

Planning with Sampling Roadmaps

Building a path to goal



EECS 398

Intro. to Autonomous Robotics

ME/EECS 567 ROB 510
Robot Modeling and Control

Fall 2018

autorob.org

Michigan EECS 398/567 ROB 510 - autorob.org

Approaches to motion planning

- Bug algorithms: Bug[0-2], Tangent Bug
- Graph Search (fixed graph)
 - Depth-first, Breadth-first, Dijkstra, A-star, Greedy best-first
- **Sampling-based Search (build graph):**
 - **Probabilistic Road Maps, Rapidly-exploring Random Trees**
- Optimization and local search:
 - Gradient descent, Potential fields, Simulated annealing, Wavefront

Where is this?



©2010 Google
©2015 Google

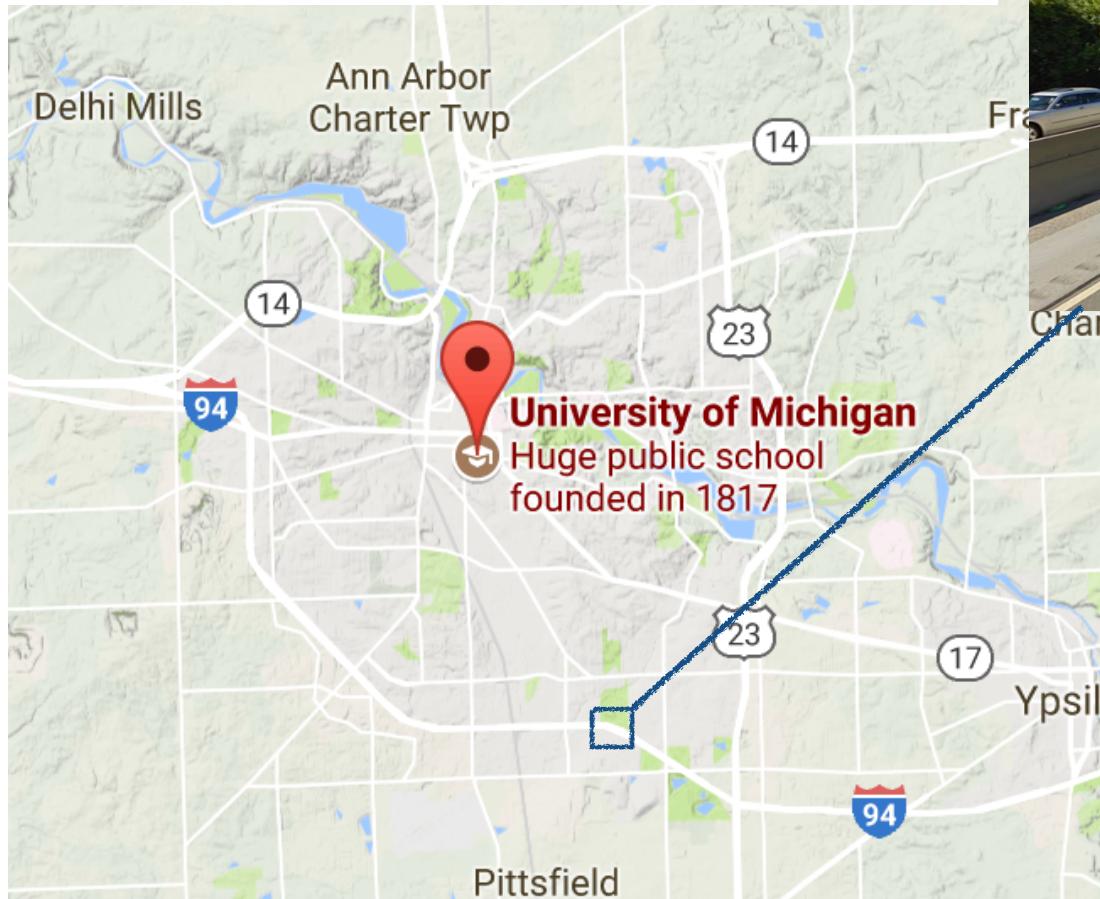
EXIT 177
State St
1 MILE



Configuration (lat,lon): [42.237631, -83.7147289]

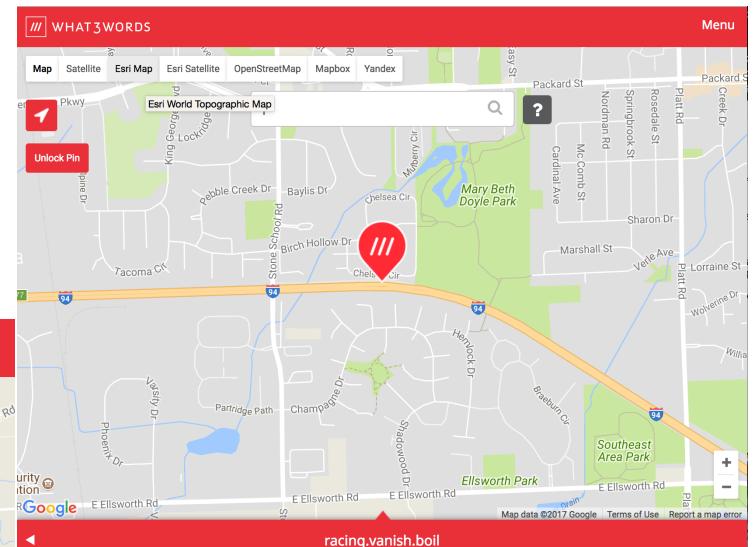
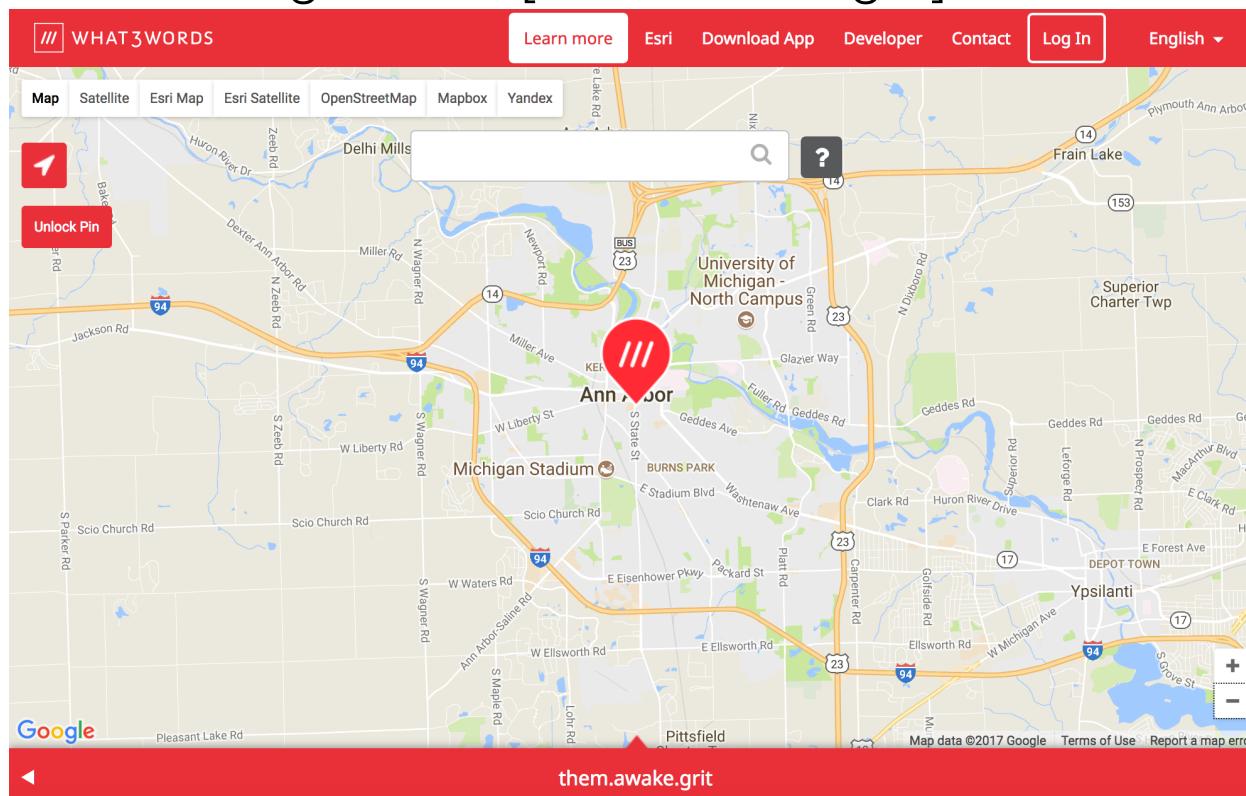
Michigan EECS 398/567 ROB 510 - autordb.org

Configuration: [42.2780436,-83.7404128]



Alternatively...

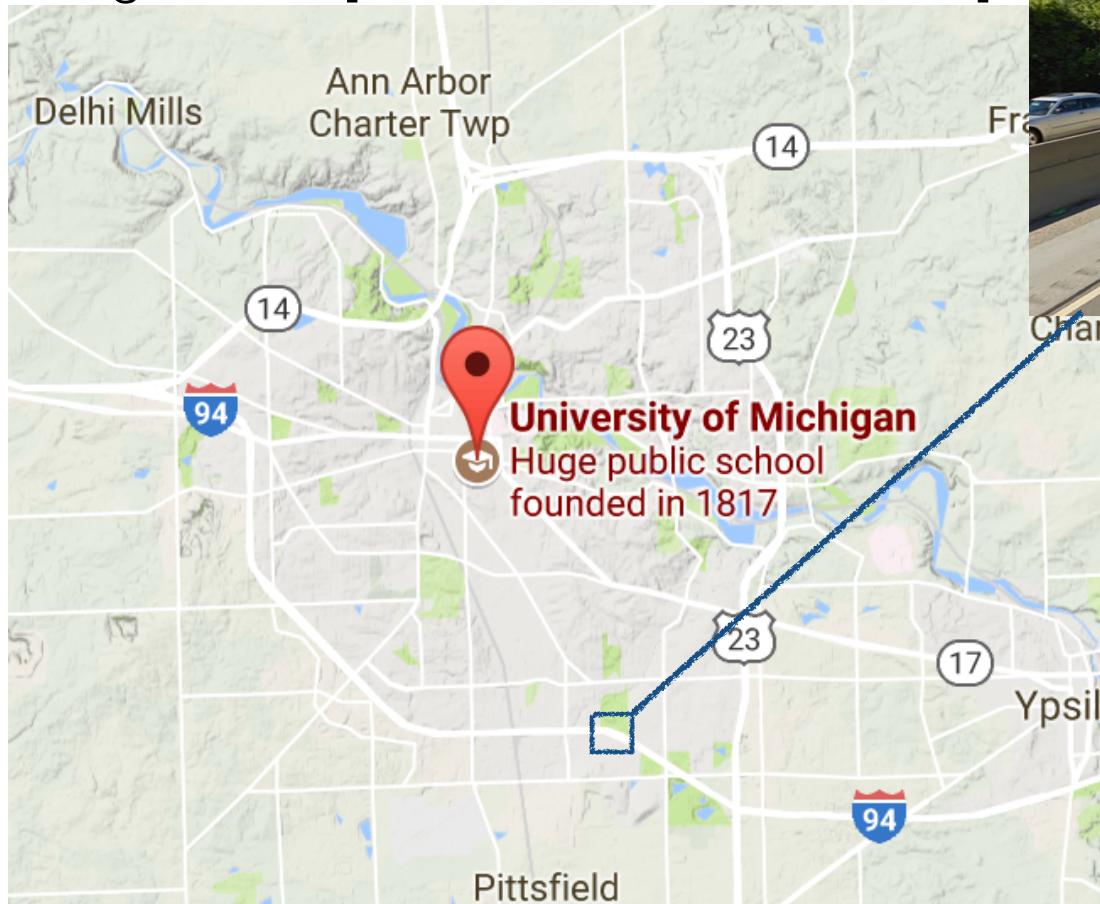
Configuration: [them.awake.grit]

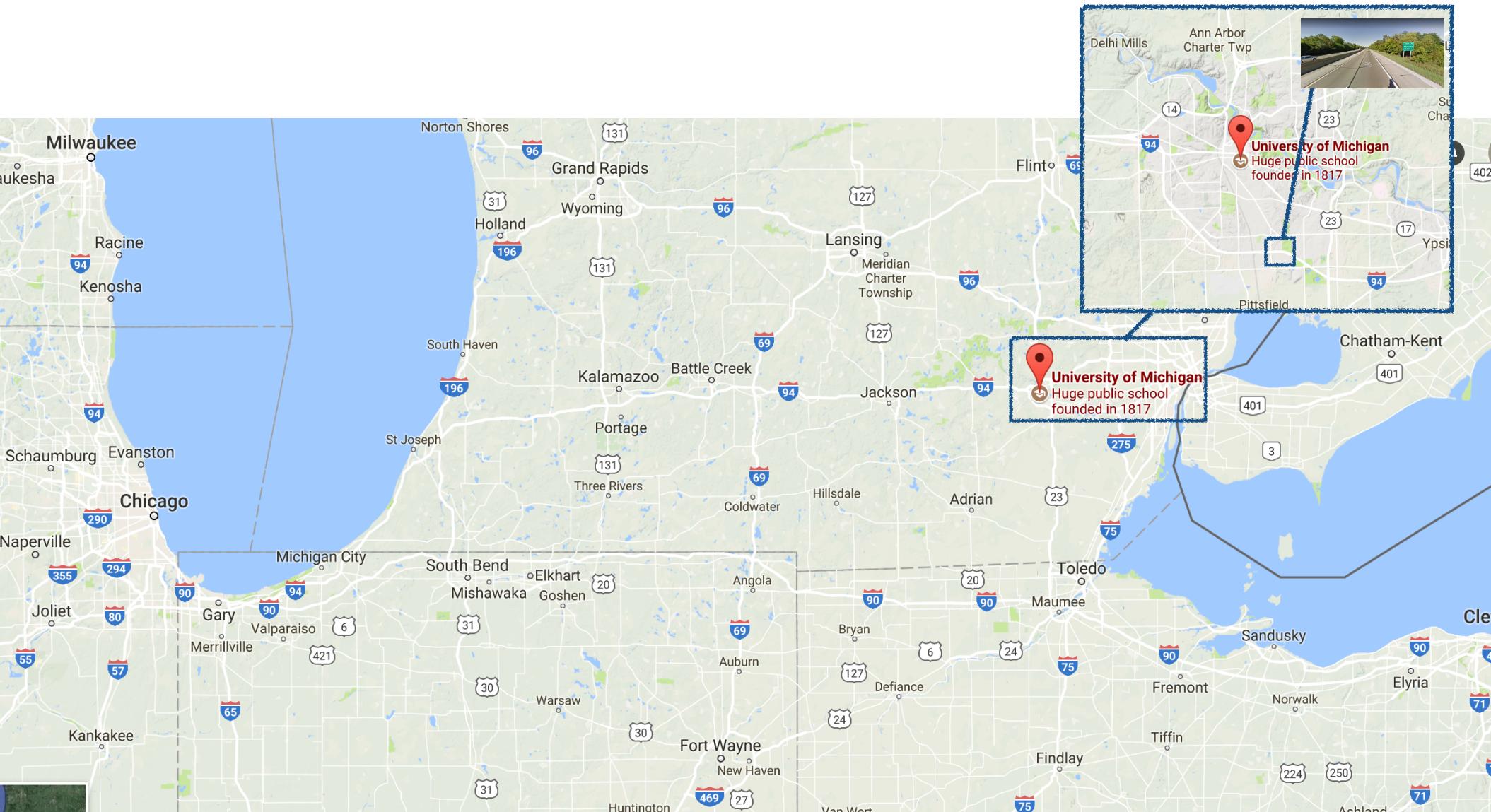


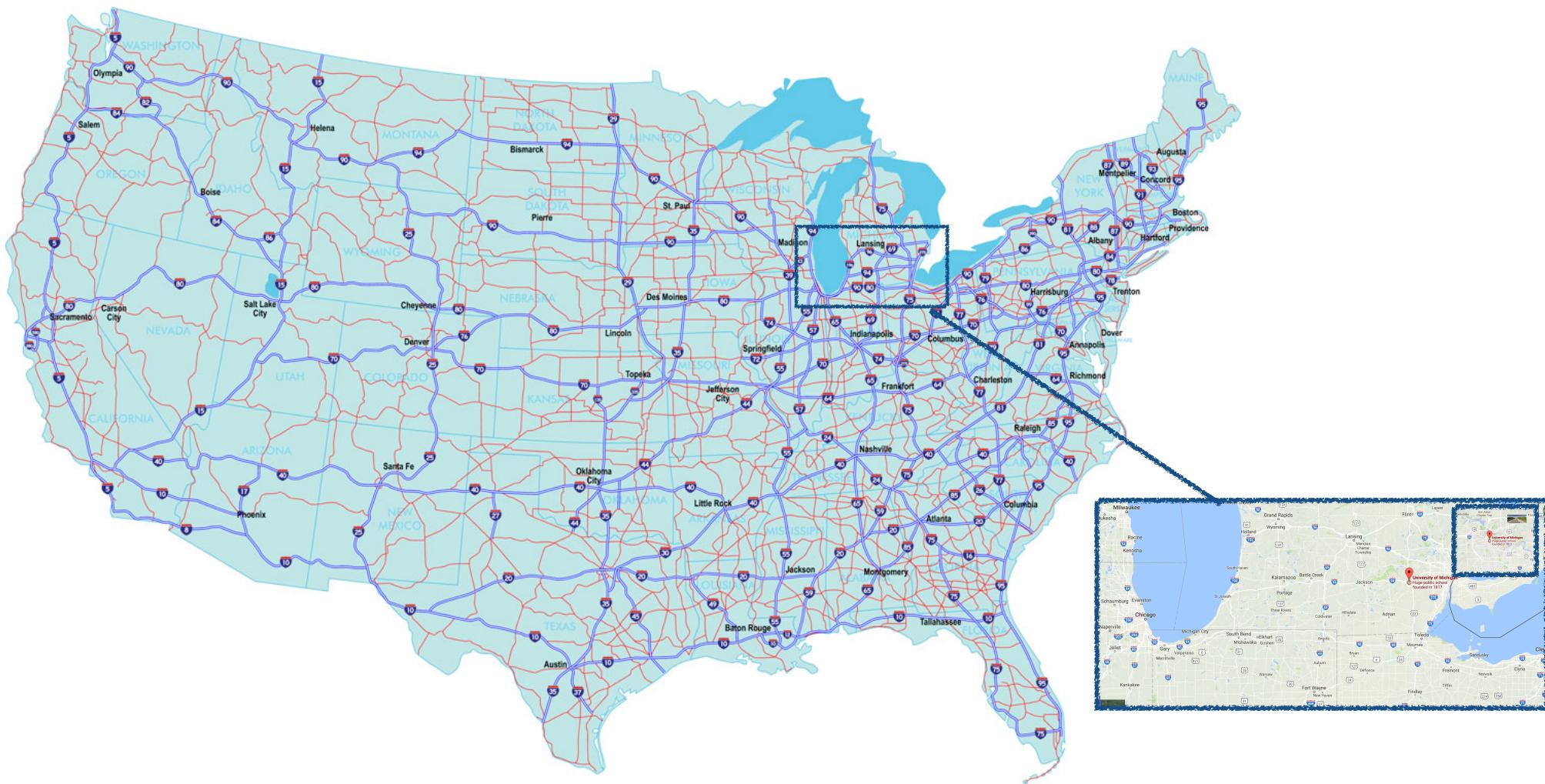
Configuration (lat,lon): [racing.vanish.boil]

Is this a usable
representation for
configuration?

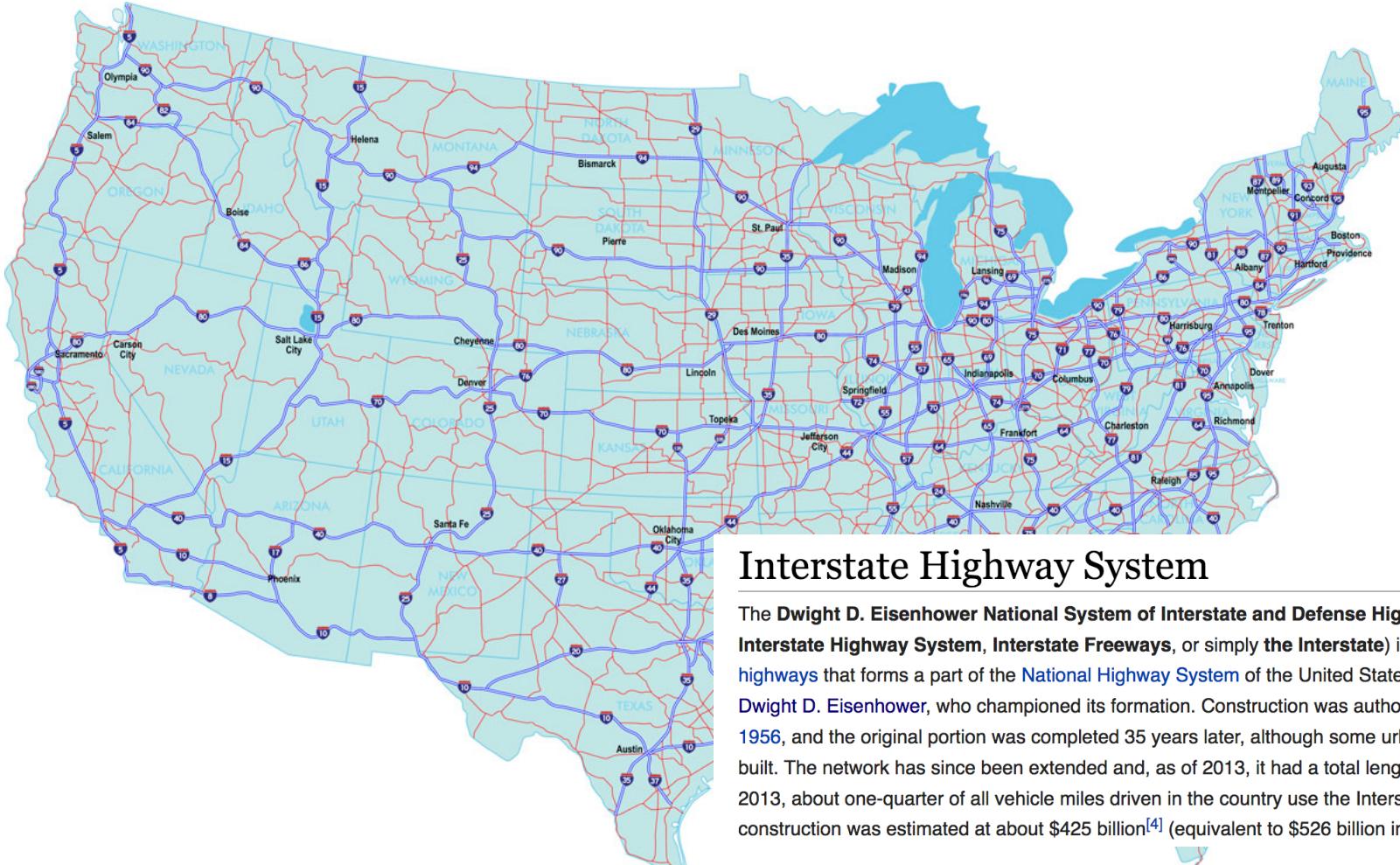
Configuration: [42.2780436, -83.7404128]







Michigan EECS 398/567 ROB 510 - autorob.org

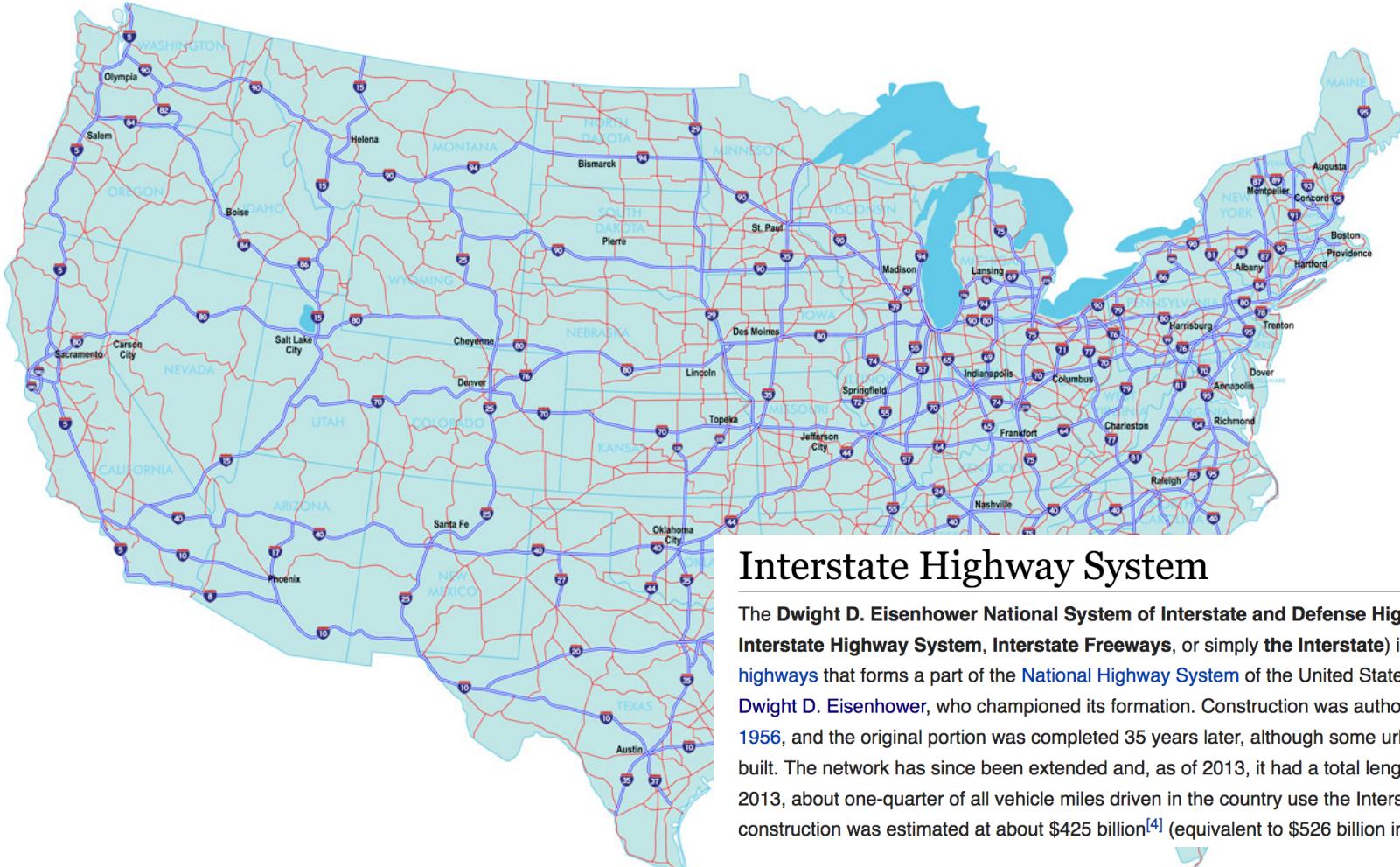


Interstate Highway System

The **Dwight D. Eisenhower National System of Interstate and Defense Highways** (commonly known as the **Interstate Highway System**, **Interstate Freeways**, or simply **the Interstate**) is a network of **controlled-access** highways that forms a part of the **National Highway System** of the United States. The system is named for President **Dwight D. Eisenhower**, who championed its formation. Construction was authorized by the **Federal Aid Highway Act of 1956**, and the original portion was completed 35 years later, although some urban routes were cancelled and never built. The network has since been extended and, as of 2013, it had a total length of 47,856 miles (77,017 km).^[2] As of 2013, about one-quarter of all vehicle miles driven in the country use the Interstate system.^[3] In 2006, the cost of construction was estimated at about \$425 billion^[4] (equivalent to \$526 billion in 2016^[5]).

CHINA'S PROPOSED BELT AND ROAD INITIATIVE

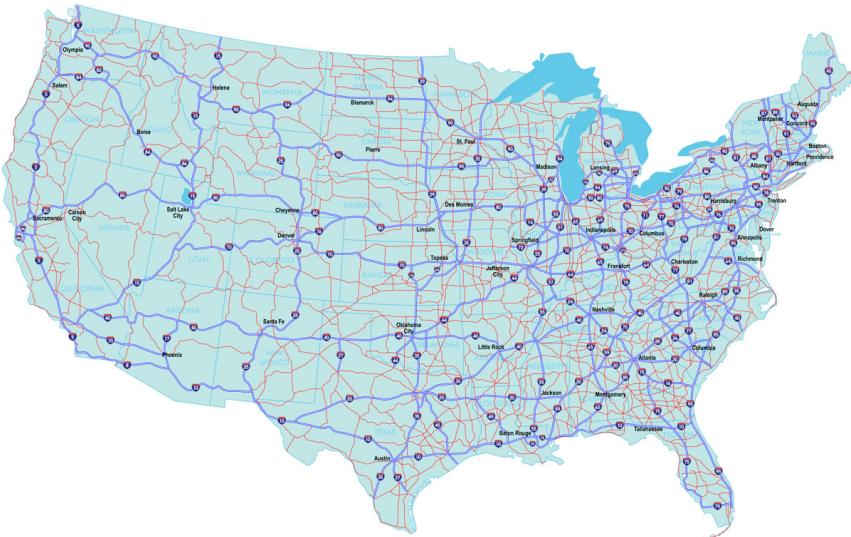




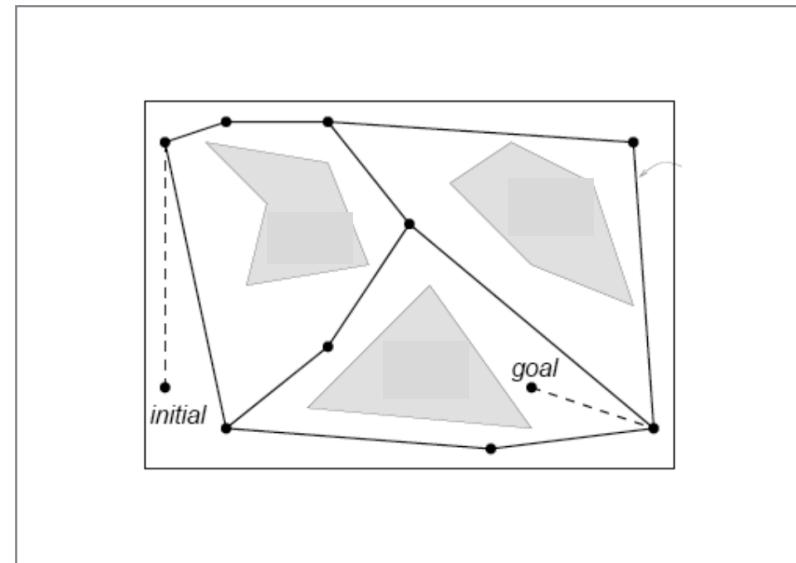
Interstate Highway System

The **Dwight D. Eisenhower National System of Interstate and Defense Highways** (commonly known as the **Interstate Highway System**, **Interstate Freeways**, or simply **the Interstate**) is a network of **controlled-access** highways that forms a part of the **National Highway System** of the United States. The system is named for President **Dwight D. Eisenhower**, who championed its formation. Construction was authorized by the **Federal Aid Highway Act of 1956**, and the original portion was completed 35 years later, although some urban routes were cancelled and never built. The network has since been extended and, as of 2013, it had a total length of 47,856 miles (77,017 km).^[2] As of 2013, about one-quarter of all vehicle miles driven in the country use the Interstate system.^[3] In 2006, the cost of construction was estimated at about \$425 billion^[4] (equivalent to \$526 billion in 2016^[5]).

Roadmaps



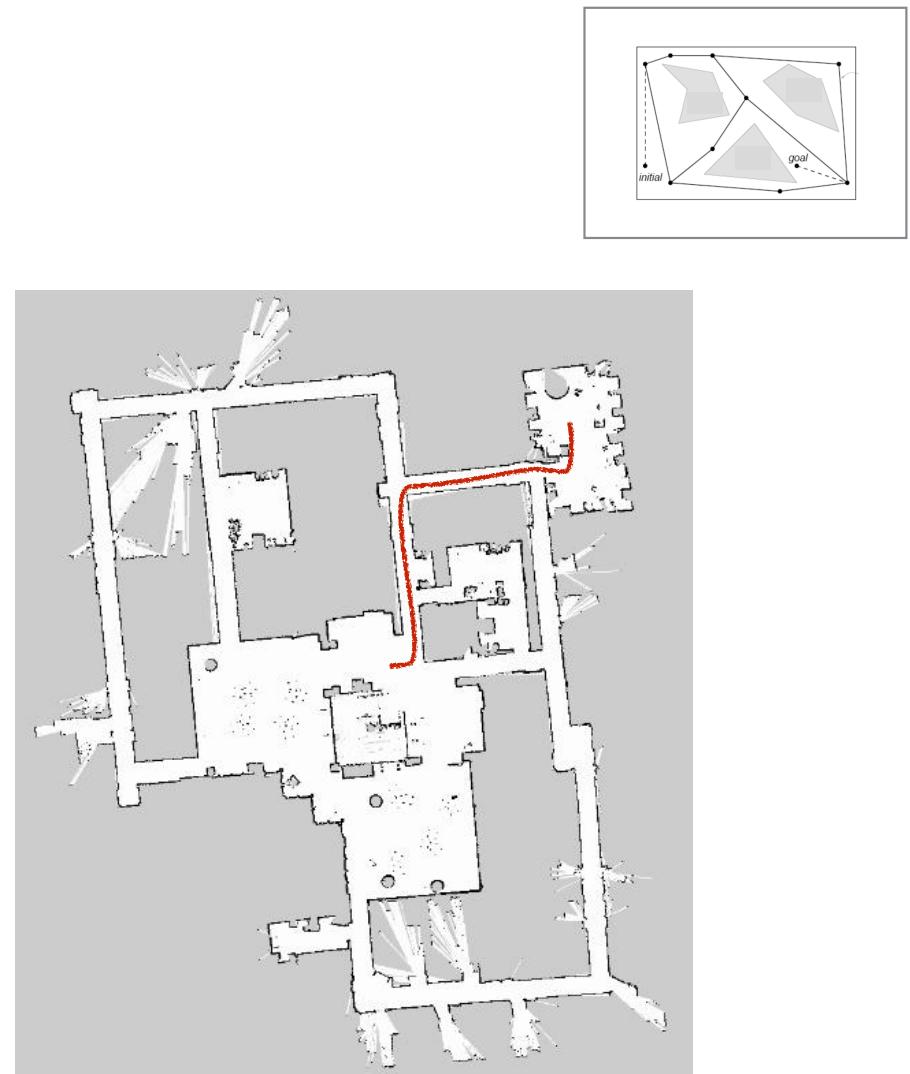
Roadmap over geolocations



Roadmap over robot configurations

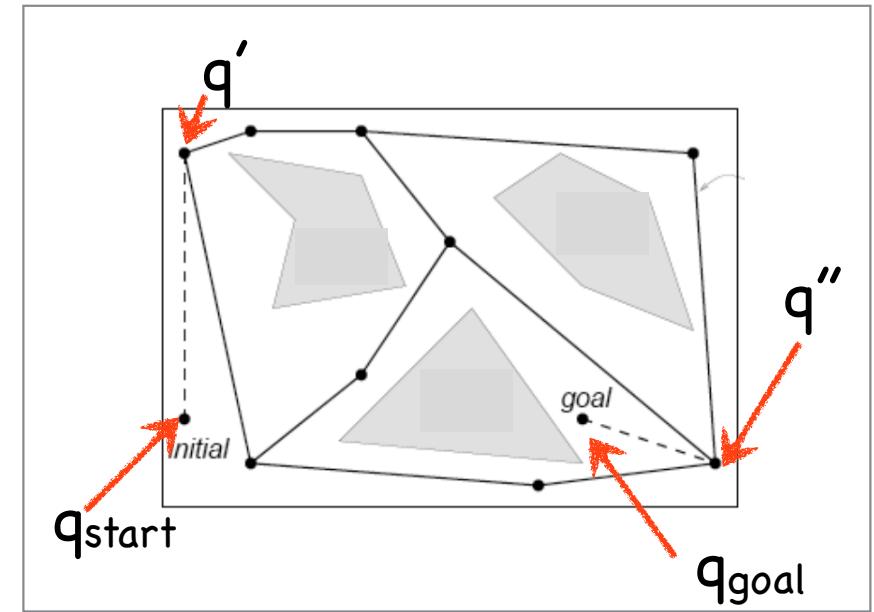
Roadmaps

- Graph search assumed C-space as a fixed uniform grid
 - finite set of discretized cells
- How does this scale beyond planar navigation?
 - curse of dimensionality
- Roadmaps are a more general notion of graphs in C-space



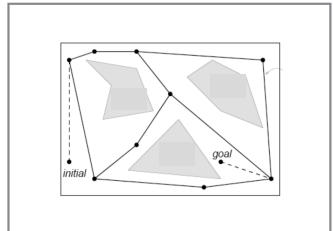
Roadmap Definition

- A roadmap RM is a union of curves s.t. all start and goal points in C-space (Q_{free}) can be connected by a path
- Roadmap properties:



- **Accessibility:** There is a path from $q_{start} \in Q_{free}$ to some $q' \in RM$
- **Departability:** There is a path from $q'' \in RM$ to $q_{goal} \in Q_{free}$
- **Connectivity:** there exists a path in RM between q' and q''

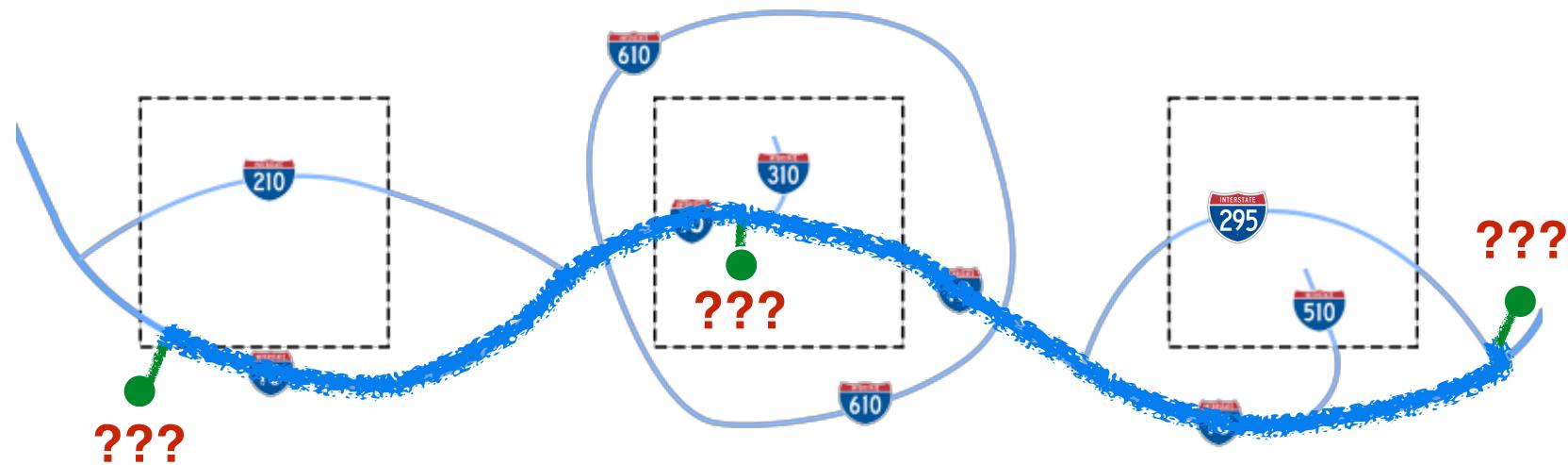
What cities? What universities?



CITY A ???

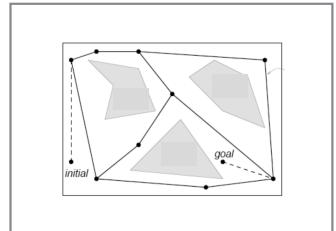
CITY B ???

CITY C ???



(to your best approximation)

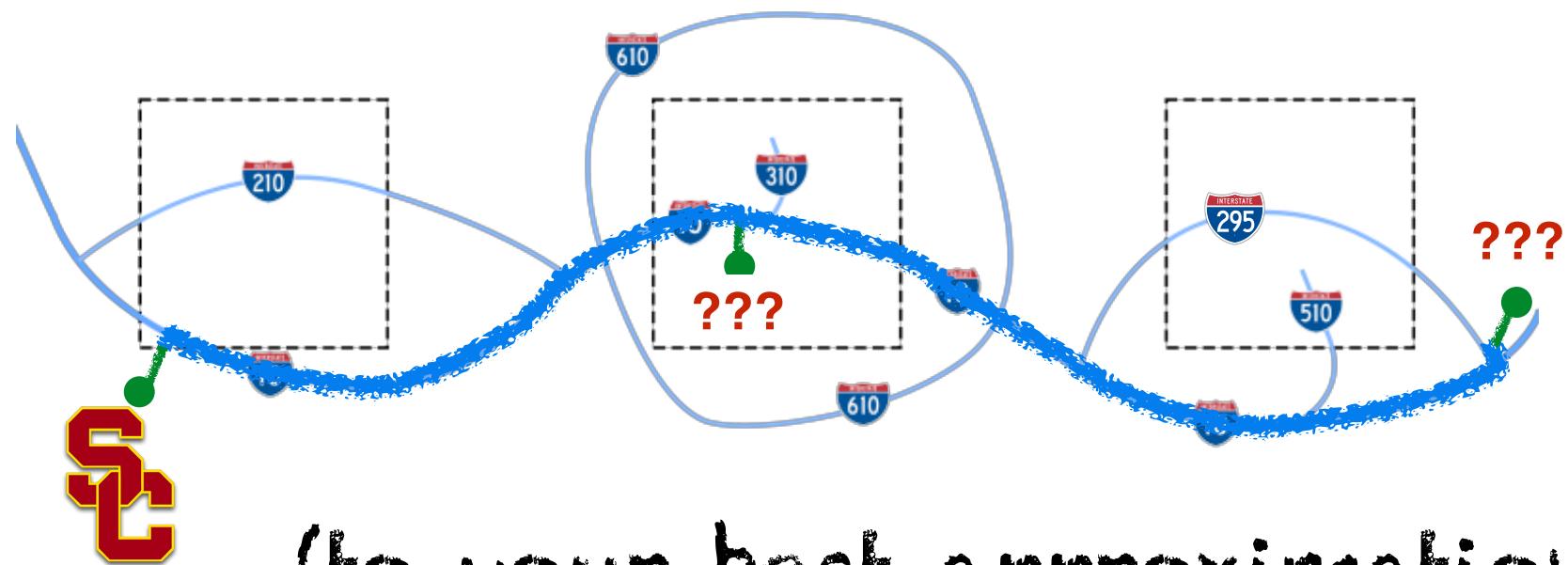
What cities? What universities?



“Los Angeles”

CITY B ???

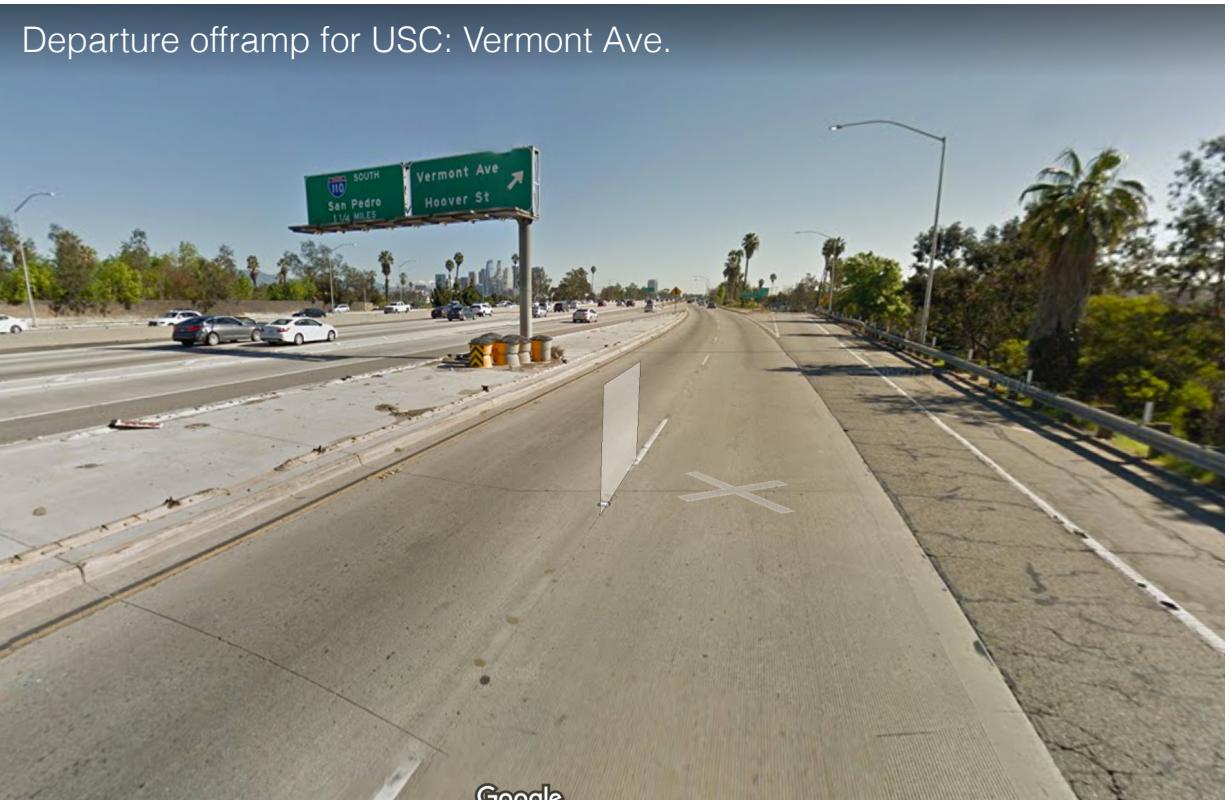
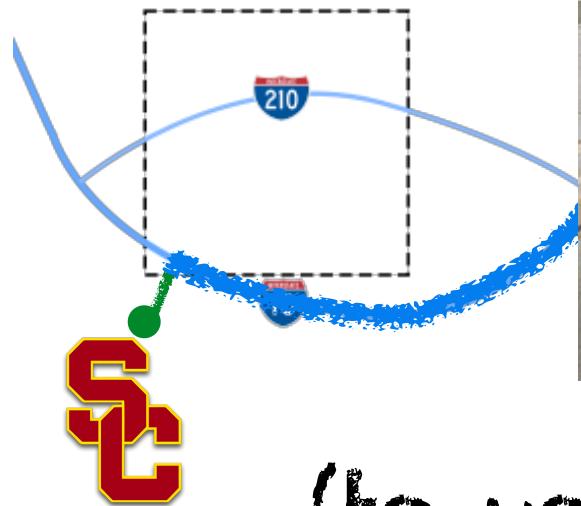
CITY C ???



(to your best approximation)

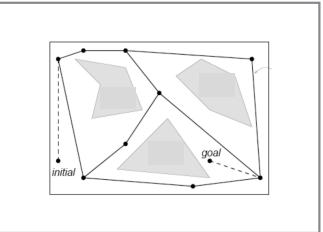
What cities? □

“Los Angeles”



(to your best approximation)

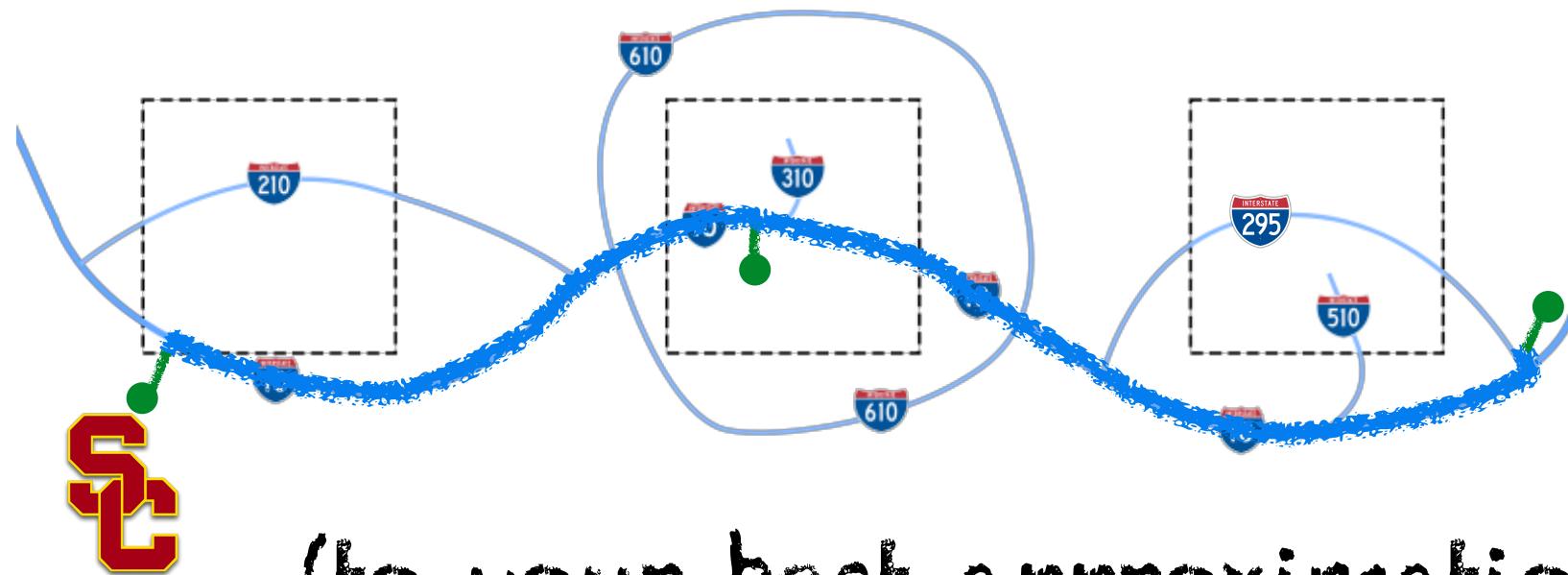
What cities? What universities?



“Los Angeles”

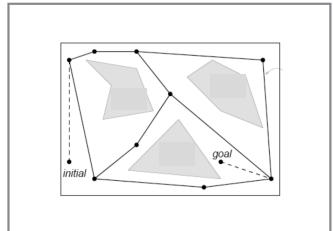
???????

?????????



(to your best approximation)

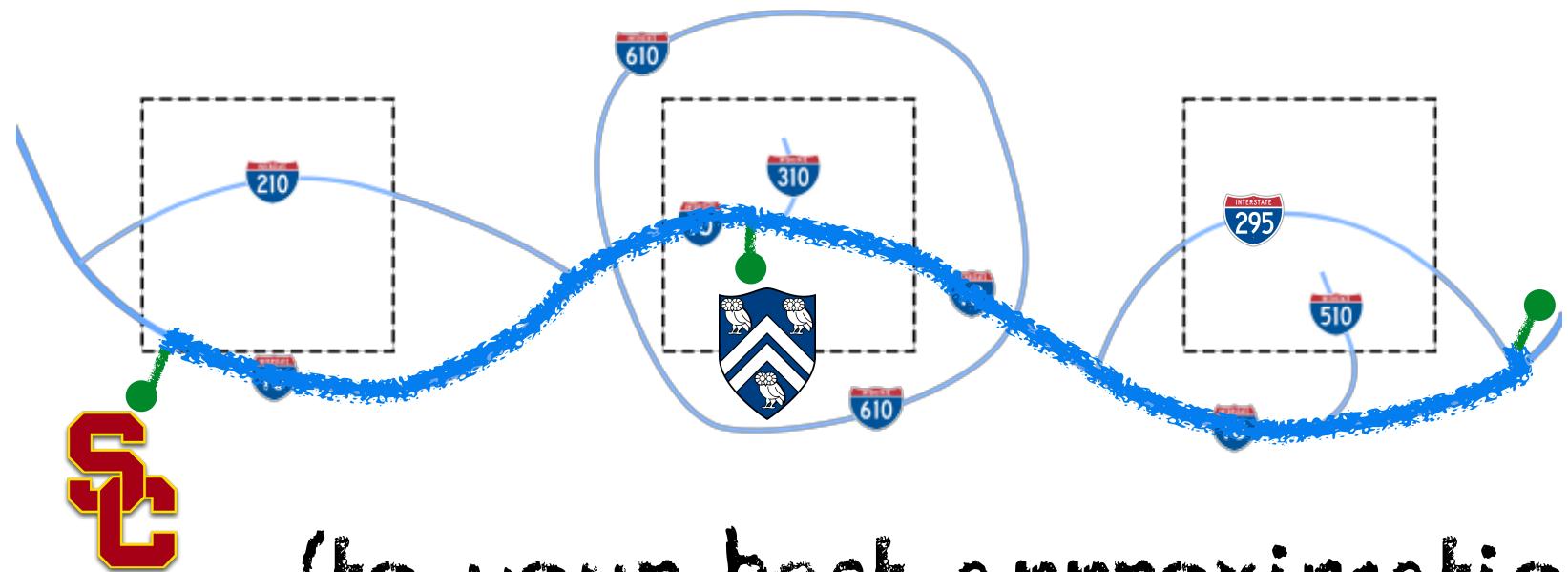
What cities? What universities?



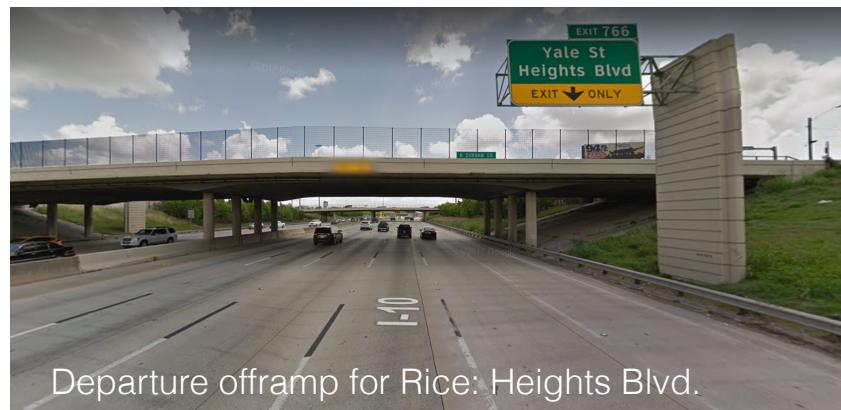
“Los Angeles”

“Houston”

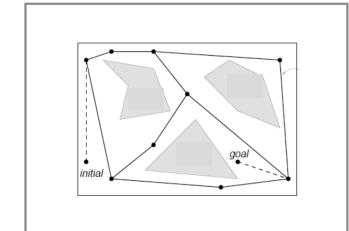
?????????



(to your best approximation)



that universities?



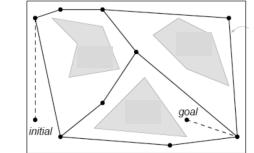
"Houston"

????????

