Maps and Graphs

Fast Robots, ECE4160/5160, MAE 4190/5190

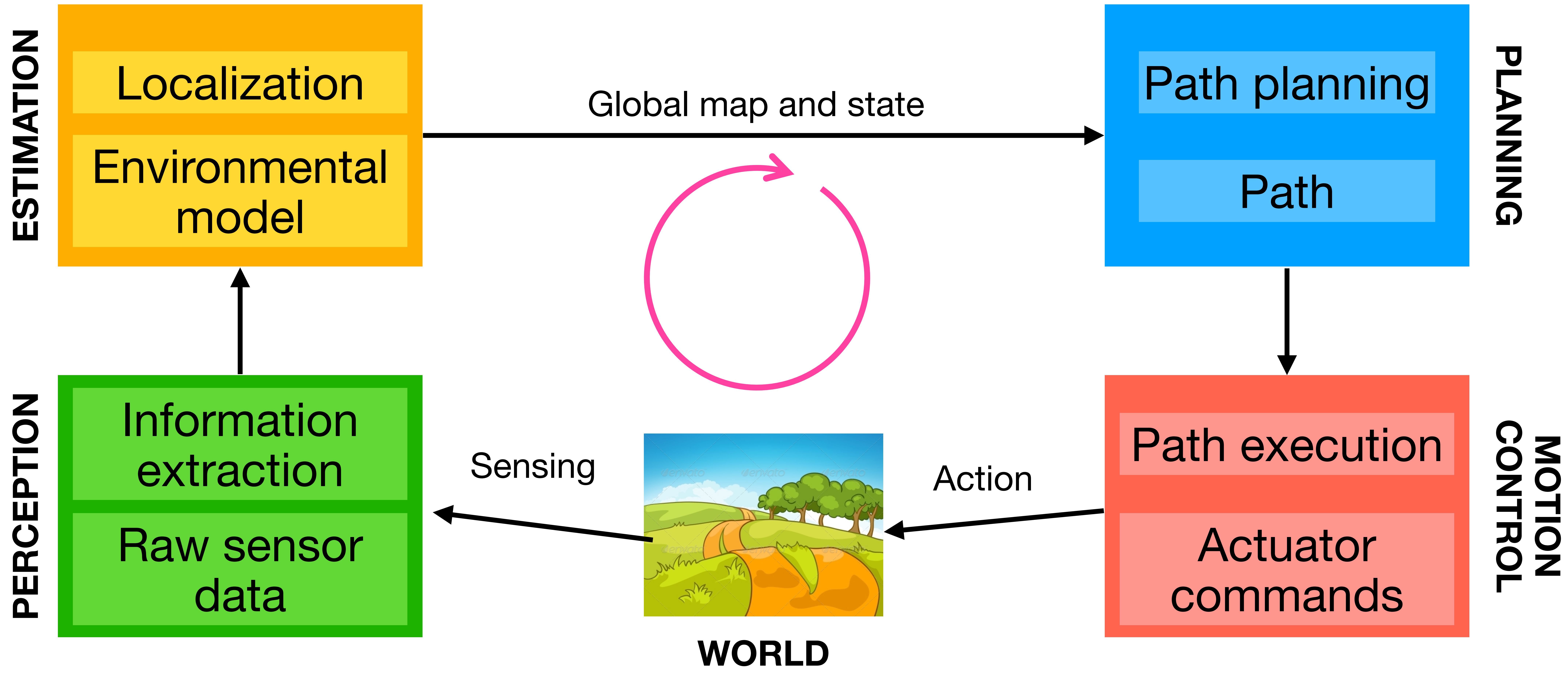
E. Farrell Helbling, 3/13/25

Class Action Items

- **Midsemester evaluations**: please fill them out! One point in your final grade is based on completing the midterm course evaluations.
  Open until March 16th!
  - They remain anonymous, I get netIDs from MTEI
- Lab 1 and Lab 2 regrade requests will close on **Sunday midnight.**
- Lab 3 regrade requests will close on **Thursday midnight.**
- Lab 6 how is it going?
- Please start Lab 7 early!
- If you get to the stunts lab next week, you definitely will not have to think about it over Spring Break!

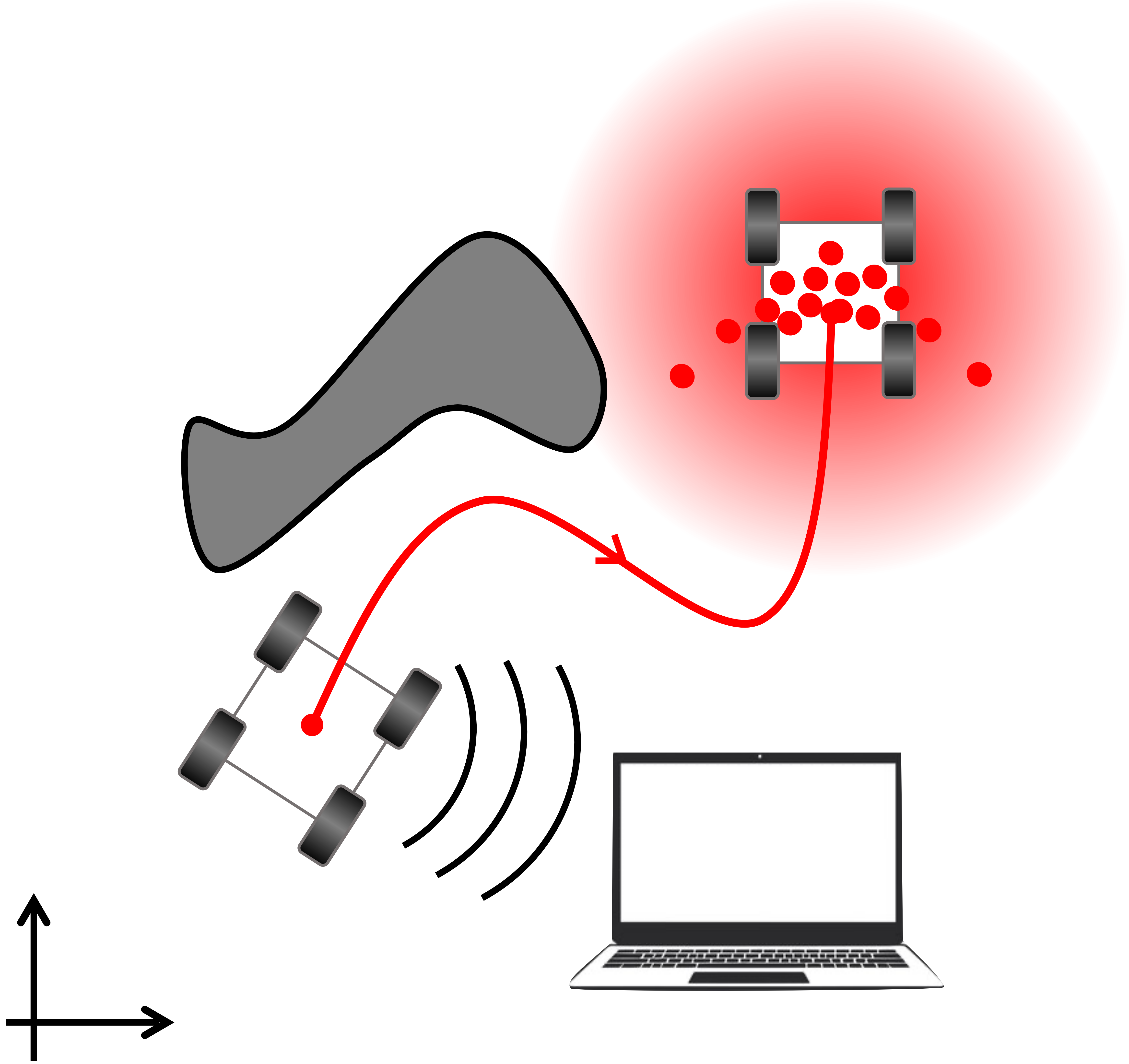
Navigation

- **Break the problem down:** localization, map building, path planning



Navigation

- Local planners
- Global localization and planning
  - Map representations
    - Continuous
    - Discrete
    - Topological
  - Maps as graphs
    - Graph search algorithms
      - Breadth first search
      - Depth first search
      - Dijkstras
      - A*



Local Planners

Local path planning/ obstacle avoidance

- Use goal position, recent sensor readings, and relative position of robot to goal
    - Can be based on a local map
    - Often implemented as a separate task
    - Runs at a much faster rate than the global planner

- 3 examples:
    - Bug algorithms
    - Vector Field Histogram (VFH)
    - Dynamic Window Approach (DWA)

Wagner, ITS 2015

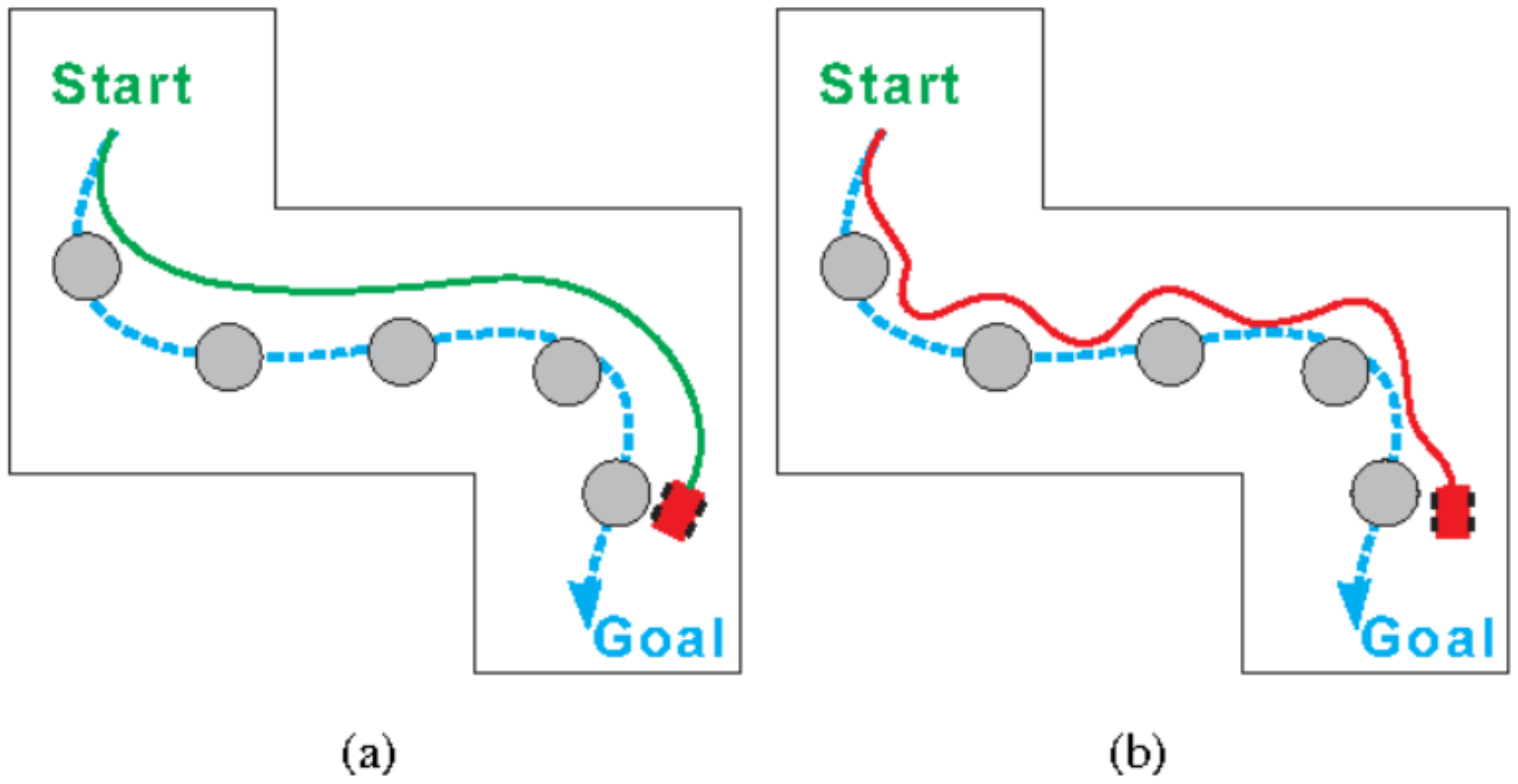
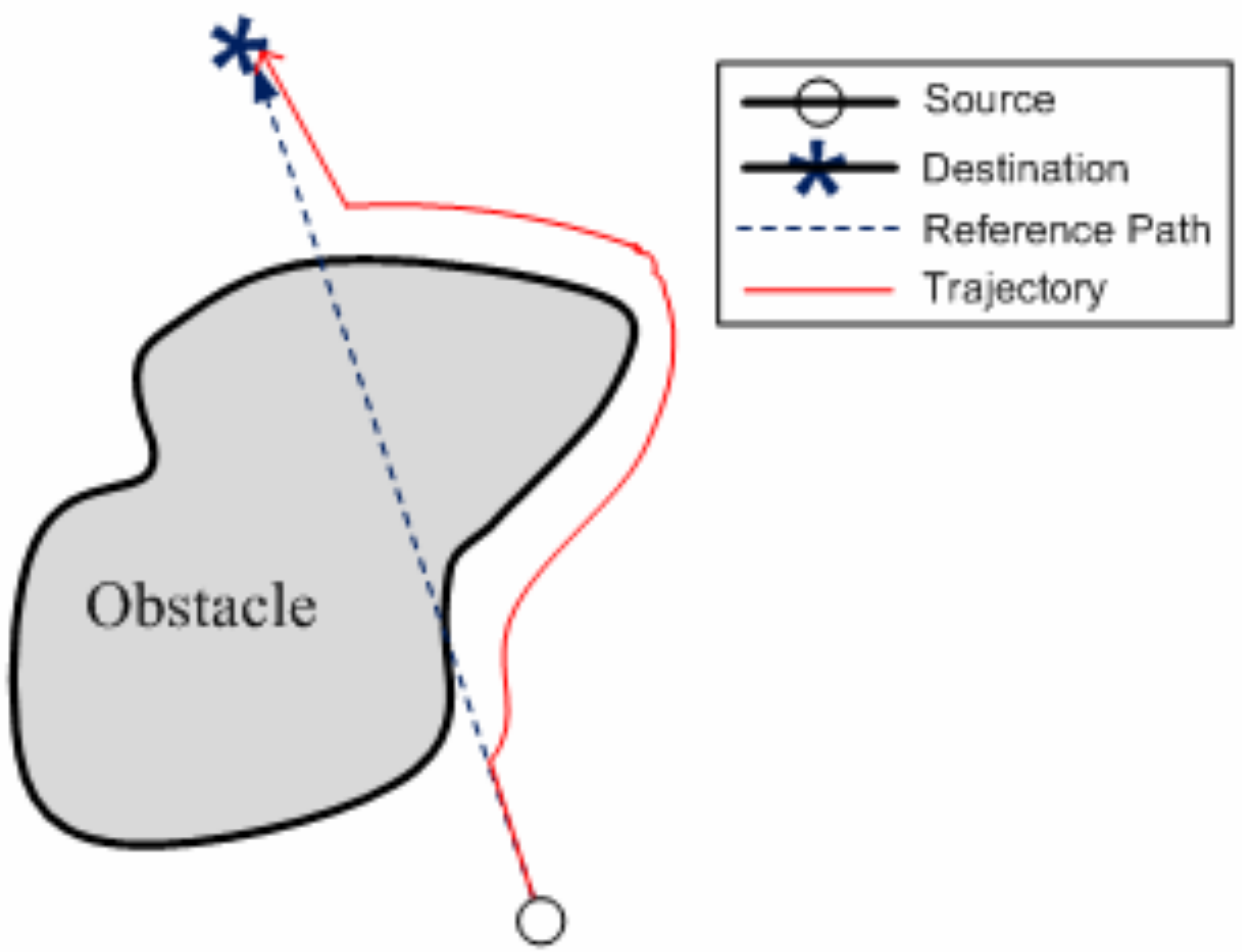


Fig. 1. Dashed blue spline is global path: a) Green spline is ideal local path; b) Red spline is actual local path

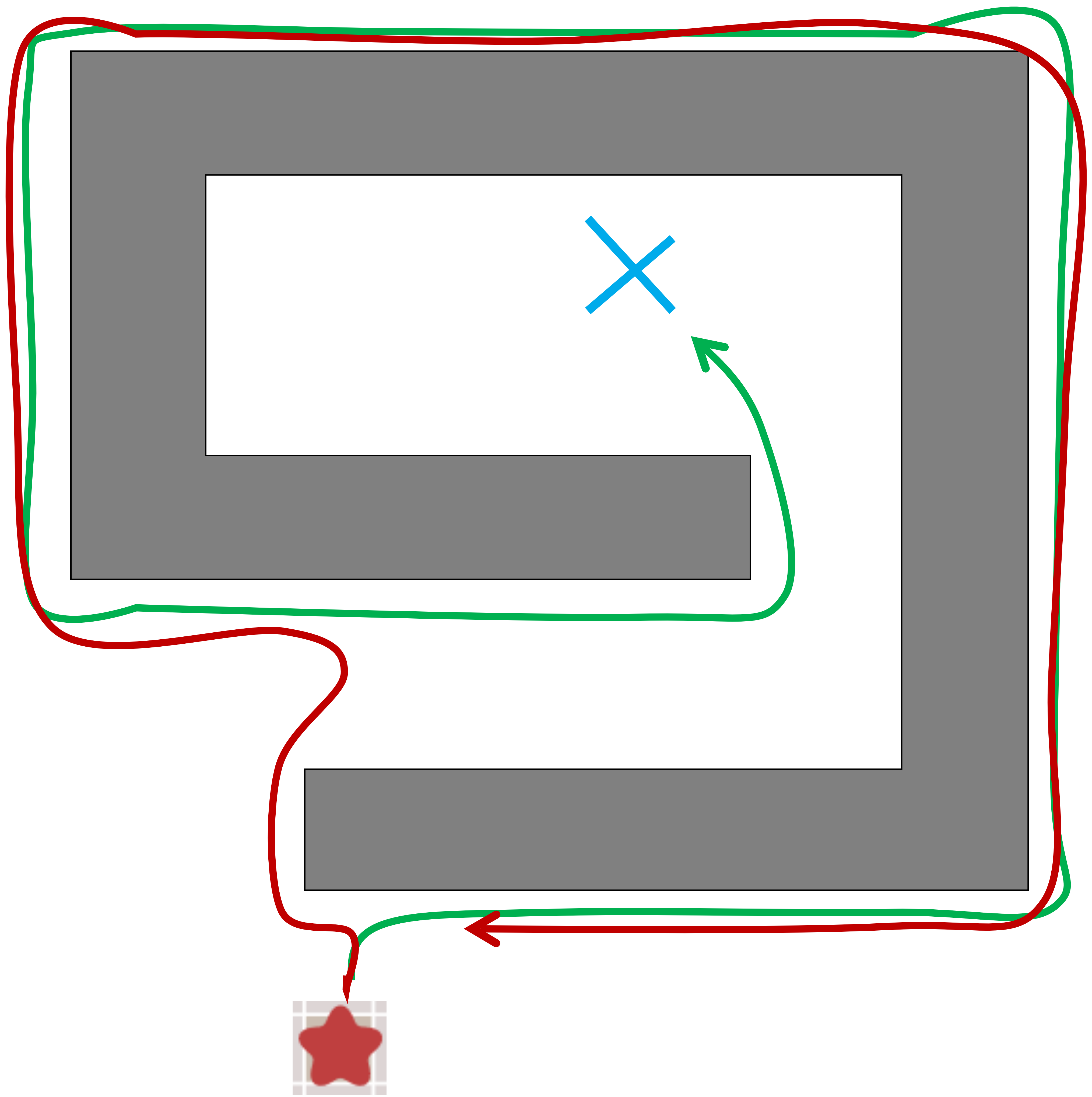
Bug algorithms

- Uses local knowledge and the direction and distance to the goal
- Basic idea
  - Follow the contour of obstacles until you see the goal
  - State 1: seek goal
  - State 2: follow wall
- Different Variants: Bug0, Bug1, Bug2
- Advantages
  - Super simple
  - No global map
  - Completeness
- Disadvantages
  - Suboptimal



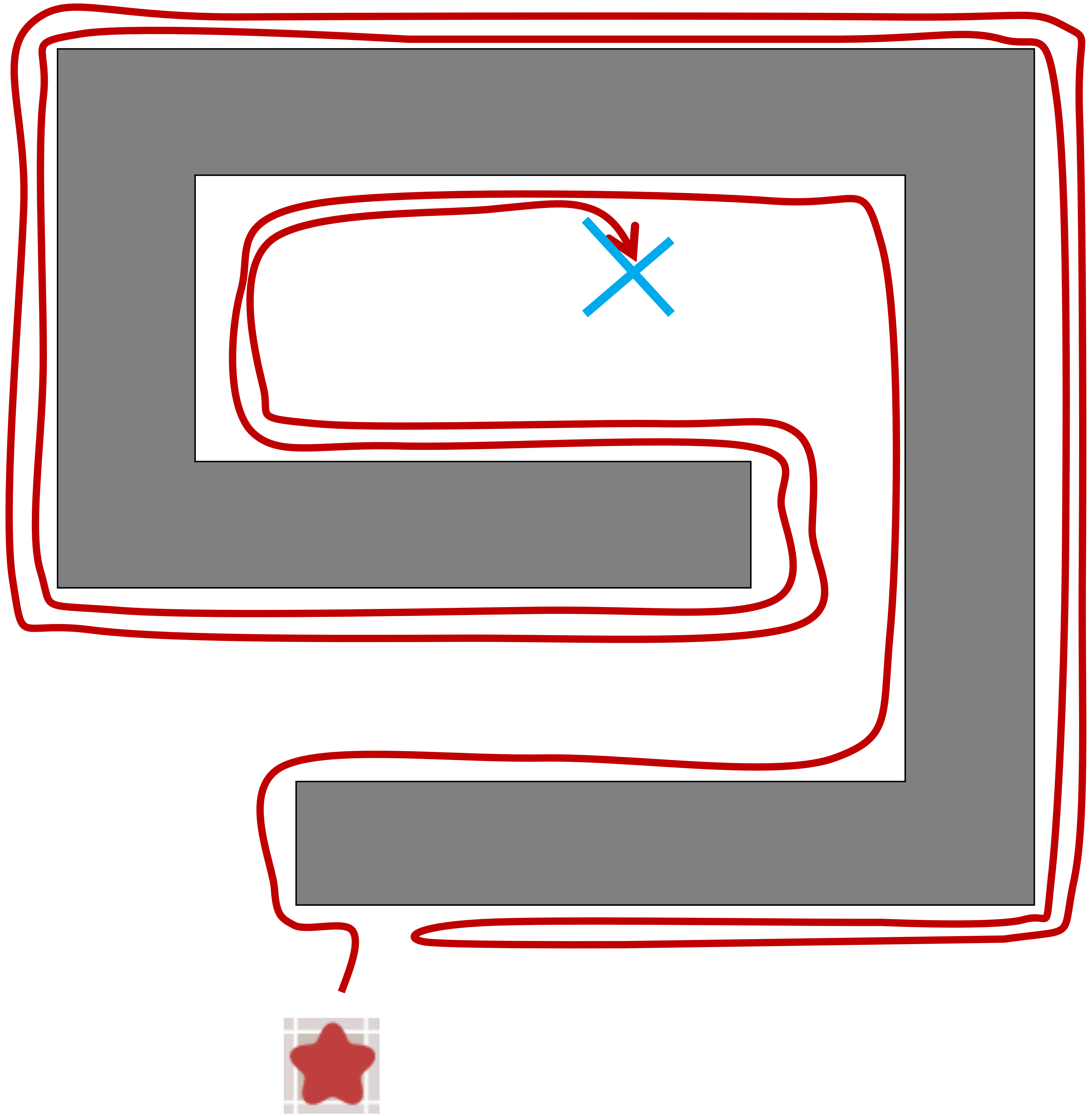
Bug 0

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
- Algorithm
  - Go towards goal
  - Follow obstacles until you can go towards goal again
  - Loop



