Principles of Robot Autonomy I

Planning and Control: A* path planning, RRT and sampling based planning

Principles of Robot Autonomy I

Announcements:

- Lab section 1 this week
- HW 1 due next Thurs, Oct 9
- Lab sections assigned, in person labs start Monday
- Slides uploaded to canvas before lecture

Lecture 4:

- Dijkstras and A* graph planning
- PRM
- RRT, RRT* and re-wiring

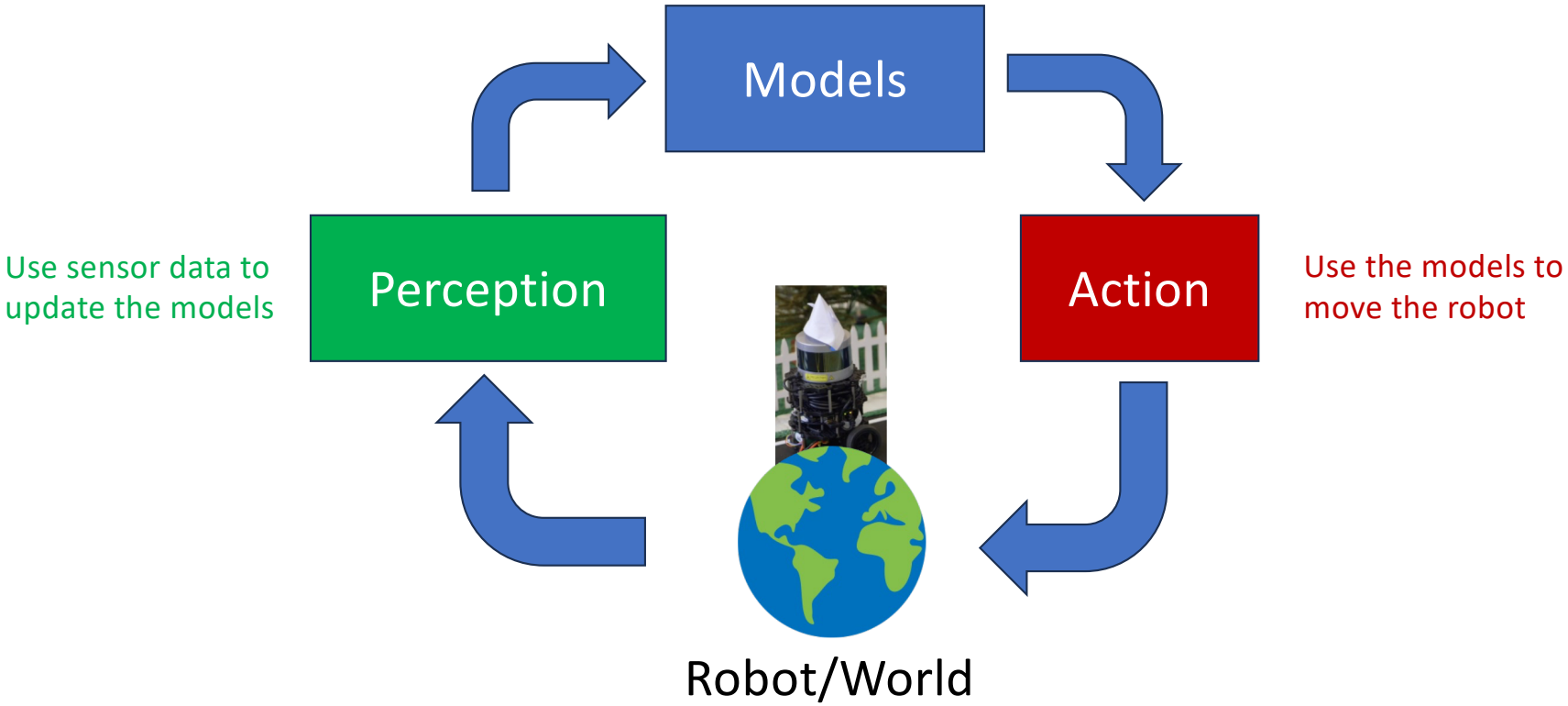
Stanford University

MSL

IPRL

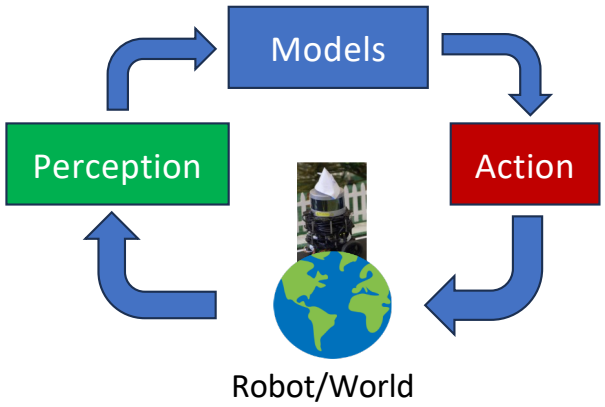
ASL Autonomous Systems Lab

Full Stack Autonomy: the Perception-Action Loop



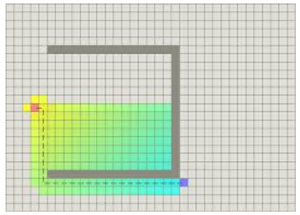
Action: Planning and Control Stack

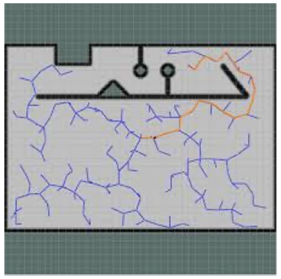
Use the models to move the robot



Path planning:

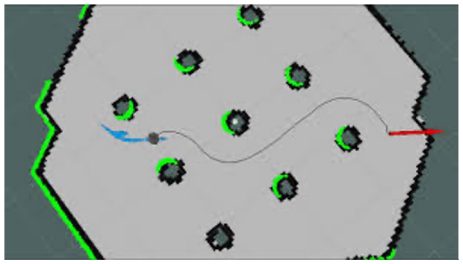
- A* planning on a graph
- RRT, RRT*





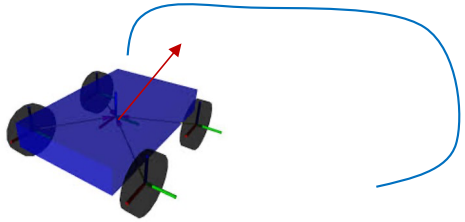
Trajectory optimization:

- Optimal control
- Numerical solvers



Trajectory following:

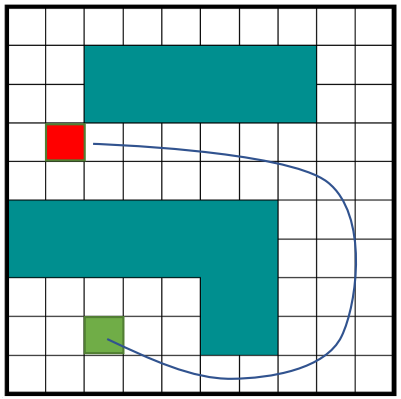
- Kinematic/dynamic models
- PID control
- LQR, gain scheduled LQR



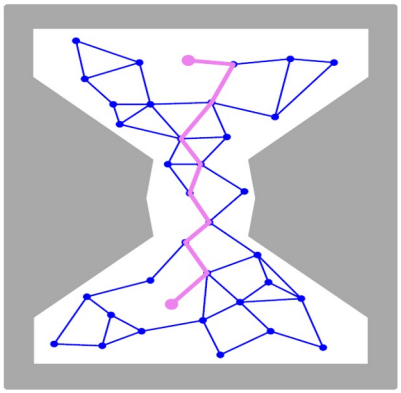
Path Planning

Map representation

Graph search planning, A*



Planning on random graph, RRT*, PRM



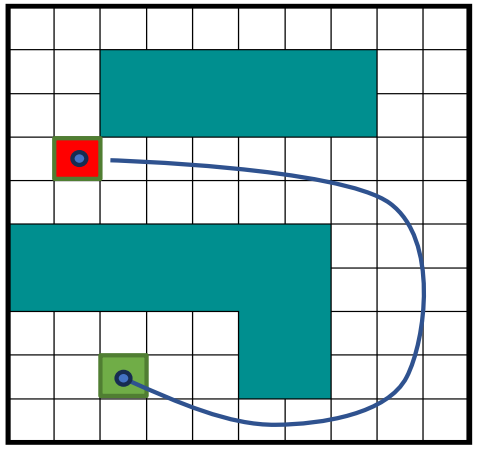
Goal: Find a path so the robot doesn't hit things!

