Thursday, April 09, 2009
12:32 PM

5.3 Collision Detection

In most (geometric) M.P. problems, most computation time is spent checking sampled configurations for collisions

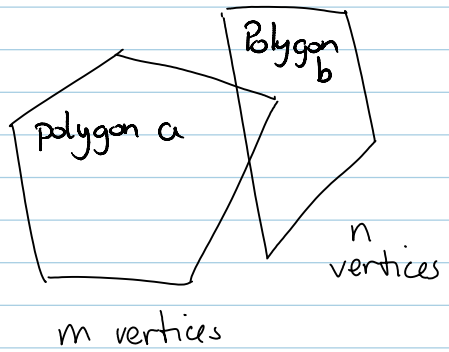
This is not true for dynamic systems with complex constraints.

Planar Case:

In the case of a convex polygonal robot & obstacles, collision checking can be done in linear time in the geometric complexity (i.e., the # of edge-vertex pairs).

a bit misleading terminology

geometric complexity of two convex polygons is 2mn



Recall: Collision checking approach of section 4.3 is $O(mn)$.

Spatial Case:

Straight forward extension to 3D polyhedra is again linear in the # of geom. primitives. The # of geometric primitives is:

$$F_a V_b + E_a E_b + V_a F_b ,$$

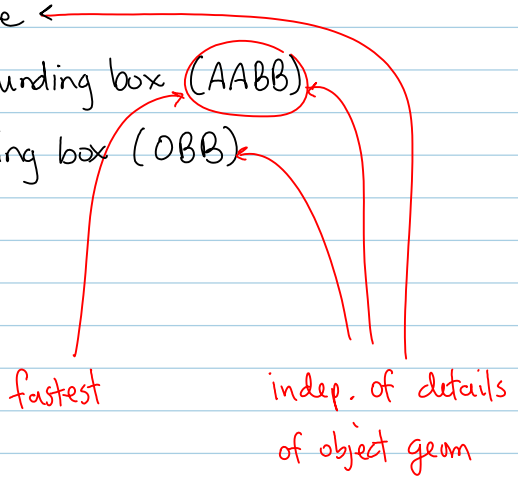
where F_a, E_a, V_a are the #'s of Faces, Edges, and vertices of polyhedron "a", and F_b, E_b, V_b are #'s for polyhedron "b".

Collision checking

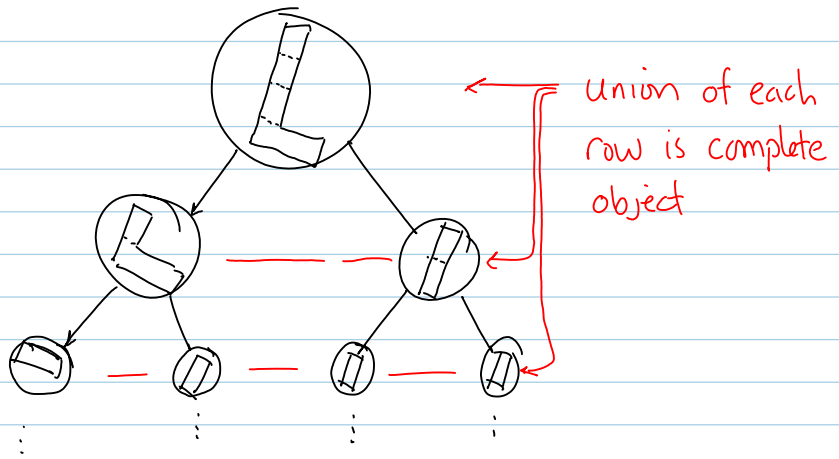
We just want a boolean result: yes, there is collision
or no, there is not

Use hierarchical tree representation of objects.

- root node is whole body
- leaf nodes are subsets w/ limited # of geom prims.
- other nodes are subsets of bodies
- each node's geometry is bounded by a simpler geom.
  - bounding sphere
  - axis-aligned bounding box (AABB)
  - oriented bounding box (OBB)
  - convex hull

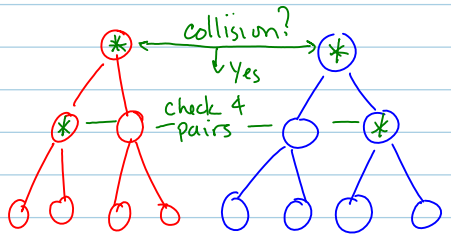


Example using spheres (in 2D):



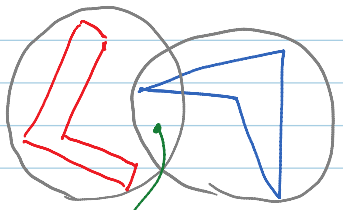
You could keep going until each leaf contains one convex polygon but you might stop with n triangles or geometric

entities per leaf



Now consider collision checking with two bodies (two trees):

1. Root level test:
   Suppose the root nodes of E & F don't intersect. Then bodies E & F do not intersect.
   Done.

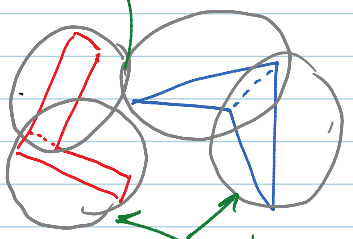


overlapping, so descend tree

2. If E & F roots intersect, then descend the trees one level

overlap again, so descend again.

