ShanghaiTech University 上海科技大学 School of Information Science and Technology 信息科学与技术学院

Introduction to Information Science and Technology (EE 100)
Part II: Intelligent Machines and Robotics
Planning and Maps

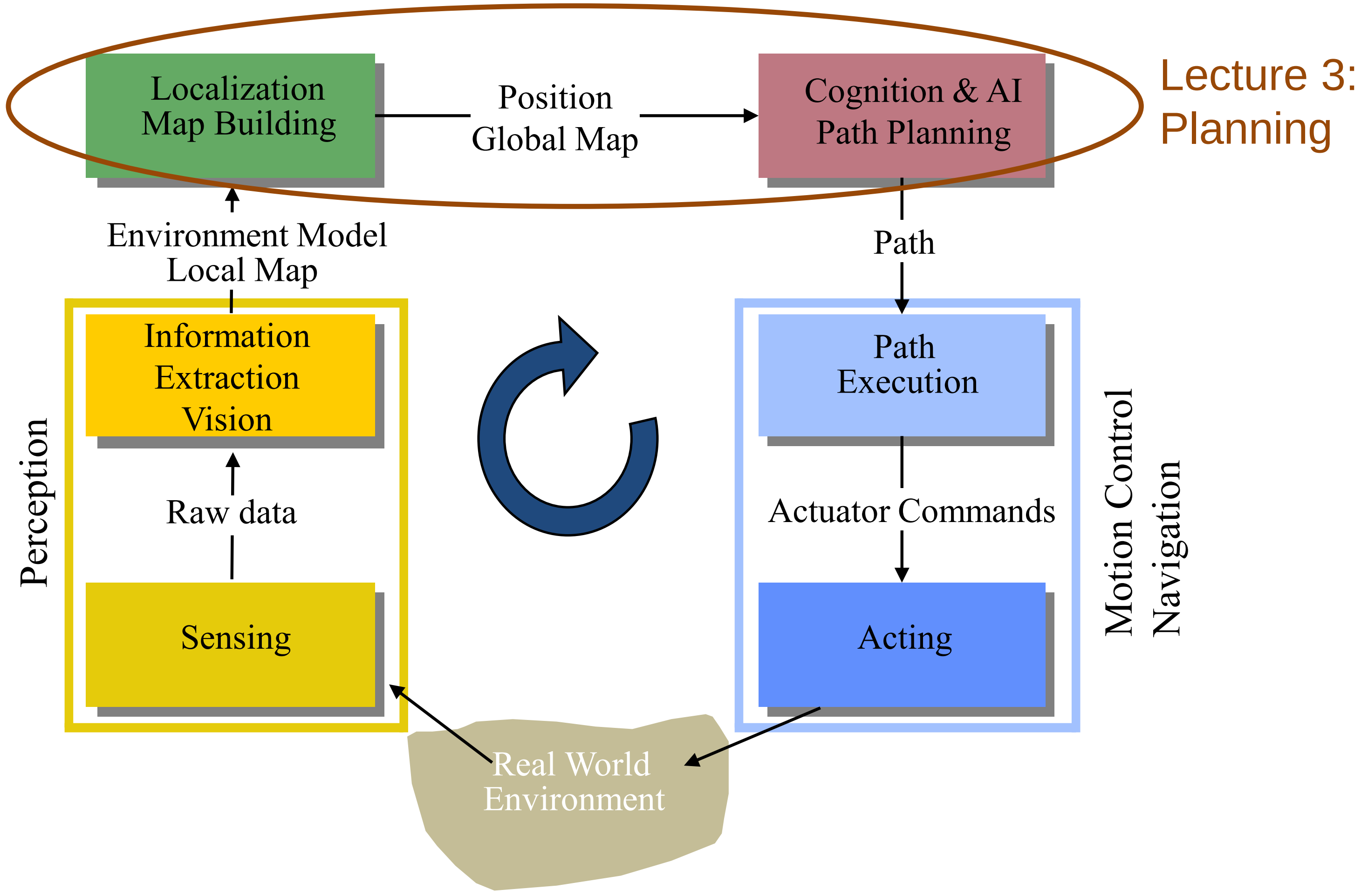
Sören Schwertfeger / 师泽仁

ShanghaiTech University

- Autonomous mobile robots move around in the environment. Therefore **ALL** of them:
  - They need to know **where** they **are**.
  - They need to know **where** their **goal** is.
  - They need to know **how** to get there.

- Different levels:
  - Control:
    - How much power to the motors to move in that direction, reach desired speed
  - Navigation:
    - Avoid obstacles
    - Classify the terrain in front of you
    - Follow a path
  - Planning:
    - Long distance path planning
    - What is the way, optimize for certain parameters

General Control Scheme for Mobile Robot Systems



With material from Roland Siegwart and Davide Scaramuzza, ETH Zurich

MAPS

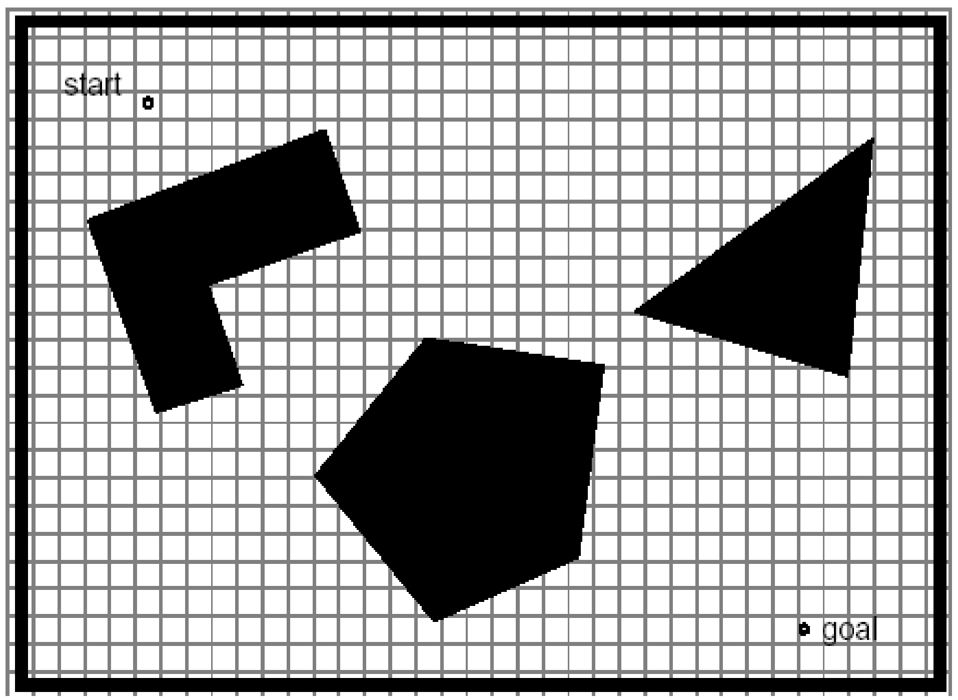
Representation of the Environment

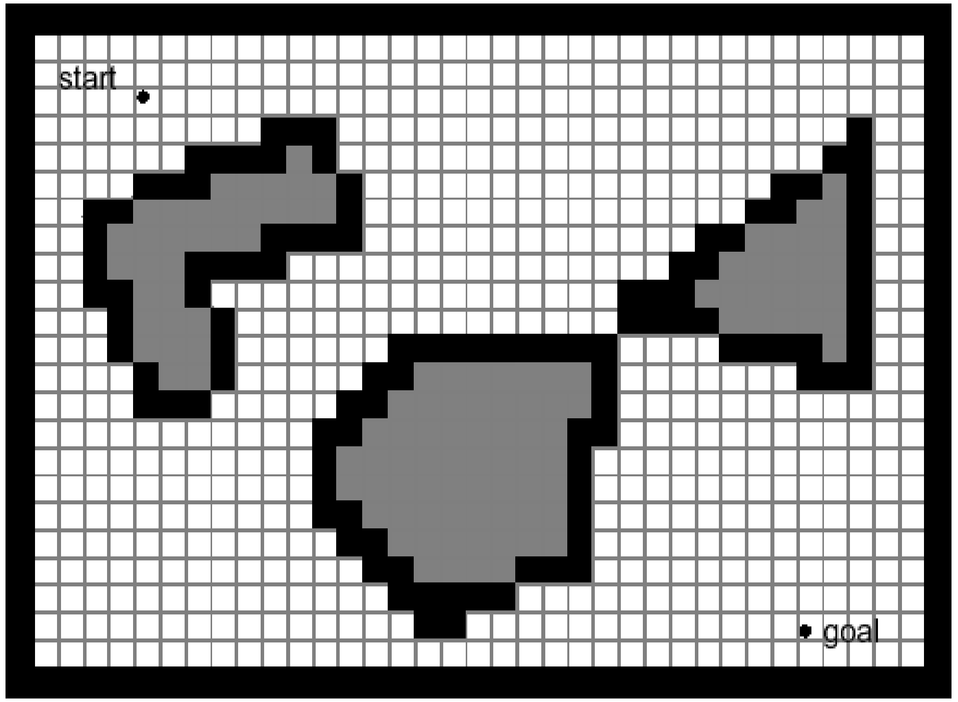
- Environment Representation
  - Continuous Metric $\rightarrow$ x, y, θ
  - Discrete Metric $\rightarrow$ metric grid
  - Discrete Topological $\rightarrow$ topological grid

- Environment Modeling
  - Raw sensor data, e.g. laser range data, grayscale images
    - large volume of data, low distinctiveness on the level of individual values
    - makes use of all acquired information
  - Low level features, e.g. line other geometric features
    - medium volume of data, average distinctiveness
    - filters out the useful information, still ambiguities
  - High level features, e.g. doors, a car, the Eiffel tower
    - low volume of data, high distinctiveness
    - filters out the useful information, few/ no ambiguities

Map Representation: Approximate cell decomposition

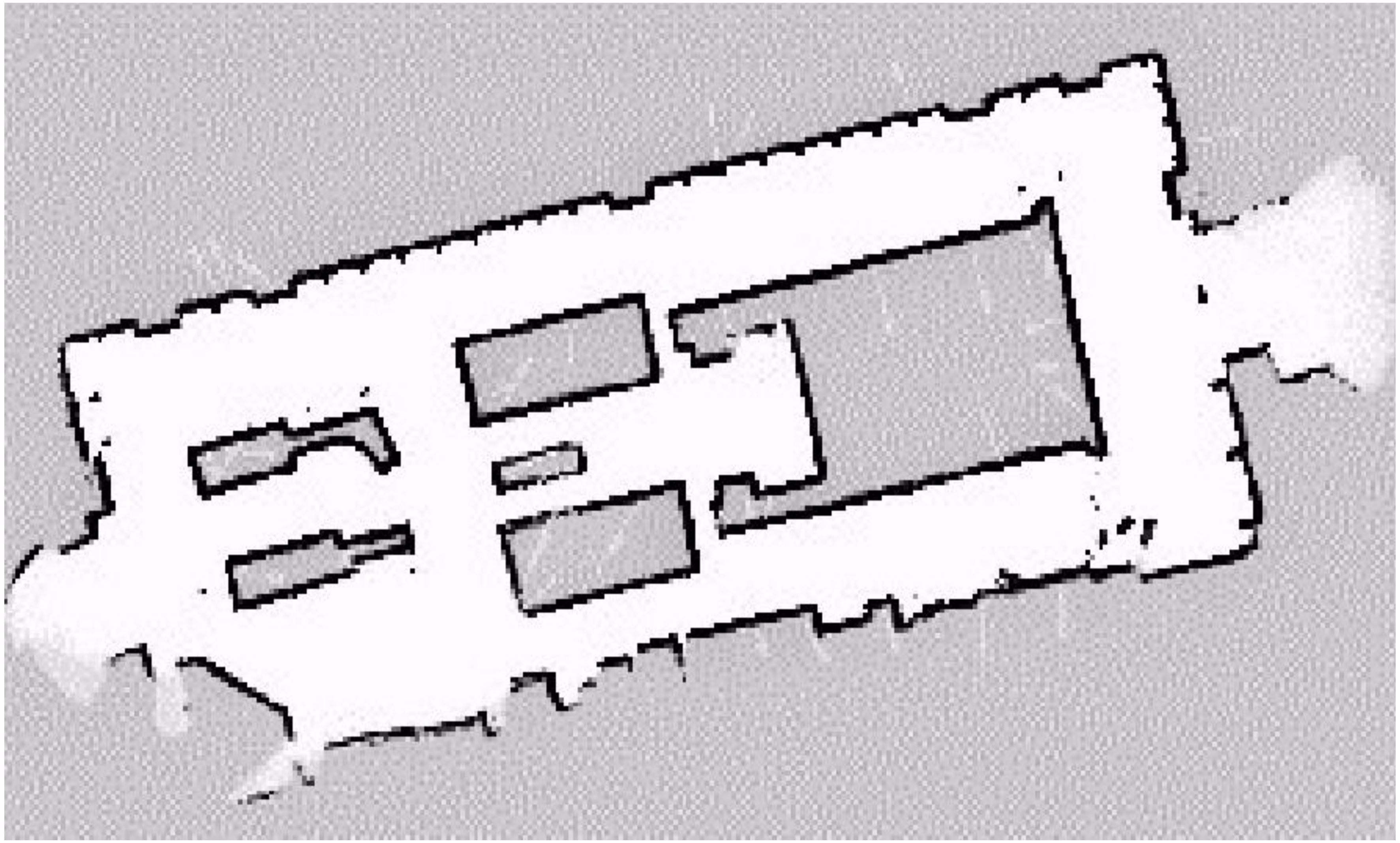
- Fixed cell decomposition => 2D grid map
  - Cells: probability of being occupied =>
    - 0 free; 0.5 (or 128) unknown; 1 or (255) occupied