Graph search planning

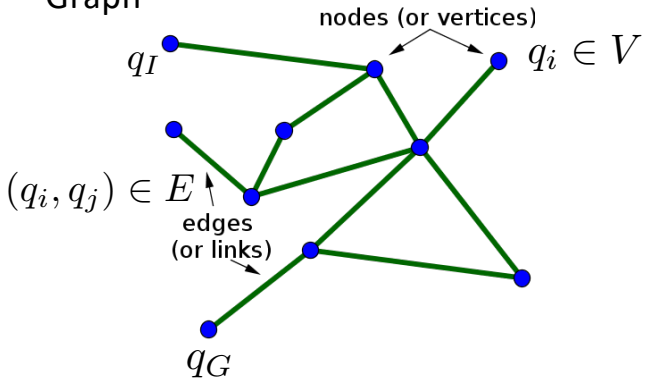
- Discretize robot's **configuration space** into a grid
  - Each grid cell is either free or occupied
  - Robot moves between adjacent free cells
  - **Goal:** find shortest sequence of free cells from start to goal

- Mathematically, this corresponds to pathfinding in a discrete graph $G = (V, E)$
  - Each vertex $q_i \in V$ represents a free cell
  - Edges $(q_i, q_j) \in E$ connect adjacent grid cells

Map

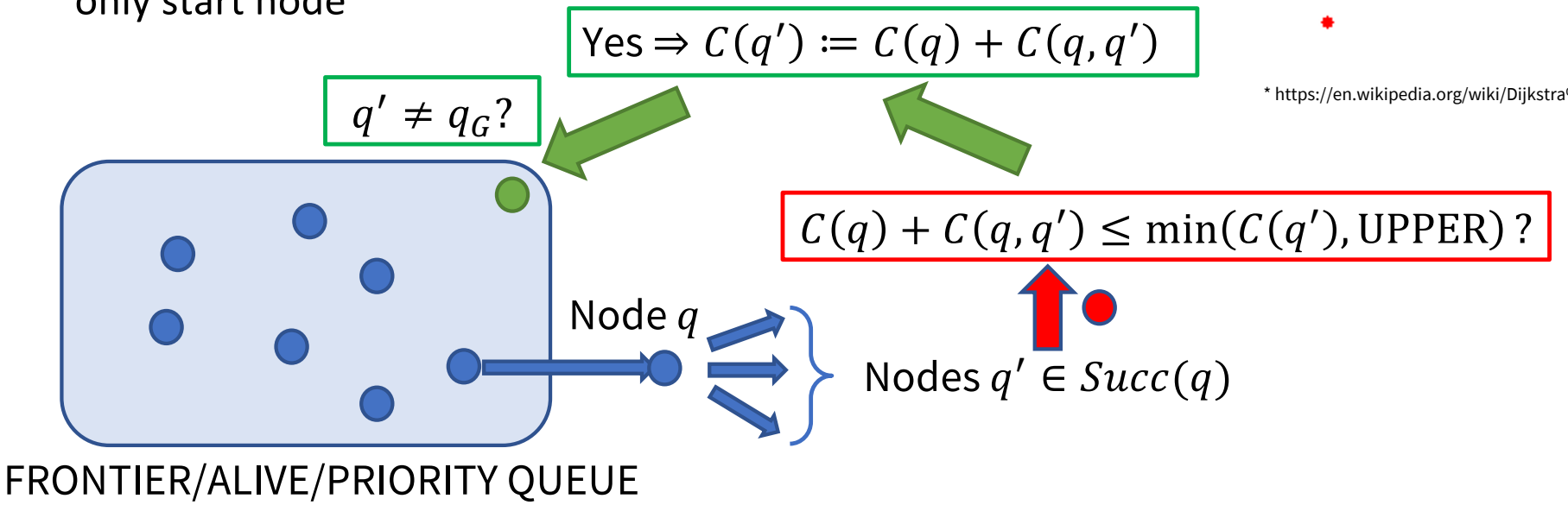


Graph



Graph search algorithms

- Having determined decomposition, how to find "best" path?
- **Label-Correcting Algorithms**: $C(q)$: *cost-to-come* from q_I to q
- Set to ∞ for all nodes, except start node to 0, frontier queue has only start node



* https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Label correcting algorithm

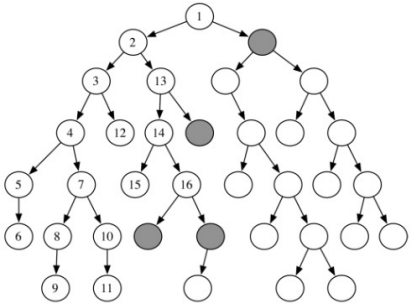
Initialization: set the labels of all nodes to ∞, except for the label of the origin node, which is set to 0, frontier queue has only start node.

Step 1. Remove a node q from frontier queue Q and for each child q' of q, execute step 2

Step 2. If $C(q) + C(q, q') \leq \min(C(q'), \text{UPPER})$, set $C(q') := C(q) + C(q, q')$ and set q to be the parent of q'. In addition, if $q' \neq q_G$, place q' in the frontier queue if it is not already there, while if $q' = q_G$, set UPPER to the new value $C(q) + C(q, q_G)$

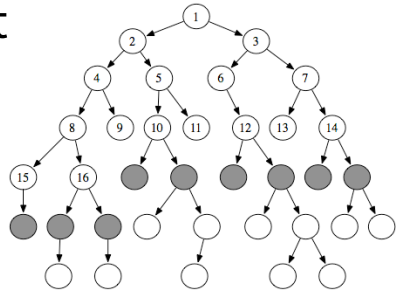
Step 3. If the frontier queue is empty, terminate, else go to step 1

How to prioritize the queue?



Depth-First-Search (DFS): Maintain Q as a **stack** – Last in/first out

- Lower memory requirement (only need to store part of graph)



Breadth-First-Search (BFS, Bellman-Ford): Maintain Q as a **list** – First in/first out

- Update cost for all edges up to current depth before proceeding to greater depth
- Can deal with negative edge (transition) costs

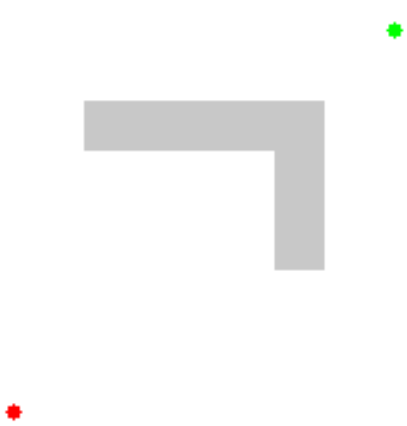
Best-First (BF, Dijkstra): Greedily select next q: $q = \operatorname{argmin}_{q \in Q} C(q)$

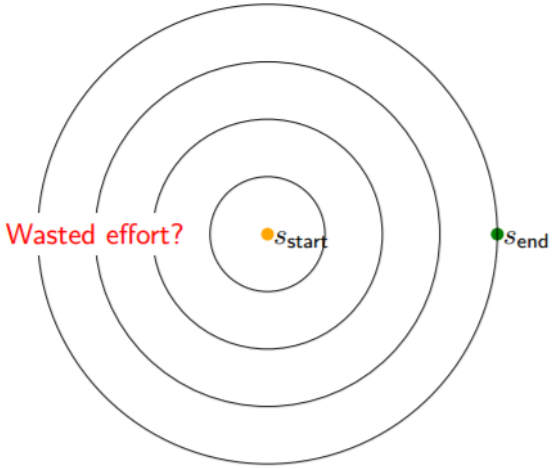
- Node will enter the frontier queue at most *once*
- Requires costs to be non-negative

Correctness and improvements

Theorem

If a feasible path exists from q_I to q_G, then algorithm terminates in finite time with $C(q_G)$ equal to the optimal cost of traversal, $C^*(q_G)$.