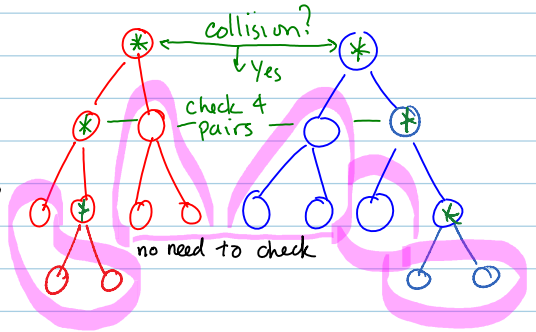
3. Continue recursively until no collisions occur at one level OR we reach the leaves.



no overlap, so don't consider their children.

4. At the leaves, collision check with geometric primitives

- Prune if bounding volumes don't overlap
- Might not have to descend to the leaves



Main advantage:
When bodies are far apart, very little work needs to be done.

What situations will not benefit from this decomposition?

Tight-fitting parts:

Medical catheterizations, assemblies

Other questions:

How do you balance the tree?

How do you decompose the object? most quickly shrinking bounding volume?

Should the tree be binary?

Is hierarchical geometric decomposition useful in distance comp?

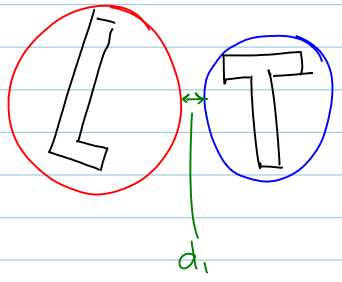
Why do I care about the last question?

Dynamics! We need ψ_n^l if $\psi_n^l <$ tolerance including $\psi_n^l < 0$ to form timestepping subproblems.

Modifications for distance computations. Does straight-forward extension work?

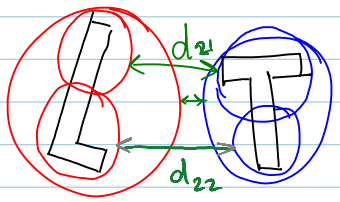
1. Compute distance between root bounding geometries first

Exact distance must be greater than or equal to d_1.



2. Descend to second level.

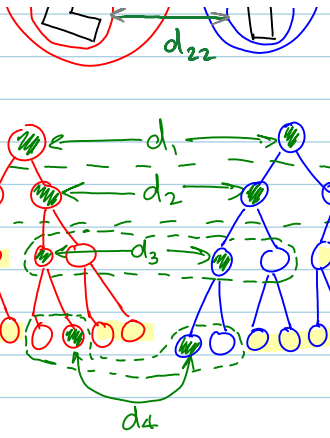
Compute distances between pairs of subsets $\Rightarrow d_2$



Let $d_2 = \min(d_{21}, d_{22})$

Can we prune

pairs of subsets → d_2

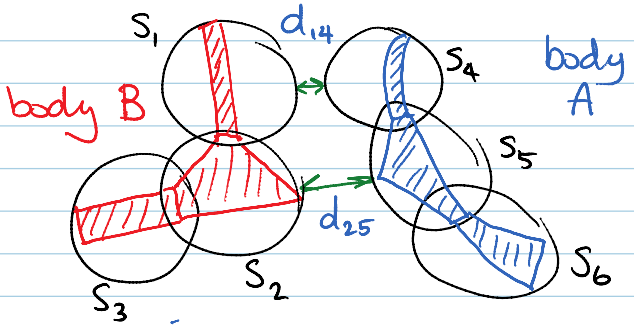


Can we prune like this ?

- Unlike the problem of C.D., to obtain distance we must recurse to the leaves!
- Expensive distance computations must be performed!
- Pruning is NOT as simple!

Actual distance is d_{25}

$d_{25} > d_{14}$, so can't prune S_2, S_3, S_5, S_6



Possible condition for pruning with bounding spheres:

Let r_i be radius of i^{th} bounding sphere.

