ROB 456: Intelligent Robots

Week 7, Lecture 1
Path Planning

Oregon State University

Today

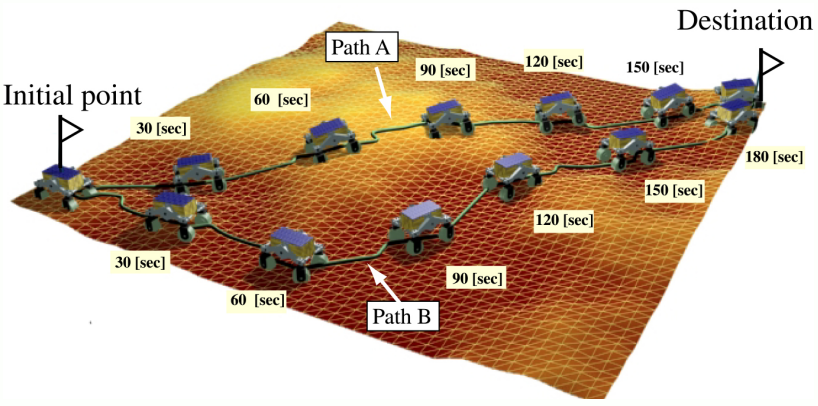
- Motion planning
- Graph search methods
- Sampling-based methods
- Planning in the real world

Oregon State University

MOTION PLANNING

Oregon State University

What is Path Planning?



Oregon State University

Planning

- How to get from the current robot state to the desired robot state
- Possible when robot possesses a **representation** of the world state and can measure the **utility** of its actions
- Challenging because the space of possible plans grows exponentially with the plan duration, dimension, etc.

Motion Planning

- Given start point S, end point G:
  - Find controls to take robot from S to G.
- What are challenges:
  - Obstacles?
  - Dynamics?
  - Feasibility?
- Typically cast as an optimization problem

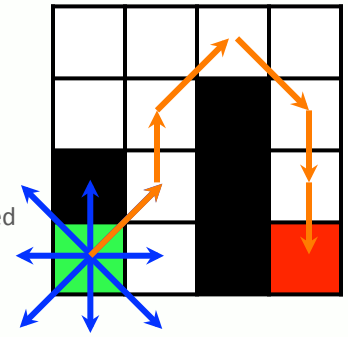
What is Path Planning?

- Path planning: detailing a motion task into discrete motions
  - Robot must get from starting location to defined waypoint
  - Robot must not run into walls, fall down stairs, etc.
  - Ideally, path taken is most efficient (shortest) path
- Inputs to algorithm:
  - Description of task (starting and ending locations)
  - Constraints (robot's physical limitations, environmental obstacles, etc.)
  - Uncertainty about robot and environment
- Outputs of algorithm:
  - Physical path
    - Set of waypoints
    - Speed and turning commands sent to robot's wheels at each time step

GRAPH SEARCH METHODS

Example 1: Discretizing the World

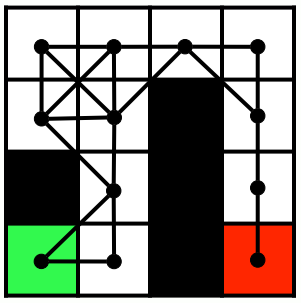
- Move from green cell to red cell
- 8-connected grid motion
- Shortest path?
- What if robot motion was restricted to 4-connected grid motion?



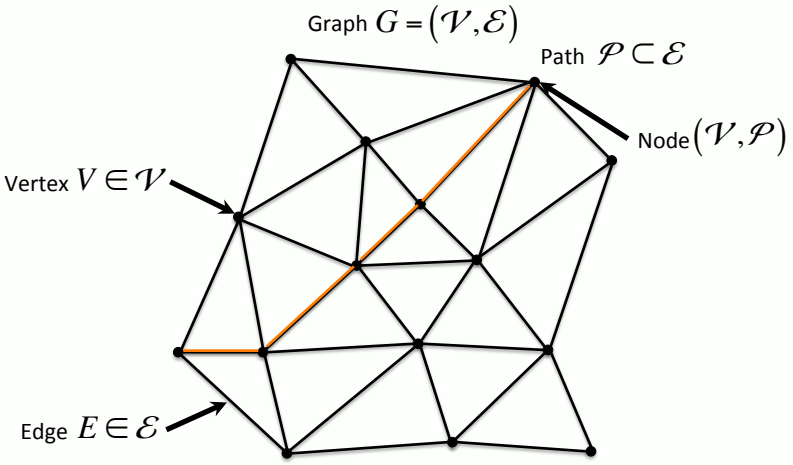
Assumptions about robot motion

Example 1: Discretizing the World

- Search over the underlying graph
- Solve for paths from any point to any other point
- Assume all edge transitions are dynamically feasible



Some Terminology



Oregon State University

Graph Search

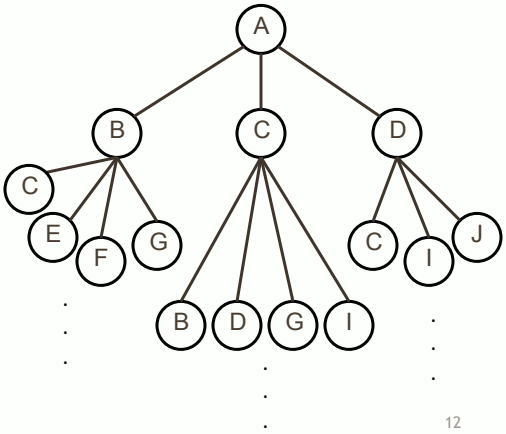
- Need to keep track of path costs
- Expand only the necessary nodes

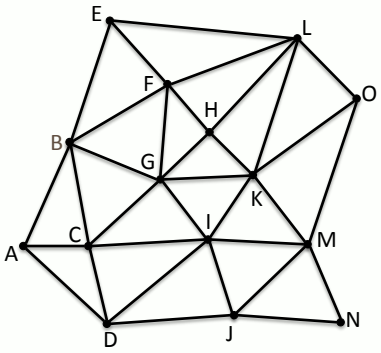
- Graph search methods
  - Breadth-first search (BFS)
  - Depth-first search (DFS)
  - Dijkstra's Algorithm
  - A*

Oregon State University

Search Trees

- We construct a "tree" through which we can search for optimal paths through the environment