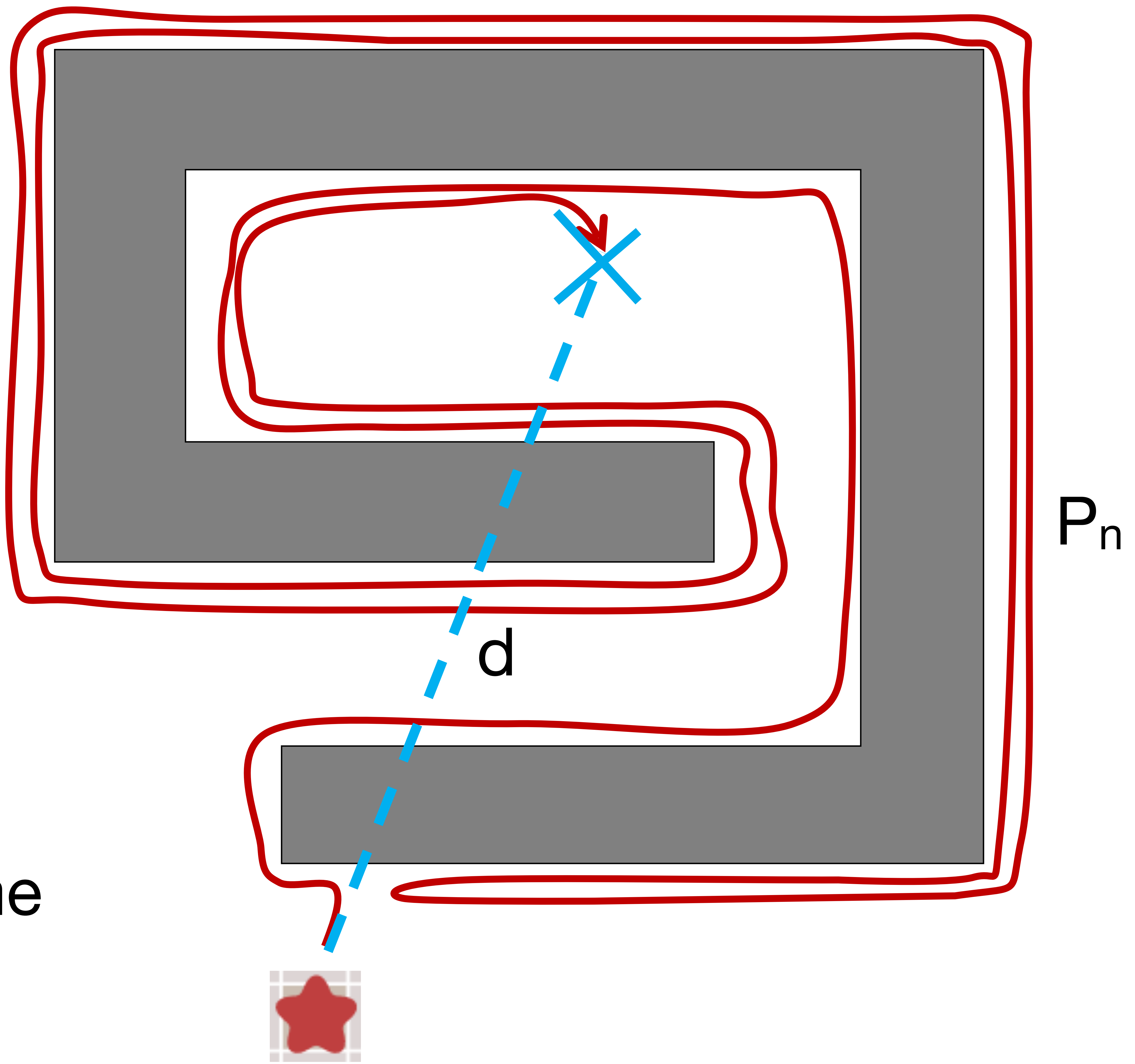
Bug 1

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
- Algorithm
  - Go towards goal
  - Follow obstacles *and remember how close you got to the goal*
  - Return to the closest point, loop



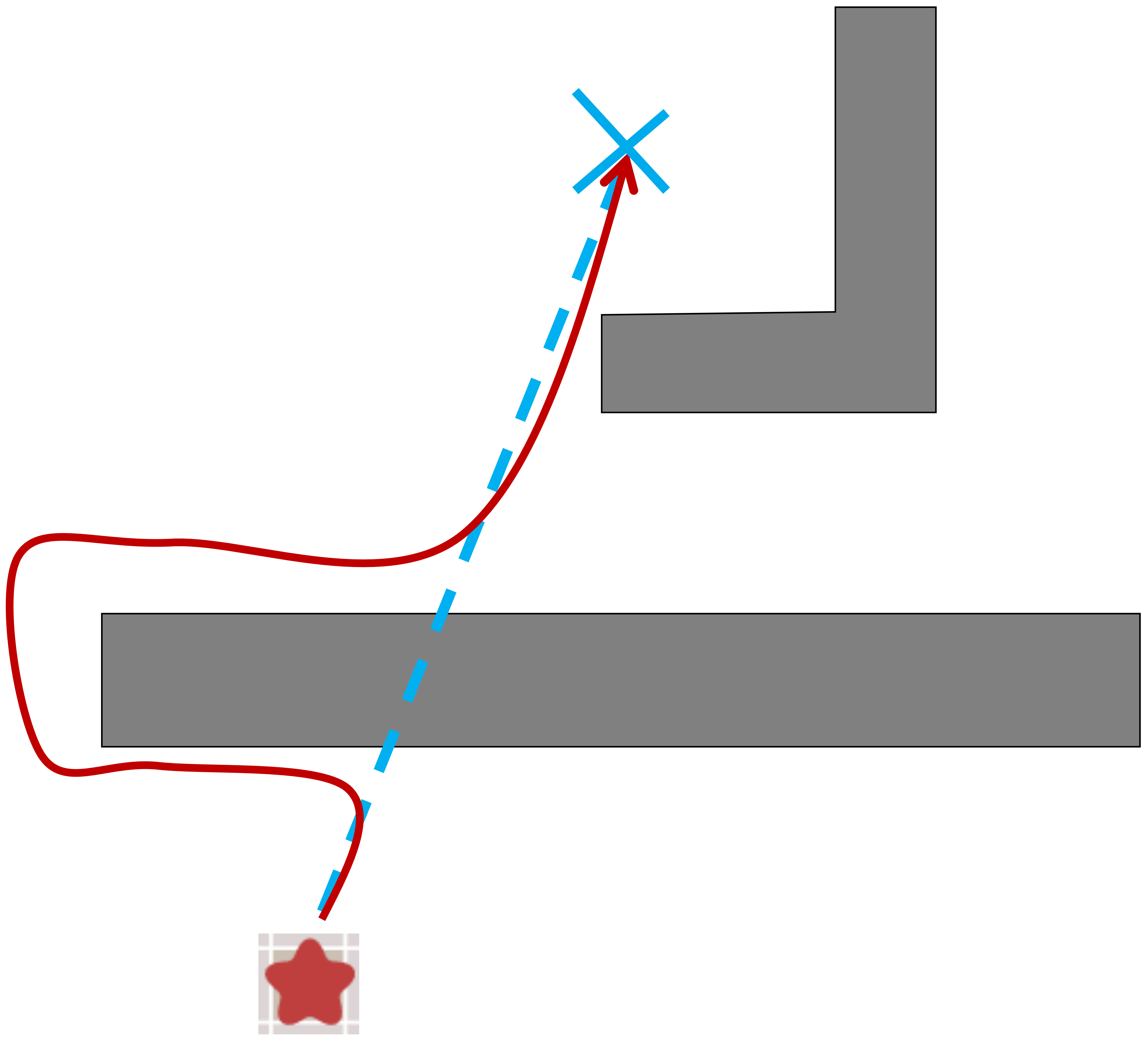
Bug 1 - formally

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
- Lower bound traversal? **d**
- Upper bound traversal? **d + 1.5∑(P**$_n$**)**
- Pros?
  - If a path exists, it returns in finite time
  - It knows if none exist!



Bug 2

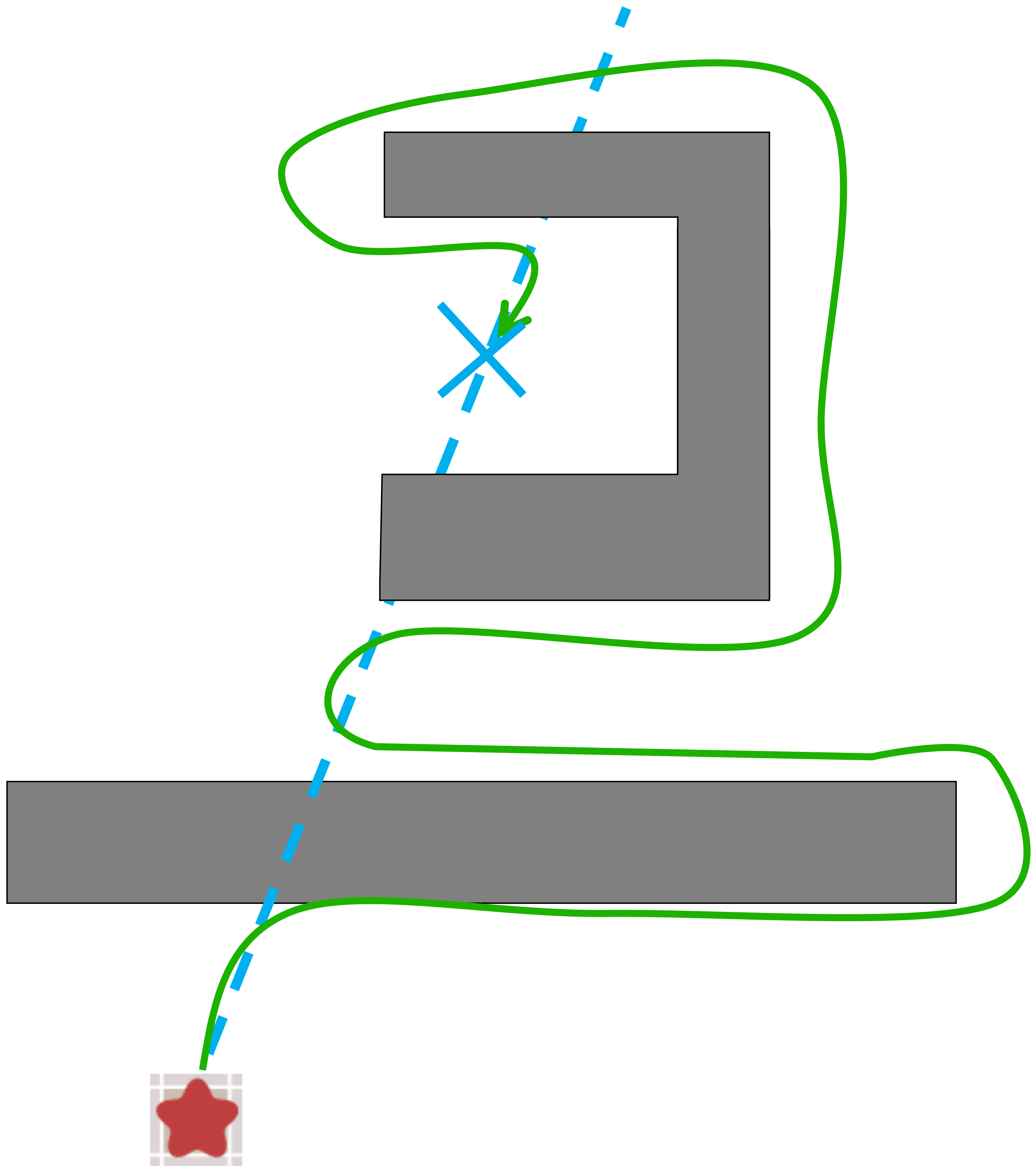
- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
  - Original vector to the goal
- Algorithm
  - Go towards goal on the vector
  - Follow obstacles *until you are back on the vector (and closer to the goal)*
  - Loop



Howie Choset 16-735

Bug 2

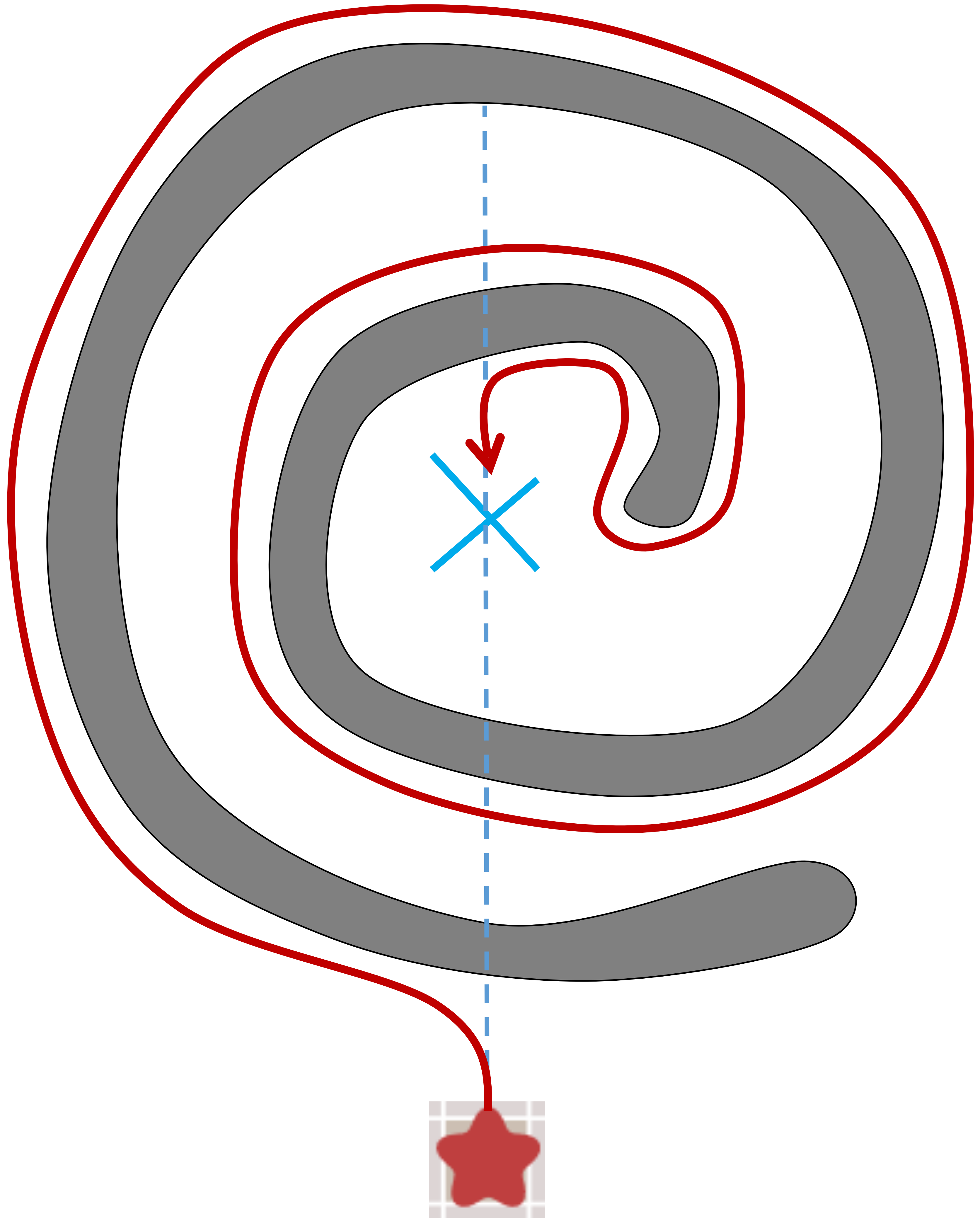
- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
  - Original vector to the goal
- Algorithm
  - Go towards goal on the vector
  - Follow obstacles *until you are back on the vector (and closer to the goal)*
  - Loop



Bug 2

- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
  - Original vector to the goal
- Algorithm
  - Go towards goal on the vector
  - Follow obstacles *until you are back on the vector (and closer to the goal)*
  - Loop

What is faster, right- or left- wall following?



Howie Choset 16-735

Battle of the bugs (1 vs 2)

Bug 1
Layout 1

Bug 1
Layout 1

https://www.youtube.com/watch?v=T2PVaKyxMmY

Battle of the bugs (1 vs 2)

Exhaustive search

Bug 1
Layout 2

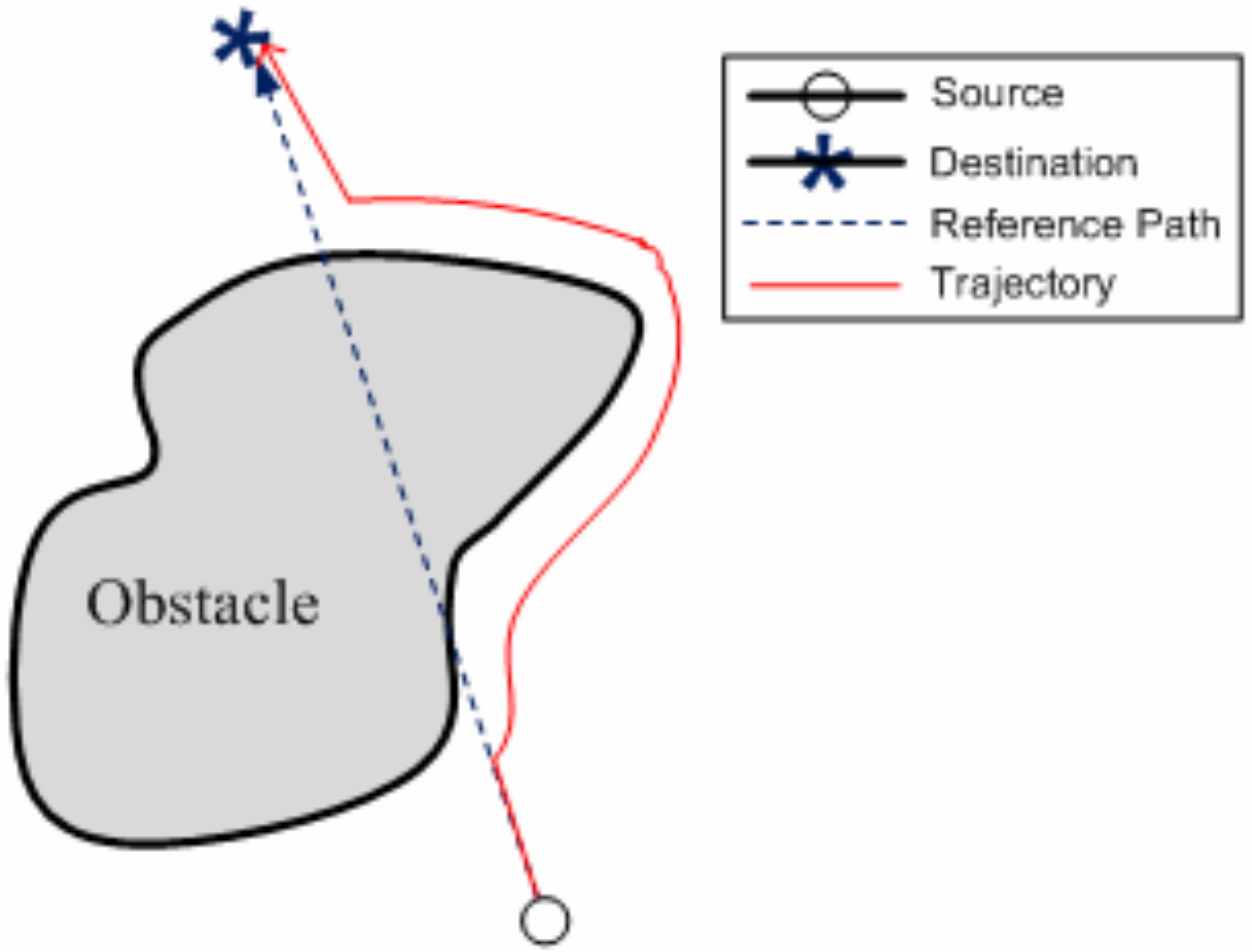
Greedy search

Bug 2
Layout 2

https://www.youtube.com/watch?v=T2PVaKyxMmY

Bug algorithms

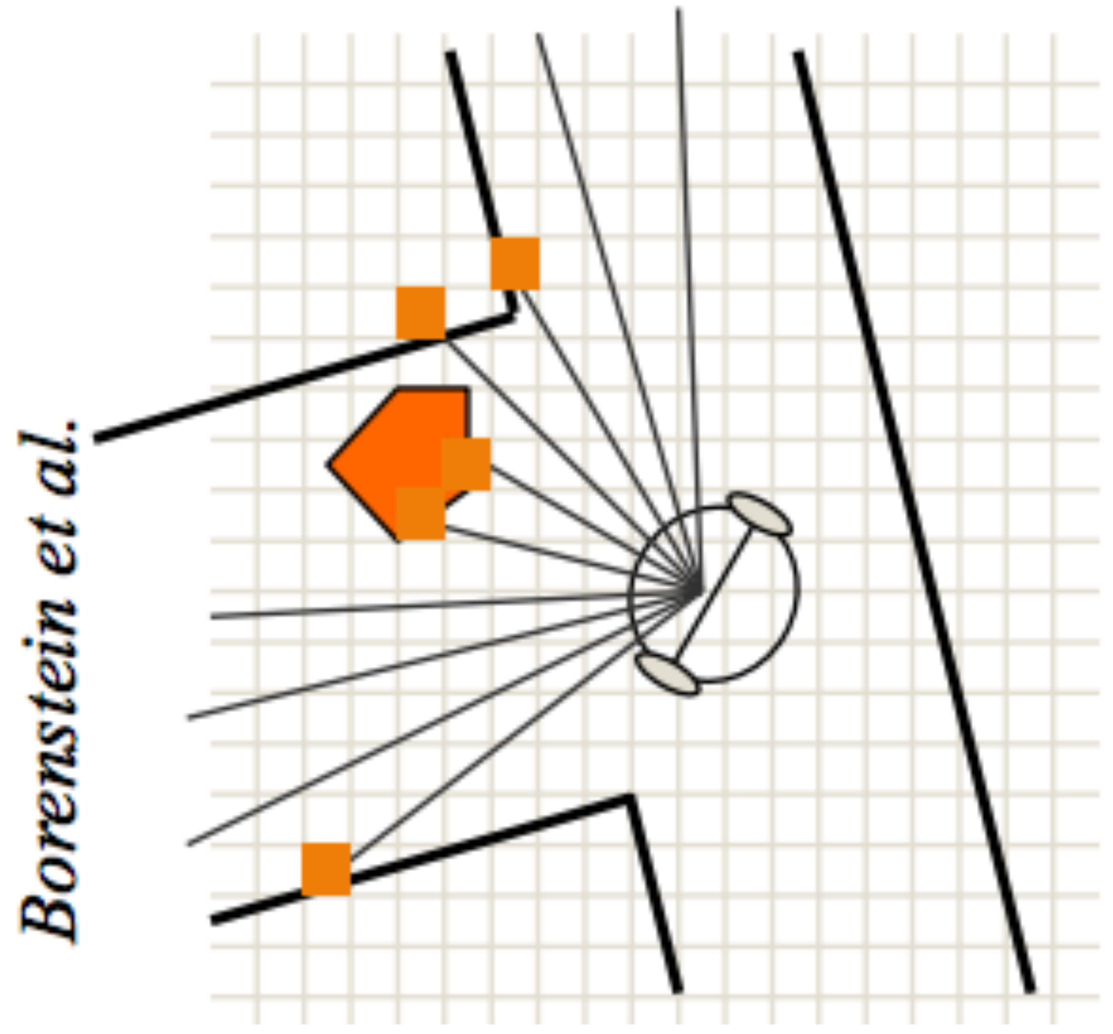
- Uses local knowledge and the direction and distance to the goal
- Basic idea
  - Follow the contour of obstacles until you see the goal
  - State 1: seek goal
  - State 2: follow wall
- Different Variants: Bug0, Bug1, Bug2

