Lecture 13
Planning - V - Potential Fields

Course Logistics

- **Quiz 11 was posted today and was due before the lecture.**
- Project 3 was posted on 10/11 and is due on 10/25.
  - Please look into updates posted by Chahyon Ku on Ed discussions!

RRT Algorithm

Slide borrowed from Michigan Robotics autorob.org

[Kuffner, LaValle 2000]

RRT Algorithm

Extend graph towards a random configuration and repeat

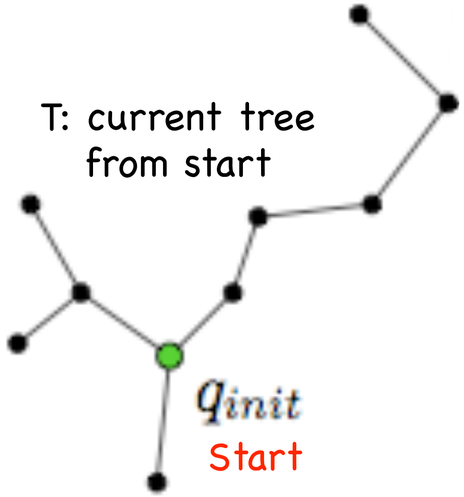
BUILD_RRT(q_{init})
1 $\mathcal{T}$.init(q_{init});
2 **for** $k = 1$ **to** K **do**
3 $q_{rand} \leftarrow$ RANDOM_CONFIG();
4 EXTEND($\mathcal{T}, q_{rand}$);
5 Return $\mathcal{T}$

[Kuffner, LaValle 2000]

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 ← RANDOM_CONFIG();
4      EXTEND(T, q_rand);
5  Return T
```



q_{rand}

Extension Goal
(randomly
selected)

Figure 3: The EXTEND operation.

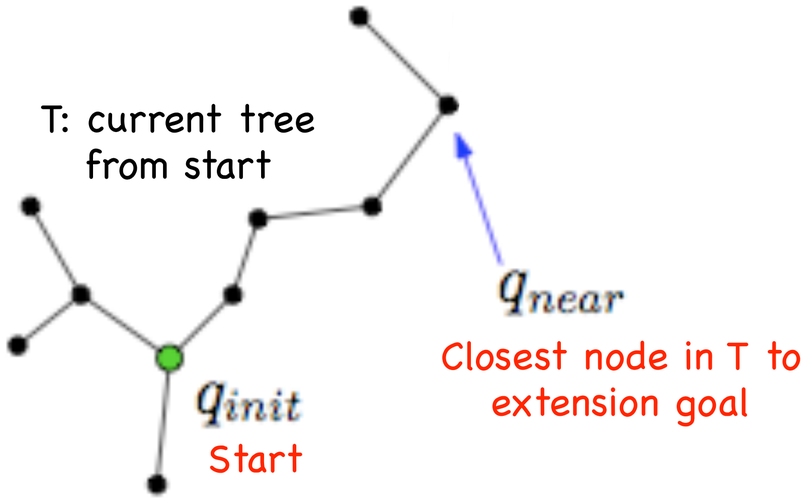
[Kuffner, LaValle 2000]

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 ← RANDOM_CONFIG();
4      EXTEND(T, q_rand);
5  Return T
```

```
EXTEND(T, q)
1  q_near ← 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;
```



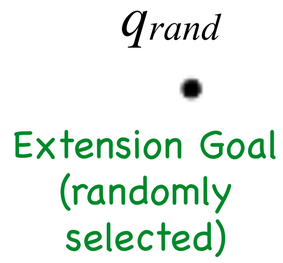


Figure 3: The EXTEND operation.

Extend graph towards a random configuration

[Kuffner, LaValle 2000]

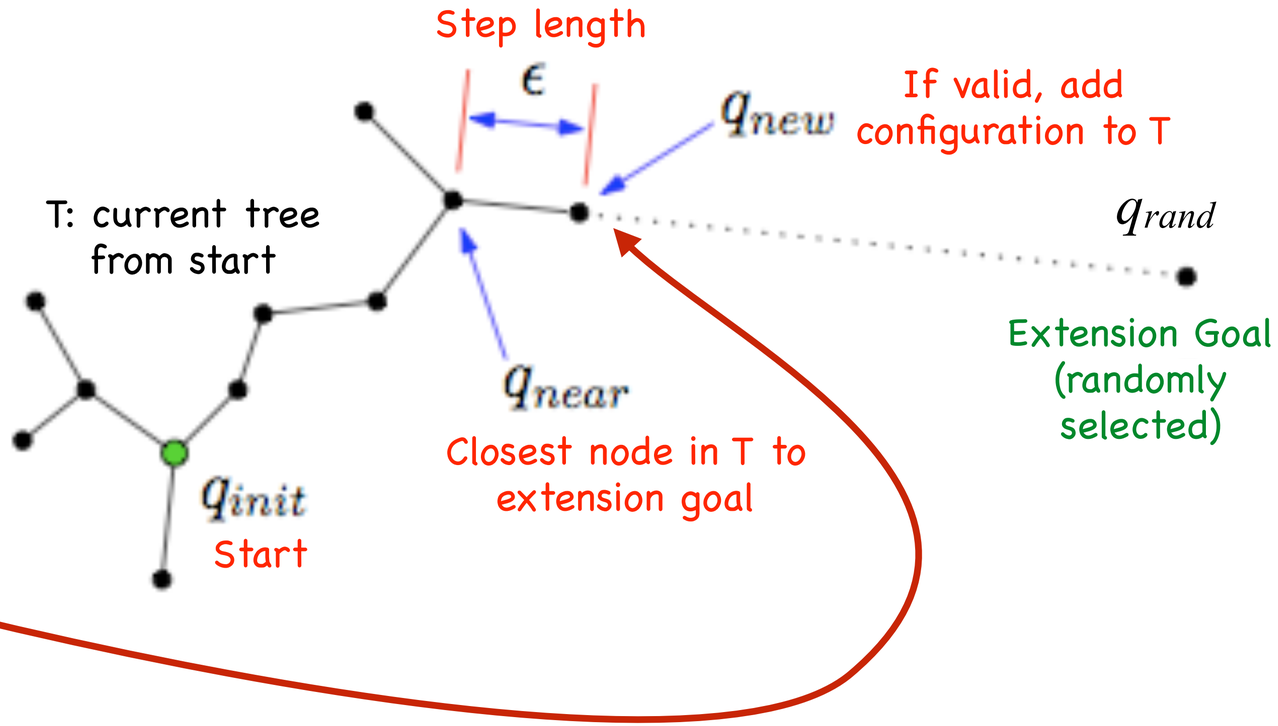
RRT Algorithm

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 ← RANDOM_CONFIG();
4      EXTEND(T, q_rand);
5  Return T
```

```
EXTEND(T, q)
1  q_near ← 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* Algorithm

Slide borrowed from Michigan Robotics autorob.org

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 $G = (V, E)$;

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

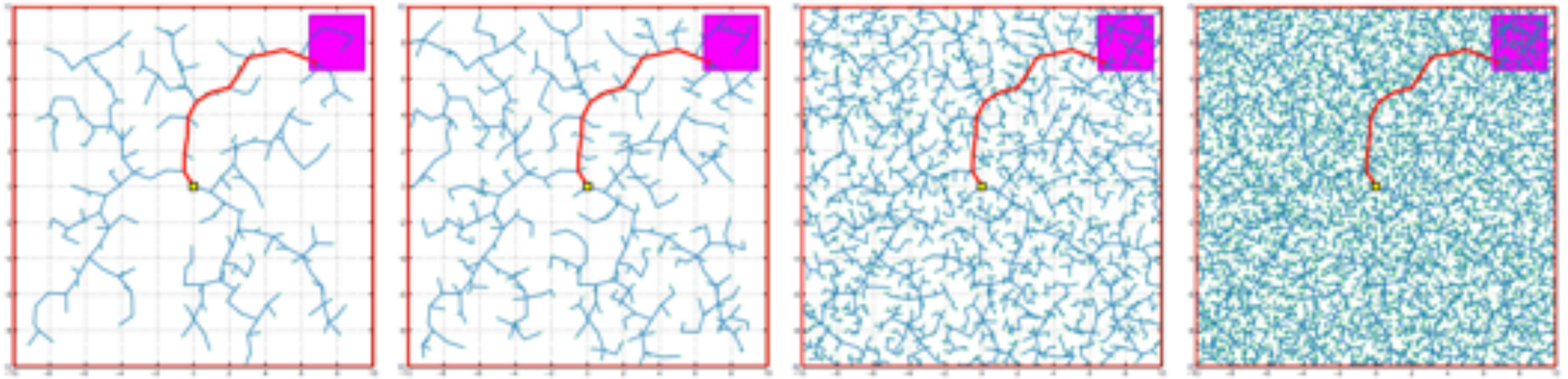
RRT*

- 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

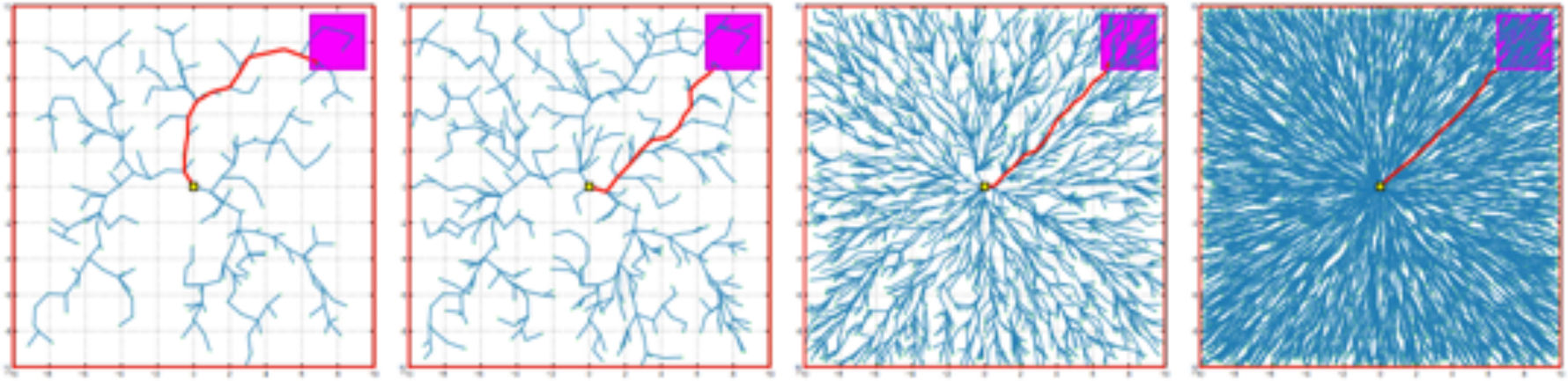
Demonstration - https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

RRT*

RRT



RRT*

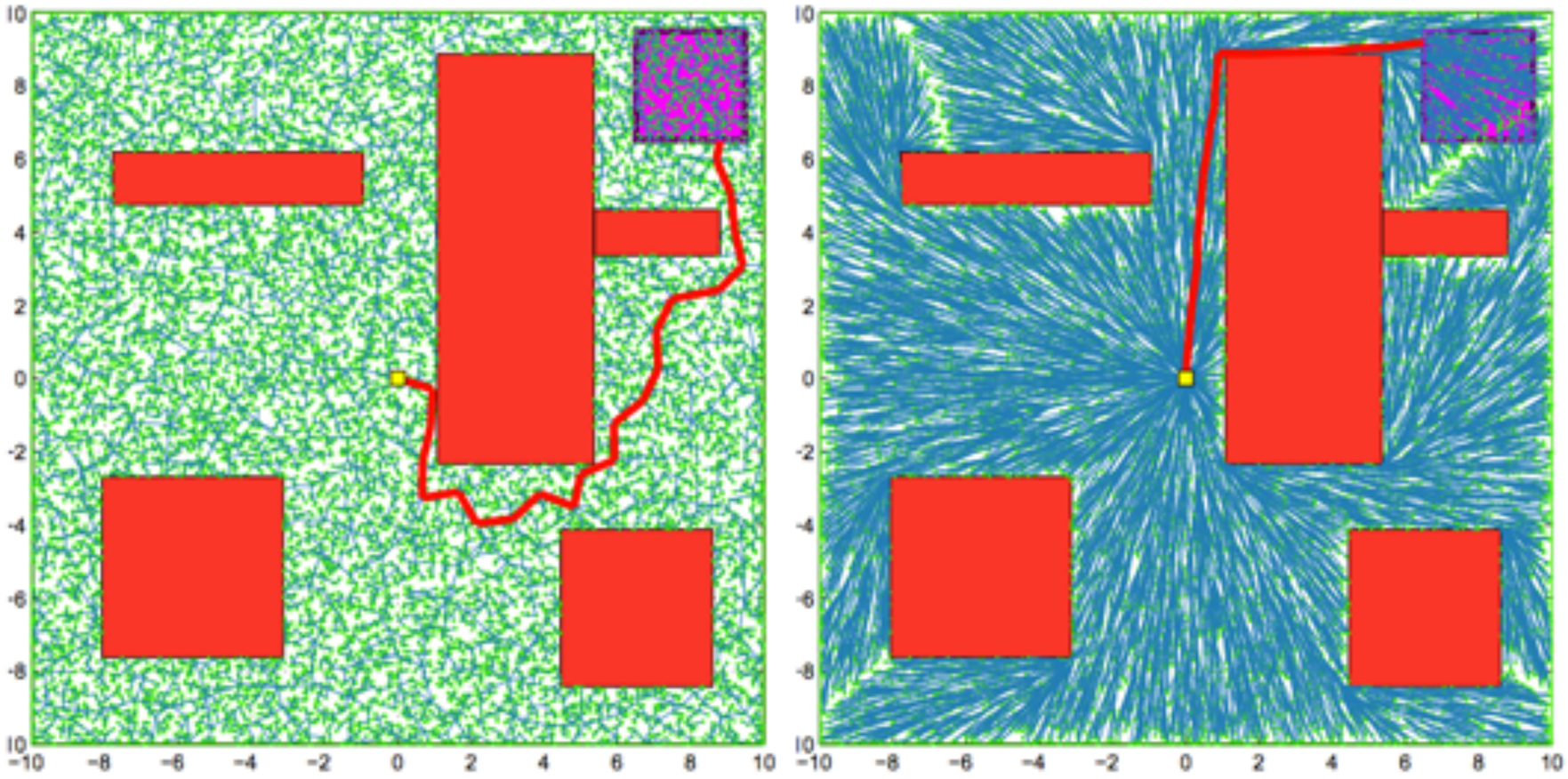


Source: Karaman and Frazzol

RRT*

RRT

RRT*



Source: Karaman and Frazzoli

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

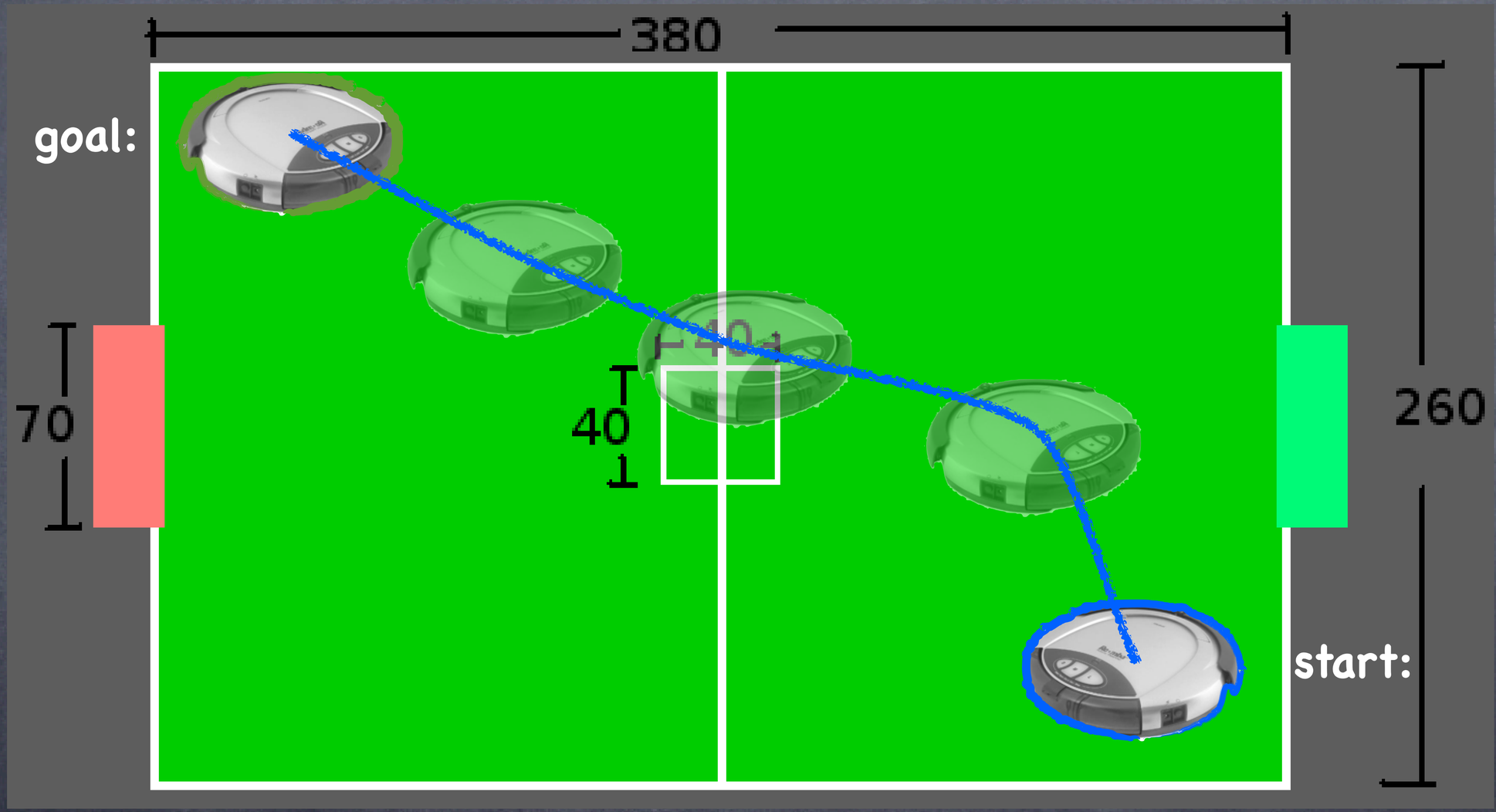
- Shortcutting:
  - along the found path, pick two vertices x_{t1}, x_{t2} and try to connect them directly (skipping over all intermediate vertices)
- Nonlinear optimization for optimal control
  - Allows to specify an objective function that includes smoothness in state, control, small control inputs,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

Slide borrowed from Dieter Fox

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")

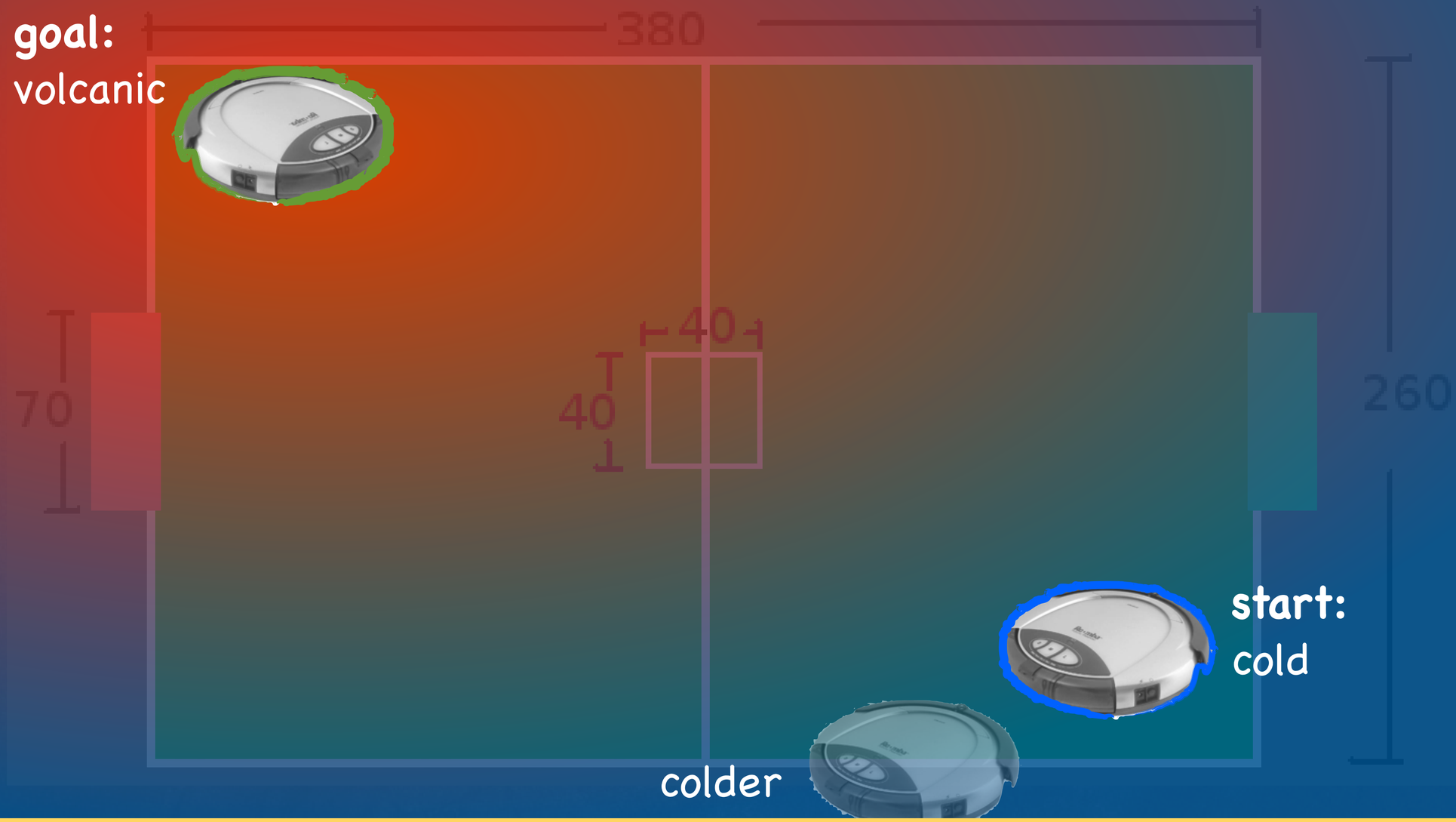
goal:
volcanic

start:
cold

Slide borrowed from Michigan Robotics autorob.org

Potential field

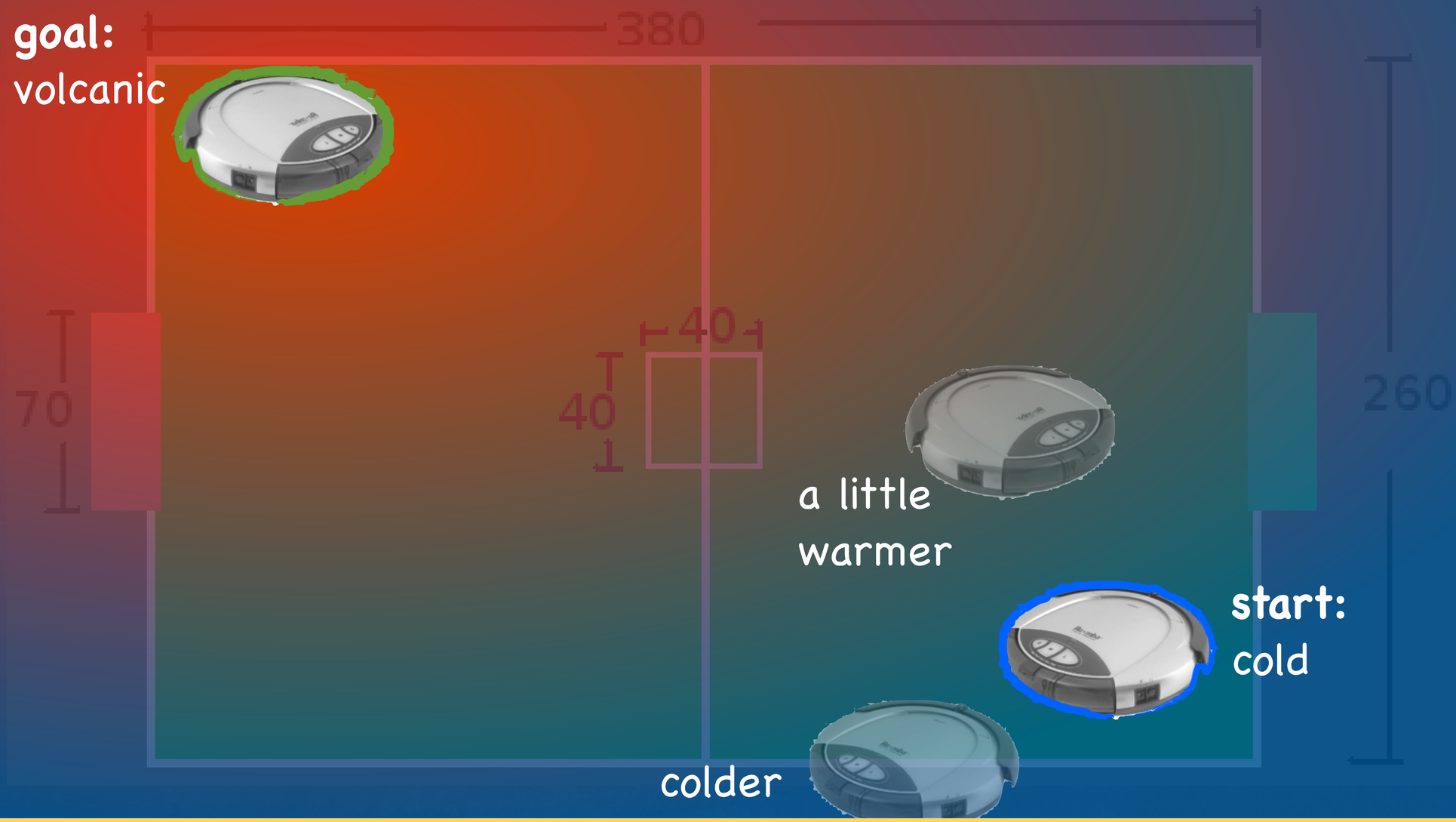
(like a game of "warmer-colder")



