Motion/Path Planning

- Task:
  find a feasible (and cost-minimal) path/motion from the current configuration of the robot to its goal configuration (or one of its goal configurations)

- Two types of constraints:
  - environmental constraints (e.g., obstacles)
  - dynamics/kinematics constraints of the robot

- Generated motion/path should (objective):
  - be any feasible path
  - minimize cost such as distance, time, energy, risk, ...

Examples (of what is usually referred to as path planning):

Examples (of what is usually referred to as motion planning):

Piano Movers’ problem
Motion/Path Planning

Examples (of what is usually referred to as motion planning)

Planned motion for a 6DOF robot arm

Motion/Path Planning

Path/Motion Planner

Controller

map update

pose update

Motion/Path Planning

Path/Motion Planner

Controller

map update

pose update

Motion/Path Planning

Path/Motion Planner

Controller

map update

pose update

i.e., determinisic registration or Bayesian update
i.e., Bayesian update(EKF)

Motion/Path Planning

Path/Motion Planner

Controller

map update

pose update

i.e., determinisic registration or Bayesian update
i.e., Bayesian update(EKF)

Uncertainty and Planning

- Uncertainty can be in:
  - prior environment (i.e., door is open or closed)
  - execution (i.e., robot may slip)
  - sensing environment (i.e., seems like an obstacle but not sure)
  - pose

- Planning approaches:
  - deterministic planning:
    - assume some (i.e., most likely) environment, execution, pose
    - plan a single least-cost trajectory under this assumption
    - re-plan as new information arrives
  - planning under uncertainty:
    - associate probabilities with some elements or everything
    - plan a policy that dictates what to do for each outcome of sensing/acting
    - minimizes expected cost-to-goal
    - re-plan if unaccounted events happen
Uncertainty and Planning

• Uncertainty can be in:
  - prior environment (i.e., door is open or closed)
  - execution (i.e., robot may slip)
  - sensing environment (i.e., seems like an obstacle but not sure)
  - pose

• Planning approaches:
  - deterministic planning:
    - assume some (i.e., most likely) environment, execution, pose
    - plan a single least-cost trajectory under this assumption
    - re-plan as new information arrives
  - planning under uncertainty:
    - associate probabilities with some elements or everything
    - plan a policy that dictates what to do for each outcome of sensing/acting
    - minimizes expected cost-to-goal
    - re-plan if unaccounted events happen

Example

Urban Challenge Race, CMU team, planning with anytime D*

Outline

• Deterministic planning
  - constructing a graph
  - search with A*
  - search with D*
Outline

- Deterministic planning
  - constructing a graph
  - search with A*
  - search with D*

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - overlay uniform grid over the C-space (discretize)

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path

Planning via Cell Decomposition

- Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path
Planning via Cell Decomposition

• Approximate Cell Decomposition:
  - construct a graph and search it for a least-cost path
  - VERY popular due to its simplicity and representation of arbitrary obstacles

• Graph construction:
  - major problem with paths on the grid:
    - transitions difficult to execute on non-holonomic robots

---

Planning via Cell Decomposition

• Graph construction:
  - lattice graph
  - pros: sparse graph, feasible paths
  - cons: possible incompleteness

---

Planning via Cell Decomposition

• Graph construction:
  - lattice graph
  - each transition is feasible (constructed beforehand)
  - action template
  - replicate it online

---

Planning via Cell Decomposition

• Graph construction:
  - lattice graph
  - outcome state is the center of the corresponding cell
  - action template
  - replicate it online
Outline

• Deterministic planning
  - constructing a graph
  - search with A*
  - search with D*

• Planning under uncertainty
  - Markov Decision Processes (MDP)
  - Partially Observable Decision Processes (POMDP)

A* Search

• Computes optimal g-values for relevant states
  at any point of time:

  ![Diagram of A* Search]

  one popular heuristic function – Euclidean distance

  - Is guaranteed to return an optimal path (in fact, for every expanded state) – optimal in terms of the solution

  - Performs provably minimal number of state expansions required to guarantee optimality – optimal in terms of the computations

  ![Diagram of A* Search](CSE-571-Courtesy-of-Maxim-Likhachev-CMU)
A* Search

- Is guaranteed to return an optimal path (in fact, for every expanded state) – helps with robot deviating off its path if we search with A* backwards (from goal to start)
- Performs provably minimal number of state expansions required to guarantee optimality – optimal in terms of the computations

\[
\begin{align*}
S_1 & : g=0, h=3 \\
S_2 & : g=1, h=2 \\
S_3 & : g=2, h=2 \\
S_4 & : g=2, h=1 \\
S_5 & : g=3, h=1 \\
S_{\text{goal}} & : g=5, h=0
\end{align*}
\]

Effect of the Heuristic Function

- A* Search: expands states in the order of \( f = g + h \) values

For large problems this results in A* quickly running out of memory (memory: \( O(n) \))

Weighted A* Search: expands states in the order of \( f = g + \epsilon h \) values, \( \epsilon > 1 \) = bias towards states that are closer to goal

Solution is always \( \epsilon \)-suboptimal: \( \text{cost(solution) \leq \epsilon \cdot \text{cost(optimal solution)} } \)
Effect of the Heuristic Function

- **Weighted A* Search**: expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

20DOF simulated robotic arm state-space size: over $10^{26}$ states

Effect of the Heuristic Function

- planning in 3D $(x, y)$ for each foothold
- heuristic is Euclidean distance from the center of the body to the goal location
- cost of edges based on kinematic stability of the robot and quality of footholds

Outline

- Deterministic planning
  - constructing a graph
  - search with A*
  - search with D*

Incremental version of A* (D*/D* Lite)

- Robot needs to re-plan whenever
  - new information arrives (partially-known environments or/and dynamic environments)
  - robot deviates off its path

ATRV navigating initially-unknown environment

planning map and path
Incremental version of A* (D*/D* Lite)

- Robot needs to re-plan whenever
  - new information arrives (partially-known environments or/and dynamic environments)
  - robot deviates off its path

Guiding of Incremental Planning (re-planning): reuse of previous planning efforts

Planning in dynamic environments

Motivation for Incremental Version of A*

- Reuse state values from previous searches
  - cost of least-cost paths to \( s_{goal} \) initially
  - cost of least-cost paths to \( s_{goal} \) after the door turns out to be closed

Costs are optimal when search is done backwards:

How to reuse these values from one search to another? Incremental A*
Motivation for Incremental Version of A*

- Reuse state values from previous searches

![Diagram](image1)

Cost of least-cost paths to $s_{goal}$ initially

Cost of least-cost paths to $s_{goal}$ after the door turns out to be closed

Would # of changed $g$-values be very different for forward A*?

Any work needs to be done if robot deviates off its path?

Incremental Version of A*

- Reuse state values from previous searches

*Initial search by backwards A*  *Initial search by D* Lite

*Second search by backwards A*  *Second search by D* Lite

Anytime Aspects
Anytime Aspects

- Deterministic planning
  - constructing a graph
  - search with A*
  - search with D*

- Planning under uncertainty
  - Markov Decision Processes (MDP)
  - Partially Observable Decision Processes (POMDP)

Summary

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>heuristic</th>
<th>states expanded</th>
<th>time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>2,019</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>$h_{2D}$</td>
<td>26,108</td>
<td>1.30</td>
<td></td>
</tr>
<tr>
<td>$h_{fush}$</td>
<td>124,794</td>
<td>3.49</td>
<td></td>
</tr>
</tbody>
</table>

Many useful approximate solvers for MDP/POMDP exist!