Motion Planning: Algorithms

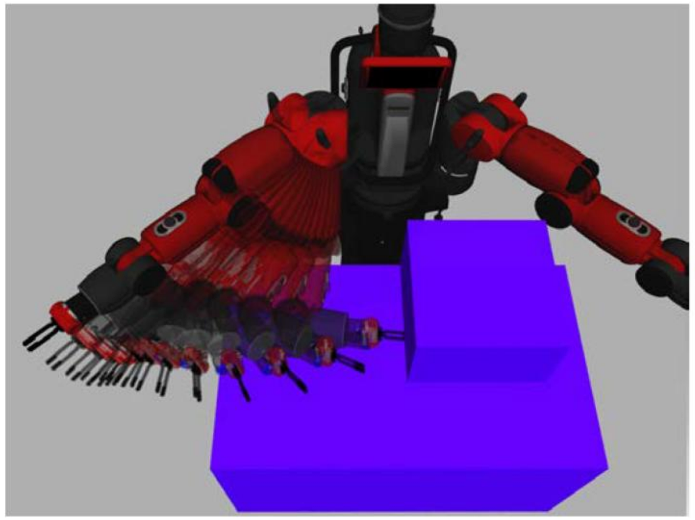
CS 6341 Robotics

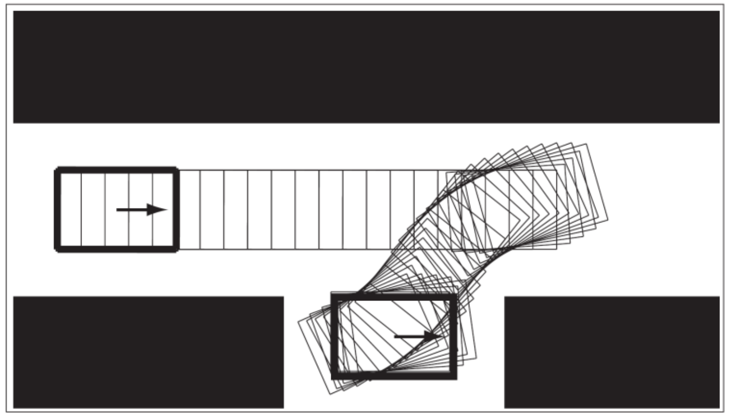
Professor Yu Xiang

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





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 $u : [0, T] \rightarrow \mathcal{U}$ 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. Otherwise, it may plan a motion that is not feasible for the robot.

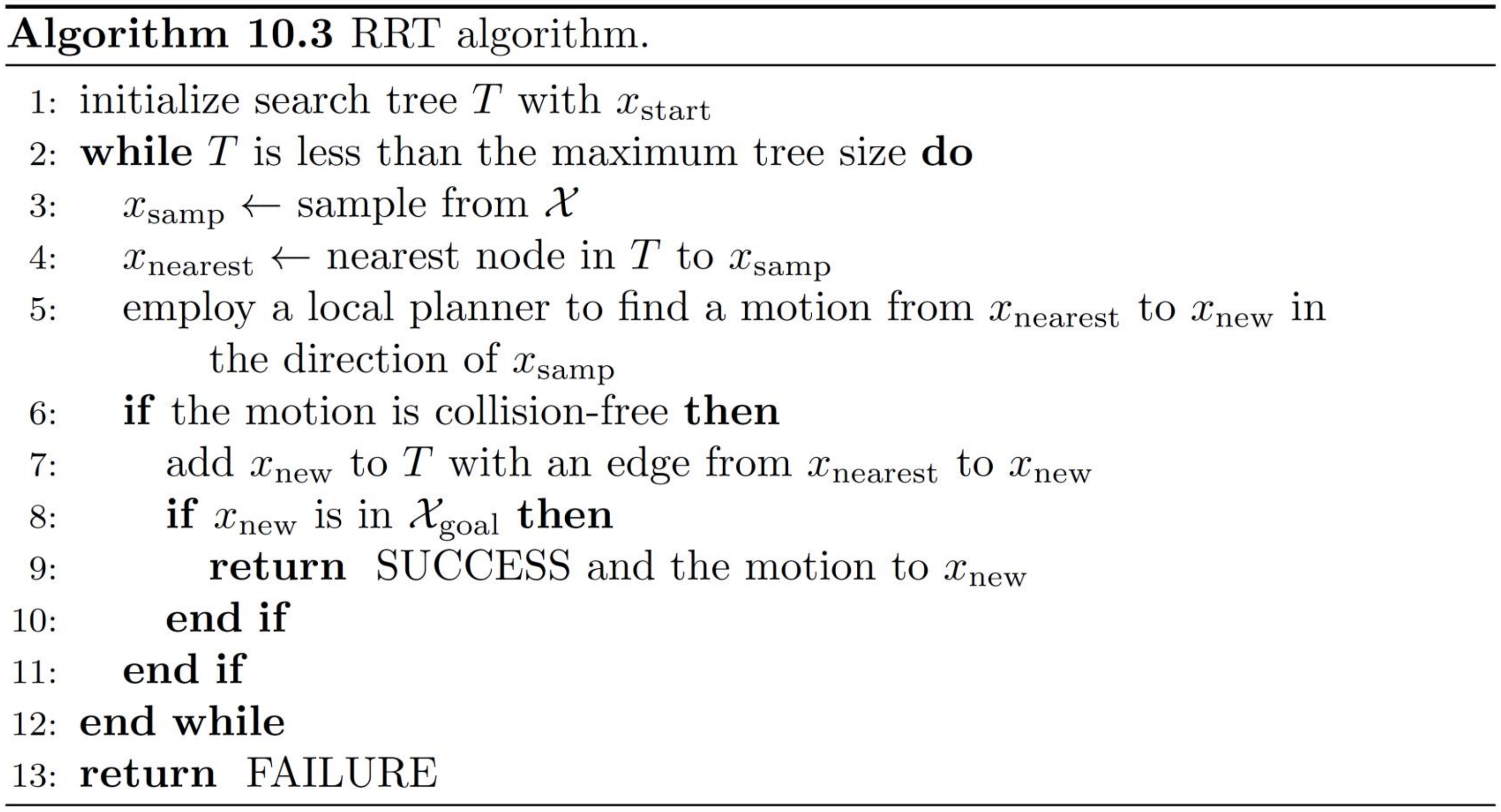
Path Planning vs. Motion Planning

- 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

Rapidly exploring Random Trees (RRTs)



Algorithm 10.3 RRT algorithm.