Slide borrowed from Michigan Robotics autorob.org

Potential field

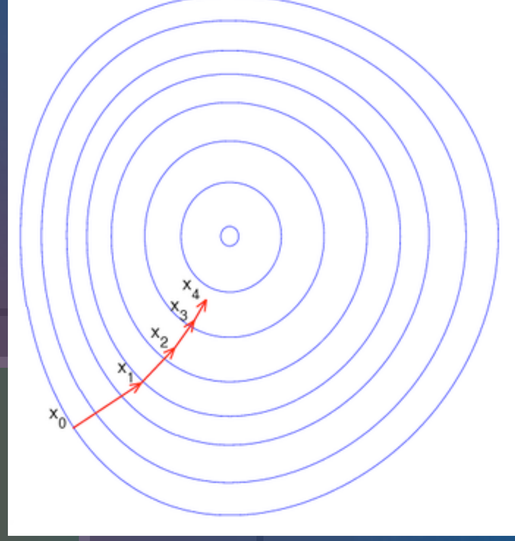
(like a game of "warmer-colder")



Slide borrowed from Michigan Robotics autorob.org

Potential field

(like a game of "warmer-colder")



Gradient descent:
Energy potential converges at goal

goal:
volcanic

a little warmer

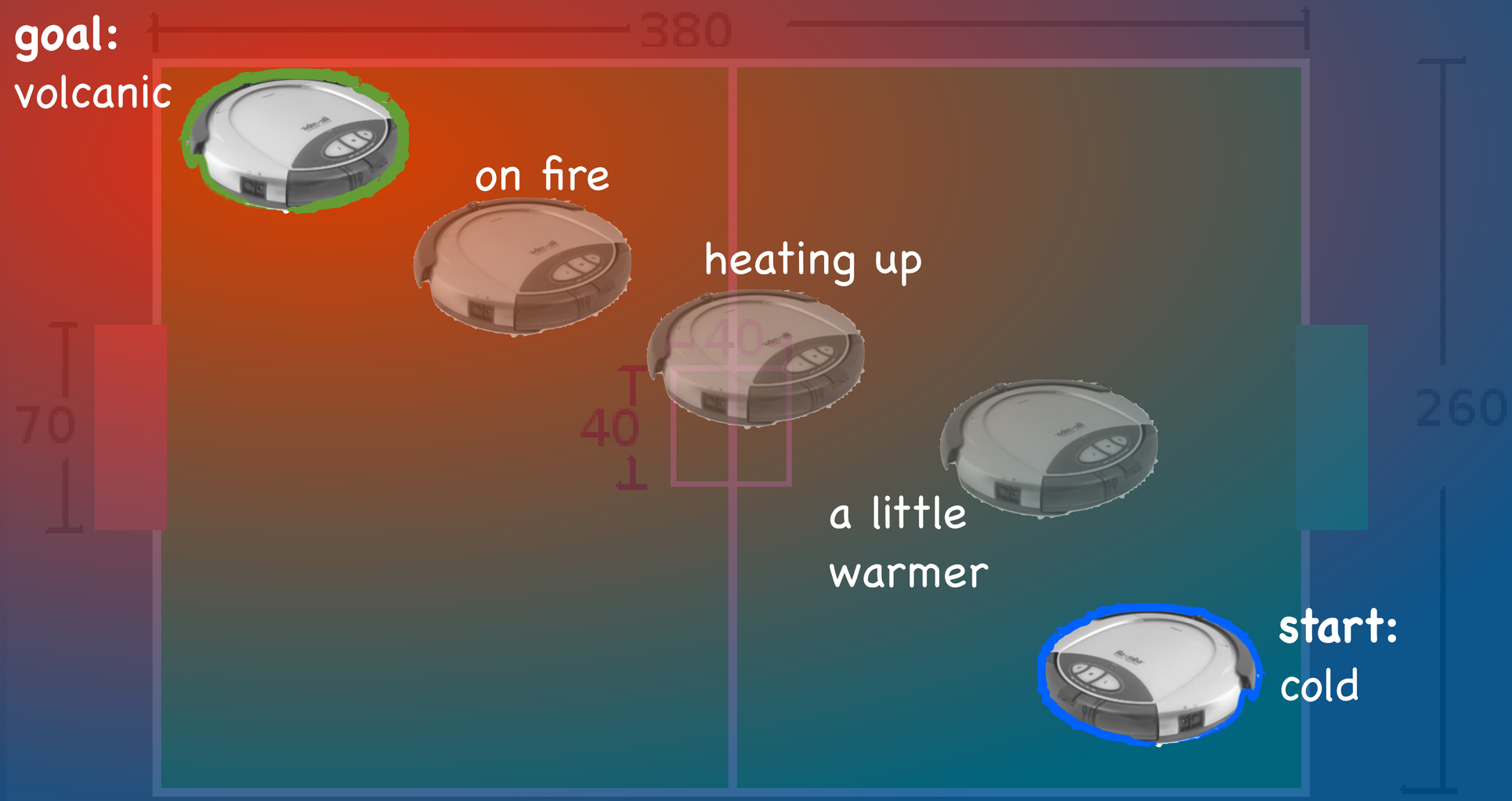
start:
cold

colder

Slide borrowed from Michigan Robotics autorob.org

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:

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

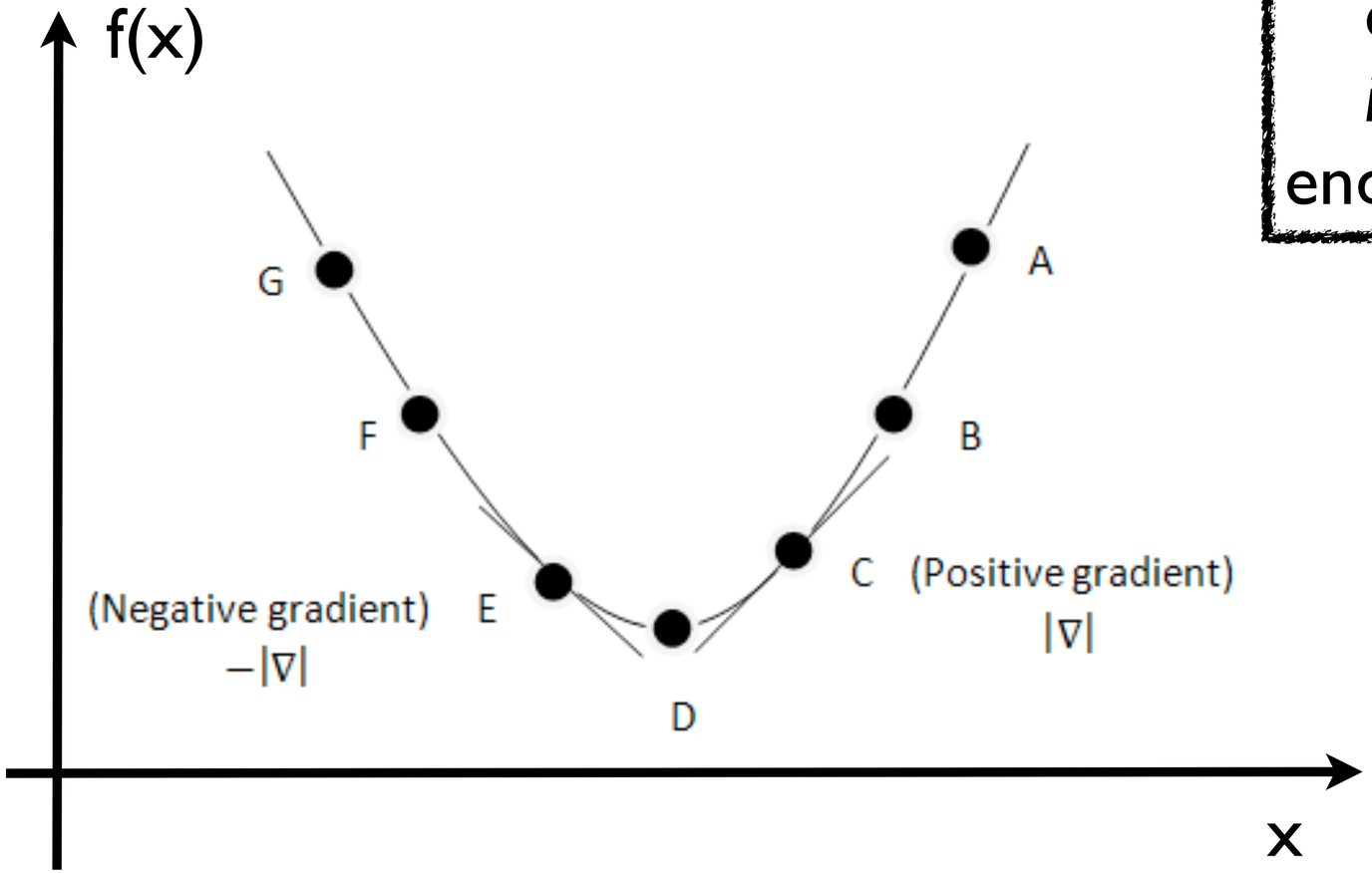
$i \leftarrow 0$

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

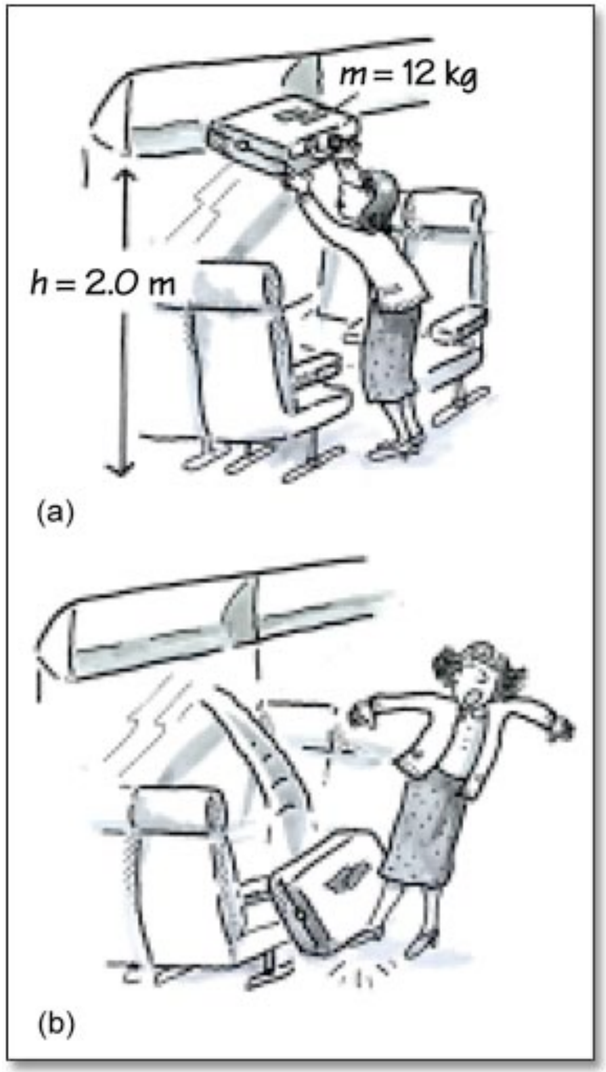


Derivative assumed to be direction of steepest ascent away from goal

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

Slide borrowed from Michigan Robotics autorob.org

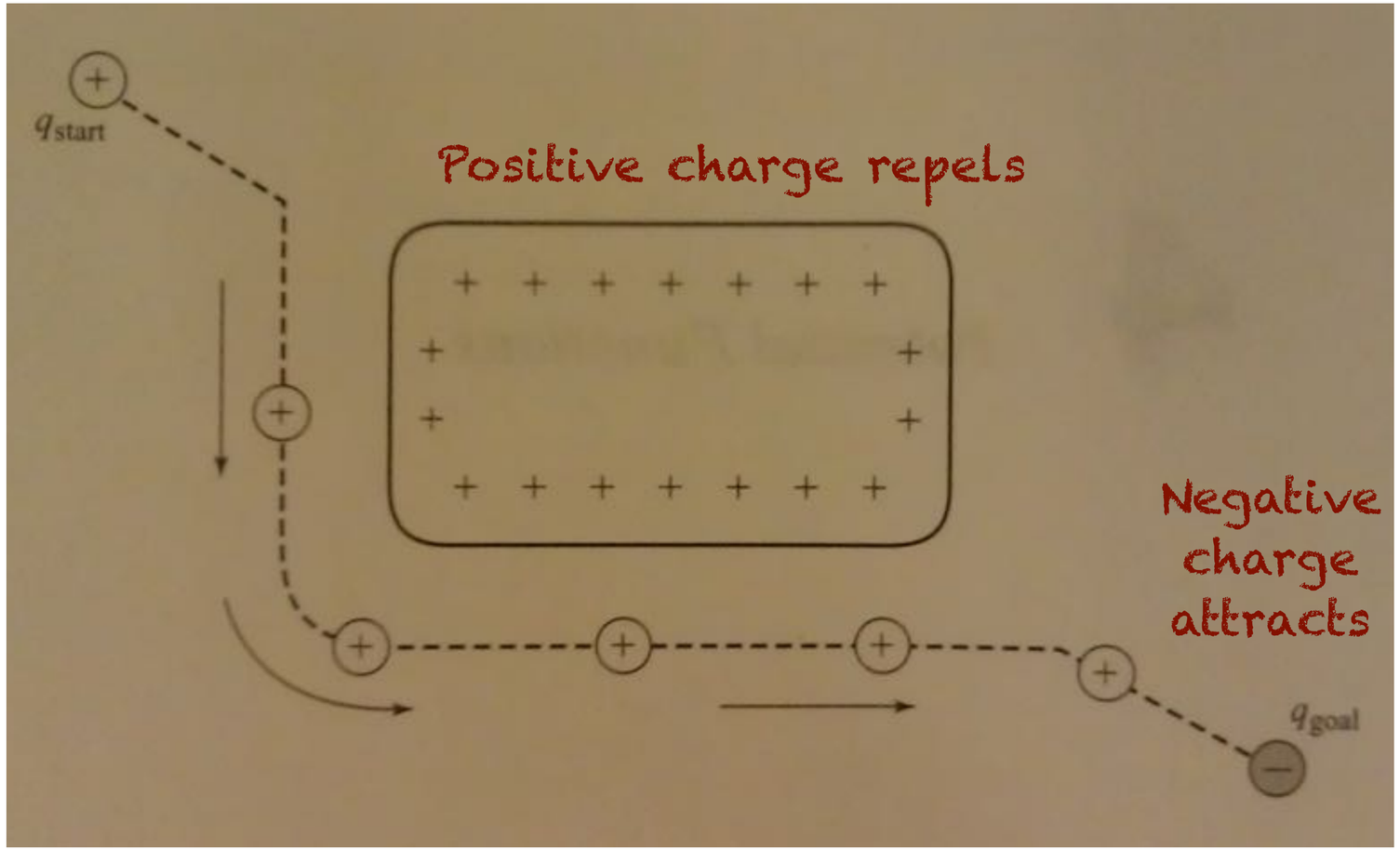
Potential Energy



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

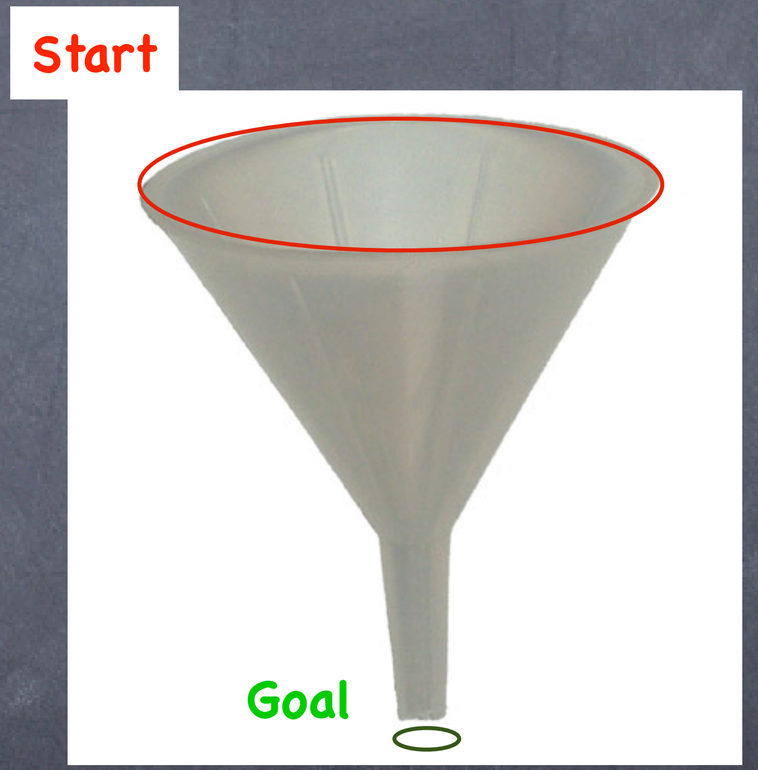
Charged Particle Example

Positively charged particle follows potential energy to goal



Convergent Potentials

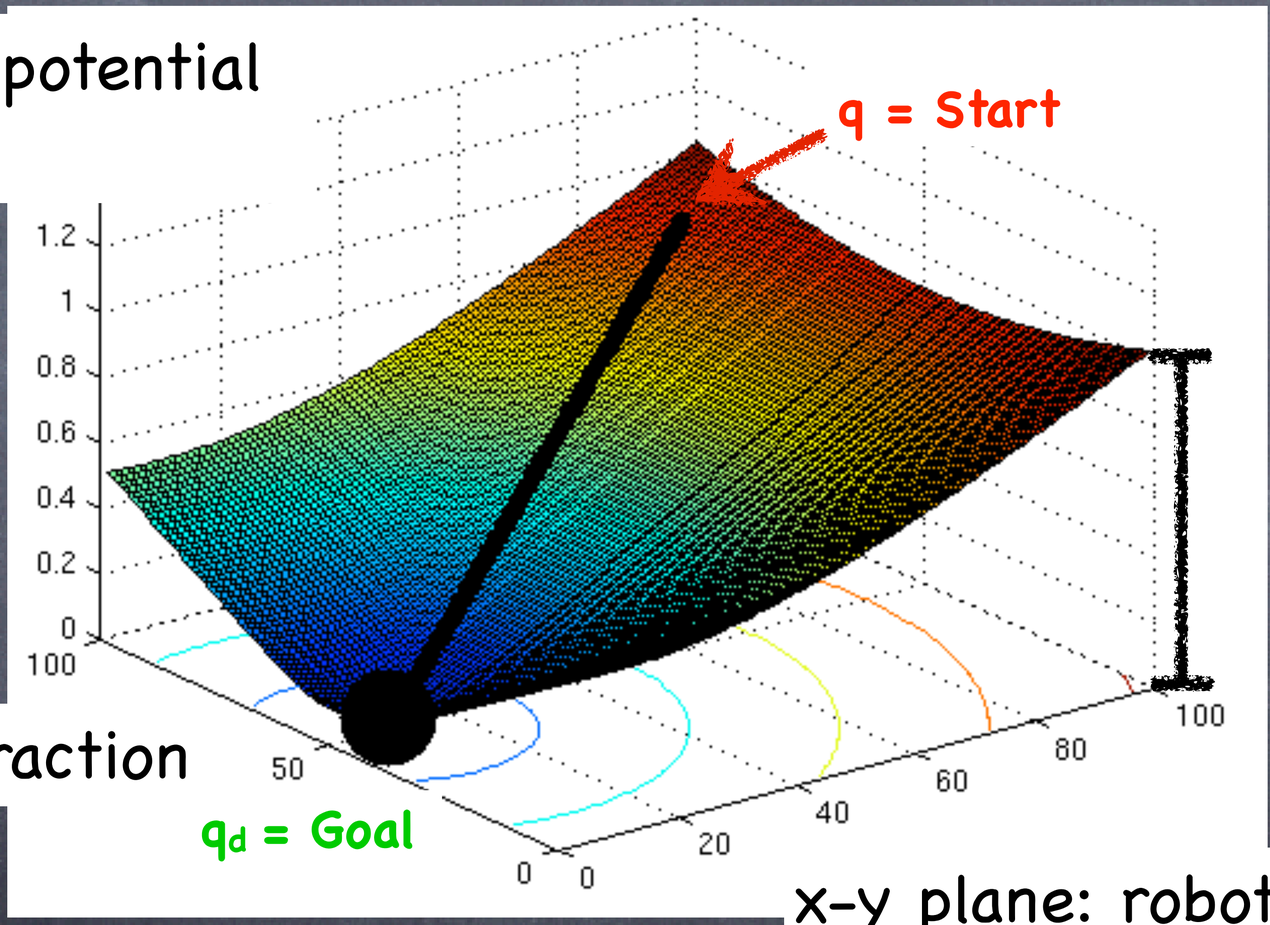
let's call these "attractor landscapes"



basin of attraction

2D potential navigation

z: height indicates potential at location



basin of attraction

x-y plane: robot position

Slide borrowed from Michigan Robotics autorob.org

"Cone" Attractor

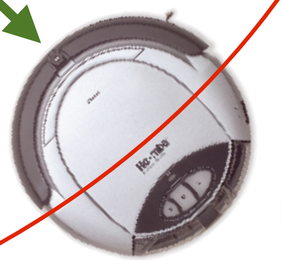
top view

q_d = Goal

"Attractor"

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

q = Start



w: weight

$(q - q_d)$: direction

$\|q - q_d\|$: distance

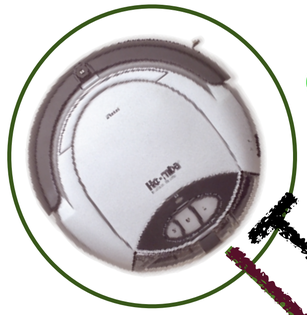
Start

Goal

side view

"Cone" Attractor

top view



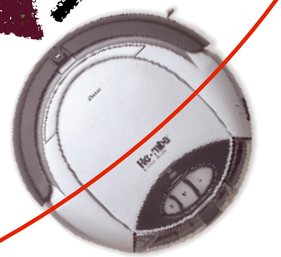
q_d = Goal

"Attractor"

$\|q - q_d\|$

$(q - q_d)$

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



q = Start

w: weight

$(q - q_d)$: direction

$\|q - q_d\|$: distance

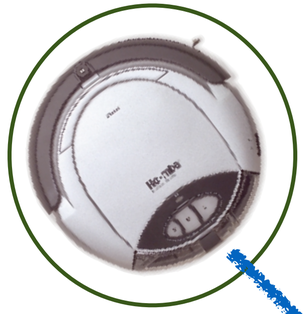
Start

Goal

side view

"Cone" Attractor

top view



q_d = Goal

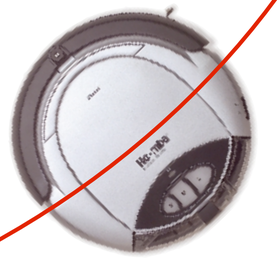
"Attractor"

unit vector

$(q - q_d)/\|q - q_d\|$

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

q = Start



w: weight

$(q_d - q)$: direction

$\|q_d - q\|$: distance

Start

Goal

side view

"Cone" Attractor

top view

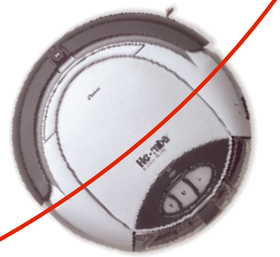
q_d = Goal

"Attractor"

w

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

x = Start



w: weight (< 1)

$(q - q_d)$: direction

$\|q - q_d\|$: distance

Start

Goal

side view

Can we modulate the range of a potential field?

