Summer School Robotics

Week 2 – Motion Planning for Industrial Robots

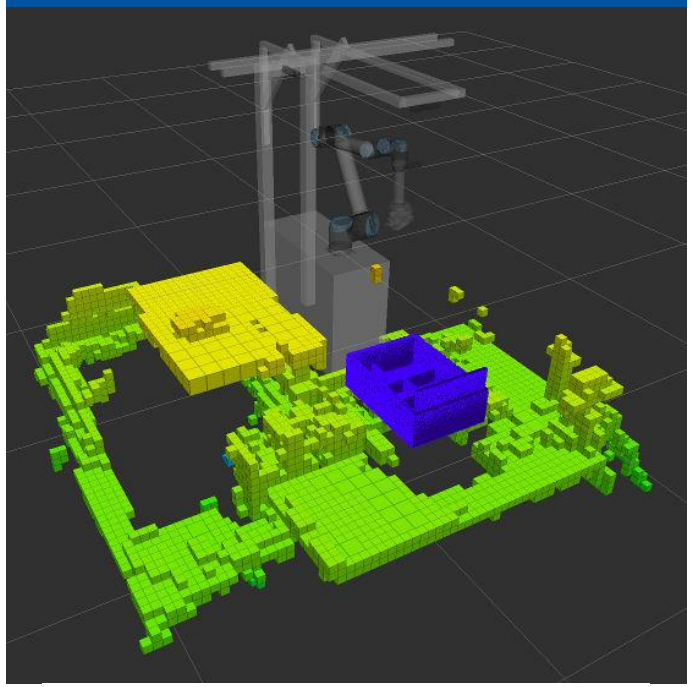
an Comprehensive Introduction

Haoming Zhang, M.Sc.
Philipp Ennen, M.Sc.

Robots as an Example for Intelligent Machines

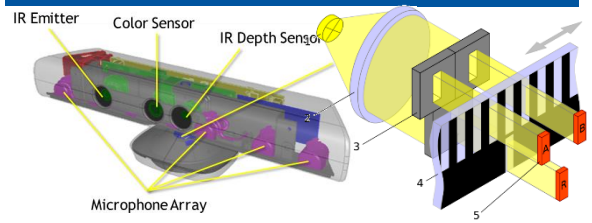
How do I (the robot) go there?

Model of the Robot and Environment



$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} u$$

Sensing

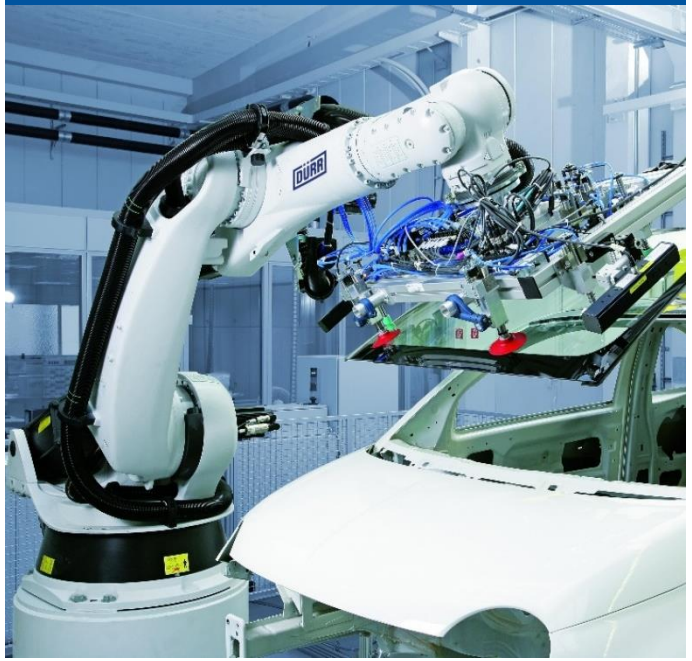


This weeks topic!

Motion Planning

Requires Goalstate:
- i.e. hand-engineered
- i.e. via a cost function

Motion Control and Execution



IfU IMA ZLW RWTH AACHEN UNIVERSITY

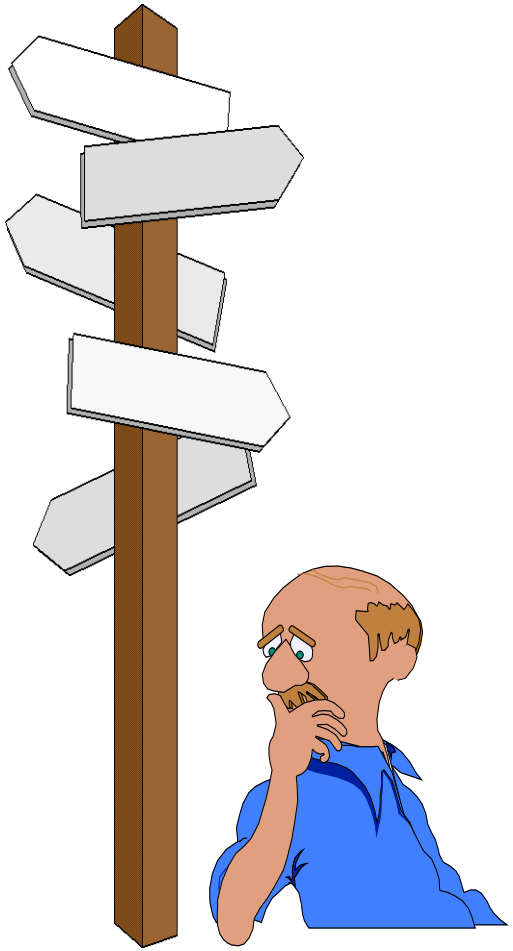
Outline

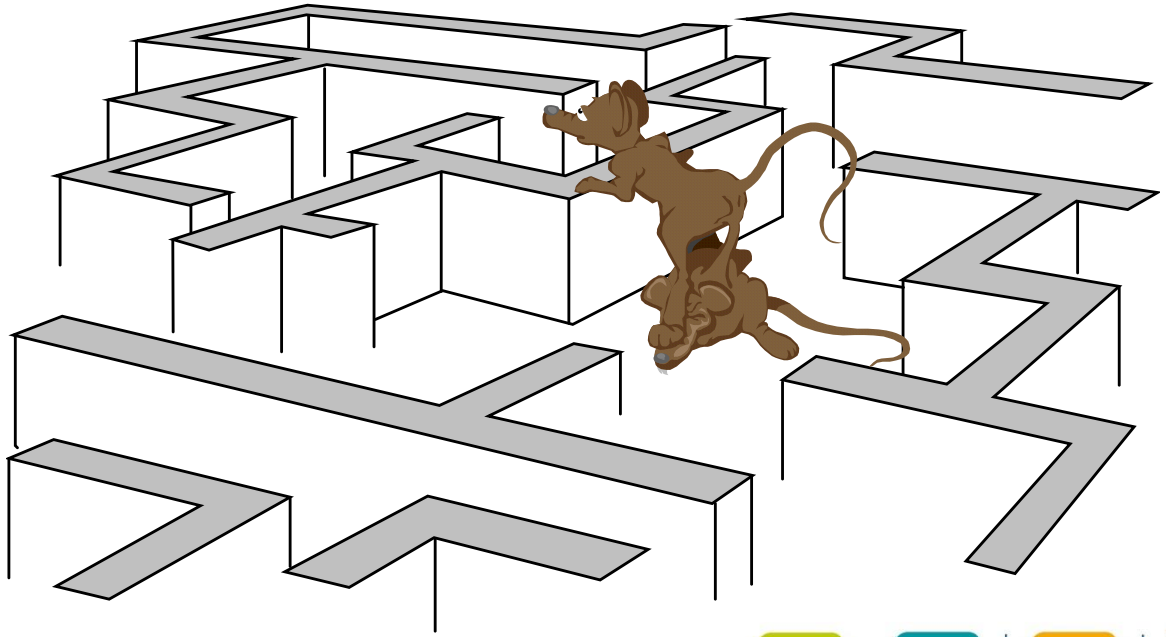
Outline

Motion Planning: Introduction

What is Motion Planning?

Determining **where to move** without **hit obstacles.**

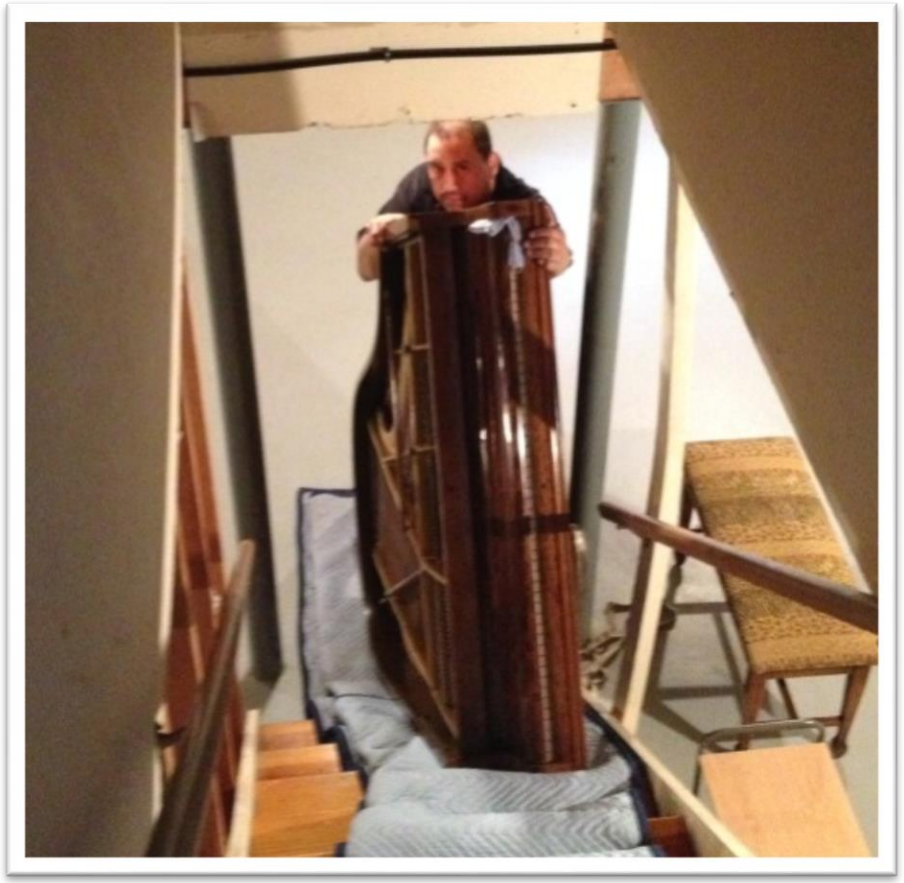




Motion Planning: Introduction

Piano Mover‘s Problem for you

Move the **Piano** in your **house** **„peacefully“** from **one room** to **another.**

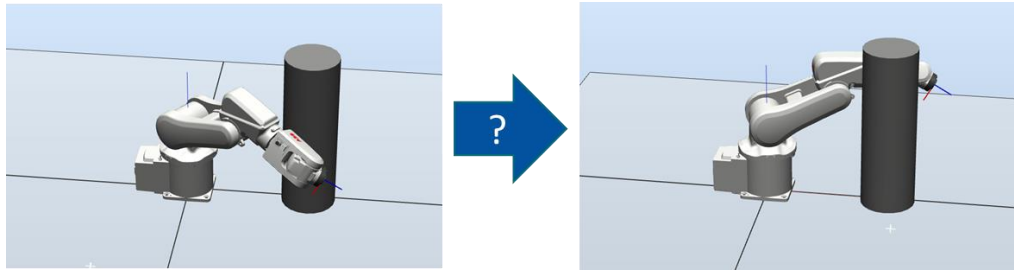


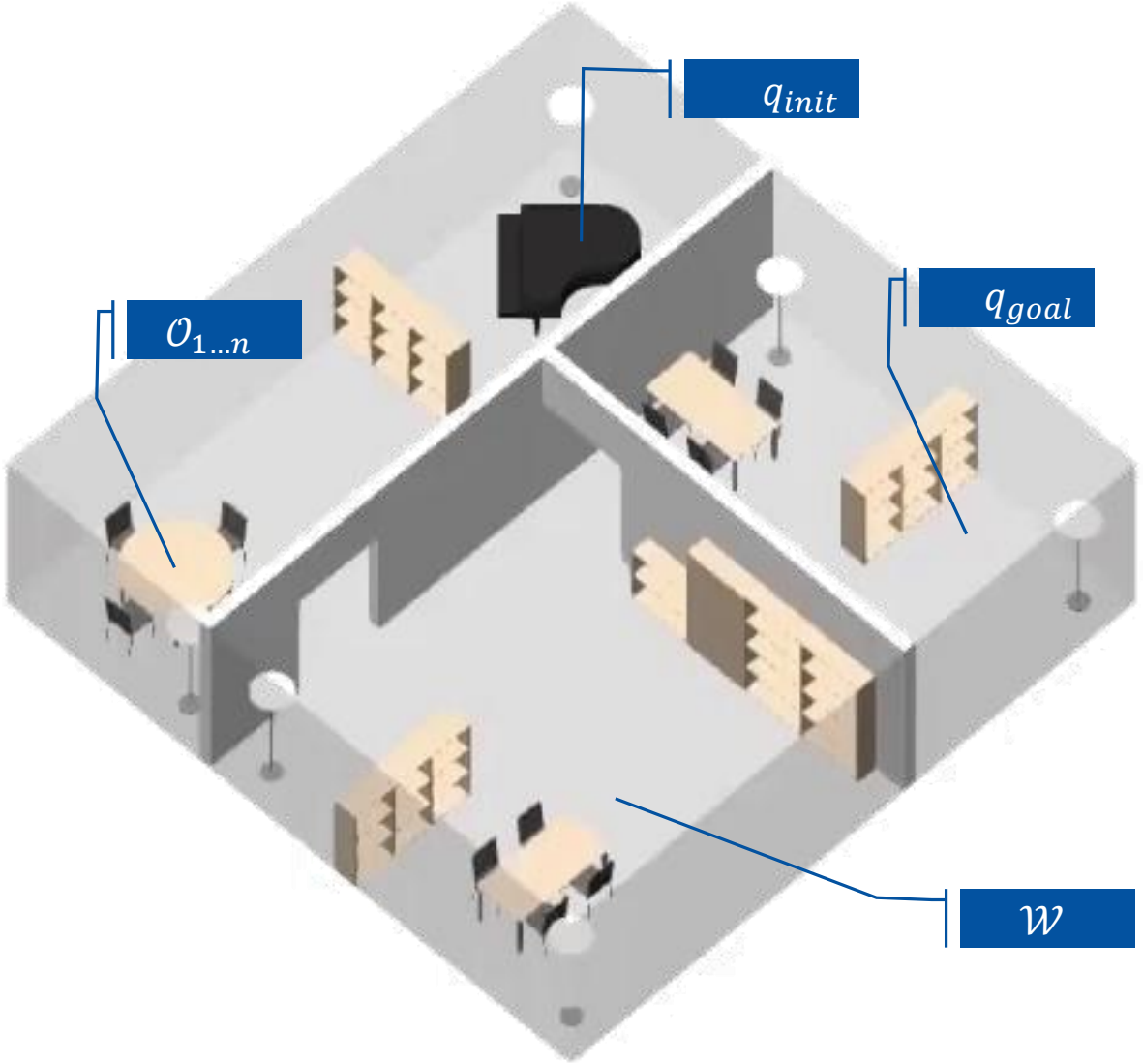
Motion Planning: Introduction

Piano Mover ‘s Problem as a robot

- Problem Formulation: Given
  - A workspace $\mathcal{W}$ of $\mathbb{R}^2$ or $\mathbb{R}^3$
  - A set of (dynamical or statical) obstacles $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$
  - A start position q_{init} and a goal position q_{goal}
  - Geometry and kinematics of the robot

- Problem Formulation: Find a path τ from q_{init} to q_{goal} that:
  - is (self-) collision-free
  - satisfies (joint, velocity, acceleration) limits
  - …





Motion Planning: Introduction

Goal of Motion Planning

- Compute motion strategies, e.g.:
  - geometric paths
  - time-parameterized trajectories
  - sequence of sensor-based motion commands

- Achieve high-level goals, e.g.:
  - Goto A without colliding with obstacles
  - Assemble product P
  - Build a map of the hallway
  - Find and track the target

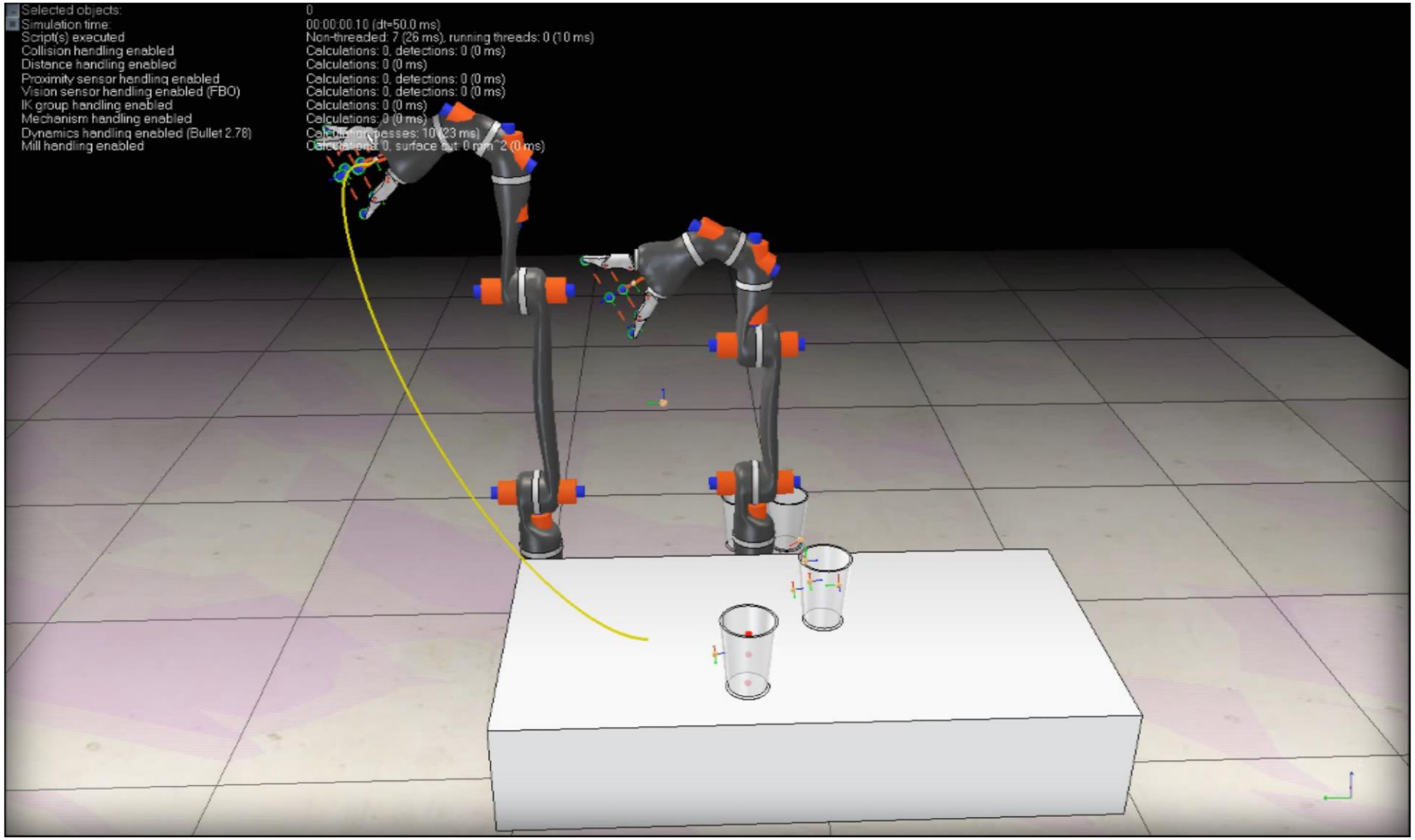
"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

[Source: Willow Garage]

Motion Planning: Introduction

Planning involves

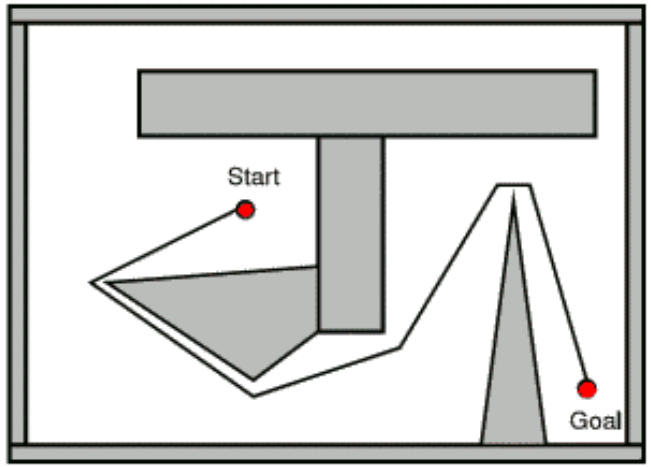
- State Space
- Time
- Action
- Initial and goal state
- A Plan
- A Criterion



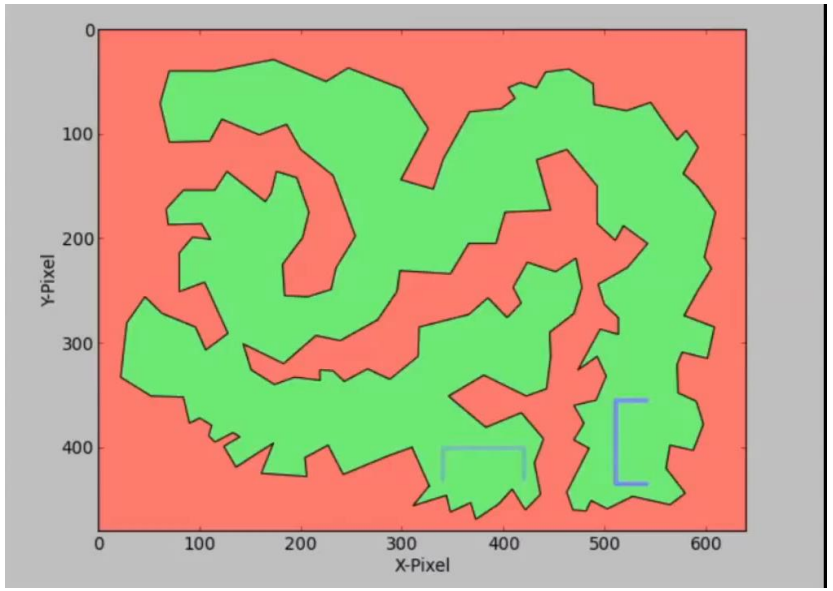
[Source: V-REP, coppeliarobotics]

