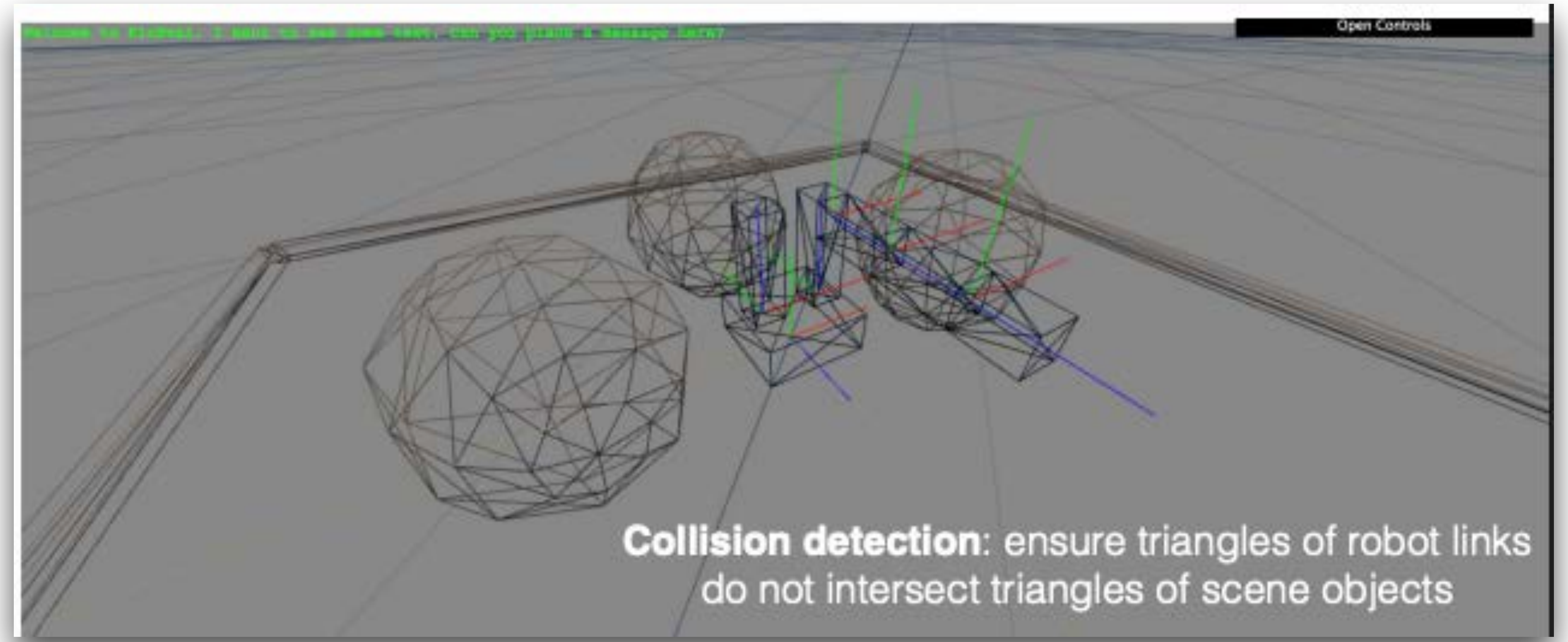
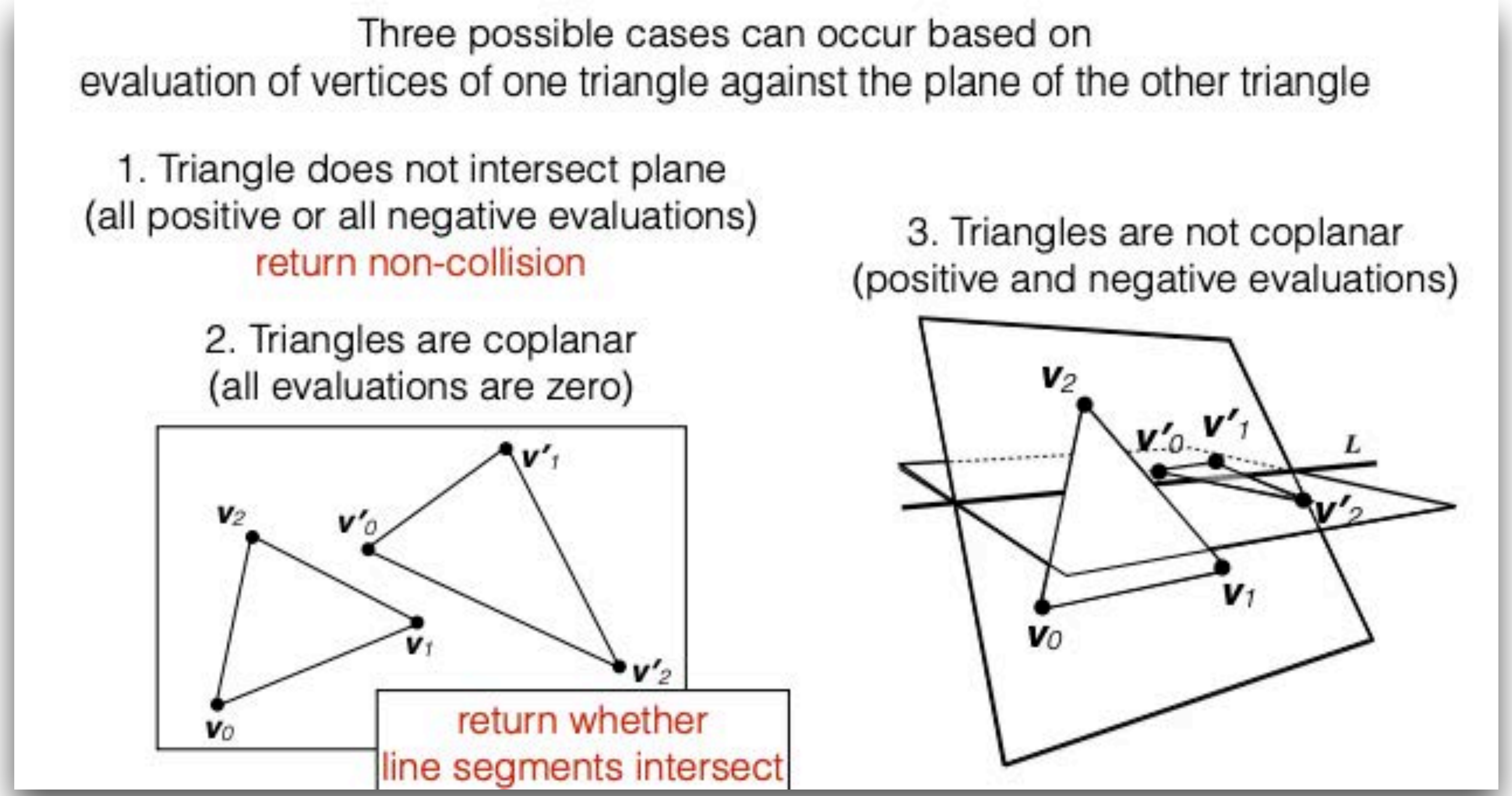
Lecture 14
Planning - VI - Potential Fields

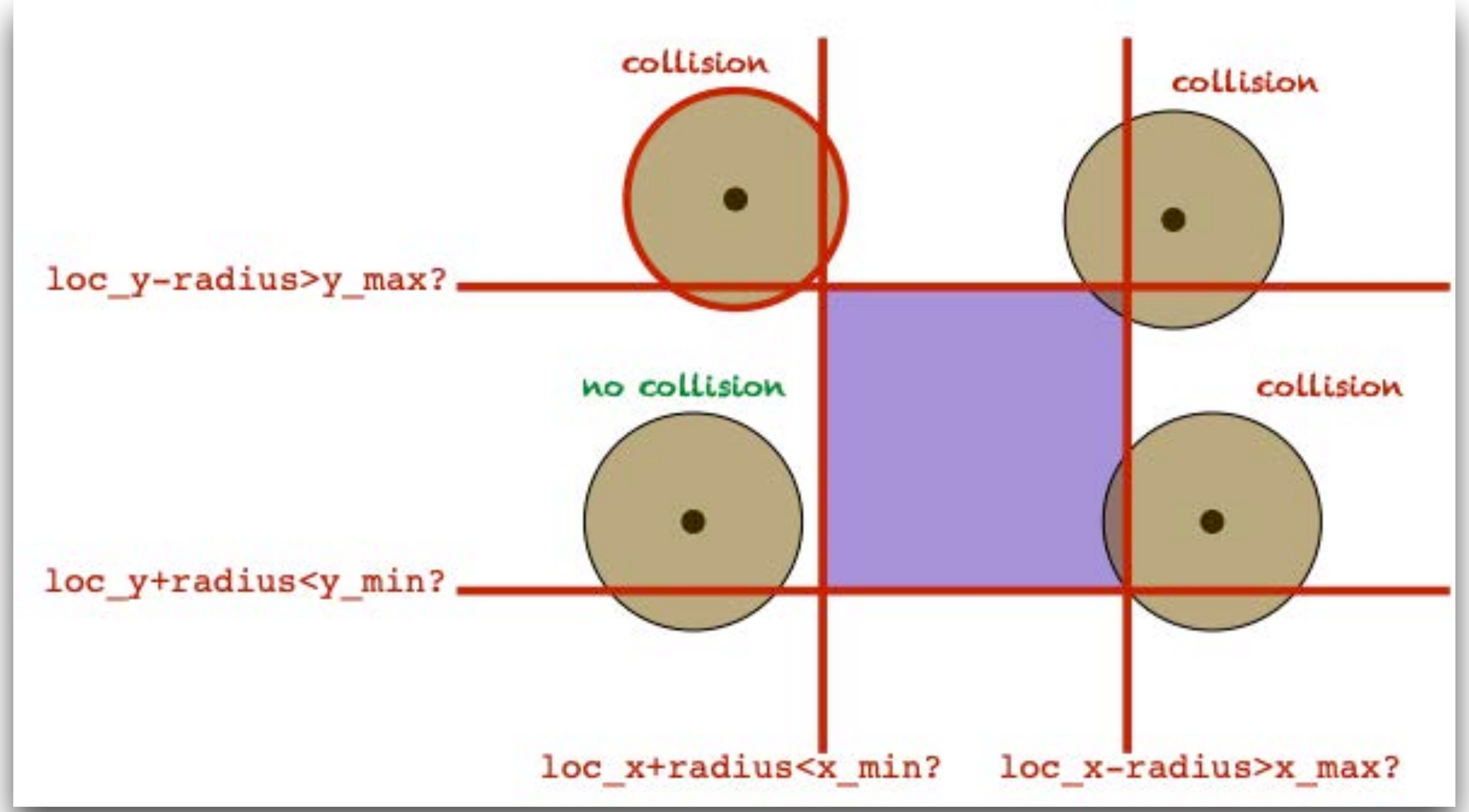
Course Logistics

- Project 5 was posted on 03/05 and is due on 03/24 (**NOTE: this is next Monday**).
- Forming groups for P7 and Final Project
  - We will send a google-form today for students to form groups of 4.
  - This will be due on 03/24 (**NOTE: this is next Monday**).
  - UNITE students who are not attending in-person, will have different group formations (3 or 4). Karthik will reach out to them.
- Project 6 will be posted on 03/24 and will be due on 04/02.
- Quiz 7 will be posted tomorrow at noon and will be due on Wed at noon.
- Final Poster Presentation is planned on **05/05 12:30-2:30 pm**.

Previously


Open Controls
Collision detection: ensure triangles of robot links do not intersect triangles of scene objects


Three possible cases can occur based on
evaluation of vertices of one triangle against the plane of the other triangle
1. Triangle does not intersect plane
(all positive or all negative evaluations)
return non-collision
2. Triangles are coplanar
(all evaluations are zero)
3. Triangles are not coplanar
(positive and negative evaluations)
v2
v1
v0
v'0
v'1
v'2
L
return whether
line segments intersect


collision
collision
no collision
collision
loc_y-radius>y_max?
loc_y+radius<y_min?
loc_x+radius<x_min?
loc_x-radius>x_max?

RRT Algorithm

RRT Algorithm

Extend graph towards a random configuration and repeat

BUILD_RRT(q_{init})

$\mathcal{T}$.init(q_{init});

for $k = 1$ **to** K **do**

    $q_{rand} \leftarrow$ RANDOM_CONFIG();

    EXTEND($\mathcal{T}, q_{rand}$);

Return $\mathcal{T}$

[Kuffner, LaValle 2000]

RRT Algorithm

Extend graph towards a random configuration and repeat

BUILD_RRT(q_{init})

$\mathcal{T}$.init(q_{init});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$ RANDOM_CONFIG();
    EXTEND($\mathcal{T}, q_{rand}$);
Return $\mathcal{T}$

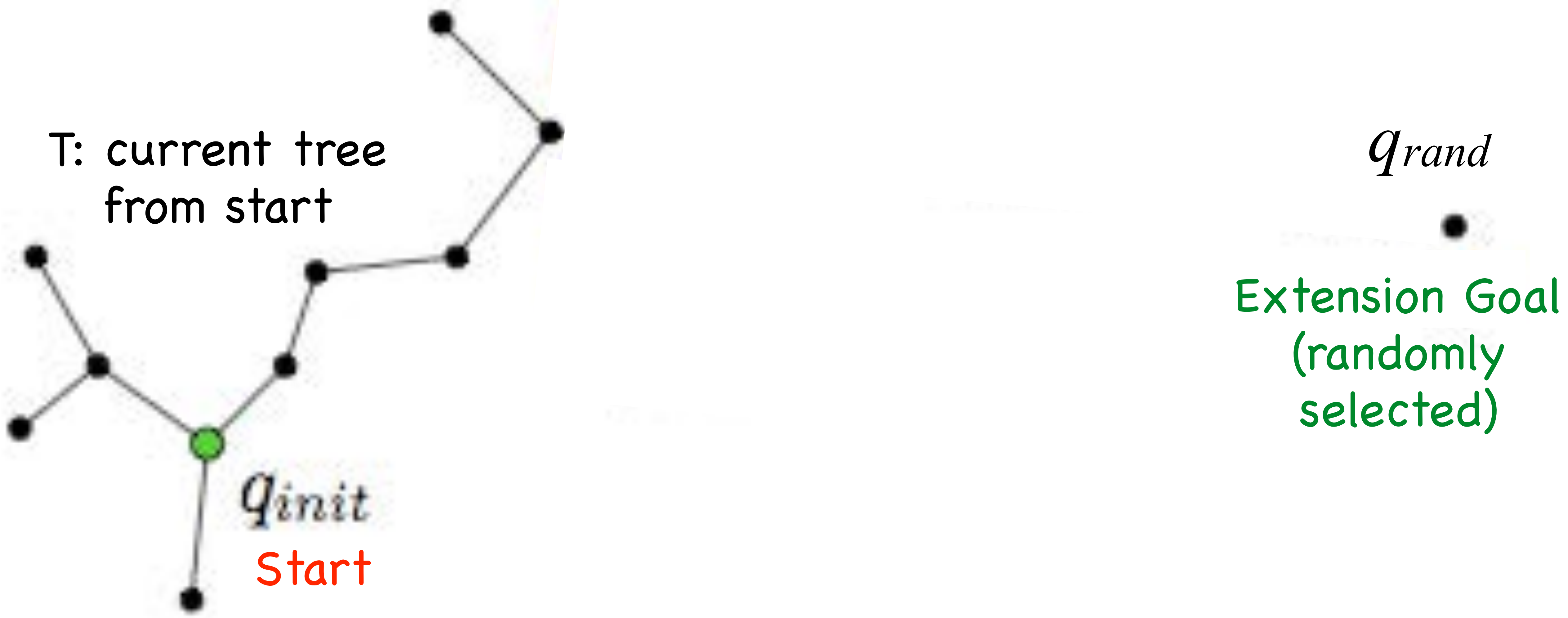


Figure 3: The EXTEND operation.

RRT Algorithm

Extend graph towards a random configuration and repeat

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

```
EXTEND(T, q)
  q_near ← NEAREST_NEIGHBOR(q, T);
  if NEW_CONFIG(q, q_near, q_new) then
      T.add_vertex(q_new);
      T.add_edge(q_near, q_new);
      if q_new = q then
          Return Reached;
      else
          Return Advanced;
  Return Trapped;
```

Extend graph towards a random configuration

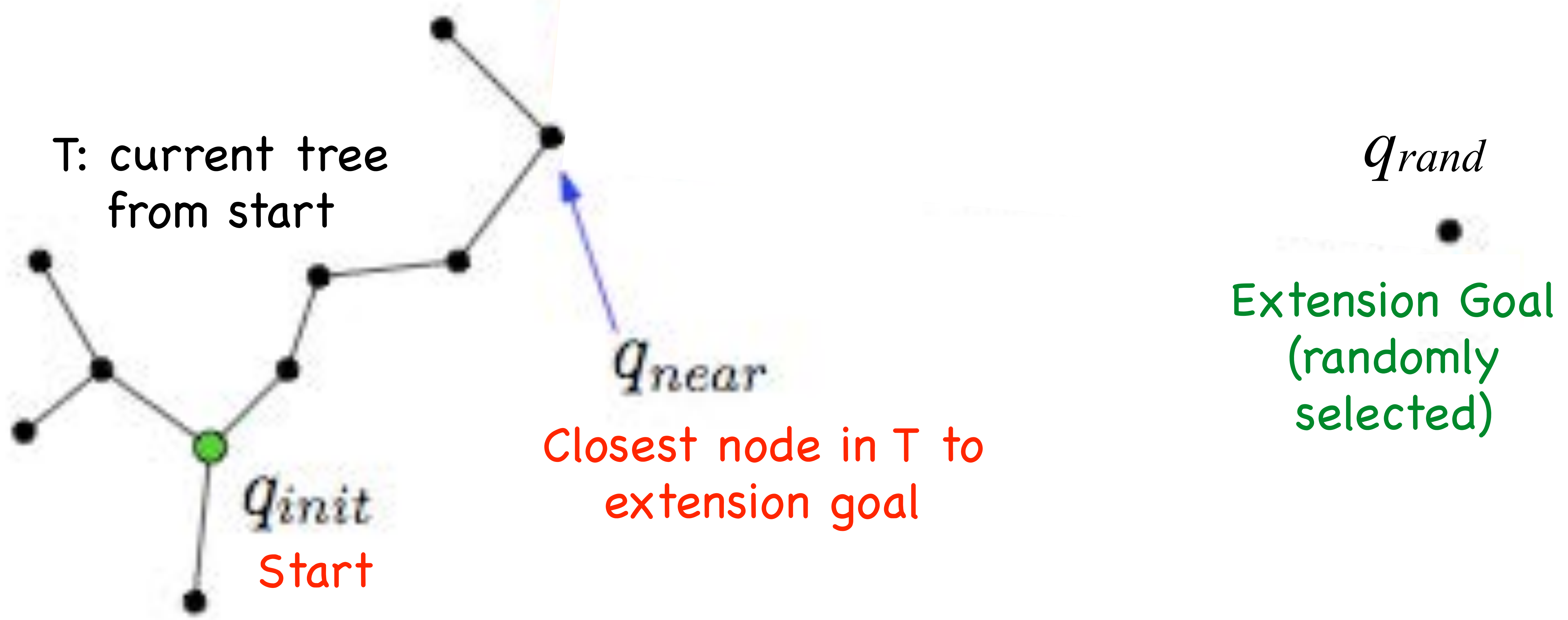


Figure 3: The EXTEND operation.

[Kuffner, LaValle 2000]

