BAYESIAN ROBOT PROGRAMMING

Contact:
Pierre Bessière
CNRS - Laboratoire GRAVIR
655 avenue de l'Europe
38334 Montbonnot
FRANCE

Email: Pierre.Bessiere@imag.fr
Telephone: +33(0)4.76.61.55.09

WWW: www-laplace.imag.fr

To the late Edward T. Jaynes

for his doubt about certitudes

and for his certitude about probabilities

This booklet provides an overview of the work done at the University of Grenoble about Bayesian Robot Programming.

It is a collection of 7 papers corresponding to 6 of the 7 PhD thesis that have been defended in Grenoble during the past 7 years on this subject plus one corresponding to an ongoing work.

The papers are the following:

1. *Bayesian Robot Programming* (2004); Lebeltel, O. et al.; Autonomous Robots

2. *The design and implementation of a Bayesian CAD modeler for robotic applications* (2001); Mekhnacha, K. et al.; Advanced Robotics

3. *Hierarchies of probabilistic models of navigation: the Bayesian Map and the Abstraction operator* (2004); Diard, J. et al.; International Conference on Robotics & Automation (ICRA)

4. *Bayesian Occupancy Filtering for Multi-Target Tracking : an Automotive Application* (2004); Coué, C. et al.; International Journal of Robotic Research (IJRR)

5. *The CyCab: a Car-Like Robot Navigating Autonomously and Safely Among Pedestrians* (2004); Pradalier, C. et al.; Robotics and Autonomous Systems

6. *Teaching Bayesian Behaviours to Video Game Characters* (2004); Le Hy, R. et al.; Robotics and Autonomous Systems

7. *The Ariadne's Clew Algorithm* (1998); Ahuactzin, J-M.; Journal of Artificial Intelligence Research (JAIR)

Autonomous Robots 16, 49–79, 2004

Bayesian Robot Programming

OLIVIER LEBELTEL, PIERRE BESSIÈRE, JULIEN DIARD AND EMMANUEL MAZER
Laboratoire GRAVIR, CNRS, INRIA, 655 Avenue de l'Europe, 38334 St. Ismier, France
Pierre.Bessiere@imag.fr

Abstract. We propose a new method to program robots based on Bayesian inference and learning. It is called BRP for Bayesian Robot Programming. The capacities of this programming method are demonstrated through a succession of increasingly complex experiments. Starting from the learning of simple reactive behaviors, we present instances of behavior combination, sensor fusion, hierarchical behavior composition, situation recognition and temporal sequencing. This series of experiments comprises the steps in the incremental development of a complex robot program. The advantages and drawbacks of BRP are discussed along with these different experiments and summed up as a conclusion. These different robotics programs may be seen as an illustration of probabilistic programming applicable whenever one must deal with problems based on uncertain or incomplete knowledge. The scope of possible applications is obviously much broader than robotics.

Keywords: Bayesian robot programming, control of autonomous robots, computational architecture for autonomous systems, theory of autonomous systems

1. Introduction

We assume that any model of a real phenomenon is *incomplete*. There are always some hidden variables, not taken into account in the model, that influence the phenomenon. The effect of these hidden variables is that the model and the phenomenon never have the same behavior.

Any robot system must face this central difficulty: how to use an incomplete model of its environment to perceive, infer, decide and act efficiently? We propose an original robot programming method that specifically addresses this question.

Rational reasoning with incomplete information is quite a challenge for artificial systems. The purpose of Bayesian inference and learning is precisely to tackle this problem with a well established formal theory. BRP heavily relies on this Bayesian framework.

We present several programming examples to illustrate this approach and define *descriptions* as generic programming resources. We show that these resources can be used to incrementally build complex programs in a systematic and uniform framework. The system is based on the simple and sound basis of Bayesian inference. It obliges the programmer to explicitly state all assumptions that have been made. Finally, it permits effective treatment of incomplete and uncertain information when building robot programs.

The paper is organized as follows. Section 2 offers a short review of the main related work, Section 3 is dedicated to definitions and notations and Section 4 presents the experimental platform. Sections 5 to 9 present various instances of Bayesian programs: learning simple reactive behaviors, instances of behavior combinations, sensor fusion, hierarchical behavior composition; situation recognition, and temporal sequencing. Section 10 describes a combination of all these behaviors to program a robot to accomplish a night watchman task. Finally, we conclude with a synthesis summing up the principles, the theoretical foundations and the programming method. This concluding section stresses the main advantages and drawbacks of BRP.

2. Related Work

Our work is based on an implementation of the principle of the Bayesian theory of probabilities.

In physics, since the precursory work of Laplace (1774, 1814), numerous results have been obtained using Bayesian inference techniques (to take uncertainty into account) and the maximum entropy principle (to take incompleteness into account). The late Edward T. Jaynes proposed a rigorous and synthetic formalization of probabilistic reasoning with his "Probability as Logic" theory (Jaynes, 2003). A historical review of this approach was offered by Jaynes (1979) and an epistemological analysis, by Matalon (1967). Theoretical justifications of probabilistic inference and maximum entropy are numerous. The entropy concentration theorems (Jaynes, 1982; Robert, 1990) are among the more rigorous, Cox theorem (Cox, 1961) being the most well known, although it has been partially disputed recently by Halpern (1999a, 1999b). Numerous applications and mathematical tools have been developed (Smith and Grandy, 1985; Tarentola, 1987; Bretthorst, 1988; Erickson and Smith, 1988a, 1988b; Mohammad-Djafari and Demoment, 1992; Kapur and Kesavan, 1992).

In artificial intelligence, the importance of reasoning with uncertain knowledge has been recognized for a long time. However, the Bayesian approach clearly appeared as one of the principle trends only since the proposal of Bayesian nets (Pearl, 1988) and graphical models (Lauritzen and Spiegehalter, 1988; Lauritzen, 1996; Jordan, 1998; Frey, 1998). Bayesian inference has been proved to be an NP-hard problem (Cooper, 1990). However, very important technical progress has been achieved recently which permits approximated computation in reasonable time (Saul et al., 1996; Zhang and Poole, 1996; Delcher et al., 1996; Darwiche and Provan, 1997; Koller and Pfeffer, 1997; Ruiz et al., 1998; Jaakola and Jordan, 1999; Jordan et al., 1999).

