Motion Planning

Jana Kosecka
Department of Computer Science

- Discrete planning, graph search, shortest path, A* methods
- Road map methods
- Configuration space

Slides thanks to http://cs.cmu.edu/~motionplanning, Jyh-Ming Lien

1

Motion Planning

- Planning in continuous state spaces
- Workspace of a robot is a continuous space
- Simplifying assumption (for now) robot is a point

- Given a point robot and a workspace described by polygons. In this case workspace is same as configuration space

- Example 2D configuration space

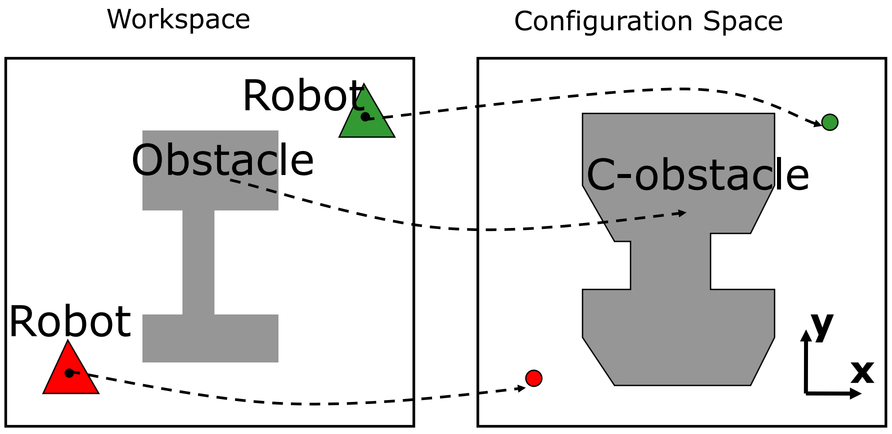
7

Configuration Space

- Well, most robot is not a point and can have arbitrary shape
- What should we do if our robot is not a point?
- Convert rigid robots, articulated robots, *etc.* into points
- Apply algorithms for moving points

8

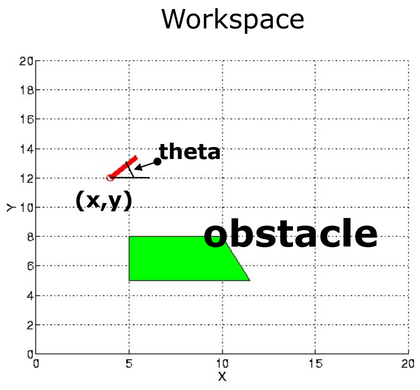
Configuration Space

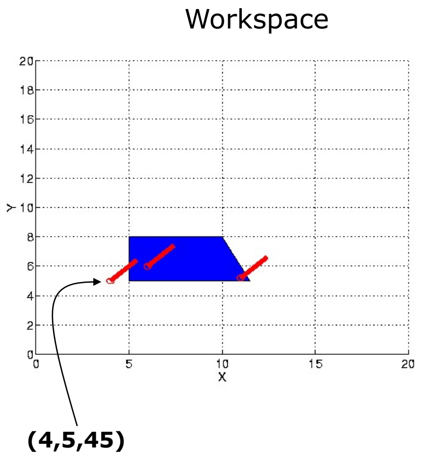


- C-obstacle is a polygon – set of configurations robot cannot attain
- Configuration space – all attainable configurations
- Planning paths in configuration space

9

Workspace Obstacle

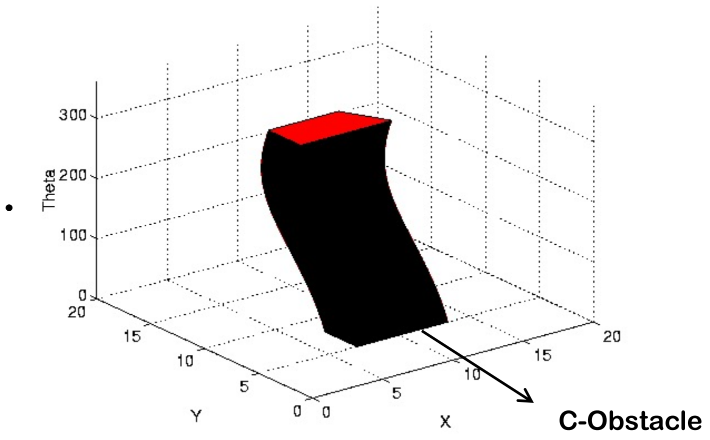




Configuration (x,y,theta)

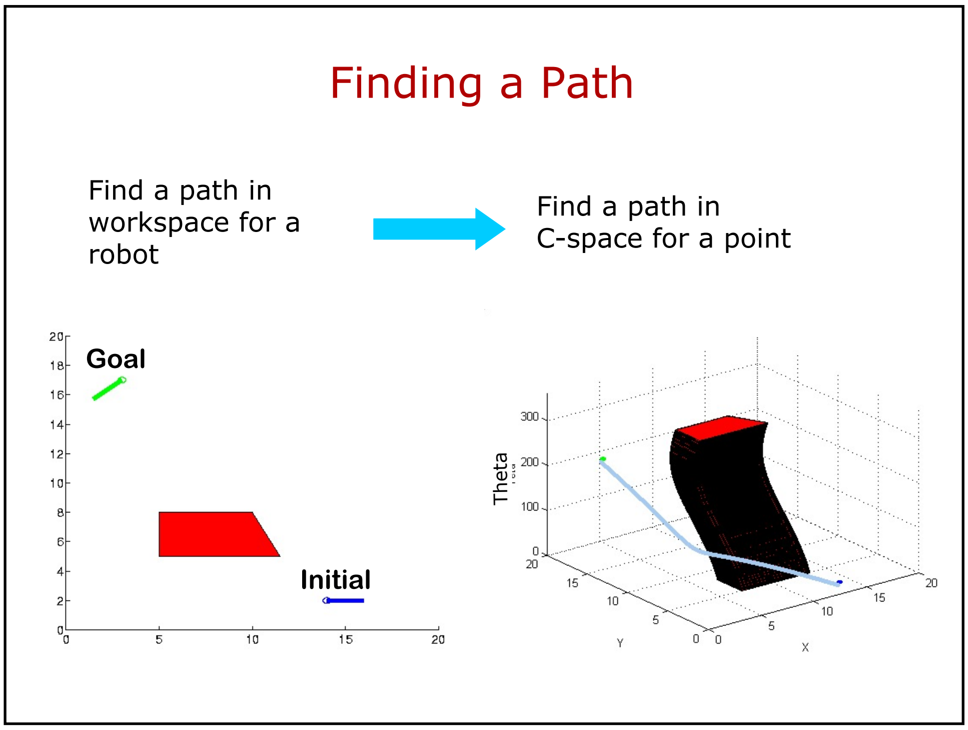
10

C-Space Obstacle



Really look like this ? Every point in the C-obst corresponds to the configuration where the robot would collide with the obstacle

11

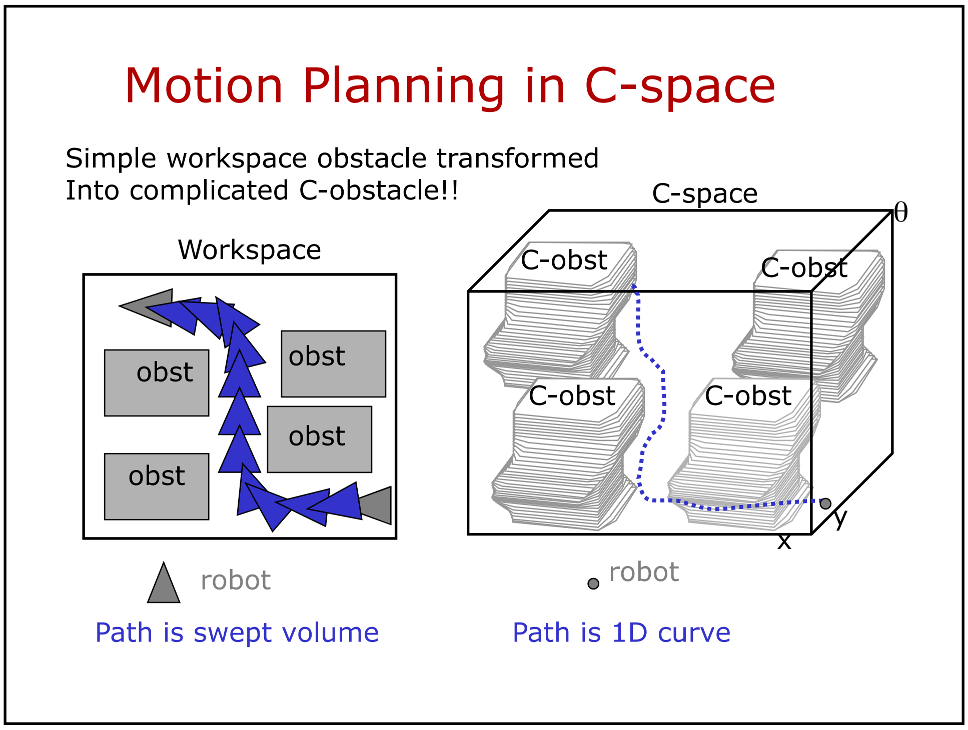


Finding a Path

Find a path in workspace for a robot → Find a path in C-space for a point

Goal
Initial
Theta
X
Y

12



Motion Planning in C-space

Simple workspace obstacle transformed
Into complicated C-obstacle!!

Workspace
obst
obst
obst
obst
robot
C-space
C-obst
C-obst
C-obst
C-obst
θ
x
y
robot

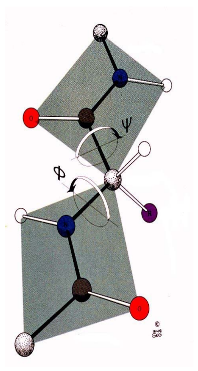
Path is swept volume

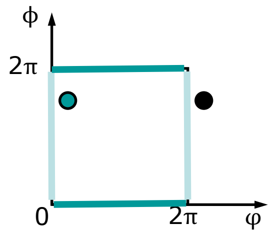
Path is 1D curve

13

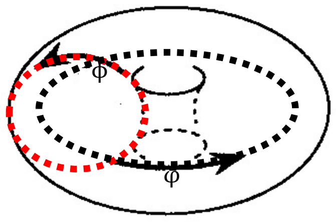
Topology of the configuration pace

- The topology of *C* is usually **not** that of a Cartesian space R^n.



