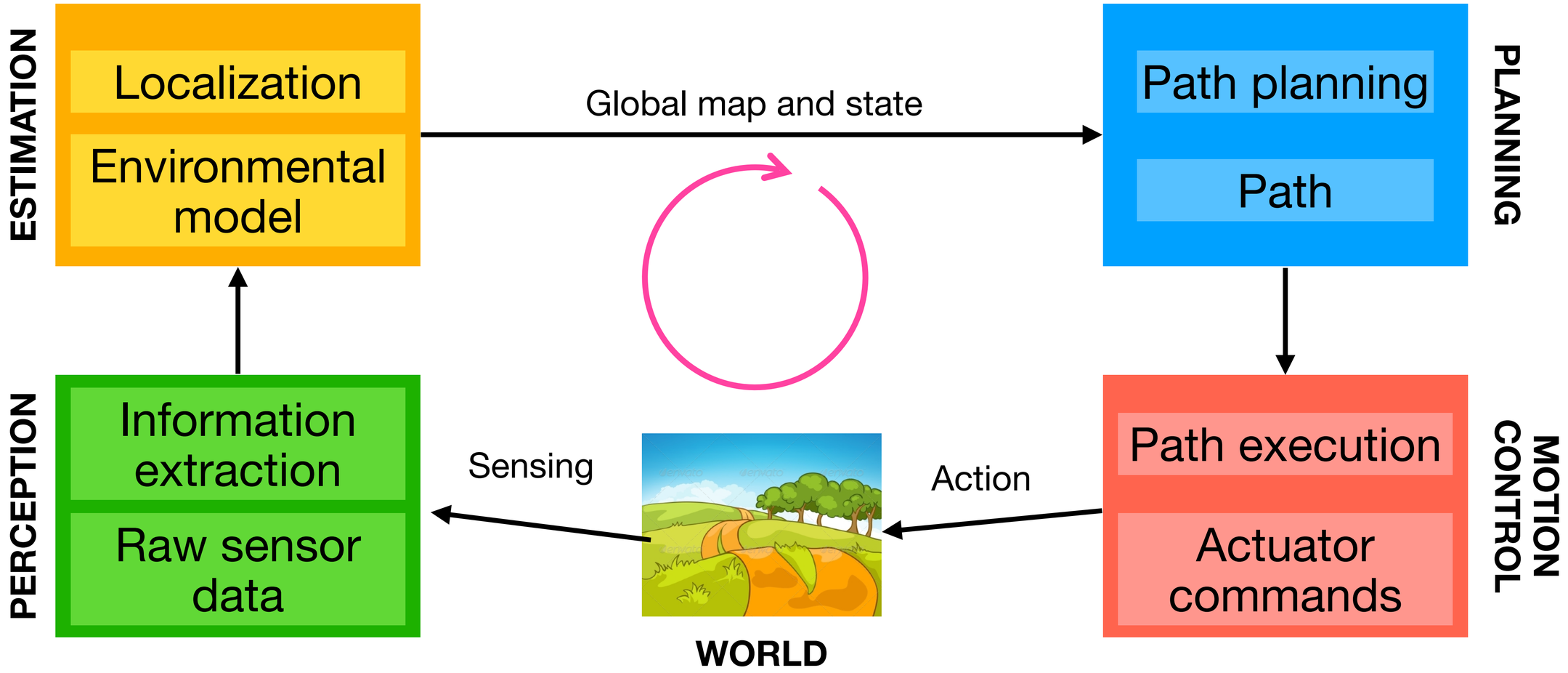
Maps and Graphs

Fast Robots, ECE4160/5160, MAE 4190/5190

E. Farrell Helbling, 3/12/26
Slides adapted from Prof. Kirstin Petersen

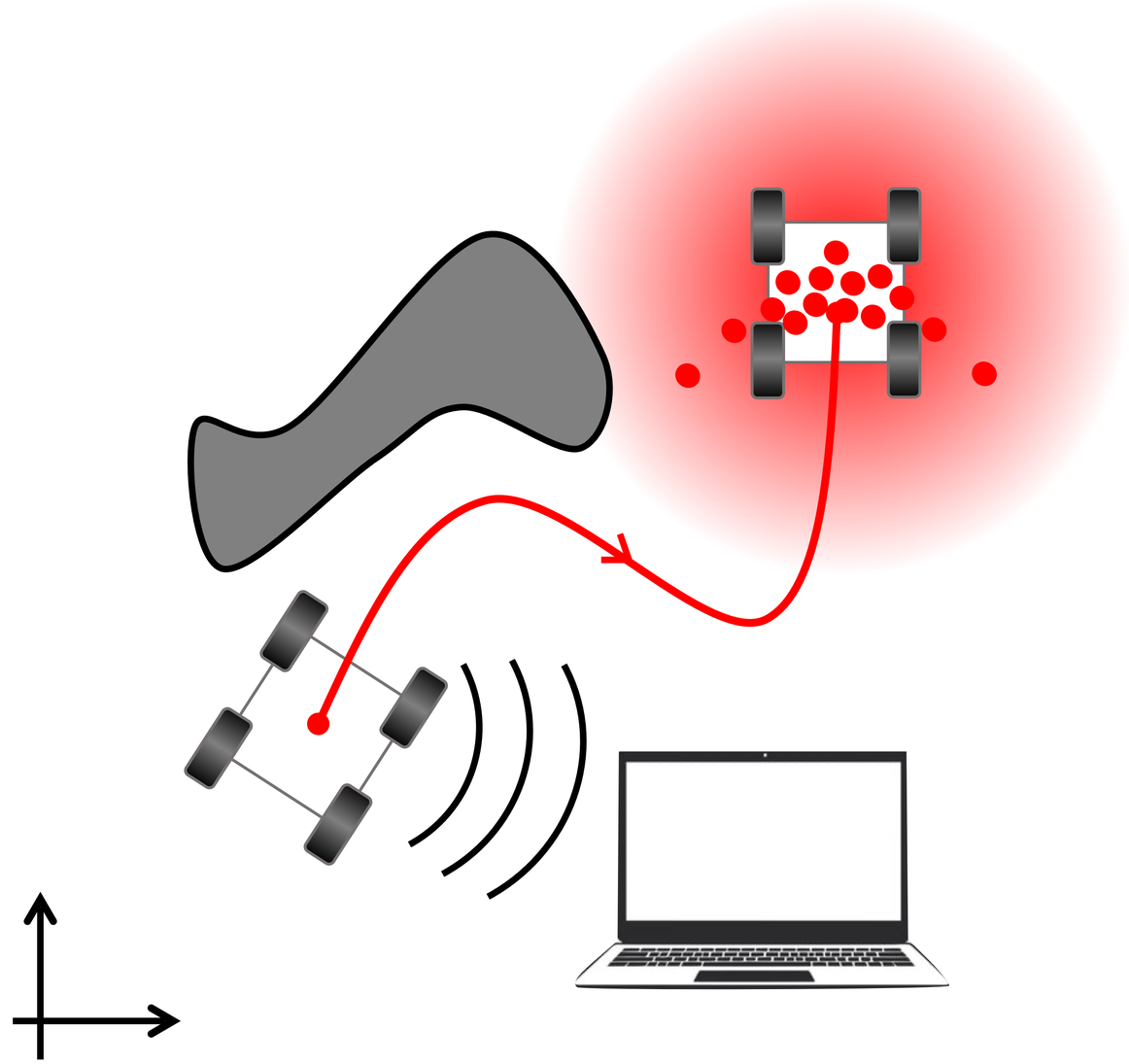
Navigation

- **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 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

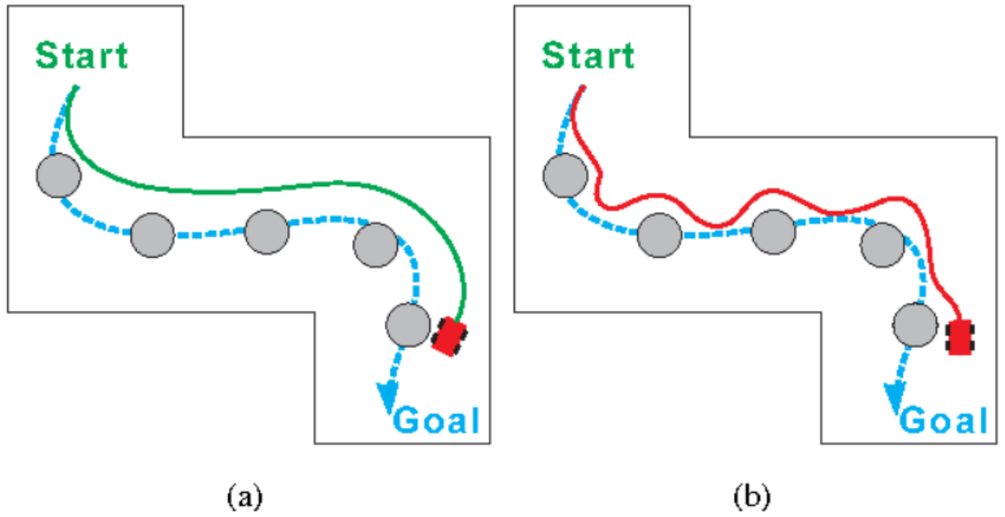
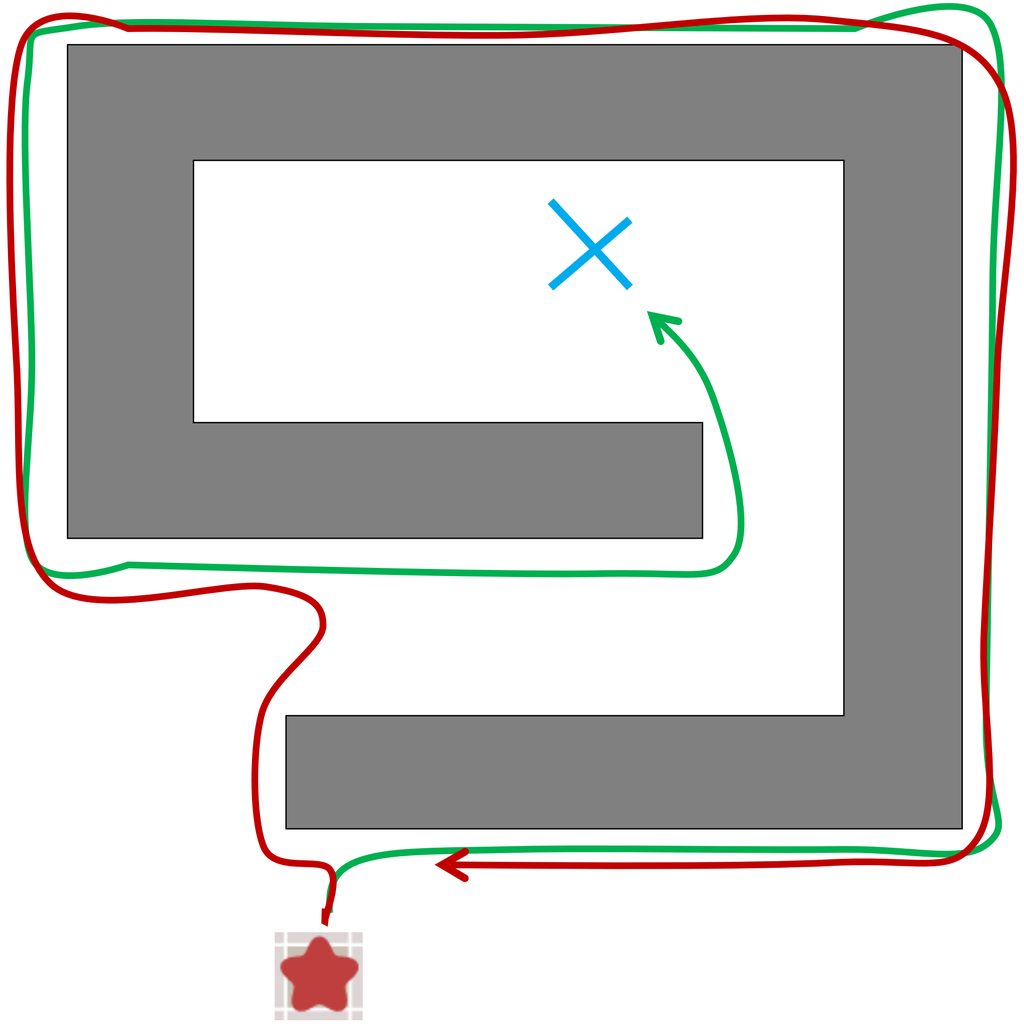


Fig. 1. Dashed blue spline is global path: a) Green spline is ideal local path; b) Red spline is actual local path

