Planning Methods Review

Lecture 9: Probabilistic vs Deterministic Planning

Autonomous Mobile Robots

Fall 2025

Today's Objective

- Review main planning methods we've covered
- Understand how planning evolved from industrial robotics
- See how different approaches solve the same problem
- Connect planning to the full robotic system architecture

The Fundamental Division

Core Distinction

How they construct the configuration space for search

Probabilistic Planning

- Random sampling of configuration space
- Example: RRT (Rapidly-exploring Random Trees)
- No explicit graph construction
- Explores through randomness

Deterministic Planning

- Systematic graph construction
- Then search the graph
- Explicit discretization
- Methodical exploration

The Planning Problem

Mathematical Formulation

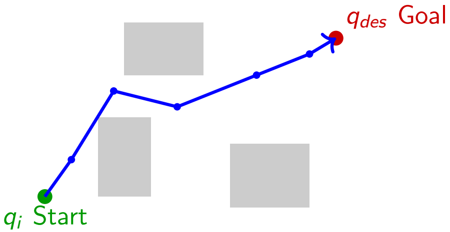
Given:

- Starting configuration: $q_0 = q_i$ (initial)
- Goal configuration: $q_n = q_{des}$ (desired)

Find: A sequence of points

$$Q = \{q_0, q_1, q_2, \ldots, q_n\}$$

That connects the robot from start to goal without collisions

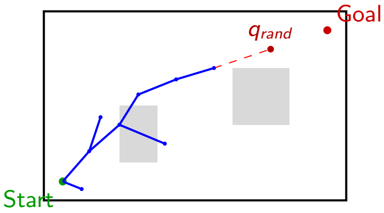


Probabilistic Planning: RRT Review

Rapidly-exploring Random Trees (RRT)

Mechanism:

1. Start from initial point
2. Generate random nodes
3. Connect to nearest node
4. Grow toward unexplored regions



Key Feature

Probabilistic sampling - no explicit map needed

Probabilistic Completeness

Theoretical Guarantee

PROBABILISTIC COMPLETENESS:
If time goes to infinity, the algorithm will converge to a solution

The Promise

- Path will eventually be found
- Probability $\rightarrow$ 1.0 as samples $\rightarrow \infty$
- No explicit discretization

The Issue

"There is no time limited"

- Convergence in infinite time
- No finite time guarantee
- Unpredictable solution time

Comparison

"Deterministic methods can at least guarantee convergence in a limited time"

When to Use Probabilistic Methods?

Advantages

- High-dimensional spaces
- 6+ DOF robot arms
- No explicit free-space map
- Proven completeness

Disadvantages

- Convergence time $\rightarrow \infty$
- No finite time guarantee
- Non-repeatable paths
- Random variance

Application Context

"Based on the situations and the problem you are solving, there might be good enough methods to use for your specific problem. You don't need to go look into more complex methods such as RRT."

Deterministic Planning Framework

General Deterministic Planning - Two Steps

STEP 1: CONSTRUCT A GRAPH

- Create nodes (V) - discrete configuration samples
- Create edges (E) - valid connections between nodes
- Discretize the continuous configuration space

STEP 2: GRAPH SEARCH

- Find optimal path from start to goal
- Use standard graph search algorithms
- Guarantee bounded time solution

Key Advantage

"Can guarantee we will converge in a limited time"

Why Deterministic for Mobile Robots?

Dimensional Advantage

Mobile robots: 2D/3D navigation (3-6 DOF)
Manipulators: 6-20+ DOF

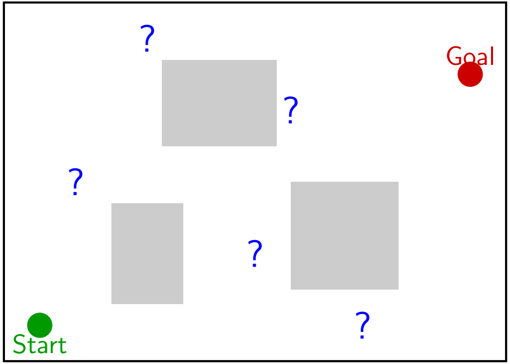
Engineering Trade-off

- Lower dimensions $\rightarrow$ deterministic feasible
- More straightforward than RRT
- Repeatable, optimal solutions

Graph Construction Challenge

The Question

"What places do you recommend to put nodes?"



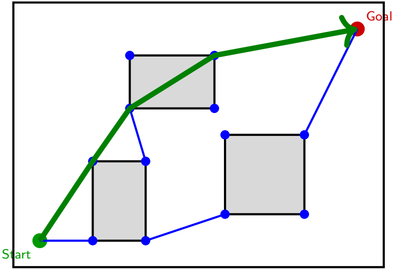
Where should we place nodes?

