Module 3: Solving Problems by State Space Search and Motion Planning

Asst. Prof. Girish Chowdhary,
UIUC

February 8, 2018

Outline

- Formulating problems
- Solving Problems by State Space Search
- Uninformed search strategies
- Informed Heuristics based search
- Locally optimal search algorithms
- Local search in continuous spaces

Reading: Chapter 3 and 4 from Russell and Norvig

Learning Outcomes

- Search:
  - Solving decision making problems through search
  - Search techniques
- Motion planning
  - Configuration spaces, groups, and SO(3)
  - Sampling based motion planning
- Planning
  - Linear Programming
  - Chance constrained optimization and the notion of Risk

Formulating Problems

- Recall: Goal driven agents: goals instead of conditional action policies
- Complex problems: many many options $\rightarrow$ actions
- Example problem: Travel itinerary, planning a semester, strategics for business, winning a war...

Formulating Problems

- Recall: Goal driven agents: goals instead of conditional action policies
- Complex problems: many many options $\rightarrow$ actions
- Example problem: Travel itinerary, planning a semester, strategics for business, winning a war...
- How to reduce the set of actions?
- Goal formulation: helps in organizing a higher level problem
- Goal formulation: the first step in problem solving

Problem formulation

- Problem formulation: deciding on sets of states and actions based on the goal
- Clearly goal definition depends on the information that an agent has
- If the agent has no information: the agent can do no better than choosing a random action
- Uncertainty in the world drives what actions the agent can choose
- Another difficulty could be lack of perception: cannot sense the state that the agent is in
- Finally, stochasticity: agent's actions don't always lead to the expected outcome

Policies and autonomous decision making

- What makes autonomy hard?
  - Lack of precise goal structure: abstract goal: ”find the bad guy”, ”win a foot ball game”...
  - difficulty in identifying general classes of problem
  - Difficulty in finding a one-fit-all solution $\rightarrow$ need higher level abstraction
- Policy: a rule to make decisions to achieve goals based on current state
- Let’s assume: full observability
- Knowledge and certainty
- Deterministic behavior
- Policy: a sequence of actions

Well defined Problems

state: a variable required to completely describe the agent's "condition"
Example: Romania problem has 20 states, one for each city.
How many states does Black Jack have?

- The initial state
- The set of actions
- A transition model
- the Goal test
- Path cost or a utility function

Formal problem formulation

A problem is a Tuple $P = \langle S, A, P, G, r \rangle$

- S is the state space, x is a state (continuous vs discrete S)
- A is the action space, a is an action
- P is the transition model $x' = p(x, a)$, the resulting state x' when action a is taken in state x
- G is a goal state, defined perhaps through a goal test
- r defines the path cost through a utility function that tells us when good things happen

Challenge: How do we come up with sufficiently abstract representations for complex problems? • removing details, creating hierarchies, building interdependencies

Example problems

- Vaccum world
- Grid world
- Chess
- The asset allocation problem that we discussed in Module 2
- Route finding problem
- The traveling salesman problem
- The N-armed bandit problem

Solving problems by searching

Example problem: The sheep, the wolf, the hay, and the astronaut (courtesy: Brian Williams)

- http://www.coolmath-games.com/Logic-wolfsheepcabbage/

Searching

- Search Trees
- Evaluate options one by one
- Some notation: Parent node, child node, leaf node, frontier
- Frontier: set of all leaf nodes for an expansion at any given point

Graph search

- Tree search: Expand nodes on frontiers until a solution is found or no more expansion
- Problem: Loopy paths: paths that run around infinite loops
- Algorithms that forget their history are doomed to repeat it!
- Graph search: Contains a list of states that have been explored

Search complexity

- Completeness: is the algorithm guaranteed to find a solution when there is one
- Optimality: is the solution optimal
- Time complexity: computational time
- Space complexity: Memory

Uninformed Search

- Uninformed $\rightarrow$ Brute-Force
- Breadth First Search: Root node expanded first and then all the successors of the root node, and so on
- expand the shallowest node always
- Breadth search is complete,
- Time complexity: $O(b^d)$, b nodes and d depth
- Space complexity: need to store every expanded node in the explored set, so $O(b^d)$

Uniform-cost search

- Simple tweak to breadth first: expand the node n with the lowest path cost $g(n)$

- Priority queue, and a check on a shorter path

Depth first search

- The other way round: expand the deepest node in the current frontier of the search tree first
- not always complete
- Depth limited search
- Iterative deepening depth-first search

Heuristics

Heuristics

Heuristics are experience based techniques that “hope” to quickly find optimal solutions

- Example problems: Buying beer, finding the quickest way home, searching for victims in a fire situation
- Heuristics are not guaranteed to be optimal: why?

Heuristics

Heuristics

Heuristics are experience based techniques that “hope” to quickly find optimal solutions

- Example problems: Buying beer, finding the quickest way home, searching for victims in a fire situation
- Heuristics are not guaranteed to be optimal: why?
- Because the autonomous agent cannot change the heuristic
- There is a fine line between heuristics and learning based search improvement: infact, Reinforcement Learning is a heuristic free class of algorithms that also attempts to leverage experience

Informed Search

- We can use information about the state space (domain knowledge) to find solutions more efficiently
- General approach: best-first-search, an extension of tree search or graph search
- using an evaluation function $f(n)$, expand the most "high value" node first, i.e. in Russell and Norvig notation exapnd lowest cost ($f(n)$) node first
- Example the heuristic function: $h(n)$=Estimated cost of the cheapest path from node n to goal node
- So, when n=Goal, $h(n) = 0$
- Greedy best first search: $f(n) = h(n)$, expand the node closest to the Goal first
- "Greedy": Myopic action to get as close as possible to the goal

A^*

A^* evaluation function

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the pat h cost from start node to node n and $h(n)$ is the estimated cost of the cheapest path from n to the goal

- The next logical solution: combine the cost to reach the node $g(n)$ with the cost to get from the node to the goal $h(n)$
- $f(n)$ turns out to be the estimated cost of the cheapest solution through n
- A very reasonable strategy, guaranteed to be complete and optimal with the right choice of the heuristic $h(n)$