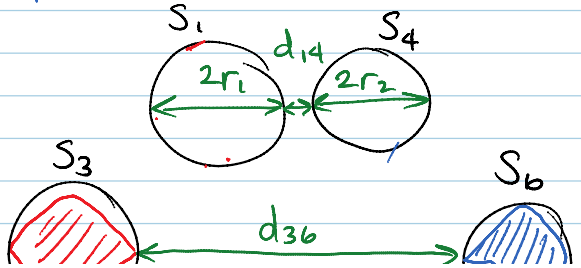
Let d_{ij} be distance between S_i & S_j i on A, j on B

Let d_{ij}^* be smallest distance found so far

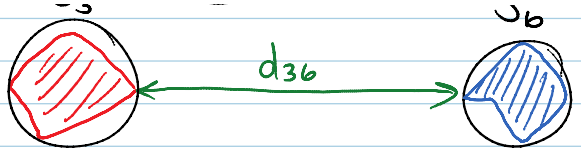
You can prune S_k & S_l if

$$d_{kl} > d_{ij}^* + 2r_i + 2r_j$$

$$d_{36} > d_{14} + 2(r_1 + r_2)$$



$d_{36} > d_{14} + 2(r_1 + r_2)$

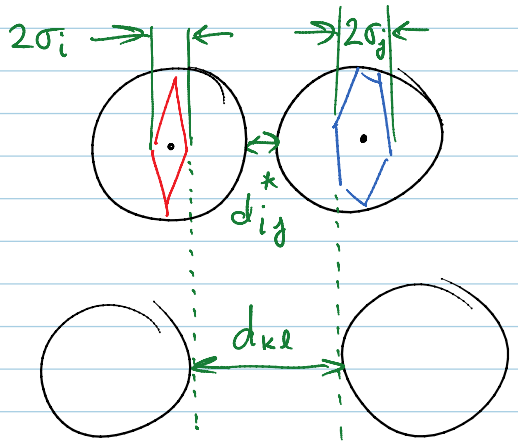


If decomposition is all convex parts and bounding volume is smallest sphere, then pruning can be more aggressive: prune if $d_{kl} > d^*_{ij} + r_i + r_j$

If you also knew minimum dimension of parts you could bound more tightly,

prune if something like

$d_{kl} > d^*_{ij} + r_i + r_j - \sigma_i - \sigma_j$



For state of the art in distance computation, see work of Dinesh Manocha at Univ. of North Carolina.

Open question: How can one most efficiently obtain all feature pairs within a given distance (and their distances (including penetration depths))?

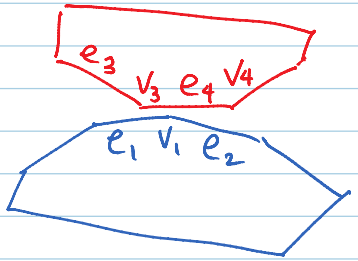
In multibody dynamics, we need to know:

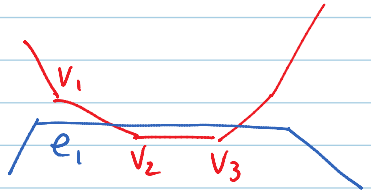
$\text{dist}(e_1, v_3) < \epsilon$

$\text{dist}(v_1, e_4) < \epsilon$

$\text{dist}(v_4, e_2) < \epsilon$

so we can construct ψ'_n correctly and ∴ simulate accurately.





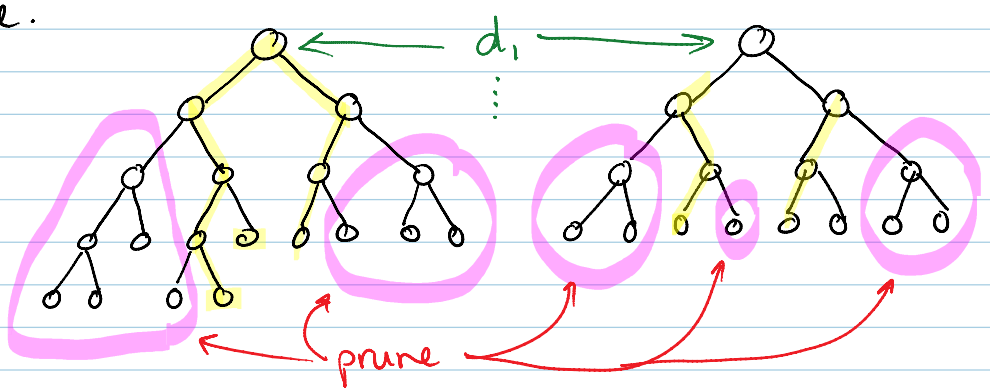
and ∴ simulate accurately.

here we need
(V_1, e_1), (V_2, e_1), (V_3, e_1)

Modify
Pruning condition

if $d_{ke} > d^*_{ig} + r_i + r_j - \sigma_i - \sigma_g + \epsilon$

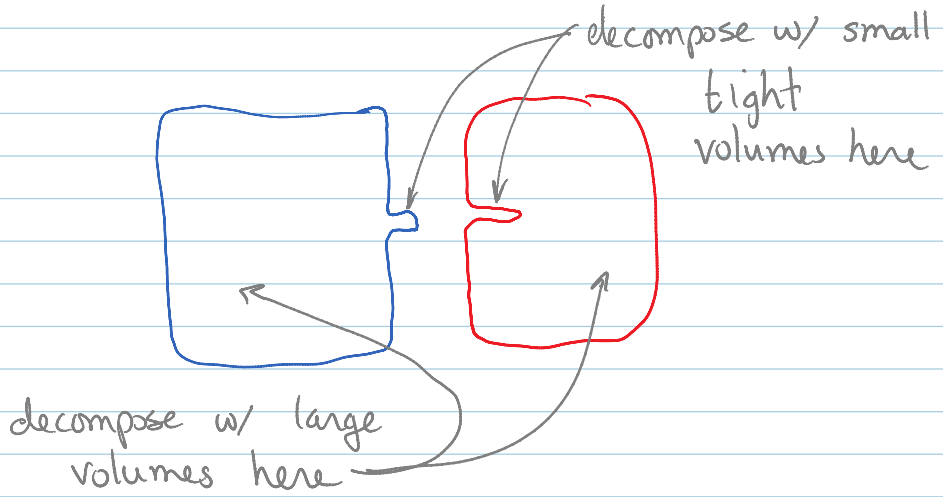
then prune.



