CS 336/436 – Algorithms for Sensor-based Robotics

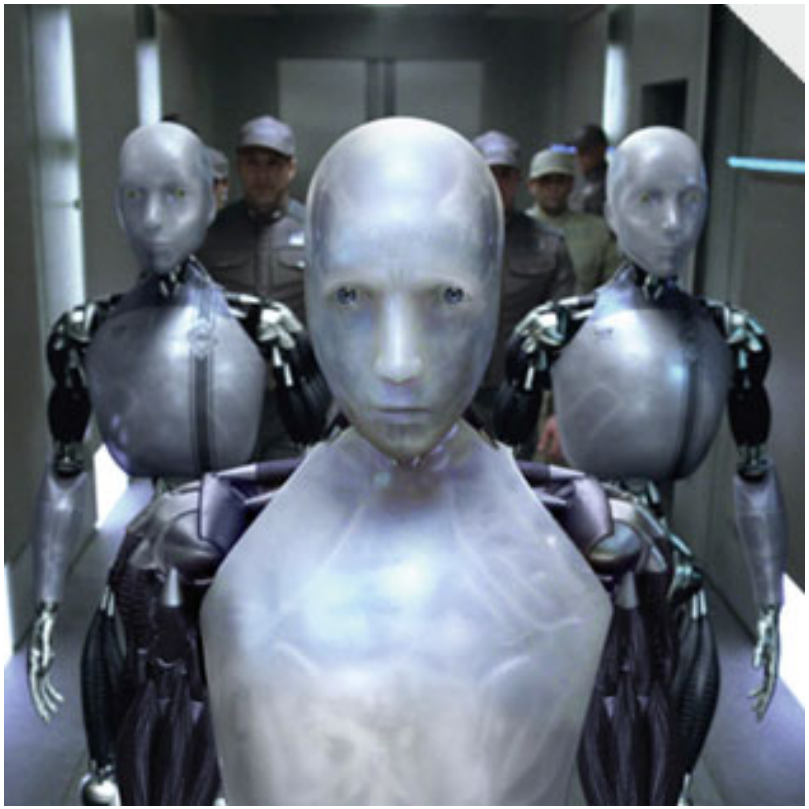
Lecture I – Introduction and Course Overview

Erion Plaku

Department of Computer Science
Laboratory for Computational Sensing and Robotics
Johns Hopkins University

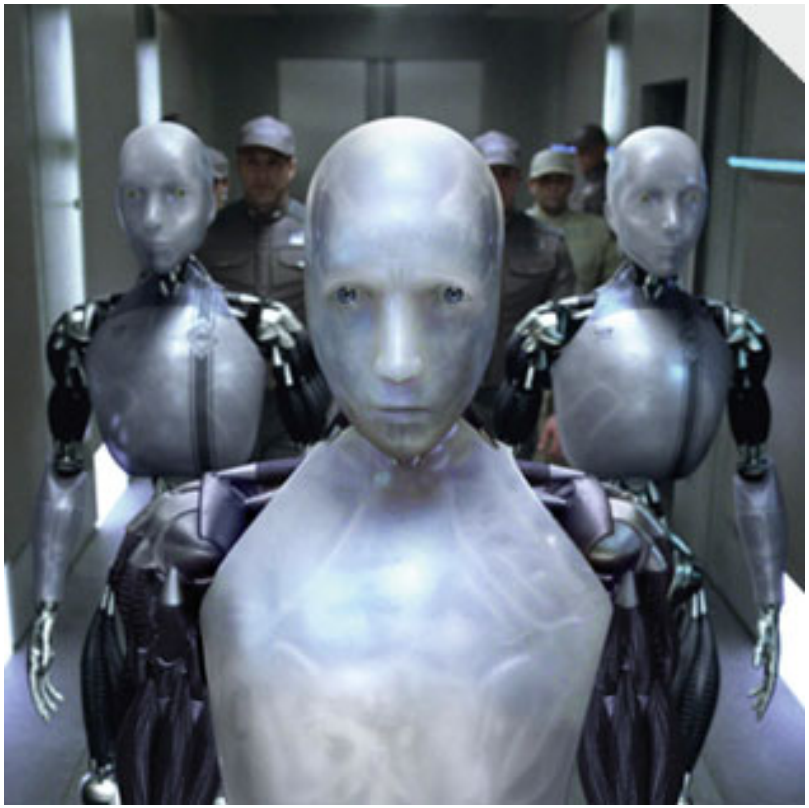
January 26, 2010

What is a Robot?



I, Robot (2004)

What is a Robot?



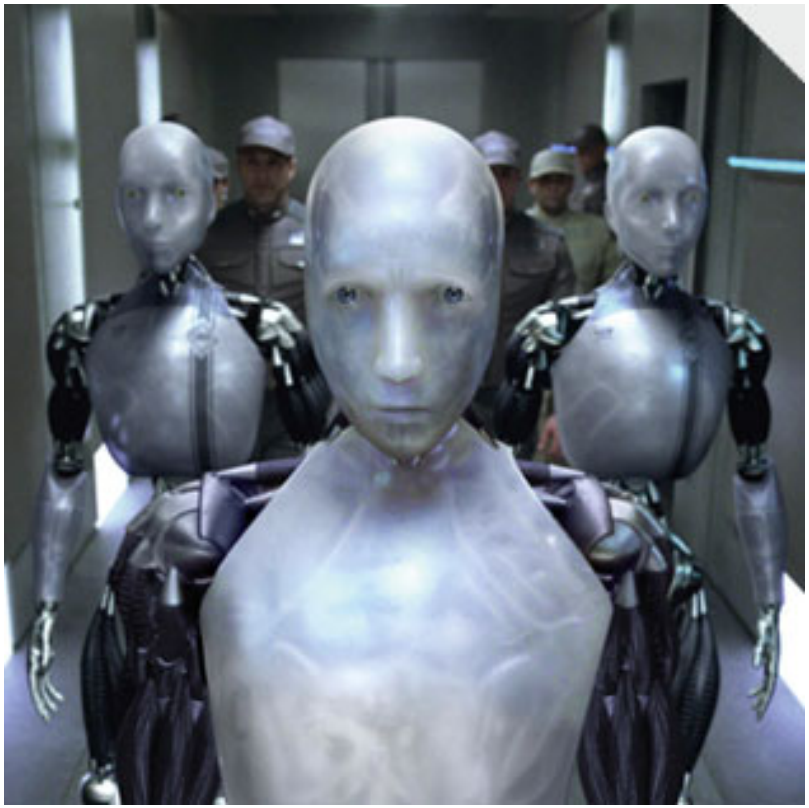
I, Robot (2004)

Automaton (Greek, autos "self" + matos "thinking, animated, willing")

http://www.etymonline.com

- from English translation of 1920 play "Rossum's Universal Robots" by Karel Capek
- from Czech robotnik (slave), robota (forced labor, drudgery), robotiti (to work, drudge)
- from Slavic (arabeit) related to German Arbeit (work)
- Word coined by Capek's brother Josef, who used it initially in a short story
- Robotics coined in 1941 in a science fiction context by Isaac Asimov

What is a Robot?



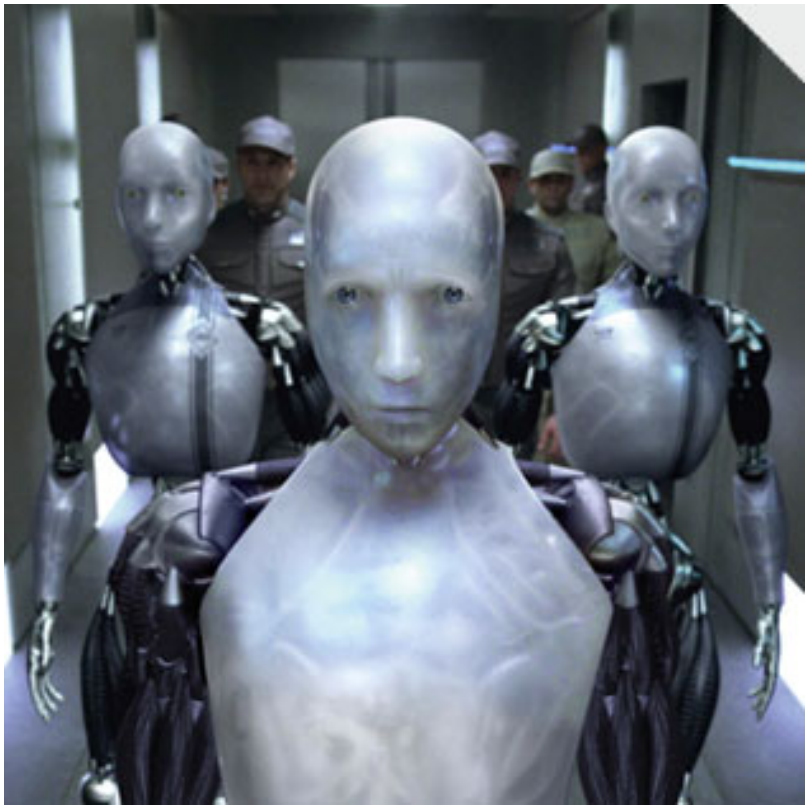
I, Robot (2004)

Automaton (Greek, autos "self" + matos "thinking, animated, willing")

http://www.etymonline.com

- from English translation of 1920 play "Rossum's Universal Robots" by Karel Capek
- from Czech robotnik (slave), robota (forced labor, drudgery), robotiti (to work, drudge)
- from Slavic (arabeit) related to German Arbeit (work)
- Word coined by Capek's brother Josef, who used it initially in a short story
- Robotics coined in 1941 in a science fiction context by Isaac Asimov
- Robotics Institute of America: re-programmable multi-functional manipulator designed to move materials, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks

What is a Robot?



I, Robot (2004)

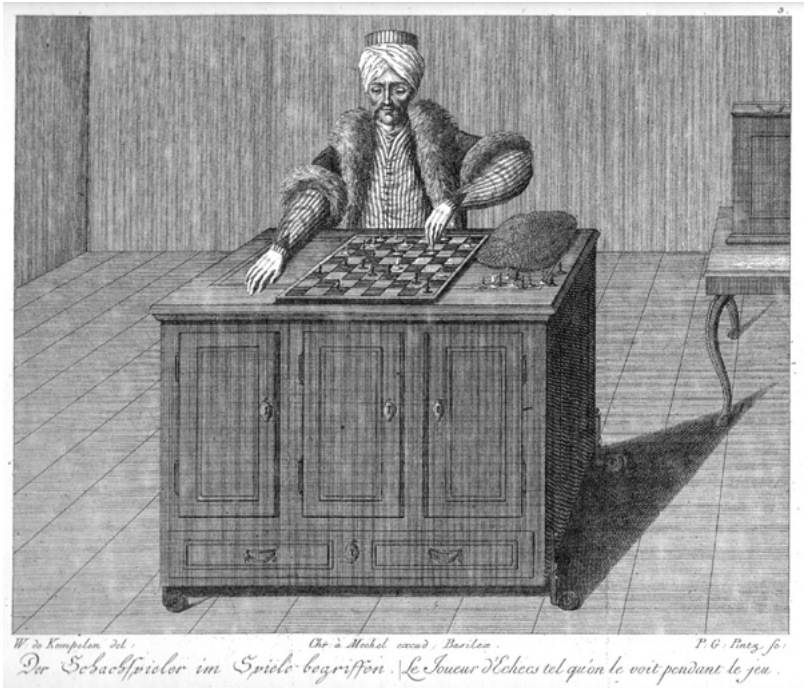
Automaton (Greek, autos "self" + matos "thinking, animated, willing")

http://www.etymonline.com

- from English translation of 1920 play "Rossum's Universal Robots" by Karel Capek
- from Czech robotnik (slave), robota (forced labor, drudgery), robotiti (to work, drudge)
- from Slavic (arabeit) related to German Arbeit (work)
- Word coined by Capek's brother Josef, who used it initially in a short story
- Robotics coined in 1941 in a science fiction context by Isaac Asimov

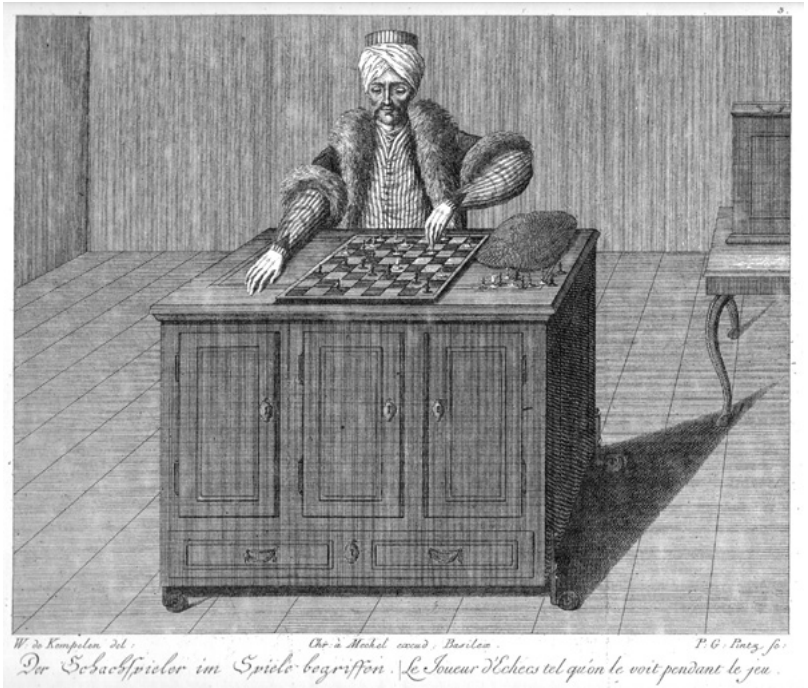
- Robotics Institute of America: re-programmable multi-functional manipulator designed to move materials, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks
- "device that automatically performs complicated often repetitive tasks," or a "mechanism guided by automatic controls"

First Robot – The Turk / Automaton Chess Player (1770)



http://en.wikipedia.org/wiki/The_Turk

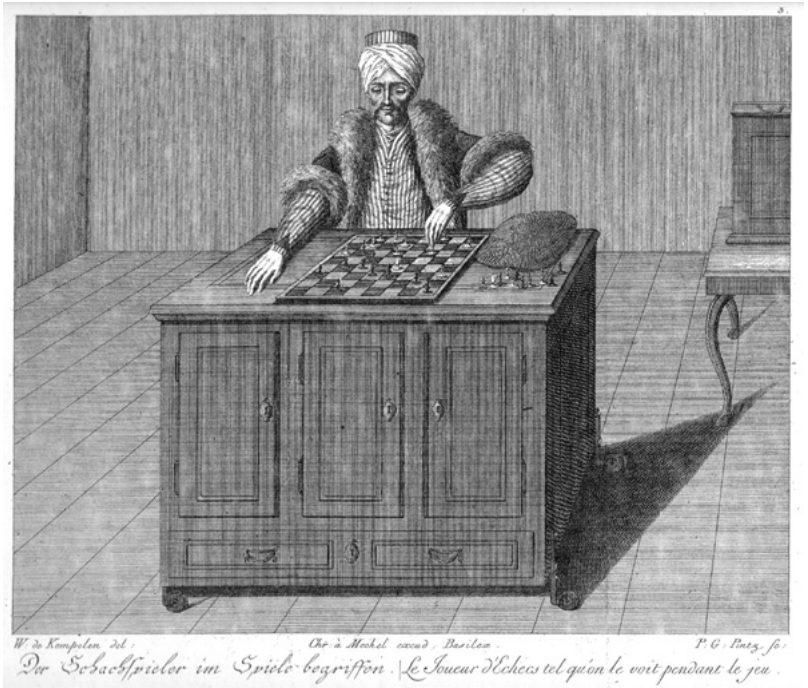
First Robot – The Turk / Automaton Chess Player (1770)



http://en.wikipedia.org/wiki/The_Turk

- Constructed by Wolfgang von Kempelen in 1770
- Played many exhibition chess games
- Solved the knight-tour problem
- Even played against Benjamin Franklin in France

First Fake Robot – The Turk / Automaton Chess Player (1770)