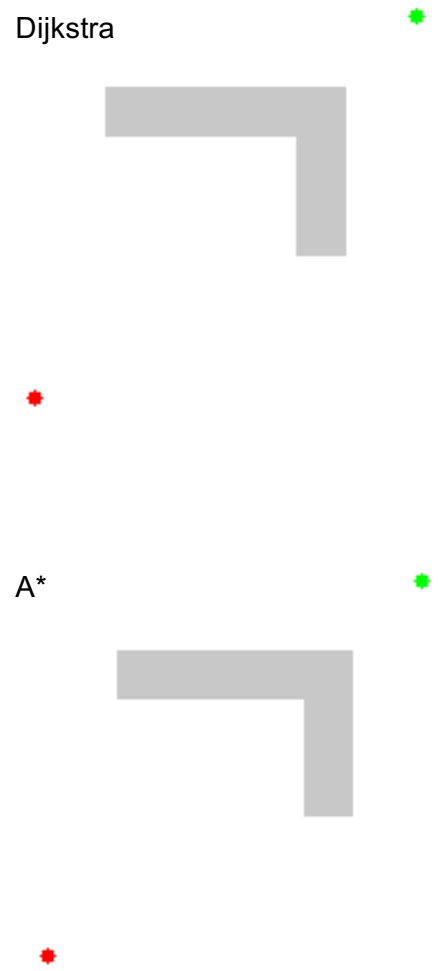
A*: Improving Dijkstra (1968)

- Dijkstra orders by optimal "*cost-to-come*"
- Faster results if order by "*cost-to-come*"+ *(approximate) "cost-to-go"*
- Take the node with lowest C(q) + h(q) from frontier set, where $h(q)$ is *a* heuristic for optimal cost-to-go (specifically, a positive *underestimate*)
- When testing to add neighbor into frontier set, use stronger

$$C(q) + C(q, q') + h(q') \leq \text{UPPER}$$

Instead of $C(q) + C(q, q') \leq \text{UPPER}$

- In this way, fewer nodes will be placed in the frontier queue
- This modification still guarantees that the algorithm will terminate with a shortest path

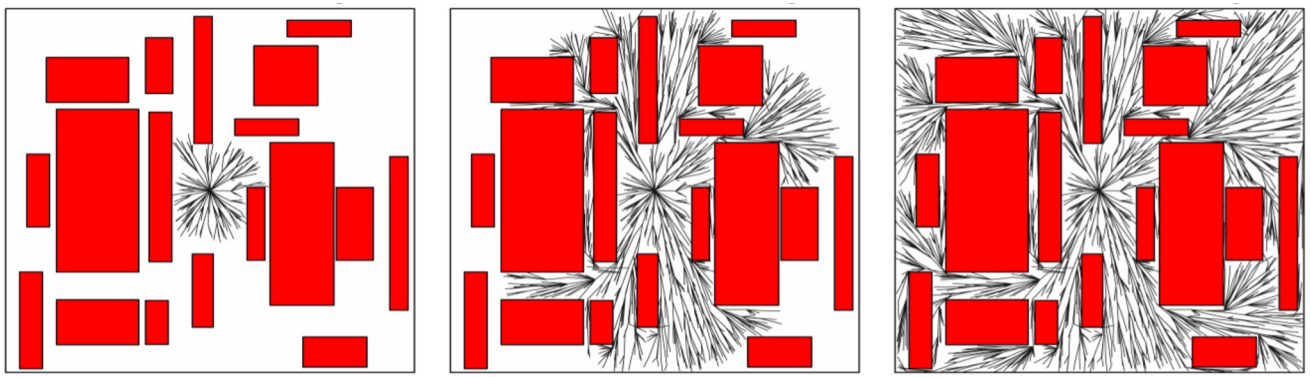


Graph search approaches: summary

- Pros:
  - Simple and easy to use
  - Fast (for some problems)
- Cons:
  - Resolution dependent
    - Not guaranteed to find solution if grid resolution is not small enough
  - Limited to simple robots
    - C-space grid size is exponential in the number of DOFs

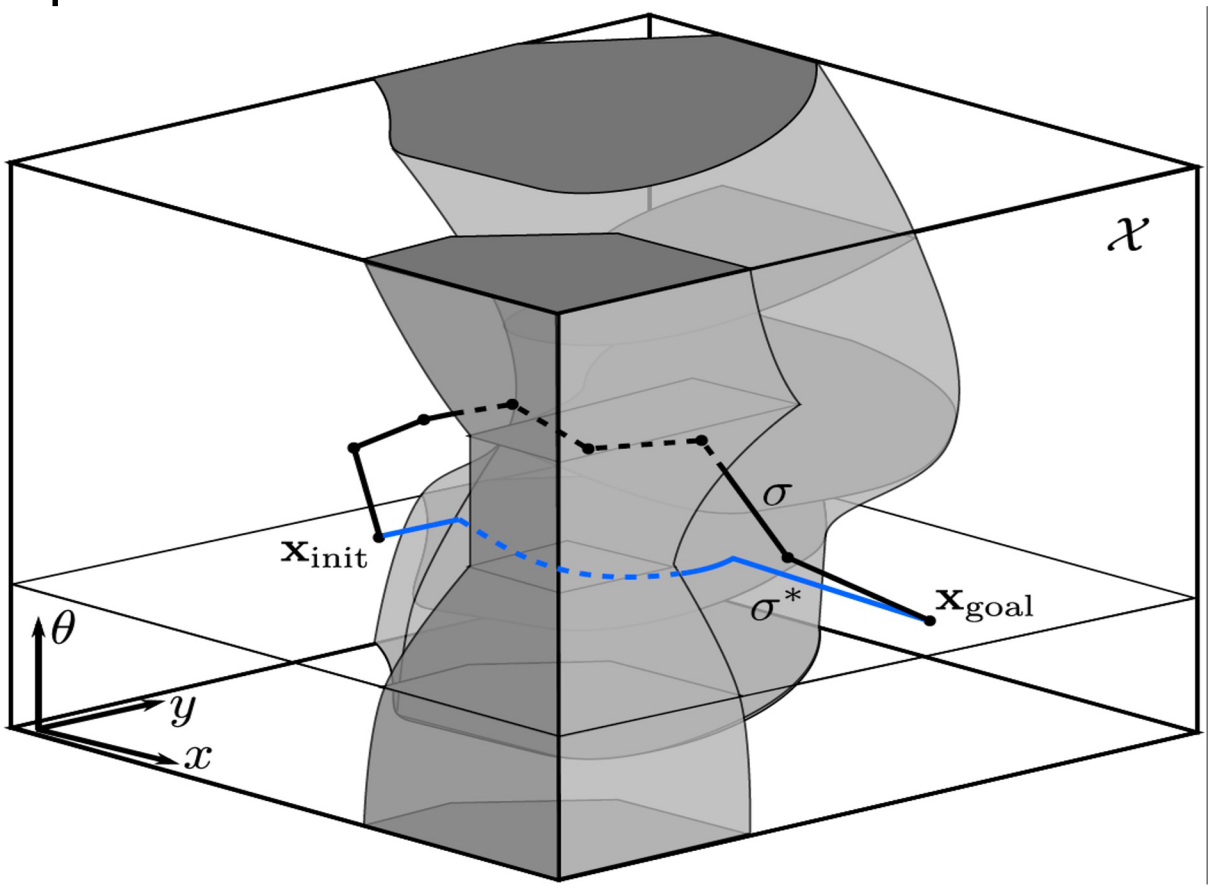
Sampling based planning

Build a **random graph** of pose waypoints and edges between them. Find shortest path in this graph either: after construction (PRM), or during construction (RRT, RRT*).



