# - Tutorial -
# *Motion autonomy & Robust navigation in Robotics*

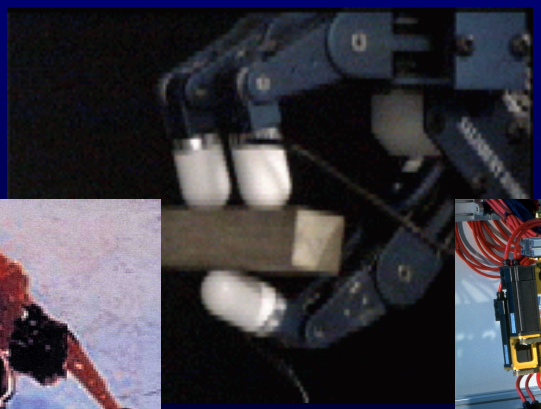
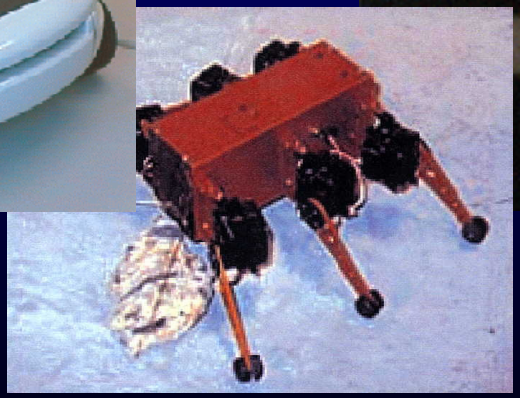
***Christian LAUGIER***

*Research Director at INRIA Rhône-Alpes, France*

- **Turorial 1** *(3h, C. Laugier)*
  ***Motion planning & Reactive control architecture***
  1. **Basic geometric models & algorithms for motion planning**
  2. **Dealing with real world constraints**
     *=> kinematic, dynamicity, uncertainty & hazard*
  3. **Reactive navigation techniques**

- **Tutorial 2** *(3h, O. Aycard)*
  ***Bayesian programming & Robust decision making***

# - Part I -
# Basic Geometric models & Algorithms for Motion Planning


# *Motion of artificial systems*

*Manipulator & mobile robots, Articulated hands, Drones, Automated vehicles … Virtual characters*

## 1. Motion planning

Geometric world modeling + A priori determination of the motions that will take a robotic system from its current *« configuration » (i.e. position & orientation of each individual components of the robot)* to a given goal « configuration »

*e.g. walking, moving around, grasping and mating objects …*

⇒ ***Motion planning is a fundamental problem in robotics***
*(largely addressed since the late 60's))*

## 2. Reactive navigation

On-line adaptation of a motion plan according to execution conditions (hazard perception in particular)

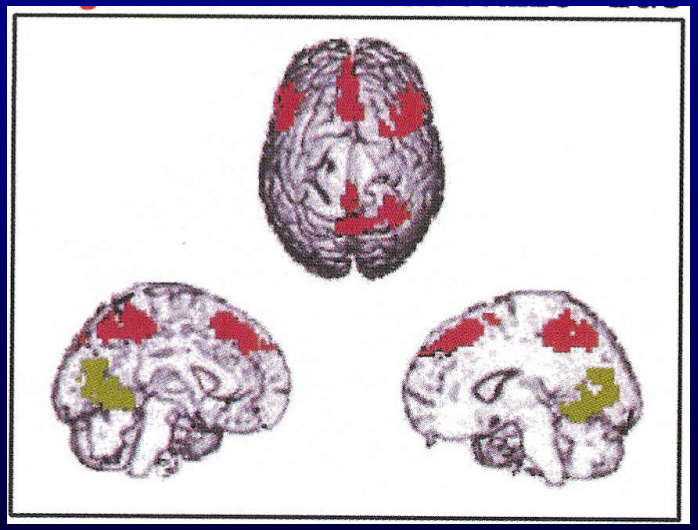
**=> *Making use of some additional models*** *(sensori-motors, behaviors …)*

# *Something to learn from biological systems ?*

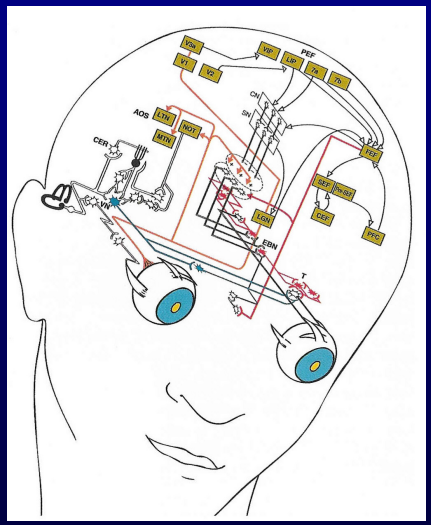
*[Berthoz 01]*

## Spatial orientation & Memory of routes :

- ***Egocentric*** **coding &** ***Allocentric*** **coding**
- ***Survey of map*** **like strategy :** *Mental map => Topo-kinetic memory*
- ***Route*** **like strategy :** *Memory of motions & perception (eyes, vestibular, muscles ...) => Topo-kinesthesic memory*



*Brain areas involved in:*
*- Egocentric tasks => parietofrontal regions*
*- Allocentric tasks => parietotemporal regions*

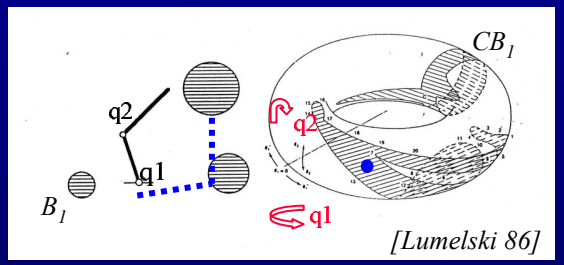


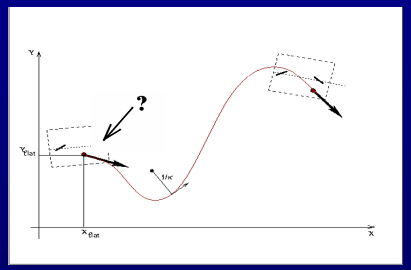
*Network of structures contributing to saccadic eye movements (brain areas, vestibular system, muscles) [Berthoz 97]*

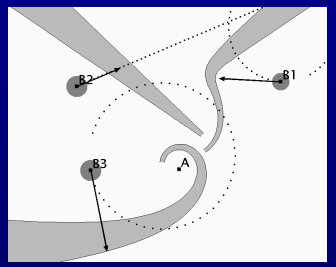
*=> Several models & planning / navigation strategies are combined*

# How to construct computational models ?

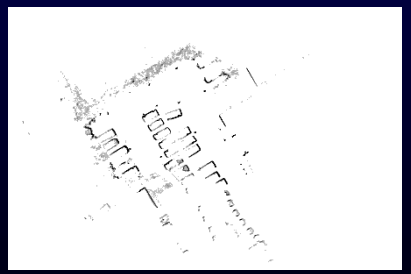
- **"Mental Map" for Motion Planning *(MP)***
  - ✓ *Mainly geometrical & kinematic models*
  - ✓ *Appropriates representations (Configuration space, Velocity space ...)*
  - ✓ *Appropriates algorithms (collision checking, graph search, random search ...)*

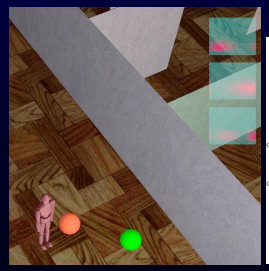


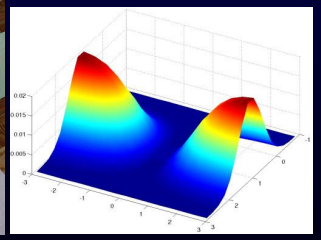




- **"Route like strategies" for Reactive Navigation**
  - ✓ *Mainly reactive architectures & probabilistic models*
  - ✓ *Appropriate representations (sensori-motor schemes, behaviors ...)*
  - ✓ *Appropriate approaches (reactive architectures, learning ...)*







# Basic Motion Planning Problem

***The Piano Mover's Problem***

- **"Free flying" mobile (rigid object(s))**
- **Stationary obstacles**
- **A priori known geometric model**

=> Compute a *path* (continuous sequence of collision-free "position-orientations") between a start and a goal "position-orientation".
*Report failure if no such path exists*

=> The focus is on the *geometric aspects* of motion planning. Physical and temporal issues are disregarded. *Collision avoidance* is the key issue.

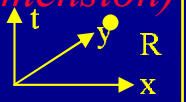
*Though simplified, this problem is already difficult. It is of practical interest and several of its solutions can be extended for more involved motion planning problems*

# *Extensions of the basic problem*

**1- Mobile obstacles** (not under control)

Planning a geometric path is not sufficient, *time* has to taken into account !

*=> Motion = Time sequence of collision-free configurations (adding the time dimension)*



Configuration-Time Space

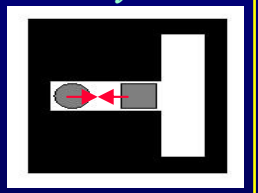
**2- Multiple mobile systems**

– Centralised approach : *The whole system is considered as a « multi-bodied mobile »*

$\Rightarrow A = \cup A_i \Rightarrow Search-space(A) = \prod Search-space(A_i)$

– Decoupled approach : *Each mobile component is considered separately + Priority*

*=> Complexity is reduced at the expense of completeness !*



**3- Kinematic constraints**

– "Holonomic" constraints

*e.g. an arm carrying a glass of water (suppressing some rotation directions)*

*⇒ Restricting the set of admissible configurations (i.e. the search space)*

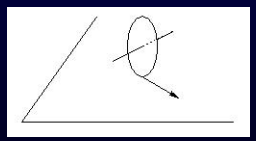
$W = \Re^3 \times SO^3$ *and* $"Search-space" = \Re^3 \times SO^1$

– "Non-holonomic" constraints

*e.g. a wheel rolling without slipping , or a car*

*=> any (x,y,θ) can be reached, but (dx/dy = tg θ) !!!*

*=> Restricting the set of admissible "differential motions" (i.e. path shape constraints)*



# *Extensions of the basic problem (C'ed)*

**4- Dynamic constraints**

The *physical properties* of the mobile reappear !

*e.g. forces & torques, velocity & acceleration bounds, inertia, etc*

*=> Trajectories have to be considered instead of geometrical paths (State-Time space)*

**5- Movable objects** (manipulation)

To reach a goal, *objects may have to be displaced* (e.g. through grasping)

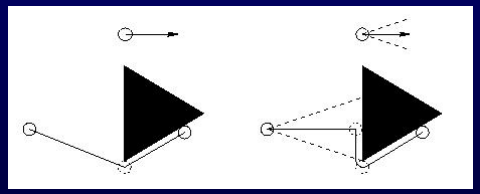
*=> Motion = Alternate sequence of "move" and "transfer" (Manipulation space)*

**6- Uncertainty**

– Perception & Sensing errors

*=> Modeling uncertainty*

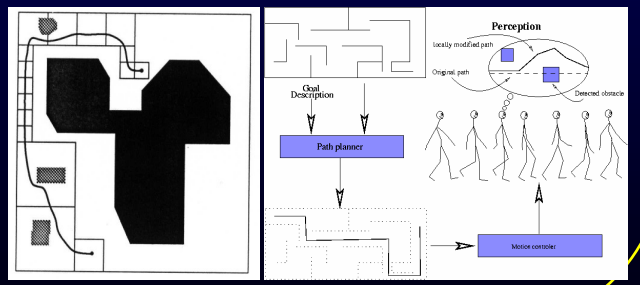
*& Using robust sensor-based strategies*



– Partly known workspace

*=> Path planning (producing a set of admissible paths)*

*& Sensor-based obstacle avoidance*



# A key concept : "Configuration space"

*[Udupa 77; Lozano-Perez 83]*

The initial *MP* formulation in the workspace is not tractable !!!

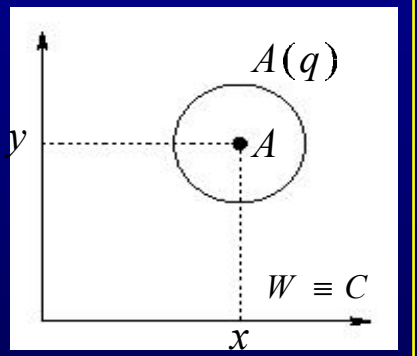
*(i.e. continuous sequence of collision-free "position-orientations" of all the robot components)*

=> Configuration Space is a fundamental tool to address motion planning

- ***Configuration of a mobile system***

*Minimal set of independent parameters uniquely specifying the position & orientation of every component of a mobile system*

*=> mobile = a point in $C_{space}$*




*e.g. A disc translating in the plane ($W = R^2$)*
*=> q = (x,y), position of the reference point A*

- ***Configuration space***

*=> Space of all the configurations of the mobile system*

*e.g. $C_{space} = R^2 = W$ for the disc in the plane (particular case)*

# A key concept : "Configuration space" (C'ed)

- **"Correspondence" between *C* and *W* for the mobile *A***

  *A(q)*: subset of *W* occupied by *A* at configuration *q*

  *=> for collision checking in W (i.e. does A(q) intersect obstacle B ?)*

- **"Correspondence" between *C* and *W* for the obstacles *B***

  *Obstacles B = subsets of W, $B_i$ , i = 1…k*

  *C-obstacle (B) = subsets of C such as A(q) intersect B*

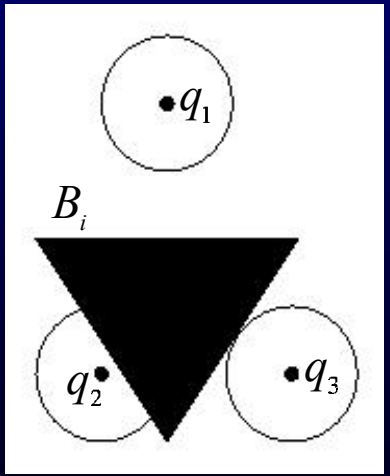
  *=> for representing forbidden configurations in C (i.e. C-obstacles)*

*Free space :* *q* is *free* iff $\forall i, A(q) \cap B_i = \emptyset$

*Contact sets :* *q* is in *contact* iff $\exists i, A(q) \cap \delta Bi \neq \emptyset$

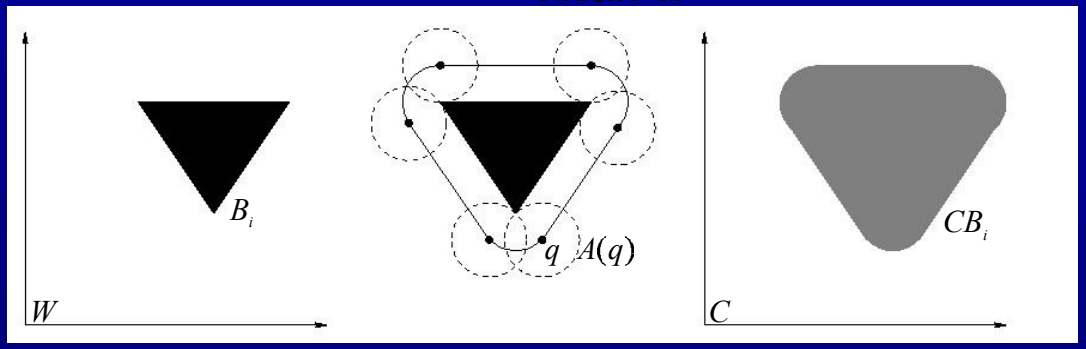
*Collision sets :* *q* is in *collision* iff $\exists i, A(q) \cap \text{int } B_i \neq \emptyset$

$$C = C_{free} \cup C_{contact} \cup C_{collision}$$



# Path planning using $C_{space}$

- **Representing the $C_{obstacles}$**



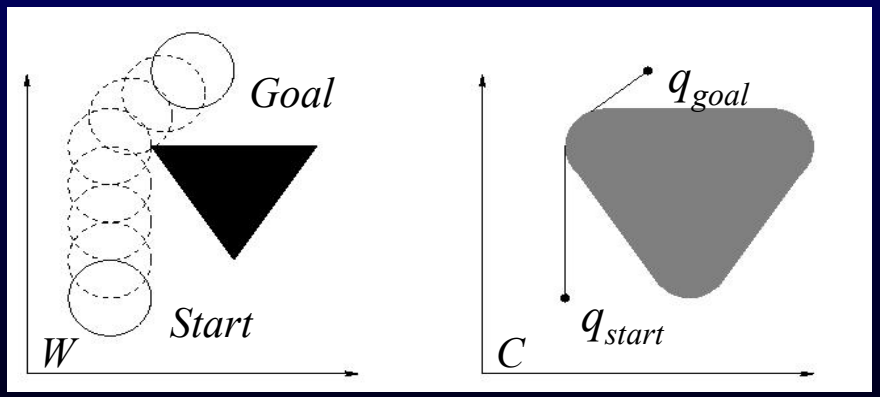

$$CB_i = \{q \mid A(q) \cap B_i \neq \phi\}$$
$$C_{free} = C - \bigcup_i CB_i$$

- **Path planning for the robot *A* in *W***

  ⇔ path planning for the point $q$ in $C$

  ⇔ search for a *collision-free path*, *i.e.* a curve of $C_{free}$ (or $C_{free} \cup C_{contact}$)

  path $\pi$:[0, 1] → $C_{free}$ (or $C_{free} \cup C_{contact}$)



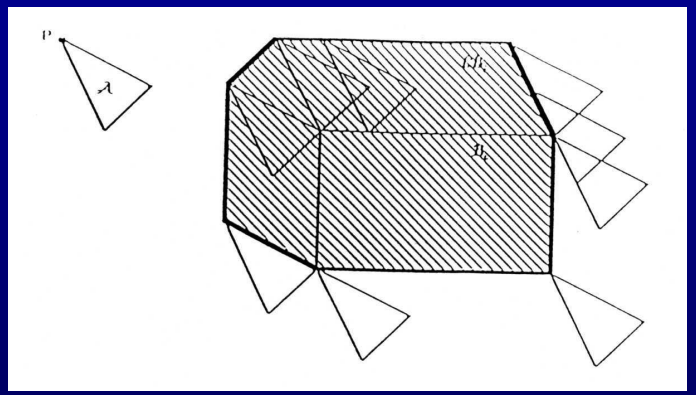

**⇒ General path planning approach :**
***Compute, Structure, then Explore $C_{free}$***

*It looks conceptually simpler but ...*
*how to do that in practice ?*
*how to deal with rotations ?*

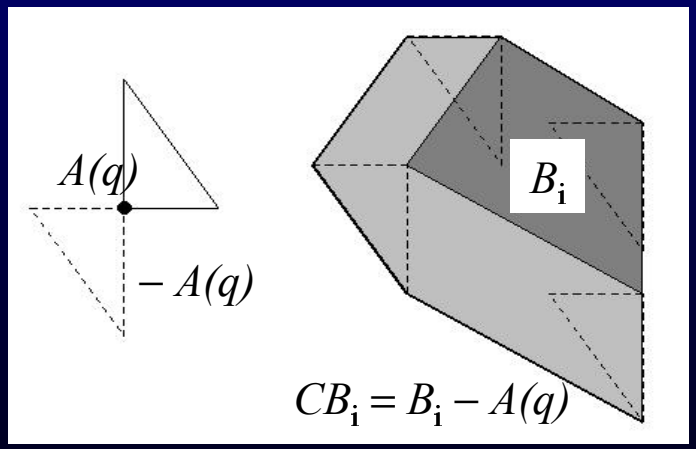
# Constructing $C_{obstacle}$

**Case study #1:** Polygonal robot translating amidst polygonal obstacles

$W = R^2$, $q = (x, y)$ $\Rightarrow$ $C = R^2$



Intuitive C-obstacle construction:
*"sliding" A along $B_i$ boundaries*
*=> $CB_i$ = locus of ref point R of A*



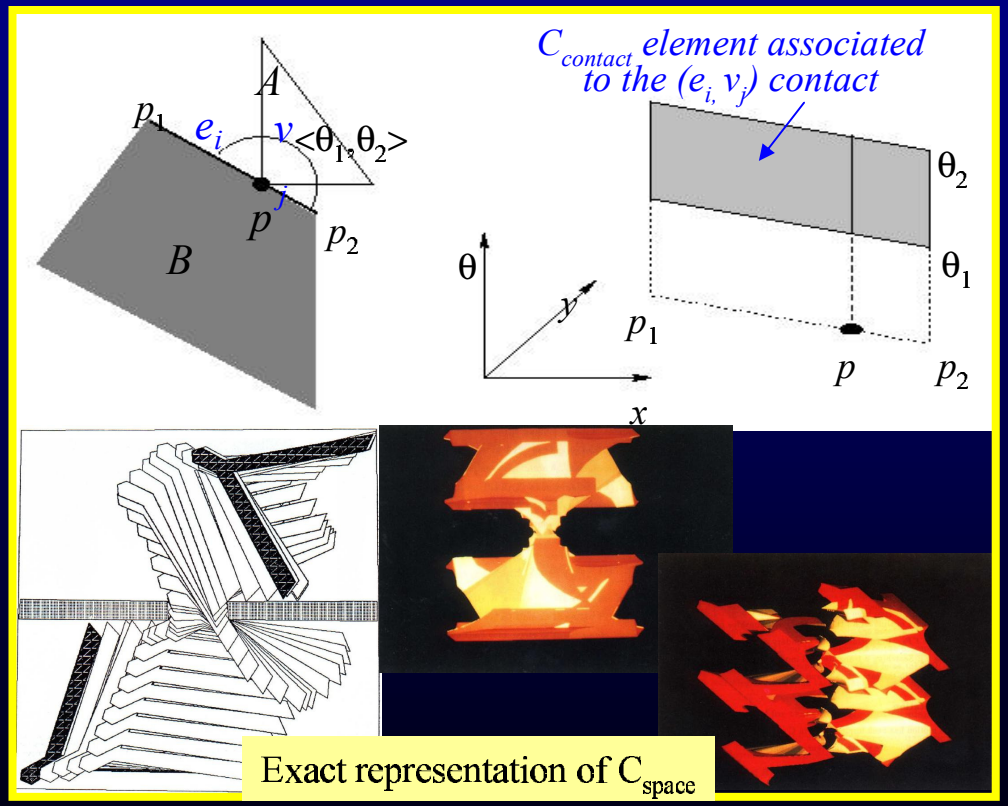

Exact C-obstacle calculation :
*=> using the Minkowski's sum*
*("growing $B_i$ inversely to the shape of A")*