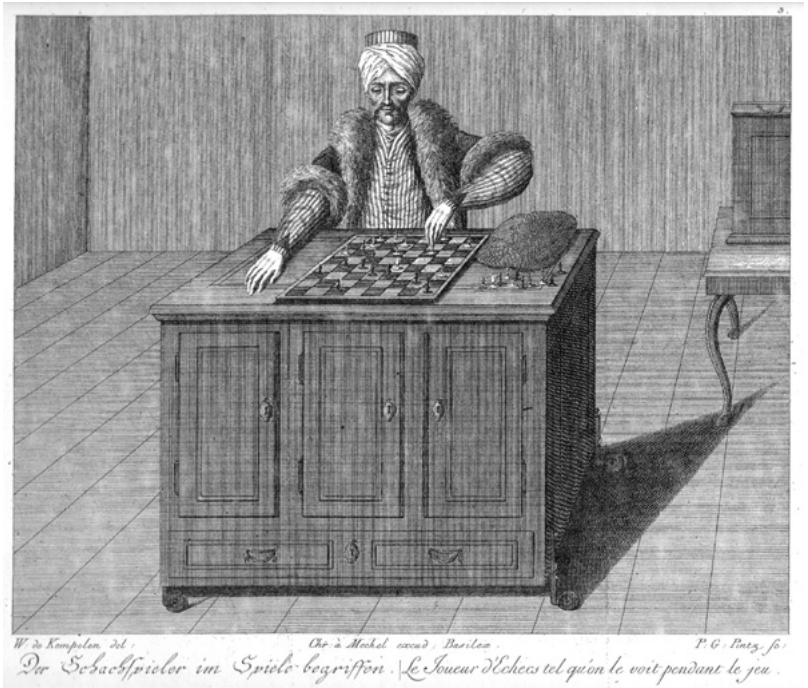


http://en.wikipedia.org/wiki/The_Turk

. . . it was a fake, however, human player hid inside machine

- Constructed by Wolfgang von Kempelen in 1770
- Played many exhibition chess games
- Solved the knight-tour problem
- Even played against Benjamin Franklin in France

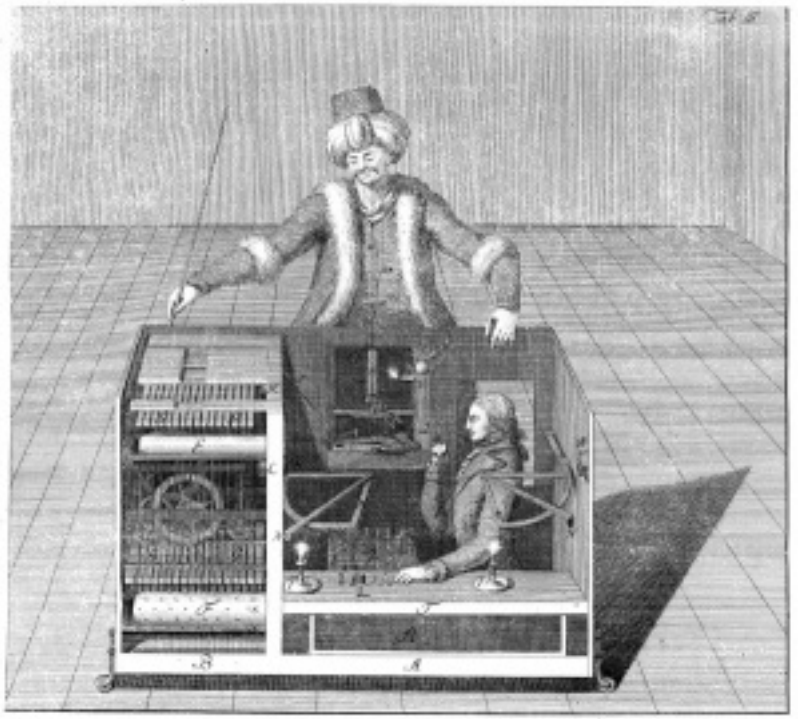
First Fake Robot – The Turk / Automaton Chess Player (1770)



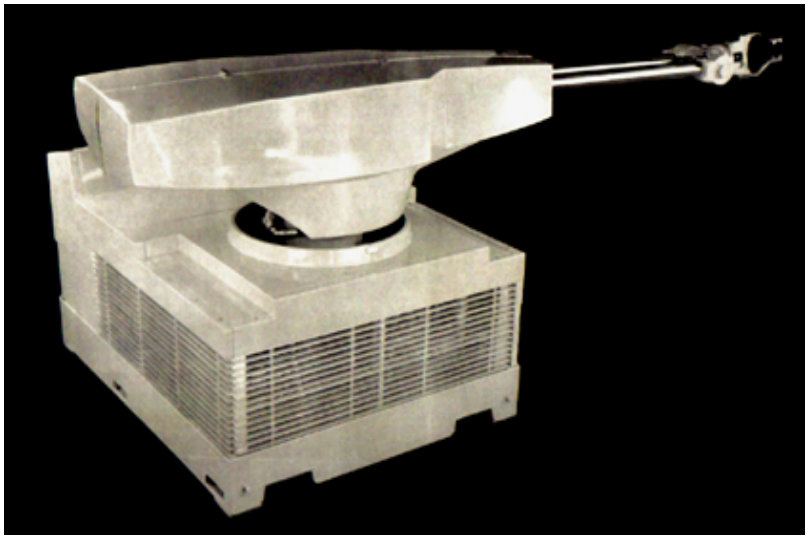
http://en.wikipedia.org/wiki/The_Turk

. . . it was a fake, however, human player hid inside machine

- Constructed by Wolfgang von Kempelen in 1770
- Played many exhibition chess games
- Solved the knight-tour problem
- Even played against Benjamin Franklin in France



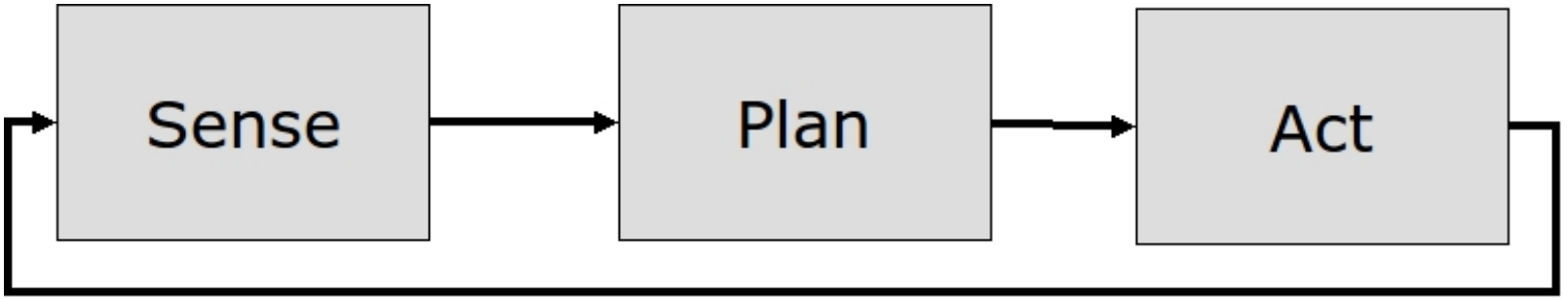
First Real Robot – Unimate (1961)



http://en.wikipedia.org/wiki/Unimate

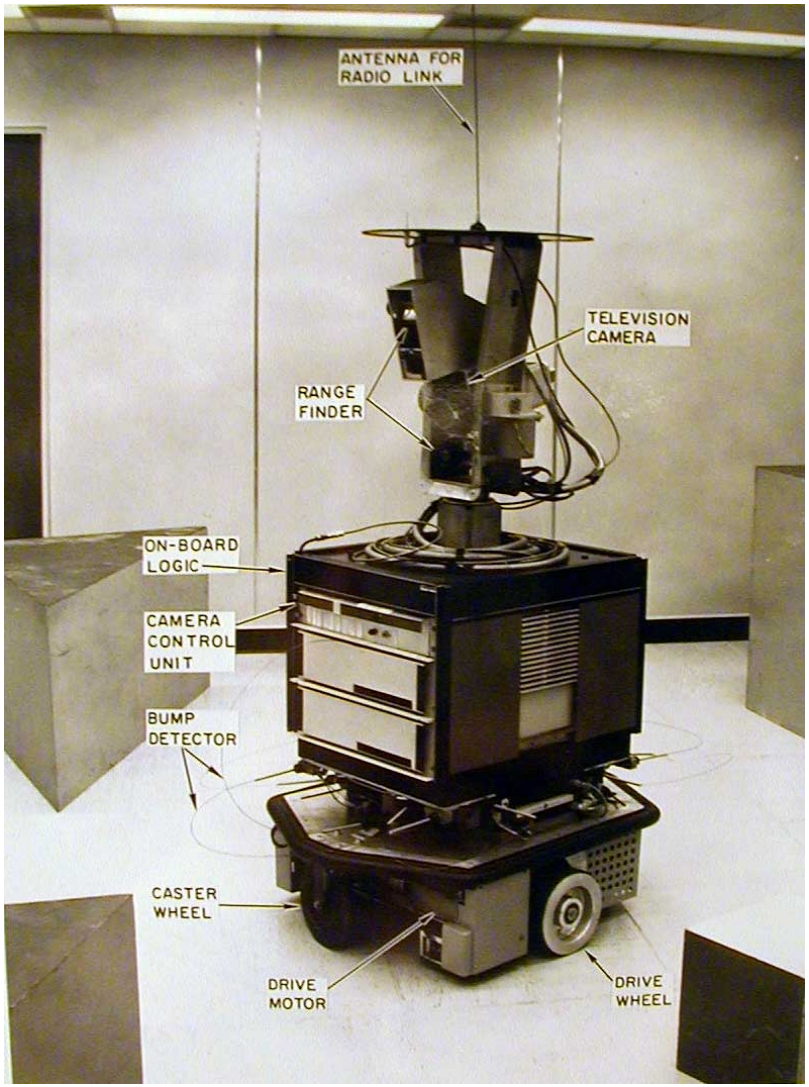
- Created by George Devol
- Worked on a General Motors assembly line in New Jersey in 1961
- Job consisted of transporting die castings from an assembly line and welding these parts on auto bodies
- Conducted in Robot Hall of Fame in 2003

Classical Paradigm



- Focus on automated reasoning and knowledge representation
- Perfect world model
- Closed world assumption: "what is not currently known to be true, is false"
- STRIPS (Stanford Research Institute Problem Solver)

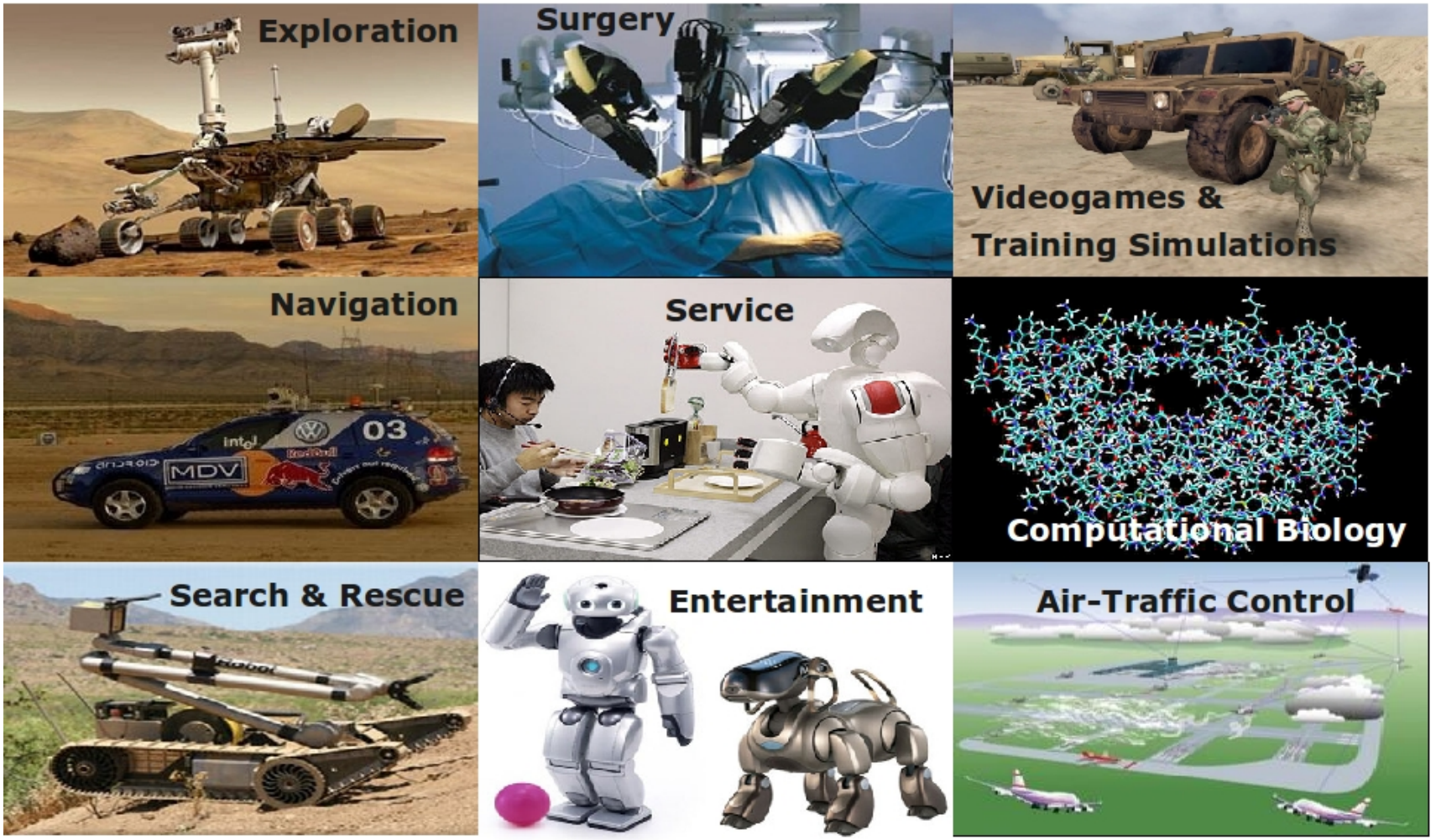
Shakey (Stanford Research Institute, 1966)



http://en.wikipedia.org/wiki/Shakey_the_robot

- First mobile robot to reason about its own actions
- Programs for "seeing," "reasoning," and "acting"
- Triangulating range-finder for sensing obstacles
- Wireless radio and video camera
- Used STRIPS to perform "block-worlds" tasks
- Conducted in Robot Hall of Fame in 2004

Robots Today



[stanley] [bdog] [ldog] [rhex] [heli] [snake] [hand] [asimo]

CS 336/436 – Algorithms for Sensor-based Robotics

Lecture IV – Configuration Space

Erion Plaku

Department of Computer Science
Laboratory for Computational Sensing and Robotics
Johns Hopkins University

February 4, 2010

Path Planning: From Point Robots to Robots with Geometric Shapes

- We have seen path-planning algorithms when a robot is a point
- How can we plan a collision-free path when the robot has a geometric shape?

