Fall 2023-24

# SAMPLING BASED MOTION PLANNING

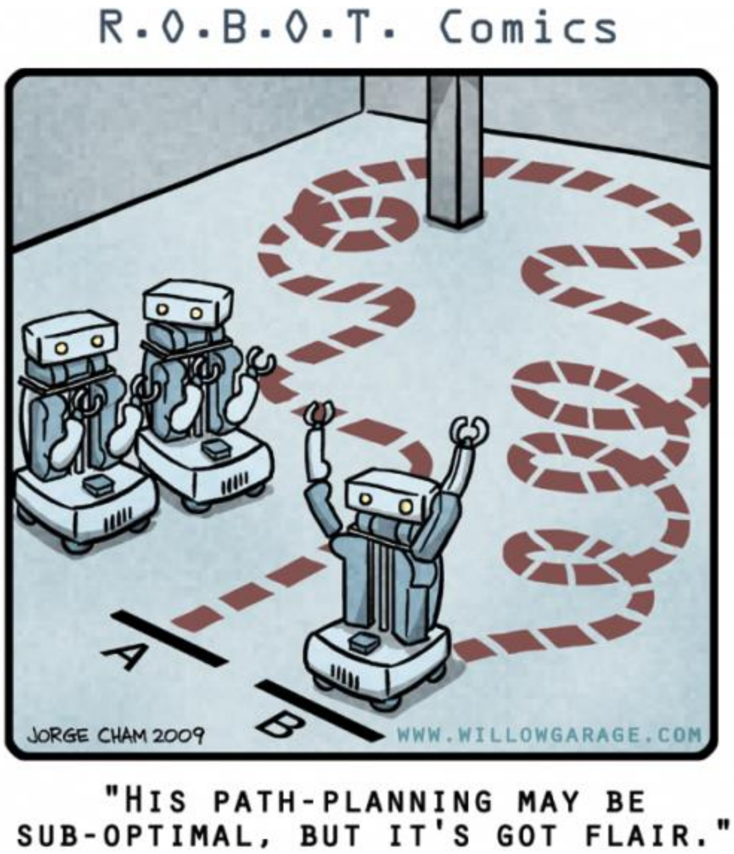
RBT350

Roberto Martin-Martin

Assistant Professor of Computer Science.

# Recap

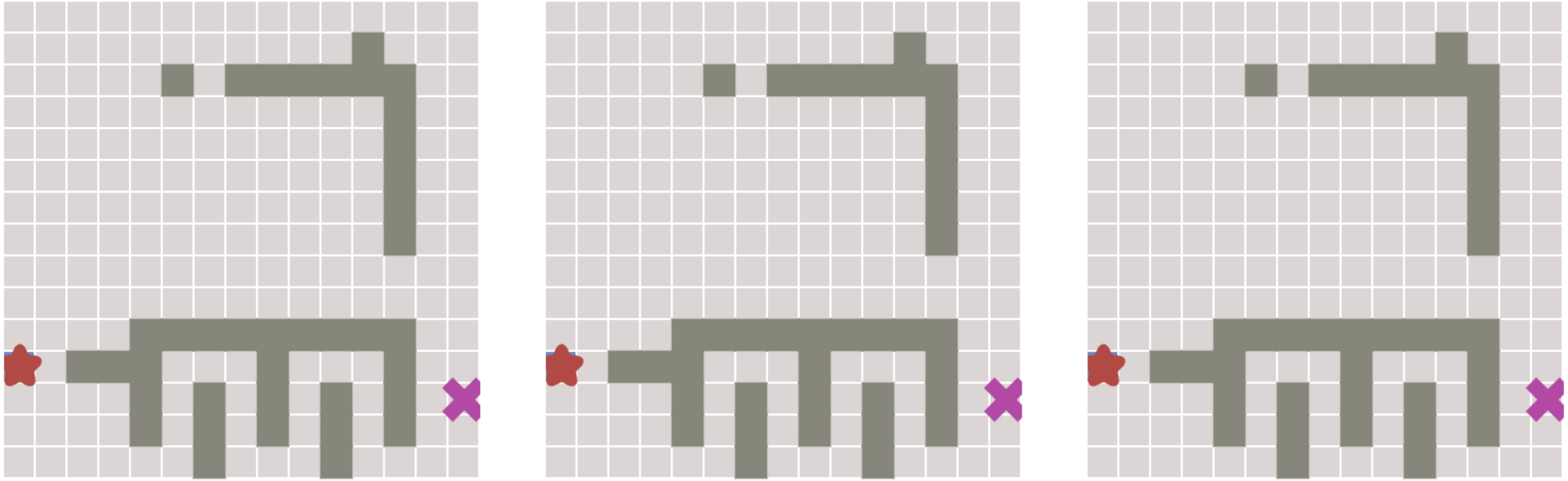
- Motion planning: finding a collision-free path from A to B
- Two families:
  - (Graph) Search Based Motion Planning
  - Sampling Based Motion Planning
- Ideally, instead of finding a path, we find the shortest path



# Exercise – Optimality

- Optimal algorithm:
  - If the algorithm finds a path, the path

Breadth First Search | Greedy Best-First Search | A* Search



# Play with them!

- https://cs.stanford.edu/people/abisee/tutorial/bfs.html
- https://cs.stanford.edu/people/abisee/tutorial/dfs.html
- https://cs.stanford.edu/people/abisee/tutorial/greedy.html
- https://cs.stanford.edu/people/abisee/tutorial/astar.html

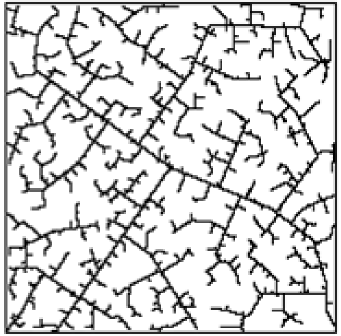
# Summary of Graph Search Algorithms

- Breadth First Search: First in first out, optimal but slow
- Depth First Search: Last in first out, not optimal and meandering
- Greedy Best First: Goes for the target, fast but easily tricked
- A* Search: "Best of both worlds": optimal and fast
- Dijkstra: Explores in increasing order of cost, optimal but slow

# What will you learn today?

- Sampling based motion planning
  - Why do we need them? Curse of dimensionality
  - PRM
  - RRT

45 iterations



390 iterations

# General Insights from Motion Planning

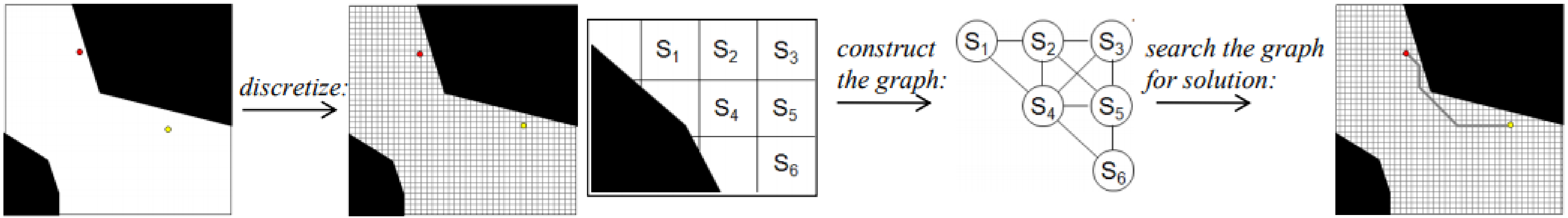
In the real world, the computational time for motion planning for robots is key and depends on two factors:

- Obstacle checking: This is significant if
  - Your robot is complex (e.g. a manipulator arm with many joints),
  - If your environment representation is complex (e.g. detailed meshes to perform collision checking with)
  - If you have a very large number of samples to check

- Priority queue operations: This is significant if
  - Planning in high-dimensional space, hence large state spaces
  - Many alternatives, with no clear inferior / superior choices

# What is the problem with search-based motion planners?

2D grid-based graph representation for 2D (*x*,*y*) search-based planning:



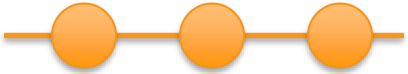

- Constructing the graph (check all cells to see if they are in collision or not) becomes the real bottleneck!
- Curse of dimensionality!
  - The complexity of the algorithm increases exponentially with the dimensions of the problem

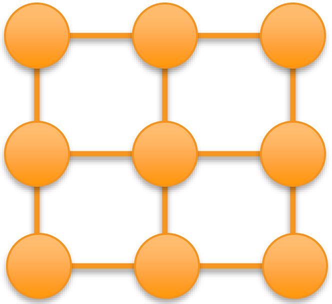
# What is the problem with search-based motion planners?