Karaman, Frizzoli, IJRR 2011, RRT*.

Recall, C-Space



Sampling-based planning

Challenges of explicit C-space computation motivated the development of sampling-based approaches

- Abandon the idea of explicitly characterizing C_{free} and C_{obs}
- Instead, capture the structure of C by random sampling
- Use a black-box component (collision checker) to determine which random configurations lie in C_{free}
- Use such a probing scheme to build a roadmap (a graph of possible paths) and then plan a path

History of sampling-based planning

Traditionally, two major approaches:

- Probabilistic Roadmap (PRM): graph-based
  - Multi-query planner, i.e., designed to solve multiple path queries on the same scenario
  - Original version: [Kavraki et al., ‘96]
  - “Lazy” version: [Bohlin & Kavraki, ‘00]
  - Dynamic version: [Jaillet & T. Simeon, ‘04]
  - Asymptotically optimal version: [Karaman & Frazzoli, '11]
- Rapidly-exploring Random Trees (RRT): tree-based
  - Single-query planner
  - Original version: [LaValle & Kuffner, ‘01]
  - RDT: [LaValle, ‘06]
  - SRT: [Plaku et al., ‘05]
  - Asymptotically optimal version [Karaman & Frazzoli, '11]

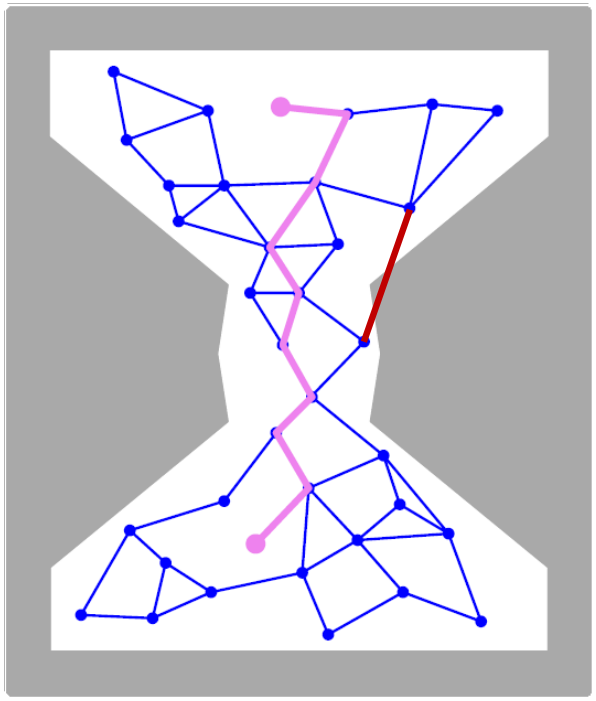
Probabilistic roadmaps (PRM)

A multi-query planner, which generates a roadmap (**graph**) G, embedded in the free space

Preprocessing step:

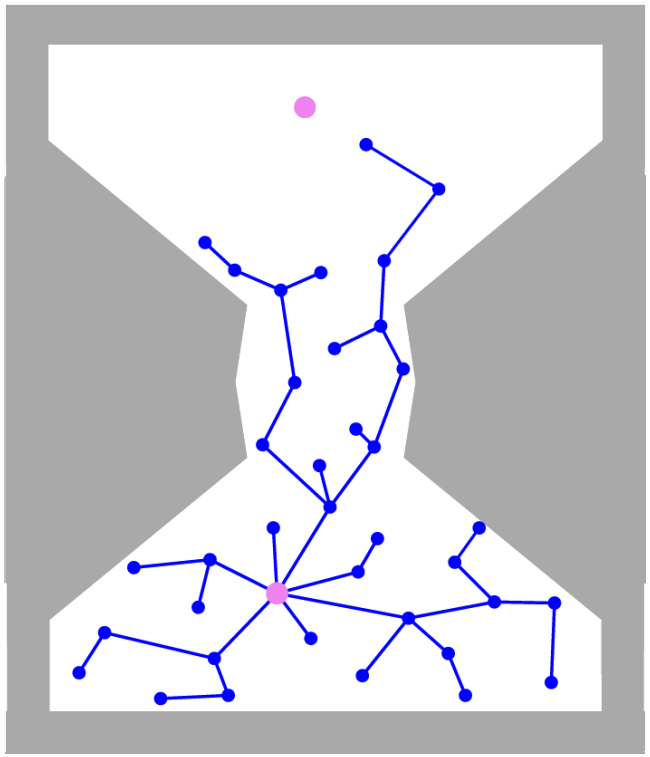
1. Sample a collection of n configurations X_n; discard configurations leading to collisions
2. Draw an edge between each pair of samples $x, x' \in X_n$ such that $\|x - x'\| \leq r$
3. And straight-line path between x and x' is collision free

Given a query $s, t \in C_{free}$, connect them to G and find a path on the roadmap (A*)



Rapidly-exploring random trees (RRT)

A single-query planner, which grows a **tree** $T = (V, E)$, rooted at the start configuration s, embedded in C_{free}



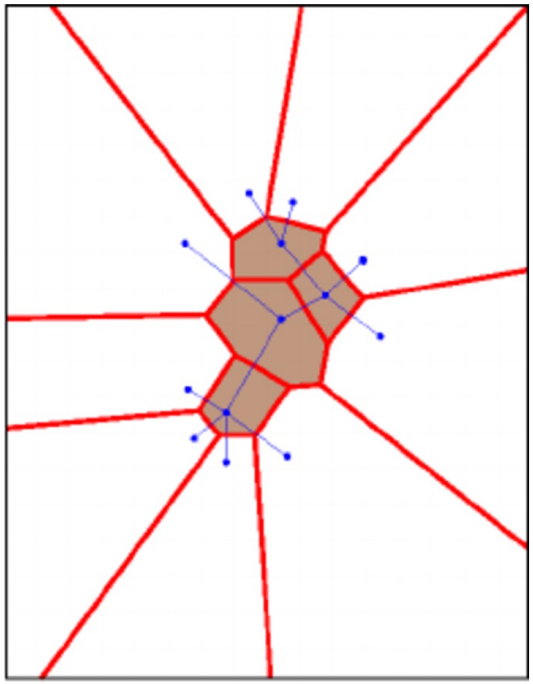
Algorithm works in n iterations:

1. Sample configuration x_{rand}
2. Find nearest vertex x_{near} in T to x_{rand}
3. Generate configuration x_{new} in direction of x_{rand} from x_{near}, such that $\|x_{near} - x_{new}\| \leq \eta$ and $\{x \mid \alpha x_{near} + (1-\alpha)x_{new}, \alpha \in [0,1]\} \in C_{free}$
4. Update tree: $T = (V \cup x_{new}, E \cup (x_{near}, x_{new}))$

Occasionally, set x_{rand} to be the target vertex t; terminate when $x_{new} = t$

Rapidly-exploring random trees (RRT)

- RRT is known to work quite well in practice
- Its performance can be attributed to its Voronoi bias:
  - Consider a Voronoi diagram with respect to the vertices of the tree
  - For each vertex, its Voronoi cell consists of all points that are closer to that vertex than to any other
  - Vertices on the frontier of the tree have larger Voronoi cells – hence sampling in those regions is more likely



Theoretical guarantees: probabilistic completeness

Question: how large should the number of samples n be? We can say something about the asymptotic behavior:

Kavraki et al. 96: For PRM, with r = const, the probability of finding a path if one exists approaches 1 as $n \to \infty$.