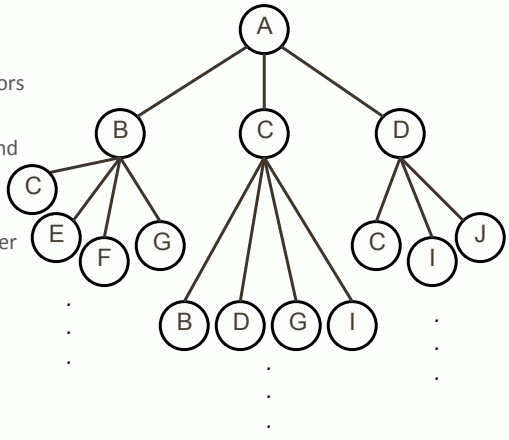




Oregon State University

Search Trees

- A search tree:
  - Start state at the root node
  - Children correspond to successors
  - Nodes contain states, correspond to PLANS to those states
  - For most problems, we can never actually build the whole tree
  - Use search algorithms to efficiently traverse tree



Oregon State University

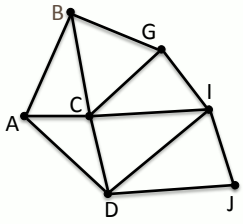
Basics of Search

- Expand out possible plans
- Try to expand as few tree nodes as possible

- **Priority queue** maintains a frontier of unexpanded plans
  - **Open** set keeps track of what nodes to expand next

- **Closed** set keeps track of nodes that have been expanded

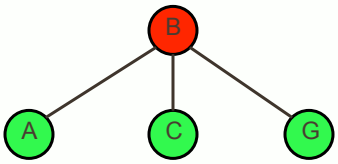
Breadth-First Search Example

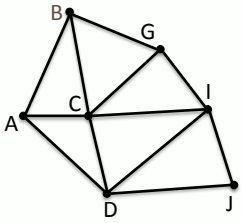


Opened: {B}

Closed: {}

Breadth-First Search Example



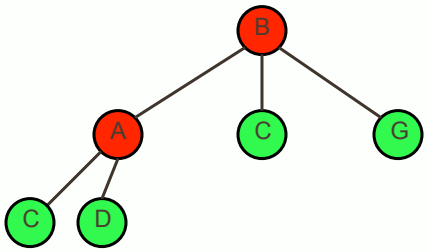


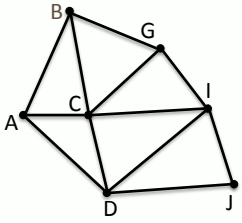
Opened: {A,C,G}

Closed: {B}

Oregon State University

Breadth-First Search Example



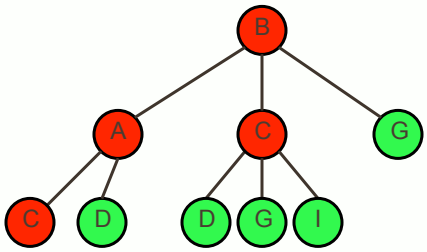


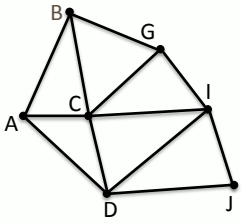
Opened: {C,G,D}

Closed: {B,A}

Oregon State University

Breadth-First Search Example



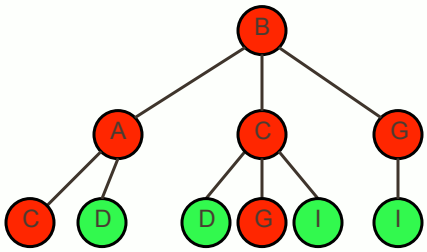


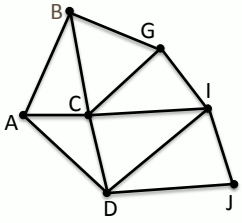
Opened: {G,D,I}

Closed: {B,A,C}

Oregon State University

Breadth-First Search Example



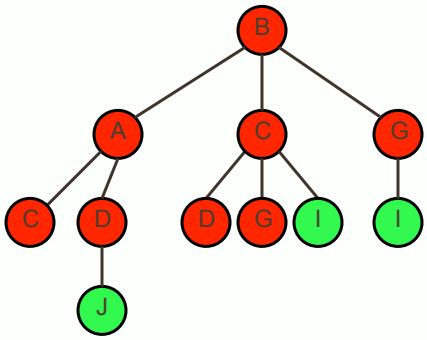


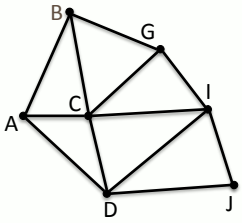
Opened: {D,I}

Closed: {B,A,C,G}

Oregon State University

Breadth-First Search Example



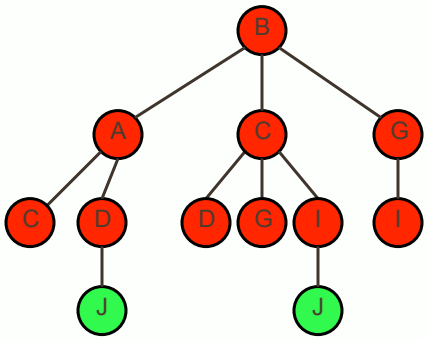


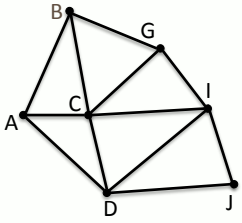
Opened: {I,J}

Closed: {B,A,C,G,D}

Oregon State University

Breadth-First Search Example



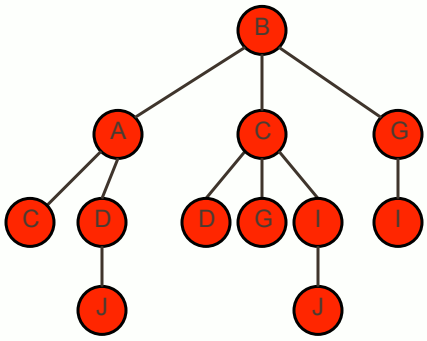


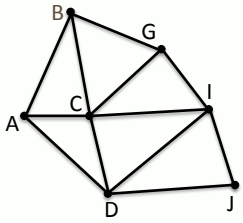
Opened: {J}

Closed: {B,A,C,G,D,I}

Oregon State University

Breadth-First Search Example





Opened:
{}

Closed:
{B,A,C,G,D,I,J}

Final path solution: B→A→D→J

Other solutions exist but also have the same number of transitions

Oregon State University

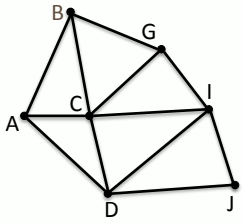
Breadth-First Search

- BFS is guaranteed to find the optimal path through the search tree
  - First solution found is the optimal path
- Unfortunately, BFS is computationally inefficient
- Time complexity of O(V+E)
  - Worst case is that each node is visited: computationally expensive!
- Consider another approach: Depth-first search

