15-780: Graduate AI

Lecture 2. Spatial Search

Geoff Gordon (this lecture)
Ziv Bar-Joseph
TAs Michael Benisch, Yang Gu

Admin

- *WEH 5409, Sep 18, 4:30-5:30pm: matlab tutorial*
- *Please send your email address to TA Michael Benisch (mbenisch at cs), who is compiling a class email list*
- *Please check the website regularly for readings (for Lec. 1–2, Ch. 1–4 of RN)*

Last episode, on *Grad AI*

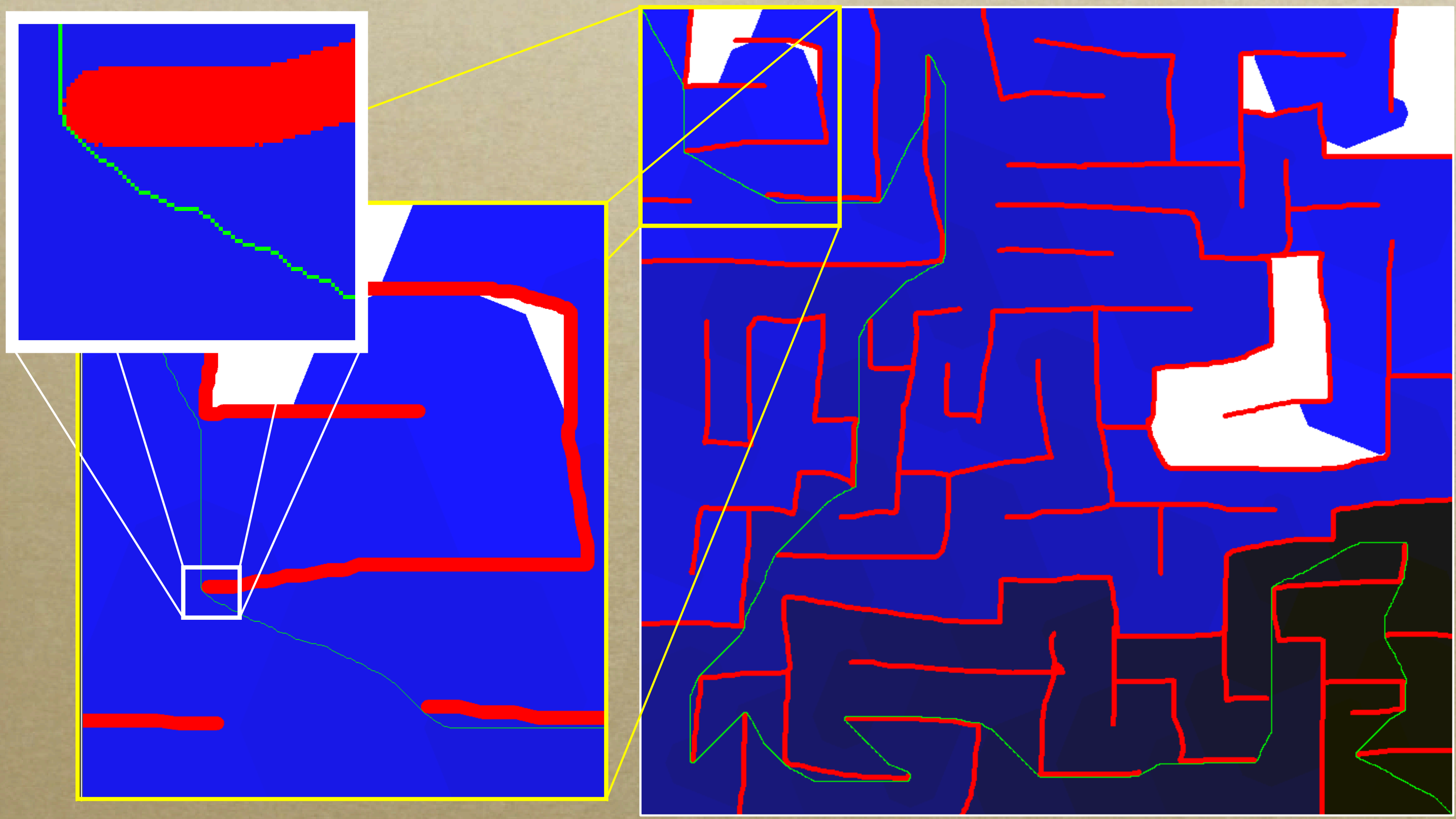
Topics covered

- *What is AI? (Be able to discuss an example or two)*
- *Types of uncertainty & corresponding approaches*
- *How to set up state space graph for problems like the robotic grad student or path planning*

Topics covered

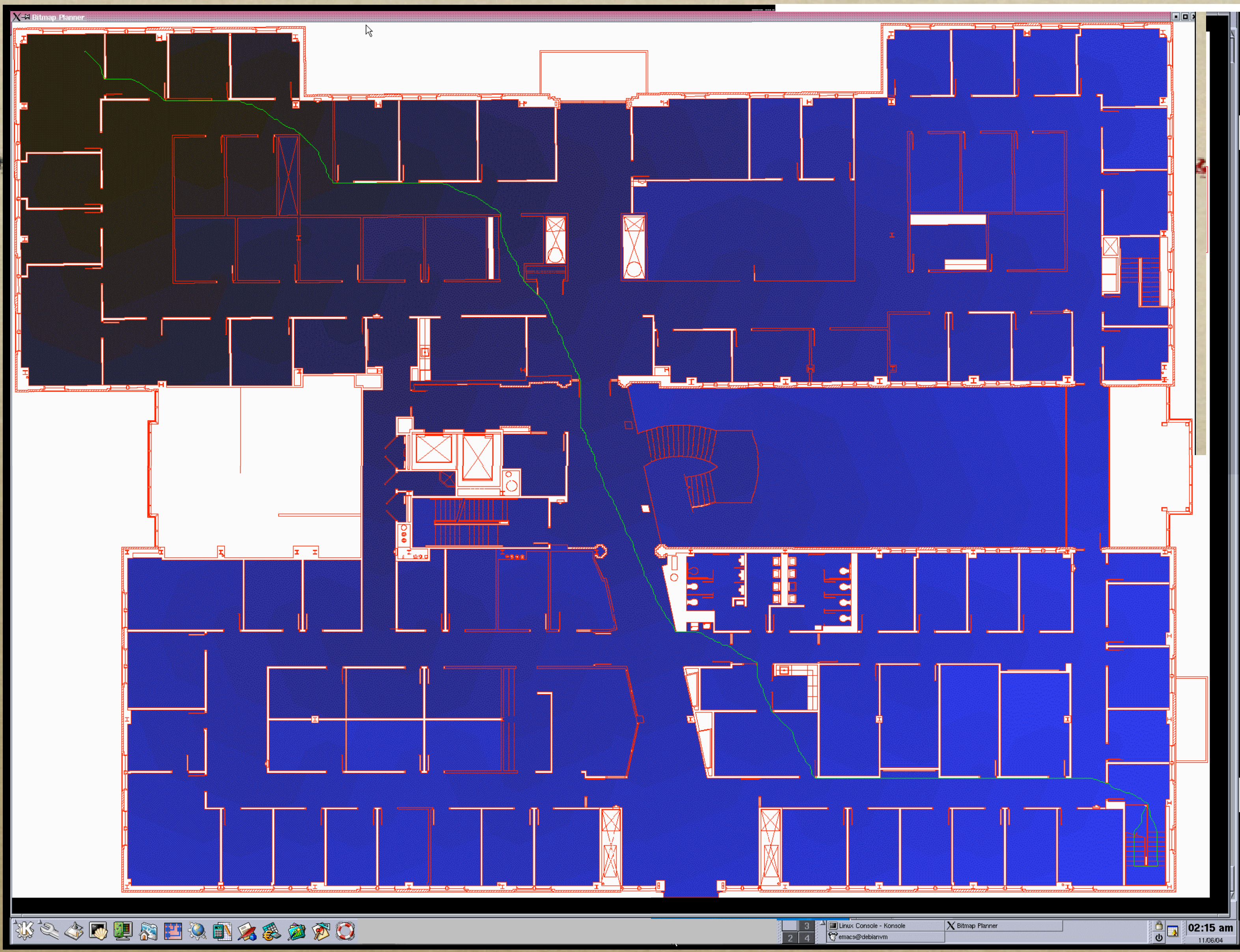
- *Generic search algorithm & data structures*
- *Search methods: be able to simulate*
  - *BFS, DFS, DFID*
  - *Heuristic search*
  - *A*: define admissibility; show optimality, efficiency*
- *What are advantages of each?*

A* Planning on Big Grids



Credit: Kuffner

2D grids: 500,000 nodes = ~ 0.8 sec
10 million nodes = ~ 12 sec

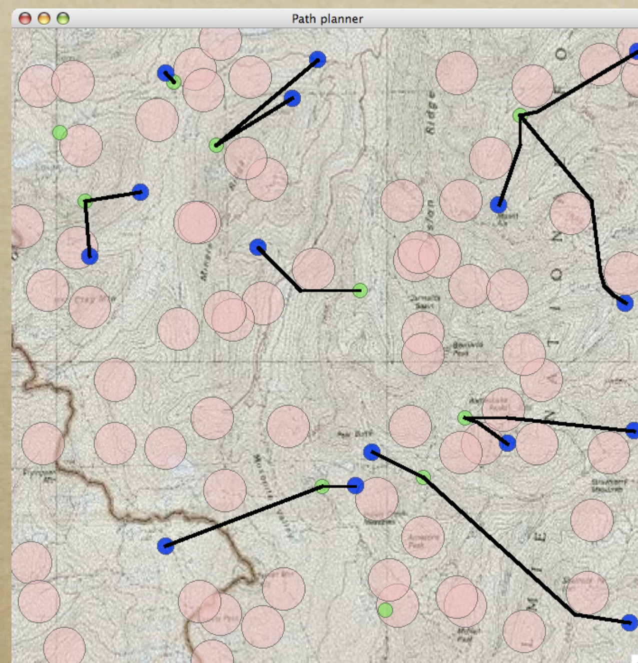

Bitmap Planner
Linux Console - Konsole
emacs@debianvm
Bitmap Planner
02:15 am

Projects

Project ideas

- *Plan a path for this robot so that it gets a good view of an object as fast as possible*

Project ideas



- *Implement a distributed market-based planner and test the contribution of learning to overall performance*

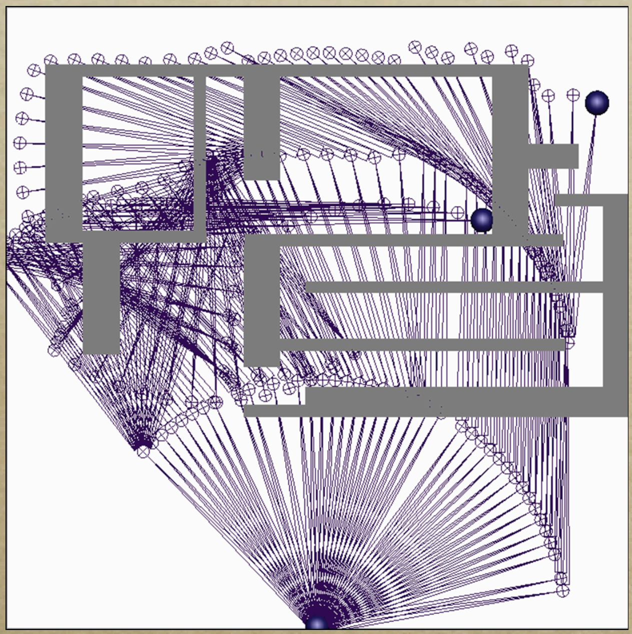
Project ideas

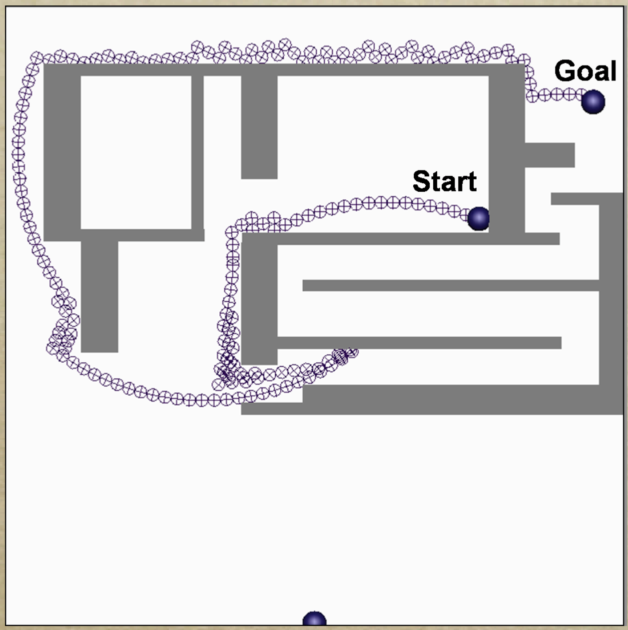
- *Give me an excuse to buy the new Lego Mindstorms set*
  - *plan footstep placements*
  - *plan how to grip objects*

Spatial Planning

Plans in Space…



Optimal Solution



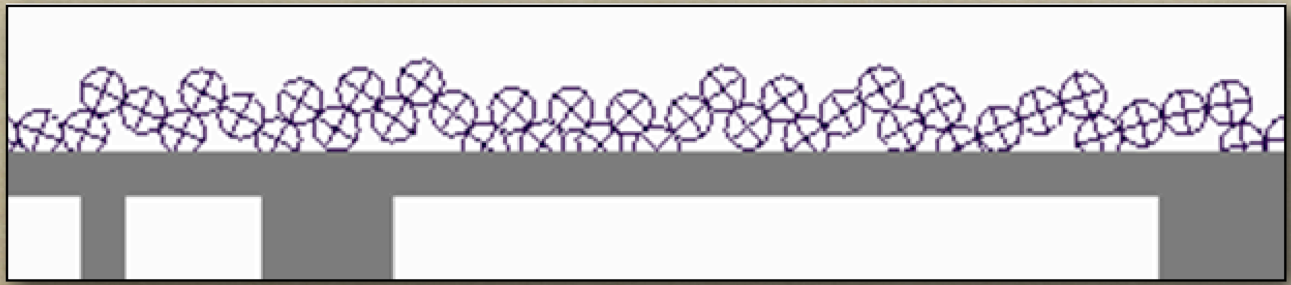
End-effector Trajectory

- *Last time, we saw A* for spatial planning*

What's wrong w/ A* guarantees?

- ***(optimality)*** *A* finds a solution of depth g**
- ***(efficiency)*** *A* expands no nodes that have f(node) > g**

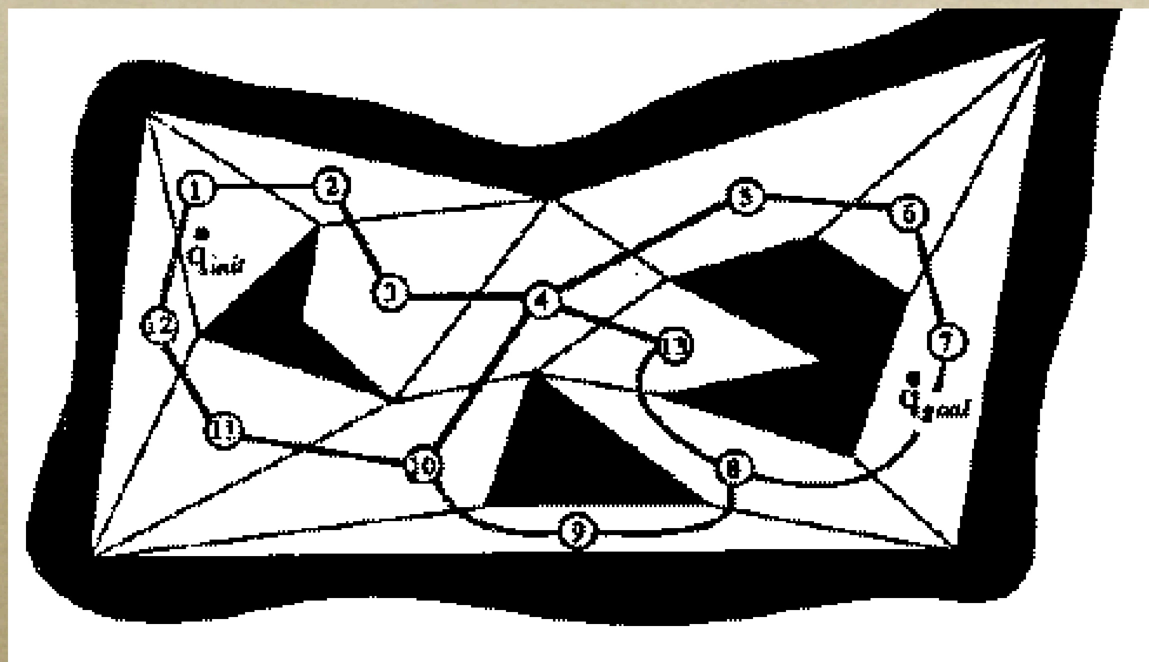
What's wrong with A*?



- *Discretized space into tiny little chunks*
  - *a few degrees rotation of a joint*
  - **Lots** *of states* $\Rightarrow$ *slow*
- *Discretized actions too*
  - *only allowed to move one joint at a time*
- *Results in jagged paths*

What's wrong with A*?

Wouldn't it be nice…

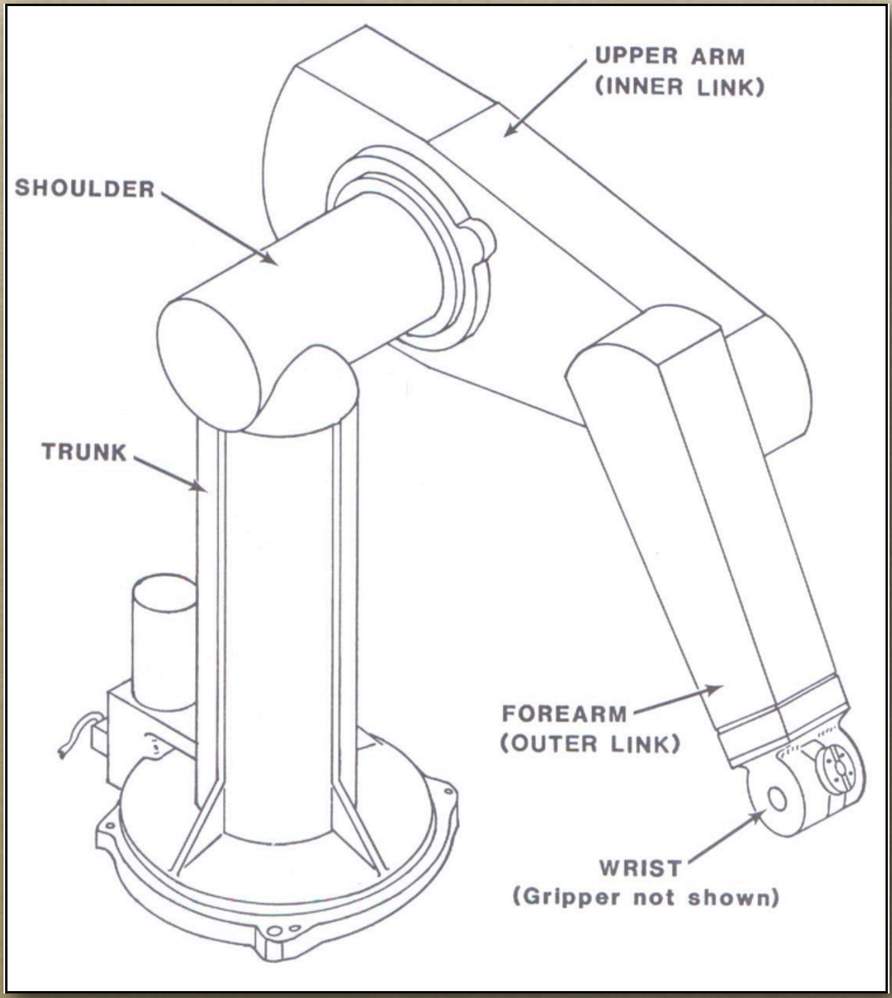


- *… if we could break things up based more on the real geometry of the world?*
- Robot Motion Planning *by Jean-Claude Latombe*

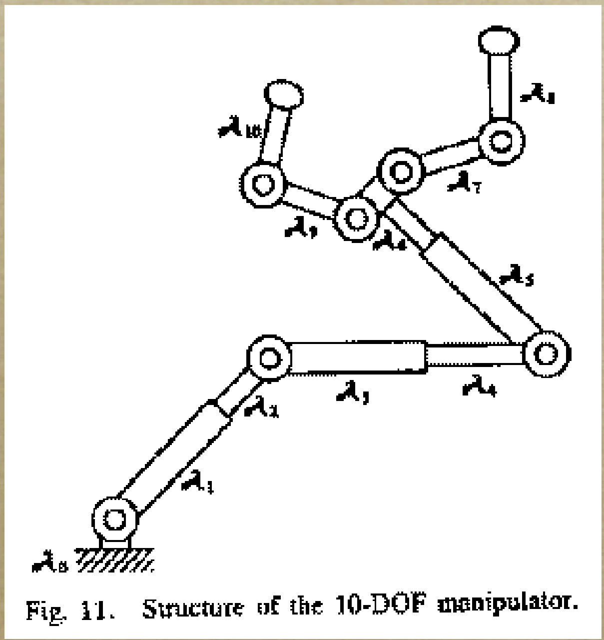
Physical system

- *A moderate number of real-valued coordinates*
- *Deterministic, continuous dynamics*
- *Continuous goal set (or a few pieces)*
- *Cost = time, work, torque, …*