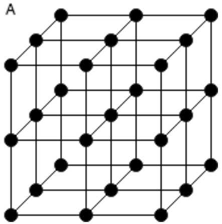
- Curse of dimensionality!
  - The complexity of the search algorithm increases exponentially with the dimensions of the problem

- How many iterations until I find a path with BFS? How many nodes/edges are there if…
  - the problem is 1D
  - the problem is 2D
  - the problem is 3D

  …



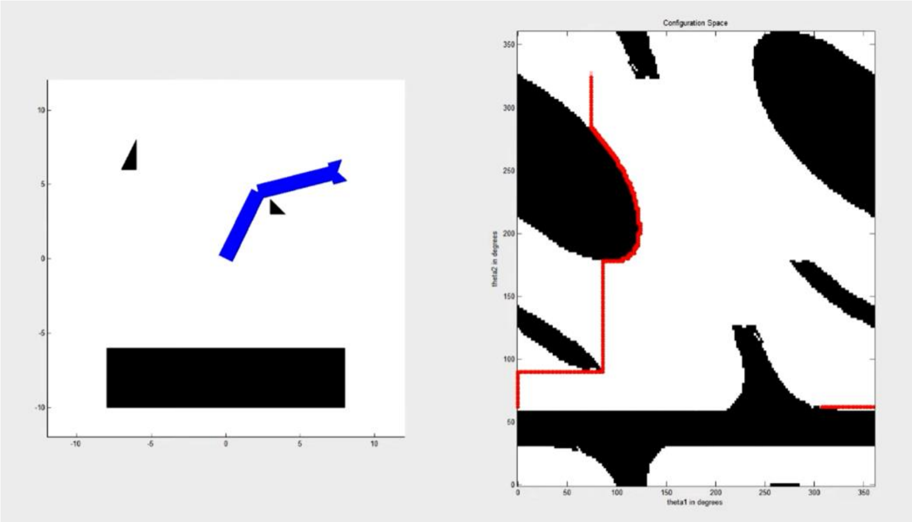




# That is really bad for robot arms

- Dimensionality of the space = number of joints
- 7 DoF robot → 7 dimensions!

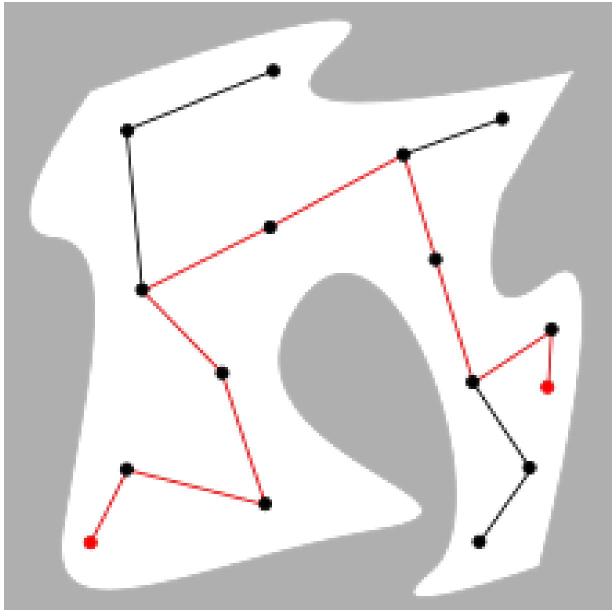
- Imagine we discretize each joint by 5 (0, 45, 90, 135, 180 degrees)
  - Number of cells: 16807



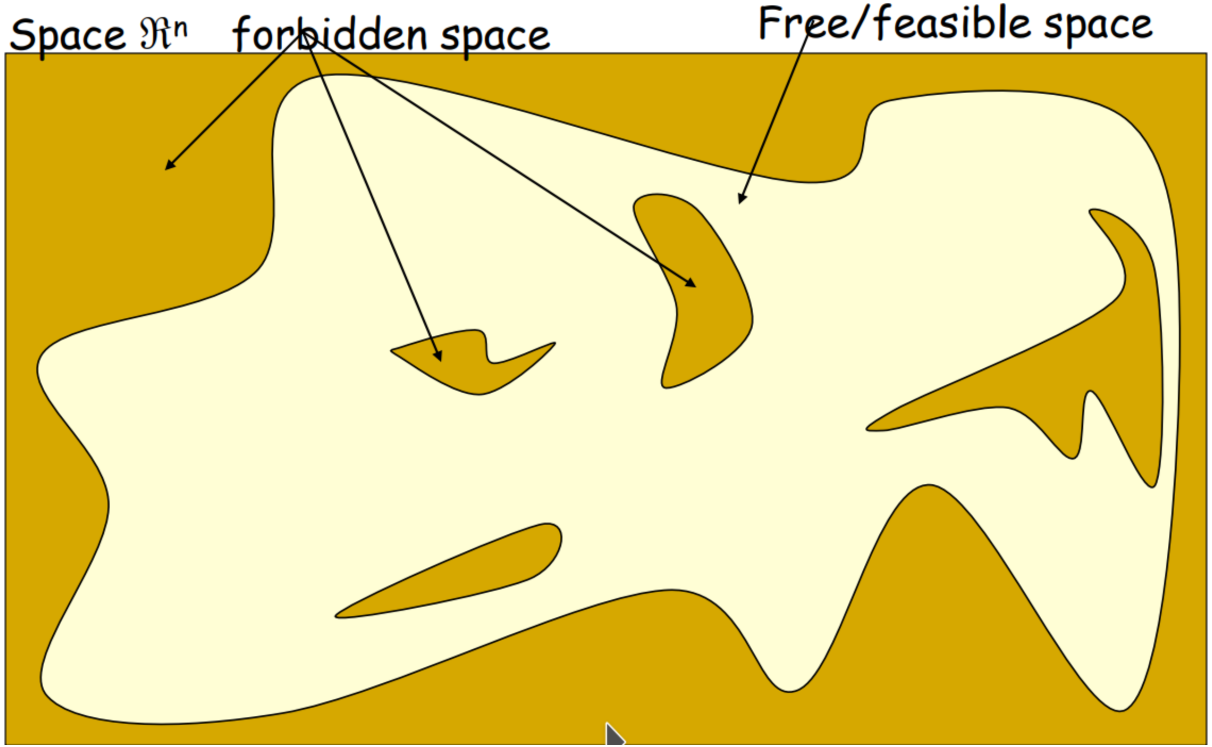
# Alternative: Sampling

- We ”control” the number of collision checks through a sampling process on the configuration space

- Most significant algorithms:
  - Probabilistic roadmaps (PRM)
  - Rapidly-exploring random trees (RRT)

