Motion Planning I

D.A. Forsyth
(with a lot of H. Choset, and some J. Li)
What is motion planning?

- The automatic generation of motion
  - Path + velocity and acceleration along the path
Basic Problem Statement

- Motion planning in robotics
  - Automatically compute a path for an object/robot that does not collide with obstacles.

Diagram:
- Robot and Obstacle Geometry
- Robot Description
- Start and Goal
- Planning Algorithm
- A path from start to goal
Why is this not just optimization?

- Find minimum cost set of controls that
  - take me from A to B
  - do not involve
    - collision
    - unnecessary extreme control inputs
    - unnecessary extreme behaviors

These will have to deal with collisions, etc.

\[
\begin{align*}
\text{minimize } f(x) & \quad (1a) \\
\text{subject to } & \quad (1b) \\
g_i(x) & \leq 0, \quad i = 1, 2, \ldots, n_{ineq} \quad (1c) \\
h_i(x) & = 0, \quad i = 1, 2, \ldots, n_{eq} \quad (1d)
\end{align*}
\]
Is motion planning hard?

Basic Motion Planning Problems

EXPSPACE
EXPTIME
PSPACE
NP
P
NL
Degrees of Freedom

- The geometric configuration of a robot is defined by $p$ degrees of freedom (DOF).
- Assuming $p$ DOFs, the geometric configuration $A$ of a robot is defined by $p$ variables:

$$A(q) \text{ with } q = (q_1, \ldots, q_p)$$

- Examples:
  - Prismatic (translational) DOF: $q_i$ is the amount of translation in some direction.
  - Rotational DOF: $q_i$ is the amount of rotation about some axis.

Our car has 3
Allowed to move only in $x$ and $y$: 2DOF

Allowed to move in $x$ and $y$ and to rotate: 3DOF ($x, y, \theta$)
Configuration Space (C-Space)

- Configuration space $\mathcal{C}$ = set of values of $q$ corresponding to legal configurations of the robot
- Defines the set of possible parameters (the search space) and the set of allowed paths

$q = (x, y, \theta)$
$\mathcal{C} = \mathbb{R}^2 \times \text{set of 2-D rotations}$

$q = (q_1, q_2)$
$\mathcal{C} = \text{2-D rotations} \times \text{2-D rotations}$
Free Space: Point Robot

- $\mathcal{C}_{\text{free}} = \{\text{Set of parameters } q \text{ for which } A(q) \text{ does not intersect obstacles}\}$
- For a point robot in the 2-D plane: $\mathbb{R}^2$ minus the obstacle regions
Free Space: Symmetric Robot

- We still have $\mathbb{C} = \mathbb{R}^2$ because orientation does not matter
- Reduce the problem to a point robot by expanding the obstacles by the radius of the robot
Free Space: Non-Symmetric Robot

- The configuration space is now three-dimensional $(x, y, \theta)$
- We need to apply a different obstacle expansion for each value of $\theta$
- We still reduce the problem to a point robot by expanding the obstacles
Any Formal Guarantees? Generic Piano Movers Problem

- Formal Result (but not terribly useful for practical algorithms):
  - $p$: Dimension of $\mathcal{C}$
  - $m$: Number of polynomials describing $\mathcal{C}_{\text{free}}$
  - $d$: Max degree of the polynomials

- A path (if it exists) can be found in time exponential in $p$ and polynomial in $m$ and $d$

Observation

- Generally, searching a graph is pretty straightforward
  - Dijkstra, A*, etc - know how to do this

- Strategy
  - get a graph we can search
Roadmaps

- **General idea:**
  - Avoid searching the entire space
  - Pre-compute a (hopefully small) graph (the roadmap) such that staying on the "roads" is guaranteed to avoid the obstacles
  - Find a path between $q_{\text{start}}$ and $q_{\text{goal}}$ by using the roadmap
Visibility Graphs

In the absence of obstacles, the best path is the straight line between $q_{\text{start}}$ and $q_{\text{goal}}$. 
Visibility Graphs

- Visibility graph \( G = \) set of unblocked lines between vertices of the obstacles + \( q_{\text{start}} \) and \( q_{\text{goal}} \)
- A node \( P \) is linked to a node \( P' \) if \( P' \) is visible from \( P \)
- Solution = Shortest path in the visibility graph
Issues

• Constructing
  • Relatively straightforward with a sweep algorithm
  • Variant (visibility complex) root cause of early computer games
    • Wolfenstein 3D, Doom II, etc
• What if configuration space is not 2D
  • You can still construct, MUCH harder
• MANY locally optimal paths
  • topology of free space clearly involved

Choset slides
Visibility Graphs: Weaknesses

- Shortest path but:
  - Tries to stay as close as possible to obstacles
  - Any execution error will lead to a collision
  - Complicated in \( \gg 2 \) dimensions
- We may not care about strict optimality so long as we find a safe path. Staying away from obstacles is more important than finding the shortest path
- Need to define other types of “roadmaps”
Voronoi Diagrams

- Given a set of data points in the plane:
  - Color the entire plane such that the color of any point in the plane is the same as the color of its nearest neighbor
Voronoi Diagrams

