HOW TO DEAL WITH ROBOT MOTION? APPLICATION TO CAR-LIKE ROBOTS

by

Dominique Luzeaux

Memorandum No. UCB/ERL M92/7

16 January 1992

HOW TO DEAL WITH ROBOT MOTION?
APPLICATION TO CAR-LIKE ROBOTS

by

Dominique Luzeaux

Memorandum No. UCB/ERL M92/7

16 January 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

Contents

Abstract

In this paper, we will compare different types of control, like nonlinear control, motion planning, fuzzy control, neural control, rule-based incremental control; we take robot motion as a comparative field, and more specifically motion of car-like robots.

We will compare the different approaches on two main points: their theoretical basis (controllability, stability, robustness for a given application) and their conviviality (easy and user-friendly implementation of the control, application of artificial intelligence methods to improve control when faced with unknown situations). Very often, control techniques of the first type are called "classical control", while methods of the second type are called "intelligent control". We do not find these appellations are very relevant, as they tend to classify the first methods as guaranteed to work but impracticable and the second methods as easy to implement but magical; we will try to see in each method the advantages and the drawbacks rather than going on with the useless dialectic between Moderns and Ancients. It is much more interesting to look for a technique that integrates both the theoretical basis and the conviviality: one solution could be rule-based incremental control or other similar hybrid approaches.

Autonomous robots are nowadays very popular, for instance in industry, in space technologies, in surgery or even in house keeping facilities. Concerning robots, two main problems arise, which are of course directly related to the tasks the robots have to perform: motion and grasping. We will only discuss in the next parts the motion problem.

This work has been done at the Department of Electrical Engineering and Computer Science at the University of California, Berkeley. The author is on leave from the Department Systèmes de Perception at the Centre de Recherche et d'Etudes d'Arcueil, France.

Chapter 1

Robot motion planning

Motion planning can be loosely stated as: how can a robot decide what motions to perform in order to achieve some goal in a physical space? This assumes the robot has some perception of the real world and interacts with its environment. We will not deal with the perception problem, but the interaction with the real world is the key point: how to avoid obstacles while performing a task? In order to simplify, we will not consider movable obstacles; however we will consider holonomic and nonholonomic robots.

Some basic definitions follow, then different techniques are discussed. These techniques try to solve the robot motion problem by finding a trajectory between an initial and a final situation that avoids all obstacles; whether a given robot can or cannot follow this trajectory is not taken into consideration; as the generic name of these methods tells us, we are dealing here with *planning* and not with *control.*

1.1 Some useful definitions

For a survey of different problems and algorithms of robot motion planning, see [Lat91]. The notations which follow are taken from this reference.

Let us call $\mathcal{A}$ the robot, moving in a physical world $\mathcal{W}$ (represented as $\mathbf{R}^N$) and let $\mathcal{B}_i$ be some fixed rigid objects called obstacles. $\mathcal{A}$ is assumed compact and the $\mathcal{B}_i$ are assumed closed but not necessarily bounded. The robot is determined by its position and its orientation in the euclidian space $\mathbf{R}^N$; a *configuration* $\mathbf{q}$ of $\mathcal{A}$ is a specification of the position and the orientation of $\mathcal{A}$ with respect to a reference frame. The *configuration space* $\mathcal{C}$ is the set of all configurations of the robot. The subset occupied by $\mathcal{A}$ at configuration $\mathbf{q}$ is denoted by $\mathcal{A}(\mathbf{q})$.

A *path* of $\mathcal{A}$ from an initial configuration $\mathbf{q}_{\text{init}}$ to a goal configuration $\mathbf{q}_{\text{goal}}$ is a continuous map $\tau : [0,1] \rightarrow \mathcal{C}$ with $\tau(0) = \mathbf{q}_{\text{init}}$ and $\tau(1) = \mathbf{q}_{\text{goal}}$.

Every obstacle $\mathcal{B}_i$ maps in $\mathcal{C}$ to a region called a *C-obstacle*:

$$\mathcal{CB}_i = \{\mathbf{q} \in \mathcal{C} \mid \mathcal{A}(\mathbf{q}) \cap \mathcal{B}_i \neq \emptyset\}$$

The complement of the union of such regions is called the *free space* $\mathcal{C}_{\text{free}}$. Any configuration in this space is a *free configuration.* A *free path* stays always in $\mathcal{C}_{\text{free}}$. It is then obvious that two configurations are connected by a free path if and only if both configurations belong to the same connected component of $\mathcal{C}_{\text{free}}$. A *semi-free path* stays in the closure of $\mathcal{C}_{\text{free}}$.

It may be shown that $\mathcal{C}$ is a smooth manifold, meaning it is locally diffeomorphic to a power of $\mathbb{R}$ (any configuration may be mapped to a vector in $\mathbb{R}^N \times SO(N)$; it is then obvious to define local coordinates [Ave83]). Other interesting results are: $\mathcal{CB}$ is closed, $\mathcal{CB}$ is compact if $\mathcal{B}$ is compact and $\mathcal{CB}$ is connected if both $\mathcal{A}$ and $\mathcal{B}$ are connected.

Contact space is the subset of all configurations at which $\mathcal{A}$ touches any obstacle without overlapping any obstacle; the union of free space and contact space is *valid space.* A *valid path* is a path that stays of course in valid space.

1.2 Roadmap methods

The roadmap approach finds the connected components of a high dimensional set, like $\mathcal{C}_{\text{free}}$, by computing a one dimensional curve, the roadmap $\mathcal{R}$; furthermore, this skeleton is connected within each connected component of the set. Path planning reduces to connecting $\mathbf{q}_{\text{init}}$ and $\mathbf{q}_{\text{goal}}$ to this roadmap. Various methods have been proposed to construct such roadmaps: visibility graphs, Voronoi diagrams and silhouette method [Can87].

The *visibility graph* is obtained by joining every two vertices of all obstacles if the resulting segment does not intersect the interior of an obstacle; this method applies to two dimensional configuration spaces with polygonal C-obstacles. To find a path between two configurations reduces to connect both configurations to the path and to follow a sequence of straight lines included in the visibility graph.

A *Voronoi diagram* is the set of all free configurations whose minimal distance to the C-obstacle region is achieved with at least two points in the boundary of the C-obstacle region; it maximizes the clearance between the robot and obstacles. Finding a path between two configurations reduces then to connect both configurations to the Voronoi diagram.

The *silhouette method* generates semi-free paths, i.e. contact with the obstacles is allowed. It consists of sweeping a hyperplane P across free space by following a given direction. The locus of all extremal points of $P \cap \mathcal{C}_{\text{free}}$ is a set of curves, called the silhouette. During the sweeping process (let us say, the hyperplane was following the x axis) the number of connected components of the silhouette has changed at a finite number of critical points x_i. For these points, curves

contained in the intersection of the sweeping hyperplane and the free space are built in the following way: a plane with one less dimension is swept across each of these intersections by following a new direction and the method is applied recursively, until no new critical points appear. The silhouette method obviously converges as for each step a $(m-1)$ dimensional plane is swept across a m dimensional space, and m decreases by 1 from one step to the other. This method has a high complexity and some improvements exist [CL90].

1.3 Cell decomposition

We give only here the general sketch of the method. Further references are to be found in [Lat91].

The principle is to decompose the robot's free space into a collection of non-overlapping regions, called *cells*, and to build a *connectivity graph* which represents the adjacency between the cells. The purpose of the algorithms is to find then a sequence of cells connecting the cell containing $\mathbf{q}_{\text{init}}$ with the cell containing $\mathbf{q}_{\text{goal}}$. Of course the cells must have a nice shape in order to compute easily a path between any two configurations inside a cell, and it must be an easy task too to test the adjacency of two cells. If these conditions are fulfilled, finding a path is reduced to searching through a graph, which is not a trivial problem for computer applications. The shapes of the cells are usually triangles, rectangles or slices. Algorithmic complexity is one of the main drawbacks of this method.

1.4 Nonholonomic constraints

1.4.1 Definitions

In the basic problem, the robot is assumed to be a free-flying object, it may move in any direction and any orientation, the only constraints are due to the obstacles. In many problems, such a situation is not realistic: for example, a usual car cannot translate sideways. Additional kinematic constraints are then to be considered; they may be divided into two classes: holonomic and nonholonomic constraints.

Let us assume the configuration space is minimal (the right number of parameters has been found to describe accurately the robot); a *holonomic* constraint is an equality relation among these parameters which can be solved for one of these parameters: the general expression is $F(\mathbf{q},t)=0$ where F is a smooth function with non-zero derivative. Such a relation reduces the dimension of the configuration space by one; holonomic constraints affect the definition of the robot's configuration space and therefore its topological properties (connectedness may be lost). A *nonholonomic* constraint is a non-integrable equation involving the configuration parameters and their derivatives (velocity parameters): the general expression is $G(\mathbf{q},\dot{\mathbf{q}},t)=0$ where G is a smooth function. Such a constraint does not reduce the dimension of the configuration

space, but reduces the dimension of the space of differential motions (the tangent space) at any configuration. By replacing the equality sign by an inequality, it is obvious to define holonomic and nonholonomic inequality constraints.

Solving the robot motion problem with nonholonomic constraints is an interesting challenge and the previous methods have to be modified in order to adapt to this new problem.

Let $\mathcal{A}$ be a robot subject to nonholonomic constraints; a path τ is *feasible* if it is piecewise f class C^1 and satisfies the constraints (if $G(\mathbf{q}, \dot{\mathbf{q}}) = 0$ is a nonholonomic constraint, then τ must satisfy $G(\tau(s), \frac{d\tau}{ds}(s)) = 0$ for all $s \in [0, 1]$). A robot is *fully controllable* if for any distribution of obstacles, if there exists a free path between any two configurations, then there also exists a feasible path between these two configurations.

1.4.2 Car-like robot with trailers: equations

We give in this section the general equations of the car-like robots with trailers, which will be used in some applications.

Let us consider a multi-body system constituted by a car (body 0) and n trailers (bodies $1, \cdots, n$). The midpoint between the rear wheels is taken as the reference point for each body; its coordinates are (x_i, y_i) in a given fixed reference frame and the orientation is given by θ_i, the angle between the main axis of the body and the x-axis. The space of all different placements is then $3(n+1)$ dimensional.

In order to form a convoy (the car pulls all the other trailers which are hooked up to the next), each trailer is assumed to be hooked up to the midpoint between the rear wheels of the preceding body (this hooking system simplifies the next equations; other hooking systems introduce new variables for each trailer which do not cancel); the distance between the hitch and the wheels of the i-th trailer is d_i. This yields the $2n$ holonomic equalities:

$$\begin{cases} x_i - x_{i-1} &= -d_i \cos\theta_i \\ y_i - y_{i-1} &= -d_i \sin\theta_i \end{cases}$$

The configuration space of this multi-body system is a submanifold of dimension $3(n+1) - 2n = n + 3$ in the placement space. A possible parametrization is $(x_0, y_0, \theta_0, \theta_1, \cdots, \theta_n)$ which yields the space $\mathbb{R}^2 \times (S^1)^{n+1}$.

Each body is rolling on the ground without sliding; it is thus submitted to the following nonholonomic equality:

$$\dot{x}_i \sin\theta_i - \dot{y}_i \cos\theta_i = 0$$

This equation tells that the speed vector is always parallel to the direction of the body. The multi-body system has $n+1$ nonholonomic constraints. The number of degrees (difference between the

dimension of the configuration space and the nonholonomic links) is therefore $(n+3)-(n+1)=2$ (obviously the two degrees of freedom of the leading car).

The equations of the multi-body system are:

$$\begin{cases} \dot{x_0} &= \alpha\cos\theta_0 \\ \dot{y_0} &= \alpha\sin\theta_0 \\ \dot{\theta_0} &= \beta \\ \dot{\theta_1} &= \frac{\alpha}{d_1}\sin(\theta_0-\theta_1) \\ \dot{\theta_2} &= \frac{\alpha}{d_2}\sin(\theta_1-\theta_2)\cos(\theta_0-\theta_1) \\ & \vdots \\ \dot{\theta_n} &= \frac{\alpha}{d_n}\sin(\theta_{n-1}-\theta_n)\prod_{i=1}^{n-1}\cos(\theta_{i-1}-\theta_i) \end{cases}$$

α and β are any two reals. Other parametrizations lead to similar equations. We see that the two inputs are the speed of the rear axis of the lead car and its angular velocity, which is related to the steering angle as we will see now.

Let us study now the car (body 0); we call R the midpoint of the rear axis with coordinates (x_0, y_0, θ_0). We write now the equations of F, the midpoint of the front axis. Its coordinates are (x_f, y_f); let us call ϕ the steering angle, it is the angle between the main axis of the car and the velocity vector of F (as the car does not slide, it happens to be the angle between the wheels and the axis of the car). Let us call v the speed of F. L will be the distance between the front axis and the rear axis. We have:

$$\begin{cases} x_f &= x_0 + L\cos\theta_0 \\ y_f &= y_0 + L\sin\theta_0 \end{cases}$$

As the robot has a front-wheel drive and the front wheels do not slide, we have:

$$\begin{cases} \dot{x}_f &= v\cos(\theta_0+\phi) \\ \dot{y}_f &= v\sin(\theta_0+\phi) \end{cases}$$

The rear wheels do not slide either, as we saw previously:

$$\dot{x_0}\sin\theta_0 = \dot{y_0}\cos\theta_0$$

By derivating the first two equations and using the three next, we have:

$$\begin{cases} \dot{\theta_0} &= v\dfrac{\sin\phi}{L} \\ \dot{x_0} &= v\cos\theta_0\cos\phi \\ \dot{y_0} &= v\sin\theta_0\cos\phi \end{cases}$$

If ϕ is constant, we may integrate the various equations and find that F describes a circle with radius $\frac{L}{\sin\phi}$ while R describes a circle with radius $\frac{L}{\tan\phi}$.