Recent robot programming architectures (Aji and McEliece, 2000; Borrelly et al., 1998; Schneider et al., 1998; Dekhil and Henderson, 1998; Mazer et al., 1998) are in general not concerned with the problem of uncertainty. In robotics, the uncertainty topic was either related to calibration (Bernhardt and Albright, 1993) or to planning problems (Brafman et al., 1997). In the latter case, some authors have considered modeling the uncertainty of the robot motions when planning assembly operations (Lozano-Perez et al., 1984; Donald, 1988) or modeling the uncertainty related to the position of the robot in a scene (Kapur and Kesavan, 1992). More recently Bayesian techniques have been largely used in POMDP[1] to plan complex paths in partially known environments (Kaelbling et al., 1996a, 1996b, 1998; Koening and Simmons, 1998; Beetz and Belker, 2001; Lane and Kaebling, 2001) or for action selection (Rosenblatt, 2000). HMM[2] are also used to plan complex tasks and recognize situations in complex environments (Aycard, 1998; Thrun, 1998). Finally, a lot of works have been done about probabilistic localization and navigation (Shatkay, 1998) either with probabilistic occupancy grids (Konolidge, 1997), Markov localization (Thrun et al., 1998; Gutmann et al., 1998; Murphy, 1999; Fox et al., 2000) correlation-based Markov localization (Konolidge and Chou, 1999), Particle filters (Fox et al., 2001) or Kalman filtering (Roumeliotis and Bekey, 2000a, 2000b).

However, to the best of our knowledge, the design of a robot programming system and architecture solely based on Bayesian inference has never been investigated before the PhD of Olivier Lebeltel, summarized in the present paper (Lebeltel, 1999; Diard and Lebeltel, 1999, 2000; Lebeltel et al., 2000). A paper by Thrun (2000) explored this same direction but with less generality. BRP is a simple and generic framework for robot programming in presence of incompletness and uncertainty. It may be used as a unique formalism to restate and compare numerous classical probabilistic models such as for instance, Bayesian Network (BN), Dynamic Bayesian Network (DBN), Bayesian Filters, Hidden Markov Models (HMM), Kalman Filters, Particle Filters, Mixture Models, or Maxim Entropy Models. This is detailed in a survey by Bessière et al. (2003).

Finally, a presentation of the epistemological foundations of BRP may be found in two articles by Bessière et al. (1998a, 1998b).

3. Basic Concepts

In this section, we introduce the concepts, postulates, definitions, notations and rules that are necessary to define a Bayesian robot program.

It may be read twice, at first rapidly to acquire the main concepts, and revisited after the instances sections (5 to 11) to understand in detail the formal definitions.

3.1. Definition and Notation

3.1.1. Proposition. The first concept we will use is the usual notion of *logical proposition*. Propositions will be denoted by lowercase names. Propositions may be composed to obtain new propositions using the usual

logical operators: $a \wedge b$ denoting the conjunction of propositions a and b, $a \vee b$ their disjunction and $\neg a$ the negation of proposition a.

3.1.2. Variable. The notion of *discrete variable* is the second concept we require. Variables will be denoted by names starting with one uppercase letter.

By definition, a *discrete variable* X is a set of logical propositions x_i such that these propositions are mutually exclusive (for all i, j with $i \neq j$, $x_i \wedge y_j$ is false) and exhaustive (at least one of the propositions x_i is true). x_i stands for "variable X takes its ith value". $\lfloor X \rfloor$ denotes the cardinal of the set X (the number of propositions x_i).

The conjunction of two variables X and Y, denoted $X \otimes Y$, is defined as the set of $\lfloor X \rfloor \times \lfloor Y \rfloor$ propositions $x_i \wedge y_j$. $X \otimes Y$ is a set of mutually exclusive and exhaustive logical propositions. As such, it is a new variable.[3] Of course, the conjunction of n variables is also a variable and, as such, it may be renamed at any time and considered as a unique variable in the sequel.

3.1.3. Probability. To be able to deal with uncertainty, we will attach probabilities to propositions.

We consider that, to assign a probability to a proposition a, it is necessary to have at least some *preliminary knowledge*, summed up by a proposition π. Consequently, the probability of a proposition a is always conditioned, at least, by π. For each different π, $\mathbf{P}(. \mid \pi)$ is an application assigning to each proposition a a unique real value $\mathbf{P}(a \mid \pi)$ in the interval $[0, 1]$.

Of course, we will be interested in reasoning on the probabilities of the conjunctions, disjunctions and negations of propositions, denoted, respectively, by $\mathbf{P}(a \wedge b \mid \pi)$, $\mathbf{P}(a \vee b \mid \pi)$ and $\mathbf{P}(\neg a \mid \pi)$.

We will also be interested in the probability of proposition a conditioned by both the preliminary knowledge π and some other proposition b. This will be denoted $\mathbf{P}(a \mid b \wedge \pi)$.

For simplicity and clarity, we will also use probabilistic formula with variables appearing instead of propositions. By convention, each time a variable X appears in a probabilistic formula $\Phi(X)$, it should be understood as $\forall x_i \in X, \Phi(x_i)$. For instance, given three variables X, Y and Z, $\mathbf{P}(X \otimes Y \mid Z \otimes \pi) = \mathbf{P}(X \mid \pi)$ stands for:

$$\begin{aligned} &\forall x_i \in X, \forall y_j \in Y, \forall z_k \in Z \\ &\mathbf{P}(x_i \wedge y_j \mid z_k \wedge \pi) = \mathbf{P}(x_i \mid \pi) \end{aligned} \qquad \text{[E3.1]}$$

3.2. Inference Postulates and Rules

This section presents the inference postulates and rules necessary to carry out probabilistic reasoning.