Typical physical system



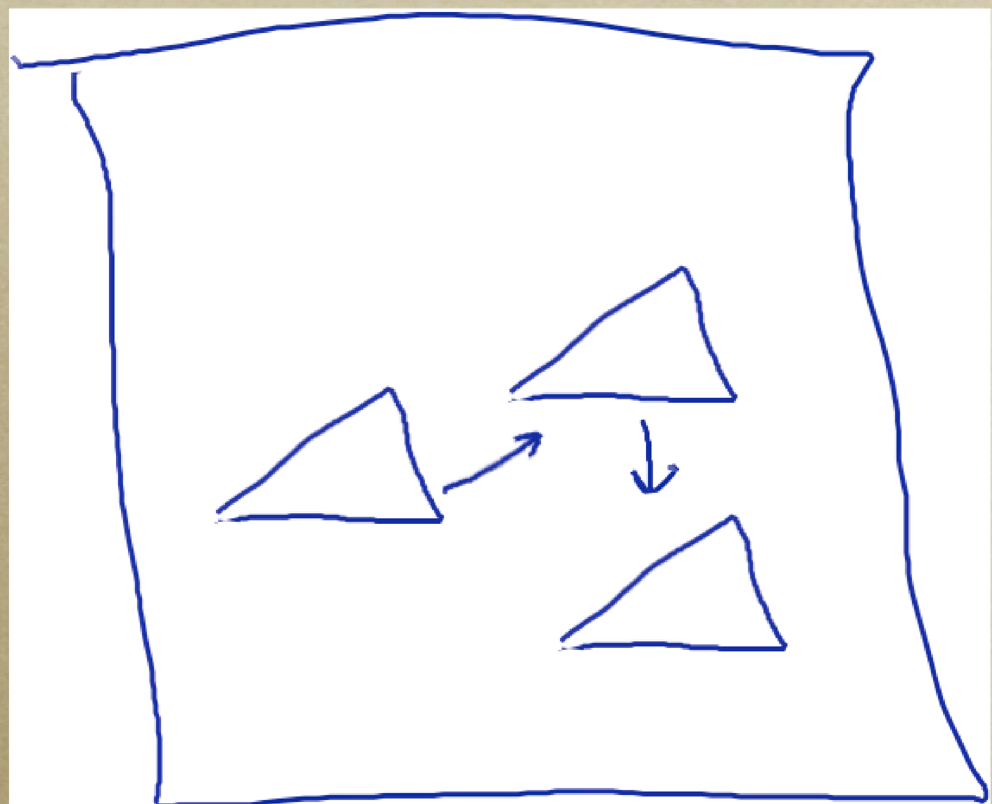
A kinematic chain



- *Rigid links connected by joints*
  - *revolute or prismatic (1 dof each)*
- *Configuration*

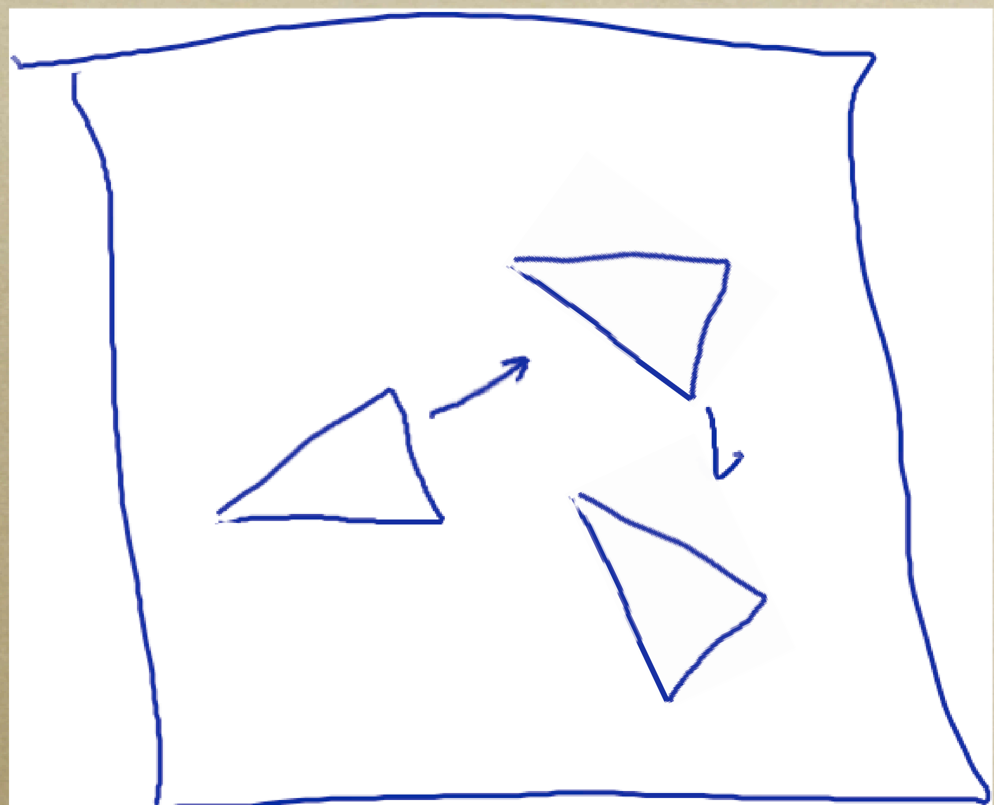
  $\mathbf{q} = (q_1, q_2, \ldots)$

Mobile robots



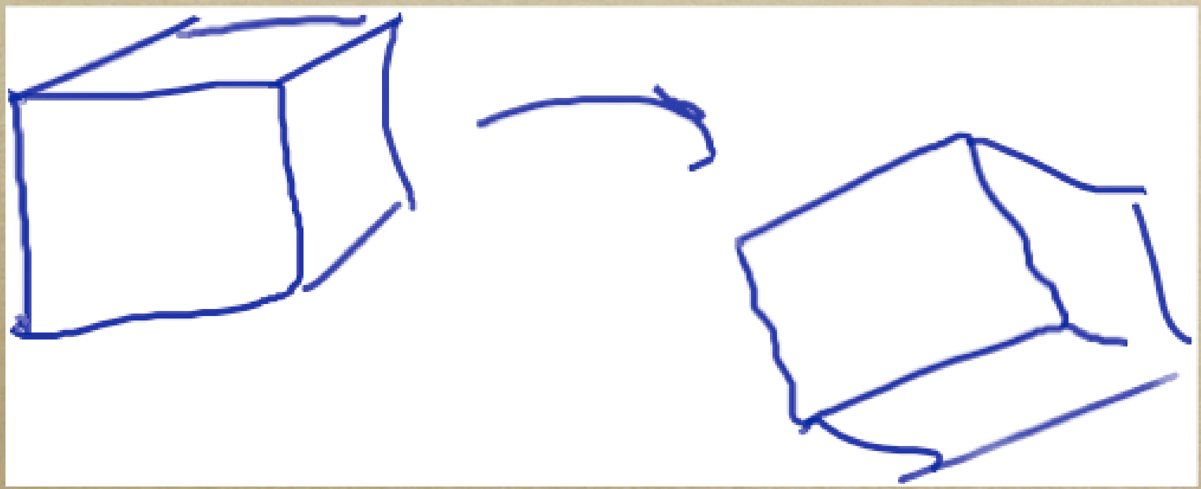
- *Translating in space = 2 dof*

More mobility

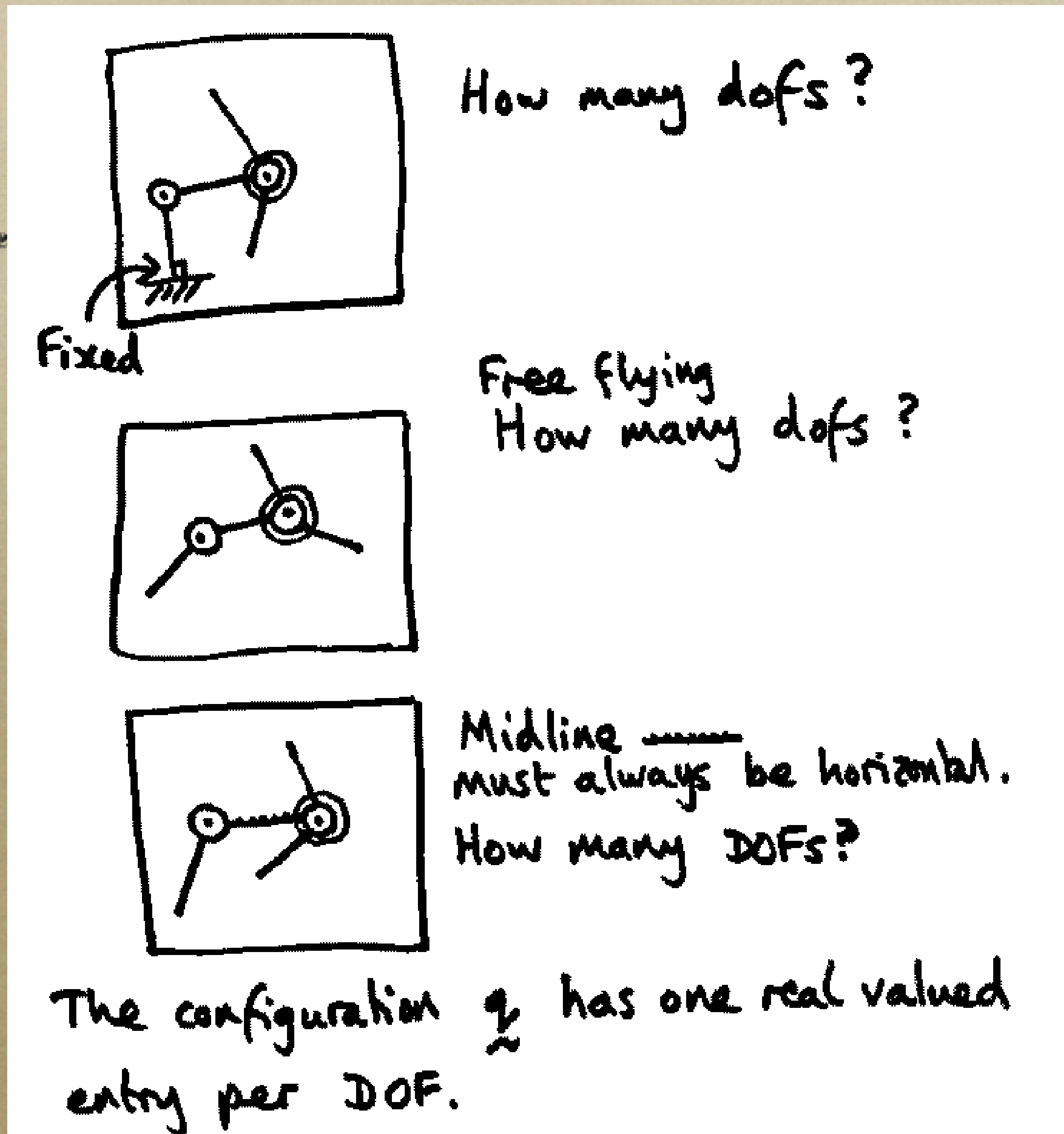


- *Translation + rotation = 3 dof*

Q: How many dofs?



- *3d translation & rotation*



credit: Andrew Moore

Robot kinematic motion planning

- *Given a robot (coordinates* **q***)*
- *… and a workspace with obstacles*
- *… get from a start to a goal*

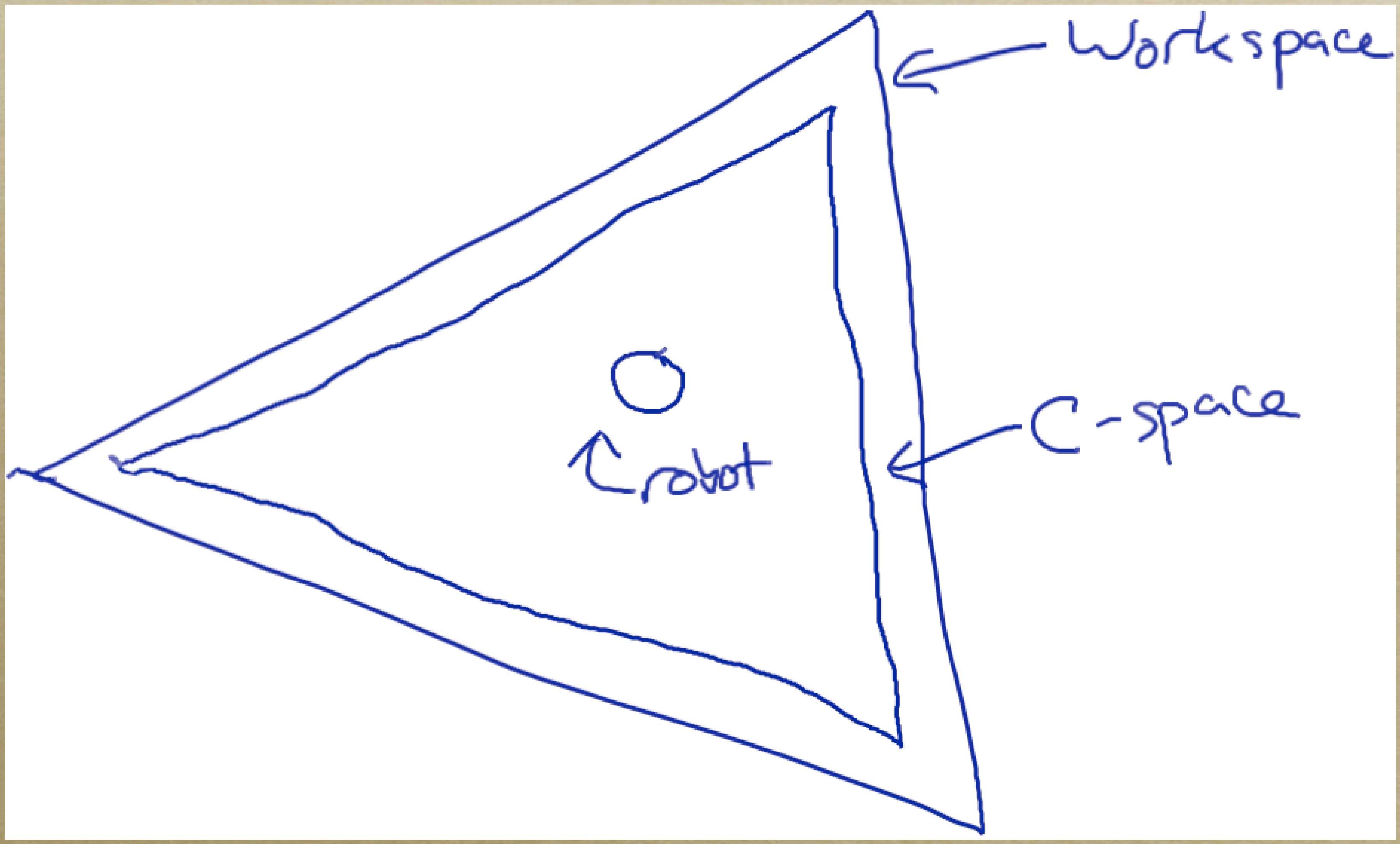
Kinematic planning

- *For any configuration* **q**, *can test whether it intersects obstacles*
- *Set of legal configs is "configuration space" C (a subset of* $\Re^{dofs}$*)*
- *Path is a continuous function q from [0,1] into C with q(0) =* $\mathbf{q_s}$ *and q(1) =* $\mathbf{q_g}$

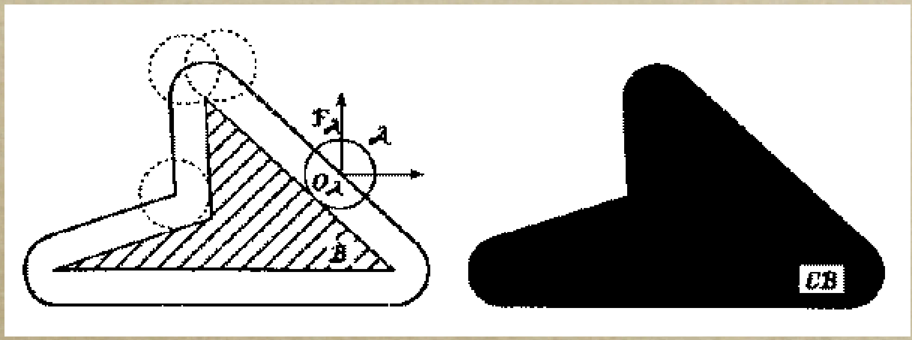
Note: dynamic planning

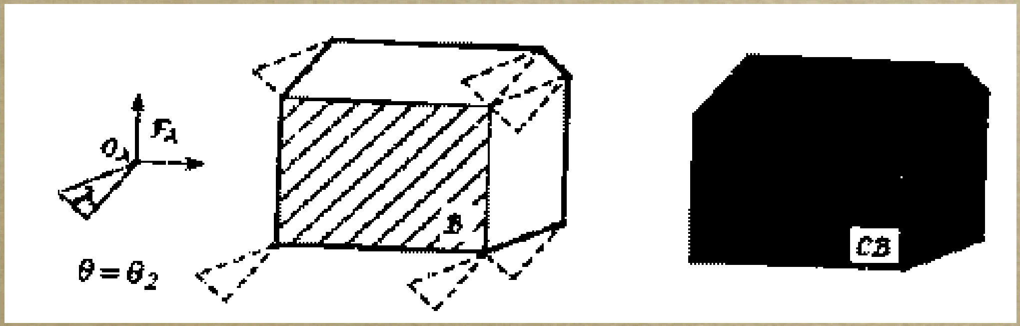
- *Includes inertia as well as configuration*
- $\mathbf{q}, \dot{\mathbf{q}}$
- *Harder, since twice as many dofs*
- *More later…*

C-space example



More C-space examples





Another C-space example

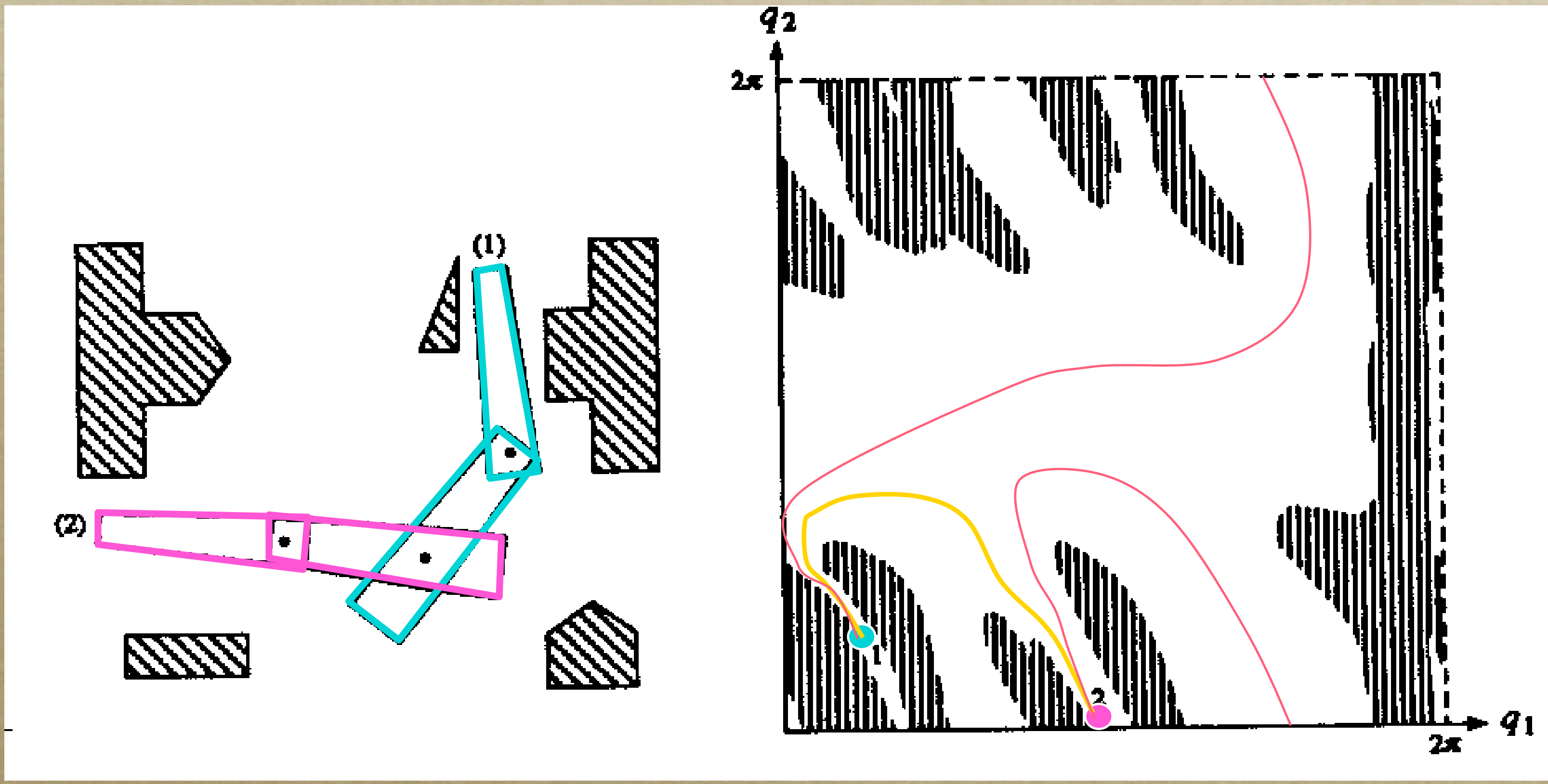
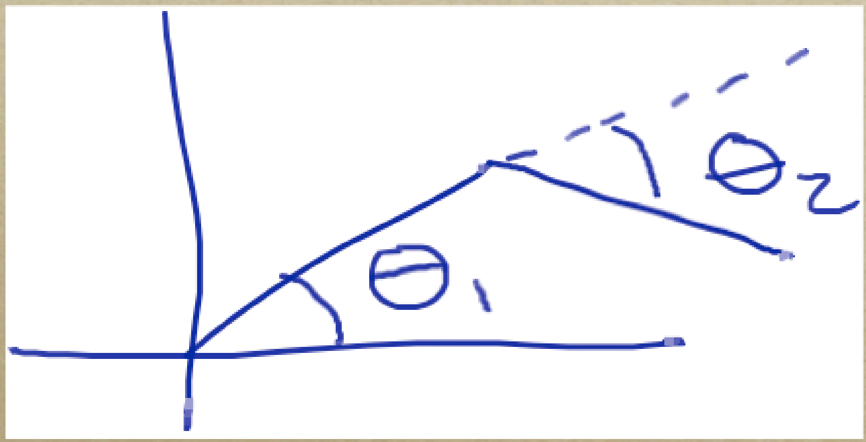