Conditions for completeness and optimality of A^*

Admissible heuristic

$h(n)$ is an admissible heuristic if it never overestimates the cost to reach the goal

The heuristic needs to be optimistic

Consistent heuristic

$h(n)$ is a consistent heuristic if for any successor node n' of n generated through action a

$$h(n) \leq c(n, a, n') + h(n')$$

- A form of **triangle inequality**
- Every consistent heuristic is also admissible
- Tree search with A^* is optimal if $h(n)$ is admissible
- Graph seaerch with A^* is optimal if $h(n)$ is consistent

A^*

- A^* is optimally efficient, that is no other algorithm is guaranteed to expand fewer nodes than A^*
- Absolute error $\Delta = h^* - h$, where h^* is the actual cost of getting to the root from the goal
- Complexity is typically exponential in the maximum absolute error
- A^* runs out of memory before it runs out of time
- several improvements over A^* exist: iterative deepening search, recursive best first search....

Why bother with the path

- Paths are important to many mobile robotics applications
- but in some cases, its most important to get to the end goal without bothering about the path
- example: the target assignment problem: could be solved in more than one way
- Instead of searching for the goal over the entire state-space, we can try to locally optimize a cost function
- Cost function terminology: local maxima, global maximum, flat local maximum, shoulder...

Hill climibing

Steepest ascent (greedy local search)

Graph search with Steepest ascent: Find the neighbor which has the highest value and go there.

Algorithm (reward function is known):

1. Evaluate reward of an action a as $V(x, a)$,
2. Take the action that has maximum reward, that is pick action $a_{next} = max_{a'}(V(x, a') - V(x, a))$

Issues

Hill climibing

Steepest ascent (greedy local search)

Graph search with Steepest ascent: Find the neighbor which has the highest value and go there.

Algorithm (reward function is known):

1. Evaluate reward of an action a as $V(x, a)$,
2. Take the action that has maximum reward, that is pick action $a_{next} = max_{a'}(V(x, a') - V(x, a))$

Issues

- Gets stuck in local maxima
- Ridges: sequence of local maxima, difficult to navigate
- Plateaux: gets lost on the plateau

Convergence of Stochastic Gradient Descent

Robbins, Herbert, and Sutton Monro. "A stochastic approximation method." The annals of mathematical statistics (1951): 400-407.

- A seminal paper that started a whole field
  - Given a cost function $J(\theta)$, where θ is random variable, find the minimum value of J.
  - Solution: Move in the direction of maximum descent pointwise in time: $\theta_{t+1} = \theta_t - \gamma_t \frac{\partial J}{\partial \theta}$
  - On the average will this converge?

Convergence of Stochastic Gradient Descent

Robbins, Herbert, and Sutton Monro. "A stochastic approximation method." The annals of mathematical statistics (1951): 400-407.

- A seminal paper that started a whole field
  - Given a cost function $J(\theta)$, where θ is random variable, find the minimum value of J.
  - Solution: Move in the direction of maximum descent pointwise in time: $\theta_{t+1} = \theta_t - \gamma_t \frac{\partial J}{\partial \theta}$
  - On the average will this converge?
  - Not necessarily!: Why: Stochastic behavior prevents convergence close to the origin
  - Robbins and Monro: Sufficient condition for convergence:
  - γ is a strictly non increasing sequence or (weaker assumption) $\sum_t \gamma_t = \infty$ and $\sum_t \gamma_t^2 < \infty$

Simulated Annealing

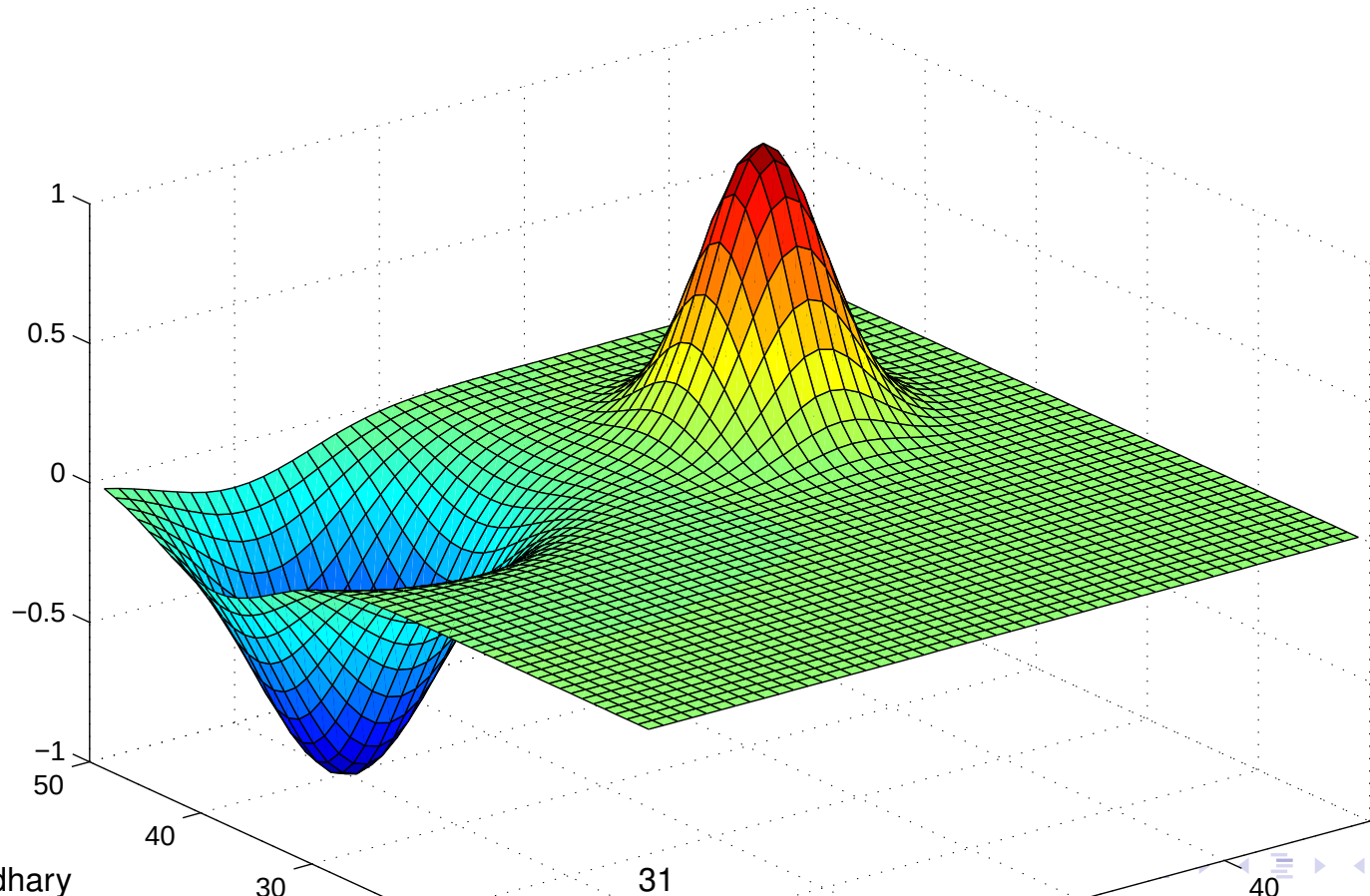
- How to dislodge from local maxima?
- Add randomness... Algorithm
  1. Pick a random action, increment time t
  2. If the action results in improved cost, i.e. $\Delta V > 0$ then pick it
  3. Else accept action with probability $p(\Delta V)$, p decreases exponentially with ΔV and with time t, e.g. $p = exp(\Delta V) + exp(t)$

