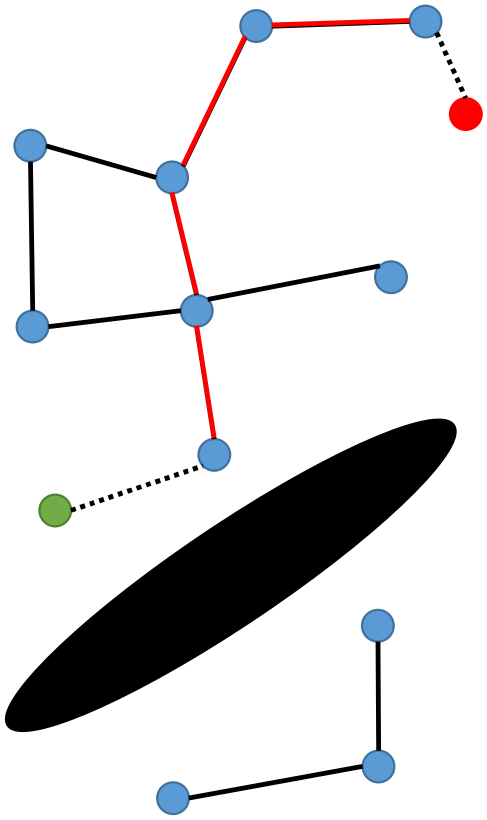
EECE 5550: Mobile Robotics

Lecture 16: Motion planning

Plan of the day (☺)

Last time: Planning as search
- Definition of planning problems & solutions
- Planning as graph search
- Graph search algorithms

Today: Application to robot motion planning (yay ☺!)
- Robot workspaces & configuration spaces
- Grid-based motion planners
- Sampling-based planners for high-dimensional spaces
  - Probabilistic road maps (PRMs)
  - Rapidly-exploring random trees (RRTs)



References

Lecture "Planning II" from ETH Zurich's Autonomous Mobile Robots course

Classic (and very beautiful!) papers:

- "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces"
- "Randomized Kinodynamic Planning"

Workspace and configuration spaces

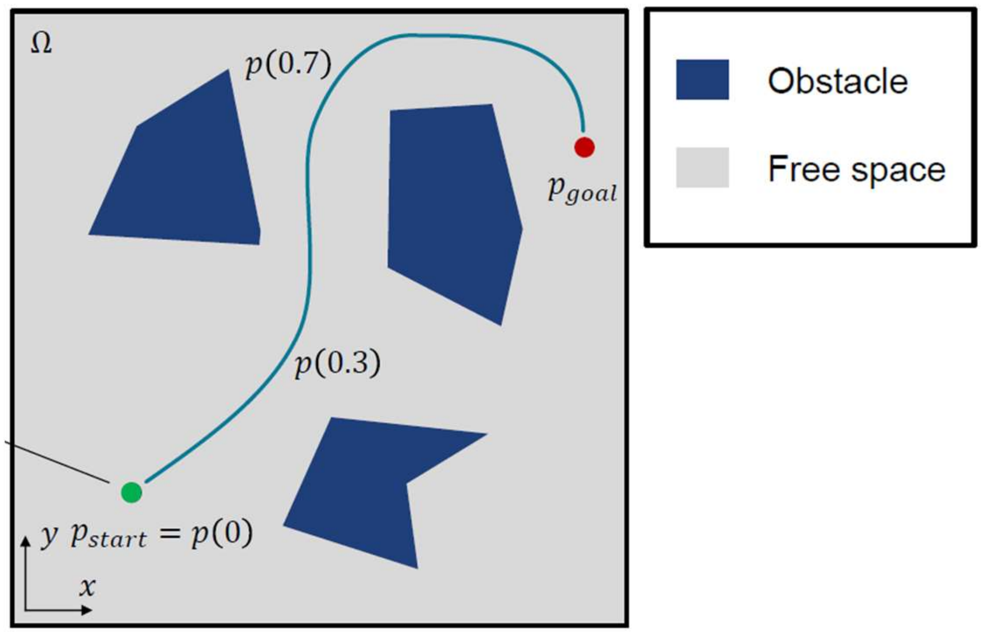
Workspace: The environment in which the robot is operating.

Often modeled at the level of *free* and *occupied* space

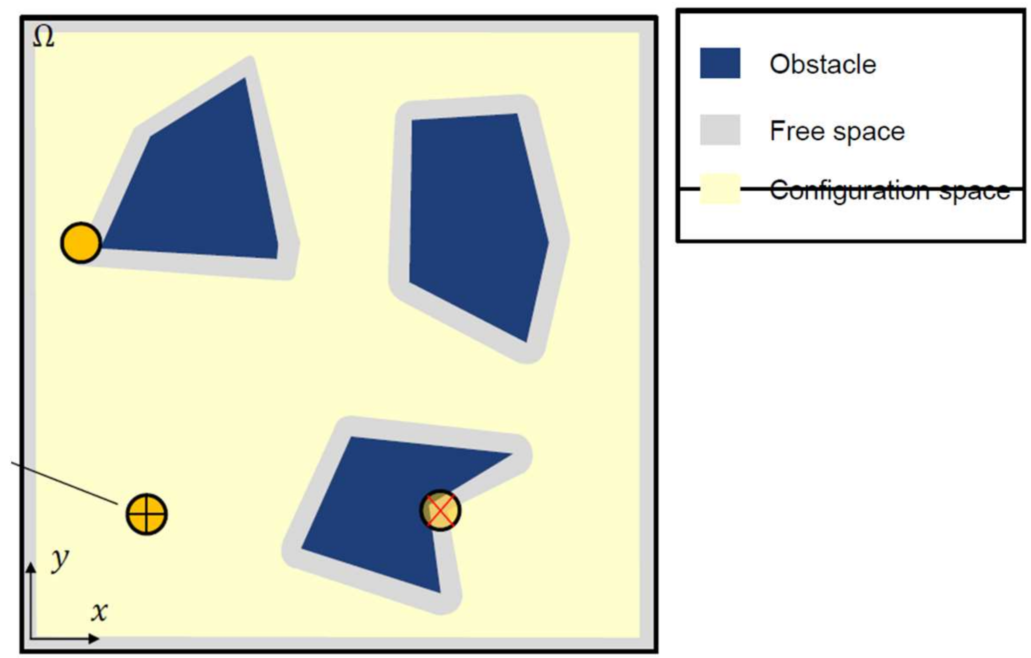
Configuration space: The set of feasible *robot states*

- Partially determined by workspace (no collisions permitted)
- For "interesting" (i.e. non-point) robots, also includes:
  - Orientations
  - Actuator limits …