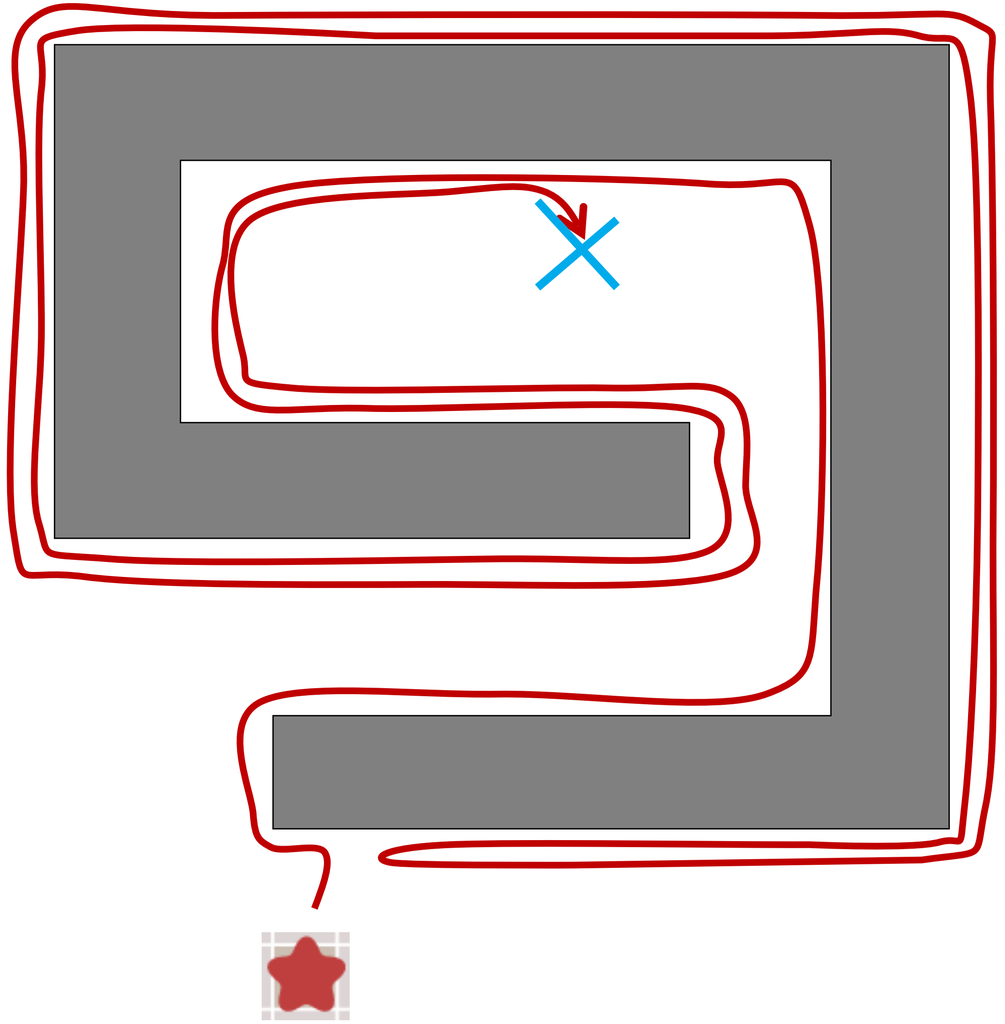
Bug 0

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



Bug 1

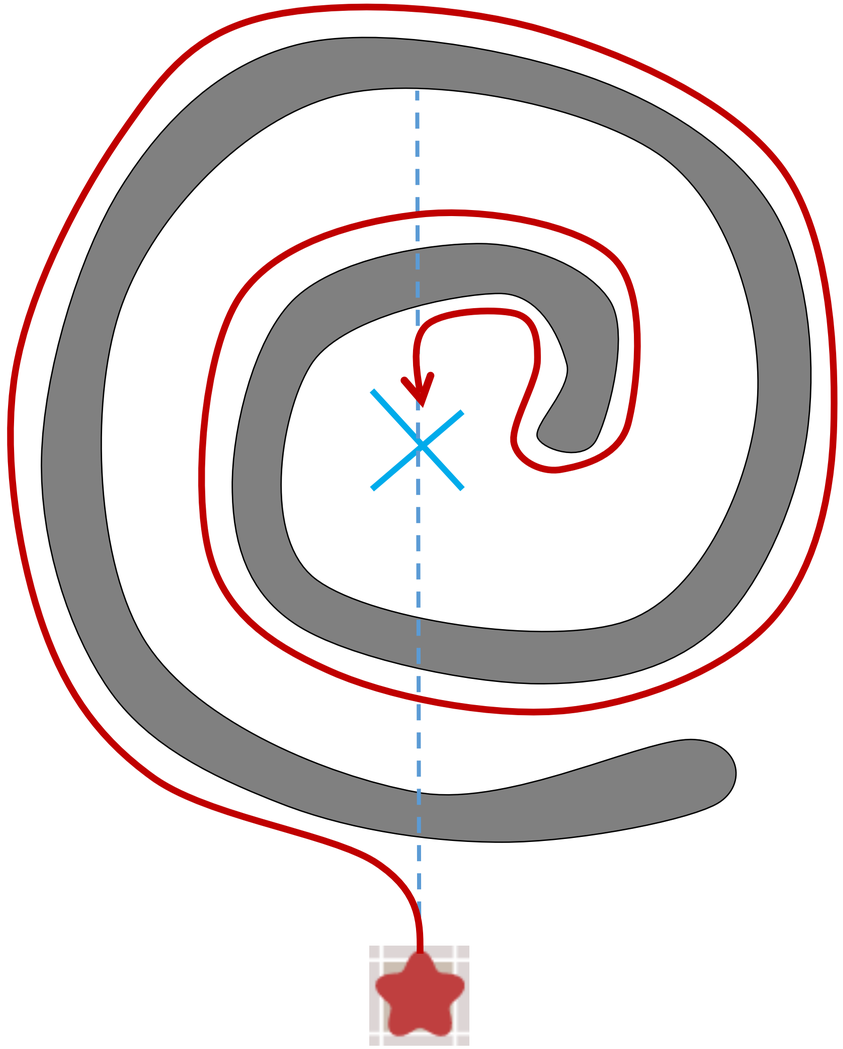
- Sensor Assumptions
  - Direction to the goal
  - Detect walls
  - Odometry
- Algorithm
  - Go towards goal
  - Follow obstacles *and remember how close you got to the goal*
  - Return to the closest point, loop



Bug 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?



Battle of the bugs (1 vs 2)

Exhaustive search

Bug 1
Layout 2

Greedy search

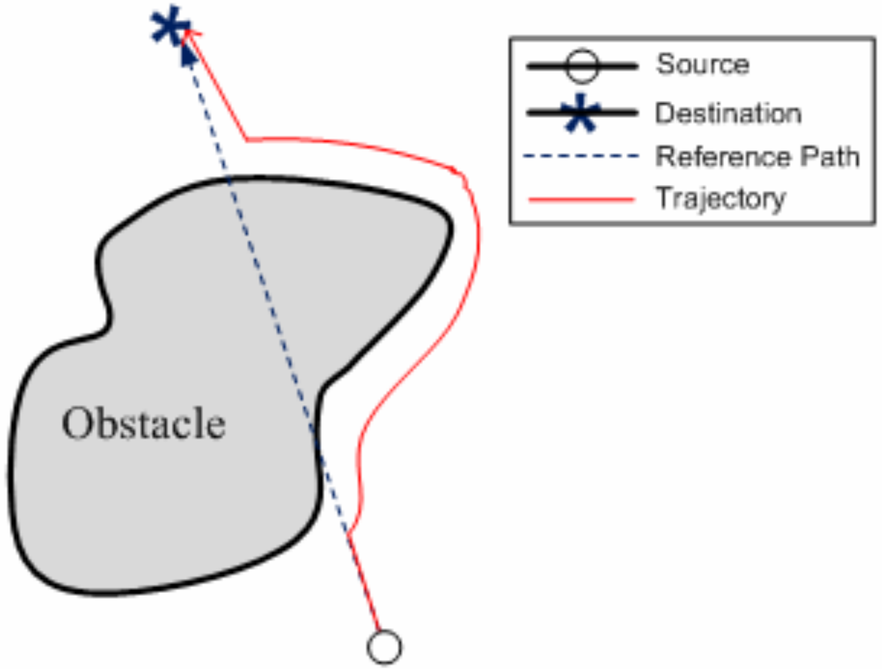
Bug 2
Layout 2

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

Bug algorithms

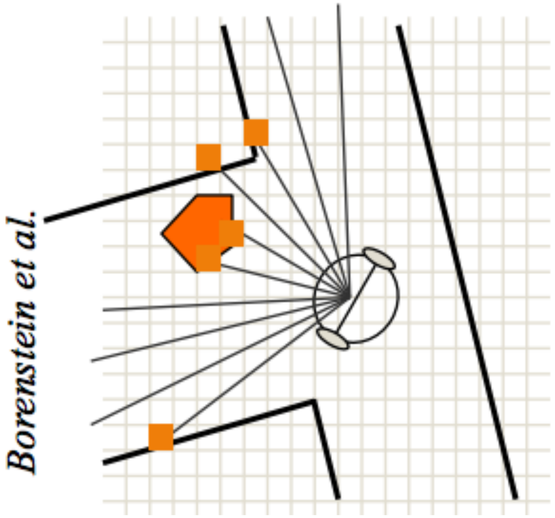
- Uses local knowledge and the direction and distance to the goal
- Basic idea
  - Follow the contour of obstacles until you see the goal
  - State 1: seek goal
  - State 2: follow wall
- Different Variants: Bug0, Bug1, Bug2

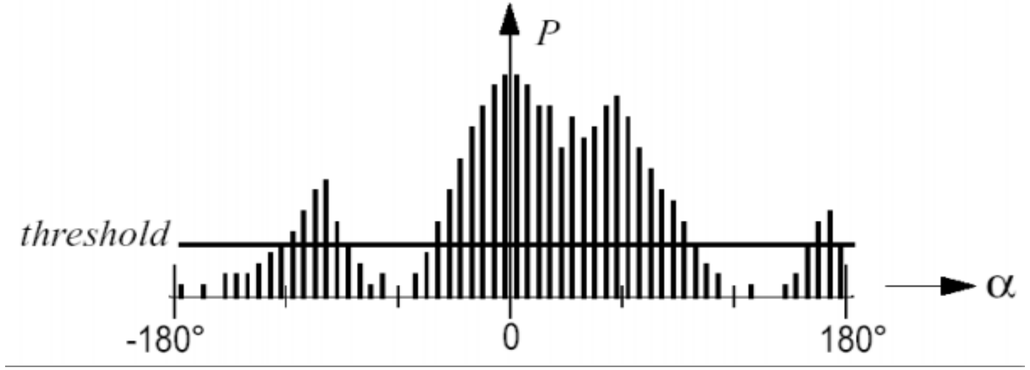
- The robot motion behavior is reactive
- Issues if the instantaneous sensor readings do not provide enough information or are noisy



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

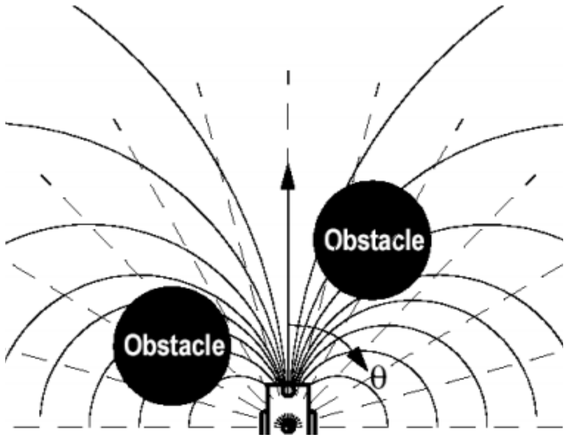


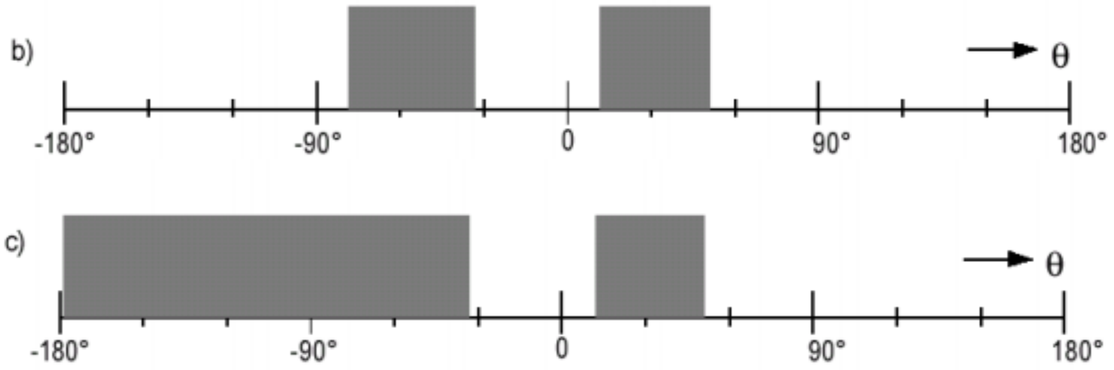


http://www.personal.umich.edu/~johannb/Papers/paper16.pdf

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

- Takes into account robot dynamics (velocity space)
- A search space, V_s, 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
- Dynamic velocities, V_d, include those the robot can achieve given limits
- The search space is then $V_r = V_s \cap V_a \cap V_d$
- Cost function: $G(v,\omega) = \sigma(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot velocity(v,\omega))$

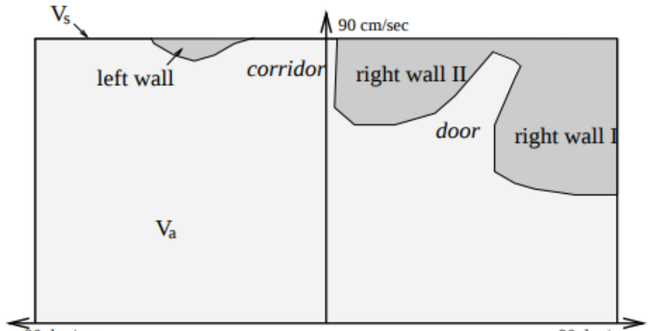
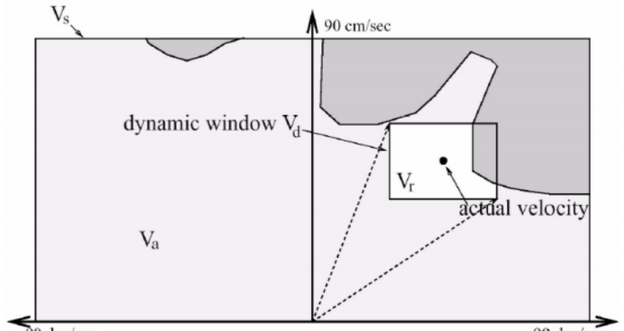


