Motion Planning: Overview and Path Planning

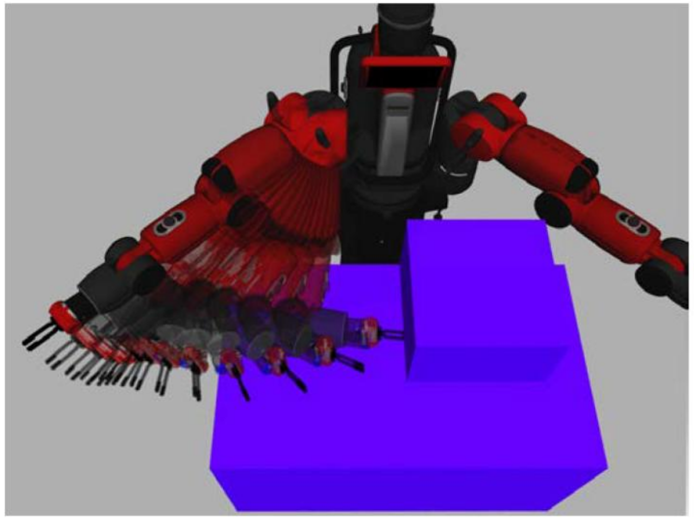
CS 6341 Robotics

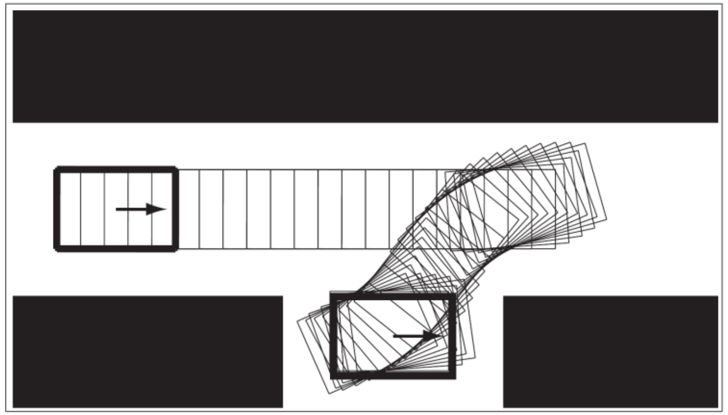
Professor Yu Xiang

The University of Texas at Dallas

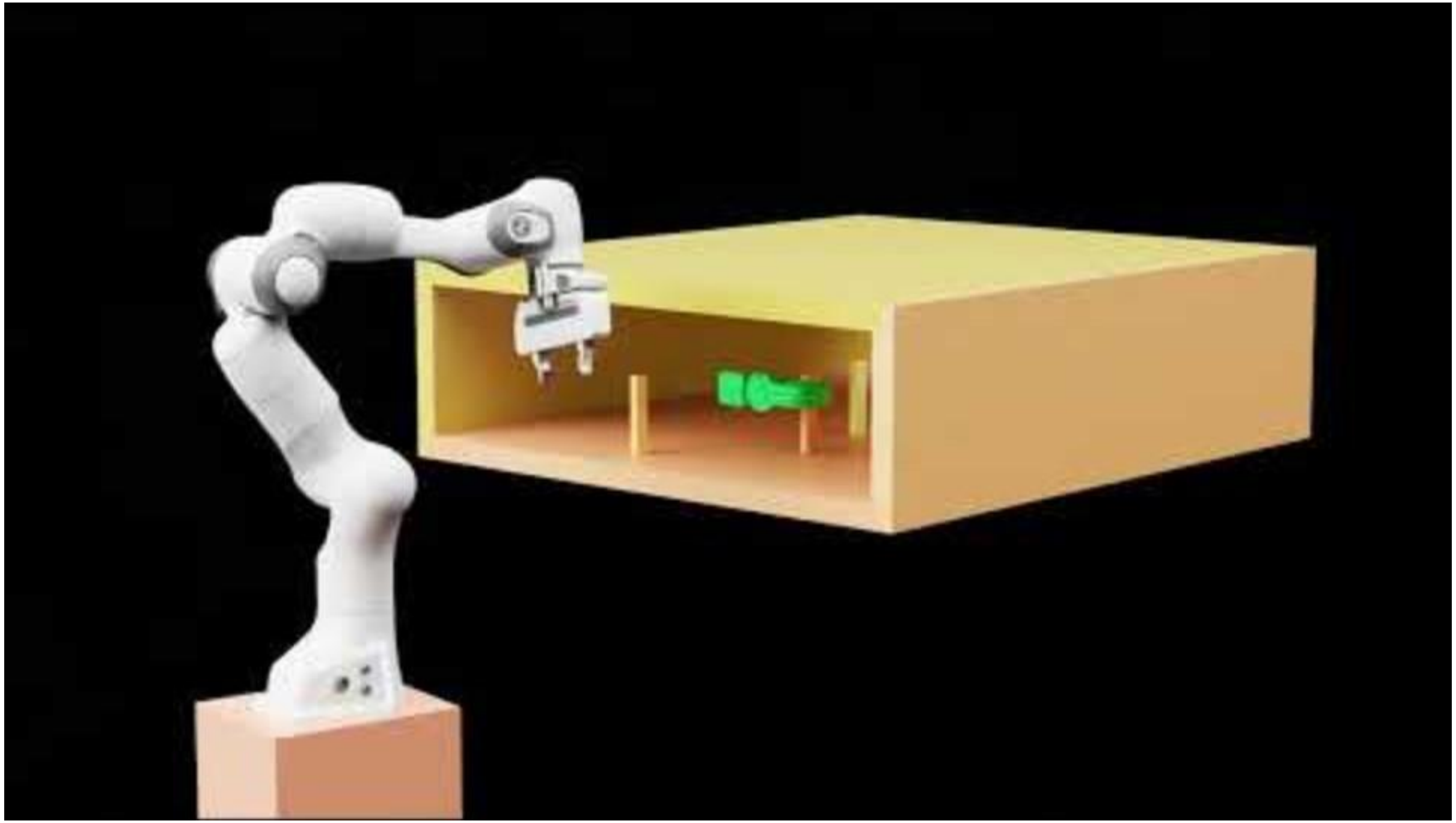
Motion Planning

- Motion planning: finding a robot motion from a start state to a goal state (A to B)
  - Avoids obstacles
  - Satisfies other constraints such as joint limits or torque limits





Example: cuRobo from NVIDIA



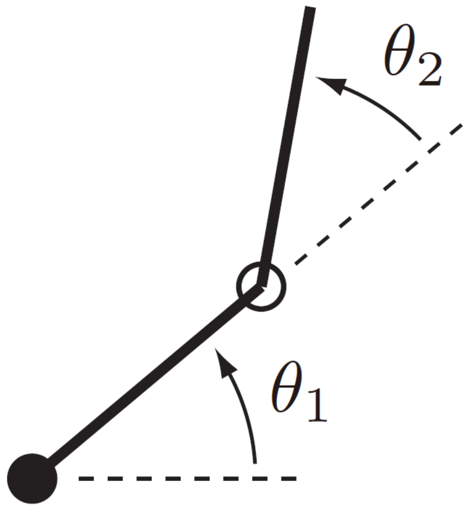
https://developer.nvidia.com/blog/cuda-accelerated-robot-motion-generation-in-milliseconds-with-curobo/