Map Representation: Occupancy grid

- Fixed cell decomposition: occupancy grid example



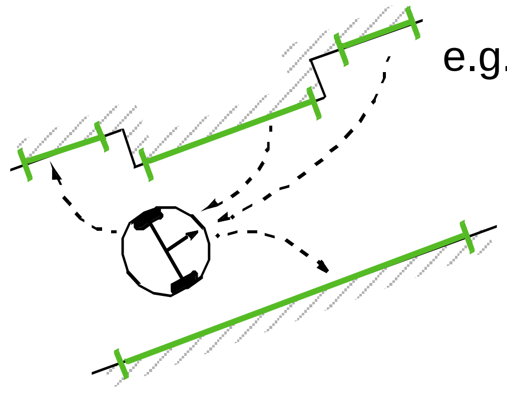
Courtesy of S. Thrun

SLAM

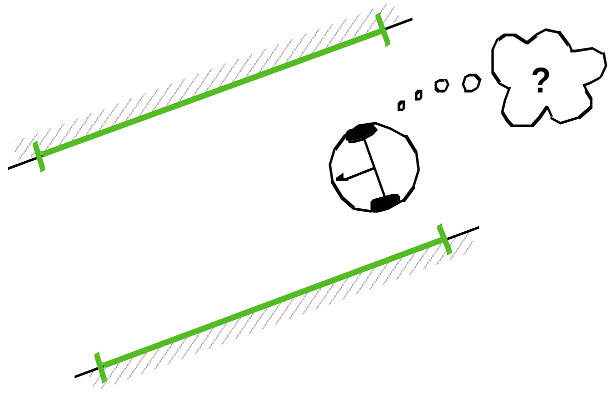
Simultaneous Localization and Mapping

Map Building: The Problems

1. **Map Maintaining:** Keeping track of changes in the environment



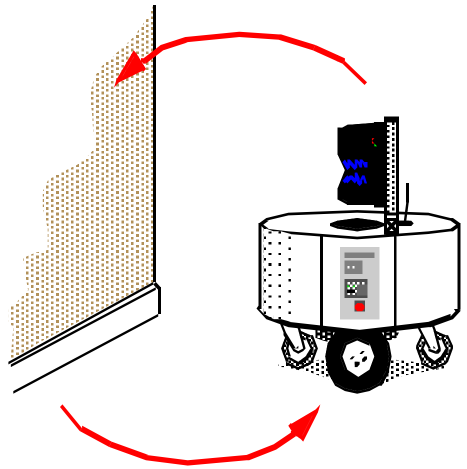
e.g. disappearing cupboard



- e.g. measure of belief of each environment feature

2. **Representation and Reduction of Uncertainty**

position of robot -> position of wall

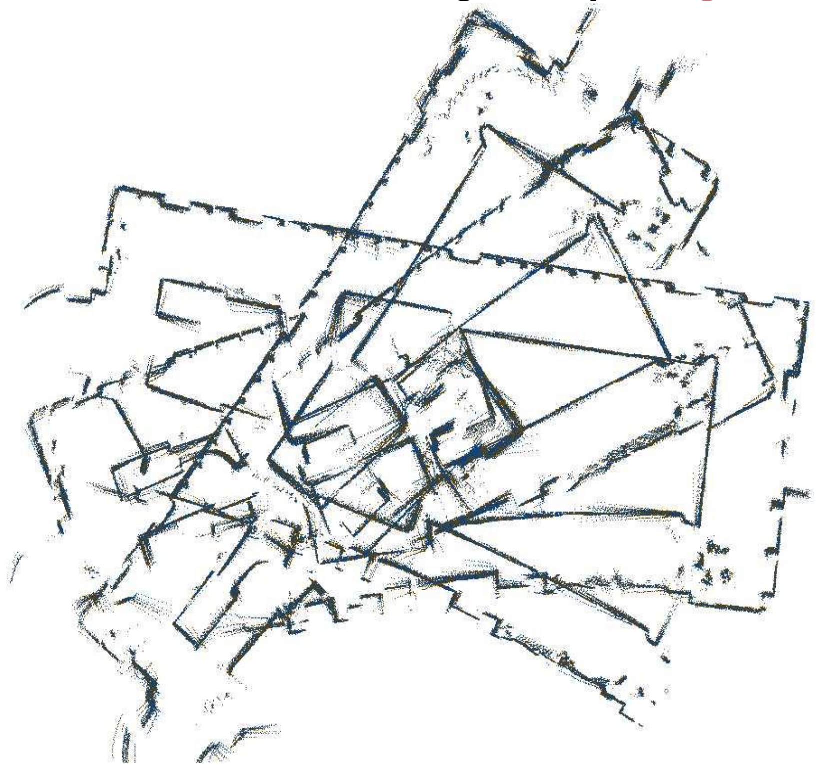


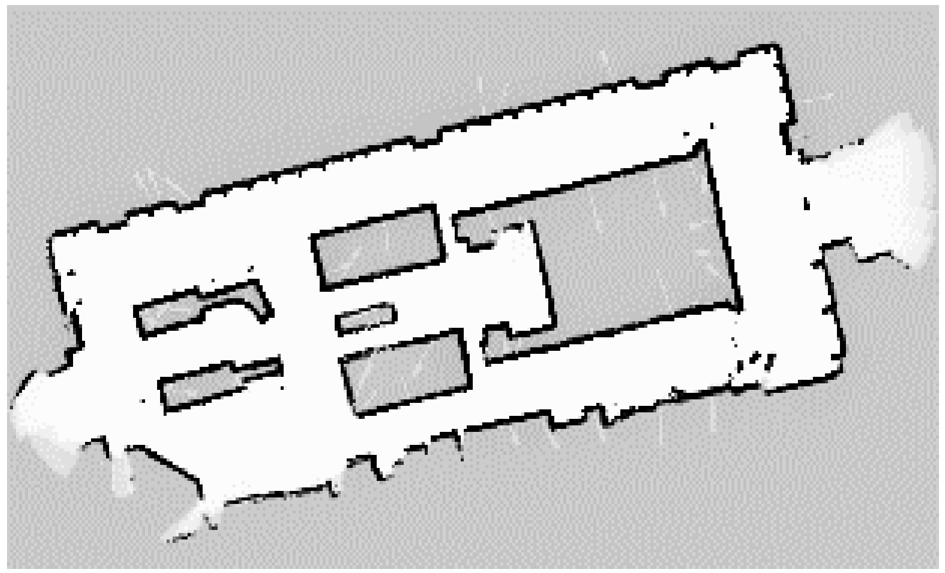
position of wall -> position of robot

- Inconsistent map due to motion drift

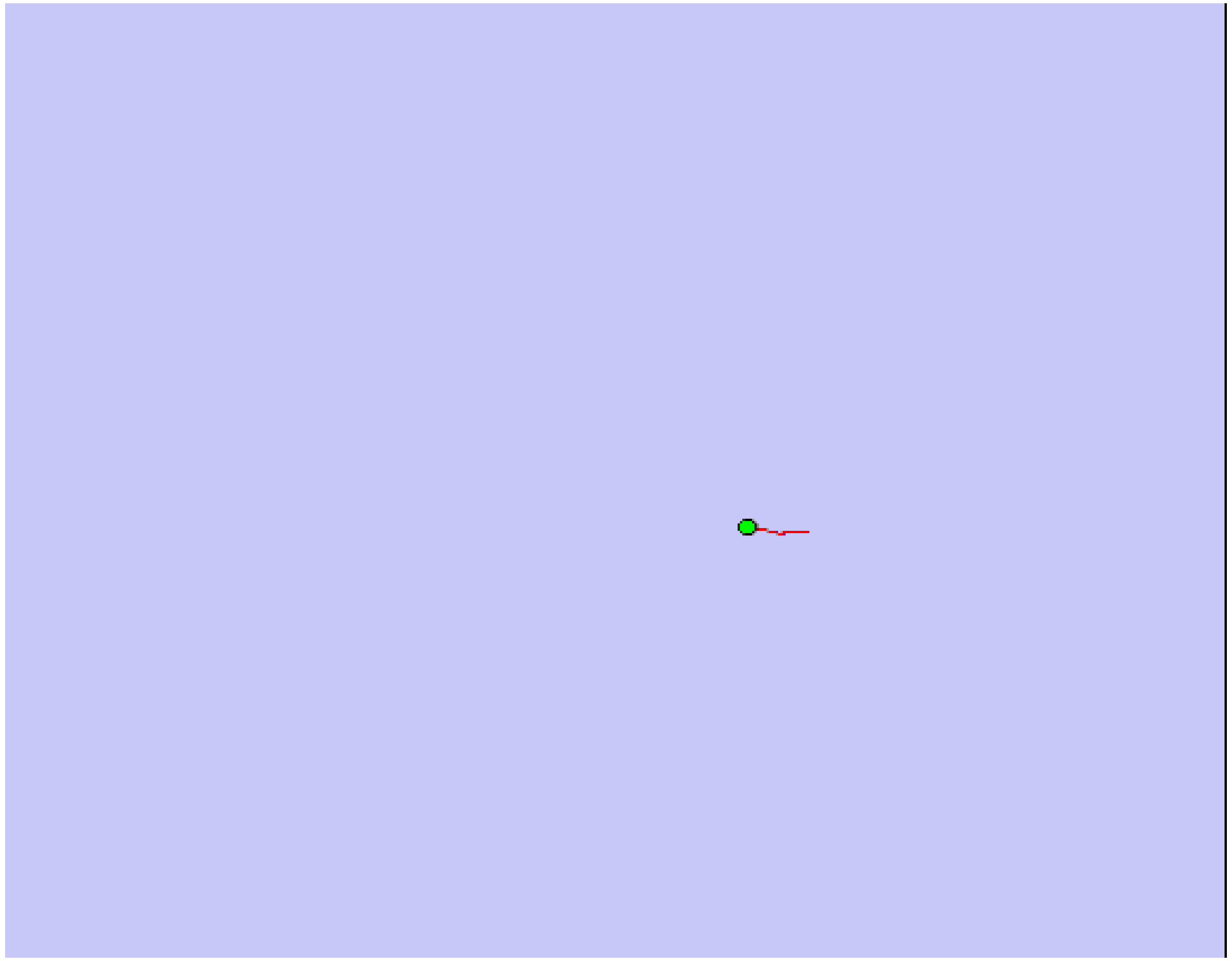
Cyclic Environments

- Small local error accumulate to arbitrary large global errors!
- This is usually irrelevant for navigation
- However, when closing loops, global error does matter



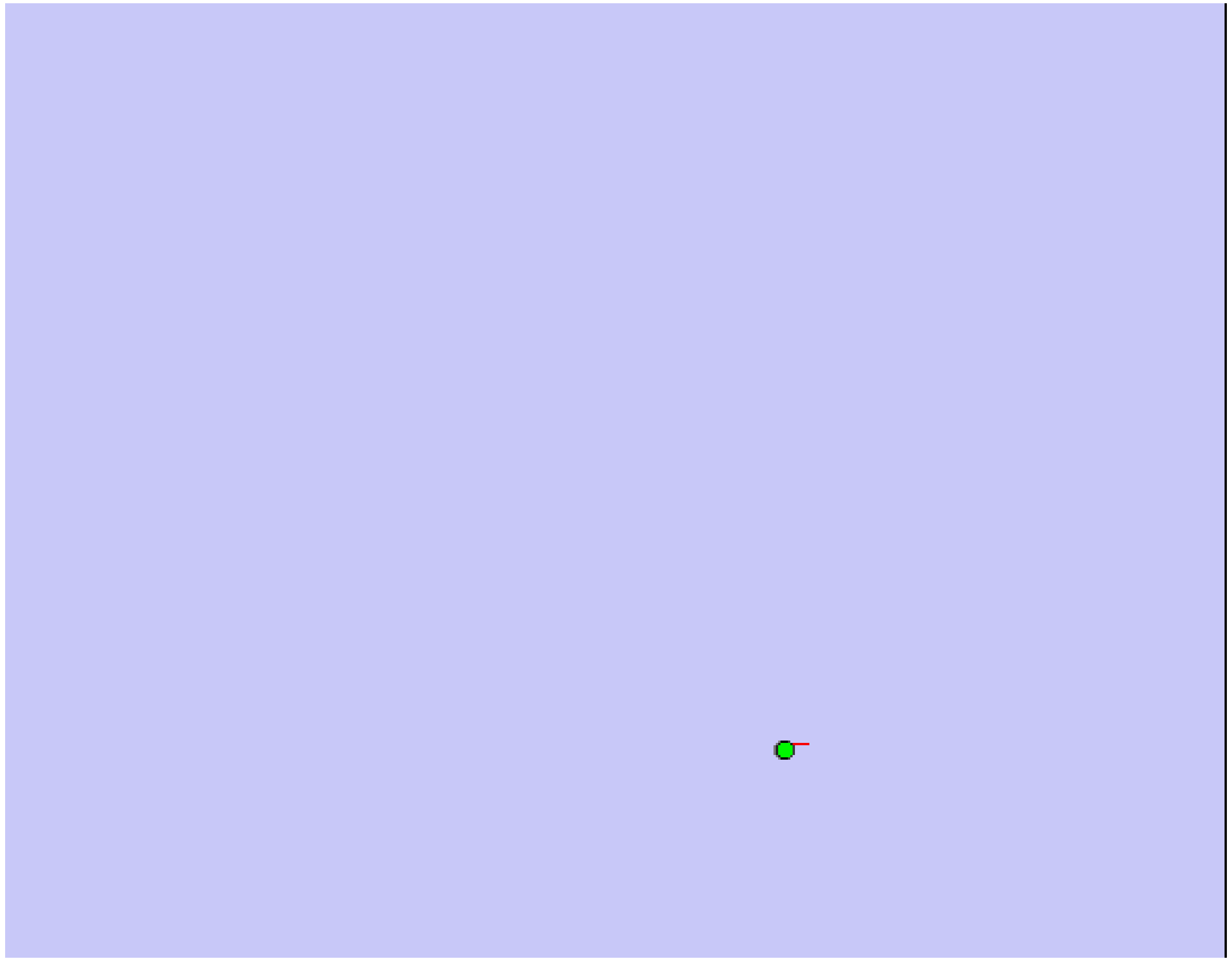


Raw Odometry …



Courtesy of S. Thrun

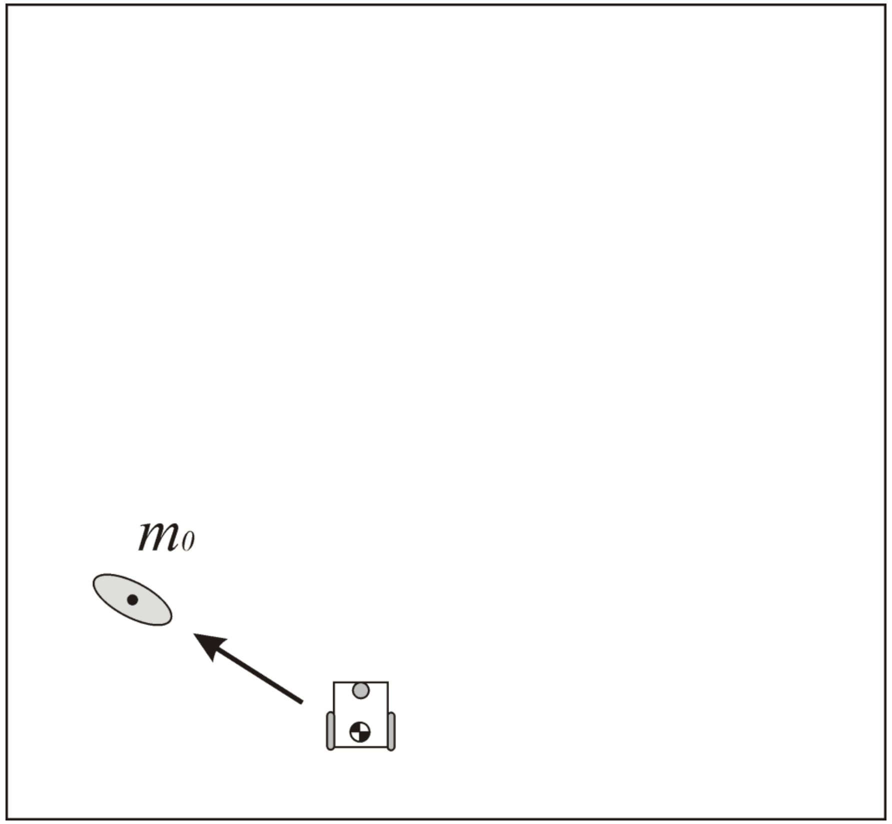
Scan Matching:
compare to sensor data from previous scan



Courtesy of S. Thrun

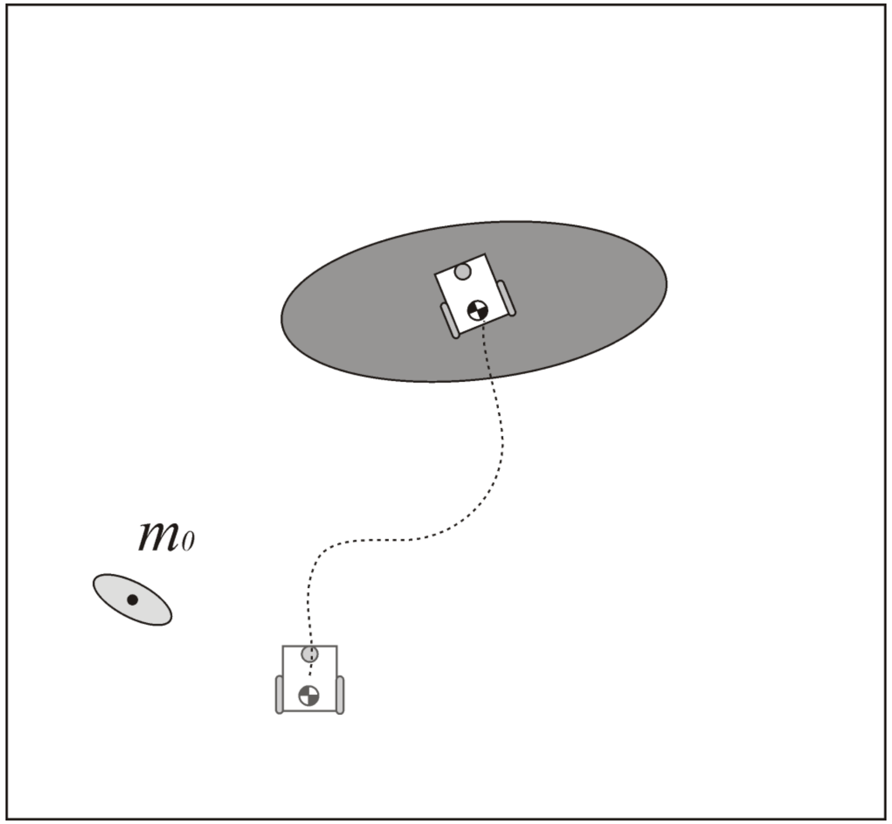
SLAM overview

- Let us assume that the robot uncertainty at its initial location is zero.
- From this position, the robot observes a feature which is mapped with an uncertainty related to the sensor error model



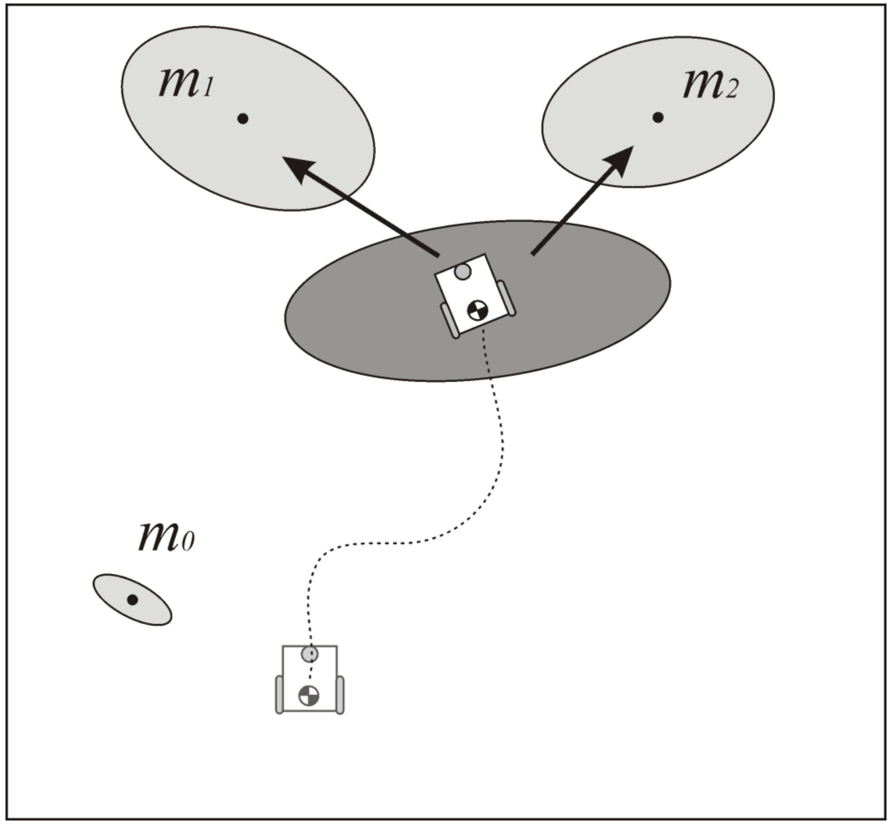
SLAM overview

- As the robot moves, its pose uncertainty increases under the effect of the errors introduced by the odometry