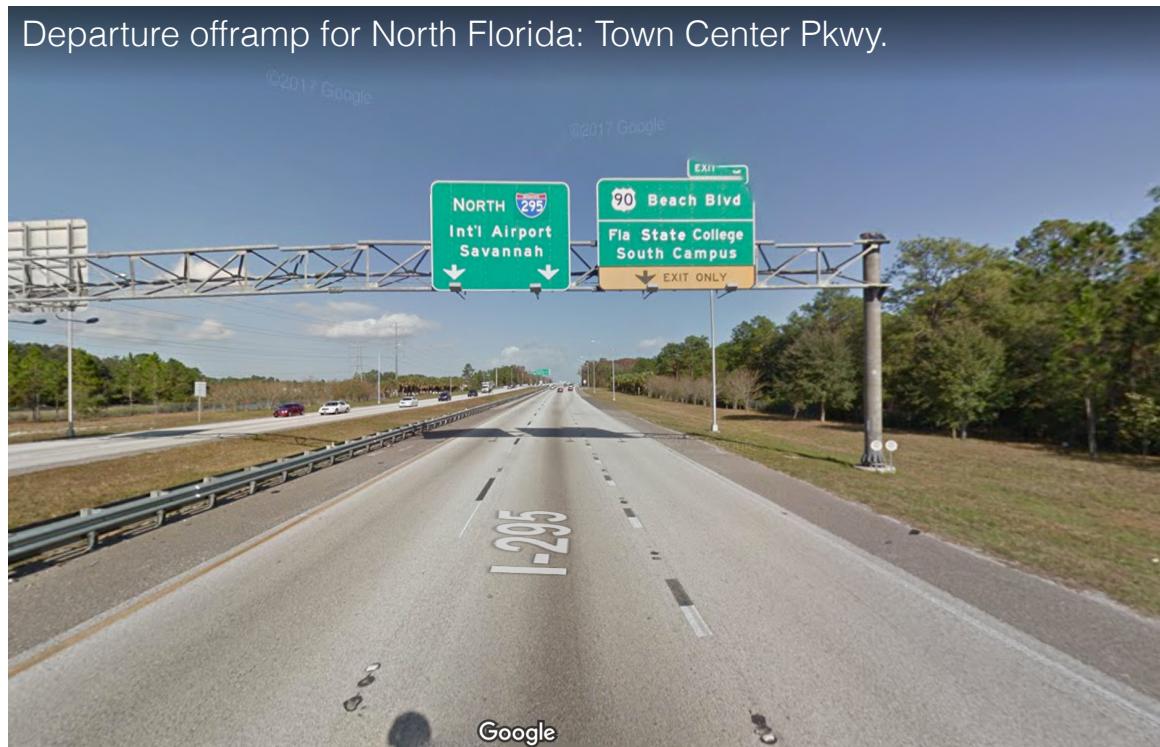


(to your best approximation)

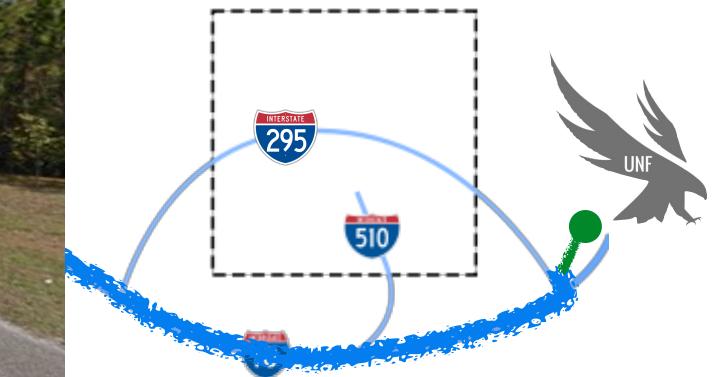
What cities? What universities?



Departure offramp for North Florida: Town Center Pkwy.



“Jacksonville”



(to your best approximation)

Departure offramp for Michigan: State St.

@2015 Google



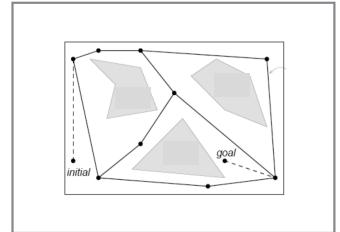
EXIT 177
State St
1 MILE

Better departure offramp for Michigan: Plymouth Rd.

@2016 Google



Basic Roadmap Planner



1) Build the roadmap RM as graph $G(V,E)$

- ④ V : nodes are “valid” in C-space in Q_{free}
 - a configuration q is valid if it is not in collision and within joint limits
- ④ E : an edge $e(q_1, q_2)$ connects two nodes if a free path connects q_1 and q_2
 - all configurations along edge assumed to be valid

2) Connect start and goal configurations to RM at q' and q'' , respectively

3) Find path in RM between q' and q''

2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
 - trace lines connecting obstacle polygon vertices
- Voronoi Planning
 - trace edges equidistant from obstacles

Probabilistic:

C-space sampling

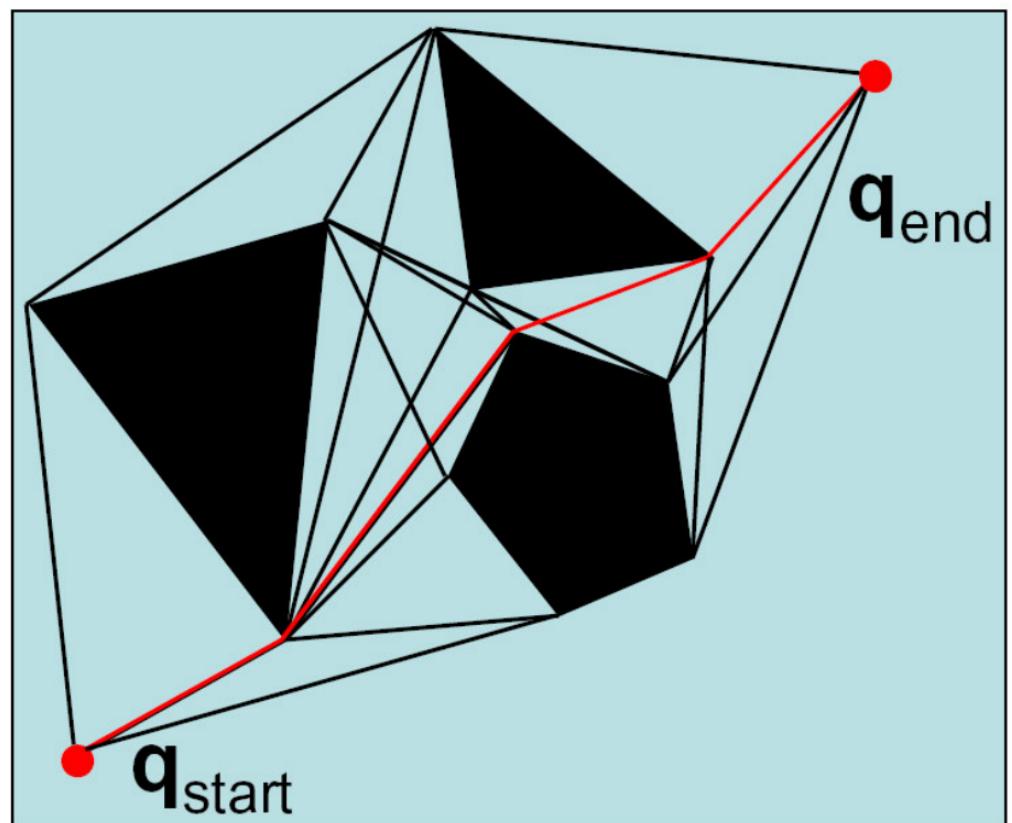
- Probabilistic Roadmap (PRM)
 - sample and connect vertices in graph for multiple planning queries
- Rapidly-exploring Random Tree (RRT)
 - sample and connect vertices in trees rooted at start and goal configuration

2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
 - trace lines connecting obstacle polygon vertices
- Voronoi Planning
 - trace edges equidistant from obstacles

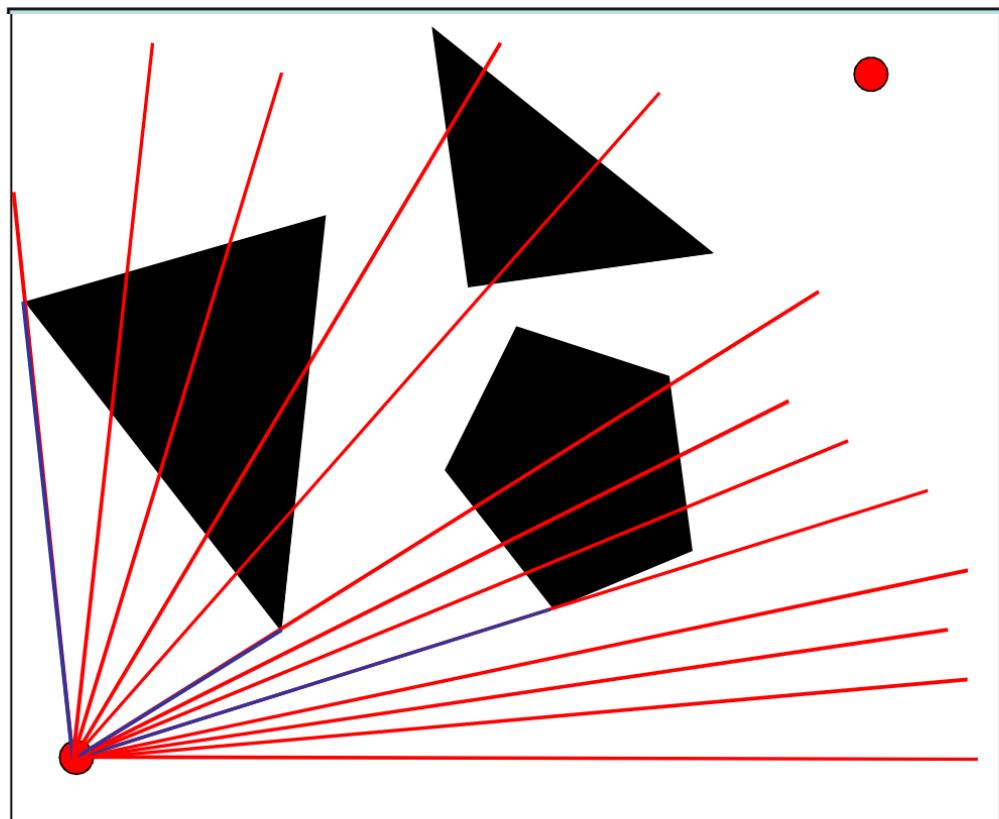


2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
 - trace lines connecting obstacle polygon vertices
- Voronoi Planning
 - trace edges equidistant from obstacles

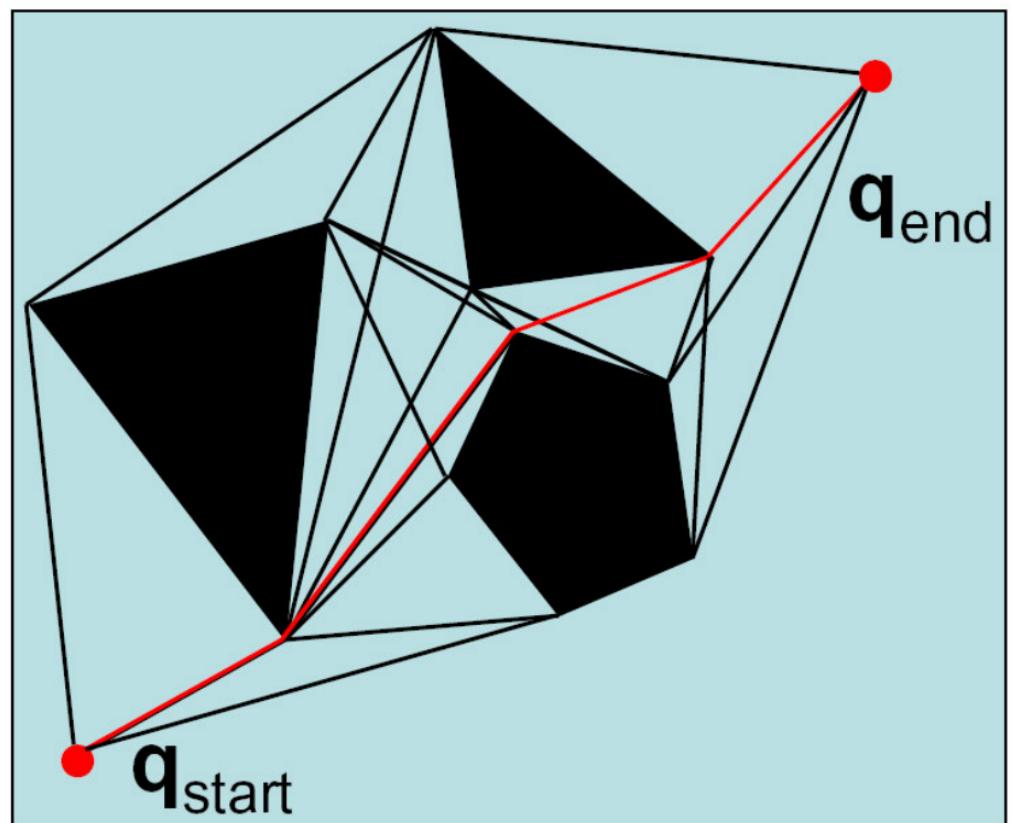


2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
 - trace lines connecting obstacle polygon vertices
- Voronoi Planning
 - trace edges equidistant from obstacles

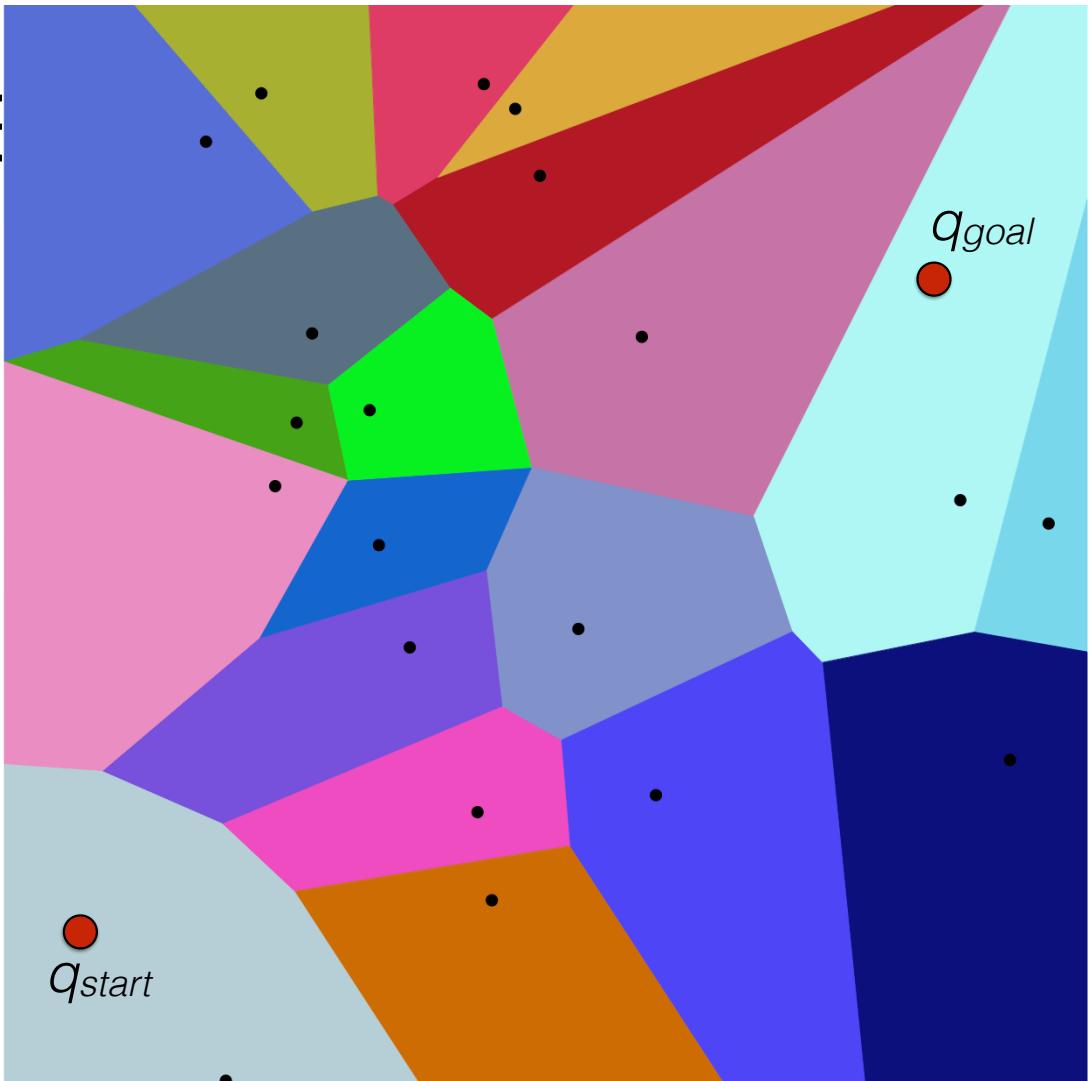


2 Approaches

Deterministic:

complete algorithms

- Visibility Graph
 - trace lines connecting obstacle polygon vertices
- Voronoi Planning
 - trace edges equidistant from obstacles

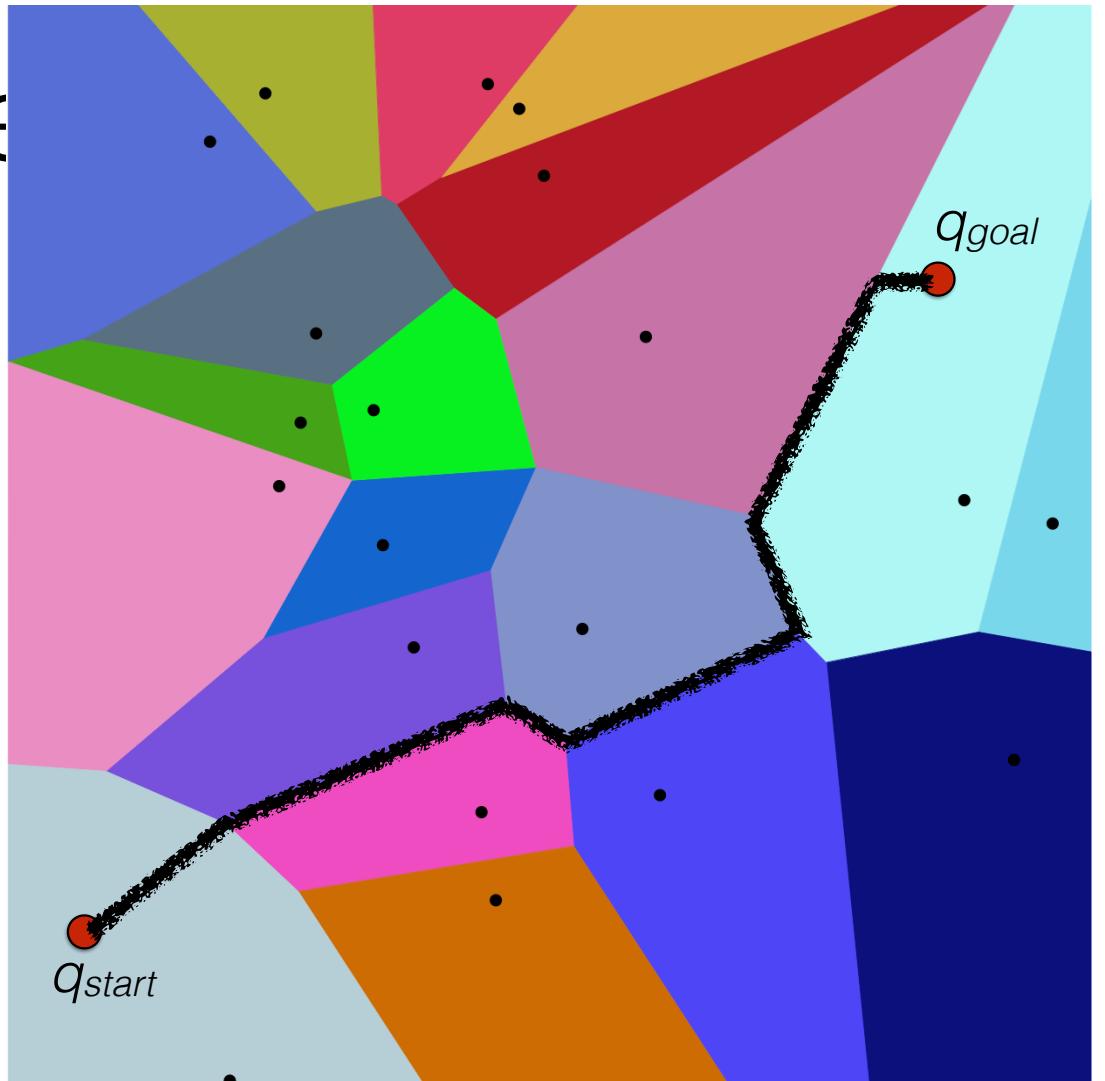


2 Approaches

Deterministic:

complete algorithms

- Visibility Graph
 - trace lines connecting obstacle polygon vertices
- Voronoi Planning
 - trace edges equidistant from obstacles



2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
 - trace lines connecting obstacle polygon vertices
- Voronoi Planning
 - trace edges equidistant from obstacles

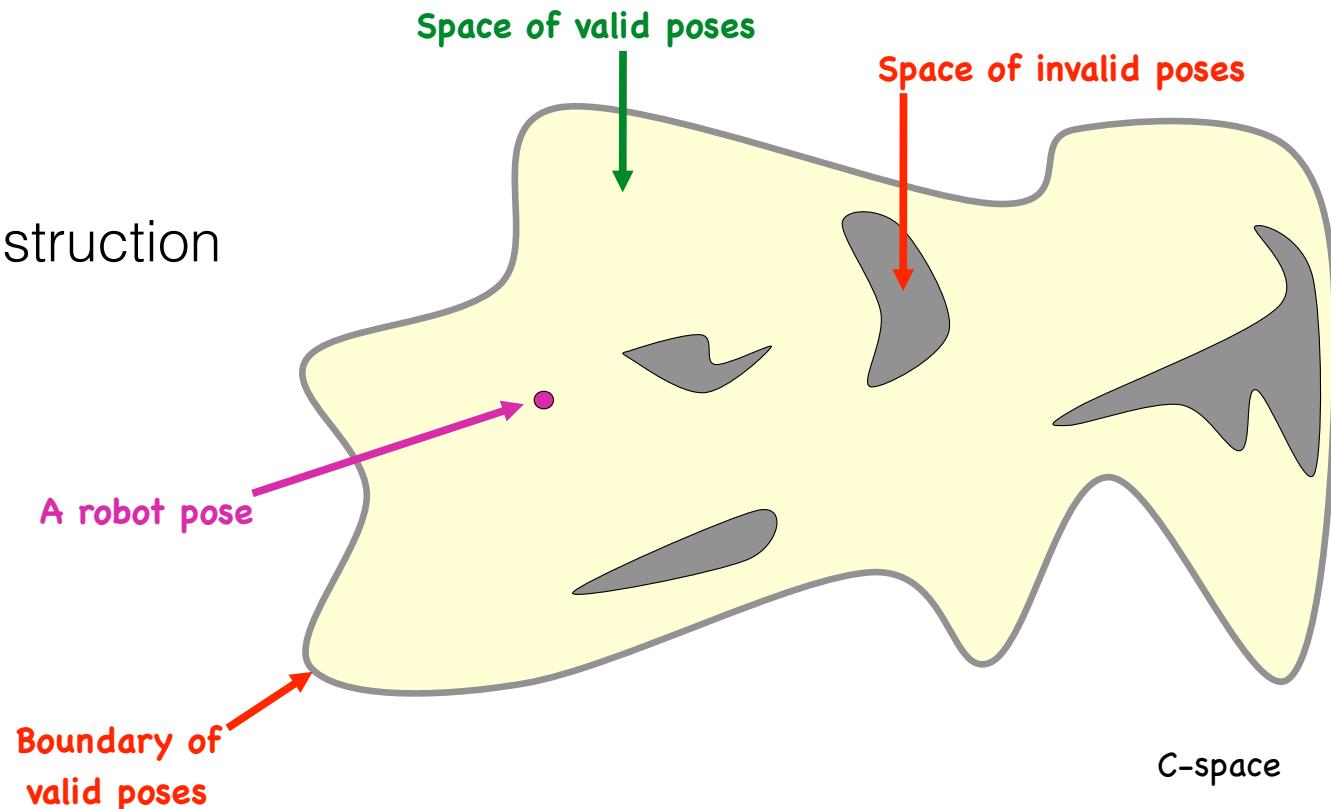
Probabilistic:

C-space sampling

- Probabilistic Roadmap (PRM)
 - sample and connect vertices in graph for multiple planning queries
- Rapidly-exploring Random Tree (RRT)
 - sample and connect vertices in trees rooted at start and goal configuration

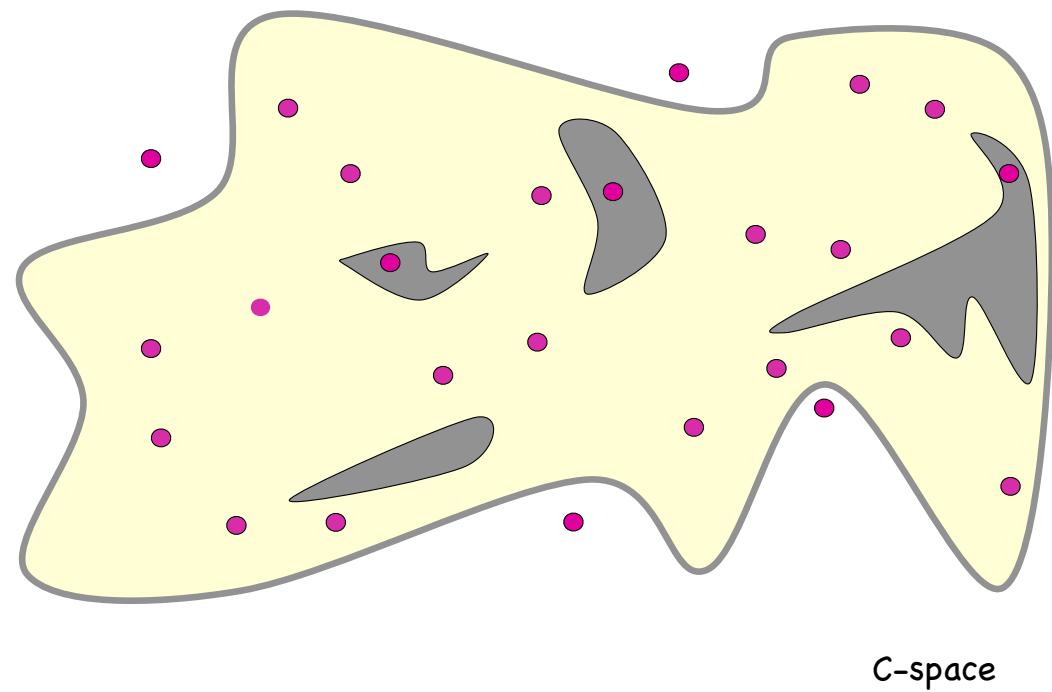
Probabilistic road maps

- Two phases
 - Roadmap construction
 - Path Query



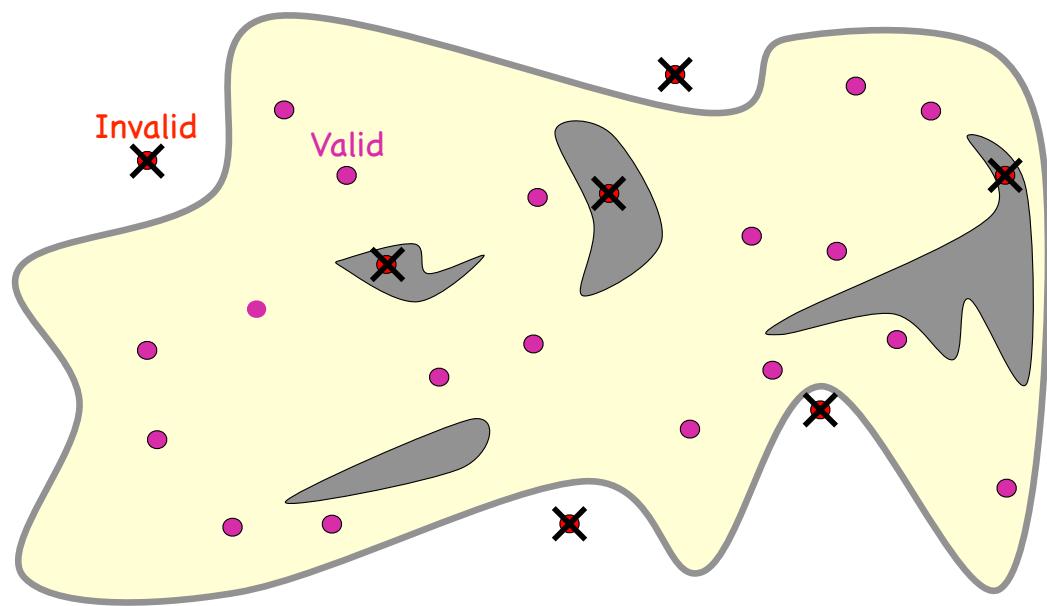
PRM: construction phase

- 1) Select N sample poses at random
- 2) Eliminate invalid poses
- 3) Connect neighboring poses



PRM: construction phase

- 1) Select N sample poses at random
- 2) **Eliminate invalid poses**
- 3) Connect neighboring poses

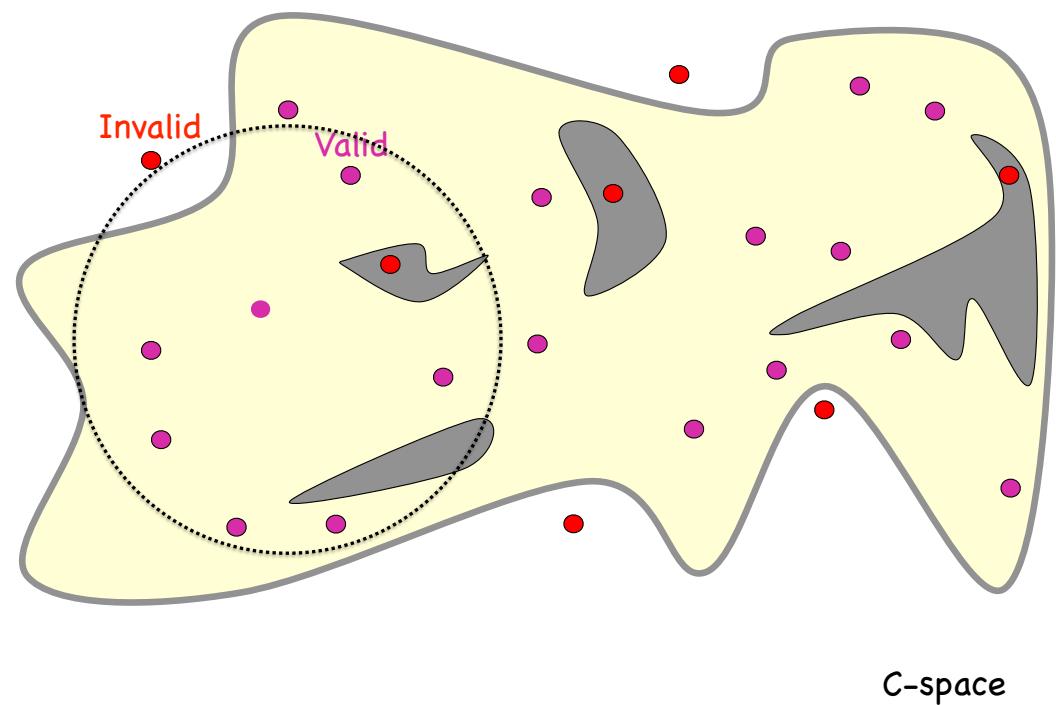


Collision detection
will be covered later

C-space

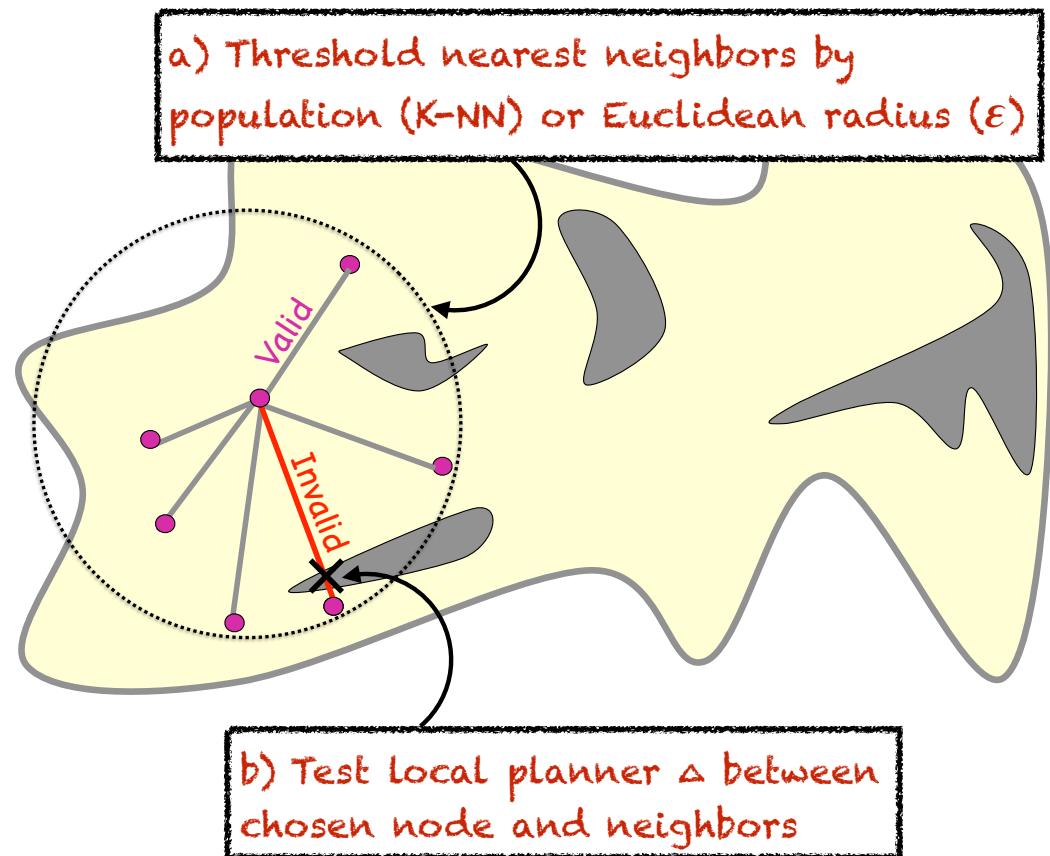
PRM: construction phase

- 1) Select N sample poses at random
- 2) Eliminate invalid poses
- 3) **Connect neighboring poses**



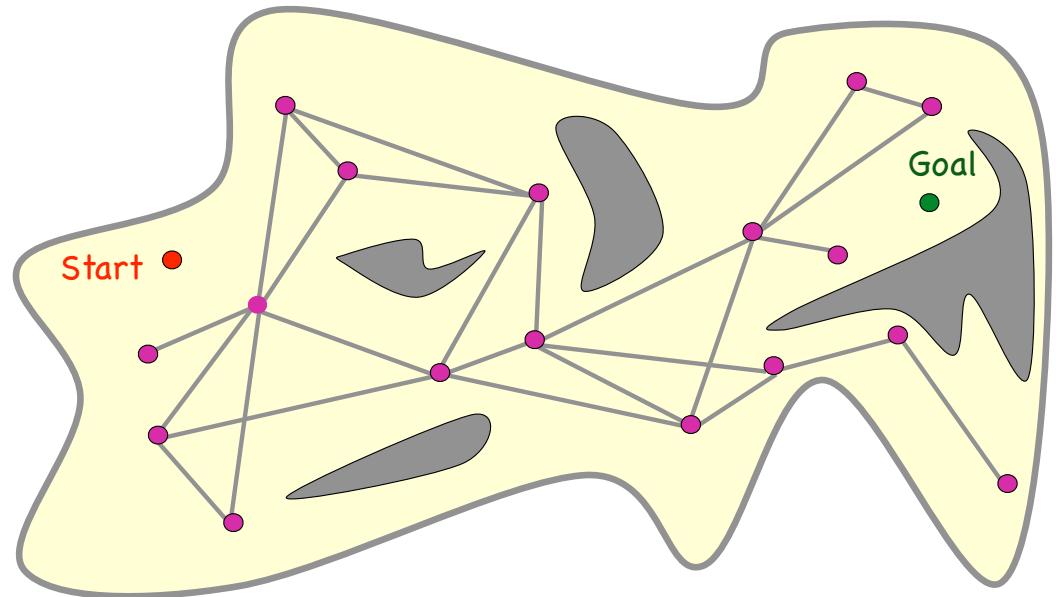
PRM: construction phase

- 1) Select N sample poses at random
- 2) Eliminate invalid poses
- 3) Connect neighboring poses



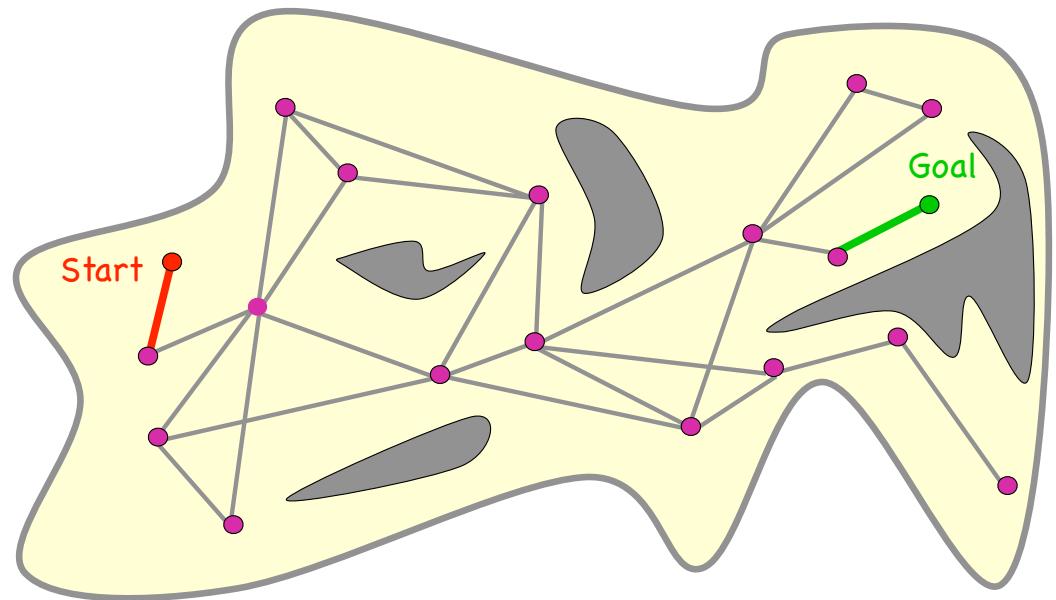
PRM: query phase

- 1) Given constructed roadmap, start pose, and goal pose
- 2) Attach goal and start to nearest roadmap entry nodes
- 3) Search for path between roadmap entry nodes
- 4) Return path with entry and departure edges



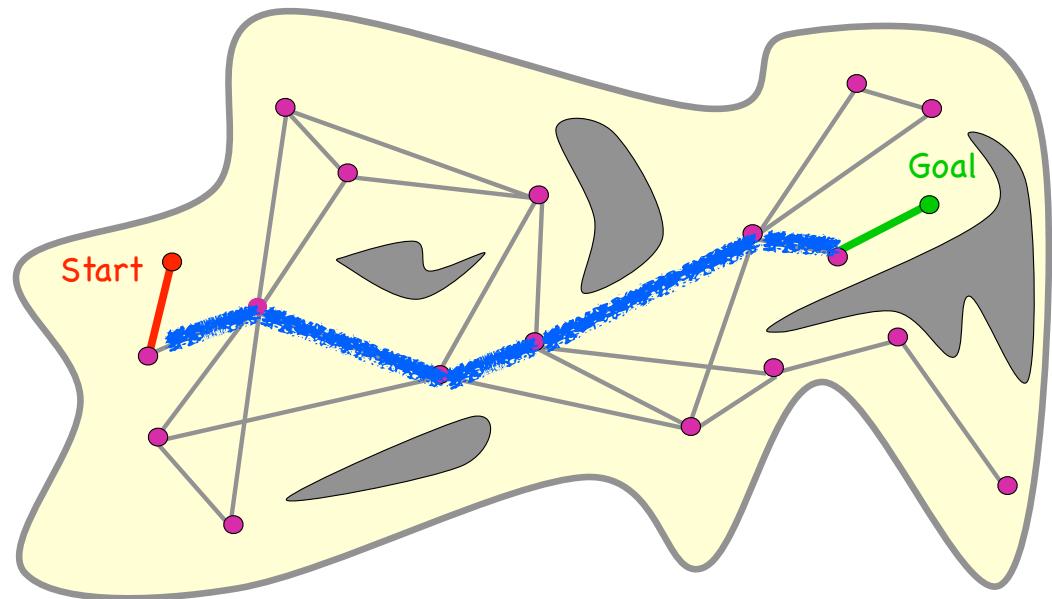
PRM: query phase

- 1) Given constructed roadmap, start pose, and goal pose
- 2) **Attach goal and start to nearest roadmap entry nodes**
- 3) Search for path between roadmap entry nodes
- 4) Return path with entry and departure edges



PRM: query phase

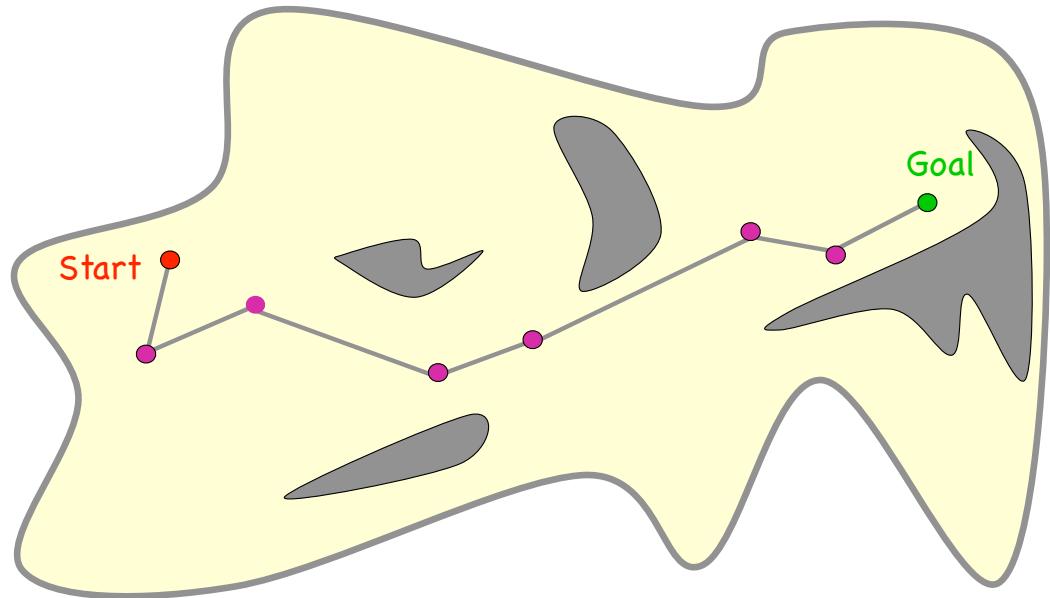
- 1) Given constructed roadmap, start pose, and goal pose
- 2) Attach goal and start to nearest roadmap entry nodes
- 3) **Search for path between roadmap entry nodes**
- 4) Return path with entry and departure edges



Remember: graph search algorithms
A*, Dijkstra, BFS, DFS

PRM: query phase

- 1) Given constructed roadmap,
start pose, and goal pose
- 2) Attach goal and start to nearest
roadmap entry nodes
- 3) Search for path between
roadmap entry nodes
- 4) **Return path with entry and
departure edges**



Multi-query planning: Considerations

- Number of samples wrt. C-space dimensionality
- Balanced sampling over C-space
- Choice of distance (e.g., Euclidean)
- Choice of local planner (e.g., line subdivision)
- Selecting neighbors: (e.g., K-NN, kd-tree, cell hashing)

2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
 - trace lines connecting obstacle polygon vertices
- Voronoi Planning
 - trace edges equidistant from obstacles

Probabilistic:

C-space sampling

- Probabilistic Roadmap (PRM)
 - sample and connect vertices in graph for multiple planning queries
- Rapidly-exploring Random Tree (RRT)
 - sample and connect vertices in trees rooted at start and goal configuration

Single Query Planning

- Given specific start and goal configurations
- Grow trees from start and goal towards each other
- Path is found once trees connect
- Focus sampling in unexplored areas of C-space and moving towards start/goal
- Common algorithms:
 - ESTs (expansive space trees)
 - **RRTs (rapidly exploring random trees)**

RRT Algorithm

Extend graph towards a random configuration and repeat

```
BUILD_RRT( $q_{init}$ )
1    $T.init(q_{init})$ ;
2   for  $k = 1$  to  $K$  do
3        $q_{rand} \leftarrow \text{RANDOM.CONFIG}()$ ;
4       EXTEND( $T, q_{rand}$ );
5   Return  $T$ 
```

RRT Algorithm

Extend graph towards a random configuration and repeat

```
BUILD_RRT( $q_{init}$ )
1    $T.init(q_{init})$ ;
2   for  $k = 1$  to  $K$  do
3        $q_{rand} \leftarrow \text{RANDOM.CONFIG}()$ ;
4       EXTEND( $T, q_{rand}$ );
5   Return  $T$ 
```

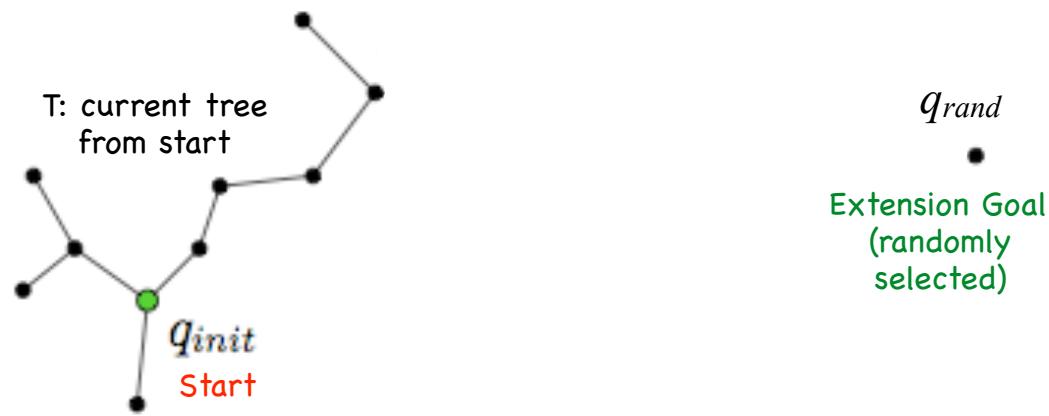


Figure 3: The EXTEND operation.

RRT Algorithm

Extend graph towards a random configuration and repeat

```
BUILD_RRT( $q_{init}$ )
1    $T.init(q_{init})$ ;
2   for  $k = 1$  to  $K$  do
3        $q_{rand} \leftarrow RANDOM.CONFIG()$ ;
4       EXTEND( $T, q_{rand}$ );
5   Return  $T$ 
```

```
EXTEND( $T, q$ )
1    $q_{near} \leftarrow NEAREST.NEIGHBOR(q, T)$ ;
2   if NEW_CONFIG( $q, q_{near}, q_{new}$ ) then
3        $T.add\_vertex(q_{new})$ ;
4        $T.add\_edge(q_{near}, q_{new})$ ;
5       if  $q_{new} = q$  then
6           Return Reached;
7       else
8           Return Advanced;
9   Return Trapped;
```

Extend graph towards a random configuration

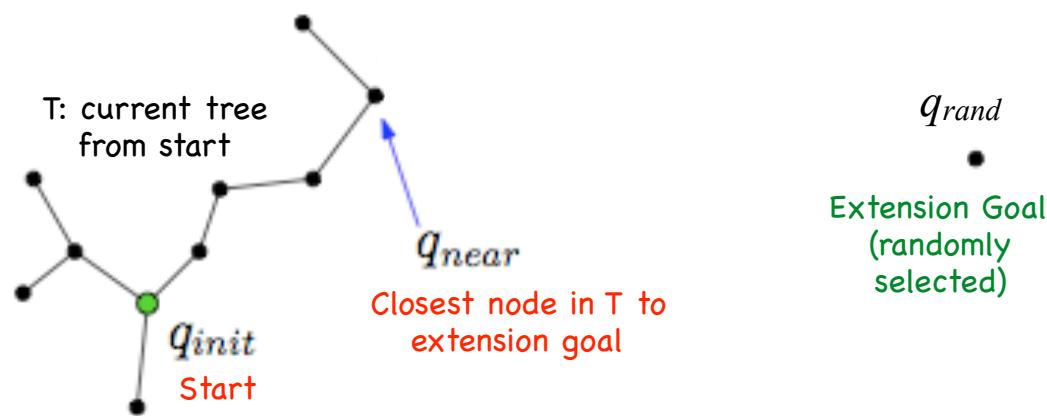


Figure 3: The EXTEND operation.

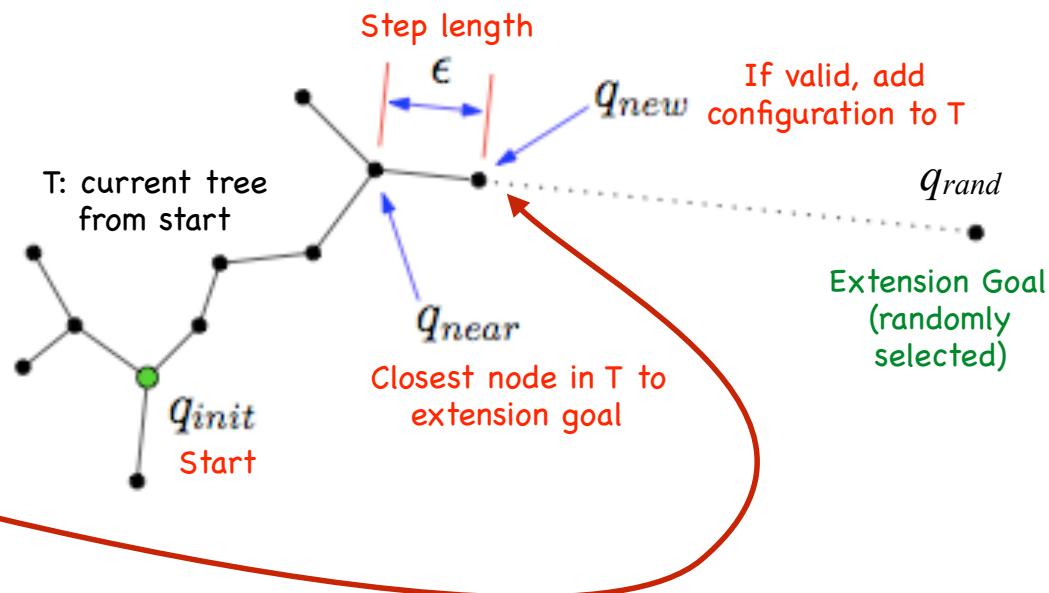
RRT Algorithm

Extend graph towards a random configuration and repeat

```
BUILD_RRT( $q_{init}$ )
1    $T.init(q_{init})$ ;
2   for  $k = 1$  to  $K$  do
3        $q_{rand} \leftarrow RANDOM.CONFIG()$ ;
4       EXTEND( $T, q_{rand}$ );
5   Return  $T$ 
```

```
EXTEND( $T, q$ )
1    $q_{near} \leftarrow NEAREST.NEIGHBOR(q, T)$ ;
2   if NEW_CONFIG( $q, q_{near}, q_{new}$ ) then
3        $T.add\_vertex(q_{new})$ ;
4        $T.add\_edge(q_{near}, q_{new})$ ;
5       if  $q_{new} = q$  then
6           Return Reached;
7       else
8           Return Advanced;
9   Return Trapped;
```

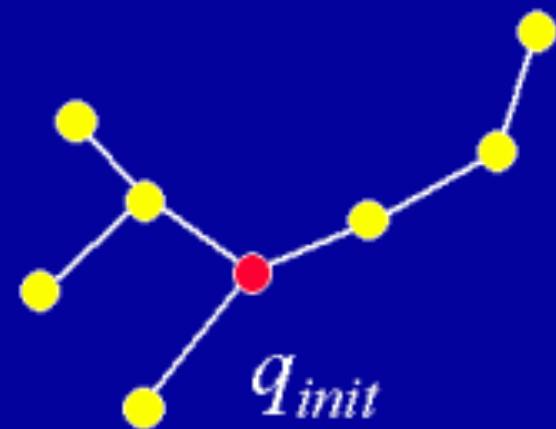
Extend graph towards a random configuration



Generate and test new configuration along vector in C-space from q_{near} to q_{rand}

RRT Extend animation

Existing RRT is “grown” as follows...