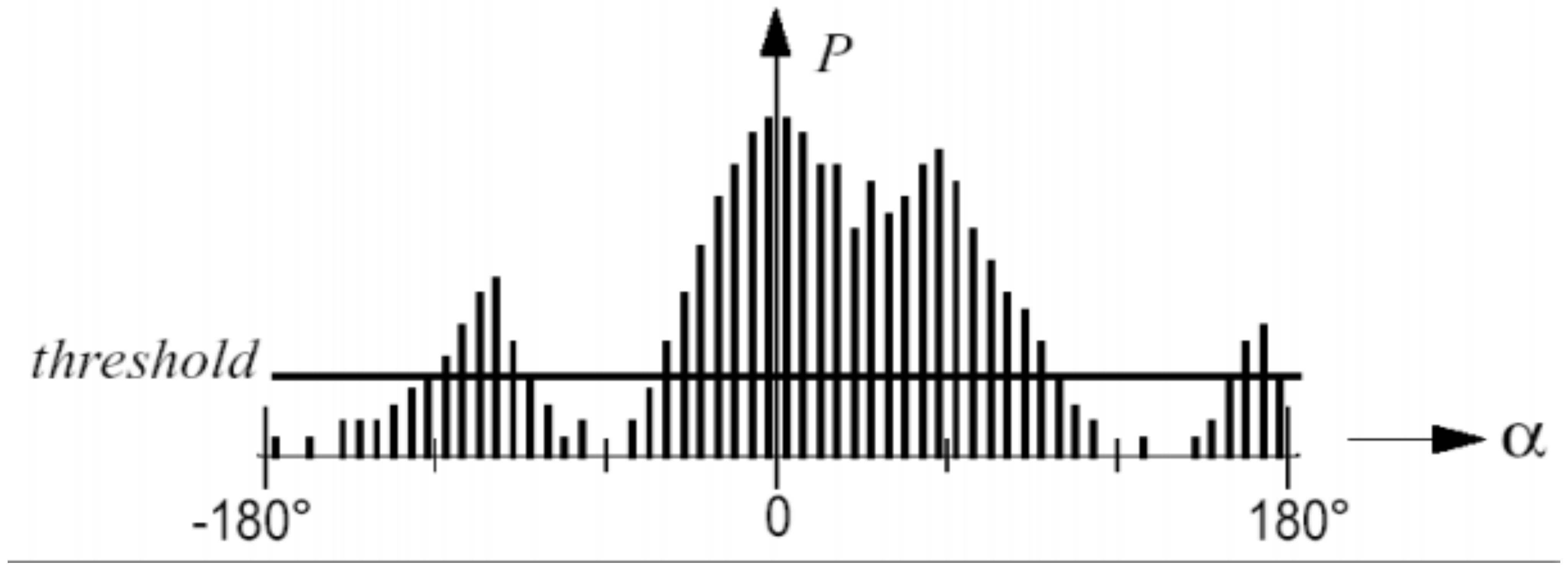


- The robot motion behavior is reactive
- Issues if the instantaneous sensor readings do not provide enough information or are noisy

Vector Field Histograms

- VFH creates a local map of the environment around the robot populated by “relatively” recent sensor readings
- Build a local 3D grid map reduce to a 1-DOF histogram
- Planning
  - Find all openings large enough for robot to pass
  - Choose the one with the lowest cost, G
  - G = a*goal_direction + b*orientation +c*prev_direction

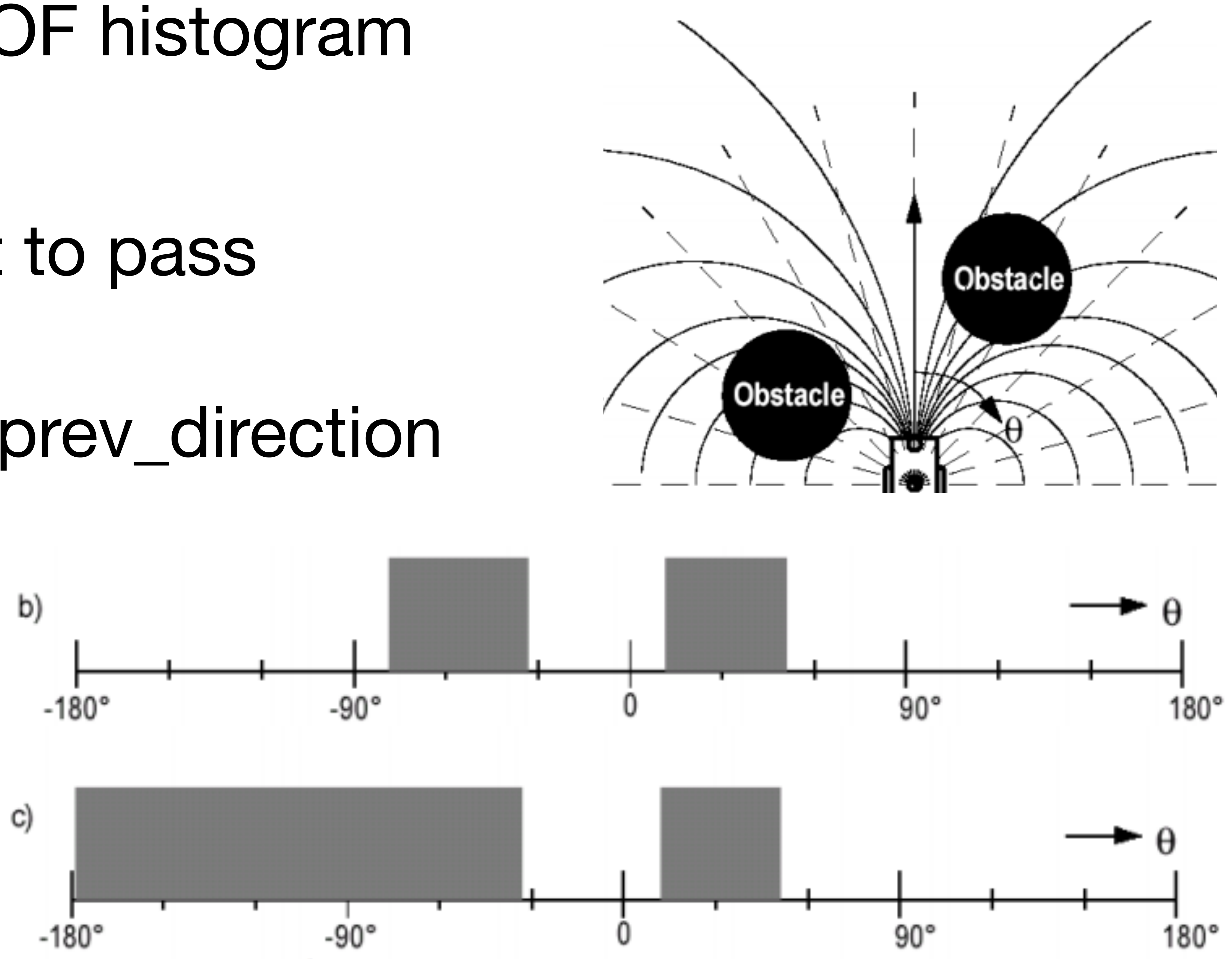




http://www.personal.umich.edu/~johannb/Papers/paper16.pdf

Vector Field Histograms

- VFH creates a local map of the environment around the robot populated by “relatively” recent sensor readings
- Build a local 3D grid map reduce to a 1-DOF histogram
- Planning
  - Find all openings large enough for robot to pass
  - Choose the one with the lowest cost, G
  - G = a*goal_direction + b*orientation +c*prev_direction
  - VFH+: incorporate kinematics
- Limitations
  - Does not avoid local minima
  - Not guaranteed to reach goal



http://www.personal.umich.edu/~johannb/Papers/paper16.pdf

Dynamic Window Approach

- Search in the velocity space (robot moves in circular arcs)
  - Takes into account robot acceleration and update rates
- A dynamic window, V_d, is the set of all tuples (v_d, ω_d) that can be reached
- Admissable velocities, V_a, include those where the robot can stop before collision
- The search space is then $V_r = V_s \cap V_a \cap V_d$
- Cost function: $G(v,\omega) = \sigma(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot velocity(v,\omega))$

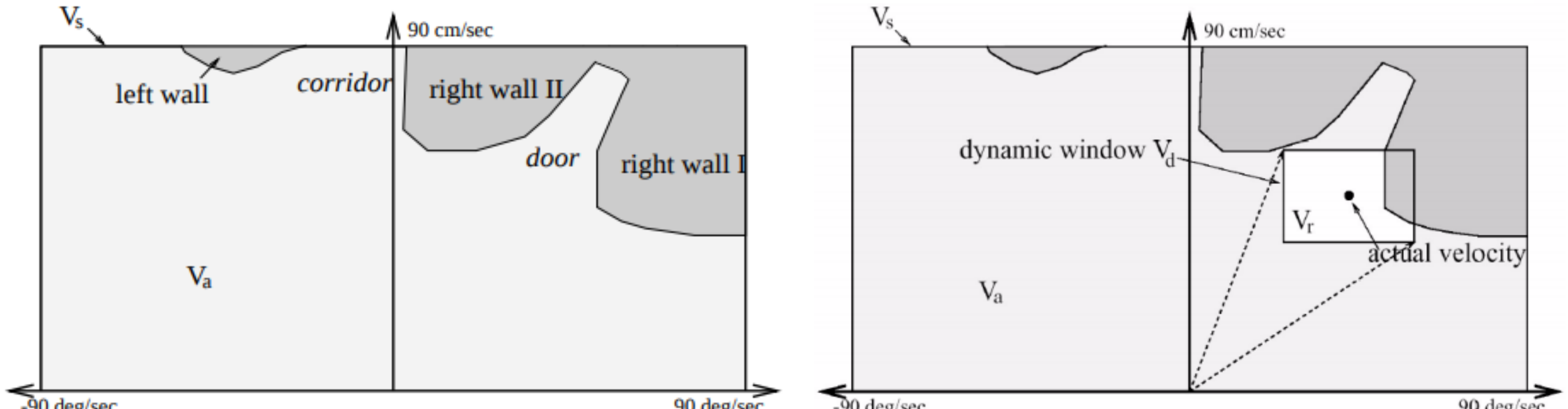


Figure 4. Velocity space

http://www4.cs.umanitoba.ca/~jacky/Teaching/Courses/74.795-LocalVision/ReadingList/fox97dynamic.pdf

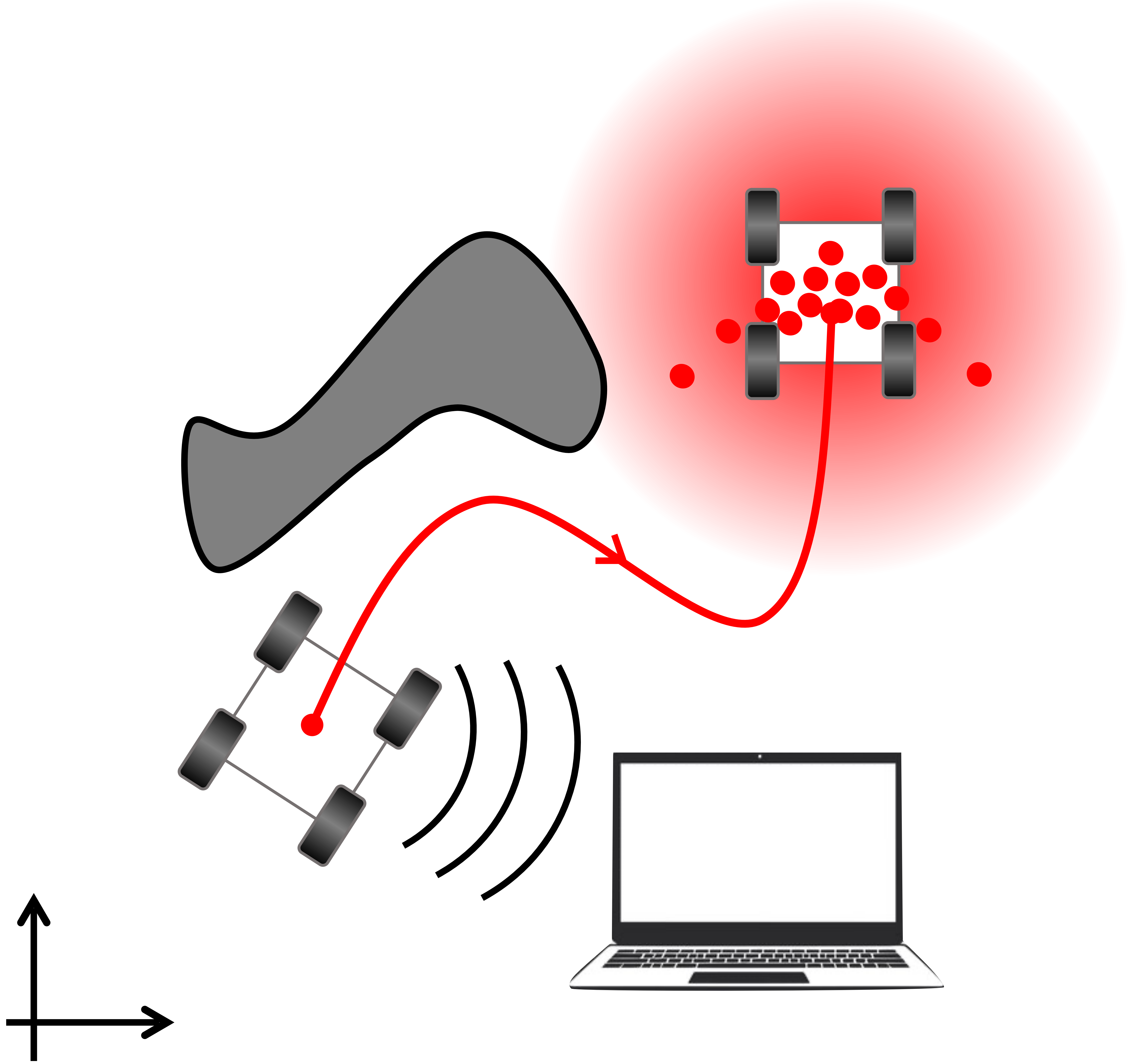
Local Planners

- Bug algorithms
  - Inefficient but can be exhaustive
- Vector Field Histograms
  - Takes into account probabilistic sensor measurements
- Vector Field Histograms+
  - Takes into account probabilistic sensor measurements and robot kinematics
- Dynamic window approach
  - Takes into account robot dynamics

Global localization

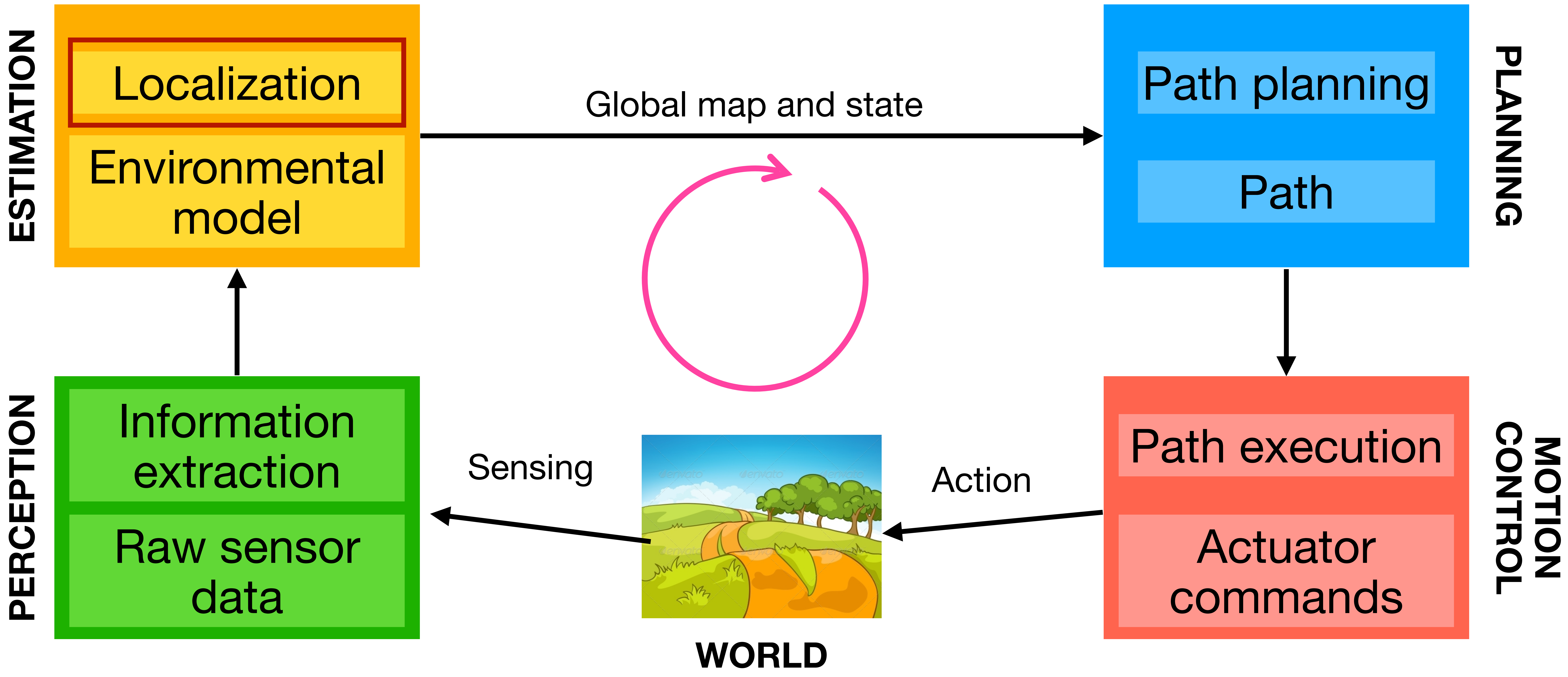
Next module on navigation

- Local planners
- Global localization and planning
  - Map representations
    - Continuous
    - Discrete
    - Topological
  - Maps as graphs
    - Graph search algorithms
      - Breadth first search
      - Depth first search
      - Dijkstras
      - A*



Navigation

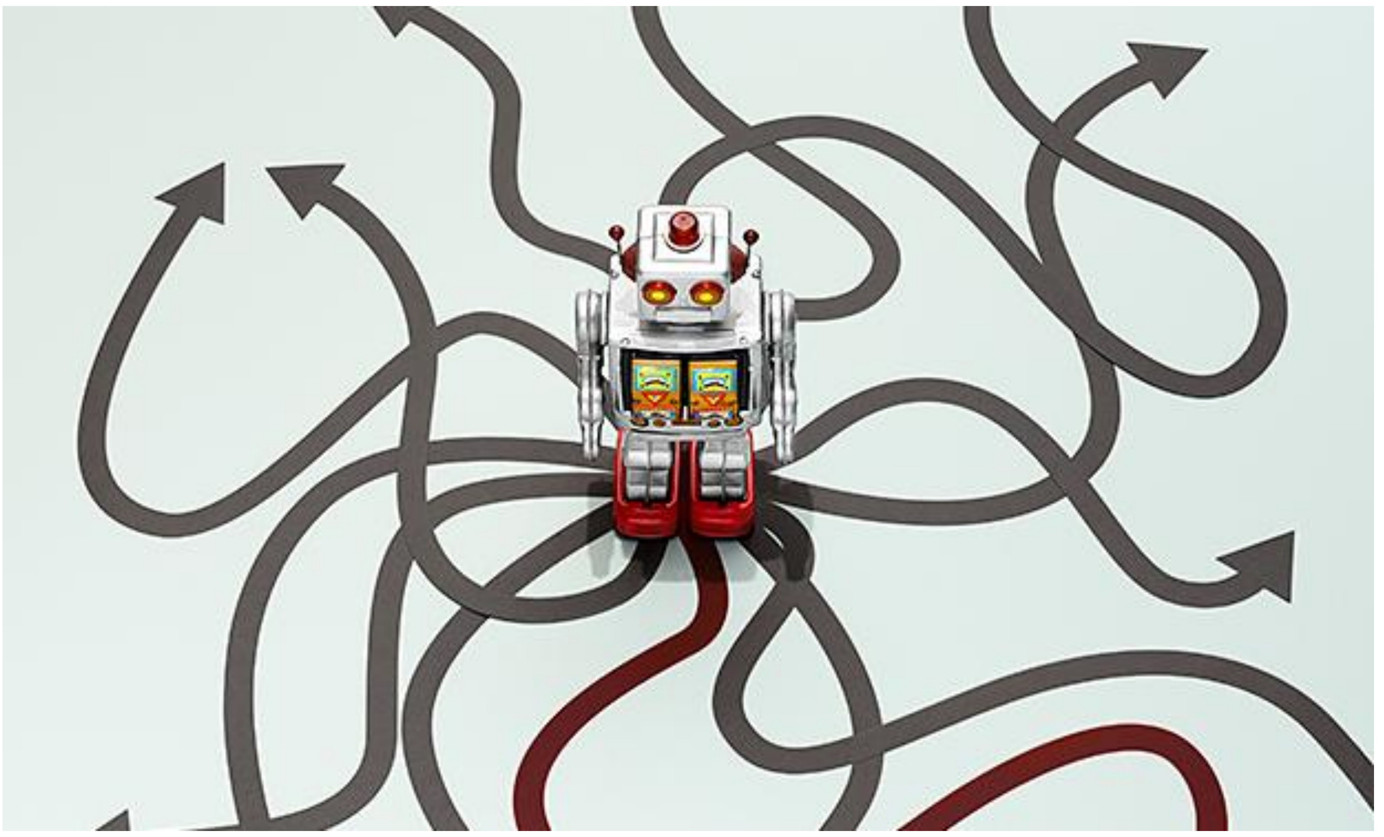
- **Break the problem down:** localization, map building, path planning