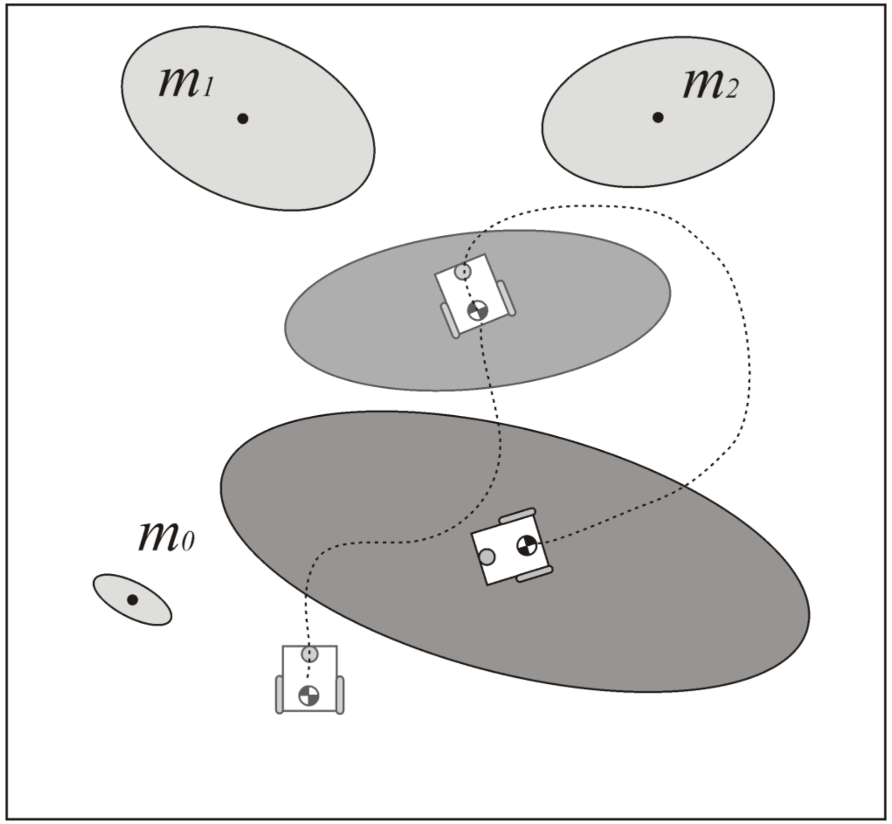
SLAM overview

- At this point, the robot observes two features and maps them with an uncertainty which results from the combination of the measurement error with the robot pose uncertainty
- From this, we can notice that the map becomes correlated with the robot position estimate. Similarly, if the robot updates its position based on an observation of an imprecisely known feature in the map, the resulting position estimate becomes correlated with the feature location estimate.



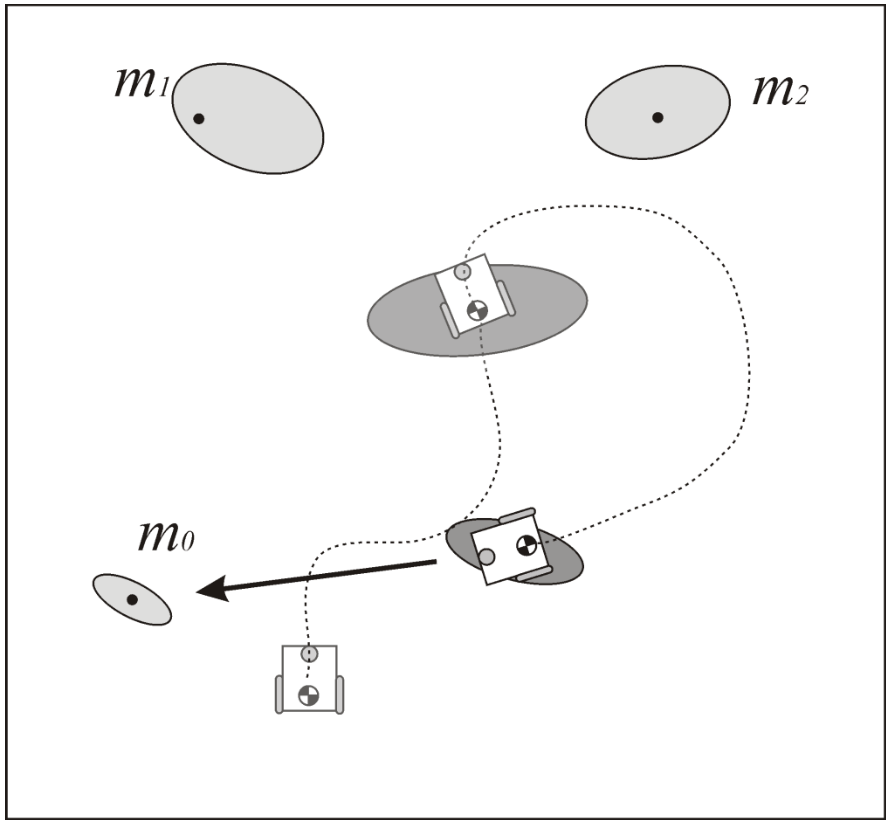
SLAM overview

- The robot moves again and its uncertainty increases under the effect of the errors introduced by the odometry

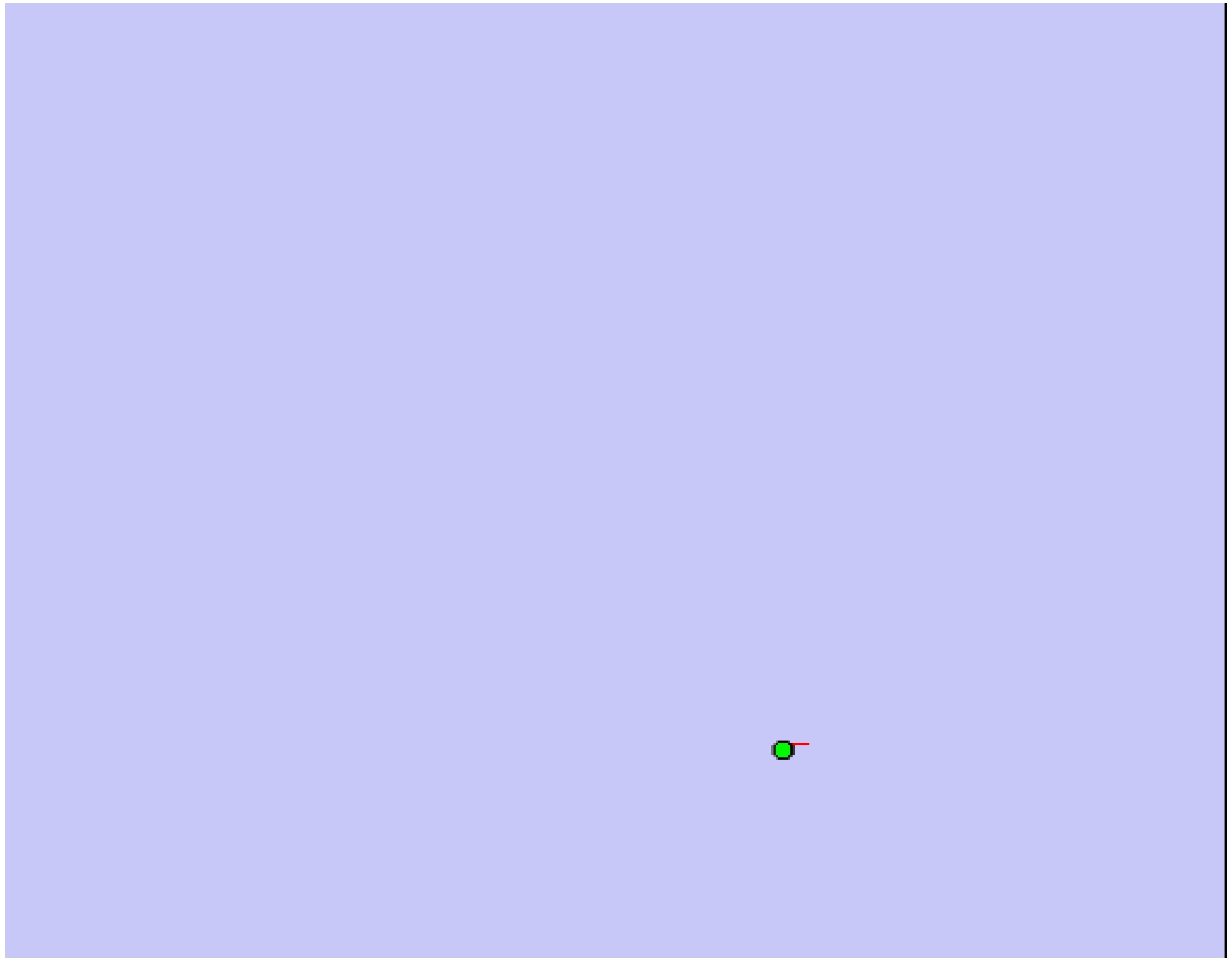


SLAM overview

- In order to reduce its uncertainty, the robot must observe features whose location is relatively well known. These features can for instance be landmarks that the robot has already observed before.
- In this case, the observation is called *loop closure detection.*
- When a loop closure is detected, the robot pose uncertainty shrinks.
- At the same time, the map is updated and the uncertainty of other observed features and all previous robot poses also reduce

