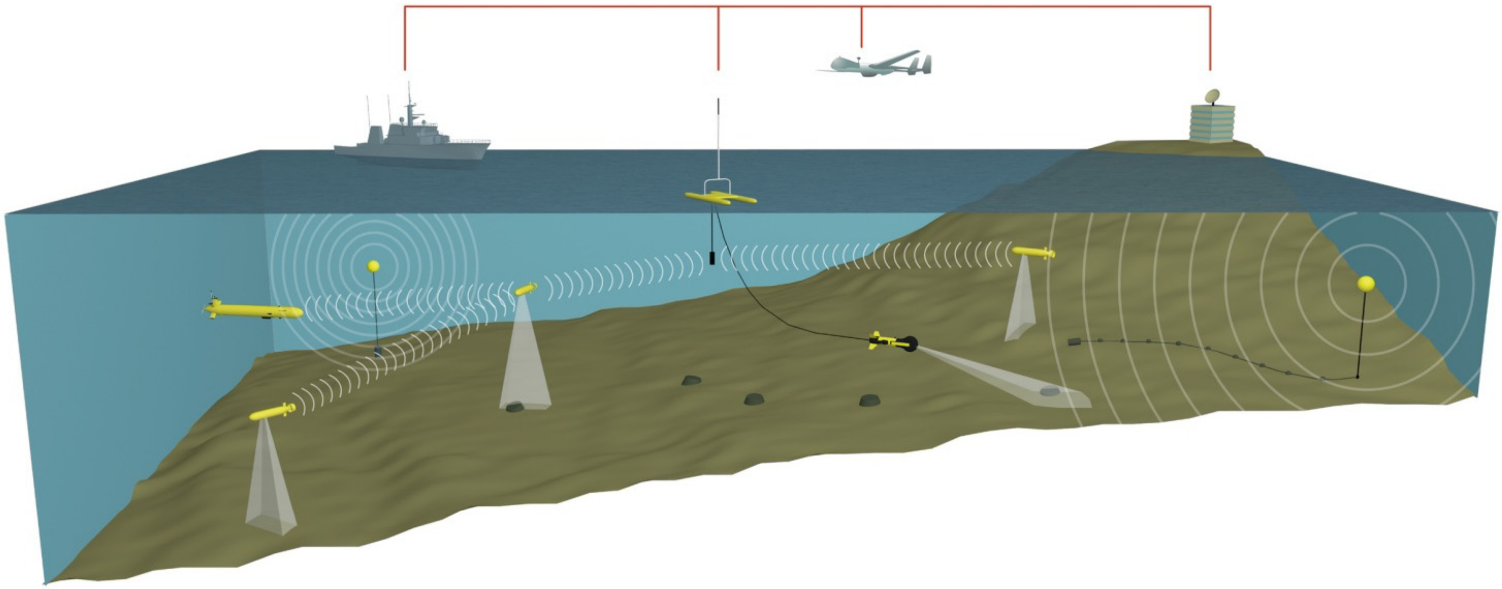
Autonomous Robotics 6905

- Introduction
- Configuration Spaces
- Global Path Planning
- Local Obstacle Avoidance
- Concluding Remarks



Lecture 5: Introduction to Path-Planning and Navigation

Dalhousie University

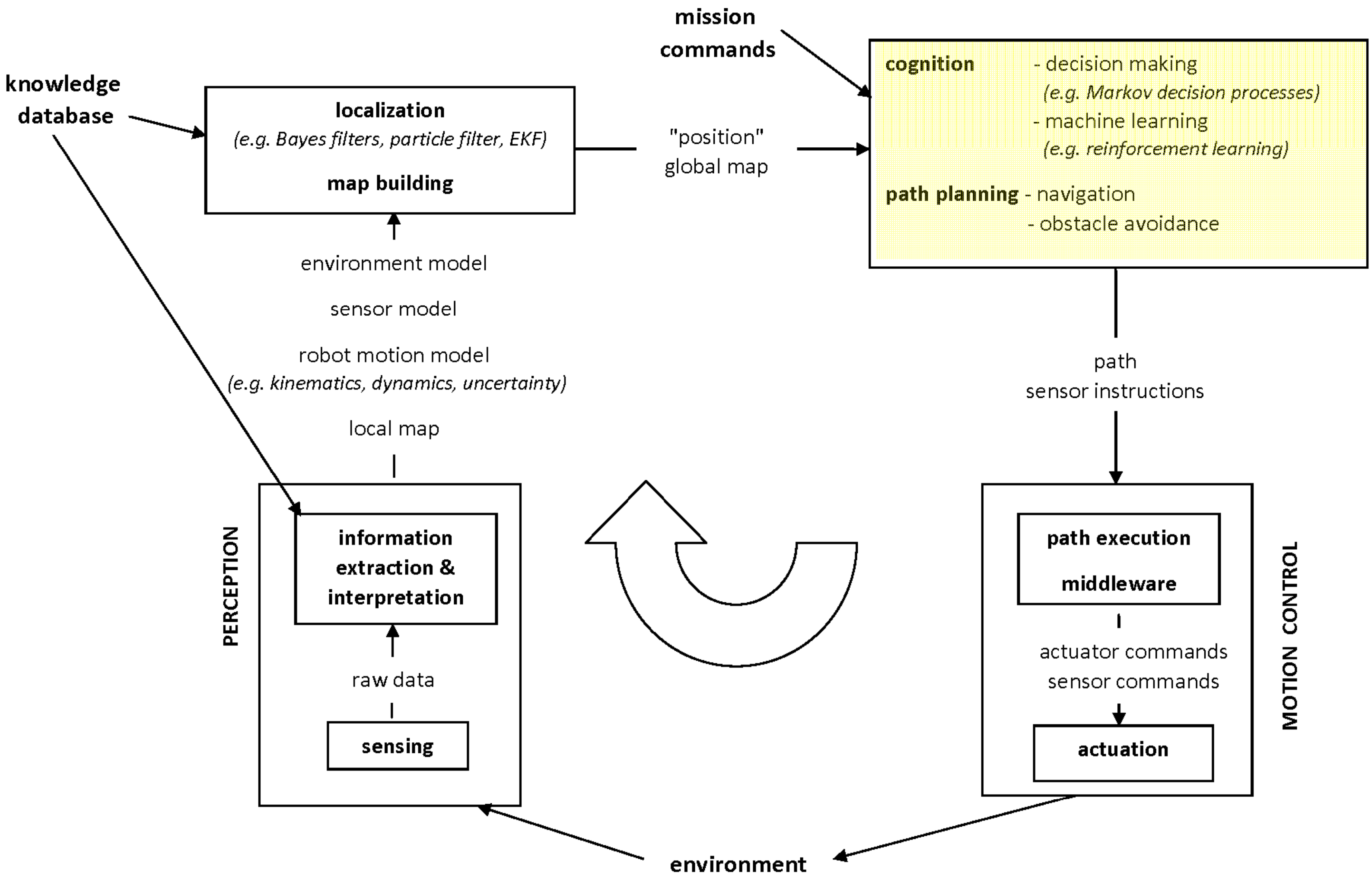
October 7, 2011

Lecture Outline

- Introduction
- Configuration Spaces
- Global Path Planning
- Local Obstacle Avoidance
- Concluding Remarks

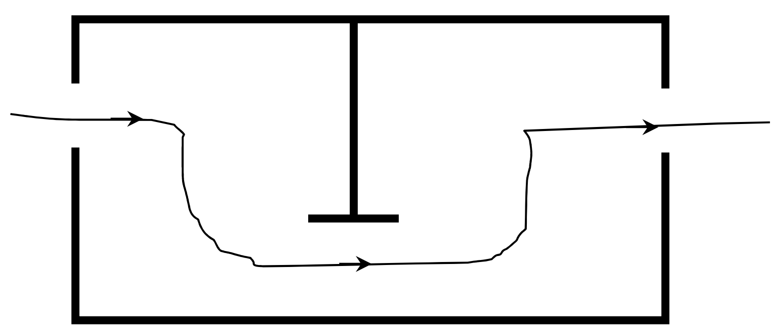
- based on diagrams and lecture notes adapted from:
  - Probabilistic Robotics (Thrun, et. al.)
  - Autonomous Mobile Robots (Siegwart, Nourbakhsh)

Control Scheme for Autonomous Mobile Robot



Control Scheme for Autonomous Mobile Robot – the plan

- Thomas covered generalized Bayesian filters for localization last week
  – Kalman filter most useful outcome for localization
- Mae today covers path-planning and navigation
- Mae then follows on next week with Bayesian filters to do a specific example, SLAM
- Thomas to follow with reinforcement learning after that



Path-Planning and Navigation

addresses for the robot:

- where am I?
- where am I going?
- how do I get there?

- distinguish between low level control and robot control
  - low level control looks after very basic behaviors in a robot (e.g. follow a straight line, maintain depth, etc.)
  - robot control effected through controllers that determine robot action with minimum computing time
  - robot planning and control merge in some areas . . .

Planning & Navigation Definitions

- *navigation*: given partial information about the environment and goal positions, act based on knowledge and sensor values to reach goal positions as efficiently and reliably as possible

- *path-planning*: given map and goal location, identify a trajectory that will cause the robot to reach the goal location

- *obstacle avoidance*: given real-time sensor readings, modulate the robot trajectory to avoid collisions

Planning and Navigation Overview

- state-space and obstacle representation
  - work space
  - configuration space
- global motion planning
  - optimal control
  - potential fields
  - deterministic graph search
- collision avoidance
  - VFH
  - DWA
  - BUG

Planning & Navigation Objectives

- find a path in the physical work space from an initial position to a goal position avoiding all collisions with obstacles

- assume: map of the environment available for navigation
  - topological
  - metric
  - hybrid methods

- distinguish between
  - global path planning
  - local obstacle avoidance

Planning & Navigation Methodology

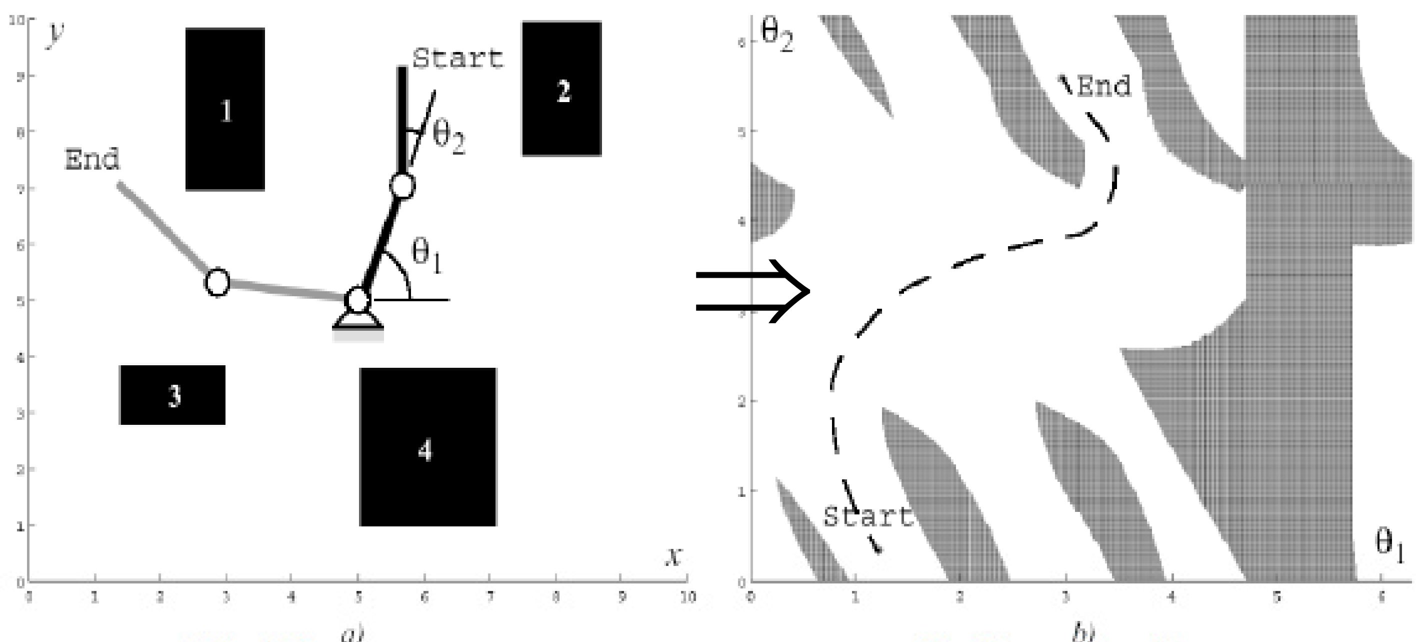
1. transform map into a representation useful for planning
   – planner dependent

2. plan path on transformed map

3. send motion commands to controller
   – planner dependent (path following, feed forward, etc.)