Note:

For both collision detection & distance computation, the specific hierarchical decomposition affects computation time

For example if we know all the "action" is in certain regions, put small bounds on those regions



Incremental Collision Checking

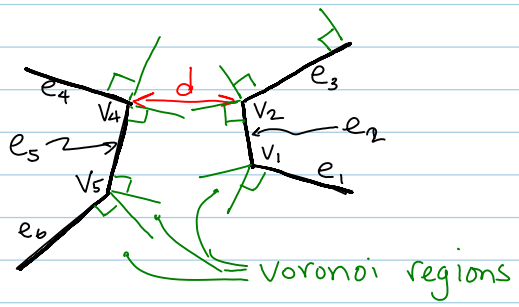
Use spatial coherence to speed collision checking.

Can achieve almost constant time.

Need knowledge of topology for this to work.

Triangle or polygon soup not good rep. for this.

Consider case of convex polygons in the plane



Voronoi region is region closest to a geometric feature. A point in a region is closest to the corresp. feature of a convex polygon.

Polygon distance condition

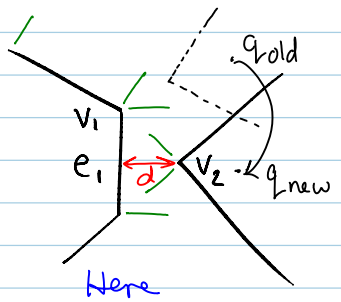
Let F_1 & F_2 be geometric features of polygons 1 & 2, respectively, where geom. features are edges and vertices.

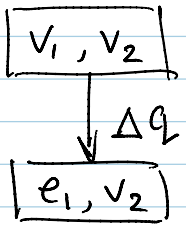
Let $(x_1, y_1) \in F_1$ and $(x_2, y_2) \in F_2$ be the closest pair of points on F_1 & F_2 among all points on F_1 & F_2

If $(x_1, y_1) \in Vor(F_2)$ and $(x_2, y_2) \in Vor(F_1)$, then the distance between (x_1, y_1) and (x_2, y_2) is the distance between the polygons.

Incremental collision checking uses knowledge of topology (i.e. connectivity of edges and vertices) to determine which feature pair is

which feature pair is likely to contain the closest points



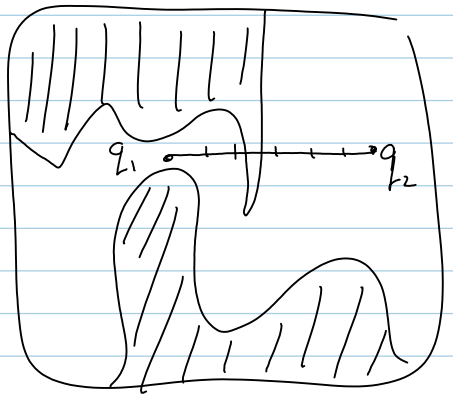


Here minimum distance shifted by one Voronoi region on one polygon.

The basic idea is that if the polygon's relative config changes only slightly, then one can quickly find the closest feature pair from the previously found pair. For example, the search should require at most moving up the tree a small # of links.

Collision checking along path segments

Sample path checking for collision at each point.



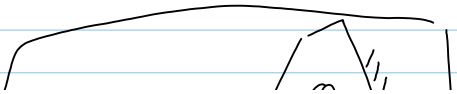
Could use multi-resolution scheme.

If collision found, then discard path segment

How can we guarantee that a finite path segment is 100% collision free?

Derive bounds. Unfortunately they tend to be loose.

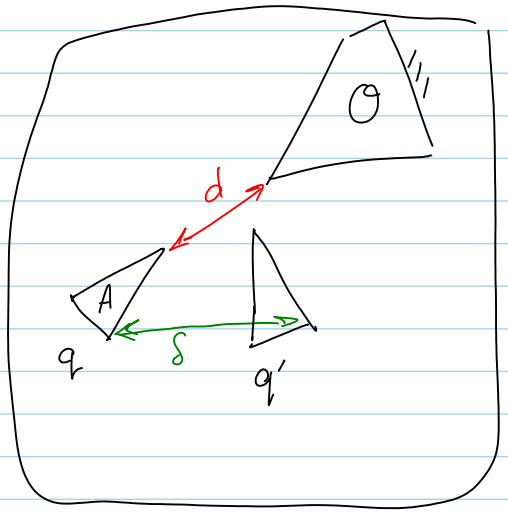
Planar Case



Planar Case

Let d be the distance at q

Let δ be the max. displacement of any point on A while moving to q'.



