6.141: Robotics systems and science
Lecture 9: Configuration Space and Motion Planning

Lecture Notes Prepared by Daniela Rus
EECS/MIT
Spring 2012

Figures by Nancy Amato, Rodney Brooks, Vijay Kumar

Reading: Chapter 3, and Craig: Robotics

http://courses.csail.mit.edu/6.141/
Challenge: Build a Shelter on Mars
During the last module we saw

- Control architectures: reactive, behavior, deliberative
- Visibility Graphs for Motion Planning
- Started Configuration Space
Today

- Understand c-space
- Motion planning with grids
- Probabilistic motion planning
Transforming to C-Space

Higher dimension

Simpler problem
C-space Overview

- robot maps to a point in higher dimensional space
- parameter for each degree of freedom (dof) of robot
- C-space = set of all robot placements
- C-obstacle = infeasible robot placements

3D C-space (x, y, z)

6D C-space (x, y, z, pitch, roll, yaw)

2n-D C-space ($\phi_1, \psi_1, \phi_2, \psi_2, \ldots, \phi_n, \psi_n$)
C-obstacle for fixed robot orientation
How do we compute C-space

- Identify dimensions
- Compute all c-obstacles
How do we compute c-obstacles?

Step 1: Reflect Robot
C-space Algorithm

Step 3: ConvexHull (Vert ( - Robot) + Vert (Obstacle))
Convex Hull Algorithm
C-obstacle with Rotations

simple 2D workspace obstacle

$\Rightarrow$ complicated 3D C-obstacle

Figure from Latombe’91
Motion Planning Algorithm

(1) Compute c-obstacle for each obstacle
    (Reflect points, Minkowsky sums, convex hull)
(2) Find path from start to goal for point robot

- The robots DOF dictate (1)
- The method for (2) differentiates among motion planning algorithms
Motion Planning Summary

Workspace

Path is swept volume

C-space

Path is 1D curve
How do we find the path? Recall Visibility Graphs

In 2D the V-graph method finds the shortest path from S to G. What about 3D?
How hard is this to compute?
The Complexity of Motion Planning

Most motion planning problems are PSPACE-hard
[Reif 79, Hopcroft et al. 84 & 86]

The best deterministic algorithm known has running
time that is exponential in the dimension of the robot’s
C-space [Canny 86]
• C-space has high dimension - 6D for rigid body in 3-space
• simple obstacles have complex C-obstacles impractical to compute
  explicit representation of freespace for high dof robots

So … attention has turned to approximation and
randomized algorithms which
• trade full completeness of the planner
• for a major gain in efficiency
Exact Cell Decomposition for finding path
Searching the Convex Cells for finding path

Build graph
Search for path
Approximate Cell Decomposition
Cell Connectivity Graph
Probabilistic Road Maps (PRM) for finding paths [Kavraki at al 96]

C-space

Roadmap Construction (Pre-processing)
1. Randomly generate robot configurations (nodes)
   - discard nodes that are invalid
2. Connect pairs of nodes to form roadmap
   - simple, deterministic local planner (e.g., straightline)
   - discard paths that are invalid

Query processing
1. Connect start and goal to roadmap
2. Find path in roadmap between start and goal
   - regenerate plans for edges in roadmap

Primitives Required:
1. Method for Sampling points in C-Space
2. Method for `validating’ points in C-Space
PRMs: Pros

1. PRMs are *probabilistically complete*
2. PRMs apply easily to high-dimensional C-space
3. PRMs support fast queries w/ enough preprocessing

Many success stories where PRMs solve previously unsolved problems
More PRMS

PRMs: Pros
1. PRMs are *probabilistically complete*
2. PRMs apply easily to high-dimensional C-space
3. PRMs support fast queries w/ enough preprocessing

Many success stories where PRMs solve previously unsolved problems

PRMs: Cons
1. PRMs don’t work as well for some problems:
   – unlikely to sample nodes in *narrow passages*
   – hard to sample/connect nodes on constraint surfaces
Sampling Around Obstacles
[Amato et al 98]

To Navigate Narrow Passages we must sample in them
• most PRM nodes are where planning is easy (not needed)

Idea: Can we sample nodes near C-obstacle surfaces?
• we cannot explicitly construct the C-obstacles...
• we do have models of the (workspace) obstacles...
OBPRM: Finding points on C-obstacles

Basic Idea (for workspace obstacle S)
1. Find a point in S’s C-obstacle (robot placement colliding with S)
2. Select a random direction in C-space
3. Find a free point in that direction
4. Find boundary point between them using binary search (collision checks)

Note: we can use more sophisticated approaches to try to cover C-obstacle
Repairing Paths [Amato et al]

Even with the best sampling methods, roadmaps may not contain valid solution paths
• may lack points in narrow passages
• may contain approximate paths that are nearly valid
Repairing Paths [Amato et al]

Even with the best sampling methods, roadmaps may not contain valid solution paths
  • may lack points in narrow passages
  • may contain approximate paths that are nearly valid

Repairing/Improving Approximate Paths

1. Create initial roadmap
2. Extract *approximate path* $P$
3. Repair $P$ (push to C-free)
   - Focus search around $P$
   - Use OBPRM-like techniques
Rapidly-Exploring Random Tree (RRT)

- [LaValle and Kuffner 2001]

- Easy to implement
- Quickly finds an answer for a large variety of systems
- Works in high dimensional spaces
RRT Algorithm

• Given
  – Obstacles
  – Start state
  – Goal set, $G$

1. $t =$ tree rooted at start

2. repeat

3. $s =$ random configuration

4. $n =$ nearest node to $s$ in $t$

5. attempt to connect $n$ to $s$ with a path

6. until $t$ contains a point in $G$
RRT Algorithm

• Given
  – Obstacles
  – Start state
  – Goal set, \( G \)

1. \( t = \text{tree rooted at } \text{start} \)

2. repeat

3. \( s = \text{random configuration} \)

4. \( n = \text{nearest node to } s \text{ in } t \)

5. attempt to connect \( n \) to \( s \) with a path

6. until \( t \) contains a point in \( G \)
RRT Algorithm

• Given
  – Obstacles
  – Start state
  – Goal set, $G$

1. $t =$ tree rooted at start

2. repeat

3. $s =$ random configuration

4. $n =$ nearest node to $s$ in $t$

5. attempt to connect $n$ to $s$ with a path

6. until $t$ contains a point in $G$
RRT Example
Motion Planning Summary

- Compute c-space for each obstacle
- Compute representation
- Find path from start to goal
  - What should we optimize?