Work Space Map → Configuration Space

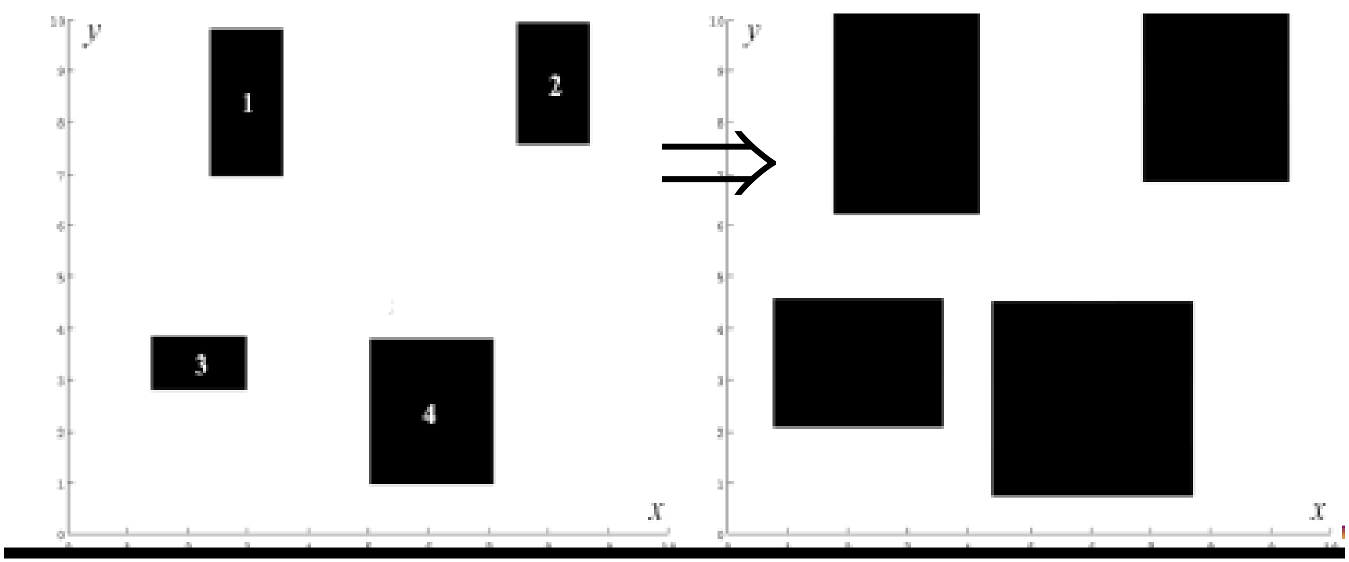
- state or configuration, q, described with k values q_i
- easier to plan path in configuration space
- motion constrained by obstacles 1 to 4
- gray sections in C-space are unachievable due to obstacles



Work Space ⟹ **Configuration Space:**

Configuration Space Mobile Robot

- mobile robot on a flat surface has 3 DOF (x,y,θ)
- assume robot is holonomic and can be reduced to a point $\rightarrow$ configuration space (or C-space) reduces to two-dimensional (x,y)
- consequently, have to inflate obstacles by the size of the robot radius to keep scaling consistent



Global Path-Planning

- lots of techniques available, the most popular are as follows:

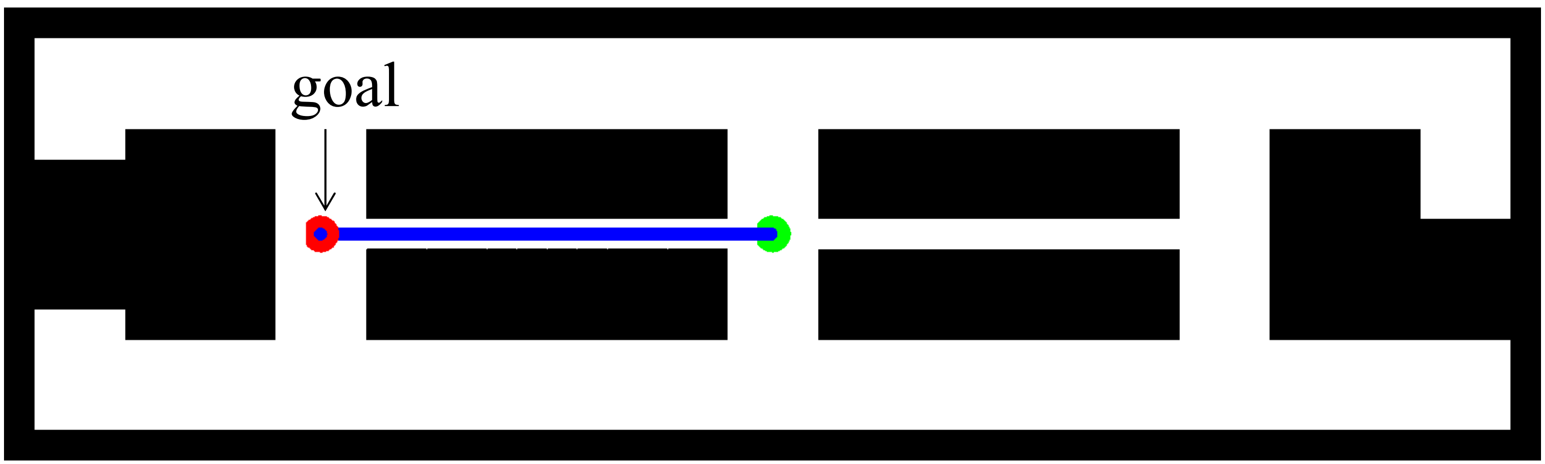
1. probabilistic control
2. potential field
3. graph search

Probabilistic Control Autonomous Mobile Robot

- planning and control objective: *choose the right action*
- selection of action tied to *uncertainty*
  - uncertainty: *action effects* and *perception*
  - action causes uncertainty as action outcomes are non-deterministic
  - robot considers the probabilities of various outcomes
  - robot control also grapples with *anticipated uncertainty*
- result of *planning* phase: *control policy* – prescribes a control policy for situations
  - maps *states* to *actions* for situations (fully observable case)
  - control policy is a *controller* that determines robot actions

Probabilistic Control
Deterministic Action Selection Limits

- with deterministic robotics the robot knows its initial pose and goal location
  - when actions are executed they have predictable effects
    - if this is the case, there is no need to sense!



deterministic case: in the absence of errors in robot motion (e.g. colliding with walls, miss the goal location, etc.), narrow shorter path is superior to longer wider ones

Probabilistic Control
Markov Decision Process

- deterministic planners are often coupled with a sensor-based, reactive controller that incorporates sensor readings to adjust the plan to avoid collisions
  - may have to slow down the robot which makes the narrow path less desirable relative to wider, longer paths
- *Markov decision process* (MDP): encompasses *uncertainty* in *robot motion* $p(z \mid x)$
  - assumes environment can be fully sensed
  - allows for stochastic action effects $p(x' \mid u, x)$
  - planner generates actions for wide variety of situations
    - generates a *policy* for action selection
    - maps states to actions

Probabilistic Control
Markov Decision Process

- **given:**

  states x

  actions u

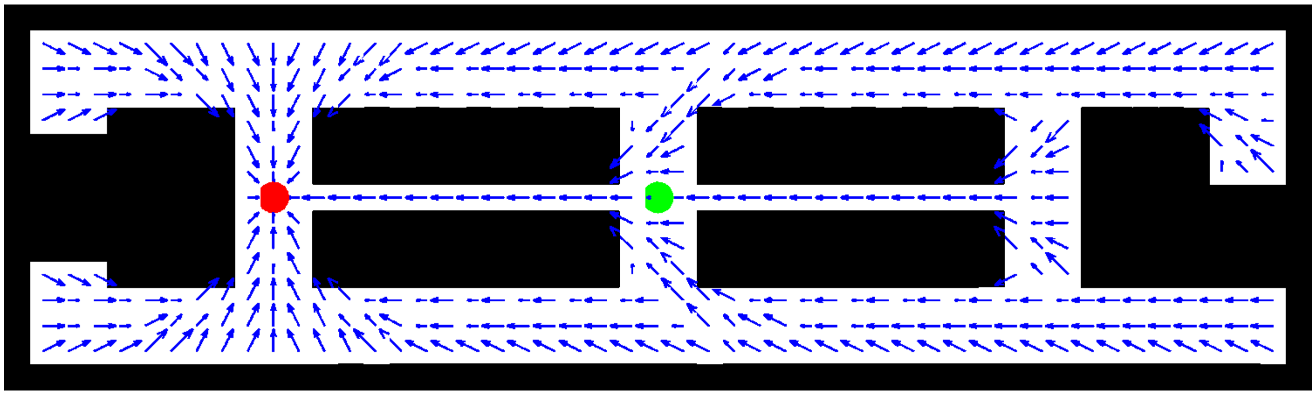
  transition probabilities $p(x'|u,x)$

  reward / payoff function $r(x,u)$

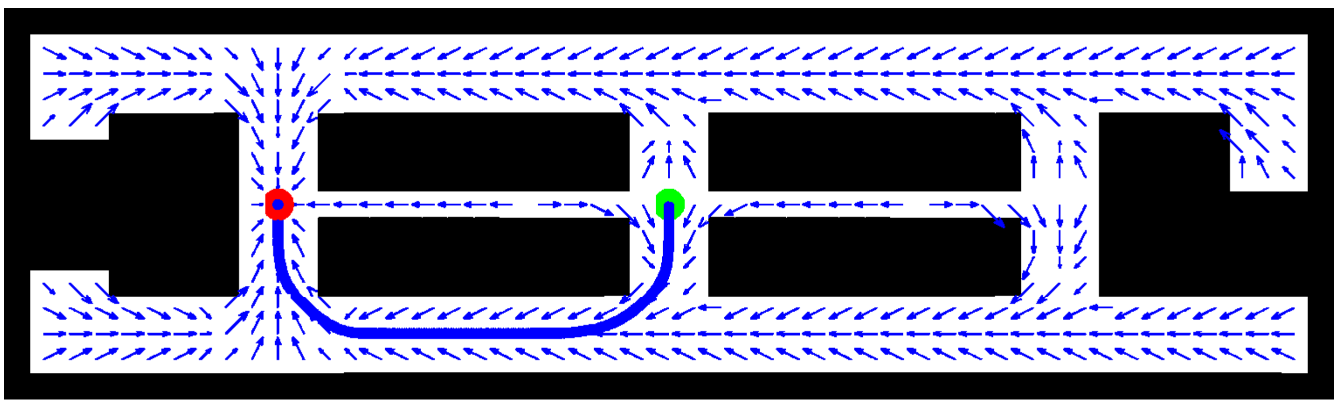
- **want:** policy $\pi(x)$ that maximizes the future expected reward