In a real car, mechanical stops in the steering gear constrain the steering angle in such a way that $|\phi| \leq \phi_{\max} < \frac{\pi}{2}$. It is easy to see that this constraint can be rewritten as:

$$\dot{x_0}^2 + \dot{y_0}^2 - {\rho_{\min}}^2 \dot{\theta_0}^2 \geq 0$$

where $\rho_{\min} = \frac{L}{\tan\phi_{\max}}$.

1.5 Potential fields

The previous planning path methods tried to summarize the connectivity of the free space into a graph which is then searched for a path. The potential fields methods proceed differently: the robot represented as a point in configuration space is considered as a particle under the influence of a *potential field* whose local variations reflect the free space: the potential function is the sum of an attractive potential, which pulls the robot towards the goal, and a repulsive potential, which pushes the robot away from the obstacles. The gradient of the potential is interpreted as an artificial force which shows the most promising direction to follow. The potential field method acts essentially as a fastest descent optimization procedure.

One of the main drawbacks of potential functions is the existence of local minima which can catch the robot-particle; hence dealing with local minima is the major issue of these methods. Several solutions have been found [BL89, Lat91]: designing potential fields with no or very few local minima and designing techniques for escaping from local minima by a series of random motions.

Chapter 2

Robot motion and control

We will now address a slightly different problem, namely the control of a car-like robot: the problem is not to find a trajectory between two situations, but to steer the robot between these two situations.

We saw previously that all paths were not feasible, because a robot is not a free-flying object and has therefore limitations on its movement capacities. We deal in this part with the feasibility. More precisely, we will first see how we may steer a robot in a space without obstacles, and then we will consider the problem with obstacles. A way to solve this last problem is, for instance, to find a trajectory by one of the motion planning techniques, and then to track this trajectory; we must then prove that this tracking is possible for any trajectory. Let us recall that most trajectories found by the motion planning techniques are in free space and therefore there exists some neighborhood of the trajectory which is safe from obstacles; if there is some way to stay as close as possible to the trajectory, then the robot will be steered between the initial and final situations and stay clear of obstacles.

2.1 Non linear control

2.1.1 Lie algebras

There has been some work on the application of Lie algebras, one of the powerful tools of non linear control theory, to the car parking problem [Lau90, Mur90, RM91]: when a car tries to park, usually it first backs up and turns right, then it drives forward and turns left; the second maneuver does not undo the first as is well known! The reason is that forward-backward motion and left-right motion do not commute; actually, the resulting drift (useful for parking!) is the Lie bracket of these two motions.

More formally, given 2 smooth vector fields (a smooth vector field is a C^∞ map[1] from U, an open set of a Banach space E, into E), the Lie bracket of these vector fields X and Y is the vector field $[X,Y] = DY \bullet X - DX \bullet Y$ and if ϕ_X is the flow of X ($\phi_X : \mathbb{R} \times U \to U$ is a flow of X if for any $x \in U$, $t \mapsto \phi_X(t,x)$ is the maximal solution of $\frac{d}{dt}\phi_X(t,x) = X(\phi_X(t,x))$ such that $\phi_X(0,x) = x$), it can be shown that:

$$\lim_{t \to 0} \frac{\phi_{-Y}(t,.)\phi_{-X}(t,.)\phi_Y(t,.)\phi_X(t,.)(x_0)}{t^2} = [X,Y](x_0)$$

which relates Lie brackets to the composition of non commutative motions thus leading to a drift. By taking Lie brackets of higher order (such as $[X,[X,Y]]$) a motion in almost any direction may be intuitively obtained. Such considerations have led to applying Lie algebras to the multi-body problem.

Connections between robot motion and control theory can be found in Chow's theorem: for systems of the form $\dot{x} = \sum_{i=1}^{m} g_i(x)u_i$, where the g_i are smooth and the u_i are the control inputs, if the closure under Lie bracketing of Δ, the subspace of the tangent space spanned by the g_i, is equal to the tangent space at all x in a neighborhood of x_0, then the problem is small-time locally controllable at x_0, i.e. any point in a neighborhood of x_0 can be reached in arbitrarily small amounts of time by remaining in the neighborhood. Intuitively this theorem states that a sufficient condition for controllability is that any differential motion is obtained by some high-order Lie bracket.

It has been shown that car-like systems with n trailers are controllable and bases spanning the control Lie algebras have been generated. Let us show how all this works for a truck with a trailer (multi-body system with $n = 1$). The equations of this 2-body system are:

$$\begin{cases} \dot{x_0} &= \alpha \cos\theta_0 \\ \dot{y_0} &= \alpha \sin\theta_0 \\ \dot{\theta_0} &= \beta \\ \dot{\theta_1} &= \frac{\alpha}{d_1}\sin(\theta_0 - \theta_1) \end{cases}$$

This can be rewritten as:

$$\begin{pmatrix} \dot{x_0} \\ \dot{y_0} \\ \dot{\theta_0} \\ \dot{\theta_1} \end{pmatrix} = \begin{pmatrix} \cos\theta_0 \\ \sin\theta_0 \\ 0 \\ \frac{1}{d_1}\sin(\theta_0 - \theta_1) \end{pmatrix} \alpha + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \beta = g_1\alpha + g_2\beta$$

Let us compute the Lie bracket $[g_1, g_2]$ (let us recall that if X is a vector field with coordinates x_i, then DX is in $\mathcal{L}(\mathcal{E}, \mathcal{L}(\mathcal{E}, \mathcal{E}))$ and can be seen as a matrix with coordinates $\frac{\partial x_i}{\partial x_j}$):

$$[g_1, g_2]^\top = (\ \sin\theta_0 \quad -\cos\theta_0 \quad 0 \quad -\tfrac{1}{d_1}\cos(\theta_0 - \theta_1)\ )$$

[1] Usually only smooth vector fields are considered in applications but the definition holds for a C^r map

If we compute then $[g_1,[g_1,g_2]]$, we have $[g_1,[g_1,g_2]]^\top = (\ 0\ \ 0\ \ 0\ \ -\frac{1}{{d_1}^2}\ )$. It is now easy to check that $\mathbb{R}^4$ is spanned by $g_1, g_2, [g_1,g_2], [g_1,[g_1,g_2]]$ (the determinant of these 4 vectors is equal to $-\frac{1}{{d_1}^2}$). We conclude, using Chow's theorem, that the 1-body system is controllable. This is really comforting if we consider the number of trucks on highways ... The last Lie bracket with the nested in Lie bracket tells us something interesting about infinitesimal motions of this 1-body system: as g_1 corresponds to a straight motion and g_2 to a right forward turn, $[g_1,[g_1,g_2]]$ corresponds to this complex maneuver: forward, forward left, backward, backward left, backward, backward left, forward, forward right. This maneuver allows to reorient the trailer while keeping the orientation of the truck invariant. We will see in the next section some other maneuvers that translate a car truck or reorient it; a combination of all these maneuvers leads to a way to control a 1-body system in a complex environment; this is however highly impractical as many infinitesimal motions are needed to produce an appreciable motion, but it is a nice application of Chow's theorem.

The system of differential equations that models a n-body system is part of a large family of systems, which are called *chained systems*. Based on the Lie algebras formalism, algorithms have been developed which use either integrally-related sinusoids [Mur90] or high-frequency sinusoids [TLM+92] in order to control such systems.

2.1.2 Tracking trajectories

Once a feasible path is found, perhaps one that was generated by an open-loop path planner, and a sequence of inputs generating this path is known, there are linear time-varying feedback laws which locally exponentially stabilize a large class of systems ($\dot{x} = f(x) + g(x)u$) to the desired trajectory [WTS+92]. This approach starts from a linearization of the system about the nominal trajectory and assumes then the linear time-varying system thus obtained is uniformly completely controllable.

It has been applied to car-like robots and is another way to solve the control problem.

2.2 Rule-based incremental control

Many systems lack a mathematical model or are ill-modeled; however human experts can control them rather well. The knowledge generated by these experts seems to be stored in a declarative (if-then rules), symbolic way (no exact values on parameters are considered, but rather ranges) and the control actions are basically incremental (at each time, an input is either incremented, decremented by a given amount or left untouched).

Rule-based incremental control has been first introduced in [ZFG84, BFHZ85] as a concept and an experimental realization and in [FL89, Fou90] almost under its current form; theoretical results are stated concerning stability in [LZ90b, Luz91]. This type of control relies on

input/output relationships and is well adapted to robots, where sensors give informations which have to be treated quickly in order to perform an action if necessary. Experimental robustness has been shown and the sensors may be programmed so as to change their symbolic interpretation if necessary (sudden defect of one of the sensors or new knowledge given to one of them) [BF89]. A theoretical frame has been given for symbolic sensors and symbolic reasoning [LZ91] allowing some precomputing in the sensors in order to speed-up the control process. Furthermore, as we will see in detail in the next part, some learning techniques may be used to write such rule-based controllers and CANDIDE is a learning program which writes such controllers starting from no a priori knowledge of the system to control [Bur88, BLZ89b, BLZ89a, LZ89, BLZ89c, LZ90a].

We give the general form of rule-based incremental controllers in the next paragraphs, state some theoretical results and discuss their use in robot motion and parking.

2.2.1 Theoretical results

A rule-based incremental controller is a finite set of rules which have the following form:

If *output in this range*

then *increase, decrease or leave the input*

This is obviously discrete control and at each sampling time, the left part (the condition) of every rule is evaluated; once this is done for all rules, one of the rules with satisfactory condition is fired (it may be the first found rule, or the last, or if a weight is attached to each rule, it can be the most promising ...) and the corresponding right part (the action) is executed.

The input u_k at time k is related to the input u_{k-1} at time $k-1$ by the equation:

$$\begin{cases} u_k &= u_{k-1} + \epsilon_k^0 u_k^1 + \epsilon_k' \Delta' \\ u_k^1 &= u_{k-1}^1 + \epsilon_k^1 u_k^2 \\ \vdots & \\ u_k^{n-1} &= u_{k-1}^{n-1} + \epsilon_k^{n-1} u_k^n \\ u_k^n &= \Delta \end{cases}$$

where: $\forall i \in \{0, \cdots, n-1\}, \forall k \in \mathbf{N}^*, \epsilon_k^i \in \{-1, 0, 1\}, \epsilon_k' \in \{-m, \cdots, m\}$, and Δ, Δ' are two positive real numbers. The coefficients ϵ_k^i and ϵ_k' will be called *signs* and the set of their values at time k is the *sign policy*. Such control laws are called (n, m) incremental control laws.

This equation may be rewritten as: $u_k = u_{k-1} + v_k + \epsilon_k \Delta'$. The real number Δ' is a given increment and the integer ϵ_k' (bounded by m) tells the amount of the action at time k. This part of the equation models the "increase, decrease or leave the input". The term v_k allows more elaborate actions as: I know my input is not enough and I cannot increase it enough, so let us increase the increment too. As can easily be seen, v_k is solution of a n-th order recurrent system and the integer ϵ_k^i tells us whether the i-th order increment has to be increased, decreased or cancelled.

The strength of such formulation appears when one considers for instance a maximal sign policy, i.e. all the various ϵ_k^i and the ϵ_k' take their maximum value (respectively $+1$ and $+m$): the ϵ_k' introduce a linear behavior for the input while the ϵ_k^i generate an exponential behavior. This is very interesting when considering for instance linear systems: the exponential part of the control law may be used to follow the dynamics of the system, while the linear part allows some finer tuning. The next propositions are the justification of this intuitive informal remark.

Let us consider a stationary discrete linear SISO system given by the equation (u_k is the input, y_k is the output, X_k is the state vector, k is time; A, B, C are matrices):

$$\begin{cases} X_{k+1} &= AX_k + Bu_k \\ y_k &= CX_k \end{cases}$$

We have the next following propositions (for detailed proof, see [Luz91]):

Proposition 1 *Let us consider a linear system with* $1 \leq \| A \| \leq 2m+1$. *Then:*

$$\exists M_1, M_2 \ \forall k \ \exists(\epsilon_i') \ \forall i \leq k \quad \| X_i \| \leq M_1 + M_2 i$$

For a naturally divergent system, it is always possible to find a sign policy such that the divergence of the system is in the worst case linear.

Proposition 2 *Let us consider a linear system with* $1 \leq \| A \| \leq 2m+1$. *Then:*

$$\exists M_1, M_2, M_3 \ \forall k \ \exists(\epsilon_i') \begin{cases} \forall i < k \quad \| X_i \| \leq M_1 + M_2 i \\ \| X_k \| \leq M_3 \end{cases}$$

Furthermore the system may be brought back to the origin on any finite time interval.

Proposition 3 *Let us consider a linear system with* $1 \leq \| A \| < 2m+1$. *Then:*

$$\exists M_1, M_2 \ \forall k, \alpha \ \exists(\epsilon_i') \begin{cases} \forall i < k \quad \| X_i \| \leq M_1 + M_2 i \\ \| X_0 \| < \eta \Longrightarrow \| X_k \| < \alpha \end{cases}$$

In fact the system can be brought arbitrarily close to the origin. As can be seen in [Luz91], the proofs of all these propositions show that the sign policy only depends on the initial state and k.

