Path Planning

A Robot's Navigation Problems

- Where am I? *Localization*
- Where have I been? *Map making*
- Where am I going? *Mission planning*
- What's the best way there? *Path planning*

Path

- A sequence of robot configurations from a starting configuration to an end configuration.
- Must be continuous.
- Must be in a specific order.
- Usually some cost values are applied to the robot for changing configurations.

Path Planning

- Also called *path finding or route finding.*
- Usually the collision-free path with minimum cost is preferred.
- The cost may be distance, time etc.
- Both optimization and search problem.

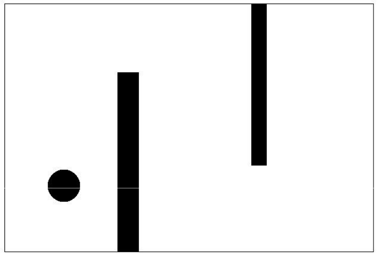
Complexity of Path Planning

- In 3D work space, finding exact solution is NP-HARD. [Xavier92]
- Path planning is PSPACE-HARD. [Reif79]
- The complexity increases exponentially with:
  - Number of DOF [Canny88]
  - Number of agents.

Configuration Space

- Also called *CSpace*,
- The set of all possible configurations of a robot
- The minimum number of parameters needed to completely specify the configuration of the object

Sample Configuration Space



Work space

its configuration space

Algorithms

- Simple algorithms
- Sampling based algorithms

Simple Algorithms

- Bug Algorithms
- Wavefront Methods
- Roadmaps
  - Meadow Maps
  - Visibility Graphs
  - Generalized Voronoi Graphs
- Cell Decomposition

Bug Algorithms

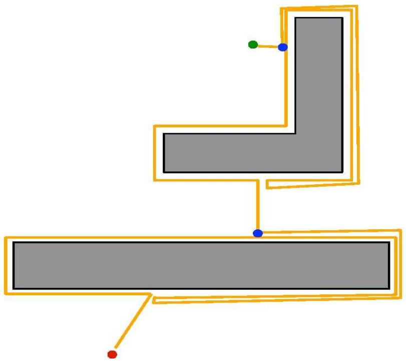
- Many planning algorithms assume global knowledge
- Bug algorithms assume only *local knowledge of the environment* and a global goal
- Bug behaviors are simple:
  - Follow a wall (right or left)
  - Move in a straight line toward goal
- Bug 1 and Bug 2 assume essentially tactile sensing
- Tangent Bug deals with finite distance sensing

Bug Algorithms

- Known direction to goal
  - The robot can measure distance between any two points *x* and *y*
- Uses local sensing for walls/obstacles and encoders
- Reasonable world assumption
  - finitely many obstacles in any finite area
  - a line will intersect an obstacle finitely many times
  - Workspace is bounded

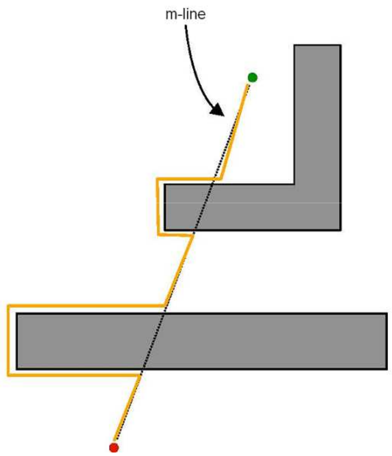
Bug 1 Algorithm

- Head toward goal
- If an obstacle is encountered, circumnavigate it *and remember* how close you get to the goal
- Return to that closest point (by wall-following) and continue



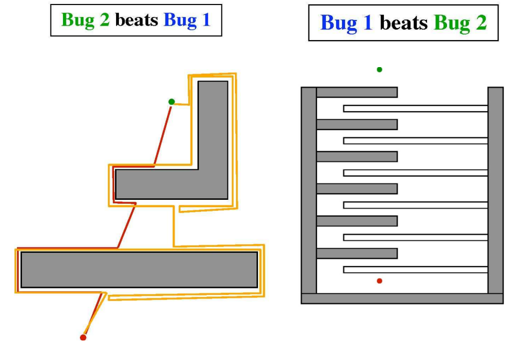
Bug 2 Algorithm

- Head toward goal on the *m-line*
- If an obstacle is in the way, follow it until you encounter the m-line again ***closer to the goal.***
- Leave the obstacle and continue toward the goal