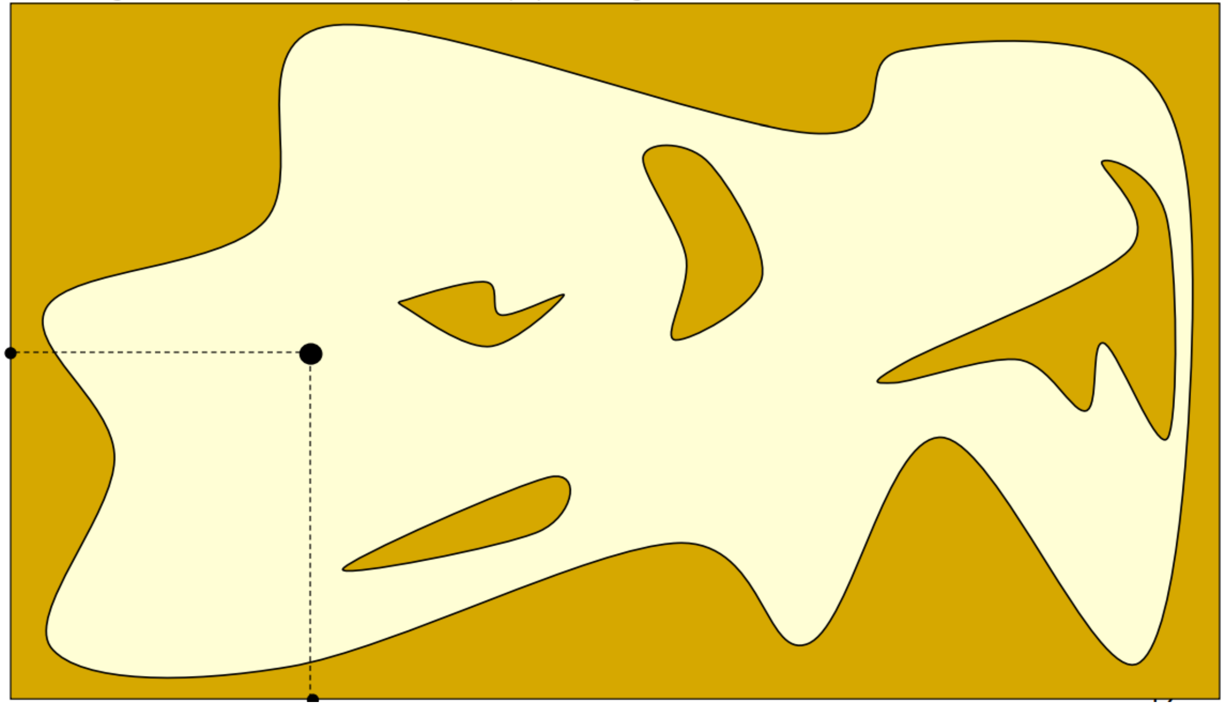


# Probabilistic Roadmap (PRM)



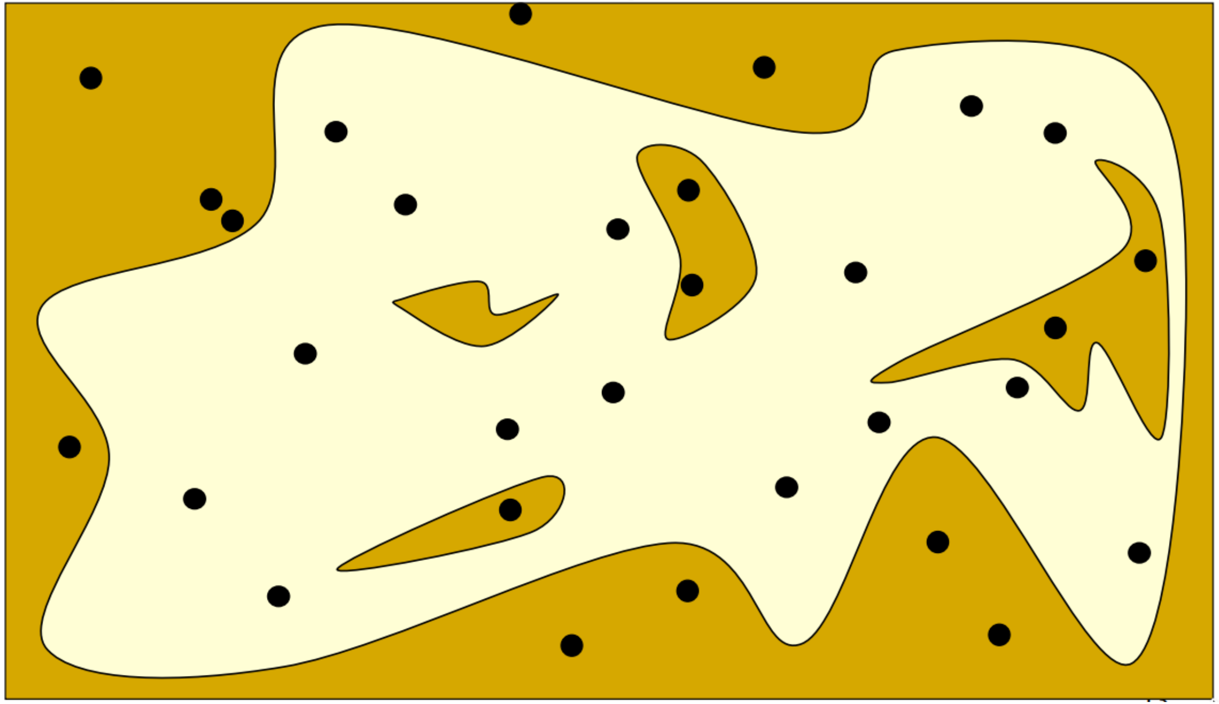
# Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



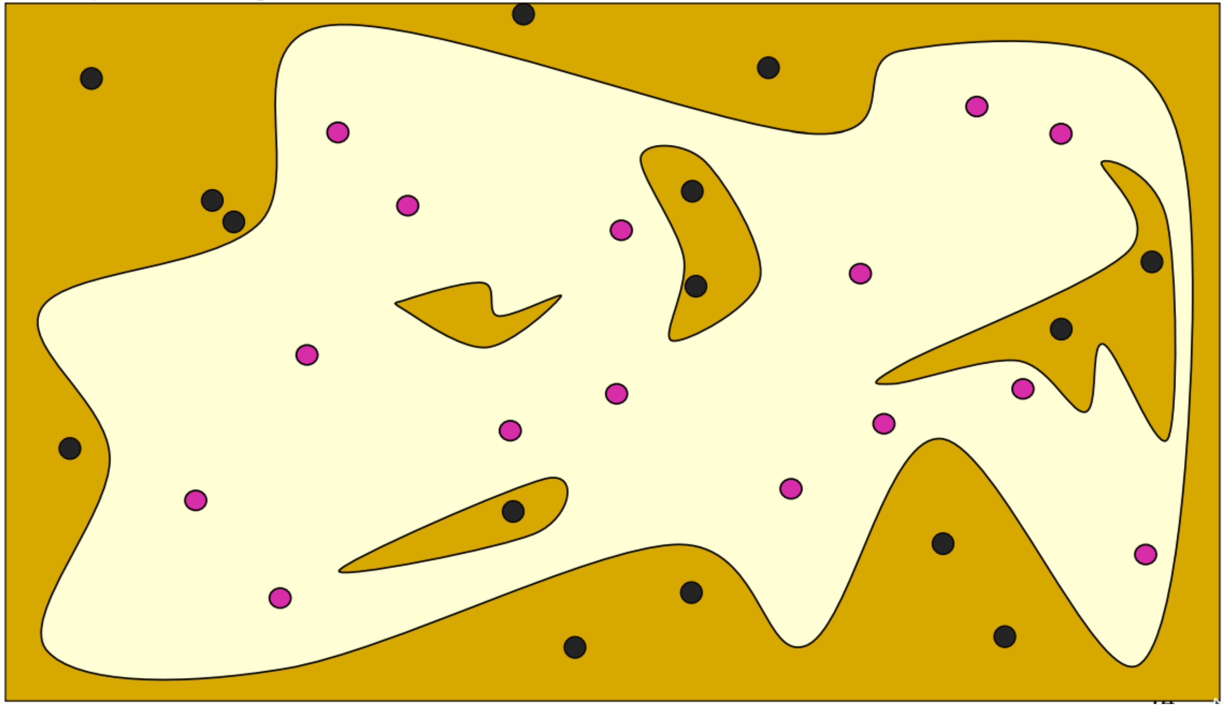
# Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



