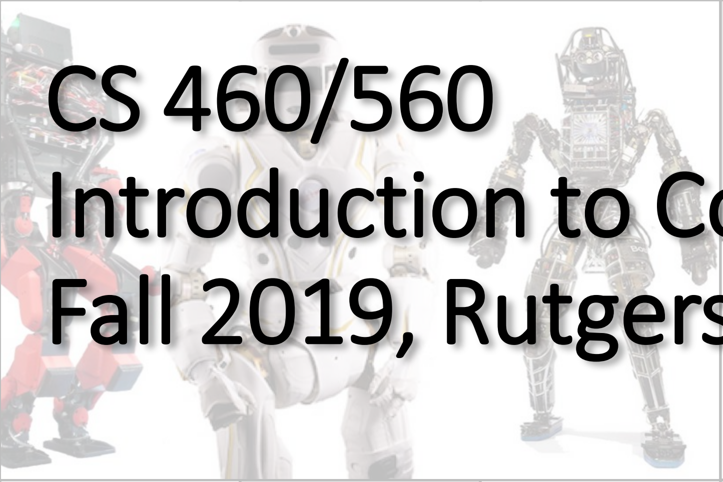
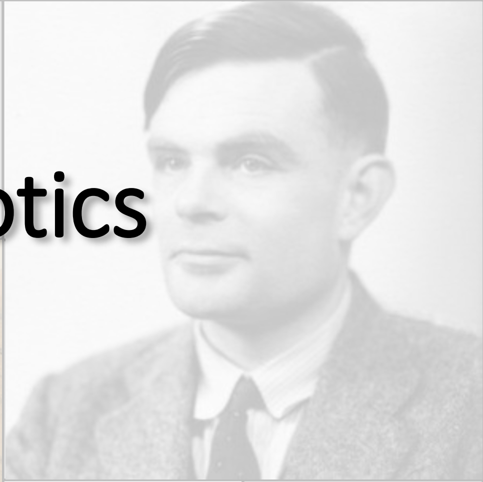
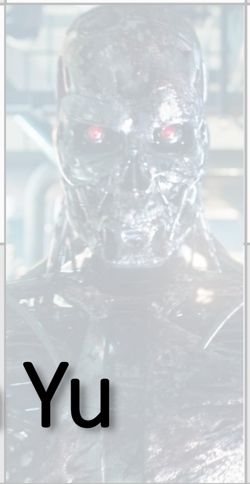
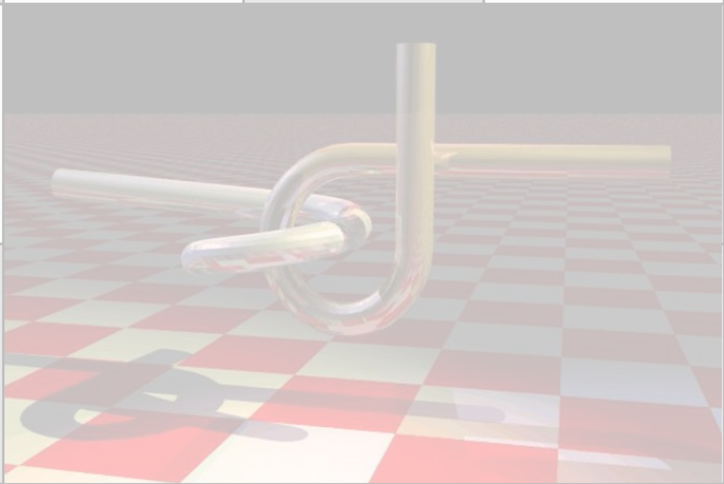
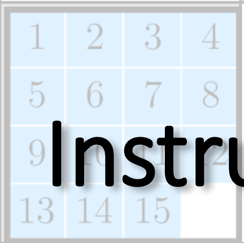




CS 460/560
Introduction to Computational Robotics
Fall 2019, Rutgers University

Midterm Review

Instructor: Jingjin Yu

Set, Set Operations, and Venn Diagram

A **set** is a collection of elements. Examples:

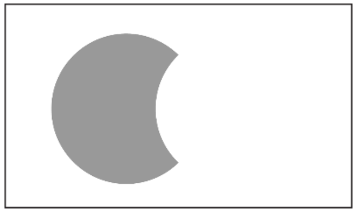
- ⇨$\{1, a, cup, \pi, \quad\}$ – elements do need not be of the same type
- ⇨Natural numbers (an infinite set), $\mathbb{N} = \{0, 1, 2, \ldots\}$
- ⇨n-dimensional Euclidean spaces, $\mathbb{R}^n$ (e.g., $\mathbb{R}^3$ is the 3-dimensional space)

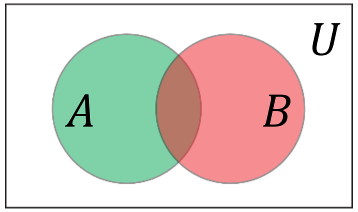
Set operations

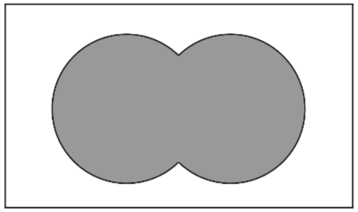
such that

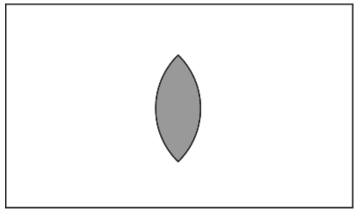
- ⇨Union: $A \cup B = \{x \mid x \in A \lor x \in B\}$
- ⇨Intersection: $A \cap B = \{x \mid x \in A \land x \in B\}$
- ⇨Complement: $\overline{A} = \{x \mid x \in U \land x \notin A\}$
- ⇨Difference: $A - B = \{x \mid x \in A \land x \notin B\}$ (or $A\backslash B$)
- ⇨Symmetric difference: $A \ominus B = A \cup B - A \cap B$

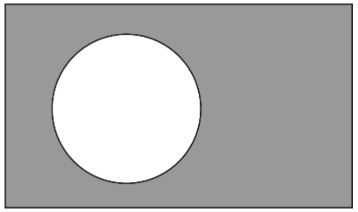
Venn diagram

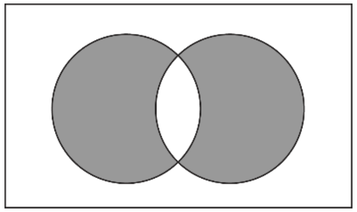

$A - B$




$A \cup B$


$A \cap B$


$\overline{A}$


$A \ominus B$

Power Set and Cardinality

Powerset: $\mathcal{P}(S) = \{A \mid A \subset S\}$, example:

- ⇨ $S = \{1,2\}$
- ⇨ $\mathcal{P}(S) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$

Cardinality: essentially the “size” of a set

- ⇨ $|\emptyset| = 0$
- ⇨ $|\{1, 2\}| = 2$
- ⇨ $|\mathcal{P}(\{1,2\})| = |\{\emptyset, \{1\}, \{2\}, \{1,2\}\}| = 4$
- ⇨ In general, $|\mathcal{P}(S)| = 2^{|S|}$
- ⇨ $|\mathbb{N}| = \aleph_0$ - the “smallest” infinite (cardinal) number, read “Aleph 0”
- ⇨ $|\mathbb{R}| = \aleph_1$ - there are “more” real number than natural numbers

Measuring the relative cardinality of sets

- ⇨ $|A| \leq |B|$ if there exists an injective function $f: A \to B$
- ⇨ If $|A| \leq |B|$ and $|B| \leq |A|$, then $|A| = |B|$
  - ⇨ This means there is a **bijective function** between A and B
- ⇨ $|\mathbb{Q}| = |\mathbb{N}|$ - countable
- ⇨ $|\mathbb{R}| > |\mathbb{N}|$, real numbers are uncountable

Group Theory Concepts

A **set** G together with a **binary operation** $\cdot$ is a **group** if the following group axioms are satisfied

- ⇨**Closed**: $\forall a, b \in G, a \cdot b \in G$
- ⇨**Associative**: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- ⇨**Identity**: $\exists e \in G, \forall a \in G, a \cdot e = e \cdot a = a$
- ⇨**Inverse**: $\forall a \in G, \exists b \in G, a \cdot b = b \cdot a = e$

From these axioms, can show

- ⇨The identity is unique (how?)
- ⇨The inverse is unique (how?)

Examples?

- ⇨The set of integers under addition
- ⇨The set of positive rational numbers under multiplication

Topological Manifolds

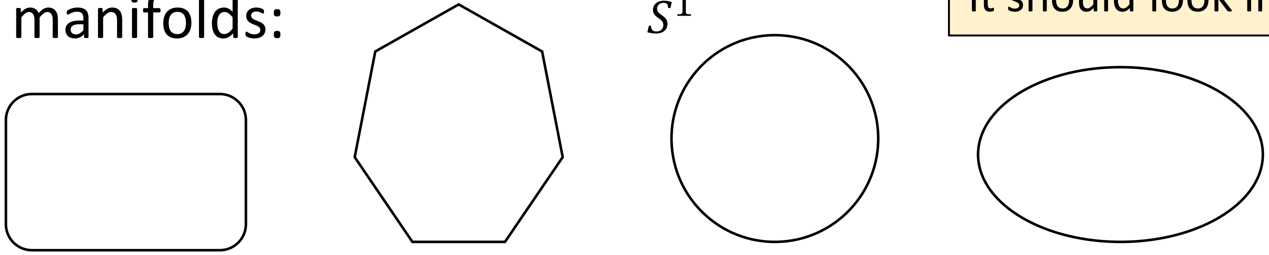
Homeomorphism: two spaces X and Y are **homeomorphic** if there is a continuous function $F: X \to Y$ that is bijective

Roughly speaking, an $\boldsymbol{n}$**-dimensional topological manifold** $\boldsymbol{M}$ is a space such that for $x \in M$, there exists a neighborhood U of x **homeomorphic** to $\mathbb{R}^n$

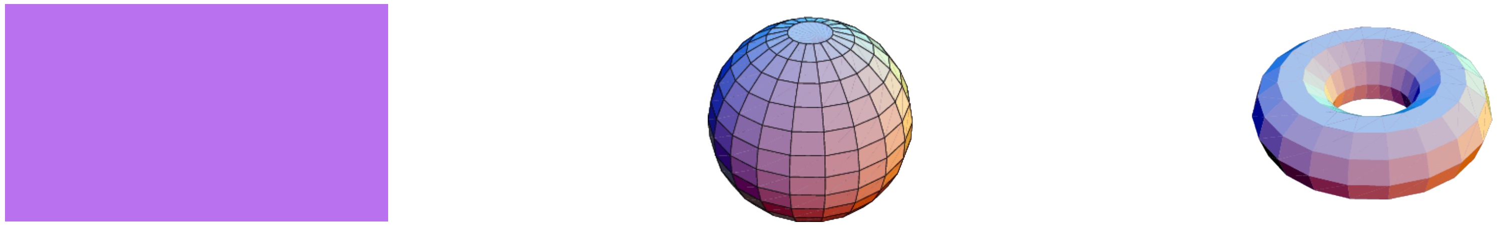
Alternative view: take any piece, and smash it… it should look like $\mathbb{R}^n$

1-dimensional manifolds:

$(a, b), \mathbb{R}$



2-dimensional manifolds: $\mathbb{R}^2, S^2, T^2, \ldots$

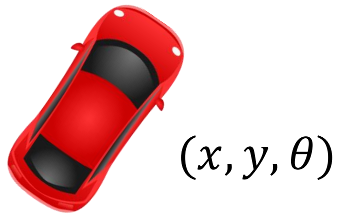


Why Topology and Manifolds?