Local beam search

- The same idea as hill climbing except with multiple searching agents
- maintain a swarm for k states, that are initialized at different locations
- Problem: agents can end up clustering with each other

Data driven gradient ascent

- How to find the optimum when you don't have the reward function?



Data driven gradient ascent

Try try and try again

When you don't have the reward function, your only option is to take random actions

Algorithm: ϵ-Greedy local grid search

1. Decide whether to explore or exploit, toss a biased coin, heads with $p = 1 - \epsilon$
2. Take a random action if *heads*, and obtain the reward at the resulting state, tabulate the reward
3. If *tails* use tabulated reward to compute best action (local gradient ascent or grid search)
4. Increase probability of getting tails

- **New complexity measure**: Sample Complexity: How many samples/trials do I need until I get ϵ close to the true solution?

Other algorithms

- Genetic algorithm

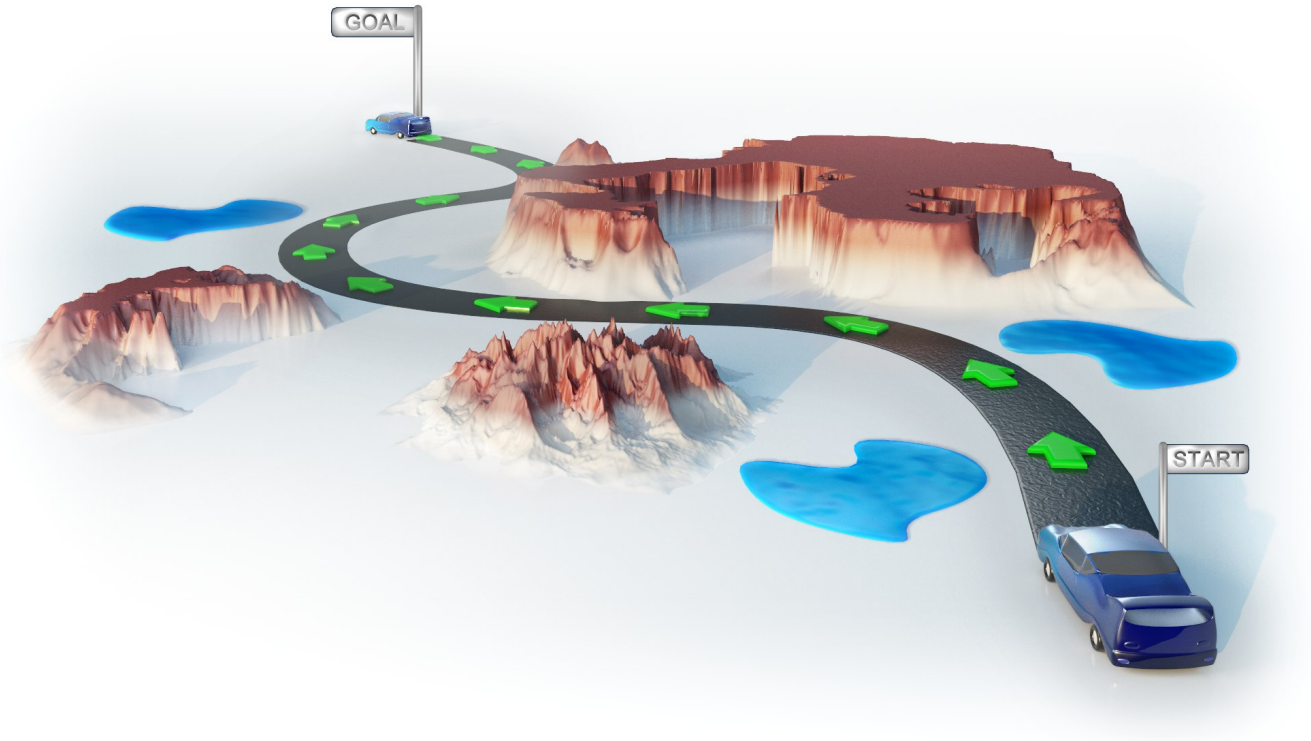
Outline: Motion Planning

- What is motion planning
- Geometric representations
- Configuration spaces
- Sampling based planning
- Reactive planning

Reading: Chapter 2, 3, 4, 5 from Lavalle Planning Algorithms, and Chapter 6 Siegwart et al. Autonomous Mobile Robots Chapter 2, until page 43, a good review of search