If $\delta < d$, is collision possible? Yes! Since part may move in strange way.

However if δ denotes the longest path taken by any $a \in A$ while moving from q to q', then if $\delta < d$, the segment from q to q' is collision free.

Let the config of A be denoted by $(x_t, y_t) \in \mathbb{R}^2$, $\theta \in S^1$

Translation

If A translates from (x_t, y_t) to (x_t', y_t'), then

$$\delta = \sqrt{(x_t - x_t')^2 + (y_t - y_t')^2}, \quad \forall\, a \in A$$

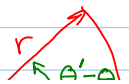
Suppose we only require path to stay in a box

Then max δ of $a \in A$ is bounded by

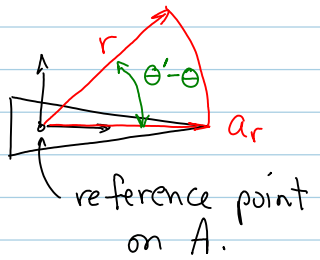
$$\delta \leq |x_t - x_t'| + |y_t - y_t'| \quad \forall\, a \in A$$

Suppose we include rotation from θ to θ' along the shortest path is S'.

Let a be point further



Let a_r be point furthest from reference point, and let r be its distance



The greatest displacement any point $a \in A$ can experience is bounded by $r|\theta' - \theta|$

If position & orientation may vary between their limits, then a collision free region of C_{free} is:

$$\text{subset of } C_{free} = \{(x_t', y_t', \theta') \in C \mid |x_t - x_t'| + \ldots$$

$$\ldots + |y_t - y_t'| + r|\theta - \theta'| < d\}$$

The bound can be used to choose a collision-checking stepsize Δq such that no collisions will be missed along the path.

An analogous approach can be used in 3D worlds.

When the robot has multiple links, the approach becomes strongly configuration dependent, making it difficult to apply cost-effectively.

5.4 Incremental Searching & Sampling

Key steps in Algorithm

1. Initialize $G(V,E)$, $E = \{\}$, $V = \{q_I, q_G\}$
2. VSM. Choose $q_{cur} \in V$

3. LPM. Attempt to connect q_{new} to q_{cur}.
(not necessarily in V.)

Create path $\tau : [0,1] \rightarrow C_{free}$

$\tau(0) = q_{cur}$, $\tau(1) = q_{new}$

4. Insert edge if step 3 succeeds

5. Check for solution. (Complicated if multiple graphs or trees)

6. If no solution found, go to step 2.

Questions:

Choose grid points up front?

How many? What resolution?

exponential growth with dimension of C

Which points should I attempt to connect next?

Points far from each other? Nearby points?

Hard to choose all points up front, so consider interleaving sampling & searching for connections

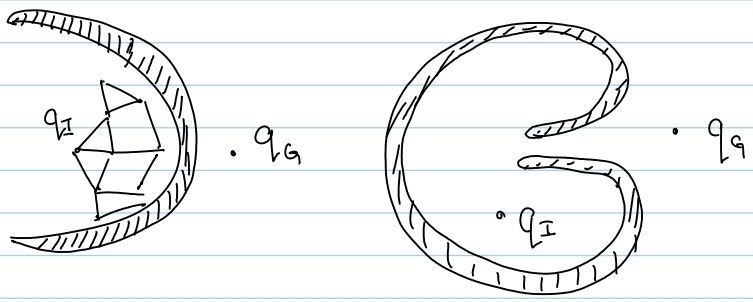
Let α be a sequence of points dense on C.

Maybe try searching after each 100 samples?

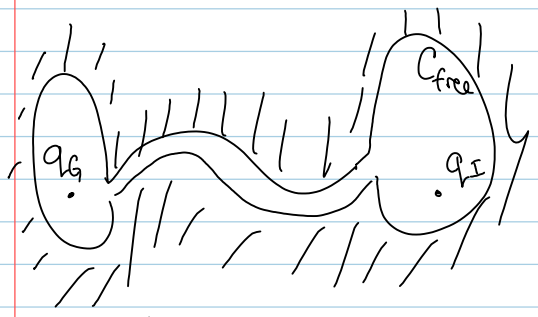
Maybe search every time the # of components of G drops?

Another option: Choose an unreasonably high # of points, and count on heuristic to solve w/o visiting a large fraction of the points.

Difficult problems:



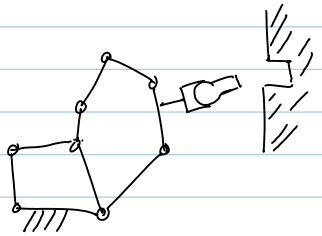
"Lobster" trap



Narrow passage

Problems living on complicated varieties.