Workspace and configuration spaces

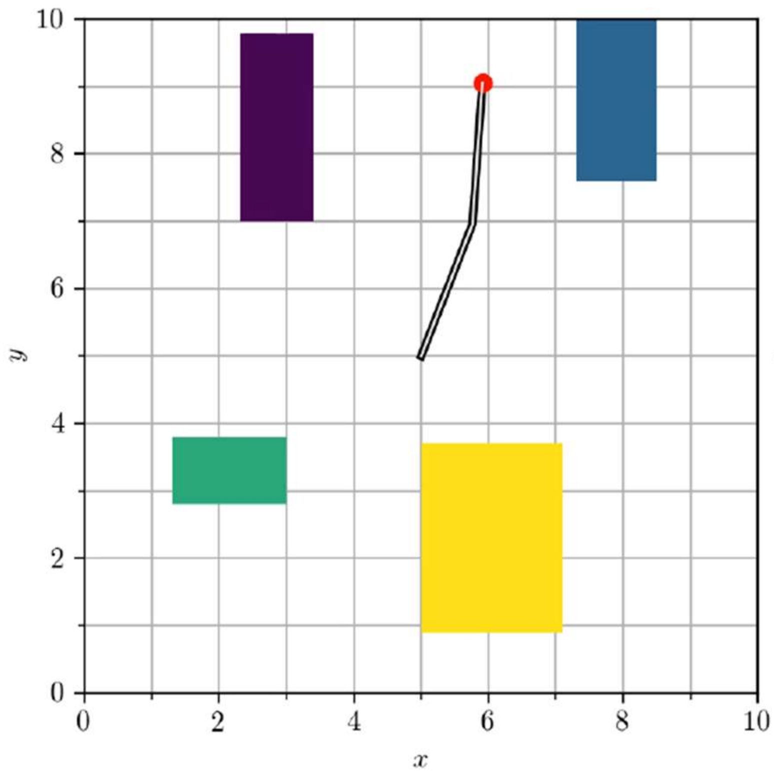


Configuration space for a point robot

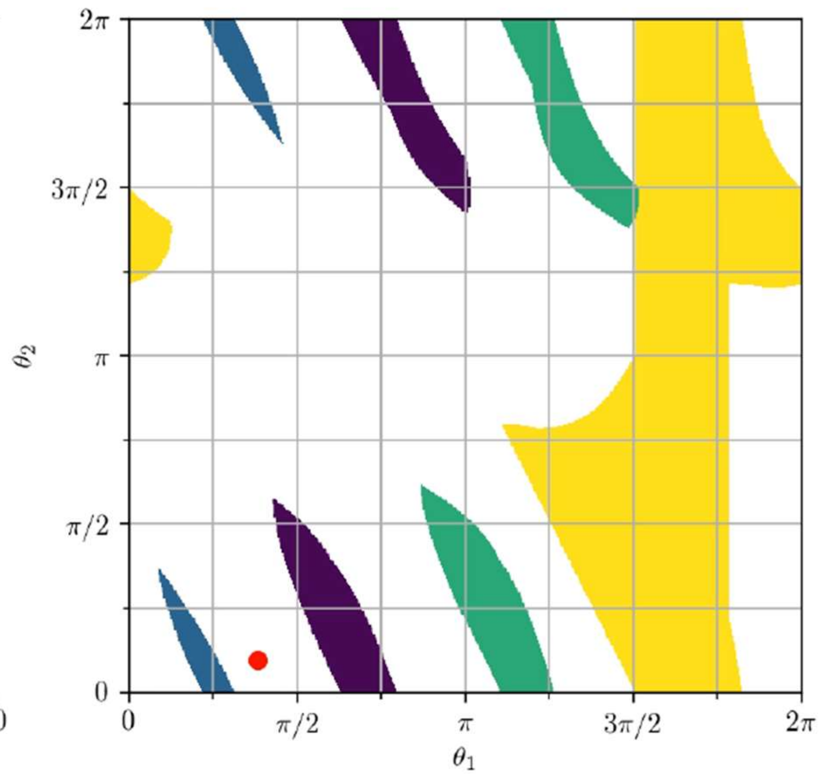


Configuration space for a disc robot

Workspace and configuration space: two-link arm



Workspace



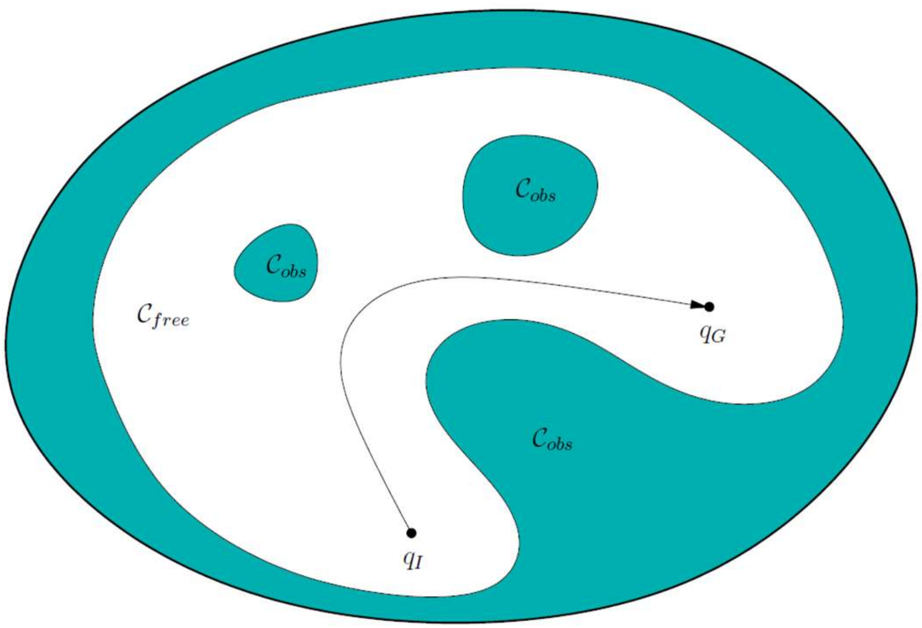
Configuration space

Robot motion planning

Given:

- Workspace W, partitioned into *free* and *occupied* subsets
- Robot *configuration space* C, partitioned into corresponding *free* and *occupied* subsets
- Initial configuration $q_I \in C_{free}$
- Goal configuration $q_G \in C_{free}$

Find: A *path* $\tau: [0,1] \rightarrow C_{free}$ such that $\tau(0) = q_I$ and $\tau(1) = q_G$