Sensing, planning, and control are all related to manifolds

Robotics examples

- ⇨ A point robot in 2D take any position $x \in \mathbb{R}^2$
  - ⇨ This is also a group $E(2)$
  - ⇨ 2-dimensional Euclidean group
- ⇨ A car in 2D has one more dimension
  - ⇨ This is called $SE(2) = \mathbb{R}^2 \times S^1$
  - ⇨ $SE(2)$ reads: Special Euclidean group of dimension 2
  - ⇨ Yes, each point in the space is also a group element, just like $\mathbb{R}$ and $\mathbb{R}^2$
  - ⇨ Using (x, y, θ), can describe all possible positions of the car

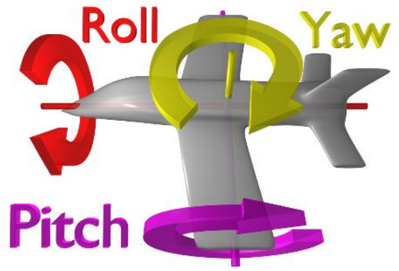


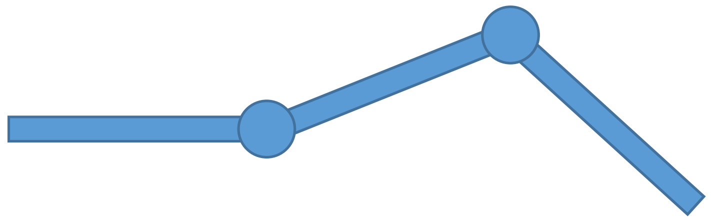
Why Topology and Manifolds? Continued

Robotics examples, continued

- ⇨A quadcopter is in a six-dimensional manifold
  - ⇨ Three positions (x, y, z)
  - ⇨ Three rotations $(yaw, pitch, roll)$
  - ⇨ This is $SE(3) = \mathbb{R}^3 \times SO(3)$
  - ⇨ Special Euclidean group of three dimensions
- ⇨A 2-link robot arm has a 2-dimensional manifold
  - ⇨ For rotations in the plane, this is T^2 (torus)
  - ⇨ Yes, a pose of such a robot arm corresponds to a point on a donut
- ⇨These are the **configuration spaces** of the robots
- ⇨More on this later

$(x, y, z, yaw, pitch, roll)$





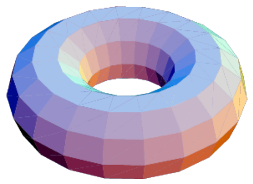
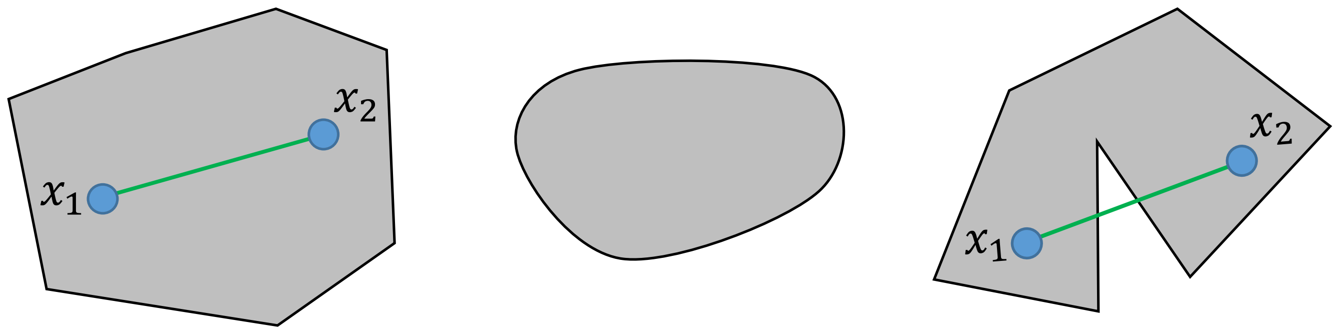


Image from wolfram wikipedia

Convexity

Convexity. In a Euclidean space, a set X is **convex** if given any $x_1, x_2 \in X$, all points on the straight-line segment $x_1 x_2$ belong to X.



Probability Essentials – Expectation

Expectation: the expected value of a random variable

- ⇨In the discrete case, for an RV X with n values $x_1, \dots, x_n$

$$E[X] = \sum_{1 \le i \le n} x_i P(X = x_i) = x_1 P(X = x_1) + \cdots + x_n P(X = x_n)$$

- ⇨This is also commonly known as the “mean” or “weighted average”
- ⇨E.g., the average score of this class
- ⇨E.g., single dice toss
  - ⇨ If we let $X\colon f_i \mapsto i$, that is, giving each face a number 1-6,
  - ⇨ Then $E[X] = 1 * \frac{1}{6} + \cdots + 6 * \frac{1}{6} = 3.5$

Linearity of Expectation

Linearity of Expectation: the expectation of an RV is the sum of the expectation of the component RVs

- ⇨Very handy in practice!
- ⇨Q: tossing a coin, how many tosses to get a first head, on average?
  - ⇨ The RV: # of tosses to get a first head
  - ⇨ Decompose
    - ⇨ Get a head in first toss: probability $\frac{1}{2}$
    - ⇨ Get a first head in second toss: $\frac{1}{4}$
    - ⇨ …
    - ⇨ Get a first head in n-th toss: $\frac{1}{2^n}$
  - ⇨ Apply linearity: $T = 1 * \frac{1}{2} + 2 * \frac{1}{4} + \cdots + n * \frac{1}{2^n} + \cdots = 2$

Linearity of Expectation, Continued

Q: tossing a coin, how many tosses to get both sides, on average?

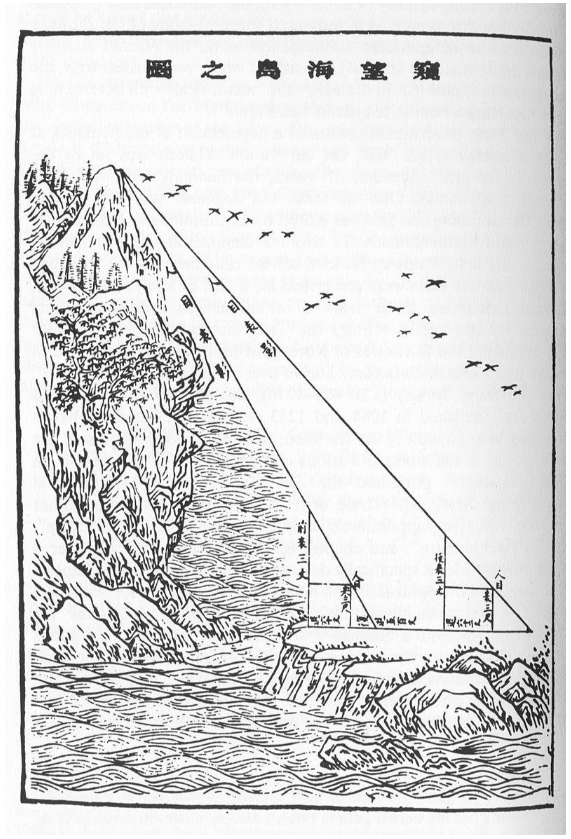
- ⇨The RV: # of tosses to get both sides
- ⇨Decompose:
  - ⇨ # of tosses to get a first side (doesn’t matter head or tail)
    - ⇨ What is this #?
    - ⇨ Yes, 1, because the first toss must produce a side
  - ⇨ # of tosses to get a different side
    - ⇨ What is this #?
    - ⇨ This is the same as asking for a specific side, like a head
    - ⇨ So the # is 2 from the previous calculation
- ⇨To total # of tosses to get both sides, in expectation, is $1 + 2 = 3$
- ⇨You should be able to generalize this to an n-sided coin

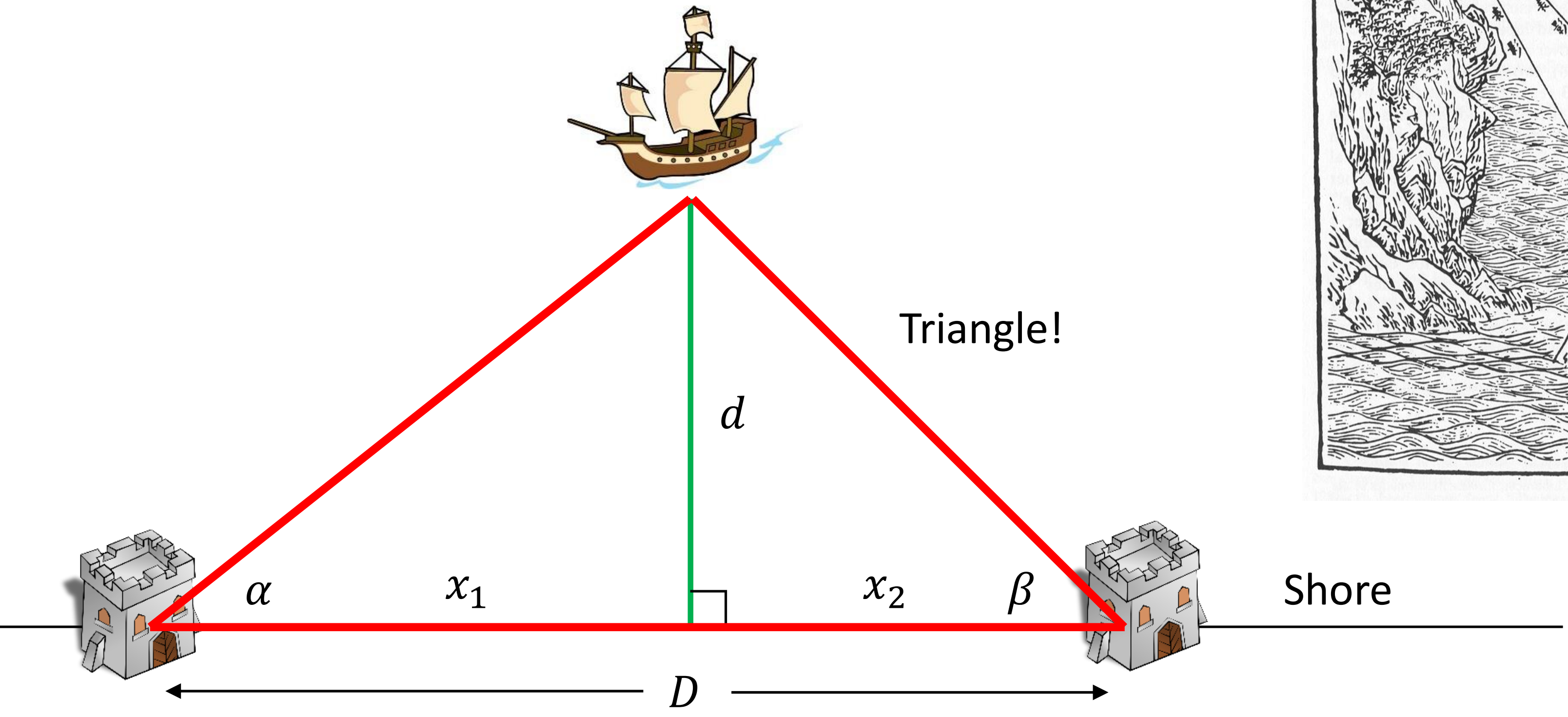
Localization with Triangulation

Triangulation is an ancient technique

⇨Known for at least 1700 years (Pei Xiu)

Straightforward principle





$$\tan\alpha = \frac{d}{x_1}, \tan\beta = \frac{d}{x_2}, x_1 + x_2 = D \quad \Rightarrow d = \frac{D}{\frac{1}{\tan\alpha} + \frac{1}{\tan\beta}}$$