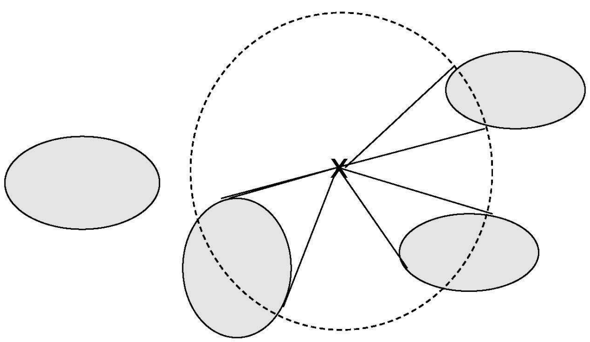
Bug 1 Algorithm vs. Bug 2 Algorithm

- BUG 1 is an *exhaustive search algorithm*
  - it looks at all choices before committing
- BUG 2 is a *greedy algorithm*
  - it takes the first thing that looks better
- In many cases, BUG 2 will outperform BUG 1, but
- BUG 1 has a more predictable performance overall



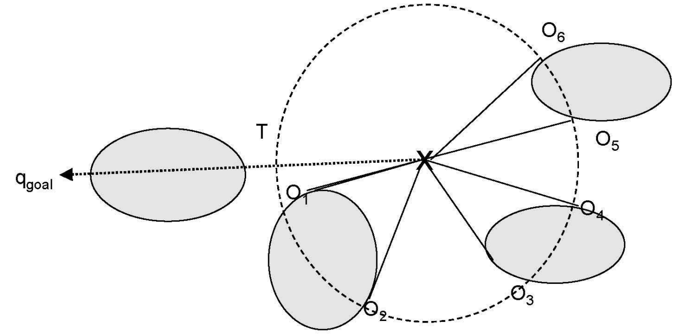
Tangent Bug

- Tangent Bug relies on finding endpoints of finite, continuous segments of ρR



Tangent Bug

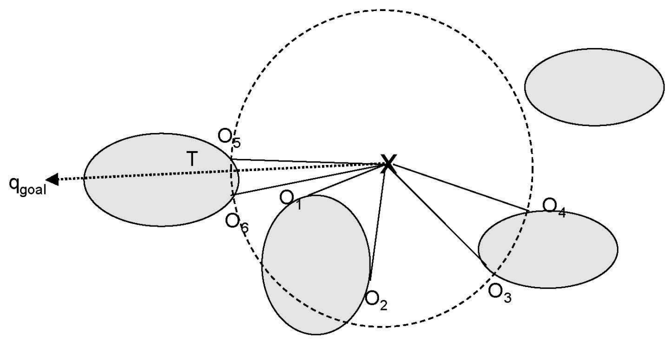
- Tangent Bug relies on finding endpoints of finite, continuous segments of ρR



Currently, the bug thinks it can get to goal!

Tangent Bug

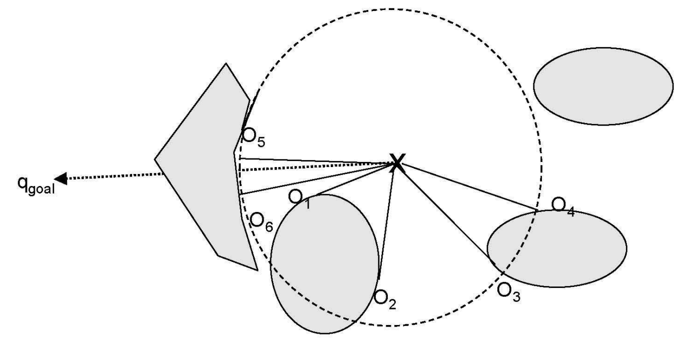
- Tangent Bug relies on finding endpoints of finite, continuous segments of ρR



It chooses the point O_i that minimizes $d(x,O_i) + d(O_i,q_{goal})$
"Heuristic distance"