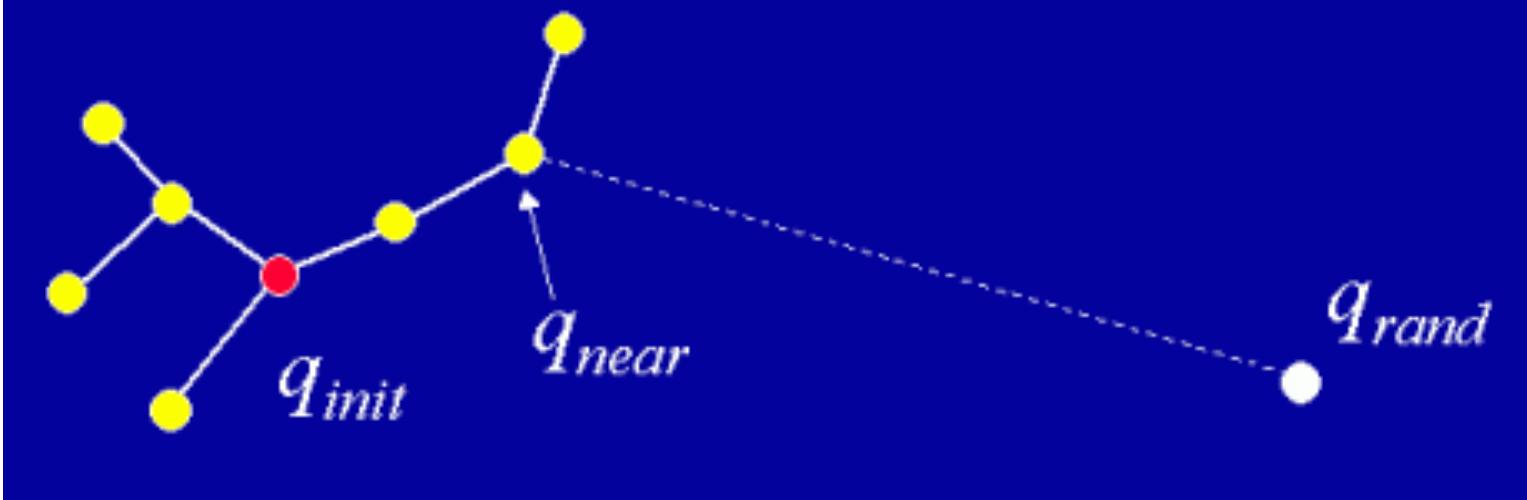
RRT Extend animation

- 1) Select a random “target” node



RRT Extend animation

2) Calculate “nearest” node in tree

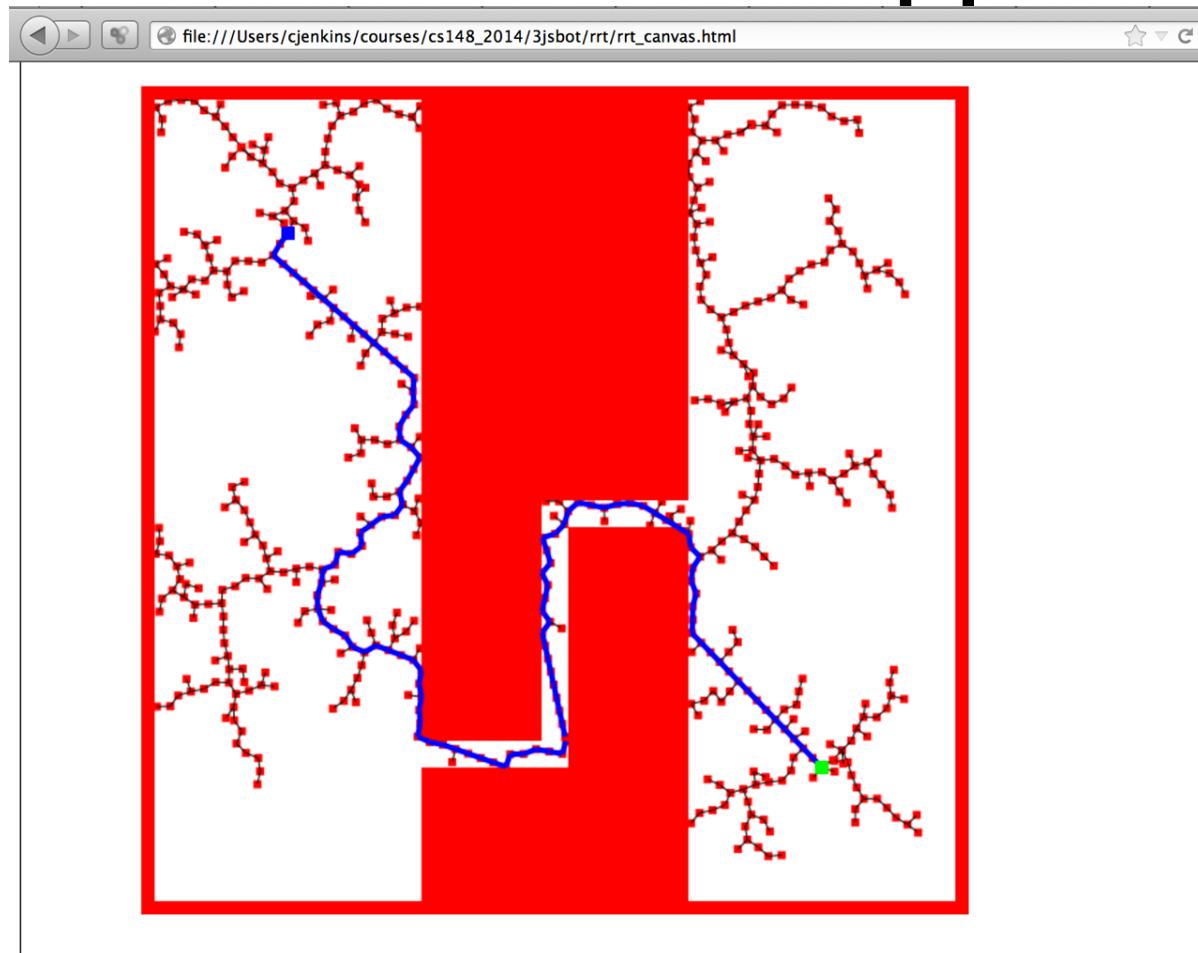


RRT Extend animation

3) Try to add new collision-free branch



Let's see what happens

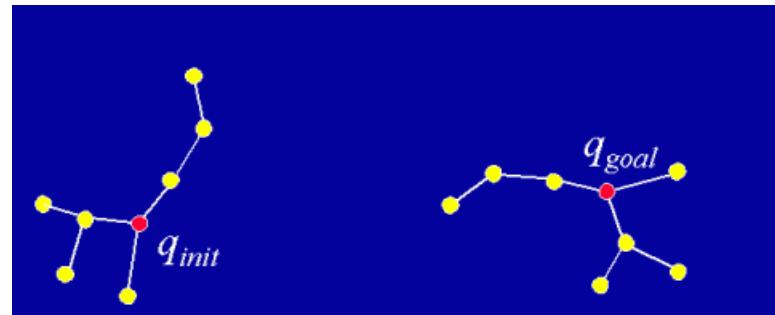


[Kuffner, LaValle 2000]

RRT Connect

- 0) Use 2 trees (A and B) rooted at start and goal configurations

```
RRT_CONNECT_PLANNER( $q_{init}, q_{goal}$ )
1    $T_a.init(q_{init}); T_b.init(q_{goal});$ 
2   for  $k = 1$  to  $K$  do
3        $q_{rand} \leftarrow \text{RANDOM\_CONFIG}();$ 
4       if not ( $\text{EXTEND}(T_a, q_{rand}) = \text{Trapped}$ ) then
5           if ( $\text{CONNECT}(T_b, q_{new}) = \text{Reached}$ ) then
6               Return PATH( $T_a, T_b$ );
7           SWAP( $T_a, T_b$ );
8   Return Failure
```



RRT Connect

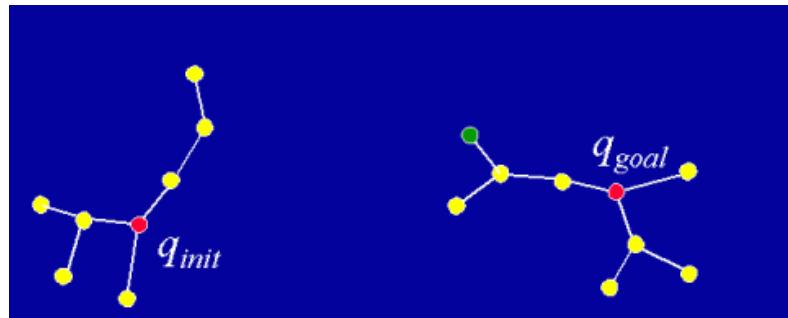
- 0) Use 2 trees (A and B) rooted at start and goal configurations

```
RRT_CONNECT_PLANNER( $q_{init}, q_{goal}$ )
1  $\mathcal{T}_a.init(q_{init}); \mathcal{T}_b.init(q_{goal});$ 
2 for  $k = 1$  to  $K$  do
3    $q_{rand} \leftarrow \text{RANDOM\_CONFIG}();$ 
4   if not (EXTEND( $\mathcal{T}_a, q_{rand}$ ) = Trapped) then
5     if (CONNECT( $\mathcal{T}_b, q_{new}$ ) = Reached) then
6       Return PATH( $\mathcal{T}_a, \mathcal{T}_b$ );
7     SWAP( $\mathcal{T}_a, \mathcal{T}_b$ );
8   Return Failure
```

EXTEND(\mathcal{T}, q)

```
1  $q_{near} \leftarrow \text{NEAREST\_NEIGHBOR}(q, \mathcal{T});$ 
2 if NEW_CONFIG( $q, q_{near}, q_{new}$ ) then
3    $\mathcal{T}.add\_vertex(q_{new});$ 
4    $\mathcal{T}.add\_edge(q_{near}, q_{new});$ 
5   if  $q_{new} = q$  then
6     Return Reached;
7   else
8     Return Advanced;
9 Return Trapped;
```

- 1) Extend tree A towards a random configuration



RRT Connect

0) Use 2 trees (A and B) rooted at start and goal configurations

RRT_CONNECT_PLANNER(q_{init}, q_{goal})

```

1  $T_a.init(q_{init}); T_b.init(q_{goal});$ 
2 for  $k = 1$  to  $K$  do
3    $q_{rand} \leftarrow RANDOM\_CONFIG();$ 
4   if not (EXTEND( $T_a, q_{rand}$ ) = Trapped) then
5     if (CONNECT( $T_b, q_{near}$ ) = Reached) then
6       Return PATH( $T_a, T_b$ );
7     SWAP( $T_a, T_b$ );
8   Return Failure

```

EXTEND(T, q)

```

1  $q_{near} \leftarrow NEAREST\_NEIGHBOR(q, T);$ 
2 if NEW_CONFIG( $q, q_{near}, q_{new}$ ) then
3    $T.add\_vertex(q_{new});$ 
4    $T.add\_edge(q_{near}, q_{new});$ 
5   if  $q_{new} = q$  then
6     Return Reached;
7   else
8     Return Advanced;
9 Return Trapped;

```

1) Extend tree A towards a random configuration

CONNECT(T, q)

```

1 repeat
2    $S \leftarrow EXTEND(T, q);$ 
3 until not ( $S = Advanced$ )
4 Return S;

```

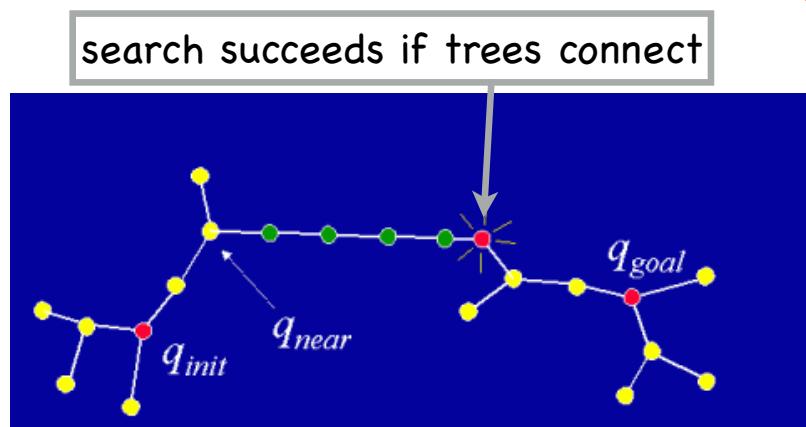
2) Try to connect tree B to tree A by extending repeatedly from its nearest neighbor



RRT Connect

0) Use 2 trees (A and B) rooted at start and goal configurations

```
RRT_CONNECT_PLANNER( $q_{init}, q_{goal}$ )
1  $T_a.init(q_{init}); T_b.init(q_{goal});$ 
2 for  $k = 1$  to  $K$  do
3    $q_{rand} \leftarrow RANDOM\_CONFIG();$ 
4   if not (EXTEND( $T_a, q_{rand}$ ) = Trapped) then
5     if (CONNECT( $T_b, q_{near}$ ) = Reached) then
6       Return PATH( $T_a, T_b$ );
7     SWAP( $T_a, T_b$ );
8   Return Failure
```



EXTEND(T, q)

```
1  $q_{near} \leftarrow NEAREST\_NEIGHBOR(q, T);$ 
2 if NEW_CONFIG( $q, q_{near}, q_{new}$ ) then
3    $T.add\_vertex(q_{new});$ 
4    $T.add\_edge(q_{near}, q_{new});$ 
5   if  $q_{new} = q$  then
6     Return Reached;
7   else
8     Return Advanced;
9 Return Trapped;
```

1) Extend tree A towards a random configuration

CONNECT(T, q)

```
1 repeat
2    $S \leftarrow EXTEND(T, q);$ 
3 until not ( $S = Advanced$ )
4 Return S;
```

2) Try to connect tree B to tree A by extending repeatedly from its nearest neighbor

RRT Connect

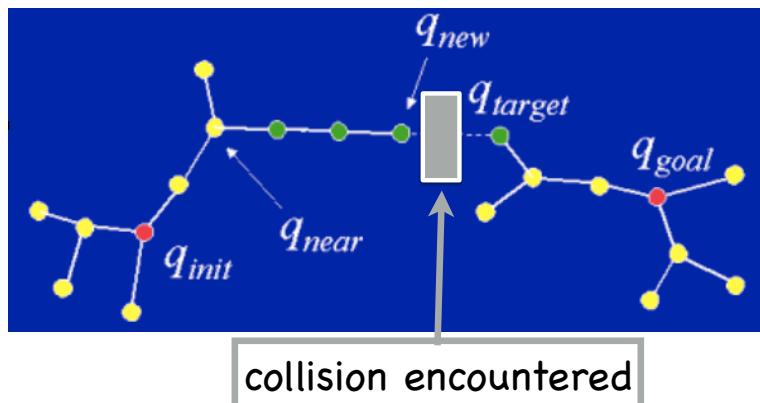
- 0) Use 2 trees (A and B) rooted at start and goal configurations

RRT_CONNECT_PLANNER(q_{init}, q_{goal})

```

1  $T_a.init(q_{init}); T_b.init(q_{goal});$ 
2 for  $k = 1$  to  $K$  do
3    $q_{rand} \leftarrow RANDOM\_CONFIG();$ 
4   if not (EXTEND( $T_a, q_{rand}$ ) = Trapped) then
5     if (CONNECT( $T_b, q_{near}$ ) = Reached) then
6       Return PATH( $T_a, T_b$ );
7     SWAP( $T_a, T_b$ );
8   Return Failure
```

- 3) reverse roles for trees A and B and repeat



EXTEND(\mathcal{T}, q)

```

1  $q_{near} \leftarrow NEAREST\_NEIGHBOR(q, \mathcal{T});$ 
2 if NEW_CONFIG( $q, q_{near}, q_{new}$ ) then
3    $\mathcal{T}.add\_vertex(q_{new});$ 
4    $\mathcal{T}.add\_edge(q_{near}, q_{new});$ 
5   if  $q_{new} = q$  then
6     Return Reached;
7   else
8     Return Advanced;
9 Return Trapped;
```

- 1) Extend tree A towards a random configuration

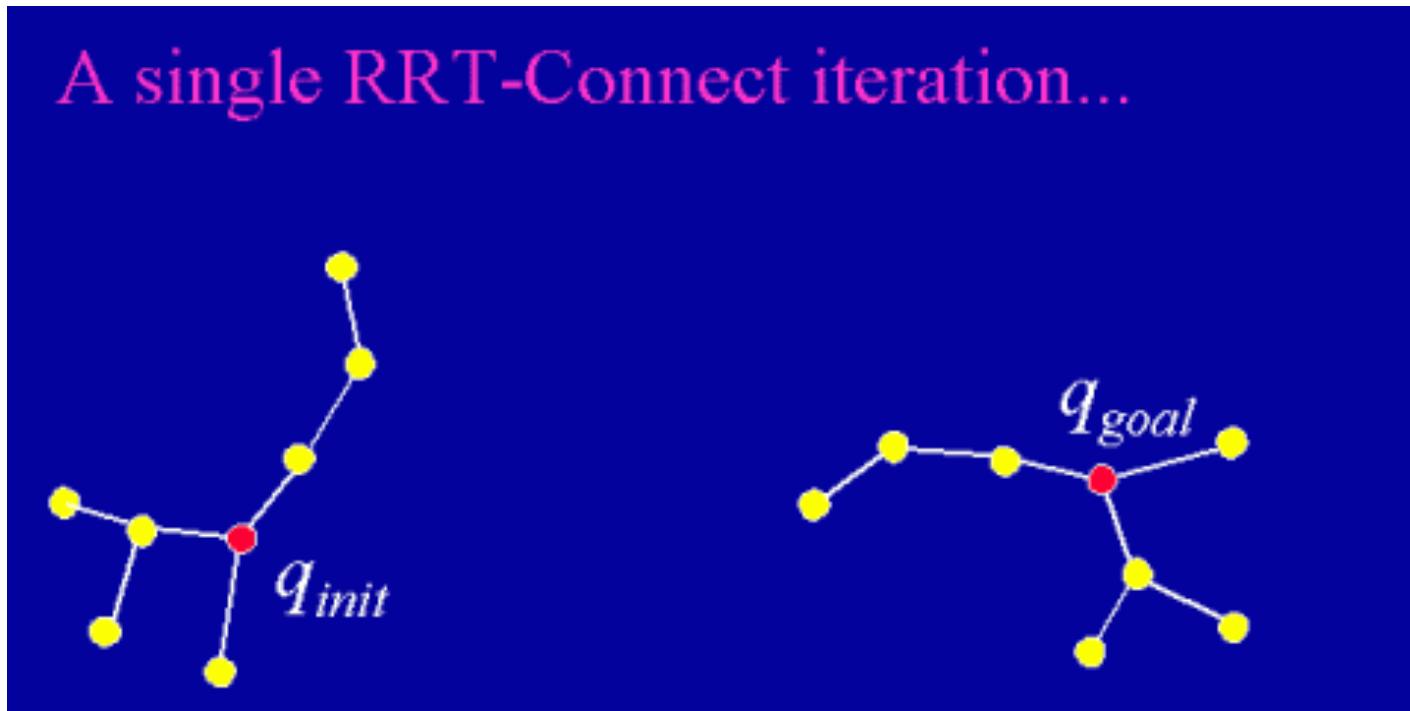
CONNECT(\mathcal{T}, q)

```

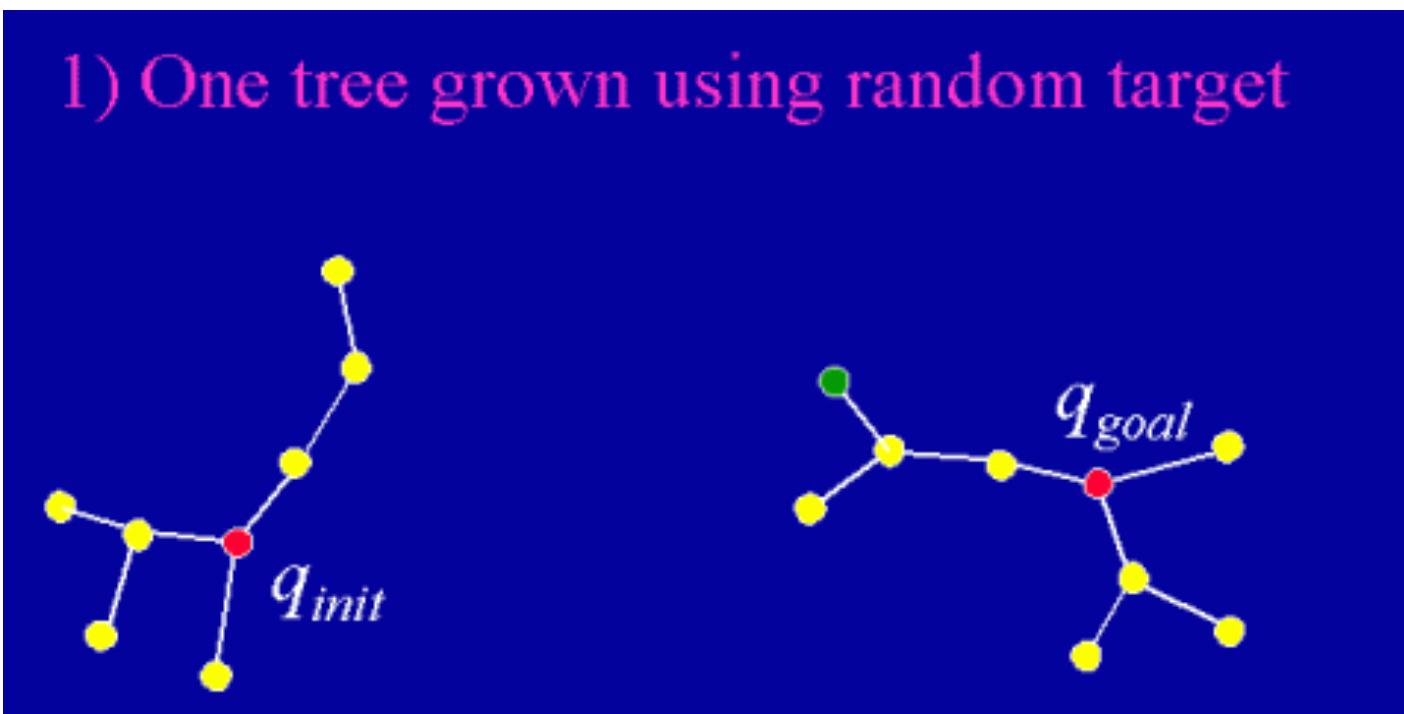
1 repeat
2    $S \leftarrow EXTEND(\mathcal{T}, q);$ 
3 until not ( $S = Advanced$ )
4 Return S;
```

- 2) Try to connect tree B to tree A by extending repeatedly from its nearest neighbor