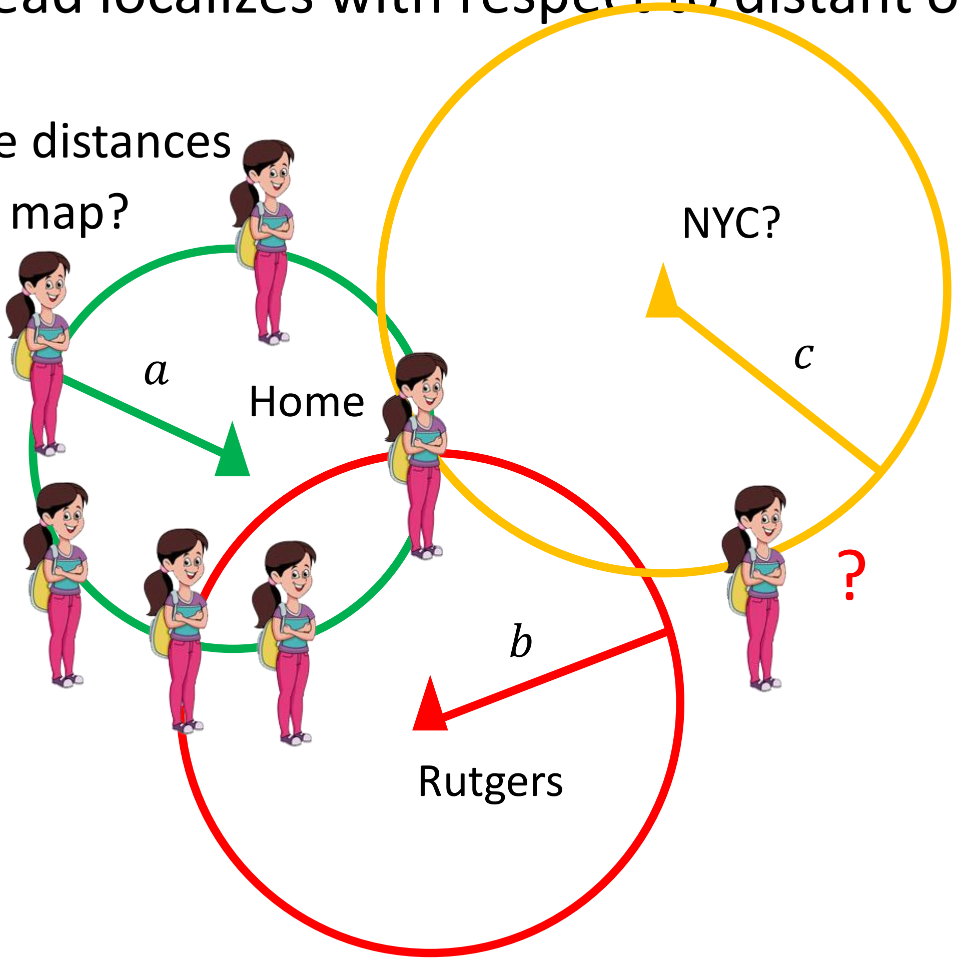
Image: Wikipedia

Localization with Trilateration

Triangulation locates the position of a distant object

Trilateration instead localizes with respect to distant objects

- ⇨2D example
- ⇨If we know the distances
- ⇨Where on the map?



How does Global Positioning System Work?

The principle is **trilateration**: determining absolute or relative location of points by **measurement of distance**

⇨We have seen 2-dimentional trilateration

⇨What about GPS? How many distances?

⇨GPS is three-dimensional

⇨4+ satellites!

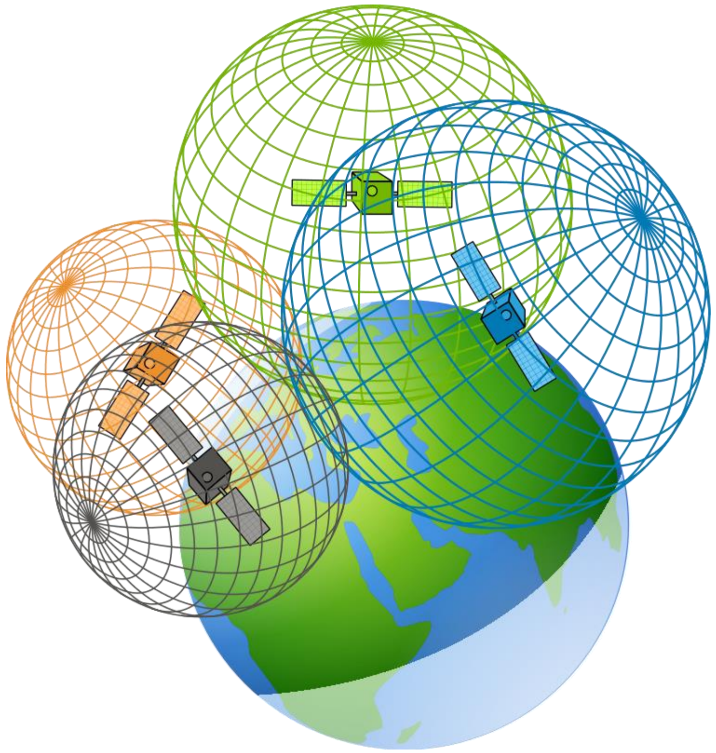
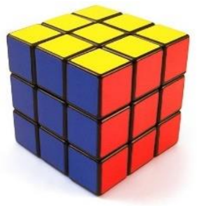
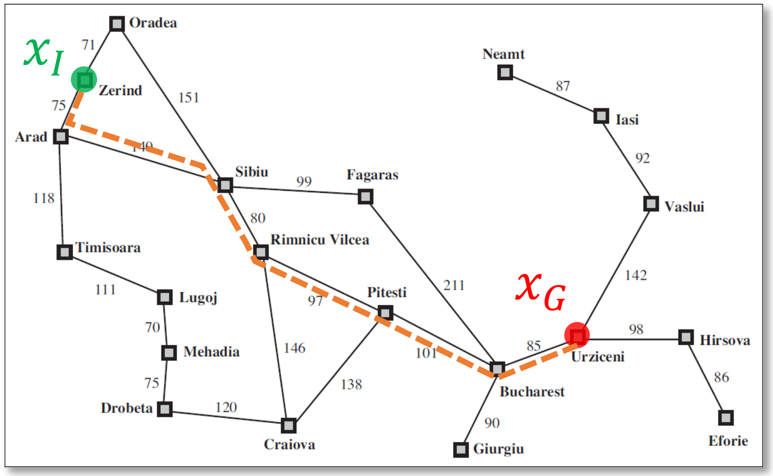


Image source: moorefamrsbg.org

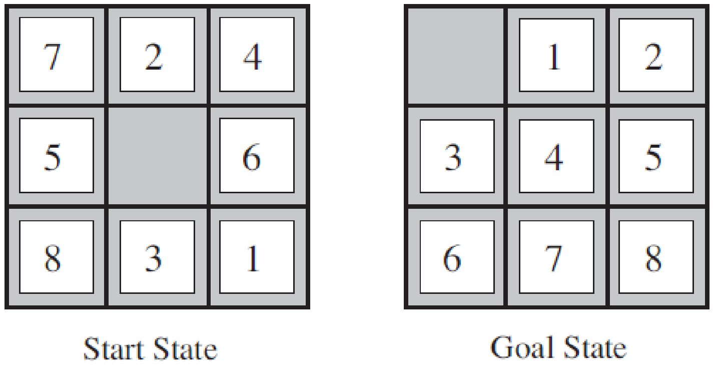
Components of a (General) Search Problem



- ➪**State space** S: in this case, an edge-weighted graph
- ➪**Initial** (start) and **goal** (final) states: x_I and x_G
  - ➪There can be more than one start/goal state: solve one side of a Rubik's cube
- ➪**Action**: in this case, moving from one state to a nearby state
- ➪**Transition model**: tuples (s_1, a, s_2) that are valid
  - ➪Sometimes written as $T(s_1, a) = s_2$
  - ➪There are usually costs/rewards associated with a transition, $R(s_1, a)$
- ➪**Solution**: valid transitions connecting x_I and x_G
  - ➪Optimal solution: solution with lowest cost (e.g., length of the path)



State Space Example: 8-puzzle



Start State

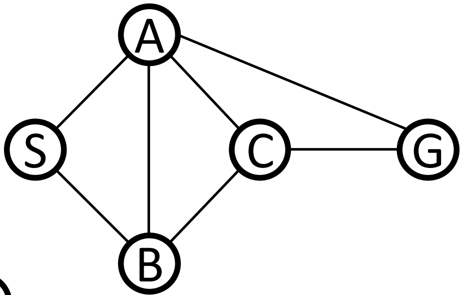
Goal State

- ⇨State space: arrangements of the 8 pieces
  - ⇨State space size: $9! = 362880$
- ⇨What if we have 1, 2, 3, 4, 5, 6, *, *?

Graph Basics

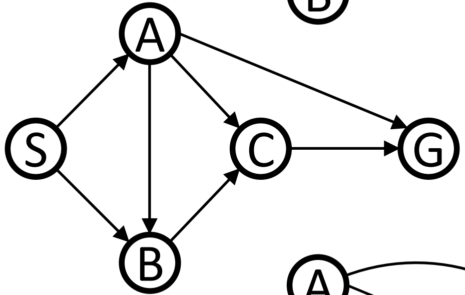
A graph $G = (V, E)$ is a set of vertices V and a set of edges E

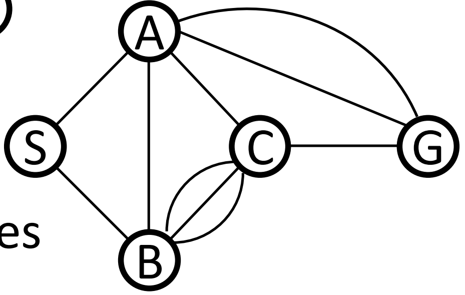
- ⇨Example
  - ⇨ $V = \{A, B, C, G, S\}$
  - ⇨ $E = \{(A,B), (A,C), (A,G), (A,S), (B,S), (B,C), (C,G)\}$



Variations

- ⇨A graph may be **directed**
- ⇨There can be **multi-edges** between two vertices
  - ⇨ This is called a **multi-graph**
  - ⇨ We will not consider multi-graphs in our course





Basic properties

- ⇨An undirected graph with n vertices has **at most** $\frac{n^2-n}{2}$ edges
  - ⇨ When this happens, the graph is a **complete** graph
- ⇨A graph is **connected** if there is a path between any two vertices
- ⇨A connected graph with $n - 1$ edges is a **tree**

A Generic Graph Search Algorithm

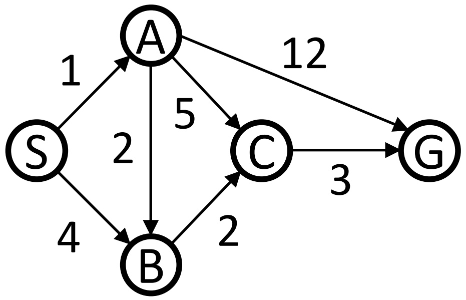
```
input: $G = (V, E), x_I, x_G$
AddToQueue($x_I$, $Queue$);  // Add $x_I$ to a queue of nodes to be expanded
while(!IsEmpty($Queue$))
    $x \leftarrow$ Front($Queue$);                // Retrieve the front of the queue
    if($x.expanded$ == true) continue;    // Do not expand a node twice
    $x.expanded$ = true;                  // Mark $x$ as expanded
    if($x == x_G$) return solution;       // Return if goal is reached
    for each neighbor $n_i$ of $x$ // Add all neighbors of to the queue
        if($n_i.expanded$ == false) AddToQueue($n_i$, $Queue$)
return failure;
```

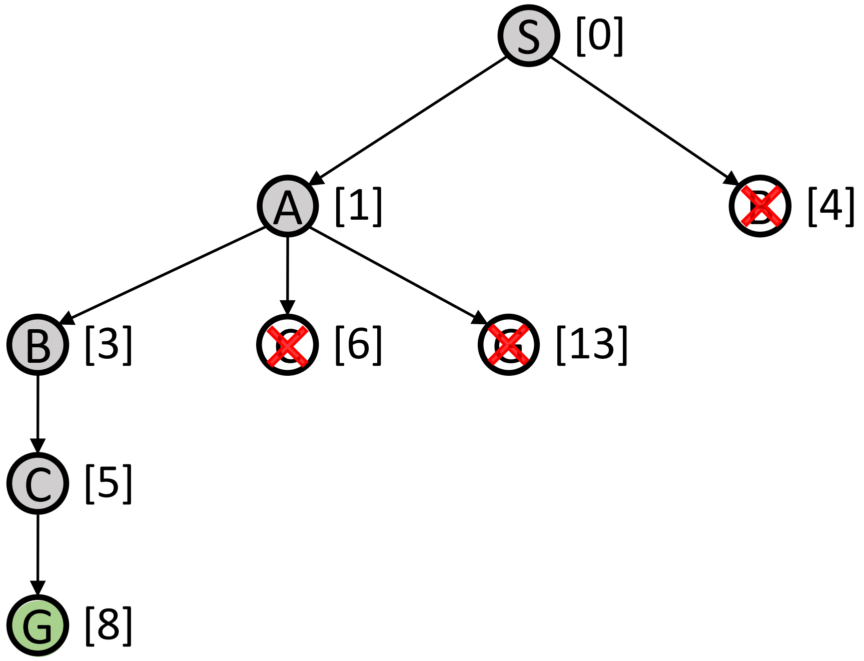
Different graph search algorithms (breadth first, depth-first, uniform-cost, …) differ at the function `AddToQueue`

To retrieve the actual path, use **back pointers**

Uniform-Cost Search

Maintain queue order based on current cost



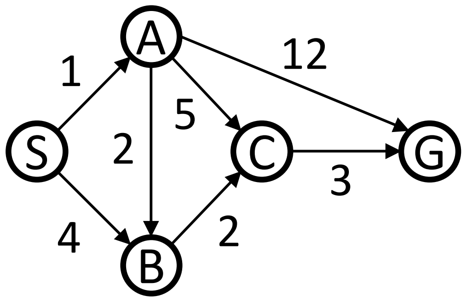


⇨Produces **optimal** path!