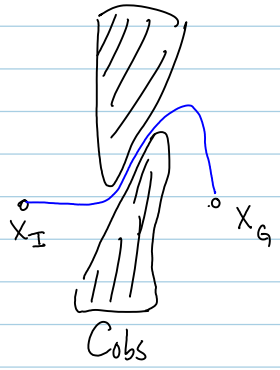
e.g. parallel manipulators



Problems living in high-dimensional spaces

eg. dynamic systems

Problems with dynamics and contact avoidance can be solved using something like dVC as the LPM.



Use forces of contact in controller to deflect robot away from contact

---

If graphs are not too large, one can solve probs. with algs in section 2.2.

G is small if: dimension of C is low
resolution is low
hueristic is very good

e.g. Suppose the robot has:
6 dof (flying robot)
7 dof (some industrial arms)

if we want 100 points per d.o.f.,
$\Rightarrow 10^{12}$, 10^{14} points.

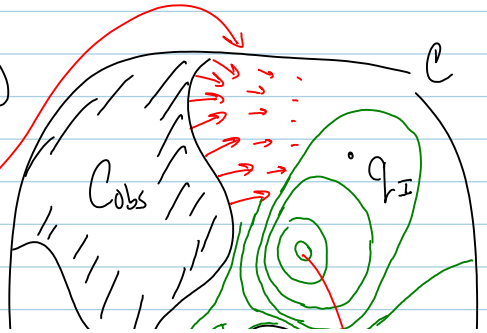
That's a big graph! What about humanoid robots w/ 50+ joints?

Best-first & A* could solve w/o visiting many points and ∴ might not have to create a large G.

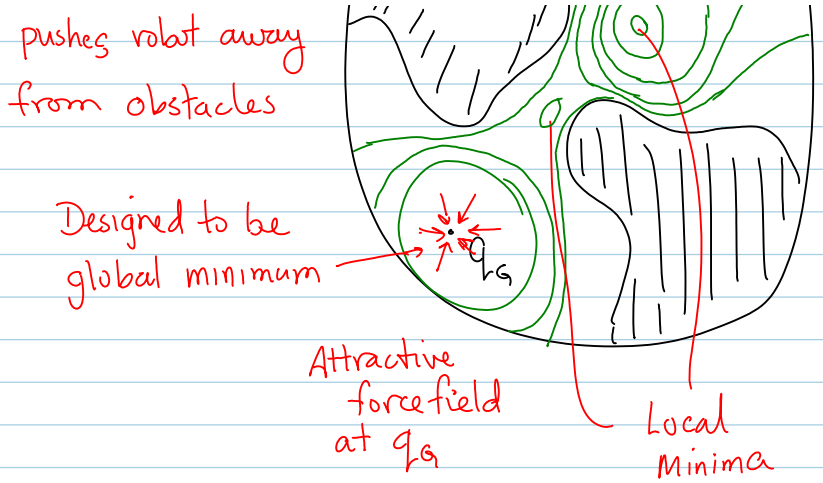
One way to guide BestFirst is by constructing and artificial potential field.

Potential Fields

(show transparency)



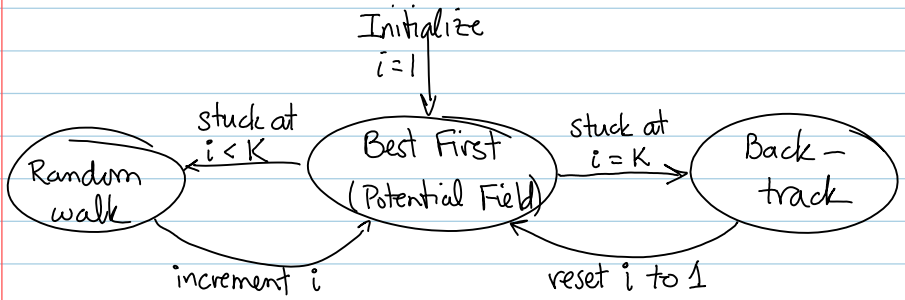
Force field pushes robot away from obstacles



Very hard to design potential field w/o local minima and global minimum at goal.

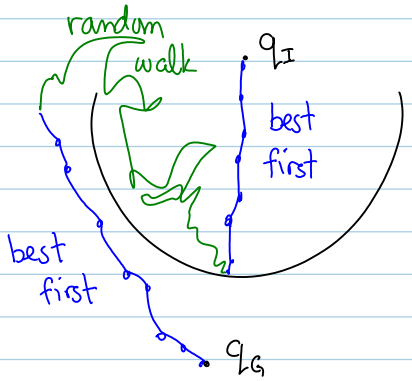
"navigation functions" by Koditschek applies to special geometries.

Randomized Potential Field Method



Follow best first search until all neighbors make negative progress.

Switch to random walk for a "while."



Key

You must know how to find neighboring

Questions:

configurations!

This is a vote in favor of grids.

How do we design good pot. fields?

Value of K?

what resolution?

From which node should walk begin?