What is motion planning

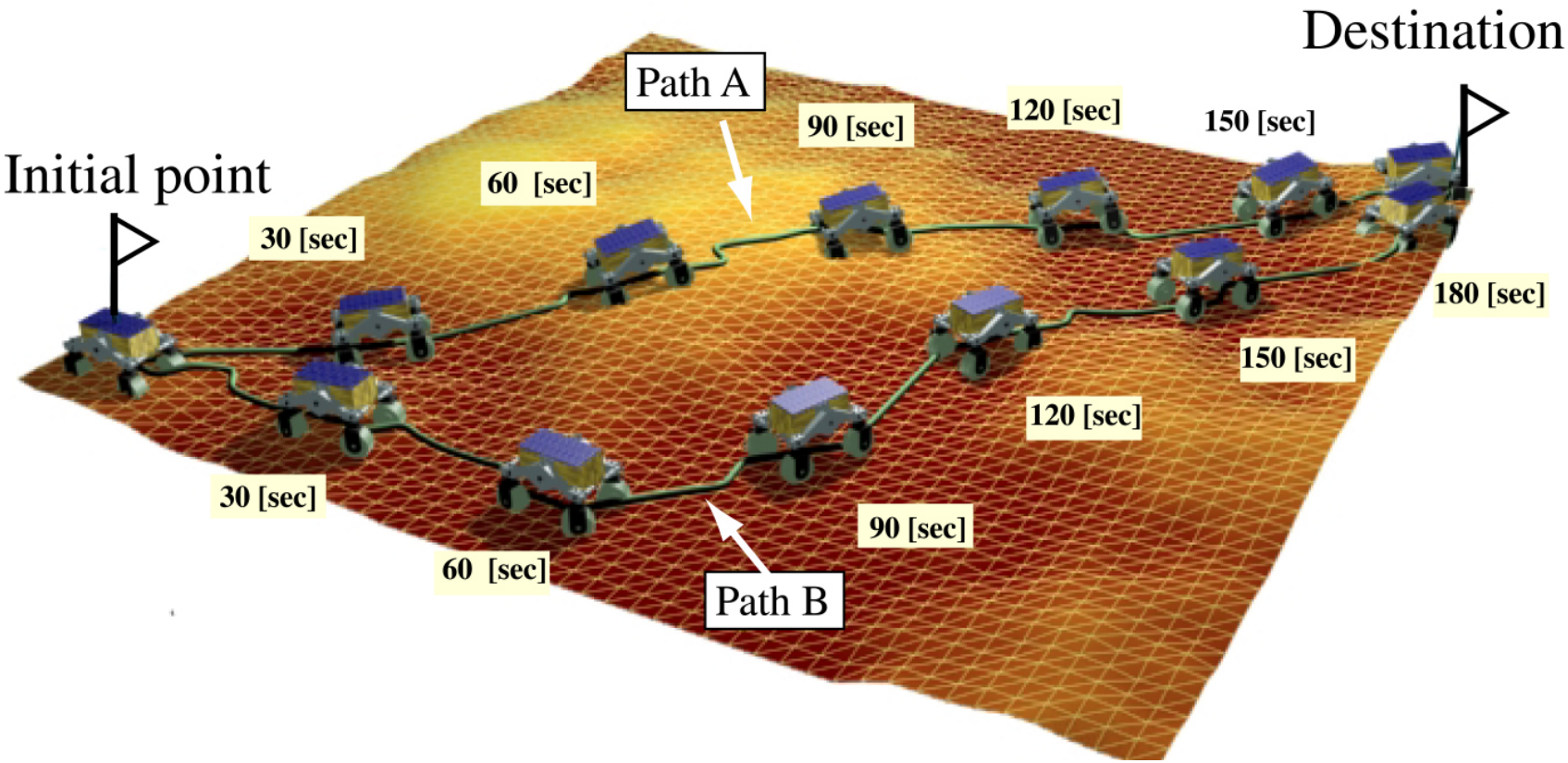
Getting from A to B



- The main difference: Planning in continuous (or very large) spaces

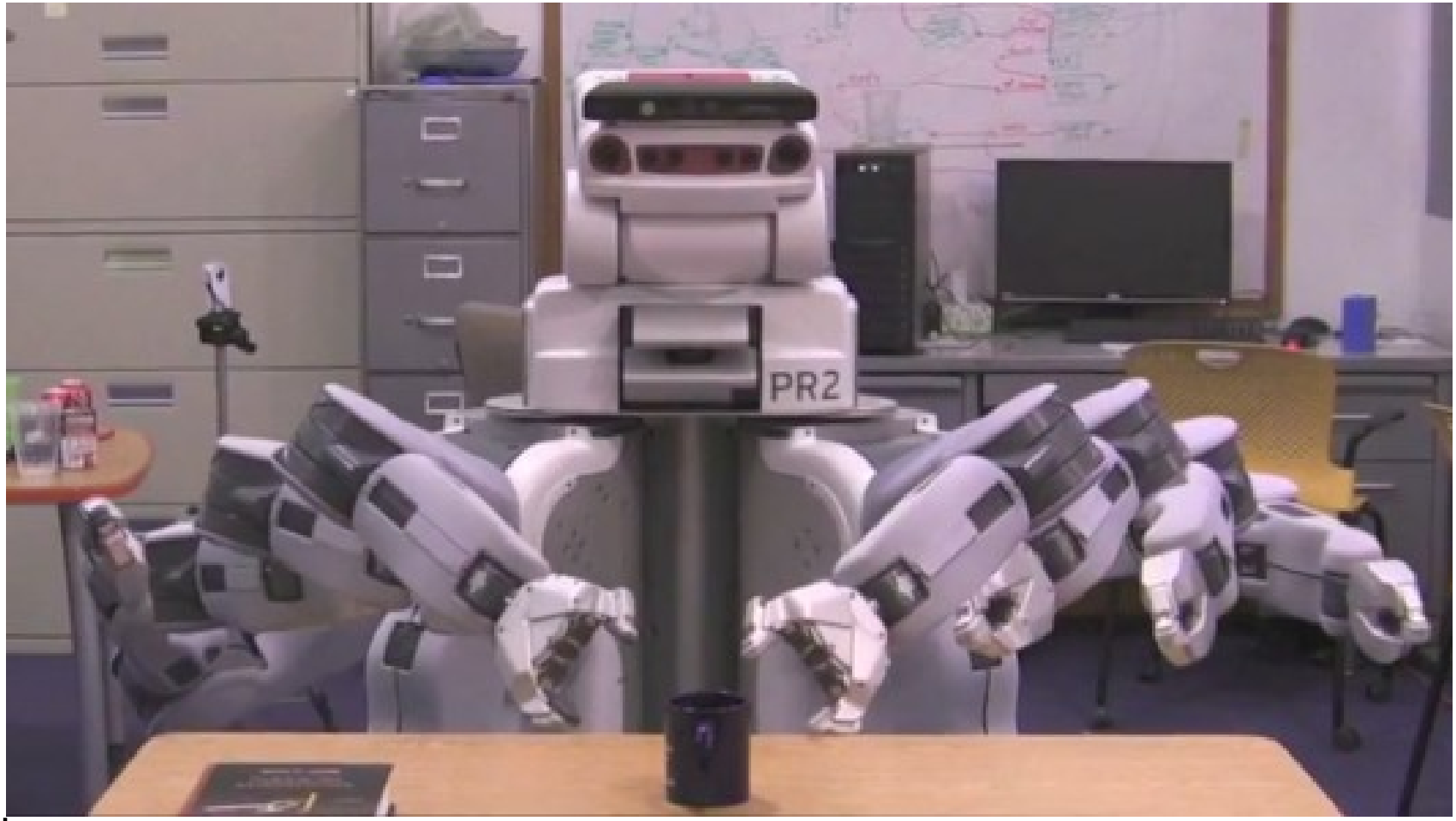
What is motion planning

What is the best path?



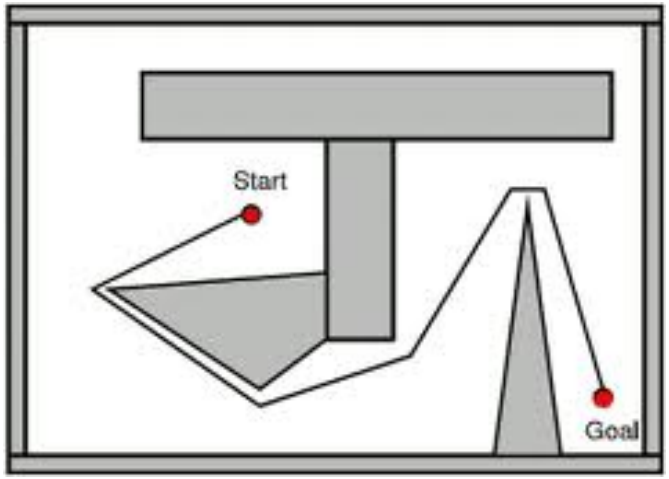
What is motion planning

Planning in robot configuration space



Geometric Abstractions

- The world W
- The objects and robots in the world O, A
- The obstacles in the world



Topological spaces

Topological Space

A set X is called a topological space if there is a collection of subsets of X called open sets for which the following axioms hold

- The union of a countable number of open sets is an open set