. . . a key concept in path planning is the notion of a *configuration space*

Configuration (denoted by q)

- a complete specification of the position of every point of the robot

Configuration Space or C-Space (denoted by Q)

- space of all possible configurations of the robot, i.e., $Q = \{q : q \textit{ is a configuration}\}$

Collision-Free Configuration

- q is collision free iff the robot does not collide with any obstacles when in configuration q, i.e., $\texttt{Robot}(q) \cap \left(\bigcup_{i=1} \texttt{Obstacle}_i\right) = \emptyset$

Collision-Free Configuration Space

- $Q_{free} = \{q \in Q : q \textit{ is collision free}\}$

Path-Planning Problem: Compute collision-free path from q_{init} to q_{goal}

- $\texttt{path} : [0, 1] \rightarrow Q_{free}$ is a continuous function with $\texttt{path}(0) = q_{init}$, $\texttt{path}(1) = q_{goal}$

Examples of Configuration Spaces

disk robot with radius r that can translate without rotating in the plane:

- How can the configuration be represented?

  as the two-dimensional position (c_x, c_y) of the robot's center

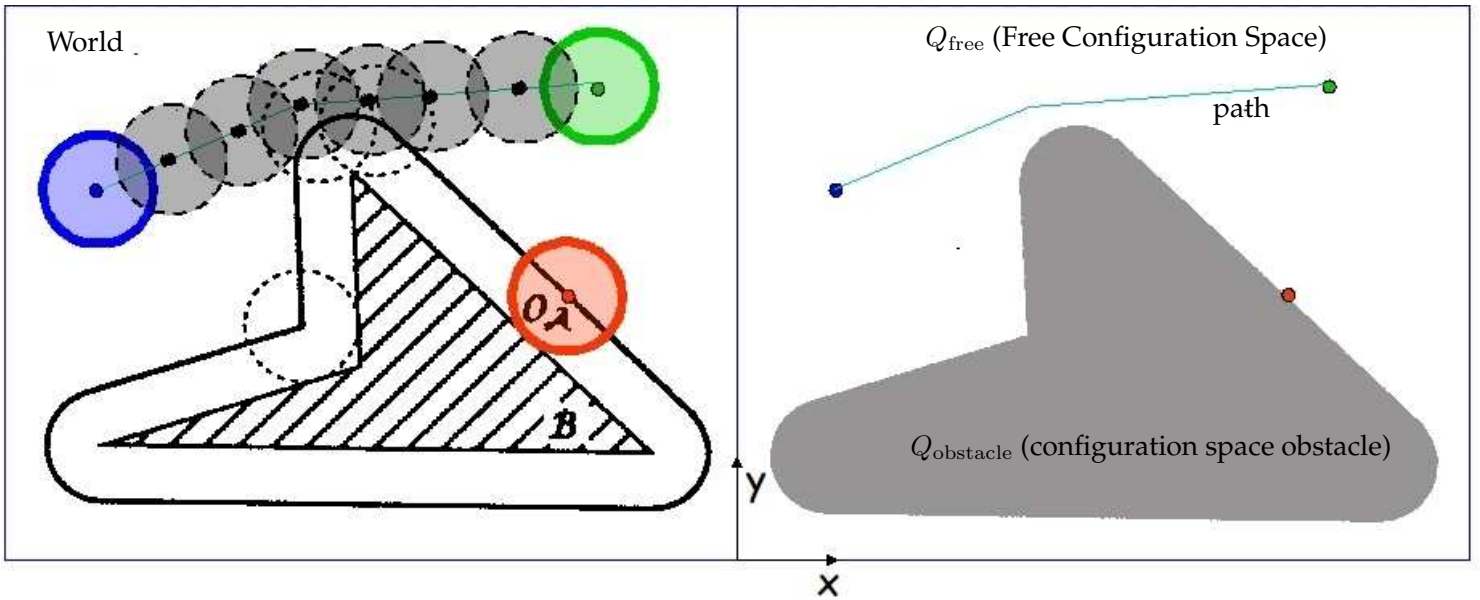
- How can the points on the robot be expressed as a function of its configuration?

$$\texttt{Robot}(c_x, c_y) = \{(x, y) : (x - c_x)^2 + (y - c_y) \leq r^2\}$$

- What is the configuration space Q?

$$Q = \mathbb{R}^2 \text{ (same as that of a point robot)}$$

- What is the free configuration space Q_{free}? Is it the same as that of a point robot?



[Fig. courtesy of Latombe]

- How would you compute Q_{free}?

Examples of Configuration Spaces

polygon P that can translate and rotate in the plane:

- How can the configuration be represented?

  (c_x, c_y, θ): position + orientation

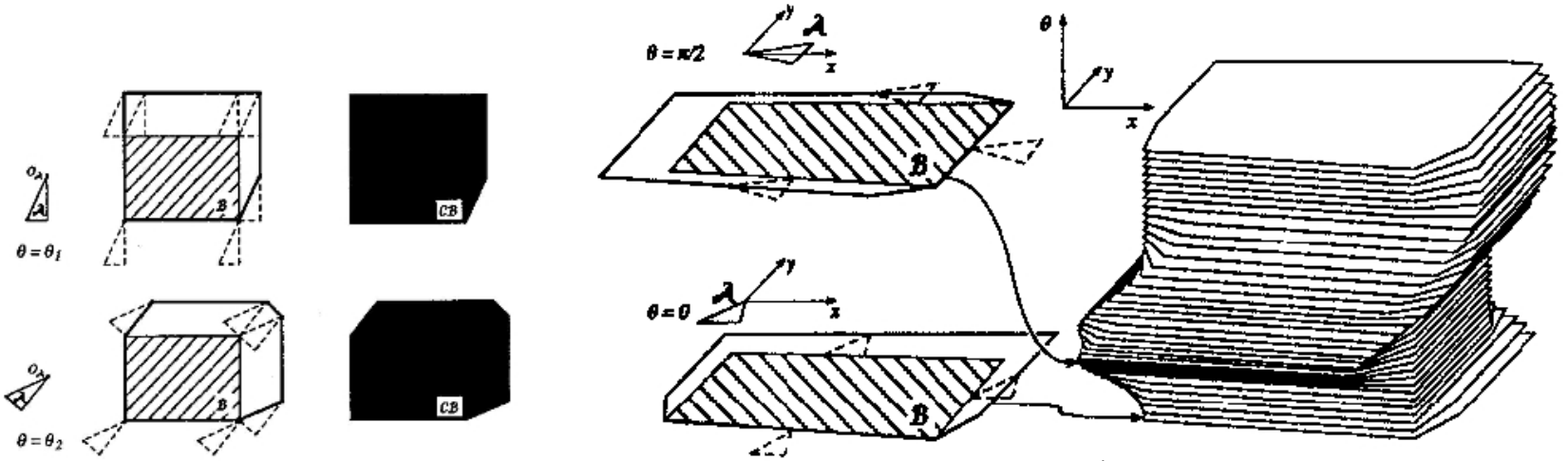
- How can the points on the robot be expressed as a function of its configuration?

$$\texttt{Robot}(c_x, c_y, \theta) = \left\{ \begin{pmatrix} \cos\theta & -\sin\theta & c_x \\ \sin\theta & \cos\theta & c_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} : (x, y) \in P \right\}$$

- What is the configuration space Q?

  $Q = \mathbb{R}^2 \times S^1$ (S^1 refers to the unit circle)

- What is the free configuration space Q_{free}?



[Fig. courtesy of Latombe]

- How would you compute Q_{free}?

Examples of Configuration Spaces

manipulator with revolute joints:

- How can the configuration be represented?

  $(\theta_1, \theta_2, \ldots, \theta_n)$: vector of joint values

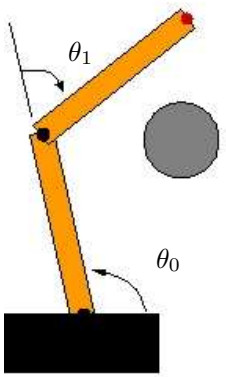
- How can the points on the robot be expressed as a function of its configuration?

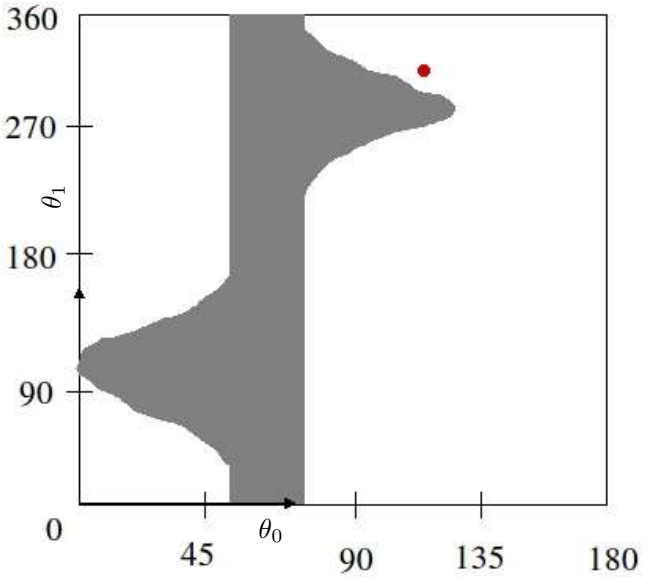
  forward kinematics (more later in the course)

- What is the configuration space Q?

  $Q = \overbrace{S^1 \times S^1 \ldots \times S^1}^{n}$ (S^1 refers to the unit circle)

- What is the free configuration space Q_{free}?





[Fig. courtesy of Hager]

- How would you compute Q_{free}?

Minkowski Sums

- The Minkowski sum of two sets A and B, denoted by $A \oplus B$, is defined as

$$A \oplus B = \{a + b : a \in A, b \in B\}$$

- The Minkowski difference of two sets A and B, denoted by $A \ominus B$, is defined as

$$A \ominus B = \{a - b : a \in A, b \in B\}$$

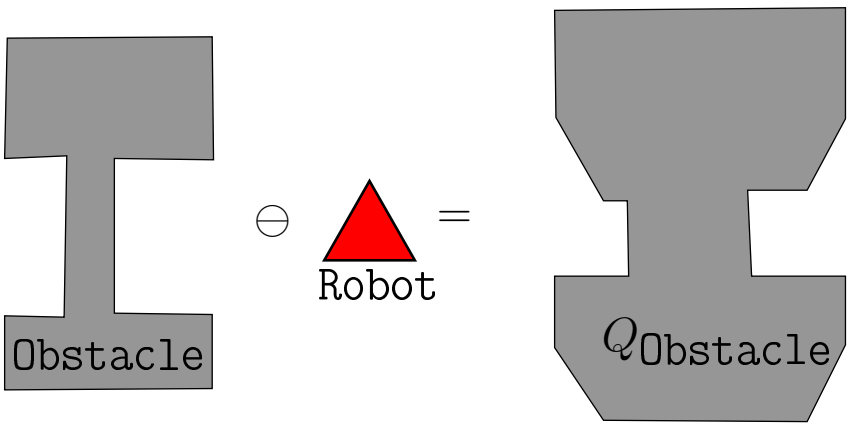
How does it relate to path planning?

- Recall the definition of the configuration-space obstacle

$$Q_{\texttt{Obstacle}} = \{q : q \in Q \textit{ and } \texttt{Robot}(q) \cap \texttt{Obstacle} \neq \emptyset\}$$

  (set of all robot configurations that collide with the obstacle)
- Classical result shown by Lozano-Perez and Wesley 1979

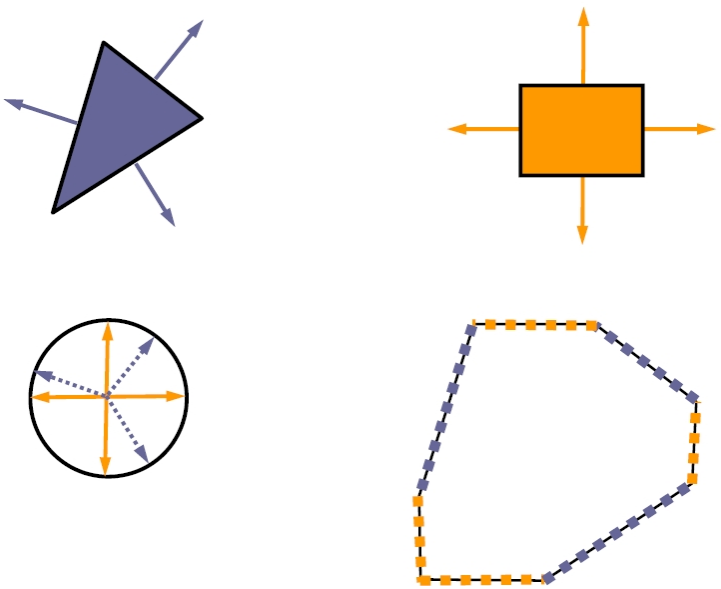
for polygons and polyhedra : $Q_{\texttt{Obstacle}} = \texttt{Obstacle} \ominus \texttt{Robot}$



[Fig. courtesy of Manocha]

Properties of Minkowski Sums

- Minkowski sum of two convex sets is convex
- Minkowski sum of two convex polygons A and B with m and n vertices ...
  - ... is a convex polygon with $m + n$ vertices
  - ... vertices of $A \oplus B$ are "sums" of vertices of A and B
  - ... $A \oplus B$ can be computed in linear time and space $O(n + m)$



[Fig. courtesy of Manocha]

Algorithm

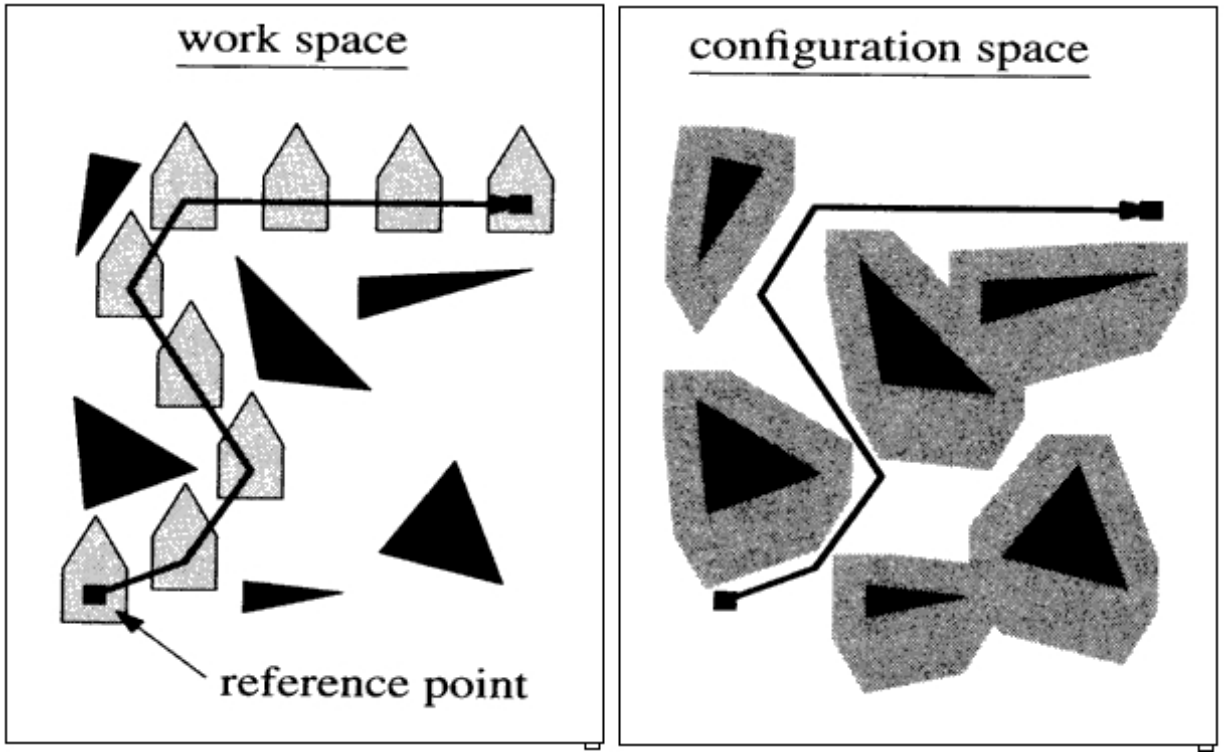
- sort edges according to angle between x-axis and edge normal
- let the sorted edges be $e_1, e_2, \ldots, e_{n+m}$
- attach edges one after the other so that edge e_{i+1} starts where edge e_i ends

- Minkowski sum for nonconvex polygons
  - Decompose into convex polygons (e.g., triangles, trapezoids)
  - Compute the minkowski sums of the convex polygons and take their union
  - Complexity: $O(n^2 m^2)$ (4-th order polynomial)
- 3D Minkowski sums: [convex: $O(nm)$ complexity] [nonconvex: $O(n^3 m^3)$ complexity]

Path Planning: From Point Robots to Robots with Geometric Shapes

- We have seen path-planning algorithms when a robot is a point
- How can we plan a collision-free path when the robot has a geometric shape?

. . . a key concept in path planning is the notion of a *configuration space*



- reduce robot to a point in the configuration space
- compute configuration-space obstacles (difficult to do in general)
- search for a path for the point robot in the free configuration space

Topology of Configuration Spaces

Why study it?

- Extend results from one configuration space to another
- Design specialized algorithms that take advantage of certain topologies

What about the topology?