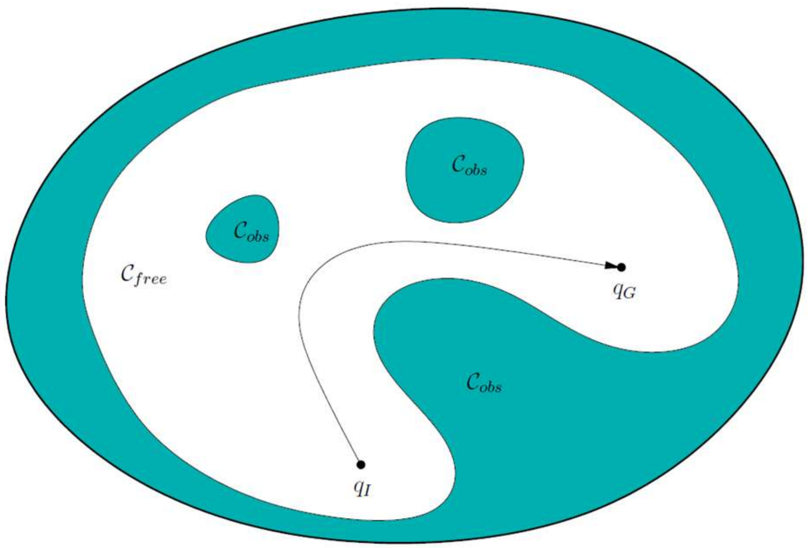


Robot motion planning: Challenges

Basic challenge: Workspaces and robot configuration spaces are generally *continuous*

⇒ **Very** hard to model arbitrary continuous shapes (Fun fact: these form *infinite-dimensional* spaces.)

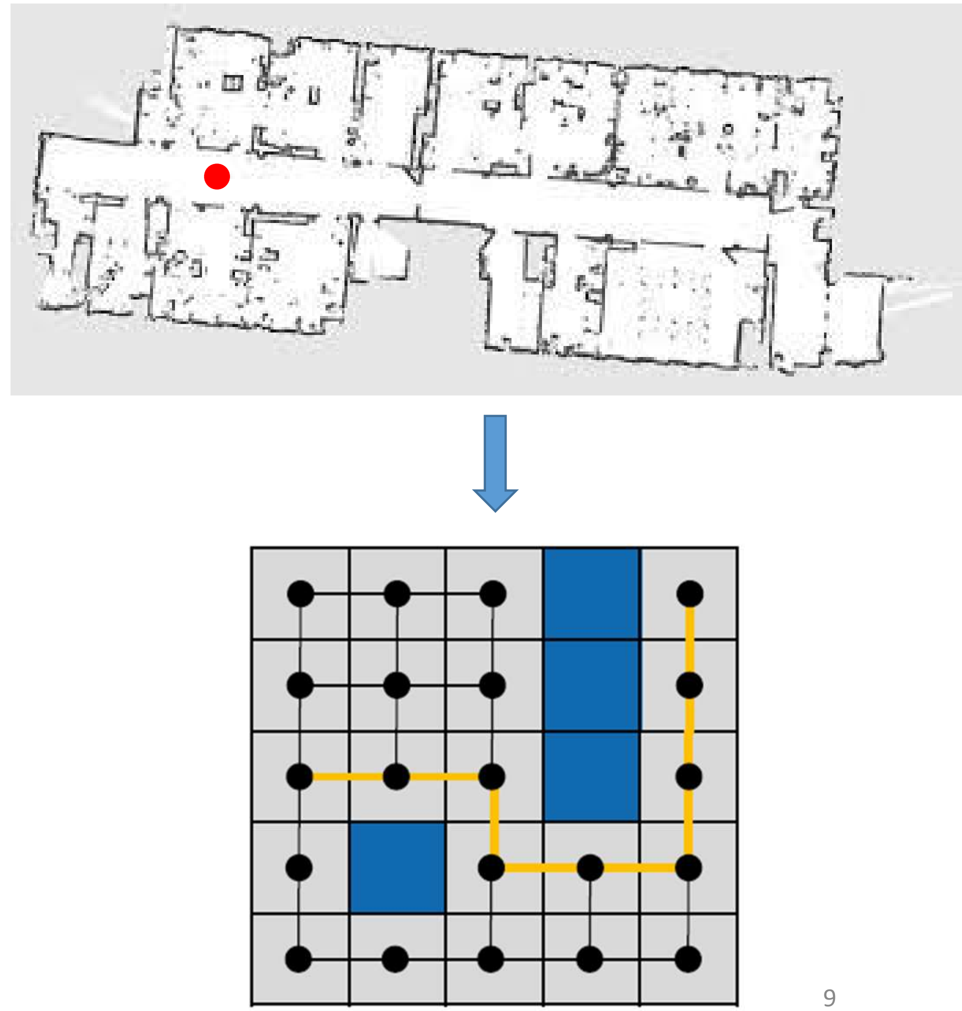
One natural approach: Discretize the configuration space to get a *finite approximation*, and then solve the problem via *search*.



Simple case: route planning for a planar robot

Recall:

- Occupancy grids already provide a discrete model of free and occupied space
- For a *point* robot, C_{free} is just the set of unoccupied cells
- For *disc* robot, can also "grow" occupied cells to account for the body



Now: Construct a graph $G = (V, E)$ where:

- Vertices are free cells
- Edges model connectivity between free cells

Then: Motion planning is just graph search!

Grid-based representations

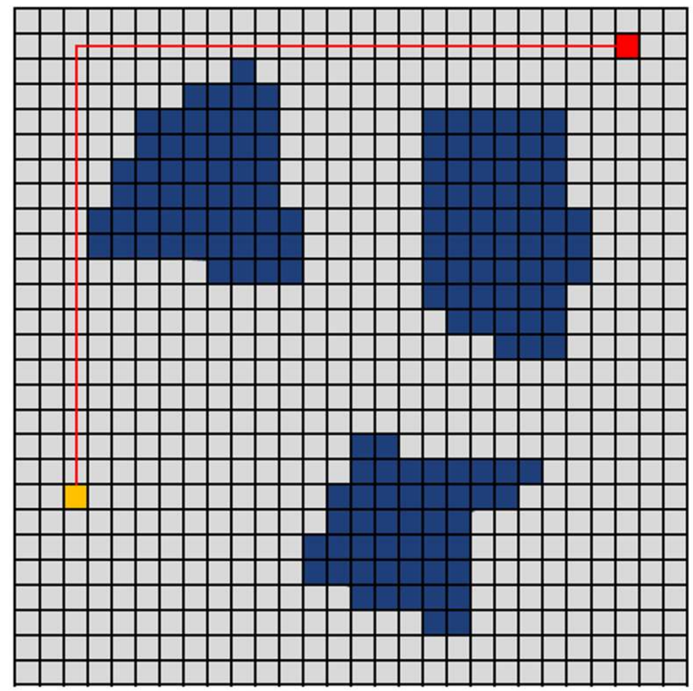
More generally, one can “voxelize” any *generic* configuration space C_{free}, and then apply the same graph search strategy

Pros:

- Very simple idea
- Easy to implement
- Resolution completeness: If there is a feasible plan, this approach is *guaranteed* to find it as grid resolution *r* -> 0.