Oregon State University

Depth-First Search Example (Iterative Deepening)

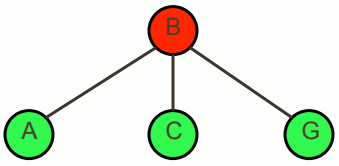
(Keep track of the least cost path to vertex)

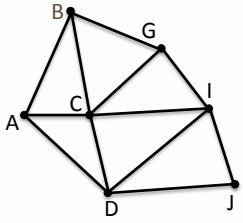


Opened: {B}

Closed: {}

Depth-First Search Example (Iterative Deepening)

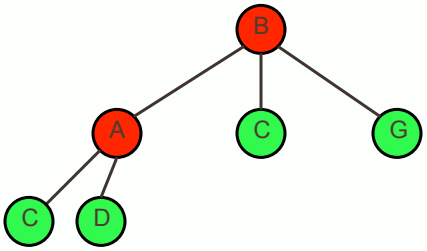


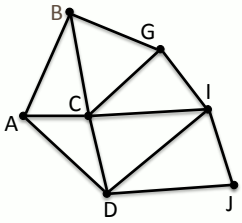


Opened: {A,C,G}

Closed: {B}

Depth-First Search Example (Iterative Deepening)



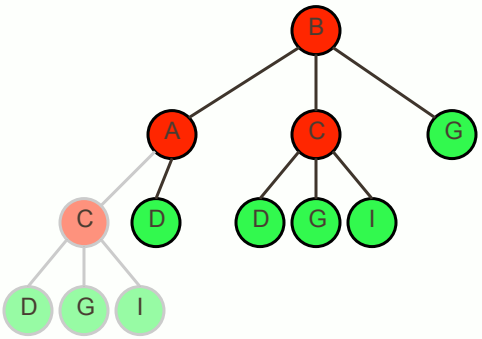


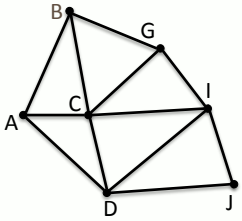
Opened: {C,G,D}

Closed: {B,A}

Oregon State University

Depth-First Search Example (Iterative Deepening)



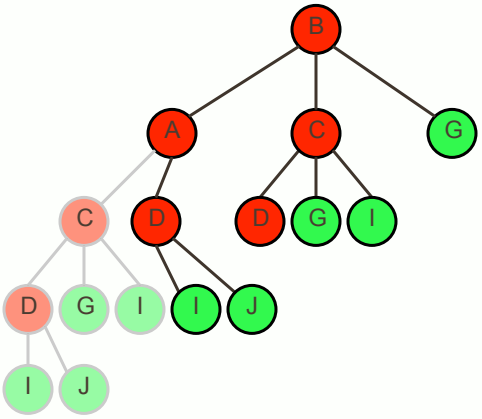


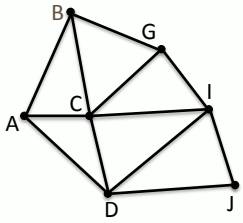
Opened: {G,D,I}

Closed: {B,A,C}

Oregon State University

Depth-First Search Example (Iterative Deepening)



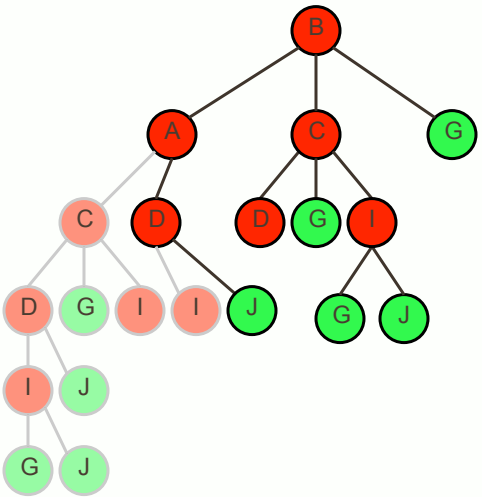


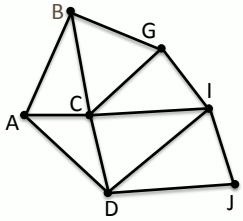
Opened: {G,I,J}

Closed: {B,A,C,D}

Oregon State University

Depth-First Search Example (Iterative Deepening)



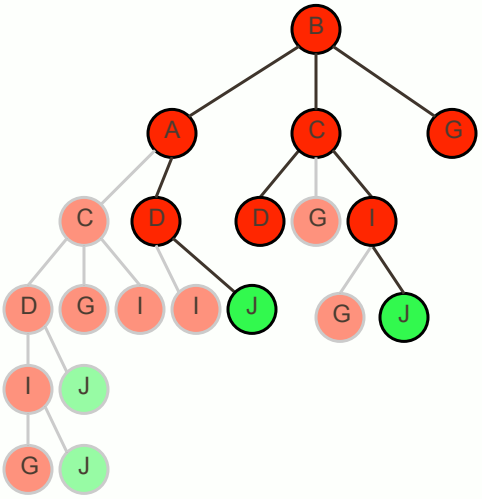


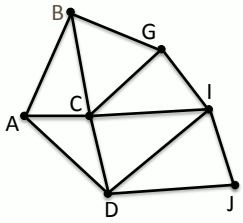
Opened: {G,J}

Closed: {B,A,C,D,I}

Oregon State University

Depth-First Search Example (Iterative Deepening)



