Aalto University
School of Electrical Engineering

ELEC-E8126: Robotic Manipulation
Motion Planning

Ville Kyrki

25.1.2021

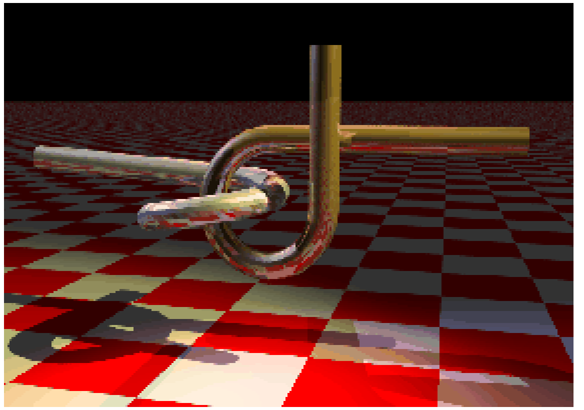
Today

- Robot motion planning problems.
- Graph search and discretization of continuous space.
- Sampling methods.
- A little bit about optimization based methods.

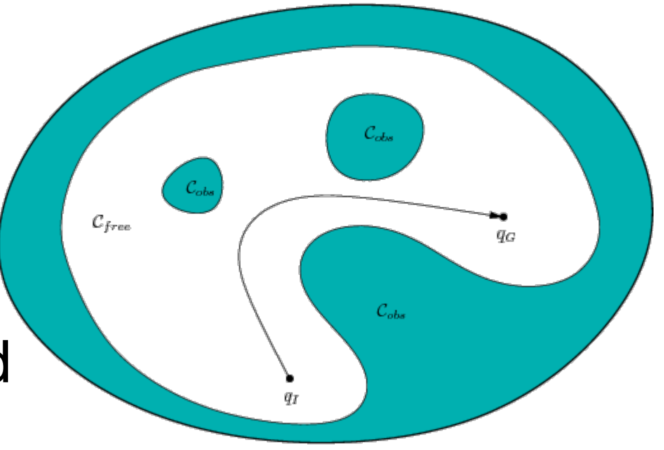
Learning goals

- Understand problems of motion planning as planning of trajectories in search space.
- Understand how discretization can be used to solve continuous planning problems.
  - Especially sampling based discretization approaches.

Motion planning (re-cap)