Tangent Bug

- Tangent Bug relies on finding endpoints of finite, continuous segments of ρR



If this distance starts it should act like a BUG and follow boundary

The Basic Ideas

- A motion-to-goal behavior as long as the path is clear or there is a visible obstacle boundary point that decreases heuristic distance
- A boundary following behavior invoked when heuristic distance increases.
- Terminate boundary following behavior when

  $d_{reach} < d_{followed}$

- where
  - $d_{followed}$: the shortest distance between the sensed boundary and the goal
  - d_{reach} : the shortest distance between *blocking* obstacle and goal (or the distance to goal if no blocking obstacle visible)

Wavefront Methods

- Divide the map into a grid,
- give a number to each grid cell according to its distance,
- follow the numbers in decreasing order

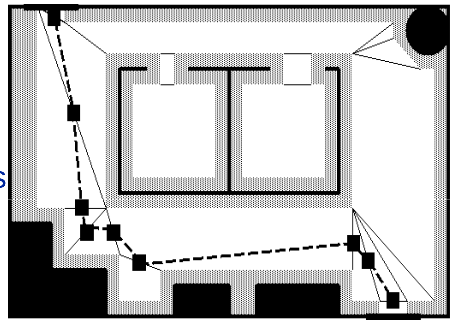
7	**18**	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	17	16	15	14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Roadmaps

- *CSpace* representation algorithms
- Meadow Maps
- Visibility Graphs
- Generalized Voronoi Graphs
- Cell Decompositions

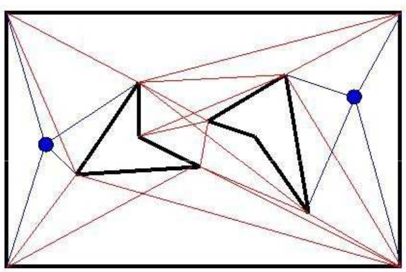
Meadow Maps

- Enlarge obstacles as big as the robot, to transform from work space to configuration space
- Find corners of the objects
- Connect the corners
- Use centers of these edges as milestone points



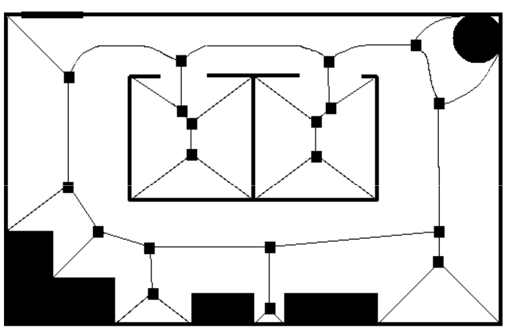
Visibility Graphs

- Connect every pair of
  - important points
  - vertices of obstacles
  - initial and final points
- Edges should not be in collision with other objects
- Graph is the edges and nodes of *Cspace* representation
- Searches will be done on this graph



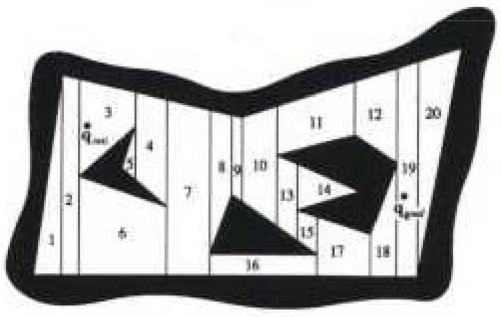
Generalized Voronoi Graphs

- Edges of the Voronoi graph
  - The points in the work space, having the same distance to the surrounding obstacles [Aur91]
- Nodes of the Voronoi graph
  - Intersections of these lines



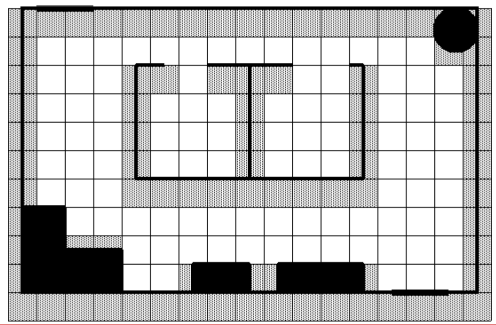
Trapezoidal Cell Decomposition

- Start a new line for each critical point
  - Corner points
- Divide the space into trapezoidals
- Move from one to the next