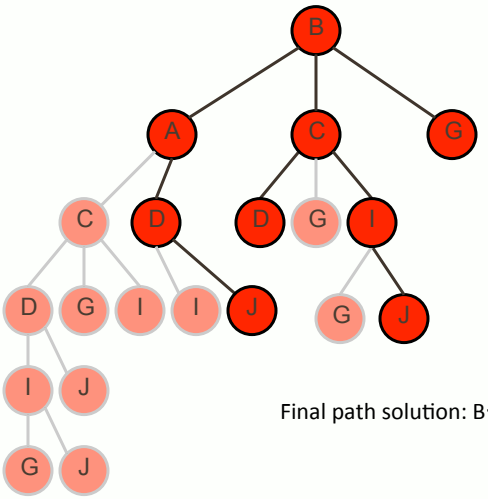


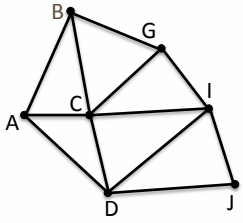
Opened: {J}

Closed: {B,A,C,D,I,G}

Oregon State University

Depth-First Search Example (Iterative Deepening)





Opened: {}

Closed: {B,A,C,D,I,G,J}

Final path solution: B→A→D→J

Oregon State University

Depth-First Search

- Depth First Search (DFS) is a search algorithm where the search starts at the root node and explores as far as possible along each branch before backtracking
- Like BFS, DFS is guaranteed to find the optimal path in the tree
- Problem with DFS: graph is often too large to visit in its entirety, and DFS may not terminate when path lengths are infinite (looping through grid)
  - Therefore we typically only search to limited depth

BFS vs. DFS

- Time bounds of DFS and BFS are equivalent: in the worst case, each algorithm searches the entire tree
- Space bounds of DFS are better than BFS: DFS takes less memory than BFS, since it isn't necessary to store all children at each level of the tree search
- Which algorithm is "better" is more dependent on the domain than computational complexity of the algorithms:
  - If solution is higher up the tree (closer to the root node), then BFS will perform better
  - If solution is farther down the tree (farther from the root node), then DFS will perform better
- Both BFS and DFS are simple to implement, however are computationally inefficient. More complex algorithms are faster, but more difficult to implement

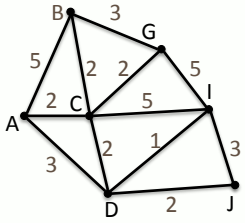
Costs on Actions

- Both BFS and DFS find the shortest path in terms of number of transitions.
  - What about costs?
- Dijkstra's Algorithm and A* search

Dijkstra's Algorithm

- Developed by Edsger Dijkstra in 1959
- Breadth-first search with edge costs
- One of the most commonly used routing algorithms in graph traversal problems
- Asymptotically the fastest known single-source shortest path algorithm for arbitrary directed graphs
- **Open** queue is ordered according to cost to arrive

Dijkstra's Algorithm Example



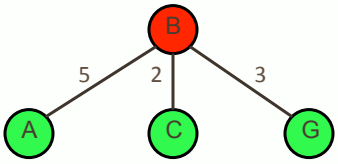
Open: Queue: {B(0),

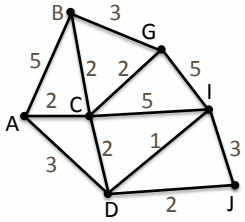
Unvisited: A(∞), C(∞), D(∞), G(∞), I(∞), J(∞)}

Closed: {}

Oregon State University

Dijkstra's Algorithm Example





Open: Queue: {C(2), G(3), A(5),

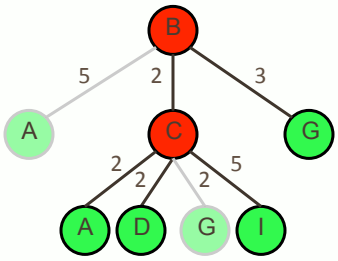
Unvisited: D(∞), I(∞), J(∞)}

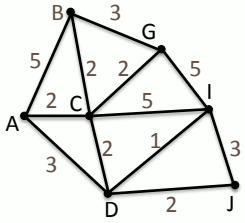
Closed: {B(0)}

Ordered according to cost to arrive

Oregon State University

Dijkstra's Algorithm Example





Queue: {G(3), A(4), D(4), I(7), J(∞)}

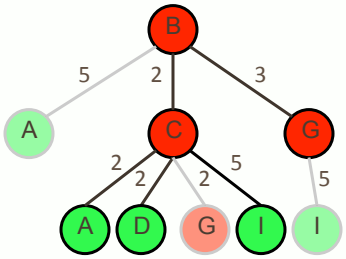
Open: {G(3), A(4), D(4), I(7),

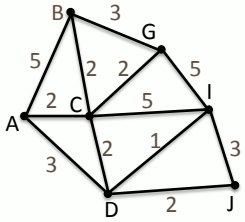
Unvisited: J(∞)}

Closed: {B(0), C(2)}

Oregon State University

Dijkstra's Algorithm Example





Queue: {A(4), D(4), I(7), J(∞)}

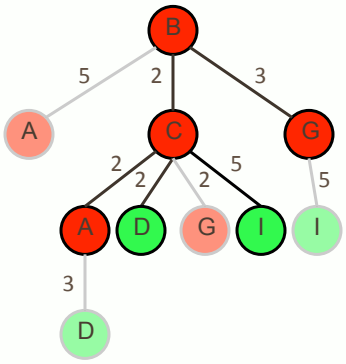
Open: {A(4), D(4), I(7),

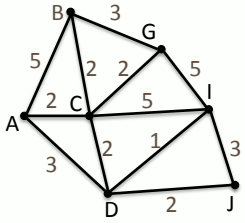
Unvisited: J(∞)}

Closed: {B(0), C(2), G(3)}

Oregon State University

Dijkstra's Algorithm Example



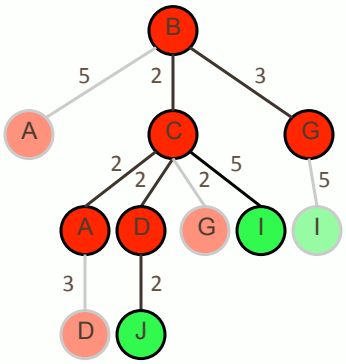


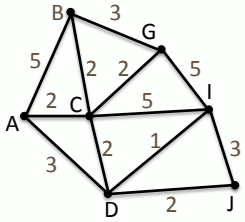
Queue:
Open: {D(4), I(7),
Unvisited: J(∞)}