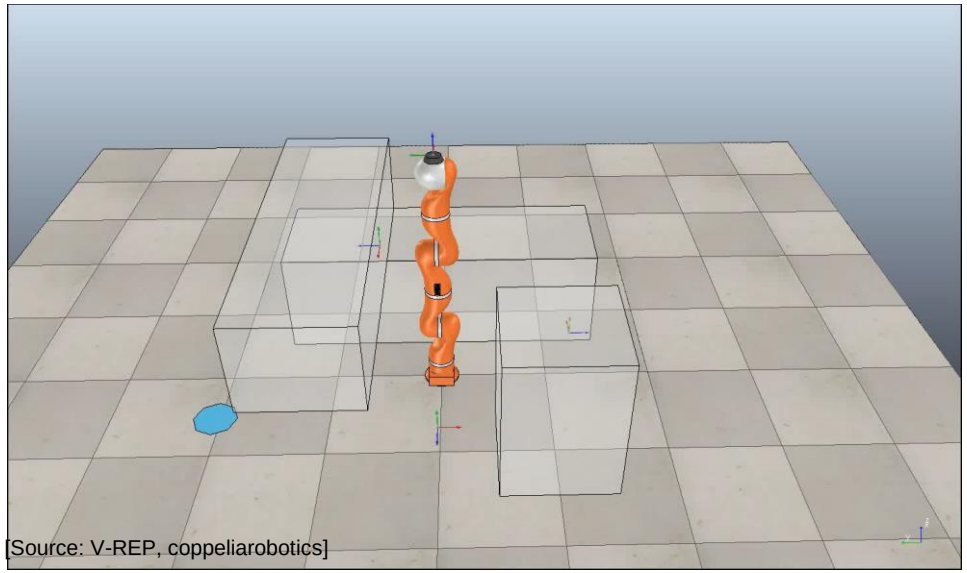
Motion Planning: Introduction

Some Motion Planning Tasks

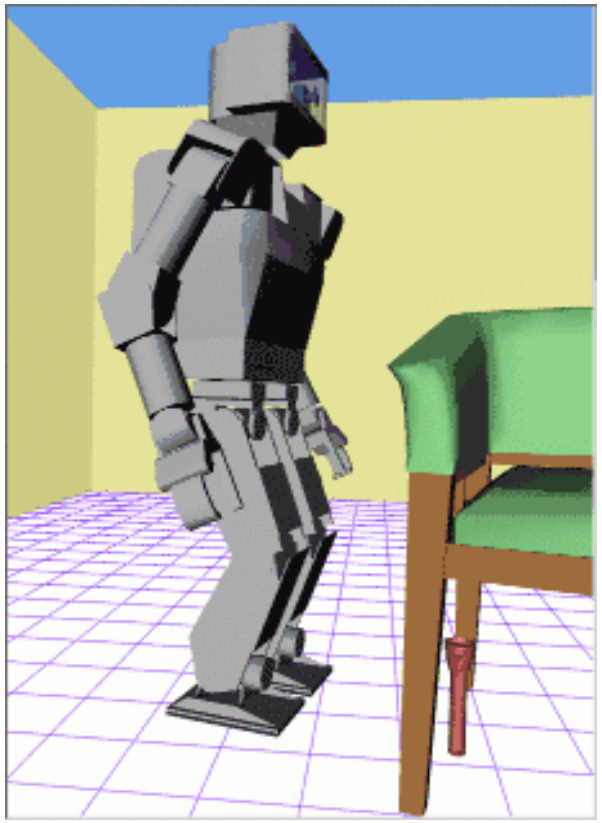


[Source: http://www.societyofrobots.com/robot_arm_tutorial.shtml]





[Source: V-REP, coppeliarobotics]

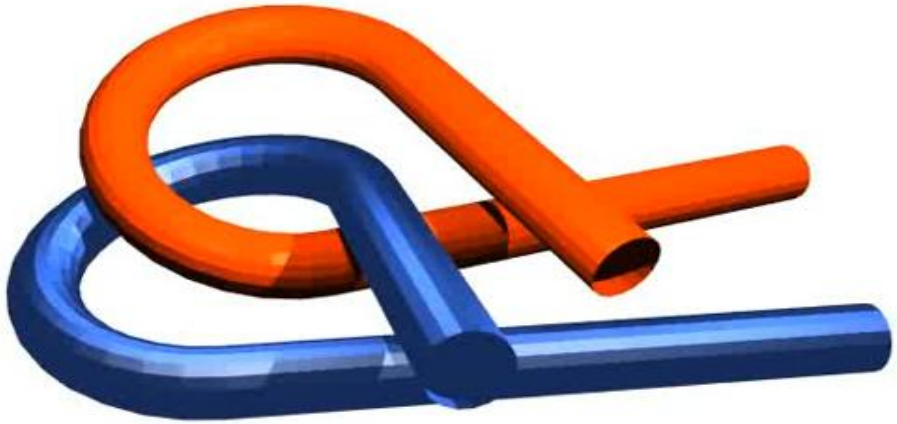


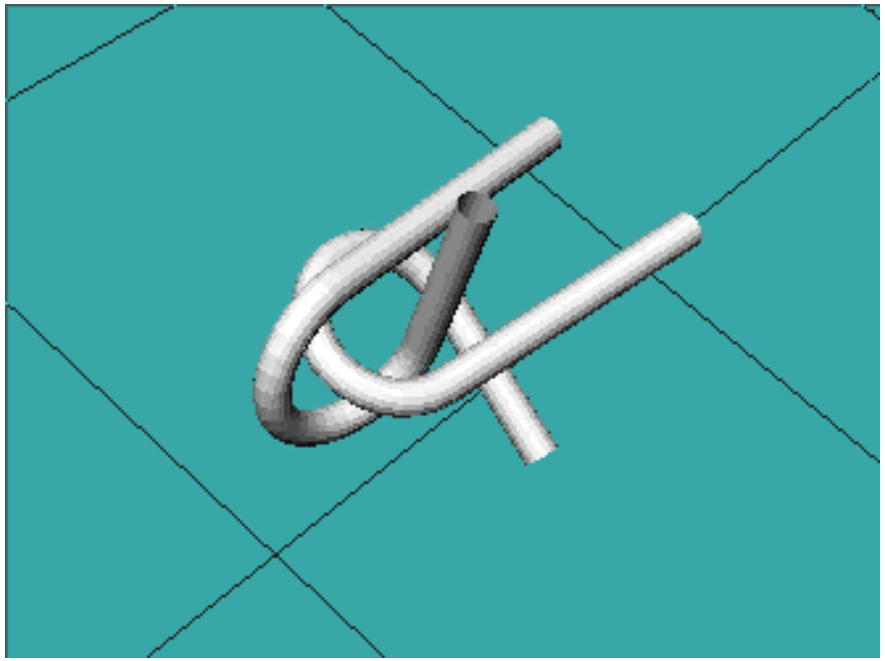
[Source: https://www.youtube.com/watch?v=UuZWCVxWAsI]

Motion Planning: Introduction

Planning involves: Is it easy?

- State Space
- Time
- Action
- Initial and goal state
- A Plan
- A Criterion





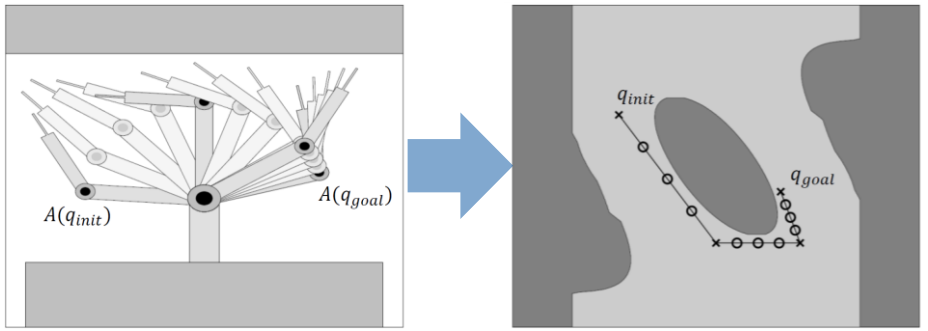
[Source: https://www.youtube.com/watch?v=UTbiAu8IXas]

Outline

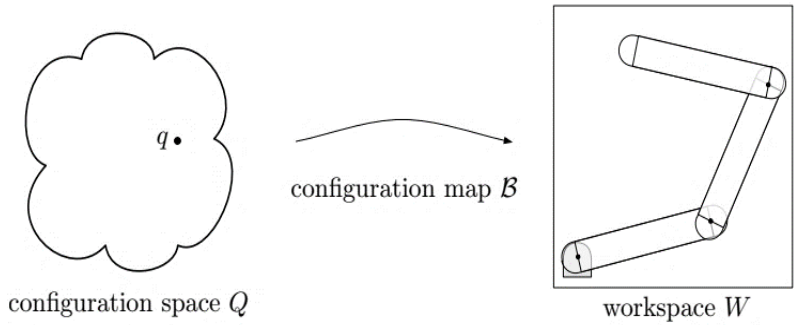
Motion Planning: Configuration Space

Context of plan: Workspaces and Configuration Spaces

- Definition of Spaces:
  - Workspace $\mathcal{W} \in \mathbb{R}^n$: Environment in which robot operates
  - Obstacle space $\mathcal{O} \in \mathbb{R}^n$: Already occupied spaces of the world
  - Free Space $\mathcal{W}_{free} \in \mathbb{R}^n$: Unoccupied space of the world
- Projection of robot and workspace into a high dimensional configuration space
  - Configuration q : A minimal set of variables that describes the position of each rigid body component of a robot, e.g. joint positions and velocities
  - Configuration space $\mathcal{C}_{space}$: A set of possible configurations the robot could take
  - C-space obstacle region $\mathcal{C}_{obs}$: $\mathcal{C}_{obs} = \{q \in \mathcal{C}_{space} | \mathcal{B}(q) \cap \mathcal{O} \neq \emptyset\}$
  - C-space without collision $\mathcal{C}_{free}$: $\mathcal{C}_{free} = \mathcal{C}_{space} \backslash \mathcal{C}_{obs}$

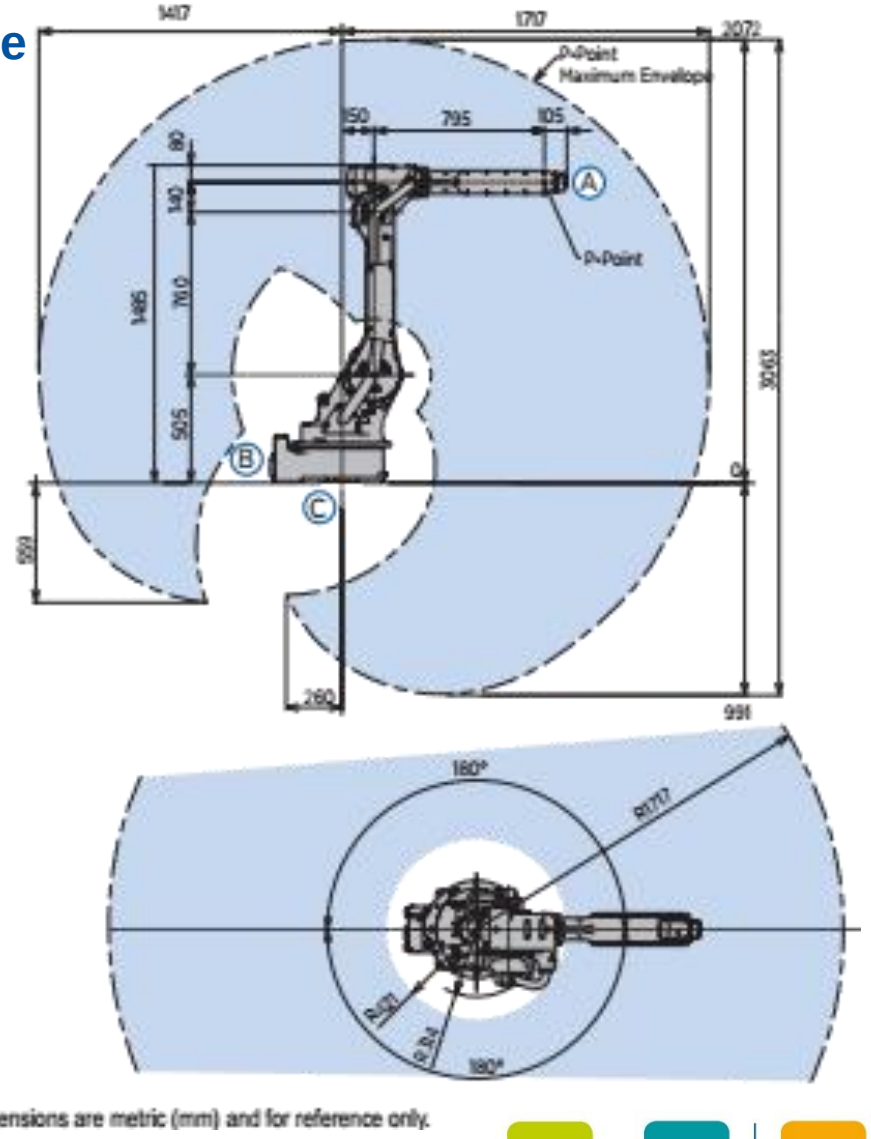


Workspace Configuration Space



Motion Planning: Configuration Space

Context of plan: Workspace and Configuration Space



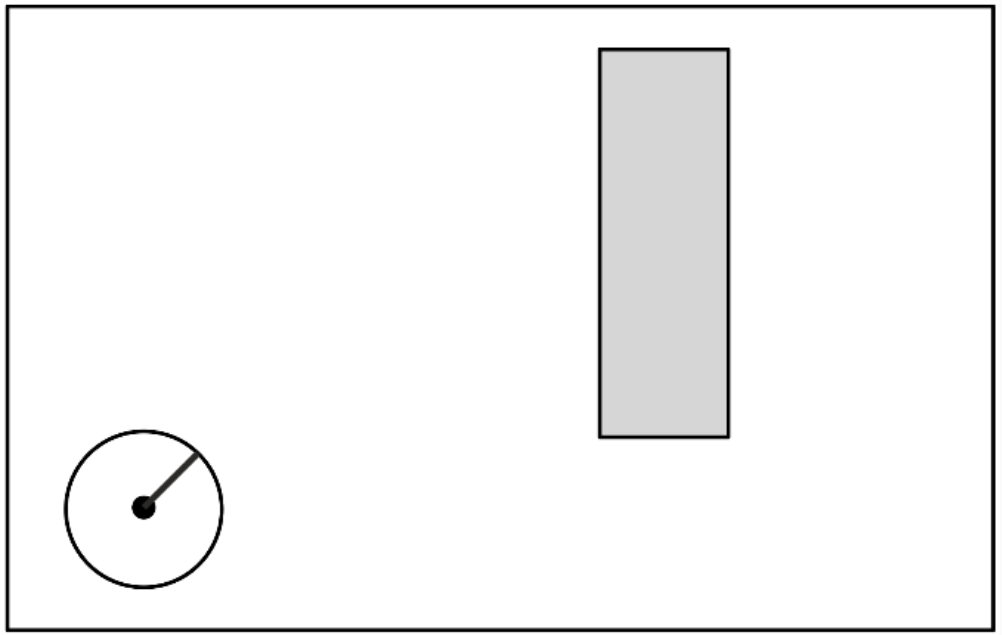
All dimensions are metric (mm) and for reference only.

Motion Planning: Configuration Space

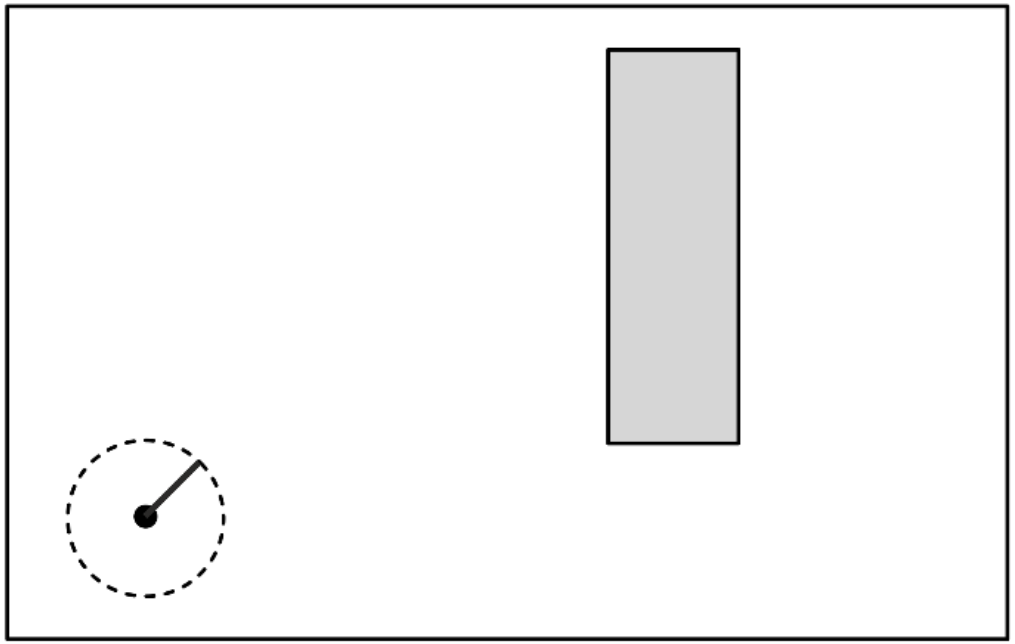
Context of plan: Workspace and Configuration Space

- Small quiz:
  - Configuration?
  - Free Configuration space?

Free Workspace



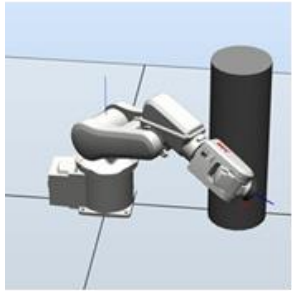
Free Configuration space

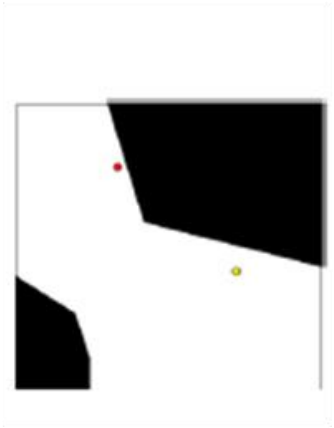


Motion Planning: Configuration Space

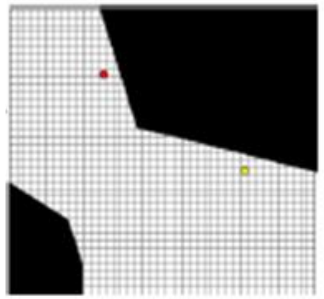
Motion Planning Pipeline within $\mathcal{C}_{space}$

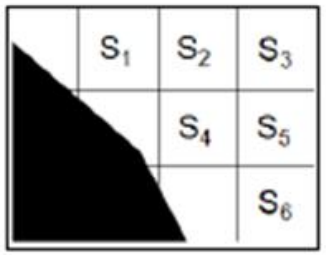
transform workspace into **configuration space**



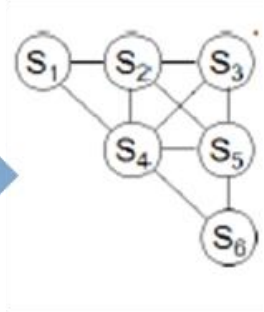


discretize configuration space

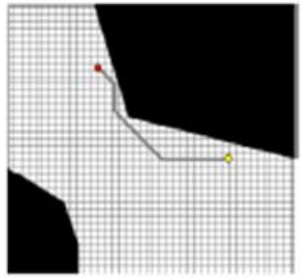




connect collision-free state transistions



perform graph or tree **search**

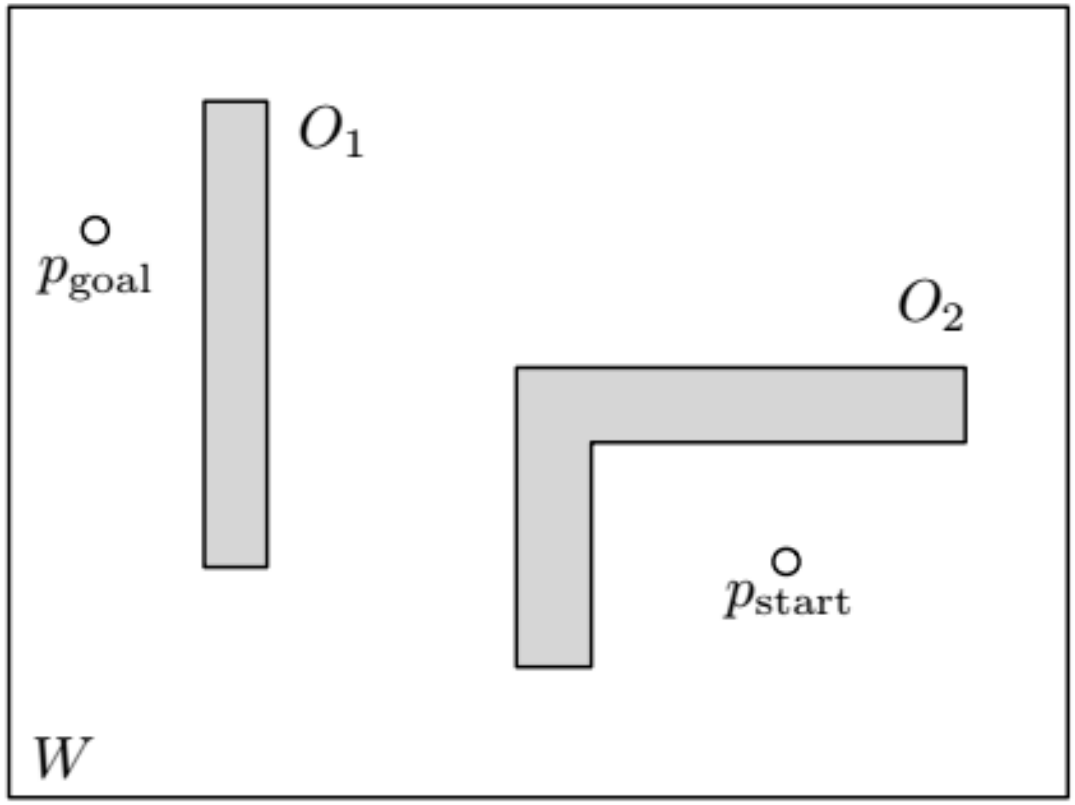


Outline

Motion Planning: Classical Approaches

Sensor-based Planning: the Bug Algorithms

- Problem setup:
  - A workspace $\mathcal{W}$ of $\mathbb{R}^2$ or $\mathbb{R}^3$
  - Some obstacles $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$
  - A state position q_{init} and a goal position q_{goal}
  - A robot described by a moving point (no size)
- Assumptions
  - Robot Assumptions:
    - Knows the direction towards the goal and distance to goal
    - Does not know anything about the obstacles (without map)
    - A (contact) Sensor to detect obstacles
    - Move either in a straight line or follow an obstacle boundary
    - Limited memory to store distances and angles
  - Envionment Assumptions:
    - The workspace is bounded
    - Finite number of obstacles
    - Start and goal position in free Workspace $\mathcal{W}_{free}$



Motion Planning: Classical Approaches

Sensor-based Planning: the Bug Algorithms

- Bug 0:

Algorithm: Bug 0

while not at goal:
    move towards the goal
    if hit an obstacle:
        while not able to move towards the goal:
            follow the obstacle's boundary moving to the left

- Bug 1:

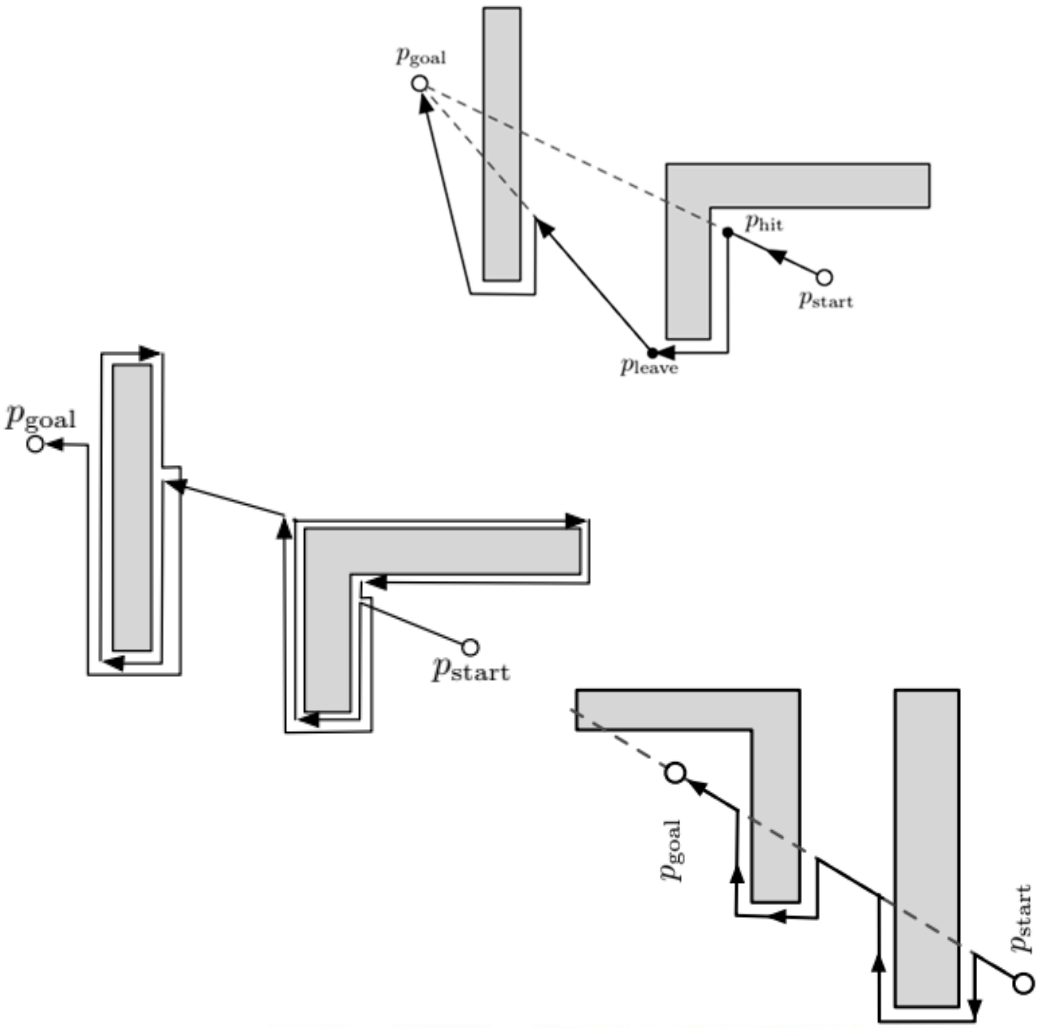
Algorithm: Bug 1

while not at goal:
    move towards the goal
    if hit an obstacle:
        circumnavigate it.
        store in memory the minimum distance from the obstacle boundary to the goal
        follow the boundary back to the boundary point with minimum distance to goal

- Bug 2:

Algorithm: Bug 2

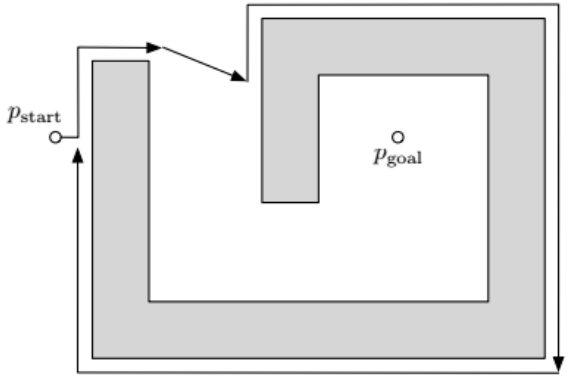
while not at goal:
    move towards the goal (along the start − goal line)
    if hit an obstacle:
        follow the boundary (either left or right)
        leave the boundary until you encounter the start - goal line



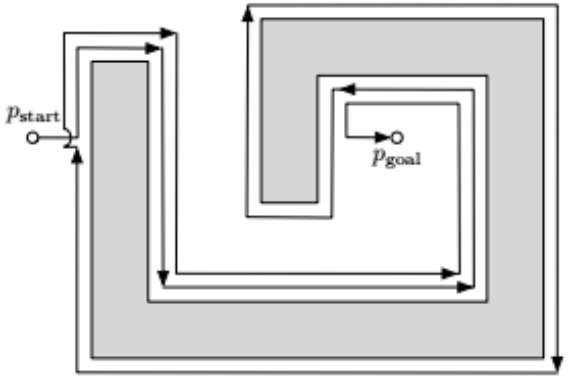
Motion Planning: Classical Approaches

Sensor-based Planning: the Bug Algorithms, could they work well?

