$$n = \sum_i a_{ij} b_j^i, \quad a_{ij} \in \{0, 1, \cdots b_j - 1\}$$

↳ and $\Phi_{b_j}(n)$ is defined as

$$\Phi_{b_j}(n) = \sum a_{i,j}\, b_j^{-(i+1)}$$

↳ The n^{th} sample is then defined by the coordinates $P_n = (\Phi_{b_1}(n), \Phi_{b_2}(n), \cdots \Phi_{b_d}(n))$

⇒ When the range space R is a set of axis-aligned rectangular subsets of X, the discrepancy for the Halton sequence is bounded by

$$\boxed{D(P,R) < O\left(\frac{\log^d N}{N}\right)}$$

⇒ When the range space R is the set of d-balls, the discrepancy is bounded by

$$\boxed{D(P,R) < O\left(N^{-\frac{(d+1)}{2}}\right)}$$

⇒ When N is specified, a Hammersley Sequence achieves the best possible asymptotic discrepancy.

⇒ The n^{th} point in a Hammersley sequence is obtained by using the first d-1 coordinates of a point in the Halton Sequence, with the ratio n/N as the first coordinate.

$$\boxed{P_n = \left(\frac{n}{N}, \Phi_{b_1}(n), \Phi_{b_2}(n), \cdots \Phi_{b_{d-1}}(n)\right) \quad n = 0, \cdots N-1}$$

⇒ The use of quasi-random sequence has the advantage that the running time is guaranteed to be the same for all the runs due to the deterministic nature of the point generation process.

↳ The resulting planner is resolution complete.

⇒ As with any deterministic sampling method however, it is possible to construct examples where performance of the planner deteriorates.

↳ As a remedy, it has been suggested to perturb the sequence.

★ Grid-Based Sampling

⇒ The nodes of a grid can be an effective sampling strategy in the PRM setting.

↳ Especially when combined with efficient ~~sampling strategy in~~ node connection scheme, they can result in powerful planners for problems arising in industrial setting.

⇒ A grid-based path-planning algorithm is resolution complete.

★ Connection Sampling

⇒ Connection Sampling generates samples that facilitate the Connection of the roadmap & can be combined with all previously described Sampling methods.

⇒ Typically, if a small number of configurations is initially generated, there may exist a few disconnected components at the end of the Construction step.

⇒ If the roadmap under construction is disconnected in a place where Q_{free} is not, this place may corresponds to some difficult area of Q_{free}.

{ Possibly to a narrow passage of Q_{free} }

⇒ The idea underlying Connection Sampling is to select a number of configurations from the roadmap that are likely to lie in such regions and expand them.

⇒ The expansion of a configuration q involves selecting a new free configuration in the neighborhood of q as described below:

- adding this configuration to the roadmap.
- Trying to connect it to other configurations of the roadmap in the same manner as in the Construction Step.

⇒ The Connection Sampling step increases the density of the roadmap in regions of Q_{free} that are believed to be difficult.

⇒ Connection sampling never creates new components in the roadmap.

↳ At worst, it fails to reduce the number of components.

⇒ Each configuration q is associated with a heuristic measure of the difficulty of the region around q expressed by a positive weight $w(q)$.

↳ $w(q)$ is large whenever q is considered to be in a difficult region.

⇒ Weights are normalized so that their sum for all configurations in the roadmap is one.

⇒ Then, repeatedly, a configuration q is selected from the roadmap with probability $w(q)$.

⇒ Then q is expanded.

⇒ The weights can be computed only once at the beginning of the proces and not not modified when new configurations are added to the roadmap, or can be modified Periodically.

⇒ There are several ways to define the heuristic weight $w(q)$.

⇒ One function that works in practice is:

⇒ Let $deg(q)$ be the number of configurations to which q is connected.

$$w(q) = \frac{\frac{1}{deg(q)+1}}{\sum_{q' \in V} \frac{1}{deg(q')+1}}$$

★ Choosing among different Sampling Strategies.