Proposition 4 *Let us now consider a linear system with* $\| A \| < 1$. *Then for* i *smaller or equal to* n, *a reference signal of form* $k \mapsto k^i$ *(*k *is time) may be followed with null asymptotic error by a* $(n, 0)$ *control law (i.e. all the* ϵ'_k *are null)*

For naturally stable systems, this proposition implies for instance that a step or a ramp reference signal may be followed asymptotically by a $(1,0)$ incremental control law, which is the easiest law: the only admissible actions are either to increase, to decrease by an increment or to leave unchanged the last input.

The next result will concern tracking trajectories for a stationary discrete linear SISO system. Let us assume we have a desired trajectory $(X_k^{(d)})_k$ and some sequence of inputs $(u_k^{(d)})_k$ that yield this trajectory, i.e. $X_{k+1}^{(d)} = AX_k^{(d)} + Bu_k^{(d)}$. Let ϵ be a positive real. Tracking the desired trajectory (with error uniformly bounded by ϵ) means finding an incremental control law such that, if $X_{k+1} = AX_k + Bu_k$, then:

$$\| X_k - X_k^{(d)} \| \leq \epsilon \Longrightarrow \| X_{k+1} - X_{k+1}^{(d)} \| \leq \epsilon$$

We will look for a $(0, m)$ incremental control law, therefore $u_k - u_{k-1}$ is bounded by $(2m+1)\Delta'$; in order to track the desired reference, we will actually try to stay close to the sequence of inputs that yields the desired trajectory. It appears then that we can only consider systems where $u_k^{(d)} - u_{k-1}^{(d)}$ is bounded too.

Proposition 5 *For a linear SISO system* $(X_{k+1} = AX_k + Bu_k)$, *a given desired trajectory* $(X_k^{(d)})_{k \in \mathbb{N}}$ *and a sequence of inputs* $(u_k^{(d)})_{k \in \mathbb{N}}$ *yielding this trajectory* $(X_{k+1}^{(d)} = AX_k^{(d)} + bu_k^{(d)})$ *such that* $(u_k^{(d)} - u_{k-1}^{(d)})_{k \in \mathbb{N}}$ *is bounded, then, for any positive real* ϵ, *there exists an incremental control law tracking the trajectory with an error smaller than* ϵ.

Proof: Let us write $[x]$ for the integral part of x. Assume the linear system is under controllable canonical form (the state vector is n-dimensional):

$$X_{k+1} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix} X_k + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u_k$$

Let us define v_k and u_k by (Δ is a non null positive real):

$$\begin{cases} v_k &= [\frac{-(B^\top B)^{-1} B^\top A(X_k - X_k^{(d)}) + u_k^{(d)}}{\Delta}]\Delta + (B^\top B)^{-1} B^\top A(X_k - X_k^{(d)}) - u_k^{(d)} \\ u_k &= -(B^\top B)^{-1} B^\top A(X_k - X_k^{(d)}) + u_k^{(d)} + v_k \end{cases}$$

We notice that $|v_k| \leq \Delta$. A straightforward computation yields:

$$X_{k+1} - X_{k+1}^{(d)} = J(X_k - X_k^{(d)}) + Bv_k$$

where J is a $n \times n$ Jordan matrix[2]. We conclude that for any k:

$$\begin{aligned} \| X_{k+n} - X_{k+n}^{(d)} \| &= \textstyle\sum_{i=1}^{n} J^{n-i} B v_{k+i} \\ &\leq (\textstyle\sum_{i=1}^{n} \| J^{n-i} \|) \| B \| \Delta \end{aligned}$$

Take Δ such that $(\sum_{i=1}^{n} \| J^{n-i} \|) \| B \| \Delta < \epsilon$ and the error on the trajectory is smaller than ϵ. Let us now verify that (u_k) is an incremental control law:

$$\begin{aligned} u_k = u_{k-1} + \\ \{-(B^\top B)^{-1} B^\top A(X_k - X_k^{(d)}) + u_k^{(d)} + v_k\} + \{(B^\top B)^{-1} B^\top A(X_{k-1} - X_{k-1}^{(d)}) - u_{k-1}^{(d)} - v_{k-1}\} \end{aligned}$$

Both terms inside braces are, by construction, multiples of Δ. Furthermore:

$$|u_k - u_{k-1}| \leq |u_k^{(d)} - u_{k-1}^{(d)}| + |v_k| + |v_{k-1}| + \| (B^\top B)^{-1} B^\top A \| (\| X_k - X_k^{(d)} \| + \| X_{k-1} - X_{k-1}^{(d)} \|)$$

which is uniformly bounded in k. There exists thus an integer m such that for all k, there exists an integer ϵ_k with $|\epsilon_k| \leq m$ verifying $u_k = u_{k-1} + \epsilon_k \Delta$. □

If we take $u_k^{(d)} = 0$ and $X_k^{(d)}$ equal to the null vector for all k, we have the next proposition.

Proposition 6 *Let us consider a linear SISO system. Then for all ϵ, there exists a state feedback incremental control law such that for the controlled system:*

$$\exists M \ \ \forall k \ \ \| X_k \| \leq M$$

If the increment Δ is chosen small enough, we have:

$$\| X_k \| \leq \epsilon$$

In the first propositions 1, 2, 3, we had found open loop incremental control laws which yielded stability or asymptotic stability if the increment Δ was chosen small enough, on any finite horizon; this last proposition guarantees stability and asymptotic stability for a closed loop incremental control law.

[2] $J = (\delta_{i,j}^{i,i+1})_{i,j \in [1,n]}$, where δ is the Kronecker symbol

2.2.2 Car-like robot with trailers: full controllability

Let us return to the car-like robot as described earlier, without trailers. This robot is modeled as a rectangle in $\mathcal{W} = \mathbb{R}^2$; the configuration space is $\mathbb{R}^2 \times S^1$. A configuration is represented as (x, y, θ) where (x, y) are the coordinates of the midpoint between the two rear-wheels and θ is the angle between the x-axis of the reference frame. Furthermore the robot is not sliding. The equations are the same as in the case of the multi-body problem.

Let us work now in configuration space: the car is represented as a point and the obstacles are mapped into C-obstacles. We will now show that any possibly non-feasible free path between two configurations which are in the same connected component of the free space can be transformed into a feasible path that can be found by an rule-based incremental controller. This proof is constructive and uses arguments similar to Laumond's proof of full controllability of a car-like robot without constraints on the control inputs [LTJ90].

Let us represent a configuration (x, y, θ) by a vector with origin at point (x, y) and making an angle θ with the x-axis. We define two basic maneuvers, which combine turns with minimal turning radius (i.e. maximal steering angle) and straight segments.

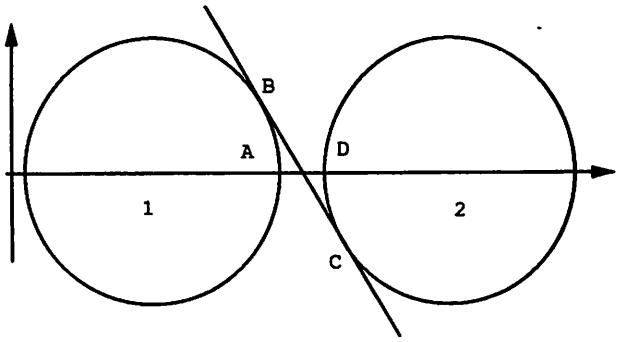

Figure 2.1: **Maneuver 1**

Maneuver 1 consists of a left turn, followed by a straight reversal and a right turn; the first turn brings the car from (x_0, y_0, θ_0) to $(x_1, y_1, \theta_0 + \theta)$; the reversal brings it to $(x_2, y_2, \theta_0 + \theta)$; the last turn brings it finally to (x_3, y_3, θ_0). This maneuver allows the robot to perform a sidewise motion. The final and the initial orientations are the same.

Figure 2.1 shows the key points of this maneuver: A is the starting point, the car turns to the left and reaches B, then it backs up to C and finally turns to the right and reaches D. The initial and final orientations are perpendicular to the straight line AD; the turns are performed on a circle with radius $\rho_{\min}$ and each turn corresponds to an angle θ. We have: $\widehat{AB}=\widehat{CD}= \rho_{\min}\,\theta$,

$BC = 2\rho_{\min} \tan\theta$ and $AD = 2\rho_{\min}\,(\frac{1}{\cos\theta} - 1)$. This maneuver is obviously included in a circle which has BC as a diameter.

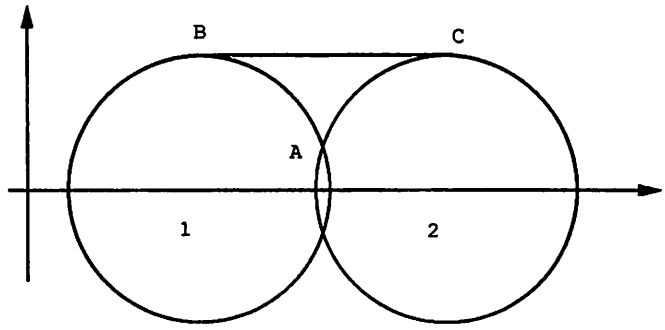

Figure 2.2: **Maneuver 2**

Maneuver 2 consists of a left turn, followed by a straight reversal and another left turn. The first turn brings the car from (x_0, y_0, θ_0) to $(x_1, y_1, \theta_0+\theta)$; the reversal brings it to $(x_2, y_2, \theta_0+\theta)$; the last turn brings it finally to $(x_0, y_0, \theta_0 + 2\theta)$. This maneuver allows the car to perform a rotation around the midpoint of its rear axis.

Figure 2.2 shows the key points of this maneuver: A is the starting point, the car turns to the left and reaches B, then it backs up to C and finally turns to the left and reaches A. The initial and final orientations are tangent respectively to circle 1 and circle 2. The turns are performed on a circle with radius $\rho_{\min}$ and each turn corresponds to an angle θ. The difference between the initial and the final orientations is 2θ. We have: $\overset{\frown}{AB}=\overset{\frown}{CA}= \rho_{\min}\,\theta$, $BC = 2\rho_{\min} \sin\theta$ and $AB = 2\rho_{\min} \sin\frac{\theta}{2}$. This maneuver is obviously included in a circle which has A as center and AB as a radius.

These maneuvers may be performed by a rule-based incremental controller: for instance, the left turn of the first maneuver is completed by: `if` *the current angle of the car is in* $[\theta_0, \theta_0+\theta]$ `then` *increase* v *and increase* ϕ *by twice the increment.* The reversal straight line is then completed by decreasing v and decreasing ϕ by once the increment. The speed increment is $2v_{\max}$ and the steering angle increment is $\phi_{\max}$. With these increments, the possible values for (v, ϕ) are $\{-v_{\max}, -v_{\max}\} \times \{-\phi_{\max}, 0, \phi_{\max}\}$.

Although both maneuvers can be done with a rule-based incremental controller, one has to keep in mind that this type of control is a discrete control: a rule is fired at each time sampling. This means that the turns and the back ups are multiples of a smallest turn and a smallest back up. It means too that the car may not exactly reach D in maneuver 1 and may not go back exactly to A in maneuver 1. Let us see more precisely the drift error. In maneuver 1, the car

will make a $\delta\theta$ error while turning with angle θ; it will make an error δl while backing up and will then make the same error $\delta\theta$ during the last turn (although the car does not reach C and is not even on the straight line BC because of the $\delta\theta$ error, we do not want it to turn more or less in order to try to cancel these errors; the last turn is assumed to be with same magnitude as the first turn, in order to keep sort of symmetry in the whole maneuver). It is easy to see that the different values $\widehat{AB}$, $\widehat{CD}= \rho_{\min}\,\theta$, BC and AD behave continuously with θ and converge to 0 when θ converges to 0. Let us recall from the equations of the car-like robot (the speed of the rear point R is taken to be constant in norm to $v_{\max}$) that:

$$\rho_{\min} = \frac{L}{\tan\phi_{\max}} \quad \text{and} \quad \dot{\theta} = v\frac{\sin\phi_{\max}}{L} = v\frac{\cos\phi_{\max}}{\rho_{\min}}$$

This shows that the angular velocity behaves continuously with $v_{\max}$ and converges to 0 when $v_{\max}$ converges to 0. Of course the error δl is bounded by $v_{\max}\,T$ where T is the sampling period, thus it behaves continuously with $v_{\max}$ too. The same remarks hold for the second maneuver.

All this shows that if the speed $v_{\max}$ is taken small enough, maneuver 1 brings the car from A to a point in any arbitrarily neighborhood of D with same initial and final orientations and maneuver 2 brings the car from A to a point in any arbitrarily neighborhood of A with a difference between the final and the initial orientations in any arbitrarily neighborhood of θ. Furthermore any maneuver can be performed in an arbitrarily small circle.

We can now proof the **full controllability** of the car-like robot.

Proposition 7 *Let* $\mathbf{q}$ *and* $\mathbf{q}'$ *be two configurations in the same component of* $\mathcal{C}_{\text{free}}$ *. For any neighborhood of* $\mathbf{q}'$*, there exists a feasible free path between* $\mathbf{q}$ *and a configuration in this neighborhood, that can be obtained by a rule-based incremental controller.*

Proof: For (x, y, θ) in $\mathcal{C}_{\text{free}}$ we call $C(x, y, \theta, \eta, \epsilon) = D_\eta \times (\theta - \epsilon, \theta + \epsilon)$ the open cylinder with radius η, of height 2ϵ parallel to the θ-axis, centered at (x, y, θ). Let C be such a cylinder included in the free space $\mathcal{C}_{\text{free}}$.