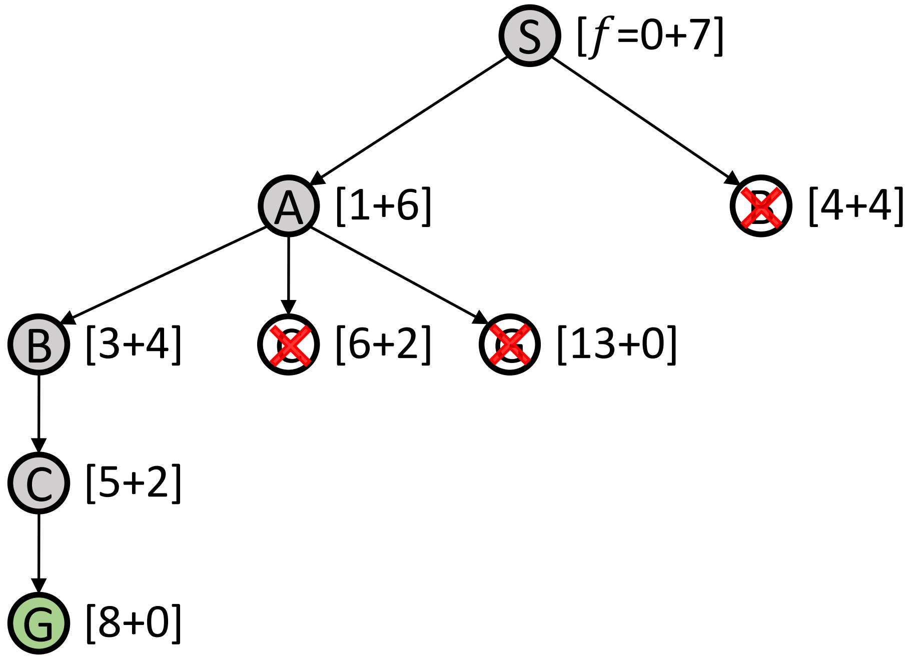
⇨This is basically the Dijkstra’s algorithm

A* Search

Maintain queue order based on current cost + guess



State	$h(x)$
S	7
A	6
B	4
C	2
G	0



A Generic Graph Search Algorithm

```
input: $G = (V,E), x_I, x_G$
AddToQueue($x_I$,$Queue$);  // Add $x_I$ to a queue of nodes to be expanded
while(!IsEmpty($Queue$))
    $x \leftarrow$ Front($Queue$);                // Retrieve the front of the queue
    if($x.expanded$ == true) continue;    // Do not expand a node twice
    $x.expanded$ = true;                  // Mark $x$ as expanded
    if($x == x_G$) return solution;       // Return if goal is reached
    for each neighbor $n_i$ of $x$ // Add all neighbors of to the queue
        if($n_i.expanded$ == false) AddToQueue($n_i$,$Queue$)
return failure;
```

A*: `AddToQueue(x)` uses $f(x) = g(x) + h(x)$

⇨ $g(x)$: the **current best cost** from start node x_I to node x

⇨ $h(x)$: the estimated cost from x to goal x_G

⇨ $g(x)$ is **cost-to-come**, $h(x)$ is a **heuristic**

⇨ The unprocessed node with the smallest $f(x)$ is placed in the front of the queue

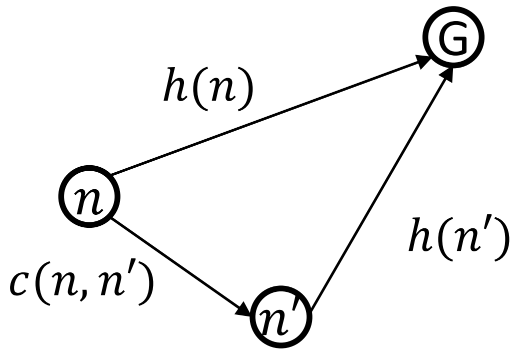
Admissible and Consistent Heuristic

- Assume the cheapest path from x to a goal is $c(x)$, an **admissible heuristic** satisfies

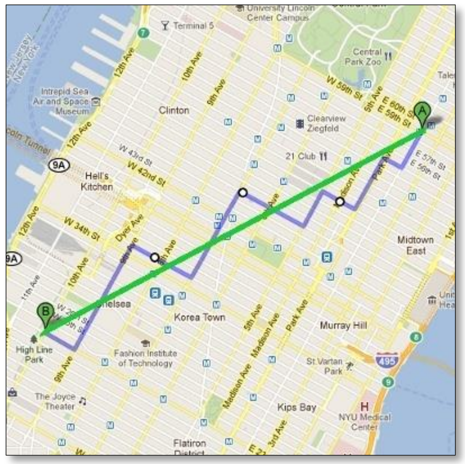
$$\boxed{h(x) \leq c(x)}$$

- A **consistent** heuristic is defined as

$$\boxed{h(n) \leq c(n, n') + h(n')}$$



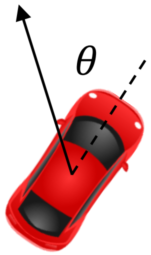
  - A form of triangle inequality
- A **consistent** heuristic is **always admissible**
  - The reverse is not always true
- Example of heuristic functions
  - Manhattan distance
  - Straight-line distance
    - Consistent



Why the Configuration Space?

A powerful abstraction for solving **motion planning** problems

- ⇨Motion planning is to find feasible motions for robots to go from x_I to x_G
- ⇨This is non-trivial, e.g., how to plan for parallel parking a car?

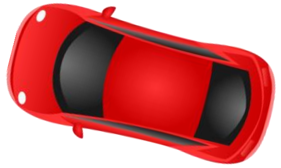
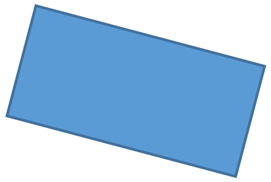


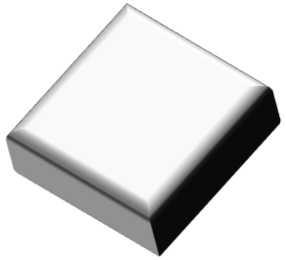
  - ⇨ A hard problem for many drivers!
  - ⇨ And this is just a problem in 2D/3D!
- ⇨Obviously, the position and the orientation must be changed together
- ⇨With C-space, this becomes **searching for a path** in the joint space of 2D position $(x, y) \in \mathbb{R}^2$ and rotation $\theta \in S^1$

Modeling Robot as Linked Rigid Bodies

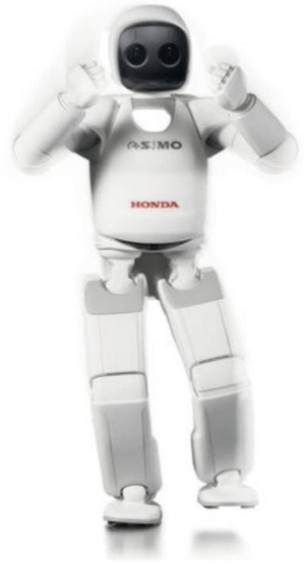
Common robot models

- ⇨A single point (point robot)
- ⇨A single rigid body



- ⇨Multiple rigid bodies (**links**) joined with **joints**



DOF and Types of Joints

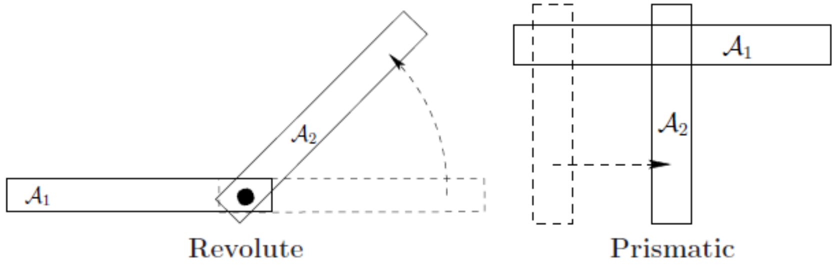
Configuration: specification of where all pieces of a robot are

Degrees of freedom (dof): the smallest number of real-valued (i.e., continuous) coordinates to fully describe configurations of a robot