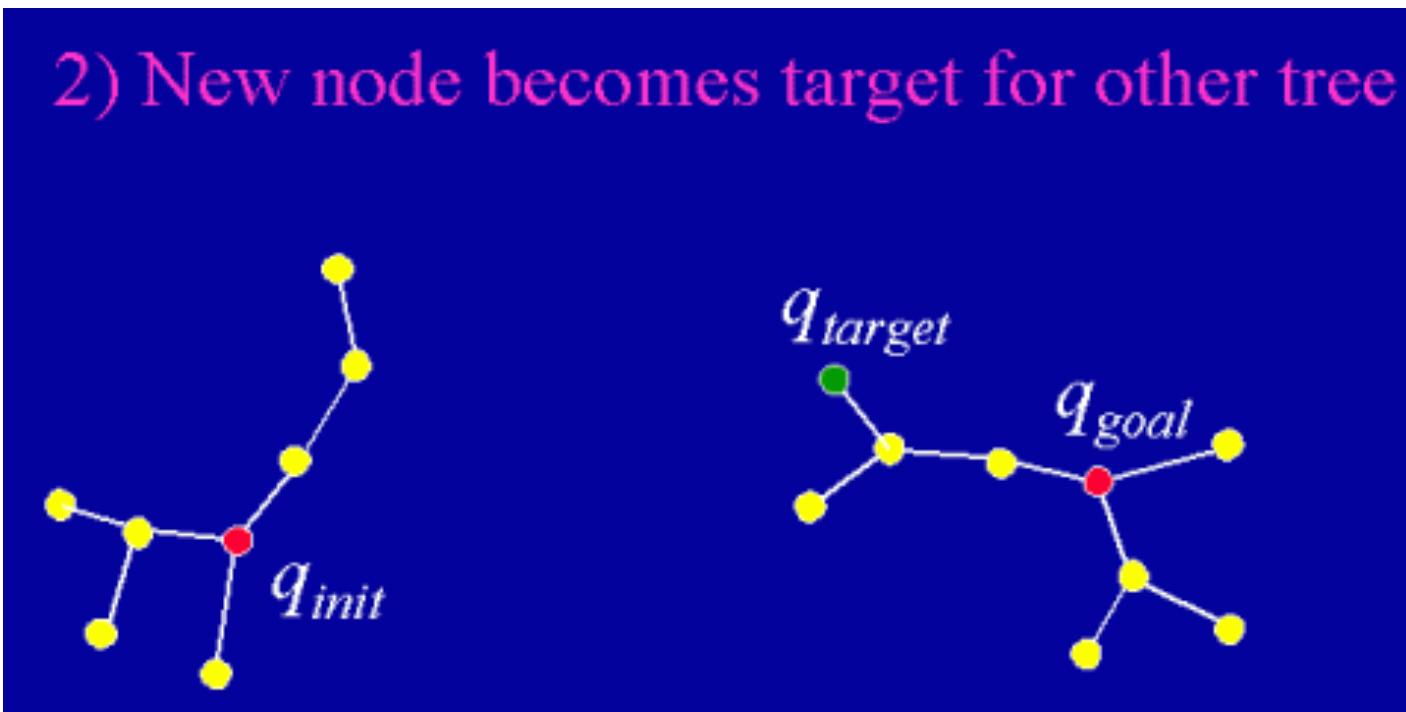
RRT-Connect animation



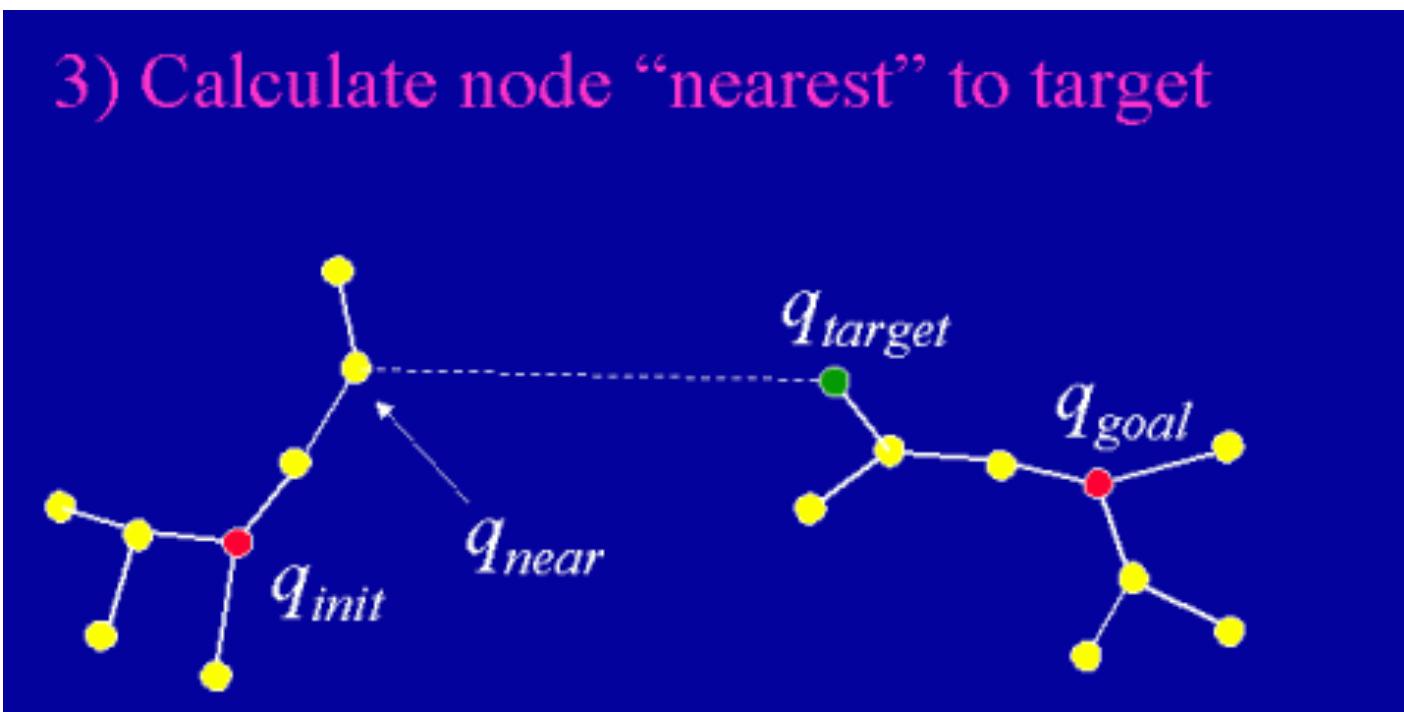
RRT-Connect animation



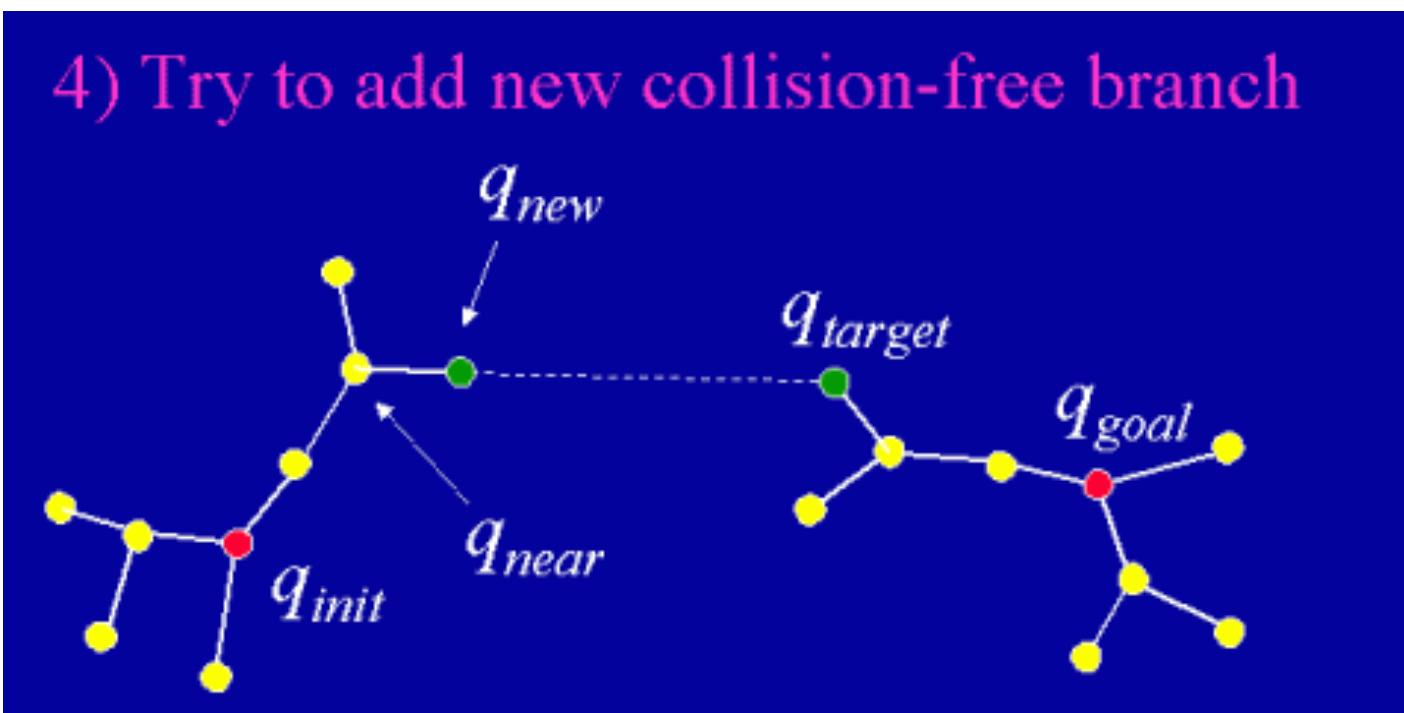
RRT-Connect animation



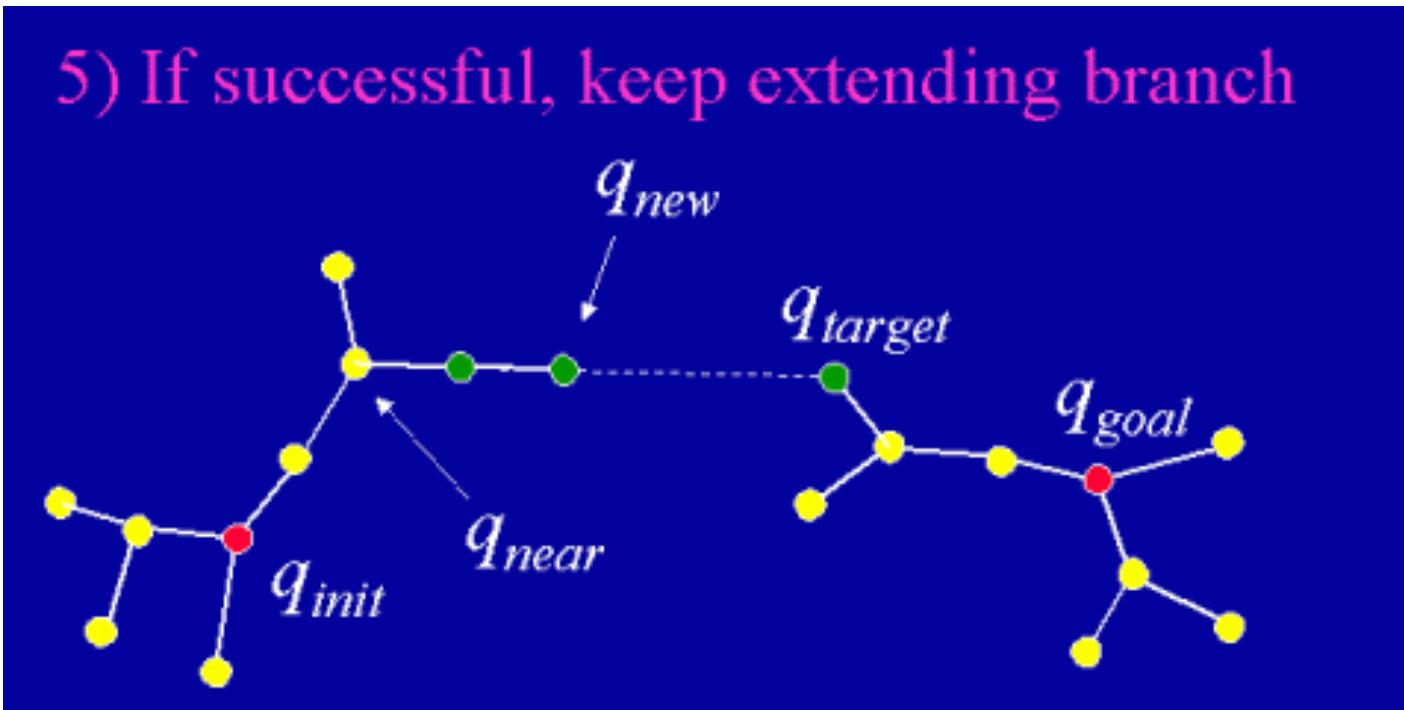
RRT-Connect animation



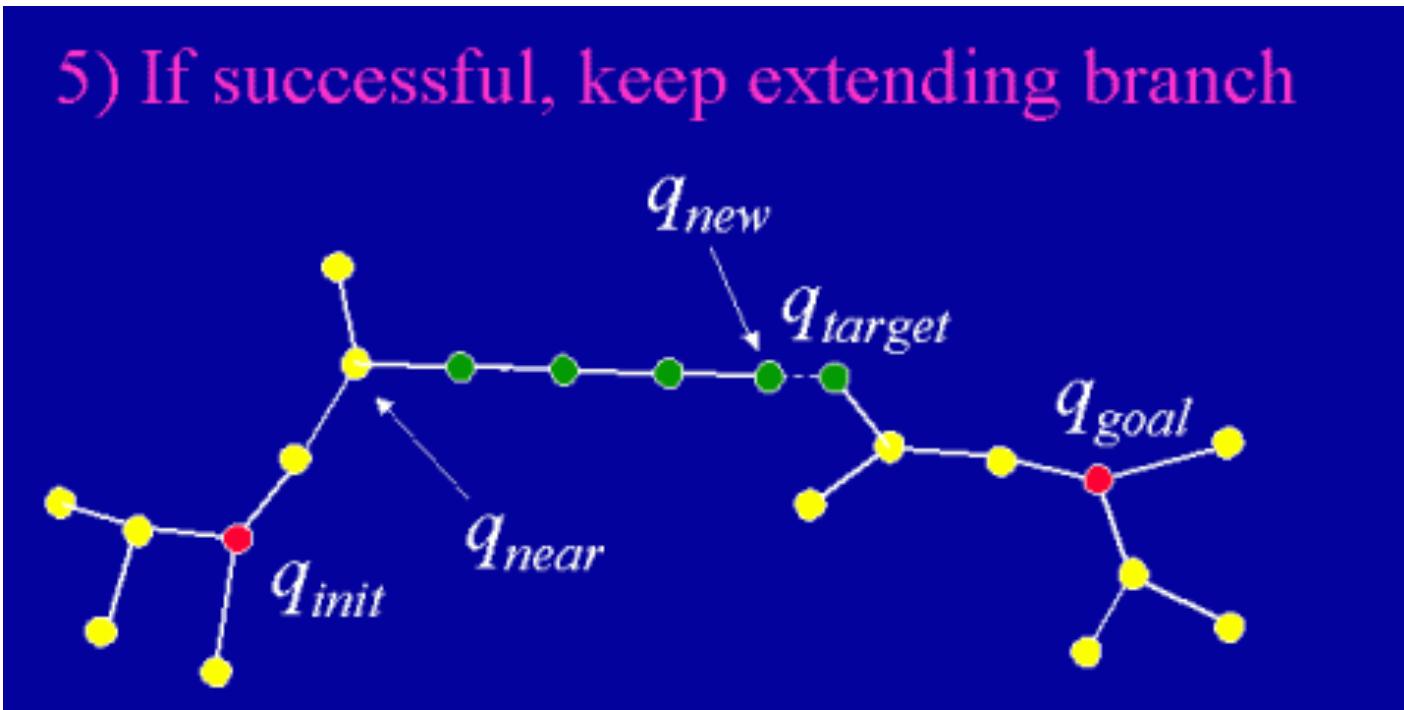
RRT-Connect animation



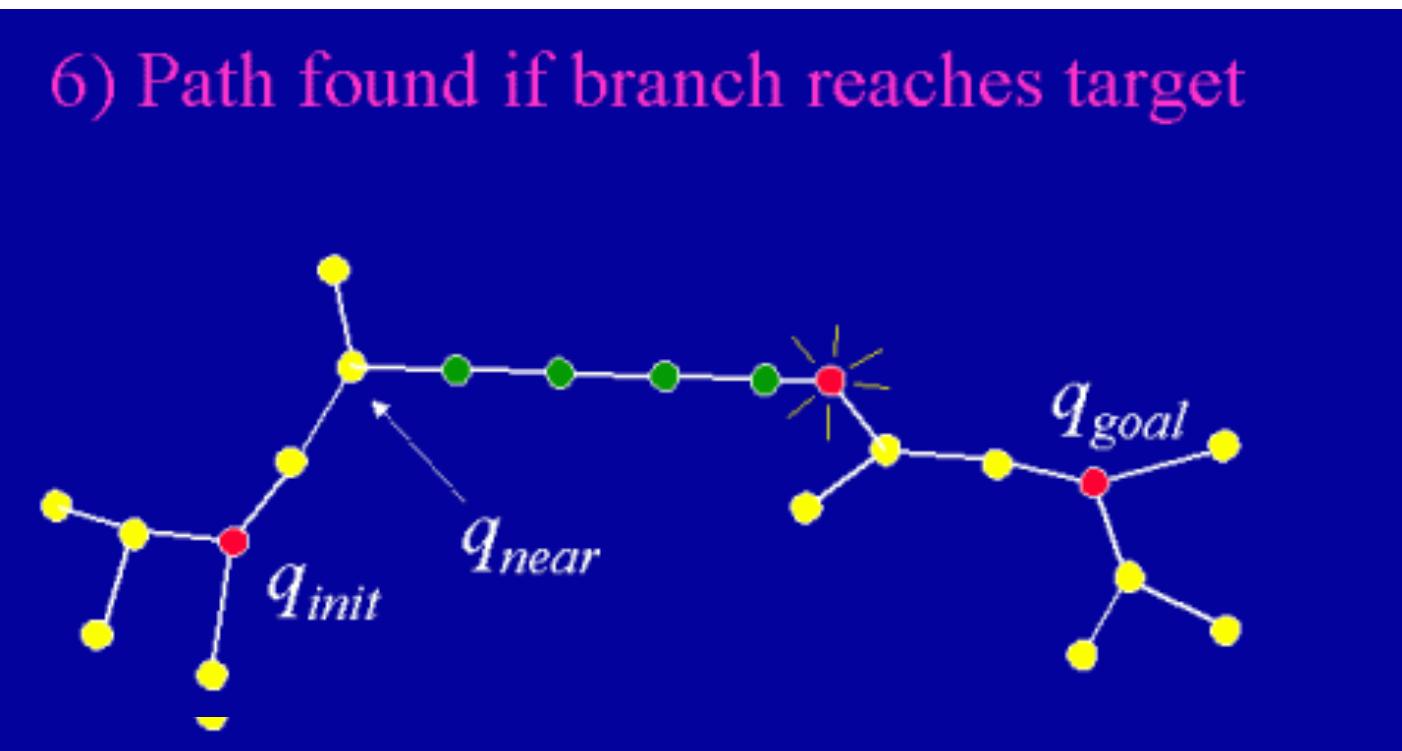
RRT-Connect animation



RRT-Connect animation

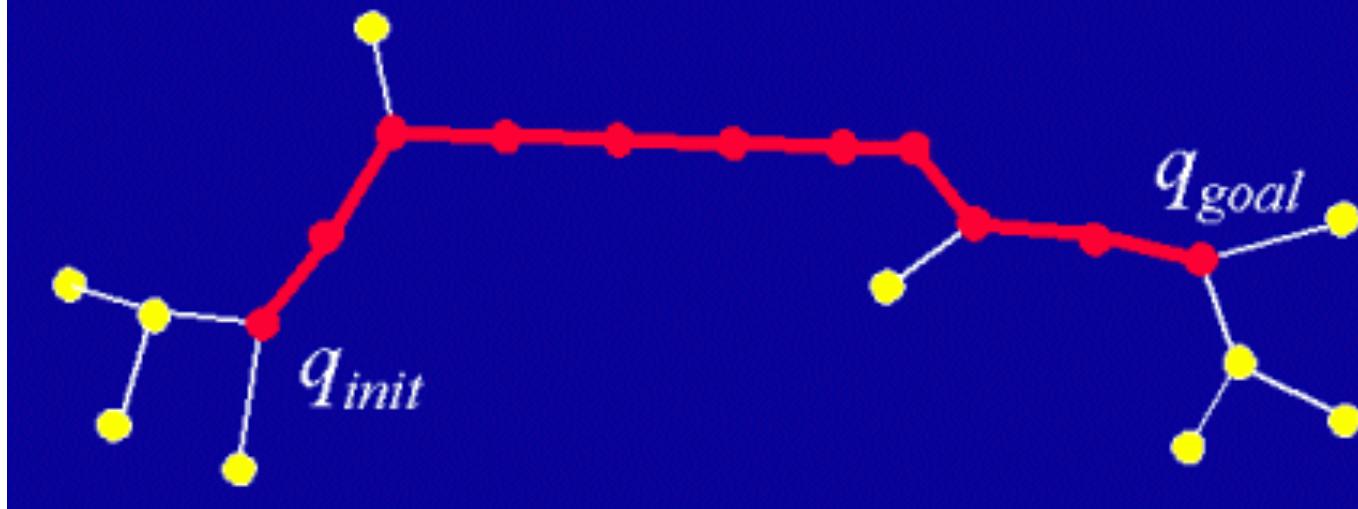


RRT-Connect animation

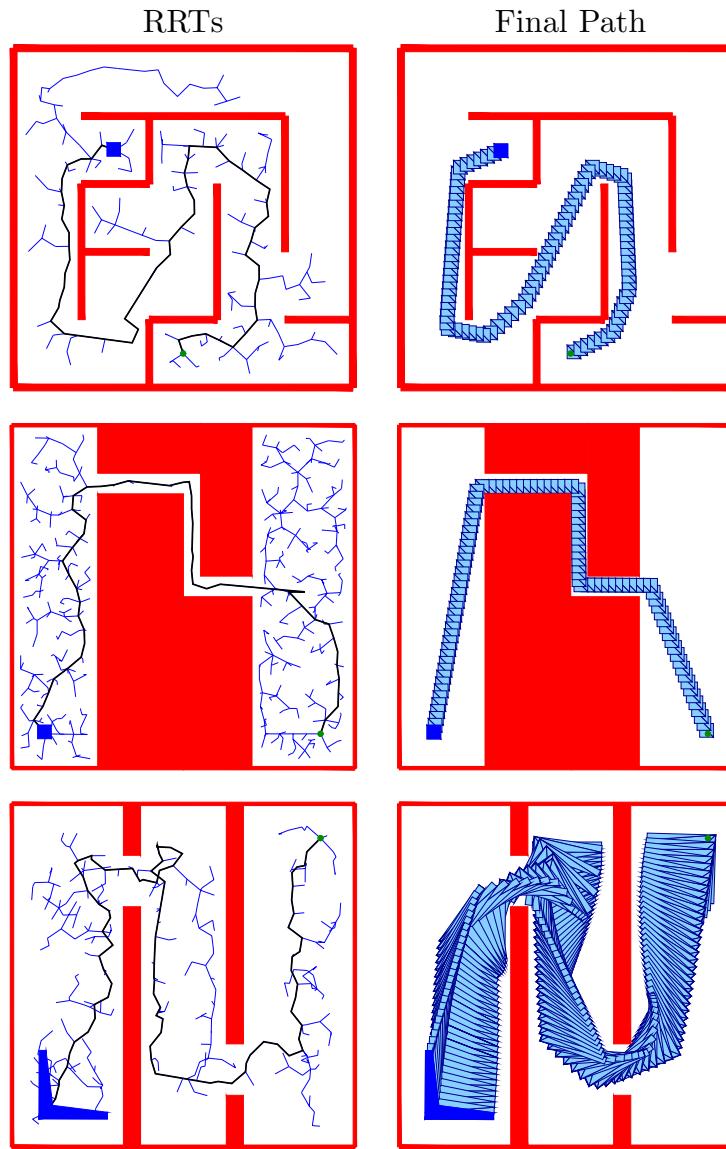


RRT-Connect animation

7) Return path connecting start and goal



Examples of RRT-Connect

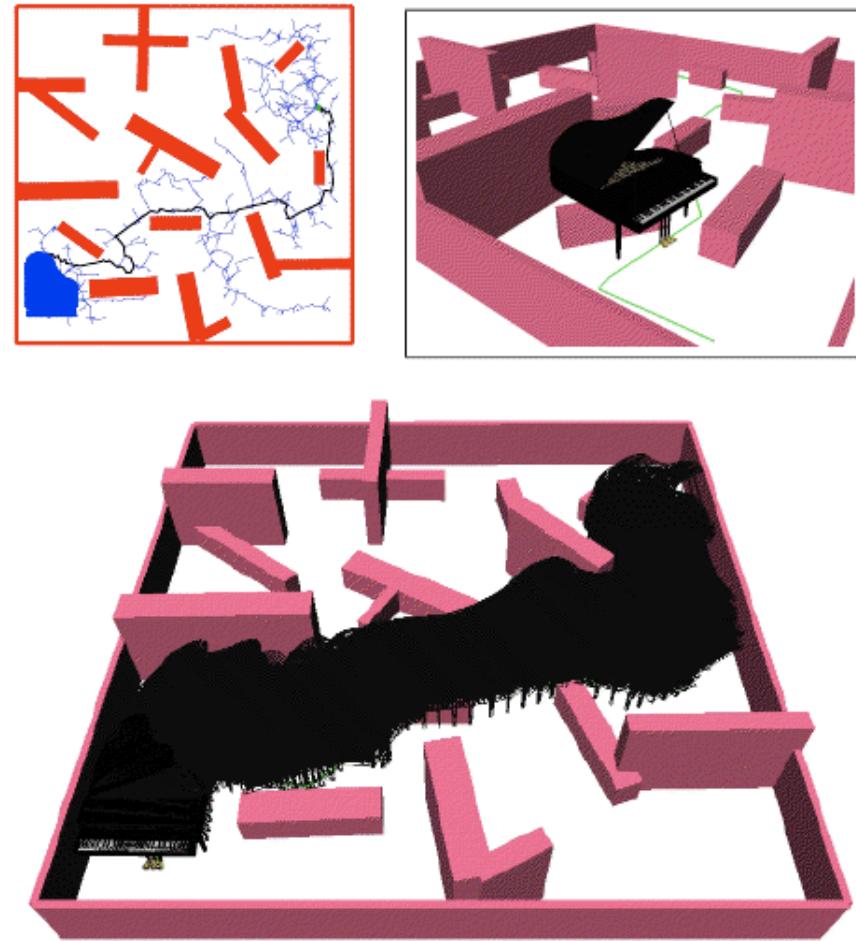


2 DOF maze

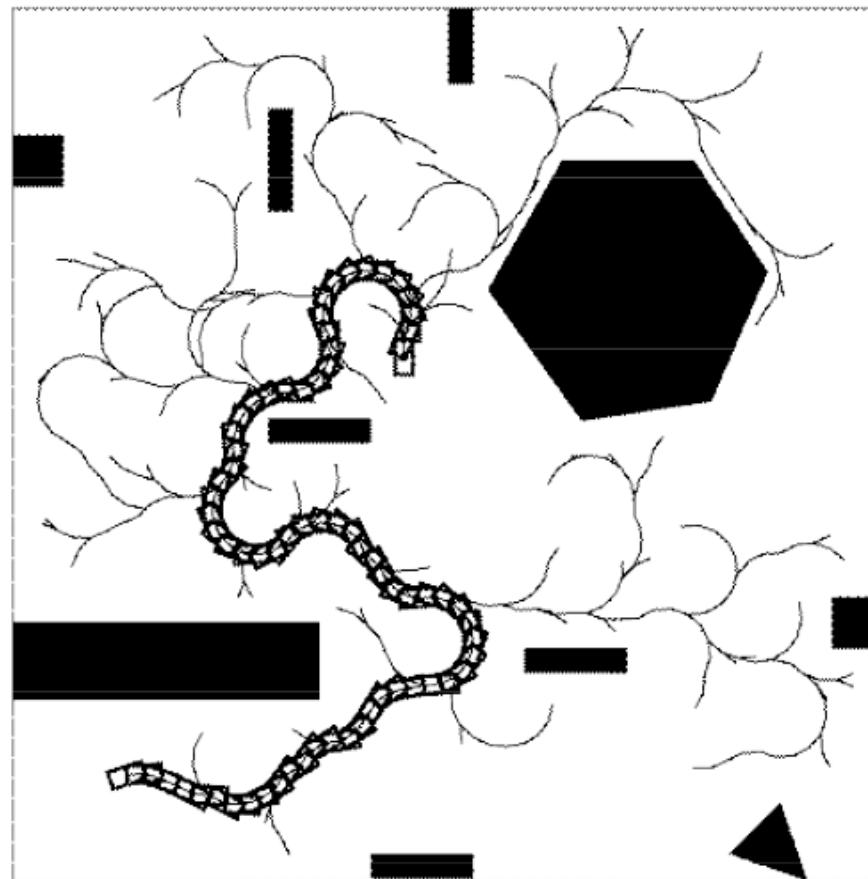
2 DOF single passway

3 DOF single passway
(with non-point geometry)

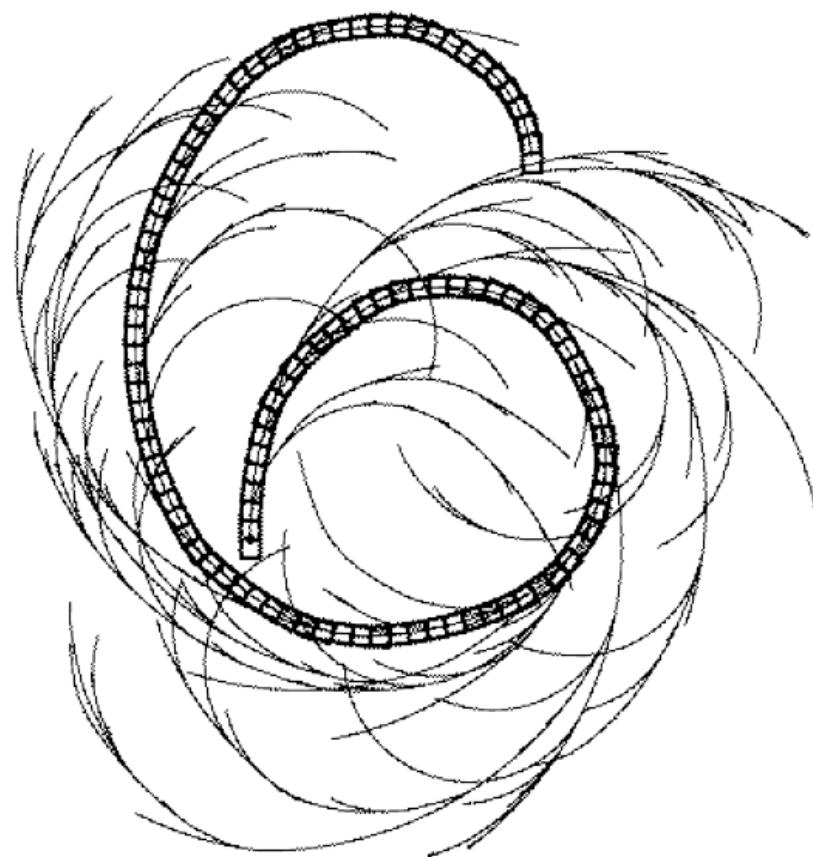
Piano Mover's Problem



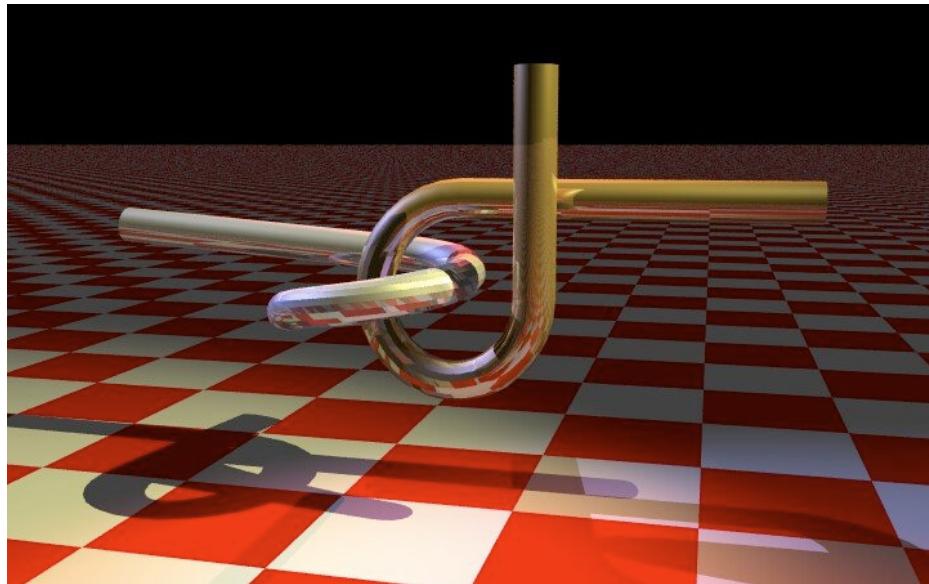
A Car-Like Robot



A Right-Turn Only Car

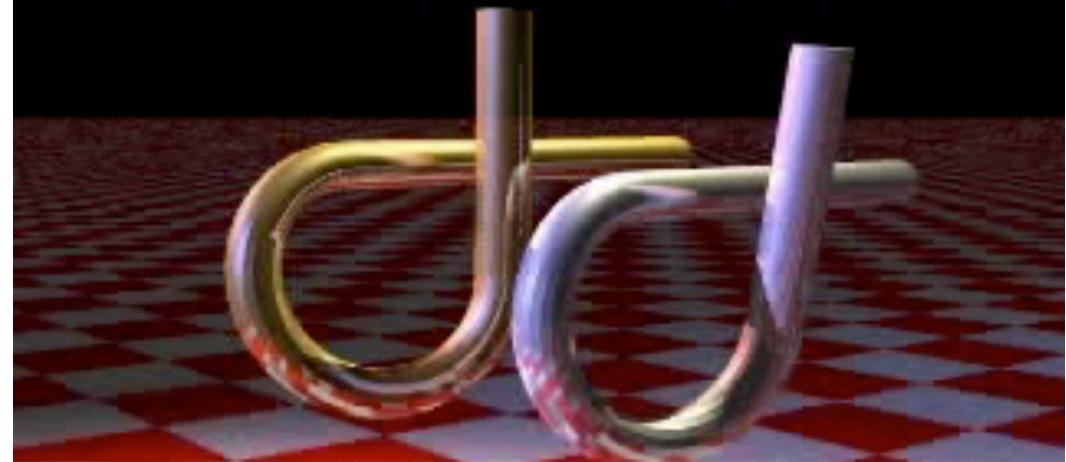


Alpha Puzzle



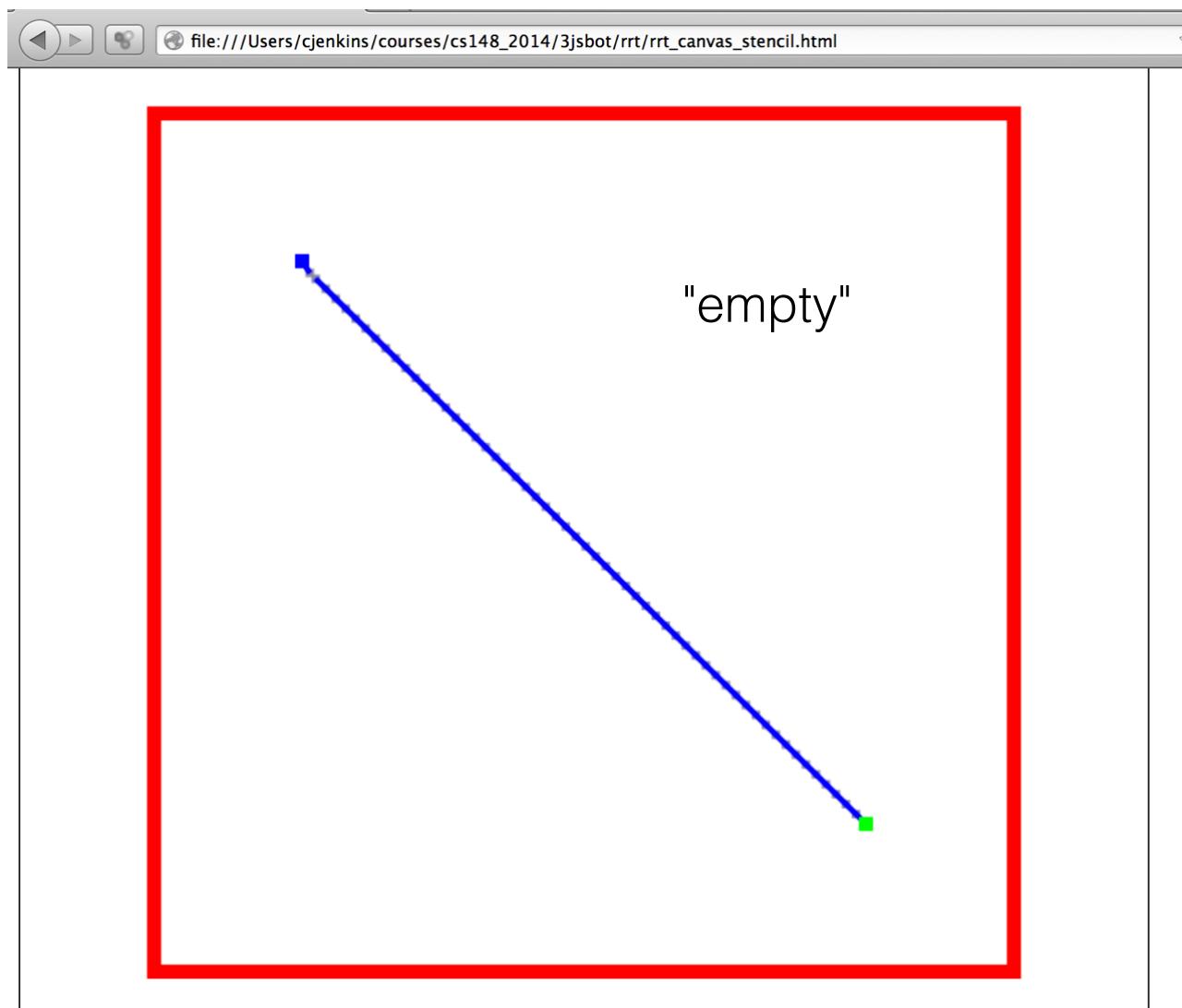
Alpha Puzzle 1.0 Solution

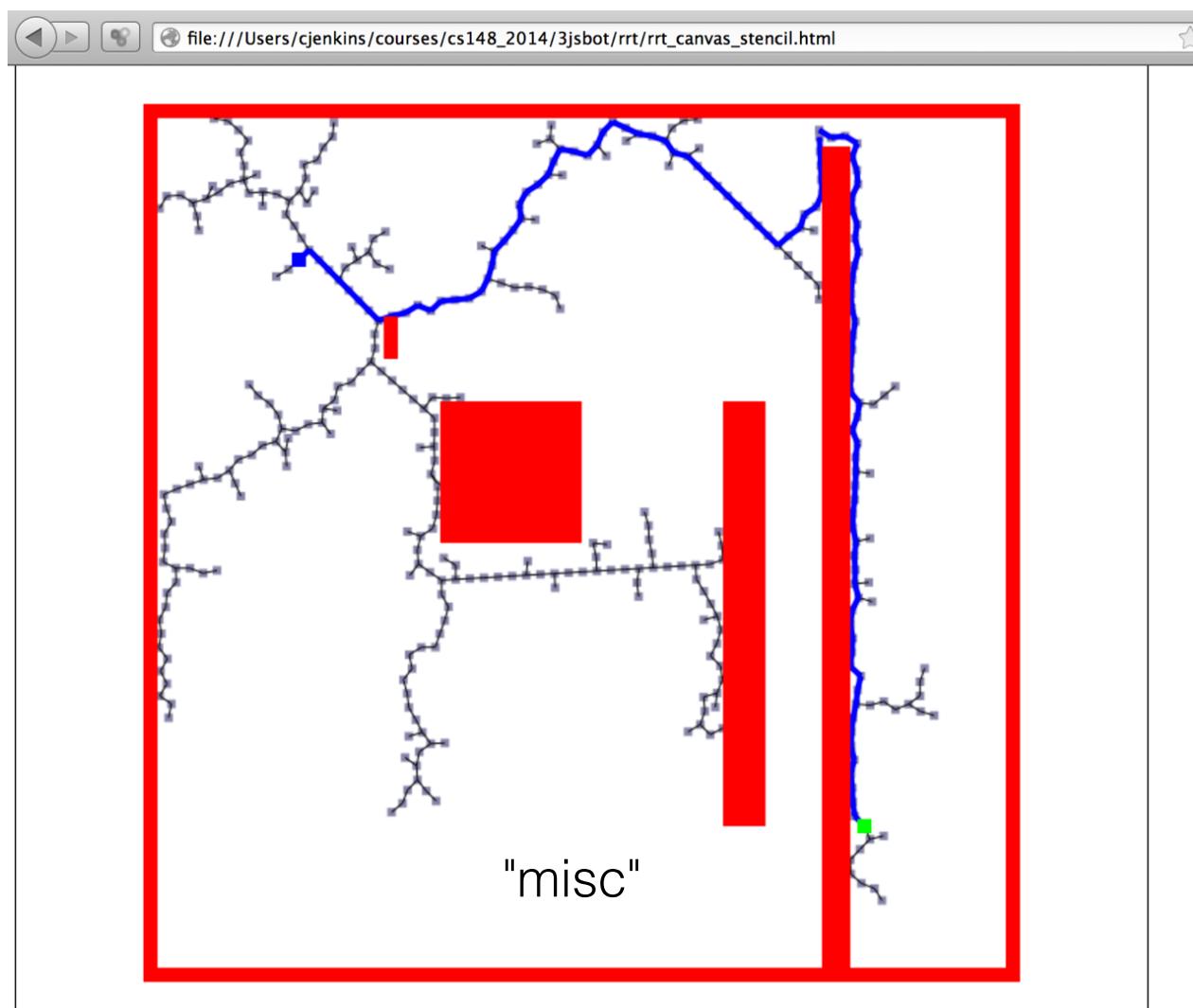
James Kuffner, Feb. 2001

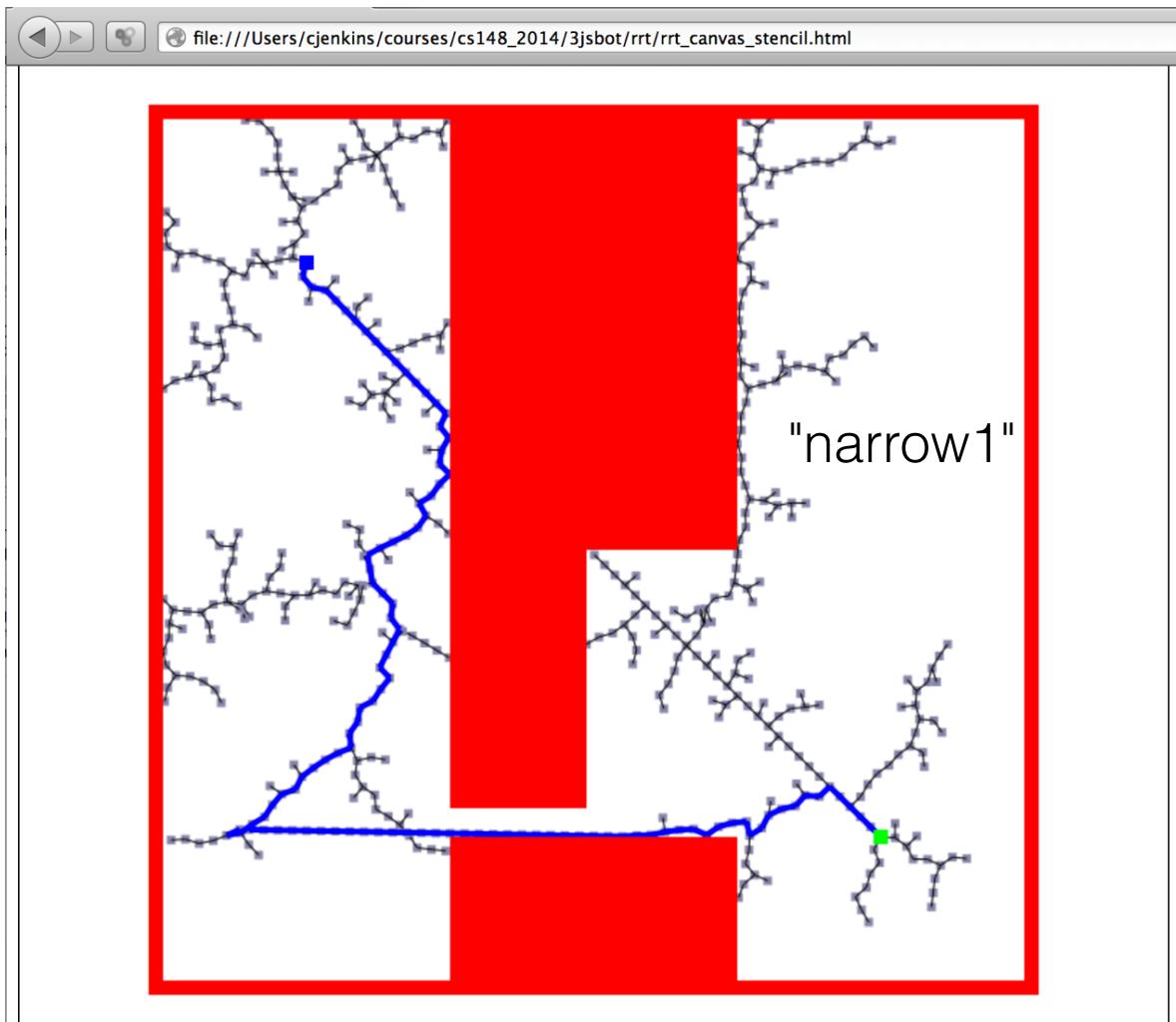


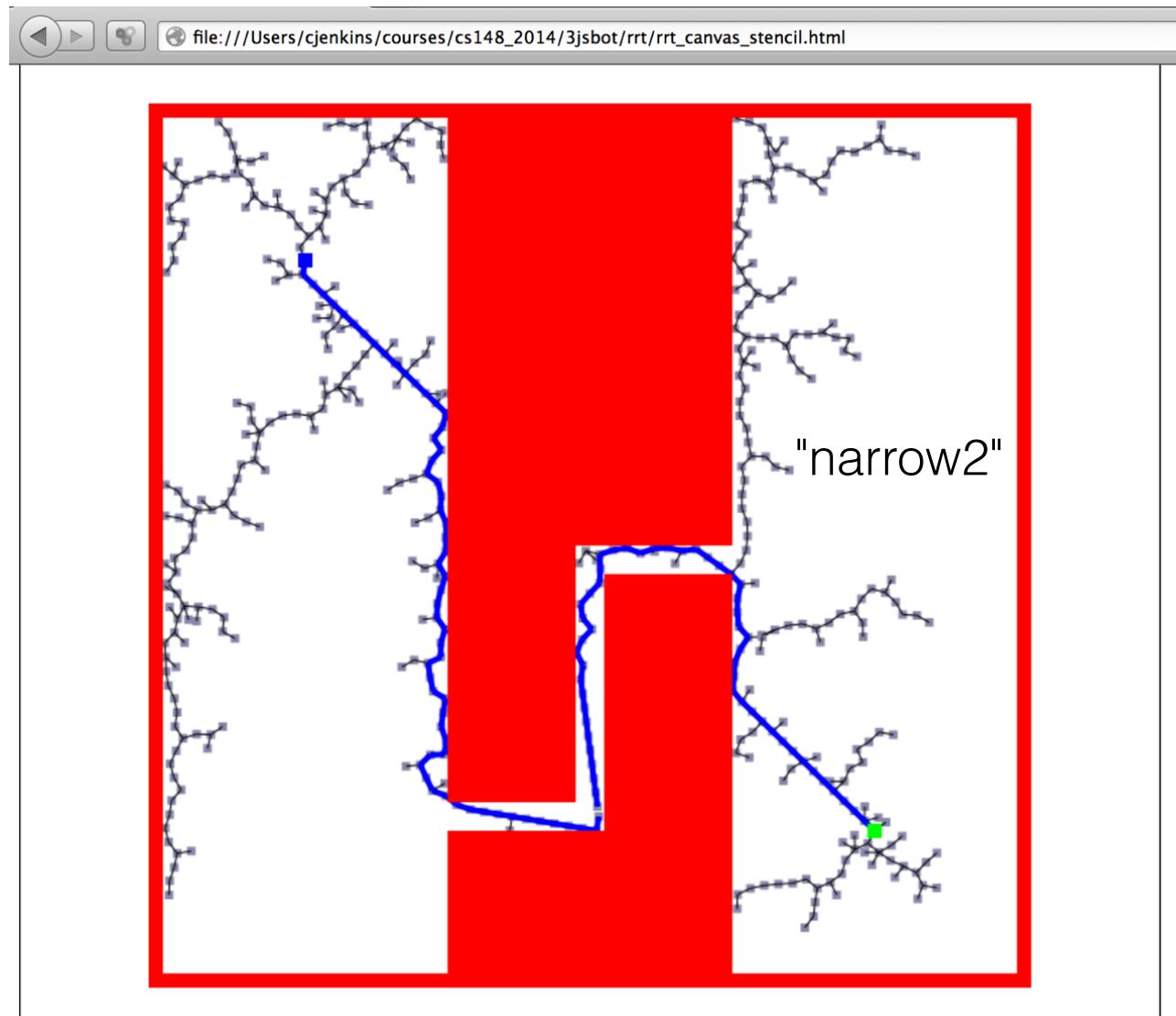
model by DSMFT group, Texas A&M Univ.
original model by Boris Yamrom, GE

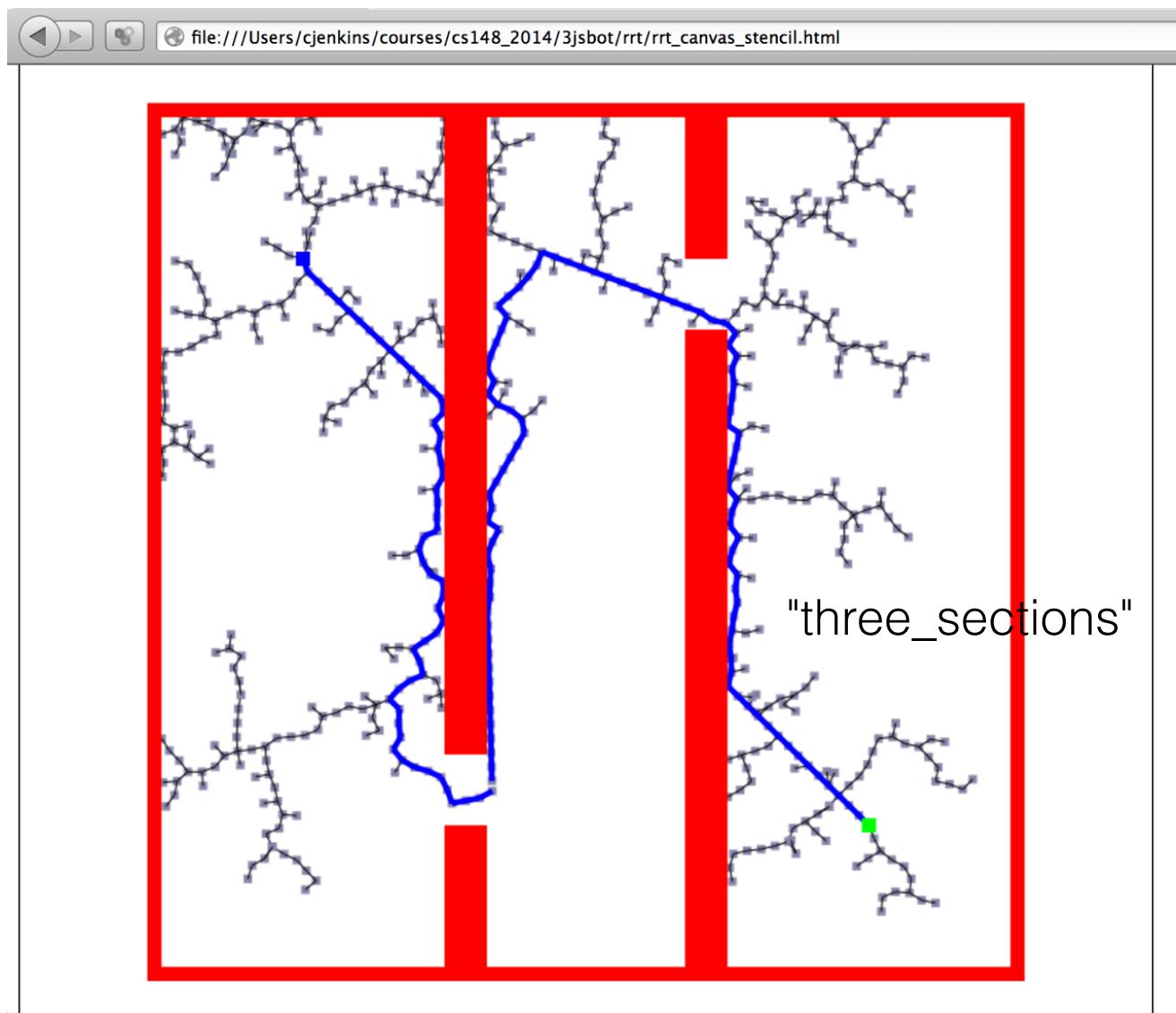
Canvas Stencil Examples



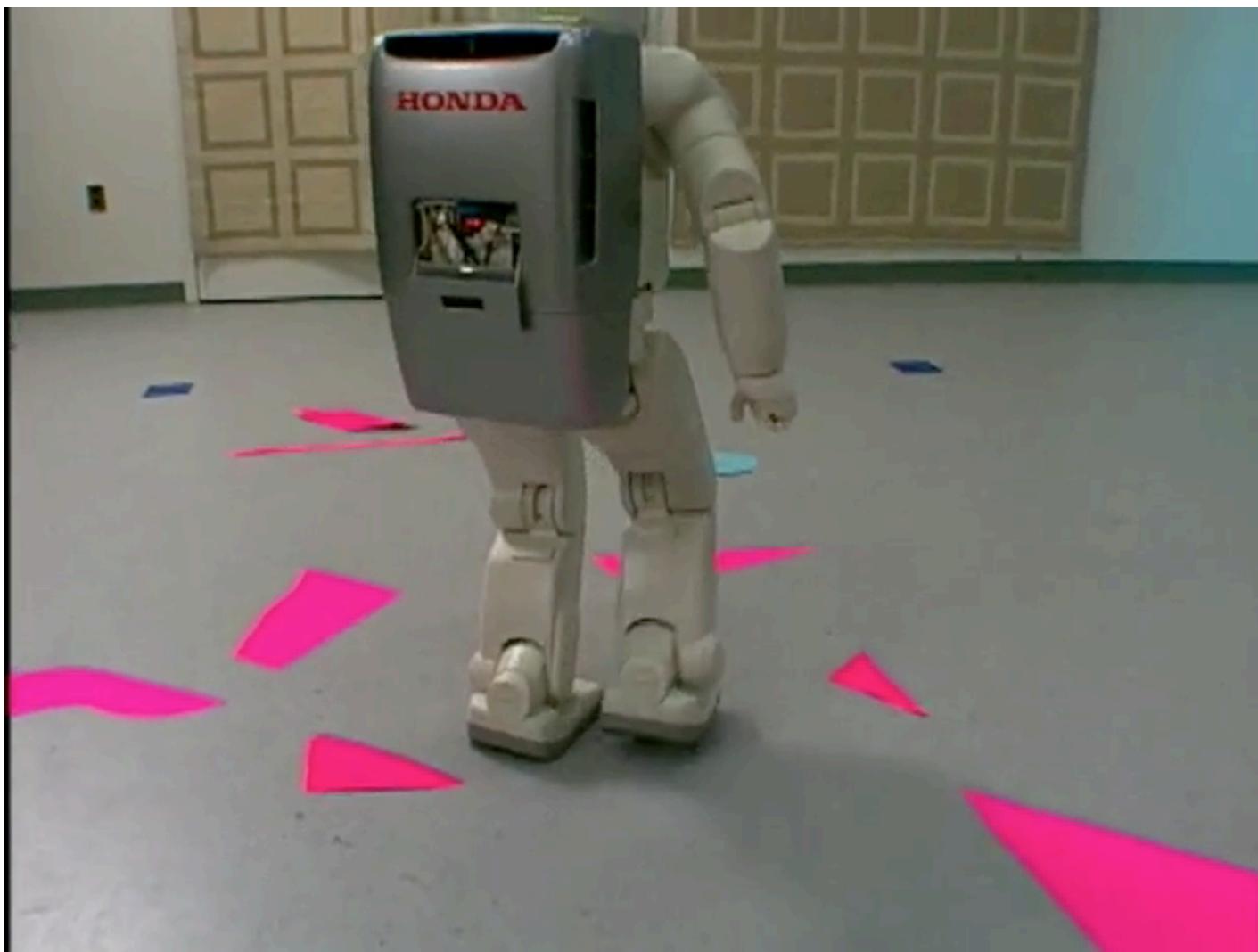








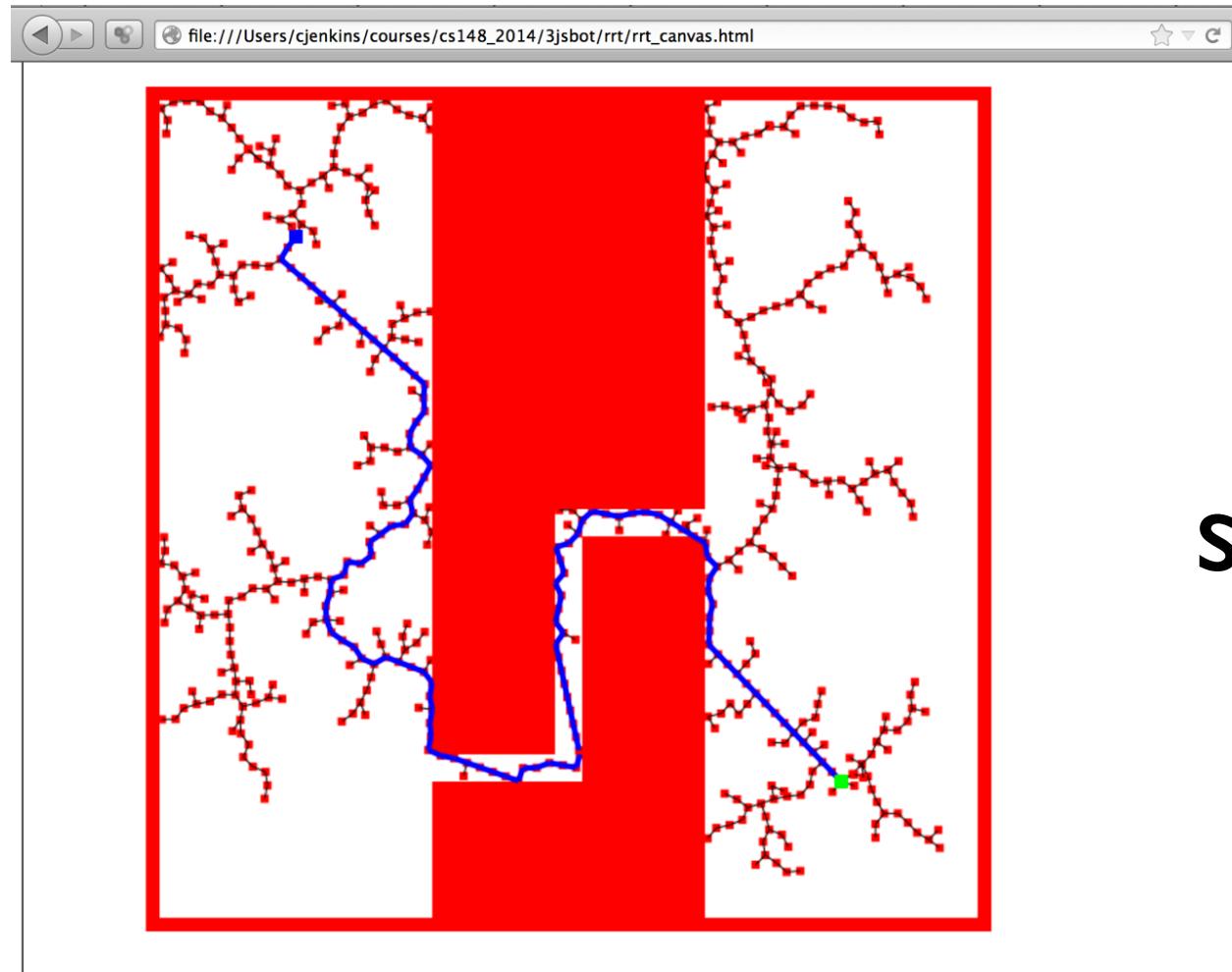
“We've made robot history”



Kuffner/Asimo Discovery Channel feature - <https://www.youtube.com/watch?v=wtVmhiTfm0Q>

Michigan EECS 398/567 ROB 510 - autorob.org

RRTs can take a lot of time...



Is there a
simpler way?



Potential fields

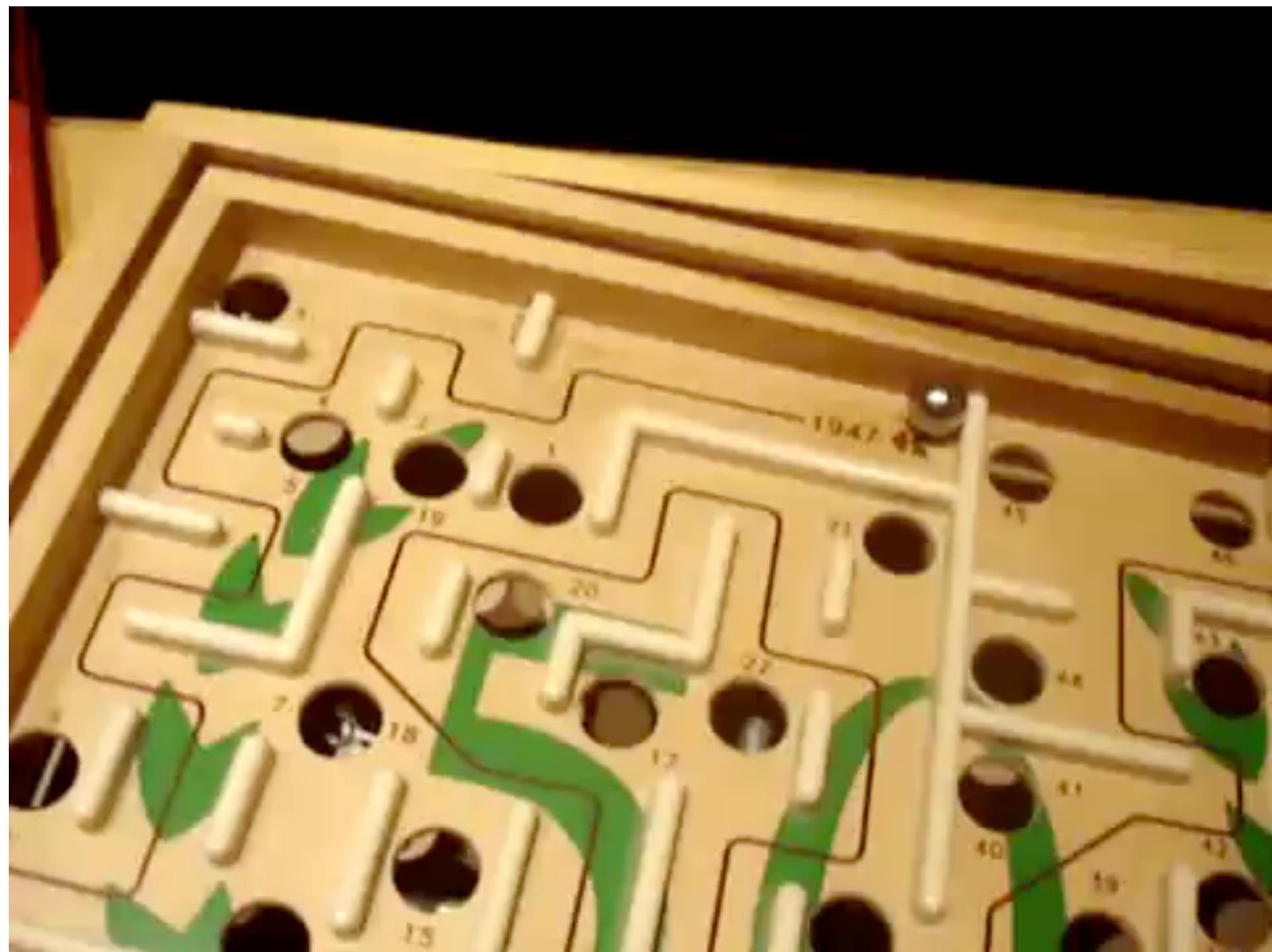
follow your
potential

Approaches to motion planning

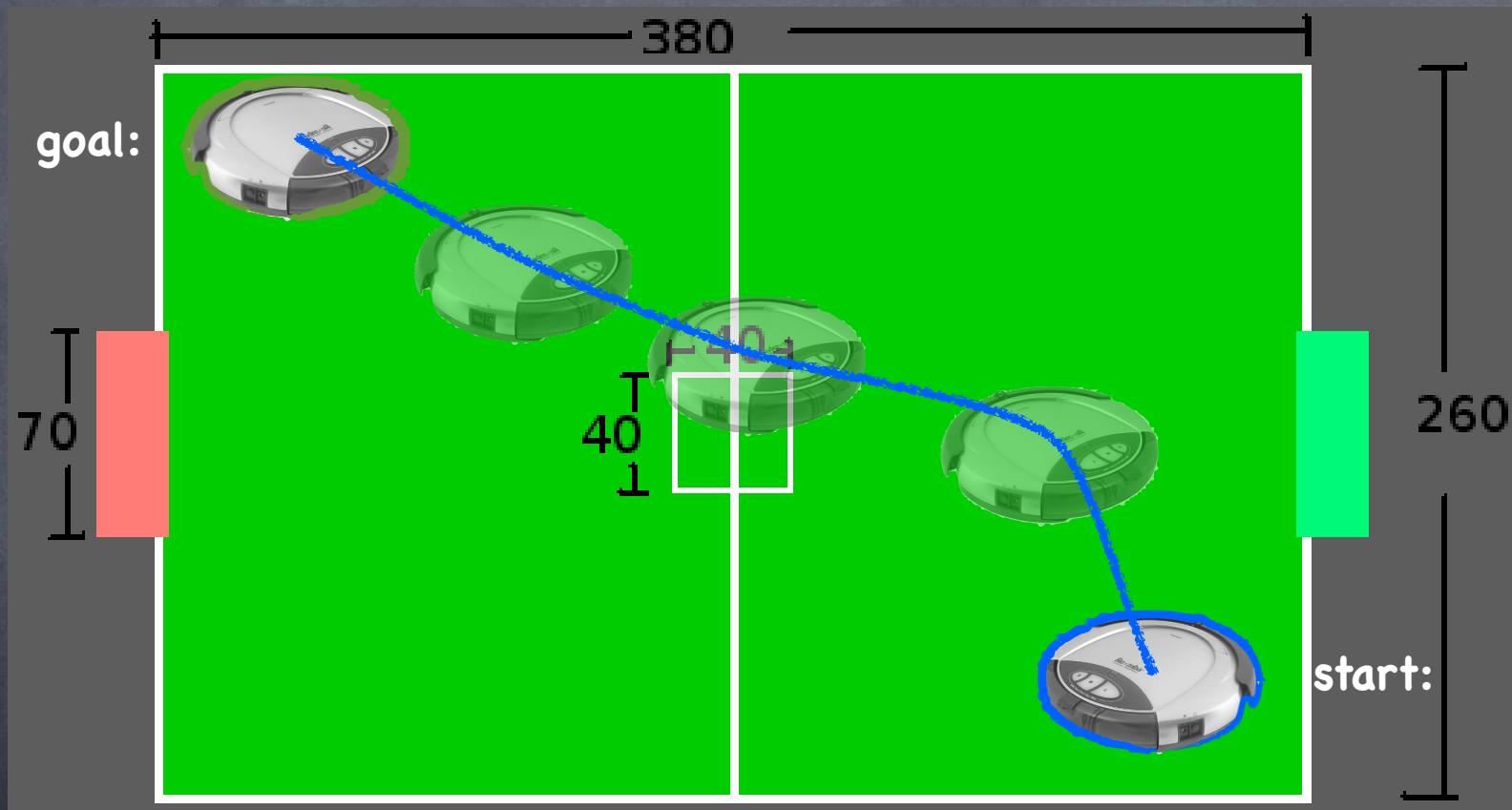
- Bug algorithms: Bug[0-2], Tangent Bug
- Graph Search (fixed graph)
 - Depth-first, Breadth-first, Dijkstra, A-star, Greedy best-first
- Sampling-based Search (build graph):
 - Probabilistic Road Maps, Rapidly-exploring Random Trees
- **Optimization and local search:**
 - **Gradient descent, Potential fields, Simulated annealing, Wavefront**



Michigan E360 Society, RoboCup 10 - autorob.org

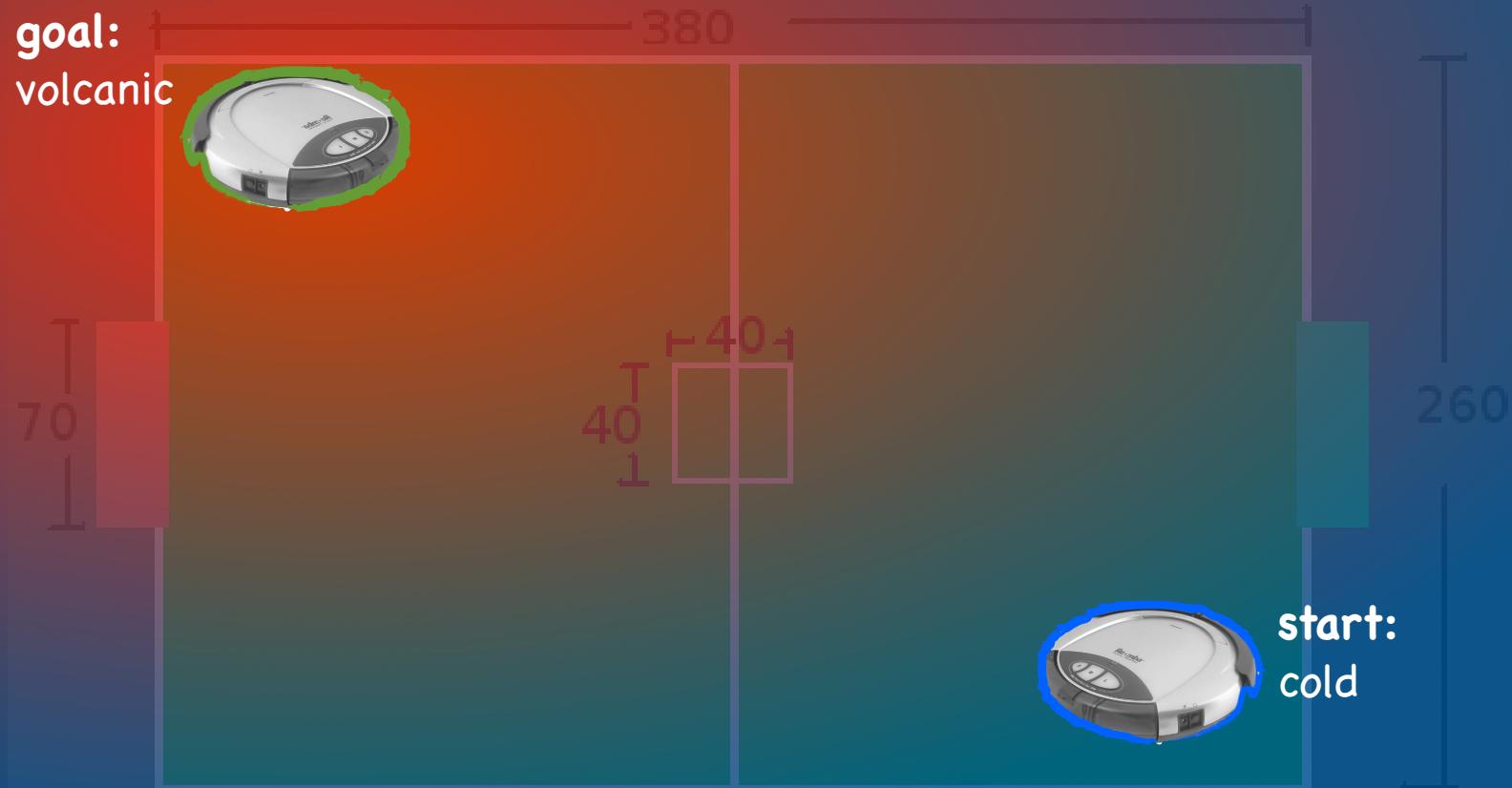


Navigation (again)



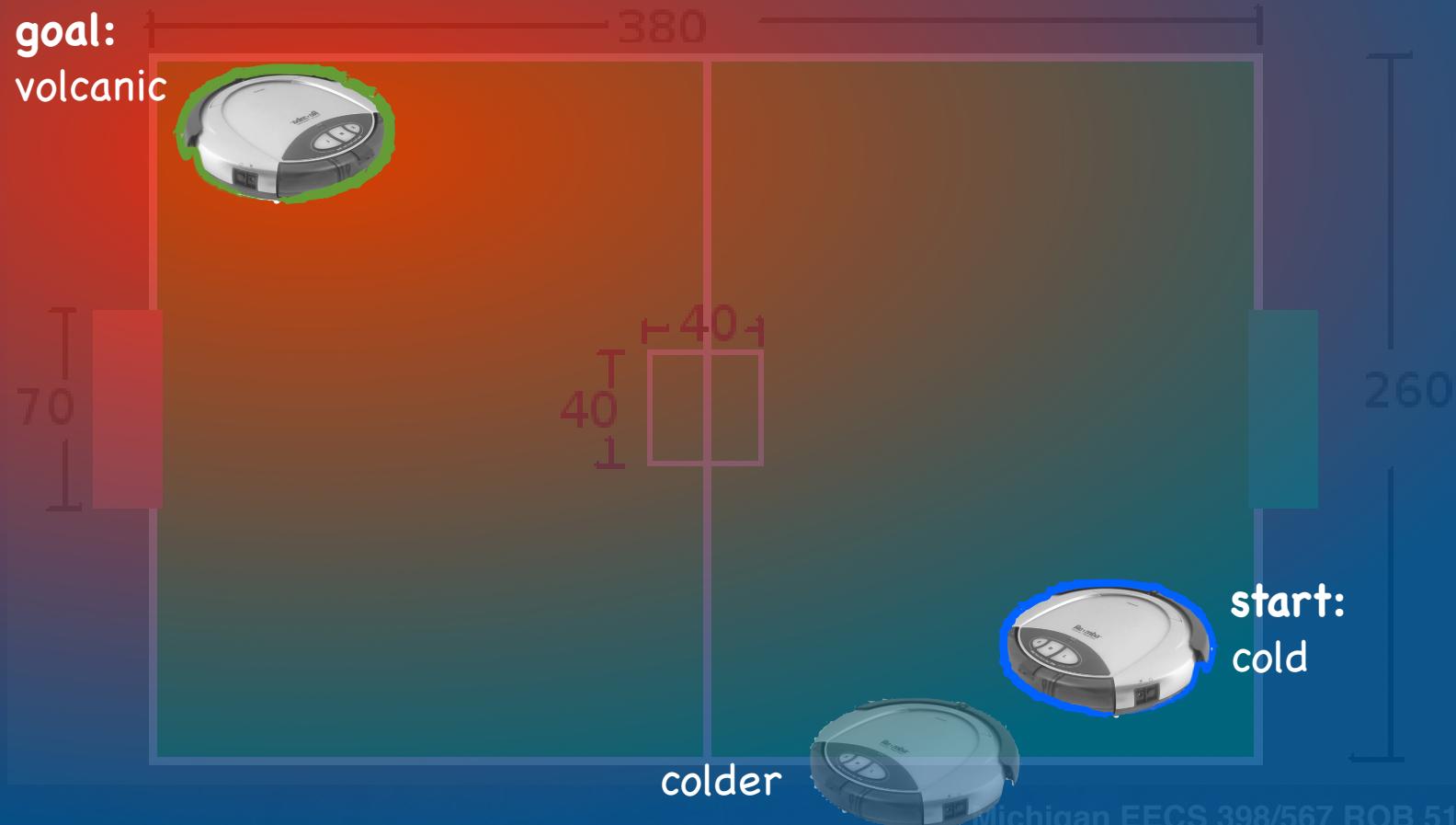
Potential field

(like a game of “warmer-colder”)



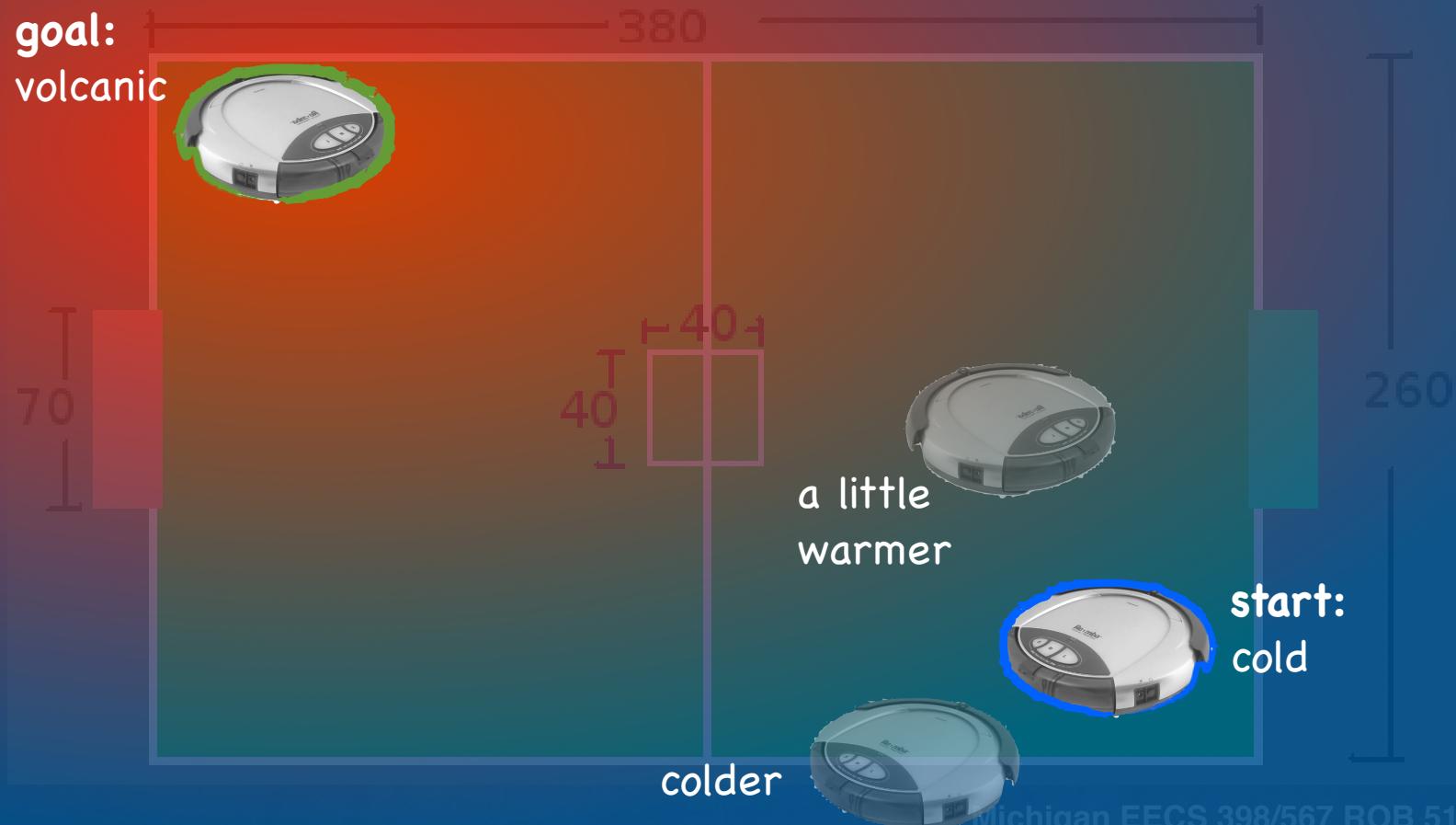
Potential field

(like a game of “warmer-colder”)



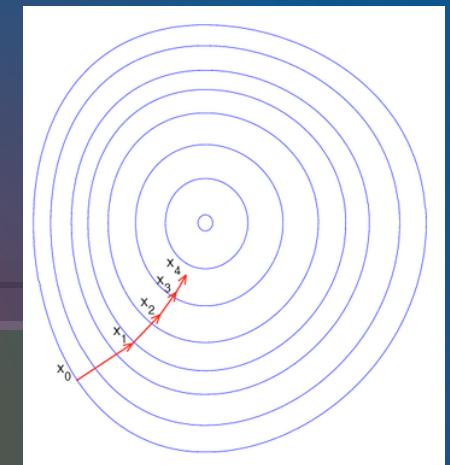
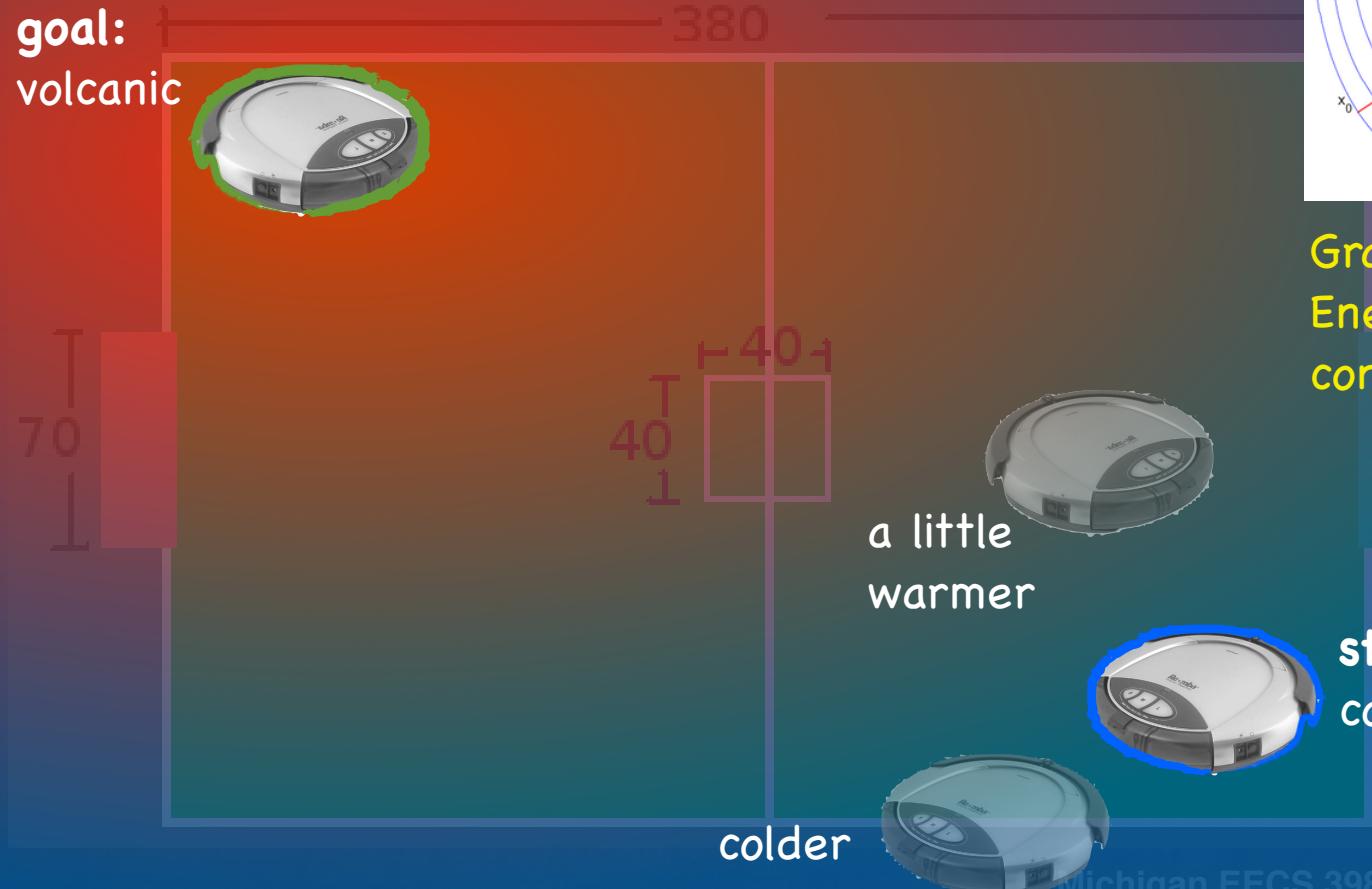
Potential field

(like a game of “warmer-colder”)



Potential field

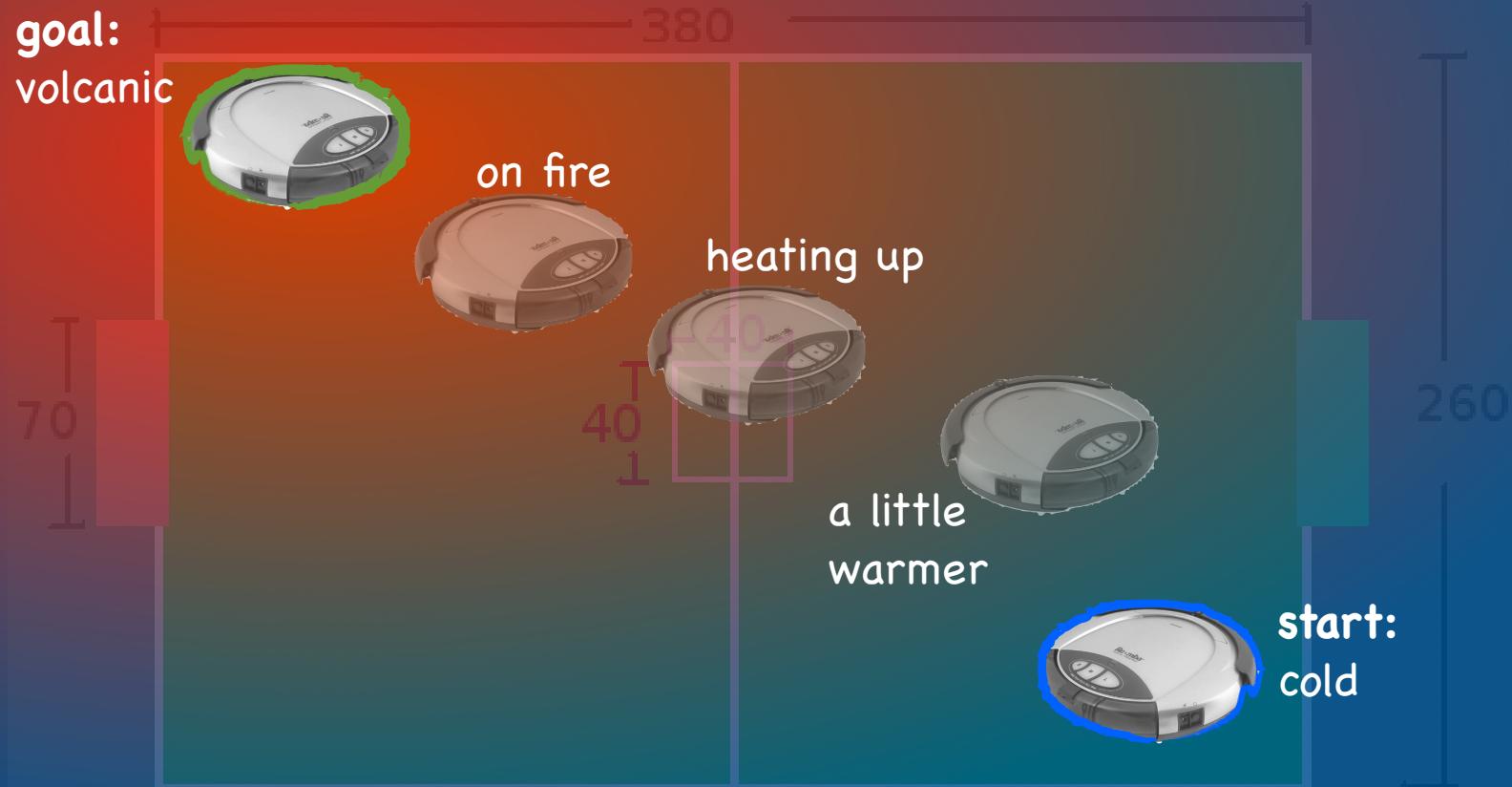
(like a game of “warmer-colder”)



Gradient descent:
Energy potential
converges at goal

Potential field

(like a game of “warmer-colder”)



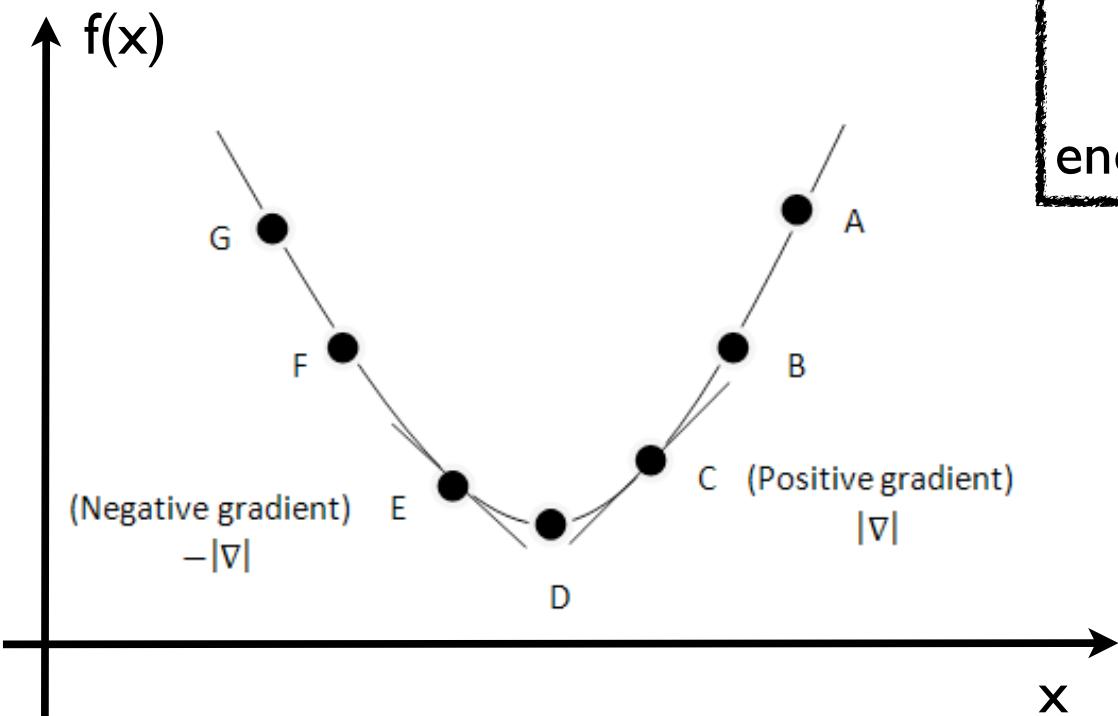
How do we define a
potential field?