Figure 4. Velocity space



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

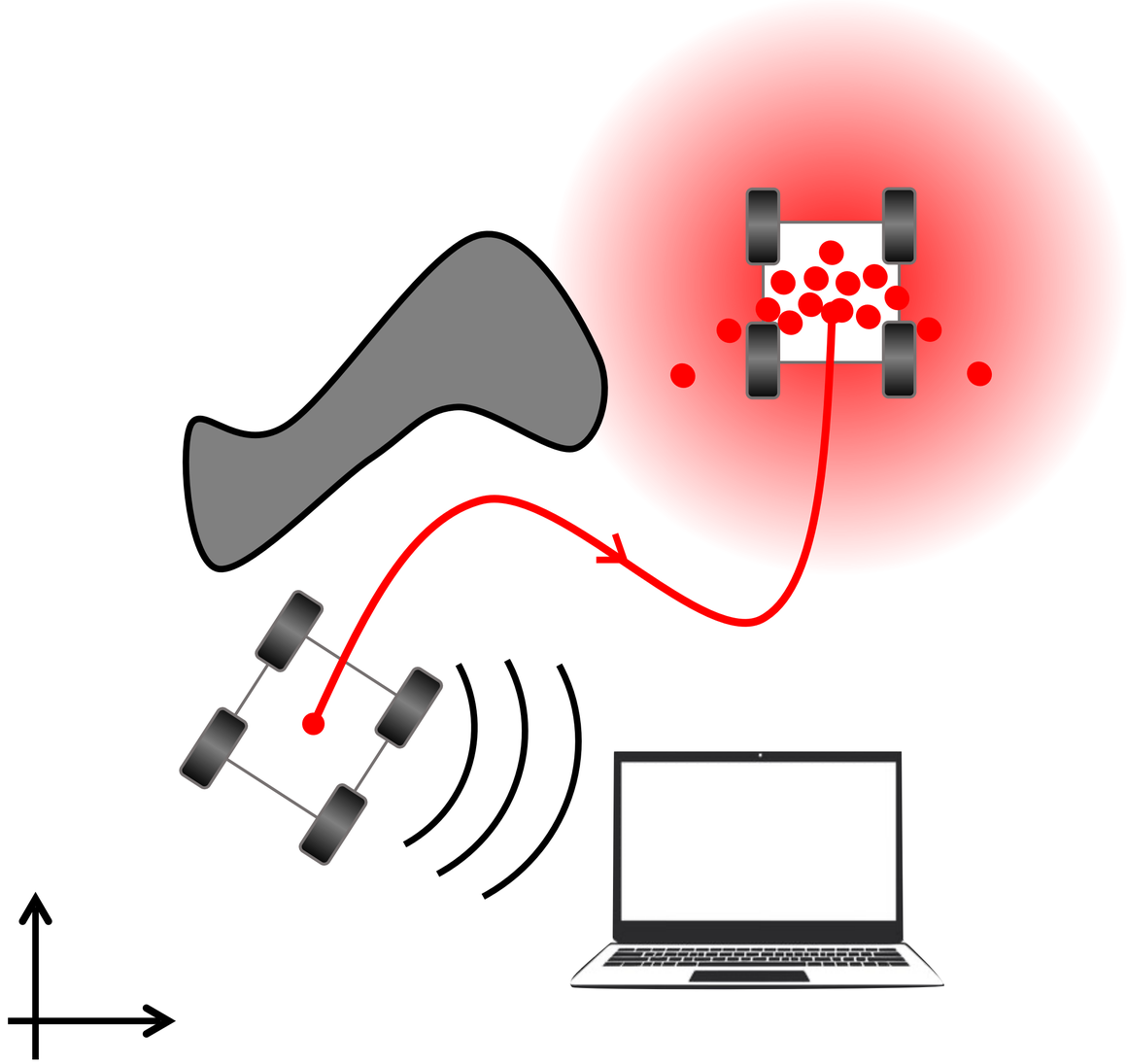
Local Planners

- 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

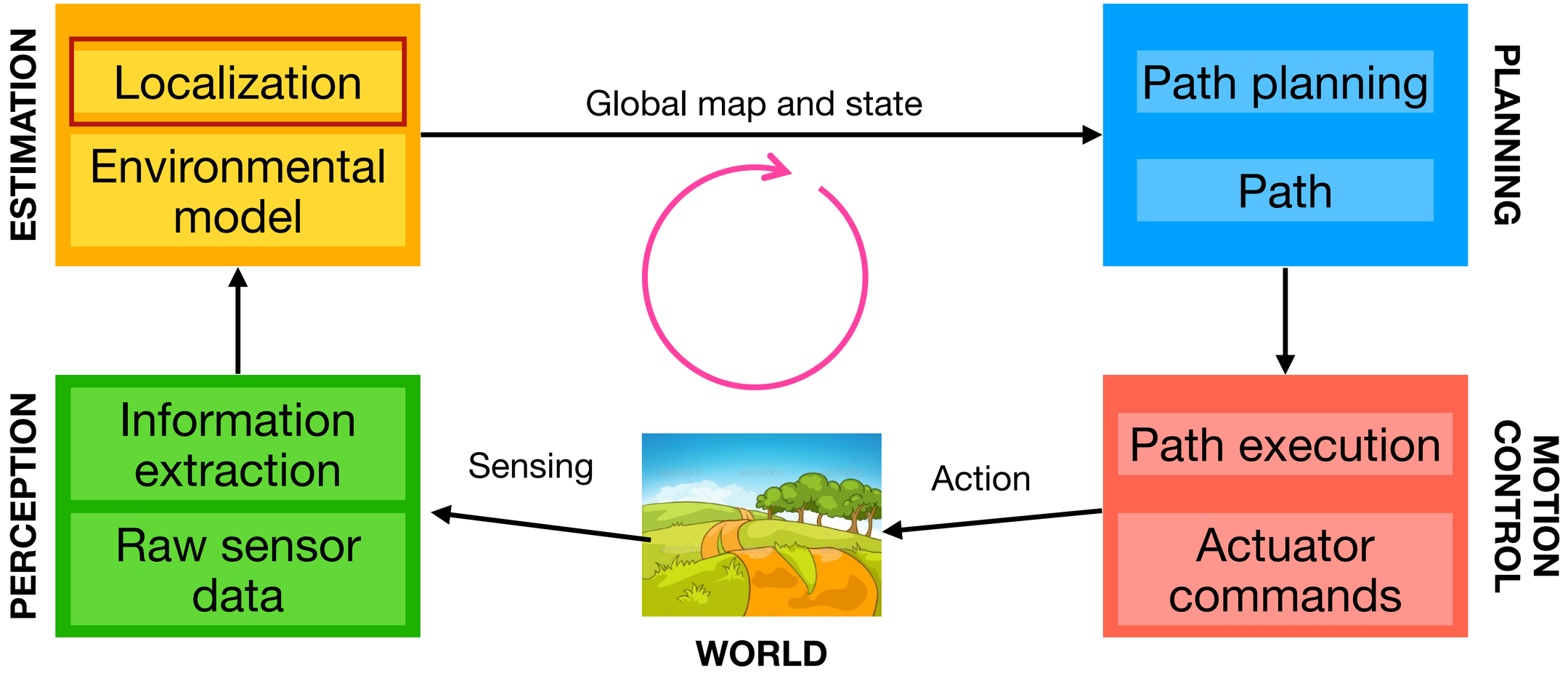
Next module on navigation

- 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

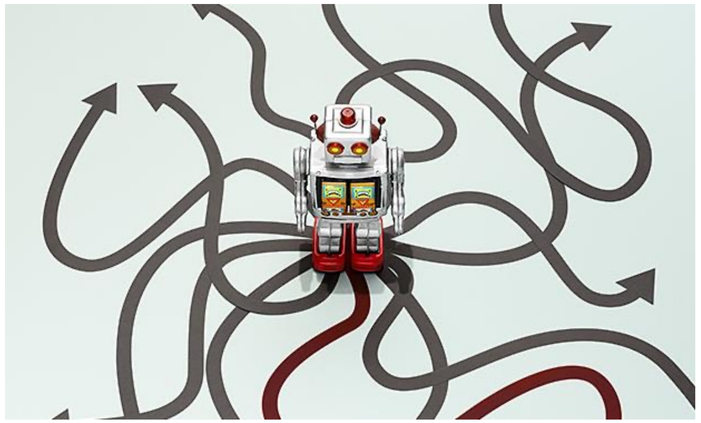
Position Tracking

- Initial **robot pose is known**
- Either deterministically (odometry) or through Bayesian statistic (motion and sensor models)
- It is a "**local**" problem, as the uncertainty is local (often small) and confined to a region near the robot's true pose

Global Localization

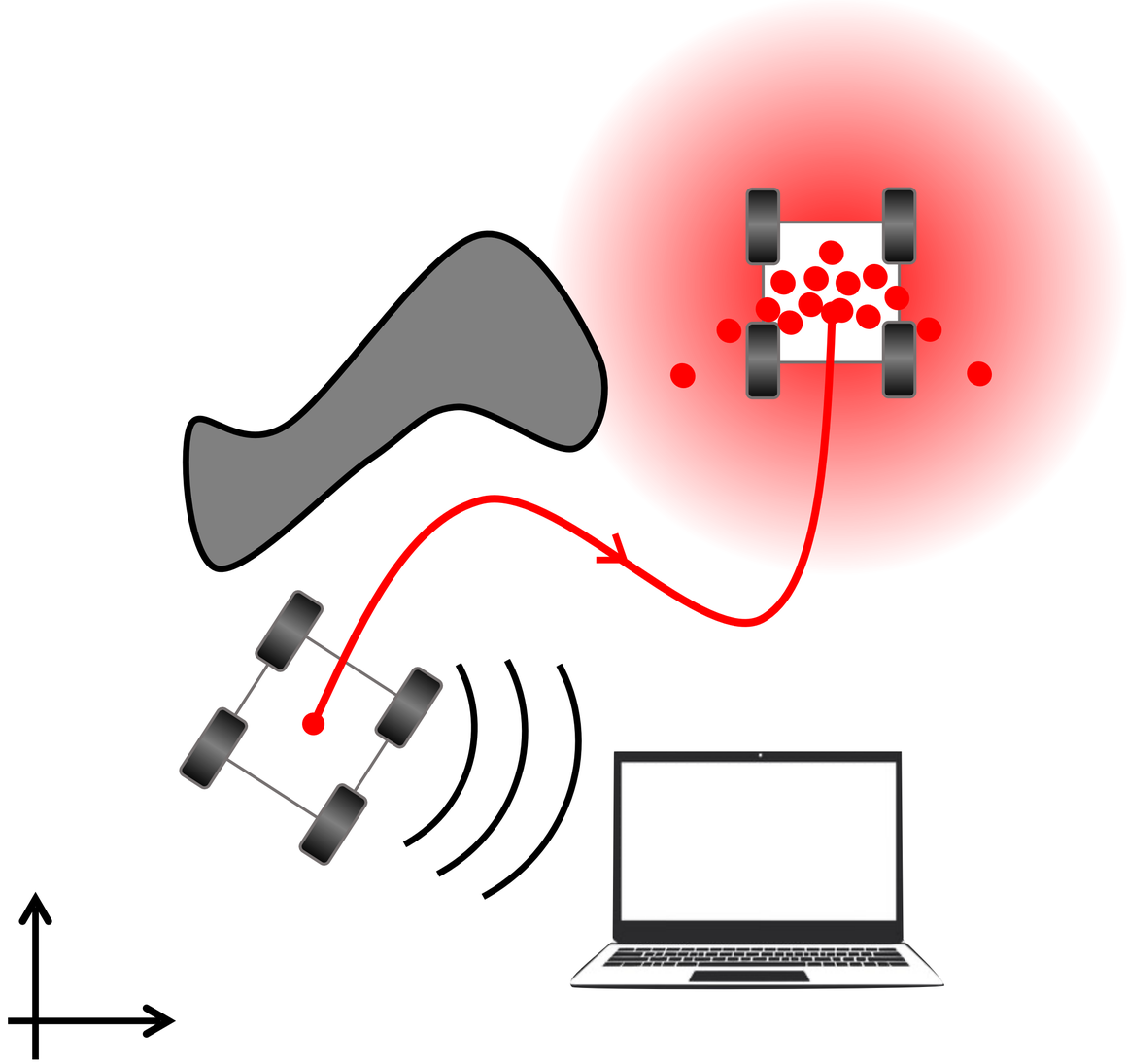
- Initial **robot pose is unknown**
- Need to estimate position from scratch
- A more difficult "**global**" problem, where you cannot assume boundedness in pose error

kidnapped robot problem



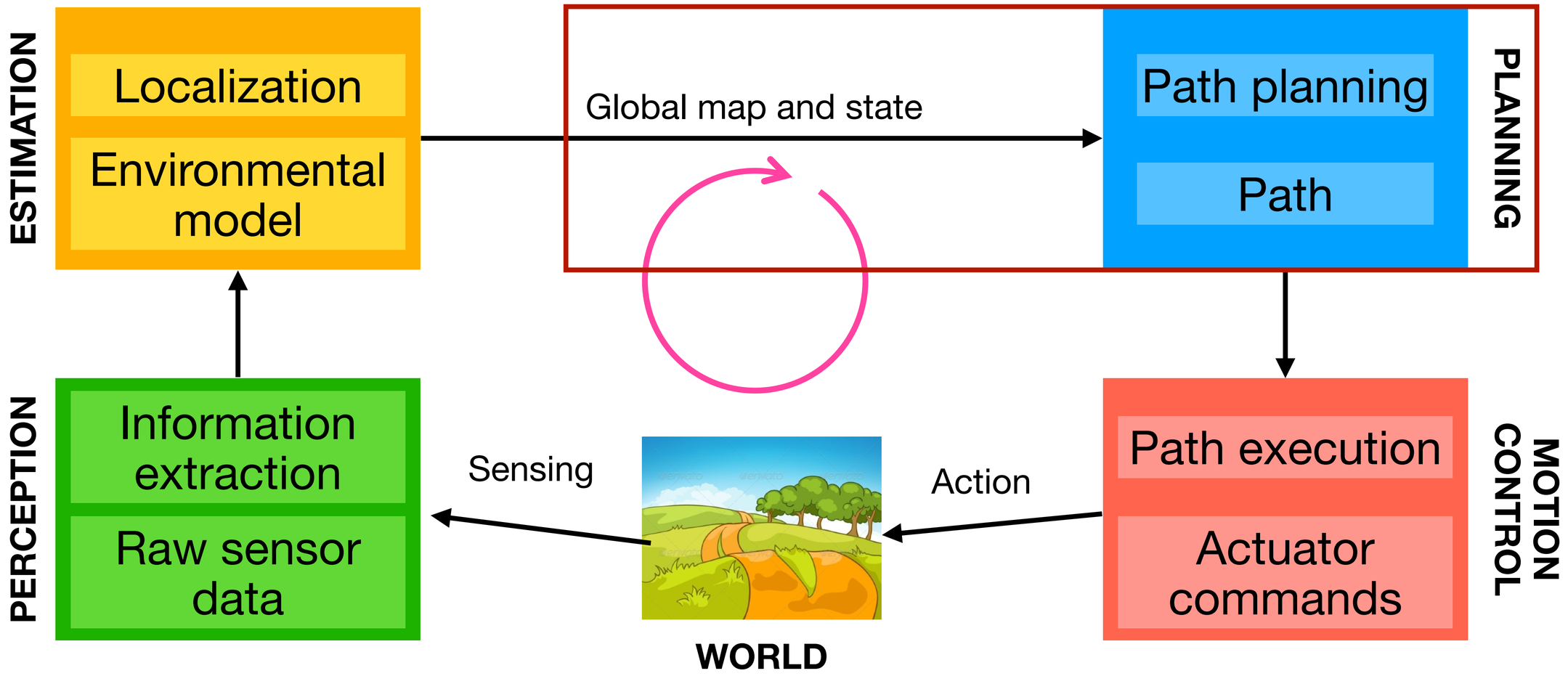
Next module on navigation

- 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

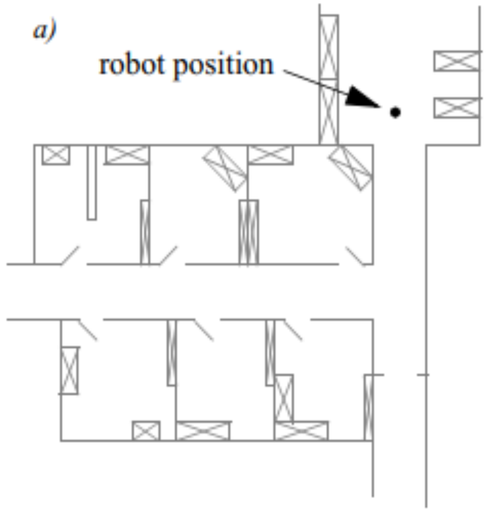
b) Line-based map

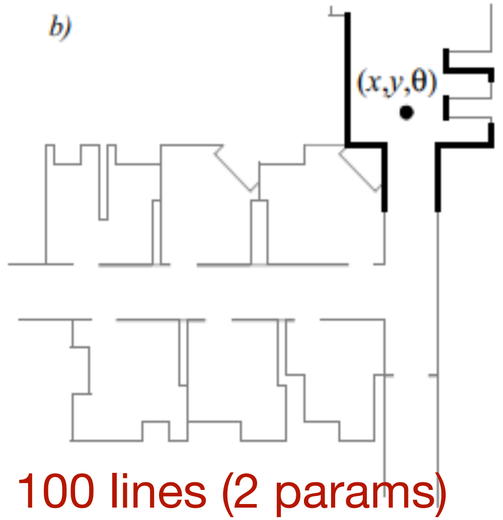
c) Occupancy grid-based map

d) Topological map

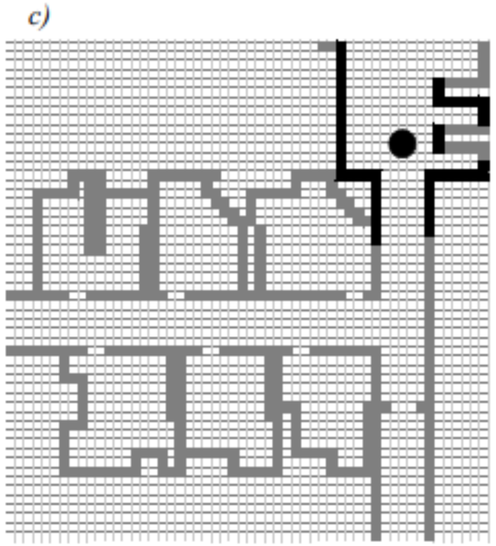
Important Properties

- Memory allocation
- Computation

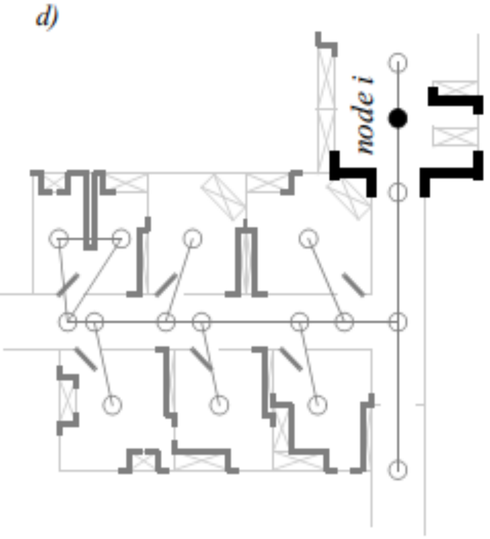




100 lines (2 params)