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

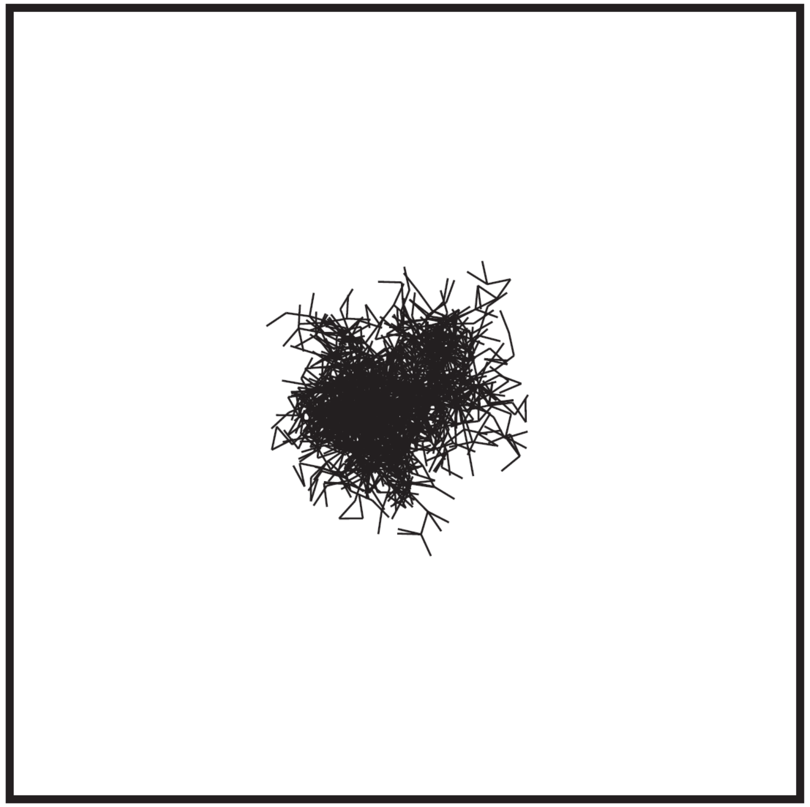
kinematic problems

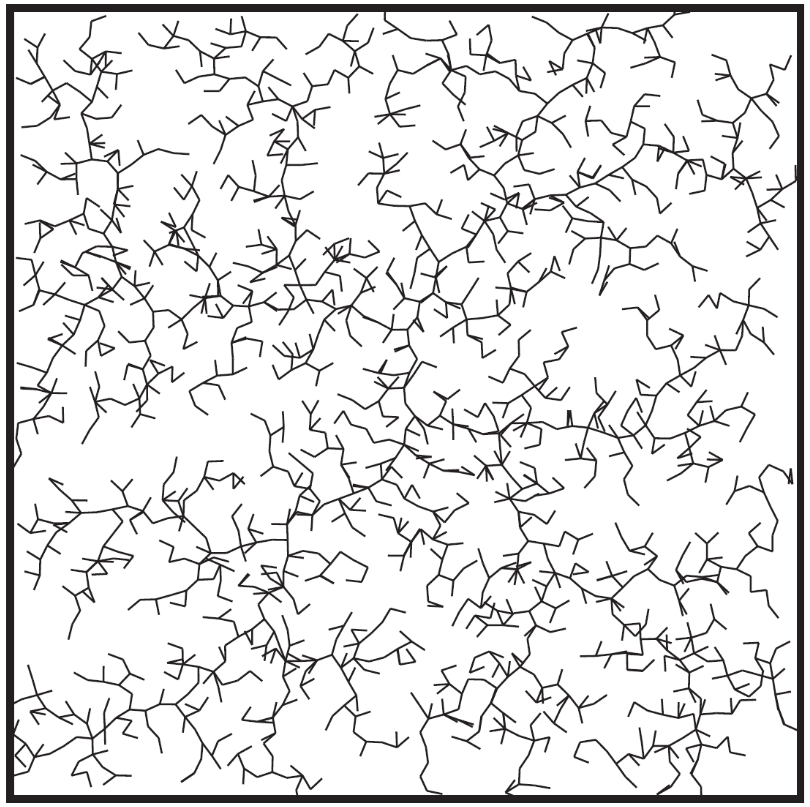
$$x = q$$

- 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}

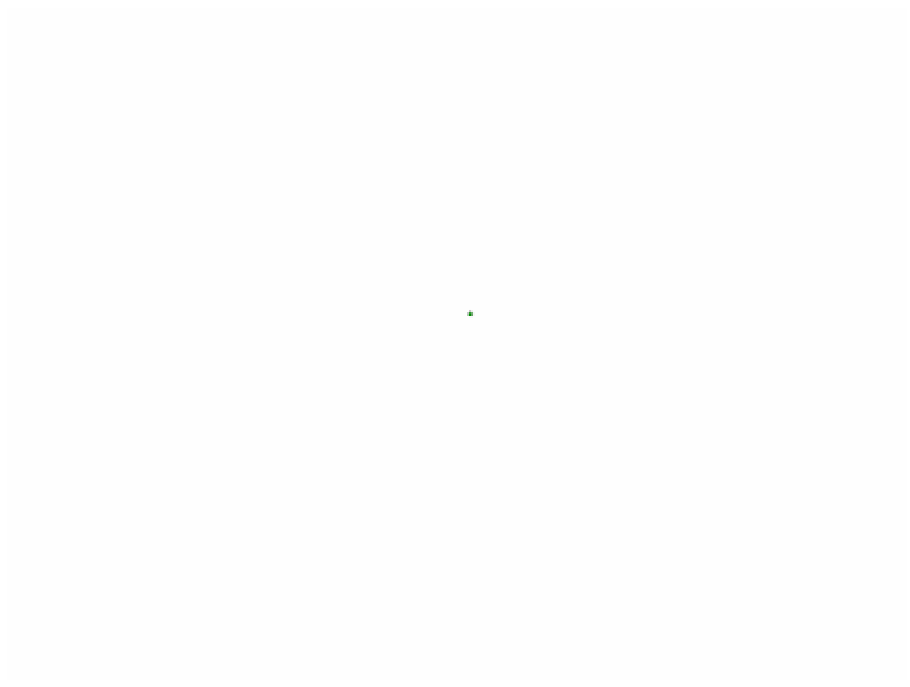
Rapidly exploring Random Trees (RRTs)