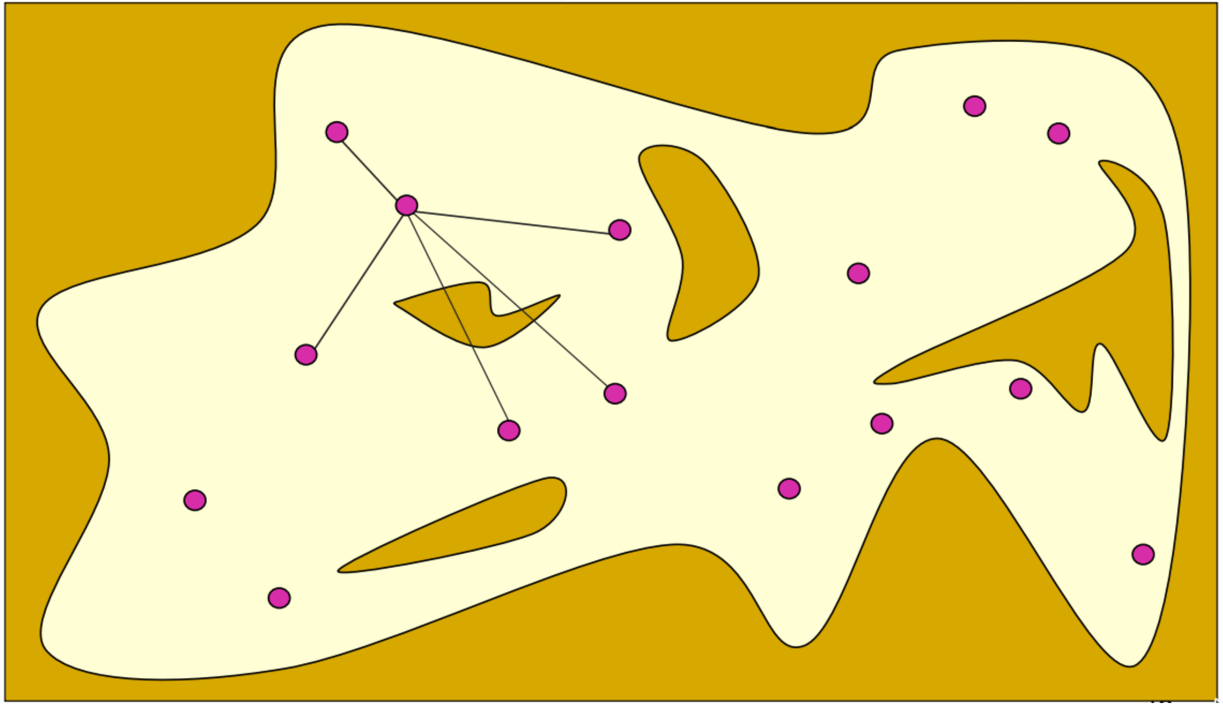
# Probabilistic Roadmap (PRM)

Sampled configurations are tested for collision



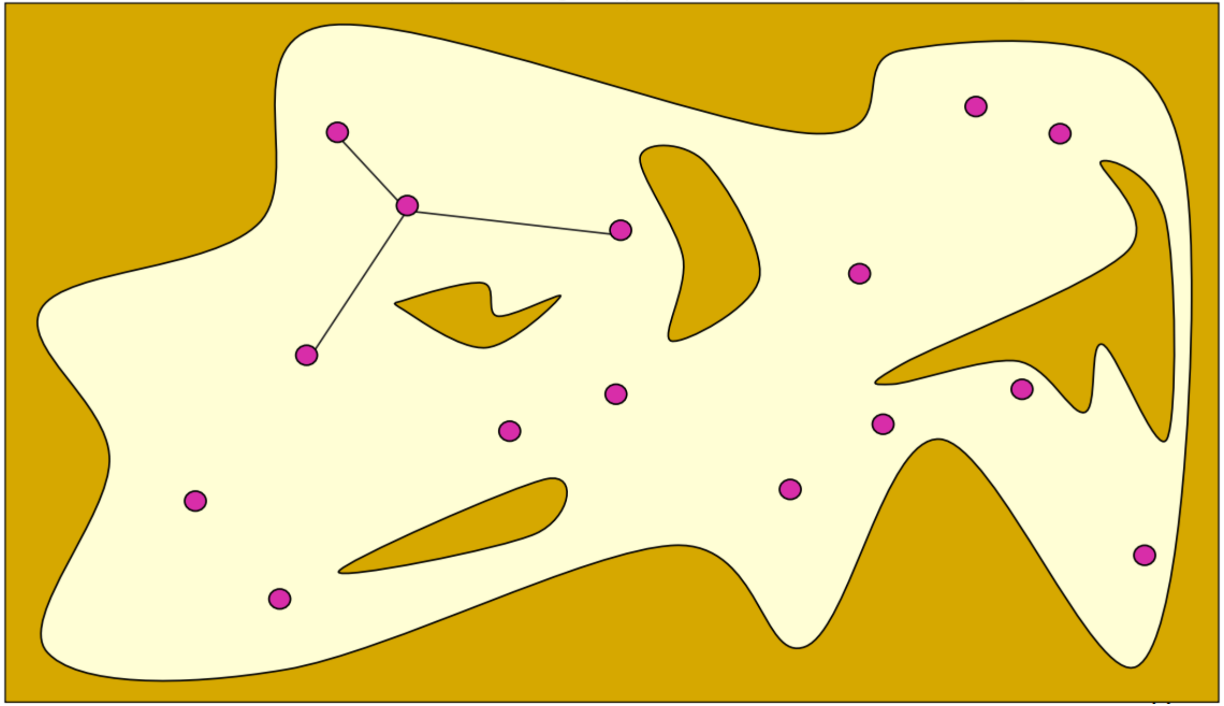
# Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



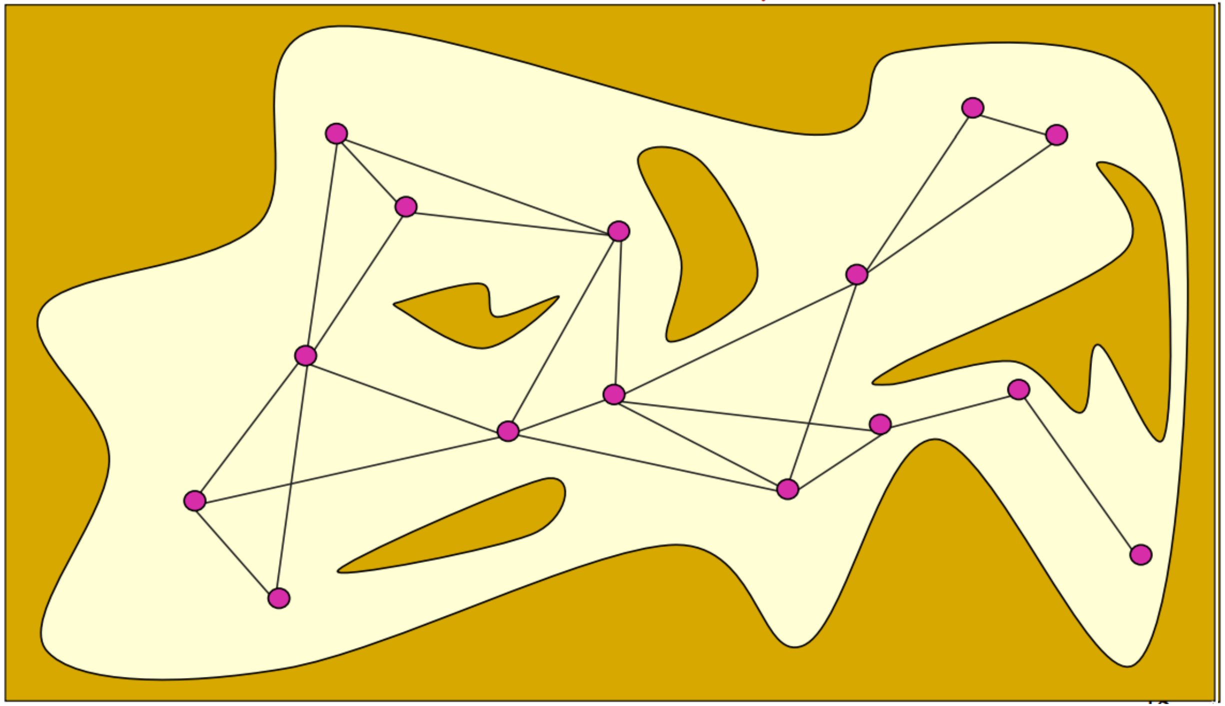
# Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