Con:

- Voxelization can introduce weird artifacts
- Not always clear how to choose resolution *r*: how small is “small enough”??
- Curse of dimensionality: Number of cells *N* in the voxel grid grows as $O(r^{-d})$!



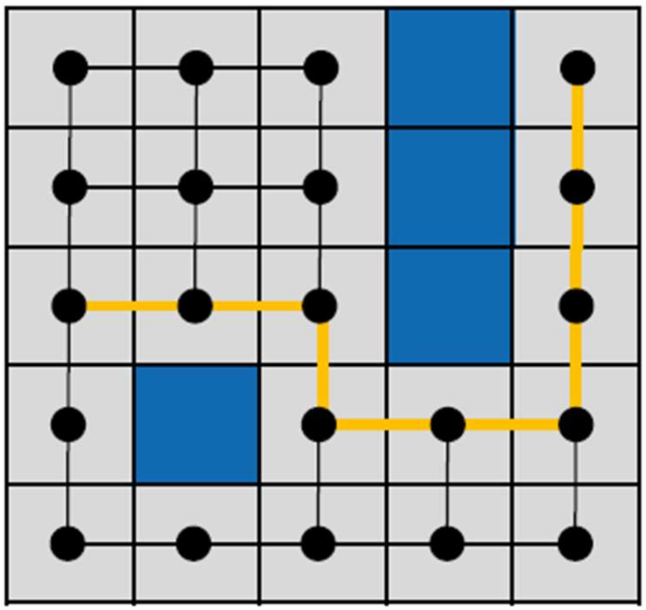
Sampling-based planners

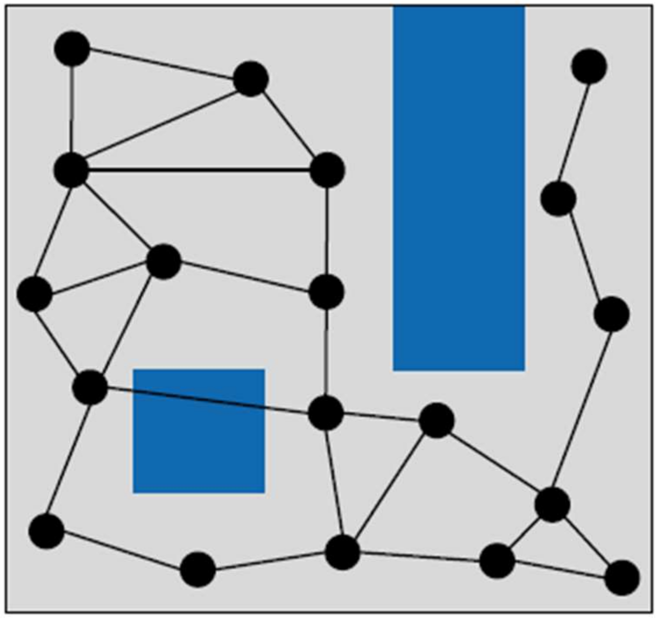
Main idea: Rather than trying to capture *every* point in the configuration space *C* via voxelization, let's *randomly sample* a *representative set S* of points in C_{free}

Then:

- $S \subset C_{free}$ provides a *sample-based* inner approximation of C_{free}
- **NB:** It's often easy to plan feasible motions between two *nearby* points $x, y \in S$ (E.g.: a straight-line path often suffices …)
- If we draw an edge *e* between two points $x, y \in S$ whenever we can *locally* plan a feasible path from one to the other, we get a graph $G = (V, E)$ that models *reachability*

Grid- vs. sample-based planners





Sampling-based planning: General framework

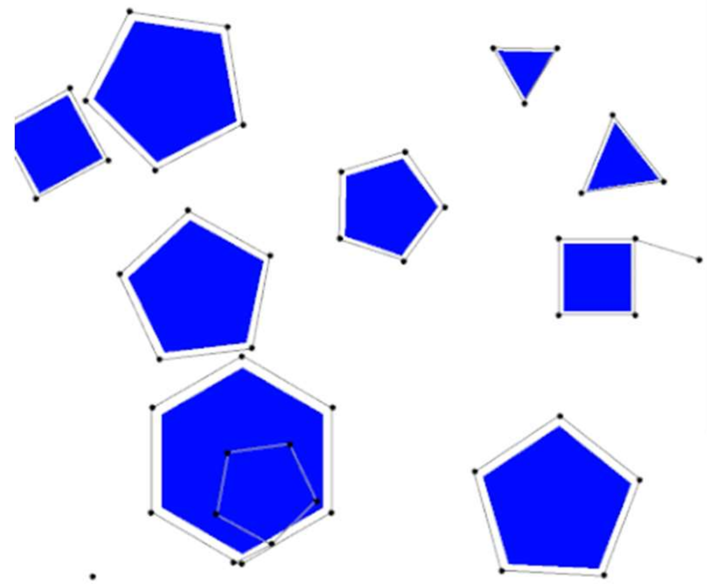
Starting with an empty graph G, repeat:

1. Sample a random point $x \in C_{free}$ and add to G
   [**Q:** How to sample *x*?]

2. For each vertex $y \in G$ s.t. $d(x,y) < \epsilon$, try to plan a path from *x* to *y*

   NB: This requires:
   - A suitable notion of *distance* for *C*
   - A *fast local planner*