Method 1: Visibility Graphs

Construction Approach

Place nodes at obstacle vertices (corners)

- Connect nodes if line of sight is clear
- No edge through obstacles
- Include start and goal



Result

Paths along obstacle boundaries - optimal length

Visibility Graphs: The "Touching the Wall" Problem

Critical Flaw

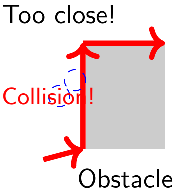
"Only the nodes that can be connected with no collision"

BUT: The optimal path grazes very close to obstacle edges

Why This Matters

- Real robots have uncertainty
- Sensor noise
- Controller imperfection
- Localization errors

"Touching wall" = collision risk



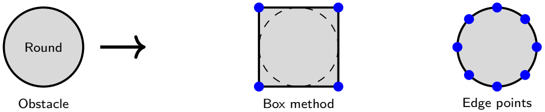
Solution

Inflate obstacles by robot radius (add buffer)

Visibility Graphs: Round Obstacle Challenge

Engineering Problem

"You have to come up with some engineering solution..."



Challenge

"Finding exact points on circle edges is computationally challenging"

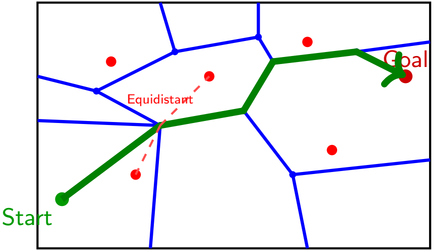
Method 2: Voronoi Diagrams

Voronoi Approach

Nodes in the middle of free space between obstacles

Key Property:

- Points equidistant from multiple obstacles
- Creates "roads" through free space
- Maximizes clearance from walls



Voronoi edges partition space into cells

Voronoi Diagrams: Safety vs Optimality

ADVANTAGE: Safety

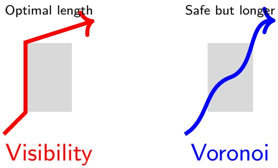
"Good advantage is uncertainty in obstacle locations and not going sharply from edges"

- Maximizes clearance
- Handles localization uncertainty
- Safer paths

DISADVANTAGE: Path Length

"The main issue is the optimality"

- Longer paths
- Takes "scenic route"
- Not near optimal

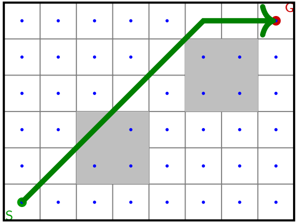


Method 3: Cell Decomposition

Grid-Based Approach

"Excel Cell Decomposition" - Exact Cell Decomposition

- Divide free space into cells
- Cell centers → nodes
- Adjacent cells → edges



Limitation

"Need to have a full map or construction"

The Map Dependency Problem

Industry Challenge

"Having a map is a limitation... I don't think any of them are fully mapless"

Current Approach

Autonomous vehicles:

- High-definition maps required
- Construct maps in advance
- Update maps periodically
- Expensive operation

Industry Examples

- **Wayve** (UK): Claims mapless
- **Waymo**: Still uses HD maps
- **Most startups**: Hybrid approach

The Challenge

- Cost of HD map creation and maintenance
- Computation is still heavy

Graph Search Overview

Once We Have a Graph...

Search for optimal path from Start to Goal

UNINFORMED SEARCH

No knowledge of goal location

- Breadth-First Search (BFS)
- Depth-First Search (DFS)

Explore blindly until goal found

INFORMED SEARCH

Use heuristics to guide search

- Dijkstra's Algorithm
- A* Search
- Weighted A*

Guide exploration toward goal

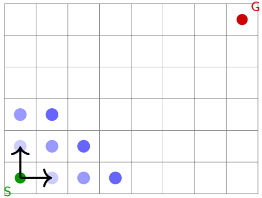
Breadth-First Search (BFS)

Level-by-Level Exploration

Mechanism: Explore all nodes at depth d before $d + 1$
"No Revisit" - mark nodes to prevent cycles

Properties:

- ✓ Complete
- ✓ Optimal (unweighted)
- × Memory intensive



Expands in waves

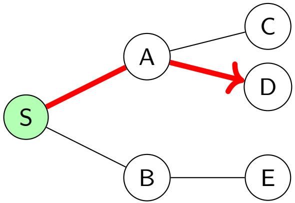
Depth-First Search (DFS)

Deep Exploration

Mechanism: *"Explore one path exhaustively"*

Properties:

- ✓ Memory efficient
- × Not complete (loops)
- × Not optimal

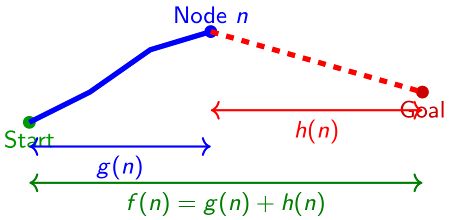


Example: $S \to A \to D$
"Plunge" behavior

Cost Functions for Informed Search

Key Definitions

- $g(n)$ = Cost from Start to node n (actual cost)
- $h(n)$ = Estimated cost from n to Goal (heuristic)
- $f(n)$ = Total cost function



Implementation

Heap (Priority Queue) - *"Sorting your Neighbor"*

Dijkstra's Algorithm

Uniform Cost Search

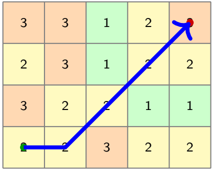
$f(n) = g(n)$ - Expands by accumulated cost

Non-Uniform Costs

Terrain types:

- Paved: 1.0
- Grass: 1.5
- Mud: 2.0

Least resistance path



Different costs

Data Structure

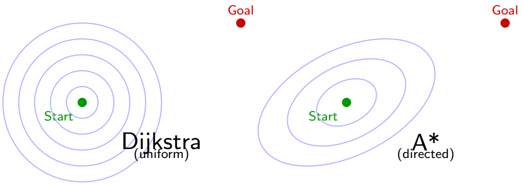
Heap - *"Sorting your Neighbor"*

A* Search: Directed Optimality

Heuristic-Guided Search

$f(n) = g(n) + h(n)$ where $h(n)$ = estimated cost to goal

Heuristics: Euclidean or Manhattan distance



Advantage

Fewer nodes expanded than Dijkstra

Weighted A*: Speed vs Optimality

The Epsilon Factor

Weighted A*: $f(n) = g(n) + \epsilon \cdot h(n)$

where ϵ (epsilon) controls the greediness

Epsilon Behavior

- $\epsilon = 1$: Standard A*

  $\rightarrow$ Optimal
- $\epsilon > 1$: Weighted A*

  $\rightarrow$ "Greedy" search

  $\rightarrow$ Faster but suboptimal
- $\epsilon = 0$: Reduces to Dijkstra

Real-Time Trade-off

"Key technique for real-time planning"

Fast "good enough" path in 10ms

vs

Perfect path in 1 second

For mobile robots, speed often matters more than perfection

Artificial Potential Fields: Continuous Planning

Physics Analogy

Concept: Treat robot as a particle moving in an energy field

Total Potential:

$$U(q) = U_{att}(q) + U_{rep}(q)$$

Control Force (gradient descent):

$$F(q) = -\nabla U(q) = -\begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix}$$

Operating Principle

Robot "rolls downhill" toward goal while avoiding obstacles

Potential Field Components

Attractive Potential

$U_{att}(q)$

Purpose:

- Pulls robot toward goal
- Like spring/magnet
- Increases with distance

$F_{att} = -\nabla U_{att}$
Points toward goal

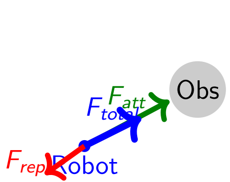
Repulsive Potential

$U_{rep}(q)$

Purpose:

- Pushes away from obstacles
- Creates safety buffer
- Limited range

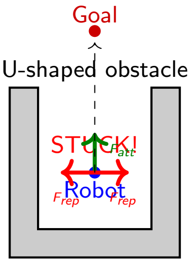
$F_{rep} = -\nabla U_{rep}$
Points away from obstacles



The Local Minima Problem

The Trap

$F_{total} = 0$ before goal - forces balance but robot stuck!



$F_{att} + F_{rep} = 0$
Forces cancel!

Solutions

"Random Walks" - noise to escape; Multiple attempts

Advanced Solution: Stein Variational Gradient Descent

Modern Approach

Stein Variational Gradient Descent (SVGD)

Brief mention in lecture:

- Uses “particle repulsion” mechanism
- Maintains multiple trajectory candidates
- Prevents “mode collapse”
- Finds diverse paths simultaneously

Concept

Instead of one path that gets stuck in local minimum, maintain multiple particles that repel each other - ensuring exploration of different solution modes

Application

Mentioned as topic for advanced study / final projects

From Planning to Control: The $X \to U$ Problem

The Disconnect

Planners output: Positions/configurations (x, y, θ) - State X
Robots need: Control inputs (v, ω) or torques τ - Control U