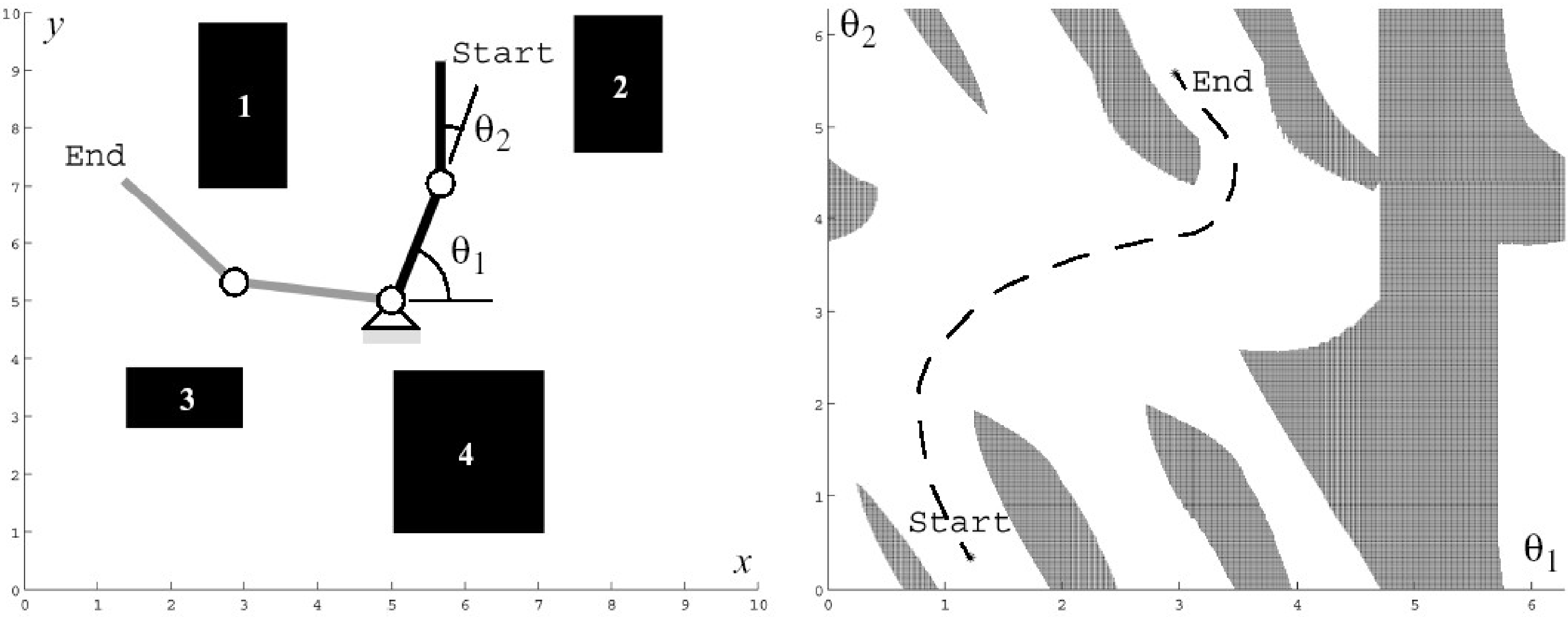


- Problem: Find actions that result in a path between two configuration space points while avoiding work space obstacles.



- Configuration (state) space: set of all transformations that can be applied to the robot.
- Work space (world): Space that robot occupies. Obstacles usually represented as Cartesian space regions.

Example:
Workspace vs configuration space



a) b)

Recap: Path vs motion planning

- Path planning: Find a collision free path in configuration space from start to end configuration.
- Motion planning: Find actions (control inputs), possibly with constraints on controls, duration, motion.

- Paths created by path planning can be turned into feasible trajectories by a trajectory planner.
- Trajectory planner determines time scale (velocity) over the path.

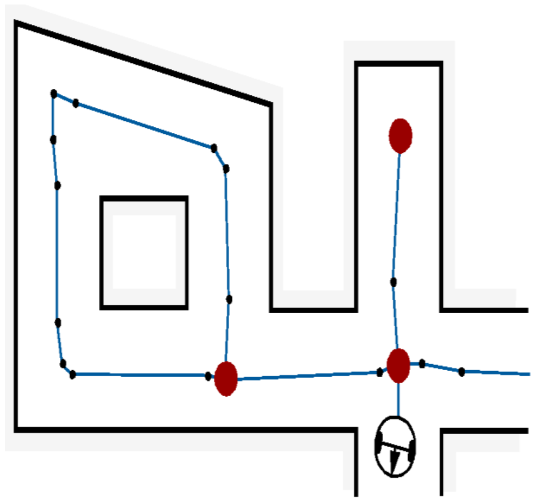
Discretization of configuration space

- Combinatorial vs sampling-based approaches

- Combinatorial: Divide free space and represent as graph.
  - Common in mobile robotics. Today a little bit of this.

- Sampling-based: Create a search tree incrementally by doing collision detection.
  - Can handle typically higher dimensions. Today mostly about this.

Continuous space planning by discretization

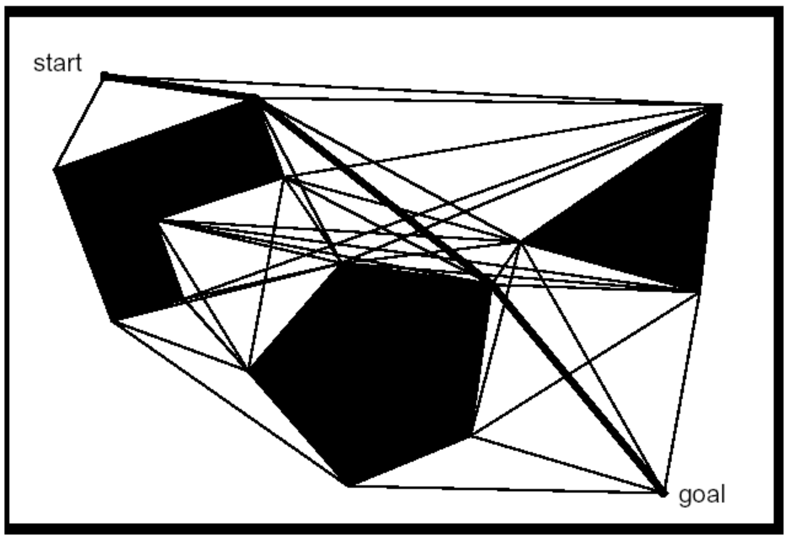


- After discretizing a continuous space, use discrete planning approaches such as Dijkstra, A*.
- Discretization builds a *roadmap*.
  - Roadmap graph: a set of routes in free space.
- How to discretize?
  - Does discretization affect solution in terms of feasibility/optimality?