3. If a feasible path is found, add edge (x,y) to G.

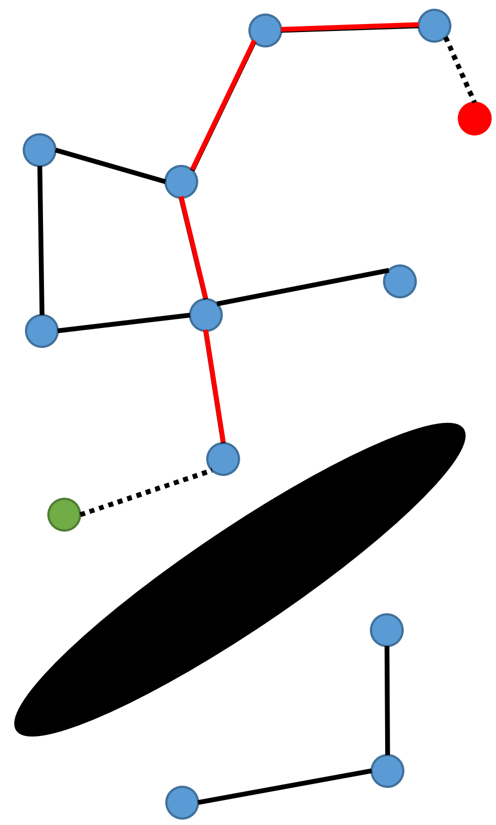


The Probabilistic Roadmap (PRM) Algorithm

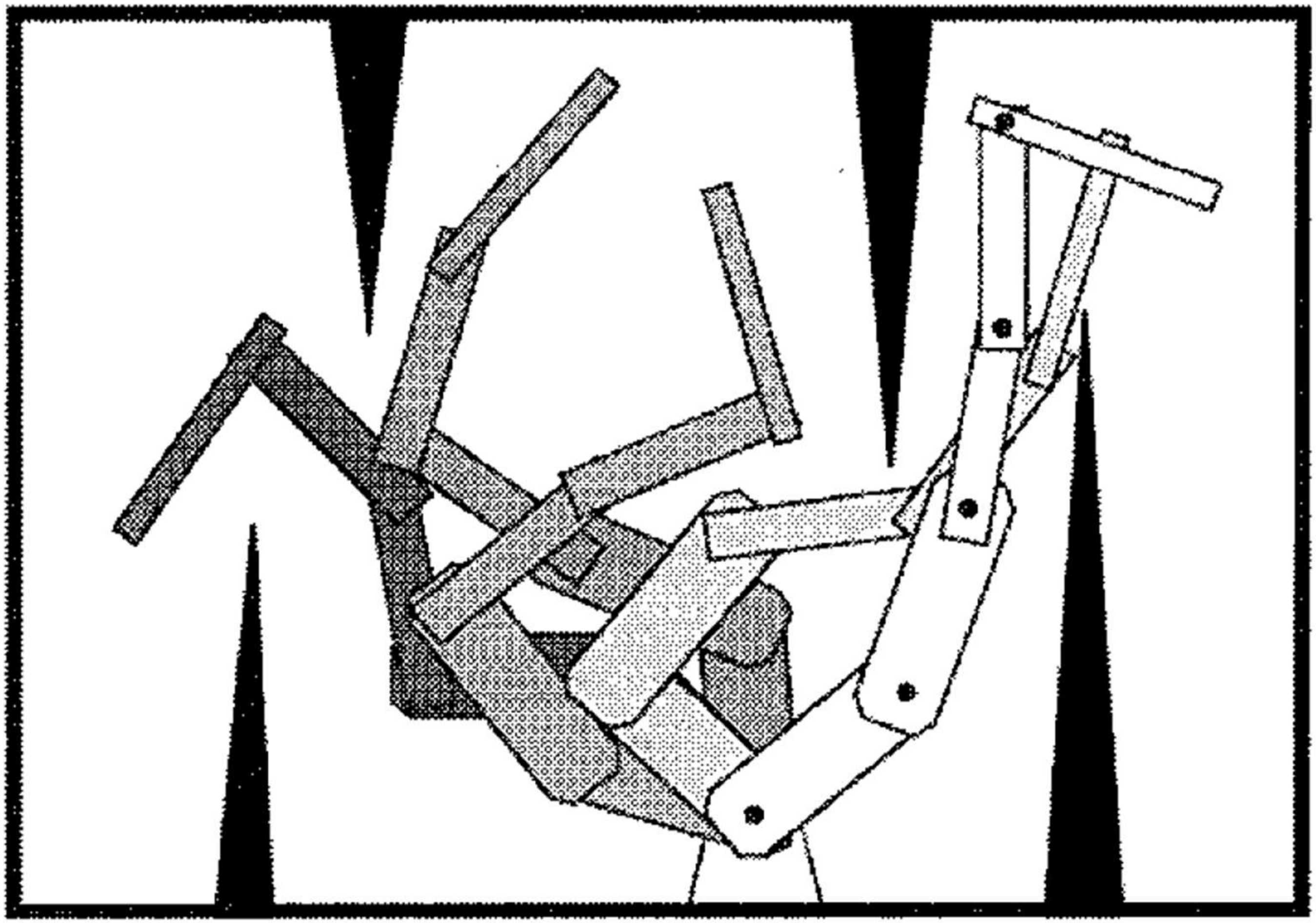
Two-phase algorithm for sampling-based planning:

1. **Construction:** Build a roadmap (graph) by randomly sampling points over the *entire* configuration space and planning local paths

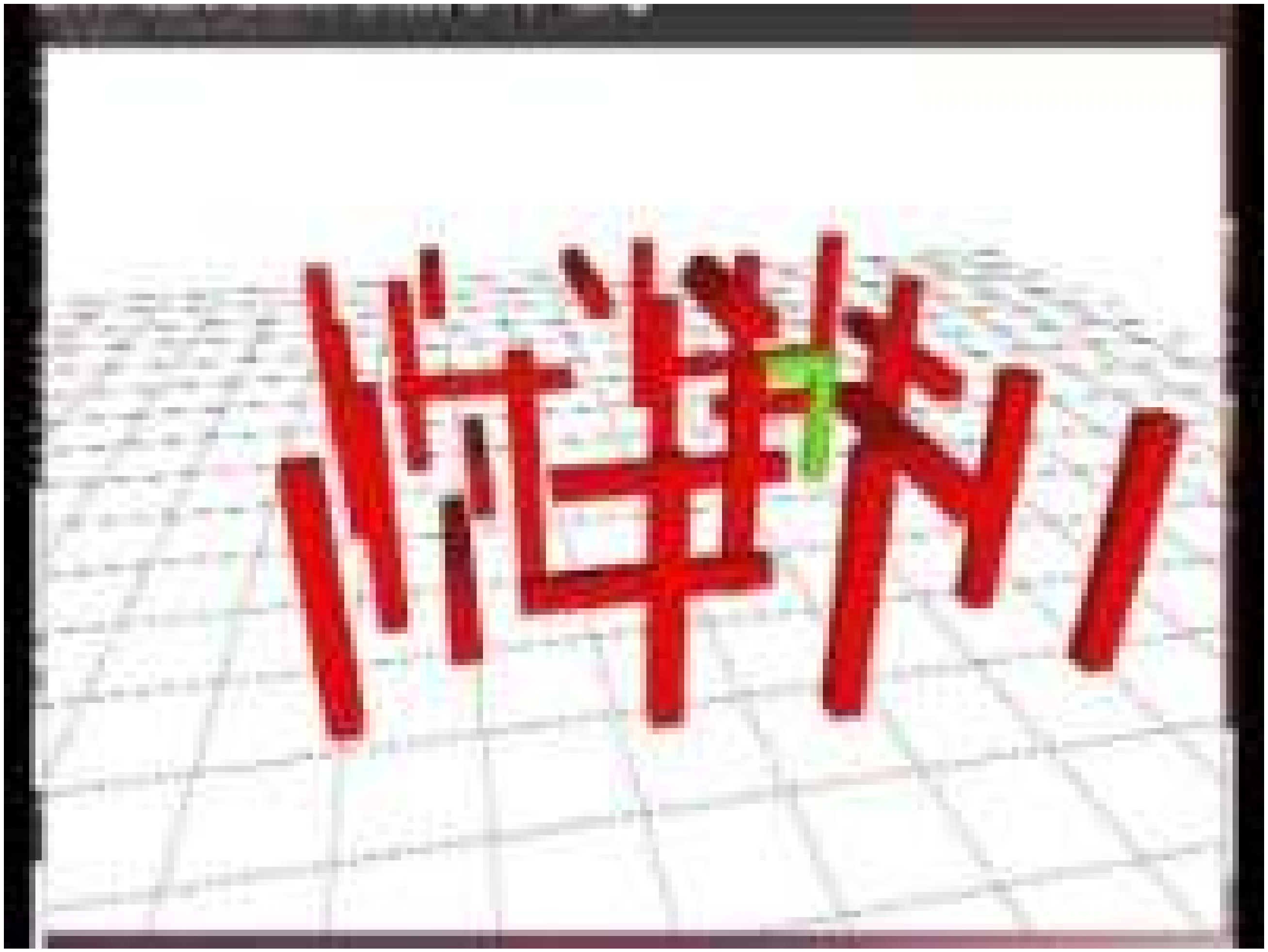
2. **Query:** At run-time, given initial point $x \in G$ and goal $y \in G$:
   - Plan local paths from *x* and *y* to (nearby) vertices in the *same connected component* of *G*
   - Graph search!