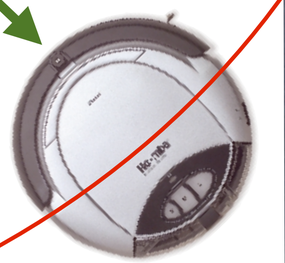
Slide borrowed from Michigan Robotics autorob.org

"Bowl" Attractor

top view

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

q_d = Goal



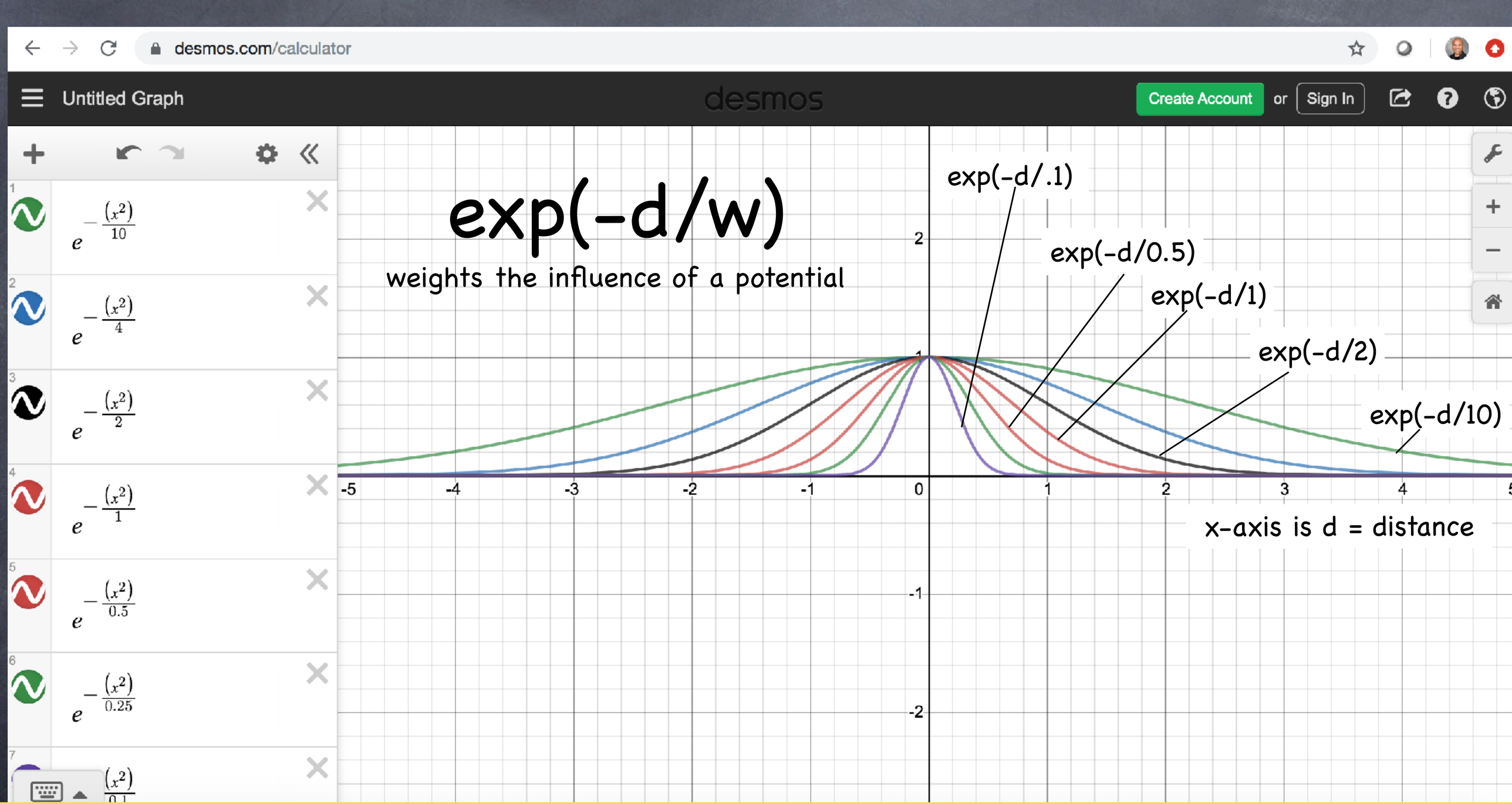
q = Start

Start

Goal

side view

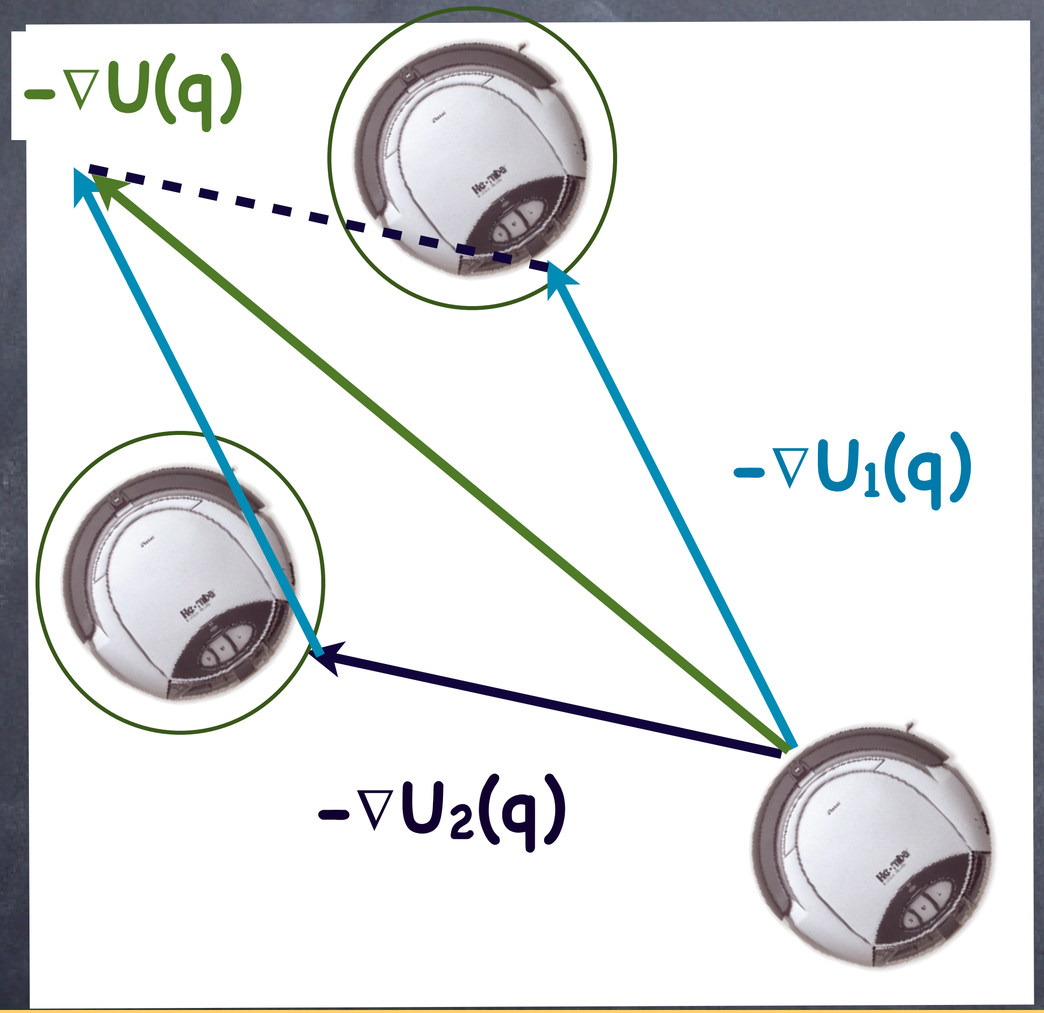
Slide borrowed from Michigan Robotics autorob.org



Slide borrowed from Michigan Robotics autorob.org

Can we combine multiple potentials?

Multiple potentials

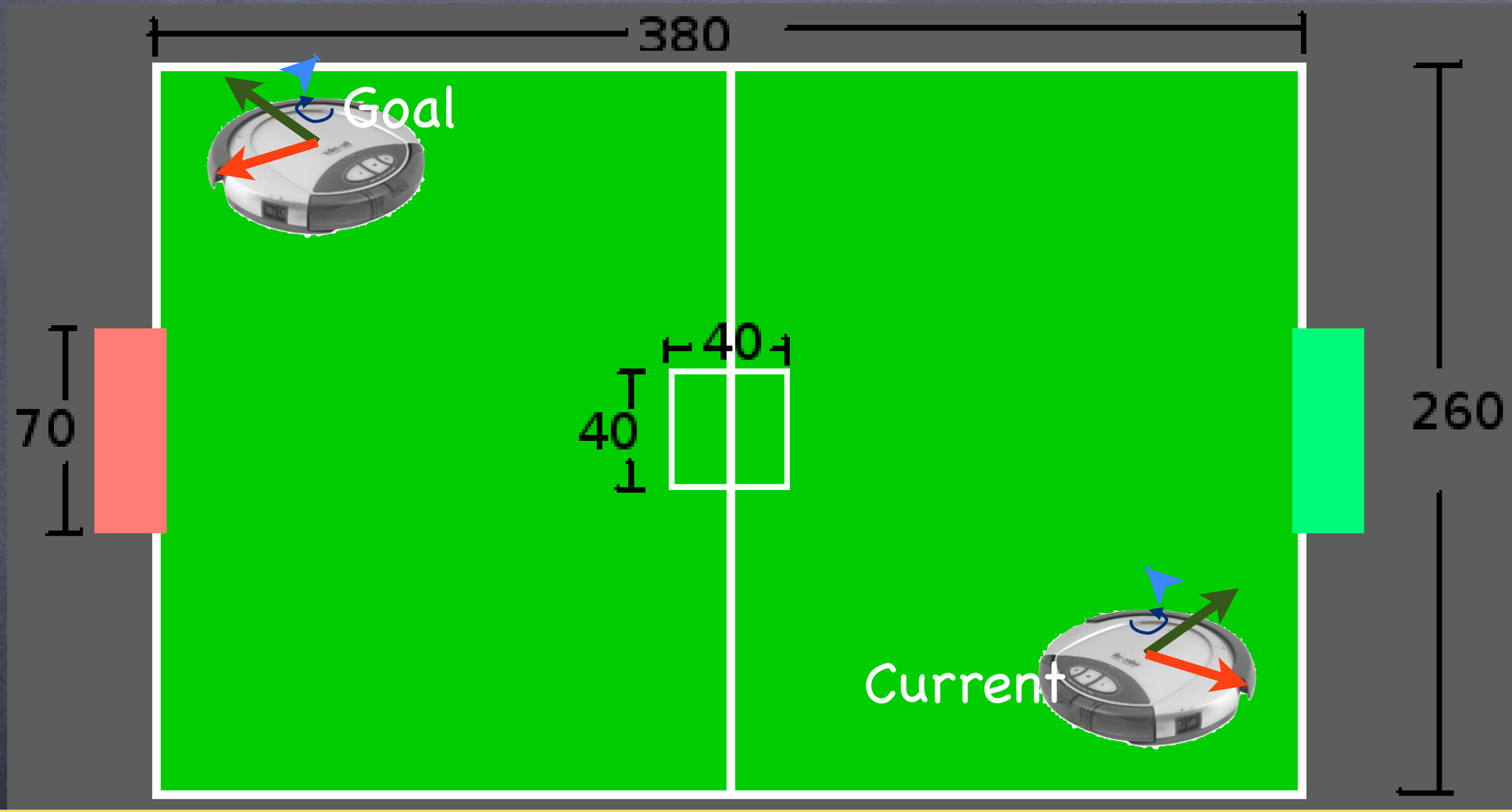


- Output of potential field is a vector
- Combine multiple potentials through vector summation

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

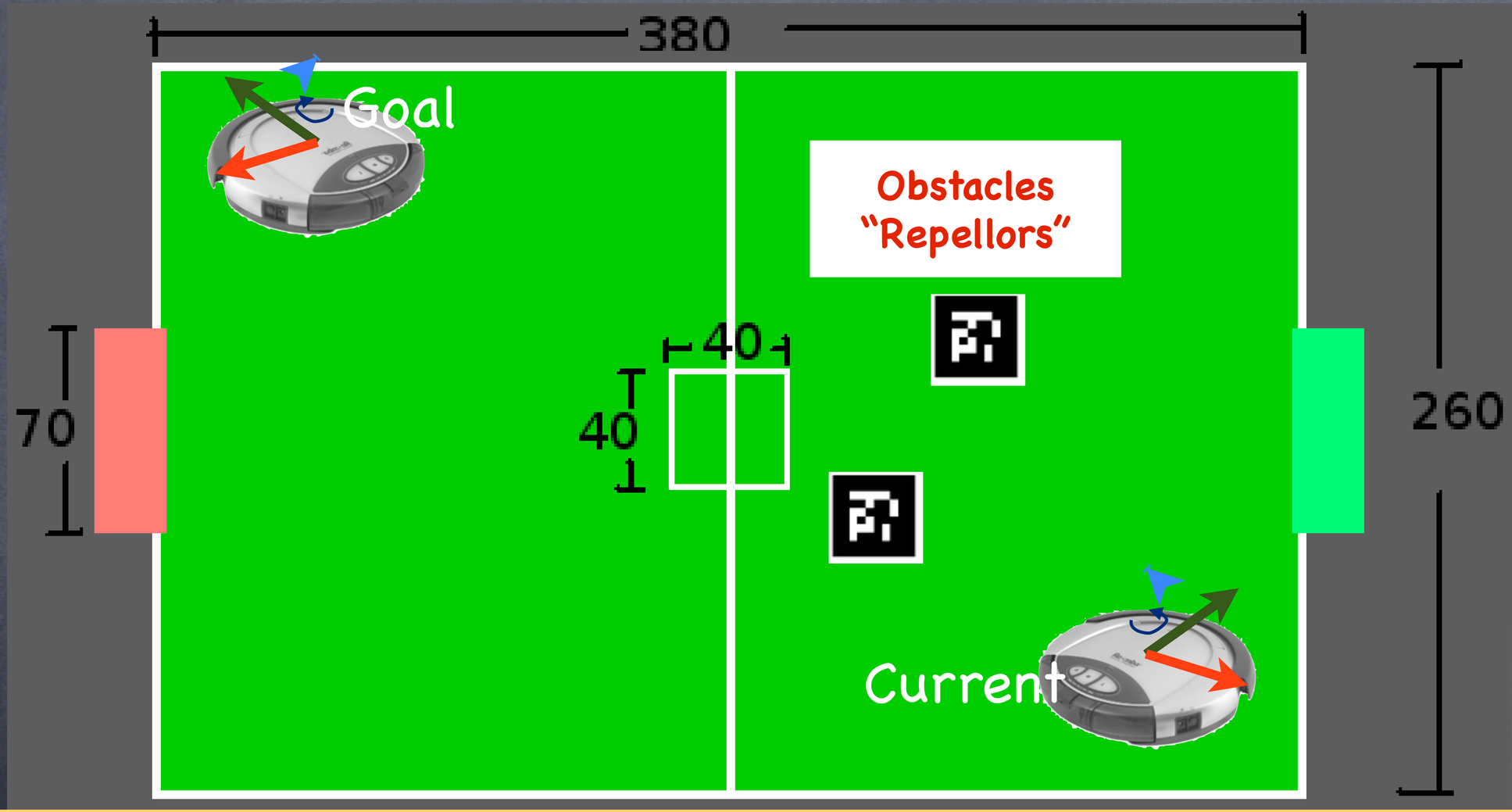
Slide borrowed from Michigan Robotics autorob.org

describe performance for this case



Slide borrowed from Michigan Robotics autorob.org

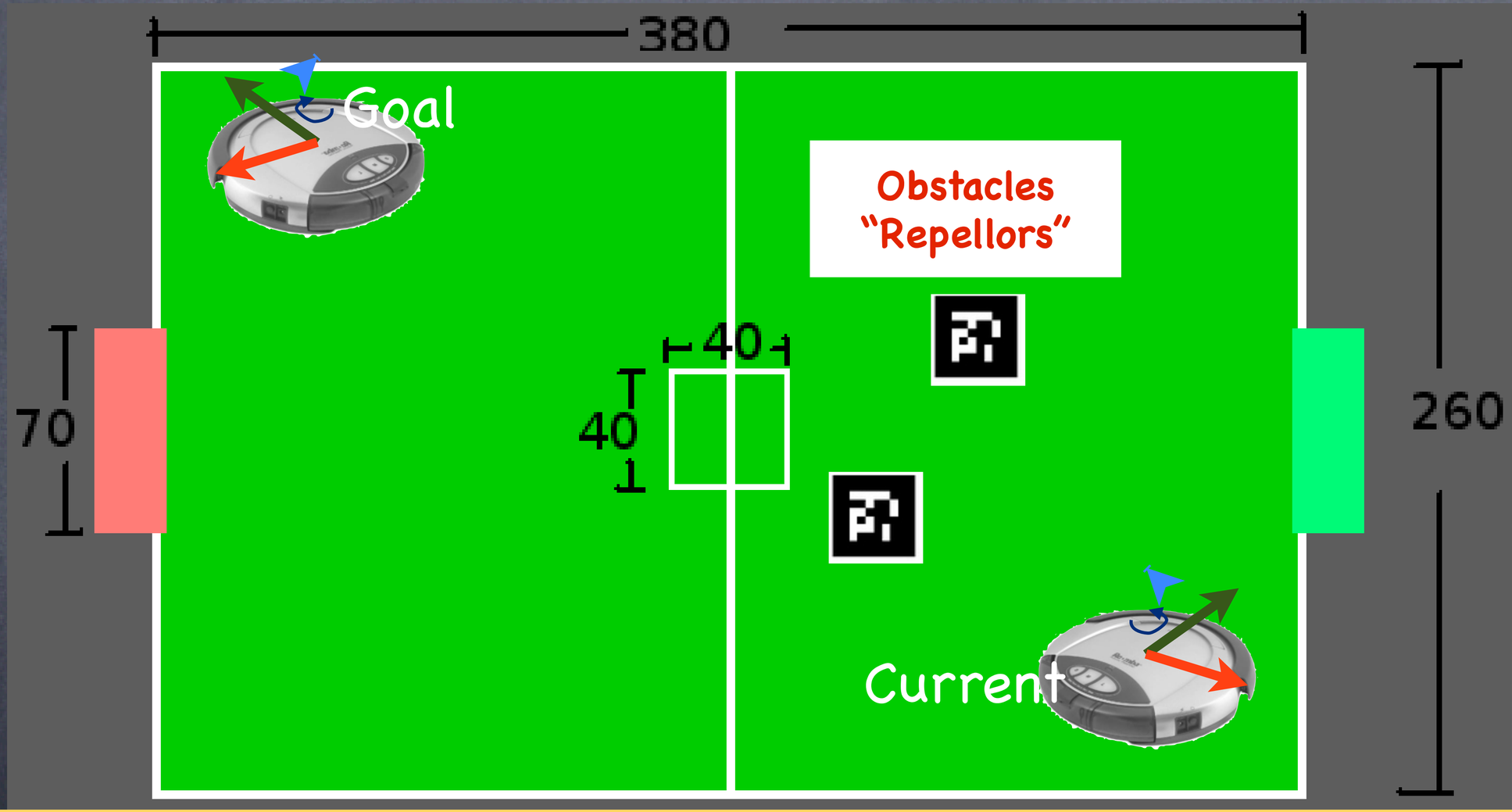
describe performance for this case



Slide borrowed from Michigan Robotics autorob.org

describe performance for this case

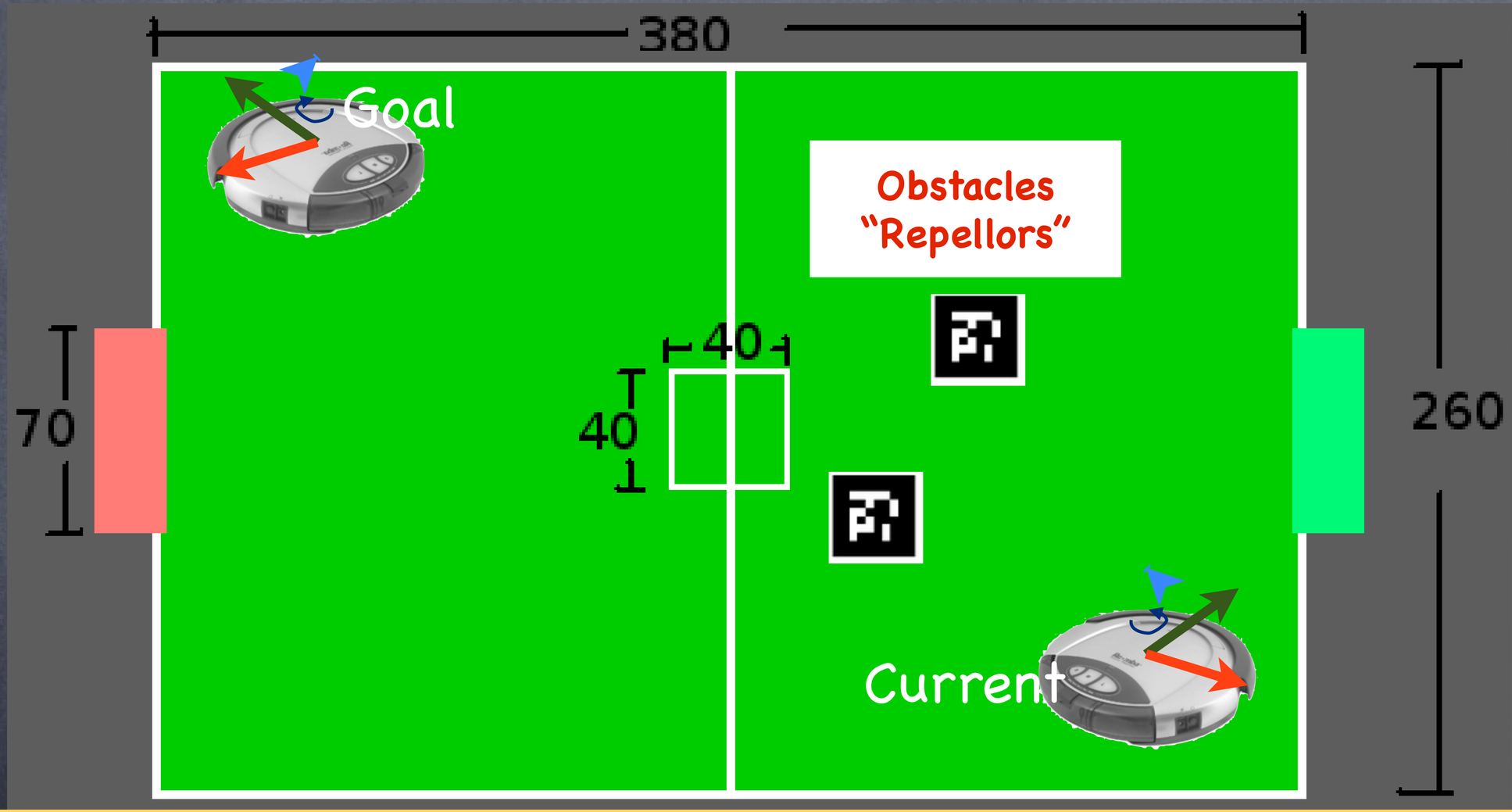
how do we deal with repellors?