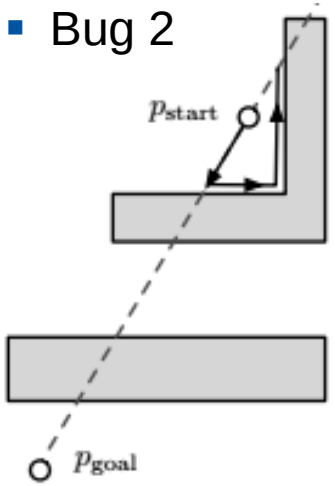
- Bug 0

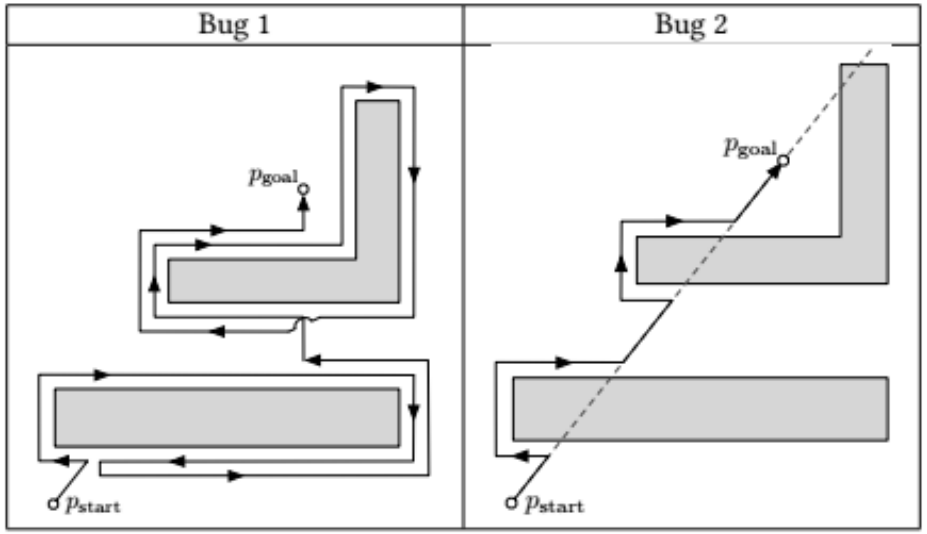


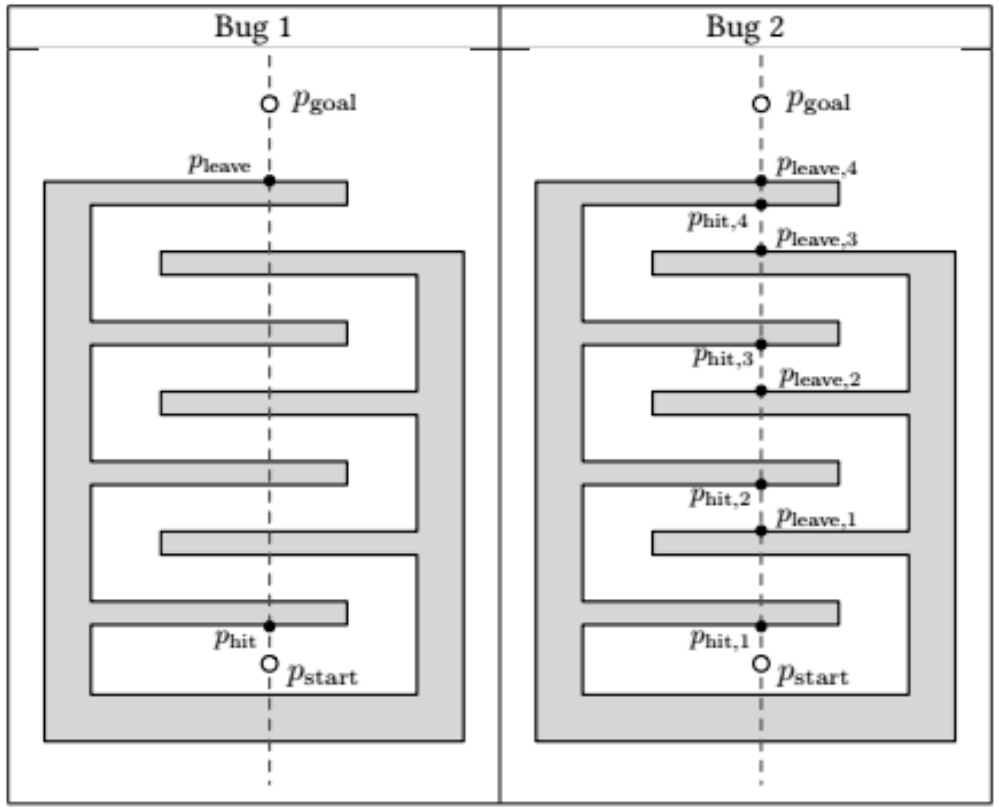
- Bug 1



- Bug 2





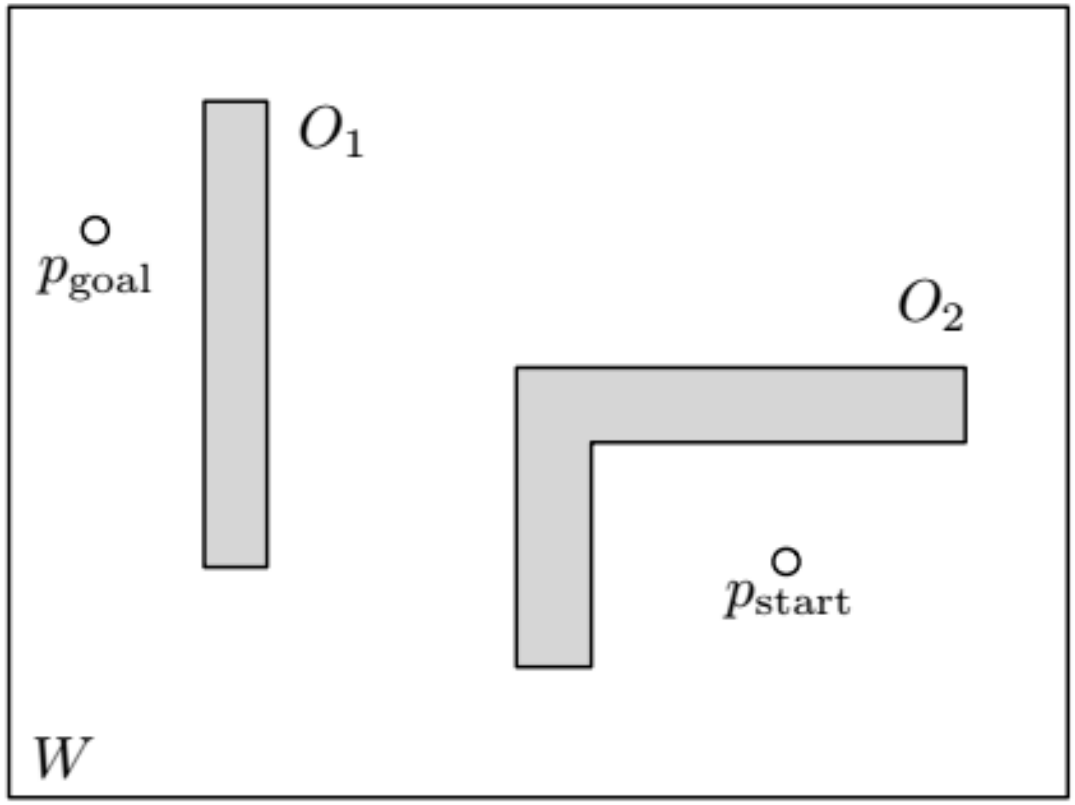


Motion Planning: Classical Approaches

Potential-based Planning

- Problem setup:
  - A workspace $\mathcal{W}$ of $\mathbb{R}^2$ or $\mathbb{R}^3$
  - Some obstacles $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$
  - A state position q_{init} and a goal position q_{goal}
  - A robot described by a moving point (no size)

- Assumptions
  - Robot Assumptions:
    - Knows the direction towards the goal and distance to goal
    - Know something about the obstacles (map)
  - Envionment Assumptions:
    - The workspace is bounded
    - Finite number of obstacles
    - Start and goal position in free Workspace $\mathcal{W}_{free}$



Motion Planning: Classical Approaches

Potential-based Planning: Idea

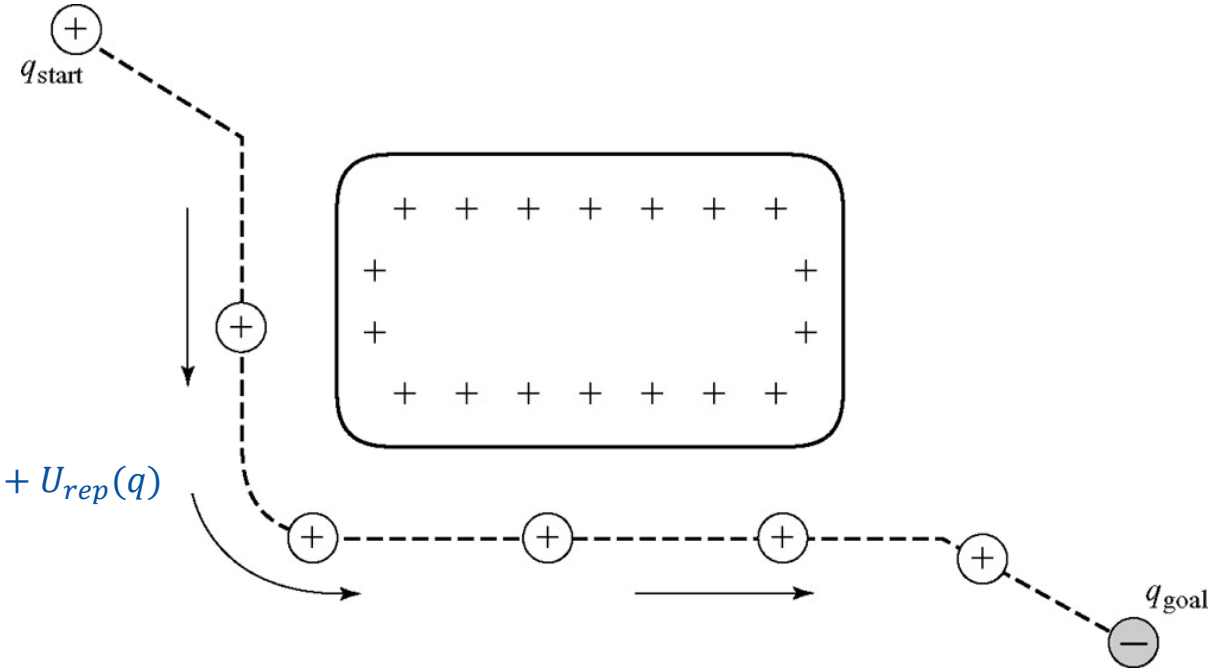
- No explicit representation of the Configuration space
- Potential function:
  - Diffentiable real-valued function $U: \mathbb{R}^m \rightarrow \mathbb{R}$
  - Motion Direction towards to the goal using gradient:

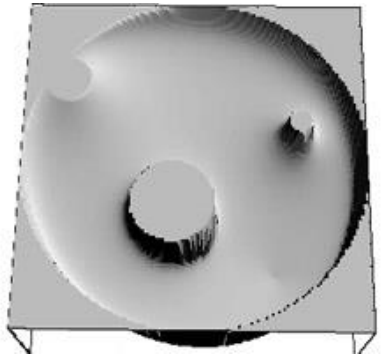
$$\nabla U(q) = DU(q)^T = \left[\frac{\partial U}{\partial q_1}(q), \ldots, \frac{\partial U}{\partial q_m}(q)\right]^T$$

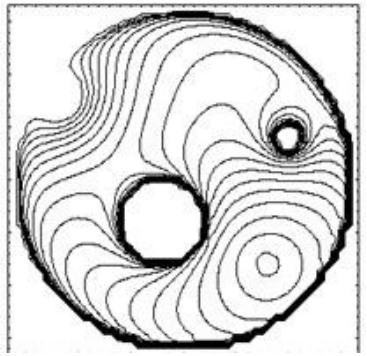
  - Gradient descebt: „downhill“: $\dot{q} = -\nabla U(q)$
  - Attractive / Repulsive potential

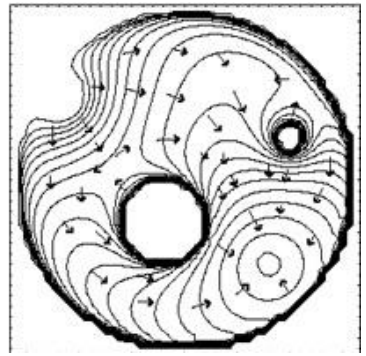
$$U(q) = U_{att}(q) + U_{rep}(q)$$

    - Attractive potential: w.r.t. Goal, e.g. $U_{att} = \frac{1}{2}\xi d^2(q, q_{goal})$
    - Repulsive potential: w.r.t. Obstacles, e.g. $U_{rep} = \frac{1}{2}\eta\left(\frac{1}{D(q)} - \frac{1}{Q^*}\right)^2$



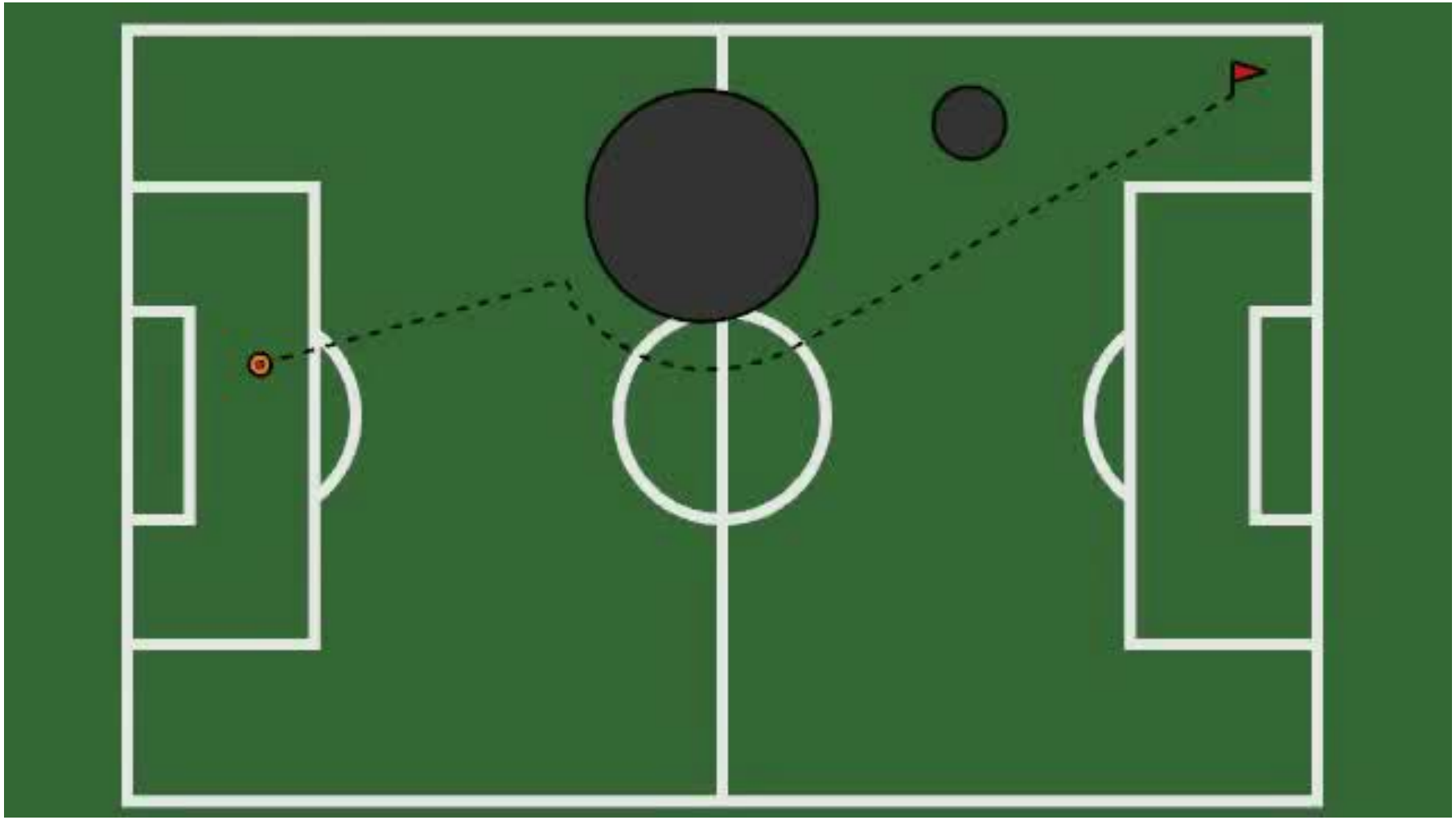






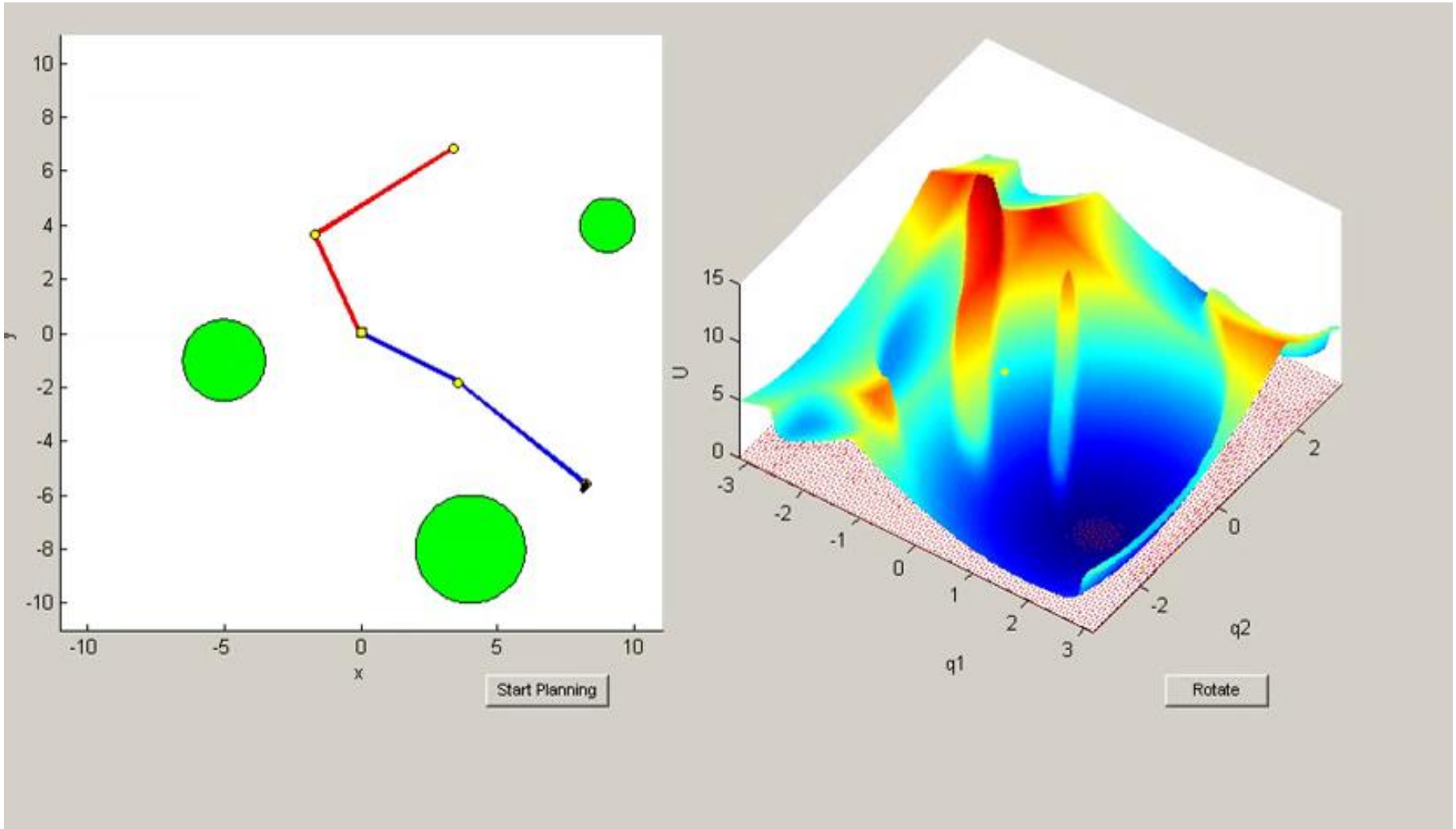
Motion Planning: Classical Approaches

Potential-based Planning: Illustration I



Motion Planning: Classical Approaches

Potential-based Planning: Illustration II

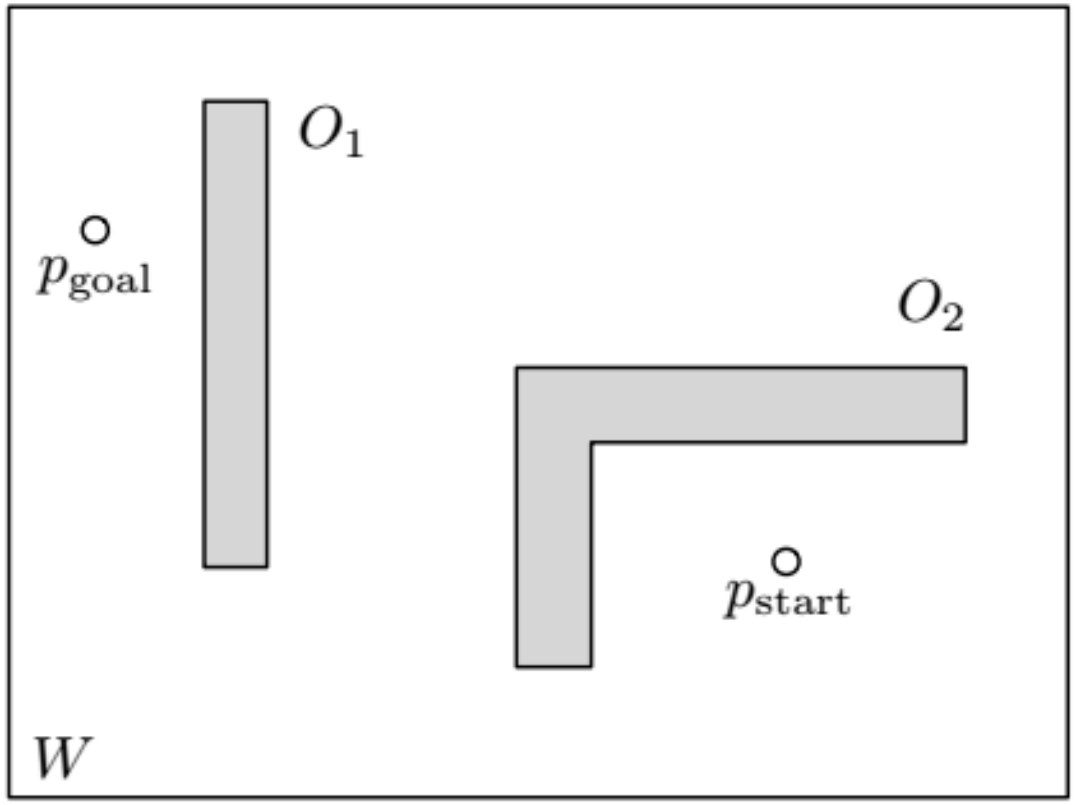


Motion Planning: Classical Approaches

Cell Decompositions

- Problem setup:
  - A workspace $\mathcal{W}$ of $\mathbb{R}^2$ or $\mathbb{R}^3$
  - Some obstacles $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$
  - A state position q_{init} and a goal position q_{goal}
  - A robot described by a moving point (no size)