Localization Problem

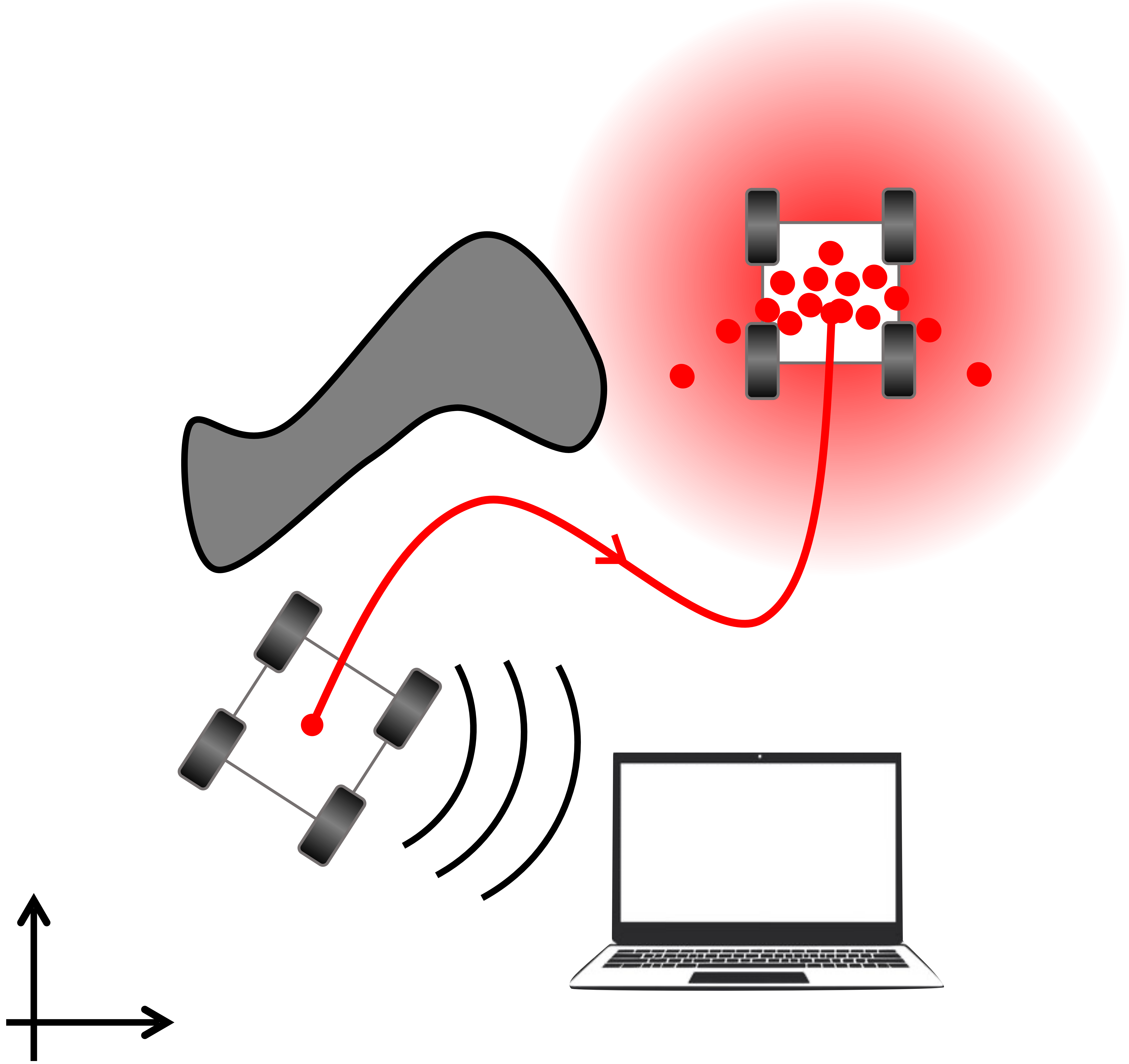
Position Tracking	Global Localization
• Initial **robot pose is known**	• Initial **robot pose is unknown**
• Either deterministically (odometry) or through Bayesian statistic (motion and sensor models)	• Need to estimate position from scratch
• It is a "**local**" problem, as the uncertainty is local (often small) and confined to a region near the robot's true pose	• A more difficult "**global**" problem, where you cannot assume boundedness in pose error

kidnapped robot problem



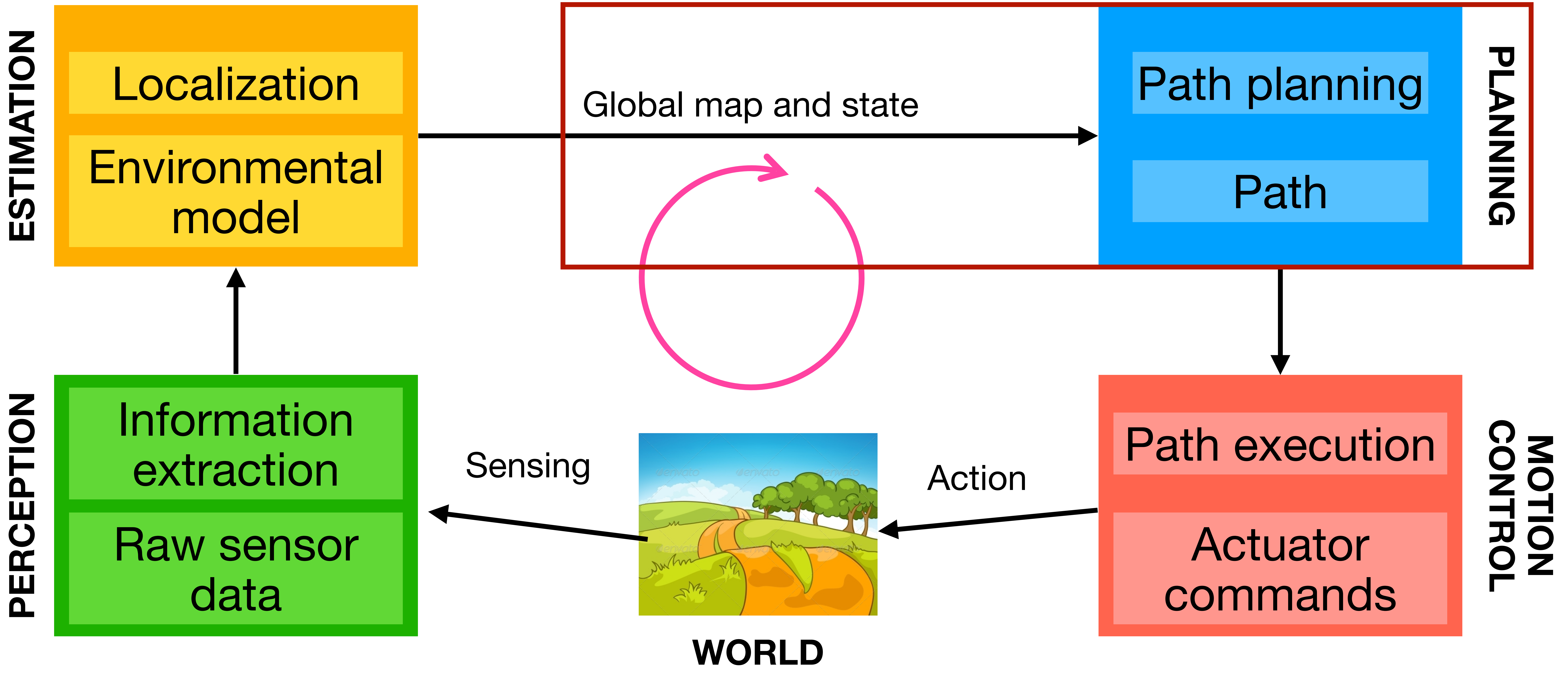
Next module on navigation

- Local planners
- Global localization and planning
  - Map representations
    - Continuous
    - Discrete
    - Topological
  - Maps as graphs
    - Graph search algorithms
      - Breadth first search
      - Depth first search
      - Dijkstras
      - A*



Navigation

- **Break the problem down:** localization, map building, path planning



Map Representation

a) Building Plan

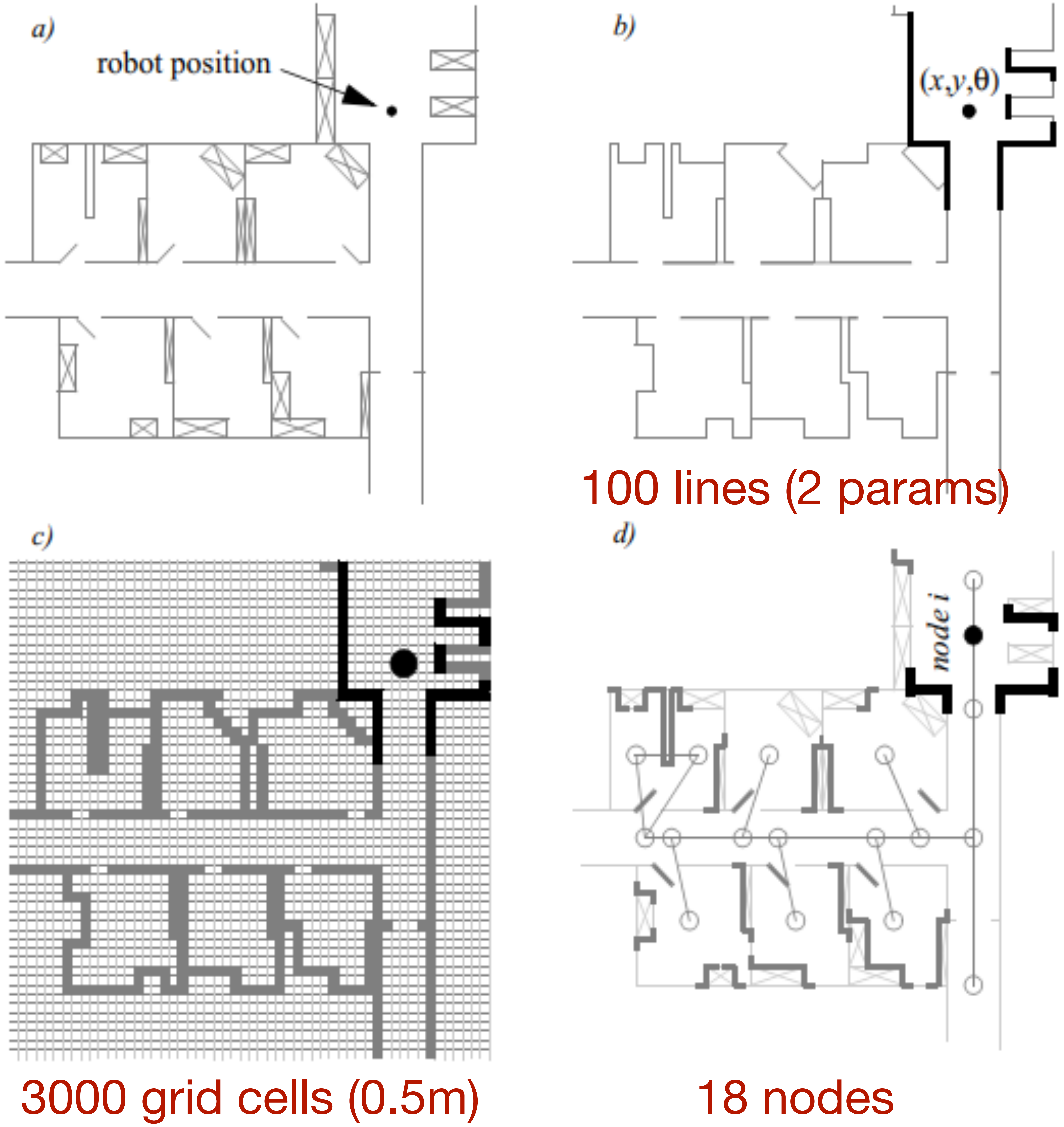
b) Line-based map

c) Occupancy grid-based map

d) Topological map

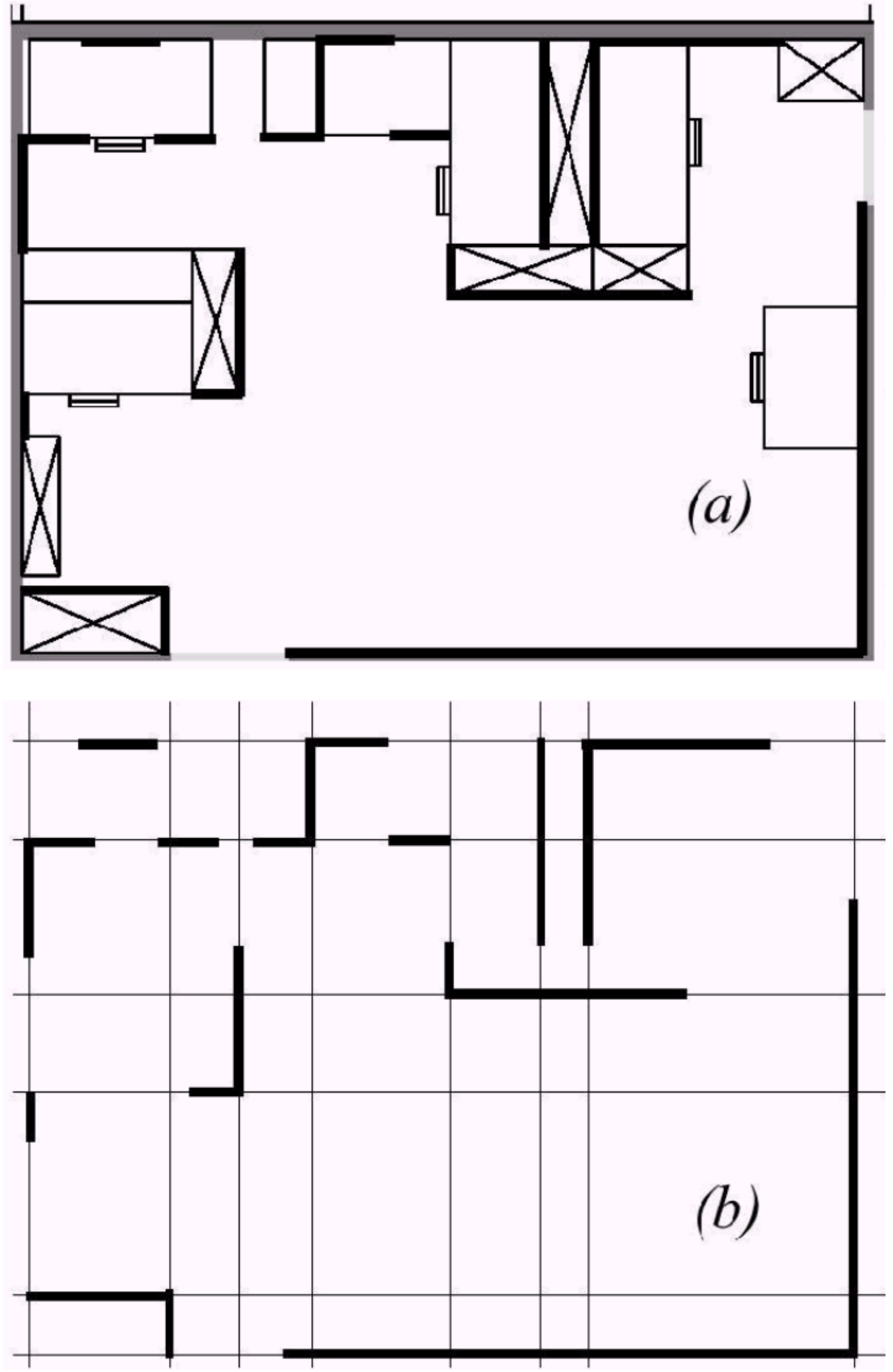
Important Properties

- Memory allocation
- Computation



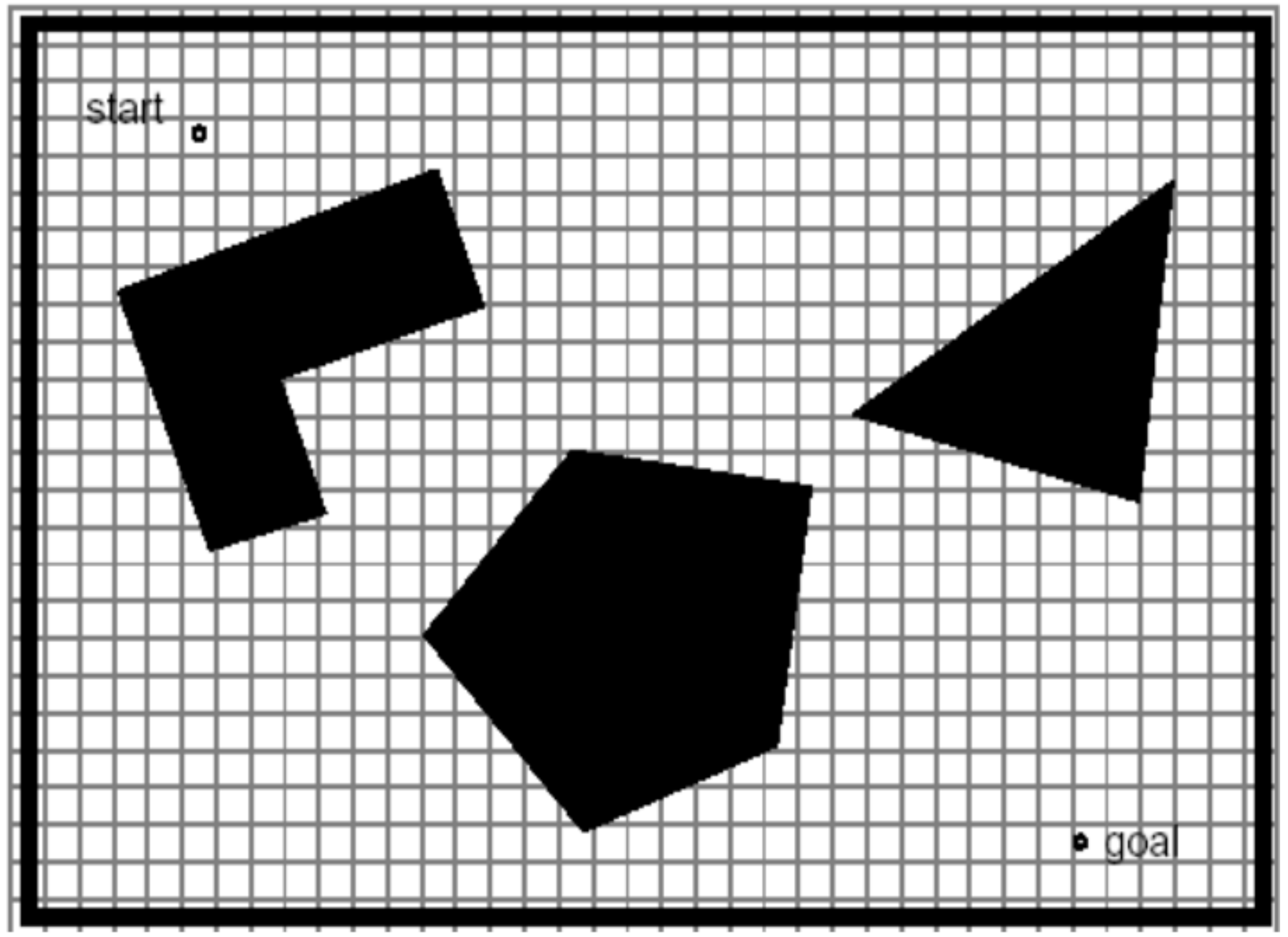
Continuous Representations

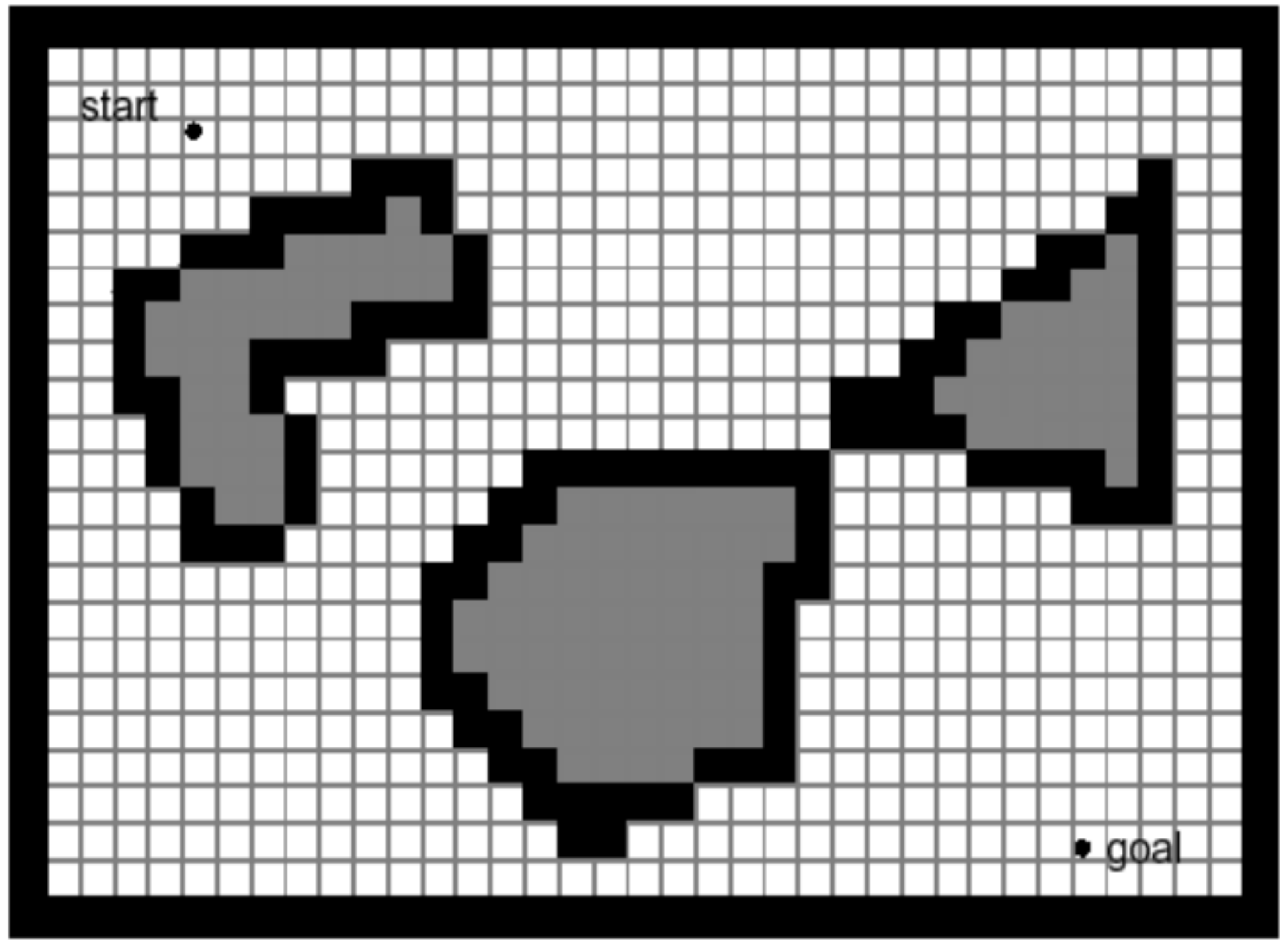
- Exact decomposition of the environment
- Used mainly in 2D representations
- Closed-world assumption
- Storage proportional to object density
- Example: Continuous line representations
  - Using range finders, we can extract lines/ line segments in the environment