Slide borrowed from Michigan Robotics autorob.org

add sum of repulsive potentials

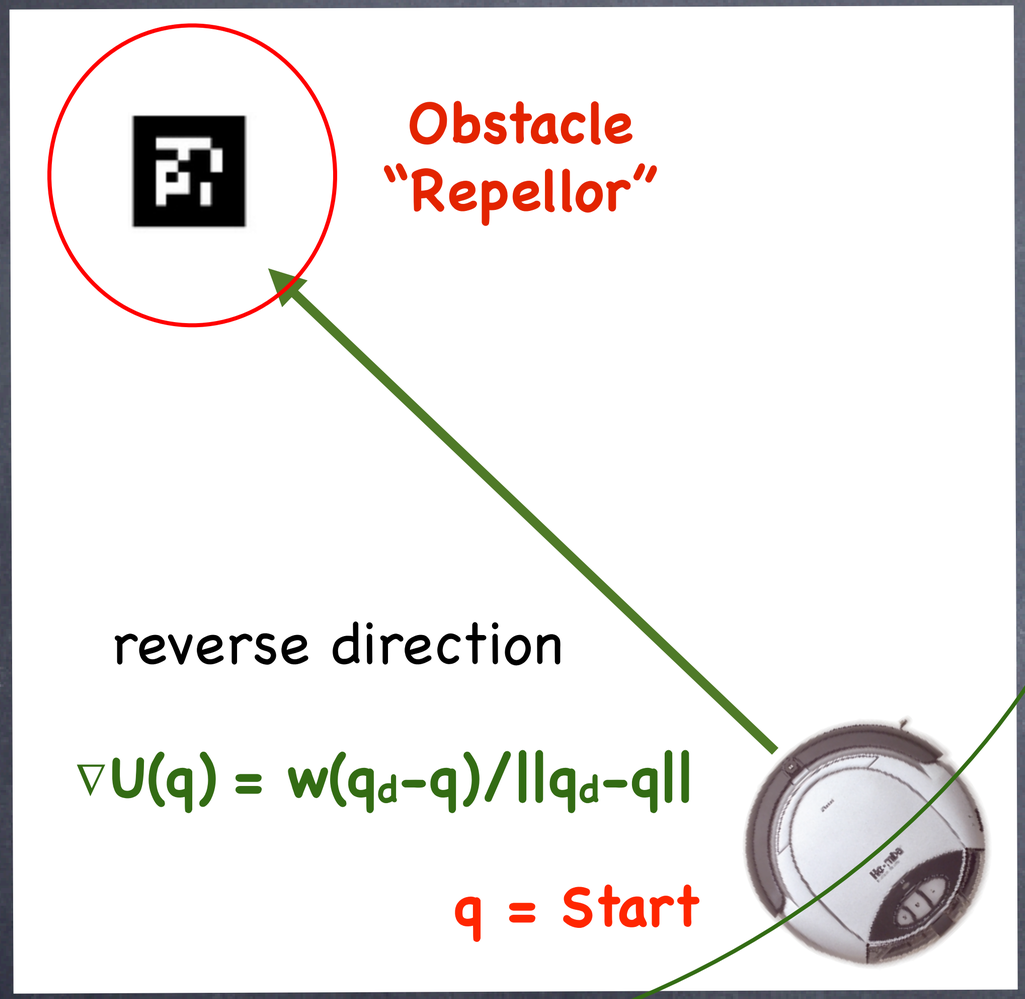
$$U(q) = U_{attracts}(q) + U_{repellors}(q)$$



Slide borrowed from Michigan Robotics autorob.org

"Cone" Repellor

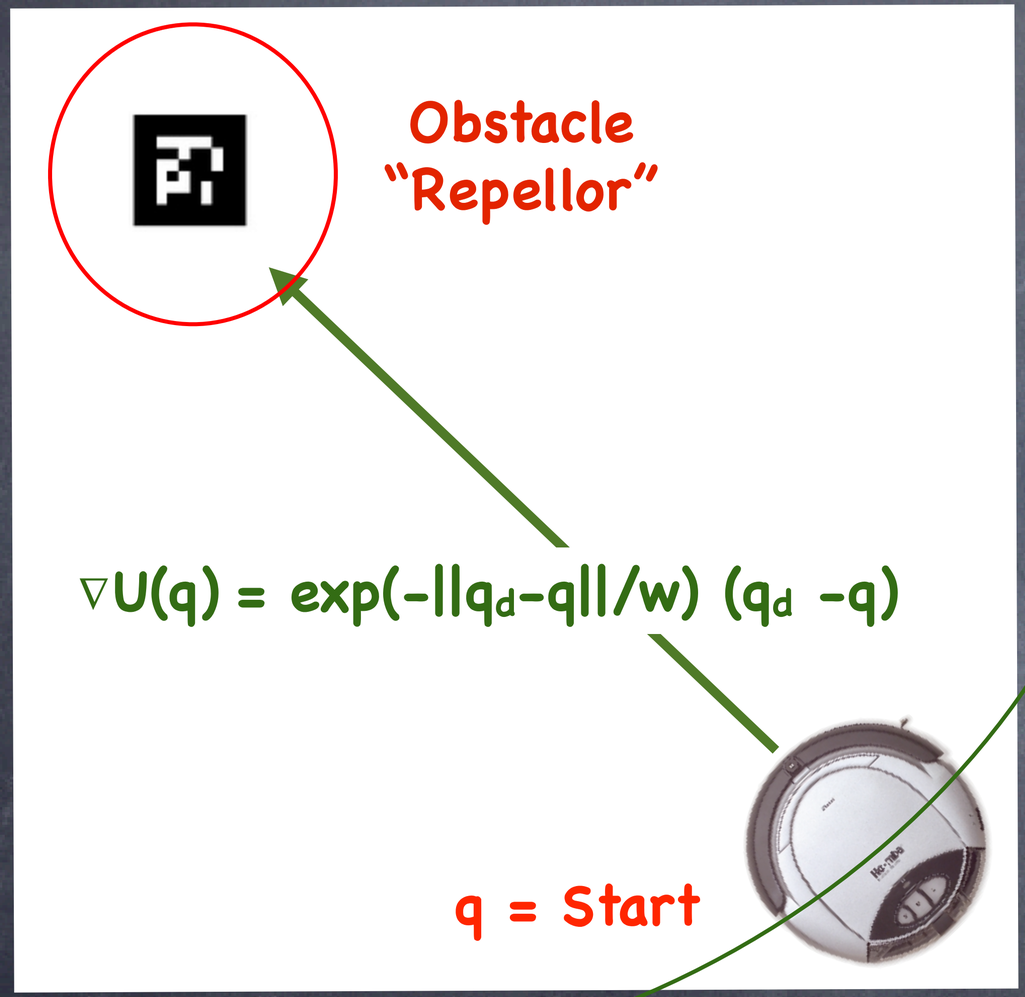
potential problems?



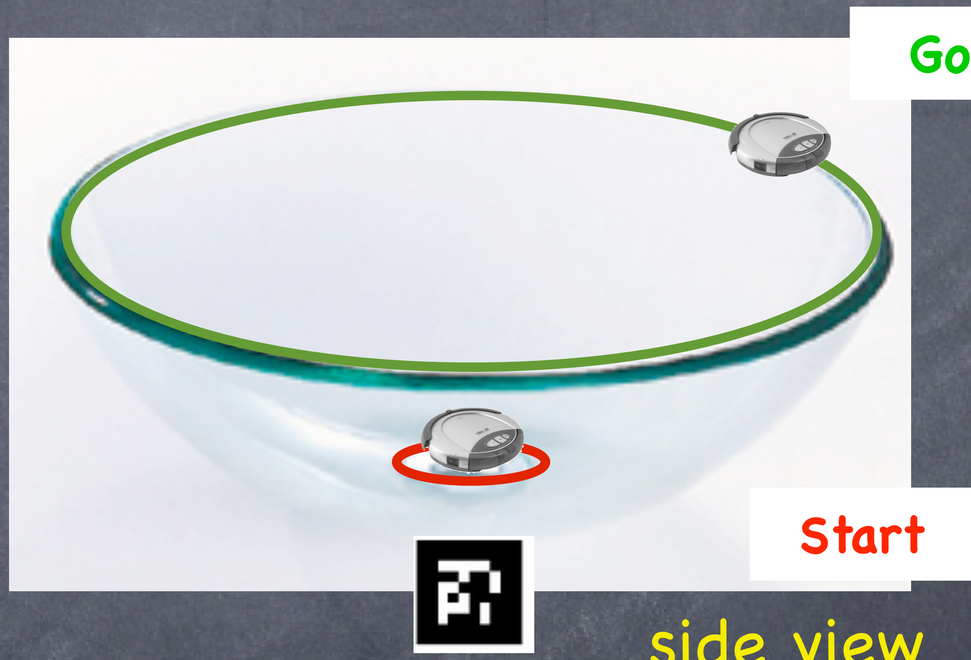
side view

Slide borrowed from Michigan Robotics autorob.org

"Bowl" Repellor



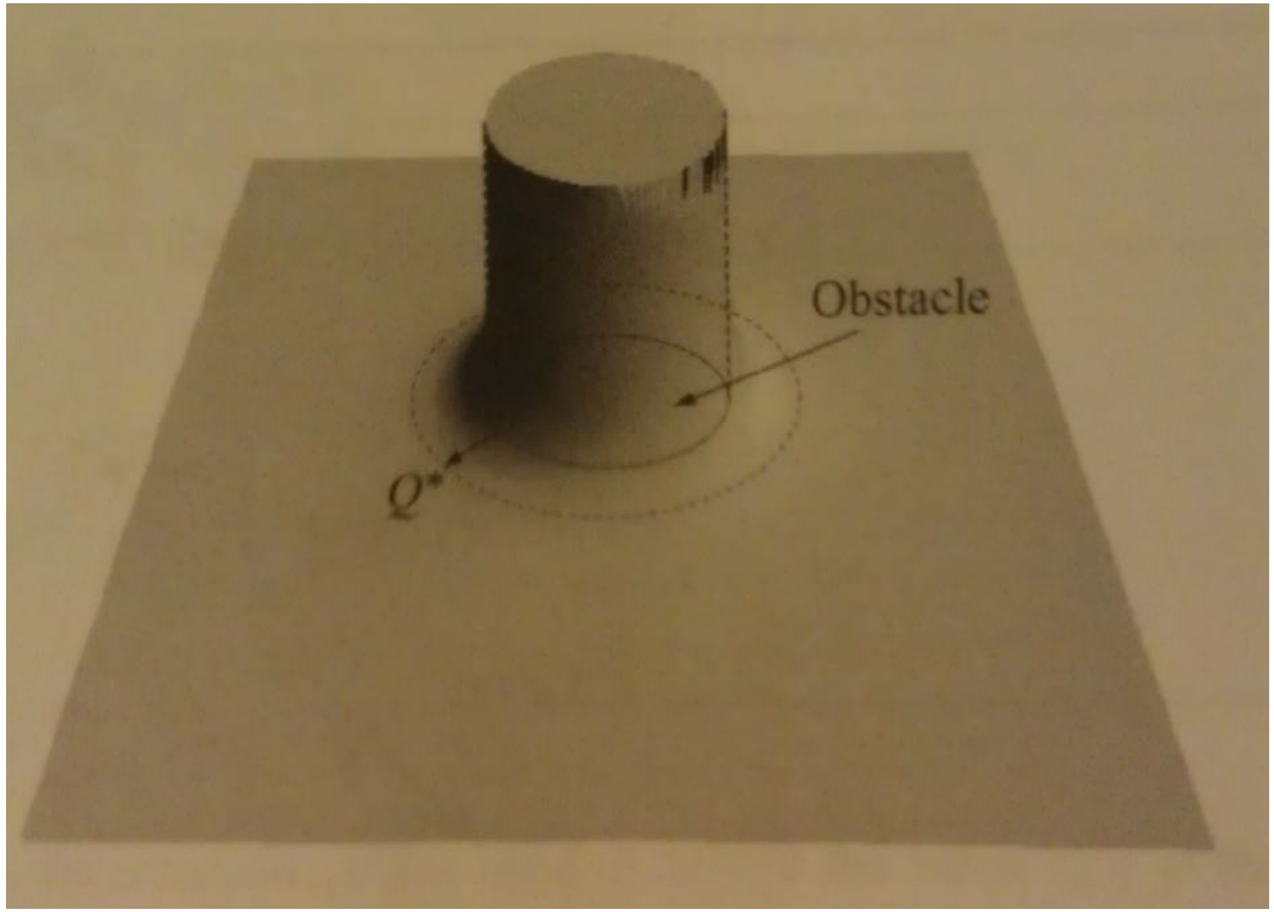
top view



side view

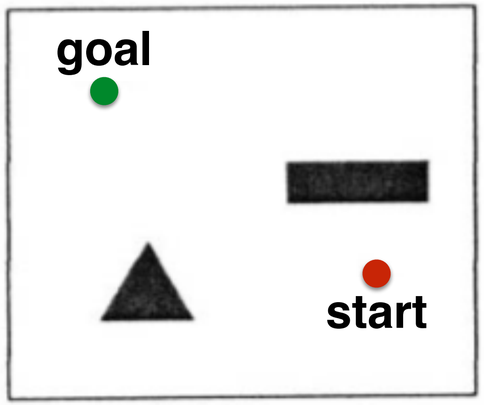
Slide borrowed from Michigan Robotics autorob.org

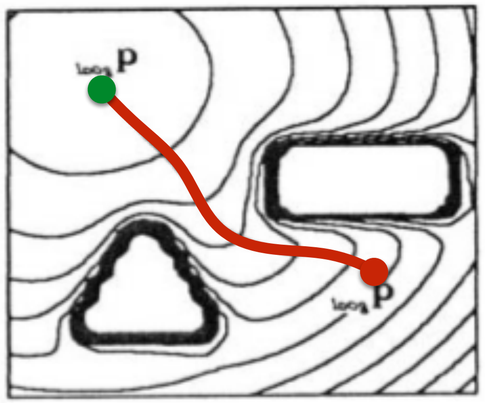
repellor should only have local influence,
repelling only around boundary improves path



2 Obstacle example

configuration space





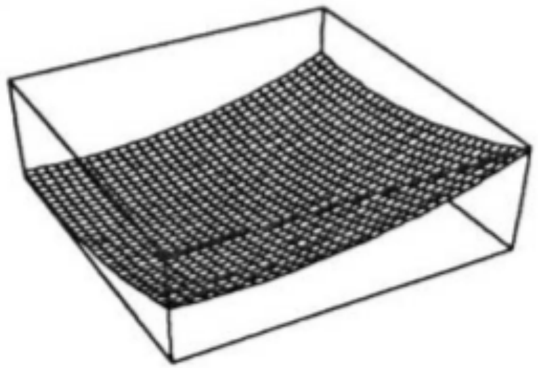
path from descent on gradient field

attractor field

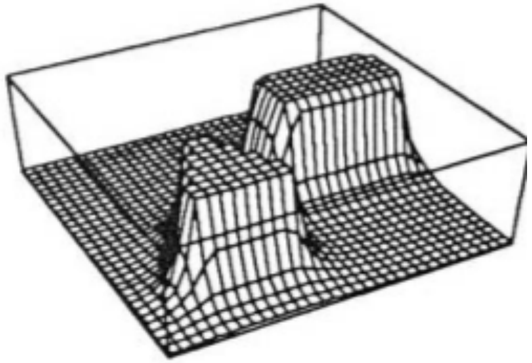
repellor fields

combined potential

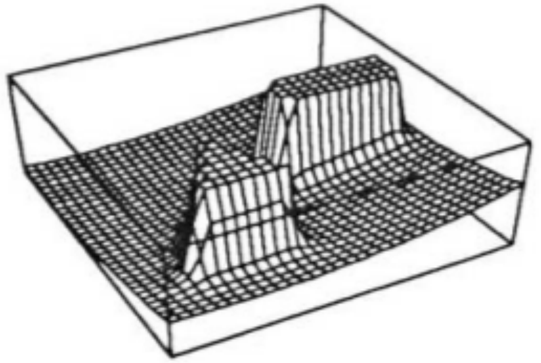
resulting gradient field

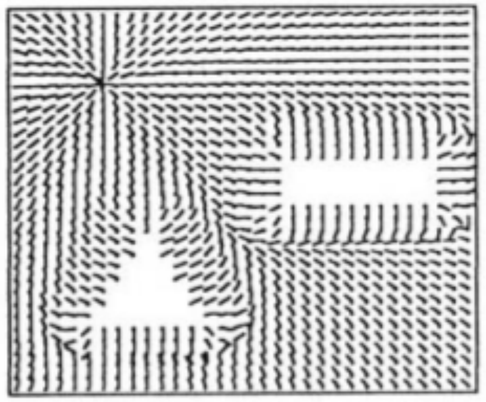


+

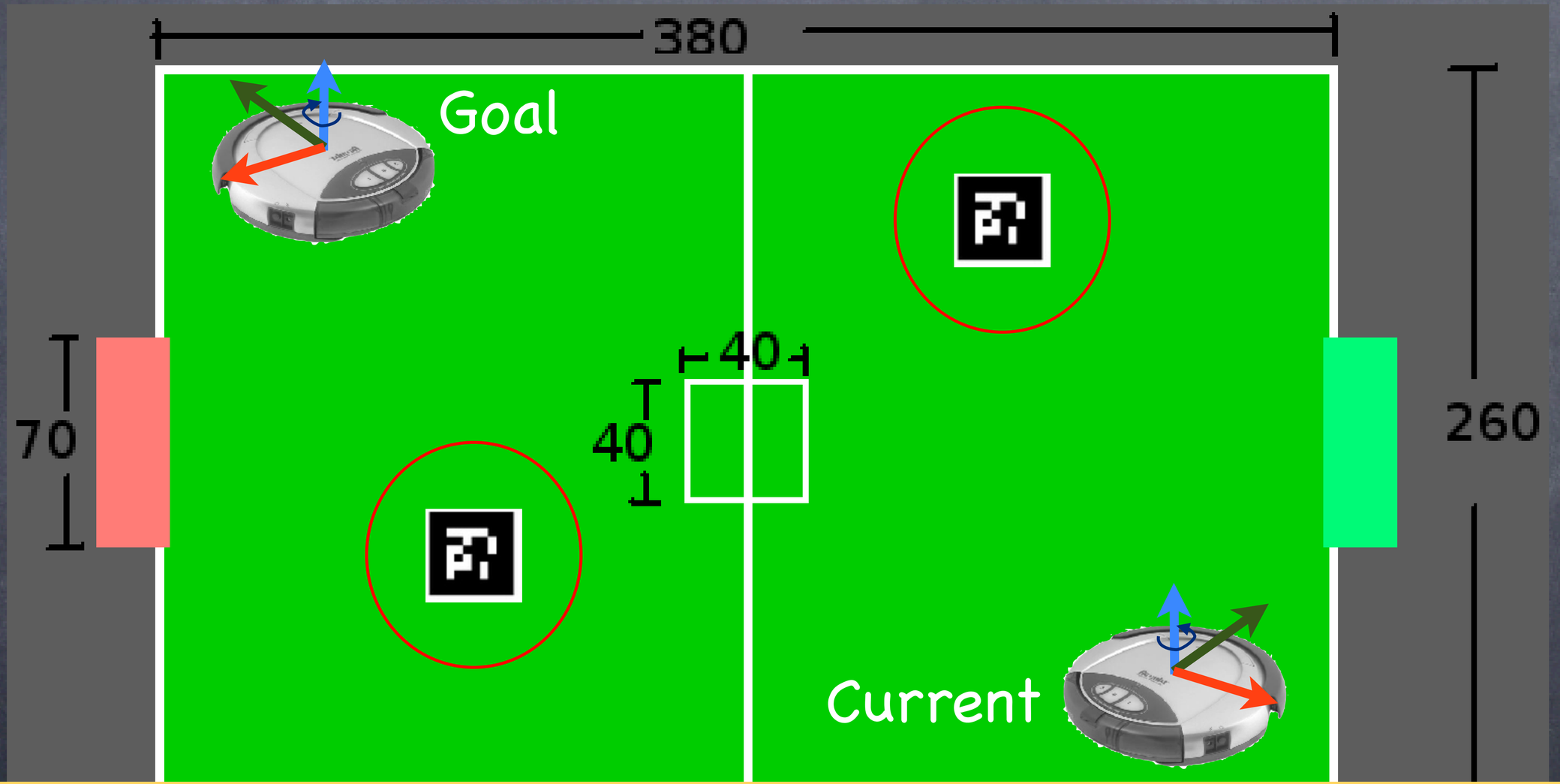


=



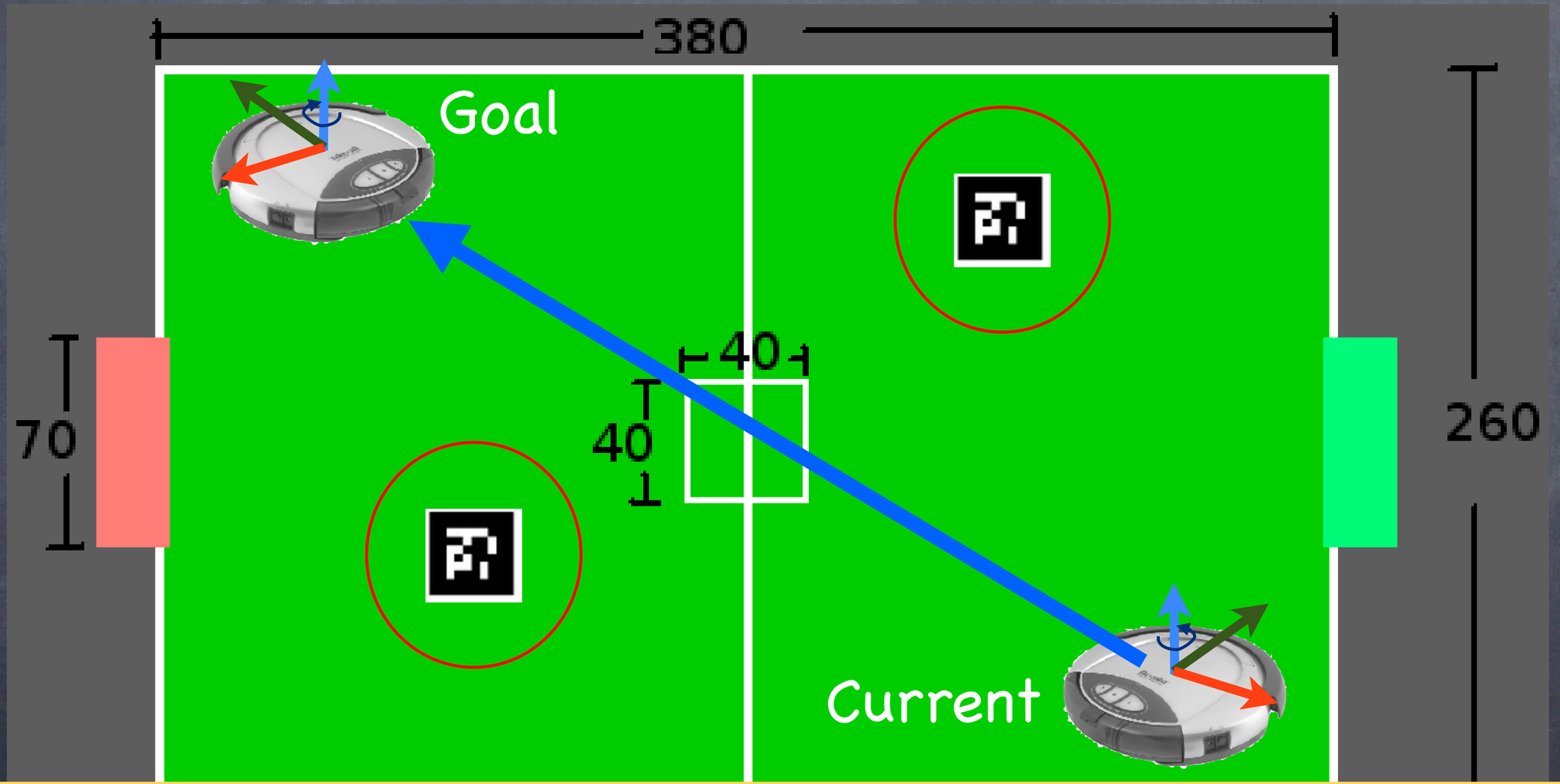


describe performance for this case
with cone attractor to goal and bowl repellors
with limited weight



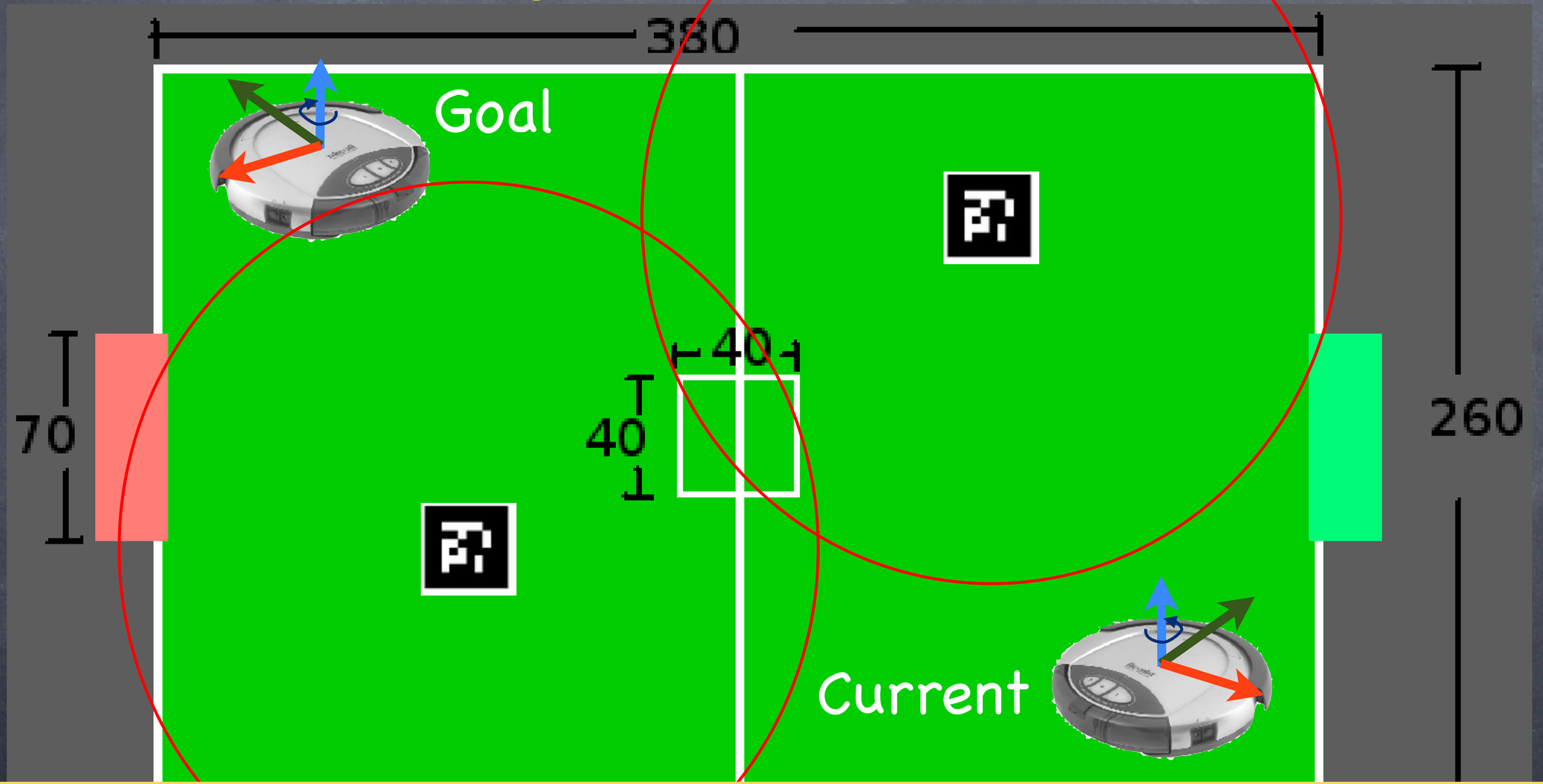
Slide borrowed from Michigan Robotics autorob.org