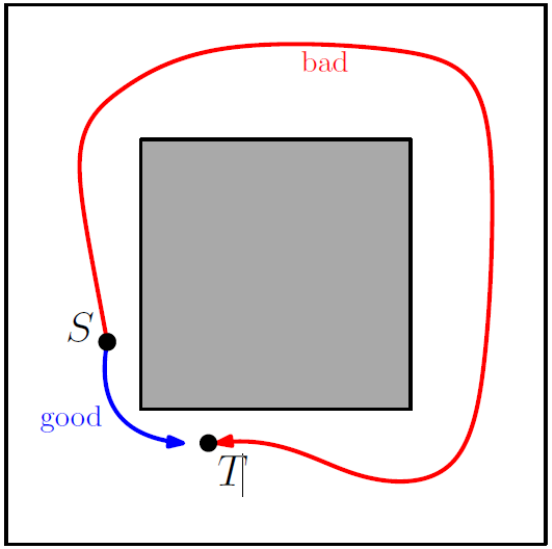
LaValle, 98; Kleinbort et al., 18: For RRT, the probability of finding a path if one exists approaches 1 as $n \to \infty$

* Unless stated otherwise, the configuration space is assumed to be the d-dimensional Euclidean unit hypercube $[0,1]^d$, with $2 \leq d < \infty$

Theoretical guarantees: quality

Question: what can be said about the quality of the returned solution for PRM and RRT, in terms of length, energy, etc.?

- Nechushtan et al. (2011) and Karaman and Frazzoli (2011) proved that RRT can produce arbitrarily-bad paths with non-negligible probability: for example, RRT might prefer to take the long (red) way.
- This is because it does not **revise** connections with more efficient choices that might be found later (recall Dijkstra's, A*).

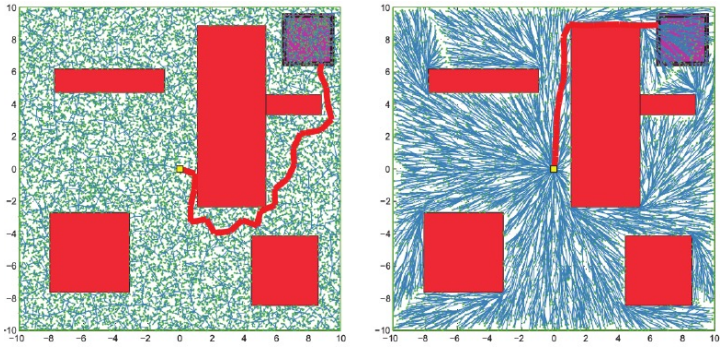


Theoretical guarantees: quality

Karaman and Frazzoli in 2011 provided the first rigorous study of optimality in sampling-based planners:

Theorem: The cost of the solution returned by PRM converges, as $n \to \infty$, to the optimum, when $r_n = \gamma \left(\frac{\log n}{n}\right)^{\frac{1}{d}}$, where γ only depends on d

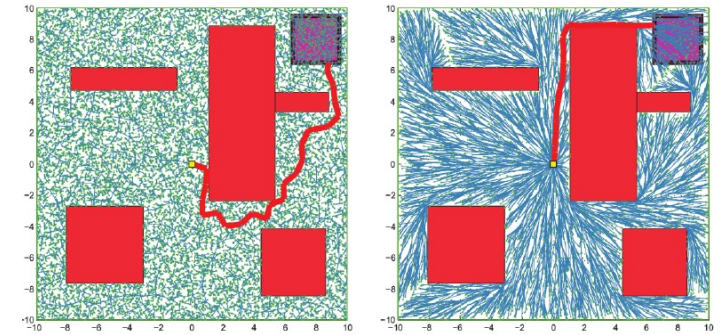
- KF11 also introduced an asymptotically optimal variant of RRT called RRT* (right)
- Introduced **re-wiring,** another name for label correcting, just like Dijkstra's/A*.

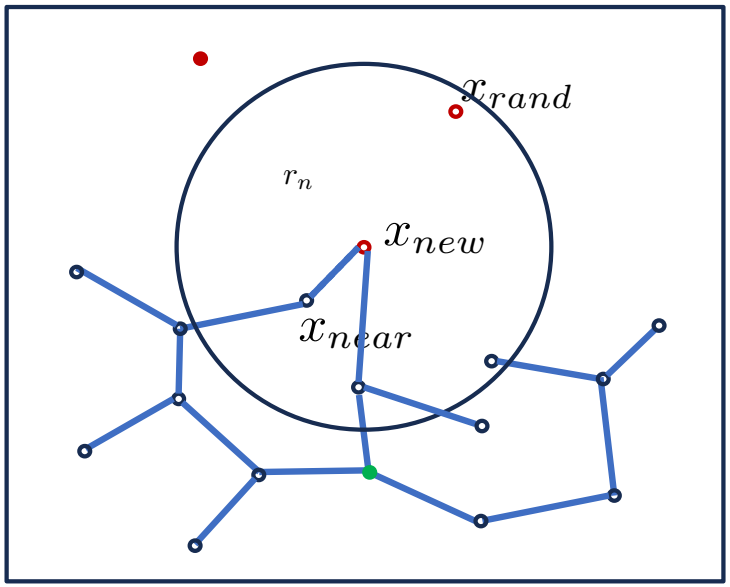


RRT* Re-wiring

- After connecting vertex x_{new} to the tree, check alternative connections to all potential parents in ball radius r_n
- If a lower cost path found through different parent, add edge to new parent, delete edge to old parent
- If a lower cost path found through x_{new} as parent, add new/delete old
- r_n shrinks as n grows, according to

$$r_n = \gamma \left(\frac{\log n}{n}\right)^{\frac{1}{d}}$$





Observations

- PRM-like motion planning algorithms
  - For a given number of nodes n, they find “good” paths
  - ...however, require many costly collision checks

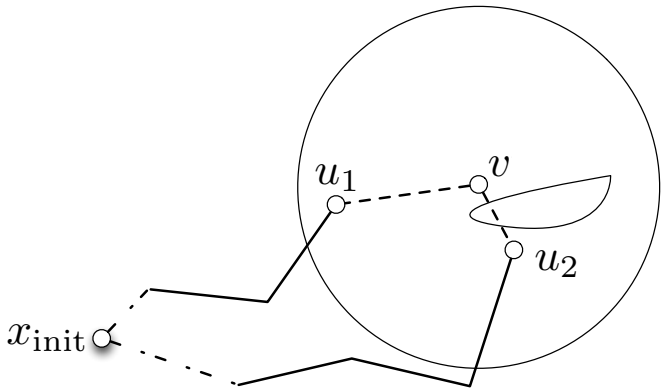
- RRT-like motion planning algorithms
  - Finds a feasible path quickly
  - ...however, the quality of that path is, in general, poor
  - “traps” itself by disallowing new better paths to emerge - RRT* performs local label correction as samples are added to help remedy this

Fast Marching Tree Algorithm (FMT*)

- Pre-sample nodes, do not connect them yet
- Key idea: run dynamic programming (DP) on sampled nodes, skipping any step in which the attempted connection causes a collision
  - lazy DP operator:

$$c(v) = \min_{u:\|u-v\|<r_n} \text{Cost}(u,v) + c(u)$$

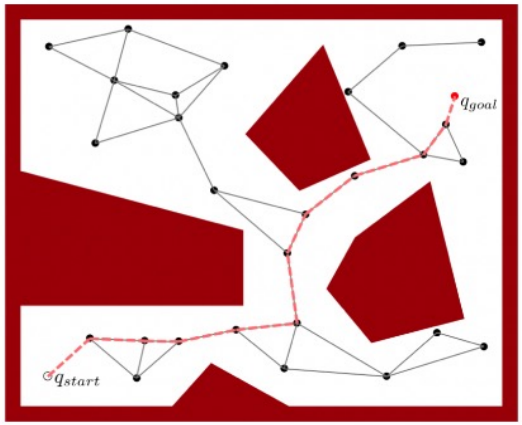
- Laziness introduces "suboptimal" connections, but such connections are vanishingly rare and FMT* is asymptotically optimal
- Ratio of # of collision-checks for FMT* versus PRM* goes to zero!