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)



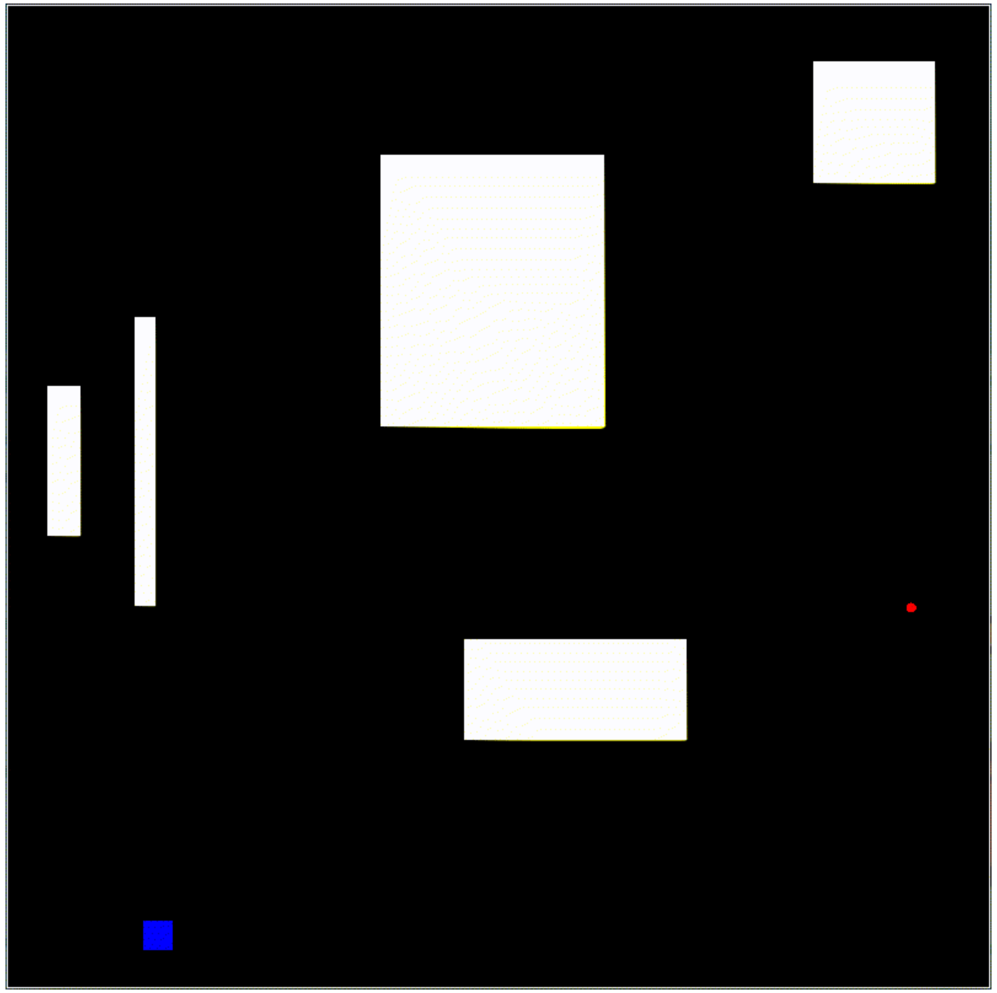
An animation of an RRT starting from iteration 0 to 10000

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

Rapidly exploring Random Trees (RRTs)

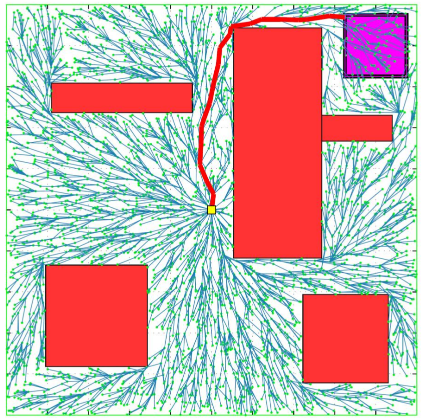
- Bidirectional RRT
  - Grows two trees, one forward from x_{start} , one backward from x_{goal}
  - Alternating between growing the two trees x_{samp}
  - Trying to connect the two trees by choosing x_{goal} from the other tree
  - Con: faster, can reach the exact goal
  - Pro: the local planer might not be able to connect the two trees