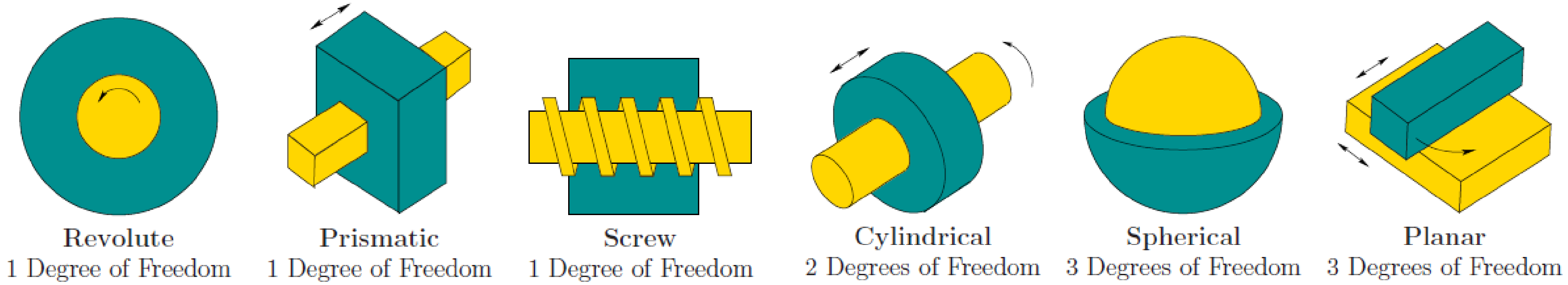
⇨More on this later

Types of joints

⇨2D



⇨3D



Robots generally are viewed as rigid bodies joined by joints

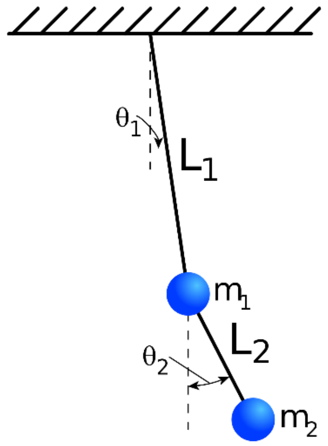
Image source: Planning Algorithms

Examples

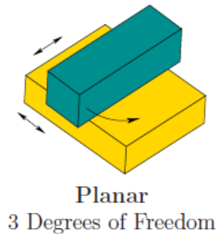
Train

A fan blade

Door



Double pendulum

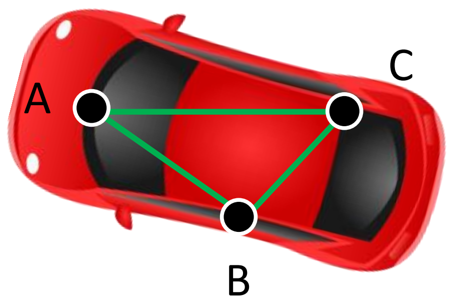


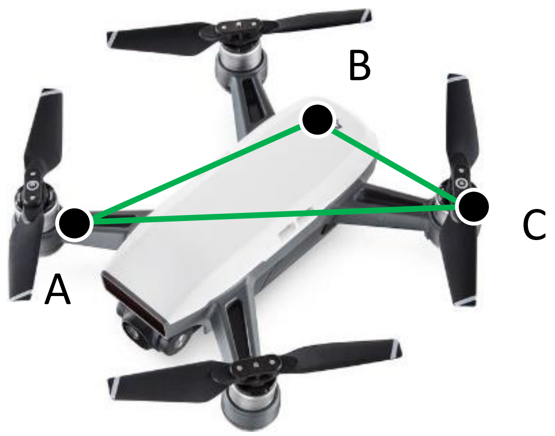
Coin lying flat on a table

Coin on edge

DOF for a Single Rigid Body

The position is fully determined by three fixed points on the body





General formula: DOF = total DOF of points - # of constraints

⇨Car: 2 x 3 – 3 = 3

⇨Quadcopter: 3 x 3 – 3 = 6

Alternatively, can do this incrementally

⇨For the car, A has 2 dofs

⇨Once A is fixed, because d_{AB} is fixed, B has 1 extra dof

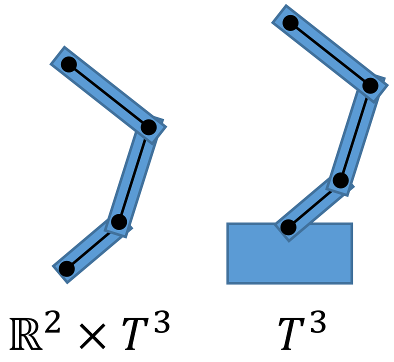
⇨For fixed AB, C is fixed, so 0 extra dof

⇨What about a quadcopter?

Determining the DOF for General Robots

2D chains

- ➪Base link is 3D ($\mathbb{R}^2 \times S^1$)
- ➪If fixed, then often 1D
- ➪Adding joints generally adds one more dimension



3D chains

- ➪Base link is 6D ($\mathbb{R}^3 \times SO(3)$)
- ➪If fixed, depending on the joint
- ➪Then add the DOF of each additional joint

Closed chains

$$DOF = N(k-1) - \sum_{i=1}^{n}(N - f_i) = N(k-n-1) + \sum_{i=1}^{n} f_i$$

- ➪We have a formula!
- ➪N: 6 for 3D, 3 for 2D
- ➪k: # of links (including the ground link)
- ➪n: the number of joints
- ➪f_i: DOF of the joint
- ➪Examples
  - ➪ 2D, 3 links
  - ➪ 2D, 4 links
  - ➪ 2D, 6 links

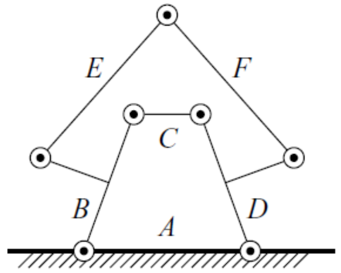
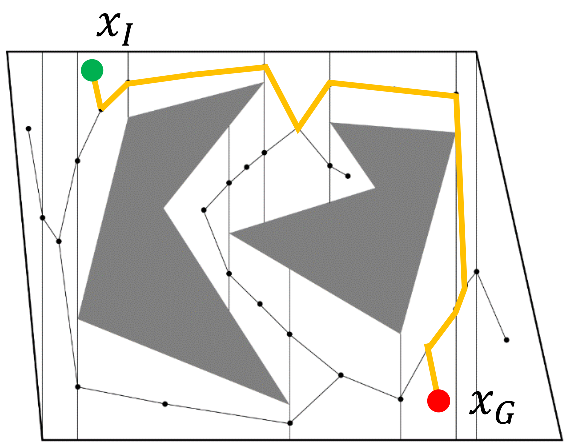


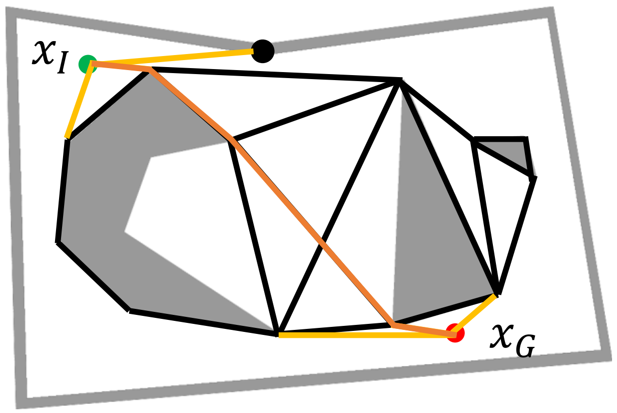
Image source: Principles of Robot Motion

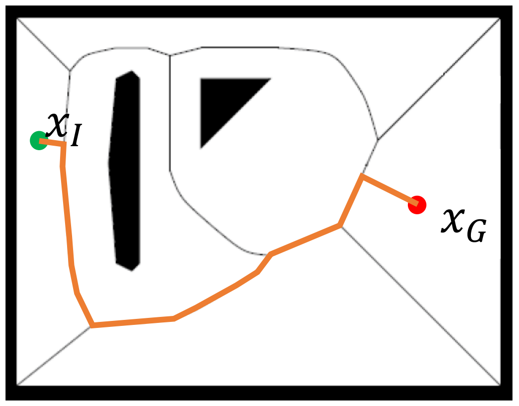
Combinatorial Motion Planning in the Plane

Last time, we covered several **combinatorial motion planning** algorithms in the plane

- ⇨Vertical cell decomposition
- ⇨Shortest-path roadmaps
- ⇨Maximum clearance roadmaps







What do these have in common?

- ⇨Each provides a (**combinatorial**) partitioning of the environment
- ⇨Which makes these algorithms **complete**

Implications of the Halting Problem

So, are all algorithms complete?

- ⇨No!
- ⇨Proof sketch
  - ⇨ There exist algorithms which we **cannot tell whether they will stop**
  - ⇨ Such algorithms **may run forever** and there is nothing we can do
  - ⇨ Such algorithms/programs are not complete
- ⇨In practice, this can be bad
  - ⇨ E.g., real time systems
  - ⇨ Solution: do not use full Turing machine

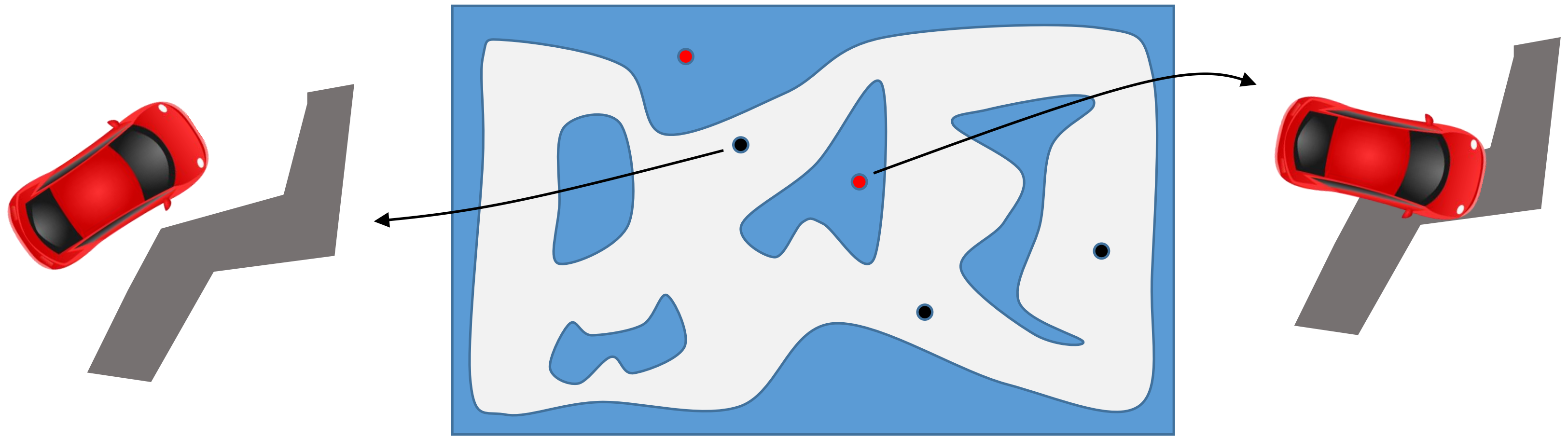
Combinatorial algorithms **are** complete

- ⇨This is because every single point in C_{free} is covered
- ⇨This is a big deal – a piece of mind
- ⇨Motivates the development of combinatorial methods for higher dimensions

Key Components of Sampling-Based Planning

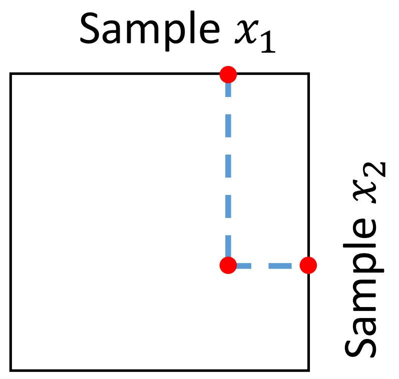
Sampling-based planning requires several important subroutines

- ⇨An **efficient sampling routine** is needed to generate the samples. These samples should **cover** C_{free} well in order to be effective
- ⇨**Efficient nearest neighbor search** is necessary for quickly building the roadmap: for each sample in C_{free} we must find its k-nearest neighbors
- ⇨The neighbor search also requires a **distance metric** to be properly defined so we know the distance between two samples
  - ⇨ This can be tricky for certain spaces, e.g., $SE(3)$
- ⇨**Collision checking** - Note that C_{free} is not computed explicitly so we actually are checking collisions between a complex robot and a complex environment



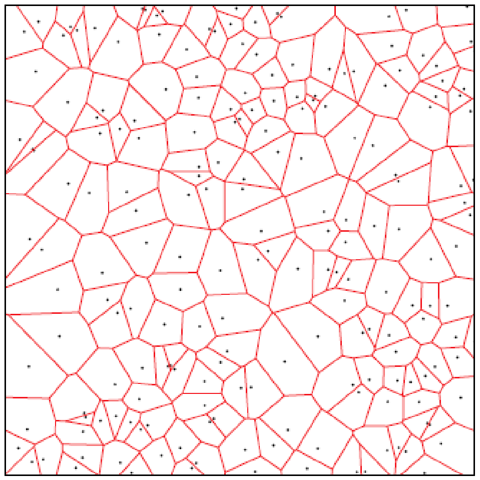
Sampling Routine

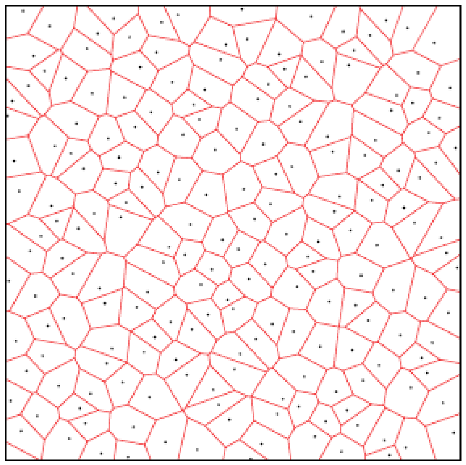
The simplest way of achieving this: **uniformly random sampling**

