EECE 5550
Mobile Robotics

Lecture 14: Motion Planning

Derya Aksaray

Assistant Professor

Department of Electrical and Computer Engineering

Northeastern University

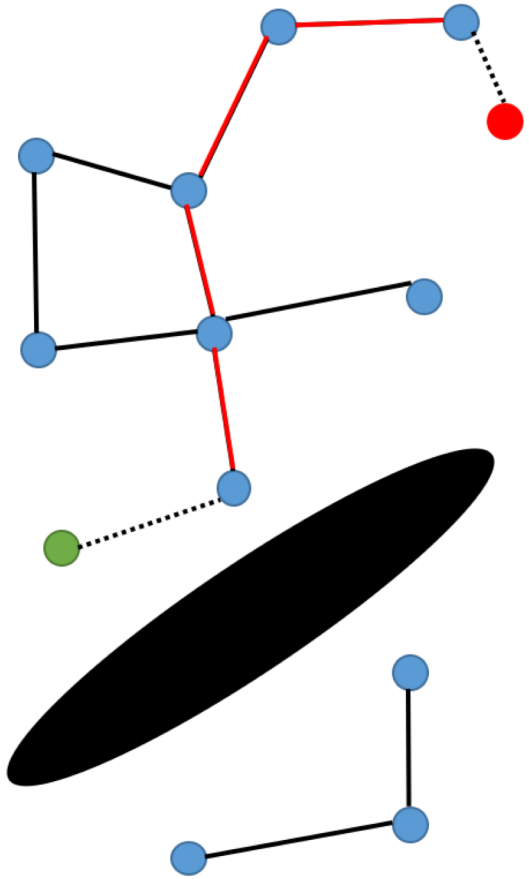
Plan of the day

Last time: Planning as search

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

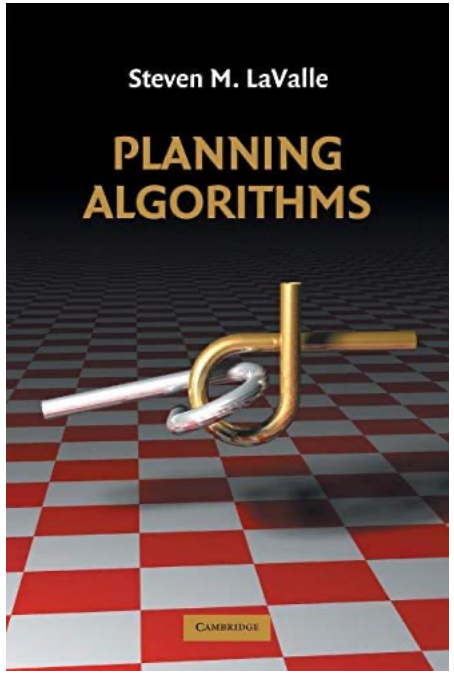
Today: Application to robot motion planning

- 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



Chapters 4,5

Classic papers:

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

Classical goals of motion planning

Assumptions

- Static world
- Robot knows about the map

Goals

- Collision-free trajectories
- Robot should reach the goal location with min cost

  (e.g., min time, min distance)

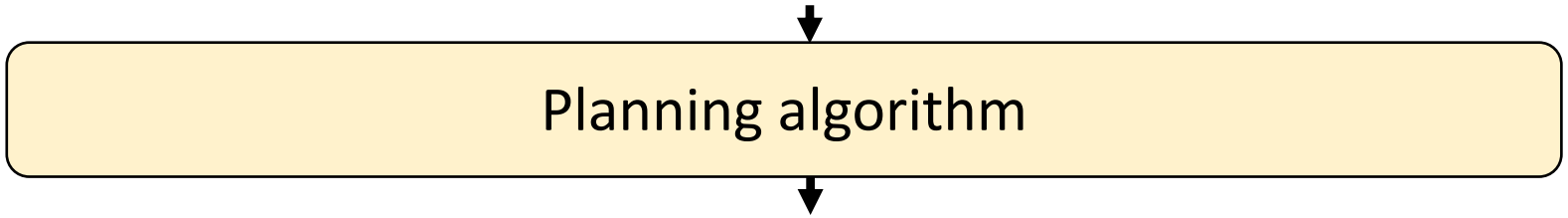
Planning in dynamic environments (not the scope of this course)

- More suitable to real-world applications
- Would require some knowledge about the movement of the surroundings

Motion planning problem

Input

- A start pose (configuration) of the robot
- A desired goal pose (configuration)
- A (geometric) description of the robot
- A (geometric) representation of the environment



Output

- A (optimal) path that moves the robot from start to goal while never colliding with the obstacles

Workspace and configuration space

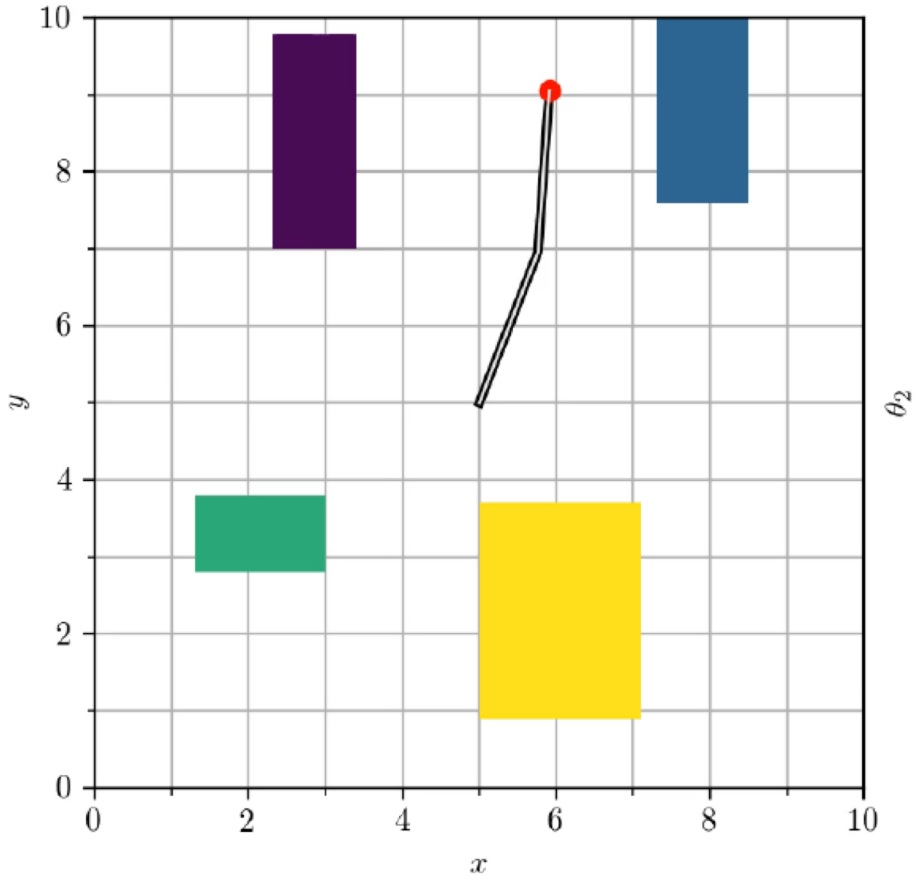
Workspace: The environment in which the robot is operating.

Often modeled at the level of free and occupied space

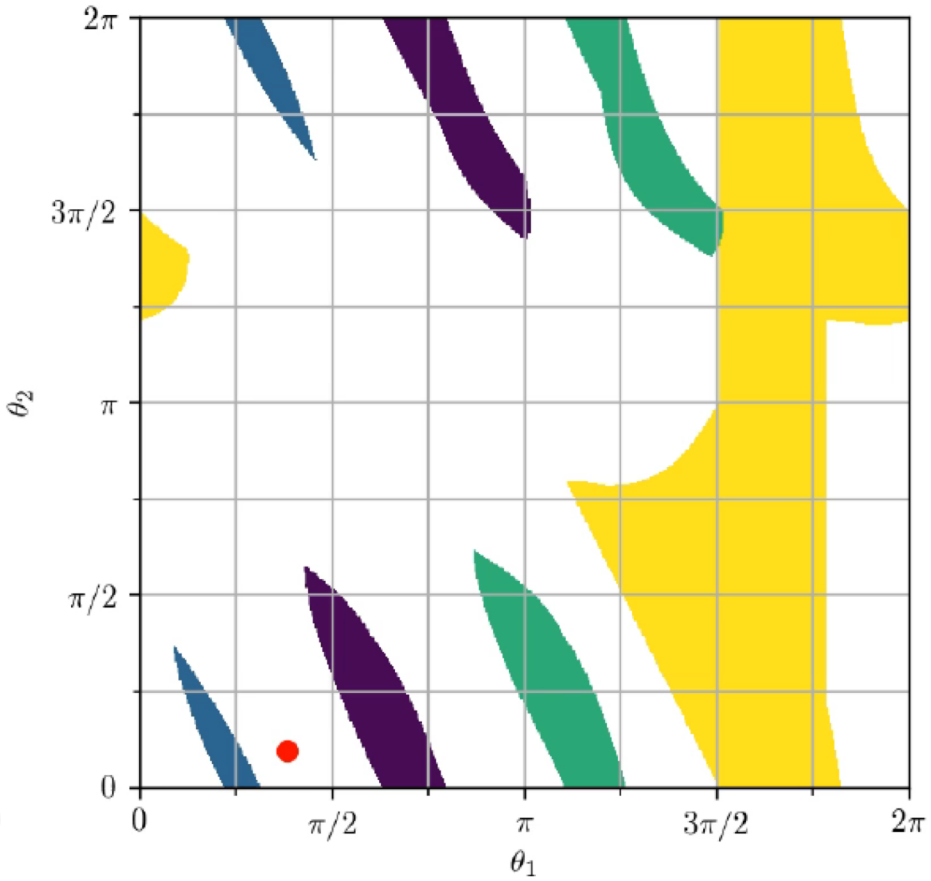
Configuration space (C-space): The set of all possible robot configurations (states)

- A configuration of a robot is a minimal set of variables that specifies the position and orientation of each rigid body composing the robot.
- The number of degrees of freedom of a robot is the dimension of the C-space.
  - The min # of variables required to fully specify the position and orientation of each rigid body belonging to the robot.
- Partially determined by the workspace (no collisions permitted).

