Planning on Roadmaps

Instructor: Chris Mavrogiannis

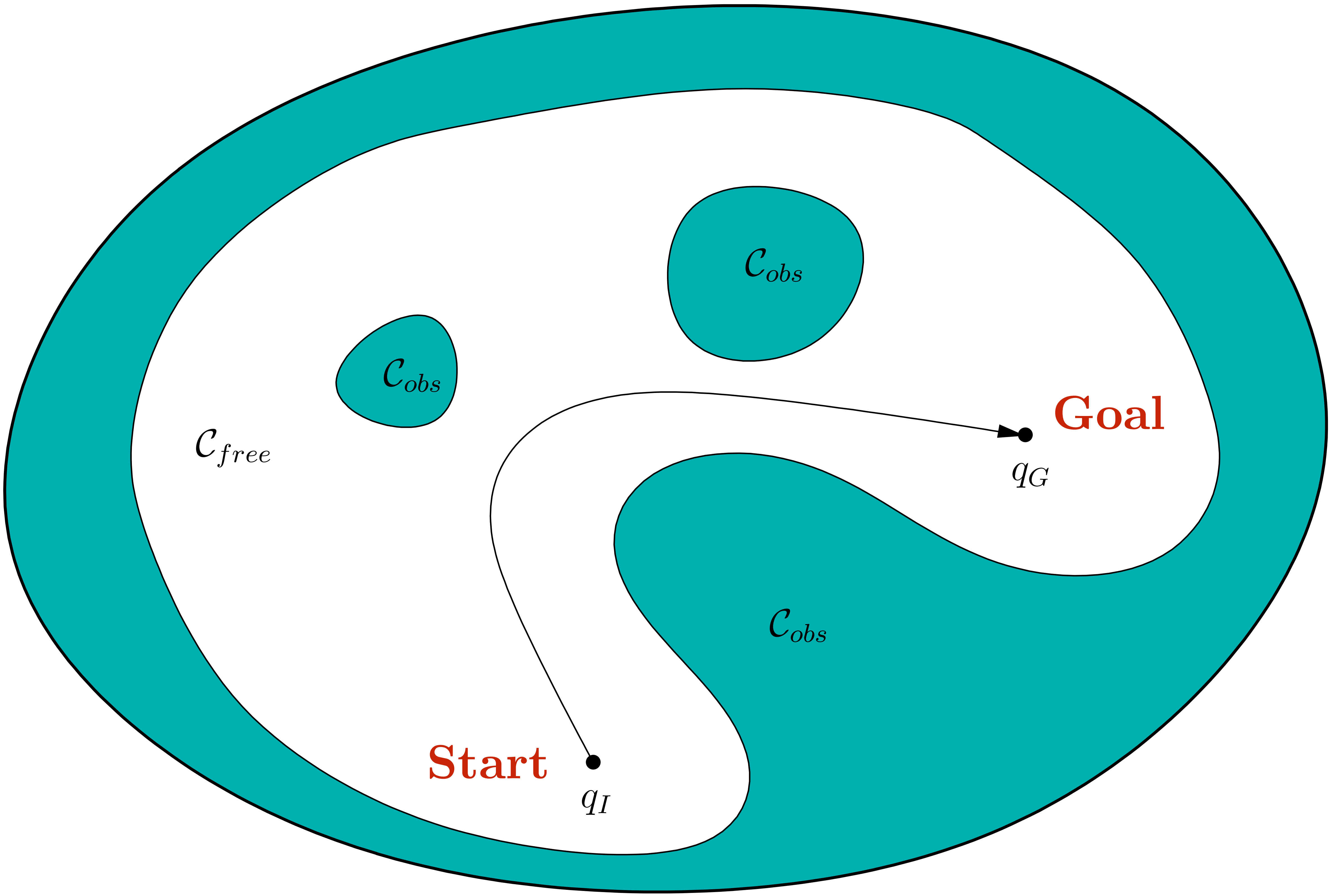
TAs: Kay Ke, Gilwoo Lee, Matt Schmittle

*Slides based on or adapted from Sanjiban Choudhury, Steve Lavalle

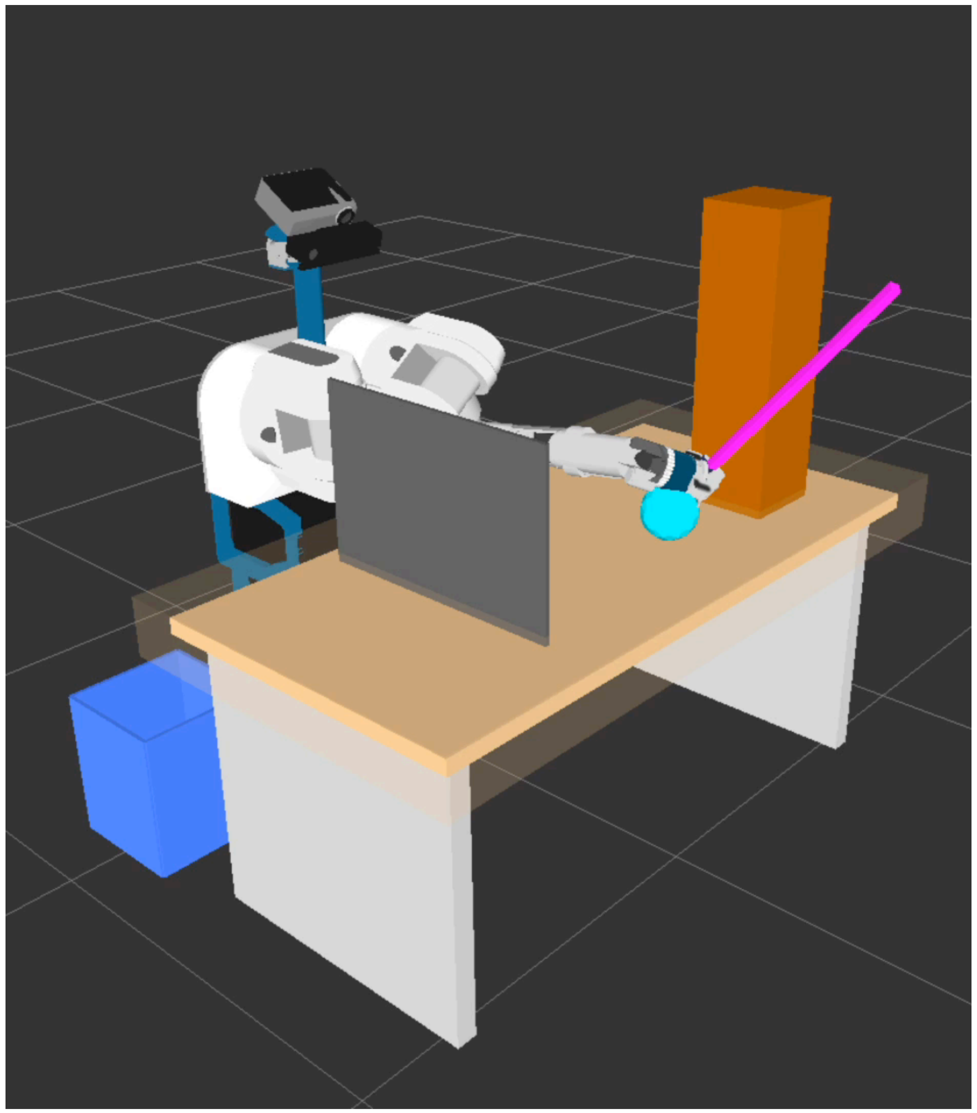
Logistics

- Lab 3
  - Deadline Friday March 6th
  - Demo Thursday March 5th (recitation slots)
  - **Extra Credit important for final project**
- Final Project
  - Out this weekend
  - Demo Thursday March 12th
  - Short writeup due Monday 16th
- Guest lecture Friday
  - Prof. Sidd Srinivasa
  - Lazy Search
  - **Lazy Search is part of Lab 3!**

Piano Mover's Problem



Geometric Path Planning Problem



Also known as

Piano Mover's Problem (Reif 79)

Given:

1. A *workspace* $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An *obstacle region* $\mathcal{O} \subset \mathcal{W}$.
3. A *robot* defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of m links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The *configuration space* $\mathcal{C}$ (C_{obs} and C_{free} are then defined).
5. An *initial configuration* $\boldsymbol{q_I} \in \mathcal{C}_{free}$.
6. A *goal configuration* $\boldsymbol{q_G} \in \mathcal{C}_{free}$. The initial and goal configuration are often called a *query* $(\boldsymbol{q_I}, \boldsymbol{q_G})$.

Compute a (continuous) path, $\tau : [0,1] \rightarrow \mathcal{C}_{free}$, *such that* $\tau(0) = \boldsymbol{q_I}$ *and* $\tau(1) = \boldsymbol{q_G}$.

Also may want to minimize cost $c(\tau)$

But I just want to know how to plan for my racecar!

Alright, let's look at differential constraints!

Differential constraints

So far we assumed only kinematic constraints

$$q \notin \mathcal{C}_{obs}$$

When is this assumption true?

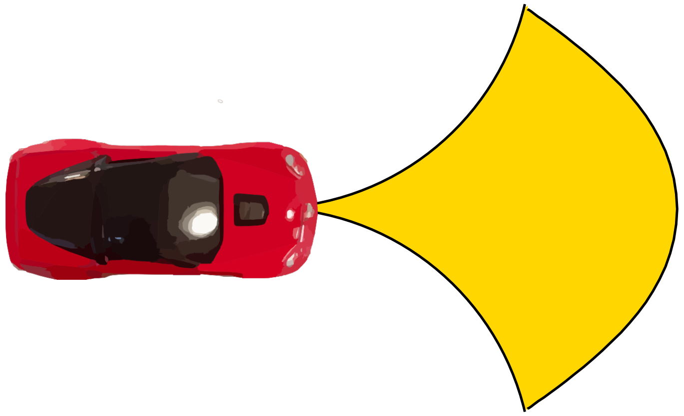
- when controller can track any path

- when robots move very slowly (stop and turn)

Differential constraints

We now introduce differential constraints

$$\dot{q} = f(q, u)$$



Two new terms:

1. Introduction of control space
2. Introduction of an equality constraint

Motion planning under differential constraints

1. Given world, obstacles, C-space, robot geometry (same)

2. Introduce state space X. Compute free and obstacle state space.

3. Given an action space U

4. Given a state transition equations $\dot{q} = f(q, u)$

5. Given initial and final state, cost function

$$J(q(t), u(t)) = \int c(q(t), u(t))dt$$

6. Compute action trajectory that satisfies boundary conditions, stays in free state space and minimizes cost.

Differential constraints make things even harder

Holonomic constraint: Can be expressed as an equation involving only system coordinates and possibly time.

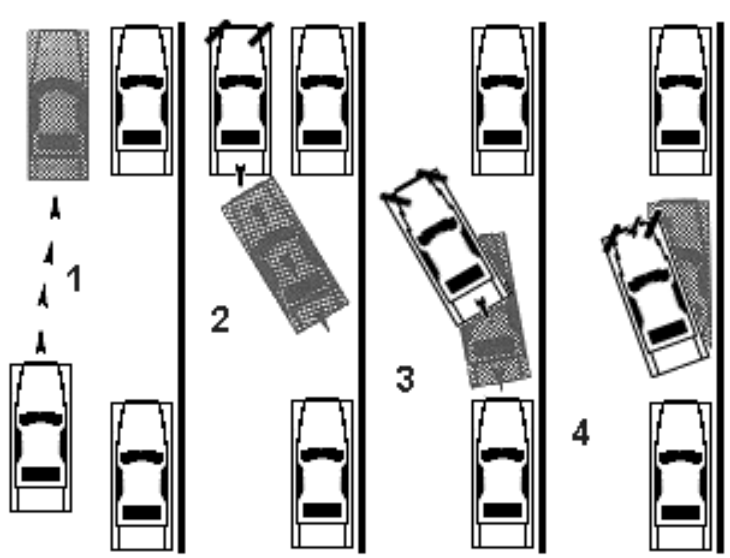
$$\phi(\mathbf{q}, \dot{\mathbf{q}}, t) = 0 \longrightarrow \phi(\mathbf{q}, t) = 0$$

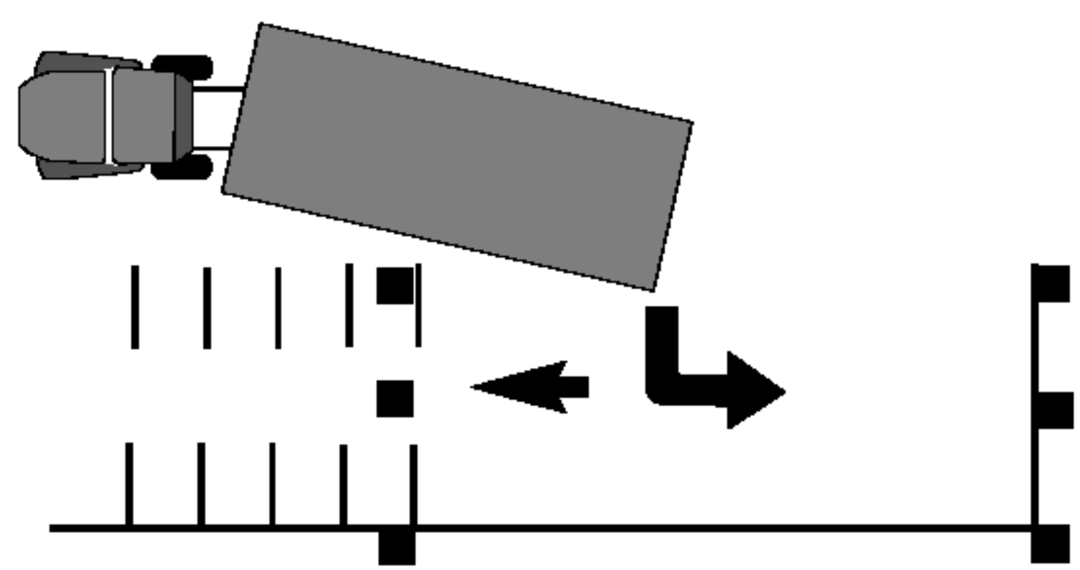
Reduces # of DoFs by one

Nonholonomic constraint: A constraint that is *not* holonomic.