Discretization approaches for polygonal obstacles

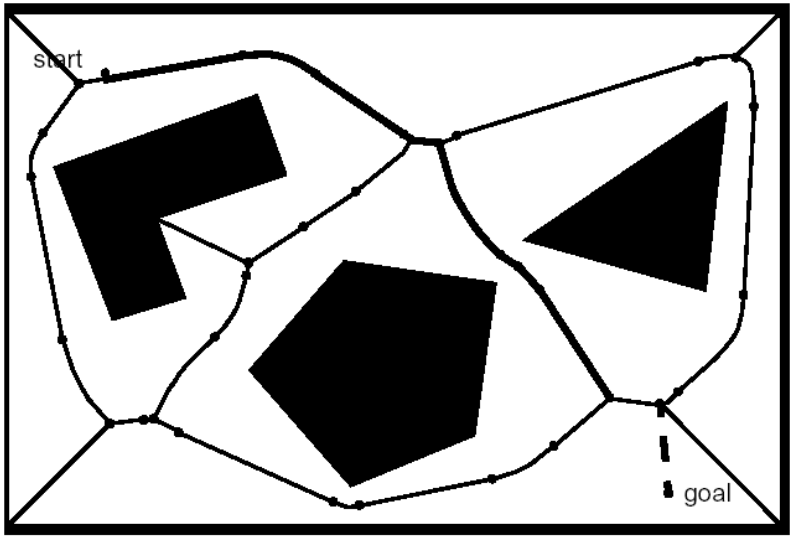
Visibility graph

- Shortest path length



Voronoi diagram

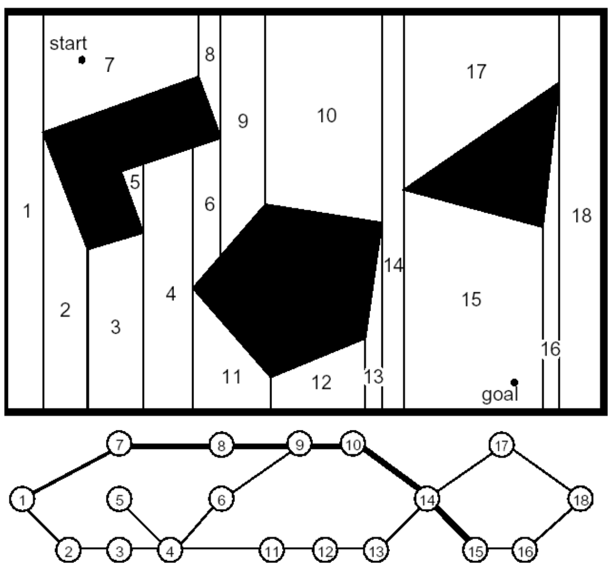
- Maximal clearance



Discretization by cell decomposition

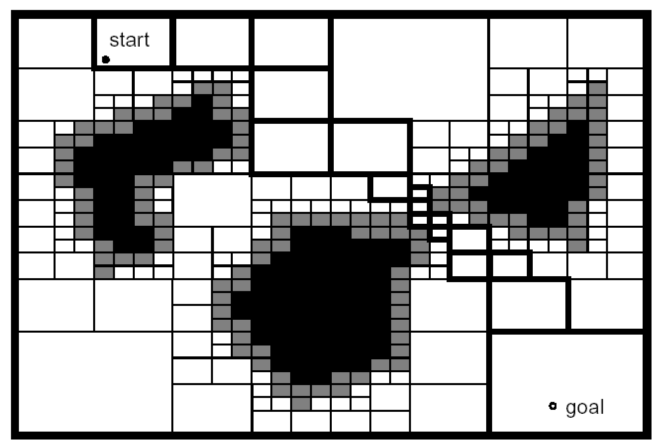
Exact cell decomposition

- Divide space into cells
- Determine which are adjacent



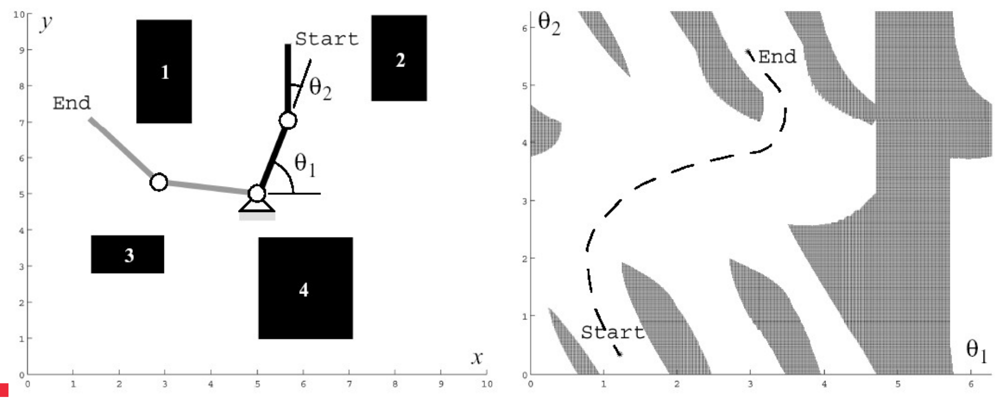
Approximate cell decomposition

- Divide space into cells of predefined shape
- Determine if each cell is free



Pros and cons of combinatorial approaches

- *Complete* approaches.
- Cannot handle well high-dimensional configuration spaces.
  - Combinatorial explosion (exponential number of states).
- Cannot handle easily non-linearities.
  - Obstacles cannot be easily represented with e.g. polygons.



a) b)

Sampling based search

- Idea: Build search graph iteratively.
  - Draw random samples of configuration space.
  - Use collision detection to determine if a state is free.

- Two common approaches:
  - Probabilistic roadmaps (Kavraki 1992)
  - Rapidly exploring random trees (LaValle & Kuffner, 1999)

Offline

On-line

Probabilistic roadmaps

- Idea: Build search graph (roadmap) iteratively (off-line).
  - Draw random samples of configuration space.
  - Check if they are free, and add to search graph if they are.
  - Try to connect nearby nodes using *local planner*.
  - Continues until roadmap dense enough.
- Local planner checks if straight-line trajectory is free.
- On-line operation:
  - Find paths from start and end configurations to nearby roadmap nodes using local planner.
  - Use the roadmap for the rest of the path.

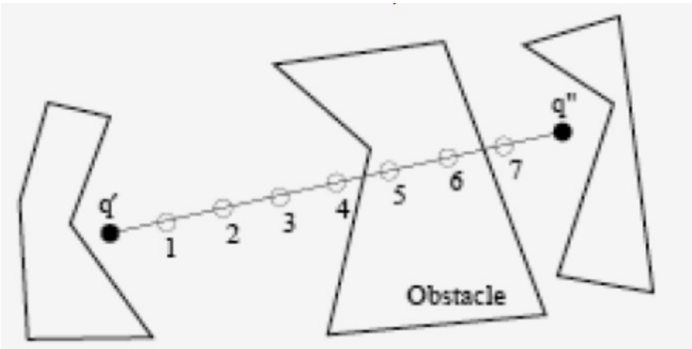
Sampling dense sequences

- Sampling has to be *dense* to allow each part of configuration space to be reachable from the roadmap.
- *Denseness* – getting arbitrarily close to any point in space.
  - Can you give an example?

- Random sequences are often dense with probability 1.
- Random sampling of e.g. orientations requires care.
  - Is it better to sample in configuration or workspace?

