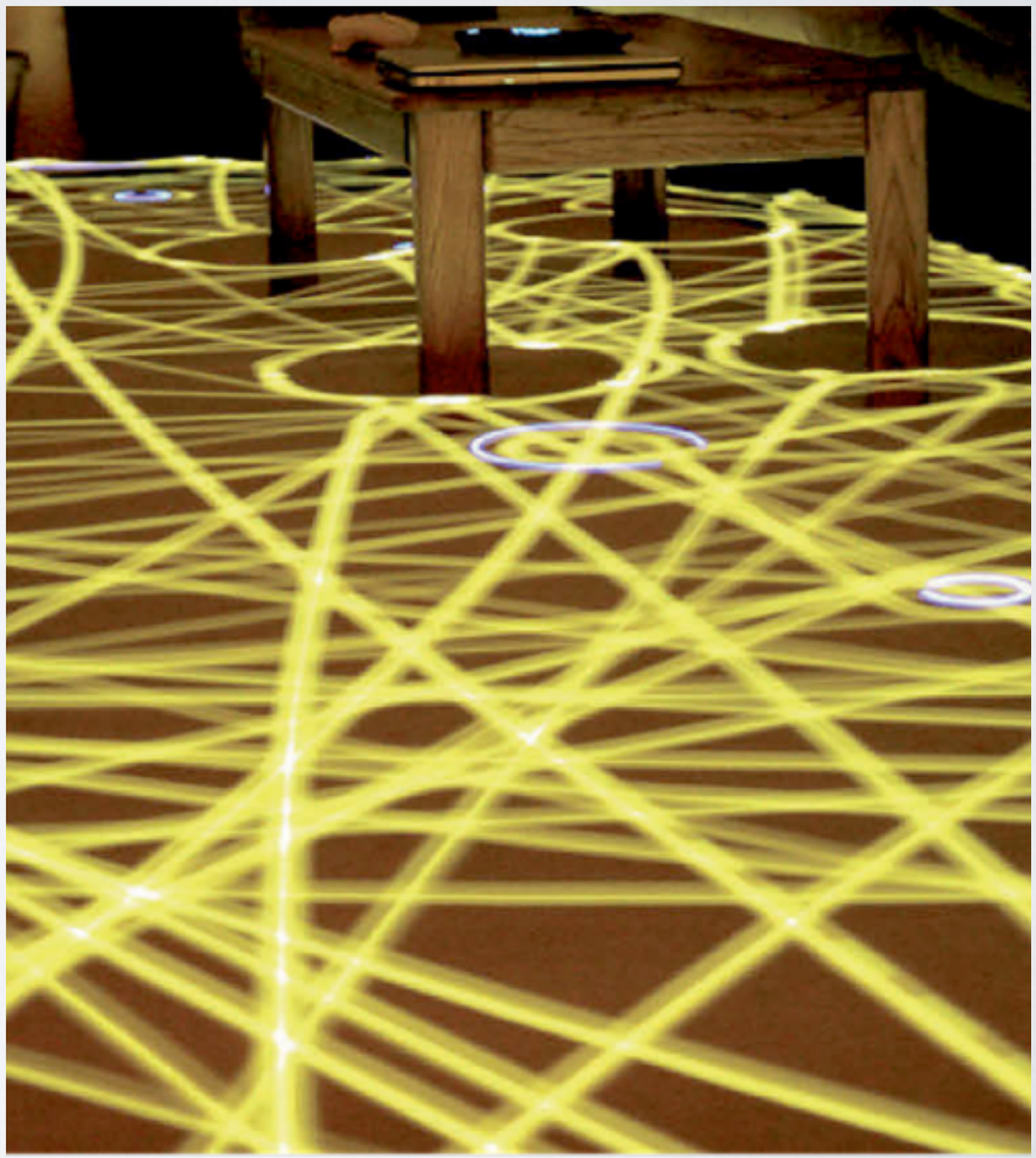
NAVIGATION

CS 3630 Introduction to Robotics and Perception

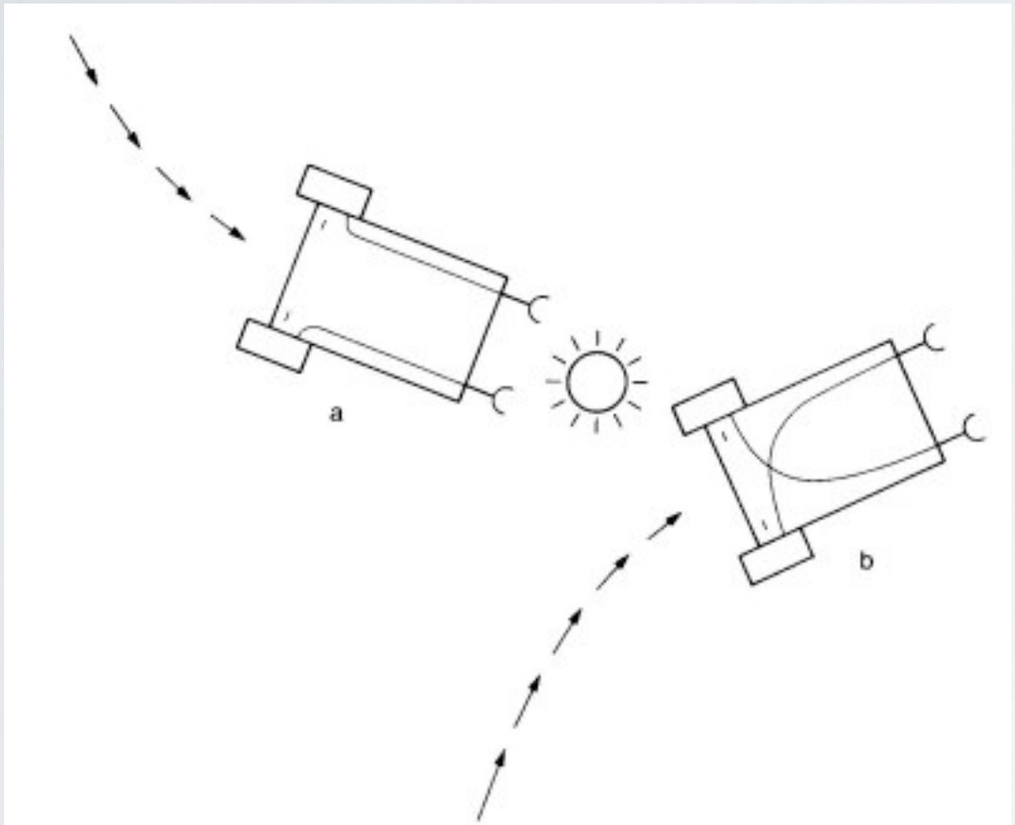
Frank Dellaert

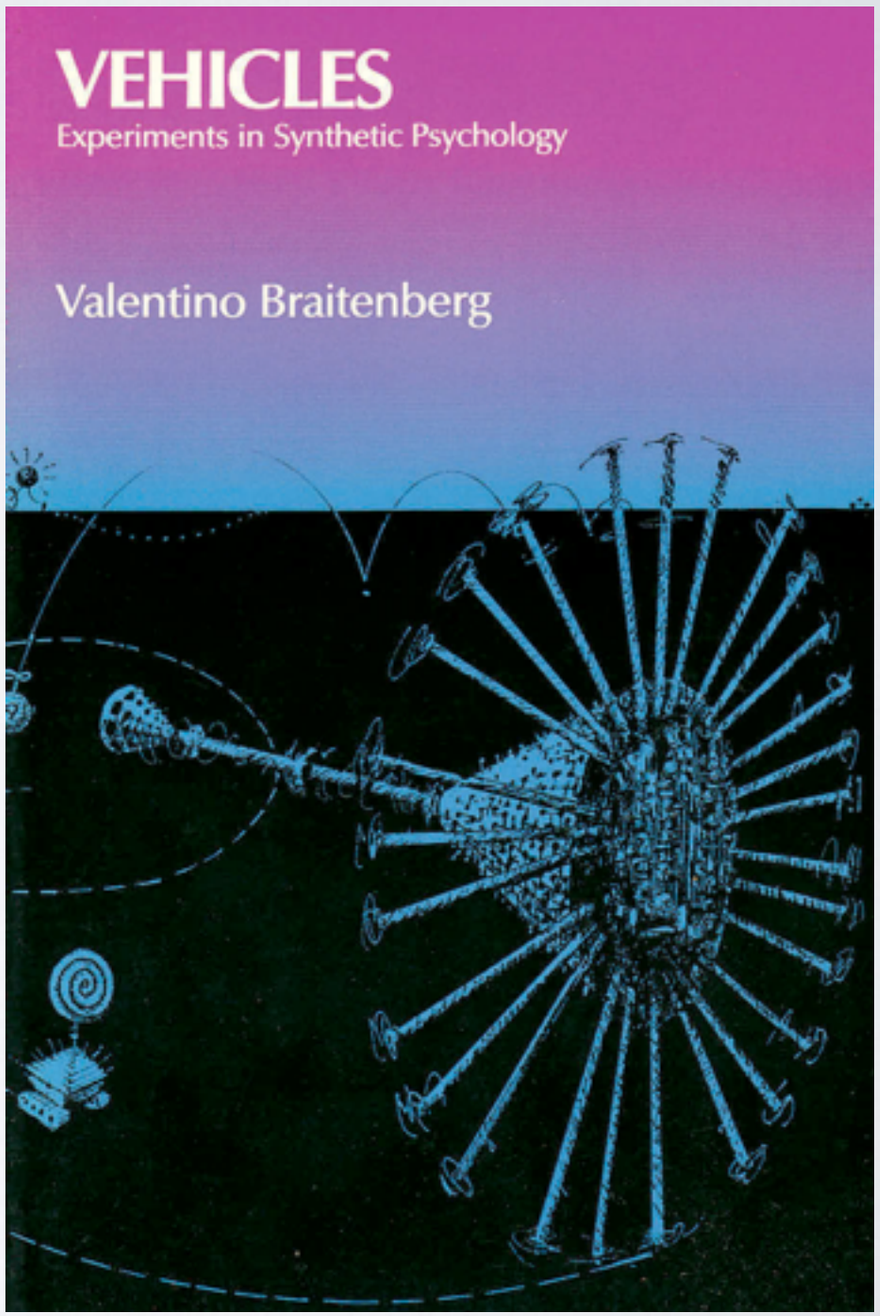
INTRODUCTION

- Map-based Navigation
  - human-like
  - requires a map
- **Reactive Navigation**
  - Elsie
  - Roomba



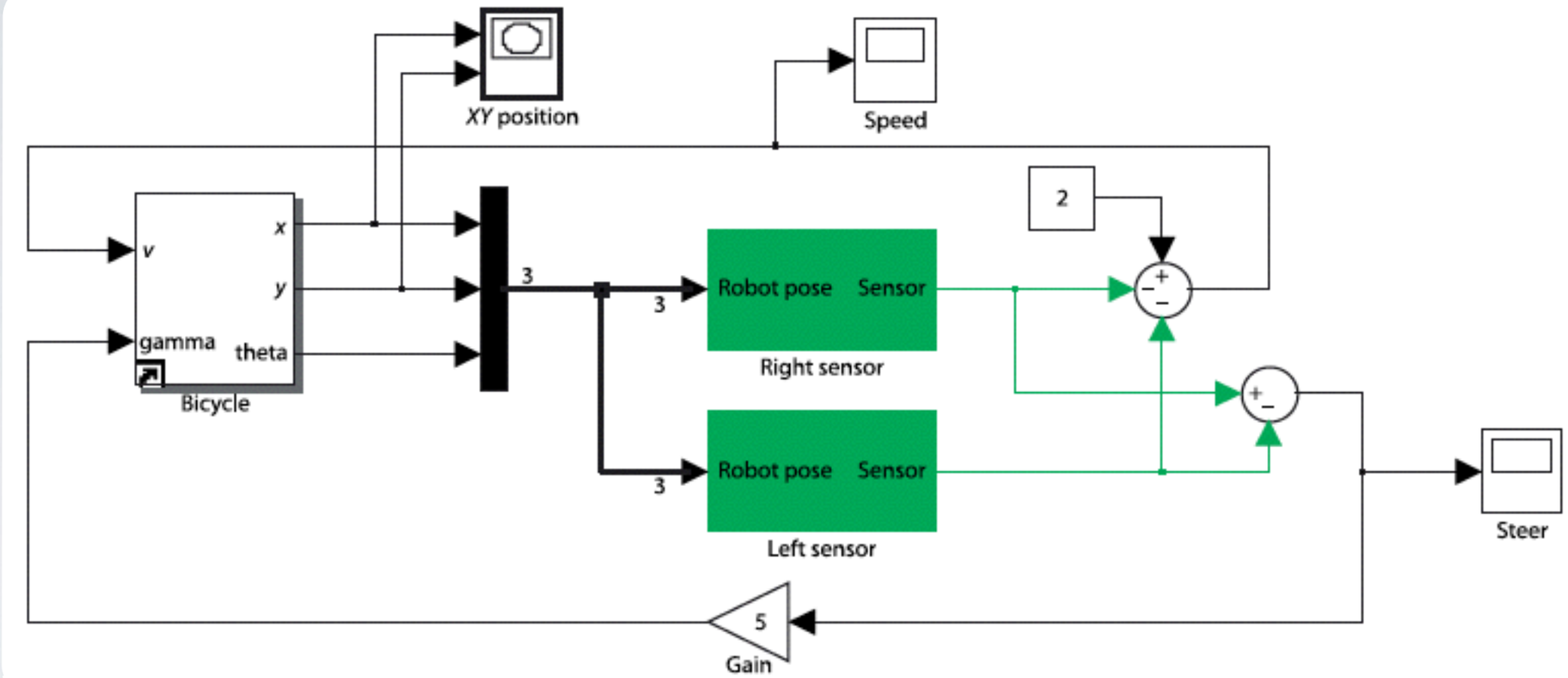
BRAITENBERG VEHICLES





- Love-Hate
- Simple Connections
- No (explicit) plan

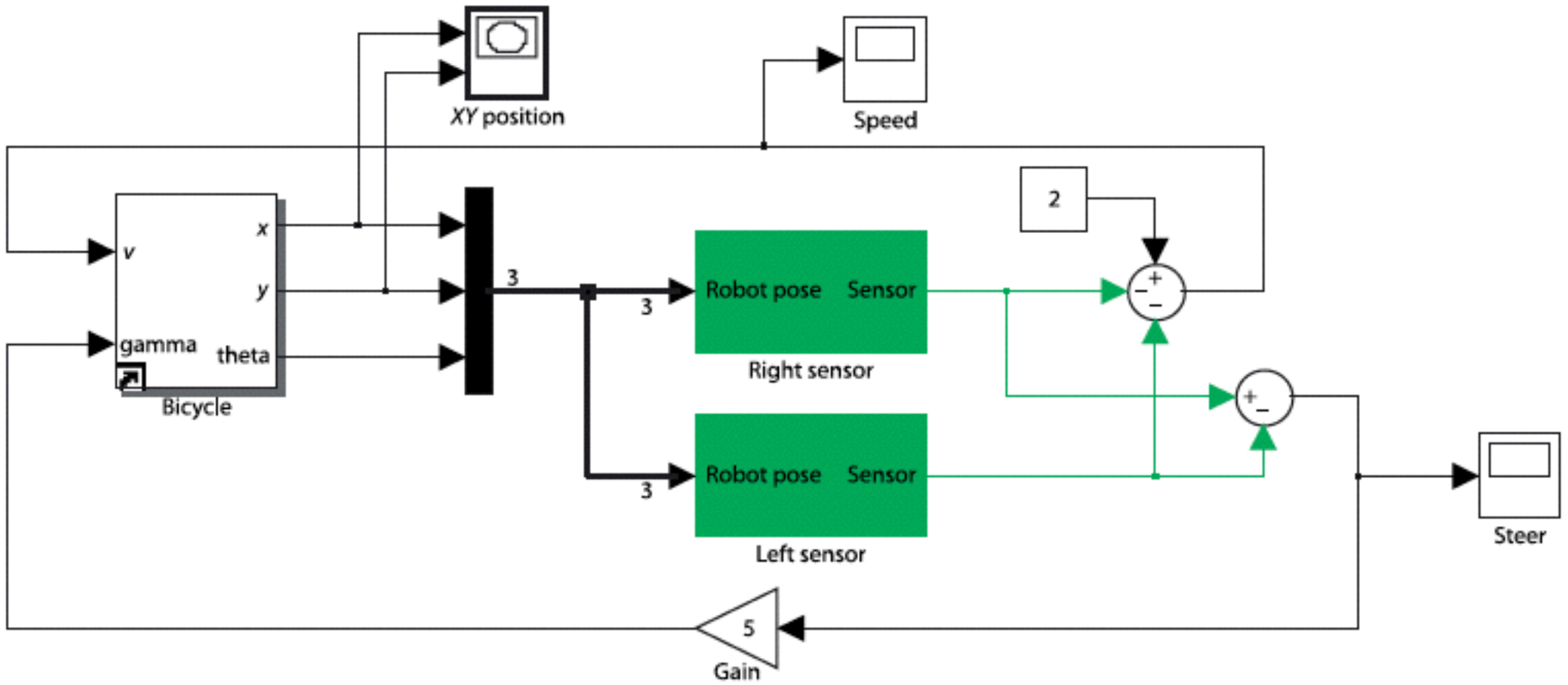
RVC BRAITNAV




```
1   function sensor = sensorfield(x, y)
2       xc = 60; yc = 90;
3       sensor = 200./((x-xc).^2 + (y-yc).^2 + 200);
```

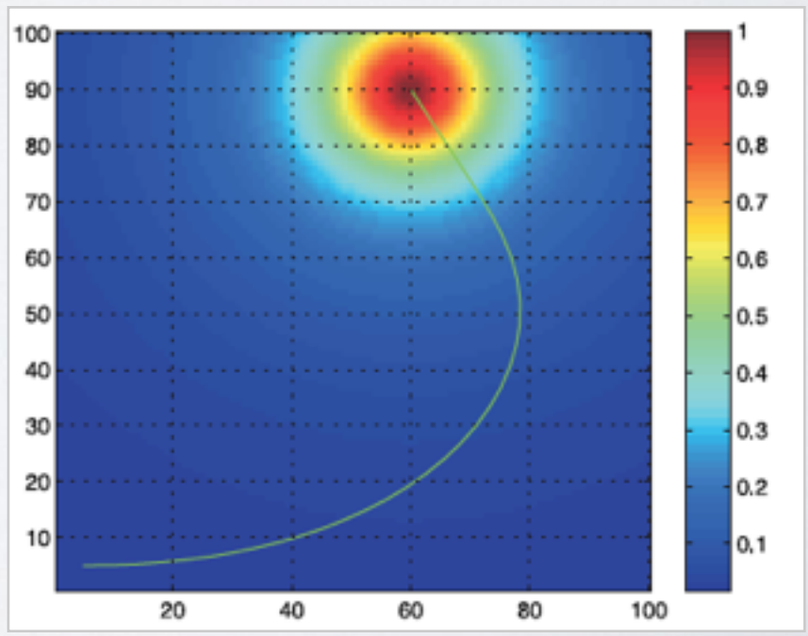
- $v = 2 - (s_r + s_l)$
- $g = k(s_r - s_l)$
- How to do this with Scribbler?
- Elsie had 4 behaviors (touch)

RVC BRAITNAV



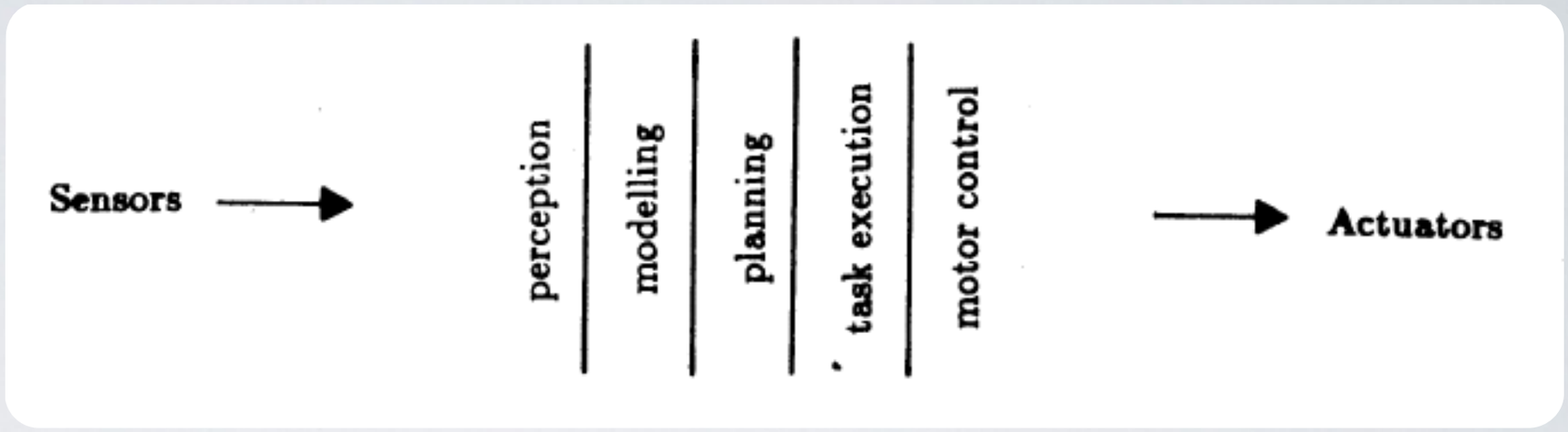

```
1   function sensor = sensorfield(x, y)
2       xc = 60; yc = 90;
3       sensor = 200./((x-xc).^2 + (y-yc).^2 + 200);
```

- $v = 2 - (s_r + s_l)$
- $g = k(s_r - s_l)$
- How to do this with Scribbler?
- Elsie had 4 behaviors (touch)