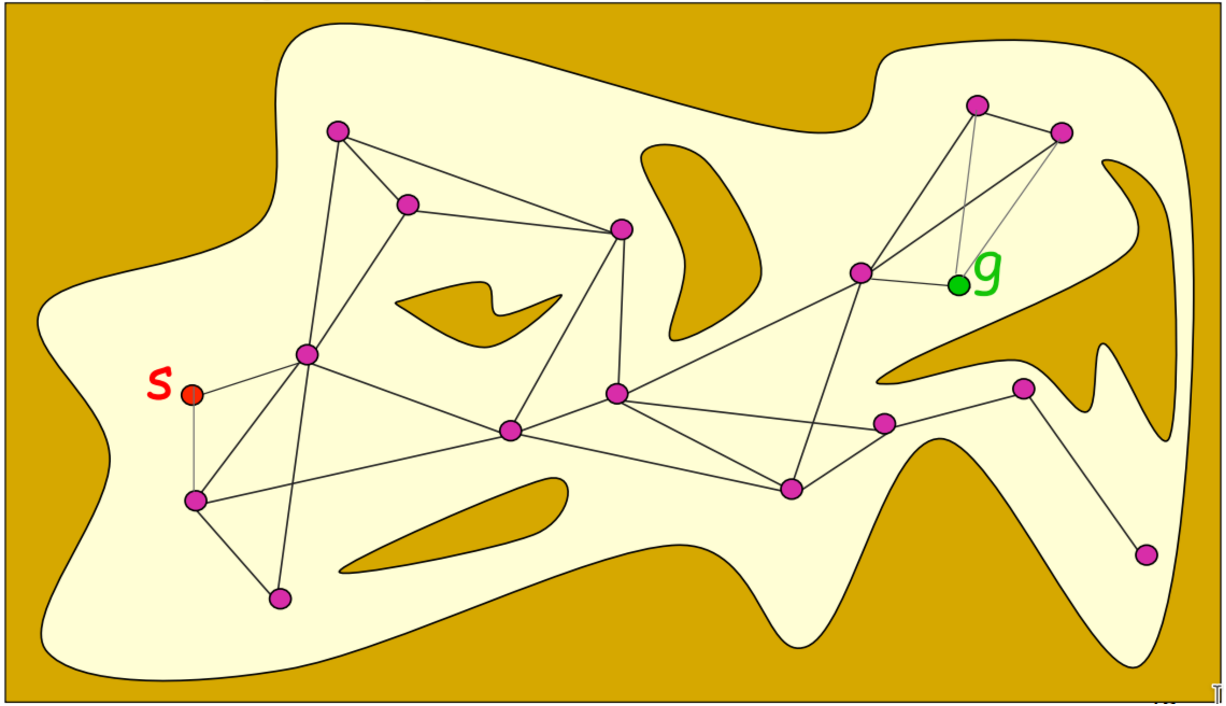
# Probabilistic Roadmap (PRM)

The collision-free links are retained as local paths to form the PRM



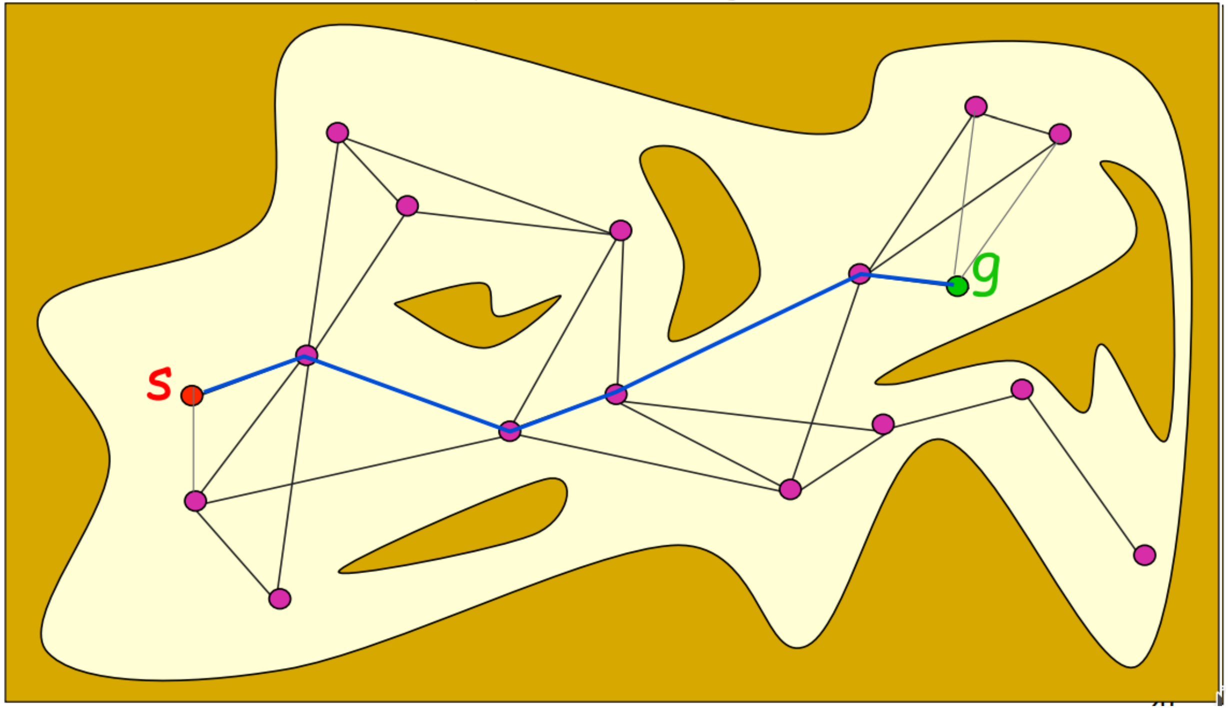
# Probabilistic Roadmap (PRM)

