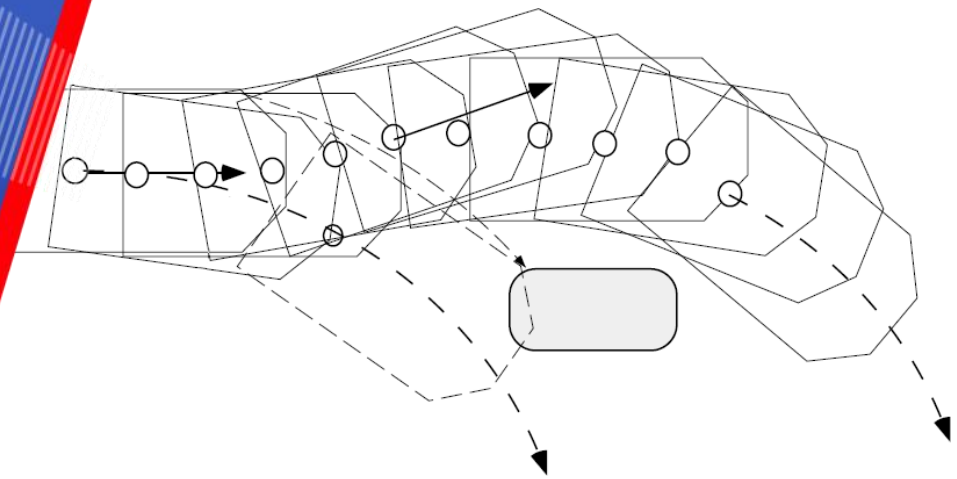
# Autonomous Mobile Robots

# 9. Navigation



**Prof. Karsten Berns**
Robotics Research Lab
Department of Computer Science
University of Kaiserslautern, Germany

# Contents

- Introduction
- Local path planning
- Global path planning

# Introduction

# Navigation

- Task: Drive vehicle through its environment
- Local path planning: Definition of course points, respect vehicle dimensions and kinematic constraints
- Global path planning: Find path from start point to goal point using a given representation of the environment

# Local Path Planning

# Local Path Planning

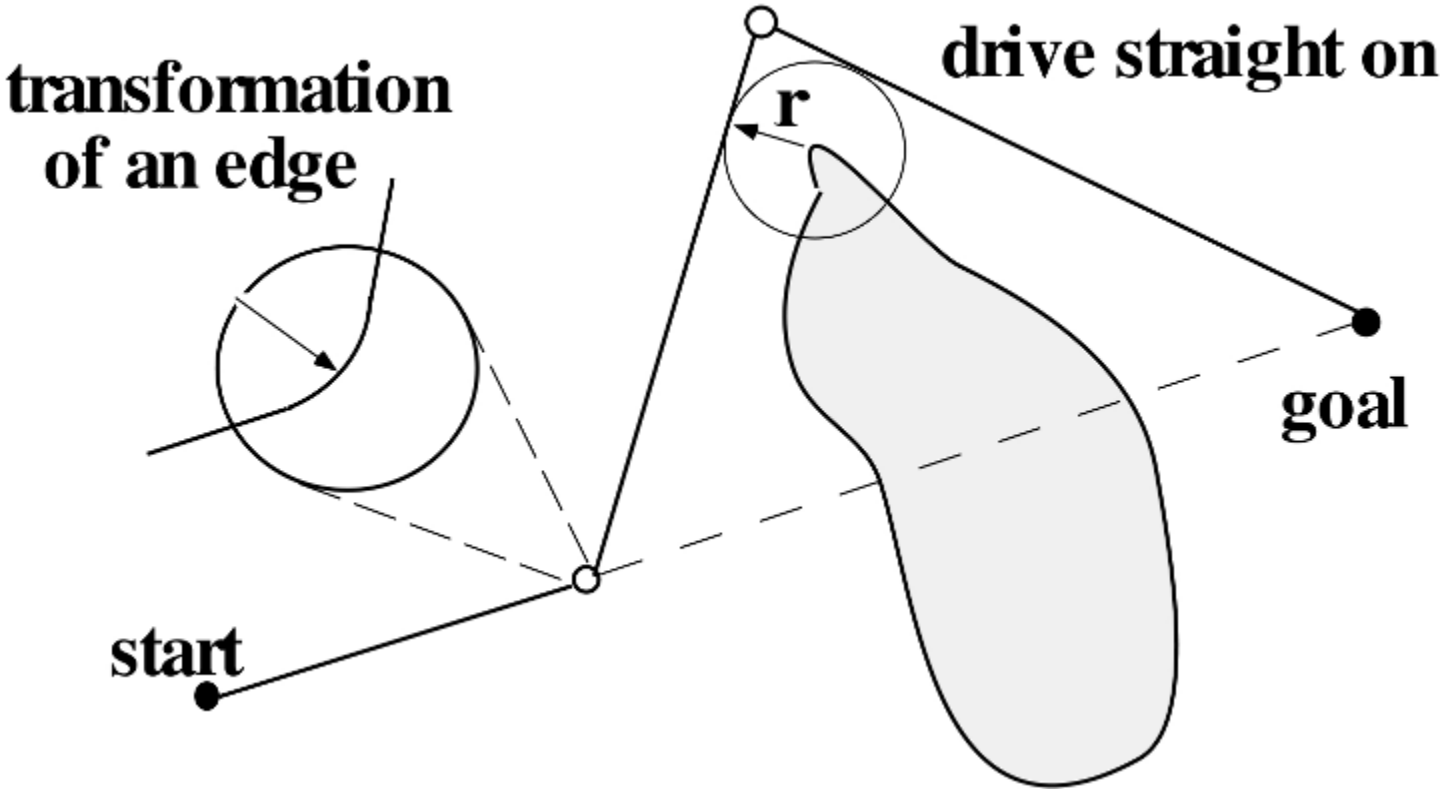
- Traverse a given environment
- Requirements for vehicle
  - Driving an s-curve
  - Forward and backward docking
  - Path planning under geometric restrictions imposed by the vehicle itself
  - Calculating velocity and turning rates

# Large Free Space

- Free space is large in comparison to vehicle dimensions

- Minimal distance to an obstacle may be $d_{min} \gg$ vehicle dimensions

# Geometrical Maps with Large Free Space




Path around an obstacle

# Geometrical Maps with Large Free Space

- Draw circle with $r = d_{min}$ at the most protruding part of the obstacle
- First course point: Go towards the goal as far as possible such that a tangential line from there to the circle comes nowhere closer to the obstacle than $d_{min}$
- Let the same be true for the tangential line from the goal. Then a second course point is the crossing point of both tangents
- Between course points the vehicle can drive straight on
- Kinematic restrictions: curved path at points

# Plan Path around Obstacles

(**) **if** the line cuts through an obstacle **then**

Look for points farthest away from the line

Take the side with the point of smallest distance to the line

Draw a circle with $r = d_{min}$ around this point;

(*)

Draw the tangents to this circle from $B$ and $E$. They will meet at point $M = P_0$

Check the lines $B - M$ and $M - E$ for obstacles going to (**)

**else if** there is no obstacle on the way **then**

Look for the point of minimal distance to the line

Draw a circle with $r = d_{min}$ around this point

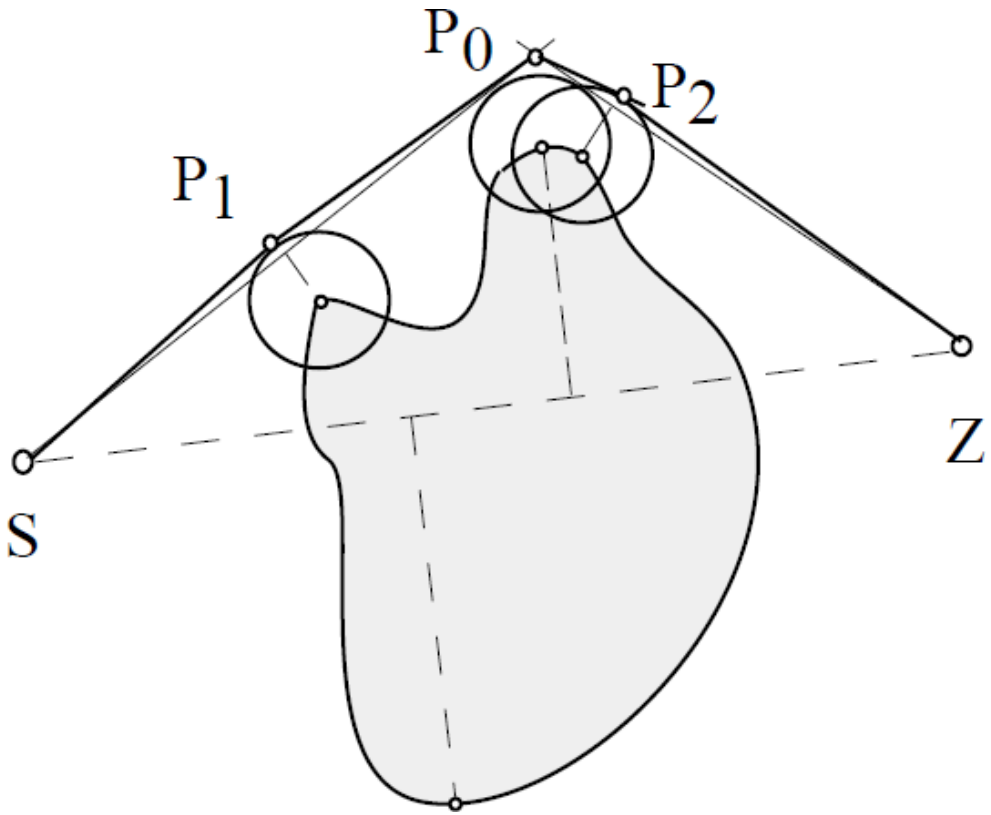
**if** the circle cuts the line **then**

Go to (*)

**end if**

**end if**

# Plan Path around Obstacles

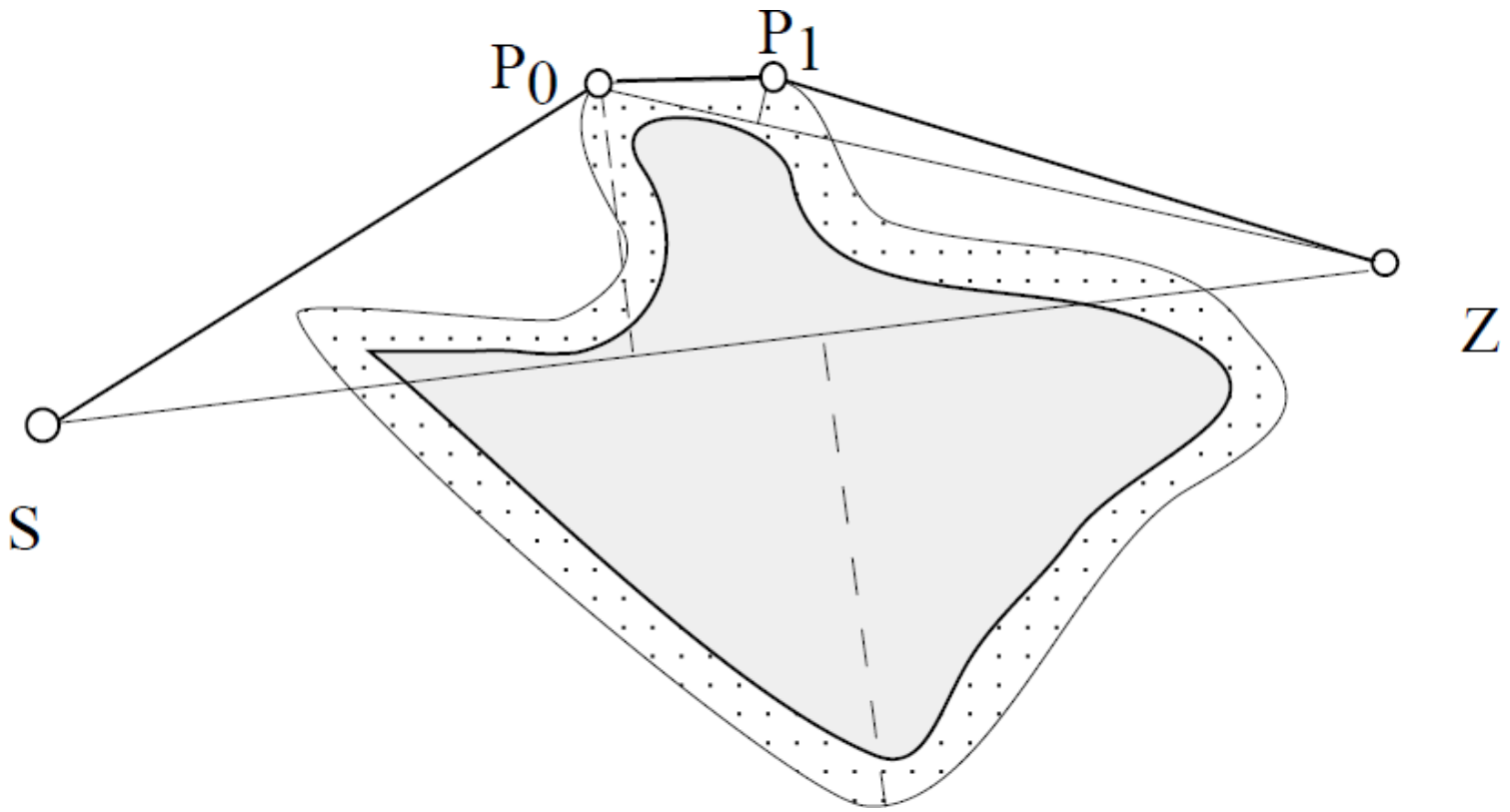



Planning a course around an obstacle

# Growing Obstacles

- Trick to plan a safe (collision-free) path around obstacles keeping minimal distance
- Assumption: Round vehicle with $r < d_{min}$
- Idea: Grow obstacles by $d_{min}$ [Lozanoperez79]
- Consequence: Calculation of course points without regarding tangens or minimal distances

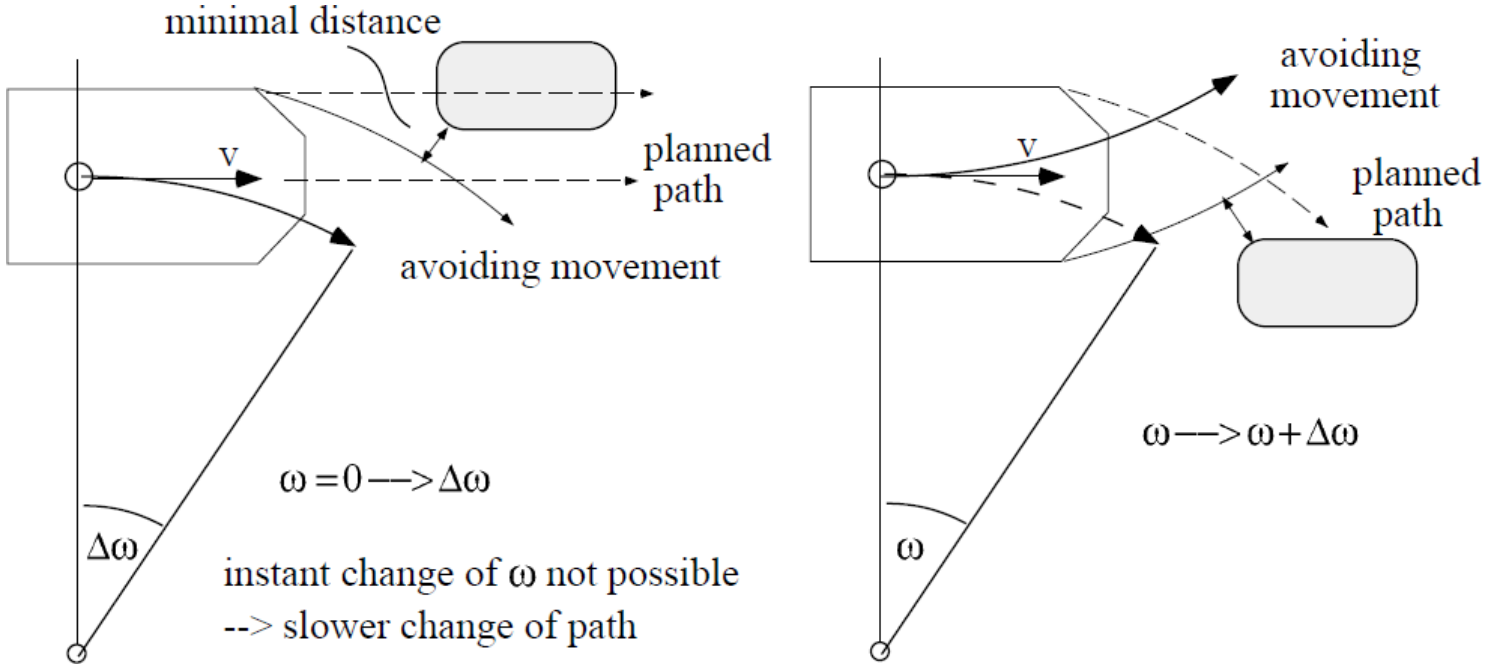
# Growing Obstacles




Path planning using grown obstacles

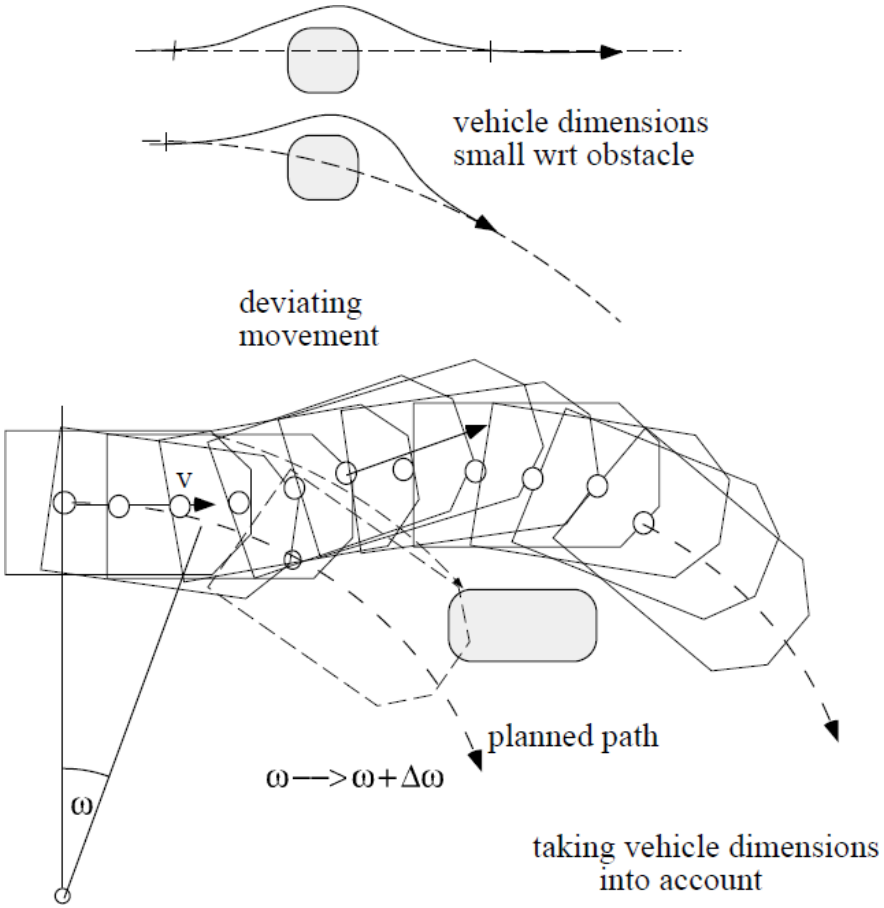
# Driving around Obstacles

- Basic ability of the AMR: drive around obstacles i.e. generate $\omega$ for given $v$ in a way that path leads around the obstacle
- No unsteady changes in $\omega$ possible → slow change in path