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



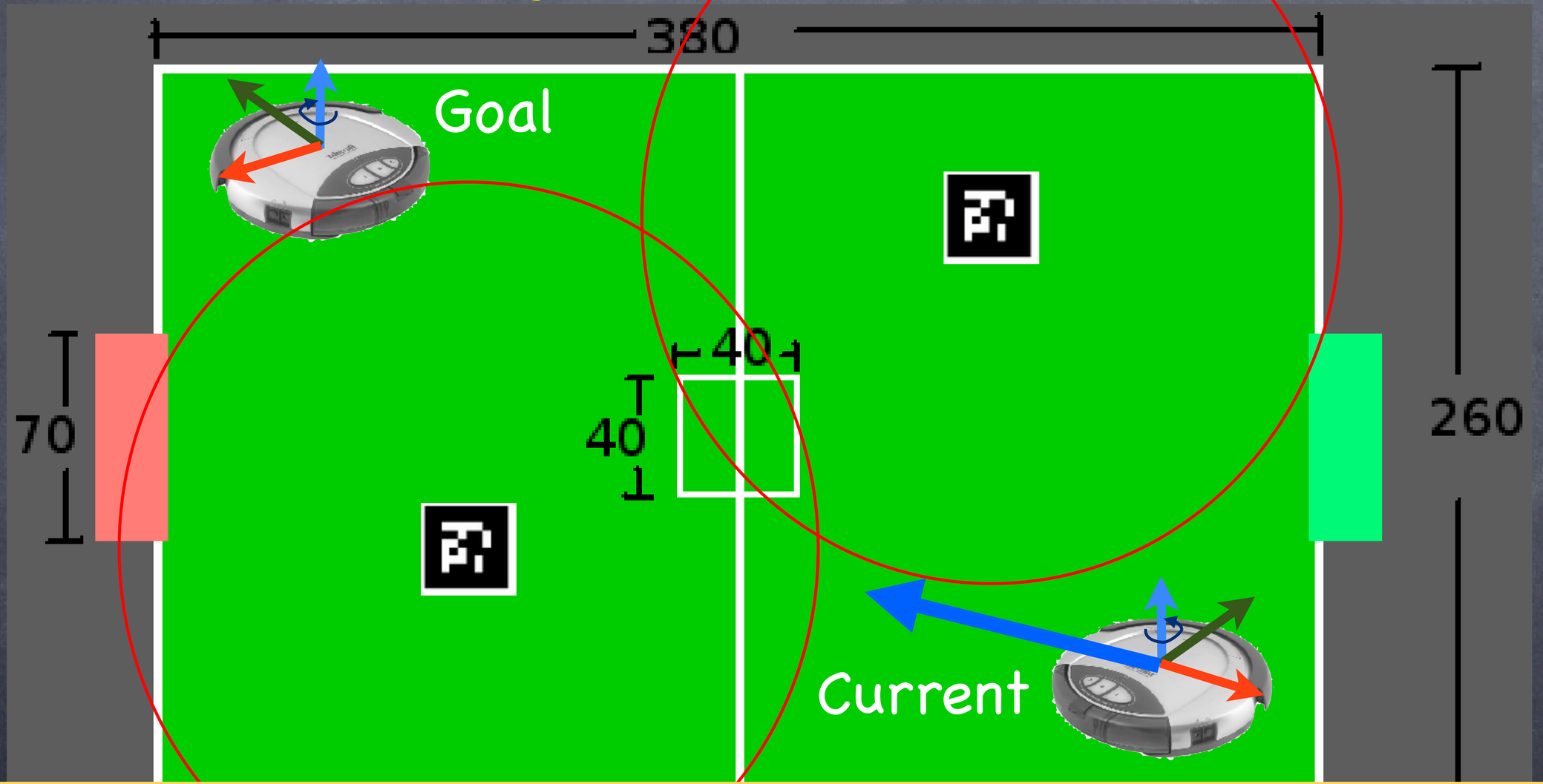
Slide borrowed from Michigan Robotics autorob.org

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



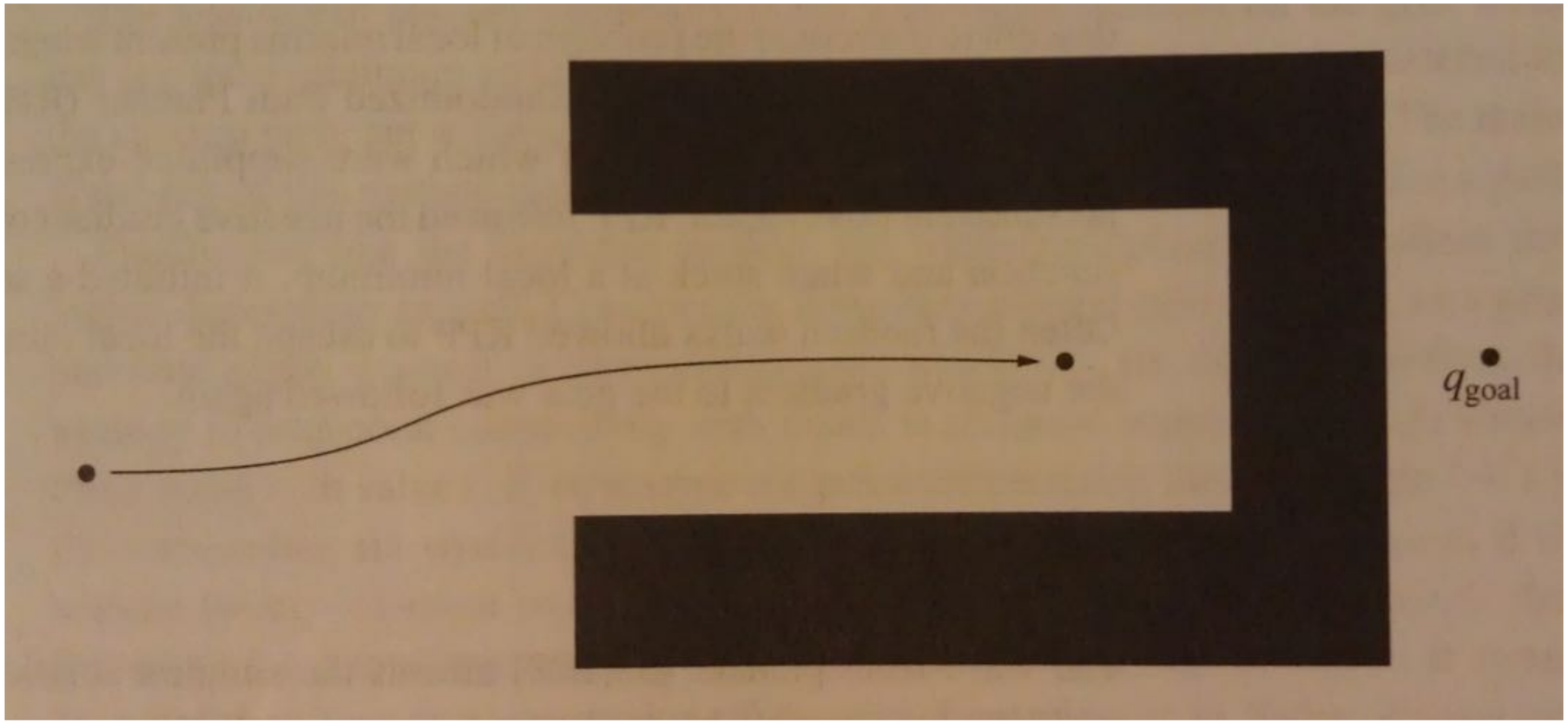
Slide borrowed from Michigan Robotics autorob.org

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



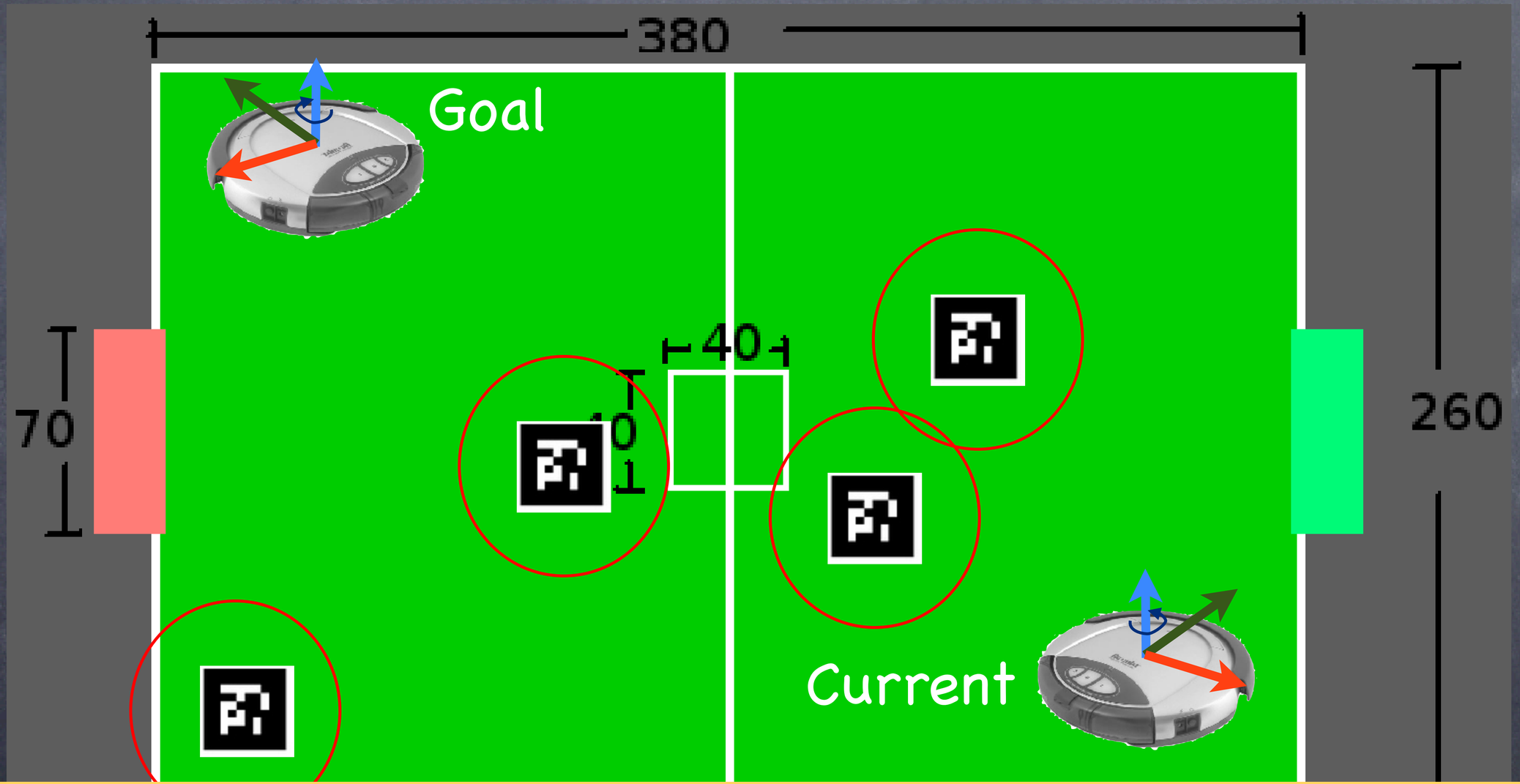
Slide borrowed from Michigan Robotics autorob.org

Local Minima



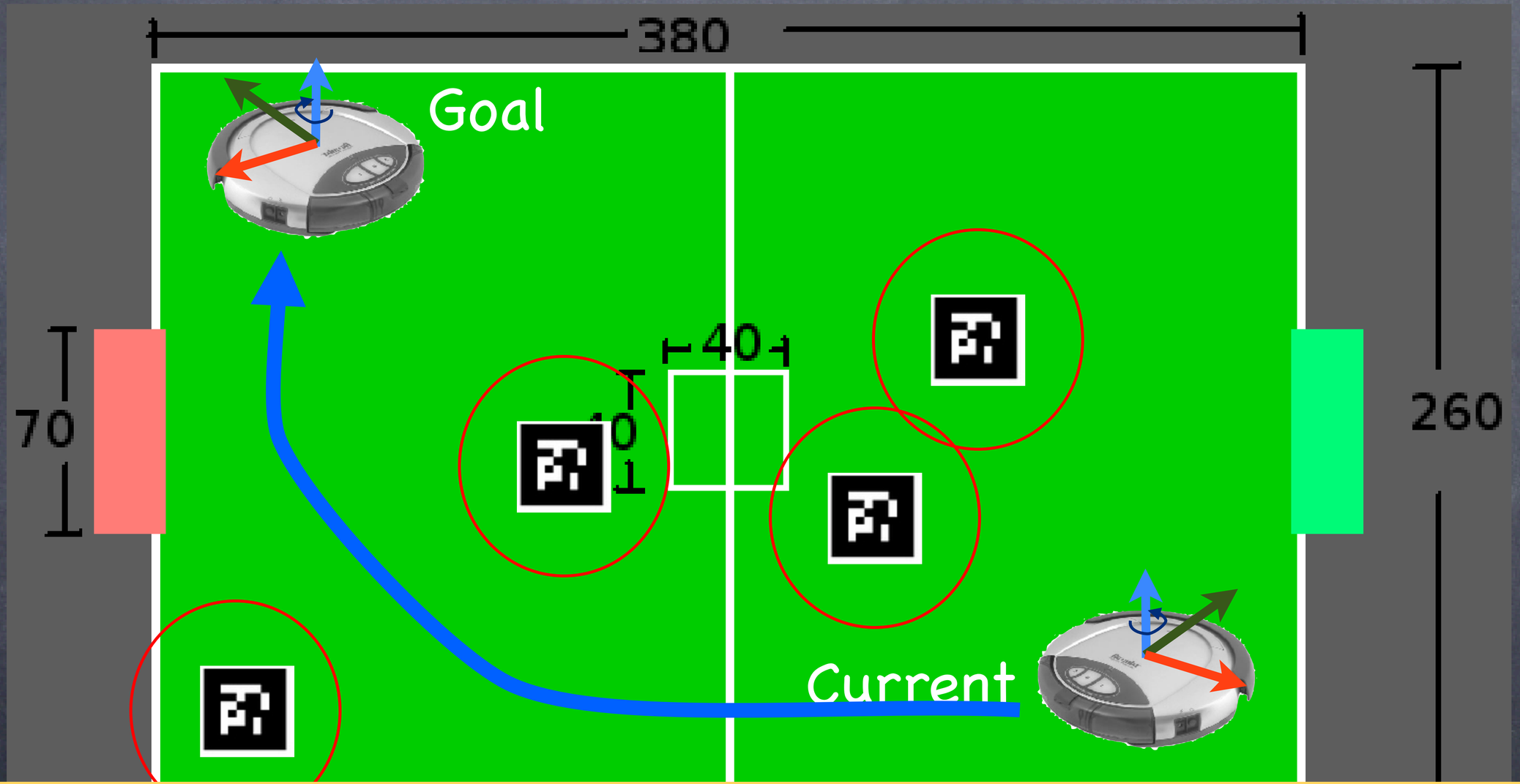
Slide borrowed from Michigan Robotics autorob.org

describe performance for this case



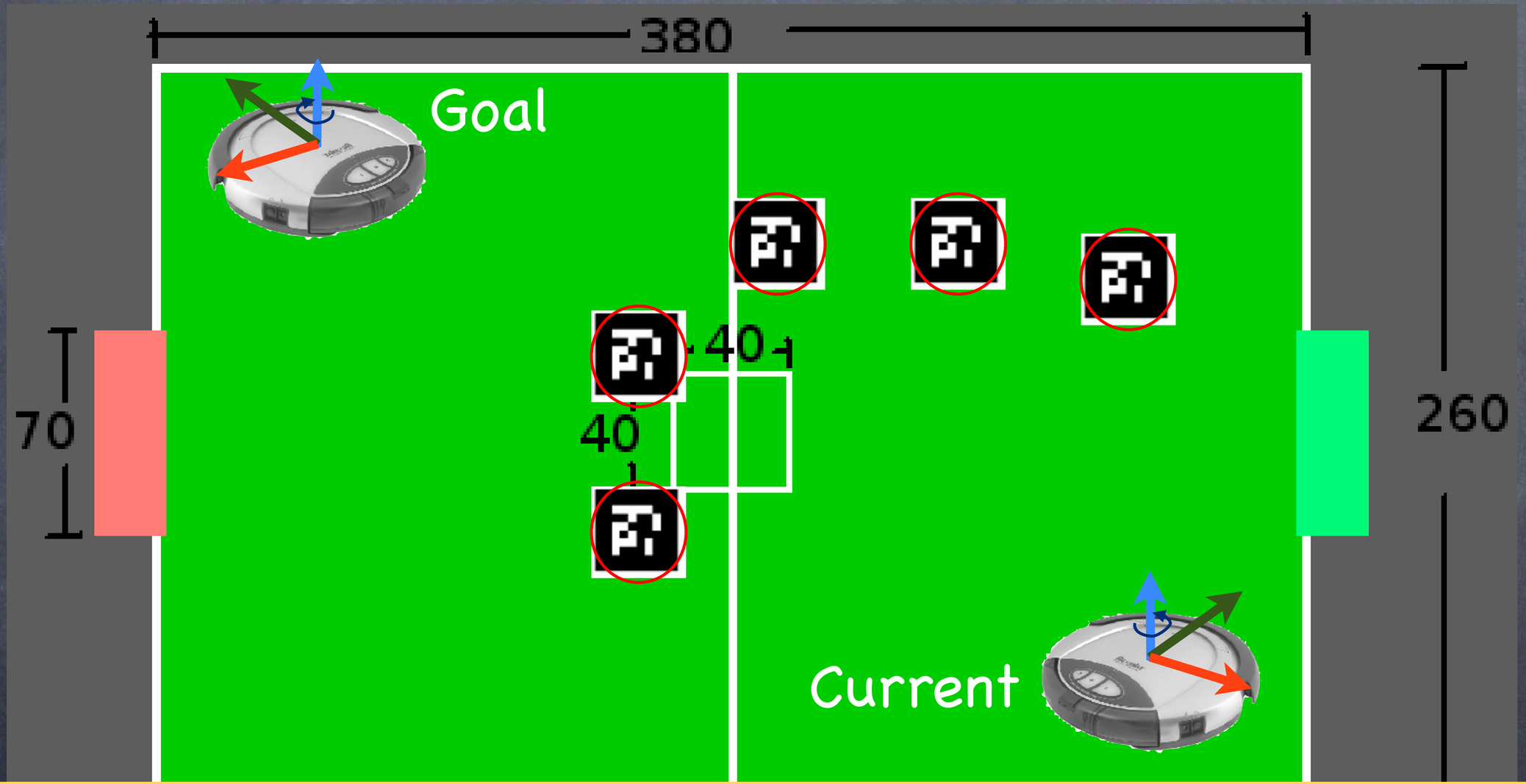
Slide borrowed from Michigan Robotics autorob.org

describe performance for this case



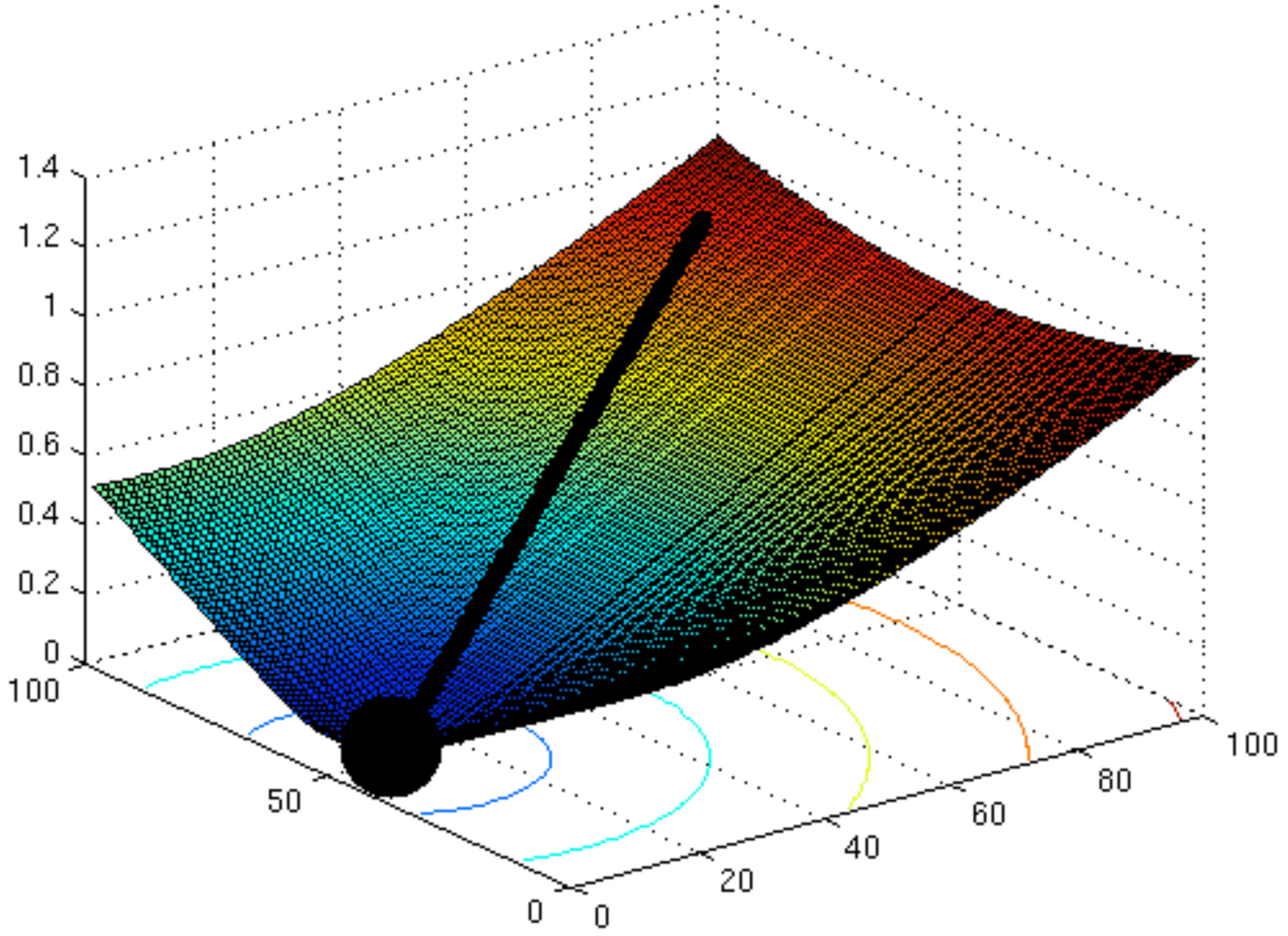
Slide borrowed from Michigan Robotics autorob.org

describe performance for this case



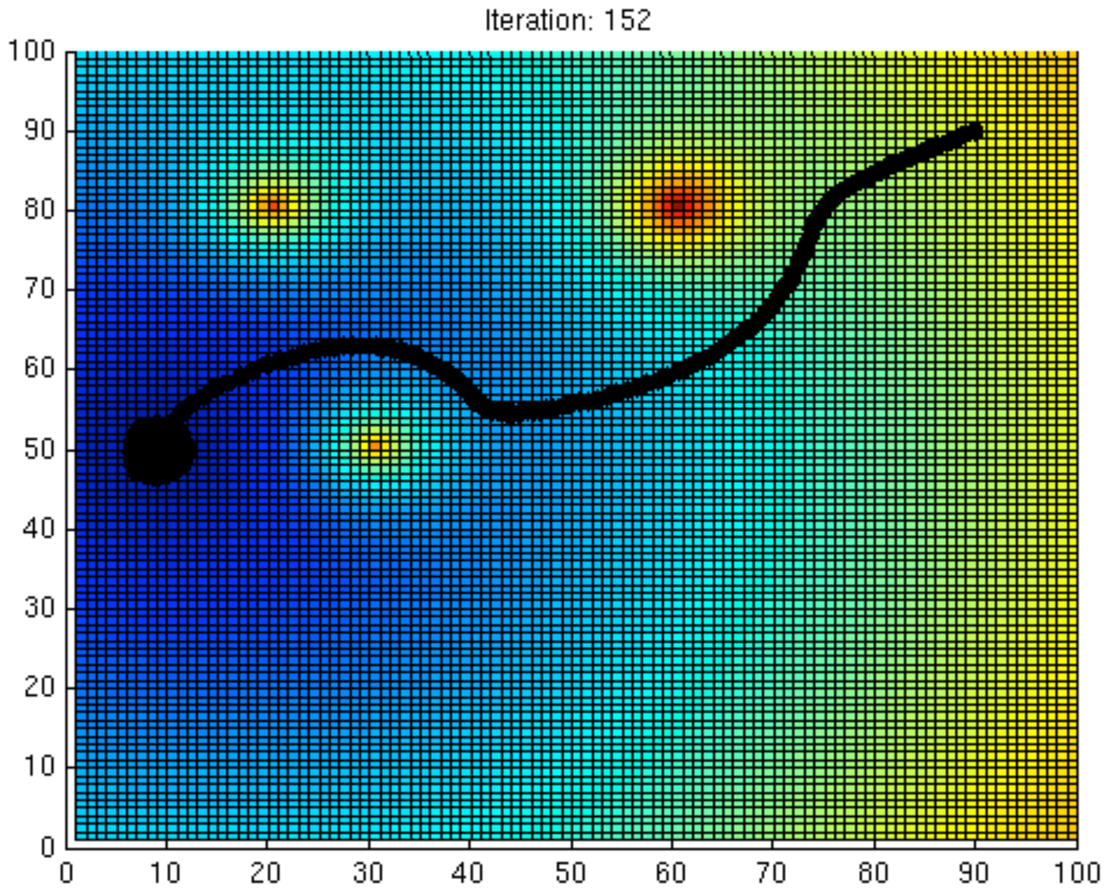
Slide borrowed from Michigan Robotics autorob.org

