Planning in robotics of the path, motion, activity. . .

Václav Hlaváč

Czech Technical University in Prague

Czech Institute of Informatics, Robotics and Cybernetics

160 00 Prague 6, Jugoslávských partyzánů 3, Czech Republic

http://people.ciirc.cvut.cz/hlavac, vaclav.hlavac@cvut.cz

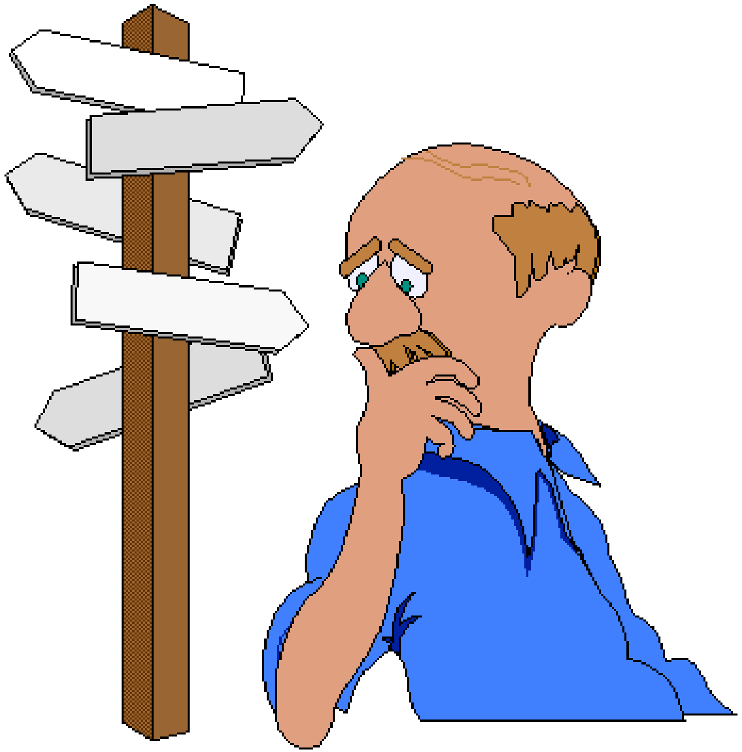
also Center for Machine Perception, http://cmp.felk.cvut.cz

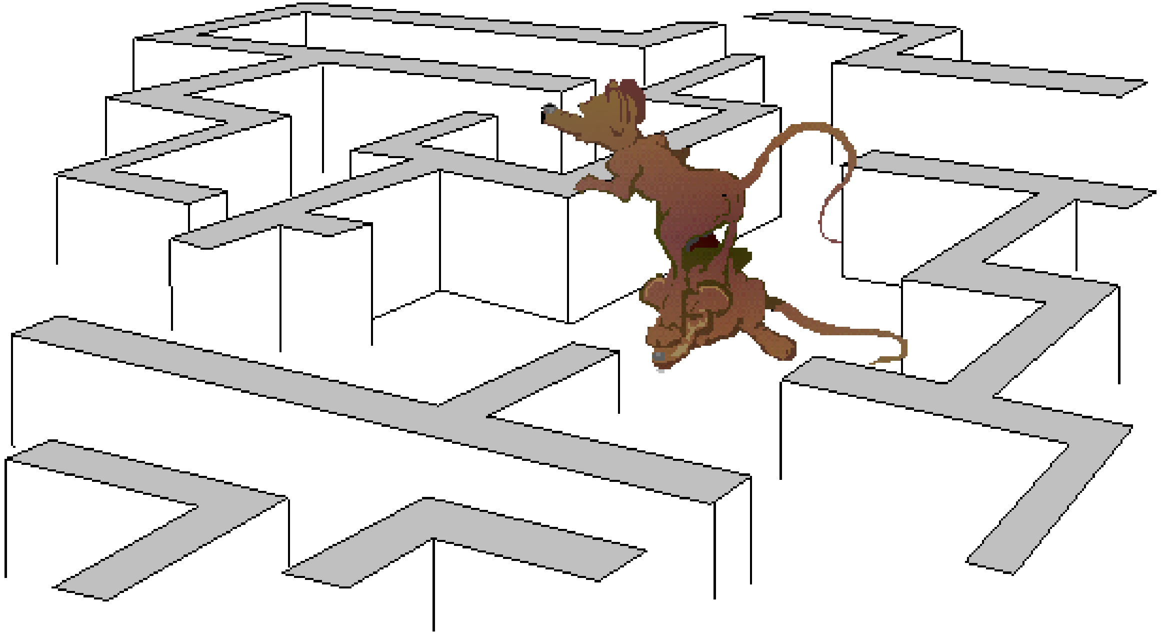
Outline of the talk:

- Holonomicity.
- Motion planning, formulation.
- Terminology, path vs. trajectory.
- Robotic planning as a spatial reasoning.
- Motion planning algorithms.
-

Motion planning in industrial and mobile robotics

Determining where to move without hitting obstacles.

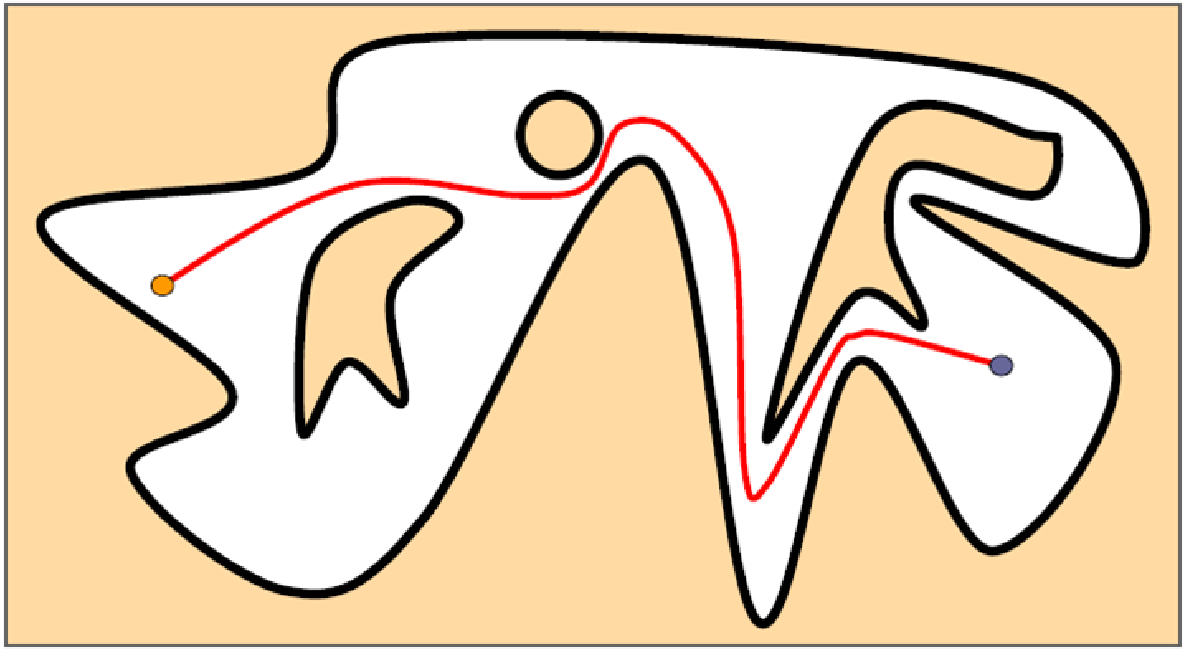




Three key questions in robotic planning

1. Where am I?
   Localization.
2. Where have I been?
   Mapping.
3. Where am I going?
   Planning.
4. How do I get there?
   Navigation.

Are two given points connected by a path?



Motivation task for motion planning

The motivation task

- The task:

  Transform the high-level task specification (provided by a human) into the low-level commands controlling the actuators.

- The solution:

  Motion planning algorithms provide the (geometric) path enabling to move a robot (or a manipulator gripper) from the start to the goal taking into account all operational constraints.

Asimo robot by Honda.

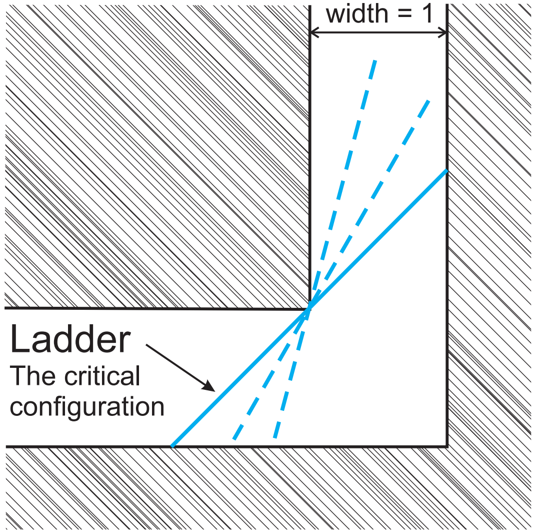
BMW spot welding.

Motion planning, the problem formulation

- **Motion planning** (a robotics term) is the process of breaking down a desired movement task into (discrete) motions satisfying given constraints (as not hitting obstacles, keeping speed limits) and possibly address optimality aspects.
- Known also as the navigation problem or piano mover's problem.
- The geometric aspect of the task (spatial reasoning) induces use of methods from computational geometry.

A computational geometry example:
The **Moving ladder problem**

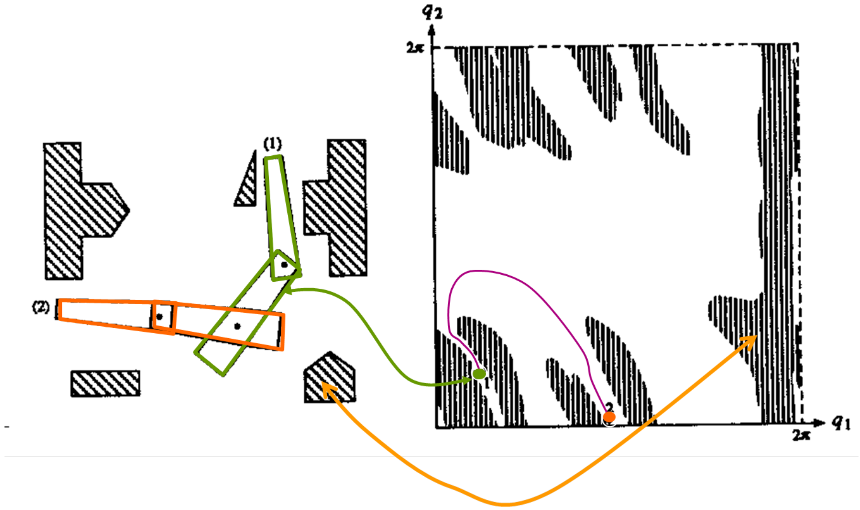
- What is the longest ladder that can be moved around a right-angled corridor of unit width?
- For a straight, rigid ladder, the answer is $2\sqrt{2}$, which allows the ladder to just pivot around the corner at a 45^o angle.



$\mathcal{C}$-space, a reminder

We studied the configuration space in the "robot world representation" lecture.

Consider a robot arm with two DOFs. The task is to move from the point (1) to the point (2) not touching the obstacles.



Euclidean (Cartesian) space $\mathcal{C}$-space

Piano mover's problem

- Given an open subset U (free space) in n-dimensional $\mathcal{C}-$space and two compact subsets C_0 (start) and C_1 (goal) of U, where C_1 is derived from C_0 by a continuous motion, is it possible to move C_0 to C_1 while remaining entirely inside U?
- A side effect of the algorithm: the 3D trajectory and the "piano" 3D configuration in any trajectory point.
- References

  - Buchberger, B., Collins, G. E., and Kutzler, B.: Algebraic Methods in Geometry. Annual Rev. Comput. Sci. 3, 85-119, 1988.
  - Finch, S. R.: Moving Sofa Constant, Section 8.12 in Mathematical Constants. Cambridge, England: Cambridge University Press, pp. 519-523, 2003.
  - Feinberg, E. B., Papadimitriou, C.: H. Finding Feasible Points for a Two-point Body, J. Algorithms 10, 109-119, 1989.
  - Leven, D., Sharir, M.: An Efficient and Simple Motion Planning Algorithm for a Ladder Moving in Two-Dimensional Space Amidst Polygonal Barriers, J. Algorithms 8, 192-215, 1987.