• We state that if q_1 is any point of C, a feasible path can be found starting at (x, y, θ) and ending in any arbitrarily small neighborhood of q_1. Proof: it is obvious that we can take $x = y = \theta = 0$ (by changing the reference frame in an isometric way, i.e. taking a new set of local coordinates [Ave83]). Let us represent a configuration (x', y', θ') as an arrow at point (x', y') with an angle θ' with the x-axis for sake of simplicity. If (x', y', θ') is in $C(0, 0, 0, \eta, \epsilon)$, a feasible path can be found that starts from the origin and reaches any neighborhood of (x', y', θ'): make a finite number of maneuvers 1 in order to join the origin and $(0, y', 0)$ (side-ways motion along

the y-axis), then do a straight line between $(0, y', 0)$ and $(x', y', 0)$, then make a finite number of maneuvers 2 in order to reach (x', y', θ') (reorienting the car). As we saw previously, the path given by the incremental rule-based controller will drift, but this drift behaves continuously with $v_{\max}$ and the speed can be adjusted in order to reach any neighborhood of (x', y', θ'); furthermore by choosing $v_{\max}$ small enough, the path will remain in C.

• Let us proof now the proposition. As both configurations $\mathbf{q}$ and $\mathbf{q}'$ are in the same connected component of $\mathcal{C}_{\text{free}}$, there exists a free path τ between them. Since τ is a compact subset of $\mathcal{C}_{\text{free}}$, it can be covered by a finite number of open cylinders like C (it is obvious that the family of all cylinders C is a generating family of neighborhoods for configuration space). Hence, τ can be transformed into a feasible path by introducing a finite number of maneuvers 1 and maneuvers 2 contained in these cylinders (these cylinders intersect τ at a finite number of points; build a sequence (q_p) with $q_0 = \mathbf{q}$ and $q_n = \mathbf{q}'$, and q_i is the furthest point on τ which is in the cylinder centered at q_{i-1}; join then two successive elements of the sequence by using the statement given at the beginning of this proof). As the whole number of maneuvers is finite, the final eventual drift can be chosen arbitrarily small by choosing adequately $v_{\max}$. □

We based the proof on the fact that all the distances occurring in the maneuvers were behaving continuously on $v_{\max}$. If we assume now that the speed $v_{\max}$ is given (like $\phi_{\max}$), we can then notice that all the distances behave continuously with T too, which is the sampling period (in fact, the errors $\delta\theta$ and δl behave like $\dot{\theta}T$ and $v_{\max}\,T$). We see then that the path needed in the previous proof can be approximated by a feasible path τ_T and the same proof can be used, mutatis mutandis, for the following proposition.

Proposition 8 *Let* $\mathbf{q}$ *and* $\mathbf{q}'$ *be two configurations in the same component of* $\mathcal{C}_{\text{free}}$ *and let* T *be the time period of the discrete sampling. If* T *is small enough, there exists a feasible free path between* $\mathbf{q}$ *and* $\mathbf{q}'$ *that can be obtained by a rule-based incremental controller.*

We have proved that a rule-based incremental controller could bring a car-like robot from any initial configuration to any final configuration if these configurations are in the same connected component of $\mathcal{C}_{\text{free}}$. Of course this constructive proof is not efficient in practice because of the very high number of maneuvers involved. This is why we have to look for rule-based incremental controllers which will lead to much less maneuvers.

2.2.3 Car parking

We give in this section two rule-based incremental controllers which have been used to park a car.

Let us give a fixed reference frame $(\vec{i}, \vec{j})$ and a frame attached to the car $(\vec{i}_1, \vec{j}_1)$ where $\vec{i}_1$ has the same orientation as the car. The car starts from a given distance of the parking place, oriented parallelly to the parking place. The steering angle is bounded by `phimax`. The length of the car place is L and its width is $2e$; the parking place is a box $[0, b] \times [-a, a]$.

In the controllers, there are several mathematical expressions: $Y_r - e\cos\theta$, $Y_r - X_r \tan\theta$ and $Y_f + (b - X_f)\tan\theta$; they correspond respectively to the y coordinate of the rear right corner of the car, to the y coordinate of the intersection between the straight line directed by $\vec{i}_1$) and the rear of the parking place (straight line $x = 0$), and finally to the y coordinate of the intersection between the straight line directed by $\vec{i}_1$ and the front of the parking place (straight line $x = b$).

The controller in one maneuver reverses the car till its rear arrives at the height of the parking place (rule $r0$), then the car turns right till its right rear is inside the parking place (rule $r1$), and then the car makes a left turn (rule $r2$). We give here the controller in Common Lisp syntax:

```
(
 (r0 (> Xr b) (setq v(min v(- v)))2)
 (r1 (and(< v 0)(> (- Yr(* e(cos theta))) a))
           (setq phi(in-bounds phimax(- phi incr)))1)
 (r2 (and(< v 0)(<= (- Yr(* e(cos theta))) a))
           (setq phi(in-bounds phimax(+ phi incr)))1)
 (dummy T T 0)
)
```

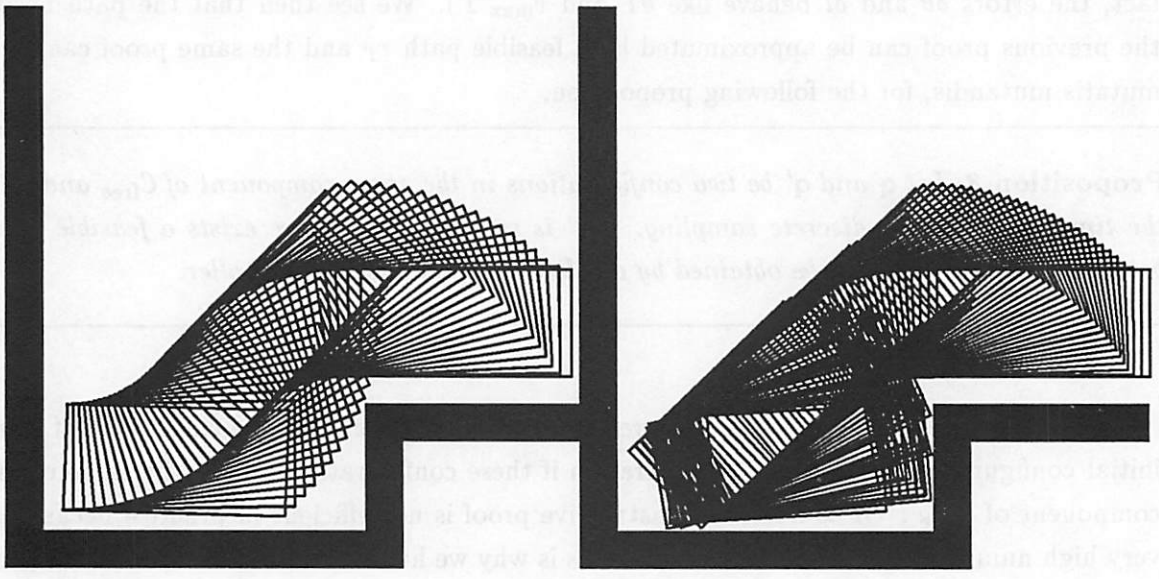

Figure 2.3: **Parking: one maneuver and several maneuvers**

The controller in several maneuvers is an improved version of the previous controller, as sometimes several maneuvers are needed. Several strategies have to be chained: enter the parking

place and orient the car in the right way. We give its Common Lisp expression and an explanation of the different rules.

$r0$ reverses the car till the rear of the car arrives at the height of the parking place. $r1$ et $r2$ are fired during the first maneuver: the right rear of the car has to reach the right rear part of the parking place (the goal is to have most of the car in the parking place) $r3$, $r4$, $r5$ and $r6$ reorient the car inside the parking place: $r3$ and $r4$ align the rear of the car with the rear middle of the parking place, $r5$ and $r6$ align the front of the car with the front middle of the parking place. $r7$ is fired when the distance between the car and the obstacles is smaller than a given range (for instance $v\ dt$ — where dt is the time sampling — which is the maximal distance the car makes in a time period) and this rules inverts all the commands (turn in the other direction and go in the other direction), and this action is maintained as long as the car stays too near from the obstacle. $r8$ and $r9$ bring the car back to the parking place in case some odd succession of rules had generated a trajectory leaving the neighborhood of the parking place. The task is finished as soon as the four wheels of the car are inside the parking place.

```
 ;; initializing some variables
  (setq n-dummy 0 init t end () action ())
(
  (r0 (>= Xr b) (setq d-last d-min v(min v(- v))) 0.5)
  (r1 (and init (<= v 0)(< Xr b)(>=(- Yr(*(tan theta)Xr))(- a)))
      (setq d-last d-min action '(setq phi(in-bounds phimax(- phi incr)))) 0.5)
  (r2 (and init (<= v 0)(< Xr b)(<(- Yr(*(tan theta)Xr))(- a)))
      (setq d-last d-min action '(setq phi(in-bounds phimax(+ phi incr)))) 0.5)
  (r3 (and end(<= v 0)(< Xr b)(>=(- Yr(*(tan theta)Xr))0))
      (setq d-last d-min action '(setq phi(in-bounds phimax(- phi incr)))) 0.5)
  (r4 (and end(<= v 0)(< Xr b)(<(- Yr(*(tan theta)Xr))0))
      (setq d-last d-min action '(setq phi(in-bounds phimax(+ phi incr)))) 0.5)
  (r5 (and end(> v 0)(< Yf a)(< Xf b)(>=(+(*(tan theta)(- b Xf))Yf)0))
      (setq d-last d-min action '(setq phi(in-bounds phimax(- phi incr)))) 0.5)
  (r6 (and end(> v 0)(< Yf a)(< Xf b)(<(+(*(tan theta)(- b Xf))Yf)0))
      (setq d-last d-min action '(setq phi(in-bounds phimax(+ phi incr)))) 0.5)
  (r7 (or(< d-front d-min)(< d-left d-min)(< d-rear d-min)(< d-right d-min))
      (setq v (if(<= d d-last)(- v)v)
            init ()
            end (if(or(and(< Ya a)(< Yb a)(< Xa b)(< Xb b)(< v 0))
                      (and(< Yc a)(< Yd a)(< Xc b)(< Xd b)(> v 0))) t ())
            action
             (if(<= d d-last)
                (cond((equal action '(setq phi(in-bounds phimax(+ phi incr))))
                     '(setq phi (in-bounds phimax(- phi incr))))
                     ((equal action '(setq phi(in-bounds phimax(- phi incr))))
                     '(setq phi (in-bounds phimax(+ phi incr)))))
                action)
            d-last d
      ) 1)
  (r8 (and(>= v 0)(or(> Xa b)(> Xb b)))
      (setq d-last d-min v(- v)
            action '(setq phi(in-bounds phimax(- phi incr)))) 0.9)
  (r9 (and(>= v 0)(> Yf a))
      (setq d-last d-min action '(setq phi(in-bounds phimax(- phi incr)))) 0.8)
  (dummy T (progn
            (setq d-last d-min)
```

```
            (setq n-dummy(1+ n-dummy))
            (if(= n-dummy 100)(setq n-dummy 0 v(min v(- v)))T))
        0)
)
```

2.2.4 Parking of a car-like robot with 1 trailer

The equations of the 2-body car-like robot are:

$$\begin{cases} \dot{x}_0 &= v\cos\phi\cos\theta_0 \\ \dot{y}_0 &= v\cos\phi\sin\theta_0 \\ \dot{\theta}_0 &= \frac{1}{L}v\sin\phi \\ \dot{\theta}_1 &= \frac{1}{d_1}v\cos\phi\sin(\theta_0-\theta_1) \end{cases}$$

Let us recall (x_0, y_0) is the middle of the rear axis of the first body, θ_0 is its angle with a reference straight line and θ_1 the angle of the second body with the same reference straight line; ϕ is the steering angle, L the length of the first body, d_1 the distance between the midpoint of the rear axis of the first body and the midpoint of the rear axis of the second body, and v the speed of the midpoint of the front axis of the first body. We assume that $|v|$ is constant.

In all practical applications, ϕ is taken in the interval $]-\frac{\pi}{2}, \frac{\pi}{2}[$. If $\theta = \theta_1 - \theta_0$, it is obvious that we have, for non zero ϕ:

$$\dot{\theta} = -\dot{\theta}_0(1 + \frac{L}{d_1\tan\phi}\sin\theta) \tag{2.1}$$

This differential equation can be integrated for constant ϕ ($\phi \neq 0$) by using the following formulas:

$$\begin{aligned} \int_\alpha^\beta \frac{dx}{a+b\sin x} &= \left[\frac{1}{\sqrt{b^2-a^2}}\ln\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right]_\alpha^\beta && (\text{if } b^2 > a^2) \\ &= \left[\frac{2}{\sqrt{b^2-a^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{b^2-a^2}}\right]_\alpha^\beta && (\text{if } b^2 < a^2) \\ \int_\alpha^\beta \frac{dx}{1+\epsilon\sin x} &= \left[-\frac{\epsilon-\sin x}{\cos x}\right]_\alpha^\beta && (\text{if } \epsilon \in \{-1,+1\}) \end{aligned}$$

If $\phi = 0$, we go back to the original equation yielding θ_1, and as θ_0 is constant, we have: $\dot{\theta}_1 = \frac{1}{d_1}v\sin(\theta_0-\theta_1)$ which is easily integrated between time t_a and time t_b into:

$$\tan\frac{\theta_1(t_b)-\theta_0}{2} = \exp^{-\frac{v}{d_1}(t_b-t_a)}\tan\frac{\theta_1(t_a)-\theta_0}{2}$$

A qualitative study of this equation shows:

- $\theta_1 = \theta_0$ is a stable equilibrium for $v > 0$
- $\theta_1 = \theta_0$ is an unstable equilibrium for $v < 0$
- $\theta_1 - \theta_0 = \pi$ is a stable equilibrium for $v < 0$

Furthermore the rates of convergence towards the stable equilibria is exponential in time (because of the form of the general solution and $\arctan x \overset{x \to 0}{\sim} x$) and the smaller d_1 the faster the convergence.