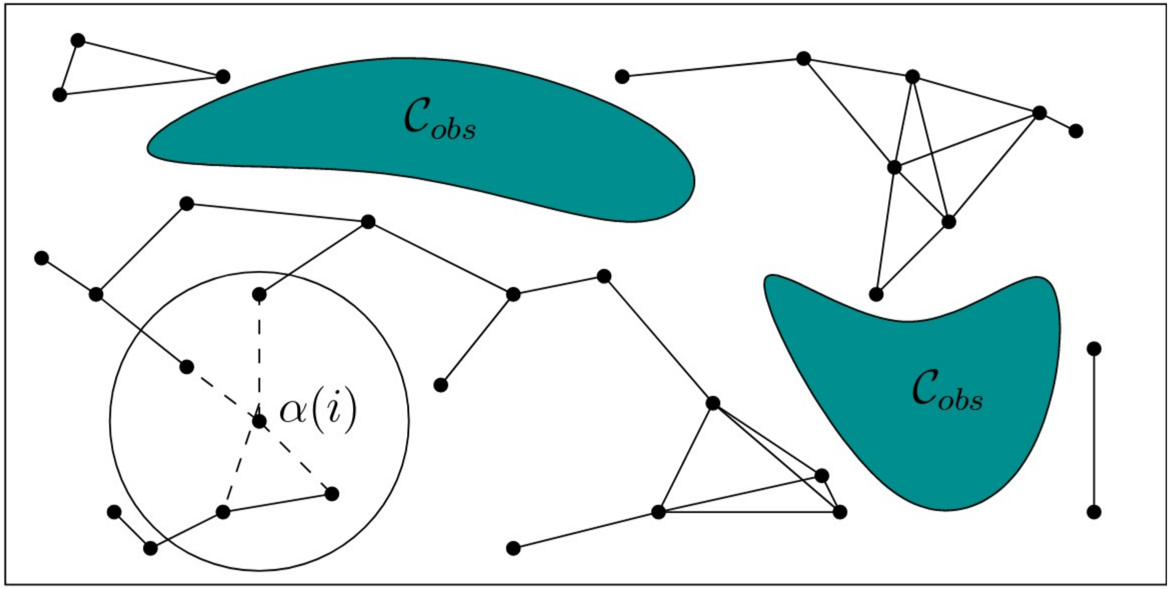
Local planner

- Check path between two points for collisions.
  - Number of points infinite.
- Local planner typically only checks discrete points along the path.
- What would be a good order to check the points?



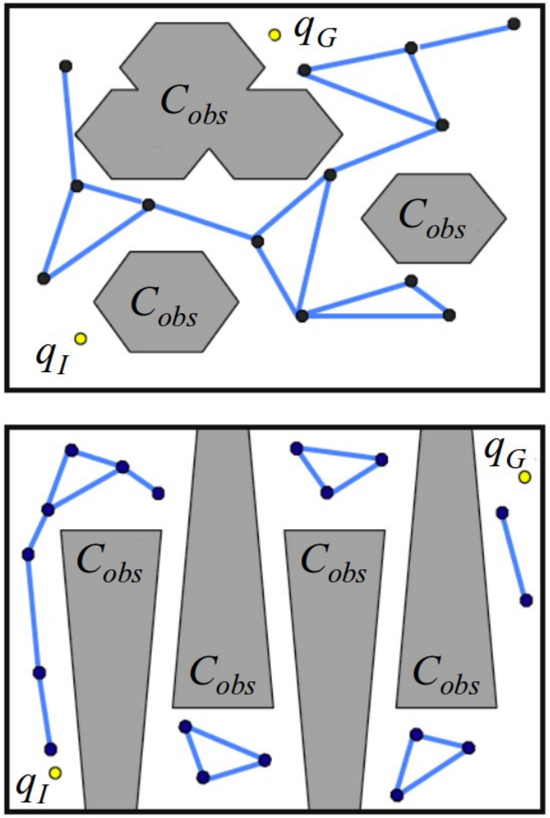
Connecting nodes

- Try to connect to points in a neighborhood using local planner.
  - K-nearest or inside a radius



PRM pros and cons

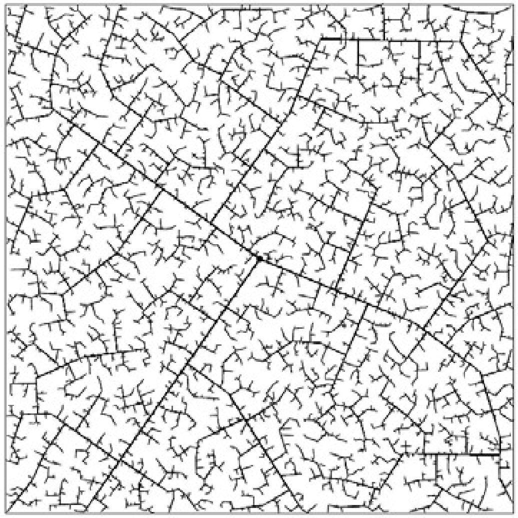
- Pros:
  - Probabilistically complete.
  - Applicable to high-dimensional configuration space.
- Cons:
  - Does not work well for some problems, e.g. narrow passages.
- Many extensions of PRMs exist.



Rapidly exploring random trees (RRT)

- Idea: Explore configuration incrementally from starting state.
  - Builds a tree rooted at starting state.

45 iterations



2345 iterations

Rapidly exploring random trees (RRT)

- Begin by choosing a random state.
  - Sample from bounded region around starting state.
  - Other sampling strategies also possible.

