Introduction to Multi-Agent Programming

4. Search Algorithms and Path-finding

Robot Motion Planning & Multi-Robot Planning

Alexander Kleiner, Bernhard Nebel
Contents

• Robot Motion Planning
  – Visibility Graphs
  – Grid-based Planning
  – Sampling-based Planning

• Multi-Robot Planning
  – Decoupled Techniques
Illustrations and content presented in this lecture where taken from:

*Artificial Intelligence – A Modern Approach, 2nd Edition*
by Stuart Russell - Peter Norvig

*Planning Algorithms*
By Steven M. LaValle

Available for downloading at: http://planning.cs.uiuc.edu/
Robot Motion Planning

Introduction

A motion computed by a planning algorithm, for a digital actor to reach into a refrigerator.

A planning algorithm computes the motions of 100 digital actors moving across terrain with obstacles.

An application of motion planning to the sealing process in automotive manufacturing.
Several mobile robots attempt to successfully navigate in an indoor environment while avoiding collisions with the walls and each other.

Using mobile robots to move a piano
Robot Motion Planning
Problem Formulation

The configuration space \( C \) is the space containing all possible configurations of the robot.

Suppose world \( W = \mathbb{R}^2 \) or \( W = \mathbb{R}^3 \).

Obstacle region \( O \subset W \)

Rigid robot \( A \subset W \)

Robot configuration \( q \in C \), \( q = (x_t, y_t, \theta) \) for \( W = \mathbb{R}^2 \)

Obstacle region \( C_{obs} \subset C \) is defined by:

\[
C_{obs} = \left\{ q \in C \mid A(q) \cap O \neq \emptyset \right\}
\]

Which is the set of all configurations \( q \) at which \( A(q) \), the transformed robot, intersects \( O \).

The free space is defined by:

\[
C_{free} = C \setminus C_{obs}
\]
Robot Motion Planning
Problem / Solution Concepts

Problem: Find continuous path $\tau : [0, 1] \rightarrow C_{free}$

With $\tau(0) = c_{start}$ and $\tau(1) = c_{goal}$

- Requirements
  - Shortest path
  - Minimal execution time (requiring a good fit with the motion model, least amount of rotations, etc.)
  - Maximal distance to obstacles (needed in dynamic environments, and when sensors are unreliable)

- Many solution concepts:
  1. Potential Fields (more details in a later lecture)
  2. Visibility Graphs
  3. Grid-based Planning
  4. Sampling-based Planning
Robot Motion Planning
Visibility Graphs

- Approximation of obstacles as **polygons**
- Visibility Graph \( S \): Build graph \( S=(V,G) \),
  - where \( V \) is the set of all vertices from the corners of polygon obstacles
  - and \( E \) the set of all visible connections between them
- Planning with **discrete** methods (e.g. A*)
- Simplification at RoboCup Soccer: Every obstacle is considered as a circle!
  \( \rightarrow \) Edges are constructed from circle tangents
- Advantage: Depends **only** on number of obstacles
- Disadvantages: (1) Paths very **close** to obstacles (2) How to get **good** polygons?
Visibility Graphs

Example

Path very close to other robots
Robot Motion Planning
Grid-based Planning

• Planning on a subdivision of $C_{\text{free}}$ into smaller cells
• Simplification: grow borders of obstacles up to the diameter of the robot, e.g., by Gaussian blur
• Construction of graph $G = (V, E)$, where $V$ is the set of cells and $E$ represents their neighbor-relations
• Planning with discrete methods (e.g. A*)
  – Resulting path is a sequence of cells
• Hierarchical planning: find path on coarse resolution and re-plan on more fine grained resolutions
• Disadvantage:
  – Memory usage grows with the size of the environment
  – Fails in narrow passages of $C_{\text{free}}$
• Advantage: No polygons!
Grid-based Planning

Example

Better: path sufficiently far from other!
Robot Motion Planning
Sampling-based Motion Planning