Movements to avoid obstacles

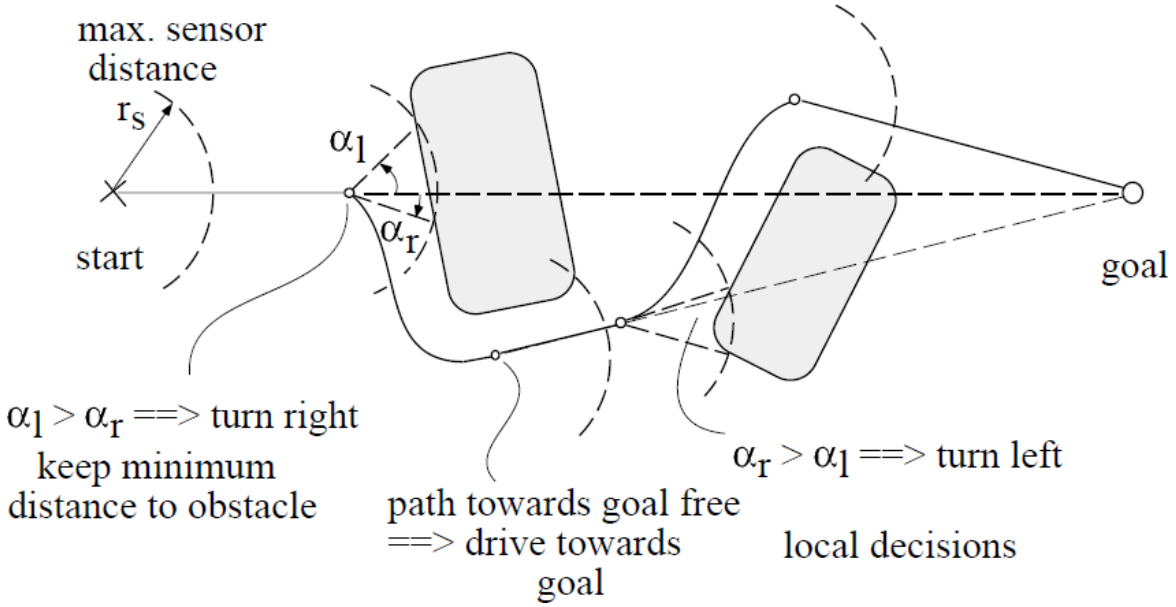
# Problem: Path Adjustment




Surrounding an obstacle

# Wandering Standpoint Algorithm

- Given
  - Robot with ranging sensor (max. range $r_s$)
  - Direction to a target
- Local decisions ok
- Global decisions?




Wandering standpoint algorithm

# Wandering Standpoint Algorithm

$(*)$

Run towards the goal until the goal is reached

**if** the sensor detects an obstacle at a distance $d > r_0$ **then**

Drive towards the obstacle till $r_s < r < r_0$

Measure the angles $\alpha_l$ and $\alpha_r$ to the obstacle

Turn towards the smaller angle and keep the distance to the obstacle constant

**else**

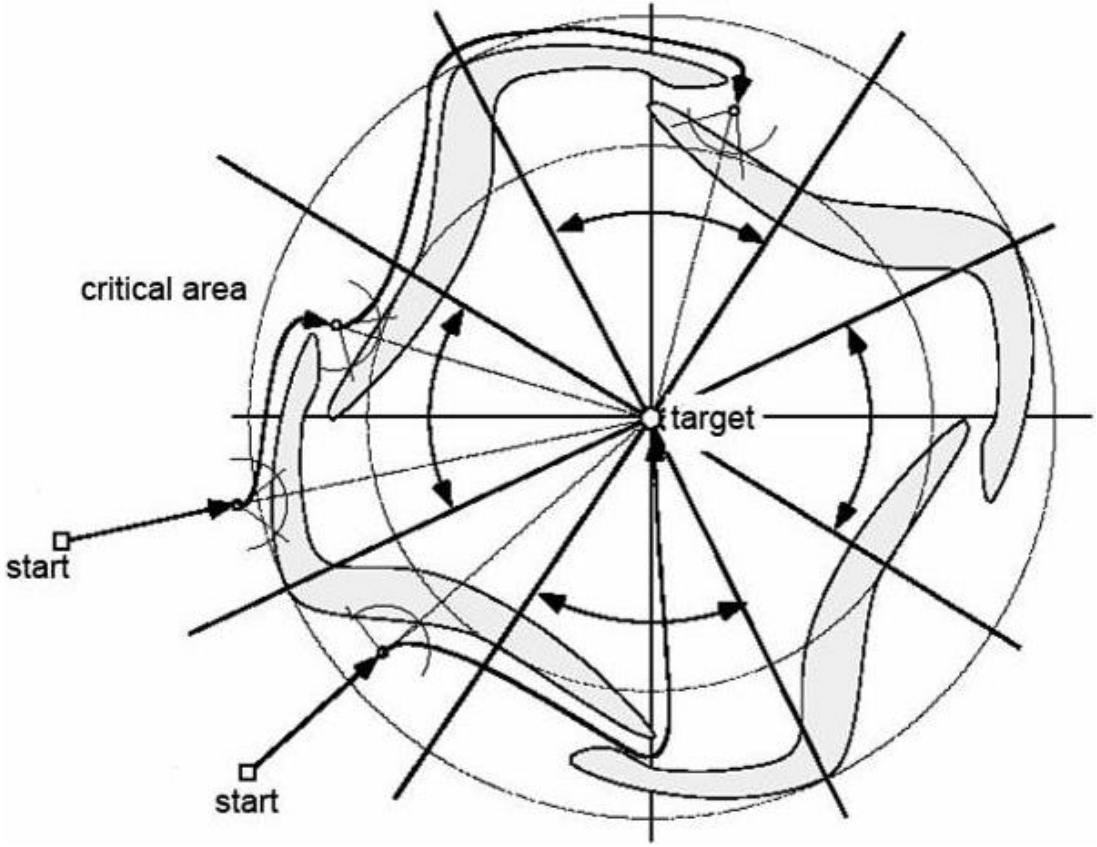
Go to $(*)$ // The direction to the obstacle is free again

**end if**

# WSA: Problems

- Apart from pathological situations, this strategy will find a way to the goal
- There are two possible cases of failure
  - A movable obstacle may push the vehicle away from the goal like a defender pushes away an attacking player from the goal
  - A pathological form of the obstacle lets the algorithm fail: The form of the border line lets the vehicle drive until the goal can be seen again, but the distance to the goal remains constant, though the vehicle went straight towards the goal for a while.

# WSA: Problems



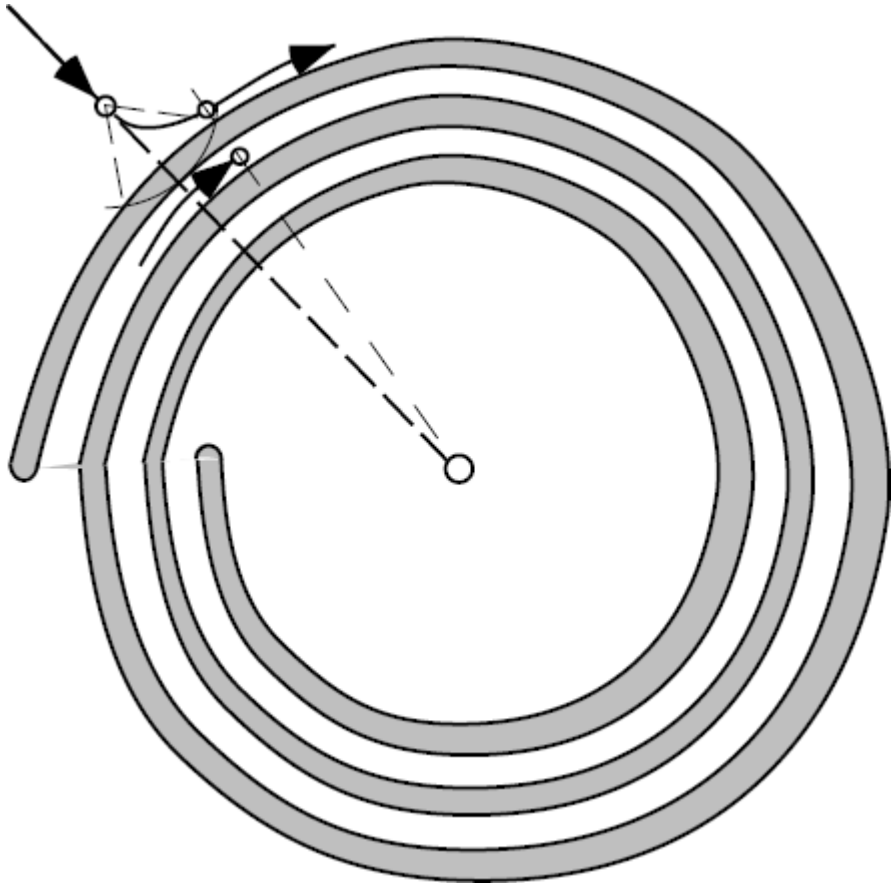

Problematic obstacle constellation for the WSA

# WSA: Problems

In order to cope with this case, the vehicle should know its overall turning.

Having turned $360°$ either the distance to the goal has become smaller then the vehicle is spiralling inwards, or the distance stood constant.

Then for a moment prefer to turn to the larger angle: if there is a way towards the goal this altered strategy will find it.



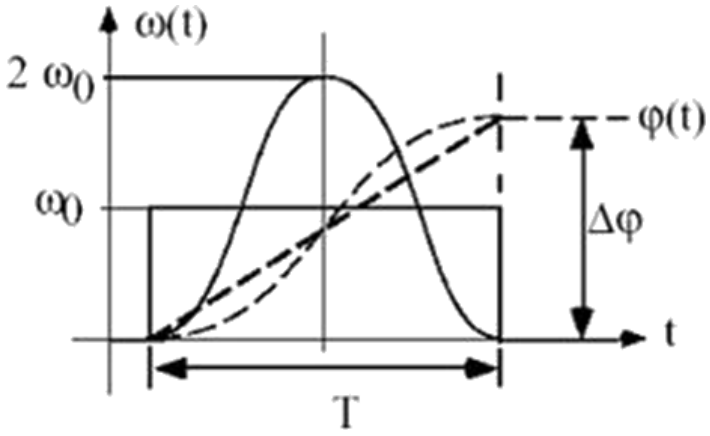
An obstacle in form of a spiral

# Basic Ability "Trajectory Drive"

- Task: drive along curved trajectory (pivot angle $\Delta\varphi$)
- Given

$$\omega(t) = \omega_0 \left(1 - \cos\frac{2\pi}{T} \cdot t\right)$$

$$\Delta\varphi = \int_0^T \omega(\tau)\, d\tau$$



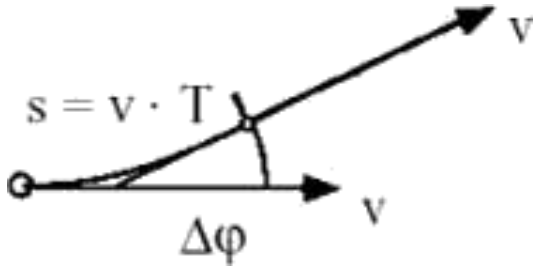

- Limitation of angular velocity

$$\frac{d\omega}{dt} = \omega_0 \cdot \frac{2\pi}{T} \cdot \sin\frac{2\pi}{T} \cdot t$$

$$\max \omega' = \omega_0 \cdot \frac{2\pi}{T}$$

$$\Delta\varphi = \int_0^T \omega_0 \left(1 - \cos\frac{2\pi}{T} \cdot t\right) d\tau = \omega_0 \cdot T$$

$$\Rightarrow \omega_0^2 = \Delta\varphi \cdot \frac{\omega'}{2\pi} \cdot T^2 = 2\pi \cdot \frac{\Delta\varphi}{\omega'}$$



# Example

- Given

$$v = 1\frac{m}{s},\ \omega_0 = 45\,°,\ \Delta\varphi = 90\,° \Rightarrow \mathrm{T} = \frac{\Delta\varphi}{\omega_0} = 2s \Rightarrow s = 2m$$

- Approximated calculation: circular arc with radius $r$

$$s = \frac{2\pi r}{4} = 2\,m \Leftrightarrow r = \frac{4}{\pi}\,m \approx 1.275\,m$$

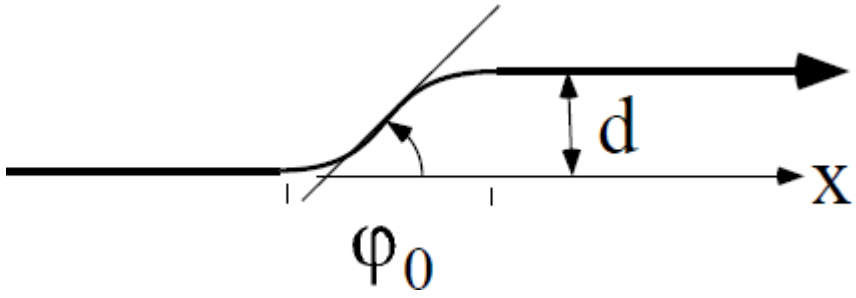
- At the beginning of the avoidance motion the AMR approaches the obstacle with approx.

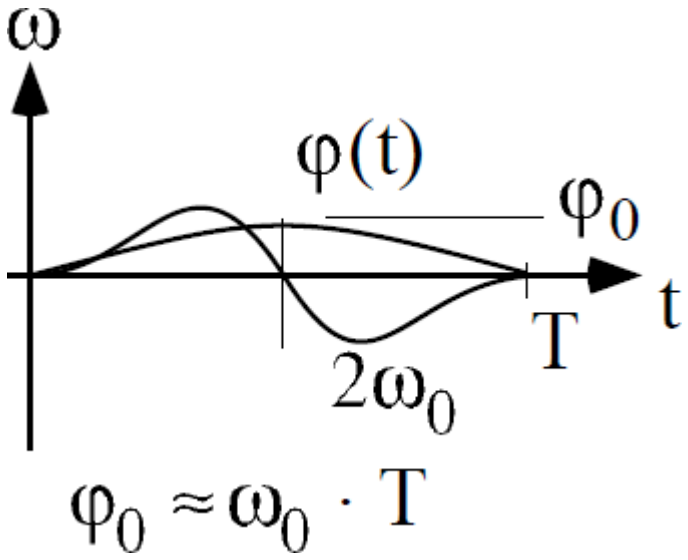
$$r = \frac{2}{\pi} \cdot v \cdot \frac{90\,°}{\omega_0} m$$