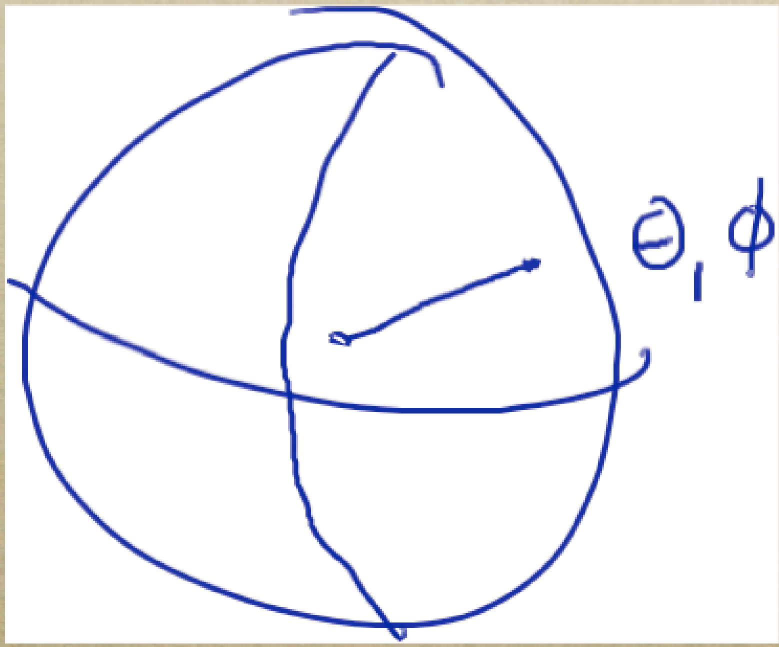
image: J Kuffner

Topology of C-space

- *Topology of C-space can be something other than the familiar Euclidean world*
- *E.g. set of angles = unit circle = SO(2)*
  - *not [0, 2π) !*
- *Ball & socket joint (3d angle) ⊆ unit sphere = SO(3)*

Topology example



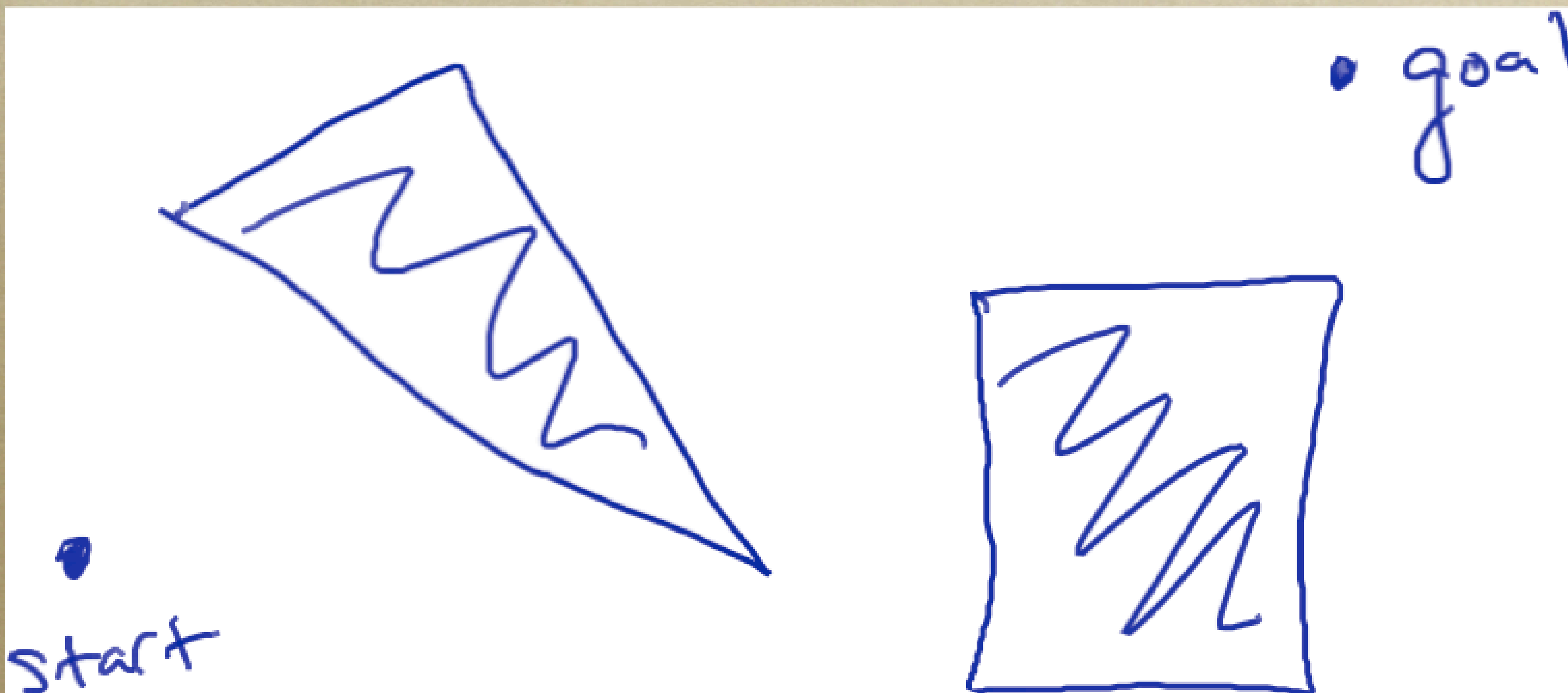


- *Compare L to R: 2 planar angles v. one solid angle — both 2 dof (and neither the same as Euclidean 2-space)*

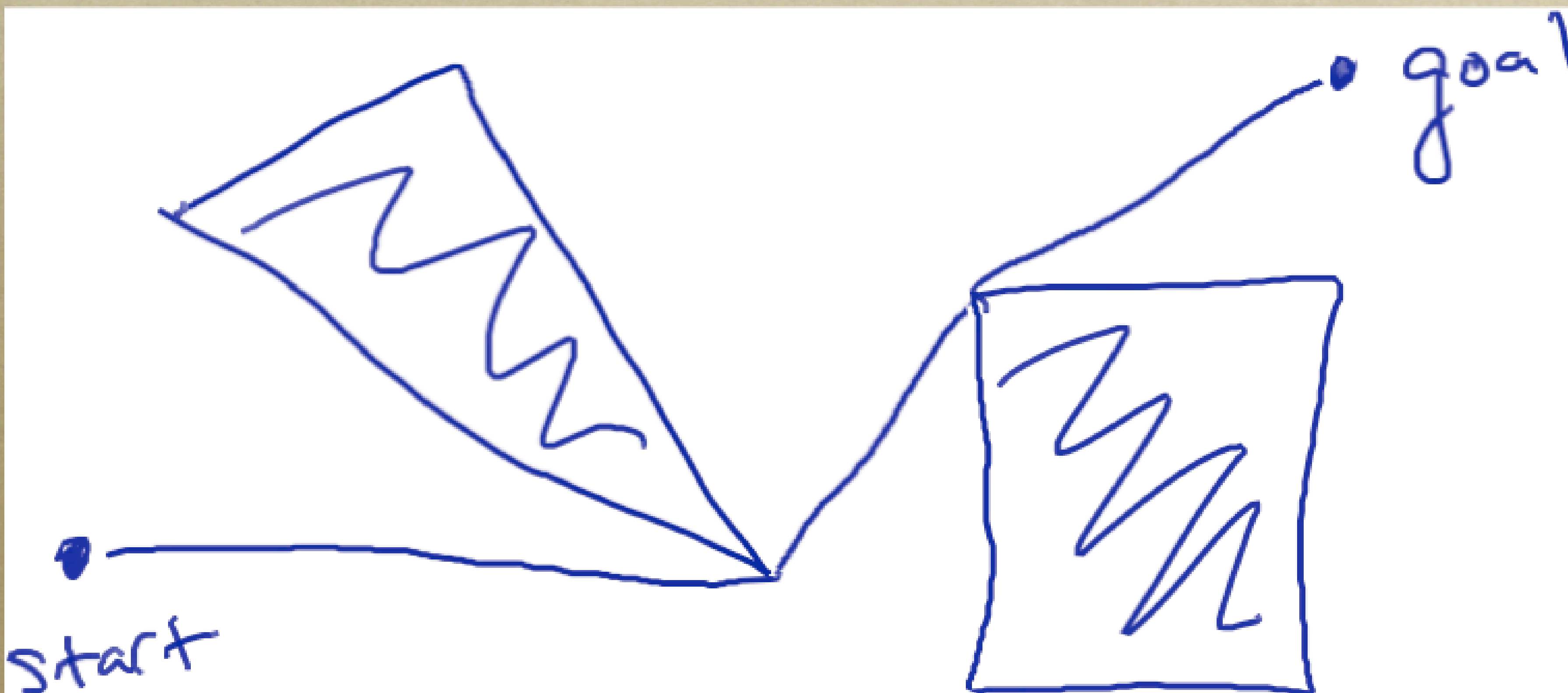
Back to planning

- *Complaint with A* was that it didn't break up space intelligently*
- *How might we do better?*
- *Lots of roboticists have given lots of answers!*

Shortest path in C-space



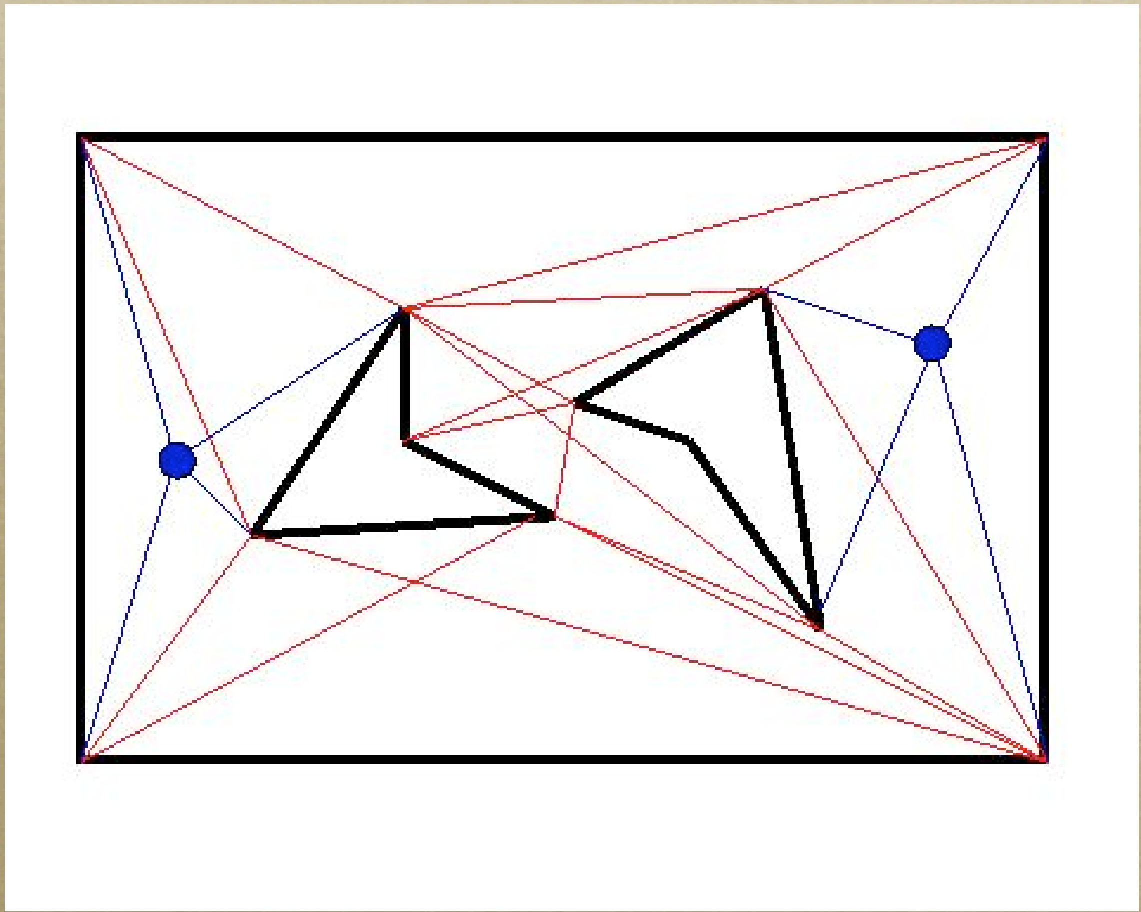
Shortest path in C-space



Shortest path

- *Suppose a polygonal C-space*
- *Shortest path in C-space is a sequence of line segments*
- *Each segment's ends are either start or goal or one of the vertices in C-space*
- *In 3-d or higher, might lie on edge, face, hyperface, …*

Visibility graph



http://www.cse.psu.edu/~rsharma/robotics/notes/notes2.html

Naive algorithm

For i = 1 … points

For j = 1 … points

included = t

For k = 1 … faces

if segment ij intersects face k

included = f

Complexity

- *Naive algorithm is $O(n^3)$ in planar C-space (grows fast with d!)*
- *For algorithms that run faster, $O(n^2)$ and $O(k + n \log n)$, see [Latombe, pg 157]*
  - *k = number of edges that wind up in visibility graph*
- *Once we have graph, search it!*

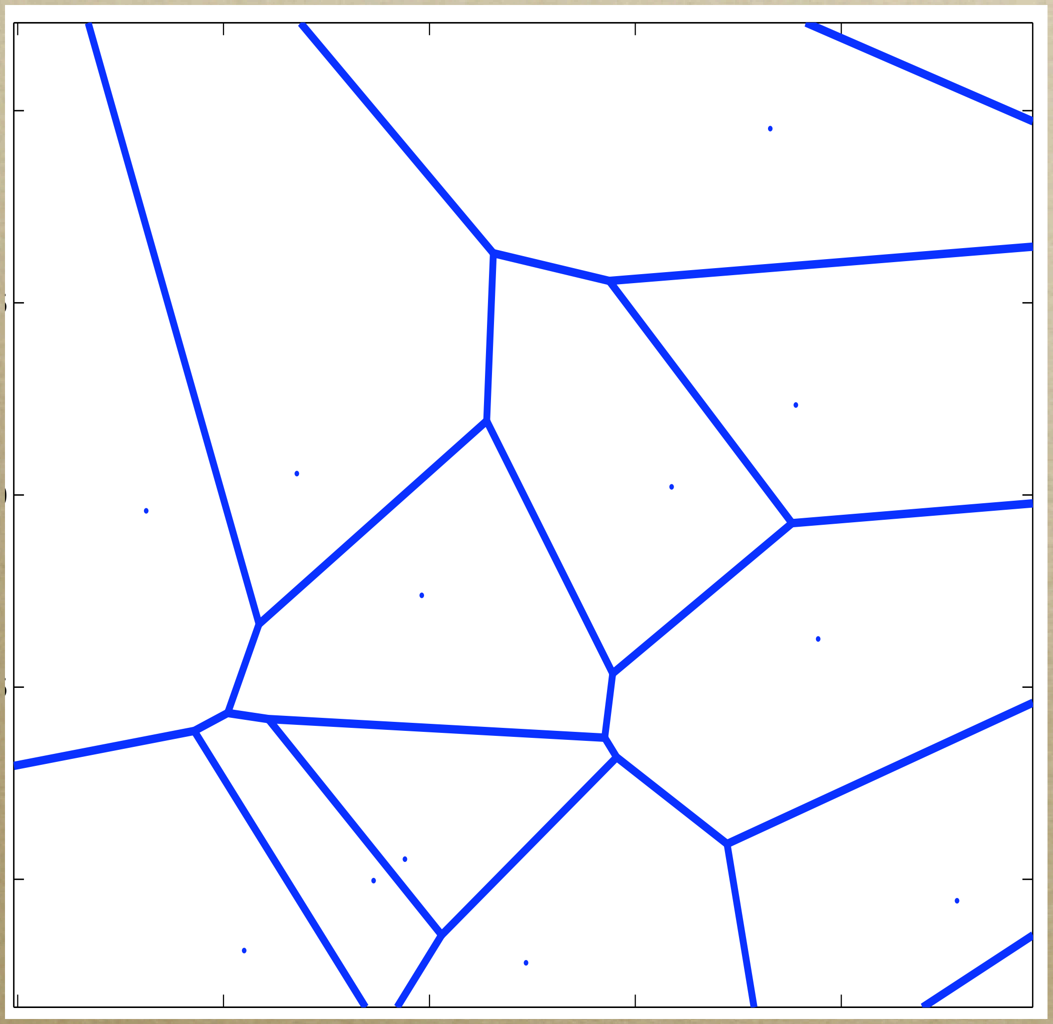
Discussion of visibility graph

- *Good: finds shortest path*
- *Bad: complex C-space yields long runtime, even if problem is easy*
  - *get my 23-dof manipulator to move 1mm when nearest obstacle is 1m*
- *Bad: no margin for error*

Getting bigger margins

- *Could just pad obstacles*
  - *but how much is enough? might make infeasible…*
- *What if we try to stay as far away from obstacles as possible?*

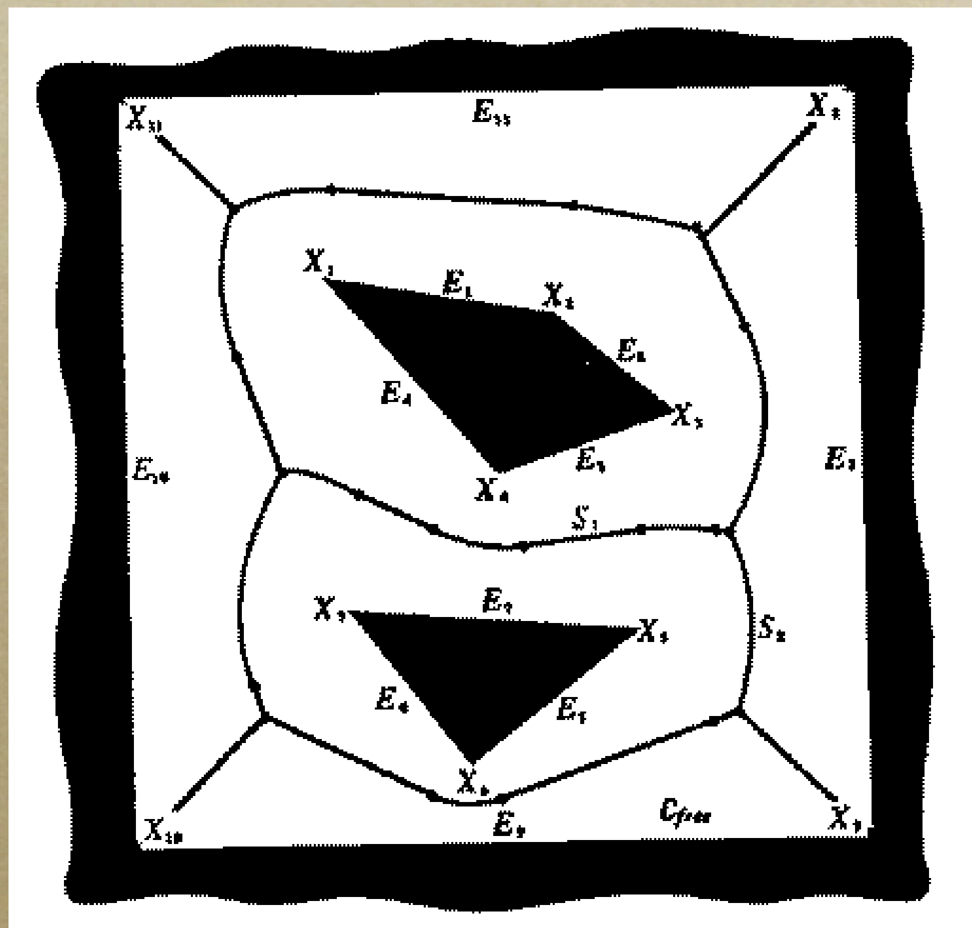
Voronoi



Voronoi

- *Given a set of point obstacles*
- *Find all places that are equidistant from two or more of them*
- *Result: network of line segments*
- *Called* Voronoi graph
- *Each line stays as far away as possible from two obstacles while still going between them*

Voronoi from polygonal C-space



Voronoi from polygonal C-space

- *Set of points which are equidistant from 2 or more closest points on border of C-space*
- *Polygonal C-space in 2d yields lines & parabolas intersecting at points*
  - *lines from 2 points*
  - *parabolas from line & point*