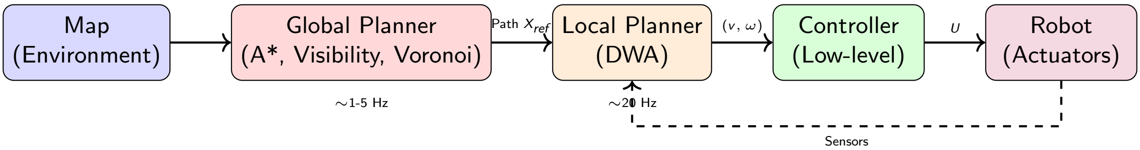
Why The Gap Exists

- **Non-holonomic constraints:** Differential-drive robot can't move sideways
- **Dynamics:** Mass, inertia, friction ($F = ma$)
- **Actuator limits:** Maximum speed, acceleration bounds
- **Real-time requirements:** Must react to dynamic obstacles

Solution

Hierarchical architecture bridging geometric planning and physical control

Hierarchical Planning Architecture



Global Planner

- Low frequency
- Solves the "maze"
- Strategic path
- Ignores dynamics

Local Planner

- High frequency
- Dynamic obstacles
- Feasible velocities
- Example: DWA

Controller

- Highest frequency
- Motor commands
- Physical execution
- Handles dynamics

Dynamic Window Approach (DWA)

Local Planning in Velocity Space

"Image Action" architecture mentioned in notes

Core Concept: Search in velocity space (v, ω) instead of position space

Dynamic Window: Reachable velocities given acceleration limits

- $V \in [V_t - a_{max} \cdot dt,\ V_t + a_{max} \cdot dt]$
- $\Omega \in [\omega_t - \alpha_{max} \cdot dt,\ \omega_t + \alpha_{max} \cdot dt]$

Process

1. Sample velocity pairs (v_i, ω_i) within dynamic window
2. Simulate forward trajectory for each pair
3. Evaluate cost: heading + clearance + velocity
4. Select optimal (v^*, ω^*)
5. Execute for short time, then replan

Historical Context: Industrial Origins

Where These Methods Come From

"A lot of these deterministic methods come from robot manipulators and industry"

Industrial Robot Context

- 4-6+ degrees of freedom arms
- More complex than mobile robots
- **BUT:** Static environments
- Planning happens once
- High efficiency crucial

"Highly efficient compared to probabilistic methods"

Mobile Robot Challenge

"These methods cannot be used for dynamic environment"

- Dynamic obstacles
- Unknown environments
- Need reactive planning
- Real-time constraints

"A lot of modifications happen to apply them for mobile robot use cases"

Method Selection Guide

Method	Optimality	Safety	Best For
Visibility Graph	$\checkmark\checkmark$	$\times$	Static, optimal paths needed
Voronoi Diagram	$\times$	$\checkmark\checkmark$	Uncertainty, safety critical
Grid/Cell Decomp	$\sim$	$\sim$	Simple environments, known maps
RRT (Probabilistic)	$\times$	$\sim$	High DOF, complex spaces

Search Algorithm Trade-offs

- **BFS/DFS:** Simple but inefficient for large spaces
- **Dijkstra:** Optimal but slow (searches all directions)
- **A*:** Optimal and efficient with good heuristic
- **Weighted A* ($\epsilon > 1$):** Fast but suboptimal - real-time preference

Key Takeaways

1. **Probabilistic vs Deterministic**
   - Time guarantees vs dimensional complexity
   - RRT for high-DOF, deterministic for mobile robots
2. **Graph Construction Determines Path Quality**
   - Visibility: Optimal but risky
   - Voronoi: Safe but longer
   - Design choice based on requirements
3. **Search Algorithms: Speed vs Optimality**
   - Weighted A* gives control over trade-off
   - Real-time often prefers "good enough" quickly
4. **Planning $\neq$ Control**
   - Hierarchical architecture bridges the gap
   - Global strategy + local reactivity + physical execution

Industry Challenges Ahead

Current Bottlenecks

- **Map dependency:** High-definition maps expensive to create/maintain
- **Computational load:** Real-time planning still computationally heavy
- **Operational cost:** *"Operation of autonomous cars still very challenging"*

Future Directions

- Truly mapless navigation
- Lighter computational requirements
- Better integration of planning and control
- *"Tons of optimization space in that ecosystem"*

Final Perspective

No Single "Best" Method

"Based on the situation and the problem you are solving..."

The right planner depends on:

- Dimensionality of configuration space
- Static vs dynamic environment
- Optimality requirements
- Computational resources
- Safety criticality
- Real-time constraints

Understanding the trade-offs enables intelligent method selection

Homework

Plotting Potential Fields

- Implement potential fields showing two scenarios:
- One where the robot successfully navigates from A to B,
- One where it fails

Questions?

Next Lecture: Advanced Planning Topics