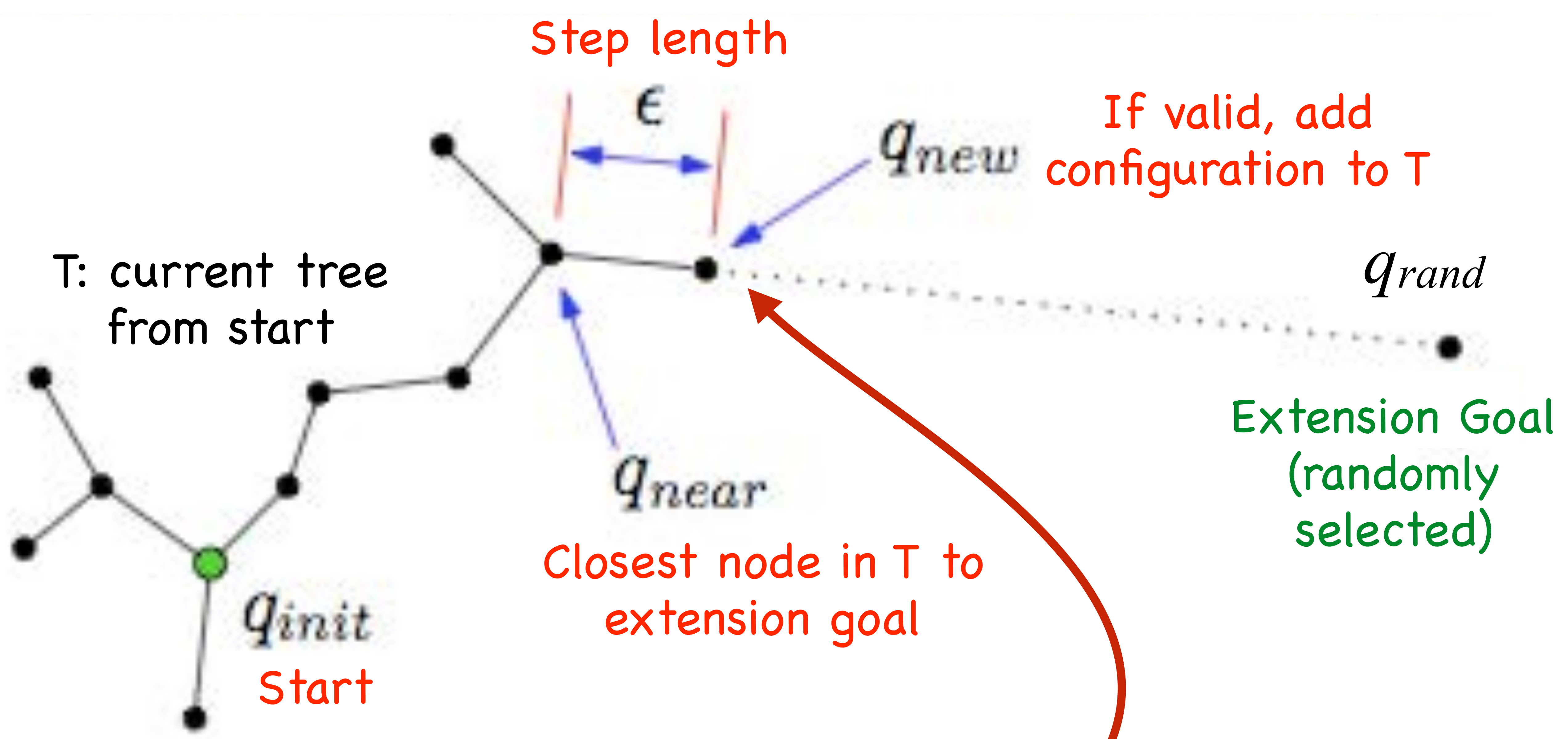
RRT Algorithm

Extend graph towards a random configuration and repeat

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

```
EXTEND(T, q)
  q_near ← NEAREST_NEIGHBOR(q, T);
  if NEW_CONFIG(q, q_near, q_new) then
      T.add_vertex(q_new);
      T.add_edge(q_near, q_new);
      if q_new = q then
          Return Reached;
      else
          Return Advanced;
  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* Algorithm

RRT*

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\};\ E \leftarrow \emptyset;$

for $i = 1, \ldots, n$ **do**

- $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i;$
- $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}});$
- $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})\ ;$
- **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  - $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})\ ;$
  - $V \leftarrow V \cup \{x_{\text{new}}\};$
  - $x_{\text{min}} \leftarrow x_{\text{nearest}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}));$
  - **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
    - **if** $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
      - $x_{\text{min}} \leftarrow x_{\text{near}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
  - $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$
  - **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
    - **if** $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}});$
    - $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$

return $G = (V, E);$

Karaman & Frazzo

RRT*