⇒ Choosing among different sampling strategies is an open issue.

★ PRM Connection Strategies

⇒ An important aspect of PRM is to Selection of pairs of Configurations that will be tried for connections by a local planner.

↳ One possible choice is to use the local Planner to connect every configuration to all of its K closest neighbors.

⊕ Creating Sparse Roadmaps

⇒ A method that can speed up the roadmap construction step is to avoid the computation of edges that are part of the same connected component.

↳ Since addition of the new edge will not improve the connectivity of the roadmap.

⇒ The simplest is to connect a configuration with the nearest node in each component that lies close enough.

⇒ With the above method, no cycles can be created and the resulting graph is a forest (i.e. collection of trees)

⇒ In some cases, the absence of cycles may lead the query phase to construct unnecessarily long paths.

⇒ This drawback can be mitigated by applying post processing techniques, such as smoothing, on the resulting path.

⇒ Recent work shows how to add useful cycles in PRM that results in higher quality shorter paths.

⑤ Connecting Connected Components

⇒ The roadmap may consist of several connected components.

⇒ The quality of the roadmap can be improved by employing strategies aimed at connecting different components of the roadmap.

⑥ Lazy Evaluation

⇒ The idea behind Lazy evaluation is to speed up performance by doing collision checks only when it is absolutely necessary.

⇒ It can be applied to almost all the sampling-based planners presented.

⇒ When lazy evaluation is employed, PRM operates on a roadmap G, whose nodes and paths have not been fully evaluated.

⇒ It is assumed that all nodes and all edges of a node to its k neighbors are free of collisions.

⇒ Once PRM is presented with a query, it connects q_{init} & q_{goal} to two close nodes of G.

⇒ The planner then performs a graph search to find the shortest path between q_{init} & q_{goal}.

⇒ Then the path is checked as follows:

- → First, the nodes of G on the path are checked for collision.
- → If a node is found in collision, it is removed from G together with all edges originating from it.
- → This procedure is repeated until a path with free nodes is discovered.

⇒ The edges of the path are then checked.

→ In order to avoide unnecessary collision checks, all edges along the path are first checked at a coarse resolution.

→ Then at each iteration the resolution become finer & finer until it reaches the desired discretization.

→ If an edge is found in collision, it is removed from G.

⇒ The process of finding paths, checking nodes and then checking their edges is repeated until a free path is found or all nodes of G have been visited.

⇒ Once it is decided that a node of G is in Q_{free}, this information is recorded to avoide future checks.

→ For the edges, the resolution at which they have been checked for collision is also recorded so that if an edge is part of a future plan, collision checks are not replicated.

⇒ If no path is found and the nodes of G have been exhausted, new nodes and edges can be added to G.

★ Single-Query Sampling-Based Planners

⇒ Single-query planners attempt to solve a query as fast as possible and do not focus on the exploration of the entire Q_{free}.

⇒ One of the first widely used Sampling-based single query planners was RPP. (Randomized Path Planner)

⇒ RPP works by constructing potential fields over the workspace and attract control Points of the robot to their corresponding Positions in the goal Configuration while pushing these robot points away from the obstacles.

⇒ The workspace potentials are combined using an arbitration function to generate a Configuration space potential.

⇒ Starting from the initial Configuration RPP performs a gradient motion until it reaches a local minimum.

⇒ If the goal configuration has not been reached, RPP executes a series of random motions to escape the local minimum.

⇒ On this way, RPP incrementally builds a graph of local minima, where the path joining two local minima is obtained by concatenating a random motion & a gradient descent motion.

⇒ PRM itself can also be used as single-query planner.
- ↳ On this case q_{init} & q_{goal} should be inserted into the roadmap at the beginning.
- ↳ Careful application of lazy evaluation has yielded an effective single-query PRM planner, which is called **Lazy PRM**.

⇒ This section describes two planners that were designed primarily for single-query planning.
- ↳ **Expansive-Spaces Trees (ESTs)**
- ↳ **Rapidly-exploring Random Trees (RRTs)**

⇒ These planners also have the advantage that they are very efficient for kinodynamic planning.