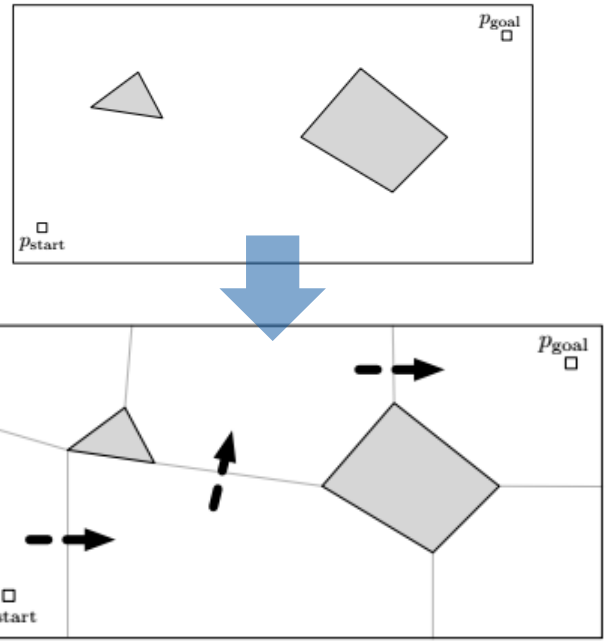
- Assumptions
  - Robot Assumptions:
    - Knows the direction towards the goal and distance to goal
    - Know the workspace and obstacles (map)
  - Envionment Assumptions:
    - The workspace is bounded
    - Finite number of obstacles
    - Start and goal position in free Workspace $\mathcal{W}_{free}$



Motion Planning: Classical Approaches

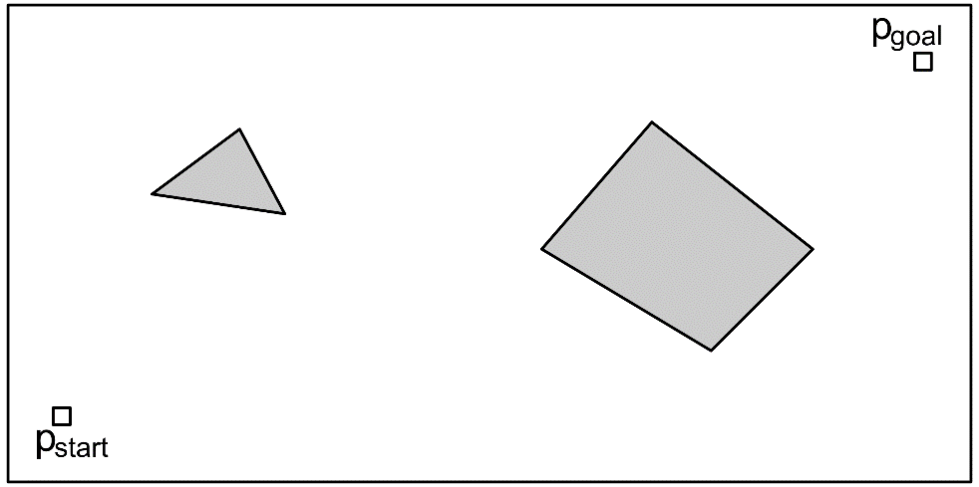
Cell Decompositions: Idea

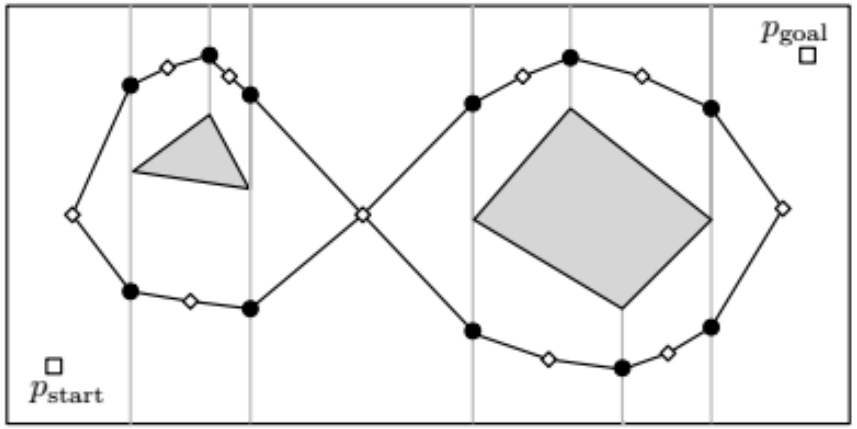
- Decompose the free space into simple cells
- Represent the connectivity of the free space by the adjacency graph of cells
- Decomposition into e.g. Triangles and **Trapezoids**



Algorithm: The roadmap-from-decomposition algorithm

Input: the trapezoidation of a polygon
Output: a roadmap
    label the center of each trapezoid with the symbol ◊
    label the midpoint of each vertical separating segment with the symbol •
    for each trapezoid:
        connect the center to all the midpoints in the trapezoid
    return the roadmap consisting of centers and connections between them through midpoints

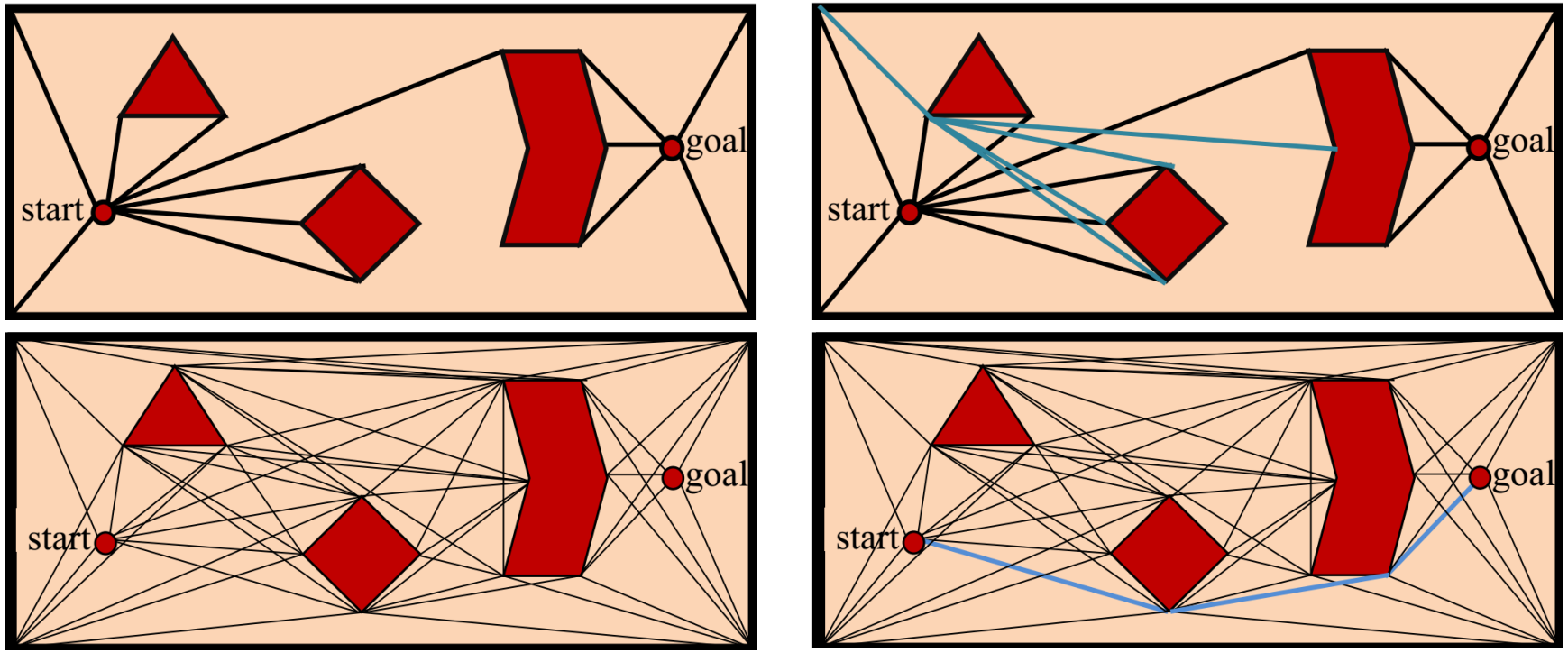




Motion Planning: Classical Approaches

Roadmaps

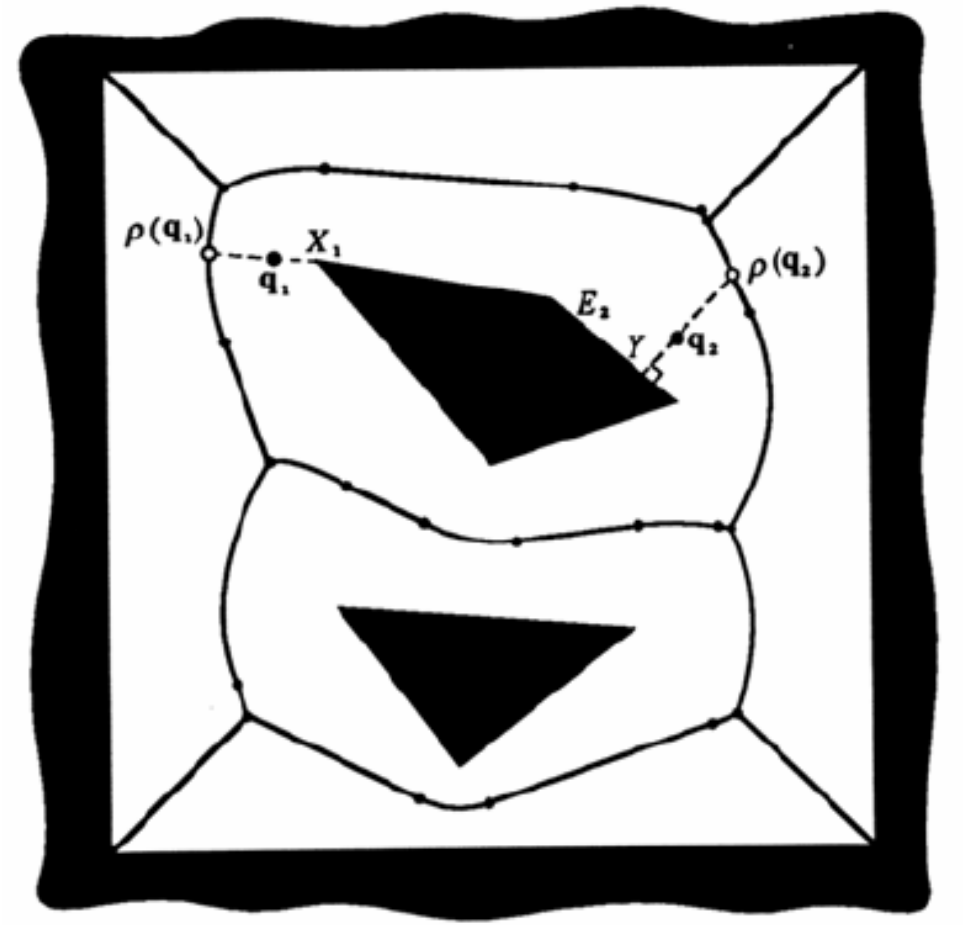
- Map: a data structure, topological, geometric, and grids
- Represent the connectivity of a free space by a network of 1-D curves
- Many types: *visibility maps*, *deformation retracts*, *retract-like structures*, *piecewise retracts* and *silhouettes*
- Visibility Maps:

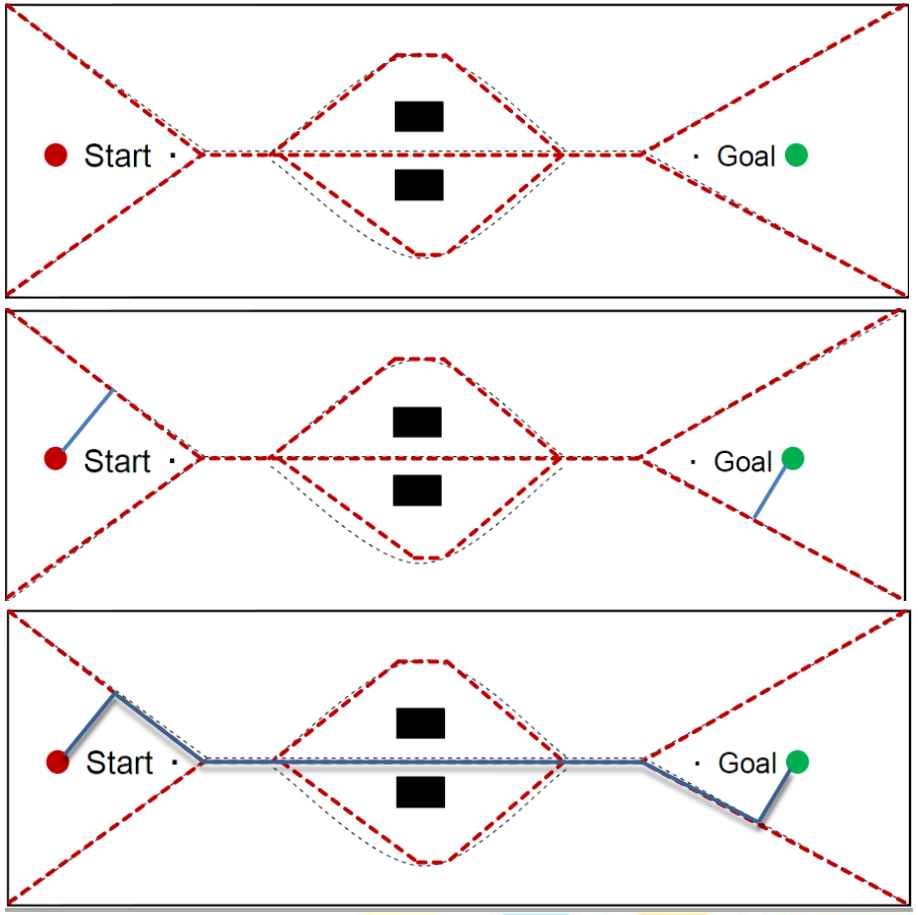


Motion Planning: Classical Approaches

Roadmaps

- Deformation reacts: **Generalized Voronoi Diagram (GVD)**
  The set of points where the distance to the two closest obstacles is the same.





Outline

Motion Planning: Sampling-Based Planning

- Ideal: Build a **complete motion planner** (complete configuration space)
- Complete motion planner: always returns a solution when one exists and indicates that no such solution exists otherwise.
- Problem:
  - Challenges of High Dimensions
  - Motion Planning is **PSPACE-hard**
  - Complexity is **exponential** in the dimension of the robot's configuration space
- Practical Algorithmus:
  - Theoretical: algorithmus strive for completeness and minimal worst-case complexity
    Difficult to implement and not robust
  - Heuristic: algorithms strive for efficiency in commonly encountered situations
    - Trade off completeness for practical efficiency
    - Weaker performance guarantee
- Ways to simplify problem:
  - Project search to lower-dimensional space
  - Limit the number of possibilities
    - Add constraints
    - Reduce "volume" of free space
  - Sacrifice optimality and completeness

Motion Planning: Sampling-Based Planning

The Rise of Monte Carlo Techniques

- Key Idea
  - Rather than exhaustively explore ALL possibilities, randomly explore a smaller subset of possibilities while keeping track of progress
  - Facilities “probing” deeper in a search tree much earlier than any exhaustive algorithm can

- What’s the catch?
  - Typically we must sacrifice both *completeness* and *optimality*
  - Classic tradeoff between solution quality and runtime performance

➢ **Sampling-based Motion Planning**

Motion Planning: Sampling-Based Planning

Concept

- Efficiently handling of planning problems
- Good for systems with many Degrees of Freedom (DoF)
- Idea:
  - Generate random sample points in configuration space
  - Connect start and goal state by connecting samples via collision free paths

➢ it is not necessary to reason about the whole configuration space (only about a finite number of sample configurations)

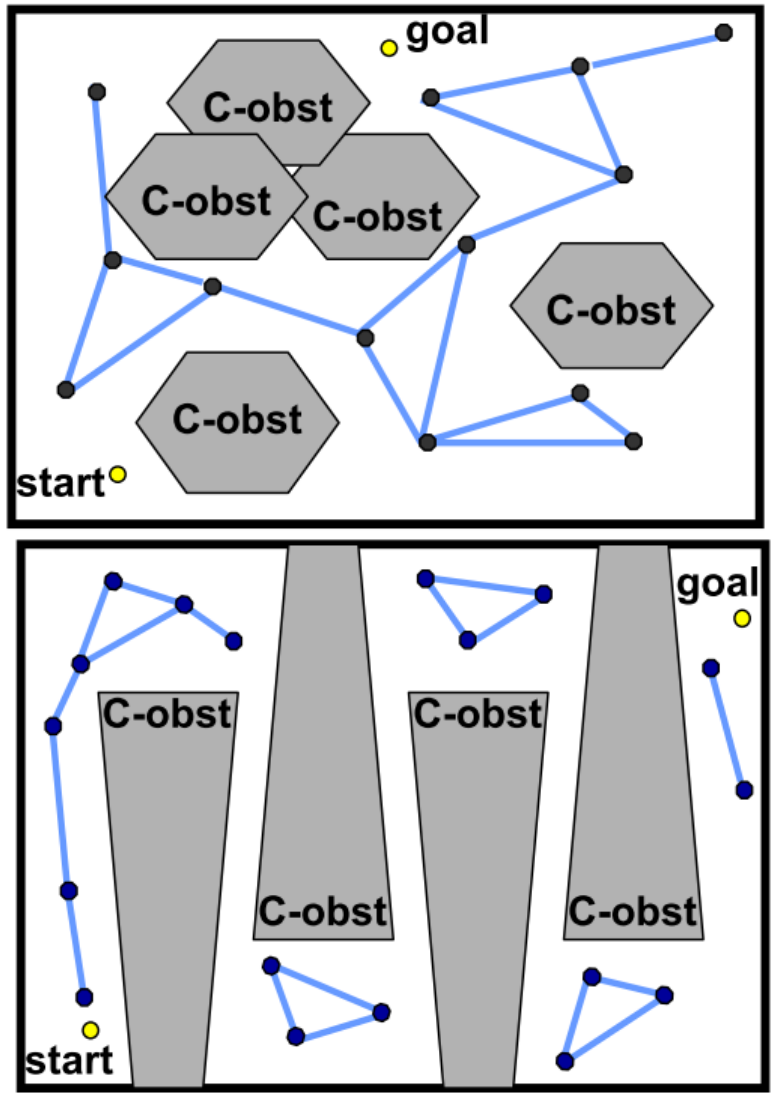
Motion Planning: Sampling-Based Planning

Good news, but bad news too

- Sample-based: **The Good News**
  - *probabilsitically complete*
  - Do not construct the C-space
  - Apply easily to high dimensional C-space
  - Support fast quieries with enough preprocessing

  ➢ Many success stories where Sampling-based Motion Planning solve previously unsolved problems

- Sample-based: **The Bad News**
  1. don’t work as well for some problems
     - unlikely to sample nodes in *narrow passages*
     - Hard to sample/connect nodes on constraint surfaces
  2. No optimality or completeness

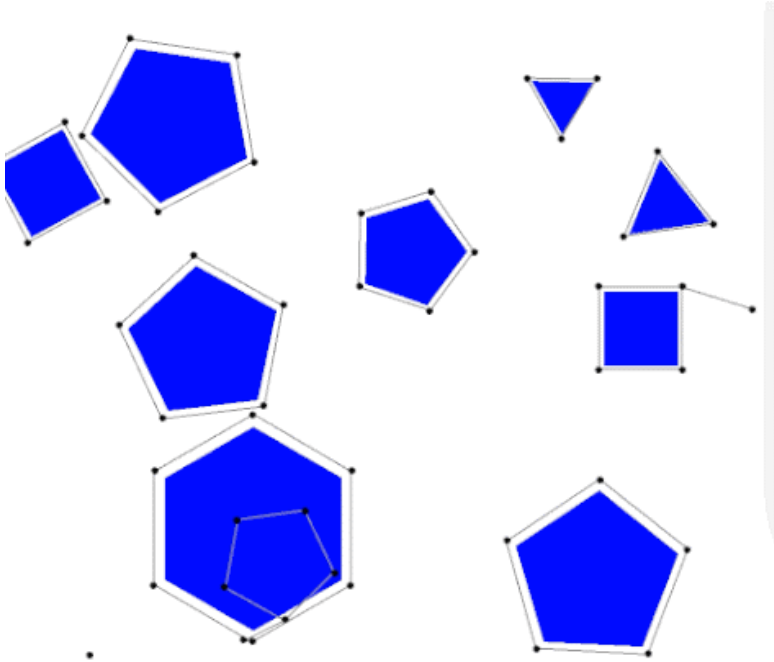


Motion Planning: Sampling-Based Planning

Methods

- Probabilistic Roadmaps (PRM)
  - Uses sampled states to create a **roadmap** of the free state space
  - Each sample point is connected to an amount of nearby samples via collision free paths
  - Collision free paths between samples are determined by a local planner (i.e. linear interpolation)
  - Learning Phase
    - Construction Step:

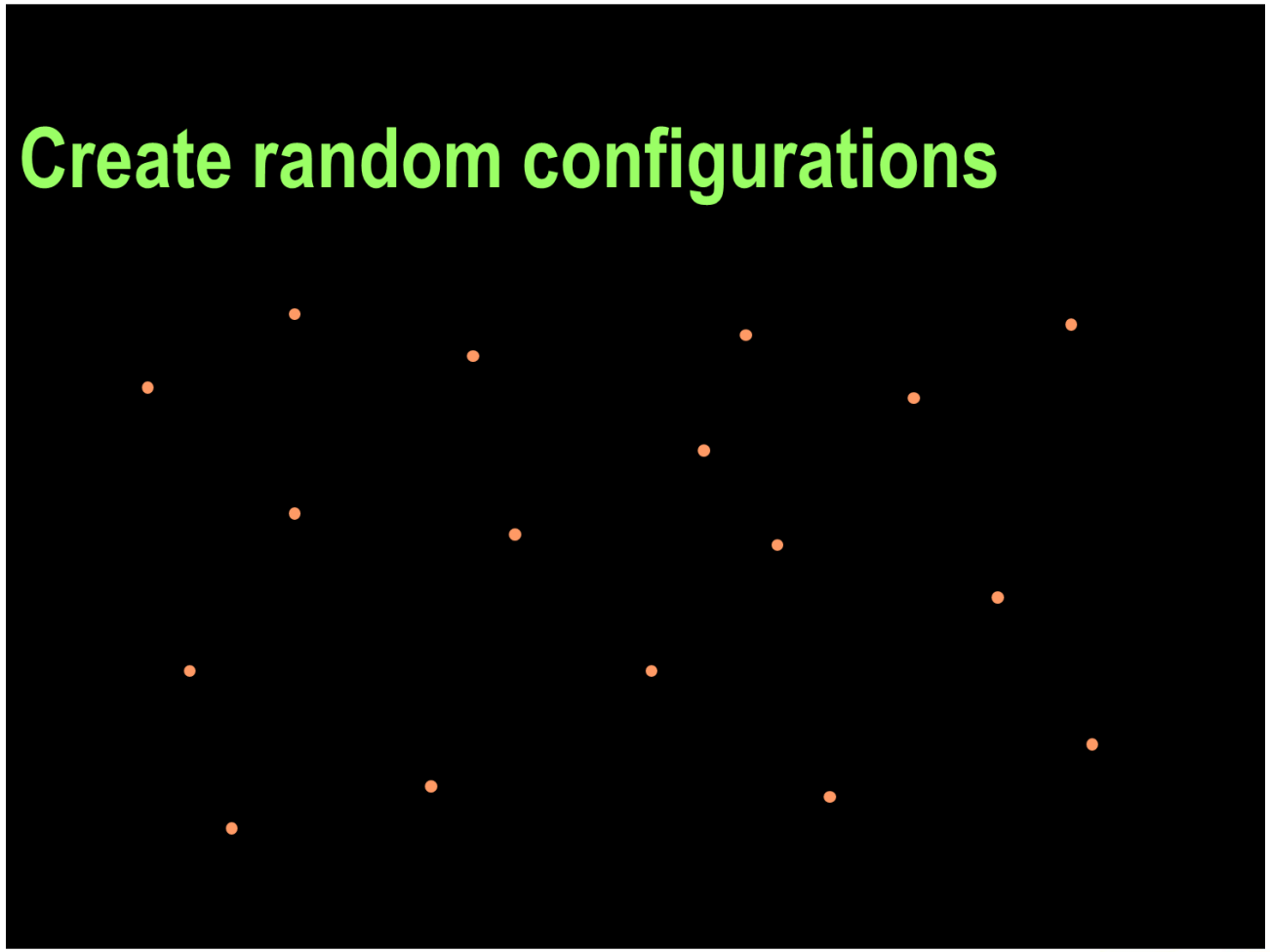
      Construct a probabilistic roadmap by generating random free configurations of the robot and Connecting them using a local planner
    - Expansion Step: Supplement and refine the Roadmap
  - Query Phase: find a path



Motion Planning: Sampling-Based Planning

Methods

- Probabilistic Roadmaps (PRM)

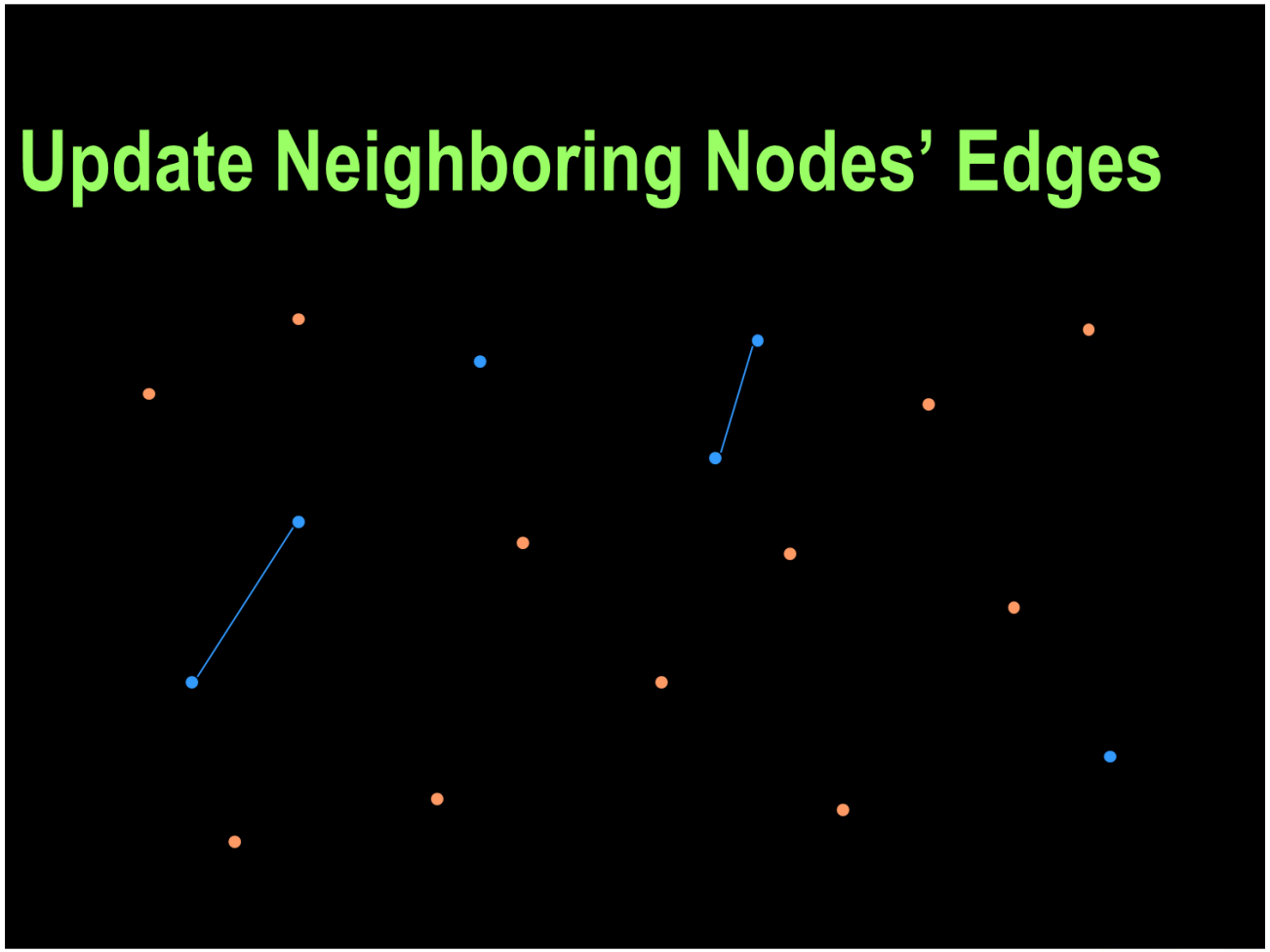


Slides from Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, and a lot from James Kuffner

Motion Planning: Sampling-Based Planning

Methods

- Probabilistic Roadmaps (PRM)

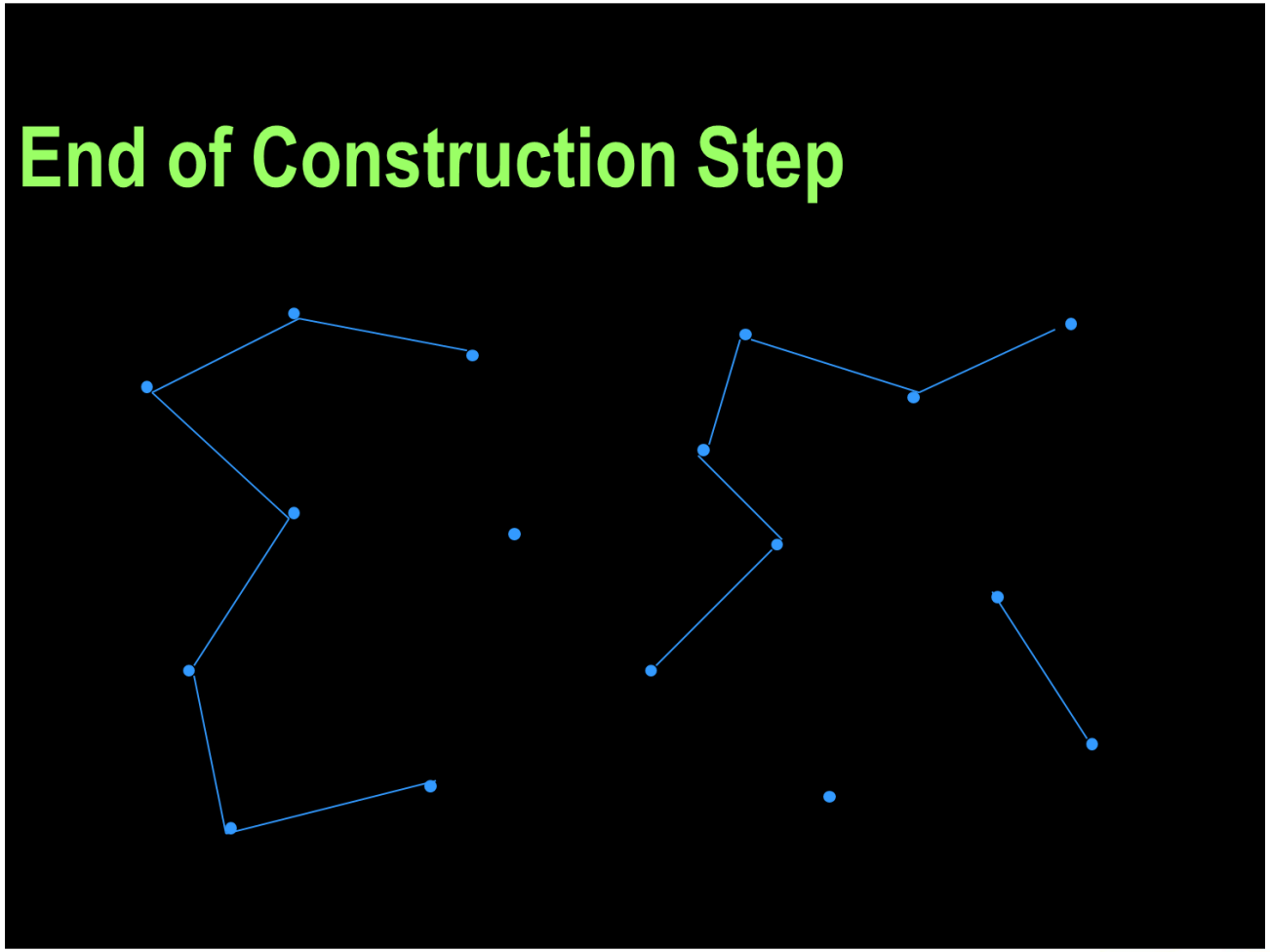


Slides from Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, and a lot from James Kuffner

Motion Planning: Sampling-Based Planning

Methods

- Probabilistic Roadmaps (PRM)

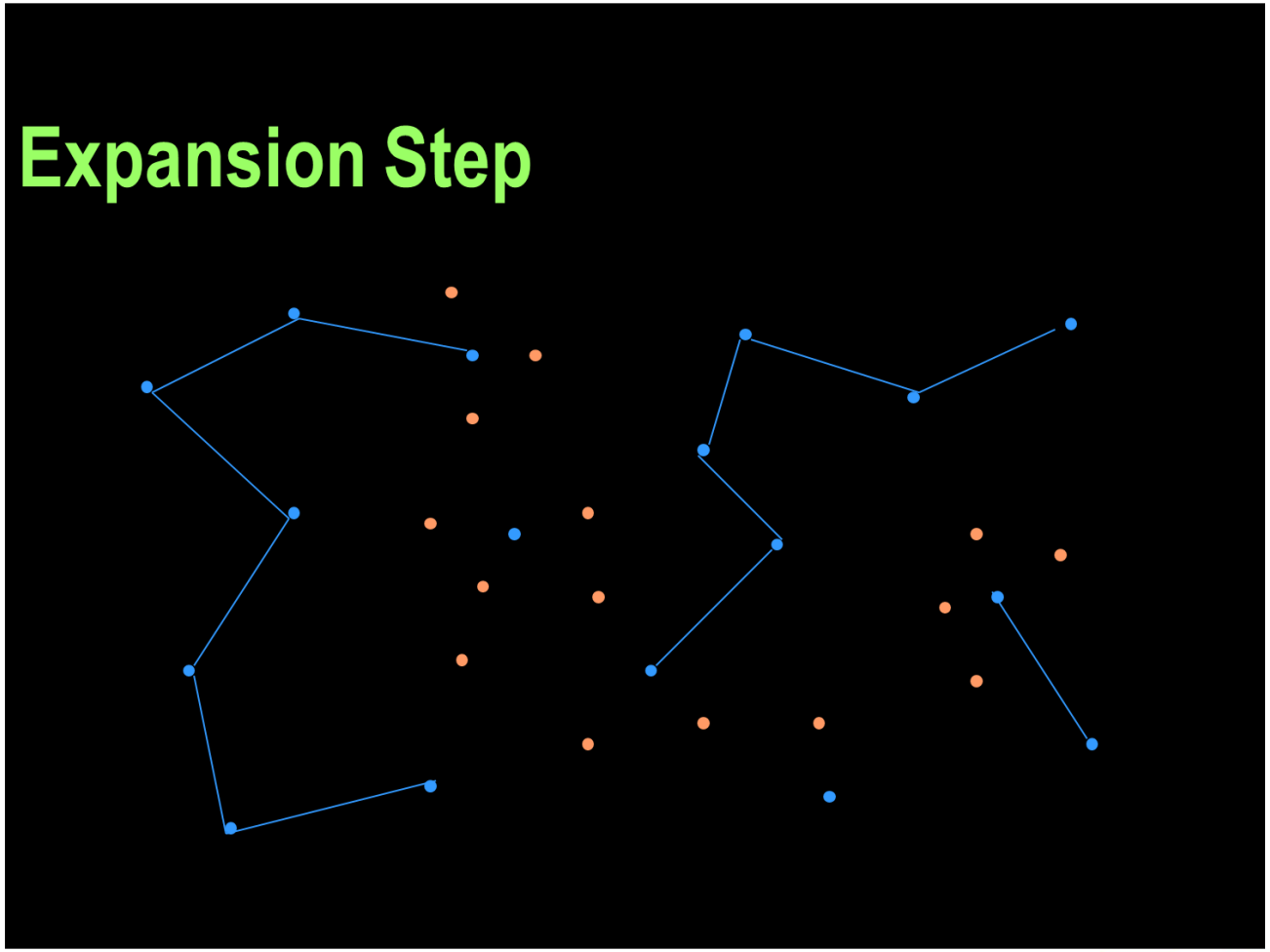


Slides from Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, and a lot from James Kuffner

Motion Planning: Sampling-Based Planning

Methods

- Probabilistic Roadmaps (PRM)

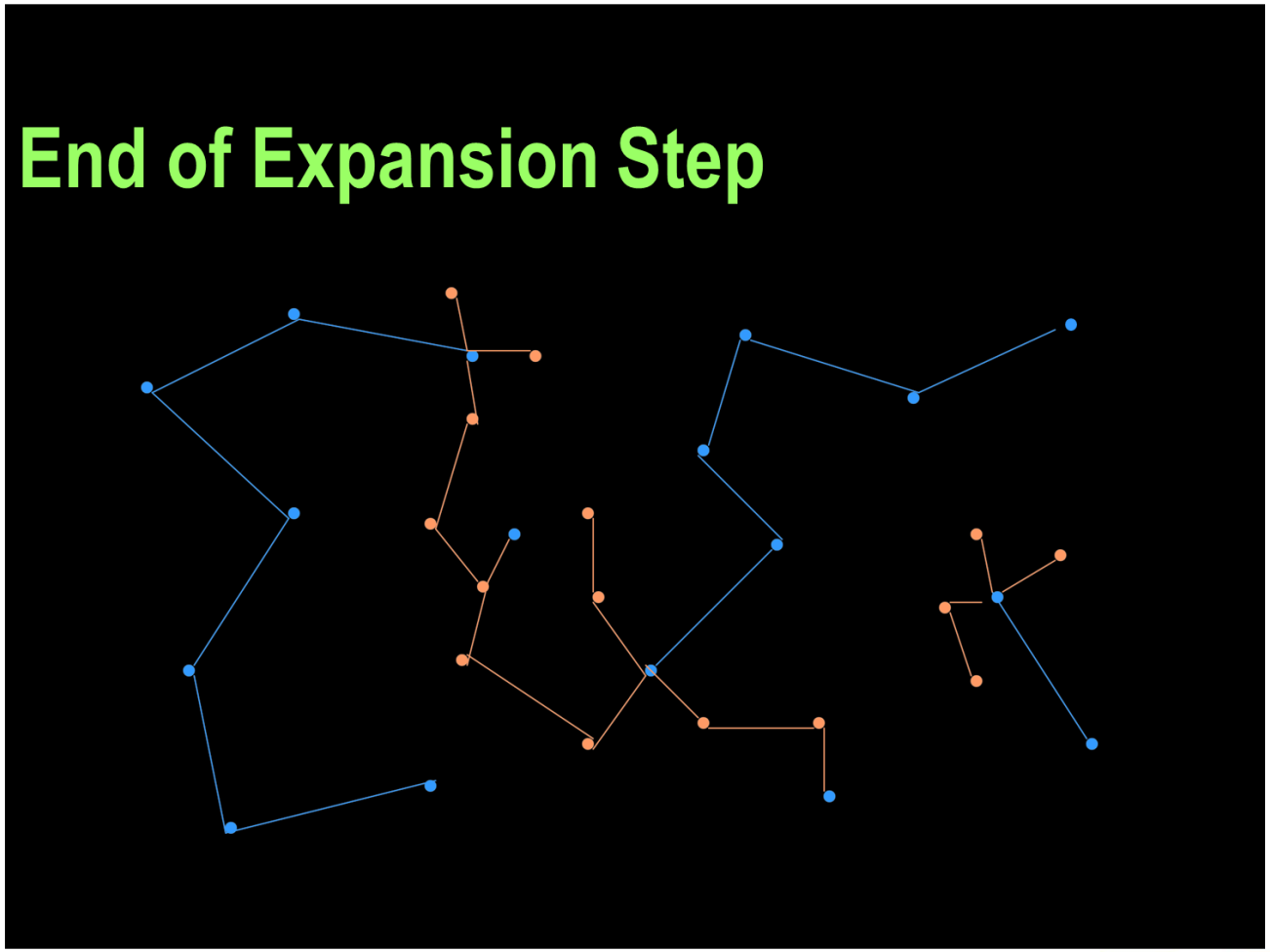


Slides from Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, and a lot from James Kuffner

Motion Planning: Sampling-Based Planning

Methods

- Probabilistic Roadmaps (PRM)

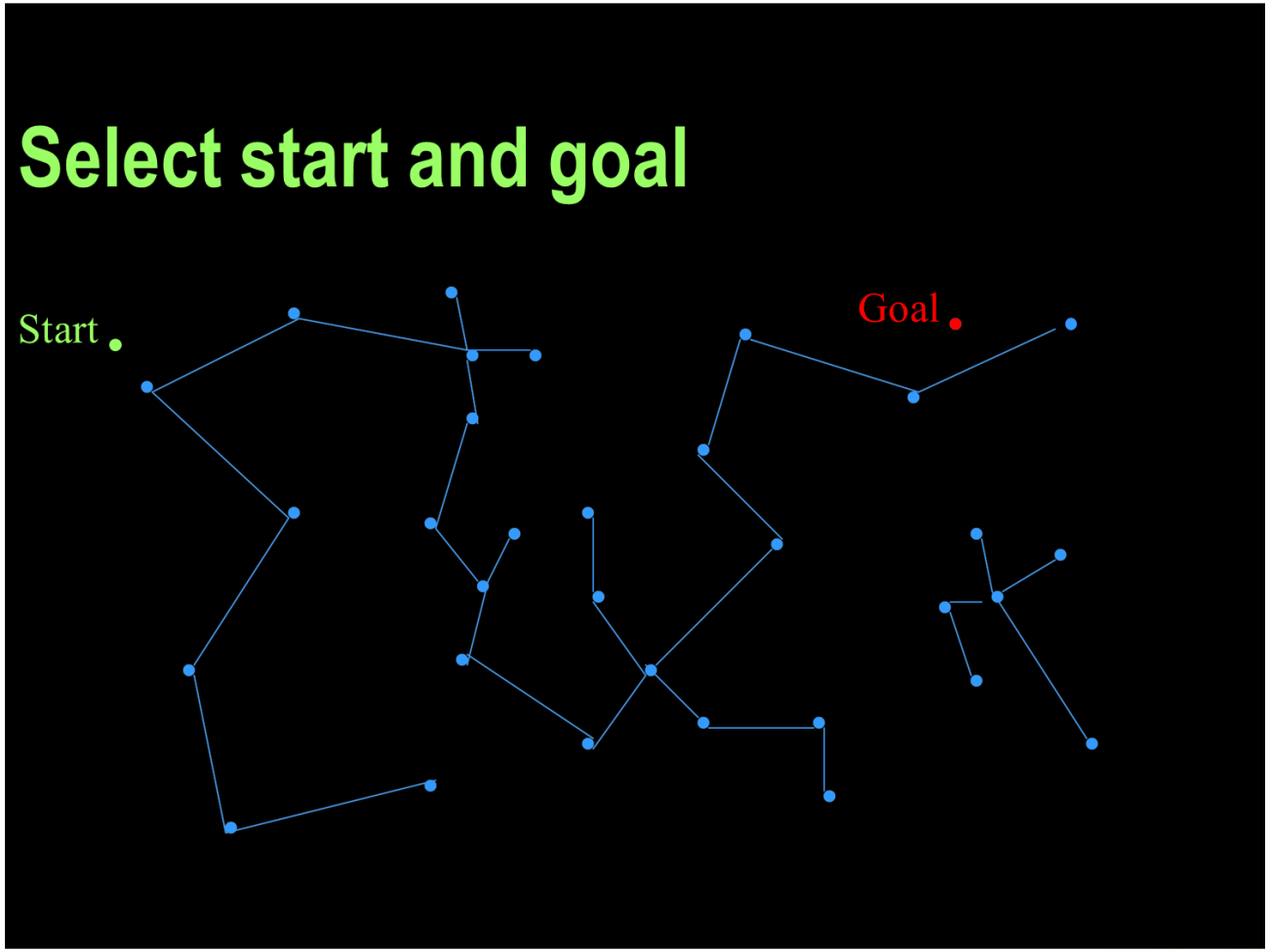


Slides from Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, and a lot from James Kuffner

Motion Planning: Sampling-Based Planning

Methods

- Probabilistic Roadmaps (PRM)



Slides from Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, and a lot from James Kuffner

Motion Planning: Sampling-Based Planning

Methods

- Probabilistic Roadmaps (PRM)

Slides from Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, and a lot from James Kuffner