Bidirectional RRT



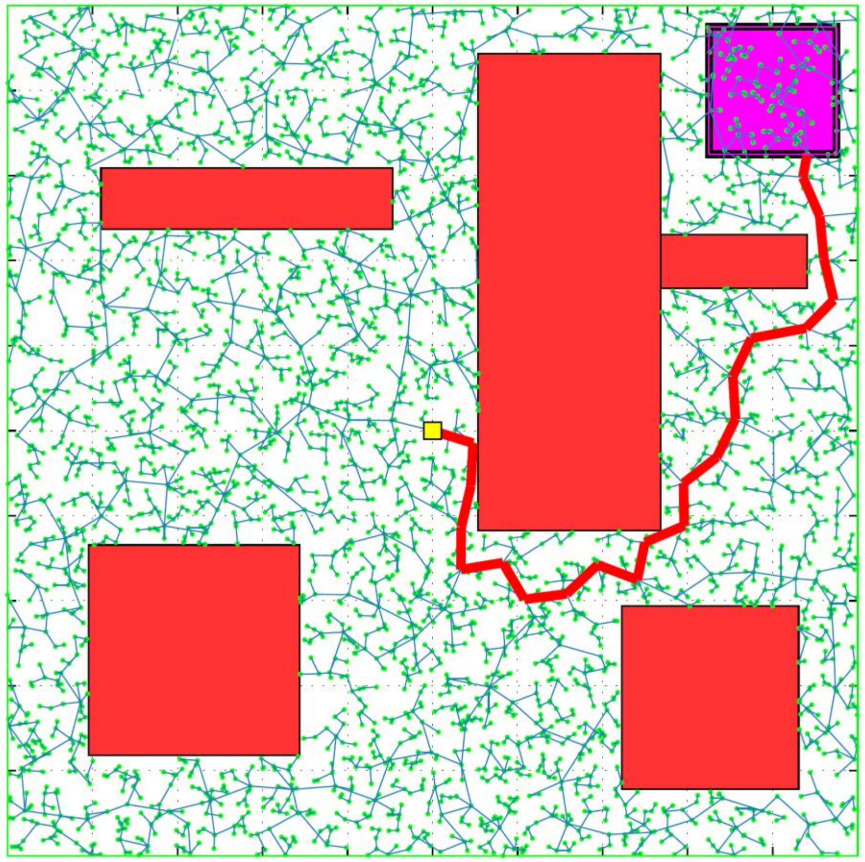
https://github.com/JakeInit/RRT

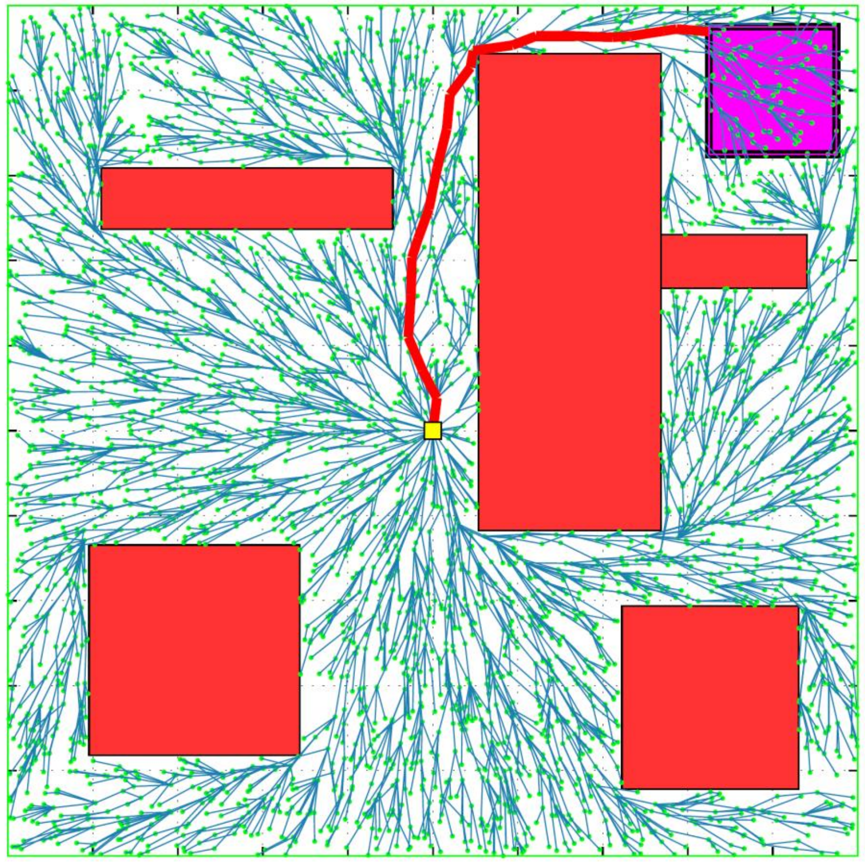
RRT*



- RRT*
  - Continually rewires the search tree to ensure that it always encodes the shortest path from x_{start} to each node in the tree
  - To insert x_{new} to the tree, consider $x \in \mathcal{X}_{\text{near}}$ sufficiently near to x_{new}
    - Collision free
    - Minimizes the total cost from x_{start} to x_{new}
  - Consider each $x \in \mathcal{X}_{\text{near}}$ to see whether it could be reached at lower cost by a motion through x_{new} , change the parent of x to x_{new} (rewiring)

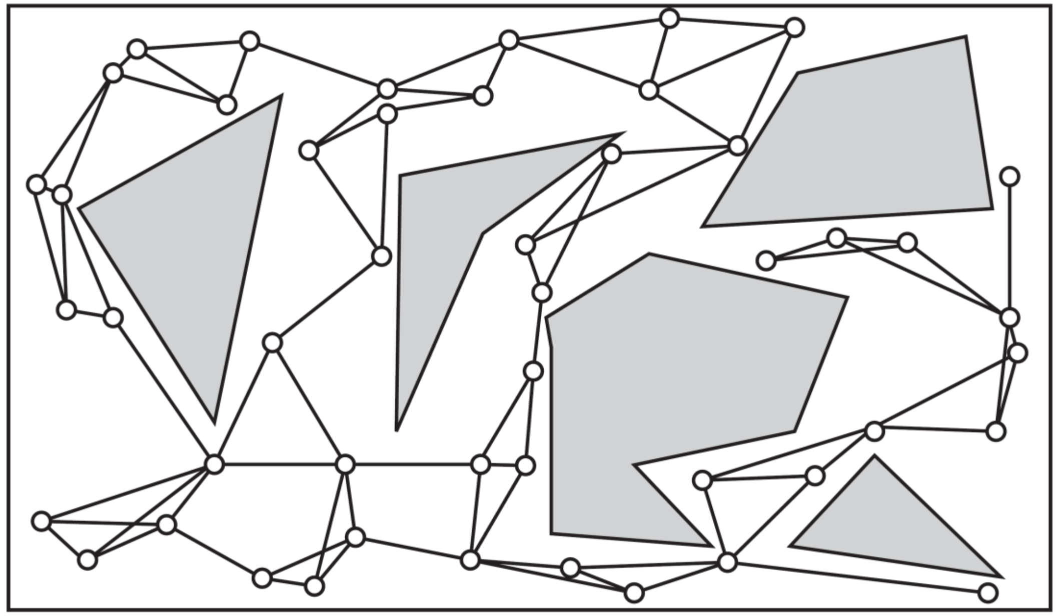
RRT vs. RRT*


RRT


RRT*

Probabilistic Roadmaps (PRMs)