Regular Grids

- Divide the map into regular grids
  - Collision free, colliding and semicolliding regions
  - Similar to enlarged pixels
- It is complete
  - If a path exists, it will be found



Potential Field Approach

- 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

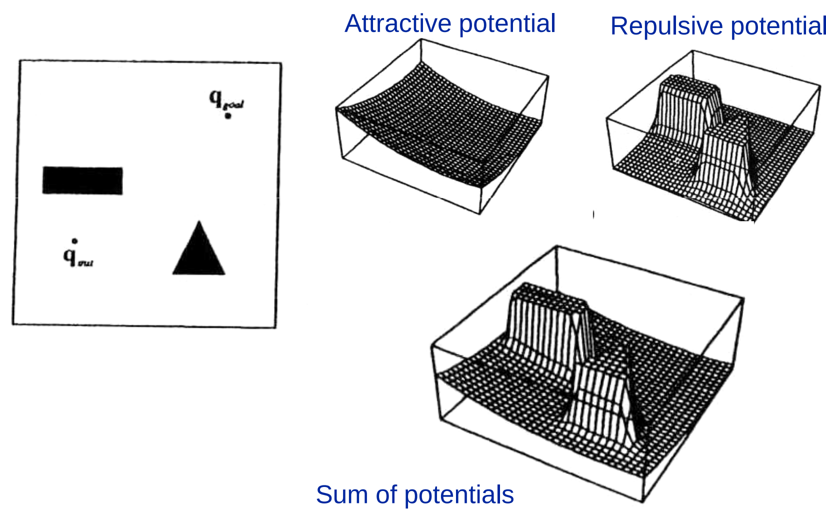
Artificial **Potential**

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

Artificial **Force Field**

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

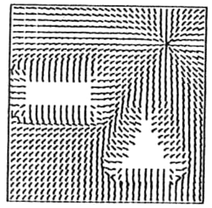
Potential Field Approach



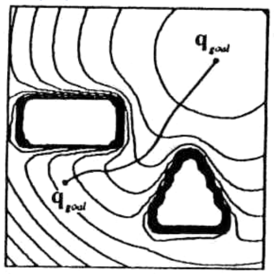
Potential Field Approach

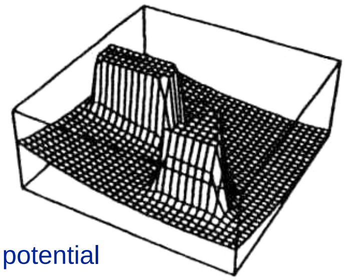
- After getting total potential, generate force field (negative gradient)
- Let the sum of the forces control the robot

Negative gradient



Equipotential contours





Total potential

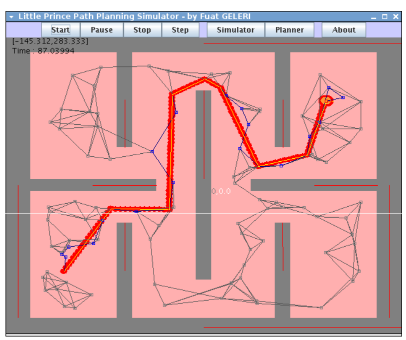
To a large extent, this is computable from sensor readings

Sampling Based Algorithms

- Probabilistic Roadmaps
- Rapidly Exploring Random Tree
- Variants based on
  - Sampling Strategies
  - Connection Strategies
  - Lazy Algorithms
  - Post Processing

Probabilistic Roadmaps

- Build a roadmap
  - Sample N random configuration
  - Collision free
  - Try to connect with each other
  - To M nearest node
- Query a path
  - Connect q_{init} and q_{goal} to the graph
- Search the graph
- Fast query step, but needs preprocessing for building the roadmap



Rapidly Exploring Random Tree (RRT)

- Select a random configuration q_{rand} in C_{free}
- Nearest configuration in the tree is found $q_{nearest}$
- Try to connect $q_{nearest}$ to q_{rand}