Motion Planning

An EXTREMELY brief overview of path planning for mobile robots
The Big Picture

“High-level” goal generation facilities

“More-or-less” motion planning for mobile robots

“Lower-level” perception and plan execution facilities
Motion Planning

- Deliberative process
  - Considers the possibilities
  - Judges the quality of each option
  - Picks the “best” option from some sense of relative merit

- Core Idea: How do I get from my current state to my desired state?
Motion Planning

- Go from point A to point B
- Cover an area (floor vacuuming)
- Exploration and/or search
Motion Planning: Example

“Test track” for duration path planning tests

Exploration of Abandoned Mine
Trade-offs

- Planning is computationally expensive since you may have many, many possible actions and configurations.
- Examining each one will take a long time.
- When a decision is finally reached, the world has already changed.
- So, what do you do?
Trade-offs

- So, simply the problem by making assumptions
- But, assumptions reduce your accuracy and understanding of the problem
- Many assumptions will speed up the process, but produce poor plans
- Key is balance!
About Motion Planning

Known Environments (Model)
- Explicit motion plans
- Implicit motion plans

Unknown Environments
- Sensor based motion planning

Usually combines Motion Planning Trajectory Generation
Recipe for Motion Planning

- 2 Cup: Modeling of World & Robot
- 1 Cup: Representation of Actions
- 1 Cup: Metric for comparing actions
- 1 Tbs: Objective
- Place in a search routine for a few milliseconds and serve
Modeling the World & Robot
Modeling the World & Robot

- Variety of methods
- Basic idea are binary worlds: “obstacles” and free space
- Obstacles are relative objects, one robot’s obstacle is another robot’s speed bump
- World is more than obstacles alone
Modeling the World & Robot

Binary World

Cost Encoded World

White is obstacle, black is free space, gray is somewhere in the middle.
Some locations in free space are more desirable than others.

Steep or rocky terrain, for example, cost more energy to traverse, increases chance of slip or tip-over, more risky to robot…

Why get close to obstacles unless absolutely necessary?
Modeling the World & Robot

- **Workspace**: A bounded region in Euclidean 2D or 3D
- **Free space**: Subset of workspace where we are free to plan (no obstacles)
- **Obstacle space**: Subset of workspace that is occupied by obstacles
Modeling the World & Robot

- Configuration space (c-space) used to transform robot “volume” to point
- More formally, c-space is set of all configurations for an objects
- Dimension: \#dof-constraints-symmetry

Example configuration space for 2 dof manipulator
Modeling the World & Robot

- C-space for non-ballistic mobile robot is $\mathbb{R}^2 \times S$
- Approximate robot body with circle for easier c-space calc
- Search complexity directly related to size of c-space and obstacle complexity

Figure 4: C Space Obstacles. A kind of rolling contact convolution produces the C space obstacle for a given robot and obstacle pair. Rightmost case is a constant heading slice of C Space. The union of all such slices is the 3D shape.
Modeling the World & Robot

Configuration space for non-circular mobile robot
Modeling the World & Robot

Continuous vs. Discrete Spaces

Figure 7 Continuous Boundary Representation. The three lines representing the vehicle can be checked for collision with the 13 lines representing the obstacles very quickly. Bounding boxes or spheres placed around each object (or each line segment) can be used to quickly eliminate unnecessary intersection checks. This is how the course simulator detects collisions.

Figure 8 Grids and Sampled Boundary Representations. The boundaries of obstacles are represented as occupied cells in a regular array. The robot may or may not be.
Modeling the World & Robot

3D Grid

Quadtrees

Figure 9 Octree Representation. Whereas the last figure required 3x4x64 = 768 cells, this one requires 8+5+18+21=47 cells.

Legend

Figure 10 Octree Implementation. Octrees are implemented as a tree containing filled, unfilled, and partially filled cells. Only partially filled cells have children.
Motion Modeling
Modeling Motion (briefly)

- Typically idealized as line segments
- Not helpful if generated path cannot be followed
- Consider constraints on curvature and velocity $|\kappa(s)| < \kappa_{\text{max}}$, $|\dot{\kappa}(s)| < \dot{\kappa}_{\text{max}}$, etc.
- Constraints are checked during path planning or post processed
- Dynamics are difficult/helpful depending on how they are applied
Techniques for planning
Planning: Bugs Algorithm

- Not something you would really implement, but demonstrates the theme of robot motion planning
- Makes assumptions of perfect positioning and idealizes robot to point
- Has provable guarantees
Planning: Bugs 1

Bugs 1 in a nutshell

- Robot is point with contact sensor
- Move in straight line towards goal
- If you encounter an obstacle, encircle it and track the closest point to the goal
- Once you reach the start point, go to closest point and continue to goal along line

Bugs 1 with Solution

Bugs 1 with No Solution
Bugs in Motion
Bugs in Motion
Better Bugs?

Tangent Bug
Properties of Motion planning

- **Soundness**: Does the plan satisfy all of the imposed constraints?
- **Completeness**: If a solution exists, will it be found?
- **Optimality**: If more than one solution exists, will the “best” solution be found? (“Best” is in the eye of the beholder)
Planning: Potential Fields

- Idea: robot is an object moving in a force field
- Obstacles produce repulsive forces
- Goal produces attractive forces
- Robot is pulled to goal and pushed from obstacles

Figure 5: Potential Fields. In this representation, goals are attractive and obstacles are repulsive.
Planning: Potential Fields

Sum of Attractive and Repulsive Forces

\[ U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q) \]

Basic Algorithm

- Calculate \( U_{\text{att}}(q) \) from distance to goal
- Calculate \( U_{\text{rep}}(q) \) from distance to obstacles
- Sum forces at given position
- Magnitude and heading used
Planning: Discrete Potential

These algorithms utilize grid structure to calculate potential.