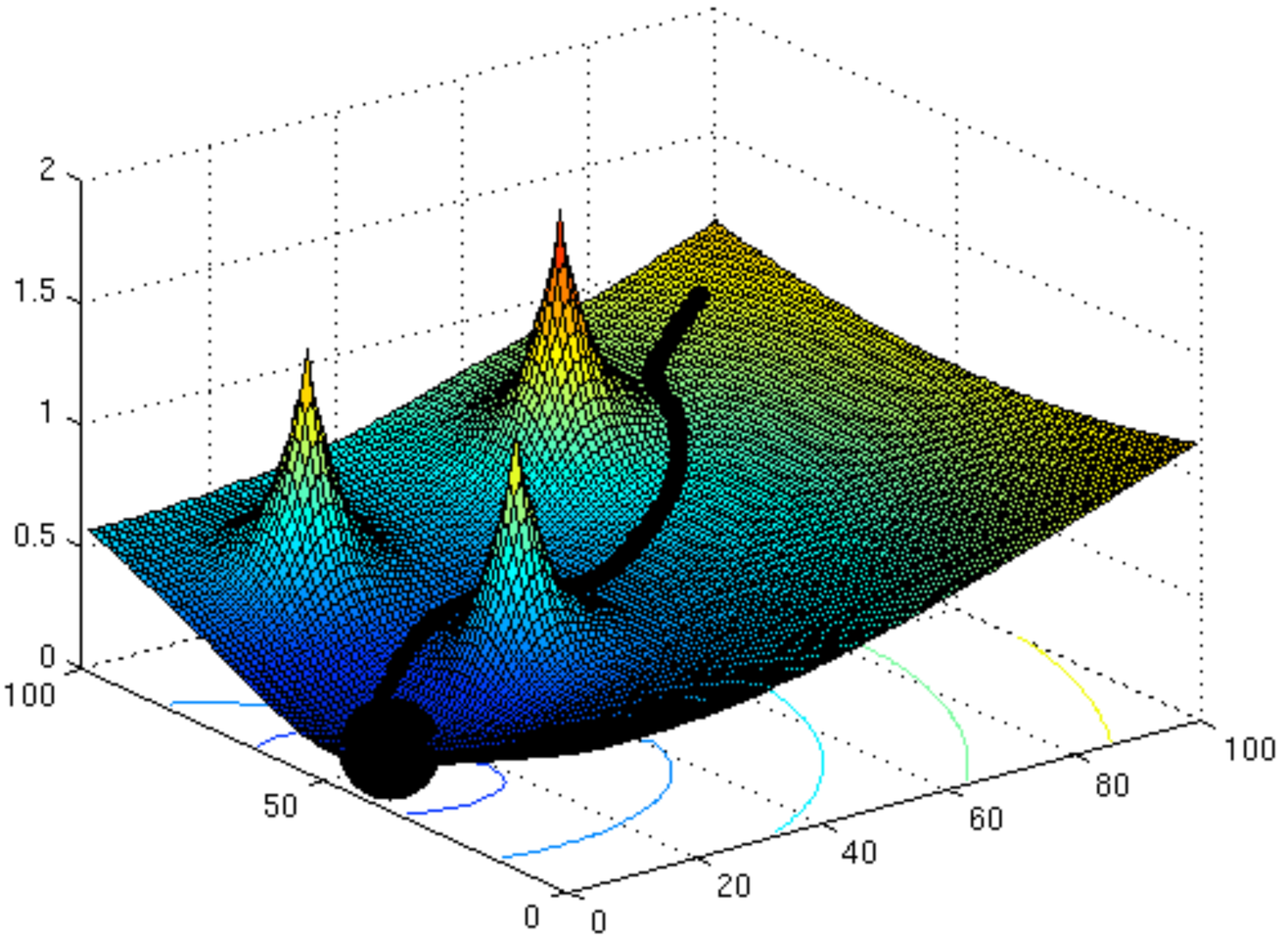
matlab example



pfield.m [1 5 8 12]

matlab example

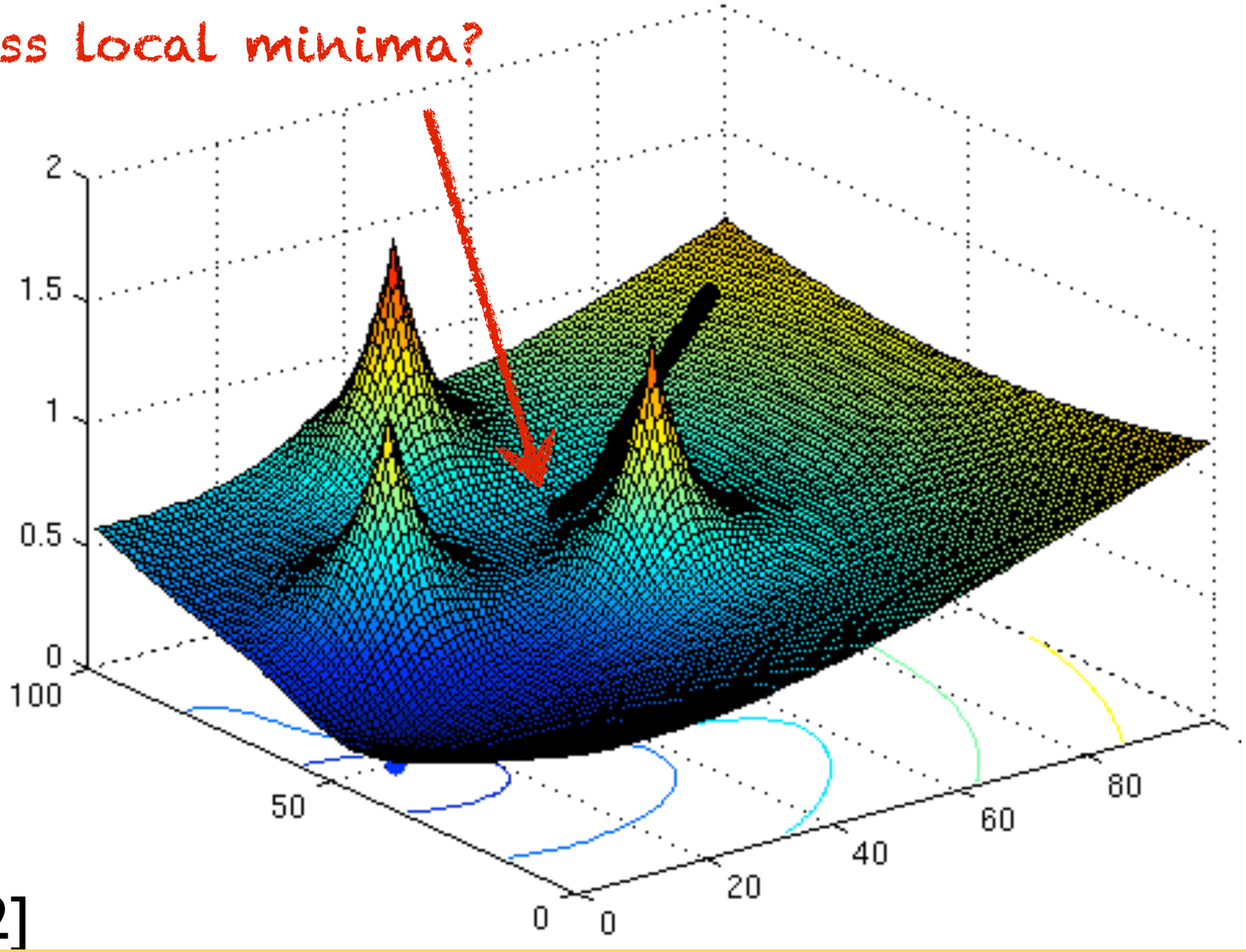




pfield.m [1 5 8 12]

matlab example

How to address local minima?



pfield.m [1 5 8 12]

How can we get out of local minima?

Slide borrowed from Michigan Robotics autorob.org

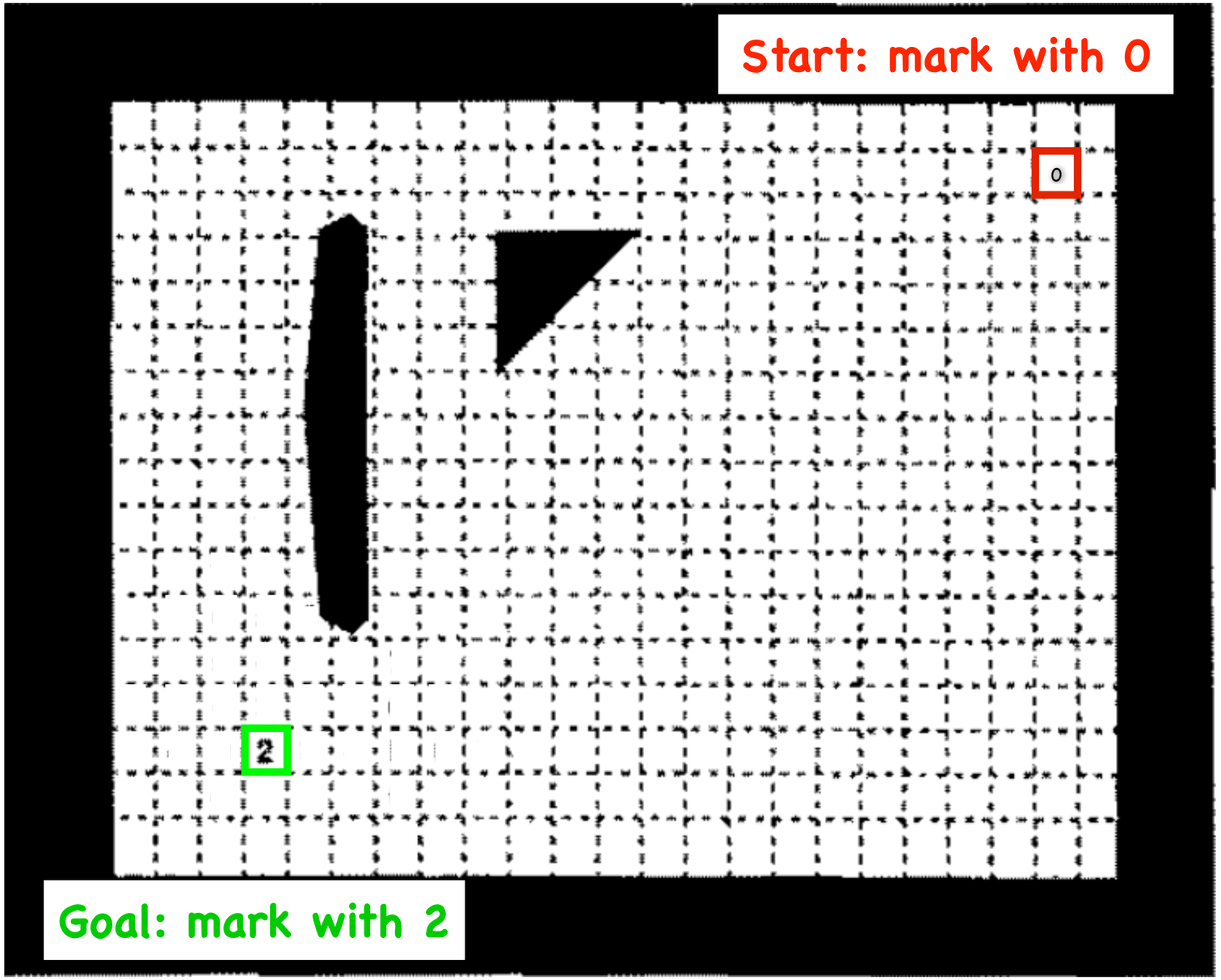
How can we get out of local minima?

Go back to planning.

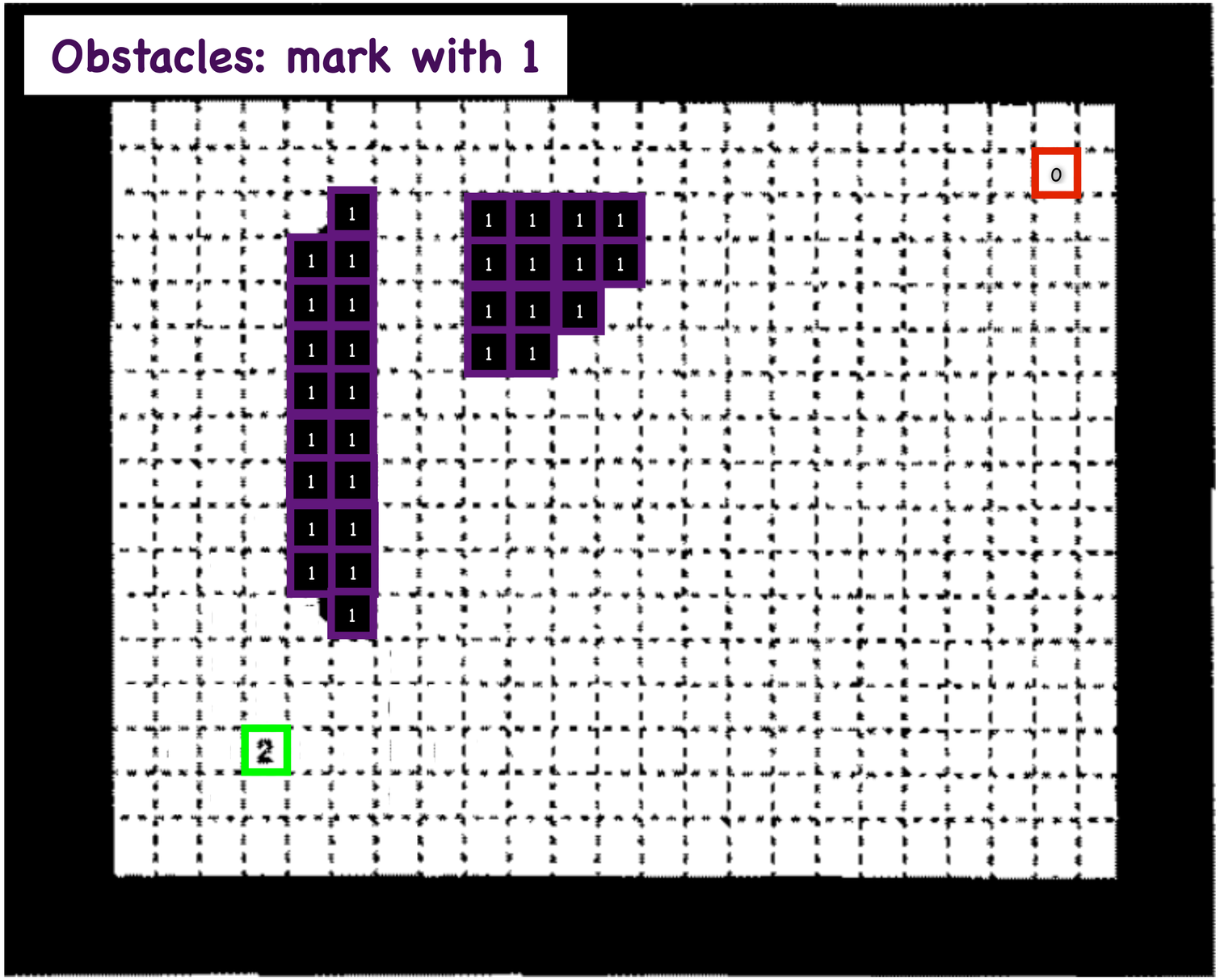
Slide borrowed from Michigan Robotics autorob.org

Wavefront Planning

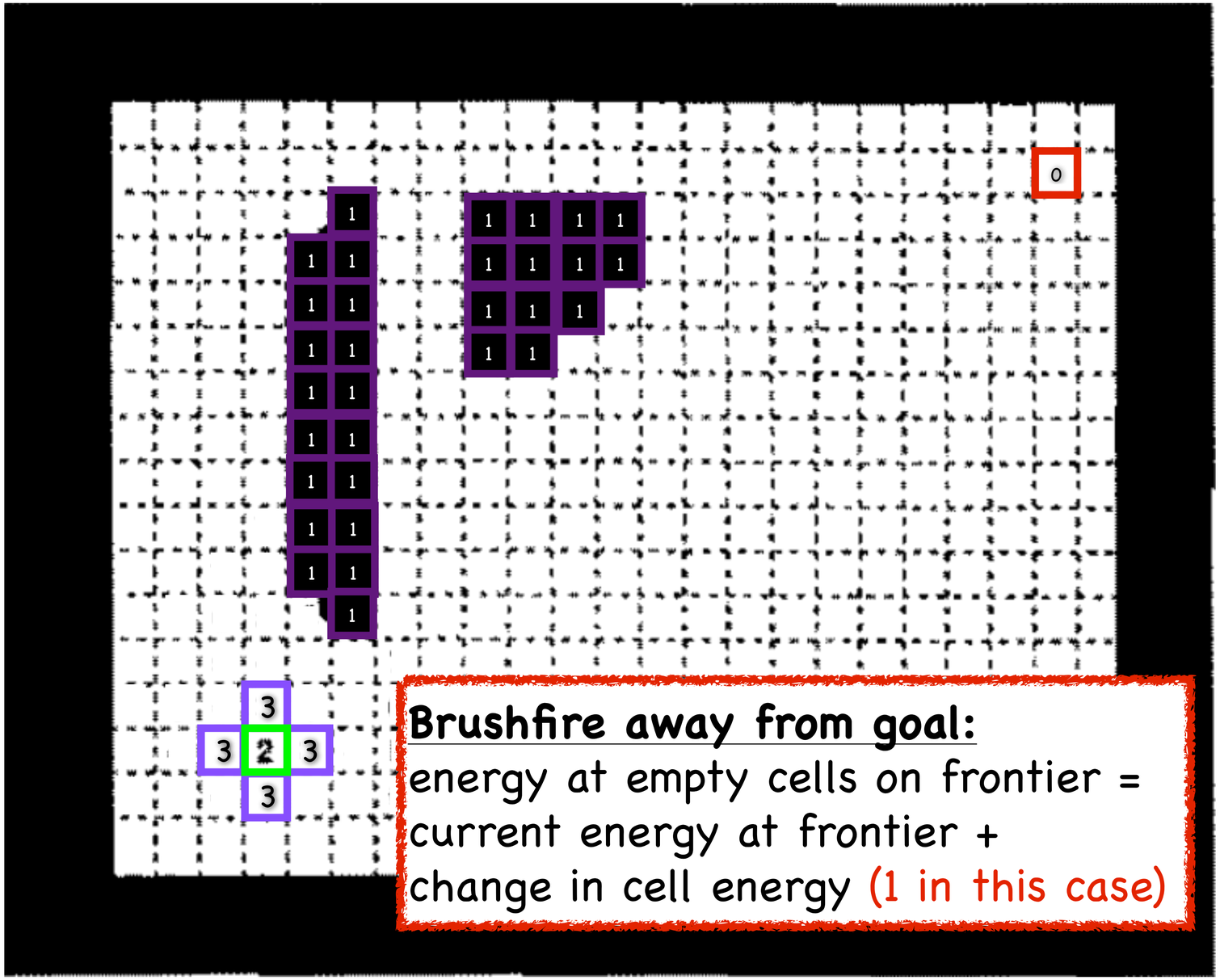
- Discretize potential field into grid
  - Cells store cost to goal with respect to potential field
  - Computed by Brushfire algorithm (essentially BFS)
- Grid search to find navigation path to goal