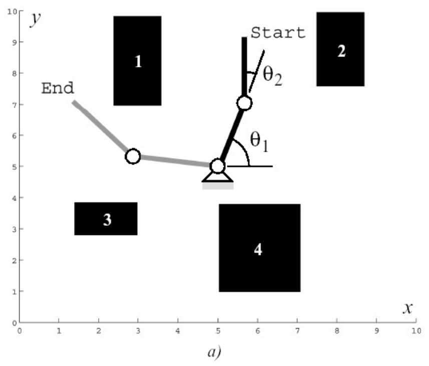


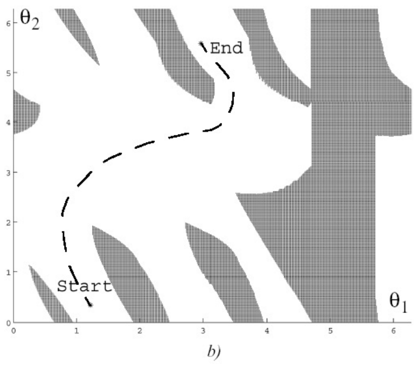
$C = S^1 \times S^1$



14

Configuration Space

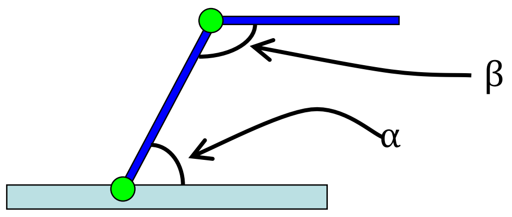




15

Workspace

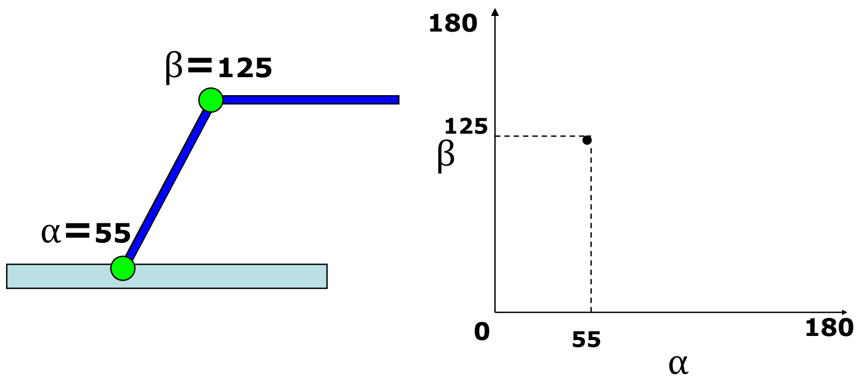
Degree of freedom (DOF)



16

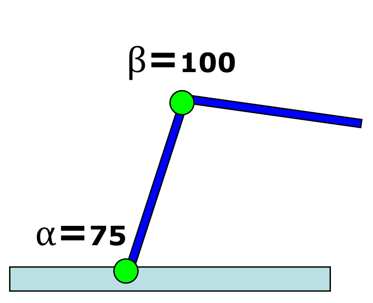
Configuration Space

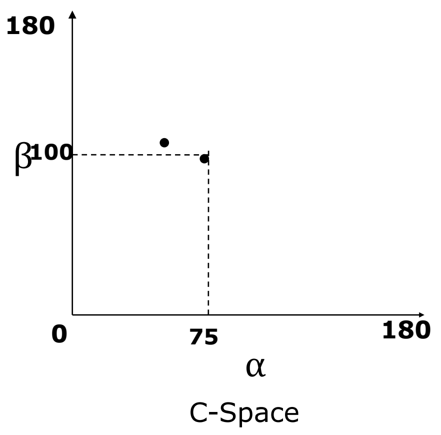
C-Space



17

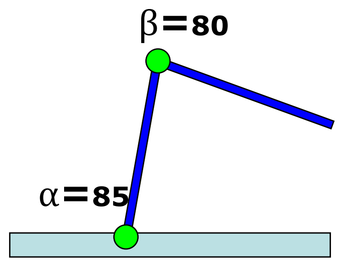
C-Space

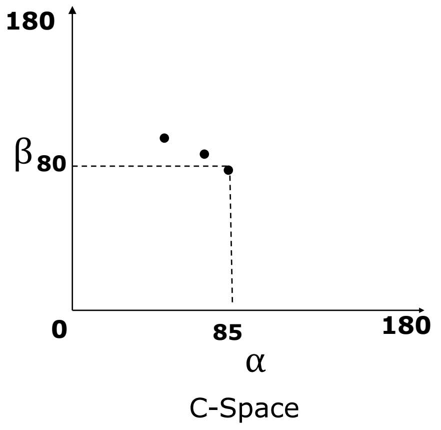




18

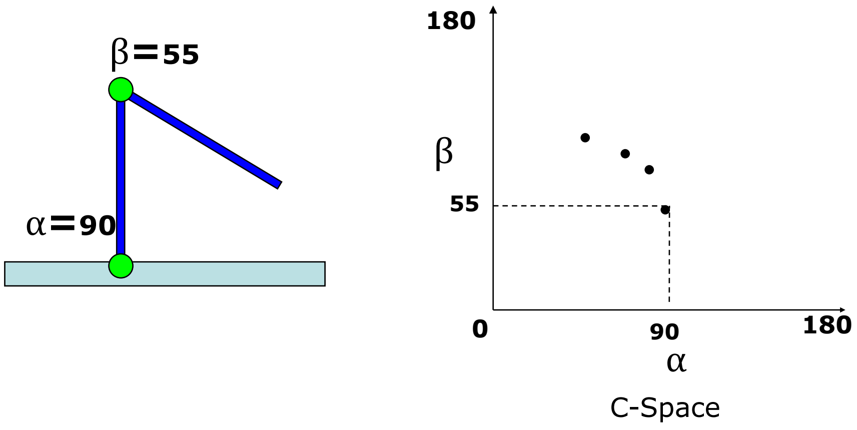
C-Space





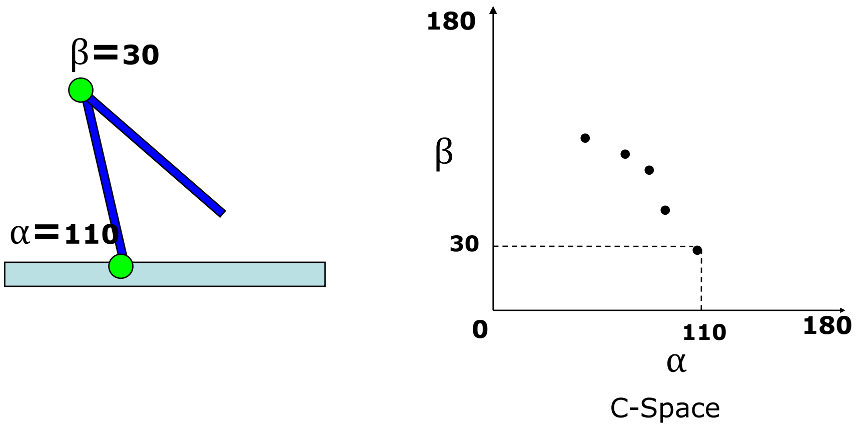
19

C-Space



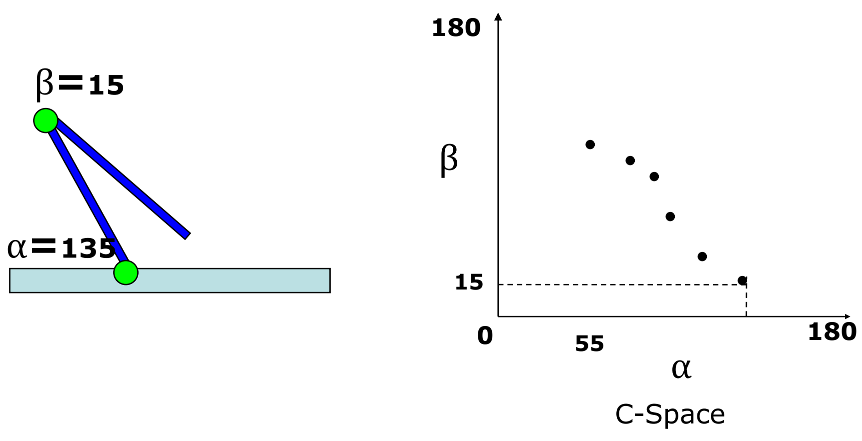
20

C-Space



21

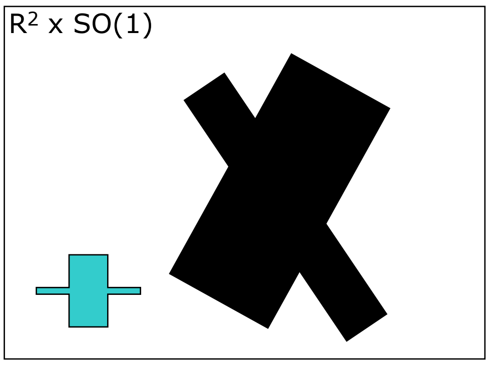
C-Space



22

Example: rigid robot in 2-D workspace

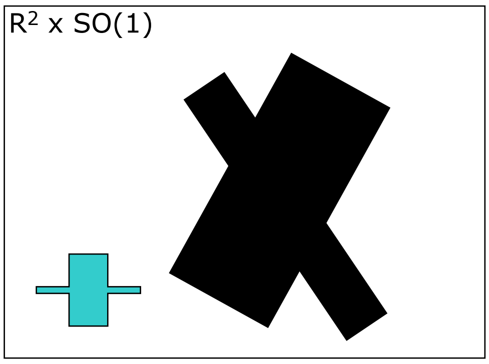
- dim of configuration space = **???**
- Topology **???**



23

Example: rigid robot in 2-D workspace

- dim of configuration space = **???**
- Topology **– cylinder**



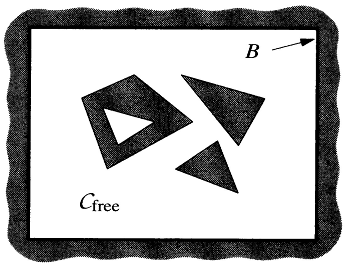
24

Motion Planning

- Planning in continuous state spaces
- Workspace of a robot is a continuous space
- Simplifying assumption (for now) robot is a point

- Given a point robot and a workspace described by polygons. In this case workspace is same as configuration space (more on this later)

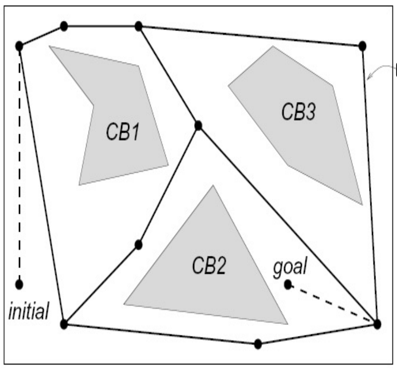
- **Roadmap methods**
  - Visibility graph
  - Cell decomposition
  - Retraction



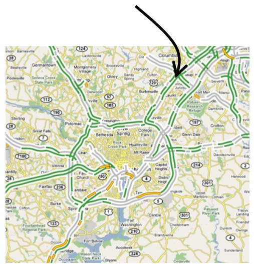
25

Roadmap Methods

Capture the connectivity of C_{free} with a roadmap (graph or network) of one-dimensional curves



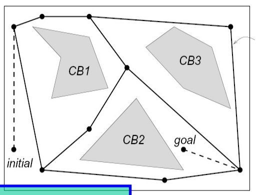
roadmap



26

Roadmap Methods

Path Planning with a Roadmap
Input: configurations q_{init} and q_{goal} , and B
Output: a path in C_{free} connecting q_{init} and q_{goal}



1. Build a roadmap in C_{free} (preprocessing)
• roadmap nodes are free configurations (or semi-free)
• two nodes connected by edge if can (easily) move between them

2. Connect q_{init} and q_{goal} to roadmap nodes v_{init} and v_{goal}

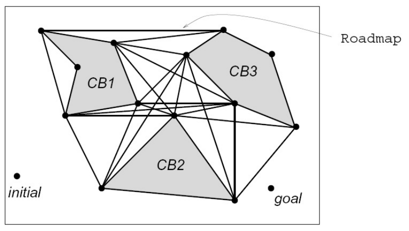
3. Find a path in the roadmap between v_{init} and v_{goal}
- directly gives a path in C_{free}

difficult part

27

Visibility Graph

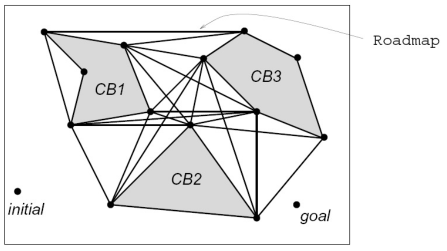
- A visibility graph of C-space for a given C-obstacle is an undirected graph G where
  - nodes in G correspond to vertices of C-obstacle
  - nodes connected by edge in G if
    - they are connected by an edge in C-obstacle, or
    - the straight line segment connecting them lies entirely in *Cfree*
  - (could add q_{init} and q_{goal} as roadmap nodes)