Workspace and configuration space: two-link arm

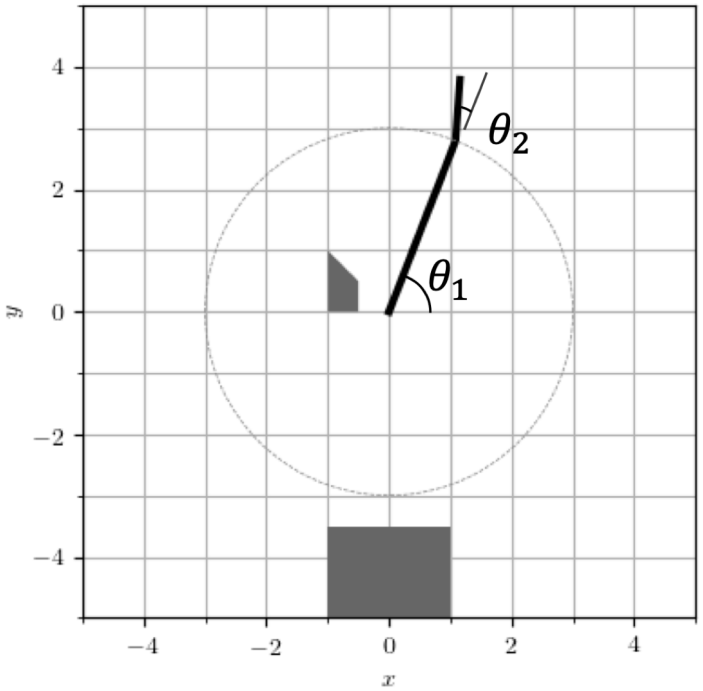


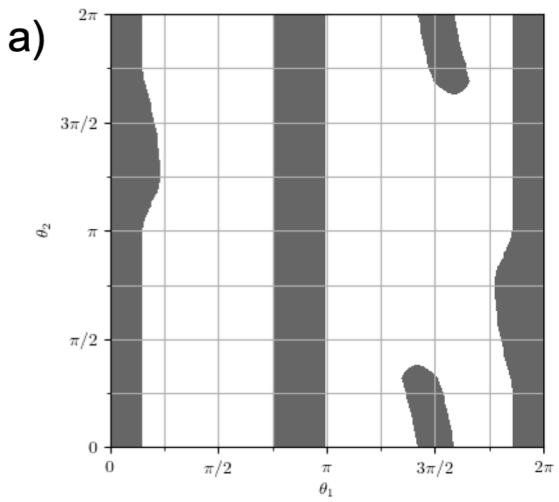
Workspace

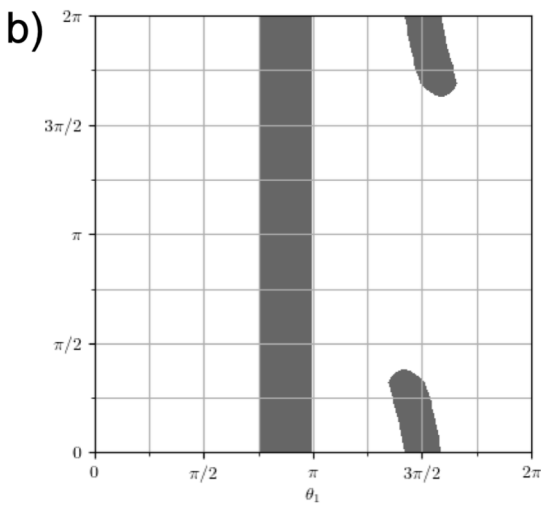


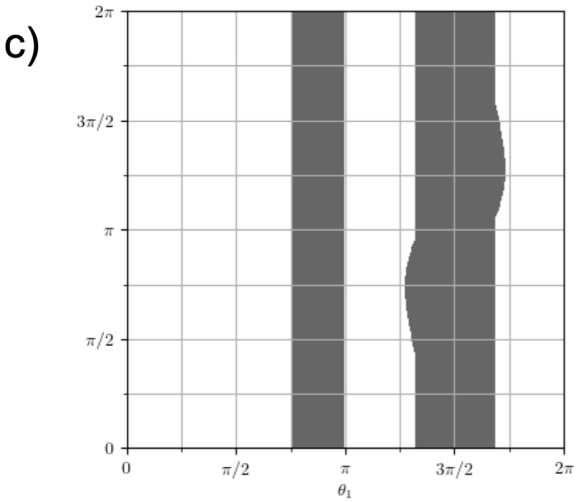
Configuration space

Exercise: configuration space









Some robot examples: C-space

- Point robot in $\mathbb{R}^2$ - Suppose that the robot can only translate through space
  - Robot's configuration: (x, y)
  - C-space: 2D plane

- 2D robot that can translate and rotate in a 2D workspace
  - Robot's configuration: (x, y, θ)
  - C-space: $SE(2) = \mathbb{R}^2 \times SO(2)$

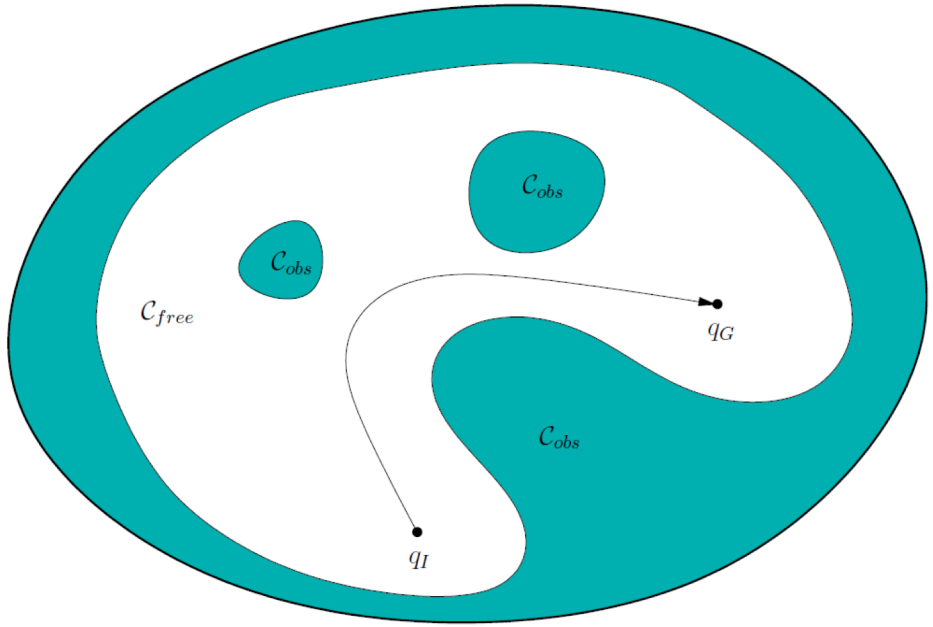
- A space robot moving in 3D space
  - Robot's configuration: $(x, y, z, \theta, \phi, \psi)$
  - C-space: $SE(3) = \mathbb{R}^3 \times SO(3)$

- A fixed robot with n arm linkages
  - Robot's configuration: n angles
  - C-space: n-dimensional 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 $q_G \in C_{free}$



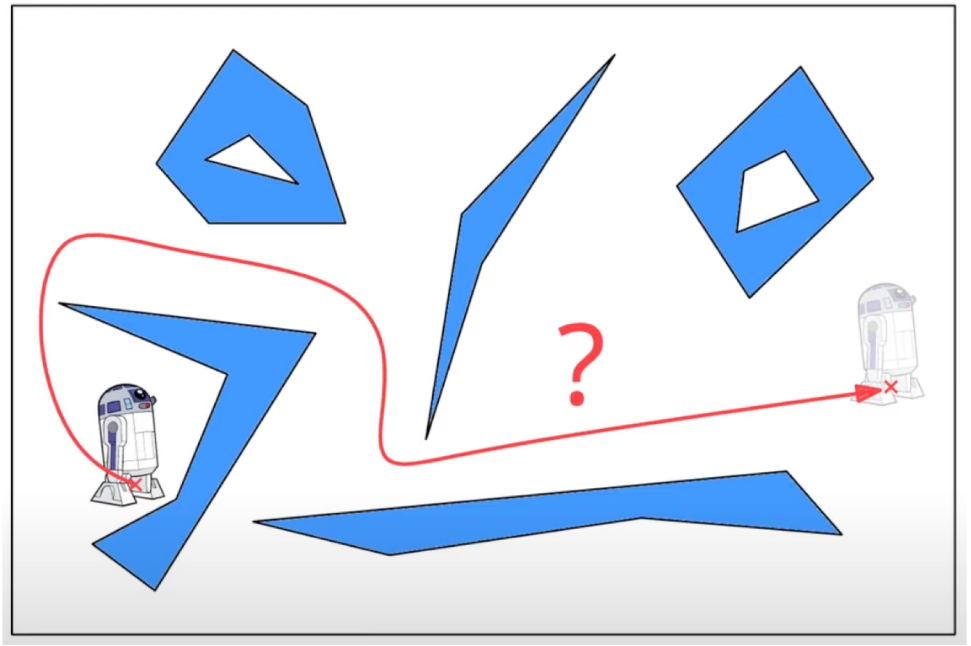
Find: A *path* τ: $[0,1] \rightarrow C_{free}$ such that $\tau(0) = q_I$ and $\tau(1) = q_G$

Configuration space

What are critical when constructing free and occupied c-space?