Start: mark with 0
0
2
Goal: mark with 2

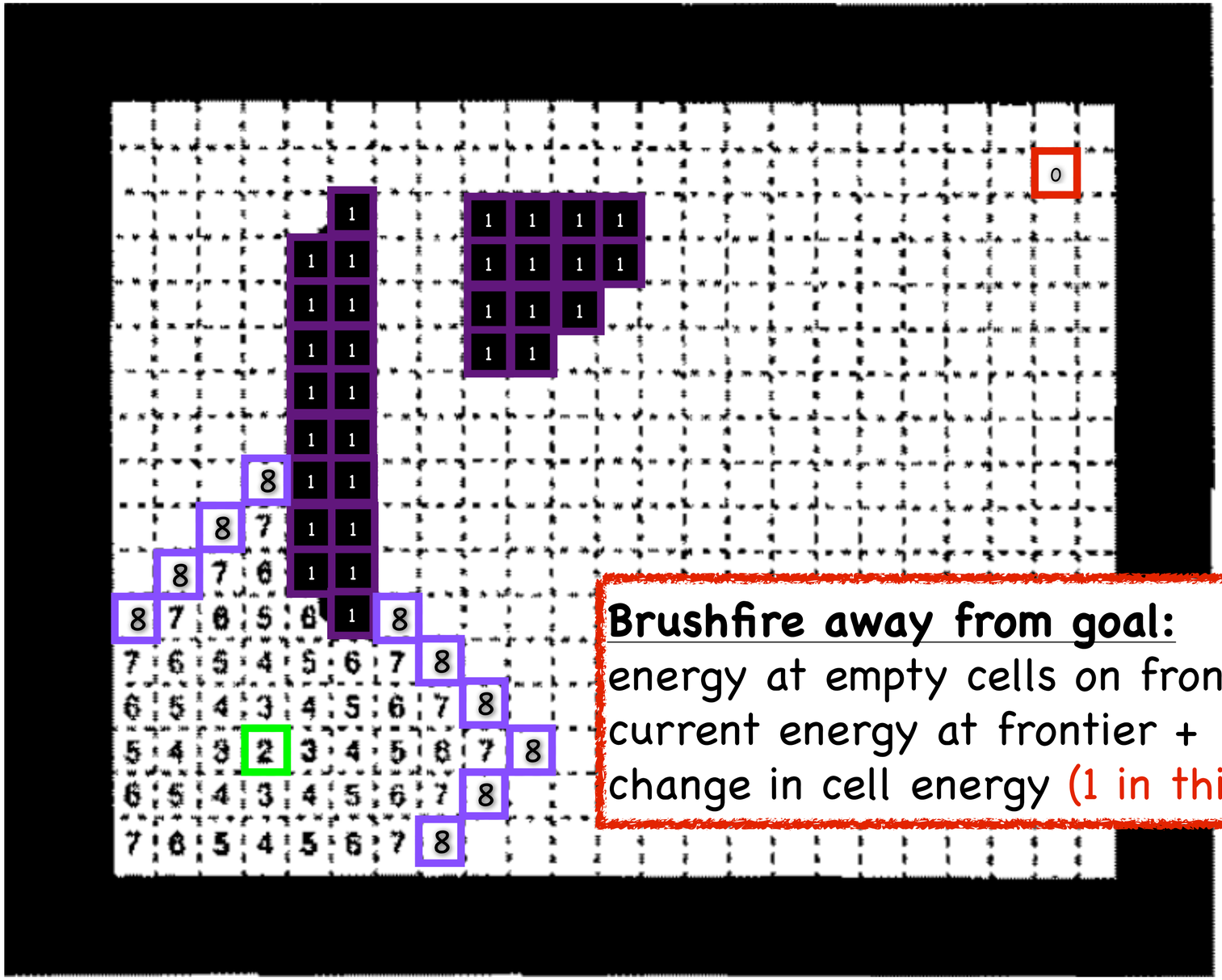
Obstacles: mark with 1

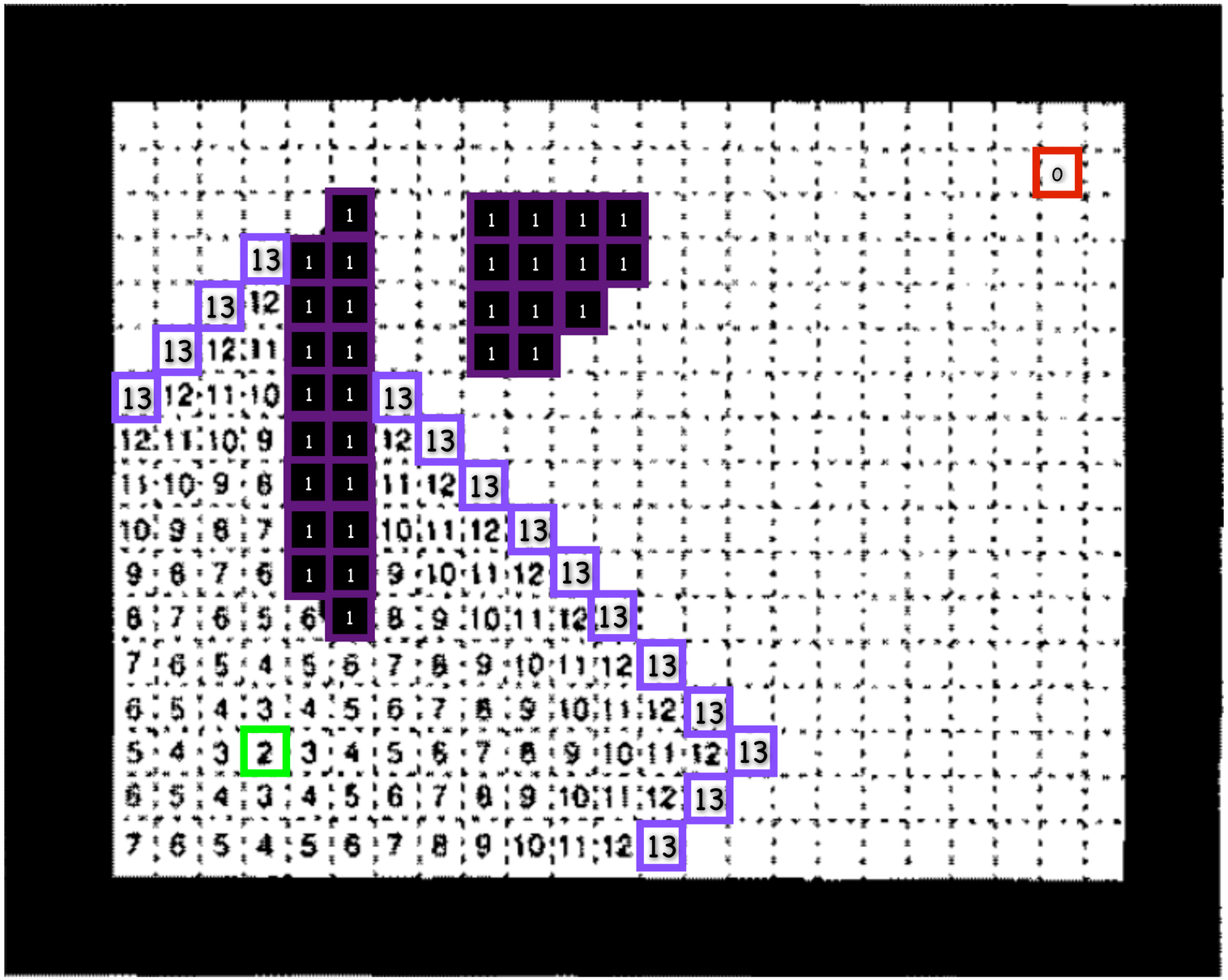


Slide borrowed from Michigan Robotics autorob.org

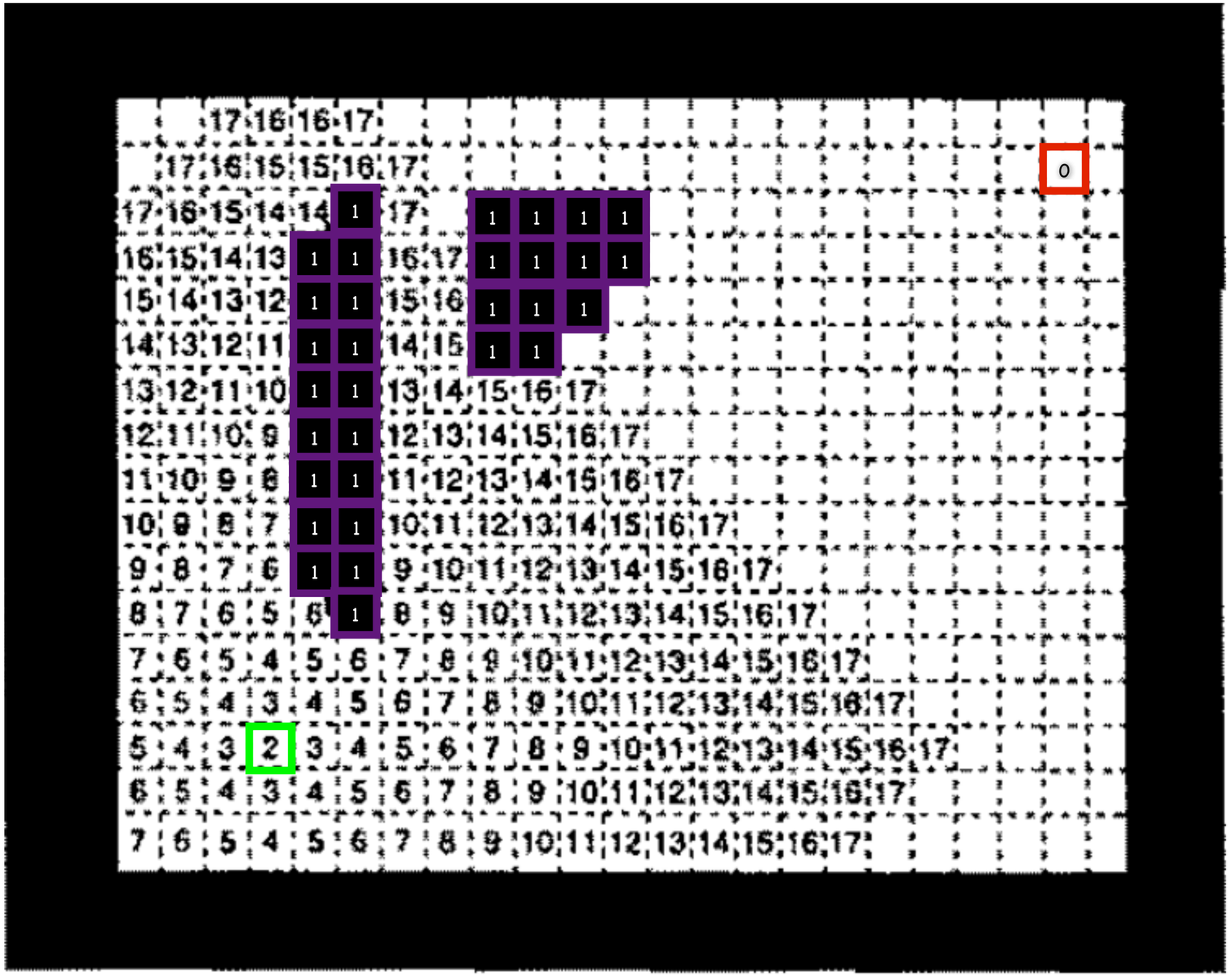


Brushfire away from goal:
energy at empty cells on frontier =
current energy at frontier +
change in cell energy (1 in this case)

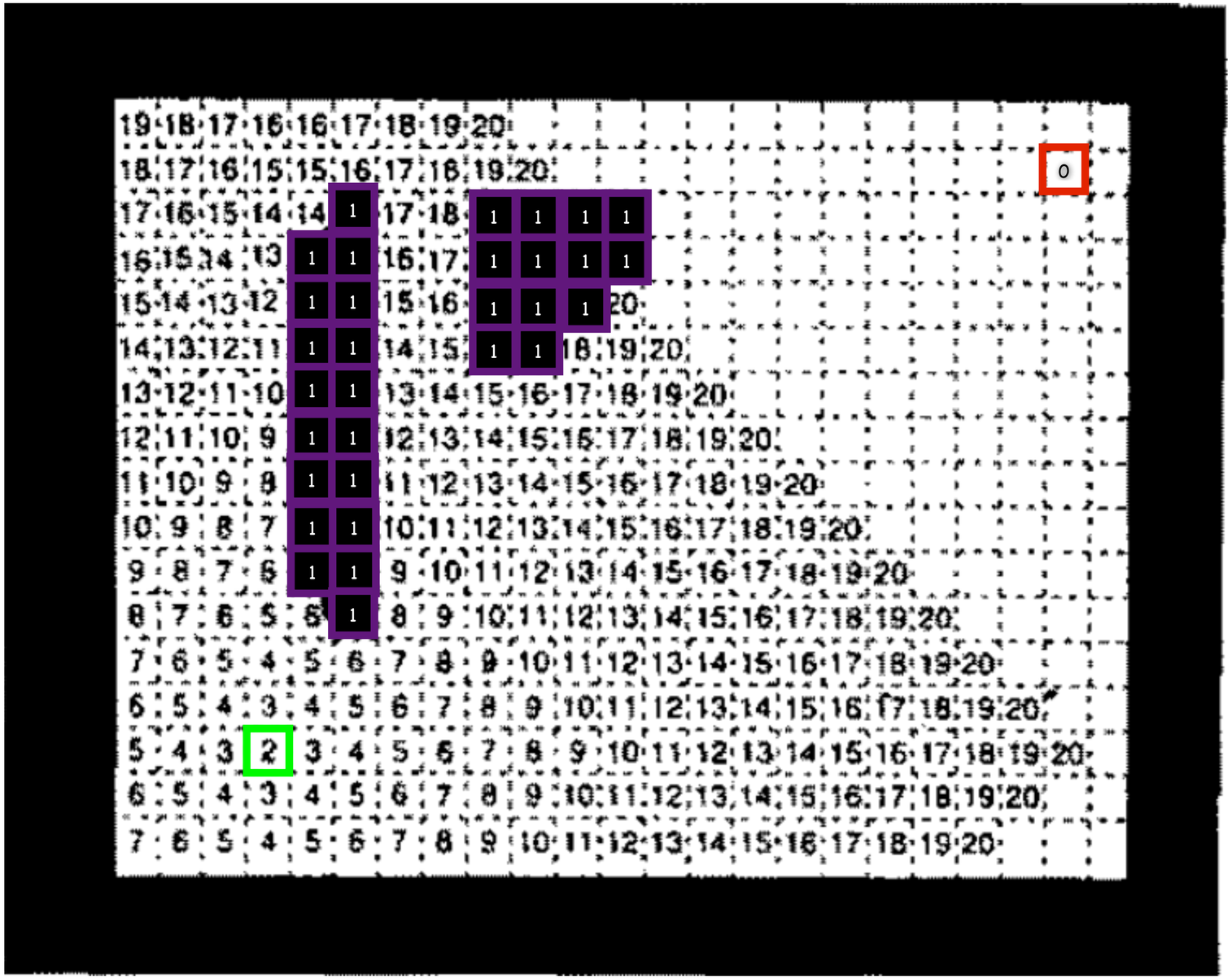

0
Brushfire away from goal:
energy at empty cells on frontier =
current energy at frontier +
change in cell energy (1 in this case)

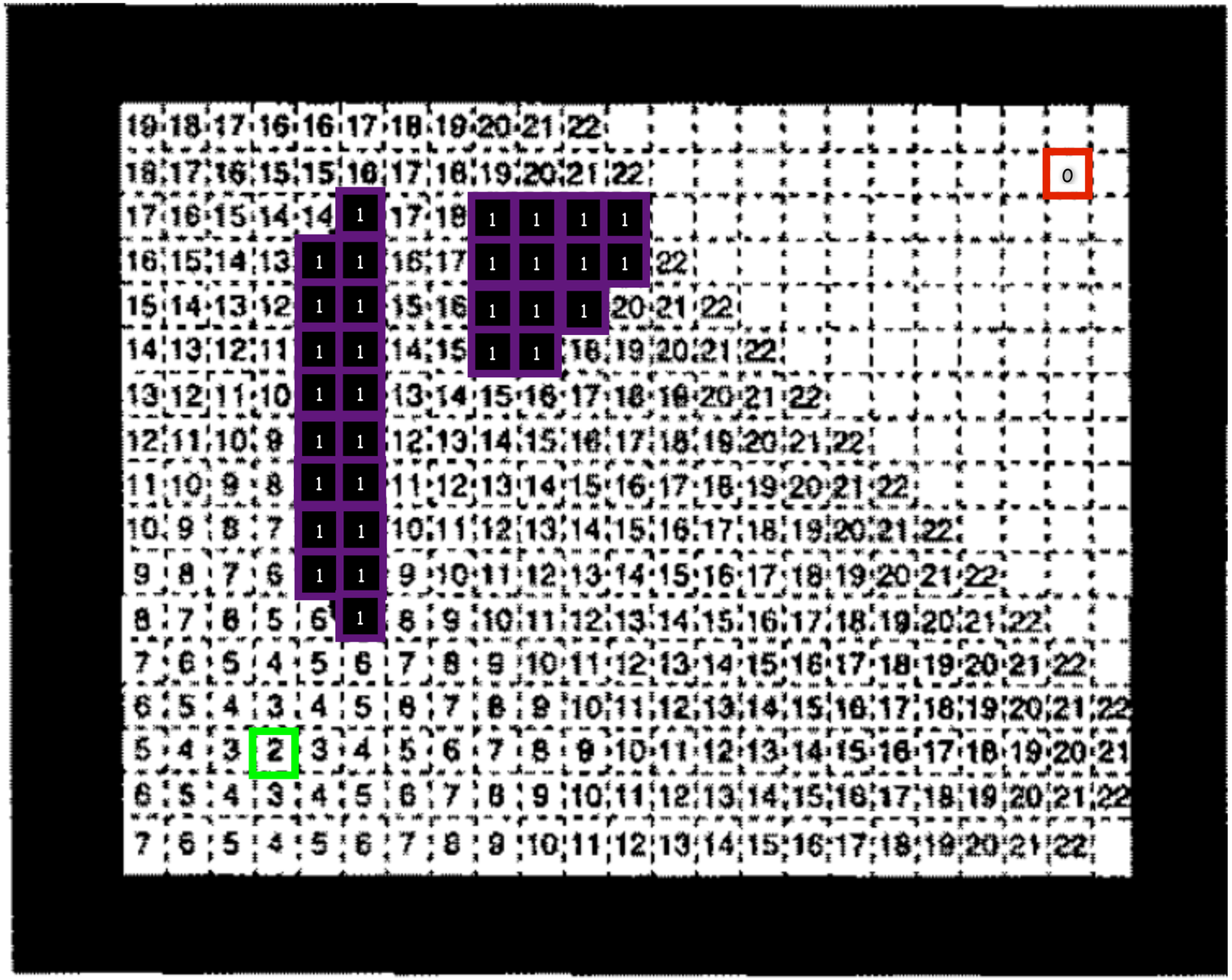


Slide borrowed from Michigan Robotics autorob.org

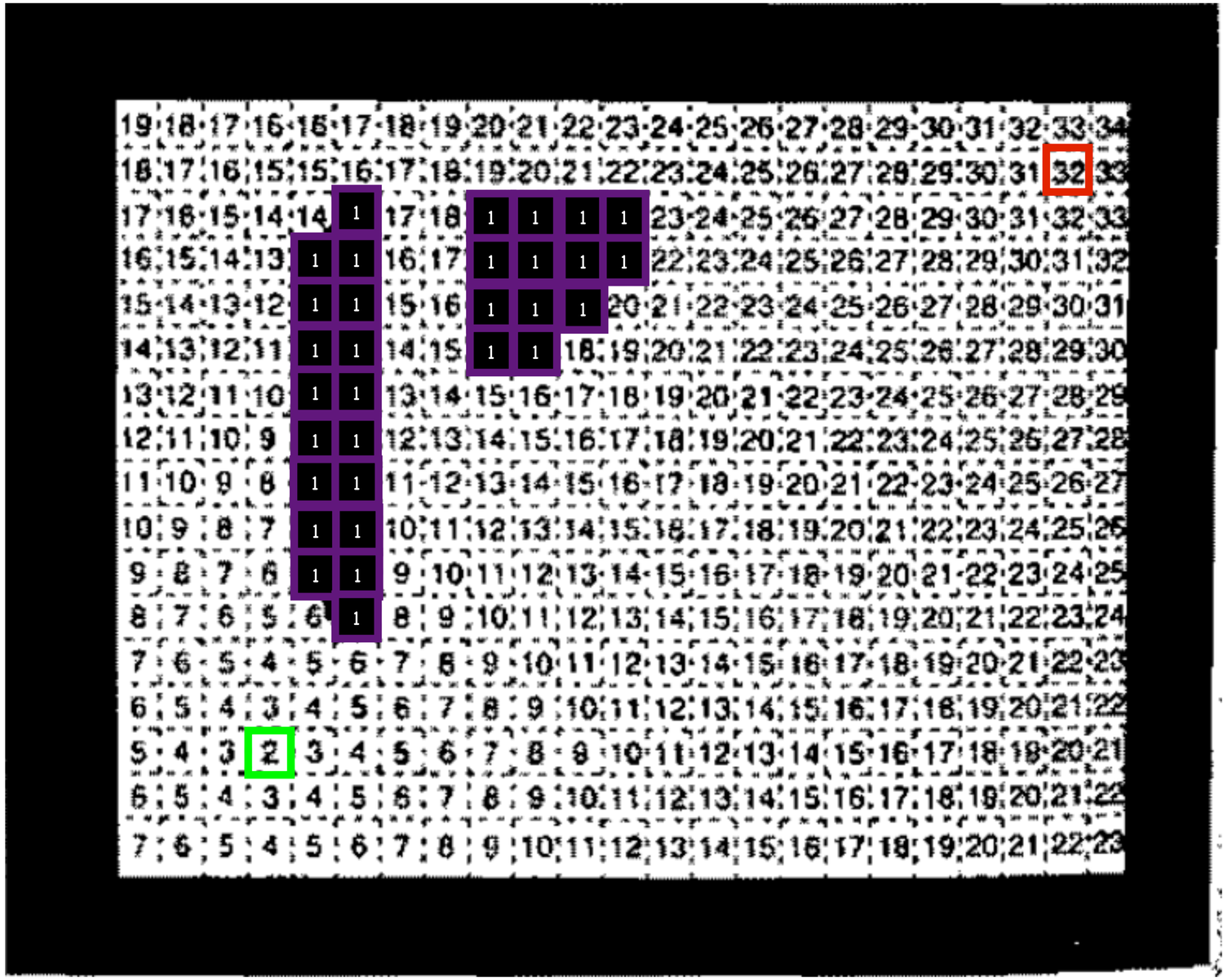


Slide borrowed from Michigan Robotics autorob.org

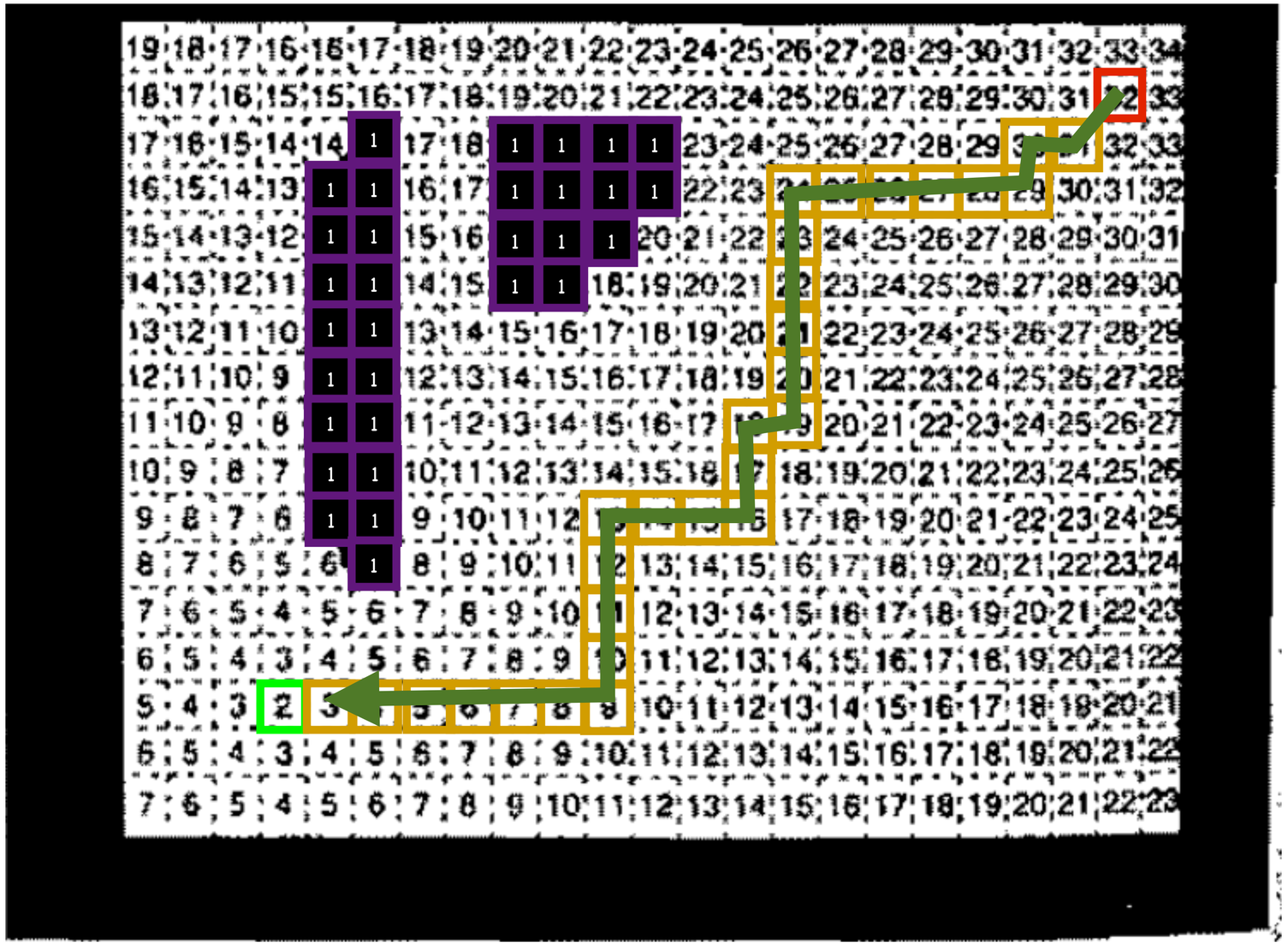

0
2



Slide borrowed from Michigan Robotics autorob.org



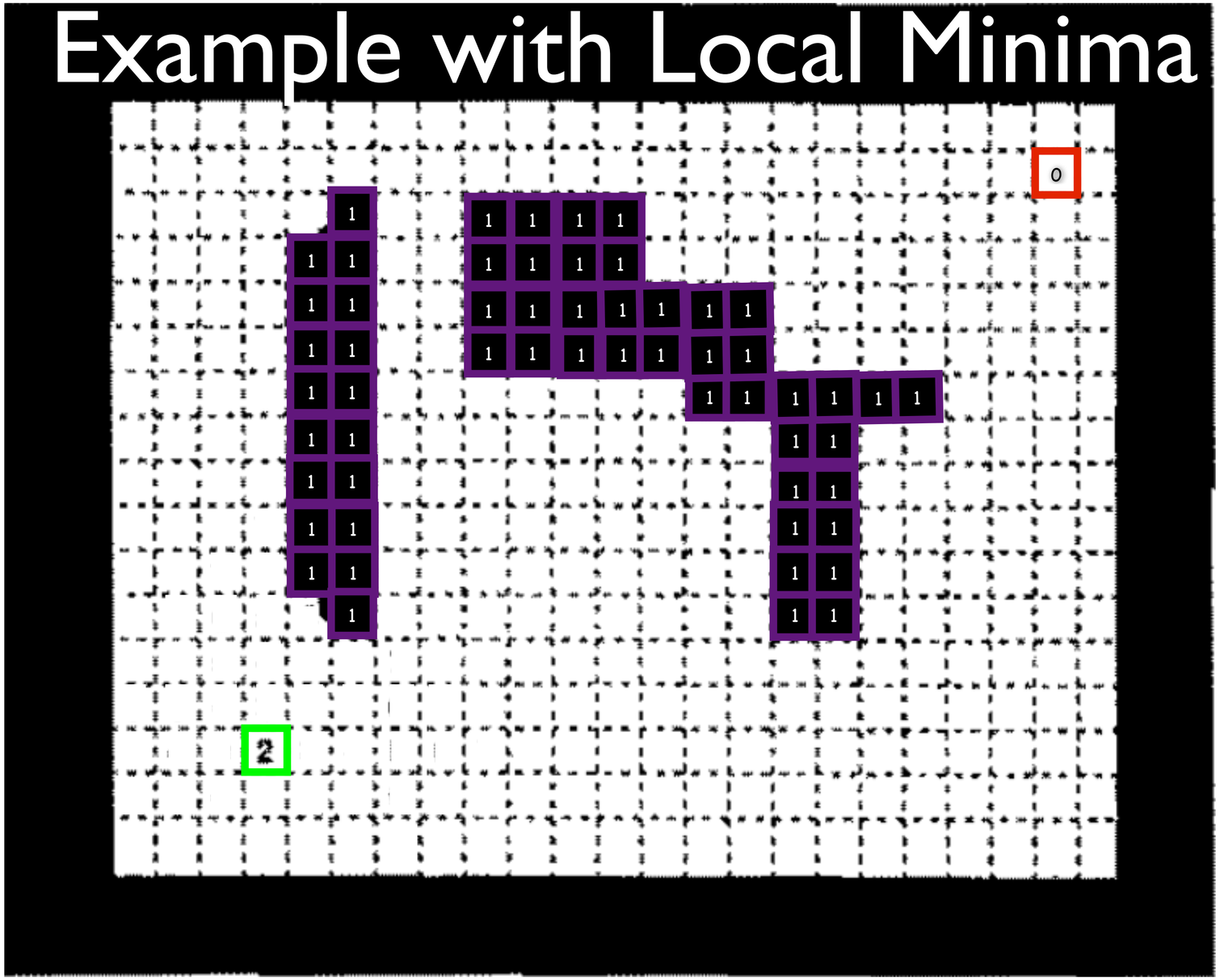
Once start reached, follow brushfire potential to goal