- The intersection of a finite number of open sets is an open set
- Both X and the empty set Φ are open sets

• **Closed set**: A set $C \subset X$ is a closed set if and only if $X \backslash C$ is an open set

• Interior points, exterior points, and boundary points

• **Compact Sets**: A set that is closed and bounded

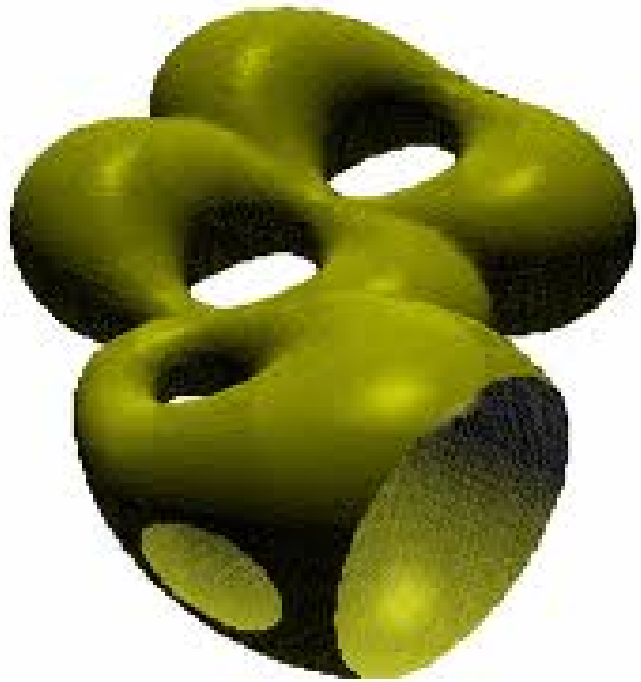
Manifolds

Manifolds

M is a manifold if $\forall\ x \in\ M\ \exists$ an open set $O \subset M$ s.t.

- $x \in M$
- O is homeomorphic to $\mathbb{R}^n$
- n is fixed

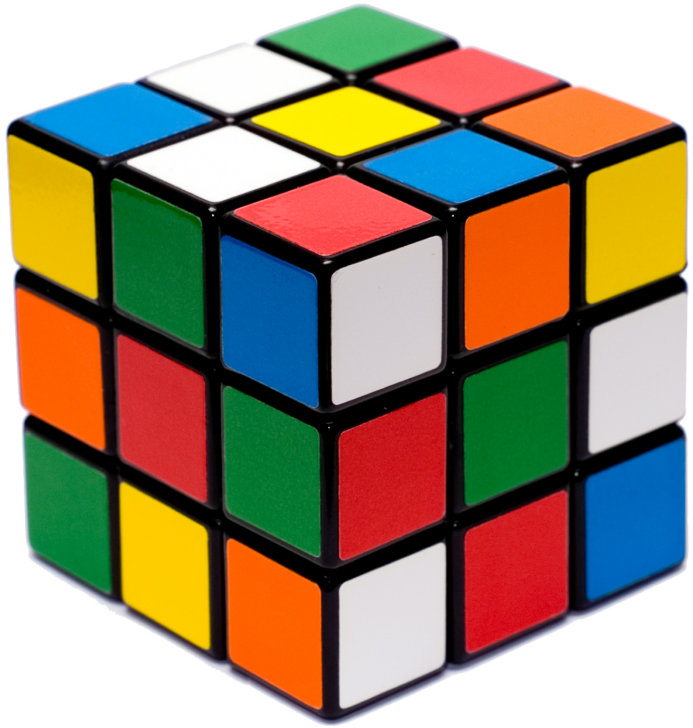
• The boundary of a manifold is by definition not included in the manifold