- PRM uses sampling to build a roadmap representation of $\mathcal{C}_{\text{free}}$
- Connect a start node q_{start} and a goal node q_{goal} to the roadmap
- Search for a path, e.g., using A*

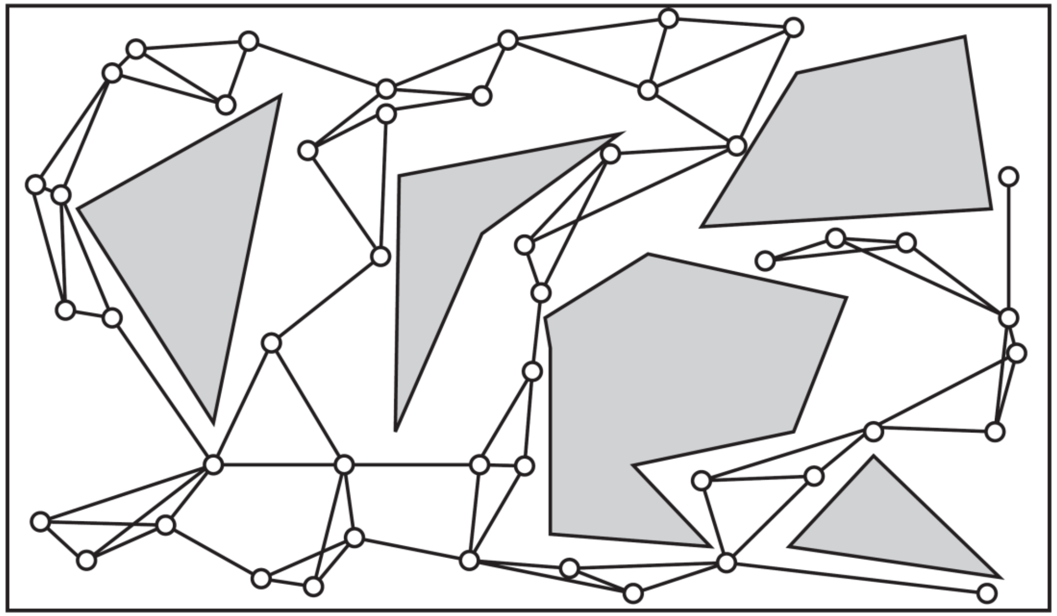


Probabilistic Roadmaps (PRMs)

- PRM uses sampling to build a roadmap representation of $\mathcal{C}_{\text{free}}$

Algorithm 10.4 PRM roadmap construction algorithm (undirected graph).

```
1: for $i = 1, \ldots, N$ do
2:     $q_i \leftarrow$ sample from $\mathcal{C}_{\text{free}}$
3:     add $q_i$ to $R$
4: end for
5: for $i = 1, \ldots, N$ do
6:     $\mathcal{N}(q_i) \leftarrow k$ closest neighbors of $q_i$
7:     for each $q \in \mathcal{N}(q_i)$ do
8:         if there is a collision-free local path from $q$ to $q_i$ and
           there is not already an edge from $q$ to $q_i$ then
9:             add an edge from $q$ to $q_i$ to the roadmap $R$
10:        end if
11:    end for
12: end for
13: return $R$
```



Time Parameterization Algorithms

- By path planning, we have a list of robot configurations $\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_N$

- Time parameterization: How fast can the robot move along this path while respecting **velocity and acceleration limits** — and while ensuring smooth motion?

Iterative Parabolic Time Parameterization (IPTP)

- Inputs
  - A sequence of N + 1 waypoints $\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_N$
  - Joint velocity limits $\dot{q}_i^{\max}$
  - Joint acceleration limits $\ddot{q}_i^{\max}$

- Compute distance between waypoints $\Delta q_i = \mathbf{q}_{i+1} - \mathbf{q}_i$

Path length
$L_i = \|\Delta q_i\|$

- Forward pass: **constant acceleration** to the next waypoint $\dot{q}_{i+1} > \dot{q}_i$

$$\dot{q}_0 = 0 \qquad \dot{q}_{i+1} = \min\left(\dot{q}_{\max}, \sqrt{\dot{q}_i^2 + 2\,\ddot{q}_{\max}\,L_i}\right)$$

constant acceleration motion $v^2 = u^2 + 2as$

Iterative Parabolic Time Parameterization (IPTP)

- Backward pass: deceleration $\dot{q}_{i+1} < \dot{q}_i$

$$\dot{q}_N = 0 \qquad \dot{q}_i = \min\left(\dot{q}_i, \sqrt{\dot{q}_{i+1}^2 + 2\,\ddot{q}_{\max} L_i}\right) \qquad i = N-1, N-2, \ldots, 0$$

- The process repeats until no velocity changes — that is, until all constraints are simultaneously satisfied

```
v
|      _____
|     /     \
|    /       \
|___/         \____
      distance →
```

- Compute Segment Times and accelerations

$$\Delta t_i = \frac{2|\Delta q_i|}{v_i + v_{i+1}} \qquad a_i = \frac{v_{i+1}^2 - v_i^2}{2\,\Delta q_i}$$