⇒ ESTs and RRTs bias the sampling of configurations by maintaining two trees.
- ↳ T_{init} rooted at q_{init} configurations.
- ↳ T_{goal} rooted at q_{goal} configurations.

⇒ and then growing the trees toward each other until they are merged into one.

⇒ It is possible to construct only a single tree rooted at q_{init} that grows toward q_{goal} but, for geometric path-planning, this is usually less efficient than maintaining two trees.

⇒ On the Construction Step:
- → new configurations are sampled from Q_{free} near the boundaries of the two tree.
- → A configuration is added to a tree only if it can be connected by the local planner to some existing configuration in the tree.

⇒ On the merging step, a local planner attempts to connect pairs of configurations selected from both trees.

⇒ If successful, the two tree become one connected component and a path from q_{init} to q_{goal} is returned.

⇒ Although it is important to search the part of Q_{free} that is relevant to the given query, the planners need to demonstrate that their sampling can potentially cover the whole Q_{free}.
↳ To ensure Probabilistic Completeness.

⇒ SBL is a bi-directional EST that uses lazy evaluation for its node connection strategy.

⇒ { Single-query Bi-directional planner with Lazy Collision checking }

★ Expansive-Spaces Trees

⇒ ESTs were initially developed as an efficient single-query planner that covers the space between q_{init} and q_{goal} rapidly.

⇒ EST was initially geared toward Kinodynamic problem and for these problems a single tree is typically build.

⇒ The EST algorithm has been shown to be Probabilistically complete.

⊛ Construction of Trees

⇒ Let T be one of the trees T_{init} or T_{goal} rooted at q_{init} & q_{goal} respectively.

⇒ The planner first selects a Configuration q in T from which to grow T.

⇒ and then samples a random Configuration q_{rand} from a uniform distribution in the neighborhood of q.

⇒ Configuration q_{rand} is selected at random with probability $\pi_T(q)$.

⇒ The local planner Δ attempts a connection between q & q_{rand}.

⇒ If successful, q_{rand} is added to the vertices of T and (q, q_{rand}) is added to the edges of T.

⇒ The process is repeated until a specified number of configurations has been added to T.

Algorithm: Build EST Algorithm

Input:

q_0: the configuration where the tree is rooted.

n: the number of attempts to expand the tree.

Output:

A tree $T = (V, E)$ that is rooted at q_0 and has $\leq n$ configurations.

---

1. $V \leftarrow \{q_0\}$
2. $E \leftarrow \phi$
3. for $i = 1$ to n do
4. $q \leftarrow$ a randomly chosen configuration from T with probability $\pi_T(q)$

5. extend EST (T, q_r)
6. end for
7. return T

Algorithm: Extend EST Algorithm

Input:

T = (V, E): an EST

q_r: a configuration from which to grow T

Output:

A new configuration q_{new} in the neighborhood of q_r, or NIL in case of failure.

---

1. $q_{new} \leftarrow$ a random collision-free configuration from the neighborhood of q_r
2. if $\Delta(q_r, q_{new})$ then
3. $\quad V \leftarrow V \cup \{q_{new}\}$
4. $\quad E \leftarrow E \cup \{(q_r, q_{new})\}$
5. $\quad$ return q_{new}
6. end if
7. return NIL

---

(11) Guiding the Sampling

⇒ The effectiveness of EST relies on the ability to avoid oversampling any region of Q_{free} especially the neighborhoods of q_{init} & q_{goal}.

⇒ Hence, careful consideration is given to the choice of the probability density function π_T.

- → Ideally, π_T should be such that it constitute a rather uniform covering of the connected components of Q_{free} containing q_{init} & q_{goal}.
- → A good choice of π is biased toward configurations of T whose neighborhoods are not dense.

⇒ There are several ways to measure density of a neighborhood. One that works well in practice is:

- → with Each configuration q_i of T, a weight $w_T(q_i)$, that counts the number of configurations within some predefined neighborhood of q_i.
- → $\pi_T(q_i)$ is defined inversely proportional to $w_T(q_i)$.