# Constructing $C_{obstacle}$ (C'ed)

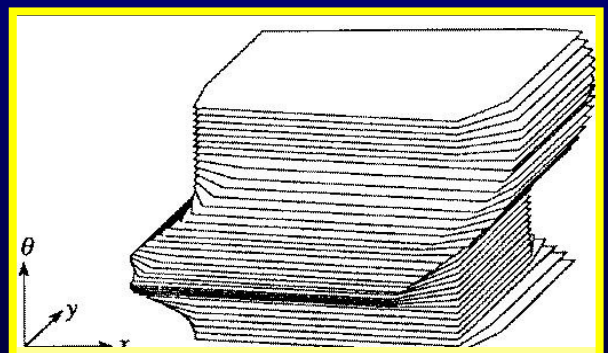
**Case study #2 :** Polygonal robot translating & rotating amidst polygonal obstacles

$W = R^2$, $q = (x, y, \theta)$ $\Rightarrow$ $C = R^2 \times S$

Exact C-obstacle calculation still possible but complex ! *[Avnain & Boissonnat 89]*
*=> $C_{contact}$ (i.e. the boundary of $C_{free}$) = Set of ruled surfaces created by the translations and rotations of Cspace segments associated with elementary contacts (edge, vertex)*



Exact representation of $C_{space}$

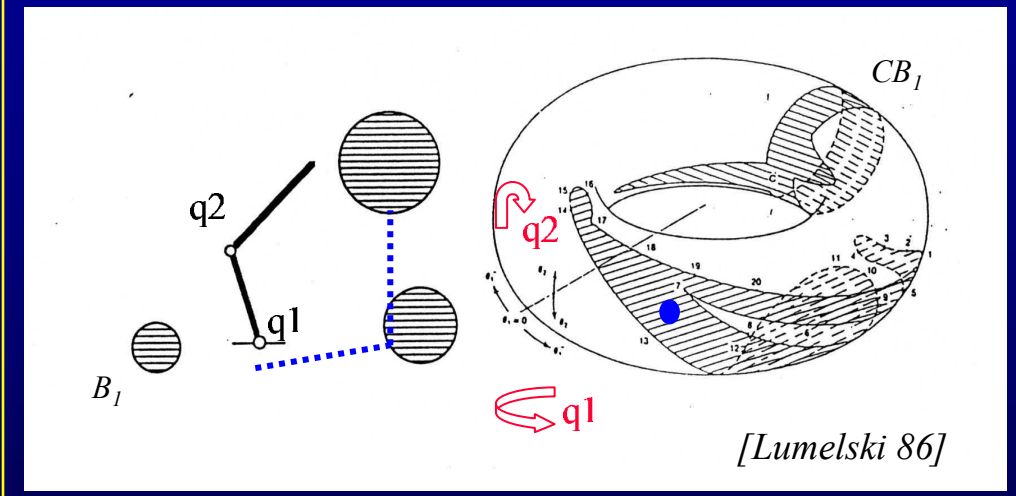


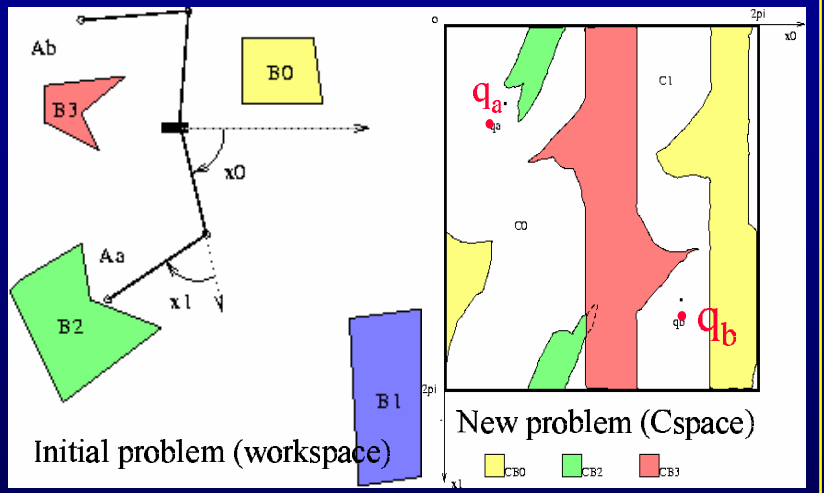
Approximated C-obstacle (θ–slices)

# Constructing $C_{obstacles}$ (C'ed)

**Case study #3:** 2-links planar manipulator arm amidst 2D obstacles

$W = R^2$, $q = (\theta_1, \theta_2)$ $\Rightarrow$ $C = S^2$



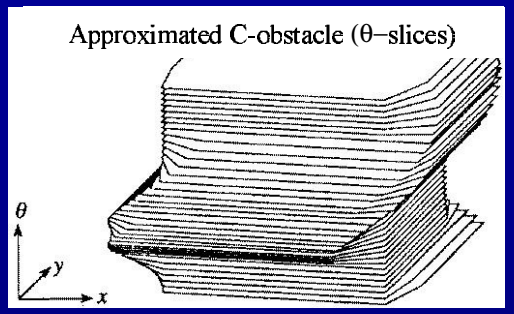

*=> No tractable general method exists for computing an exact representation of $C_{obstacle}$ (and of $C_{free}$) in the general case !!!*

*=> Appropriate approaches have to be designed for constructing & searching a "manageable" representation of $C_{free}$*

# How to do this in practice ?

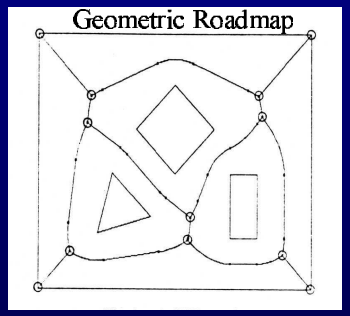
- **Cell decomposition**

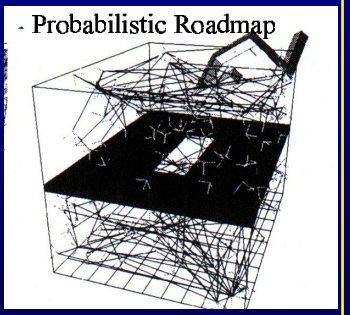
  *Exact or approximated decomposition of $C_{space}$ for constructing & searching a graph representing the topological & geometric structure of $C_{free}$*




- **Roadmaps**

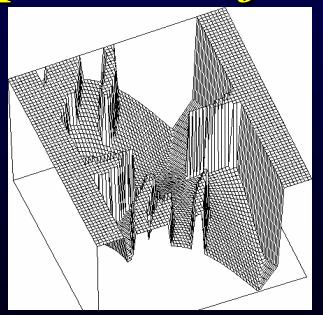
  *Constructing & searching a network representing the connectivity of $C_{free}$ (geometric or probabilistic approaches can be used)*

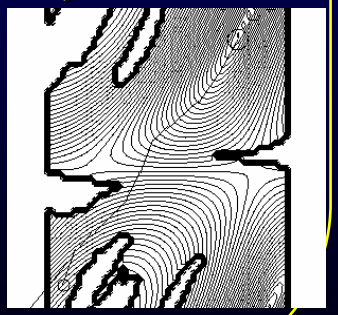



- **Gradient descent methods *(e.g. Artificial potential fields)***

  *Representing $C_{free}$ using an underlying energy / cost function, and searching this space using a gradient descent method*

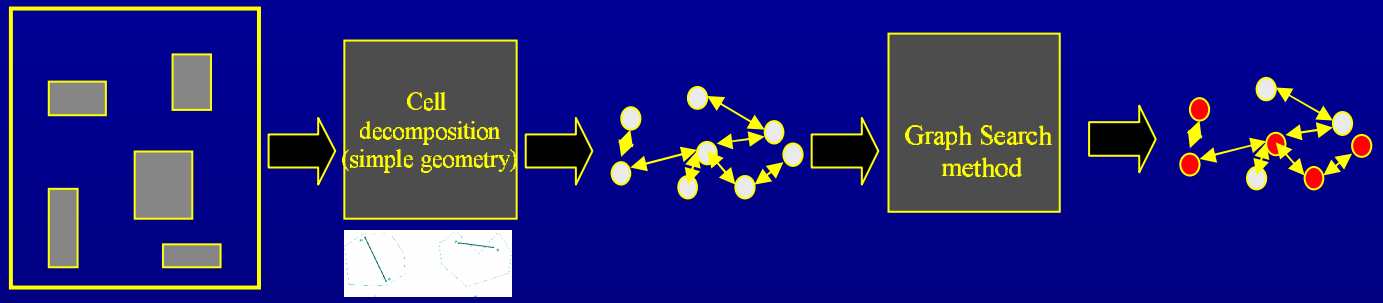




# "Cell decomposition" approaches

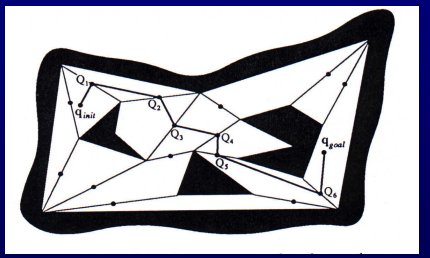
*Constructing and searching a « Connectivity Graph » representing $C_{free}$*

- **Basic idea**




- **$C_{space}$ representation**
  - **Exact decomposition** $\bigcup_{i=0}^{m} c_i = C_{free}$

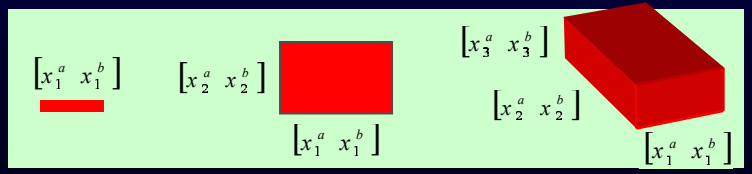
    *=> Complete, Reduced number of cells … But difficult to build !*
    *(apply only for some simple cases)*



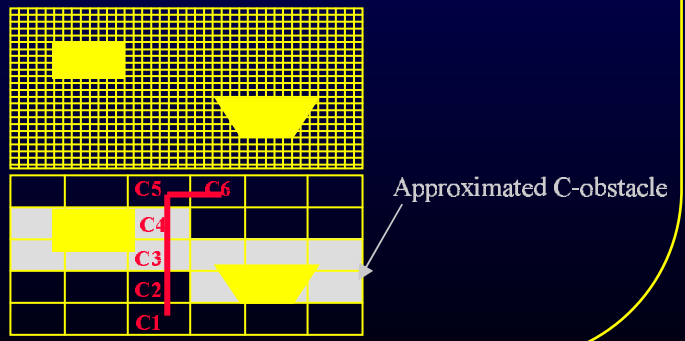
  - **Approximated decomposition** $\bigcup_{i=0}^{m} c_i \subset C_{free}$

    *=> Complete at a given resolution only, Large number of cells*
    *… Much more easy to implement (cells of predefined shape, e.g. rectangloids or $2^m$-trees) !*

$$Cell_k = \{(x_1, \ldots, x_n) | x_1 \in [x_1^a, x_1^b], \ldots, x_n \in [x_n^a, x_n^b]\}$$

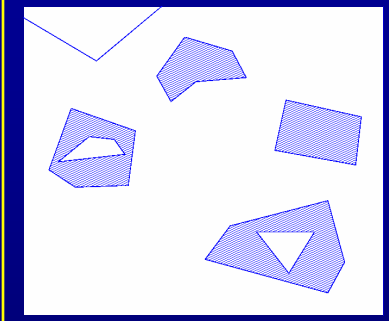



*A widely used decomposition technique : "Rectangloids"*

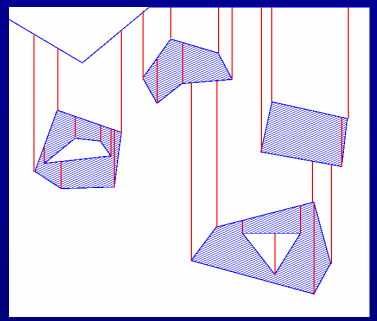


# *Some "Cell decomposition" algorithms*

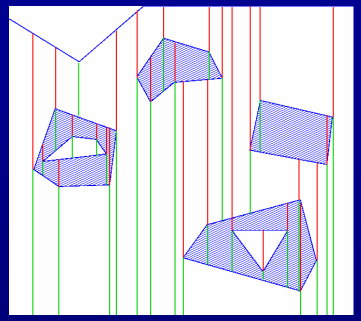
- **Exact decomposition using trapezoids (in $R^2$)**



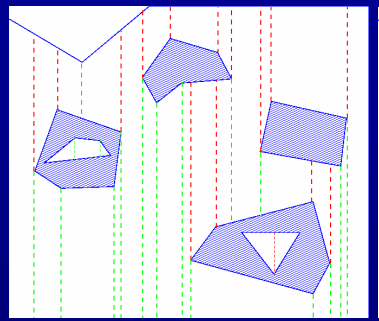
C-space



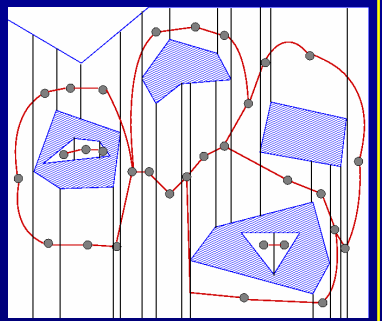
Upward extensions



Downward extensions

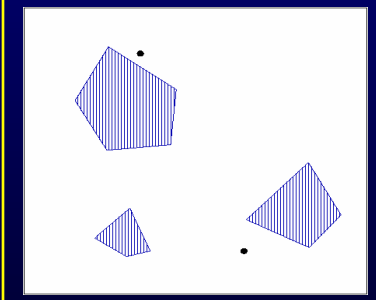


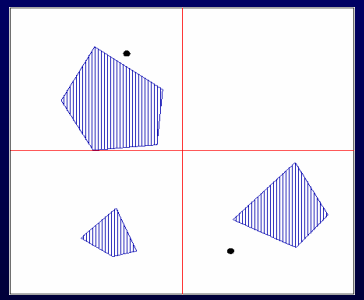
Erasing the trapezoids inside the obstacles

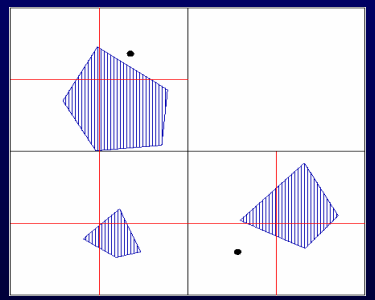


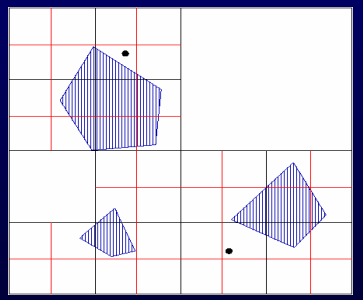
Constructing the connectivity graph

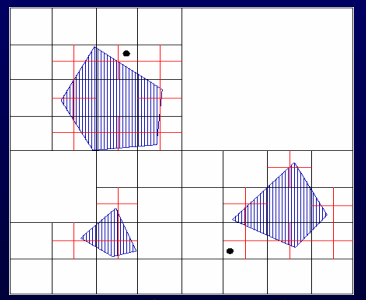
- **Approximated decomposition using $2^m$-trees**



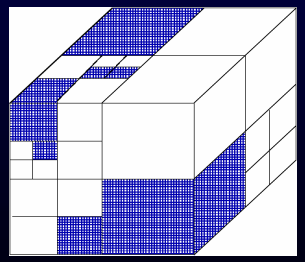








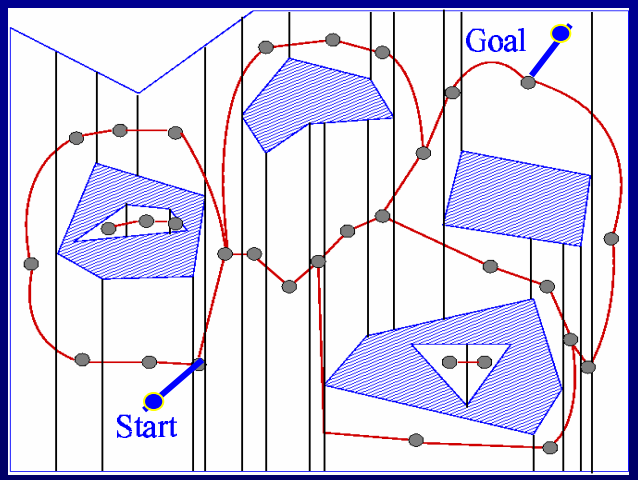
*Quadtree (2D)*



*Octree (3D)*

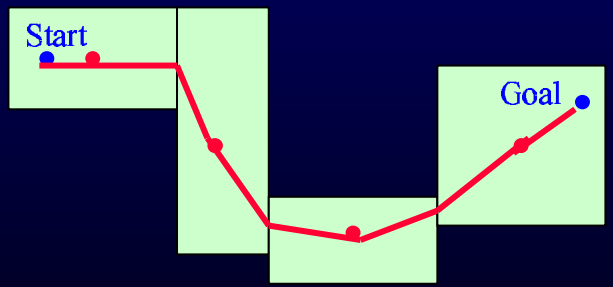
# "Cell decomposition" approaches (C'ed)

- **Searching for a "generic collision-free path"**

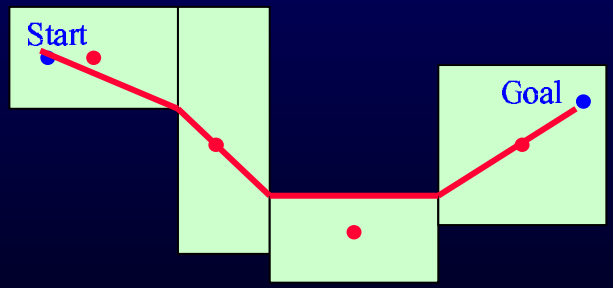



1. *Constructing a graph connecting cells sharing a common frontier (connectivity graph)*
2. *Searching the graph (e.g. using a A* algorithm)*
3. *A "generic collision-free path" = An ordered set of cells {C1, C2 ....Ck} (a "channel")*

- **Constructing a "collision-free path"**




A possible solution (connecting middle points)
*=> More robust according to uncertainty*



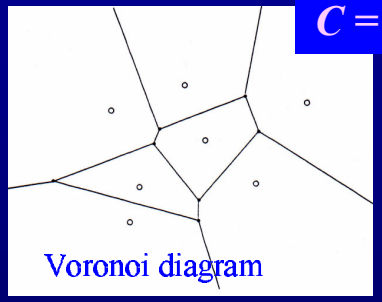

An other solution (connecting closest safe points)
*=> Shorter, but closer to obstacles*

# "Geometric Roadmap"

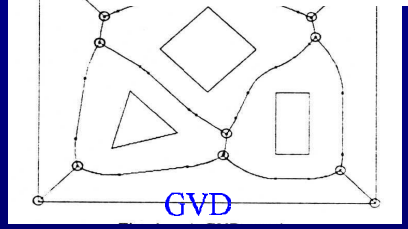
*Constructing a network of "roads" representing the connectivity of $C_{free}$*
*=> Apply only for low dimensional spaces*

- **Retraction of $C_{free}$**

$C = R^2$



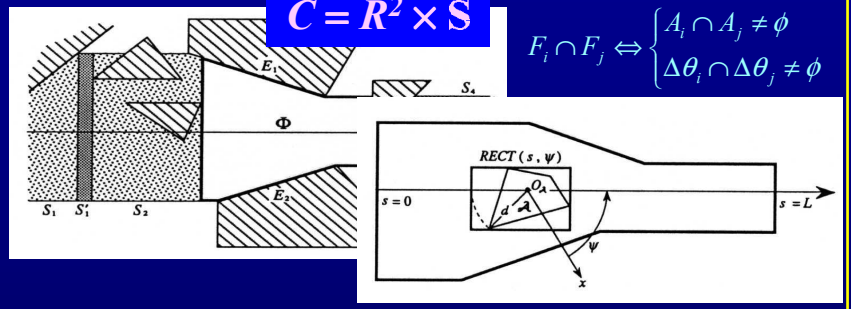

*(vertex, vertex) => median line*
*(edge, edge) => bisector*
*(vertex, edge) => parabolic arc*