The start and goal configurations are included as milestones



# Probabilistic Roadmap (PRM)

The PRM is searched for a path from s to g



# Probabilistic Roadmap (PRM)

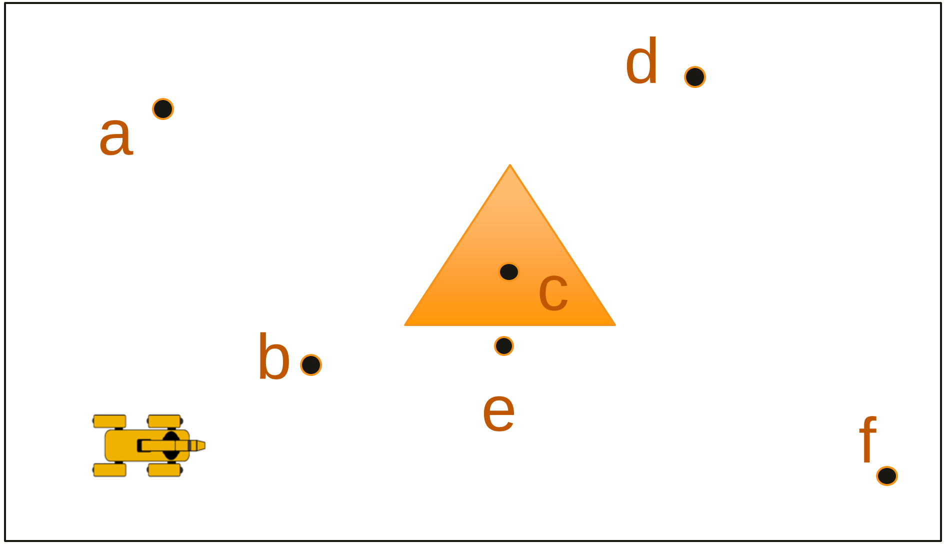
- Build Phase:
  - Randomly sample points in configuration-space. Reject the ones in collision
  - Connect points:
    - We can limit length of the connections: within a ball D
    - We can limit the number of connections per point: at most K connections
    - But we only connect them if the direct line is collision-free
  - Done!

- Query Phase:
  - Add start and goal nodes
  - Run graph search (A* / Dijkstra / ... )

https://pollev.com/robertomartinmartin739

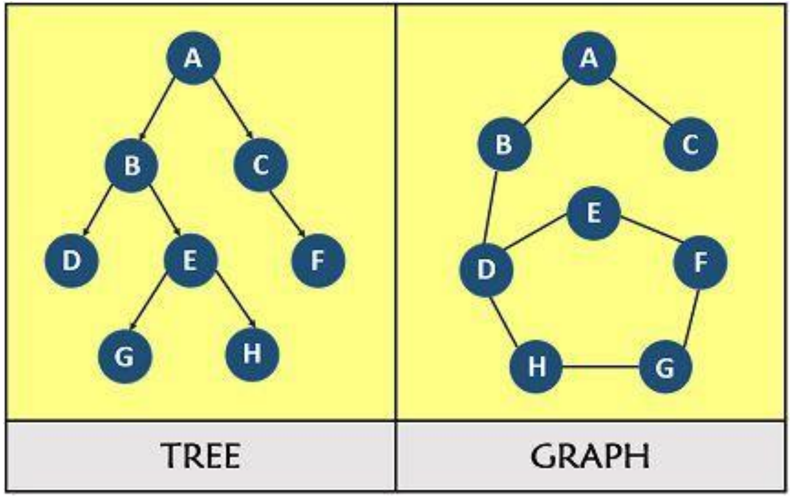
# Exercise – PRM

- Given the problem of the map (no limit in connection length or number of connections)



# Rapidly-Exploring Random Trees (RRT)

- What is a tree?
  - A type of graph that:
    - it is connected
    - doesn't have cycles
    - has a single node that is the root (has no parents)



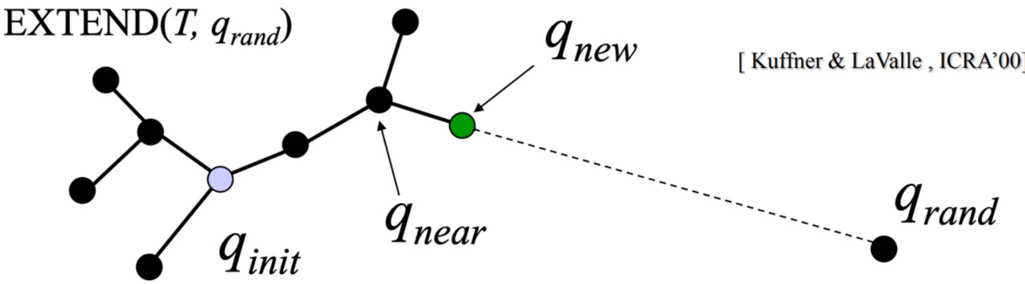
# Rapidly-Exploring Random Trees (RRT)

- Sample a random point in the space
- Create a new node in the tree going from the closest node to the random point towards the point
- Check if the new node is in collision
  - Yes: go to the next iteration
  - No: add to the tree (connected to the closest node) and go to next iteration

# Rapidly-Exploring Random Trees (RRT)

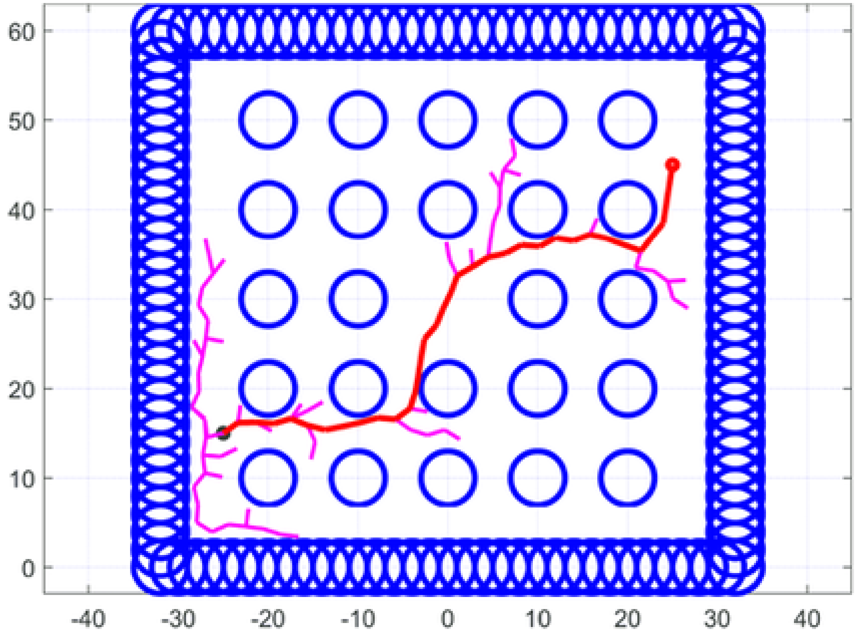
- Sample a random point in the space
- Create a new node in the tree going from the closest node (qnear) to the random point towards the point a distance of ”step size” (ε)
- Check if the new node is in collision
  - Yes: go to the next iteration
  - No: add to the tree (connected to the closest node) and go to next iteration

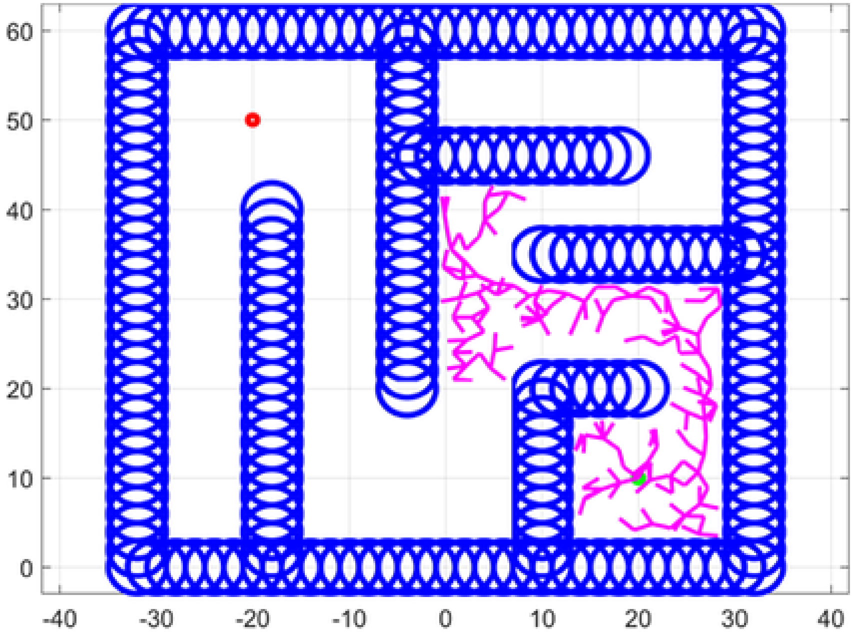
```
BUILD_RRT (q_init) {
    T.init(q_init);
    for k = 1 to K do
        q_rand = RANDOM_CONFIG();
        EXTEND(T, q_rand)
}
```