Probabilistic Control
MDP Control Policy

- deterministic action effects – fine with narrow paths



- non-deterministic action effects – prefer wider (but longer) paths because of the lower collision risk

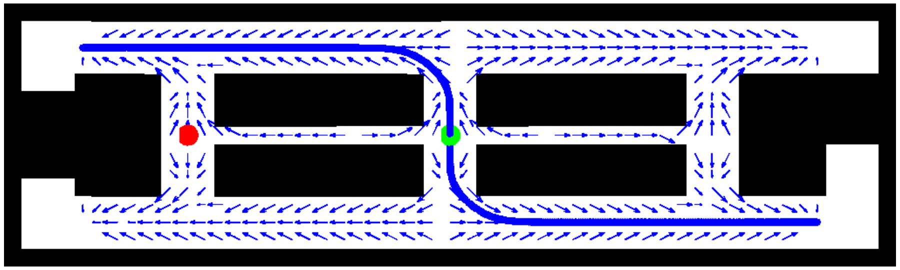


Probabilistic Control
Fully Probabilistic Case

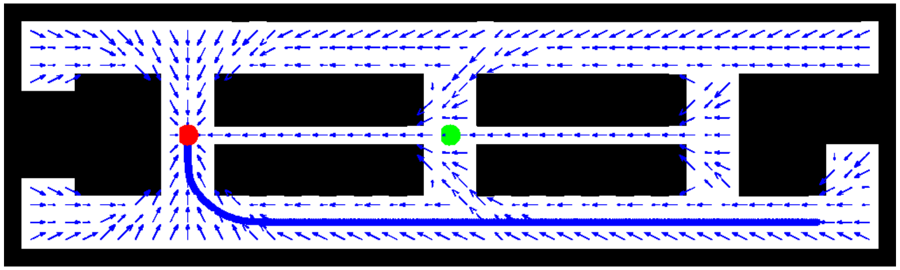
- Partially Observable Markov Decision Process (POMDP): state is not fully observable (i.e. measurable)
  - measurement z is a noisy projection of state x
  - state can be estimated to a certain point
- the optimal plan for previous problem is to head towards a corner which is perceptually sufficiently rich to help determine its orientation

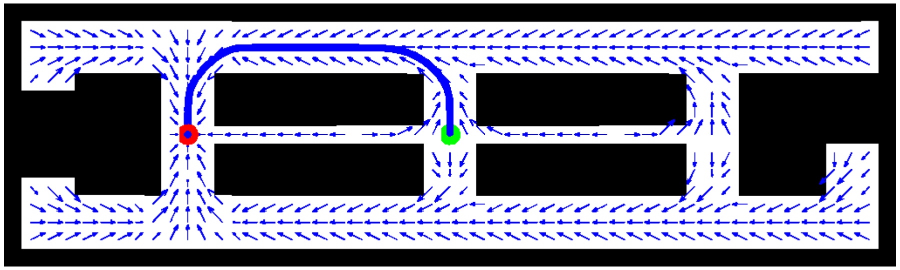
Probabilistic Control
Fully Probabilistic Case

- robot knows initial location, execute either of two plans shown to determine its orientation and hence localize itself



- once localized, can safely navigate to goal location





- robot actively gathers information even if it means a detour
- sensors have intrinsic limitations that limit what the robot can know and where information can be got

Probabilistic Control
Fully Probabilistic Case

- how can one devise a control policy with such uncertainty?
- planning in a partially observable environment cannot be solved through looking at all possible situations
- generate solutions in the *belief space*
  – space of all posterior beliefs the robot holds about world
  – belief space is the 3 panels on previous slide – hence belief space is finite
  – in practice, with finitely many states the belief space is continuous but of finite dimensions
  – if state space is continuous, belief space possesses infinitely many dimensions
  – *a planner must consider the state of its knowledge when making control decisions*

Probabilistic Control
Control Policy Generation

- ability to devise optimal control policies, $\pi(x)$, is an advantage of probabilistic approach vice deterministic, omniscient one
  - increased complexity with the planning problem!

- control policy generates actions to optimize future payoff in expectation $\pi: z_{1:t-1}, u_{1:t-1} \rightarrow u_t$
  - maps past data into controls, or states in controls when the state is observable (measurable by a sensor)
  - fast reactive algorithm that bases its decision on most recent data or an elaborate planning algorithm
  - choose actions so that the sum of all future payoff is maximal- the *expected cumulative payoff*

Probabilistic Control
Control Policy Generation

- *value iteration*: recursively calculates utility of each action relative to a payoff function

- assume uncertainty in robot motion and state of world is observable at all times

- deal with robot motion uncertainty by generating a *policy* for action selection

Probabilistic Control
Value Iteration

- robot action selection is driven by *goals*
- balance reaching a goal configuration with simultaneously optimizing other variables – the *cost*
- *payoff function*: function of state and robot control and it is a single payoff variable for both goal achieved and costs
  - captures trade-off between goal achieved and costs – select actions under uncertainty
  - answers: *is increasing probability of achieving a goal worth the extra effort *e.g. time, energy)?*

Probabilistic Control
MDP Policy and Reward

- policy (uncertainty in motion and perception):

$$\pi: \; z_{1:t-1}, u_{1:t-1} \rightarrow u_t$$

- policy (fully observable case):

$$\pi: \; x_t \rightarrow u_t$$

- expected cumulative payoff:

$$R_T = E\left[\sum_{\tau=1}^{T} \gamma^{\tau} r_{t+\tau}\right] \qquad \gamma^T = \text{discount factor}$$

T is the planning horizon

- $T = 1$: greedy policy
- $T > 1$: finite horizon case, typically no discount
- $T = \infty$: infinite-horizon case, finite reward if discount < 1

Probabilistic Control
MDP Policy and Reward

- expected cumulative payoff of policy:

$$R_T^\pi(x_t) = E\left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau} \mid u_{t+\tau} = \pi(z_{1:t+\tau-1} u_{1:t+\tau-1})\right]$$

- optimal policy:

$$\pi^* = \operatorname*{argmax}_{\pi} R_T^\pi(x_t)$$

- 1-step optimal policy:

$$\pi_1(x) = \operatorname*{argmax}_{u} \; r(x,u)$$

- value function of 1-step optimal policy:

$$V_1(x) = \gamma \max_{u} r(x,u)$$

Probabilistic Control
Value Iteration - Two Step Policies

- optimal policy:

$$\pi_2(x) = \underset{u}{\text{argmax}} \left[r(x,u) + \int V_1(x') p(x' \mid u, x) dx' \right]$$

- value function:

$$V_2(x) = \gamma \max_u \left[r(x,u) + \int V_1(x') p(x' \mid u, x) dx' \right]$$

Probabilistic Control
Value Iteration - T-step Policies

- optimal policy

$$\pi_T(x) = \underset{u}{\operatorname{argmax}} \left[r(x,u) + \int V_{T-1}(x')\, p(x' \mid u,x)\, dx' \right]$$

- value function

$$V_T(x) = \gamma \max_{u} \left[r(x,u) + \int V_{T-1}(x')\, p(x' \mid u,x)\, dx' \right]$$

Probabilistic Control
Value Iteration - ∞ Horizon ($T = \infty$)

- optimal policy:

$$V_{\infty}(x) = \gamma \max_{u} \left[r(x,u) + \int V_{\infty}(x') p(x' \mid u, x) dx' \right]$$

  – Bellman equation

  – fix point is optimal policy

  – necessary and sufficient condition

Probabilistic Control
Value Iteration Algorithm

- for all x do

$$\hat{V}(x) \leftarrow r_{\min}$$

- end_for

- repeat until convergence
  - for all x do

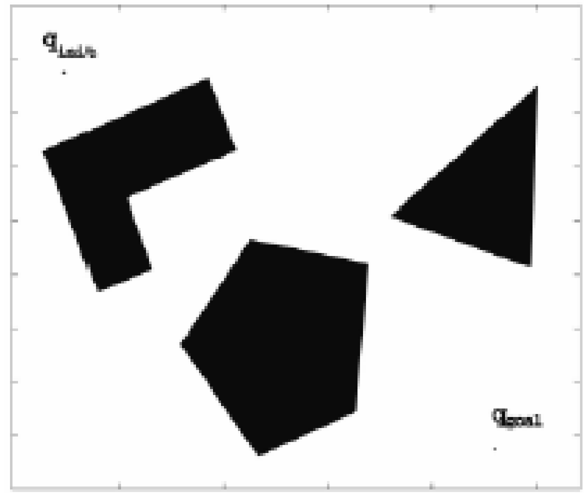
$$\hat{V}(x) \leftarrow \gamma \max_{u} \left[r(x,u) + \int \hat{V}(x')\, p(x' \mid u,x)\, dx' \right]$$

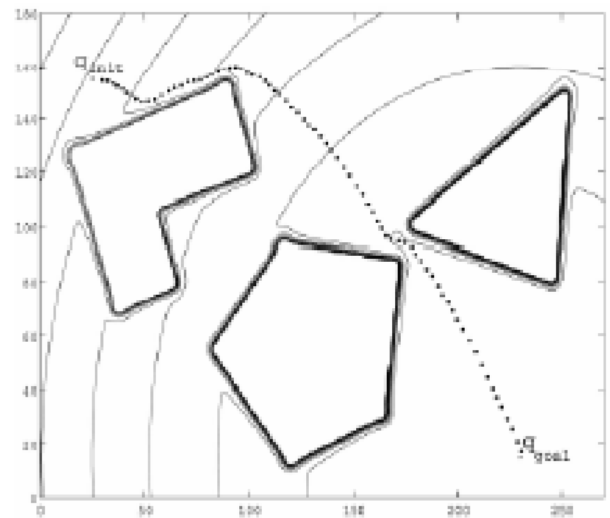
  - end_for
- end_repeat

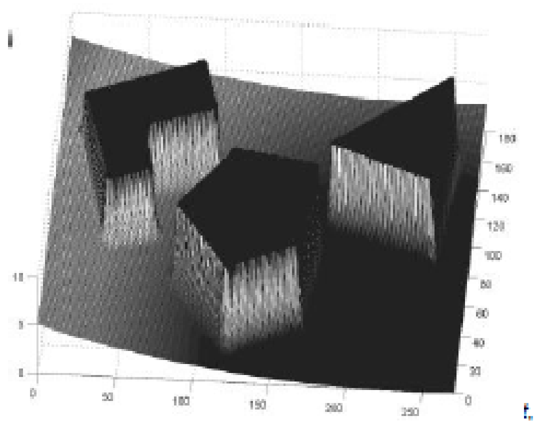
$$\pi(x) = \operatorname*{argmax}_{u} \left[r(x,u) + \int \hat{V}(x')\, p(x' \mid u,x)\, dx' \right]$$

Potential Field

- robot treated as a point under the influence of artificial potential field
- operates in the continuum
  - generated robot motion similar to a ball rolling down the hill
  - goal generates attractive force
  - obstacles generate repulsive force







Potential Field Generation

- generation of potential field function $U(q)$
  - attracts to goal and repulses from obstacle fields
  - sum the fields
  - functions must be differentiable
- generate artificial force field $F(q)$

$$F(q) = -\nabla U(q) = -\nabla U_{att}(a) - \nabla U_{rep}(q) = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix}$$

- set robot speed (v_x, v_y) α to $F(q)$ generated by the field
  - force field drives the robot to the goal
  - robot is assumed to be a point mass
  - yields robot plan and control

Potential Field
Attractive Force

- parabolic function representing Euclidean distance $\rho_{goal} = | q - q_{goal} |$ to the goal:

$$U_{att}(q) = \frac{1}{2} k_{att} \rho_{goal}(q)^2$$

$$= \frac{1}{2} k_{att} (q - q_{goal})^2$$

- attracting force converges linearly toward 0 (goal)

$$F_{att}(q) = -\nabla U_{att}(q)$$

$$= k_{att} \cdot (q - q_{goal})$$

Potential Field Repulsing Force

- should generate a barrier around all obstacles
  - the field is strong when robot close to the obstacle
  - does not feel influence if robot far from the obstacle