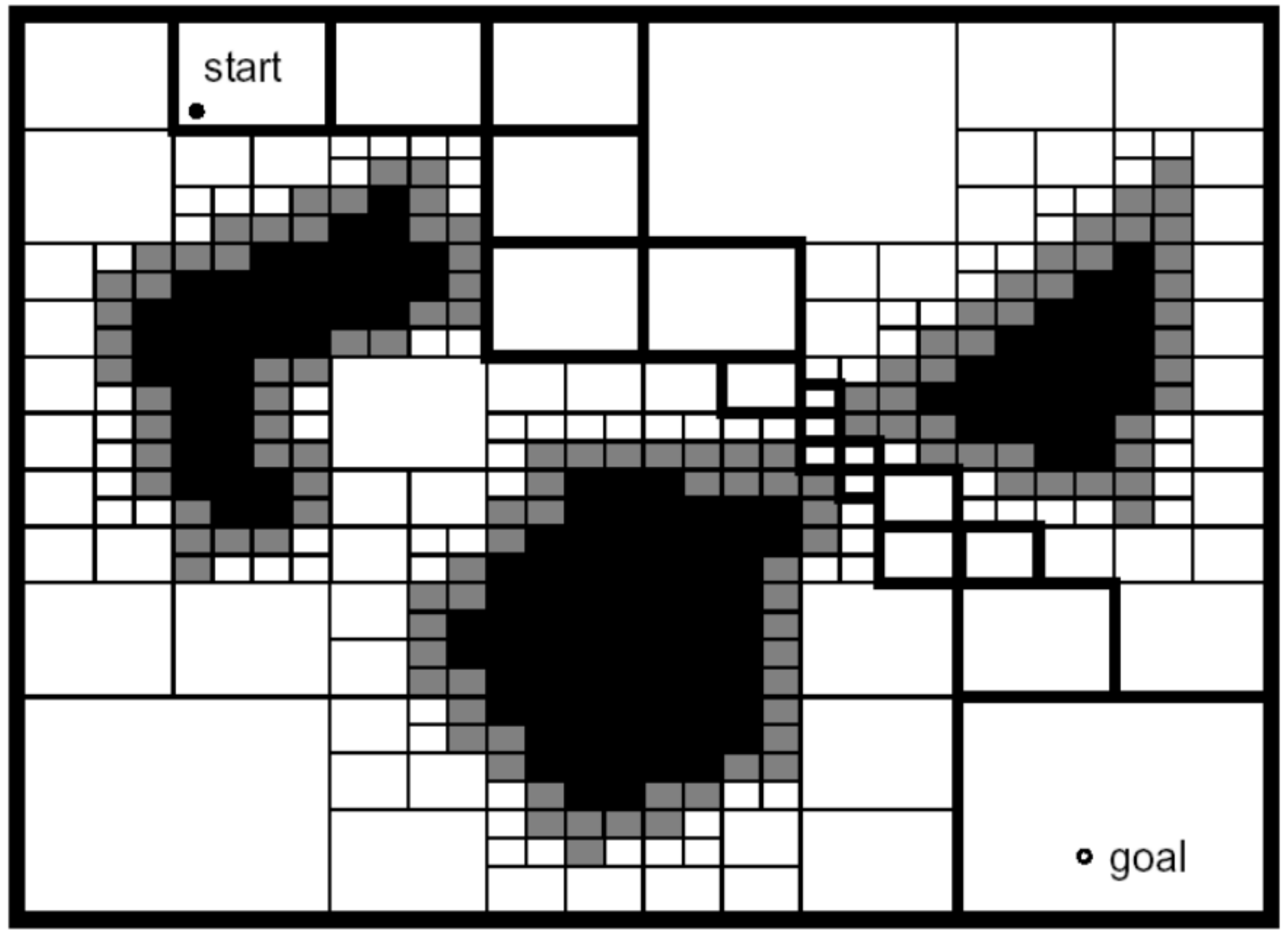


Cell Decomposition

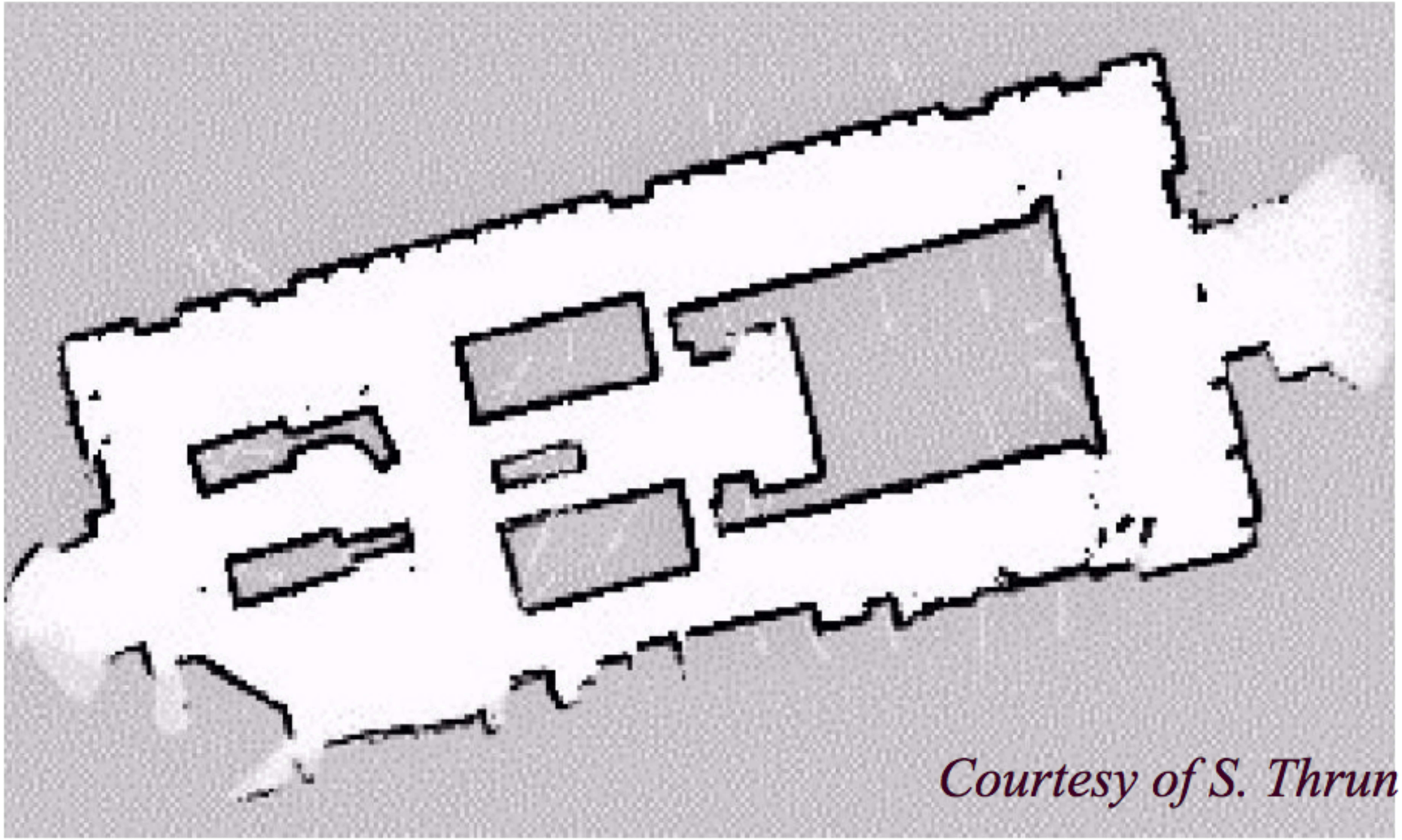
- Fixed: Tesselate the world at a fixed resolution
- Approximate features given the resolution
- Most commonly used: occupancy grid
- Adaptive: Tesselate the world at varying resolutions, finer near objects





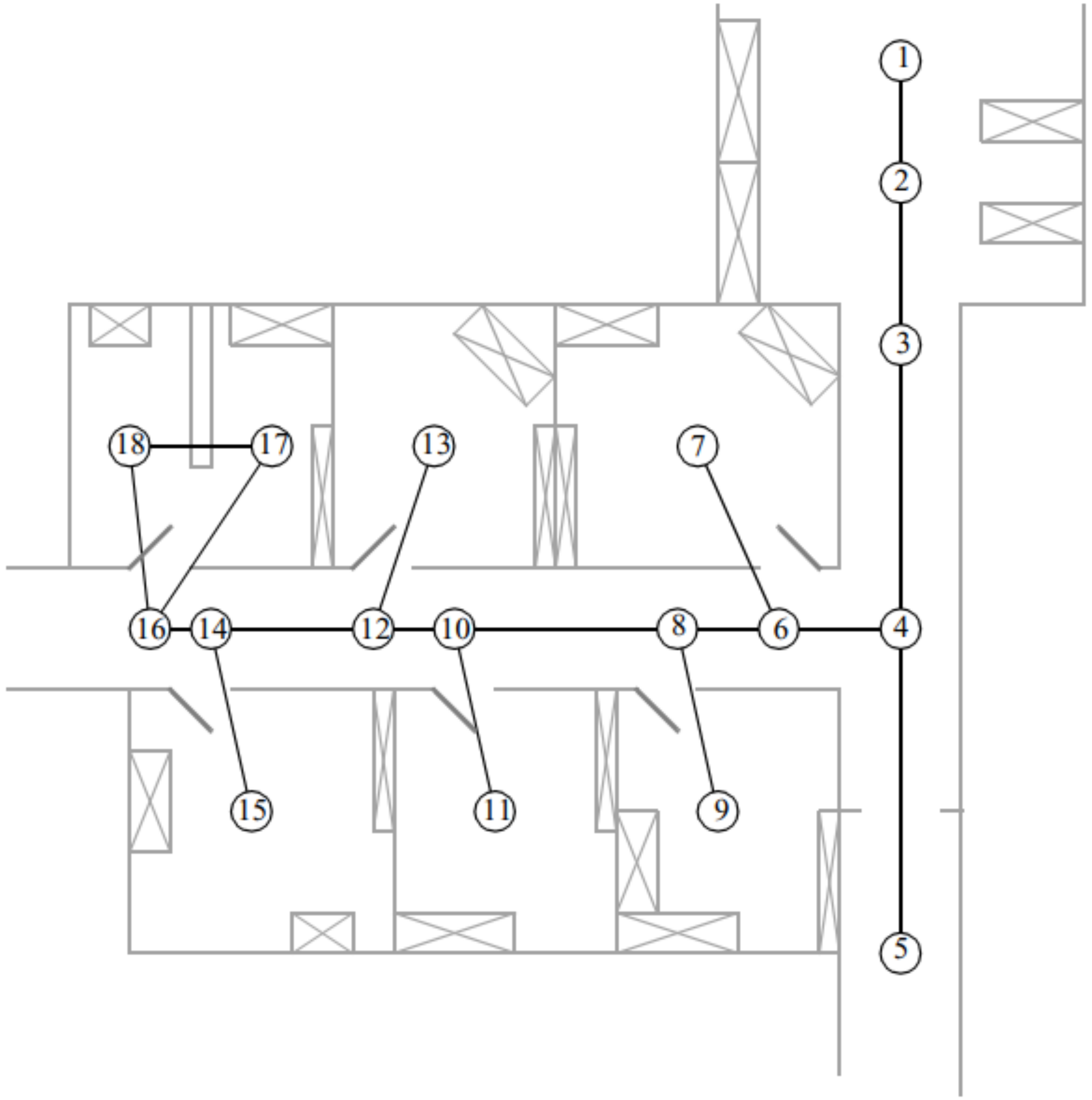


Fixed Decomposition



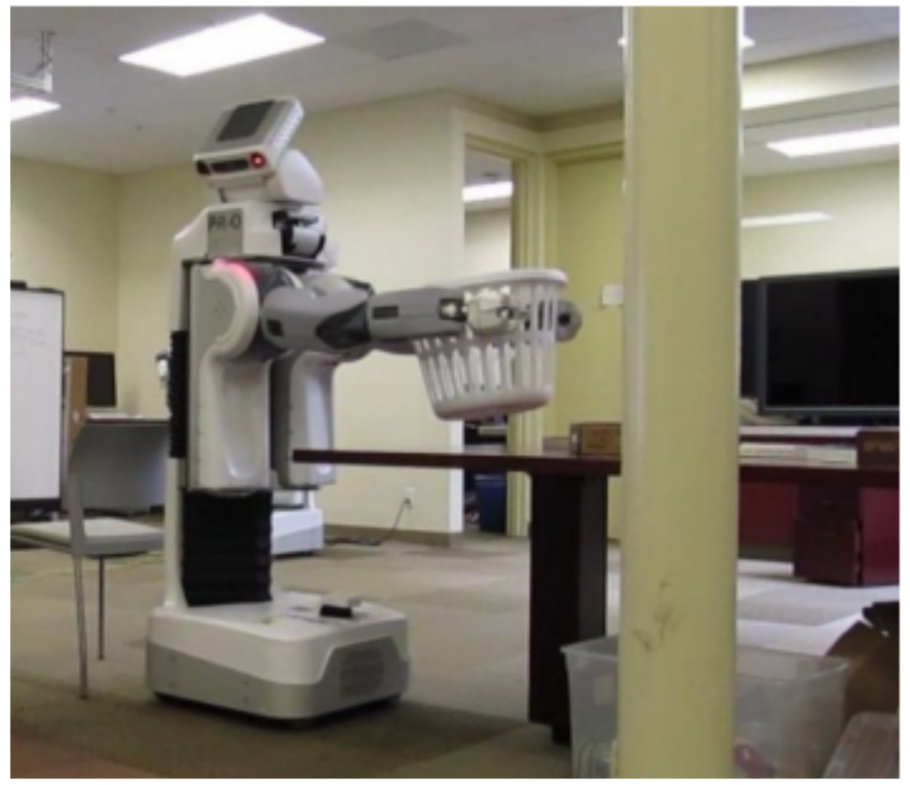
Topological decomposition

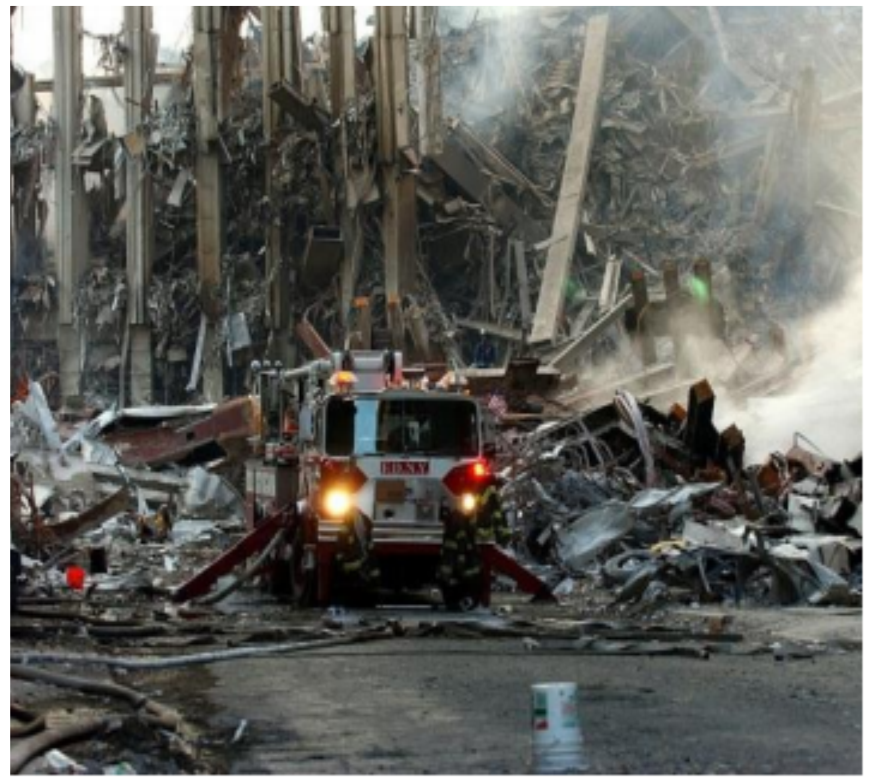
- A topological representation is a graph that specifies nodes and edges
  - Nodes denotes areas in the environment
  - Edges describe environment connectivity
- Robots can…
  - …detect their current position in terms of the nodes of the topological graph
  - …travel between nodes using robot motion



Topological decomposition

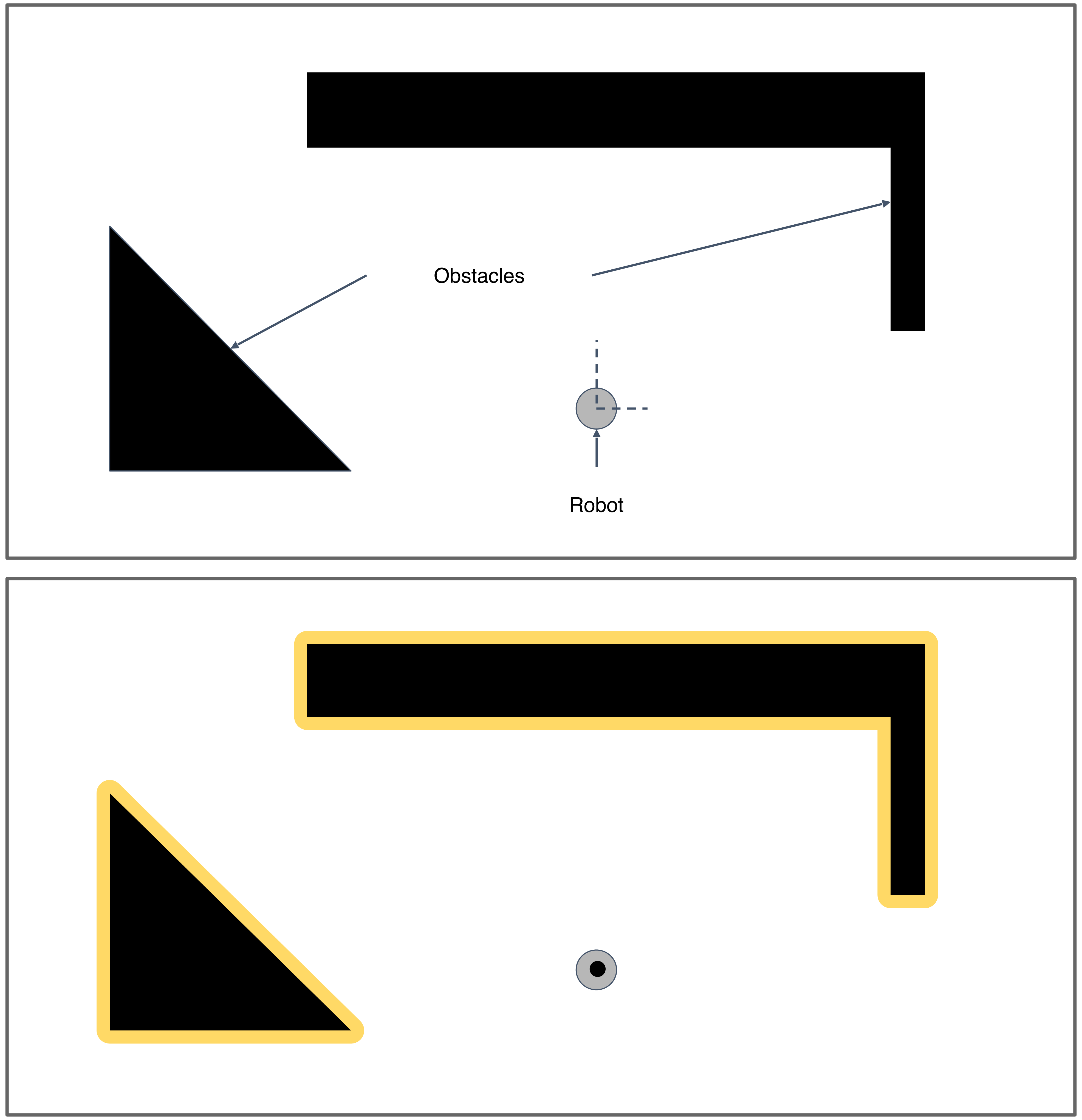
- A topological representation is a graph that specifies nodes and edges
  - Nodes denotes areas in the environment
  - Edges describe environment connectivity
- Robots can…
  - …detect their current position in terms of the nodes of the topological graph
  - …travel between nodes using robot motion
- Typical for 3D maps





How to represent the robot pose?

- Physical robots take up space
- Expand obstacles
- Represent maps in configuration space instead of Euclidean space



Robot can be treated as a point object

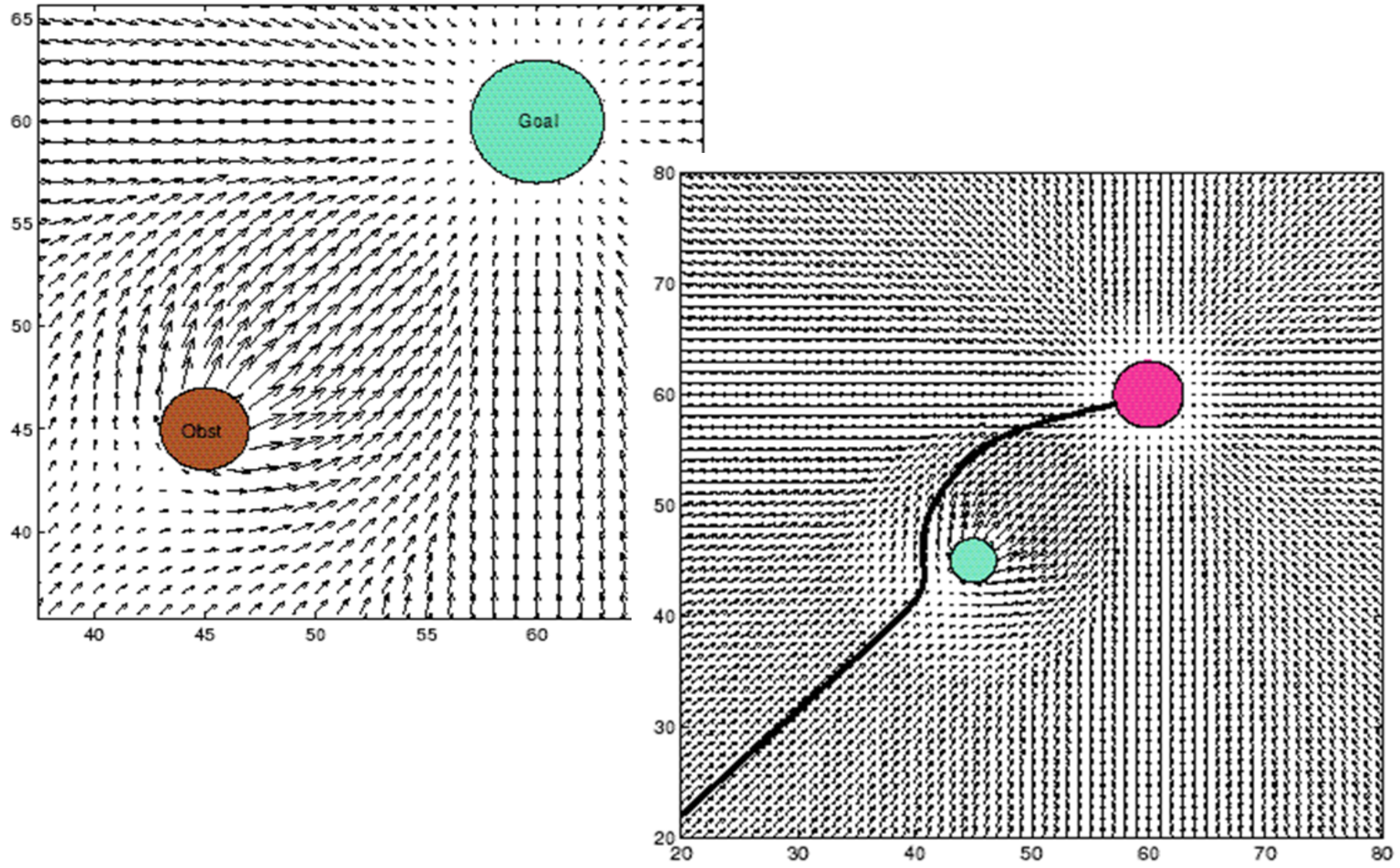
Map Representation Considerations

- The precision of the map must appropriately match the precision with which the robot needs to achieve its goals
- The precision of the map and the type of features represented must match the precision and data types returned by the robot’s sensors
- The complexity of the map representation has direct impact on the computational complexity of reasoning about mapping, localization, and navigation

Constructing Graphs

Modeling path planning as a graph search problem

Real World → Configuration Space → Map Representation → Graph Construction → Graph Search