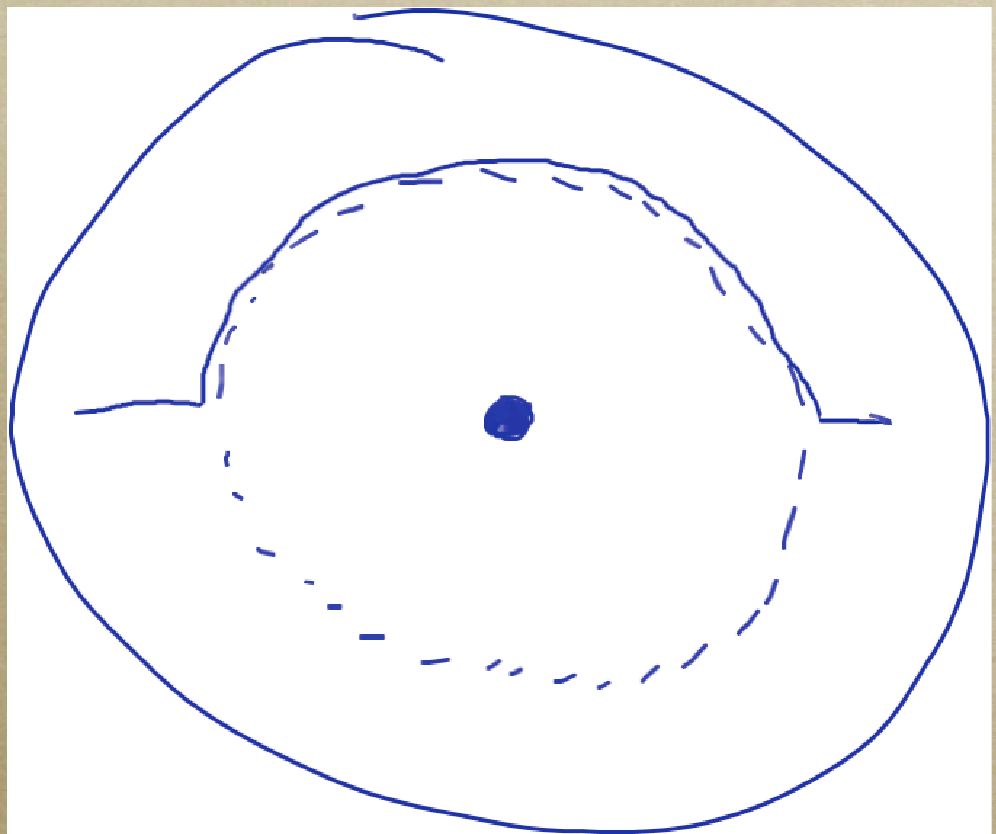
Voronoi method for planning

- *Compute Voronoi diagram of C-space*
- *Go straight from start to nearest point on diagram*
- *Plan within diagram to get near goal (guess which algorithm)*
- *Go straight to goal*

Discussion of Voronoi

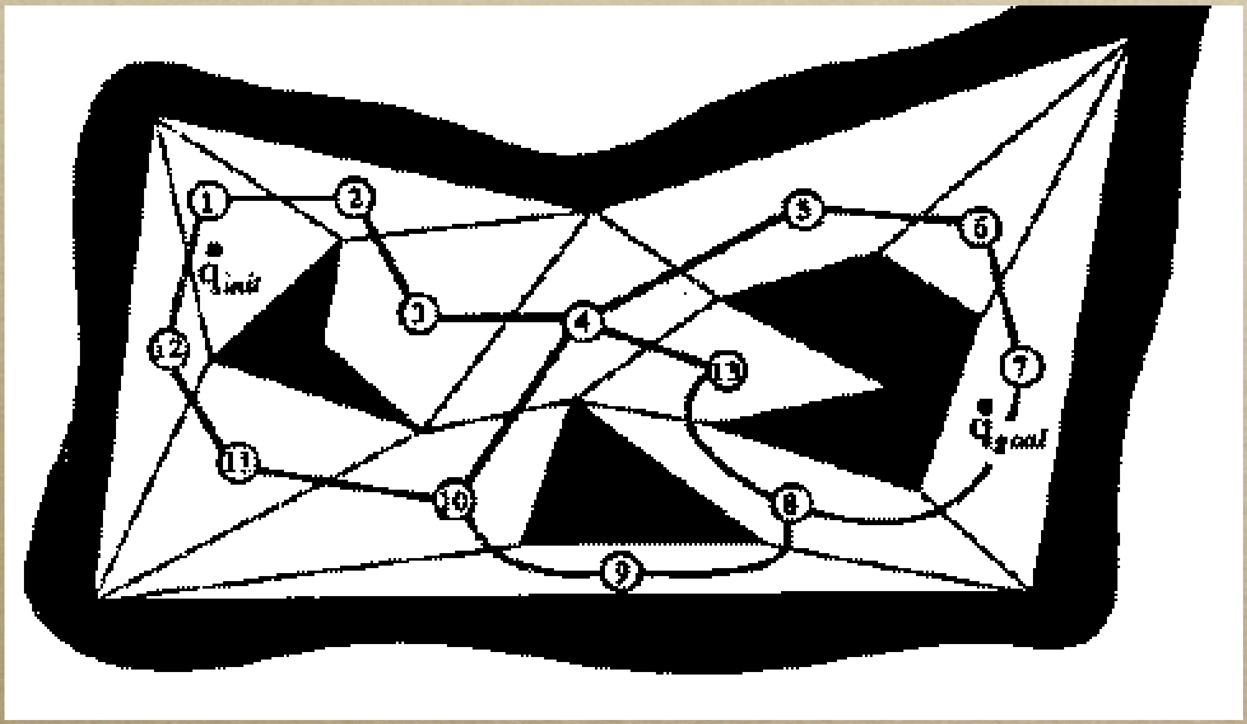
- *Good: stays far away from obstacles*
- *Bad: assumes polygons*
- *Bad: assumes 2d, gets kind of hard in higher dimensions (but see http://voronoi.sbp.ri.cmu.edu/~motion/)*

Voronoi discussion



- *Bad: kind of gun-shy about obstacles*

Exact cell decompositions

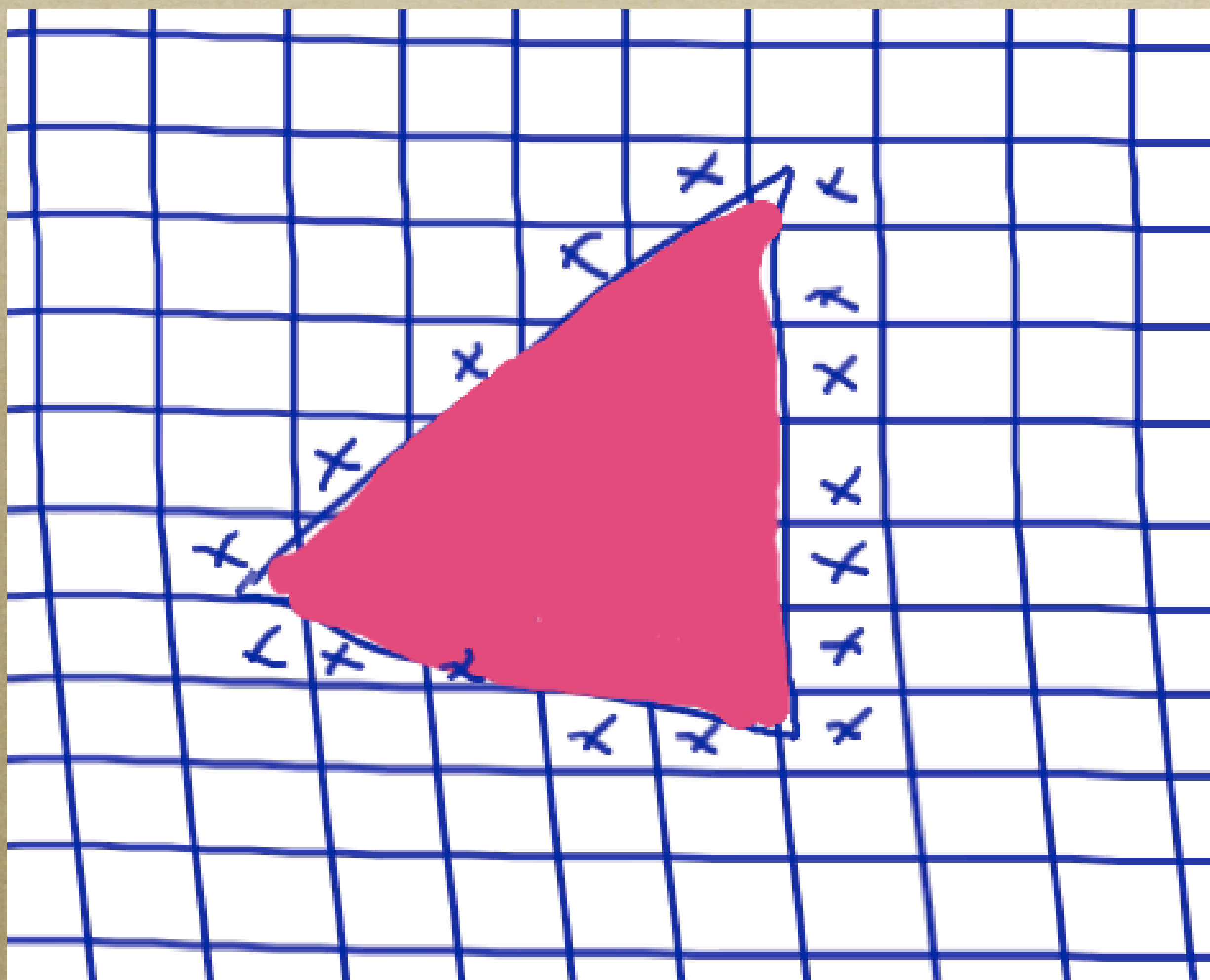


- *We can try to break C-space into a bunch of convex polygons*

Exact cell decompositions

- *Will not discuss how to do*
- *Common approach for video game NPCs*
- *But is also hard in higher than 2d*
- *And can result in wobbly paths*

Approximate cell decompositions



Planning algorithm

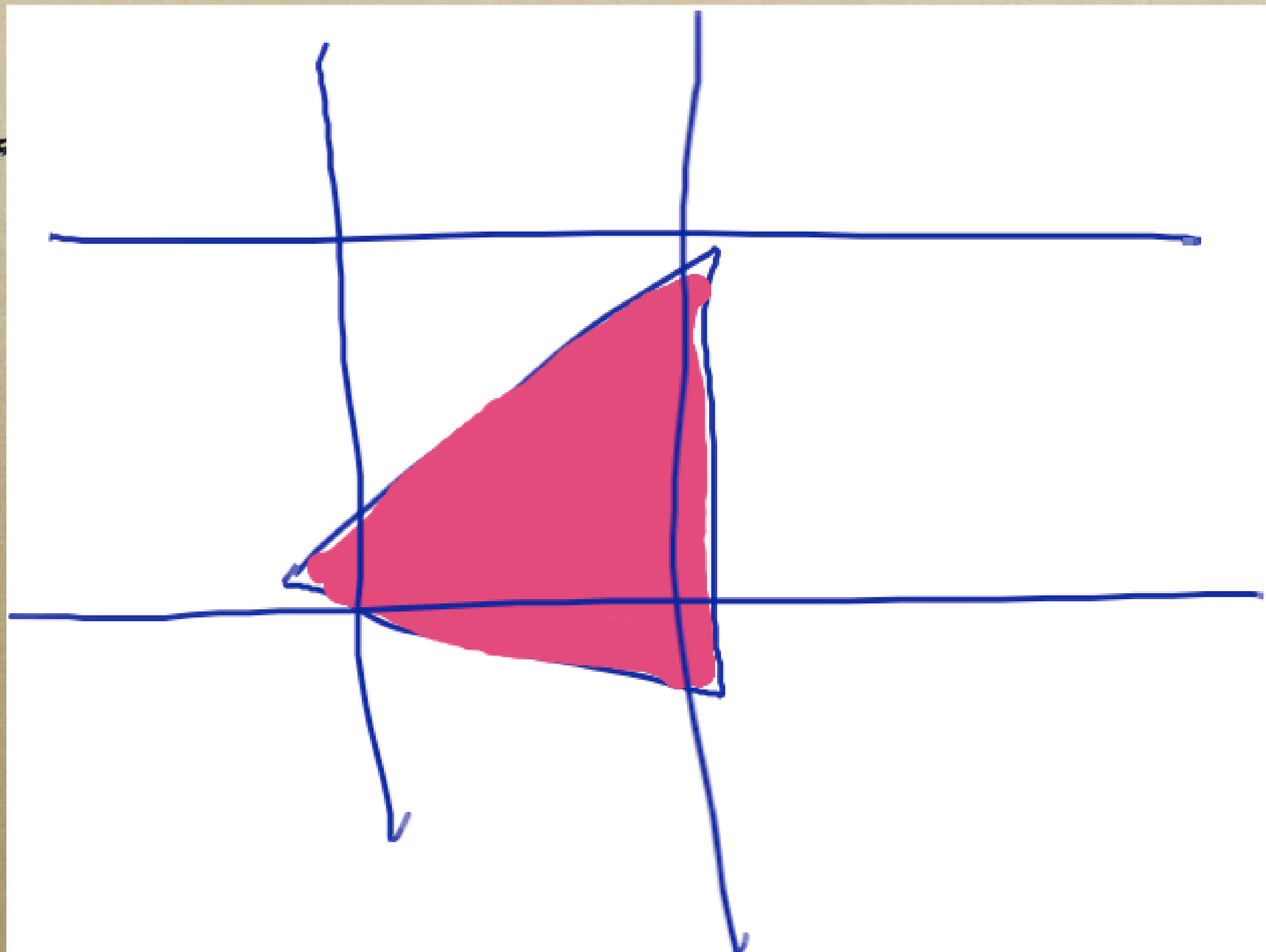
- *Lay down a grid in C-space*
- *Delete cells that intersect obstacles*
- *Connect neighbors*
- *A* (surprise!)*
- *If no path, double resolution and try again*
  - *never know when we're done*

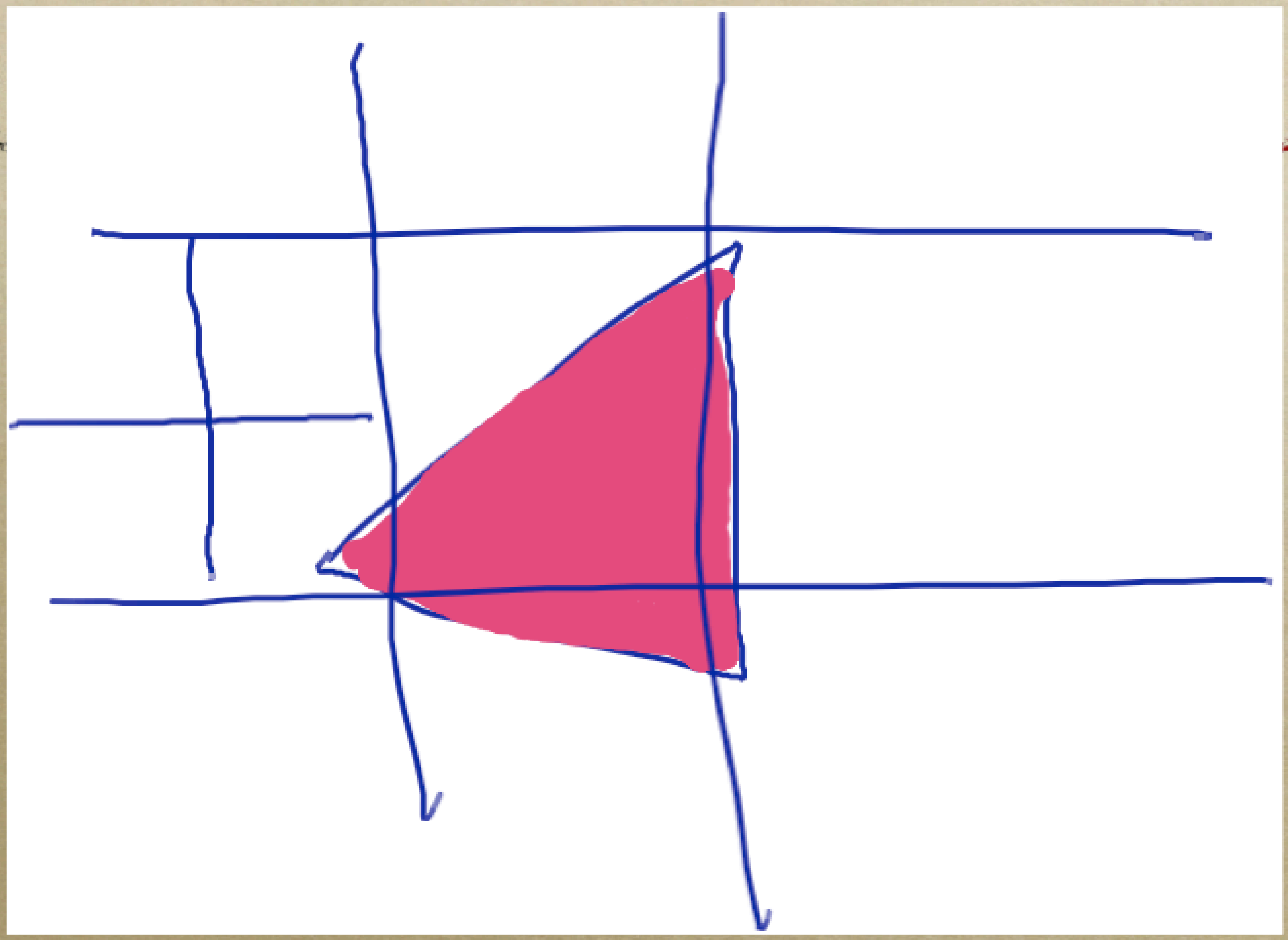
Approximate cell decomposition

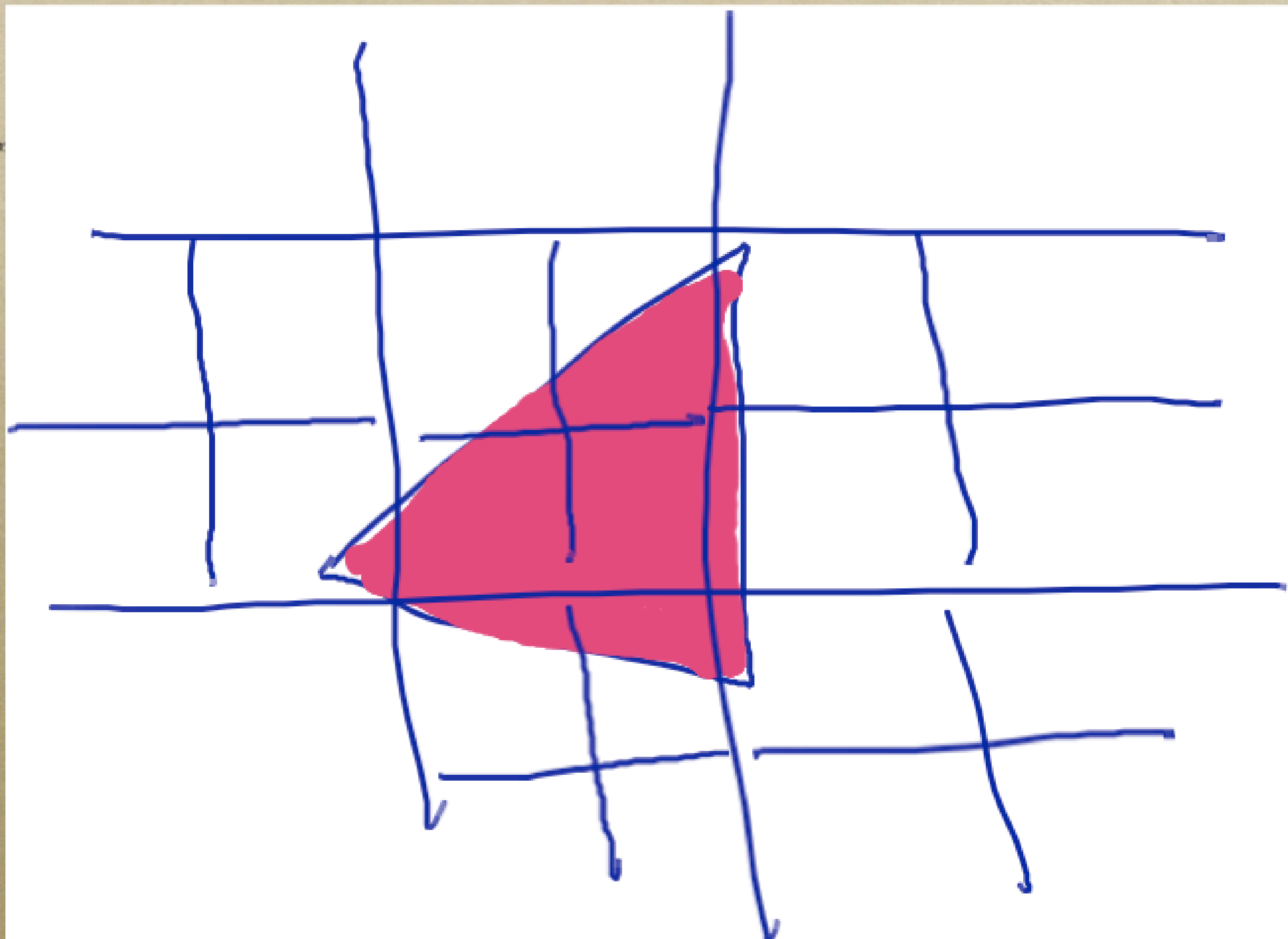
- *This decomposition is what we were using for A* in examples from last class*
- *Works pretty well except:*
  - *need high resolution near obstacles*
  - *want low res away from obstacles*