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

- $\rho(q)$ is the minimum distance to the object
- the field is positive or zero and $\rightarrow \infty$ as q approaches the object

$$F_{rep}(a) = -\nabla U_{rep}(q) = \begin{cases} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_o} \right) \frac{1}{\rho^2(q)} \times \frac{q - q_{obstacle}}{\rho(q)} & \text{if } \rho(q) \le \rho_o \\ 0 & \text{if } \rho(q) \le \rho_o \end{cases}$$

- problem more complex if robot is not considered a point mass

Graph Search

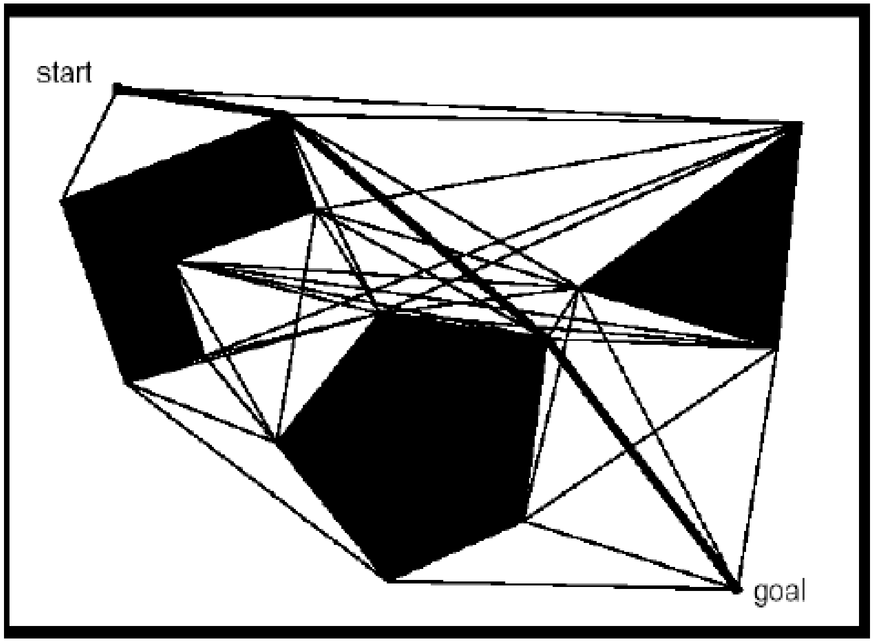
- overview
  - solves a least cost problem between two states on a graph
  - graph structure is a discrete representation
- limitations
  - state space discretized so completeness not guaranteed
  - feasibility of paths if often not inherently encoded
- algorithms:
  - breadth first
  - depth first
  - Dijkstra
  - A* and variants

Graph Construction

- transform continuous environment model into a discrete map for path planning algorithm
  - have to construct this discrete map
- methods used for the pre-processing steps
  - visibility graph
  - Voronoi diagram
  - cell decomposition

Graph Construction
Visibility Graph

- suited for polygon shaped obstacles
- shortest path length- robot as close as possible to obstacles that are on the way to obstacles
  - optimal length of solution path
- grow obstacles to avoid collisions



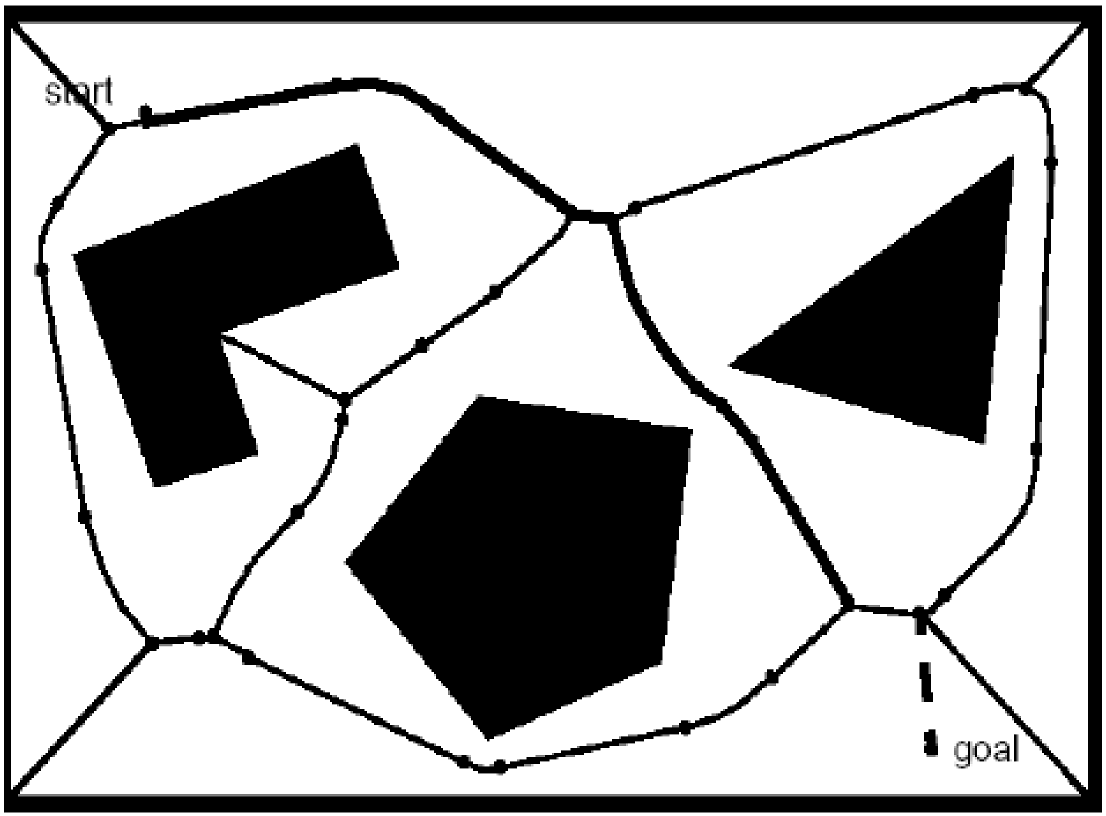
Graph Construction
Visibility Graph

- pros
  - path is optimal because it is the shortest length path
  - implementation simple when obstacles are polygons

- cons
  - solution path tends to take the robot as close to the obstacles as possible
    - grow obstacles > robot radius
  - number of edges and nodes increases with the number of polygons
    - inefficient in densely populated environments

Graph Construction
Voronoi Diagram

- tends to maximize distance between robot and obstacles
- easily executable, maximize sensor readings

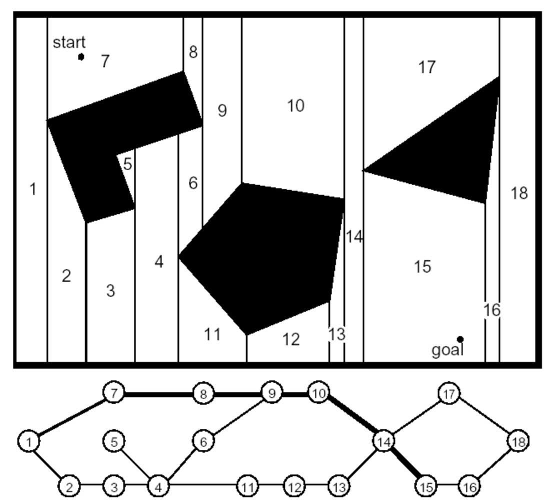


Graph Construction
Voronoi Diagram

- pros
  - with a range sensor like laser or sonar, robot can navigate along the Voronoi diagram using simple control rules
- cons
  - since the robot is as far as possible from obstacles, short range sensors may not work
- peculiarities
  - for polygonal obstacles, the Voronoi map consists of straight and parabolic segments

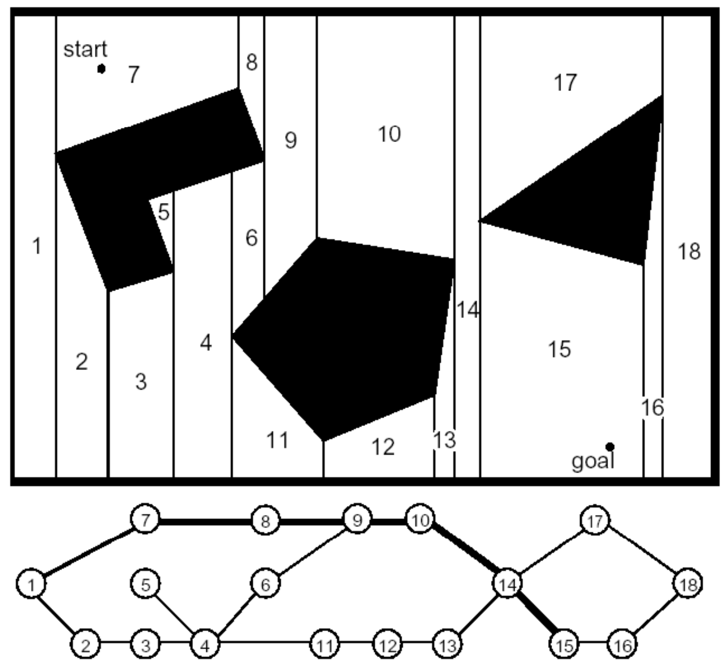
Graph Construction
Cell Decomposition

- divide space into simple, connected regions – cells that are either free or occupied; construct a connectivity graph
- find cells where initial and goal configuration (state) lie and search for a path in the connectivity graph to connect them
- from sequence of cells found compute path within each cell
  - e.g. passing through the midpoints of cell boundaries or by sequence of wall following movements
- types of cell decomposition:
  - exact cell decomposition
  - approximate cell decomposition
    - fixed cell decomposition
    - adaptive cell decomposition



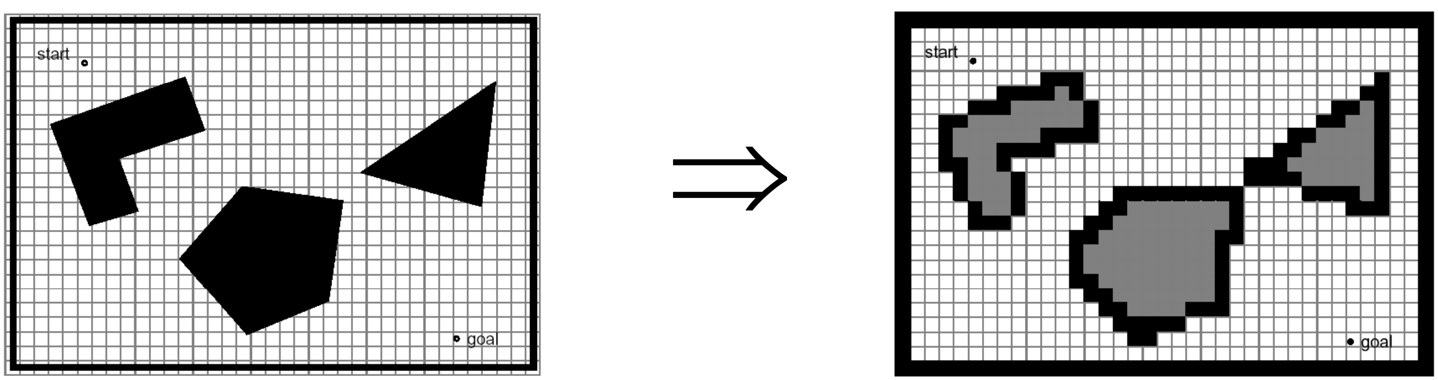
Graph Construction
Exact Cell Decomposition