- Topology is the "intrinsic character" of a space
- Two spaces have different topologies if cutting and pasting is required to make them the same (think of rubber figures — if we can stretch and reshape "continuously" without tearing, one into the other, they have the same topology)
- Mathematical mechanisms for talking about topology: homeomorphism/diffeomorphism

  $f : X \to Y$ is called a homeomorphism iff
  - f is a bijection (one-to-one and onto)
  - f is continuous
  - f^{-1} (the inverse of f) is continuous

  examples of homeomorphisms: [disc to square]; [$(-1, 1)$ to $\mathbb{R}$]

  X is diffeomorphic to Y iff exists $f : X \to Y$ such that
  - f is a homeomorphism where f and f^{-1} are smooth (derivatives of all orders exist)

  example of diffeomorphism: circle to ellipse
- An n-dimensional configuration space Q is a *manifold* if it locally looks like $\mathbb{R}^n$, i.e., every $q \in Q$ has a neighborhood homeomorphic to $\mathbb{R}^n$
- A manifold is path-connected if there is a path between any two points

CS 336/436 – Algorithms for Sensor-based Robotics

Sampling-based Motion Planning - Parts I-IV

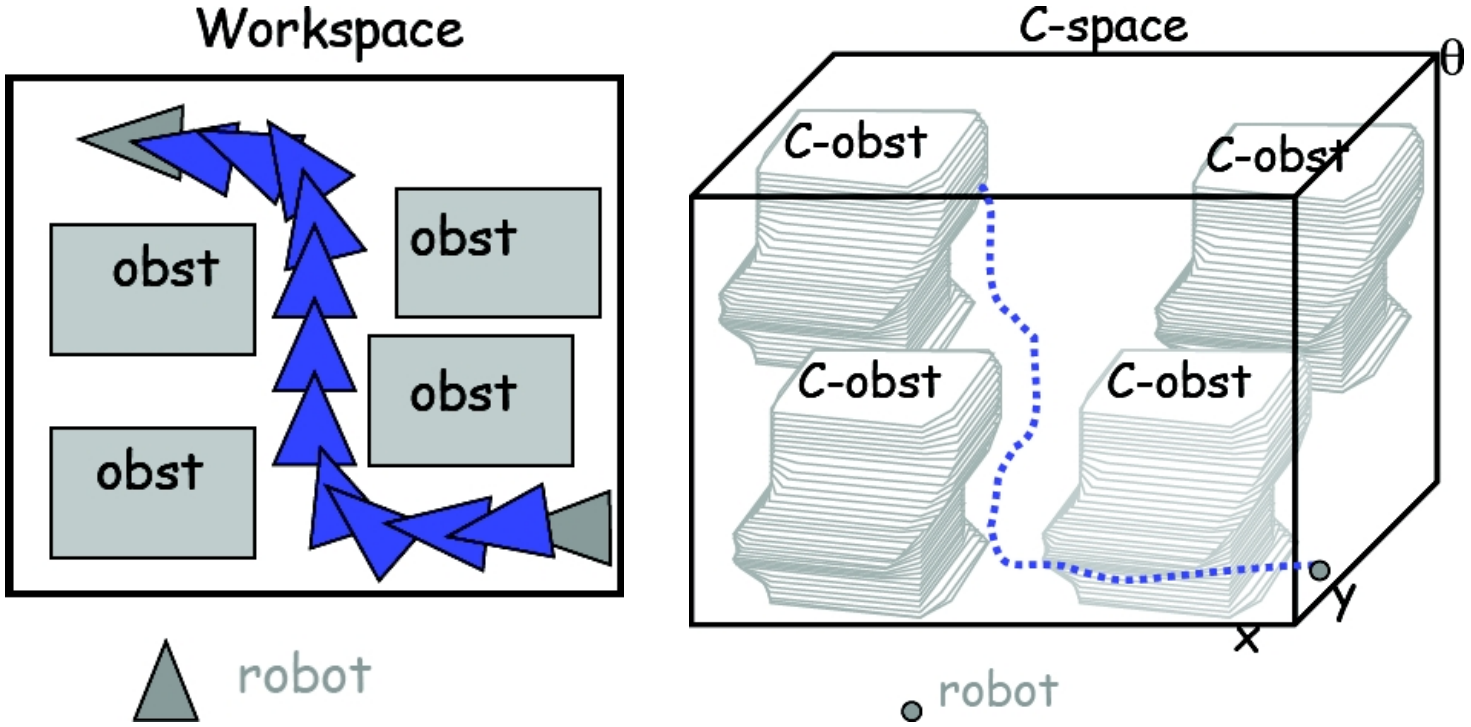
Erion Plaku

Department of Computer Science
Laboratory for Computational Sensing and Robotics
Johns Hopkins University

February 25, 2010 March 4, 2010 March 9, 2010 March 11, 2010

From Workspace to Configuration Space

- simple workspace obstacle transformed into complex configuration-space obstacle
- robot transformed into point in configuration space
- path transformed from swept volume to 1d curve



[fig from Jyh-Ming Lien]

Explicit Construction of Configuration Space/Roadmaps

- PSPACE-complete
- Exponential dependency on dimension
- No practical algorithms

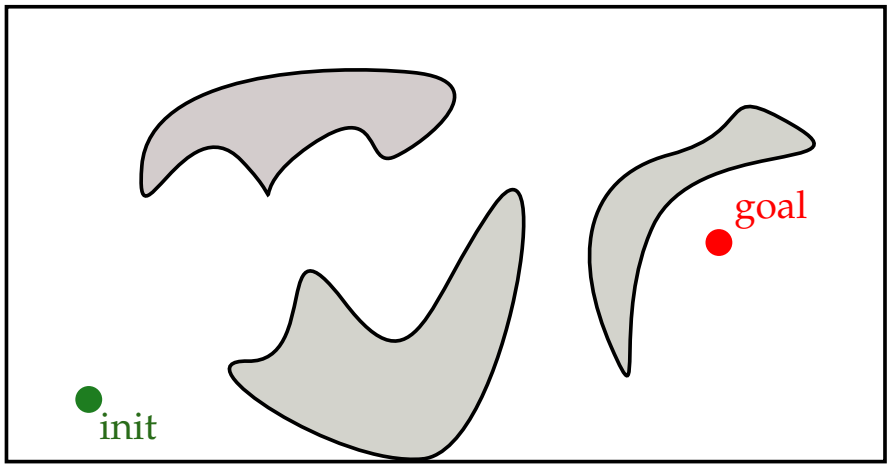
Motion Planning for a Point Robot in 2D

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

Motion Planning for a Point Robot in 2D

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

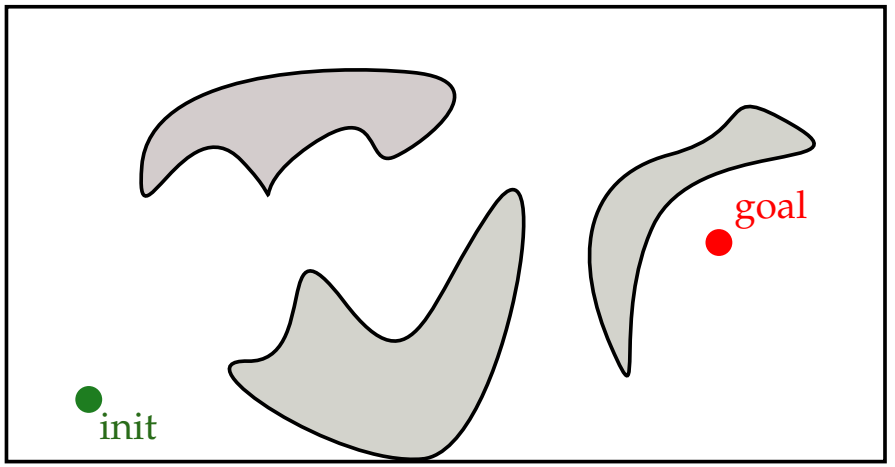
How would you solve it?



Motion Planning for a Point Robot in 2D

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

How would you solve it?

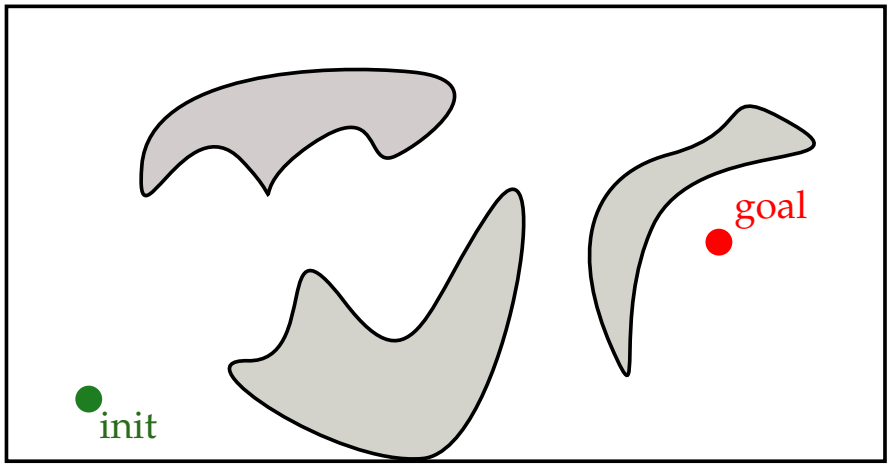


Hint: How would you approximate π?

Motion Planning for a Point Robot in 2D

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

How would you solve it?

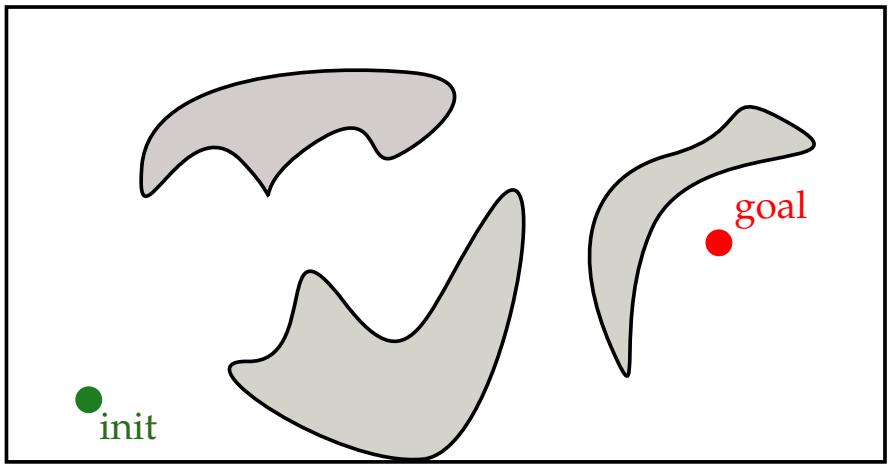


Hint: How would you approximate π?

Motion Planning for a Point Robot in 2D

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

How would you solve it?

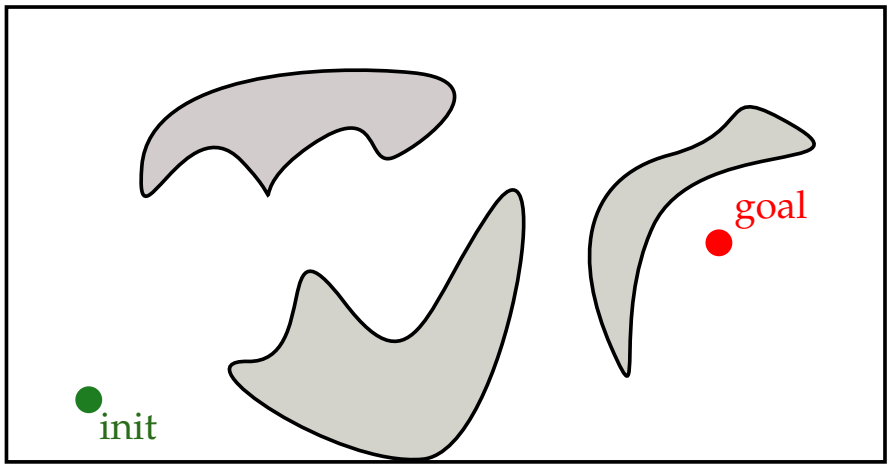


Hint: How would you approximate π?

Motion Planning for a Point Robot in 2D

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

How would you solve it?

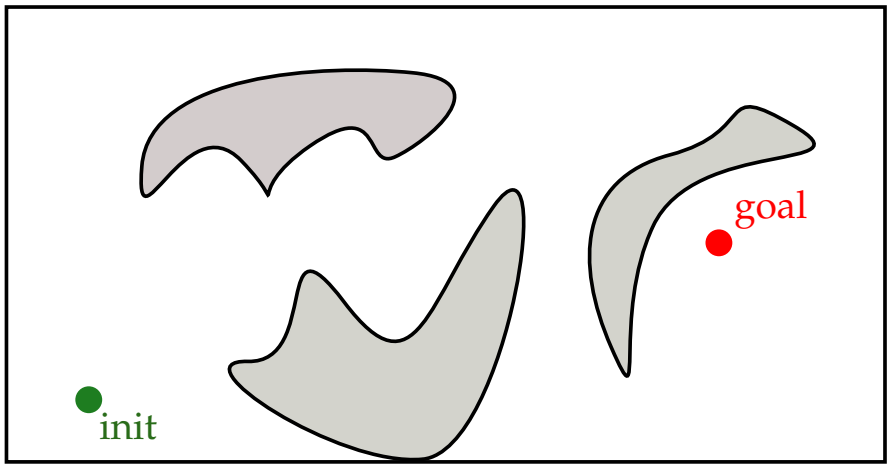


Hint: How would you approximate π?

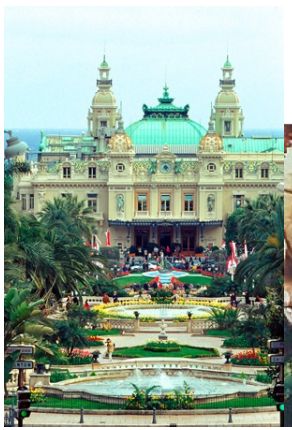
Motion Planning for a Point Robot in 2D

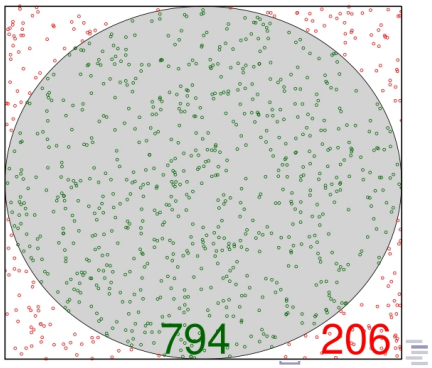
- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

How would you solve it?



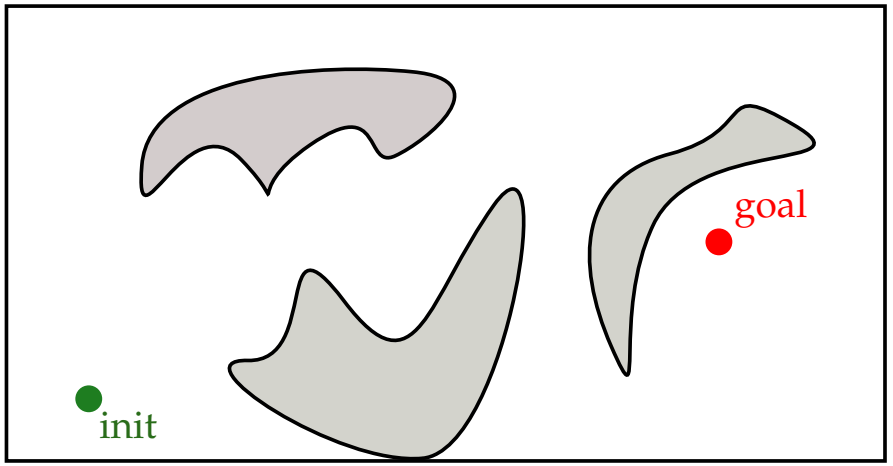
Hint: How would you approximate π?