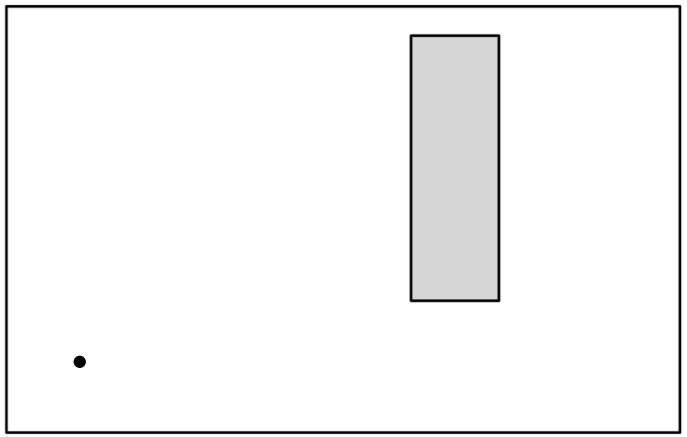
- Shape of robot
  - (point shaped)
- Shape of obstacle
  - (polygonal)
- Motion (e.g., rotation) of robot
  - (only translation)
- Motion of obstacles
  - (no motion)
- 2D, 3D
  - (2D)

Simplest case



Workspace and configuration space of a robot

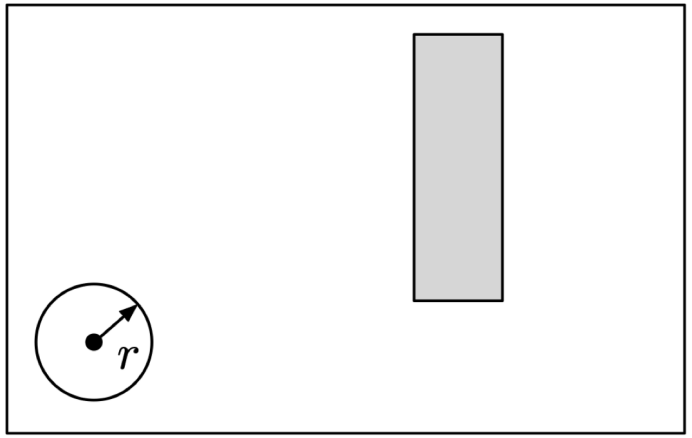
Simplest case



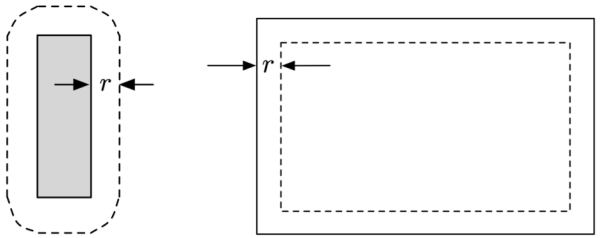
Workspace: Rectangular 2D region with an obstacle

Robot: Point robot

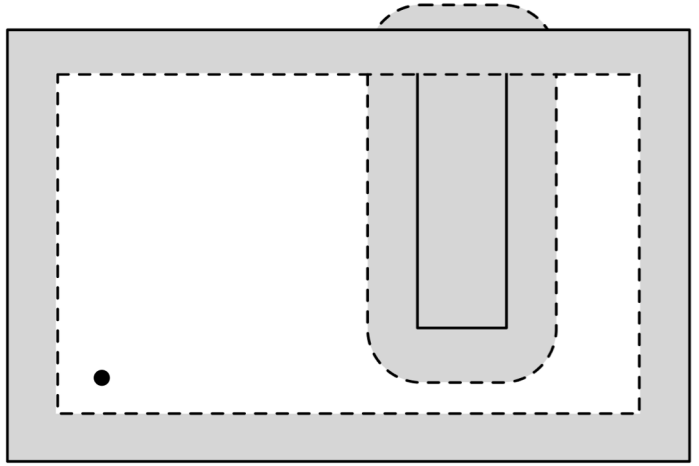
Free configuration space: The white region



Robot: Disc robot

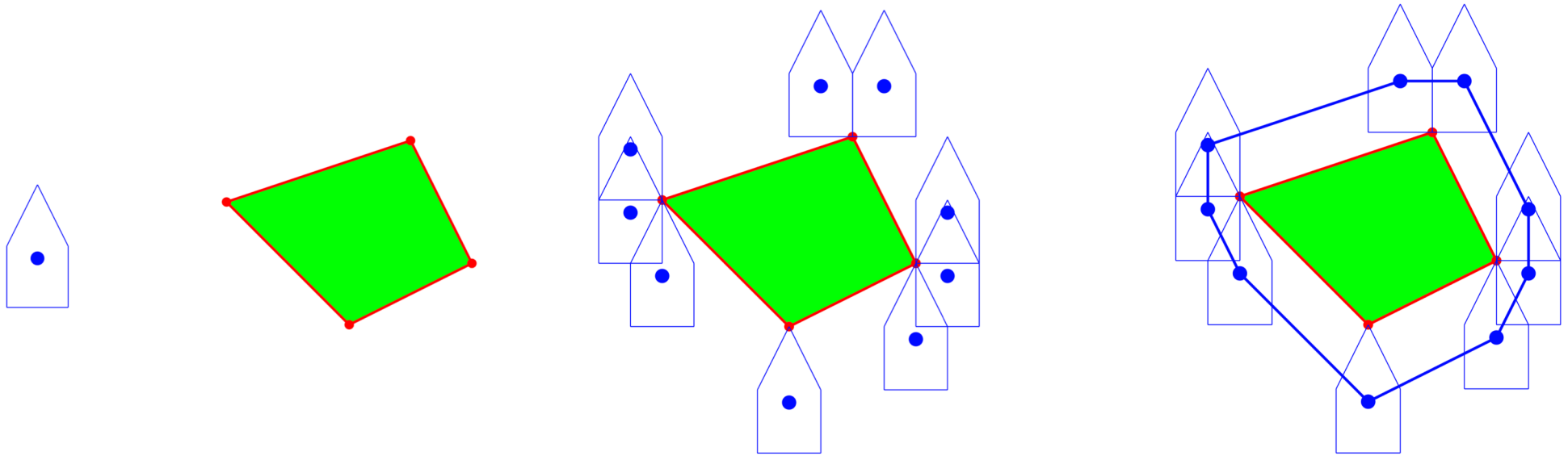


Expand all obstacles by radius r



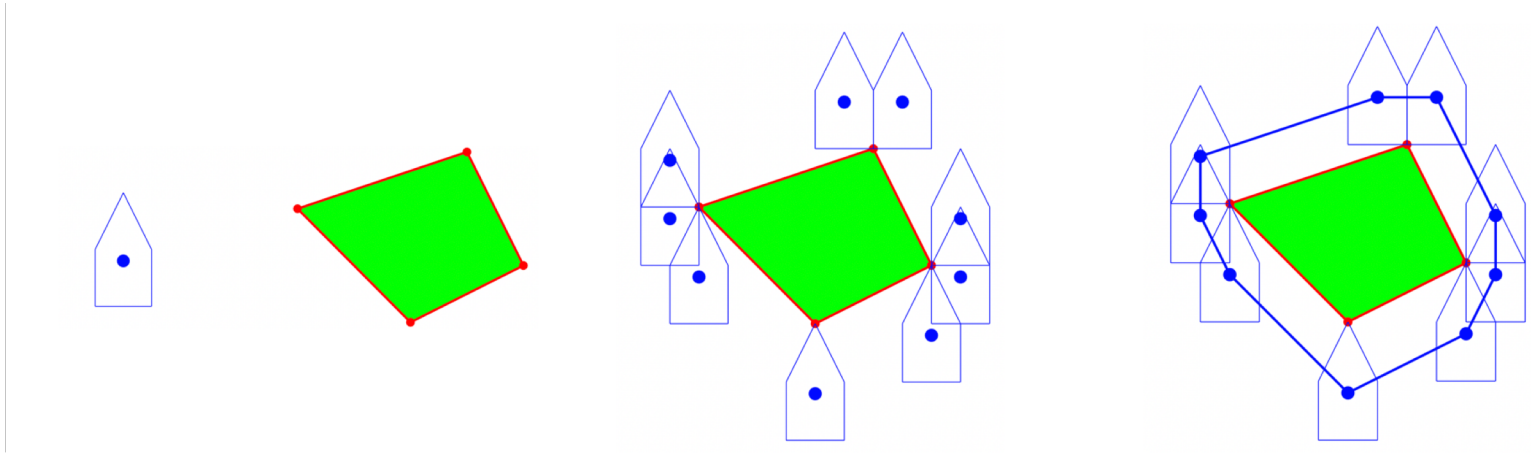
Free configuration space of non-circular robot

Example: Consider a polygonal robot R which will navigate around a polygon obstacle P with planar translation.



Free configuration space of non-circular robot

- How do we formally grow the obstacles by the robot's shape so that we can transform the motion planning problem into one where the robot is treated as a point?



Minkowski sum: Let P and Q *be* sets of points. *The Minkowski sum* $P \oplus Q$ *of* P *and* Q is

$$P \oplus Q = \{ p + q \mid p \in P, q \in Q \},$$

where $p+q=(p_x+q_x, p_y+q_y)$, for $p=(p_x,p_y)$ and $q=(q_x,q_y)$.