- when boundaries between cells are a function of the structure of the environment
- cells are completely free or completely occupied
- specific robot position within each cell of free space does not matter, but rather the robot’s ability to from free cell to adjacent free cell



Graph Construction
Approximate Cell Decomposition

- popular for mobile robot path planning as grid-based environmental representations are generally popular
- fixed sized cells – can make it fairly small → low computational complexity
- e.g. *grassfire* algorithm uses wave front expansion from the goal position out, marking each cell's distance to the goal until it reaches the initial robot position cell
  - then planner links the best adjacent cells together

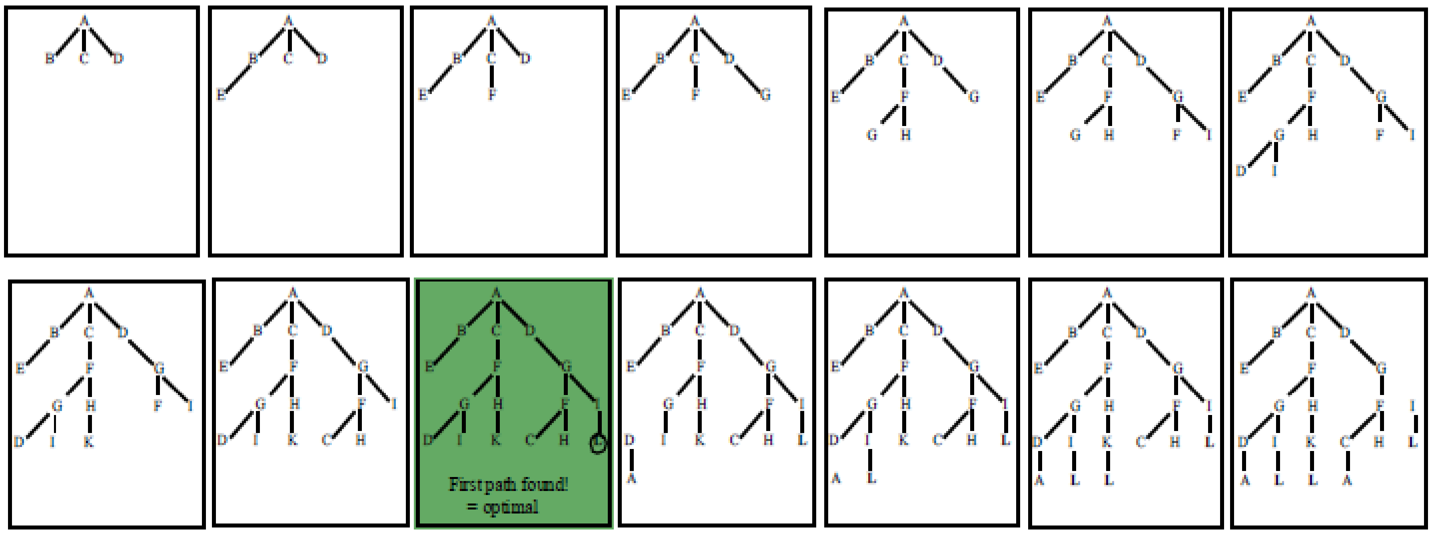


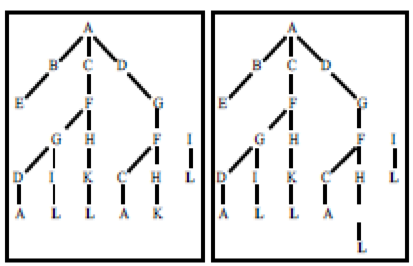
Graph Search

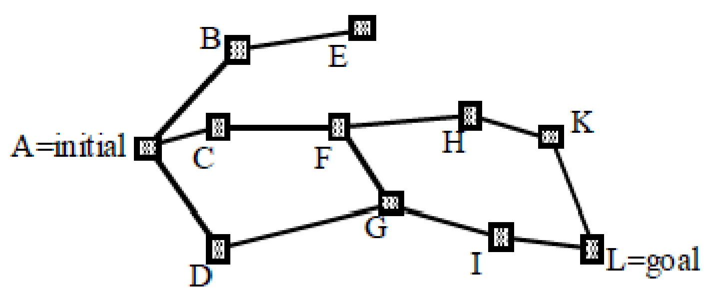
- methods
  - breadth first
  - depth first
  - A* and variants
  - D* and variants

Graph Search
Breadth First Strategies

- starts at root node then exhaustively explores neighbouring nodes until it finds the goal

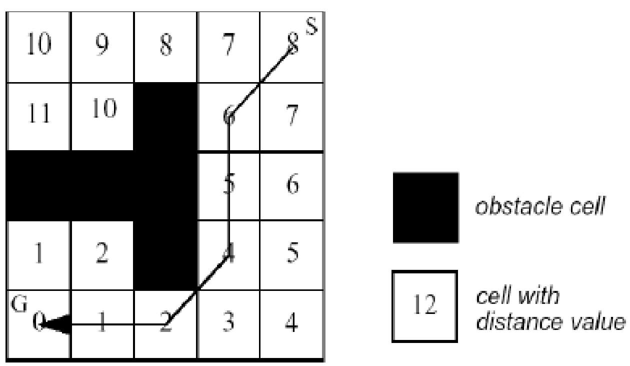






Graph Search
Breadth First Strategies

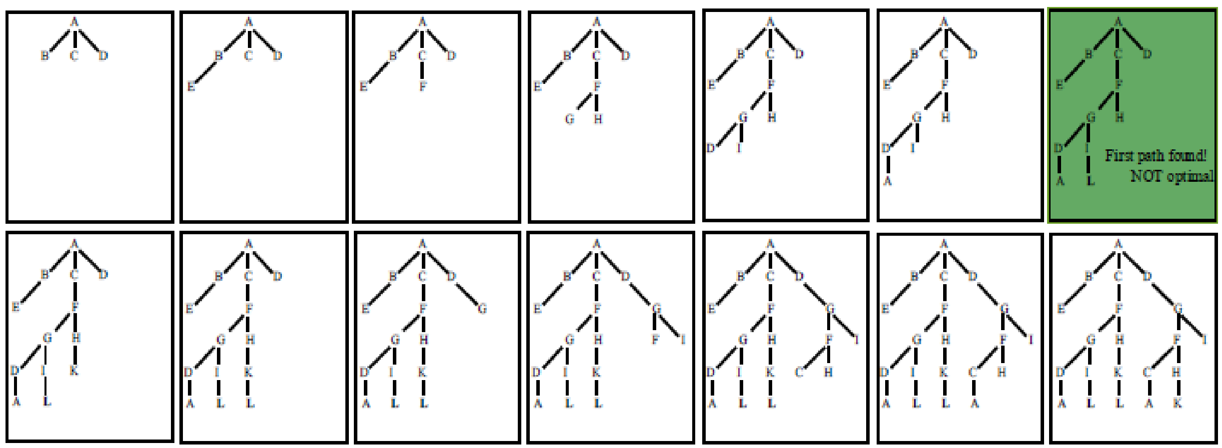
- corresponds to a wavefront expansion on a 2D grid
- use of a first-in-first-out queue for adding child nodes
  - first found solution optimal if all edges have equal costs
- Dijkstra's search is an '*g(n)*-sorted' HEAP variation of breadth first
  - first found solution guaranteed to be optimal no matter the cell cost

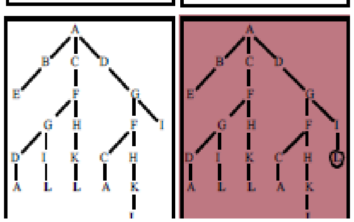


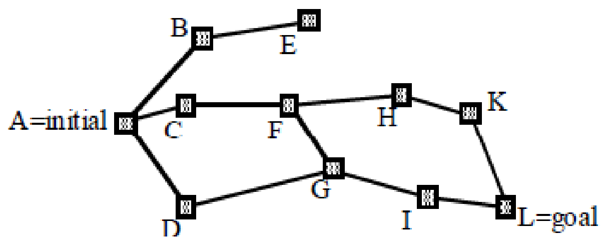
resulting path generated by grassfire path planner after cell decomposition. S = start, G = goal

Graph Search
Depth First Strategies

- explores each branch completely before backtracking to explore most recent node it has not finished exploring
- use of a LIFO (last-in-first-out) queue
- memory efficient (fully explored sub-trees can be deleted)

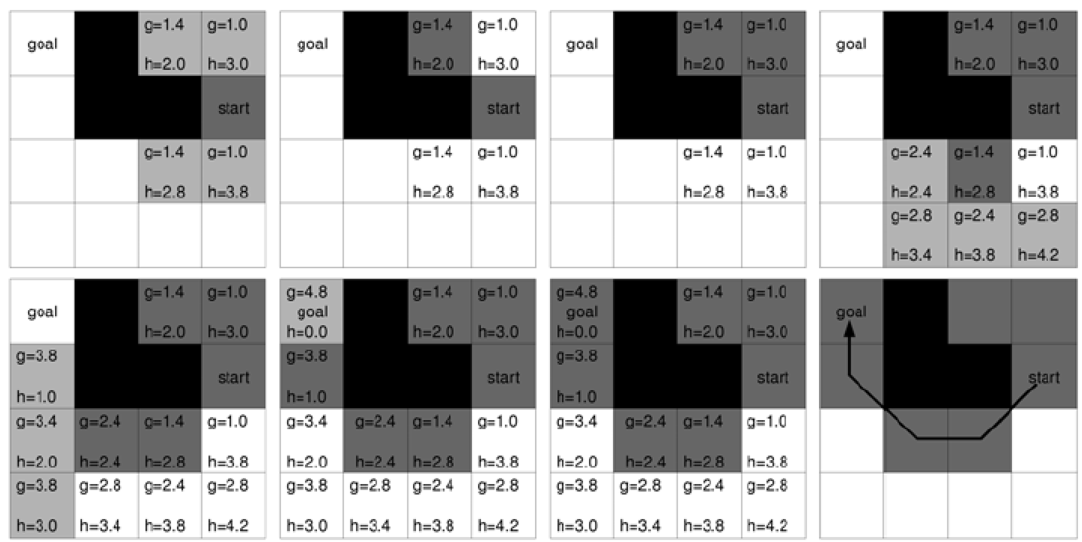






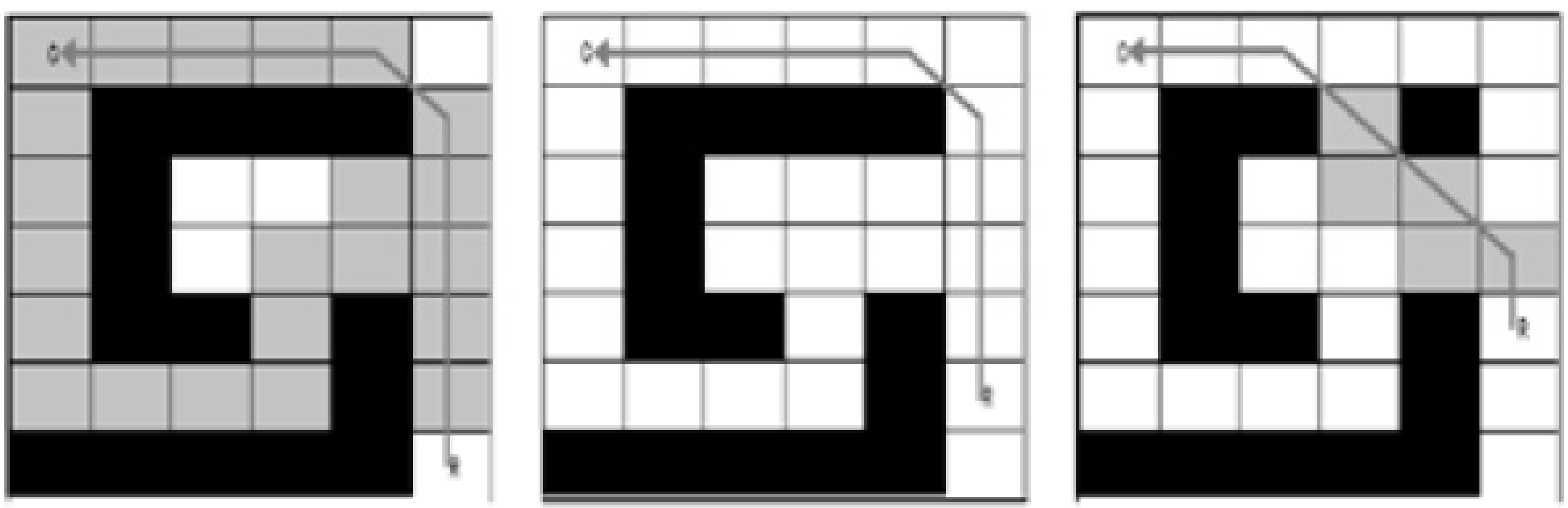
Graph Search
A^* Strategies

- widespread use – good performance and accuracy
- looks for least path cost, $f(n)$, from an initial to a goal point
- similar to Dijkstra's algorithm, A^* also uses a HEAP ($f(n)$ sorted)
- A^* uses heuristic function $h(n)$ – Euclidean distance
- $g(n)$ is the path cost function for going from initial point to current point
- $f(n) = g(n) + \varepsilon h(n)$