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$

return $G = (V, E)$;

FIND x_{new}

FIND neighbors to x_{new} in G
ADD x_{new} to G

FIND edge to x_{new} from neighbors with least cost
ADD that to G

REWIRE the edges in the neighborhood if any least cost path exists from the root to the neighbors via x_{new}

Karaman & Frazzo

RRT*

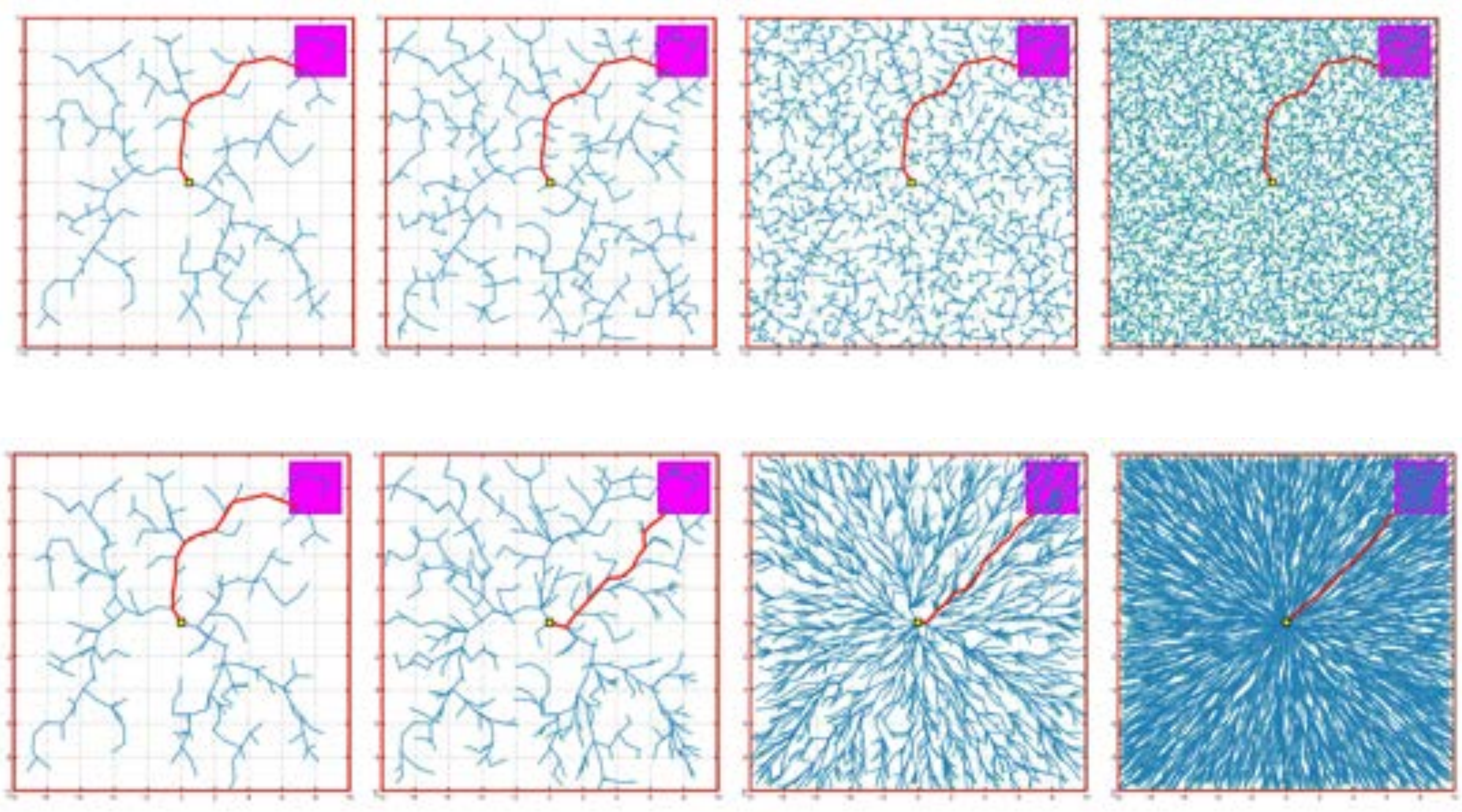
- Asymptotically optimal
- Main idea:
  - Swap new point in as parent for nearby vertices who can be reached along shorter path through new point than through their original (current) parent

Demonstration - https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

RRT*

RRT

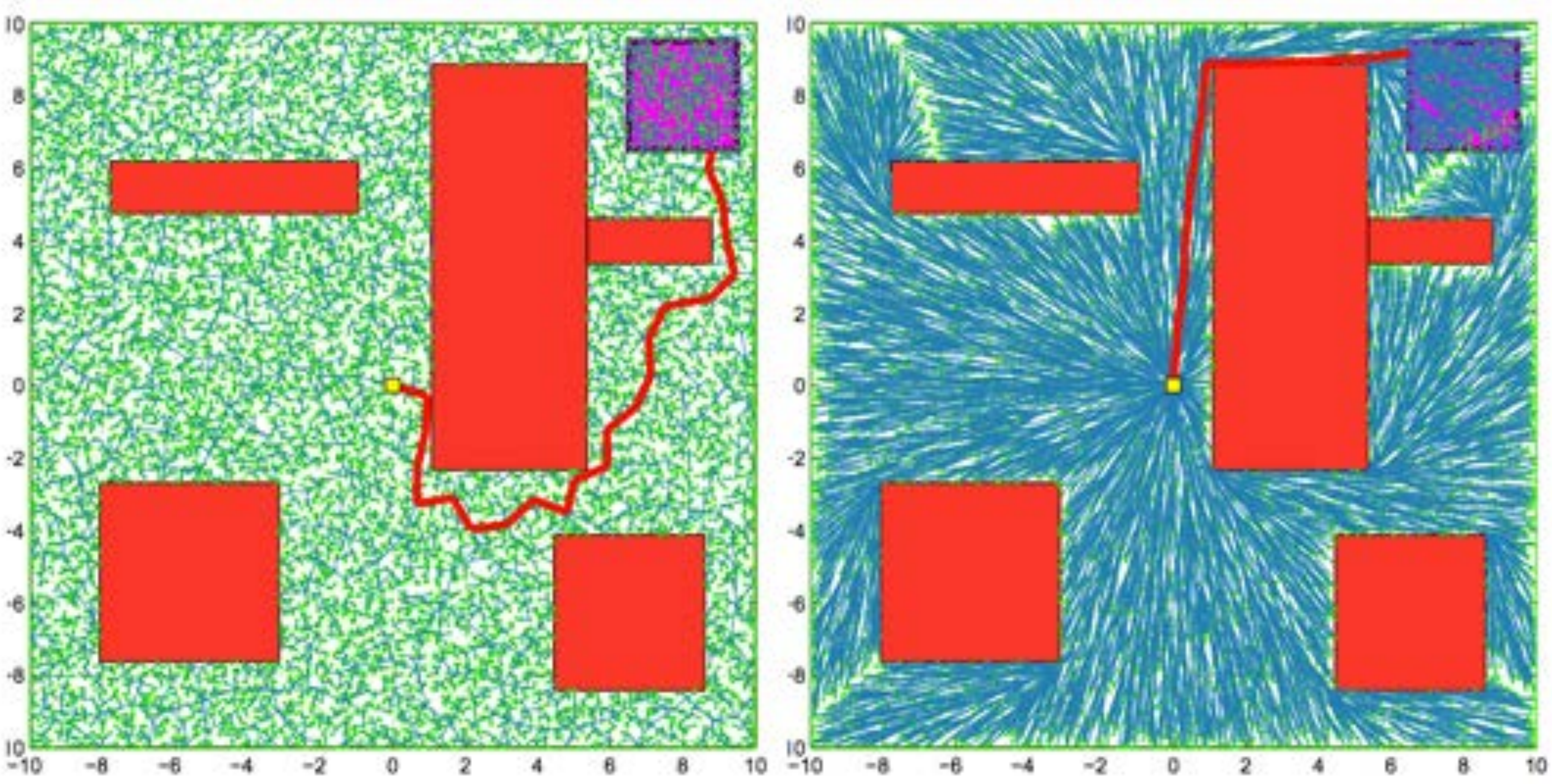
RRT*



Source: Karaman and Frazzoli

RRT*

RRT RRT*



Source: Karaman and Frazzoli

Slide borrowed from Dieter Fox

Smoothing

Randomized motion planners tend to find not so great paths for execution: very jagged, often much longer than necessary.

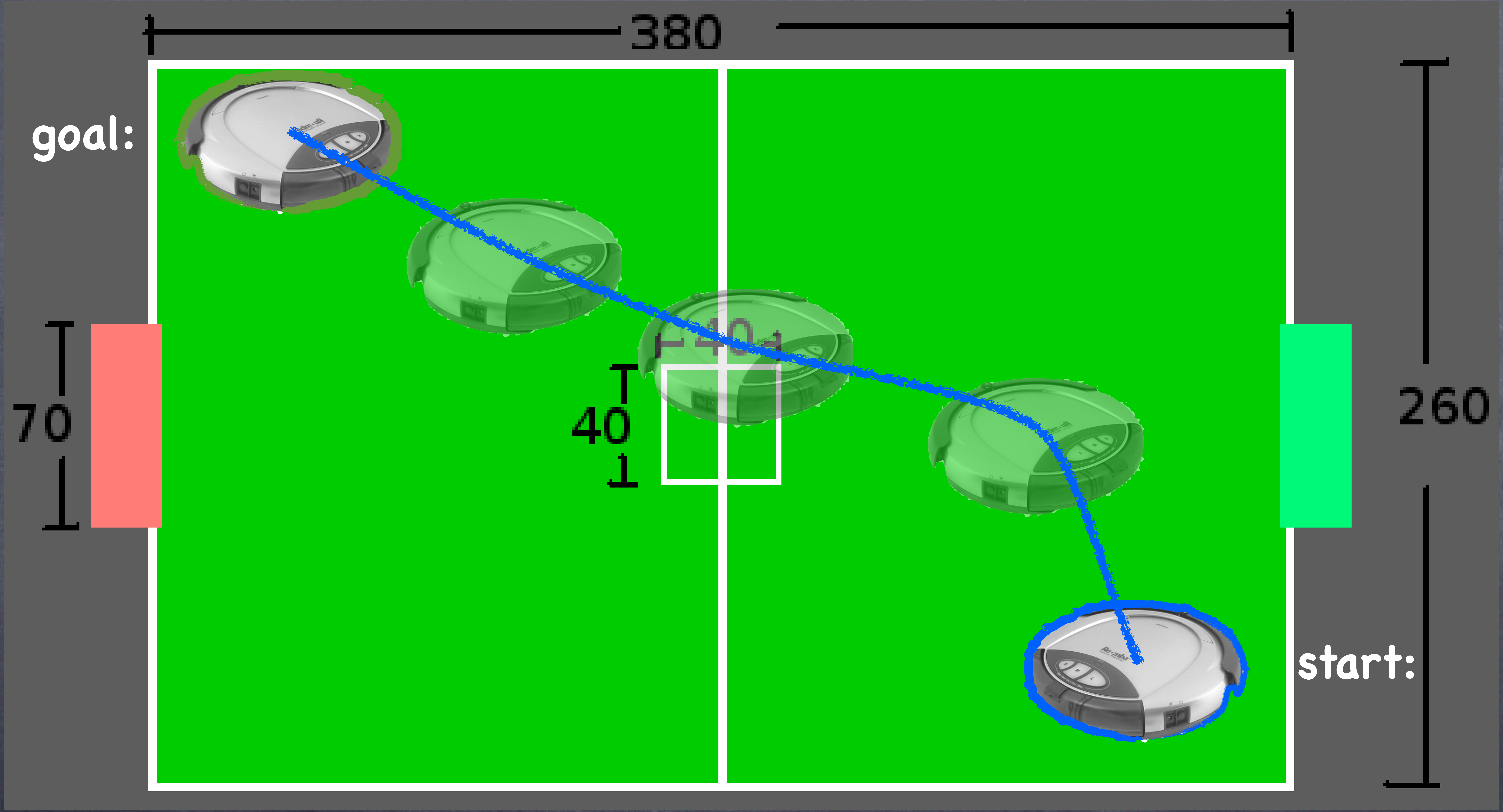
→ In practice: do smoothing before using the path

- Shortcutting:
  - along the found path, pick two vertices x_{t1}, x_{t2} and try to connect them directly (skipping over all intermediate vertices)
- Nonlinear optimization for optimal control
  - Allows to specify an objective function that includes smoothness in state, control, small control inputs, etc.

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**

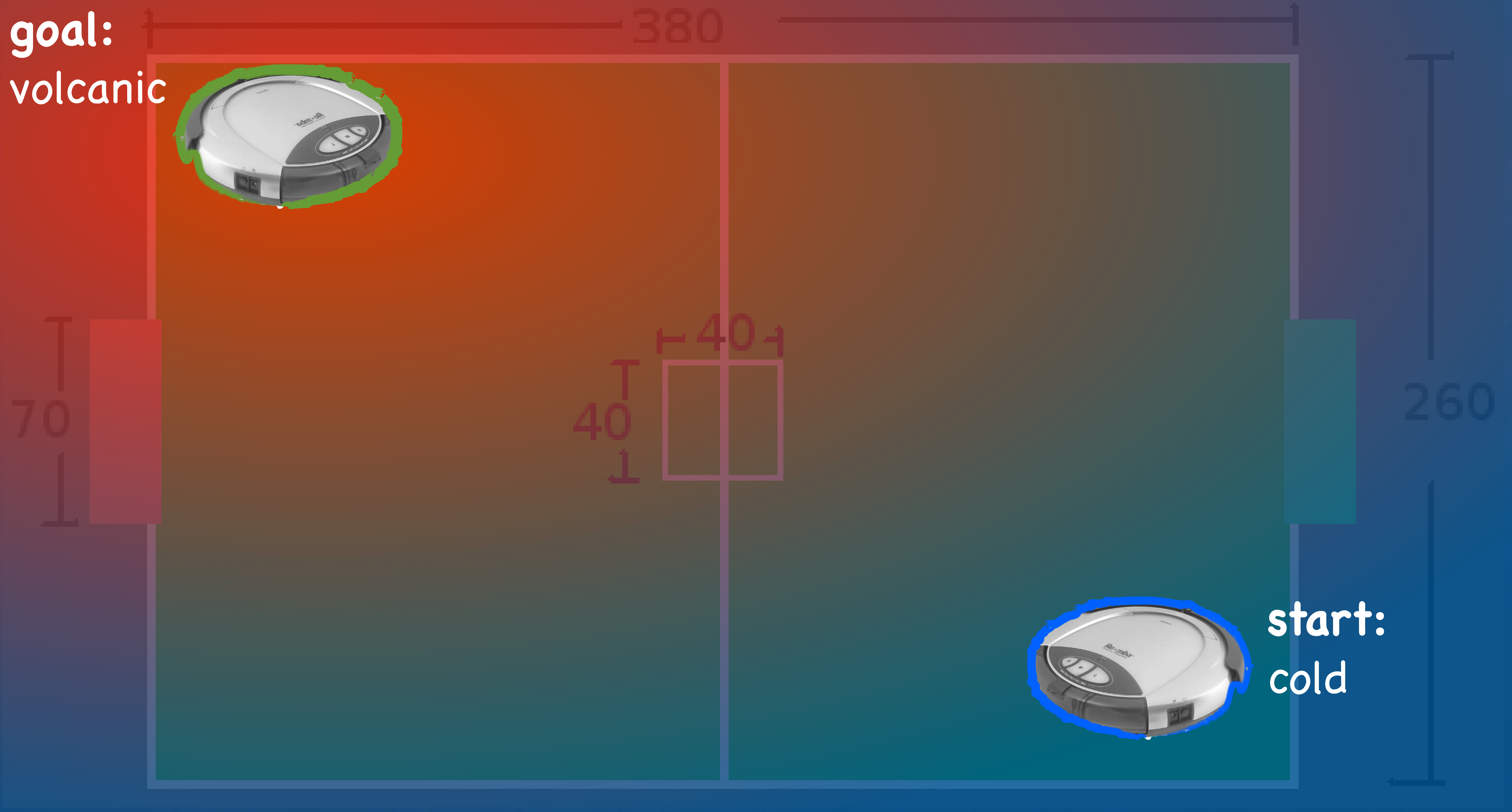
Navigation (again)



Slide borrowed from Michigan Robotics autorob.org

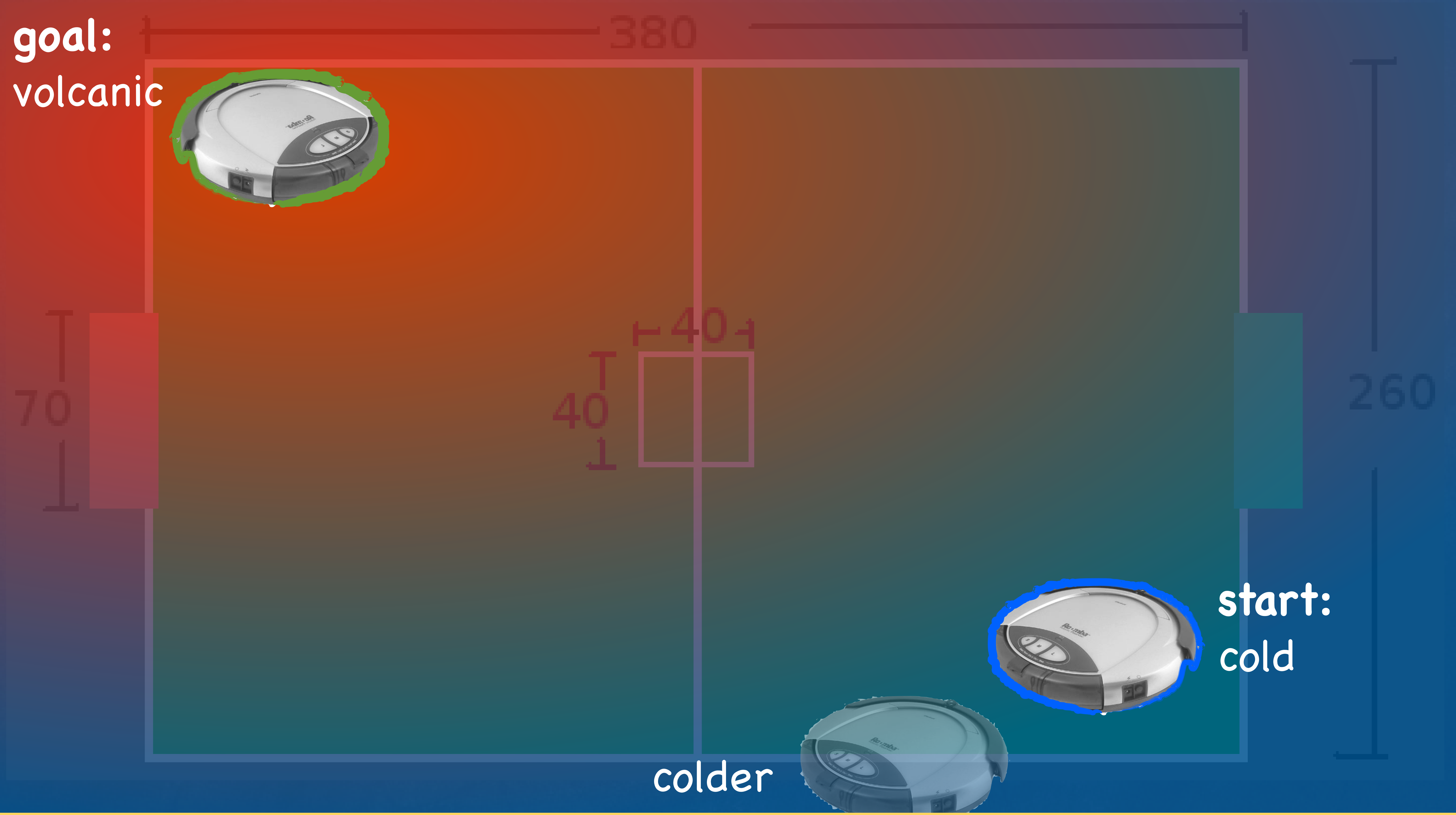
Potential field

(like a game of "warmer-colder")