Generalized Voronoi Diagram (GVD) *[O'Dunlaing & Yap 82]*

$C = R^2 \times S$

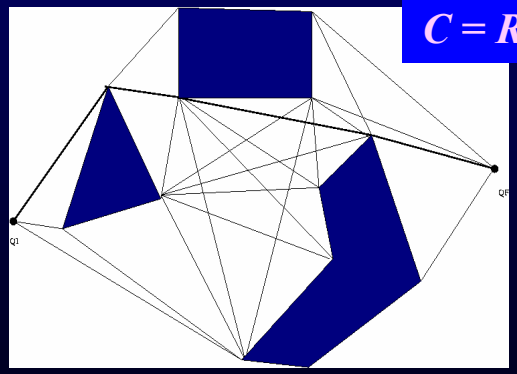
$$F_i \cap F_j \Leftrightarrow \begin{cases} A_i \cap A_j \neq \phi \\ \Delta\theta_i \cap \Delta\theta_j \neq \phi \end{cases}$$



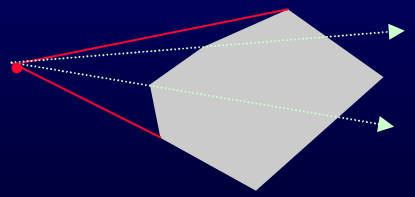

"Freeways" (generalized cylinders) *[Brooks 82]*

- **Visibility graph** *[Nilsson 69]*

$C = R^2$



*Connecting the "visible" vertices of the C-obstacles*



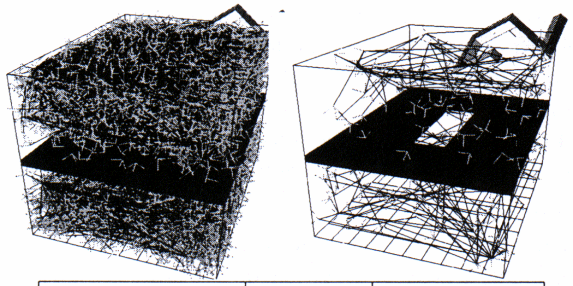
*Pruning the visibility graph*
*(deleting non-tangential edges)*

# *"Probabilistic Roadmaps"*

*Constructing a network representing the connectivity of $C_{free}$*
*=> Apply for complex & high dimensional spaces*

- **Probabilistic Path Planner *(PPP)***

*Simple local planner*
*+*
*Random exploration of $C_{free}$*



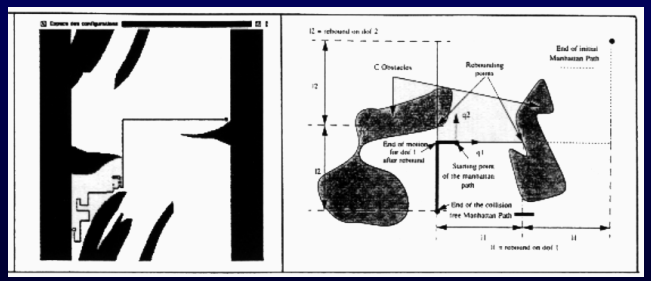
| PRM | Basic | Visibility |
|---|---|---|
| Roadmap size | 4723 | 103 |
| $CS_{free}$ coverage | 99.9% | 99.7% |
| CPU time | 3367 sec | 281 sec |

*Basic PPP*
*[Svestka & Overmars 95]*

*Visibility based PR*
*[Simeon 99]*

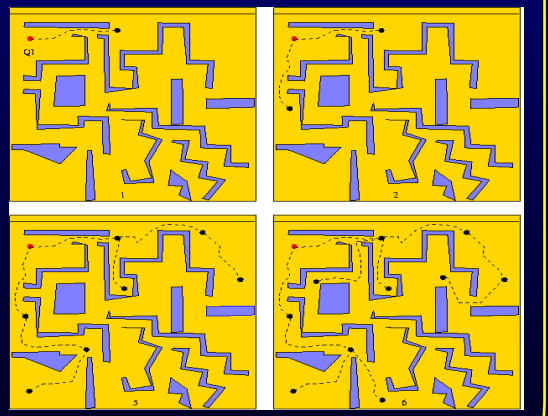
- **Adriane's Clew Algorithm *(ACA)*** [Ahuactzin 94]

*Simple local planner*
*+*
*Random exploration of $C_{free}$*
*+*
*Optimization function*



Search function (local planner)

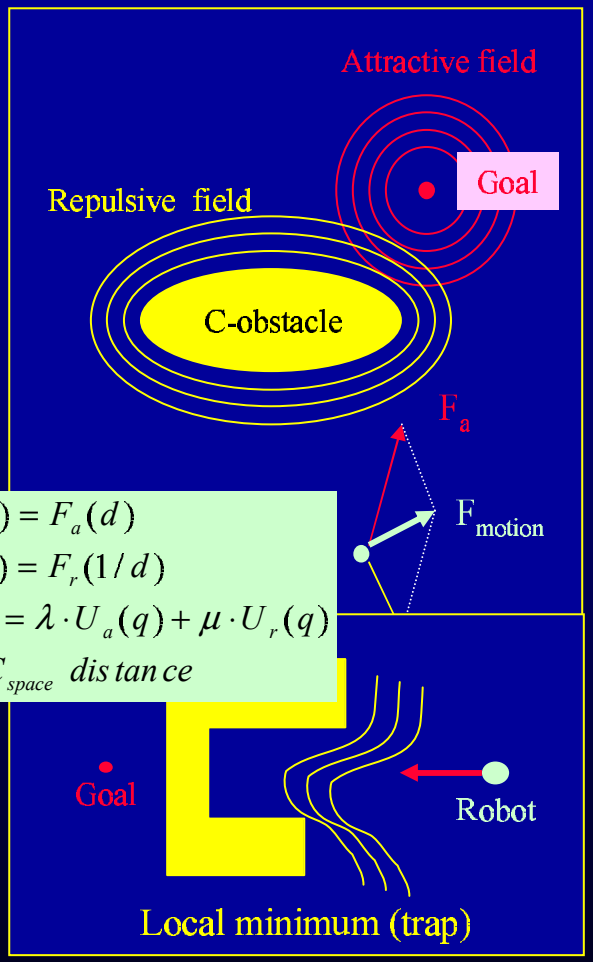
*=> Manhattan paths, Random draws, Rebound technique, Collision checking in W*

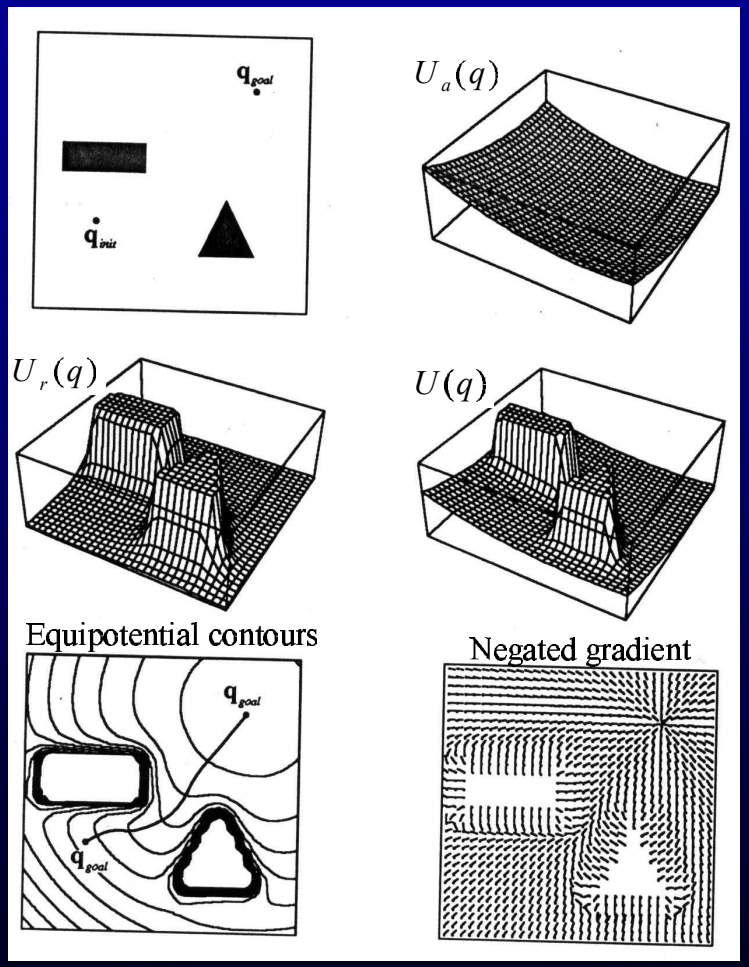


Explore function
*=> Random draws & Optimization algorithm (GE)*

# "Artificial potential fields" (GDM)

*Succession of local decision using a minimization function applied on a local evaluation of $C_{free}$ (done using an underlying energy function)*




*[Khatib 86] [Latombe 90]*

# "Constraint based method" (GDM)

*Minimization of a "task function" under "anti-collision constraints"*

*[Tournassad & Faverjon 88]*

$$\text{Minimizing} \quad \left\| \dot{\tau}(q) - \tilde{\tau} \right\|^2$$
$$\text{under} \quad \delta_{ij} \geq \varepsilon$$

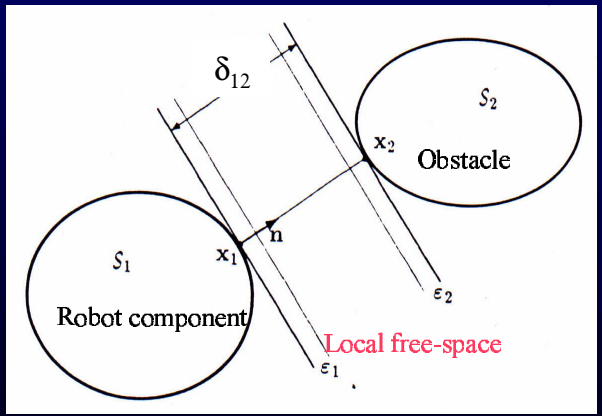
*Making use of derivatives in order to take into account the motion*

- Task function

$$\tau(q) = \left\| q_{goal} - q \right\|$$
$$\dot{\tau}(q) = J_\tau \cdot \dot{q}$$
$$\tilde{\tau} : \text{nominal value of } \dot{\tau}(q)$$
$$J_\tau : \text{task jacobian}$$

- Anti-collision constraint

$$\delta_{ij} \geq \varepsilon \quad \Rightarrow d\delta \geq K \cdot dt$$
$$\text{i.e.} \quad J_{robot} \cdot dq \cdot n \geq K \cdot dt$$
$$\text{or} \quad n \cdot dq \geq K \cdot J_{robot}^{-1} \cdot dt$$




*At each planning step, the "control points" (x1, x2) have to be computed for each pair of "interacting convex components" (e.g. using GJK algorithm)*

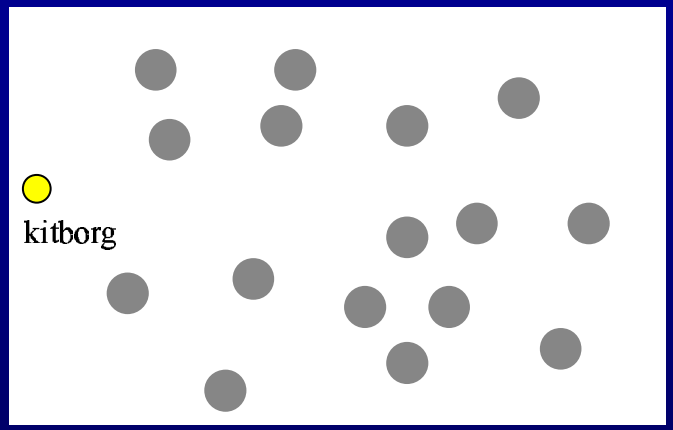
# A simple motion planning example

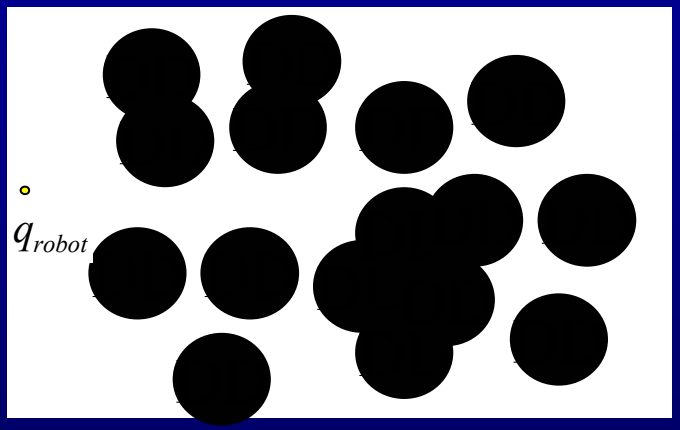
$Robot = disque(O, \rho)$

$Obstacles = disques(O_i, r_i)$

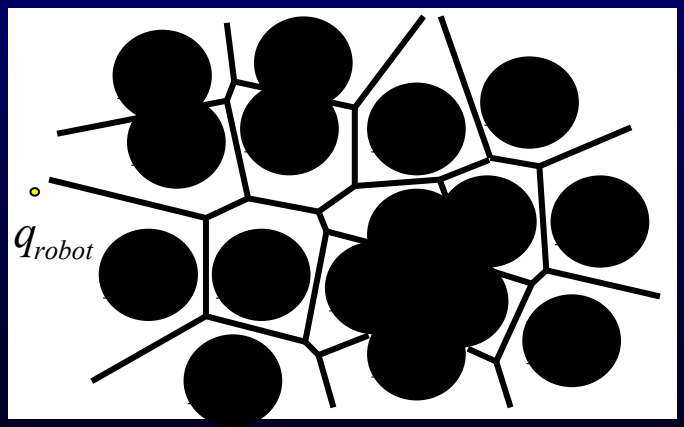
$W = \Re^2, \quad C_{robot} = \Re^2 \times S$

$Symetry + holonomy \Rightarrow C^*_{robot} = \Re^2 = W$

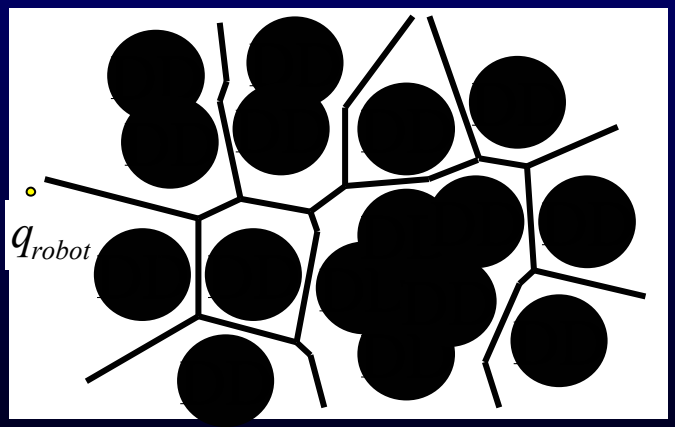



$C - Obstacle = disc(O_i, \rho + r_i)$

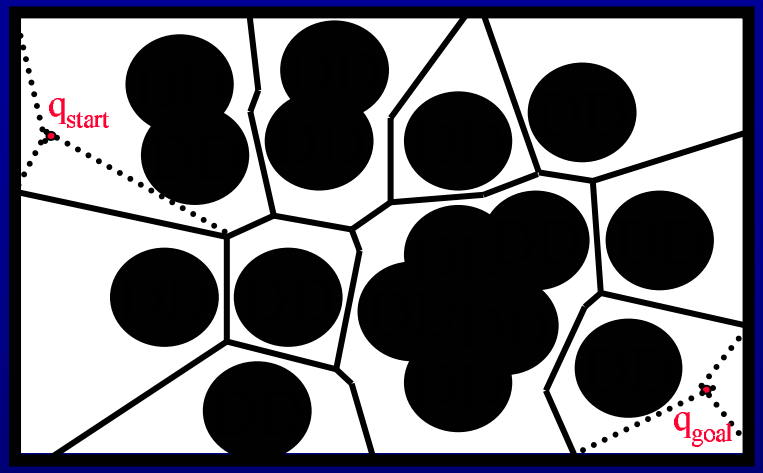



*Voronoi diagram:* $(O_1, O_2 .... O_m)$

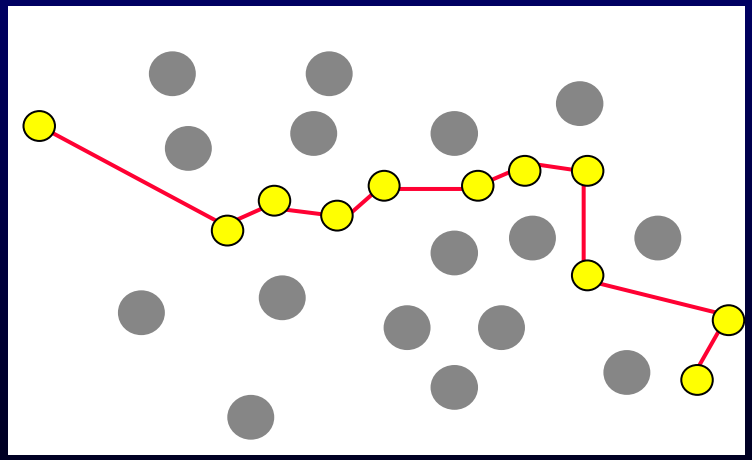


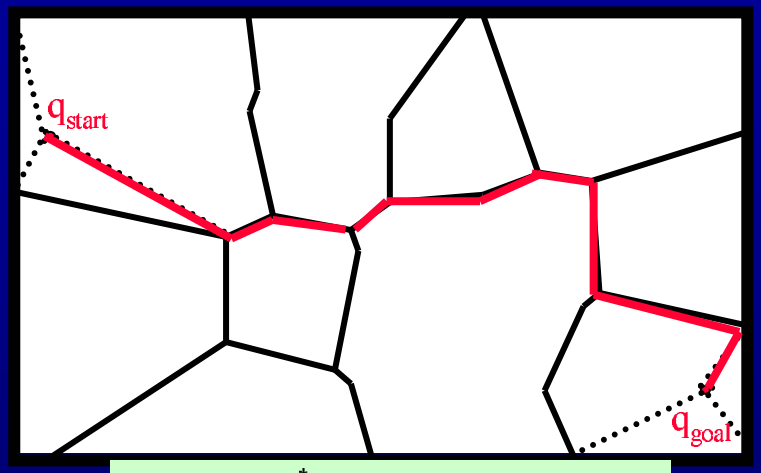

*Connectivity Graph :*
$\Rightarrow Remove\ (edge_{ij} \mid d(O_i, O_j) \le r_i + r_j + 2\rho)$

# A simple motion planning example (C'ed)




*Search Graph :*
$\Rightarrow$ *Adding* $(q_{start}, q_{goal}, edge_{ij}, boundaries)$






*Search :* $A^*$ *algorithm*

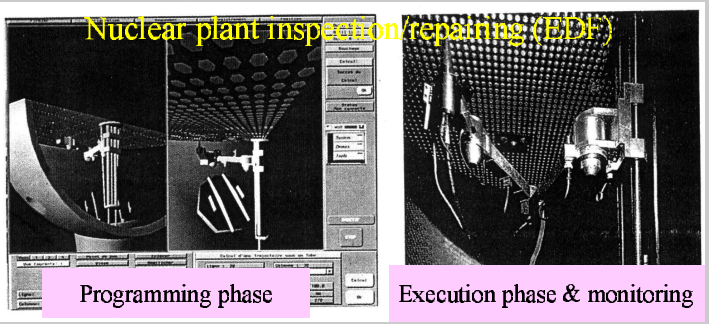
$f(s) = g(s) + h(s)$

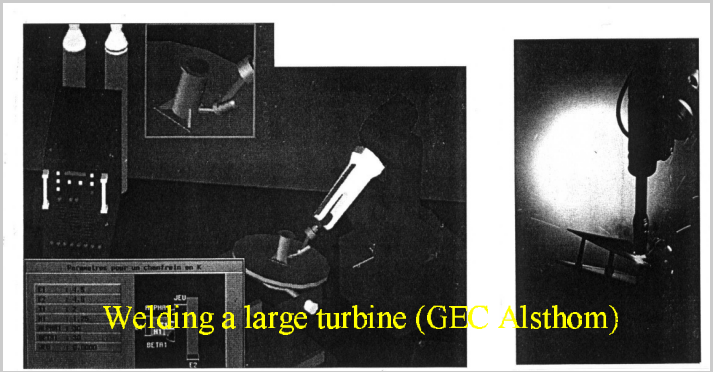
$g(s)$: *lenght* $(q_{start}, s)$

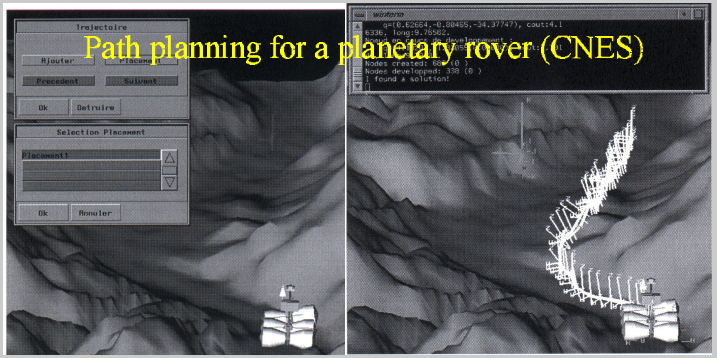
$h(s)$: *lenght – assessment* $(s, q_{goal})$