Potential Field

- A potential field is a differentiable function $U(q)$ that maps configurations to scalar “energy” value
- At any q , gradient $\nabla U(q)$ is the vector that maximally increases U
- At goal q_{goal} , energy is minimized such that $\nabla U(q_{goal}) = 0$
- Navigation by descending field $-\nabla U(q)$ to goal

Gradient descent

From Wikipedia, the free encyclopedia



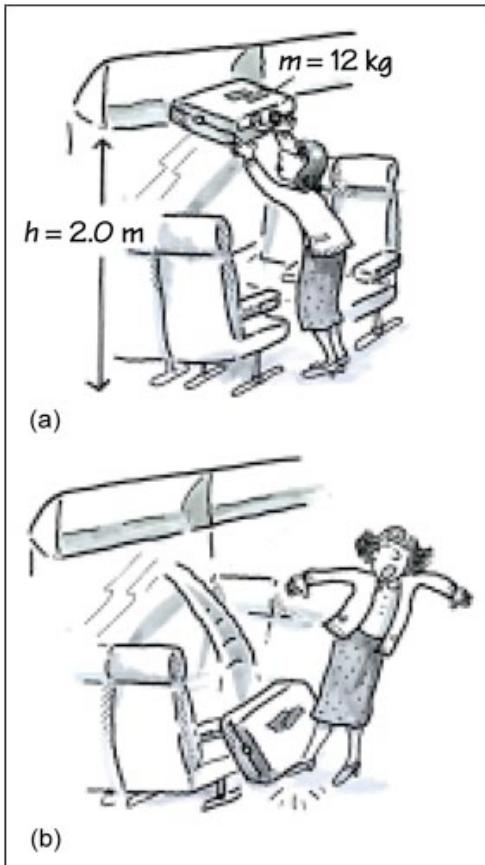
Gradient Descent Algorithm:

```
qpath[0] ← qstart
i ← 0
while (||∇U(q[i])|| > ε)
    qpath[i+1] ← qpath[i] - α ∇U(qpath[i])
    i ← i+1
end
```

Derivative assumed to be direction
of steepest ascent away from goal

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)$$

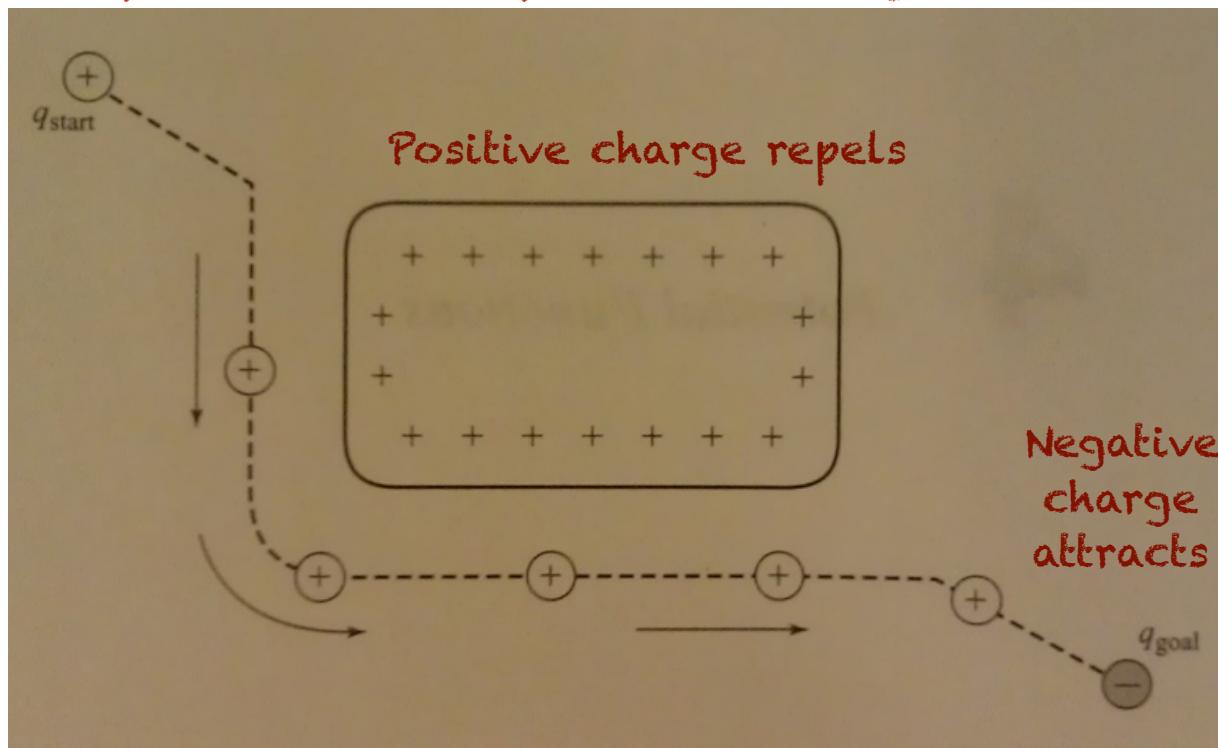
Potential Energy



- Energy stored in a physical system
- Kinetic motion caused by system moving to lower energy state
- For objects acting only w.r.t. gravity
- $\text{potential_energy} = \text{mass} * \text{height} * \text{gravity}$

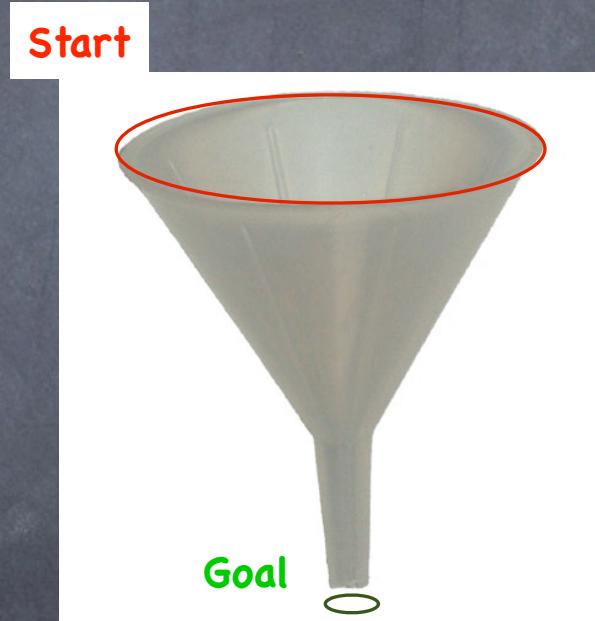
Charged Particle Example

Positively charged particle follows potential energy to goal



Convergent Potentials

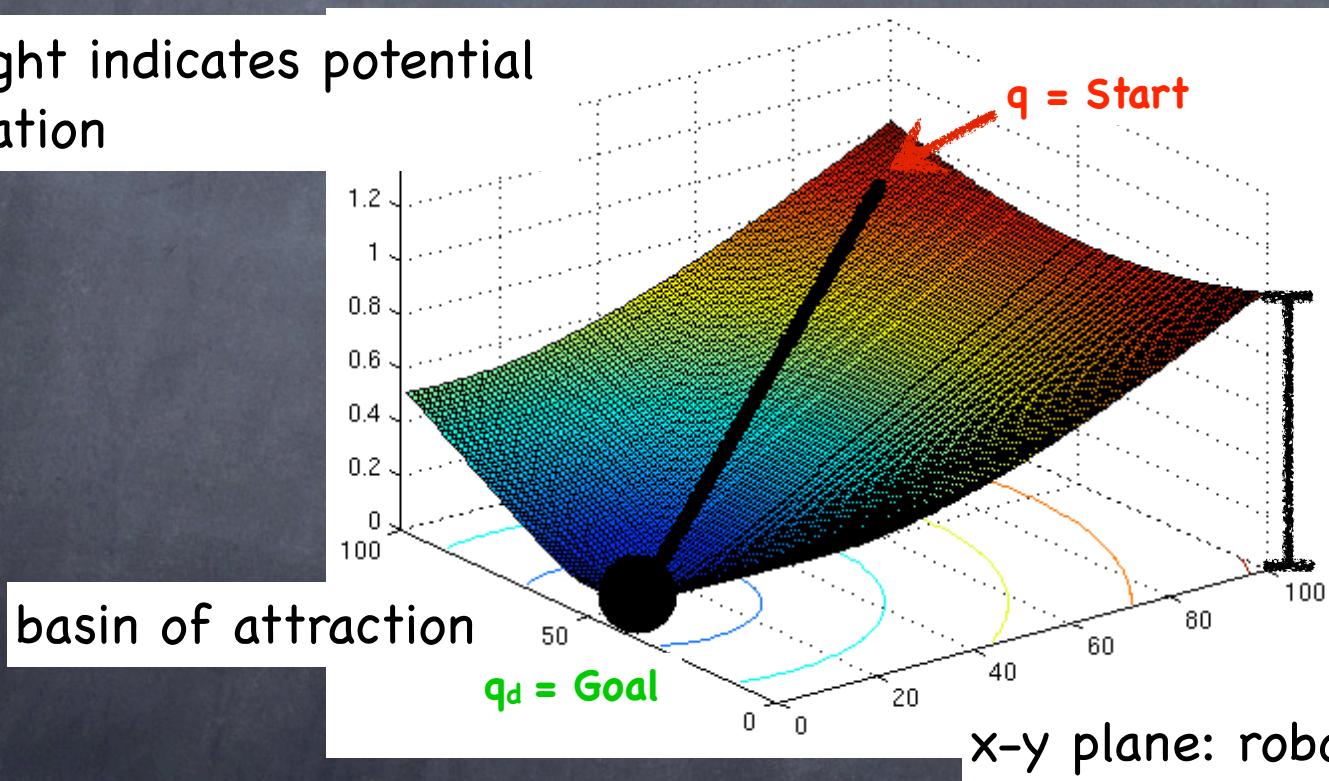
let's call these "attractor landscapes"



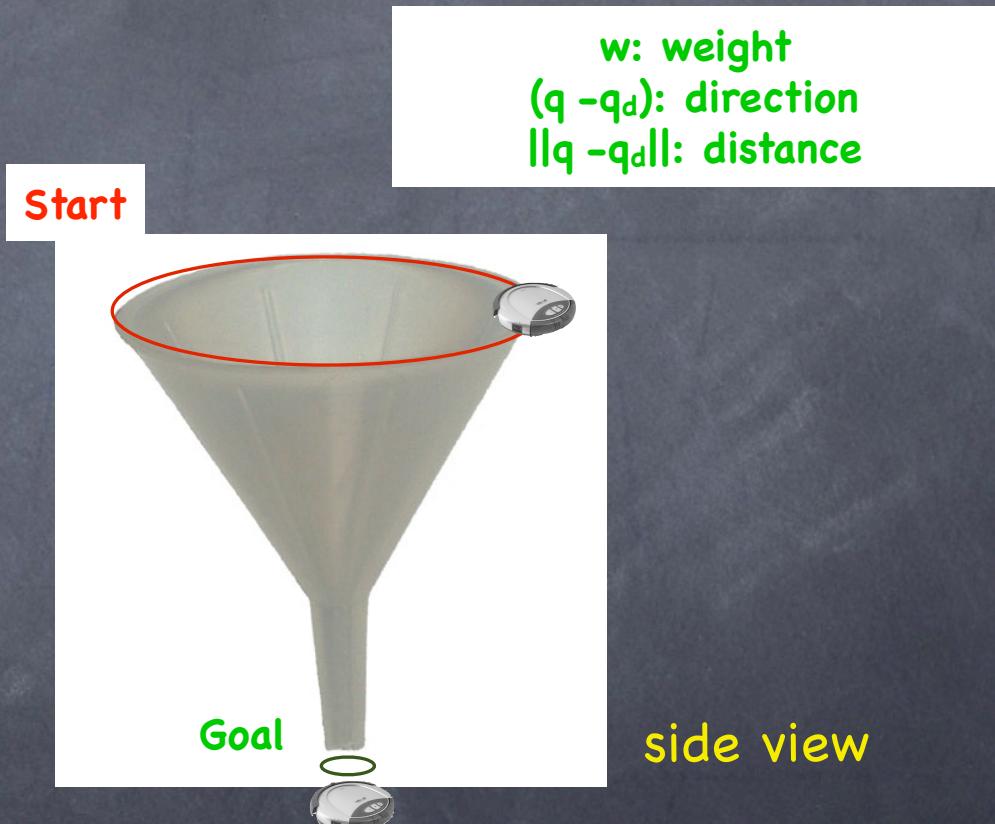
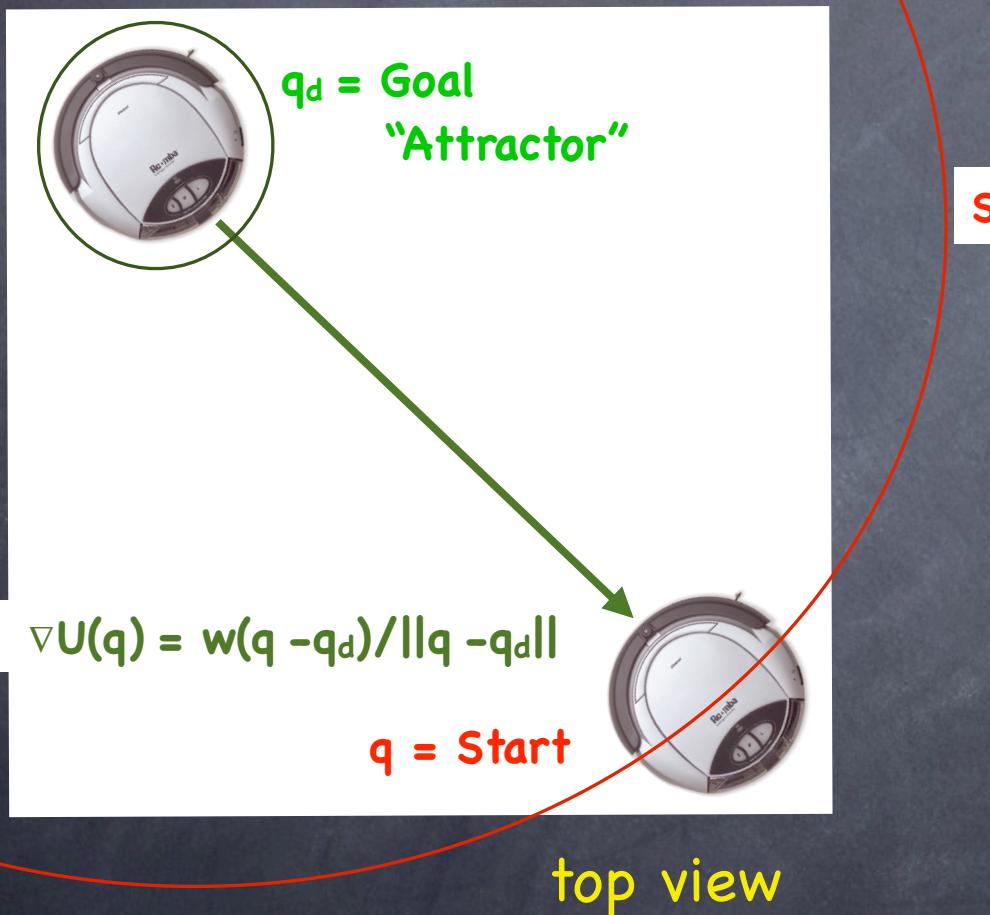
basin of attraction

2D potential navigation

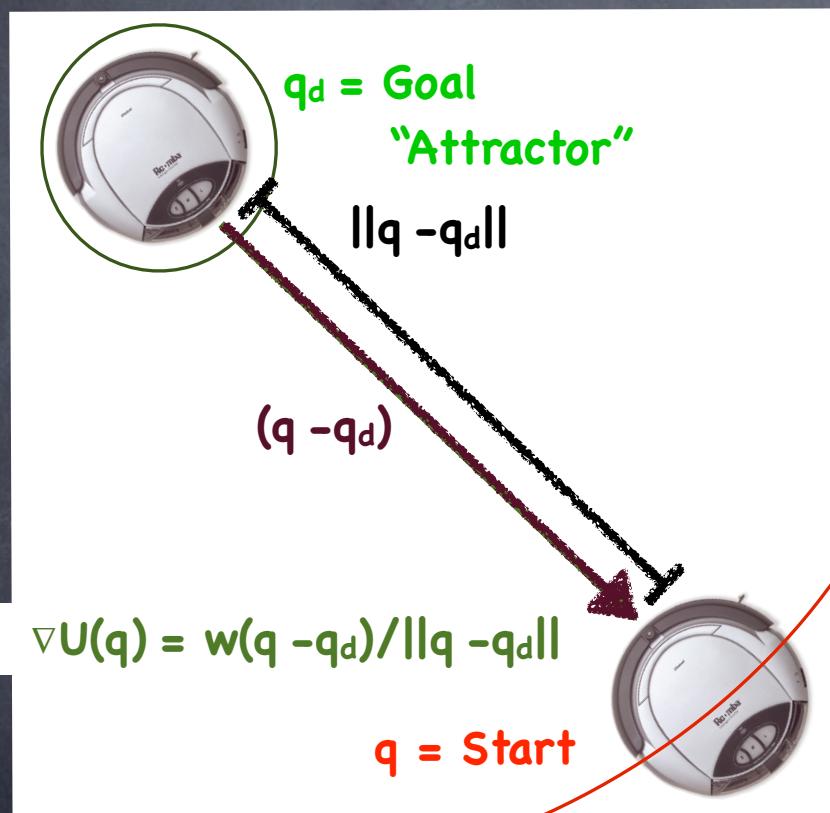
z : height indicates potential at location



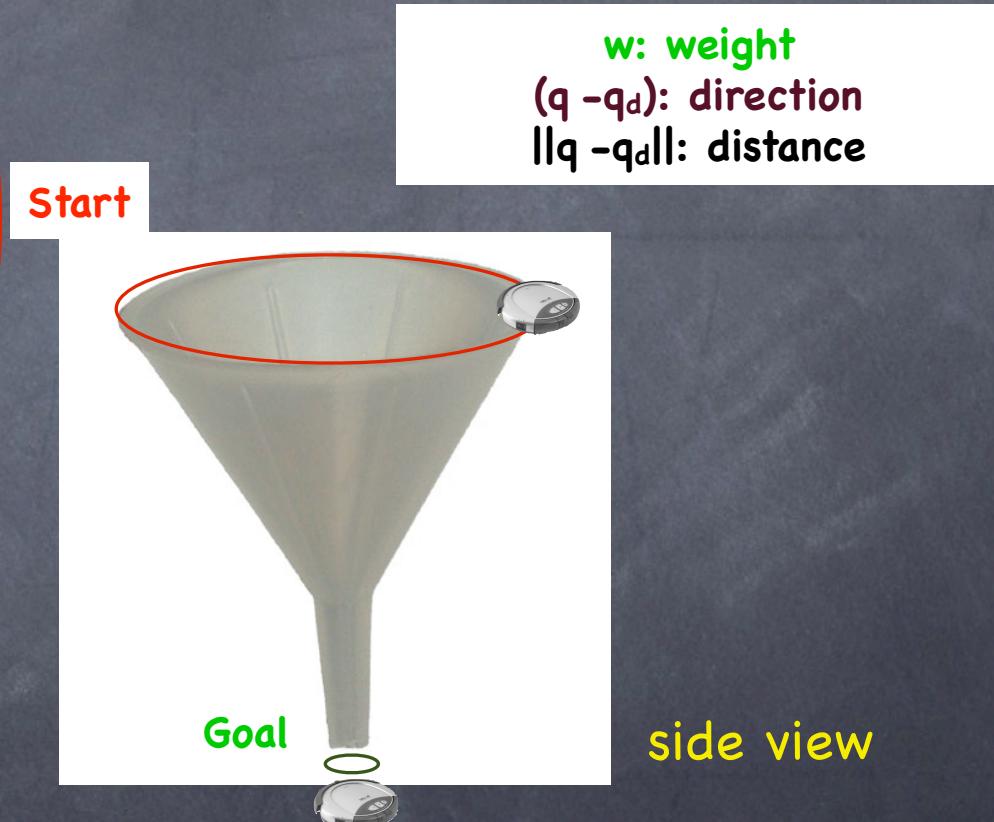
“Cone” Attractor



“Cone” Attractor

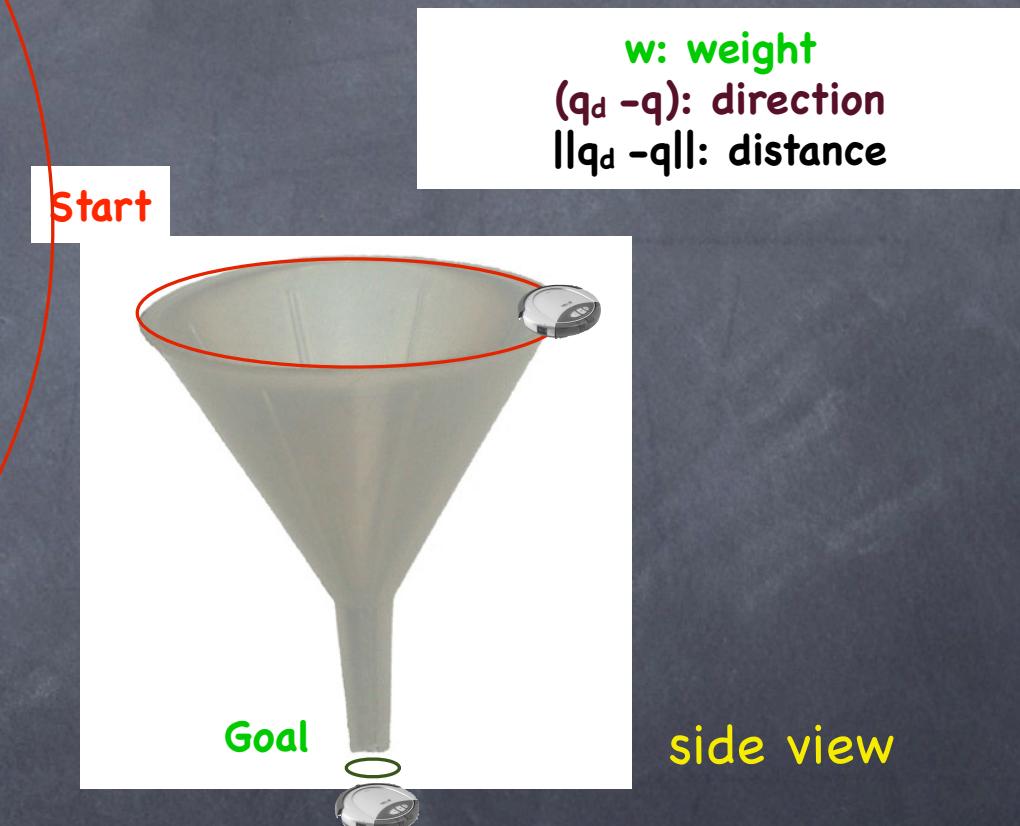
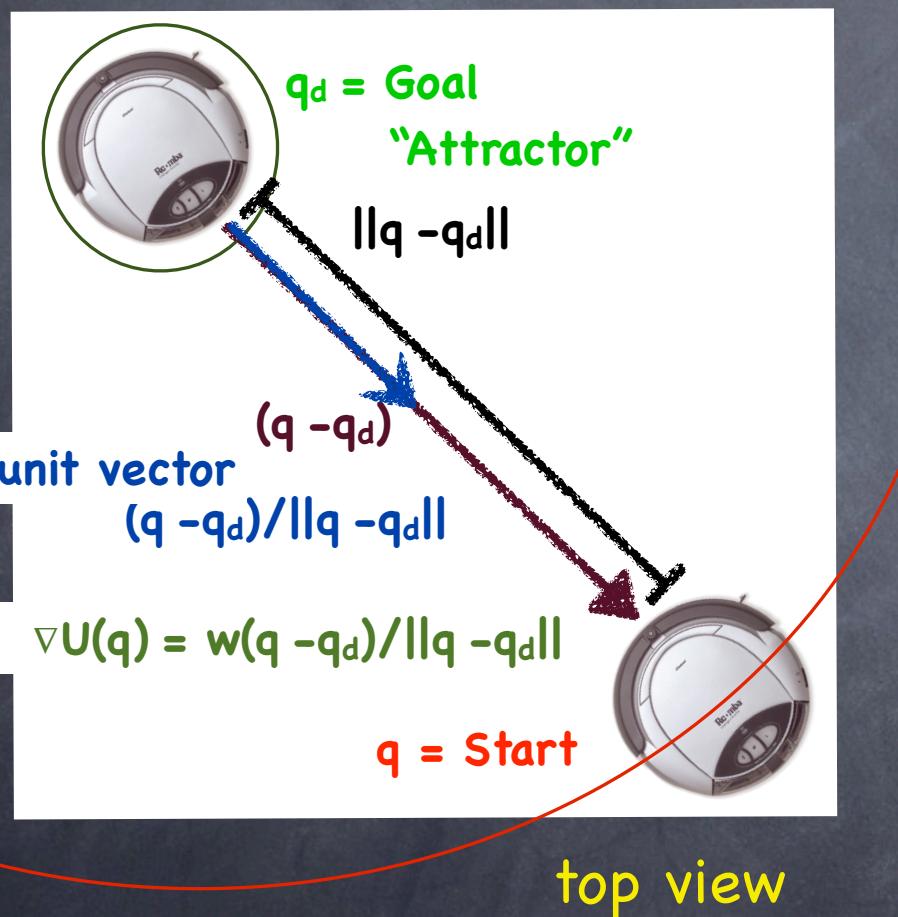


top view

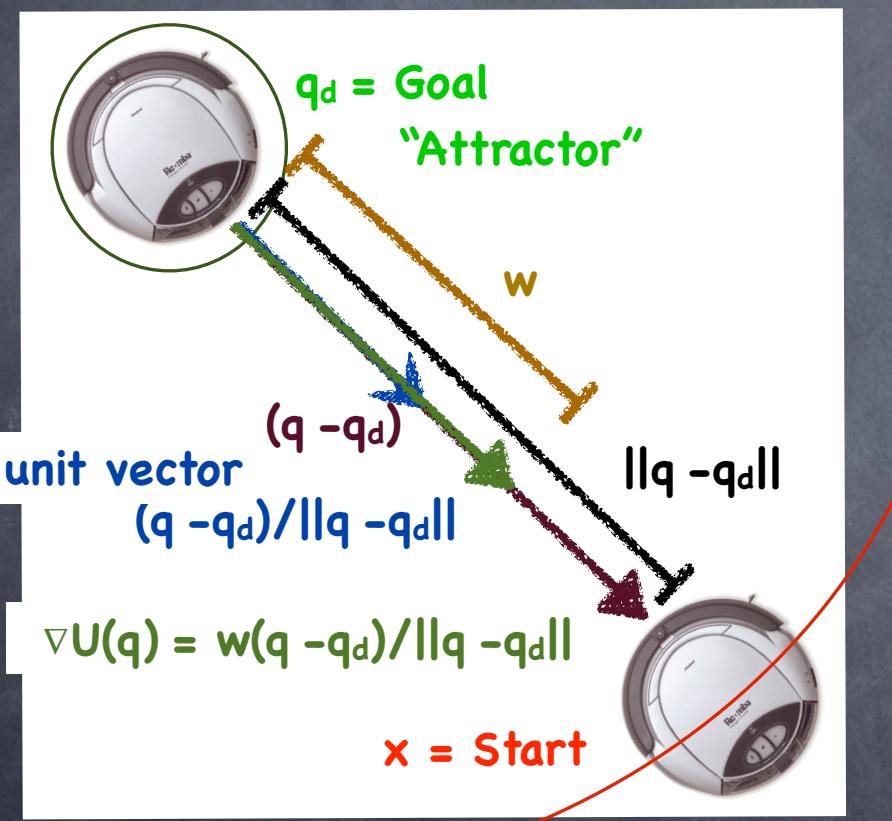


side view

“Cone” Attractor

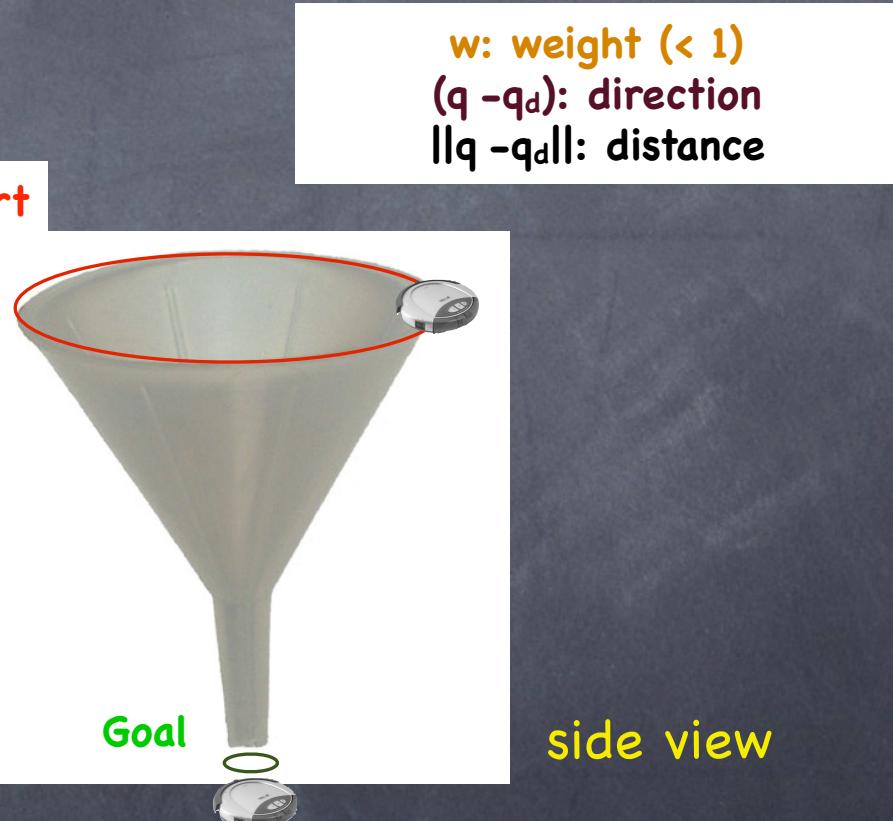


“Cone” Attractor



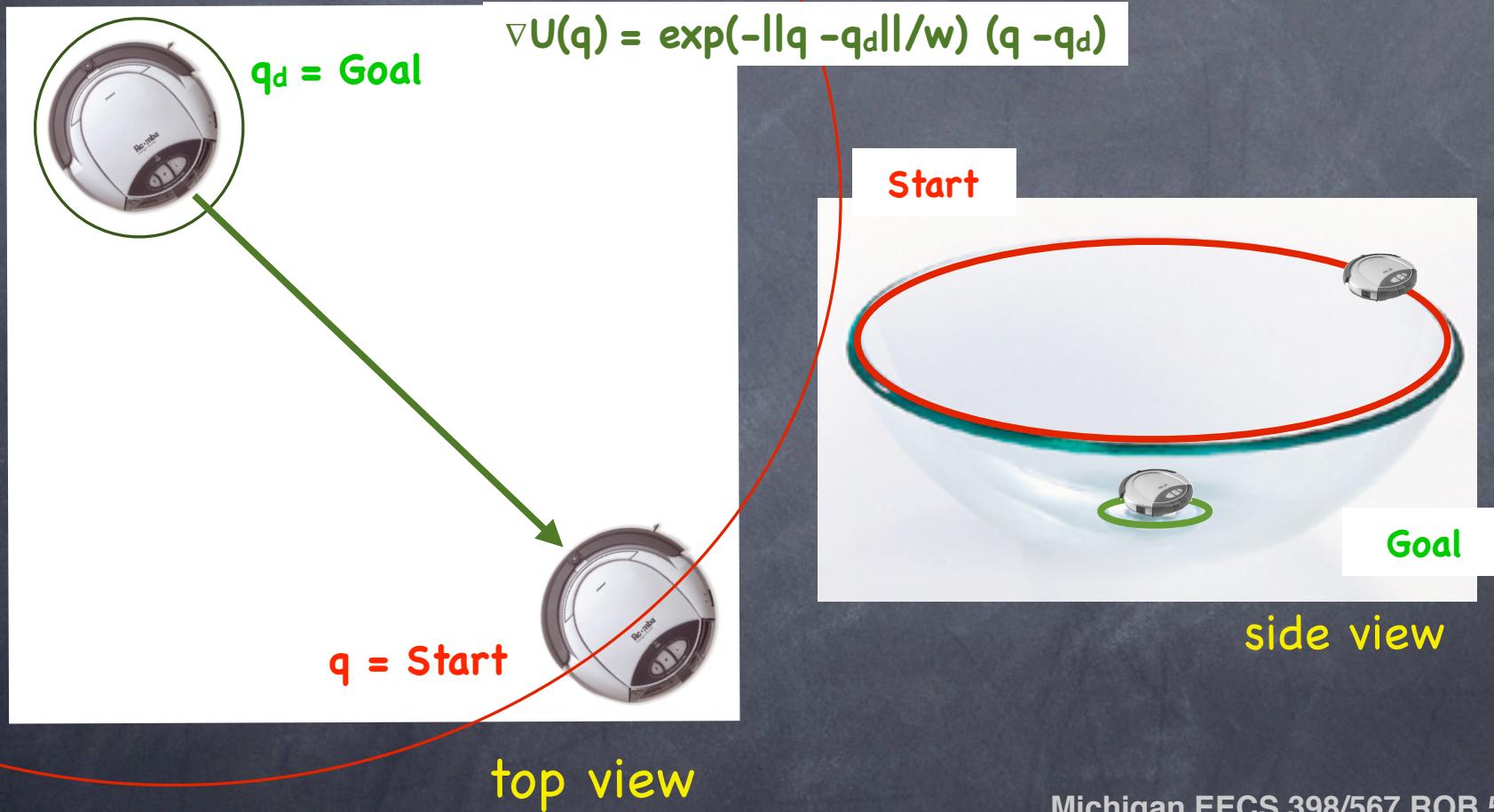
top view

Start



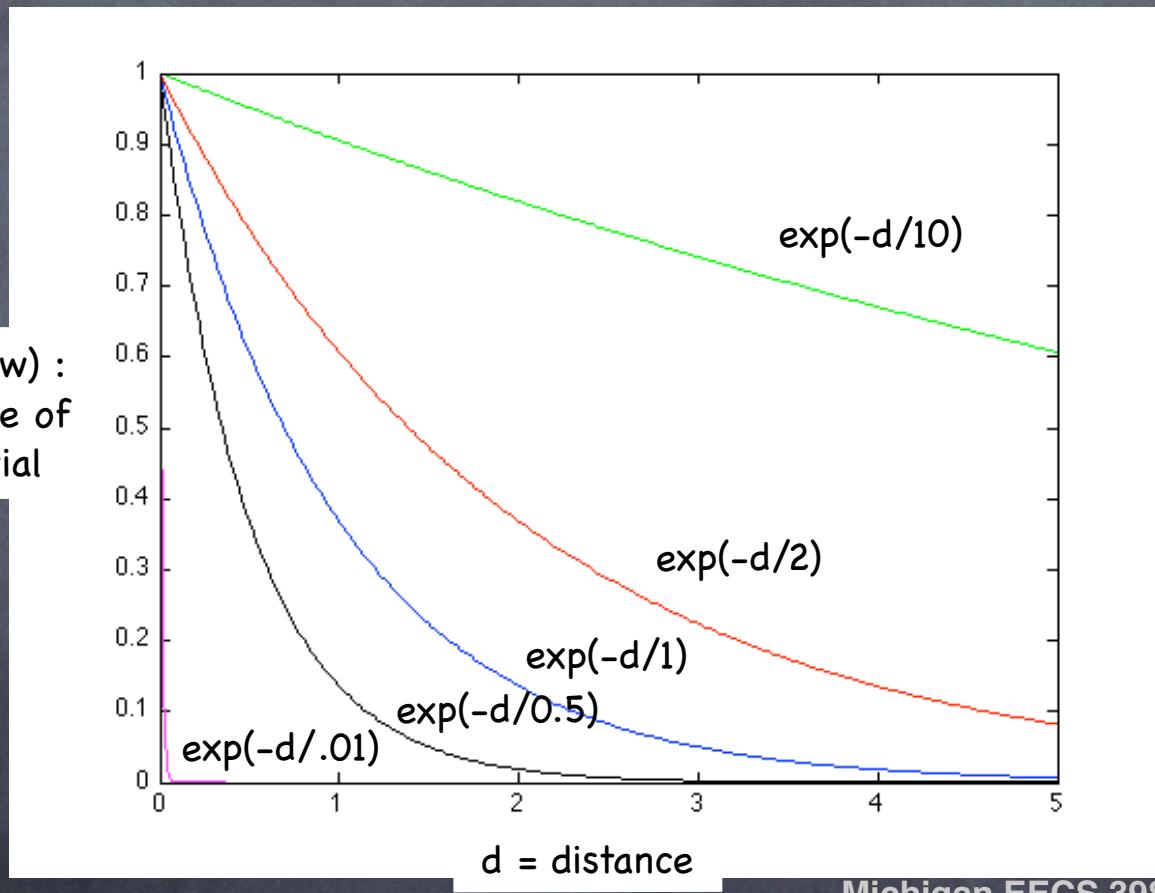
side view

“Bowl” Attractor

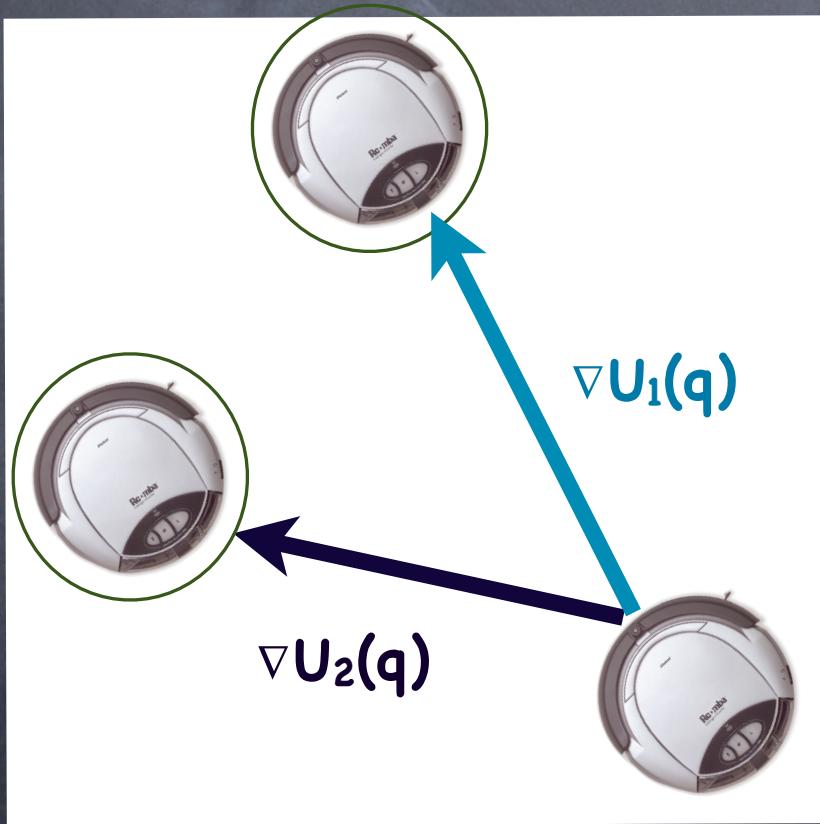


$\exp(-d/w)$

$\exp(-d/w)$:
influence of
potential

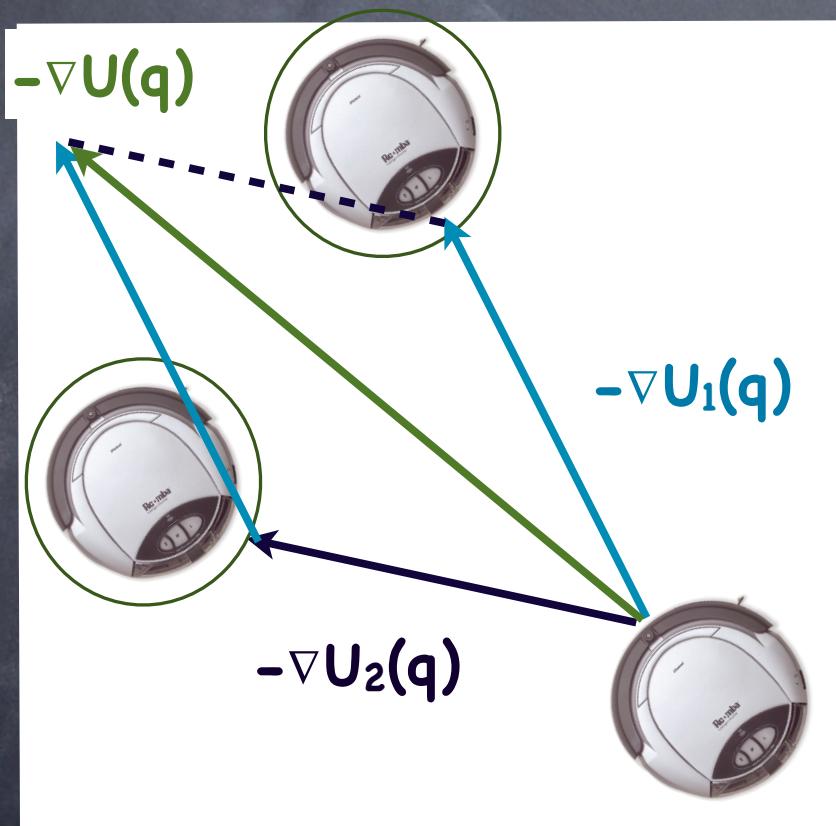


Multiple potentials



- Output of potential field is a vector
- How to combine or select between multiple potentials?

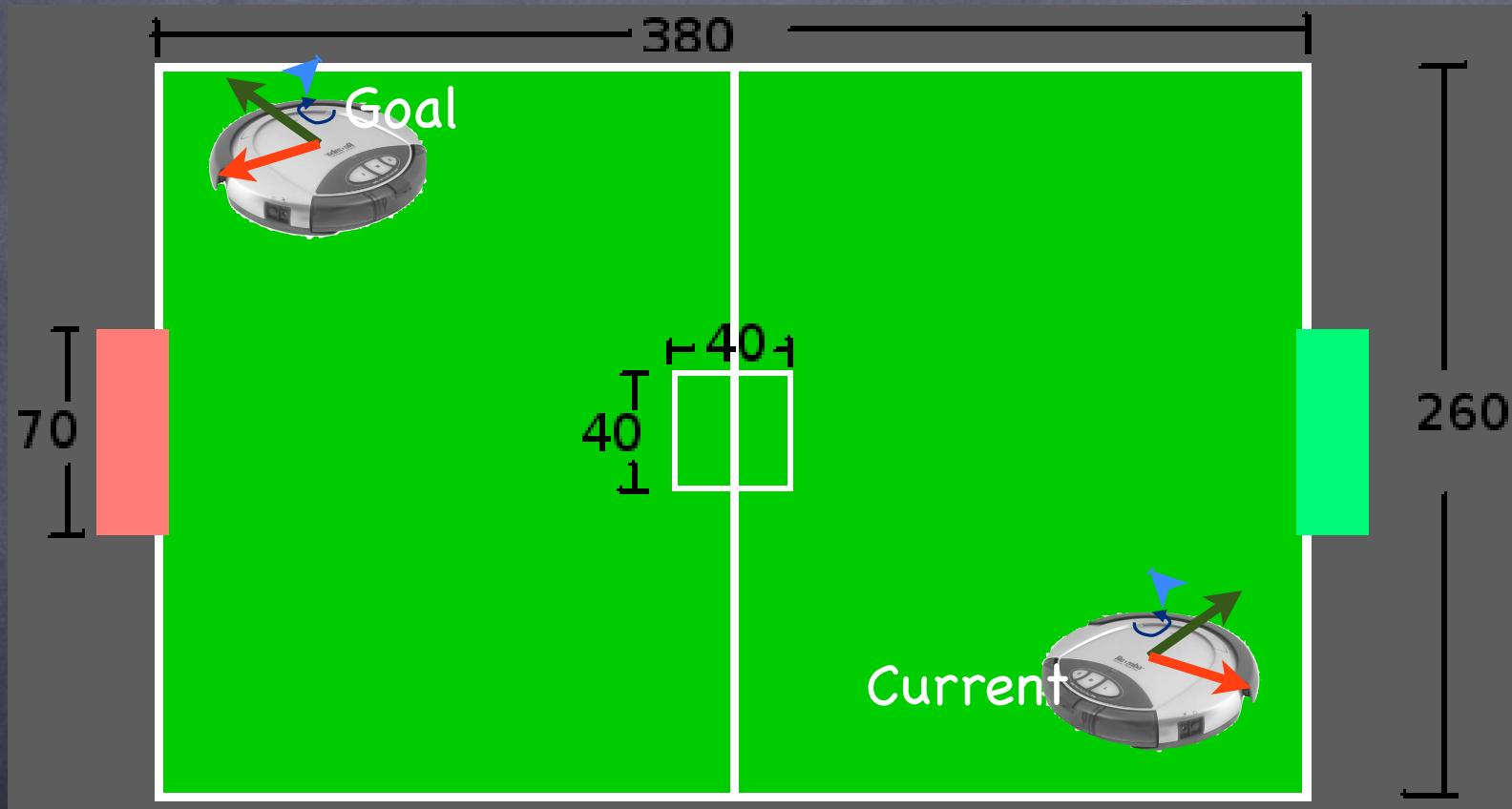
Multiple potentials



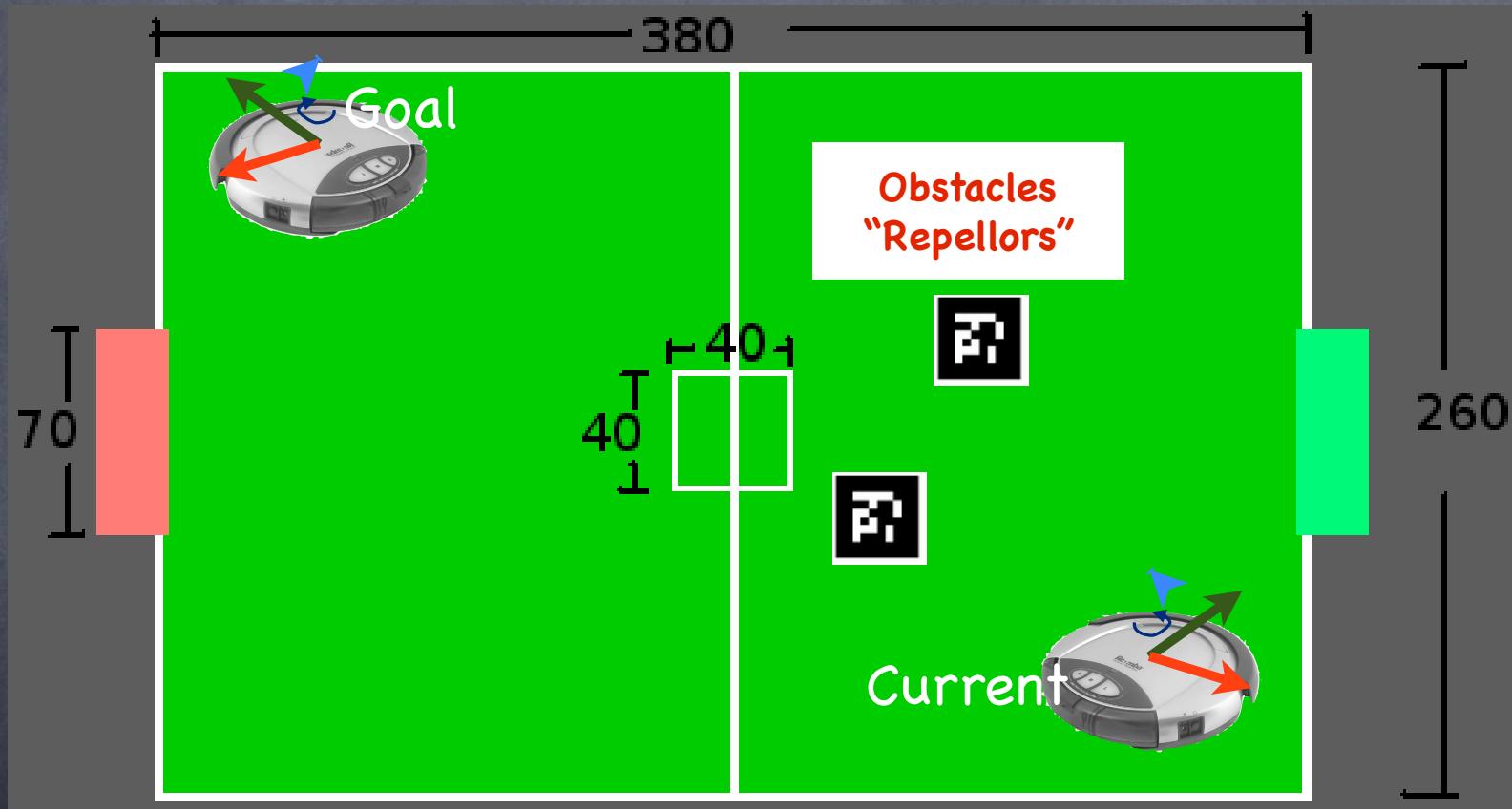
- Output of potential field is a vector
- Combine multiple potentials through vector summation

$$U(q) = \sum_i U_i(q)$$

describe performance for this case

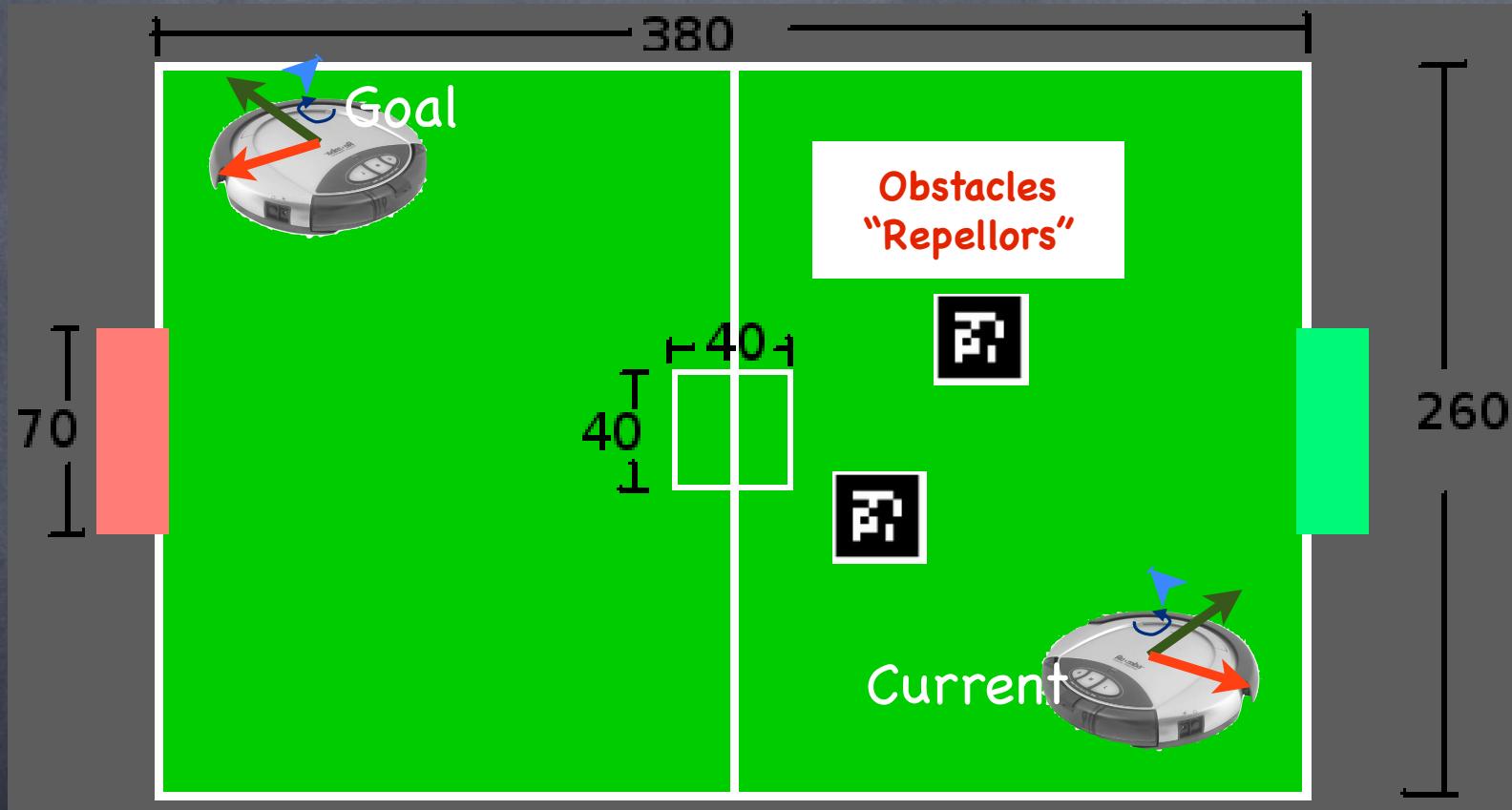


describe performance for this case



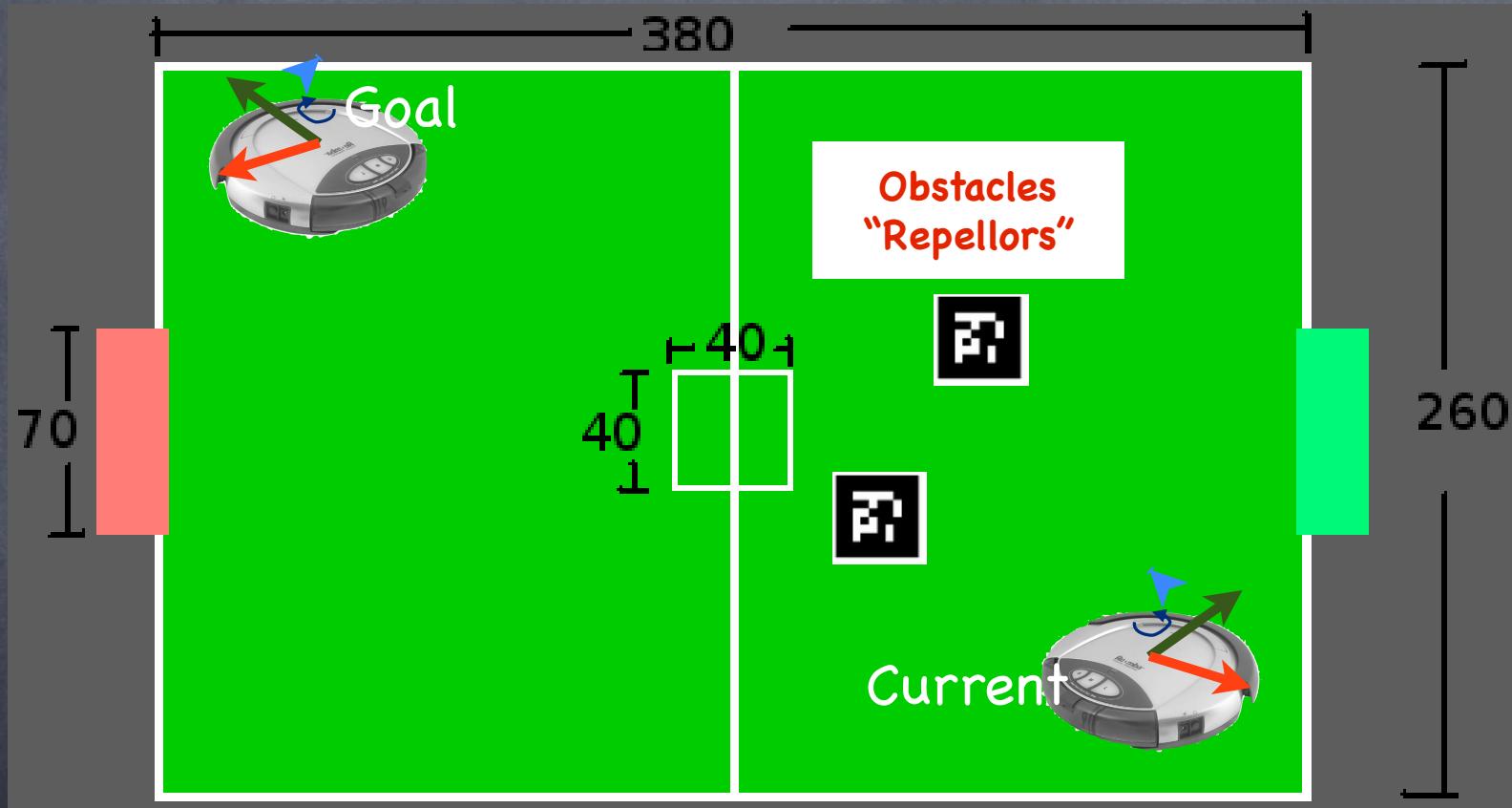
describe performance for this case

how do we deal with repellors?