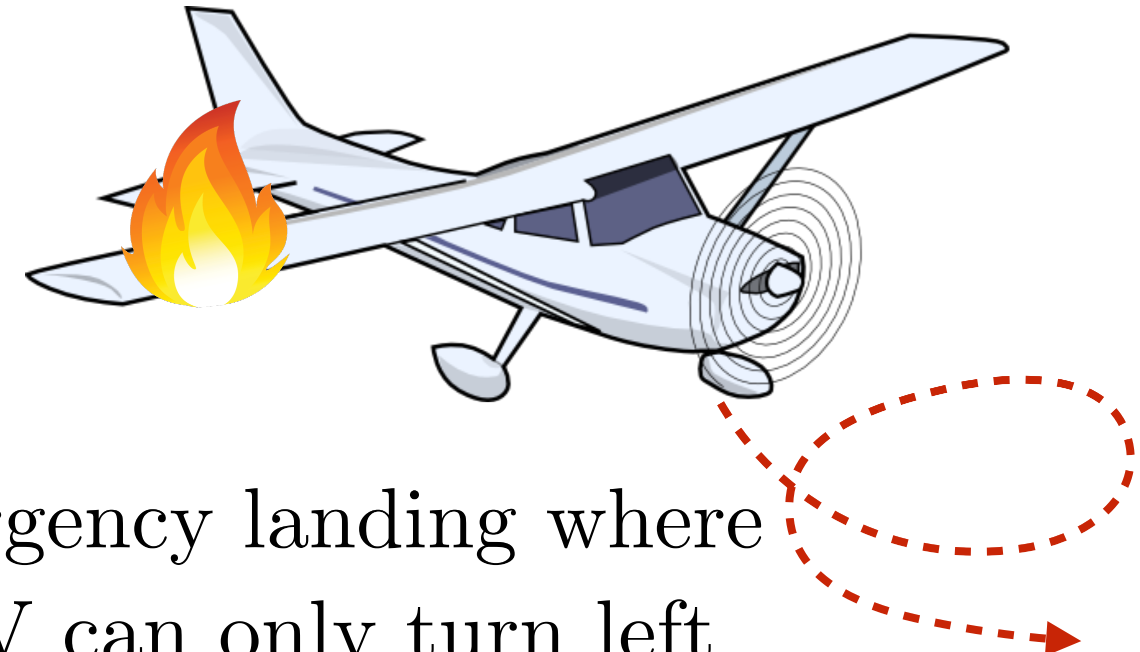
$$\phi(\mathbf{q}, \dot{\mathbf{q}}, t) = 0$$

Constrains the way a configuration can be reached.

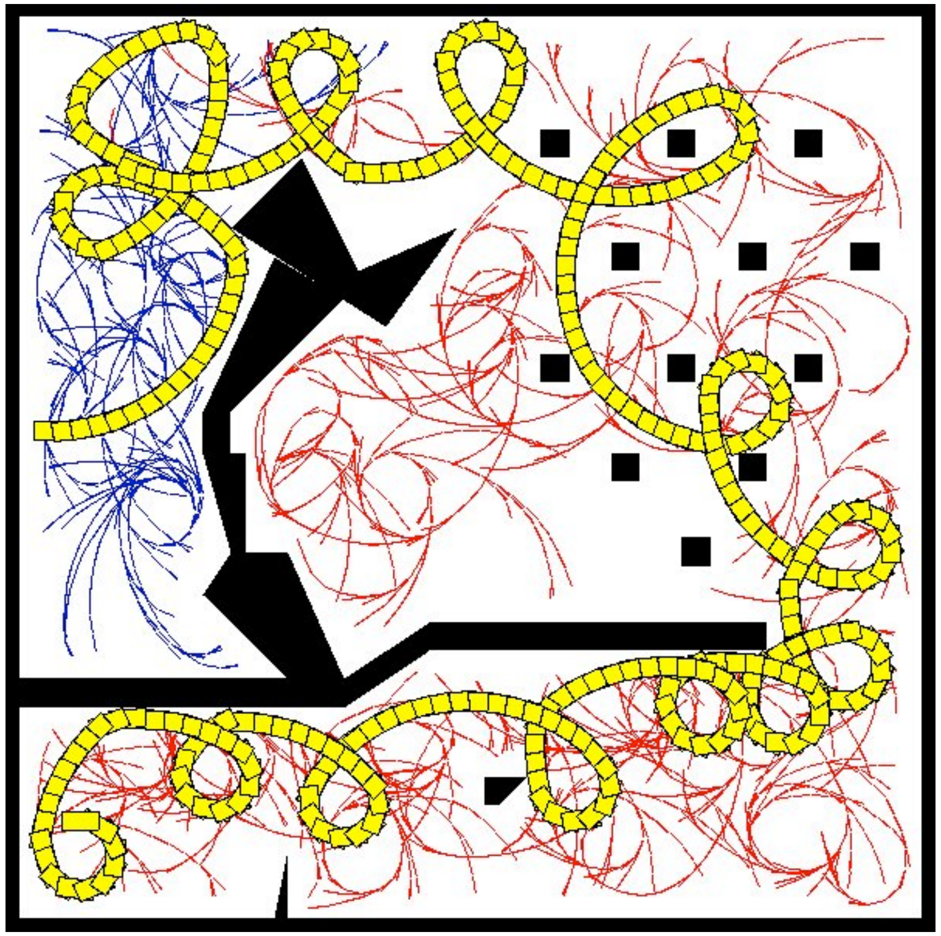




Differential constraints make things harder



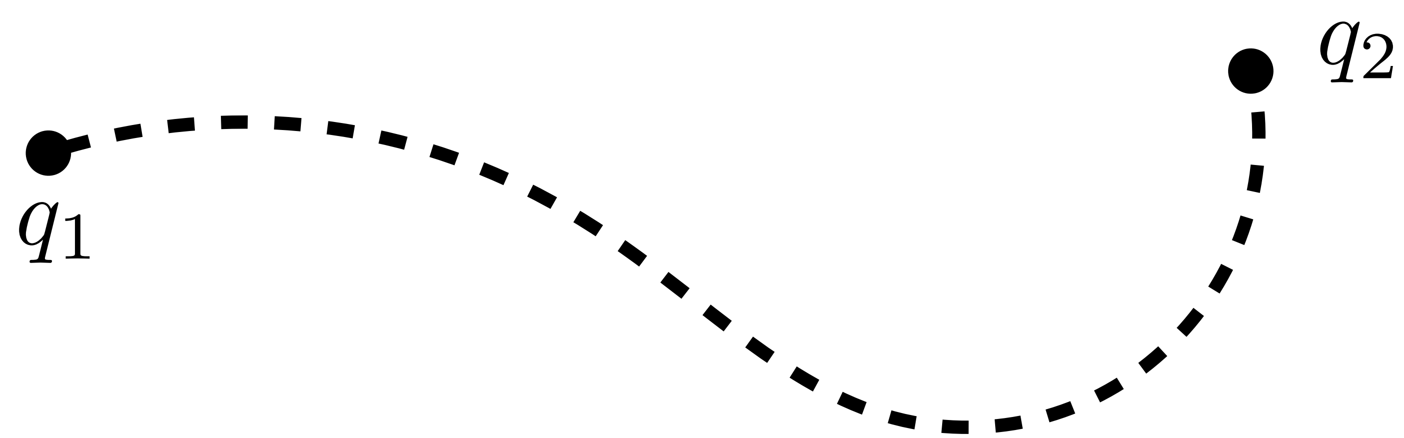
Emergency landing where UAV can only turn left



"Left-turning-car"

Nonholonomic constraint: state depends on path taken to achieve it; expression constrains time derivatives of configuration.
(system is trapped in some sub-manifold of the configuration space)

Plan that incorporates differential constraints



Formally called the boundary value problem (BVP)

Find a control trajectory $u(t) \in U$

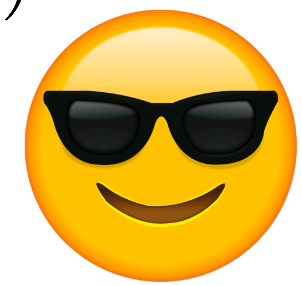
Such that $q(0) = q_1$, $q(t_f) = q_2$

$$\dot{q}(t) = f(q(t), u(t))$$

How do we solve the BVP?

There are three possible cases

Case 1: We can analytically solve the BVP :)

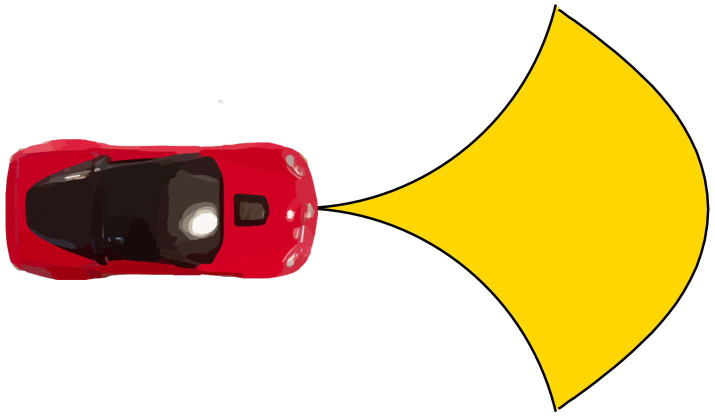


Case 2: We need to numerically solve the BVP.

Case 3: We can't even solve the BVP!

Case 1: We can analytically solve the BVP :)

Consider the dynamics of your racecar



$$\begin{bmatrix} \dot{x} \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} V\cos\theta \\ V\sin\theta \\ C\tan\delta \end{bmatrix}$$

$$|\delta| \leq \delta_{max}$$

$$q_1 = (x_1, y_1, \theta_1)$$

$$q_2 = (x_2, y_2, \theta_2)$$

Solution is called the Dubins Path

Dubins Path: Shortest curve connecting two points in $\mathbb{R}^2$ satisfying:

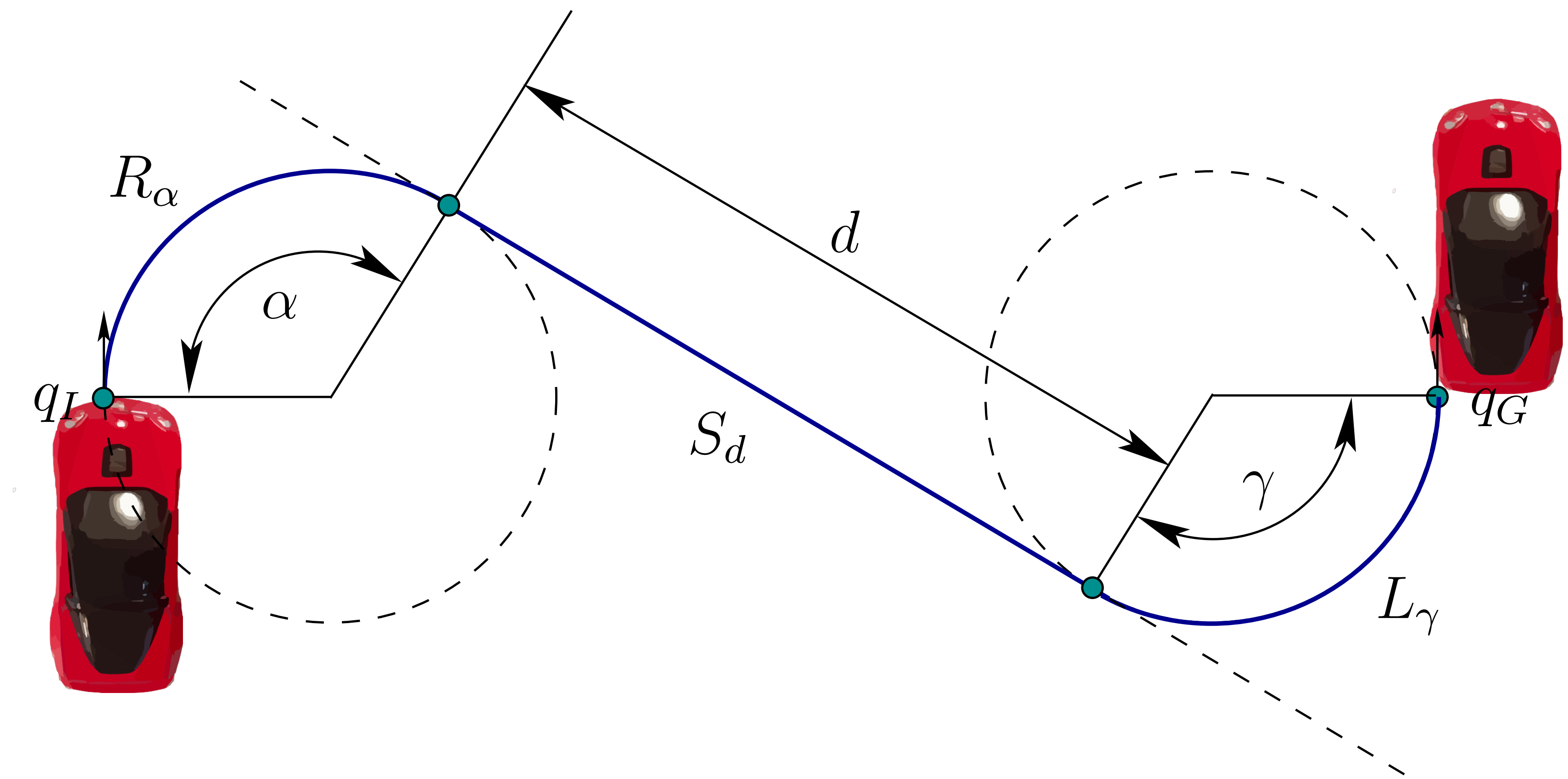
- A maximum path curvature constraint
- Prescribed initial and terminal tangents to path
- Vehicle can only travel forward

Dubins showed that if a solution exists, the shortest path comprises only **maximum-curvature** (L, R) and/or **straight-line** (S) segments.