Potential field

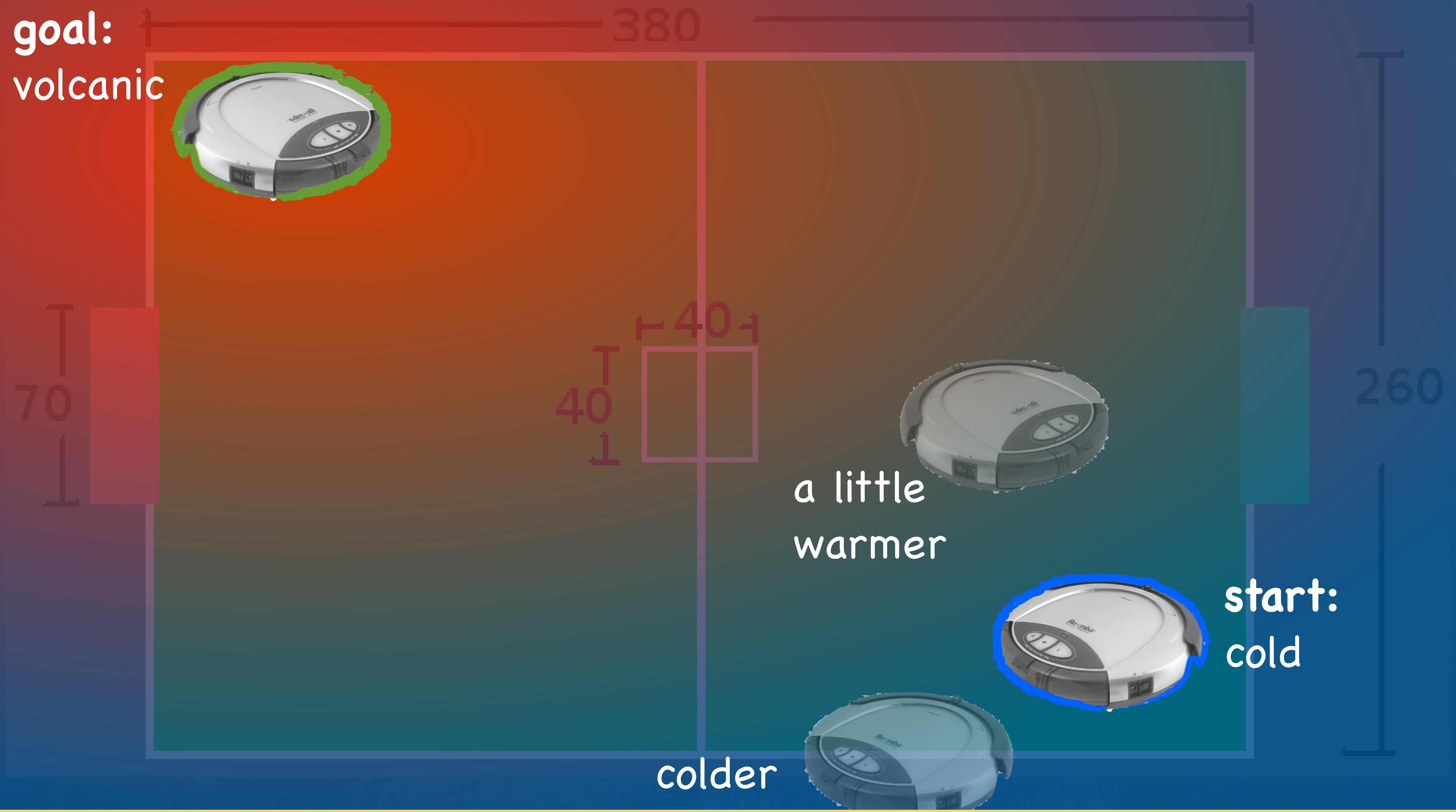
(like a game of "warmer-colder")



Slide borrowed from Michigan Robotics autorob.org

Potential field

(like a game of "warmer-colder")

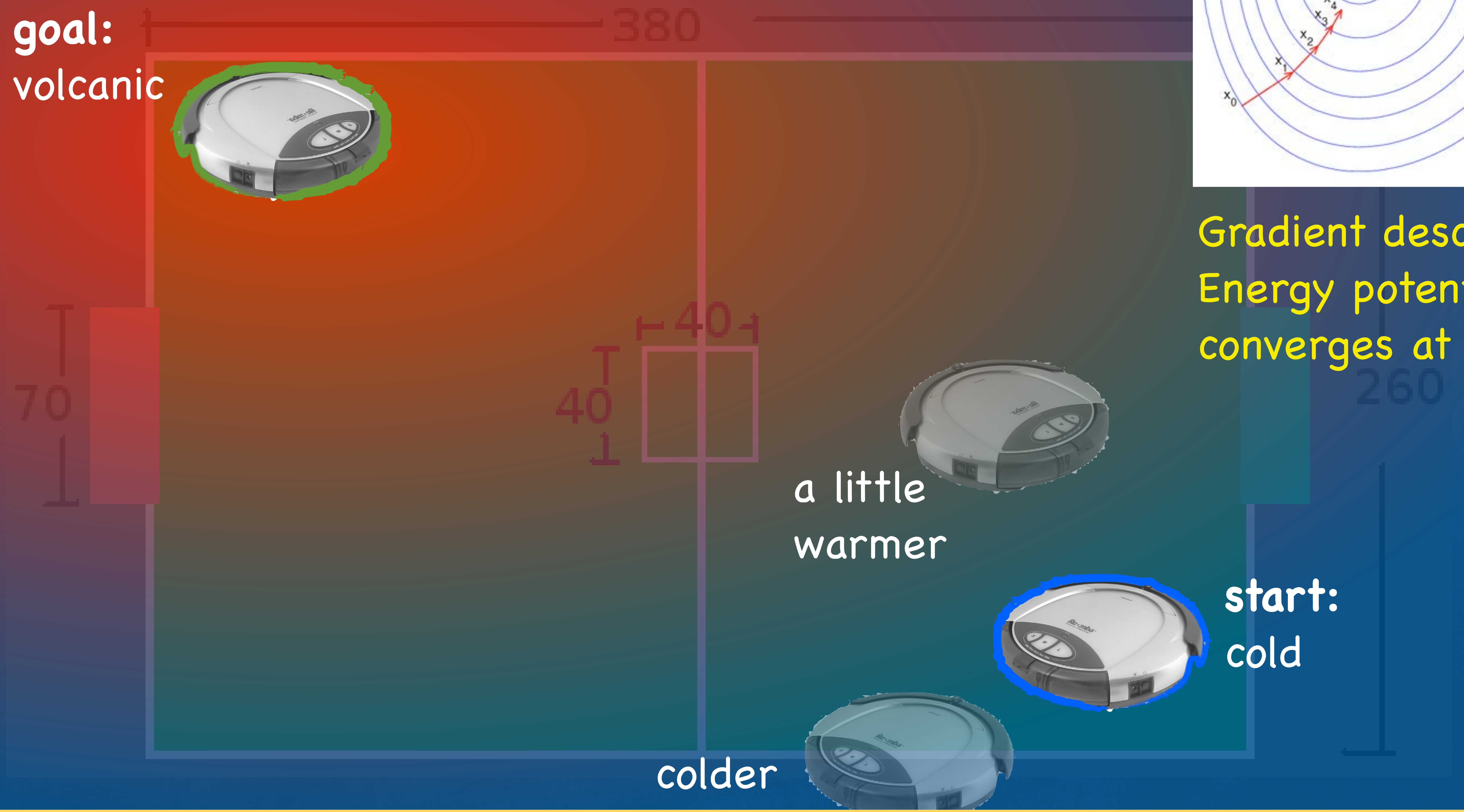


Slide borrowed from Michigan Robotics autorob.org

Potential field

(like a game of "warmer-colder")

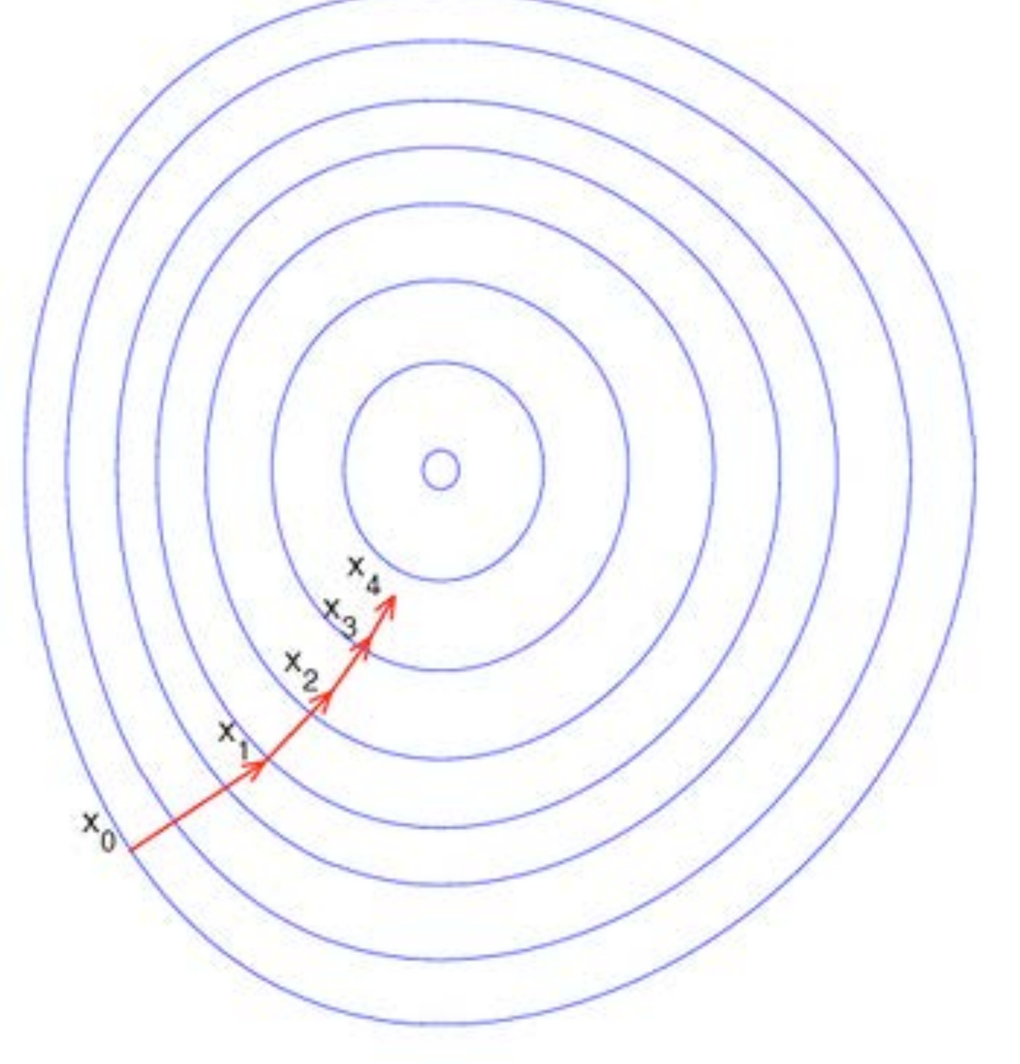
goal:
volcanic



a little
warmer

start:
cold

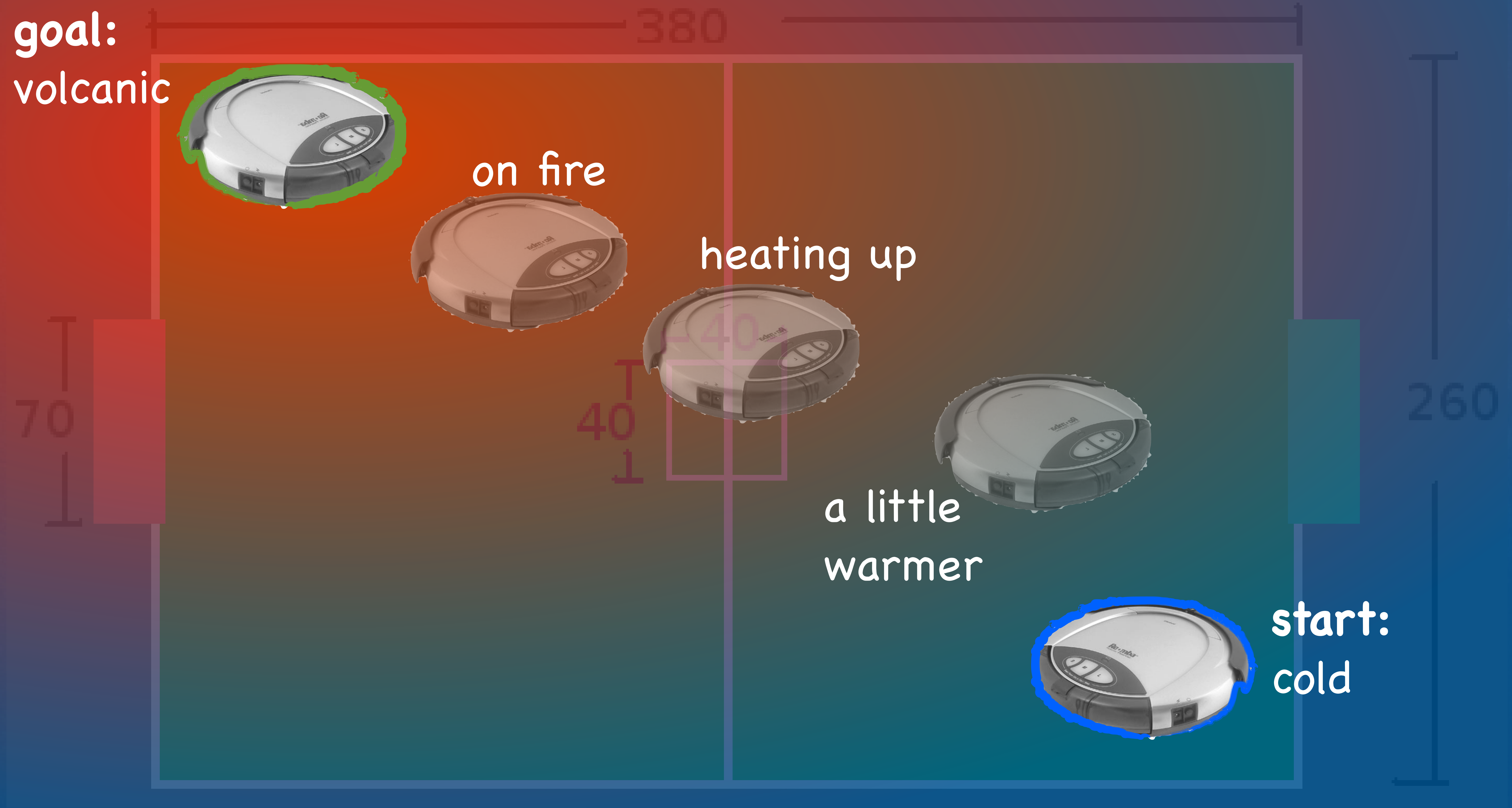
colder



Gradient descent:
Energy potential
converges at goal

Potential field

(like a game of "warmer-colder")



Slide borrowed from Michigan Robotics autorob.org

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 Algorithm:

$q_{path}[0] \leftarrow q_{start}$

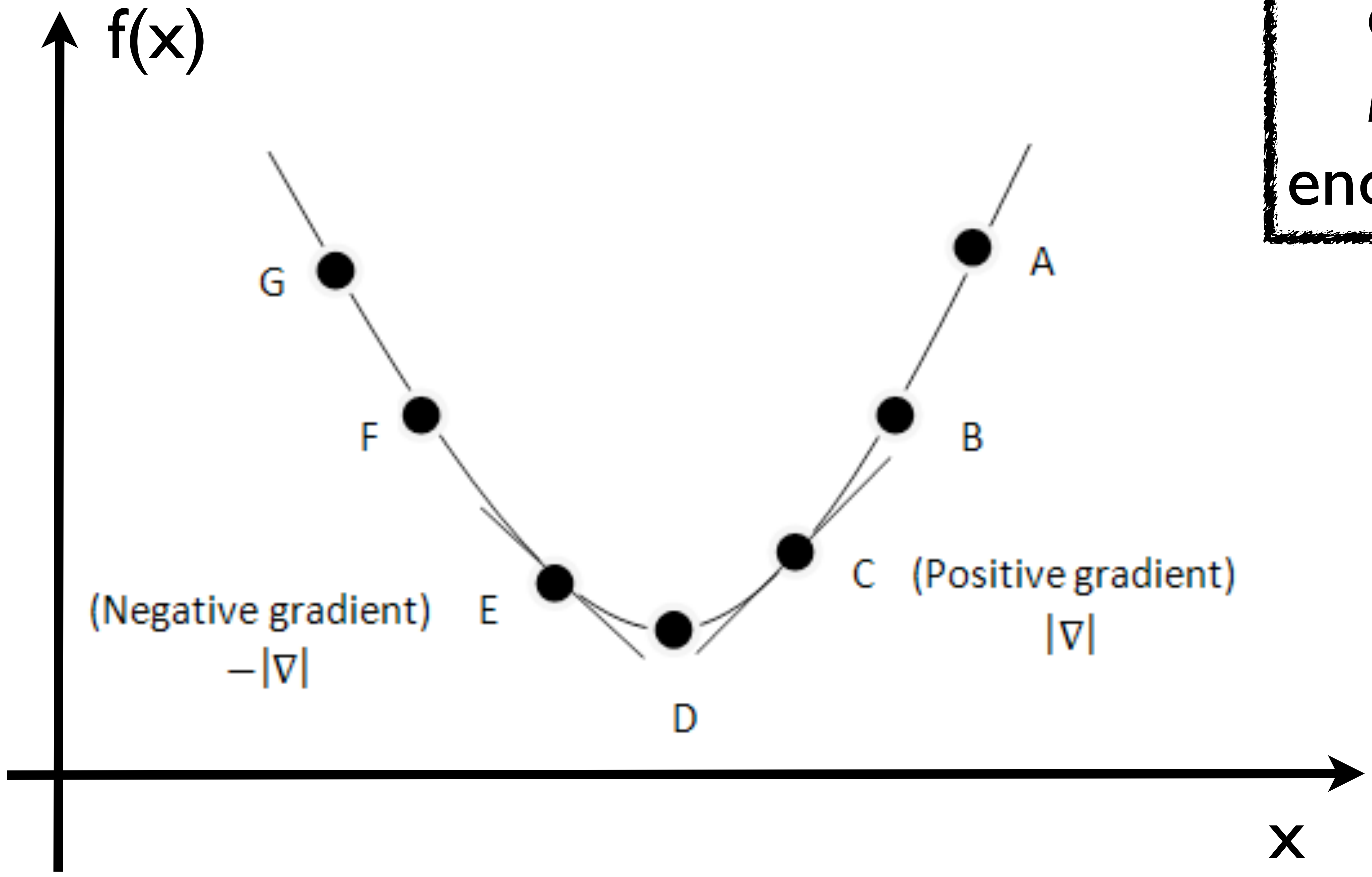
$i \leftarrow 0$

while $(||\nabla U(q[i])|| > \varepsilon)$

  $q_{path}[i+1] \leftarrow q_{path}[i] - \alpha \nabla U(q_{path}[i])$

  $i \leftarrow i+1$

end

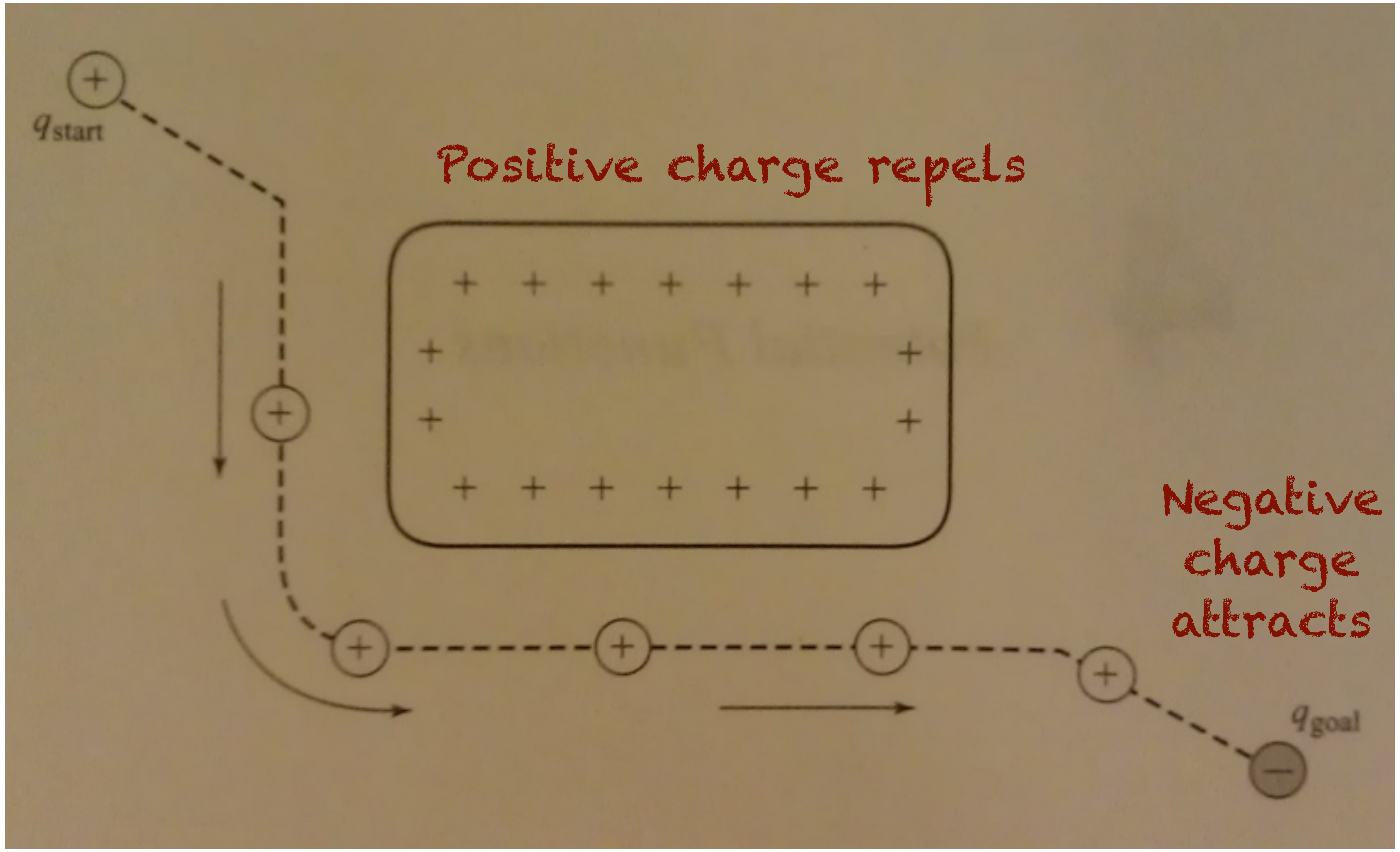


Derivative assumed to be direction of steepest ascent away from goal

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

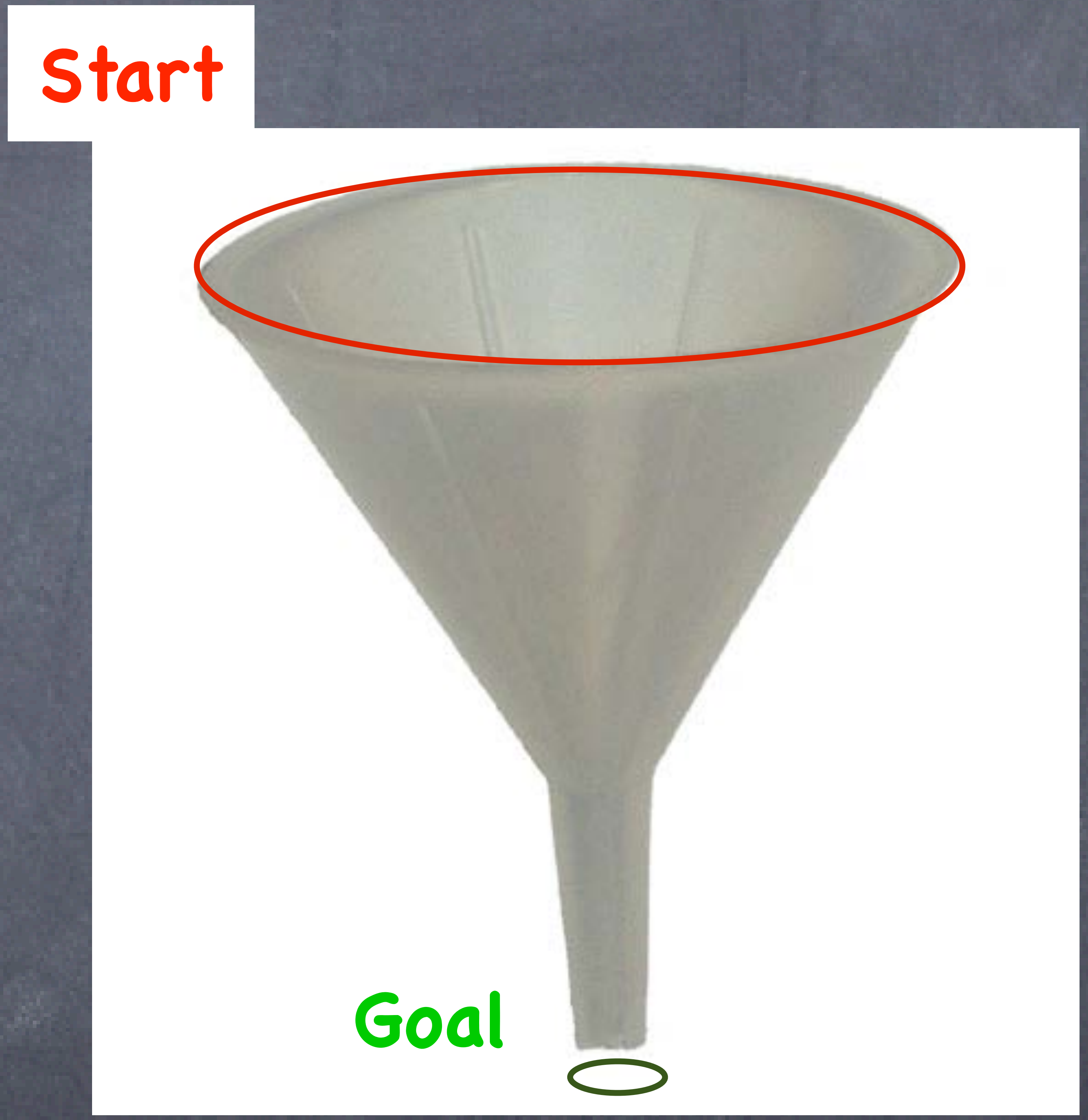
Charged Particle Example

Positively charged particle follows potential energy to goal



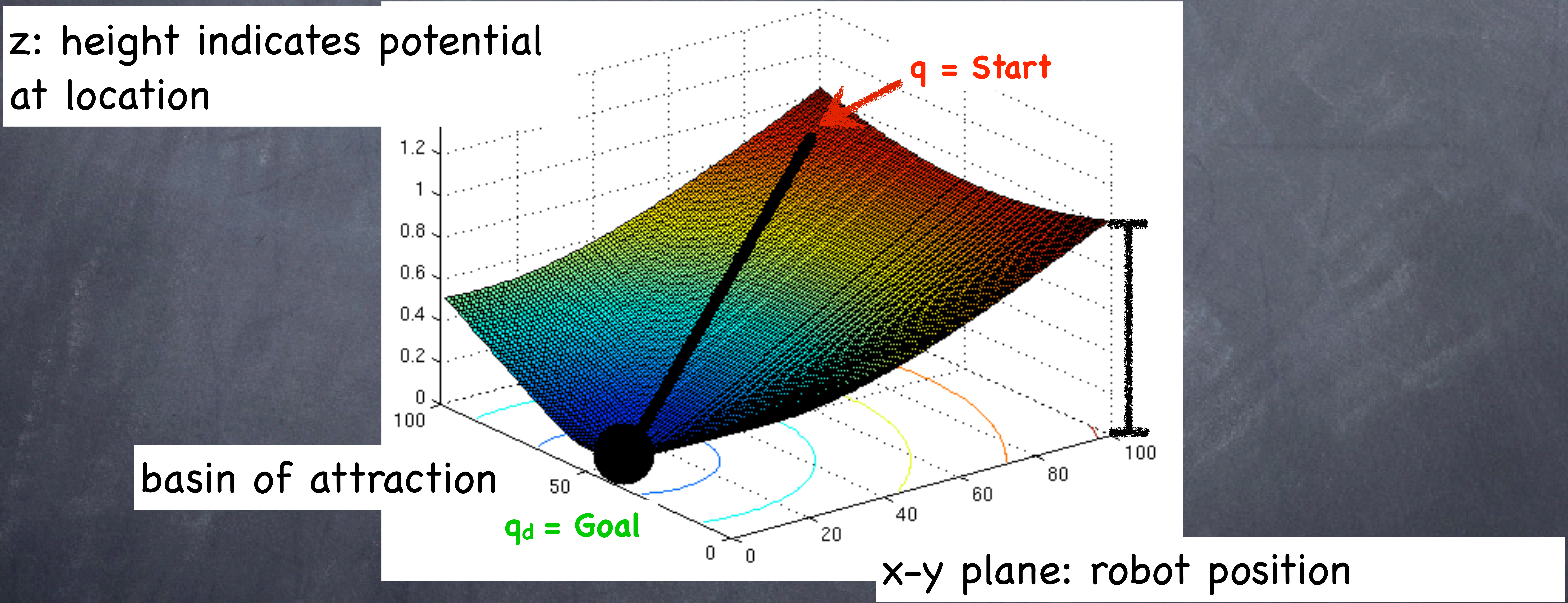
Convergent Potentials

let's call these "attractor landscapes"



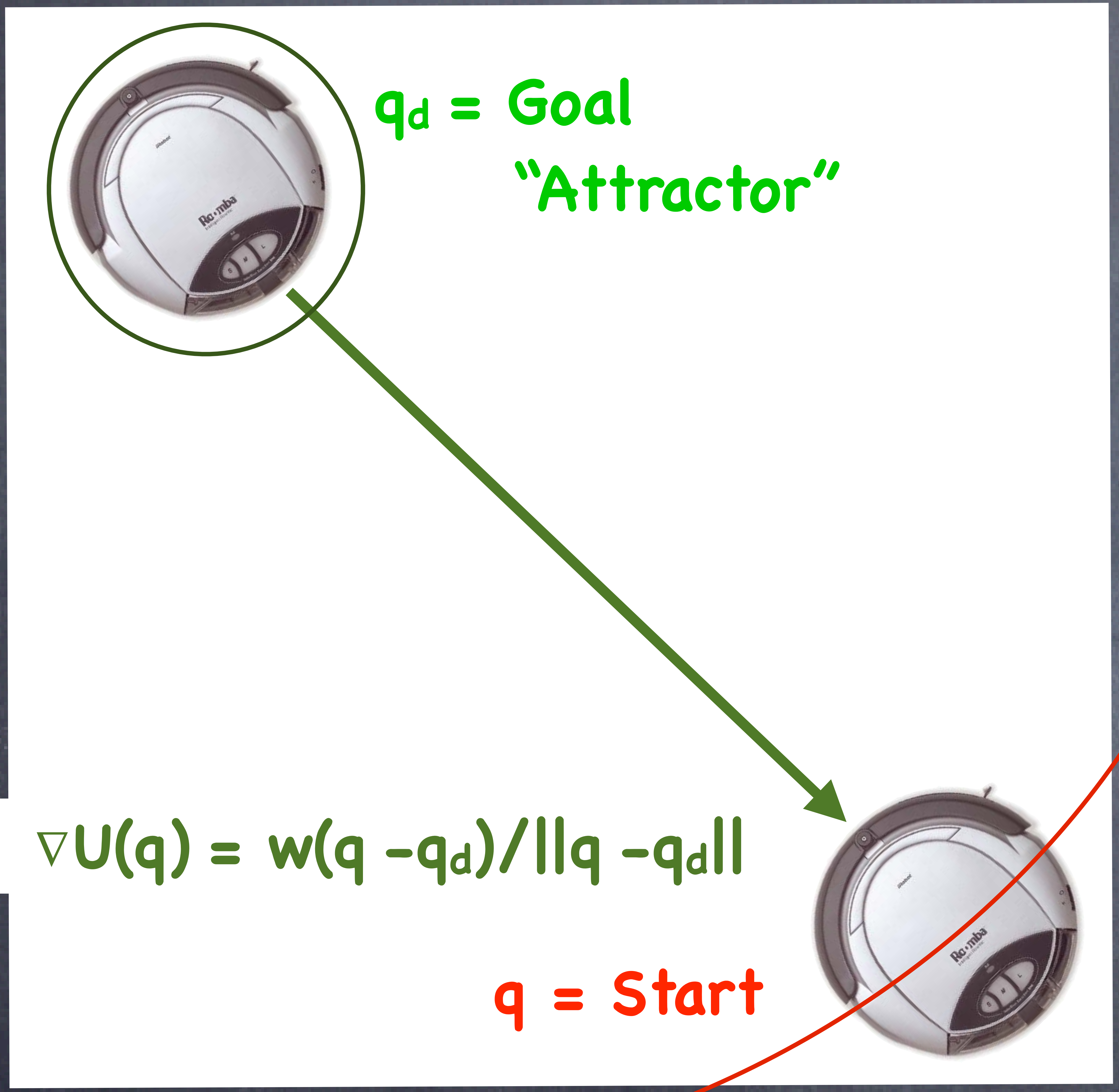
basin of attraction

2D potential navigation



“Cone” Attractor

top view



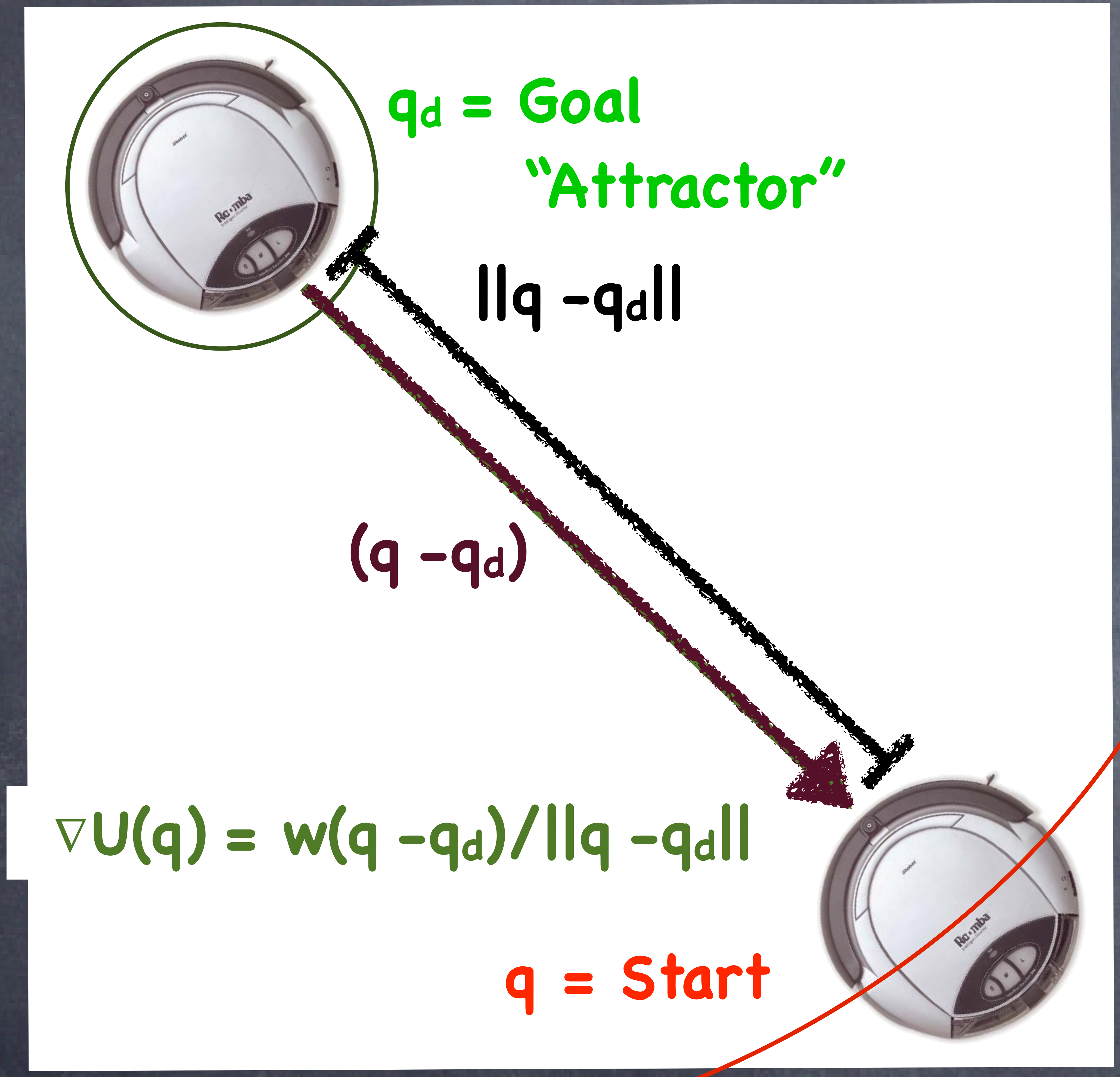
$$\nabla U(q) = w(q - q_d)/||q - q_d||$$

w: weight
$(q - q_d)$: direction