Motion Planning: Sampling-Based Planning

Methods

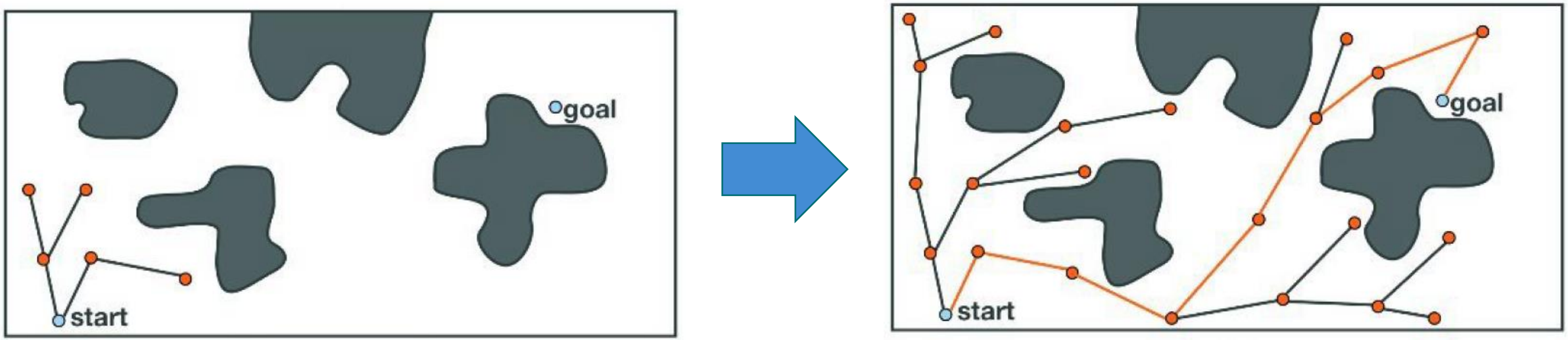
- Probabilistic Roadmaps (PRM)

Slides from Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, and a lot from James Kuffner

Motion Planning: Sampling-Based Planning

Methods

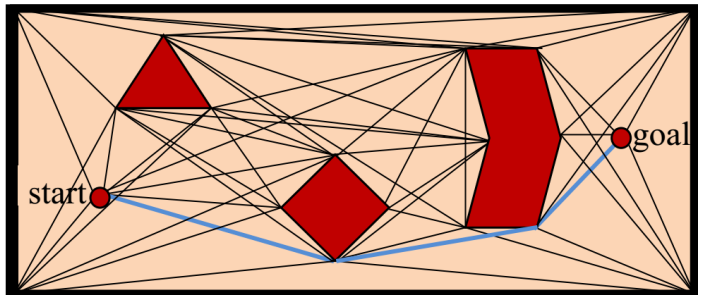
- Tree-based Planners
  - Uses sampled states to create a **tree** of the free state space
  - Root of the tree is the start state (i.e. A)
  - Based on this, the tree is expanded towards the goal state by creating collision free connections between samples

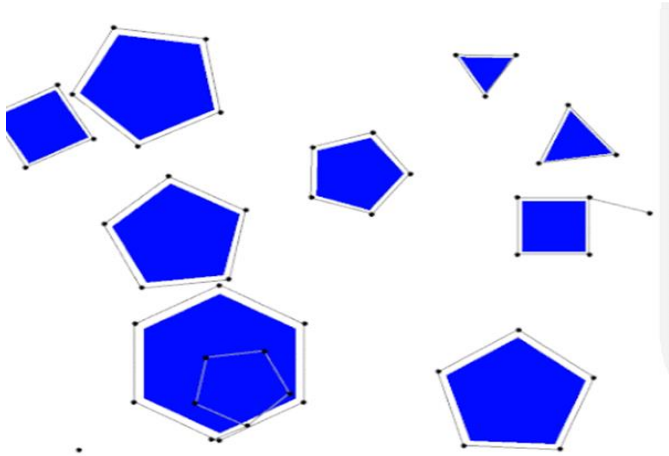


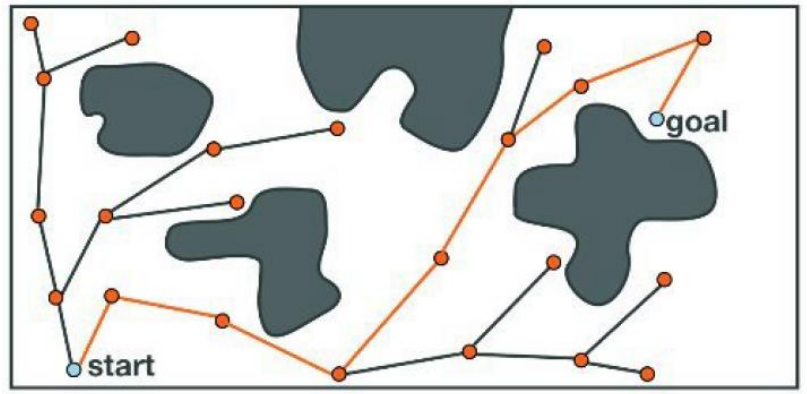
Motion Planning: Sampling-Based Planning

Compare

- Complete Motion Planning
  - Always terminate
  - Not efficient
  - Not robust even for low DoF
- Probabilistic Roadmap
  - Construction and query phase
  - Can be reused for subsequent queries
  - Efficient
  - Work for complex problems with many DoFs
  - Difficult for narrow passages
  - May not terminate when no path exists
- **Tree-based Planners**
  - Good for *single query planning*
  - Does not cover the whole free state space
  - Many up to date methods are tree-based planners



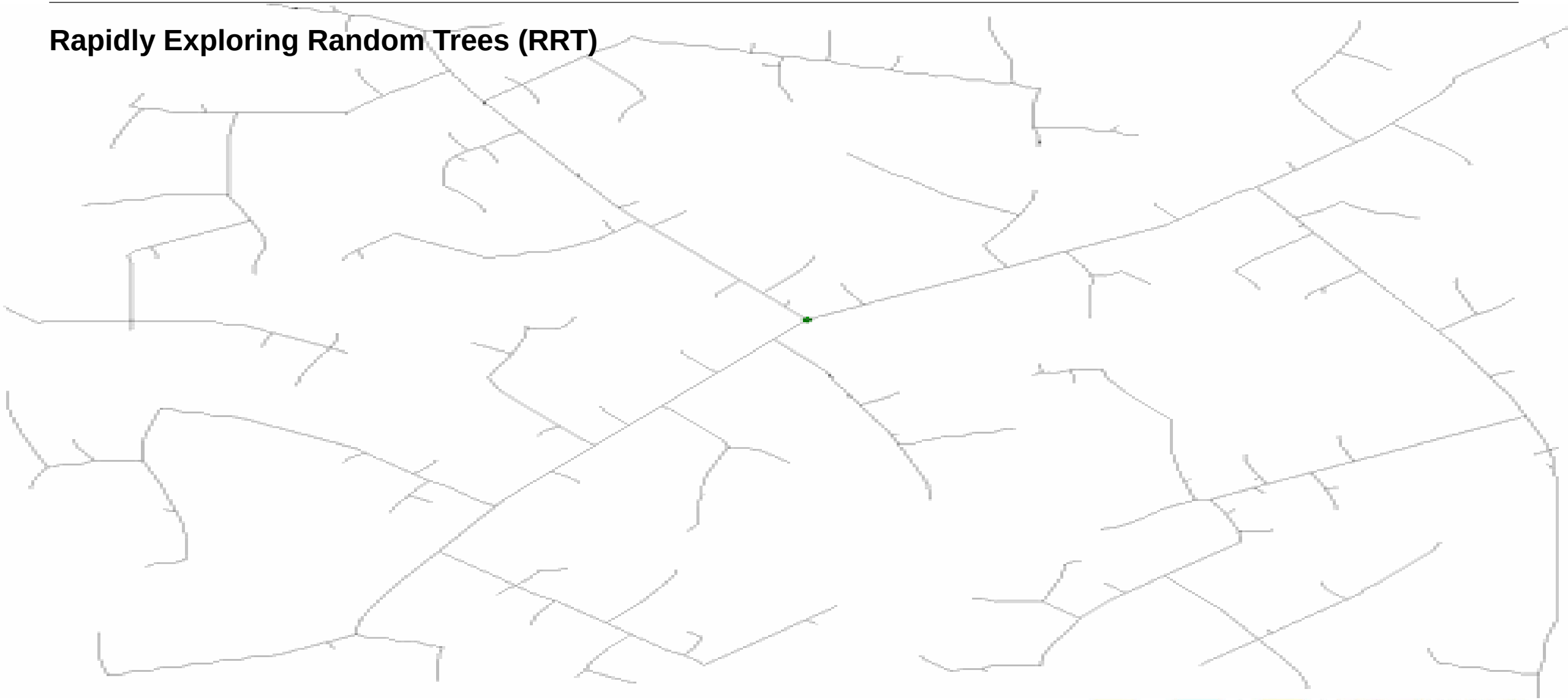




Outline

Motion Planning: Tree-based Planners

Rapidly Exploring Random Trees (RRT)



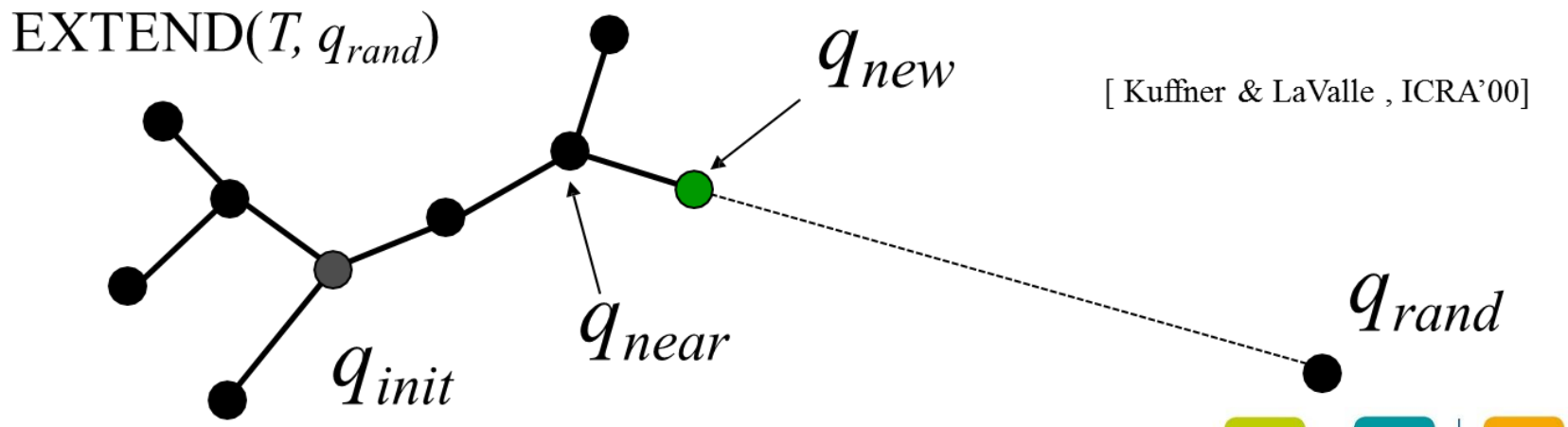
Motion Planning: Tree-based Planners

Rapidly Exploring Random Trees (RRT)

Algorithm: Build RRT

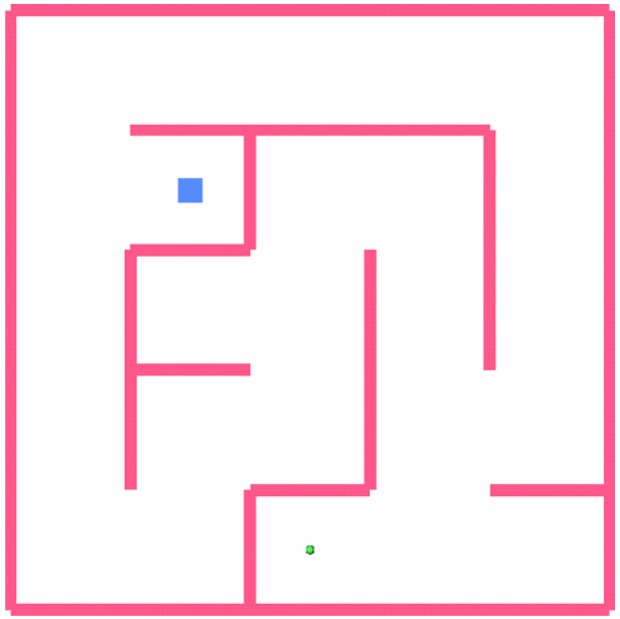
Input: Initial configuration q_{init}, number of vertices in RRT K, incremental distance Δq
Output: RRT tree T

Initialize tree T with initial configuration q_{init}
for k=1 to K:
    $q_{rand} \leftarrow$ random point in configuration space
    $q_{near} \leftarrow$ nearest vertex in tree T of q_{rand}
    $q_{new} \leftarrow$ new config by moving an incremental dist Δq from q_{near} in the direction of q_{rand}
    Add vertex q_{new} to tree T
    Add edge (q_{near}, q_{new}) to tree T

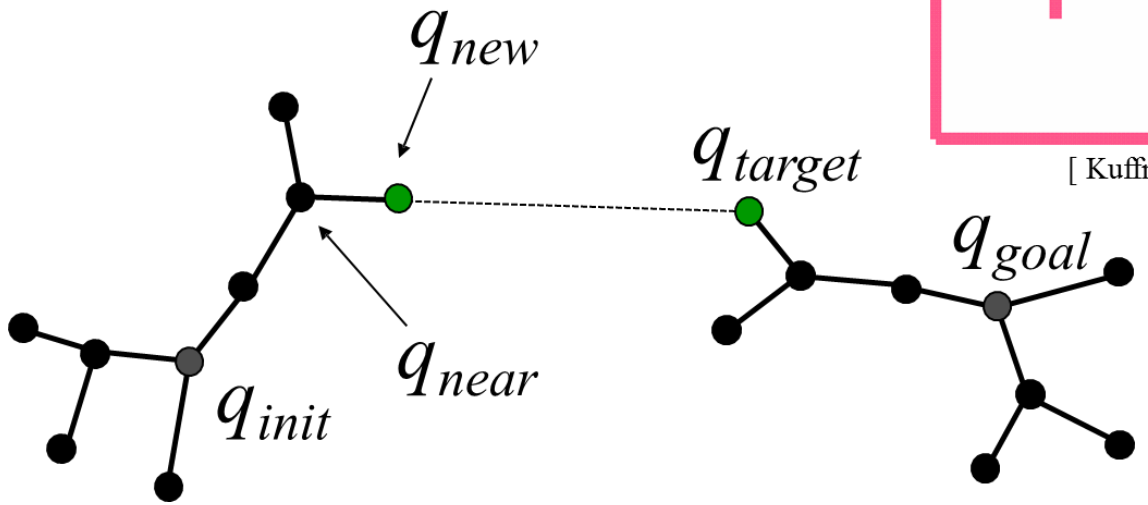


Motion Planning: Tree-based Planners

- Grow two RRTs towards each other



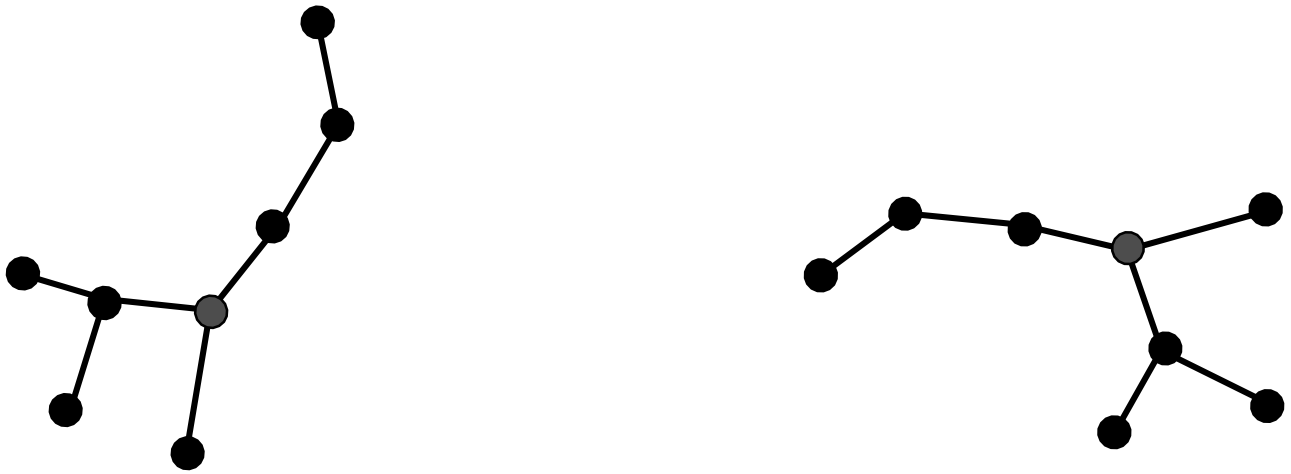
[Kuffner, LaValle ICRA '00]



Motion Planning: Tree-based Planners

- Grow two RRTs towards each other

A single RRT-Connect iteration..

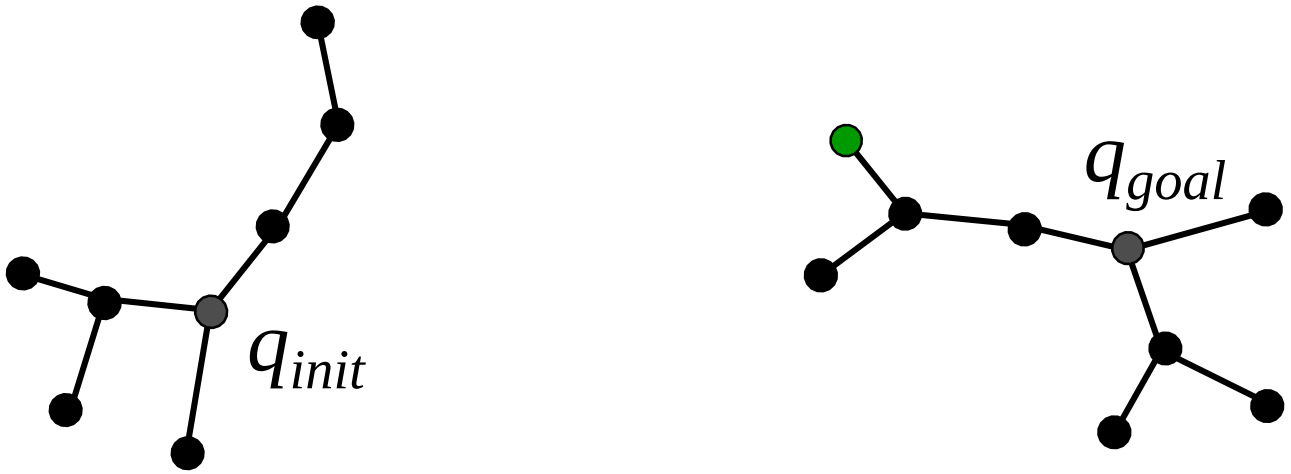


[Kuffner & LaValle , ICRA'00]

Motion Planning: Tree-based Planners

- Grow two RRTs towards each other

1) One tree grown using random target

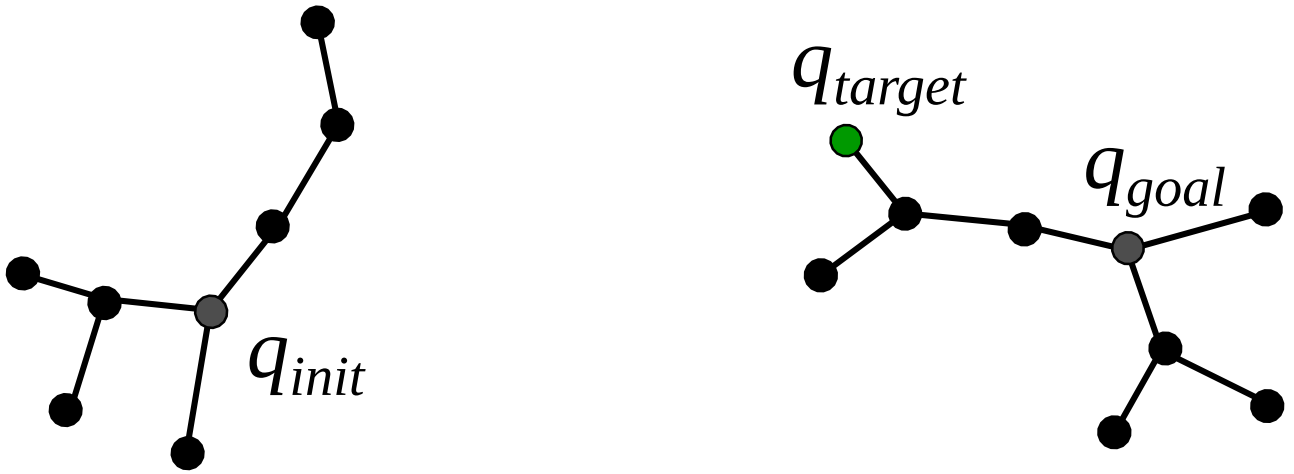


[Kuffner & LaValle , ICRA'00]

Motion Planning: Tree-based Planners

- Grow two RRTs towards each other

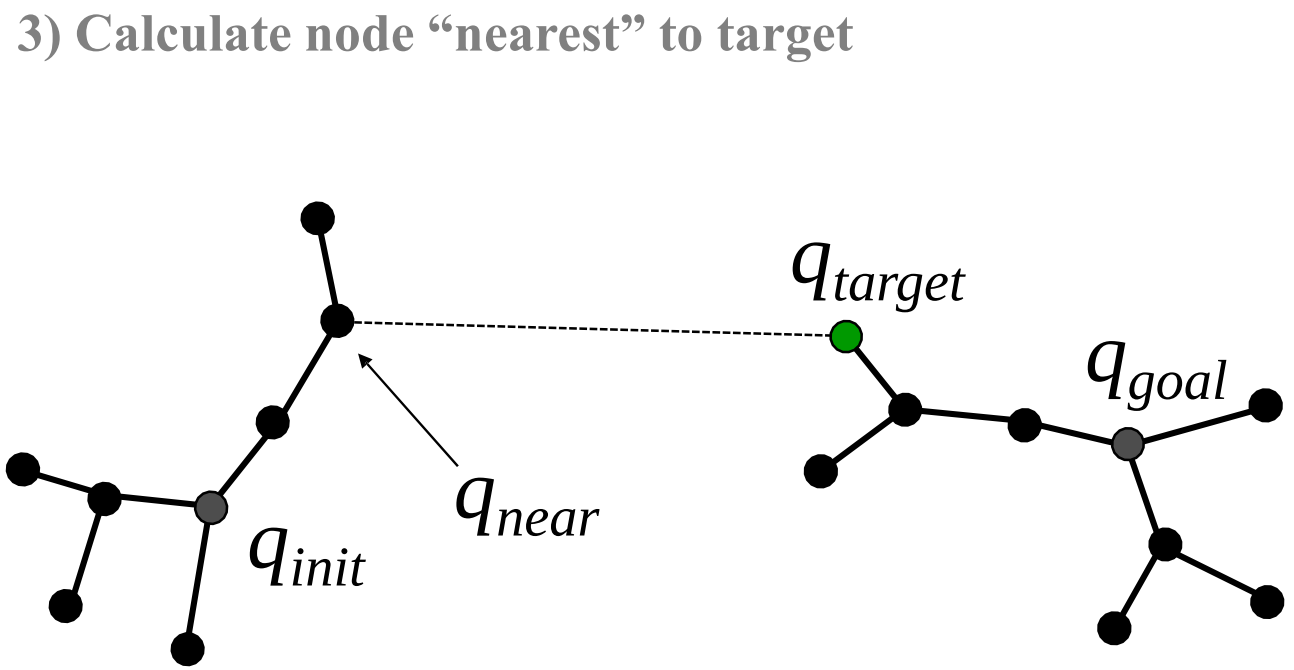
2) New node becomes target for other tree



[Kuffner & LaValle , ICRA'00]