28

Visibility Graph

- Brute Force Algorithm
  - add all edges in C-obstacle to G
  - for each pair of vertices (x, y) of C-obstacle, add the edge (x, y) to G if the straight line segment connecting them lies entirely in cl(C-free) – i.e. robot is allowed to touch the obstacles, but not penetrate them (cl – closure)
    - test (x; y) for intersection with all O(n) edges of C-obstacle
    - O(n^2) pairs to test, each test takes O(n) time

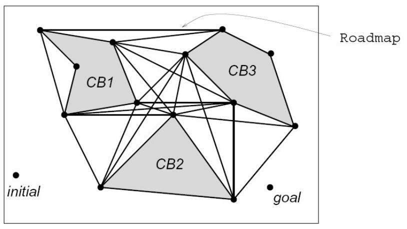


Complexity: O(n^3), n is number of vertices in C-obstacle

29

Visibility Graph

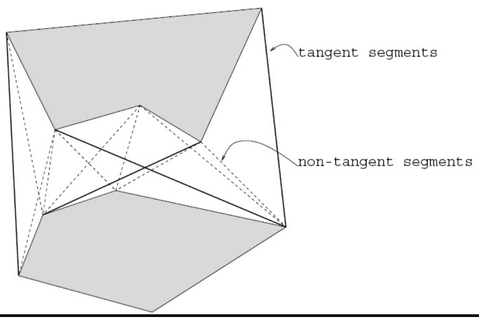
- Visibility graphs – good news
  - are conceptually simple
  - shortest paths (robot is grazing obstacle)
  - we have efficient algorithms if workspace is polygonal
    - *$O(n^2)$,* where n is number of vertices of C-obstacle
    - *$O(k + n \log n)$*, where k is number of edges in G
  - we can make a 'reduced' visibility graph (don't need all edges)



30

Reduced Visibility Graph

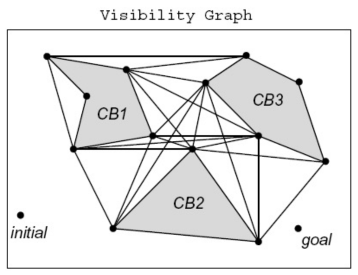
- we don't really need all the edges in the visibility graph (even if we want shortest paths)
- Definition: Let *L* be the line passing through an edge (x; y) in the visibility graph G. The segment (x; y) is a tangent segment *iff* L is tangent to C-obstacle at both x and y.
- Line segment is tangent if extending the line beyond each of the end points would not intersect the obstacles

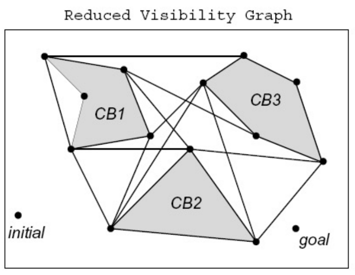


31

Reduced Visibility Graph

- It turns out we need only keep
  - convex vertices of C-obstacle
  - non-CB edges that are tangent segments

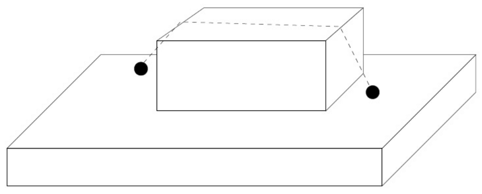




32

Visibility Graph in 3-D

- Visibility graphs don't necessarily contain shortest paths in R^3
  - in fact finding shortest paths in R^3 is NP-hard [Canny 1988]
  - $(1 + \varepsilon^2)$ approximation algorithm [Papadimitriou 1985]

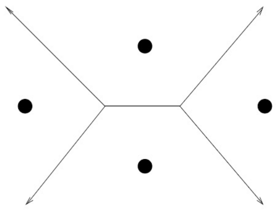


Bad news: Visibility graphs really only suitable for 2D

33

Voronoi Diagram for Point Sets

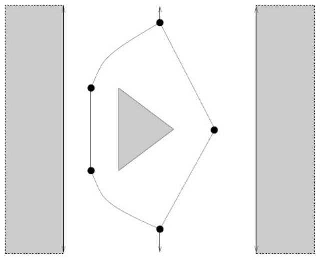
- Voronoi diagram of point set *X* consists of straight line segments, constructed by
  - computing lines bisecting each pair of points and their intersections
  - computing intersections of these lines
  - keeping segments with more than one nearest neighbor
- segments of Vor(*X*) have largest clearance from *X* and regions identify closest point of *X*



37

Voronoi Diagram for Point Sets

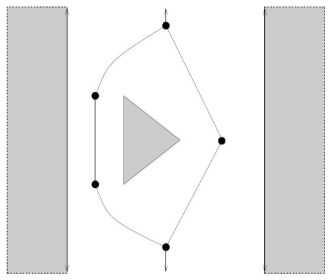
- When C = R^2 and polygonal C-obstacle, Vor(Cfree) consists of a finite collection of straight line segments and parabolic curve segments (called arcs)
  - straight arcs are defined by two vertices or two edges of C-obstacle, i.e., the set of points equally close to two points (or two line segments) is a line
  - parabolic arcs are defined by one vertex and one edge of C-obstacle, i.e., the set of points equally close to a point and a line is a parabola



38

Voronoi Diagram for Point Sets

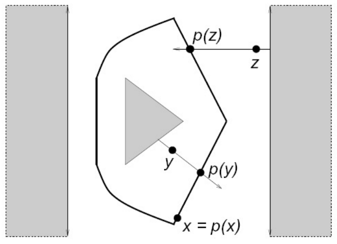
- Naive Method of Constucting V or(Cfree)
  - compute all arcs (for each vertex-vertex, edge-edge, and vertex-edge pair)
  - compute all intersection points (dividing arcs into segments)
  - keep segments which are closest only to the vertices/edges that defined them



39

Retraction

- Retraction $\rho : C_{free} \rightarrow \mathrm{Vor}(C_{free})$



To find a path:
1. compute $\mathrm{Vor}(C_{free})$
2. find paths from q_{init} and q_{goal} to $\rho(q_{init})$ and $\rho(q_{goal})$, respectively
3. search $\mathrm{Vor}(C_{free})$ for a set of arcs connecting $\rho(q_{init})$ and $\rho(q_{goal})$

40

Cell Decomposition

- Idea: decompose C_{free} into a collection K of non-overlapping cells such that the union of all the cells exactly equals the free C-space
- Cell Characteristics:
  - geometry of cells should be simple so that it is easy to compute a path between any two configurations in a cell
  - it should be pretty easy to test the adjacency of two cells, i.e., whether they share a boundary
  - it should be pretty easy to find a path crossing the portion of the boundary shared by two adjacent cells
- Thus, cell boundaries correspond to 'criticalities' in C, i.e., something changes when a cell boundary is crossed. No such criticalities in C occur within a cell.

41

Cell Decomposition

- **Preprocessing:**
  - represent C_{free} as a collection of cells (connected regions of C_{free})
    - planning between configurations in the same cell should be 'easy'
  - build connectivity graph representing adjacency relations between cells
    - cells adjacent if can move directly between them
- **Query:**
  - locate cells k_{init} and k_{goal} containing start and goal configurations
  - search the connectivity graph for a 'channel' or sequence of adjacent cells connecting k_{init} and k_{goal}
  - find a path that is contained in the channel of cells

Difficult

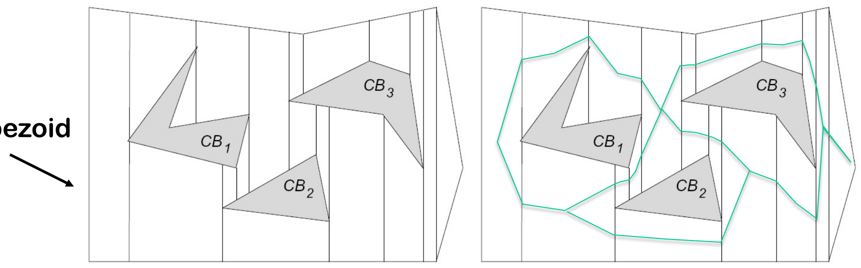
- Two major variants of methods:
  - exact cell decomposition:
    - set of cells exactly covers C_{free}
    - complicated cells with irregular boundaries (contact constraints)
    - harder to compute
  - approximate cell decomposition:
    - set of cells approximately covers C_{free}
    - simpler cells with more regular boundaries
    - easier to compute

42

Trapezoidal Decomposition

- Basic Idea: at every vertex of C-obstacle, extend a vertical line up and down in Cfree until it touches a C-obstacle or the boundary of Cfree

trapezoid



Roadmap – e.g. nodes centroids of cells and centers of vertical lines

43

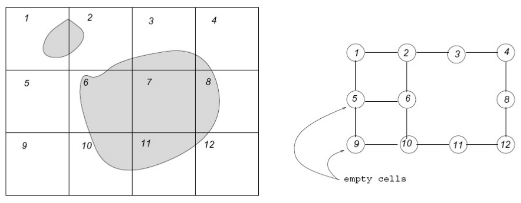
Approx. Cell Decomposition

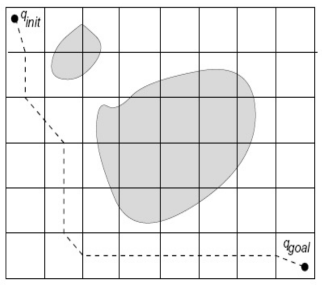
- Construct a collection of non-overlapping cells such that the union of all the cells approximately covers the free C-space!
- Cell characteristics
  - Cell should have simple shape
  - Easy to test adjacency of two cells
  - Easy to find path across two adjacent cells

45

Approx. Cell Decomposition

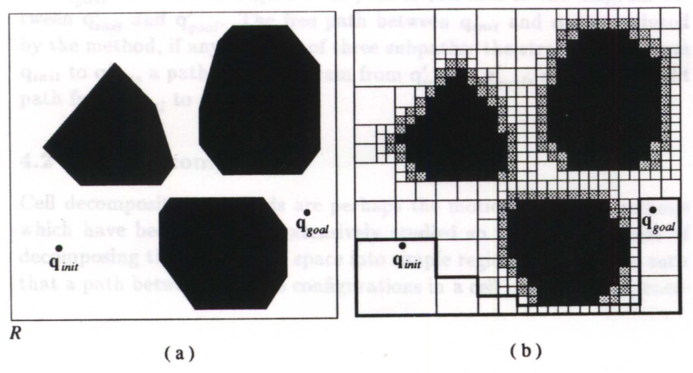
- Each cell is
  - Empty
  - Full
  - Mixed
- Different resolution
  - Different roadmap





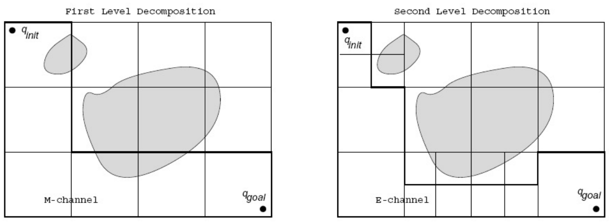
Approx. Cell Decomposition

- Higher resolution around CBs



Approx. Cell Decomposition

- Hierarchical approach
  - Find path using empty and mixed cells
  - Further decompose mixed cells into smaller cells



48

Approx. Cell Decomposition

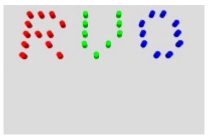
- Advantages:
  - simple, uniform decomposition
  - easy implementation
  - adaptive
- Disadvantages:
  - large storage requirement
  - Lose completeness

- Bottom line 1: We sacrifice exactness for simplicity and efficiency

- Bottom line 2: Approx. cell decomposition methods are practical for lower dimension C, i.e., dof <5, b/c they generate too many cells, i.e. (N^d) cells in d dimension

49

Example: Multiple robots

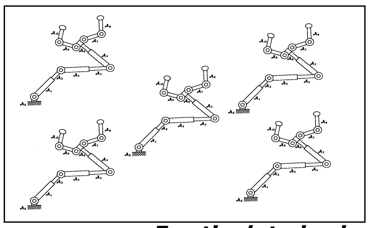


ROV, GAMMA group, UNC

J.J. Kuffner et al.