Motion Planning for a Point Robot in 2D

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

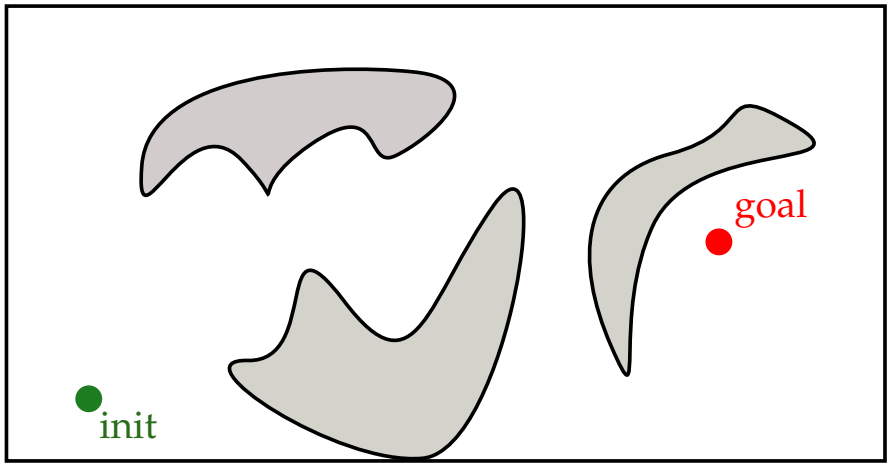


Monte-Carlo Idea:

- Define input space
- Generate inputs at random by *sampling* the input space
- Perform a deterministic computation using the input samples
- Aggregate the partial results into final result

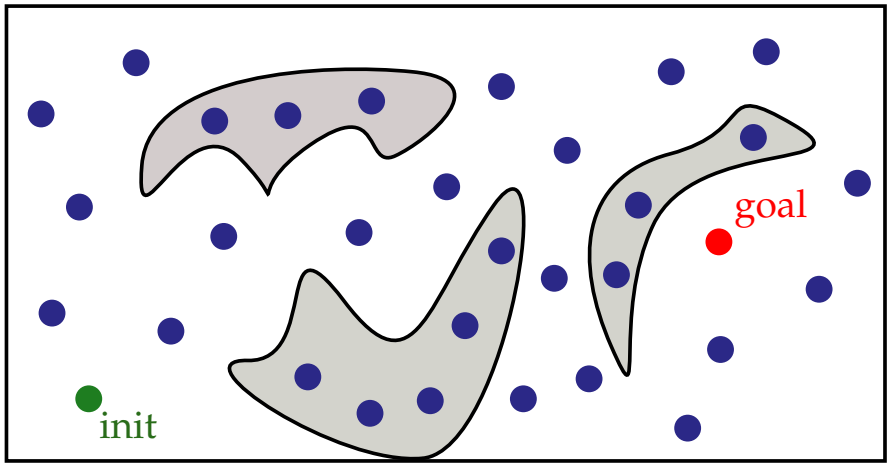
Sampling-based Motion Planning

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



Sampling-based Motion Planning

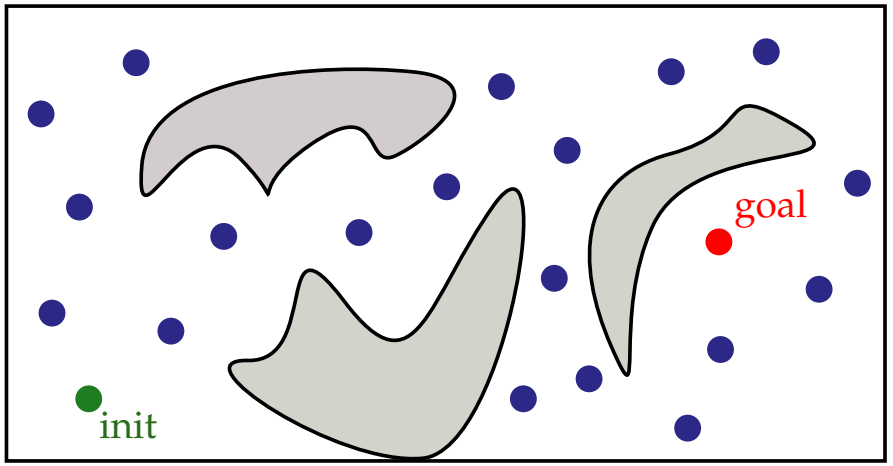
- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



- Sample points

Sampling-based Motion Planning

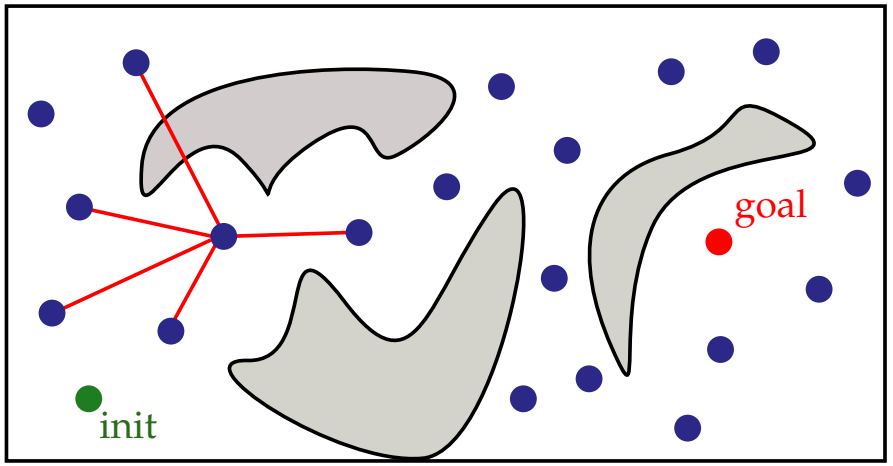
- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



- Sample points
- Discard samples that are in collision

Sampling-based Motion Planning

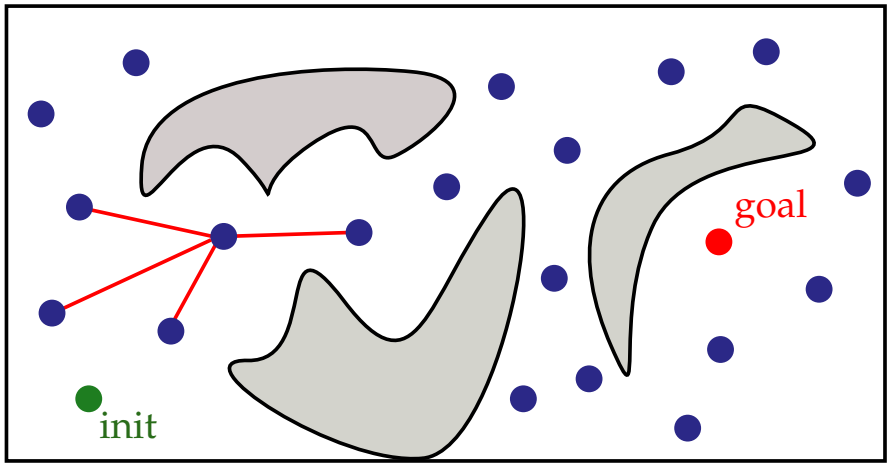
- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



- Sample points
- Discard samples that are in collision
- Connect neighboring samples via straight-line segments

Sampling-based Motion Planning

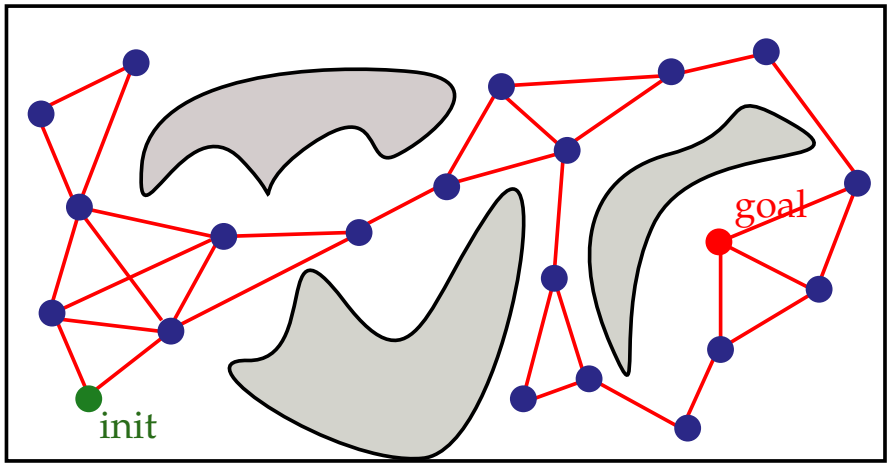
- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



- Sample points
- Discard samples that are in collision
- Connect neighboring samples via straight-line segments
- Discard straight-line segments that are in collision

Sampling-based Motion Planning

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

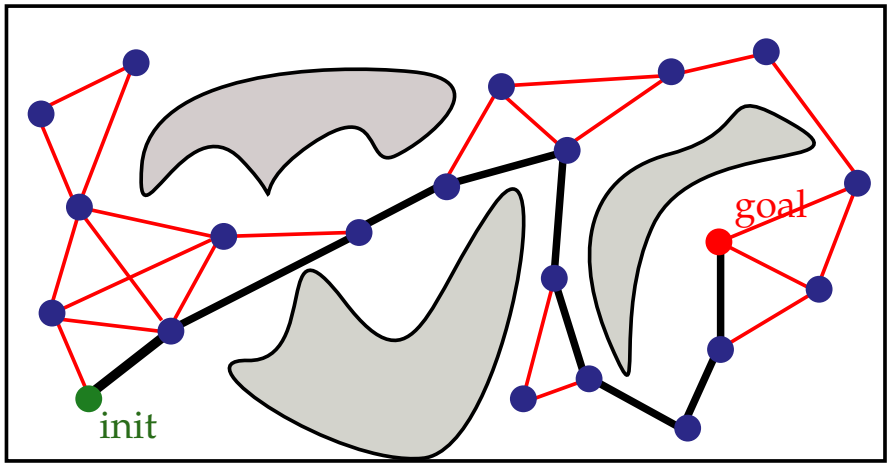


- Sample points
- Discard samples that are in collision
- Connect neighboring samples via straight-line segments
- Discard straight-line segments that are in collision

$\Rightarrow$ Gives rise to a graph, called the *roadmap*

Sampling-based Motion Planning

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



- Sample points
- Discard samples that are in collision
- Connect neighboring samples via straight-line segments
- Discard straight-line segments that are in collision

$\Rightarrow$ Gives rise to a graph, called the *roadmap*

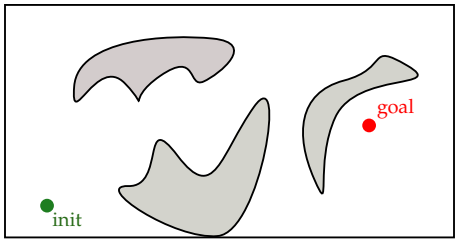
$\Rightarrow$ Collision-free path can be found by performing graph search on the roadmap

Probabilistic RoadMap (PRM) Method

[Kavraki, Švestka, Latombe, Overmars 1996]

0. Initialization

add q_{init} and q_{goal} to roadmap vertex set V



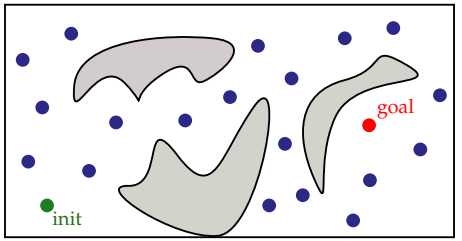
1. Sampling

repeat several times

$q \leftarrow \textsc{Sample}()$

if $\textsc{IsCollisionFree}(q) = \texttt{true}$

add q to roadmap vertex set V



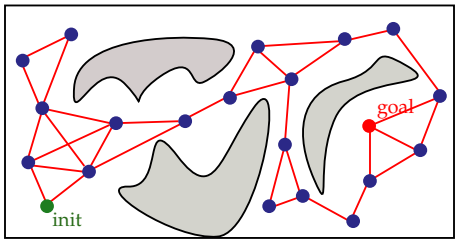
2. Connect Samples

for each pair of neighboring samples $(q_a, q_b) \in V \times V$

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

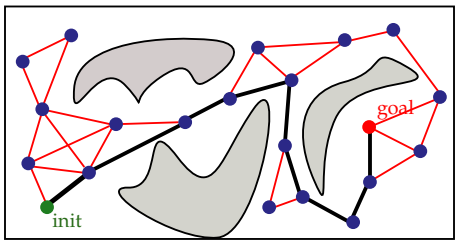
if $\textsc{IsCollisionFree}(\text{path}) = \texttt{true}$

add (q_a, q_b) to roadmap edge set E



3. Graph Search

search graph (V, E) for path from q_{init} to q_{goal}



Sampling-based Path Planning

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Sampling-based Path Planning

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Sampling-based Path Planning

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Is it then just a heuristic approach?

Sampling-based Path Planning

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Is it then just a heuristic approach? No. It's more than that

Sampling-based Path Planning

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Is it then just a heuristic approach? No. It's more than that

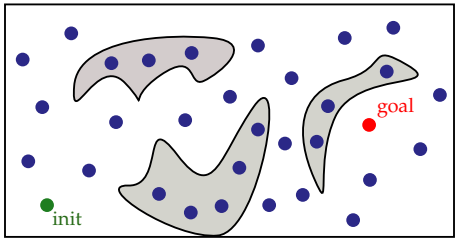
It offers *probabilistic completeness*

- When a solution exists, a probabilistically complete planner finds a solution with probability as time goes to infinity.
- When a solution does not exists, a probabilistically complete planner may not be able to determine that a solution does not exist.

PRM Applied to 2D-point Robot

$q = (x, y) \leftarrow \text{SAMPLE}()$

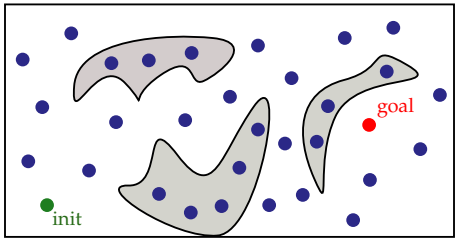
- $x \leftarrow \text{RAND}(\min_x, \max_x)$
- $y \leftarrow \text{RAND}(\min_y, \max_y)$



PRM Applied to 2D-point Robot

$q = (x, y) \leftarrow$ SAMPLE()
- $x \leftarrow$ RAND($\min_x, \max_x$)
- $y \leftarrow$ RAND($\min_y, \max_y$)

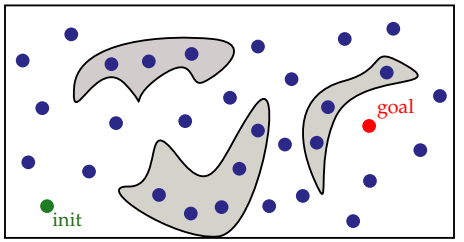
ISSAMPLECOLLISIONFREE(q)
- Point inside/outside polygon test



PRM Applied to 2D-point Robot

$q = (x, y) \leftarrow$ SAMPLE()

- $x \leftarrow$ RAND($\min_x, \max_x$)
- $y \leftarrow$ RAND($\min_y, \max_y$)



ISSAMPLECOLLISIONFREE(q)

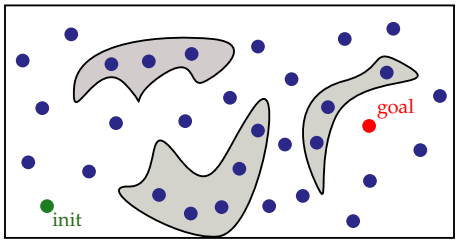
- Point inside/outside polygon test

path $\leftarrow$ GENERATELOCALPATH(q_a, q_b)

- Straight-line segment from point q_a to point q_b

PRM Applied to 2D-point Robot

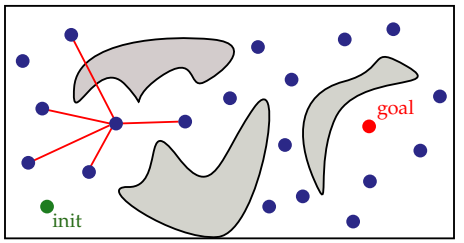
$q = (x, y) \leftarrow \text{SAMPLE}()$
- $x \leftarrow \text{RAND}(\min_x, \max_x)$
- $y \leftarrow \text{RAND}(\min_y, \max_y)$



$\text{ISSAMPLECOLLISIONFREE}(q)$
- Point inside/outside polygon test

$\text{path} \leftarrow \text{GENERATELOCALPATH}(q_a, q_b)$
- Straight-line segment from point q_a to point q_b

$\text{ISPATHCOLLISIONFREE}(\text{path})$
- Segment-polygon intersection test

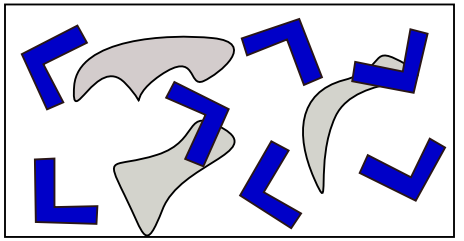


PRM Applied to 2D Rigid-Body Robot

PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

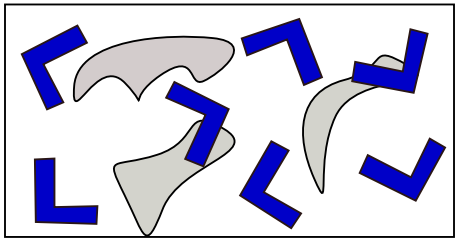
- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$



PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

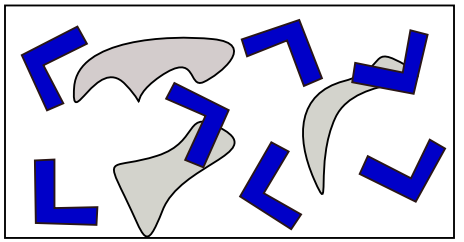


$\textsc{IsSampleCollisionFree}(q)$

PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

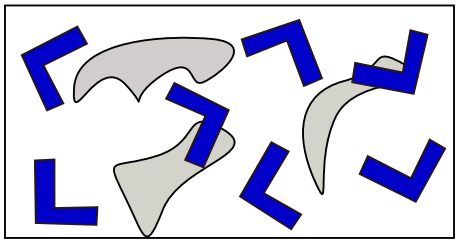


$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \textsc{Sample}()$
- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