3.2.1. Conjunction and Normalization Postulates for Propositions. Probabilistic reasoning needs only two basic rules:

1. The *conjunction rule*, which gives the probability of a conjunction of propositions.

$$\begin{aligned} \mathbf{P}(a \wedge b \mid \pi) &= \mathbf{P}(a \mid \pi) \times \mathbf{P}(b \mid a \wedge \pi) \\ &= \mathbf{P}(b \mid \pi) \times \mathbf{P}(a \mid b \wedge \pi) \end{aligned} \qquad \text{[E3.2]}$$

2. The *normalization rule*, which states that the sum of the probabilities of a and $\neg a$ is one.

$$\mathbf{P}(a \mid \pi) + \mathbf{P}(\neg a \mid \pi) = 1 \qquad \text{[E3.3]}$$

For the purpose of this paper, we take these two rules as postulates.[4]

As in logic, where the resolution principle (Robinson, 1965, 1979) is sufficient to solve any inference problem, in discrete probabilities, these two rules ([E3.2], [E3.3]) are sufficient for any computation. Indeed, we may derive all the other necessary inference rules from those two, especially the rules concerning variables:

1. Conjunction rule for variables:

$$\begin{aligned} \mathbf{P}(X \otimes Y \mid \pi) &= \mathbf{P}(X \mid \pi) \times \mathbf{P}(Y \mid X \otimes \pi) \\ &= \mathbf{P}(Y \mid \pi) \times \mathbf{P}(X \mid Y \otimes \pi) \end{aligned} \qquad \text{[E3.4]}$$

2. Normalization rule for variables:

$$\sum_X \mathbf{P}(X \mid \pi) = 1 \qquad \text{[E3.5]}$$

3. Marginalization rule for variables:

$$\sum_X \mathbf{P}(X \otimes Y \mid \pi) = \mathbf{P}(Y \mid \pi) \qquad \text{[E3.6]}$$

3.3. Bayesian Programs

We define a *Bayesian Program* as a mean of specifying a family of probability distributions. Our goal is to show that by using such a specification one can effectively control a robot to perform complex tasks.

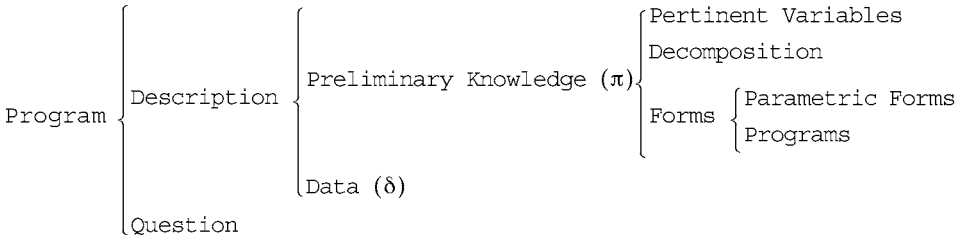


Figure 1. Structure of a Bayesian program.

The constituent elements of a Bayesian program are presented in Fig. 1:

- A program is constructed from a description and a question.
- A description is constructed from preliminary knowledge and a data set.
- Preliminary knowledge is constructed from a set of pertinent variables, a decomposition and a set of forms.
- Forms are either parametric forms or Bayesian programs.

3.3.1. Description. The purpose of a description is to specify an effective method to compute a joint distribution on a set of variables $\{X^1, X^2, \ldots, X^n\}$ given a set of experimental data δ and preliminary knowledge π. This joint distribution is denoted as: $\mathbf{P}(X^1 \otimes X^2 \ldots \otimes X^n \mid \delta \otimes \pi)$.

3.3.2. Preliminary Knowledge. To specify preliminary knowledge the programmer must undertake the following:

1. *Define the set of relevant variables* $\{X^1, X^2, \ldots, X^n\}$ on which the joint distribution is defined.
2. *Decompose the joint distribution*: Given a partition of $\{X^1, X^2, \ldots, X^n\}$ into k subsets we define k variables $L^1, \ldots, L^k$ each corresponding to one of these subsets.

   Each variable L^1 is obtained as the conjunction of the variables $\{X^{i_1}, X^{i_2}, \ldots\}$ belonging to the subset i. The conjunction rule [E3.4] leads to:

$$\begin{aligned}&\mathbf{P}(X^1 \otimes X^2 \otimes \cdots X^n \mid \delta \otimes \pi)\\ &= \mathbf{P}(L^1 \mid \delta \otimes \pi) \times \mathbf{P}(L^2 \mid L^1 \otimes \delta \otimes \pi) \times \cdots\\ &\quad \times \mathbf{P}(L^k \mid L^{k-1} \otimes \cdots \otimes L^2 \otimes L^1 \otimes \delta \otimes \pi)\end{aligned} \quad \text{[E3.7]}$$

   Conditional independence hypotheses then allow further simplifications. A conditional independence hypothesis for variable L^i is defined by picking some variables X^j among the variables appearing in conjunction $L^{i-1} \otimes \cdots \otimes L^2 \otimes L^1$, calling R^i the conjunction of these chosen variables and setting:

$$\begin{aligned}&\mathbf{P}(L^i \mid L^{i-1} \otimes \cdots \otimes L^2 \otimes L^1 \otimes \delta \otimes \pi)\\ &\quad = \mathbf{P}(L^i \mid R^i \otimes \delta \otimes \pi)\end{aligned} \quad \text{[E3.8]}$$

   We then obtain:

$$\begin{aligned}&\mathbf{P}(X^1 \otimes X^2 \otimes \cdots \otimes X^n \mid \delta \otimes \pi)\\ &\quad = \mathbf{P}(L^1 \mid \delta \otimes \pi) \times \mathbf{P}(L^2 \mid R^2 \otimes \delta \otimes \pi)\\ &\qquad \times \mathbf{P}(L^3 \mid R^3 \otimes \delta \otimes \pi) \times \cdots\\ &\qquad \times \mathbf{P}(L^k \mid R^k \otimes \delta \otimes \pi)\end{aligned} \quad \text{[E3.9]}$$

   Such a simplification of the joint distribution as a product of simpler distributions is called a decomposition.
3. *Define the forms*: Each distribution $\mathbf{P}(L^i \mid R^i \otimes \delta \otimes \pi)$ appearing in the product is then associated with either a parametric form (i.e., a function $f_\mu(L^i)$) or another Bayesian program. In general, μ is a vector of parameters that may depend on R^i or δ or both. Learning takes place when some of these parameters are computed using the data set δ.

3.3.3. Question. Given a description (i.e., $\mathbf{P}(X^1 \otimes X^2 \otimes \cdots \otimes X^n \mid \delta \otimes \pi)$), a question is obtained by partitioning $\{X^1, X^2, \ldots, X^n\}$ into three sets: the searched variables, the known variables and the unknown variables.

We define the variables *Search*, *Known* and *Unknown* as the conjunction of the variables belonging to these sets. We define a question as the distribution:

$$\mathbf{P}(\mathit{Searched} \mid \mathit{Known} \otimes \delta \otimes \pi). \quad \text{[E3.10]}$$

3.4. *Running Bayesian Programs*

Running a Bayesian program supposes two basic capabilities: Bayesian inference and decision-making.