Piano mover's problem, a video example

Courtesy: Jan Faigl et al., The Czech Technical University in Prague

Piano mover's problem, a formal guarantee

- Given:
  - p – dimension of the configuration space, abbreviated $\mathcal{C}-$space.
  - m – number of polynomials describing $\mathcal{C}_{\text{free}}$.
  - d – Maximal degree of polynomials (in the preceding item).
- Theorem *(which is not very useful practically)*:
  A path (if it exists) can be found in time exponential in p and polynomial in m and d.
- J. Canny: The complexity of robot motion planning, MIT Ph.D. dissertation, 1987.

Terminology: path vs. trajectory

- *Note: Terms path, trajectory are often confused. They are used as synonyms informally.*
- **Path** is an ordered locus of points in the space (either joint or operational), which the robot should follow.
  - Path provides a pure geometric description of motion.
  - Path is usually planned globally taking into account obstacle avoidance, traversing a complicated maze, etc.
- **Trajectory** is a path plus velocities and accelerations in its each point.
  - A design of a trajectory does not need global information, which simplifies the task significantly.
  - The trajectory is specified and designed locally. Parts of a path are covered by individual trajectories.
  - It is often required that pieces of trajectories join smoothly, which induces that a single trajectory design takes into account only neighboring trajectories from the path.

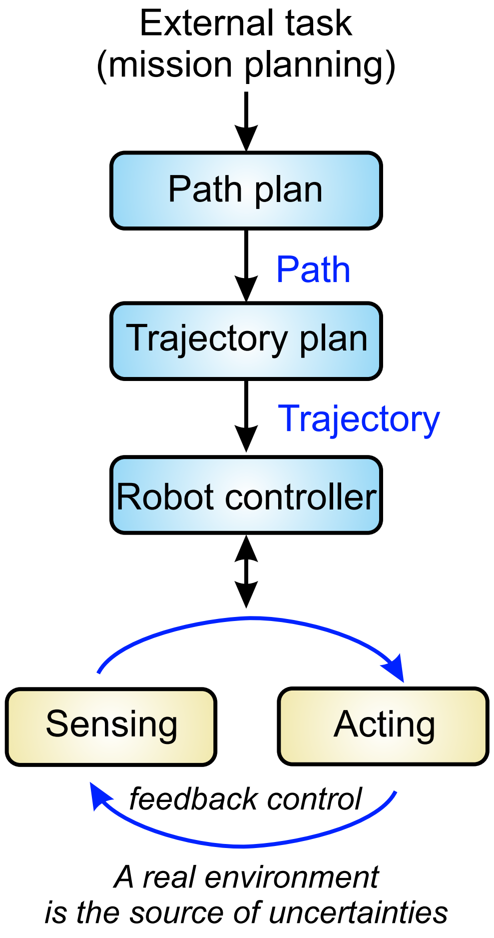
Robot motion planning, an overview

Path planning (global)

- Geometric path.
- Issues: obstacle avoidance, shortest path.

Trajectory generating (local)

- The path planning provides the input – the chunk of a path usually given as a set of points defining the trajectory.
- "Approximate" the desired path chunk by a class of polynomial functions and
- generate a sequence of time-based "control set points" for the control of manipulator from the initial configuration to its destination.



Path planning framework

1. Continuous robot world/workspace representation.
   - Represented often in configuration space or operational space.
   - Represent related constraints as obstacles, minimal curvature of the path. It is more complicated with dynamic constraints.
2. Discretization.
   - Deterministic discretization as the occupancy grid.
   - Random sampling.
   - Critical geometric events and their representation.
3. Path finding by graph searching.
   - Breath-first.
   - A*
   - Approximation methods, etc.

Problem solving vs. planning

Basic problem solving

- Problem solving (search in a state- space, a basic tool in AI) and planning have a similar core. However, they are considered different.
- Basic problem solving searches a state-space of possible actions, starting from an initial state and following any path to the goal state.

Planning differs from the basic problem solving in:

1. Planning "opens up" the representation of states, goals and actions so that the planner can deduce direct connections between states and actions.
2. The planner does not have to solve the problem in order. It can suggest actions to solve any sub-goals at any time.
3. Planners assume that most parts of the world are independent. Decomposition to subproblems into practically sized chunks simplifies the solution considerably.

Path planning (dealt in this lecture)

- Goals:
  - **Achieve high-level goals**, e.g.:
    Assemble/disassemble the engine. Build a map of the hallway. Find a collision free path for the robot from one configuration to another configuration.
  - **Compute motion strategies**, e.g.:
    Geometric paths; Sequence of sensor-based motion commands. Time-parameterized trajectories.
- Path planning is a difficult search problem.
  - The involved task has an exponential complexity with respect to the degrees of freedom (controllable joints).
  - With industrial robots, path planning has been often solved by human operators showing (teaching in) the desired paths. Recently, automatic planning has been used more often.

Trajectory generating (covered in a separate lecture)

- Planned path is typically represented by via-points.
  - Via-points = sequence of points (or end-effector poses) along the path.
- Trajectory generating = creating a trajectory connecting two or more via points.
  - Trajectory generating approximates / interpolates the path.
  - In industrial settings, a trajectory is performed by a human expert and later played back (by teach-and-playback).
  - Recent research utilizes as the input several tens of trajectories performed by human experts. They vary statistically.
  - Machine learning techniques are used to create the final trajectory.

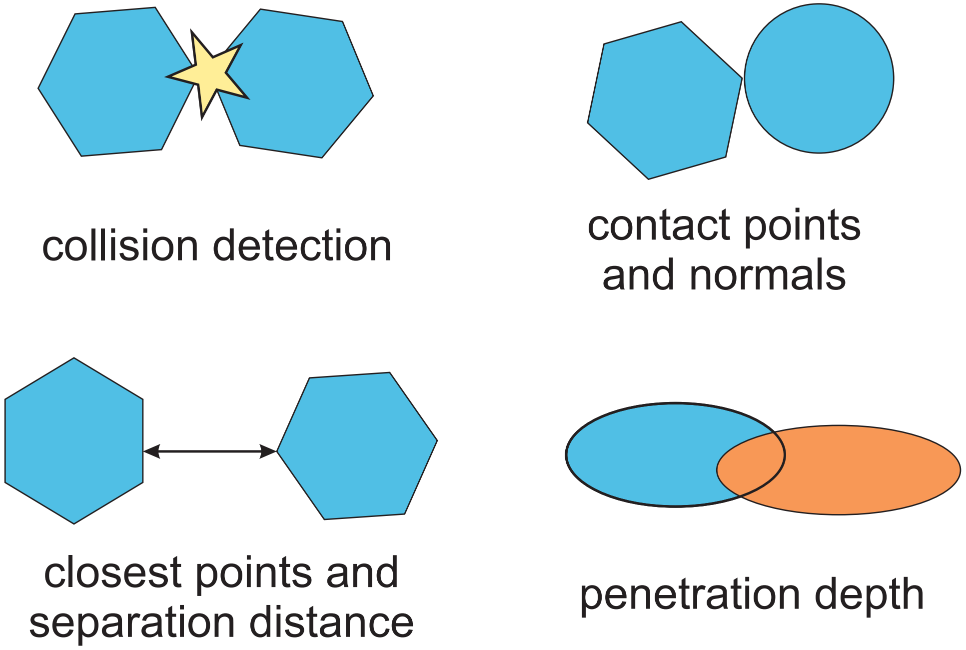
Robotic planning as spatial reasoning

- Application of earlier search approaches from artificial intelligence. (A*, stochastic search, etc.)
- Search in geometric structures ⇒ **Spatial reasoning**.
- A more complex variant considering time: **Spatial-temporal reasoning**.
- Challenges:
  - Continuous state space.
  - Large dimensional space.

- **The main strategy in motion planning**:
  - Reduction to point robot.
  - Configuration space.
  - Solution: convert to a search problem, usually the graph search.

Collision and proximity queries

require geometric reasoning of spatial relationships among objects, often in a dynamic environment.

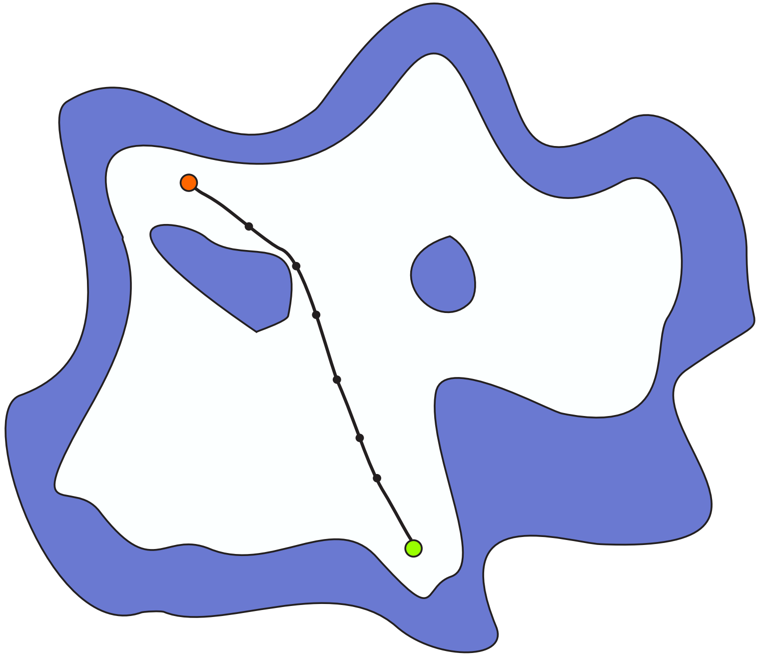


Collision and proximity computations

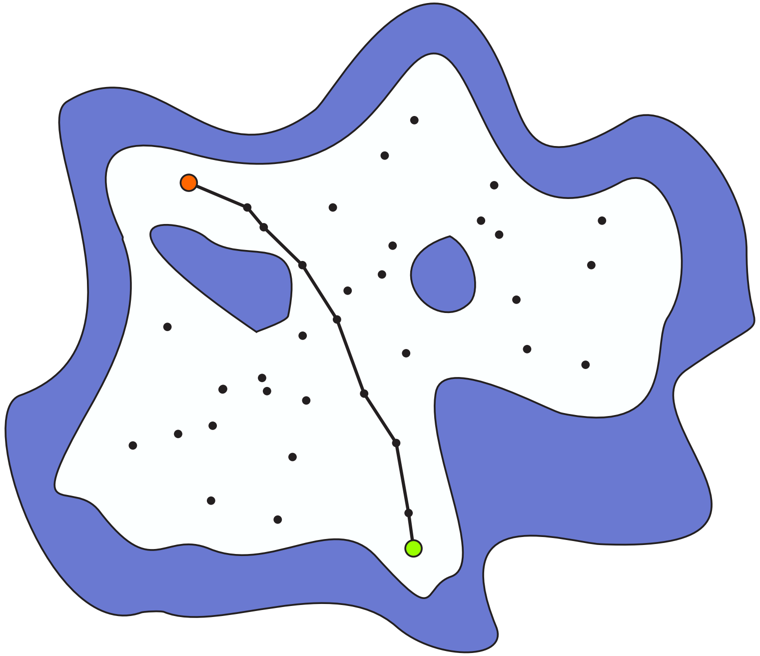
- A key component of motion planning algorithms (estimated 90% of a total run time).
- Widely used in CAD/CAM, simulation and virtual prototyping.
- Supported in robot simulation and CAD systems
- Studied in academia for 30+ years.
- Widely used recent implementations:
  - FCL (The Flexible Collision Library, University of Northern Carolina, Chapel Hill).
  - MoveIt! (part of ROS).

Motion planning algorithms, two main groups

Optimization-based algorithms



Random sampling-based algorithms



The green circle denotes the start. The orange circle denotes the goal.

Deterministic motion planning methods

Global approaches

- Road-map [Nilsson, 1969], [Jorgensen et al., 1986]
- Cell decomposition [Chazelle, 1987]
- Potential field [Khatib, 1986]

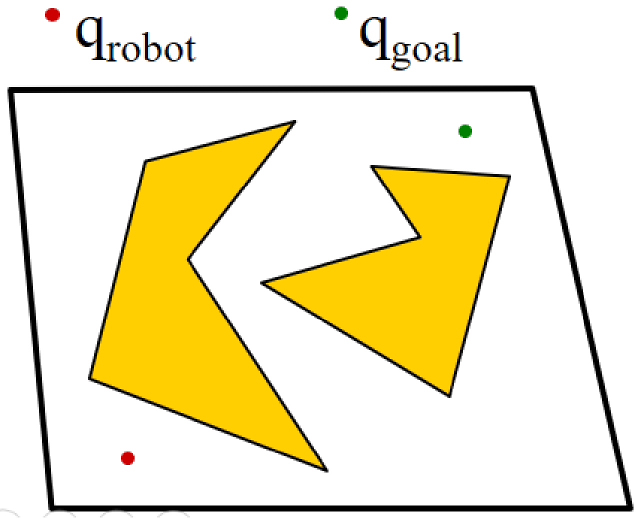
Local or reactive approaches

- Bug algorithm [Lumelsky, 1990]
- Vector field histogram [Borenstein and Koren, 1991]
- Histogramic in motion mapping [Borenstein and Koren, 1991]
- Dynamic window [Fox et al., 1997]

Planning: input, output, applications

Input

- Geometrical representation of a robot and its environment (e.g. obstacles).
- Initial and goal configurations.