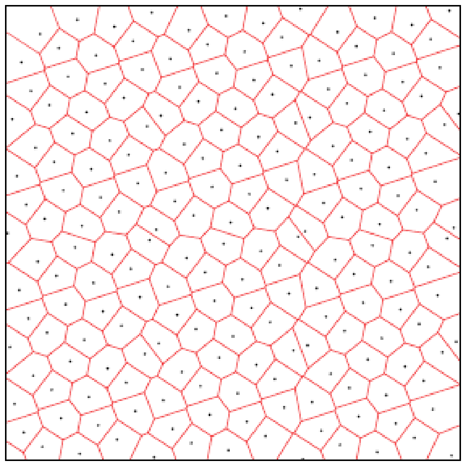


➡ A sample $(x_1, x_2) \in \mathbb{R}^2$

Generally, **incremental, dense** sampling w/ good **dispersion**


(a) 196 pseudorandom samples


(a) 196 Halton points

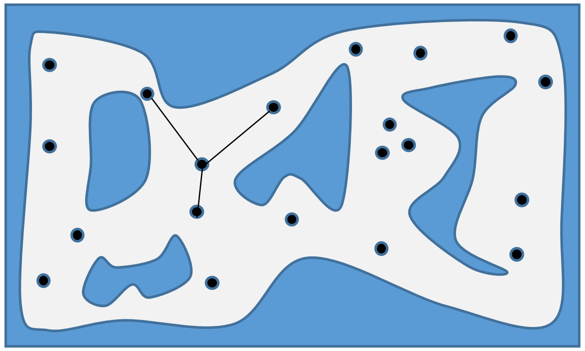

(b) 196 Hammersley points

Nearest Neighbor Search w/ k-d Tree

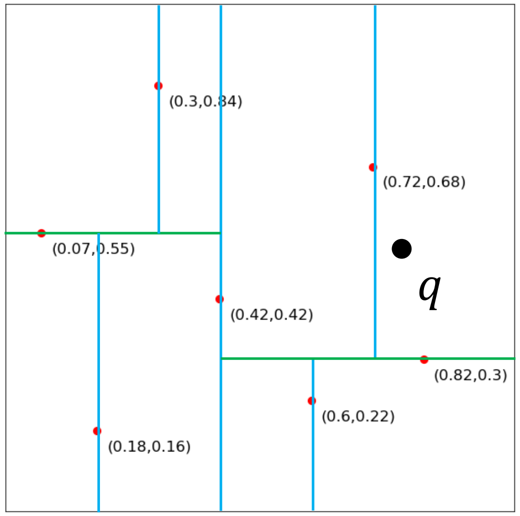
Connecting the samples

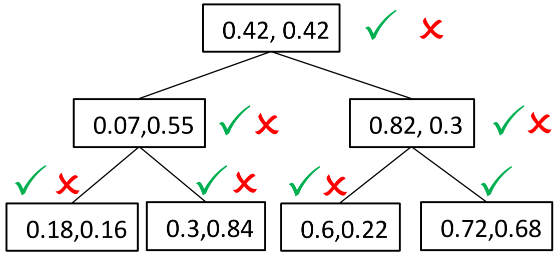
⇨Building the graph requires connecting the samples