Algorithm 1: RRT

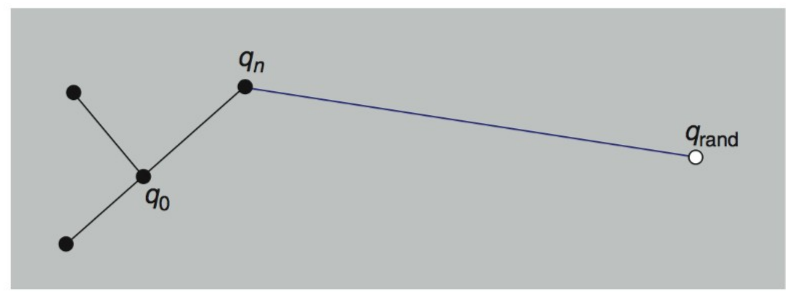
G.init(q_0)
repeat
    $q_{rand} \rightarrow$ RANDOM_CONFIG($\mathcal{C}$)
    $q_{near} \leftarrow$ NEAREST(G, q_{rand})
    G.add_edge(q_{near}, q_{rand})
until *condition*

Rapidly exploring random trees (RRT)

- Choose the nearest point in existing tree.
  - Choice of distance function affects.
  - Other similar strategies also possible.

Algorithm 1: RRT

$G.\mathrm{init}(q_0)$
repeat
  $q_{rand} \rightarrow \mathrm{RANDOM\_CONFIG}(\mathcal{C})$
  $q_{near} \leftarrow \mathrm{NEAREST}(G, q_{rand})$
  $G.\mathrm{add\_edge}(q_{near}, q_{rand})$
until *condition*

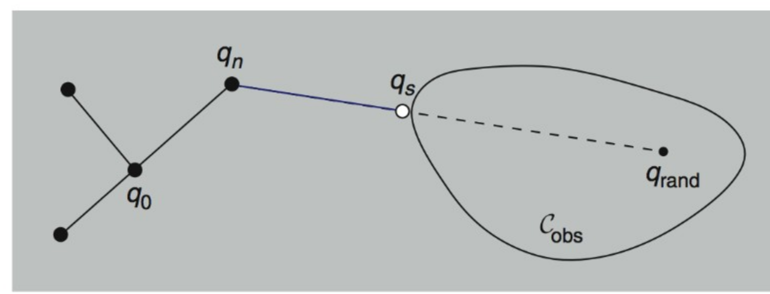


Rapidly exploring random trees (RRT)

- Check for collision free path using local planner.
  - If it exists, connect nodes.
  - It not, connect to last state before obstacle.

Algorithm 1: RRT

G.init(q_0)
repeat
  $q_{rand} \rightarrow$ RANDOM_CONFIG($\mathcal{C}$)
  $q_{near} \leftarrow$ NEAREST(G, q_{rand})
  G.add_edge(q_{near}, q_{rand})
until *condition*



Rapidly exploring random trees (RRT)

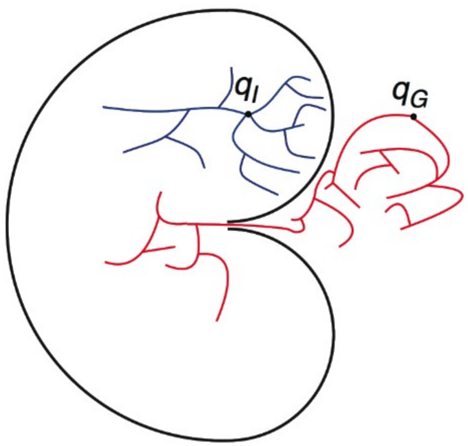
- From time to time, choose goal state instead of the random, to check if a solution can be found.

Algorithm 1: RRT

G.init(q_0)
repeat
  $q_{rand} \rightarrow$ RANDOM_CONFIG($\mathcal{C}$)
  $q_{near} \leftarrow$ NEAREST(G, q_{rand})
  G.add_edge(q_{near}, q_{rand})
until *condition*

Rapidly exploring random trees (RRT)