• **Basic Idea:** To avoid explicit construction of $C_{\text{obs}}$
  - Instead: probe $C_{\text{free}}$ with a sampling scheme
  - Builds a graph $G=(V,E)$ by connecting sampled locations
    - each $e \in E$ has to be collision free!
    - on $G$ a solution can be found by discrete search methods (e.g. A*)

• **Critical part:** Random Sampling

• **Time consuming part:** Collision Checks
Sampling-based Motion Planning
General Procedure

1. Initialization:
   - Let $G=(V,E)$ be an undirected search graph with $(q_{\text{start}}, q_{\text{goal}}) \in V$, $E = \emptyset$

2. Vertex Selection Method (VSM):
   - Select a vertex $q_{\text{curr}} \in V$ for expansion

3. Local Planning Method (LPM):
   - Select any $q_{\text{new}} \in C_{\text{free}}$ by sampling
   - Find a path $\tau_s : [0:1] \rightarrow C_{\text{free}}$ such that $\tau(0) = q_{\text{curr}}$ and $\tau(1) = q_{\text{new}}$
   - $\tau_s$ must be collision free, if not, go to 2)

4. Insert new Vertex & Edge in the Graph:
   - Insert $q_{\text{new}}$ to $V$
   - Insert edge between $q_{\text{curr}}$ and $q_{\text{new}}$

5. Check for a Solution:
   - Check if there is a valid path on $G$ from $q_{\text{start}}$ to $q_{\text{goal}}$, if yes: terminate

6. Return to step 2) until any termination criterion is met
Sampling-based Motion Planning

Difficulties

Multi-resolution search required to quickly overcome cavities

(a)

Bidirectional search needed in some cases

(b)

Sometimes even multi-dimensional search needed

(c)

Hard to solve even with multi-dimensional search

(d)
A Sampling sequence should reach every point in C! However, C is uncountably infinite ...

In practice, sampling has to terminate early. Hence the sequence of sampling matters!

Dense Sequence: A sequence getting with increasing size arbitrarily close to every element in C

Random sampling:
- Suppose C=[0,1] and I ⊆ C is an interval of length e. If k samples are chosen independently at random, the probability that none of them falls into I is \((1−e)^k\). As k approaches infinity, this probability converges to zero. This means random sampling is probably dense.

Deterministic sampling:
- Suppose C=[0,1] and we want to place 16 samples
- Simple approach:
  - Select the set \(S=\{i/16 \mid 0<i<16\}\) so that all samples are evenly distributed
  - What if we want to make S into a sequence? What is the best ordering? What if 16 points are not enough, i.e., are not reaching every interesting point in C?
    - Problem with “sorting by increasing value”: after \(i=8\) half of C has been neglected! It would be preferable to have a nice covering of C for every i
Sampling-based Motion Planning
The Van der Corput sequence

- Idea: to **reverse the order of the bits**, when the sequence is represented with binary decimals
- By reversing the bits, the most significant bit toggles in every step, which means that the sequence **alternates** between the **lower** and **upper** halves of C

<table>
<thead>
<tr>
<th>i</th>
<th>Naive Sequence</th>
<th>Binary</th>
<th>Reverse Binary</th>
<th>Van der Corput</th>
<th>Points in [0, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1/16</td>
<td>.0001</td>
<td>.1000</td>
<td>1/2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1/8</td>
<td>.0010</td>
<td>.0100</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3/16</td>
<td>.0011</td>
<td>.1100</td>
<td>3/4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1/4</td>
<td>.0100</td>
<td>.0010</td>
<td>1/8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5/16</td>
<td>.0101</td>
<td>.1010</td>
<td>5/8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3/8</td>
<td>.0110</td>
<td>.0110</td>
<td>3/8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7/16</td>
<td>.0111</td>
<td>.1110</td>
<td>7/8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1/2</td>
<td>.1000</td>
<td>.0001</td>
<td>1/16</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9/16</td>
<td>.1001</td>
<td>.1001</td>
<td>9/16</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5/8</td>
<td>.1010</td>
<td>.0101</td>
<td>5/16</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>11/16</td>
<td>.1011</td>
<td>.1101</td>
<td>13/16</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3/4</td>
<td>.1100</td>
<td>.0011</td>
<td>3/16</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>13/16</td>
<td>.1101</td>
<td>.1011</td>
<td>11/16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7/8</td>
<td>.1110</td>
<td>.0111</td>
<td>7/16</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15/16</td>
<td>.1111</td>
<td>.1111</td>
<td>15/16</td>
<td></td>
</tr>
</tbody>
</table>