Closed:
{B(0), C(2), G(3), A(4)}

Oregon State University

Dijkstra's Algorithm Example



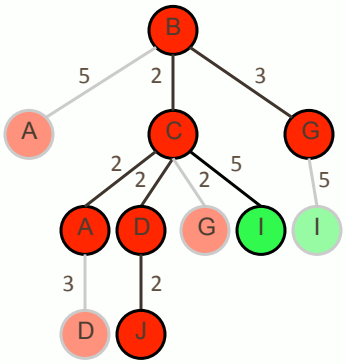


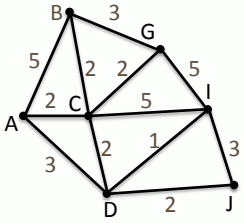
Queue:
Open: {J(6), I(7)}

Closed:
{B(0), C(2), G(3), A(4), D(4)}

Oregon State University

Dijkstra's Algorithm Example





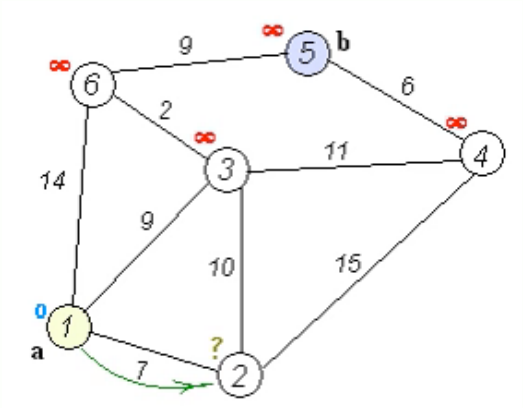
Queue:
{I(7)}

Closed:
{B(0),
C(2),
G(3),
A(4),
D(4),
J(6)}

Final path solution: B→C→D→J
with path cost 6

Oregon State University

Dijkstra's Algorithm Overview



Courtesy wikipedia (http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)

Oregon State University

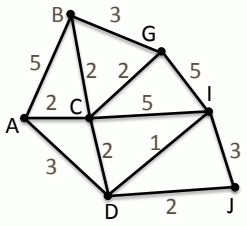
A* Heuristic Search

- Heuristic:
  - Any *estimate* of how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance

- A* : Cost is f(n) = g(n) + h(n)

Cost to arrive Heuristic cost to goal

A* Example

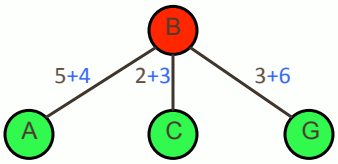


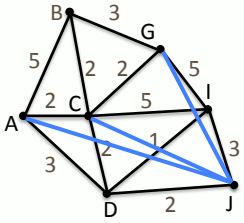
Open: Queue: {B(0),

Unvisited: A(∞), C(∞), D(∞), G(∞), I(∞), J(∞)}

Closed: {}

A* Example

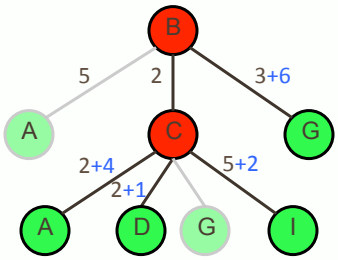


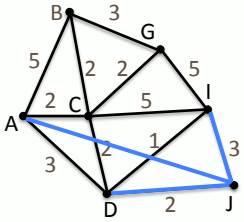


Queue: Closed:
Open: {C(2+3), {B(0)}
G(3+6),
A(5+4),
Unvisited: D(∞),
I(∞),
J(∞)}

Ordered according to A* cost

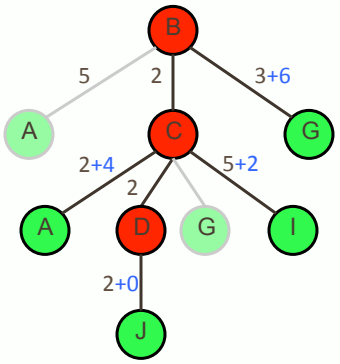
A* Example

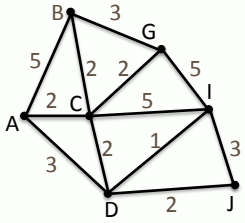




Queue: Closed:
Open: {D(4+1), {B(0),
A(4+4), C(2)}
G(3+6),
I(7+2),
Unvisited: J(∞)}

A* Example



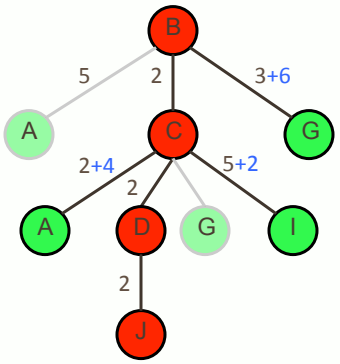


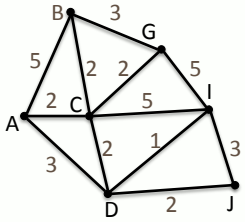
Open: Queue: {J(6+0, A(4+4), G(3+6), I(7+2)}

Closed: {B(0), C(2), D(4)}

Oregon State University

A* Example





Open: Queue: {A(4+4), G(3+6), I(7+2)}

Closed: {B(0), C(2), D(4), J(6)}

Final path solution: B→C→D→J
with path cost 6

Oregon State University

A* Heuristic

- The heuristic must be ***admissible***
  - It never overestimates the cost

$$h(x) \leq d(x, goal)$$

True cost to goal

- The heuristic must be ***consistent***
  - For any pair of adjacent nodes x and y, where d(x,y) is the cost of edge between them

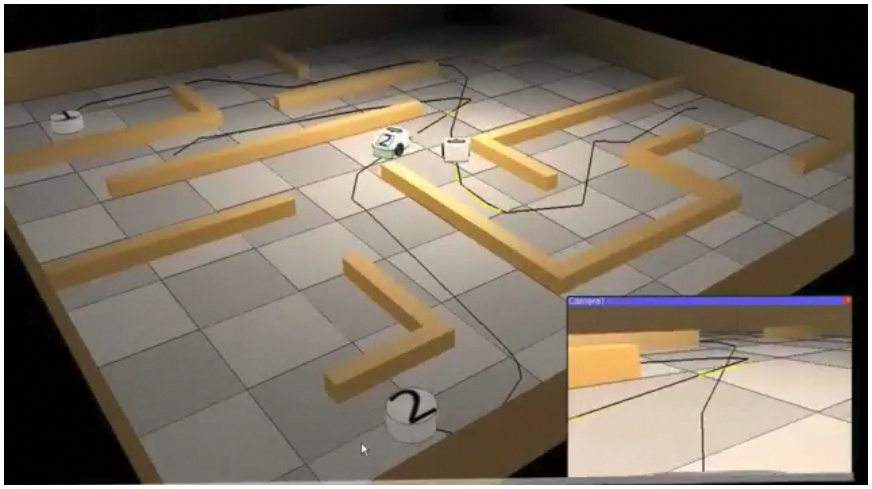
$$h(x) \leq d(x, y) + h(y)$$

- Typical valid heuristics:
  - Euclidean distance
  - Manhattan distance
  - Zero (Dijkstra's algorithm)