- The circular arch is approximated with a circle (forth of diameter is therefore $s$)

$$r = \frac{2s}{\pi}$$

# Basic Ability "S-Trajectory Drive"



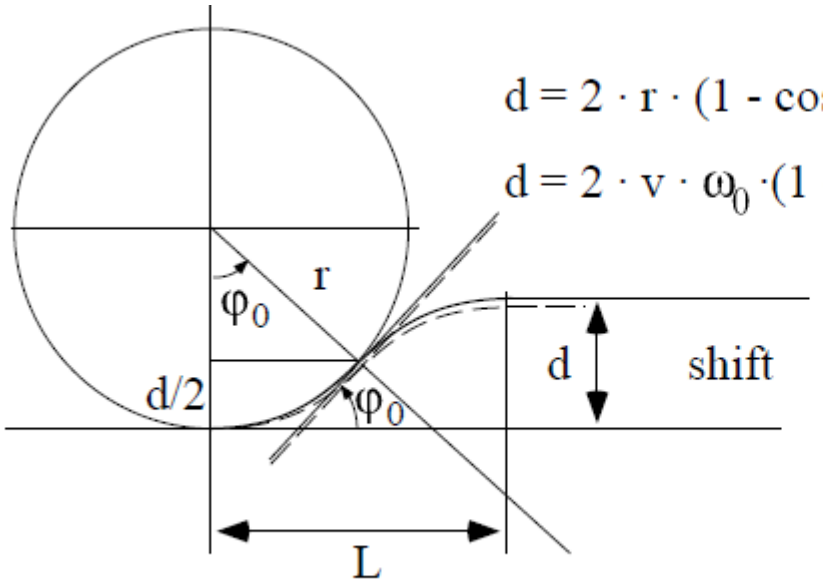




$$\varphi_0 \approx \omega_0 \cdot T$$

S-trajectory drive: $\varphi_0 \approx \omega_0 \cdot T$

# Basic Ability "S-Trajectory Drive"

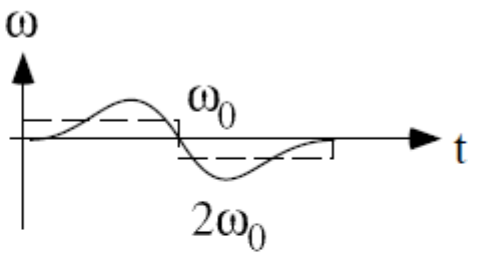
- The "S" shaped trajectory is characterized by offset $d$ and angle $\varphi_0$
- Approximation is pessimistic: $d > d_{true}$, $L > L_{true}$




$$d = 2 \cdot r \cdot (1 - \cos \varphi_0)$$

$$d = 2 \cdot v \cdot \omega_0 \cdot (1 - \cos \varphi_0)$$

$$L = 2r \sin \varphi_0$$



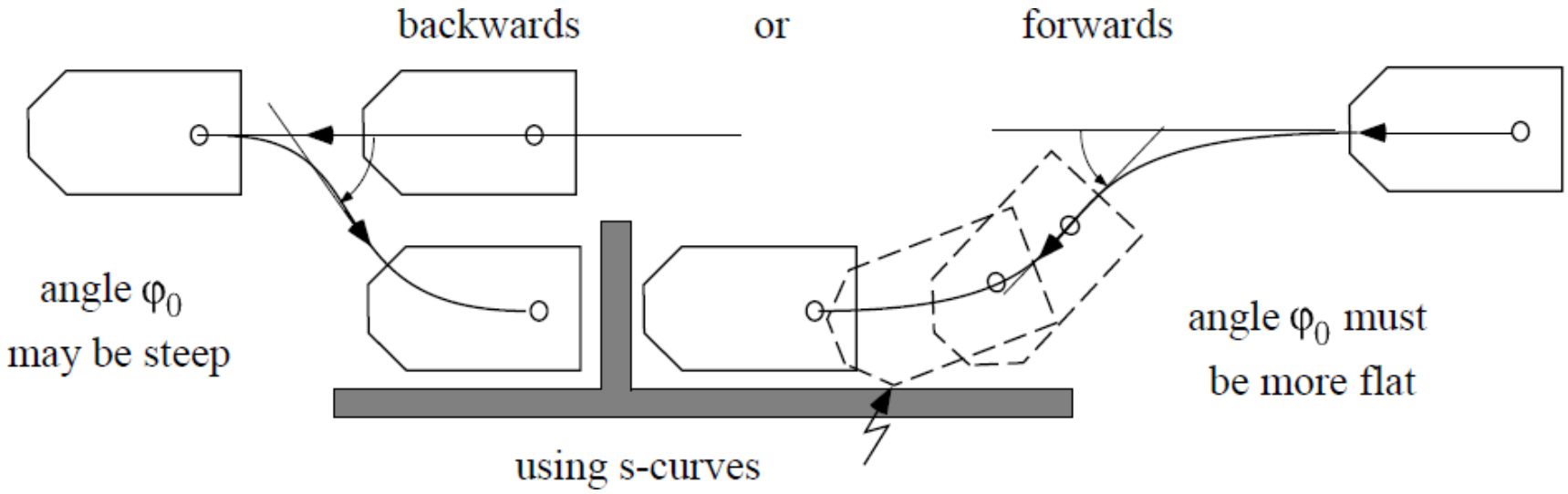

pessimistic approximation

$$d > d_{real}$$

$$L > L_{real}$$

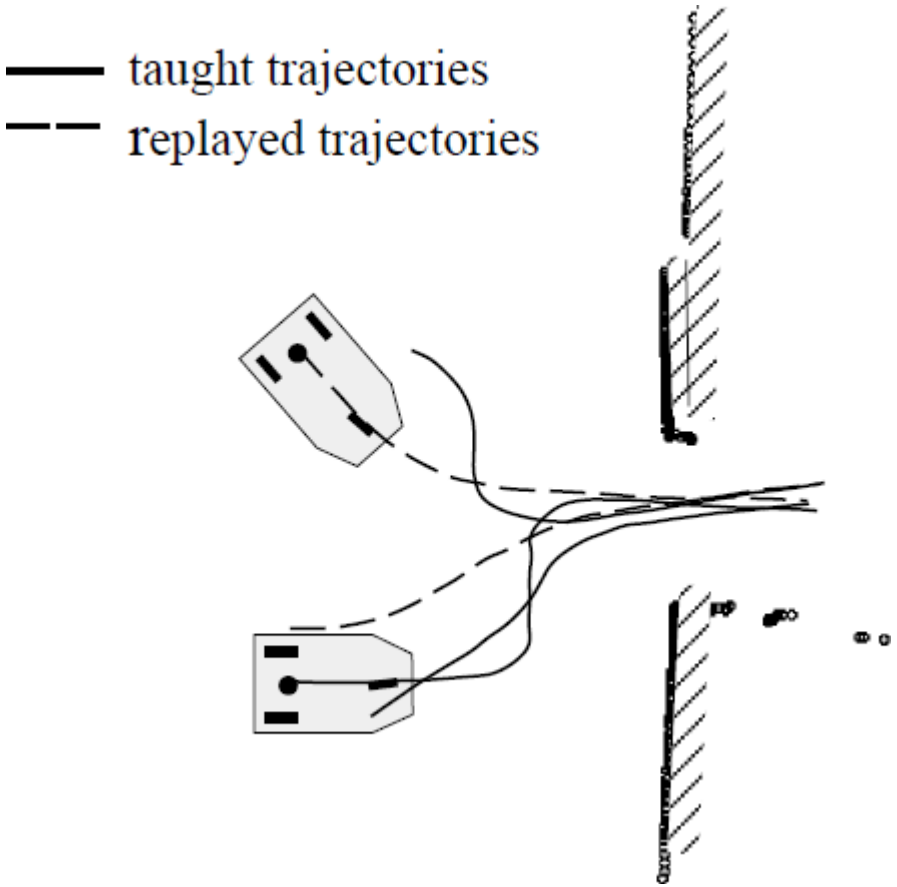
S-trajectory drive

# Further Basic Abilities



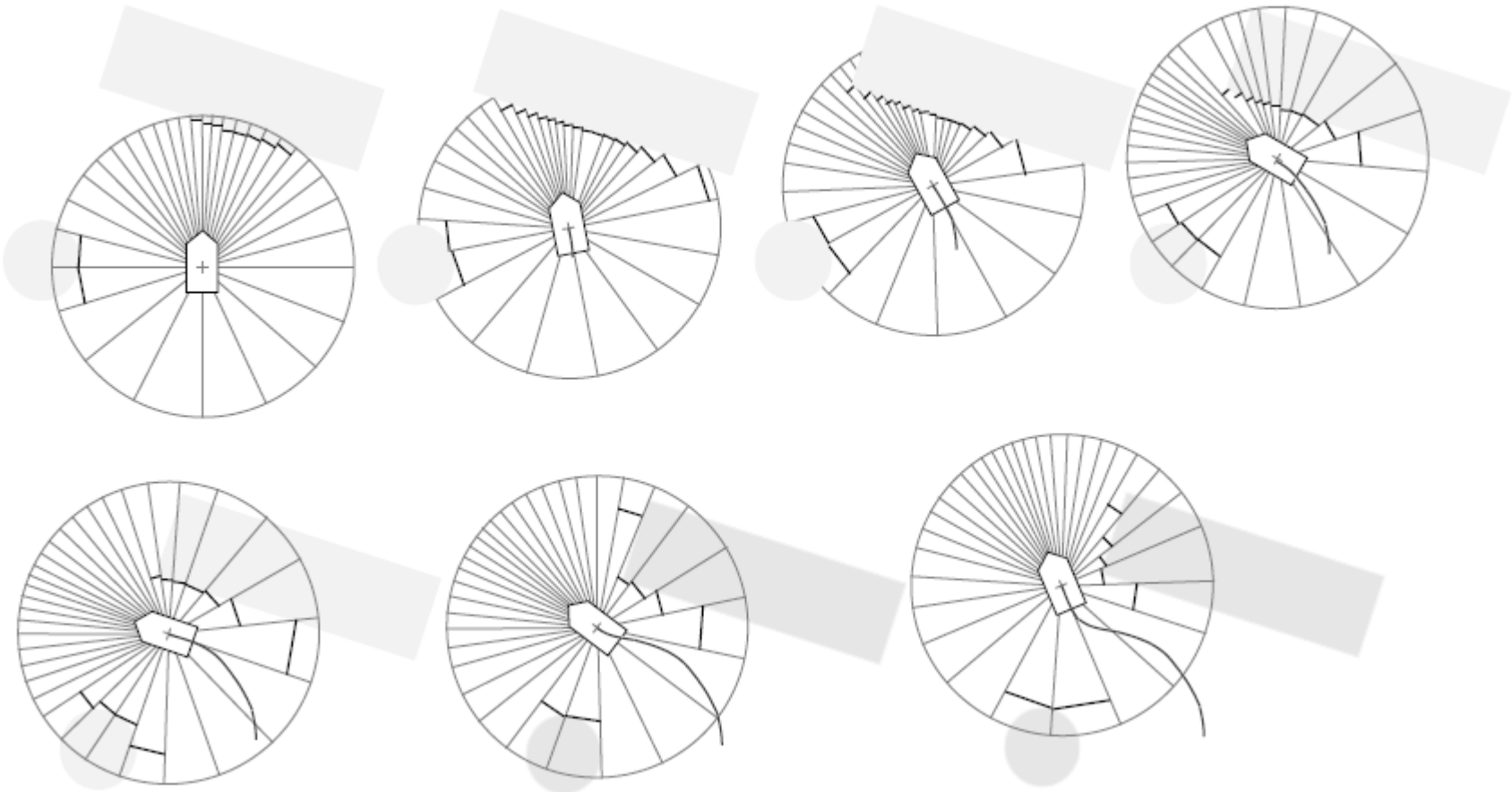

Docking maneuvers

# Further Basic Abilities



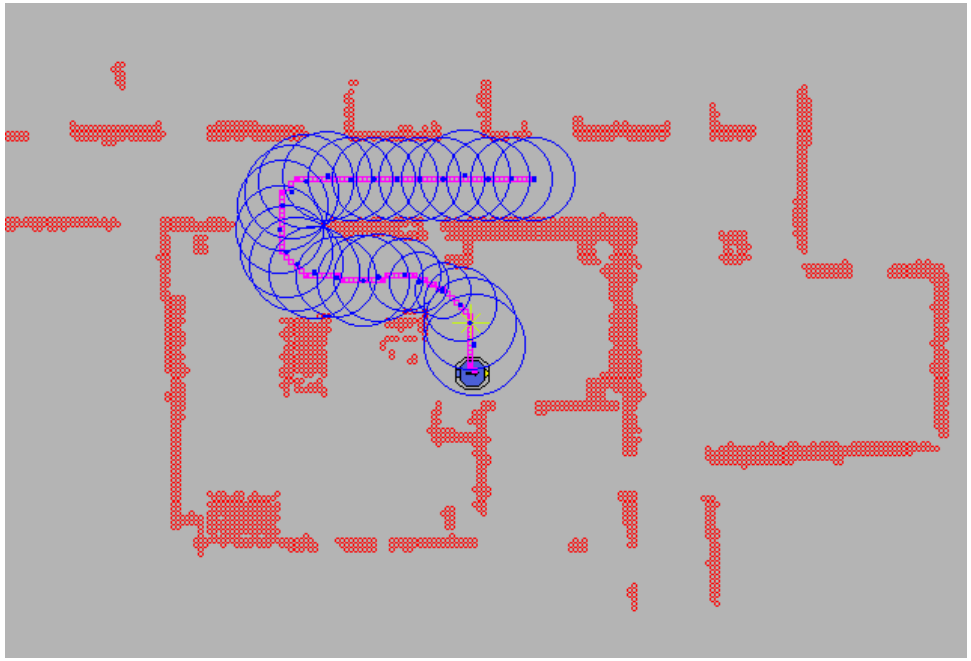

Door passing

# Wall Following



# Elastic Band Method: Introduction

- Introduced by [Quinlan93a], used to adapt a planned path online according to the robot motion and dynamic obstacles
- The band consists of $n$ bubbles $b_i$, $i = 1, \ldots, \mathrm{n}$ representing the free space along the path where the robot can move



# Elastic Band Method: Bubble Definition

- Main idea and algorithmic procedure according to [Philippsen03]
- Each bubble is defined by its center $\vec{c}_{bi}$, radius $r_{bi}$ and obstacle masking distance $D_{mi}$
- The bubbles are spread along the path to guarantee an overlapping according to the robot dimensions
- Thus, the area covered by the bubbles mark the collision-free motion space of the robot

$$L_i = \sum_{j=1}^{i} \left\| \vec{c}_{bj-1} - \vec{c}_{bj} \right\| \quad (1)$$

# Elastic Band Method: Masking Distance

- The masking distance $D_{mi}$ is related to the position $L_i$ of the bubble along the path according to $(1)$

- It specifies the range around each bubble center in which obstacles are ignored

$$D_{mi} = D_{m,max} \cdot \begin{cases} 0 & \text{if } L_i \leq L_{min} \\ 1 & \text{if } L_i \leq L_{max} \\ \dfrac{L_i - L_{min}}{L_{max} - L_{min}} & \text{otherwise} \end{cases} \qquad (2)$$

# Elastic Band Method: Stretching

- The parameters $L_{min}$ and $L_{max}$ represent the stretching limits of $D_{mi}$ (accumulated path lengths)
- $D_{m,max}$ is the maximum range at which obstacles are neglected
- Obstacles are represented by the center points $\vec{p}_j = (x_{pj}, y_{pj})$ of the grid cells
- Each bubble is related to a set of masked obstacles $\{\vec{p}_{m,ij}\}$ according to

$$\{\vec{p}_{m,ij}\} = \{\vec{p}_j : \|\vec{c}_{bi} - \vec{p}\| > D_{mi}\} \tag{3}$$

# Elastic Band Method: Bubble Radius

- The obstacle $\vec{p}_i^*$ closest to $b_i$ is the critical one

$$\vec{p}_i^* = \arg\left(\min_{\vec{p}\in\{\vec{p}_{m,ij}\}} \|\vec{c}_{bi} - \vec{p}\|\right) \tag{4}$$

- It defines the bubble radius $r_{bi}$ (space guaranteed to be free) according to

$$r_{bi} = \min_{\vec{p}\in\{\vec{p}_{m,ij}\}} \|\vec{c}_{bi} - \vec{p}\| \tag{5}$$

# Elastic Band Method: Procedure

- Based on these relationships, the bubbles are completely defined
- The first step of the elastic band method is to translate a given path into a minimal set of bubbles covering the free space, corresponding to the robot dimensions
- The next step is to adapt these bubbles in size and position while the robot is moving along the path
- For that purpose, the first bubble follows the robot and the last one is stuck fixed at the goal
- All other bubbles move iteratively due to the influence of two forces $\vec{f}_{int}$ and $\vec{f}_{ext}$ on their center $\vec{c}_{bi}$

# Elastic Band Method: Shift in Time

This shift from time $t$ to $t+1$ is given in

$$\vec{c}_{bi,t+1} = \vec{c}_{bi,t} + \Delta\vec{c}_{bi} \tag{6}$$

with

$$\Delta\vec{c}_{bi} = \alpha_{tot,i} \cdot \left(\vec{f}_{int,i,i-1} + \vec{f}_{int,i,i+1} + \vec{f}_{ext,i}\right)$$

and

$$\alpha_{tot,i} = \begin{cases} 1 & \text{if } r_{bi} \leq r_{lim} \\ \dfrac{r_{bi}}{r_{lim}} & \text{otherwise} \end{cases}$$

# Elastic Band Method: Forces

- The internal force $\vec{f}_{int,ij}$ determines the strength of cohesion between adjacent bubbles $i$, $j$

$$\vec{f}_{int,ij} = \alpha_{int} \cdot \begin{cases} 0 & \text{if } \left\|\vec{c}_{bi} - \vec{c}_{bj}\right\| \leq \varepsilon \\ \dfrac{\vec{c}_{bj} - \vec{c}_{bi}}{\left\|\vec{c}_{bi} - \vec{c}_{bj}\right\|} & \text{otherwise} \end{cases} \tag{7}$$

- The external force $\vec{f}_{ext,i}$ represents a repulsion of bubble $i$ from its critical obstacle $\vec{p}_i^*$

$$\vec{f}_{ext,i} = \alpha_{ext} \cdot \begin{cases} 0 & \text{if } r_{bi} \leq \varepsilon \text{ or } r_{bi} \geq r_{lim} \\ \dfrac{r_{lim} - r_{bi}}{r_{bi}} (\vec{c}_{bi} - \vec{p}_i^*) & \text{otherwise} \end{cases} \tag{8}$$

# Elastic Band Method: Parameter Description

- $r_{lim}$ specifies the distance limit at which the elastic band is influenced by obstacles
- $\alpha_{int}$ and $\alpha_{ext}$ are force weighting factors
- $\alpha_{tot,i}$ creates a proportional relation between bubble size and agility
- $\varepsilon$ avoids division by zero

# Elastic Band Method: Parameter Values

The following table lists all parameters introduced so far and their values used during tests with mobile robot MARVIN.

| Parameter | Value |
|---|---|
| $L_{min}$ | 2000 mm |
| $L_{max}$ | 8000 mm |
| $D_{m,max}$ | 0 mm |
| $r_{lim}$ | 1600 mm |
| $\alpha_{int}$ | 10 |
| $\alpha_{ext}$ | 10 |
| $\varepsilon$ | 50 mm |
| To compare: robot radius | 400 mm |

# Elastic Band Method: Procedure

- Based on (6) to (8), the bubble centers are iteratively moved at each cyclic calculation step
- Their respective radii are calculated according to (5), taking possible dynamic obstacles into account
- Whenever the robot-fixed bubble is approximately coincident with its neighbor, it is deleted and replaced by this one
- Consequently the band represents a dynamically smoothed version of the path around obstacles from the robot to the goal position
- Its changing thickness marks narrow passages and free areas where the robot can move faster
- The whole procedure stops when the robot has reached the goal of the path (last bubble)

# Elastic Band Method: Algorithm

- Given
  - Grid map with obstacles as center of occupied cells $\vec{p}_j = (x_{pj}, y_{pj})$
  - Path represented by center of visited grid cells
- Preparation
  - Calculate minimal set of $n$ bubbles $b_i$ with center $\vec{c}_{bi}$, radius $r_{bi}$
  - $\vec{c}_{b1}$ = robot pose
  - $\vec{c}_{bn}$ = goal pose
  - $\|\vec{c}_{bi+1} - \vec{c}_{bi}\|$ according to robot dimension
  - $r_{bi}$ according to eq. 5