Rodney Brooks

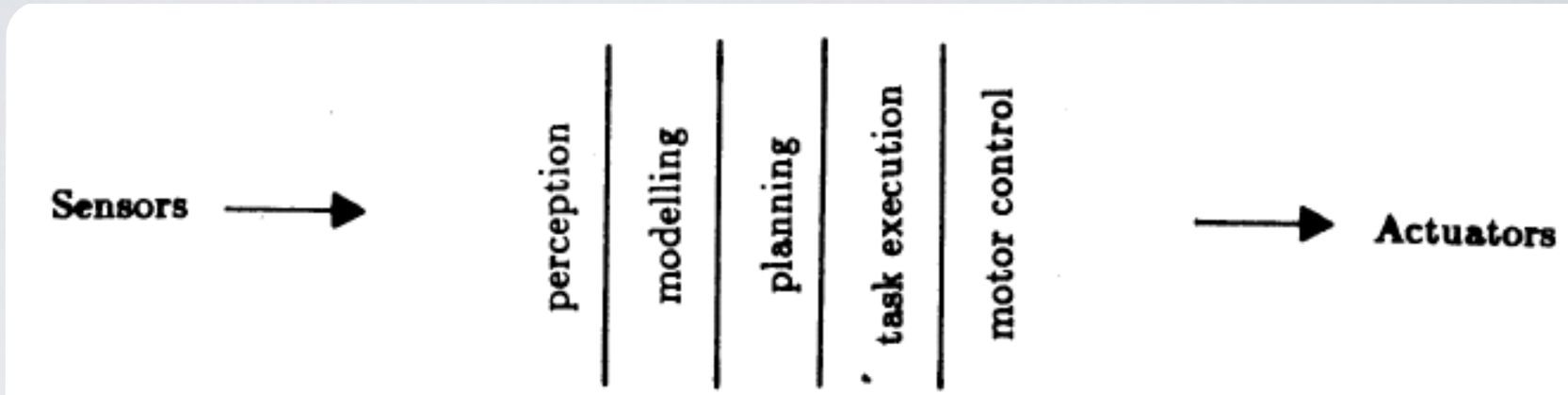
SUBSUMPTION ARCHITECTURE

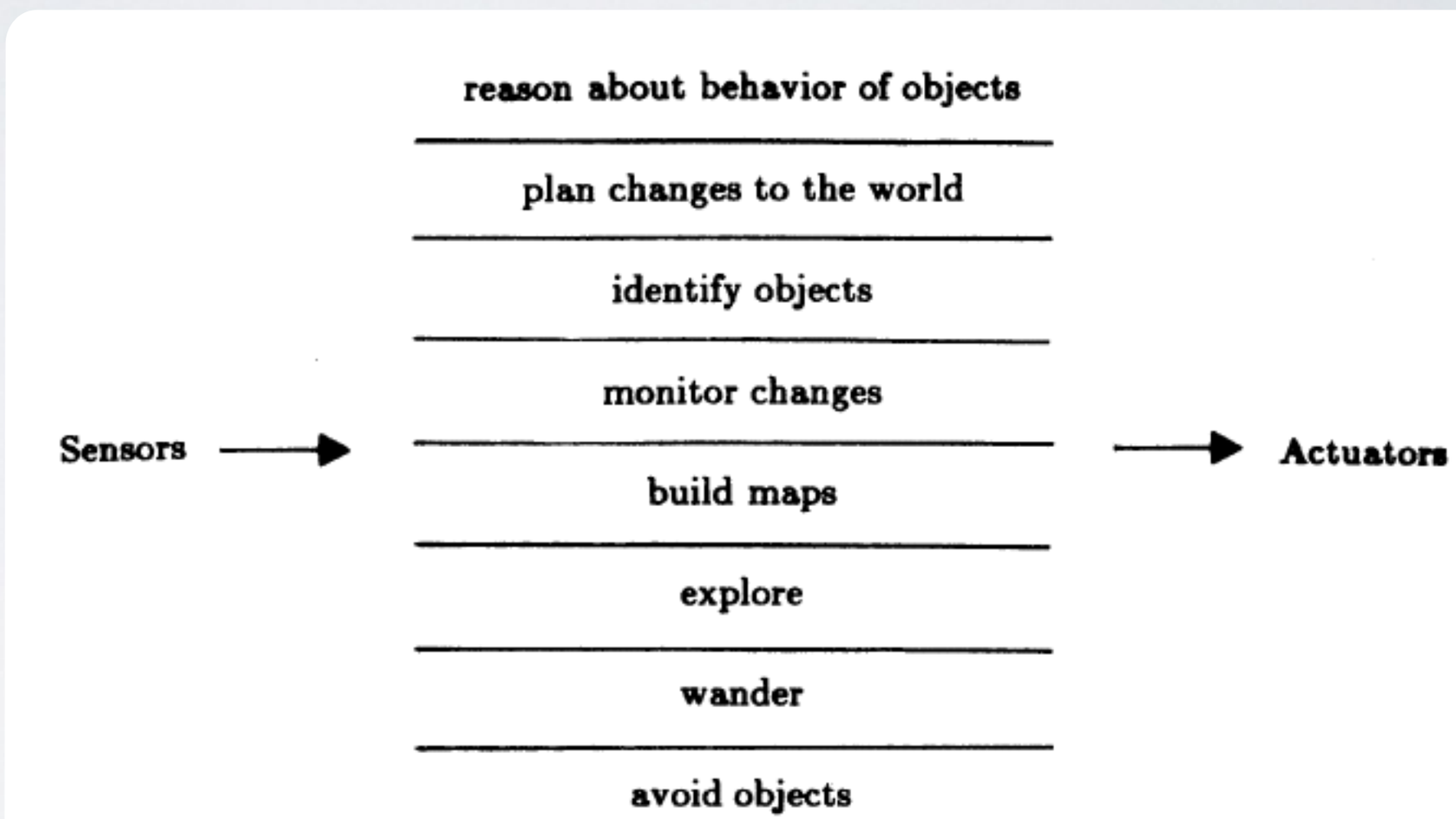


Rodney Brooks

SUBSUMPTION ARCHITECTURE

Rodney Brooks





SUBSUMPTION ARCHITECTURE

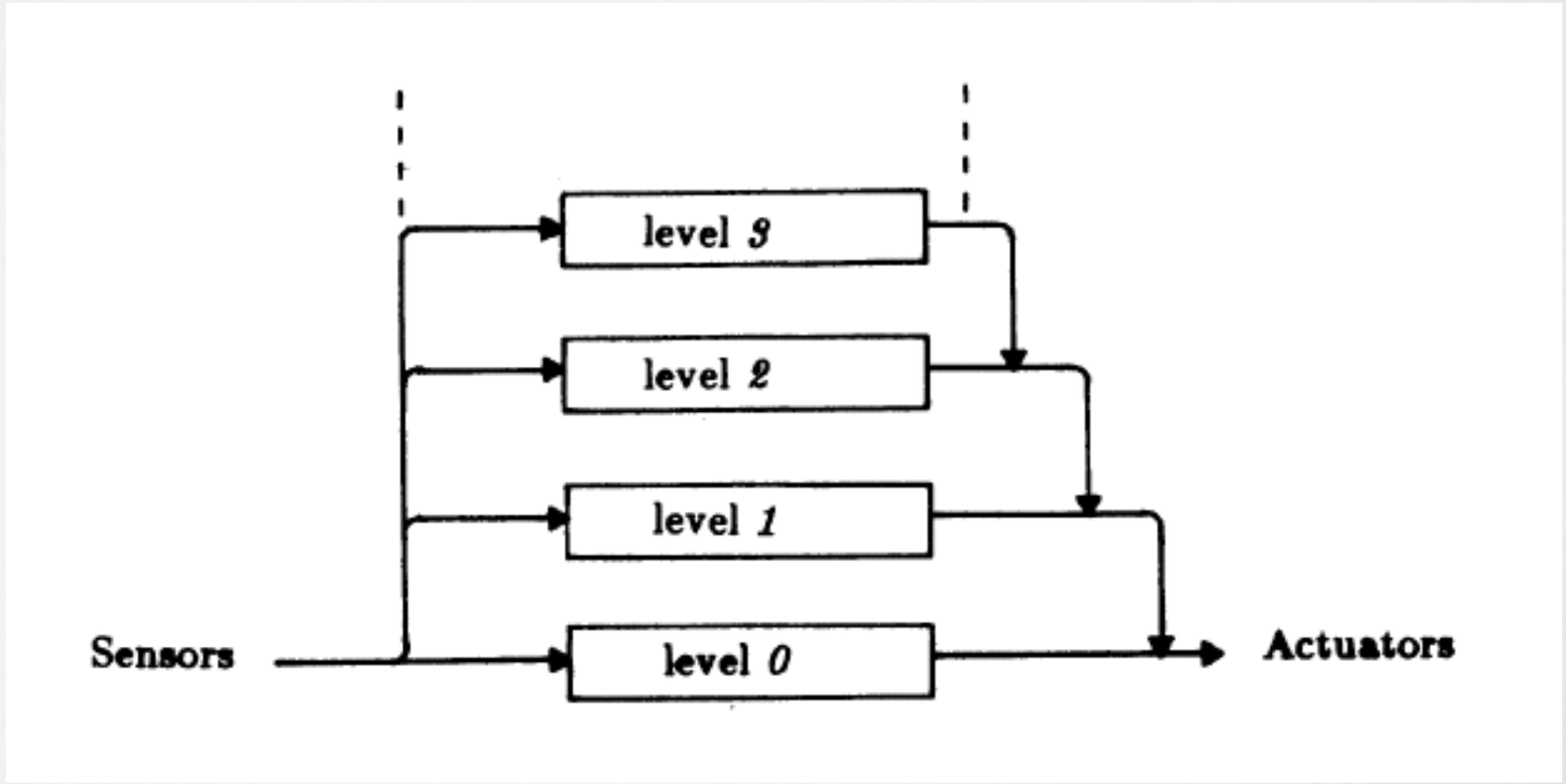
LEVELS OF COMPETENCE

- Assumptions:
  - Simple control, complex behavior
  - Relative, not absolute
  - Real world
  - Robust

LEVELS OF COMPETENCE

- Assumptions:
  - Simple control, complex behavior
  - Relative, not absolute
  - Real world
- Robust

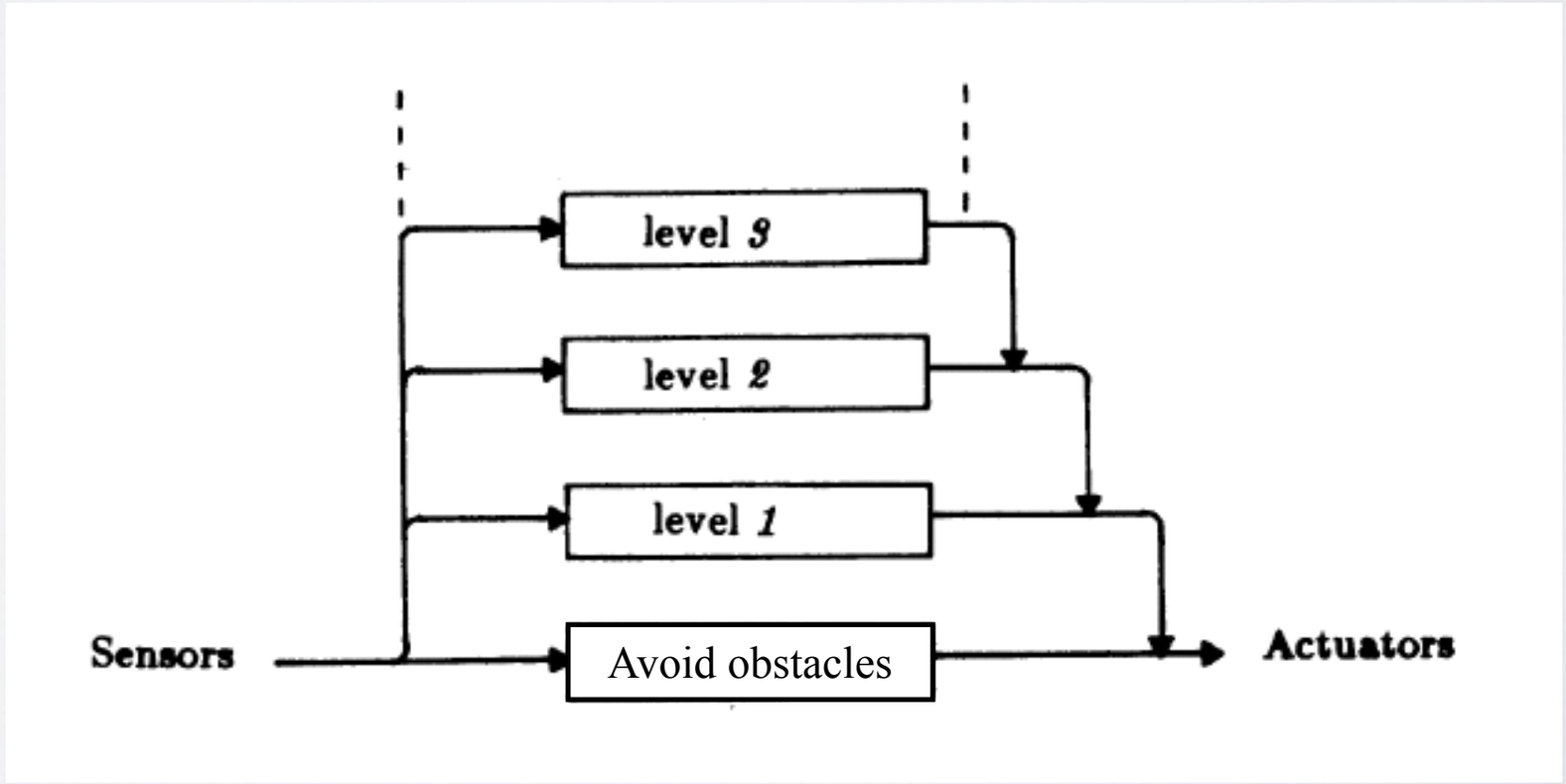
Subsumption:



LEVELS OF COMPETENCE

- Assumptions:
  - Simple control, complex behavior
  - Relative, not absolute
  - Real world
- Robust

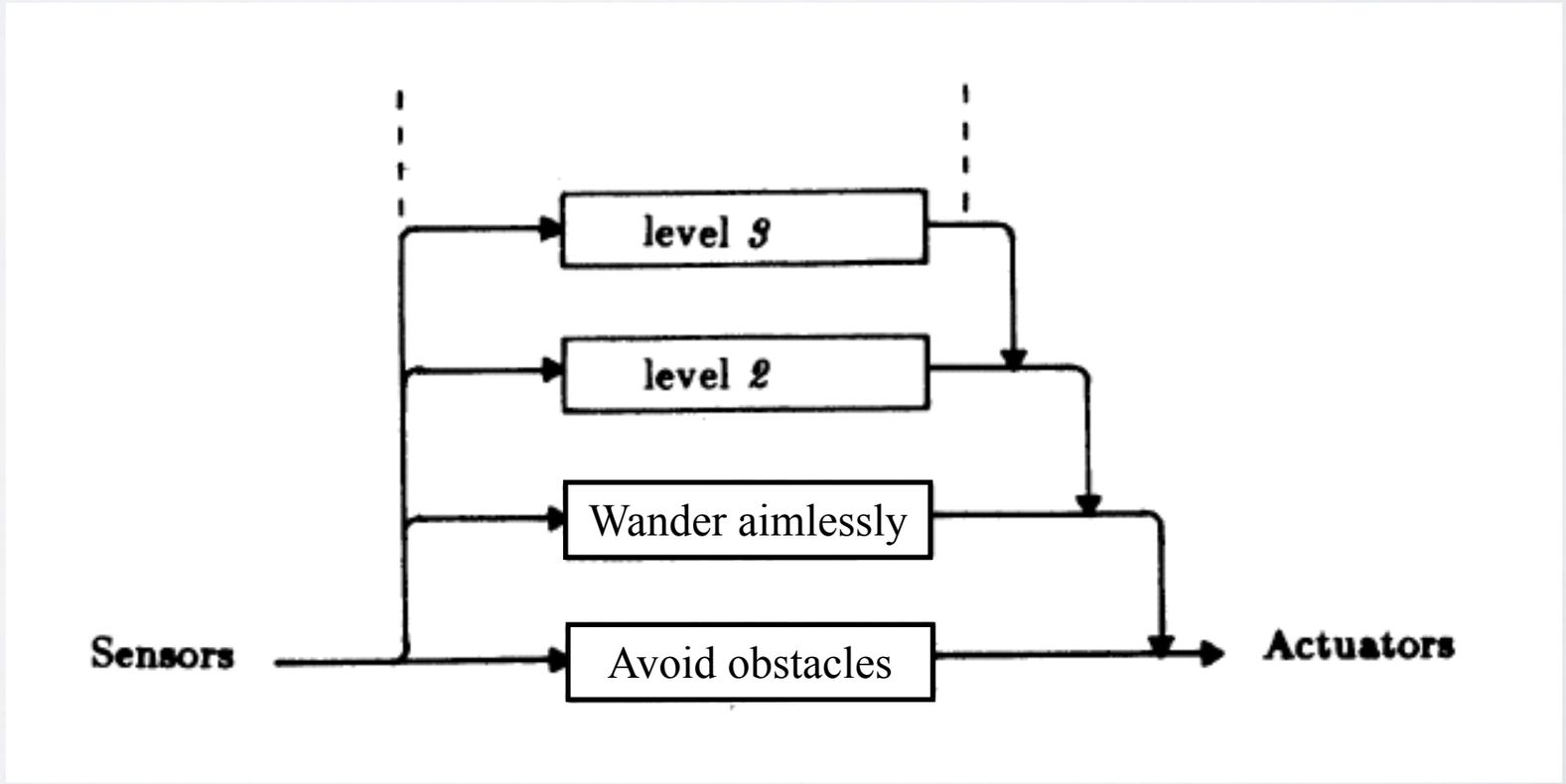
Subsumption:



LEVELS OF COMPETENCE

- Assumptions:
  - Simple control, complex behavior
  - Relative, not absolute
  - Real world
- Robust

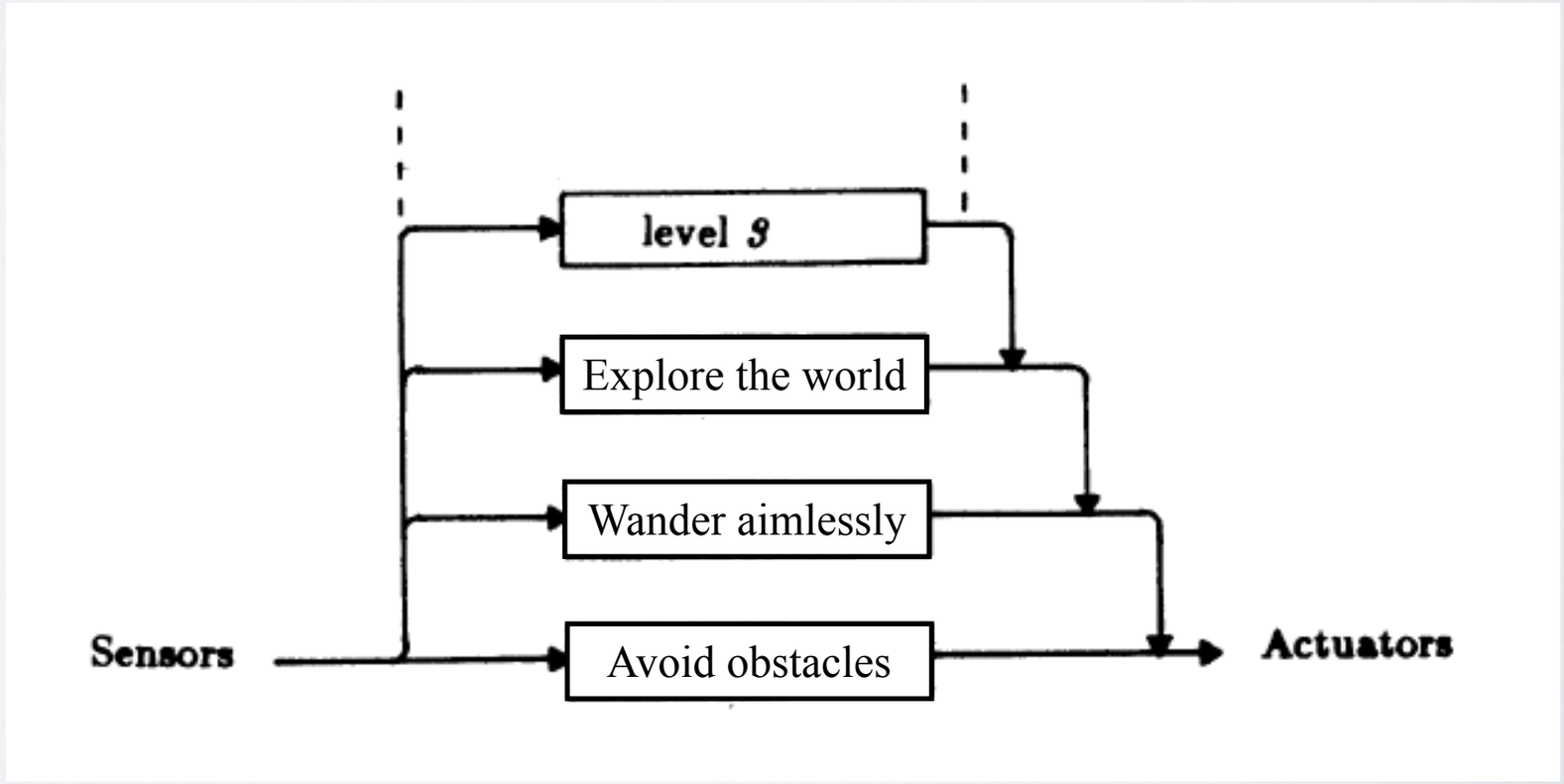
Subsumption:



LEVELS OF COMPETENCE

- Assumptions:
  - Simple control, complex behavior
  - Relative, not absolute
  - Real world
- Robust

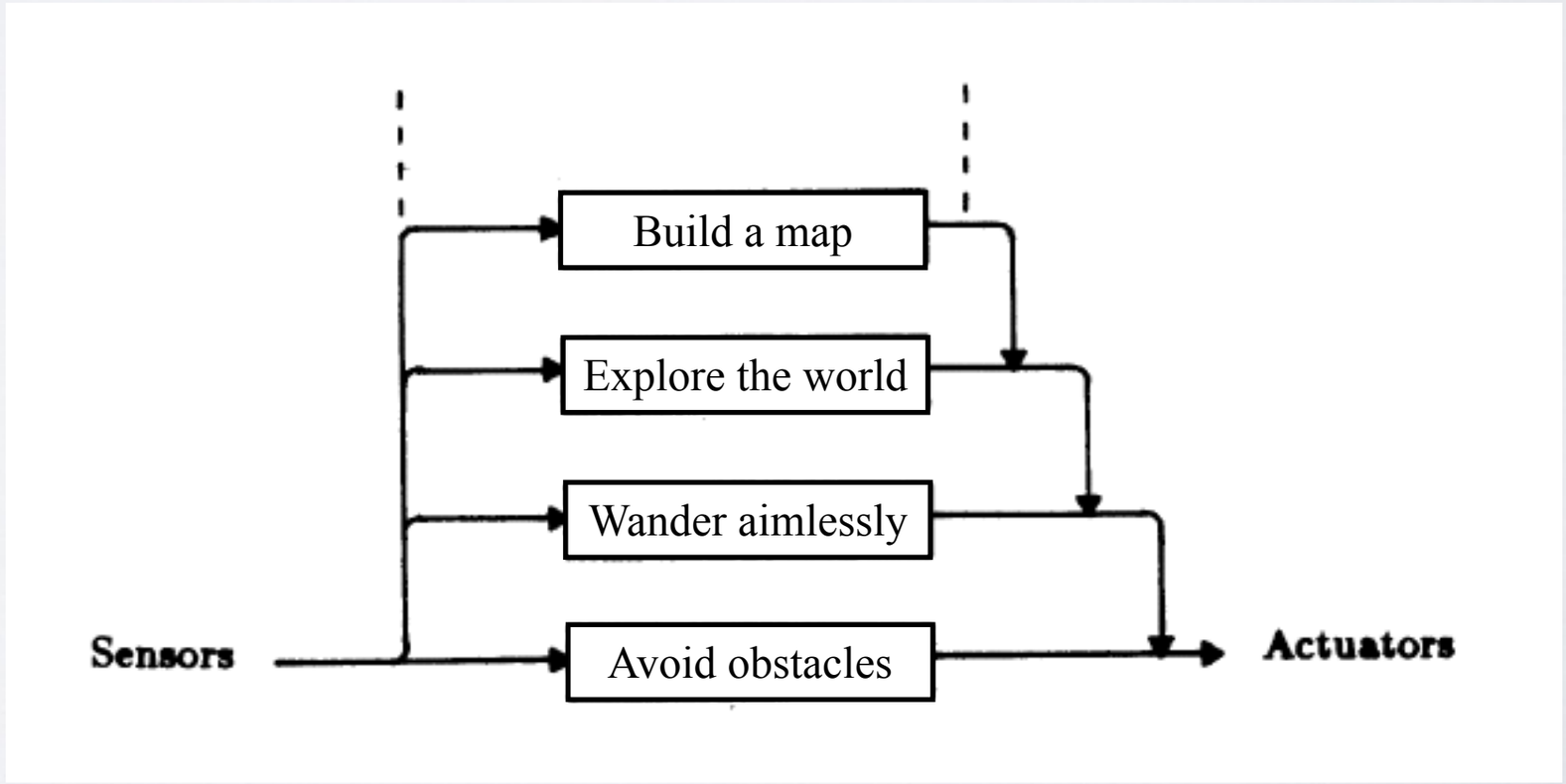
Subsumption:



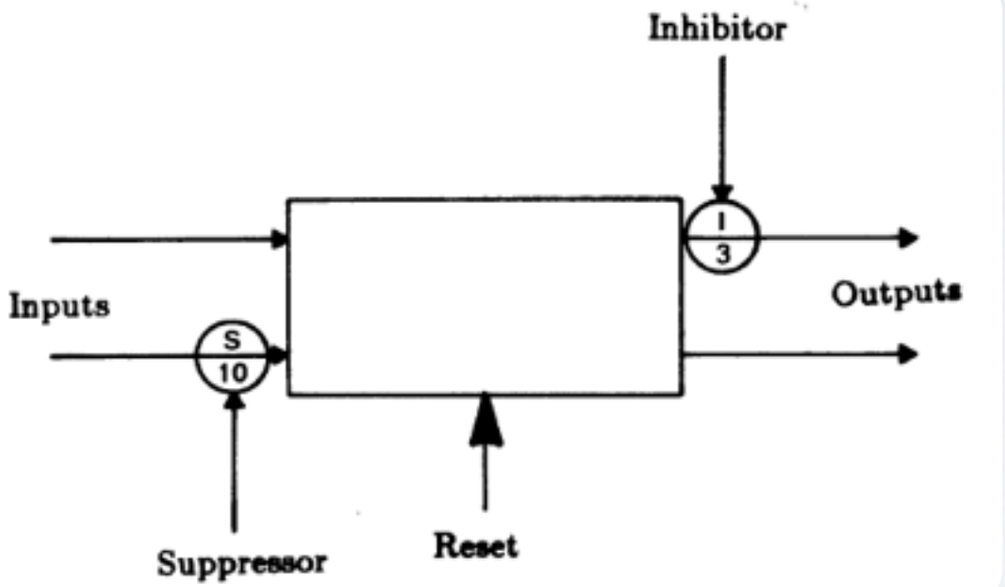
LEVELS OF COMPETENCE

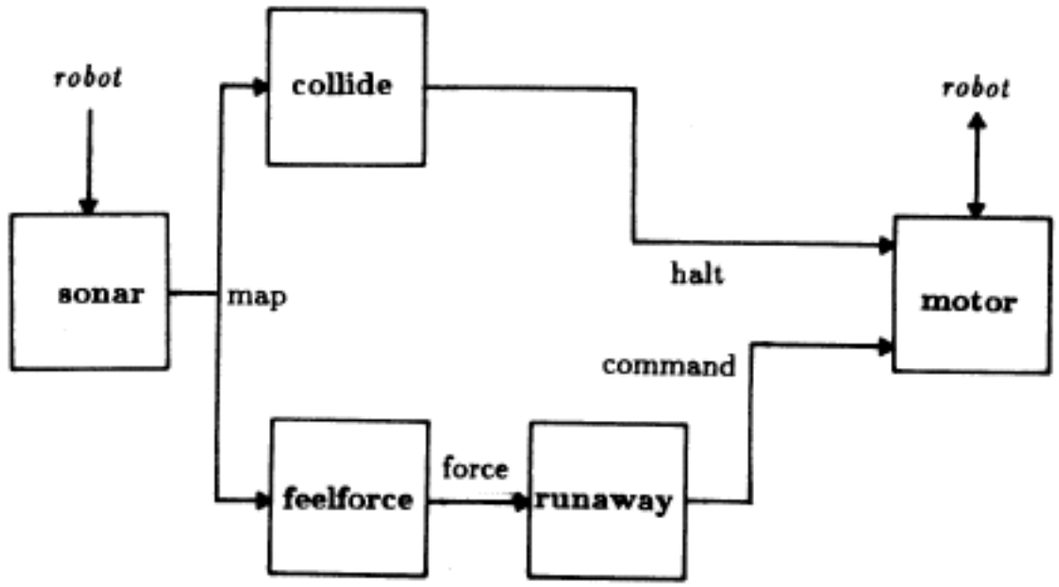
- Assumptions:
  - Simple control, complex behavior
  - Relative, not absolute
  - Real world
- Robust

Subsumption:



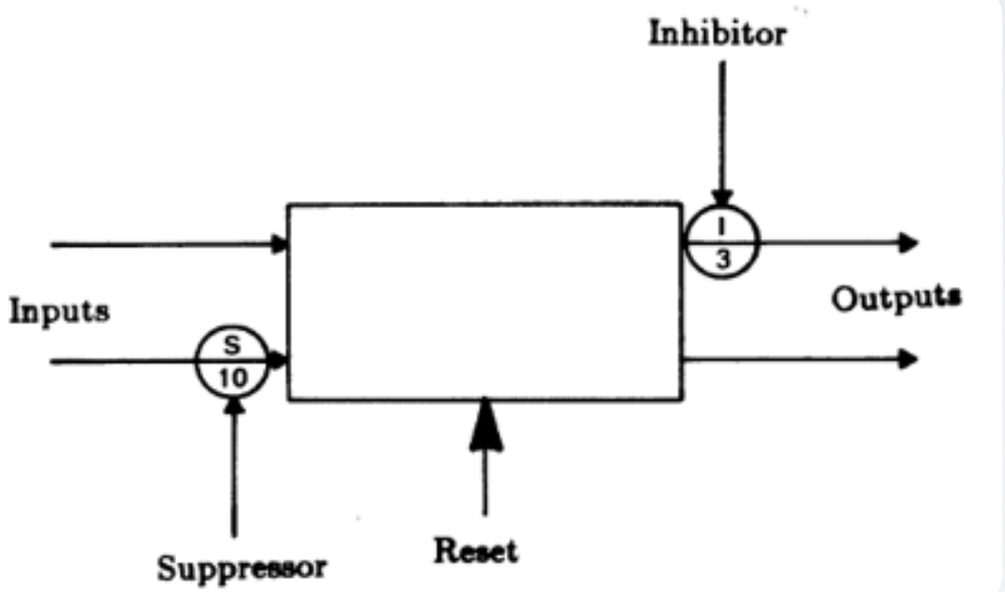
SUBSUMPTION

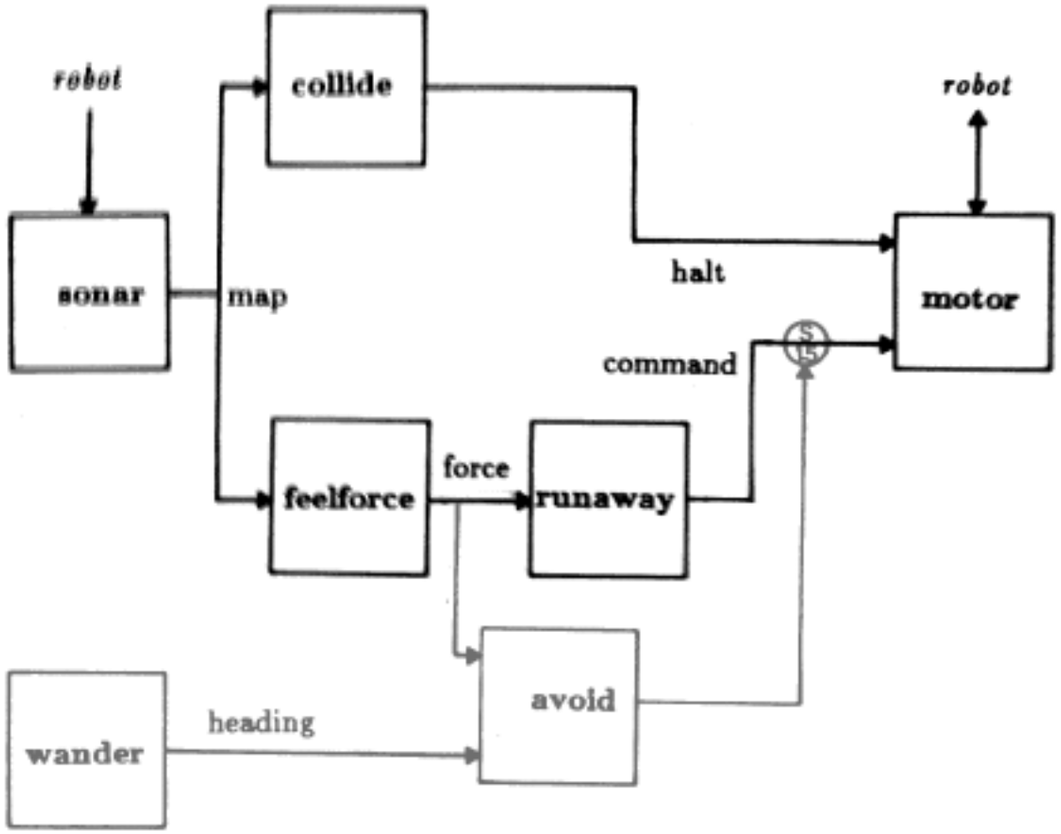




- Inputs can be suppressed
- Outputs can be inhibited

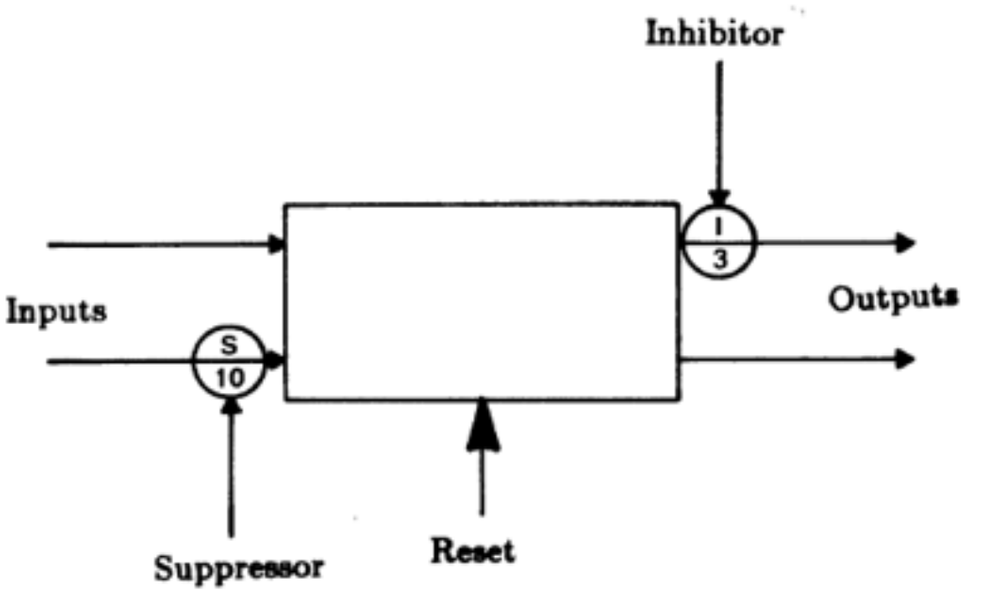
SUBSUMPTION



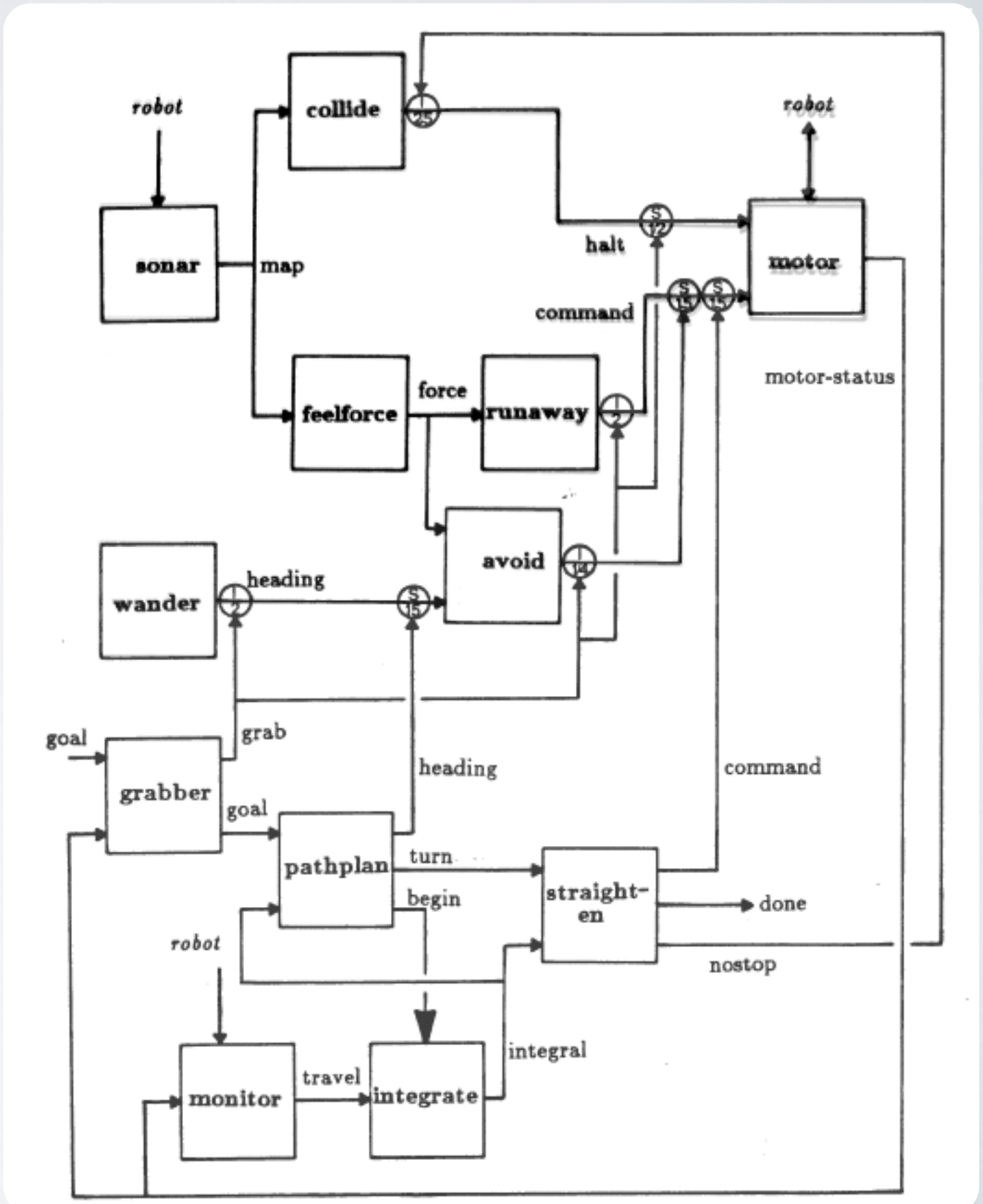


- Inputs can be suppressed
- Outputs can be inhibited

SUBSUMPTION

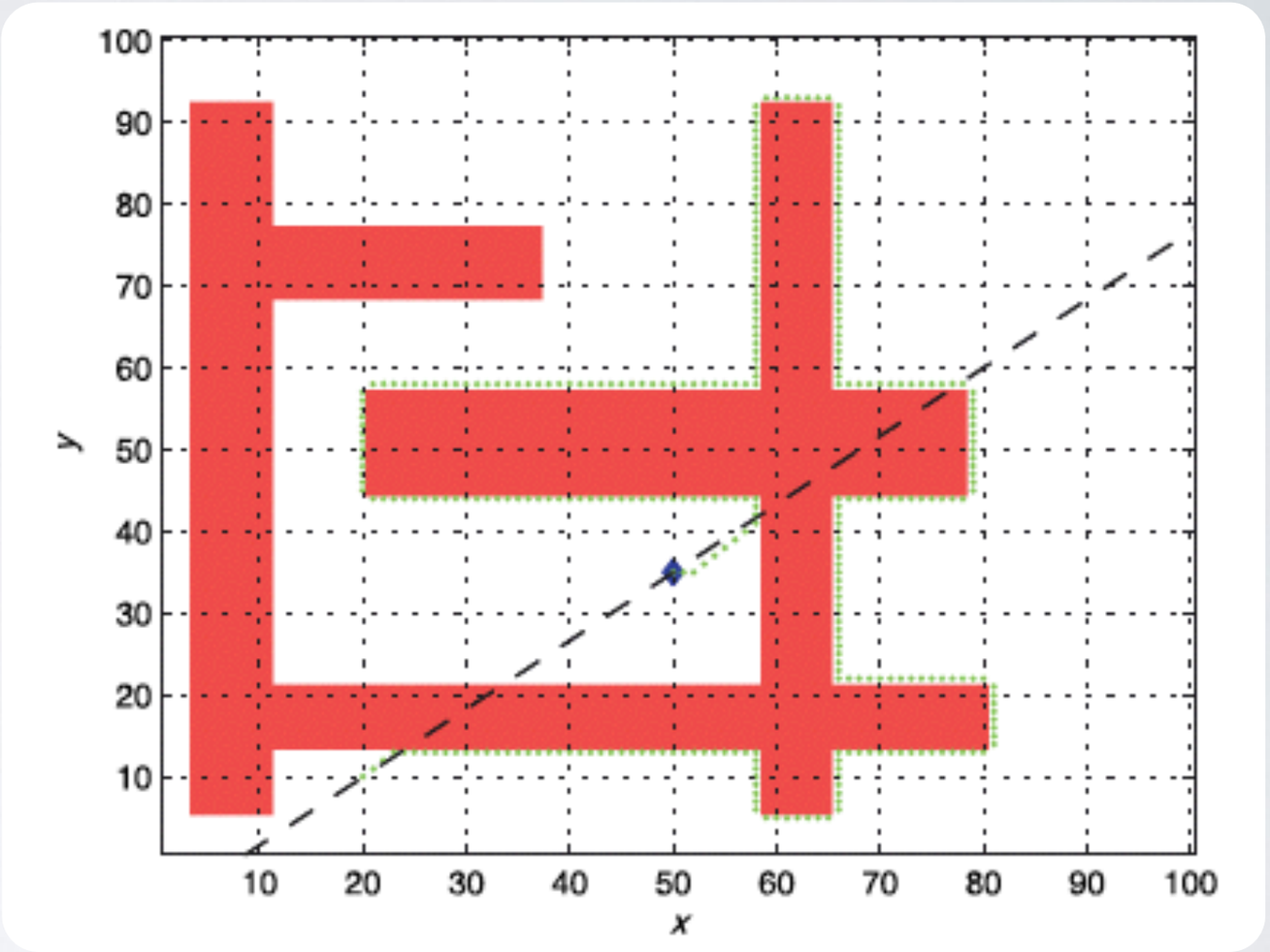


- Inputs can be suppressed
- Outputs can be inhibited



SIMPLE AUTOMATA

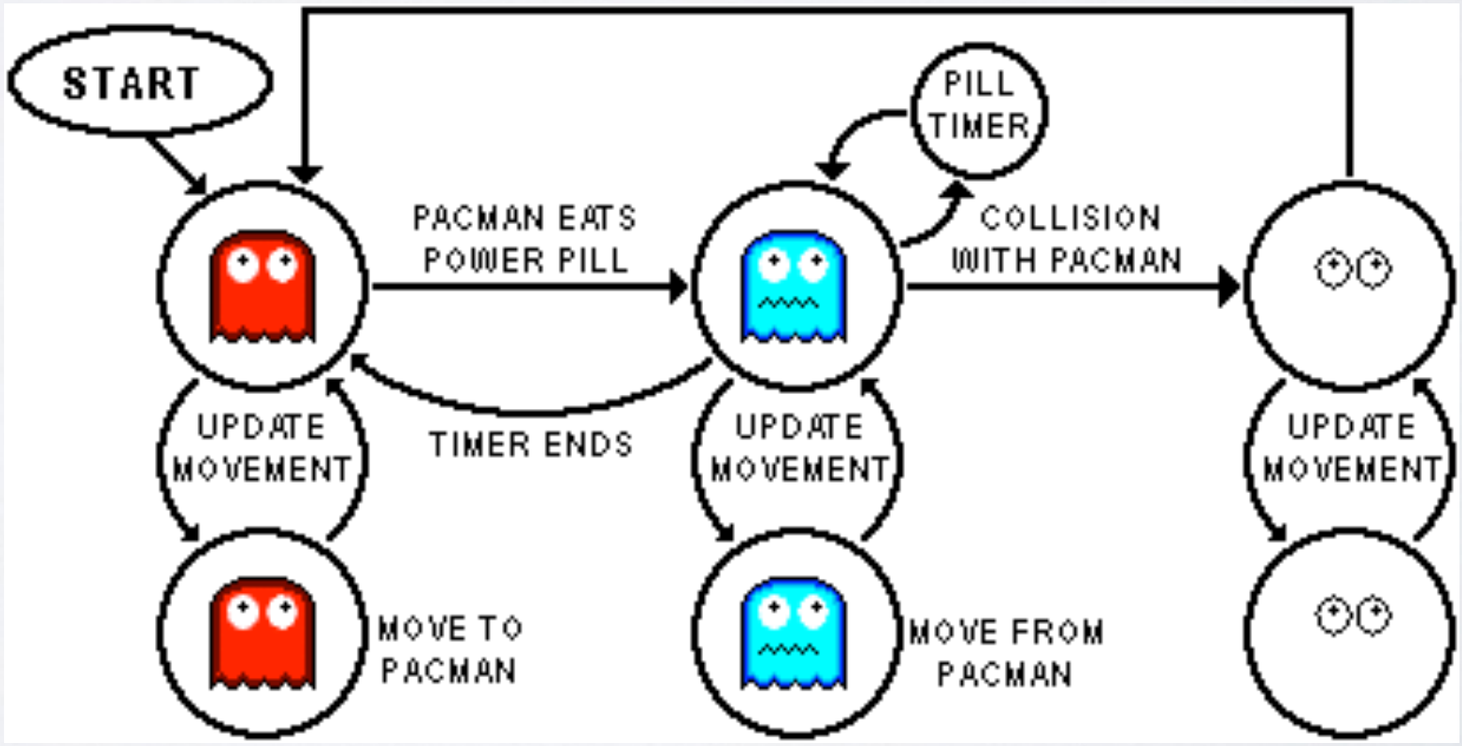
- Bug Algorithms
- Bug2 Demo:
  - straight line
  - go around obstacles CCW
- Suboptimal !




```
load map1
bug = Bug2(map);
bug.goal = [50; 35];
bug.path([20; 10]);
```

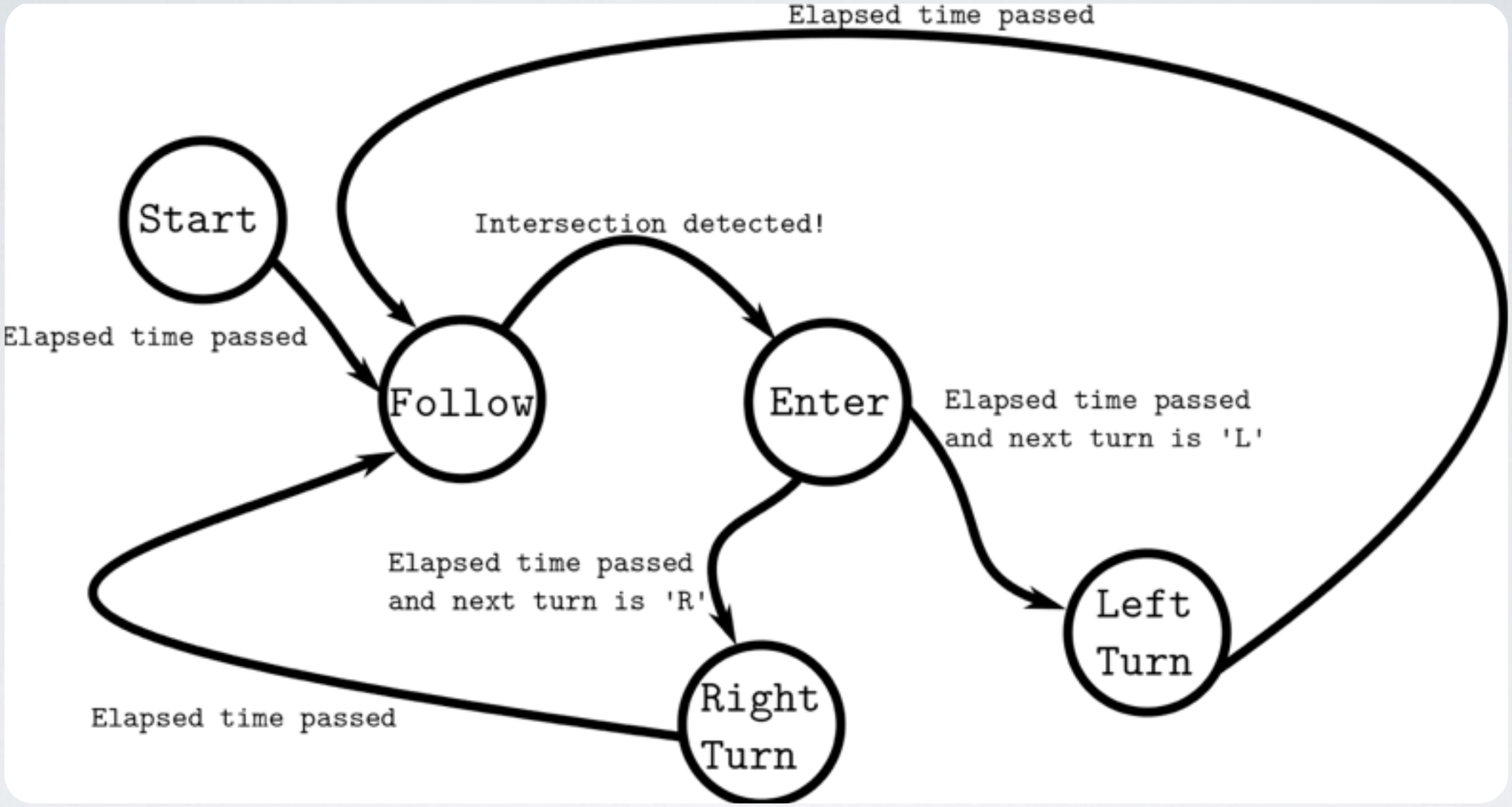
FINITE STATE MACHINES

- States
- Transitions
  - Events



http://oddwiring.com/archive/websites/mndev/MSB/GD100/fsm.htm

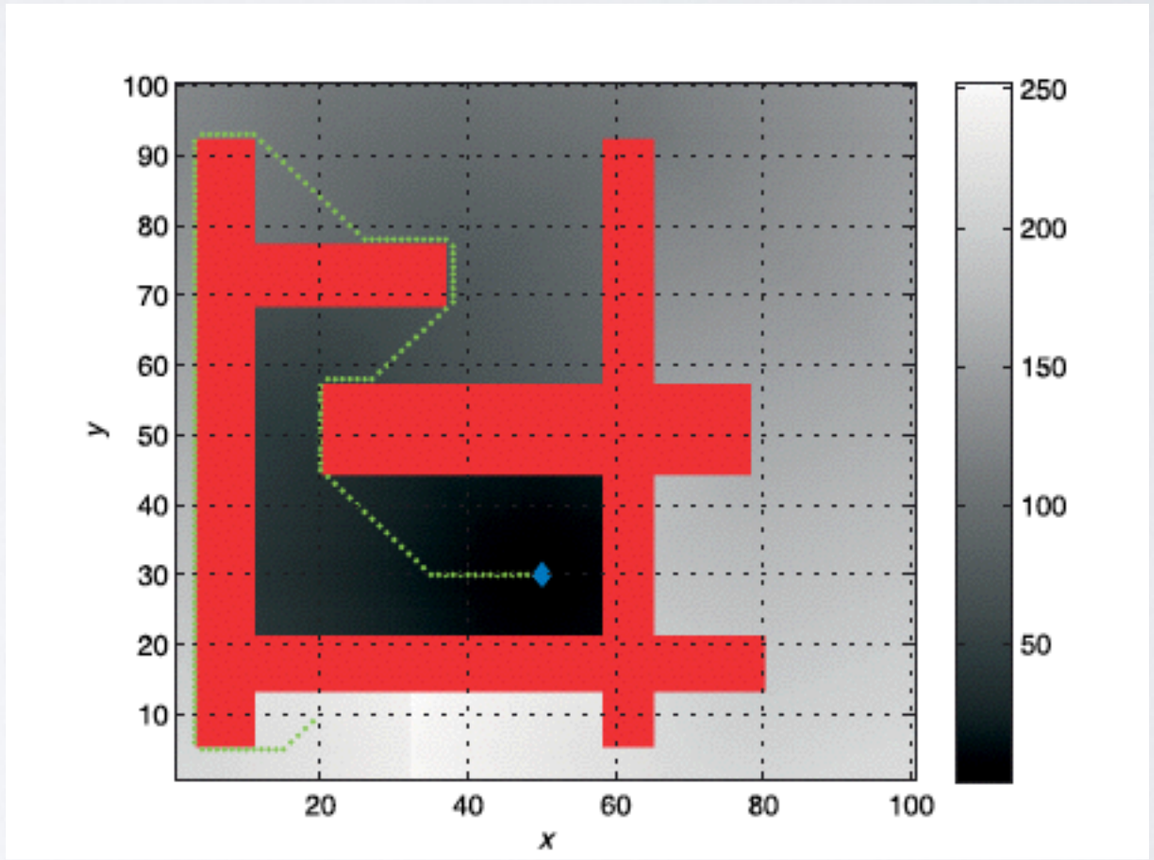
MAZE RUNNER



- Andrew Explains

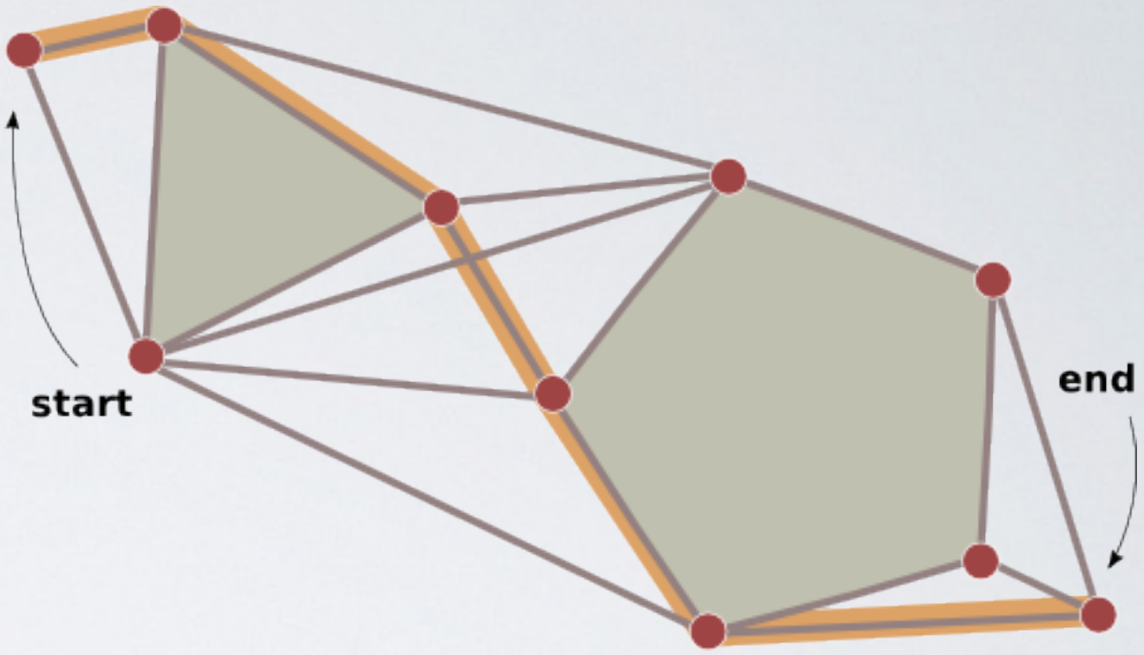
MAP-BASED PLANNING

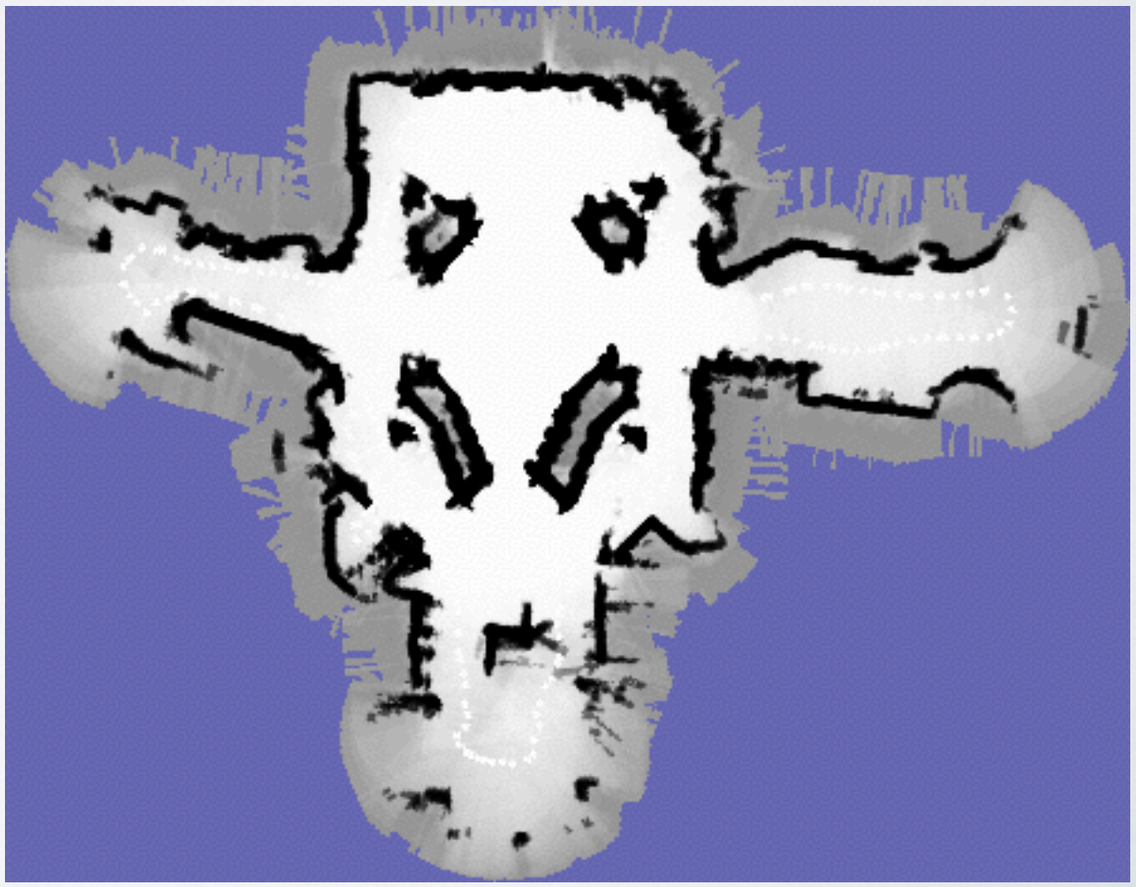
- Maps:
  - Polygons
  - Occupancy Grid
- Shortest Path
  - Dijkstra
  - Distance Transform



MAP REPRESENTATIONS

- Polygons
  - vertices edges
    - Very compact
- Occupancy Grid
  - Memory ~ area
  - Quite doable nowadays





Navigation superclass. The examples in this chapter are all based on classes derived from the `Navigation` class which is designed for 2D grid-based navigation. Each example consists of essentially the following pattern. Firstly we create an instance of an object derived from the `Navigation` class by calling the class constructor.

```
>> nav = MyNavClass(world)
```

which is passed the occupancy grid. Then a plan to reach the goal is computed

```
>> nav.plan(goal)
```

The plan can be visualized by

```
>> nav.plot()
```

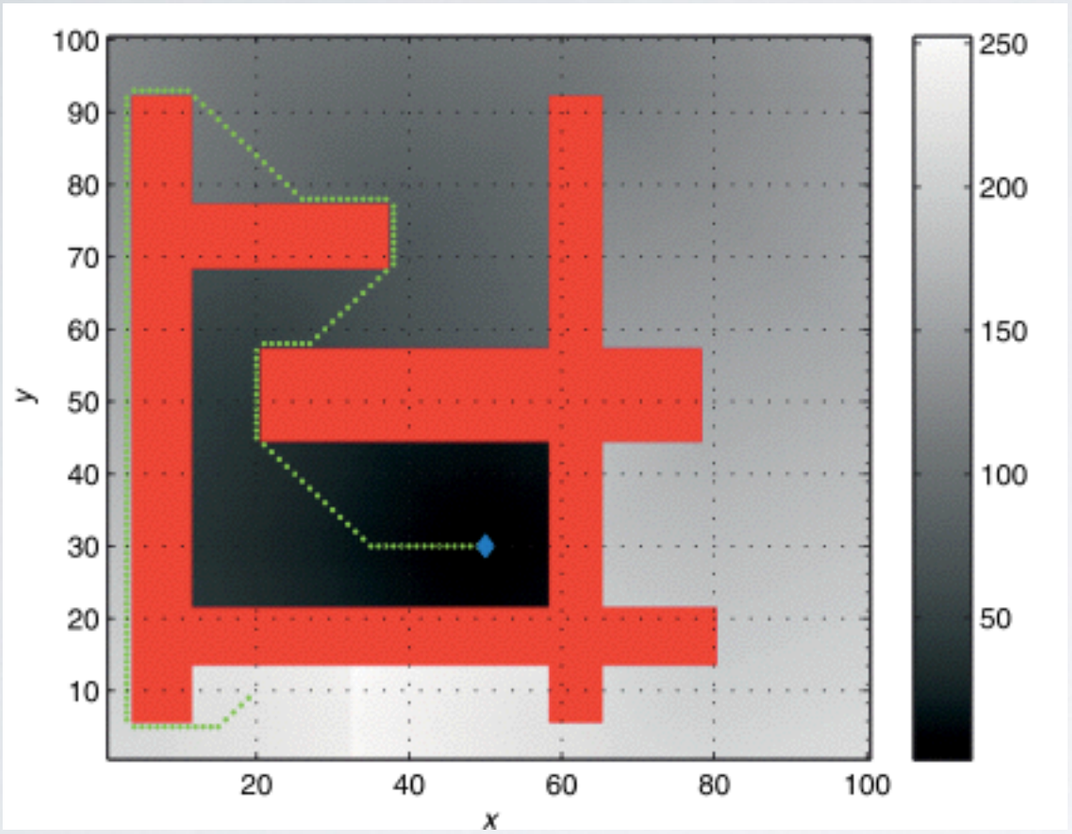
and a path from an initial position to the goal is computed by

```
>> p = nav.path(start)
>> p = nav.path()
```

where `p` is the path, a sequence of points from `start` to `goal`, one row per point, and each row comprises the x- and y-coordinate. If `start` is not specified, as in the second example, the user is prompted to interactively click the start point. If no output argument is provided an animation of the robot's motion is displayed.