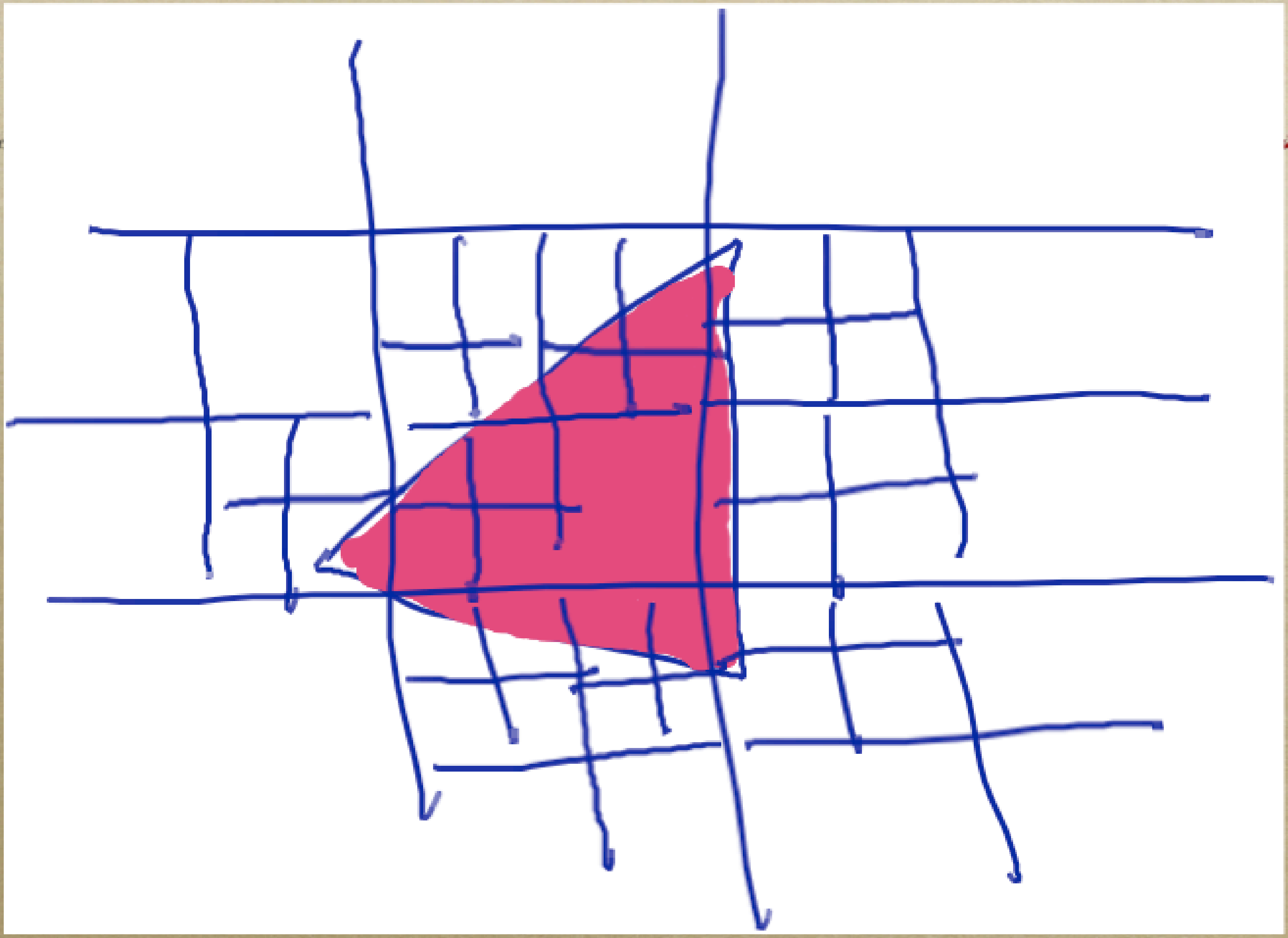
Fix: variable resolution

- *Lay down a coarse grid*
- *Split cells that intersect obstacle borders*
    - *empty cells good*
    - *full cells also don’t need splitting*
- *Stop at fine resolution*
- *Data structure: quadtree*









Discussion

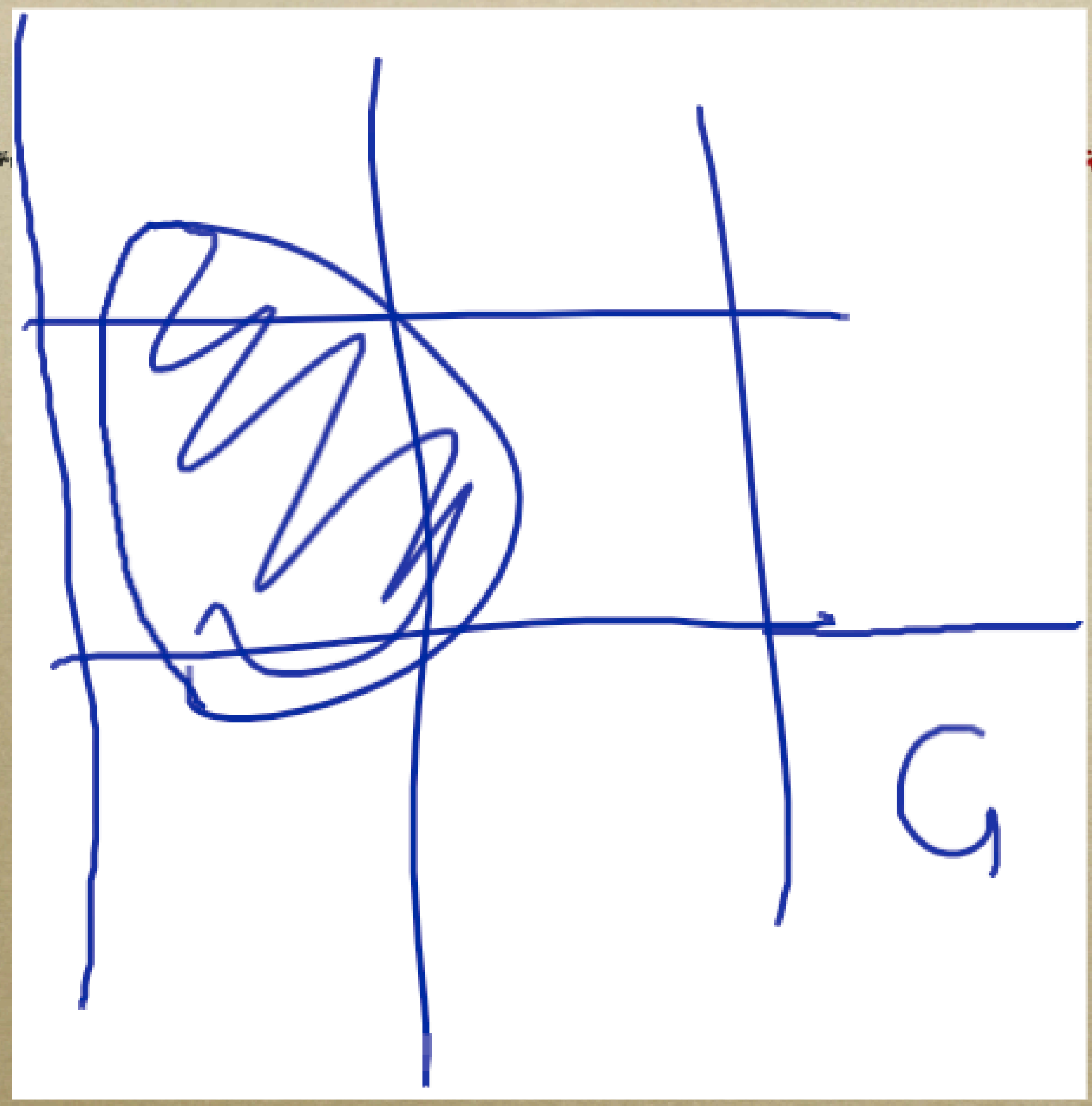
- *Works pretty well, except:*
  - *Still don't know when to stop*
  - *Won't find shortest path*
  - *Still doesn't really scale to high-d*

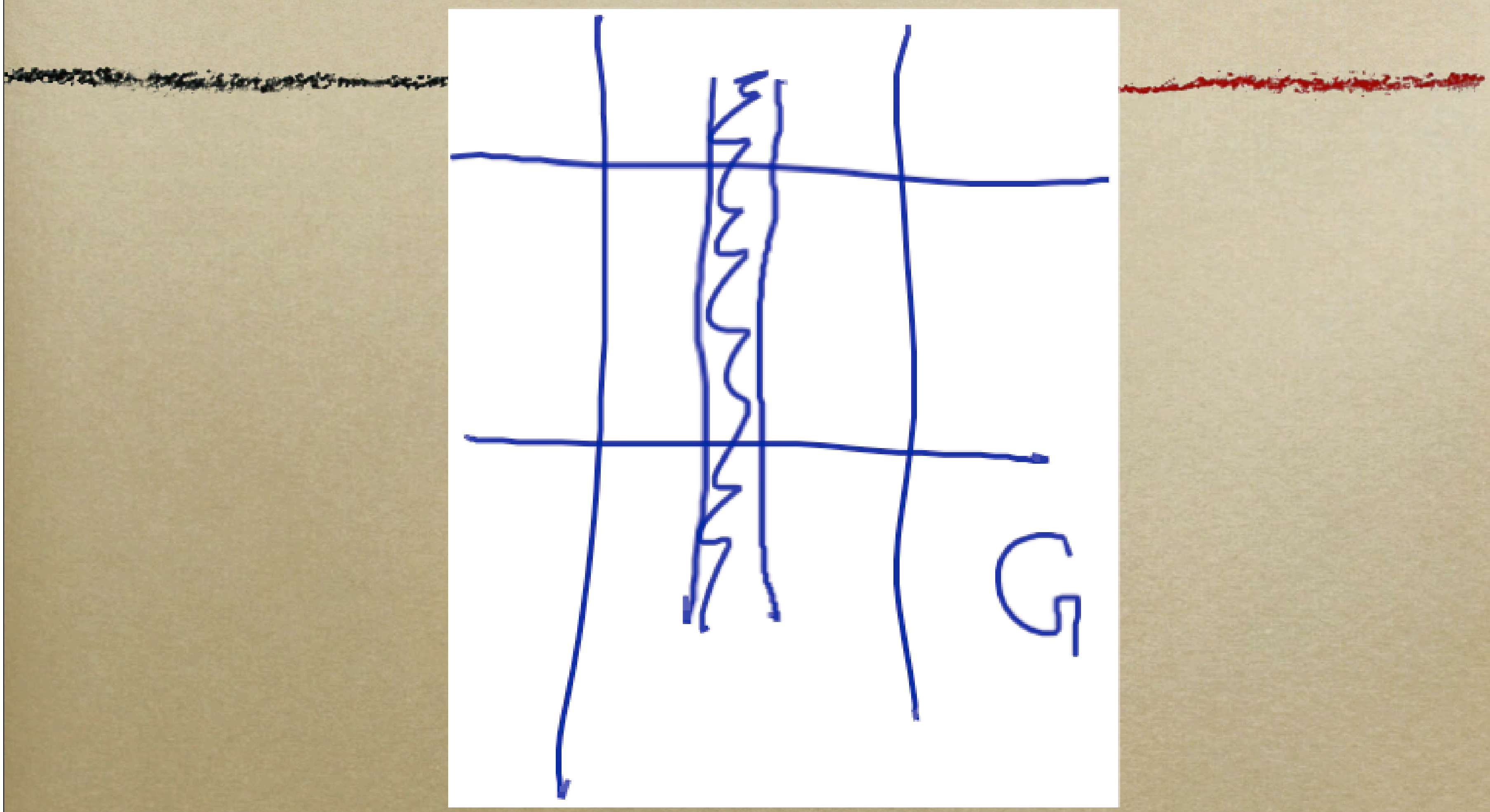
Better yet

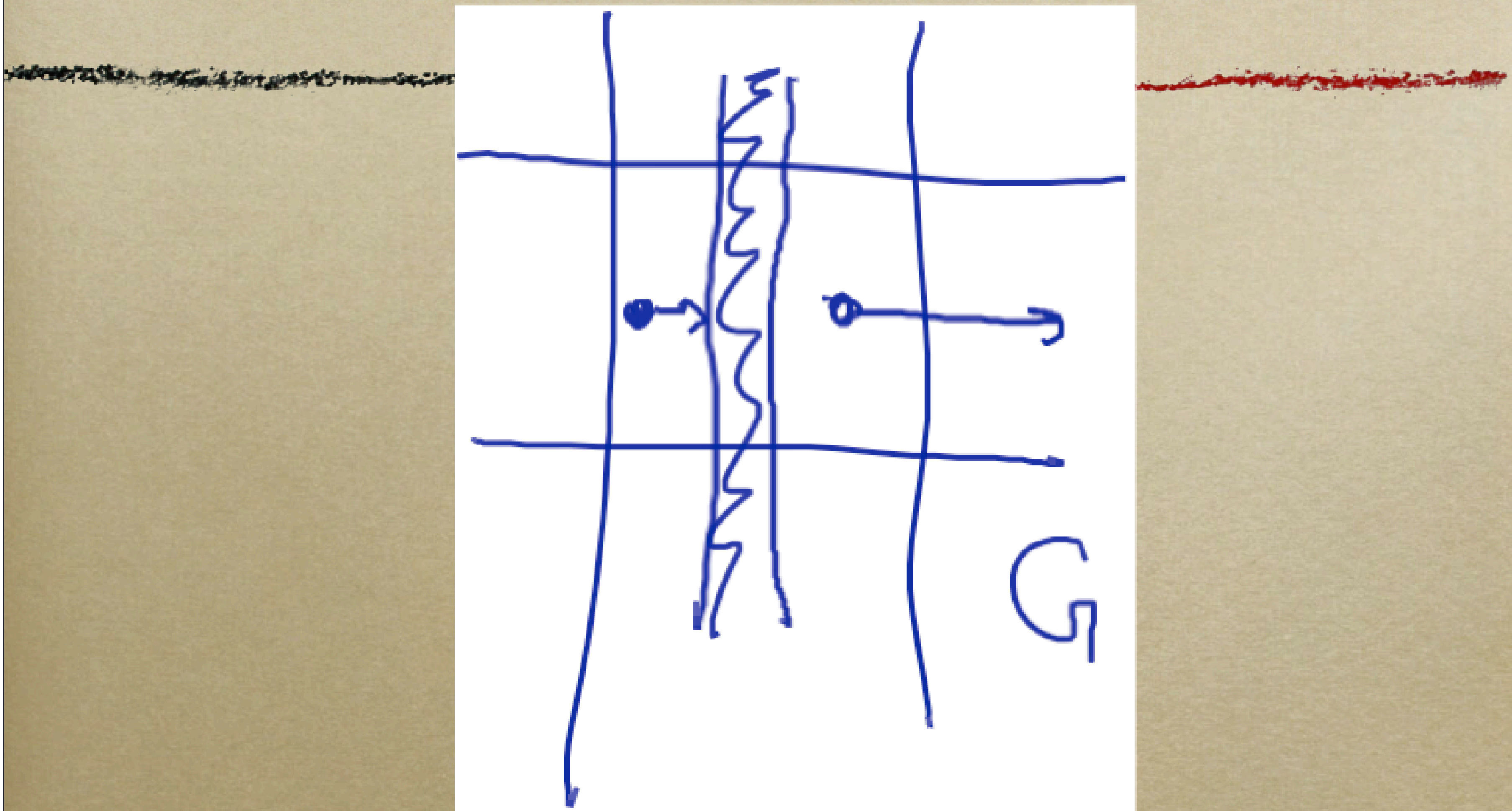
- *Adaptive decomposition*
- *Split only cells that actually make a difference*
  - *are on path from start*
  - *make a difference to our policy*

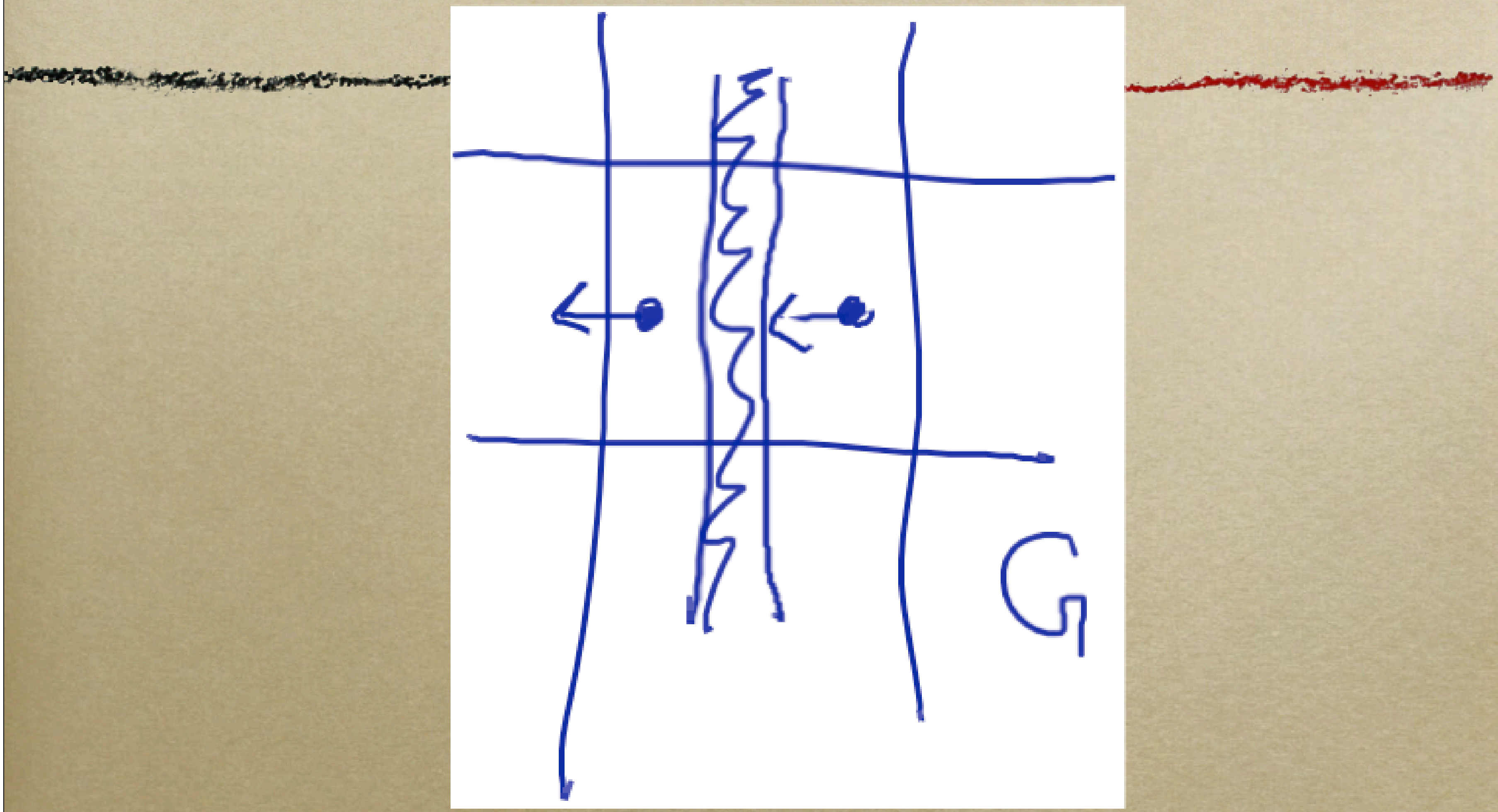
Parti-game paper

- *Andrew Moore and Chris Atkeson*. The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces
- *http://www.autonlab.org/autonweb/14699.html*