3.4.1. Bayesian Inference. Given the joint distribution $\mathbf{P}(X^1 \otimes X^2 \otimes \cdots \otimes X^n \mid \delta \otimes \pi)$, it is always possible to compute any possible question, using the following general inference:

$$\begin{aligned}
&\mathbf{P}(\mathit{Searched} \mid \mathit{Known} \otimes \delta \otimes \pi) \\
&= \sum_{\text{Unknown}} \mathbf{P}(\mathit{Searched} \otimes \mathit{Unknown} \mid \mathit{Known} \otimes \delta \otimes \pi) \\
&= \frac{\sum_{\text{Unknown}} \mathbf{P}(\mathit{Searched} \otimes \mathit{Unknown} \otimes \mathit{Known} \mid \delta \otimes \pi)}{\mathbf{P}(\mathit{Known} \mid \delta \otimes \pi)} \\
&= \frac{\sum_{\text{Unknown}} \mathbf{P}(\mathit{Searched} \otimes \mathit{Unknown} \otimes \mathit{Known} \mid \delta \otimes \pi)}{\sum_{\text{Searched Unknown}} \mathbf{P}(\mathit{Searched} \otimes \mathit{Unknown} \otimes \mathit{Known} \mid \delta \otimes \pi)} \\
&= \frac{1}{\Sigma} \times \sum_{\text{Unknown}} \mathbf{P}(\mathit{Searched} \otimes \mathit{Unknown} \otimes \mathit{Known} \mid \delta \otimes \pi) \\
&= \frac{1}{\Sigma} \times \sum_{\text{Unknown}} \left[\mathbf{P}(L^1) \times \prod_{i=2}^{k} \mathbf{P}(L^i \mid R^i) \right] \qquad \text{[E3.11]}
\end{aligned}$$

where the first equality results from the marginalization rule (Eq. [E3.6]), the second results from the product rule (Eq. [E3.4]) and the third corresponds to a second application of the marginalization rule. The denominator appears to be a normalization term. Consequently, by convention, we will replace it by Σ. Finally, the joint distribution is replaced by its decomposition.

Two main problems have to be solved: searching the modes in a high dimensional space, and marginalizing in a high dimensional space.

Since *Searched* may be a conjunction of numerous variables, each of them possibly having a lot of values or even being continuous, it is seldom possible to exhaustively compute $\mathbf{P}(\mathit{Searched} \mid \mathit{Known})$. One may then decide either to build an approximate representation of this distribution or to directly sample from this distribution. In both cases the challenge is to find the modes where most of the probability density is concentrated. This may be very difficult, as most of the probability may be concentrated in very small sub-spaces of the whole searched space.

The situation is even worse, as computing the value of $\mathbf{P}(\mathit{Searched} \mid \mathit{Known})$ for a given value of *Searched* (a single point of the searched space of the preceeding paragraph) is by itself a difficult problem. Indeed, it supposes to marginalize the joint distribution on the space defined by *Unknown*. *Unknown* (like *Searched*) may be a conjunction of numerous variables, each of them possibly having a lot of values or even being continuous. Consequently, the sum should also be either approximated or sampled. The challenge is then to find the modes of

$$\mathbf{P}(L^1) \times \prod_{i=2}^{k} \mathbf{P}(L^i \mid R^i) \qquad \text{[E3.12]}$$

(on the search space defined by *Unknown*), where most of the probability density is concentrated and which mostly contribute to the sum. Finally, marginalizing in a high dimensional space appears to be a very similar problem to searching the modes in a high dimensional space.

It is well known that general Bayesian inference is a very difficult problem, which may be practically intractable. Exact inference has been proved to be NP-hard (Cooper, 1990) and the general problem of approximate inference too (Dagum and Luby, 1993).

However, approximate inference is often tractable in practical cases for three main reasons:

1. The conditional independencies, as expressed by the decomposition of the joint distribution, break the complexity of the problem by reducing drastically the size of the searched space (see Section 6 for an instance of that). The importance of the decomposition has already been stressed by many authors (e.g., Zhang and Poole, 1996) and explains mainly the good performances of our engine (10 inferences per second[5]).
2. Some powerful symbolic simplifications can be made before any numerical computation (see next section on ProBT).
3. Numerical optimization and marginalization have a long history and impressive numerical methods have been developed which can be reused in this context (see next section on ProBT).

3.4.2. ProBT: An API to Automate Bayesian Inference. An inference engine and the associated programming API[6] (named PRoBT for Open Probabilistic Language) has been developed and used for the experiments presented in this paper and other industrial applications.

PRoBT proceeds in two phases: a symbolic simplification of the required computation followed by some intensive numerical crunching.

The main goal of the simplification phase is to reduce the number of sums necessary to compute the distribution:

$$\mathbf{P}(\mathit{Searched} \mid \mathit{Known}) = \frac{1}{\Sigma} \times \sum_{\text{Unknown}} \left[\mathbf{P}(L^1) \times \prod_{i=2}^{k} \mathbf{P}(L^i \mid R^i) \right] \quad \text{[E3.13]}$$

These kinds of simplification techniques are largely used in the litterature. For instance, the well known JLO or junction tree algorithm (Jensen, 1990) may be seen as such a simplification technique in the case of Bayesian Networks.

In ProBT, a large spectrum of such simplifications is used.

First, considering the different terms of the product $\mathbf{P}(L^1) \times \prod_{i=2}^{k} \mathbf{P}(L^i \mid R^i)$, three possibilities of obvious simplifications may appear:

1. When a term is a uniform distribution it can be simplified: it vanishes from the expression and its value will implicitly be taken into account in the normalization constant Σ.
2. When a term is a distribution where all the variables have *Known* values, then it is a constant for this question and may also be simplified.
3. When a term is a distribution where all the variables are either *Searched* or *Known*, then it can be factorized out of the sum.

After these three first steps, we get a new expression of the form:

$$\mathbf{P}(\mathit{Searched} \mid \mathit{Known}) = \frac{1}{\Sigma} \times \prod_{f \in j} \mathbf{P}(L^j \mid R^j) \times \sum_{\text{Unknown}} \left[\prod_{i \in I} \mathbf{P}(L^i \mid R^i) \right] \quad \text{[E3.14]}$$

Now, considering $\sum_{\text{Unknown}} [\prod_{i \in I} \mathbf{P}(L^i \mid R^i)]$, we can try to find an order on the sum to simplify terms that sum to 1.