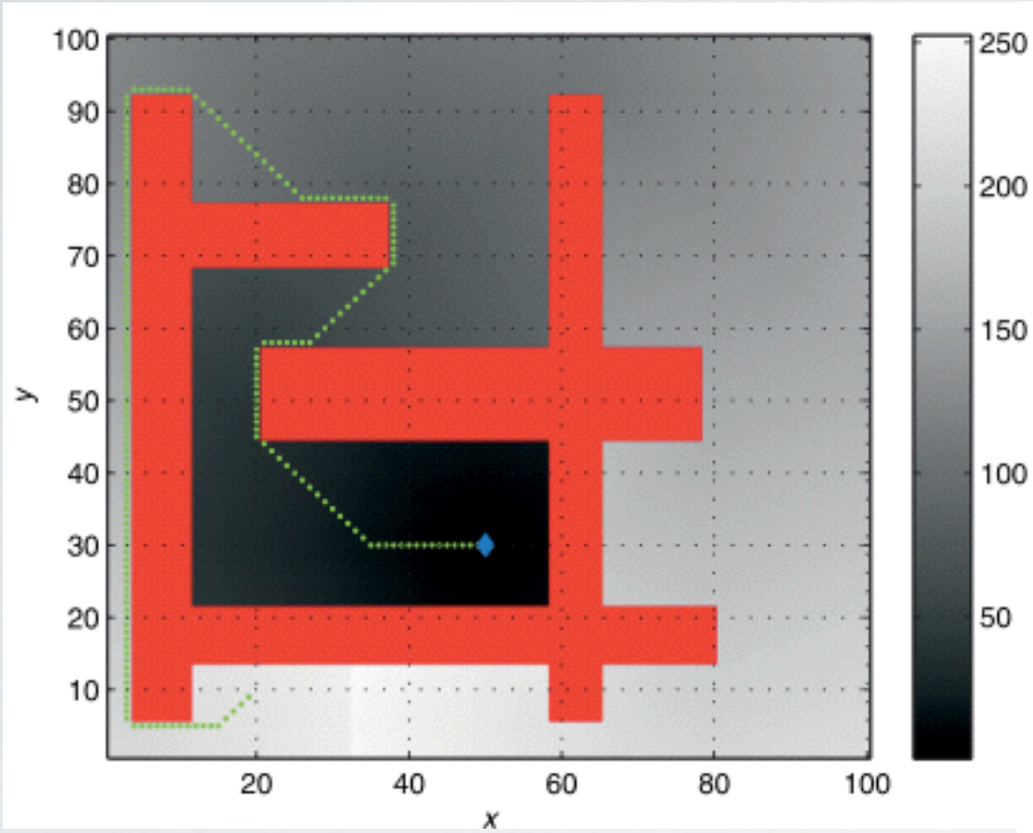
NAVIGATION RVC CLASS

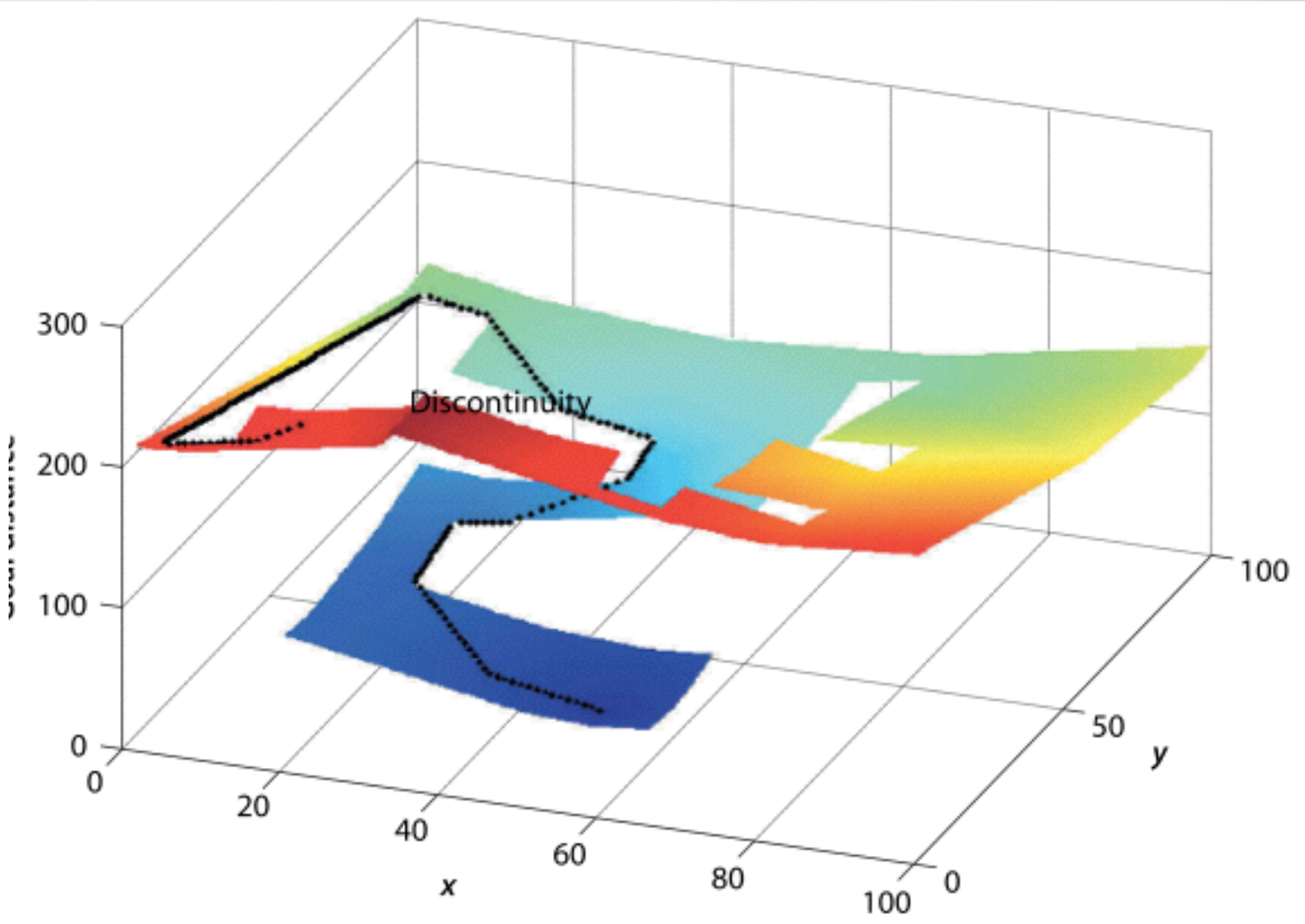
DISTANCE TRANSFORM




```
dx = DXform(map);
dx.plan([50;35]);
dx.plot();
dx.path([20; 10]);
dx.plan(goal, 0.1);
```

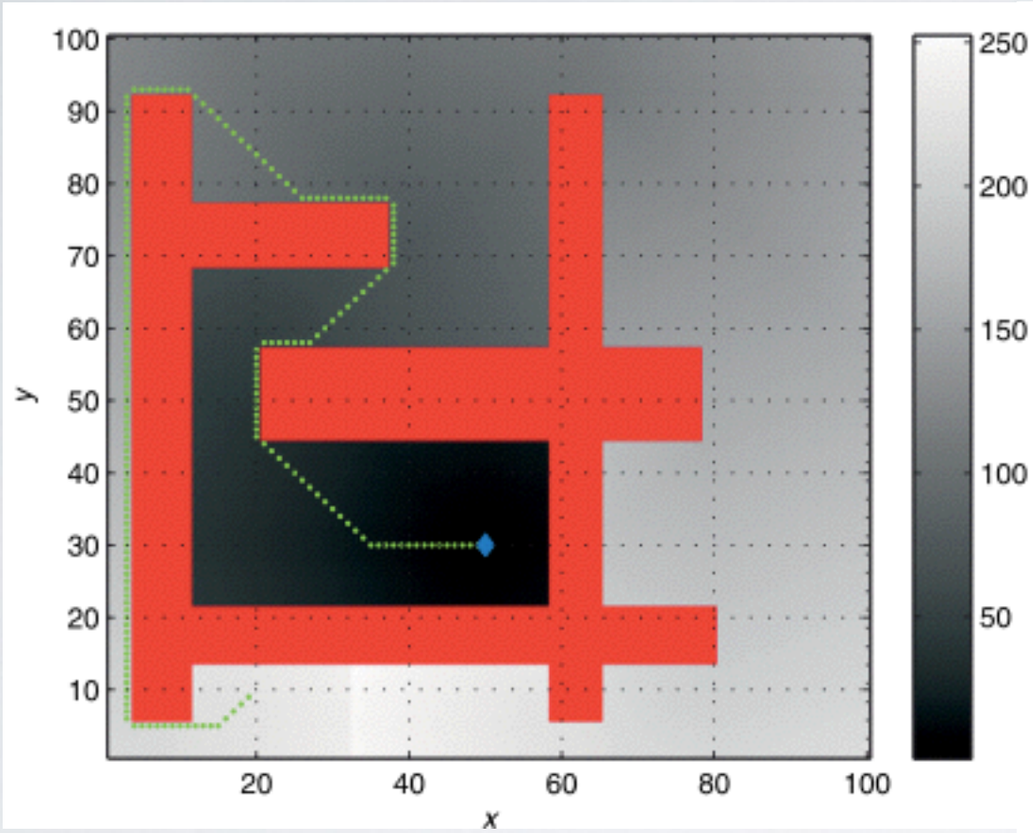
DISTANCE TRANSFORM

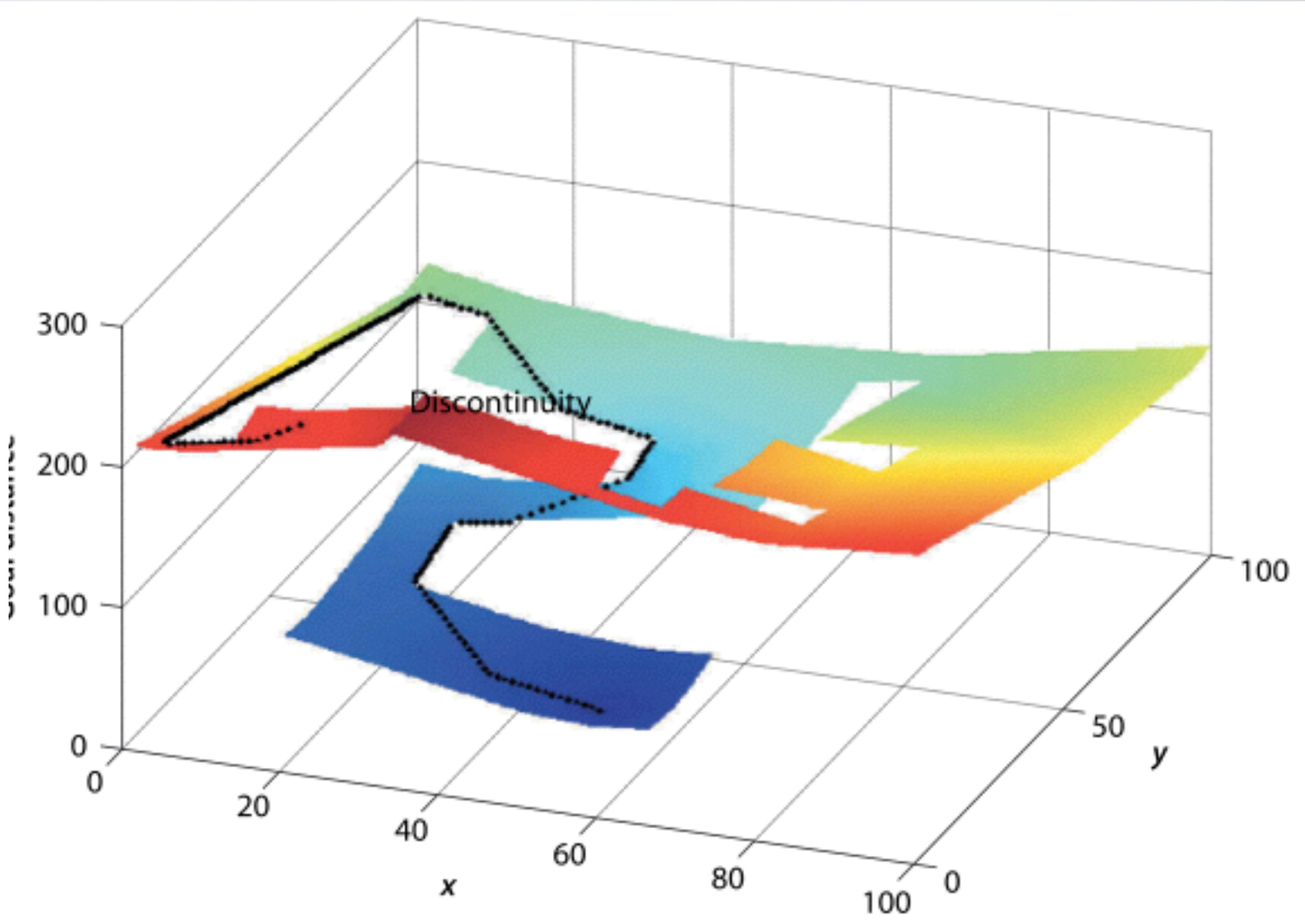





```
dx = DXform(map);
dx.plan([50;35]);
dx.plot();
dx.path([20; 10]);
dx.plan(goal, 0.1);
```

DISTANCE TRANSFORM






```
dx = DXform(map);
dx.plan([50;35]);
dx.plot();
dx.path([20; 10]);
dx.plan(goal, 0.1);
```

- Rolling downhill!

Next: excerpts from…

Real-Time Planning in Dynamic and Partially Known Domains

Maxim Likhachev
University of Pennsylvania
maximl@seas.upenn.edu

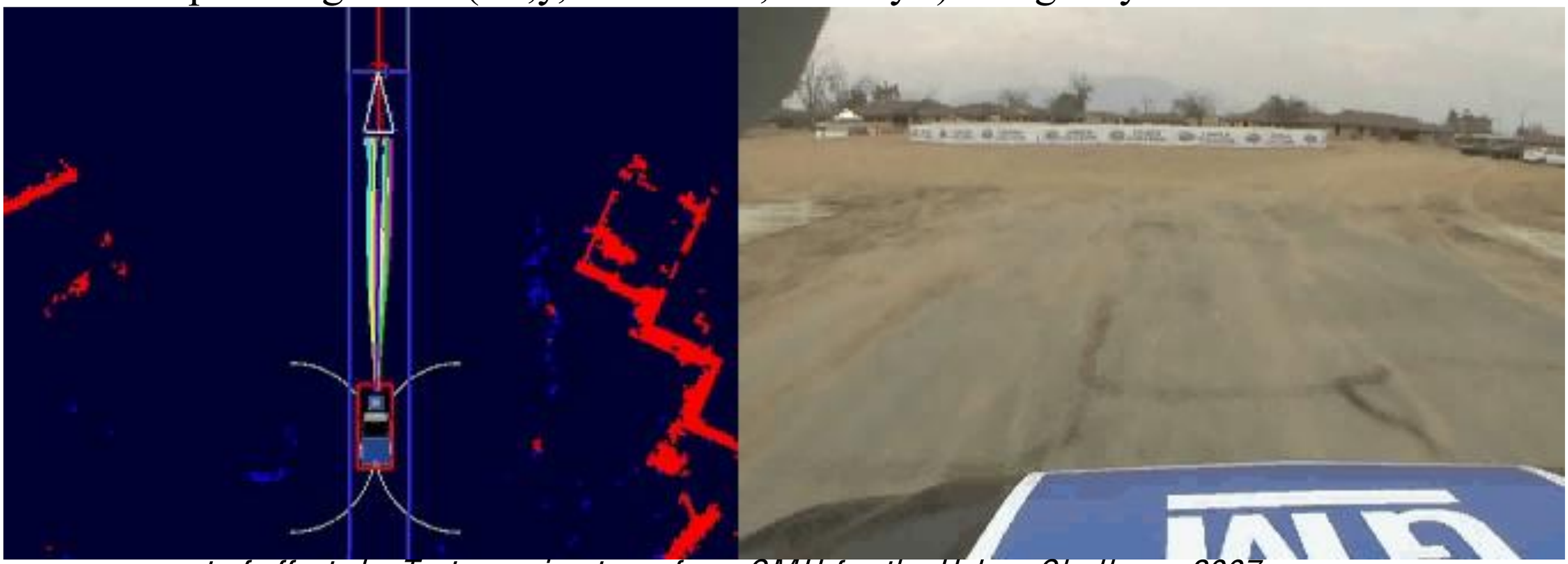
Sven Koenig
University of Southern California
skoenig@usc.edu

Real-time Planning in Dynamic and Partially-known Domains

Maxim

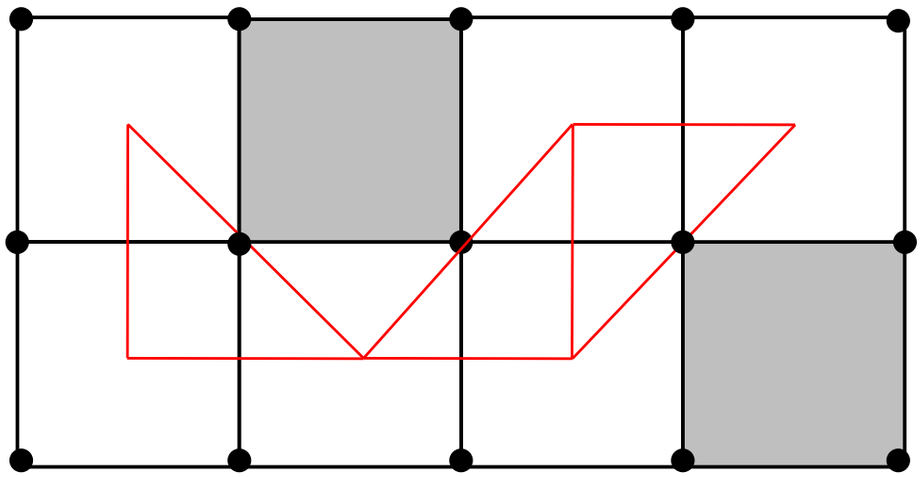
- Challenges
  - complexity/size (high-dim., expensive to compute costs, etc.)
  - severe time constraints (e.g., tens of msecs to few seconds)
  - robustness to uncertainties in execution, sensing, environment

planning in 4D (<x,y,orientation,velocity>) using Anytime D*

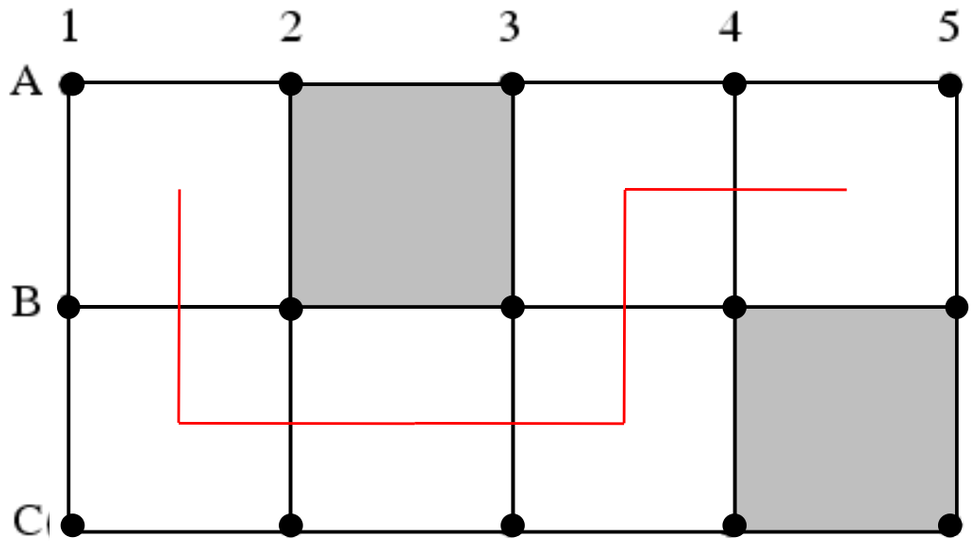


part of efforts by Tartanracing team from CMU for the Urban Challenge 2007 race

Discretizing Configuration Space



8-neighbor grid



4-neighbor grid

A*

$$f(x) = g(x) + h(x)$$

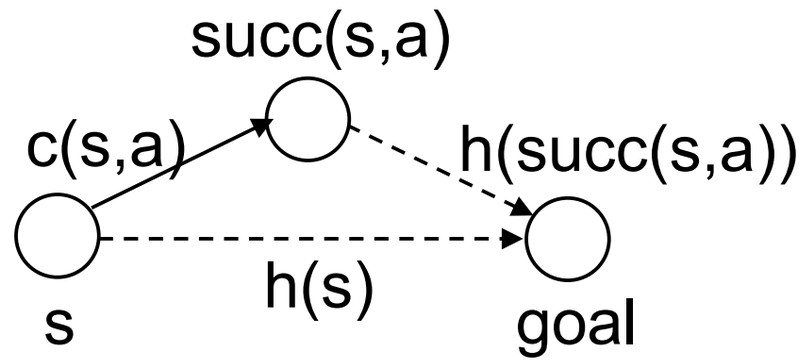
estimated = cost-so-far + heuristic est.

(Forward) A*

1. Create a search tree that contains only the start.
2. Pick a generated but not yet expanded state s with the smallest f-value.
3. If state s is the goal then stop.
4. Expand state s.
5. Go to 2.

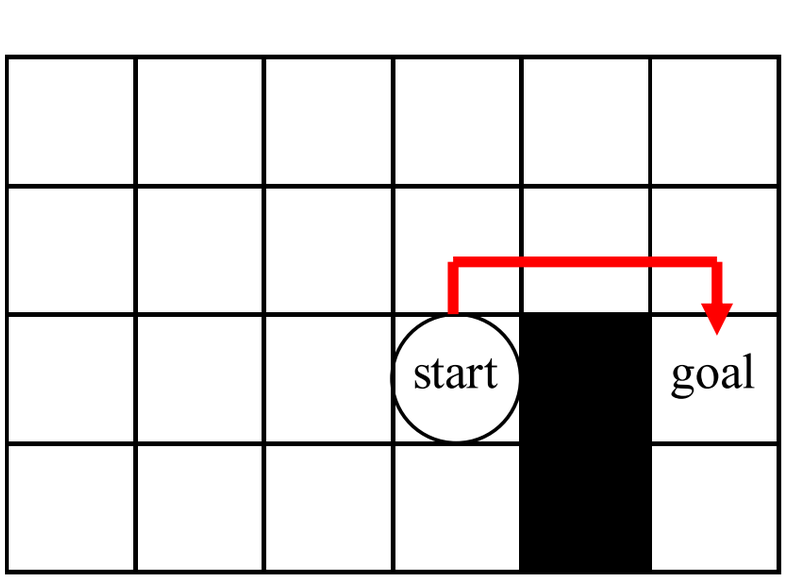
A*

- A* [Hart, Nilsson and Raphael, 1968] uses user-supplied h-values to focus its search.
- The h-values approximate the goal distances.
- We always assume that the h-values are consistent!
- The h-values h(s) are consistent iff they satisfy the triangle inequality:
  h(s) = 0 if s is the goal and
  h(s) ≤ c(s,a) + h(succ(s,a)) otherwise.