Original occupancy grid

Brushfire Algorithm

Initialize fifo will all cells with cost == LETHAL
while cell w/ cost == FREE exists
Pop cell off fifo and look at adjacent cells
If adjacent cell has cost == FREE
Cost = Parent Cell Cost – 1
Add to fifo

Wavefront Algorithm

Initialize fifo will start cell
While fifo not empty
Pop cell off fifo and look at adjacent cells
If adjacent cell != LETHAL && cell cost < Parent Cell + 1
Cost = Parent Cell Cost + 1
Add to fifo
Planning: Potential Problems

Potential fields fail on this simple case due to local minima.
Planning: Potential Fields

Advantages:
- Fast computation
- Represent non-polygonal worlds
- Continuous & discrete space
- "Sensor based" (relax world modeling)

Disadvantages:
- Local minima
- Does not explicitly represent the motions of robot
- As stated, works on a binary world
Planning: Roadmaps

- Reduces free-space to 1D manifolds
- Graph-like structure
- Computationally feasible
Planning: Roadmaps

- Concepts of Roadmaps
  - **Accessibility**: there is a way to get from some point in free space onto the RM
  - **Departability**: there is a way to get to some point in free space from the RM
  - **Connectivity**: there is a series of connected paths that takes you from access point to point of departure
Planning: RM – Visibility Maps

- World: set of polygonal obstacles
- Connect all vertices that are visible to one another
- Connect start and goal to visible vertices
- Search* network for a path from start to goal
- Plane sweeping
Planning RM: GVD

Constructing the GVD from sensor data

- Move until equidistant to two obstacles
- Maintain equidistance by moving along tangent
- At three or more equidistant points, mark and move down next segment
- Repeat until all edges are traversed
Planning RM: GVD

Planning motion on the GVD

- Access by driving in a straight line to the closest point on the graph
- Follow edge segments until the point on the GVD closest to the goal
- Depart the GVD in a direct line to the goal
Planning: Roadmaps

Advantages:
- Computationally fast
- Graphs
- “Sensor based” - GVD

Disadvantages:
- Polygonal assumption
- Difficult to extend beyond two dimensions
- Binary world
- Erroneous motions
- Close to obstacles
- Little consideration for dynamics
Planning: Cell Decomposition

- Cell Decomposition
- Scan line in 1D to create cells
- Plan by searching* adjacent cells
- Path is midpoints on borders

Trapezoidal Decomposition
Planning: Cell Decomposition

- Place empty cell on root
- If mixture of obstacle and free, break into N cells
- Label each cell free, full, or mixed and insert into tree
- Search for solution
- If no solution found, repeat decomposition at next resolution
- Terminate at some max resolution threshold and declare no solution found
Planning: Cell Decomposition

Advantages:
- Computationally feasible (if properly bounded)
- Graphs structures, many existing techniques to search

Disadvantages:
- Difficult to extend beyond two dimensions
- Binary world
- Prefers known environments
- Erroneous motions
- Close to obstacles
- Little consideration for dynamics
Graph Searches

- Graphs, grids, lattices, networks, trees, etc. are very popular in motion planning.
- Methods of efficiently searching these spaces determines efficiency of motion planning.

*Figure 5 Grids as Networks. Grids are equivalent to networks with regular, locally connected structure.*
Graph Searches

Graph is a collection of nodes and edges

Undirected graph with many cycles

Graph with no cycles is a tree

Directed graph with a cycle

Line implies undirected edges
Graph Searches

- Depth-first
- Iterative Deepening
- Breadth-first
- Best-first

- $N =$ number of states in problem
- $b =$ average branching factor
- $L =$ path length from start to goal with the shortest number of steps
DFS

- Initialize with start state on stack
- While (goal state has not been popped and queue is not empty)
  - Pop current state and expand successors
  - If (successor has not been visited)
    - Add to stack
    - Remember parent
DFS

- Complete? Yes
- Guaranteed Optimal? No
- Time Complexity? $O(b^{L_{\text{max}}})$
- Space Requirements? $O(L_{\text{max}})$
Iterative Deepening

- DFS which searches for paths of length 1
- If “1” failed, do DFS for length 2
- Repeat until success

- Complete? Yes
- Guaranteed Optimal? Yes
- Time Complexity? $O(b^L)$
- Space Requirements? $O(L)$
BFS

- Initialize with start state on queue
- While (goal state has not been popped and queue is not empty)
  - Pop current state and expand successors
  - If (successor has not been visited)
    - Add to queue
    - Remember parent
BFS

- Complete? Yes
- Guaranteed
- Optimal? Yes
- Time Complexity? Min($b^L,bN$)
- Space Requirements? O(N)
Best-first “Greedy Search”

- Initialize with start state on priority-queue
- While (goal state has not been popped and queue is not empty)
  - Pop least cost state and expand successors
  - If (successor has not been visited)
    - Add to priority queue with heuristic cost to goal
    - Remember parent
Best-first “Greedy Search”

- Complete? Yes
- Guaranteed Optimal? No
- Time Complexity? \( \text{Min}(b^L, bN) \)
- Space Requirements? \( O(N) \)
Even Better… to be continued

- Better grid searches, A*
- Replanning: D*, RRTs
- Probabilistic approaches
- Dealing with nonholonomic constraints
- Example of system used in subterranean robotics