⇒ The naive method to compute $\pi_T(q)$ enumerates all the configurations of T and tests if they are close to q.

↳ This works only for relatively small n.

⇒ A reasonable approximation to $\pi_T(q)$ can be obtained by ~~imposing~~ imposing a grid on Q.

↳ At each iteration, the planner selects the configuration from which to grow the tree by choosing at random a cell & a configuration from this cell.

⇒ Several other π_T functions have been proposed.

⇒ $\pi_T(q)$ is defined to be a function of:

↳ Order in which q is generated

↳ its number of neighbors

↳ its out degree

↳ an A* cost function A^*_{cost}.

↳ Sum of the total cost from the root of the tree to q and the estimated cost from q to the goal configuration.

⇒ This weight functions combines in a natural way standard EST heuristics with potential field methods.

(#) Merging of Trees

⇒ The merging of the trees is achieved by pushing the exploration of the space from one tree toward the space explored by the other tree.

⇒ Initially, a configuration in Tinit is moved as described, to produce a new configuration q_s.

⇒ Then the local planner attempts to connect q_s to its closest K configurations in Tgoal

⇒ If a connection is successful, the two trees are merged.

↳ Otherwise the trees are swapped & the process is repeated for a specified number of times.

★ Rapidly-Exploring Random Trees (RRT)

→ Single-query planning algorithm

→ Initially developed for Kinodynamic motion planning.

→ It is probabilistically complete

⊛ Construction of Trees

⇒ Let T be one of the trees T_{init} or T_{goal} rooted at q_{init} & q_{goal}, respectively.

⇒ Each tree T is incrementally extended.

⇒ At each iteration, a random configuration q_{rand} is sampled uniformly in Q_{free}.

⇒ The nearest configuration, q_{near}, to q_{rand} in T is found and an attempt is made to make progress from q_{near} toward q_{rand}.

⇒ The newly generated configuration, q_{new}, if it is collision-free, is added to the vertices of T, and the edge (q_{near}, q_{new}) is added to the edge of T.

Algorithm: Build RRT Algorithm

Input:

q_0 : the configuration where the tree is rooted
n : the number of attempts to expand the tree

Output:

A tree $T=(V,E)$ that is rooted at q_0 and has $\leq n$ configurations.

---

1. $V \leftarrow \{q_0\}$
2. $E \leftarrow \emptyset$
3. for $i=1$ to n do
4. $\quad q_{rand} \leftarrow$ a randomly chosen free configuration
5. $\quad$ extend RRT(T, q_{rand})
6. end for
7. return T

Algorithm: Extend RRT Algorithm

Input:

$T=(V,E)$: an RRT
q : a configuration toward which the tree T is grown

Output:

A new configuration q_{new} toward q, or Nil in case of failure

---

1. $q_{near} \leftarrow$ Closest neighbor of q in T
2. $q_{new} \leftarrow$ Progress q_{near} by step_size along the straight line in Q between q_{near} and q_{rand}.

3. if q_{new} is collision-free then
4. $V \leftarrow V \cup \{q_{new}\}$
5. $E \leftarrow E \cup \{(q_{near}, q_{new})\}$
6. return q_{new}
7. end if
8. return NIL

⇒ A natural question to consider is how step-size is determined.

⇒ One possible way is to choose step-size dynamically based on the distance between q_{near} & q_{rand}.

⇒ It makes sense to chose a large value of step-size if the two configurations are far from colliding & small otherwise.

⇒ RRT is sensitive to the distance function used, since it is this function that determines step-size and guides the sampling.

⇒ It is also interesting to consider a greedier alternative that tries to move q_{new} as close to q_{rand} as possible.