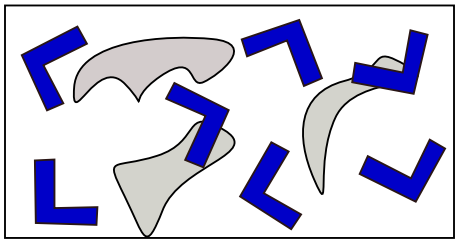


$\textsc{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \textsc{Sample}()$
- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

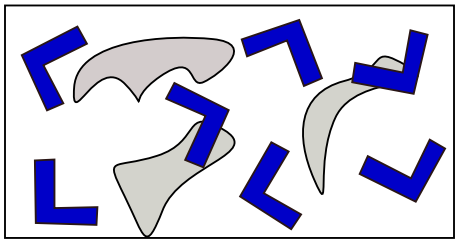


$\textsc{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$

PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \textsc{Sample}()$
- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

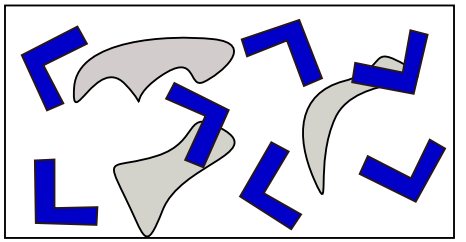


$\textsc{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$

PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \textsc{Sample}()$
- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

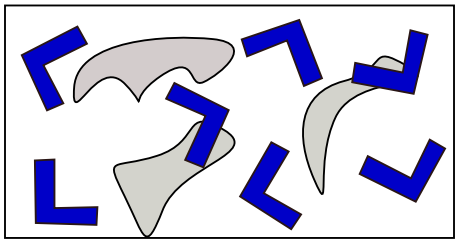


$\textsc{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

PRM Applied to 2D Rigid-Body Robot



$q = (x, y, \theta) \leftarrow \textsc{Sample}()$
- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

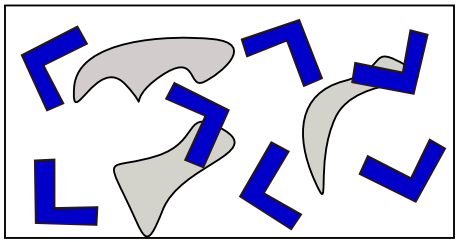
$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\textsc{IsPathCollisionFree}(\text{path})$

PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \text{SAMPLE}()$

- $x \leftarrow \text{RAND}(\min_x, \max_x)$; $y \leftarrow \text{RAND}(\min_y, \max_y)$; $\theta \leftarrow \text{RAND}(-\pi, \pi)$



$\text{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

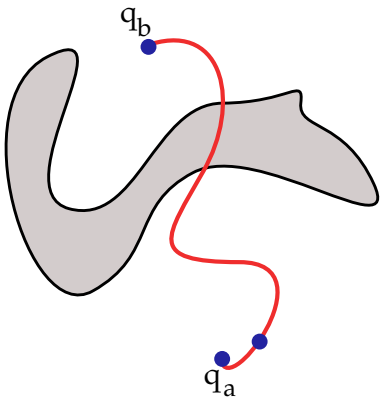
$\text{path} \leftarrow \text{GenerateLocalPath}(q_a, q_b)$

- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

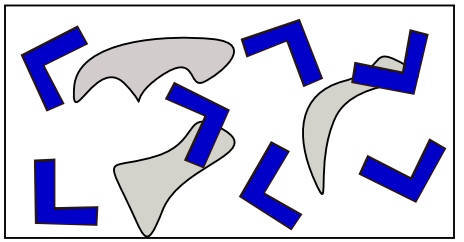
$\text{IsPathCollisionFree}(\text{path})$

- Incremental approach



PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \text{SAMPLE}()$
- $x \leftarrow \text{RAND}(\min_x, \max_x)$; $y \leftarrow \text{RAND}(\min_y, \max_y)$; $\theta \leftarrow \text{RAND}(-\pi, \pi)$

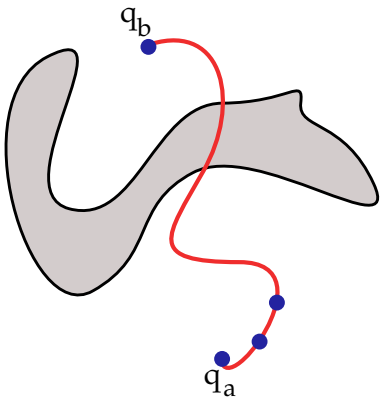


$\text{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \text{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

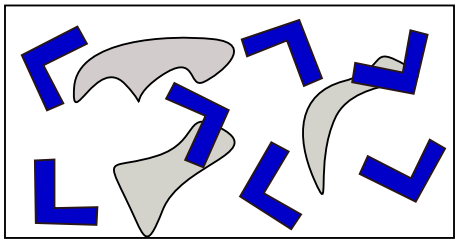
$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\text{IsPathCollisionFree}(\text{path})$
- Incremental approach



PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \text{SAMPLE}()$
- $x \leftarrow \text{RAND}(\min_x, \max_x)$; $y \leftarrow \text{RAND}(\min_y, \max_y)$; $\theta \leftarrow \text{RAND}(-\pi, \pi)$

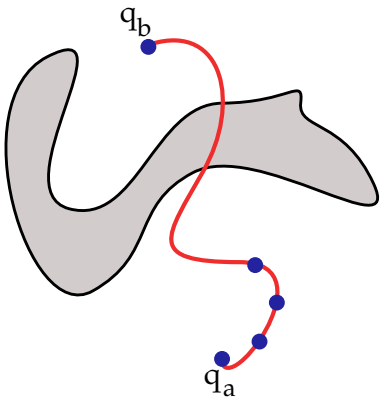


$\text{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \text{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

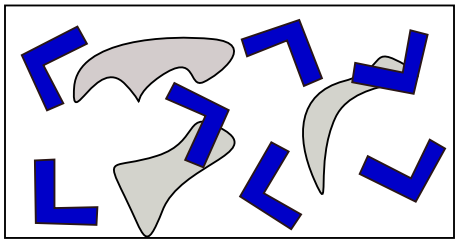
$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\text{IsPathCollisionFree}(\text{path})$
- Incremental approach



PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \text{SAMPLE}()$
- $x \leftarrow \text{RAND}(\min_x, \max_x)$; $y \leftarrow \text{RAND}(\min_y, \max_y)$; $\theta \leftarrow \text{RAND}(-\pi, \pi)$

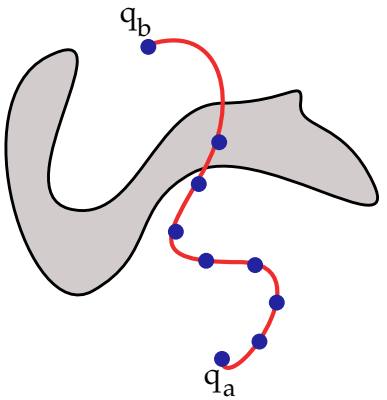


$\text{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \text{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

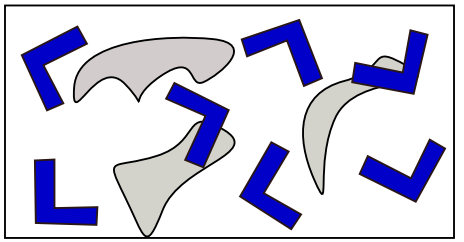
$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\text{IsPathCollisionFree}(\text{path})$
- Incremental approach



PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \text{SAMPLE}()$
- $x \leftarrow \text{RAND}(\min_x, \max_x)$; $y \leftarrow \text{RAND}(\min_y, \max_y)$; $\theta \leftarrow \text{RAND}(-\pi, \pi)$

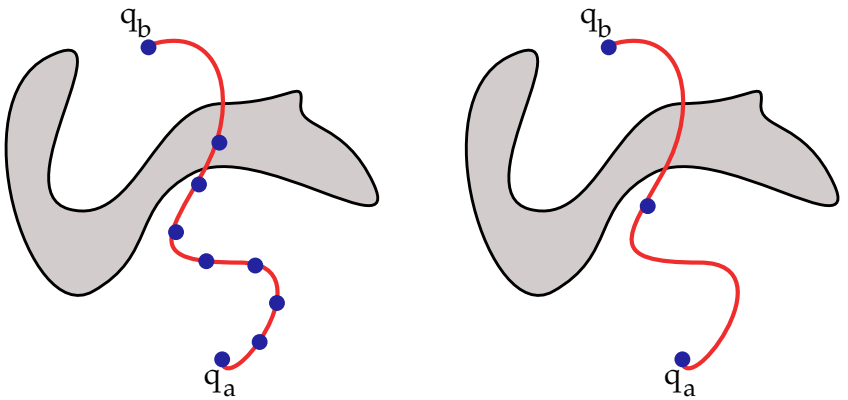


IsSampleCollisionFree(q)
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

path $\leftarrow$ GenerateLocalPath(q_a, q_b)
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

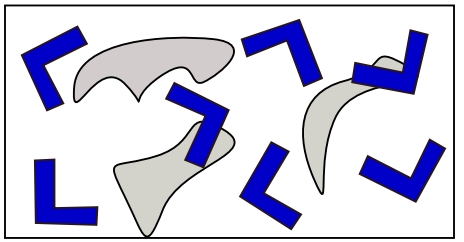
$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

IsPathCollisionFree(path)
- Incremental approach
- Subdivision approach



PRM Applied to 2D Rigid-Body Robot

$q = (x, y, \theta) \leftarrow \text{SAMPLE}()$
- $x \leftarrow \text{RAND}(\min_x, \max_x)$; $y \leftarrow \text{RAND}(\min_y, \max_y)$; $\theta \leftarrow \text{RAND}(-\pi, \pi)$

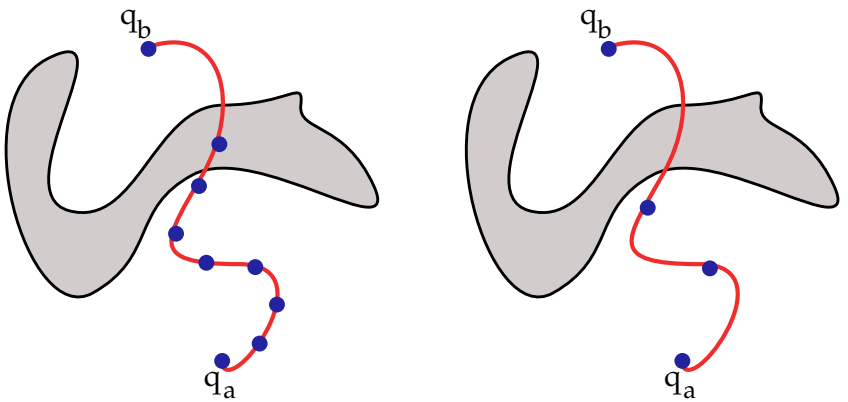


$\text{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \text{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

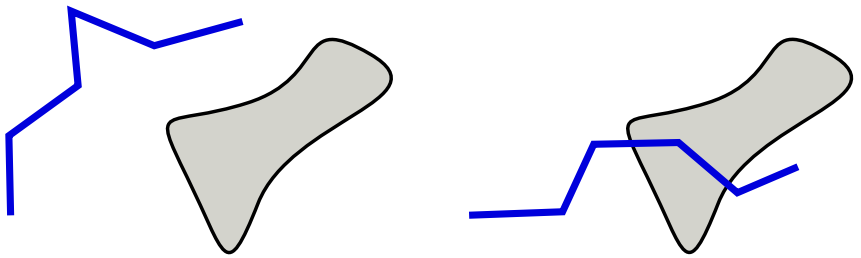
$\text{IsPathCollisionFree}(\text{path})$
- Incremental approach
- Subdivision approach



PRM Applied to Articulated Chain

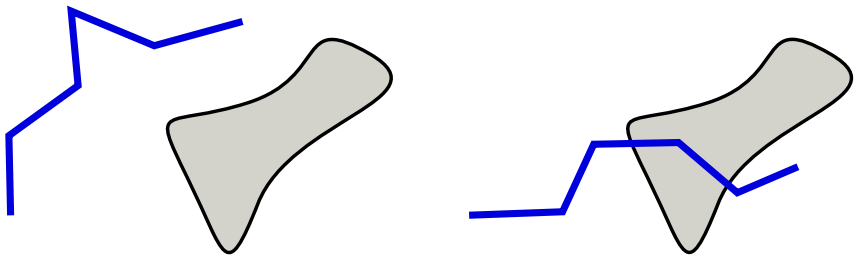
PRM Applied to Articulated Chain

$q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow \textsc{Sample}()$
- $\theta_i \leftarrow \textsc{rand}(-\pi, \pi)$, $\forall i \in [1, n]$



PRM Applied to Articulated Chain

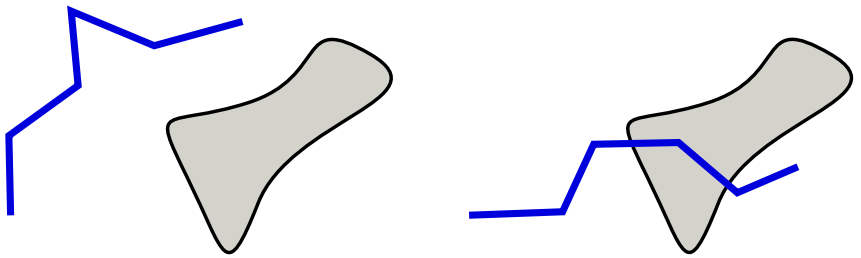
$q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow \textsc{Sample}()$
- $\theta_i \leftarrow \textsc{rand}(-\pi, \pi)$, $\forall i \in [1, n]$



$\textsc{IsSampleCollisionFree}(q)$
- Place chain in configuration q (forward kinematics)
- Check for collision with obstacles

PRM Applied to Articulated Chain

$q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow \textsc{Sample}()$
- $\theta_i \leftarrow \textsc{rand}(-\pi, \pi)$, $\forall i \in [1, n]$



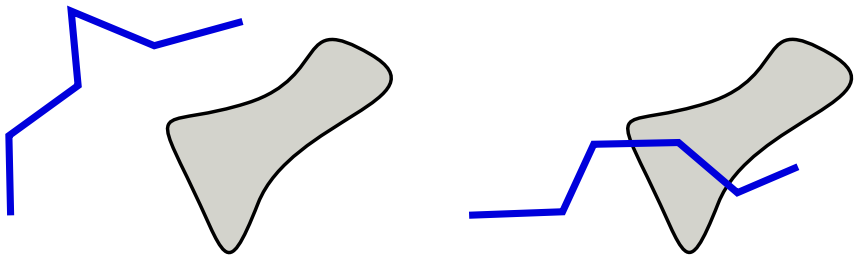
$\textsc{IsSampleCollisionFree}(q)$
- Place chain in configuration q (forward kinematics)
- Check for collision with obstacles

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

PRM Applied to Articulated Chain

$q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow \textsc{Sample}()$
- $\theta_i \leftarrow \textsc{rand}(-\pi, \pi)$, $\forall i \in [1, n]$

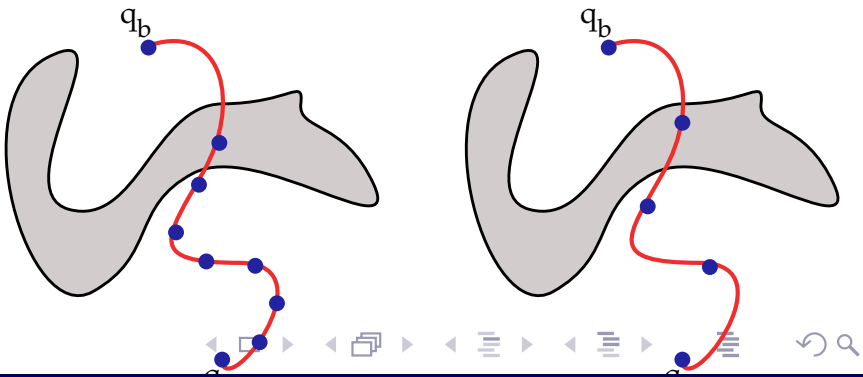


$\textsc{IsSampleCollisionFree}(q)$
- Place chain in configuration q (forward kinematics)
- Check for collision with obstacles

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\textsc{IsPathCollisionFree}(\text{path})$
- Incremental approach
- Subdivision approach



[everest] [skeleton] [knot] [manip]