A*

- Search problem with uniform cost



4-neighbor grid

A*

- Possible consistent h-values

7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0
6	5	4	3	2	1

Manhattan Distance

5	4	3	2	2	2
5	4	3	2	1	1
5	4	3	2	1	0
5	4	3	2	1	1

Octile Distance

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Zero h-values

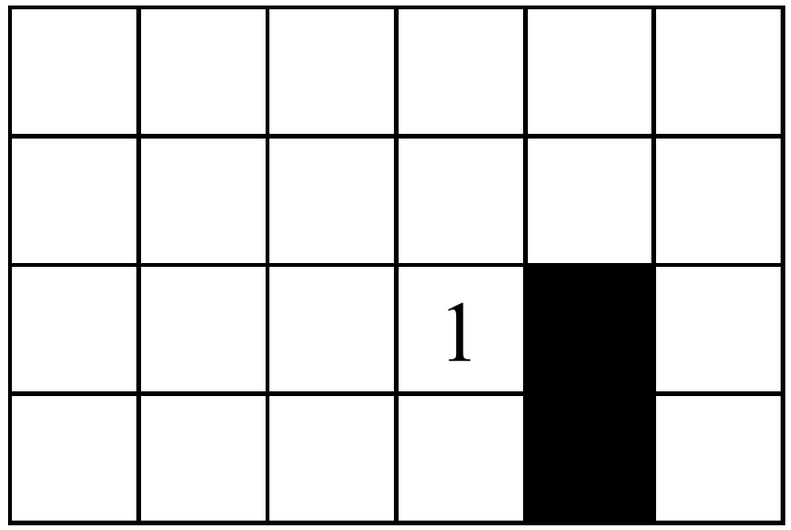
more informed (dominating)

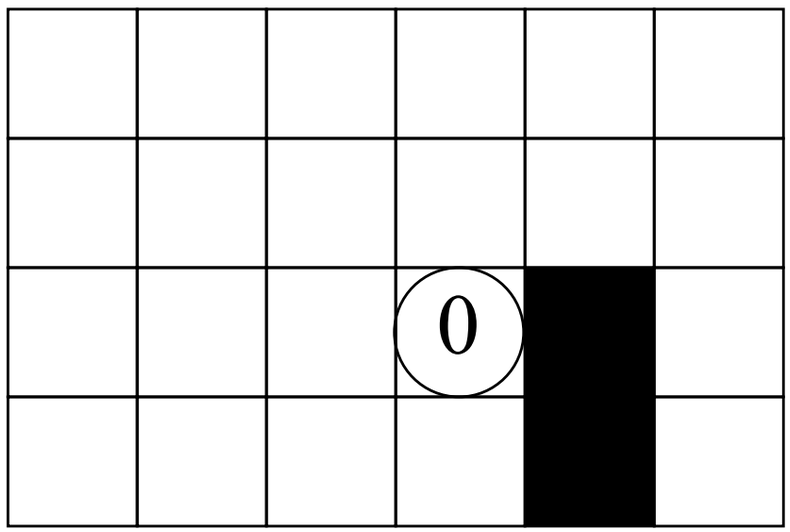
4-neighbor grid

A*

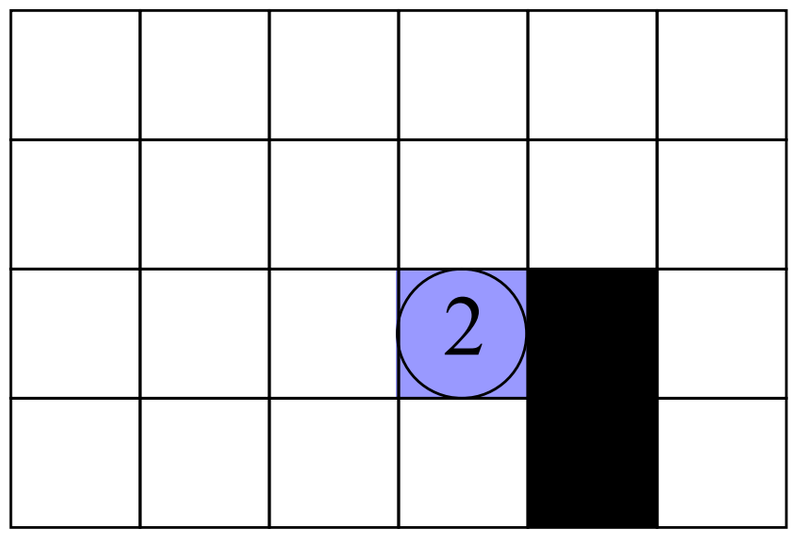
- First iteration of A*

order of expansions





7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0
6	5	4	3	2	1



g-values + h-values = f-values

cost of the shortest path in the search tree from the start to the given state

generated but not expanded state (OPEN list)

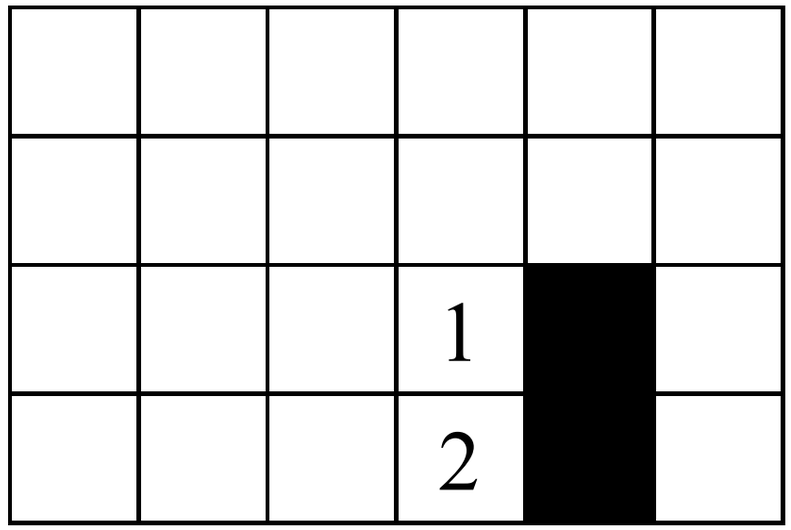
expanded state (CLOSED list)

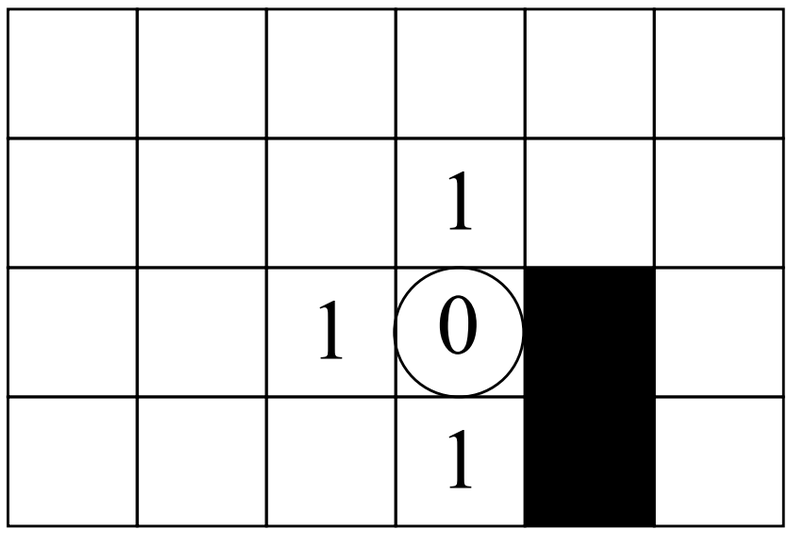
4-neighbor grid

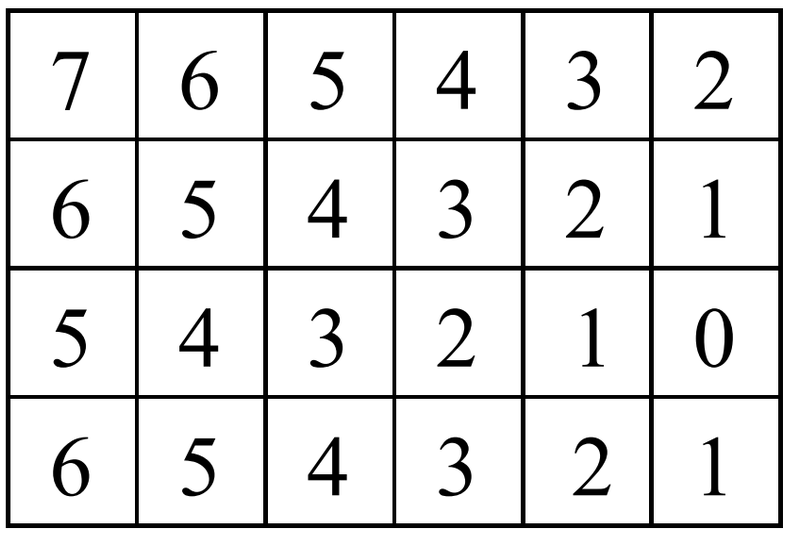
A*

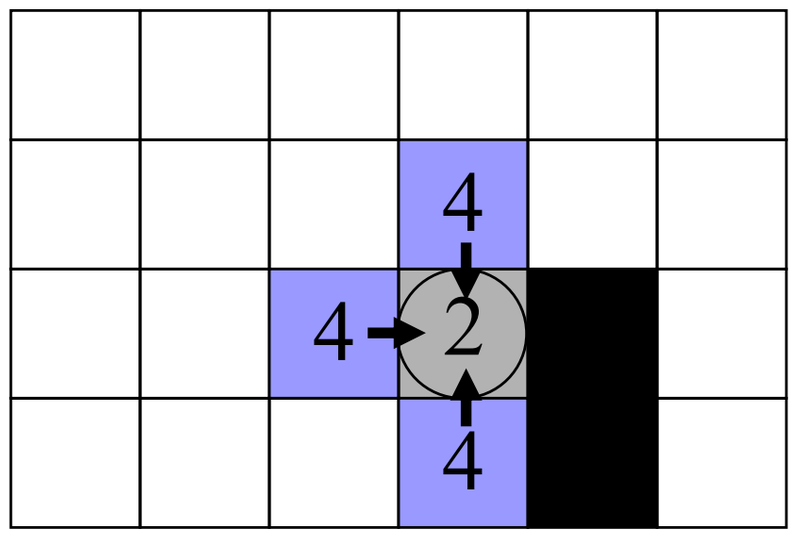
- Second iteration of A*

order of expansions









g-values + h-values = f-values

cost of the shortest path in the search tree from the start to the given state

generated but not expanded state (OPEN list)

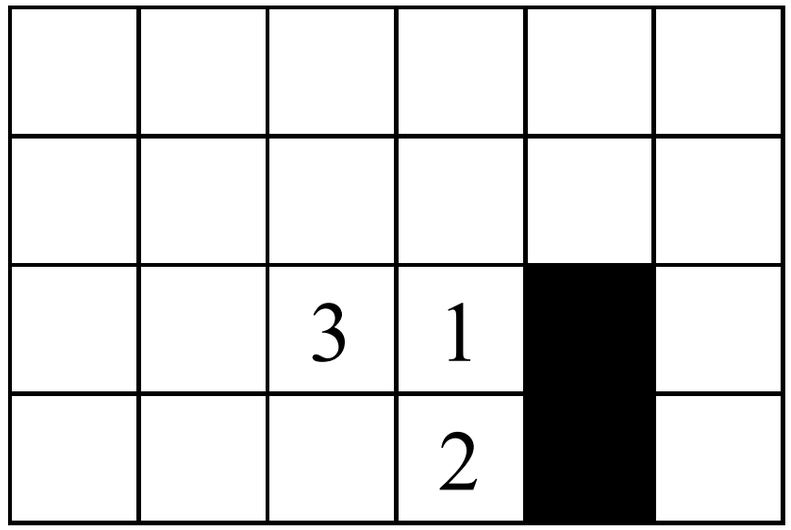
expanded state (CLOSED list)

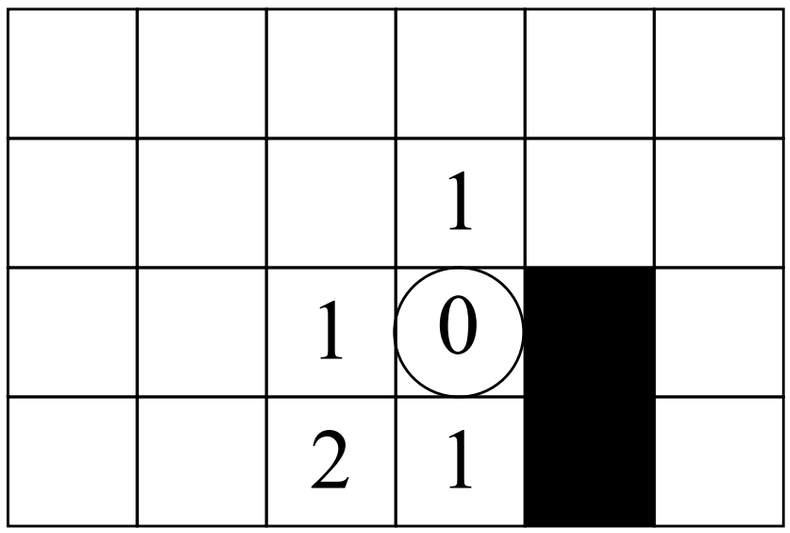
4-neighbor grid

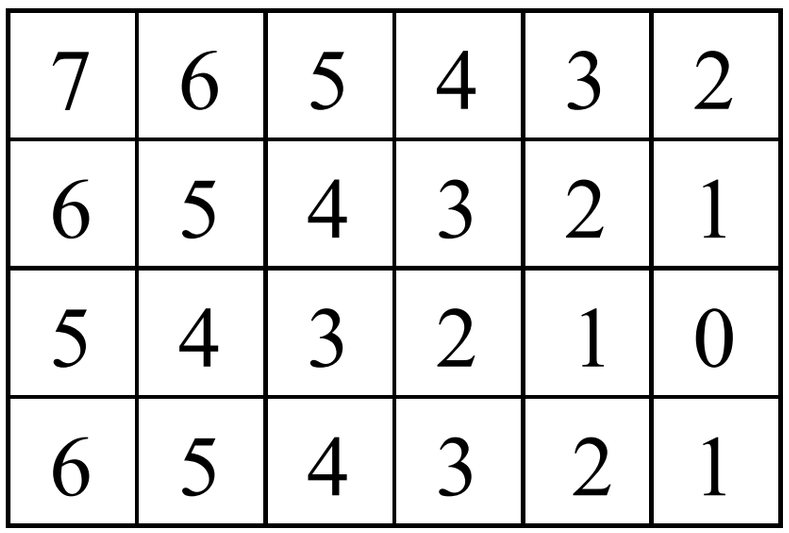
A*

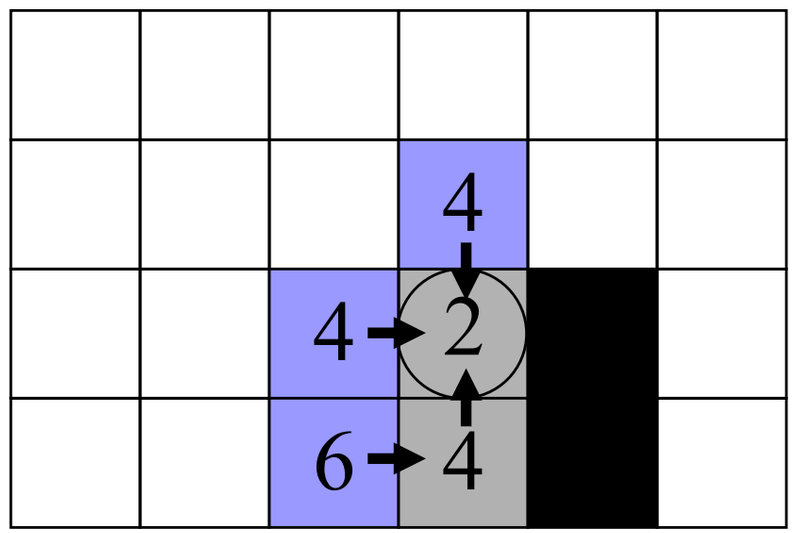
- Third iteration of A*

order of expansions









g-values + h-values = f-values

cost of the shortest path in the search tree from the start to the given state

generated but not expanded state (OPEN list)

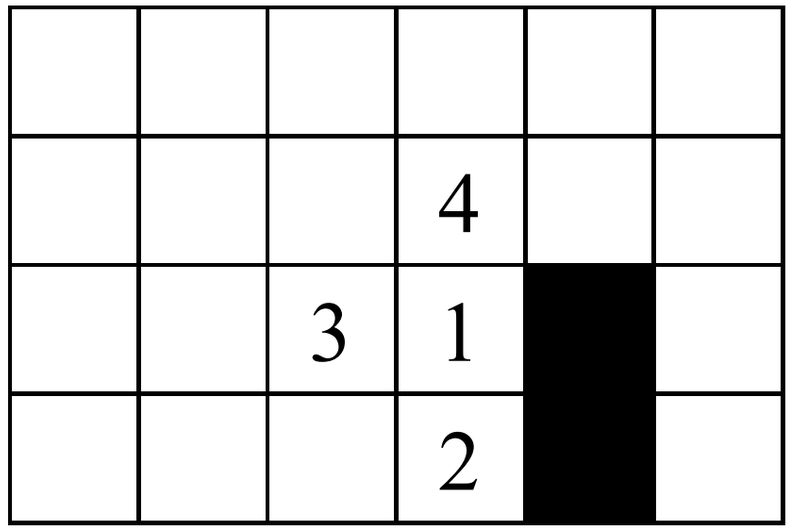
expanded state (CLOSED list)

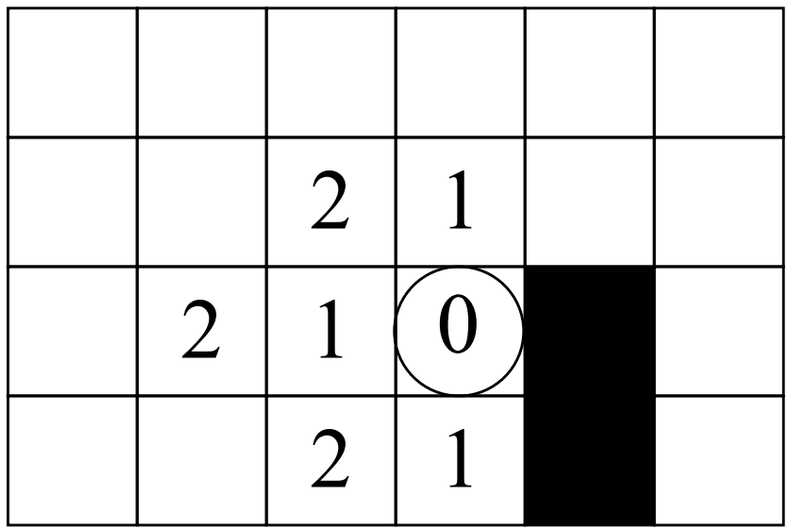
4-neighbor grid

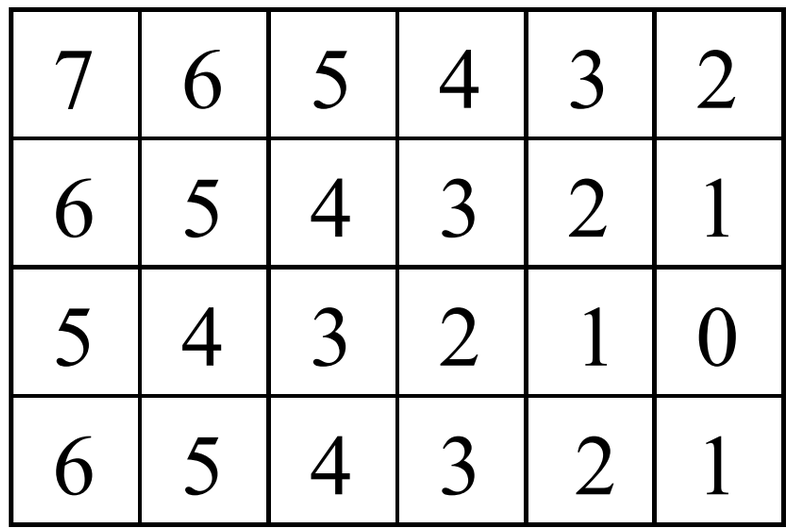
A*

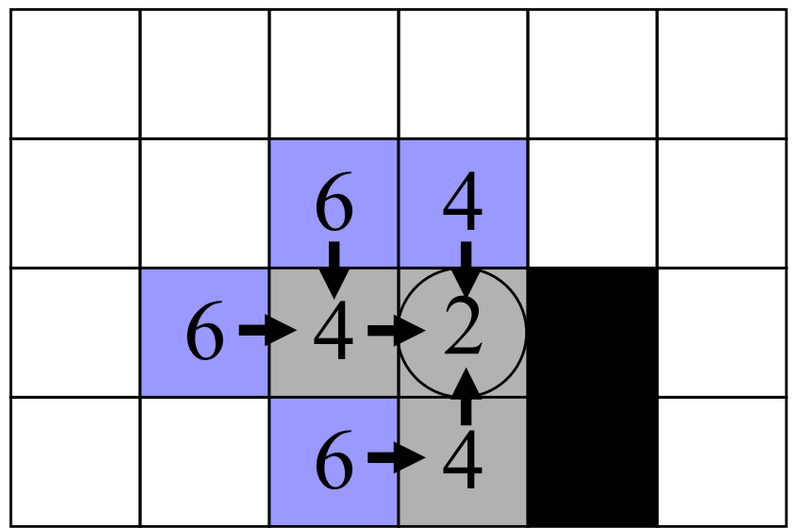
- Fourth iteration of A*

order of expansions









g-values + h-values = f-values

cost of the shortest path in the search tree from the start to the given state

generated but not expanded state (OPEN list)