*A possible solution including*
*10 straight motions & 11 re-orientations*

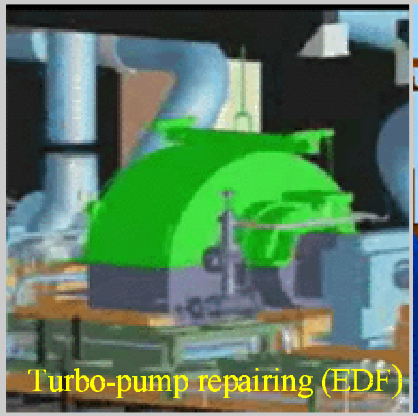
# Some industrial applications

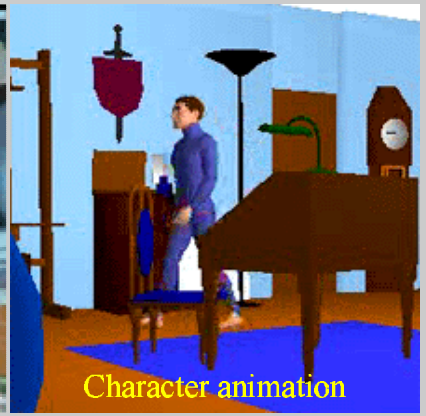


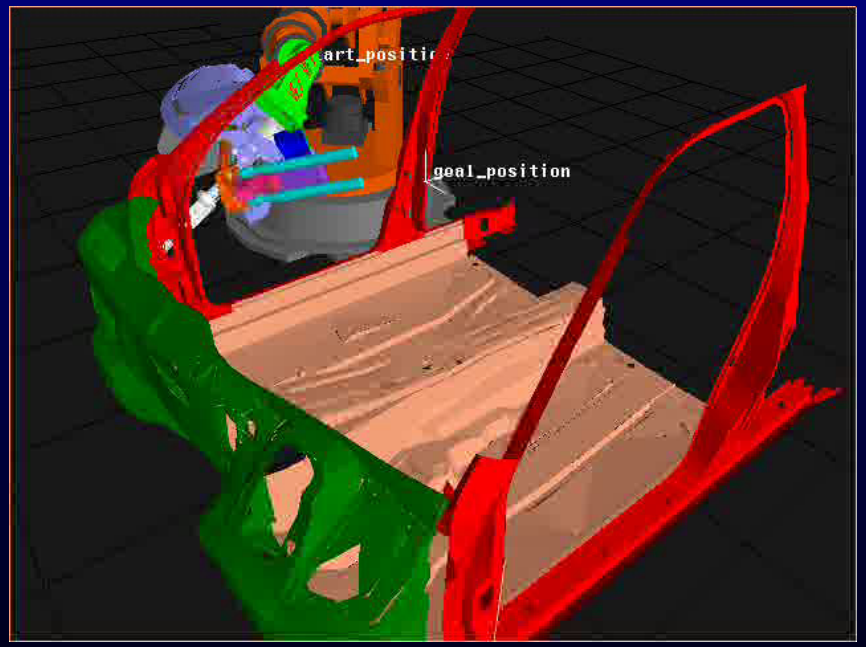




ACT system (Aleph Technologies)




Move3D system (Kineo)

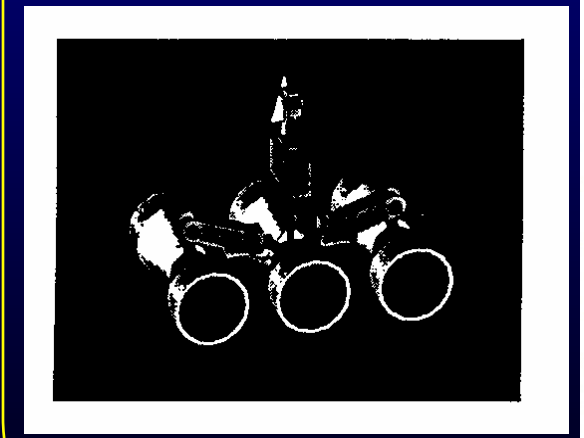


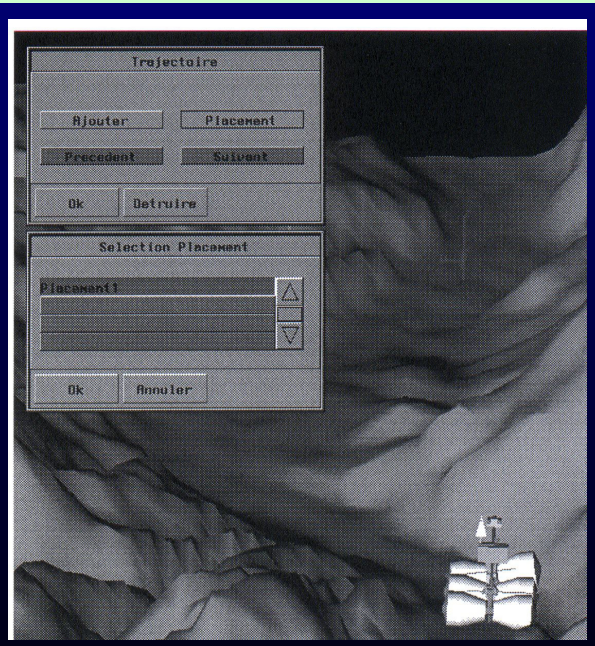

Assembly task for car constructor (IGRIP, Delmia)

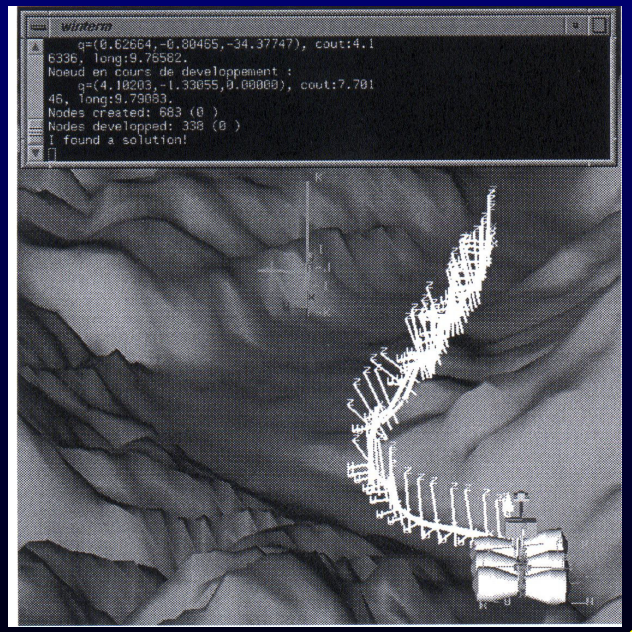
# *Motion Planning for an articulated Rover*

- **3 independent axles with 2 conical wheels**
- **The front & back axles can rotate along a longitudinal axis**
- **The chassis is made of 2 articulated arms :**
  - *linking the front and back axles to the central one*
  - *rotating around the central axle*
  - *articulated in order to modify the gap between the front and back axles*
- **The joints: - 6 velocity controls** *(wheels)*
  - **2 position controls** *(front & back axles gaps)*
  - **3 passive joints** *(vehicle/terrain "shape" adaptation)*

  *=> modeled using Spring-Damper connectors*





# Motion Planning for an articulated Rover (C'ed)

```
p_1= Creer_Point (0.730688, -1.026698, -34.377465);
p_2= Creer_Point (0.903161, -1.121580, -22.918309);
p_3= Creer_Point (1.093741, -1.180915, -11.459155);
p_4= Creer_Point (1.290266, -1.200995, 0.000002);
p_5= Creer_Point (1.486180, -1.220451, -11.459155);
p_6= Creer_Point (1.677126, -1.239262, 0.000002);
p_7= Creer_Point (1.868955, -1.259003, -11.459155);
p_8= Creer_Point (2.064329, -1.278596, 0.000002);
p_9= Creer_Point (2.261734, -1.298304, -11.459155);
p_10= Creer_Point (2.459179, -1.318172, 0.000002);
p_11= Creer_Point (2.656802, -1.337920, -11.459155);
p_12= Creer_Point (2.855045, -1.358103, 0.000002);
```

*Safe trajectory = $\{CP_1\ CP_2\ \ldots CP_n\}$*
*A sequence of safe "control points"*

```
Exec_Traj (trajectoire_0, ligne_droite);
Exec_Traj (trajectoire_1, giration);
Exec_Traj (trajectoire_2, ligne_droite);
Exec_Traj (trajectoire_3, giration);
Exec_Traj (trajectoire_4, ligne_droite);
Exec_Traj (trajectoire_5, giration);
Exec_Traj (trajectoire_6, pente_montante);
Exec_Traj (trajectoire_7, ligne_droite);
Exec_Traj (trajectoire_8, giration);
Exec_Traj (trajectoire_9, ligne_droite);
```

*Safe locomotion plan = $\{CM_1\ CM_2\ \ldots CM_{n-1}\}$*
*A sequence of "control modes" for moving between $CP_i$*

```
ligne_droite= Creer_Mode("ligne_droite");
Add_Critere(ligne_droite, "avant_moyen", 0.9, 1.0);
Add_Critere(ligne_droite, "arriere_moyen", 0.9, 1.0);
Add_Terminaison(ligne_droite, TRUE, 0.8, "pente_montante",
                "tangage_montee");
Add_Terminaison(ligne_droite, TRUE, 0.8, "pente_descendante",
                "tangage_descente");
Add_Alarme(ligne_droite, TRUE,0.2,"GARDE AU SOL", "garde_sol");
Add_Alarme(ligne_droite, FALSE,0.2,"ROULIS TROP IMPORTANT", "roulis");
Add_Stabilite_params(ligne_droite, 4, 1.0, 1.0);
Add_Vitesse_params(ligne_droite, 1.0, 1.0);
```

*Example of a "control mode" ("go straight")*
*Constructed using "Pre, Post, and Failure" conditions*

# - Part II -
# Dealing with real world constraints

*How to take into account*
*Non-holonomic kinematic constraints ?*

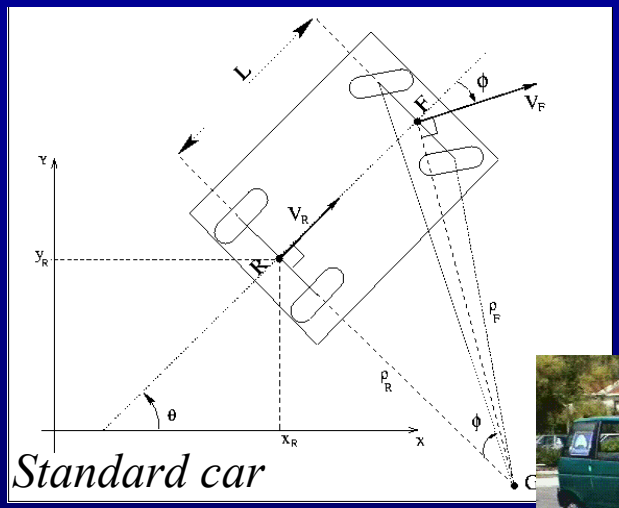
*How to process the dynamics of both*
*the robot and its environment ?*

*How to deal with uncertainty & hazards*
*of the physical world ?*

# Non-holonomic kinematic constraints

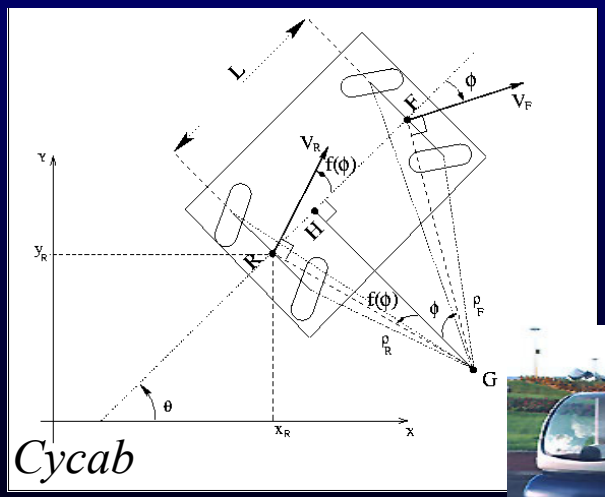
*A NH system, is a system having less control variables than configuration parameters*

$$q = (x, y, \theta, \phi)$$
$$u = (u_1, u_2)$$




Standard car

$$\begin{pmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \\ \frac{\tan(\phi)}{L} \\ 0 \end{pmatrix} u_1 + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} u_2 \qquad \textit{with:} \quad (u_1 = V_R \quad u_2 = \dot{\phi})$$




Cycab

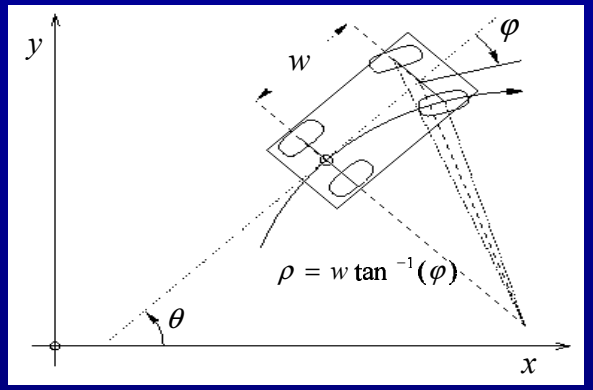
*Smaller turning radius bound & sweeping volume*
*=> Better maneuverability in cluttered environments*
*... But Planning & Control much more difficult !*

$$\begin{pmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} \cos(\theta + f(\phi)) \\ \sin(\theta + f(\phi)) \\ \frac{\sin(\phi - f(\phi))}{L\cos(\phi)} \\ 0 \end{pmatrix} u_1 + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} u_2$$

$(u_1 = V_R \quad u_2 = \dot{\phi})$

$f(\phi)$: *characteristic function*

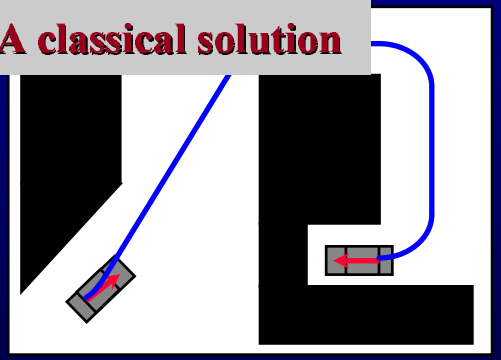
# Non-Holonomic Path Planning problem




*Perfect rolling & Limited steering*

$$q = (x, y, \theta)$$
$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0$$
$$\rho > \rho_{\min}$$

**A classical solution**

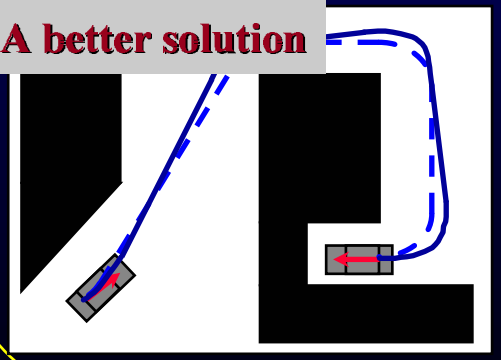


***Shortest paths*** = *line segments + tangential circular arcs*

*[Dubins 57] [Reeds & Shepp 90] [Laumond 86]*

***=> Executable only if the vehicle stops at each turn !!!***

**A better solution**



***Continuous Curvature Paths (CC-paths)*** =

*CC-curves + Bounded curvature & curvature derivative*

- *No obstacle: [Ijima & al. 81] [Nelson 89] [Liscano & Green 89]*
- *With obstacle avoidance: [Scheuer 97] [Scheuer & Laugier 98]*

# A key concept : Differential Flatness

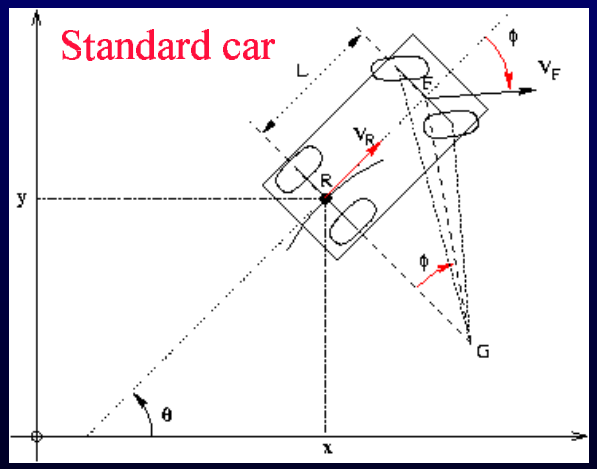
*[Fliess et al. 93]*

A system is "differentially flat" if there exist *linearizing outputs* $\mathbf{y} = (y_1, \ldots, y_m)$ differentially independent such that:

- $y = h(x, u_1 \ldots, u_1^{(\beta_1)} \ldots u_m, \ldots, u_m^{(\beta_m)})$ where $x = (x_1, \ldots, x_n)$, $u = (u_1, \ldots, u_m)$

*and*

- $x = A(y_1, \ldots y_1^{(\alpha_1)} \ldots y_m, \ldots, y_m^{(\alpha_m)})$
- $u = B(y_1, \ldots, y_1^{(\alpha_1+1)} \ldots y_m, \ldots, y_m^{(\alpha_m+1)})$




Kinematic model

$$\dot{X} = \begin{pmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \\ \frac{\tan(\phi)}{L} \\ 0 \end{pmatrix} u_1 + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} u_2$$

Differential flatness

$X = (x_R, y_R, \theta, \phi)$ *and* $u = (V_R, \dot{\phi})$

$Y = (x_R, y_R)$

$1st\ derivative : \dot{Y} \Rightarrow (\cos\theta, \sin\theta) \Rightarrow \theta$

$2nd\ derivative : \dot{\theta} \Rightarrow \frac{\tan\phi}{L} \Rightarrow \phi$

$$\kappa = \frac{\tan(\phi)}{L} \quad \forall \phi \in ]-\tfrac{\pi}{2}, \tfrac{\pi}{2}[$$

# Differential Flatness of the Cycab

- **Flatness property for the Cycab** *[ Sekhavat & Hermosillo 00]*

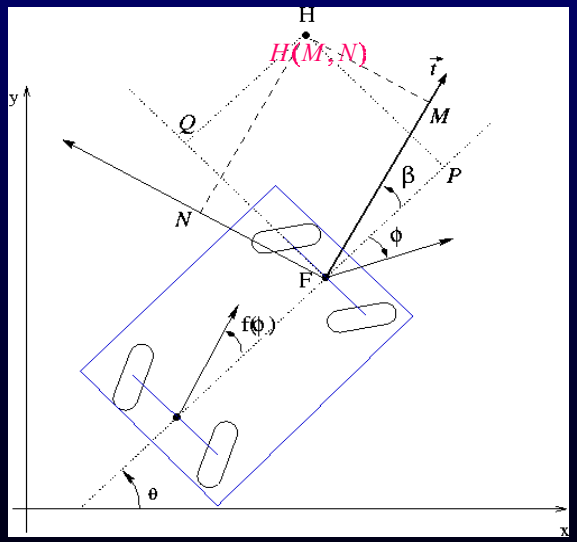
$$X = (x_R, y_R, \theta, \phi) \quad and \quad u = (V_R, \phi)$$

$$Cycab: \quad f(\phi) = k \cdot \phi \quad , \forall k \neq 1$$

$$Flat\ outputs: \quad H = (y_1(\phi), y_2(\phi))$$

$$\Rightarrow \quad \kappa = \frac{\sin((1+k)\phi_0)}{L \cos\phi_0 \cos(k\phi_0)} \quad if\ \phi \in \{0, -\phi_0, \phi_0\}$$

- **Flat outputs for the Cycab** *[ Sekhavat & Hermosillo 01]*




$$Turning\ frame: (F, \vec{t}, \vec{t}^{\perp}) \quad with \quad \beta(\phi) = \tan^{-1}\frac{B(\phi)}{A(\phi)}$$

$$\vec{t} = \cos\phi\, f'(\phi)\, \vec{u}_{\theta+\phi} - \cos f(\phi)\, \vec{u}_{\theta+f(\phi)}$$

$$A(\phi) = \cos^2(\phi) f'(\phi) - \cos^2 f(\phi)$$

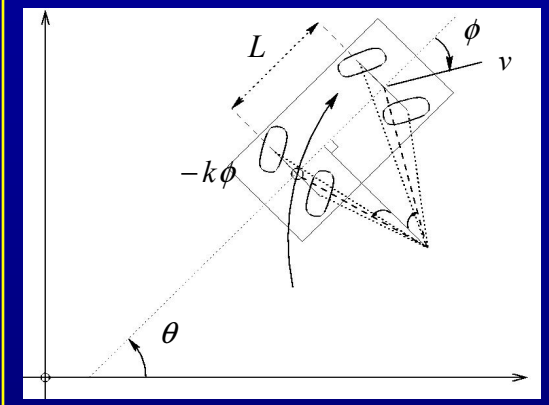
$$B(\phi) = \cos(\phi)\sin(\phi) f'(\phi) - \cos(f(\phi))\sin(f(\phi))$$

$$M(\phi) = \frac{L\cos^2(f(\phi))}{\sqrt{A^2(\phi) + B^2(\phi)}}$$

$$N(\phi) = -\int_0^{\phi} \frac{L\cos^2(f(u))(B'(u)A(u) - A'(u)B(u))}{\left(A^2(u) + B^2(u)\right)^{3/2}}$$