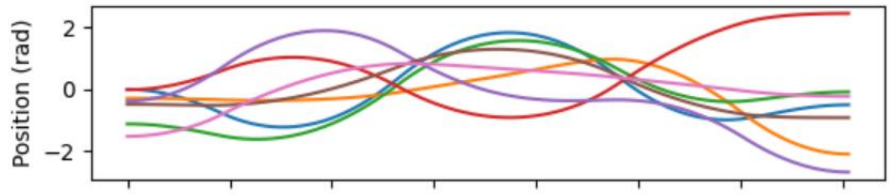
$$a_i = \frac{v_{i+1} - v_i}{\Delta t_i} \qquad \Delta q_i = v_i\,\Delta t_i + \frac{1}{2} a_i\,\Delta t_i^2$$

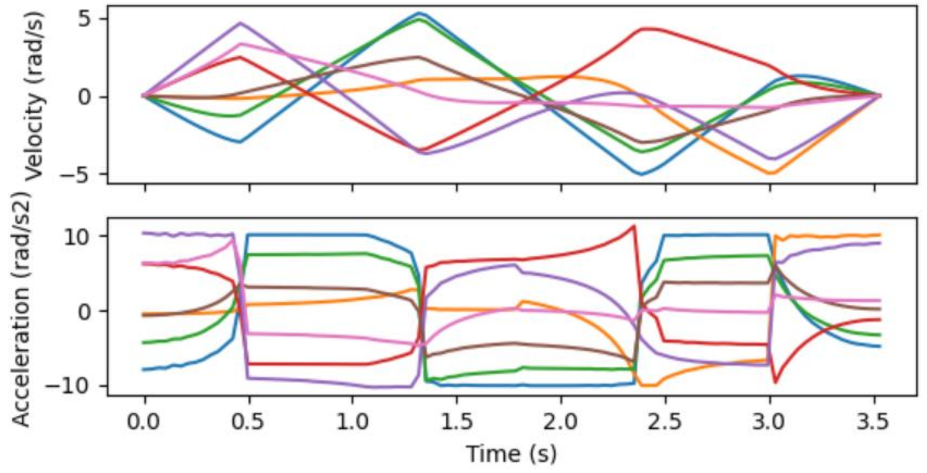
toppra: Time-Optimal Path Parameterization

- A new approach to Time-Optimal Path Parameterization based on Reachability Analysis, *IEEE Transactions on Robotics*, vol. 34(3), pp. 645-659, 2018. https://github.com/hungpham2511/toppra

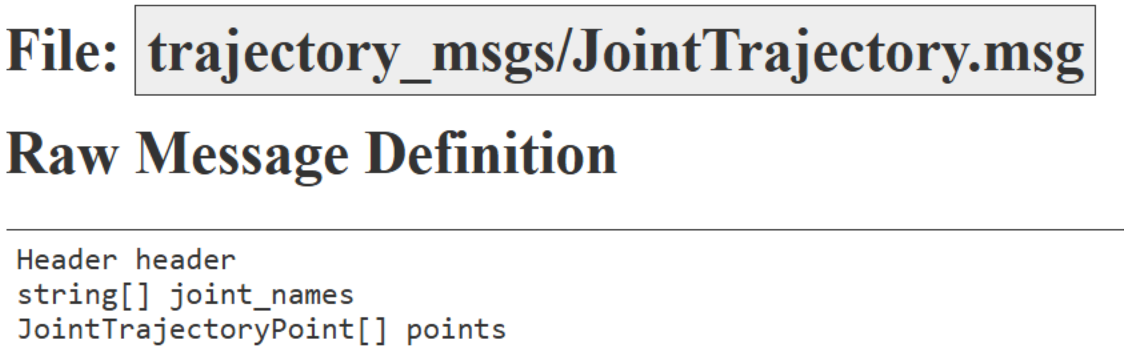
```
>>>   path = ta.SplineInterpolator(ss, way_pts)
>>>   pc_vel = constraint.JointVelocityConstraint(vlims)
>>>   pc_acc = constraint.JointAccelerationConstraint(alims)
>>>   instance = algo.TOPPRA([pc_vel, pc_acc], path)
```

```
jnt_traj = instance.compute_trajectory(0, 0)
```





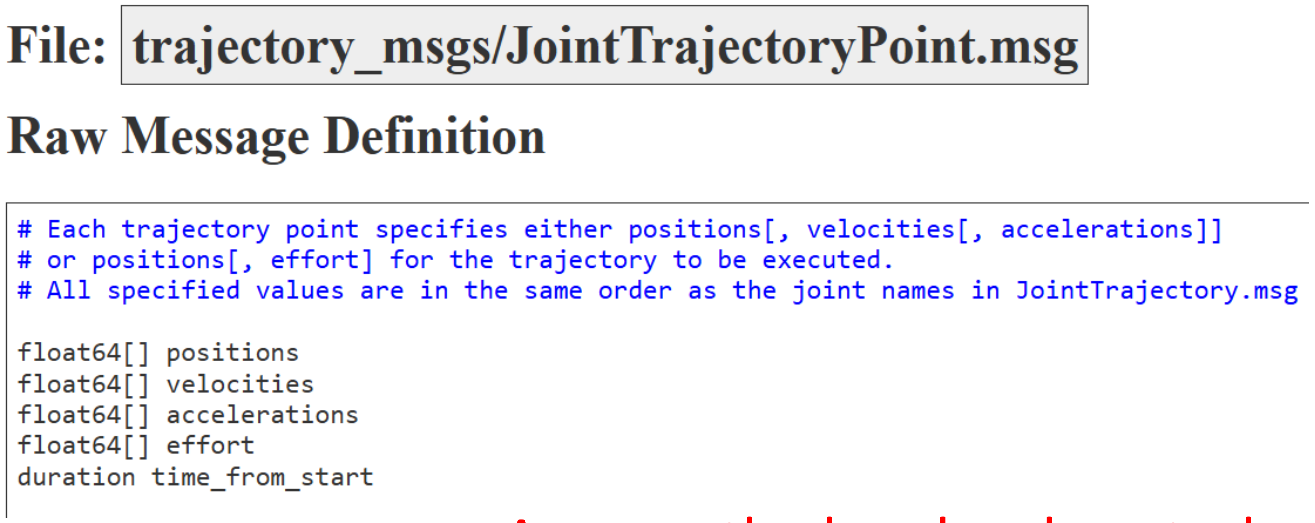
ROS Joint Trajectory



File: trajectory_msgs/JointTrajectory.msg

Raw Message Definition

```
Header header
string[] joint_names
JointTrajectoryPoint[] points
```