Indeed, when a term $\mathbf{P}(L^i \mid R^i)$ appears in the sum, if all the variables appearing in L^i are part of *Unknown* (summed) and if all the variables appearing in R^i are either part of *Known* or *Unknown*, then $\mathbf{P}(L^i \mid R^i)$ sums to 1 and vanishes out of the global sum. This operation often leads to impressive simplifications.

Finally, the last simplification that can be made is to reorder the sums on the different unknown variables in order to minimize the number of operations to make. ProBT uses the general distributive law algorithm to do this. A description of this algorithm may be found in a paper by Aji and McEliece (2000).

A more detailed description of this simplification phase and of related work may be found in Bessière's survey (Bessière et al., 2003).

The main goal of the numerical crunching phase is to estimate the distribution $\mathbf{P}(\mathit{Searched} \mid \mathit{Known})$. A necessary subgoal is to estimate the corresponding sum.

Two main approaches are possible to reach these objectives, either by building approximated explicit representation of these distributions or by sampling these distributions.

ProBT includes different algorithms related to both approaches. It may approximate the distribution using either particle filters (Arulampalam et al., 2001) or Multi Resolution Binary Trees (MRBT), a homemade representation described in a pending patent (Bessière, 2002). ProBT also uses sampling techniques, mainly Monte Carlo sampling integration methods (Neal, 1993; MacKay, 1996) and an improved version of these techniques proposed by Mekhnacha et al. (2001), where they are combined with simulated annealing.

3.4.3. Decision-Making. For a given distribution, different decision policies are possible: for example, searching the best (highest probability) values or drawing at random according to the distribution. For our purposes, we will always use this second policy and refer to this query as: $\mathbf{Draw}(\mathbf{P}(\mathit{Searched} \mid \mathit{Known} \otimes \delta \otimes \pi))$.

Utility functions could also be used to make the decision but they do not appear to be necessary for the work described in this paper.

3.4.4. Control Loop of the Robot. To control our robot using a Bayesian program, a decision is made every tenth of a second. A typical question is to select the values of the motor variables knowing the values of the sensory variables. Consequently, the basic loop to operate the robot is to loop on the following instructions every tenth of a second:

1. Read the values of the sensors
2. **Draw**(**P**(*Motors* | *Sensors* $\otimes \delta \otimes \pi$))
3. Send the returned values to the motors

4. Experimental Platform

4.1. Khepera Robot

Khepera is a two-wheeled mobile robot, 57 millimeters in diameter and 29 millimeters in height, with a total weight of 80 g (See Fig. 2). It was designed at EPFL[7] and is commercialized by K-Team.[8]

The robot is equipped with eight light sensors (six in front and two behind), taking values between 0 and 511 in inverse relation to light intensity, stored in variables $L1, \ldots, L8$ (see Fig. 3). These eight sensors can also be used as infrared proximeters, taking values between 0 and 1023 in inverse relation to the distance from the obstacle, stored in variables $Px1, \ldots, Px8$ (see Fig. 3).

The robot is controlled by the rotation speeds of its left and right wheels, stored in variables *Mg* and *Md*, respectively.

From these 18 basic sensory and motor variables, we derived three new sensory variables (*Dir*, *Prox* and *Theta1*) and one new motor one (*Vrot*). They are described below.

Figure 2. The Khepera mobile robot.

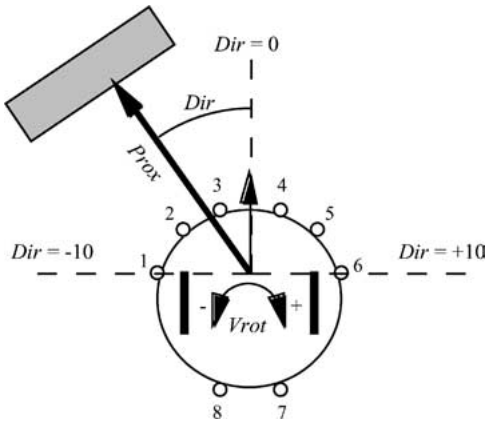


Figure 3. The sensory-motor variables of the Khepera robot.

- *Dir* is a variable that approximately corresponds to the bearing of the closest obstacle (see Fig. 3). It takes values between -10 (obstacle to the left of the robot) and $+10$ (obstacle to the right of the robot), and is defined as follows:

$$Dir = \mathbf{Floor}\left(\frac{90(Px6 - Px1) + 45(Px5 - Px2) + 5(Px4 - Px3)}{9(1 + Px1 + Px2 + Px3 + Px4 + Px5 + Px6)}\right) \quad \text{[E4.1]}$$

- *Prox* is a variable that approximately corresponds to the proximity of the closest obstacle (See Fig. 3). It takes values between zero (obstacle very far from the robot) and 15 (obstacle very close to the robot), and is defined as follows:

$$Prox = \mathbf{Floor}\left(\frac{\mathbf{Max}(Px1, Px2, Px3, Px4, Px5, Px6)}{64}\right) \quad \text{[E4.2]}$$

- *Theta1* is a variable that approximately corresponds to the bearing of the greatest source of illumination. It takes on 36 values from -170° to 180°.
- The robot is piloted solely by its rotation speed (the translation speed is fixed). It receives motor commands from the *Vrot* variable, calculated from the difference between the rotation speeds of the left and right wheels. *Vrot* takes on values between $+10$ (fastest to the right) and -10 (fastest to the left).

Khepera accepts turrets on its top to augment either its sensory or motor capacities. For the final experiment

(the nightwatchman task), a linear camera of 64 pixels and a micro turbine were added on top of the robot.

4.2. Environment

For all experiments described in the current paper, the Khepera is placed in a 1 m by 1 m environment. This environment has walls around its contour, textured to be easily seen by the robot. Inside this square, we place walls made of Lego© bricks that can be moved easily to set any configuration we need quickly. We usually build a recess made of high Lego© walls in a corner, and place a small light over this recess, to create a "base" for the robot (see Fig. 12).

5. Reactive Behavior

5.1. Goal and Experimental Protocol

The goal of the first experiment was to teach the robot how to push objects.