# Motion Planning using the Flat Outputs

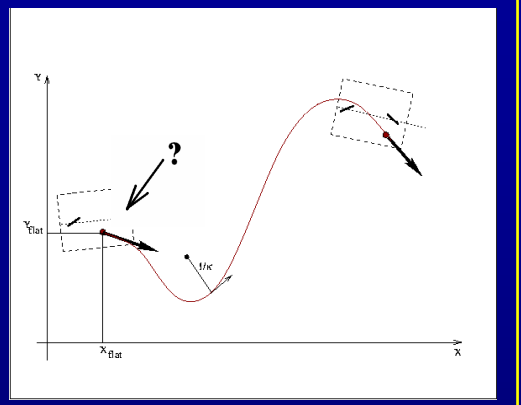
Configuration space




$(x, y, \theta, \phi)$ $\qquad$ $y(t)$

*$(y_1 \ldots y_m)$ differentially independant*
*=> any smooth curve is an admissible path*

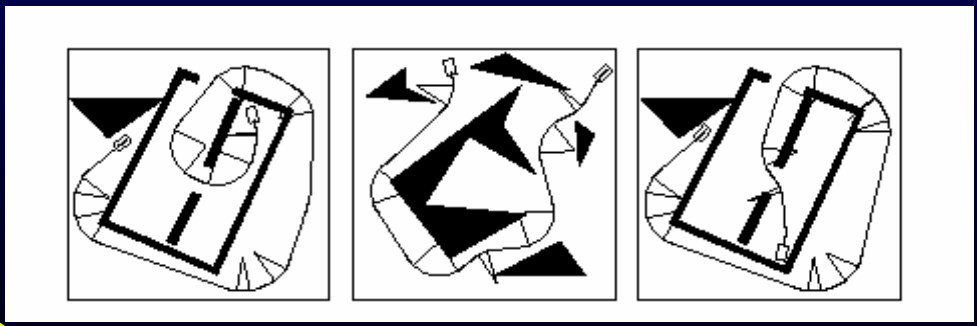
Equivalent "flat" space




Motion Planner : Geometric Planner *(Collision free path)* + Steering Method *(Feasible sub-paths)*

## Some implemented solutions

- Global Geometric Planner : *ACA [Ahuactzin 94]* or *PPP [Svestka & Overmars 98] ....*
- Steering Method : *Shortest paths [Dubins 57] [Reeds & Shepp 90]* or *CC-paths [Nelson 89] [Scheuer 97] ...*



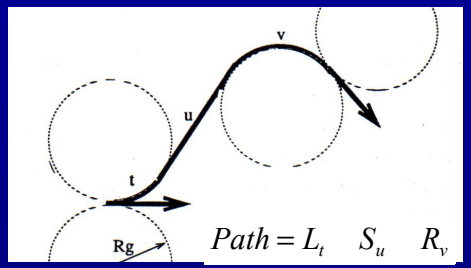
- *LAAS Approach :*
  *PPP + Shortest paths [Laumond 86]*
- *INRIA Approach :*
  *ACA + CC-Path ("extended Dubins")*
  *+ Appropriate collision checker*
  *[Scheuer & Laugier 98]*

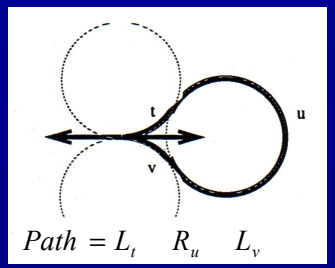


INRIA RHÔNE-ALPES

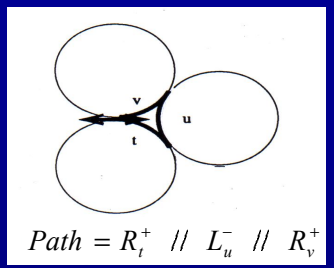
# Main steering methods

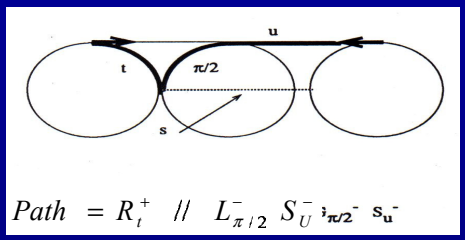
- **Shortest paths** *(Straight lines + Circular arcs)*



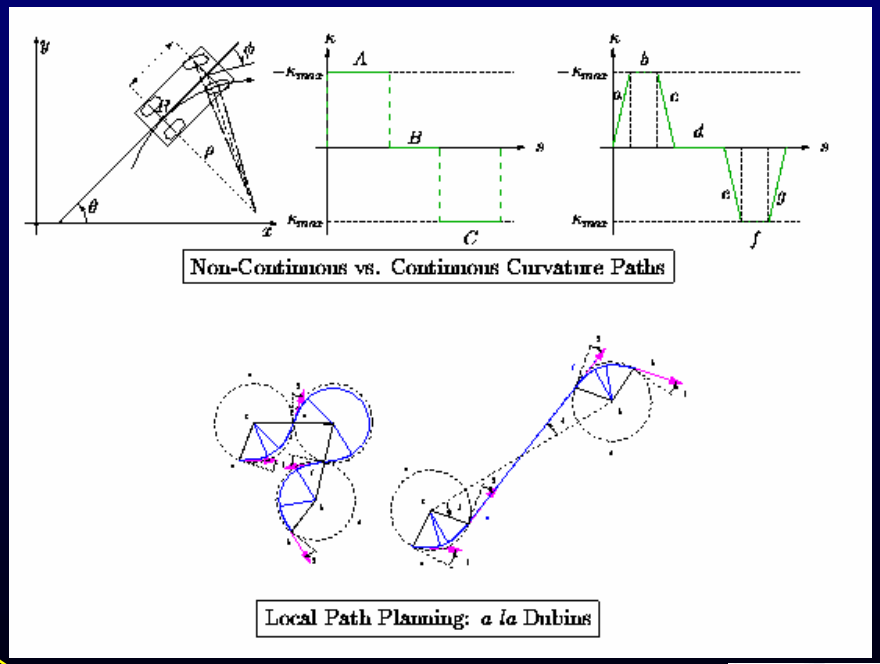

*Shortest paths without manoeuvre [Dubins 57]*



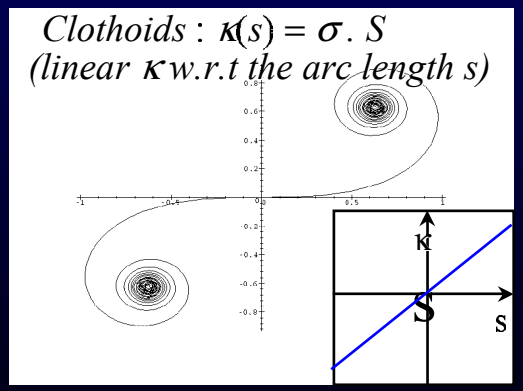




*Shortest paths with manoeuvres [Reeds & Shepp 90]*

- **CC-paths** *(CC-curves +Bounded curvature & curvature derivative)*




*A possible solution:*
*Straight lines + Circular arcs + Arcs of clothoids*
*[Nelson 89] [Scheuer 97]*



# *An example : Planning & Executing parking maneuvers for the Cycab*

# - Part II -
# Dealing with real world constraints

*How to take into account*
*Non-holonomic kinematic constraints ?*

***How to process the dynamics of both the robot and its environment ?***

*How to deal with uncertainty & hazards*
*of the physical world ?*

# How to deal with dynamic constraints ?

| Path Planning | Trajectory Planning |
|---|---|
| *Geometry* | *Time* |
| *(1) Stationary Obstacles* | *Moving Obstacles (3)* |
| *(2) Kinematic Constraints* | *Dynamic Constraints (4)* |

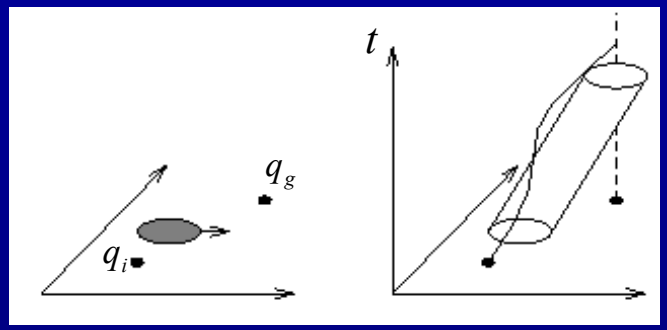
- **Objective** :

  *To plan motions for a robot subjected to kinematic & dynamic constraints (2 + 4) in a dynamic workspace (1 + 3), e.g. a car-like vehicle on the road network*

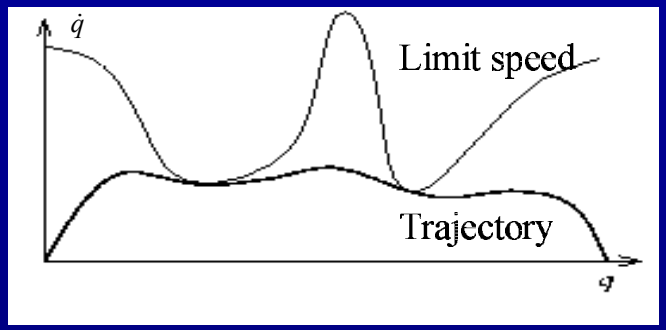
Admissible trajectory = Collision-free *(at some time)*
+ Feasible *(i.e. meets kinematic & dynamic constraints)*

- **Approaches** :
  - Off-line planning (a few known moving obstacles)
    - => *Adding time & states constraints (Kinodynamic Motion Planning [Canny al. 88])*
  - On-line planning (known & sensed moving obstacles, arbitrary number)
    - => *Deforming trajectories (Elastic strips [Khatib & Brock 99 ])*
    - => *Selecting safe controls (Global Dynamic Windows [Khatib & Brock 00] , Velocity obstacles [Fiorini 95] [Large& Shiller 00] [Large et al. 03])*

# Kinodynamic Motion Planning (KMP)

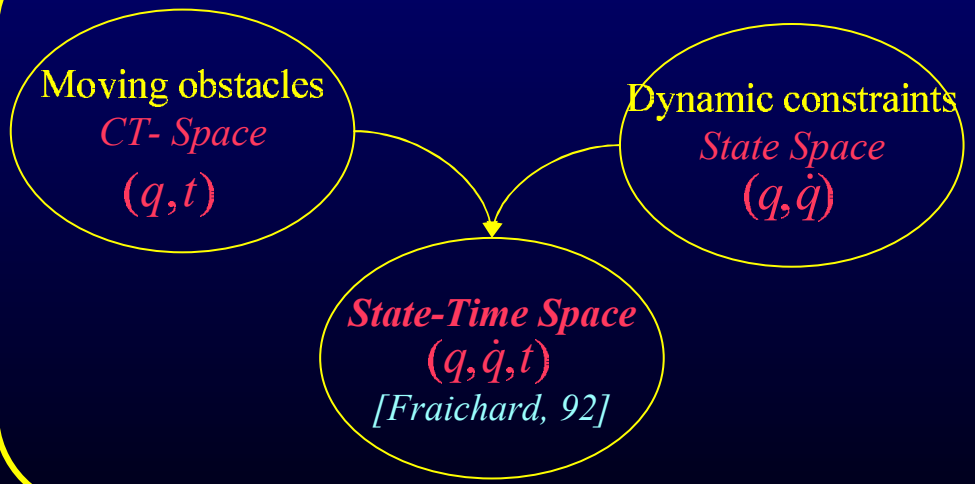



Moving obstacles : "Configuration-Time Space"
*[Erdmann & Lozano-Perez 86]*
*[Fujimura & Samet 89-90] [Shih et al. 90]*




Dynamic constraints : "State Space"
*[Bobrow et al. 85] [Jacobs et al. 89]*
*[Shiller & Shen 90] [Xavier 92]*

## *KMP : The State-Time Space approach (ST-Space)*

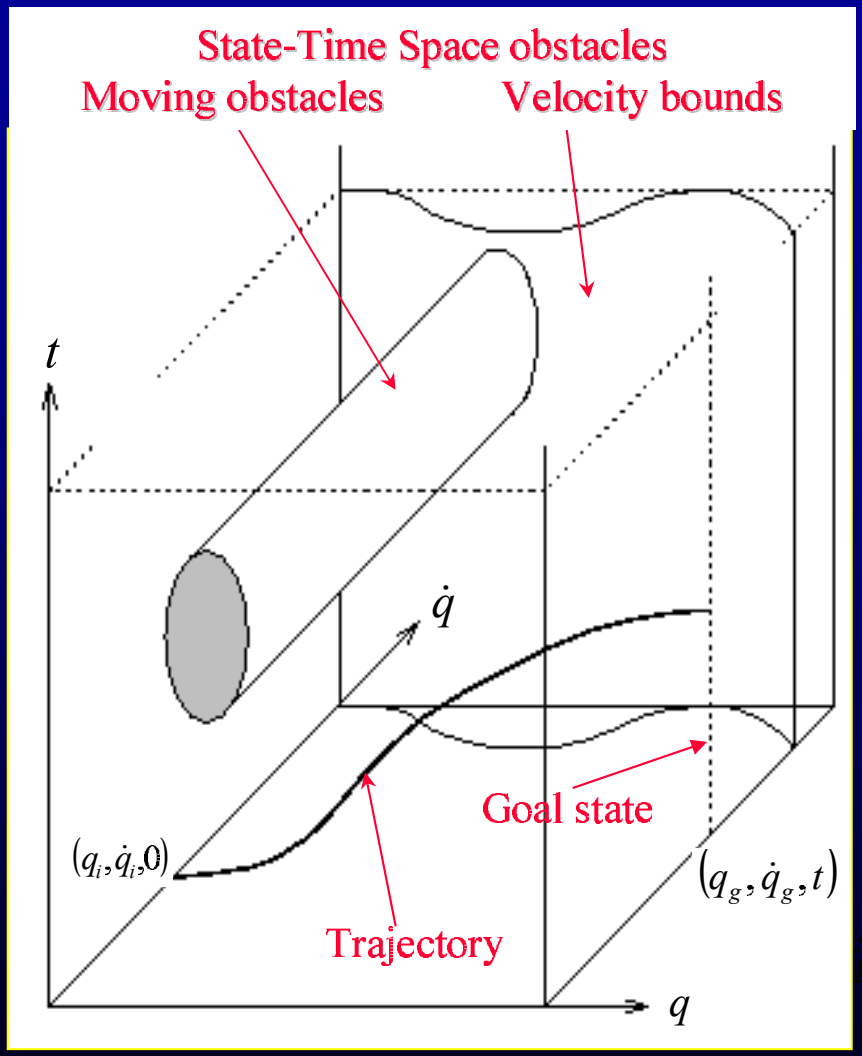



1. Unified modelling of different types of constraints (obstacles & velocities)
   => *ST-Space obstacles*

2. Trajectory = *Curve in ST-Space*
   => *Avoids ST-Space obstacles*
   => *Verifies additional shape properties* $(\ddot{q},t)$

# ST-Space : The Car-like robot case

*[Fraichard & Laugier 92 & 93]*




- Velocity bounds + Moving obstacles
  => *ST-Space obstacles (1)*

- Acceleration bounds + Time constraint
  => *ST-Space curve shape constraints (2)*

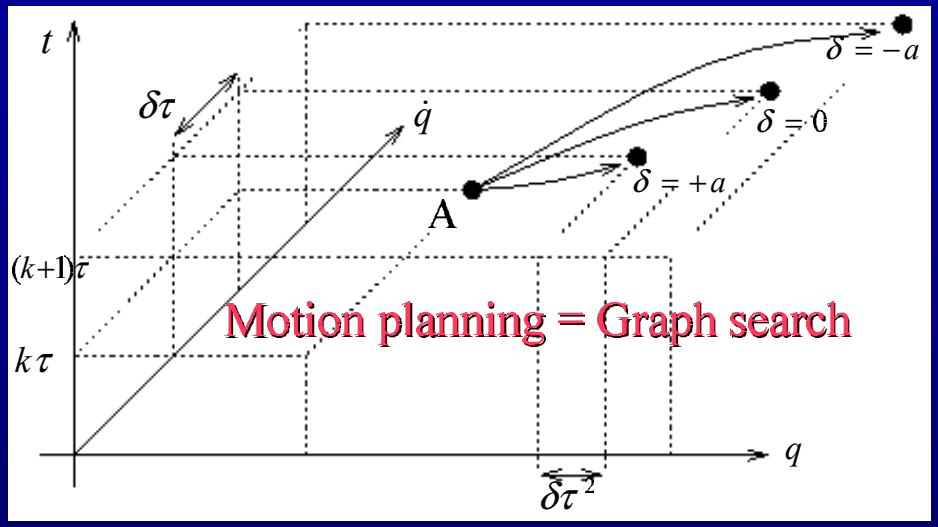
$$\begin{cases} F_{min} \leq F \leq F_{max} & \text{(engine force limit)} \\ \sqrt{R_t^2 + R_n^2} \leq \mu R_b & \text{(no sliding constraint)} \\ 0 \leq \dot{s} \leq \dot{s}_{max} & \text{(speed limit)} \end{cases}$$

$$\Rightarrow \begin{cases} 0 \leq \dot{s} \leq \min(\dot{s}_{max}, \sqrt{\frac{\mu g}{\kappa_s}}) & \text{Velocity bounds} \\ \max(\frac{F_{min}}{m}, \sqrt{\mu^2 g^2 - \kappa_s^2 \dot{s}^4}) \leq \ddot{s} \leq \min(\frac{F_{max}}{m}, \sqrt{\mu^2 g^2 - \kappa_s^2 \dot{s}^4}) & \text{Acceleration bounds} \end{cases}$$

Feasible Trajectory =
*Curve in ST-Space from* $(q_i, \dot{q}_i, 0)$ *to* $(q_g, \dot{q}_g, t)$
*and verifying (1) and (2)*

# Motion planning in the ST-Space (using KMP)

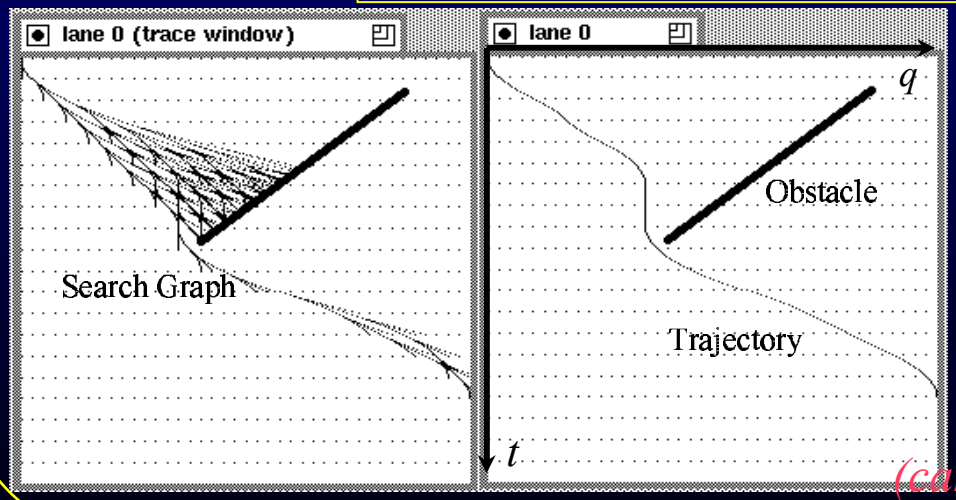
*[Fraichard & Laugier 92 & 93]*




- Canonical Trajectories :
  - Piecewise constant acceleration *(finite & discrete set)*
  - Acceleration step $\delta$, Time-step $\tau$

  *=> A graph embedded in ST-Space*
- An edge is valid iff :
  - *It avoids ST-Obstacles*
  - *It meets acceleration constraints*

**Result for a 1D workspace (along a given path)**




- Moving obstacles:

  *=> A priori knowledge / prediction on a given "time horizon" $T_{max}$*
- Running time: $P(\tau, \delta, T_{max})$

  *=> High complexity !!*

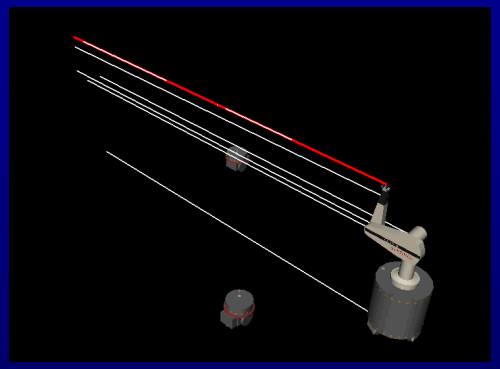
  *(cannot be applied in most of practical cases)*

# *Weakly dynamic workspace: Deforming trajectories*

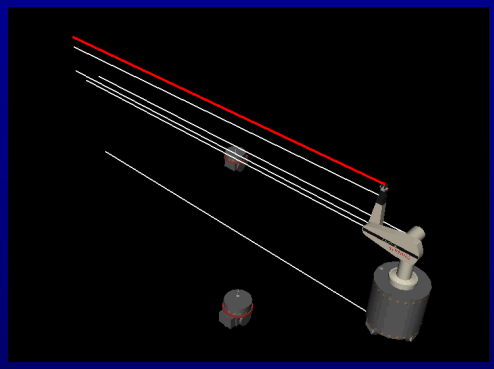
*Deform a nominal trajectory under the effects of a "repulsive potential field"*

