EN.530.678: Nonlinear Control and Planning in Robotics

Lecture# 12
Sampling-based Motion Planning

April 20, 2015

The trajectory planning problem

- design a reference trajectory $x(t) \in \mathbb{R}^n$ and control inputs $u(t) \in \mathbb{R}^m$ by solving the constrained optimal control problem:

$$\min_{x(\cdot),u(\cdot),t_f} J = \phi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt, \quad (1)$$

subject to:

$$x(t_0) = x_0, \quad x(t_f) \text{ and } t_f \text{ free} \quad (2)$$

$$\dot{x}(t) = f(x(t), u(t), t) \quad (3)$$

$$c(x(t), u(t), t) \leq 0, \text{ for all } t \in [t_0, t_f] \quad (4)$$

$$\psi(x(t_f), t_f) \leq 0, \quad (5)$$

where

- $[t_0, t_f]$ - time horizon, x_0-initial state
- L - trajectory cost: e.g. control effort, energy, time, distance
- ϕ - terminal cost: e.g. reaching a desired region
- c - trajectory constraints: e.g. control bounds, forbidden regions in X
- ψ - terminal constraint defines algebraically a goal region

Issues and Challenges

Generally, it's a hard problem:

- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Issues and Challenges

Generally, it's a hard problem:

- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Solution Techniques

- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal

Issues and Challenges

Generally, it's a hard problem:

- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Solution Techniques

- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal
- stochastic trajectory optimization
  - cannot handle complex constraints, e.g. narrow passages

Issues and Challenges

Generally, it's a hard problem:

- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Solution Techniques

- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal
- stochastic trajectory optimization
  - cannot handle complex constraints, e.g. narrow passages
- linearize/convexify the problem
  - might become too conservative or not realizable; might not scale to complex constraints

Issues and Challenges

Generally, it's a hard problem:

- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

Solution Techniques

- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal
- stochastic trajectory optimization
  - cannot handle complex constraints, e.g. narrow passages
- linearize/convexify the problem
  - might become too conservative or not realizable; might not scale to complex constraints
- discretize the space and use discrete search
  - not scalable: exponential in state dimension and time
  - dynamic constraints difficult to handle

Issues and Challenges

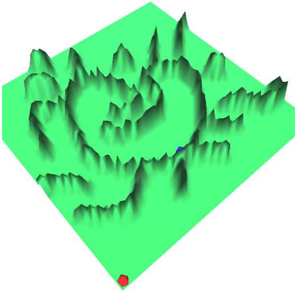
Generally, it's a hard problem:

- no closed-form solution in general (beyond the linear-quadratic case)
- infinite dimensional; numerically NP-complete

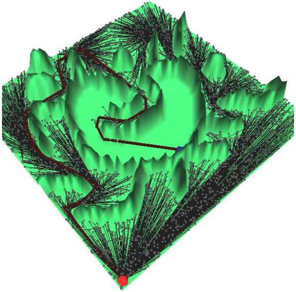
Solution Techniques

- nonlinear optimization (over finite trajectory parameterization)
  - could be slow, might not converge, only locally optimal
- stochastic trajectory optimization
  - cannot handle complex constraints, e.g. narrow passages
- linearize/convexify the problem
  - might become too conservative or not realizable; might not scale to complex constraints
- discretize the space and use discrete search
  - not scalable: exponential in state dimension and time
  - dynamic constraints difficult to handle
- sampling-based methods
  - randomized approximation of the space of trajectories (e.g. as a graph with randomly sampled nodes) and then search
  - By law of large numbers it approaches the optimal solution but typically at a very slow rate

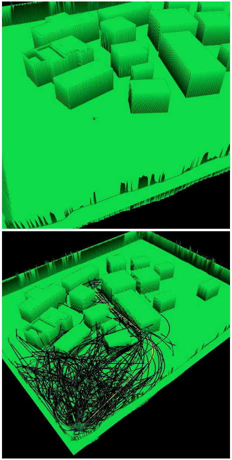
Example: Tree-based Sampling Motion Planning



optimal motion?



probabilistic roadmap (PRM)