Graph Search
D* Strategies

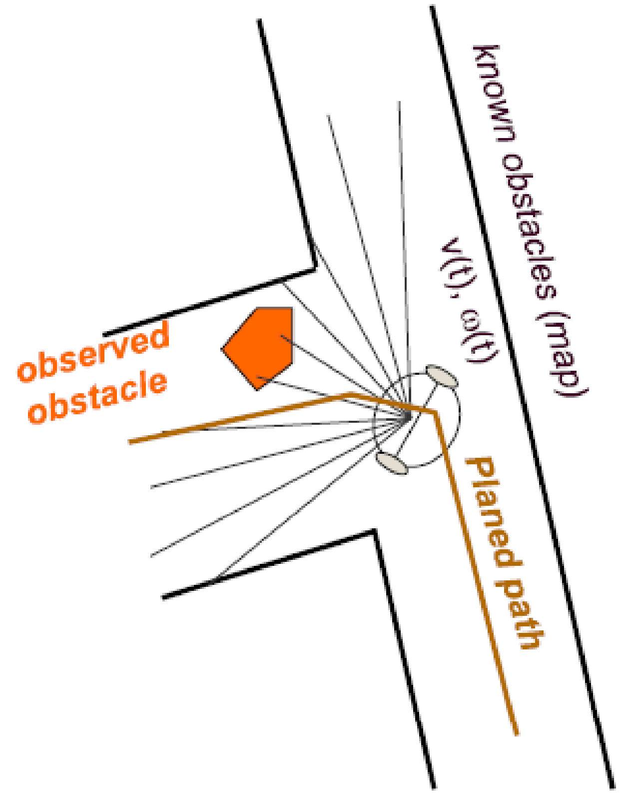
- similar to A* except search starts from goal outward
- $f(n) = g(n) + \varepsilon h(n)$
- first pass identical to A*
- subsequent passes re-use info from previous searches



C M. Likhachev

Obstacle Avoidance
Local Path Planning

- objective: avoid collisions with obstacles
  – based on a local map
- optimize obstacle avoidance with respect to
  – overall goal
  – actual speed and kinematics of the robot
  – on-board sensors
  – actual and future risk of collisions



Vector Field Histogram (VFH)

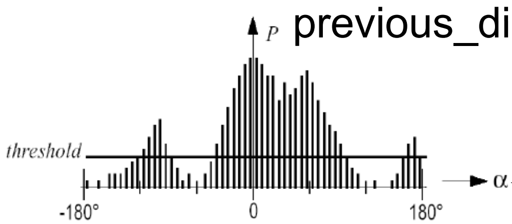
- unlike Bug algorithms, environment rep as occupancy grid map around robot with relatively recent range readings
  - cell values equivalent to probability there is an obstacle
- histogram of angle α obstacle found vs probability P there is an obstacle in that direction based on occupancy grid value
  - steering direction computed from this as follows:
    - all openings that robot can pass through are found
    - one with lowest cost function, G, is selected

$$G = a \cdot \text{target\_direction} + b \cdot \text{wheel\_orientation} + c \cdot \text{previous\_directon}$$

target_direction = alignment of robot path with goal

wheel_orientation = diff tween new direction and current wheel orientation

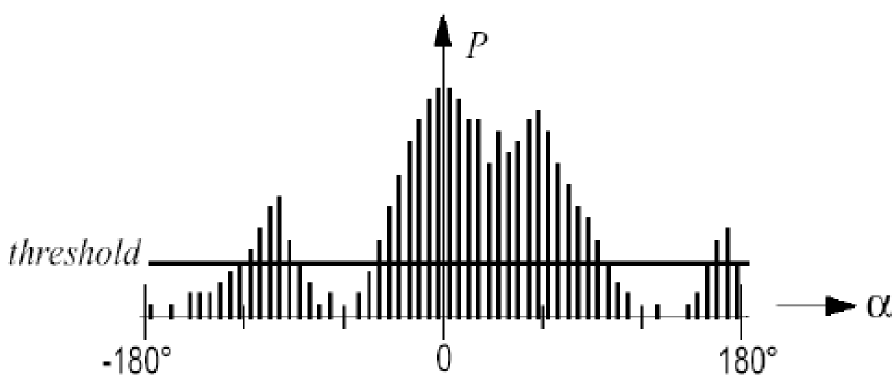
previous_direction = diff tween previously selected direction and new direction



ic Term

Vector Field Histogram (VFH)

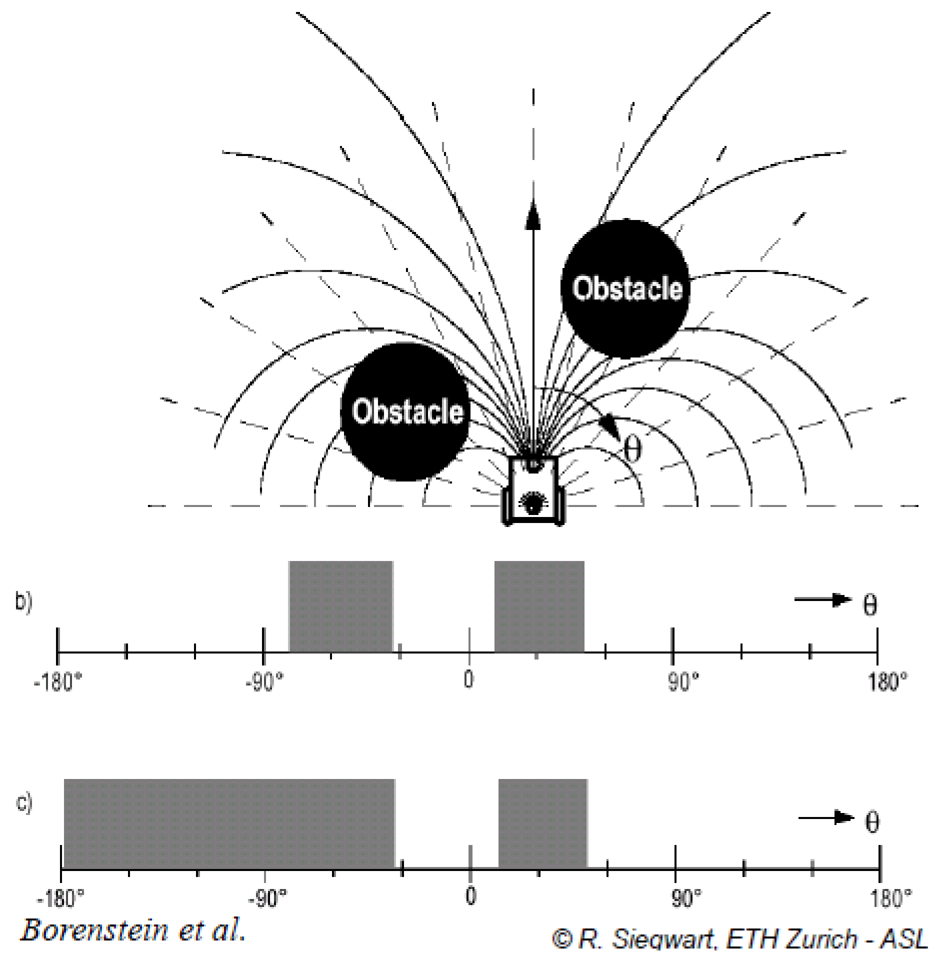
- fast and robust, good in densely populated obstacle areas
- consecutive angles with a polar obstacle density below a threshold is selected based on the proximity to the goal
- once direction determined, robot angle is steered to match



polar histogram on direction to steer robot to

Vector Field Histogram

- accounts in simplified way for vehicle kinematics
  - robot moves in arcs or straight segments
  - obstacles blocking a travel direction also blocks all the arcs and straight segments through this direction
  - obstacles enlarged kinematically so blocked trajectories taken into account



Vector Field Histogram Limitations

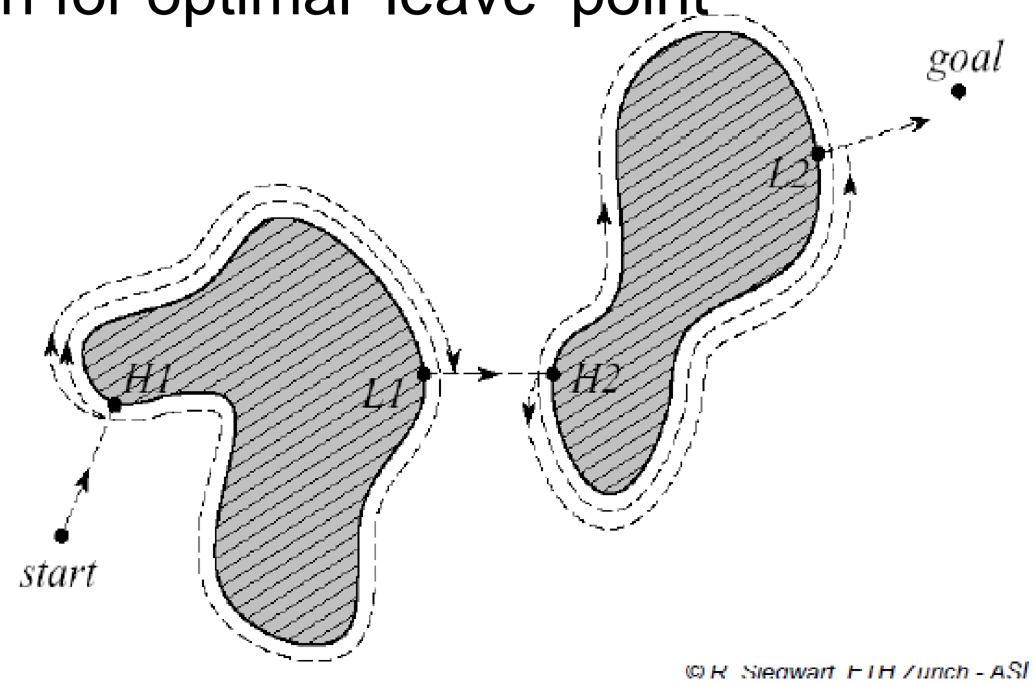
- passing through narrow openings (like doorways)
- local extrema
- no guarantee that goal is reached
- robot dynamics not captured