# Elastic Band Method: Algorithm

{—**online update** (robot moving along path)—}
**while** robot is not at goal **do**
    Update grid map (dynamic obstacles)
    Remove $b_1$ if $\|b_2 - b_1\|$ too small
    **for** every remaining bubble **do**
        Determine critical obstacle $\vec{p}_i^*$
        Calculate $r_{bi}$ according to eq. 5
        Calculate external and internal forces
        Apply shift to bubble center (cf. eq. 6)
    **end for**
**end while**

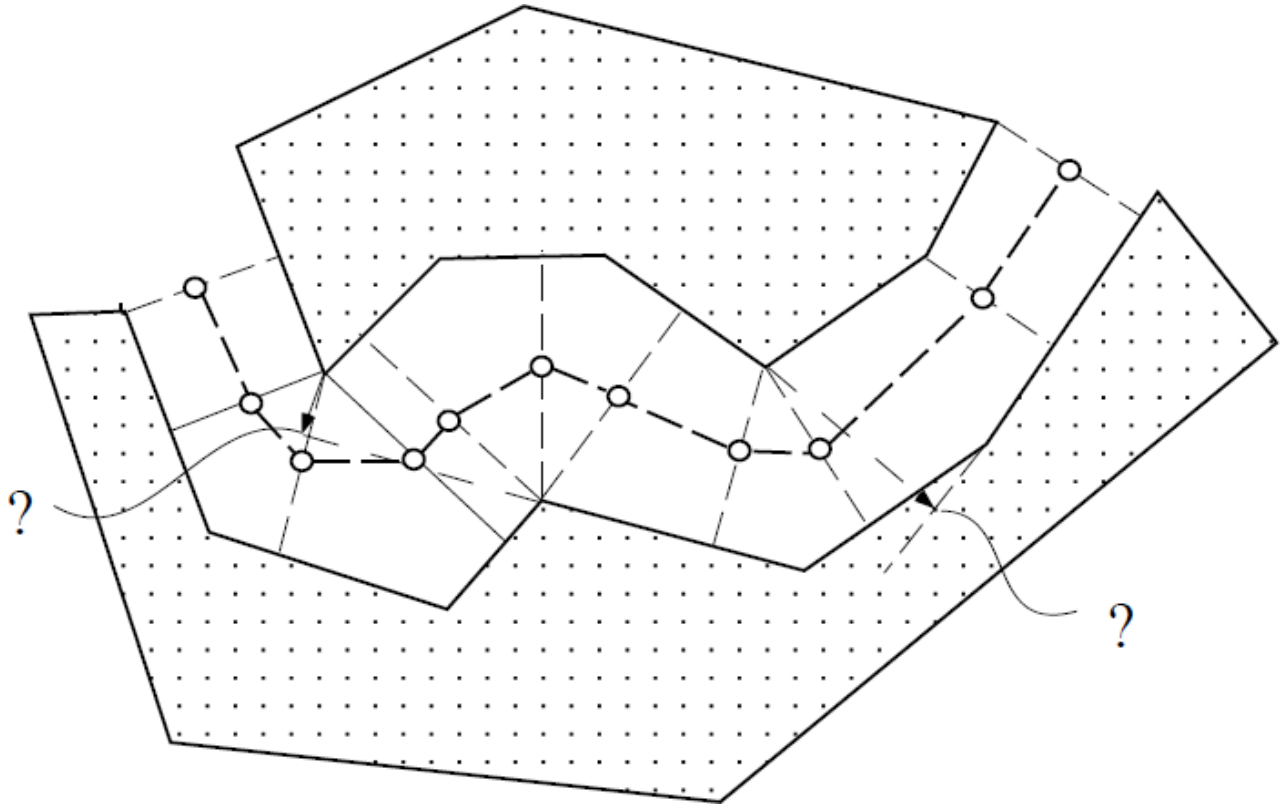
# Elastic Band Method: Discussion

- Remaining problem: what to do if some bubbles shrink below robot dimensions
- In this case there is not enough free space for the robot to follow the path
- When this situation is stable for a certain amount of time, a replanning has to be triggered to take new obstacles into account
- Of course the planning should not be started immediately when it gets invalid as obstacles might vanish before the robot is getting close to them (e. g. a person crossing the path)
- The new plan is then given as input for a new run of the elastic band algorithm

# Narrow Free Space

- Move between nearby obstacles
- Stay in middle of free space
  - Look for convex obstacle points
  - Build the normal through this point towards the free space
  - Take into account only normals with foot points at the other shore
  - The mid points of these normals are the wanted course points

# Geometrical Maps with Narrow Free Space

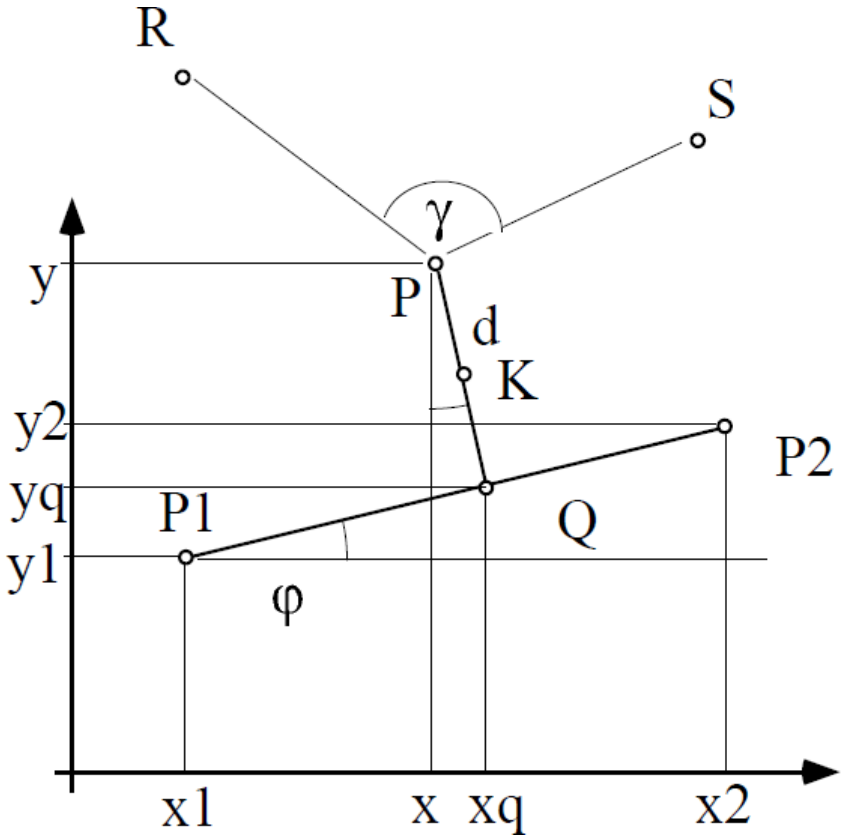



Narrow river course points

# Geometrical Maps with Narrow Free Space

- Determination of a path point
- Assumptions
  - Let $(R, P)$ and $(P, S)$ be polygon lines of one shore, $(P_1, P_2)$ be a polygon line at the other shore
  - Let the angle $(S, P, R) = \gamma < 180\,°$ then $P$ is a convex point
  - Let $Q$ be the footpoint of the normal on $(P_1, P_2)$

# Geometrical Maps with Narrow Free Space




Finding a course point at a passage

# Balloon Algorithm

- Generate mid points between smooth shores
- Map given as continuous curves $f_1(x)$ and $f_2(x)$
- Find midpoints: pressing a balloon through the passage: mid point defines the curve with maximal distance to the shores

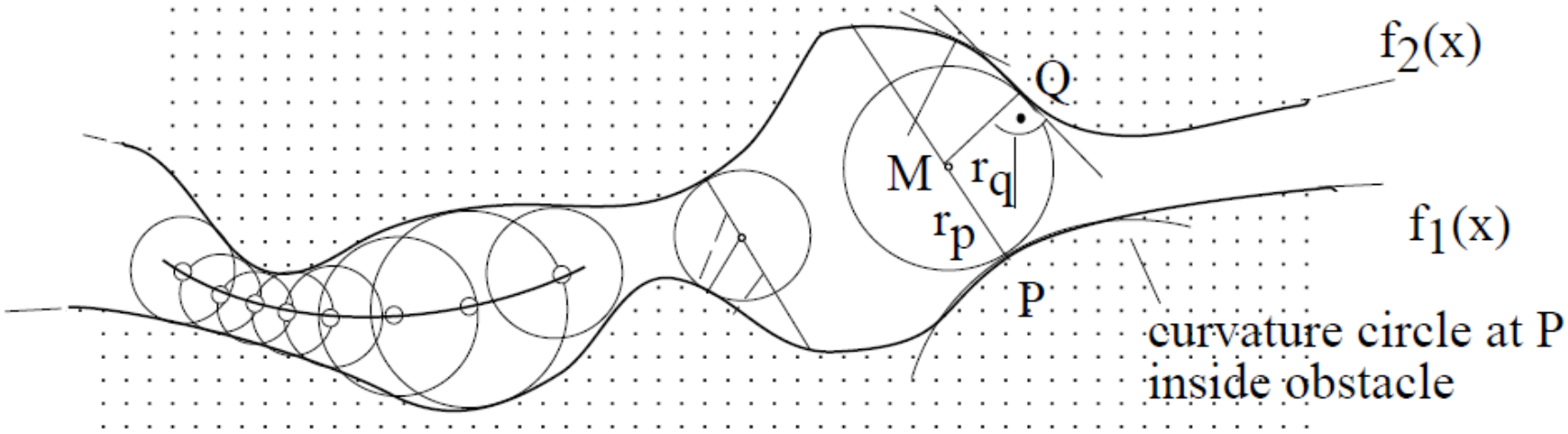
# Balloon Algorithm

- Let $P$ be a convex point on $f_1(x)$ defined by

$$\frac{\left(d^2 f_1(x)\right)}{dx^2} < 0$$

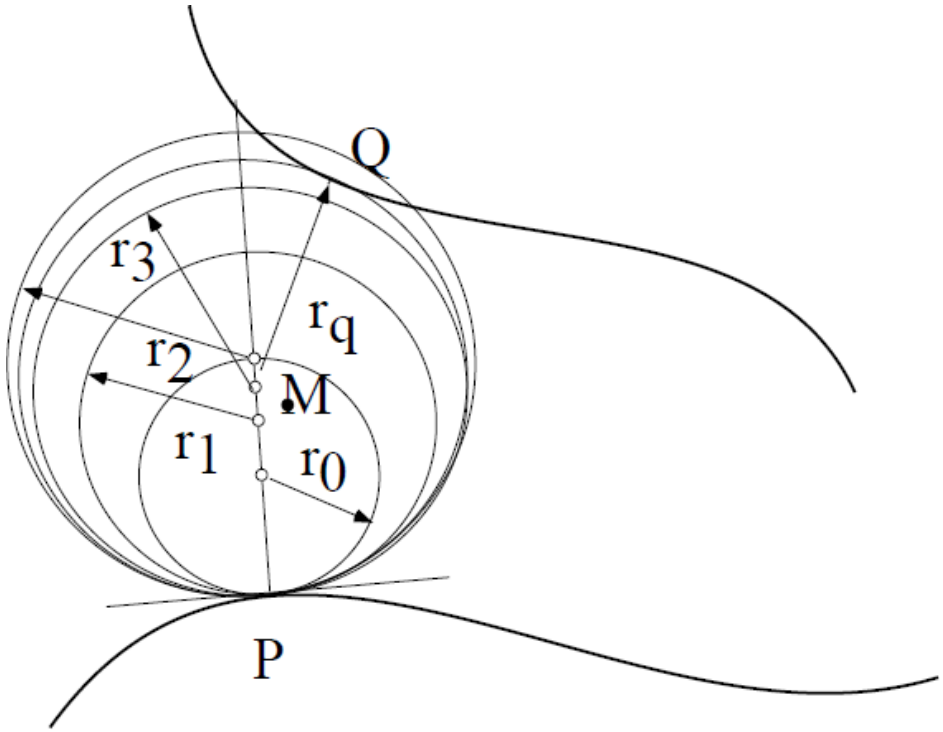
- Form the normal at $P$: $y = mx + b$ with $m = -1/f_1'(x_p)$ and $b = y_p - mx_p$
- Form the normal from a point $Q$ from $f_2(x)$ cutting the normal on $P$ at $M$
- Vary $Q$ until $r_p = |M, P| = r_q = |M, Q|$
- Then $M$ is the mid point of the touching circle and one of the wanted points

# Balloon Algorithm




Course points as mid points of balloons

# Balloon Algorithm




Finding radius of touching circle

# Balloon Algorithm

Finding the radius of the balloon:

$r \coloneqq r_0$
$\Delta r \coloneqq r_0$
$(*)$ Form a circle with $r$ cutting $P$ with its center on line orthogonal to tangent in $P$
**if** the circle cuts $f_2(x)$ **then**
    **if** the distance of the cutting points is $< \varepsilon$ **then**
        $r_q \coloneqq r$
        $M$ is a course point
    **else**
        $\Delta r \coloneqq \Delta r/2$
        $r \coloneqq r - \Delta r$
        Go to $(*)$
    **end if**
**else**
    $r \coloneqq r + \Delta r$
    Go to $(*)$
**end if**

# Area Filling on Grid Maps: Assumptions

- Given: a grid map
- Task: The vehicle has to clean or paint (with a brush) all free cells.
- Within a cell the vehicle can turn or change its direction.

# Area Filling on Grid Maps

(*)

Try to drive straight paths

Check parallel cells points

Mark beginning and end points of paths on a stack

Mark points already driven through

**if** the path ends **then**

{there is either an obstacle or a marked cell hit}

Pop the stack

**else if** the stack is not empty **then**

The mark of the beginning of a path not jet driven is taken from the stack

Drive to that point

Go to (*)

**else**

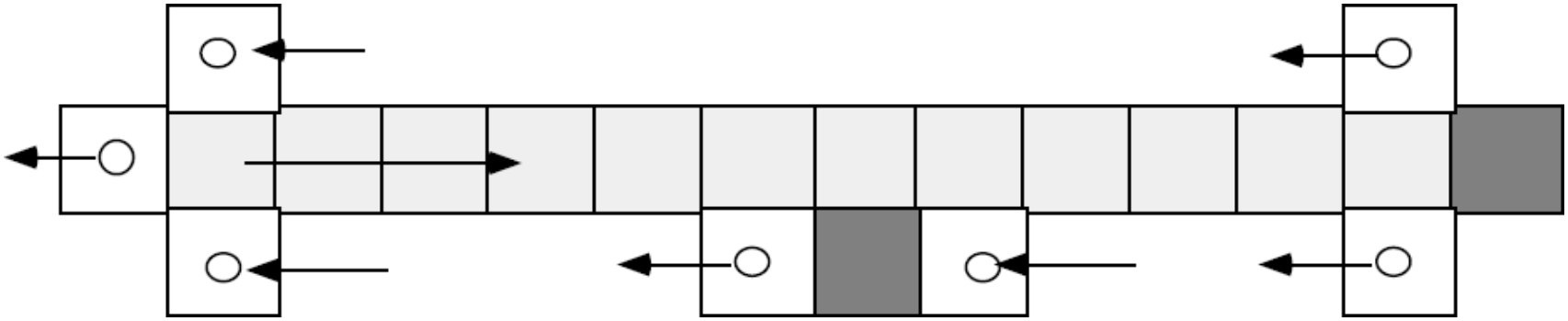
The area is filled

**end if**

# Area Filling on Grid Maps

- Open circles: start points to be laid down on the stack with the direction where to drive further on

- If there is no free space left at the moment, drive through an already cleaned region to a next starting point.