Example: PRM planning with a 5 DOF robotic arm



Example: Solving the piano mover's problem with PRMs



Probabilistic Roadmaps: Key Properties

Recap: Builds a roadmap (graph) G over the *entire* space C_{free} by joining a *sparse sample set* using *local planning*

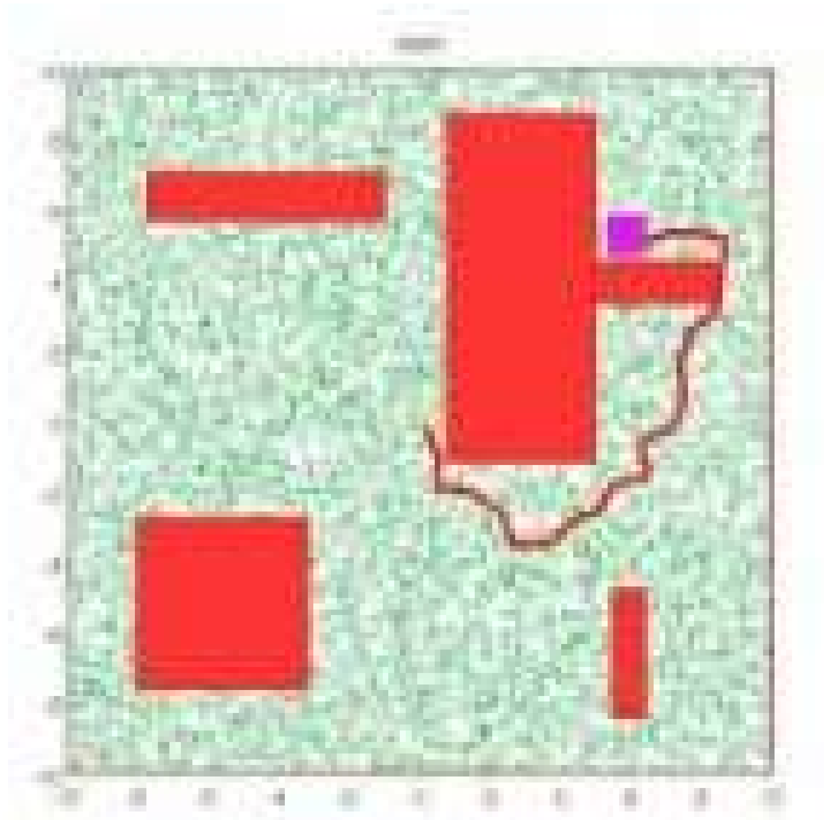
- **Much** more scalable than grid methods
  - We sparsely (inner) approximate true space C_{free} using sample-set S
  - Works well for high degree-of-freedom robots (e.g. robot arms, etc.)
- Probabilistically complete: If a feasible plan exists, the probability of finding it approaches 1 as the number of samples grows.
- Convenient "anytime" flavor: We can stop building the graph whenever we want, and we still get something useful
- Reusable! Enables fast multi-query planning

BUT: Costs a lot up-front (we build the roadmap G over the *entire* space C_{free})

Q: Can we get something faster in the single-query case?

Rapidly-exploring Random Trees (RRTs)

Main idea: Instead of building a *graph* by sampling vertices *uniformly* over C_{free}, we build a tree outwards from the initial state *x* towards the goal *y*.



RRT algorithm

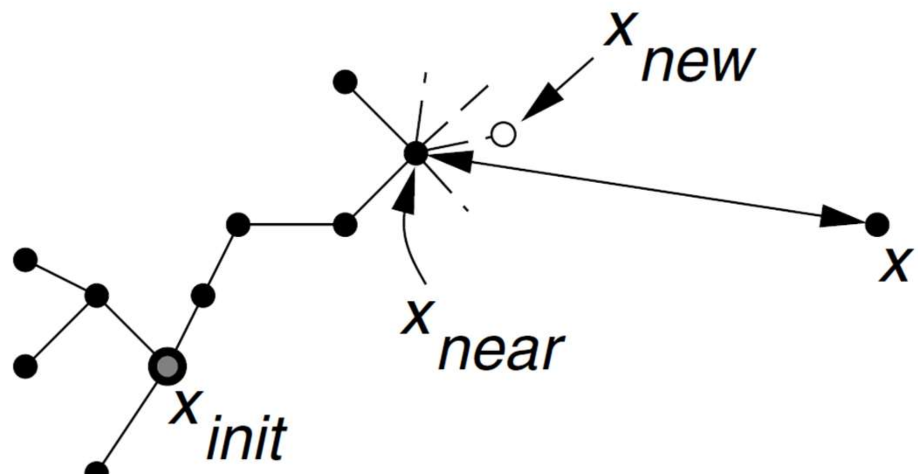
BUILD_RRT(x_{init})

$\mathcal{T}$.init(x_{init});
for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE();
    EXTEND($\mathcal{T}, x_{rand}$);
Return $\mathcal{T}$

EXTEND($\mathcal{T}, x$)

$x_{near} \leftarrow$ NEAREST_NEIGHBOR($x, \mathcal{T}$);
if NEW_STATE($x, x_{near}, x_{new}, u_{new}$) **then**
    $\mathcal{T}$.add_vertex(x_{new});
    $\mathcal{T}$.add_edge($x_{near}, x_{new}, u_{new}$);
    if $x_{new} = x$ **then**
        Return *Reached*;
    else
        Return *Advanced*;
Return *Trapped*;

Fig. 5. Basic rapidly exploring random tree construction algorithm.