G


G

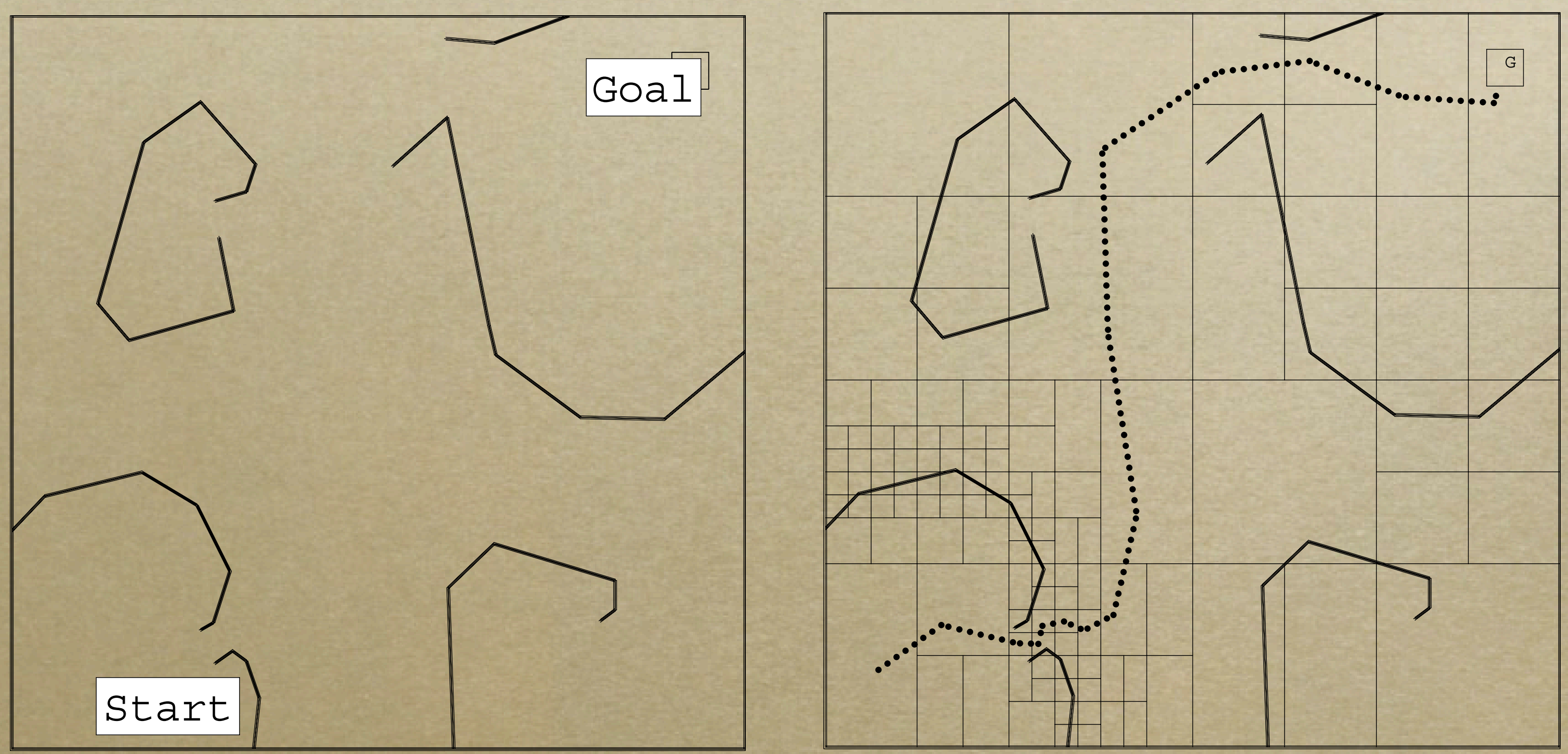

G


G

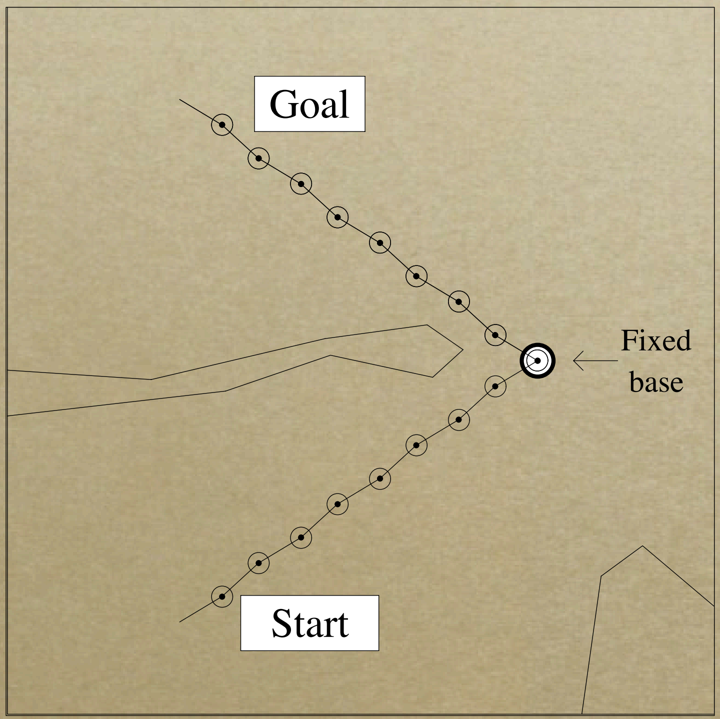
Parti-game algorithm

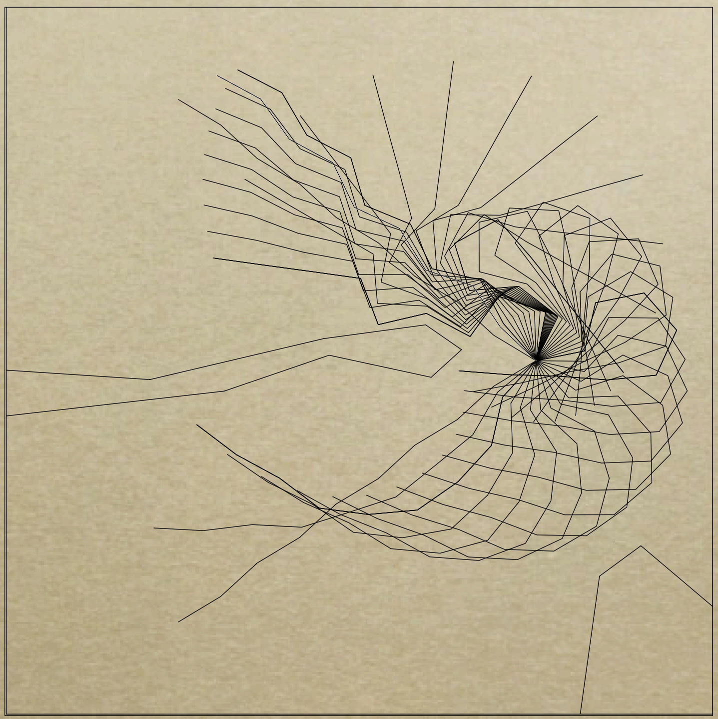
- *Try actions from several points per cell*
- *Try to plan a path from start to goal*
- *On the way, pretend an opponent gets to choose which outcome happens (out of all that have been observed in this cell)*
- *If we can get to goal, we win*
- *Otherwise we can split a cell*

Parti-game example



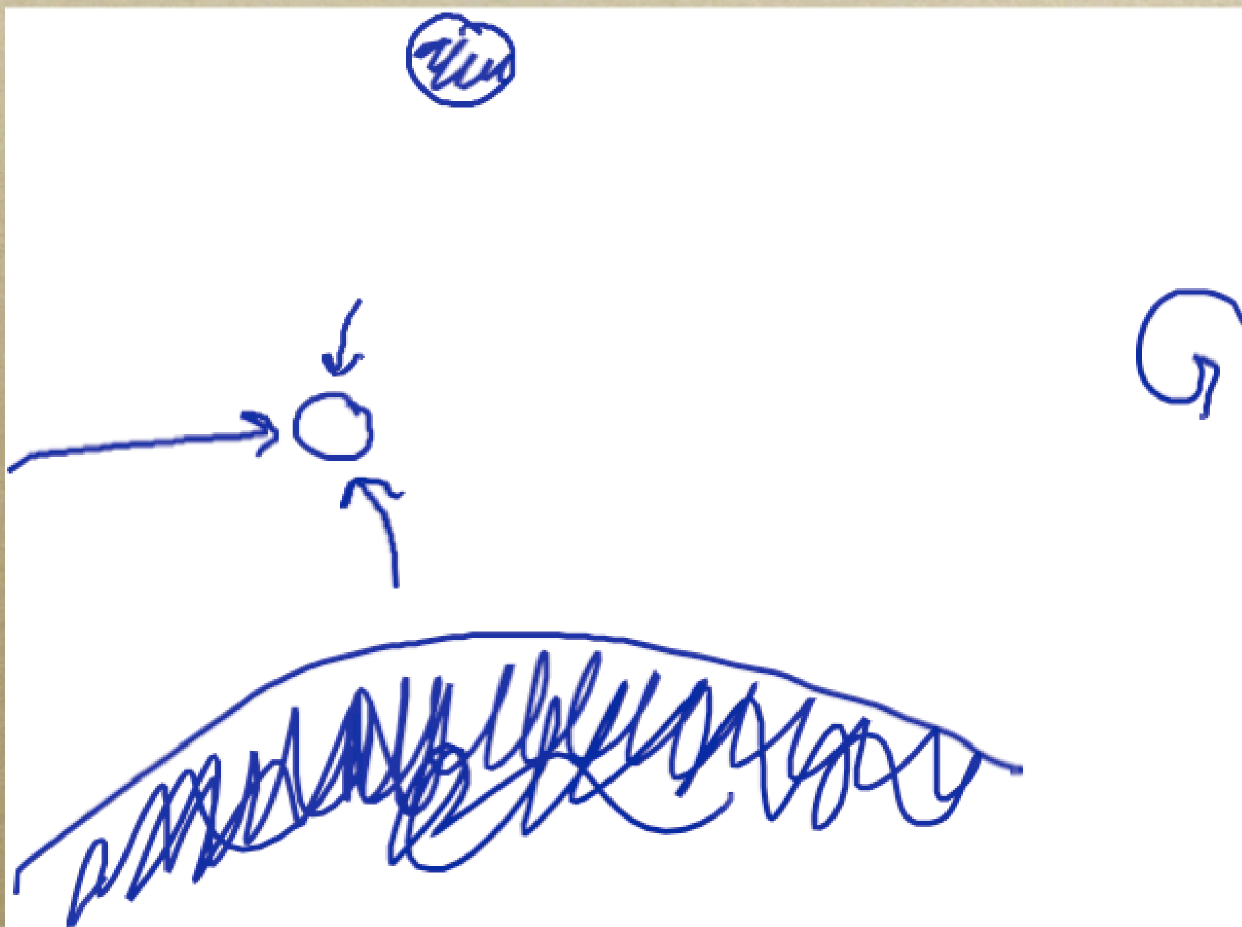
9dof planar arm





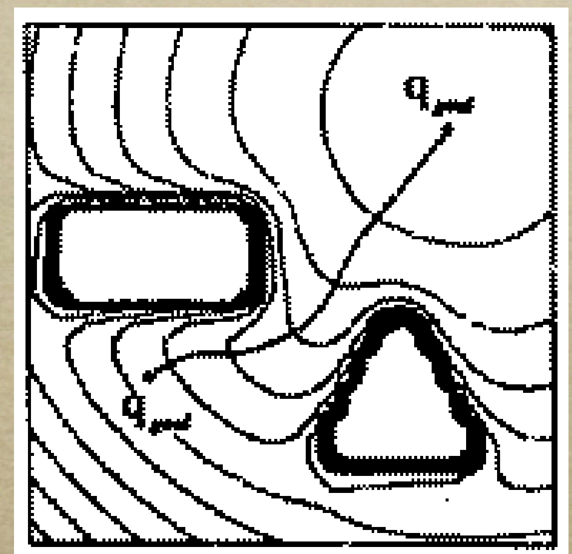
85 partitions total

Potential-based algorithms



Potential-based algorithms

- *Add a fictitious force that moves us away from obstacles*
  - *stronger when closer*
- *Add a force towards goal*
- *Local minima galore …*
- *Or, expensive but cool ways to calculate potentials that don't have local minima*



Randomness in search

We can be very lucky or unlucky


The GWYDYR CASTLE in the Doldrums, while sharks hang around her, awaiting the next offering of galley slops

Doldrums: One Of Murphy's Yarns

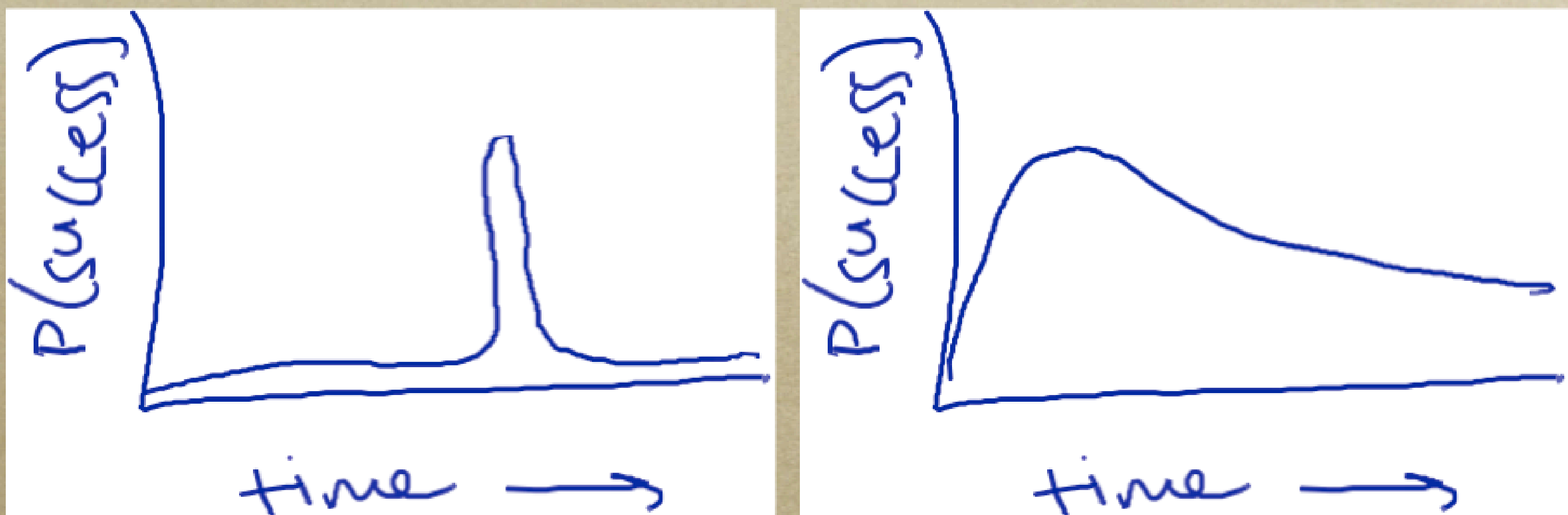
http://oldpoetry.com/opoem/56157 **Cicely Fox Smith**

"I heard onst of a barque," said Murphy.
"Becalmed, that couldn't get a breath,
Till all the crowd was sick with scurvy
An' the skipper drunk himself to death."

Simple idea

- *Try multiple starting points, random seeds for order of expanding neighbors*
- *Interleave computation (or iterative lengthening)*
- *When does this work?*

Randomization cont'd



- *Randomization works well if search times are sometimes short but have heavy tail*

RRTs

- *We will come back to randomness for more planning algorithms later*
- *For now, here’s a randomized way of dividing up C-space that seems to work quite well in high-dimensions*
- *Rapidly-exploring Random Trees*

RRTs

- *Put landmarks into C-space*
- *Break up C-space into Voronoi regions around landmarks*
- *Put landmarks densely only if high resolution is needed to find a path*
- *Will not guarantee optimal path*

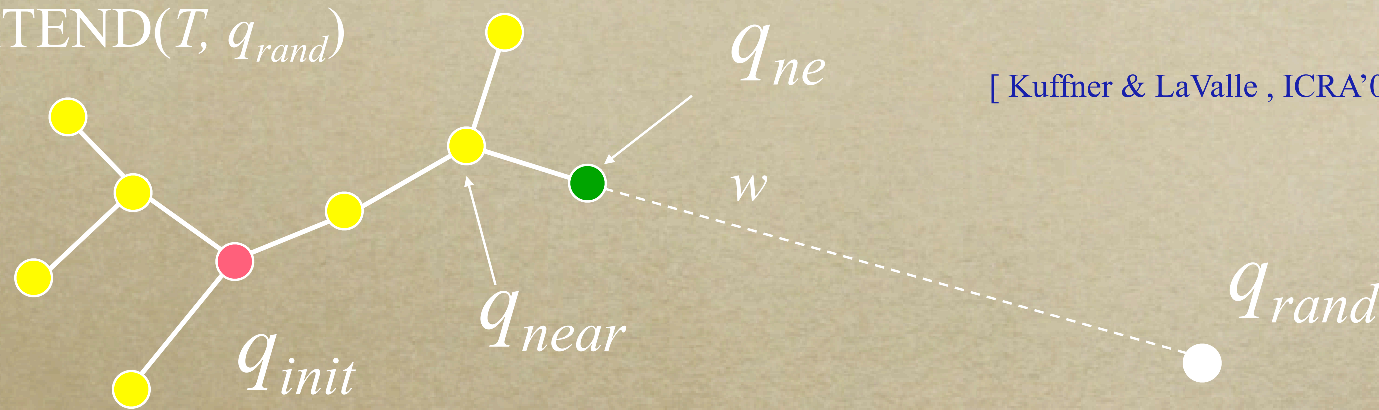
RRT assumptions

- *RANDOM_CONFIG*
  - *samples from some distribution on C-space; can use to bias search*
- *EXTEND(***q***,* **q***')*
  - *uses a local controller to head towards* **q***' from* **q**
  - *stops before hitting obstacle*
- *FIND_NEAREST(***q***, Q)*

Path Planning with RRTs

RRT = Rapidly-Exploring Random Tree

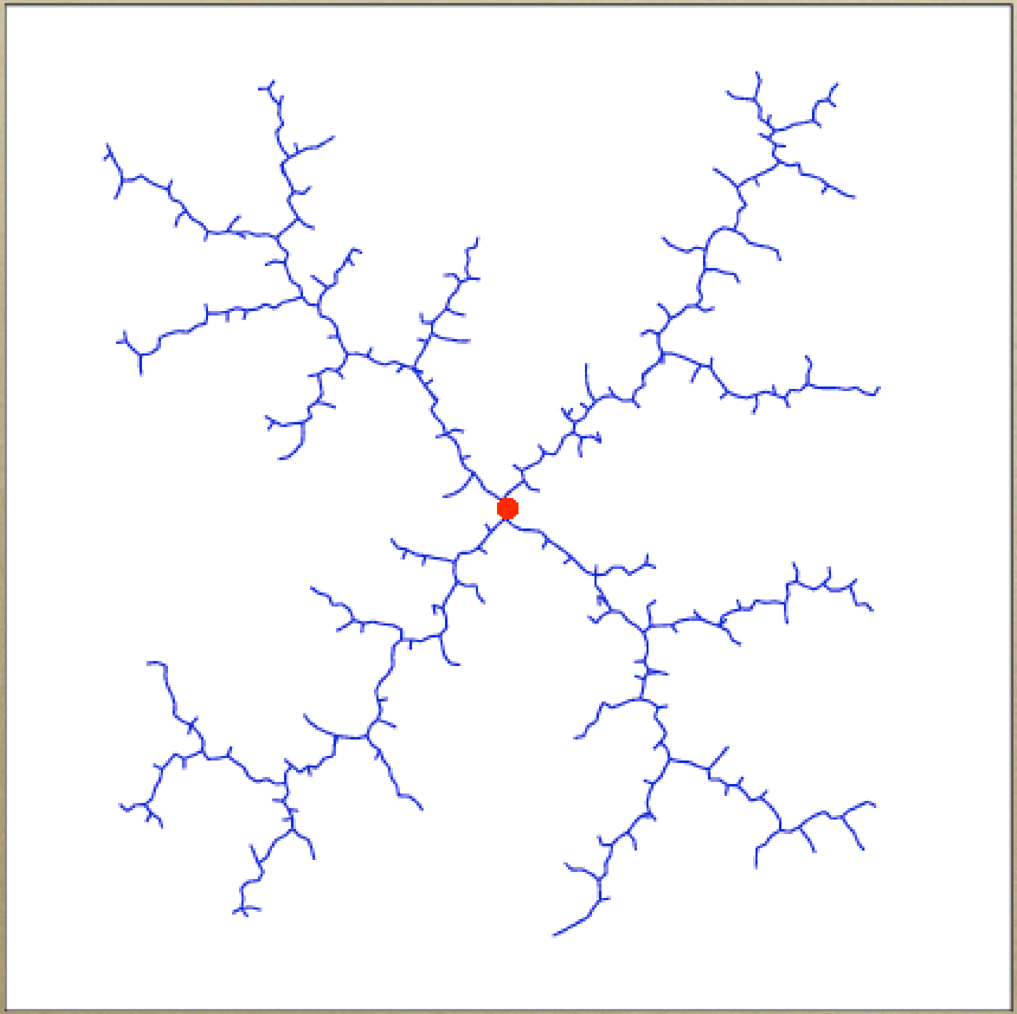
EXTEND(T, q_{rand})



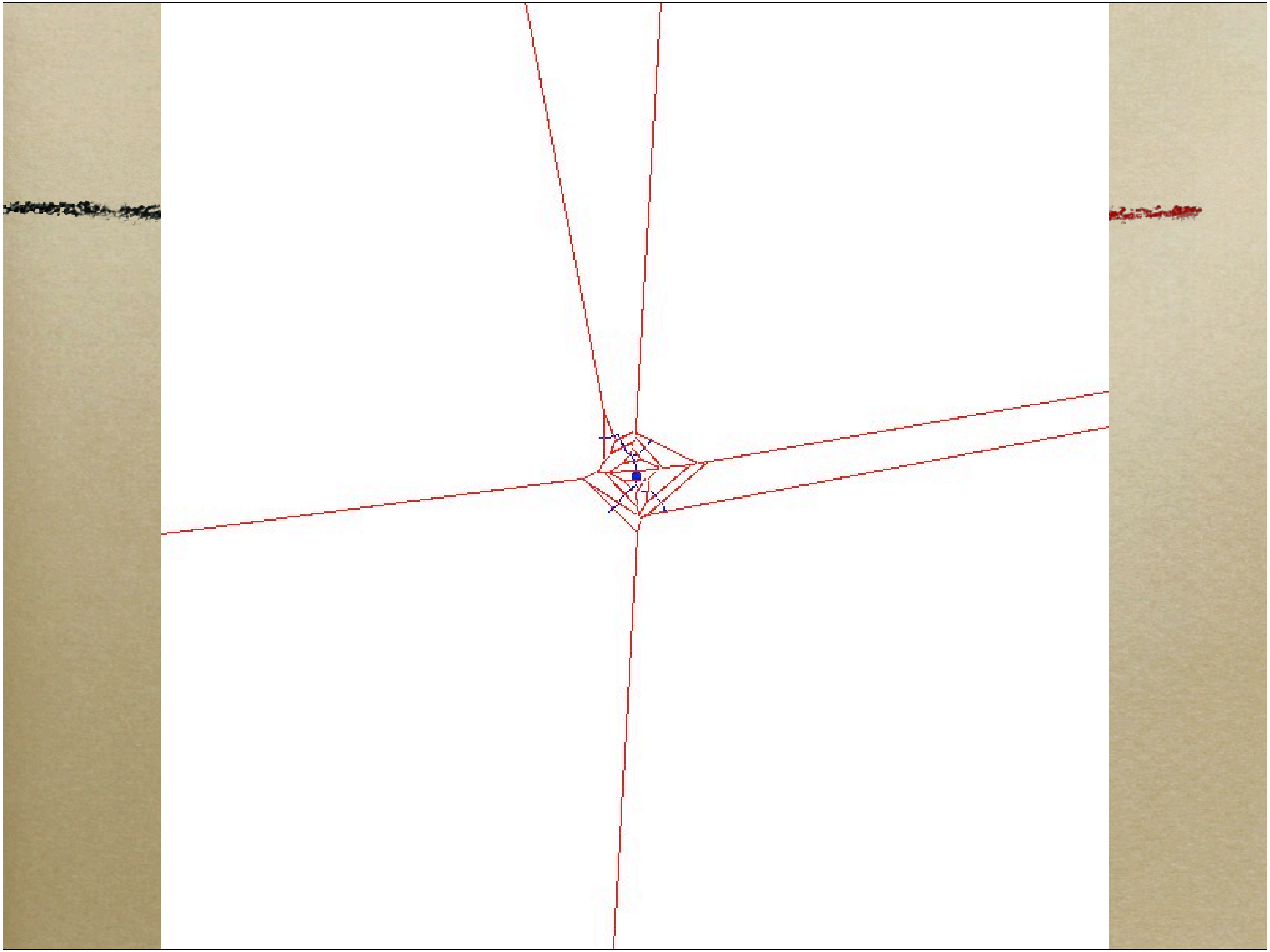
[Kuffner & LaValle , ICRA'00]