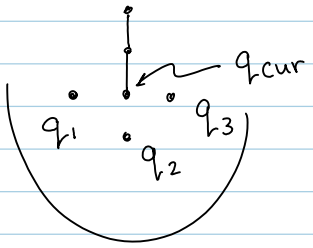
More detail

Let $g = g_r + g_a$

g_r: repulsive, g_a: attractive

e.g. $g_r = a/\psi_n(q)$

$g_a = b\|q - q_G\|$



Let $g(q)$ denote the potential of q.

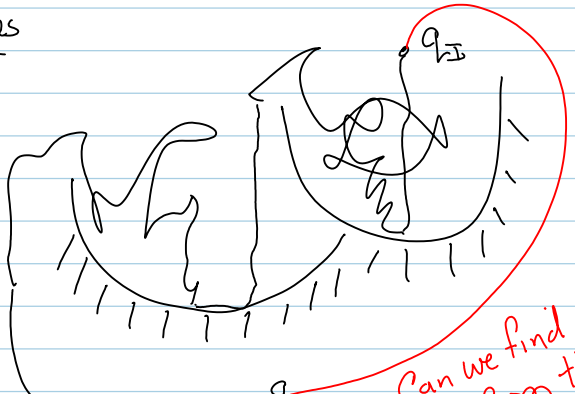
Move next to q_i s.t.

$g(q_i) < g(q_j)$, $j \neq i$

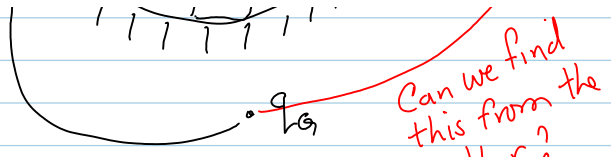
If $g(q_i) \geq g(q_{cur})$, then best first is stuck.

Random walks make very bad paths!

Smoothing issues

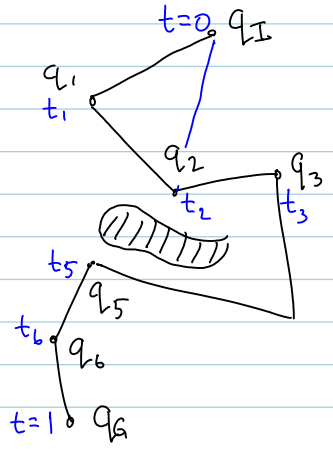


One way:



One way:
Recursively smooth the path, τ,
is to pick pairs of points along the path & connect them with a straight line in $\mathcal{C}$. Then the new path is:

$$\tau' = \begin{cases} \tau(t) & 0 \le t \le t_1 \\ a\tau(t_1) + (1-a)\tau(t_2) & t_1 \le t \le t_2 \\ \tau(t) & t_2 \le t \le 1 \end{cases}$$

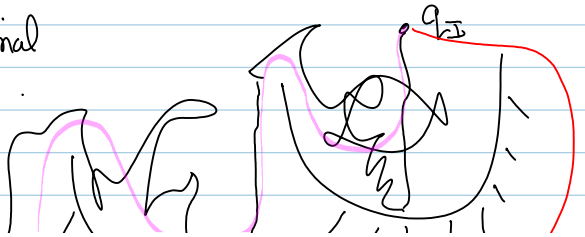


Try $q_I \to q_2$. If successful cut out q_1

What if we tried $q_1 \to q_6$?
we could potentially cut out q_2, q_3, q_4, q_5

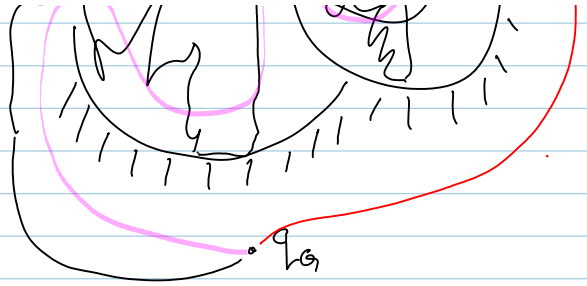
The order of pairs to connect is not cut and dry.
A path metric would be helpful. What's important?
# of turns, energy expended, path length, ...

Suppose we suspect that our smoothed path is from a sub-optimal equivalence class (homotopy class)?



(homotopy class)?

Maybe the red one is better.



You could continue searching until you believe you have a path from every homotopy class, but this would be very hard to determine definitively. Determining this is likely to be as hard as computing C_{obs} exactly.

In general much parameter tuning is need for good performance. And one tuning may not work well for all problems.

Other methods

Ariadne's Clew

Key idea: interleave search & exploration

Algorithm:

① VSM – choose vertex q_e in G at random

② LPM – find new point q_{new} maximally far from q_e that can be "easily" connected.

③ Try to connect q_{new} to the other component of G

Drawback – finding q_{new} is a difficult optimization problem. Optimization method have to be tuned to each motion planning problem.

To each motion planning problem.

Init w/ q_I & q_G

Attempt connect (q_I, q_G)
fail.