$||q - q_d||$: distance

side view

"Cone" Attractor

top view



w: weight

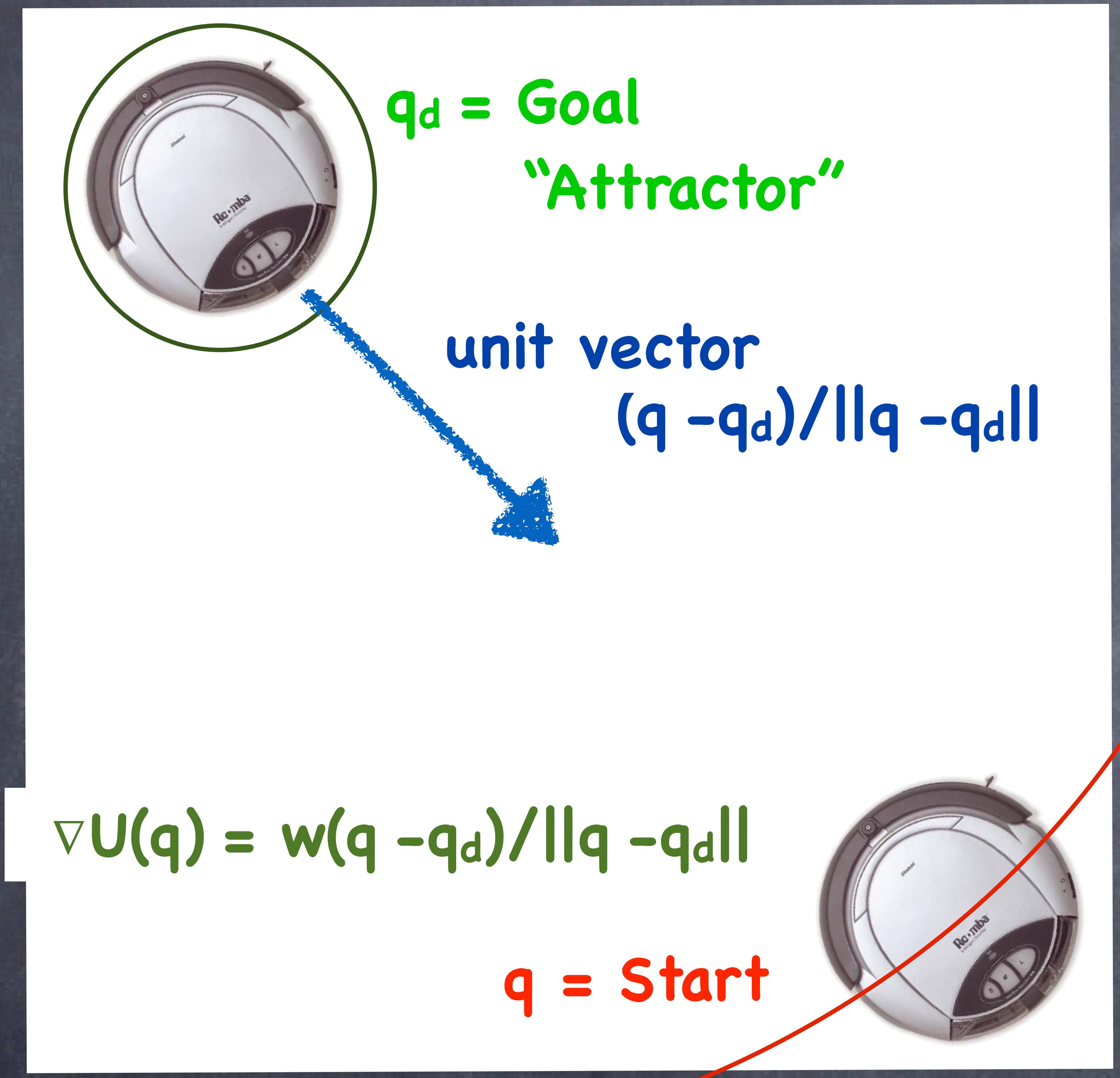
$(q - q_d)$: direction

$\|q - q_d\|$: distance

side view

“Cone” Attractor

top view

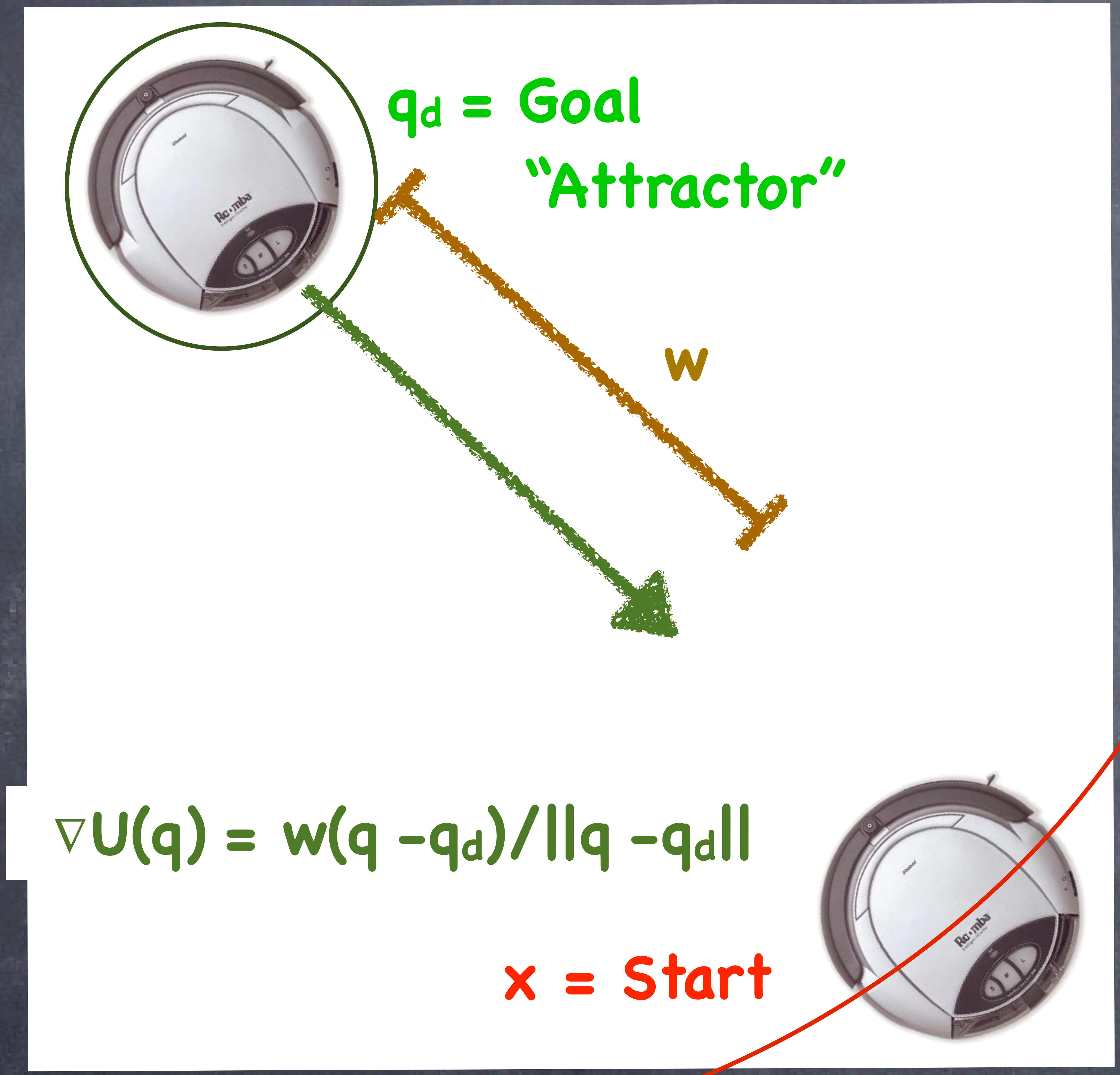


w: weight
$(q_d - q)$: direction
$\|q_d - q\|$: distance

side view

"Cone" Attractor

top view



w: weight (< 1)
$(q - q_d)$: direction
$||q - q_d||$: distance

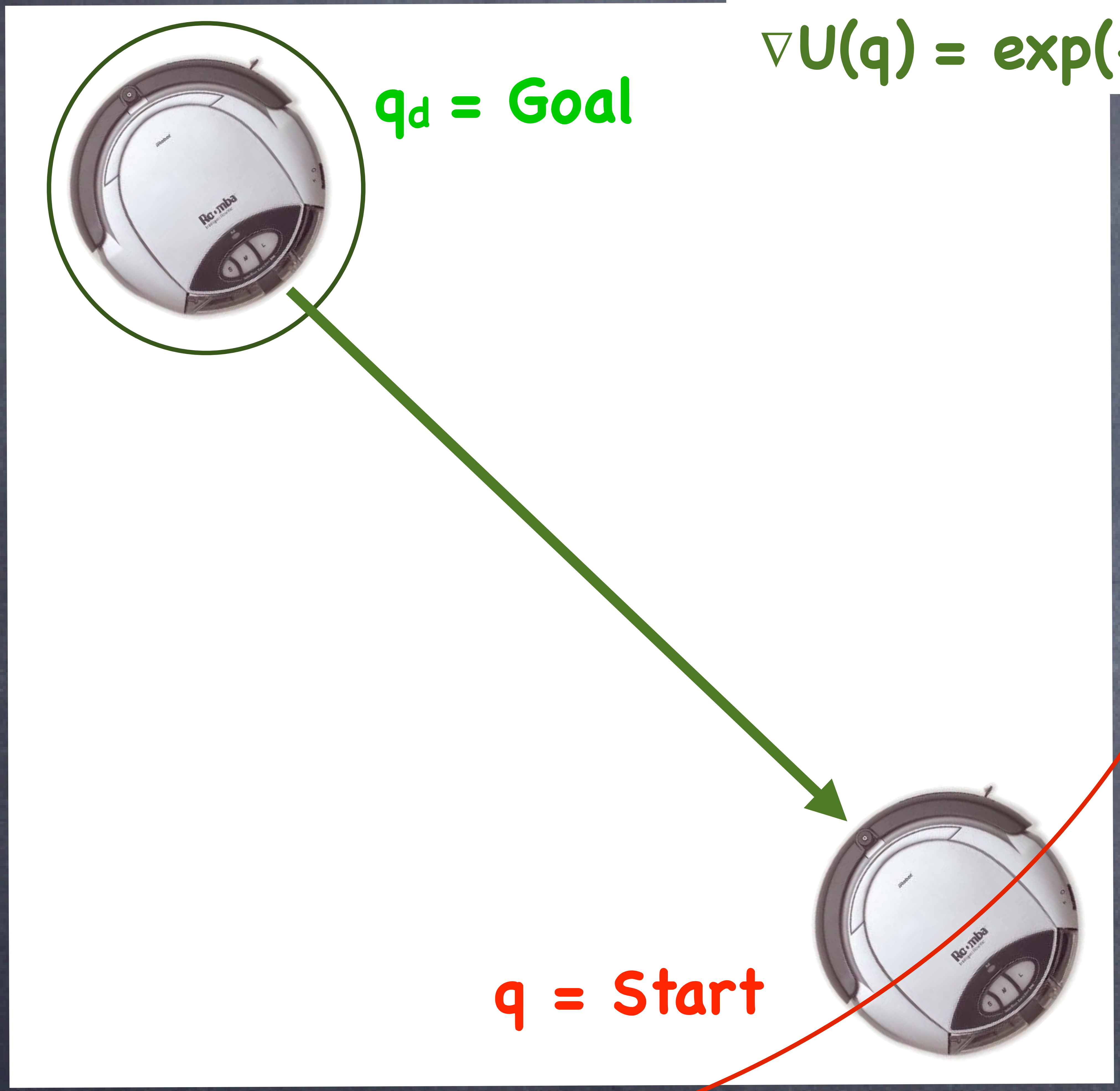
side view

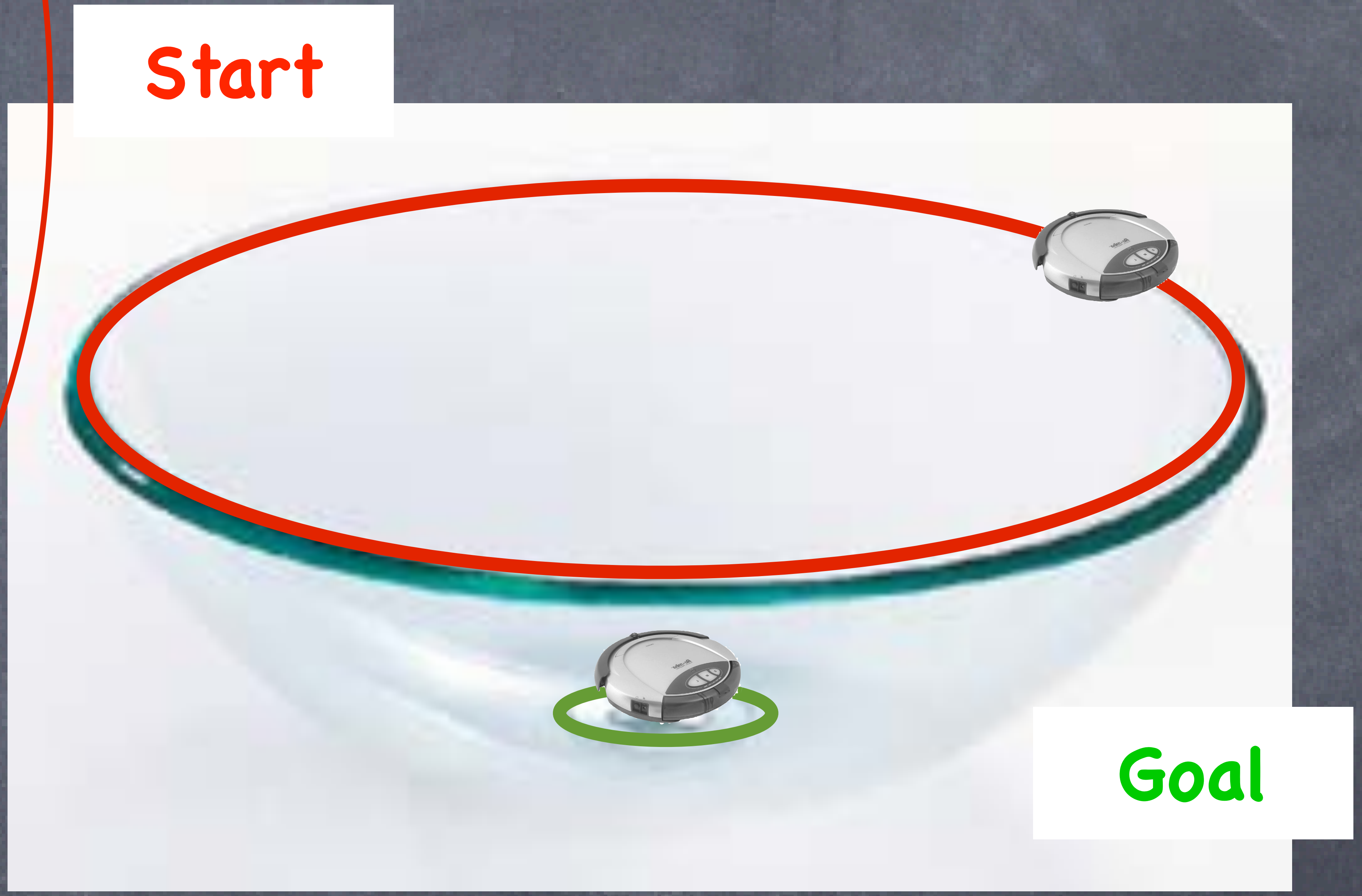
Can we modulate the range of a potential field?

"Bowl" Attractor

$$\nabla U(q) = \exp(-\|q - q_d\|/w)\,(q - q_d)$$

top view



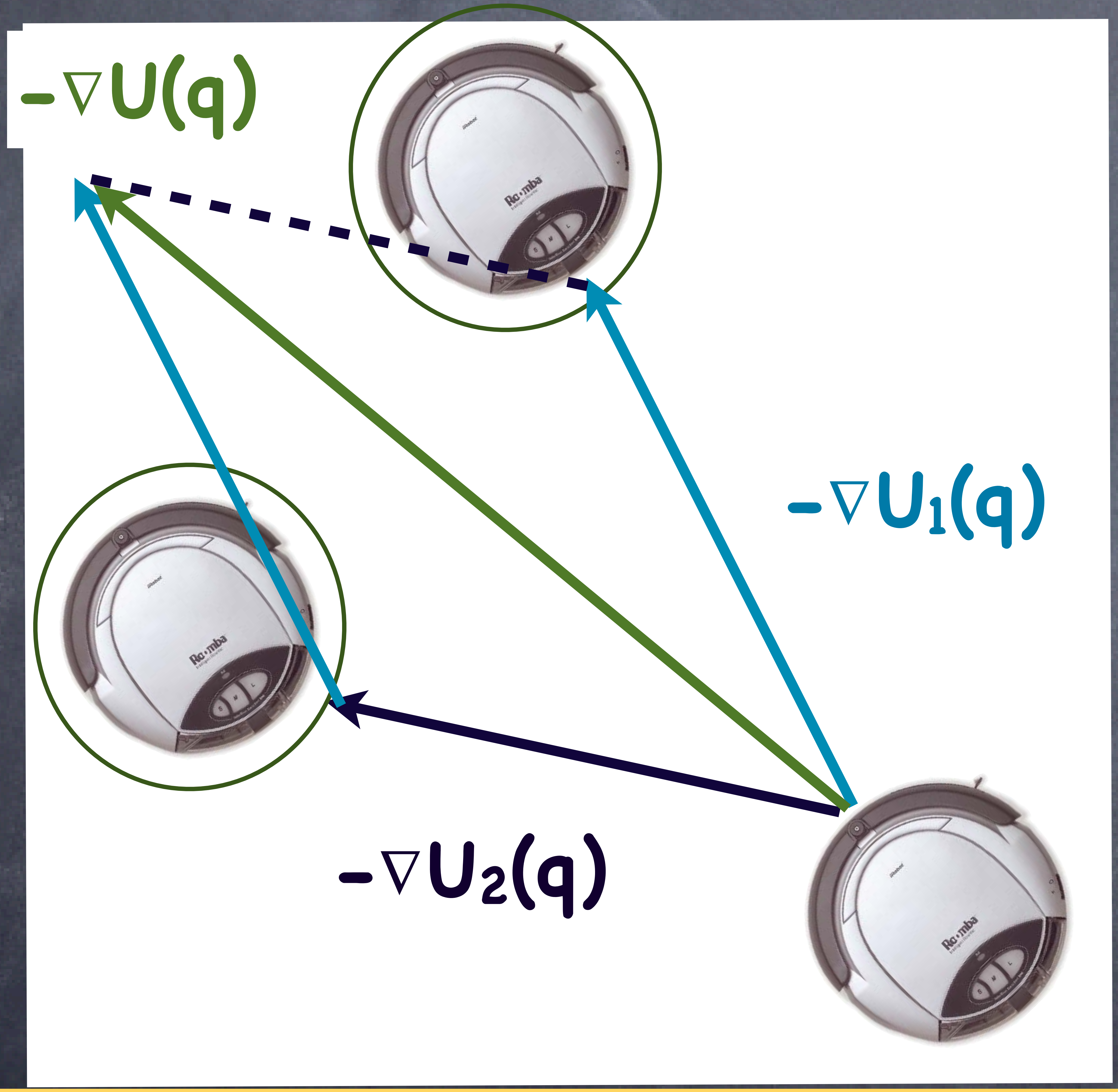


side view

Slide borrowed from Michigan Robotics autorob.org

Can we combine multiple potentials?

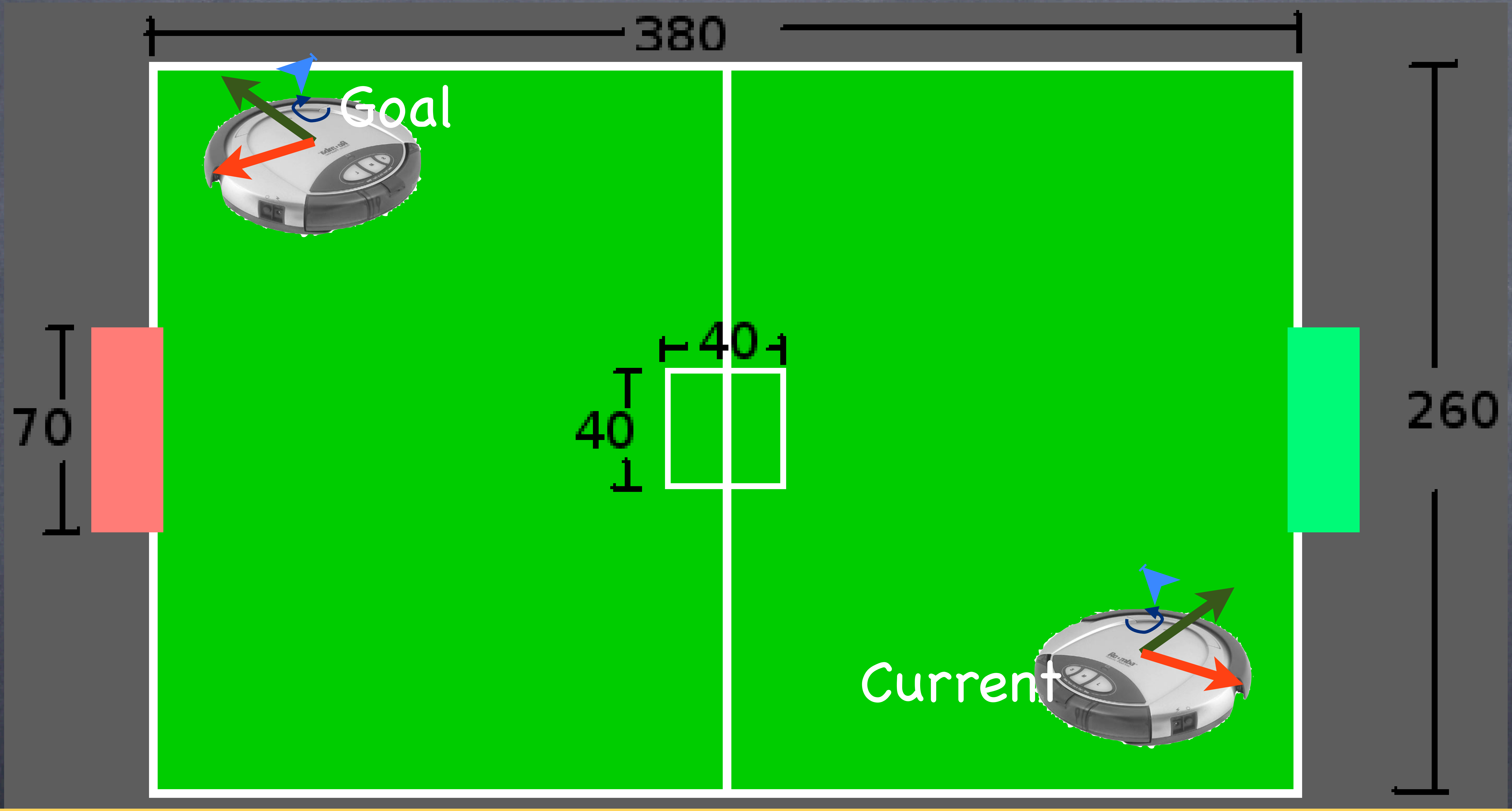
Multiple potentials



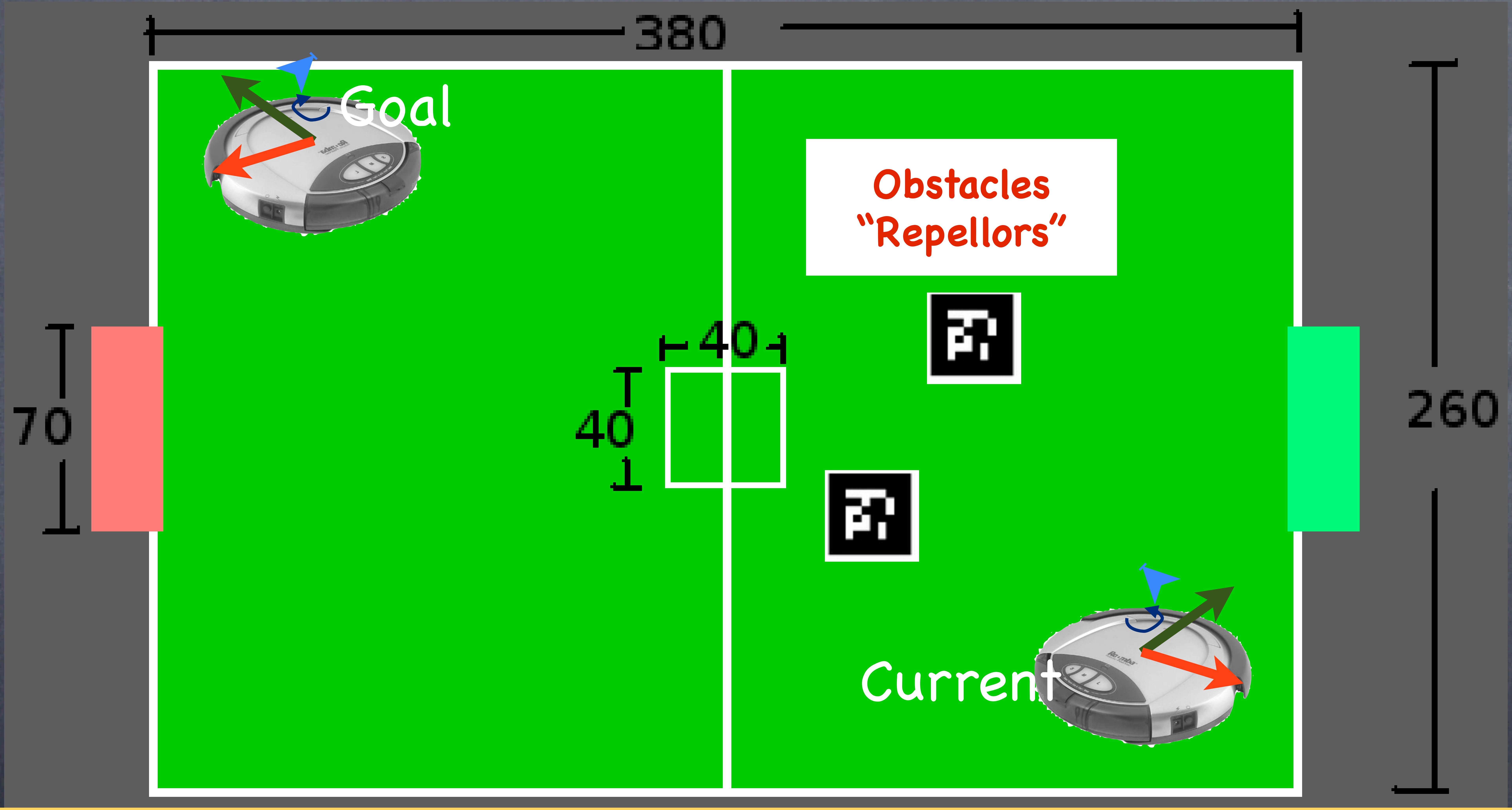
- Output of potential field is a vector
- Combine multiple potentials through vector summation

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

describe performance for this case

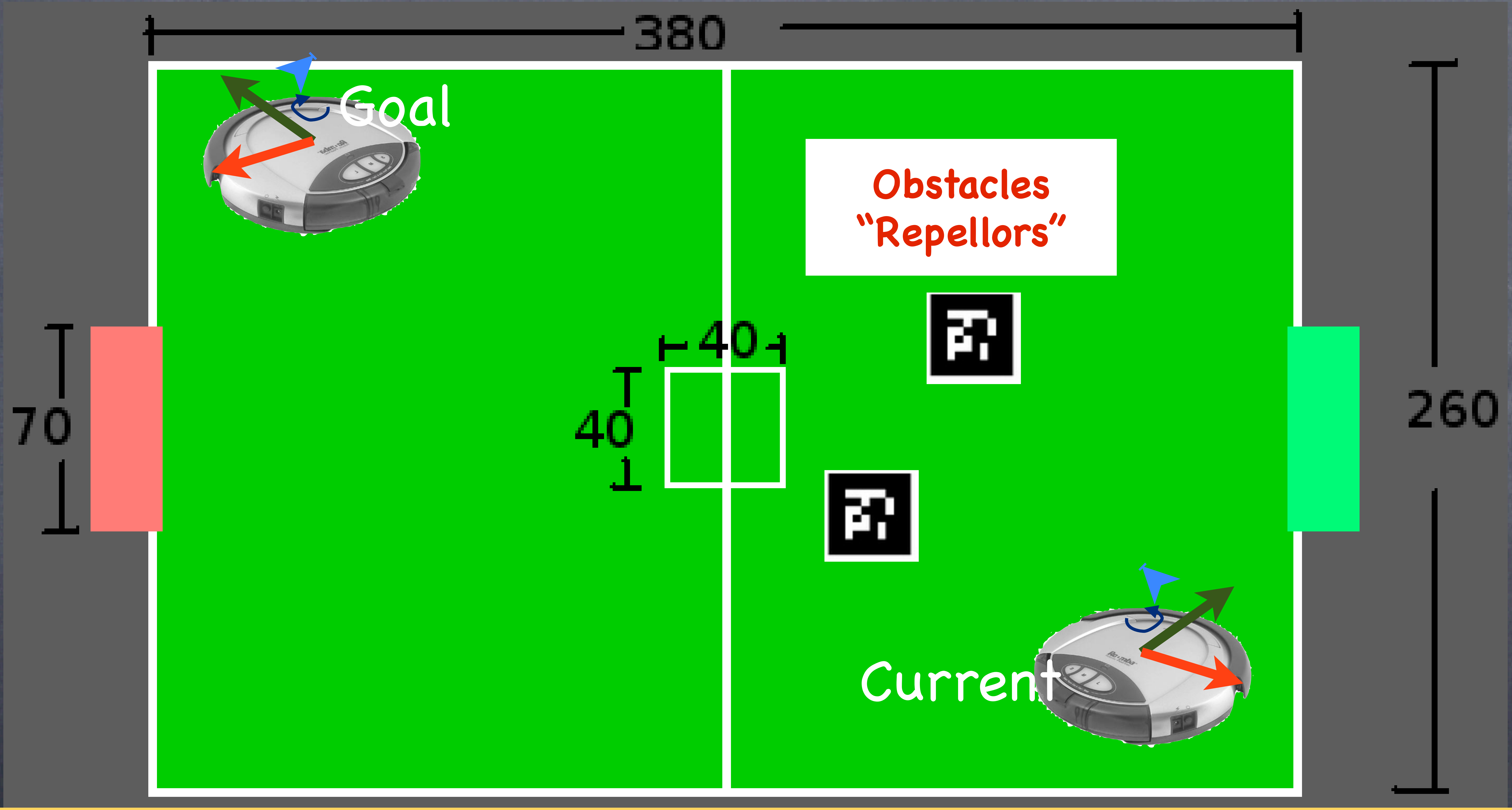


describe performance for this case



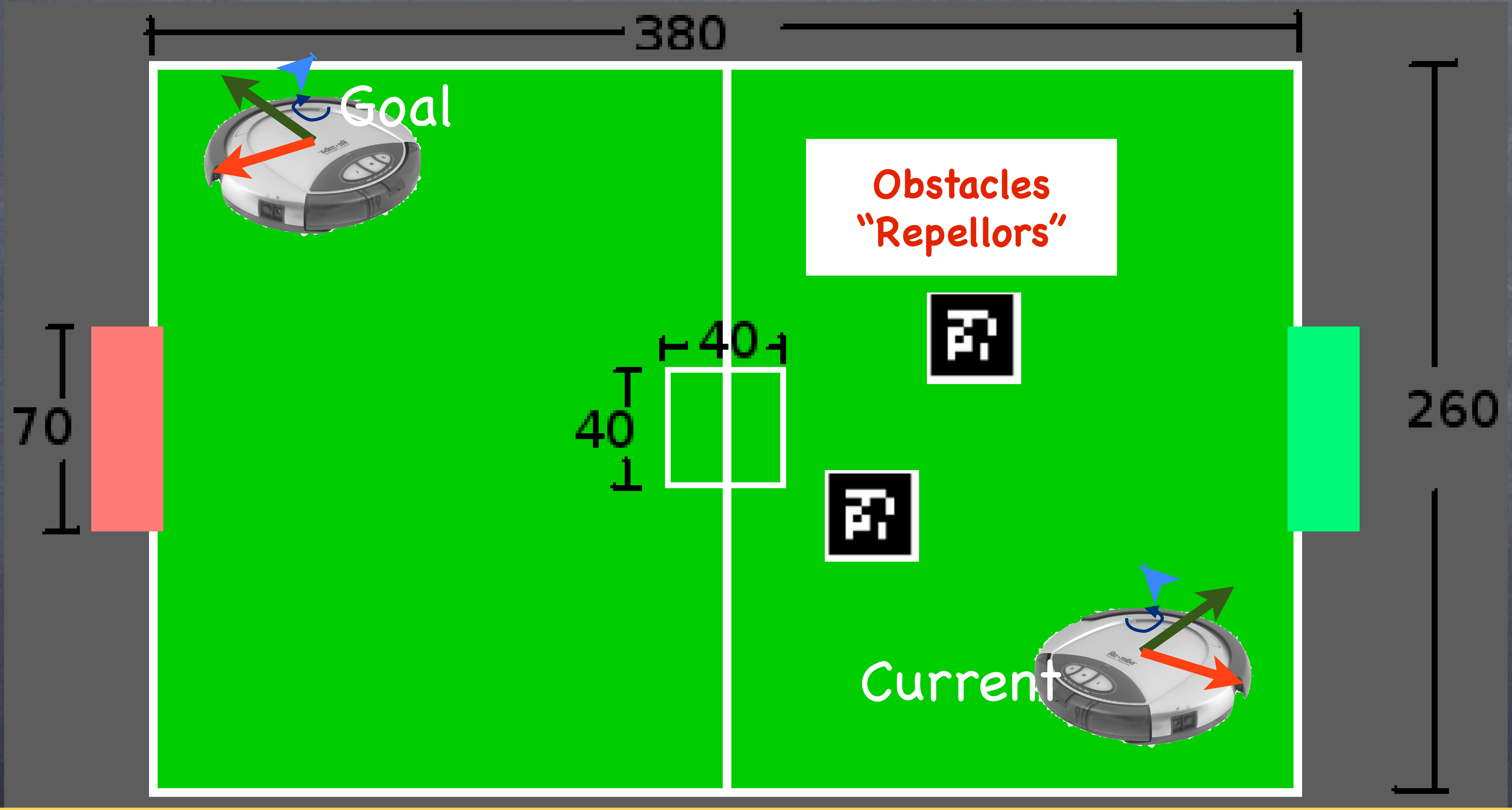
describe performance for this case

how do we deal with repellors?



add sum of repulsive potentials

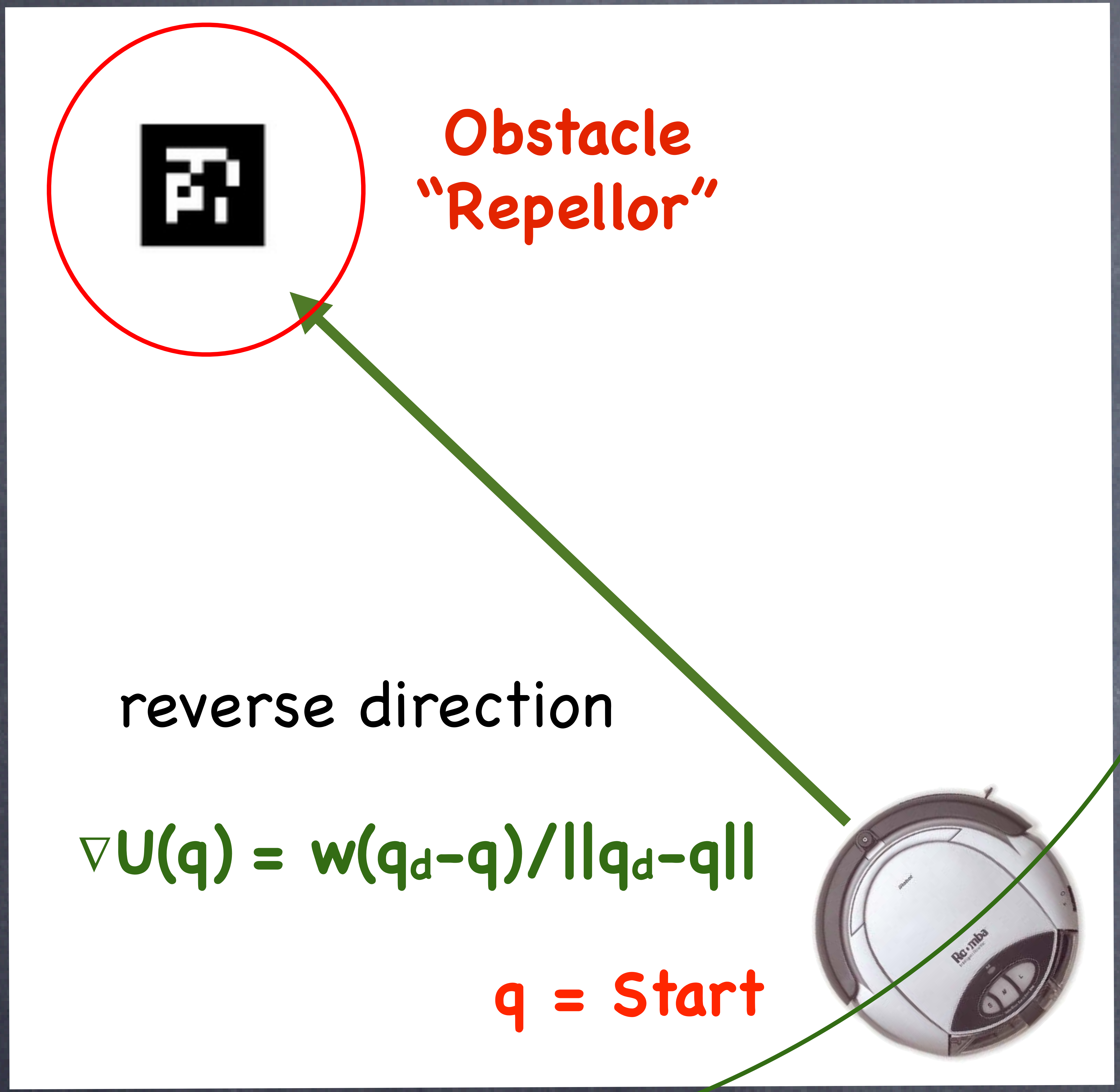
$$U(q) = U_{attracts}(q) + U_{repellors}(q)$$



Slide borrowed from Michigan Robotics autorob.org

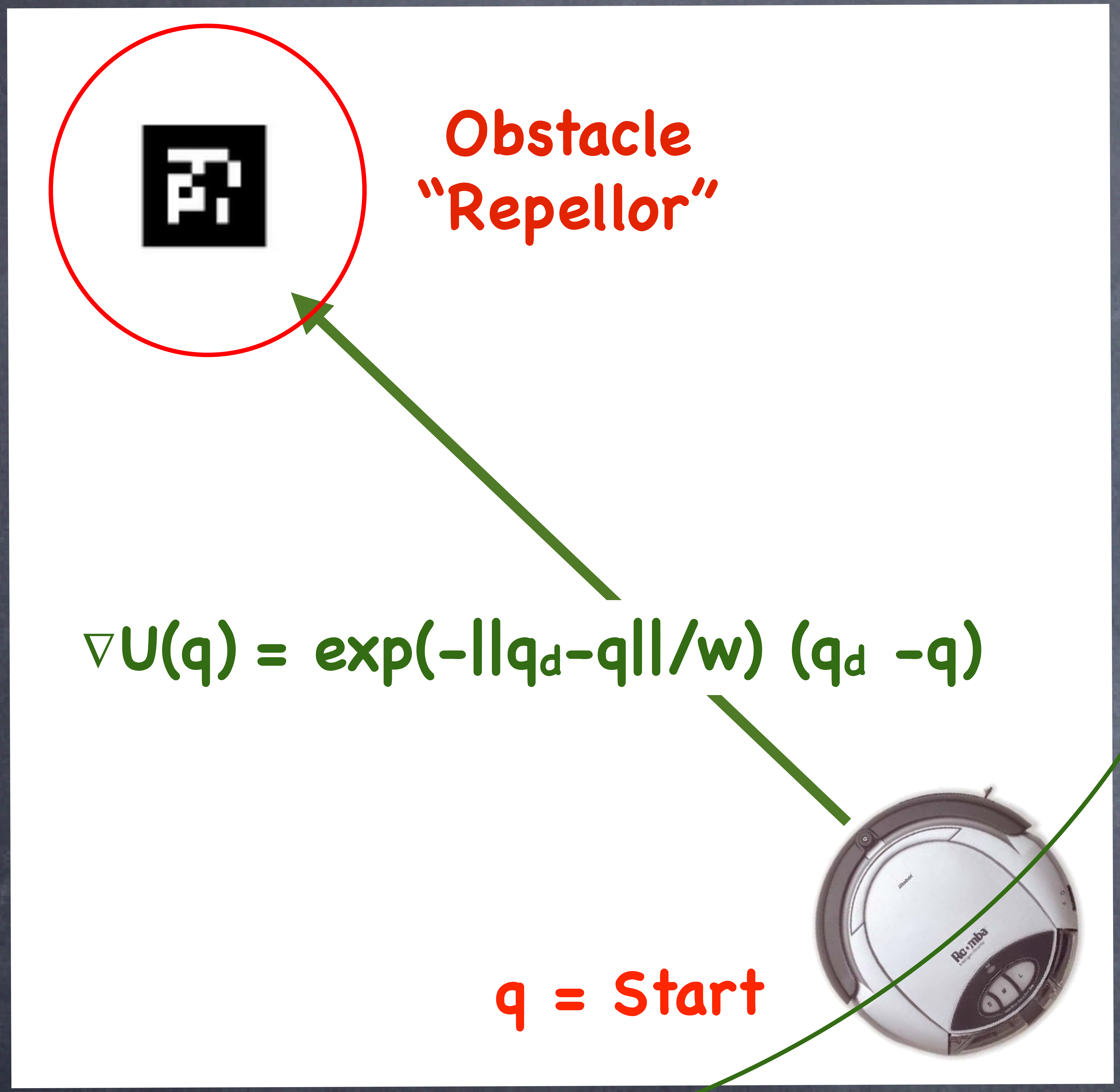
“Cone” Repellor

potential problems?

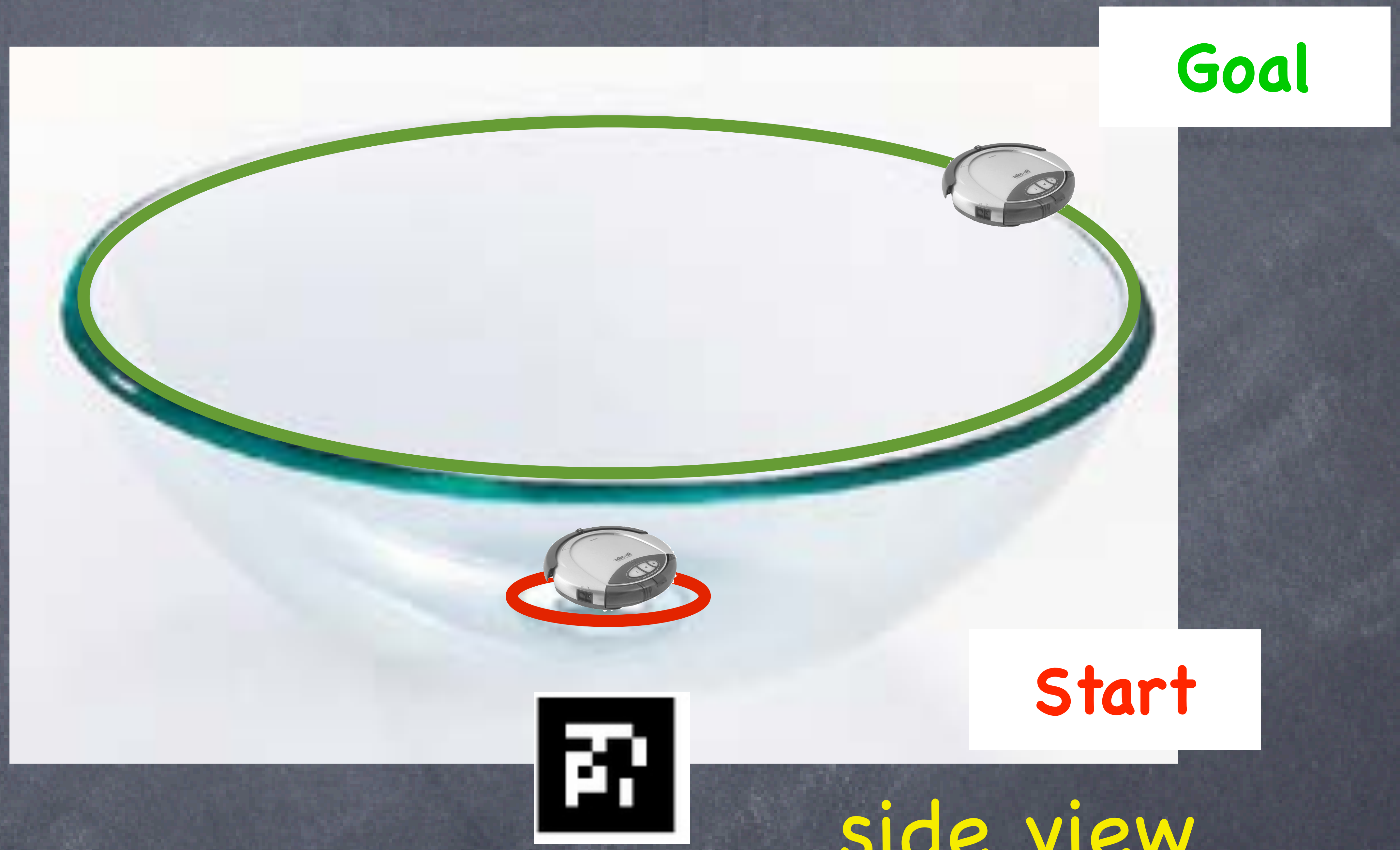


side view

“Bowl” Repellor