There are 6 classes of possibly optimal paths

$$\{LRL,\ RLR,\ LSL,\ LSR,\ RSL,\ RSR\}.$$

Dubins Path



$$R_\alpha S_d L_\gamma$$

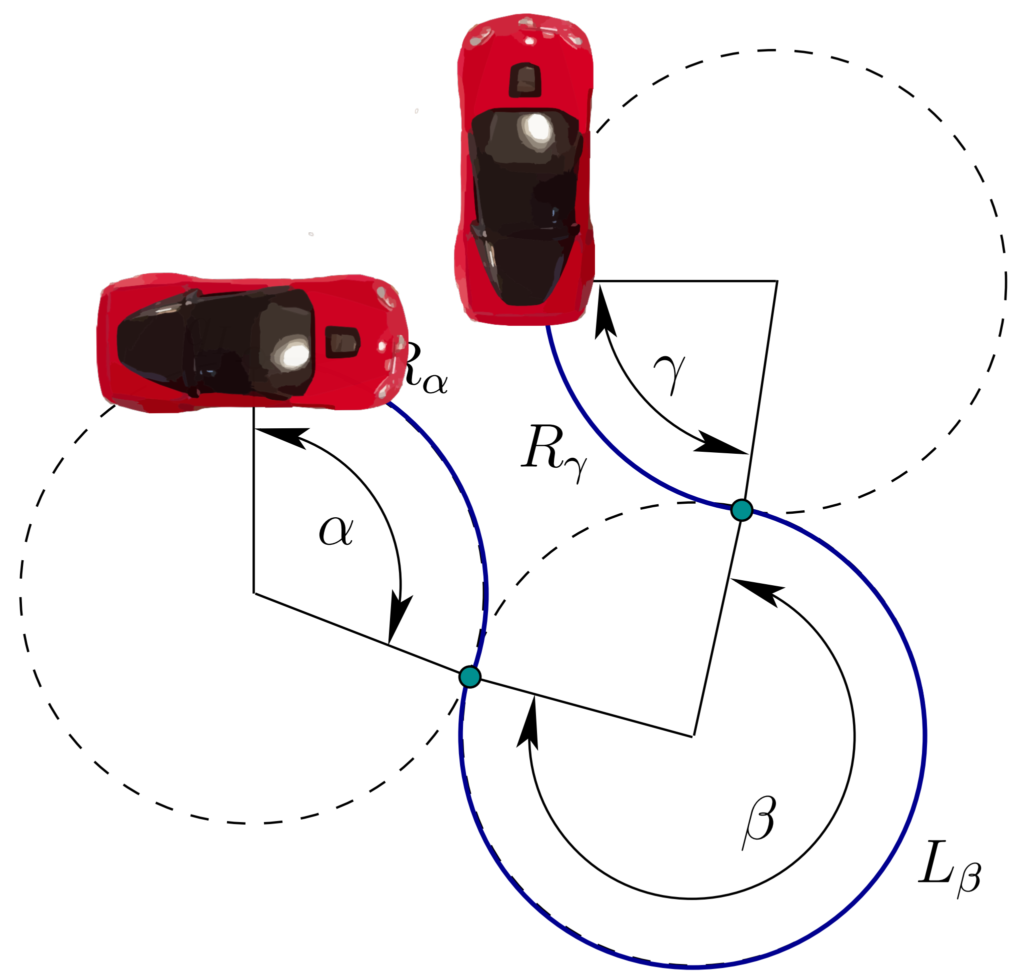
Example Dubins path

Planning with Nonholonomic Constraints

$$\{LRL,\ RLR,\ LSL,\ LSR,\ RSL,\ RSR\}.$$

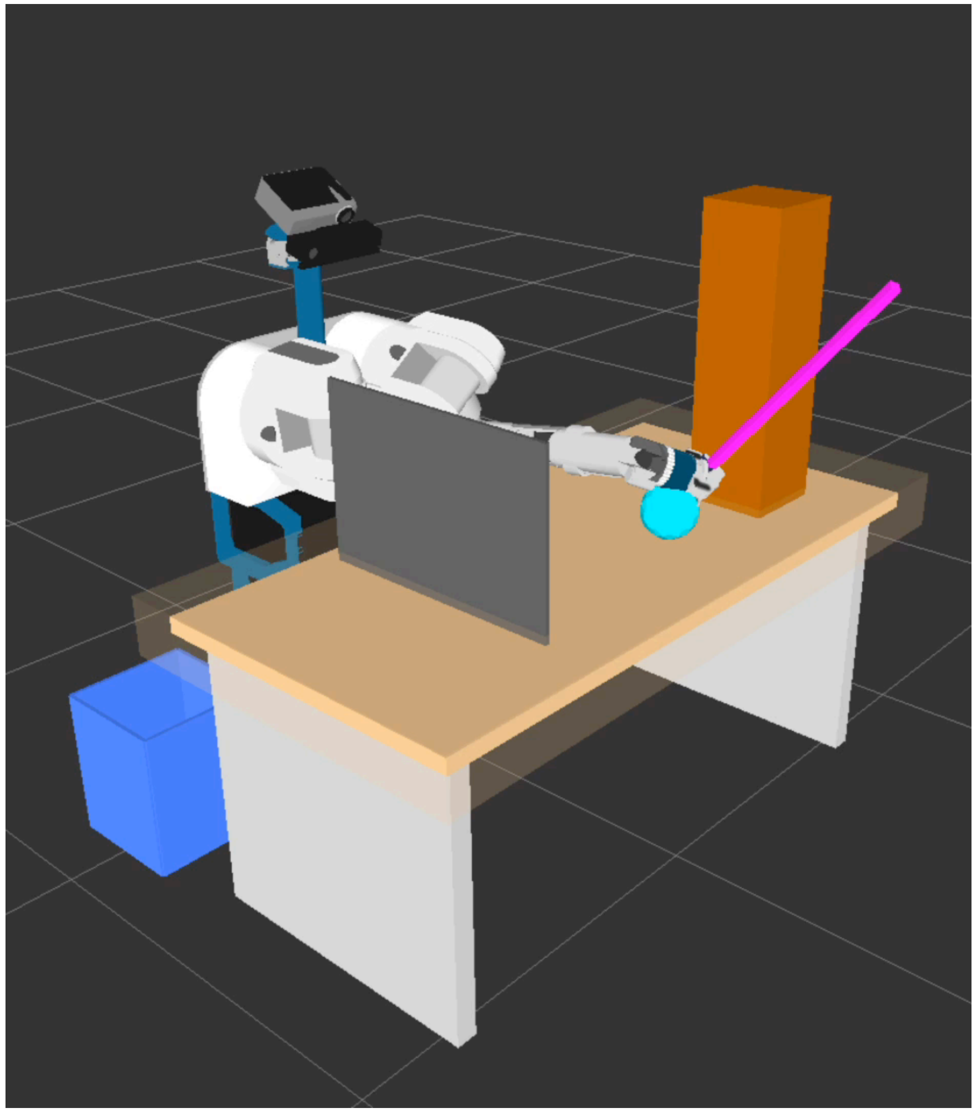
Given a query, generate ALL 6 options, and pick the shortest one!

We provide code for generating Dubins paths for Lab 3!



$R_\alpha L_\beta R_\gamma$

Geometric Path Planning Problem



Also known as

Piano Mover's Problem (Reif 79)

Given:

1. A *workspace* $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An *obstacle region* $\mathcal{O} \subset \mathcal{W}$.
3. A *robot* defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of m links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The *configuration space* $\mathcal{C}$ (C_{obs} and C_{free} are then defined).
5. An *initial configuration* $\boldsymbol{q_I} \in \mathcal{C}_{free}$.
6. A *goal configuration* $\boldsymbol{q_G} \in \mathcal{C}_{free}$. The initial and goal configuration are often called a *query* $(\boldsymbol{q_I}, \boldsymbol{q_G})$.

Compute a (continuous) path, $\tau : [0,1] \rightarrow \mathcal{C}_{free}$, *such that* $\tau(0) = \boldsymbol{q_I}$ *and* $\tau(1) = \boldsymbol{q_G}$.

Also may want to minimize cost $c(\tau)$

Theoretical guarantees that we desire

Completeness

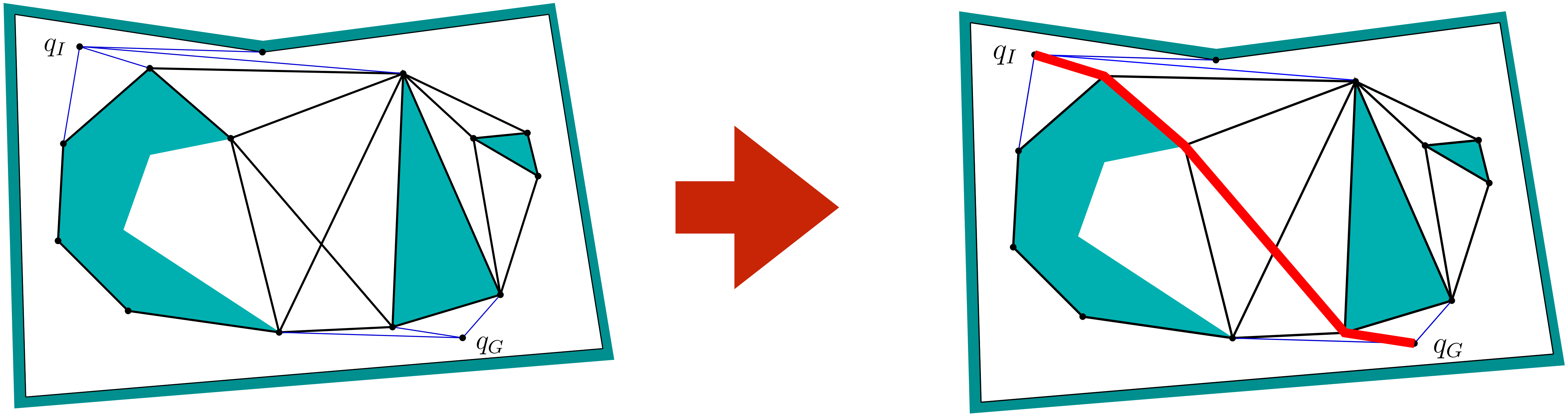
A planner is complete if for any input, it correctly reports whether or *not* a feasible path exists is finite time.

Optimality

Returns the best solution in finite time.

Is there any planner that guarantees this?

Yes! 2D Visibility Graphs!



E.g. 2D polygon robots / obstacles can be solved with visibility graphs

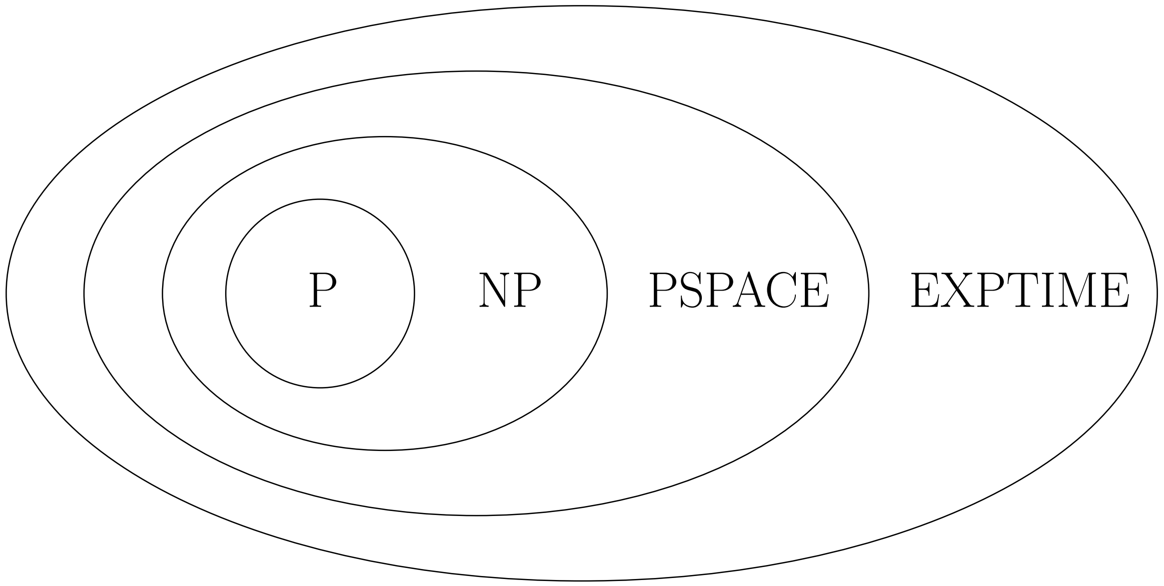
Typical runtime: $O(N^2 \log N)$

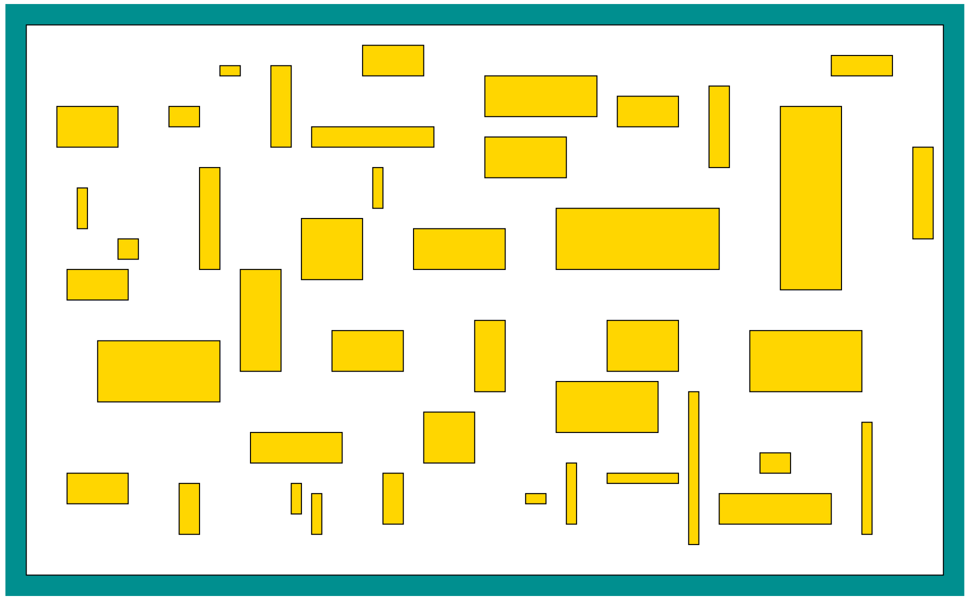
So, are we done ... ?

No! Planning in general is hard

Hardness of motion planning

Piano Mover's problem is PSPACE-hard (Reif et al. 79)





Even planning for translating rectangles is PSPACE-hard!
(Hopcroft et al. 84)

Certain 3D robot planning under uncertainty is NEXPTIME-hard!

(Canny et al. 87)

Why is it so hard?

1. Computing the C-space obstacle in high dimensions is hard

2. Planning in continuous high-dimension space is hard

Exponential dependency on dimension

Why is it so hard?

~~1. Computing the C space obstacle in high dimensions is hard~~

We won't! Instead we will use a collision checker!

~~2. Planning in continuous high dimension space is hard~~

We will bring it to discrete space by sampling configurations!