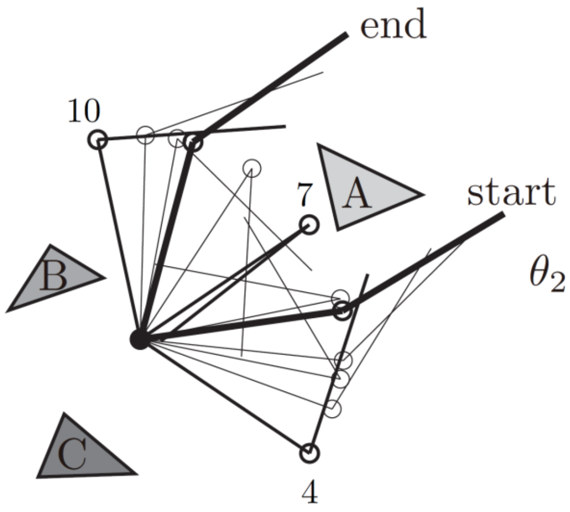
Configuration Space

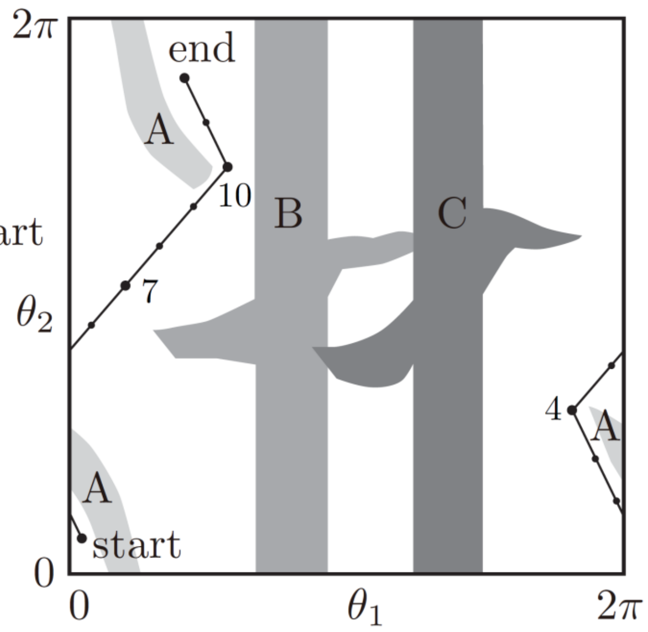
- The configuration of a robot arm with n joints
  - n joint positions $q = (\theta_1, \ldots, \theta_n)$
- Free C-space $\mathcal{C}_{\text{free}}$
  - Configurations where the robot neither penetrates an obstacle nor violated a joint limit
- Obstacles in C-space $\mathcal{C}_{\text{obs}}$ $\quad \mathcal{C} = \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{obs}}$
  - Joint limits are treated as obstacle in the configuration space

Configuration Space Obstacles

- A 2R planar arm







Configuration space

Distance to Obstacles in Configuration Space

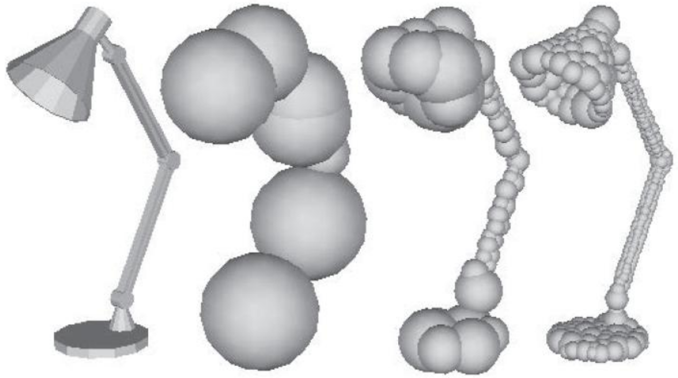
- Given a C-obstacle $\mathcal{B}$ and a configuration q, the distance between a robot and the obstacle

$$\begin{aligned} d(q,\mathcal{B}) &> 0 && \text{(no contact with the obstacle)}, \\ d(q,\mathcal{B}) &= 0 && \text{(contact)}, \\ d(q,\mathcal{B}) &< 0 && \text{(penetration)}. \end{aligned}$$

- A distance measurement algorithm determines $d(q,\mathcal{B})$
- A collision detection algorithm determines whether $d(q,\mathcal{B}_i) \leq 0$