Motion Planning: Tree-based Planners

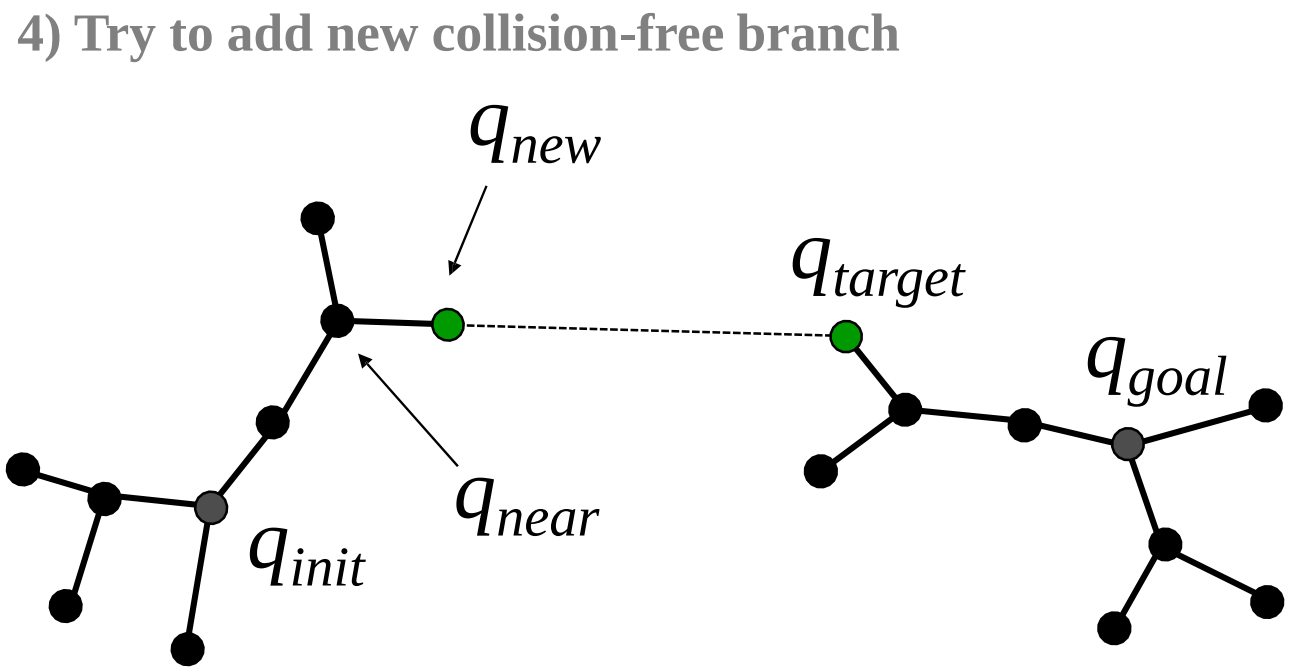
- Grow two RRTs towards each other



[Kuffner & LaValle , ICRA’00]

Motion Planning: Tree-based Planners

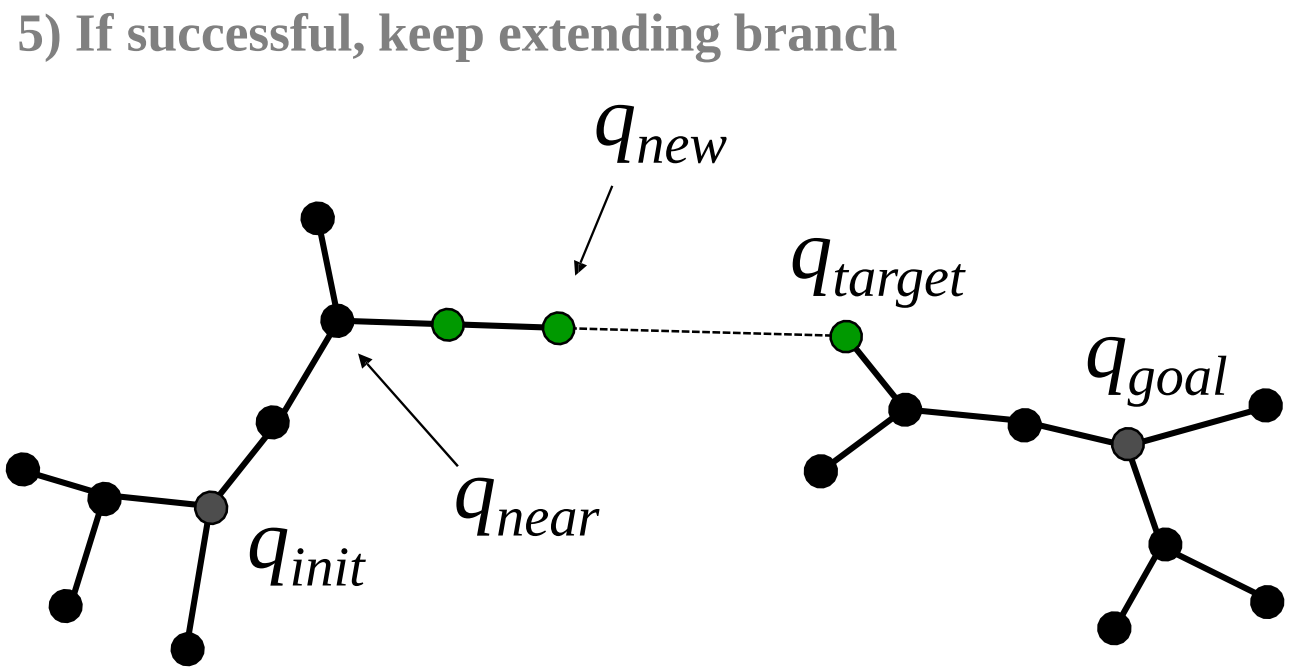
- Grow two RRTs towards each other



[Kuffner & LaValle , ICRA'00]

Motion Planning: Tree-based Planners

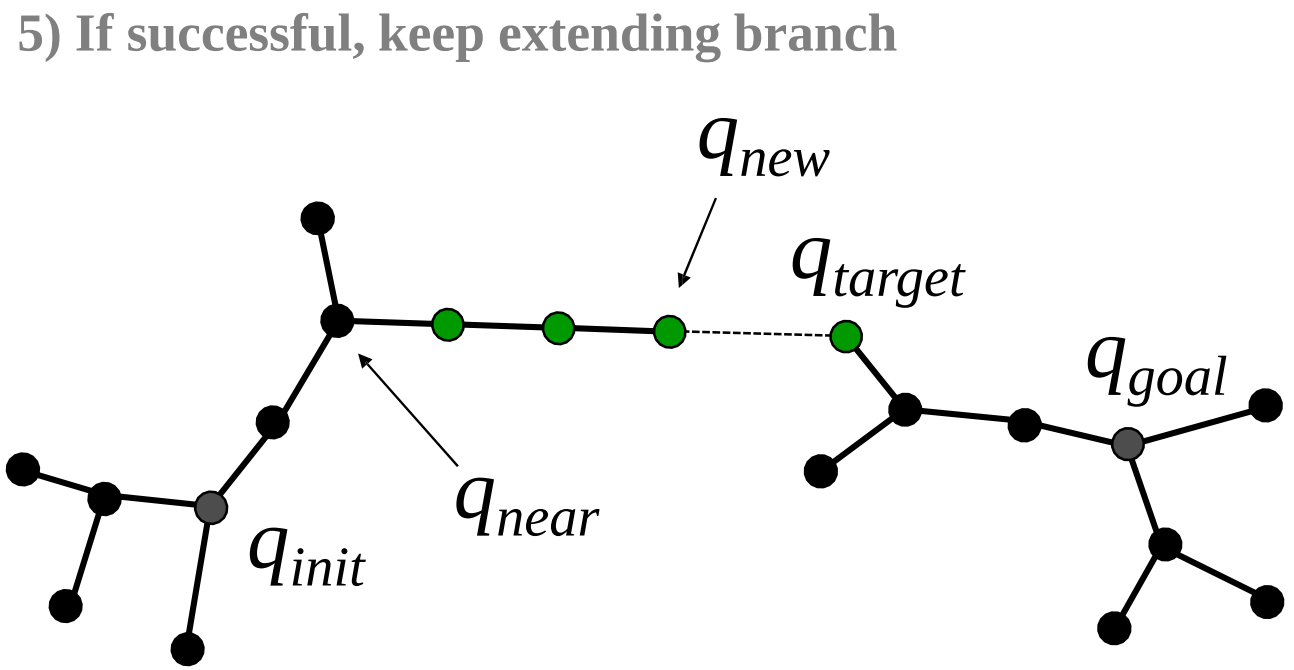
- Grow two RRTs towards each other



[Kuffner & LaValle , ICRA'00]

Motion Planning: Tree-based Planners

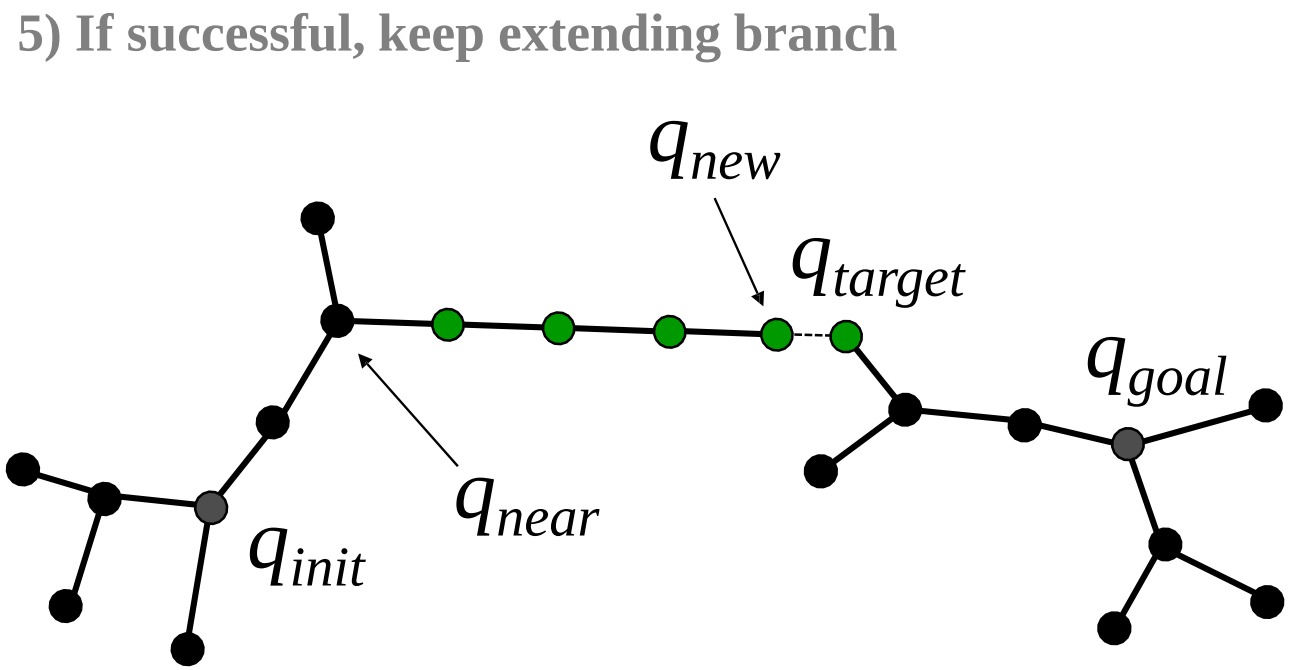
- Grow two RRTs towards each other



[Kuffner & LaValle , ICRA'00]

Motion Planning: Tree-based Planners

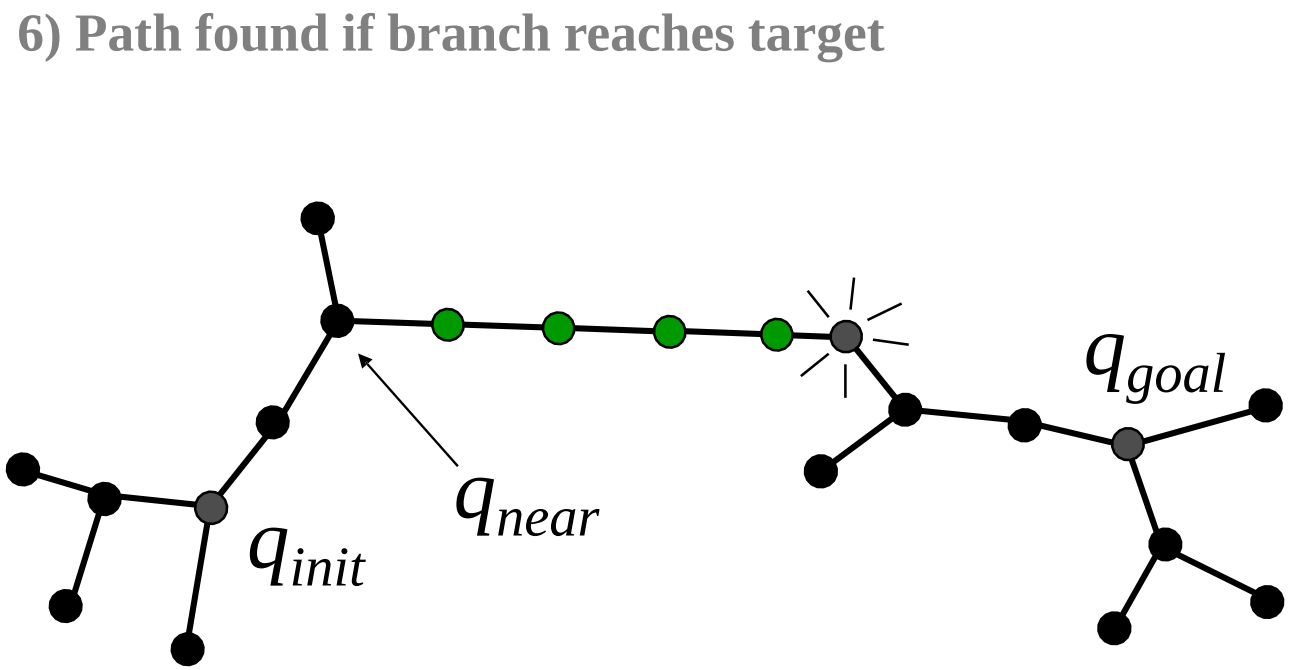
- Grow two RRTs towards each other



[Kuffner & LaValle , ICRA'00]

Motion Planning: Tree-based Planners

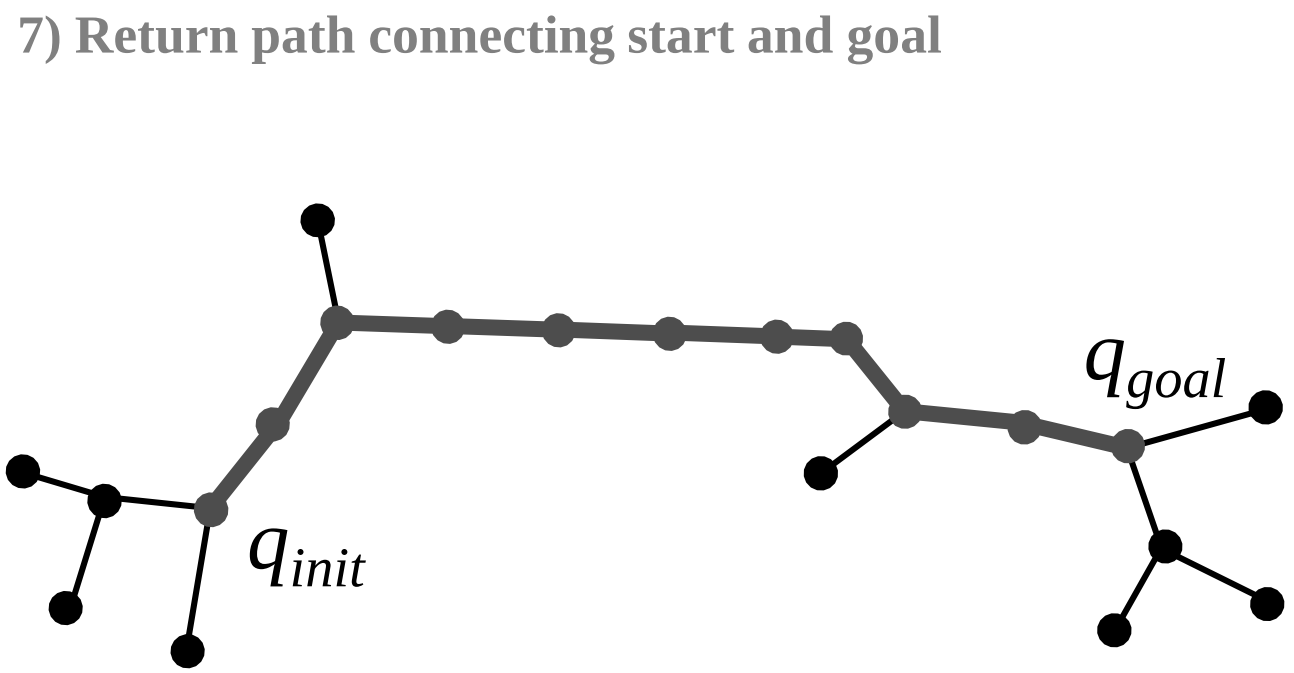
- Grow two RRTs towards each other



[Kuffner & LaValle , ICRA'00]

Motion Planning: Tree-based Planners

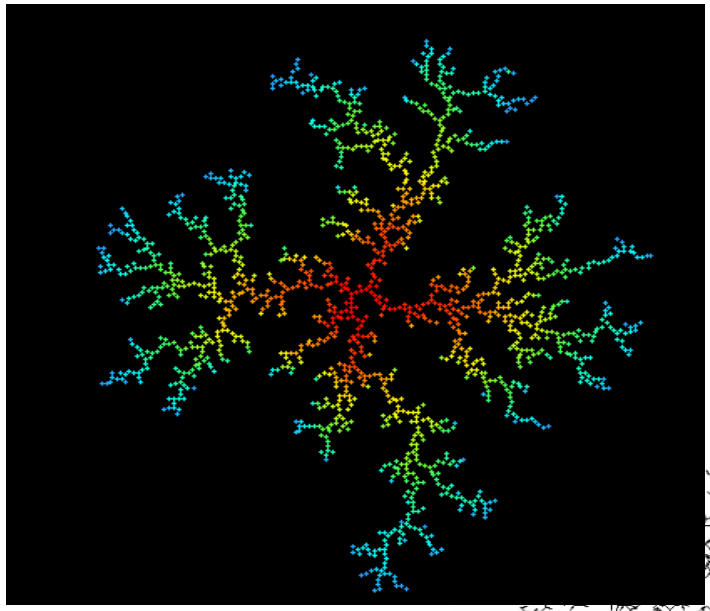
- Grow two RRTs towards each other

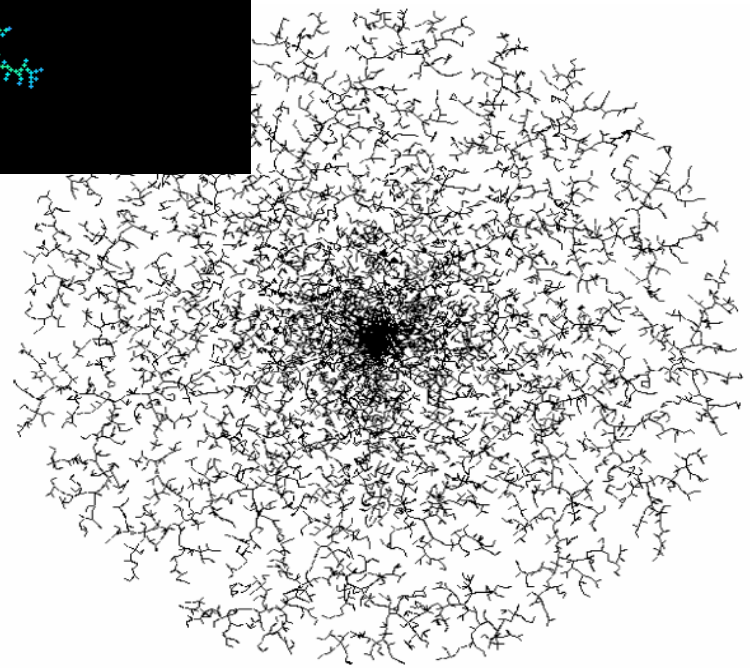


[Kuffner & LaValle , ICRA'00]

Motion Planning: Tree-based Planners

- Applications of RRTs
  - Robotics Applications
    - mobile robotics
    - manipulation
    - Humanoids
  - Other Applications
    - biology (drug design)
    - manufacturing and virtual prototyping (assembly analysis)
    - verification and validation
    - computer animation and real-time graphics aerospace
- RRT Extensions
  - discrete planning (STRIPS and Rubik's cube)
  - real-time RRTs
  - anytime RRTs
  - dynamic domain RRTs
  - deterministic RRTs
  - parallel RRTs
  - hybrid RRTs





Motion Planning: Tree-based Planners

Review of Piano Mover‘s Problem

Balancing Exploration and Exploitation in Sampling-Based Motion Planning

"The piano mover's problem"

Markus Rickert, Arne Sieverling, and Oliver Brock

fortiss GmbH, An-Institut Technische Universität München, München, Germany
Robotics and Biology Laboratory, Technische Universität Berlin, Berlin, Germany

IEEE Transactions on Robotics

Motion Planning: Tree-based Planners

Did we concern the Real World? Not Really!