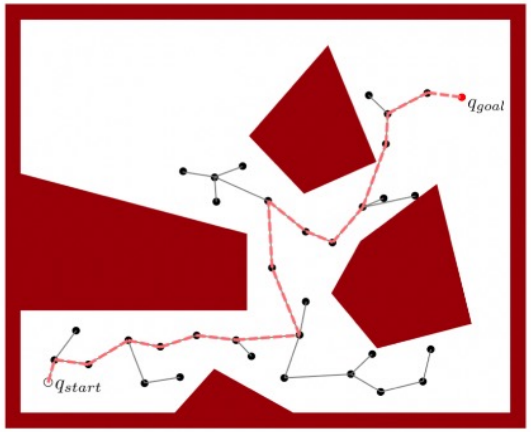
Reference: Janson et al. Fast Marching Tree: A Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions. 2015

Which of these two roadmaps has been created by RRT?

(A)



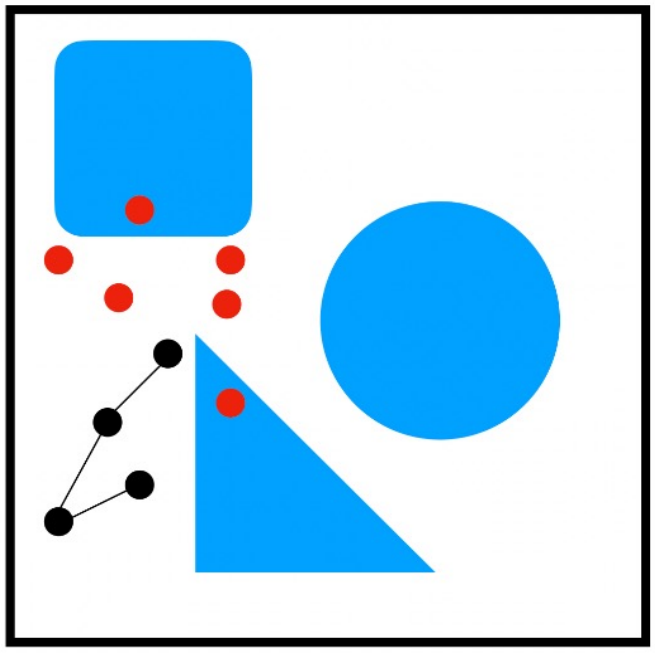
(B)



(A)

(B)

How many of the red sample points could lead to a new node in the roadmap constructed by RRT?



0

1

2

3

4

5

6

Details and notes:

- Sampling-based planners transform the difficult global problem into a large set of local and easy problems
- A key ingredient is collision checking, which can be computationally time consuming. Speed ups avoid collision checking
- Local planning (edge validation) is typically performed by dense sampling of path and collision detection
- Another key ingredient is nearest-neighbor search (best done with k-d tree): given a query point find its nearest neighbor(s) within a set of points -- also well studied theoretically and practically

Sampling-based planning

Pros:

- Conceptually simple
- Relatively-easy to implement
- Flexible: one algorithm applies to a variety of robots and problems
- Beyond the geometric case: can cope with complex differential constraints, uncertainty, etc.

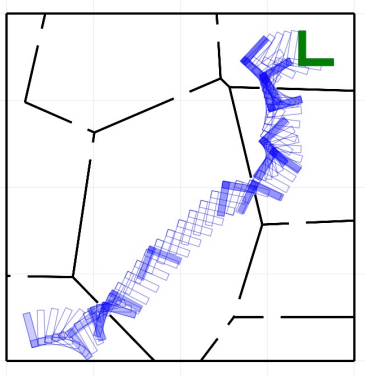
Cons:

- Asymptotic
- Unclear how many samples should be generated to retrieve a solution
- Cannot determine whether a solution does not exist

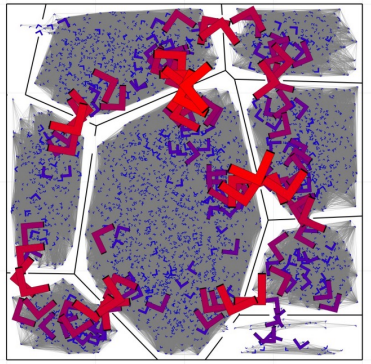
Biased sampling for SBMP

- Potential issue with uniform sampling: narrow corridors in C-space require many samples to identify/traverse
- Key idea: bias sampling towards suspected challenging regions of C-space
- Biased sampling distributions can be hand-constructed and/or adapt online (e.g., Hybrid Sampling PRM)
- Or learned from prior experience solving similar planning problems

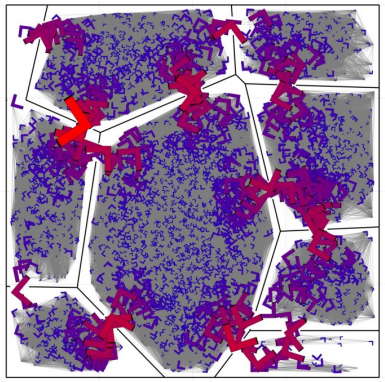
References:
Hsu et al. Hybrid PRM sampling with a cost-sensitive adaptive strategy. 2005.
Ichter et al. Learned Critical Probabilistic Roadmaps for Robotic Motion Planning. 2020



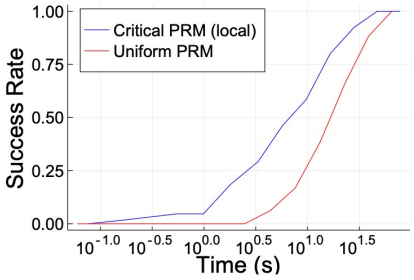
(a) SE(2) Planning



(b) Ground Truth Criticality



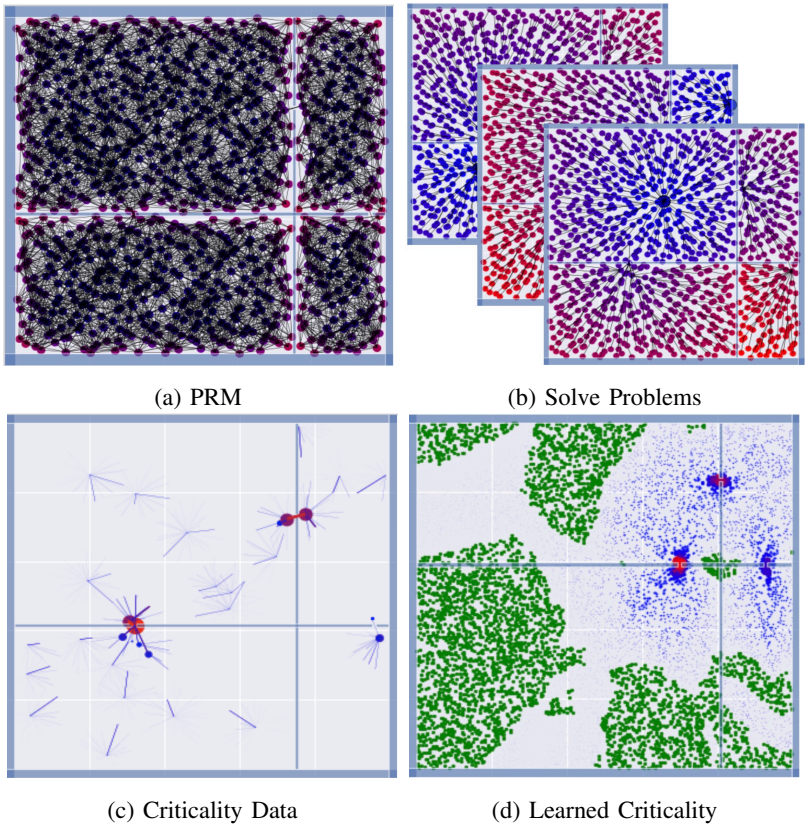
(c) Learned Criticality



(d) Time (s) vs. Success

Learning criticality from local map features

- On a collection of different maps, solve for optimal PRM paths between multiple start-goal pairs
- Score criticality of all nodes based on number of optimal paths passing through node
- Discount if node can be removed without collision
- Train CNN from local map patch centered at point to criticality of point