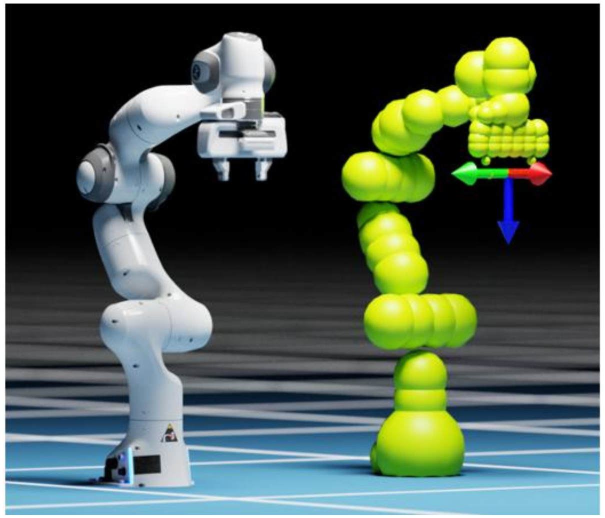
Distance to Obstacles

- Approximation of 3D shapes using 3D spheres



- Robot: k spheres of radius R_i centered at $r_i(q)$
- Obstacle: l spheres of radius B_j centered at b_j
- The distance between the robot and the obstacle

$$d(q, \mathcal{B}) = \min_{i,j} \|r_i(q) - b_j\| - R_i - B_j$$



cuRobo

Robot State

- For first order dynamics, state is the configuration

State

Joint position $x = q$ $\dot{x} = \dot{q}$

Control input: velocity Velocity Control

- For second order dynamics, state is configuration and velocity

State $x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}$ $\dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix}$

Control input $u \in \mathcal{U} \subset \mathbb{R}^m$ Force (acceleration)

Force/Torque Control

Equations of Motion

- The equations of motion of a robot

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

Forward dynamics

Robot state

Control inputs $u \in \mathcal{U} \subset \mathbb{R}^m$

First order dynamics

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

Second order dynamics

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \quad \dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} \quad u = \tau$$

For example

$$\ddot{q} = M^{-1}(q)\left[\tau - C(q,\dot{q})\dot{q} - g(q)\right]$$

$$f(x, u) =$$

$$\dot{x} = \begin{bmatrix} \dot{q} \\ M^{-1}(q)\left[u - C(q,\dot{q})\dot{q} - g(q)\right] \end{bmatrix}$$

Equations of Motion

- The equations of motion of a robot

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

Forward dynamics

Robot state

Control inputs $u \in \mathcal{U} \subset \mathbb{R}^m$

- Integral form

$$x(T) = x(0) + \int_0^T f(x(t), u(t))dt$$

Numerical approximation

$$x_{k+1} = x_k + f(x_k, u_k)\,\Delta t$$

Motion Planning

- Given an initial state $x(0) = x_{\text{start}}$ and a desired final state x_{goal} find a time T and a set of control $u : [0, T] \to \mathcal{U}$ such that the motion

$$x(T) = x(0) + \int_0^T f(x(t), u(t))dt$$

satisfies

$$x(T) = x_{\text{goal}}$$

$$q(x(t)) \in \mathcal{C}_{\text{free}} \ \textit{for all } t \in [0, T]$$

Robot motion planning needs to find the control inputs

Path Planning vs. Motion Planning

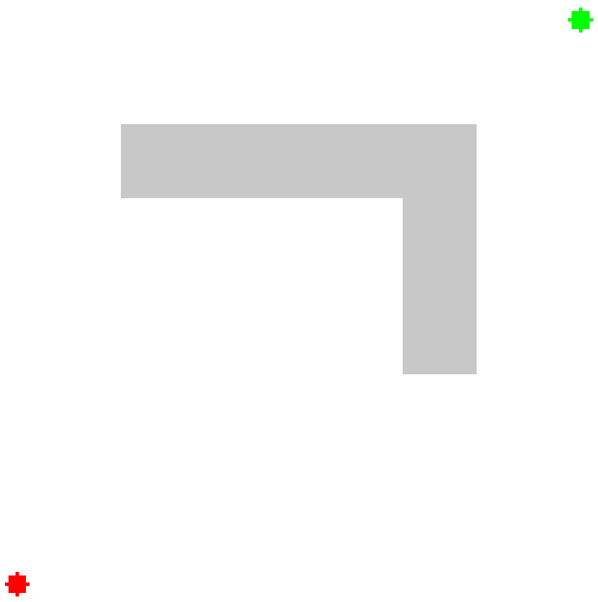
- Path planning is a purely geometric problem of finding a collision-free path

$$q(s), s \in [0, 1] \quad q(0) = q_{\text{start}} \quad q(1) = q_{\text{goal}}$$