A* Search Algorithm

- A* is an extension of Dijkstra's algorithm, and achieves faster performance by using heuristics
- Uses best-first search: A* traverses a graph following a path of lowest expected total cost or distance
- Uses a knowledge plus heuristic cost function to determine the order in which the search visits nodes in the tree.
- The cost function is a sum of two functions:
  - Past path-cost function, which is known cost from the starting node to the current node
  - Future path-cost function, which is a "heuristic estimate" of the distance from the current node to the goal

Planning Example



Oregon State University

Overview of Planning So Far

- We have explained how to discretize world into a grid, and then use search algorithms to find shortest paths from starting point to endpoint in that grid.
- We covered BFS, DFS, Dijkstra's algorithm, and A* search

- Assumptions:
  - World is static
  - Paths on graphs are dynamically feasible
  - Robot has control actions that allow it to follow paths exactly

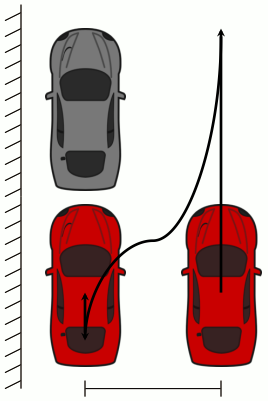
Oregon State University

SAMPLING-BASED PLANNING

Oregon State University

Planning Space

- Reverse parallel parking example:

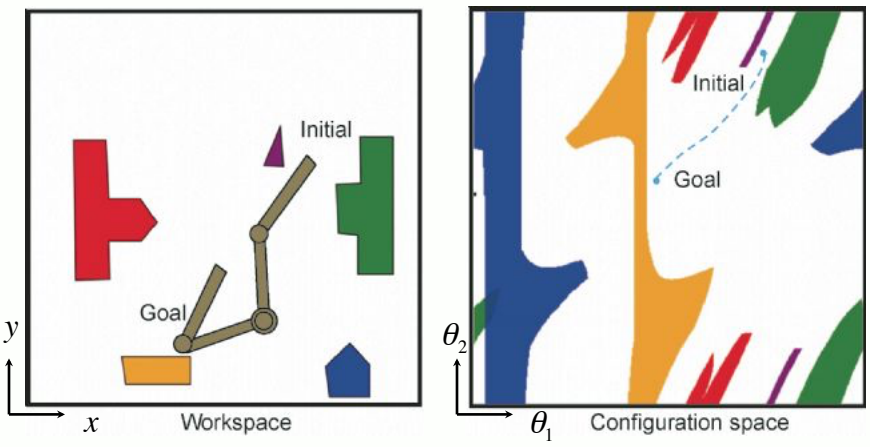


$$d_{workspace} = \|\mathbf{x}_1 - \mathbf{x}_0\|$$

1. Drive straight forward
2. Reverse and turn steering wheel to left
3. Reverse and go straight
4. Reverse and turn steering wheel to right
5. Drive straight forward

Oregon State University

Planning Space: Robot Arm Example



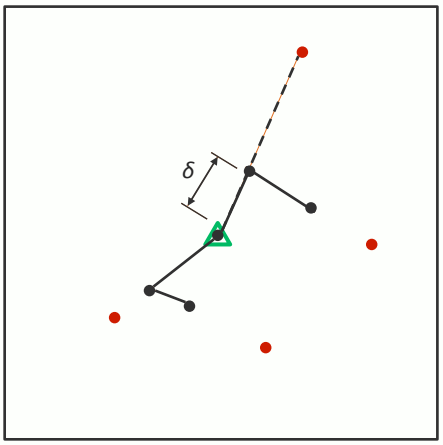
Oregon State University

Sampling-Based Planning

- Sample available configurations and control actions
- Create a tree or graph of valid transitions
- Find the best path that gets you closest to the goal
- Methods:
  - Rapidly-exploring random trees (RRTs)
  - Probabilistic roadmaps (PRMs)

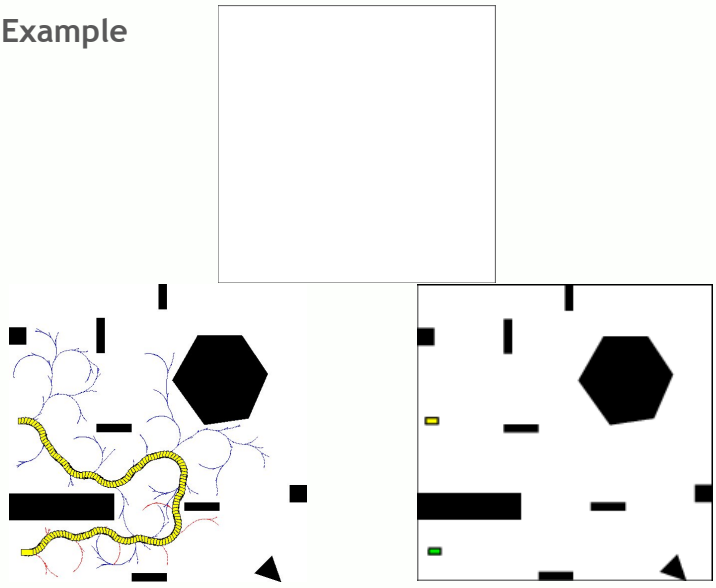
Oregon State University

Rapidly-Exploring Random Trees (RRTs)



Oregon State University

RRT Example

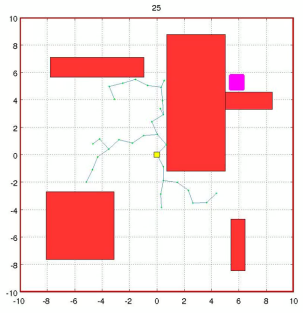


Steve LaValle (http://msl.cs.uiuc.edu/rrt/gallery.html)