Output

A path from start to finish (or the recognition that none exists).

Applications

- Selfdriving car, robot plans.
- Automated assembly plans.
- Robot-assisted surgery.
- Molecule docking and its analysis.
- Moving pianos around ...

Connection to next slides

Note to students:

- The following slides are taken from my older PowerPoint presentation, which I compiled from several presentations of other authors/teachers.
- I amended these slide to my LaTeX presentation at the pdf level. That is the reason why the numbering starts wrongly from 1 again.
- I intend to include/rewrite these slides into my LaTeX presentation.

Motion planning methods

- **Global approaches**
  - Road-map [Nilsson, 1969], [Jorgensen et al., 1986]
  - Cell decomposition [Chazelle, 1987]
  - Potential field [Khatib, 1986]

- **Local or reactive approaches**
  - Bug algorithm [Lumelsky, 1990]
  - Vector field histogram [Borenstein and Koren, 1991]
  - Histogrammic in motion mapping [Borenstein and Koren, 1991]
  - Dynamic window [Fox et al., 1997]

Task input, output

Input:

- Geometric descriptions of a robot and its environment (obstacles).
- Initial and goal configurations.

Output:

- A path from start to finish (or the recognition that none exists).

Applications:

- Robot-assisted surgery.
- Automated assembly plans.
- Mobile robot plans.
- Drug-docking and analysis.
- Moving pianos around …

Taxonomy of methods

1. Roadmap approaches.

Goal: Reduce the *N*-dimensional configuration space to a set of 1-dimensional paths to search.

2. Cell decomposition.

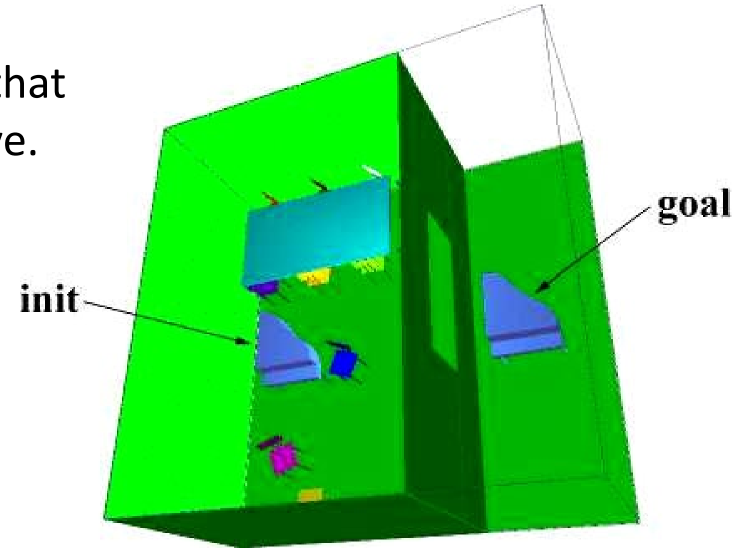
Goal: Account for all of the free space.

3. Potential fields.

Goal: Create local control strategies that will be more flexible than those above.

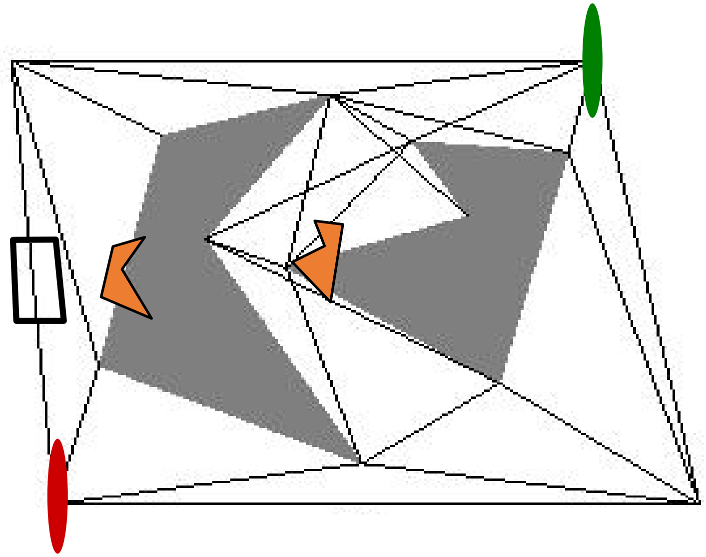
4. Bug algorithm.

Limited knowledge path planning.



Roadmap: Visibility graphs

- **Visibility graphs:** In a polygonal (or polyhedral) configuration space, construct all of the line segments that connect vertices to one another (and that do not intersect the obstacles themselves).
- Formed by connecting all “visible” vertices, the start point and the end point, to each other.
- For two points to be “visible”, no obstacle can exist between them.
- Paths exist on the perimeter of obstacles.



- From Cfree, a graph is defined.
- Converts the problem into graph search.

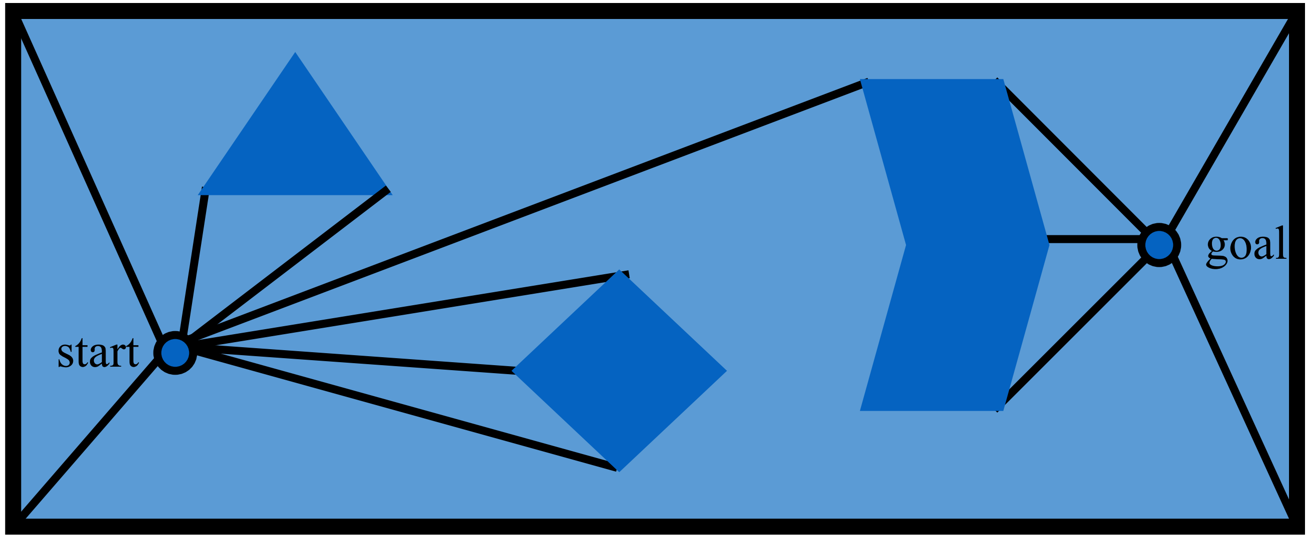
↓

Dijkstra’s algorithm $O(N^2)$

N = the number of vertices in the C-space

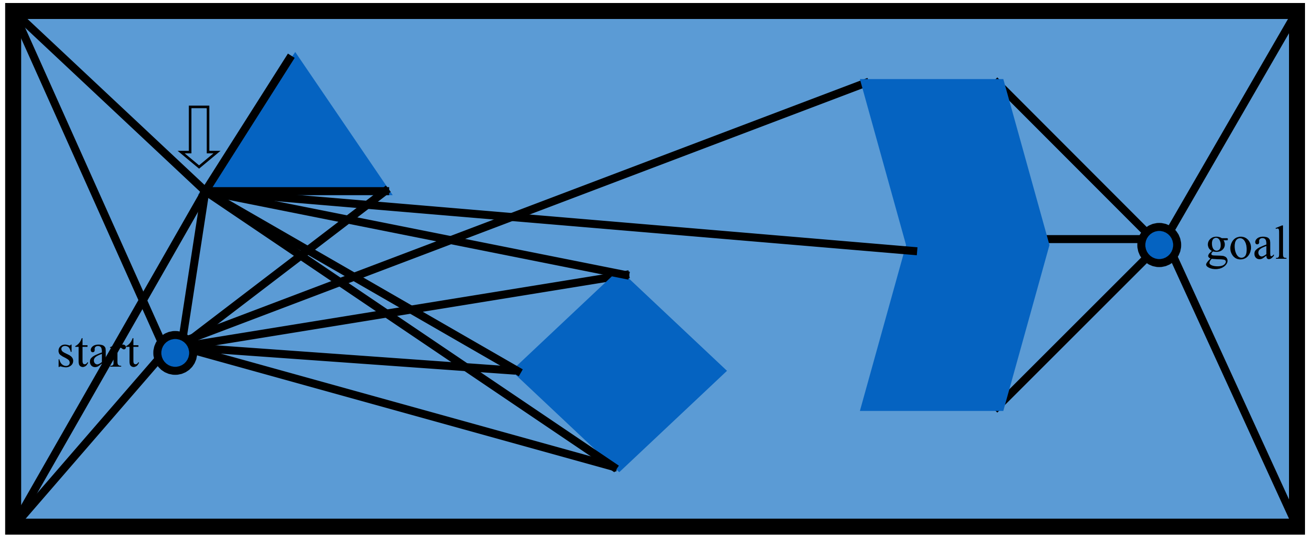
Visibility graph in action 1

- First, draw lines of sight from the start and goal to all “visible” vertices and corners of the world.