- No concern about dynamics/control inputs

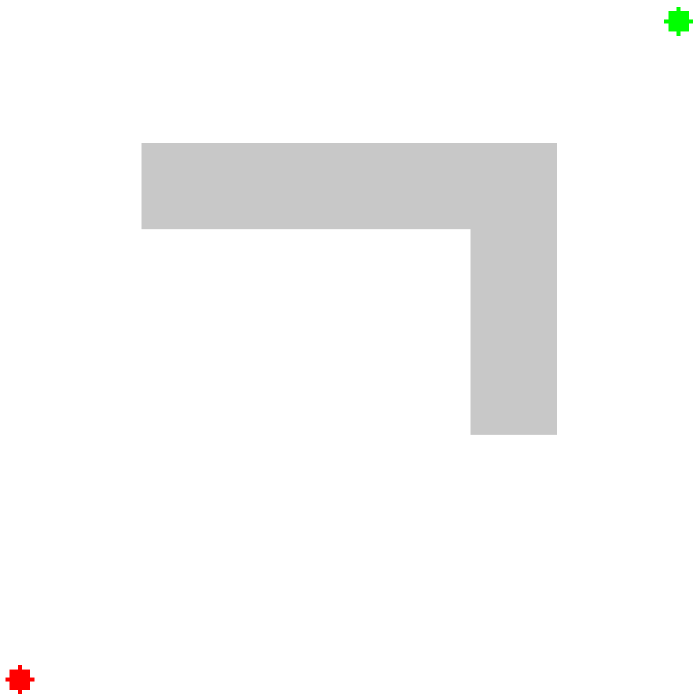
2D Path Planning

Dijkstra's Algorithm



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

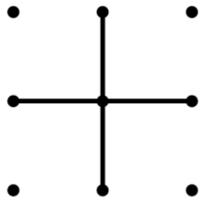
A* Search Algorithm



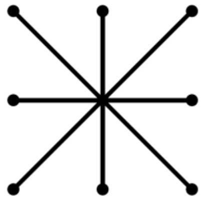
https://en.wikipedia.org/wiki/A*_search_algorithm

C-Space Path Planning: Grid Methods

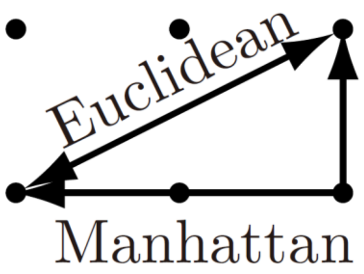
- Discretize the configuration space into a grid
  - If the C-space is n dimension, we use k grid points along each dimension
  - The C-space is represented by k^n grid points
- We can apply the A* search algorithm for path planning with a C-space grid