EXTEND

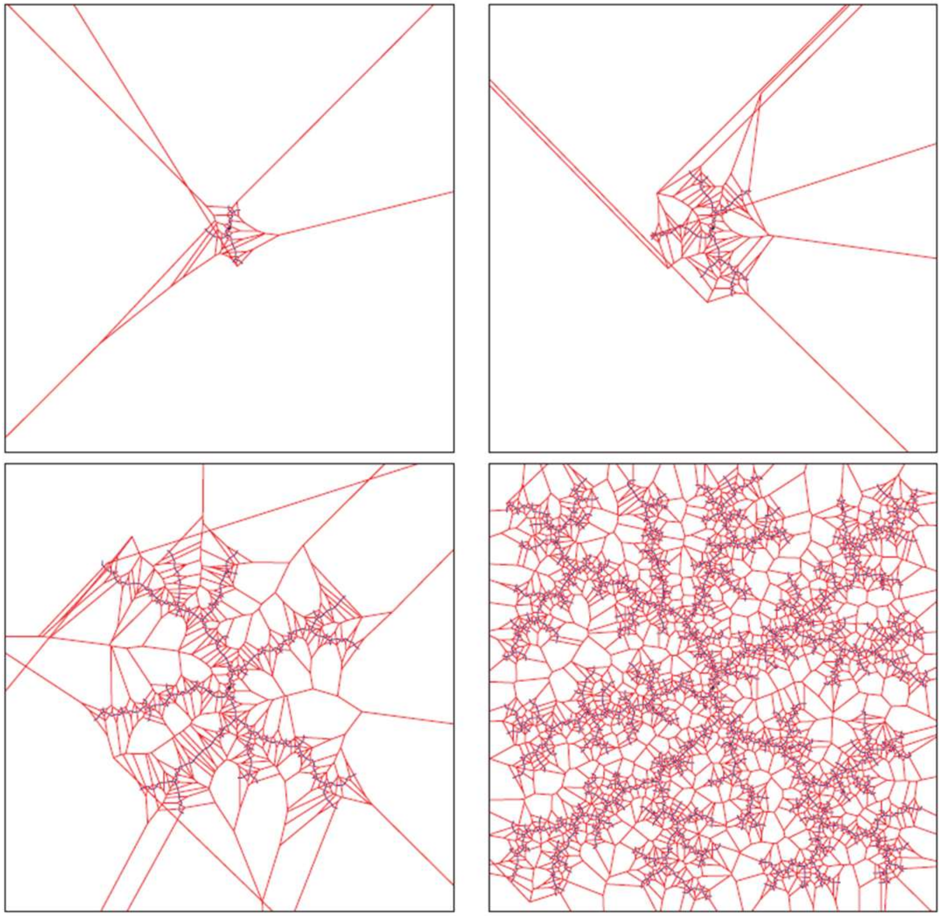
NEW_STATE: f_{new} is gotten from x_{near} and u_{new} by forward simulation using system dynamics:

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

Key point: Unlike PRM, in RRT we *do not* require that the control u_{new} drives x_{near} to *x*; only that it *makes progress towards* *x*

What makes RRTs "rapidly exploring"?

Voronoi bias: Random sampling tends to place new vertices in larger Voronoi cells (⇒ unexplored regions)



RRT: Key Properties

- Probabilistically complete

- Single query & directed
  - Edges in RRTs are directed: travel is from the initial location towards the goal (x -> y $\neq$ y -> x)
  - RRTs retain control information u in their edges
  - **Key payoff:** Unlike PRMs, RRTs can easily handle dynamic constraints:

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

In fact, RRTs were explicitly developed for *kinodynamic planning* (planning w/ kinematic and dynamic constraints)

⇒Often applied to planning in *phase space* (position *and* velocity)