Algorithm: Connect RRT Algorithm

Input:

$T = (V, E)$: an RRT

q: a configuration toward which the tree T is grown

Output:

Connected if q is connected to T; failure otherwise

---

1. repeat
2. $q_{new} \leftarrow$ extend RRT (T, q)
3. until $(q_{new} = q$ or $q_{new} = NIL)$
4. if $q_{new} = q$ then
5. return connected
6. else
7. return failure
8. end if

(#) Guiding the Sampling

⇒ Selecting a node uniformly at random is a basic mechanism of RRT.

⇒ It is also of interest to consider other sampling functions that are biased toward the connected components of Q that contain q_{init} or q_{goal}.

⇒ Experimental evidence has shown that Setting q_{rand} to q_{goal} with probability p, or randomly generating q_{rand} with probability $1-p$ from a uniform distribution works well.

⇒ This simple function can be further improved by sampling in a region around q_{goal} instead of setting q_{rand} to q_{goal}.

Merging of Trees

⇒ On the merging step, RRT tries to connect the two trees, T_{init} and T_{goal}, rooted at q_{init} and q_{goal}, respectively.

Algorithm: Merge RRT Algorithm

Input:

T_1: first RRT

T_2: Second RRT

k: number of attempts allowed to merge T_1 & T_2

Output:

merged if the two RRTs are connected to each other; failure otherwise.

---

1. for $i=1$ to k do
2. $q_{rand} \leftarrow$ a randomly chosen free configuration
3. $q_{new,1} \leftarrow$ extend RRT (T_1, q_{rand})
4. if $q_{new,1} \neq$ NIL then
5. $q_{new,2} \leftarrow$ extend RRT $(T_2, q_{new,1})$
6. if $q_{new,1} = q_{new,2}$ then

```
7.          return merged
8.        end if
9.        SWAP(T1, T2)
10.     end if
11.   end for
12.   return failure
```

⇒ This greedier approach has been reported to work well.

⇒ It is also reasonable to replace only one of 'extend RRT' call, and thus obtain a balance between the two approaches.

⇒ The implementation of RRT is easier than that of EST.

★ Connection Strategies and the SBL planner

⇒ Lazy evaluation can also be used in the context of tree-building sampling methods.

⇒ A combination of lazy evaluation & ESTs has been presented in the context of the Single-query Bi-directional, Lazy collision-checking (SBL) planner.

⇒ SBL constructs two EST trees rooted at qinit and qgoal.

⇒ SBL Creates new Samples according to the EST Criteria but does not immediately test Connections between Samples for collisions.

⇒ A Connection between two configurations is checked exactly once & this is done only when the Connection is part of the path joining the two trees together.

⇒ This results in substantial time Savings.

* Integration of Planners : Sampling-Based Roadmap of Trees

⇒ This shows how to effectively combine a sampling-based method primarily designed for multiple-query planning (PRM) with sampling-based tree methods primarily designed for single-query planning (EST, RRT etc)

⇒ The sampling based Roadmap of Trees (SRT) Planner takes advantage of the local sampling schemes of tree planners to populate a PRM like roadmap.

⇒ SRT replaces the local planner of PRM with a single-query sampling-based tree planner.

⇒ As in the PRM formulation, SRT constructs a roadmap aiming at capturing the connectivity of Q_{free}.

⇒ The nodes of the roadmap are not single configuration but trees.

⇒ Connections between trees are computed by a bi-directional tree algorithms such as EST or RRT.

⇒ SRT constructs a roadmap of trees.

⇒ The undirected graph $G_T = (V_T, E_T)$ is induced subgraph of G which is defined by partitioning G into a set of subgraphs $T_1, \dots T_m$ which are trees and contracting them into the vertices of G_T.

$$V_T = \{T_1, \dots T_m\}$$

$(T_i, T_j) \in E_T$ if $\exists$ $q_i \in T_i$ & $q_j \in T_j$ such that q_i & q_j have been connected by a local path.

Adding Trees to the Roadmap

① Sample points uniformly at random in Q_{free}.

② Grow the trees using a sampling-based tree planner.

⊛ Adding Edges to the Roadmap

⇒ For each tree T_i, a set N_{T_i} consisting of closest k random tree neighbors is computed and a connection is attempted between T_i & each tree T_j in N_{T_i}.