- **Reactive trajectory deformation** *(Elastic strips [Khatib & Brock 99 ])*

*"Zero order" deformation (path level)*

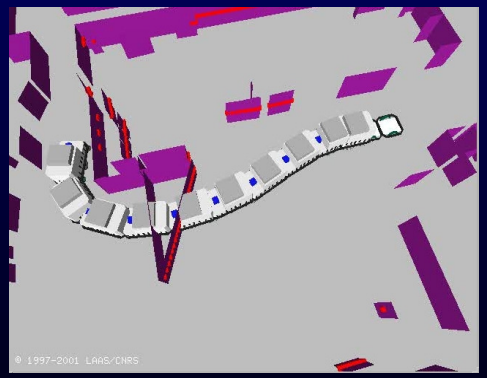


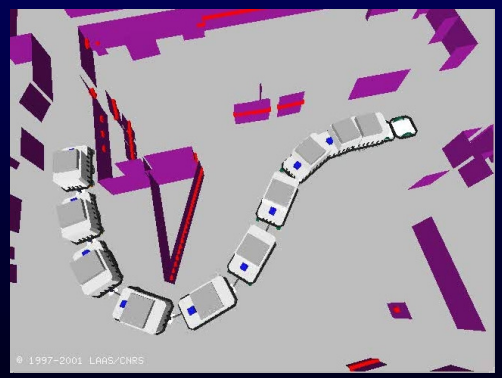
*No task constraint*



*Task constraints to satisfy*

- **Dealing with NH kinematic constraints** *[Lamiraux & Bonnafous 03 ]*





*"First order" deformation (perturbation of the control)*

# Sketch of the « second order method »

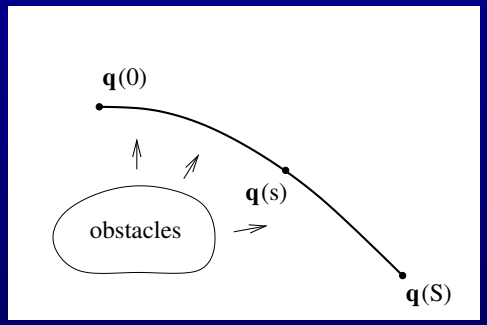
*[Lamiraux & Bonnafous 03 ]*

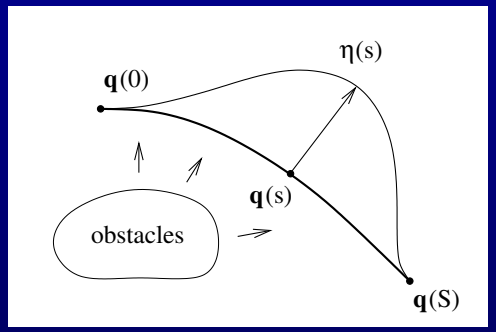
*1- Define a repulsive potential field* $u(q) = \frac{1}{d_{\min}(q)}$

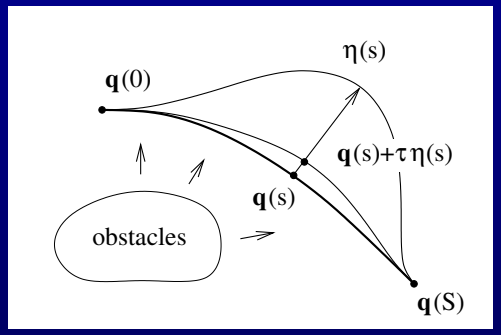
*2- Integrate this potential field along the trajectory* $U(path) = \int_0^s u(q(s)).ds$

*3- Compute a direction of deformation* $\eta(s)$ *that satisfies NH constraints & decrease U(path)*

*4- Apply this deformation scaled by a small real number* $\tau$*, until the collision desappears*



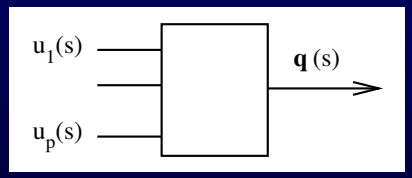



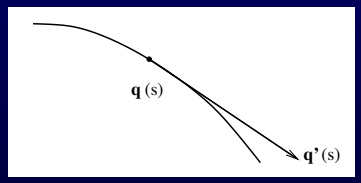




=> Trajectory deformation is obtained by perturbing the input functions (controls) *u(s)*

*Controls characterizing s :* $u(s) = (u_1(s), u_2(s) \ldots u_p(s))$

$$\dot{q}(s) = \sum_{i=1} u_i(s).X_i.q(s))$$

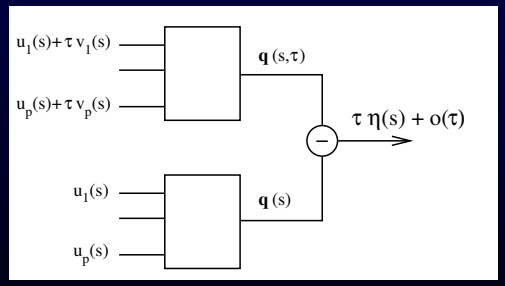


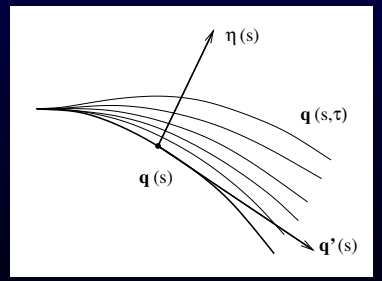




*Input perturbations :*

$v(s) = (v_1(s), v_2(s) \ldots v_p(s))$

$\Rightarrow u(s,\tau) = u(s) + \tau.v(s)$



# Dynamic workspace : Selecting safe controls

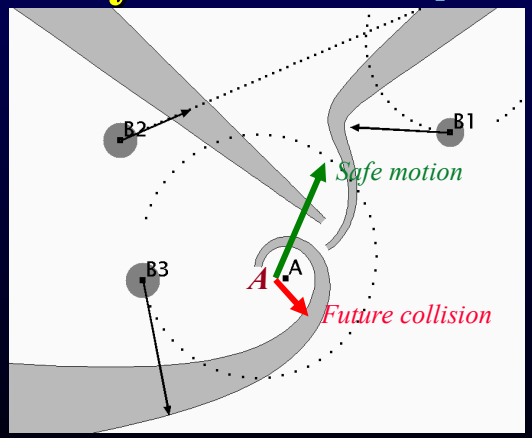
*=> On-line avoidance of obstacles moving along arbitrary trajectories (known or sensed)*
*=> The traditional state-time approach (zero order search) is not tractable (complexity & real-time) ... Instead, reason at the "velocity level" (first order search) !*

- **Global Dynamic Windows** *[Khatib & Brock 00]*

*• Generating on the fly goal-directed motions*
*=> Sequence of safe controls*

*• Alternating reconstruction & planning phases*
*=> Low dynamicity is allowed*

- **Velocity Obstacles** *[Fiorini 95][Large & Shiller 00][Large et al. 03]*

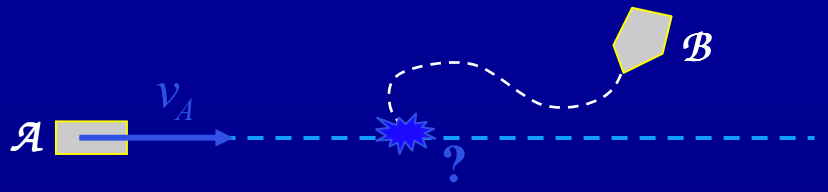



*• Real-time computation of V-Obstacles (velocity space)*
*=> Instantaneous colliding velocities*

*• Strategies for navigating among any moving obstacles*
*=> Obstacle avoidance & Iterative trajectory planning*

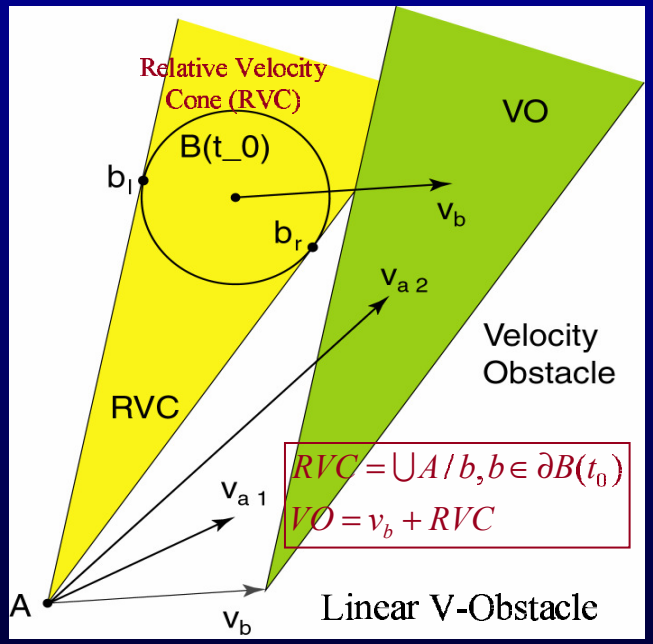
# "Velocity Obstacle" principle

*[Fiorini 95][Large & Shiller 00]*



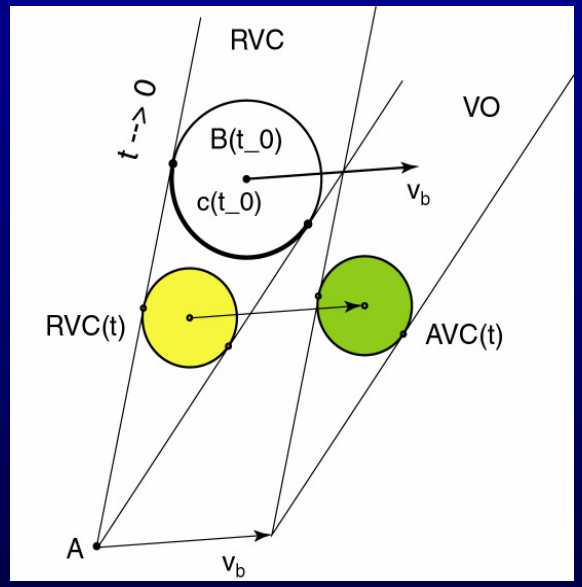

$$VO = \left\{ \overrightarrow{v_A} \in \mathcal{V} \,\middle|\, \exists t \in [t_0\ Th],\ \mathcal{A}(t) \cap \mathcal{B}(t) \neq \phi \right\}$$

with $\mathcal{A}(t) = \mathcal{A}(t_0) + \overrightarrow{v_A} \cdot t$ ; $\mathcal{B}(t)$ known on $[t_0\ Th]$



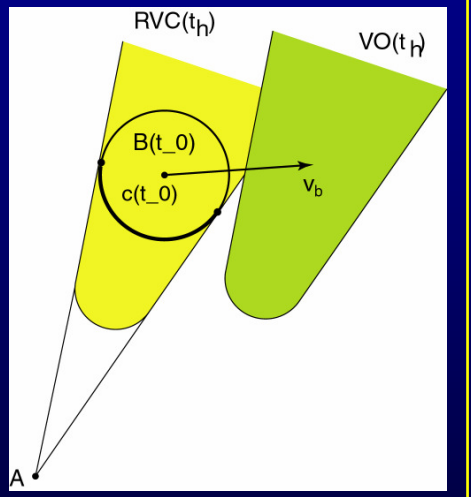

$$RVC = \bigcup A / b, b \in \partial B(t_0)$$
$$VO = v_b + RVC$$

- *Any absolute velocity of A, pointing inside VO, would result in collision at some time* $t \in [0, \infty]$
- *A grazes B at tangency points between RVC and B(t0)*




*AVC(t) : Velocities resulting in a collision at time t*

$$RVC(t) = \frac{B(t_0)}{t - t_0}$$
$$AVC(t) = v_b + RVC(t)$$

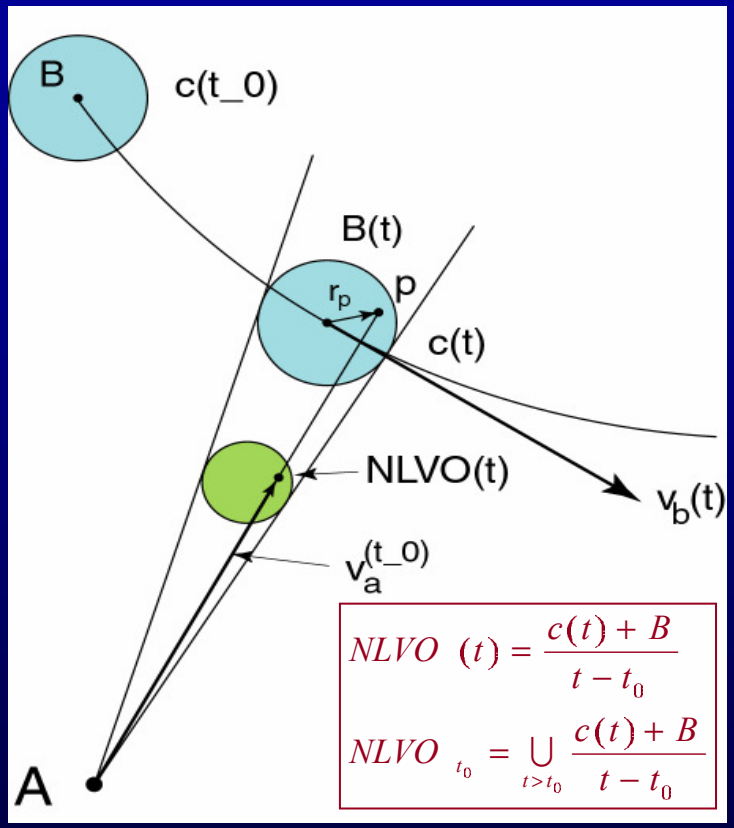



*Adding a time horizon* $t_h$

$$VO(t_1) = v_b + \bigcup_{t>t_1} RVC(t)$$

# Non-linear V-Obstacles

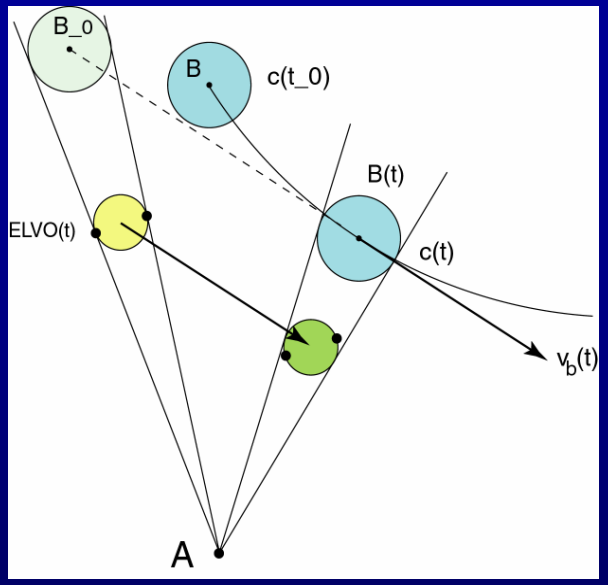
*[Large & Shiller 00] [Large et al. 03]*




$$NLVO\ (t) = \frac{c(t) + B}{t - t_0}$$

$$NLVO_{t_0} = \bigcup_{t > t_0} \frac{c(t) + B}{t - t_0}$$

- *Obstacle B moves along trajectory c(t)*
- *NLVO(t) = Absolute velocities of A at t0 that would collide with B(t)*
- *NLVO is the union of all NLVO(t) for t> t0*




- *Tangency points form the boundary of NLVO*
- *These points are determined from ELVO(t)*

*=> Approximate boundaries of NLVO (On-line computation ! )*

*Obstacle trajectory* : $c(t) = d(t)e^{i\theta(t)}$

$$c_v(t) = \frac{d(t)}{t} e^{i\theta(t)}$$

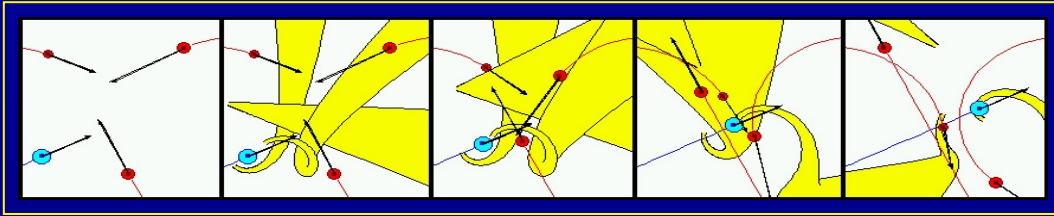
$$vo_r(t) = c_v(t) + i\frac{r}{t}\hat{c}_l(t)$$

$$vo_l(t) = c_v(t) - i\frac{r}{t}\hat{c}_l(t)$$

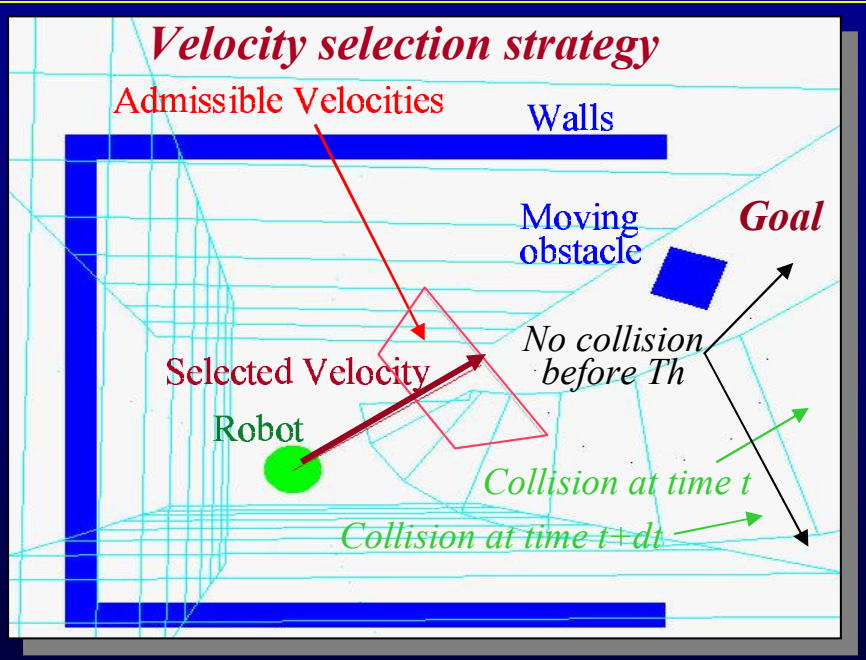
# Safe navigation using V-Obstacles

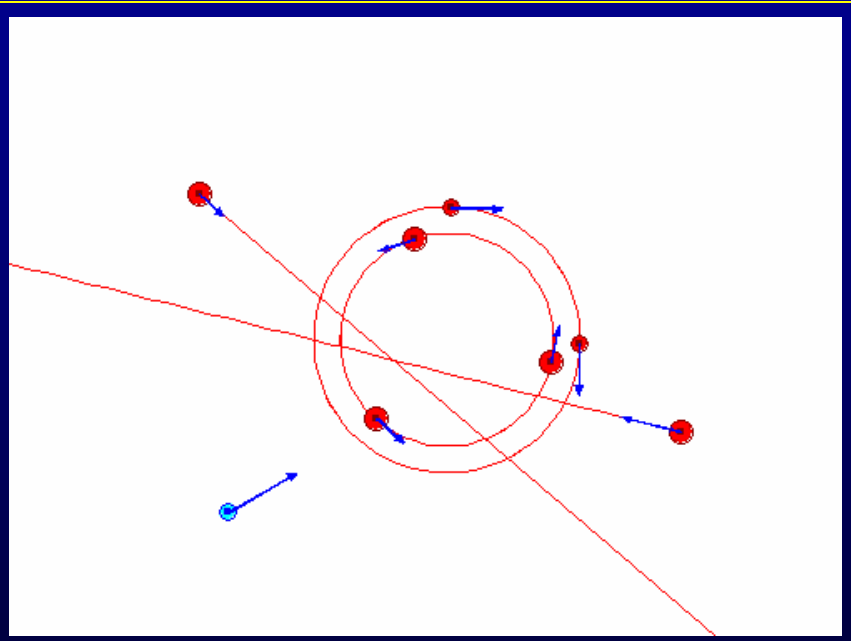
## (1) Instantaneous escaping trajectories

[Large et al. 03]

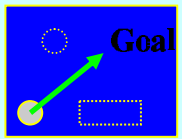


*A single velocity outside NLVO avoids the obstacle during the time interval for which the v-obstacle was generated*

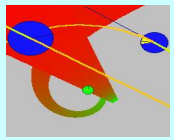


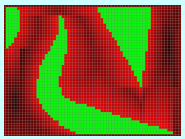
1 – Best velocity towards the goal



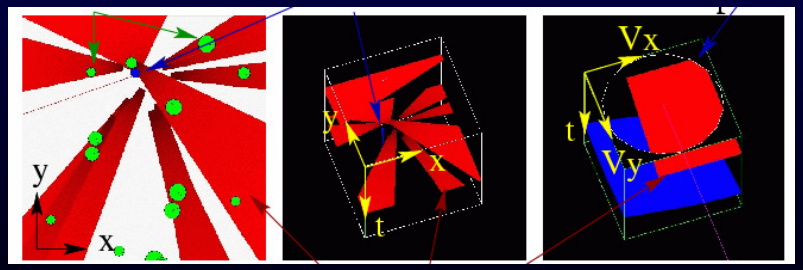

2 - Gradient on "time to collision"



3 - Gradient on "pseudo-distance to NLVO"