Planning with phase space as configuration space

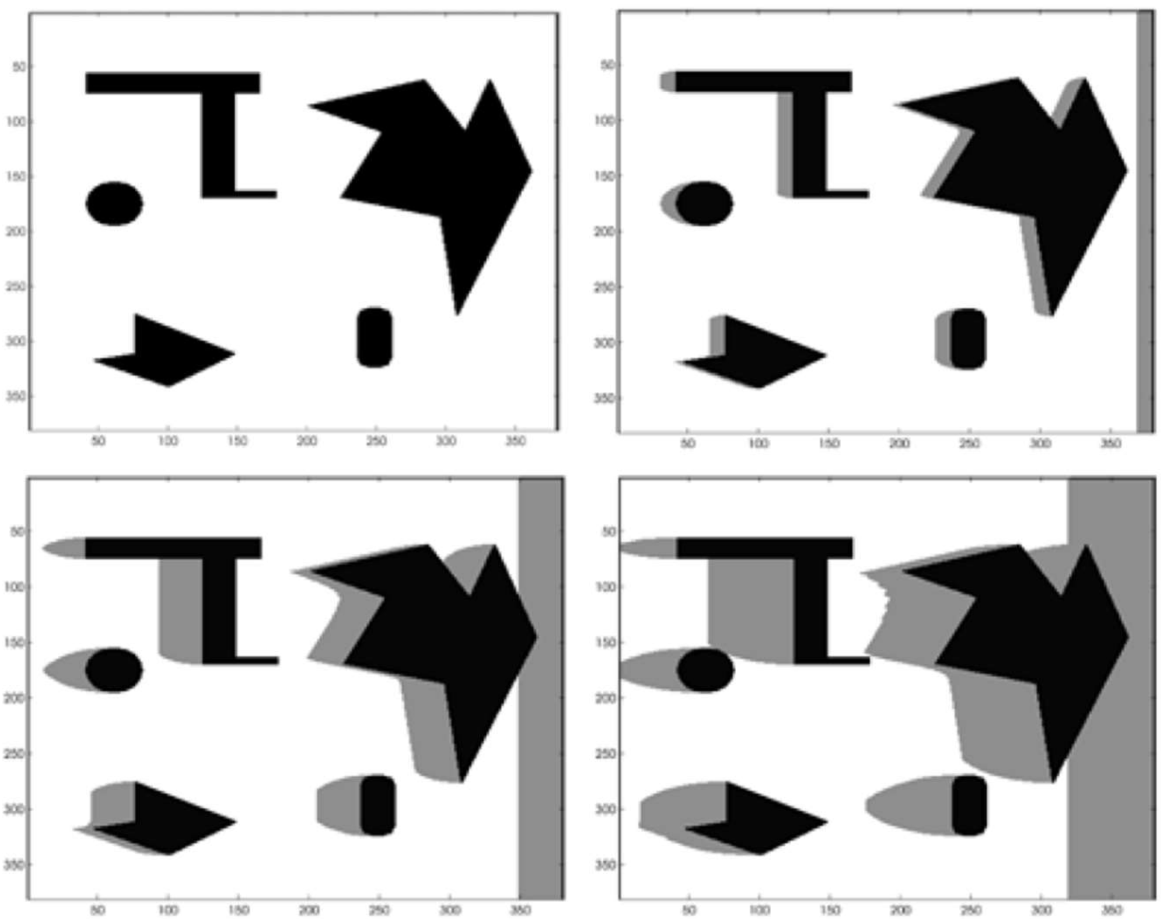
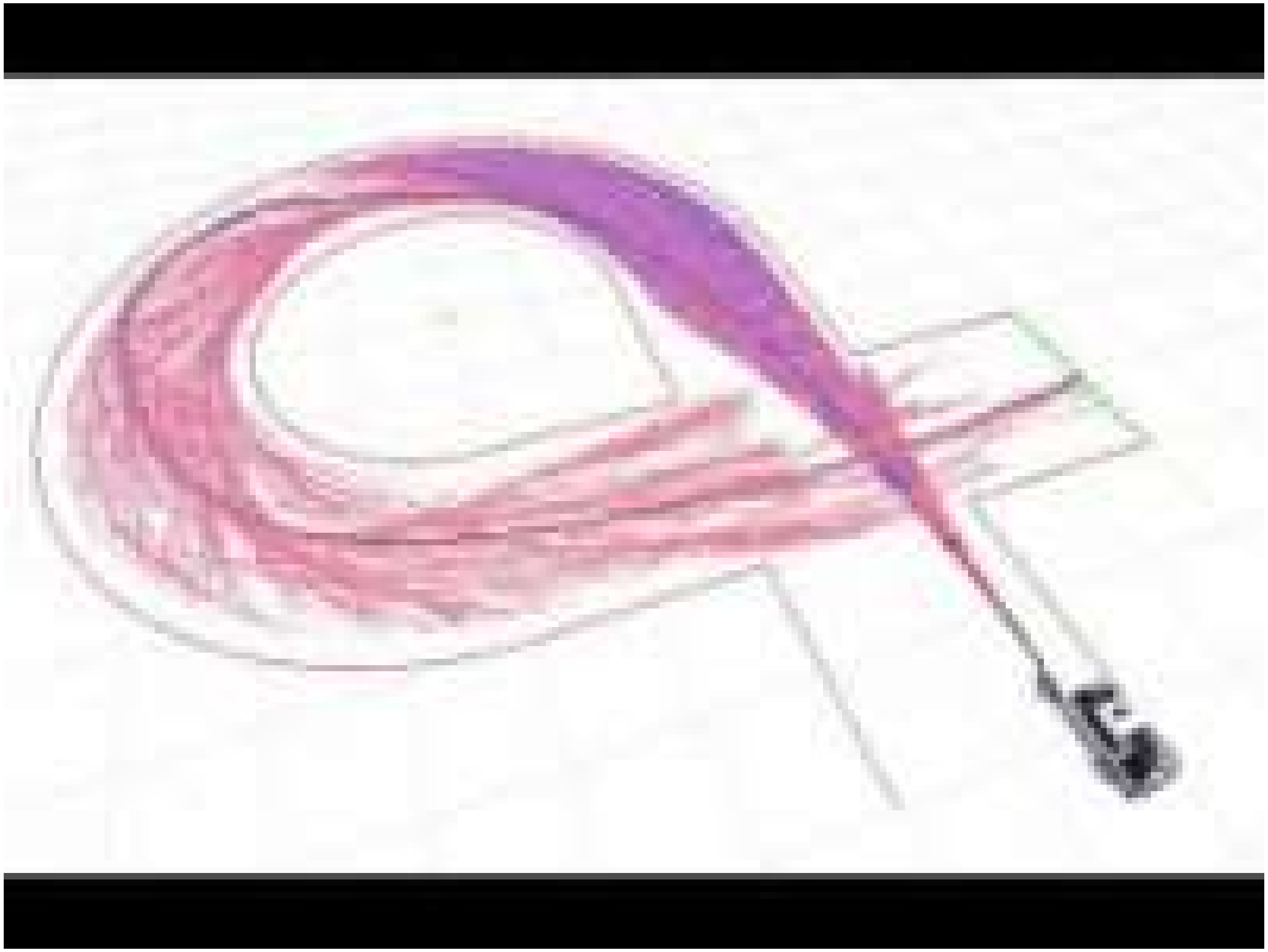


Fig. 2. Slices of $\mathcal{X}$ for a point mass robot in two dimensions with increasingly higher initial speeds. White areas represent $\mathcal{X}_{free}$, black areas are $\mathcal{X}_{obst}$, and gray areas approximate $\mathcal{X}_{ric}$.

Example: RRT Dubin's path planner

RRT trajectory planning with racecar dynamics



Variations on a Theme: Biased RRT sampling

Input: q_{start}, q_{goal}, number n of nodes, stepsize α, β
Output: tree $T = (V, E)$

initialize $V = \{q_{\text{start}}\}$, $E = \emptyset$
for $i = 0 : n$ **do**
    if rand$(0, 1) < \beta$ **then** $q_{\text{target}} \leftarrow q_{\text{goal}}$
    else $q_{\text{target}} \leftarrow$ random sample from Q
    $q_{\text{near}} \leftarrow$ nearest neighbor of q_{target} in V
    $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{target}} - q_{\text{near}}|}(q_{\text{target}} - q_{\text{near}})$
    if $q_{\text{new}} \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q_{\text{new}}\}$, $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
end for

Biasing tree expansion towards the goal: Sample the goal state q_{goal} itself with probability β > 0

Variations on a Theme: Bi-directional RRT

RRT_BIDIRECTIONAL(x_{init}, x_{goal});
  $\mathcal{T}_a$.init(x_{init}); $\mathcal{T}_b$.init(x_{goal});
  for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE();
    if not (EXTEND($\mathcal{T}_a, x_{rand}$) $=$ *Trapped*) **then**
      if (EXTEND($\mathcal{T}_b, x_{new}$) $=$ *Reached*) **then**
        Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
    SWAP($\mathcal{T}_a, \mathcal{T}_b$);
  Return *Failure*

Fig. 7. A bidirectional rapidly exploring random trees–based planner.

Bi-directional RRT grows two trees towards each other: one from the initial state, and one from the goal

Example: Bi-directional RRT path-planning