First, in a learning phase, we drove the robot with a joystick to push objects. During that phase, the robot collected, every tenth of a second, both the values of its sensory variables and the values of its motor variables (determined by the joystick position). This data set was then used to identify the free parameters of the parametric forms.

Then, in a restitution phase, the robot has to reproduce the behavior it had just learned. Every tenth of a second it decided the values of its motor variables, knowing the values of its sensory variables and the internal representation of the task.

5.2. Specification

Having defined our goal, we describe the three steps necessary to define the preliminary knowledge.

```
1. Choose the pertinent variables
2. Decompose the joint distribution
3. Define the parametric forms
```

5.2.1. Variables. First, the programmer specifies which variables are pertinent for the task.

To push objects it is necessary to have an idea of the position of the objects relative to the robot. The front proximeters provide this information. However, we chose to sum up the information of these six proximeters by the two variables *Dir* and *Prox*.

We also chose to set the translation speed to a constant and to operate the robot by its rotation speed *Vrot*.

These three variables are all we need to push obstacles. Their definitions are summed up as follows:

$$\begin{aligned} Dir &\in \{-10, \ldots, 10\}, \lfloor Dir \rfloor = 21 \\ Prox &\in \{0, \ldots, 15\}, \lfloor Prox \rfloor = 16 \\ Vrot &\in \{-10, \ldots, 10\}, \lfloor Vrot \rfloor = 21 \end{aligned} \qquad [S5.1]$$

5.2.2. Decomposition. In the second specification step, we give a decomposition of the joint probability $\mathbf{P}(Dir \otimes Prox \otimes Vrot \mid \Delta \otimes \pi\text{-}obstacle)$ as a product of simpler terms. This distribution is conditioned by both π-*obstacle*, the preliminary knowledge we are defining, and Δ a data set that will be provided during the learning phase.

$$\begin{aligned} &\mathbf{P}(Dir \otimes Prox \otimes Vrot \mid \Delta \otimes \pi\text{-}obstacle) \\ &\quad = \mathbf{P}(Dir \mid \Delta \otimes \pi\text{-}obstacle) \\ &\qquad \times \mathbf{P}(Prox \mid Dir \otimes \Delta \otimes \pi\text{-}obstacle) \\ &\qquad \times \mathbf{P}(Vrot \mid Prox \otimes Dir \otimes \Delta \otimes \pi\text{-}obstacle) \\ &\quad = \mathbf{P}(Dir \mid \Delta \otimes \pi\text{-}obstacle) \\ &\qquad \times \mathbf{P}(Prox \mid \Delta \otimes \pi\text{-}obstacle) \\ &\qquad \times \mathbf{P}(Vrot \mid Prox \otimes Dir \otimes \Delta \otimes \pi\text{-}obstacle) \end{aligned} \qquad [S5.2]$$

The first equality results from the application of the product rule (Eq. [E3.4]). The second results from the simplification $\mathbf{P}(Prox \mid Dir \otimes \pi\text{-}obstacle) = \mathbf{P}(Prox \mid \Delta \otimes \pi\text{-}obstacle)$, which means that we consider that *Prox* and *Dir* are independent. The distances to the objects and their bearings are not contingent.

5.2.3. Parametric Forms. To be able to compute the joint distribution, we finally need to assign parametric forms to each of the terms appearing in the decomposition:

$$\begin{aligned} &\mathbf{P}(Dir \mid \Delta \otimes \pi\text{-}obstacle) \equiv Uniform \\ &\mathbf{P}(Prox \mid \Delta \otimes \pi\text{-}obstacle) \equiv Uniform \\ &\mathbf{P}(Vrot \mid Prox \otimes Dir \otimes \Delta \otimes \pi\text{-}obstacle) \\ &\quad \equiv \mathbf{G}(\mu(Prox, Dir), \sigma(Prox, Dir)) \end{aligned} \qquad [S5.3]$$

We have no *a priori* information about the direction and the distance of the obstacles. Hence, $\mathbf{P}(Dir \mid \Delta \otimes \pi\text{-}obstacle)$ and $\mathbf{P}(Prox \mid \Delta \otimes \pi\text{-}obstacle)$ are uniform distributions; all directions and proximities have the same probability.

For each sensory situation, we believe that there is one and only one rotation speed that should be preferred. The distribution $\mathbf{P}(Vrot \mid Prox \otimes Dir \otimes \Delta \otimes \pi\text{-}obstacle)$ is unimodal. However, depending of the situation, the decision to be made for *Vrot* may be more or less certain. This is resumed by assigning a Gaussian parametric form to $\mathbf{P}(Vrot \mid Prox \otimes Dir \otimes \Delta \otimes \pi\text{-}obstacle)$.

5.3. Identification

We drive the robot with a joystick (see Movie 1[9]), and collect a set of data Δ. Let us call the particular set of data corresponding to this experiment δ-*push*. A datum collected at time t is a triplet $(vrot_t, dir_t, prox_t)$.

The free parameters of the parametric forms (means and standard deviations for all the $\lfloor Dir \rfloor \times \lfloor Prox \rfloor$ Gaussians) can then be identified by computing the means and standard deviations of *Vrot* for each position of the obstacle.

Finally, it is possible to compute the joint distribution:

$$\begin{aligned}&\mathbf{P}(Dir \otimes Prox \otimes Vrot \mid \delta\text{-}push \otimes \pi\text{-}obstacle)\\ &= \mathbf{P}(Dir \mid \pi\text{-}obstacle) \times \mathbf{P}(Prox \mid \pi\text{-}obstacle)\\ &\quad \times \mathbf{P}(Vrot \mid Prox \otimes Dir \otimes \delta\text{-}push \otimes \pi\text{-}obstacle)\end{aligned} \quad \text{[E5.1]}$$

According to Eq. [E3.11], the robot can answer any question concerning this joint distribution.

We call the distribution $\mathbf{P}(Dir \otimes Prox \otimes Vrot \mid \delta\text{-}push \otimes \pi\text{-}obstacle)$ a *description* of the task. A description is the result of identifying the free parameters of a preliminary knowledge using some given data. Hence, a description is completely defined by a couple preliminary knowledge + data. That is why a conjunction $\delta \otimes \pi$ always appears to the right of a description.