⇨Need efficient methods for this



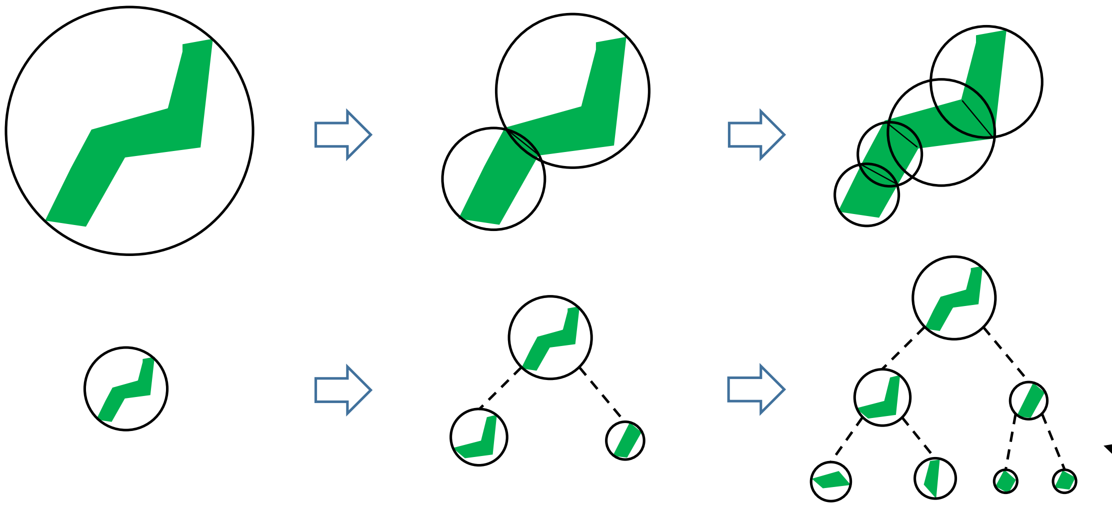
k-d Tree





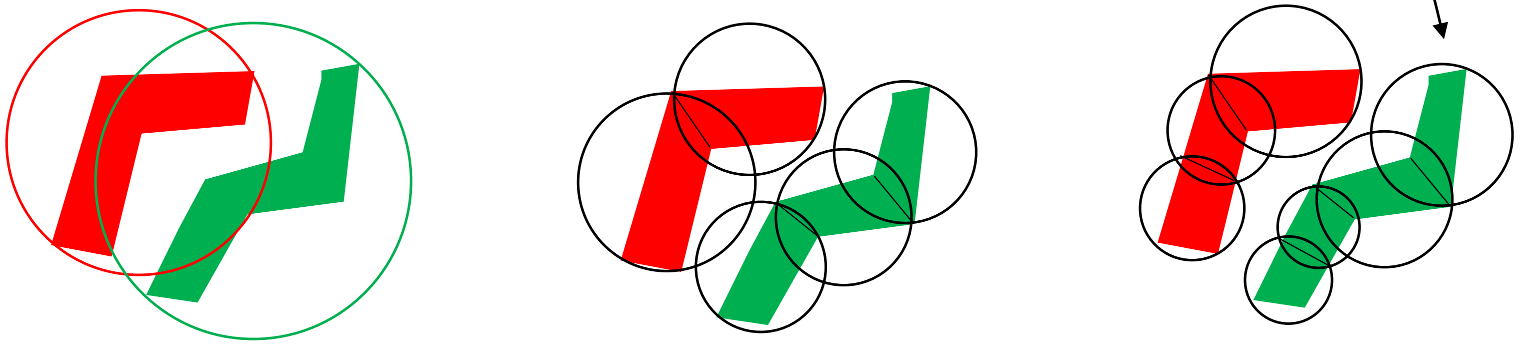
BVH for Collision Checking

BVH (Bounded Volume Hierarchy) breaks complex objects into pieces

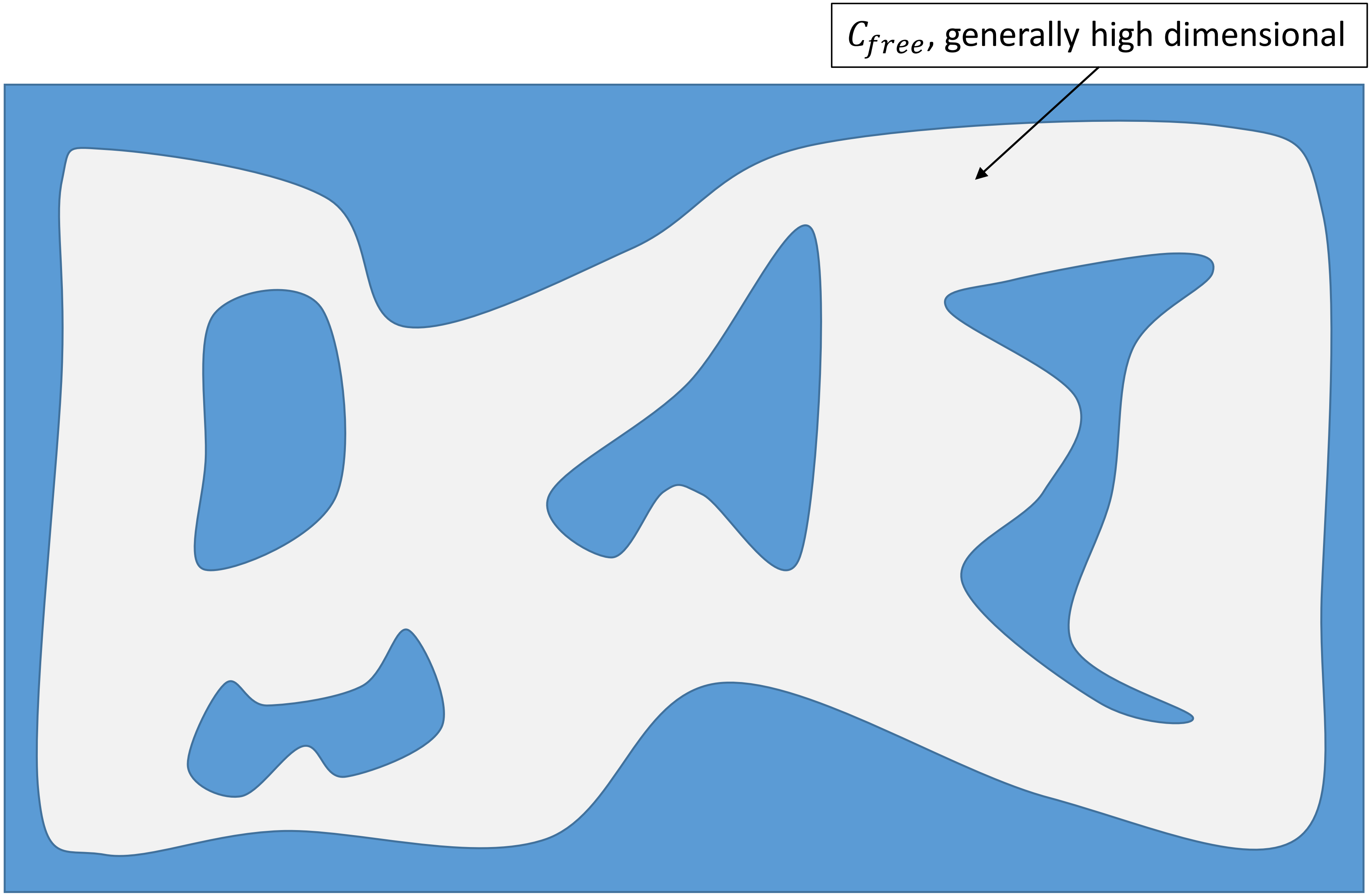


For collision checking, it works with two BVHs

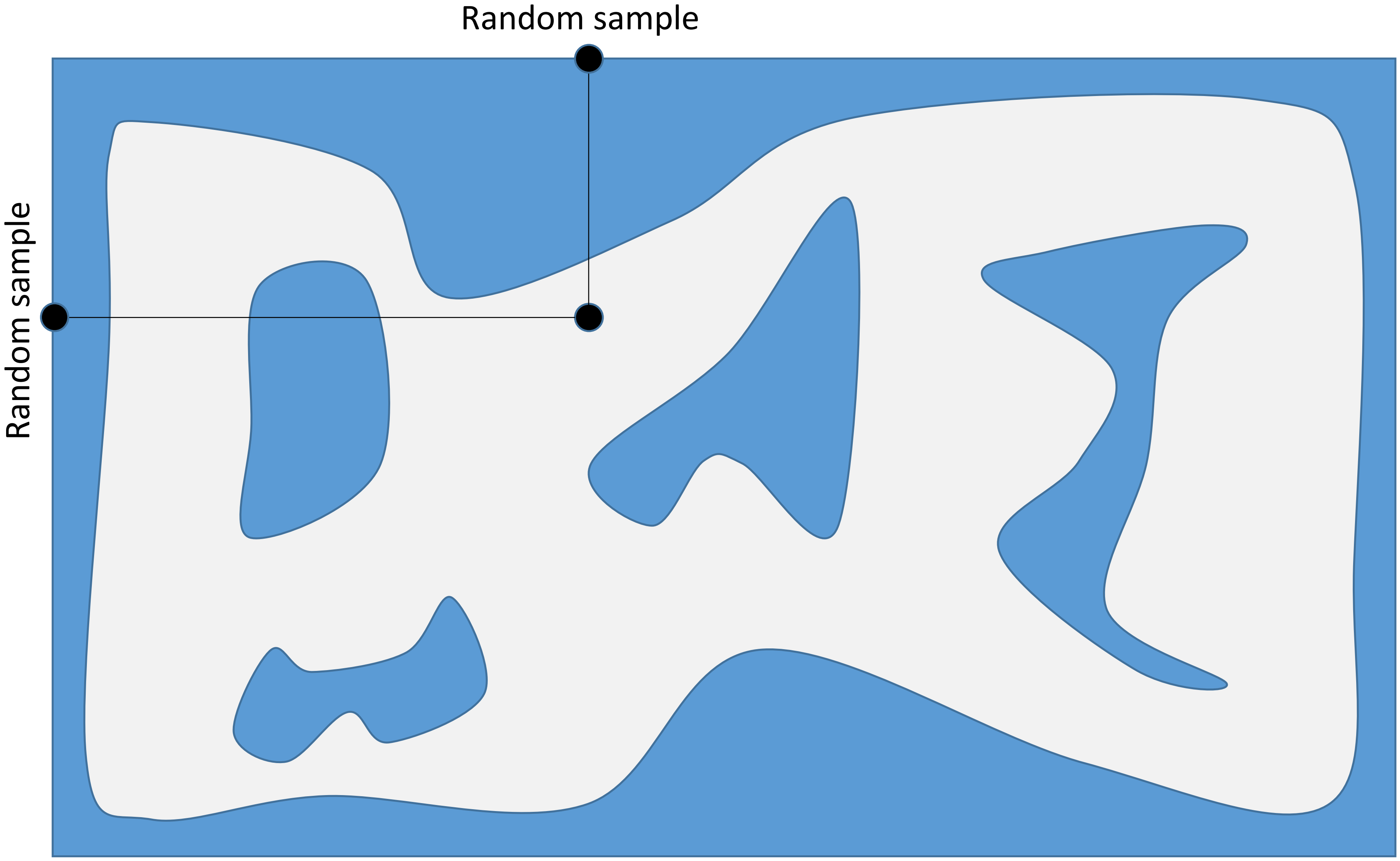
- ⇨BVs collide ⇒ possible collision
- ⇨BVs not colliding ⇒ no collision



