FCL
and
Motion Planning with Uncertainty

Jia Pan
Summer 2011
Mentor: Sachin Chitta
Motivation

- Human environments
  - Clutter
  - Dynamic obstacles
- Data from 3D sensors
  - Large number of points
    - (~10,000 for laser scans, ~20,000 for stereo)
- Realtime computation important for fast online reactive grasping, motion planning

Fast, efficient collision checking is essential for any fast online robot operations in human environments
Motivation

• Visual sensor range / coverage limited
  • E.g. PR2 cannot see top of bookshelf

• Object occlusion, sensor occlusion
  • Big issue in cluttered environment
  • Parts of environment are unknown

• Imperfect, noisy sensing
  • Large uncertainty in environment representation

Motion planning should account for uncertainty in the environment
My Contributions

• Collision checking
  – FCL (Fast Collision Library)
• Motion planning with uncertainty
Collision Checking
Current Implementation in ROS

• Based on ODE
  – about 10 years old
  – state of the art in collision checking has advanced greatly
  – no proximity (distance) computation
  – cannot be parallelized easily (need thread lock)
  – C style code without good documentation, difficult to extend
Fast Collision Library (FCL)

- Fully templated code (C++)
- Designed to be easily extendable
  - New state-of-the-art algorithms can be easily added
- Current features include
  - Low level (object level) collision
  - Broad phase collision checking
  - Proximity
  - Continuous collision detection (CCD)
  - Deal with noisy point cloud data (experimental)
Class Architecture
Low level Collision Checking

• Mesh collision and mesh / shape collision
  • Based on bounding volume hierarchy (BVH) to cull out the collision test of far-away parts
  • Similar idea is used in distance computation

• Shape collision
  • Based on GJK / MPR algorithm to provide collision/penetration depth information
High Level Collision Checking

• Apply hierarchical culling technique to accelerate collision test among many objects
  – Especially useful for robot self collision

• Called ‘collision manager’ in FCL
  – Four types of collision managers are provided
    • Brute force
    • ODE style
    • Sweep and prune
    • Interval tree
  – Spatial subdivision methods will be added later.
Continuous Collision Detection

• More rigorous guarantee for collision state of motion within a given time interval
  – Return the contact time
Results

• Implemented in two packages
  – *fcl* - an independent library with the core collision/proximity functions.
  – *collision_space_fcl*
    • Same API as *collision_space*
    • Easy integration with motion planners, kinematics, etc.

• Integrated with ROS and the PR2
  – Integrated with OMPL and the motion planning pipeline
  – Tested in simulation and the PR2 robot

• Separate repository on kforge
  • (https://kforge.ros.org/project/fcl)
Results

• Tested against ODE
  – 100 obstacles in random poses within robot workspace
  – Robot model used is the PR2
  – 1000 random configurations of the robot arms

• Average timings (in seconds)

<table>
<thead>
<tr>
<th>Collision type</th>
<th>Sphere/Mesh</th>
<th>Cylinder/Mesh</th>
<th>Box/Mesh</th>
<th>Mesh/Mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODE</td>
<td>0.27</td>
<td>0.31</td>
<td>0.34</td>
<td>6.19</td>
</tr>
<tr>
<td>FCL</td>
<td>0.34</td>
<td>0.37</td>
<td>0.37</td>
<td>0.72</td>
</tr>
</tbody>
</table>

  – Shapes (basic primitives) are converted into meshes in FCL
  – Shape collision checking needs further integration
Fast Collision Library (FCL)

Jia Pan, Dinesh Manocha
UNC Chapel Hill

Sachin Chitta
Willow Garage Inc.
Point Cloud Results
Motion Planning with Uncertainty
Environment Representation

- Octomap
  - Compact representation for large scene
  - Can model free space, occupied space and unknown space efficiently
  - Encode the uncertainty in each cell
  - Update when environment changes
Planning with Costmap

• Transition-based RRT [Jaillet et al.]
  – Compute a path with minimum overall variation
  – Use annealing-alike method to balance between exploration and exploitation of random search process
Planning with Costmap

• Temperature ($T$)
  • low $T$: only low-cost RRT edges can be added $\Rightarrow$ more exploitation
  • High $T$: high-cost RRT edges are allowed $\Rightarrow$ more exploration

• $T$ update
  • When one RRT edge is added, decrease $T$
  • When RRT extend fails many times, increase $T$
Planning with Uncertainty

• T-RRT does not meet our requirements
  – TRRT: path with minimum variations
  – We need: path with minimum uncertainty

• Modification
  – Modify the cost of each RRT edge

\[
\text{cost}(a, b) = f(b) + h(b) - f(a) - h(a)
\]

\[
= u(a, b) + h(b) - h(a)
\]

• u: uncertainty cost from octomap intersection
• h: distance to goal
Results

• Integrated with OMPL & ROS
• Tested on PR2
• Further testing needed
  – Explore right set of parameters
  – More scenarios
Movie
Conclusion

• Collision checking
  – Integrated into ROS and OMPL
  – Released on kforge (separate repository called FCL)
  – Should be ready for regular use
  – Paper for ICRA 2012

• Planning with uncertainty
  – Maybe for ICRA 2012

• Future work
  – Combine RRT and A*