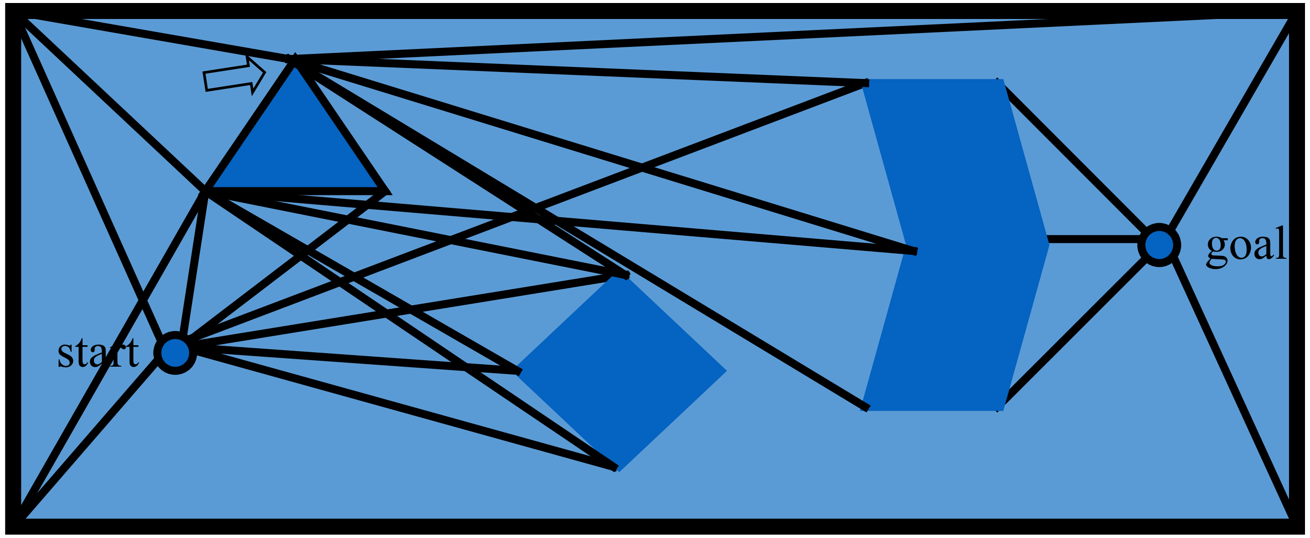
Visibility graph in action 2

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



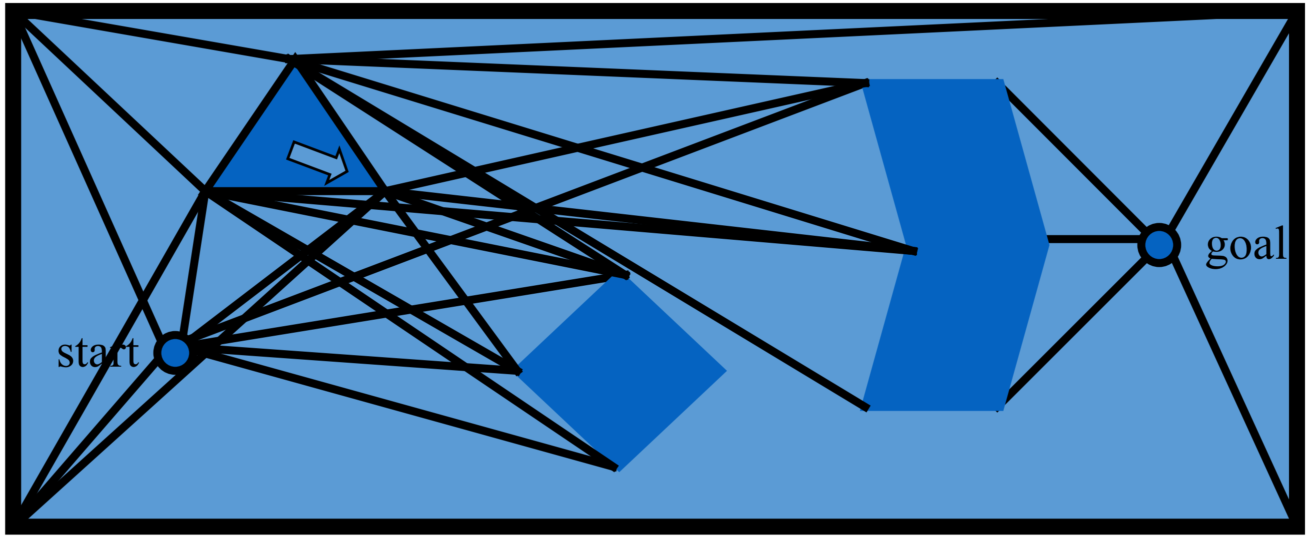
Visibility graph in action 3

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



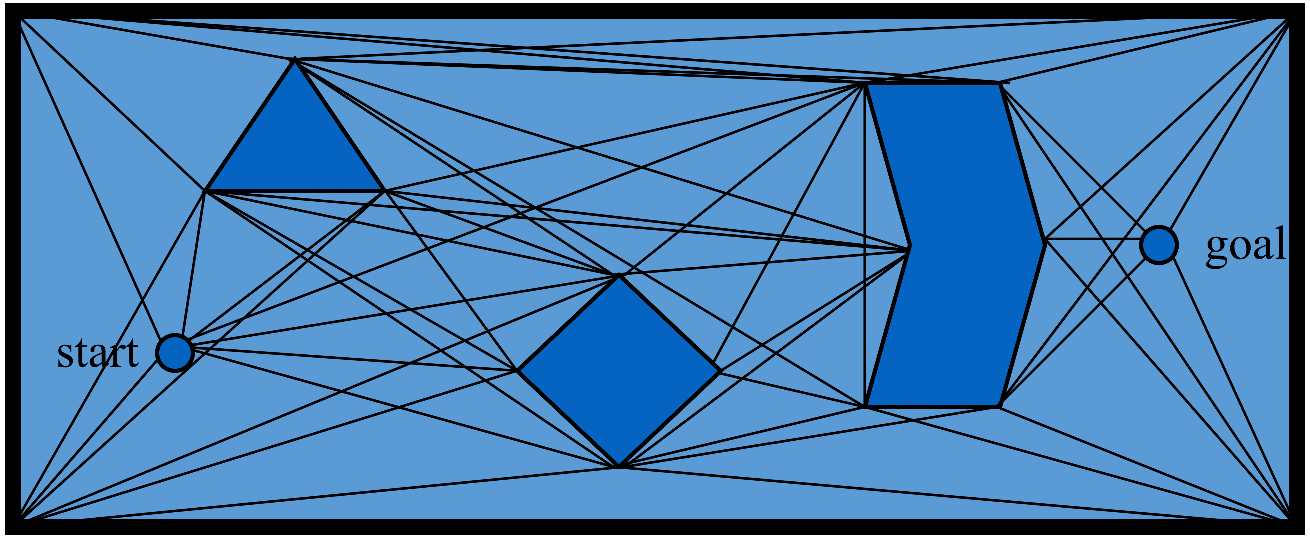
Visibility graph in action 4

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



Visibility graph, finishing

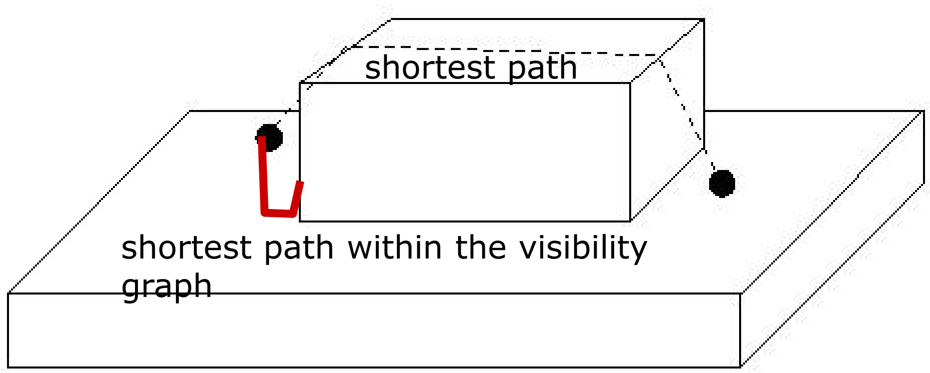
- Repeat until you're done.



Since the map was in C-space, each line potentially represents part of a path from the start to the goal.

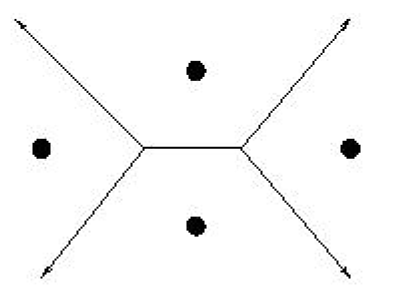
Visibility graph drawbacks

Visibility graphs do not preserve their optimality in higher dimensions:



- In addition, the paths they find are “semi-free,” i.e. in contact with obstacles.
- No clearance.

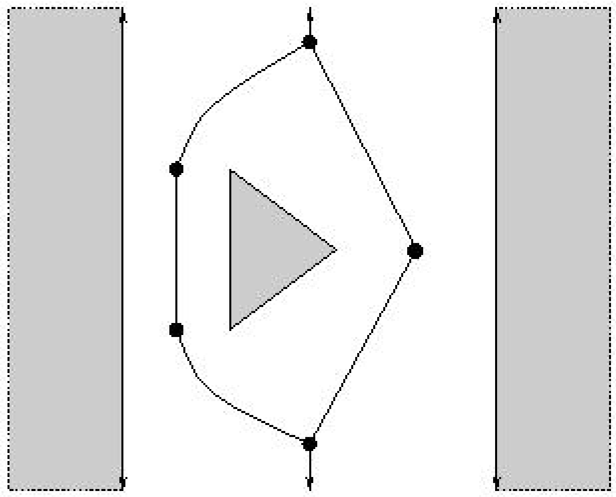
Roadmap: Voronoi diagrams



Voronoi diagram

Line segments make up the **Voronoi diagram** (isolates a set of points).

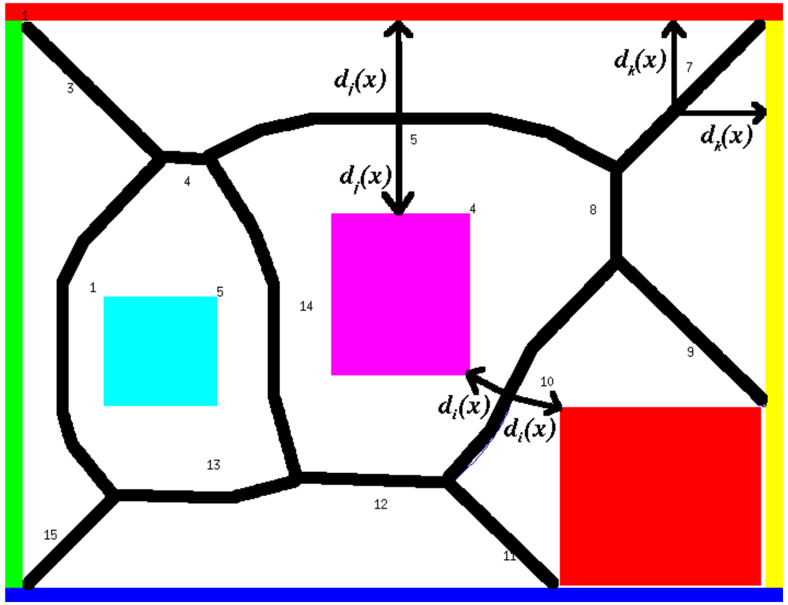
Property: maximizing the clearance between the points and obstacles.



Generalized Voronoi Graph (GVG):
locus of points is equidistant from the closest two or more obstacle boundaries, including the workspace boundary.

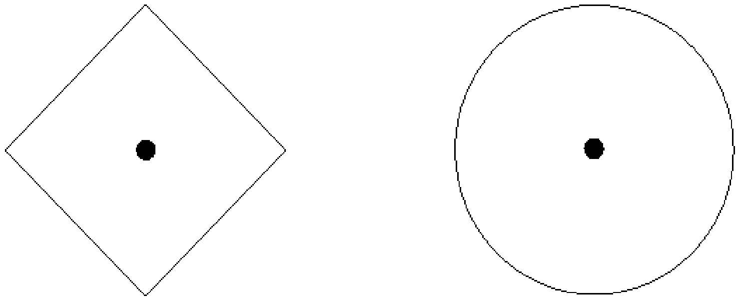
Roadmap: Voronoi diagrams

- GVG is formed by paths equidistant from the two closest objects.
- Maximizing the clearance between the obstacles.
- This generates a very safe roadmap which avoids obstacles as much as possible.