Groups

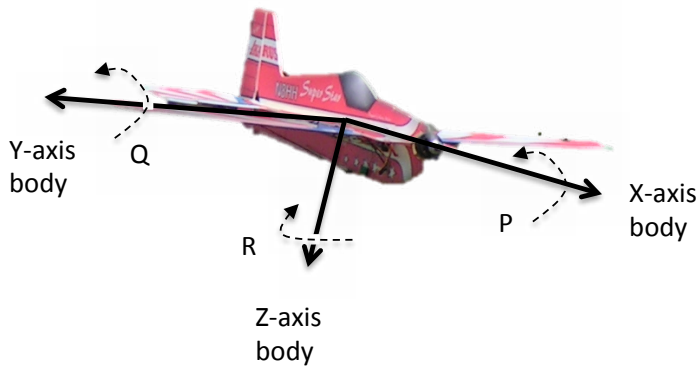
A group is a set G with a binary operation $\circ$ such that the following group axioms are satisfied

- Closure: $\forall\ a, b\ \in G$, the product $a\ \circ\ b \in G$
- Associativity: *forall* $a, b, c\ \in G$, $(a \circ b) \circ c = a \circ (b \circ c)$
- Identity: $\exists e \in G$ s.t. $a \circ e = a$ and $e \circ a = a$
- Inverse: $\forall a \in G\ \exists a^{-1}$ s.t. $a \circ a^{-1} = e$ and $a^{-1} \circ a = e$



the manipulations of a Rubik cube form a group

Quaternions



Aircraft body frame

- Euler angle representation runs into singularities
- Group theory provides us with a more robust representation: quaternions
- $q = (\mathring{q}, \vec{q})$ is typically written in terms of
- a scalar part $\mathring{q} \in \mathbb{R}$ and a vector part $\vec{q} \in \mathbb{R}^3$

Quaternions

A unit quaternion $q = (\mathring{q}, \vec{q})$ is such that

$$q \cdot q = \mathring{q}^2 + \vec{q} \cdot \vec{q} = 1.$$

The set of unit vectors in $\mathbb{R}^4$ is denoted as the sphere $\mathbb{S}^3$.

- the vector part of a unit quaternion is parallel to the fixed axis of the rotation,
- the scalar part determines the rotation angle
- Quaternion multiplication

$$q_1 \circ q_2 = (\mathring{q}_1 \mathring{q}_2 - \vec{q}_1 \cdot \vec{q}_2, \quad \mathring{q}_1 \vec{q}_2 + \mathring{q}_2 \vec{q}_1 - \vec{q}_1 \times \vec{q}_2),$$

or in matrix form:

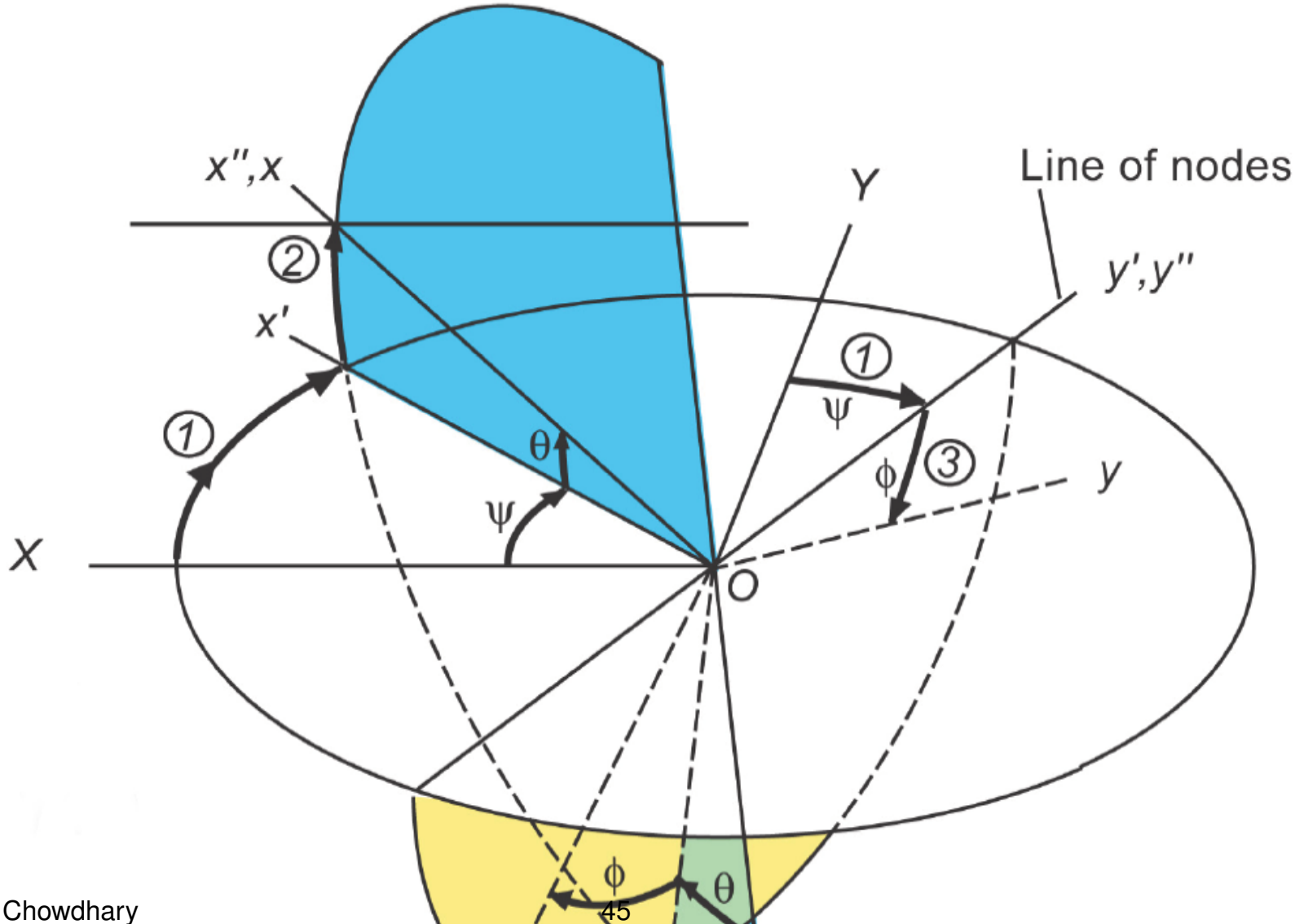
$$q_1 \circ q_2 = \begin{bmatrix} \mathring{q}_1 & -\vec{q}_1{}^T \\ \vec{q}_1 & \hat{\vec{q}} \end{bmatrix} q_2.$$