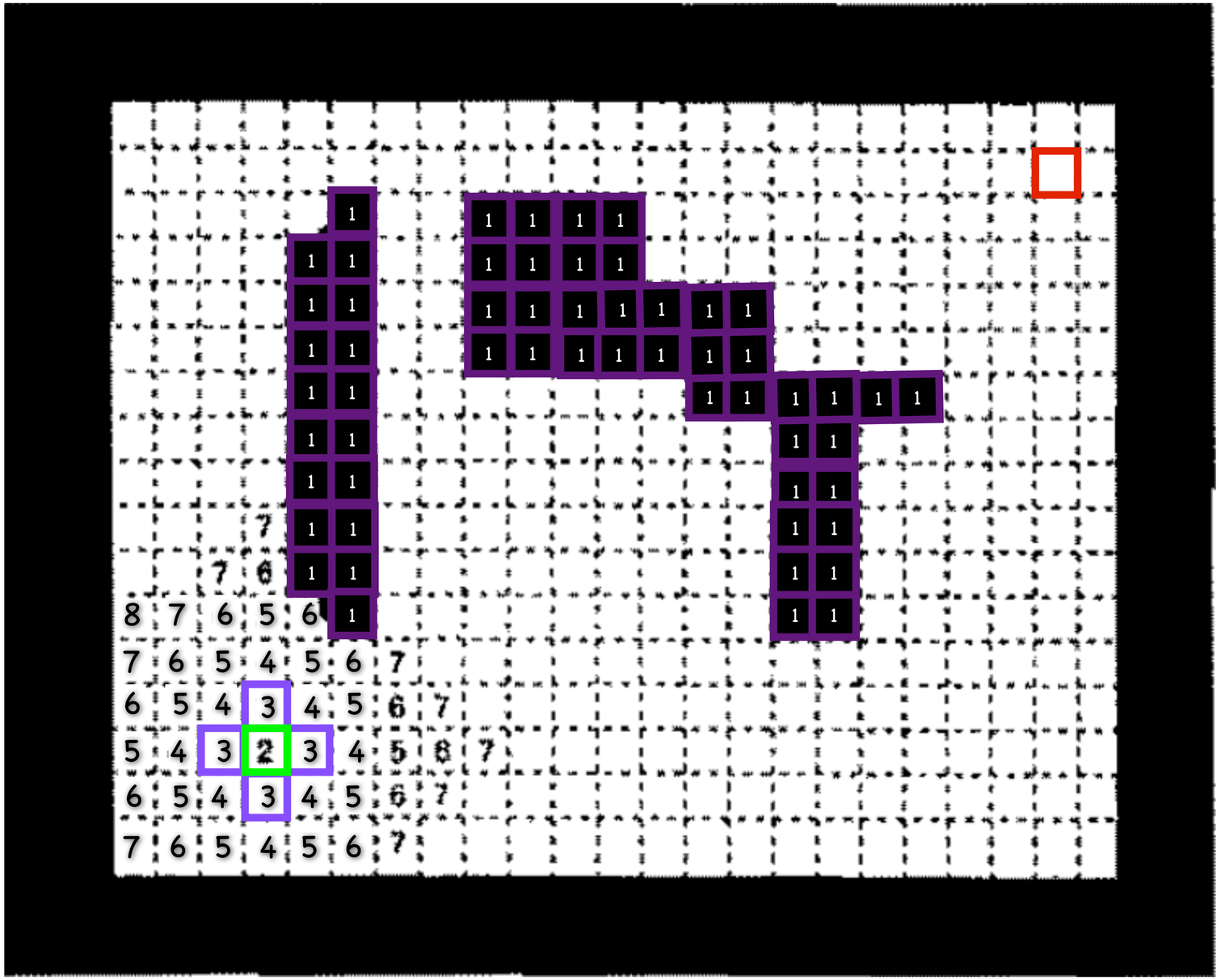


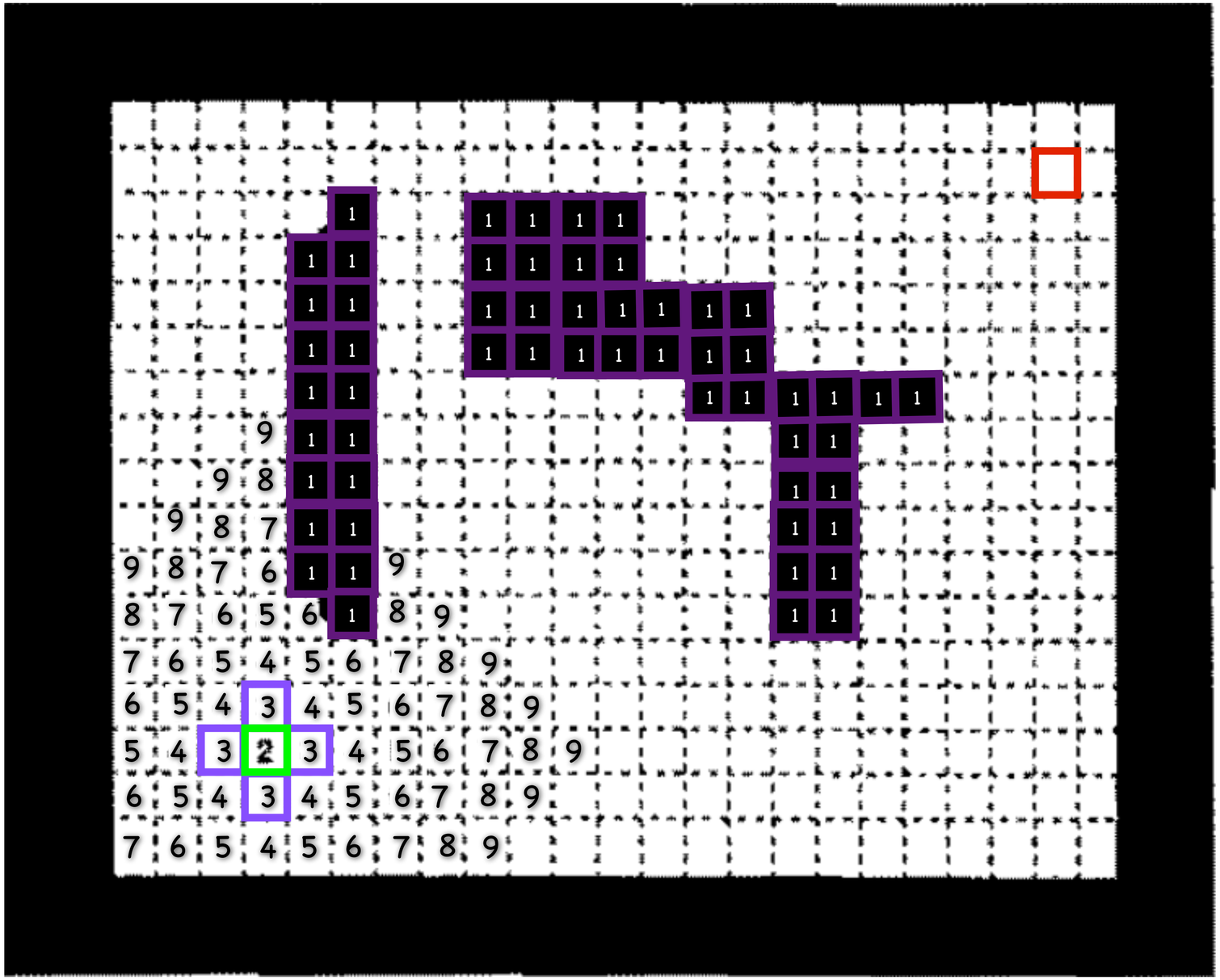
Example with Local Minima

Slide borrowed from Michigan Robotics autorob.org

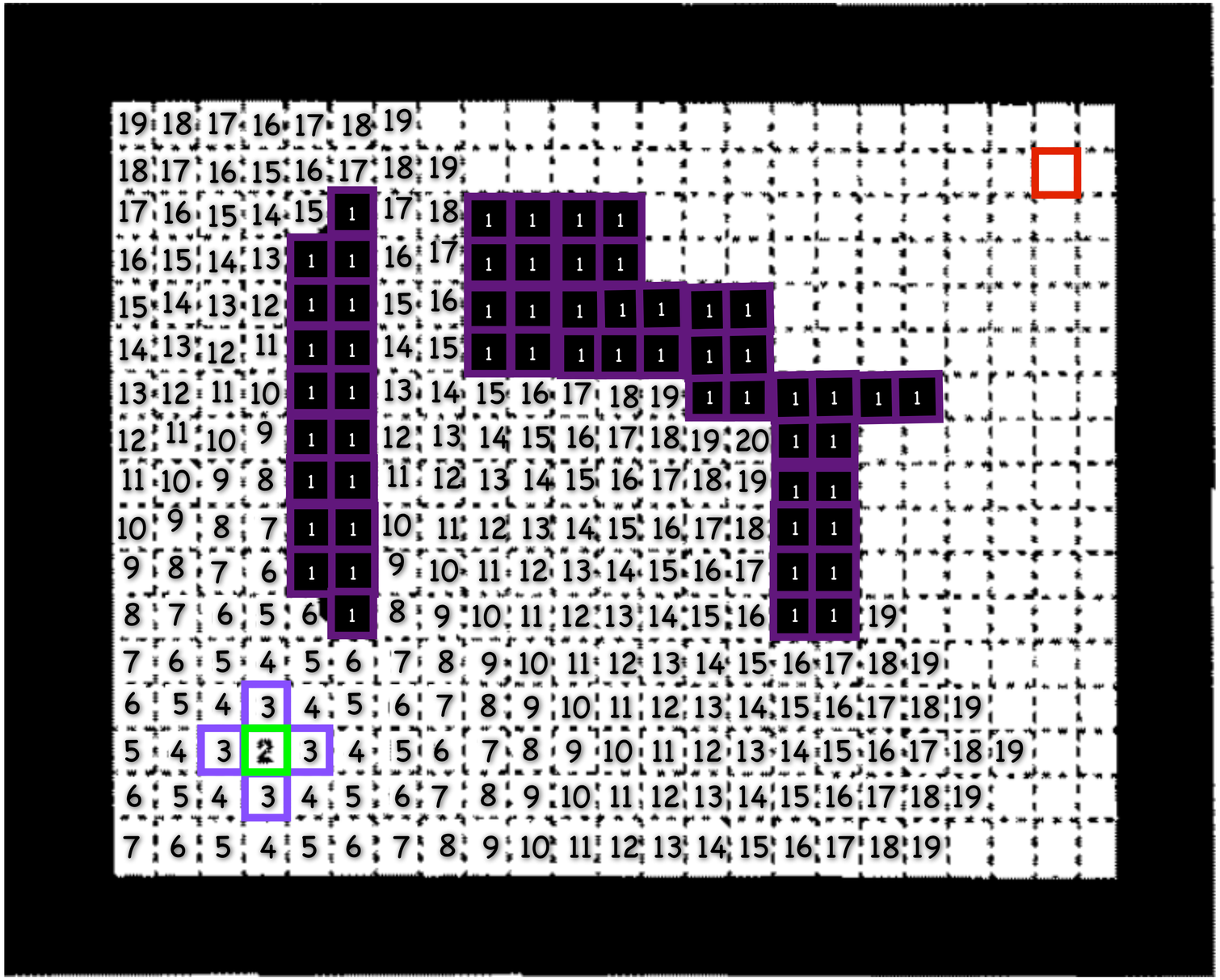
Example with Local Minima



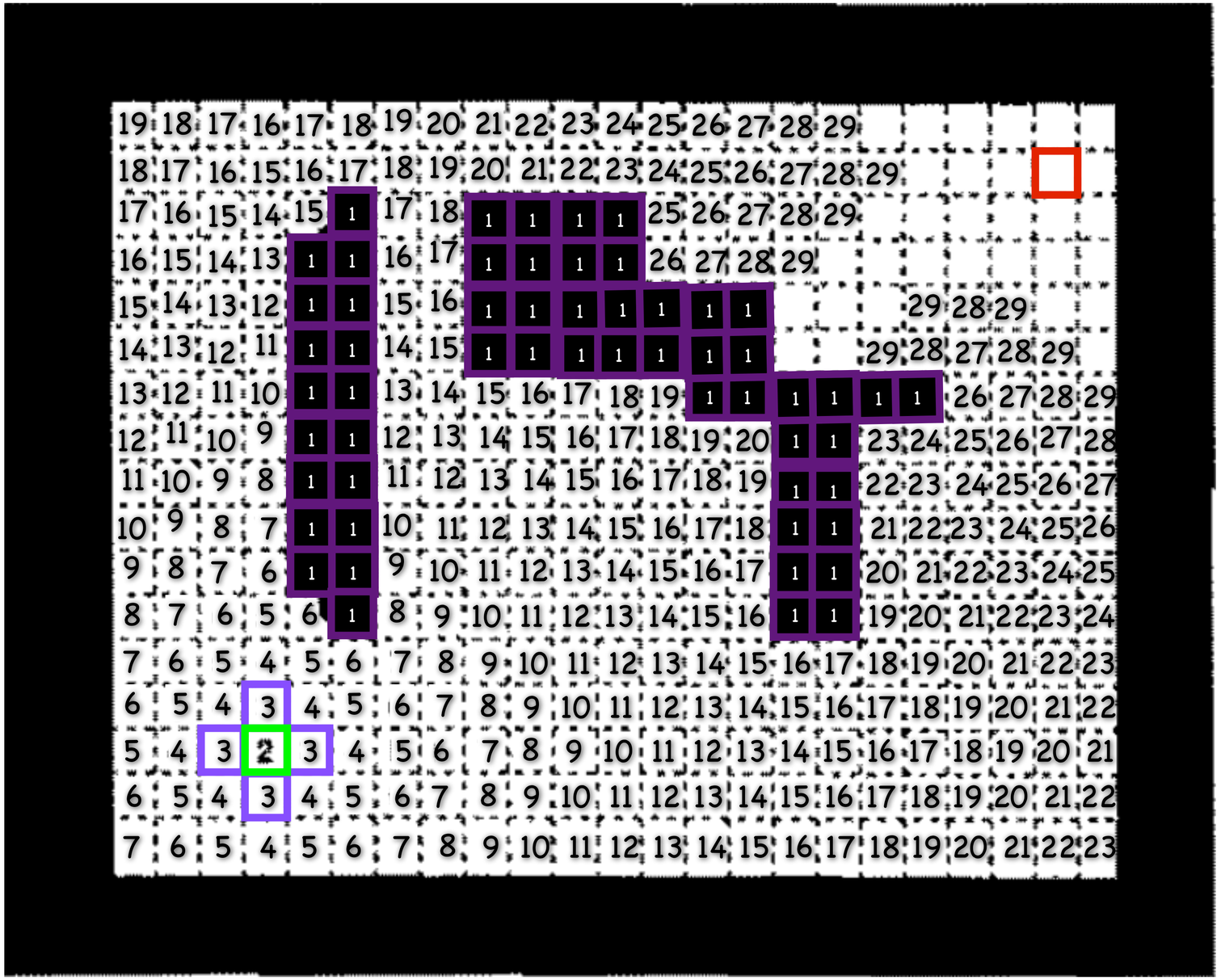


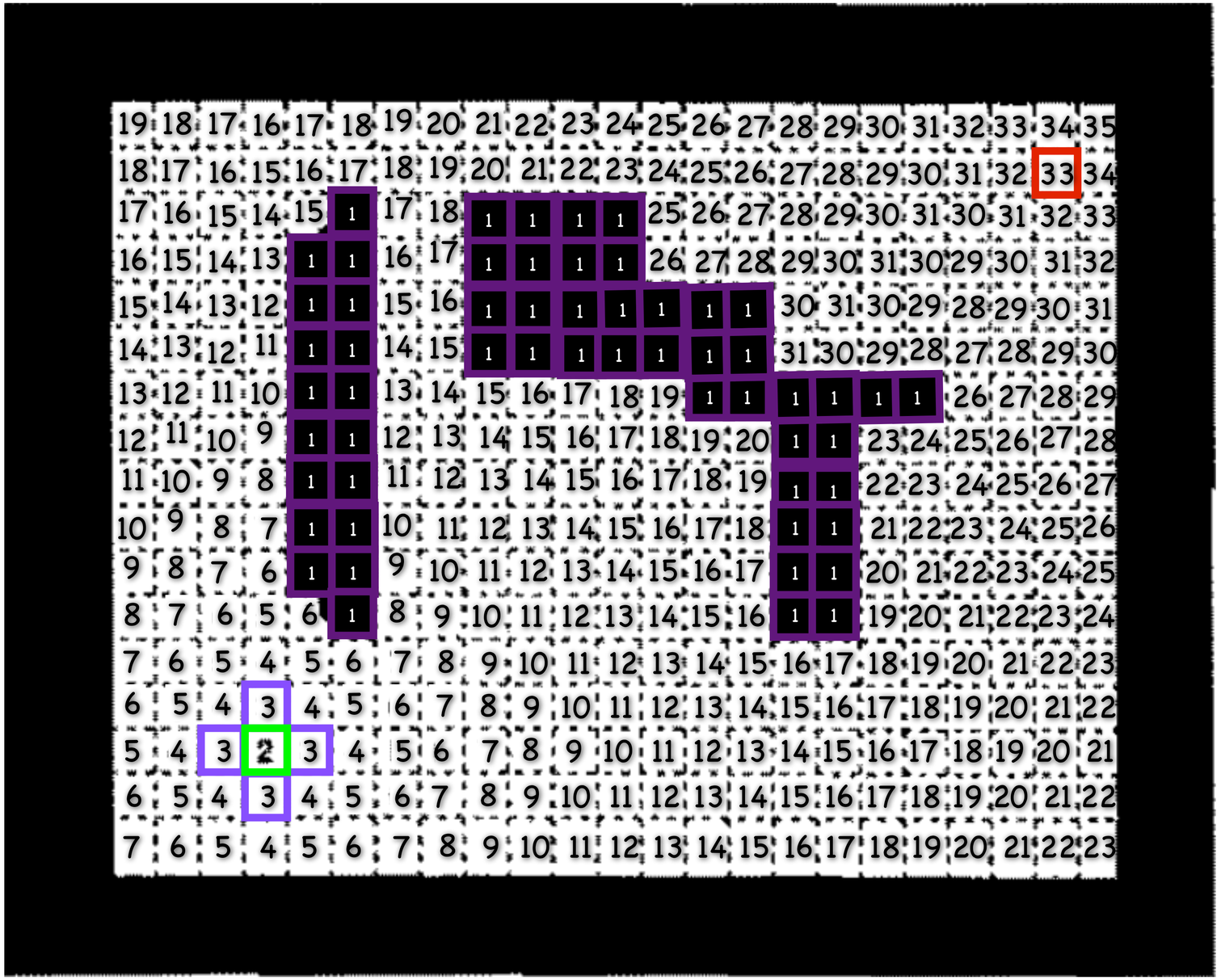


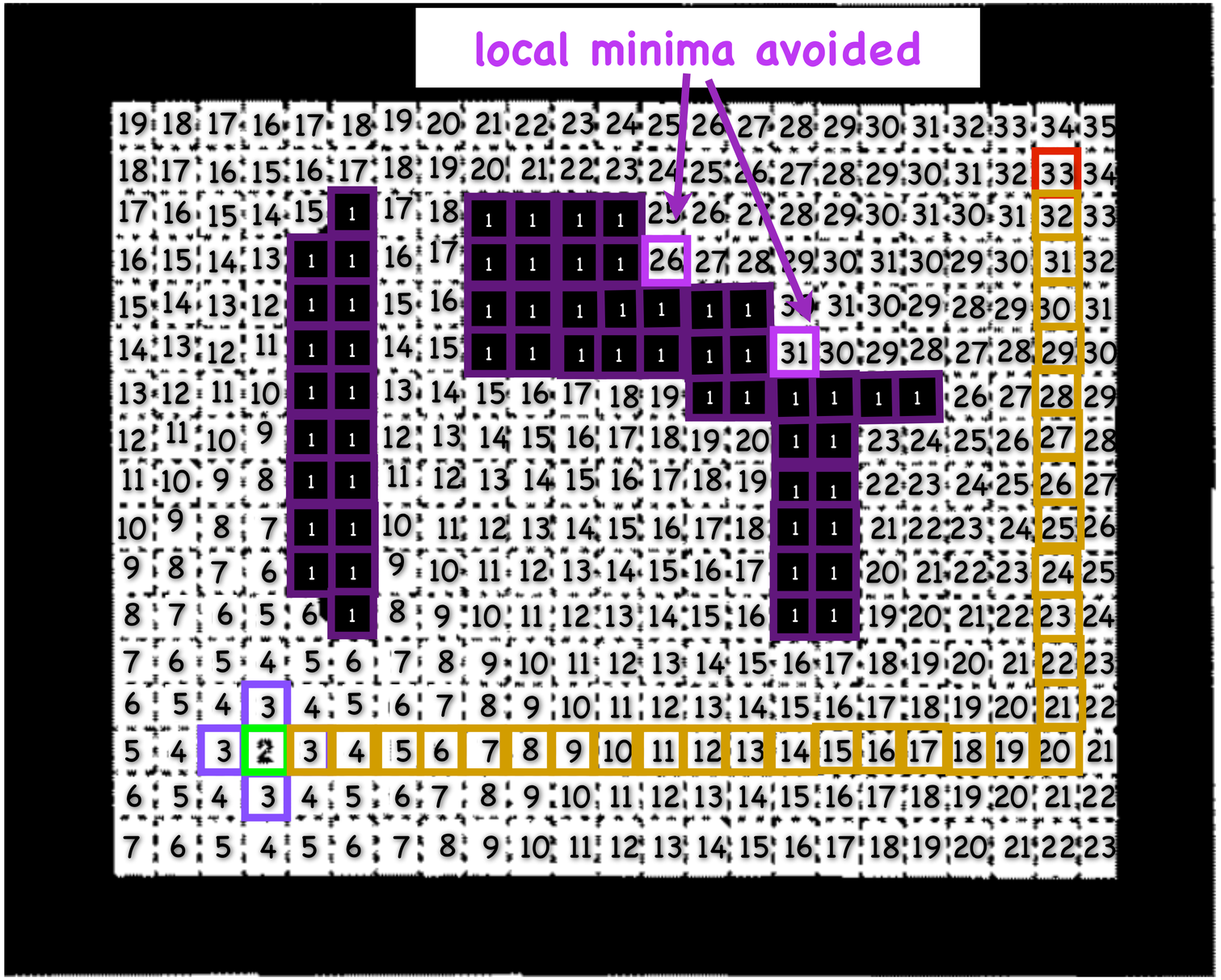
Slide borrowed from Michigan Robotics autorob.org



Slide borrowed from Michigan Robotics autorob.org







Slide borrowed from Michigan Robotics autorob.org

Navigation Recap

Slide borrowed from Michigan Robotics autorob.org

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

Slide borrowed from Michigan Robotics autorob.org

Next Lecture
Planning - VI - Collision Detection