Research in Motion Planning:

Make good approximations

(that have guarantees)

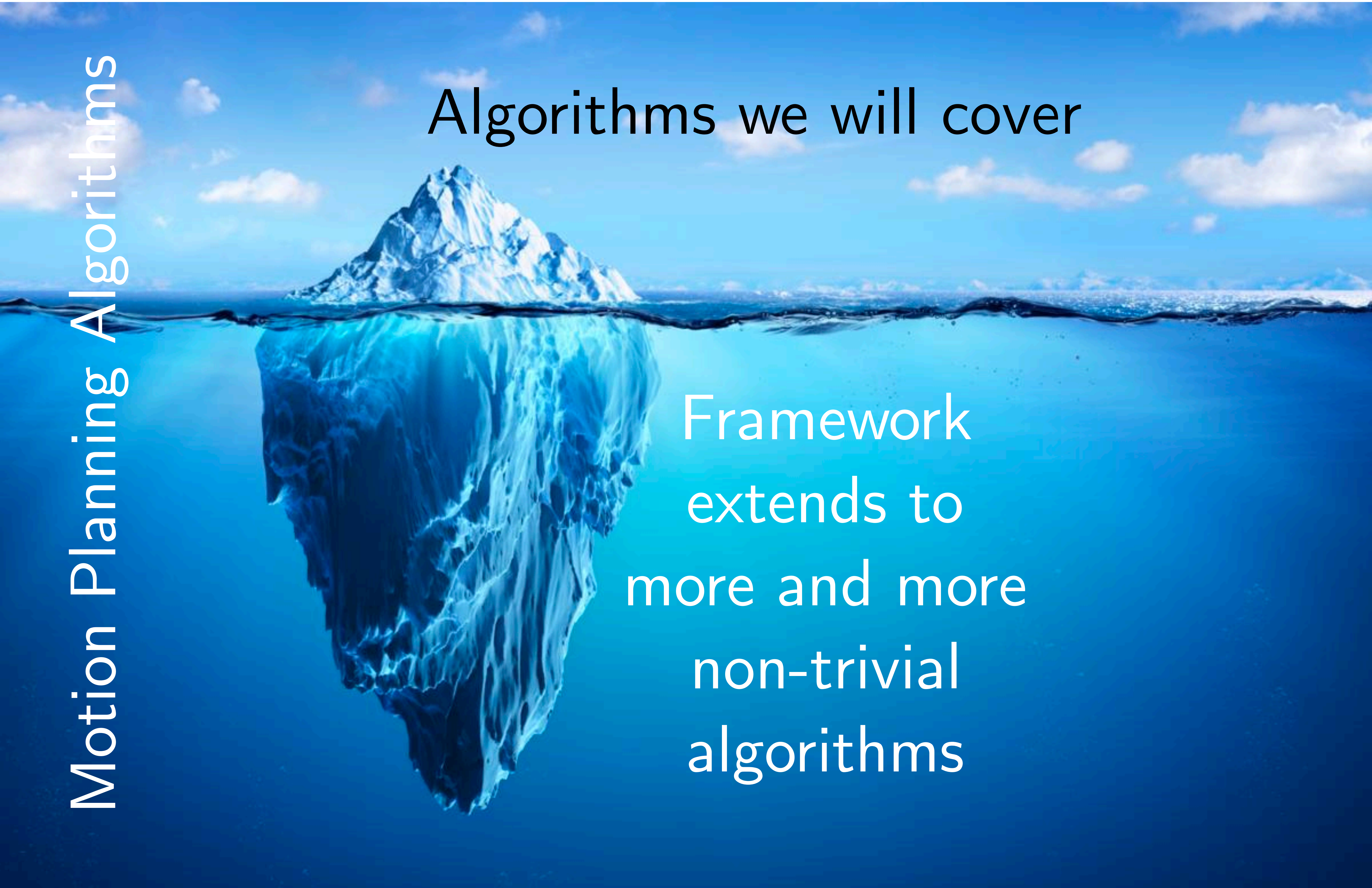
Today's objective

1. General framework for motion planning

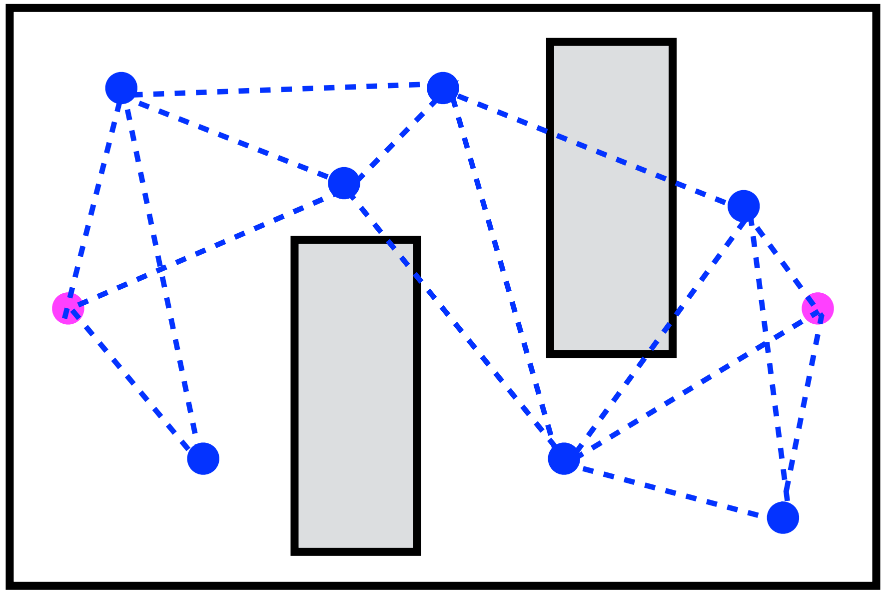
2. Inputs to any planner: Collision checking and steering

3. Planning on roadmaps - one class of instantiations of the framework

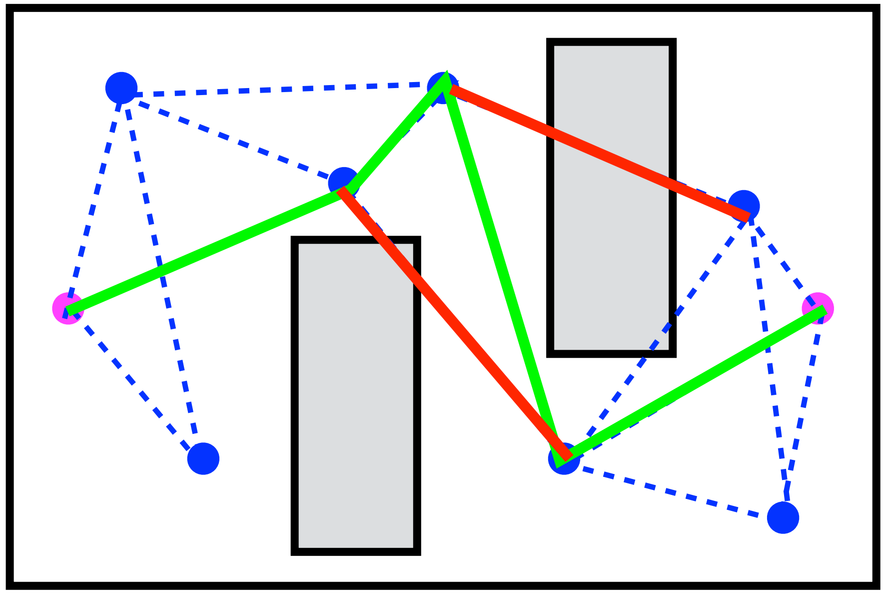
Why an abstract framework?



General framework for motion planning



Create a graph



Search the graph

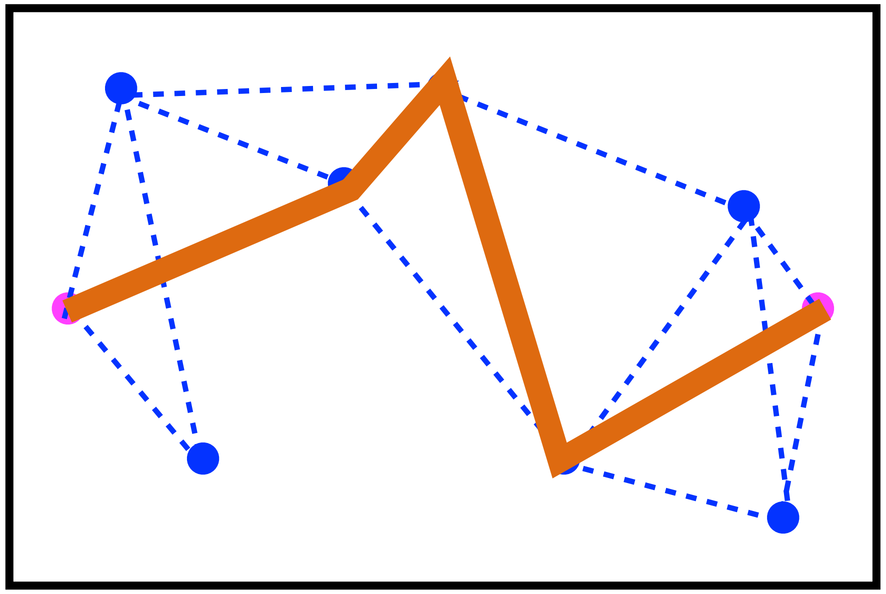
Interleave

General framework for motion planning

Any planning algorithm		Create graph		Search graph		Interleave
	$=$	e.g. fancy random sampler	$\times$	e.g. fancy heuristic	$\times$	e.g. fancy way of densifying
		Whats the best we can do?		Whats the best we can do?		Whats the best we can do?

For this lecture....

Assume you are given a super awesome search subroutine!



Optimal Path = SHORTESTPATH(V,E, start, goal)

(Next lecture we will talk about how we get this)

Assume complexity is $O(|V| \log |V| + |E|)$

API for motion planning

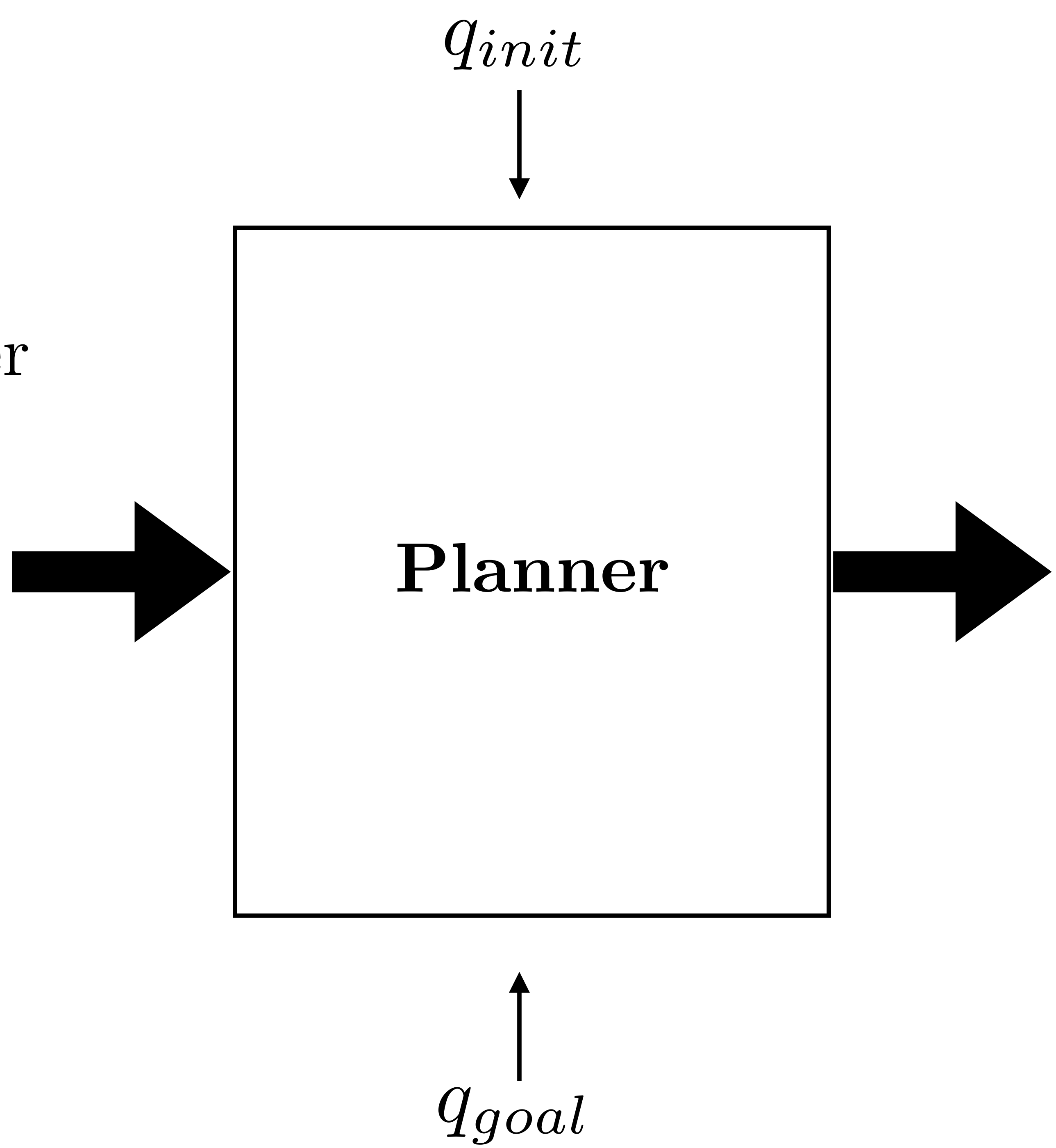
Input

1. A collision checker

`coll`(q)

2. Steering method

`steer`(q_1, q_2)



Output

Collision free path joining start and goal

Let's take a look at the inputs

We need to give the planner a collision checker

$$\texttt{coll}(q) = \begin{cases} 0 & \text{in collision, i.e. } q \in \mathcal{C}_{obs} \\ 1 & \text{free, i.e. } q \in \mathcal{C}_{free} \end{cases}$$

What work does this function have to do?

Collision checking is expensive!