Sequence for i<=16

Note: Both method can also be applied for $C \subseteq \mathbb{R}^m$ by sampling each dimension independently
Sampling-based Motion Planning
Rapidly Exploring Dense Trees (RDTs)

Basic algorithm for RDTs (without obstacles):

\[
\text{SIMPLE\_RDT}(q_0) \\
1 \quad \mathcal{G}.\text{init}(q_0); \\
2 \quad \text{for } i = 1 \text{ to } k \text{ do} \\
3 \quad \mathcal{G}.\text{add\_vertex}(\alpha(i)); \\
4 \quad q_n \leftarrow \text{NEAREST}(\mathcal{S}(\mathcal{G}), \alpha(i)); \\
5 \quad \mathcal{G}.\text{add\_edge}(q_n, \alpha(i)); \\
\]

• Requires a dense sequence \( \alpha(i) \)
• Let \( \mathcal{S}(\mathcal{G}) \) be the set of all points reached by \( \mathcal{G} \) (either vertices or edges)
• Connects iteratively edges from \( \alpha(i) \) to those nearest in \( \mathcal{G} \)

Result:

Case 1: Nearest point is a vertex

Case 2: Nearest point is on an edge

Result:

45 iterations

2345 iterations
Sampling-based Motion Planning
Rapidly Exploring Dense Trees (RDTs)

Basic algorithm for RDTs (with obstacles):

\[
\text{RDT}(q_0) \\
1 \quad \mathcal{G}.\text{init}(q_0); \\
2 \quad \text{for } i = 1 \text{ to } k \text{ do} \\
3 \quad \quad q_n \leftarrow \text{NEAREST}(S, \alpha(i)); \\
4 \quad \quad q_s \leftarrow \text{STOPPING-CONFIGURATION}(q_n, \alpha(i)); \\
5 \quad \quad \text{if } q_s \neq q_n \text{ then} \\
6 \quad \quad \quad \mathcal{G}.\text{add_vertex}(q_s); \\
7 \quad \quad \quad \mathcal{G}.\text{add_edge}(q_n, q_s); \\
\]

• \text{STOPPING-CONFIGURATION()} returns the nearest configuration possible in \( C_{\text{free}} \)
Bug trap video on YouTube

http://www.youtube.com/watch?v=qci_AktcrD4
Multi Robot Planning

Problem

- So far, we considered problems with single agents in static environments.

- When multiple robots plan and navigate at the same time, robot-robot collisions might occur.

- Obvious solution: To use a central planner that plans trajectories for all robots simultaneously.
Multi Robot Planning
Problem Formulation

1. World \( \mathcal{W} \) and obstacle region \( \mathcal{O} \)

2. There are \( m \) robots: \( A^1, A^2, \ldots, A^m \)

3. Each robot \( A_i \) has both initial and goal configuration \( q_i^{\text{init}}, q_i^{\text{goal}} \)

4. The state space considers configuration of all robots simultaneously:
\[
X = C^1 \times C^2 \times \cdots \times C^m
\]

A state \( x \in X \) specifies a combination of all robot configurations and may be expressed as:
\[
x = (q^1, q^2, \ldots, q^m).
\]