- Given *n* robots in 2-D
- What are the possible representations?

- What is the number of dofs?
- Cross product of configuration spaces



5 articulated robots

51

Metric in configuration space

- A metric or distance function *d* in a configuration space *C* is a function

  such that

  $$d : (q, q') \in C^2 \rightarrow d(q, q') \geq 0$$

  - $d(q, q') = 0$ if and only if $q = q'$,
  - $d(q, q') = d(q', q)$

  $$d(q,q') \leq d(q,q'') + d(q'',q')$$ aka. Triangle inequality

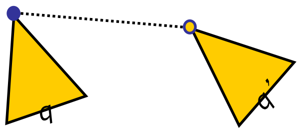
52

Example

- Robot *A* and a point *x* on *A*
- $x(q)$: position of *x* in the workspace when *A* is at configuration *q*
- A distance *d* in *C* is defined by

  $d(q, q') = \max_{x \in A} || x(q) - x(q') ||$

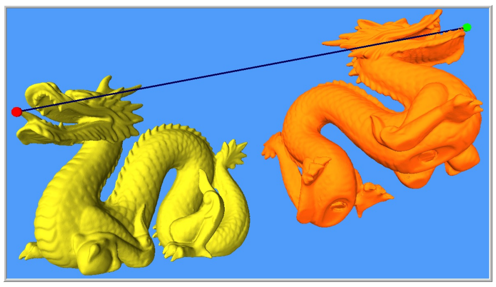
  where $||x - y||$ denotes the Euclidean distance between points *x* and *y* in the workspace.
- Distance between two configurations is the maximum distance between two points of the robot in these two different configurations

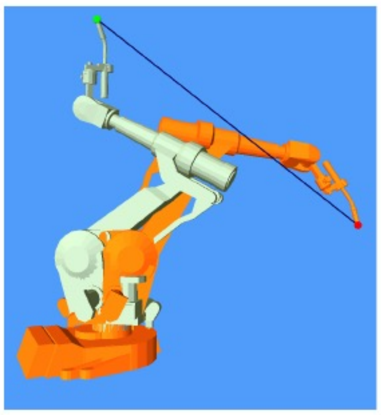


53

Examples

- Maximum distance between the object in two configurations



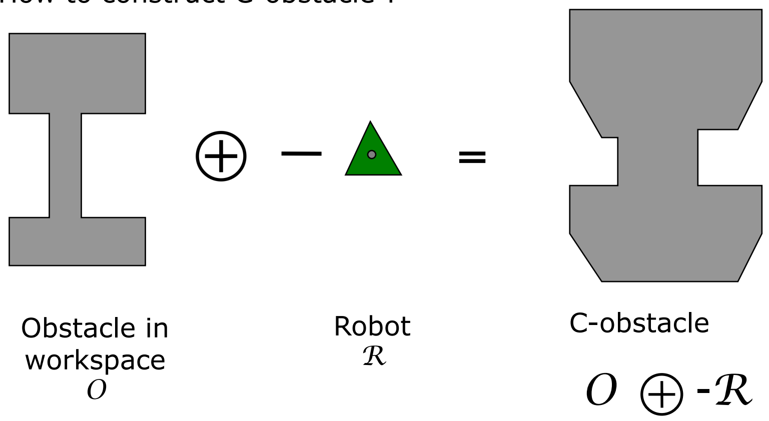


54

C-Space Obstacle

Configuration space obstacle
C-obstacle is (translational motion) $\mathcal{O} \oplus -\mathcal{R}$

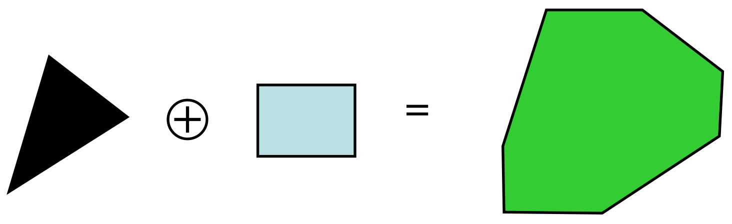
Classic result by Lozano-Perez and Wesley 1979
How to construct C-obstacle ?



56

Minkowski sum of convex polygons

- There is a simple algorithm for computing the boundary
- The Minkowski sum of two convex polygons *P* and *Q* of *m* and *n* vertices respectively is a convex polygon *P* + *Q* of *m* + *n* vertices.
  - The vertices of *P* + *Q* are the “sums” of vertices of *P* and *Q.*



63

Algorithm

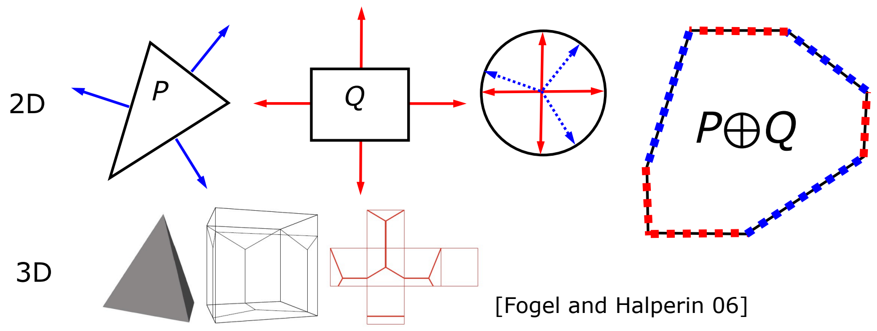
- Sort normals to the edges of the polygon
- Every edge of C_{obst} is either edge of the polygon or edge of the robot. □Every edge is used exactly once, we need to determine the ordering of the edges
- Sort inward angles on the robot counterclockwise
- Sort outward angles of the obstacle normals
- Use incrementally the edges which correspond to the sorted normals in the order they are encountered

- See more details Chapter 4, section 4.3.2 S. Lavalle: Motion Planning.

64

Compute Minkowski Sum

- Convex object
  - Use Gaussian map
  - Compute convex hull of Point-based Minkowski sum (slower)



[Fogel and Halperin 06]

65

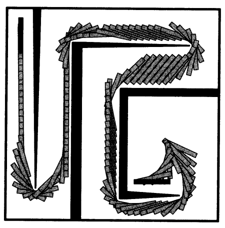
Back to Motion Planning

- Minkowski sum allows us to solve problems with translational robots
- Translational case also generalizes to polyhedra and 3D translations

66

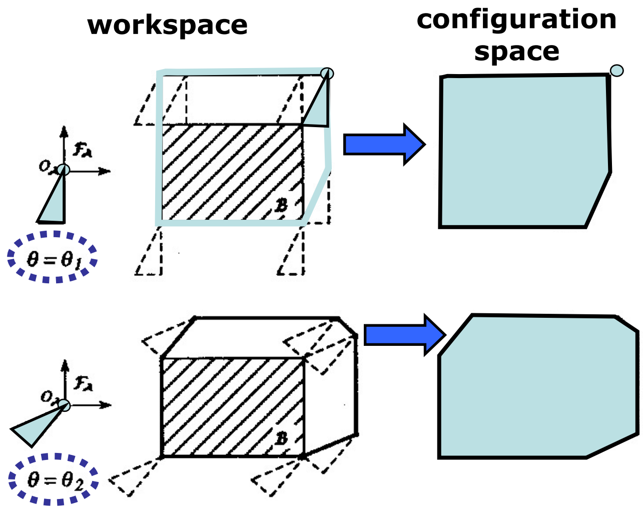
Robot with Rotations

- If a robot is allowed rotation in addition to translation in 2D then it has 3 DOF
- The configuration space is 3D: (x,y,φ) where φ is in the range [0:360)



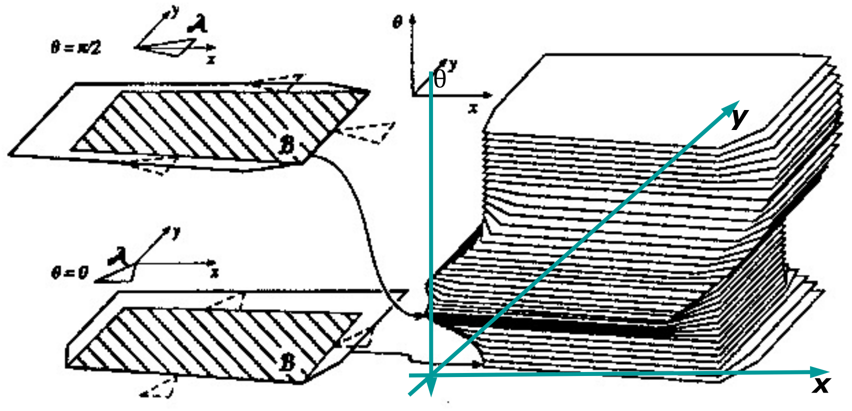
67

Polygonal robot translating & rotating in 2-D workspace



68

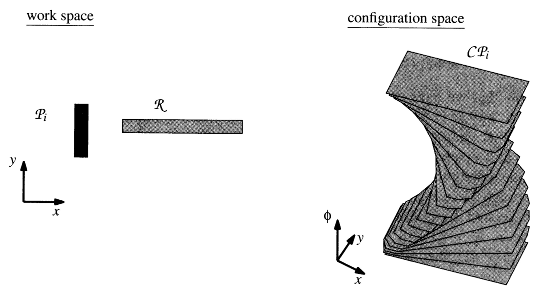
Polygonal robot translating & rotating in 2-D workspace



69

Mapping to C-Space

- The obstacles map to “twisted pillars” in C-Space
- They are no longer polygonal but are composed of curved faces and edges



70

Computing Free Space

- Exact cell decomposition in 3D is really hard
- Compute z: a finite number of slices for discrete angular values
- Each slice will be the representation of the free space for a purely translational problem
- Robot will either move within a slice (translating) or between slices (rotating)

71

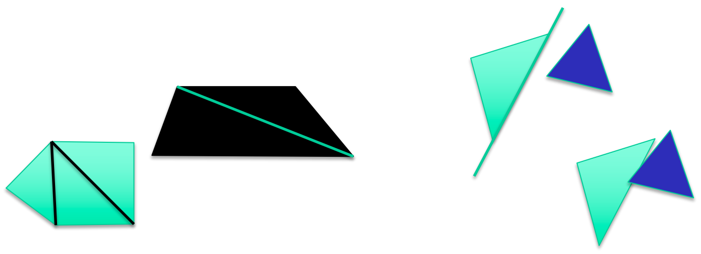
Hard Motion Planning

- Configuration Space methods – complex even for low dimensional configuration spaces
- Plus – always guarantee finding a plan if it exists in finite time (or answer no)
- Idea behind sampling cased motion planning – sacrifice completeness for efficiency – weaker guarantee – notion on probabilistic completeness

72

Collison Checking

- Obstactles defined implicitely via function
- ColisionCheck (x) -> 0 if the configuration does not yield collision -> 1 is the configuration yields collision
- How to check for collisions ?
- Consider robot and obstacles are made of convex shapes – i.e. triangles – make sure that none of robot triangles intersects obstacle triangles



Check if one of the triangle sides
Is separating line – generalizes to 3D (separating planes)

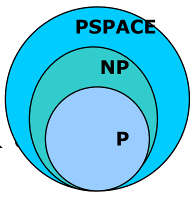
73

Building Roadmap in Config Space

- This can be used to make a discrete approximation of the configuration space
- Sample the configurations uniformly and tightly
- Check for each config. if it is in free space
- Connect neighbouring configurations in a graph

74

The Complexity of Motion Planning



- General motion planning problem is
- PSPACE-hard [Reif 79, Hopcroft et al. 84 &
- PSPACE-complete [Canny 87]

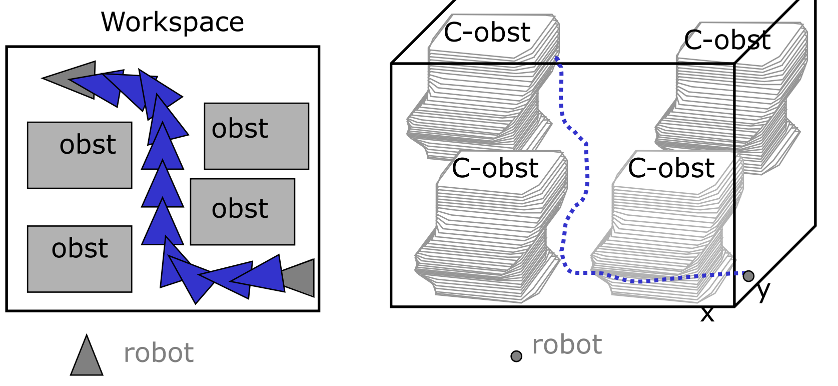
The best deterministic algorithm known has running time that is exponential in the dimension of the robot's C-space [Canny 86]

- C-space has high dimension - 6D for rigid body in 3D space
- simple obstacles have complex C-obstacles impractical to compute explicit representation of free space for more than 4 or 5 dof

75

Motion Planning in C-space

Simple workspace obstacle transformed
Into complicated C-obstacle!!