Voronoi Diagram: Metrics

- Many ways to measure distance; two are:
  - L_1 metric
    - $(x,y) : |x| + |y| = const$
  - L_2 metric
    - $(x,y) : x^2 + y^2 = const$

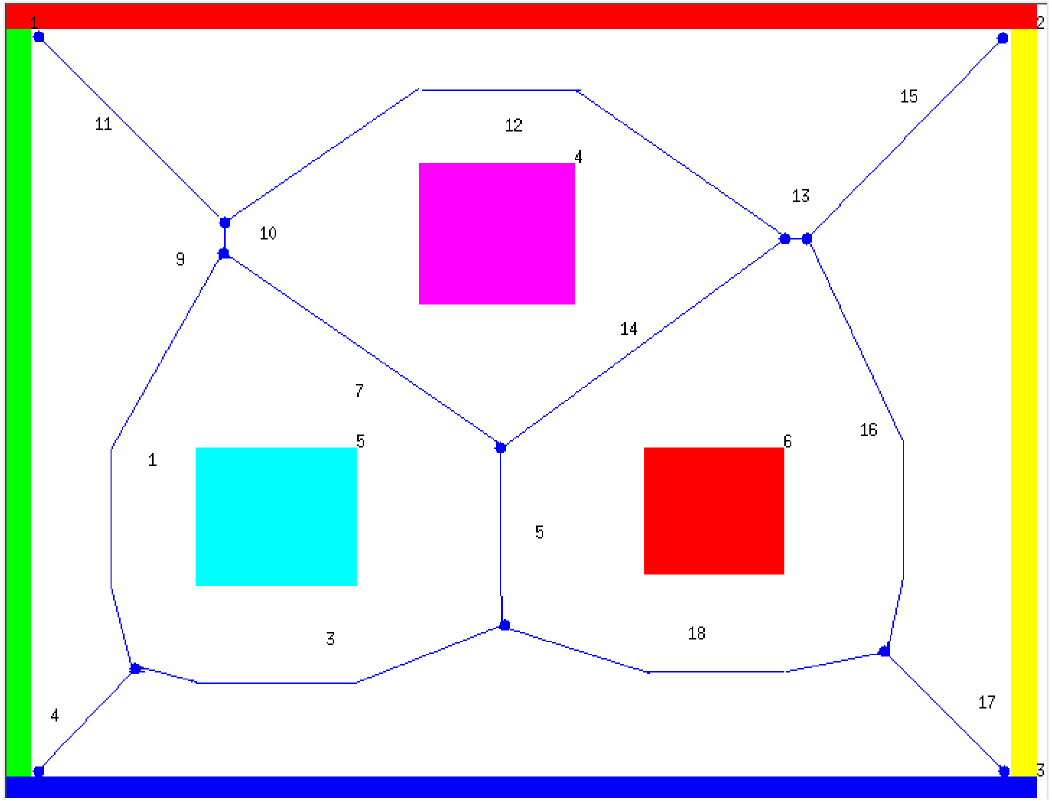


$\{(x,y) : |x| + |y| = const\}$ $\{(x,y) : x^2 + y^2 = const\}$

L1 L2

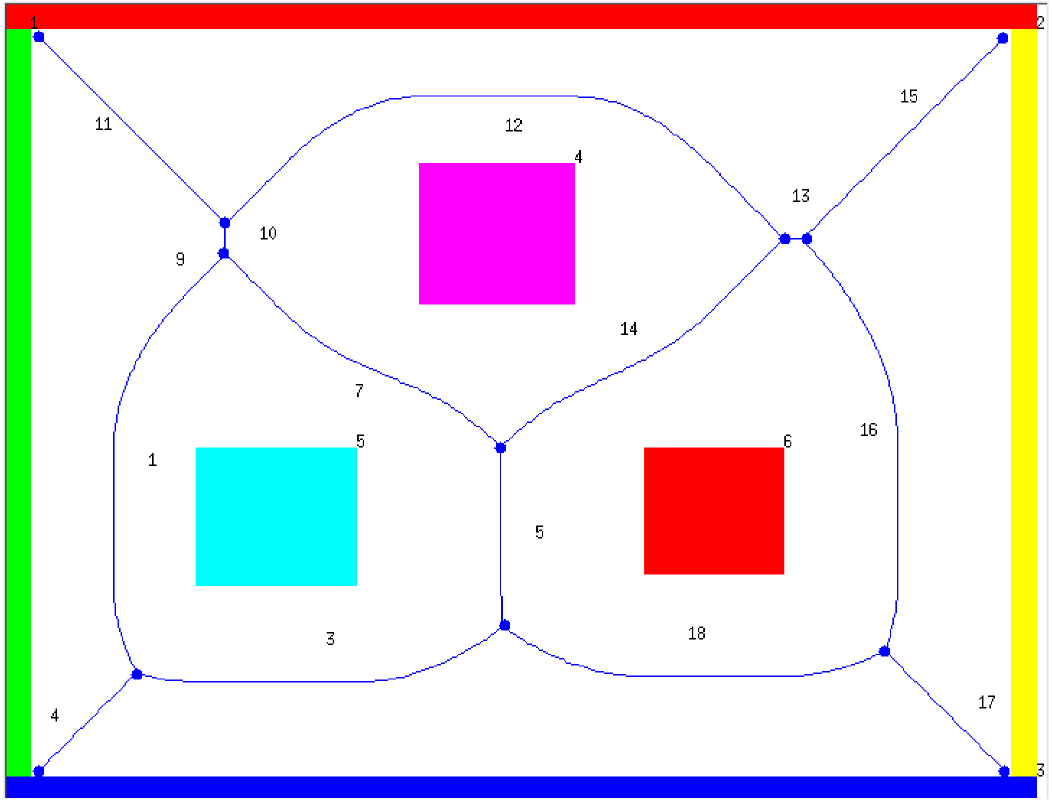
Voronoi diagram in L_1

Note the lack of curved edges.



Voronoi diagram in L_2

Note the curved edges.

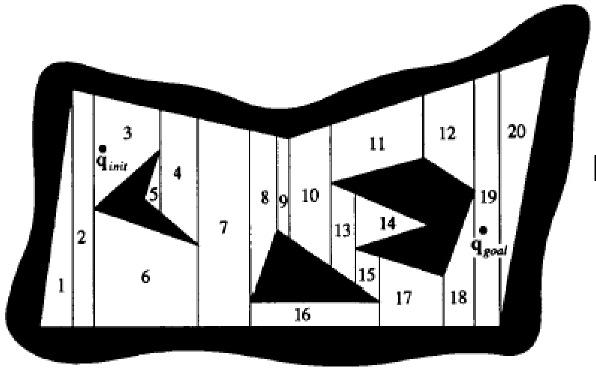


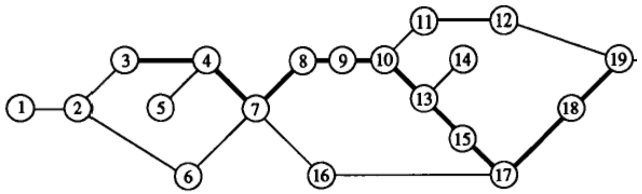
Exact cell decomposition 1

Trapezoidal Decomposition:

Decomposition of the free space into trapezoidal & triangular cells

Connectivity graph representing the adjacency relation between the cells

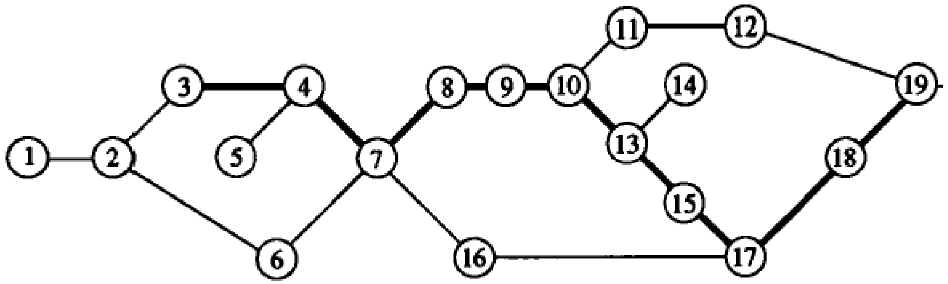


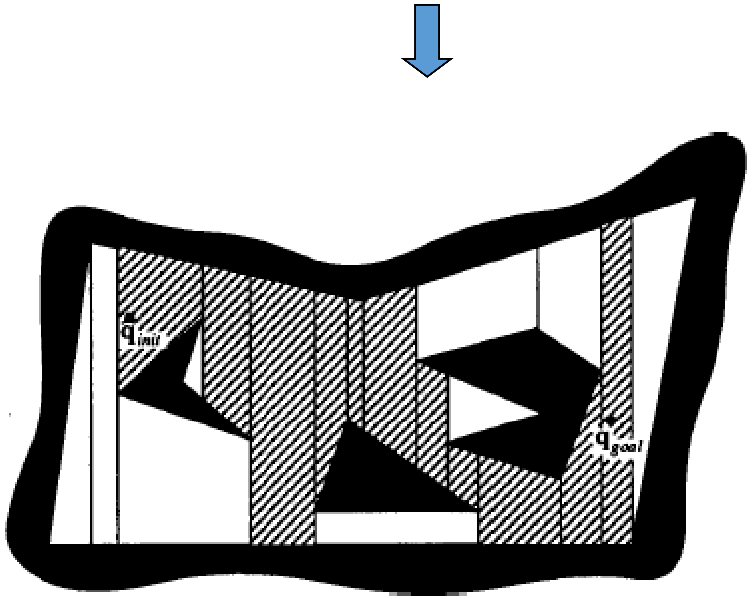


(Sweepline algorithm)

Exact cell decomposition 2

Trapezoidal Decomposition:

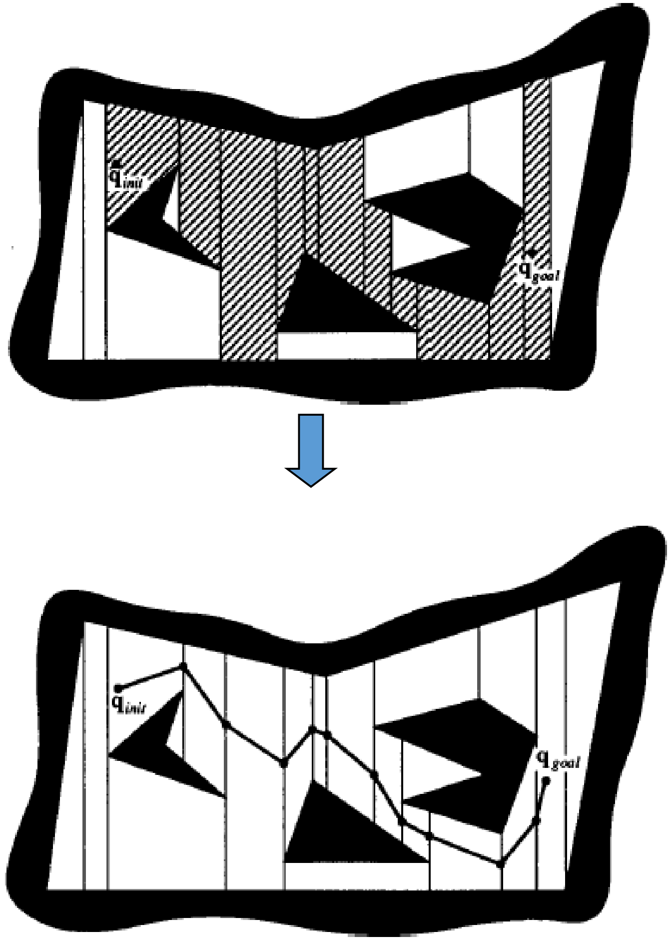




Search the graph for a path
(sequence of consecutive cells)

Exact cell decomposition 3

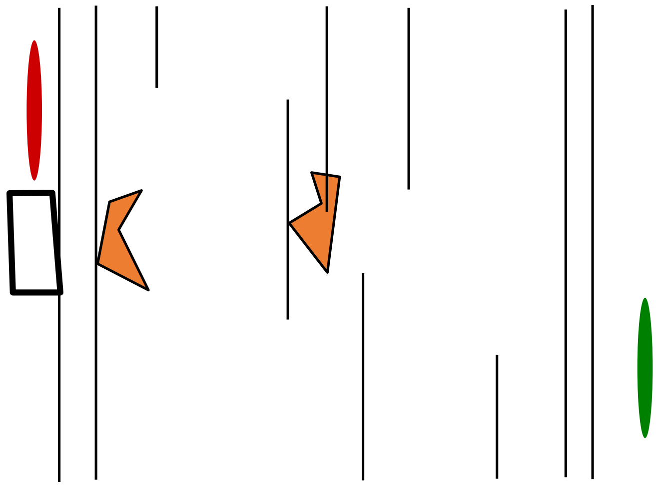
Trapezoidal Decomposition:



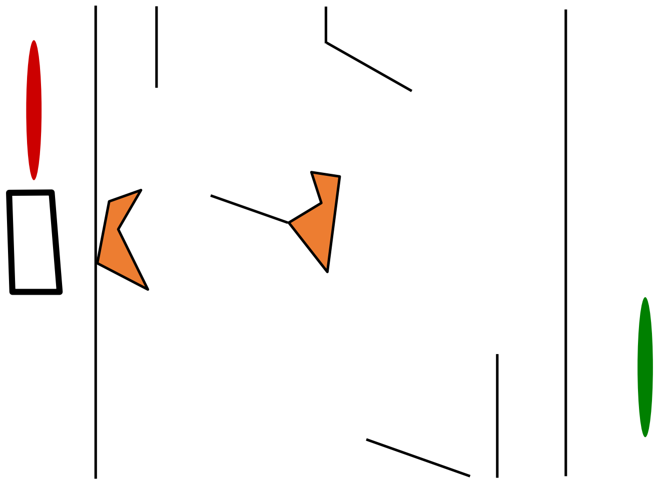
Transform the sequence of cells into a free path (e.g., connecting the mid-points of the intersection of two consecutive cells)

Optimality

Trapezoidal Decomposition:



15 cells



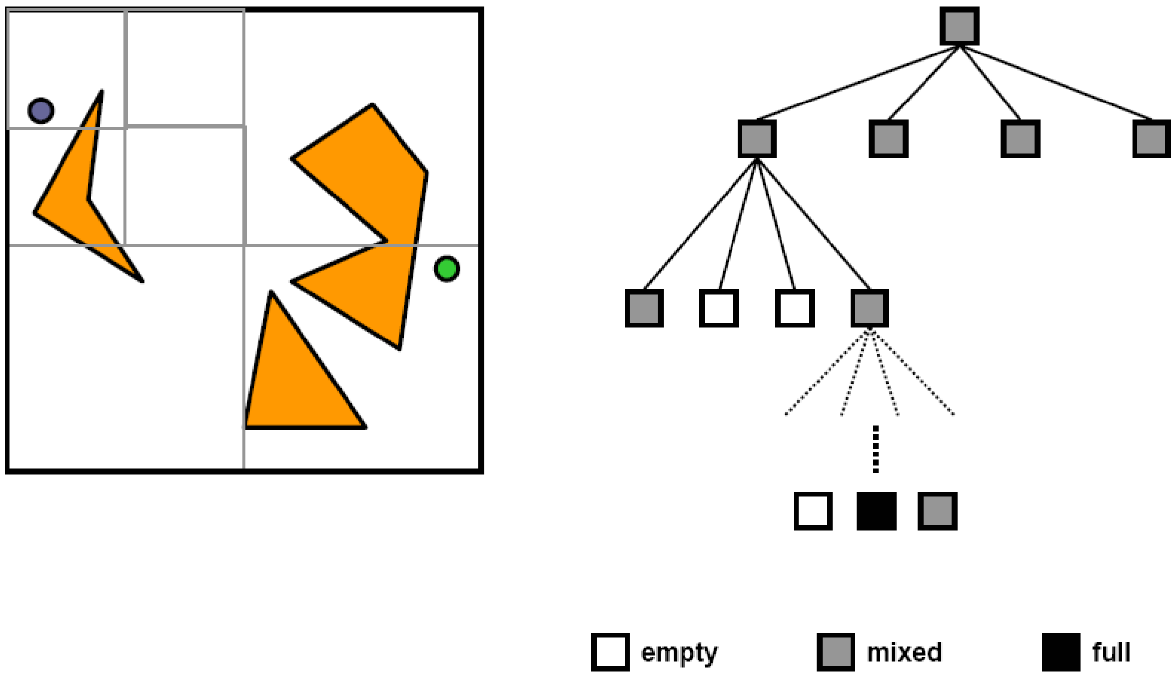
9 cells

Trapezoidal decomposition is exact and complete, but not optimal.

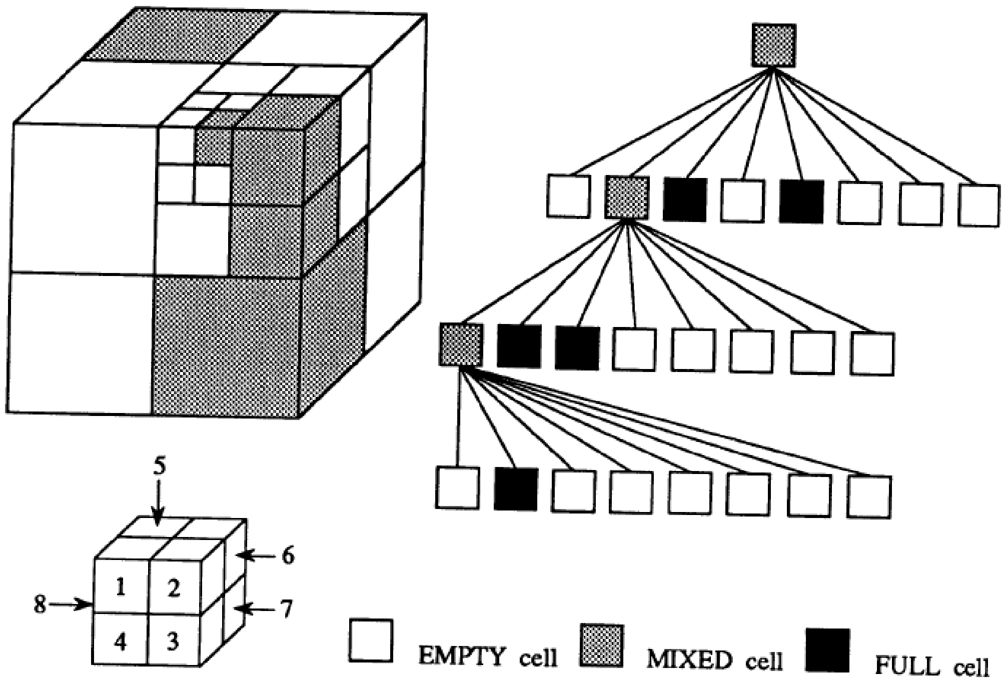
Obtaining the *minimum* number of convex cells is NP-complete.

There may be more details in the world than the task needs to worry about...

Quadtree Decomposition

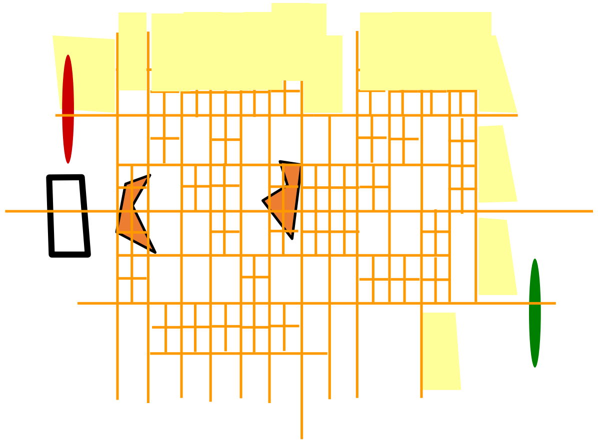


Octree Decomposition



Further decomposing …

Quadtree Decomposition:



- The rectangle cell is recursively decomposed into smaller rectangles.
- At a certain level of resolution, only the cells whose interiors lie entirely in the free space are used.
- A search in this graph yields a collision free path.

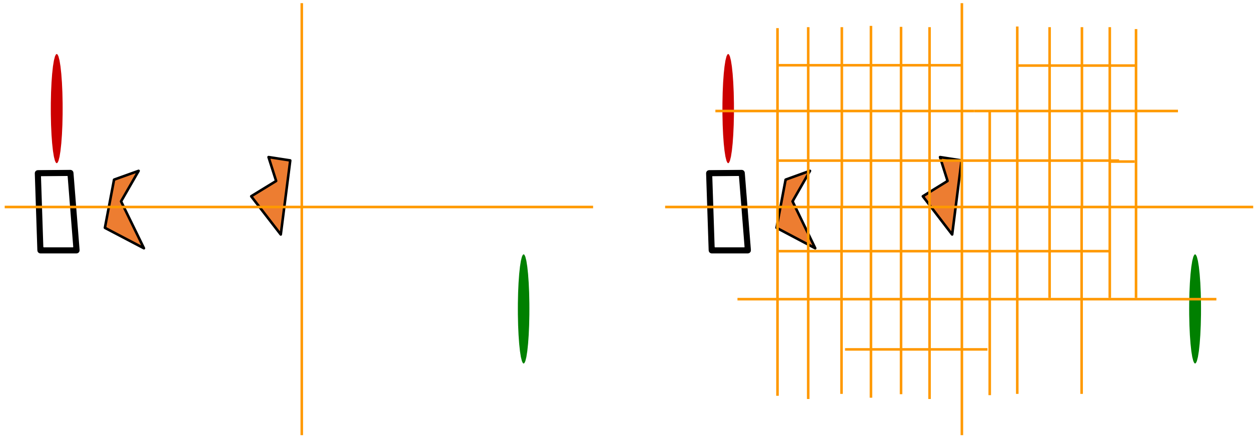
Again, use a graph-search algorithm to find a path from the start to goal.

Quadtre
e

- is this a **complete** path-planning algorithm?
- i.e., does it find a path when one exists ?

Approximate cell decomposition

Quadtree Decomposition:

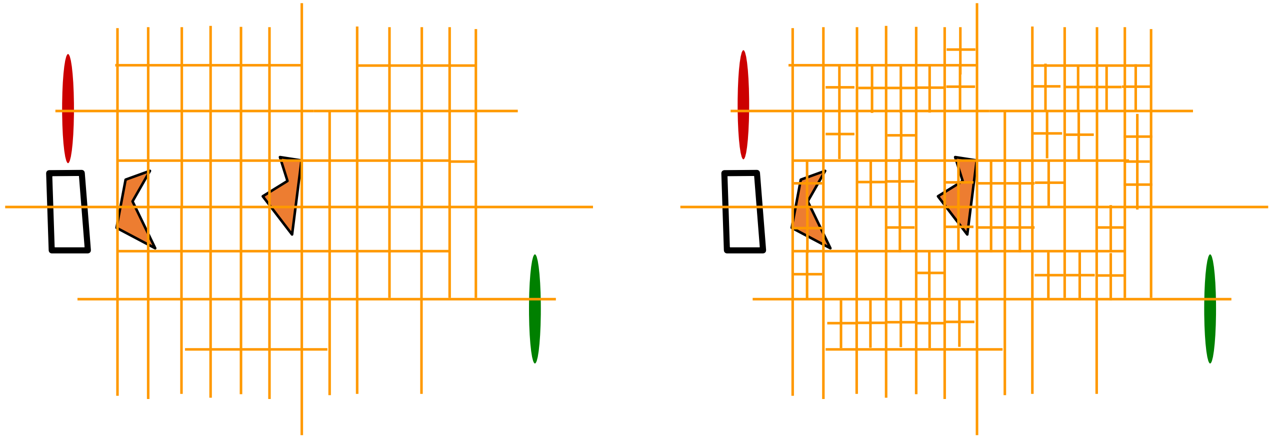


Quadtree: subdivides recursively each *mixed* obstacle/free (sub)region into four quarters

…

Further decomposing ...

Quadtree Decomposition:

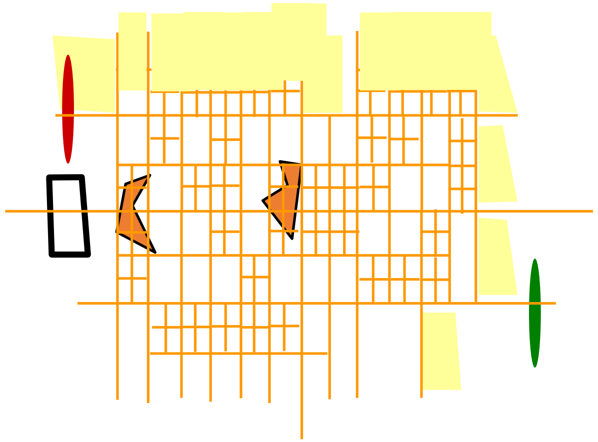


Quadtree: subdivides recursively each *mixed* obstacle/free (sub)region into four quarters

...

Further decomposing …

Quadtree Decomposition:



Quadtree

- The rectangle cell is recursively decomposed into smaller rectangles.
- At a certain level of resolution, only the cells whose interiors lie entirely in the free space are used.
- A search in this graph yields a collision free path.

Again, use a graph-search algorithm to find a path from the start to goal.

- is this a **complete** path-planning algorithm?
- i.e., does it find a path when one exists ?

Potential field method

Working Principle:

- The goal location generates an **attractive potential** – pulling the robot towards the goal.
- The obstacles generate a **repulsive potential** – pushing the robot far away from the obstacles.
- The **negative gradient of the total potential** is treated as an artificial force applied to the robot.
- Let the sum of the forces control the robot.