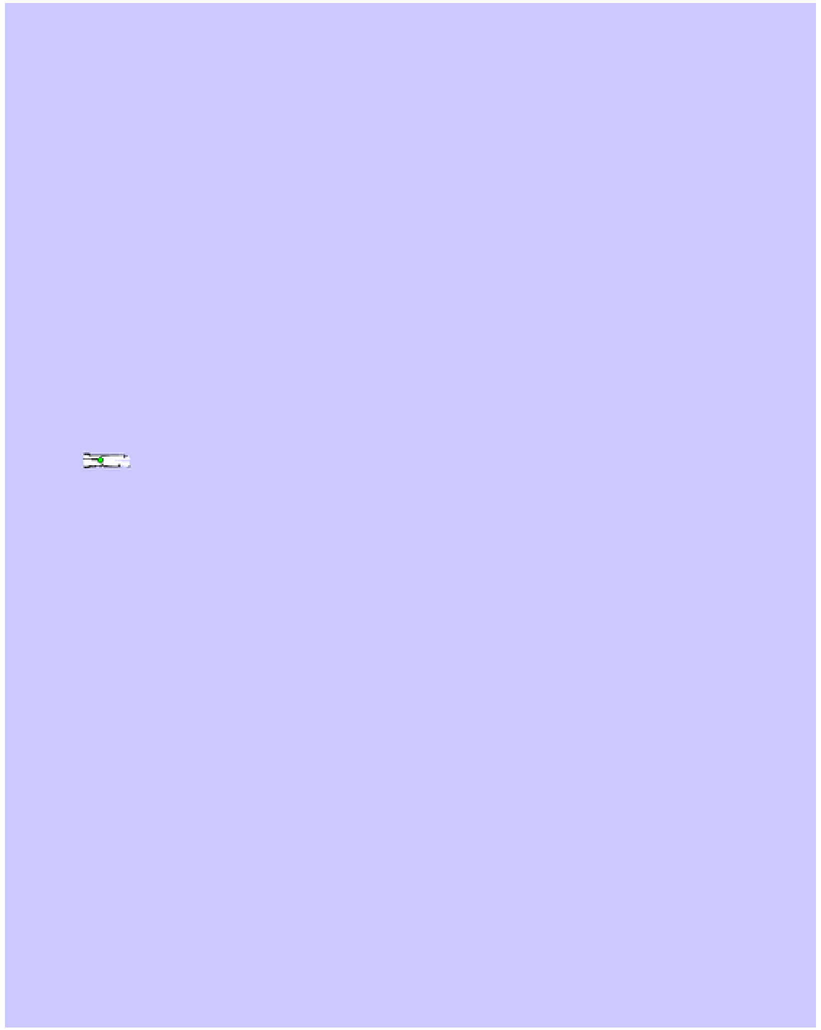


FAST SLAM Example



Courtesy of S. Thrun

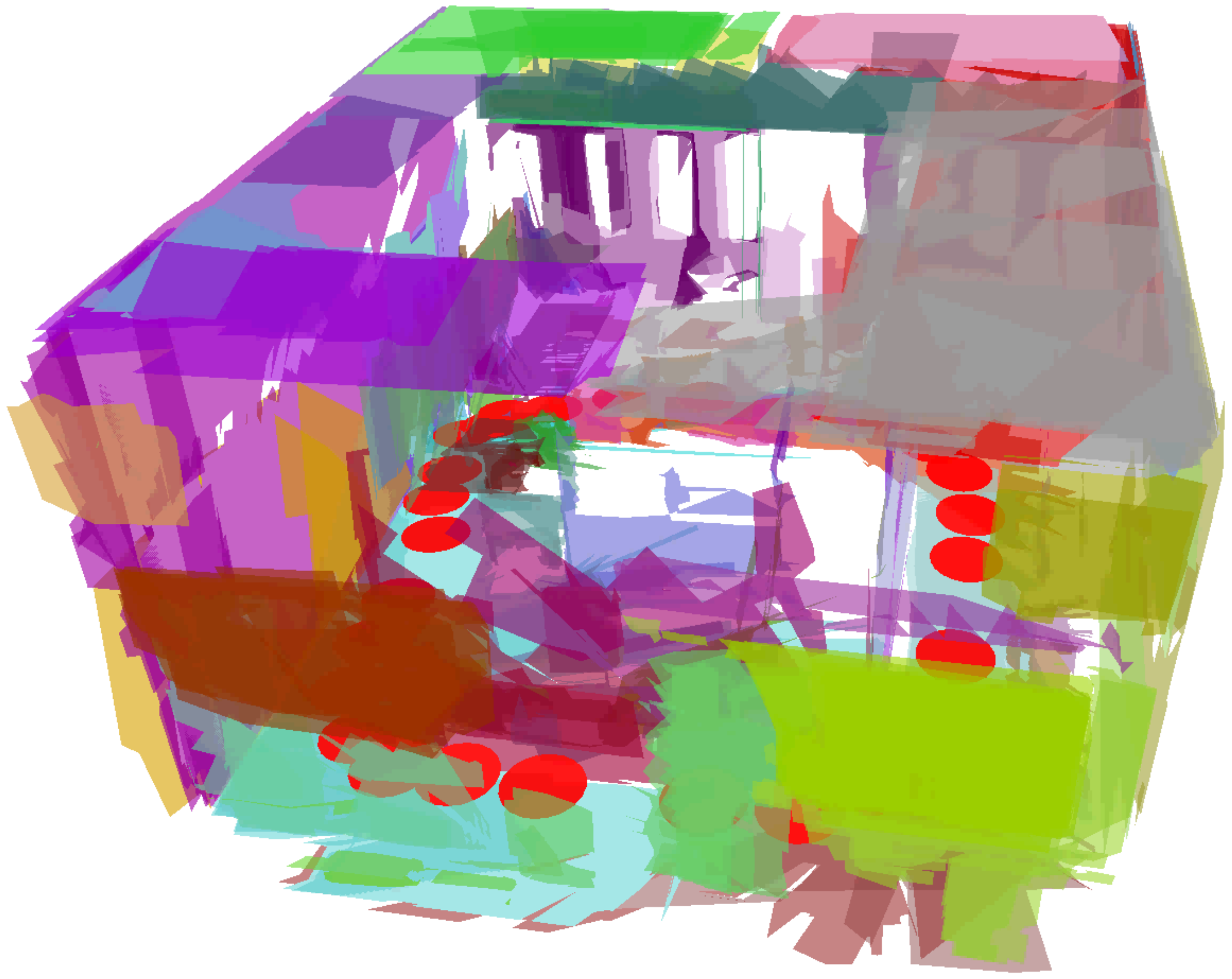
FAST SLAM example



Courtesy of S. Thrun

Jacobs 3D Mapping – Plane Mapping

Experiment Lab Run: 29 3D point-clouds; size of each: 541 x 361 = 195,301



Jacobs 3D Mapping – Plane Mapping

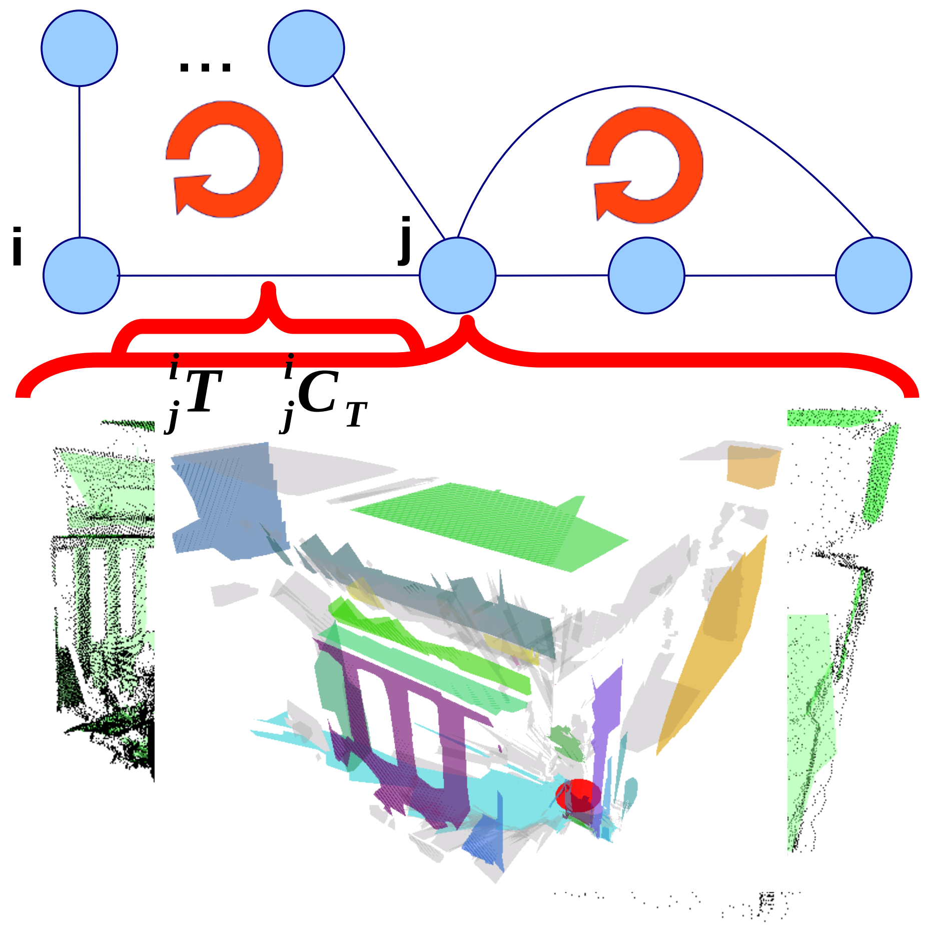
Pose Graph

3D Range Sensing

Plane Extraction

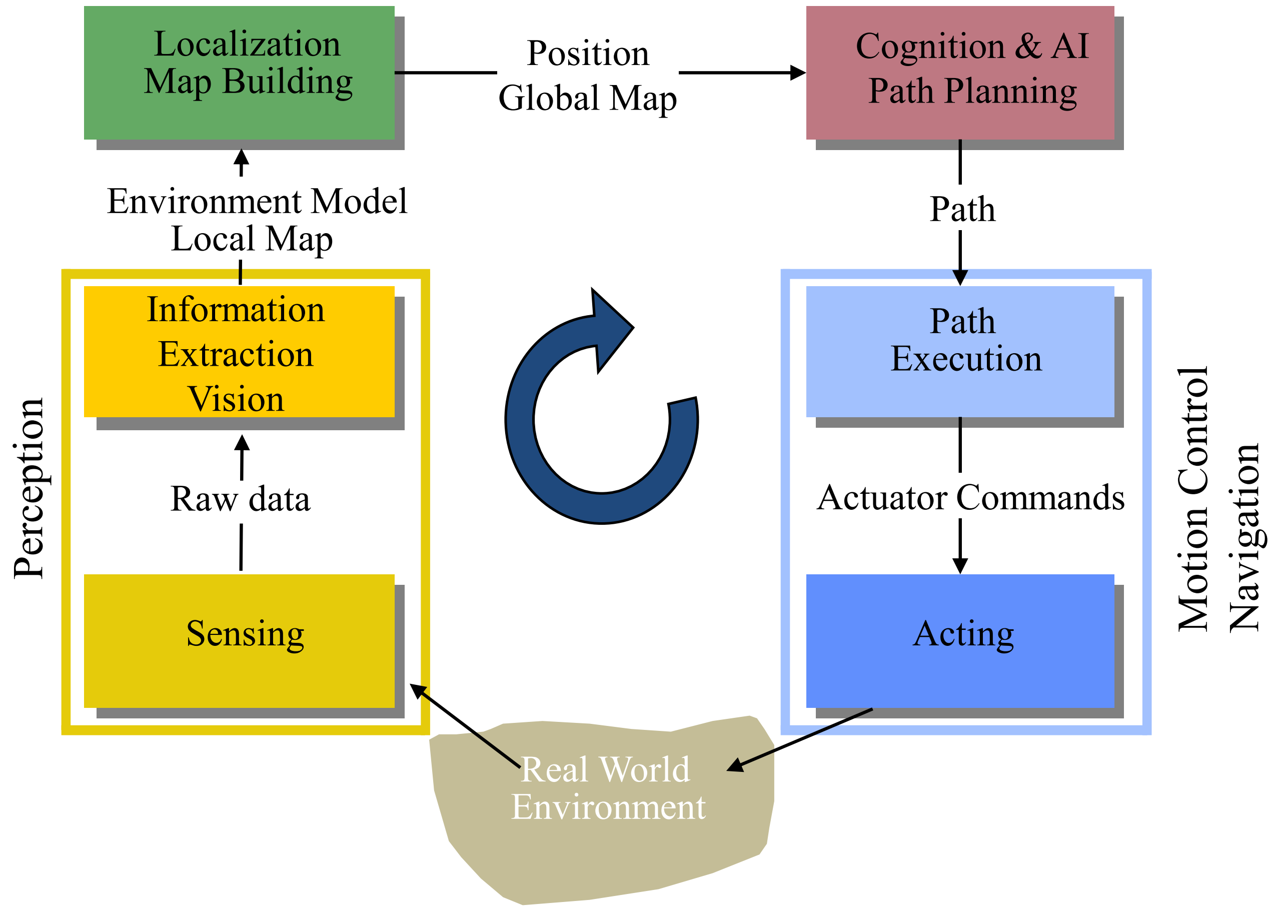
Planar Scan Matching

Relax Loop-Closing Errors



PLANNING

General Control Scheme for Mobile Robot Systems

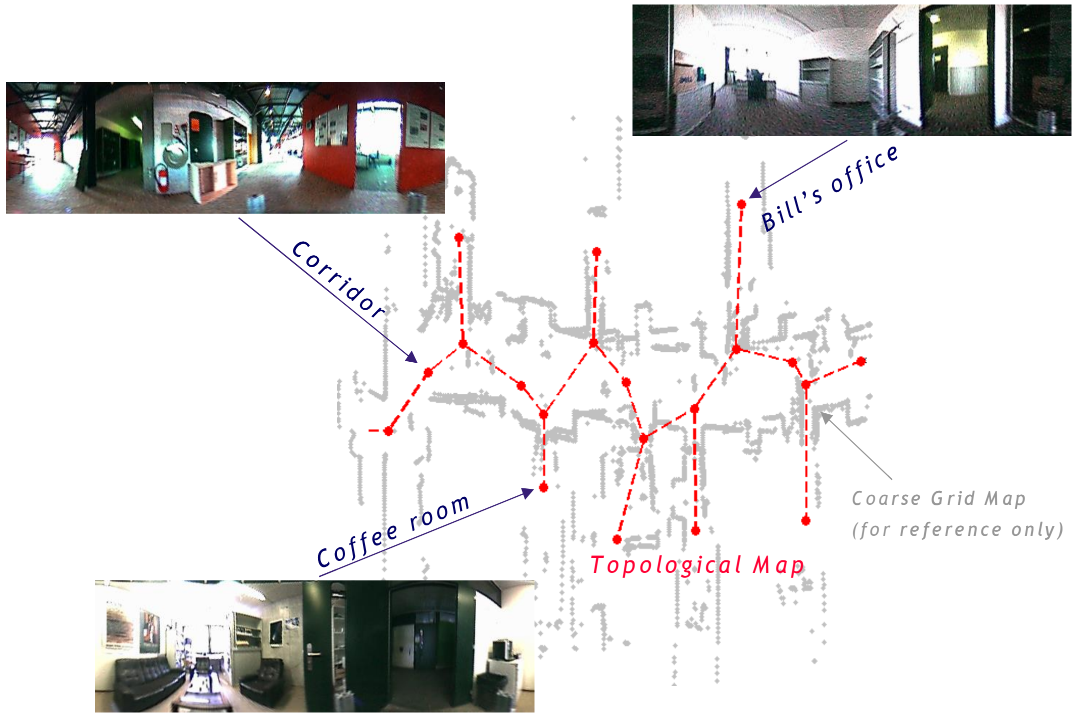


With material from Roland Siegwart and Davide Scaramuzza, ETH Zurich

The Planning Problem

- The problem: find a path in the work space (physical space) from the initial position to the goal position avoiding all collisions with the obstacles

- Assumption: there exists a good enough map of the environment for navigation.

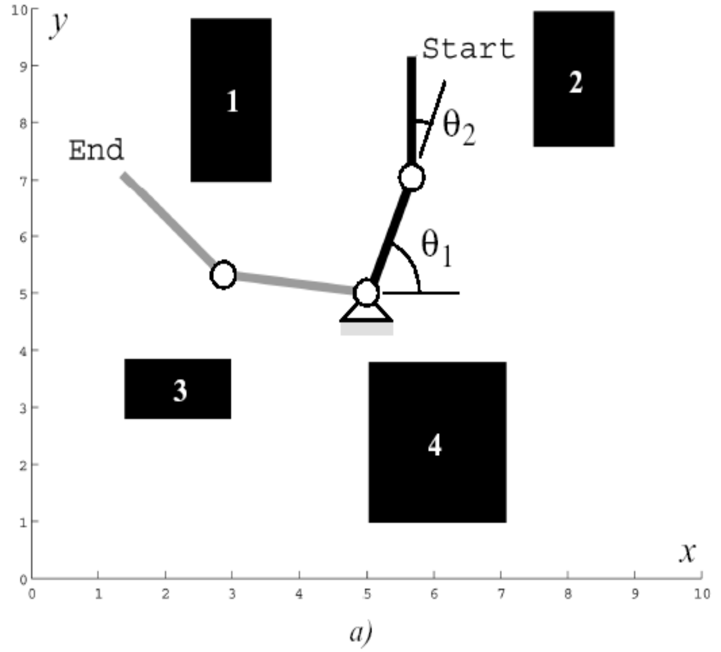


The Planning Problem

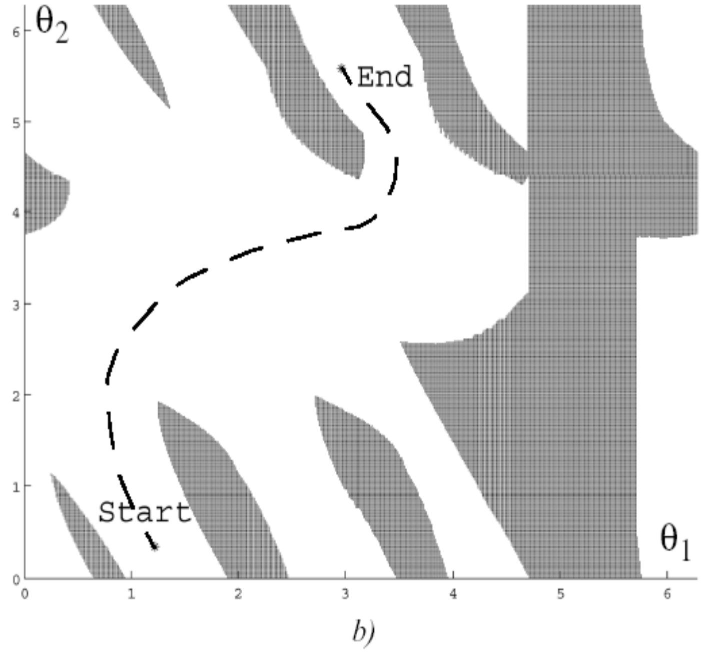
- We can generally distinguish between
  - (global) path planning and
  - (local) obstacle avoidance.
- First step:
  - Transformation of the map into a representation useful for planning
  - This step is planner-dependent
- Second step:
  - Plan a path on the transformed map
- Third step:
  - Send motion commands to controller
  - This step is planner-dependent (e.g. Model based feed forward, path following)

Work Space (Map) → Configuration Space

- State or configuration *q* can be described with *k* values q_i



Work Space

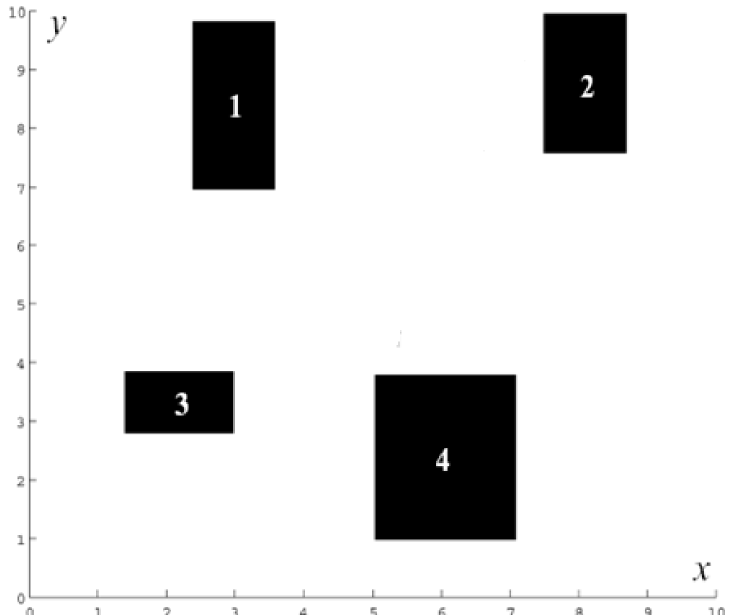


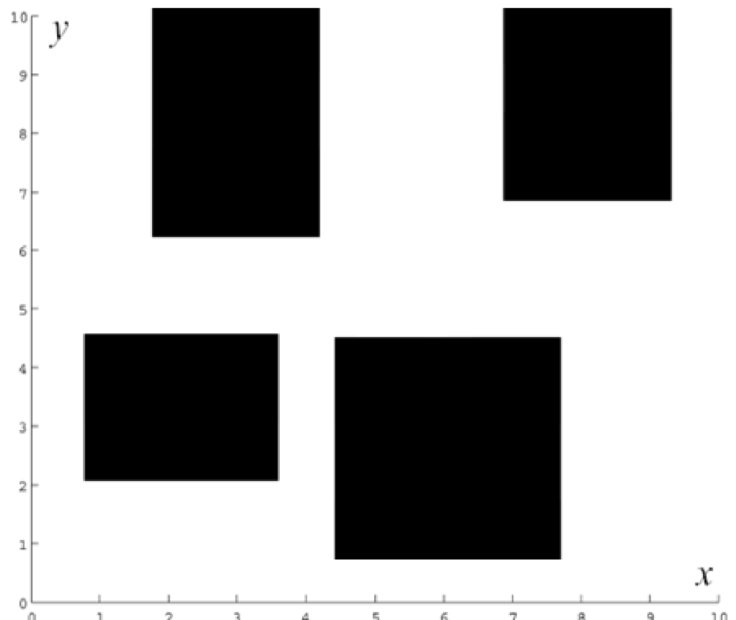
Configuration Space:
the dimension of this
space is equal to the Degrees of Freedom (DoF)
of the robot

- What is the configuration space of a mobile robot?

Configuration Space for a Mobile Robot

- Mobile robots operating on a flat ground (2D) have 3 DoF: (x, y, θ)
- Differential Drive: only two motors => only 2 degrees of freedom directly controlled (forward/ backward + turn) => non-holonomic
- Simplification: assume robot is holonomic and it is a point => configuration space is reduced to 2D (x,y)
- => inflate obstacle by size of the robot radius to avoid crashes => obstacle growing

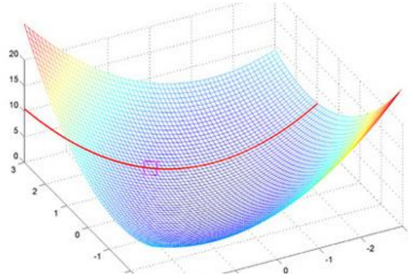




Path Planning: Overview of Algorithms

1. Optimal Control

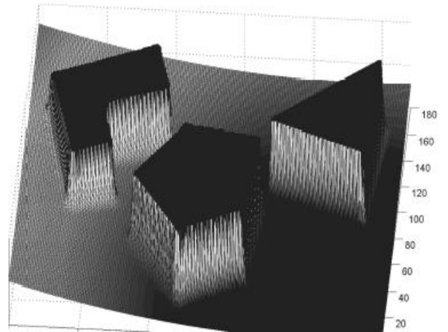
- Solves truly optimal solution
- Becomes intractable for even moderately complex as well as nonconvex problems



Source: http://mitocw.udsm.ac.tz

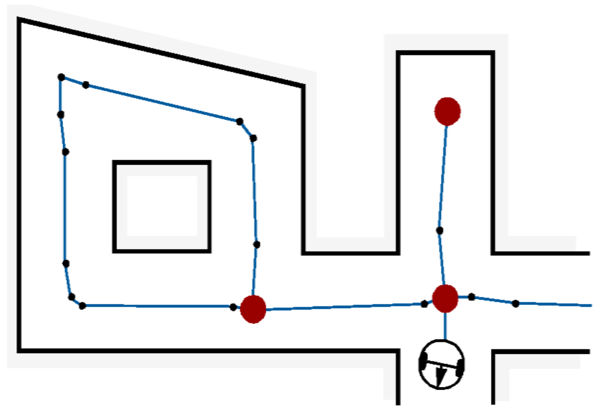
2. Potential Field

- Imposes a mathematical function over the state/configuration space
- Many physical metaphors exist
- Often employed due to its simplicity and similarity to optimal control solutions

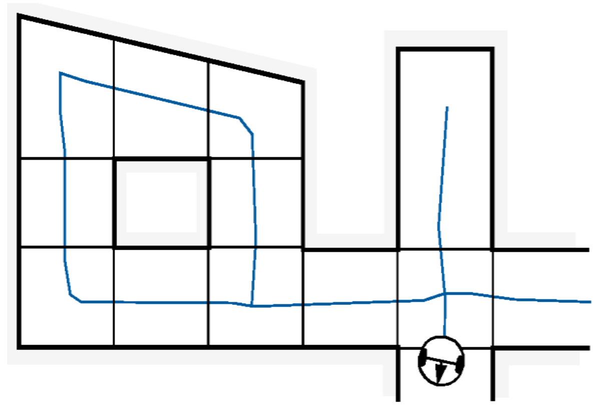


3. Graph Search

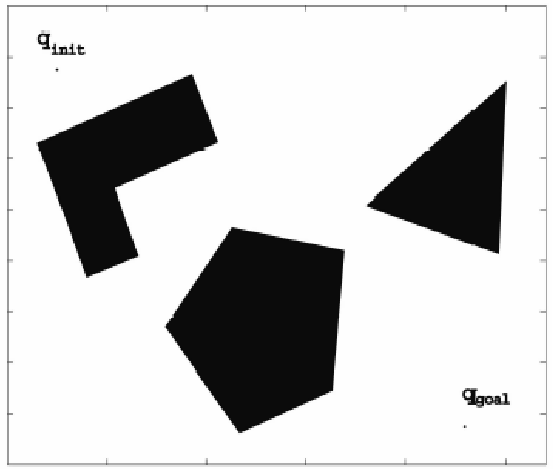
- Identify a set edges between nodes within the free space



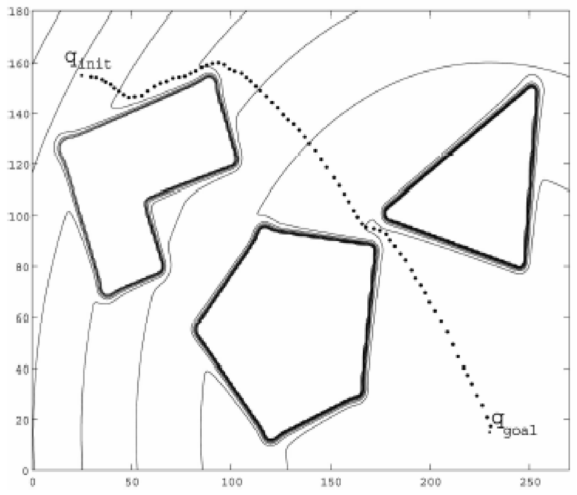
- Where to put the nodes?