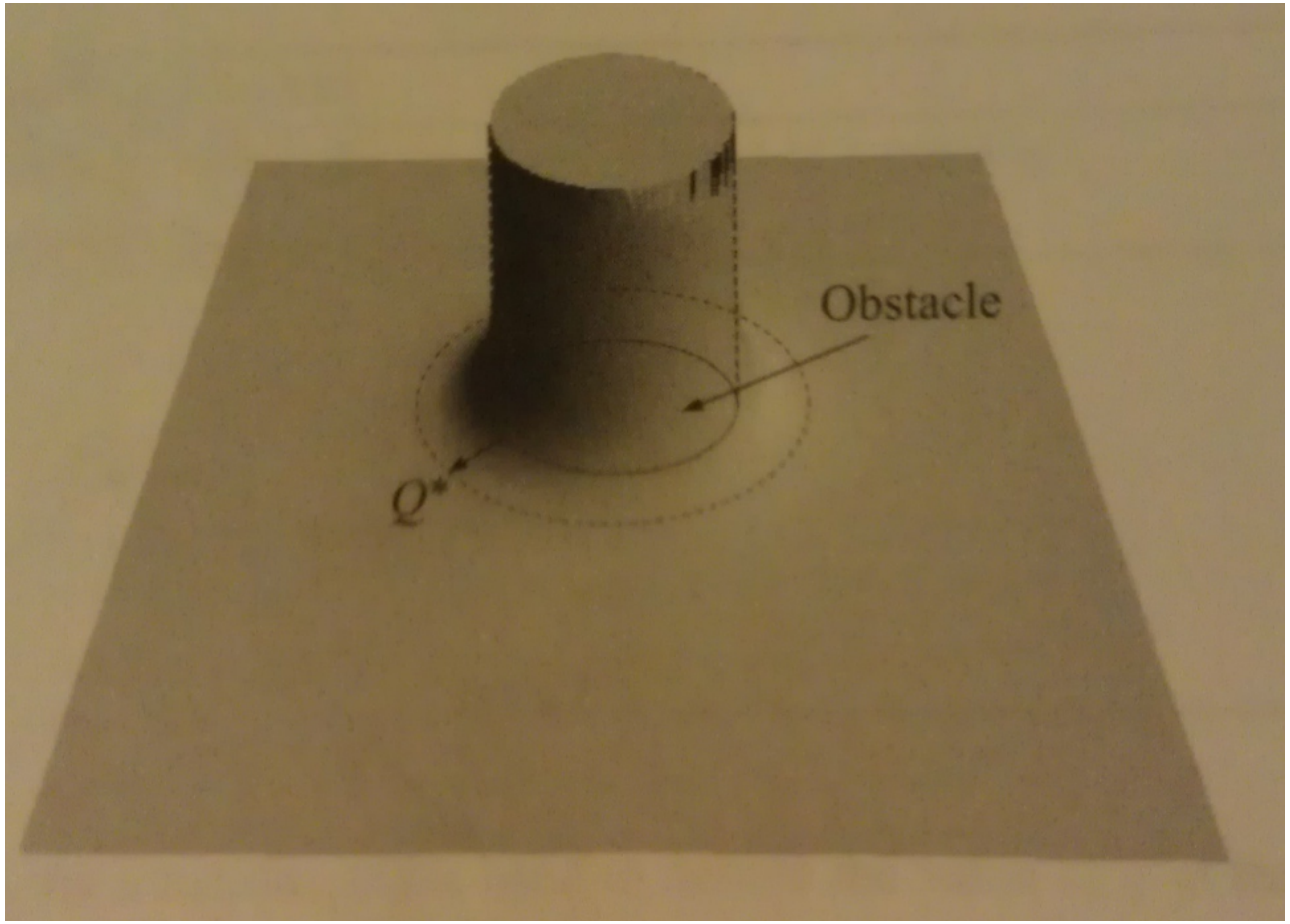


top view



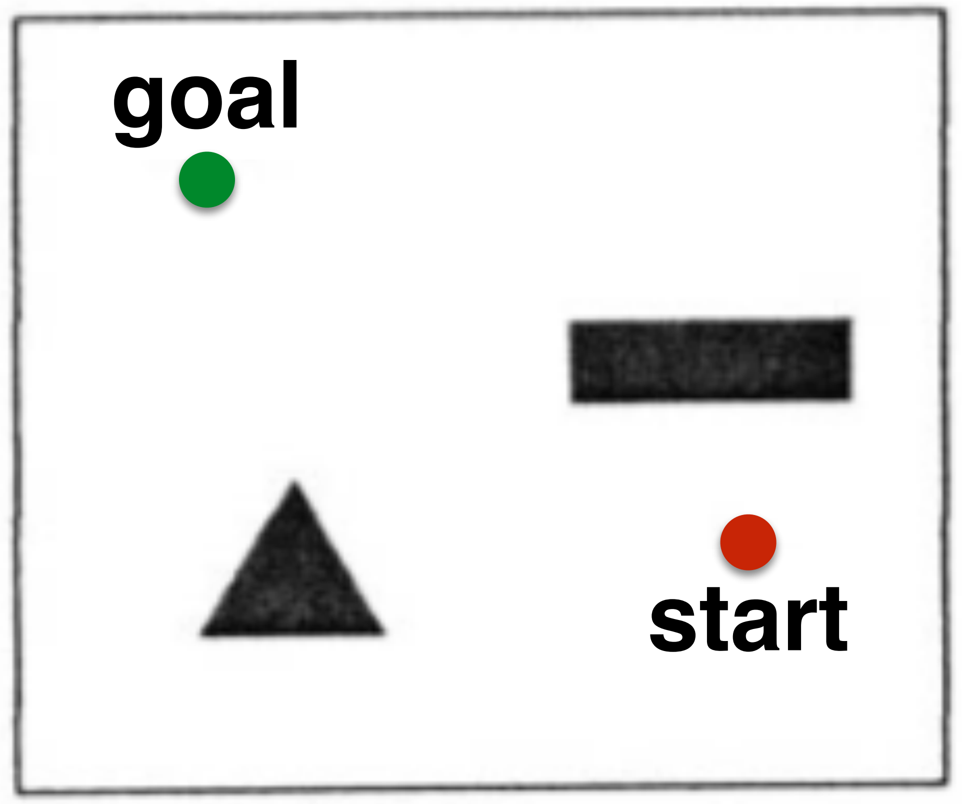
side view

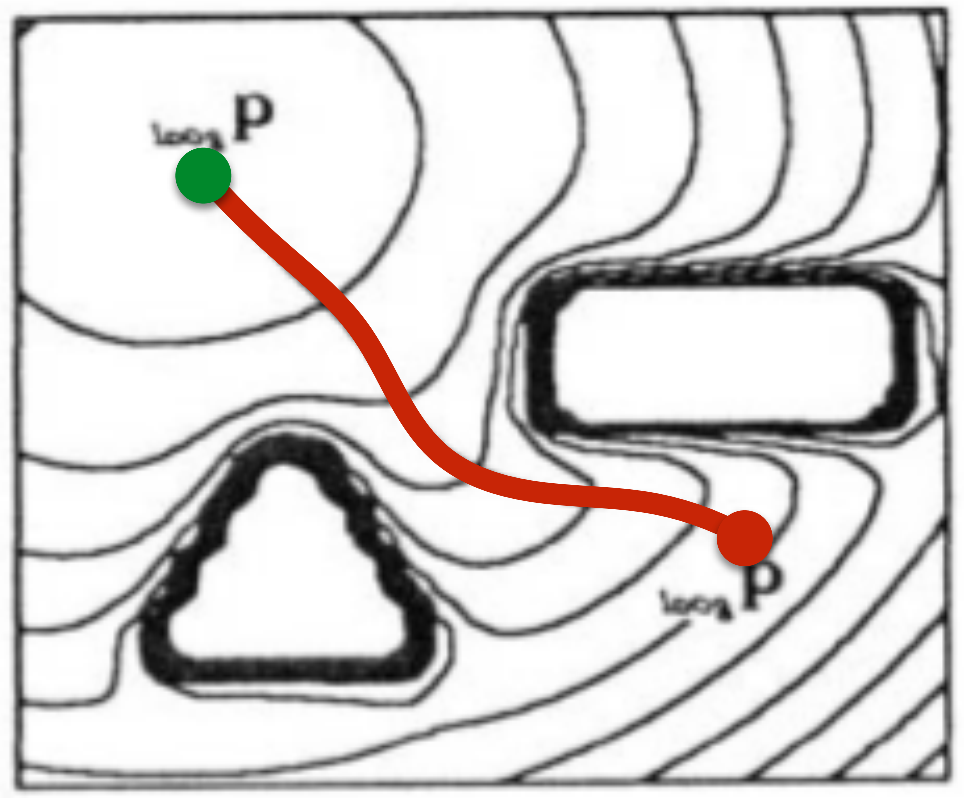
repellor should only have local influence,
repelling only around boundary improves path



2 Obstacle example

configuration space





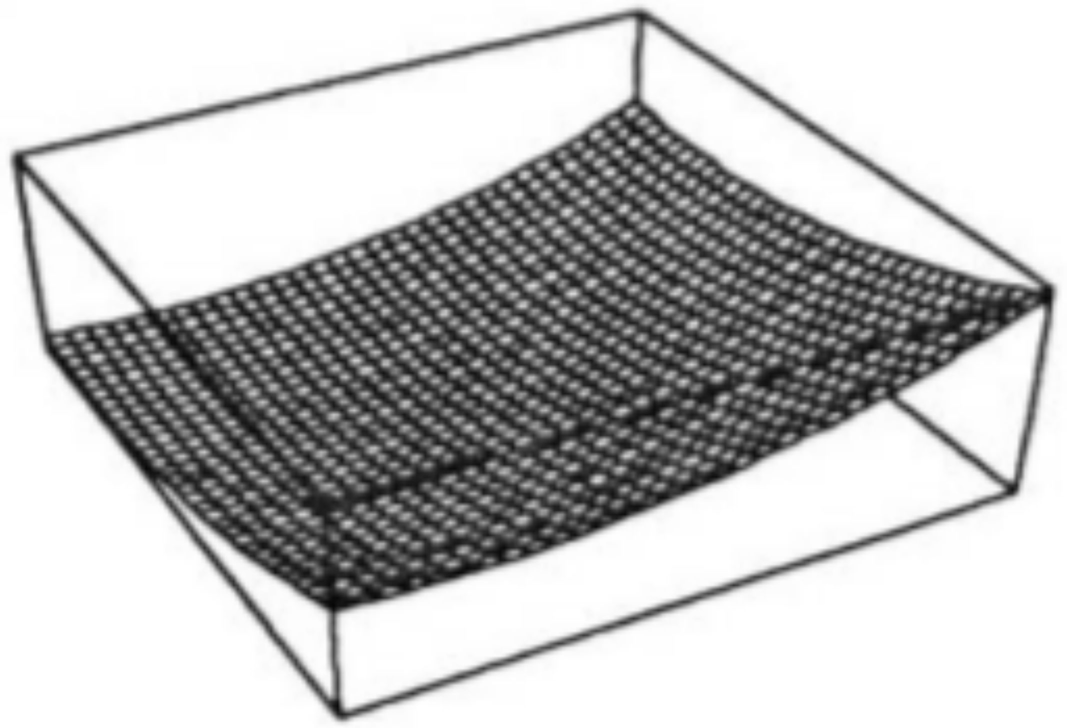
path from descent on gradient field

resulting gradient field

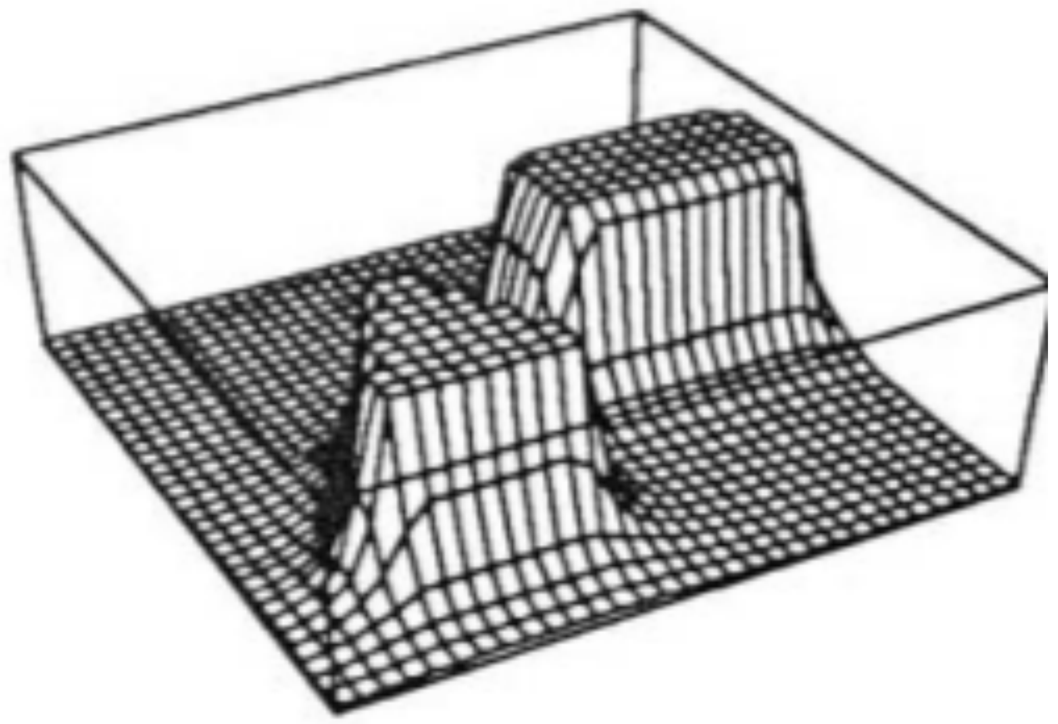
attractor field

repellor fields

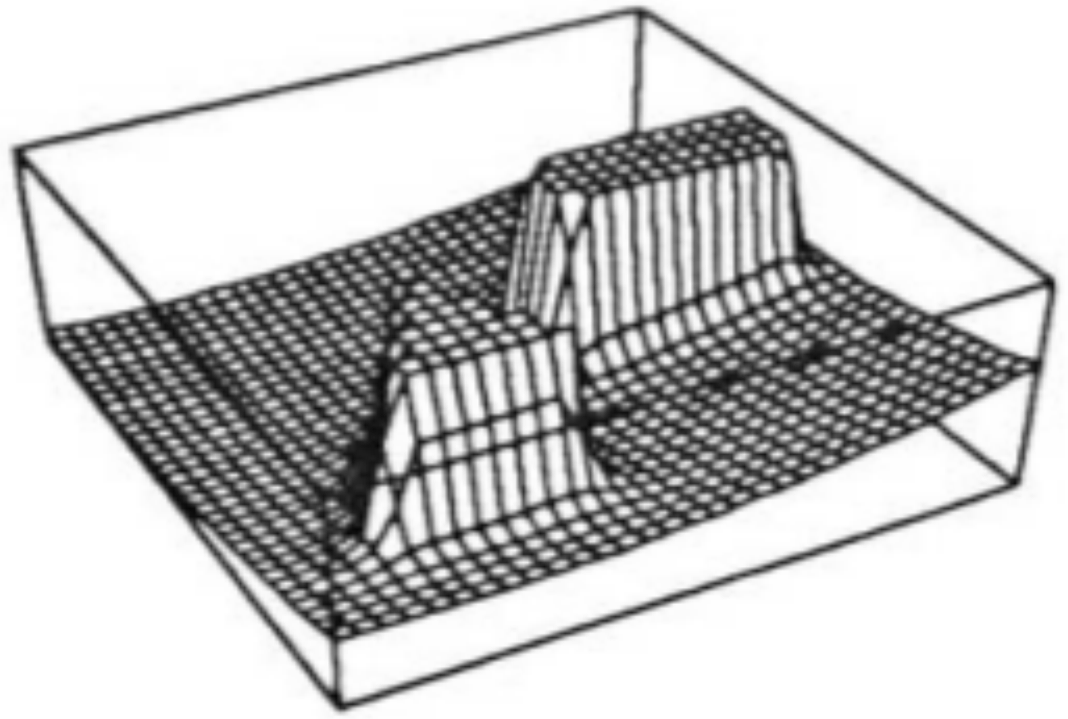
combined potential

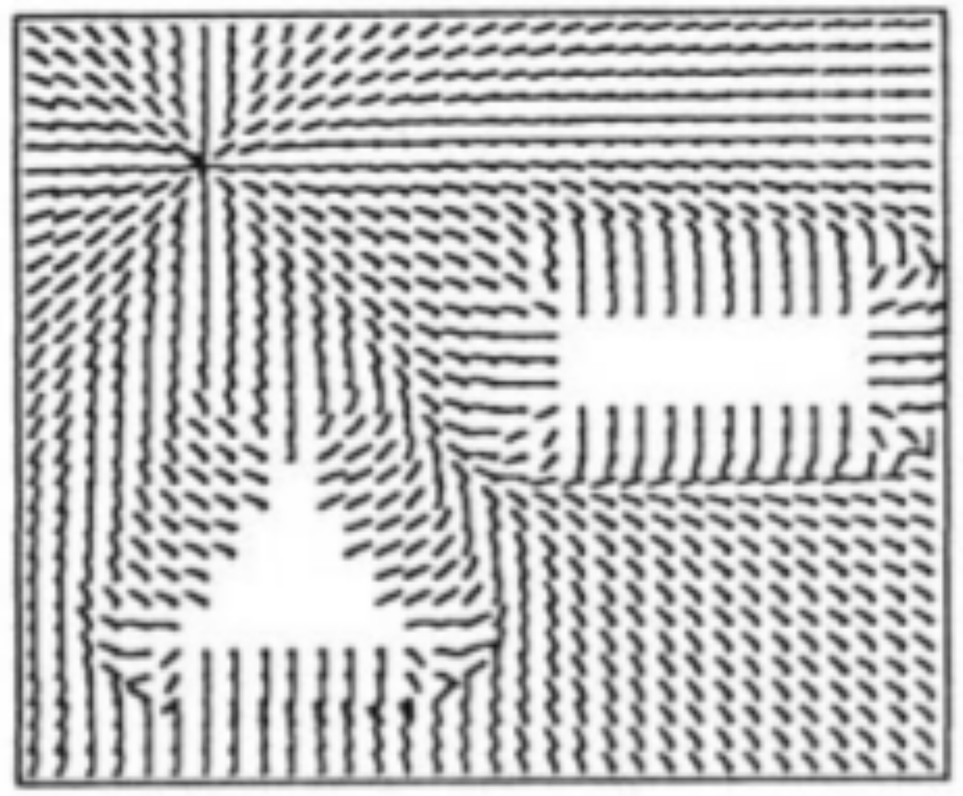


+



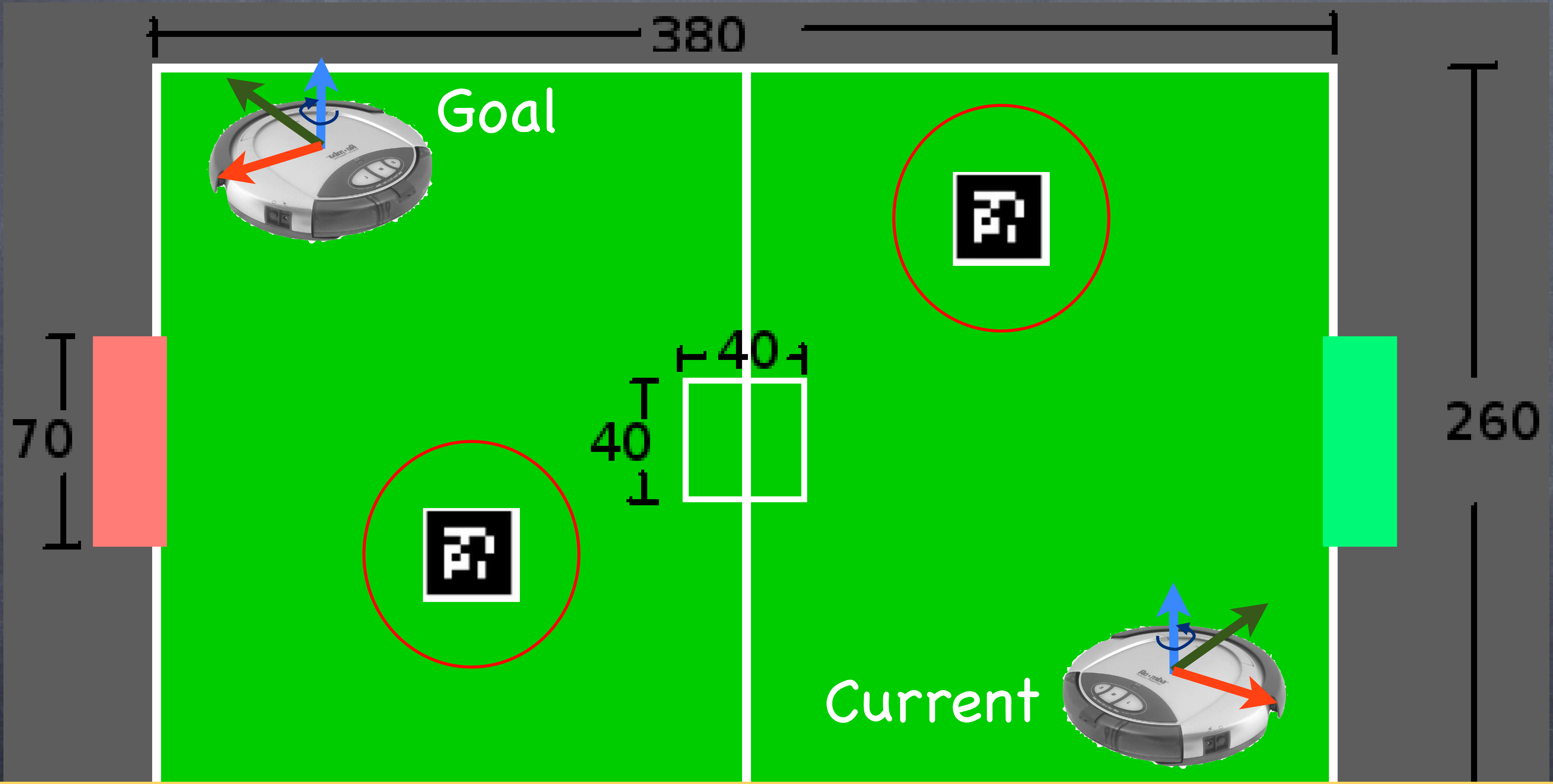
=



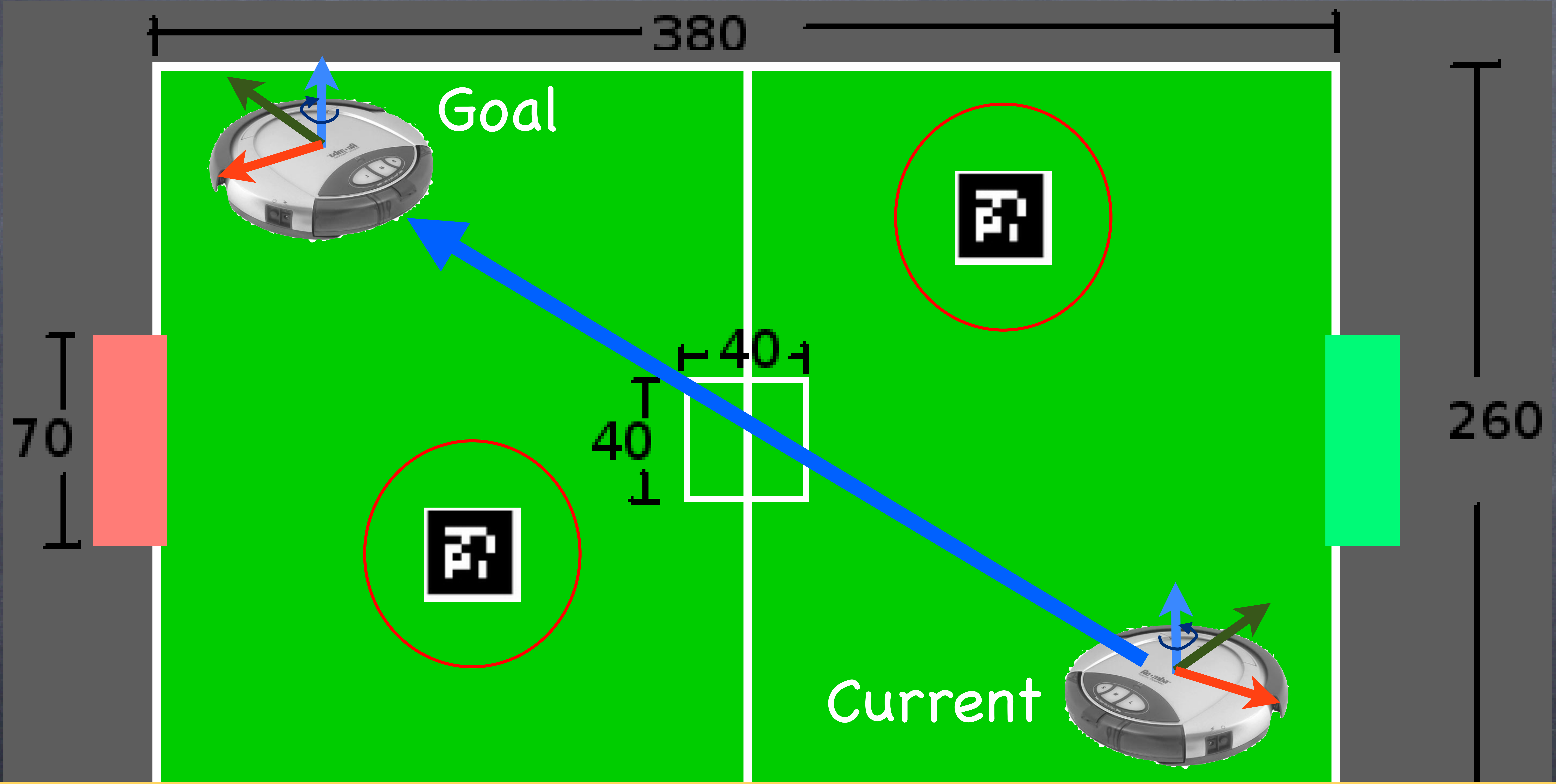


Slide borrowed from Michigan Robotics autorob.org

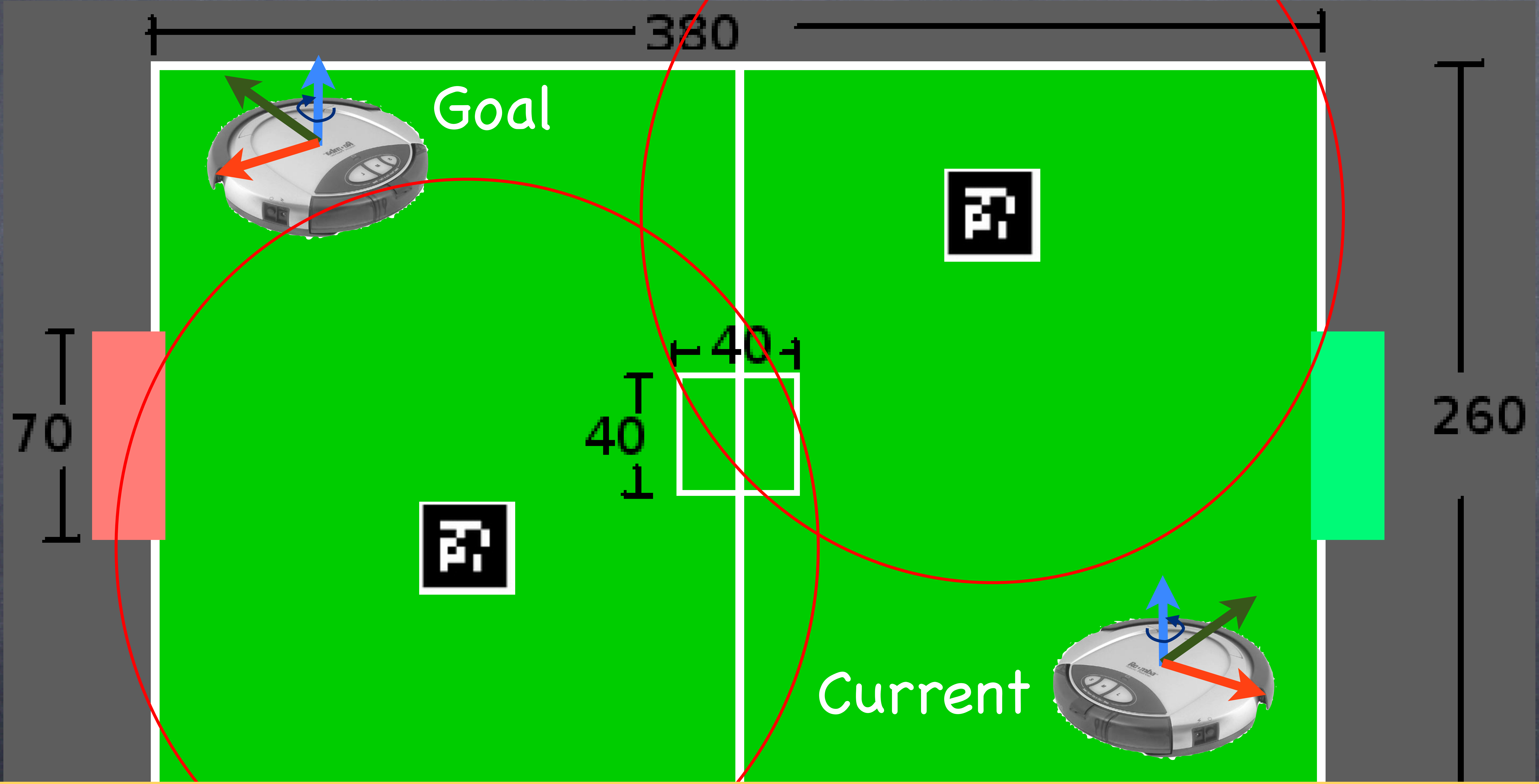
describe performance for this case
with cone attractor to goal and bowl repellors
with limited weight



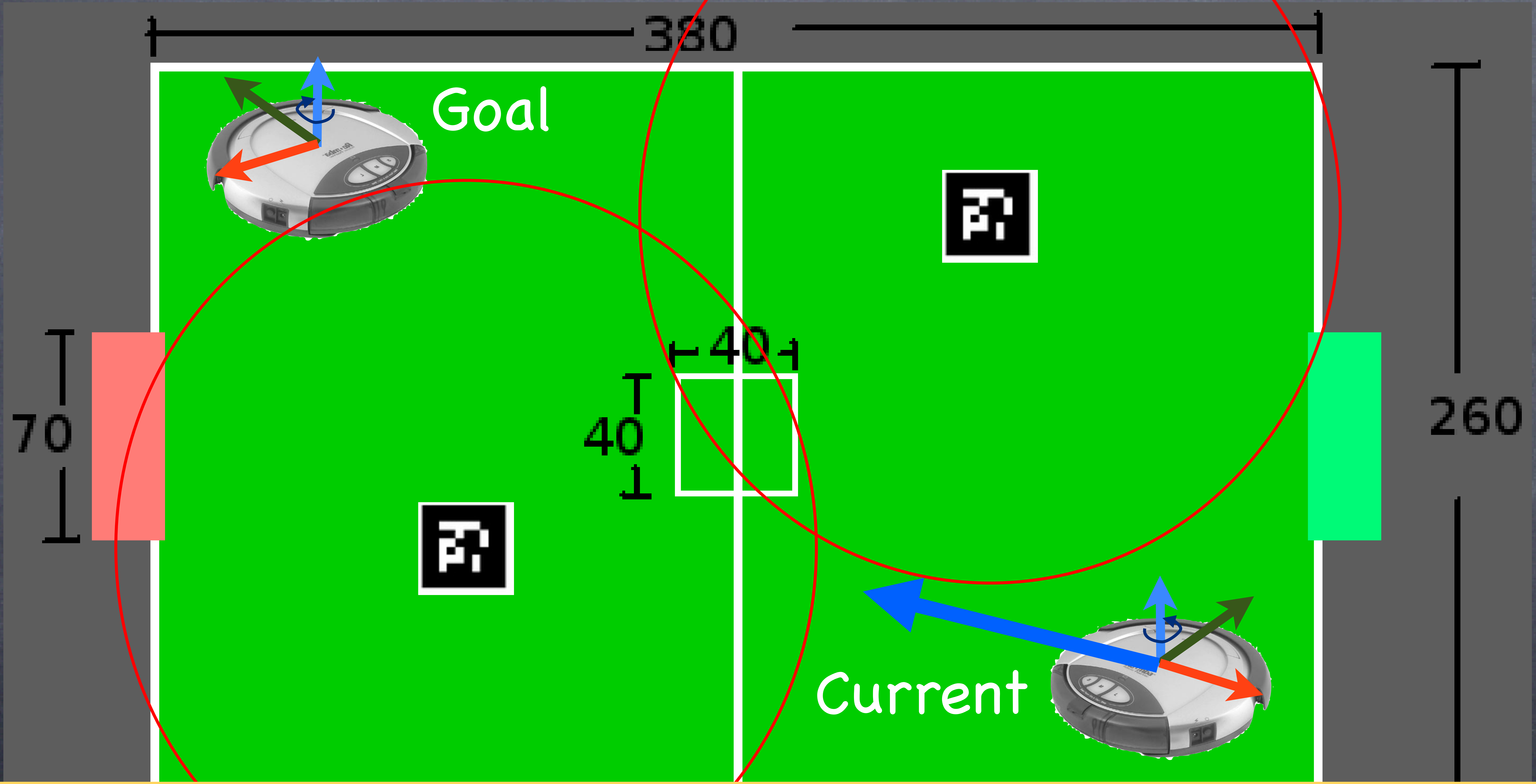
describe performance for this case
with cone attractor to goal and bowl repellors
with limited weight



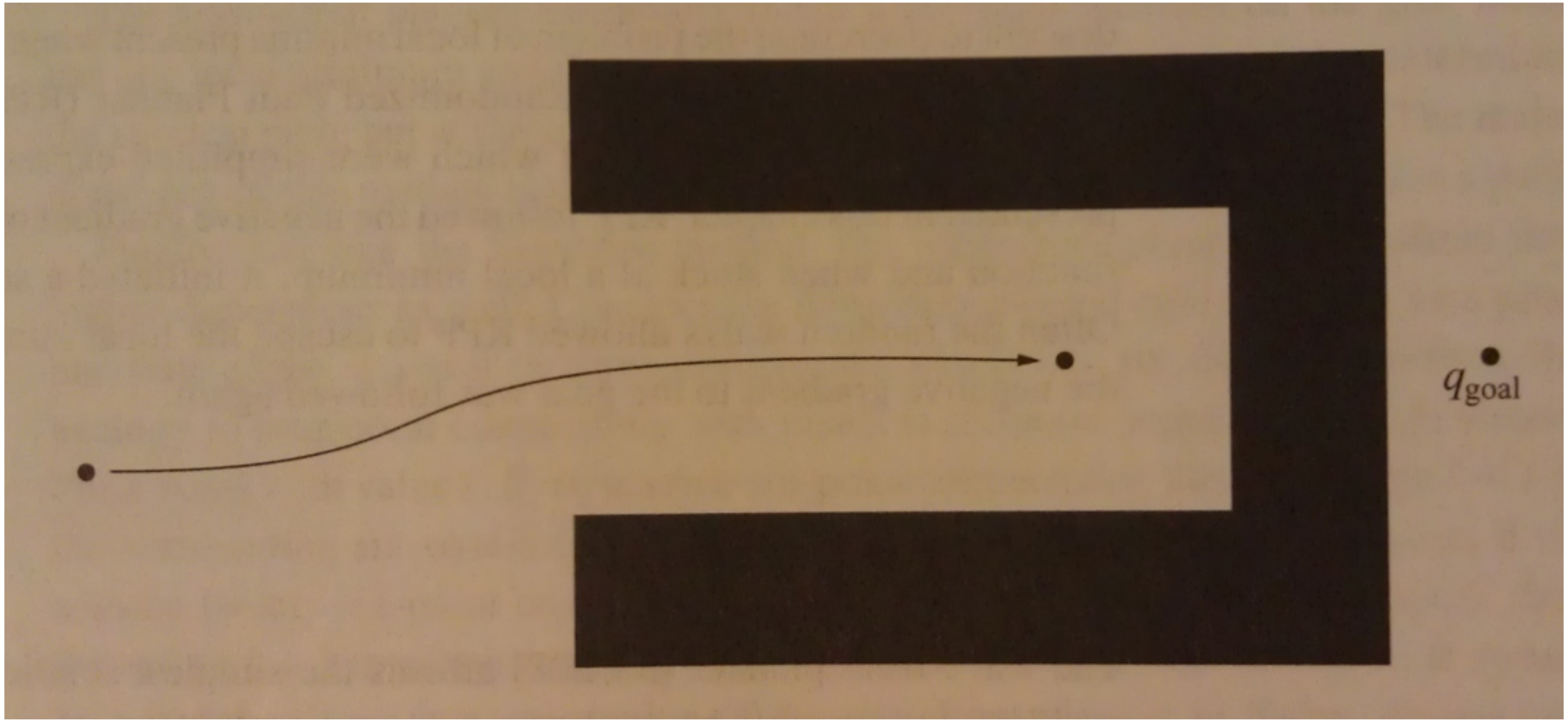
describe performance for this case
with cone attractor to goal and bowl repellors
with limited weight



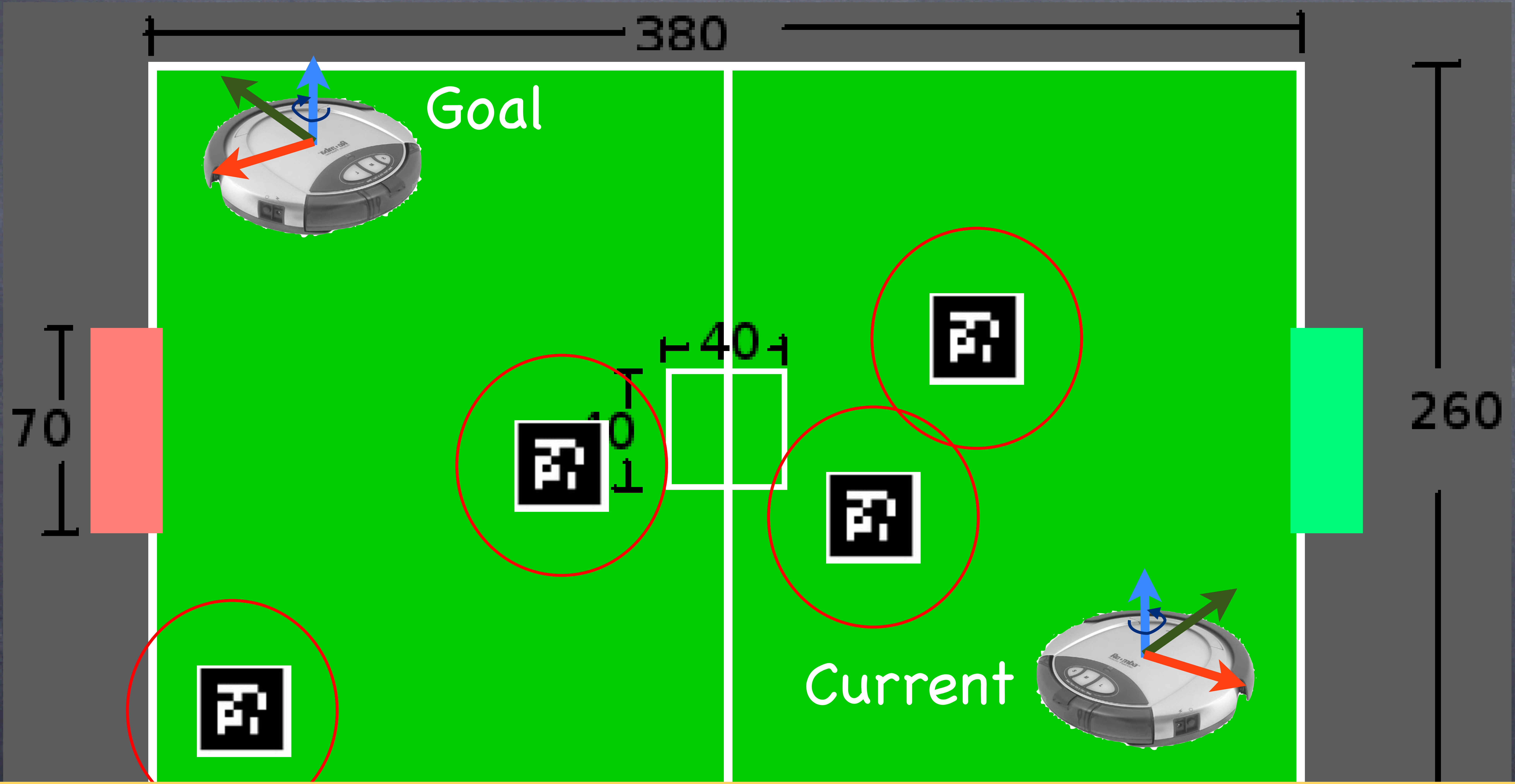
describe performance for this case
with cone attractor to goal and bowl repellors
with limited weight



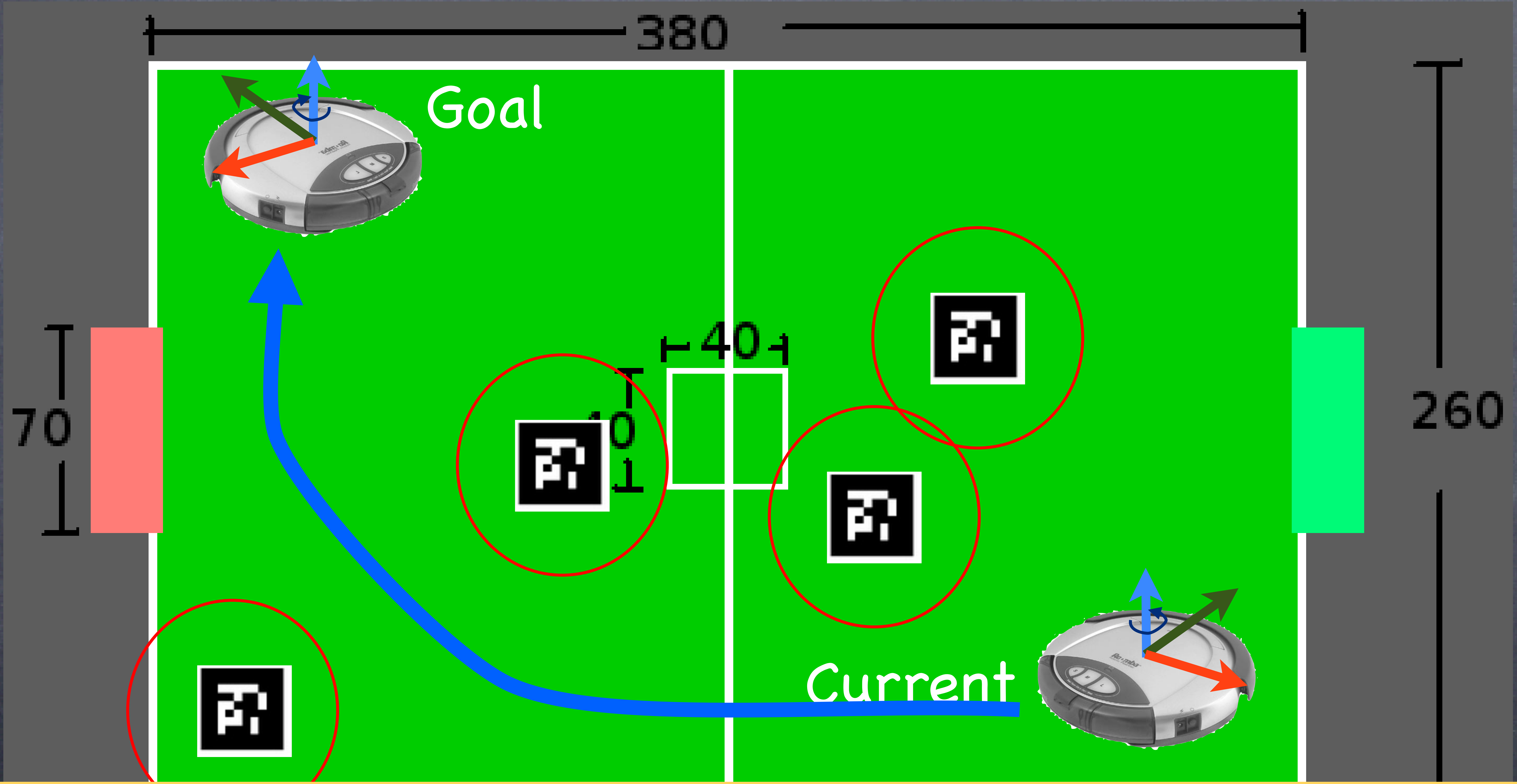
Local Minima



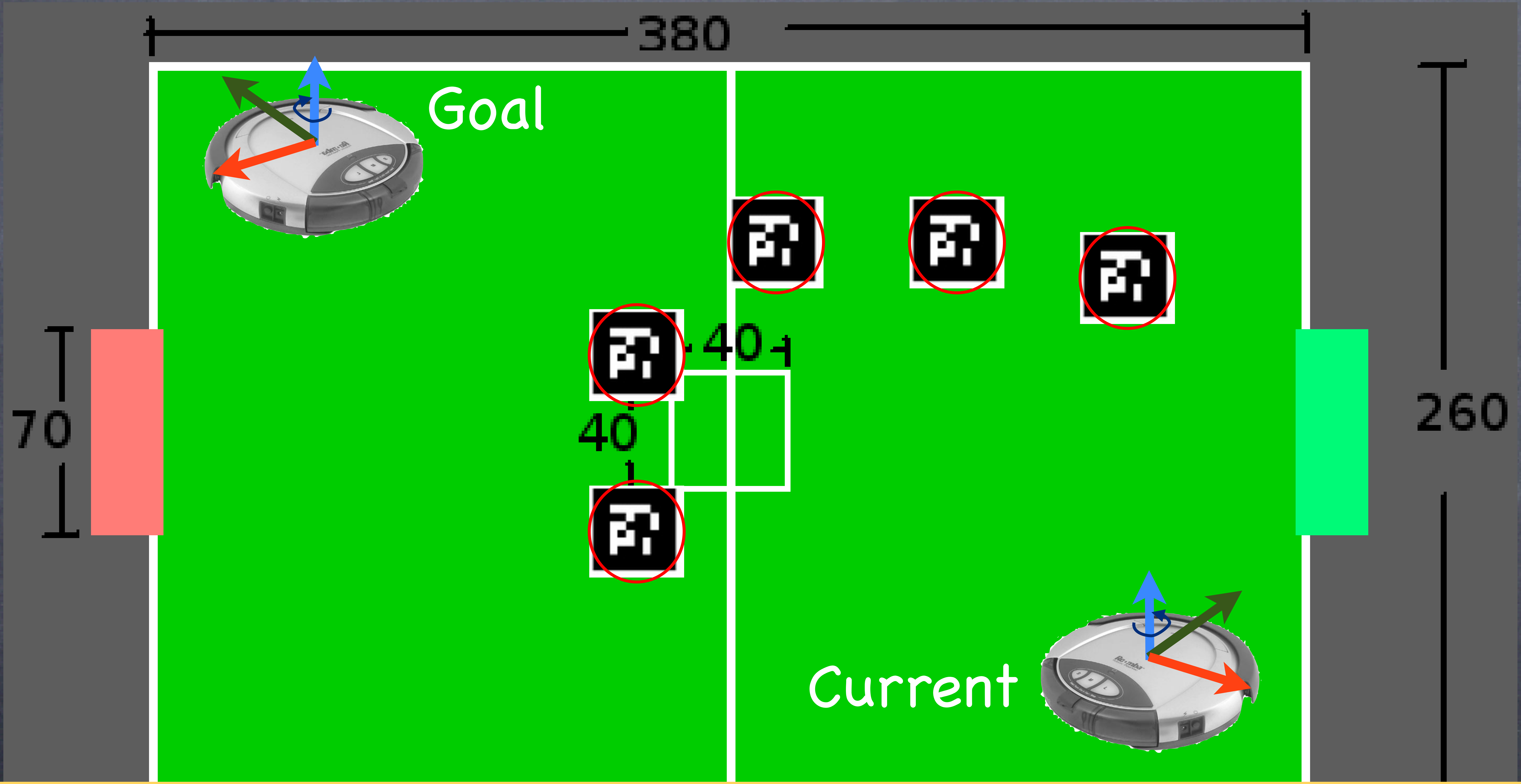
describe performance for this case



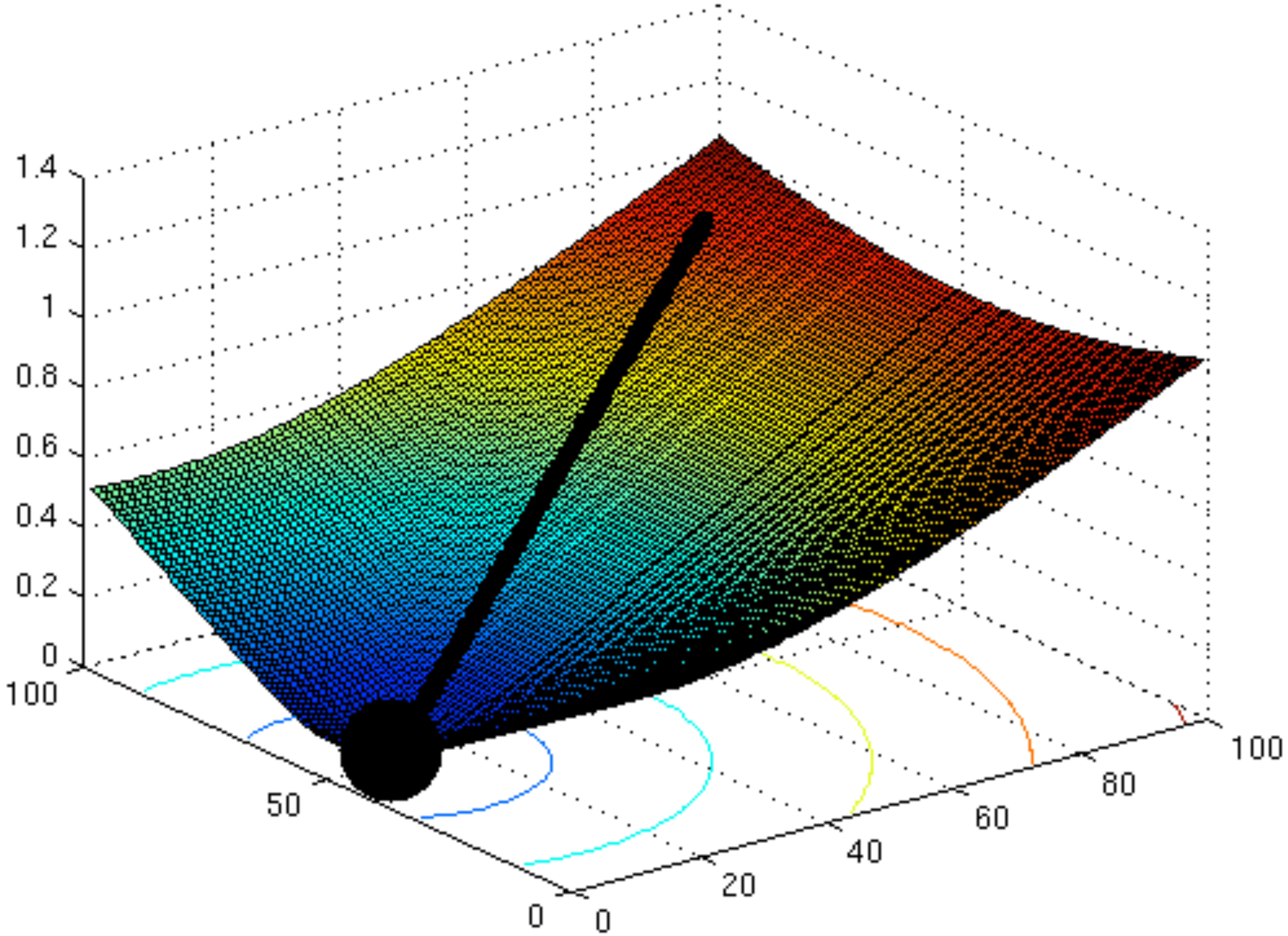
describe performance for this case



describe performance for this case

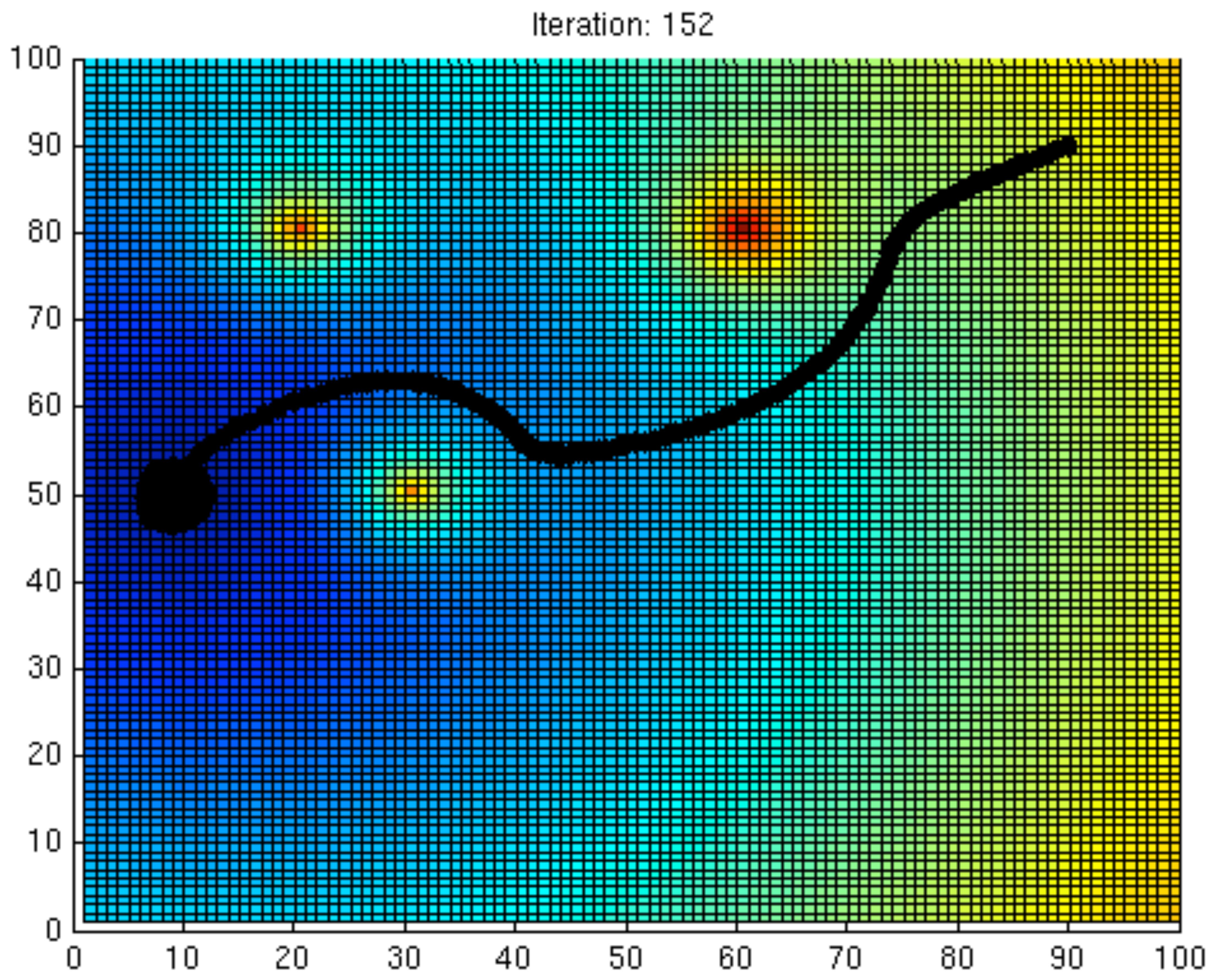


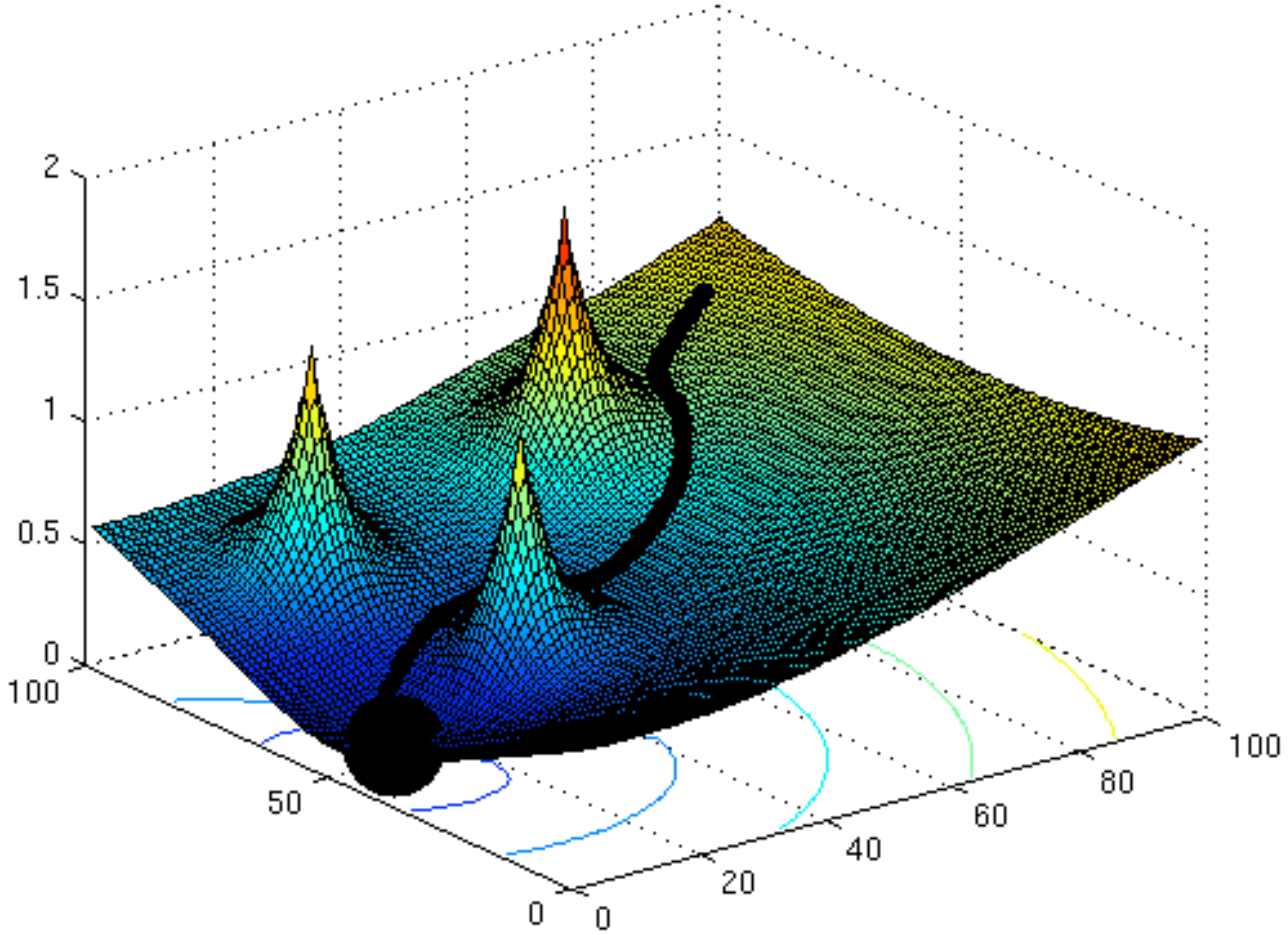