*=> Combines (1) with a risk function "(2) + (3)"*



# Safe navigation using V-Obstacles

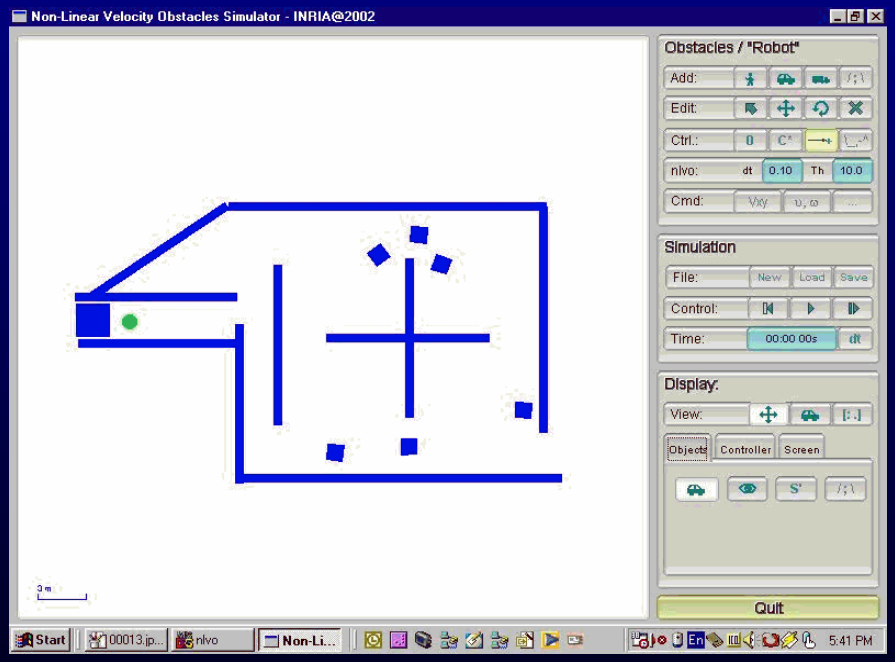
## (2) Iterative trajectory planning (ITP) in Vspace

*[Large et al. 03]*

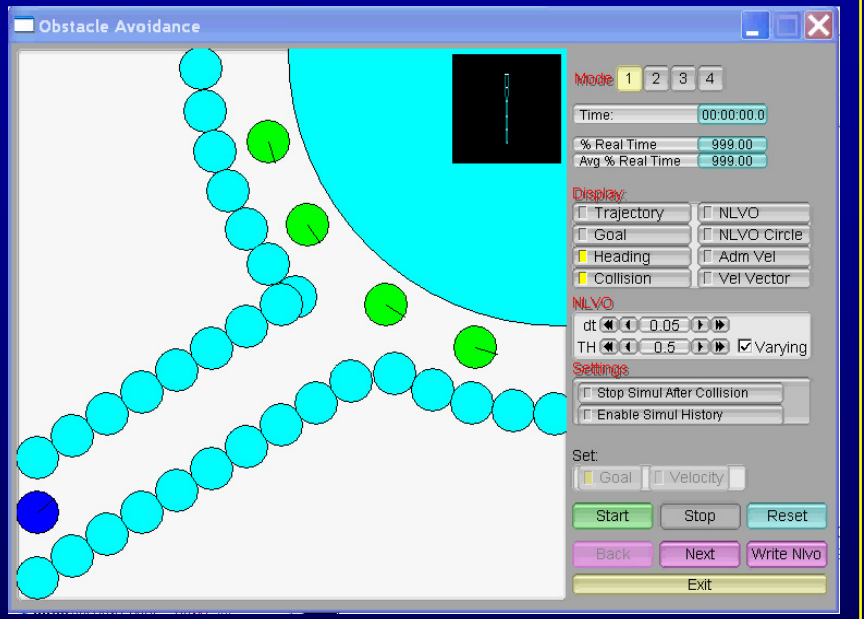
**ITP in Vspace =** *Graph search based on an alternate sequence of :*

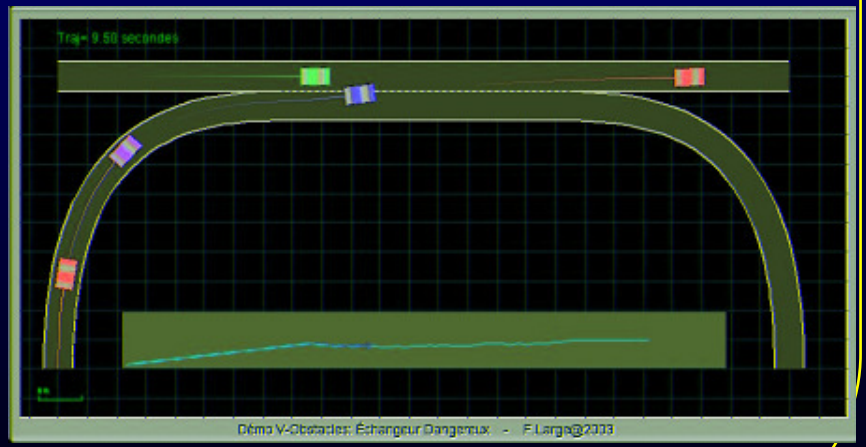
*1- Vspace evaluation at time $t_i$ (using NLVO)*

*2- Selection (using a cost function) of some related safe velocities for the robot*




*ITP among 20 obstacles (including walls)*
*Computation time < 0.01sec on PII-450MHz*



# - Part II -
# Dealing with real world constraints

*How to take into account Non-holonomic kinematic constraints ?*

*How to process the dynamics of both the robot and its environment ?*

***How to deal with uncertainty & hazards of the physical world ?***

# *Robust motion planning : The problem*

- **Problem**

  Path planning: *Perfect model assumption (robot, world)*

  Uncertainty problem: *Reality is not the model (control, sensing, model errors)*

  => *Possible failure at execution time*

- **Dealing with uncertainty**

  – **Model errors** (shape & location): *Growing the C-obstacles can be a solution, but at the expense of completeness …*

  – **Control & sensing errors** : *Necessity to use sensors both to "monitor the actual robot motion" and "command corrective motions"*

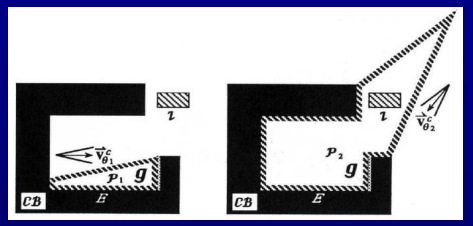
  => The motion plan becomes a "*motion strategy*" combining motion commands and sensor operations, even *sensor-based* motion commands (less sensitive to uncertainty)

- **Robust motion planning**

  Given a priori information about such uncertainties (*e.g.* bounds), the issue is to compute *safe motion strategies*, *i.e.* that take uncertainty into account *explicitly* so as to guarantee that the goal will be reached reliably

# *Robust motion planning : Main approaches*

- **Assembly planning:** *Contacts, high accuracy required for assembly*
  - **Skeleton refinement** *[Lozano-Perez 76] [Taylor 76] [Brooks 82][Puget 89]*
    *=> Nominal Plan + Critical points identification + Local patches*
  - **Preimage backchaining** *[Lozano-Perez 84] [Erdmann 86] [Lazanas & Latombe 92]*
    *=> Regions of C from which a "motion command" is guaranteed to attain a given goal recognizably*




- **Mobile robots:** *Odometry ⇒ increasing configuration uncertainty*
  - Usually dealt with at execution time *=> uncertainty is reduced by sensing appropriate environmental features*
  - Specific solutions (e.g. uncertainty fields) *[Takeda & Latombe 94] [Bouilly et al. 95] [Lambert 96][Fraichard & Mermond 98] [Lambert & Fraichard 00]*

- **Uncertainty models**: *Non-deterministic (bounded set) vs. Probabilistic*

# *Robust motion planning : Car-like robots*

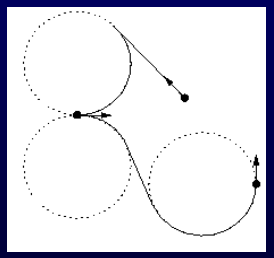
*[Fraichard & Mermond 98]*

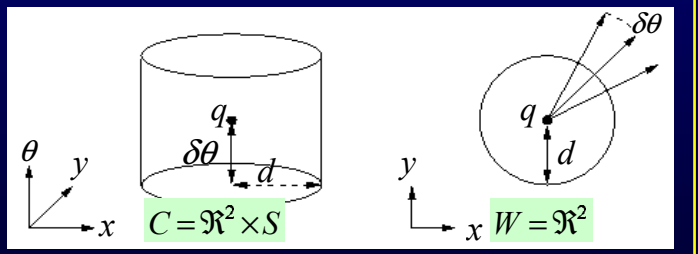
- **The problem :** NH constraints + Uncertainty *(Odometry & Control & Sensing)*
  - *=> Cumulative & unbounded "configuration uncertainty"*
  - *=> Relocalization devices: "Landmarks "*
  - *=> Motion strategy: "jump" from landmark to landmark until the goal is reached*

- **A possible approach :**
  - 1- Design of a 'local', *i.e.* uncomplete, robust path planner (LRPP)
  - 2. Embedding of LRPP within a global path planning scheme
    - *e.g. ACA [Mazer et al., 98],* or *PPP [Svestka & Overmars, 98]*

➢ **LRPP**

*1- Computes Reeds & Shepp paths*
*2- Evaluates the uncertainty evolution*
*3- Performs robust collision checking*






*Uncertain configuration:* $(q,u)$ where: $u = (d, \delta\theta)$

➢ **Global Path Planning**

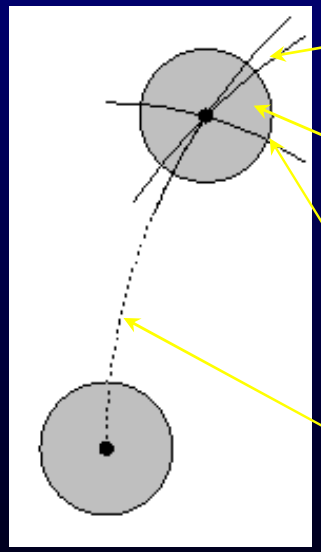
*PPP was selected (easy to implement, efficient)*

# Uncertainty evolution

- **Configuration uncertainty**
  - Travelling uncertainty (in %) $\kappa$ => *Actual travelling distance bounded by* $l \pm \kappa l$
  - Steering uncertainty $\delta\varphi$ => *Actual steering angle bounded by* $\varphi \pm \delta\varphi$

- **Uncertainty evolution**

  *The problem is to determine the set of configurations "possibly reached" at the end of an elementary path*

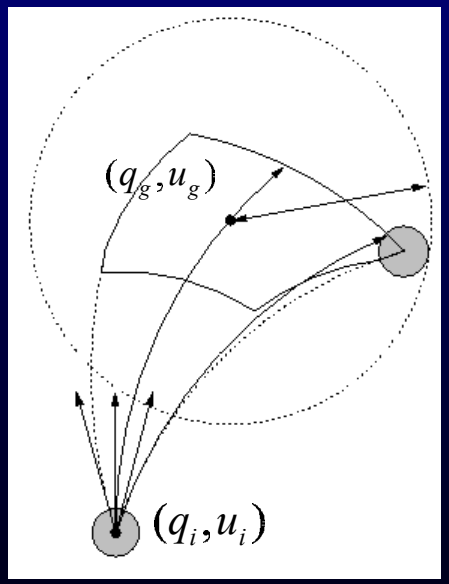


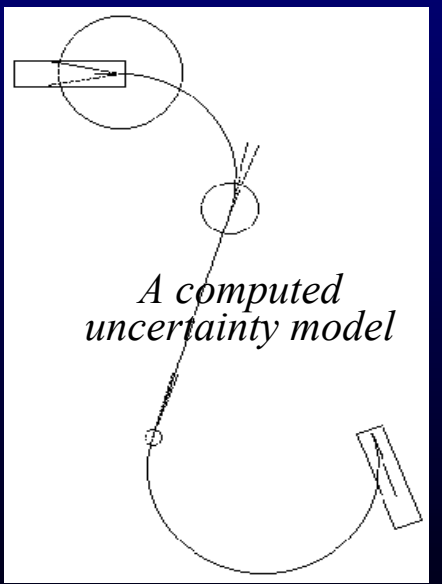
*Steering uncertainty (arc of curve)*

*Position uncertainty (disc)*

*Orientation uncertainty (circular arc)*

*Travelling uncertainty (circular arc)*

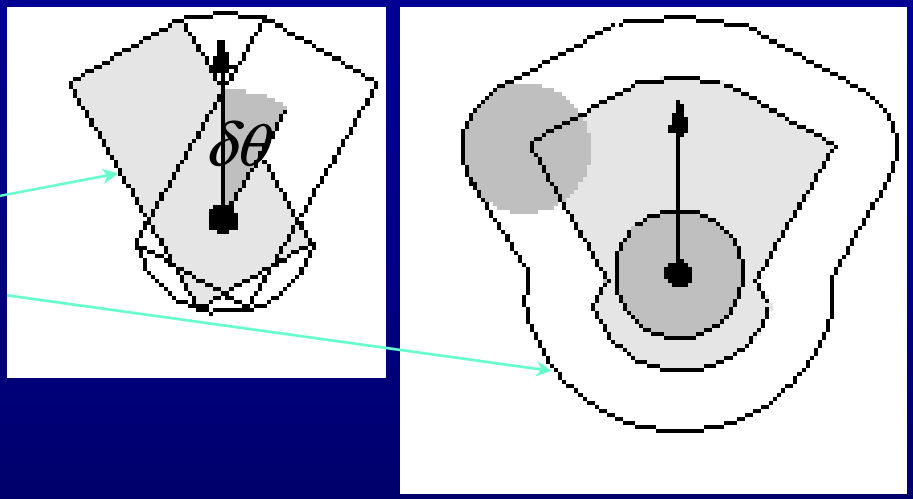


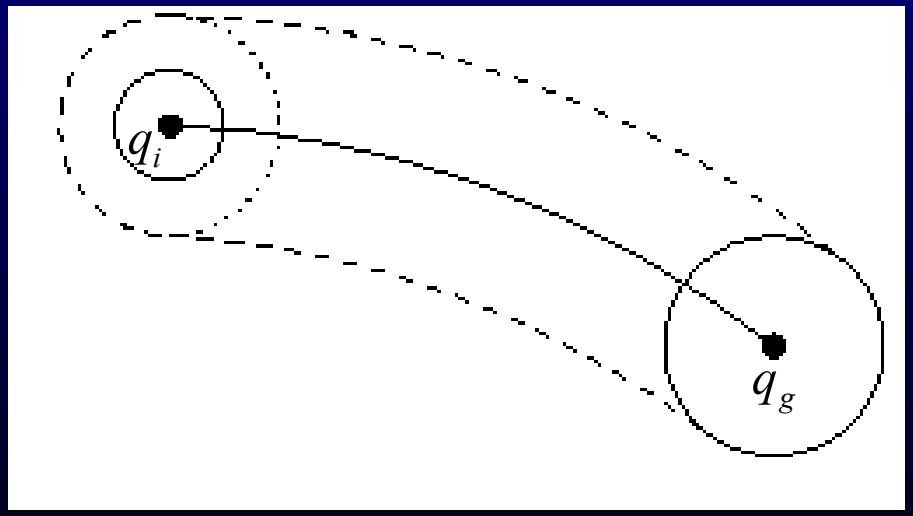
# Robust collision checking

***=> Collision checking is performed in the workspace (swept region analysis)***

- Workspace region occupied by the vehicle
  - *With orientation uncertainty*
  - *With orientation + position uncertainty*



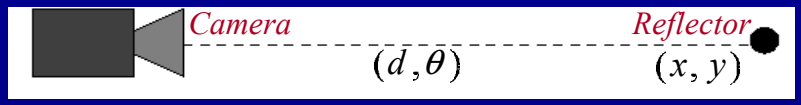

- Region swept along an elementary path
  - *Conservative approximation*
  - *Recursive and resolution-dependent collision checking function*



# Some experimental results

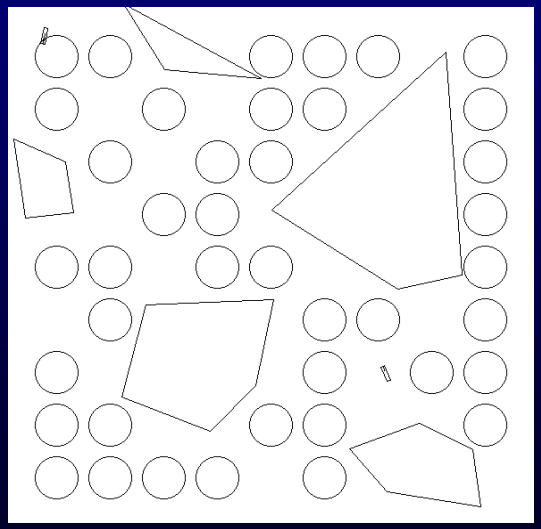
- **Landmark**

  *On-board linear camera + on-site reflectors*

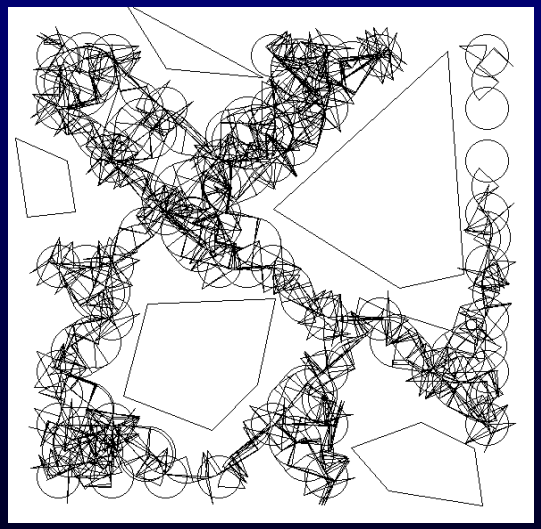



$(d,\theta)+(x,y) \Rightarrow$ *relocalization* $\quad (q,u) \rightarrow (q,u_{landmark})$

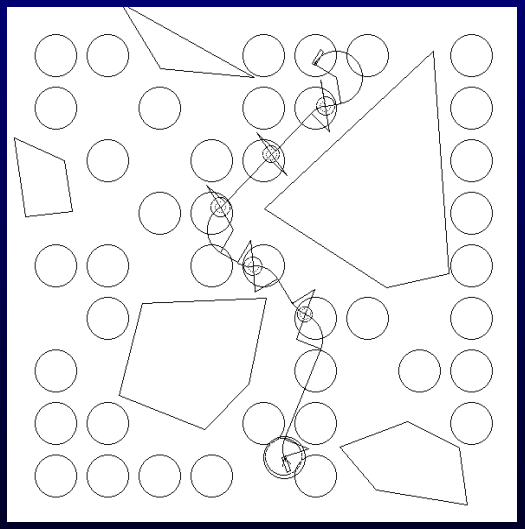

- **Computed solutions**



Workspace + landmarks



Roadmap

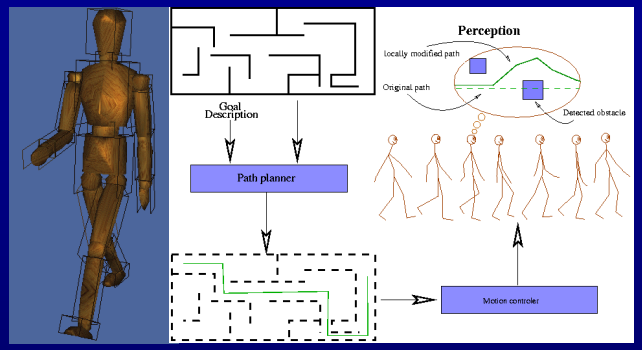


A robust motion strategy

# Dealing with hazards at execution time

*Motion planning + Reactive navigation*

- **Combining on-line planning & navigation functions**




Dynamic path planning

=> *Adriane's Clew Algorithm*

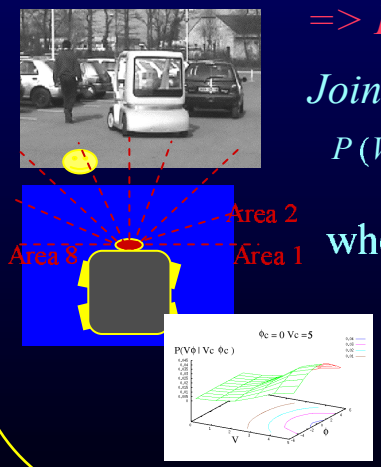
*[Ahuactzin 94]*

Reactive navigation

=> *Path tracking & Obstacle avoidance*

*[Raulo & Ahuactzin &Laugier 00]*

- **Obstacle avoidance using learned « behaviors » (bayesian programming)**




=> *Probability distributions on the controls (v,ϕ)*

*Joint distribution for the fusion :*

$$P(V \otimes \phi \otimes D_1 \otimes \ldots \otimes D_8) = P(V \otimes \phi) \prod_{i=1}^{8} P_i(D_i / V \otimes \phi)$$

where :

$$P(V \otimes \phi) = \text{Uniform}$$

$$P_i(D_i / V \otimes \phi) = \frac{P_i(D_i) P_i(V / D_i) P_i(\phi / D_i)}{\sum_{D_i} P_i(D_i) P_i(V / D_i) P_i(\phi / D_i)}$$

**=> *See tutorial 2***

# - Part III -
# Reactive Navigation techniques

# *Decisional & Control Architectures*

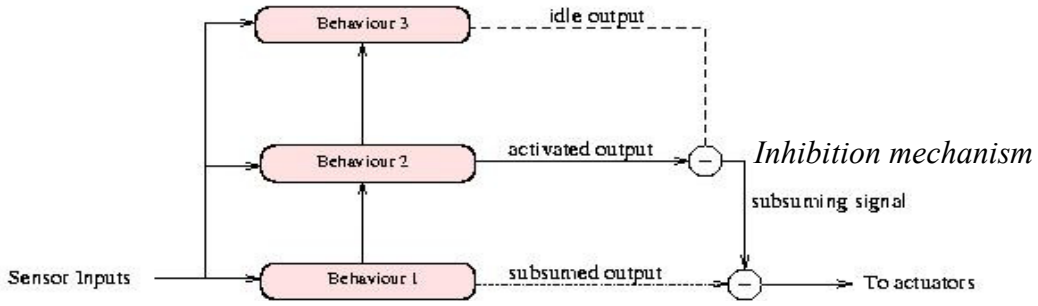
- **Deliberative approaches** *[Moravec83][Nilsson84]*
  - **Sequential processing *(SMPA)***