5.4. Utilization

To render the pushing obstacle behavior just learned, the Bayesian controller is called every tenth of a second:

1. The sensors are read and the values of dir_t and $prox_t$ are computed.
2. The Bayesian program is run with the query:

$$\mathbf{Draw}(\mathbf{P}(Vrot \mid prox_t \otimes dir_t \otimes \delta\text{-}push \otimes \pi\text{-}obstacle)) \quad \text{[E5.2]}$$

3. The drawn $vrot_t$ is sent to the motors.

5.5. Results, Lessons and Comments

5.5.1. Results. As shown in Movie 1,[9] the Khepera learns how to push obstacles in 20 to 30 seconds. It learns the particular dependency, corresponding to this specific behavior, between the sensory variables *Dir* and *Prox* and the motor variable *Vrot*.

This dependency is largely independent of the particular characteristics of the objects (weight, color, balance, nature, etc.). Therefore, as shown in Movie 2,[10] the robot is also able to push different objects. This, of course, is only true within certain limits. For instance, the robot will not be able to push the object if it is too heavy.

Lesson 1: A generic method for Bayesian Robot Programming

In this experiment we apply a precise three-step method to program the robot.

1. *Specification*: define the preliminary knowledge.
   - 1.1 Choose the pertinent variables.
   - 1.2 Decompose the joint distribution.
   - 1.3 Define the parametric forms.
2. *Identification*: identify the free parameters of the preliminary knowledge.
3. *Utilization*: ask a question to the joint distribution.

In the sequel, we will use the very same method for all the other BRP experiments.

Lesson 2: Bayesian Program = Preliminary Knowledge + Data + Question

Numerous different behaviors may be obtained by changing some of the different components of a Bayesian program in the following ways.

- It is possible to *change the question*, keeping the description unchanged. For instance, if the *Prox* information is no longer available because of some

Figure 4. Contour following (superposed images).

failure, the robot may still try to push the obstacles knowing only their direction. The query is then:

$$\mathbf{Draw}(\mathbf{P}(Vrot \mid dir_t \otimes \delta\text{-}push \otimes \pi\text{-}obstacle)) \qquad \text{[E5.3]}$$

- It is possible to *change the data*, keeping the preliminary knowledge unchanged. For instance, with the same preliminary knowledge π-*obstacle*, we taught the robot to avoid objects or to follow their contour (see Fig. 4 and Movie 3[11]). Two new descriptions[12] were obtained by changing only the driving of the robot during the learning phase. As a result, two new programs were obtained leading to the expected behaviors: "obstacle avoidance" and "contour following."
- Finally, it is possible to *change the preliminary knowledge*, which leads to completely different behaviors. Numerous examples will be presented in the sequel of this paper. For instance, we taught the robot another reactive behavior called phototaxy. Its goal is then to move toward a light source. This new preliminary knowledge π-*phototaxy1* uses the variables *Vrot* and *Theta1*. *Theta1* roughly corresponds to the direction of the light.

6. Sensor Fusion

6.1. Goal and Experimental Protocol

The goal of this experiment is to fuse the data originating from the eight light sensors to determine the position of a light source.

This will be obtained in two steps. In the first one, we specify one description for each sensor individually. In the second one, we mix these eight descriptions to form a global one.

6.2. Sensor Model

6.2.1. Specification.

6.2.1.1. Variables. To build a model of the light sensor i, we only require two variables: Li the reading of the ith sensor, and *Theta2*, the bearing of the light source.

$$\begin{array}{c} Li \in \{0, \ldots, 511\}, \lfloor Li \rfloor = 512 \\ Theta2 \in \{-170, \ldots, 180\}, \lfloor Theta2 \rfloor = 36 \end{array} \qquad \text{[S6.1]}$$

6.2.1.2. Decomposition. The decomposition simply specifies that the reading of a sensor obviously depends on the position of the light source

$$\begin{aligned} &\mathbf{P}(Theta2 \otimes Li \mid \Delta \otimes \pi\text{-}sensor) \\ &\quad = \mathbf{P}(Theta2 \mid \pi\text{-}sensor) \\ &\qquad \times \mathbf{P}(Li \mid Theta2 \otimes \Delta \otimes \pi\text{-}sensor) \end{aligned} \qquad \text{[S6.2]}$$

6.2.1.3. Parametric Forms. As we have no *a priori* information on the position of the source, we state:

$$\mathbf{P}(Theta2 \mid \pi\text{-}sensor) \equiv Uniform \qquad \text{[S6.3]}$$

The distribution $\mathbf{P}(Li \mid Theta2 \otimes \Delta \otimes \pi\text{-}sensor)$ is usually very easy to specify because it corresponds exactly to the kind of information that the sensor supplier provides: the expected readings of its device when exposed to a light. For Khepera's light sensors, we obtain (see Fig. 5):

$$\begin{aligned} &\mathbf{P}(Li \mid Theta2 \otimes \pi\text{-}sensor) \equiv \mathbf{G}_{\mathbf{K}(Theta2,\theta_i),\sigma}(Li) \\ &\mathbf{K}(Theta2, \theta_i) = 1 - \frac{1}{1 + e^{-4\beta(|Theta2-\theta_i|-\alpha)}} \\ &\qquad (\alpha = 45), (\beta = 0.03) \end{aligned} \qquad \text{[S6.4]}$$

In specification [S6.4], θ_i stands for the position of the sensor with respect to the robot, and will be used later to "rotate" this model for different sensors.

Specifications [S6.1], [S6.2], [S6.3] and [S6.4] are the preliminary knowledge corresponding to this sensor model. This preliminary knowledge is named π-*sensor*.

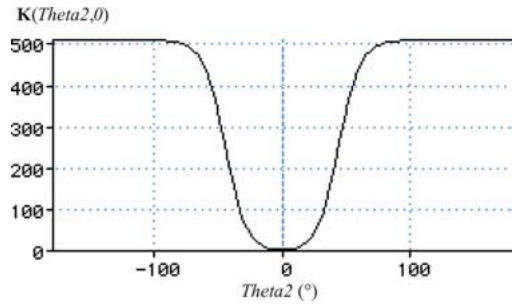


Figure 5. **K**(*Theta2*, 0).

6.2.2. Identification. No identification is required as there are no free parameters in π*-sensor*.

However, it may be easy and interesting to calibrate specifically each of the eight light sensors. This could be achieved, for instance, by identifying parameters α and β independently for each sensor, by observing the response of the particular sensor to a light source.