- summing all vertices of P and Q into the set S,
- taking the convex hull of the points in S.

Robot motion planning: Challenges

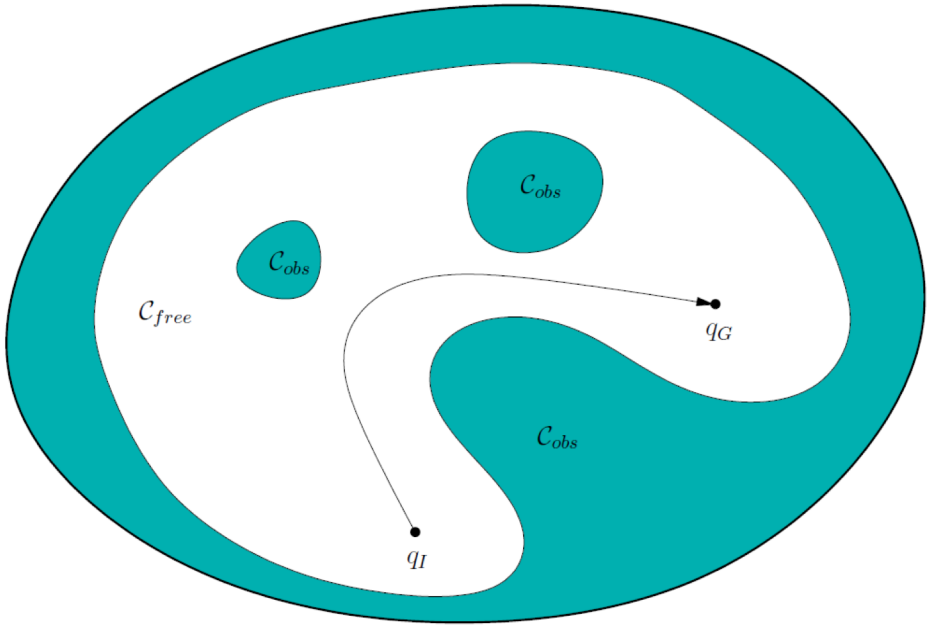
Basic challenge: Workspaces and robot configuration spaces are generally *continuous*

⇒ **Very** hard to model arbitrary continuous shapes

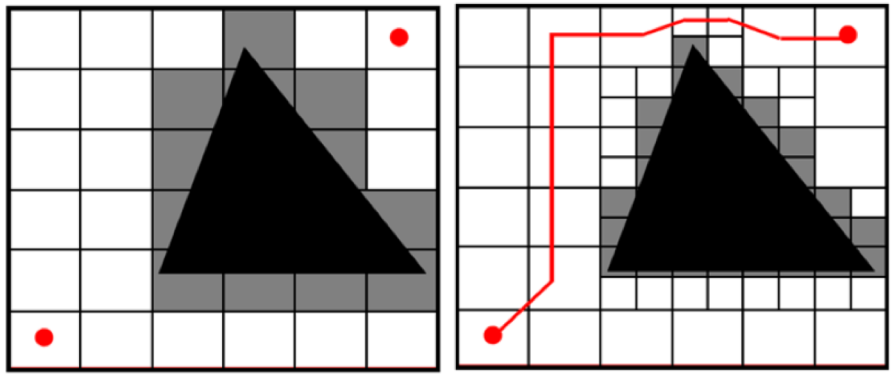
One natural approach: Discretize the C-space to get a *finite approximation*, and then solve the problem via *search*.

Two general approaches to discretize C-spaces:

- **Combinatorial planning:** Characterize the free C-space by explicitly capturing the connectivity of the free space into a graph, then graph search
- **Sampling-based planning:** Incrementally search C-space with the consideration of collision detection



Some important definitions for search alg.



A path planning algorithm is:

complete:

- If a path exists, it finds it in *finite* time
- If a path does not exist, it returns in *finite* time

probabilistically complete:

- if a path exists, the probability of finding it tends to go to 1 as the iterations go to infinity

resolution complete:

- If a path exists, it finds one; otherwise, it terminates and reports no solution within a specified resolution

sound:

- if guaranteed to never cross an obstacle

optimal:

- if guaranteed to find the shortest path (if it exists)

Combinatorial planning

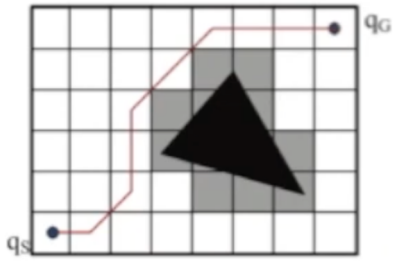
Step 1: Compute C-space

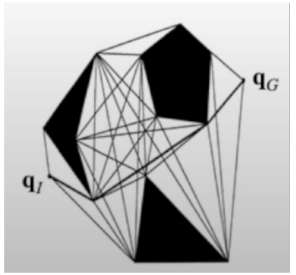
Step 2: Generate a roadmap (i.e., a graph) in free C-space

- Cell decomposition methods, visibility graphs, occupancy grid maps, etc.

Step 3: Compute the min cost path from initial configuration to the goal configuration

- Cast as a graph search algorithm
- Dijkstra, A*, etc.

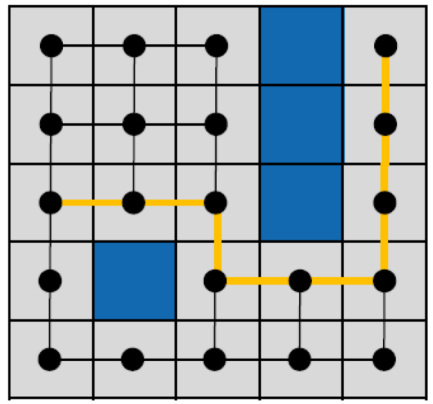




Simple case: route planning for a planar robot

Recall:

- Occupancy grids 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

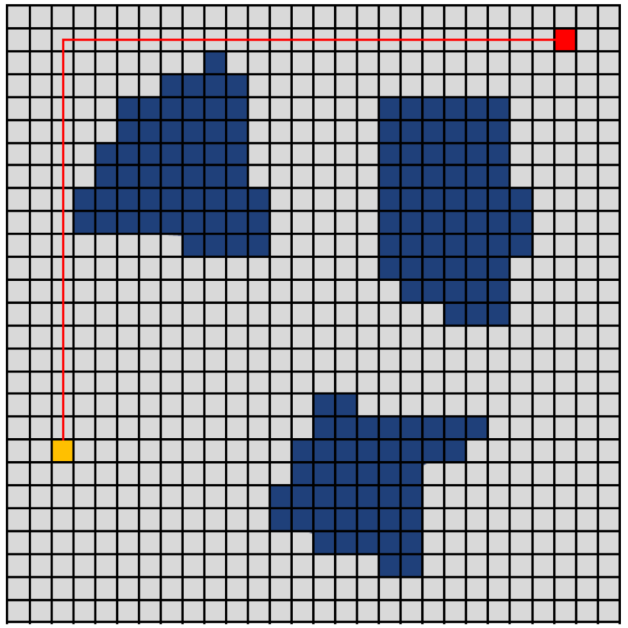
More generally, one can "voxelize" any *generic* configuration space C_{free}, and then apply the same graph search strategy

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.

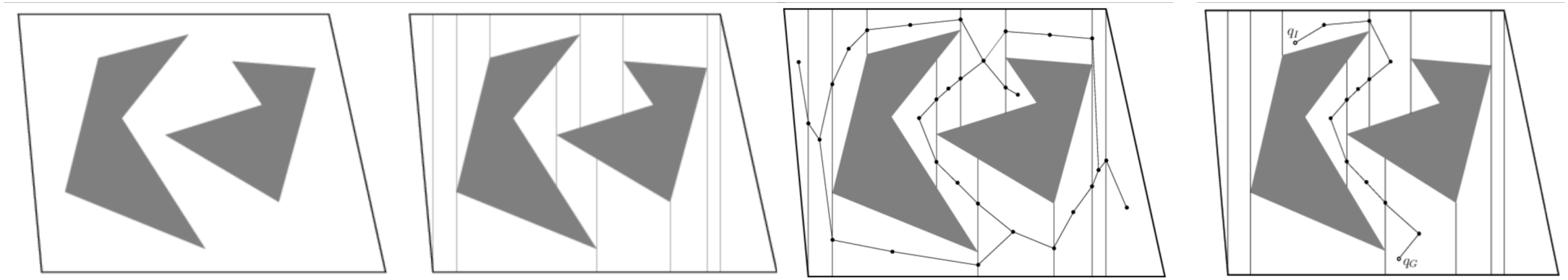
Con:

- 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 $O(r^d)$!



Vertical cell decomposition

- Divides the space into non-overlapping cells.
- Add vertical line segment to every obstacles' vertices.
- Create graph (roadmap): vertices in trapezoids and vertical segments



- If there is a path from start to goal, such a decomposition and graph search over that would guarantee to find the solution.
- Complex shape obstacles can lead to graphs that can not result in the actual shortest path.