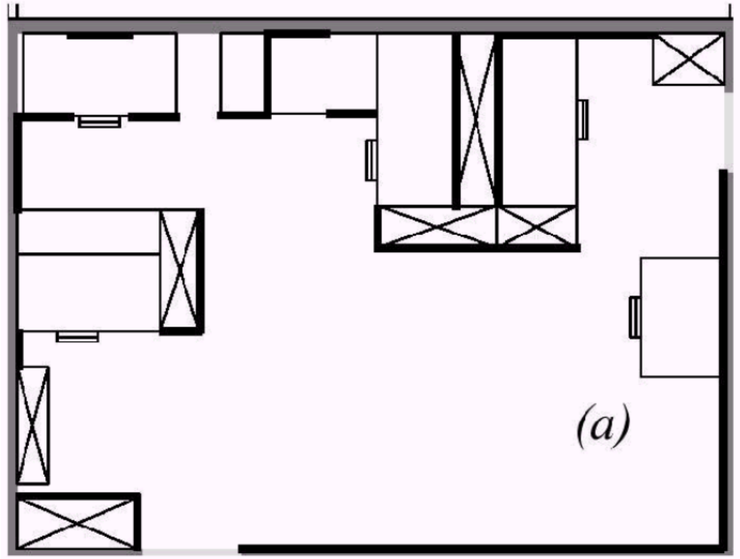
3000 grid cells (0.5m)

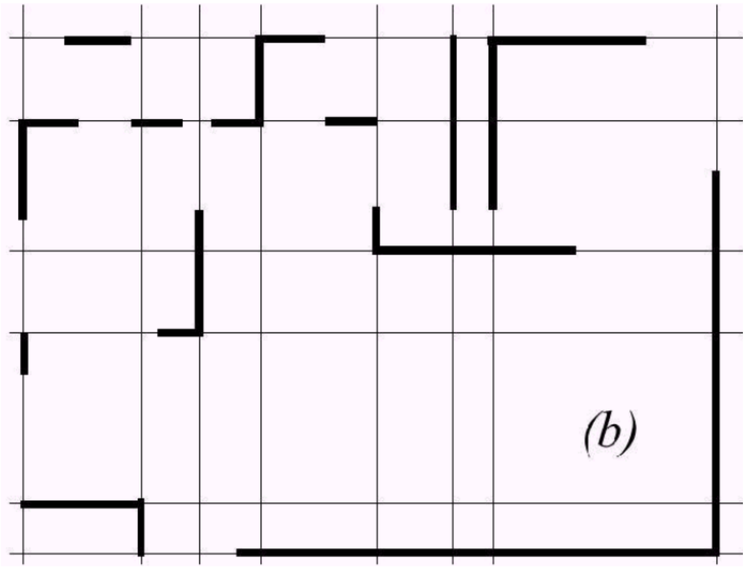


18 nodes

Continuous Representations

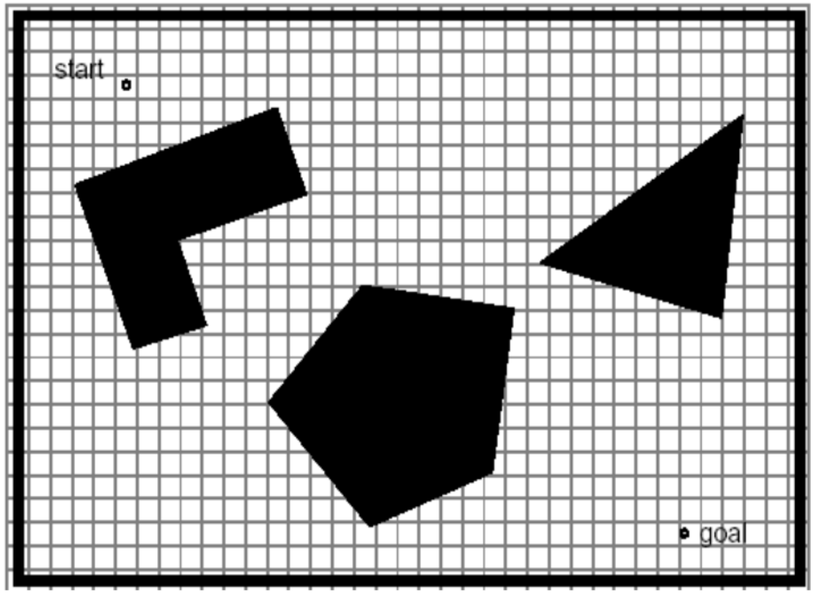
- Exact decomposition of the environment
- Used mainly in 2D representations
- Closed-world assumption
- Storage proportional to object density
- Example: Continuous line representations
  - Using range finders, we can extract lines/ line segments in the environment

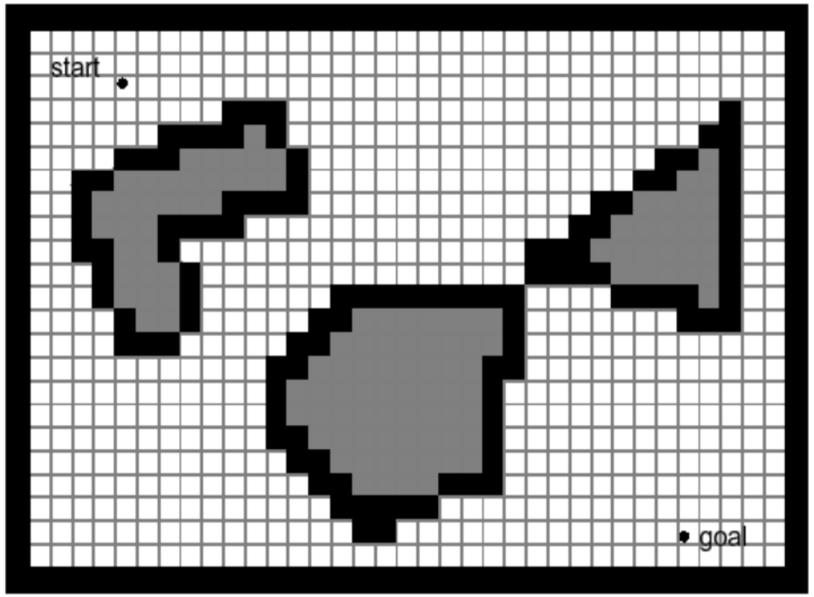


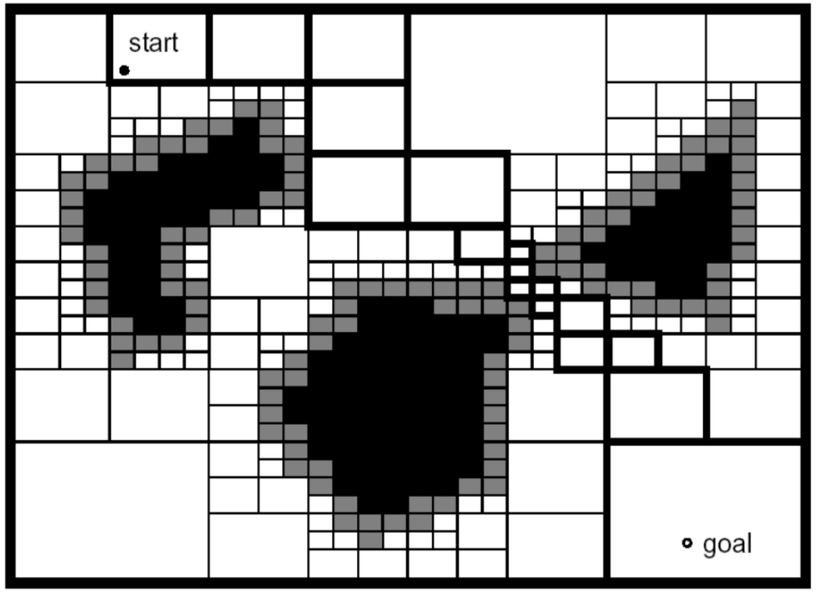


Cell Decomposition

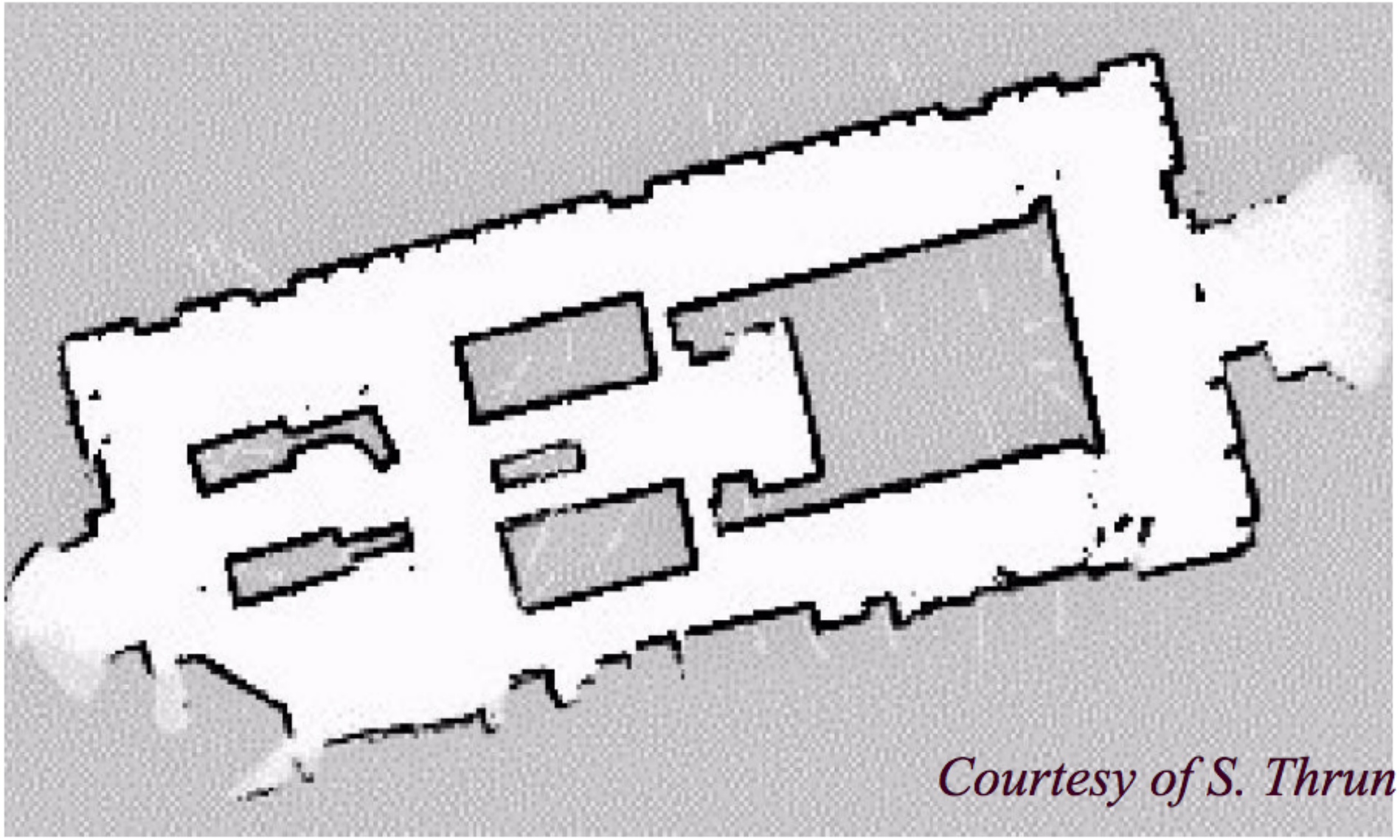
- Fixed: Tesselate the world at a fixed resolution
- Approximate features given the resolution
- Most commonly used: occupancy grid
- Adaptive: Tesselate the world at varying resolutions, finer near objects