Path is swept volume

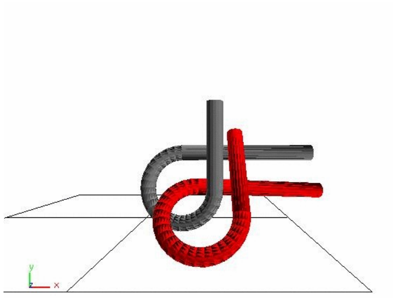
Path is 1D curve

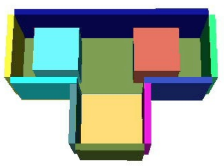
76

Hard Motion Planning Problems

The Alpha Puzzle

Swapping Cubes Puzzle





- Separate two shapes (one considered robot) – another obstacle
- Exchange the positions of two cubes
 (one needs move to empty space)
- All these planning problems are considered in continuous spaces

77

Hard Motion Planning Problems Highly Articulated (Constrained) Systems

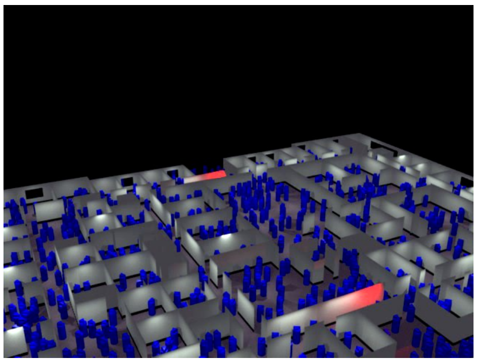
Collision-free reaching
for object manipulation
grasping objects
with right or left hand

Reaching and grasping

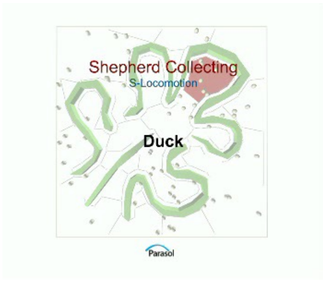
79

Hard Motion Planning Problems Flocking: Covering, Homing, Shepherding

Motion for coordinated entities



Interactive Navigation of Multiple Agents in Crowded Environments. Jur van den Berg, Sachin Patil, Jason Sewall, Dinesh Manocha, Ming Lin, i3D 2008

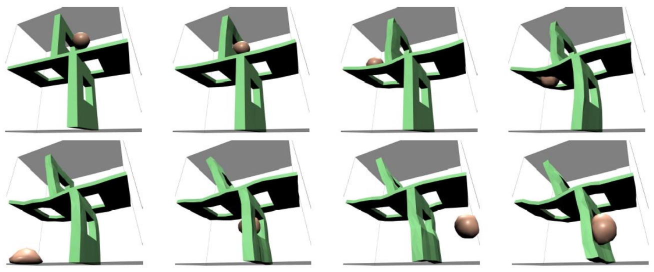


Control the motion of coordinated entities

80

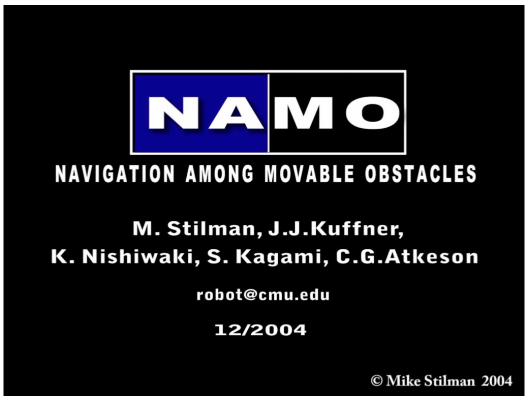
Hard Motion Planning Problems
Deformable Objects

- Find a path for a deformable object that can deform to avoid collision with obstacles
- move a mattress in a house, elastic or air-filled objects, metal sheets or long flexible tubes
- virtual surgery applications
- computer animation and games
- Issue: difficult to find natural deformation efficiently



81

Hard Motion Planning Problems
Movable Objects


NAMO

NAVIGATION AMONG MOVABLE OBSTACLES

M. Stilman, J.J.Kuffner,
K. Nishiwaki, S. Kagami, C.G.Atkeson

robot@cmu.edu

12/2004

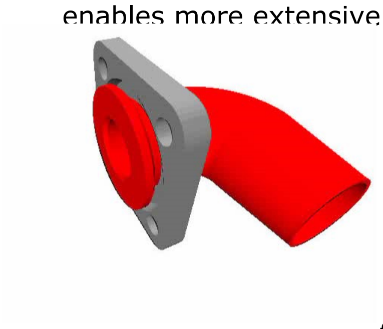
© Mike Stilman 2004

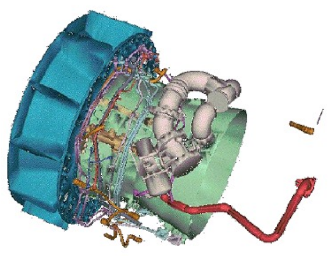
- **M. Stilman and J.J. Kuffner** Planning Among Movable Obstacles with Artificial Constraints Workshop on the Algorithmic Foundations of Robotics, July, 2006

82

Hard Motion Planning Problems Intelligent CAD Applications

- **<u>Using Motion Planning to Test Design Requirements</u>**
  - Accessibility for servicing/assembly tested on physical "mock ups"
  - Digital testing saves time and money, is more accurate, enables more extensive testing, and is useful for training (VR


flange

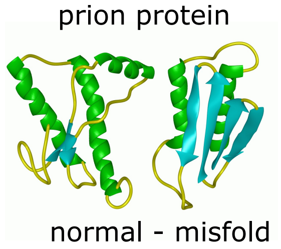

Airplane engine

Maintainability Problems:
Mechanical Designs from GE

83

Hard Motion Planning Problems computational biology & chemistry

- Motion of molecules
- help understand important interactions - protein structure/function prediction
- diseases such as Alzheimer's and Mad Cow are related to misfolded proteins

prion protein

normal - misfold

84

Probabilistic Methods

- Resort to sampling based methods
- Avoid computing C-obstacles
  - Too difficult to compute efficiently

- Idea: Sacrifice completeness to gain simplicity and efficiency

- Probabilistic Methods
  - Graph based
  - Tree based

85

Sampling Based Motion Planning

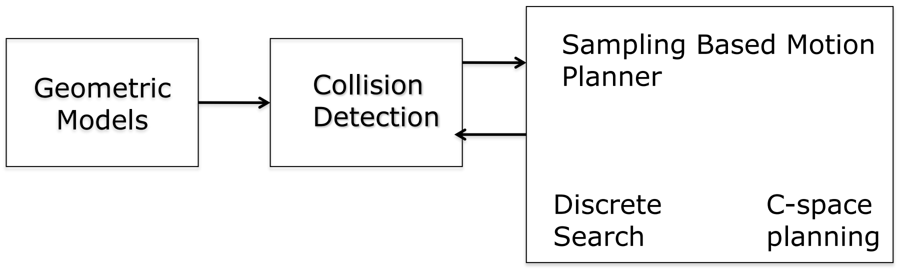
- Recall: Algorithm is considered complete if for any input it correctly reports the path if it exists in finite amount of time

- Sampling based methods cannot achieve completeness

- Deterministic approach which samples densely is called Resolution complete
- Random Sampling Based Methods
  Probabilistically complete with enough samples the probability of finding solution approaches 1

86

Sampling Based Motion Planning

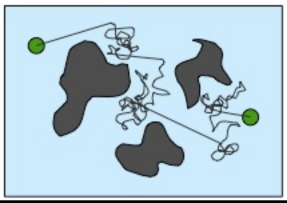


Idea : Generate random configurations
Check whether they are collision free
Connect them using Local planners
Discrete Search: (q0, qG) – single query search until you find qG
Multi-query search: Rapidly Exploring Random Trees

87

Probabilistic Motion Planning

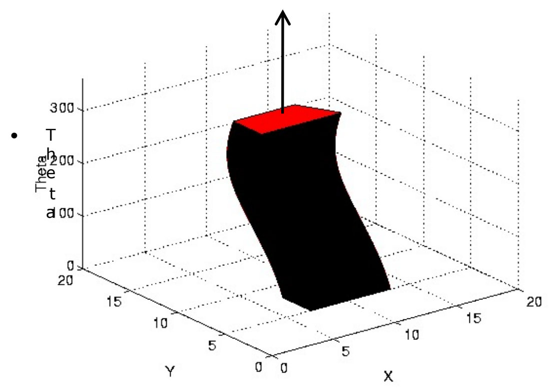
- First encounter with randomized techniques – in the context of potential field based methods
- Use random walk to escape local minima (can take long time)
- Idea – potential function gives as a cost to go g(q)
- If local planner is not successful reducing the cost to go
- Switch to random walk mode from current node, terminate if node with lower g(q) is found or number of iterations have been reached
- If better node has not been found back-track – pick one of the nodes encountered in Random walk and restart best first search



88

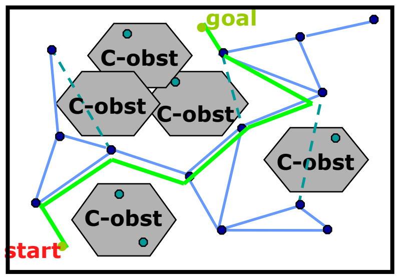
Probabilistic Roadmap Method
[Kavraki, Svestka, Latombe,Overmars 1996]

Explicit representation of the configuration space is unknown



Probabilistic Roadmap Method

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., straight line)
   - discard paths that are invalid

Query processing

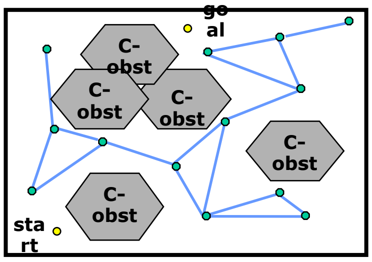
1. Connect *start* and *goal* to roadmap
2. Find path in roadmap between *start* and *goal*
   - regenerate plans for edges in roadmap

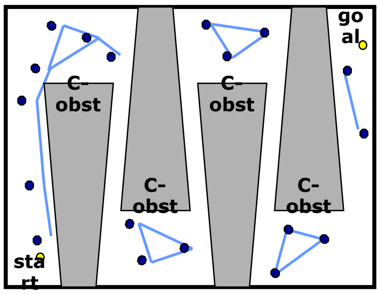
Probabilistic Roadmap Method

- Important sub-routines
  - Generate random configurations
  - Local planners
  - Distance metrics
  - Selecting k-nearest neighbors (becoming dominant in high dimensional space)
  - Collision detection (>80% computation)