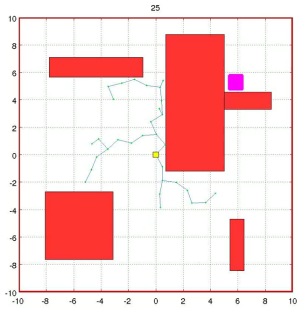
Oregon State University

RRT* Comparison

RRT



RRT*



https://www.youtube.com/watch?v=FAFw8DoKvik

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

Oregon State University

RRT Summary

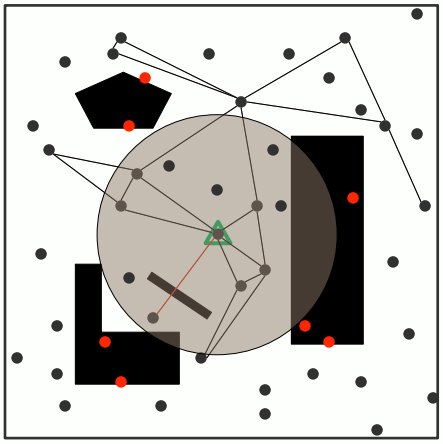
- Paths are dynamically feasible
- RRT* is ***probabilistically complete***
  - As the number of points you sample goes to infinity, you will find a path from start to goal
- RRT* is ***asymptotically optimal***
  - As the number of points you sample goes to infinity, your path solution converges to the optimal path
- Single query path planning

Oregon State University

Probabilistic Roadmaps (PRMs)

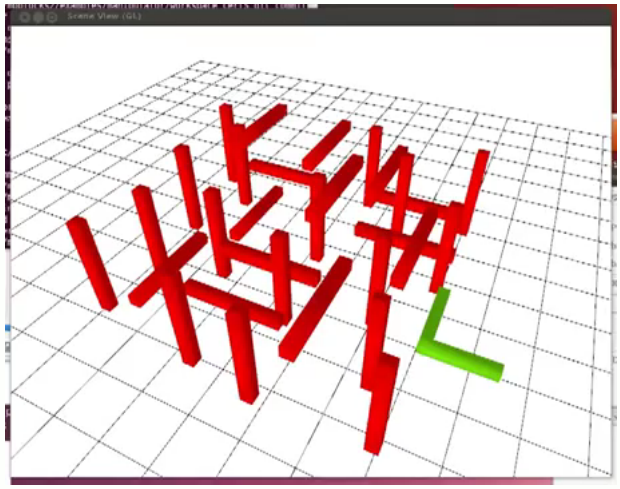
- Step 1: Roadmap construction
- Step 2: Graph search
  - Pick your favourite
- PRM*
  - Connection radius as function of number of vertices

Configuration space



Oregon State University

PRM* Example: Piano Mover's Problem



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

Oregon State University

PRM Summary

- Paths are feasible in configuration space, therefore feasible in workspace
- PRM* is ***probabilistically complete***
  - As the number of points you sample goes to infinity, you will find a path from start to goal
- PRM* is ***asymptotically optimal***
  - As the number of points you sample goes to infinity, your path solution converges to the optimal path
- Multi-query path planning

Oregon State University

PLANNING IN THE REAL WORLD

Oregon State University

Difficulties in Planning

- Dynamic environments
- Uncertainty

When the environment changes or we don't know everything about the robot/environment, planned paths may need to be altered

Uncertainty always complicates robotics!

Let's look at these problems in detail

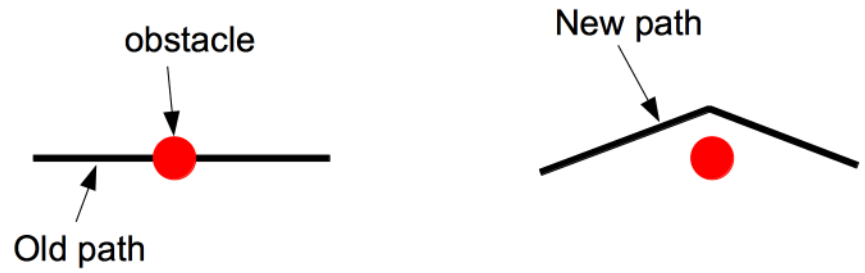
Oregon State University

Planning in Non-Static Maps

- What if the environment isn't static?
  - People (obstacles) moving around in the world
  - Doors opening and closing (paths which used to exist may no longer exist, and vice versa)
  - In disaster exploration missions, parts of buildings may collapse, closing off previously existing paths
- How do we account for environment changing after we have already found an optimal path?
- Two main ways to deal with dynamic maps:
  - Keep original plan, and deviate from plan as little as possible
  - Replan a new path based on new environmental conditions

Oregon State University

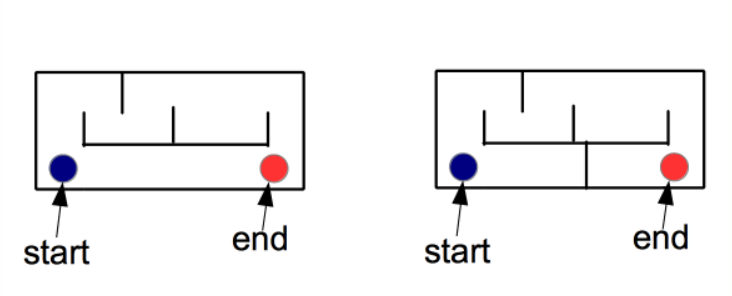
Keep Original Plan with Small Deviation



Oregon State University

Keep Original Plan with Small Deviation

- This works fine if a person gets in the way and the robot simply needs to move around them
- What if a door is closed, or part of a hallway collapses?
  - The "small deviation" in the path is no longer well defined



Oregon State University

Replan a New Map

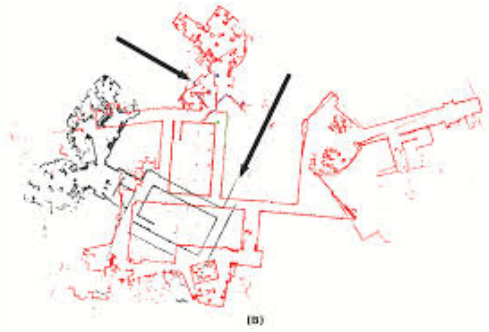
- If the environment changes, construct a new search tree and re-run the search algorithm to find the new best path based on changing environmental conditions
- What are the problems with this approach?
  - Search algorithms are slow, often difficult to compute in real-time
  - If we have to replan many times, this could slow down the mission substantially
- Path replanning algorithms exist, such as D* (dynamic A*), to deal with these types of problems

Which to Use?