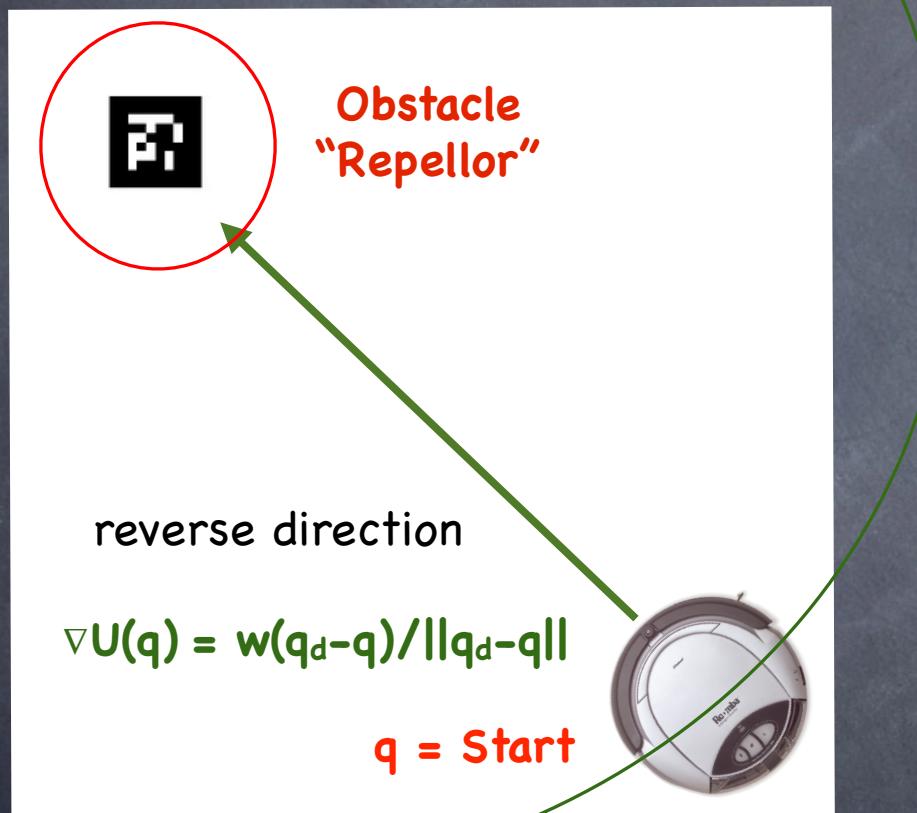
add sum of repulsive potentials

$$U(q) = U_{\text{attracts}}(q) + U_{\text{repellors}}(q)$$



“Cone” Repellor

potential problems?

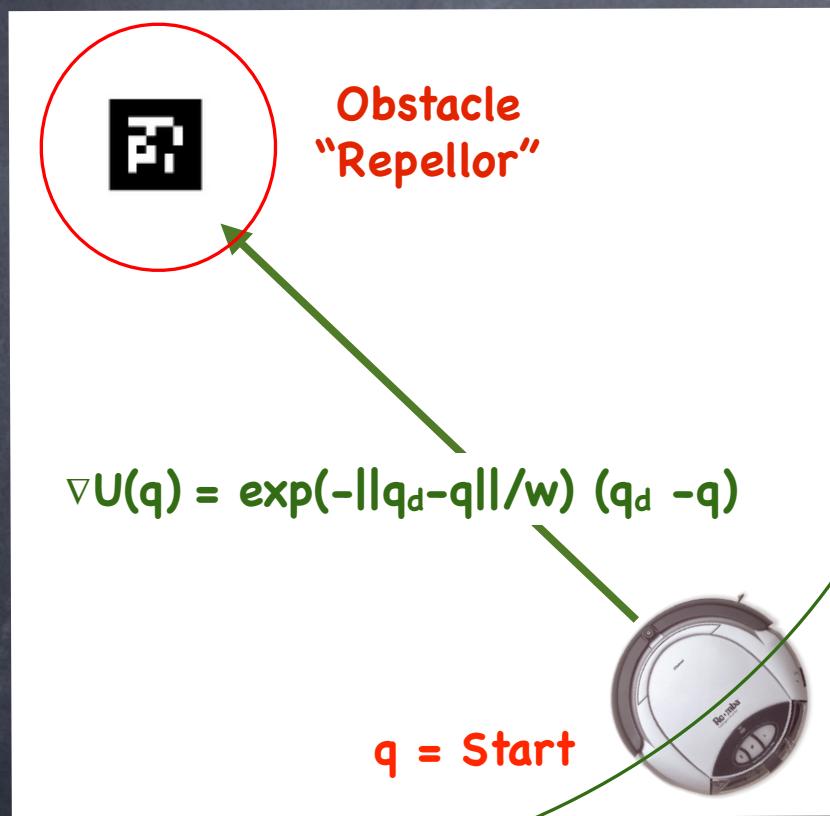


top view

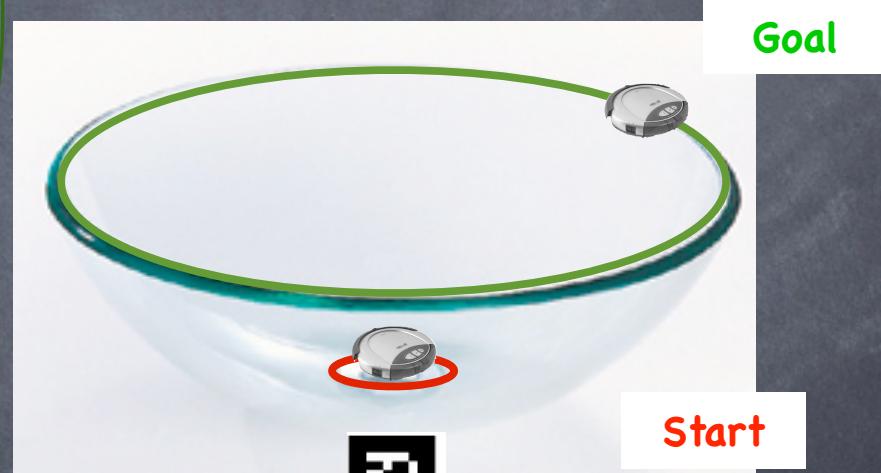


side view

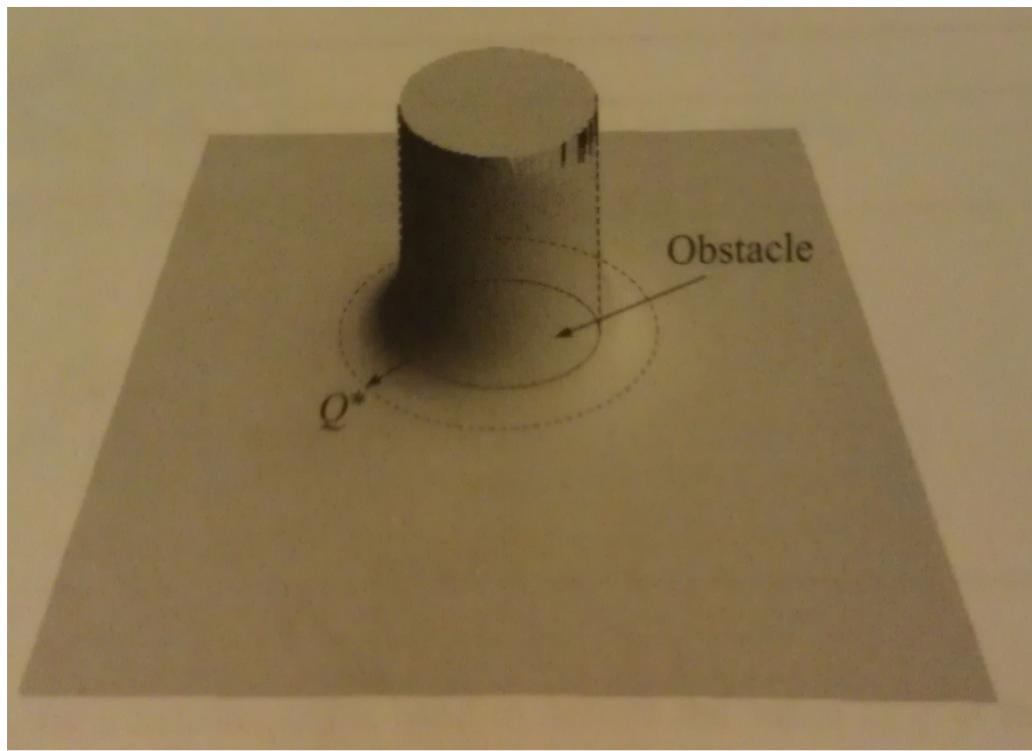
“Bowl” Repellor



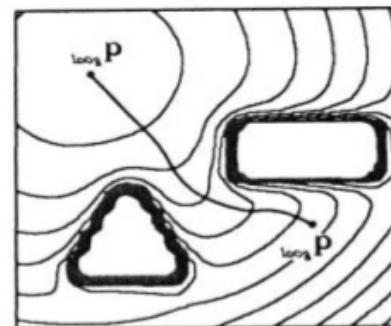
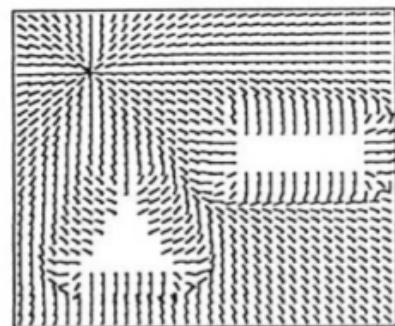
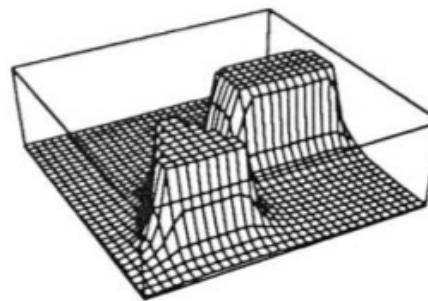
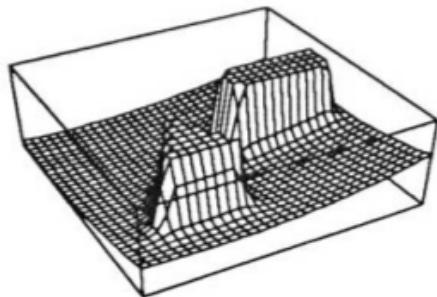
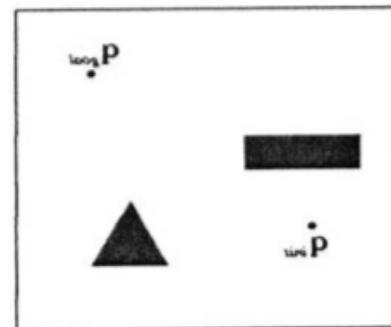
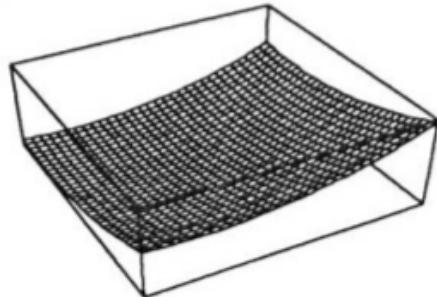
top view



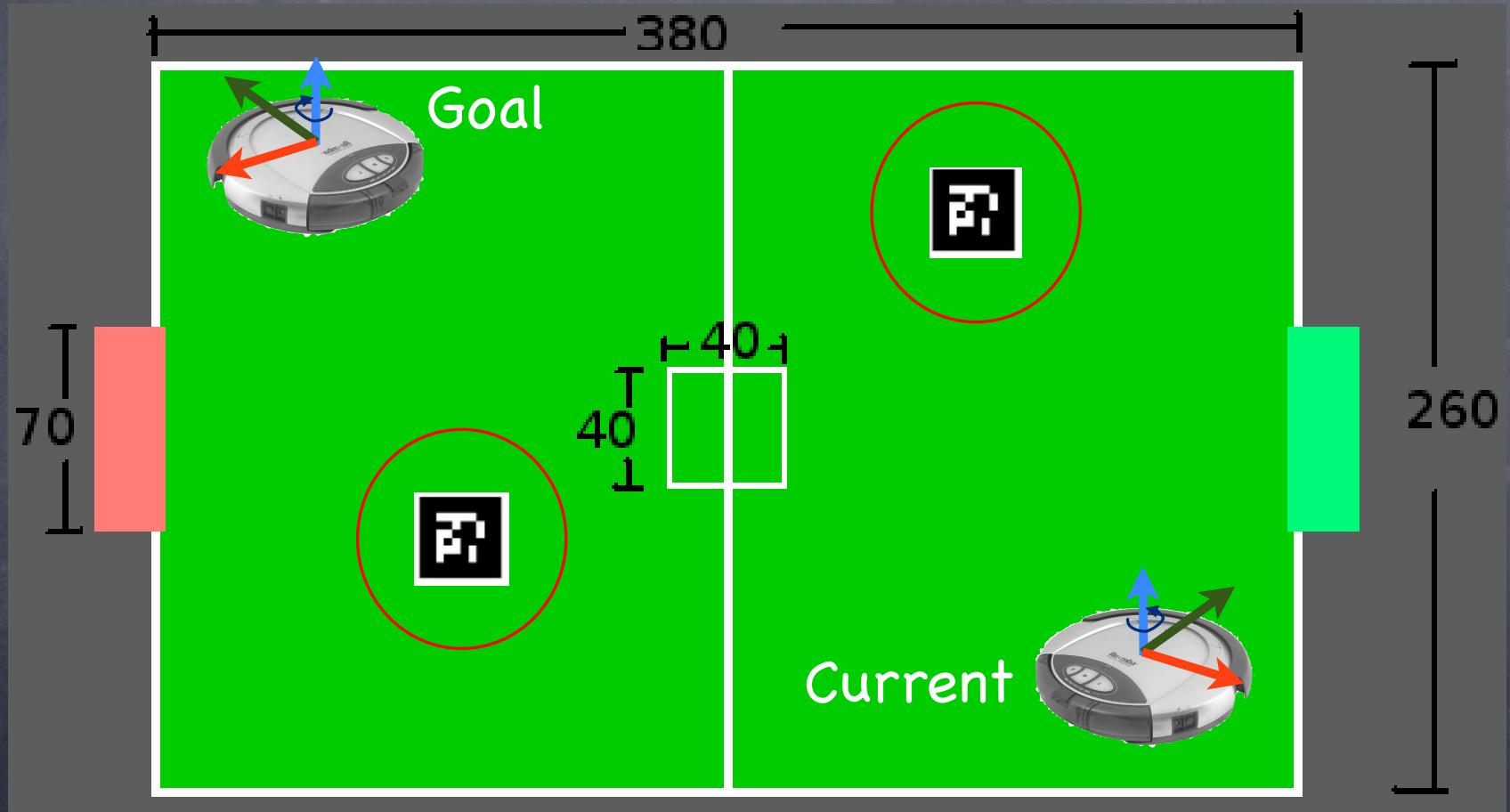
repellor should only have local influence,
repelling only around boundary improves path



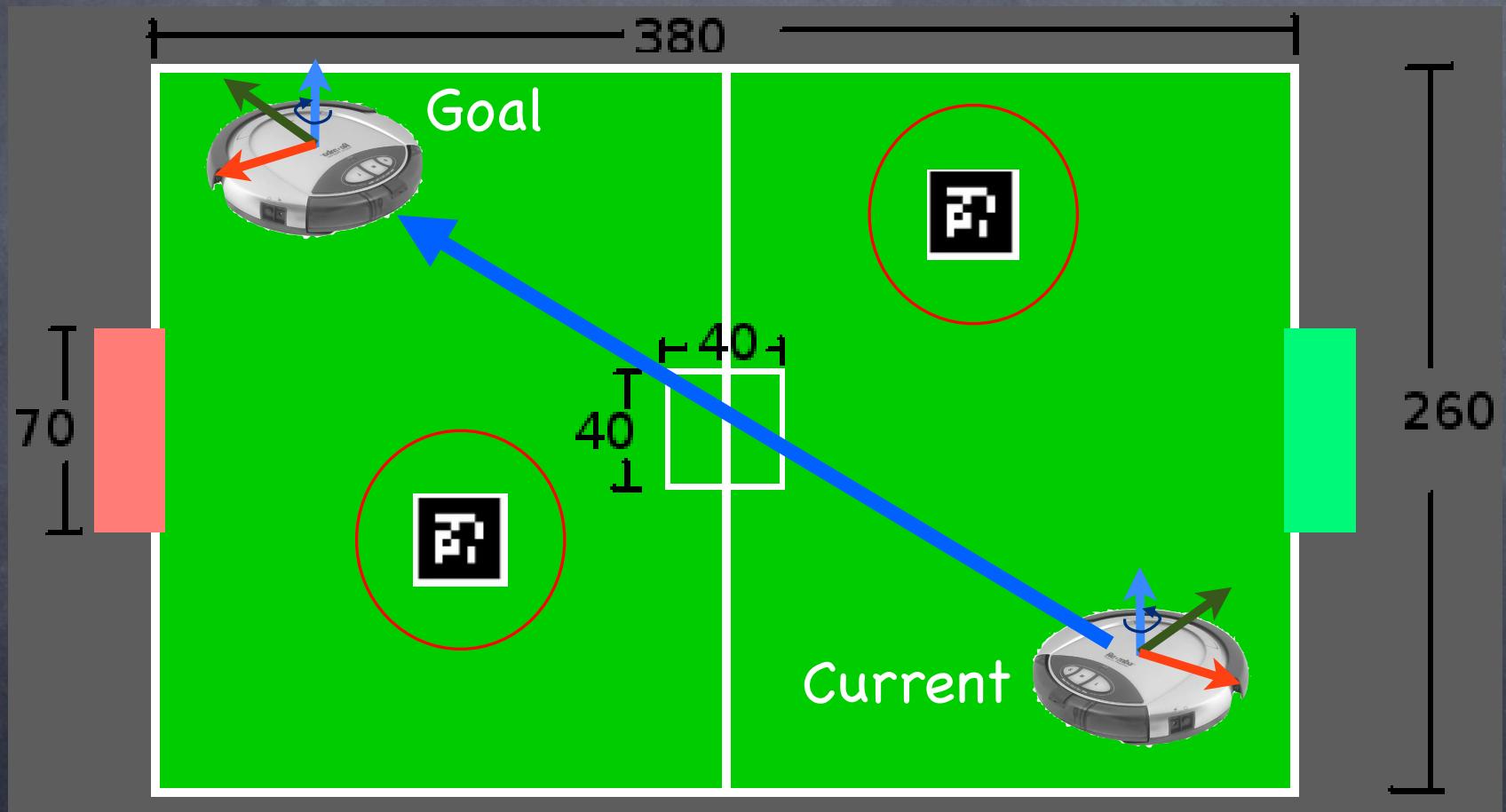
2 Obstacle example



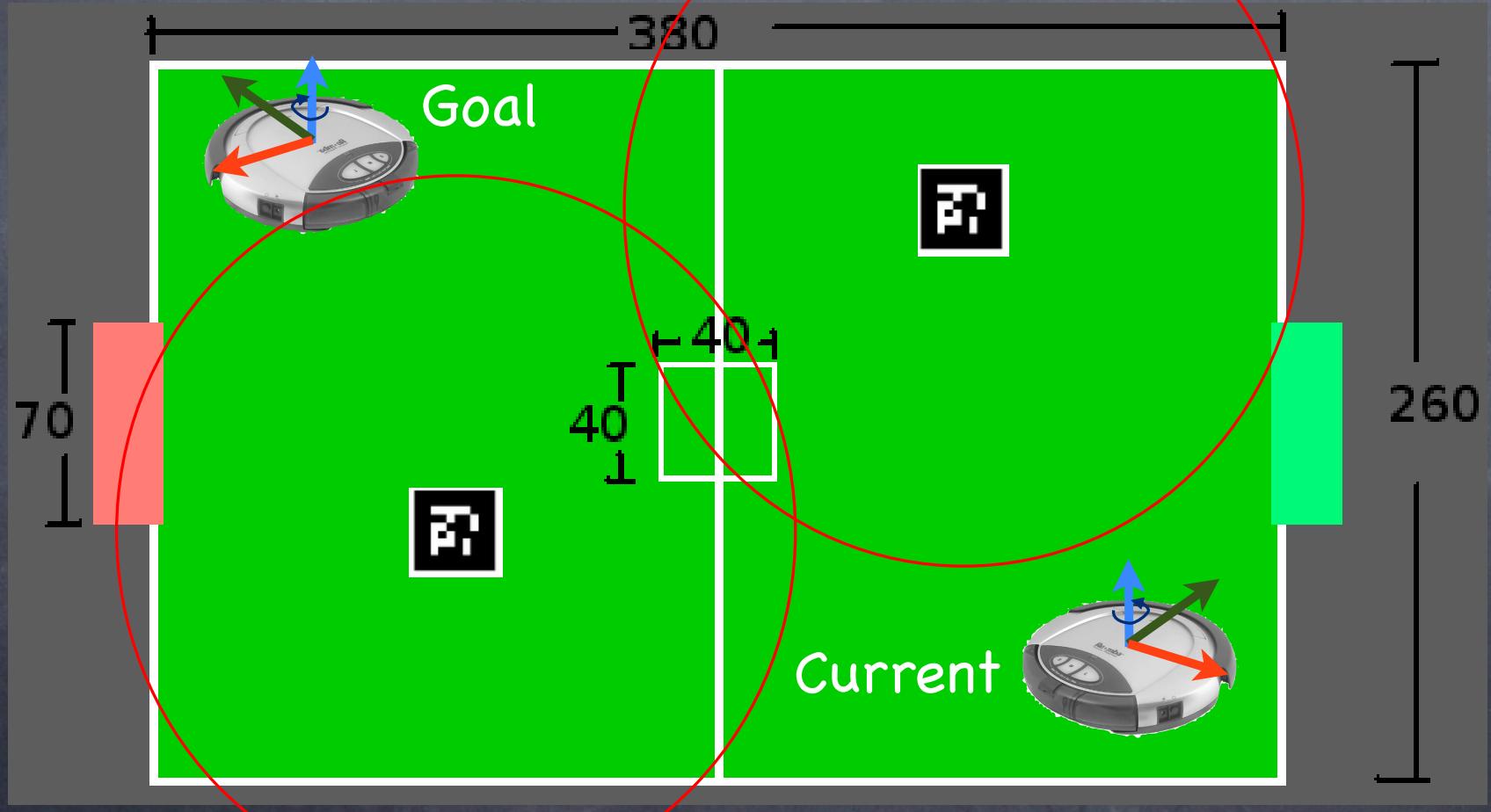
describe performance for this case
with cone attractor to goal and bowl repellors
with limited weight



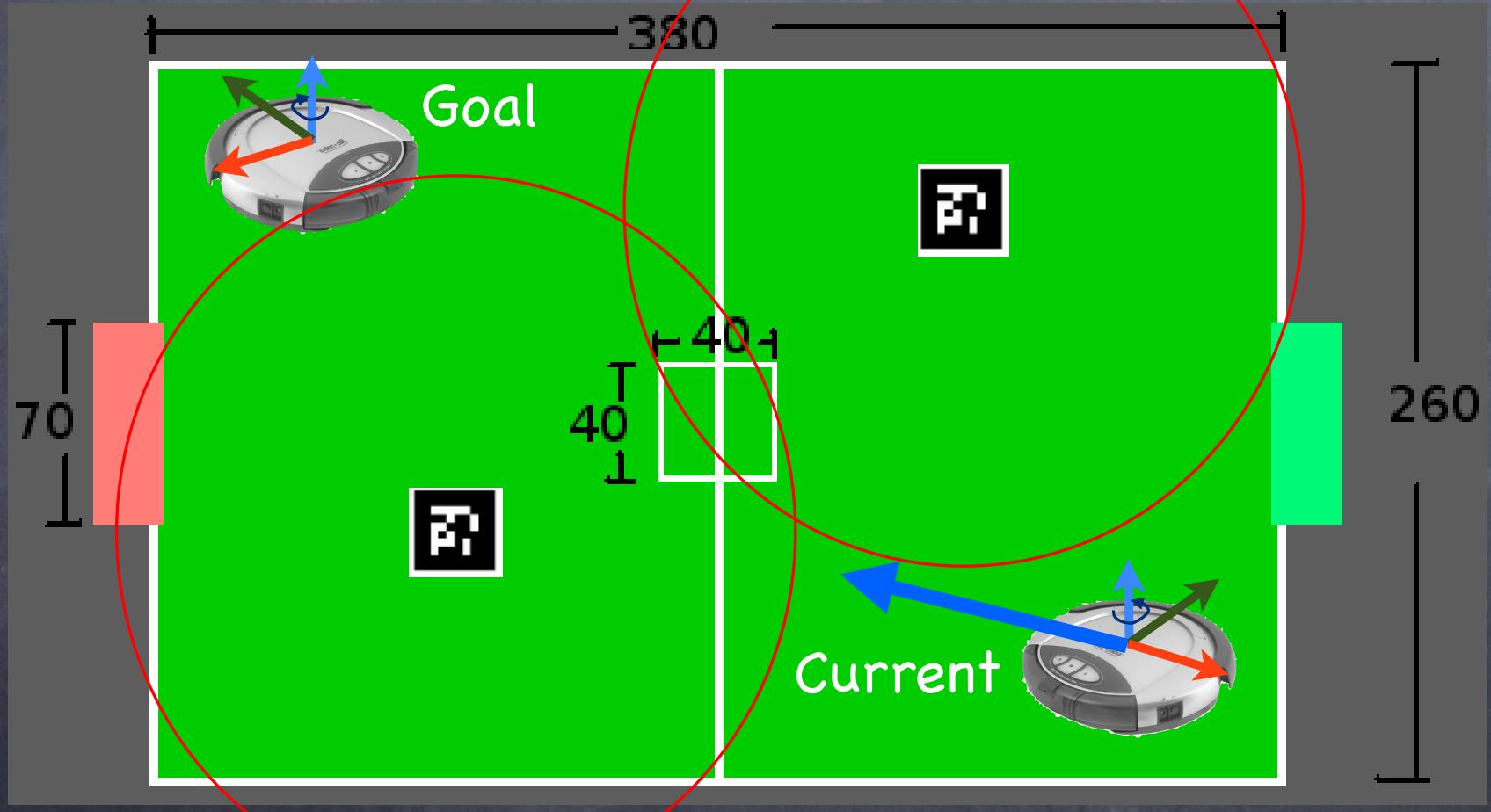
describe performance for this case
with cone attractor to goal and bowl repellors
with limited weight



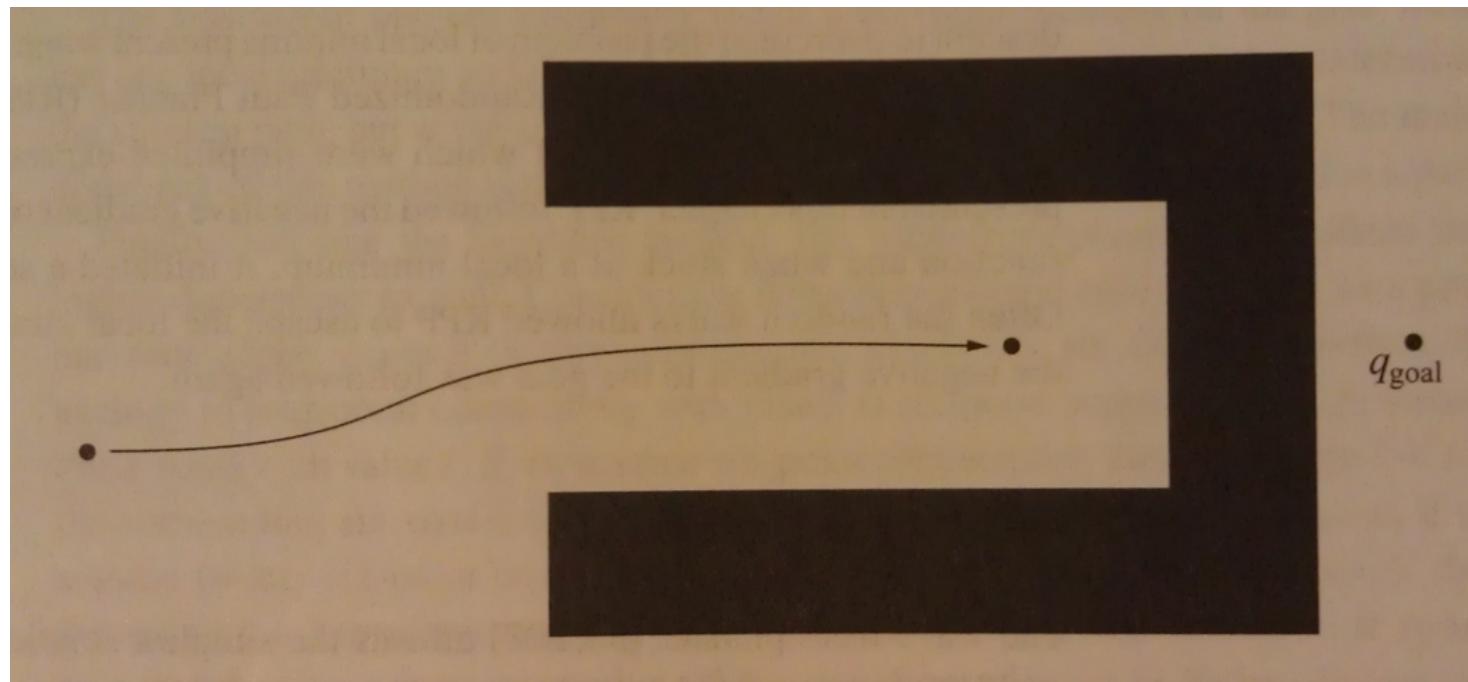
describe performance for this case
with cone attractor to goal and bowl repellors
with limited weight



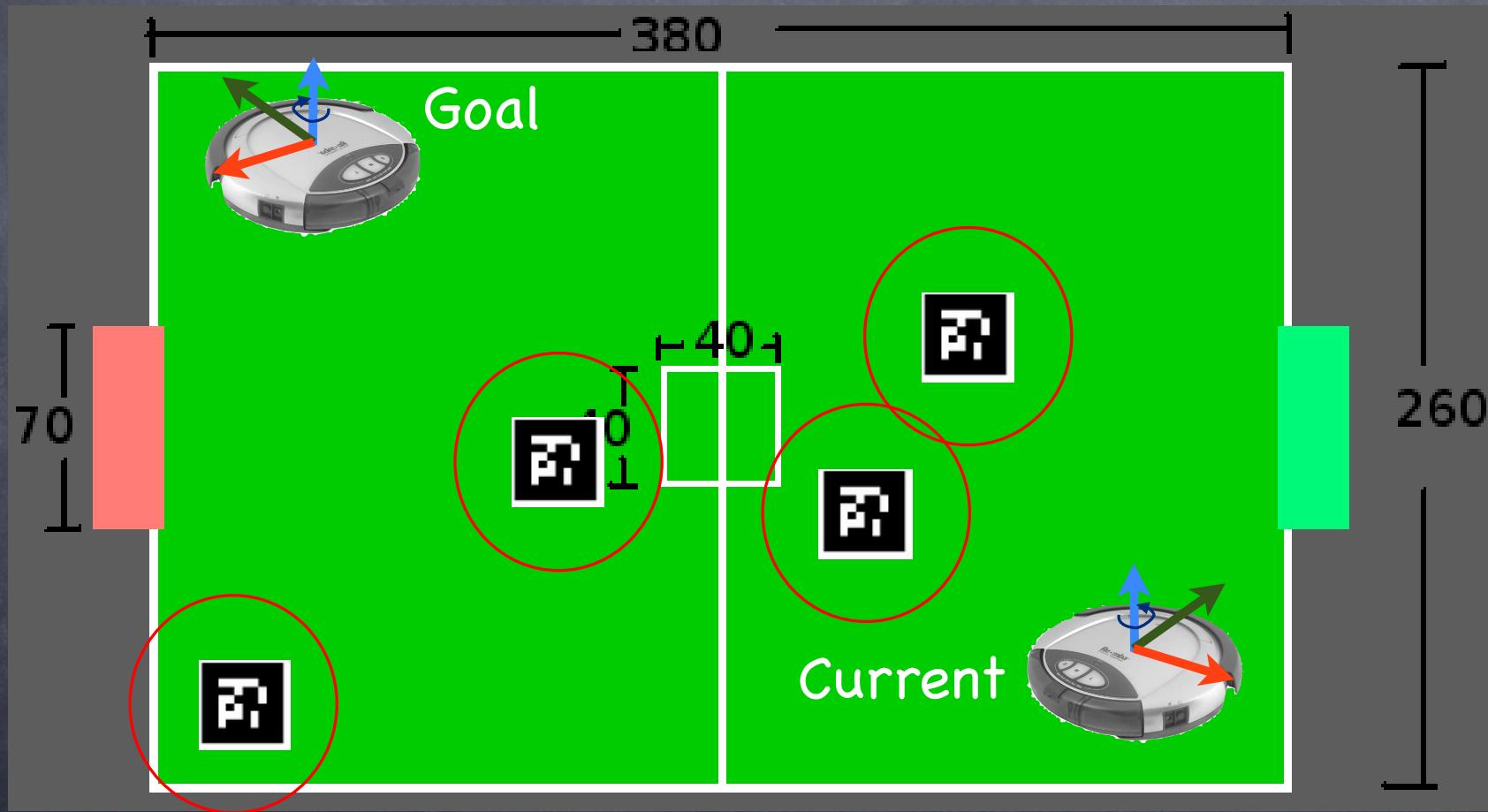
describe performance for this case
with cone attractor to goal and bowl repellors
with limited weight



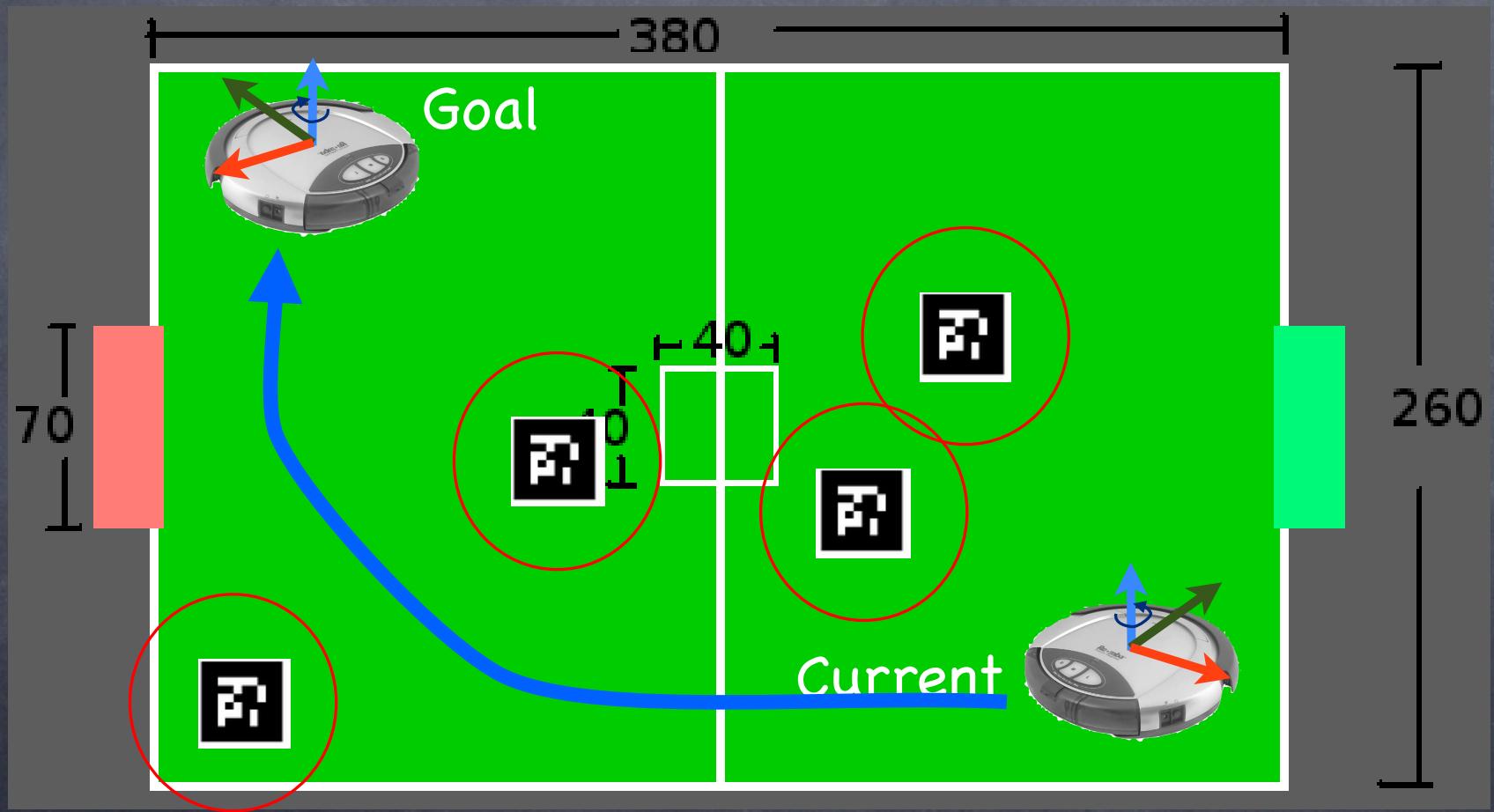
Local Minima



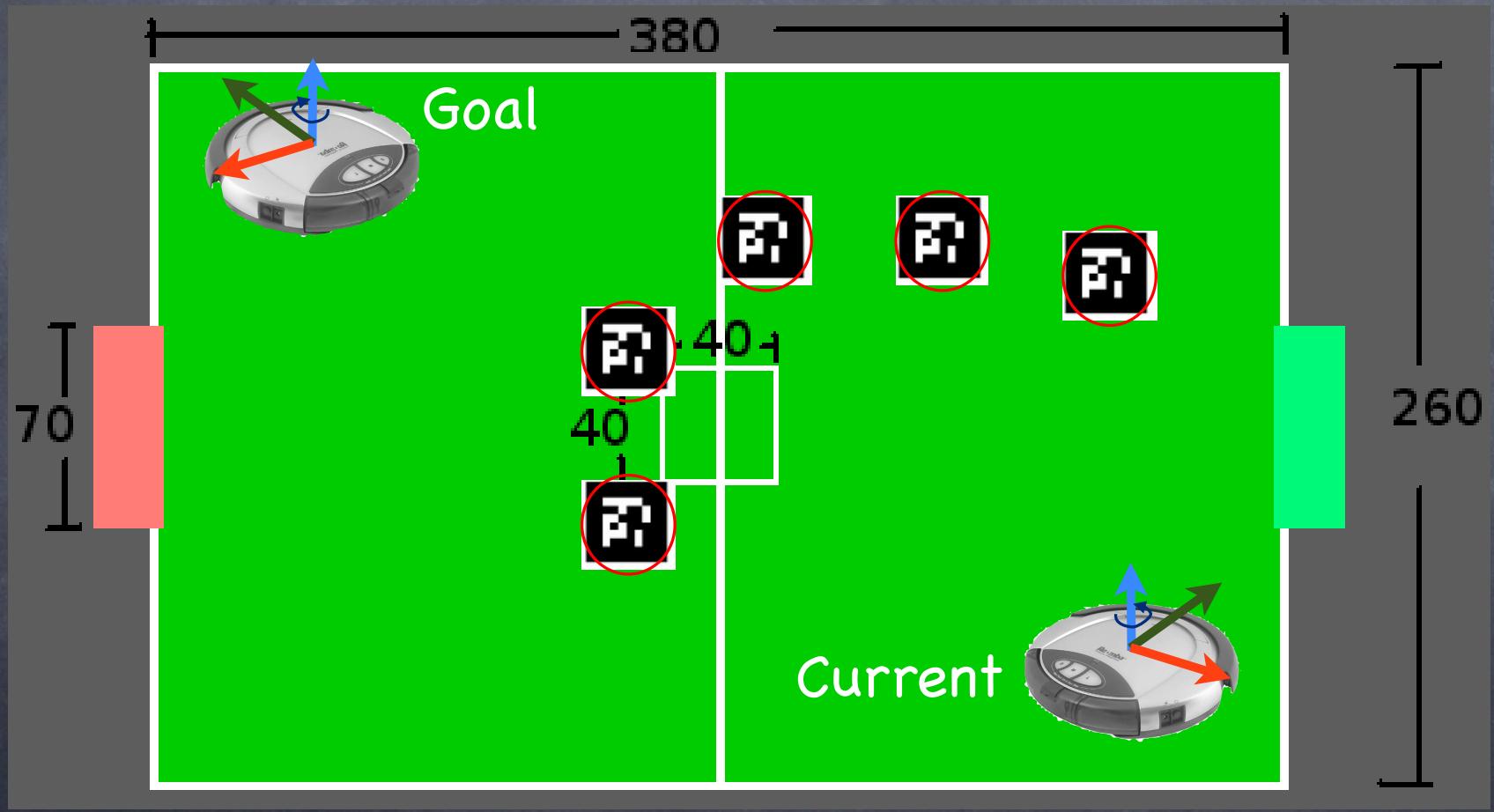
describe performance for this case



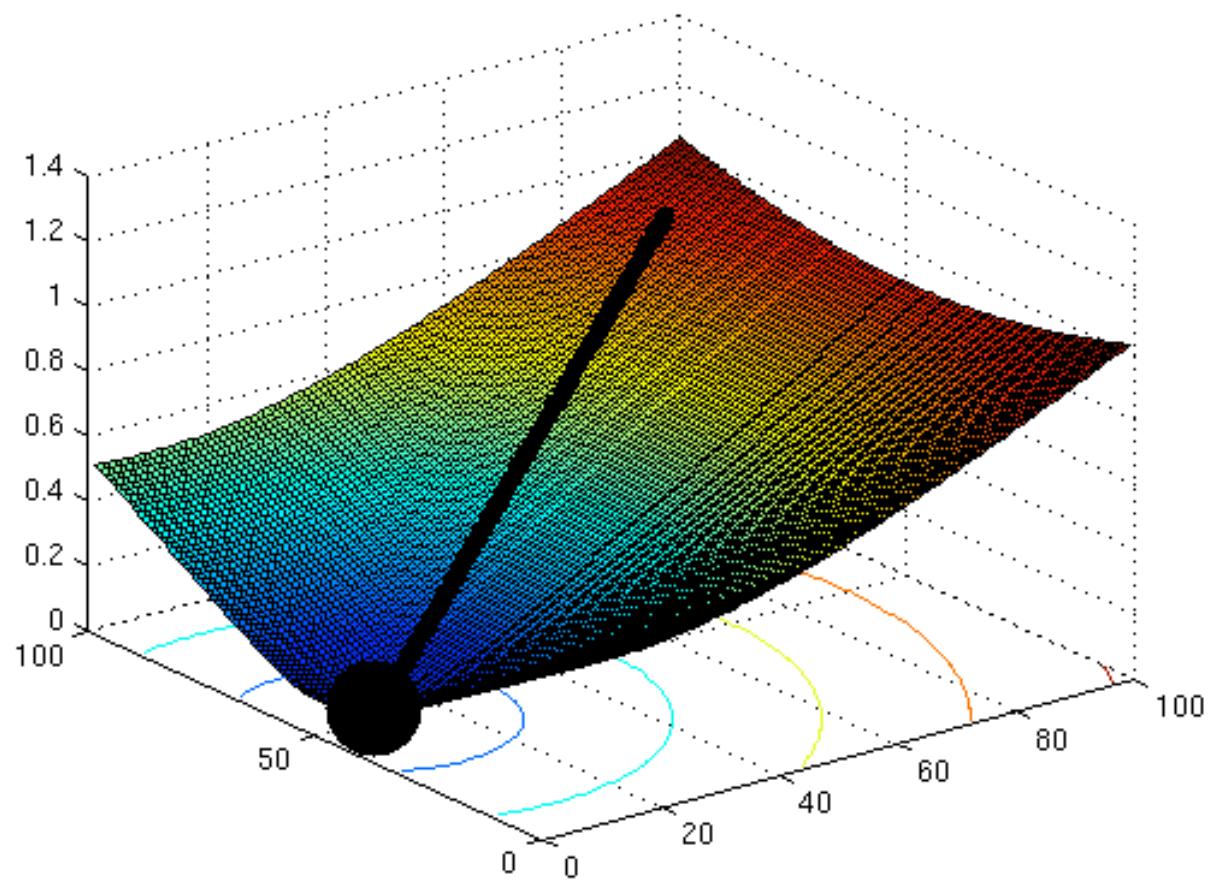
describe performance for this case



describe performance for this case



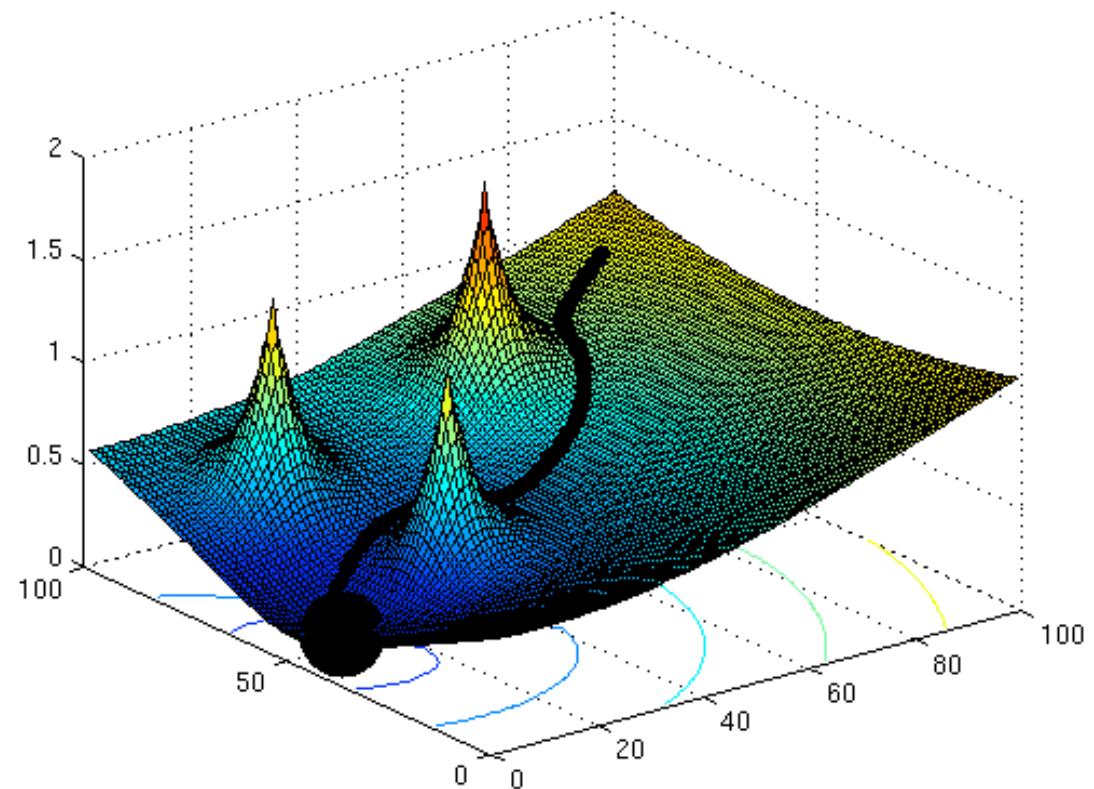
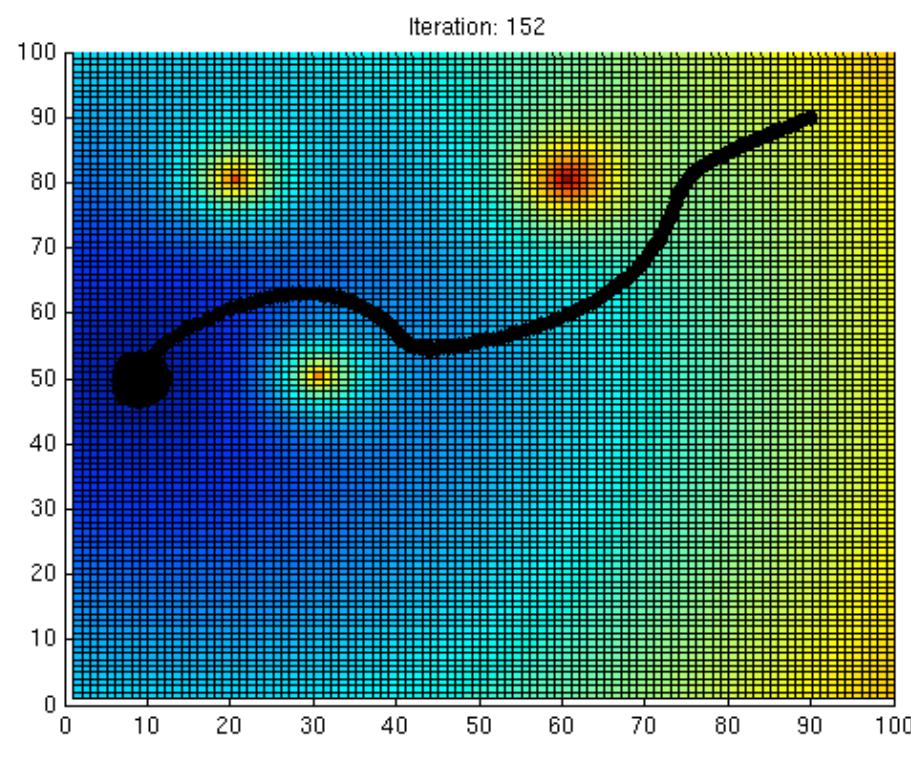
matlab example



pfield.m [1 5 8 12]

Michigan EECS 398/567 ROB 510 - autorob.org

matlab example

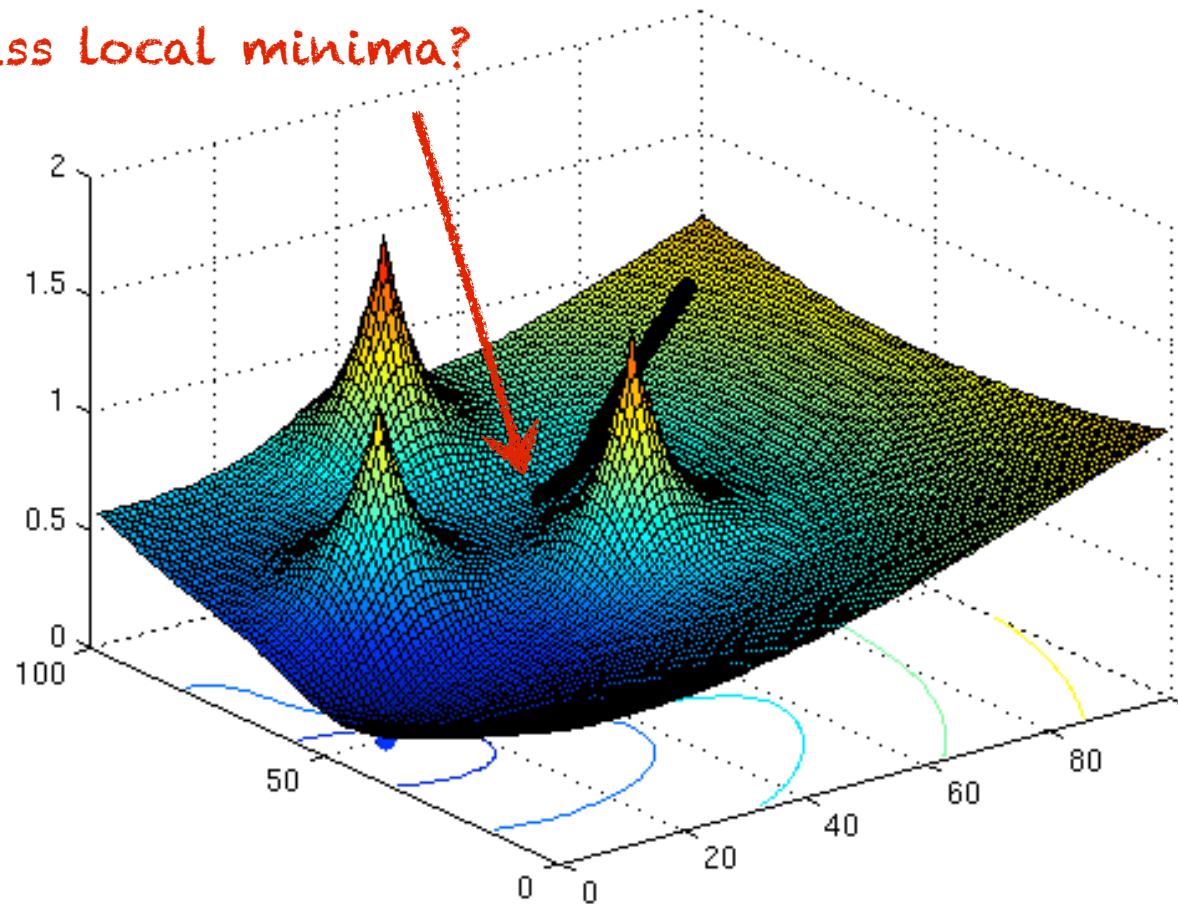


pfield.m [1 5 8 12]

Michigan EECS 398/567 ROB 510 - autorob.org

matlab example

How to address Local minima?



pfield.m [1 5 8 12]

Michigan EECS 398/567 ROB 510 - autorob.org

How can we get out of
local minima?

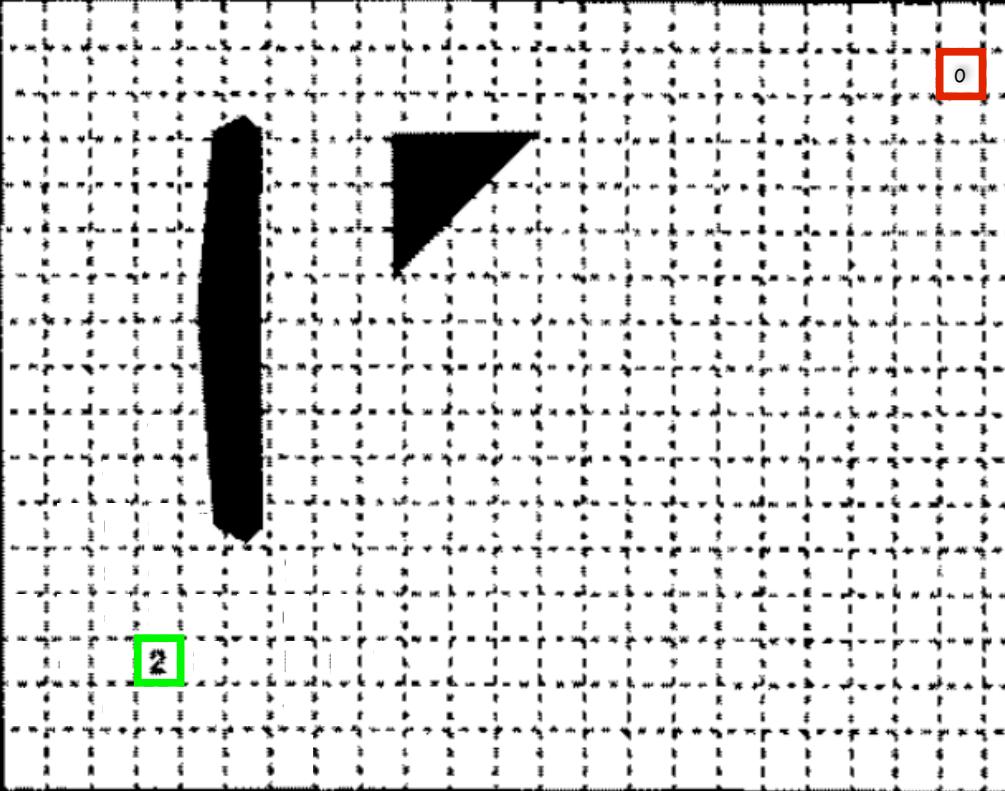
How can we get out of
local minima?

Go back to planning.