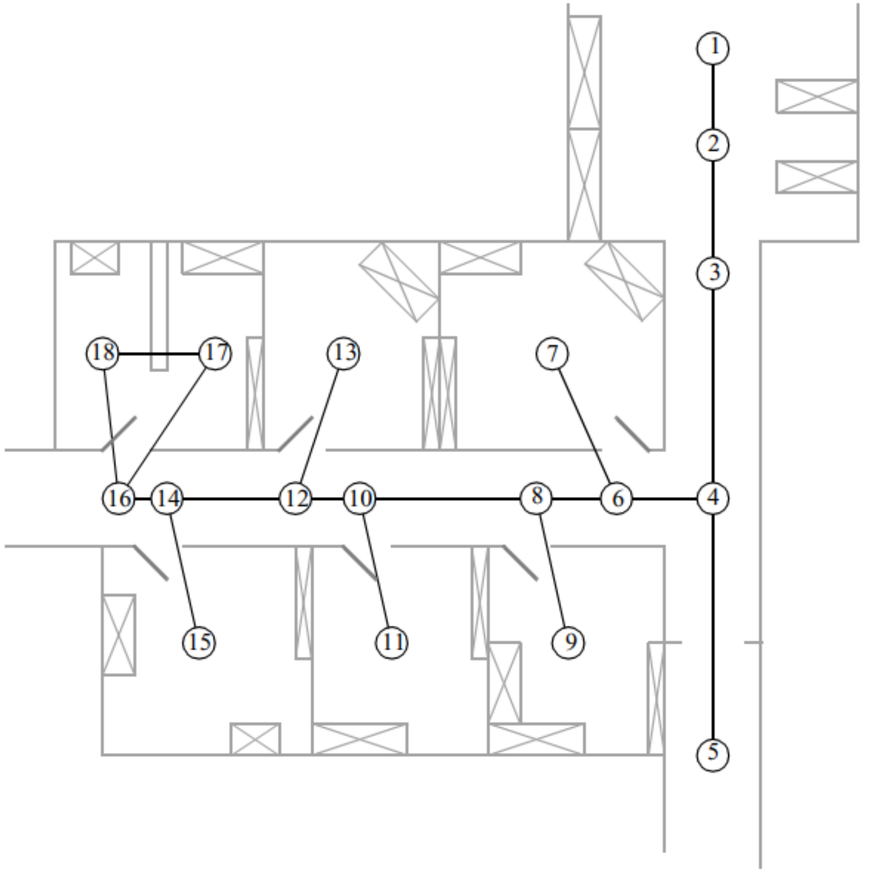


Fixed Decomposition



Topological decomposition

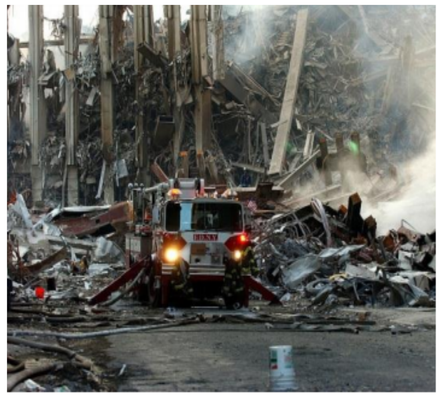
- A topological representation is a graph that specifies nodes and edges
  - Nodes denotes areas in the environment
  - Edges describe environment connectivity
- Robots can…
  - …detect their current position in terms of the nodes of the topological graph
  - …travel between nodes using robot motion



Topological decomposition

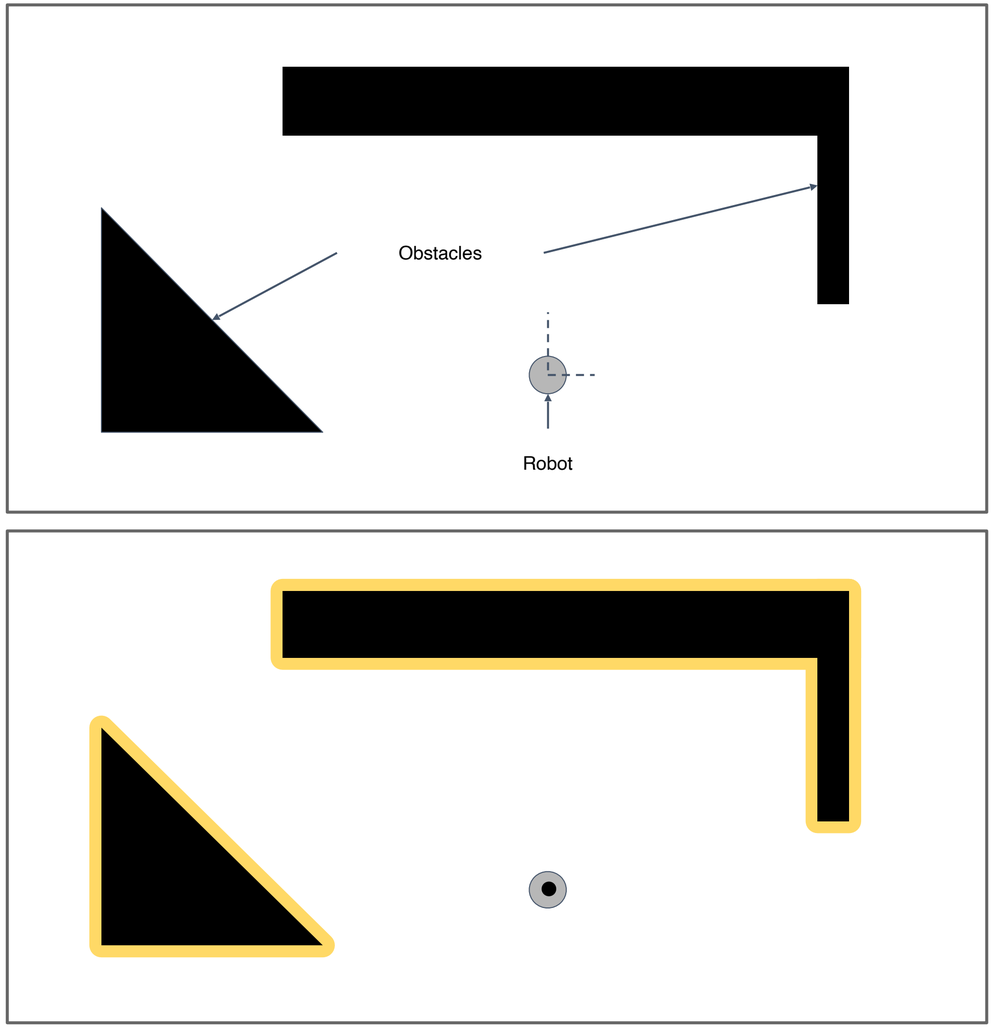
- A topological representation is a graph that specifies nodes and edges
  - Nodes denotes areas in the environment
  - Edges describe environment connectivity
- Robots can…
  - …detect their current position in terms of the nodes of the topological graph
  - …travel between nodes using robot motion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

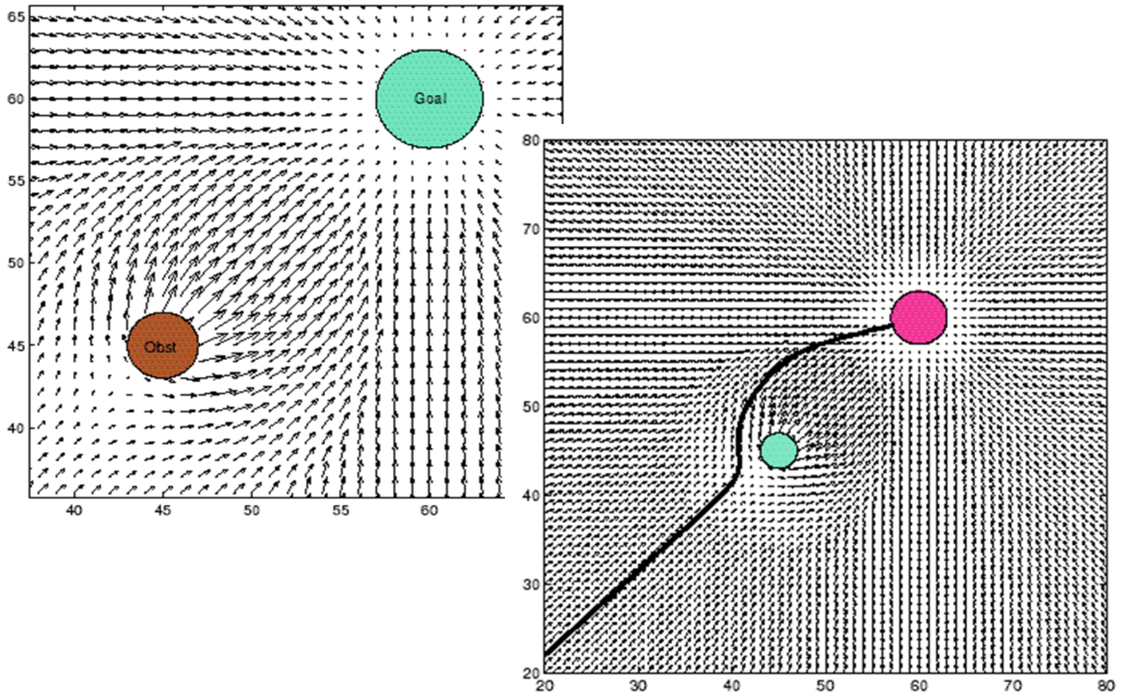
Map Representation Considerations

- 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

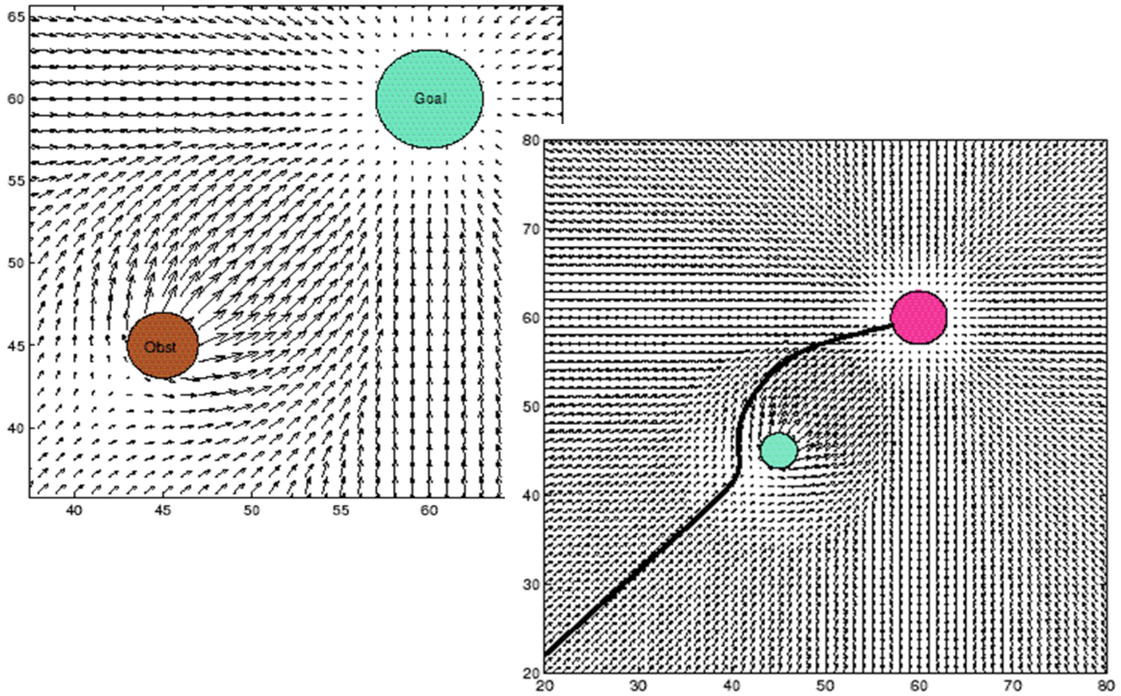
Real World → Configuration Space → Map Representation → Graph Construction → Graph Search



Common alternatives

- Optimal control
- Potential fields

Modeling path planning as a graph search problem

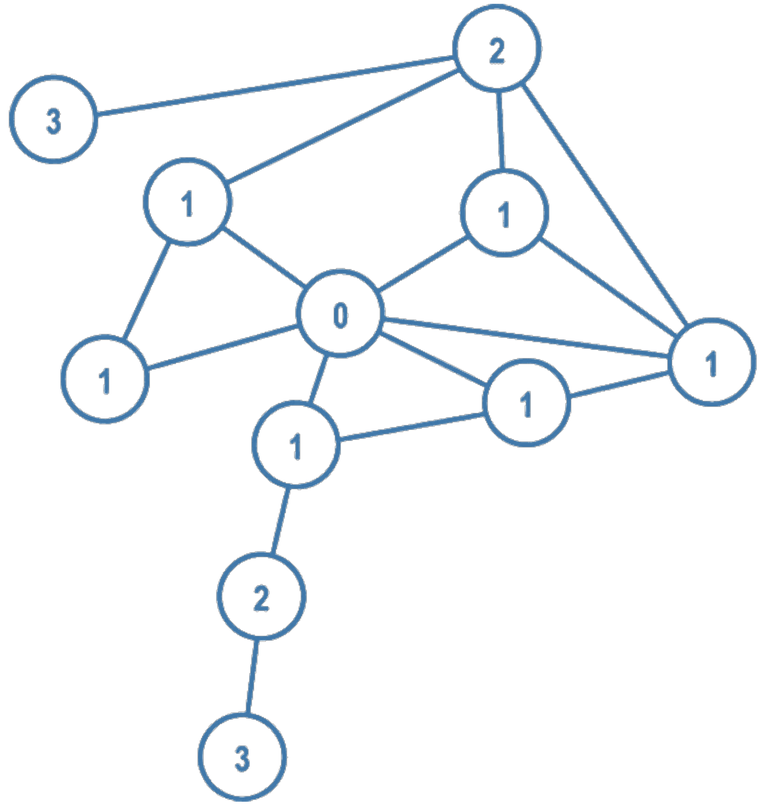


- Topological Graphs
- Cell decomposition
- Visibility Graphs
- RRT
- PRM

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

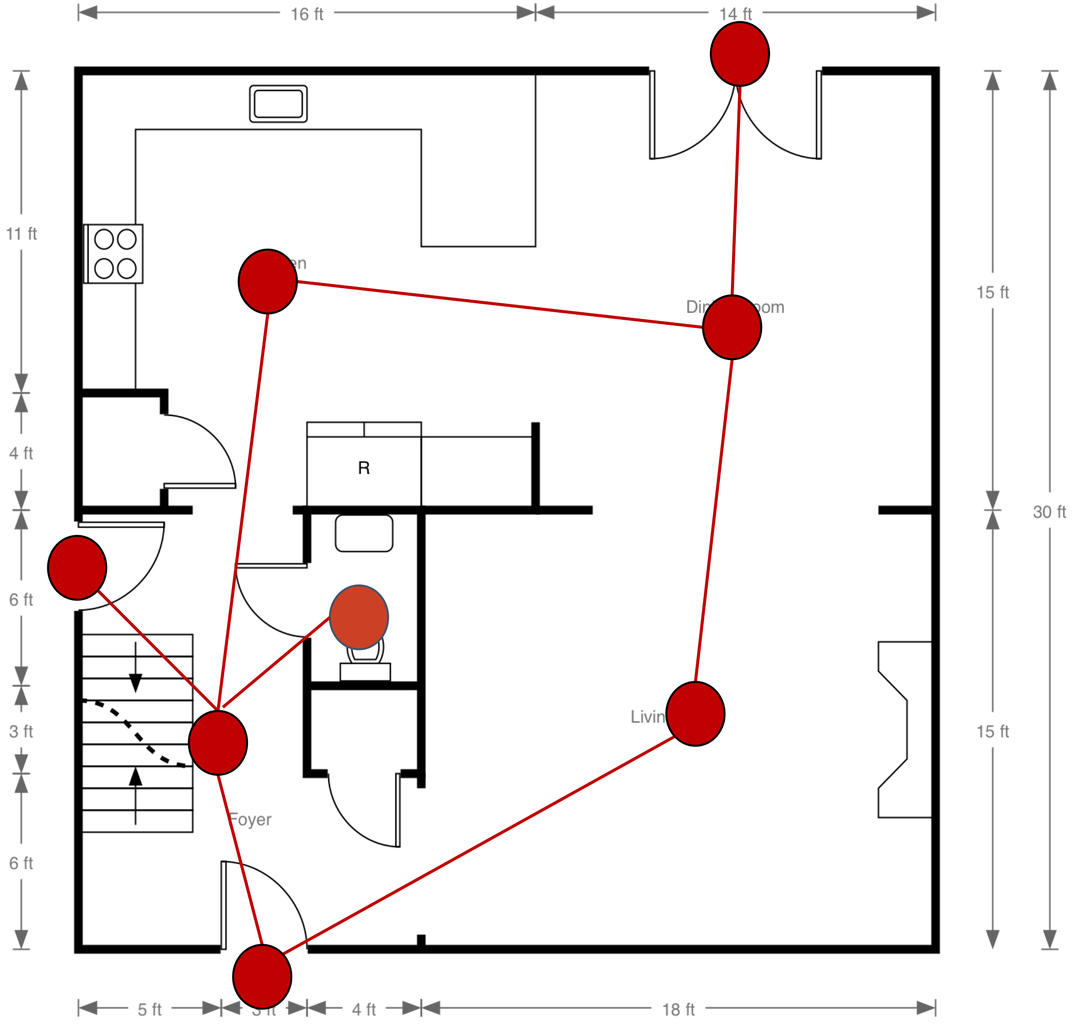
1. Divide space into simple, connected regions, or “cells”
2. Determine adjacency of open cells
3. Construct a connectivity graph
4. Find cells with initial and goal configuration
5. Search for a path in the connectivity graph to join them
6. From the sequence of cells, compute a path within each cell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