- the conjugate quaternion $q^* = (\mathring{q}, -\vec{q})$ corresponds to the "inverse" rotation, in the sense that $q \circ q^* = q^* \circ q = (1, 0)$.

Quaternions and SO(3)

- The set of all quaternions $\mathbb{H}$ with the quaternion multiplication operator forms a group
- Quaternion multiplication is not commutative
- Note that $\mathbb{H}$ is restrictied to $a^2 + b^2 + c^2 + d^2 = 1$
- A unit quaternion belongs to SO(3), but the representation is not unique
- So, $\mathbb{H}$ forms a double covering of SO(3)
- $\mathbb{H}$ is homeomorphic to $\mathbb{S}^3$ (the sphere manifold)
- to enforce uniqueness stay in the upper half of $\mathbb{S}^3$
- several ways to accomplish this, one is to enforce $a > 0$ and handle the $a = 0$ case

Euler angles



Euler angle and quaternions

A direct relationship can also be given between Euler angles and quaternions. Let the components of a quaternion be given by $q = [\mathring{q}, q_x, q_y, q_z]$; then

$$\begin{aligned} \theta &= -\arcsin 2(q_x q_z - \mathring{q} q_y), \\ \phi &= \operatorname{arctan2}\left[2(q_y q_z + \mathring{q} q_x), 1 - 2(q_x^2 + q_y^2)\right], \\ \psi &= \operatorname{arctan2}\left[2(\vec{q}_x q_y + \mathring{q} q_z), 1 - 2(q_y^2 + q_z^2)\right], \end{aligned} \tag{1}$$

and

$$\begin{aligned} \mathring{q} &= \pm(\cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2)), \\ q_x &= \pm(\sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2)), \\ q_y &= \pm(\cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2)), \\ q_z &= \pm(\cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2)), \end{aligned}$$

where the sign is arbitrary, but must be consistent.

Chains of bodies

The idea of C-space extends easily to multiple bodies through Cartesion producs:

$$C_{\text{combined}} = C_1 \times C_1 \times C_2 \times C_3 \ldots$$

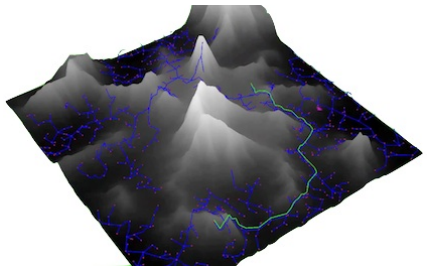
The motion planning problem in C-space

Simple formulation

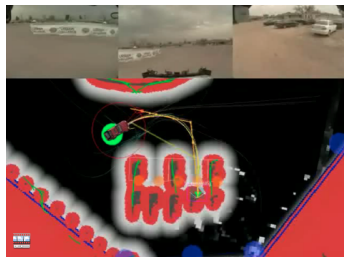
- World W, with robot A , and obstacles O
- Configuration space C, with obstacle, and free space
- Initial and final configurations
- A complete algorithm must compute a continuous path $\tau : [0, 1] \rightarrow C_{\text{free}}$ from the initial to the goal configuration

Why sampling based motion planning

- Avoid explicit modeling of *C*-space
- Instead probe the *C* space with some samples
- idea is to create an approximation based on the samples and search for optimal paths in that approximation
- Sampling based approaches may not be complete in the deterministic sense
- they may be probabilistically complete: i.e. complete with probability 1



http://ompl.kavrakilab.org/



Karaman and Frazzoli's RRT based motion planning

Sampling

- The C space could be infinite, but we can have only countably many samples
- The efficiency of our algorithim will depend on how good these samples are
- Challenge 1: pick the right measure for sampling:
  - For example, when sampling on $SO(3)$ Euler angles will introduce an unintentional bias
  - Quaternion parameterization instead will not, it is a Haar measure on $\mathbb{S}^3$
- since we are interested in quickly obtaining our solution, the *sequence* of the samples is also very important

Sampling sequences

- denseness of samples: A sequence is dense over a set $C \subseteq \mathbb{R}^n$ if it gets arbitrarily close to any element in C
- more formally: A topological subset U is dense over V if $cl(U) = V$, that is the closure of V is U
- A random sequence is probably dense (almost surely)