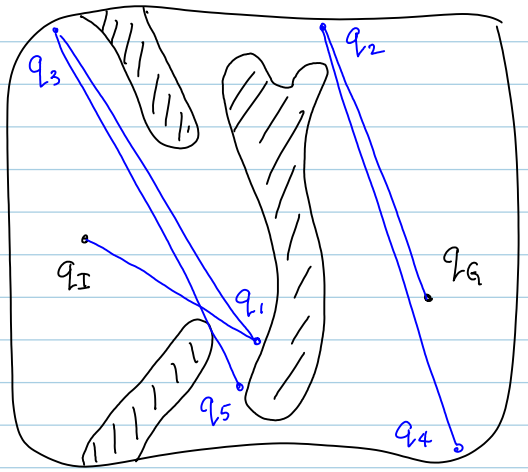
Choose $q_e = q_I$
find q_1

Connect (q_1, q_G) = fail

Choose $q_e = q_G$

Find q_2

Connect (q_2, q_I) = fail



Choose $q_e = q_1$, Find q_3, Connect (q_3, q_G) = fail.

Alg fails in this example, since only 5 new q's will ever be found. That is choosing any $q_e \in S$ will find q_{new} already in S.

Expansive space planner

Goal: generate samples in unexplored regions of C_{free}

① VSM – select q_e from S w/ probability inv. proportional to its # of neighbors

② LPM – expand q_e to get q_{new} w/in a nbhd of q_e

③ Insert q_{new} into S w/ probability inv. prop. to # of neighbors of q_{new}

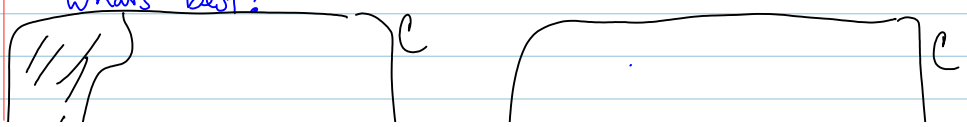
Three parameters to tune.

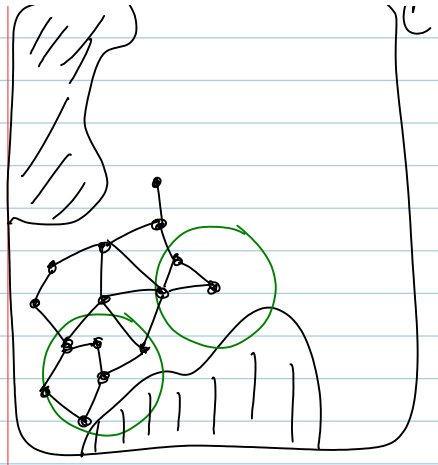
Sizes of nbhds important:

Too small yields uniform prob. over S

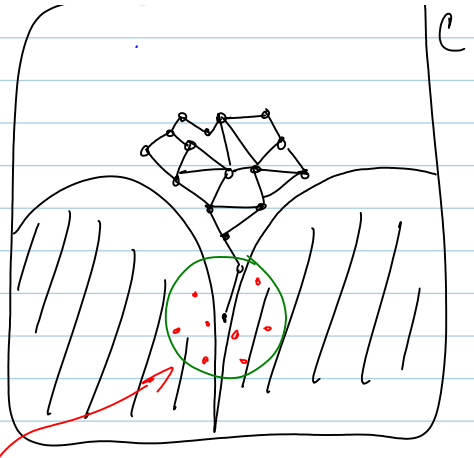
Too large " " " " S.

What's best?





Bias search to explore from nodes on bndry of graph.

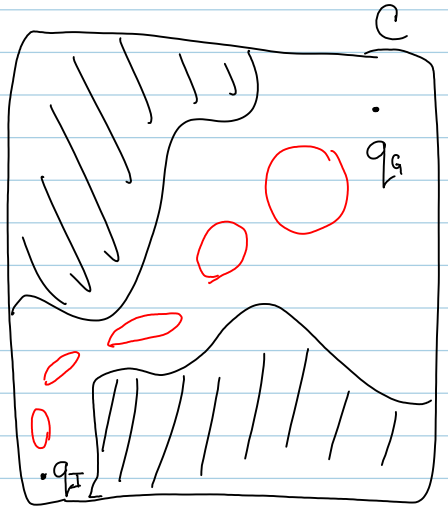


Bias search from node with little hope of success. Could keep failed points, but that would be bad if there was a narrow passage.

Random Walk Planner

Choose new points from a pdf that adapts to C_{obs}.

For example new points could be drawn from a multivariate Gaussian with changing mean and covariance matrix



As search progresses in the example, the pdf narrows & aligns in passages and becomes less biased in wide open areas.

Recall the definition of a multivariate gaussian

$$f(q) = \frac{\exp\left(-\frac{1}{2}(q-u)^T \Sigma^{-1} (q-u)\right)}{2\pi^{N/2} \, |\Sigma|^{1/2}}$$

where μ is mean of p.d.f.

Σ is covariance matrix

N is length of vector q

$|\Sigma|$ is determinant of Σ