Artificial **Potential**

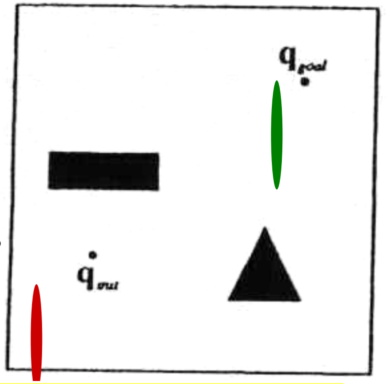
$$U(q) = \underbrace{U_{goal}(q)}_{\text{attractive potential}} + \underbrace{\sum U_{obstacles}(q)}_{\text{repulsive potential}}$$

Artificial **Force Field**

$$F(q) = -\nabla U(q)$$

Negative gradient

C-obstacles



Example: free-flying robot modeled as a point

$$F(q) = -\nabla U(q) = -\begin{bmatrix} \partial U / \partial x \\ \partial U / \partial y \end{bmatrix}$$

Potential field method

Compute an attractive force toward the goal

- Attractive Potential

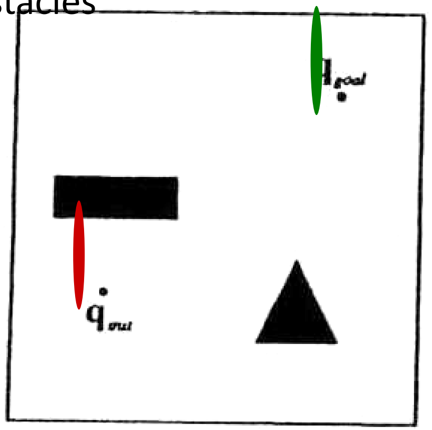
$$U_{goal}(q) = \frac{1}{2}\xi \| q - q_{goal} \|^2$$

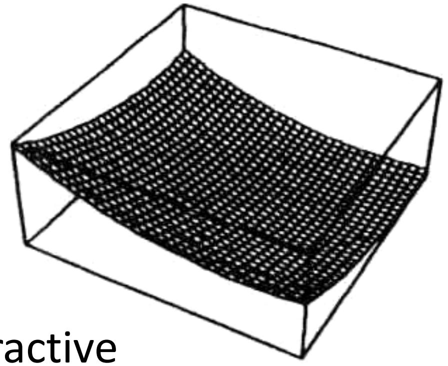
$$F_{att}(q) = -\xi (q - q_{goal})$$

Parabolic
Positive or null
Minimum at q_{goal}

Tends to zero when the robot gets closer to the goal configuration

C-obstacles





Attractive potential

Potential field method

Compute a repulsive force away from obstacles

Repulsive Potential

- Create a potential barrier around the C-obstacle region that cannot be traversed by the robot's configuration.
- It is usually desirable that the repulsive potential does not affect the motion of the robot when it is sufficiently far away from C-obstacles.

Potential field method

Compute a repulsive force away from obstacles.

- Repulsive Potential

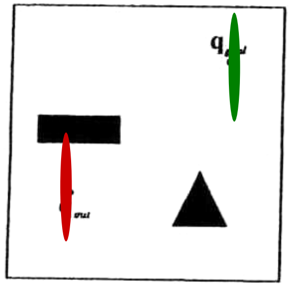
$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) > \rho_0 \end{cases}$$

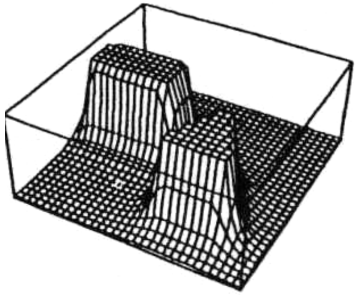
η → positive scaling factor

$\rho(q) = \min_{q' \in CB} \| q - q' \|$ → Distance from the actual configuration q to the C-obstacle region CB

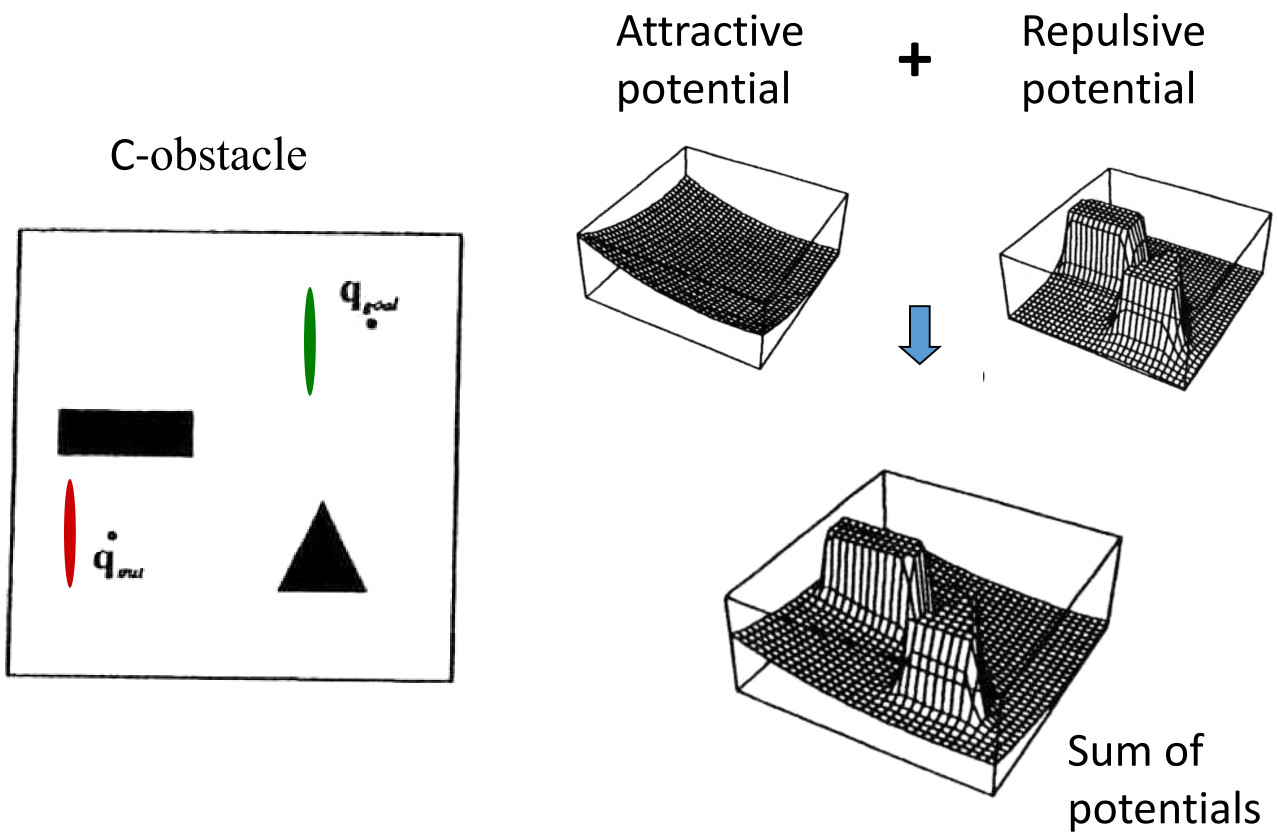
ρ_0 → Positive constant (**distance of influence**) of the C-obstacles

$$F_{rep}(q) = \begin{cases} \eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2(q)}\nabla\rho(q) & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) > \rho_0 \end{cases}$$





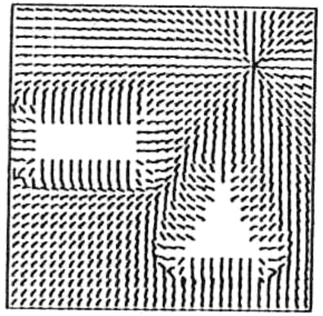
Sum of potentials



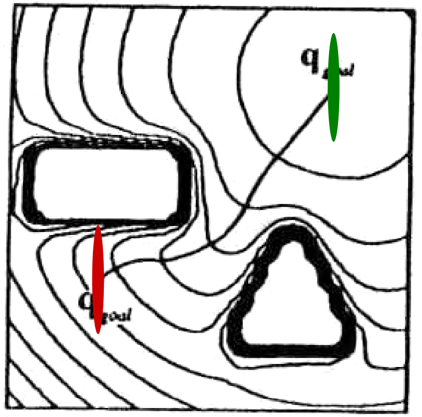
Potential field method

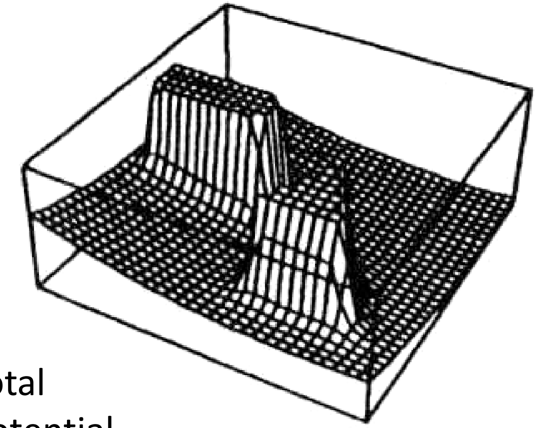
- After the total potential is obtained, generate force field (negative gradient)
- Let the sum of the forces control the robot.

Negative gradient



Equipotential contours





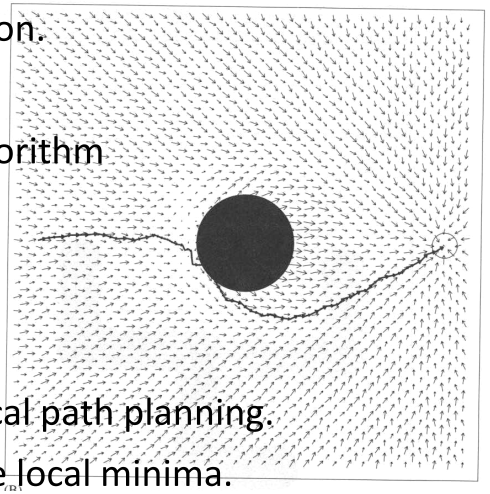
Total potential

To a large extent, this is computable from sensor readings.

Potential field method

Pros:

- Spatial paths are not preplanned and can be generated in real time.
- Planning and control are merged into one function.
- Smooth paths are generated.
- Planning can be coupled directly to a control algorithm



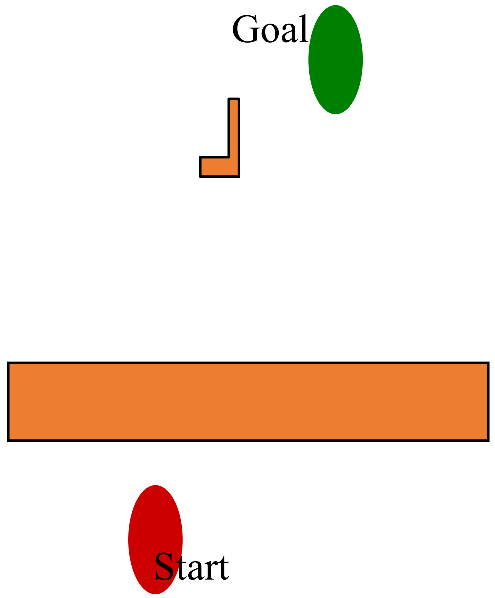
Cons:

- Trapped in local minima in the potential field.
- Because of this limitation, commonly used for local path planning.
- Use random walk, backtracking, etc to escape the local minima.

Random walks are not perfect …

Bug algorithms

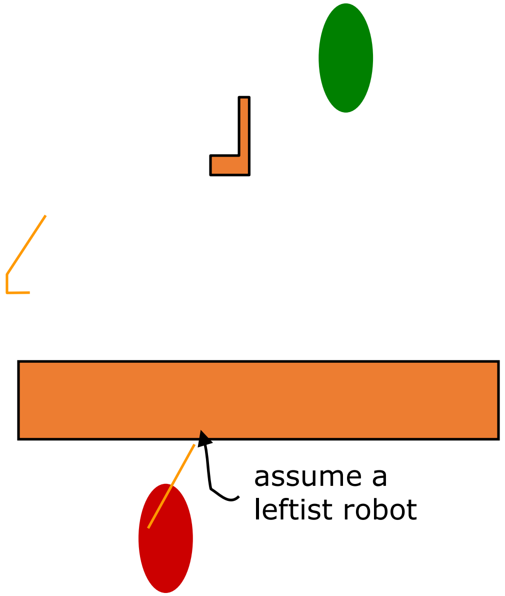
- Path planning with limited knowledge.
- Insect-inspired “bug” algorithms.



- Known direction to goal.
- Only local sensing (walls/obstacles encoders).
- “Reasonable” world.
- Finite obstacles in any finite range.
- A line will intersect the obstacle finite times.