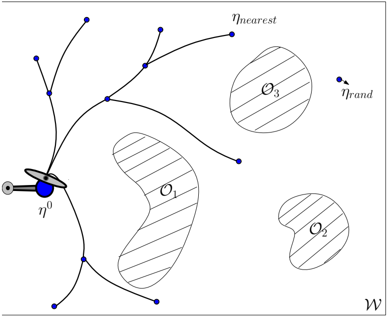


kinodynamic planning

The basic algorithm: rapidly-exploring random tree (RRT)

Algorithm 1: $\mathcal{T} \leftarrow \texttt{RRT}(\eta_0)$

$\mathcal{T} \leftarrow \texttt{InitializeTree}()$
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, \eta_0, \mathcal{T})$
for $i = 1 : N$ **do**
  $\eta_{\text{rand}} \leftarrow \texttt{Sample}$
  $\eta_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, \eta_{\text{rand}})$
  $(\bar{x}_{\text{new}}, \bar{u}_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(\eta_{\text{nearest}}, \eta_{\text{rand}})$
  if $\texttt{ObstacleFree}(\bar{x}_{new})$ **then**
    $\mathcal{T} \leftarrow \texttt{InsertNode}(\eta_{\text{nearest}}, \eta_{\text{new}}, \mathcal{T})$
return $\mathcal{T}$

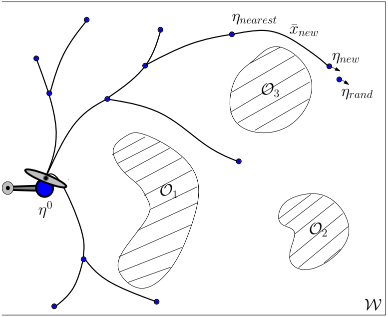


- A node is the tuple $\eta_i = (x_i, p_i, J_i) \in \mathcal{N} = \mathcal{X} \times \mathbb{N} \times \mathbb{R}_+$ where
  - $x_i \in \mathcal{X}$ is the state
  - $p_i \in \mathbb{N}$ is the index of the *parent* node of i, i.e. η_{p_i} is the parent of η_i
  - J_i is the cumulative cost from the start η_0 to η_i
- A tree $\mathcal{T} \subset \mathcal{N}$ is a particular arrangement of nodes

The basic algorithm: rapidly-exploring random tree (RRT)

Algorithm 2: $\mathcal{T} \leftarrow \texttt{RRT}(\eta_0)$

$\mathcal{T} \leftarrow \texttt{InitializeTree}()$
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, \eta_0, \mathcal{T})$
for $i = 1 : N$ **do**
  $\eta_{\text{rand}} \leftarrow \texttt{Sample}$
  $\eta_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, \eta_{\text{rand}})$
  $(\bar{x}_{\text{new}}, \bar{u}_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(\eta_{\text{nearest}}, \eta_{\text{rand}})$
  if $\texttt{ObstacleFree}(\bar{x}_{new})$ **then**
    $\mathcal{T} \leftarrow \texttt{InsertNode}(\eta_{\text{nearest}}, \eta_{\text{new}}, \mathcal{T})$
return $\mathcal{T}$



- A node is the tuple $\eta_i = (x_i, p_i, J_i) \in \mathcal{N} = \mathcal{X} \times \mathbb{N} \times \mathbb{R}_+$ where
  - $x_i \in \mathcal{X}$ is the state
  - $p_i \in \mathbb{N}$ is the index of the *parent* node of i, i.e. η_{p_i} is the parent of η_i
  - J_i is the cumulative cost from the start η_0 to η_i
- A tree $\mathcal{T} \subset \mathcal{N}$ is a particular arrangement of nodes

Key ingredients

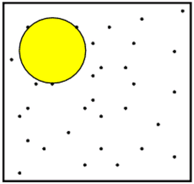
- sampling routine `Sample`
- distance function $\rho(x_a, x_b) \geq 0$ for determining `Nearest` $(\mathcal{T}, \eta)$
- steering method `Steer` (η_a, η_b)
- collision detection `ObstacleFree`(x)