Geometry-based planners

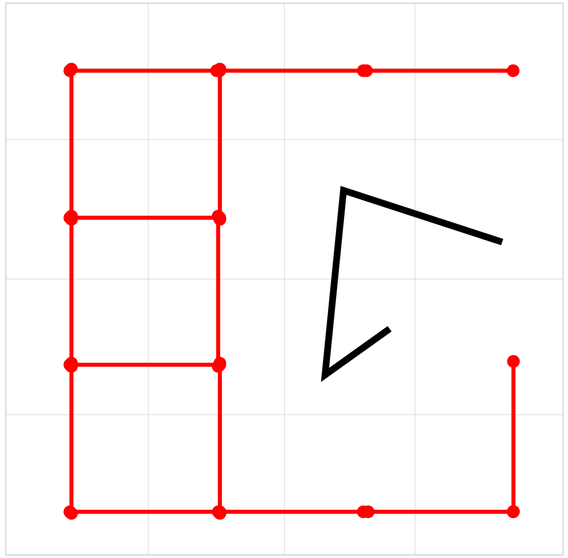
Topological Maps

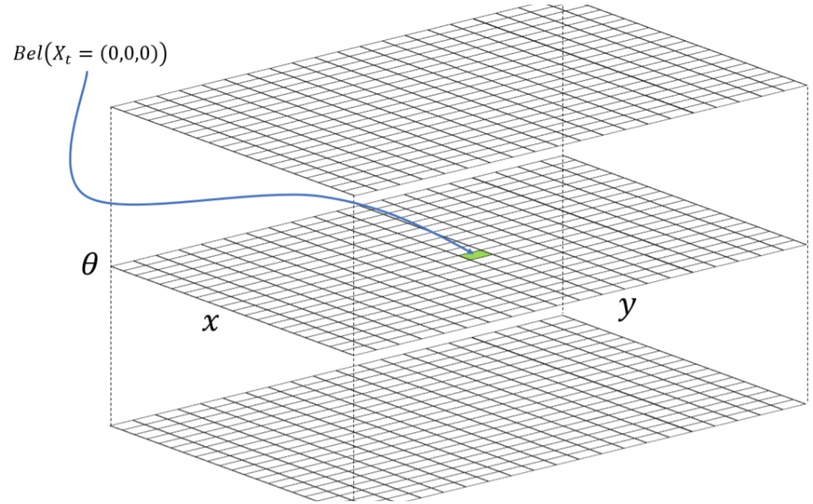
- Good abstract representation
- Tradeoff in # of nodes
  - Complexity vs. accuracy
  - Efficient in large, sparse environments
  - Loss in geometric precision
- Edges can carry weight
- Con: limited information



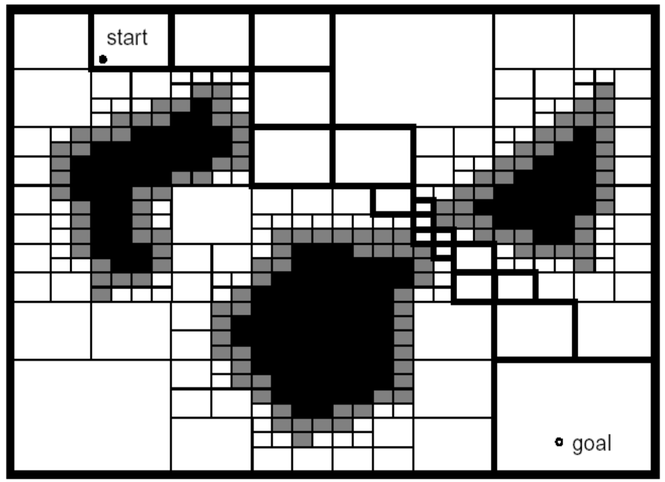
Fixed Cell Decomposition

(Lab 9-12)

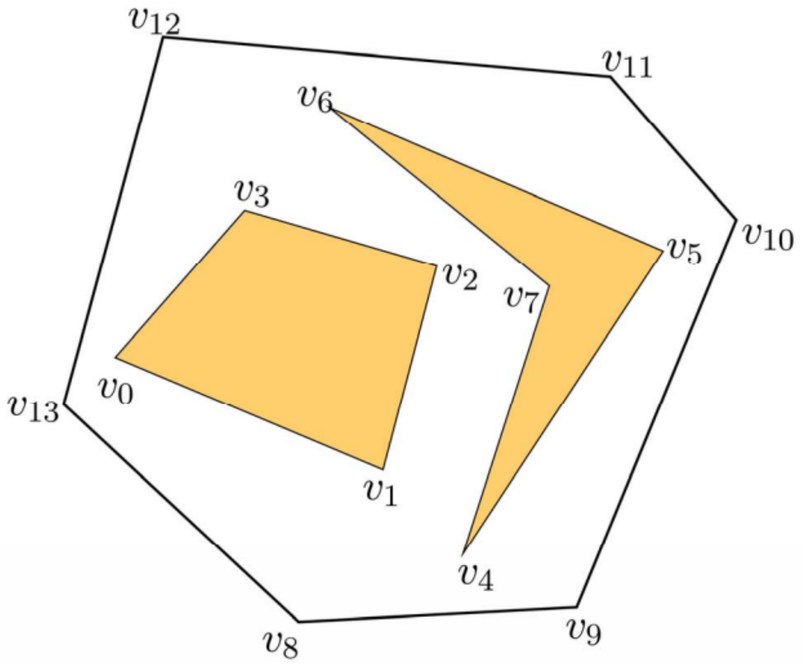




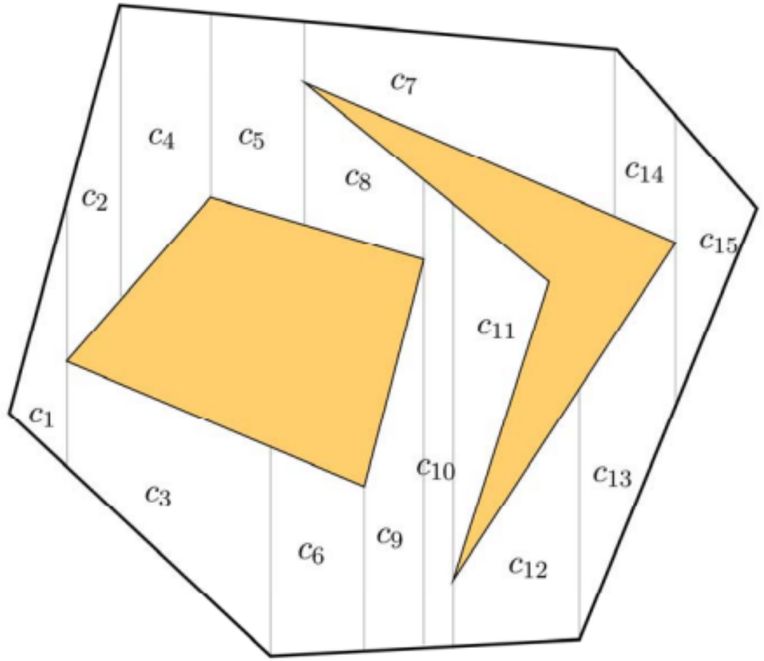
Adaptive Cell Decomposition

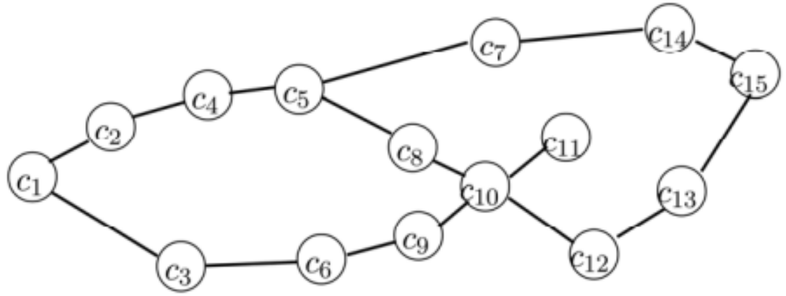


Trapezoidal Cell Decomposition



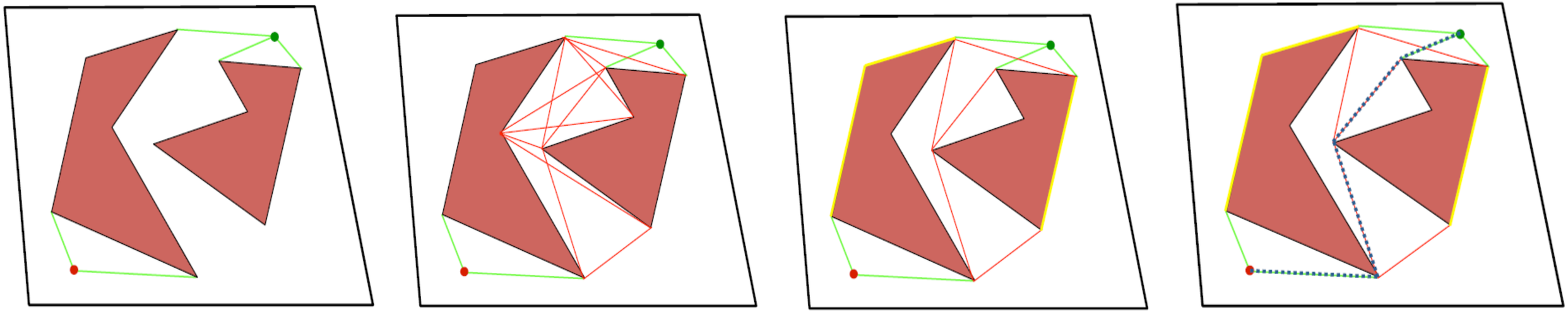
RI 16-735 Howie Choset





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



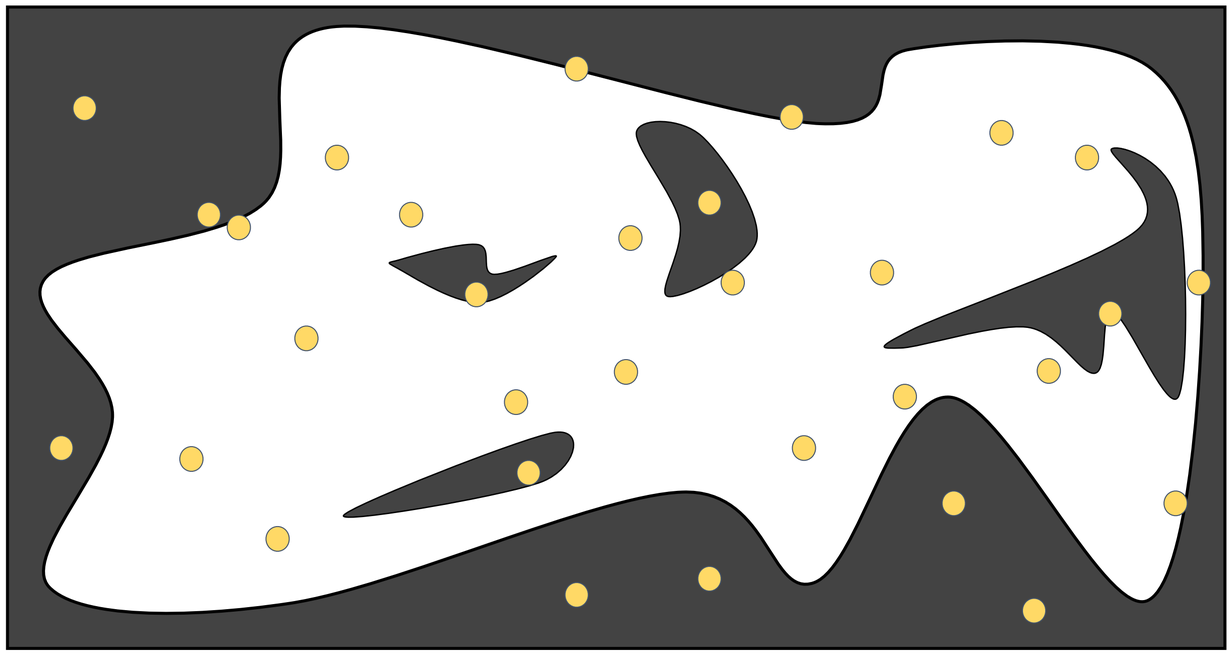
Ioannis Rekleitis, South Carolina

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)