The dimension of \( X \) is \( N \), which is:
\[
N = \sum_{i=1}^{m} \dim(C^i)
\]
Multi Robot Planning
Problem Formulation

Obstacle region 1: robot-obstacle (walls, etc.);

\[ X_{obs}^i = \{ x \in X \mid A^i(q^i) \cap O \neq \emptyset \} \]

Obstacle region 2: robot-robot:

\[ X_{obs}^{ij} = \{ x \in X \mid A^i(q^i) \cap A^j(q^j) \neq \emptyset \} \]

Entire obstacle region:

\[ X_{obs} = \left( \bigcup_{i=1}^{m} X_{obs}^i \right) \bigcup \left( \bigcup_{ij, \ i \neq j} X_{obs}^{ij} \right) \]

5. Initial state: \( x_I \in X_{free} \) with \( x_I = (q_I^1, \ldots, q_I^m) \)

6. Goal state: \( x_G \in X_{free} \) with \( x_G = (q_G^1, \ldots, q_G^m) \)

7. Compute \( \tau : [0, 1] \rightarrow X_{free} \) such that \( \tau(0) = x_{\text{init}}, \ \tau(1) \in x_{\text{goal}} \)
Multi Robot Planning
Complexity

• X can be considered as an ordinary C space (and all the methods we learned can be applied)
  – However, the dimension of X grows linearly with the number of robots!
  – Complete planning algorithms require time that is at least exponential in the dimension of X!
  – Sample-based methods are more likely to scale well in practice when there many robots, but the dimension of X might still be too high
Decoupled planning

• Decoupled approaches
  – search first for motion plans for each single robot while ignoring plans of other robots
  – Solve then occurring conflicts by different strategies, e.g., stop, drive back, driver slower, ...

• Prioritized Planning:
  – Straightforward approach that sorts robots by priority and plans for higher priority robots first
  – Lower priority robots plan by viewing the higher priority robots as obstacles (but in time-space!)

• Incomplete: Planning can fail to find a valid multi-robot path although there exists one!

![Diagram of decoupled planning](image-url)
Prioritized Planning
Optimizing priority schemes

- **Idea:** to **interleave** the search for collision-free paths with the search for a solvable priority scheme
- **Randomized hill-climbing** search
- Randomly **flips** priorities of two robots (max_flips)
- Performs random **restarts** to avoid local minima (max_tries)
- **Any-time algorithm!**
- Can also be implemented as a Genetic Algorithm

```plaintext
for tries := 1 to MAX_TRIES do
    select random order Π
    if (tries = 1) then
        Π* := Π
    for flips := 1 to MAX_FLIPS do
        chose random i,j with i<j
        Π' := swap(i,j,Π)
        If cost(Π') < cost(Π) then
            Π := Π'
    end for;
    If cost(Π) < cost(Π*) then
        Π* := Π
end for;
return Π*
```

M. Benewitz et al.2001
Prioritized Planning
Performance of prioritized schemes

Greedy Priorities: The shorter the path the higher the priority
Multi-Robot Path Planning
Road-Map based approaches

- Automatic generation of road maps (offline)
- A road map consists of streets and street crossings
- On lanes vehicles drive in convoys
- Robots are coordinated at street crossings, otherwise convoy driving
  - Right before left
  - Robot priorities according to cargo
  - Etc.
- Automatic generation by computing Voronoi diagrams
- Street merging by Linear Programming (LP)
- Complete solution (but suboptimal in terms of distance)
Automatic Road Map Generation
Input: Grid Map
Automatic Road Map Generation
Voronoi Graph: Automatic Detection of drivable areas
Automatic Road Map Generation
Constraint solver: Generating the road map
Automatic Road Map Generation
Final Result: Road map with lanes & crossings
Summary

- Sampling-based Planning methods are well suited for Robot Motion Planning

- To find complete solutions in multi-Robot Planning is generally intractable

- However, solutions sufficient for many problem domains can be found by decoupled techniques

- Complete solutions can be found by Road-Map planners