91

PRMs: Pros & Cons





PRMs: The Good News

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: The Bad News

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

92

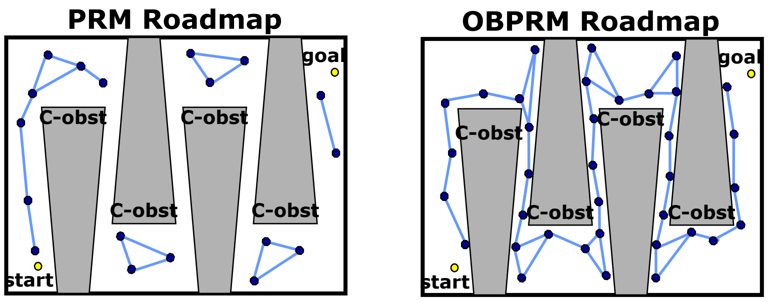
Related Work (selected)

- **Probabilistic Roadmap Methods**
- Uniform Sampling (original) [Kavraki, Latombe, Overmars, Svestka, 92, 94, 96]
- Obstacle-based PRM (OBPRM) [Amato et al, 98]
- PRM Roadmaps in Dilated Free space [Hsu et al, 98]
- Gaussian Sampling PRMs [Boor/Overmars/van der Steppen 99]
- Bridge test [Hsu et al 03]
- Visibility Roadmaps [Laumond et al 99]
- Using Medial Axis [Kavraki et al 99, Lien/Thomas/Wilmarth/Amato/Stiller 99, 03, Lin et al 00]
- Generating Contact Configurations [Xiao et al 99]
- Using workspace clues

93

An Obstacle-Based PRM

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

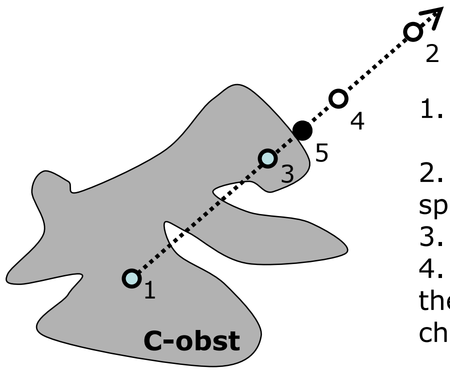


Idea: Can we sample nodes near C-obstacle surfaces?
- we cannot explicitly construct the C-obstacles...
- we do have models of the (workspace) obstacles...

94

Finding Points on C-obstacles

Basic Idea (for workspace obstacle S)

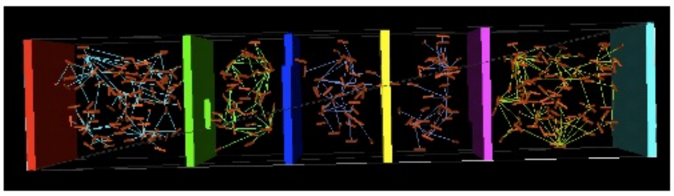


1. Find a point in S’s C-obstacle (robot placement colliding with S)
2. Select a random direction in C-space
3. Find a free point in that direction
4. Find boundary point between them using binary search (collision checks)

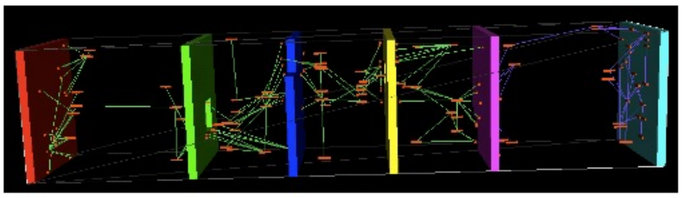
Note: we can use more sophisticated heuristics to try to cover C-obstacle

95

OBPRM



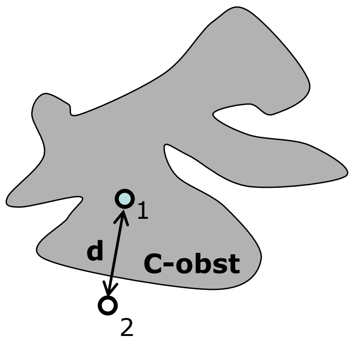
PRM
- 328 nodes
- 4 major CCs



OBPRM
- 161 nodes
- 2 major CCs

96

Gaussian Sampling PRM



1. Find a point in S's C-obstacle (robot placement colliding with S)

2. Find another point that is within distance *d* to the first point, where d is a random variable in a *Gaussian distribution*

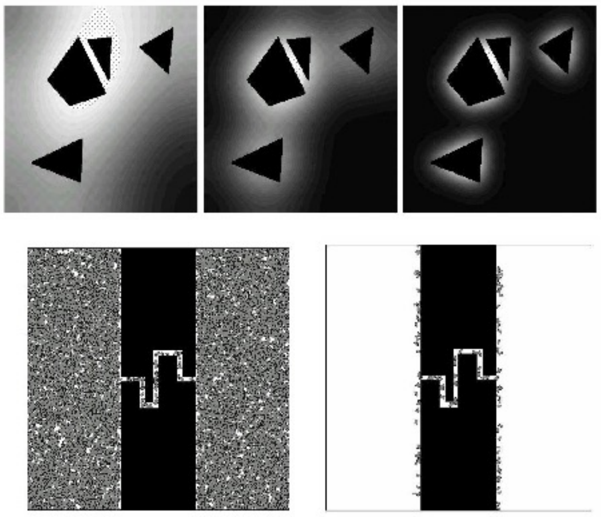
3. Keep the second point if it is collision free

Note

- Two paradigms: (1) OBPRM: Fix the samples (2) Gaussian PRM: Filter the samples
- None of these methods can (be proved to) provide guarantee that the samples in the narrow passage will increase!

97

Gaussians



98

Issues

- How do we determine a random free configuration
- We would like to sample nodes uniformly from C_{free}
- Draw each of the coordinates from the interval of corresponding DOF (use uniform probability per interval)
- For each sample check for collision between the robot and obstacles and robot itself
- If collision free add to V otherwise discard
- Collision detection and sampling – large topics

100

Collision Detection

- Treated as black box - takes most of the computation
- In 2D convex robot and obstacle, there exist linear time collision detection algorithms
- Construct polygonal C_{obst}
- Define a logical predicate which indicates whether configuration is free or not
- Hierarchical Methods or Incremental Methods
- Section 5.3.4 Motion Planning Book

101

Planning in high dimensional spaces

- Single query planning $q_{initial}$ and q_{goal} are given once – no pre-computation (greedy search technique can take a long time)
- Multiple query planning – spreads out uniformly, requires lot of samples to cover the space
- Next incremental sampling and search methods that yields good performance without parameters tunning. Idea gradually construct search tree, such that it densely covers the space

104

Incremental Sampling and Searching

- Single query model – given start and goal q find a path
- Analogy with the discrete search algorithms
- Samples are states, edges are paths connected them (as opposed to actions previously)
- Graphs are undirected; Ingredients

1. Initialize the graph
2. Select vertex for expansion
3. Generate set of new vertices
4. For some new vertices run a local planner and check whether its collision free
5. If yes insert an edge to the graph
6. Keep on going until termination condition is satisfied

105

Incremental Search and Sample

- Why not just discretizing configuration space ?
- For high dimensions large number of states can be wasted exploring various cavities of the C-space
- For low dim spaces grid points themselves can serve as roadmap points (need to be checked for collisions etc)

- How to choose a resolution of the discretization
  (start coarse , iteratively refine)
- Another option – abandon discretization and work with continuous problem (like randomized potential fields) or RRT’s

106

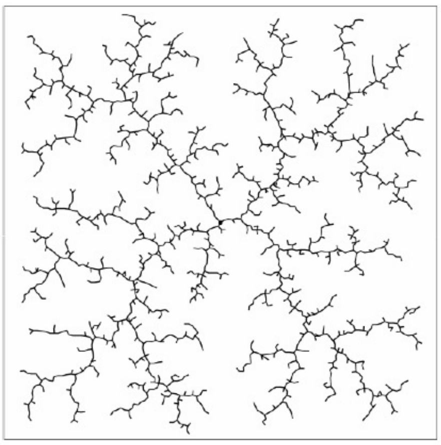
Rapidly-Exploring Random Tree (RRT)

- Tree Based single shot planners – compute the respresentation of C_{free} for single start and goal

- RRTs: Rapidly-exploring Random Trees
  Rapidly-exploring random trees: Progress and prospects. S. M. LaValle and J. J. Kuffner. In *Proceedings Workshop on the Algorithmic Foundations of Robotics*, 2000.)
  Incrementally builds the roadmap tree
- Extends to more advanced planning techniques
  –Integrates the control inputs to ensure that the kinodynamic constraints are satisfied

107

Rapidly-Exploring Random Trees



Idea: Incrementaly construct the search tree, that improves with resolution
Previous incremental search methods could spend long time exploring nodes inside unimportant cavities

108

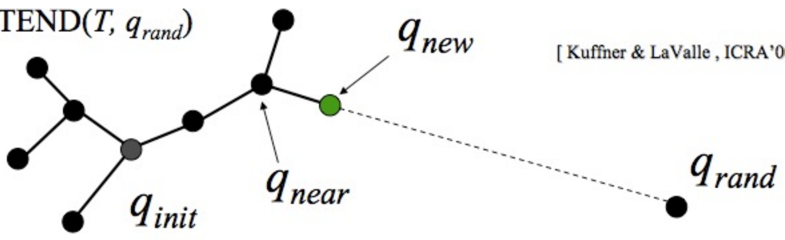
RRT's

```
BUILD_RRT ($q_{init}$) {
  $T.init(q_{init})$;
  for $k$ = 1 to K do
    $q_{rand}$ = RANDOM_CONFIG();
    EXTEND($T$, $q_{rand}$)
}
```

Random sample connects to the nearest node so far

If the nearest point lies on an edge, the edge is split in two

EXTEND(T, q_{rand})



[Kuffner & LaValle , ICRA'00]

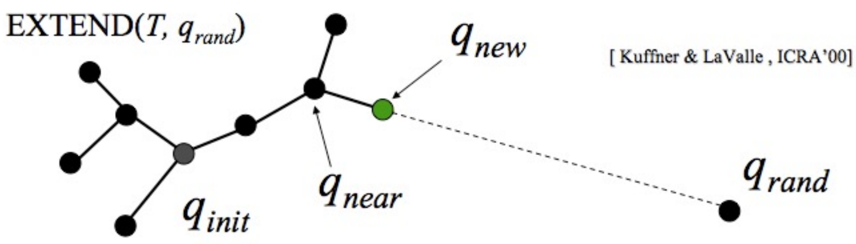
109

RRT's

```
BUILD_RRT ($q_{init}$) {
  $T.init(q_{init})$;
  for $k$ = 1 to K do
    $q_{rand}$ = RANDOM_CONFIG();
    EXTEND($T$, $q_{rand}$)
}
```

Details:
Step length: how far to sample
Sample just at the end point
Sample all along, small steps