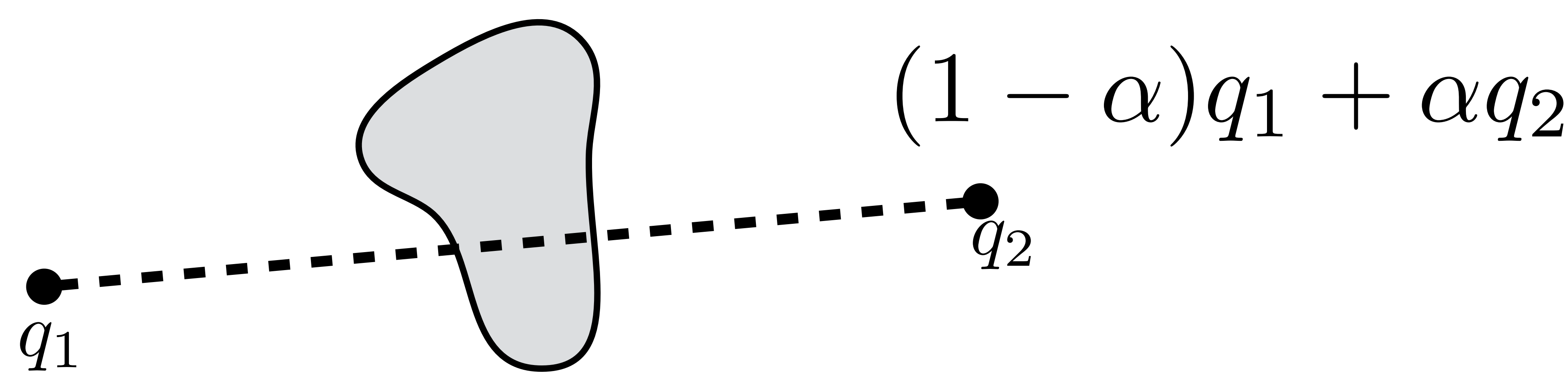
Let's take a look at the inputs

We need to give the planner a steer function

$$\texttt{steer}(q_1, q_2)$$

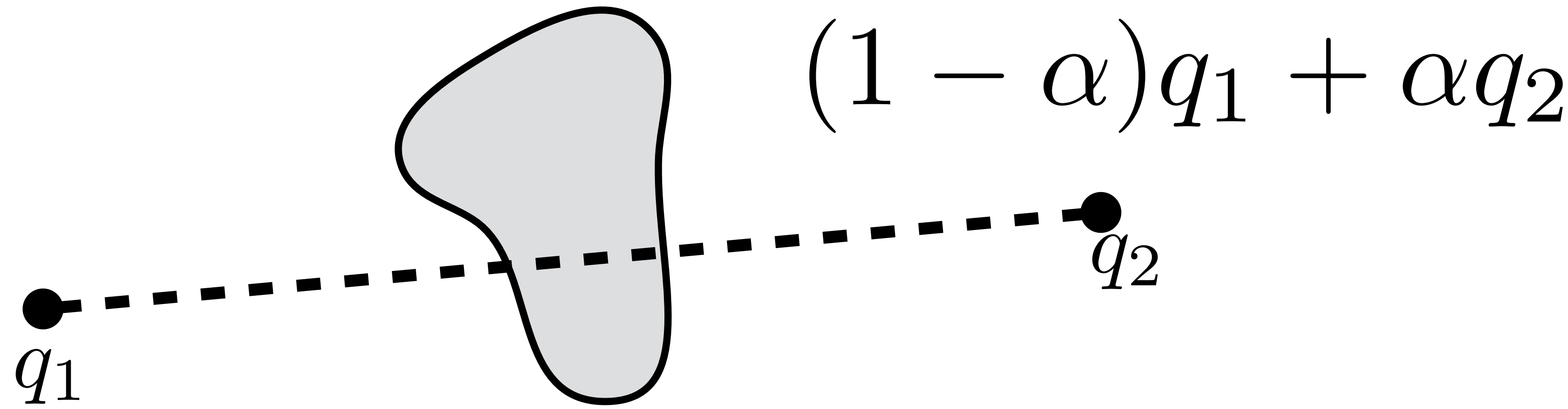
A steer function tries to join two configurations with a feasible path

Computes simple path, calls `coll`(q), and returns success if path is free



Example: Connect them with a straight line and check for feasibility

Can steer be smart about collision checking?



`steer`(q_1, q_2) has to assure us line is collision free (up to a resolution)

Things we can try:

1. Step forward along the line and check each point

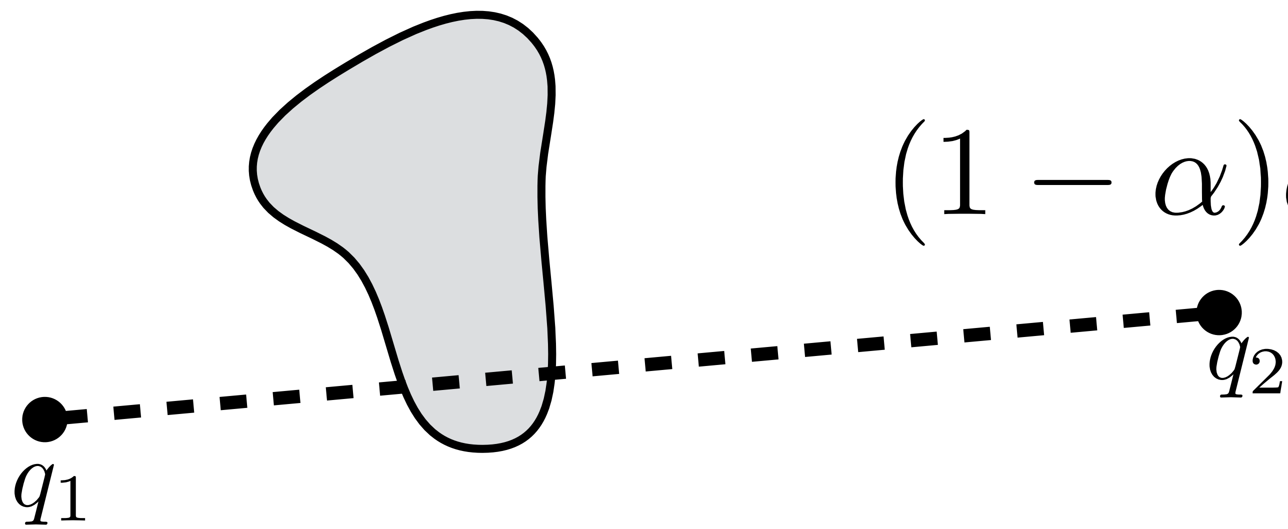
2. Step backwards along the line and check each point

....

Can steer be smart about collision checking?

Say we chunk the line into 16 parts

$$(1-\alpha)q_1 + \alpha q_2$$



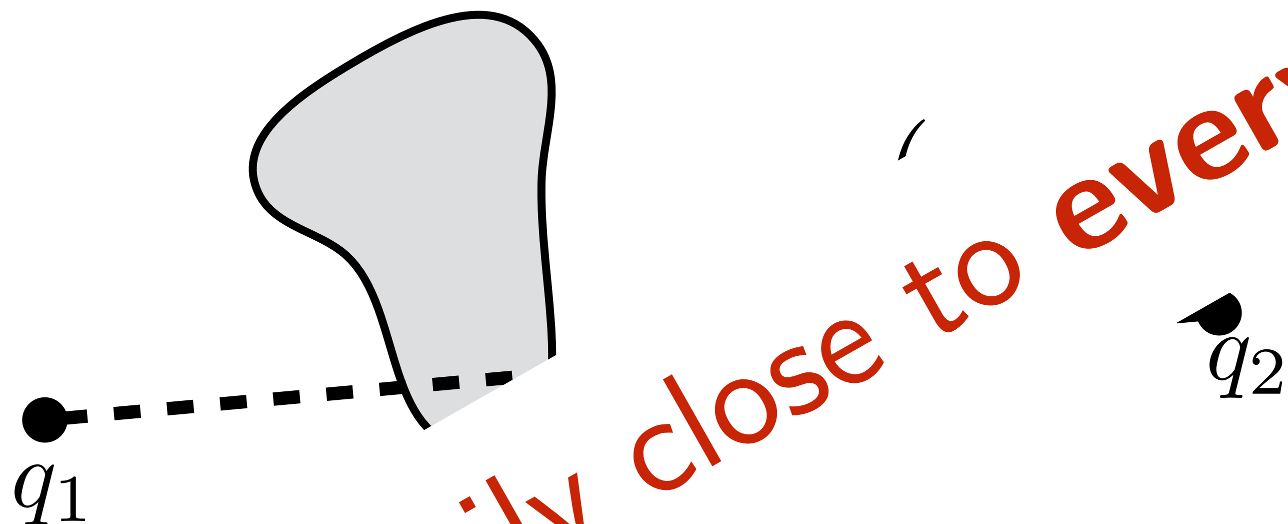
Any collision checking strategy corresponds to sequence

(Naive) $\alpha = 0, \frac{1}{16}, \frac{2}{16}, \frac{3}{16}, \cdots, \frac{15}{16}$

(Bisection) $\alpha = 0, \frac{8}{16}, \frac{4}{16}, \frac{12}{16}, \cdots, \frac{15}{16}$

Can steer be smart about collision checking?

Say we chunk the line into 16 parts



$+ \alpha q_2$

Can we get arbitrarily close to **every** element?

Any ... strategy corresponds to sequence

$$\alpha = 0, \frac{1}{16}, \frac{2}{16}, \frac{3}{16}, \cdots, \frac{15}{16}$$

(Bisection) $$\alpha = 0, \frac{8}{16}, \frac{4}{16}, \frac{12}{16}, \cdots, \frac{15}{16}$$

Answer: Sample Densely (Van Der Corput Sequence)

i	Naive Sequence	Binary
1	0	
2	1/16	
3	1/8	
4	3/16	
5	1/4	
6	5/16	
7	3/8	
8	7/16	
9	1/2	
10	9/16	
11	5/8	
12	11/16	
13	3/4	
14	13/16	
15	7/8	
16	15/16	

How can we ensure that we get *better* coverage?

Alternate between bounds of Config Space

Now we are ready to talk about planner!

Input

1. A collision checker

   $\texttt{coll}(q)$

2. Steering method

   $\texttt{steer}(q_1, q_2)$

q_{init}

Planner

q_{goal}

Output

Collision free path joining start and goal

Framework for planner

1. Create a graph

(Think about what makes a good graph as we go along)