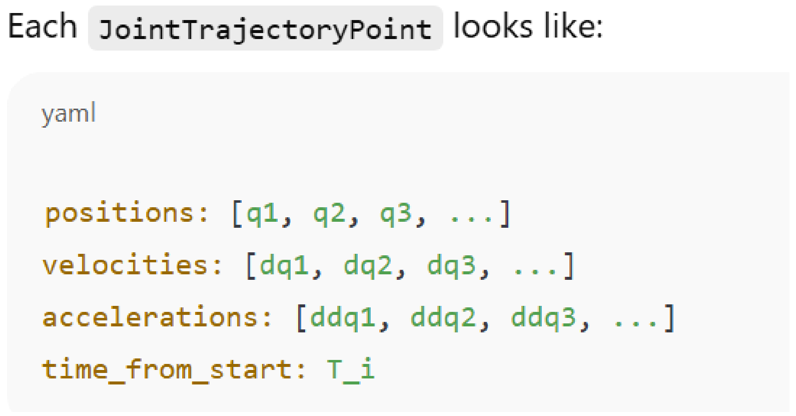


File: trajectory_msgs/JointTrajectoryPoint.msg

Raw Message Definition

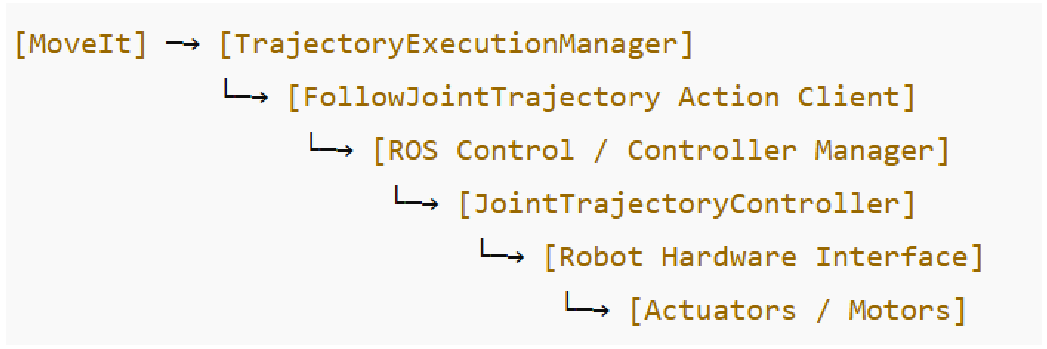
```
# Each trajectory point specifies either positions[, velocities[, accelerations]]
# or positions[, effort] for the trajectory to be executed.
# All specified values are in the same order as the joint names in JointTrajectory.msg

float64[] positions
float64[] velocities
float64[] accelerations
float64[] effort
duration time_from_start
```



Each `JointTrajectoryPoint` looks like:

```yaml
positions: [q1, q2, q3, ...]
velocities: [dq1, dq2, dq3, ...]
accelerations: [ddq1, ddq2, ddq3, ...]
time_from_start: T_i
```

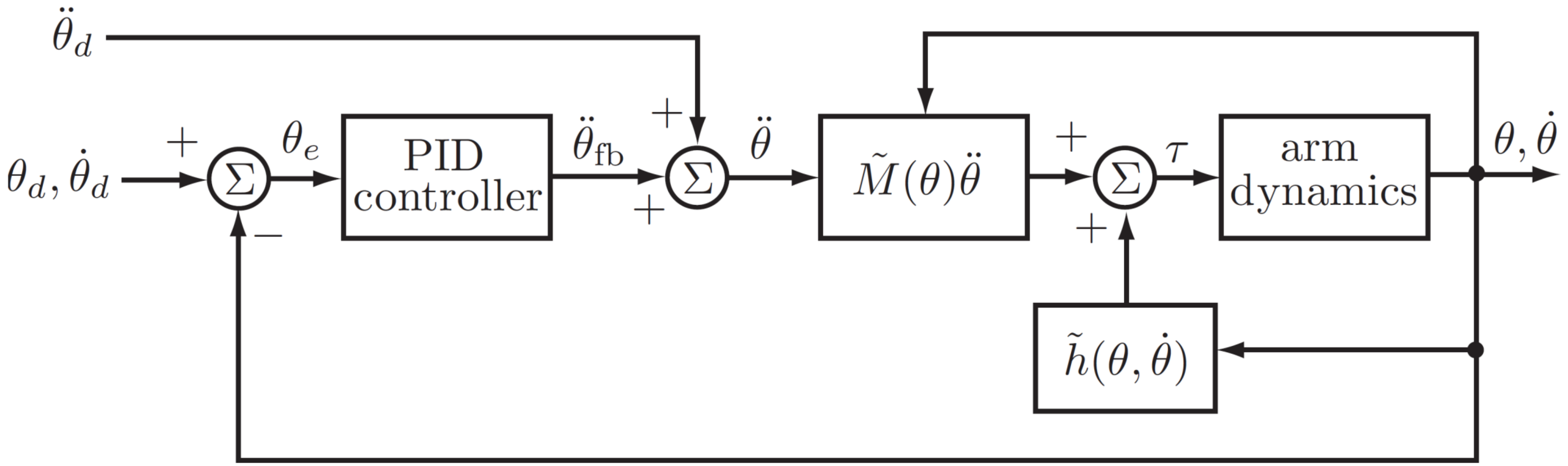


```
[MoveIt] ⟶ [TrajectoryExecutionManager]
              └→ [FollowJointTrajectory Action Client]
                    └→ [ROS Control / Controller Manager]
                          └→ [JointTrajectoryController]
                                └→ [Robot Hardware Interface]
                                      └→ [Actuators / Motors]
```