Bug Algorithms

- applied to 2D terrestrial robots
- require only local environment knowledge and a global goal
- simple behaviors:
  - follow a boundary (right or left side)
  - head in straight line for the goal
- sensor based planners
- robot is modelled as a point moving around on a plane with a contact sensor (short range) to detect obstacles
- assume perfect positioning
- robot can measure distance between any two points
- Bug 1 and Bug 2 algorithms use tactile (local) sensing
- Tangent Bug uses the sensor feedback to find better paths

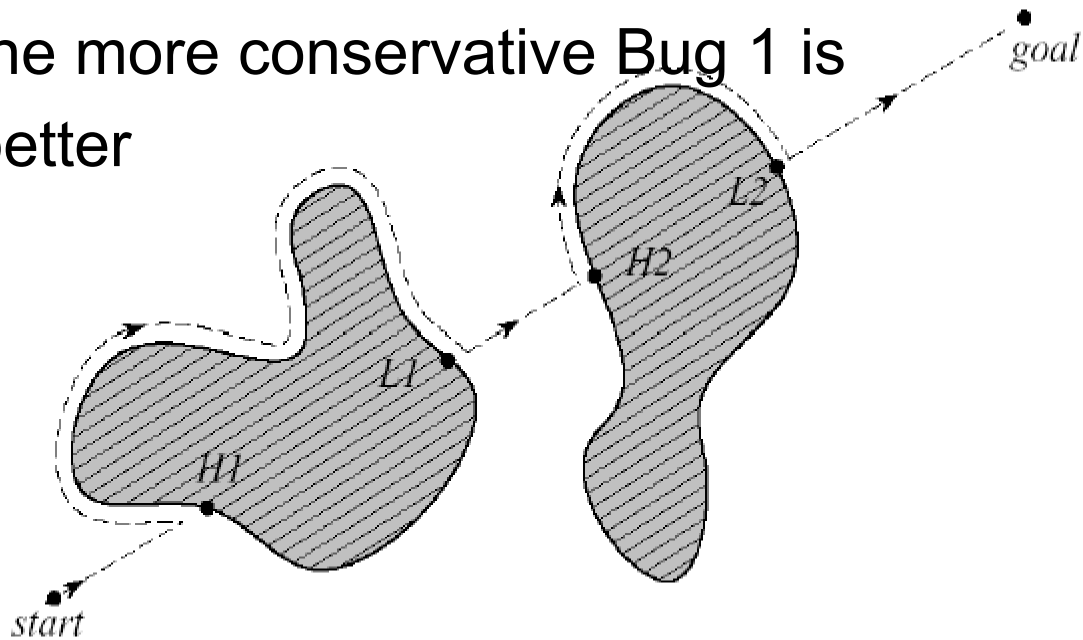
Bug 1 Algorithm

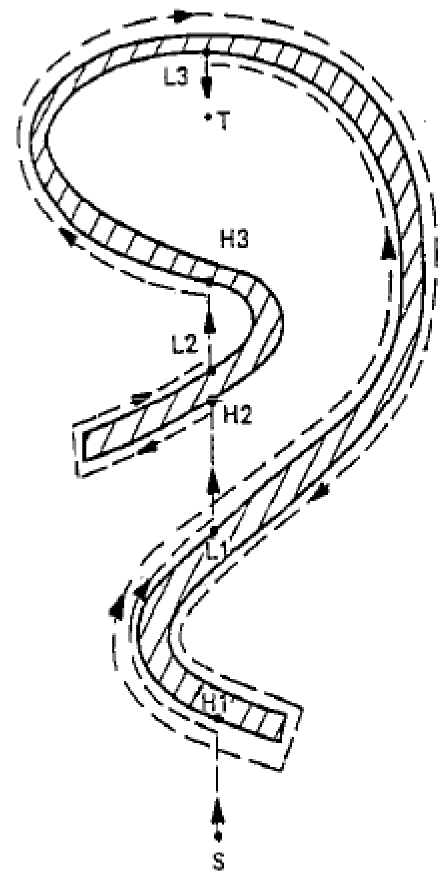
- “common sense” approach
- contour around the obstacle and remember closest point of approach (CPA) to the goal
- return to CPA and head straight for goal
- performs exhaustive search for optimal ‘leave’ point
- advantages
  - no global map required
  - completeness satisfied
- disadvantage
  - solution not optimal



Bug 2 Algorithm

- follow the obstacle on one side till the direct connection between start and goal is crossed
- uses an opportunistic or *greedy* approach for optimal leave point
  – good when obstacles are simple o/w the more conservative Bug 1 is better





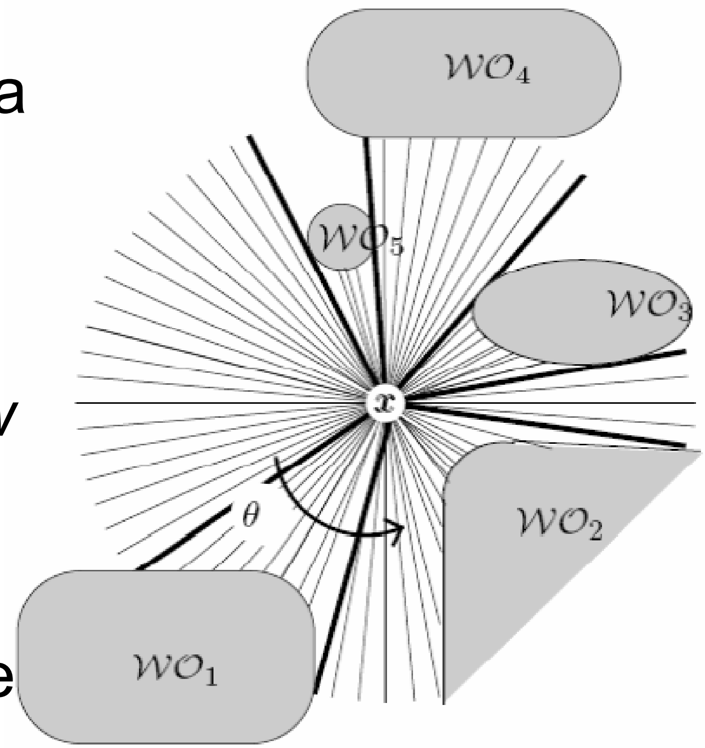
© R. Siegwart, ETH Zurich - ASL

Tangent Bug

- improvement over Bug2 since it determines a short path to goal using a range sensor with 360 degrees of infinite azimuthal angle orientation resolution
  - range sensor modelled with the *raw distance function*

$$\rho : \Re^2 \times S^1 \rightarrow \Re$$

  - $\rho(x,\theta)$ is distance to closest obstacle along the ray from x at angle θ
  - S^1 is circle that accounts for θ *which wraps at 2π*



Tangent Bug

Saturated Raw Distance Function

- sensors have limited range so define a *saturated raw distance function*, ρ_R :

$$\rho_R : \Re^2 \times S^1 \rightarrow \Re$$

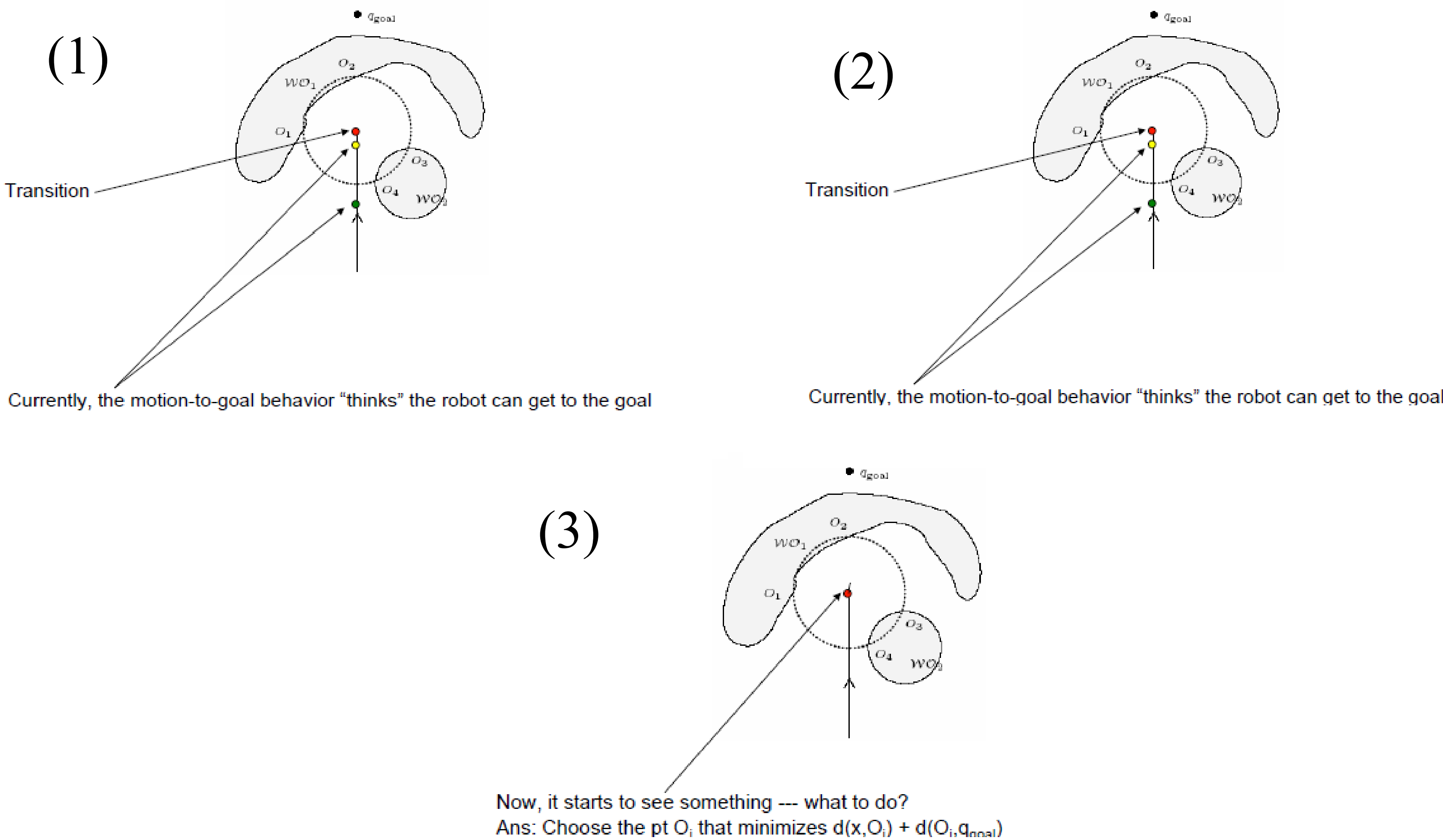
- which is the same as ρ when the obstacle is within sensing range, R, and is ∞ when ray lengths $>$ R, i.e.

$$\rho_R(x,\theta) = \begin{cases} \rho(x,\theta), & \text{if } \rho(x,\theta) < R \\ \infty, & \text{otherwise} \end{cases}$$

- Tangent Bug planner can detect discontinuities in ρ_R due to transitions in obstacles blocking one another or rays extending into infinity because the are > R