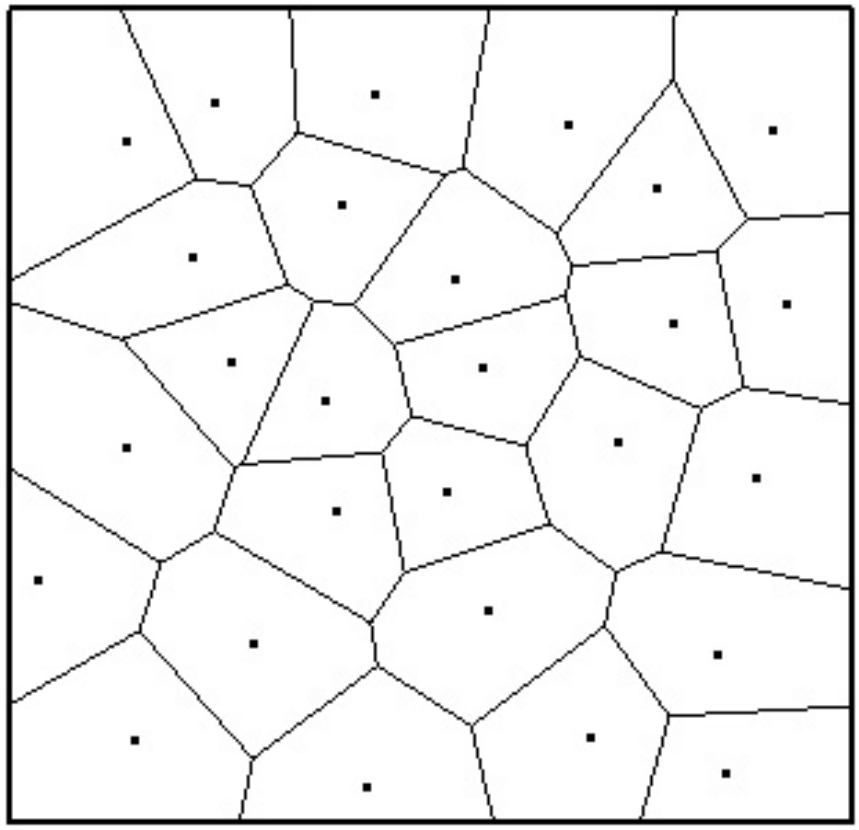
Almost Surely

For a probability space (Ω, F, P) an event $E \in F$ happens almost surely if $P[E] = 1$, equivalently $P[E^C] = 0$

e.g. getting snake-eyes in the game of Craps eventually is almost surely true

e.g. The square example: almost surely you will never hit a point on the diagonal of a square: holds because the area of the diagonal of the square is zero, so it is a set of measure zero

Sampling



How good is my sampling

- Naive: I want to cover as much of my space possible with as little samples as possible
- Low dispersion sampling: Make the largest covered area as small as possible
- Similar to putting grids of different size over the space: resolution complete
- Dispersion of a set p over a metric space (X, ρ)

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

- like the radius of the largest empty ball
- Discrepancy: how many points fall in box that I pick: Largest error in volume estimate using the samples

Incremental sampling and searching

Problem formulation: find an optimal path between q_1 and q_g

- Initialize the undirected graph: $G(V, E)$
- Select vertex sets
- Plan locally
- build the graph
- Check quality of solution

Rapidly Exploring Dense Trees

- Incrementally construct a search tree that does not need to explicitly set resolution
- This tree should densely cover the space
- Avoids having to set parameters, naturally encodes memory

RDT Vanilla

Algorithm: Rapidly Exploring Random Trees (RRT LaValle and Kuffner 01)

- initialize
- Add a vertex based on a sample $\alpha(i)$
- Find the nearest point in the Graph *Swath* q_n
- Add an edge (q_n, α_i), this may require edge splitting

RRT

Algorithm 1: RRT

1: Initialize Graph $G(V, E)$
2: **for** $i = 1..k$ **do**
3: sample α_i
4: $q_n \leftarrow$ Find-nearest(G, α_i)
5: add-edge(q_n,$alpha_i$)
6: **end for**

RRT*

Algorithm 2: RRT*

1: Initialize Graph $G(V, E)$
2: **for** $i = 1..k$ **do**
3: sample n points α_i
4: $q_n \leftarrow$ Find-nearest(G, α_i)
5: find all nearest *log n* neighbors
6: add-edge*(q_n,*alpha*$_i$) {Only add those edges that result in minimum cost path}
7: reorder-edge(G) {Create new edge to α_i if path through α_i has less cost than current parent}
8: prune-edge(G) {Remove old edges to maintain the tree structure}
9: **end for**

Probablistic Roadmap

- What if there are multiple initial and goal locations
- Idea: spend time carefully creating a graph that makes it easy to find multiple paths
  - pre-processing: Sample n points α_i attempt connections to α_i from other vertices in V within some radius r from α_i (if the radius is a function of n we get PRM*)
  - build a path using a simple planner: straight lines
  - Add edges in collision free connections, avoid redundant connections in the same connected component

Feedback motion planning

- The techniques we have seen until now have been somewhat “open-loop”
- They provided us with a path in the configuration space, and we hoped that we could track it as well as we can
- Feedback, or policy based, motion planning techniques provide us instead with a feedback law
- Implementing this law should result in an “optimal” path

Formal problem description (Formulation 8.1 Lavalle) modified with $k = t$ with a discrete time step

- State space X
- State-dependent action space $U(x)$, each action $u \in U(x)$
- A state transition function $f(x, u)$ that provides next state $x_{t+1} = f(x, u)$
- A cost function, a final state x_f:

$$L(\bar{x}_f, \bar{u}_t) = \sum_{t=1}^{T} l(x_t, u_t) + l_f(x_f)$$

- we seek for a policy $\pi(x)$ that returns the optimal u w.r.t. $L(\bar{x}_f, \bar{u}_t)$
- The problem is said to be feasibly if a (possibly suboptimal) policy *pi* exists such that the goal can be reached

Navigation function based planning

- Potential function $\phi : X \to [0, \infty]$: the underlying idea is to emulate a force field acting on the agent
- Purely reactive planning:

$$u^* = \arg \min_{u \in U(x)} [\phi(f(x, u))]$$

- Optimal planning

$$u^* = \arg \min_{u \in U(x)} [l(x, u) + \phi(f(x, u))]$$

- if $l(x, u) = l(x, u')$, that is stage-cost is not action dependent, then the both are the same. example: minimum time problems
- Feasible Navigation functions are well-behaved Potential functions:
  - $\phi(x) = 0$ at the goal stage (i.e. $\forall x \in X_G$)
  - $\phi(x) = \infty$ if and only if no point in X_G is reachable
  - For every state $x \in X \backslash X_G$, u^* is such that $\phi(f(x, u^*)) \leq \phi(x)$