Key ingredients: sampling routine `Sample`

- Baseline: uniform sampling
  - *low-dispersion*: reduce largest unsampled space between all samples

$$\delta(P) = \sup_{x \in \mathcal{X}} \{ \min_{x' \in P} \{ \rho(x, x') \} \},$$

    where P is a set of sampled points

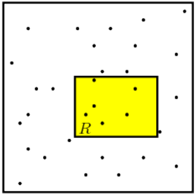


dispersion

  - *low-discrepancy*: $\#$ of samples inside a set are consistent with the volume of the set

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

    where $\mathcal{R}$ are all subsets of $\mathcal{X}$ and μ measures the volume of a set



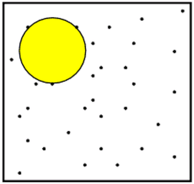
discrepancy

Key ingredients: sampling routine `Sample`

- Baseline: uniform sampling
  - *low-dispersion*: reduce largest unsampled space between all samples

$$\delta(P) = \sup_{x \in \mathcal{X}} \{ \min_{x' \in P} \{ \rho(x, x') \} \},$$

    where P is a set of sampled points

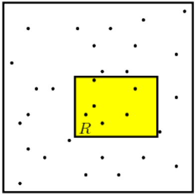


dispersion

  - *low-discrepancy*: $\#$ of samples inside a set are consistent with the volume of the set

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

    where $\mathcal{R}$ are all subsets of $\mathcal{X}$ and μ measures the volume of a set



discrepancy

- non-uniform sampling: exploiting problem structure (more later)

Key ingredients: distance function

Distance function $\rho(x_a, x_b) \geq 0$ for determining $\texttt{Nearest}\,(\mathcal{T}, \eta)$

- ideal distance is the true cost-to-go from x_a to x_b, i.e. $\rho(x_a, x_b) = J(\bar{x}_{a \to b}, \bar{u}_{a \to b})$

Key ingredients: distance function

Distance function $\rho(x_a, x_b) \geq 0$ for determining $\texttt{Nearest}(\mathcal{T}, \eta)$

- ideal distance is the true cost-to-go from x_a to x_b, i.e. $\rho(x_a, x_b) = J(\bar{x}_{a\to b}, \bar{u}_{a\to b})$
- which typically unavailable or expensive to compute so use a lower bound *heuristic cost*, e.g.

$$\rho(x_a, x_b) = \sqrt{(x_b - x_a)^T W (x_b - x_a)},$$

i.e. a weighted Euclidean distance (for some matrix $W > 0$)

Key ingredients: distance function

Distance function $\rho(x_a, x_b) \geq 0$ for determining $\texttt{Nearest}\,(\mathcal{T}, \eta)$

- ideal distance is the true cost-to-go from x_a to x_b, i.e. $\rho(x_a, x_b) = J(\bar{x}_{a\to b}, \bar{u}_{a\to b})$
- which typically unavailable or expensive to compute so use a lower bound *heuristic cost*, e.g.

$$\rho(x_a, x_b) = \sqrt{(x_b - x_a)^T W (x_b - x_a)},$$

  i.e. a weighted Euclidean distance (for some matrix $W > 0$)
- $\texttt{Nearest}\,(\mathcal{T}, \eta)$ can be set by:
  - $\rho(x_a, x_b)$: local distance ordering, i.e. standard RRT
  - $J_a + \rho(x_a, x_b)$: cost-to-come to parent + local distance ordering, i.e. RRT with optimal cost-to-come

Key ingredients: steering method $\texttt{Steer}(\eta_a, \eta_b)$

- Structured models (assume controllability)
  - open-loop trajectory generation: exploit nonholonomy, flatness, symmetries, if possible
  - employ efficient closed-form local methods, e.g. polynomial boundary value solutions

Key ingredients: steering method $\texttt{Steer}(\eta_a, \eta_b)$

- Structured models (assume controllability)
  - open-loop trajectory generation: exploit nonholonomy, flatness, symmetries, if possible
  - employ efficient closed-form local methods, e.g. polynomial boundary value solutions
- Complicated / Black box models:
  - only possible to sample control space
  - observe/simulate generated trajectories
  - must be *resolution complete*: i.e. reach infinitely close to any state
  - typically implies a regularity condition: that small change in u result in small change in x