Tangent Bug
Motion to Goal Transition

-

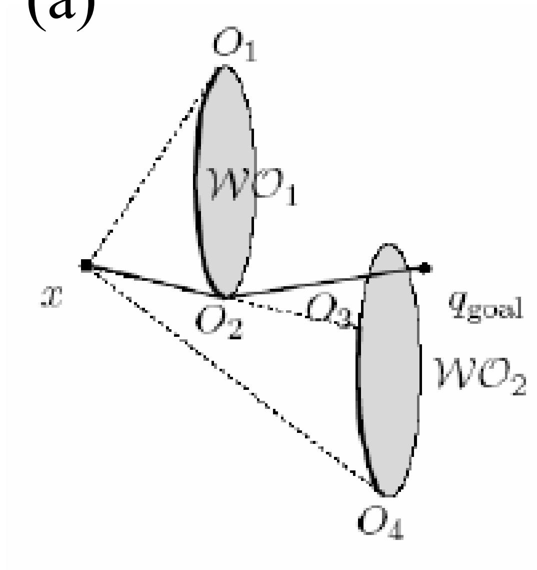


Tangent Bug

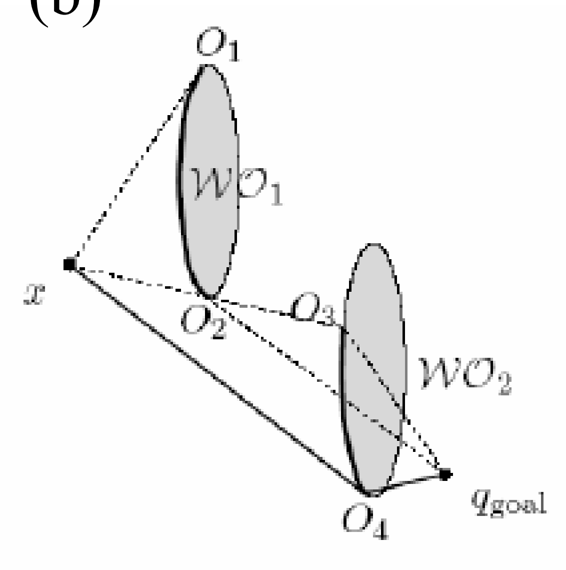
(a) planner selects O_2 as robot subgoal, $i=2$ minimizes $d(x,O_i) + d(O_i, q_{goal})$; at x, cannot know WO_2 blocks path from O_2 to q_{goal}

(b) planner selects O_4 as robot subgoal

(a)

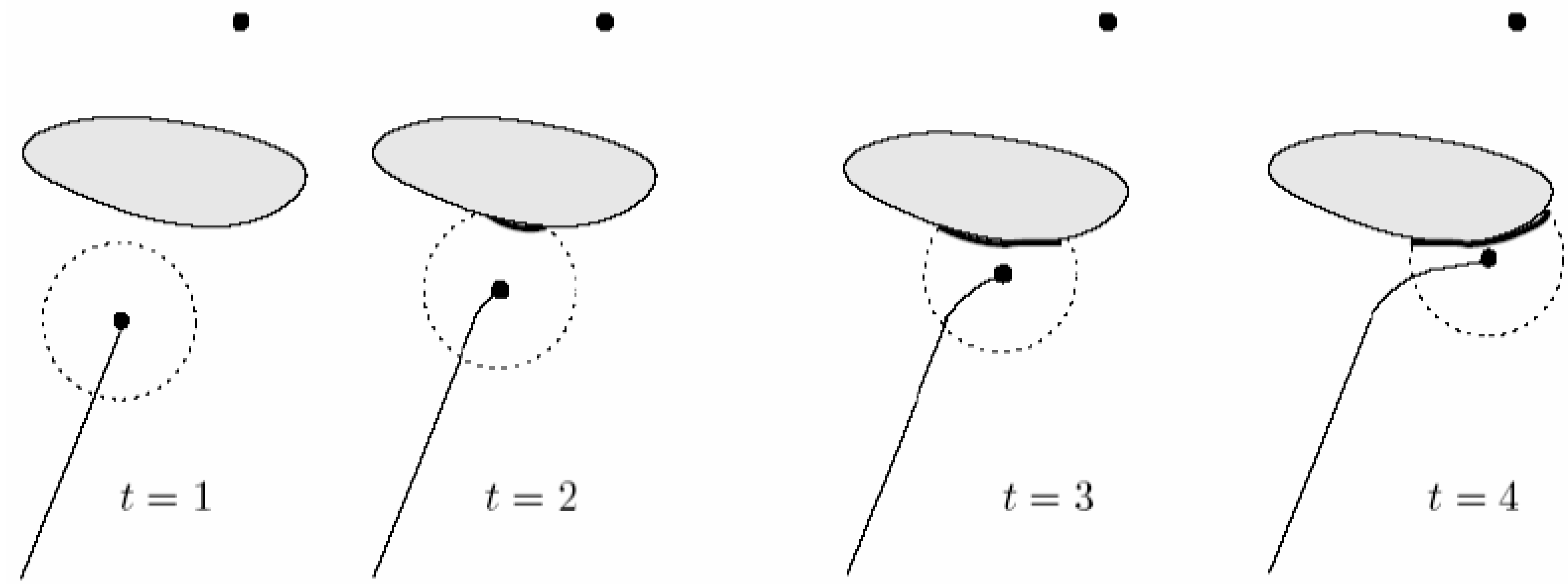


(b)



Tangent Bug

- motion to goal behavior for robot with finite range sensor moving toward goal (black dot) – continue till it makes no progress getting closer to goal



obstacle not sensed yet, move to goal

obstacle sensed, move to right & still closing in on goal

robot senses more of obstacle, continues closing in on obstacle by hugging obstacle boundary

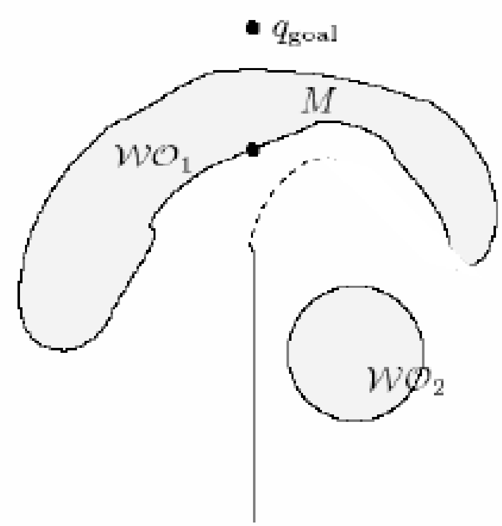
Tangent Bug

- when robot switches to boundary following looks for point, M, on sensed portion of obstacle which is the shortest distance on the *followed obstacle* (as opposed *to blocked obstacle*) to the goal

Choose the pt O_i that minimizes $d(x,O_i) + d(O_i,q_{goal})$

Problem: what if this distance starts to go up?

Ans: start to act like a BUG and follow boundary



M is the point on the "sensed" obstacle which has the shorted distance to the goal

Followed obstacle: the obstacle that we are currently sensing

Blocking obstacle: the obstacle that intersects the segment $(1-\lambda)x + \lambda q_{goal} \quad \forall \lambda \in [0,1]$

They start as the same

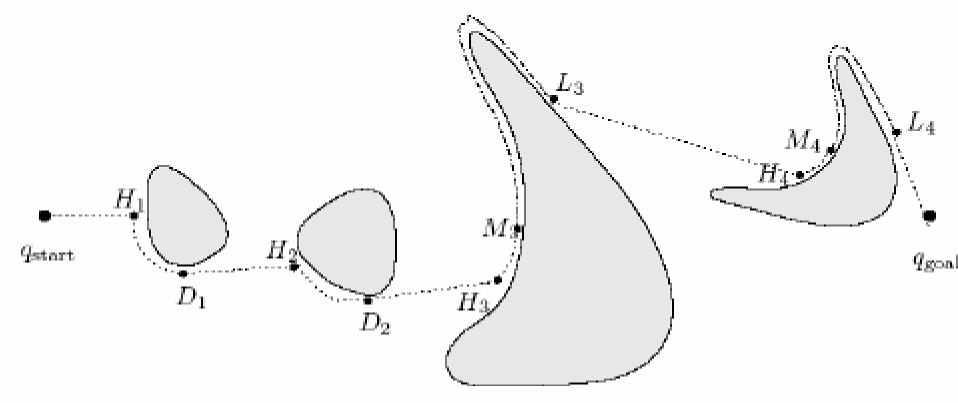
- here, blocking obstacle = followed obstacle

Tangent Bug

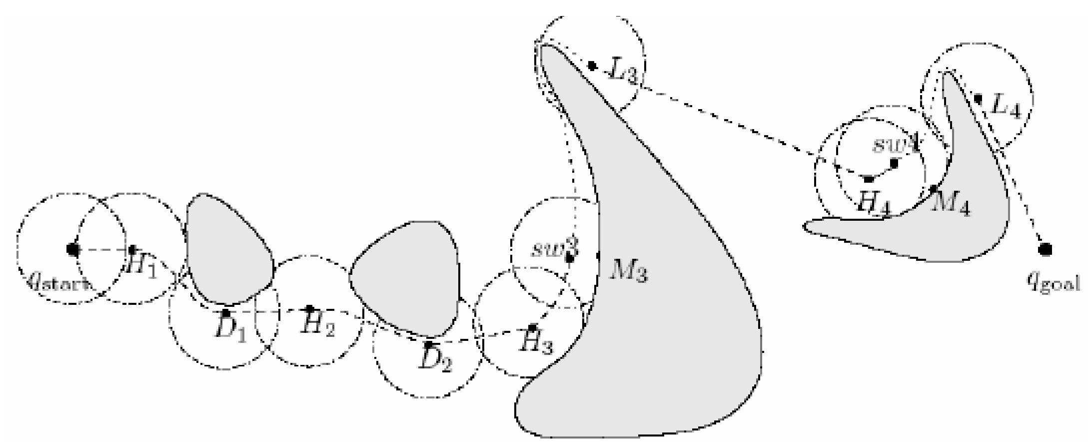
- in this mode, planner continually updates: $d_{followed}$ and d_{reach}
  - $d_{followed}$ = shortest distance between sensed boundary and goal
  - d_{reach} = distance between the goal and closest point on followed obstacle that is within robot line-of-sight
- when d_{reach} < $d_{followed}$, robot stops boundary following behavior

Tangent Bug

- comparison of planned paths for a zero range sensor and finite range sensor



zero range sensor

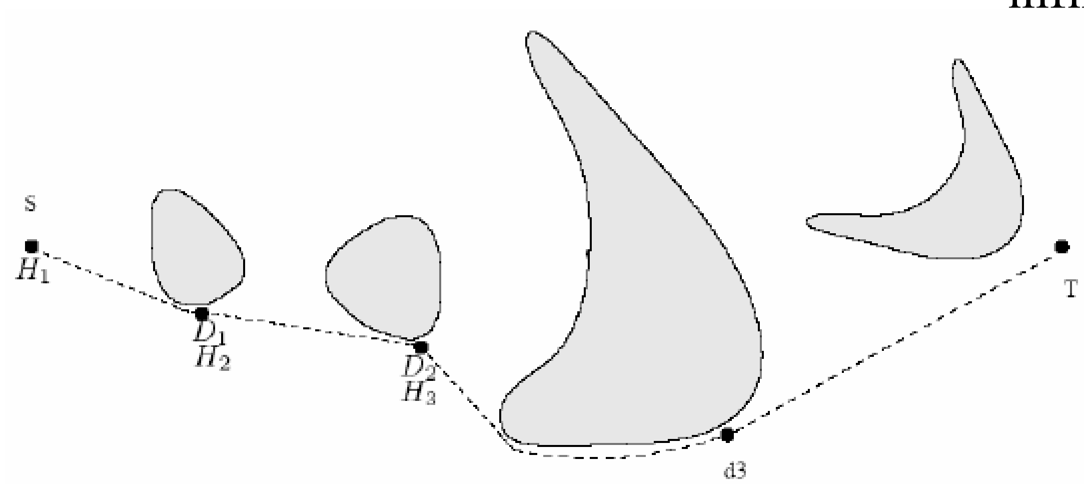


finite range sensor

Tangent Bug

- infinite range sensor

infinite range sensor



Concluding Remarks

- path-planning and navigation for mainly two-dimensional robots
- break the problem down into global (deliberative) path planning and local (reactive) obstacle avoidance
- global path planning implementations can use probabilistic control, potential function, and graph searches
- local obstacle avoidance have many algorithms the two studied here are vector field histograms and Bug
- assignment #4 investigates Bug algorithms within MATLAB®