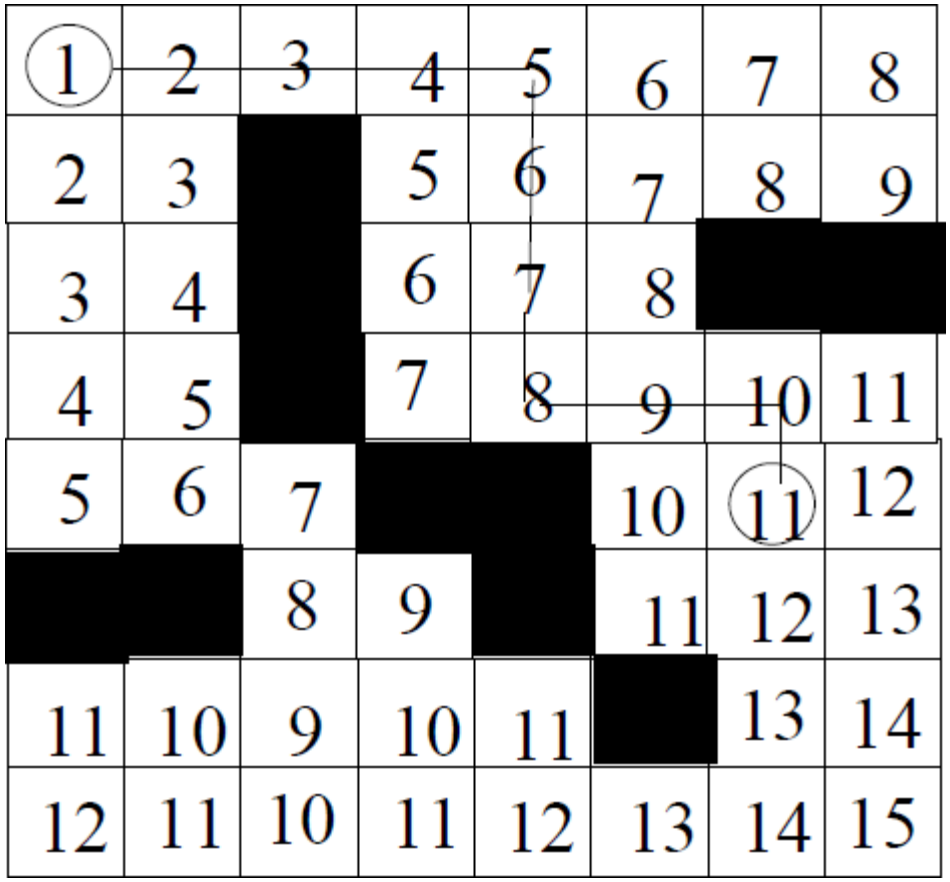


Typical situation in area filling

# Navigation on a Grid Map

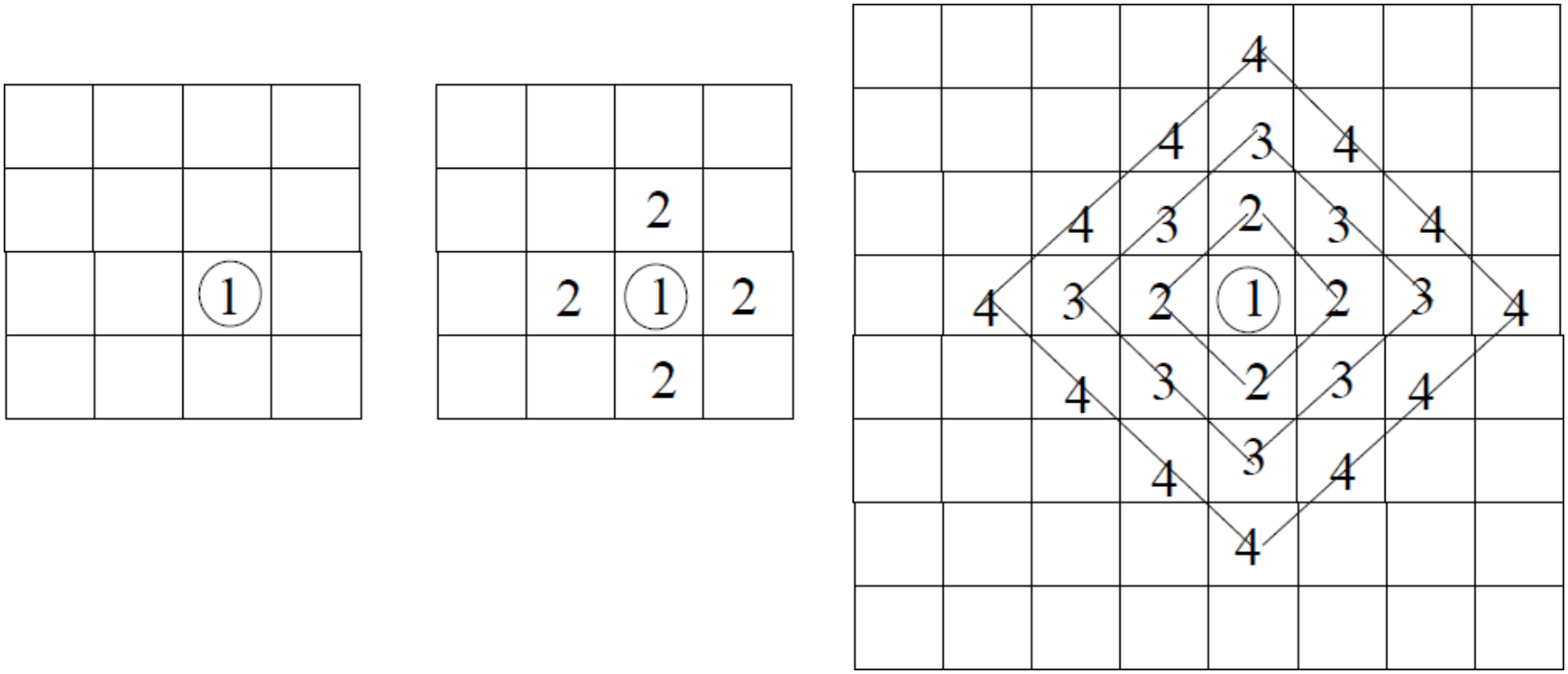
- Task: Find the shortest path from start to goal in a grid map
- Flood the grid beginning with the start point
- A wave runs around obstacles and labels the complete free space
- From the goal descend to the start forming a list of passed cells

# Navigation on a Grid Map




Flooding a grid map

# Navigation on a Grid Map



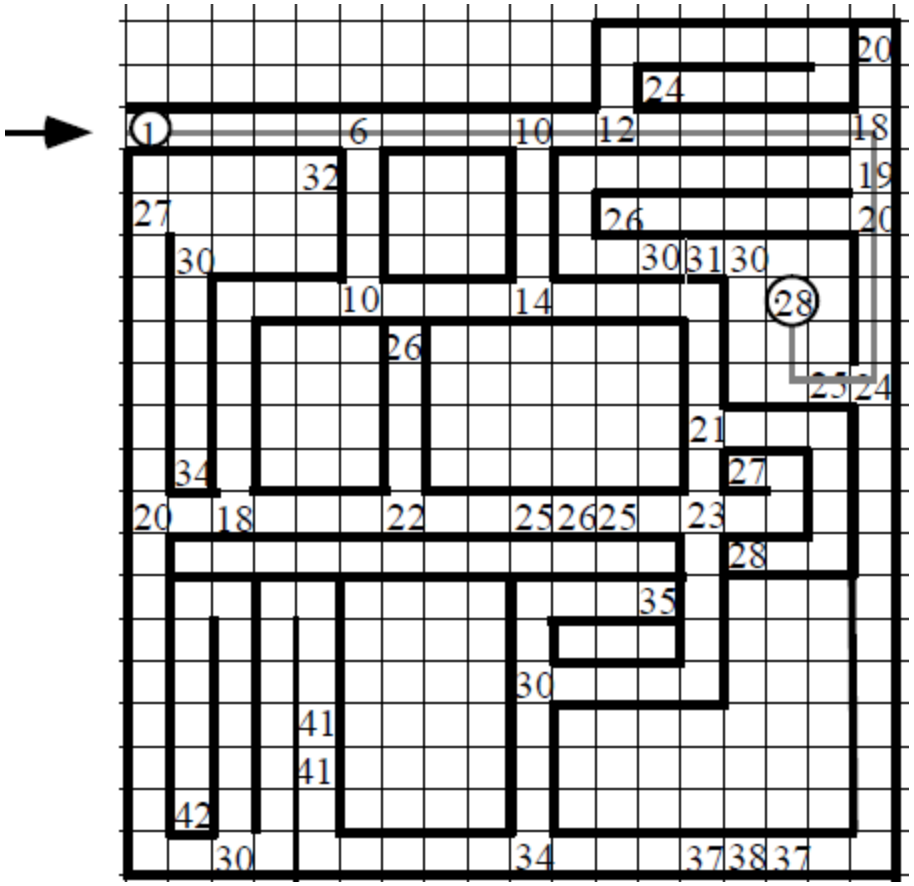

Flooding an empty grid

# Navigation on a Grid Map

Flooding a grid with thick walls:

Initialization: $i \coloneqq 1$; $j \coloneqq 0$
Starting cell $z \coloneqq 1$
Push $S_i$
**repeat**
    $i \coloneqq (i + 1) \bmod 2$
    $j \coloneqq (j + 1) \bmod 2$
    **repeat**
        Pop stack $S_i$
        Check surrounding of the cell taken out of stack:
        **for all** cells with $z = 0$ do
            **if** $min_{z>0}(O, S, W, N)$ **then**
                $z \coloneqq min_{z>0}(O, S, W, N) + 1$
                Push to stack $S_j$
            **end if**
        **end for**
    **until** stack $S_i$ is empty
**until** stack $S_j$ is empty

# Navigation on a Grid Map




Solving a maze by flooding a grid map

# Navigation on a Grid Map

Solving a maze in a grid map:

Initialization: start point $z \coloneqq 1$
**for** all cells **do**
  **if** $z^n = 0$ **then**
    **if** all neighbors $O, S, W, N = 0$ **then**
      $z^{n+1} \coloneqq 0$
    **else if** $1 \dots 3$ neighbors are obstacle points $H$ denoted by $(-1)$ **then**
      $z^{n+1} \coloneqq 0$
    **else if** $min_{z>0}(O, W, S, N) > 0$ **then**
      $z^{n+1} \coloneqq min_{z>0}(O, W, S, N) + 1$
    **end if**
  **else**
    $z^{n+1} \coloneqq z^n$ {color is preserved}
  **end if**
**end for**

# Quadtree: One Step Path Planning

- Given a grid map as quadtree with white and black nodes
- Then path planning with an A*-algorithm can be undertaken with …
  - Expansion of a node: look for neighboring nodes with free room
  - Expansion of a node in the next step
  - Expansion of vertical or horizontal obstaclefree neighbor nodes
  - Expand the node with the minimal value of the weight
  - function

# Quadtree: One Step Path Planning

- The weight function is

$$f(C) = g(P) + d(P, C) + \alpha \cdot \left(O_{max} - O(C)\right) + h(C)$$

- $C$: Obstacle free node
- $P$: Predecessor of $C$ on this path
- $g(P)$: Costs of the path from start point $S$ to $P$
- $d(P, C)$: Actual distance from $P$ to $C$
- $O(C)$: Distance from $C$ to the next obstacle
- $O_{max}$: Maximal distance from any $C$ to the next obstacle
- $\alpha$: Constant, describing the minimally allowed distance value to obstacles
- $h(C)$: Optimistically guessed distance between $C$ and the goal $Z$

# Quadtree: One Step Path Planning

- The result is a list of obstacle free nodes of different sizes
- The merits of this one step planning are …
  - Efficient calculation of the path
  - $\alpha$ gives the minimum distance to obstacles
  - No large overhead in the quadtree representation

# Quadtree: Two Step Path Planning

- Given a grid map as quadtree with white, gray and black nodes
- The path is planned on as high a level as possible
- In a second step the gray nodes are expanded
- Gray nodes are expanded only to the extent that is absolutely necessary
- Smallest squares have vehicle dimension

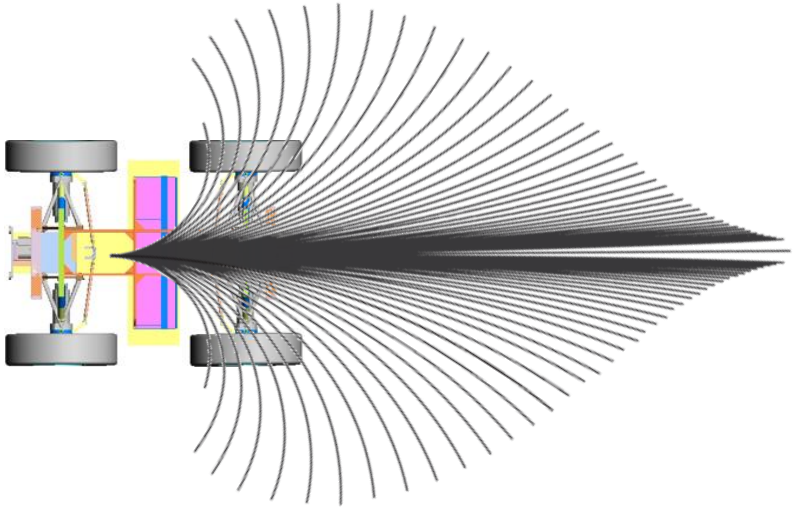
# Merits of Two Step Planning

- The algorithm is cost effective, if many small obstacles allow only a few large white nodes
- Unknown regions may be treated efficiently as gray nodes with large costs

# Tentacle Based Navigation

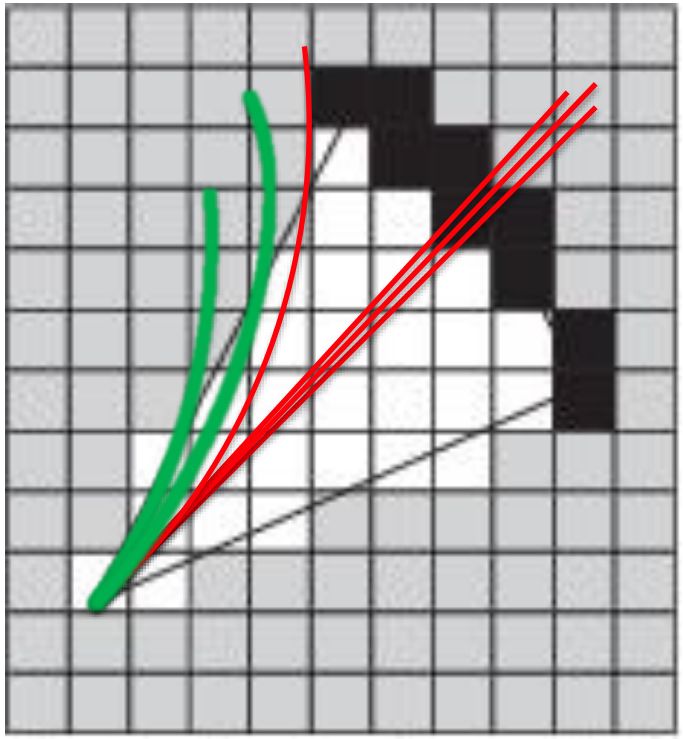
- Simple bio-inspired navigation approach
- Considers robot kinematic/dynamic

- Problem: Which arcs/trajectories can be safely driven?
- Solution: Check set of predefined arcs (tentacles)
- Approach applied to VW-Passat at C-Elrob 2007 and DARPA Urban Challenge 2007



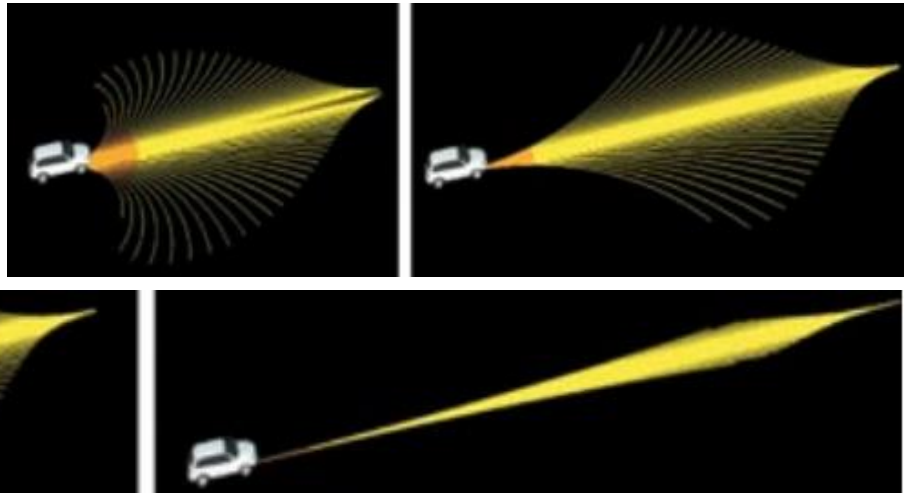
# Tentacle Based Navigation

- Given:
  - Ackerman based mobile robot
  - 2D Laserscanner
- Problem:
  - Avoid near obstacles
- Approach:
  - Calculate robot-centered grid map for each scan
  - Select tentacle set according to velocity
  - Evaluation of tentacle set
  - Selection fo suitable trajectory



# Tentacles

- Represented in local coordinate system of vehicle
- Start at kinematic center



- Represent trajectory corresponding to a specific steering angle
- Length increases with higher speed set
- Within a set less cured tentacles are longer

- $N = 16$ speed sets
- $K = 81$ tentacles per set

# Geometry

- Clothoid path approximation by concatenation of small arcs
- Tentacle radius:

$$r_k = \begin{cases} p^k R_n & | \quad k = 0, \dots, 39 \\ \infty & | \quad k = 40 \\ -p^{k-41} R_n & | \quad k = 41, \dots, 80 \end{cases}$$

  - $p = 1.15$ exponential factor
  - $R_n = \frac{l}{\Delta\Phi(1-q^{0.9})}$ for speed set $n$ with $q = \frac{n}{(N-1)}$
  - $\Delta\Phi = 1.2\left(\frac{\pi}{2}\right)$ angle subtended by outmost tentacle of lowest speed set
  - ➢ At low speeds vehicle can look around, e.g., a road branch more than 90 degree
  - Exponential form of radius equation to have more tentacles with small curvatures

# Geometry

- Length of outmost tentacles:

$$l = 8m + 33.5m\, q^{1.2}$$

  - Exponential factor $q^{1.2}$ to sample low speeds more frequently
- Length of $k$th tentacle:

$$l_k = \begin{cases} l + 20m\sqrt{\dfrac{k}{40}} & | \quad k = 0, \dots, 40 \\ l + 20m\sqrt{\dfrac{k-40}{40}} & | \quad k = 41, \dots, 80 \end{cases}$$

  - Straight tentacles need minimum length for sufficient lookahead
  - Outer tentacles shorter to prevent bending behind vehicle

# Geometry

- Velocity for speed set $j$:

$$v_j = v_s + q^{1.2}(v_e - v_s)$$

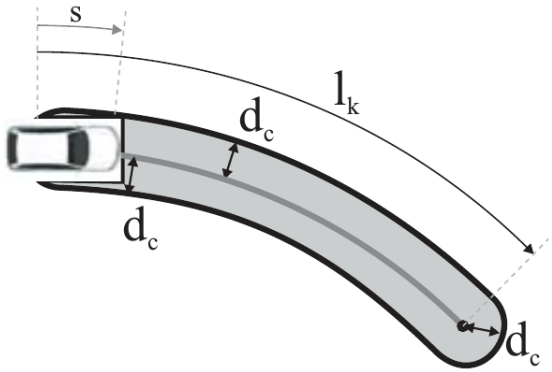
  - $v_e = 0.25\,m/s$ lowest speed set
  - $v_s = 10\,m/s$ maximum speed set

- Outermost radii could also be designed according to
  - Underlying physical properties like centripetal force
  - Environment (highway, forest, construction side, …)

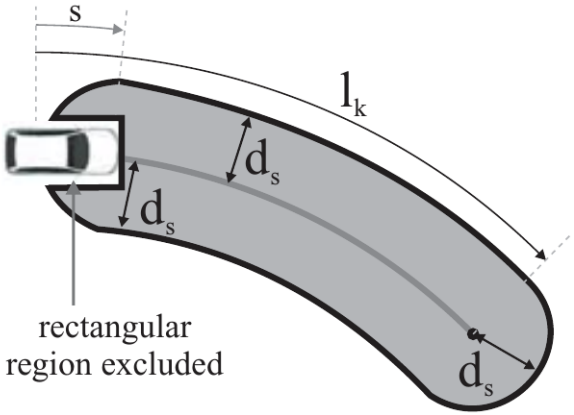
- Here: tailored to sensor range and general purpose usage