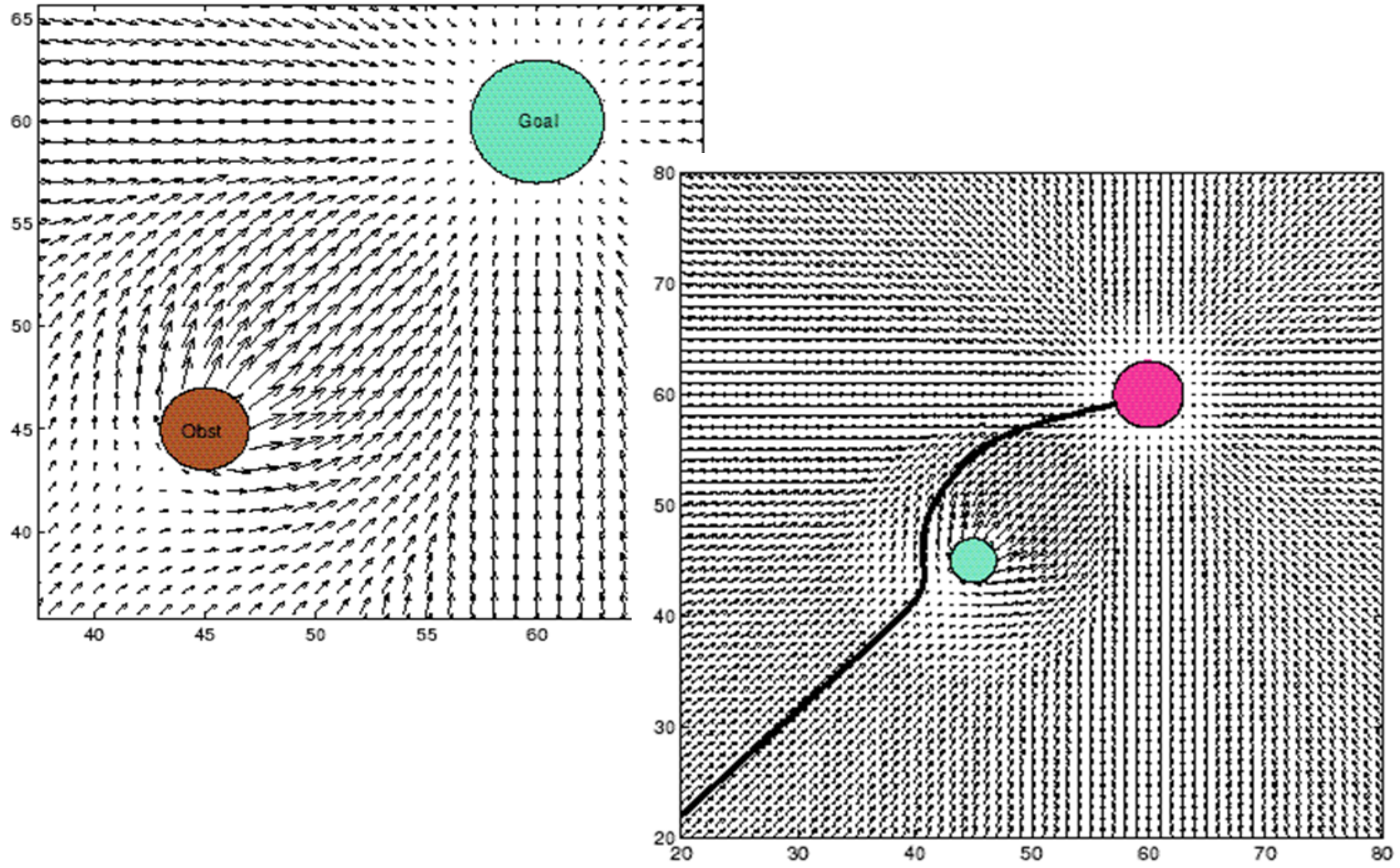


Common alternatives

- Optimal control
- Potential fields

Modeling path planning as a graph search problem

Real World → Configuration Space → Map Representation → **Graph Construction** → Graph Search

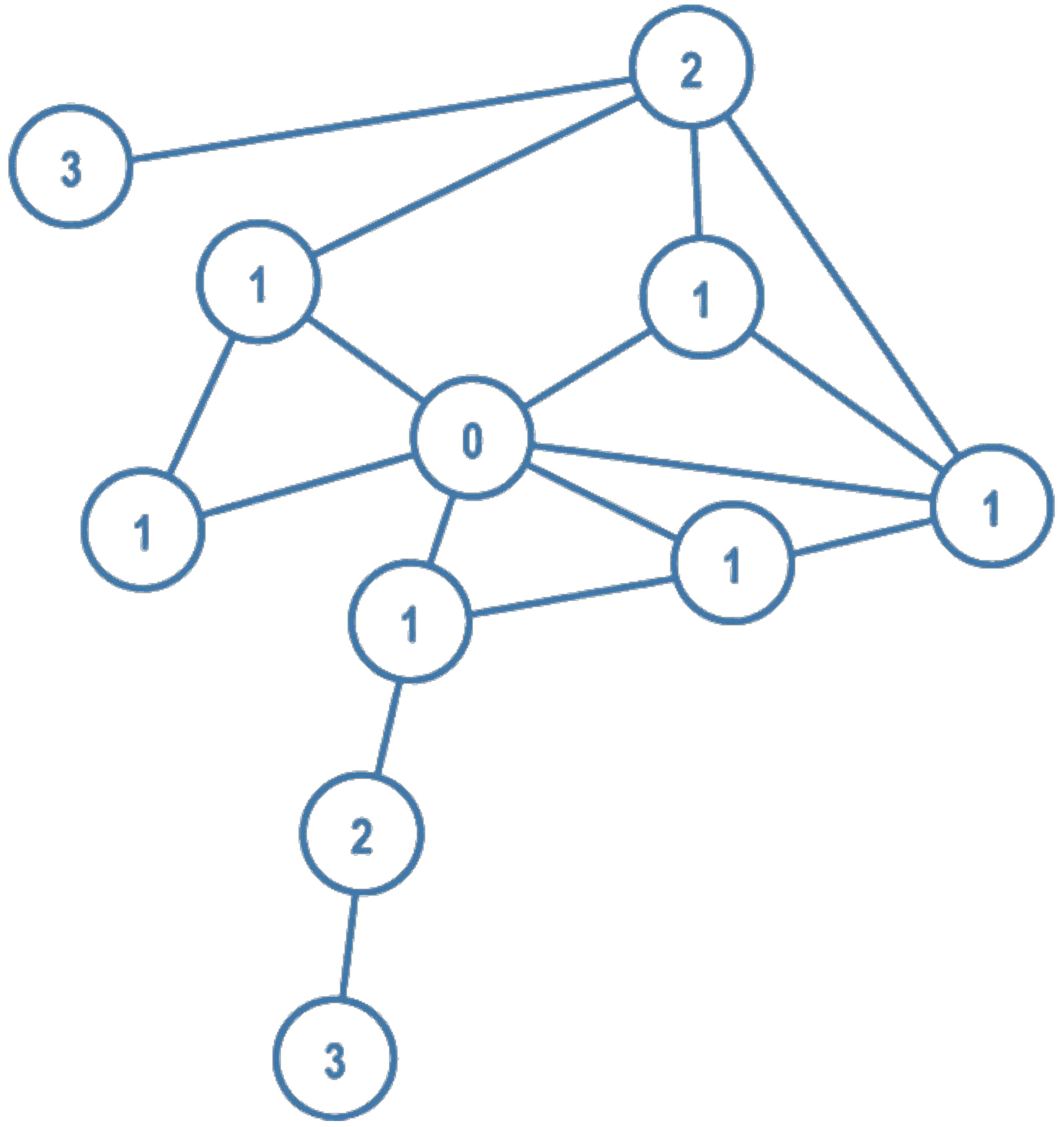


- Topological Graphs
- Cell decomposition
- Visibility Graphs
- RRT
- PRM

Graph Construction

- Transform continuous/ discrete/ topological maps to a discrete graph
- Why?
  - Model the path planning problem as a search problem
  - Graph theory has lots of tools
  - Real-time capable algorithms
  - Can accommodate for evolving maps

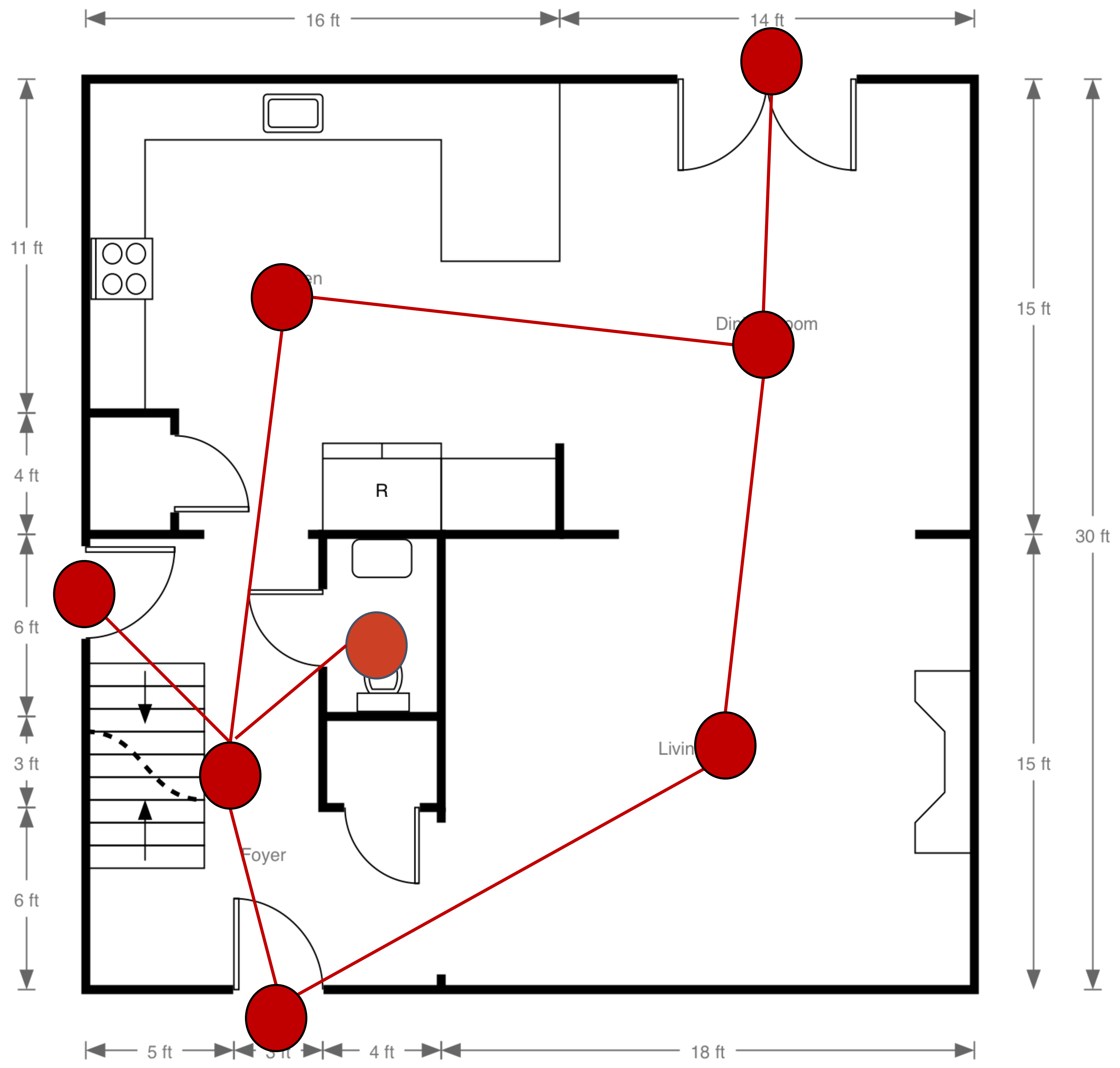
1. Divide space into simple, connected regions, or "cells"
2. Determine adjacency of open cells
3. Construct a connectivity graph
4. Find cells with initial and goal configuration
5. Search for a path in the connectivity graph to join them
6. From the sequence of cells, compute a path within each cell
   - e.g. passing through the midpoints of cell boundaries or by sequence of wall following movements



Geometry-based planners

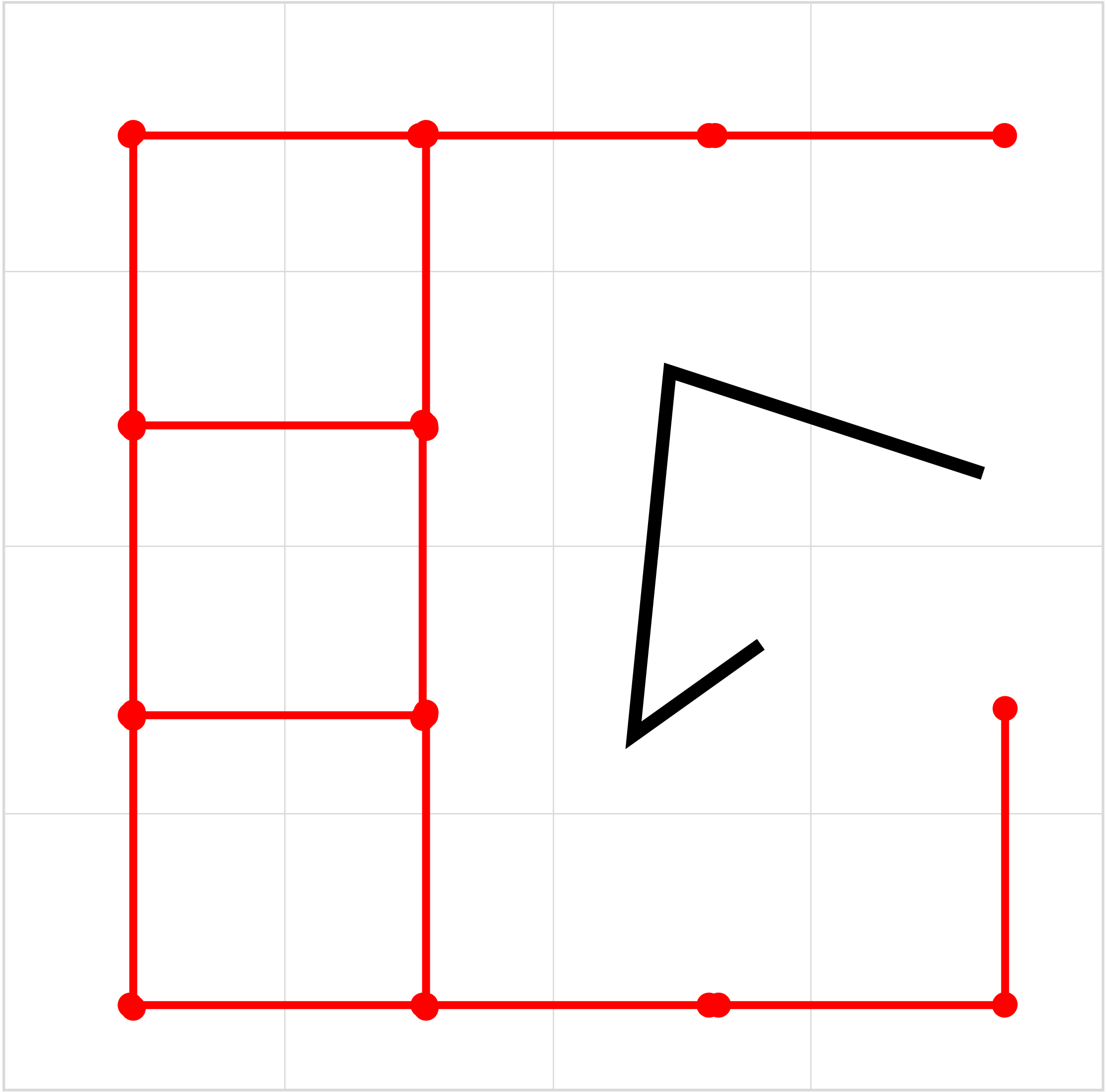
Topological Maps

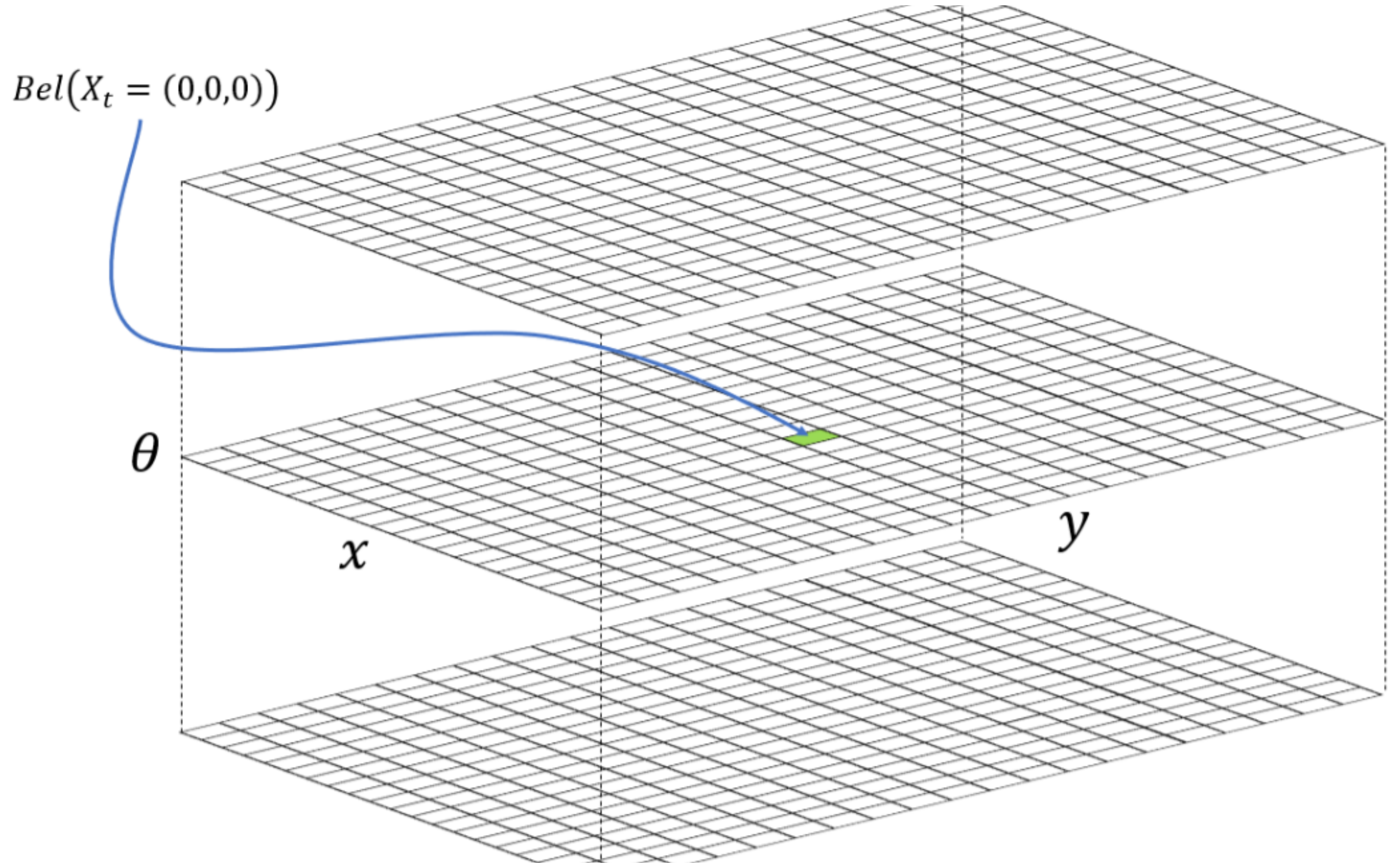
- Good abstract representation
- Tradeoff in # of nodes
    - Complexity vs. accuracy
    - Efficient in large, sparse environments
    - Loss in geometric precision
- Edges can carry weight
- Con: limited information



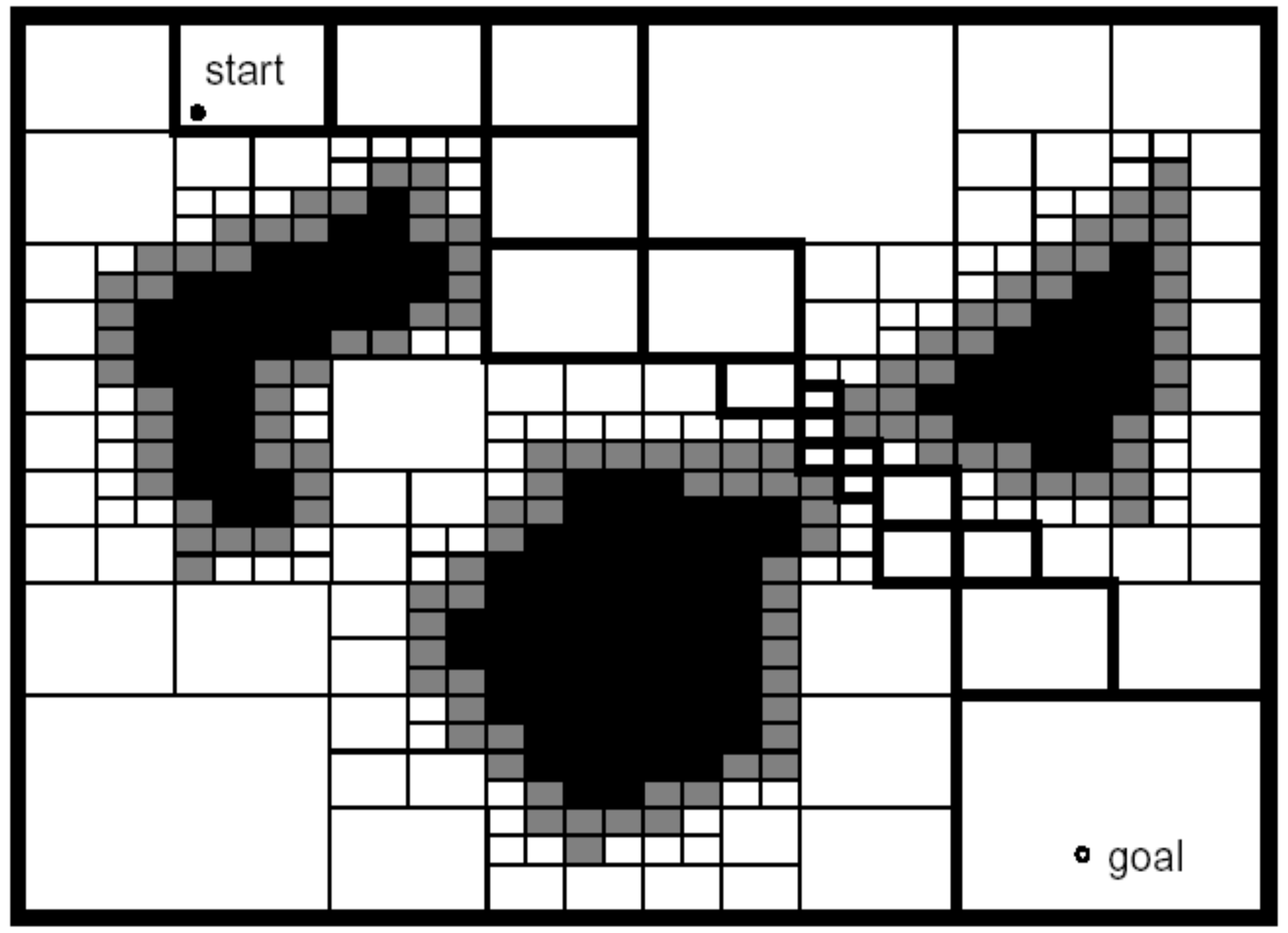
Fixed Cell Decomposition

(Lab 9-12)