If $\phi \neq 0$ and ϕ constant, we will undertake a qualitative study of equation 2.1. There are two different cases:

• $|\frac{L}{d_1 \tan\phi}| \leq 1$

During a forward or a backward maneuver, $\dot{\theta}$ has always the same sign (as ϕ and v are constant, $\dot{\theta}_0$ is constant too). If $|\frac{L}{d_1 \tan\phi}| = 1$, $\dot{\theta}$ may be null for some values of θ but is non null in some open neighborhood of these values and has a constant sign on this neighborhood. It is thus obvious that there are no equilibria for $|\frac{L}{d_1 \tan\phi}| < 1$, and for $|\frac{L}{d_1 \tan\phi}| = 1$ (resp. -1) the equilibrium $\theta = -\frac{\pi}{2}$ (resp. $\theta = \frac{\pi}{2}$) is unstable. The behavior around $\theta = 0$ is interesting (let us call $s(x)$ the sign of x): as $s(\dot{\theta}) = s(-\dot{\theta}_0) = s(-v\phi)$, we have:

- during a right forward turn, θ increases
- during a left forward turn, θ decreases
- during a right backward turn, θ decreases
- during a left backward turn, θ increases

This yields a way to "control" θ around 0 by backing up while turning either to the right or to the left: if $\theta > 0$ (resp. $\theta < 0$), a right (resp. left) backward turn brings θ to negative (resp. positive) values. Of course, with forward turns the same "control" is also possible.

• $|\frac{L}{d_1 \tan\phi}| > 1$

In this case, $\dot{\theta}$ will not have a constant sign and when studying the behavior around $\theta = 0$, this will break the symmetry between forward and backward movements, as we will see. The solution of 2.1 shows there is a stable equilibrium for any maneuver and it is reached exponentially in time. For $v > 0$, the equilibrium for either a right or a left turn is such that $|\theta_{eq}| < \frac{\pi}{2}$. But for $v < 0$, the equilibrium is such that body 1 is almost folded on body 0; as we restrict our study to angles θ smaller than $\frac{\pi}{2}$ in norm (the trailer must not crash into the truck!), we see that there is no stable position in this case for $v < 0$. The behavior of θ around 0 is more complex in this case:

- if $|\theta| < |\arcsin(\frac{d_1 \tan\phi}{L})|$ we have $s(\dot{\theta}) = s(\dot{\theta}_0) = s(-v\phi)$ and the behavior is the same as in the other case
- if $|\theta| > |\arcsin(\frac{d_1 \tan\phi}{L})|$ we have $s(\dot{\theta}) = s(-\dot{\theta}_0\phi\theta) = s(-v\theta)$, which means that backward maneuvers do not change the sign of θ (in fact, they increase the absolute value of θ till the corresponding equilibrium is reached).

For small θ it is then possible to "control" θ around 0 by backing up and turning, but not for large θ.

This qualitative study of the car-like robot with 1 trailer has shown that the behavior of the trailer depends on the ratio $\frac{d_1 \tan\phi}{L}$. For constant ϕ and L, this means that a truck with a long trailer and a car with a small trailer cannot be driven in the same way: the truck can reach $\theta = 0$ from any initial position by backing up and turning in the appropriate way, while the car can do it only for some initial positions (θ must be small). On the other hand, when the car rides forward, it can turn to the left or to the right as long as desired, while the truck cannot (the trailer would crash into the truck).

We will now try to solve the parking problem for a truck with a trailer, using the qualitative knowledge we have just acquired.

We assume $|\frac{L}{d_1 \tan\phi}| < 1$. We will now study the following maneuver: a forward right turn and then a forward left turn, both turns lasting the same time. In order to simplify the next equations, we will now consider only absolute values for $\dot{\theta}_0$ and ϕ. This means a right turn is characterized by $-\phi$ and a left turn by ϕ; in the same way, a forward left turn corresponds to $\dot{\theta}_0$ and a forward right turn to $-\dot{\theta}_0$. We rewrite equation 2.1 as:

$$\begin{cases} d\theta &= (1 - \frac{L}{d_1 \tan\phi} \sin\theta)\dot{\theta}_0 \, dt & \text{(forward right turn)} \\ d\theta &= -(1 - \frac{L}{d_1 \tan\phi} \sin\theta)\dot{\theta}_0 \, dt & \text{(forward left turn)} \end{cases}$$

Integrating the first equation between time 0 and t yields: $f(\theta(t)) - f(\theta(0)) = \dot{\theta}t$ where:

$$f(\theta) = \frac{2}{\sqrt{1 - \frac{L^2}{d_1^2 \tan^2\phi}}} \arctan \frac{\tan\frac{\theta}{2} - \frac{L}{d_1 \tan\phi}}{\sqrt{1 - \frac{L^2}{d_1^2 \tan^2\phi}}}$$

Integrating the second equation between time t and $2t$ yields: $g(\theta(2t)) - g(\theta(t)) = -\dot{\theta}t$ where:

$$g(\theta) = \frac{2}{\sqrt{1 - \frac{L^2}{d_1^2 \tan^2\phi}}} \arctan \frac{\tan\frac{\theta}{2} + \frac{L}{d_1 \tan\phi}}{\sqrt{1 - \frac{L^2}{d_1^2 \tan^2\phi}}}$$

A straight use of trigonometric formulas transforms then $f(\theta(t)) - f(\theta(0)) = -(g(\theta(2t)) - g(\theta(t)))$ into:

$$(\tan\frac{\theta(2t)}{2} - \tan\frac{\theta(0)}{2})(1 + \tan^2\frac{\theta(t)}{2}) = \frac{2L}{d_1 \tan\phi}(\tan\frac{\theta(2t)}{2}\tan\frac{\theta(0)}{2} - \tan^2\frac{\theta(t)}{2})$$

But we have seen that for a forward turn to the right, θ is increasing and for a forward left turn, θ is decreasing; thus $\theta(0) < \theta(t)$ and $\theta(2t) < \theta(t)$. As we are dealing with angles inside the interval $]-\frac{\pi}{2}, \frac{\pi}{2}[$, the same inequalities hold for $\tan\theta(0)$, $\tan\theta(t)$ and $\tan\theta(2t)$. The right member of the previous equality is thus negative, which implies $\theta(2t) < \theta(0)$. As this maneuver consisting in

two turns with same length does not change the orientation of the truck, we conclude that the trailer rotates clockwise.

After this maneuver, the truck has its original orientation but it has advanced. Let us assume $\theta > 0$. In order to correct this advance, we can either backup or perform first a backward turn to the right during time t' and then a backward turn to the left during the same time t'. In both cases, it is easy to see that θ increases (in the first case, θ is initially positive, thus $\dot{\theta}_1 = \dot{\theta}$ which varies like $-v\theta$ is positive; in the second case, the equations are very similar to the previous study but this time θ first decreases and then increases); therefore, as θ_0 will be the same at the beginning and at the end of either maneuver, we conclude that the trailer rotates counterclockwise.

In the second case, the truck translates sideways and parking thus needs less maneuvers. However the trailer slowly leaves the parking place. On the contrary, backing up ensures the trailer remains in the parking place.

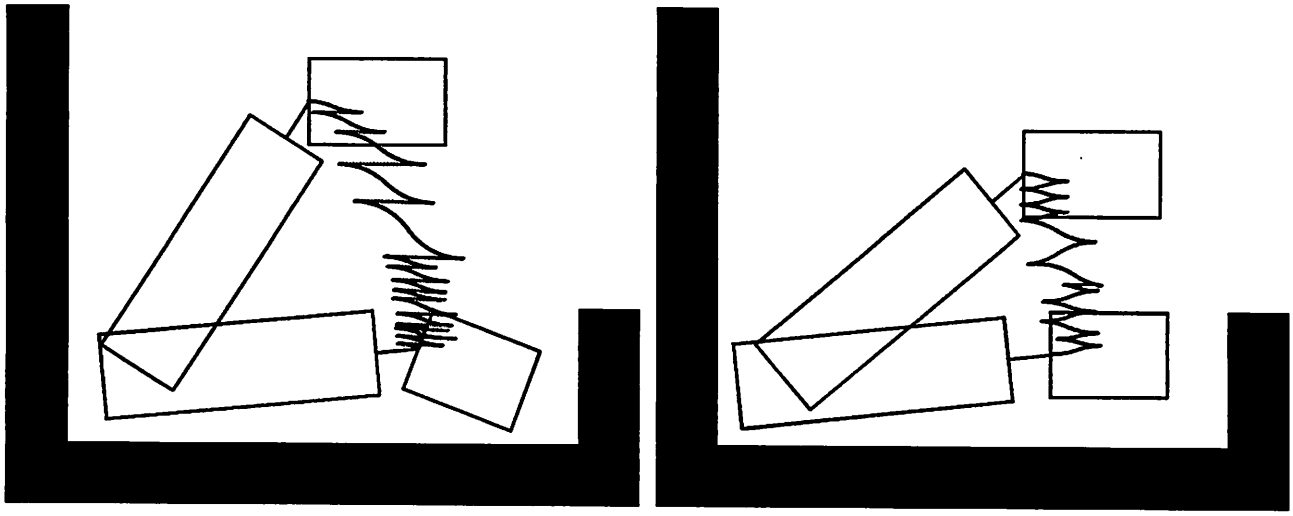

Figure 2.4: **Two ways of parking a truck with 1 trailer**

The previous maneuvers can obviously be performed by a rule-based incremental controller and their combination allows to park a truck with 1 trailer. On the figures the initial and the final positions are drawn as well as the position of (x_0, y_0) during the parking maneuver. The first two figures start from an intermediate position $(\theta_0 = 0, \theta_1 > 0)$ where the rear of the trailer is already in the rear corner of the parking place; the figure on the left uses forward right turns, forward left turns and backups, while the figure on the right uses forward right turns, forward left turns, backward right turns and backward left turns. It is easy to reach a position $(theta_0 = 0, \theta_1 > 0)$ with the rear of the trailer inside the rear corner of the parking place: starting from an initial

position parallel to the parking place, the truck first turns backward to the left during time t_1, then backs up, turns backward to the right during time t_1 and eventually goes on backing up. The whole parking process is shown in the next figure.

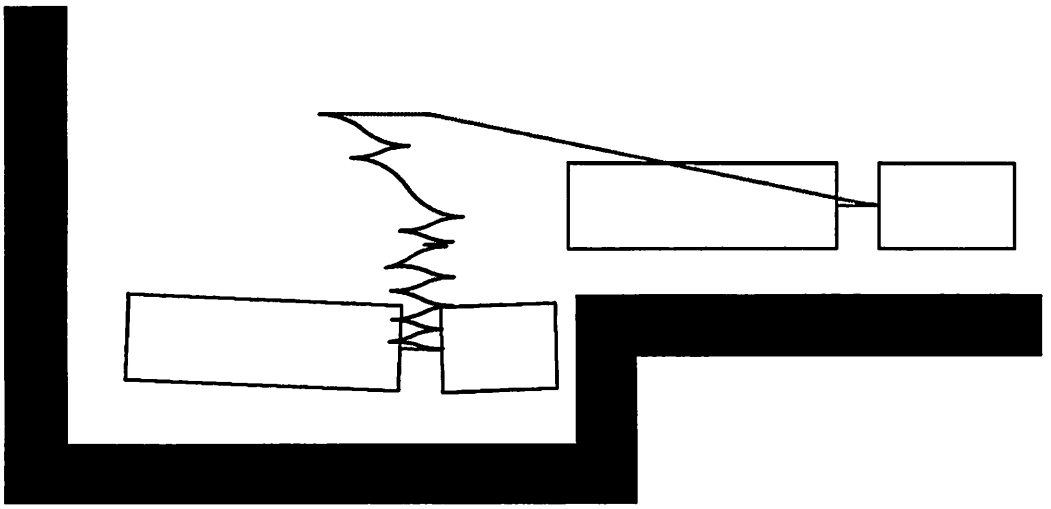

Figure 2.5: **Parking a truck with 1 trailer**

Chapter 3

Practice oriented techniques: towards learning

We now deal with another aspect of robot motion: instead of focusing on the theoretical proofs that guarantee the controllability of a robot, or that ensure the robot will be able to move amidst obstacles whatever the initial conditions, we will try to find some practical ways to solve the robot motion problem. By practical, we mean easy to implement on real-world systems, easy to find; instead of looking for theoretical often complex solutions, we will look for "natural" solutions. Of course, this is not – and cannot be – formalized, and therefore attracts many critics. These critics are only justified if practice oriented techniques claim their ability to solve the general problem of robot motion, unless they prove it of course.

In order to achieve some universality in solving the robot motion problem (what we could formalize by perturbation and model robustness, or stability), most of these techniques see in learning a way to palliate their lack of formal proof. But the word *learning* must be used with caution too, as we will see: the different techniques are not concerned with the same level of learning and this must be stated very clearly. Furthermore, learning is useful if there is something to learn and some hope to learn it! This leads us back to theory, but this time the subject is learnability and convergence of the learning process.