matlab example



pfield.m [1 5 8 12]

matlab example

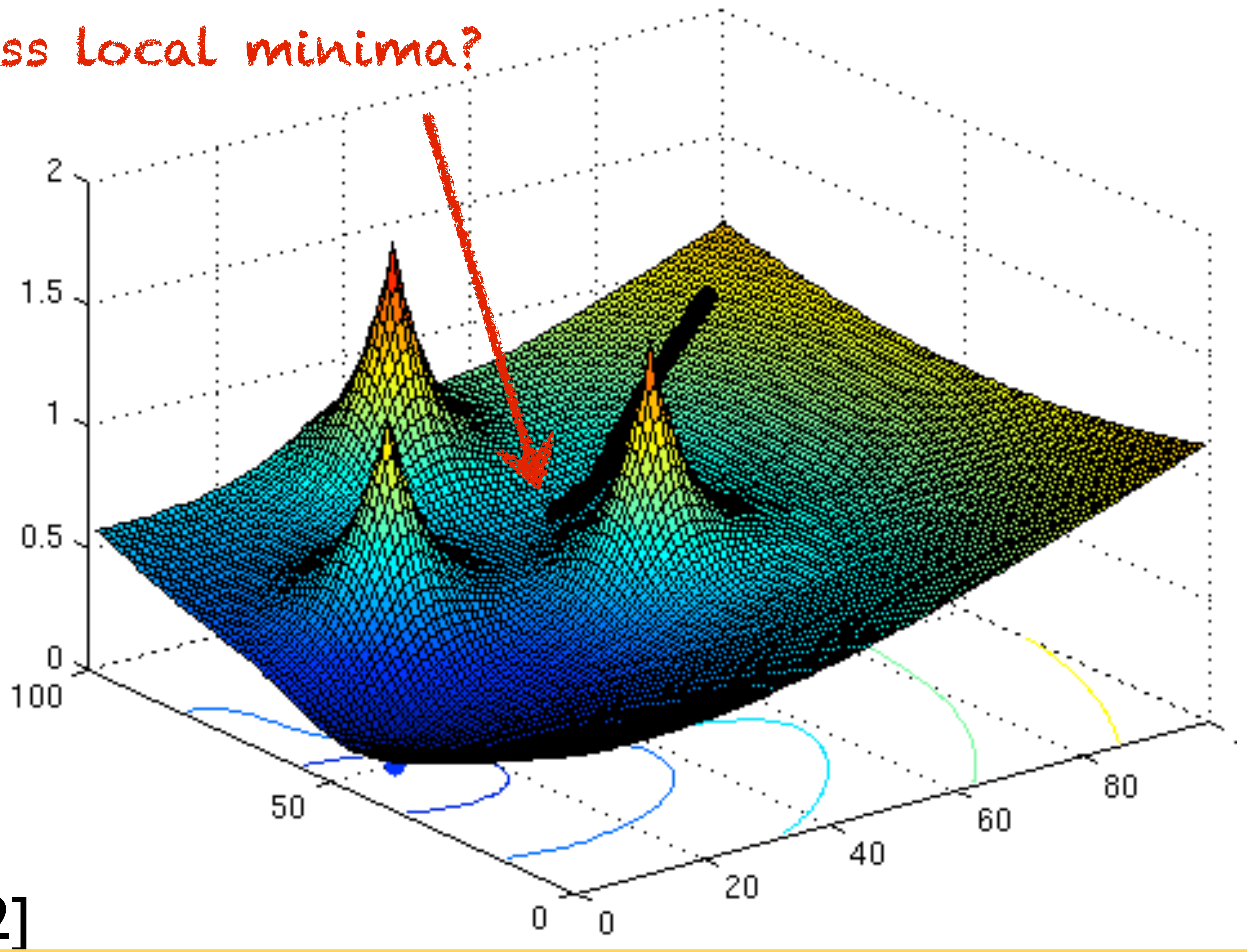




pfield.m [1 5 8 12]

matlab example

How to address local minima?



pfield.m [1 5 8 12]

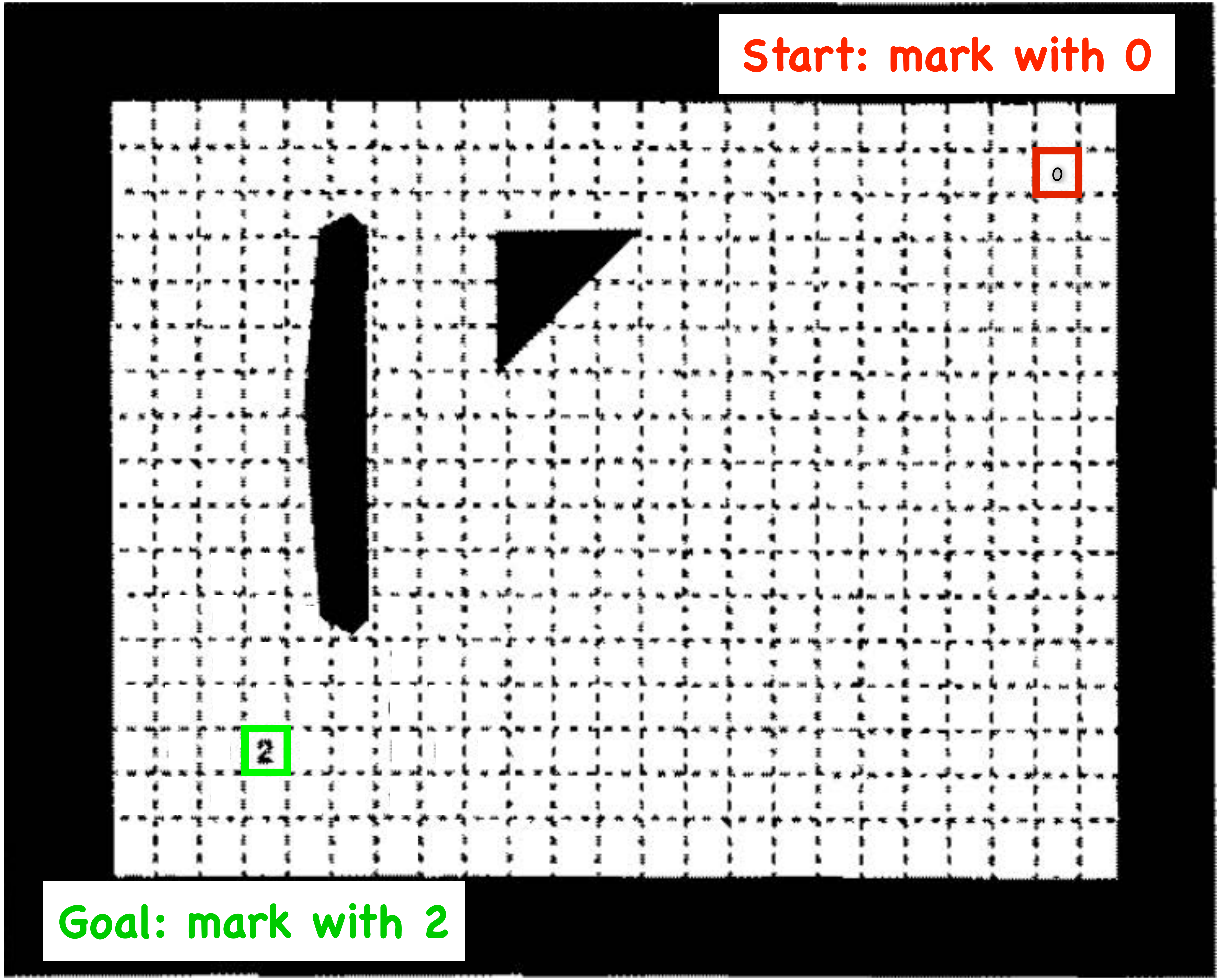
How can we get out of local minima?

How can we get out of local minima?

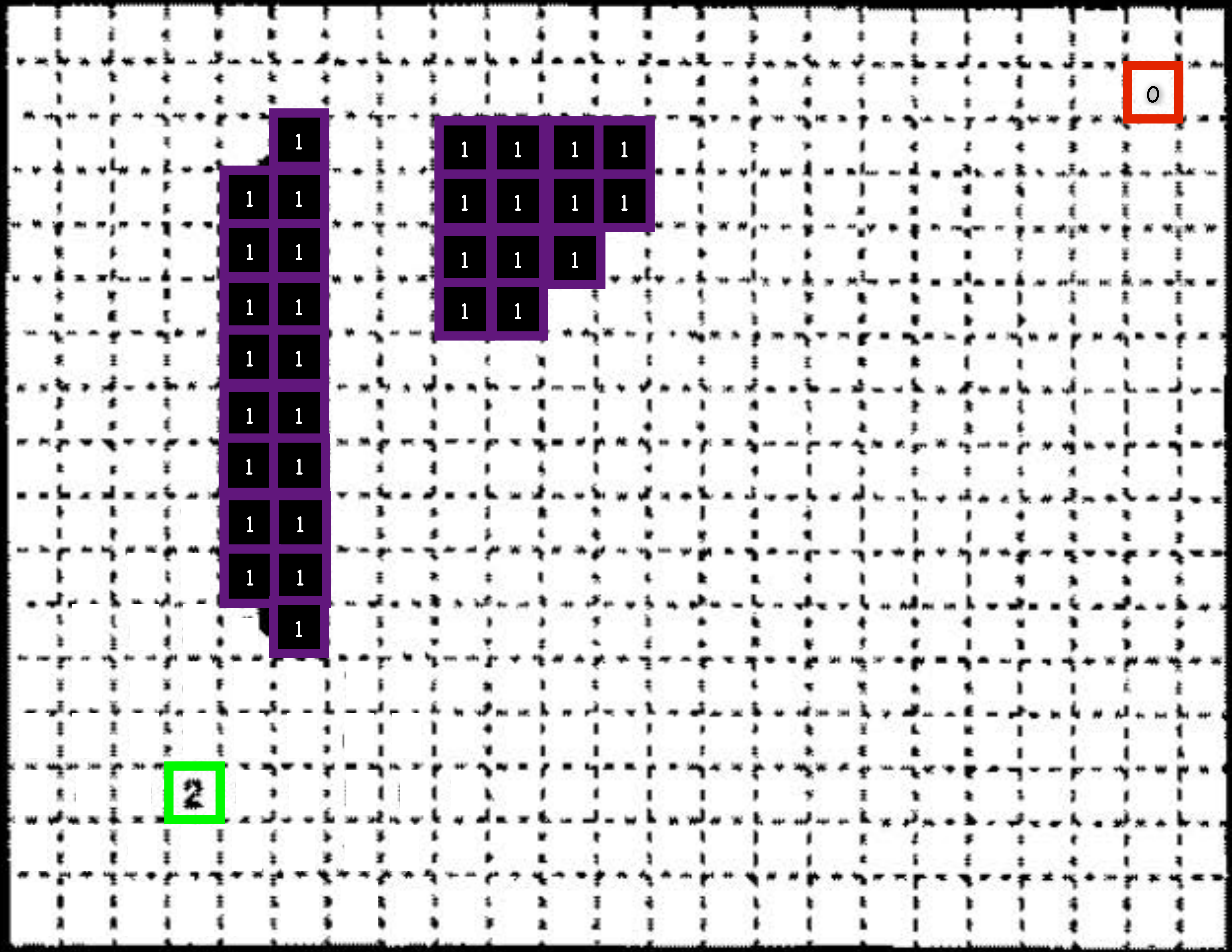
Go back to planning.

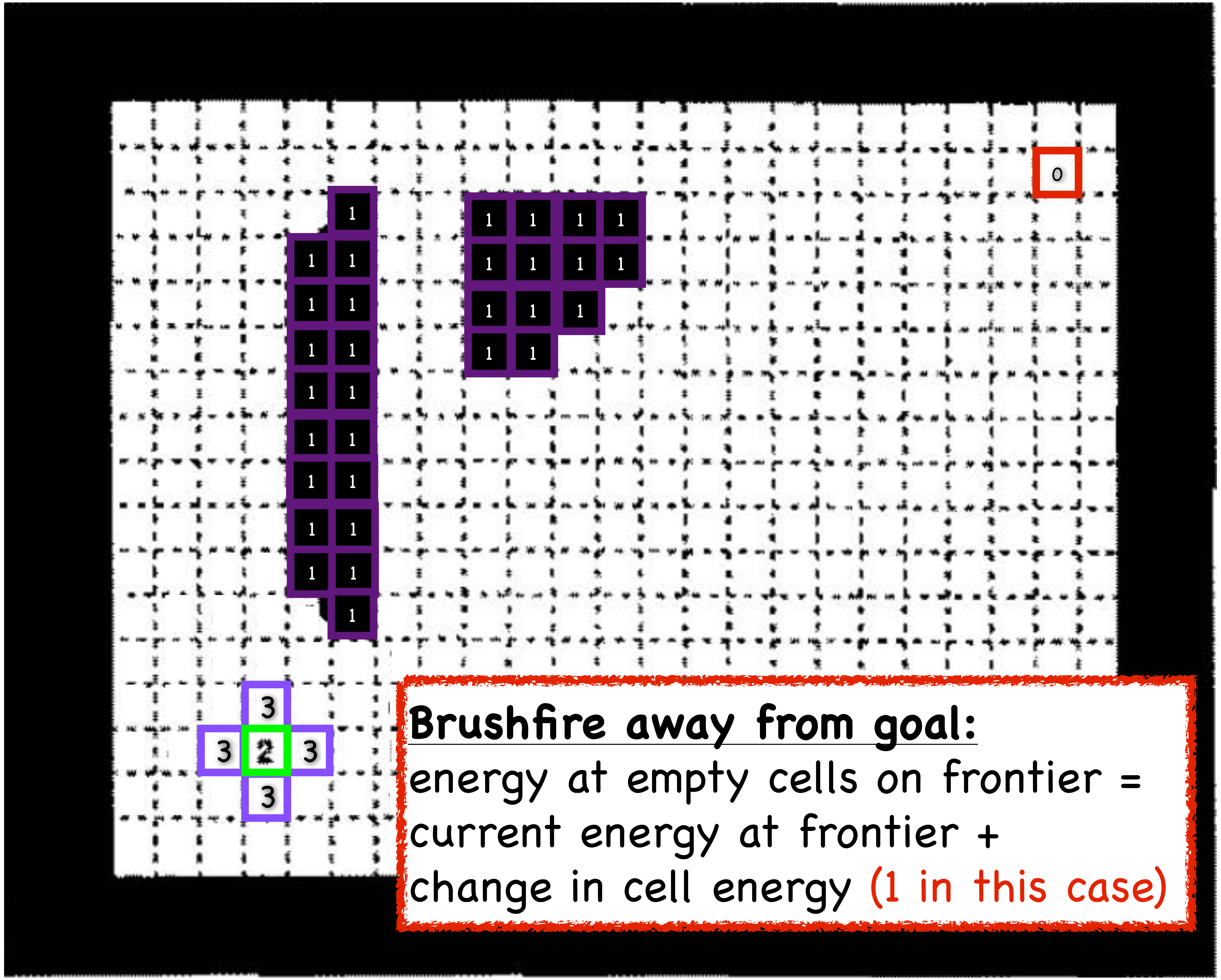
Wavefront Planning

- 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
0
2
Goal: mark with 2

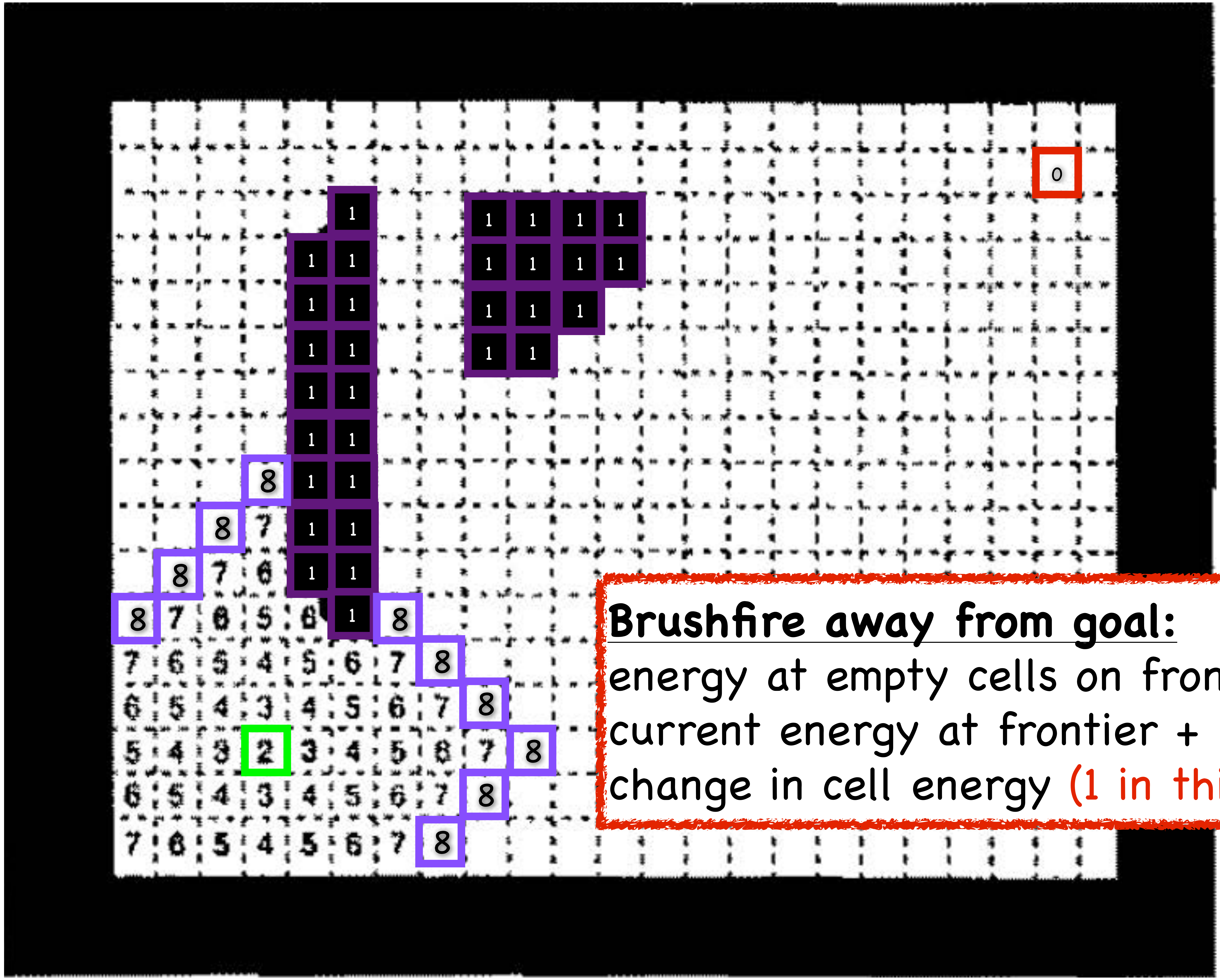
Obstacles: mark with 1





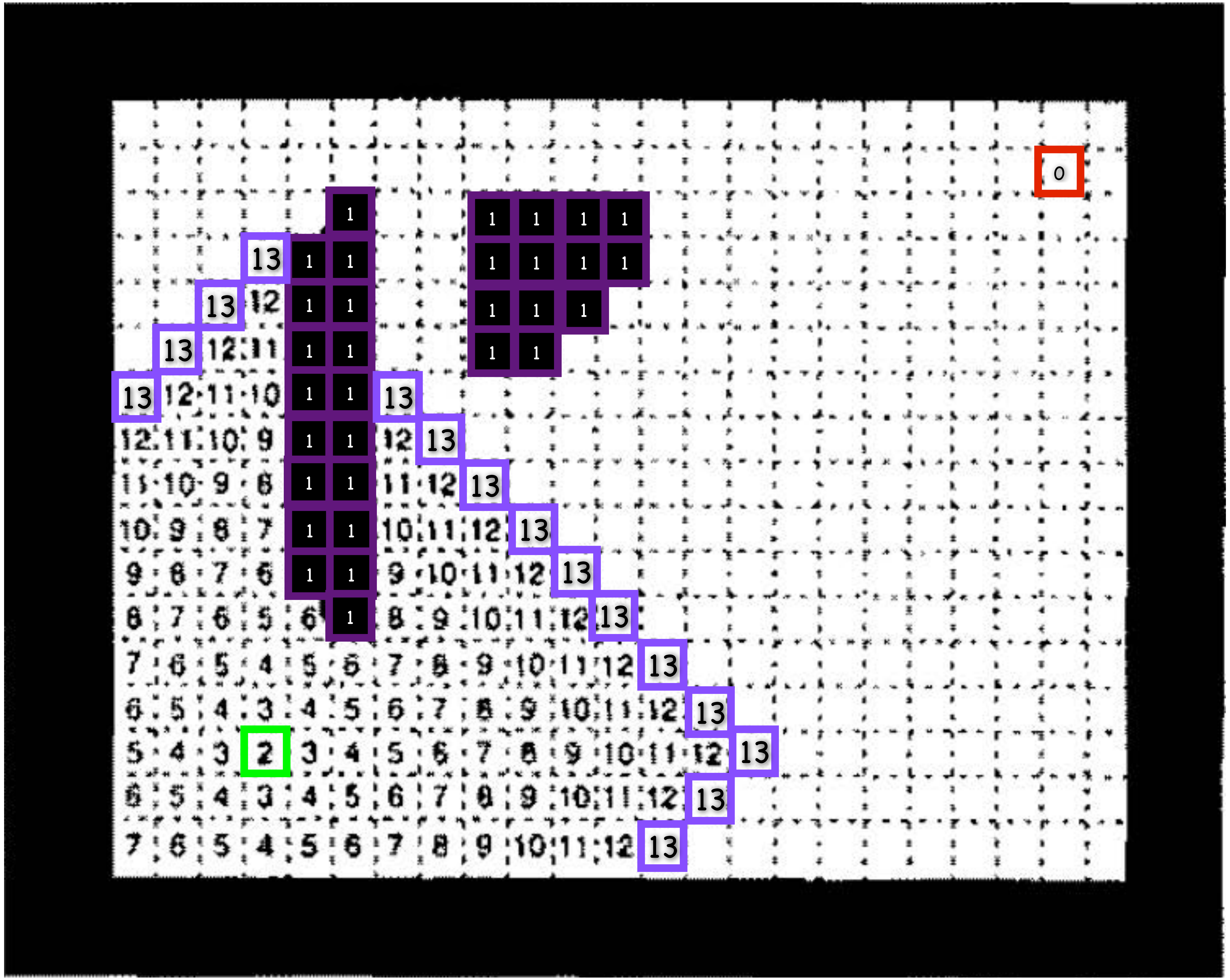
Brushfire away from goal:

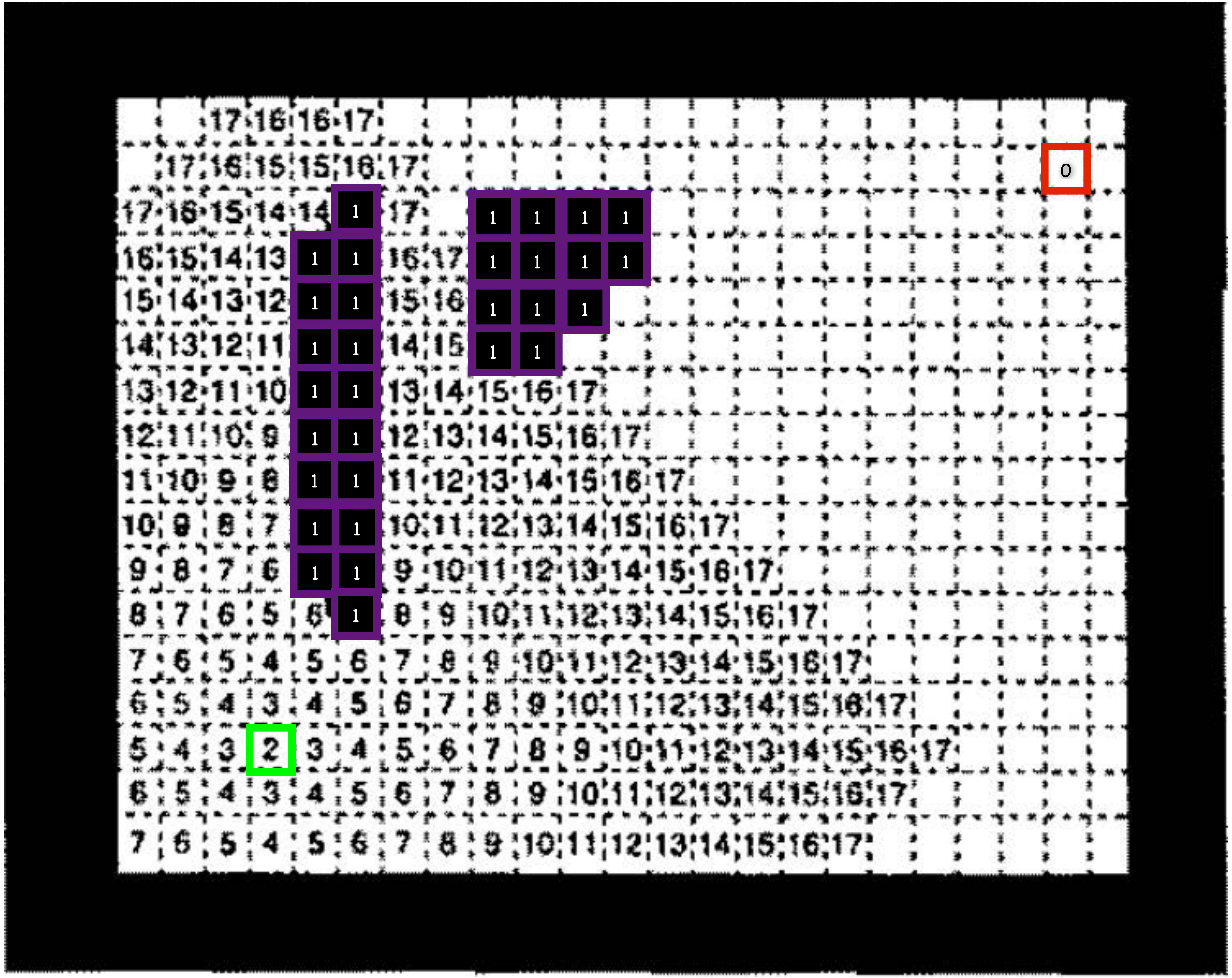
energy at empty cells on frontier = current energy at frontier + change in cell energy (1 in this case)

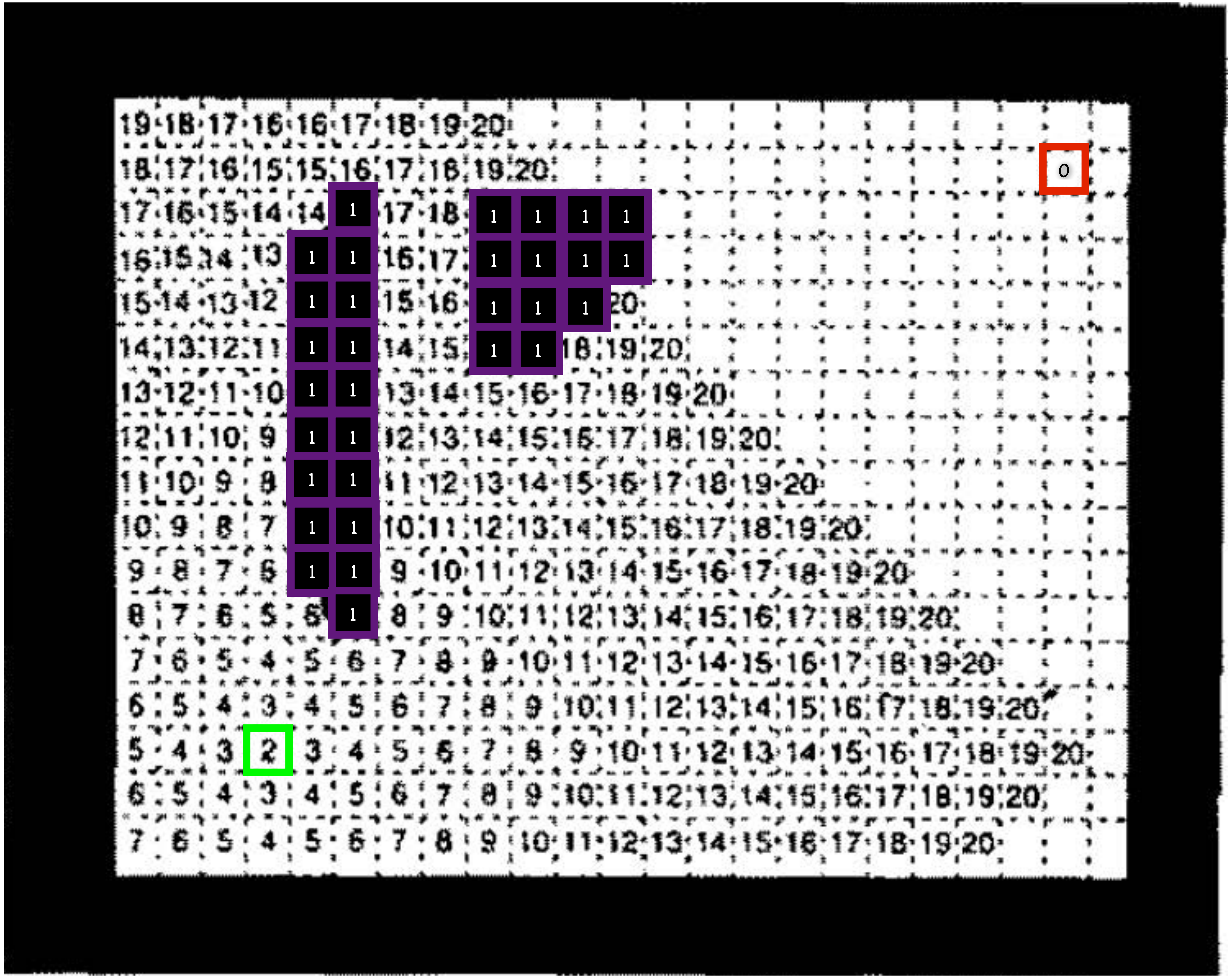


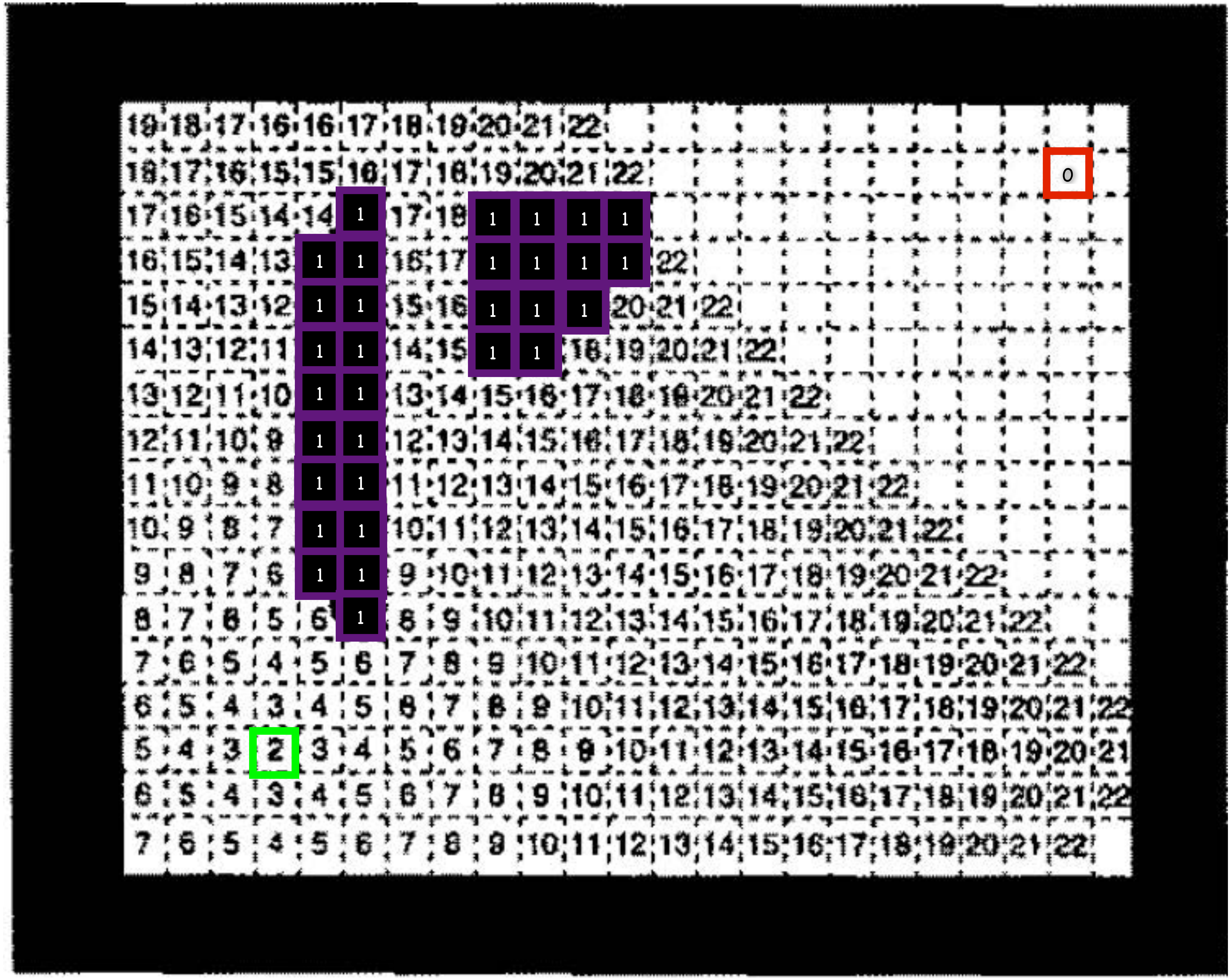
Brushfire away from goal:

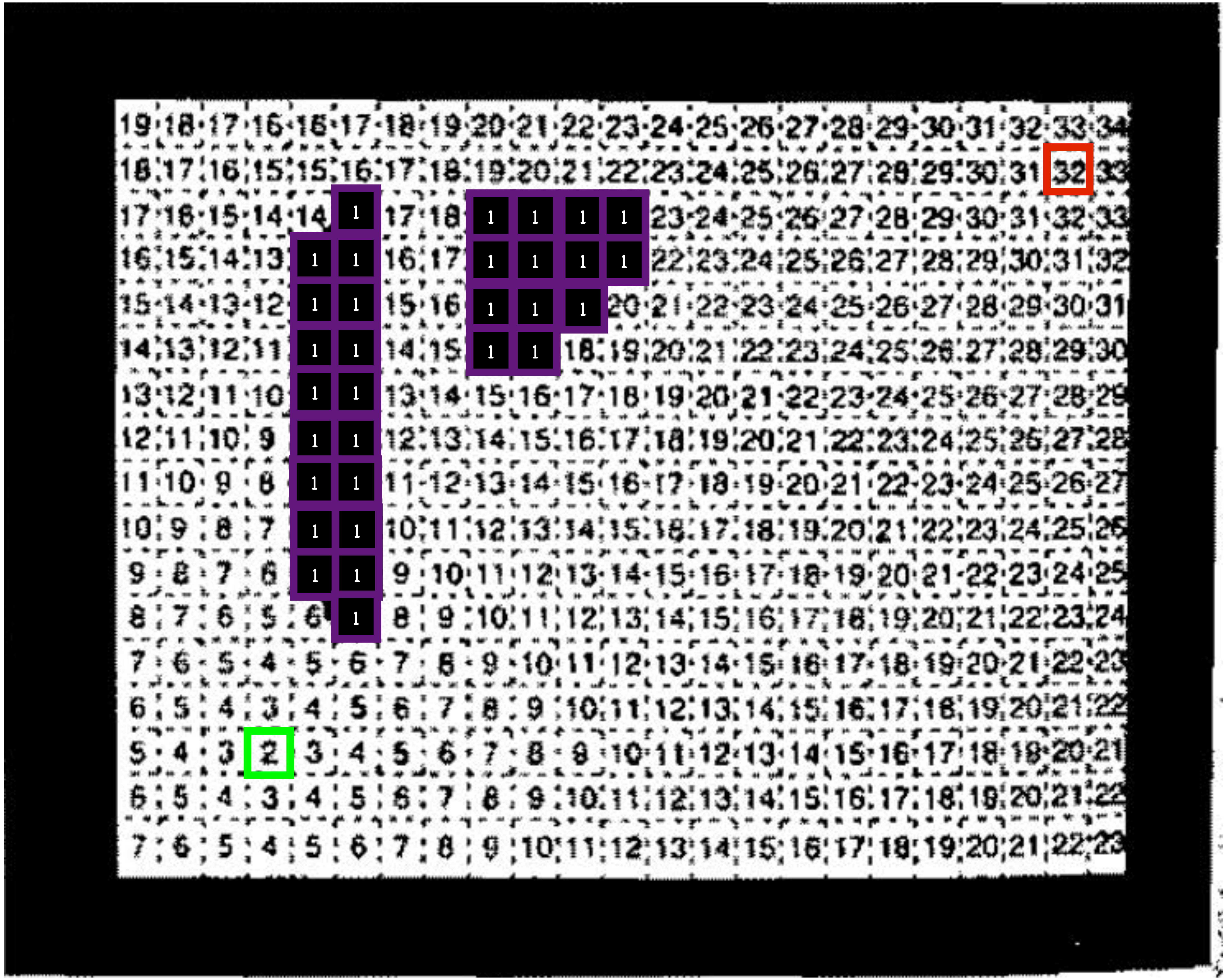
energy at empty cells on frontier = current energy at frontier + change in cell energy (1 in this case)



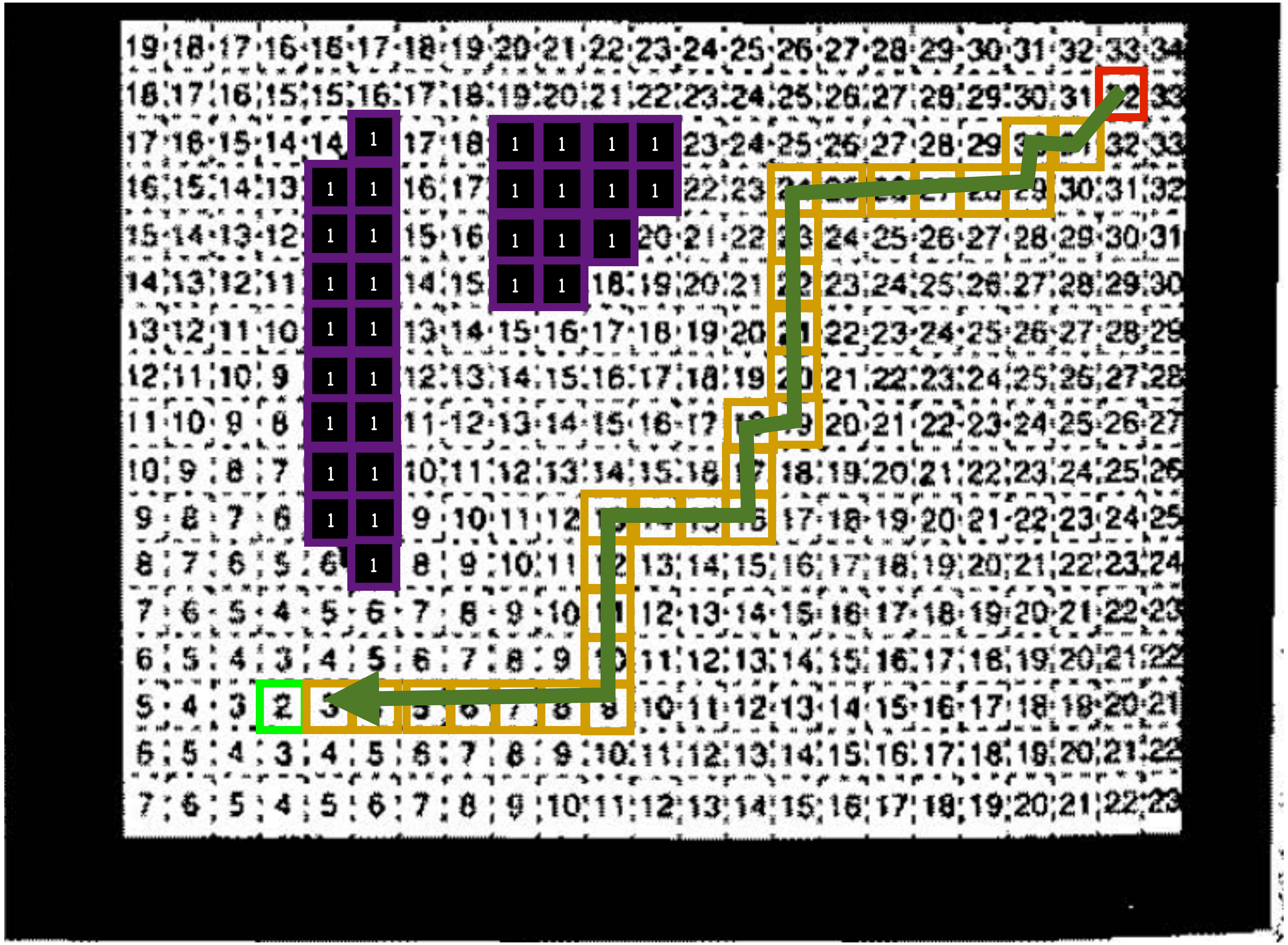






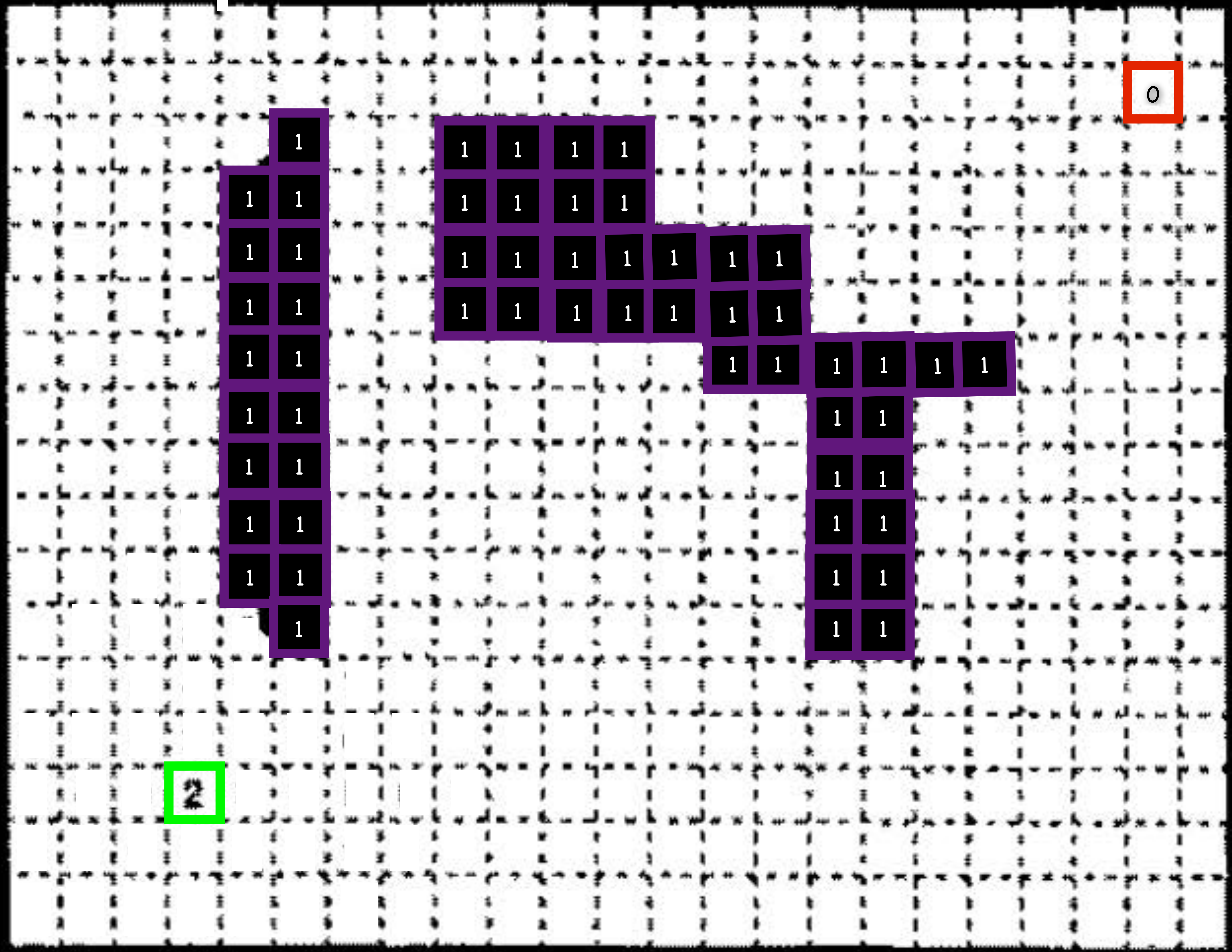


Once start reached, follow brushfire potential to goal

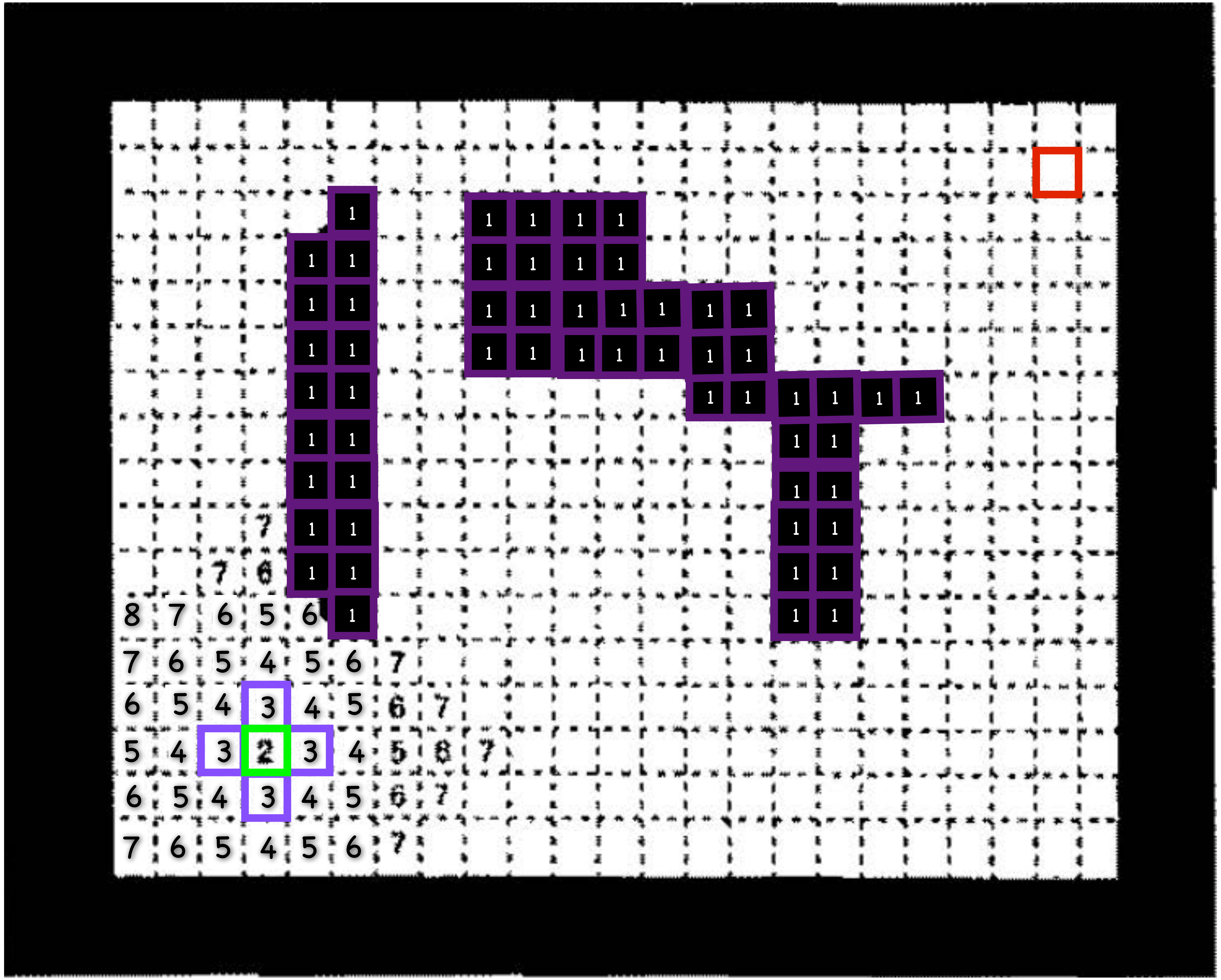


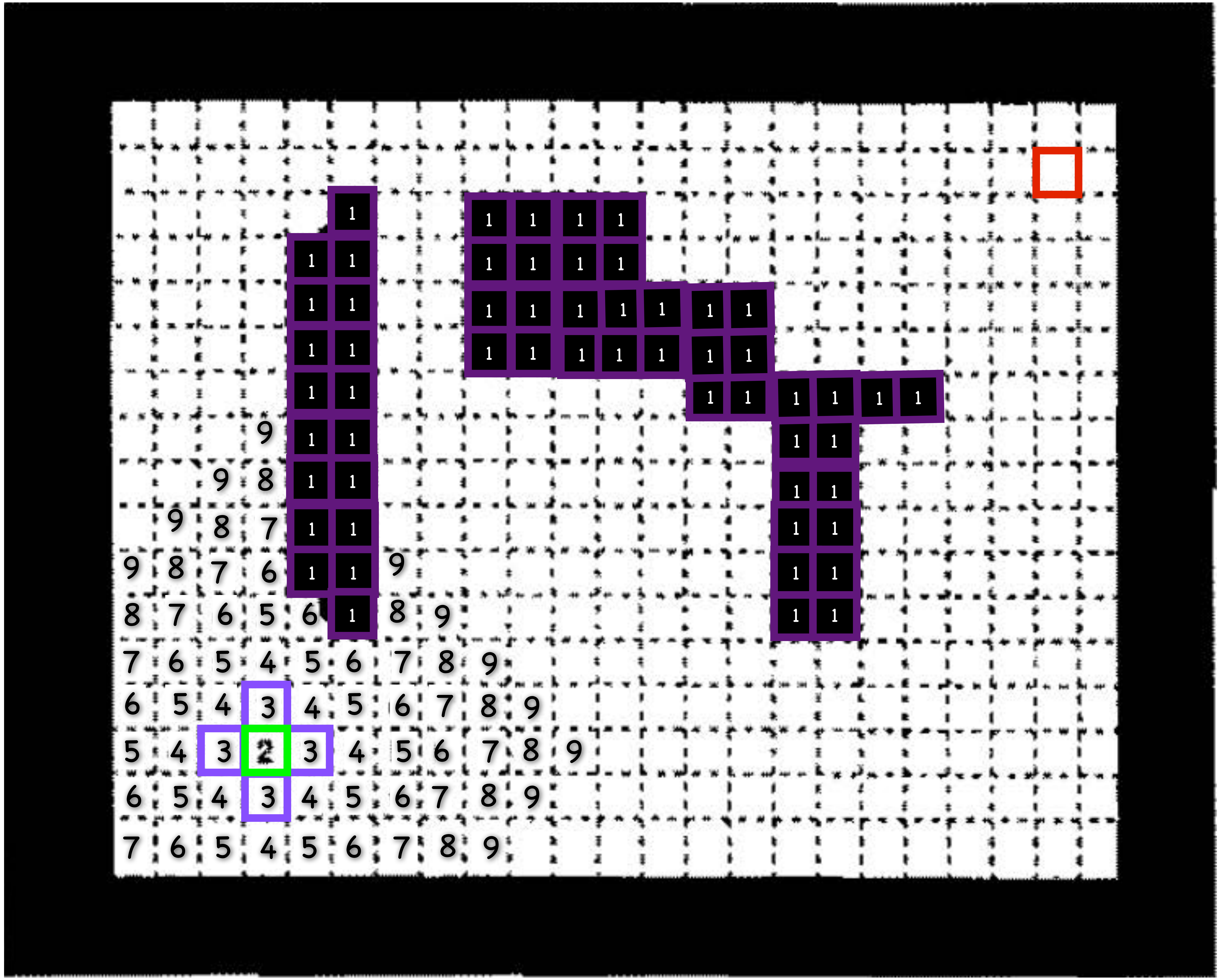
Example with Local Minima

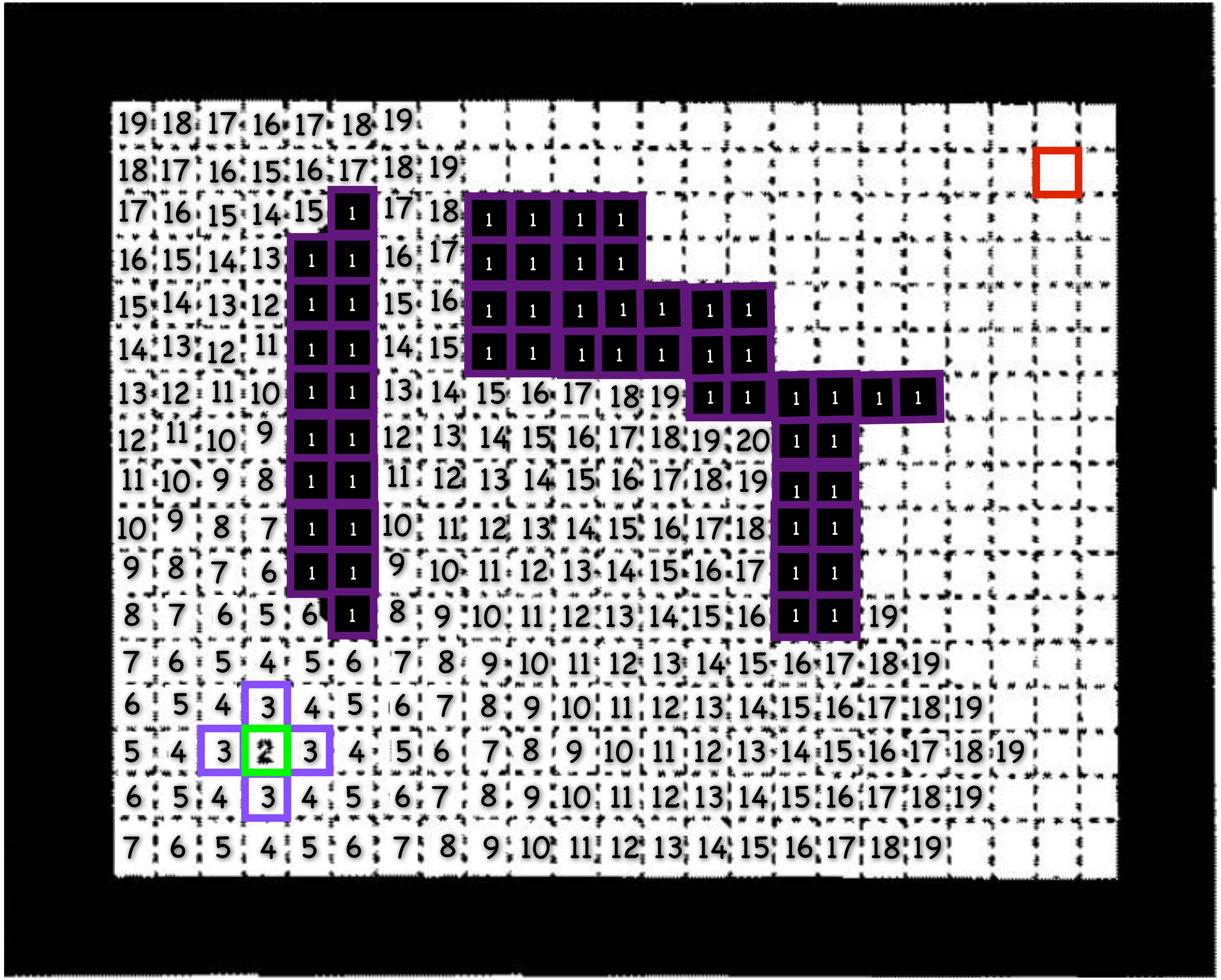
Example with Local Minima

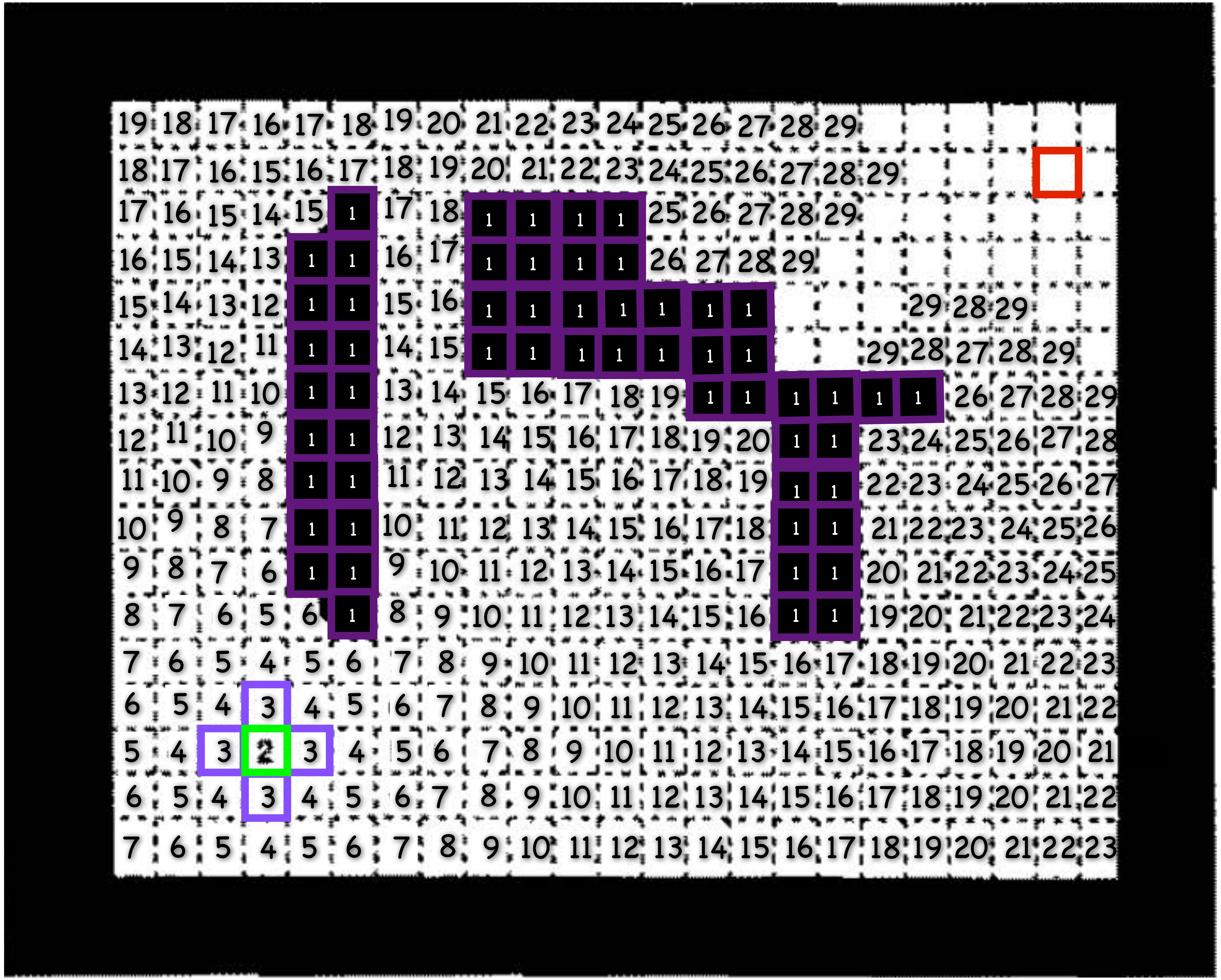


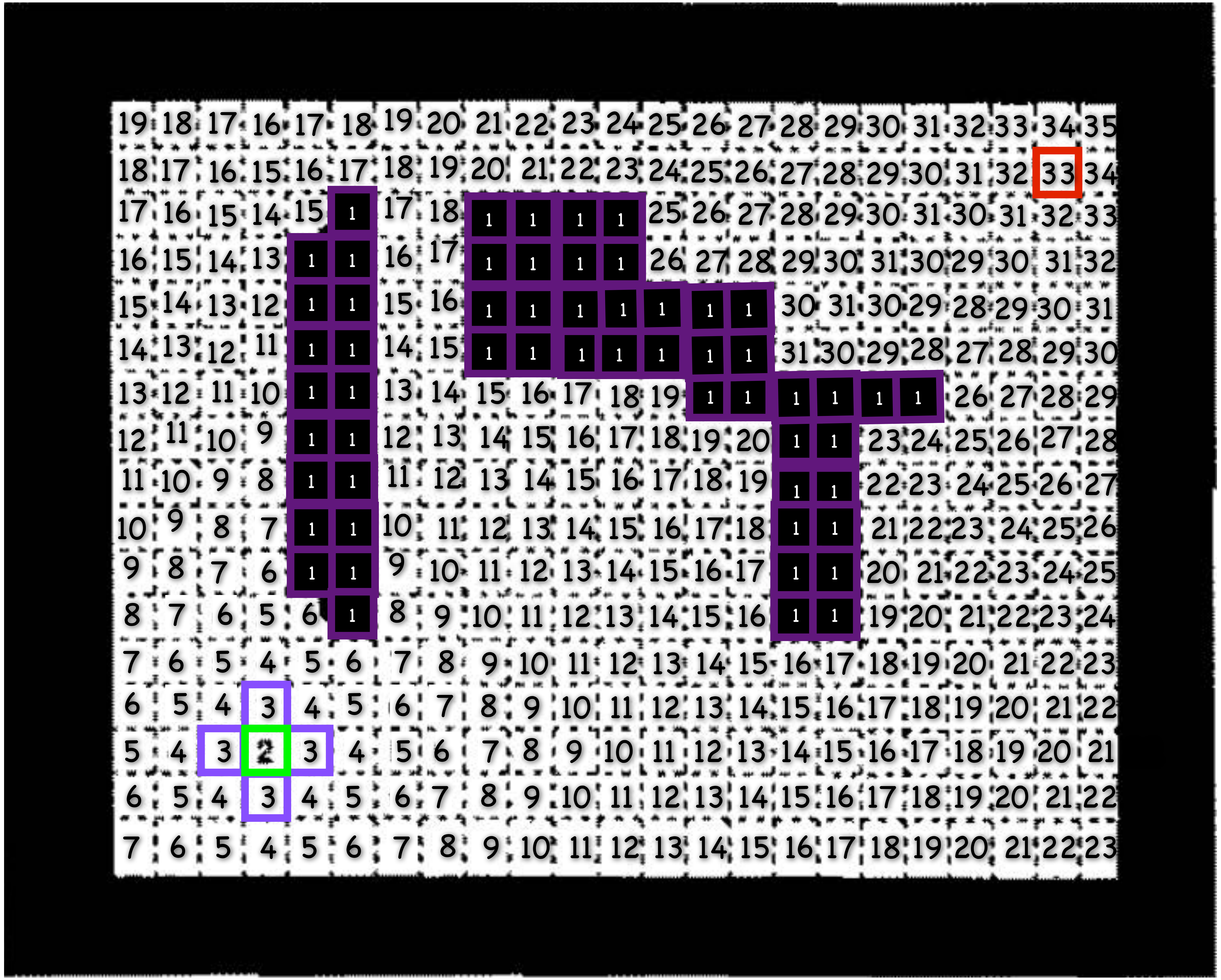
Slide borrowed from Michigan Robotics autorob.org

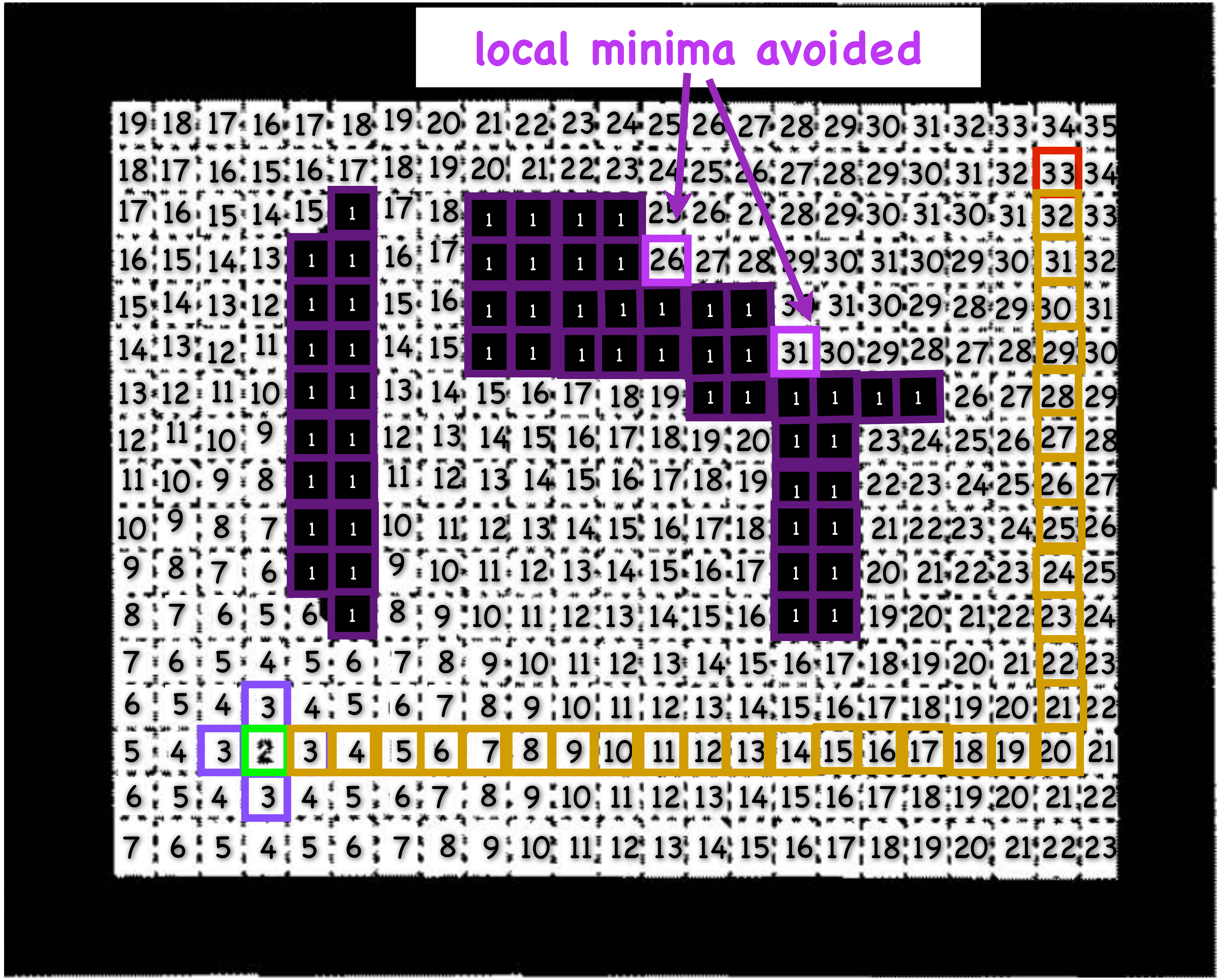



9
9 8
9 8 7
9 8 7 6 9