# Evaluation – Classification & Support Area

- Classification area: grid cells within radius $d_c$ to any point on the tentacle
  - Drivability estimation
- Support area: grid cells within a radius $d_s > d_c$
  - Ranking among drivable tentacles
- $d_c$ and $d_s$ speed dependent




Classification Area




Support Area

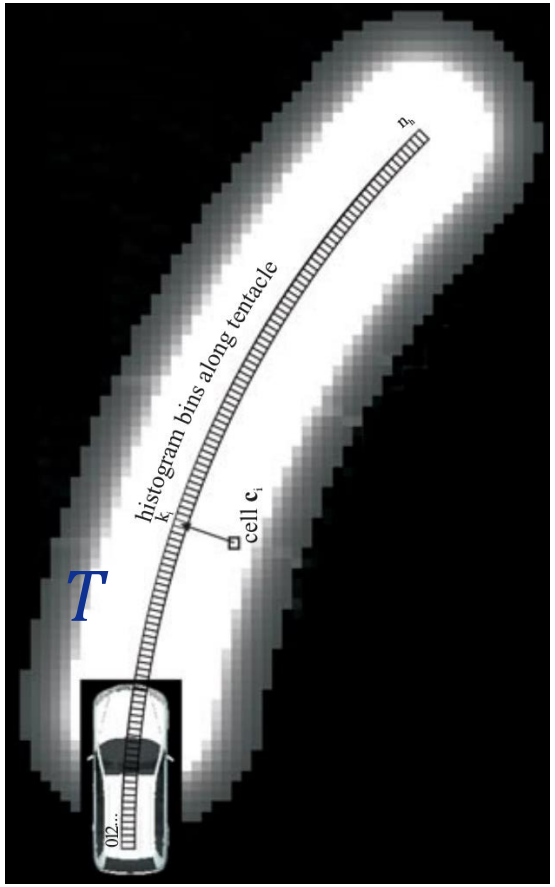
# Evaluation

- Vehicle centered map + vehicle centered fixed tentacles
- ➢ Precomputation of indices ($n$ covered cells) and histogram projection in tentacle data structure

$\mathrm{T} = \{c_0, \ldots, c_{n-1}\}$ with

$$c_i = (o_i, w_i, k_i, f_i)$$

  - $o_i$ cell offset
  - $w_i$ weight (depending on distance of cell to tentacle)
  - $k_i$ histogram index
  - $f_i$ classification area flag ($f_i = 1 \rightarrow$ cell is classification cell)



# Tentacle Classification

- All cells $c_i$ in the classification area are evaluated
- If cell is blocked, corresponding histogram bin marked as blocked
- No weighting in this step due to assured damage in blocked case
- Definition of *crash distance* $l_c = l_s + \frac{v^2}{2a}$ with
  - $l_s$ security distance (e.g. 6m)
  - $v$ current vehicle velocity
  - $a$ constant convenient deceleration aprox. (e.g. $1.5\ m/s^2$)
- If no bin of histogram is blocked up to $l_c$ the tentacle is drivable

# Tentacle Selection

- Selection of "best" tentacle based on 2 metrics $\in [0, \dots, 1]$

- Clearance Value

$$v_{clear}(l_0) = \begin{cases} 0 & | \quad tentacle\ entirely\ free \\ 2 - \dfrac{2}{1 + e^{-c_{clear} - l_0}} & | \quad otherwise \end{cases}$$

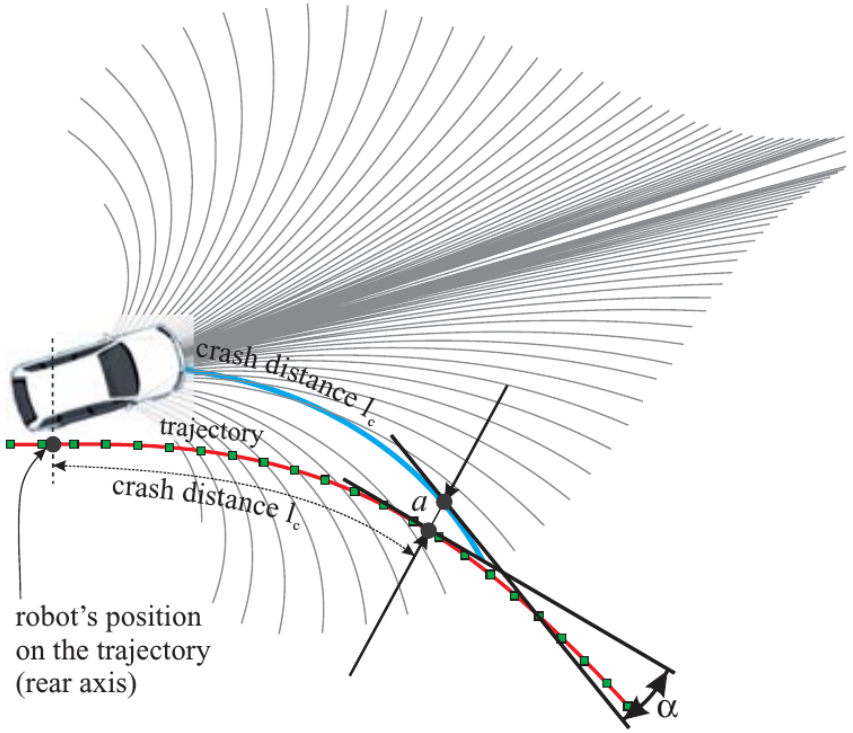
  - $l_0$ distance to first obstacle
  - $c_{clear} = \frac{\ln(1/3)}{-l_{0.5}}$ const to yield $v_{clear}(l_{0.5}) = 0.5$ with $l_{0.5} = 20m$

# Tentacle Selection

- Trajectory Value

$$v_{traj} = \frac{v_{dist} - v_{min}}{v_{max} - v_{min}}$$

  - $v_{dist} = a + c_\alpha \alpha$
  - $a$ distance between point on tentacle at crash distance and corresponding point on trajectory
  - $\alpha$ relative tangent orientation
  - $c_\alpha = 3.0 m/rad$ constant
  - $v_{min}, v_{max}$ min/max values of $v_{dist}$ over all tentacles of current speed set




Pushes vehicle toward following a given trajectory

# Tentacle Selection

- Combination of metrics:

$$v_{combined} = a_0 v_{clear} + a_1 v_{traj}$$

- E.g. $a_0 = 1$, $a_1 = 0.5$ at 2007 DARPA Urban Challenge
- Selection of tentacle with highest rating
- No tentacle drivable → select tentacle with largest clearence value and perform stop

- Execution of tentacle by application of simple recursive filter to overcome discretization

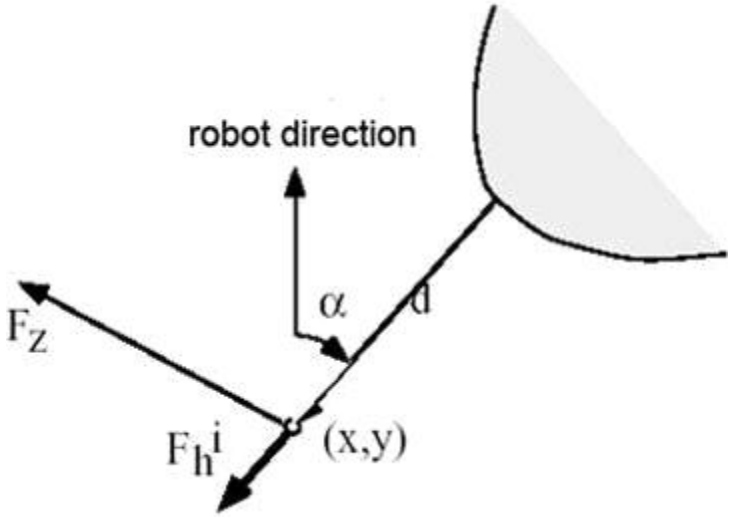
# Potential Field Method

- Given
  - Robot with obstacle map
  - Obstacles modeled as probabilities
  - Target (direction or position)
- Potential Field
  - Target force $F_z(x, y)$ for every point $(x, y)$
  - Obstacle force $F_h^i(x, y)$ for each obstacle

# Potential Field Method

$$F_h \approx \frac{1}{d}\cos\alpha \text{ for } |\alpha| < 90\,°$$

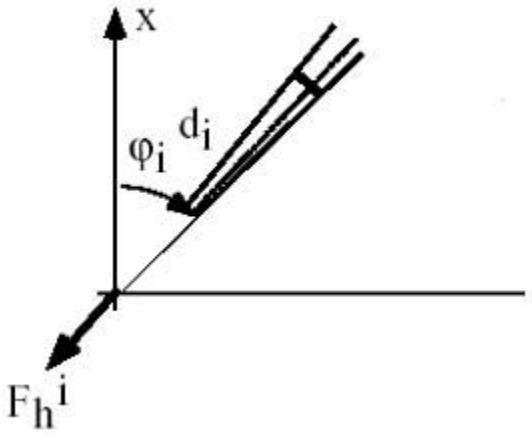
$$F_h = 0 \text{ for } |\alpha| \geq 90\,°$$



# Potential Field Method

## Calculation based on sensor data

$$F_h^i = -\cos\varphi_i \cdot \frac{c}{d_i} \text{ for } 0° \leq \varphi_i \leq 180° \quad (9)$$

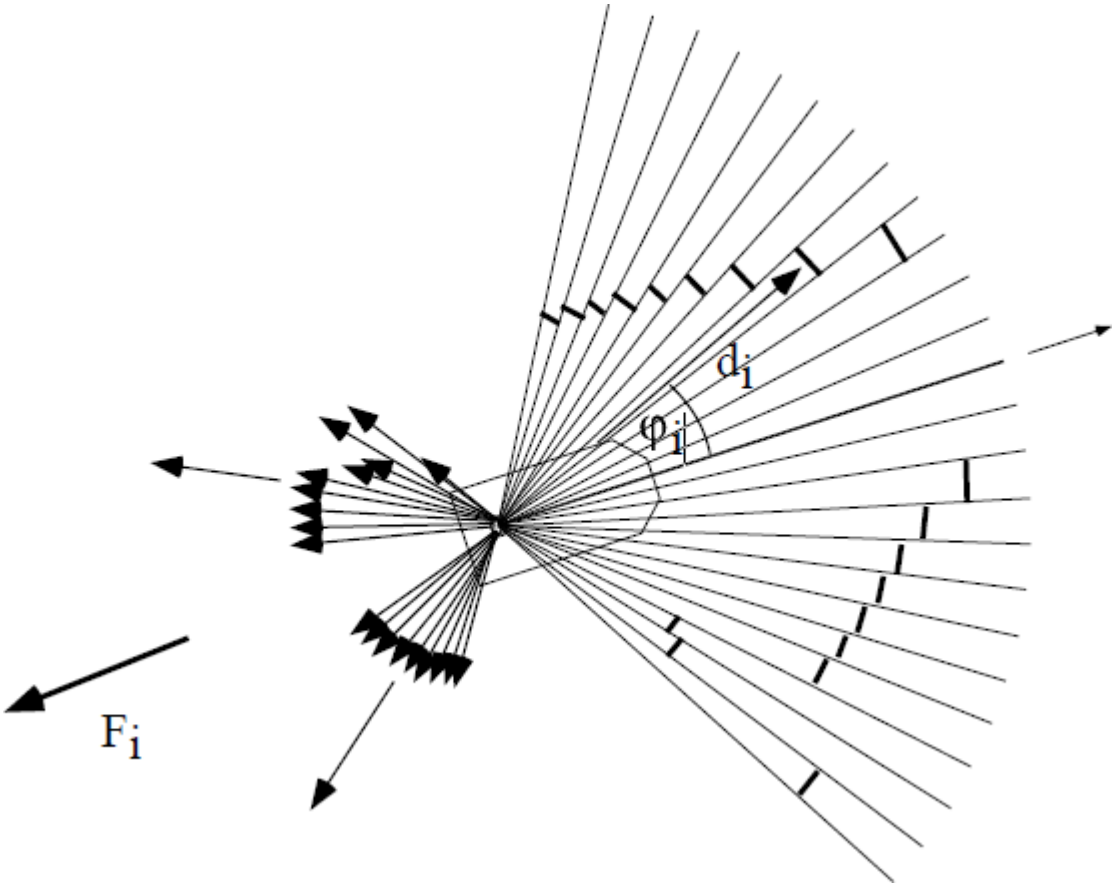



Potential field calculation

# Potential Field Method

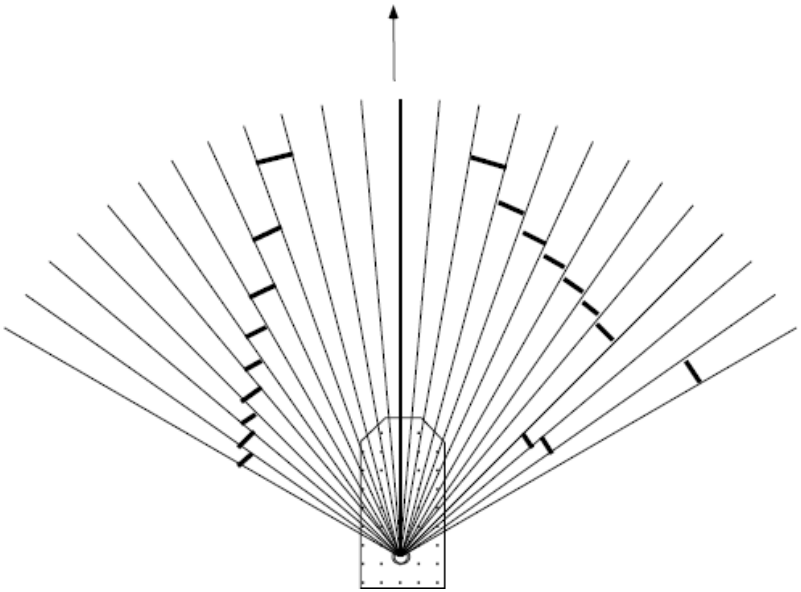
- Resulting force: $F_g(x,y) = F_z(x,y) + \sum_{i=1} F_h^i(x,y)$
  vector field $\left(f_x(x,y), f_y(x,y)\right)$
- Potential field: $F_g = -grad\ \phi(x,y)$ scalar field $\phi(x,y)$
  $grad = \left(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}\right)$ direction of steepest incline
- Robot is driving in direction of resulting force
- Problems
  - Local minimum (target force needs to be strong enough)
  - Narrow passages (target force needs to be strong enough)
  - Robot controller without overshooting
- The contribution of each sector of the map to the resulting force is inverse proportional to the distance of the obstacle and also to the cosine of the angle

# Calculation of Obstacle Force Based on Sector Map

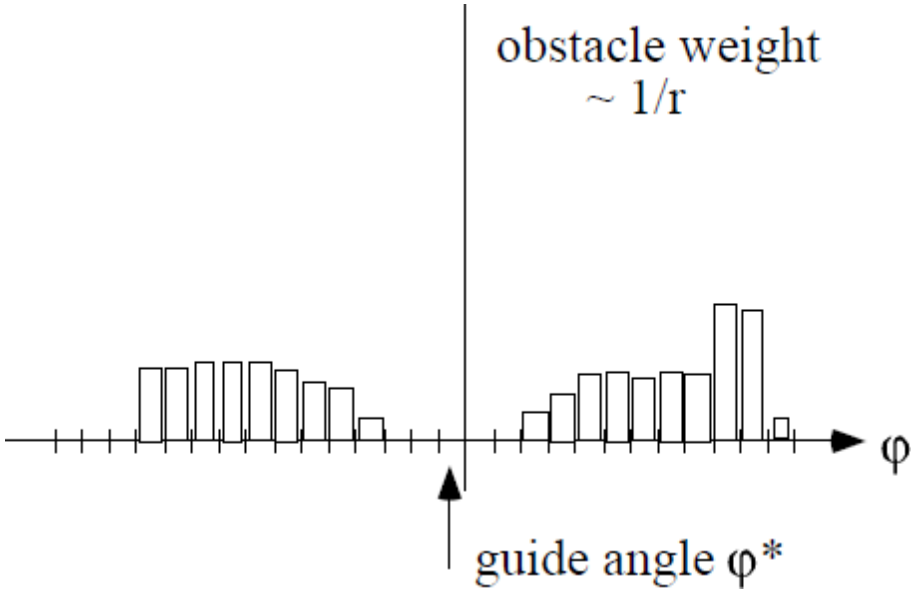



Force based sector map

# Polar Histogram



Sector map

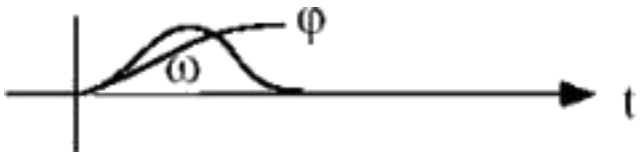



Polar histogram

# Polar Histogram

Control task: choose $\omega(t)$ in a way that $\varphi(t)$ equals the steering angle

$$\varphi(t) = \int_0^t \omega(t)\, dt$$



# Global Path Planning

# Global Path Planning

- Describe path: start point $S \rightarrow$ goal point $Z$
- Side conditions
  - Shortest path
  - Lowest cost (e. g. concerning energy or time)
  - Coverage of free space e. g. for cleaning vehicles
- The total path is often represented by a set of points (subgoals)
- Often these points are nodes in a topological map where the details are to be planned later in a metrical map

# Applications for Global Path Planning

- Minimize the “costs” of traversal in a topological map: A* algorithm

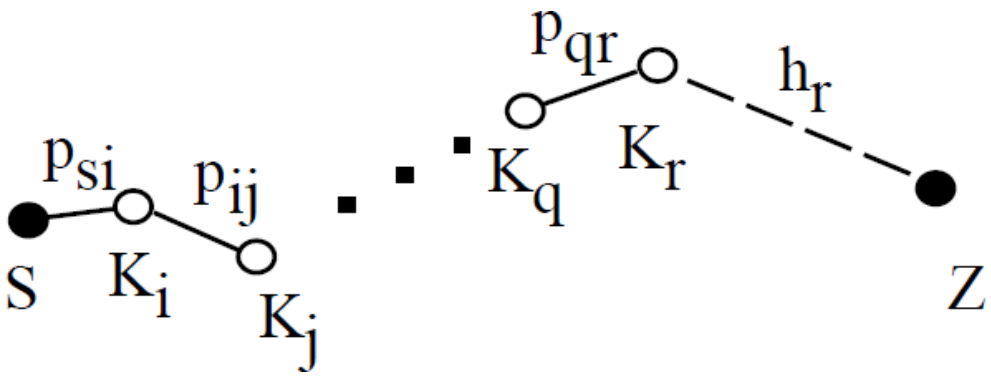
- Find a goal point in an unknown maze and the way back: Backtracking