3.1 Neural networks

Let us begin with a quick introduction of neural networks: they are made of an interconnection of devices, called *neurons*, and local external inputs. The input/output characteristics of a neuron can be modeled by a continuous monotonically increasing function taking its values in $[0, 1]$, usually called a *sigmoidal function* (in order to have a saturating behavior instead of a mere one or zero behavior). The inputs to the neurons consist of weighted sums of the neuron

outputs. The goal is to choose the weights of the network to achieve a desired input/output relationship; this process is known as *training* or *learning*. One of the strengths of the neural networks relies in the flexibility of the connections between neurons: the "learning power" of a neural net, e.g. the functions it may learn, depend on the connections between neurons. Many such connection topologies may be considered; examples are fully interconnected networks, multi-layered networks, feedforward networks (the input of the i-th layer has the output of the $i-1$-th layer as a subset), networks with local feedback interconnections.

3.1.1 Truck backer-upper

The next section deals with the "truck backer-upper", a neural network controller steering a trailer truck while backing up (no forward movement allowed) to a loading dock [NW90].

For this application, the neurons used are *Adalines* and they are connected together to form a two-layer feedforward network. An Adaline has $n+1$ inputs (the n inputs of the network and one additional input set to $+1$ which adds a constant bias to the weighted sum) and a single output. The output equals the sum of all inputs multiplied by the weights and passed through a sigmoid (tanh in this case). The network is trained by *back propagation*, which aims at minimizing the mean-square error between the desired output and the actual one (this error is propagated from the last layer of neurons to the first layer, hence the name of the algorithm, and the weights are then updated by a proportion of the computed propagated error).

In order to solve the problem of the truck backer-upper, two stages are involved: first a network (25 Adalines on the first layer and 4 on the second) is trained in order to simulate the dynamics of the truck and its trailer, and then another network with a similar topology (25 Adalines on the first layer and 1 on the second) is trained to perform the task.

The first stage may be seen as identification: the inputs of the network are x_{trailer}, y_{trailer}, θ_{truck}, θ_{trailer} and the steering signal u_k; the outputs are the new state representing position and orientation of the truck and the trailer when the steering signal is applied to the previous state. The training process starts with the truck and the trailer in any configuration; the neural net is trained to predict the next state.

The second stage is the control learning part: the goal is to bring the trailer in front of the dock with a null angle, however the truck may have any final orientation. A lesson goes on for a given time or as soon as the truck or the trailer crashes into the dock; the error fed to the network is a combination of the various position and orientation errors. The input of the network is a state and the output a steering signal; this signal is given to the truck and trailer emulator in order to compute the resulting state; the necessity of the emulator is obvious: in order to learn something, the neural controller needs to compute an error which will then be propagated. But the error on the steering signal is not available, only the error in the final state

is available; the emulator is used to compute by back propagation this steering error which is then back propagated through the neural controller.

About 20000 backups were needed to train the neural controller and the experimental results are good: starting from non trivial initial states, a satisfactory path using only back-ups is found. Furthermore by changing the performance function (the error to minimize), some more complex goals can be achieved like backing up while minimizing control energy.

3.1.2 Mobile robots

In this section we see another neural network which is trained to control mobile robots moving in a confined space with obstacles while achieving a given goal [NSA90].

The mobile robots considered here have four wheels aligned in the same direction and two motors which can be used independently. The robot can move freely in direction and position; however there is a torsion limit constraining the right and left turns. Up to 12 sensors are placed on the robot: ultrasonic sensors, infrared sensors, tactile sensors and torsion limit sensors. The neural network used to control the robot is a combination of two networks called a *reason* and an *instinct* network. The reason network has 3 layers with 13 units on the input layer, 9 units on the hidden layer and 7 units on the output layer, while the instinct network has only 5 units on each of its three layers. Both are highly interconnected. The instinct network receives as input the right and the left torsion limit and two signals from the reason network, called excitatory and inhibitory signals. The five other outputs of both networks are controls for the motors: forward, backward, right, left motion as well as a buzzer signal.

The particularity of this architecture is its hierarchical structure: the reason network determines the correspondence between the sensory input and the behavior pattern (like "move forward when the infrared sensor on the head detects light") and the instinct network determines the correspondence between sensory input and some critical behavior patterns that should be taken over for a certain period of time; this instinct network is activated when torsion limit is reached or when there is an obstacle which has to be avoided. This decomposition into two networks decreases the number of training lessons and necessary connections.

Instead of using back propagation, another algorithm (*pseudo impedance control*) is used in order to avoid local minima. The learning is divided in three stages: in the first stage, there is a robot behavior simulator; in the second stage, the actual robot is used but it is always connected to the network. In the last stage, the network is stored in the robot in ROM and the learning is considered finished if the robot behaves well.

In order to test the methodology, several robots are used and divided in two categories: robots tracking other robots emitting infrared rays and robots that escape when they sense infrared rays. For the first class of robots, the reason network has been taught 62 patterns and the instinct

network 9 patterns; learning was completed in 500 lessons. The robots have proved to be able of determining their motion in response to the other robots. It is important to note that the patterns used for training are well chosen and already encode the action to perform in such or such situation; in this application, the neural nets are a convenient way to control the mobile robots and allow some distributed control.

3.1.3 Connectionist robot motion planning: MURPHY

This section describes MURPHY [Mel90], a robot-camera system associated with a connectionist architecture; the problem is to guide a multi-link arm to visual targets in a cluttered workspace. No *a priori* model of the arm kinematics and of the characteristics of the vision system is assumed.

Usually, when given a robot, one writes down kinematic equations which map joint angles into the workspace coordinates of the control points of the arm; then an inverse-kinematic model is computed, which computes the joint angles necessary to bring the robot to a given position. The major difficulty with inverse kinematics is that there are usually many settings of the joint angles that put the gripper of a robot arm at a given point in workspace. One of the aims of MURPHY is to achieve its task without computing the inverse kinematics map by using only a forward model: the robot will change randomly its joint angles and see where it goes and compare this position to the goal. Of course these movements must not be done by the real robot; it is why MURPHY will build an inner model of itself and simulate these movements with this model before it applies, this time in real world, the chosen movement.

The connectionist architecture consists in several interconnected neural nets: a visual-field net (24×24 units which respond when a visual feature falls into the receptive field), a hand-velocity net (24 units which respond to the visual image of the hand and are selective for the direction and amplitude of the hand motion), a joint-angle net (273 units which encode static joint angles) and a joint-velocity net (24 units which encode the velocity for every joint). All these nets are single-layered and the units are grouped into clusters; the output is a weighted sum of contributions from a set of multiplicative clusters of input weights ($y = \sum_j w_j \prod_i v_{ij} x_i$, hence the name sigma-pi learning for such nets).

MURPHY builds its forward kinematic model by stepping its arm through a relatively small uniform sample (17000) of the 3.3 billion legal arm configurations; the relationships between joint angles and states are learned. Once the robot has learned to picture its arm in an arbitrarily joint configuration, it can use heuristic search to guide "mentally" (this term will be used for all actions that are not made in real world but are computed by using the learned inner model) its arm to a visually-specified target. If the mental image of the hand falls in superposition with the target, the robot has found a path joining the initial and the goal configuration. However this path is not always feasible (the arm becomes tangled up in its own joints or with obstacles)

and in this case the arm has to be backed up and other ways of performing the task have to be found. This is done by a search procedure which first labels the target and any visible obstacles; the distance between hand and target is computed, taking in account visible obstacles; the arm is sent mentally in a random direction and if the new distance is shorter than the previous, this is the new state; if not, the state is stored on the stack of things to try later and another direction is chosen; if none are available, the state computed one step earlier is taken and the next most promising move stored in the stack is tried. If this depth first search leads to the final configuration, the arm is moved physically following the mentally computed path dispensing with all backtracking dead-ends. If no path is found, the robot has to give up.

Experience shows the robot copes well with its task; however the paths involve sometimes some useless and complex motions (although all the dead-ends found during the mental computing of the path are now skipped, depth-first search does not find the shortest path, it only gives the first available solution). In order to smooth its movements and to speed up the research of a new motion, the inverse differential map is computed, which takes joint angles and a desired workspace perturbation into a set of joint-angle perturbations.

The methodology used by MURPHY is interesting because it is a learning system where the injected knowledge is not too important; however the strategy used to bring the hand towards the goal follows a steepest gradient method, which is a simple algorithm but nevertheless a definite help for the learning system. The interesting part lies in the identification stage when the robot learns by doing and acquires an internal model of its kinematics map which is then used internally to simulate the motion. The difference with the previous neural nets is that the examples needed to compute the weighting coefficients inside the neural net are not given by an external teacher, but they are acquired by the robot using its own visual system and following some initial exploring motions.

3.2 Fuzzy control

We give some elementary definitions and then we show how fuzzy control deals with car-like robots. Let us call I the interval $[0,1]$ and let X be a set; then a *fuzzy set* on X is a function from X to I. The set of all fuzzy subsets on X is denoted I^X. Given $\mu \in I^X$ and $x \in X$, the value $\mu(x)$ can be interpreted as the "degree membership" of x to μ and therefore μ is often called a *membership function.*

Fuzzy sets have been intensively used in situations in which impreciseness is not apparently of a probabilistic nature. One of the basic ideas in fuzzy theory is to associate real numbers to certain objects; other theories have the same goal, such as metric spaces or probability theory. An extended theoretical study of fuzzy sets can be found in [Low85], where topological and probabilistic considerations are taken into account.

Fuzzy theory can be used to model input/output relationships and this is the key for fuzzy control. A fuzzy controller is basically a set of fuzzy if-then rules like: if A and B then C, where A, B are fuzzy outputs of the system to control and C is a fuzzy input. Of course, in order to control a system, you need real inputs and you observe real outputs; translating real data into fuzzy sets is called *fuzzification* while the converse operation is *defuzzification.* Different ways of performing either of these two operations have been presented in fuzzy literature. The general scheme of fuzzy control is then: first fuzzify the outputs, then scan through the fuzzy rule-base, in order to obtain a bunch of fuzzy inputs (this stage is often called *fuzzy inference*), and defuzzify in order to have a real input.

One of the advantages of fuzzy control is the "natural" way of expressing things: fuzzy rules look often like "if error on position is small then the input has to be large". Of course, the fuzzy values "small" or "large" depend on the given membership functions, and although most applications claim they have some sort of robustness concerning the local shape of the membership functions, the global shape (width, grade of incline) depends heavily on every application, as we will see for instance in the next section. This is one of the main drawbacks when dealing with learning systems used for automatic acquisition of a fuzzy controller for a given system.

3.2.1 Truck Backer-upper

The fuzzy controller presented here [Kos92] backs up a simulated truck, with and without trailer, to a loading dock; it has been compared to the neural truck backer-upper described before, and an adaptive fuzzy controller is presented too, that merges these two approaches, and allows to learn parts of the fuzzy controller from a set of data.

The truck has three state variables x, y and θ which determine its position (x and y are the coordinates of the rear center of the truck) and its angle with the horizontal. The goal is to make the truck arrive at the loading dock (align the position (x, y) of the truck with the desired (x_f, y_f)) at a right angle. Enough clearance between the initial position and the dock is assumed, so the y coordinate may be ignored; the loading zone where the truck moves corresponds to the square $[0, 100] \times [0, 100]$ and $(x_f, y_f) = (50, 100)$; ϕ is smaller than 30 degrees in norm.

To each of the variables is associated a set of linguistic variables (5 for x: LE, LC, CE, RC, RI standing for left, left center, center, right center, right; 7 for θ: RB, RU, RV, VE, LV, LU, LB standing for right below, right upper, right vertical, vertical, left vertical, left upper, left below; 7 for ϕ: NB, NM, NS, ZE, PS, PM, PB standing for negative big, negative medium, negative small, zero, positive small, positive medium, positive big) and fuzzy membership functions (triangular shape); there are 35 fuzzy rules (ϕ as a function of x and θ):

$\theta \setminus x$	LE	LC	CE	RC	RI
RB	PS	PM	PM	PB	PB
RU	NS	PS	PM	PB	PB
RV	NM	NS	PS	PM	PB
VE	NM	NM	ZE	PM	PM
LV	NB	NM	NS	PS	PM
LU	NB	NB	NM	NS	PS
LB	NB	NB	NM	NM	NS

Each fuzzy rule produces a fuzzy output (the output fuzzy set is clipped at the degree of membership determined by the input conditions and the rule); to defuzzify this output and obtain a real output, centroid defuzzification is used (sum of the products of angle times membership value over sum of membership values), which yields the inertia center of the fuzzy sets produced by the fuzzy rules.

Comparison with the neural backer-upper described previously shows that the fuzzy controller is computationally lighter than the neural controller: most operations for the neural controller involve multiplication and addition, while the fuzzy controller involves comparison and addition of two real numbers. Furthermore some trajectories found by the neural controller are far more complex than the trajectories found by the fuzzy controller for same initial conditions.

Of course, neural nets offer a way to infer a controller starting from raw data (these data are the samples chosen – mostly carefully – for the learning stage), while the fuzzy controller has been written by encoding some common-sense knowledge. It could be interesting to merge both methodologies in order to write a fuzzy controller starting from the raw data; this is called the "adaptive fuzzy truck backer-upper".

The 3 variables are divided respectively into 5, 7 and 7 fuzzy-set values like previously, which gives $5 \times 5 \times 7 = 245$ cells; then 2230 truck samples from seven different initial conditions and varying angles are generated. By finding clusters among the data, 35 fuzzy rules can then be found. These fuzzy rules produce nearly equivalent truck-backing behavior. Of course, one should notice that much *a priori* knowledge is needed to "learn" such fuzzy rules: the number of fuzzy linguistic values is known and their size is given; for instance, uniform partitions of the product space give poor results.

If a trailer is added to the truck, a fuzzy controller can be found too (7 fuzzy variables for the angle of the trailer with horizontal, 3 fuzzy variables for the relative angle of the cab with the trailer and always 5 fuzzy variables for x and 7 for ϕ). The adaptive fuzzy controller needed this time 6250 data and found a set of fuzzy rules yielding comparable performance. The same remarks apply of course, concerning the a priori knowledge.