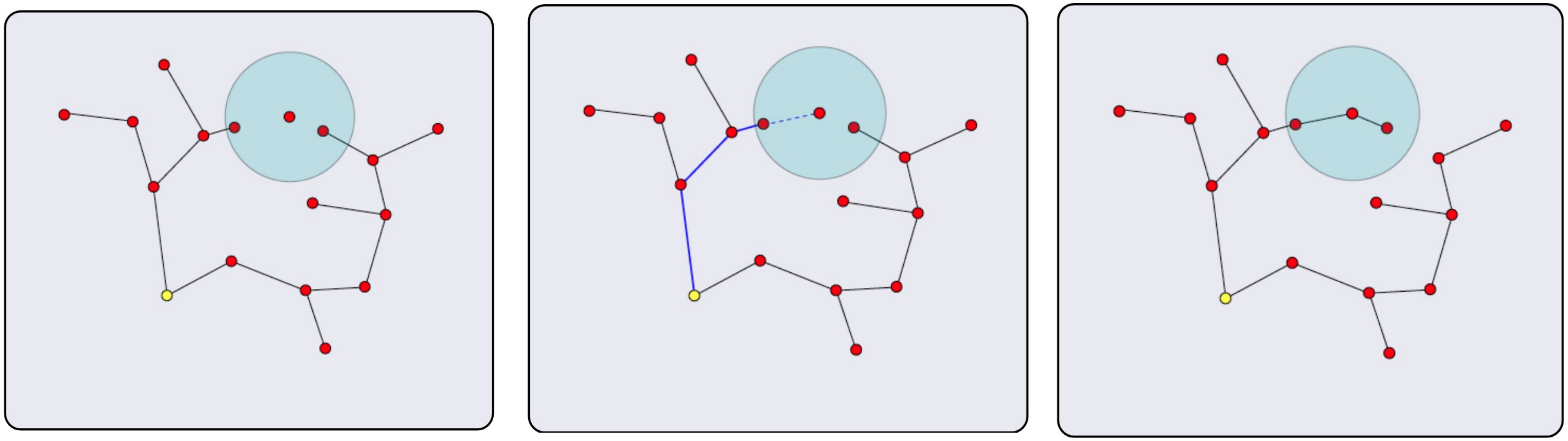
- Many extensions available.
  - For example, expand both from starting and goal states (BiRRT).
- Easy to implement.
- Probabilistically complete.
- Unknown rate of convergence.
- Widely used.
- Narrow corridors still problematic.