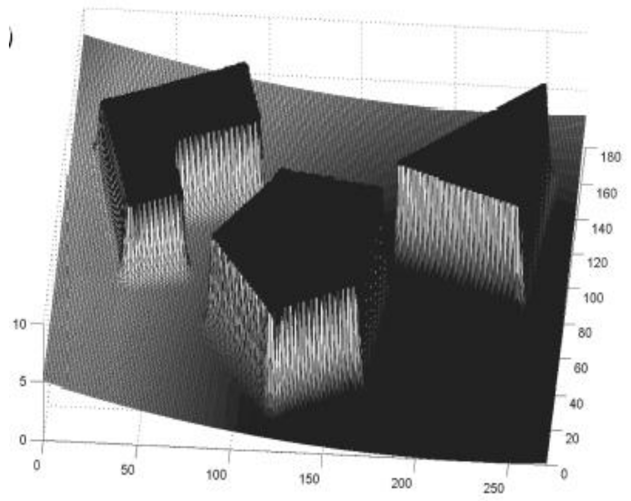


Potential Field Path Planning Strategies



- Robot is treated as a *point under the influence* of an artificial potential field.
- Operates in the continuum
  - Generated robot movement is similar to a ball rolling down the hill
  - Goal generates attractive force
  - Obstacle are repulsive forces





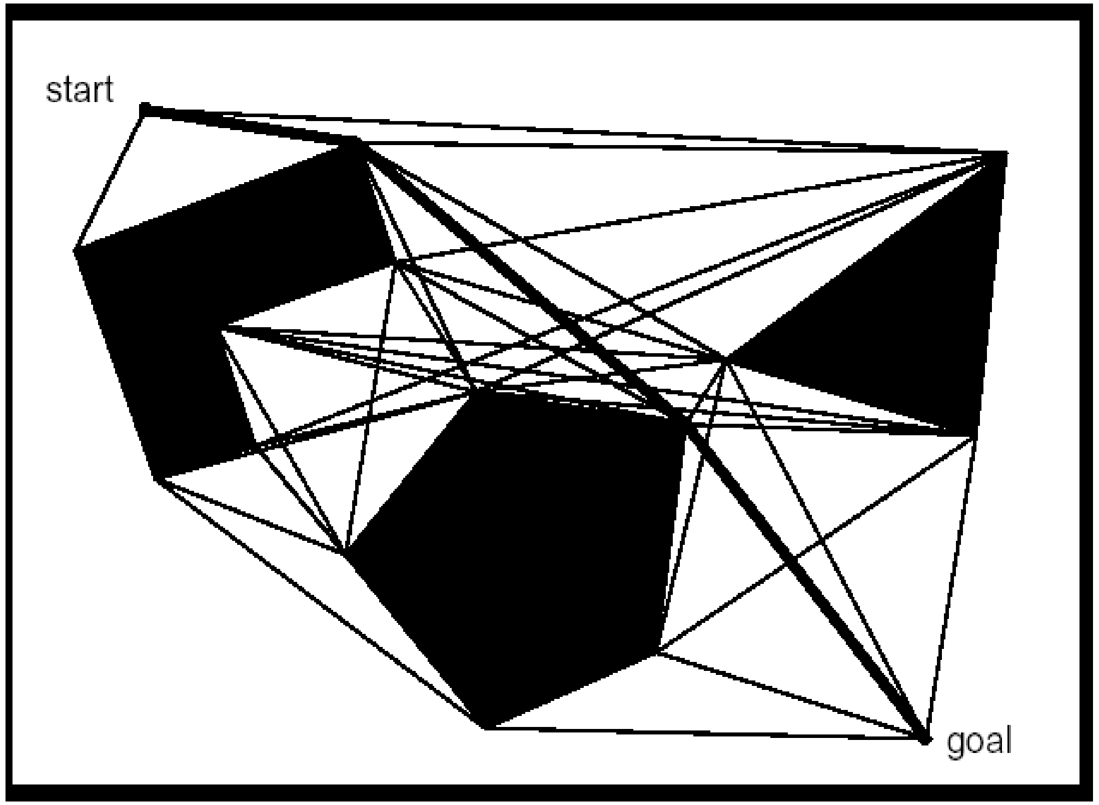
Potential Field Path Planning:

- Notes:
  - Local minima problem exists
  - problem is getting more complex if the robot is not considered as a point mass
  - If objects are non-convex there exists situations where several minimal distances exist $\rightarrow$ can result in oscillations

Graph Search

- Overview
  - Solves a least cost problem between two states on a (directed) graph
  - Graph structure is a discrete representation
- Limitations
  - State space is discretized → completeness is at stake
  - Feasibility of paths is often not inherently encoded
- Algorithms
  - (Preprocessing steps)
  - Breath first
  - Depth first
  - Dijkstra
  - A* and variants
  - D* and variants

Graph Construction: Visibility Graph



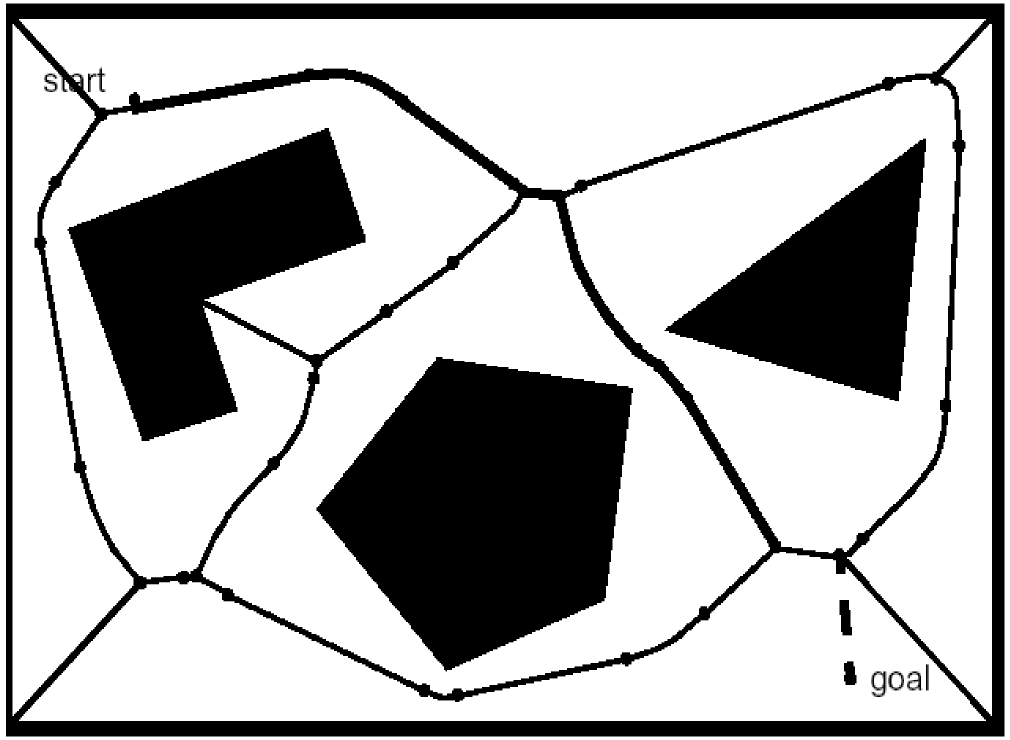
- Particularly suitable for polygon-like obstacles
- Shortest path length
- Grow obstacles to avoid collisions

Graph Construction: Visibility Graph

- Pros
  - The found path is optimal because it is the shortest length path
  - Implementation simple when obstacles are polygons

- Cons
  - The solution path found by the visibility graph tend to take the robot as close as possible to the obstacles: the common solution is to grow obstacles by more than robot's radius
  - Number of edges and nodes increases with the number of polygons
  - Thus it can be inefficient in densely populated environments

Graph Construction: Voronoi Diagram



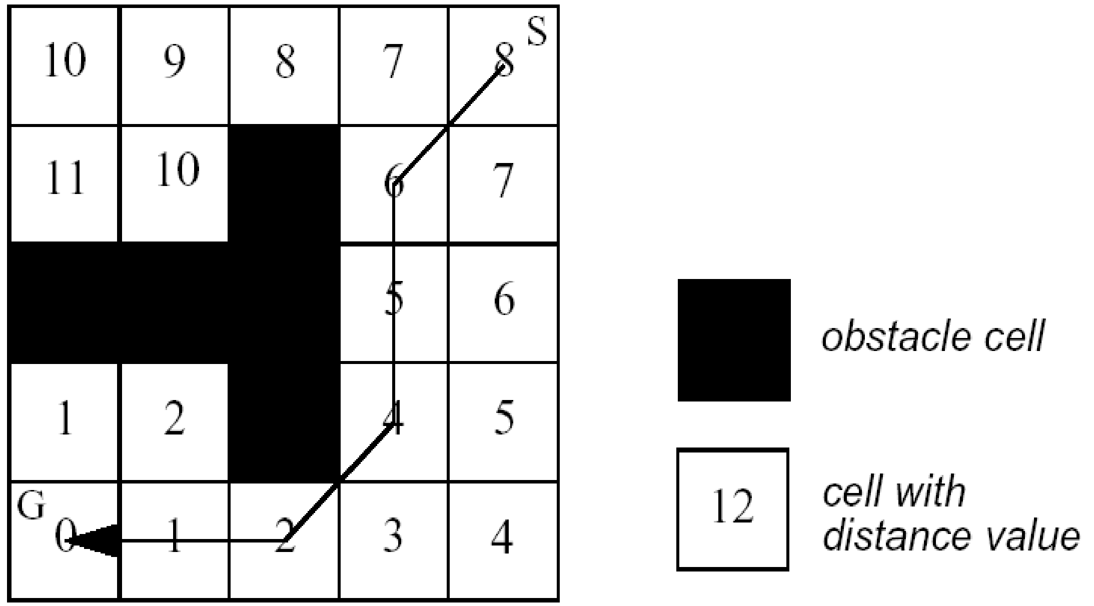
- Tends to maximize the distance between robot and obstacles

Graph Construction: Voronoi Diagram

- Pros
  - Using range sensors like laser or sonar, a robot can navigate along the Voronoi diagram using simple control rules
- Cons
  - Because the Voronoi diagram tends to keep the robot as far as possible from obstacles, any short range sensor will be in danger of failing
  - Voronoi diagram can change drastically in open areas

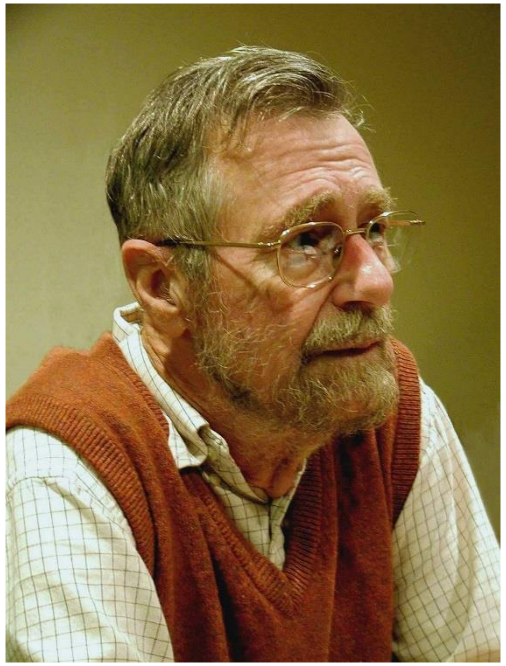
Deterministic Graph Search

- Methods
  - Breath First
  - Depth First
  - **Dijkstra**
  - A* and variants
  - D* and variants
  - ...



DIJKSTRA‘S ALGORITHM

EDSGER WYBE DIJKSTRA



1930 - 2002

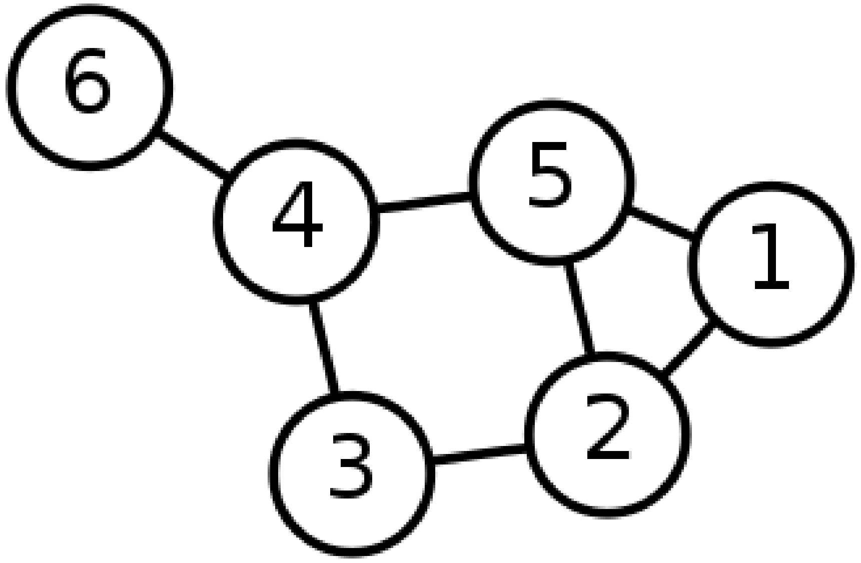
"Computer Science is no more about computers than astronomy is about telescopes."

http://www.cs.utexas.edu/~EWD/

SINGLE-SOURCE SHORTEST PATH PROBLEM

- **Single-Source Shortest Path Problem** - The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.