Assume the low-level control can achieve any acceleration within limit

Feedforward Plus Feedback Linearization

$$\tau = \tilde{M}(\theta)\left(\ddot{\theta}_d + K_p\theta_e + K_i\int\theta_e(\mathrm{t})d\mathrm{t} + K_d\dot{\theta}_e\right) + \tilde{h}(\theta,\dot{\theta})$$



Optimization for Motion Generation

- CuRobo optimization

Reaching an end-effector pose

$$
\begin{aligned}
\underset{\theta_{[1,T]}}{\arg\min} \quad & C_{\text{task}}(X_g, \theta_T) + \sum_{t=1}^{T} C_{\text{smooth}}(\cdot) \\
\text{s.t.,} \quad & \theta^- \leq \theta_t \leq \theta^+, \forall t \in [1,T] \\
& \dot{\theta}^- \leq \dot{\theta}_t \leq \dot{\theta}^+, \forall t \in [1,T] \\
& \ddot{\theta}^- \leq \ddot{\theta}_t \leq \ddot{\theta}^+, \forall t \in [1,T] \\
& \dddot{\theta}^- \leq \dddot{\theta}_t \leq \dddot{\theta}^+, \forall t \in [1,T] \\
& \dot{\theta}_T, \ddot{\theta}_T, \dddot{\theta}_T = 0 \\
& C_r(K_s(\theta_t)) \leq 0, \forall t \in [1,T] \\
& C_w(K_s(\theta_t)) \leq 0, \forall t \in [1,T]
\end{aligned}
$$

Self-collision

Workspace collision

Dynamic Motion Planning

- The general motion planning problem

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

Smoothing cost function

$$J = \frac{1}{2} \int_0^T \dot{u}^{\mathrm{T}}(t)\dot{u}(t)dt$$

Nonlinear Optimization

Kinodynamic RRT

“Kino” → *Kinematics*

“Dynamic” → *Dynamics*

Algorithm Kinodynamic-RRT

```
1. $T \leftarrow \{x_0\}$.
2. for $i = 1, \ldots, N$ do
3.     $x_{rand} \leftarrow Sample()$
4.     $x_{near} \leftarrow Nearest(T, x_{rand})$
5.     $u_e \leftarrow$Choose-Control$(x_{near}, x_{rand})$
6.     $x_e \leftarrow$Simulate$(x_{near}, u_e)$
7.     if the path traced out from $x_{near}$ to $x_e$ is collision-free, then
8.         Add edge $x_{near} \rightarrow x_e$ to $T$
9.     if $x_e \in G$ then
10.        return the path in $T$ from $x_0$ to $x_e$
11. return "no path"
```

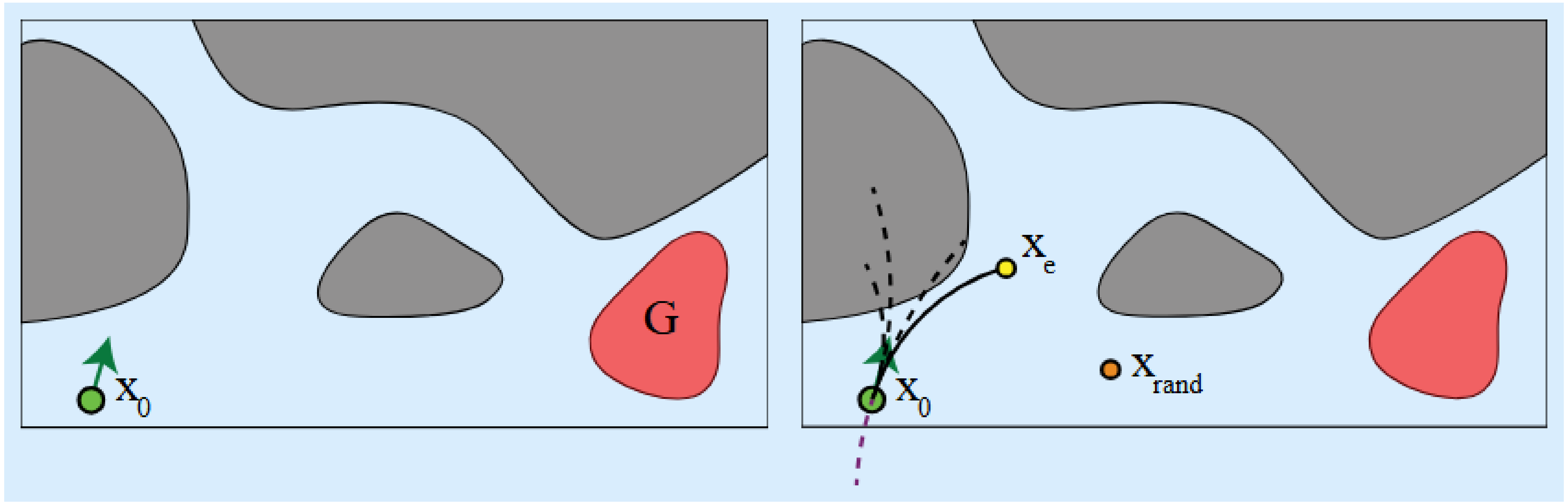
Choose-control: random sampling a few candidate controls and finding the one that gets the closest

Forward dynamics

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

https://motion.cs.illinois.edu/RoboticSystems/PlanningWithDynamicsAndUncertainty.html

Kinodynamic RRT



Summary

- Sampling methods
  - RRT, Bidirectional RRT, RRT*
  - PRMs

- Time Parameterization Algorithms
  - Iterative Parabolic Time Parameterization (IPTP)

- Dynamic Motion Planning
  - Kinodynamic RRT

Further Reading

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