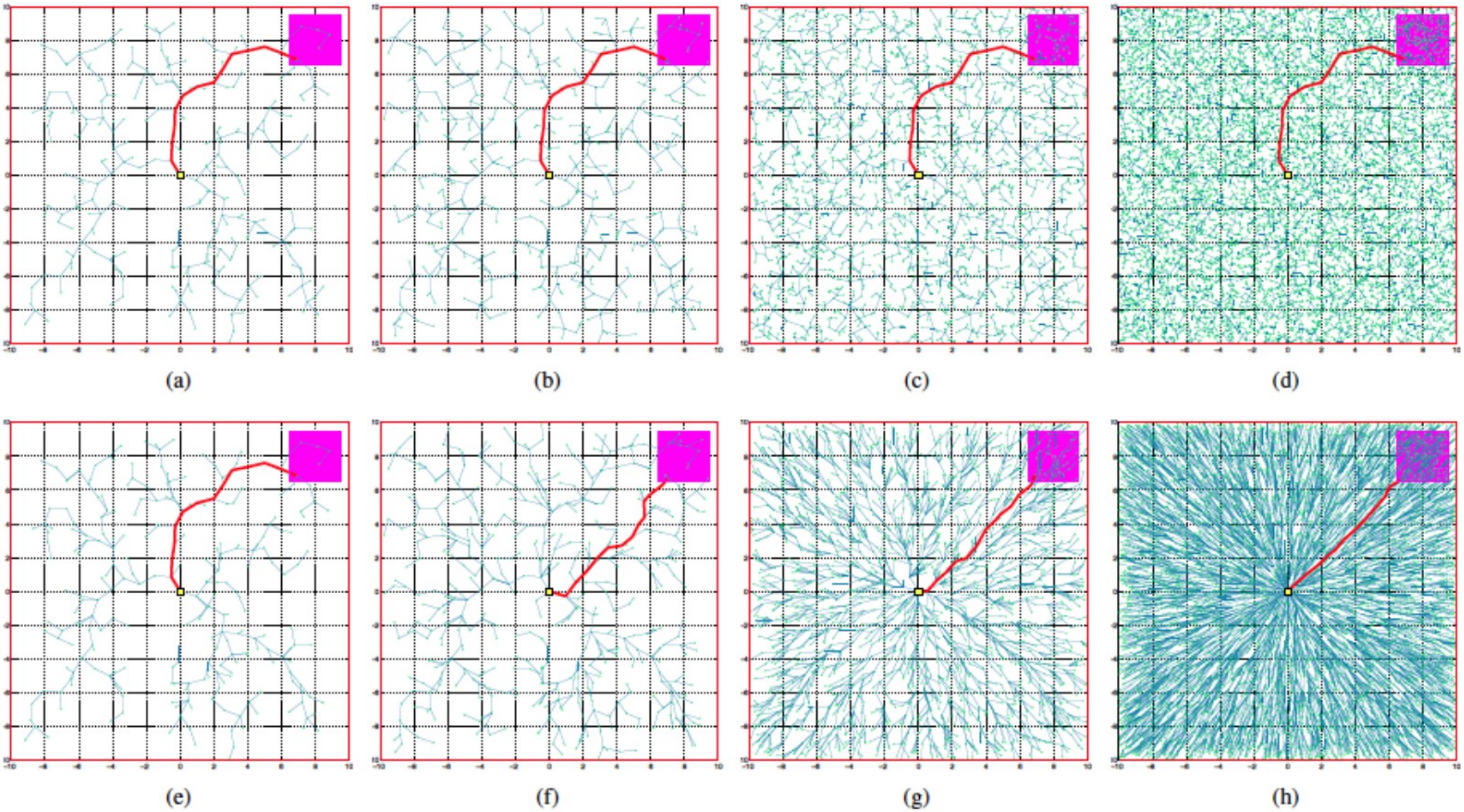
RRT*

When a new node is added, tree can be locally rewired in small area around added node.

This will optimize path lengths.



RRT*



Kinematic vs dynamic planning

- So far planning considered as finding a state-space trajectory, without considering constraints on dynamics.
- If inverse dynamics is available, it can be used to solve actions for a path.

Do inverse dynamics always exist?

- RRTs can be turned into *control-based* planners by substituting sampling of state by sampling of control.
- How to sample controls is a central question.
- This approach can be used for general continous space planning problems.

Motion planning as optimization

- Motion planning can be solved as nonlinear optimization
- **Optimal paths**, fast computation when good initial guess available, **possible local minima**.

inputs,
e.g. joint velocities

cost function,
e.g. path length

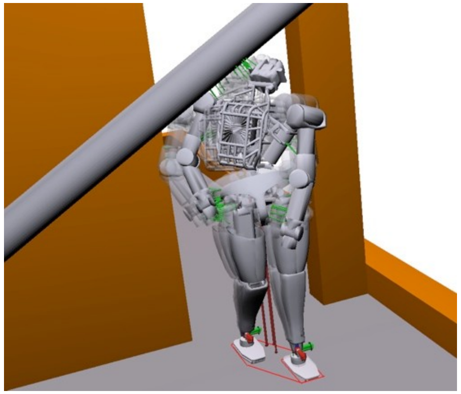
$$
\begin{array}{rll}
\text{find} & u(t), q(t), T & \\
\text{minimizing} & J(u(t), q(t), T) & \\
\text{subject to} & \dot{x}(t) = f(x(t), u(t)), & \forall t \in [0, T], \\
 & u(t) \in \mathcal{U}, & \forall t \in [0, T], \\
 & q(t) \in \mathcal{C}_{\text{free}}, & \forall t \in [0, T], \\
 & x(0) = x_{\text{start}}, & \\
 & x(T) = x_{\text{goal}}. &
\end{array}
$$

motion
constraints

collision
constraints

Motion planning as optimization

- Methods e.g. TRAJOPT, CHOMP.
- Typical solution uses sequential convex optimization.
  - Iterate solving convex approximations of non-convex problem around current solution.
  - For example, handle constraints by turning into penalties.



$$min_{u,q} J_{TOT}(u,q)$$
$$J_{TOT}(u,q) = J_{SOL}(u,q) + \mu J_{CONSTR}(u,q)$$

Solution cost, e.g. trajectory length

Large enough factor

Constraint cost, e.g. penetration depth

Software

- Open motion planning library (OMPL) encapsulates many motion planning algorithms.
- In robotics, ROS MoveIt uses OMPL.

- https://vimeo.com/58709589
- https://www.youtube.com/watch?v=eUpvbOxrbwY

Summary

- Kinematic motion planning searches for admissible state space trajectories.
- Search in continuous state space requires discretization.
- In high dimensional state spaces stochastic discretization often applicable.
- Controls can be sampled instead of states to solve more general planning problems.

Next time: Perception for manipulation

- Readings:
  – Lynch & Park, Chapter 10

Note: Non-holonomic motion planning

- Robot is *underactuated*, if control space has fewer dimensions than configuration space.
  - E.g. a car.

- Robot is *nonholonomic*, if its motion is constrained by a non-integrable equation of form $f\left(\theta, \dot{\theta}\right) = 0$
  - What's the constraint for a car?