- **Graph**
- Set of vertices and edges
- Vertex:
  - Place in the graph; connected by:
- Edge: connecting two vertices
  - Directed or undirected (undirected in Dijkstra's Algorithm)
  - Edges can have weight/ distance assigned



Dijkstra material from http://www.cs.utexas.edu/~tandy/barrera.ppt

Diklstra's Algorithm

- Assign all vertices infinite distance to goal
- Assign 0 to distance from start
- Add all vertices to the queue

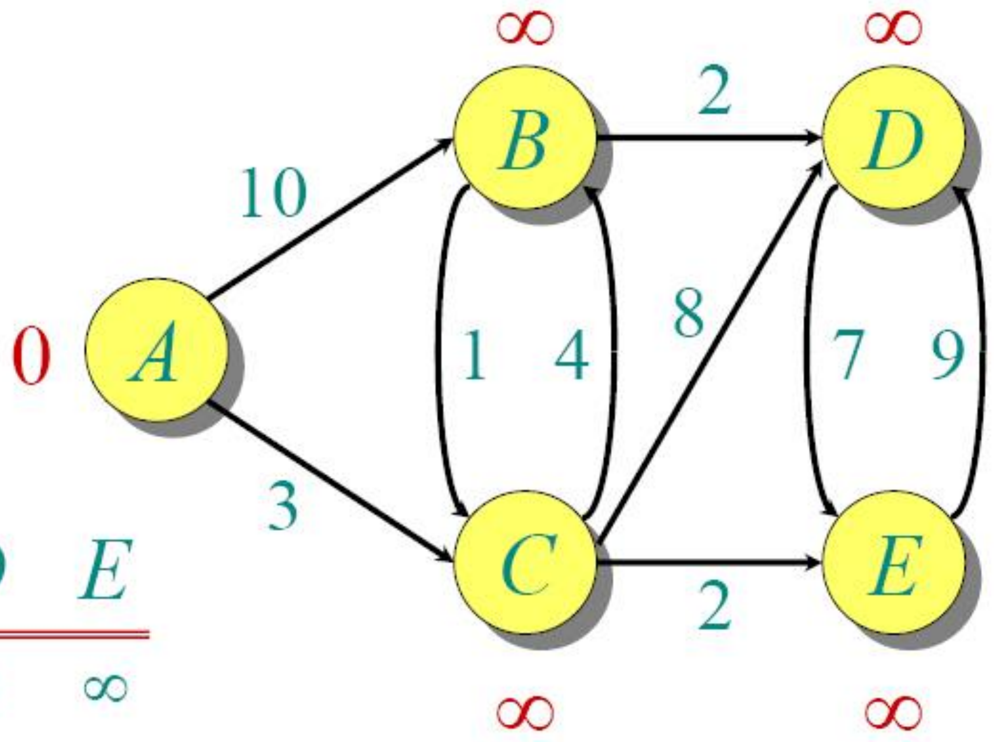
- While the queue is not empty:
  - Select vertex with smallest distance and remove it from the queue
  - Visit all neighbor vertices of that vertex,
  - calculate their distance and
  - update their (the neighbors) distance if the new distance is smaller

Diklstra’s Algorithm - Pseudocode

```
dist[s] ← 0                          (distance to source vertex is zero)
for  all v ∈ V–{s}
        do  dist[v] ← ∞              (set all other distances to infinity)
S ← ∅                                (S, the set of visited vertices is initially empty)
Q← V                                 (Q, the queue initially contains all vertices)
while Q ≠∅                           (while the queue is not empty)
do   u ← mindistance(Q, dist)        (select the element of Q with the min. distance)
      S←S∪{u}                        (add u to list of visited vertices)
      for all v ∈ neighbors[u]
            do  if   dist[v] > dist[u] + w(u, v)        (if new shortest path found)
                     then      d[v] ←d[u] + w(u, v)     (set new value of shortest path)
           (if desired, add traceback code)
return dist
```

Dijkstra Example

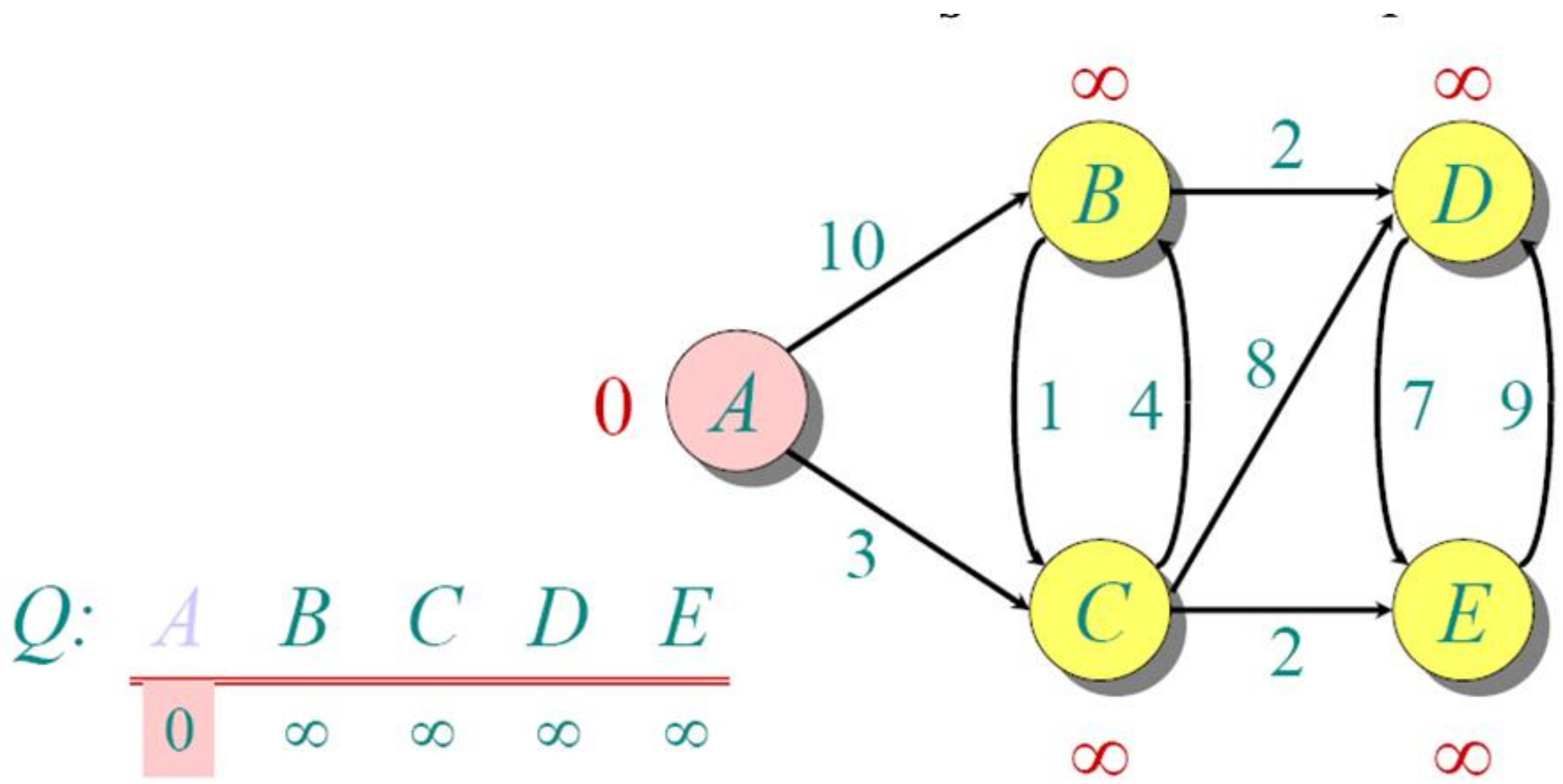
Initialize:



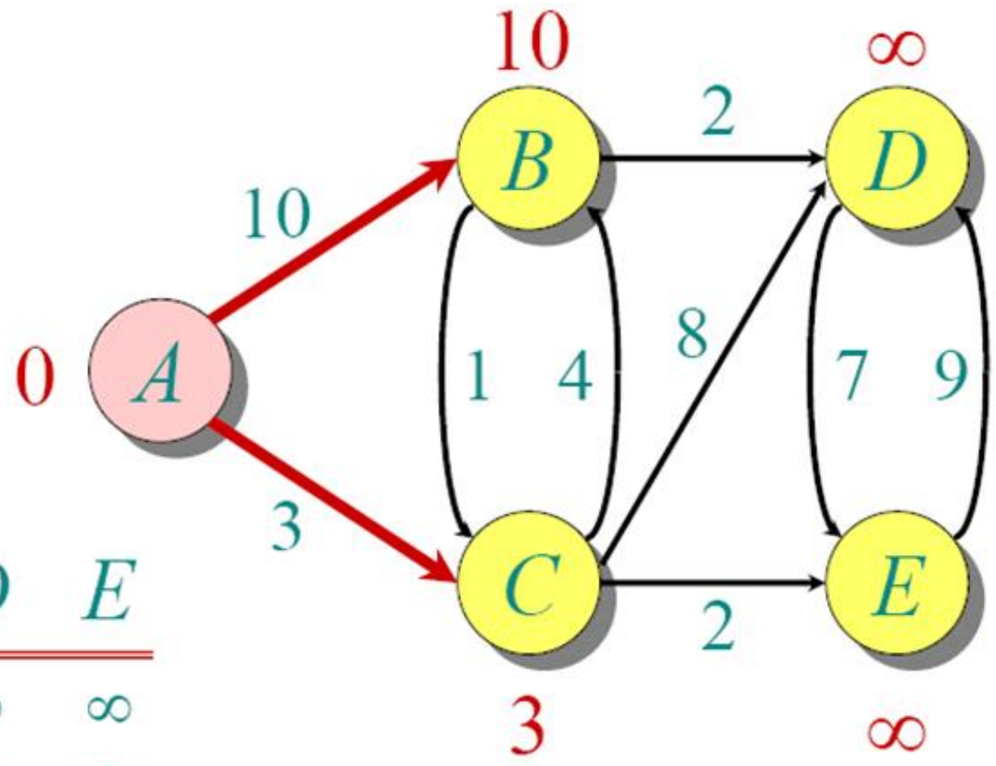
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

$S: \{\}$

Dijkstra Example



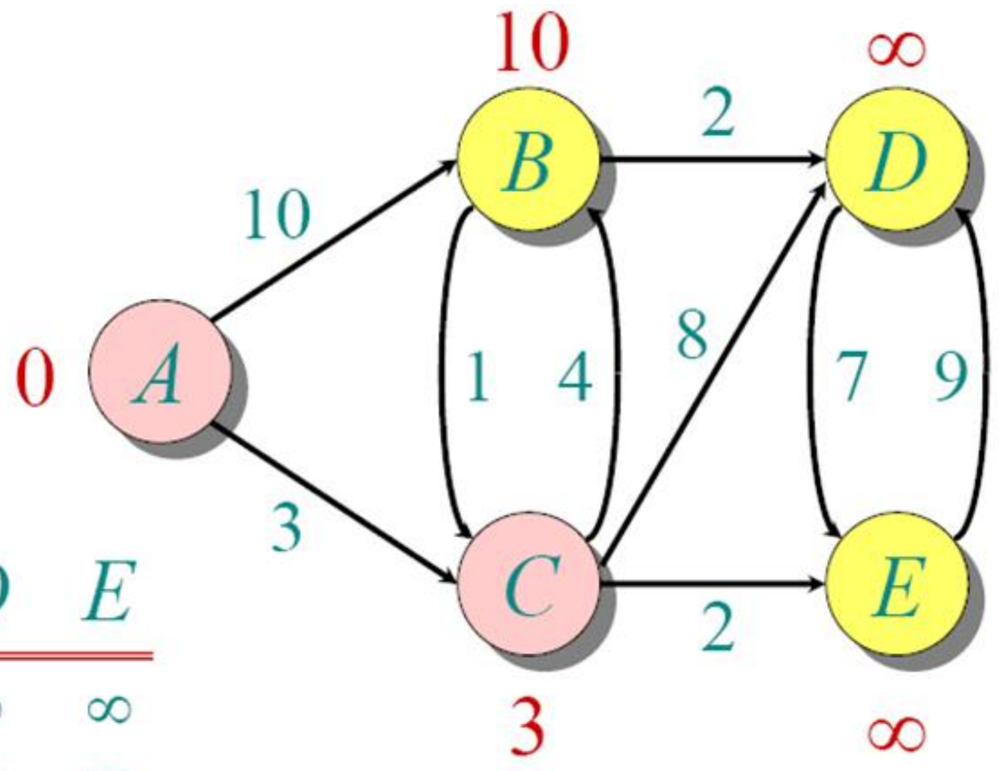
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

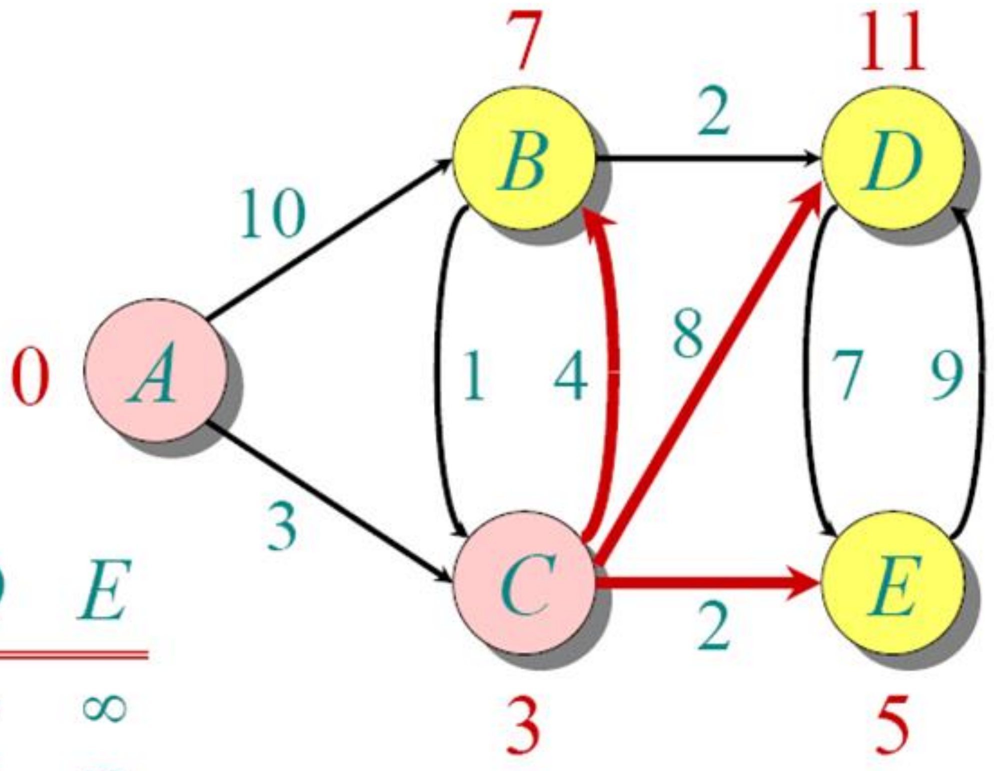
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

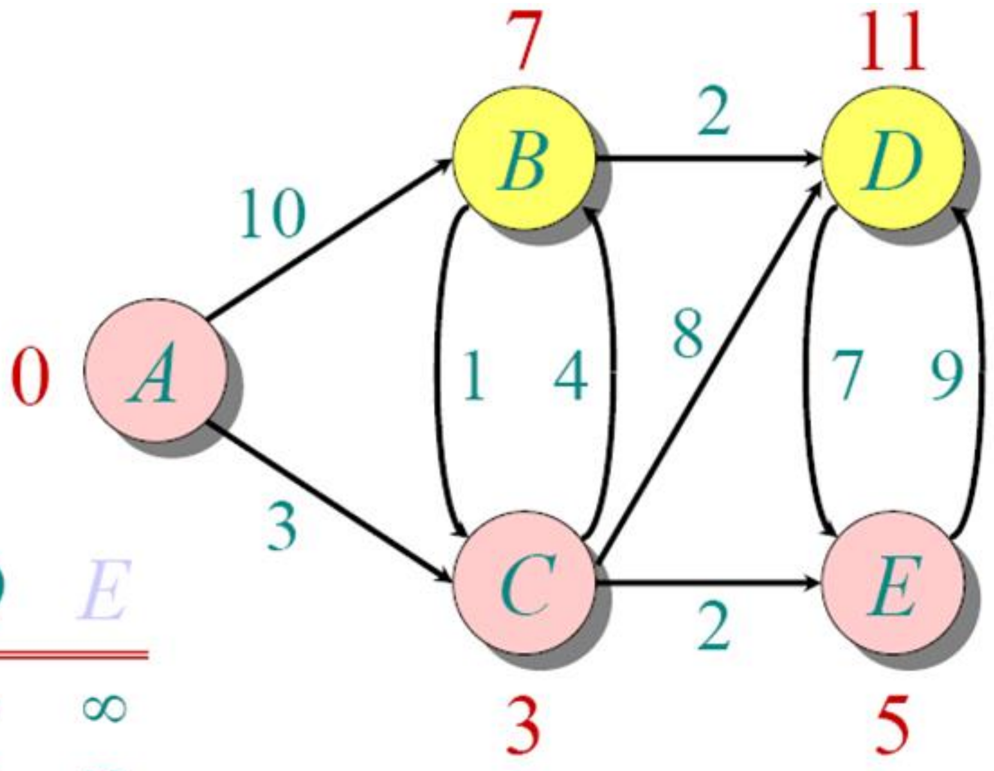
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

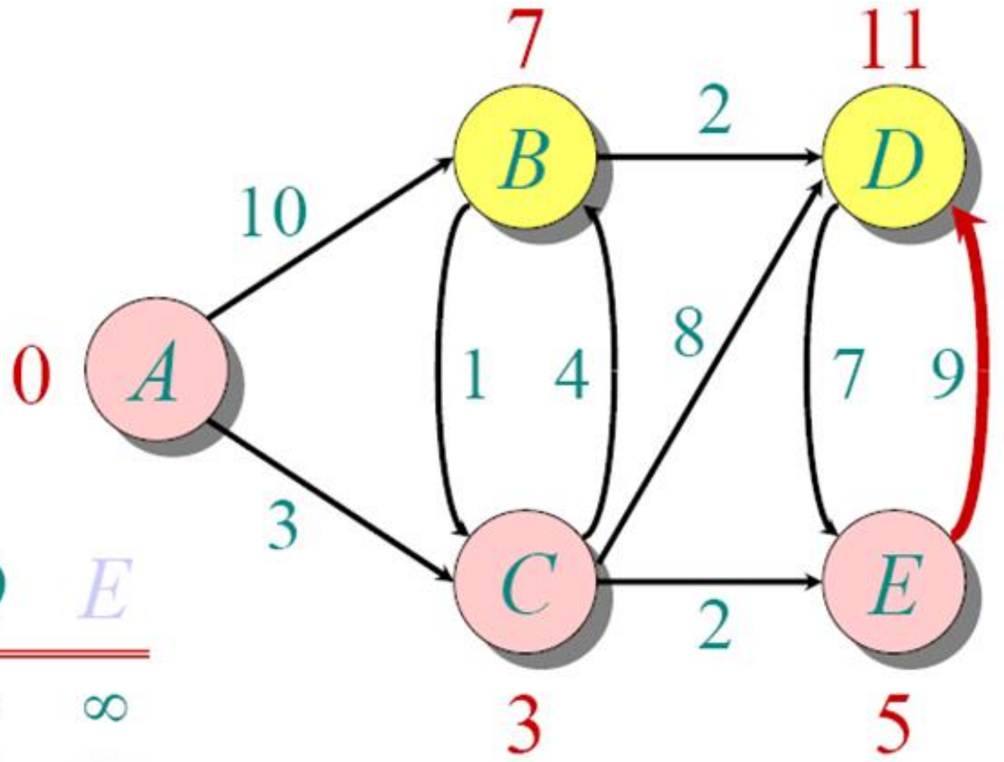
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