3.3 Rule-based incremental control

We have already seen this type of control when dealing with robot motion and control. We will now see how learning techniques may be used to write such controllers. We describe briefly the learning program CANDIDE which learns to control a process without a priori knowledge about this process; it observes random initial evolutions of the process and acquires a qualitative model: monotonous and derivative relationships between inputs and outputs are looked for; then a rule-based incremental controller is deduced from this model.

This learning program has been applied to the car driving problem, and we give here the results obtained by CANDIDE. These may be compared advantageously to other rule-based incremental controllers written manually [Fou90].

3.3.1 Acquisition of a qualitative model

Let us come back now to the learning program CANDIDE which will learn to write such rule-based incremental controllers as described in a previous part for a given process starting from no initial knowledge about this process. This process is assumed controllable and observable; this means we suppose we have the right sensors to observe it, or at least among our sensors we have the right sensors (there may be redundant data or even useless data available, this will only slow the learning program) and we have the right commands (we can control the process).

Initially the only available thing is the process; we will proceed by random perturbation of the process in order to control it later. Random initial values are given to the inputs and all inputs are maintained constant except one which is either decreased or increased; this is done several times for all different inputs, and at each sampling time and for each trial, the values of all inputs and outputs are stored in an *experimental data base* (edb). The aim of this step is to explore the evolution space as much as possible. The obtained edb is given to the learning program which will first acquire a qualitative model and then find a controller C_1. This controller is then applied to the process, random initial values are always given to the inputs but this time the process is controlled by C_1 and in the same way all values of inputs and outputs are stored in an edb, which is given to CANDIDE, yielding then a controller C_2. This recurrent learning stops as soon as the controller is satisfactory. We do not give in this paper any results on convergence of this learning, we are currently working on it with E. Martin (thesis to appear). We will now explain more precisely how CANDIDE works.

Let us assume the physical process we are studying is measurable via a given number of sensors. Every sensor may be considered as a real-valued parameter with values sampled in time.

The aim of the qualitative model is to express easily relationships between these parameters.

If one does not want to be blinded by the huge amount of data, it is necessary to find an appropriate representation, much more concise while keeping a maximal information on the evolution of the parameters: grammars[ASU86]. As we are studying parameters through a time sampling, the problem is a multivariable problem with a privileged dimension. Regular grammars are very appropriate in this case. We have to find a mapping between the numerical level and the grammatical formalism; this is done in two steps: a coding step and a grammatical inference step.

The coding step will transform the numerical data into strings written on a reduced alphabet. Then we use very efficient algorithms derived from[Fu82], which extract a regular grammar from a given set of strings written on a finite alphabet. The particularity of the resulting grammar is to accept the given strings and to optimize a criterion, so that we can assume the found grammar is one of the best we could obtain[LZ89].

In the next subsections, we will explain how we code the evolutions of all parameters and we will then show how we obtain a grammar modeling these evolutions.

How evolutions may be coded

The variations of each parameter will be coded by three explicit signs, $\uparrow$, $\downarrow$ and 0, meaning respectively increasing, decreasing and constant.

For a given time sampling $(t_i)_{i\in I, |I|<\infty}$, each parameter x can be understood as a list:

$$(x_{t_0}, x_{t_1}, x_{t_2}, \cdots)$$

The qualitative evolution of each parameter is given by *derivated lists*:

$$(\phi[x_{t_1} - x_{t_0}], \phi[x_{t_2} - x_{t_1}], \cdots)$$

The application ϕ is defined by:

$$\phi[y] = \begin{cases} \uparrow & \text{if } y > 0 \\ \downarrow & \text{if } y < 0 \\ 0 & \text{if } y = 0 \end{cases}$$

An expression written on the 3-letter alphabet is an image of the temporal evolution of the parameter, it may be seen as a sentence.

We will try to find a regular grammar modeling these sentences. We chose regular grammars because they are the adequate tool to describe repetitions and succession of patterns: we want to have for a given parameter a symbolic representation which tells us for instance that the parameter is first increasing, then decreasing and then constant; this representation is easily performed

by regular grammars. To each parameter we will then assign its *grammatical representation* and the *interval of validity* (maximum and minimum values).

The following subsection describes the process leading to a regular grammar from a set of strings.

Robust grammatical inference

We are now concerned with inductive inference related to regular grammars: regular grammatical inference is a way of inferring a continuous description (a regular grammar) from a discrete description (a finite set of strings). More formally, it can be expressed as: *let* $s_1, \cdots, s_n$ *be* n *strings, find a regular grammar* G *accepting the* n *strings.*

Such a problem has many solutions, many being uninteresting because of their excessive generalization (the universal grammars which accepts everything is always a solution). The "right" grammars should then be chosen so as to verify a given criterion.

As we saw in the previous section, a string models the variations of a parameter during one experiment; we would like to identify some underlying structures in this string and these structures have to be long enough in order to bring relevant information, and should appear frequently, or they could be interpreted as sheer random. These considerations have been formalized into the following criterion: maximize the product of the length of a substring by its number of contiguous occurrencies. This leads to robust grammatical inference (for formal proof of what follows, see [LZ89]). Robustness has to be understood in the following sense: let us consider a given set of strings and let G be the grammar inferred from these strings; let us now change the strings a little (deleting some letters, substituting some letters by others, adding some letters) and let us infer a grammar G' from this new set of strings; then it is very easy to change G' into G. In other words, a small variation on the initial strings does not dramatically change the inferred grammar. This is very important as the strings we will be working on are symbolizing the variations of real parameters measured by sensors and some errors on these measurements (hence on the strings) are likely. The inferred grammar can then be seen as sort of invariant. The transformation used to deduce G from G' is called *filtering method* as it consists in deleting the isolated substrings at the top level in G', as appears in the next examples.

We give now some examples to see how this grammatical inference works. Let us recall at this point some basics on regular grammars. Given an *alphabet* T (a finite set of elements called *letters*), we define the three following operators: *concatenation*, a binary operator, written multiplicatively, which builds the word xy from the two words x and y, *union*, a binary operator, written additively with its usual set theory meaning, and *power*, a unary operator, which is defined by $x^+ = \{x, xx, \cdots, x^n, \cdots\}$. The class of *regular languages* is the smallest set closed for these three operators; a *regular grammar* is then defined recursively by:

- $\forall x \in T$, x is a regular grammar
- If R_1 and R_2 are regular grammars, so are $R_1 + R_2$, R_1^+ and $R_1 R_2$

Let us consider for example the following set of strings: {*aaaaaaababababab* , *aaaaaaaaaa* , *aaaaaaaabab*}; we infer the regular grammar $a^+ + a^+(ab)^+$, i.e. all the words containing only a's and the words starting with a's and ending with succession of the pattern ab. If we add to the previous set of strings the noisy string *aaaaaaeaaaaa*, we infer the regular grammar $a^+ + a^+(ab)^+ + a^+ea^+$; applying the filtering method on this grammar, we transform a^+ea^+ into a^+ which is the grammar we would have obtained if the letter e (a 1-order noise) had not been present in the sample string.

Let us take now a real-world example used while applying CANDIDE to the car driving problem. We will give more details on the car simulation later; let us just give here one small part of the edb, where two parameters are considered during one experiment: speed and acceleration.

```
(SPEED
  (0.5   1.12  1.768 2.423 3.08  3.736  4.391  5.045  5.698  6.349
   6.999 7.647 8.317 9.302 9.999 11.106 11.752 13.227 13.716))
(ACCELERATION
  (2.5   3.099 3.243 3.275 3.281 3.28   3.276  3.271  3.264  3.257
   3.249 3.241 3.346 4.925 3.488 5.535  3.229  7.377  2.444))
```

The variations of these parameters are then computed: however due to sensor reliance margins, when computing these variations, two elements are declared equal if their relative difference if smaller than 1%; the first element is greater (resp. smaller) than the second if the relative difference is greater (resp. smaller) than 1% (resp. -1%). Then the grammatical inference with the filtering method is applied and the results are:

```
(SPEED (↑+))
(ACCELERATION (0+(↑↓)+))
```

Extracting causal dependencies

For each parameter we have now after the coding step and the inference step a regular grammar which describes its qualitative evolution. A qualitative behavior model will be found by extracting relations of *monotony* and *derivative* between the parameters.

Let us call $R(x)$ the regular grammar inferred from the variations of the parameter x and $\bar{R}(x)$ the regular grammar deduced from $R(x)$ if uparrows are changed into downarrows and conversely. We introduce for each x the new parameter x' computed with $x'_{t_i} = x_{t_{i+1}} - x_{t_i}$.

Monotony and derivative relations are based upon the mathematical ones, more exactly the qualitative definitions are defined so as to cover the quantitative definitions.

The relationship between two physical parameters x and y is *monotonically increasing* if the regular grammar $R(x)$ and $R(y)$ modeling the variations of both parameters are equal. If $\bar{R}(x)$ and $R(y)$ are equal, the relationship is *monotonically decreasing.* In the first case, we write $M^+(x,y)$ and in the second case $M^-(x,y)$.

The notion of derivatives is based upon the mean values theorem (t_i, t_j are consecutive time steps):

$$\| f(t_i) - f(t_j) - (t_i - t_j)f'(t_i) \| \leq (t_i - t_j)^2 \sup_{t \in]t_j, t_i[} f''(t)$$

If we suppose the parameters behave as continuous differential functions of time with uniformly bounded derivatives, which is the case in most encountered physical processes, we may consider that the right term of this equation is very small (it is in fact true in the limit, cf. Taylor-Young theorem). We say y is a qualitative *positive* (resp. *negative*) *derivative* of x if $R(x') = R(y)$ (resp. $R(x') = \bar{R}(y)$). We write in the first case $D^+(x,y)$ and in the second case $D^-(x,y)$.

This notion of qualitative derivative has to be handled with care; we keep the name because it relates the variations of one parameter with the variations of the variations of another parameter, but it should not be interpreted as an exact derivative. All these relations express trends, they show how one parameter acts on another. We want to make another point: delays are not taken into account; if $y(k) = x(k - k_0)$, the evolutions of x and y are modeled by the same grammatical grammar, hence x is monotonically increasing with y. If x is the input and y an output, this means that any action on x will in fact influence y after a delay k_0 (for instance in a shower: turning the tap does not affect immediately the temperature). This is not a drawback in our approach, as incremental control deals well with this kind of situations: when an input is incremented, the effect is not immediate, but may well take place some moments later. During these few moments, the input will keep increasing and the effect will be reinforced (however this may yield overshooting).

If we consider again the real-world given before, by computing the regular grammar inferred from the variations of the variations of SPEED, we see it is equal to the regular grammar inferred from ACCELERATION which allows us to state that ACCELERATION is a positive derivative of SPEED.

3.3.2 How to write rule-based incremental controllers

For each parameter x, let us compute the minimal and the maximal values taken during the various experiments. We introduce then $b_{\min}(x)$ and $b_{\max}(x)$ as being respectively the maximum of all minimal values and the minimum of all maximal values. Let us introduce too $b'_{\min}(x)$ and $b'_{\max}(x)$ as respectively the minimum of all minimal values and the maximum of all maximal values.

The interval $[b_{\min}(x), b_{\max}(x)]$ gives strong constraints on x and we may have $b_{\min}(x) > b_{\max}(x)$. On the contrary, $[b'_{\min}(x), b'_{\max}(x)]$ is always ordered in this way, and gives weak constraints on x.

The general scheme in order to write rule-based incremental controllers is to constrain x between $b_{\min}(x)$ and $b_{\max}(x)$, while keeping x (it is a safety measure in the case of $b_{\min}(x) > b_{\max}(x)$) between $b'_{\min}(x)$ and $b'_{\max}(x)$. For instance, for the driving car application, if x is the distance between the car and the middle of the road, $b'_{\min}(x)$ and $b'_{\max}(x)$ give the width of the road, while $b_{\min}(x)$ and $b_{\max}(x)$ only give intermediate values, usually very near to 0; thus x will be constrained between these two small values while being kept between the boundaries of the road (it is important to see that nowhere the car is told to follow as a reference signal the middle of the road; this is an immediate consequence of the definition of these four extremal bounds).

It is then easy to write the control rules: we look at the parameters which are related to an input through some qualitative relation and we constrain these parameters between the associated boundary values, by acting on the input (let u be the input and x the current parameter):

- $M^+(x,u) \rightarrow$ "if $x > b_{\max}(x)$ then decrease u"
  "if $x < b_{\min}(x)$ then increase u"

  "if $x > b'_{\max}(x)$ then decrease u"
  "if $x < b'_{\min}(x)$ then increase u"

- $D^+(u,x) \rightarrow$ "if $x > b_{\max}(x)$ then decrease u"
  "if $x < b_{\min}(x)$ then increase u"

  "if $x > b'_{\max}(x)$ then decrease u"
  "if $x < b'_{\min}(x)$ then increase u"

The rules for $M^-(x,u)$ and $D^-(x,u)$ are deduced from the previous rules, by replacing increase with decrease and conversely.

The rules corresponding to derivative relationships have to be distinguished from the other rules: these rules do not imply an immediate reaction, their action will be fulfilled after some delay. By decreasing u, the aim is to have it negative; by increasing u, we want it in fact to become positive.

The rules corresponding to qualitative derivative relations seem to be the same as the other rules corresponding to monotonous relations. The previous remark shows it is not in fact the same thing, but however we concede that some ameliorations could be made in order to draw much more information from the qualitative model.