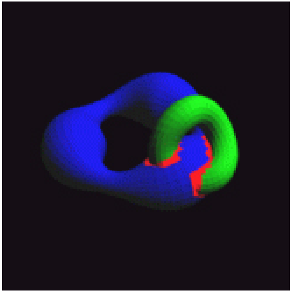
Key ingredients: steering method $\texttt{Steer}(\eta_a, \eta_b)$

- Structured models (assume controllability)
  - open-loop trajectory generation: exploit nonholonomy, flatness, symmetries, if possible
  - employ efficient closed-form local methods, e.g. polynomial boundary value solutions
- Complicated / Black box models:
  - only possible to sample control space
  - observe/simulate generated trajectories
  - must be *resolution complete*: i.e. reach infinitely close to any state
  - typically implies a regularity condition: that small change in u result in small change in x
- Steering using a finite set of primitives
  - primitives must be carefully chosen to satisfy controllability
  - in this case controllability is equivalent to *resolution completeness*

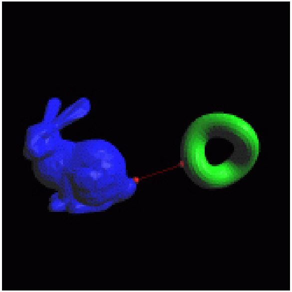
Key ingredients: collision checking $\texttt{ObstacleFree}(\bar{x}_{\text{new}})$

- ensure constraints $c(t, x, u) \leq 0$ are satisfied
- often the free configuration space is difficult to compute
- easiest to use a black-box collision checking package
- simulate controls $u(t)$ and check collision

Example: Proximity Query Package (PQP)
`http://gamma.cs.unc.edu/SSV/`



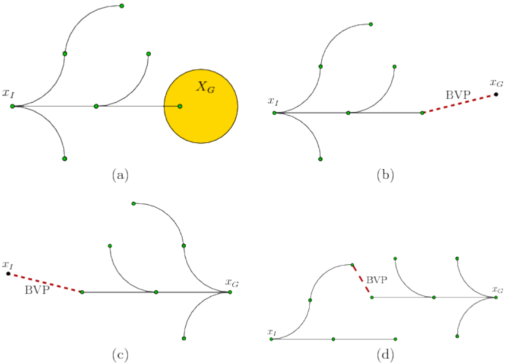
PQP collision checking



PQP distance and direction

Tree-based planners

Various tree-based planners are possible (LaValle, 2006)



It is critical to solve the boundary value (steering) problem (BVP)

a) standard planning to a goal set X_G

b) reaching a specific goal

c) tree grown backwards from goal

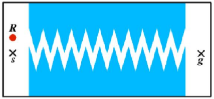
d) bidirectional tree: forward from start and backward from goal

Key challenges in motion planning

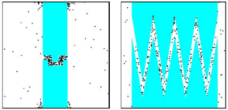
- achieving efficiency even in high dimensions
- handling complicated constraints, e.g. narrow passages
- finding optimal not just feasible solutions
- hybrid and non-smooth systems
- distributed systems planning, parallel processing
- dealing with uncertainty
  - partially known system dynamics
  - unstructured dynamic uncertain environment
  - formal robustness guarantees
- holy grail: unifying planning, estimation, and control

Workspace Adaptivity in Sampling-based methods

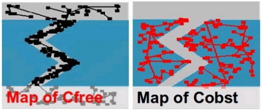
- Complex planning problems can be addressed through adaptation
- Example: handling narrow passages