Wavefront Planning

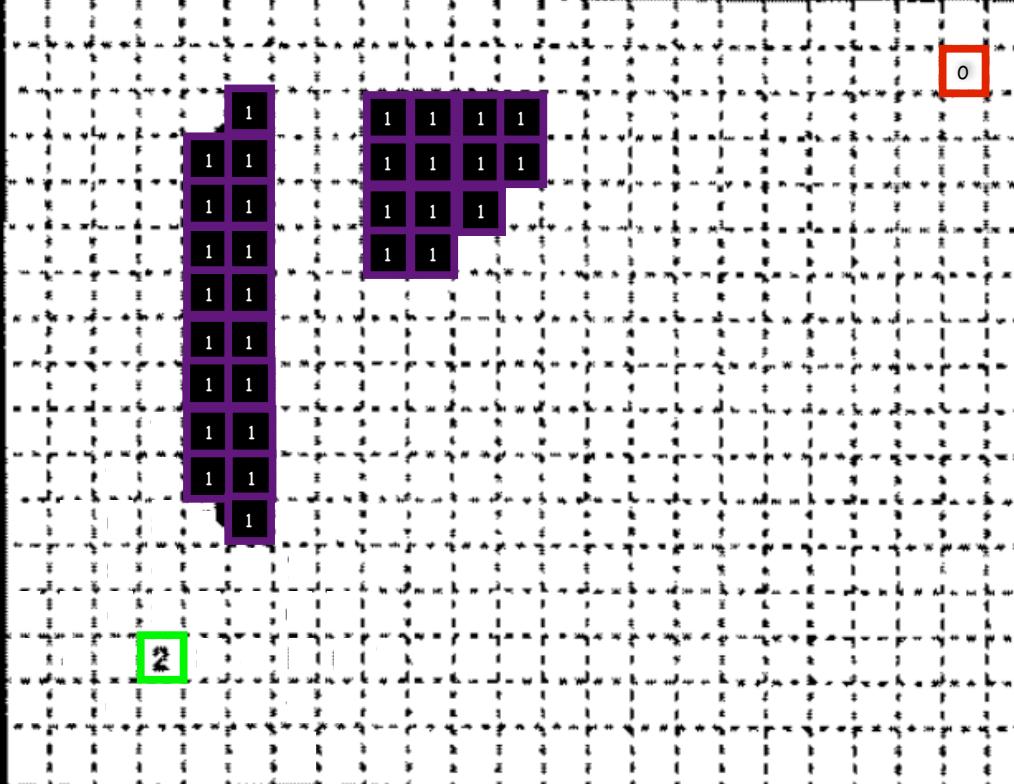
- Discretize potential field into grid
 - Cells store cost to goal with respect to potential field
 - Computed by Brushfire algorithm (essentially BFS)
- Grid search to find navigation path to goal

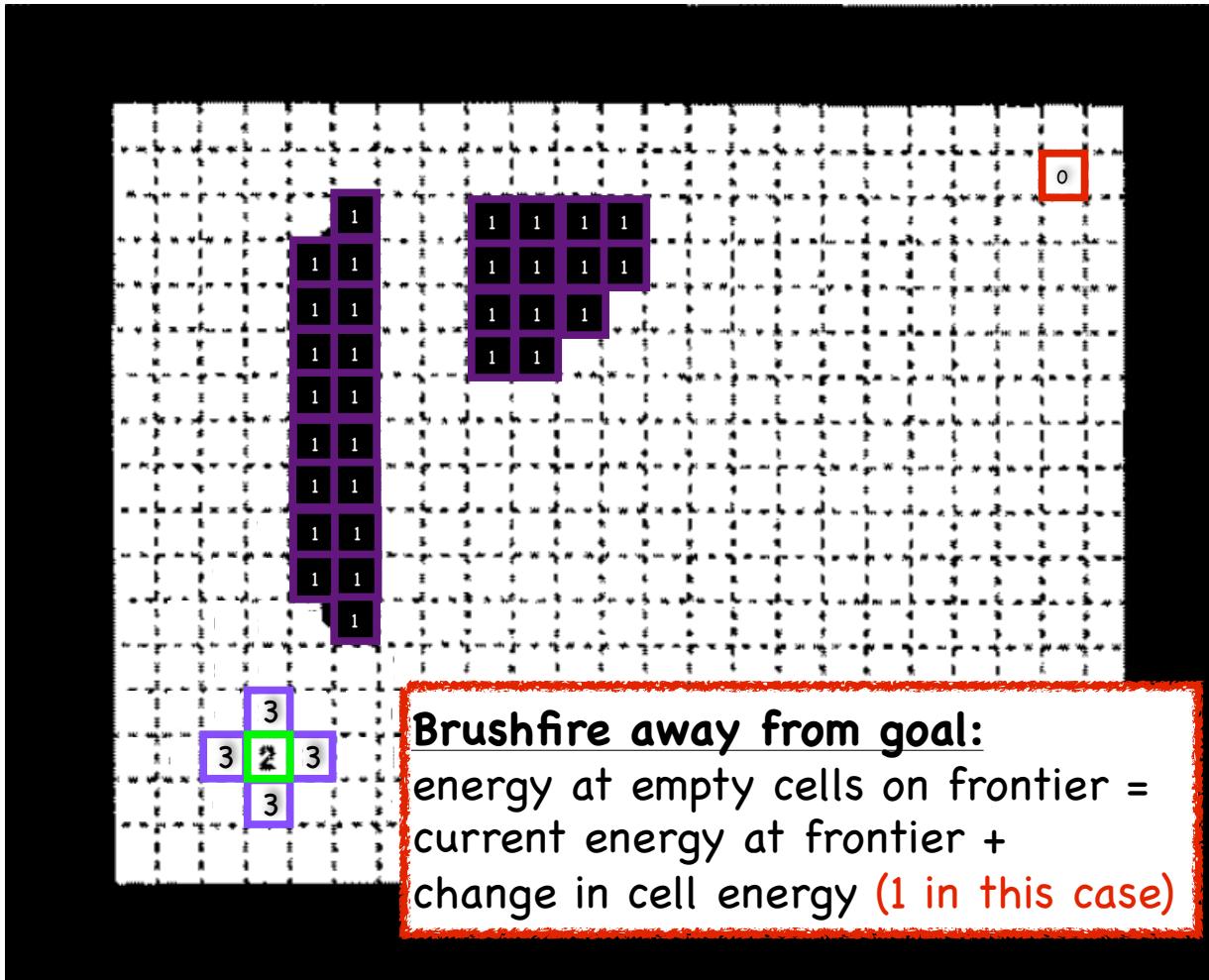
Start: mark with 0

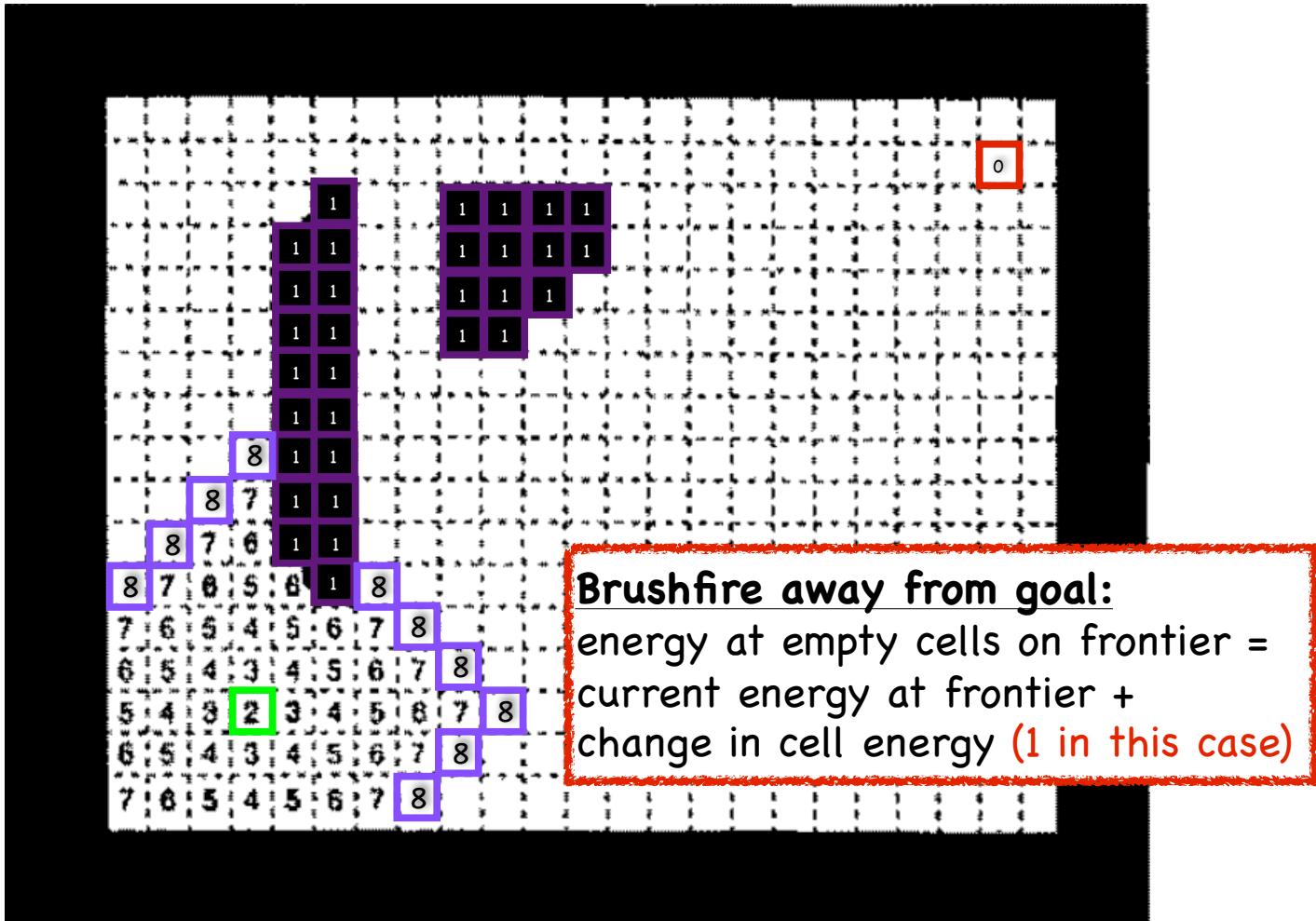


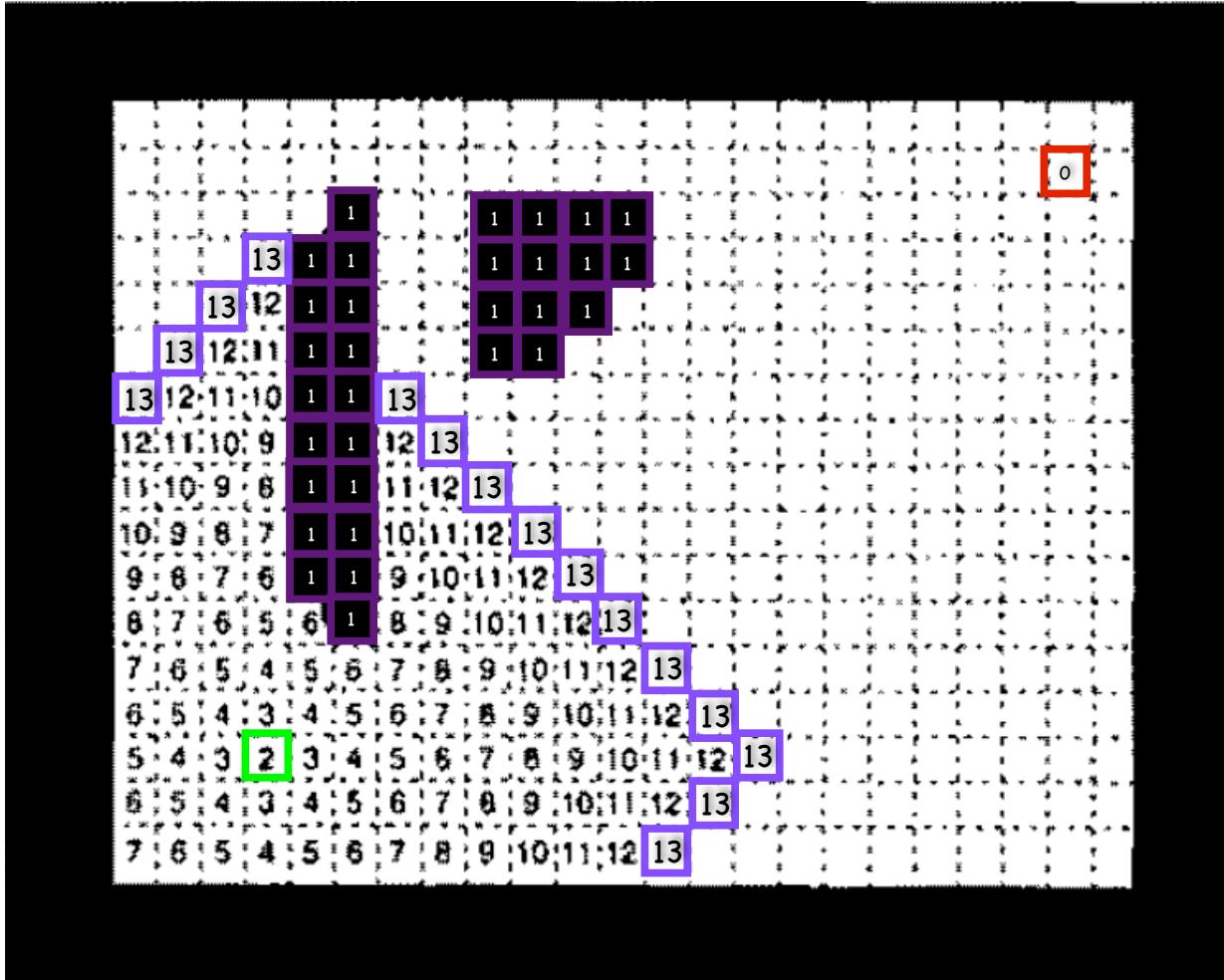
Goal: mark with 2

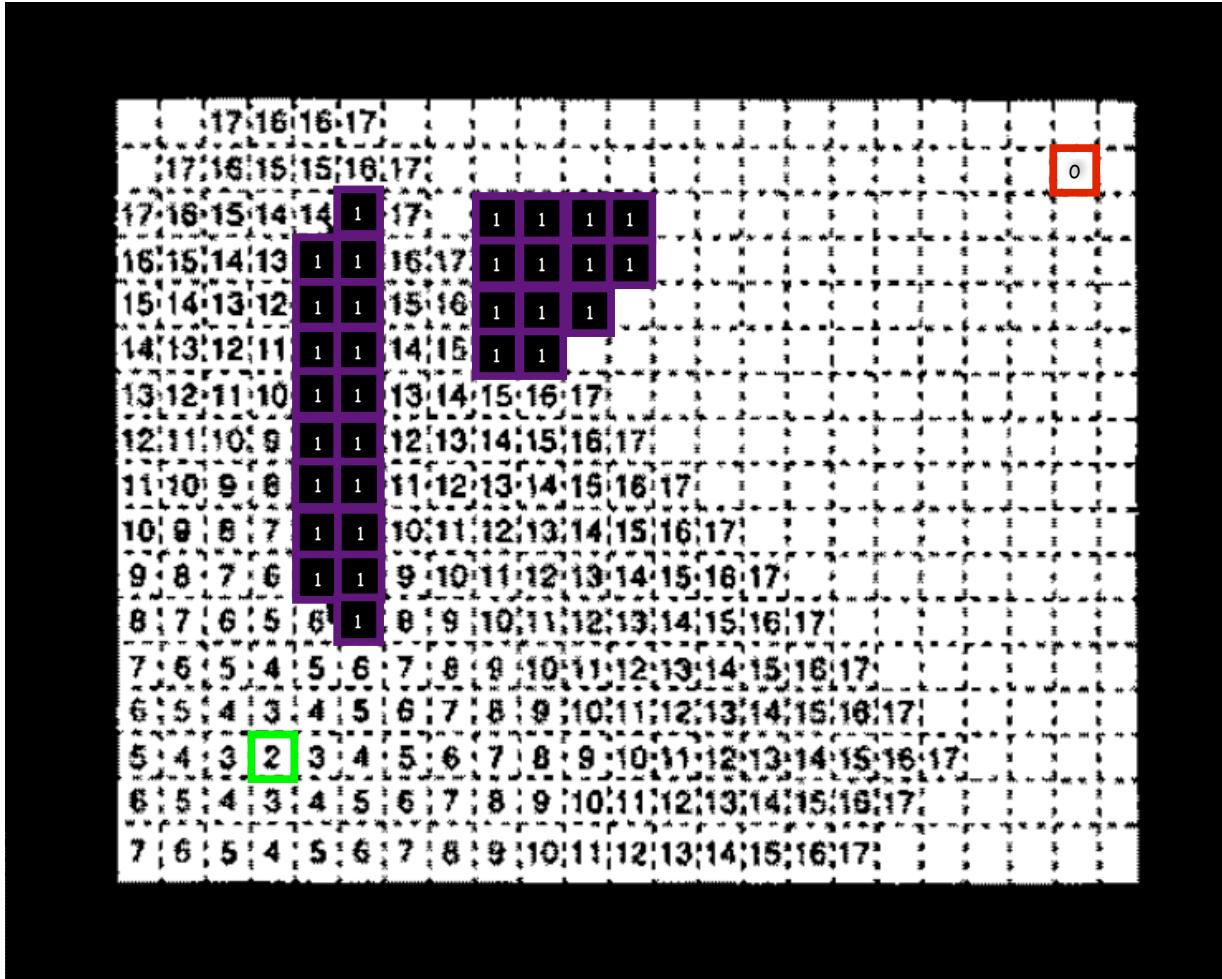
Obstacles: mark with 1

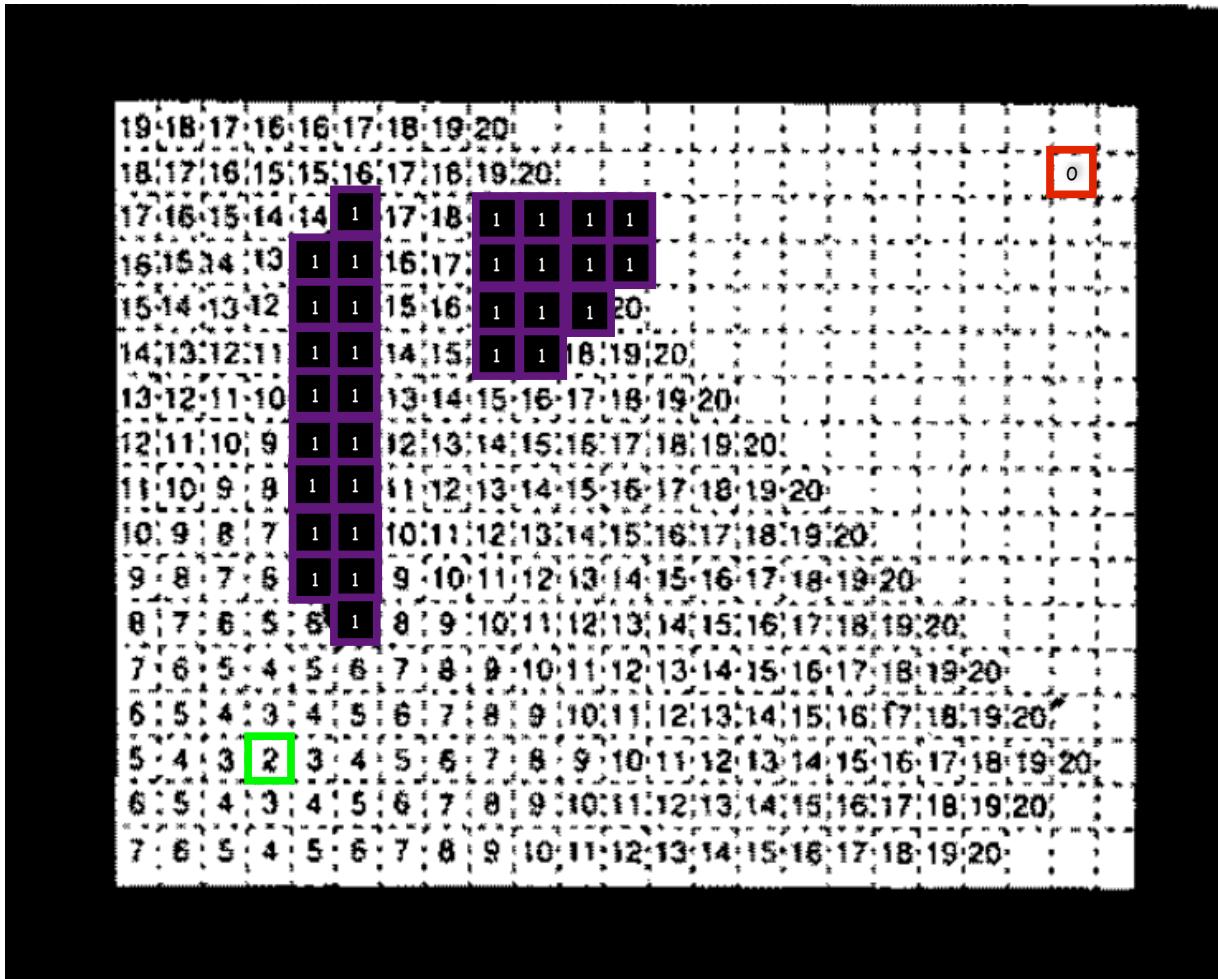


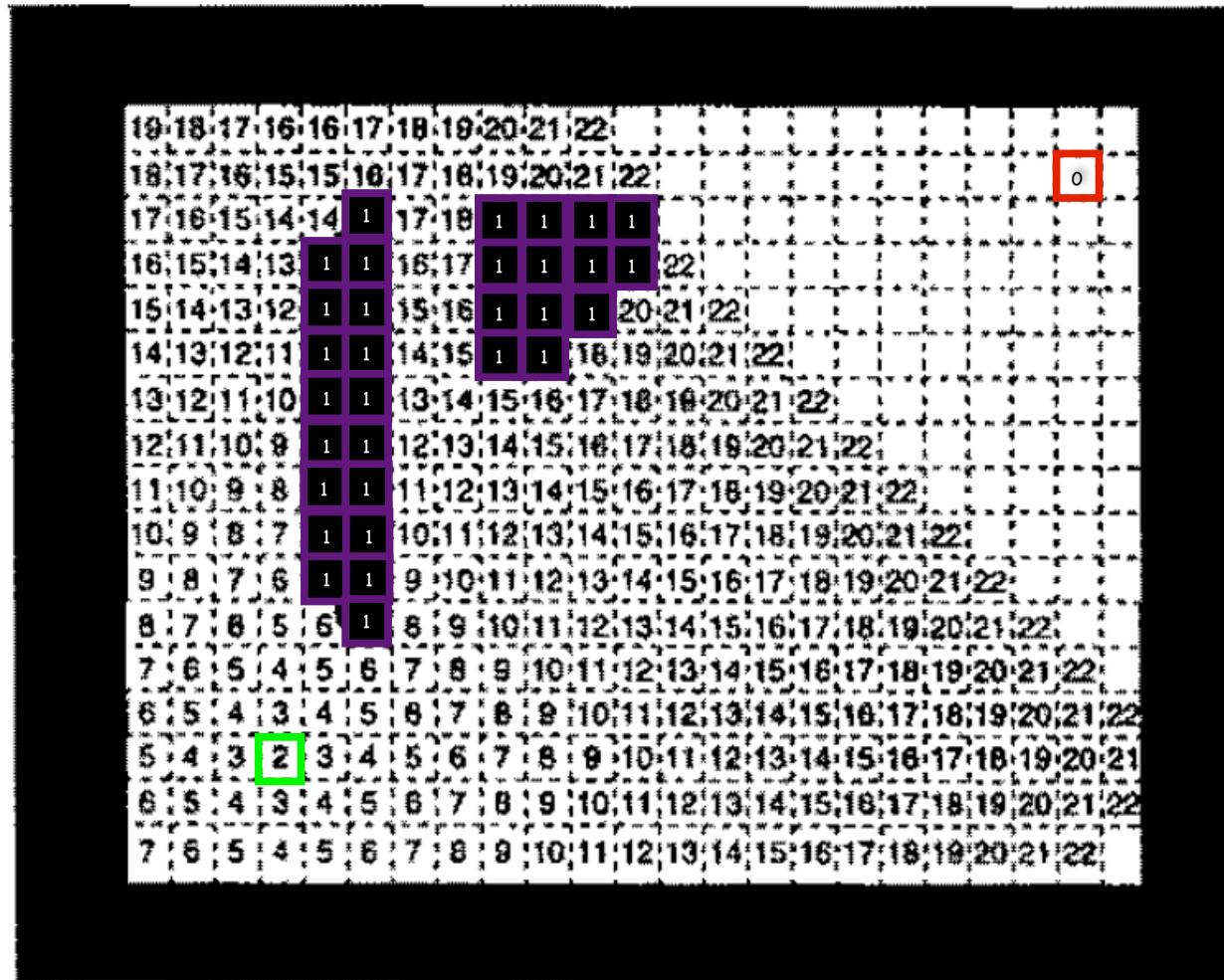


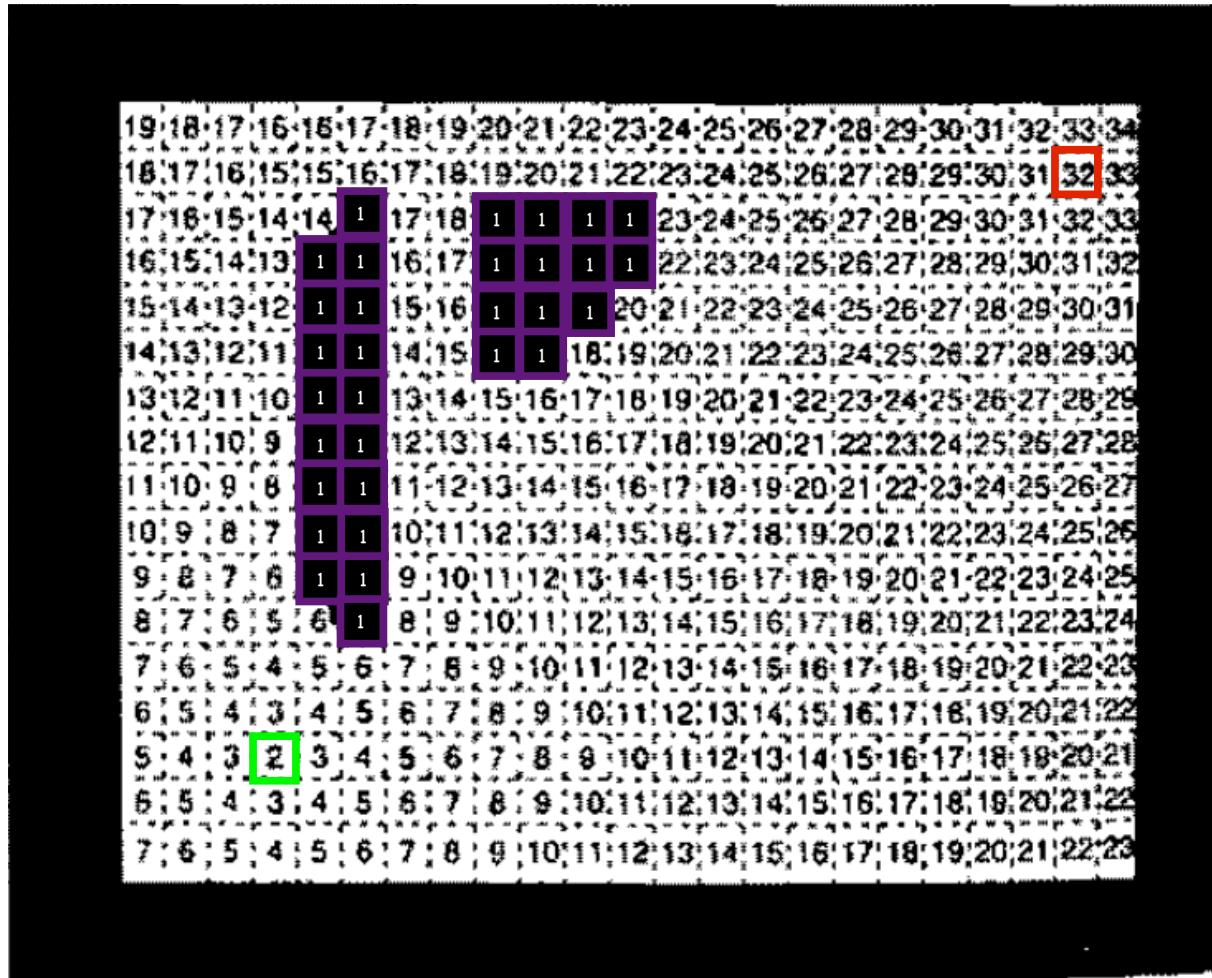




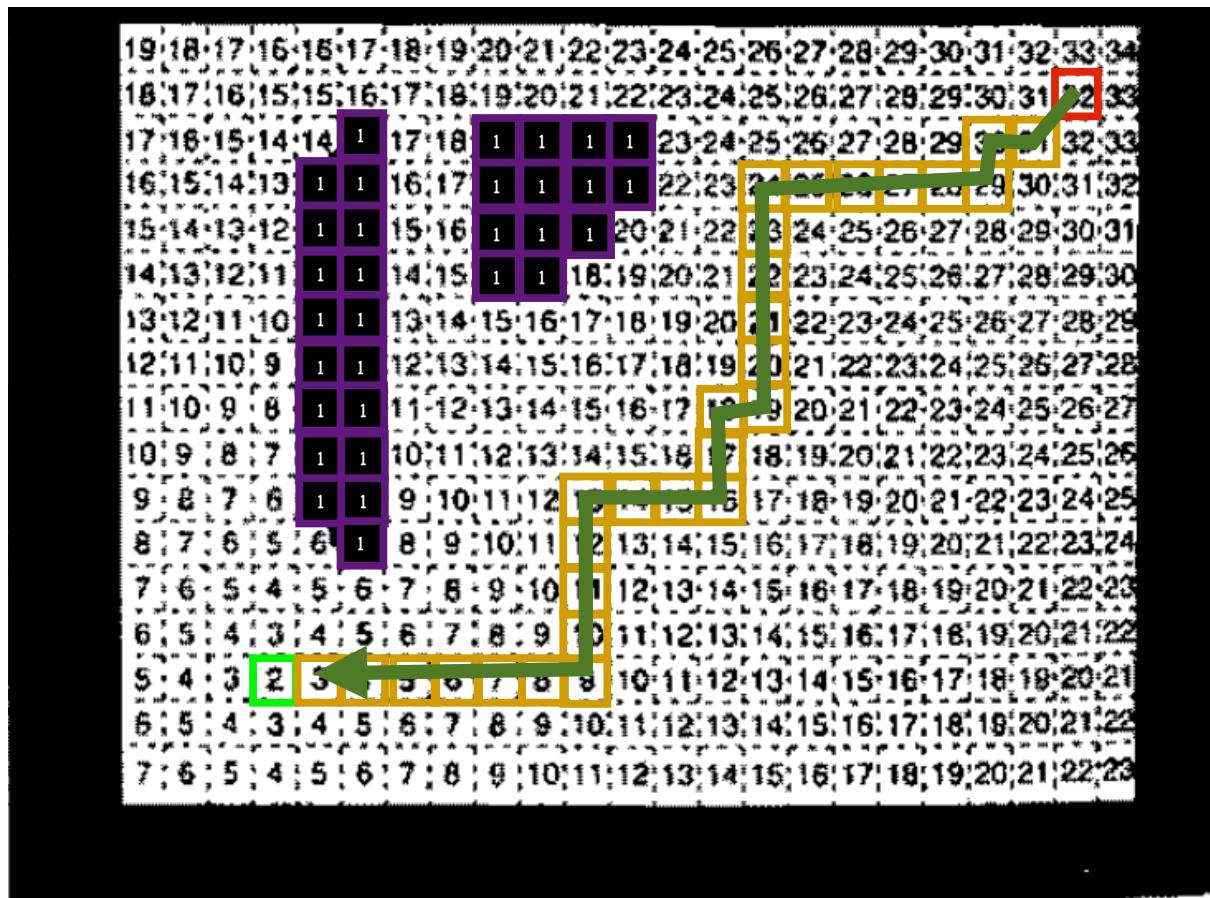






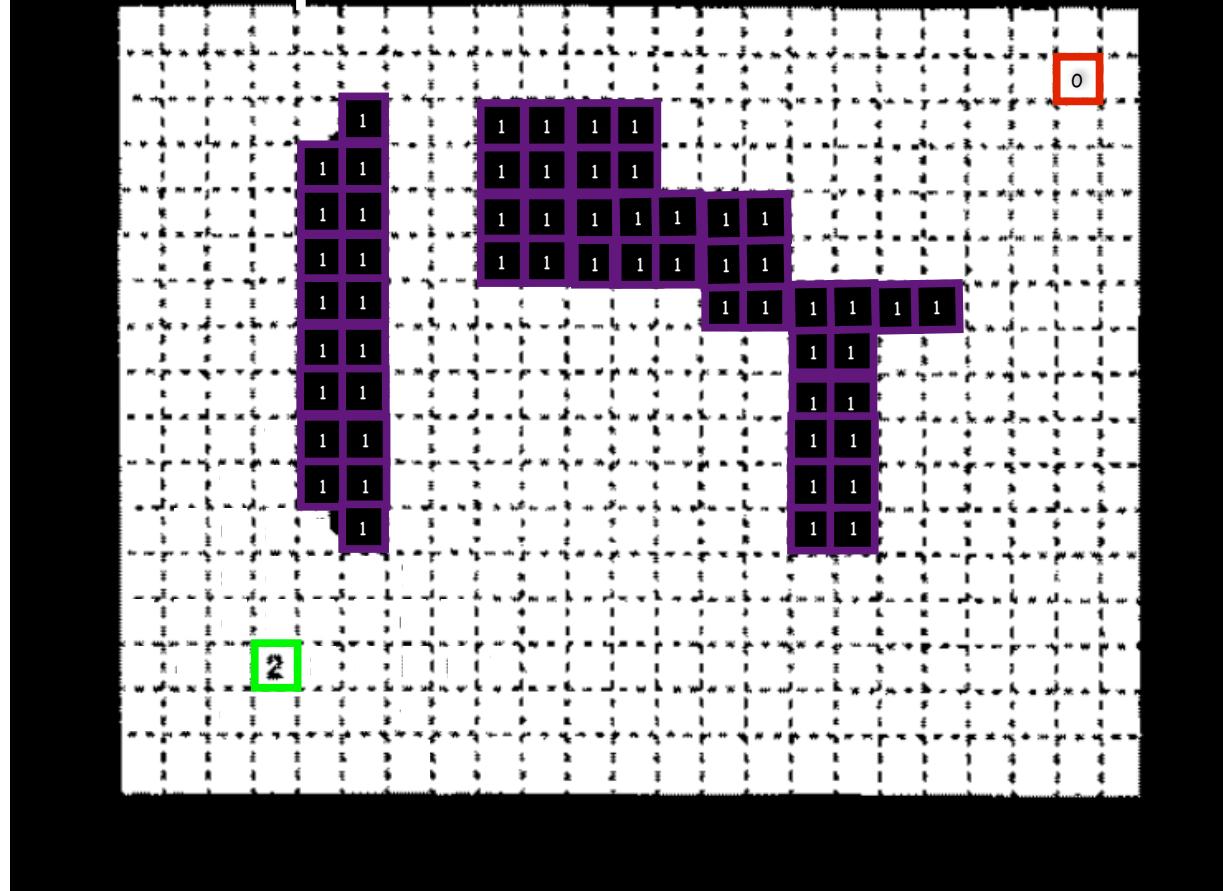


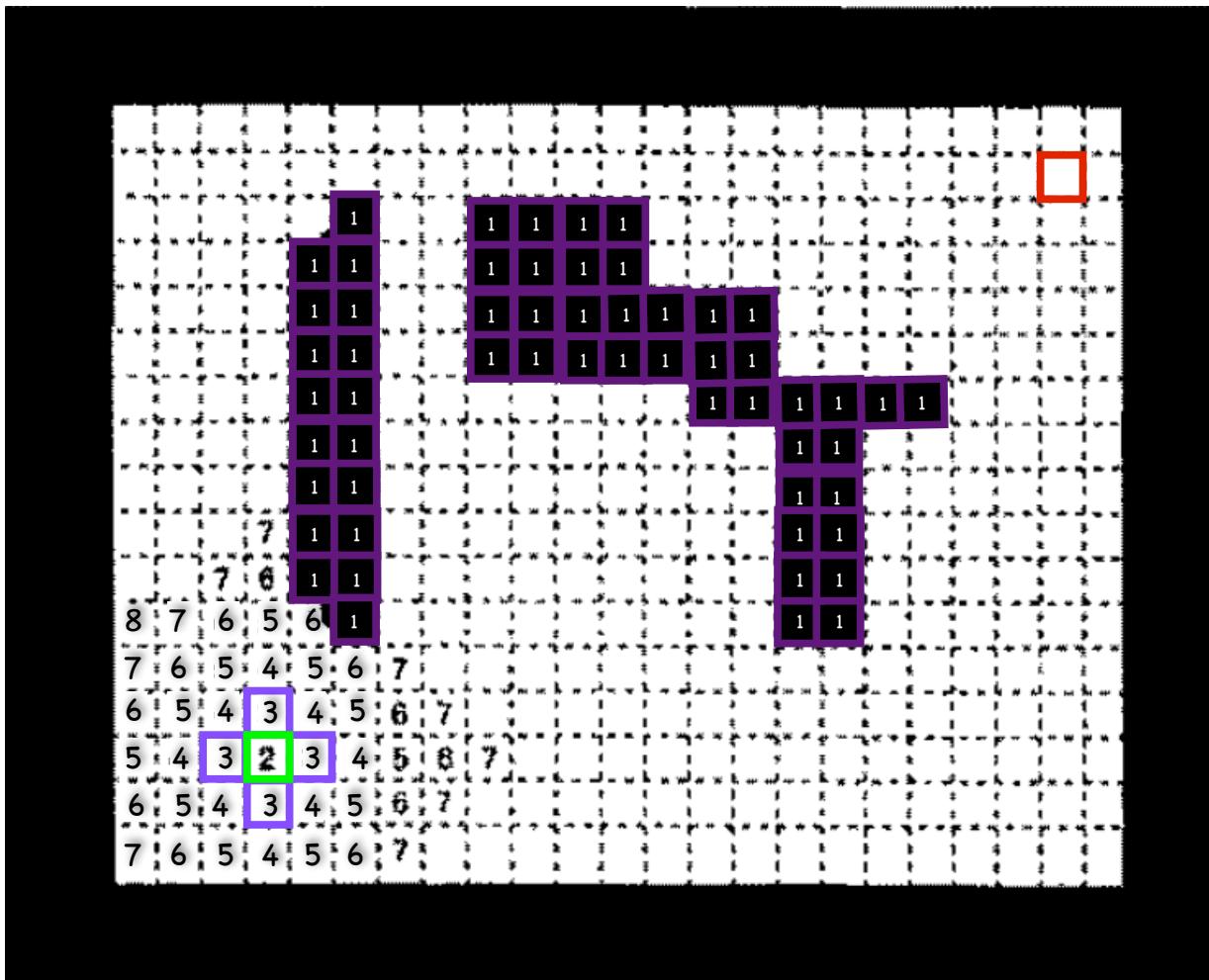
Once start reached,
follow brushfire potential to goal

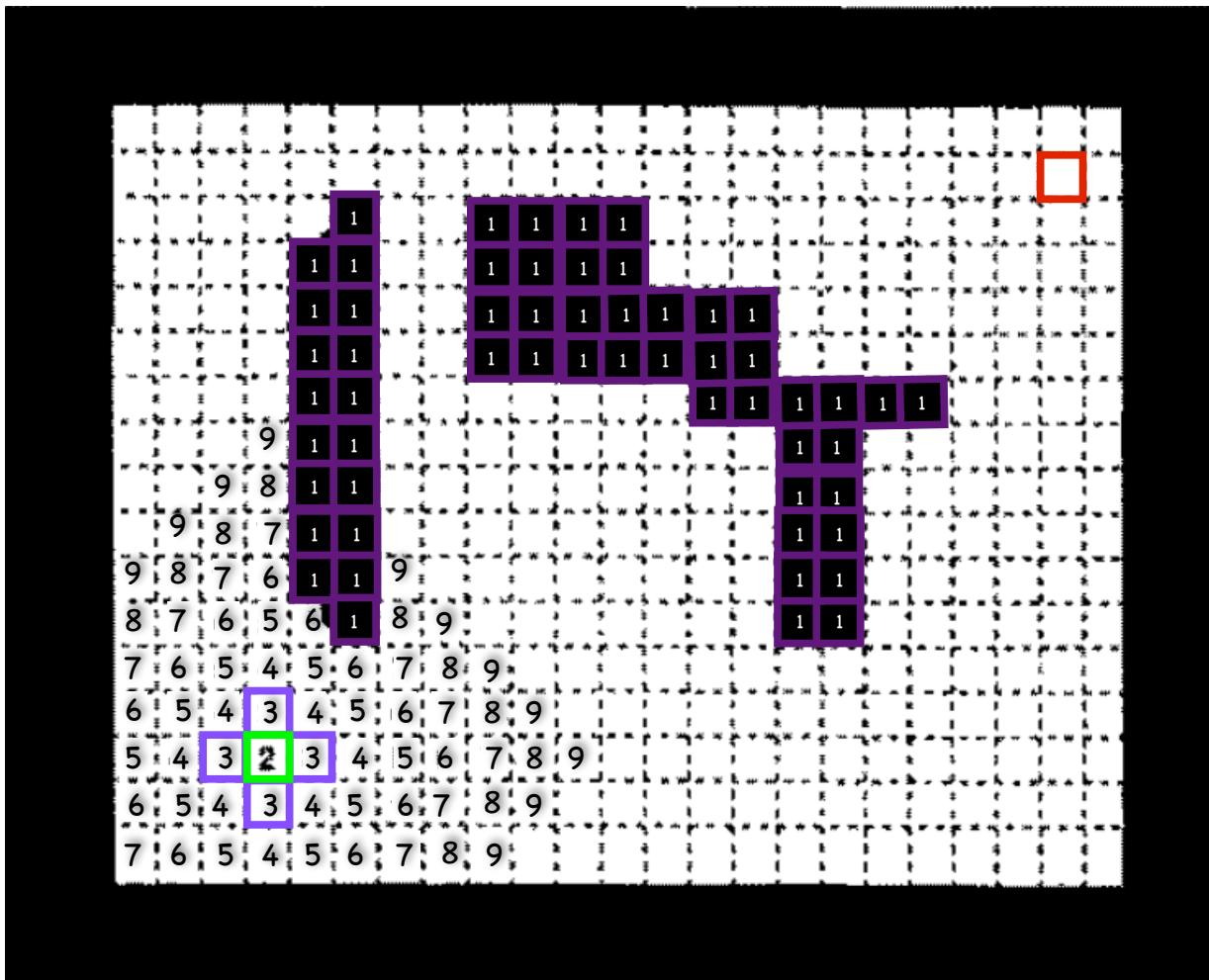


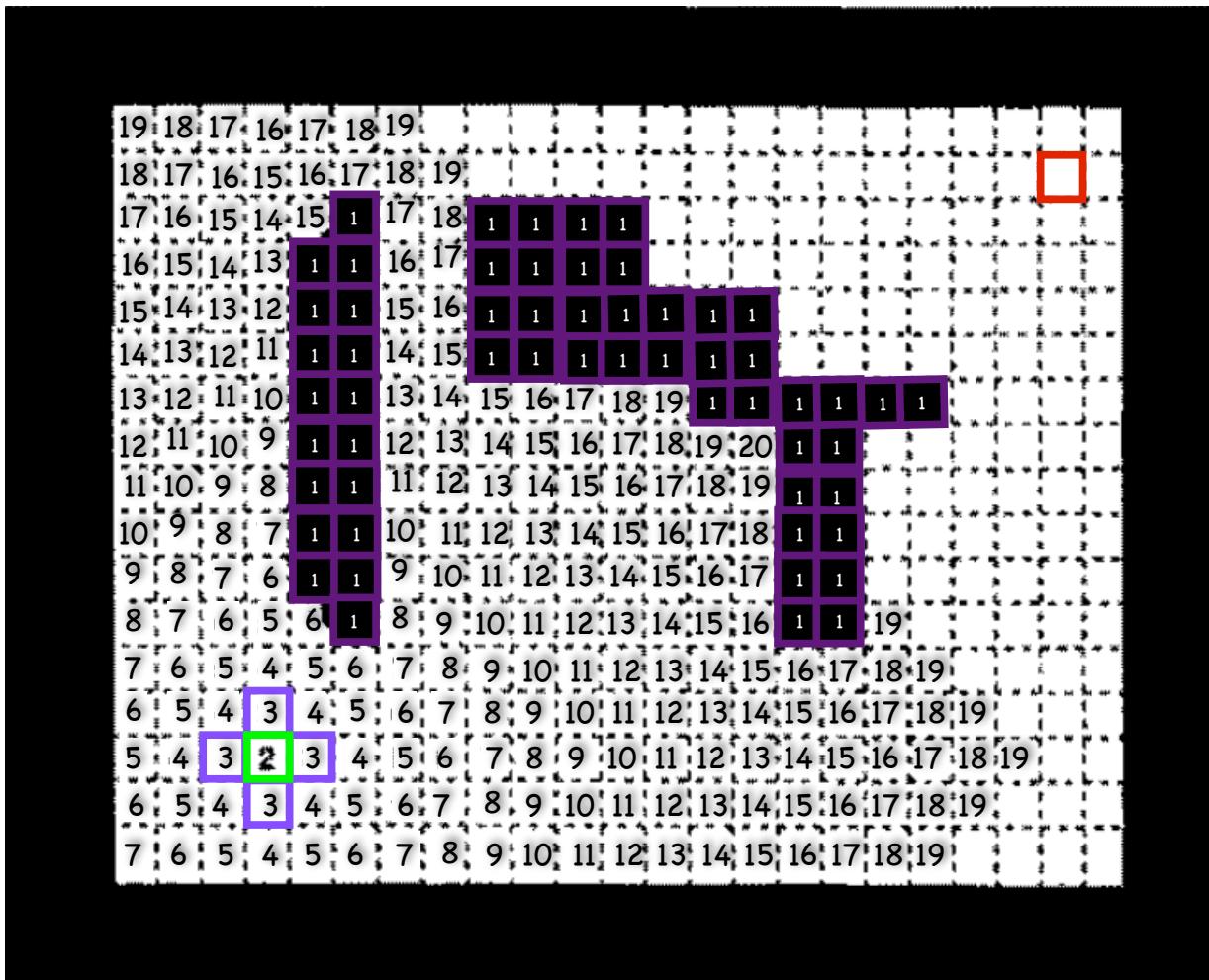
Example with Local Minima

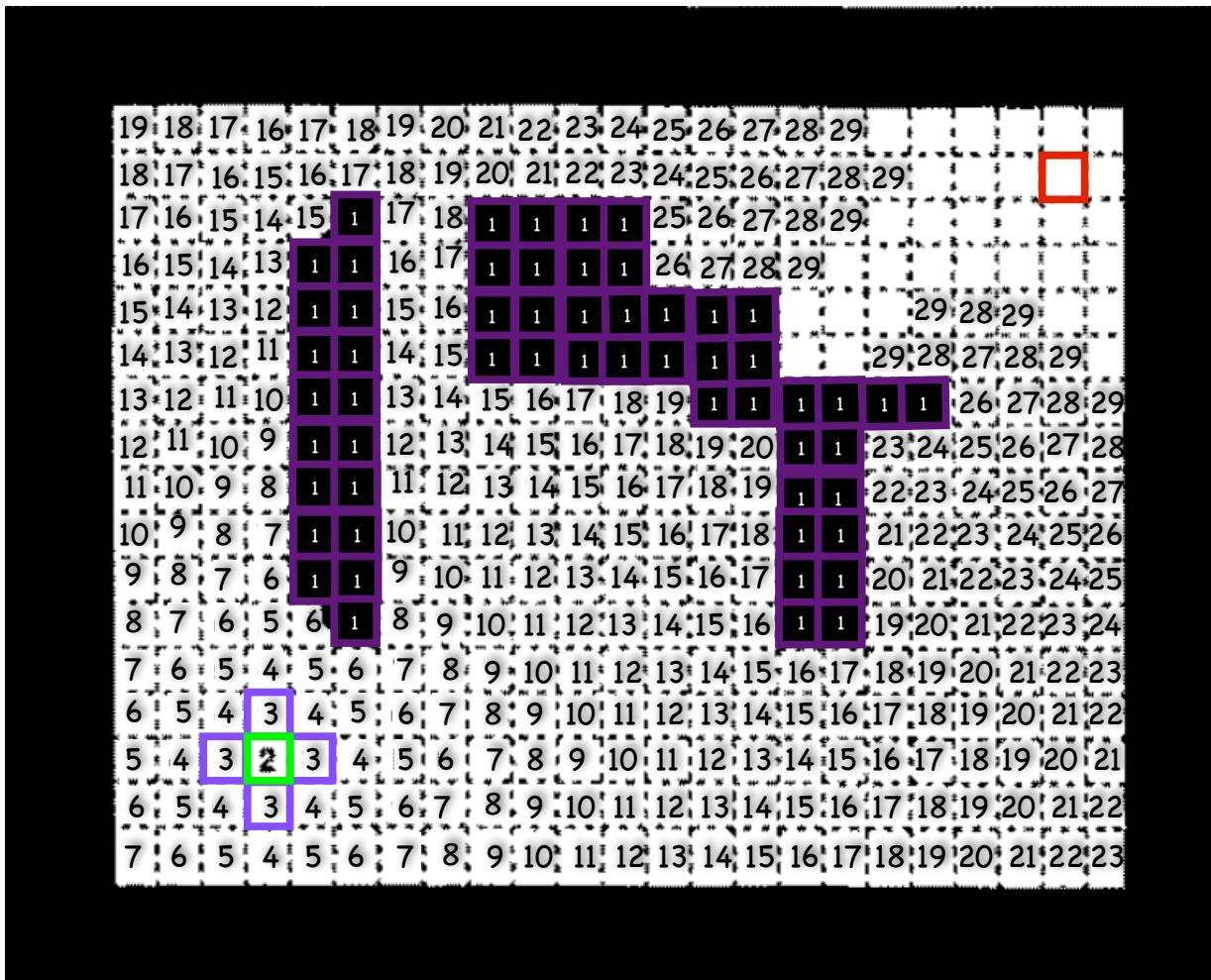
Example with Local Minima



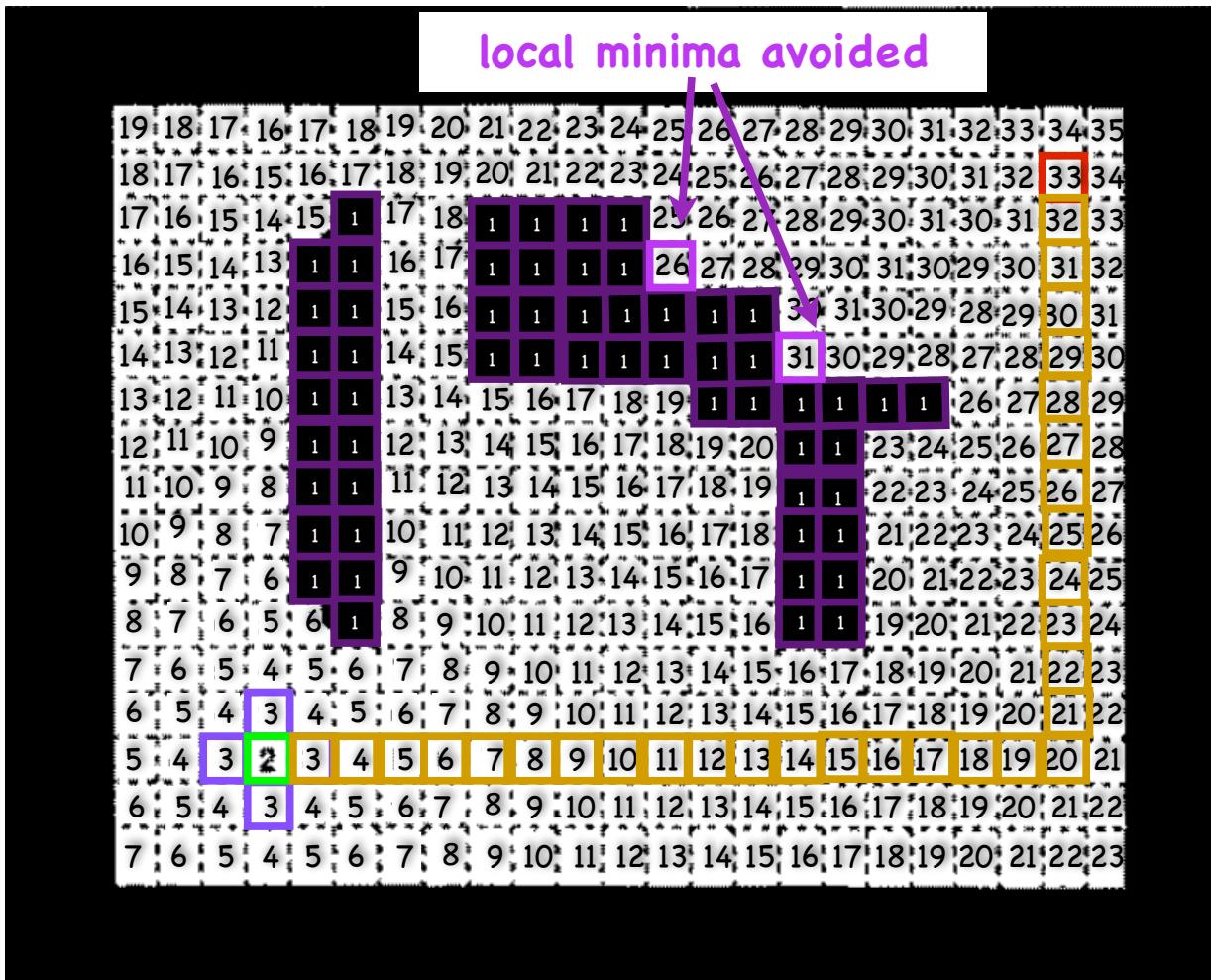






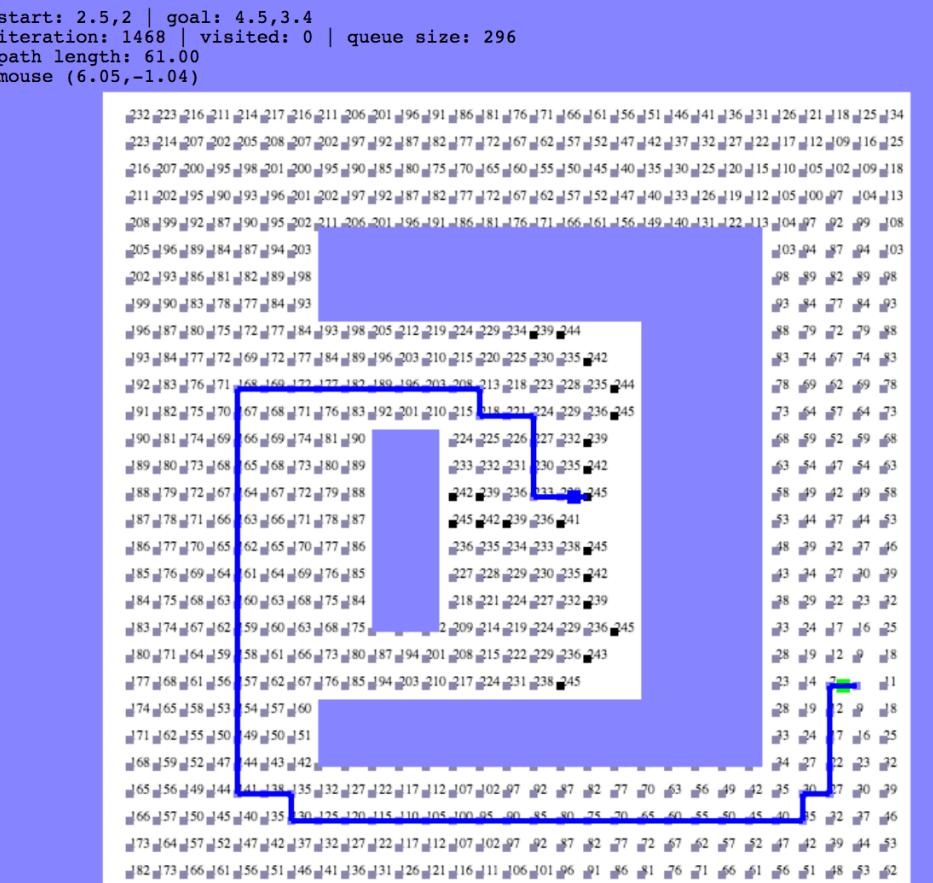


19	18	17	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
18	17	16	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	
17	16	15	14	15	1	17	18	1	1	1	1	25	26	27	28	29	30	31	30	31	32	33	
16	15	14	13	13	1	1	16	17	1	1	1	1	26	27	28	29	30	31	30	29	30	31	
15	14	13	12	12	1	1	15	16	1	1	1	1	1	1	30	31	30	29	28	29	30	31	
14	13	12	11	11	1	1	14	15	1	1	1	1	1	1	31	30	29	28	27	28	29	30	
13	12	11	10	10	1	1	13	14	15	16	17	18	19	1	1	1	1	1	1	26	27	28	29
12	11	10	9	9	1	1	12	13	14	15	16	17	18	19	20	1	1	23	24	25	26	27	28
11	10	9	8	8	1	1	11	12	13	14	15	16	17	18	19	1	1	22	23	24	25	26	27
10	9	8	7	7	1	1	10	11	12	13	14	15	16	17	18	1	1	21	22	23	24	25	26
9	8	7	6	6	1	1	9	10	11	12	13	14	15	16	17	1	1	20	21	22	23	24	25
8	7	6	5	6	1	8	9	10	11	12	13	14	15	16	1	1	19	20	21	22	23	24	
7	6	5	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
5	4	3	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	
6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
7	6	5	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	



Maithili's wavefront planner

My 2D planner



Can we extend potential fields
for arm navigation?