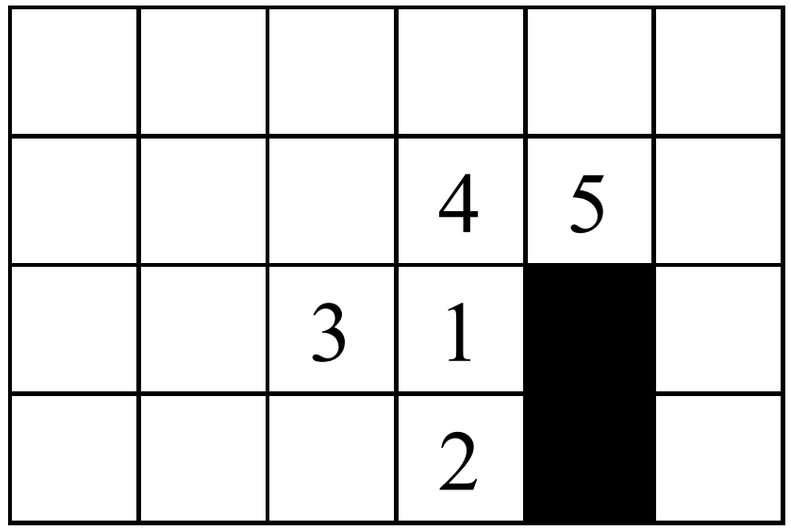
expanded state (CLOSED list)

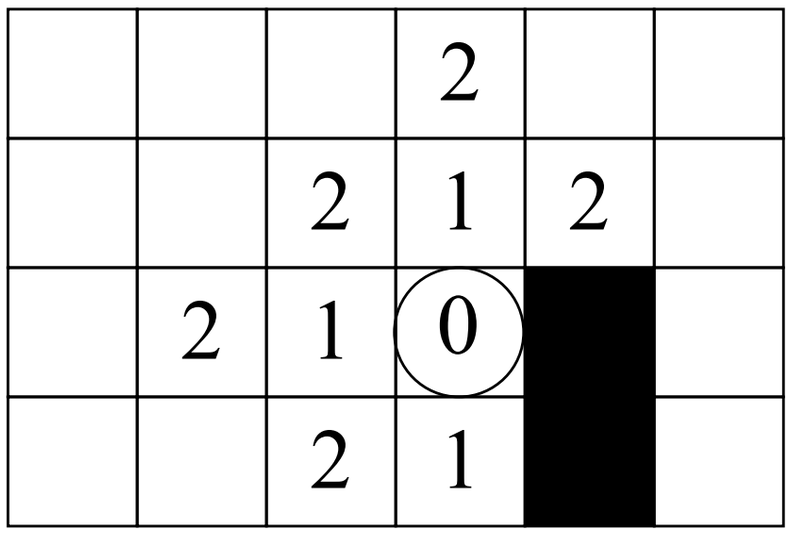
4-neighbor grid

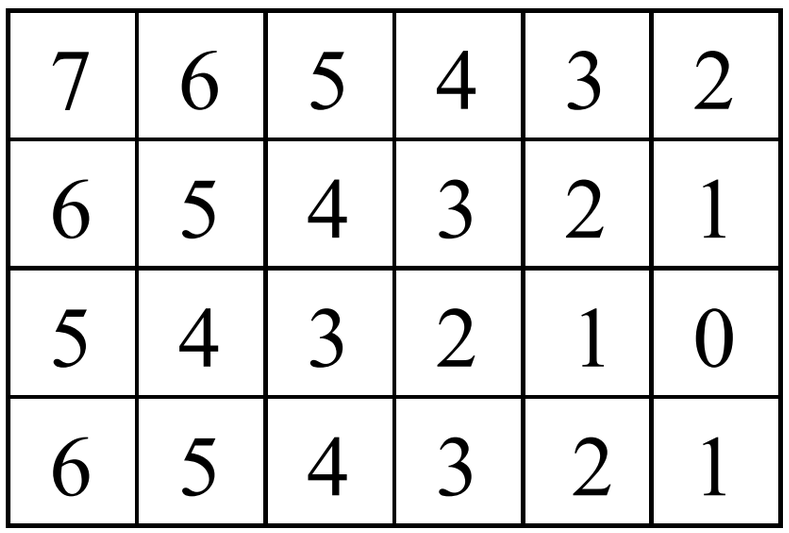
A*

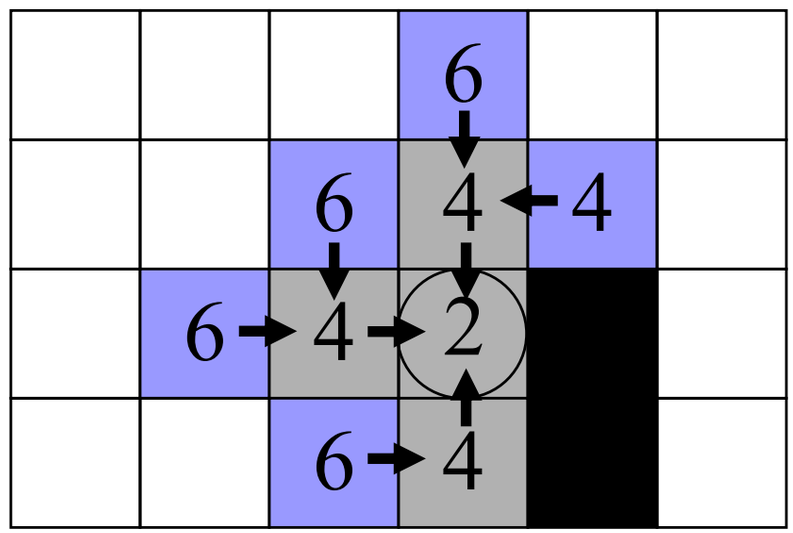
- Fifth iteration of A*

order of expansions









g-values + h-values = f-values

cost of the shortest path in the search tree from the start to the given state

generated but not expanded state (OPEN list)

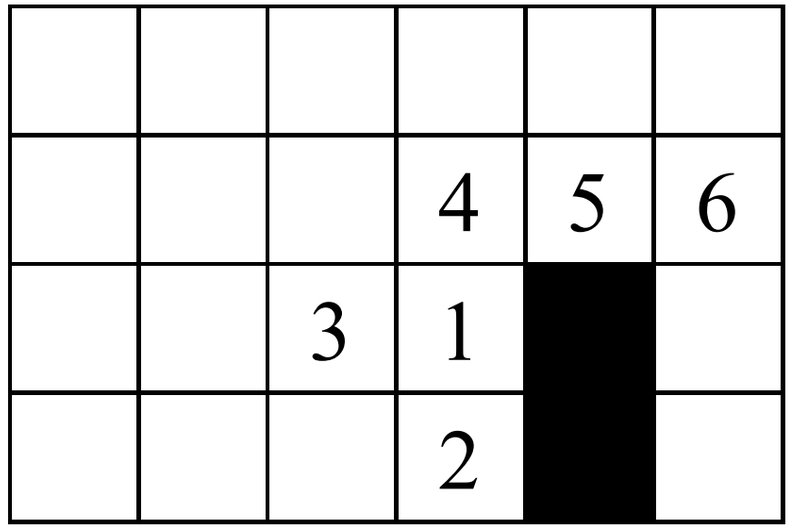
expanded state (CLOSED list)

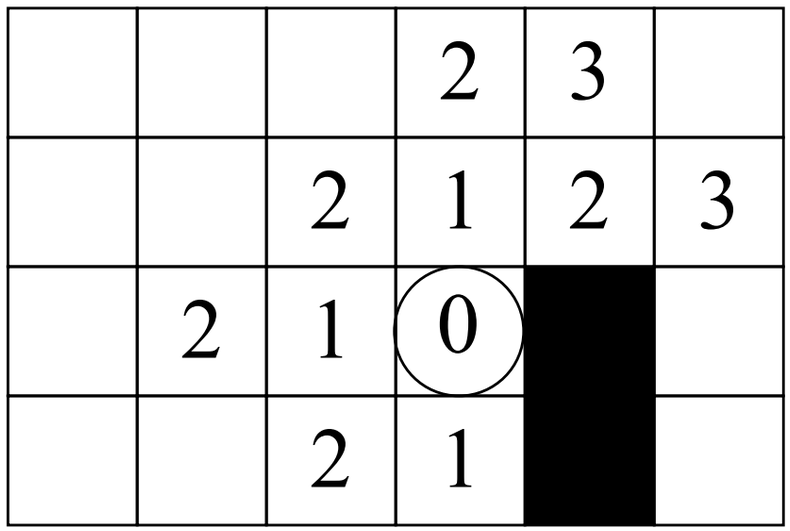
4-neighbor grid

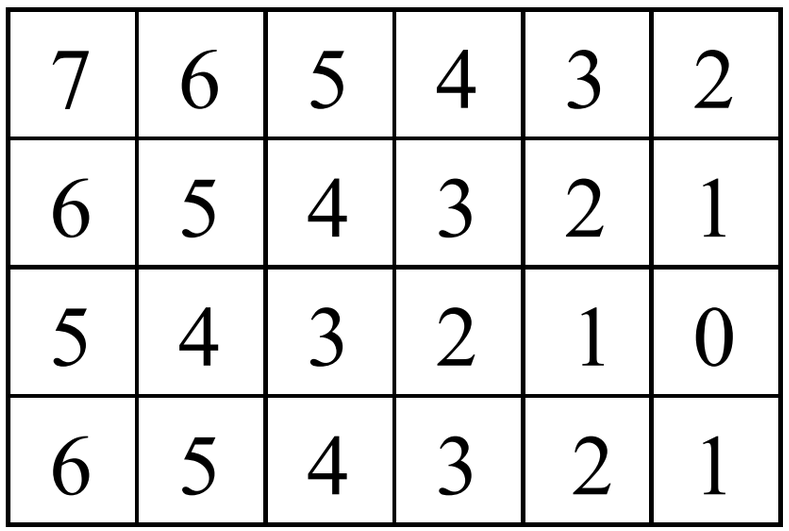
A*

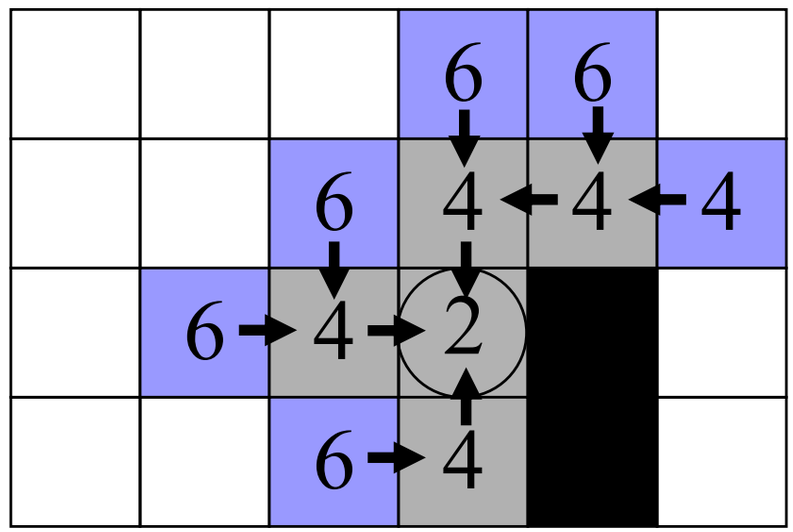
- Sixth iteration of A*

order of expansions









g-values + h-values = f-values

cost of the shortest path in the search tree from the start to the given state

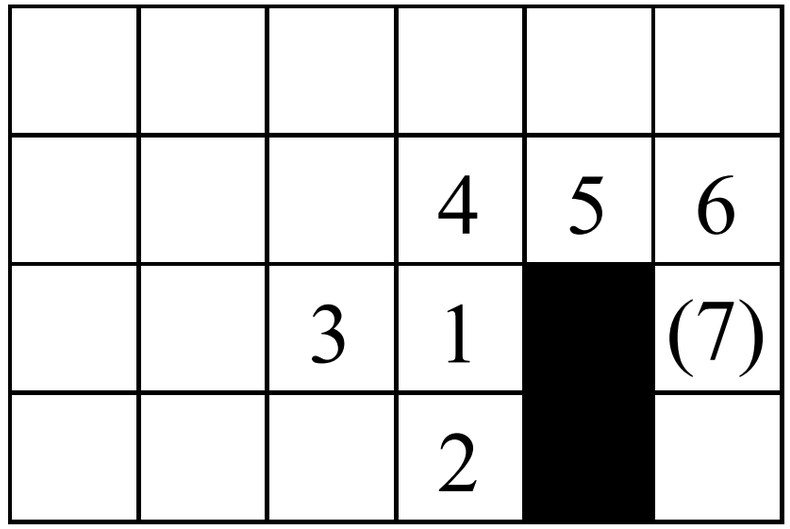
generated but not expanded state (OPEN list)

expanded state (CLOSED list)

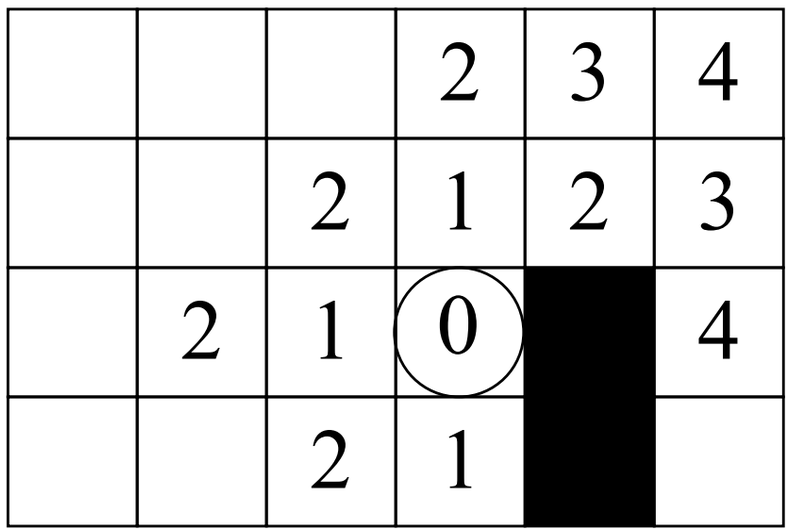
4-neighbor grid

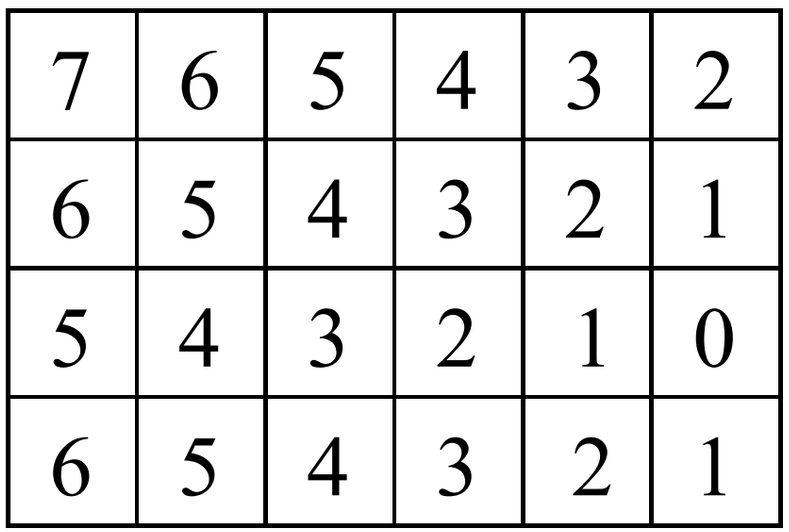
A*

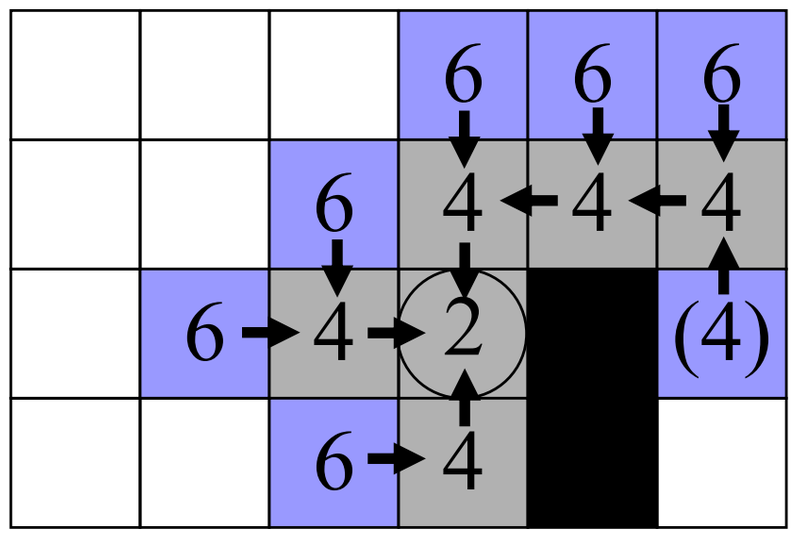
order of expansions



- Seventh and last iteration of A*







g-values + h-values = f-values

cost of the shortest path in the search tree from the start to the given state

generated but not expanded state (OPEN list)

expanded state (CLOSED list)

4-neighbor grid

Sven

A*

Uniform-cost search

Breadth-first search

7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0
6	5	4	3	2	1

5	4	3	2	2	2
5	4	3	2	1	1
5	4	3	2	1	0
5	4	3	2	1	1

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Manhattan Distance

			4	5	6
		3	1		(7)
			2		

Octile Distance

			6		
			3	4	7
		5	1		(8)
			2		

Zero h-values

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

more informed (dominating)

4-neighbor grid

Incremental Heuristic Search

search task 1	slightly different search task 2	slightly different search task 3

Incremental Heuristic Search

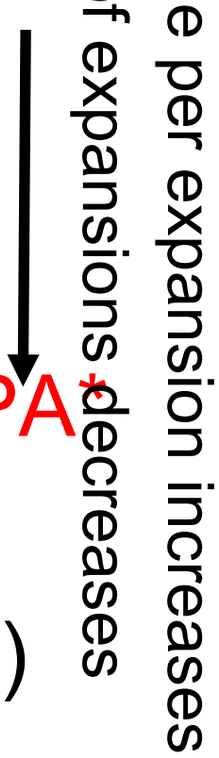
- Incremental heuristic search speeds up A* searches for a sequence of similar search problems by exploiting experience with earlier search problems in the sequence. It finds shortest paths.
- In the worst case, incremental heuristic search cannot be more efficient than A* searches from scratch

  [Nebel and Koehler 1995].