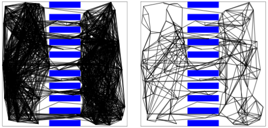
- Constructing roadmaps adaptively



Bridge Test



Toggle PRM

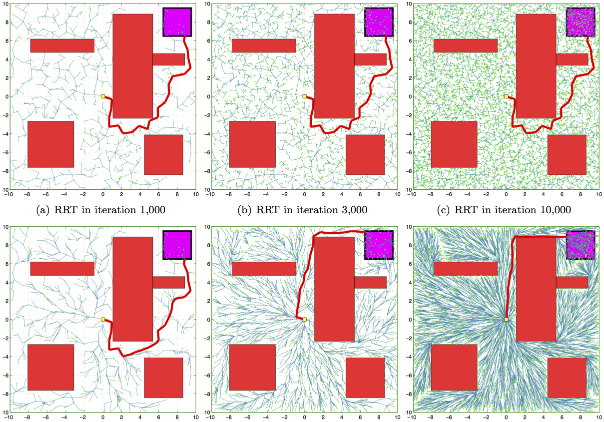


Roadmap Spanner

- Toggle PRM: A Coordinated Mapping of C-free and C-obstacle in Arbitrary Dimension, Jory Denny, Nancy M. Amato, WAFR, 2012. Proceedings
- Marble J, Bekris KE. 2013. Asymptotically Near-Optimal Planning with Probabilistic Roadmap Spanners. IEEE Transactions on Robotics. 29(3)
- David Hsu, Tingting Jiang, John Reif, Zheng Sun, The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners, ICRA, 2003
- many more: Kurniawati, Hsu; Ladd, Kavraki; Rickert, Brock, Knoll; etc...

From exploration to optimality

- Sampling-based methods are good at exploring the space to find “a path” but notoriously slow in converging to the “optimal” path.
- An important recent method: RRT* (Karaman, Frazzoli, 2011)
- Idea: rewire tree to maintain optimal cost-to-go
- Key result: only need to rewire by checking $\approx \log(n)$ neighbors
- Challenges: extend theory to complex dynamics; principled neighbor selection; CPU time?



(a) RRT in iteration 1,000 (b) RRT in iteration 3,000 (c) RRT in iteration 10,000

(d) RRT* in iteration 1,000 (e) RRT* in iteration 3,000 (f) RRT* in iteration 10,000

The RRT* algorithm

Algorithm 3: $\mathcal{T} \leftarrow \texttt{RRT}^*(\eta_0, \mathcal{X}_g)$

$\mathcal{T} \leftarrow \texttt{InitializeTree}()$
$\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, \eta_0, \mathcal{T})$
for $i = 1 : N$ **do**
  $\eta_{\text{rand}} \leftarrow \texttt{Sample}(i)$
  $\eta_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, \eta_{\text{rand}})$
  $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(\eta_{\text{nearest}}, \eta_{\text{rand}})$
  if $\texttt{ObstacleFree}(x_{new})$ **then**
    $\mathcal{N}_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, \eta_{\text{new}}, |V|)$
    $\eta_{\text{min}} = \texttt{ChooseParent}(\mathcal{N}_{\text{near}}, \eta_{\text{nearest}}, x_{\text{new}})$
    $\mathcal{T} \leftarrow \texttt{InsertNode}(\eta_{\text{min}}, \eta_{\text{new}}, \mathcal{T})$
    $\mathcal{T} \leftarrow \texttt{Rewire}(\mathcal{T}, \mathcal{N}_{\text{near}}, \eta_{\text{min}}, \eta_{\text{new}})$
return $\mathcal{T}$

The RRT* algorithm (cont.)

Algorithm 4: $\eta_{\text{min}} \leftarrow \texttt{ChooseParent}(\mathcal{N}_{\text{near}}, \eta_{\text{nearest}}, x_{\text{new}})$

$\eta_{\text{min}} \leftarrow \eta_{\text{nearest}}$
$c_{\text{min}} \leftarrow \texttt{CostToCome}(\eta_{\text{nearest}}) + \texttt{Cost}(x_{\text{new}})$
for $\eta_{near} \in \mathcal{N}_{near}$ **do**
    $(x', u', T') \leftarrow \texttt{Steer}(\eta_{\text{near}}, \eta_{\text{new}})$
    if $\texttt{ObstacleFree}(x')$ *and* $x'(T') = \eta_{new}$ **then**
        $c' = \texttt{CostToCome}(\eta_{\text{near}}) + \texttt{Cost}(x')$
        if $c' < c_{min}$ **then**
            $\eta_{\text{min}} \leftarrow \eta_{\text{near}}$
            $c_{\text{min}} \leftarrow c'$

return η_{min}

Algorithm 5: $\mathcal{T} \leftarrow \texttt{Rewire}(\mathcal{T}, \mathcal{N}_{\text{near}}, \eta_{\text{min}}, x_{\text{new}})$