Potential Fields for Robot Arm

- Define endeffector goal as the attractive potential with cone potential
- Define repulsive potentials wrt. collision objects
 - Select points on robot links with “bowl” potential from nearest object
- Use manipulator Jacobian to transform potential at each point into velocities at robot joints
- Weighted sum of transformed velocities to generate control

Navigation Recap

Navigation Recap

Bug X

- Complete
- Non-optimal
- Planar

Subsumption and FSMs

- Fast but not adaptive
- Emphasis on good design

Potential Fields

- Complete in special cases
- Non-optimal
- General C-spaces
- Scales w/dimensionality

Grid Search/Wavefront

- Complete
- General C-spaces
- Limited dimensionality

Random walk

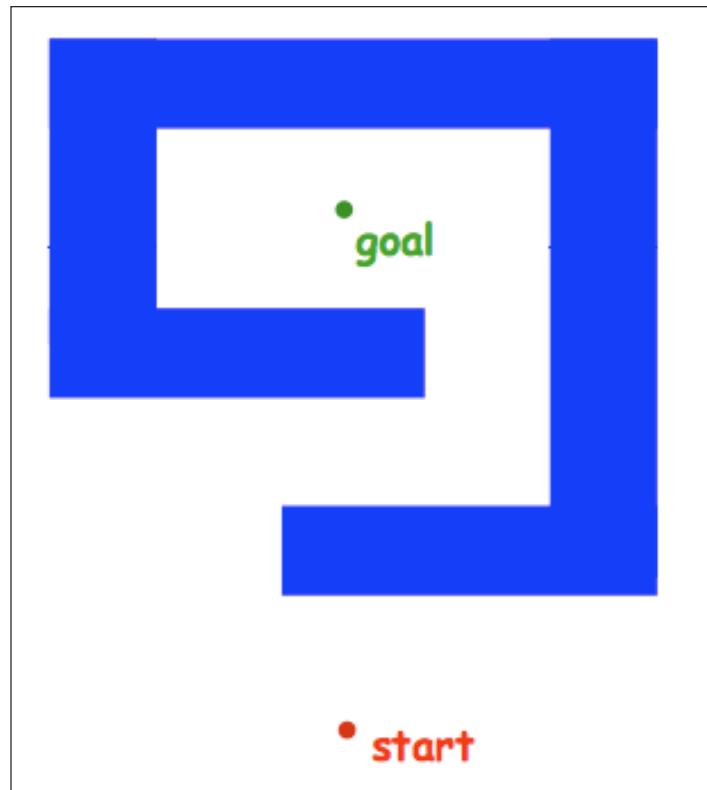
- Will find path eventually

Sampling roadmaps/RRT

- Probabilistically complete
- General
- Tractable (with good sampling)
- Scales w/dimensionality
- Not necessarily optimal

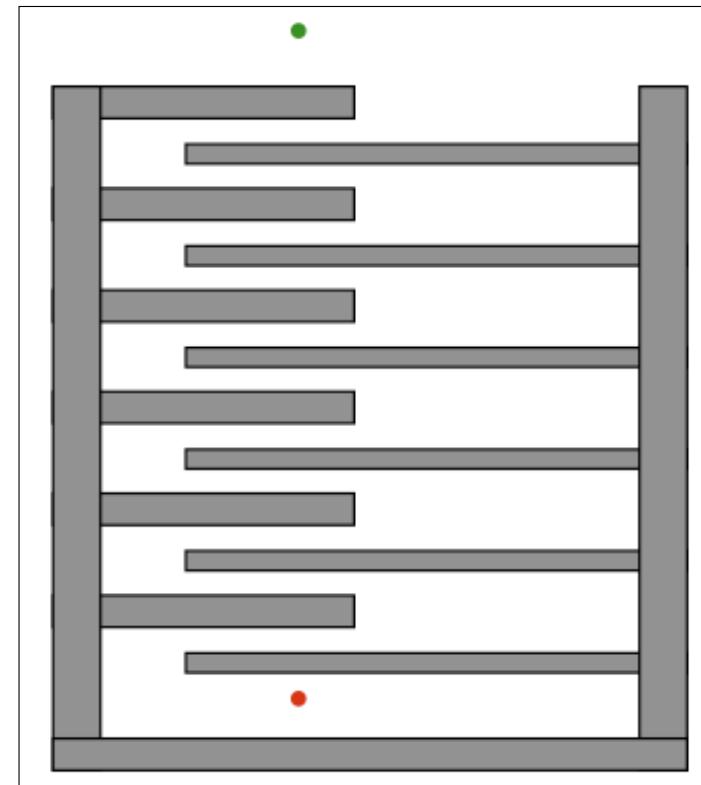
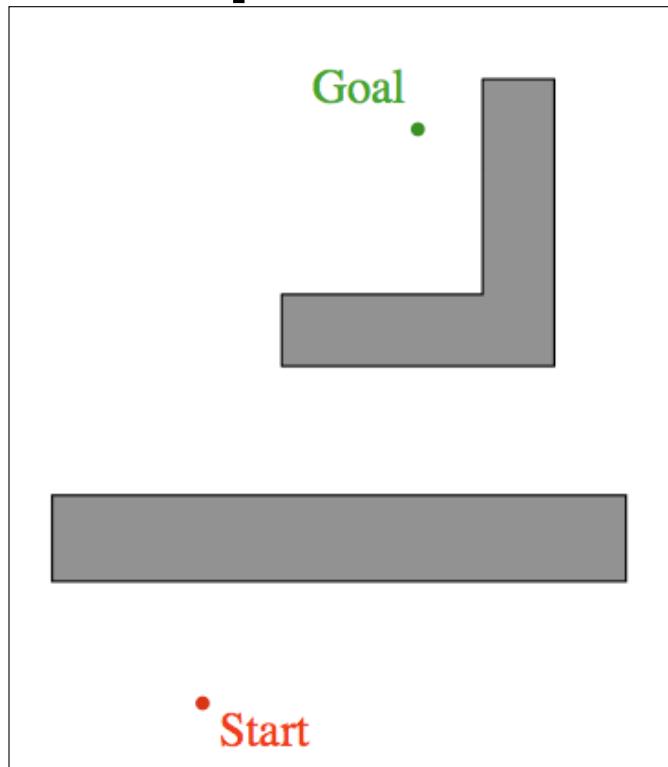
Compare: BugX examples

Describe likely paths from potential field and wavefront planners



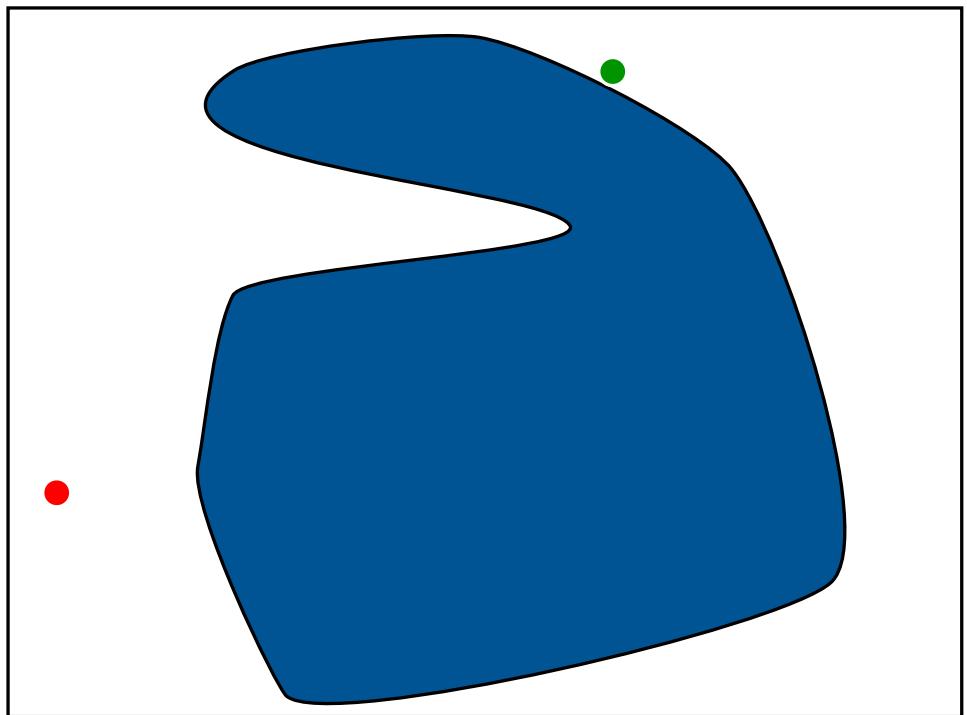
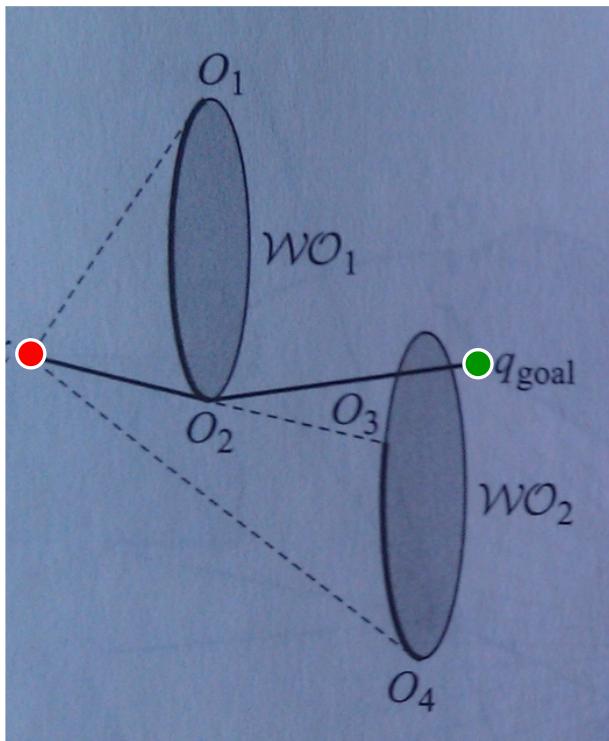
Compare: BugX examples

Describe likely paths from potential field and wavefront planners



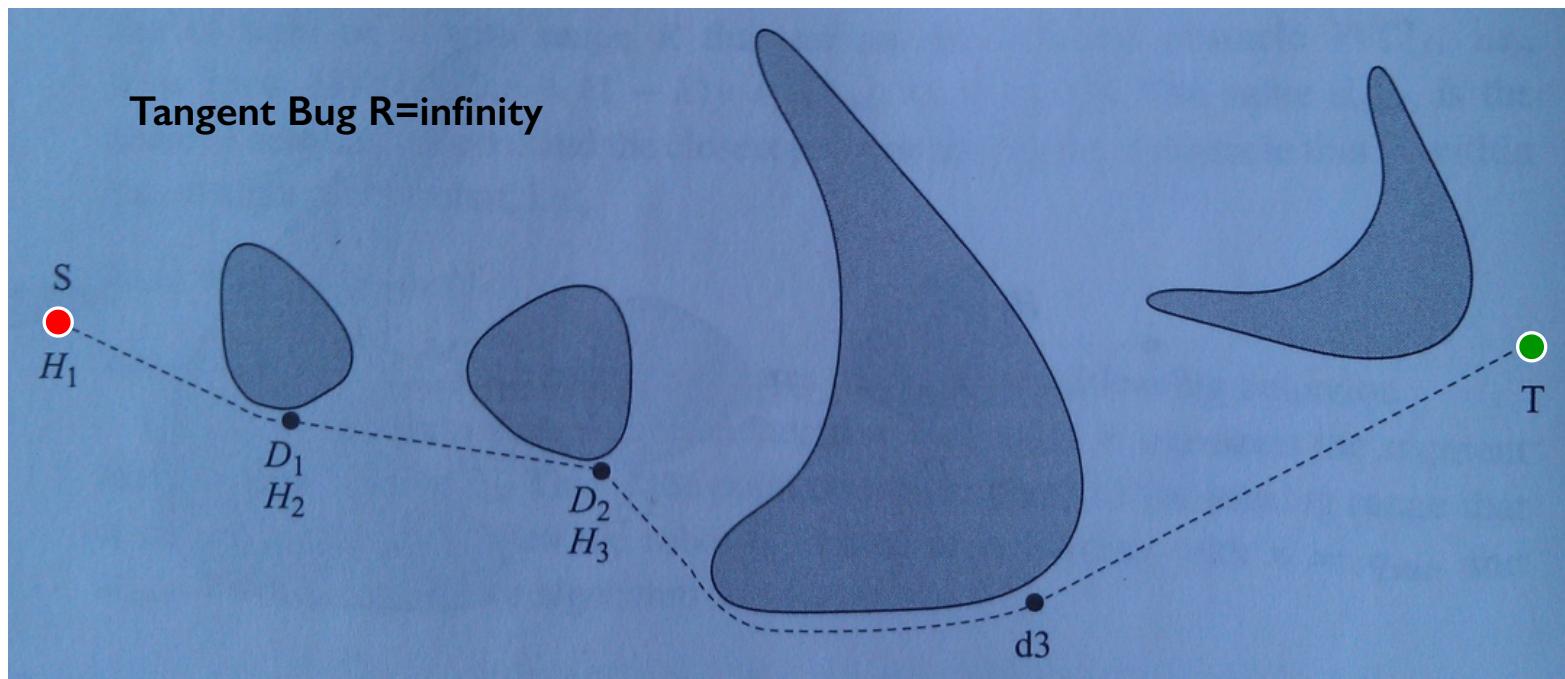
Compare: BugX examples

Describe likely paths from potential field and wavefront planners



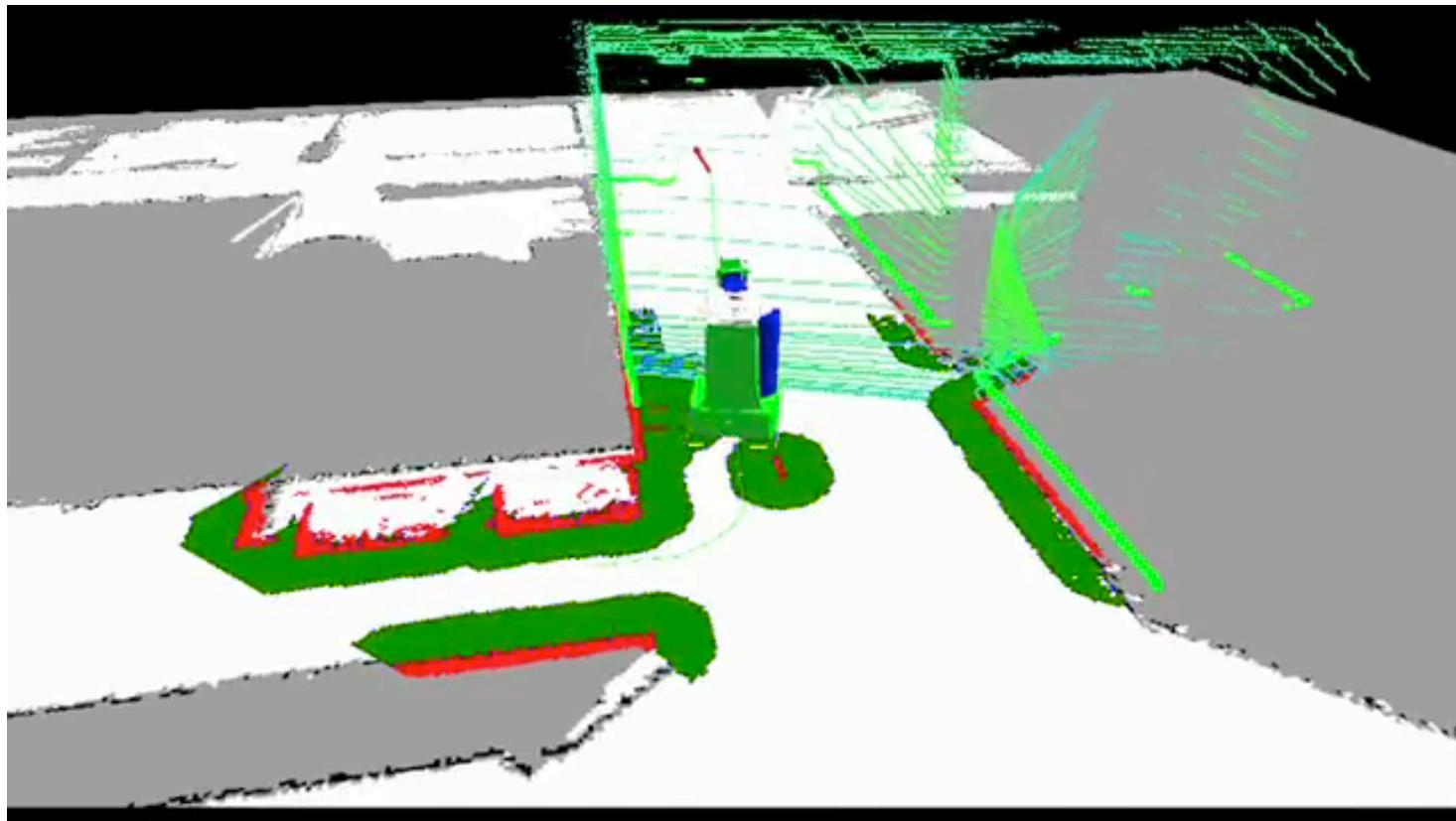
Compare: BugX examples

Describe likely paths from potential field and wavefront planners



Navigation in ROS

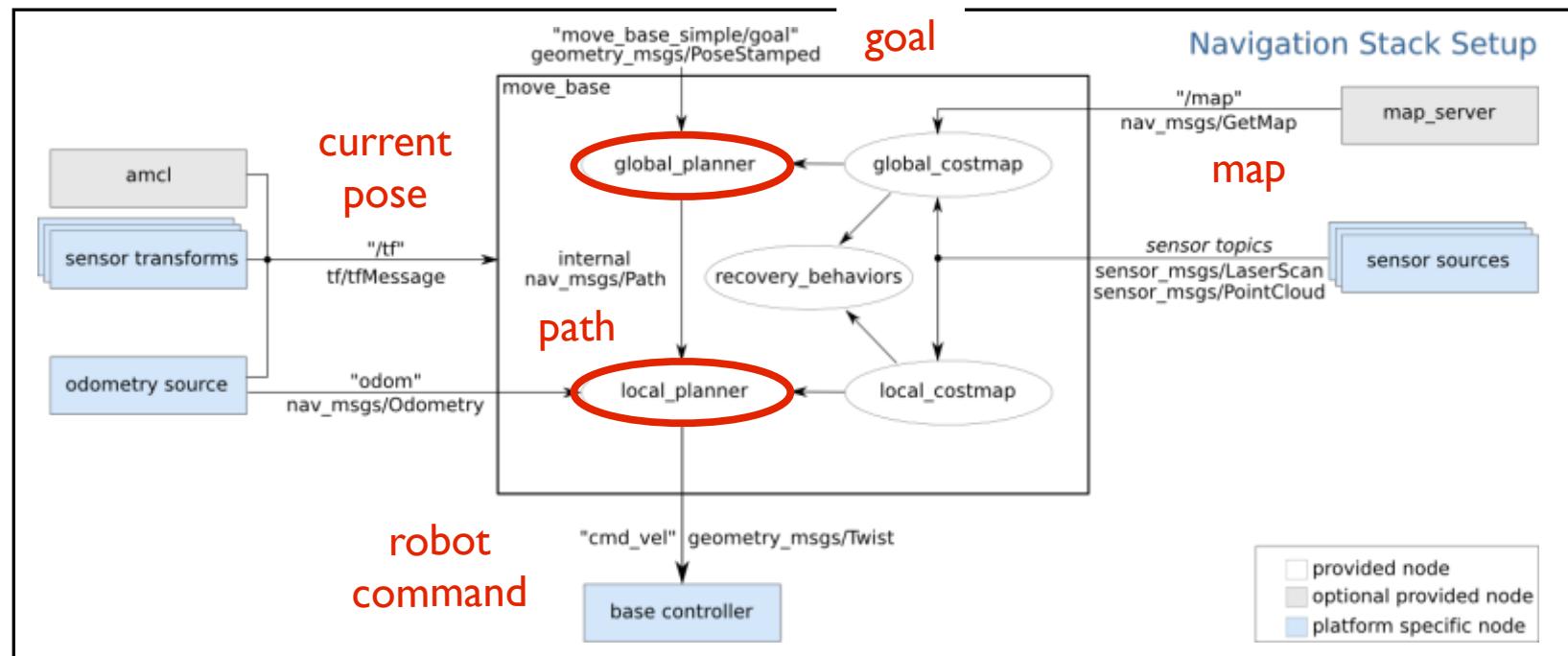
ROS Navigation Stack



<http://www.ros.org/wiki/navigation>

Michigan EECS 398/567 ROB 510 - autorob.org

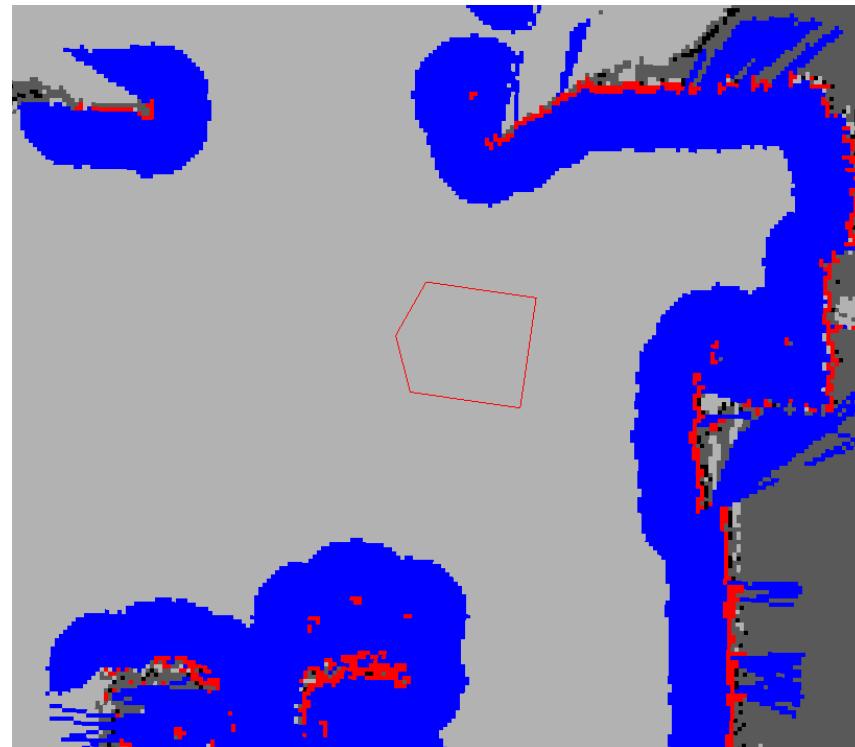
move_base



- ④ Core of nav_stack for planning and motion control

Costmap in ROS

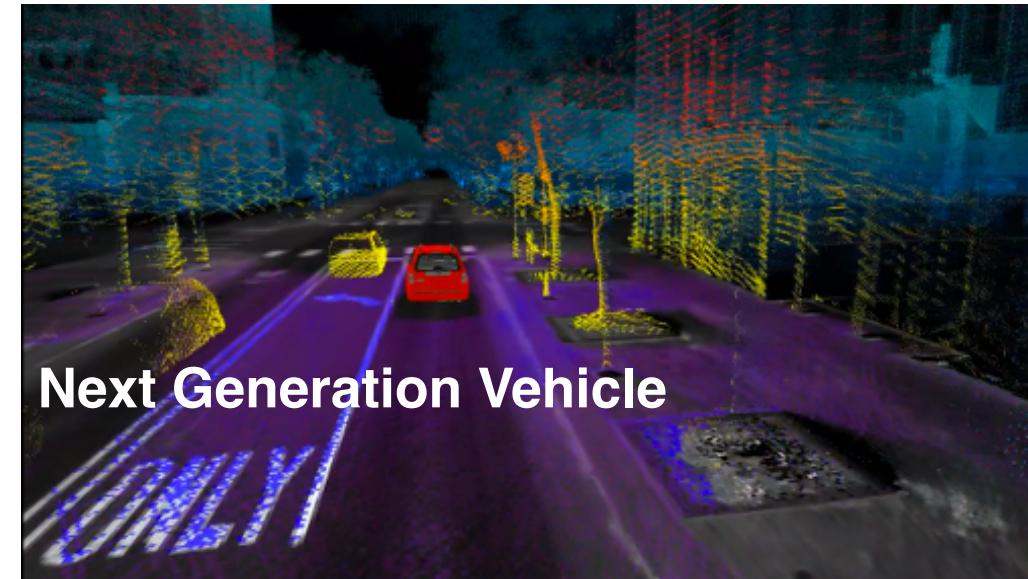
- Treat as a Minkowski sum over the given map



http://www.ros.org/wiki/costmap_2d

Michigan EECS 398/567 ROB 510 - autorob.org

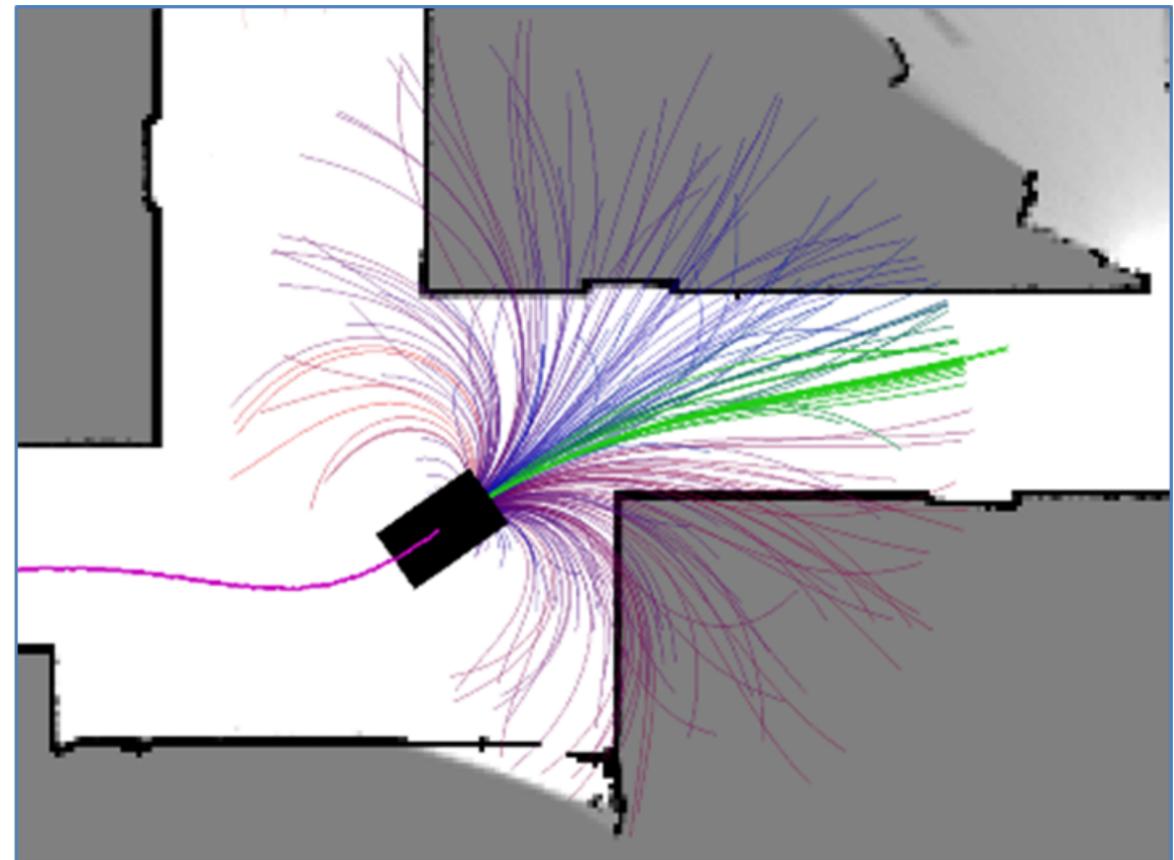
Navigation in LCM



Michigan EECS 398/567 ROB 510 - autorob.org

[Park, Johnson, Kuipers 2012]

Vulcan Navigation



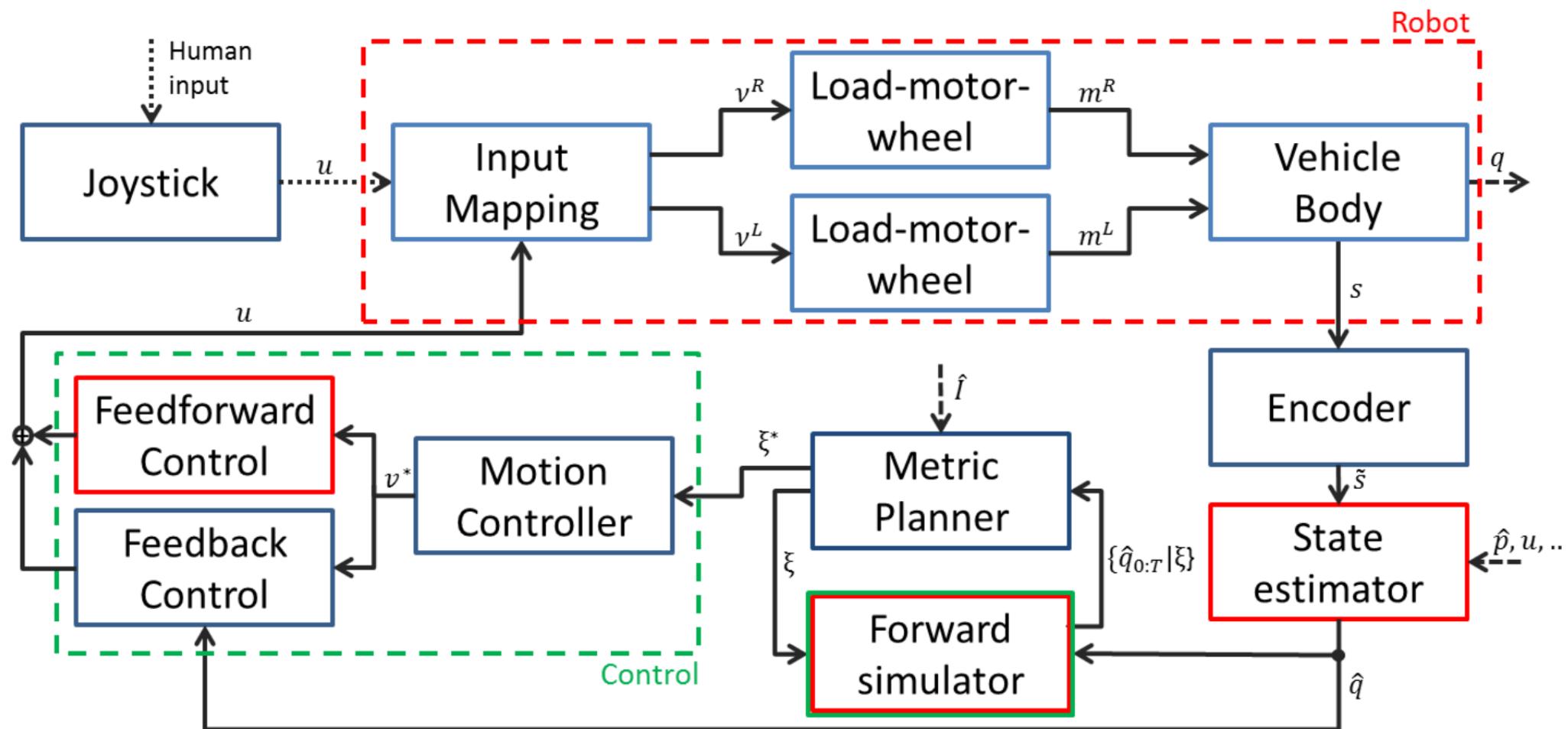


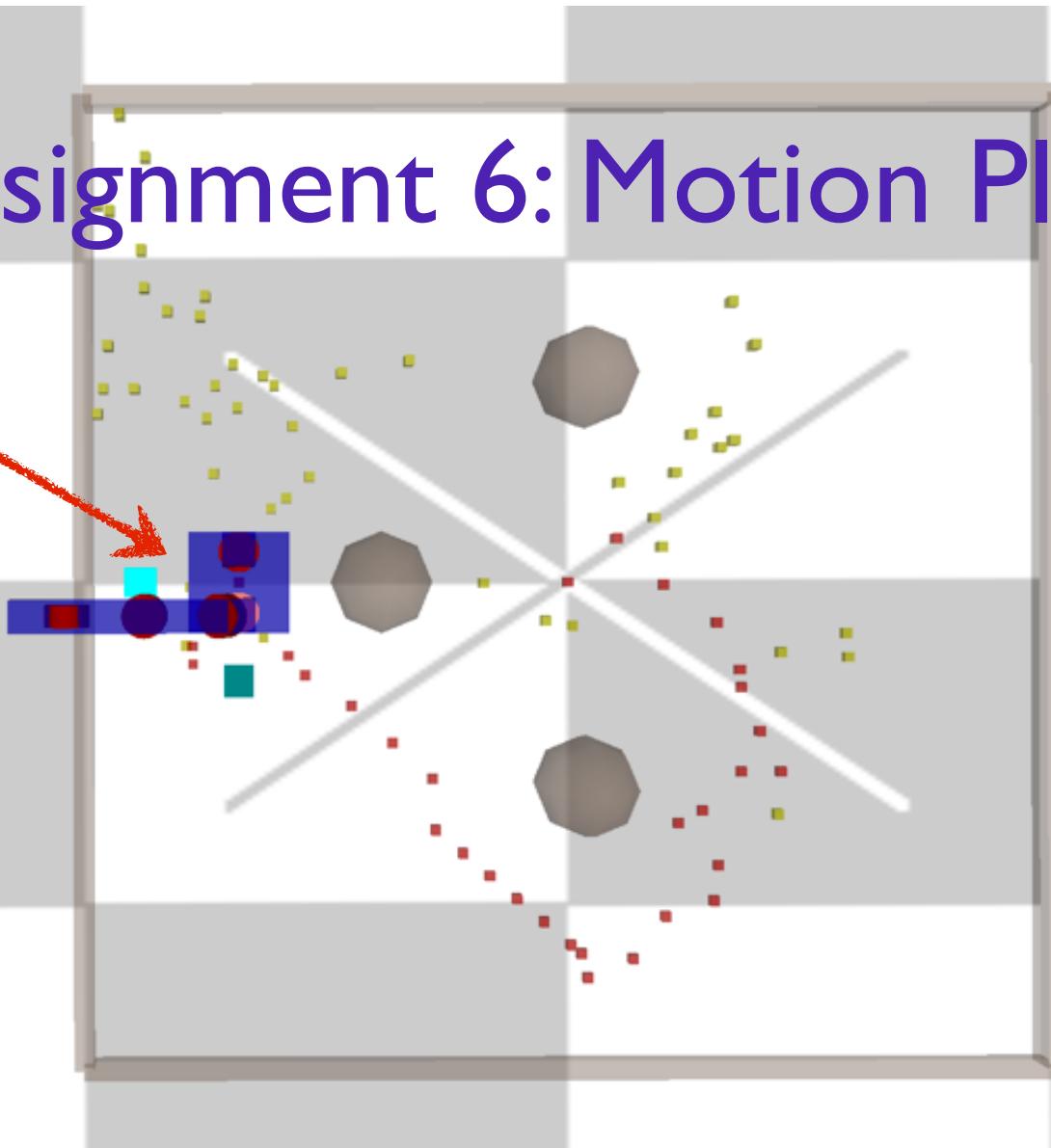
Figure A.1: Vulcan system diagram.

Assignment 6: Motion Planning

- Generate a collision free motion plan to the world origin and zero joint angle configuration

Assignment 6: Motion Planning

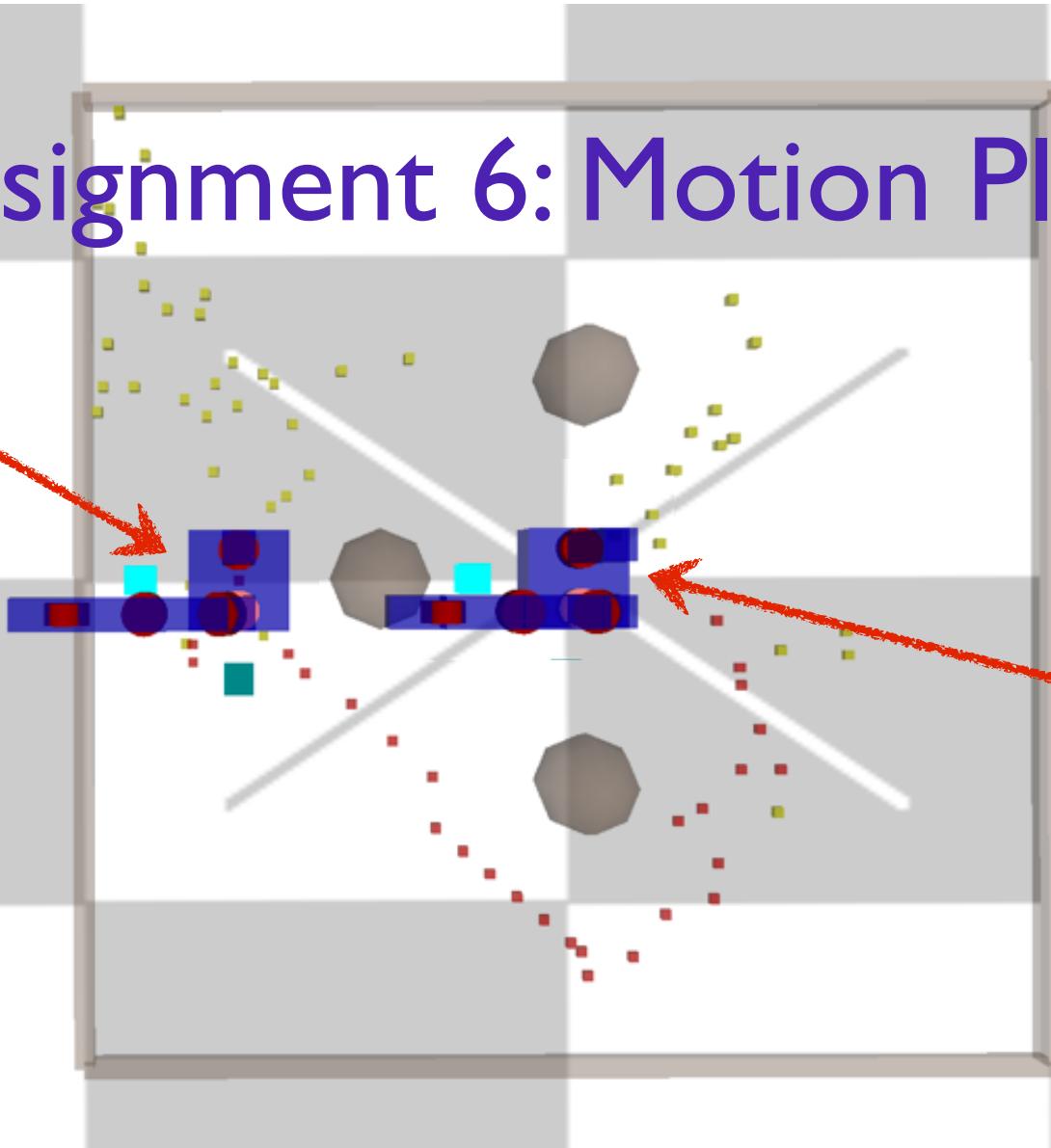
Start: random
non-colliding
configuration



Assignment 6: Motion Planning

Start: random
non-colliding
configuration

Goal: zero
configuration at
world origin

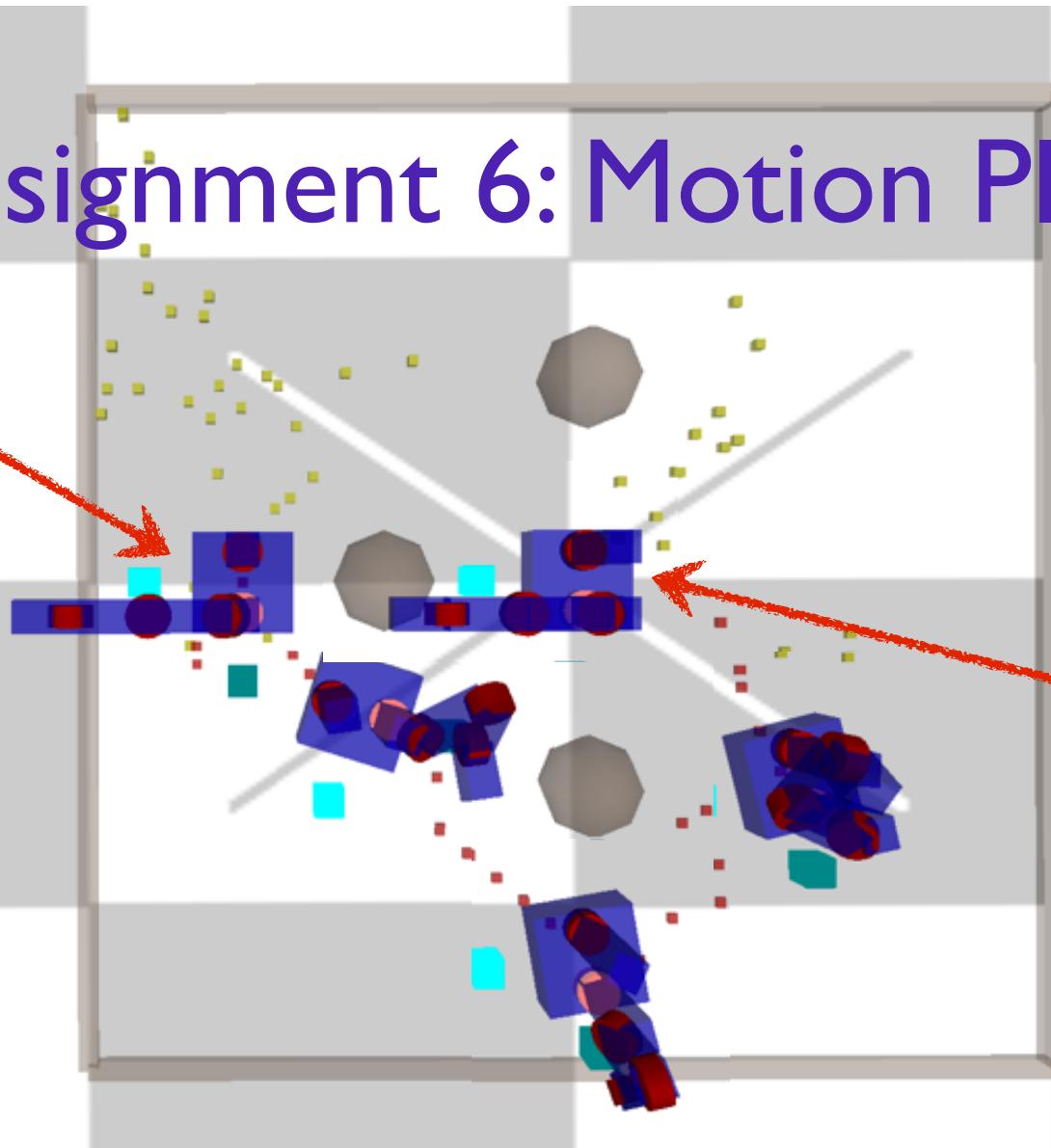


Assignment 6: Motion Planning

Start: random
non-colliding
configuration

Goal: zero
configuration at
world origin

Generate
collision-free
motion plan



<https://github.com/ohseejay/kineval-stencil-fall16>

ohseejay / kineval-stencil-fall16

Stencil code for KinEval (Kinematic Evaluator) for robots

Recommended starting point:
Update code stencil from Path Planning project

2 commits 1 branch
Branch: master ▾ New pull request

odestcj initial commit
js
kineval
project_pathplan
project_pendularm
robots
tutorial_heapsort
tutorial_js
worlds

Initial commit
initial commit

2 contributors
Upload files Find file Clone or download ▾
Latest commit 5fd521e 26 days ago
26 days ago
26 days ago
26 days ago

various 3D worlds for testing robots included by:
`home.html?world=worlds/world_local_minima.js`

rob.org

GitHub, Inc. (US) | https://github.com/ohseejay/kineval-stencil | rrt connect

ohseejay / kineval-stencil

Code Issues 0 Pull requests 0 Wiki Pulse Graphs

Stencil code for KinEval (Kinematic Evaluator) for robot control

2 commits 1 branch

Branch: master New pull request New file Upload

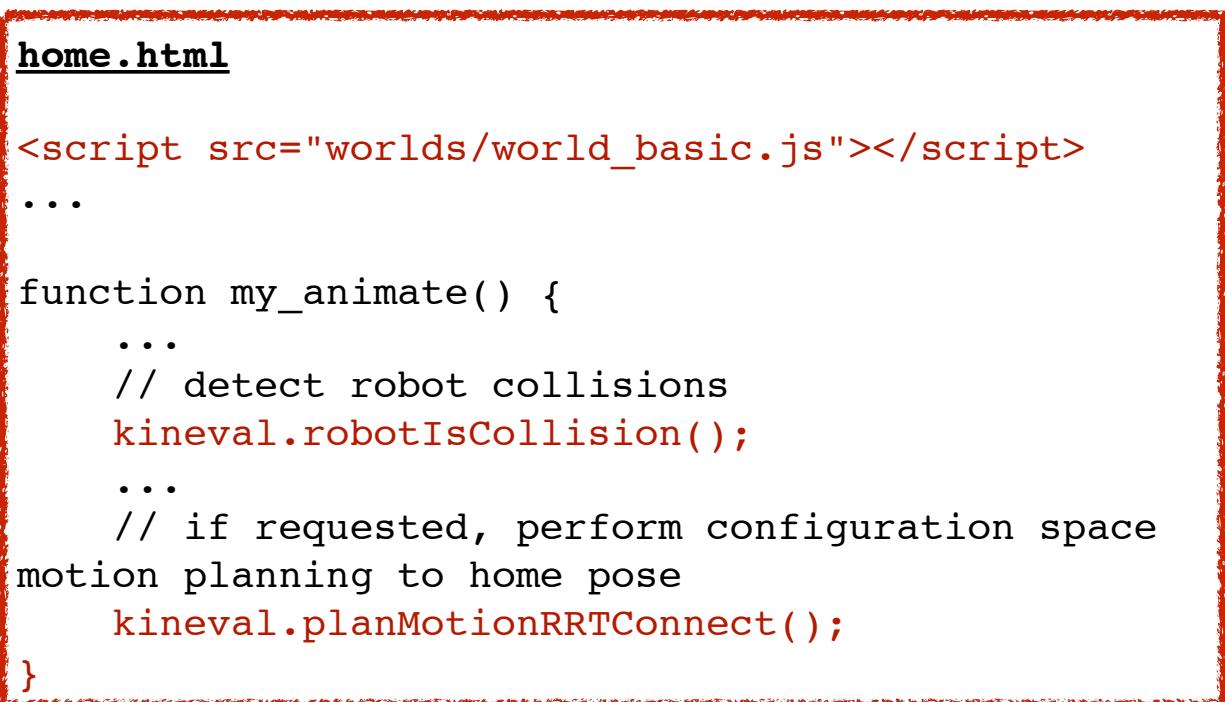
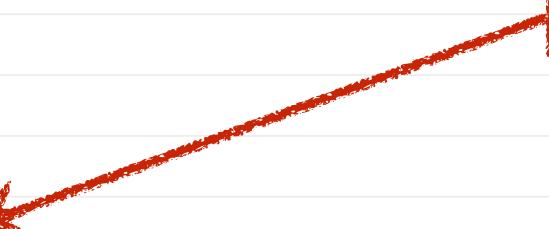
odestc initial commit

- js
- kineval
- pendularm
- robots
- rrt
- tutorial_js
- worlds
- README.md
- home.html

home.html

```
<script src="worlds/world_basic.js"></script>
...
function my_animate() {
    ...
    // detect robot collisions
    kineval.robotIsCollision();
    ...
    // if requested, perform configuration space
    motion planning to home pose
    kineval.planMotionRRTConnect();
}
```

initial commit 2 months ago
initial commit 2 months ago
initial commit 2 months ago



world file can be alternatively
loaded by a script tag

home.html

```
<script src="worlds/world_basic.js"></script>
```

```
...
```

```
function my_animate() {
```

```
...
```

```
// detect robot collisions
```

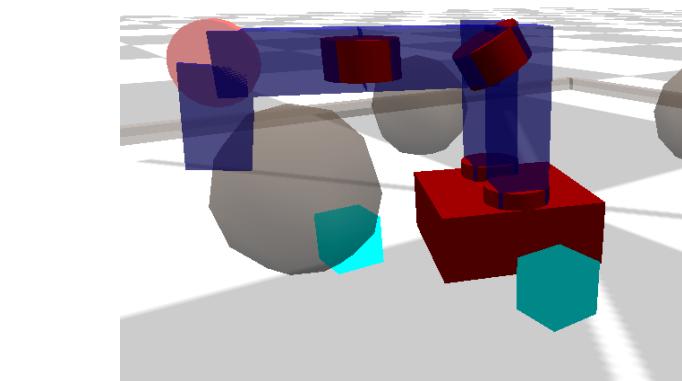
```
kineval.robotIsCollision();
```

```
...
```

```
// if requested, perform configuration space motion planning to home pose
```

```
kineval.planMotionRRTConnect();
```

```
}
```



detect if current
configuration is in collision
(colliding link turns red)

iterate motion planner

Branch: master **kineval-stencil / kineval /**

New file Upload files Find file History

 **odestcj** initial commit

Latest commit 2a1bd6e on Jan 11

..		
kineval.js	initial commit	2 months ago
kineval_collision.js	initial commit	2 months ago
kineval_controls.js	initial commit	2 months ago
kineval_forward_kinematics.js	initial commit	2 months ago
kineval_inverse_kinematics.js	initial commit	2 months ago
kineval_matrix.js	initial commit	2 months ago
kineval_quaternion.js	initial commit	2 months ago
kineval_robot_init.js	initial commit	2 months ago
kineval_rosbridge.js	initial commit	2 months ago
kineval_rrt_connect.js	initial commit	2 months ago
kineval_servo_control.js	initial commit	2 months ago
kineval_startingpoint.js	initial commit	2 months ago
kineval_threejs.js	initial commit	2 months ago
kineval_userinput.js	initial commit	2 months ago

Update collision detection with your forward kinematics



Implement RRT-Connect planner



kineval_collision.js

```
kineval.poseIsCollision = function robot_collision_test(q) {  
    // perform collision test of robot geometry against planning world  
  
    // test base origin (not extents) against world boundary extents  
    if ((q[0]<robot_boundary[0][0])||(q[0]>robot_boundary[1][0])||  
        (q[2]<robot_boundary[0][2])||(q[2]>robot_boundary[1][2]))  
        return robot.base;  
  
    // traverse robot kinematics to test each body for collision  
    // STENCIL: implement forward kinematics for collision detection  
    return robot_collision_forward_kinematics(q);  
}
```

input: q (robot configuration)
output: false (for no collision)
or name of link in collision



"electric fence"
world boundary
detection is
provided

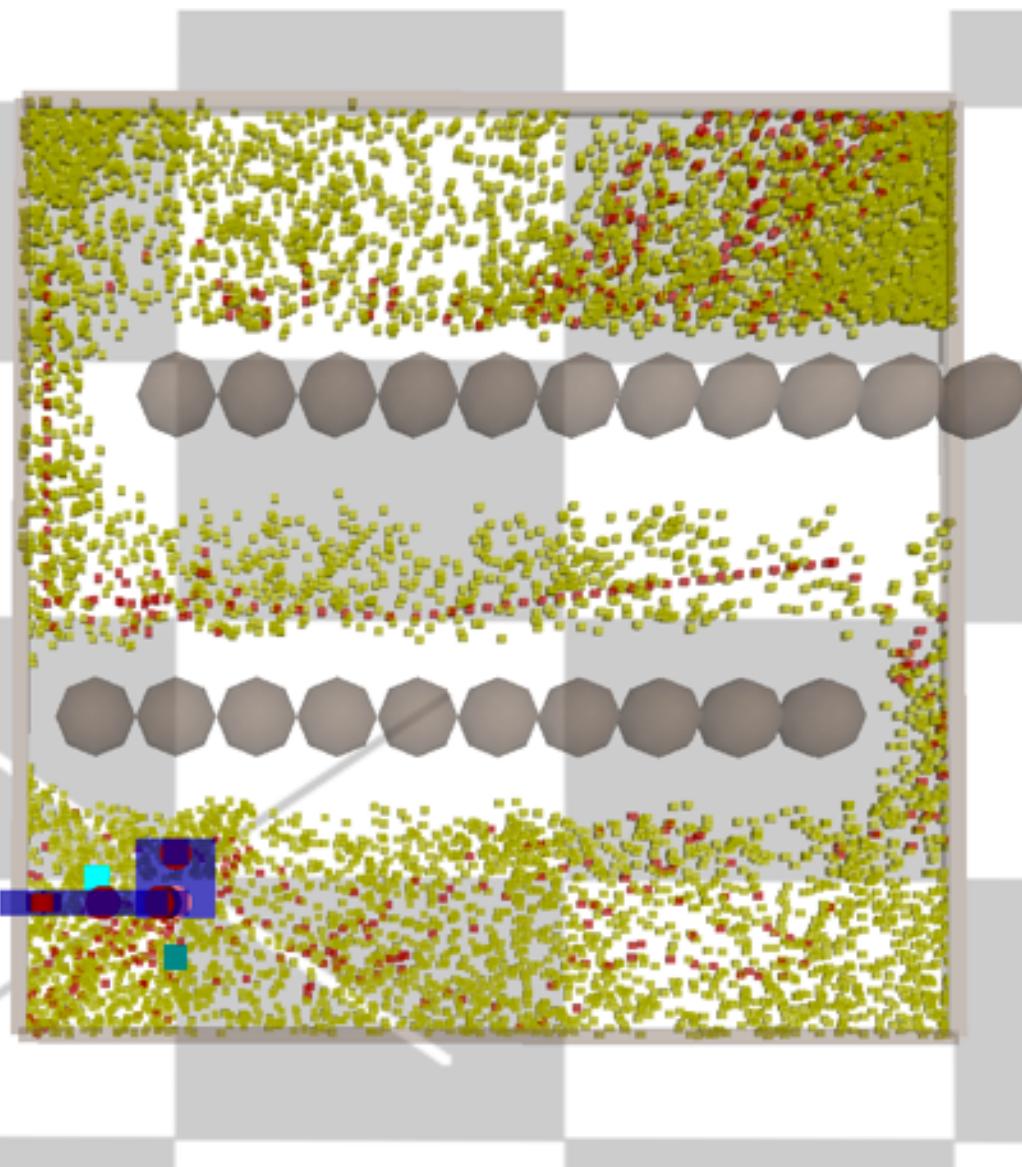
Uncomment this call; and

Implement this function to traverse forward kinematics for collisions;
Use provided link collision function for AABB test of each link

kineval_rrt_connect.js

```
kineval.robotRRTPlannerInit = function robot_rrt_planner_init() {  
    // set start (q_init) and goal (q_goal) configurations  
}  
  
function robot_rrt_planner_iterate() {  
  
    var i;  
    rrt_alg = 1; // 0: basic rrt (OPTIONAL), 1: rrt_connect (REQUIRED)  
  
    if (rrt_iterate && (Date.now()-cur_time > 10)) {  
        cur_time = Date.now();  
  
        // implement one planning iteration here with calls to rrt_extend, etc.  
  
        // if plan found, store configuration sequence in kineval.motion_plan  
        //   kineval.motion_plan[kineval.motion_plan_traversal_index]  
    }  
}
```

make sure to test
against all provided
worlds!



Next Lecture:
Collision Detection