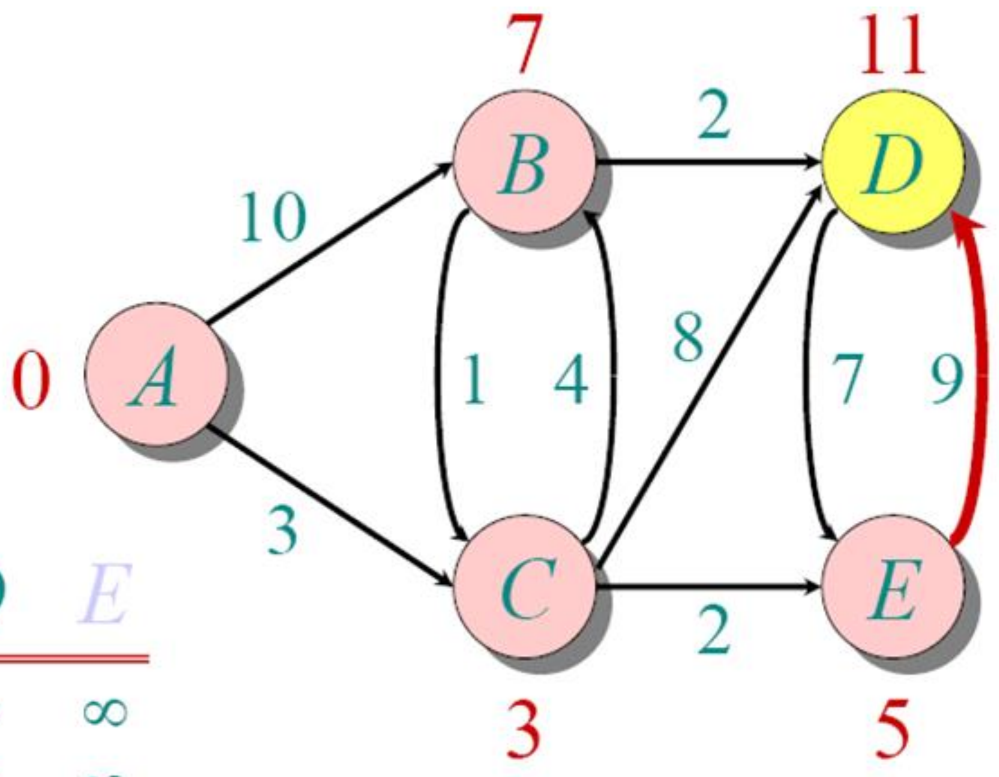
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E* }

Dijkstra Example

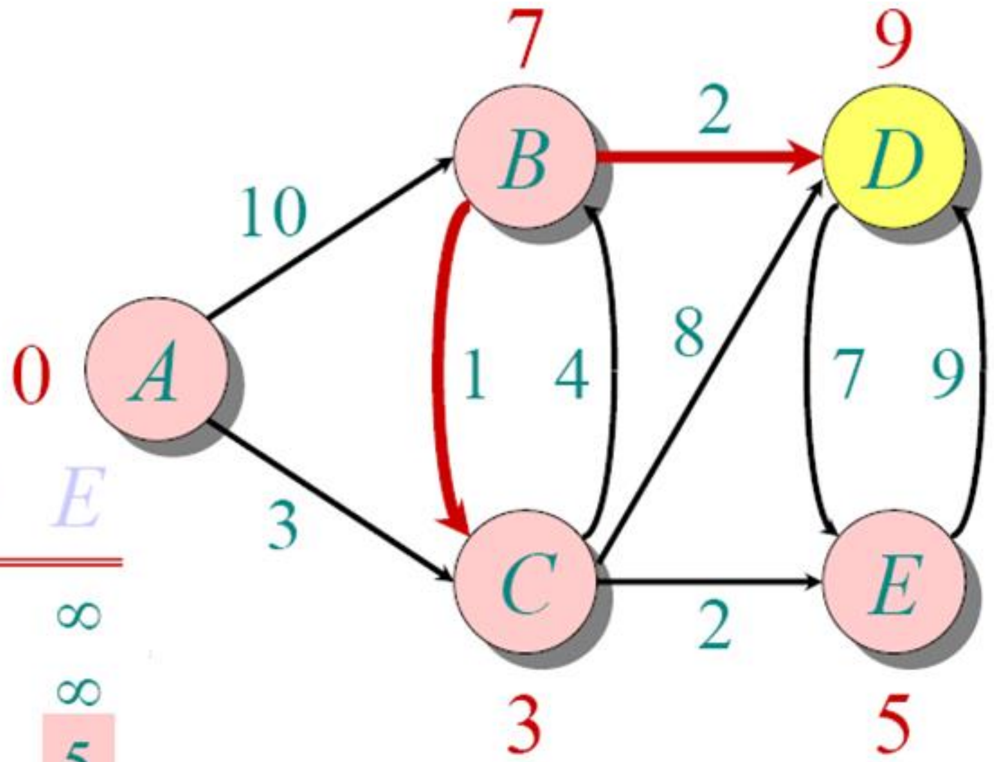


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

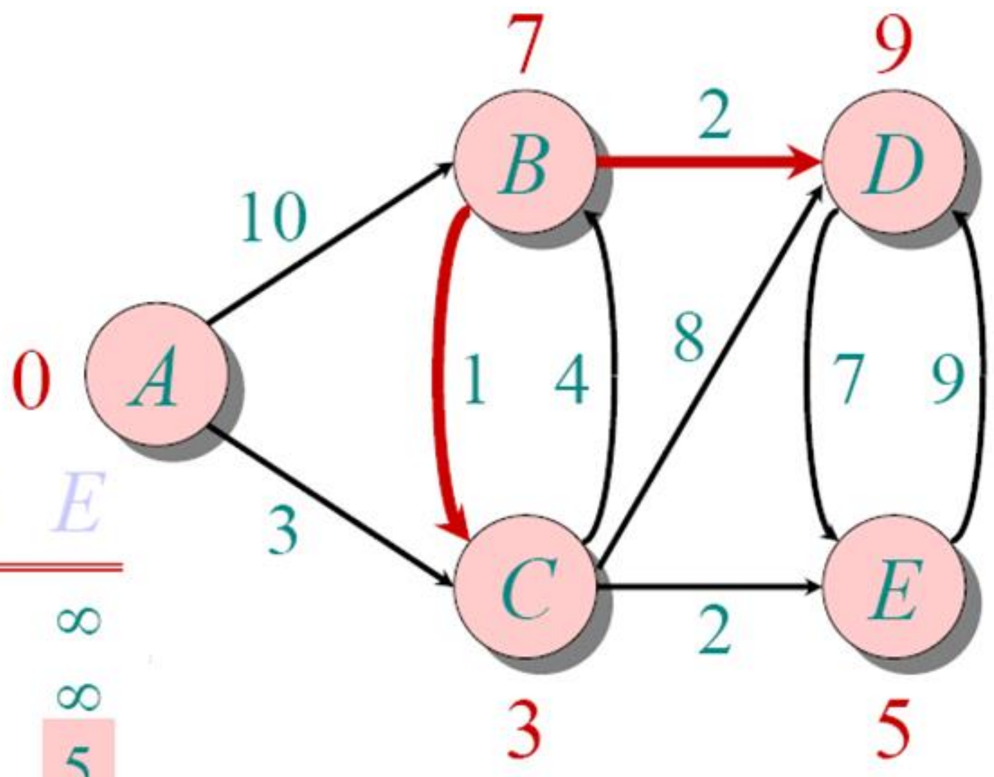
Dijkstra Example

Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	



S: { A, C, E, B }

Dijkstra Example

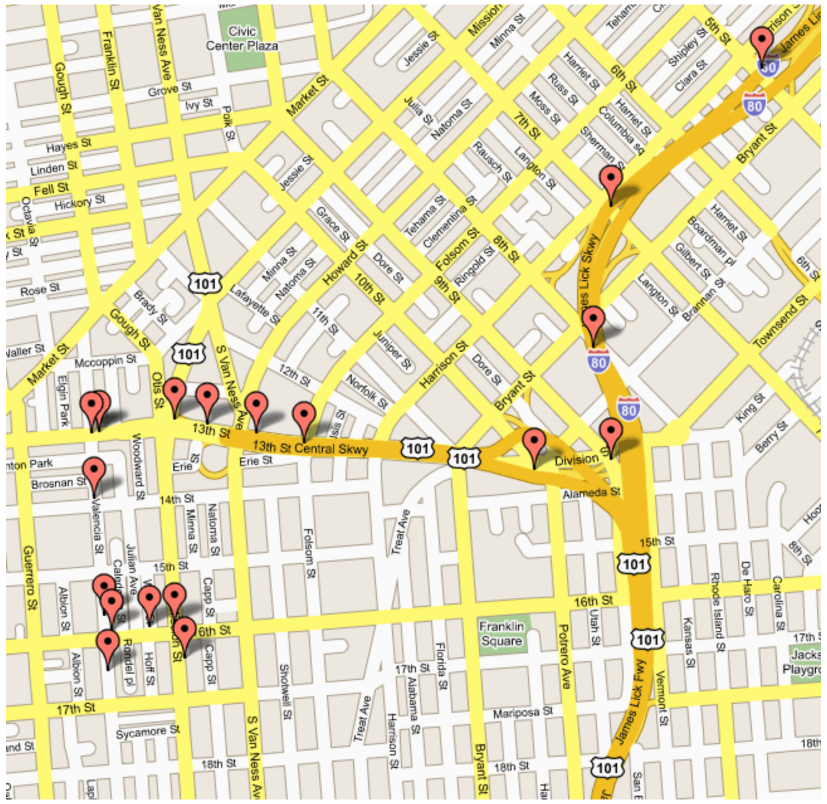


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

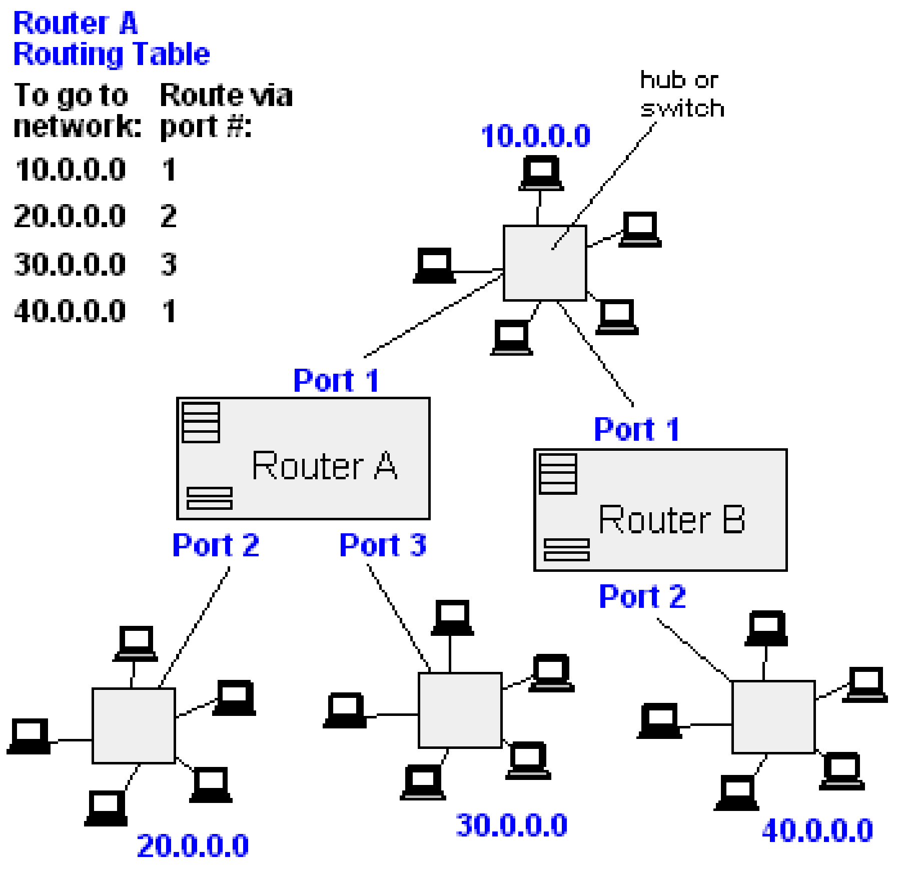
S: { *A*, *C*, *E*, *B*, *D* }

APPLICATIONS OF DIJKSTRA'S ALGORITHM

- Navigation Systems
- Internet Routing



From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.



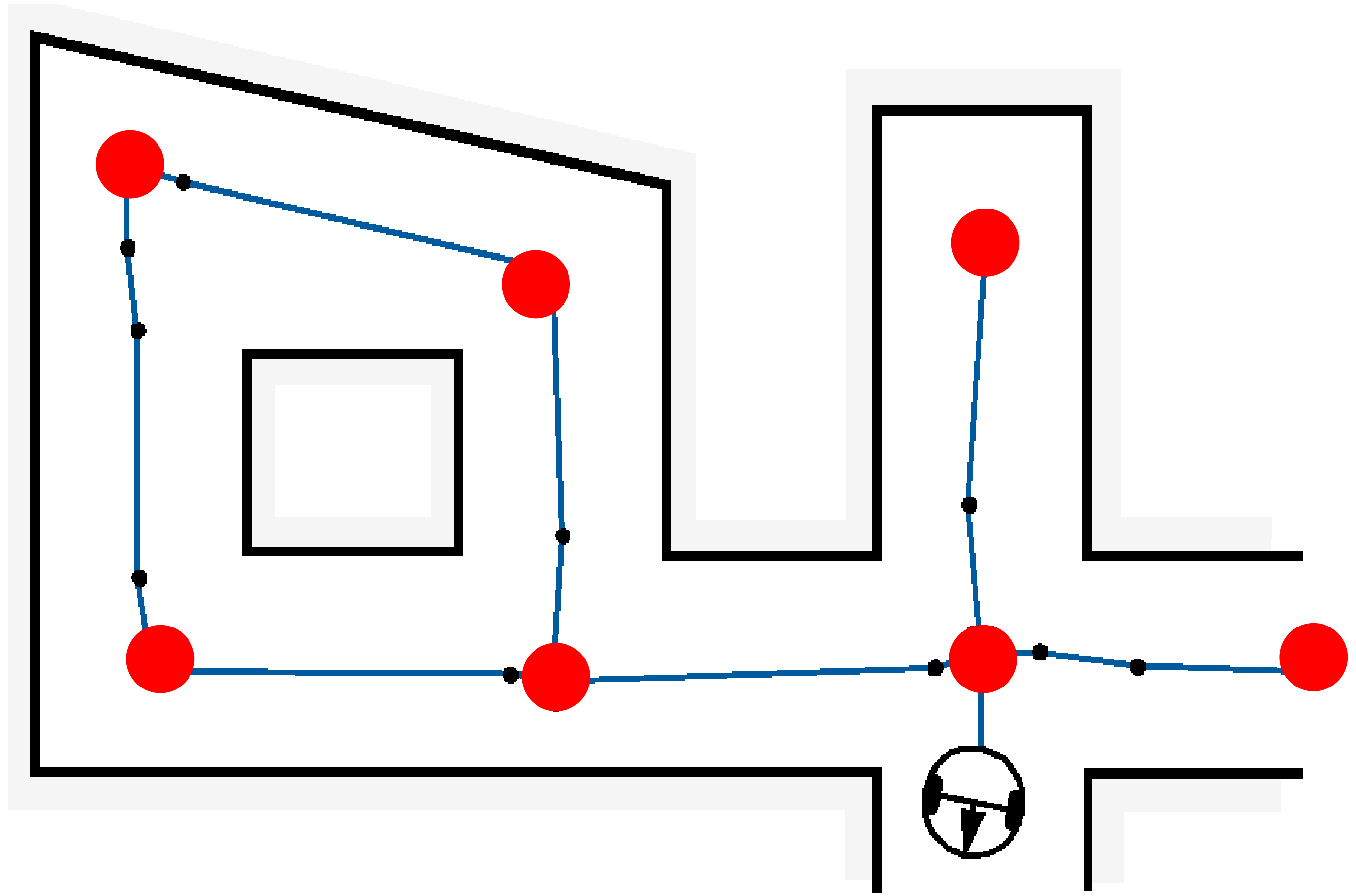
Router A Routing Table

To go to network:	Route via port #:
10.0.0.0	1
20.0.0.0	2
30.0.0.0	3
40.0.0.0	1

Dijkstra’s Algorithm for Path Planning: Topological Maps

- Topological Map:
  - Places (vertices) in the environment (red dots)
  - Paths (edges) between them (blue lines)
  - Length of path = weight o edge