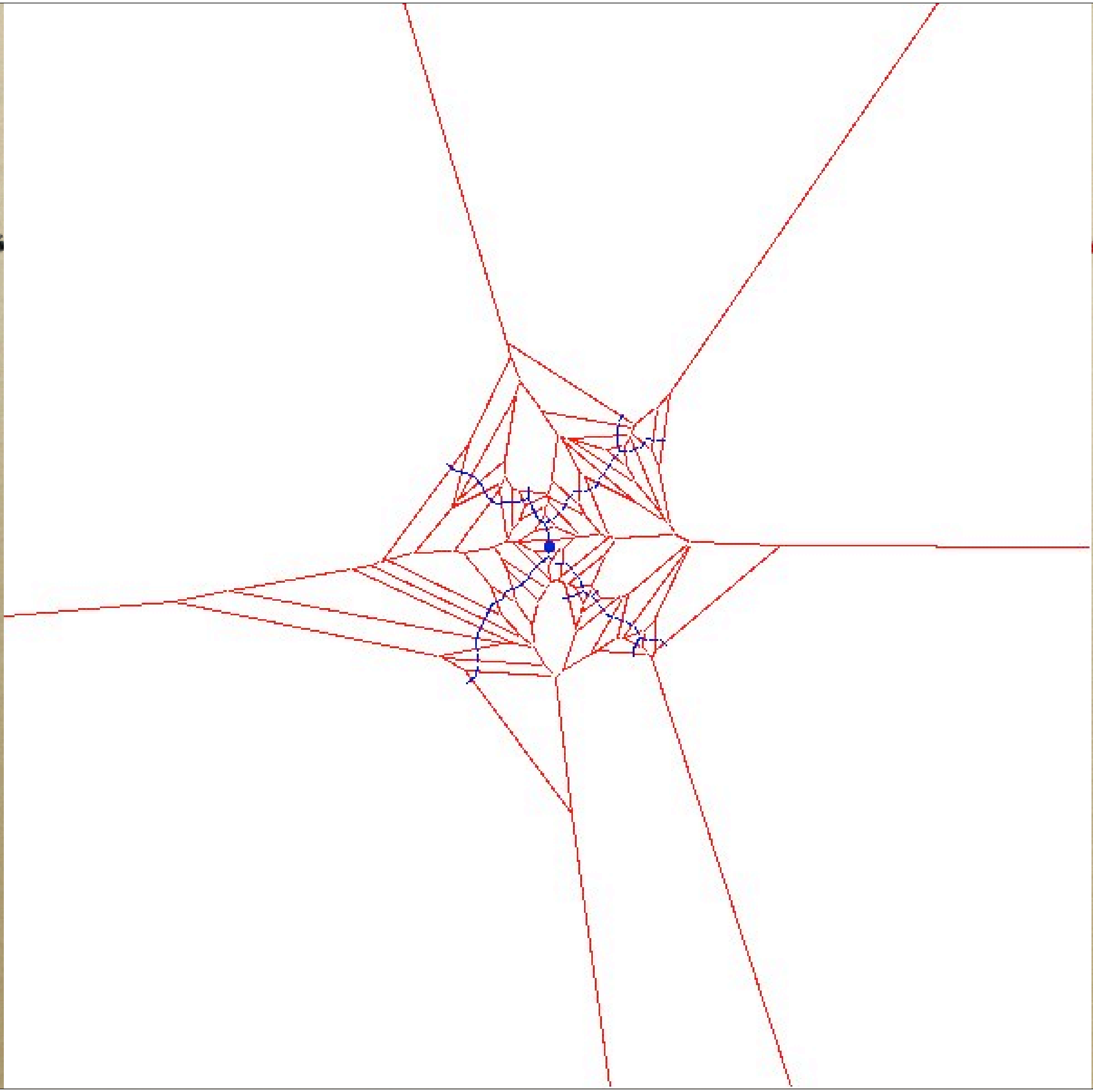
```
BUILD_RRT (q_init)  {
   T.init(q_init);
   for k =  1 to K do
      q_rand = RANDOM_CONFIG();
      EXTEND(T, q_rand)
}
```

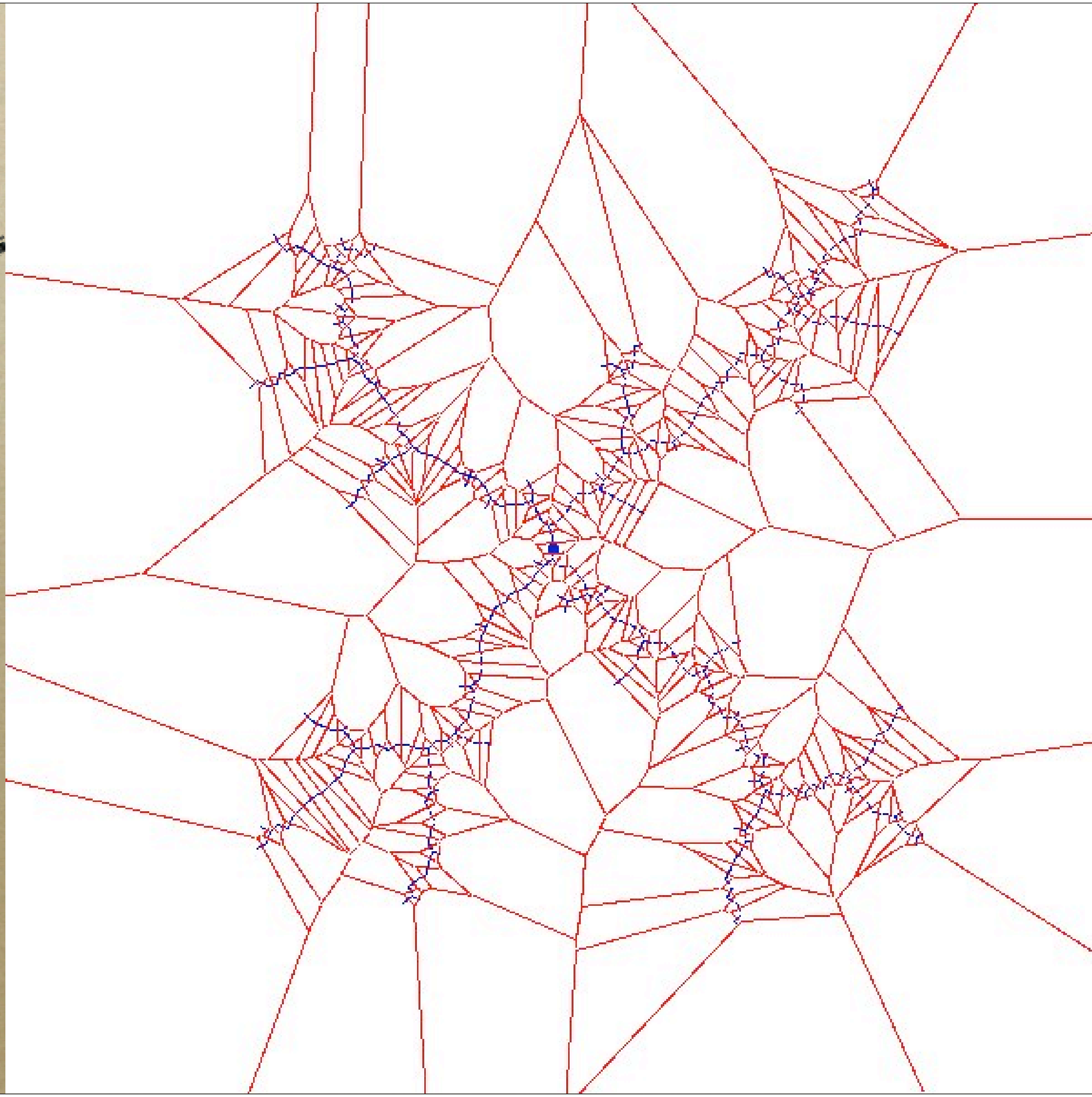
RRT example

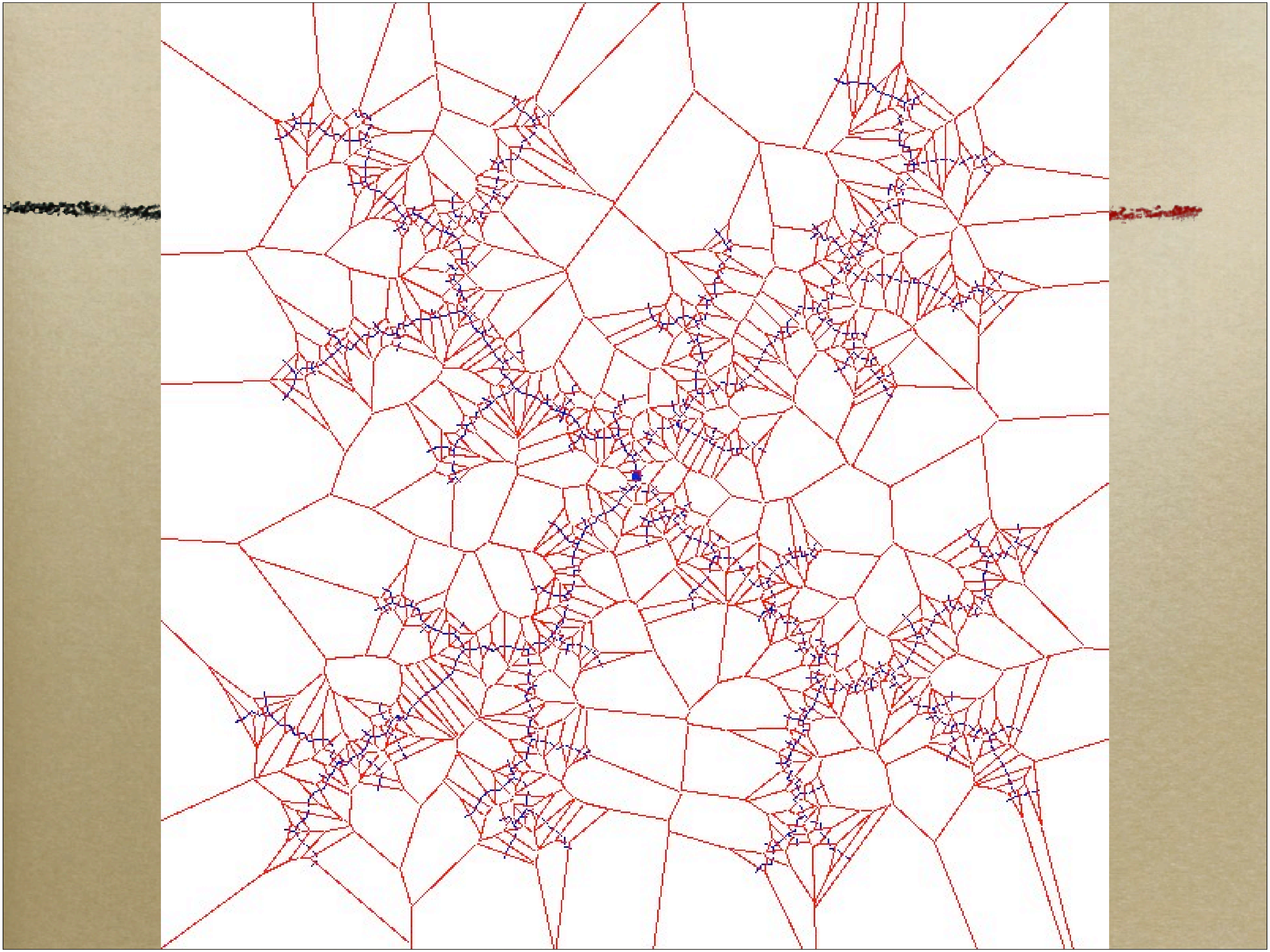


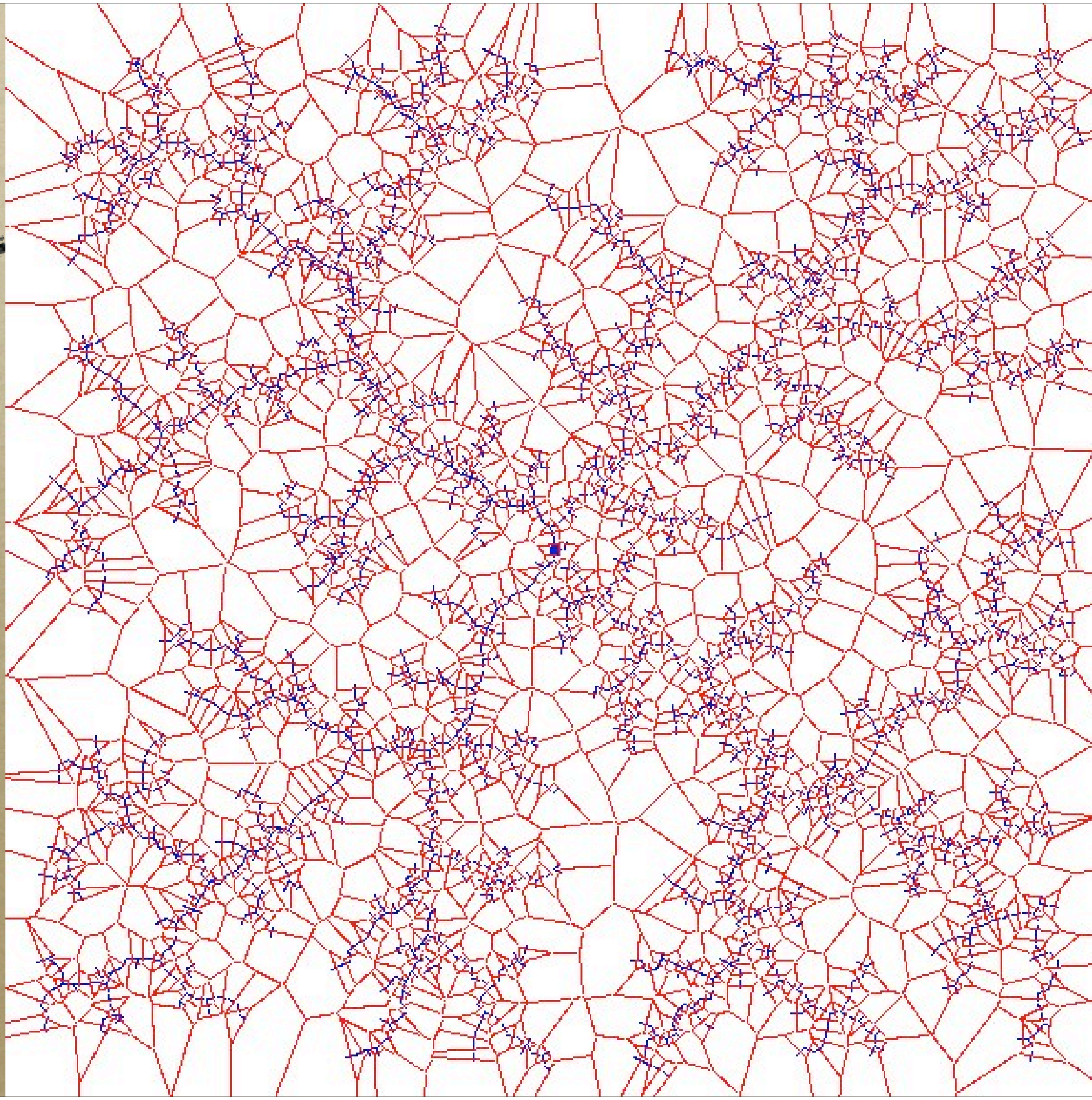
Planar holonomic robot



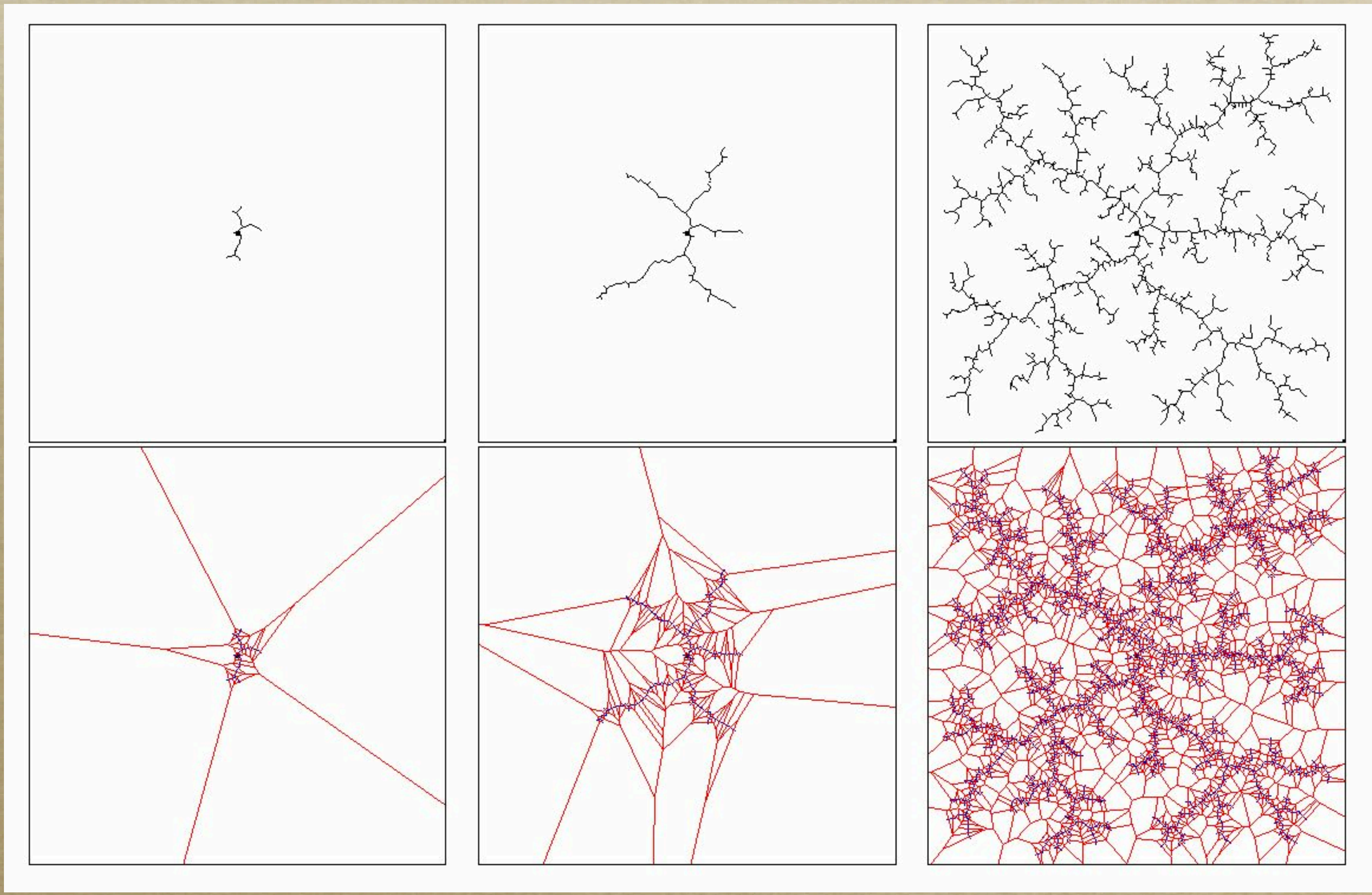




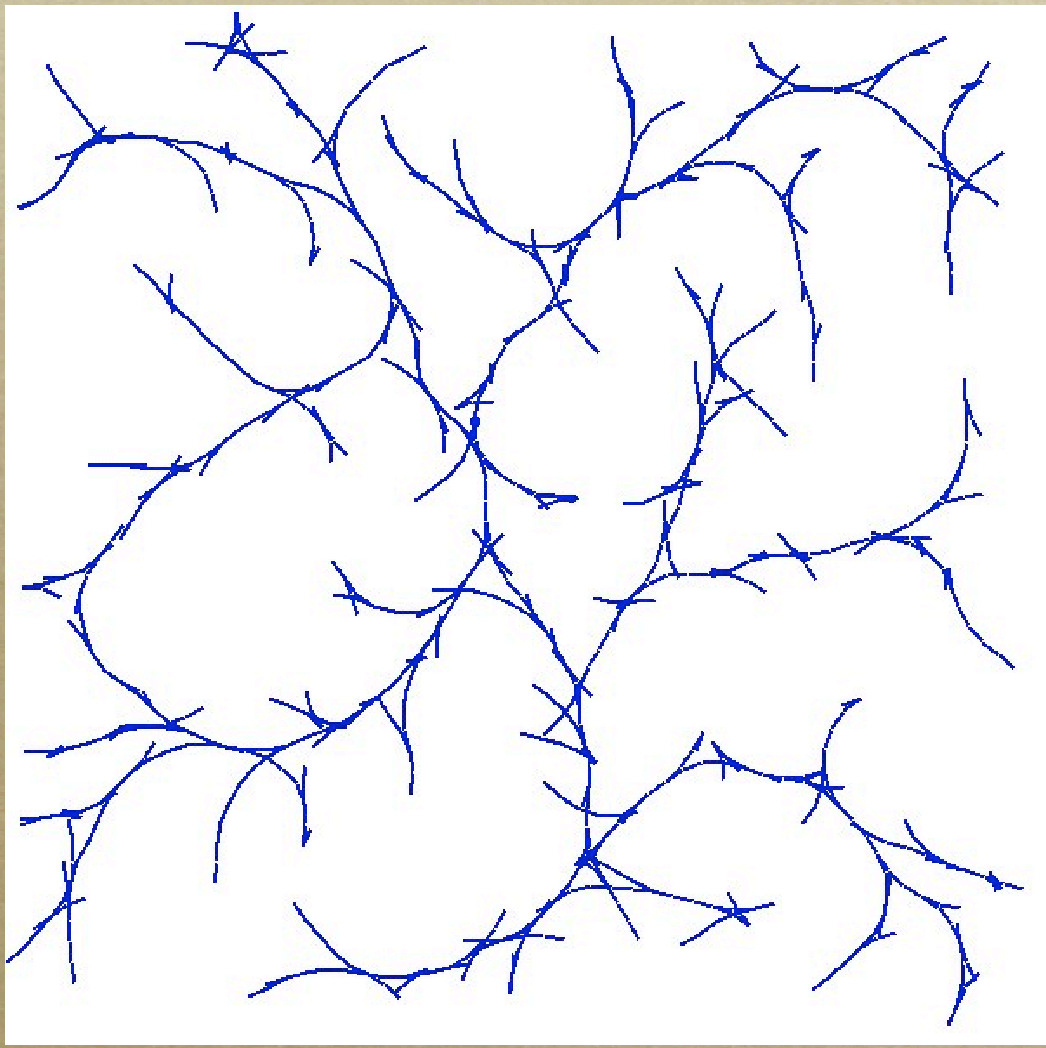




RRT example



RRT for a car (3 dof)