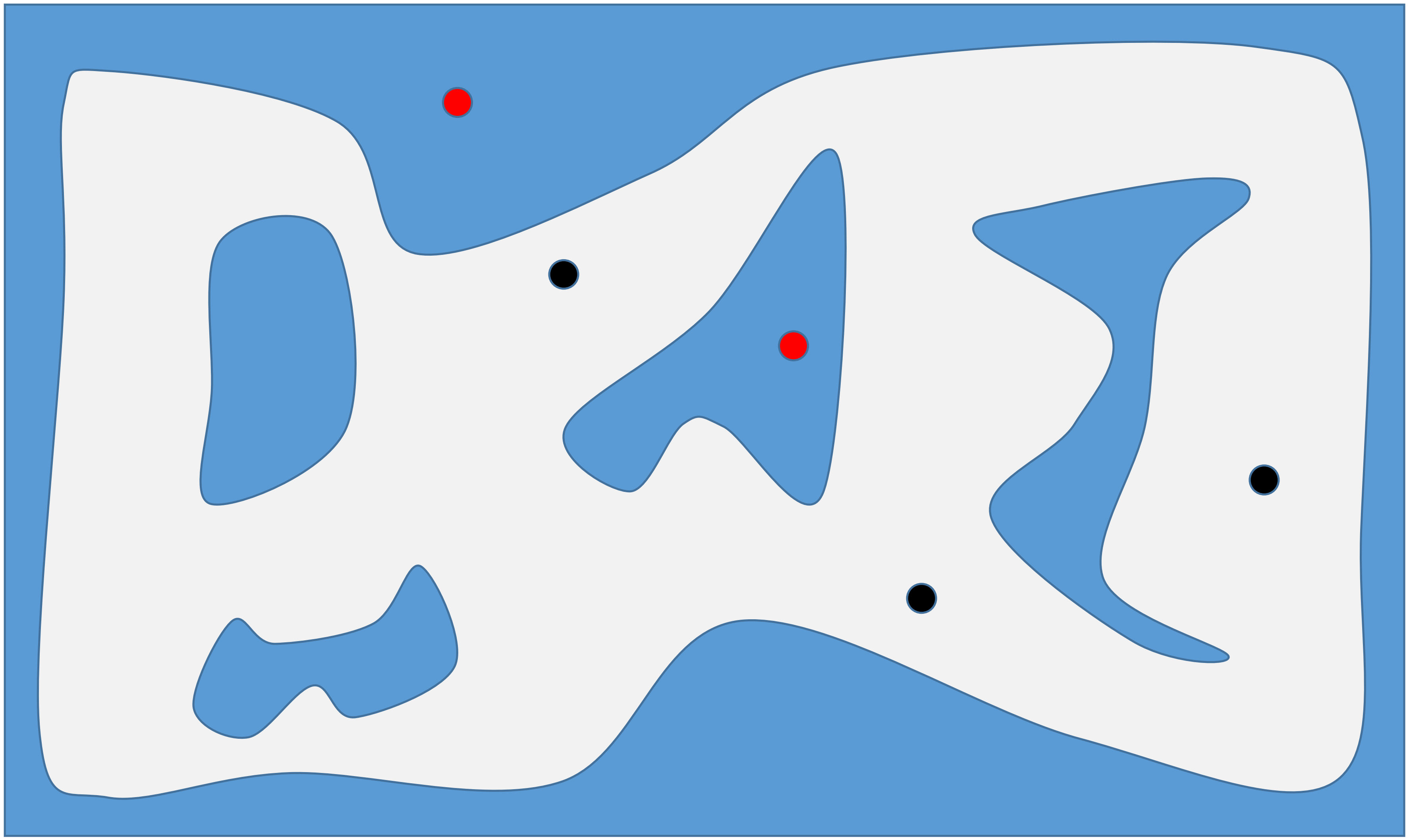
Probabilistic Roadmap in More Detail



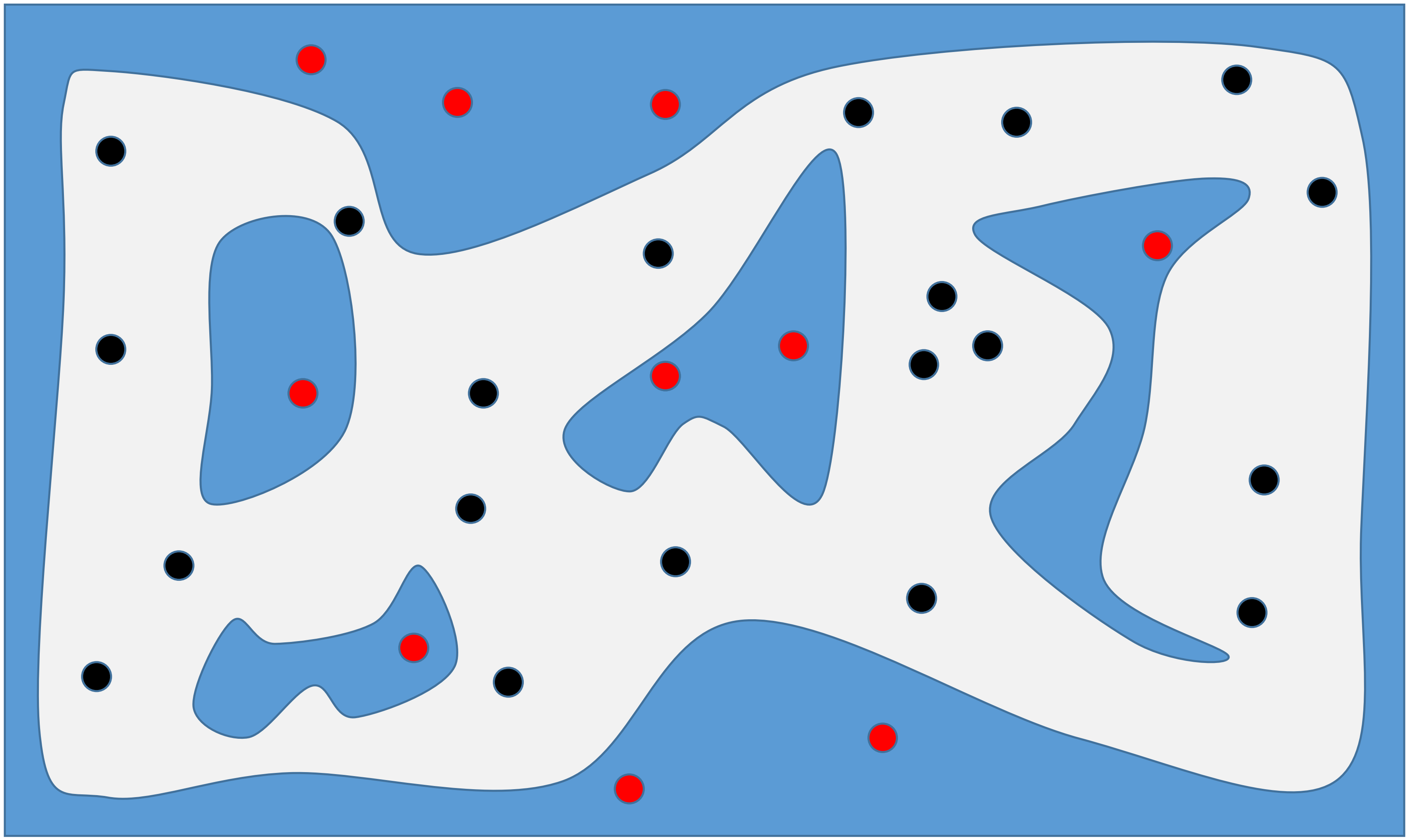
Generating Random Samples



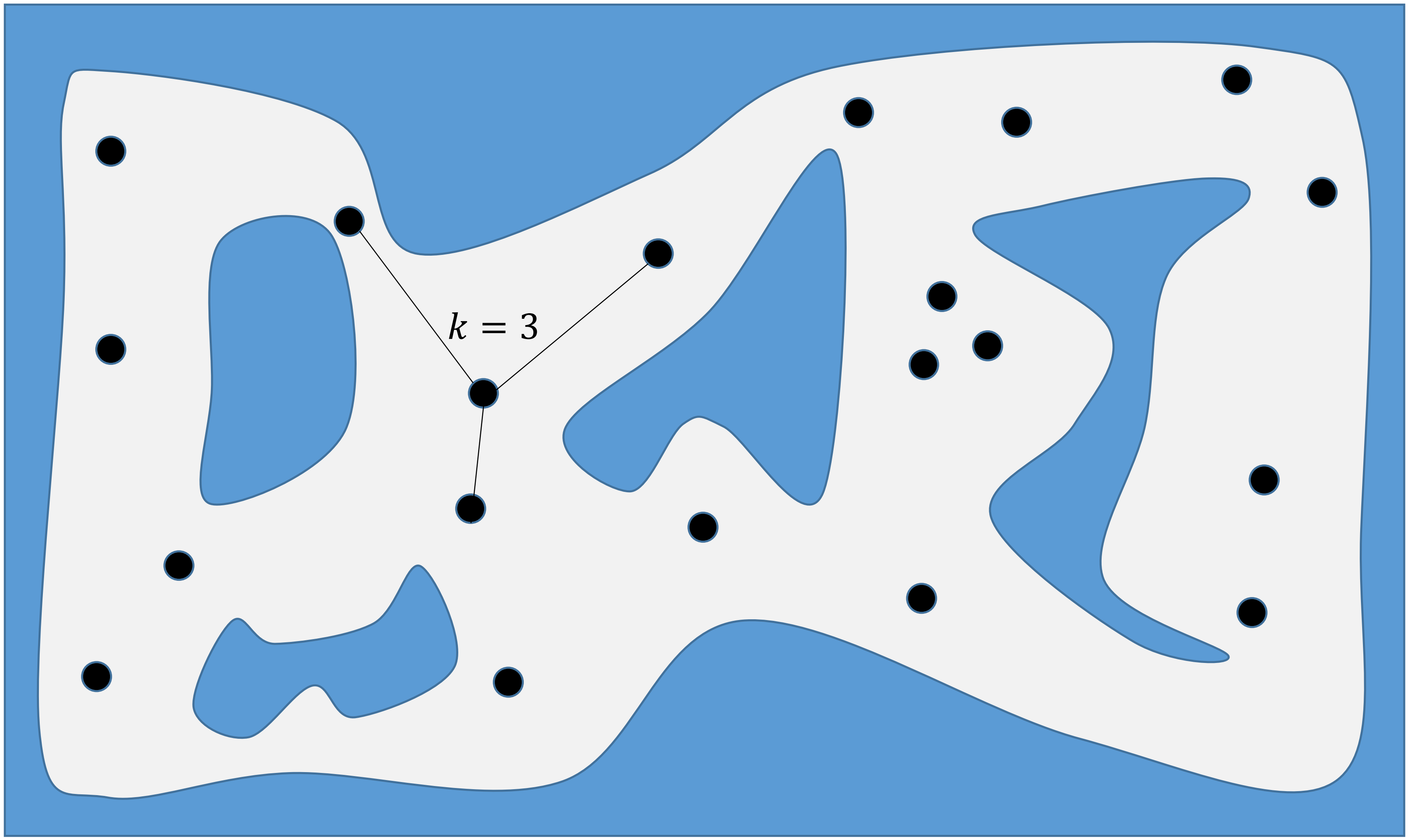
Rejecting Samples Outside C_{free}



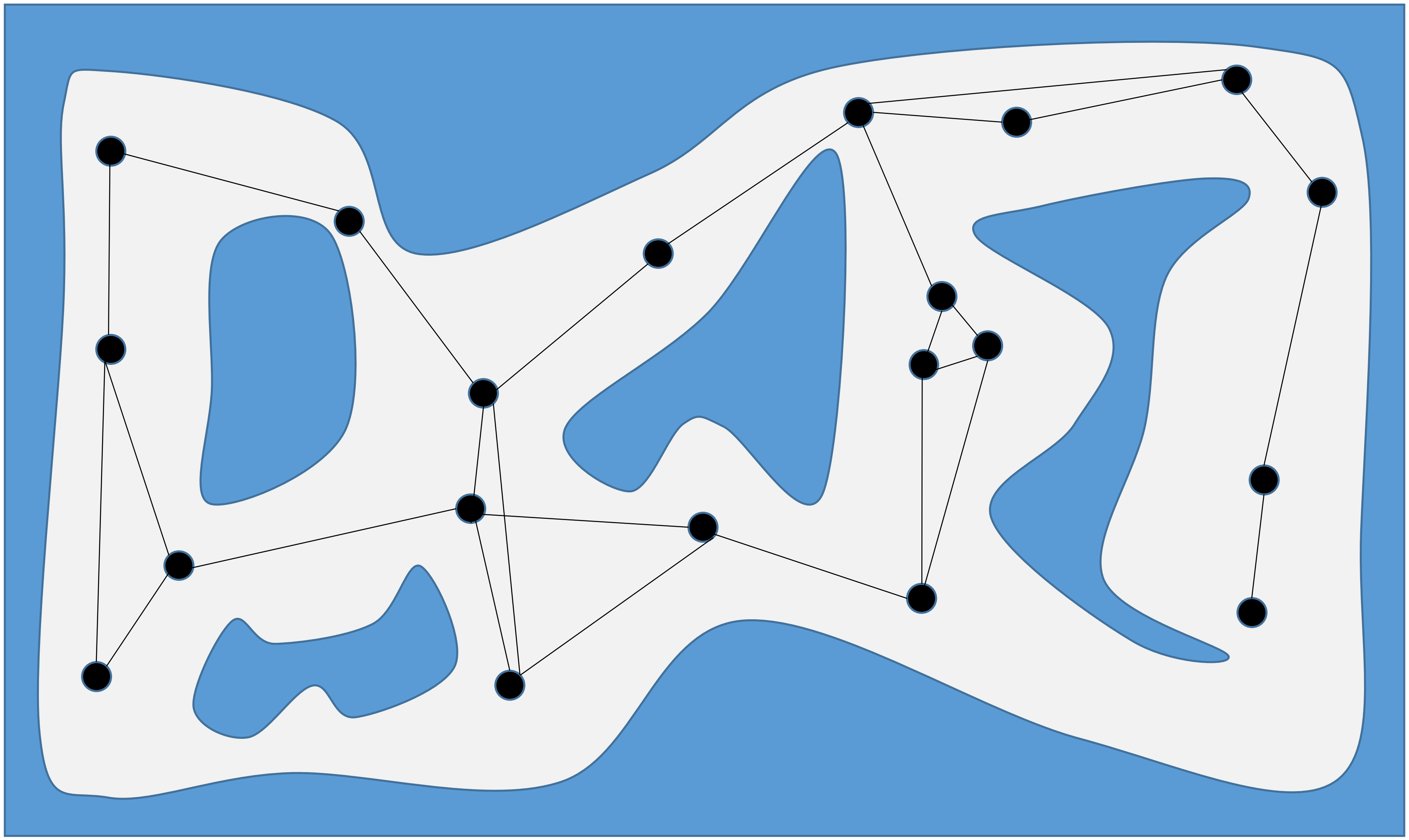
Collecting Enough Samples in C_{free}



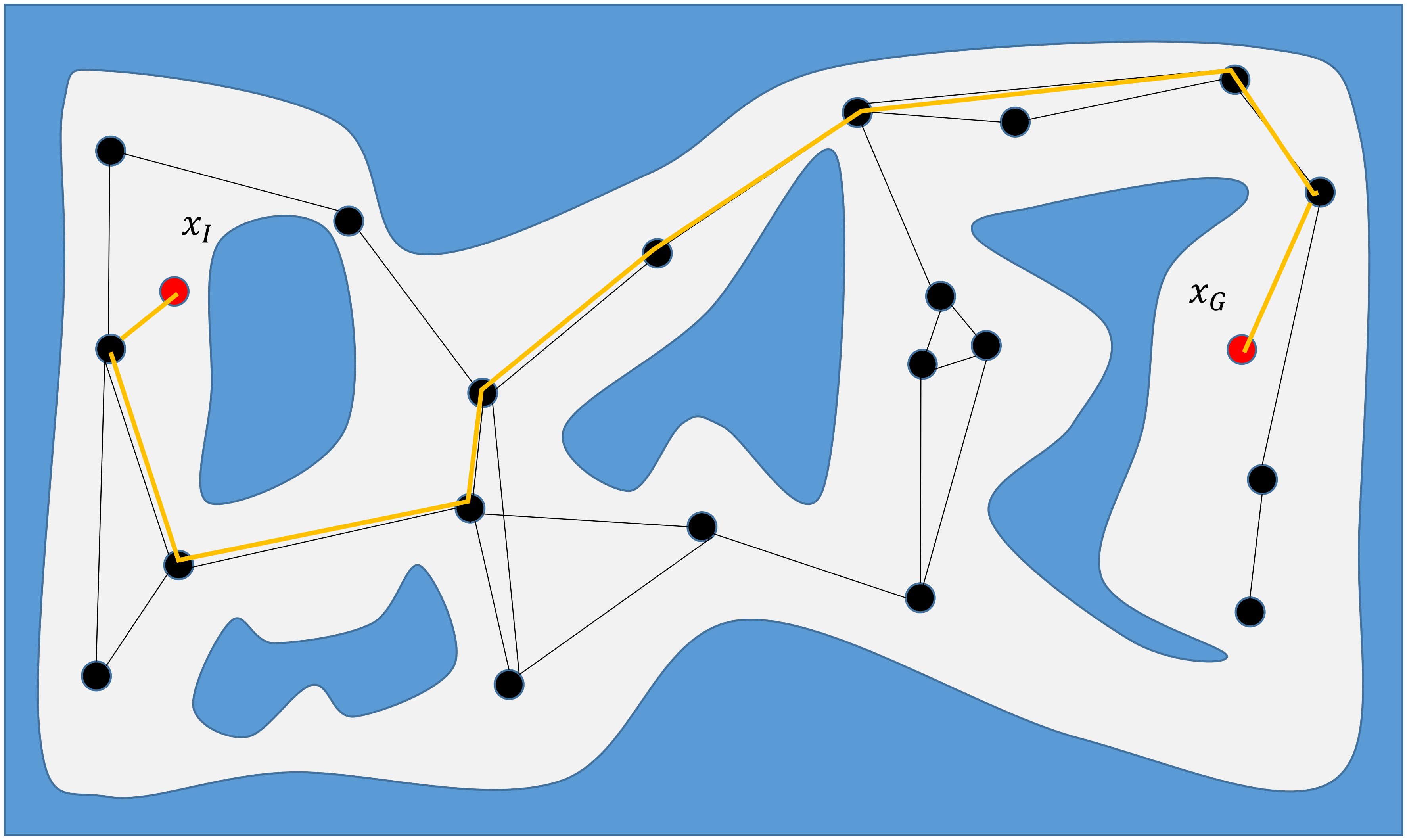
Connect to k Nearest Neighbors (If Possible)



Connect to k Nearest Neighbors (If Possible)



Query Phase

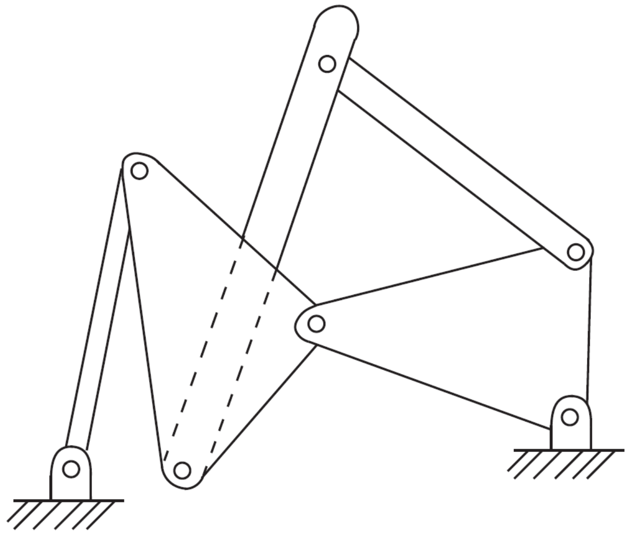


Understand Homework

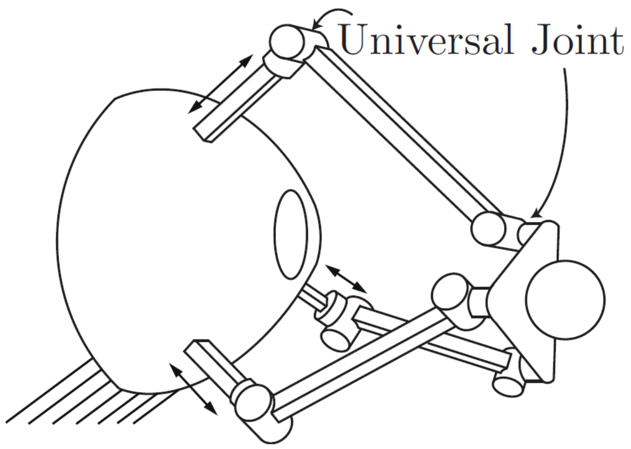
You should understand HW solutions

Focus on these that do not require you to do heavy computation

Examples – DoF Computation



$$\boxed{DOF = N(k-1) - \sum_{i=1}^{n}(N - f_i) = N(k-n-1) + \sum_{i=1}^{n} f_i}$$


Universal Joint