⇒ In order to determine the closest neighbors, each tree T_i defines a representative configuration q_{T_i} which is computed as an aggregate of the configurations in T_i.

↳ distance between two tree $\overset{T_i \& T_j}{\uparrow}$ $dist(q_{T_i}, q_{T_j})$.

⇒ Computation of candidate edges is typically carried out by a sampling-based tree planner.

⇒ First, for each ~~config~~ candidate edge (T_i, T_j), a number of close pairs of configurations of T_i & T_j are quickly checked with a fast deterministic local planner.

⇒ If the connection is successful, the edge (T_i, T_j) is added to E_T

↳ And the graph components to which T_i and T_j belongs are merged into one.

Algorithm: Connect SPT Algorithm

Input:

V_T: a set of trees

K: number of closest neighbors to examine for each tree.

M: number of random neighbors to examine for each tree.

Output:

A roadmap $G_T = (V_T, E_T)$ of trees.

---

1. $E_T \leftarrow \phi$
2. for all $T_i \in V_T$ do
3. $N_{T_i} \leftarrow$ K nearest and M random neighbors of T_i in V_T
4. for all $T_j \in N_{T_i}$ do
5. If T_i & T_j are not in the same connected component of G_T then
6. merged $\leftarrow$ FALSE
7. $S_i \leftarrow$ a set of random closer configurations from T_i
8. for all $q_i \in S_i$ & merged = FALSE do
9. $q_j \leftarrow$ closest configuration in T_j to q_i
10. If $\Delta(q_i, q_j)$ then
11. $E_T \leftarrow E_T \cup \{(T_i, T_j)\}$
12. merged $\leftarrow$ True

```
            end if
        end for
        if mergd = FALSE & Merge Trees(T_i, T_j) then
            E_T ← E_T ∪ {(T_i, T_j)}
        end if
    end if
end for
end for
```

Answering Queries

⇒ Given q_{init} & q_{goal}, the trees T_{init} & T_{goal} rooted at q_{init} & q_{goal} respectively, are grown for a small number of iterations and added to the roadmap.

⇒ Neighbors of T_{init} & T_{goal} are computed as a union of the k closest & or random trees.

⇒ The tree-connection algorithm alternates between attempts to connect T_{init} & T_{goal} to each of their respective neighbor trees.

★ SRT is significantly more decoupled than tree planners such as ESTs and RRTs.

- → This decoupling allows for an efficient parallelization of SRT.
- → Can be used to solve very high dimensional problems.

★ Beyond Basic Path Planning

⊛ Control-Based Planning

⇒ Consider a nonholonomic robot

$U \rightarrow$ Set of controls

$f \rightarrow$ well-behaved control function

$f: Q \times U \rightarrow Q$

{describes a method for propagating a robot state into the future.}

⇒ Many of the Sampling-based planners can be used with such system.

↳ When f is given a simple way of generating new samples in state space may be available.

⇒ When such a forward propagation of the system is relatively inexpensive, tree-based planners such as ESTs and RRTs can be directly applied.

⊛ Multiple Robots

⇒ A collision free path from an initial configuration of the robot to a goal configuration of the robot implies that at every step there is no collision between a robot & an obstacle or between a robot and another robot.

⇒ There are two classic approaches to the multiple robots problem:

- Centralized planning
- Decoupled planning.

⊛ Centralized planning

⇒ Centralized planning considers the different robot as if they were forming a single multibody robot and represents Q as the Cartesian product of the configuration space of all the robots.

⇒ The difficulty of centralized planning arises from high dimensionality of Q.

⊛ Decoupled Planning

⇒ Decoupled planning works in two stages:

① Initially, collision-free paths are computed for each robot individually, not taking into account the other robots.

② Coordination is achieved by computing the relative velocities of the robots along their individual paths that will avoid collision among them.

⇒ Decoupled planning is incomplete however.

↳ It may be impossible to coordinate some of the paths generated during the first stage.