Combinatorial planning - challenges

Exact method (no approximation needed) and complete but,

C-space has high dimensionality

- A rigid body in 3D space has a C-space with 6 dimensions ($C = \mathbb{R}^3 \times SO(3)$)

Simple obstacles have complex C-obstacles

- Impractical to compute explicit representation of free space for high dof robots

More attention on sampling-based planning…

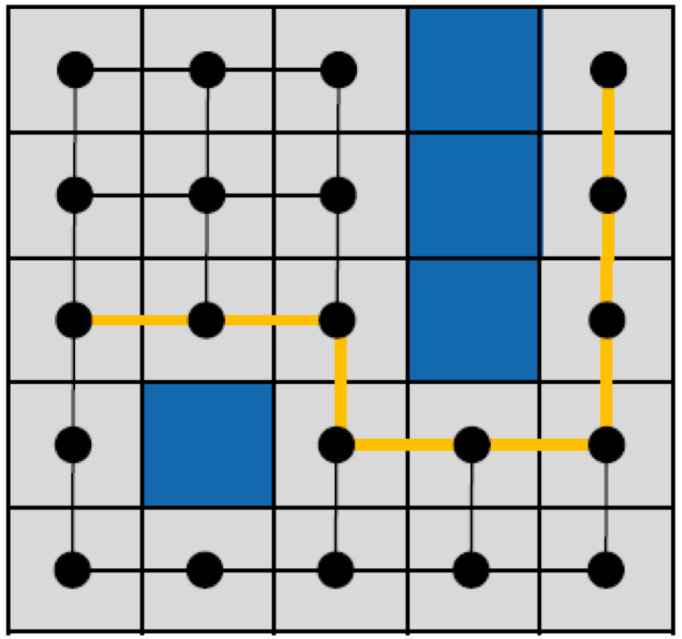
Sampling-based planning

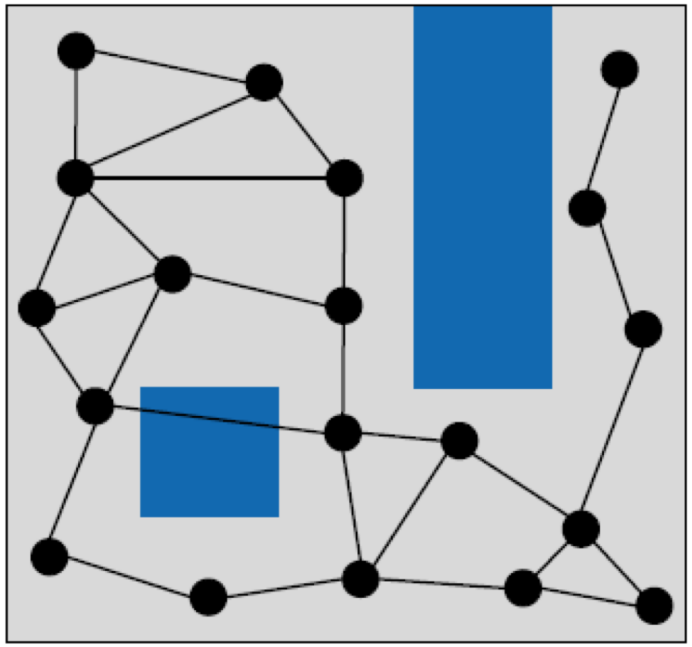
Main idea: Rather than trying to capture *every* point in the configuration space *C* via voxelization, let's *randomly sample* a *representative set S* of points in C_{free}

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

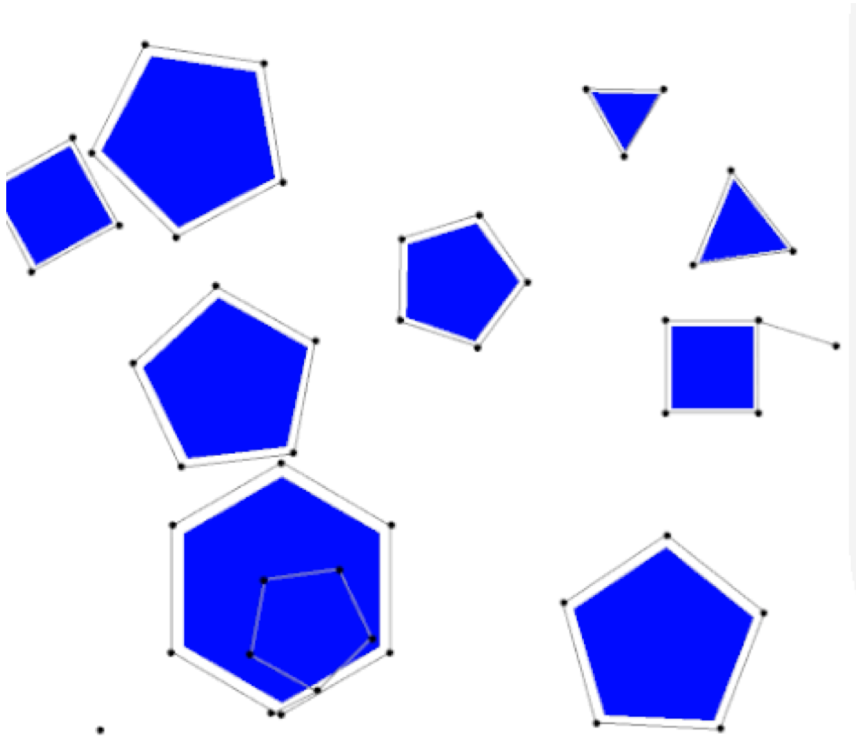
Starting with an empty graph G, repeat:

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.

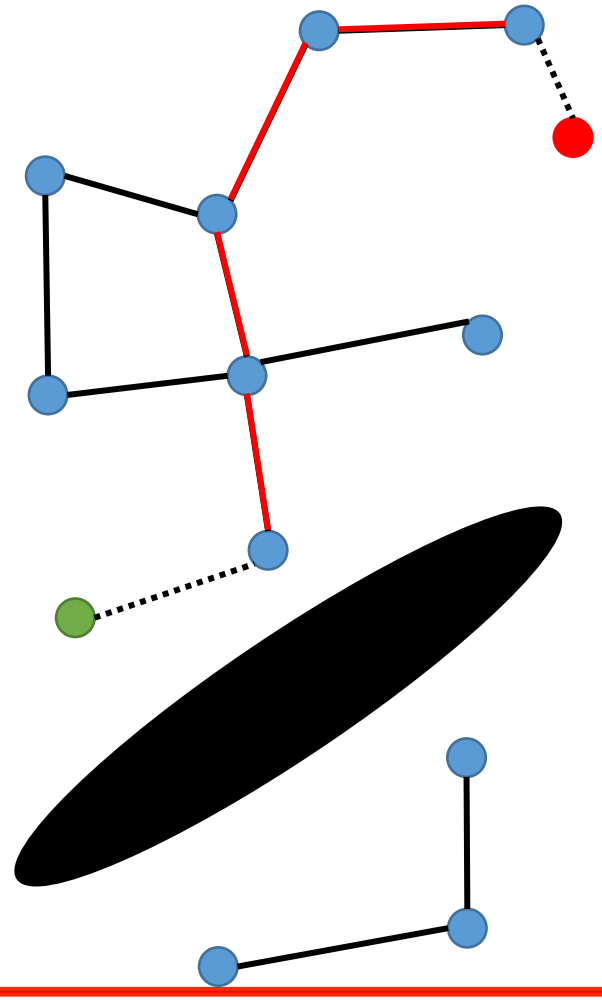


The Probabilistic Roadmap (PRM) Algorithm

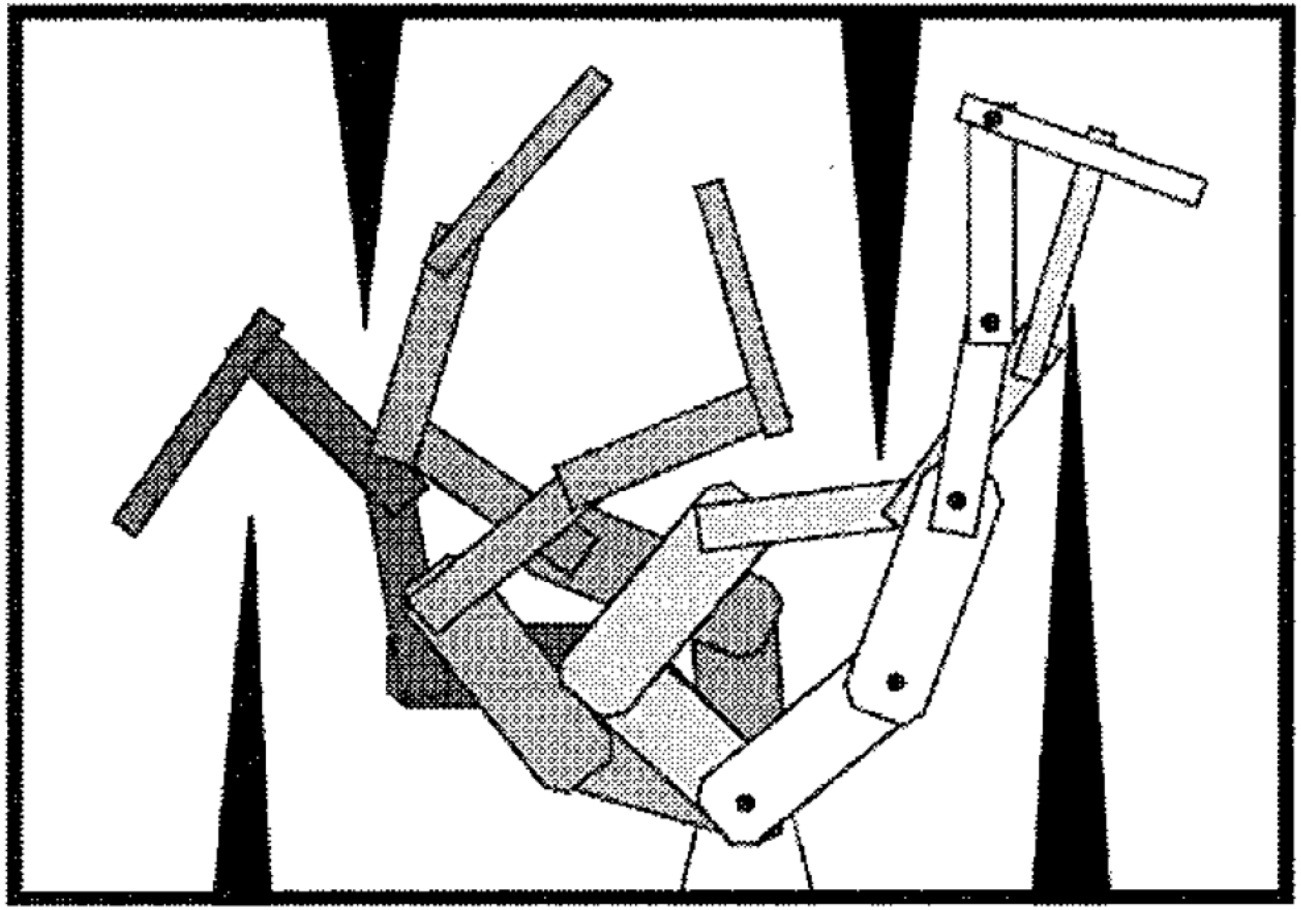
Two-phase algorithm for sampling-based planning:

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

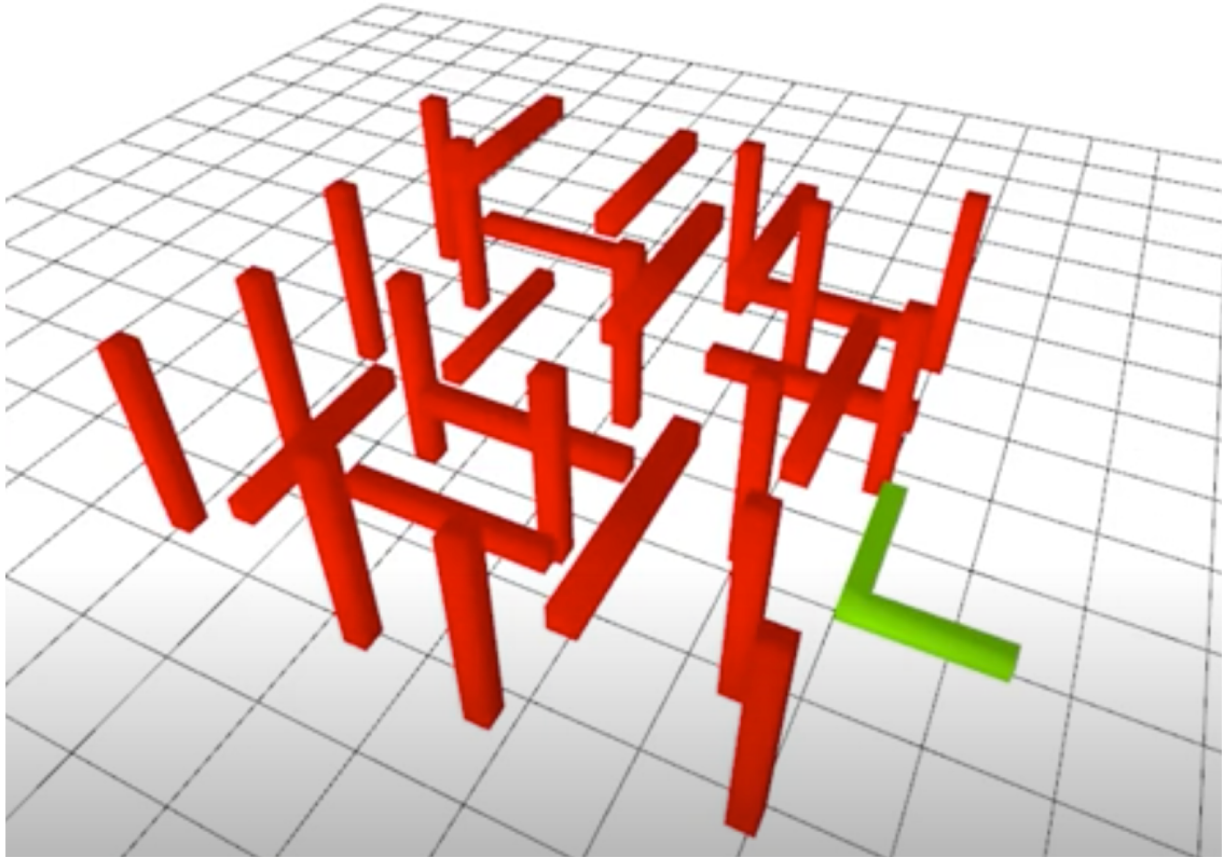
2. **Query:** At run-time, given initial point $x \in G$ and goal $y \in G$:
   - Plan local paths from *x* and *y* to (nearby) vertices in the *same connected component* of *G*
   - Graph search!



Example: PRM planning with a 5 DOF robotic arm



Example: Solving the piano mover's problem with PRMs



https://www.youtube.com/watch?v=3_S3GPxAMYA

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*

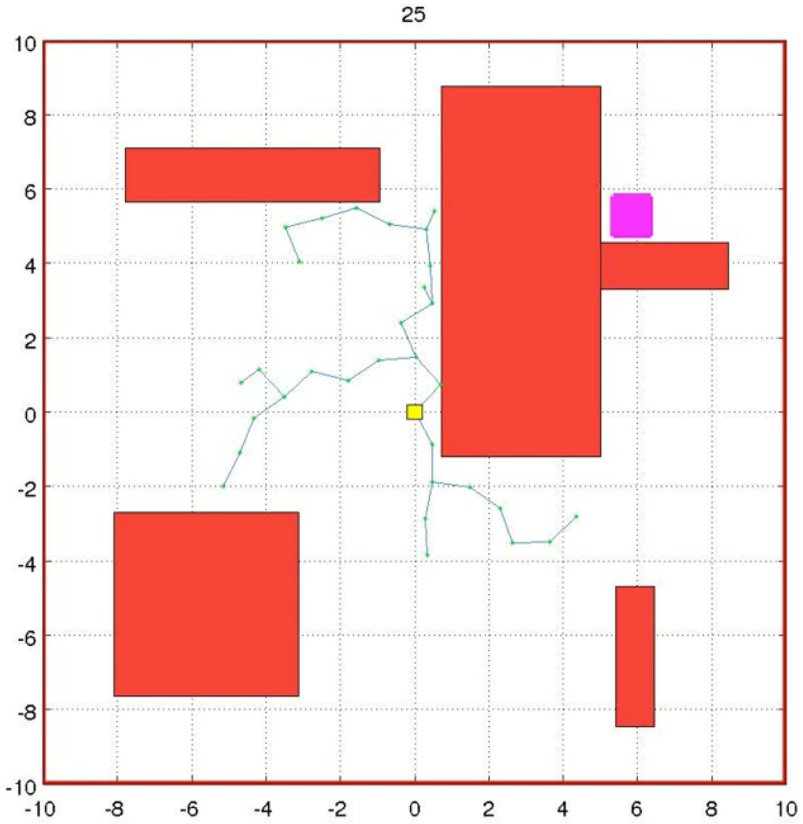
- **Much** more scalable than grid methods
  - We sparsely (inner) approximate true space C_{free} using sample-set *S*
  - Works well for high degree-of-freedom robots (e.g. robot arms, etc.)
- Probabilistically complete: If a feasible plan exists, the probability of finding it approaches 1 as the number 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*.



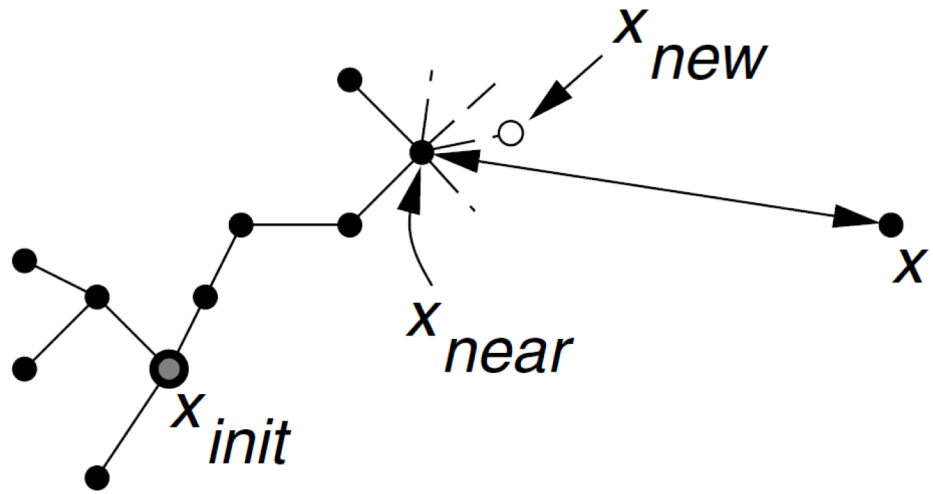
https://www.youtube.com/watch?v=YKiQTJpPFkA

RRT algorithm

```
BUILD_RRT(x_init)
1    T.init(x_init);
2    for k = 1 to K do
3        x_rand ← RANDOM_STATE();
4        EXTEND(T, x_rand);
5    Return T
```

```
EXTEND(T, x)
1    x_near ← NEAREST_NEIGHBOR(x, T);
2    if NEW_STATE(x, x_near, x_new, u_new) then
3        T.add_vertex(x_new);
4        T.add_edge(x_near, x_new, u_new);
5        if x_new = x then
6            Return Reached;
7        else
8            Return Advanced;
9    Return Trapped;
```

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



EXTEND

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

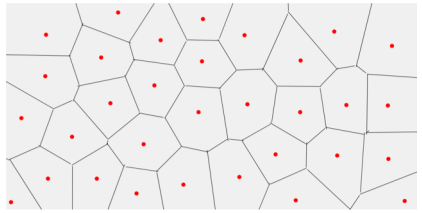
$$\dot{x} = f(x, u)$$

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 diagram:

- Each point decomposes the space around it into some region of influence
- All the points inside a Voronoi partition is closest to the point generated that partition

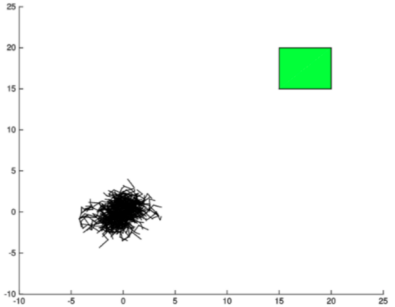


RRT does not have a particular step to create Voronoi diagram; however, one can see a Voronoi interpretation of the algorithm.