Extend returns the new edge



110

Naïve Random Tree

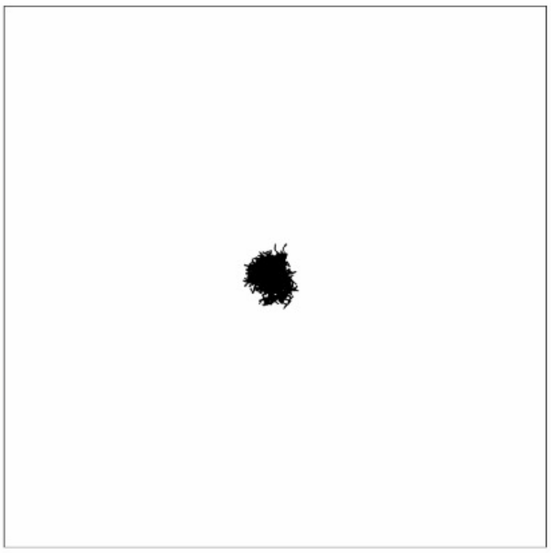
Start with middle

Sample near this node

Then pick a node at random in tree

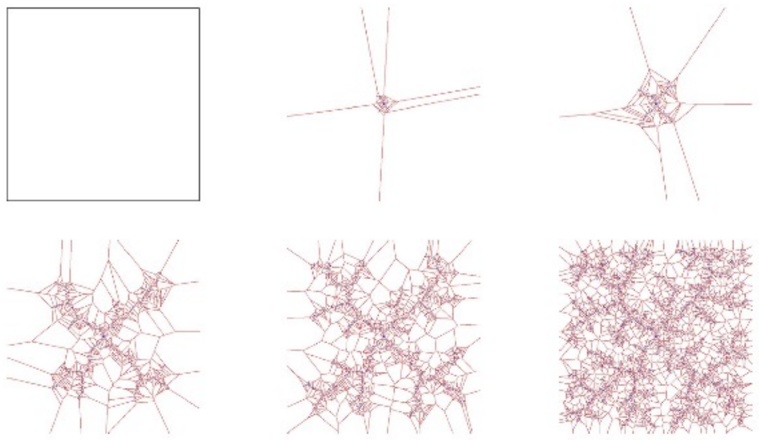
Sample near it

End up Staying in middle



111

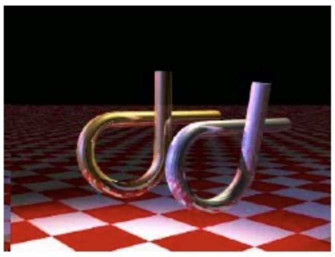
RRT’s are biased towards large Voronoi cells

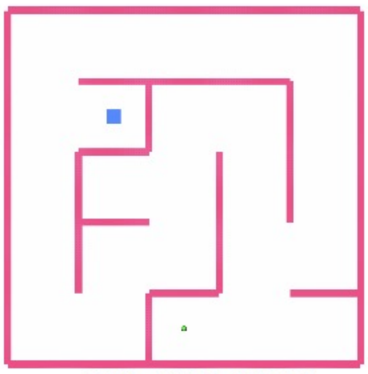


The nodes most likely to be closest to a randomly chosen point in state space are those with the largest Voronoi regions. The largest Voronoi regions belong to nodes along the frontier of the tree, so these frontier nodes are automatically favored when choosing which node to expand.

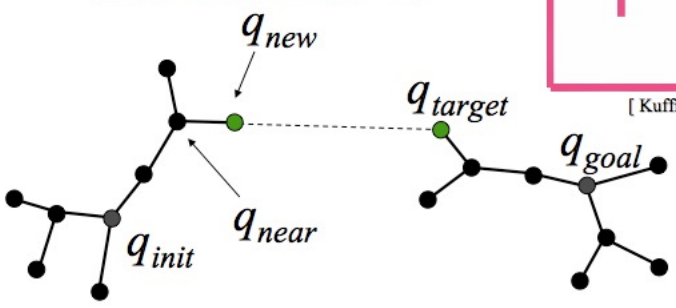
112

Grow two RRT’s together

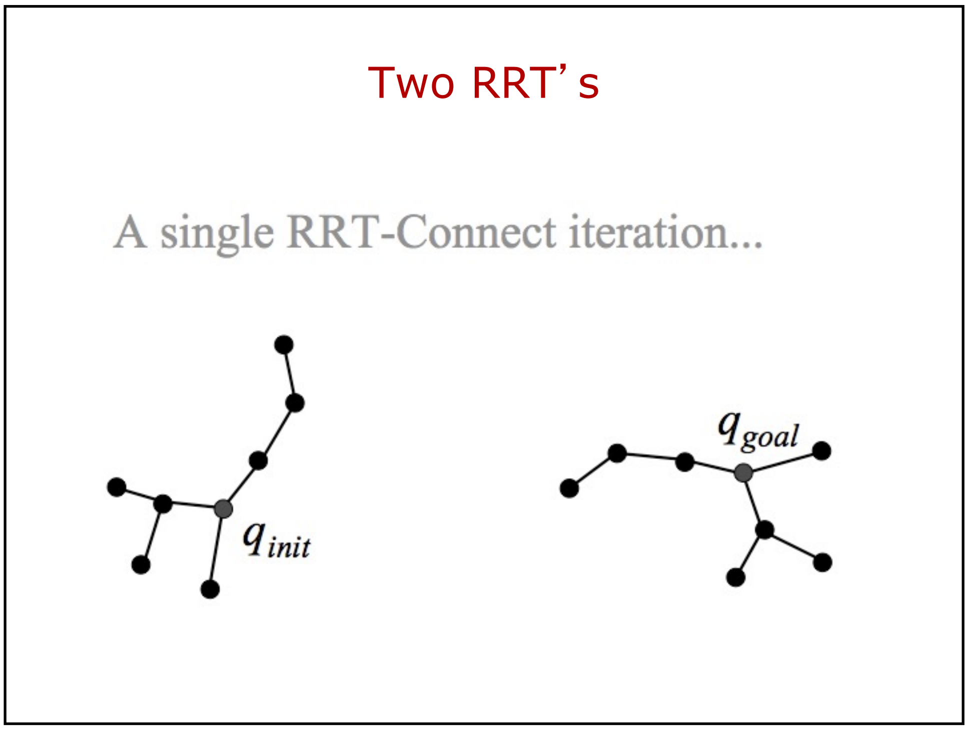




[Kuffner, LaValle ICRA ‘00]

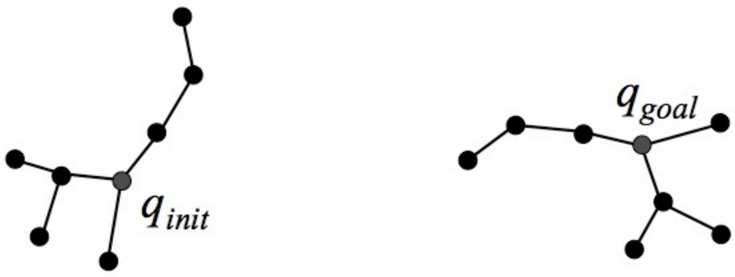


113

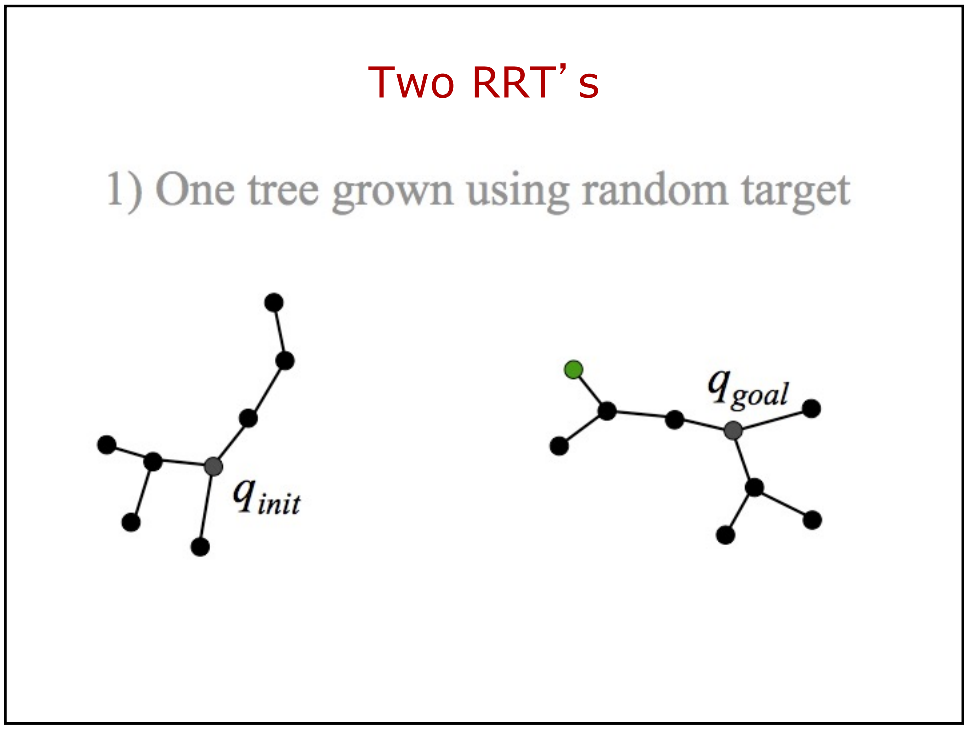


Two RRT's

A single RRT-Connect iteration...

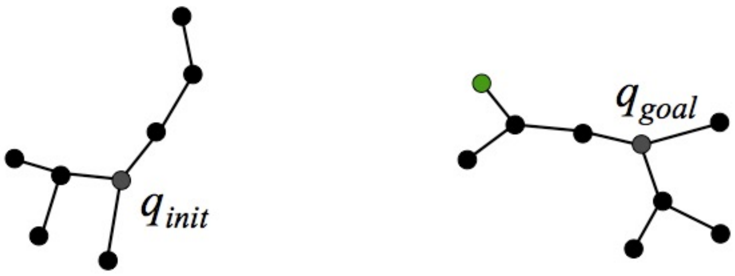


114



Two RRT's

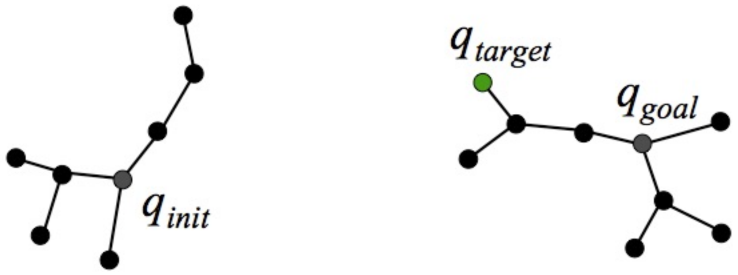
1) One tree grown using random target



115

Two RRT’s

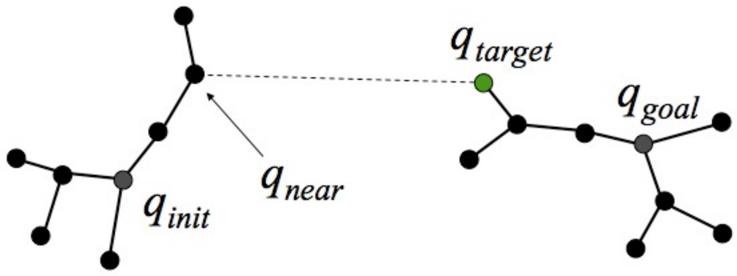
2) New node becomes target for other tree