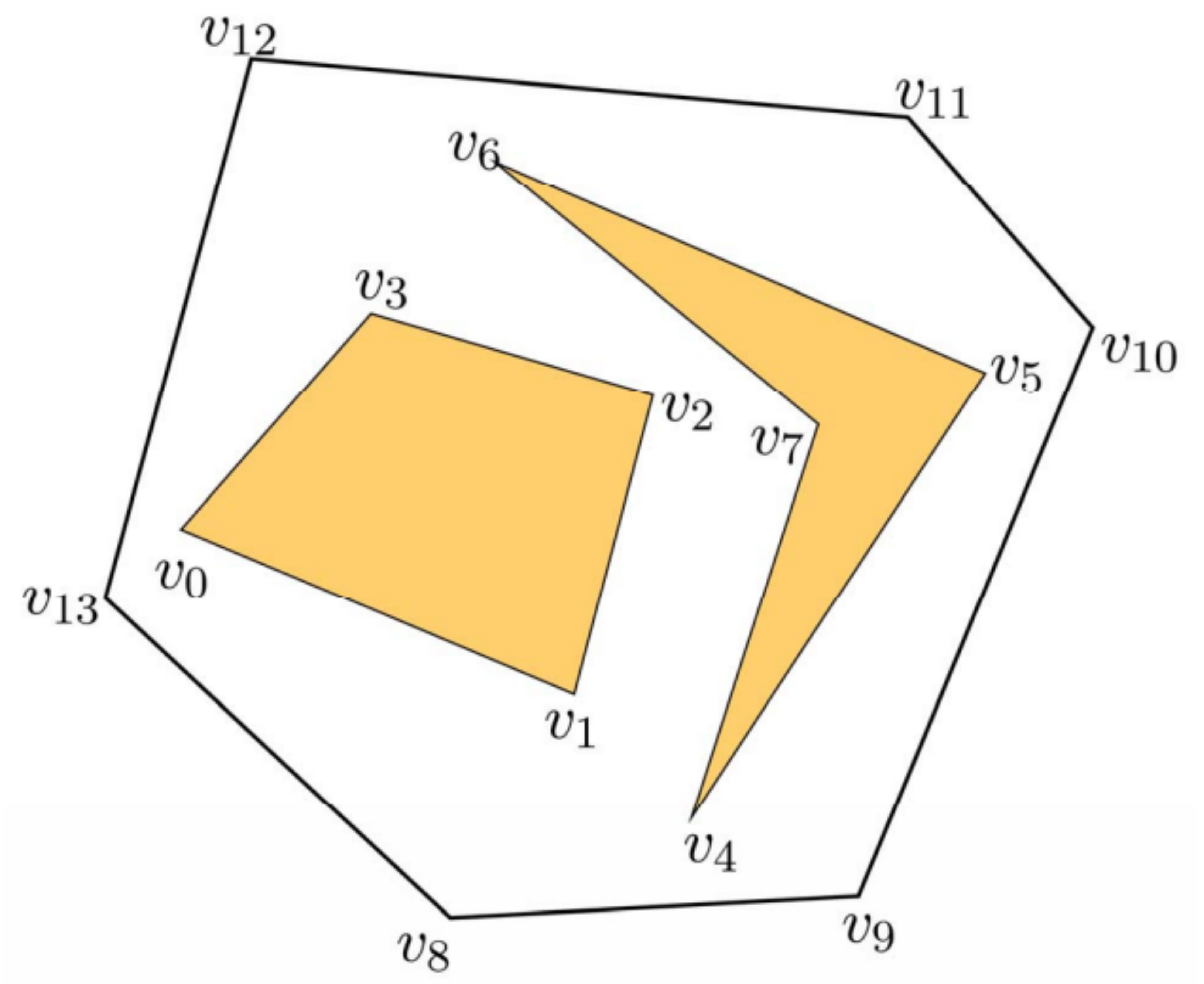




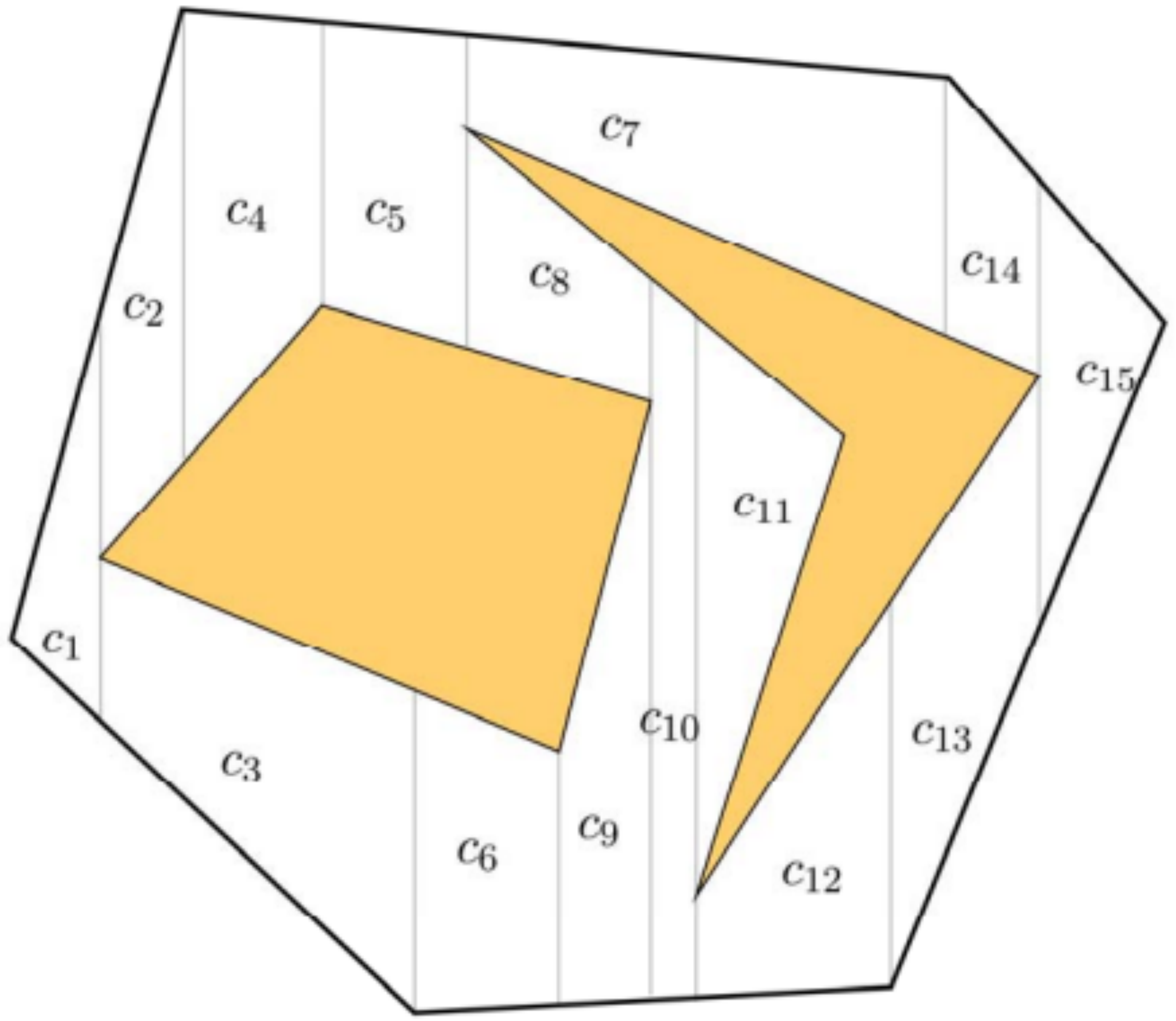
Adaptive Cell Decomposition

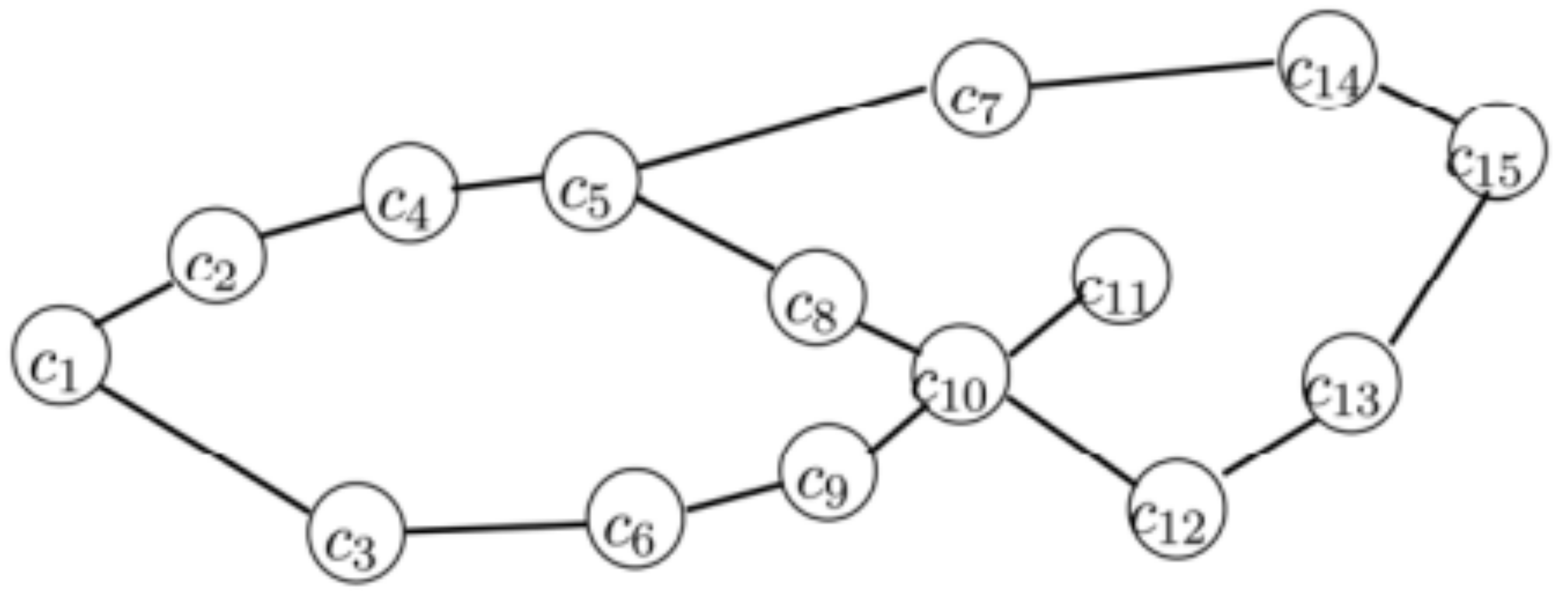


Trapezoidal Cell Decomposition



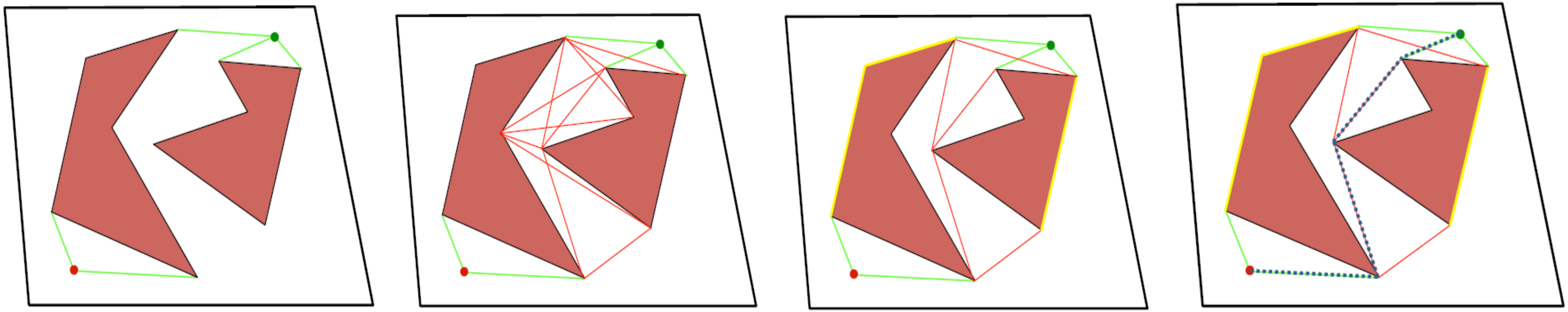
RI 16-735 Howie Choset





Visibility Graphs

- Connect initial and goal locations with all visible vertices
- Connect each obstacle vertex to every visible obstacle vertex
- Remove edges that intersect the interior of an obstacle
- Plan on the resulting graph



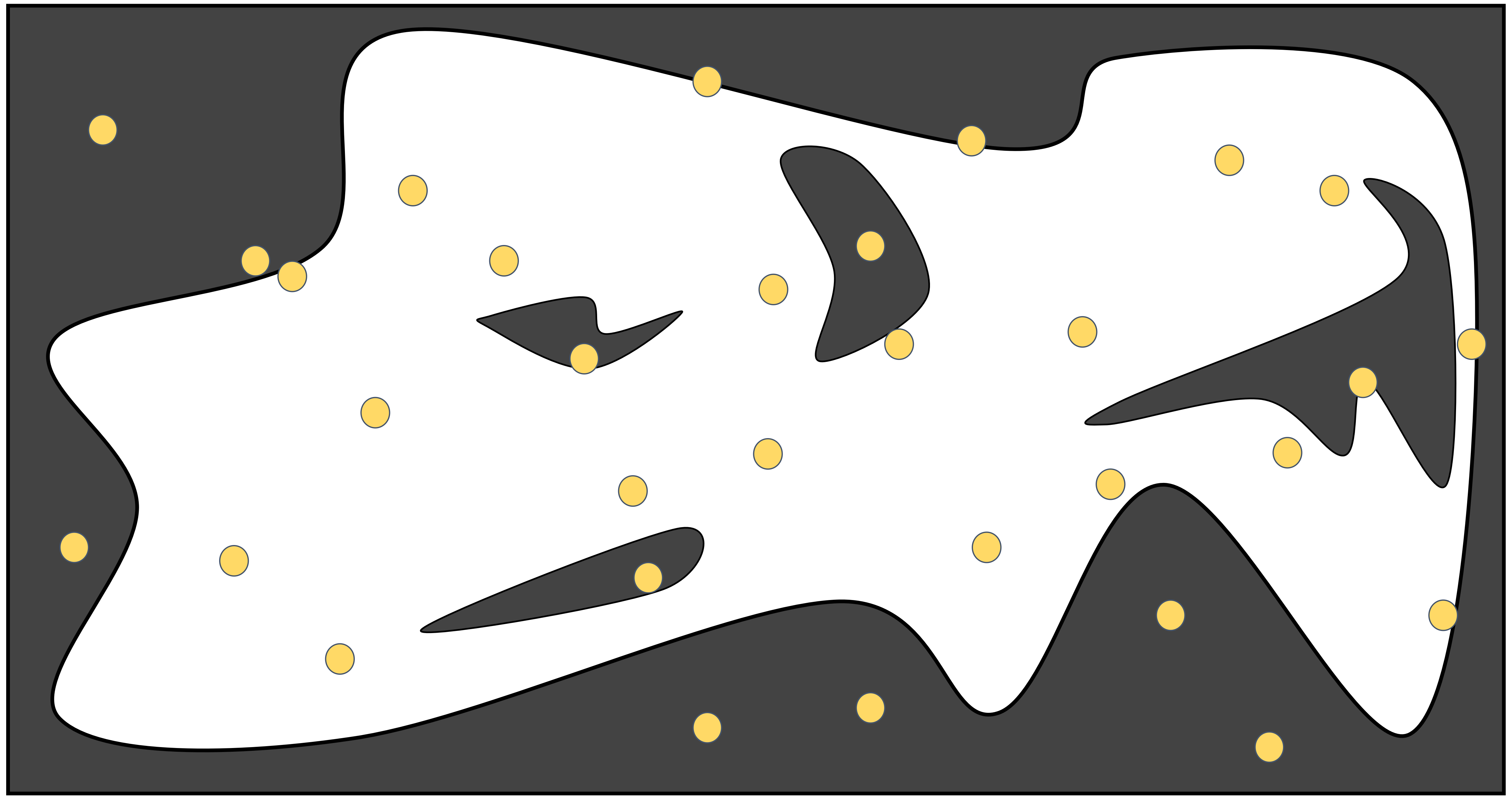
Ioannis Rekleitis, South Carolina

Sampling-based planners

- Rather than computing the C-Space explicitly, we sample it
- Often efficient in high dimensional spaces
- Compute if a robot configuration has collisions
  - Just requires forward kinematics
  - (Local path plans between configurations)
- Examples
  - Probabilistic Roadmaps (PRM)
  - Rapidly Exploring Random Trees (RRT)