    *sensing => modeling => planning => action (motion execution)*
  - **High-level reasoning …. *but time consuming & high uncertainty***

  *=> Unsuitable for (fast) changing & complex environments*
- **Reactive approaches** *[Brooks86][Zapata90]*
  - **Parallel decomposition into simple behaviors**

    *sensing => sensor-based behaviors => action (controls)*
  - **Real-time processing, robust, incremental … *but no high-level reasoning***

  *=> Well adapted to dynamic environments, but unsuitable for complex missions*




*Behaviors = finite state machines communicating using messages*

*e.g. Move towards a goal, Explore, Avoid obstacles, Follow a wall*

Subsumption architecture
*[Brooks86]*

# Decisional & Control Architectures (C'ed)

- **Hybrid approaches**
  - **Adding a** *"reflex layer"* **to a SMPA architecture** *[Payton86][Gat90]*
  - **Adding a** *"planning layer"* **to a reactive architecture** *[Arkin87] [Connel92]*
  - *Three-layered* **hybrid architectures** *[Alami et al. 98] [Laugier et al. 97]*
    *=> Decision layer + Reactive mechanisms + Skills*

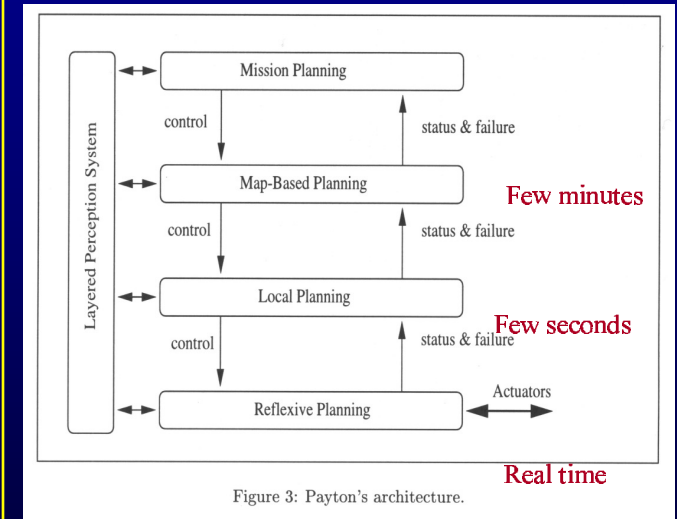
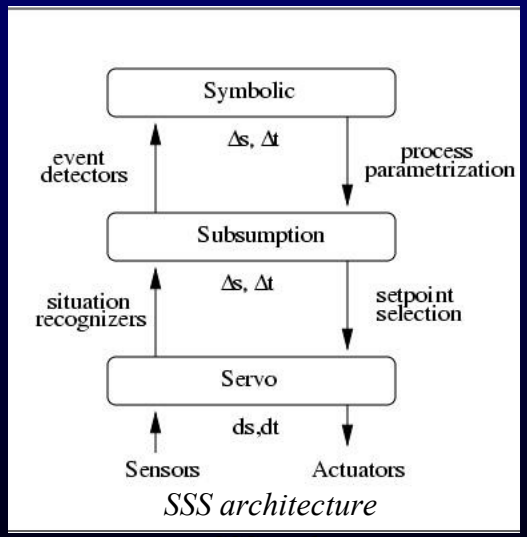


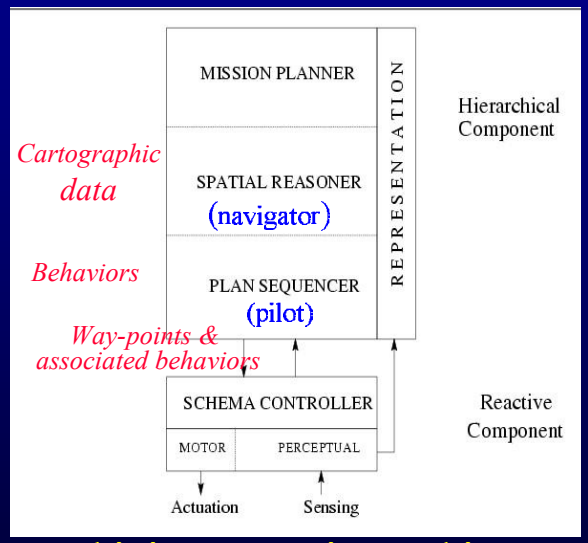
Figure 3: Payton's architecture.

Hierarchical architecture including a "reflex layer" *[Payton 86]*

Reactive architecture including a symbolic layer *[Connell 92]*

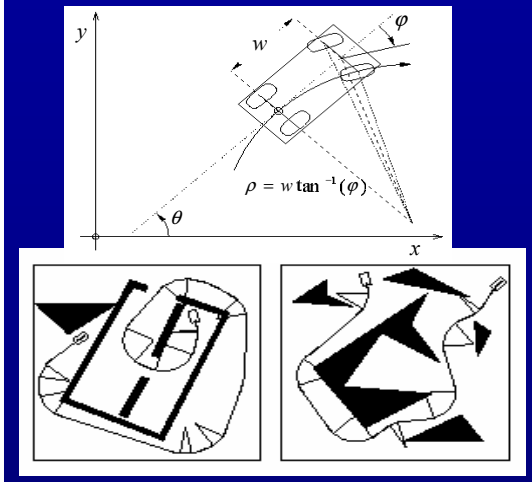


*SSS architecture*



Combining a reactive architecture with a deliberative layer *[Arkin87]*

# Automatic driving architecture (Inria)

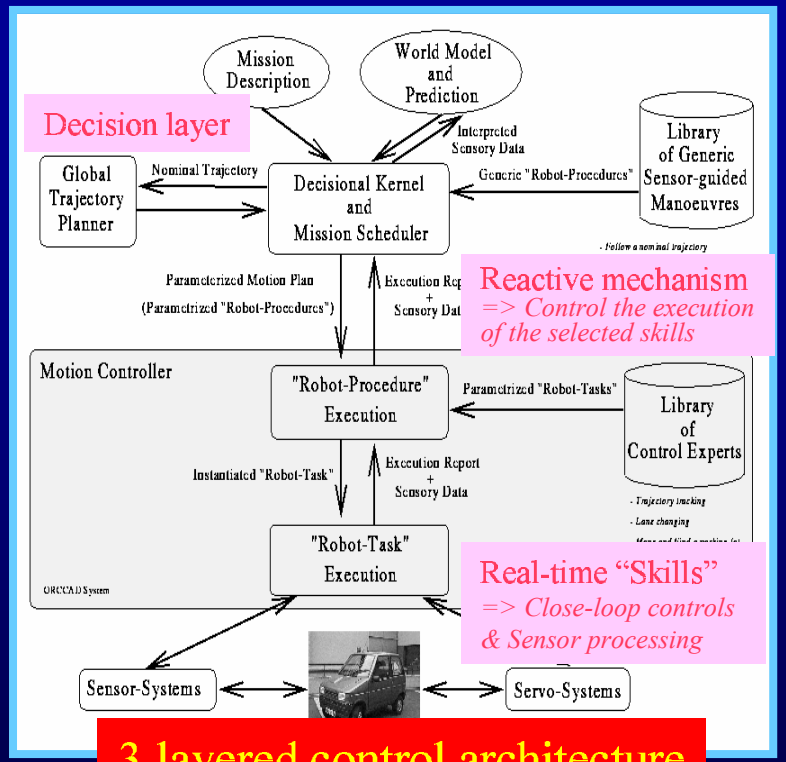



Planning CC-paths

*(kinematic constraints ...)*

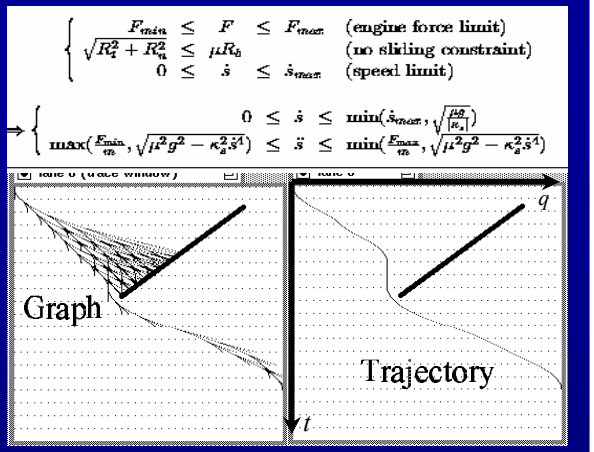
continuous curvature profile + upper-bounded curvature & curvature derivative

*[Scheuer & Laugier 98 ]*




3-layered control architecture

*[Laugier et al. 98 ]*




Kinodynamic Motion Planning

*(Dynamic constraints ...)*

*[Fraichard 92 ]*

Platooning *[Parent & Daviet 96]*

Lane Changing & Obstacle avoidance

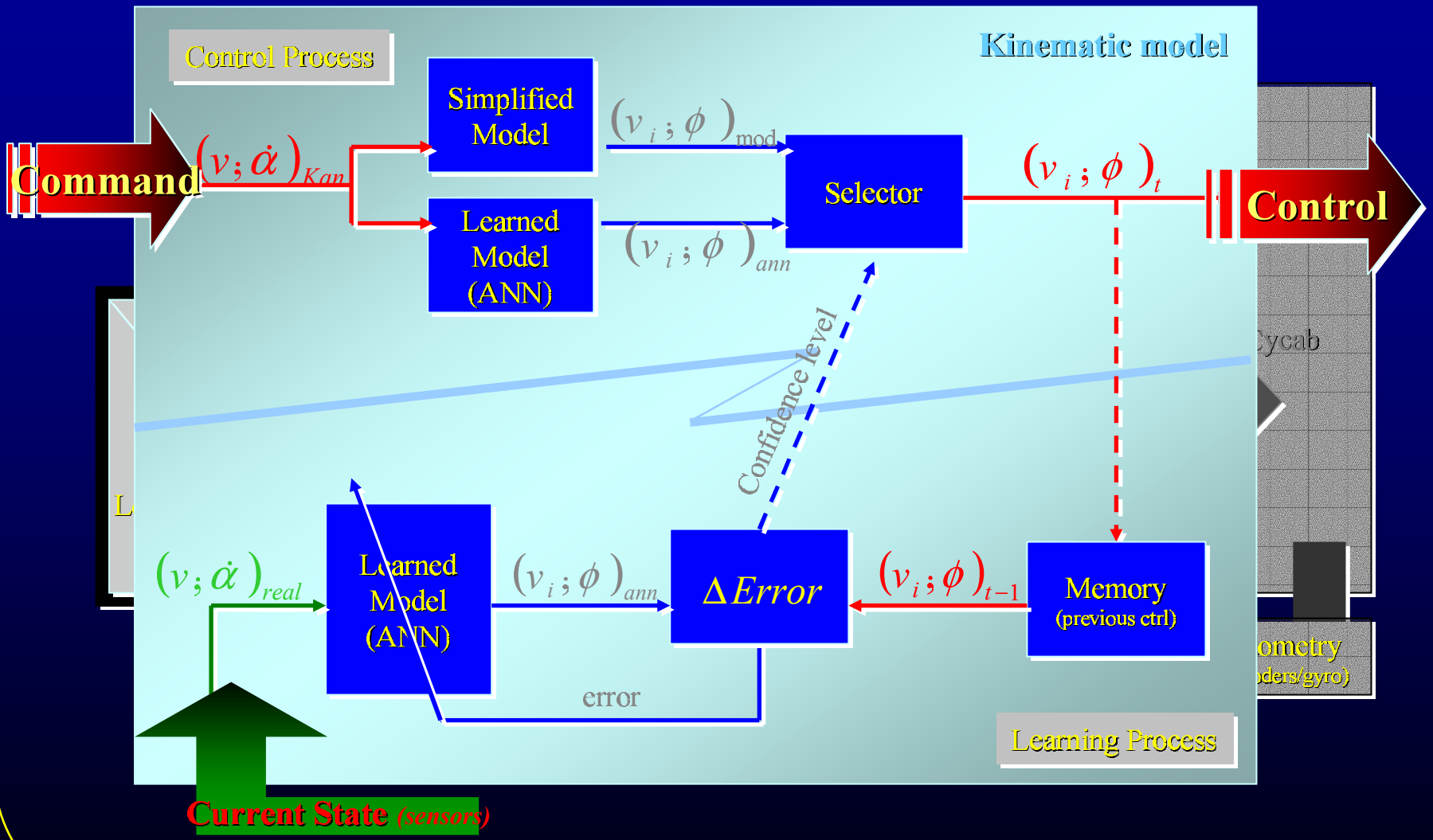
*[Laugier et al. 98]*

Automatic Parallel Parking

*[Paromtchik & Laugier 96]*

# Model learning & Improved Trajectory Tracking

*[Large & Laugier 00]*



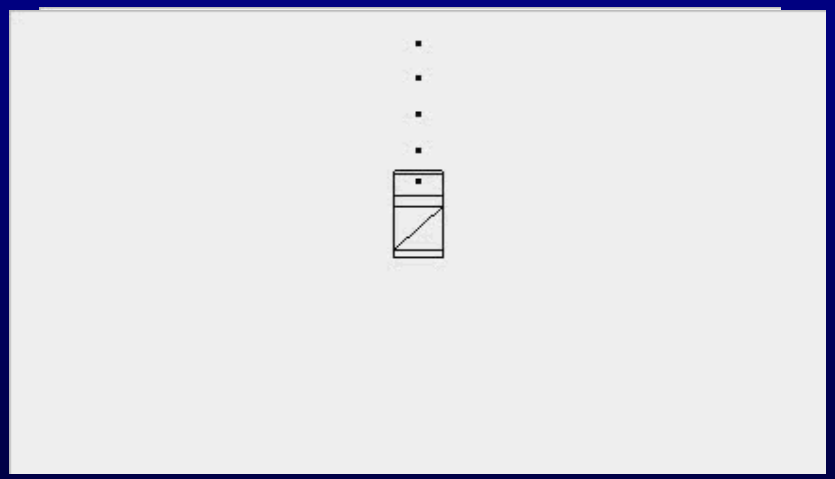



INRIA RHÔNE-ALPES

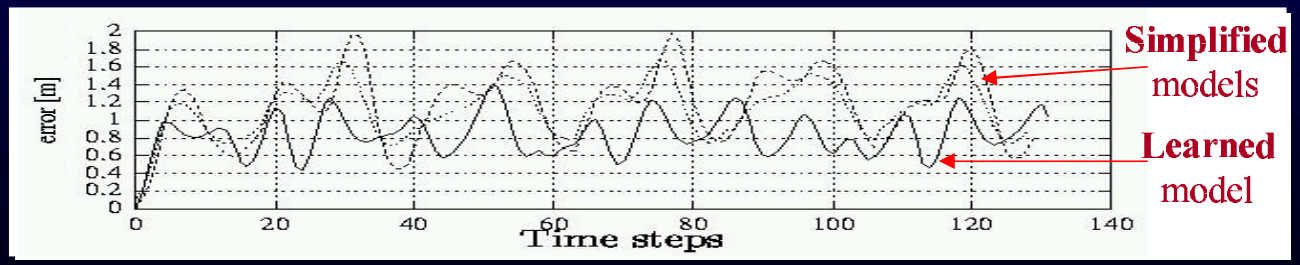
# Model learning & Improved Trajectory Tracking

## Experimental results


Sensing error:
*0.15 rad on the steering angle*



Our control system
after a first training



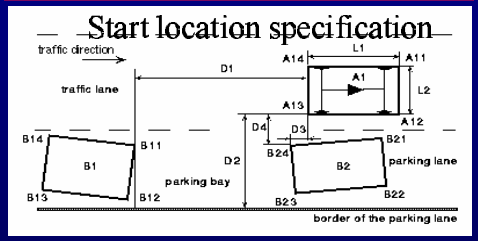
# « Platooning » *[Parent & Daviet 96]*

*Electronic « Tow-bar »*

*CCD Linear camera + Infrared target*
*(high rate & resolution)*

# Automatic parking maneuvers [Paromtchik & Laugier 96]

*On-line local world reconstruction & Incremental motion planning*




$$\begin{cases} \phi(t) = \phi_{\max}\, k_\phi\, A(t), & 0 \le t \le T \\ v(t) = v_{\max}\, k_v\, B(t), & 0 \le t \le T' \end{cases} \qquad \phi_{\max} > 0,\ v_{\max} > 0,\ k_\phi =$$

$$A(t) = \begin{cases} 1, & 0 \le t < t' \\ \cos\dfrac{\pi(t-t')}{T^*}, & t' \le t \le T - t', \\ -1, & T - t' < t \le T \end{cases} \qquad t' = \frac{T - T^*}{2}, T^* < T$$

$$B(t) = 0.5\left(1 - \cos 4\pi t/T\right), \qquad 0 \le t \le T$$

=> On-line motion planning using sinusoidal controls $\phi(t)$ and $v(t)$

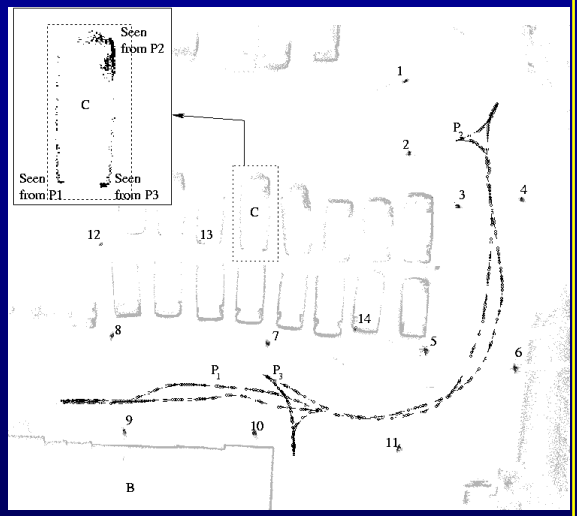
*(search for control parameters $T$ and $\phi max$)*

# Autonomous navigation in a learned environment

*[Pradalier & Hermosillo 03]*

⇒ *Several functionalities (learned and downloaded) have to be robustly combined*

*Incremental world modeling & localization + Motion planning + Autonomous sensor-based navigation*

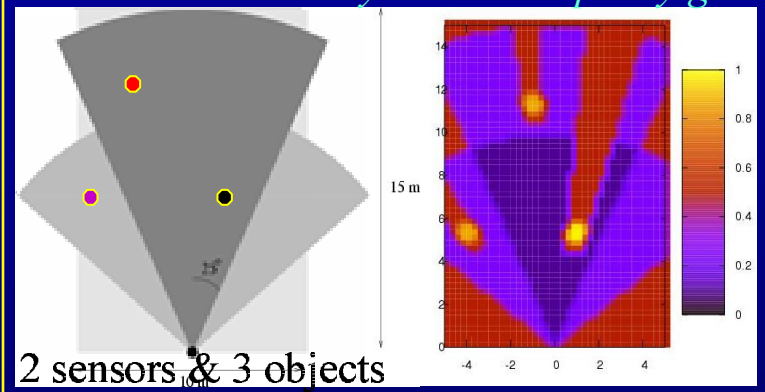



***SLAM***
***+***
***Motion planning***
***+***
***Reactive navigation***

# Robust obstacles tracking & avoidance

## Bayesian Occupancy Filter (BOF) [Coué et al. 03]

*Dynamic occupancy grid*

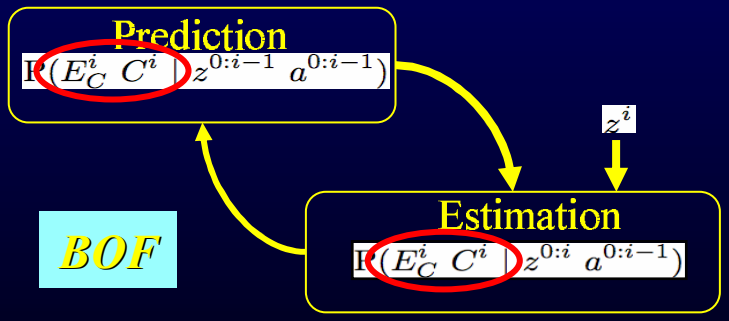



$P([E_c=1] \mid z_{1,1}\, z_{1,2}\, z_{2,1}\, z_{2,2}\, c)$

$c = [x, y, 0, 0]$

$z_{1,1} = (5.5, -4, 0, 0)$ $z_{1,2} = (5.5, 1, 0, 0)$

$z_{2,1} = (11, -1, 0, 0)$ $z_{2,2} = (5.4, 1.1, 0, 0)$




Pedestrian avoidance using the BOF

=> *See tutorial 2*

# Conclusion (Tutorial 1)

- **A formal framework and various algorithmic tools exists for solving basic motion planning problem**

- **New techniques have also been developed for taking into account some of characteristics of reel problems** *(kinematic constraints, moving obstacles, uncertainty & hazards)*

***… BUT***

- **The scalability is still an open problem** *(i.e. how to master the complexity of most of real world applications ?)*

- **Coping with the full characteristics of the physical world** *(dynamicity, uncompleteness …)* **is still a largely open problem.** *New probabilistic approaches are also required to solve such problems (see tutorial 2)*