# A* Algorithm

- Assumptions
  - Connected graph with nodes $K_1, \ldots, K_n$ and edges $s_{ij}$ between nodes $K_i$ and $K_j$
  - Edges $s_{ij}$ are annotated with costs $p_{ij}$ (path lengths, number of narrow curves etc.)
  - Optimistic guess $h_i$ to be payed for the direct route from $K_i$ to the goal $Z$ (at $Z$ is $h = 0$)
  - Costs $g_i$ of already known best path from $S$ to $K_i$
- Wanted: cheapest path for $S \rightarrow Z$

# A* Algorithm

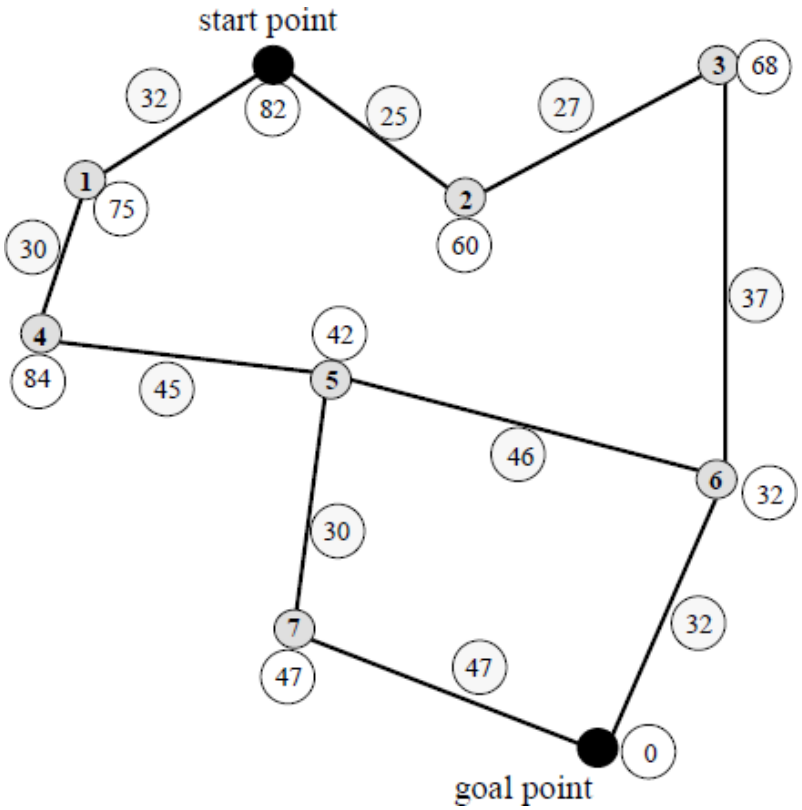
- The total costs of a path $(S, K_i, K_j, \ldots, K_q, K_r)$ are $g = (p_{si} + p_{ij} +$



# A* Algorithm (Algo. 8)

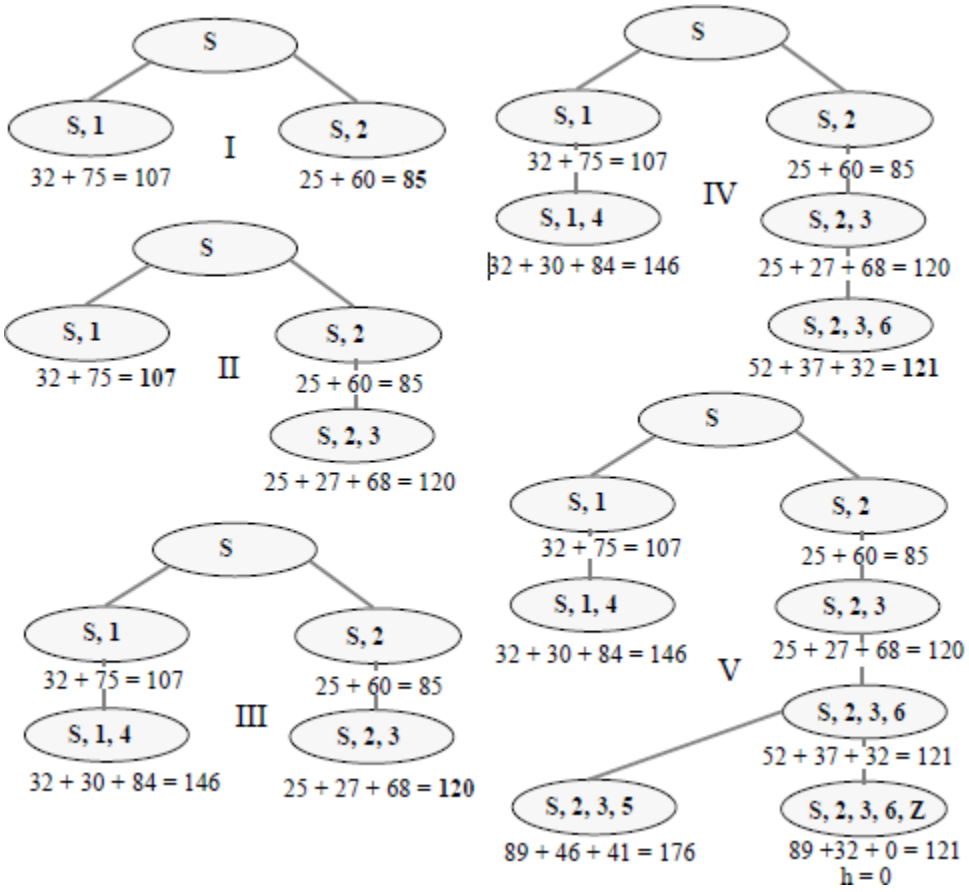
**Require**: graph $(K_i, s_{ij})$, costs $p_{ij}/f_i$, $g_i/h_i$, predecessors $PREDECESSOR_i$, lists $OPEN/CLOSE$, start $S$, goal $Z$
add $S$ to $OPEN$
$g_S \coloneqq 0$
**while** $OPEN$ is not empty **do**
    $K_v \coloneqq$ first from $OPEN$
    **if** $K_v = Z$ **then**
        extract reversed path as $K_v$, $PREDECESSOR_v$, $PREDECESSOR_{PREDECESSOR_v}$, …, $S$
    **end if**
    **for all** successors $K_j$ of $K_v$ that are not in $CLOSED$ **do**
        $g \coloneqq g_v + p_{vj}$
        **if** $K_j$ not in $OPEN$ or $g < g_j$ **then**
            $PREDECESSOR_j \coloneqq K_v$; $g_j \coloneqq g$; $f_j \coloneqq g + h_j$
        **end if**
        **if** $K_j$ not in $OPEN$ **then**
            Add $K_j$ to $OPEN$
        **end if**
    **end for**
    Move $K_v$ from $OPEN$ to $CLOSED$
    Sort $OPEN$ by current cost estimation $f$
**end while**
no path found

# A* Algorithm



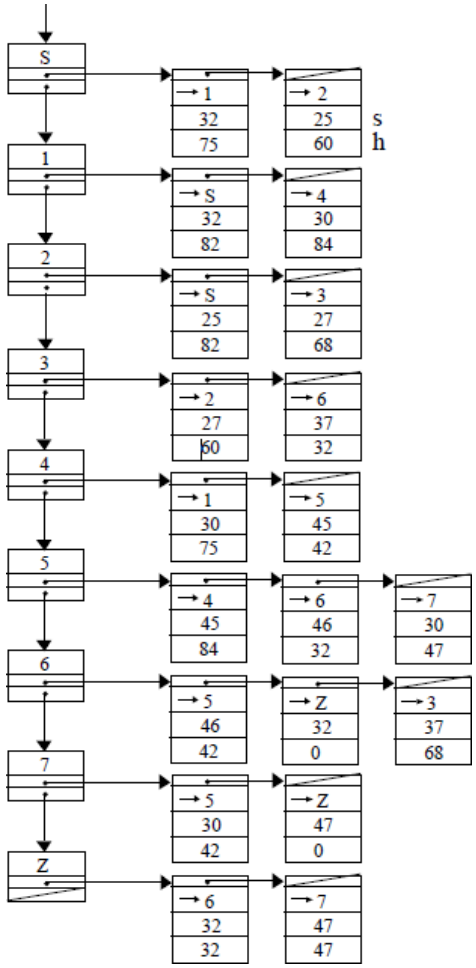

Example of a graph used with A*

# A* Algorithm




Development of the A*-algorithm for the given example

# A* Algorithm

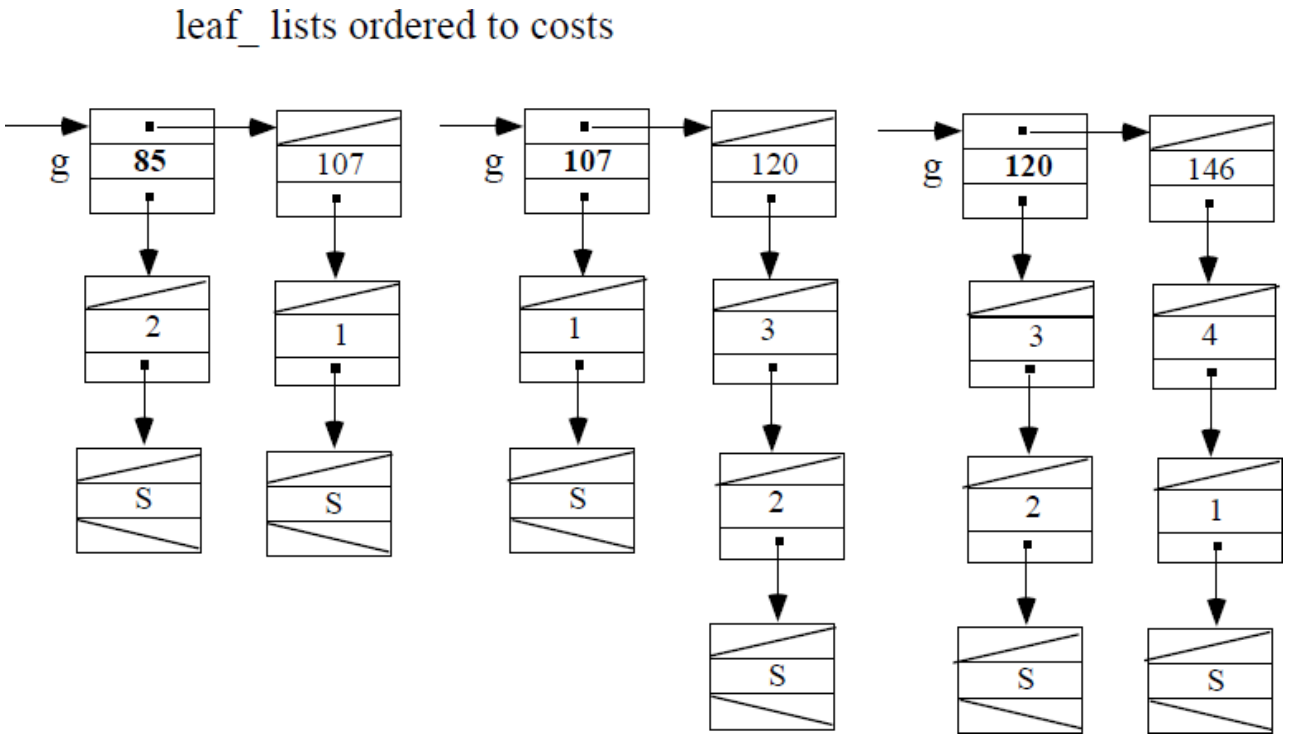



Adjacency list for the example graph

# Implementation of A* Algorithm

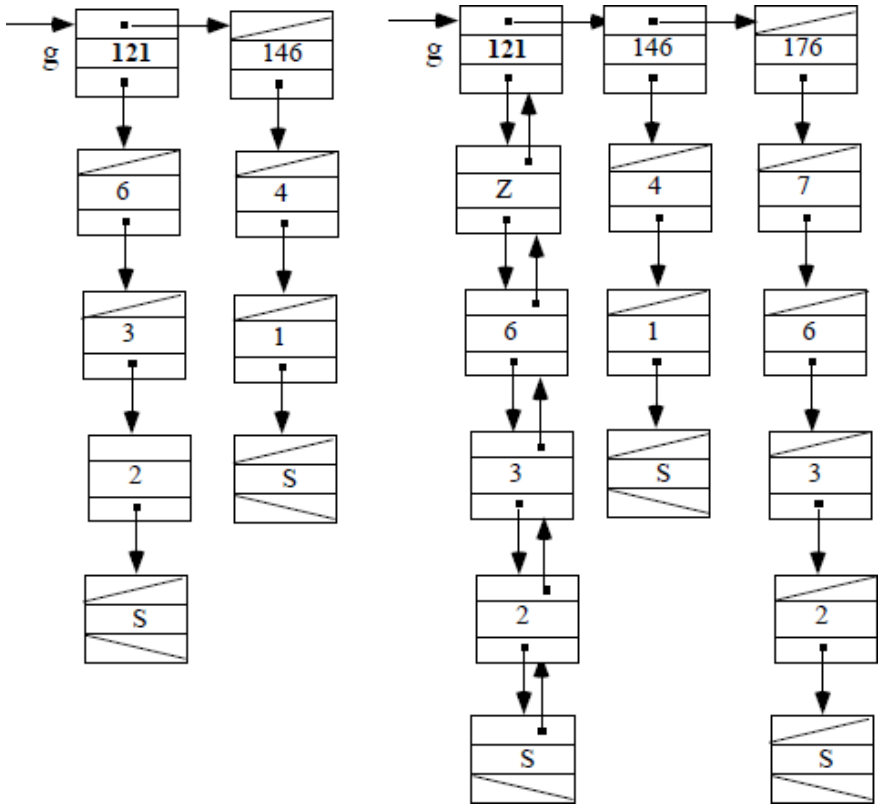
- Leaf list denoting paths already checked
- Pointers oriented from momentary end node to the start node
- Once an optimal path has been found the path list is reversely pointered from the start to the goal

# Implementation of A* Algorithm




Leaf list representing the first part of the development of the A*-algorithm for the example (ordered according to the costs)

# Implementation of A* Algorithm




Leaf list representing the second part of the development of the A*-algorithm for the example (ordered according to the costs)

# Navigation in an Unknown Maze

- Task: Find path through maze to (hidden) goal Z and back to entrance without a map
- “hidden goal” means: Z might be unknown until arrival
- No offline planning

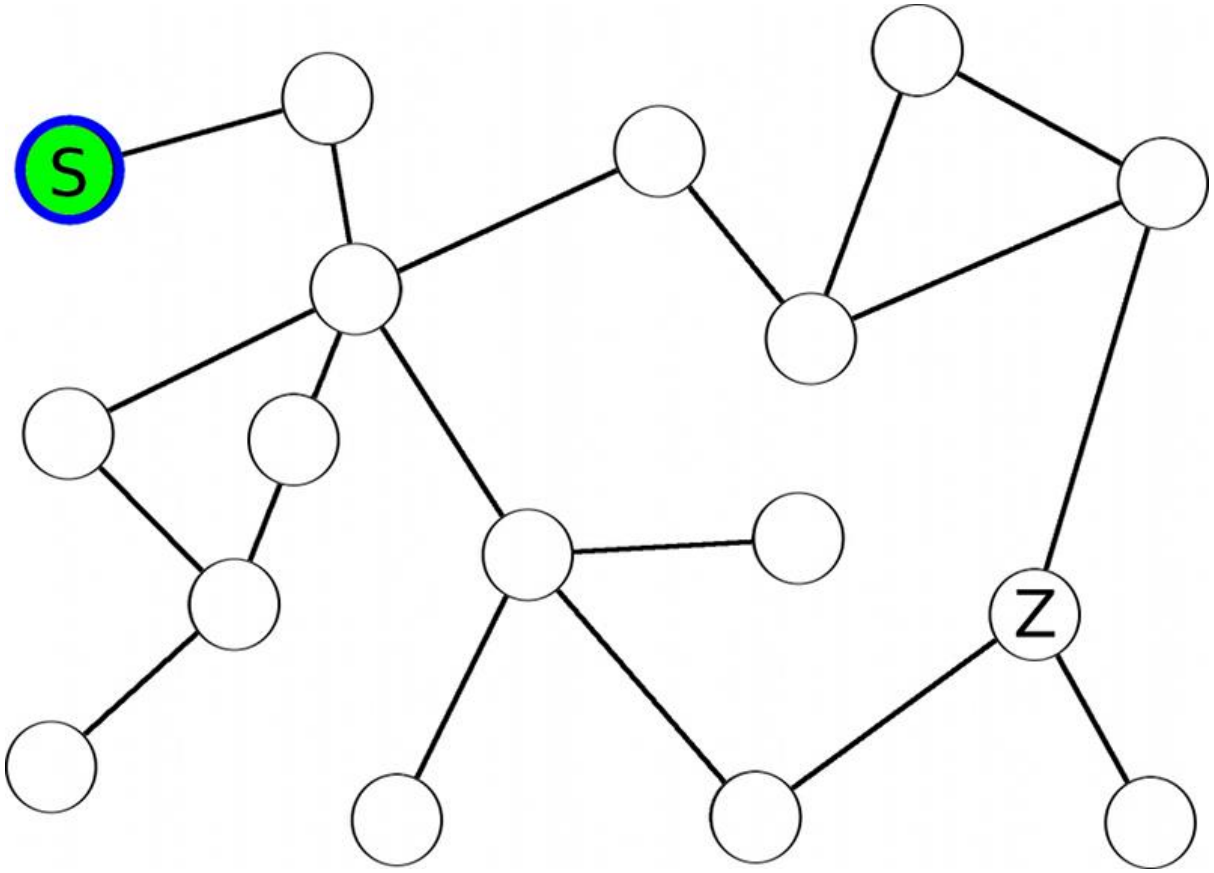
# Navigation as Depth-First Search in Graph

- Typical solution to graph problem: depth-first search
- Traversal of spanning tree → detect cycles
- Underlying principle of algorithmic class “Backtracking”, Trial-and-Error, Eight Queens Puzzle, …
- Allows to explore complete search space in finite time
- Typically uses recursion, but needs iterative implementation for robotic application