2. Search the graph (assume solved for now)

Creating a graph: Abstract algorithm

$$G = (V, E)$$

Vertices: set of configurations

Edges: paths connecting configurations

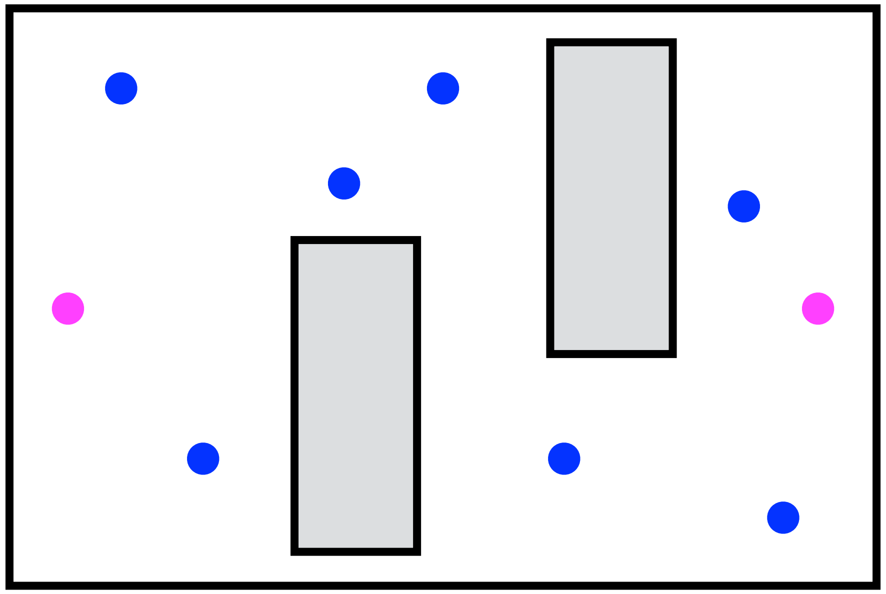
Creating a graph: Abstract algorithm

$$G = (V, E)$$

Vertices: set of configurations

Edges: paths connecting configurations

1. Sample a set of collision free vertices V (add start and goal)



Sample a configuration q

if $\texttt{coll}(q) = 1$

$$V \leftarrow V \cup \{q\}$$

Creating a graph: Abstract algorithm

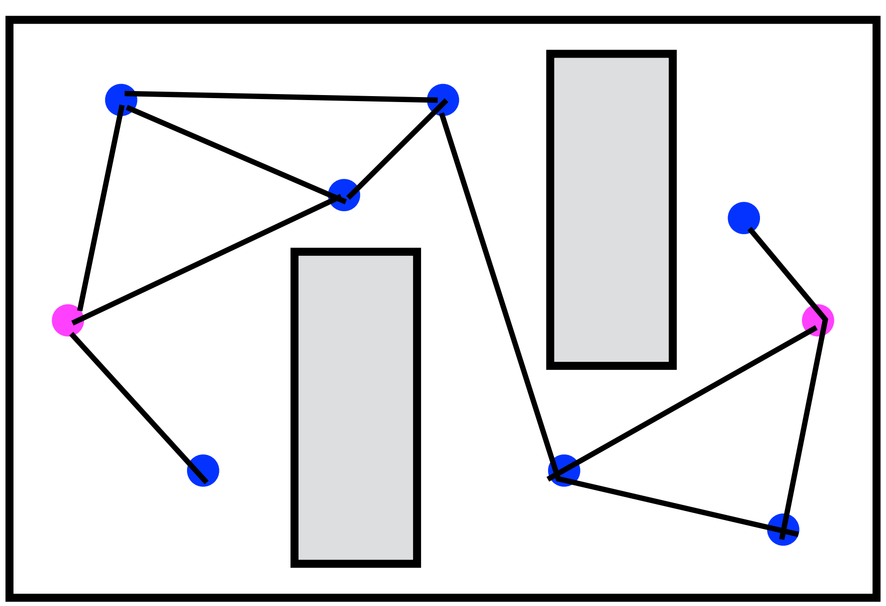
$$G = (V, E)$$

Vertices: set of configurations

Edges: paths connecting configurations

1. Sample a set of collision free vertices V (add start and goal)

2. Connect "neighboring" vertices to get edges E



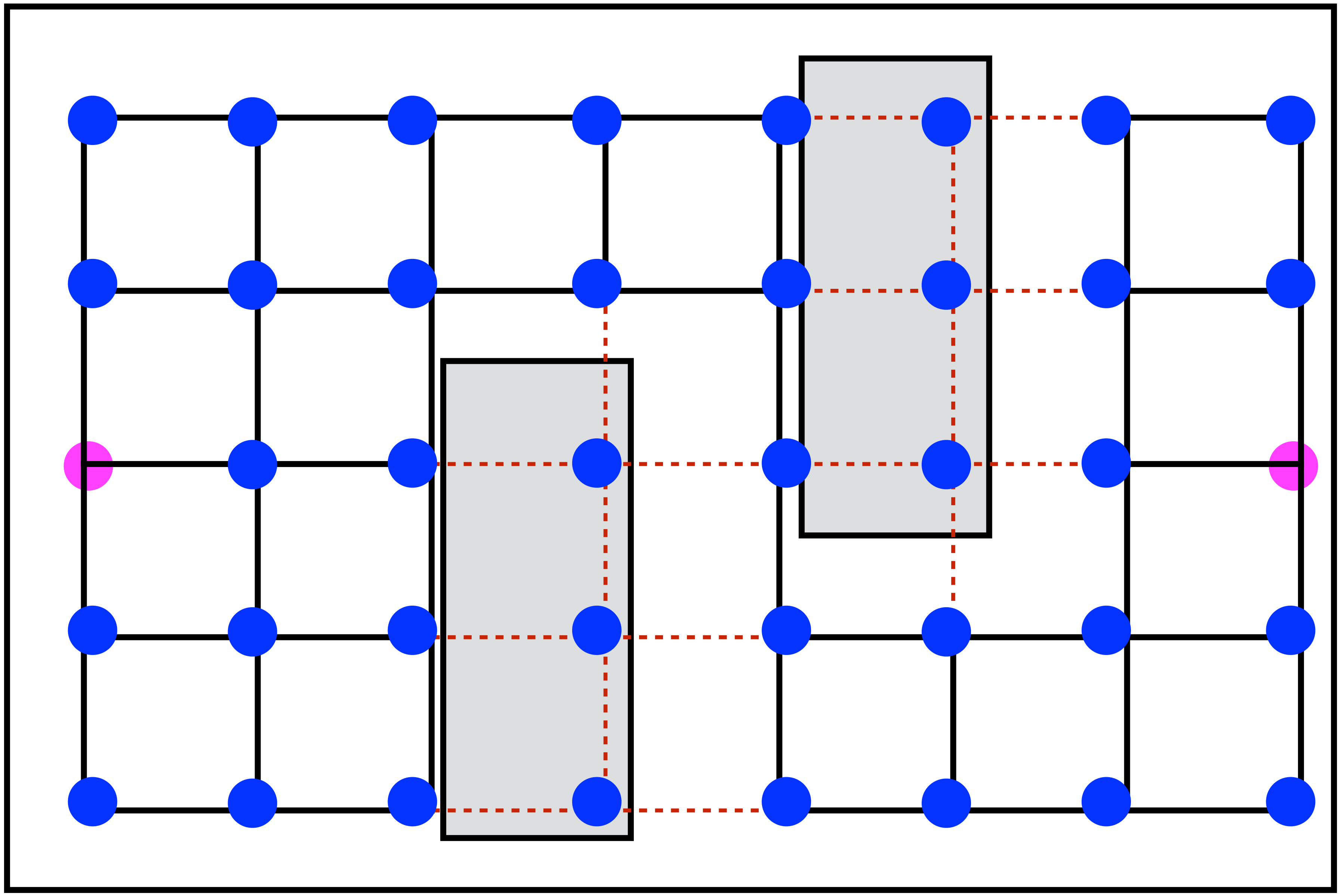
for each candidate pair (v_1, v_2)

if $\texttt{steer}(v_1, v_2)$ succeeds

$$E \leftarrow E \cup (v_1, v_2)$$

Strategy 1: Discretize configuration space

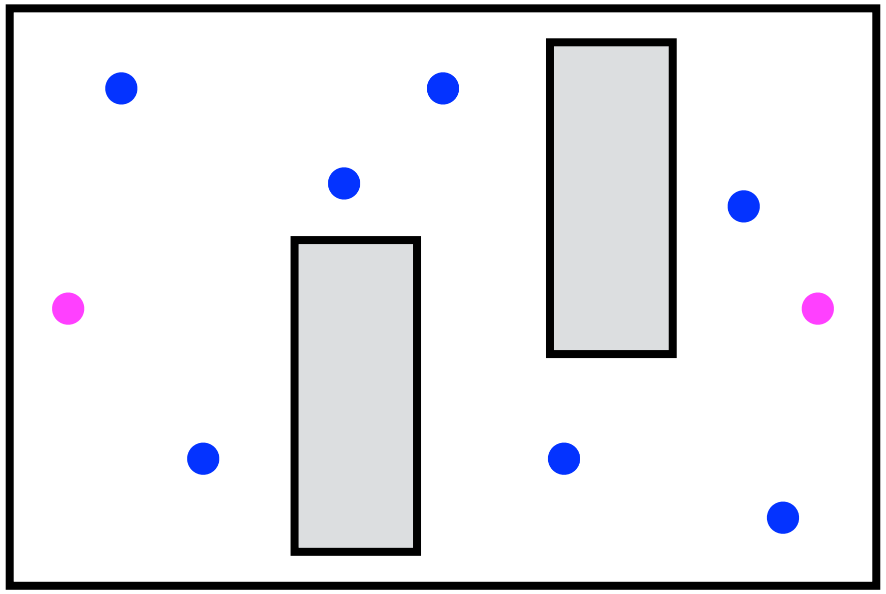
Create a lattice. Connect neighboring points (4-conn, 8-conn, ...)



Theoretical guarantees: Resolution complete

What are the pros? What are the cons?

Strategy 2: Uniformly randomly sample



If C-space is a real vector space

for each dimension i
    sample $q(i) \sim [lb, ub]$

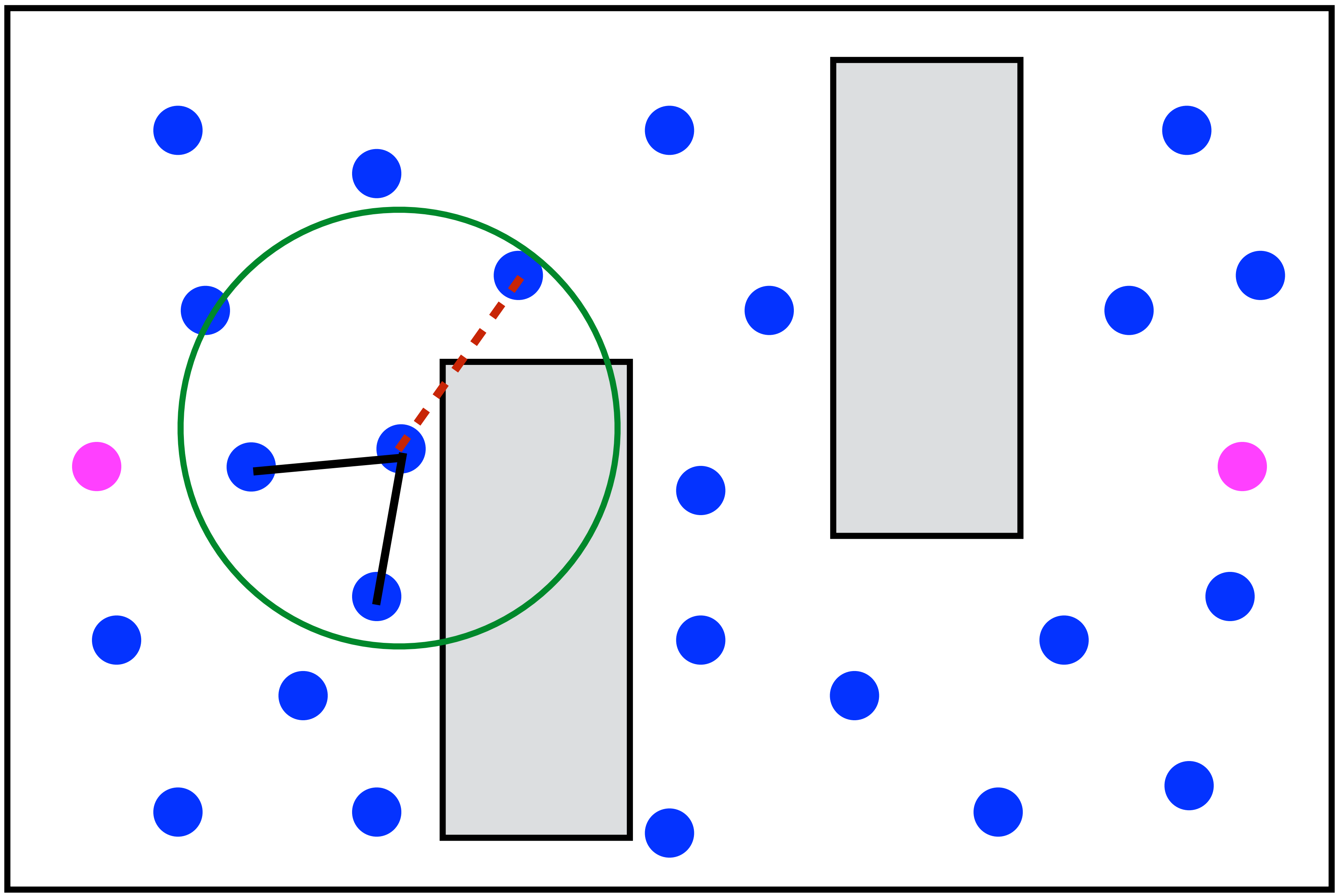
What are the pros of random sampling? Cons?

Question:
How do we decide which vertices to connect?

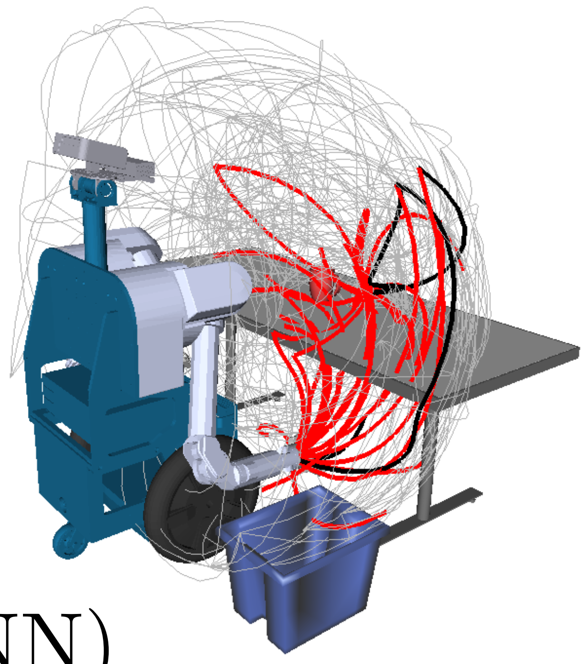
Strategy 2: Uniformly randomly sample

Connect vertices that are a within a radius

(Alternatively can connect k-nearest neighbors)



This is the PRM Algorithm!



PRM = Probabilistic Roadmap

1. Sample vertices randomly
2. Connect vertices within radius (or k NN)
3. Search graph to find a solution

Theoretical Guarantees: It depends ...

L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics & Automation, 12(4):566–580, June 1996.

Questions we can ask PRM

1. When is it a good idea to collision check every single edge?

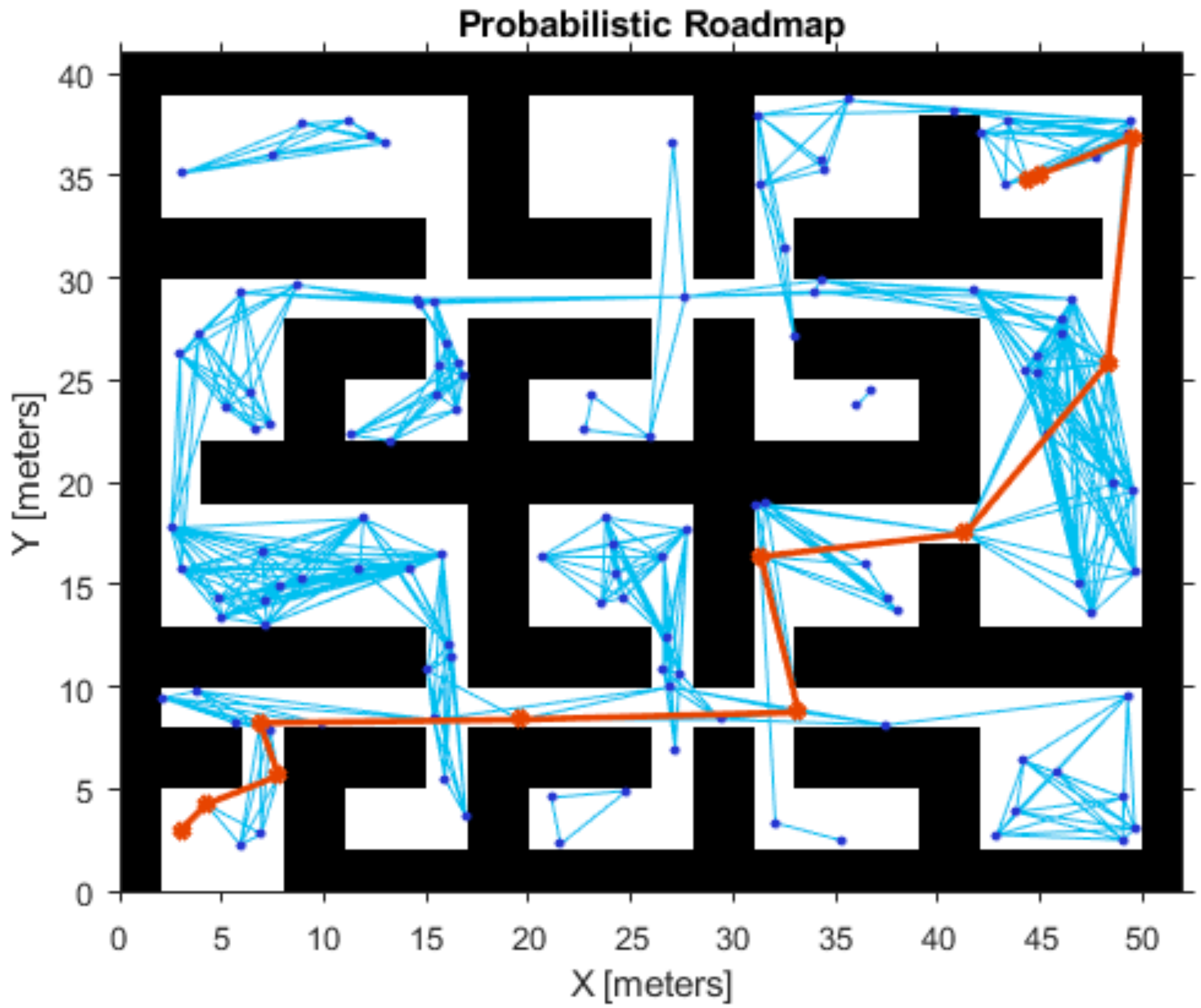
   Ans: Multi-query!