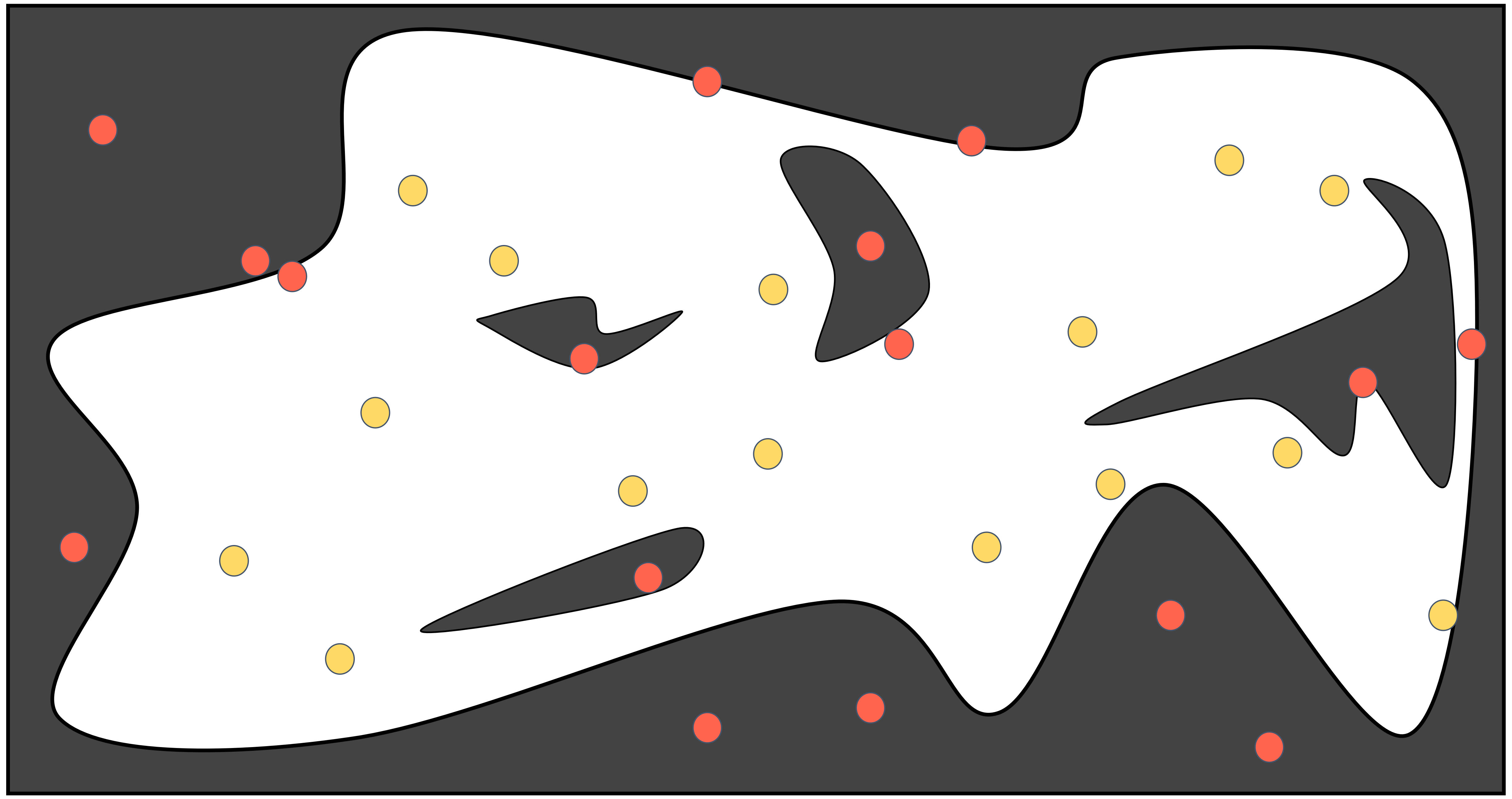
Probabilistic Roadmaps

- Configurations are sampled by picking coordinates at random



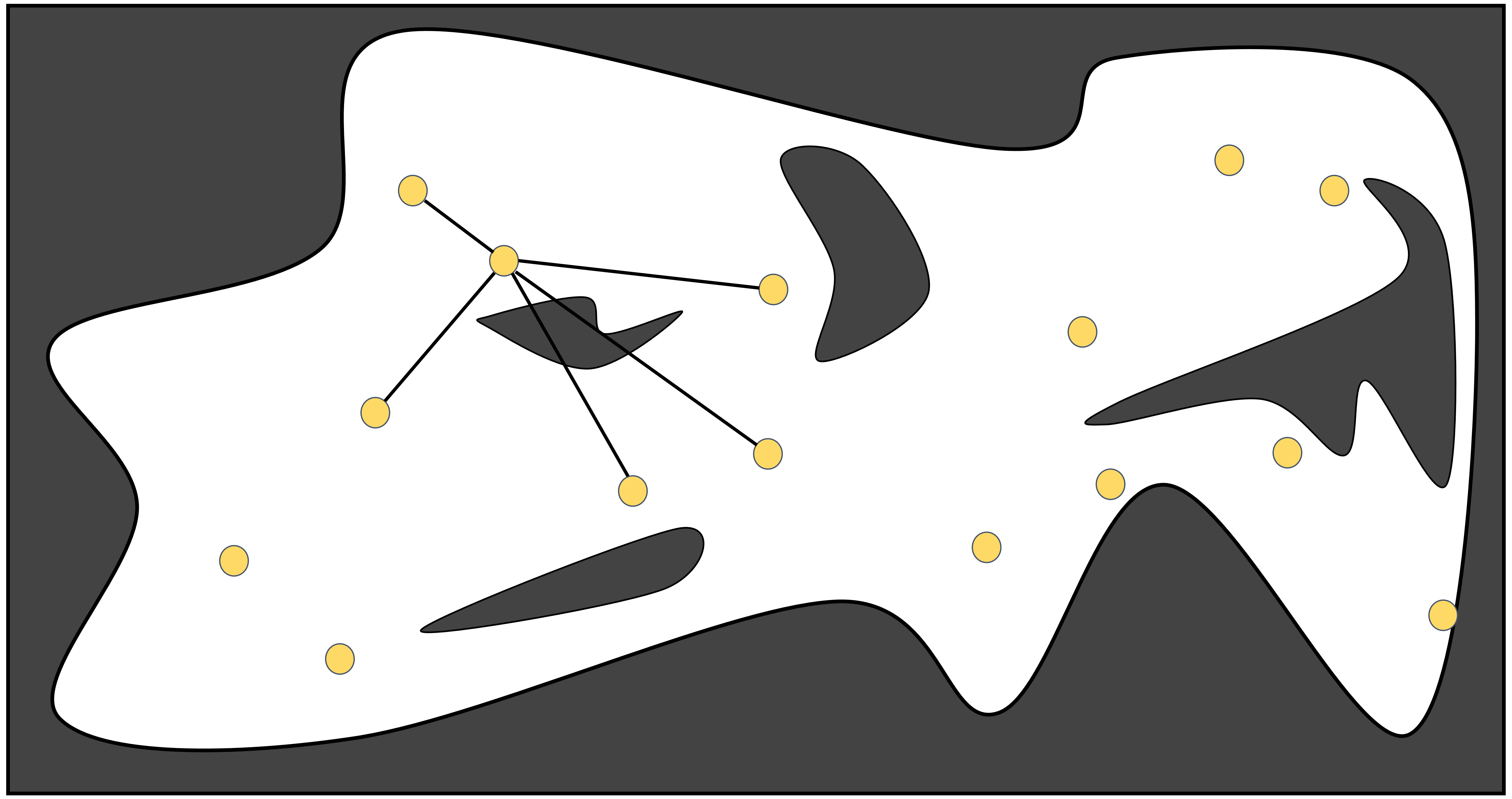
Probabilistic Roadmaps

- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision



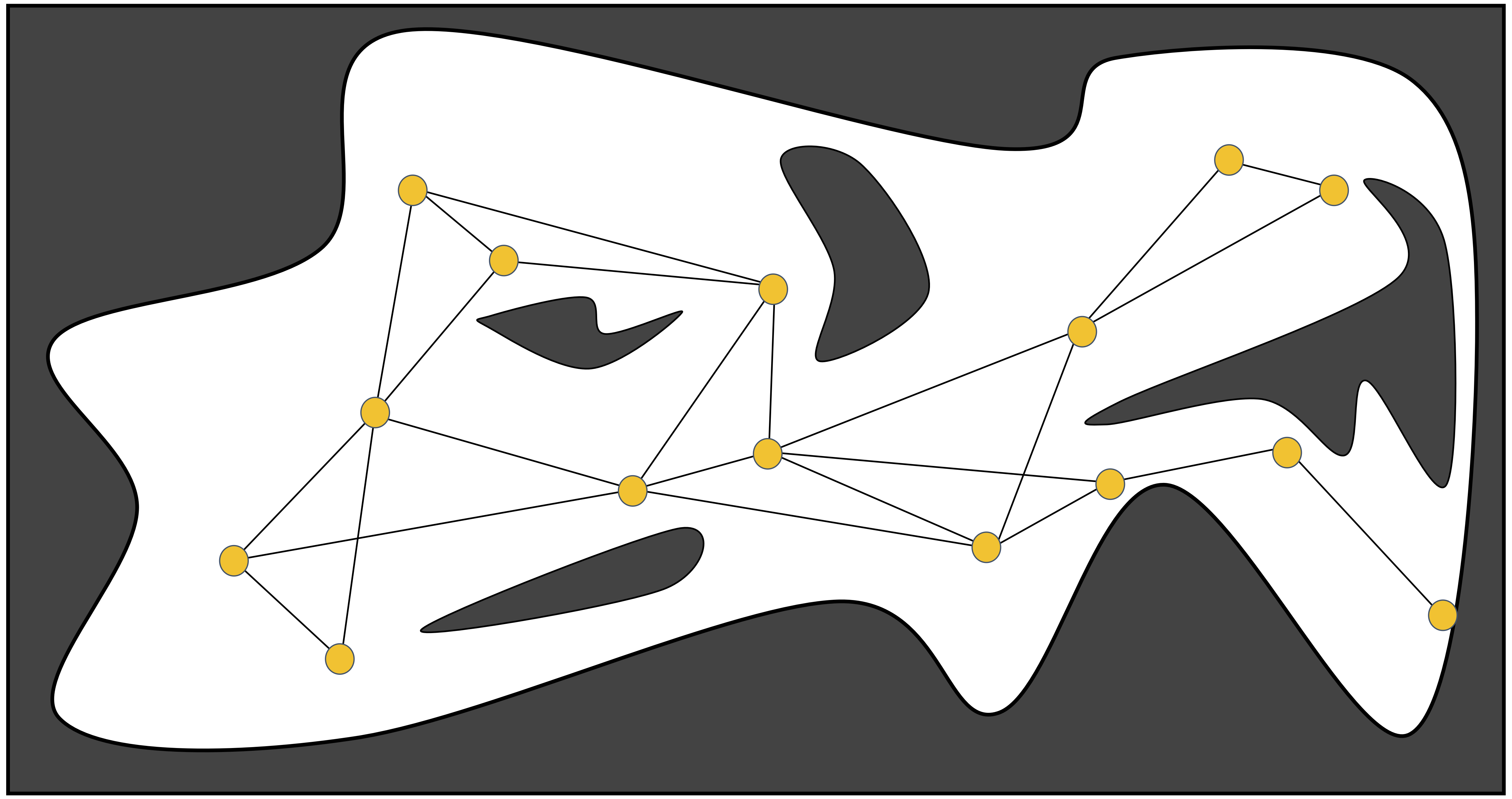
Probabilistic Roadmaps

- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision
- Each configuration is linked by straight paths to its nearest neighbors



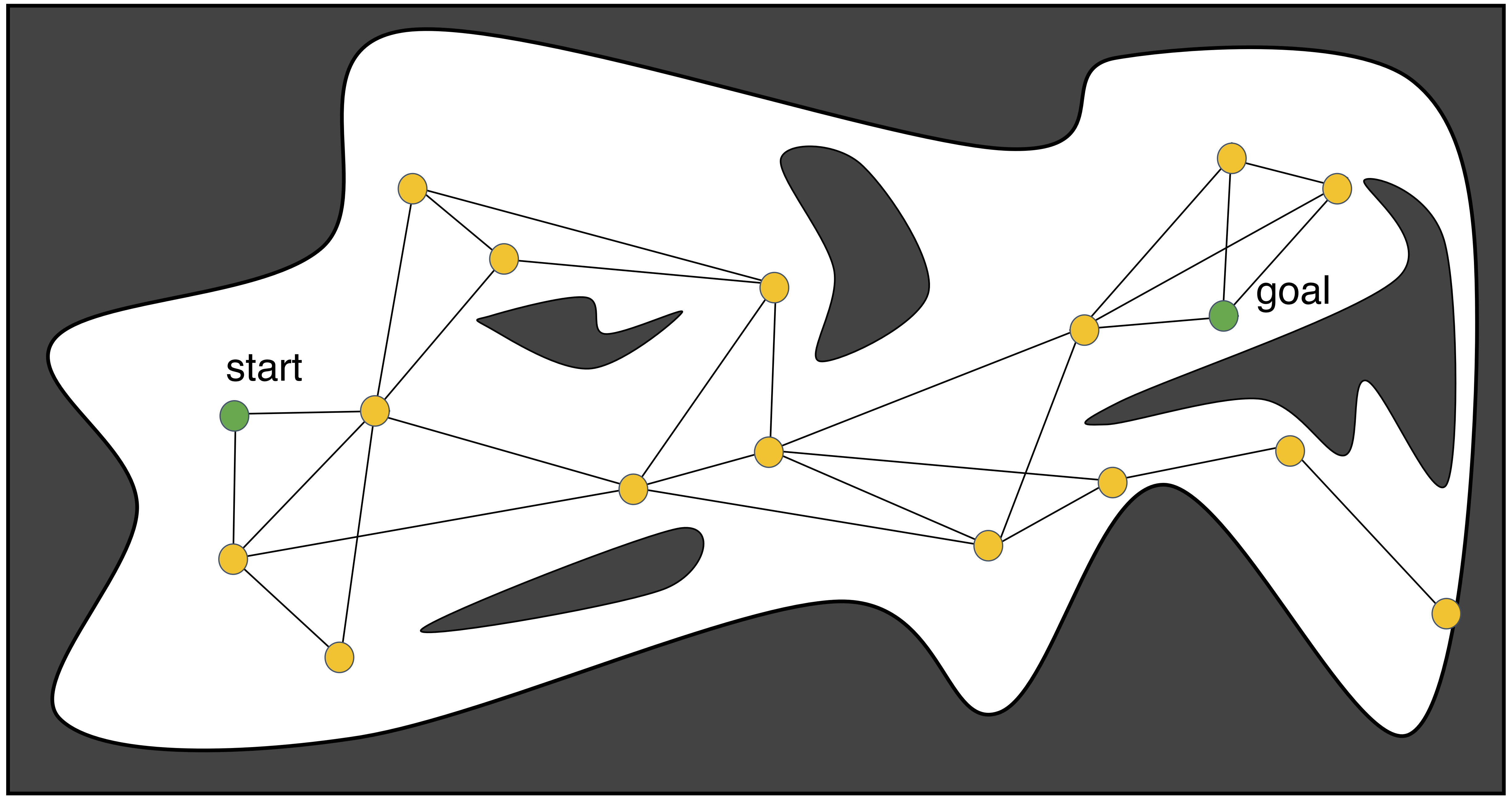
Probabilistic Roadmaps

- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision
- Each configuration is linked by straight paths to its nearest neighbors
- The collision-free links are retained as local paths to form the PRM



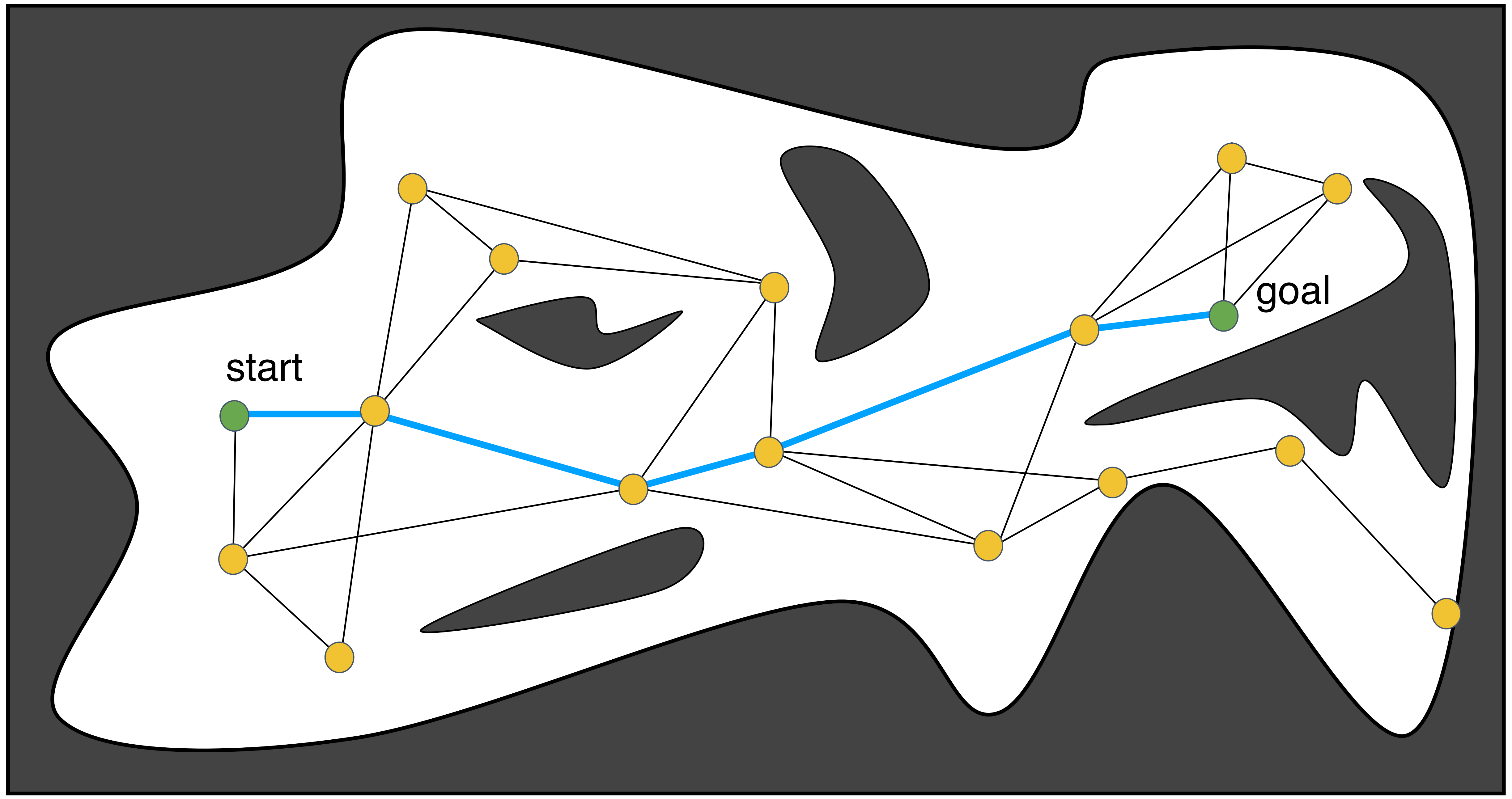
Probabilistic Roadmaps

- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision
- Each configuration is linked by straight paths to its nearest neighbors
- The collision-free links are retained as local paths to form the PRM
- The start and goal configurations are included as milestones



Probabilistic Roadmaps

- Configurations are sampled by picking coordinates at random
- Sampled configurations are tested for collision
- Each configuration is linked by straight paths to its nearest neighbors
- The collision-free links are retained as local paths to form the PRM
- The start and goal configurations are included as milestones
- The PRM is searched for a path from start to goal

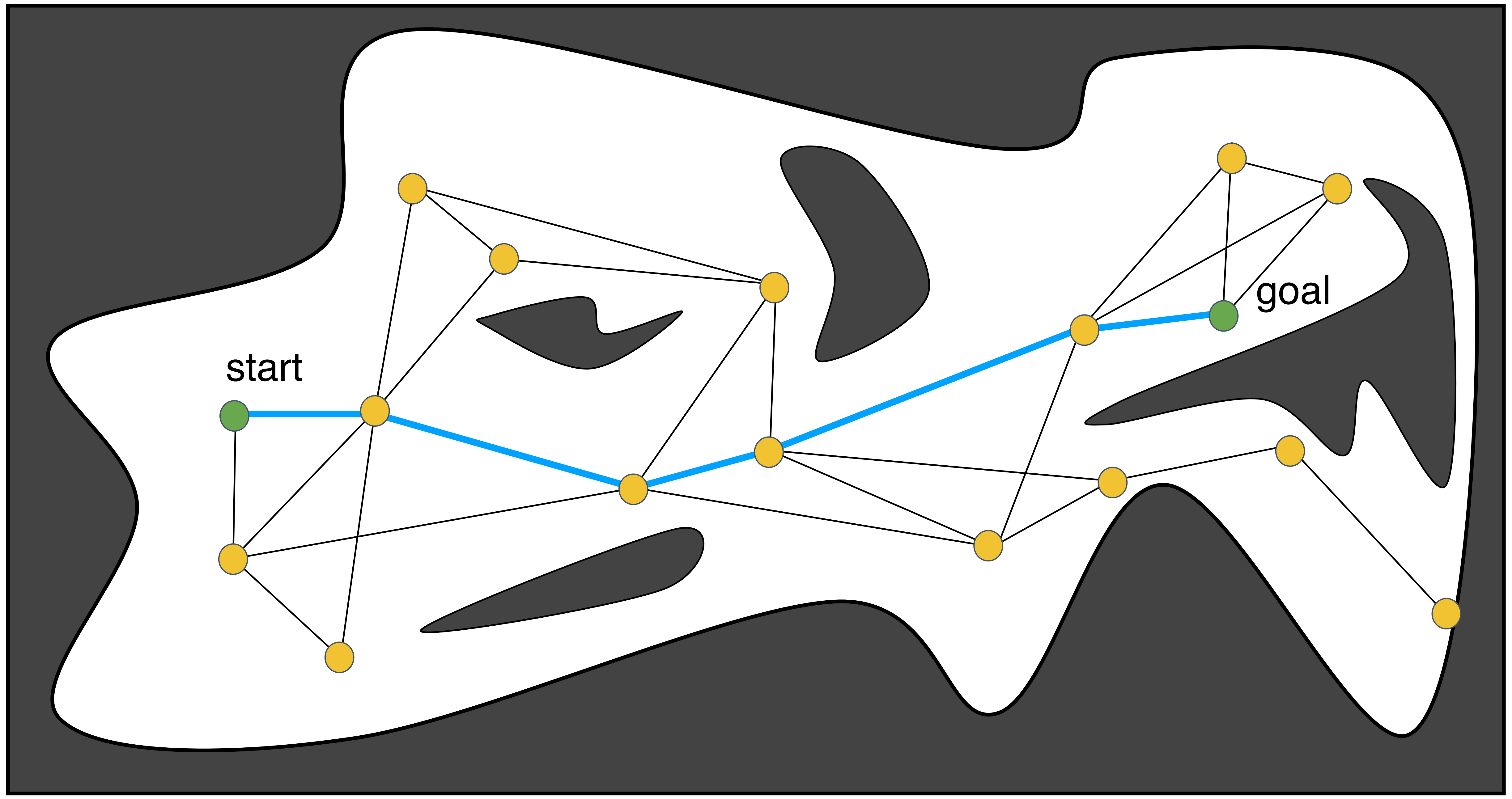


Probabilistic Roadmaps

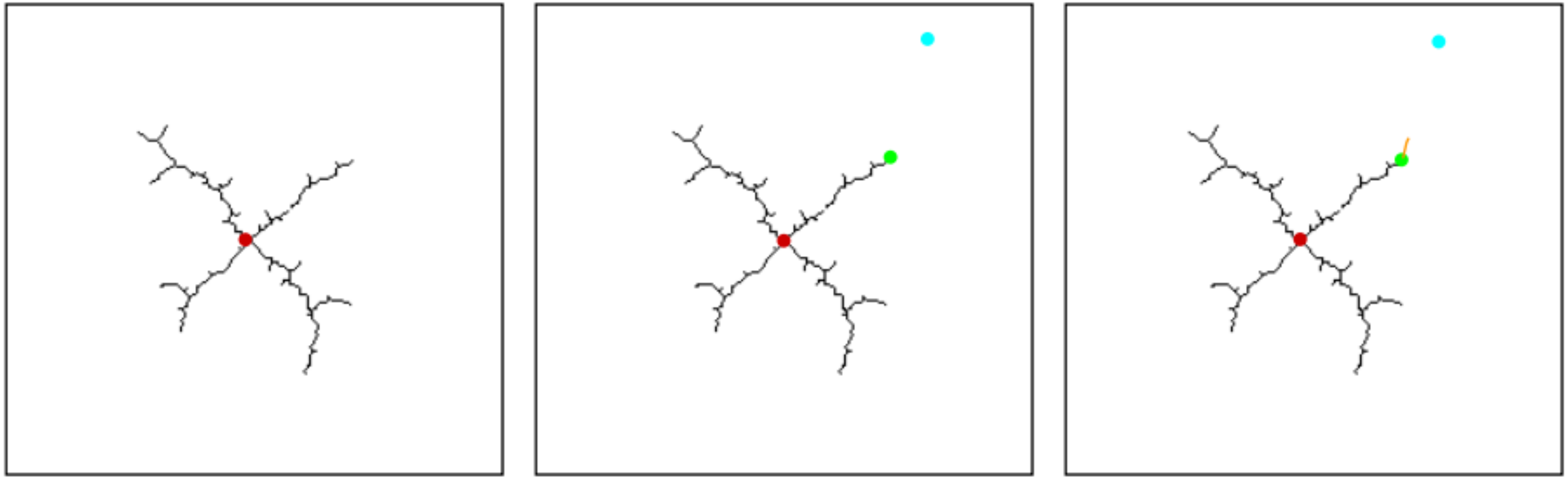
Considerations

- Single query/ multi query
- How are nodes placed?
  - Uniform sampling strategies
  - Non-uniform sampling strategies
- How are local neighbors found?
- How is collision detection performed?
  - Dominates time consumption in PRMs

realtime robotics



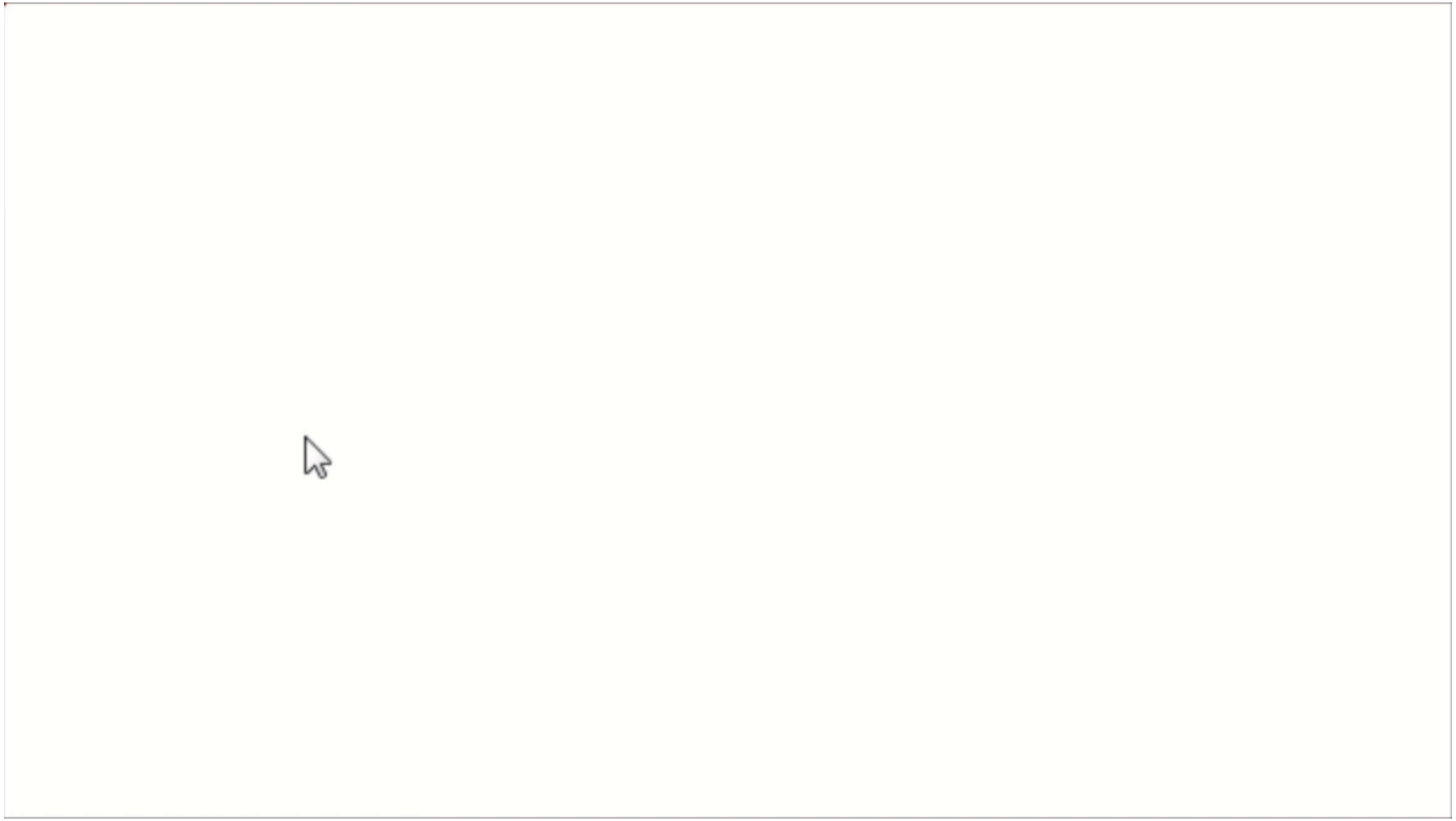
Rapidly Exploring Random Trees (RRT)



1. Maintain a tree rooted at the starting point
2. Choose a point at random from free space
3. Find the closest configuration already in the tree
4. Extend the tree in the direction of the new configuration

Rapidly Exploring Random Trees (RRT)

Uniform/ biased sampling



Aaron Becker, UH, Wolfram Player Example

Rapidly Exploring Random Trees (RRT)
Considerations

- Sensitive to step-size (Δq)
  - Small: many nodes, closely spaced, slowing down nearest neighbor computation
  - Large: Increased risk of suboptimal plans / not finding a solution
- How are samples chosen?
  - Uniform sampling may need too many samples to find the goal
  - Biased sampling towards goal can ease this problem
- How are closest neighbors found?
- How are local paths generated?
- Variations
  - RRT Connect, A*-RRT, Informed RRT*, Real-Time RRT*, Theta*-RRT, etc.

Next class… graph search…
see you Tuesday!