4-connected

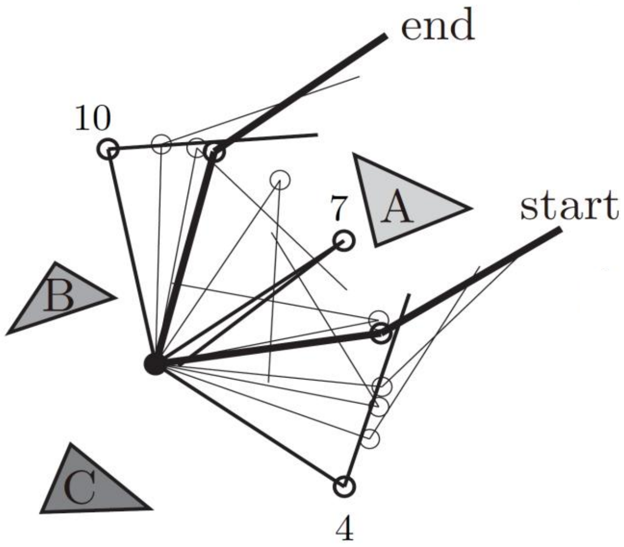


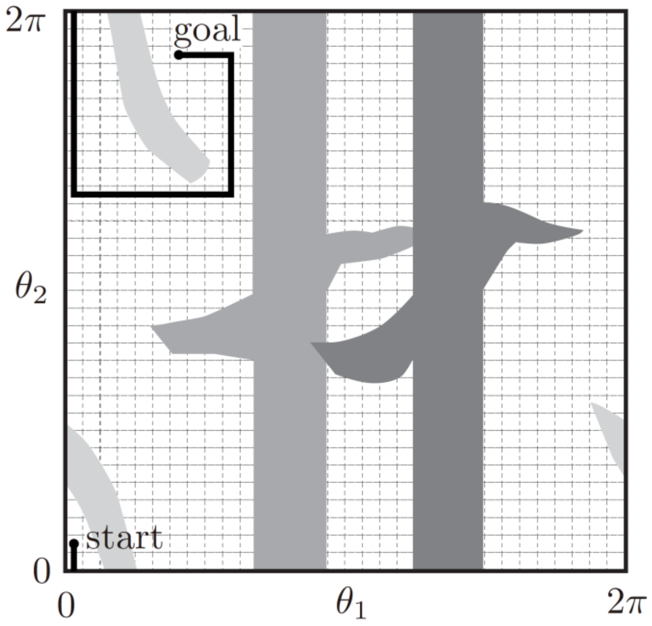
8-connected



C-Space Path Planning: Grid Methods

- A* grid-based path planner





$$k = 32$$

Grid-based path planning is only suitable for low-dimensional C-space

Number of grid points k^n

```
>>> np.power(32, 7.0)
34359738368.0
```

C-Space Path Planning: Sampling Methods

- Sampling methods
  - Randomly or deterministically sampling the C-space or state-space to find the motion plan
  - Give up resolution-optimal solutions of a grid search, quickly find solutions in high-dimensional state space
  - Most sampling methods are probabilistically complete: the probability of finding a solution, when one exists, approaches 100% as the number of samples goes to infinity

Rapidly exploring Random Trees (RRTs)

Algorithm 10.3 RRT algorithm.

1: initialize search tree T with x_{start}
2: **while** T is less than the maximum tree size **do**
3: $x_{\text{samp}} \leftarrow$ sample from $\mathcal{X}$
4: $x_{\text{nearest}} \leftarrow$ nearest node in T to x_{samp}
5: employ a local planner to find a motion from x_{nearest} to x_{new} in the direction of x_{samp}
6: **if** the motion is collision-free **then**
7: add x_{new} to T with an edge from x_{nearest} to x_{new}
8: **if** x_{new} is in $\mathcal{X}_{\text{goal}}$ **then**
9: **return** SUCCESS and the motion to x_{new}
10: **end if**
11: **end if**
12: **end while**
13: **return** FAILURE