- Robots in Real World
  - Have inertia (not a point)
  - Have limited controllability (nonholonomic motion)
  - Have limited sensors
  - Face a dynamic environment
  - Face an unreliable environment

Motion Planning: Tree-based Planners

Kinodynamic Motion Planning

- **Kinodynamic Motion Planning**: Takes dynamic constraints into account (i.e. velocity, acceleration, friction, bounded forces)
  - Important for realistic robots
  - Enables motion planning for complex dynamics (e.g complex environment)
- Challenges
  - Dimensionality of the state space is typically higher because dynamics (velocity, acceleration, …) are included in the state space
  - State space may not be entirely reachable from the robot's initial state
  - Difficult to define a meaningful metric for these complexe state spaces

Motion Planning: Tree-based Planners

Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE)

- **KPIECE**:
  - Addresses previously mentioned challenges
  - Uses a physic simulation to create motion samples
- Design goals
  - Ease of use for systems where only forward propagation is available (simulation can be done forward in time)
  - No state sampling and distance metric is required
  - For complex systems, described by physical models (instead of equations of motions)
- Advantages
  - Fast and accurate
  - Is applicable in real-time motion planning

Motion Planning: Tree-based Planners

Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE)

Definition: KPIECE Motion Planning P

A KPIECE motion planning problem is defined as a tuple $\langle Q, U, I, F, f \rangle$

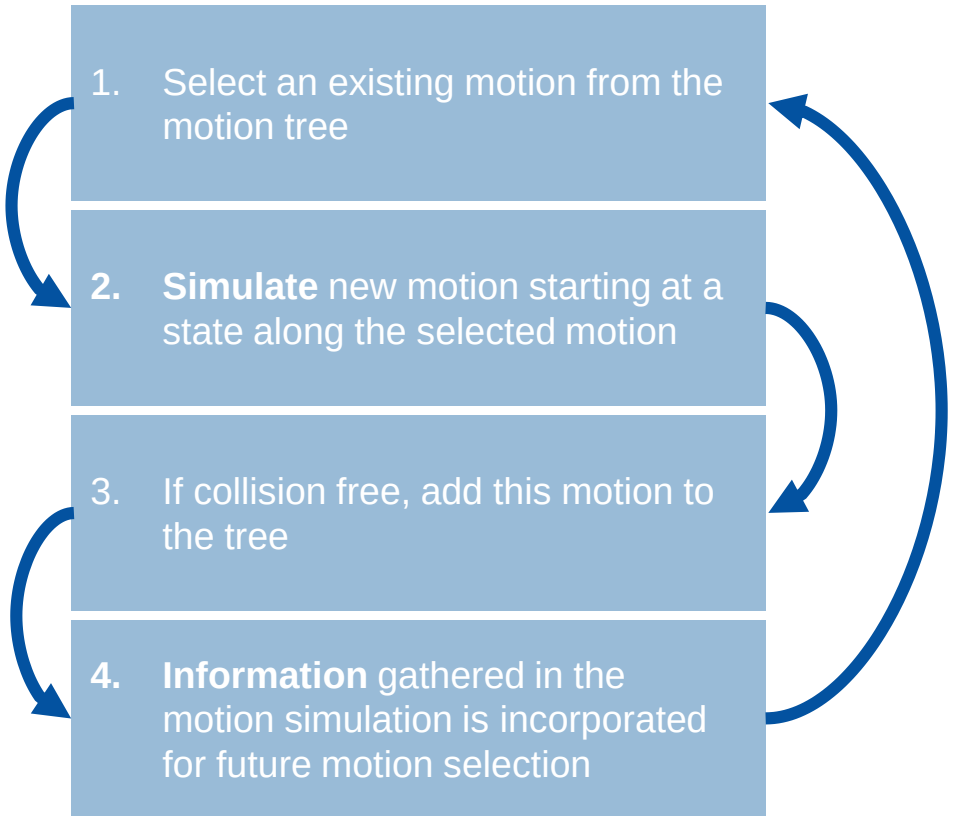
- Q is the state space
- U is the control space
- $I \in Q$ is the set of initial states
- $F \in Q$ is the set of final states
- f is the dynamics (forward propagation routine)

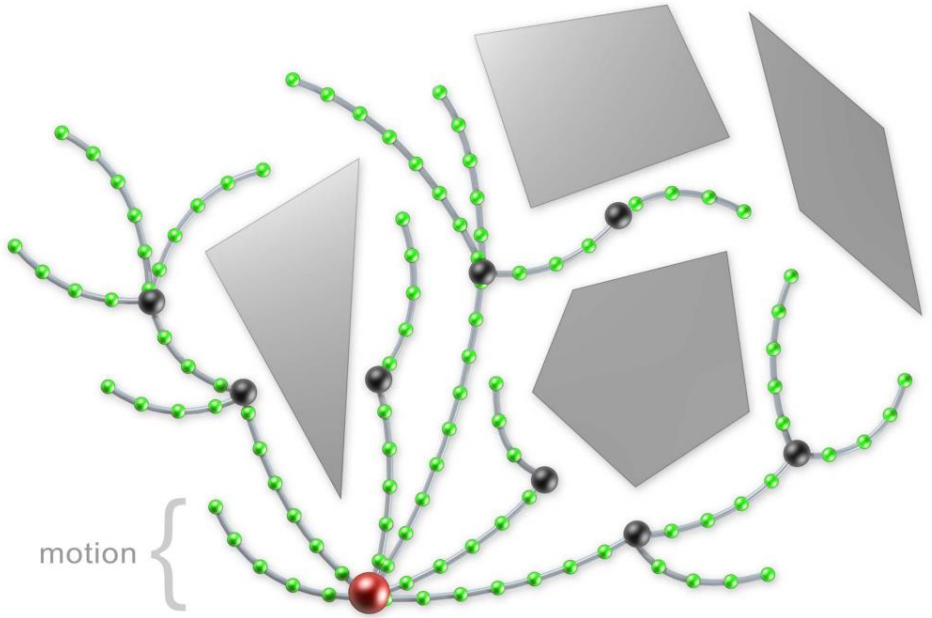
A solution to this motion planning problem consists of a sequence of controls $u_1, \dots, u_n$ and time $t_1, \dots t_n$

Motion Planning: Tree-based Planners

KPIECE: High-Level Description of the algorithm

- Iterative Construction of a tree of motions in state space of the robot.
- Motion $\mu = (s, u, t),\ s \in Q, u \in U, t \in \mathbb{R}^{\geq 0}$





Tree of motions as grown by KPIECE. The states of the start of motions are depicted as larger vertices. The motion is computed by forward integration at fixed step size.

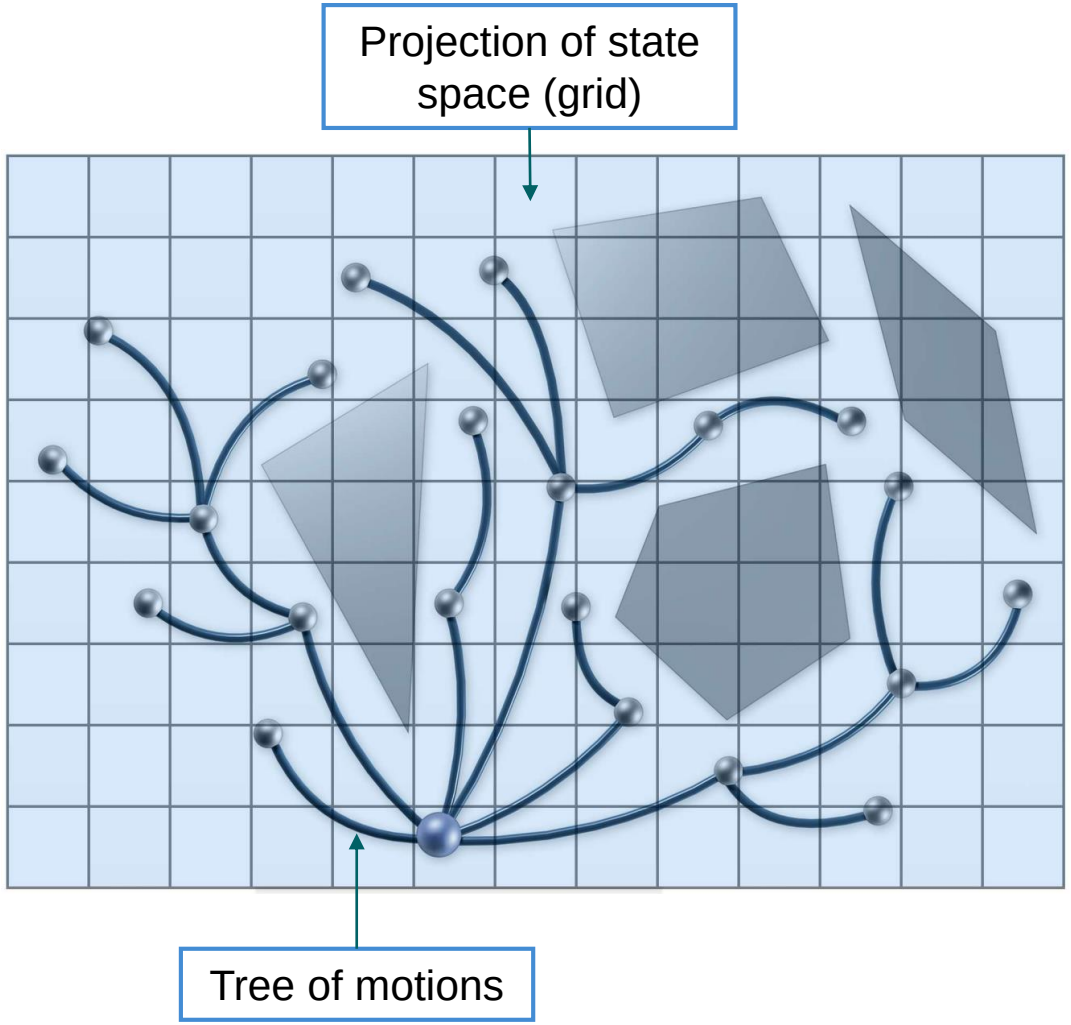
Motion Planning: Tree-based Planners

KPIECE: Estimating State Space Coverage

Which samples to create?

- Key difficulty
  - Avoiding over-exploration of certain regions
  - Avoiding under-exploration of other regions
- **Idea:** Employ a projection of the state space

  Assumption: if the tree of motions covers the projection well, it also covers the state space
- How to define the projection?



Motion Planning: Tree-based Planners

KPIECE: State Space Coverage

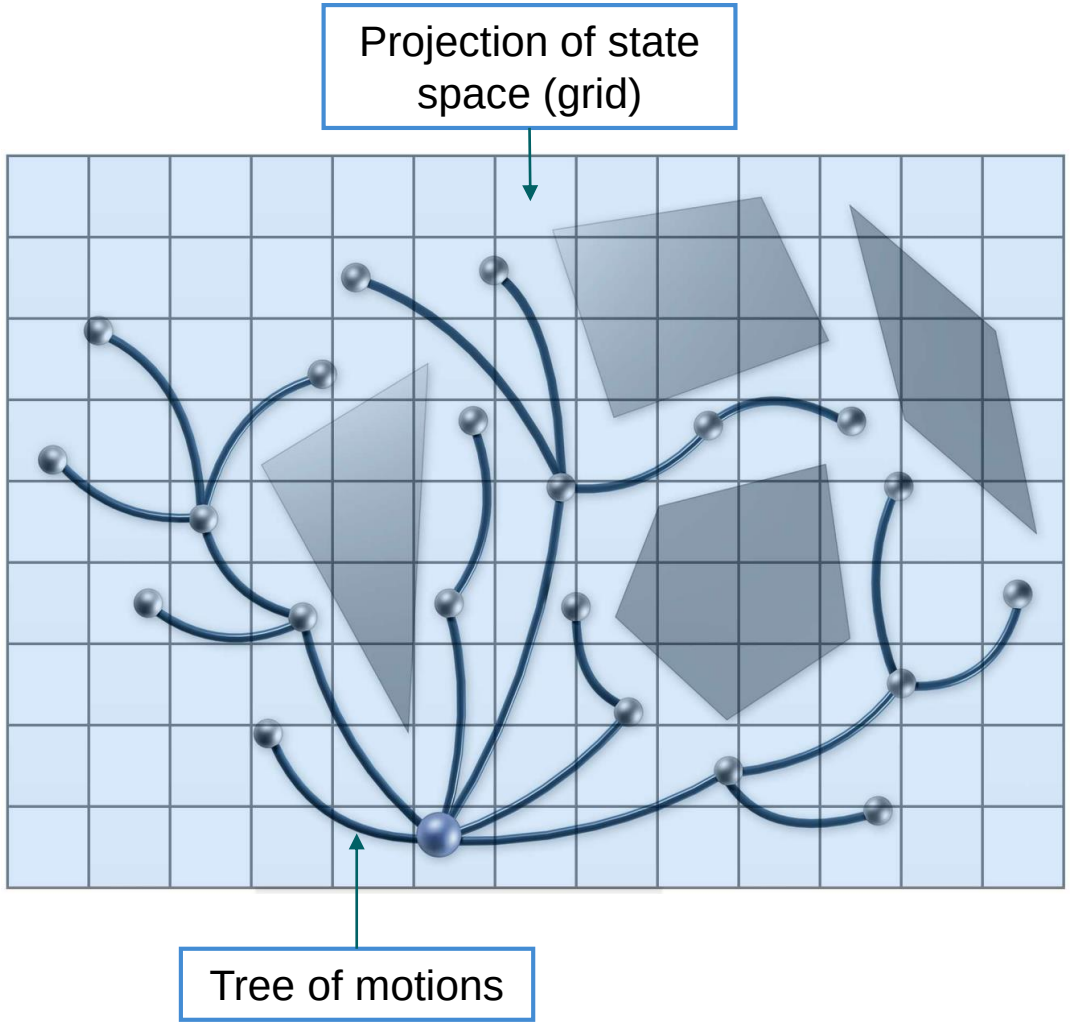
- KPIECE carefully selects motions for further expansion

  Selection strategy is based on estimating the coverage of the state space that the tree of motions achieves

- Experiments shown that projection is often easy to specify manually

- An algorithm for calculating the projection is also available

  [Sucan, Ioan Alexandru. Task and motion planning for mobile manipulators. Diss. Rice University Houston, 2011]

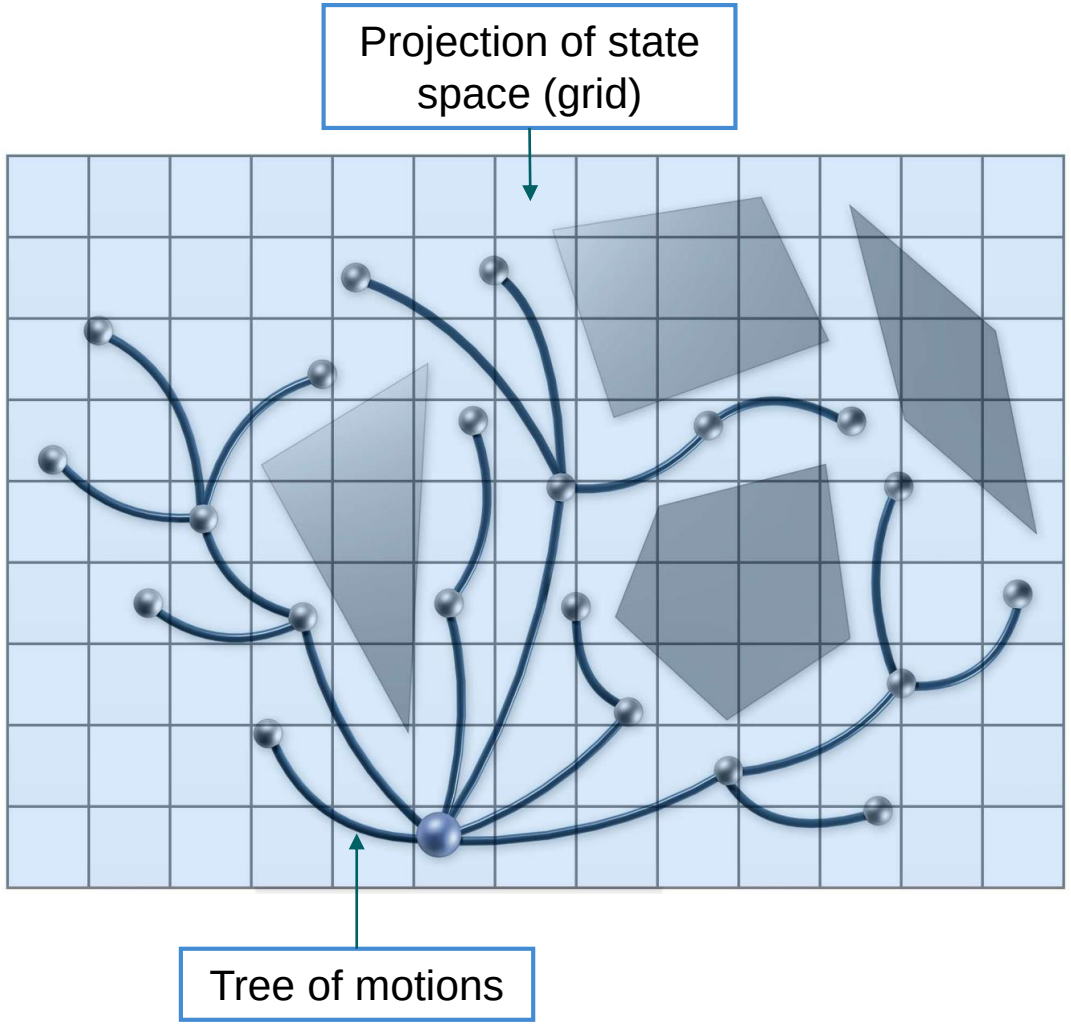


Motion Planning: Tree-based Planners

Find the shortest path between two points

We assume to have a motion tree

- How to find a path starting at the current state to a goal state?
- This is done by standard search algorithms (similar to navigation system in cars)
  - Breadth- and depth-search
  - Variant of A*-Algorithm
  - …

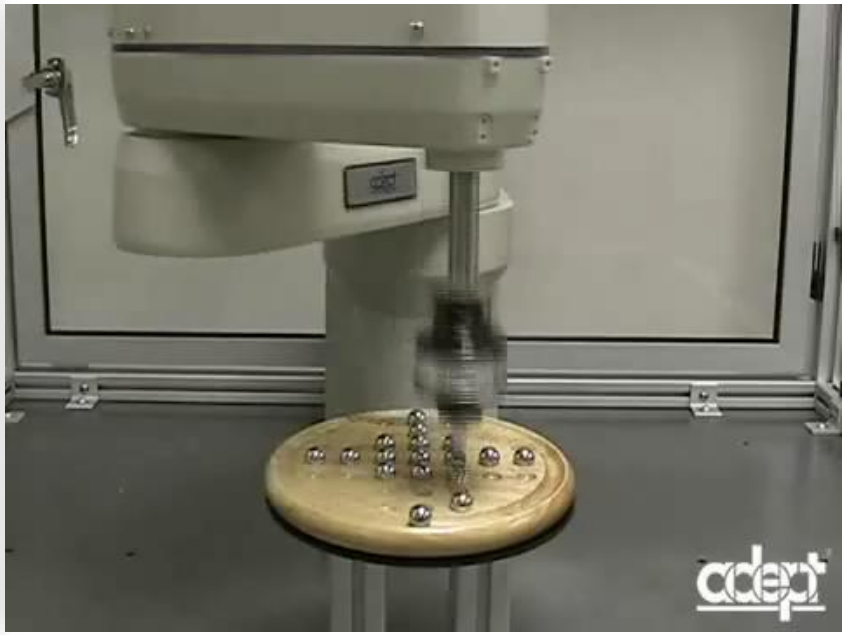


Outline

Motion Planning in Practical

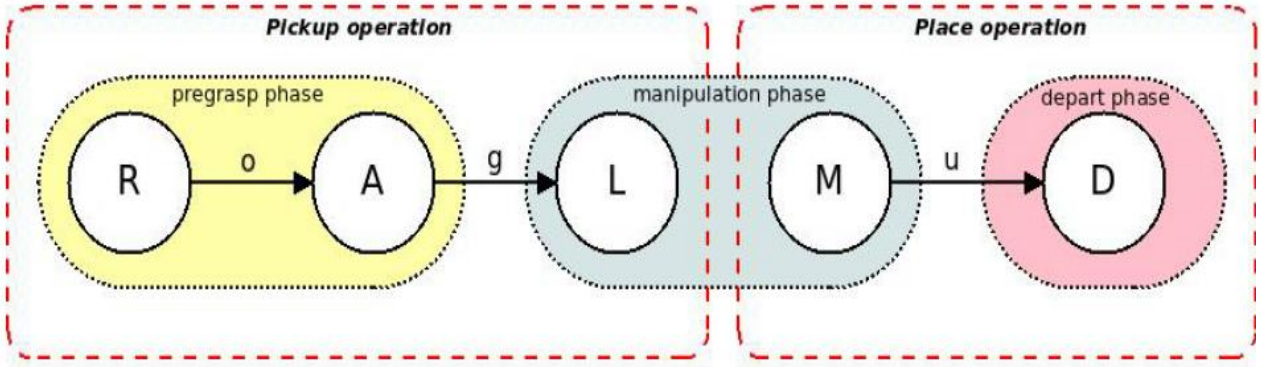
Pick and Place Tasks

- Common task in industry
- Take an object at position A and bring it to position B



Motion Planning in Practical

Pick and Place Tasks: Robot vs. Human



Stages
R: reach phase
A: approach phase
L: lift phase
M: movement phase
D: depart phase

Transitions
o: preshape phase
g: grasp phase
u: ungrasp phase

1. Pregrasp phase
   - Phase starts at an arbitrary robot configuration
   - Gripper is moved towards the grasp location
2. Grasp phase
   - The gripper gets in contact with the object and closes
3. Manipulation phase
   - The grasped object is enclosed by the gripper
   - The object is translated towards the place location by enforcing path constraints
4. Ungrasp phase
   - The gripper releases the object
5. Depart phase
   - The manipulator retreats from the object

Motion Planning in Practical

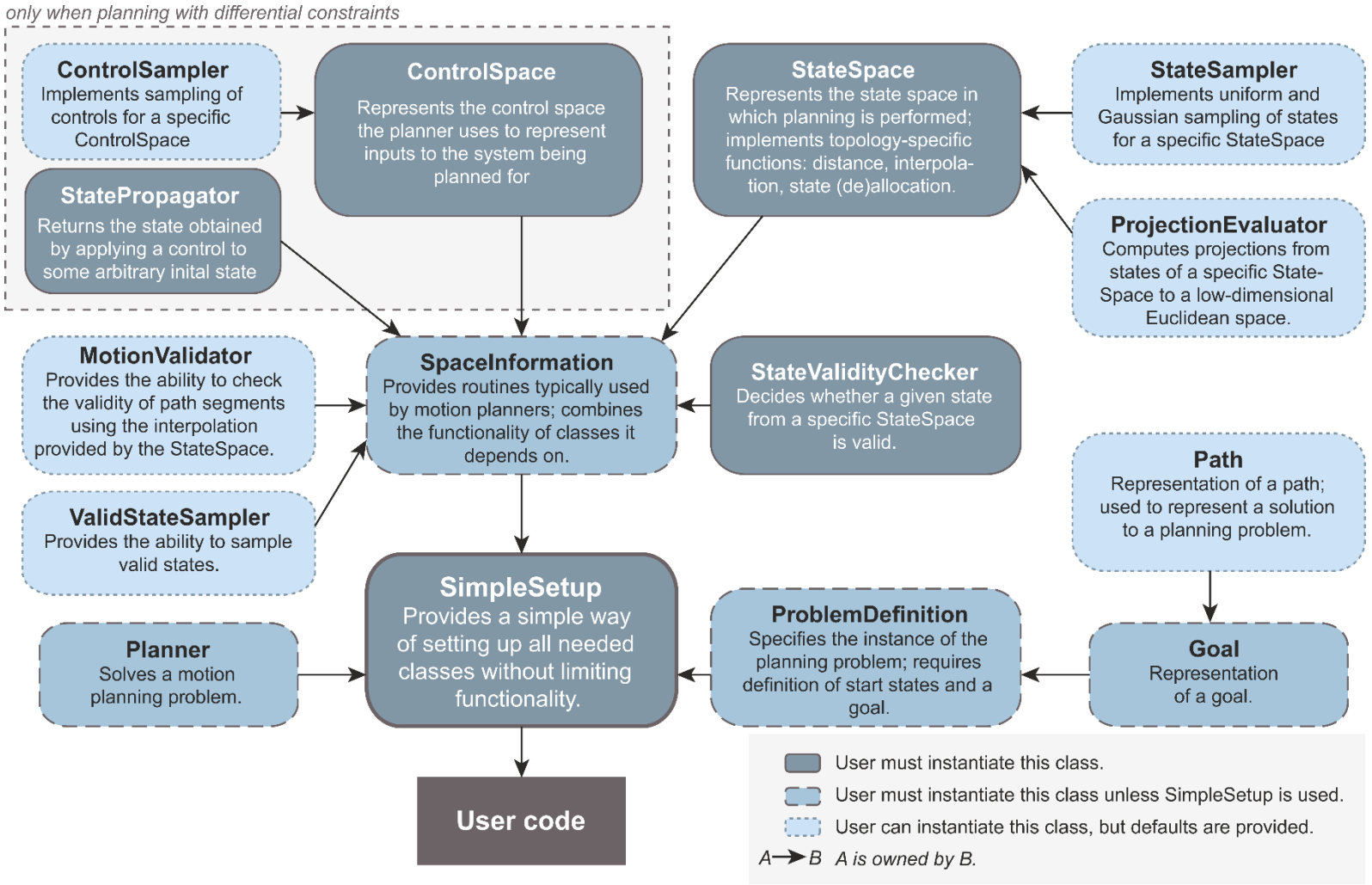
The Open Motion Planning Library (OMPL)

- Open Source Implementation of many sampling-based algorithms and core low-level data structures
- C++ with Python Bindings
- For Academic and industrial Settings

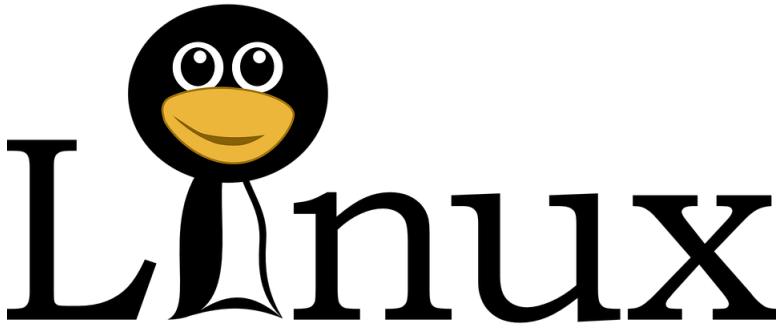
- Conceptual Overview of OMPL:
  - Efficiency
  - Simple integration with other software packages
  - Straightforward integration of external contributions
  - Clarity of Concepts

Motion Planning in Practical

The Open Motion Planning Library (OMPL)



What is coming?



Outline

1	Motion Planning	10:00 – 12:10
2	Lunch Time	12:10 – 14:30

Enjoy your Lunch
and
See you later!