Path Smoothing

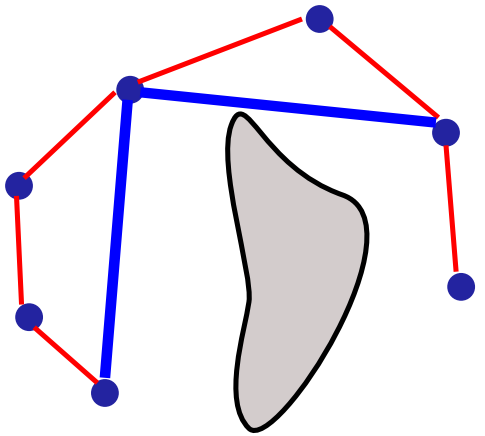
- Solution paths produced by PRM planners tend to be long and non-smooth (due to sampling and edge connections)
- Post processing is commonly used to improve the quality of the paths
- A common practice is to repeatedly replace long paths by short paths

Path Smoothing

- Solution paths produced by PRM planners tend to be long and non-smooth (due to sampling and edge connections)
- Post processing is commonly used to improve the quality of the paths
- A common practice is to repeatedly replace long paths by short paths

SmoothPath($q_1, q_2, \ldots, q_n$) – one version

1: **for** several times **do**
2: select i and j uniformly at random from $1, 2, \ldots, n$
3: attempt to directly connect q_i to q_j
4: if successful, remove the in-between nodes, i.e., $q_{i+1}, \ldots, q_j$

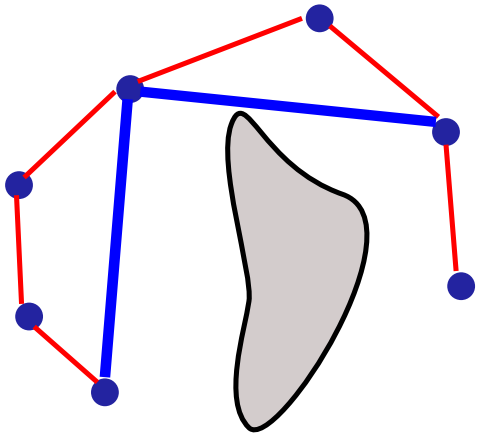


Path Smoothing

- Solution paths produced by PRM planners tend to be long and non-smooth (due to sampling and edge connections)
- Post processing is commonly used to improve the quality of the paths
- A common practice is to repeatedly replace long paths by short paths

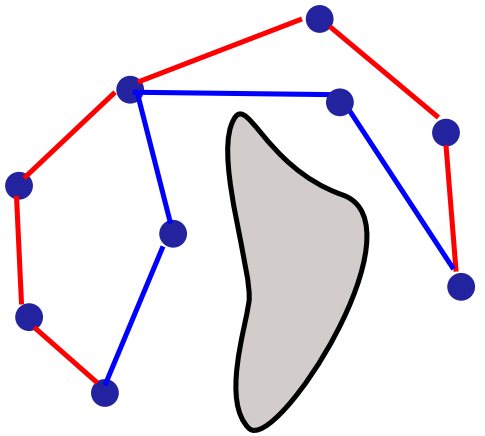
SmoothPath($q_1, q_2, \ldots, q_n$) – one version

1: **for** several times **do**
2: select i and j uniformly at random from $1, 2, \ldots, n$
3: attempt to directly connect q_i to q_j
4: if successful, remove the in-between nodes, i.e., $q_{i+1}, \ldots, q_j$



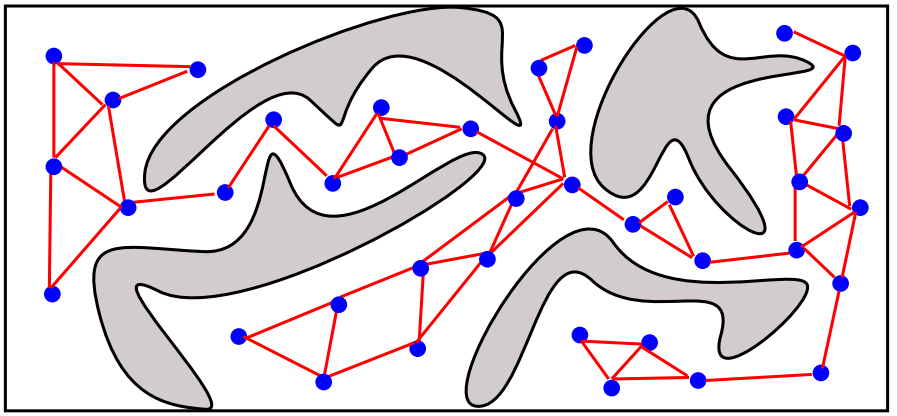
SmoothPath($q_1, q_2, \ldots, q_n$) – another version

1: **for** several times **do**
2: select i and j uniformly at random from $1, 2, \ldots, n$
3: $q \leftarrow$ generate collision-free sample
4: attempt to connect q_i to q_j through q
5: if successful, replace the in-between nodes $q_{i+1}, \ldots, q_j$ by q



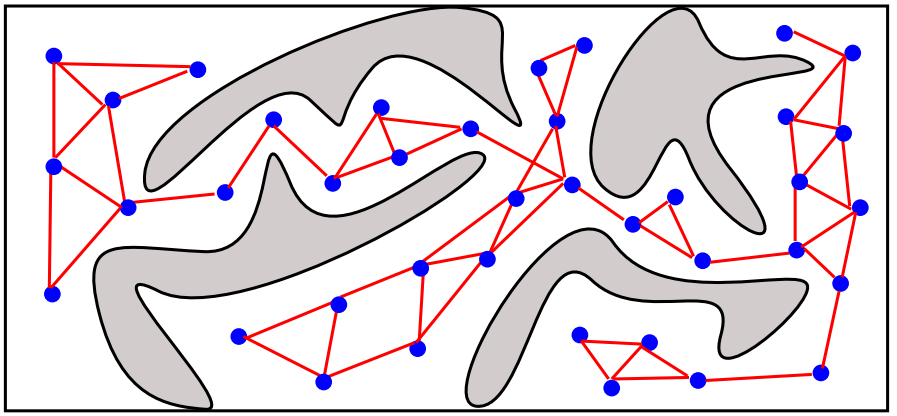
Motivation: Tree-based Motion Planning

- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



Motivation: Tree-based Motion Planning

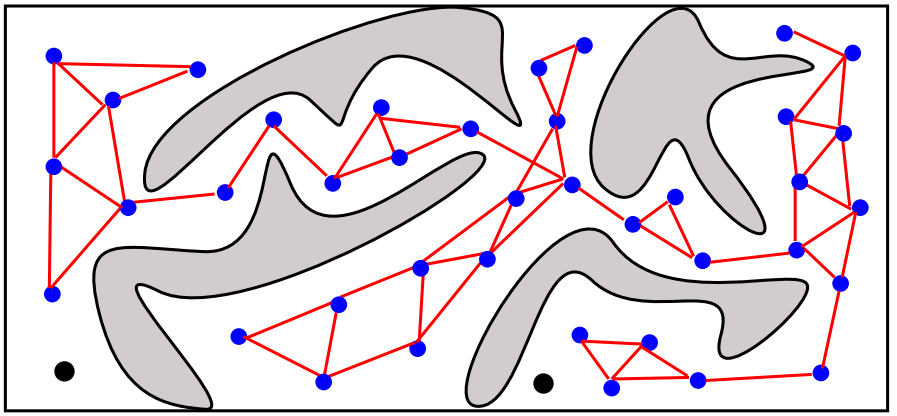
- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



- Good when the objective is to solve *multiple* queries

Motivation: Tree-based Motion Planning

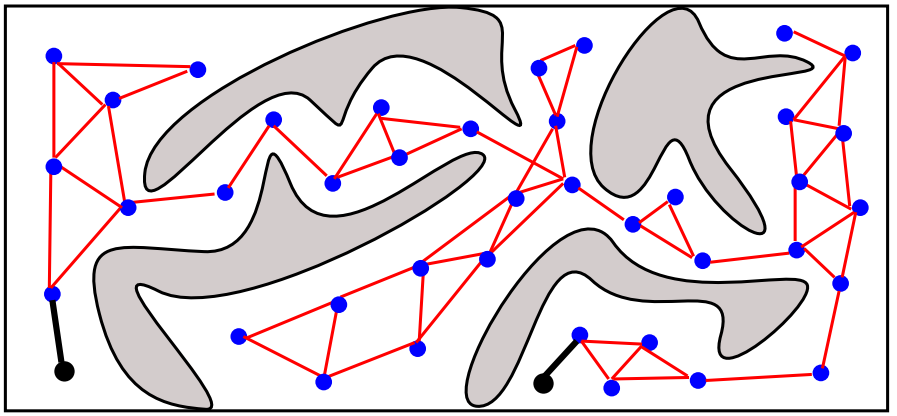
- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



- Good when the objective is to solve *multiple* queries

Motivation: Tree-based Motion Planning

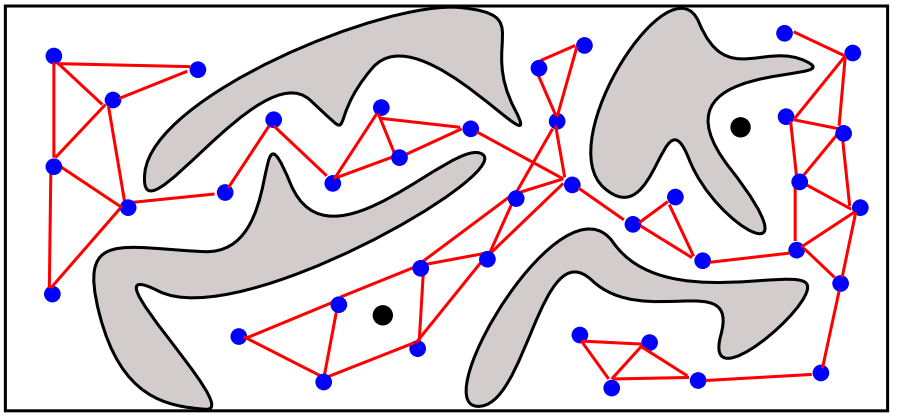
- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



- Good when the objective is to solve *multiple* queries

Motivation: Tree-based Motion Planning

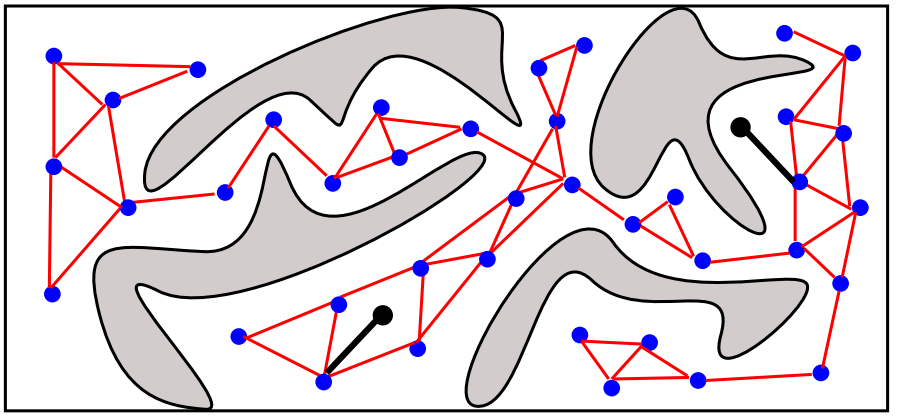
- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



- Good when the objective is to solve *multiple* queries

Motivation: Tree-based Motion Planning

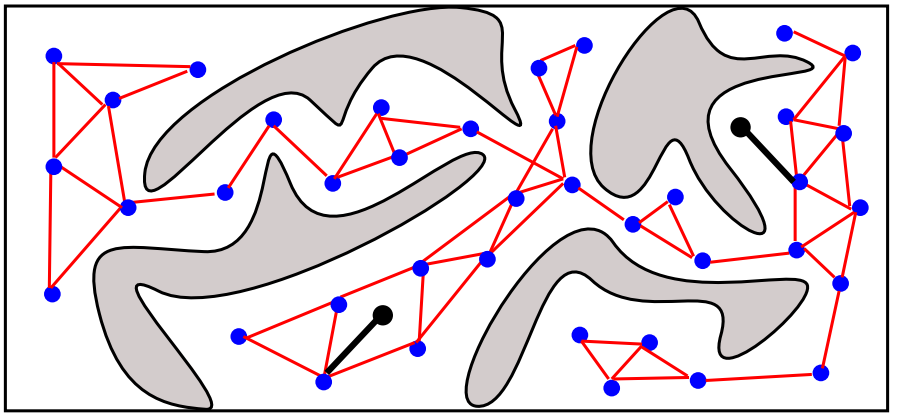
- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



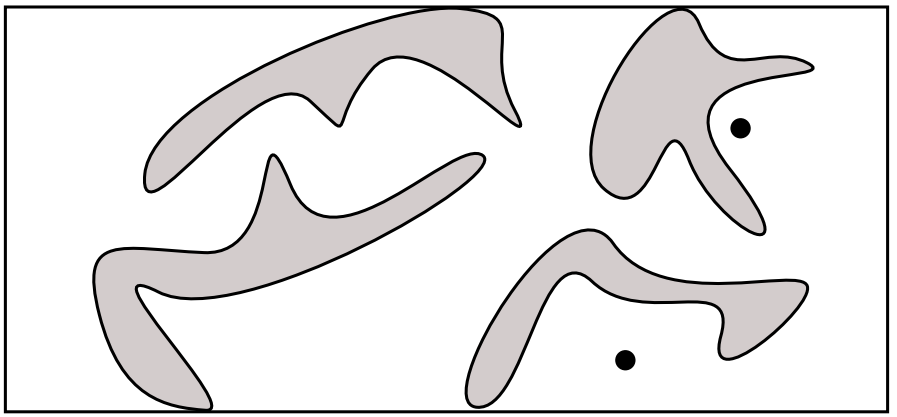
- Good when the objective is to solve *multiple* queries

Motivation: Tree-based Motion Planning

- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space

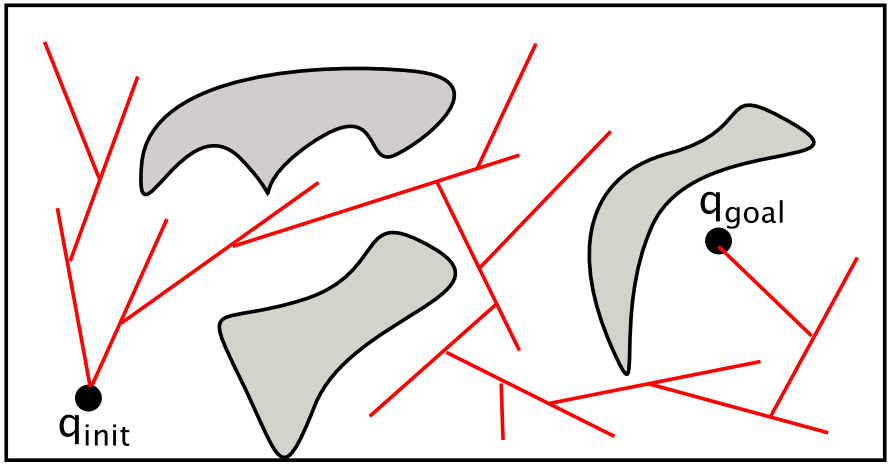


- Good when the objective is to solve *multiple* queries
- Maybe a bit too much when the objective is to solve a *single* query



General Idea

Grow a tree in the free configuration space from q_{init} toward q_{goal}



TREESEARCHFRAMEWORK($q_{\text{init}}, q_{\text{goal}}$)
1: $\mathcal{T} \leftarrow$ ROOTTREE(q_{init})
2: **while** q_{goal} has not been reached **do**
3: $q \leftarrow$ SELECTCONFIGFROMTREE($\mathcal{T}$)
4: ADDTREEBRANCHFROMCONFIG($\mathcal{T}, q$)

Critical Issues

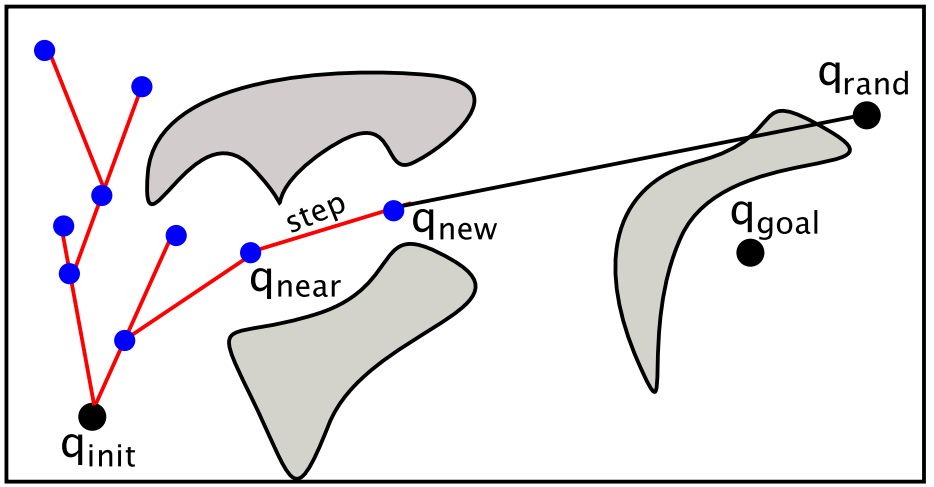
- How should a configuration be selected from the tree?
- How should a new branch be added to the tree from the selected configuration?