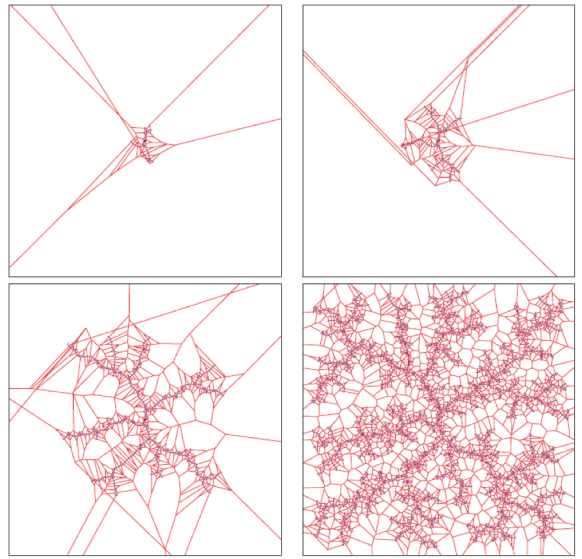
Voronoi bias:

- Samples are selected uniformly over the space.
- Such random sampling tends to place new vertices in larger Voronoi cells (⇒ unexplored regions)
- This helps to expand the tree to rapidly cover the space.
- Generating new points in the space, and extending from there better than naive sampling

Naive sampling



Pick a node on the tree
Extend from there



RRT: Key Properties

- Probabilistically complete

- Single query & directed
  - Edges in RRTs are directed: travel is from the initial location towards the goal (x -> y ≠ 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

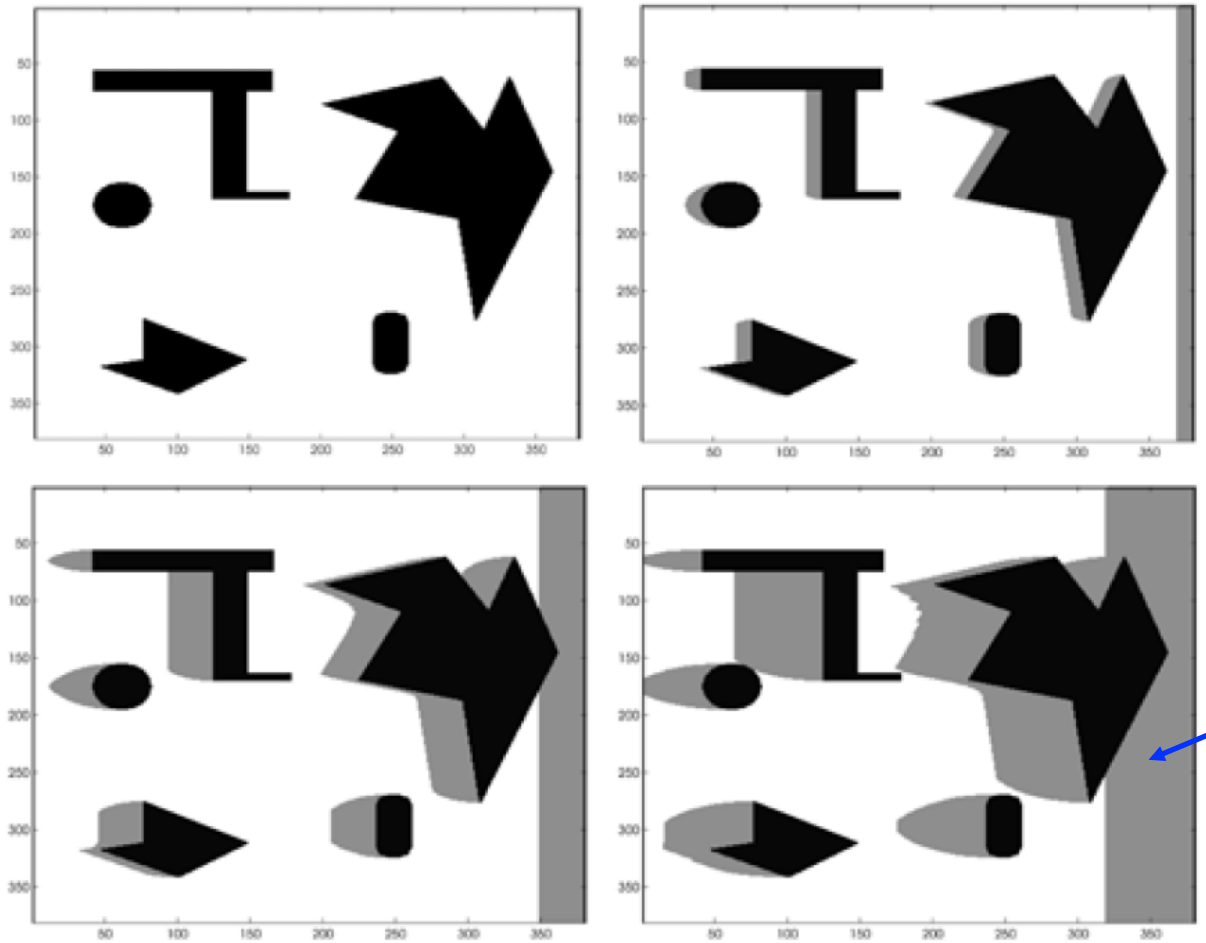
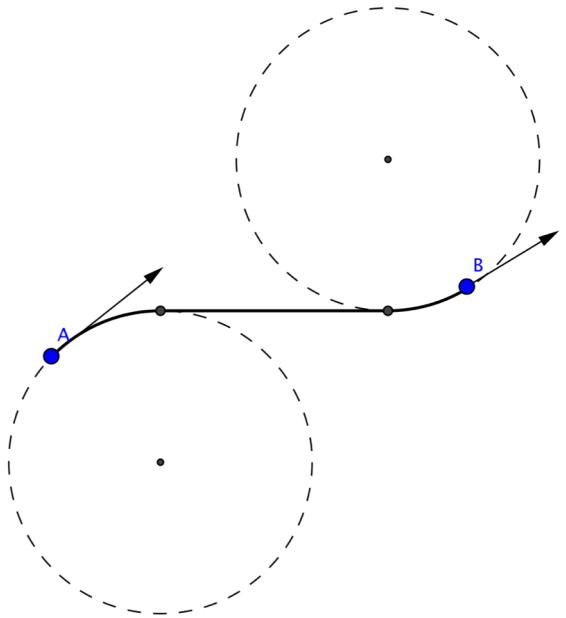


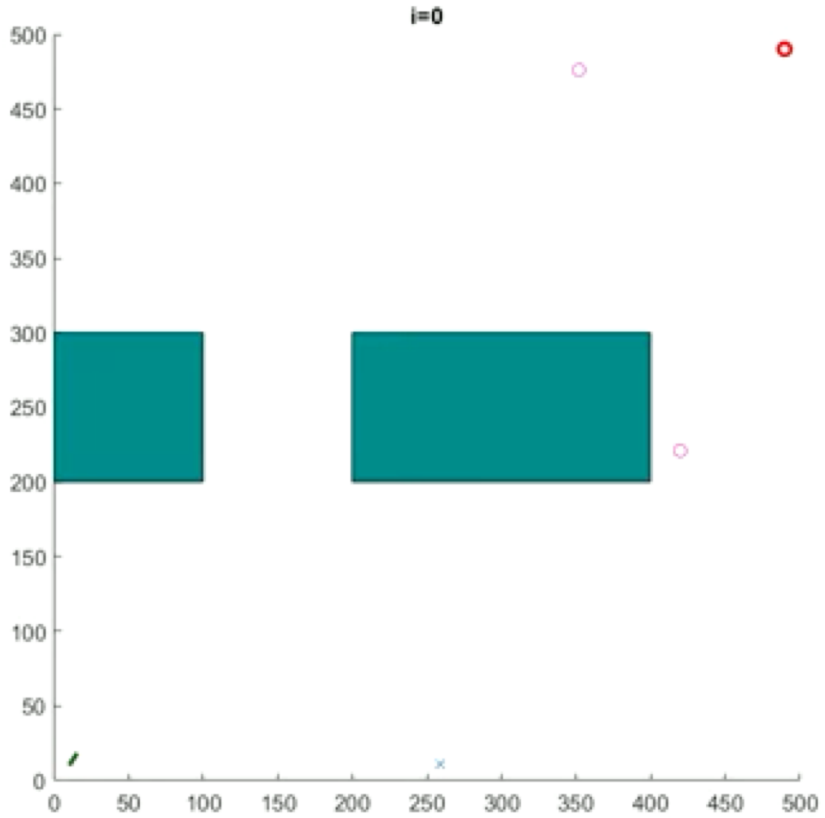
Fig. 2. Slices of $\mathcal{X}$ for a point mass robot in two dimensions with increasingly higher initial speeds. White areas represent $\mathcal{X}_{free}$, black areas are $\mathcal{X}_{obst}$, and gray areas approximate $\mathcal{X}_{ric}$.