3.3.3 CANDIDE and car driving

The previous methodology has been applied to a car driving on a circuit. The model of the car used in the simulation is rather complete; the inputs are the steering angle (ANGLE) and the power of the engine (POWER, positive to accelerate and negative to decelerate); the outputs are the speed of the car (SPEED), the sliding speed (SLIDE), the acceleration (ACCELERATION), the distance of the car with the middle of the road (VISION-POS-CAR), the angle between the axis of the car and the middle of the road (VISION-ANG-CAR), the reaction force between the road and the wheels and the instantaneous curvature radius.

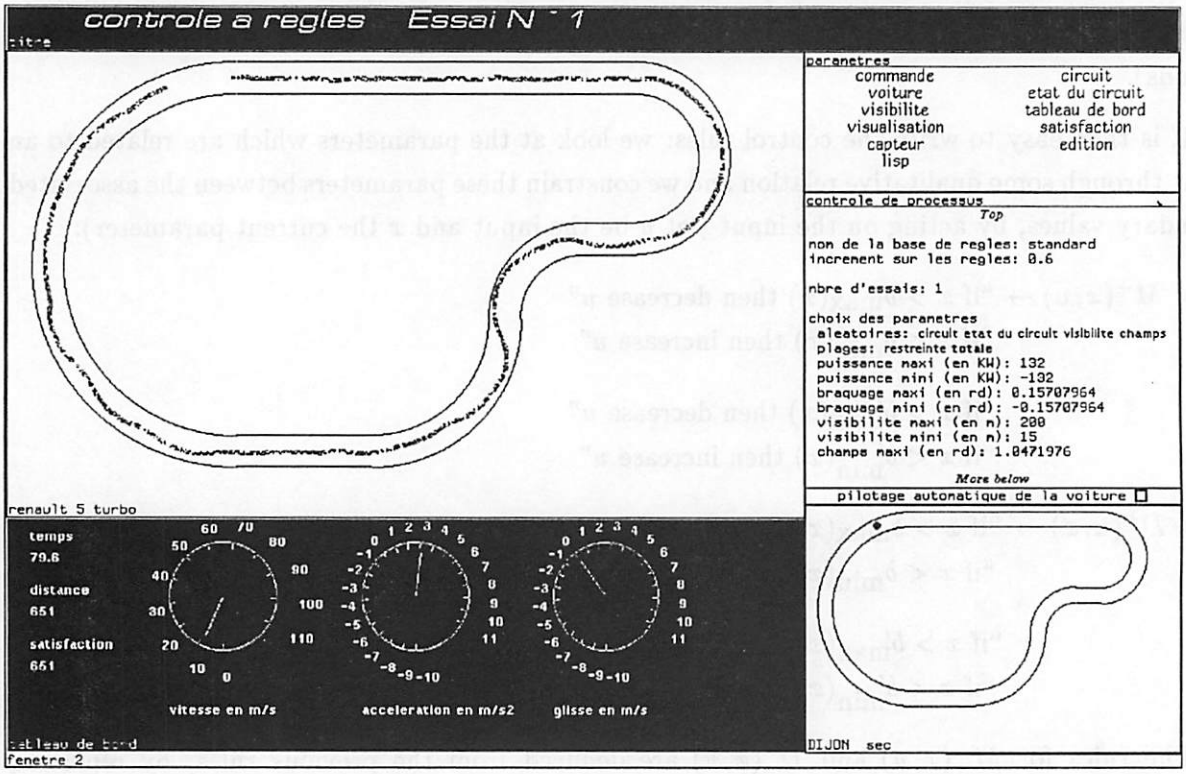

Figure 3.1: CANDIDE drives a car

The controller found by CANDIDE is:

```
(
 (rule 1  (<  SPEED 0.5)(increase-ANGLE))
 (rule 2  (<  SPEED 0.5)(increase-POWER))
 (rule 3  (>  SPEED 12.387)(decrease-POWER))
 (rule 4  (>  SPEED 31.454)(decrease-ANGLE))
 (rule 5  (<  SLIDE 0.0)(decrease-ANGLE))
```

```
(rule 6  (<  SLIDE -0.269)(decrease-ANGLE))
(rule 7  (>  SLIDE 1.153)(increase-ANGLE))
(rule 8  (>  SLIDE 12.353)(increase-ANGLE))
(rule 9  (<  ACCELERATION -0.0)(increase-POWER))
(rule 10 (>  ACCELERATION 3.281)(decrease-POWER))
(rule 11 (<  VISION-POS-CAR -1.647)(increase-ANGLE))
(rule 12 (<  VISION-POS-CAR -8.341)(increase-ANGLE))
(rule 13 (>  VISION-POS-CAR 1.647)(decrease-ANGLE))
(rule 14 (>  VISION-POS-CAR 7.542)(decrease-ANGLE))
(rule 15 (<  VISION-ANG-CAR 0.0)(increase-ANGLE))
(rule 16 (<  VISION-ANG-CAR -0.74)(increase-ANGLE))
(rule 17 (>  VISION-ANG-CAR 0.0)(decrease-ANGLE))
(rule 18 (>  VISION-ANG-CAR 0.919)(decrease-ANGLE))
)
```

The first experiments have been done by giving first random initial values to the power and the steering wheel angle and then by increasing or decreasing the steering wheel angle till the car leaves the road. This yields a first half of the edb. Then starting always with random initial values for both commands, this time the power is increased or decreased; this yields the second half of the edb. CANDIDE works then on this edb, finds a qualitative model of the car and writes a rule-based incremental controller. On the circuit shown on the figure, the learning loop has converged after the first learning loop.

3.3.4 Learnability and convergence of the learning process

We will not develop these points here, as they are currently under research and all the results will appear in Eric Martin's thesis (around June 92). This is the first formalization of the application of learning techniques to control problems and a theoretical basis will be given based on the theory of recursive functions; learnability and reducibility between learning problems are then developed.

3.4 Miscellaneous

The word learning is often used in a too broad meaning; one should distinguish real learning (acquisition of new structures) from mere modification of coefficients. In [FT87] a cell decomposition of the configuration space is given; probabilities of entry of each cell are given and they are updated while moving the robot. Many neural networks are trained to perform adaptive control laws and this is tuning of coefficients rather than learning. In the same way, expert systems used to control robots like path planners, which follow a given strategy in order to adapt local techniques, should not be considered as learning systems. In the same way as it is important

to distinguish between techniques which have theoretical ability to solve the problem of robot motion and techniques which do not have it, it is necessary to separate techniques with real learning ability from the others.

Chapter 4

Conclusion

We have seen in this paper several ways to solve the robot motion problem; instead of considering these different methods as competing solutions to the same problem, we have looked for some common motivation in order to classify them and see the advantages of each separately, rather than emphasizing the drawbacks.

We have seen that the robot motion problem could be understood as a geometrical problem, a control problem or an artificial intelligence problem; it is thus related to many different fields like algebraic geometry, graph theory, non linear control, differential geometry, learning, among others. The various methods focus on different issues: either *finding a geometrical solution* when the movement constraints of the robot are not taken into account, or *proving the existence of a solution* with a given robot, or *designing a way to find such a solution.* Different goals imply different tools ranging from explicit resolution of equations and formal reasoning to application of machine learning techniques.

Robot motion has to be considered in its whole and cooperation between the different existing methods could yield an efficient way to solve it. Already some small-scale cooperation is available, like mixing neural nets and fuzzy rule-based control [Kos92], rule-based reasoning and neural nets [HLG91], or geometrical motion planning and non linear control. We have discussed incremental rule-based control too, where a theoretical approach proving the existence of a solution copes well with a learning program. These are only first steps and with the rising use of robots, further research is necessary if one wants to have a robot that can find its way in a very short time in a clustered and changing environment ... Reliability and algorithmic complexity have never managed too well together and no particular method should be discarded to achieve this ultimate goal.

Back to more short-term considerations: further work on rule-based incremental control includes the study of the rule-based controllers themselves. It has been shown, for instance for linear systems, that an incremental control could be found to achieve such or such goal; but it

would be useful to show that these incremental control laws can be expressed in an *easy* way by a finite set of rules, like the rules found by the learning program CANDIDE. This would be another step toward proving the theoretical convergence of the learning process.

But this is another story ...

Bibliography

[ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers, Principles, Techniques and Tools.* Addison-Wesley, 1986.

[Ave83] A. Avez. *Calcul différentiel.* Masson, 1983.

[BF89] E. Benoit and L. Foulloy. Configuration coopérative d'un capteur symbolique. Technical report, LAMI, Université de Savoie, 1989.

[BFHZ85] B. Burg, L. Foulloy, J.C. Heudin, and B. Zavidovique. Behaviour rule systems for distributed process control. In *IEEE Conf. On Artificial Intelligence Applications*, Miami, FL, 1985.

[BL89] J. Barraquand and J.C. Latombe. Robot motion planning: a distributed representation approach. Technical report, Stanford University, No STAN-CS-89-1257, 1989.

[BLZ89a] B.Burg, D. Luzeaux, and B. Zavidovique. A system which learns to control a process. In *IFAC Symposium*, Nancy, 1989.

[BLZ89b] B. Burg, D. Luzeaux, and B. Zavidovique. Apprentissage de règles de comportement destinées au contrôle d'un système. In *JFA 4*, St-Malo, 1989.

[BLZ89c] B. Burg, D. Luzeaux, and B. Zavidovique. Candide: a learning system for process control. In *IEA/AIE-89*, Tullahoma, TE, 1989.

[Bur88] B. Burg. *Apprentissage de Règles de Comportement destinées au contrôle d'un système.* Thèse de Troisième Cycle, Université d'Orsay, Paris, 1988.

[Can87] J.F. Canny. *The complexity of robot motion planning.* The MIT Press, 1987.

[CL90] J.F. Canny and M.C. Lin. An opportunistic global path planner. In *ICRA 90*, 1990.

[FL89] L. Foulloy and M. LeGoc. Towards a methodology to write rules for expert controllers. In *Advanced Information Processing in Automatic Control, IFAC Symposium*, Nancy, 1989.

[Fou90] L. Foulloy. *Du contrôle symbolique des processus: démarche, outils, exemples.* Thèse d'Etat, Université d'Orsay, Paris, 1990.

[FT87] B. Faverjon and P. Tournassoud. The mixed approach for motion planning: learning global strategies from a local planner. In *IJCAI 87*, 1987.

[Fu82] K.S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, 1982.

[HLG91] D.A. Handelman, S.H. Lane, and J.J. Gelfand. Goal-directed encoding of task knowledge for robotic skill acquisition. In *IEEE International Symposium of Intelligent Control*, Arlington, VA, 1991.

[Kos92] B. Kosko. *Neural networks and fuzzy systems: a dynamical approach to machine intelligence.* Prentice Hall, 1992.

[Lat91] J.C. Latombe. *Robot Motion Planning.* Kluwer academic publishers, 1991.

[Lau90] J.P. Laumond. Planning versus controllability via the multibody car system example. Technical report, Stanford University, No STAN-CS-90-1345, 1990.

[Low85] R. Lowen. *On the existence of natural non-topological, fuzzy topological spaces.* Heldermann Verlag Berlin, Research and Exposition in mathematics 11, 1985.

[LTJ90] J.P. Laumond, M. Taix, and P. Jacobs. A motion planner for car-like robots based on a mixed global/local approach. In *IROS, Tscuchiura. Ibaraki, Japan*, 1990.

[Luz91] D. Luzeaux. *Pour l'apprentissage, une nouvelle approche du contrôle: théorie et pratique.* Thèse de Troisième Cycle, Université d'Orsay, Paris, 1991.

[LZ89] D. Luzeaux and B. Zavidovique. Robust grammatical inference. In *AFCET, Paris*, 1989.

[LZ90a] D. Luzeaux and B. Zavidovique. Acquisition of a qualitative model. In *IEA-AIE*, Charleston, SC, 1990.

[LZ90b] D. Luzeaux and B. Zavidovique. Process control and machine learning. In *IEEE Workshop on Motion Control*, Istanbul, Turkey, 1990.

[LZ91] D. Luzeaux and B. Zavidovique. Numerical-symbolic duality: a formal approach. In *IEEE International Symposium of Intelligent Control*, Arlington, VA, 1991.

[Mel90] B. W. Mel. *Connectionist robot motion planning: a neurally-inspired approach to visually-guided reaching.* Academic Press, Perspectives in artificial intelligence: volume 7, 1990.

[Mur90] R.M. Murray. *Robotic control and non holonomic motion planning.* PhD thesis, University of California, No UCB-ERL-M90-117, 1990.

[NSA90] S. Nagata, M. Sekiguchi, and K. Asakawa. Mobile robot control by a structured hierarchical neural network. *IEEE Control Systems Magazine*, 10-3:69–76, 1990.

[NW90] D. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 10-3:18–23, 1990.

[RM91] S.S. Sastry R.M. Murray. Nonholonomic motion planning: steering using sinusoids. Technical report, University of California, No UCB-ERL-M91-45, 1991.

[TLM+92] D. Tilbury, J.P. Laumond, R. Murray, S. Sastry, and G. Walsh. Steering car-like systems with trailers using sinusoids. In *IEEE Conference on robotics and automation (preprint)*, Nice, 1992.

[WTS+92] G. Walsh, D. Tilbury, S. Sastry, R. Murray, and J.P. Laumond. Stabilization of trajectories for systems with nonholonomic constraints. In *IEEE Conference on robotics and automation (preprint)*, Nice, 1992.

[ZFG84] B. Zavidovique, L. Foulloy, and D. Gerbet. Towards the adaptative laser robot. In *SPIE, Intelligent robot and Computer vision*, 1984.