for $\eta_{near} \in \mathcal{N}_{near} \backslash \{\eta_{min}\}$ **do**
    $(x', u', T') \leftarrow \texttt{Steer}(\eta_{\text{new}}, \eta_{\text{near}})$
    if $\texttt{ObstacleFree}(x')$ *and* $x'(T') = \eta_{near}$ *and*
      $\texttt{CostToCome}(\eta_{new}) + \texttt{Cost}(x') < \texttt{CostToCome}(\eta_{near})$ **then**
        $\mathcal{T} \leftarrow \texttt{Reconnect}\,(\eta_{\text{new}}, \eta_{\text{near}}, \mathcal{T})$

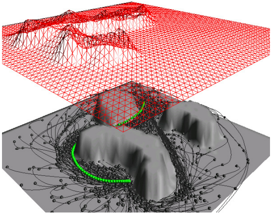
return $\mathcal{T}$

Towards optimal adaptive sampling

But still all these methods sample from the space of states: information about trajectory cost is not fully exploited

New method: Cross-entropy motion planning

- it is not necessary to sample everywhere uniformly
- adaptively sample nodes by exploiting cost information
- perform *density estimation* of low-cost regions in trajectory space
- "learn" regions in state space where "good" trajectories lie

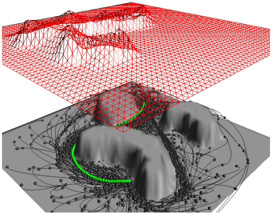


Adaptive density discovers salient regions for obtaining samples

TCE Cross-entropy Planning

Algorithm Overview: Trajectory-Cross-Entropy (TCE) Motion Planning

0. Expand RRT/PRM and attempt to connect to goal region
1. Obtain all RRT/PRM trajectories $\{\pi_i\}_{i=1}^{N}$ reaching the goal
2. Construct parametrized trajectories $Z_i = \psi(\pi_i)$
3. Update $p_{\mathcal{Z}}$ using the elite subset of these parameters
4. Sample a trajectory $Z \sim p_{\mathcal{Z}}$
5. Select one or more states $X = \varphi(Z, t)$ for a random t and add to RRT/PRM
6. Repeat from either (0) or (1) with some probability. Stop on a termination condition.



The density over trajectories $p_Z(Z)$ induces a density $p_{\mathcal{X}}(X)$ over states:

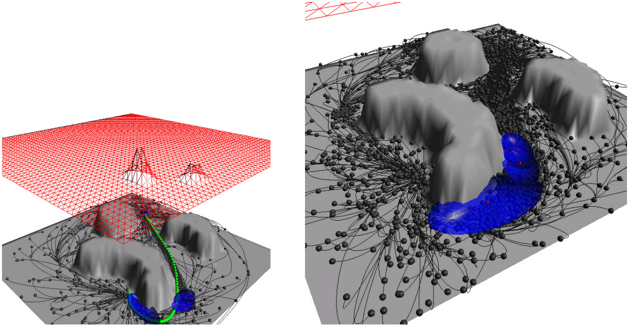
$$p_{\mathcal{X}}(X) = \eta \cdot \max_{Z \in \mathcal{Z}_{\text{con}}} \{p_{\mathcal{Z}}(Z) \mid X = \varphi_x(Z, t) \text{ for some } 0 < t < \tau(Z)\}, \qquad (6)$$

M. Kobilarov: Cross-Entropy Motion Planning, *International Journal of Robotics Research*, (2012).

SCE Cross-entropy Planning

Algorithm Overview: State-Cross-Entropy (SCE) Motion Planning

0. Expand RRT/PRM and attempt to connect to goal region
1. Obtain all RRT/PRM trajectories $\{\pi_i\}_{i=1}^N$ reaching the goal
2. Discretize each trajectory π_i into a set of states
3. Update $p_{\mathcal{X}}$ using the elite subset of all states of discretized trajectories
4. Sample a state $X \sim p_{\mathcal{X}}$ and add to RRT/PRM
5. Repeat from either (0) or (1) with some probability. Stop on a termination condition.



SCE-RRT* with adaptive Gaussian Mixture Model sampling