Region of inevitable collision grows with respect to the robot's initial speed.

Example: RRT Dubin's path planner

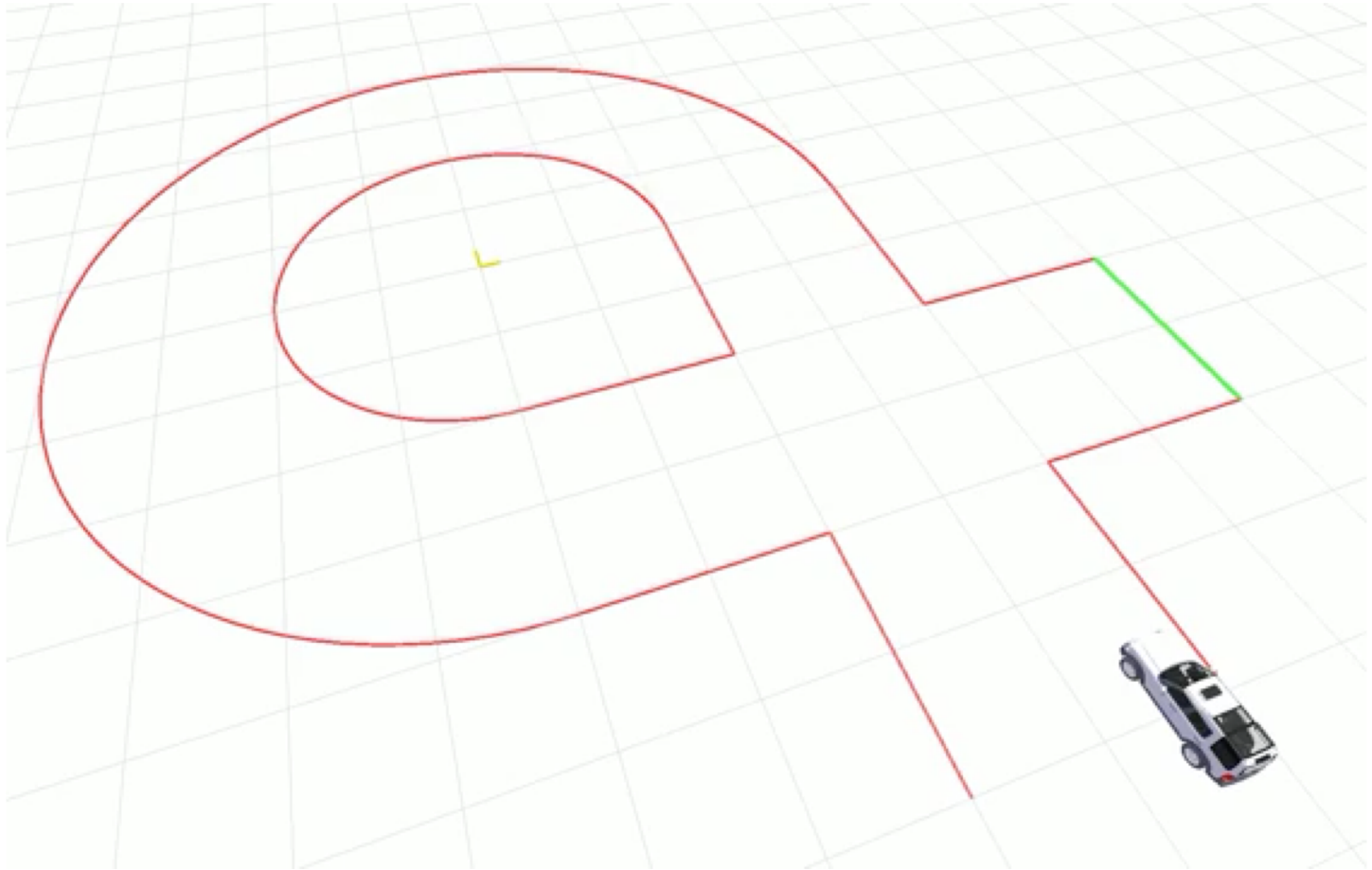
$$\dot{x} = V\cos(\theta)$$
$$\dot{y} = V\sin(\theta)$$
$$\dot{\theta} = u$$





https://www.youtube.com/watch?v=5UoErV8yoCA

RRT trajectory planning with racecar dynamics



https://www.youtube.com/watch?v=p3nZHnOWhrg

Other variants of RRT

- Various sampling ideas
  - Finding goal quickly
  - Sampling from narrow passages
- Construction of the tree in two directions
- RRT* (rewiring and cost incorporation) – asymptotically optimal

Variations on a Theme: Biased RRT sampling

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

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

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

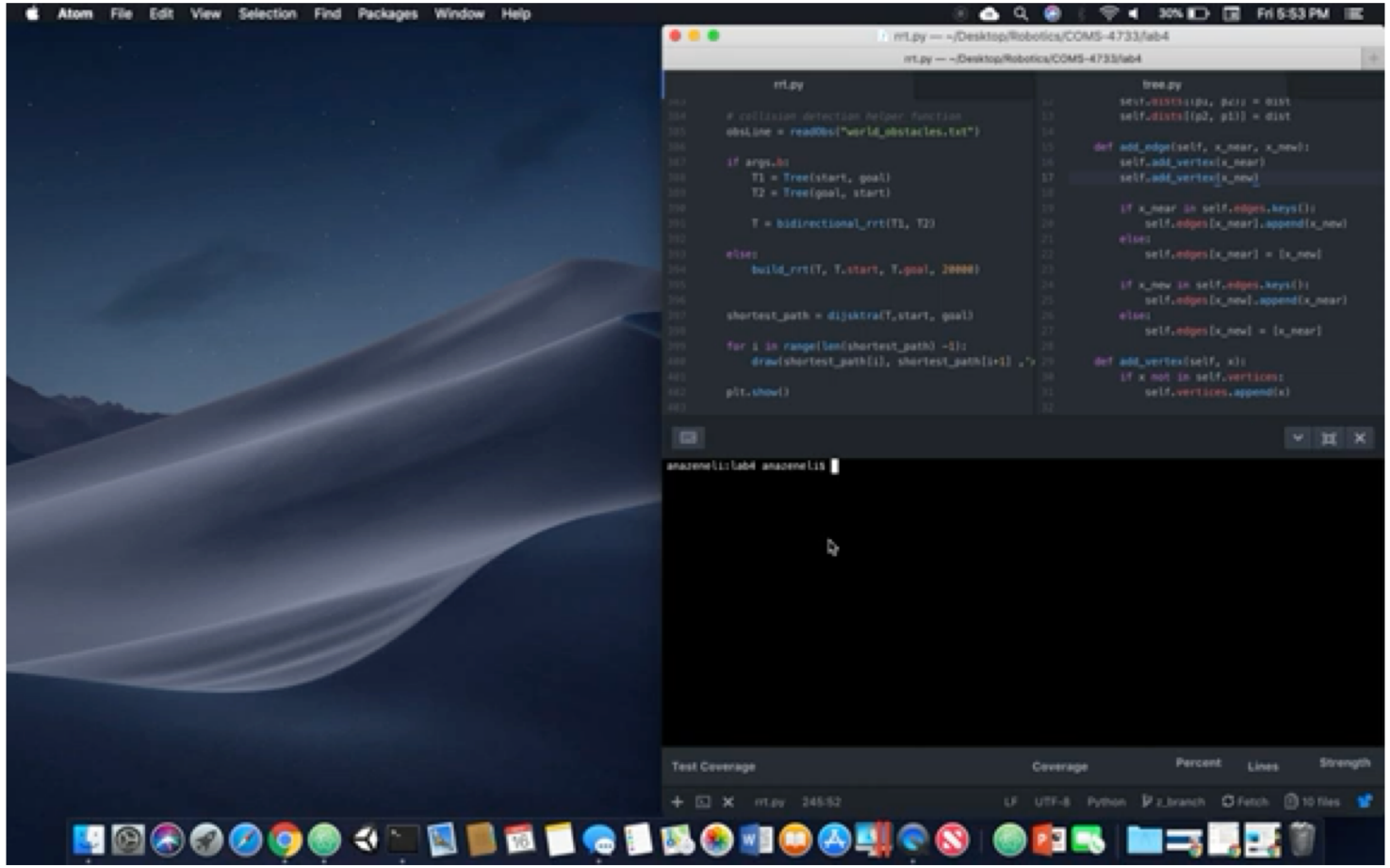
Variations on a Theme: Bi-directional RRT

RRT_BIDIRECTIONAL(x_{init}, x_{goal});
1 $\mathcal{T}_a$.init(x_{init}); $\mathcal{T}_b$.init(x_{goal});
2 **for** $k = 1$ **to** K **do**
3 $x_{rand} \leftarrow$ RANDOM_STATE();
4 **if not** (EXTEND($\mathcal{T}_a$, x_{rand}) = *Trapped*) **then**
5 **if** (EXTEND($\mathcal{T}_b$, x_{new}) = *Reached*) **then**
6 Return PATH($\mathcal{T}_a$, $\mathcal{T}_b$);
7 SWAP($\mathcal{T}_a$, $\mathcal{T}_b$);
8 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



https://www.youtube.com/watch?v=RtkMpPvUONo