116

Two RRT’s

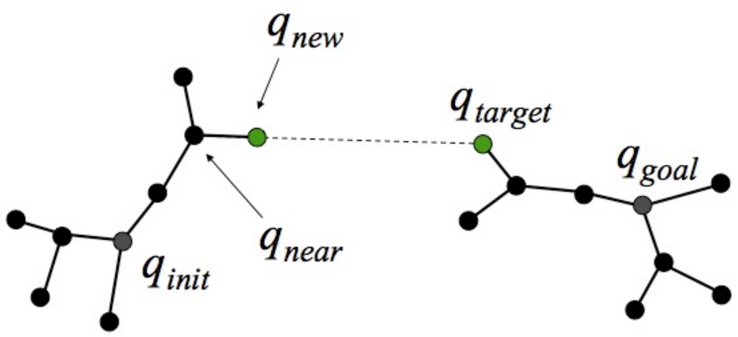
3) Calculate node “nearest” to target



117

Two RRT's

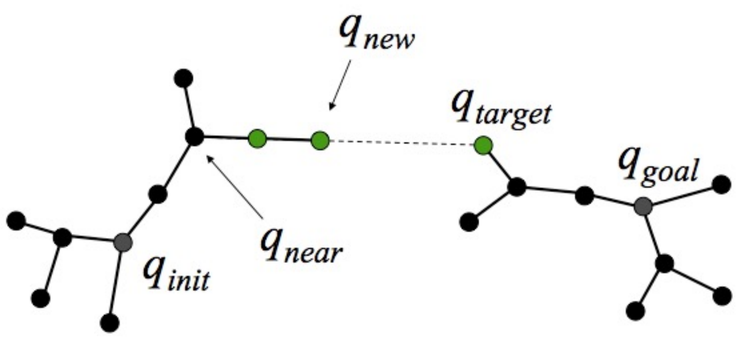
4) Try to add new collision-free branch



118

Two RRT's

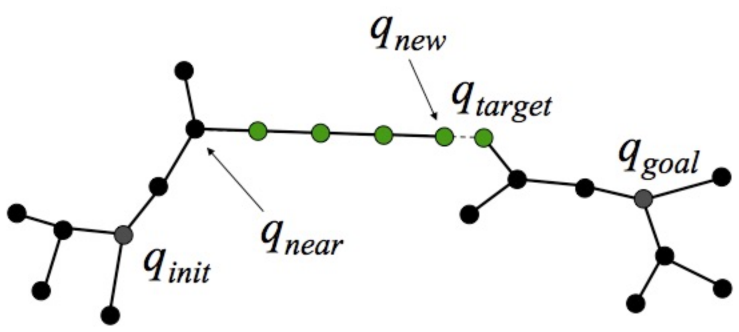
5) If successful, keep extending branch



119

Two RRT' s

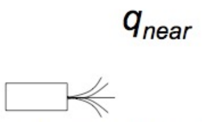
5) If successful, keep extending branch



120

Taking actions into account

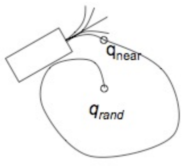
Instead of moving in a straight line for some distance, take into account kinematic constraints

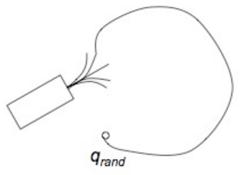


$q' = f(q,u)$ - - - use action u from q to arrive at q'

chose $u_* = \arg\min(d(q_{rand}, q'))$

Is this the best?

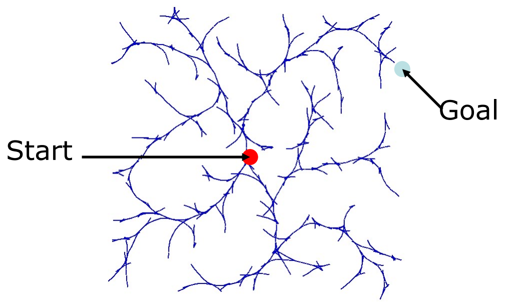




121

How it Works

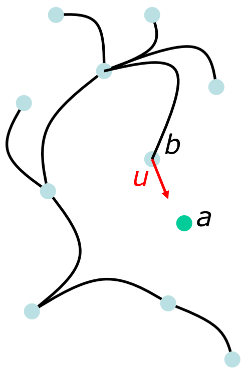
- Build a rapidly-exploring random tree in state space (X), starting at s_{start}
- Stop when tree gets sufficiently close to s_{goal}



122

Building an RRT

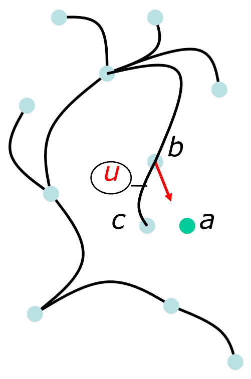
- To extend an RRT:
  - Pick a random point a in X
  - Find b, the node of the tree closest to a
  - Find control inputs u to steer the robot from b to a



123

Building an RRT

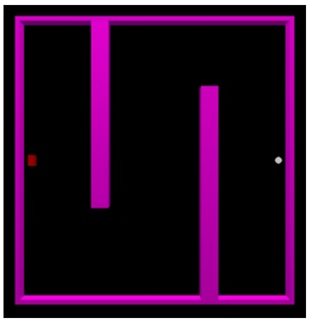
- To extend an RRT (cont.)
  - Apply control inputs *u* for time δ, so robot reaches *c*
  - If no collisions occur in getting from *a* to *c*, add *c* to RRT and record *u* with new edge

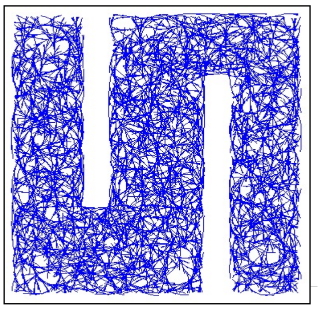


Executing the Path

- Once the RRT reaches s_{goal}
  - Backtrack along tree to identify edges that lead from s_{start} to s_{goal}
  - Drive robot using control inputs stored along edges in the tree

Problem of Simple RRT Planner



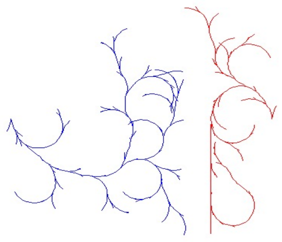


- Problem: ordinary RRT explores *X* uniformly
  → slow convergence
- Solution: bias distribution towards the goal – once in a while choose goal as new random configuration (5-10%)
- If goal is choosen 100% time then it is randomized potential planner

126

Bidirectional Planners

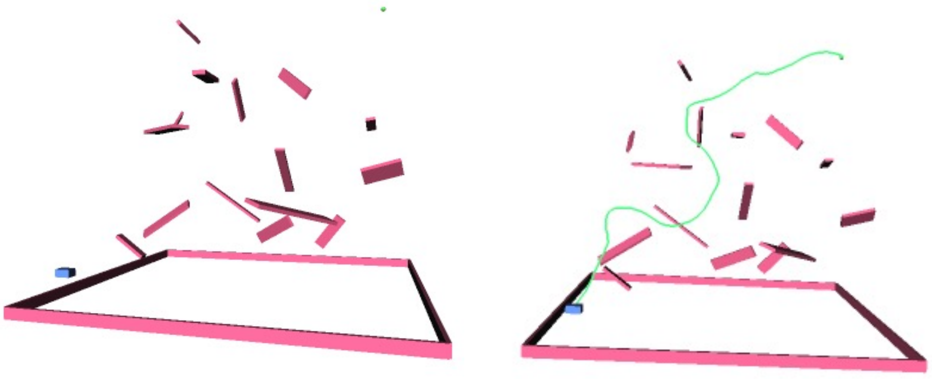
- Build two RRTs, from start and goal state



- Complication: need to connect two RRTs
  - local planner will not work (dynamic constraints)
  - **bias** the distribution, so that the trees meet

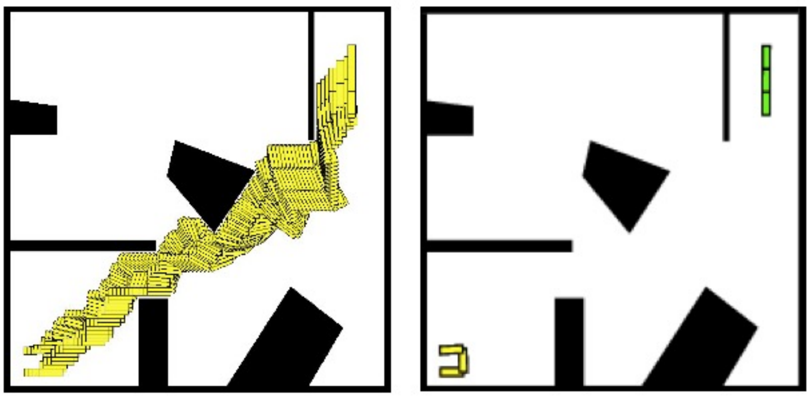
127

Bidirectional RRT Example



128

Articulated Robot example



129

RRT' s

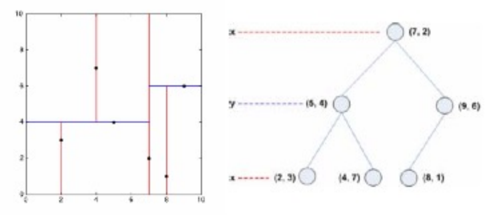
- Link
- http://msl.cs.uiuc.edu/rrt/gallery.html

- Issues/problems
- Metric sensitivity
- Nearest neighbour efficiency
- Optimal sampling strategy
- Balance between greedy search and exploration

- Applications in mobile robotics, manipulation, humanoids, biology, drug design, areo-space, animation
- Extensions – real-time RRT' s, anytime RRT' s dynamic domains RRT' sm deterministic RRTs, hybrid RRT' s

130

Efficient nearest neighbour algorithms



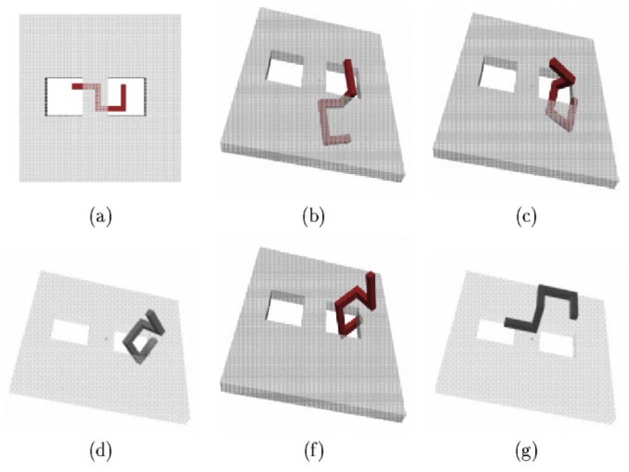
- How to find NN in high dimensional spaces

- KD trees – recursively choose a plane P that splits the set
  evenly in a coordinate direction
- Store P at the node
- Apply to children sets Sl and Sr
- Requires O(dn) storage

- Various hashing strategies

131

Computed example



138

Conclusion

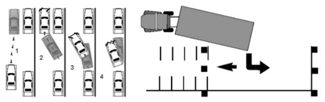
- Motion planning is difficult (intractable)
- Roadmap methods
  - Probabilistic Motion Planners

We will return to planning when considering partial information, dynamically changing worlds, uncertainty

139

What is not covered?

- Other types of motion planning
  - With constraints
    - Close-chain constraint
    - Nonholonomic constraint
    - Differential constraints
  - Manipulate planning
  - Assembly planning
  - Planning with uncertainty
  - Planning for multiple robots, dynamic env
  - Planning for highly articulated objects
  - Planning for deformable objects
  - …

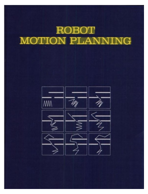


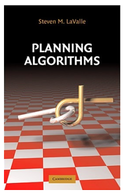
Little Seiko

140

Additional Readings

- **Gross motion planning—a survey**, Y. K. Hwang and N. Ahuja, ACM Computing Surveys, 1992 (survey paper)
- **Robot Motion Planning**. J.C. Latombe. Kluwer Academic Publishers, Boston, MA, 1991.
- **Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts**. Jean-Claude Latombe, IJRR, 1999 (survey paper)
- **Planning Algorithms**, Steven LaValle, 2006, Cambridge University Pres, (Free download at http://planning.cs.uiuc.edu/)





141

Examples

- Road Map methods and behavior based strategies
- Homing https://parasol.tamu.edu/dsmft/movies/flocking_Homing_web.mpg
- Flocking, Goal Search

- https://parasol.tamu.edu/dsmft/movies/flocking_GoalSearch_web.mpg
- https://parasol.tamu.edu/dsmft/movies/flocking_RBFlock_narrow_homing.mpeg

142