Beginner Strategy

Insect-inspired “bug” algorithms



Switching between two simple behaviors:

1. Moving directly towards the goal.
2. Circumnavigating an obstacle.

Bug algorithm:

1. Head toward goal.
2. Follow obstacles until you can head toward the goal again.
3. Continue.

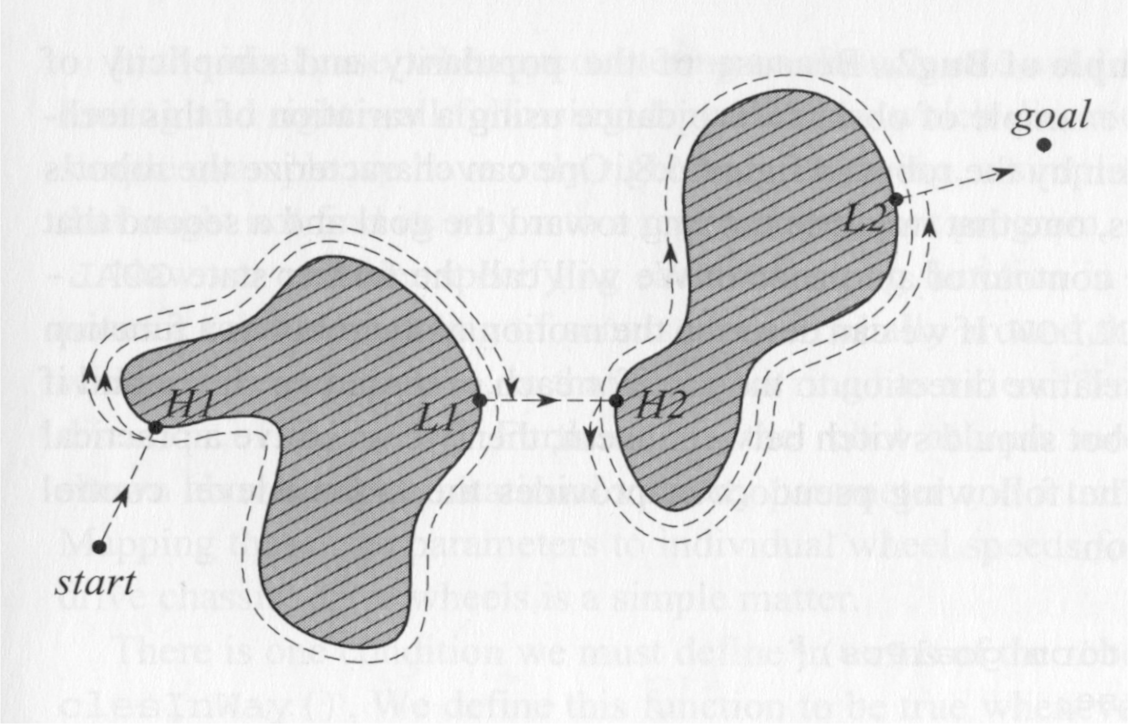
Bug algorithms 2

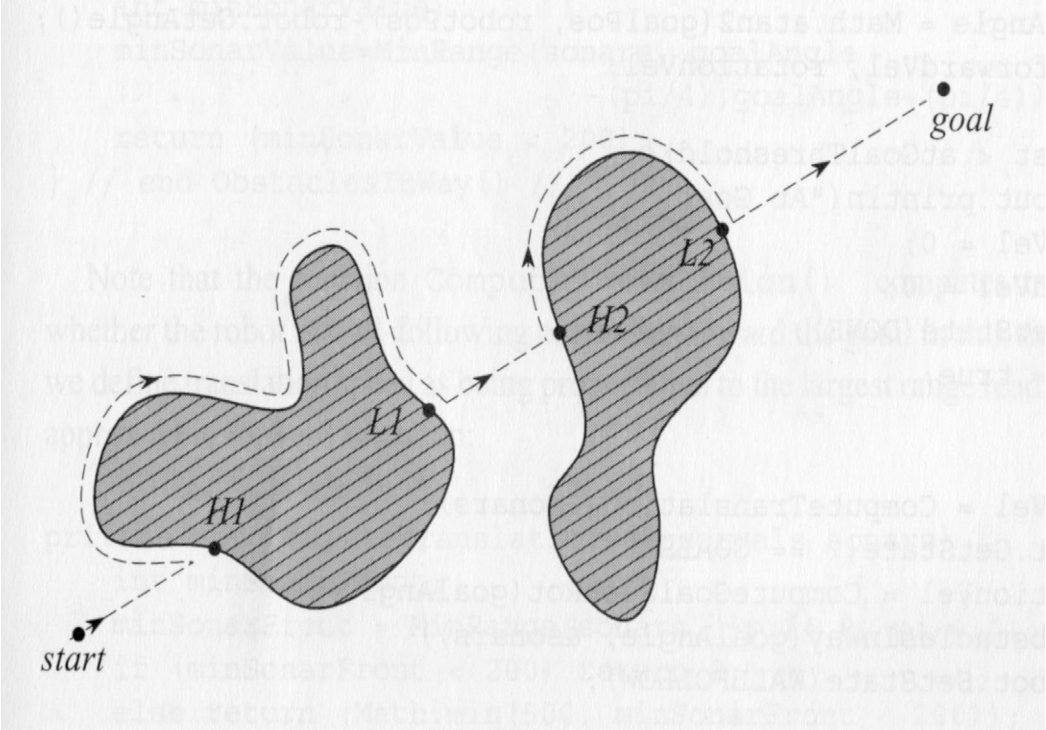
- In many cases, a global map of the environment is not available when the robot begin moving towards its goal.
- Local potential field-based planners cannot be guaranteed to find a path to the goal.
  - Bug1 algorithm / Bug2 algorithm
  - Tangent Bug algorithm
- These algorithms are used for path planning from a starting location to a goal with known coordinates, on the assumption of:
  - a holonomic point robot with perfect odometry,
  - an ideal contact sensor (zero range sensor),
  - and infinite memory.

Bug1 / Bug 2 / Tangent algorithms

- **Bug1 algorithm** exhibits two behaviors:
  - Motion-to-Goal.
  - Boundary-following (hit point / leave point).
- **Bug2 algorithm** shows similar behaviors:
  - The line from a start point to the goal is fixed.
- **Tangent Bug algorithm**
  - An improvement to the Bug2 algorithm in that it determines a shorter path to the goal using a range sensor with a 360 degree infinite orientation resolution.

Bug algorithms, example

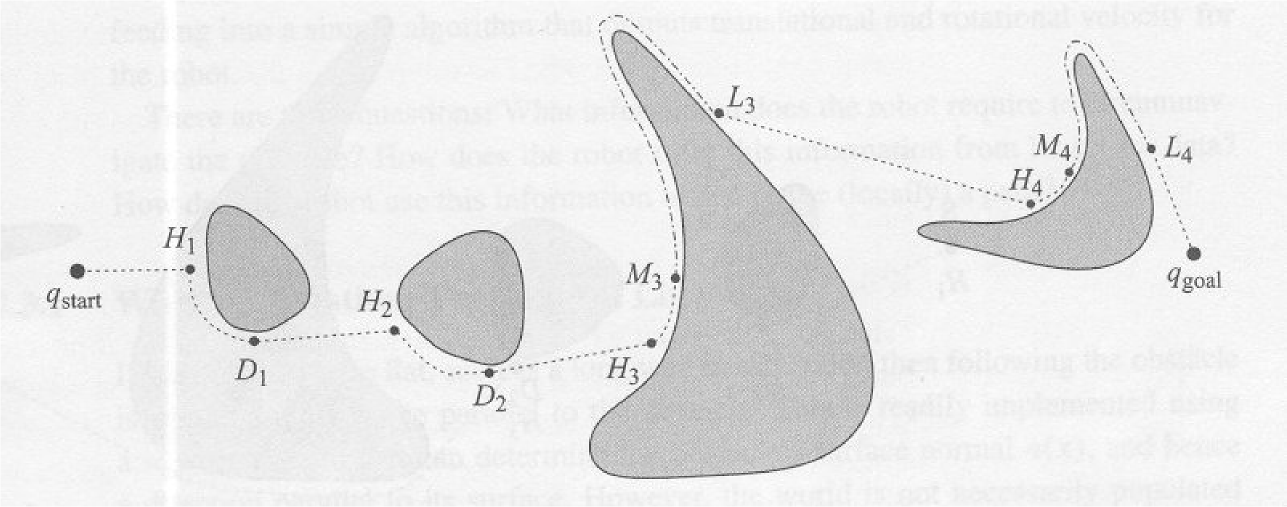




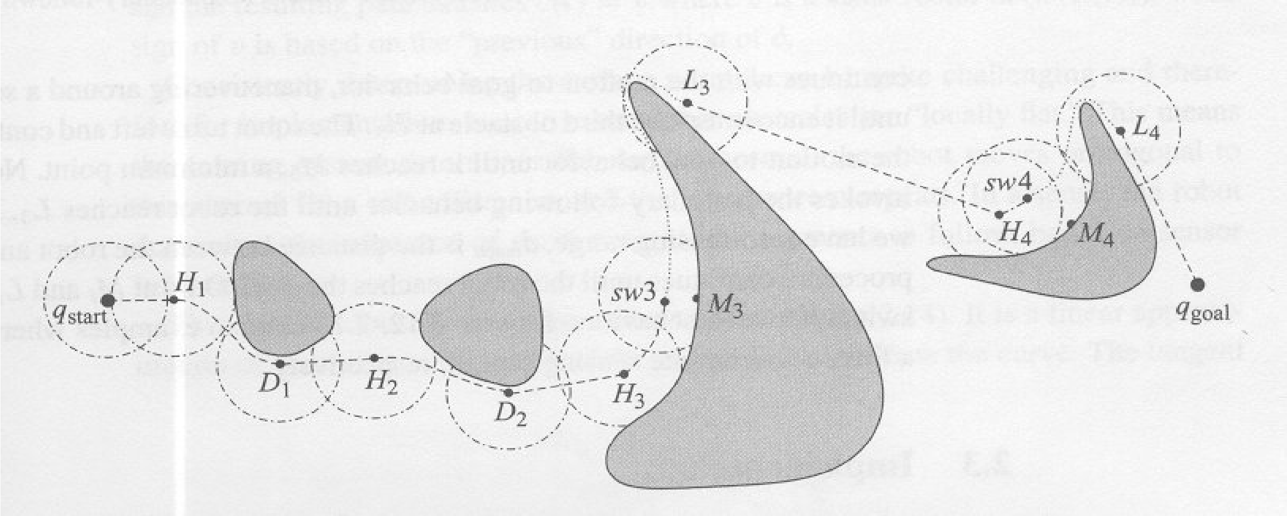
The path generated by Bug1

The path generated by Bug2

Bug algorithms, example 2

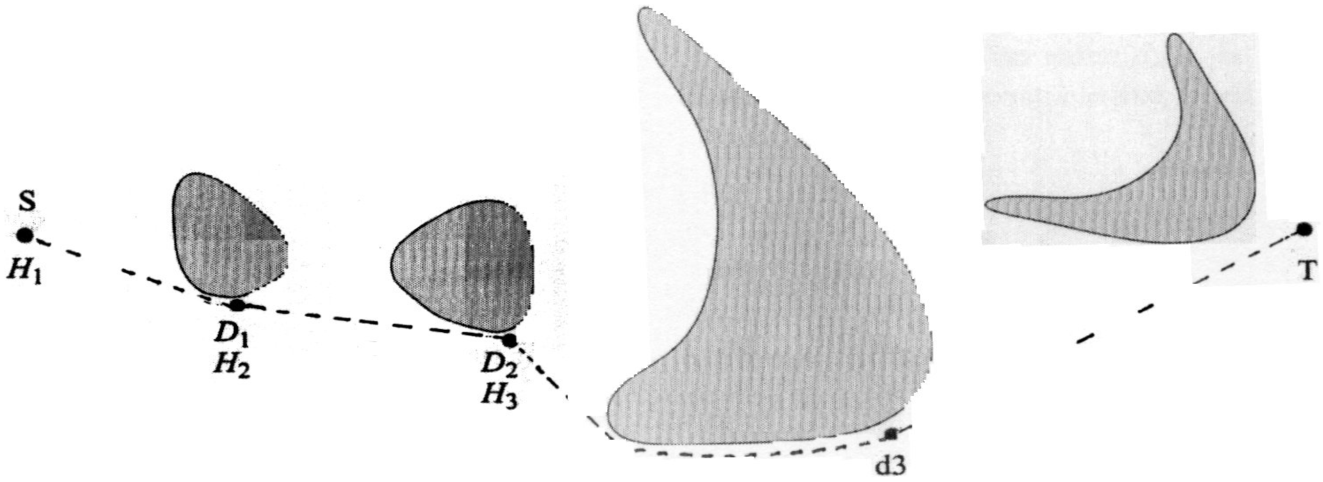


The path generated by Tangent Bug with zero sensor range



The path generated by Tangent Bug with finite sensor range

Bug algorithms, example 3



The path generated by Tangent Bug with infinite sensor range