- Voronoi diagram = The set of line segments separating the regions corresponding to different colors
  - Line segment = points equidistant from 2 data points
  - Vertices = points equidistant from > 2 data points
Voronoï Diagrams

- Complexity (in the plane):
  - $O(N \log N)$ time
  - $O(N)$ space

(See for example http://www.cs.cornell.edu/Info/People/chew/Delaunay.html for an interactive demo)
Voronoï Diagrams (Polygons)

- Key property: The points on the edges of the Voronoï diagram are the *furthest* from the obstacles
- Idea: Construct a path between $q_{\text{start}}$ and $q_{\text{goal}}$ by following edges on the Voronoï diagram
- (Use the Voronoï diagram as a roadmap graph instead of the visibility graph)
Voronoi Diagrams: Planning

- Find the point $q_{\text{start}}^*$ of the Voronoi diagram closest to $q_{\text{start}}$
- Find the point $q_{\text{goal}}^*$ of the Voronoi diagram closest to $q_{\text{goal}}$
- Compute shortest path from $q_{\text{start}}^*$ to $q_{\text{goal}}^*$ on the Voronoi diagram
Voronoi: Weaknesses

- **Difficult to compute in higher dimensions or nonpolygonal worlds**
- Approximate algorithms exist
- Use of Voronoi is not necessarily the best heuristic ("stay away from obstacles") Can lead to paths that are much too conservative
- Can be unstable → Small changes in obstacle configuration can lead to large changes in the diagram
Approximate Cell Decomposition

- Define a discrete grid in C-Space
- Mark any cell of the grid that intersects $C_{\text{obs}}$ as blocked
- Find path through remaining cells by using (for example) $A^*$ (e.g., use Euclidean distance as heuristic)
- Cannot be complete as described so far. Why?

Choset slides
Approximate Cell Decomposition

- Cannot find a path in this case even though one exists
- Solution:
  - Distinguish between
    - Cells that are entirely contained in \( C_{\text{obs}} \) (FULL) and
    - Cells that partially intersect \( C_{\text{obs}} \) (MIXED)
- Try to find a path using the current set of cells
- If no path found:
  - Subdivide the MIXED cells and try again with the new set of cells
Approximate Cell Decomposition: Limitations

• Good:
  – Limited assumptions on obstacle configuration
  – Approach used in practice
  – Find obvious solutions quickly

• Bad:
  – No clear notion of optimality (“best” path)
  – Trade-off completeness/computation
  – Still difficult to use in high dimensions
Exact Cell Decomposition

Any path within one cell is guaranteed to not intersect any obstacle.
Exact Cell Decomposition

- The graph of cells defines a roadmap
Exact Cell Decomposition

- The graph can be used to find a path between any two configurations

Choset slides
A version of exact cell decomposition can be extended to higher dimensions and non-polygonal boundaries ("cylindrical cell decomposition")

- Provides exact solution → completeness
- Expensive and difficult to implement in higher dimensions
Potential Fields

- Stay away from obstacles: Imagine that the obstacles are made of a material that generate a repulsive field
- Move closer to the goal: Imagine that the goal location is a particle that generates an attractive field
Move toward lowest potential Steepest descent (Best first search) on potential field

Choset slides
$U_g(q) = d^2(q, q_{goal})$

$U_o(q) = \frac{1}{d^2(q, \text{Obstacles})}$

$U(q) = U_g(q) + \lambda U_o(q)$

Distance to goal state

Distance to nearest obstacle point.
Note: Can be computed efficiently by using the *distance transform*

$\lambda$ controls how far we stay from the obstacles
Potential Fields: Limitations

- Completeness?
- Problems in higher dimensions

Choset slides
Local Minimum Problem

- Potential fields in general exhibit local minima.
- Special case: Navigation function
  - \( U(q_{goal}) = 0 \)
  - For any \( q \) different from \( q_{goal} \), there exists a neighbor \( q' \) such that \( U(q') < U(q) \)
Getting out of Local Minima I

- Repeat
  - If $U(q) = 0$ return Success
  - If too many iterations return Failure
  - Else:
    - Find neighbor $q_n$ of $q$ with smallest $U(q_n)$
    - If $U(q_n) < U(q)$ OR $q_n$ has not yet been visited
      - Move to $q_n$ ($q \leftarrow q_n$)
      - Remember $q_n$

May take a long time to explore region “around” local minima
Think of this the following way:
- impose a grid
- do depth first search on the potential

Idea:
- other kinds of search
- randomization should help a lot

Concern:
- what if q has lots of neighbors?
Getting out of Local Minima II

• Repeat
  – If $U(q) = 0$ return Success
  – If too many iterations return Failure
  – Else:
    • Find neighbor $q_n$ of $q$ with smallest $U(q_n)$
    • If $U(q_n) < U(q)$
      – Move to $q_n$ ($q \leftarrow q_n$)
    • Else
      – Take a random walk for $T$ steps starting at $q_n$
      – Set $q$ to the configuration reached at the end of the random walk

Similar to stochastic search and simulated annealing: We escape local minima faster
• **Intuition:**
  • random walk should get you out of local minima
  • then slide down the potential function

• **Concern:**
  • what if dimension is high?
    • random walk may not get out of local minima efficiently