# Iterative Backtracking Algorithm for Navigation

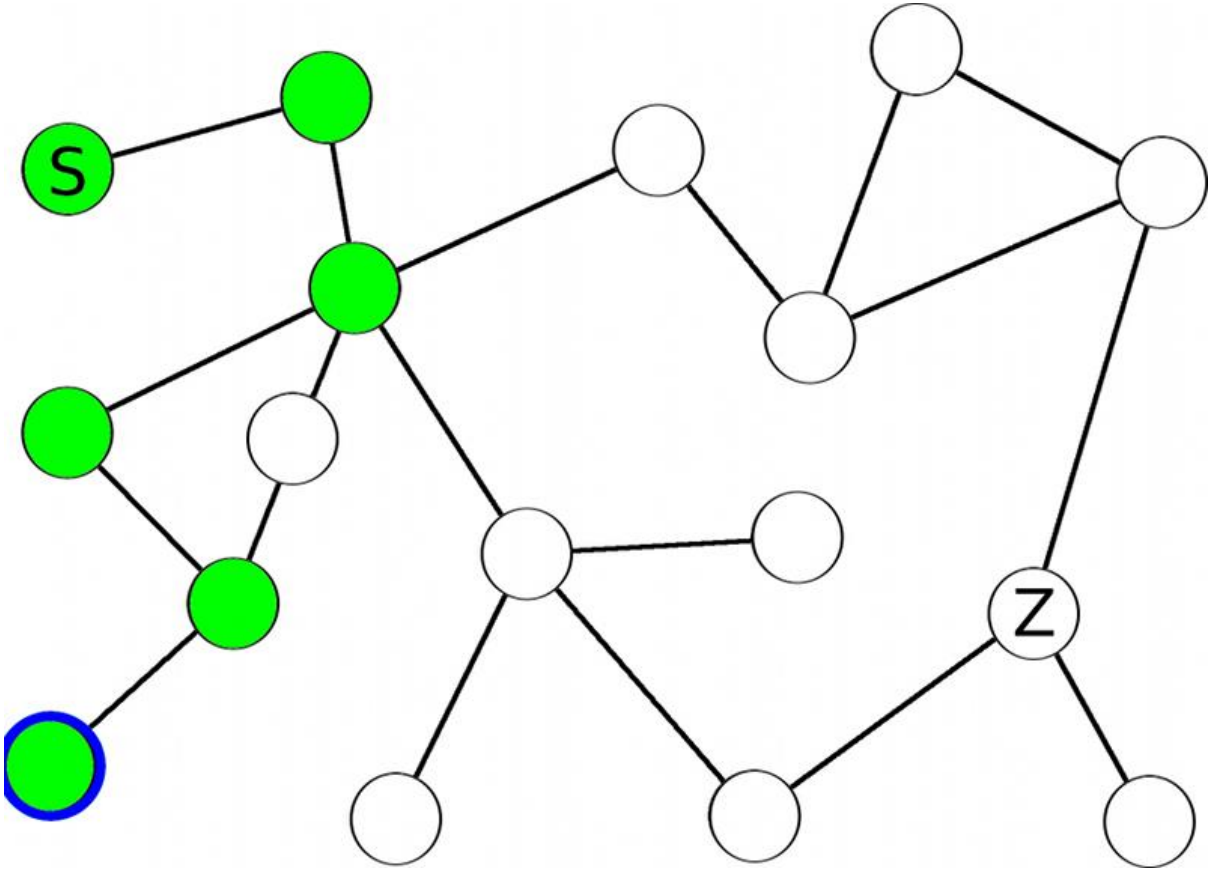
**Require:** graph $(V, E)$, markers $OPEN/CLOSED$, start $S$, goal $Z$, $S \neq Z$
Mark $S$ as $OPEN$; $V_{current} \coloneqq$ any node directly connected to $S$;
$E_{entry} \coloneqq$ edge from $S$ to $V_{current}$; $exiting \coloneqq false$
**while** $\neg exiting \wedge V_{current} \neq S$ **do**
    **if** $V_{current} = Z$ **then**
        $exiting \coloneqq true$
    **end if**
    **if** $\neg exiting$ **then**
        $E_{exit} \coloneqq$ first edge counter-clockwise starting at $E_{entry}$ that leads to umarked node $V_{next}$
        **if** $E_{exit}$ **then**
            Mark $V_{current}$ as $OPEN$; $V_{current} \coloneqq V_{next}$; $E_{entry} \coloneqq E_{exit}$
        **end if**
    **end if**
    **if** $V_{current}$ was not already updated **then**
        $E_{exit} \coloneqq$ first edge clockwise starting at $E_{entry}$ that leads to $OPEN$ node $V_{next}$
        Mark $V_{current}$ as $CLOSED$; $V_{current} \coloneqq V_{next}$; $E_{entry} \coloneqq E_{exit}$
    **end if**
**end while**

# Example for Backtracking Algorithm



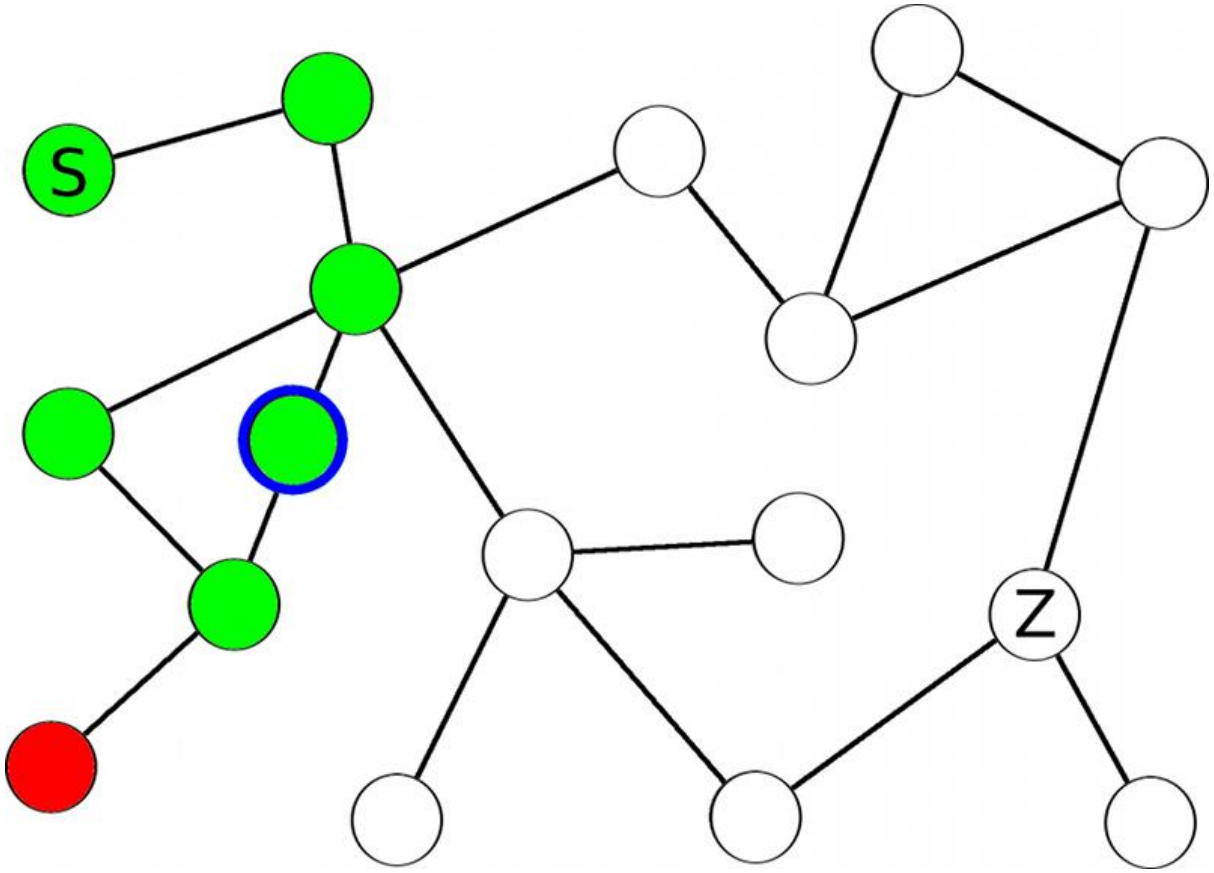

Example for backtracking: blue = current node,
green = open, red = closed

# Example for Backtracking Algorithm



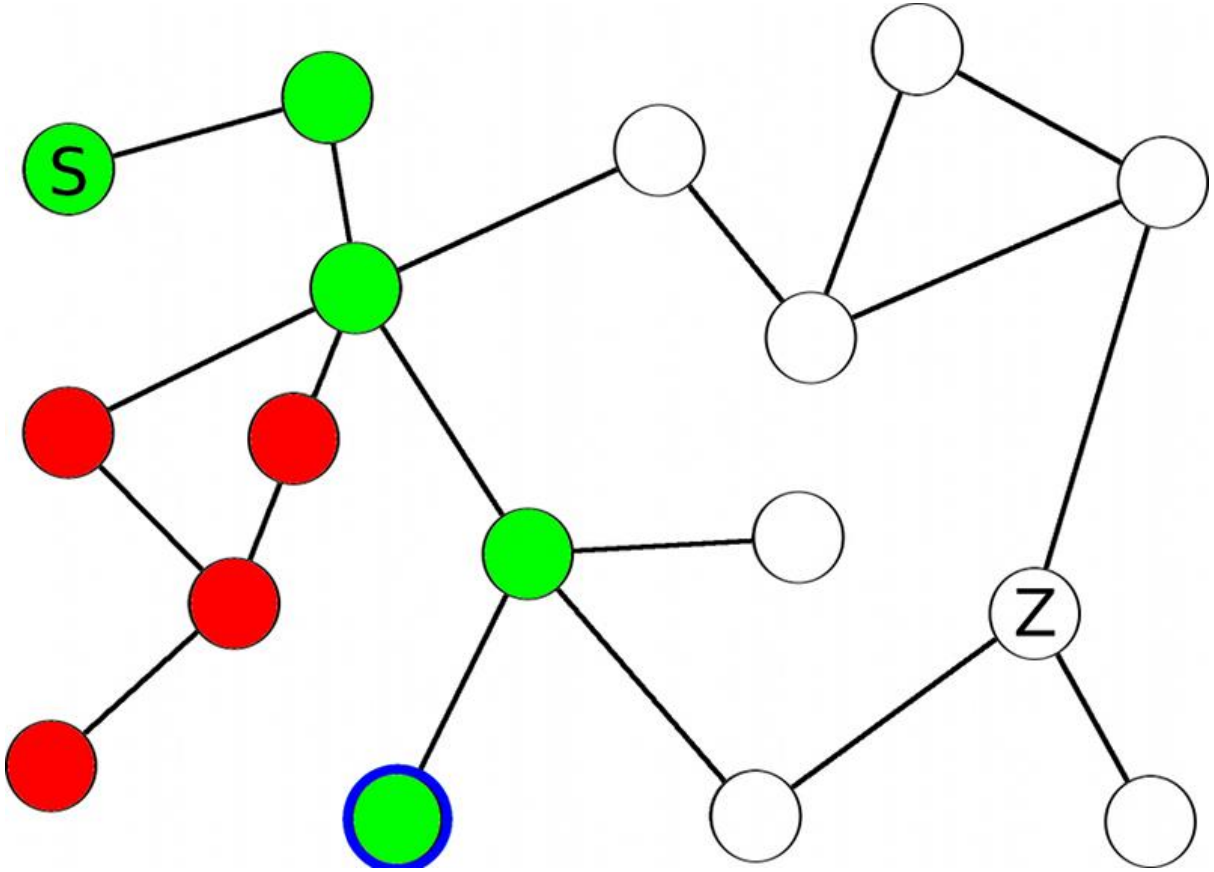

Example for backtracking: blue = current node,
green = open, red = closed

# Example for Backtracking Algorithm



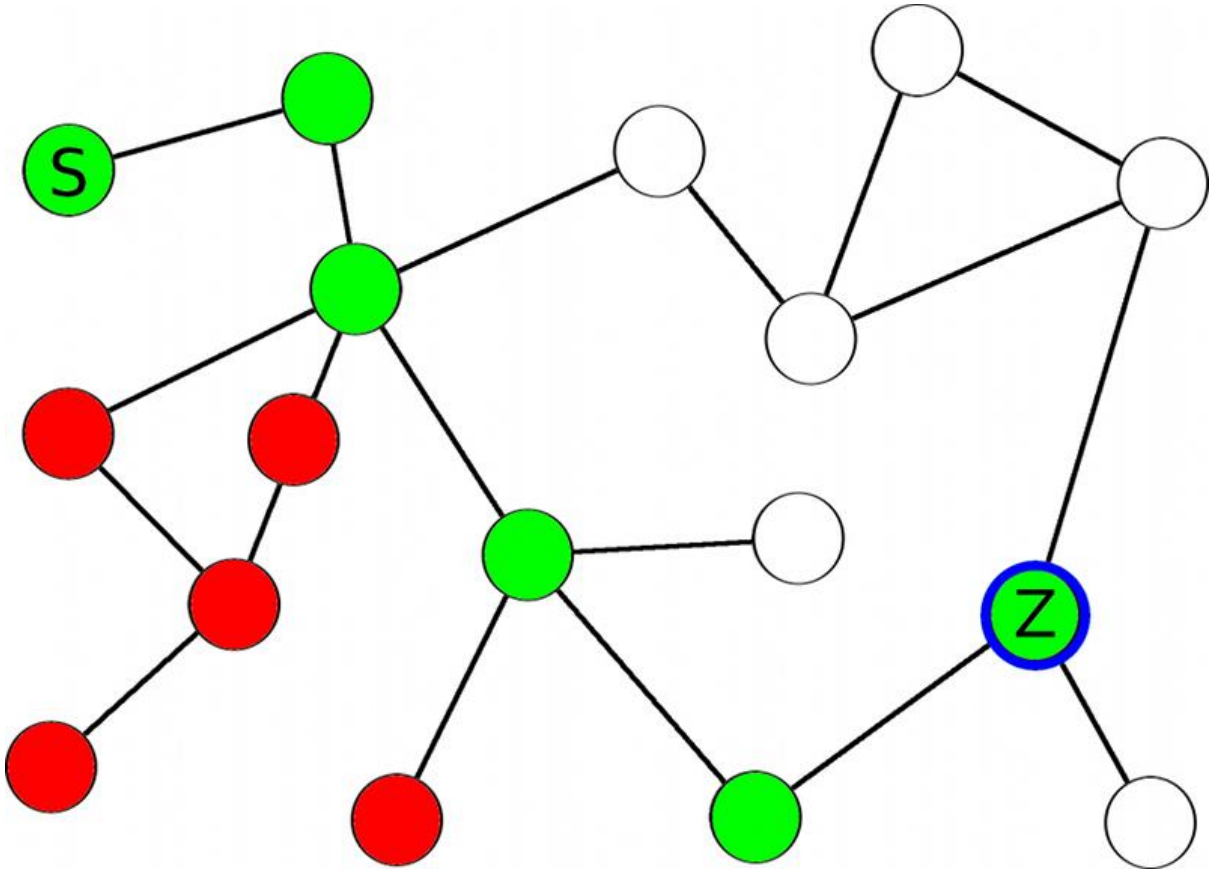

Example for backtracking: blue = current node,
green = open, red = closed

# Example for Backtracking Algorithm




Example for backtracking: blue = current node,
green = open, red = closed

# Example for Backtracking Algorithm




Example for backtracking: blue = current node,
green = open, red = closed

# Application on Real Robot

- A real robot needs a method to "mark" nodes as $OPEN$ or $CLOSED$

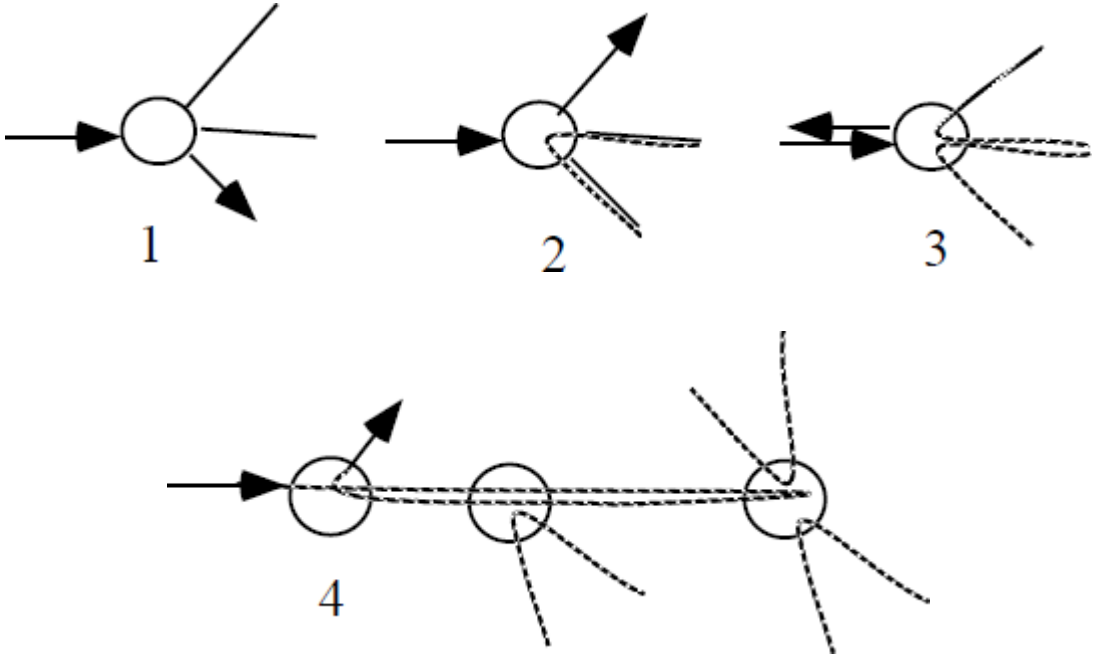
1) Construction to drop and collect RFID tags

2) Construction to unroll and detect a string

   - Greek mythology: The string Ariadne gave to Theseus

   - Today this principle is used in e.g. cave research

# String of Ariadne




Using a string to solve a maze

# Coming Next

## Control Architectures