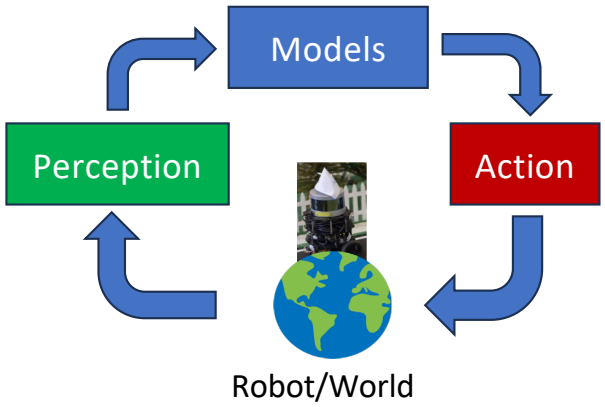
(a) PRM (b) Solve Problems

(c) Criticality Data (d) Learned Criticality

Ichter et al. Learned Critical Probabilistic Roadmaps for Robotic Motion Planning. 2020

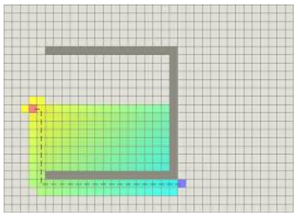
Action: Planning and Control Stack

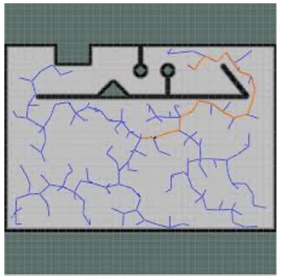
Use the models to move the robot



Path planning:

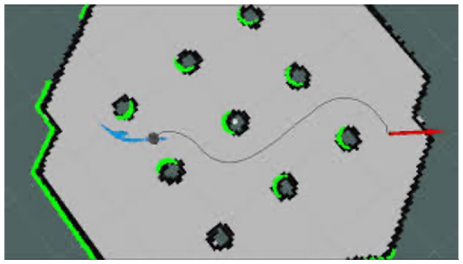
- A* planning on a graph
- RRT, RRT*





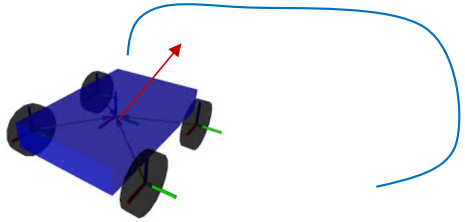
Trajectory optimization:

- Optimal control
- Numerical solvers



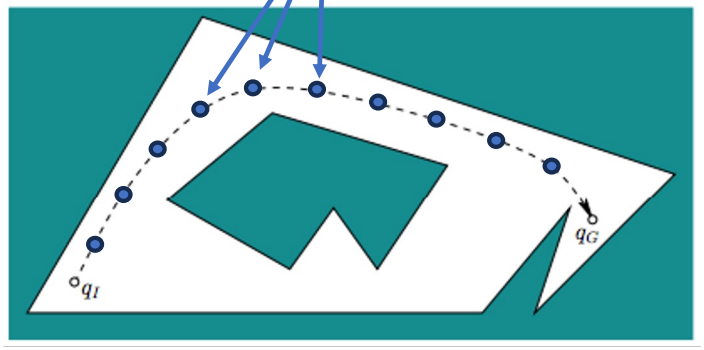
Trajectory following:

- Kinematic/dynamic models
- PID control
- LQR, gain scheduled LQR



Trajectory Optimization

Waypoint poses from path planner
(e.g. A*, RRT*)



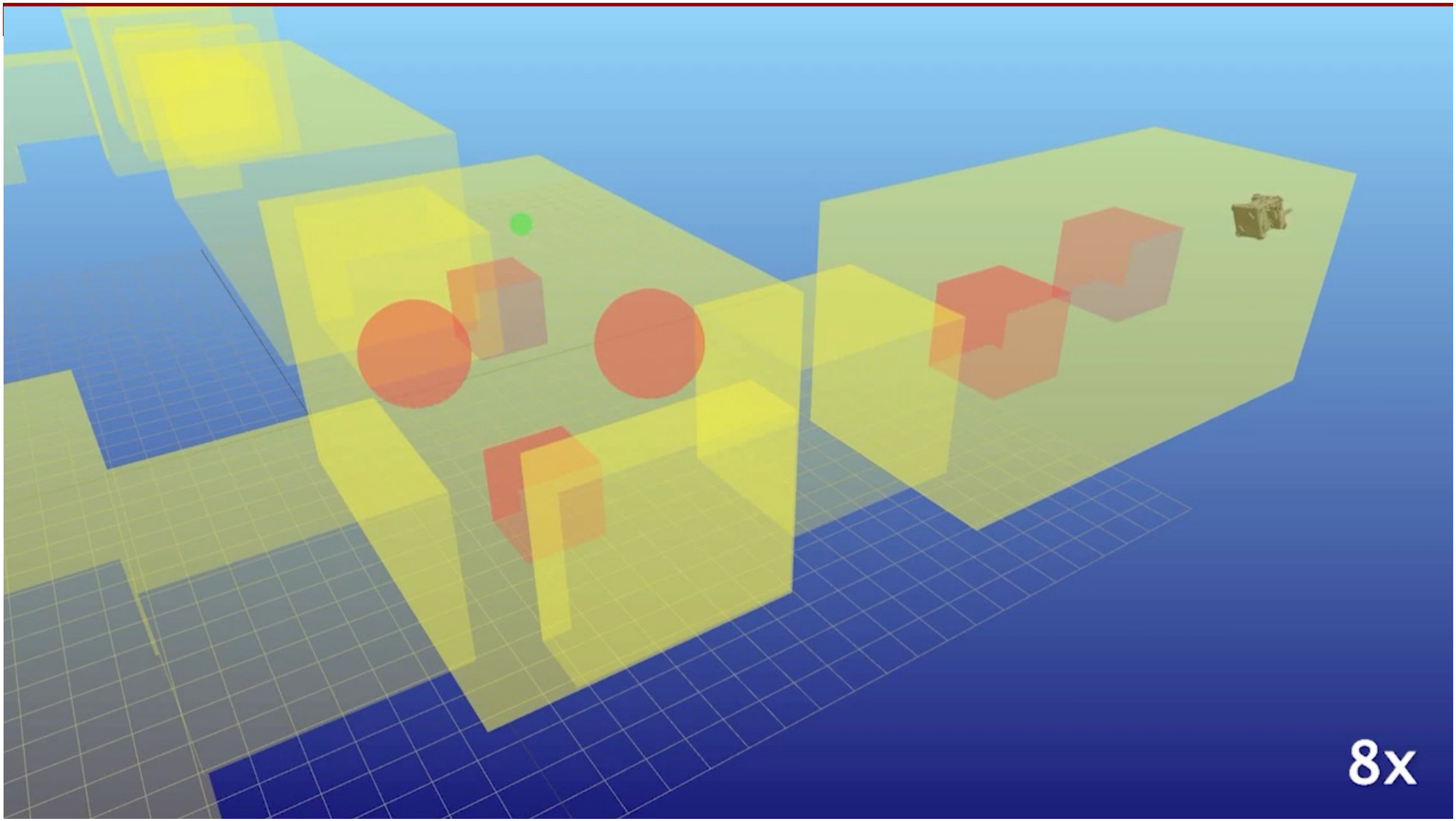
Trajectory optimization problem:

$$\min_{u_{0:T}} \sum_{t=0}^{T} \boxed{c_t(x_t, u_t)} \quad \text{Traj. cost}$$

$$\text{s.t.} \quad \boxed{x_{t+1} = f(x_t, u_t)} \quad \text{Dynamic constraints}$$

$$\boxed{x_t \in \mathbb{X}_{\text{Free}}} \quad \text{Collision constraints}$$

Goal: Find a dynamically feasible trajectory (close to the path) so the robot doesn't hit things!


8x

Kinodynamic motion planning for vehicle high speed maneuvering

- Yanbo Li

Admissible Velocity Propagation : Beyond Quasi-Static Path Planning for High-Dimensional Robots

Quang-Cuong Pham, Stéphane Caron, Puttichai Lertkultanon, Yoshihiko Nakamura. IJRR 2017.

Kinodynamic sampling based traj opt

- https://github.com/rikkimelissa/baxter_throw
- Rikki Valverde, Northwestern University

Optimal control problem: traj opt and feedback control combined

The problem:

$$\begin{aligned} \min_{\mathbf{u}} \quad & h(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t)\, dt \\ \text{subject to} \quad & \dot{\mathbf{x}}(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t) \\ & \mathbf{x}(t) \in \mathcal{X}, \quad \mathbf{u}(t) \in \mathcal{U} \end{aligned}$$

where $\boldsymbol{x}(t) \in R^n, \boldsymbol{u}(t) \in R^m$, and $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$

- Good reference: D. K. Kirk. Optimal Control Theory: An introduction. 2004.

Form of optimal control

- **Closed loop:** If a functional relationship can be found in form

$$\mathbf{u}^*(t) = \pi(\mathbf{x}(t), t)$$

- **Open loop:** If the optimal control law is determined as a function of time for a specified initial state value

$$\mathbf{u}^*(t) = \mathbf{f}(\mathbf{x}(t_0), t)$$

- A good compromise: two-step design

$$\mathbf{u}^*(t) = \mathbf{u}_d(t) + \pi(\mathbf{x}(t), \mathbf{x}(t) - \mathbf{x}_d(t))$$

Reference trajectory

Reference control (open-loop), from traj opt

Trajectory-tracking law (closed-loop), from feedback control

Tracking error

Open-loop control

- We want to find

$$\mathbf{u}^*(t) = \mathbf{f}(\mathbf{x}(t_0), t)$$

- In general, two broad classes of methods:

  1. Direct methods: transcribe infinite problem into finite dimensional, nonlinear programming (NLP) problem, and solve NLP ⇒ "First discretize, then optimize"
  2. Indirect methods: attempt to find a minimum point "indirectly," by solving the necessary conditions of optimality ⇒ "First optimize, then discretize"

- For an in-depth study of direct and indirect methods, see AA203 "Optimal and Learning-based Control"

Extra slides

Should probabilistic planners be probabilistic?

Key question: would theoretical guarantees and practical performance still hold if these algorithms were to be de-randomized, i.e., run on deterministic samples?

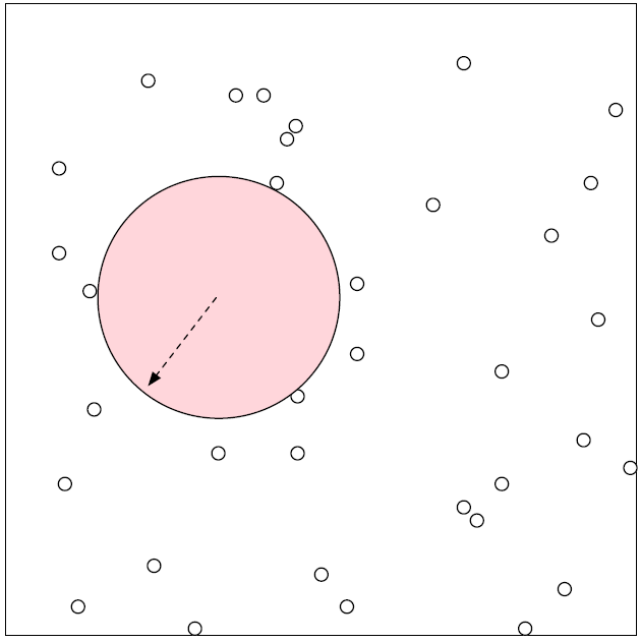
Important question as de-randomization would:

- Ease certification process
- Ease use of offline computation
- Potentially simplify several operations (e.g., NN search)

Designing “good” sequences

ℓ_2-dispersion: For a finite set S of points contained in $X \subset \mathbb{R}^d$, its ℓ_2-dispersion $D(S)$ is defined as

$$D(S) \coloneqq \sup_{x \in X} \min_{s \in S} \|s - x\|_2$$

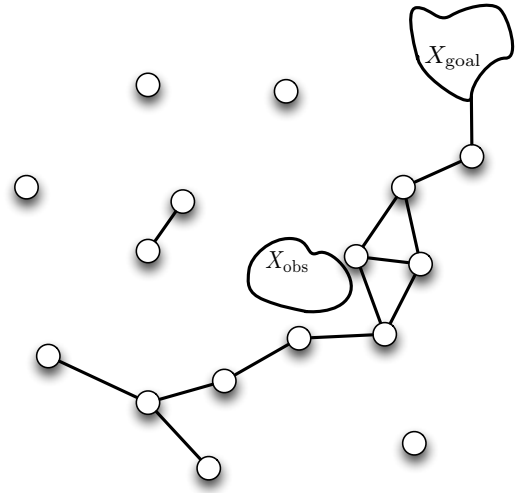


Key facts:

- There exist deterministic sequences with $D(S)$ of order $O\left(n^{-1/d}\right)$, referred to as **low-dispersion** sequences
- Sequences minimizing ℓ_2-dispersion only known for $d = 2$

Optimality of deterministic planning

```
1  V ← {x_init} ∪ SampleFree(n); E ← ∅
2  for all v ∈ V do
3      X_near ← Near(V\{v}, v, r_n)
4      for x ∈ X_near do
5          if CollisionFree(v, x) then
6              E ← E ∪ {(v, x)} ∪ {(x, v)}
7          end if
8      end for
9  end for
10 return ShortestPath(x_init, V, E)
```



Optimality: Let c_n denote the arc length of the path returned with n samples. Then if

1. Samples set S has dispersion $D(S) \leq \gamma n^{-1/d}$ for some $\gamma > 0$,
2. $n^{1/d} r_n \to \infty$,

then $\lim_{n\to\infty} c_n = c^*$, where c^* is the cost of an optimal path

Deterministic sampling-based motion planning

- Asymptotic optimality can be achieved with deterministic sequences and with a smaller connection radius
- Deterministic convergence rates: instrumental to the certification of sampling-based planners
- Computational and space complexity: under some assumptions, arbitrarily close to theoretical lower bound
- Deterministic sequences appear to provide superior performance

Reference: Janson et al. Deterministic Sampling-Based Motion Planning: Optimality, Complexity, and Performance. 2018