Sven

Incremental Heuristic Search

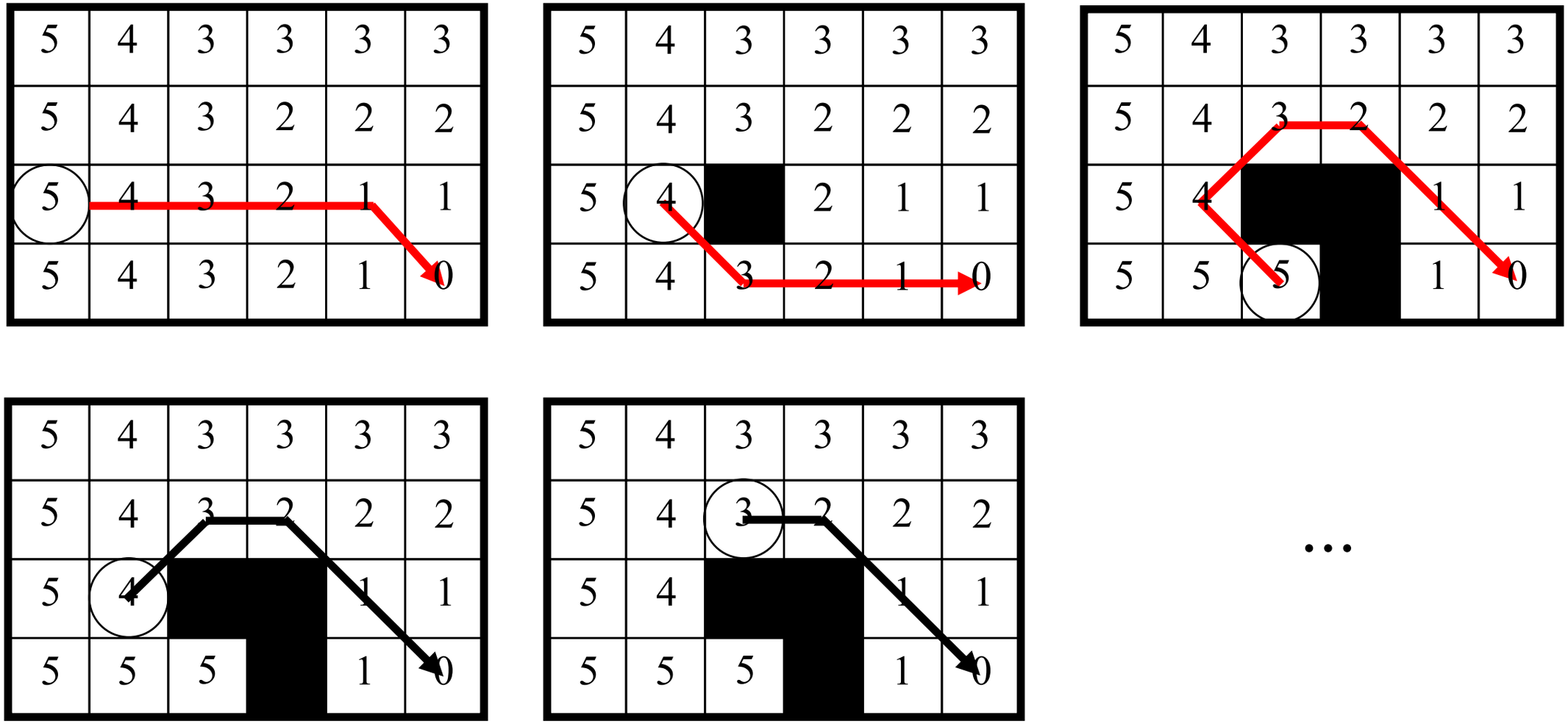
- Fringe Saving A* (FSA*)
- Adaptive A* (AA*)
- Lifelong Planning A* (LPA*), D* Lite and Minimax LPA*
- Comparison of D* Lite and Adaptive A*
- Eager and Lazy Moving-Target Adaptive A* (MTAA*)
- Anytime Replanning A* (ARA*)
- Anytime D*



Sven

goal
distance

D* Lite



8-neighbor grid

D* Lite

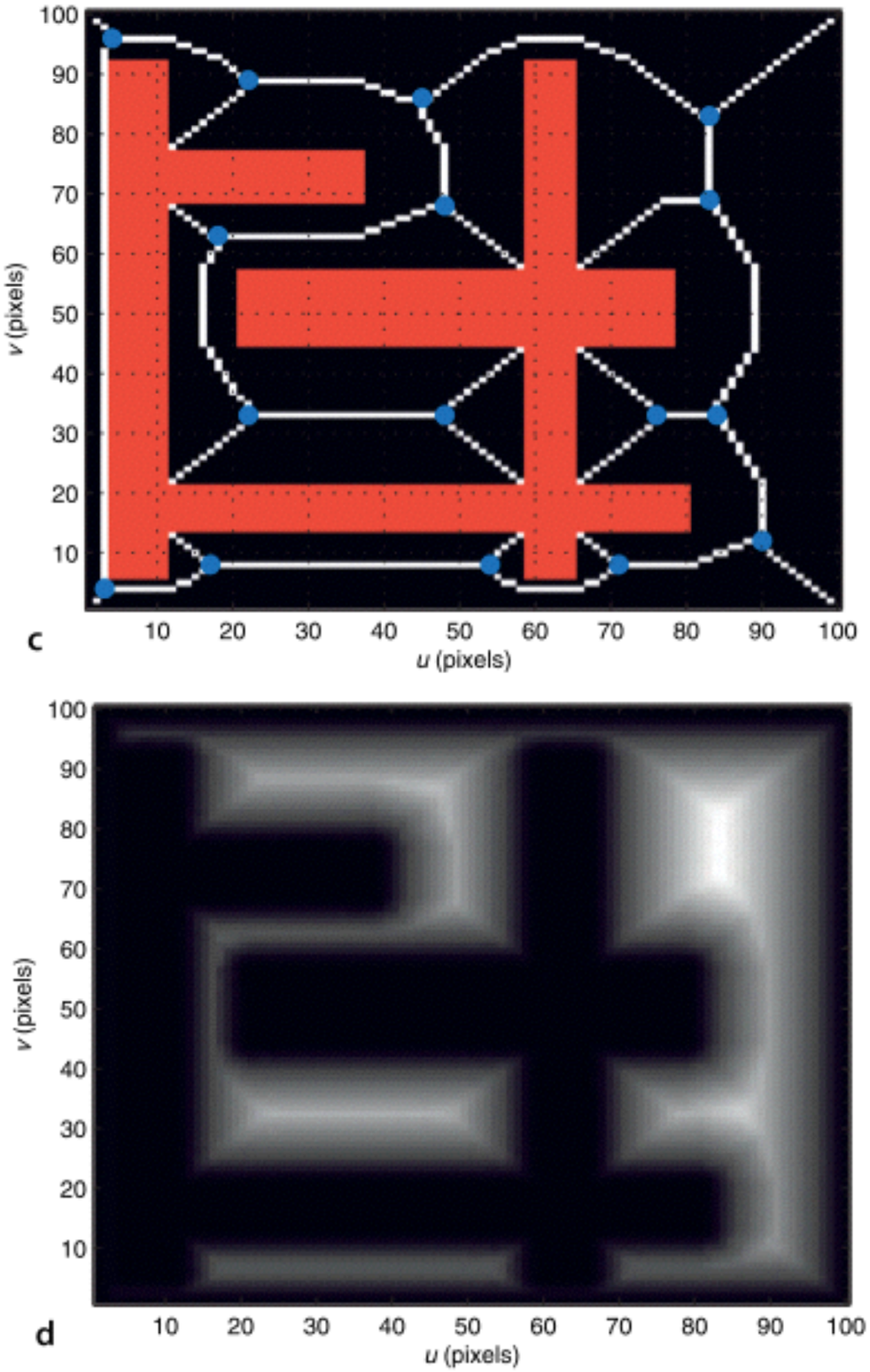
- Random grids of size 129 x 129

	replanning time
uninformed search from scratch	296.0 ms
heuristic search from scratch	10.5 ms
incremental uninformed search	6.1 ms
incremental heuristic search	
D* [Stentz, 1995] D* was probably the first true incremental heuristic search algorithm, way ahead of its time.	4.2 ms
D* Lite	2.7 ms

speed-up 110x

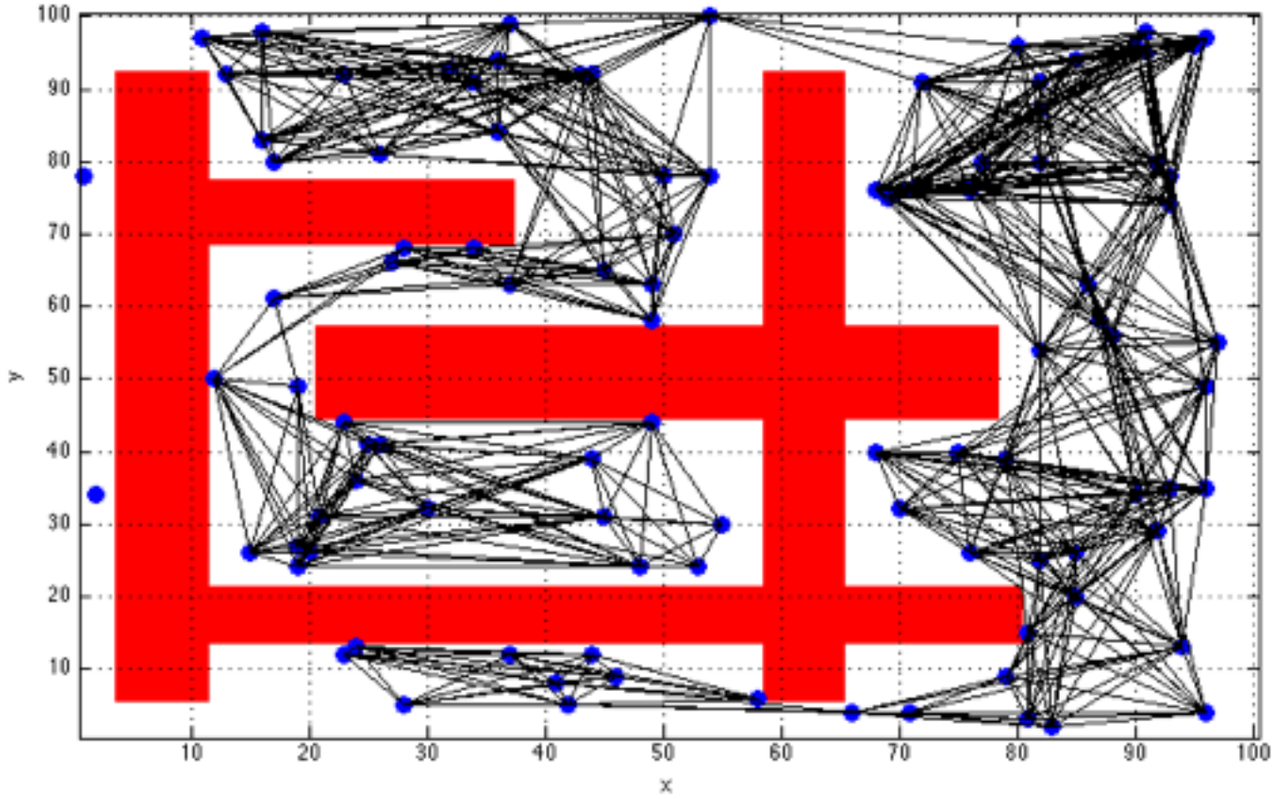
ROADMAP METHODS

- Do some setup work
- Make queries cheap
- Two methods:
  - ***Voronoi Roadmap***
  - PRM



PROBABILISTIC ROADMAPS

Lydia E. Kavraki



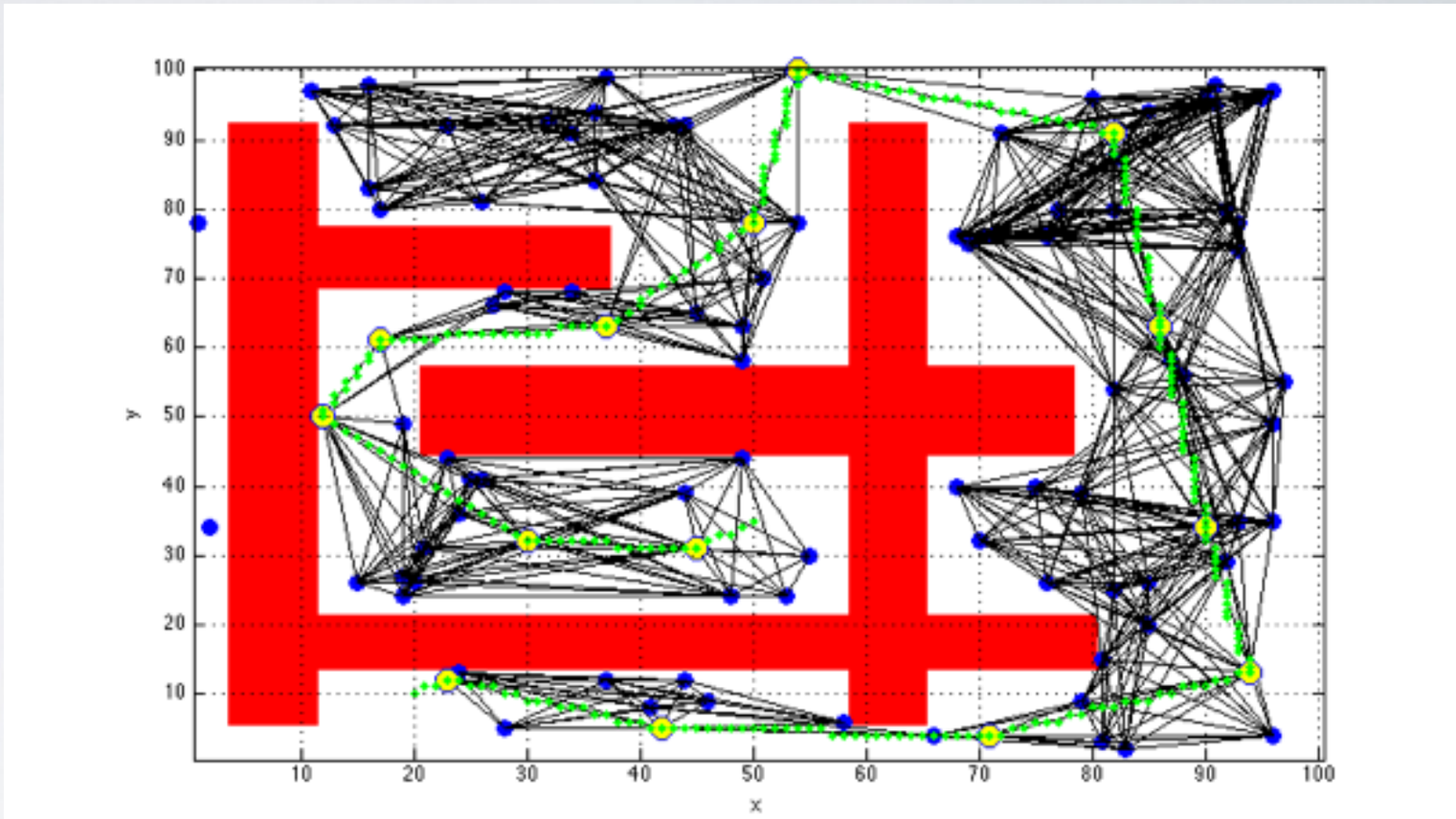
- Randomly sample points
- Connect if no obstacle
- Plan using graph-search
- Disadvantages?

```
randinit
prm = PRM(map)
prm.plan()
prm.plot()
prm.path([20;10],[50;35]);
```

http://www.youtube.com/watch?v=SG6BlmrHGk4

PROBABILISTIC ROADMAPS

Lydia E. Kavraki



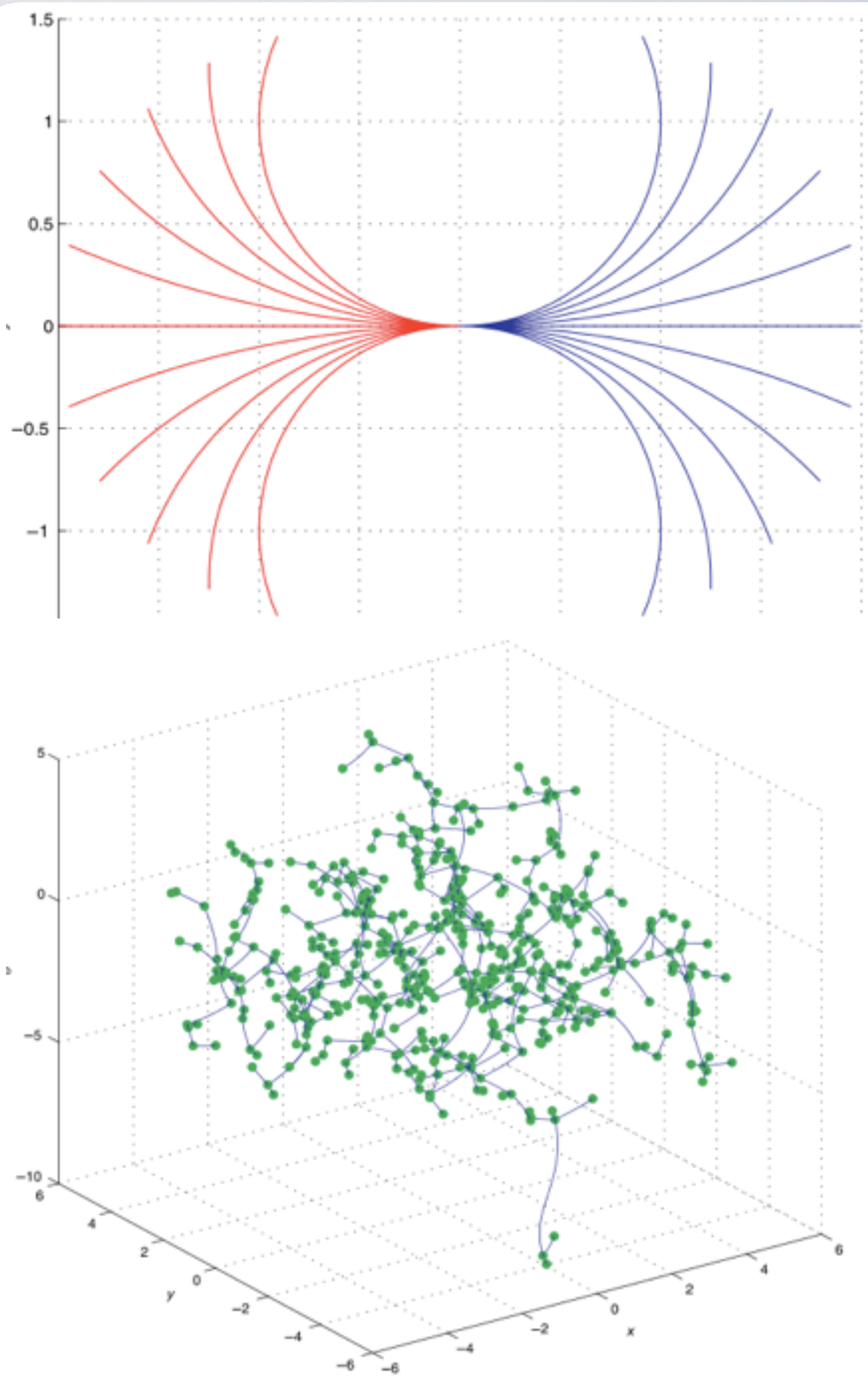
- Randomly sample points
- Connect if no obstacle
- Plan using graph-search
- Disadvantages?

```
randinit
prm = PRM(map)
prm.plan()
prm.plot()
prm.path([20;10],[50;35]);
```

http://www.youtube.com/watch?v=SG6BlmrHGk4

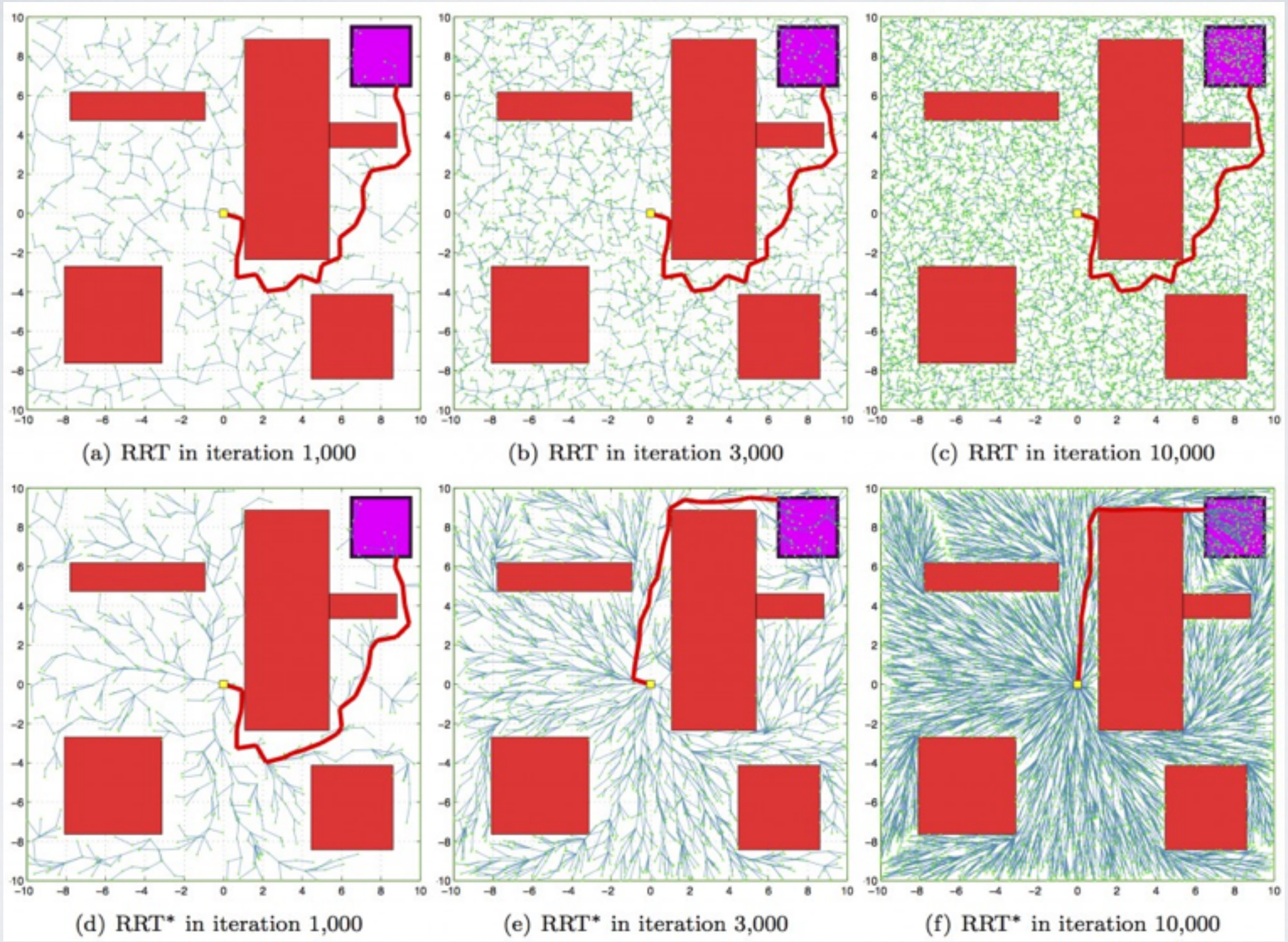
RRT

- Rapidly-exploring Random Tree
- Uses exact kinematic model
- Start with initial pose ξ_0
- At each time:
  - pick random pose ξ_{rand}
  - find nearest pose ξ_{near}
  - move towards it for fixed time, reaching ξ_{new}



see also http://msl.cs.uiuc.edu/rrt/gallery.html

RRT*



(a) RRT in iteration 1,000 (b) RRT in iteration 3,000 (c) RRT in iteration 10,000

(d) RRT* in iteration 1,000 (e) RRT* in iteration 3,000 (f) RRT* in iteration 10,000

- Rewires the tree if shorter paths can be found