2. How should we efficiently find nearest neighbors?

   Ans: Use a KD-Tree data-structure

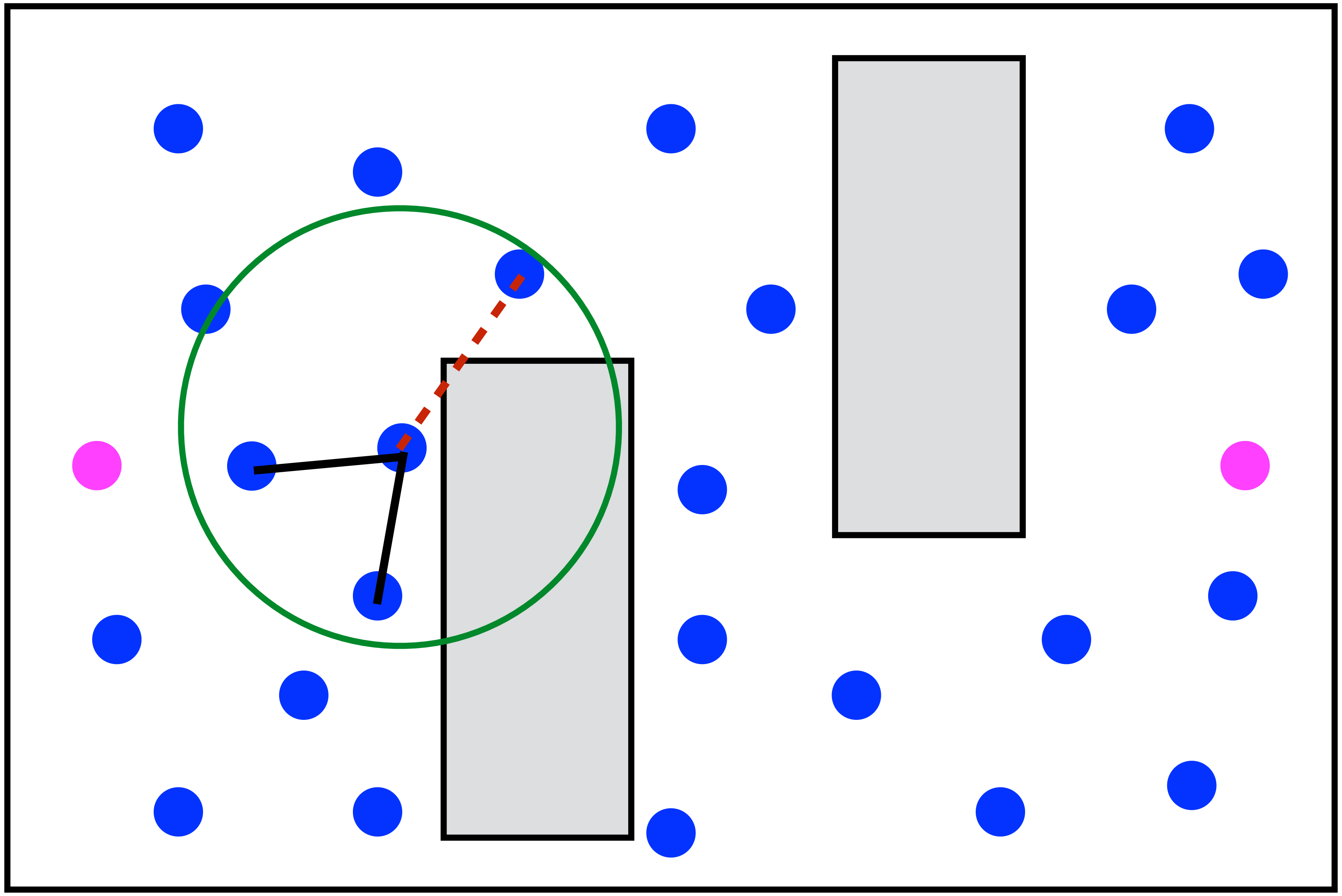
3. How should we choose which vertices to connect?

   Ans: Up Next!



What is the optimal radius?

What happens if radius too large? too small?

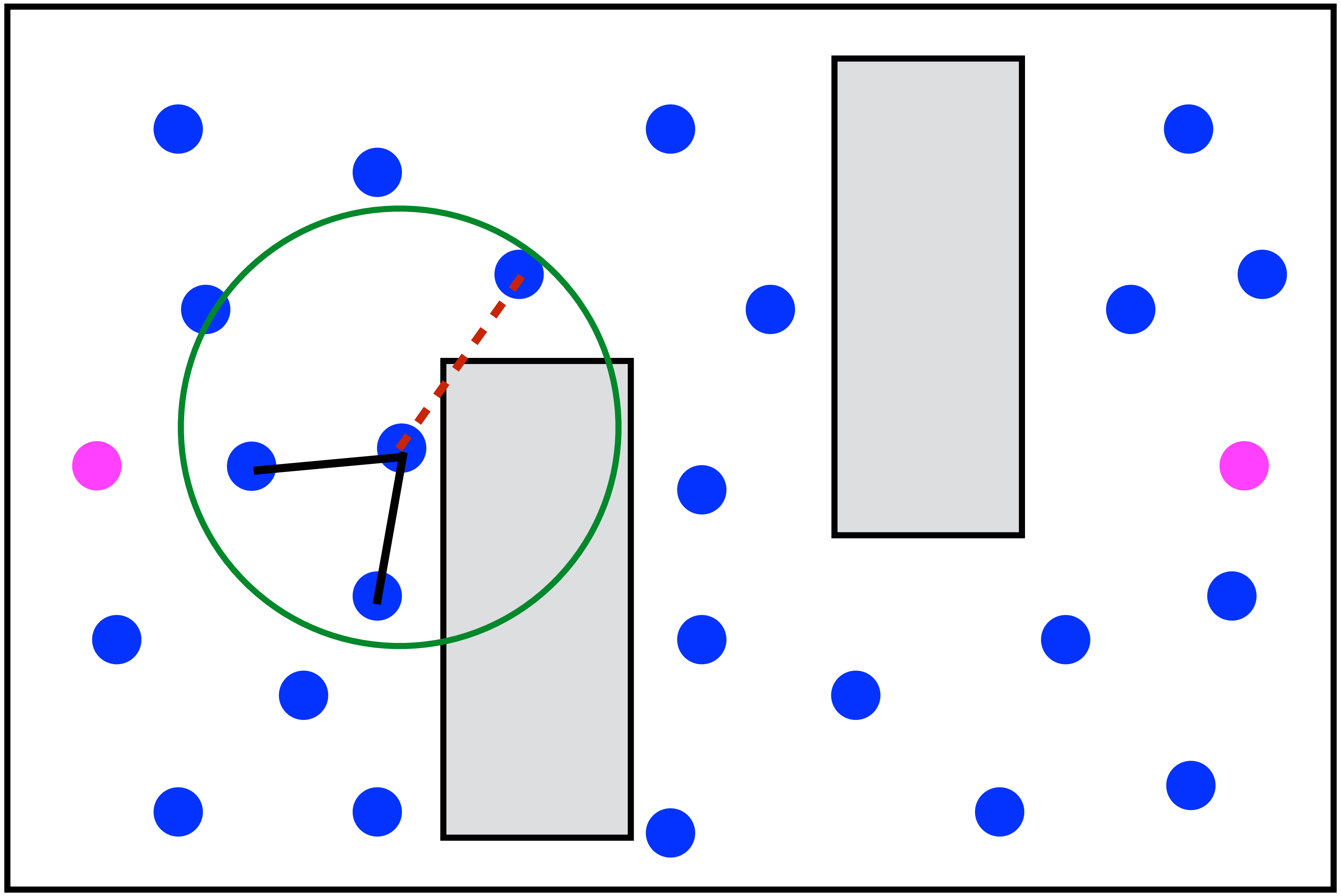


What is the optimal radius?

Set the radius to $r = \gamma \left(\frac{\log |V|}{|V|} \right)^{1/d}$

where magic constant!

$$\gamma \geq 2(1 + 1/d)^{1/d} \frac{\mu(\mathcal{C}_{free})}{\zeta_d}$$



Also known as a Random Geometric Graph (RGG)

This is the PRM* Algorithm!

1. Sample vertices randomly

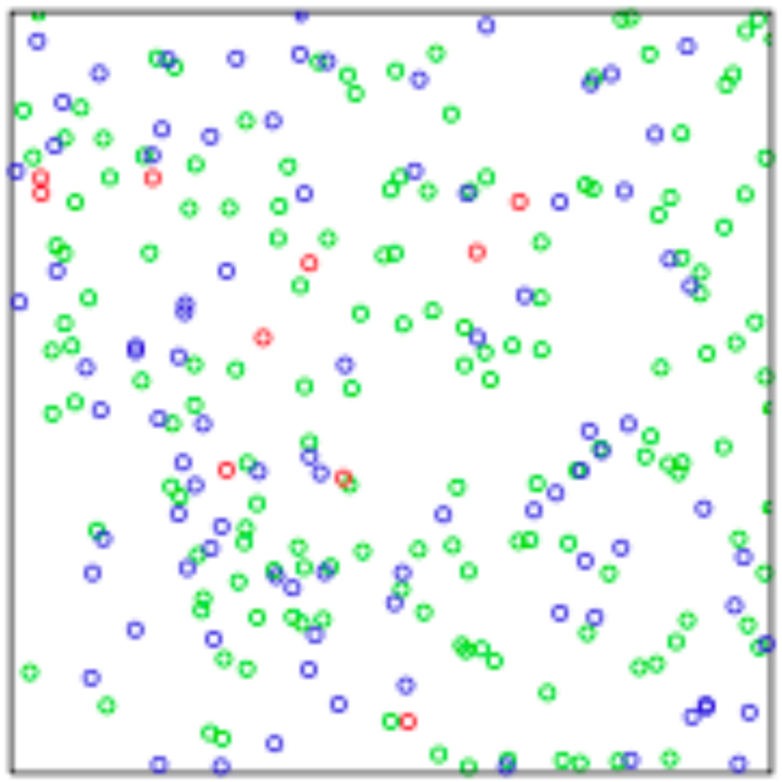
2. Use optimal radius formula to connect vertices

3. Search graph to find a solution

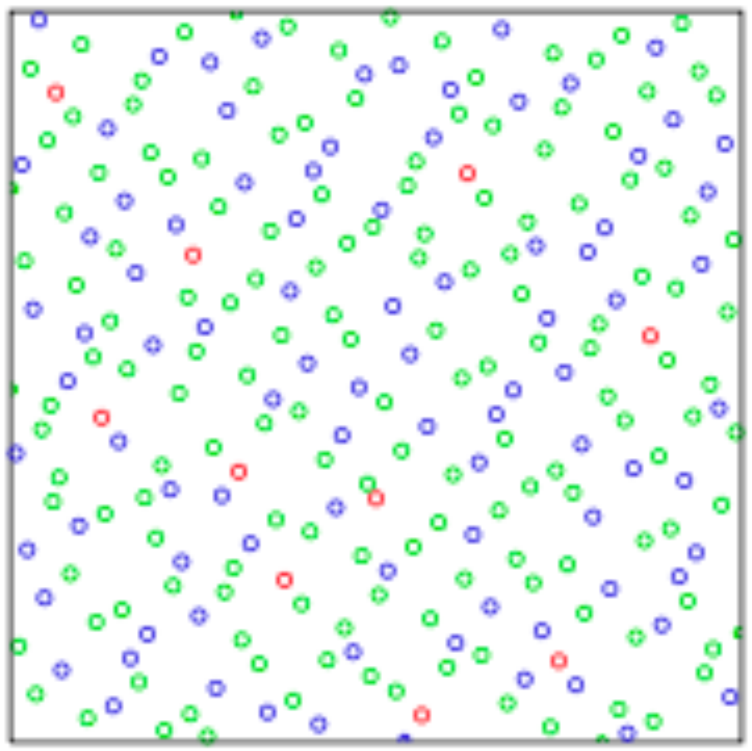
Theorem: Probabilistically complete AND Asymptotically optimal

"Sampling-based Algorithms for Optimal Motion Planning"
Sertac Karaman and Emilio Frazzoli, IJRR 2011

Can we do better than random?