- So, do we deviate slightly from paths or plan new paths?
  - There is no canonical answer to this question.
  - It depends on nature of how the environment changes
- As a general rule of thumb:
  - If dynamics are minor and we can get away with small deviation, do it
  - If major change in environment occurs and it isn't clear what a "small deviation" to path is, then replan
- Dynamic environments are very difficult to deal with
  - If environment is static, then we can easily find and execute a solution
  - If environment is dynamic, then we often have to adjust solution during execution to deal with new environmental conditions

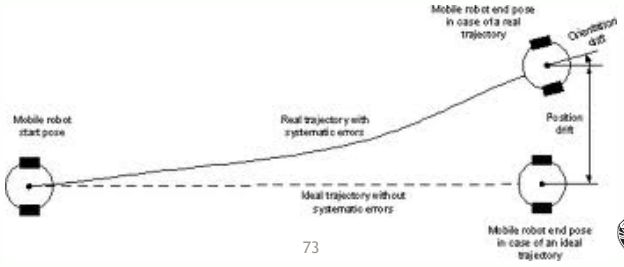
Planning with Uncertainty

- There are many types of uncertainty in robotics
  - Motion uncertainty
  - Missing information about environment
- How do these different types of uncertainty affect planning?



Motion Uncertainty

- A path planning algorithm gives a series of commands for a robot to move from an initial to ending point
- What if a command calls for a forward motion of 10 cm, but results in a forward motion of 9.7 cm?
- What if a command calls for a turn of 1 radian, but results in a turn of 1.2 radians?
- These errors will propagate over time, resulting in a large deviation from the planned course



Motion Uncertainty

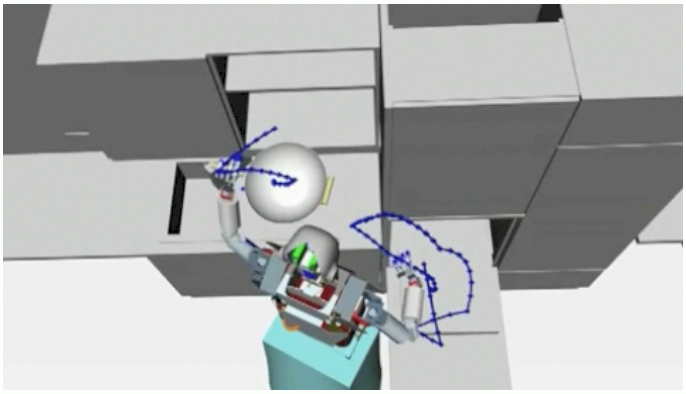
- We need to perform localization during execution of the planned path
- When the robot deviates from planned path, we need to adjust commands sent to wheels to get back to desired path
- Key idea: the planned path works perfectly in simulation, but in hardware, we need to ensure motion uncertainty doesn't cause us to deviate from this path
  - Real robots never work exactly like their simulated counterparts!

Missing Information About Environment

- Very commonly, we do not have all the information about an environment
  - Consider the case where we send a robot into a building after an earthquake to assess damage, damage may cause building map to be incorrect
  - What if we send a robot into a previously unknown (or only partially known) environment, such as Mars?
- We can plan paths based on the limited/incorrect knowledge we have, but the plan will probably need to be modified during execution
  - As more information about the environment is discovered, we will need to replan a path
- Missing and incorrect information has similar effects when compared to dynamic environments. An unexpected aspect of the environment is discovered, and the path is changed to account for this new information

Other Applications

- Path planning is not limited to determining how a mobile robot will move across an environment…



Oregon State University

PR2 Arm Planning Example

Oregon State University

PR2 Arm Planning Example

Oregon State University

A Note on Computational Complexity

- We have covered robot localization (state estimation), grid mapping, SLAM, and path planning
- All of these algorithms are computationally expensive
- Computational complexity grows with size and dimensionality of the problem

Oregon State University

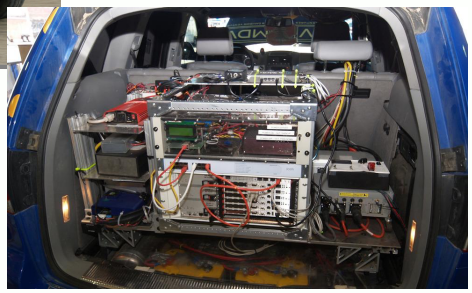
Overall Computational Complexity

- We never run one of these algorithms by itself
- If we are path planning, we also need localization to figure out where the robot is (to account for actuation noise), and possibly a mapping algorithm if the map isn't static
- If we have a camera on the robot, we need to run an image processing algorithm, which typically consider each pixel in the image as an input
- This requires a MASSIVE amount of computation

Robots in the Real World

The more capabilities…

…the more computers



A Note on Computational Complexity

- The computationally complexity of problems in robotics are not intuitive, but you need to keep it in mind when designing algorithms
- If you can get away with ignoring information, do it!
  - In mapping and localization, we generally don't consider every possible grid cell
- Use the simplest algorithm that works!
  - Roomba navigation algorithm is very low level:
    - Follow walls
    - Zig zag randomly through rooms
    - Spiral around dirty areas

Hierarchical Planning

- Basic idea: Create high level plans first, then provide detail as needed:
  - Incrementally creates a plan by refining more abstract plan steps, expanding them into more detailed subplans.
  - Exploits knowledge captured in the form of a library of subplans: rather than constructing a plan directly as a search through primitive executable actions, retrieve and combine subplans that have been prebuilt to achieve typical (sub)goals.
  - The planning process is complete when the plan is refined down to the level of executable actions.

Summary

- Path planning: given an environment, a starting location, and an ending location, find the optimal path for the robot to take
  - Graph search methods
  - Sampling-based methods
- Process:
  - Enumerate all possible plans in a tree
  - Search through the tree to find the best path from initial to final location
- When it isn't possible to follow previously planned path:
  - Take a slight deviation around obstacle and return to planned path, or
  - Plan a new path
- Difficulties:
  - Dynamic environments, motion uncertainty
  - Missing/incorrect information about environment
- Keep computational complexity in mind when designing algorithms

Next Time

- Final project code setup
- Reinforcement Learning