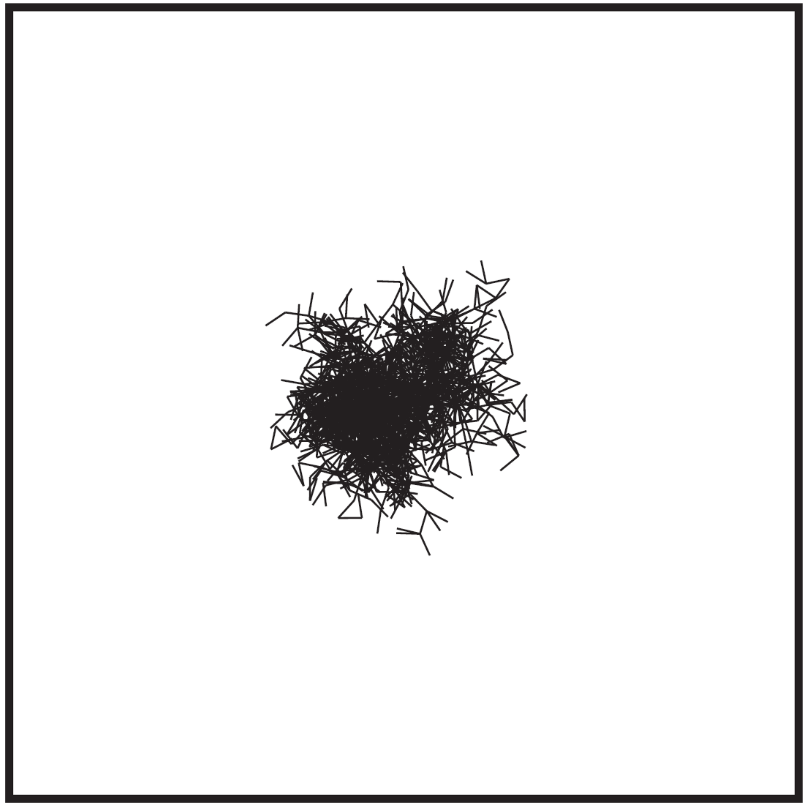
kinematic problems

$$x = q$$

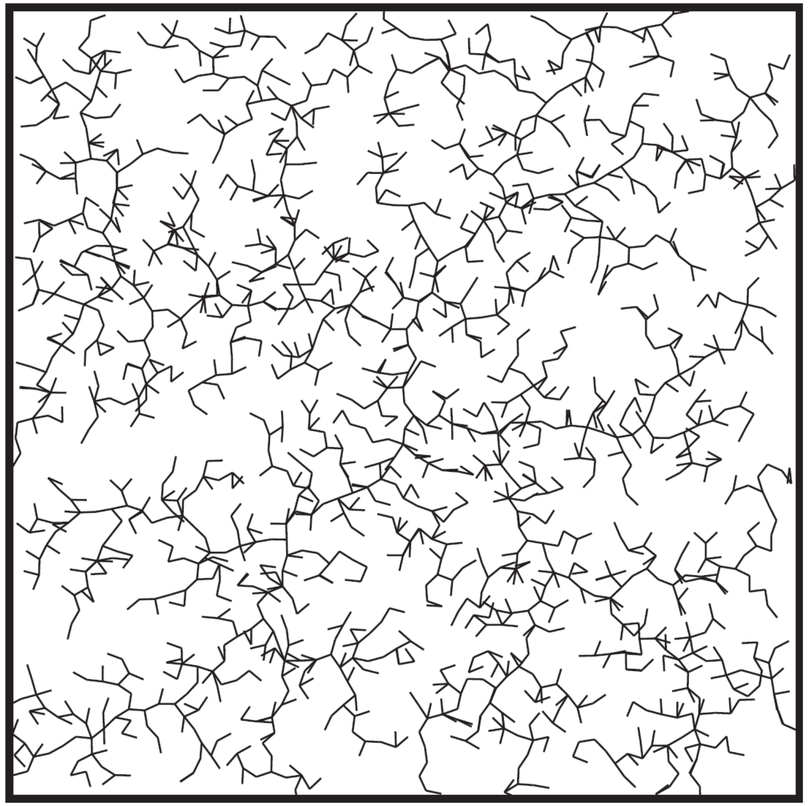
- Line 3, uniform sampling with a bias towards goal
- Line 4, Euclidean distance
- Line 5, use a small distance d from, check collision along the line

x_{nearest} on the straight line to x_{samp}

Rapidly exploring Random Trees (RRTs)



A tree generated by applying a uniformly-distributed random motion from a randomly chosen tree node does not explore very far.



A tree generated by the RRT algorithm

2000 nodes

Rapidly exploring Random Trees (RRTs)

An animation of an RRT starting from iteration 0 to 10000

https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

Summary

- Overview of motion planning
  - Configuration space obstacle
  - Distance to obstacles
  - Robot state
  - Equation of motion

- Path Planning
  - Grid methods
  - Sampling methods

Further Reading

- Chapter 10 in Kevin M. Lynch and Frank C. Park. Modern Robotics: Mechanics, Planning, and Control. 1st Edition, 2017.