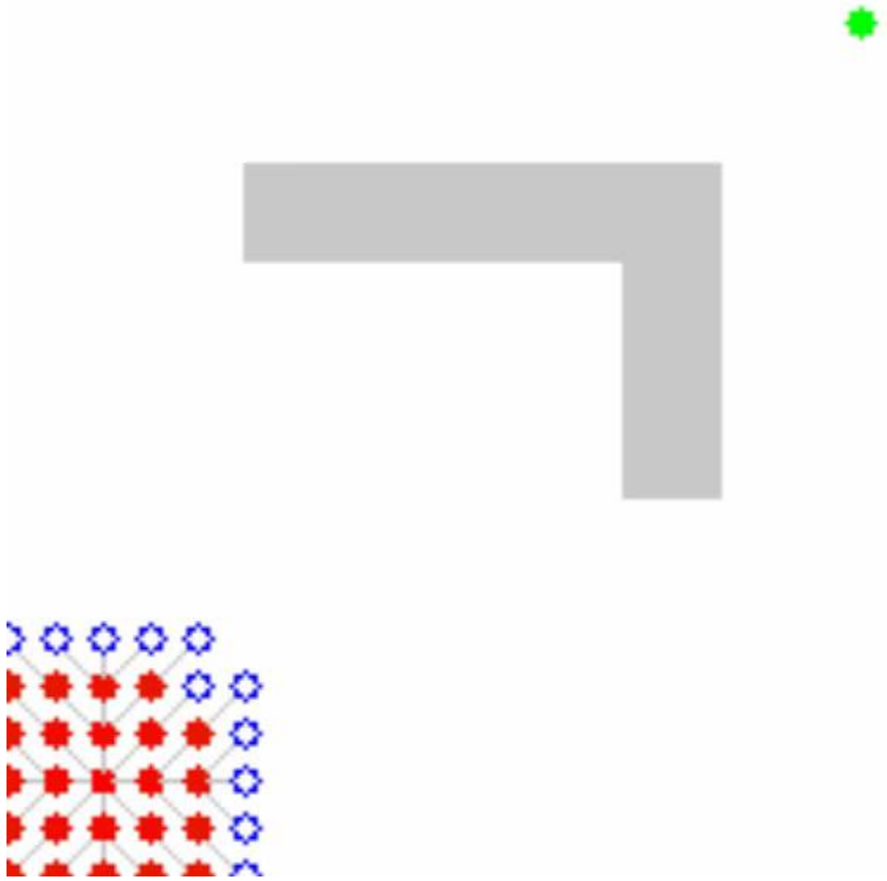
- => Apply Dijkstra’s Algorithm to find path from start vertex to goal vertex



Dijkstra’s Algorithm for Path Planning: Grid Maps

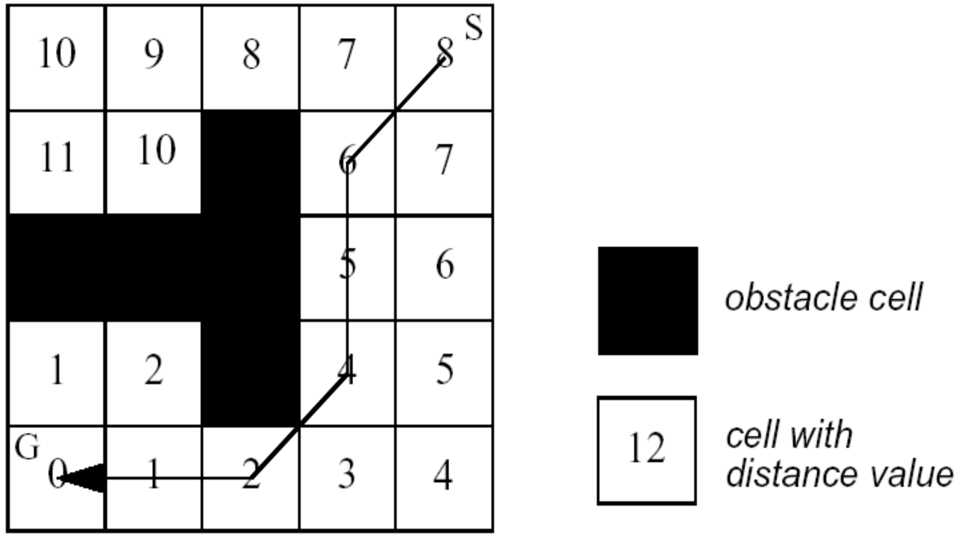
- Graph:
  - Neighboring free cells are connected:
    - 4-neighborhood: up/ down/ left right
    - **8-neighborhood**: also diagonals
  - All edges have weight 1

- Stop once goal vertex is reached
- Per vertex: save edge over which the shortest distance from start was reached => Path

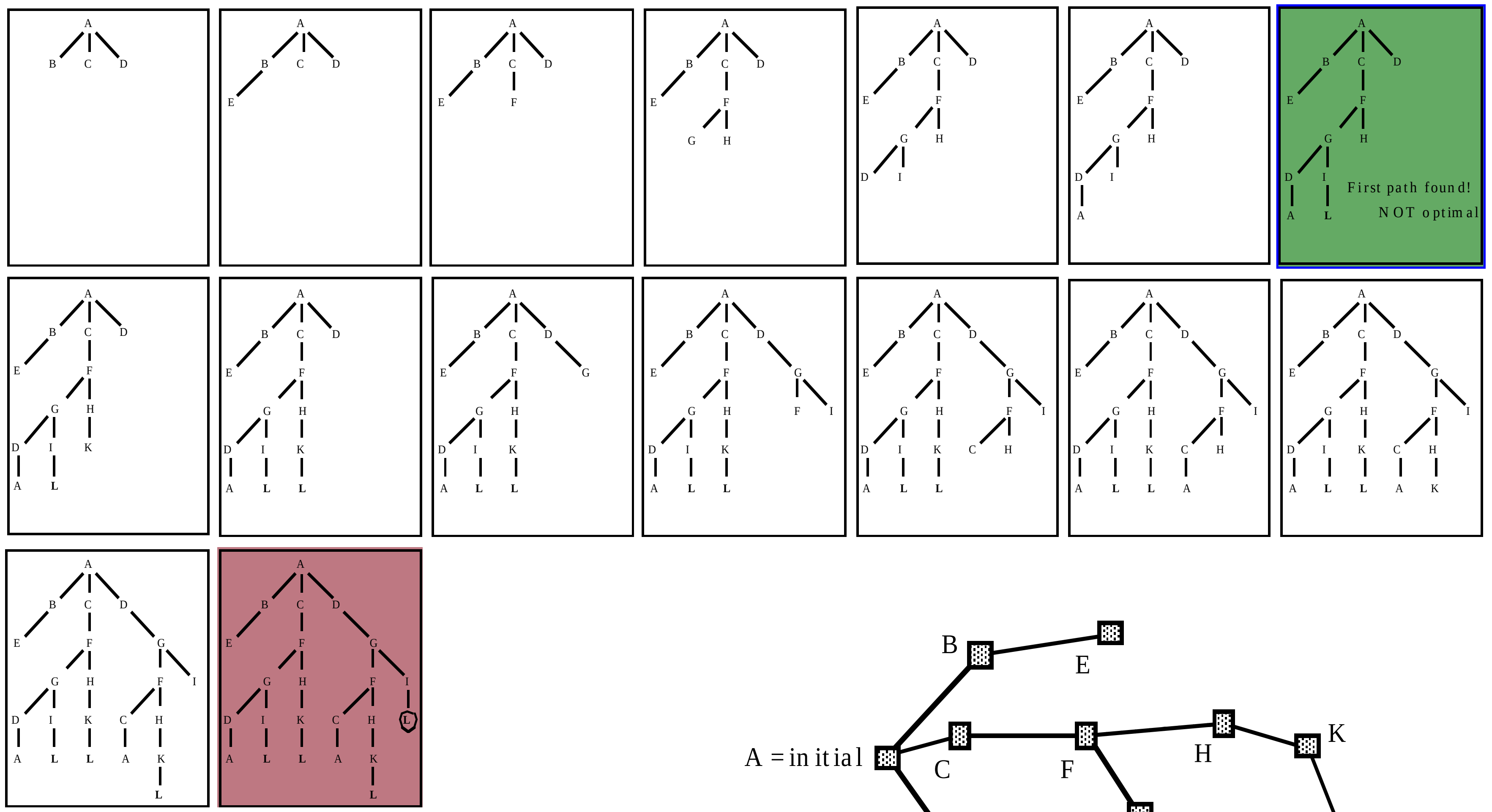


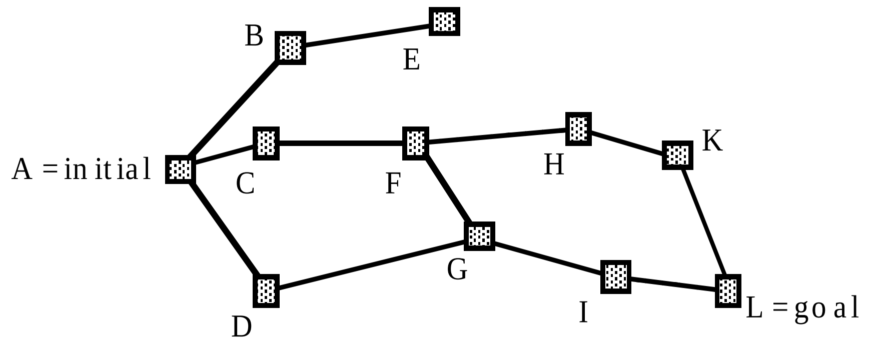
Graph Search Strategies: Breath-First Search

- Corresponds to a wavefront expansion on a 2D grid
- Breath-First: Dijkstra‘s search where all edges have weight 1



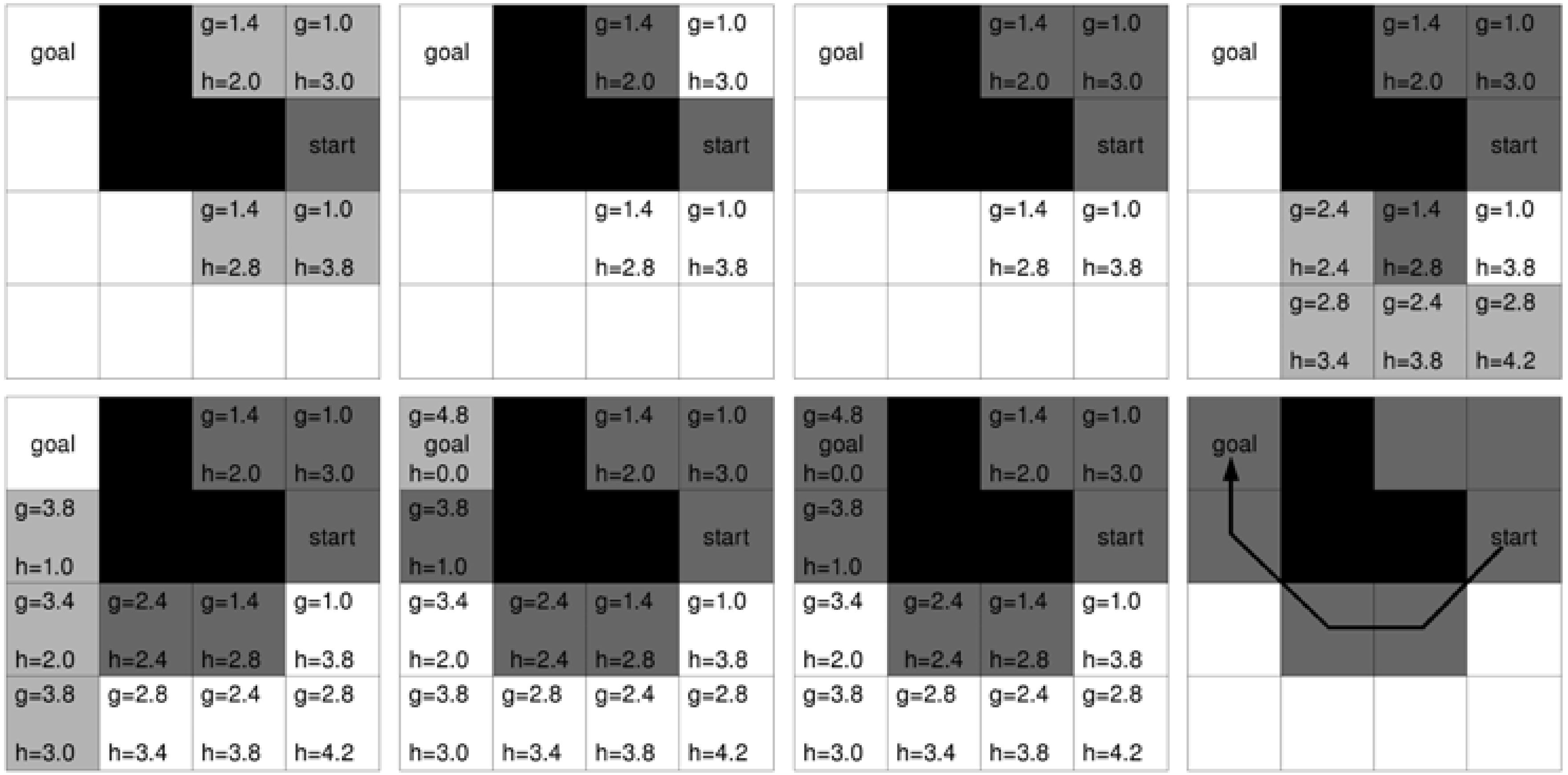
Graph Search Strategies: Depth-First Search





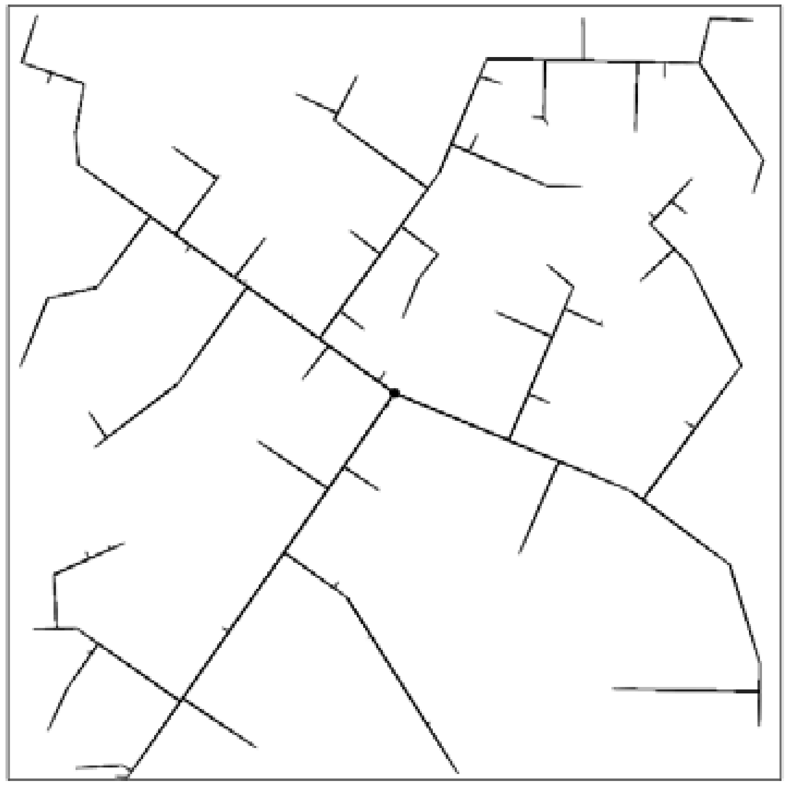
Graph Search Strategies: A* Search

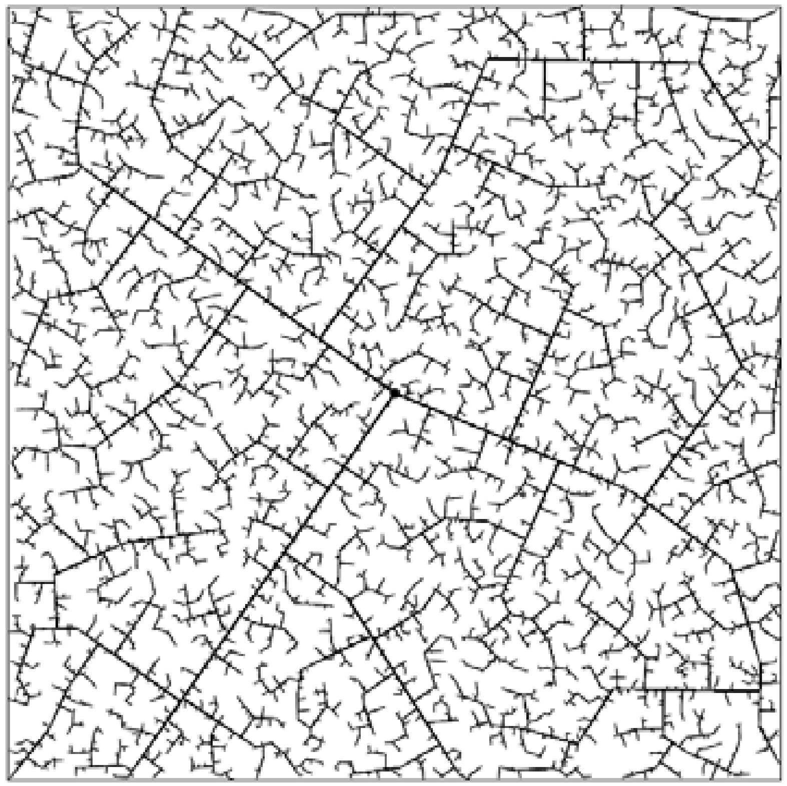
- Similar to Dijkstra‘s algorithm, except that it uses a heuristic function h(n)
- f(n) = g(n) + ε h(n)



Graph Search Strategies: Randomized Search

- Most popular version is the rapidly exploring random tree (RRT)
  - Well suited for high-dimensional search spaces
  - Often produces highly suboptimal solutions


45 iterations


2345 iterations