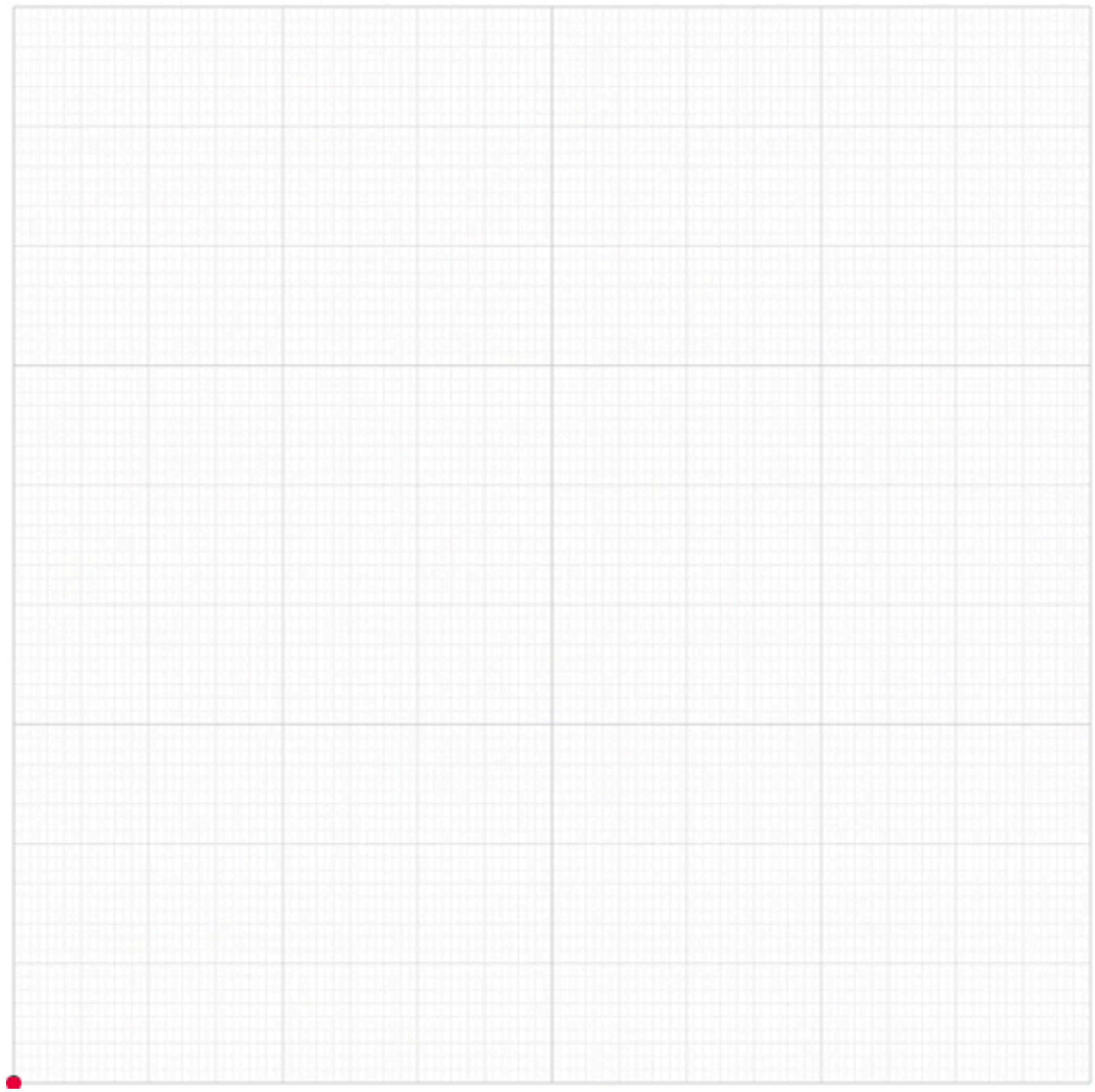


Uniform random sampling tends to clump



Ideally we would want points to be spread out evenly

Halton Sequence

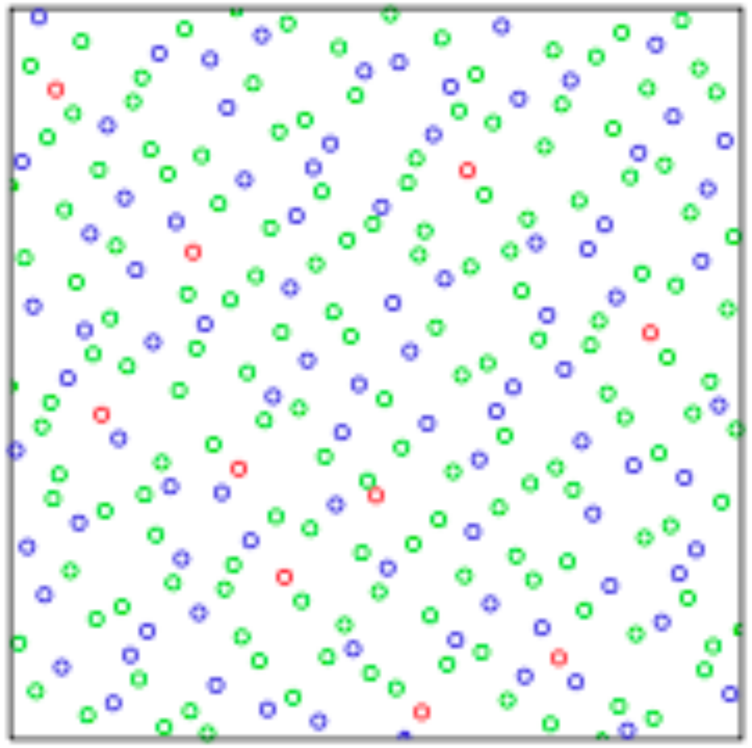


Generalization of
Van de Corput Sequence

Intuition: Create a sequence using prime numbers that uniformly densify space

Link for exact algorithm:
https://observablehq.com/@jrus/halton

How do we connect vertices?



Halton sequences have much better coverage

(i.e. they are low **dispersion**)

Connect vertices that are within a radius of

$$r = \gamma \left(\frac{1}{|V|} \right)^{1/d}$$

(as opposed to:)

$$r = \gamma \left(\frac{\log |V|}{|V|} \right)^{1/d}$$

This is the gPRM Algorithm!

1. Sample vertices randomly

2. Use optimal radius formula to connect vertices

3. Search graph to find a solution

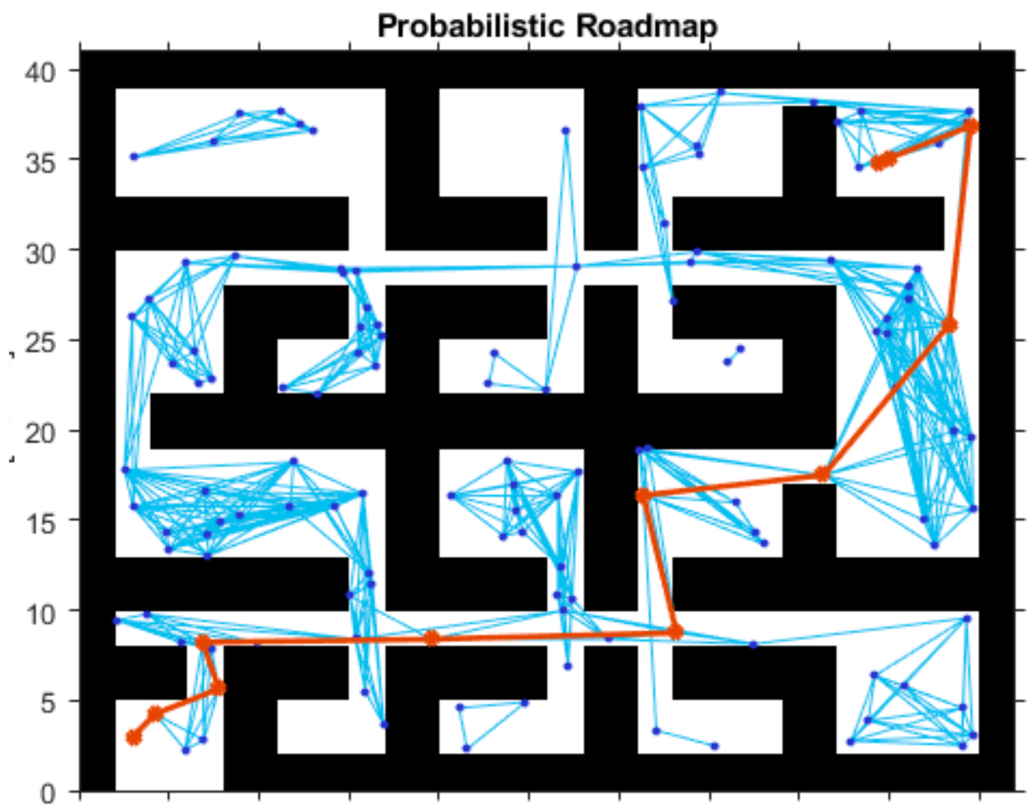
Theorem: Probabilistically complete AND Asymptotically optimal AND Asymptotic rate of convergence

"Deterministic Sampling-Based Motion Planning: Optimality, Complexity, and Performance"
Lucas Janson, Brian Ichter, Marco Pavone, IJRR 2017

What makes a good graph?

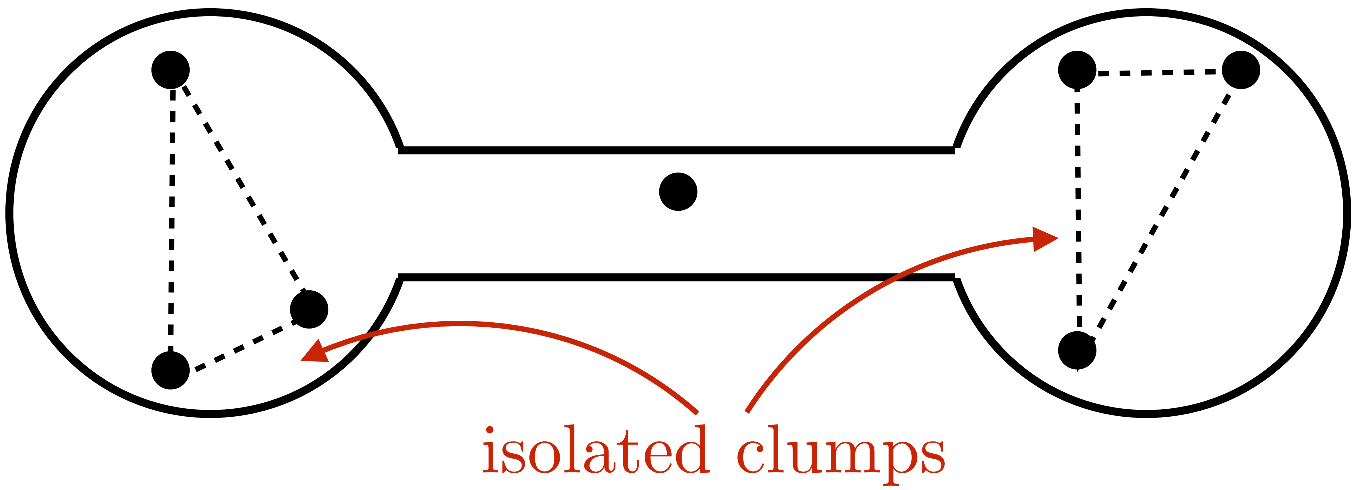
1. A good graph must be sparse (both in vertices and edges)

2. A good graph must have good coverage



3. A good graph must have good connectivity of free space

The Narrow Passage: Planning's boogie man!



Why is narrow passage mathematically hard to plan in?

Mathematical Question:

How many samples do we need to connect the space (with high probability)?

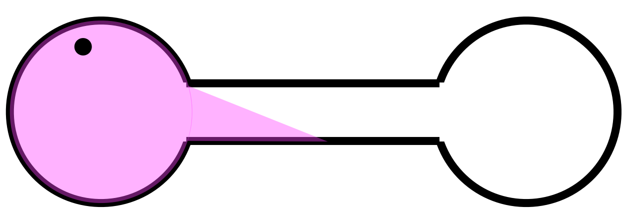
How many samples do we need?

Theorem [Hsu et al., 1999] Let $2n$ vertices be sampled from X_{free}.
Then the roadmap is connected with probability at least $1-\gamma$ if:

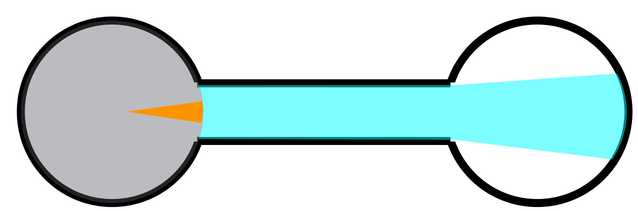
$$n \geq \left\lceil 8 \frac{\log\left(\frac{8}{\epsilon\alpha\gamma}\right)}{\epsilon\alpha} + \frac{3}{\beta} + 2 \right\rceil$$

The shape of free C-space is dictated by $\alpha,\ \beta,\ \epsilon \in [0,1]$

Visibility of free space (ϵ)



Expansion of visibility (α, β)

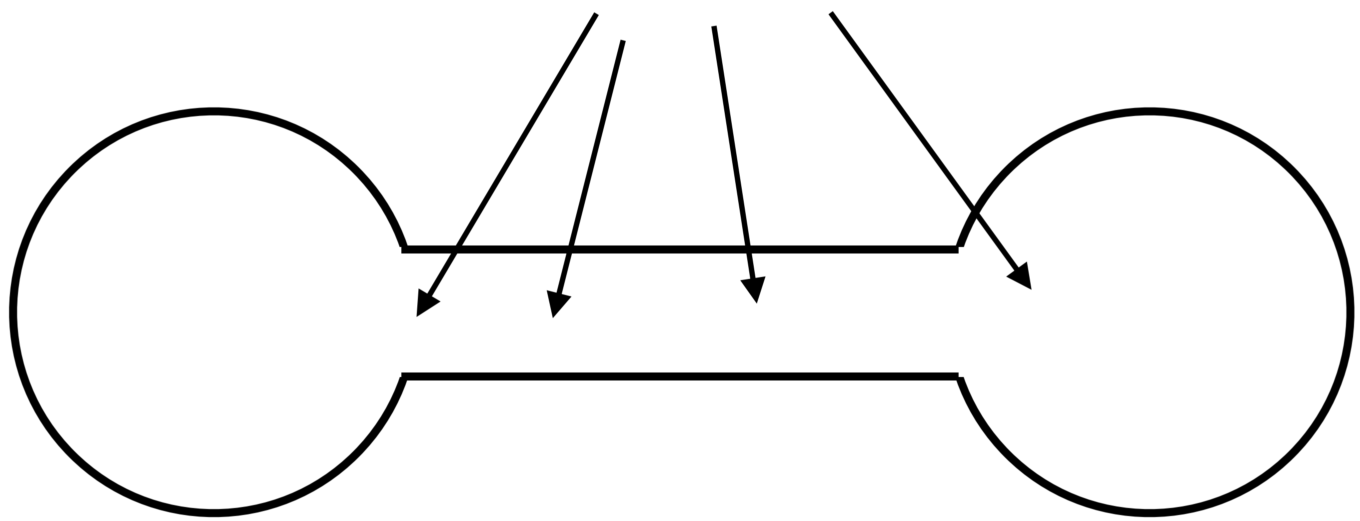


Narrow passage has small values of $\alpha,\ \beta,\ \epsilon$

Hence, needs more samples to find a path

How do we bias sampling?

We somehow need more samples here



1. Sample near obstacle surface?

   V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. 1999

2. Add samples that are in between two obstacles?

   D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners.2003.

3. Train a learner to detect the narrow passages?

   B. Ichter, J. Harrison, M. Pavone. Learning Sampling Distributions for Robot Motion Planning, 2018

Summary of ways to create graphs

	How to sample vertices?	How to connect vertices?
Lattice	Discretize	connectivity rule
PRM	Uniform random	r-disc, k-nn
PRM*	Uniform random	optimal r-disc, k-nn
gPRM	Halton sequence	optimal r-disc, k-nn
Bridge	Sample with bridge test	any visible points
Gaussian	Sample near obstacles	r-disc, k-nn
MAPRM	Sample along medial axis	r-disc, k-nn
Approx. Visibility Graph	Sample on surface of obstacles	any visible points
Learnt Sampler	Use CVAE to approximate free space	optimal r-disc, k-nn