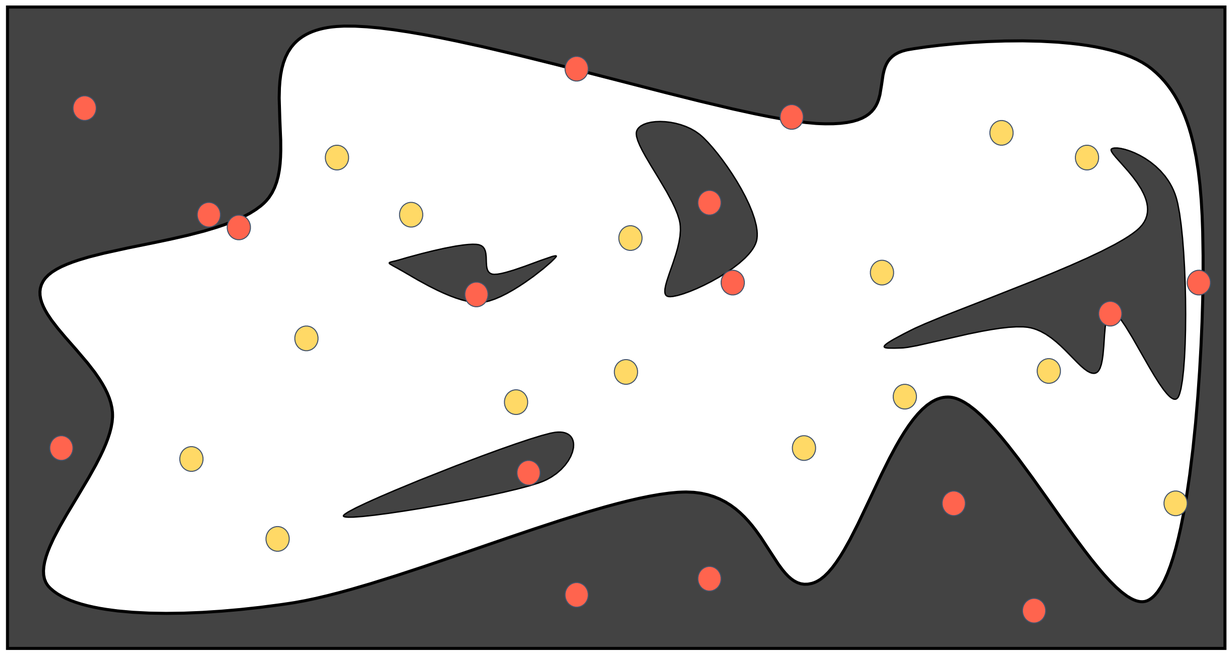
Probabilistic Roadmaps

- Configurations are sampled by picking coordinates at random



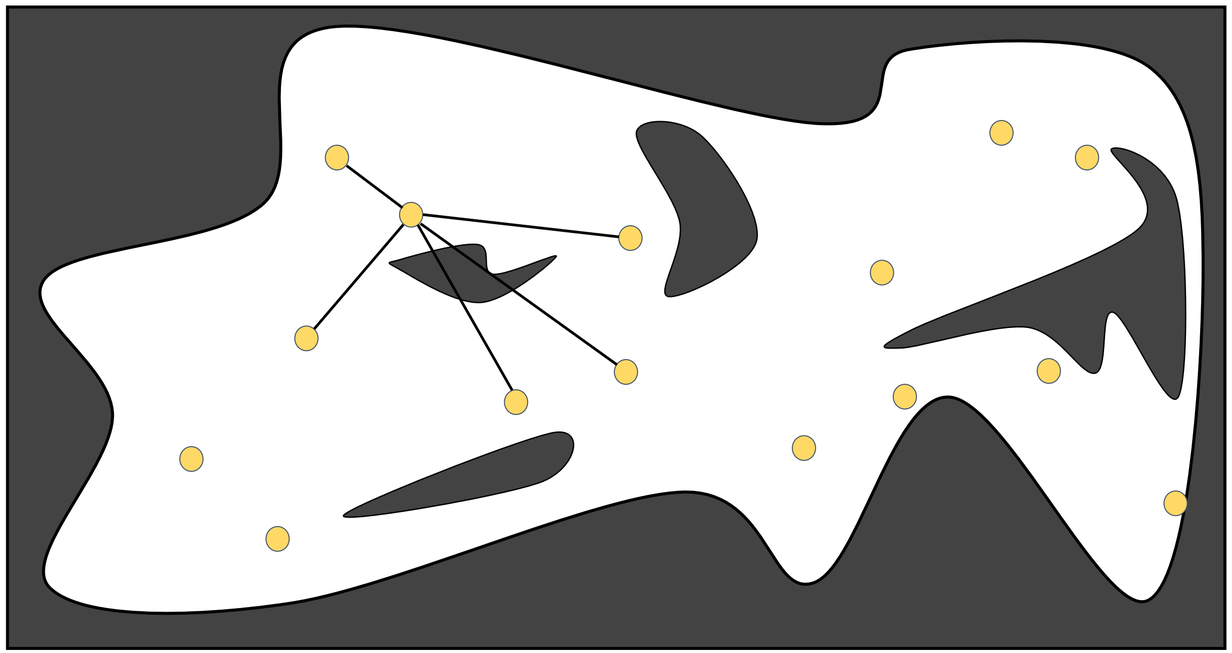
Probabilistic Roadmaps

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



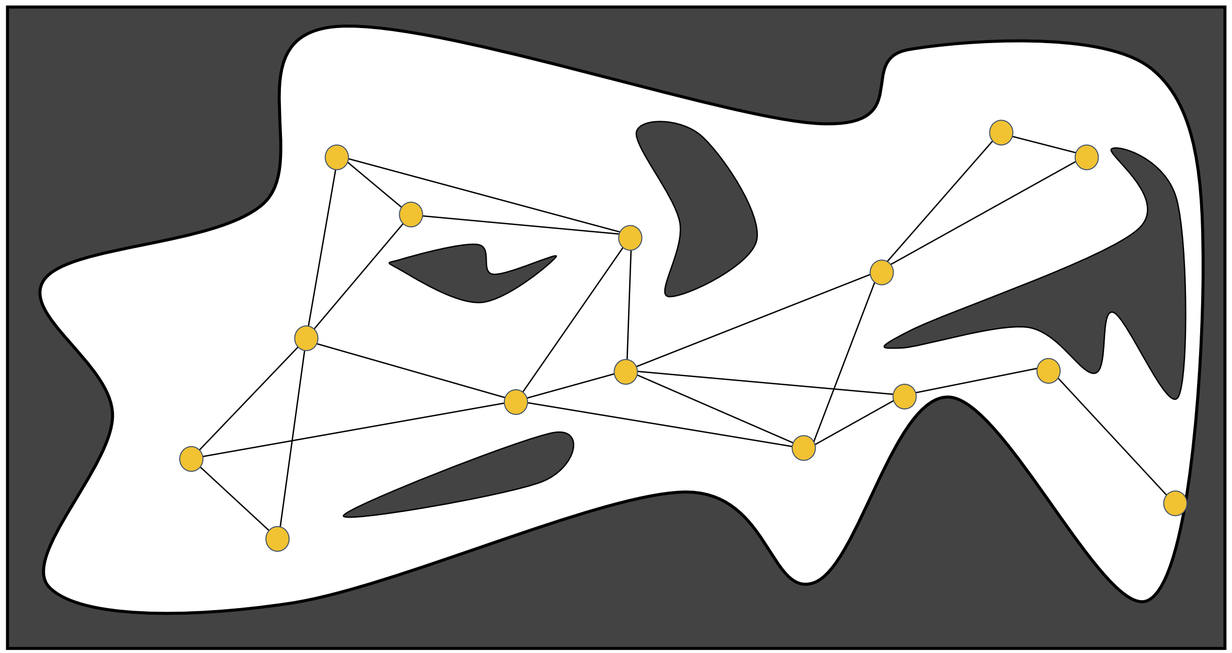
Probabilistic Roadmaps

- 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



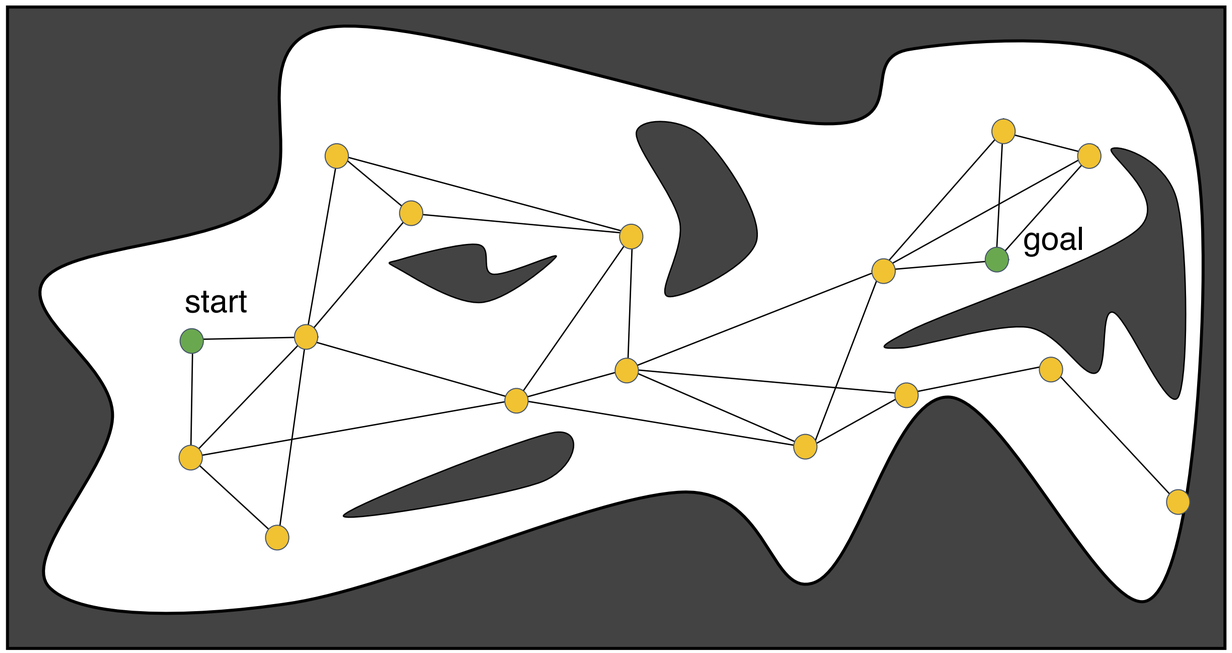
Probabilistic Roadmaps

- 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



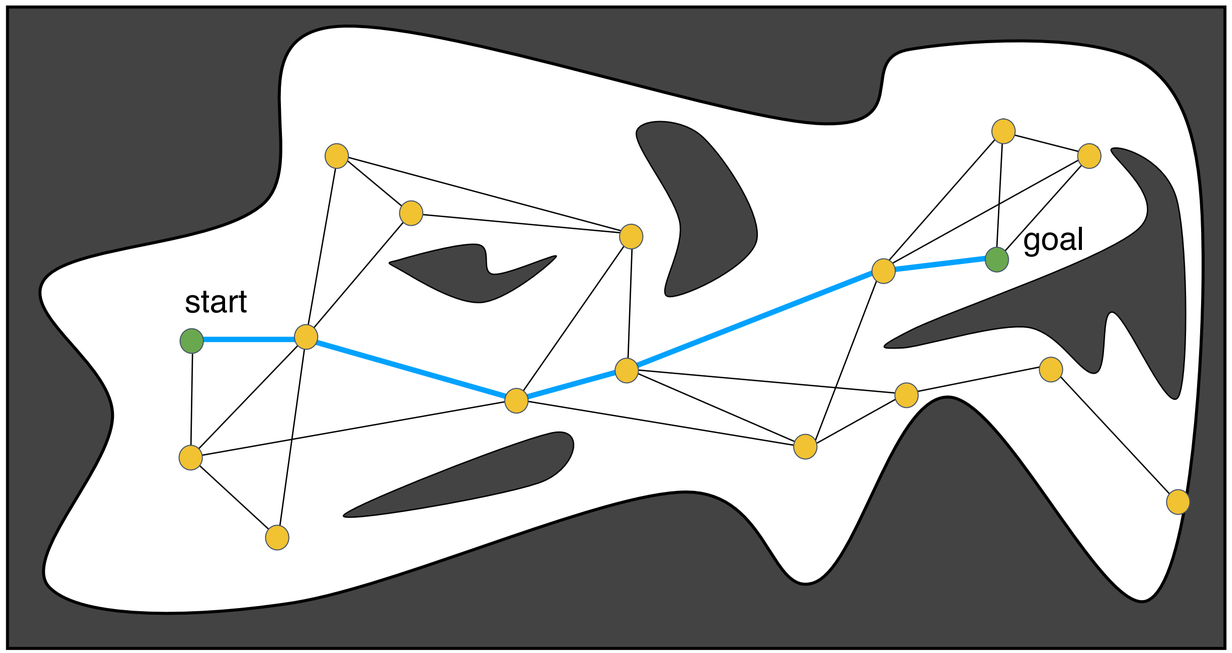
Probabilistic Roadmaps

- 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

Constructing the graph

- Initially empty Graph G
- A configuration q is randomly chosen
- If $q \in Q_{free}$ then add to G
- Repeat until N vertices chosen
- For each q, select k closest neighbors
- Local planner, Δ, connects q to neighbor q'
- If connection is collision free, add edge (q, q')

Algorithm 6 Roadmap Construction Algorithm

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

```
1:  V ← ∅
2:  E ← ∅
3:  while |V| < n do
4:     repeat
5:        q ← a random configuration in Q
6:     until q is collision-free
7:     V ← V ∪ {q}
8:  end while
9:  for all q ∈ V do
10:    N_q ← the k closest neighbors of q chosen from V according to dist
11:    for all q' ∈ N_q do
12:       if (q, q') ∉ E and Δ(q, q') ≠ NIL then
13:          E ← E ∪ {(q, q')}
14:       end if
15:    end for
16: end for
```

Probabilistic Roadmaps

Finding the Path

- Connect q_{init} and q_{goal} to the roadmap
- Find k closest neighbors of q_{init} and q_{goal} in roadmap, plan local path Δ
- Compute cost of path
- Repeat until graphs are connected
- Choose cheapest path

Algorithm 7 Solve Query Algorithm

Input:

q_{init}: the initial configuration
q_{goal}: the goal configuration
k: the number of closest neighbors to examine for each configuration
$G = (V, E)$: the roadmap computed by algorithm 6

Output:

A path from q_{init} to q_{goal} or failure

$N_{q_{\text{init}}} \leftarrow$ the k closest neighbors of q_{init} from V according to *dist*
$N_{q_{\text{goal}}} \leftarrow$ the k closest neighbors of q_{goal} from V according to *dist*
$V \leftarrow \{q_{\text{init}}\} \cup \{q_{\text{goal}}\} \cup V$
set q' to be the closest neighbor of q_{init} in $N_{q_{\text{init}}}$
repeat
  if $\Delta(q_{\text{init}}, q') \neq$ `NIL` **then**
    $E \leftarrow (q_{\text{init}}, q') \cup E$
  else
    set q' to be the next closest neighbor of q_{init} in $N_{q_{\text{init}}}$
  end if
until a connection was succesful or the set $N_{q_{\text{init}}}$ is empty
set q' to be the closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
repeat
  if $\Delta(q_{\text{goal}}, q') \neq$ `NIL` **then**
    $E \leftarrow (q_{\text{goal}}, q') \cup E$
  else
    set q' to be the next closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
  end if
until a connection was succesful or the set $N_{q_{\text{goal}}}$ is empty
$P \leftarrow$ shortest path$(q_{\text{init}}, q_{\text{goal}}, G)$
if P is not empty **then**
  return P
else
  return `failure`
end if