8 7 6 5 6 8 9
7 6 5 4 5 6 7 8 9
6 5 4 3 4 5 6 7 8 9
5 4 3 2 3 4 5 6 7 8 9
6 5 4 3 4 5 6 7 8 9
7 6 5 4 5 6 7 8 9

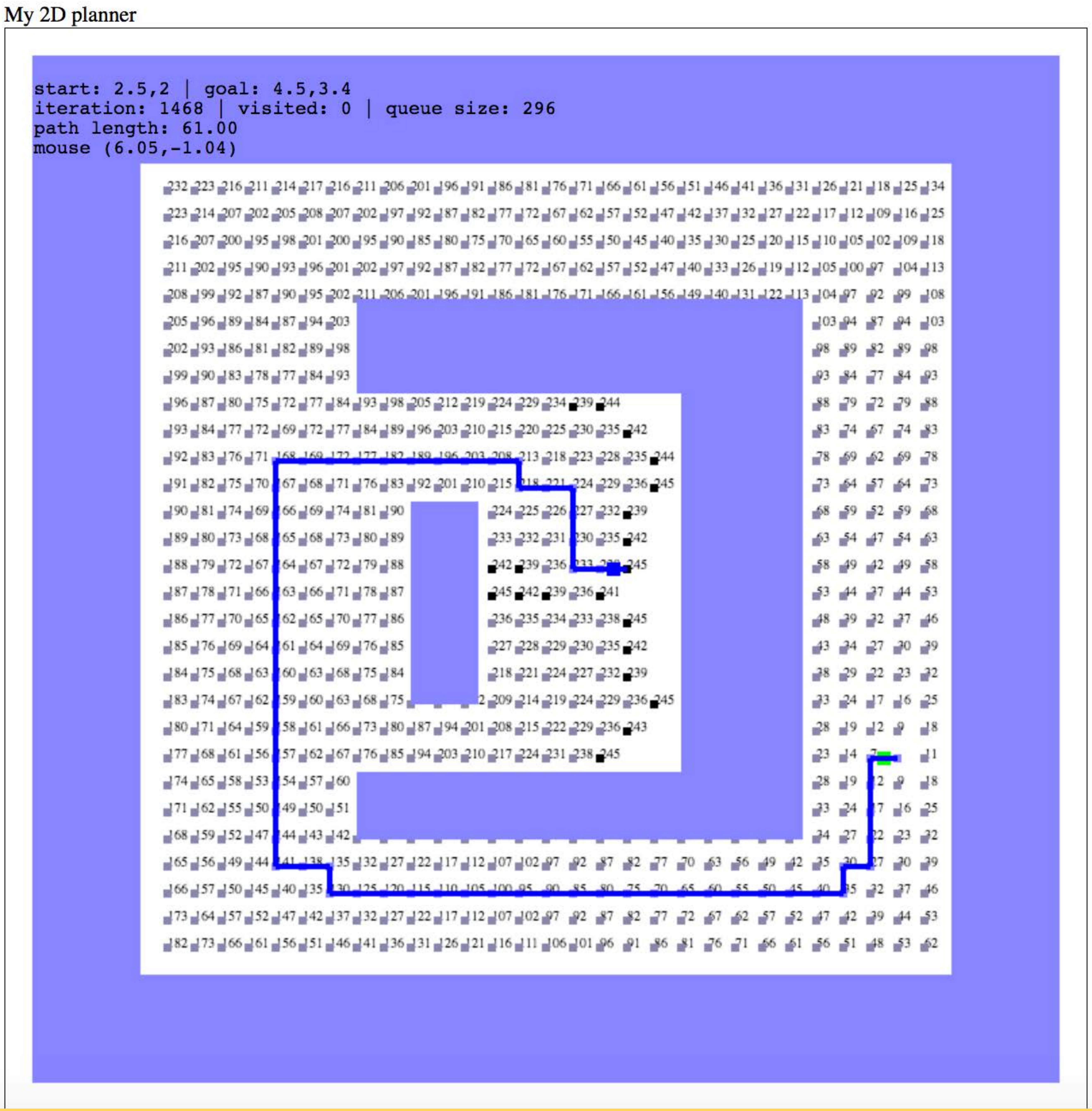

19 18 17 16 17 18 19
18 17 16 15 16 17 18 19
17 16 15 14 15 1 17 18 1 1 1 1
16 15 14 13 1 1 16 17 1 1 1 1
15 14 13 12 1 1 15 16 1 1 1 1 1 1 1
14 13 12 11 1 1 14 15 1 1 1 1 1 1 1
13 12 11 10 1 1 13 14 15 16 17 18 19 1 1 1 1 1 1
12 11 10 9 1 1 12 13 14 15 16 17 18 19 20 1 1
11 10 9 8 1 1 11 12 13 14 15 16 17 18 19 1 1
10 9 8 7 1 1 10 11 12 13 14 15 16 17 18 1 1
9 8 7 6 1 1 9 10 11 12 13 14 15 16 17 1 1
8 7 6 5 6 1 8 9 10 11 12 13 14 15 16 1 1 19
7 6 5 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
6 5 4 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
5 4 3 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
6 5 4 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
7 6 5 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19




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 1 1 16 17 1 1 1 1 26 27 28 29 30 31 30 29 30 31 32
15 14 13 12 1 1 15 16 1 1 1 1 1 1 1 30 31 30 29 28 29 30 31
14 13 12 11 1 1 14 15 1 1 1 1 1 1 1 31 30 29 28 27 28 29 30
13 12 11 10 1 1 13 14 15 16 17 18 19 1 1 1 1 1 1 26 27 28 29
12 11 10 9 1 1 12 13 14 15 16 17 18 19 20 1 1 23 24 25 26 27 28
11 10 9 8 1 1 11 12 13 14 15 16 17 18 19 1 1 22 23 24 25 26 27
10 9 8 7 1 1 10 11 12 13 14 15 16 17 18 1 1 21 22 23 24 25 26
9 8 7 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


local minima avoided

Kineval wavefront planner



Planning Recap

Recap

- 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

Next Lecture
Motion Control