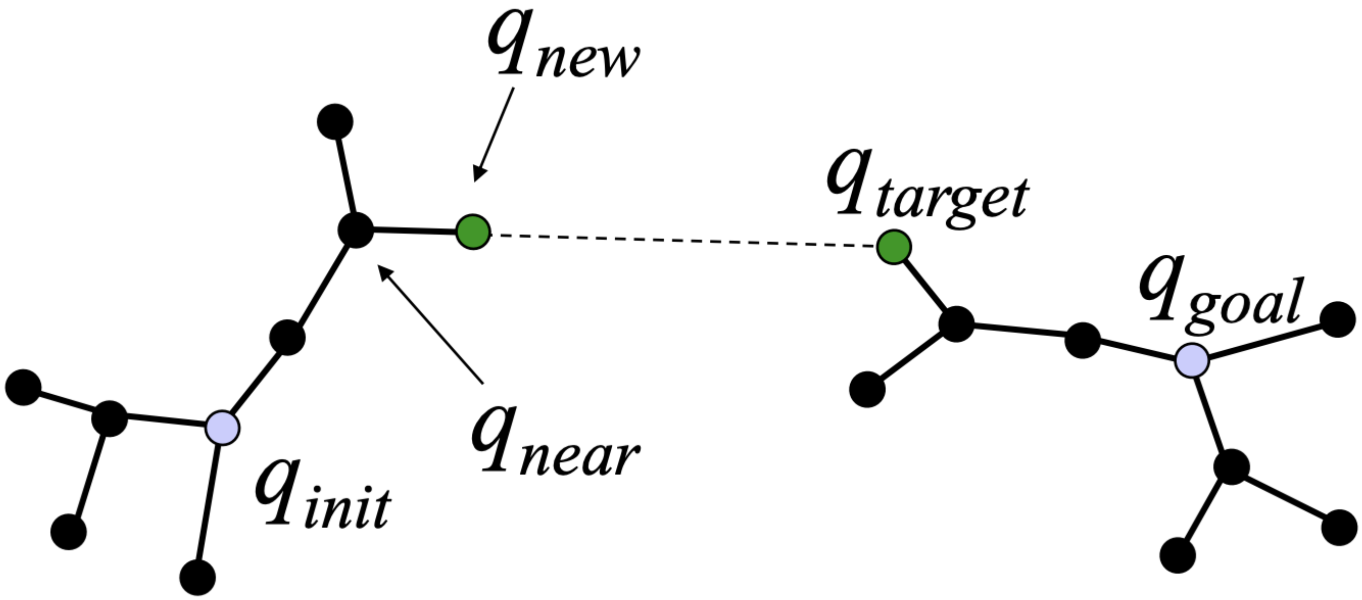

[ Kuffner & LaValle , ICRA'00]

# Variant 1: Goal-Biased RRT



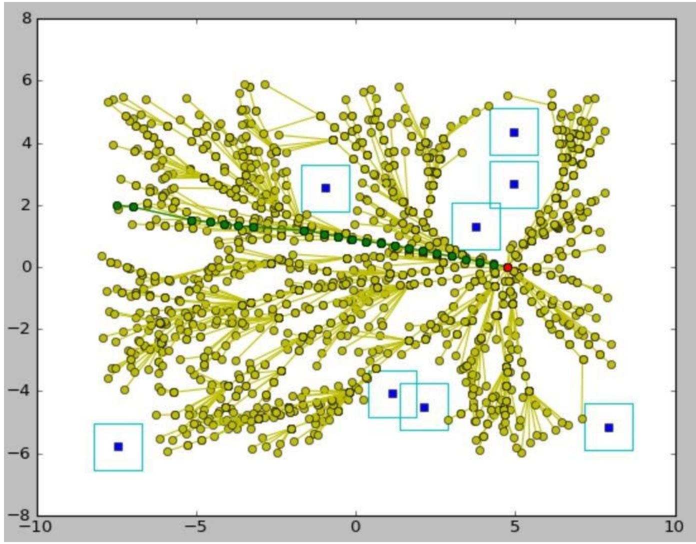



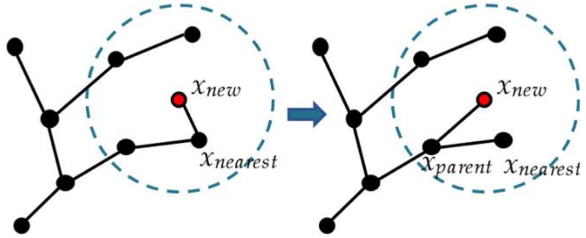
# Variant 2: Bidirectional RRT / RRT-Connect

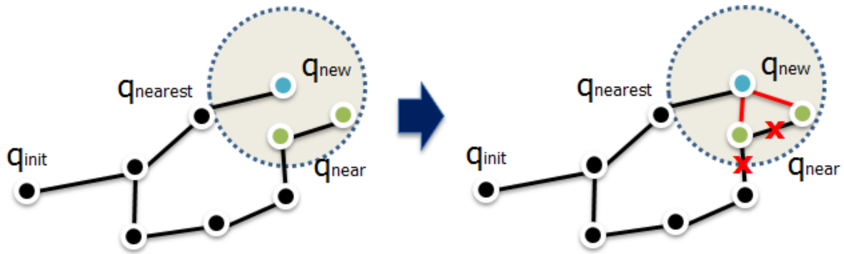


# Variant 3: RRT*

- Computes and stores the distance from each node to the parent node
- RRT* algorithm rewires the tree when it finds new connections
- Finds optimal paths (after infinite time...)!