Probabilistic Roadmaps

Finding the Path

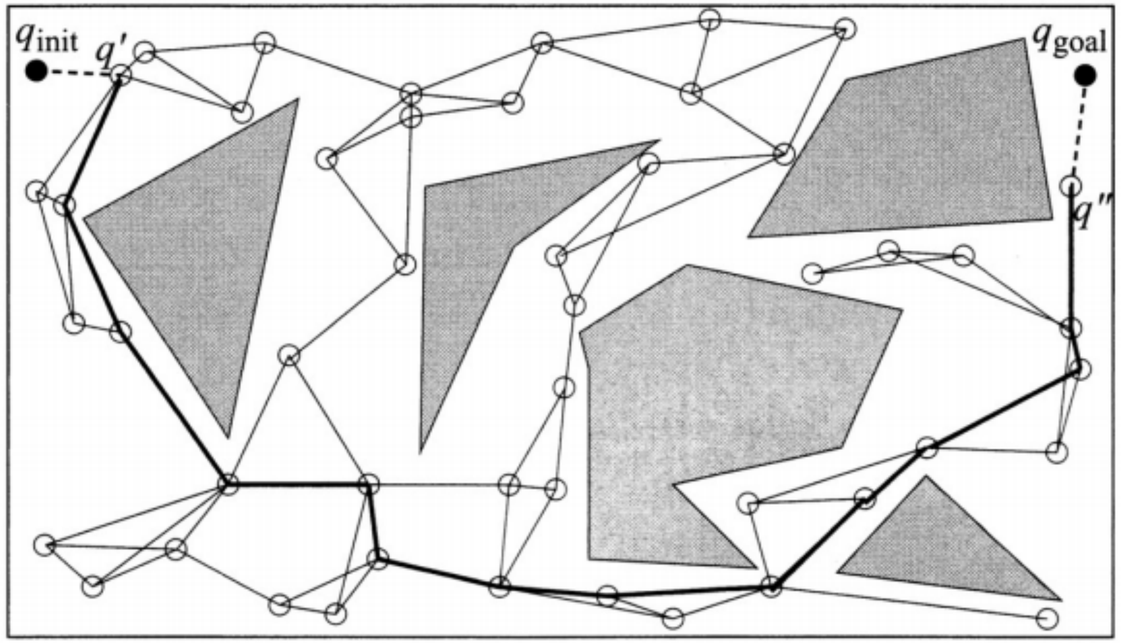


Figure 7.4 An example of how to solve a query with the roadmap from figure 7.3. The configurations q_{init} and q_{goal} are first connected to the roadmap through q' and q''. Then a graph-search algorithm returns the shortest path denoted by the thick black lines.

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

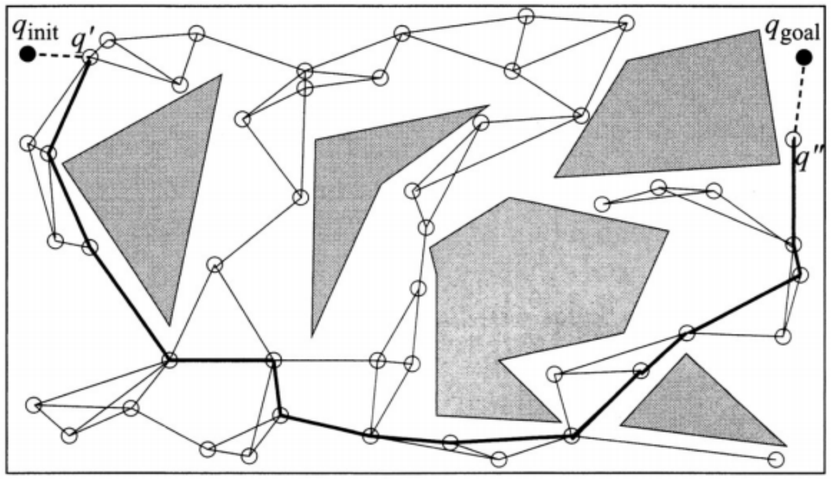


Figure 7.4 An example of how to solve a query with the roadmap from figure 7.3. The configurations q_{init} and q_{goal} are first connected to the roadmap through q' and q''. Then a graph-search algorithm returns the shortest path denoted by the thick black lines.

Probabilistic Roadmaps

Robot Motion Planning on a Chip, Murray et al. RSS 2016

- PRM on an FPGA
- Collision detection circuits on each edge in logic gates for massive parallel operation
- 6DOF planning in <1ms

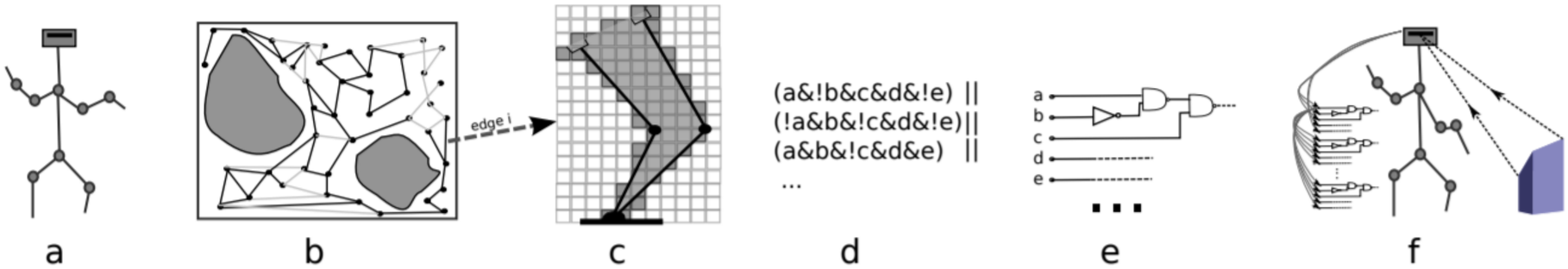
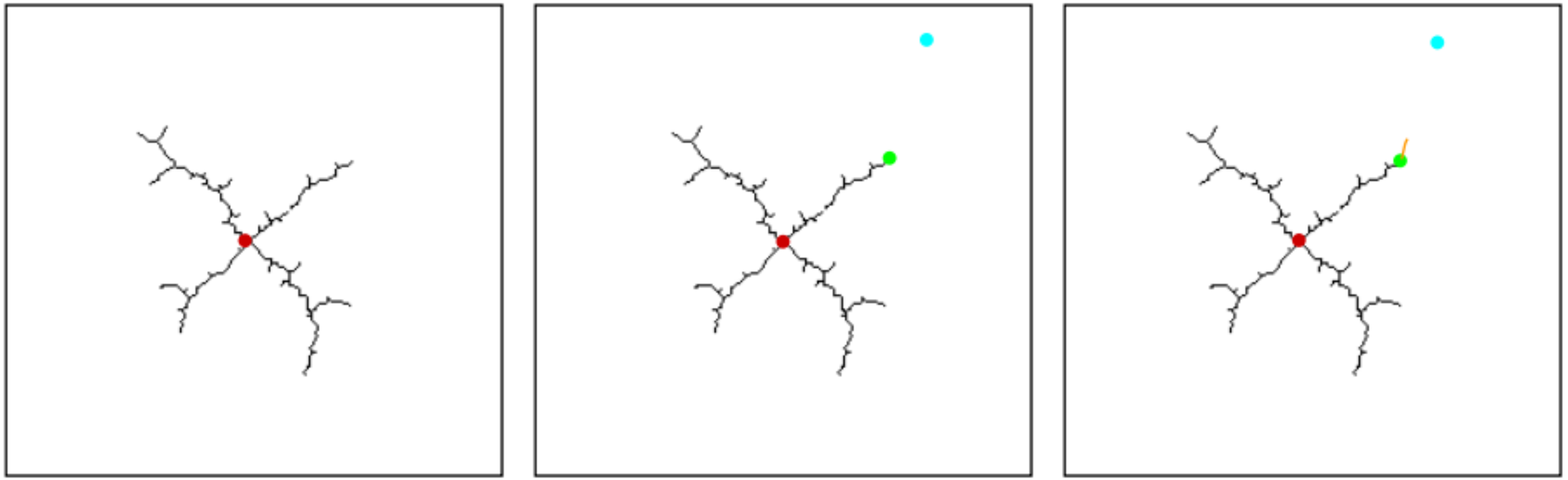


Fig. 3: Our process for producing robot-specific motion planning circuitry. Given a robot description (a), we construct a PRM (b), most likely subsampled for coverage from a much larger PRM. We discretize the robot's reachable space into depth pixels and, for each edge i on the PRM, precompute all the depth pixels that collide with the corresponding swept volume (c). We use these values to construct a logical expression that, given the coordinates of a depth pixel encoded in binary, returns `true` if that depth pixel collides with edge i (d); this logical expression is optimized and used to build a collision detection circuit (CDC) (e). For each edge in the PRM there is one such circuit. When the robot wishes to construct a motion plan, it perceives its environment, determines which depth pixels correspond to obstacles, and transmits their binary representations to every CDC (f). All CDCs perform collision detection *simultaneously, in parallel* for each depth pixel, storing a bit which indicates

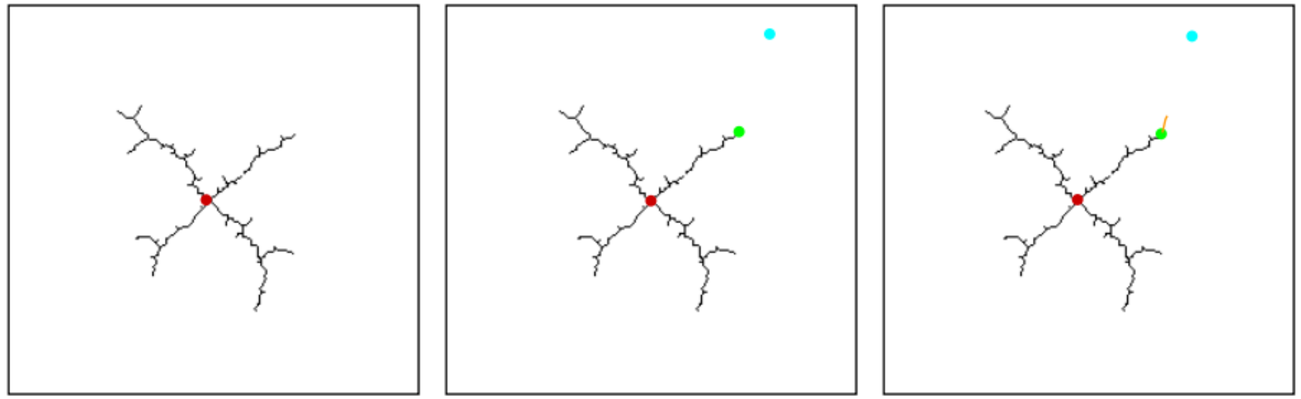
Rapidly Exploring Random Trees (RRT)



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

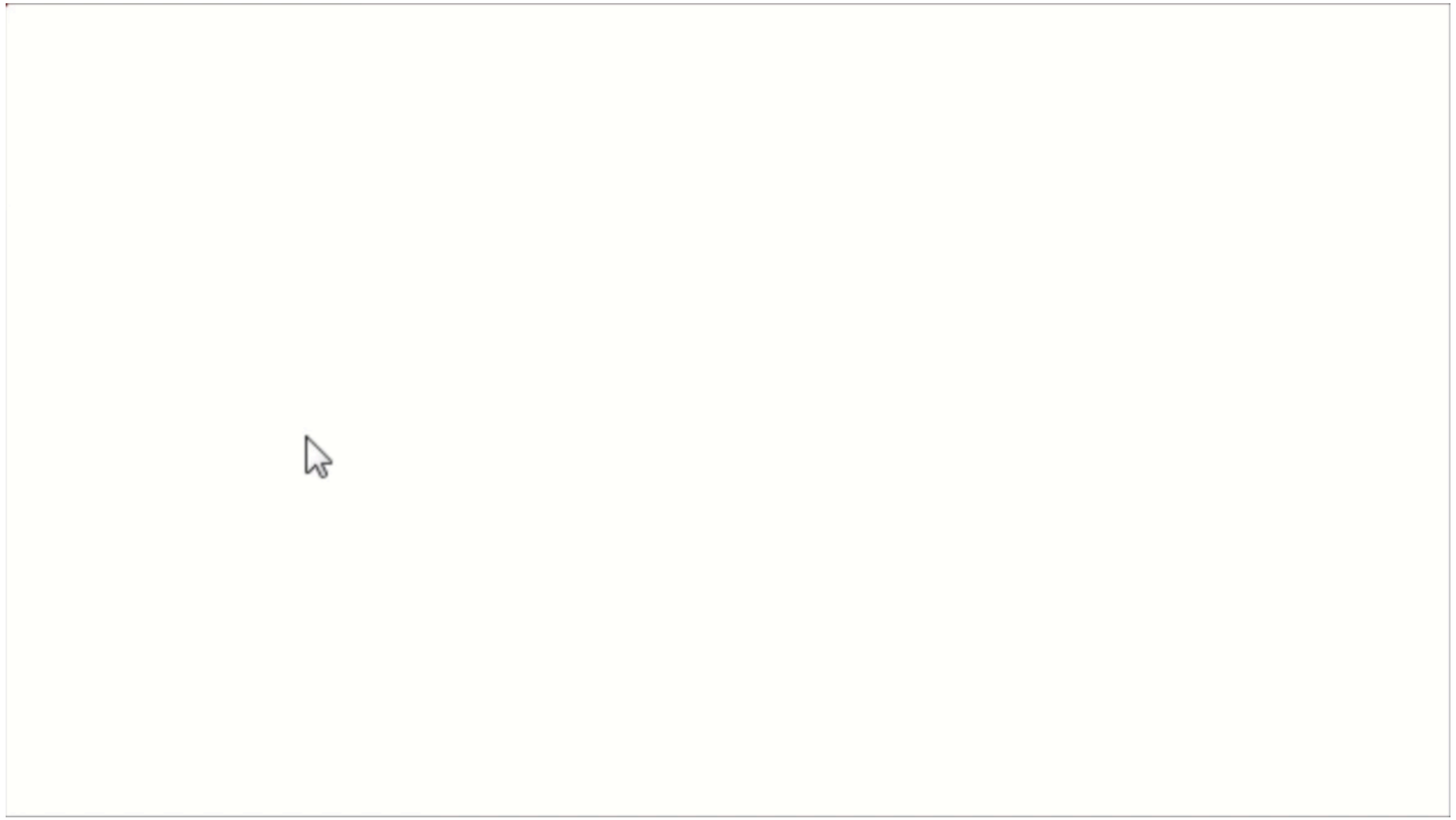
Rapidly Exploring Random Trees (RRT)

```
1. Algorithm BuildRRT
2.   Input: Initial configuration q_init, number of vertices K,
     incremental distance Δq)
3.   Output: RRT graph G
4.   G.init(q_init)
5.   for k = 1 to K
6.     qrand ← RAND_CONF()
7.     qnear ← NEAREST_VERTEX(qrand, G)
8.     qnew ← NEW_CONF(qnear, qrand, Δq)
9.     G.add_vertex(qnew)
10.    G.add_edge(qnear, qnew)
11. return G
```



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!