Rapidly-exploring Random Tree (RRT)

Pull the tree toward random samples in the configuration space

- RRT relies on nearest neighbors and distance metric $\rho : Q \times Q \leftarrow \mathbb{R}^{\geq 0}$
- RRT adds Voronoi bias to tree growth

[LaValle, Kuffner: 1999]



$\mathrm{RRT}(q_{\mathrm{init}}, q_{\mathrm{goal}})$

▷ *initialize tree*
1: $\mathcal{T} \leftarrow$ create tree rooted at q_{init}

2: **while** solution not found **do**

▷ *select configuration from tree*
3: $q_{\mathrm{rand}} \leftarrow$ generate a random sample
4: $q_{\mathrm{near}} \leftarrow$ nearest configuration in $\mathcal{T}$ to q_{rand} according to distance ρ

▷ *add new branch to tree from selected configuration*
5: $\mathrm{path} \leftarrow$ generate path (not necessarily collision free) from q_{near} to q_{ra}
6: **if** $\textsc{IsSubpathCollisionFree}(\mathrm{path}, 0, \mathtt{step})$ **then**
7: $q_{\mathrm{new}} \leftarrow \mathrm{path}(\mathtt{step})$
8: add configuration q_{new} and edge $(q_{\mathrm{near}}, q_{\mathrm{new}})$ to $\mathcal{T}$

▷ *check if a solution is found*
9: **if** $\rho(q_{\mathrm{new}}, q_{\mathrm{goal}}) \approx 0$ **then**
10: **return** solution path from root to q_{new}

Rapidly-exploring Random Tree (RRT) (cont.)

Aspects for Improvement

Suggested Improvements in the Literature

Rapidly-exploring Random Tree (RRT) (cont.)

Aspects for Improvement

- BasicRRT does not take advantage of q_{goal}
- Tree is pulled towards random directions based on the uniform sampling of Q
- In particular, tree growth is not directed towards q_{goal}

Suggested Improvements in the Literature

Rapidly-exploring Random Tree (RRT) (cont.)

Aspects for Improvement

- BasicRRT does not take advantage of q_{goal}
- Tree is pulled towards random directions based on the uniform sampling of Q
- In particular, tree growth is not directed towards q_{goal}

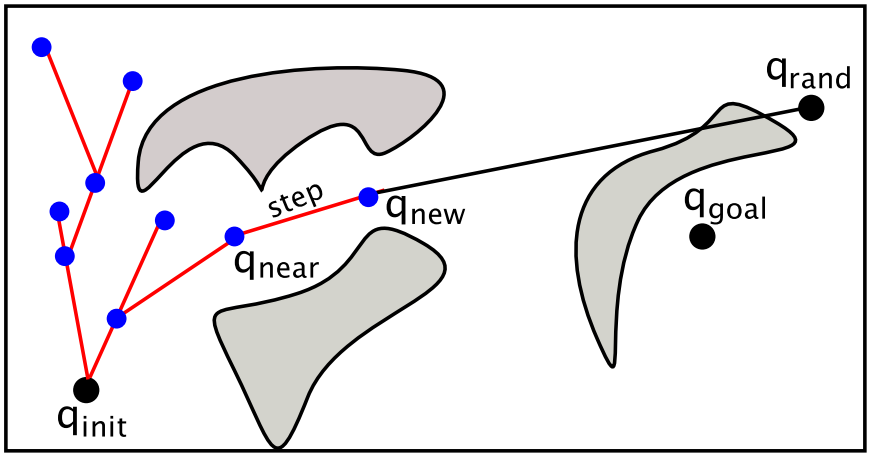
Suggested Improvements in the Literature

- Introduce goal-bias to tree growth (known as GoalBiasRRT)
  - q_{rand} is selected as q_{goal} with probability p
  - q_{rand} is selected based on uniform sampling of Q with probability $1 - p$
  - Probability p is commonly set to ≈ 0.05

Rapidly-exploring Random Tree (RRT) (cont.)

Aspects for Improvement

- BasicRRT takes only one small step when adding a new tree branch



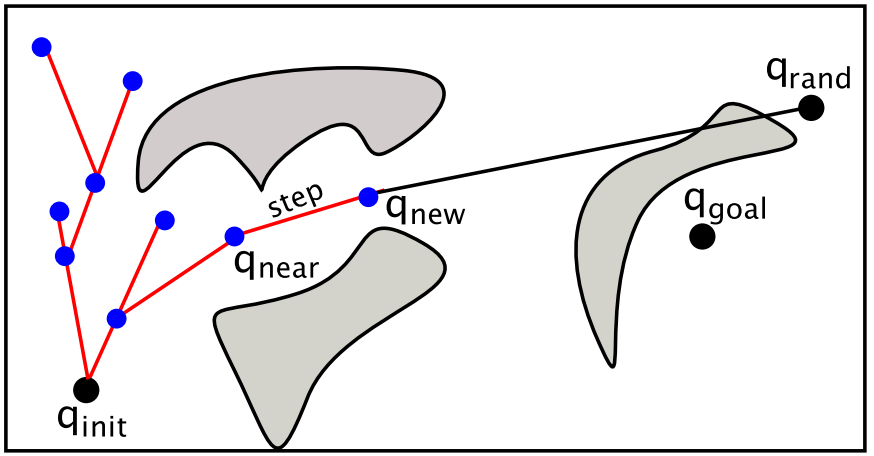
- This slows down tree growth

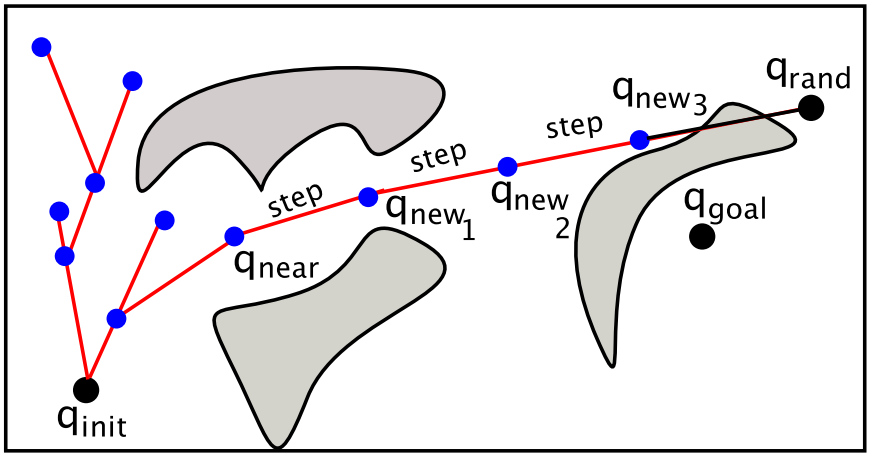
Suggested Improvements in the Literature

Rapidly-exploring Random Tree (RRT) (cont.)

Aspects for Improvement

- BasicRRT takes only one small step when adding a new tree branch





- This slows down tree growth

Suggested Improvements in the Literature

- Take several steps until q_{rand} is reached or a collision is found (ConnectRRT)
- Add all the intermediate nodes to the tree

Observations in High-Dimensional Problems

- Tree generally grows rapidly for the first few thousand iterations
- Tree growth afterwards slows down quite significantly
- Large number of configurations increases computational cost
- It becomes increasingly difficult to guide the tree towards previously unexplored parts of the free configuration space

Observations in High-Dimensional Problems

- Tree generally grows rapidly for the first few thousand iterations
- Tree growth afterwards slows down quite significantly
- Large number of configurations increases computational cost
- It becomes increasingly difficult to guide the tree towards previously unexplored parts of the free configuration space

Possible improvements?

Bi-directional Trees

Grow two trees, rooted at q_{init} and q_{goal}, towards each other

- Bi-directional trees improve computational efficiency compared to a single tree
- Growth slows down significantly later than when using a single tree
- Fewer configurations in each tree, which imposes less of a computational burden
- Each tree explores a different part of the configuration space

BITREE($q_{\text{init}}, q_{\text{goal}}$)

1: $\mathcal{T}_{\text{init}} \leftarrow$ create tree rooted at q_{init}
2: $\mathcal{T}_{\text{goal}} \leftarrow$ create tree rooted at q_{goal}
3: **while** solution not found **do**
4: add new branch to $\mathcal{T}_{\text{init}}$
5: add new branch to $\mathcal{T}_{\text{goal}}$
6: attempt to connect neighboring configurations from the two trees
7: if successful, return path from q_{init} to q_{goal}

Advantages

- Explores small subset of possibilities by sampling
- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Is it then just a heuristic approach? No. It's more than that

It offers *probabilistic completeness*

- When a solution exists, a probabilistically complete planner finds a solution with probability as time goes to infinity.
- When a solution does not exists, a probabilistically complete planner may not be able to determine that a solution does not exist.

Motion-Planning Problem for Systems with Differential Constraints

Given

- State space S
- Control space U
- Equations of motions as differential equations $f : S \times U \rightarrow \dot{S}$
- State-validity function $\textsc{valid} : S \rightarrow \{\texttt{true}, \texttt{false}\}$, e.g, check collisions
- Goal function $\textsc{goal} : S \rightarrow \{\texttt{true}, \texttt{false}\}$
- Initial state s_0

Compute a control trajectory $u : [0, T] \rightarrow U$ such that the resulting state trajectory $s : [0, T] \rightarrow S$ obtained by integration is valid and reaches the goal, i.e.,

$$s(t) = s_0 + \int_{h=0}^{h=t} f(s(t), u(t))dh \qquad (1)$$

$$\forall t \in [0, T] : \textsc{valid}(s(t)) = \texttt{true} \qquad (2)$$

$$\exists t \in [0, T] : \textsc{goal}(s(t)) = \texttt{true} \qquad (3)$$

Motion-Planning Methods for Systems with Differential Constraints

Decoupled approach

1. Compute a geometric solution path ignoring differential constraints
2. Transform the geometric path into a trajectory that satisfies the differential constraints

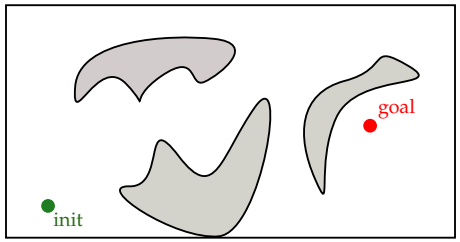
Sampling-based Motion Planning

- Take the differential constraints into account during motion planning

Roadmap Approaches

0. Initialization

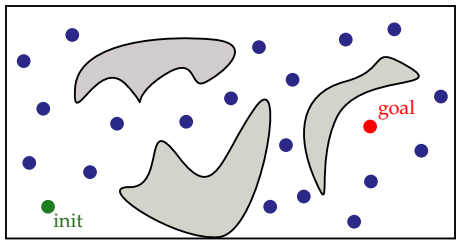
add s_{init} and s_{goal} to roadmap vertex set V



1. Sampling

repeat several times

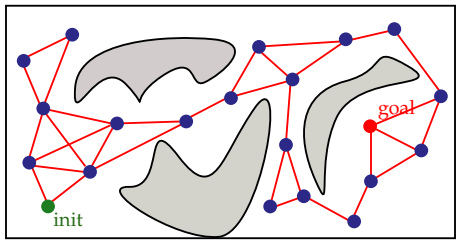
$s \leftarrow \textsc{StateSample}()$
if $\textsc{IsStateValid}(s) = \texttt{true}$
add s to roadmap vertex set V



2. Connect Samples

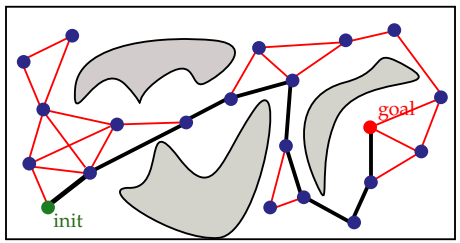
for each pair of neighboring samples $(s_a, s_b) \in V \times V$

$\lambda \leftarrow \textsc{GenerateLocalTrajectory}(s_a, s_b)$
if $\textsc{IsTrajectoryValid}(\lambda) = \texttt{true}$
add (s_a, s_b) to roadmap edge set E



3. Graph Search

search graph (V, E) for path from s_{init} to s_{goal}



Implementation of Roadmap Approaches with Differential Constraints

$s \leftarrow$ StateSample()

- generate random values for all the state components

IsStateValid(s)

- place the robot in the configuration specified by the position and orientation components of the state
- check if the robot collides with the obstacles
- check if velocity and other state components are within desired bounds

IsTrajectoryValid(s)

- use subdivision or incremental approach to check if intermediate states are valid

$\lambda \leftarrow$ GenerateLocalTrajectory(s_a, s_b)

- linear interpolation between s_a and s_b won't work as it does not respect underlying differential constraints
- need to find control function $u : [0, T] \rightarrow U$ such that trajectory obtained by applying u to s_a for T time units ends at s_b
- known as two-point boundary value problem
- cannot always be solved analytically, and numerical solutions increase computational cost

Tree Approaches with Differential Constraints

RRT

1: $\mathcal{T} \leftarrow$ create tree rooted at s_{init}
2: **while** solution not found **do**
▷ *select state from tree*
3: $s_{\text{rand}} \leftarrow \textsc{StateSample}()$
4: $s_{\text{near}} \leftarrow$ nearest configuration in $\mathcal{T}$ to q_{rand} according to distance ρ
▷ *add new branch to tree from selected configuration*
5: $\lambda \leftarrow \textsc{GenerateLocalTrajectory}(s_{\text{near}}, s_{\text{rand}})$
6: **if** $\textsc{IsSubTrajectoryValid}(\lambda, 0, \texttt{step})$ **then**
7: $s_{\text{new}} \leftarrow \lambda(\texttt{step})$
8: add configuration s_{new} and edge $(s_{\text{near}}, s_{\text{new}})$ to $\mathcal{T}$
▷ *check if a solution is found*
9: **if** $\rho(s_{\text{new}}, s_{\text{goal}}) \approx 0$ **then**
10: **return** solution trajectory from root to s_{new}

✓ $\textsc{StateSample}()$: random values for state components
✓ $\rho : S \times S \rightarrow \mathbb{R}^{\geq 0}$: distance metric between states
✓ $\textsc{IsSubTrajectoryValid}(\lambda, 0, \texttt{step})$: incremental approach

$\lambda \leftarrow \textsc{GenerateLocalTrajectory}(s_{\text{near}}, s_{\text{rand}})$

- will it not create the same two-boundary value problems as in PRM?
- is it necessary to connect to s_{rand}?
- would it suffice to just come close to s_{rand}?

Avoiding Two-Boundary Value Problem

Rather than computing a trajectory from s_{near} to s_{rand} compute a trajectory that starts at s_{near} and extends toward s_{rand}

Approach 1 – extend according to random control

- Sample random control u in U
- Integrate equations of motions when applying u to s_{near} for Δt units of time, i.e.,

$$\lambda \rightarrow s(t) = s_{\text{near}} + \int_{h=0}^{h=\Delta t} f(s(t), u) dh$$

Approach 2 – find the best-out-of-many random controls

1. for $i = 1, \ldots, m$ do
   1. $u_i \leftarrow$ sample random control in U
   2. $\lambda_i \rightarrow s(t) = s_{\text{near}} + \int_{h=0}^{h=\Delta t} f(s(t), u_i) dh$
   3. $d_i \leftarrow \rho(s_{\text{rand}}, \lambda_i(\Delta t))$
2. return λ_i with minimum d_i

Tree approaches require only the ability to simulate robot motions

- Physics engines can be used to simulate robot motions
- Physics engines provide greater simulation accuracy
- Physics engines can take into account friction, gravity, and interactions of the robot with objects in the evironment