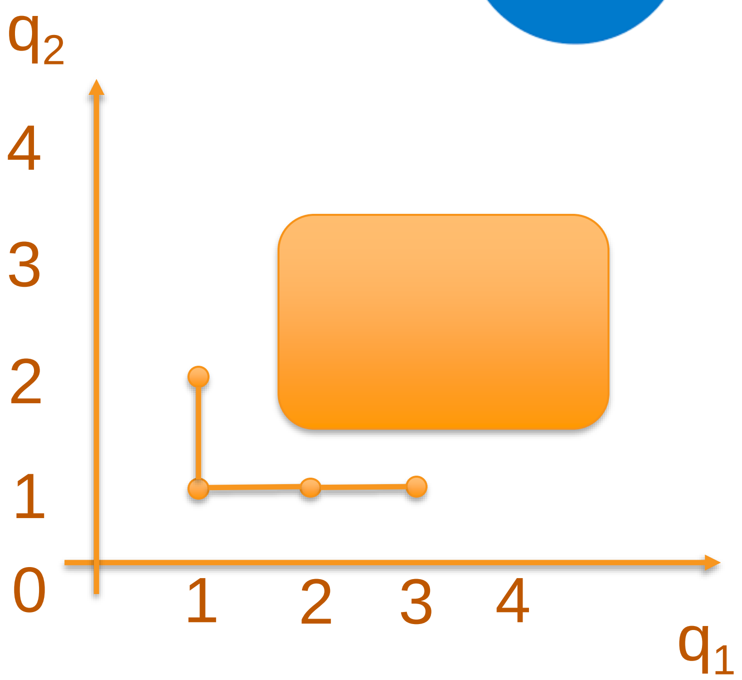



https://pollev.com/robertomartinmartin739

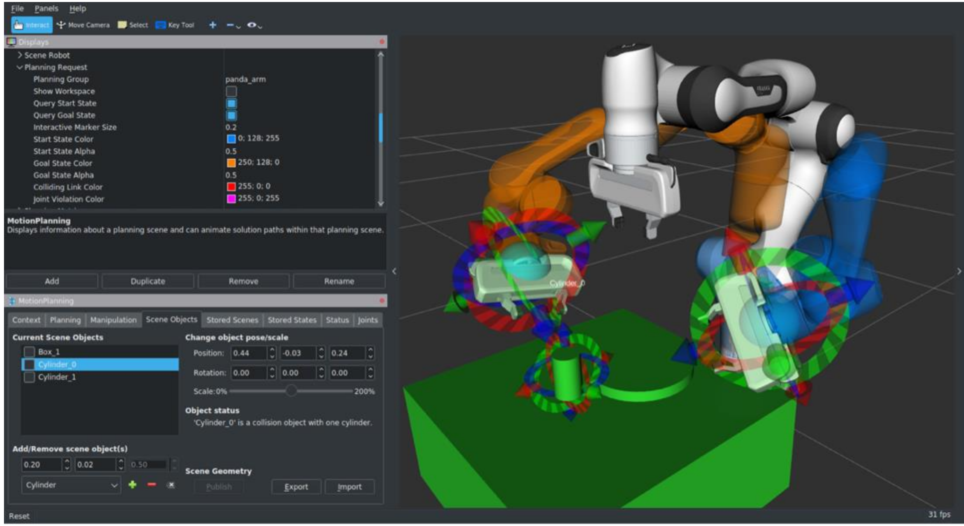
# Exercise – RRT

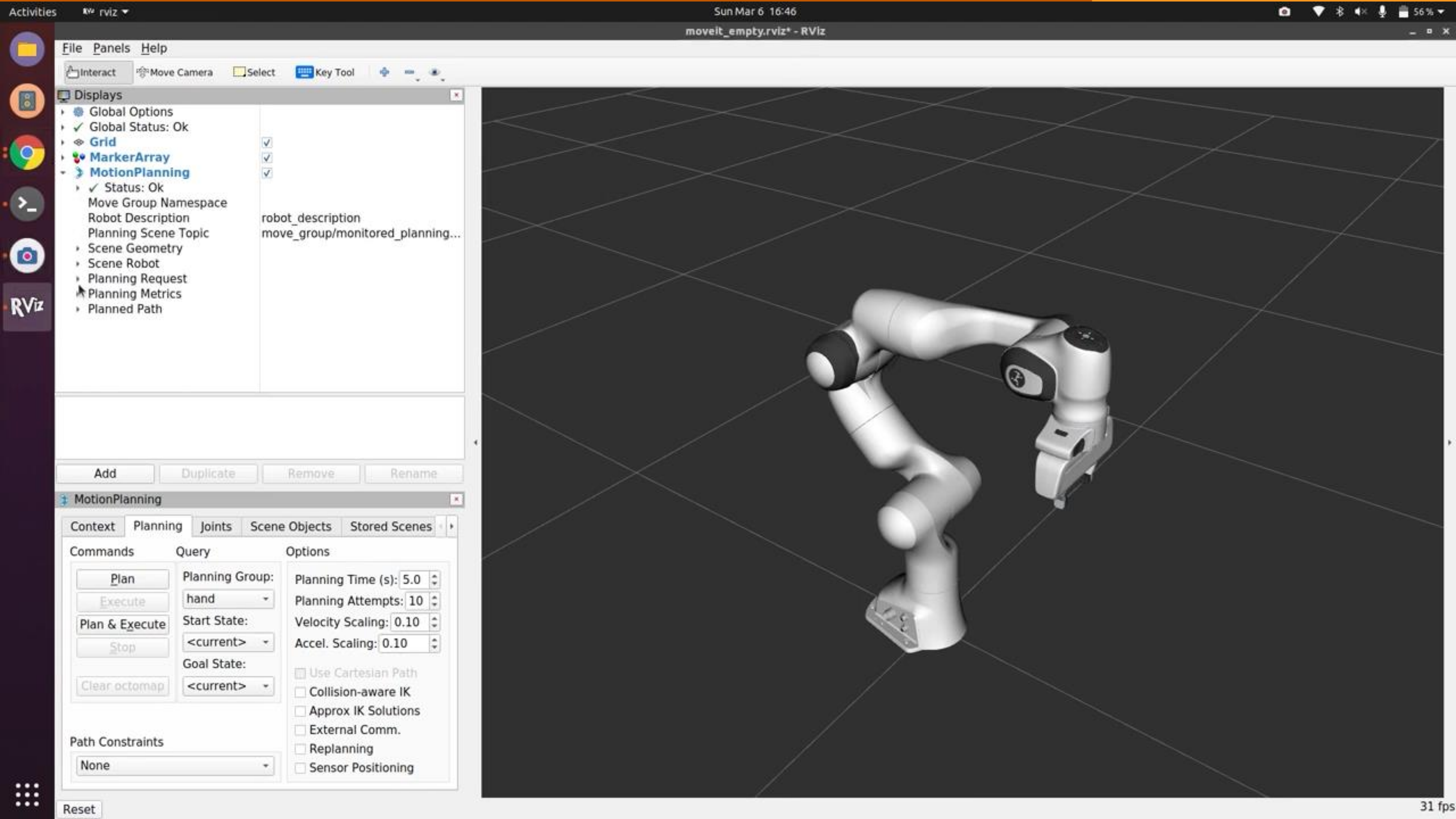
- Given the problem of the map



# A user-guide to motion planning

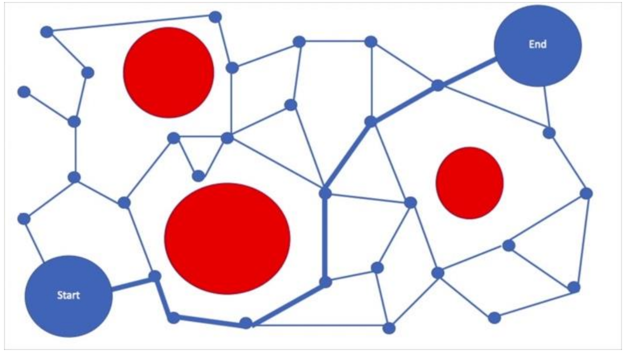
- ROS – MoveIt
  - Library of motion planning algorithms
  - Connection to the ROS environment
    - Get configuration (state) of a robot
    - Uses the viewer from ROS
    - Can execute the plan with a trajectory controller
    - Connects to sensors




Activities
rviz
Sun Mar 6 16:46
56 %
moveit_empty.rviz* - RViz
File
Panels
Help
Interact
Move Camera
Select
Key Tool
Displays
Global Options
Global Status: Ok
Grid
MarkerArray
MotionPlanning
Status: Ok
Move Group Namespace
Robot Description
robot_description
Planning Scene Topic
move_group/monitored_planning...
Scene Geometry
Scene Robot
Planning Request
Planning Metrics
Planned Path
Add
Duplicate
Remove
Rename
MotionPlanning
Context
Planning
Joints
Scene Objects
Stored Scenes
Commands
Query
Options
Plan
Execute
Plan & Execute
Stop
Clear octomap
Planning Group:
hand
Start State:
<current>
Goal State:
<current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
Use Cartesian Path
Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning
Path Constraints
None
Reset
31 fps

# Recap

- For large search spaces, discretizing and checking all cells for collisions is unfeasible
- We use sampling (random picked locations) instead
- PRM
  - We sample random locations and connect them. Then use a graph search algorithsm
- RRT
  - We sample at each step and grow the tree until we find a path




- Next: but how do we execute a path that is a sequence of states?