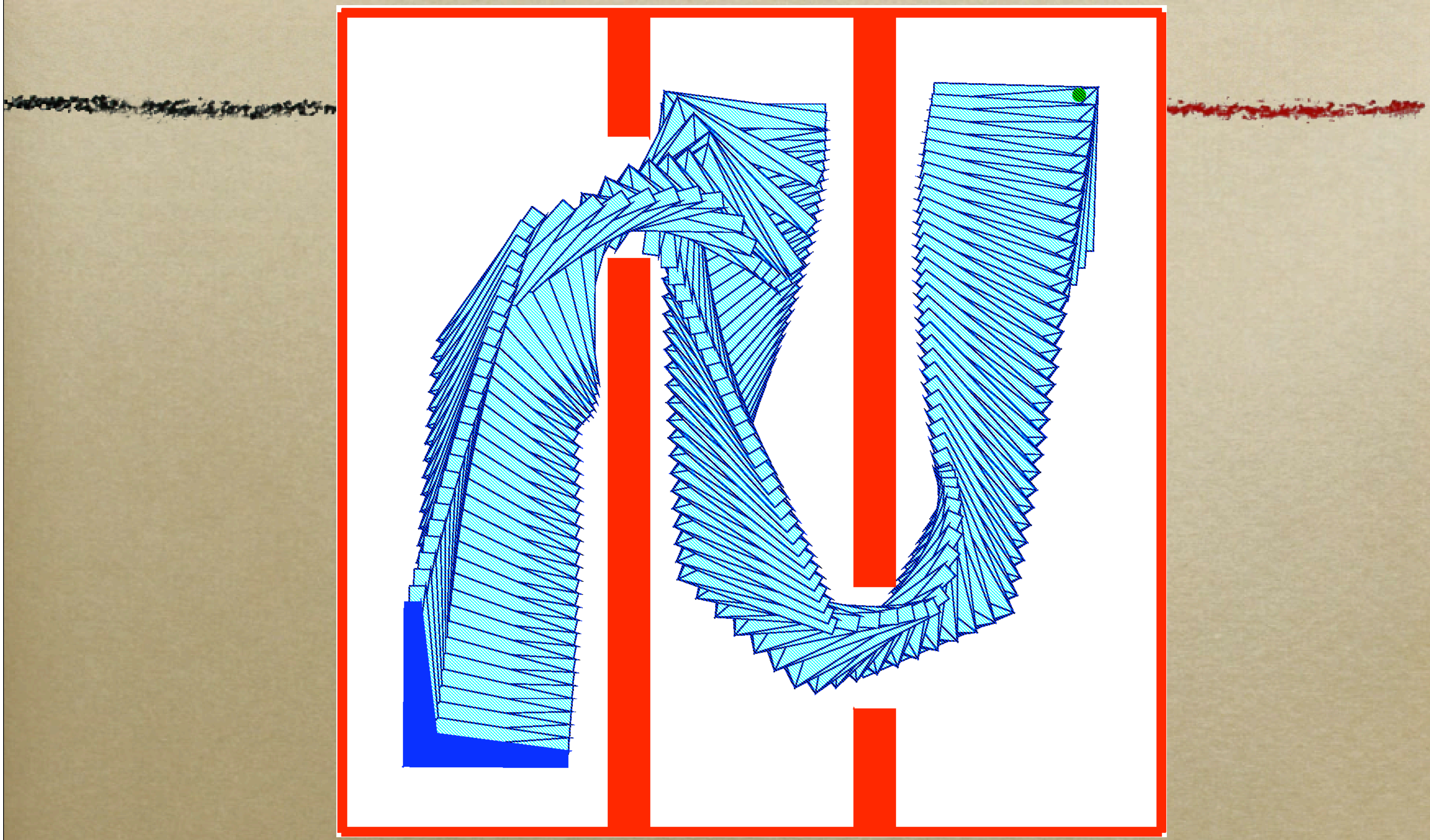


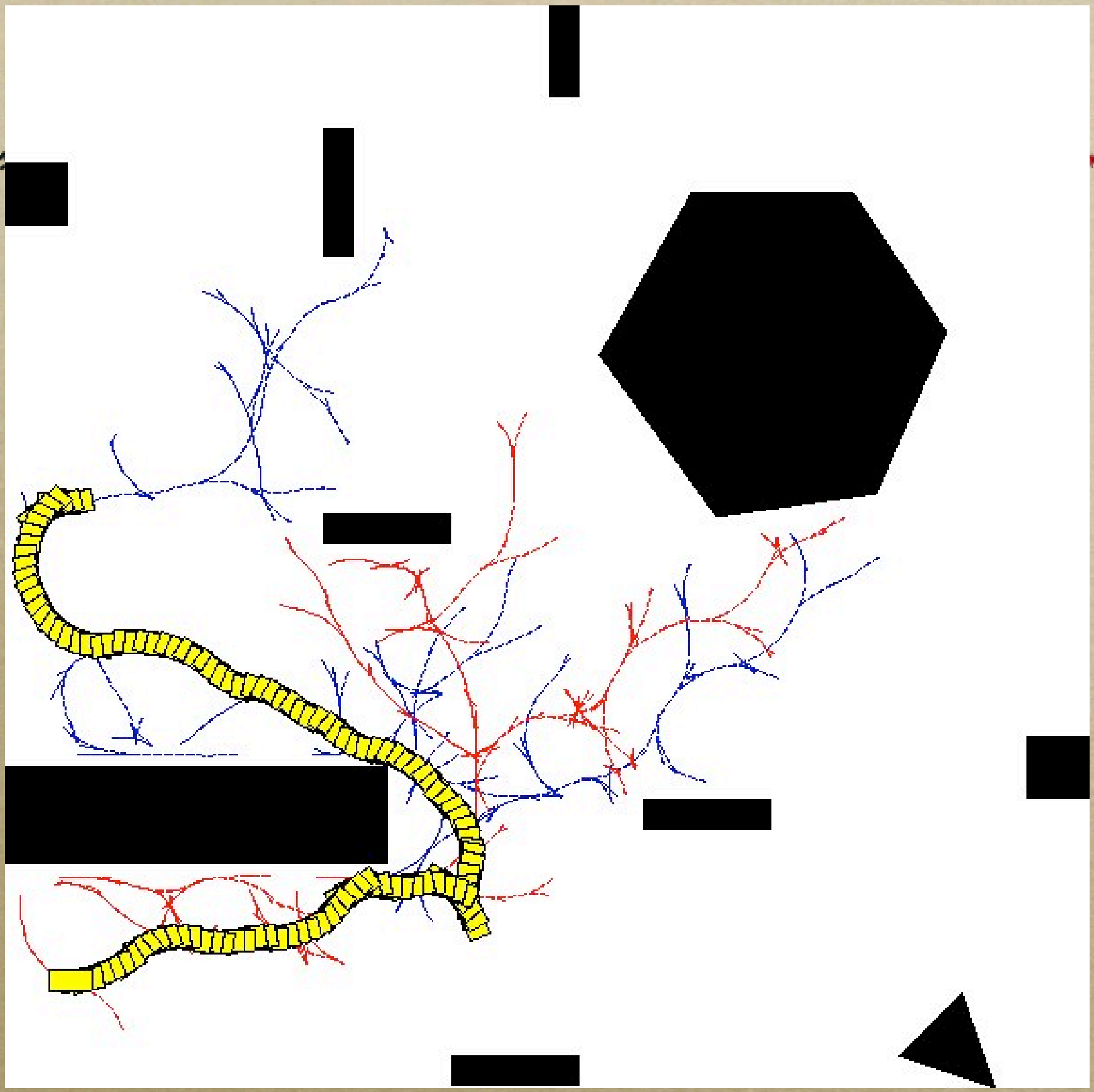
RRTs explore coarse to fine

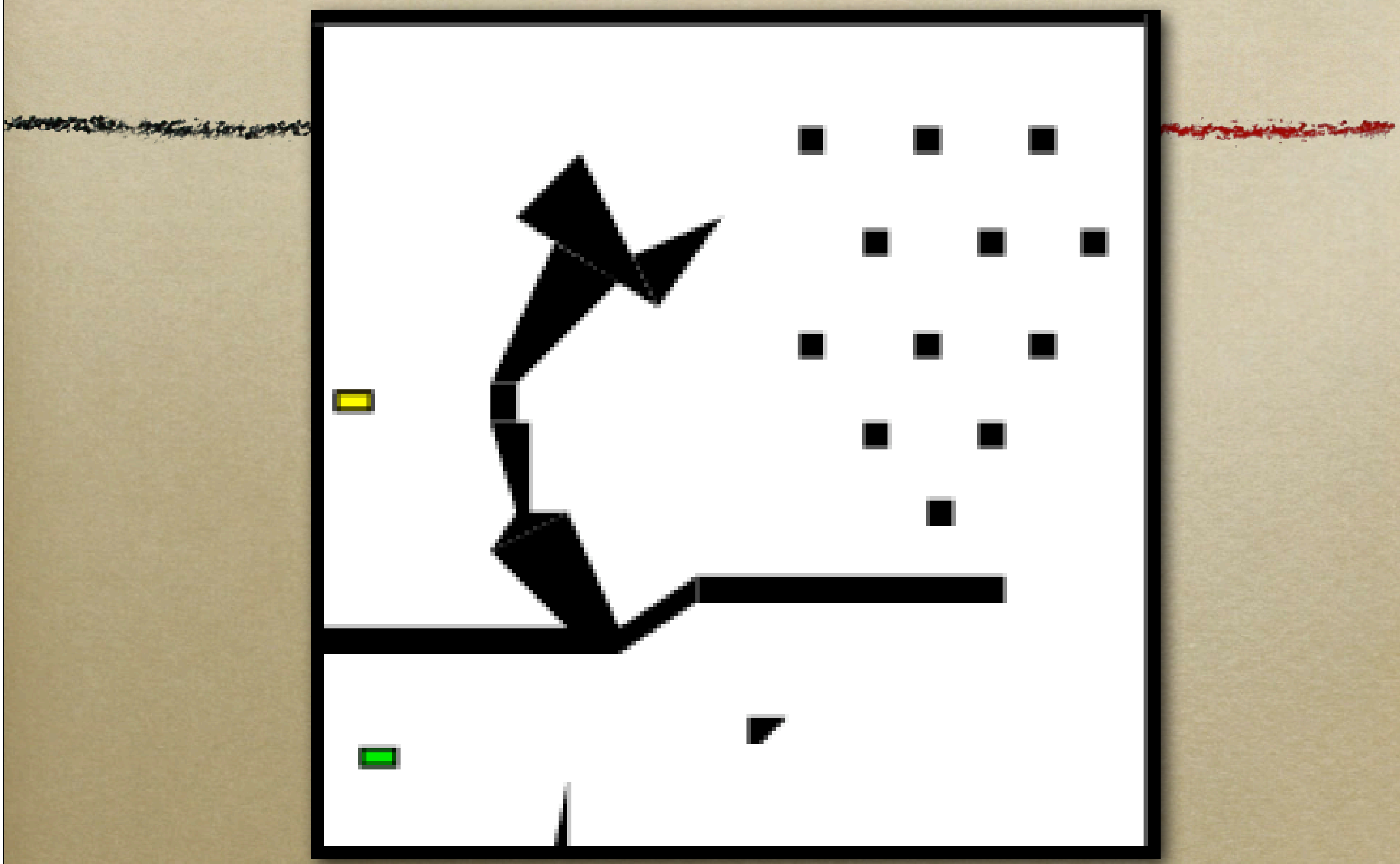
- *Tend to break up large Voronoi regions*
- *Limiting distribution of vertices is RANDOM_CONFIG*
  - *Key idea in proof: as RRT grows, probability that qrand is reachable with local controller (and so immediately becomes a new vertex) approaches 1*
- *In limit, we get a Voronoi cell decomposition*

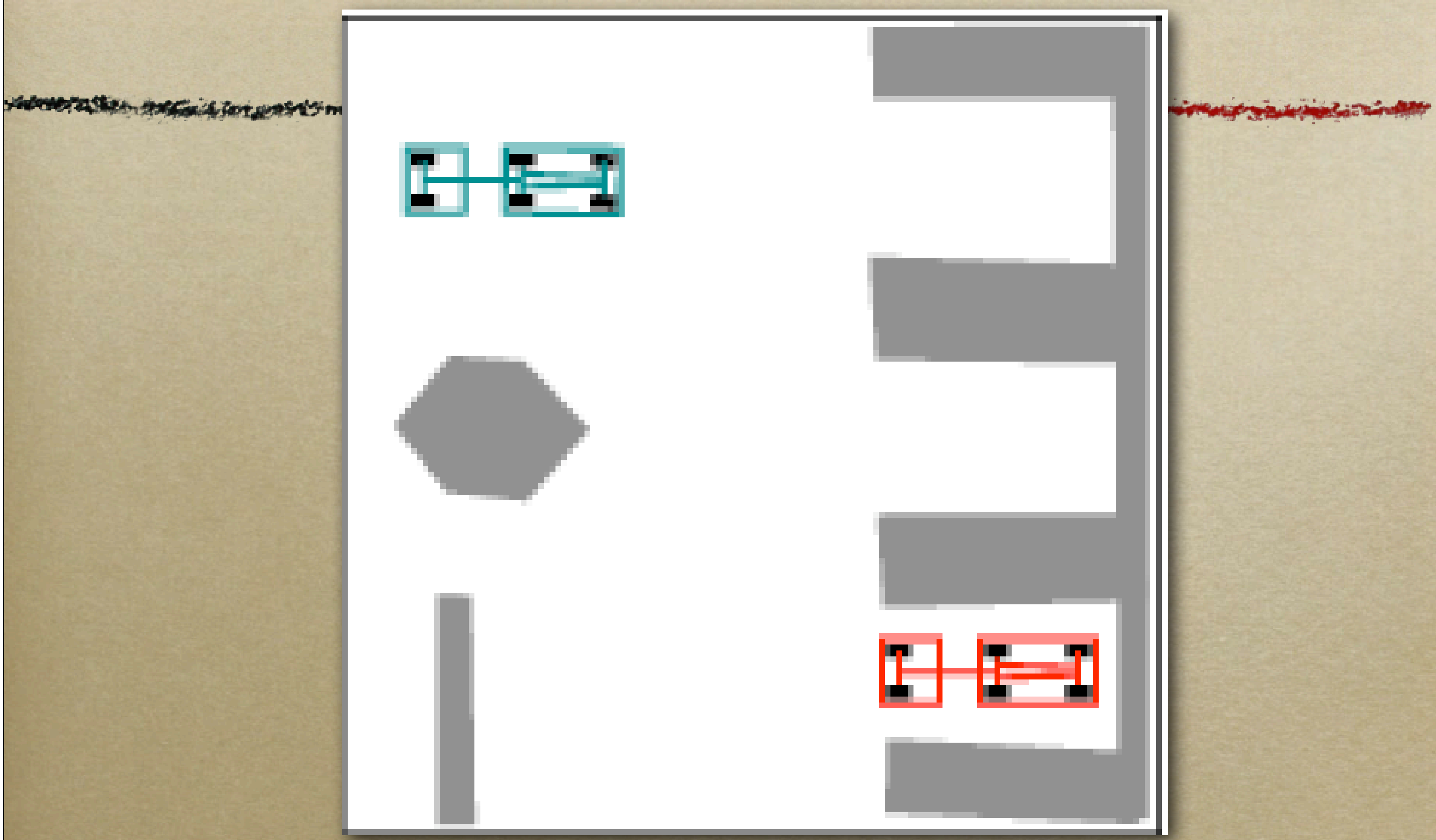
Planning with RRTs

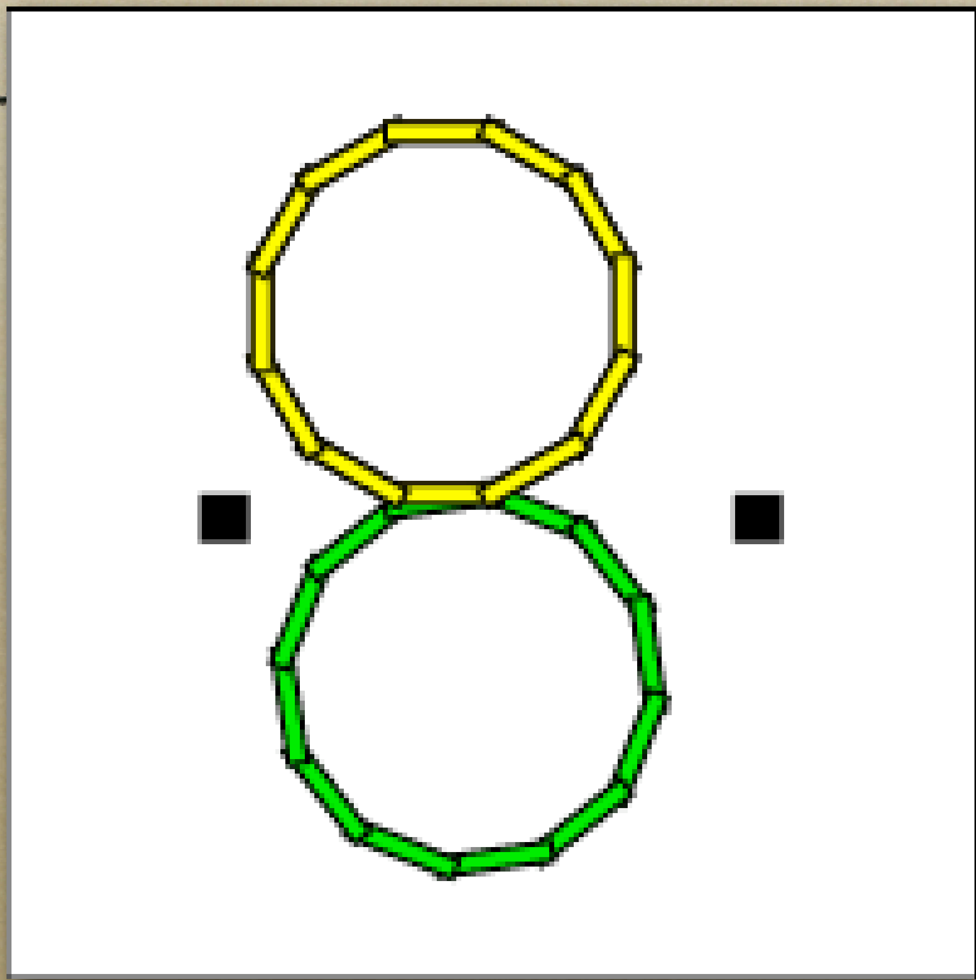
- *Build RRT from start until we add a node that can reach goal using local controller*
- *(Unique) path: root → last node → goal*
- *Optional: cross-link tree by testing local controller, search within tree using A**
- *Optional: grow forward and backward*











What you should know

- *C-space*
- *Ways of splitting up C-space*
  - *Visibility graph*
  - *Voronoi*
  - *Exact, approximate cell decomposition*
  - *Adaptive cells (quadtree, parti-game)*
- *Potential fields*
- *RRTs*