6.3. Fusion

6.3.1. Specification.

6.3.1.1. Variables. The interesting variables are the eight variables *Li* and *Theta2*:

$$\begin{gathered} L1 \in \{0, \ldots, 511\}, \lfloor L1 \rfloor = 512 \\ \cdots \\ L8 \in \{0, \ldots, 511\}, \lfloor L8 \rfloor = 512 \\ Theta2 \in \{-170, \ldots, 180\}, \lfloor Theta2 \rfloor = 36 \end{gathered} \quad [S6.5]$$

6.3.1.2. Decomposition. The decomposition of the joint distribution is chosen to be:

$$\begin{aligned} &\mathbf{P}(Theta2 \otimes L1 \otimes L2 \otimes L3 \otimes L4 \otimes L5 \otimes L6 \\ &\quad \otimes L7 \otimes L8 \mid \Delta \otimes \pi\text{-}fusion) \\ &= \mathbf{P}(Theta2 \mid \Delta \otimes \pi\text{-}fusion) \times \mathbf{P}(L1 \mid Theta2 \otimes \Delta \\ &\quad \otimes \pi\text{-}fusion) \times \mathbf{P}(L2/L1 \otimes Theta2 \otimes \Delta \otimes \pi\text{-}fusion) \\ &\quad \times \ldots \times \mathbf{P}(L8 \mid L7 \otimes L6 \otimes L5 \otimes L4 \otimes L3 \otimes L2 \\ &\quad \otimes L1 \otimes Theta2 \otimes \Delta \otimes \pi\text{-}fusion) \qquad [S6.6] \\ &= \mathbf{P}(Theta2 \mid \pi\text{-}fusion) \\ &\quad \times \prod_{i=1}^{8} \mathbf{P}(Li \mid Theta2 \otimes \Delta \otimes \pi\text{-}fusion) \end{aligned}$$

The first equality results from the product rule [E3.4]. The second from simplifications of the kind:

$$\begin{aligned} &\mathbf{P}(Lj \mid Lj-1 \otimes \cdots \otimes L1 \otimes Theta2 \otimes \Delta \otimes \pi\text{-}fusion) \\ &\quad = \mathbf{P}(Lj \mid Theta2 \otimes \Delta \otimes \pi\text{-}fusion) \qquad [E6.1] \end{aligned}$$

These simplifications may seem peculiar as obviously the readings of the different light sensors are not independent. The exact meaning of these equations is that we consider *Theta2* (the position of the light source) to be the main reason for the contingency of the readings. Consequently, we state that, knowing *Theta2*, the readings *Lj* are independent. *Theta2* is the cause of the readings and knowing the cause, the consequences are independent. This is, indeed, a very strong hypothesis. The sensors may be correlated for numerous other reasons. For instance, ambient temperature influences the functioning of any electronic device and consequently correlates their responses. However, we choose, as a first approximation, to disregard all these other factors.

6.3.1.3. Parametric Forms. We do not have any *a priori* information on *Theta2*:

$$\mathbf{P}(Theta2 \mid \pi\text{-}fusion) \equiv Uniform \qquad [S6.7]$$

$\mathbf{P}(Li \mid Theta2 \otimes \Delta \otimes \pi\text{-}fusion)$ is obtained from the model of each sensor as specified in previous Section (6.2):

$$\begin{aligned} &\mathbf{P}(Li \mid Theta2 \otimes \Delta \otimes \pi\text{-}fusion) \\ &\quad \equiv \mathbf{P}(Li \mid Theta2 \otimes \pi\text{-}sensor) \qquad [S6.8] \end{aligned}$$

6.3.2. Identification. As there are no free parameters in π*-fusion*, no identification is required.

6.3.3. Utilization. To find the position of the light source the standard query is:

$$\mathbf{Draw}(\mathbf{P}(Theta2 \mid l1_t \otimes \cdots \otimes l8_t \otimes \pi\text{-}fusion)) \qquad [E6.2]$$

This question may be easily answered using Eq. [E3.11] and specification [S6.8]:

$$\begin{aligned} &\mathbf{P}(Theta2 \mid l1_t \otimes \cdots \otimes l8_t \otimes \pi\text{-}fusion) \\ &\quad = \frac{1}{\Sigma} \times \prod_{i=1}^{8} \mathbf{P}(li_t \mid Theta2 \otimes \pi\text{-}sensor) \qquad [E6.3] \end{aligned}$$

Values drawn from this distribution may be efficiently computed given that the distribution $\mathbf{P}(Theta2 \mid l1_t \otimes \cdots \otimes l8_t \otimes \pi\text{-}fusion)$ is simply a product of eight very simple ones, and given that the normalizing constant Σ does not need to be computed for a random draw.

Many other interesting questions may be asked of this description, as the following:

- It is possible to search for the position of the light source knowing only the readings of a few sensors:

$$\mathbf{P}(Theta2 \mid l1_t \otimes l2_t \otimes \pi\text{-}fusion) = \frac{1}{\Sigma} \times \mathbf{P}(l1_t \mid Theta2 \otimes \pi\text{-}sensor) \times \mathbf{P}(l2_t \mid Theta2 \otimes \pi\text{-}sensor) \quad \text{[E6.4]}$$

- It is possible to check whether the sensor i is out of order. Indeed, if its reading li_t at time t, persists in being inconsistent with the readings of the others for some period, it is a good indication of a malfunction. This inconsistency may be detected by a very low probability for li_t:

$$\mathbf{P}(l1_t \mid l2_t \otimes \cdots \otimes l8_t \otimes \pi\text{-}fusion) = \frac{1}{\Sigma} \times \sum_{Theta2} \prod_{i=1}^{8} \mathbf{P}(li_t \mid Theta2 \otimes \pi\text{-}sensor) \quad \text{[E6.5]}$$

6.4. *Results, Lessons and Comments*

6.4.1. Results. Figure 6 presents the result obtained for a light source with a bearing of 10°.

The eight peripheral figures present the distributions $\mathbf{P}(Theta2 \mid Li \otimes \pi\text{-}sensor)$ corresponding to the eight light sensors. The central schema presents the result of the fusion, the distribution $\mathbf{P}(Theta2 \mid l1_t \otimes \cdots \otimes l8_t \otimes \pi\text{-}fusion)$. Even poor information coming from each separate sensor may blend as a certainty.

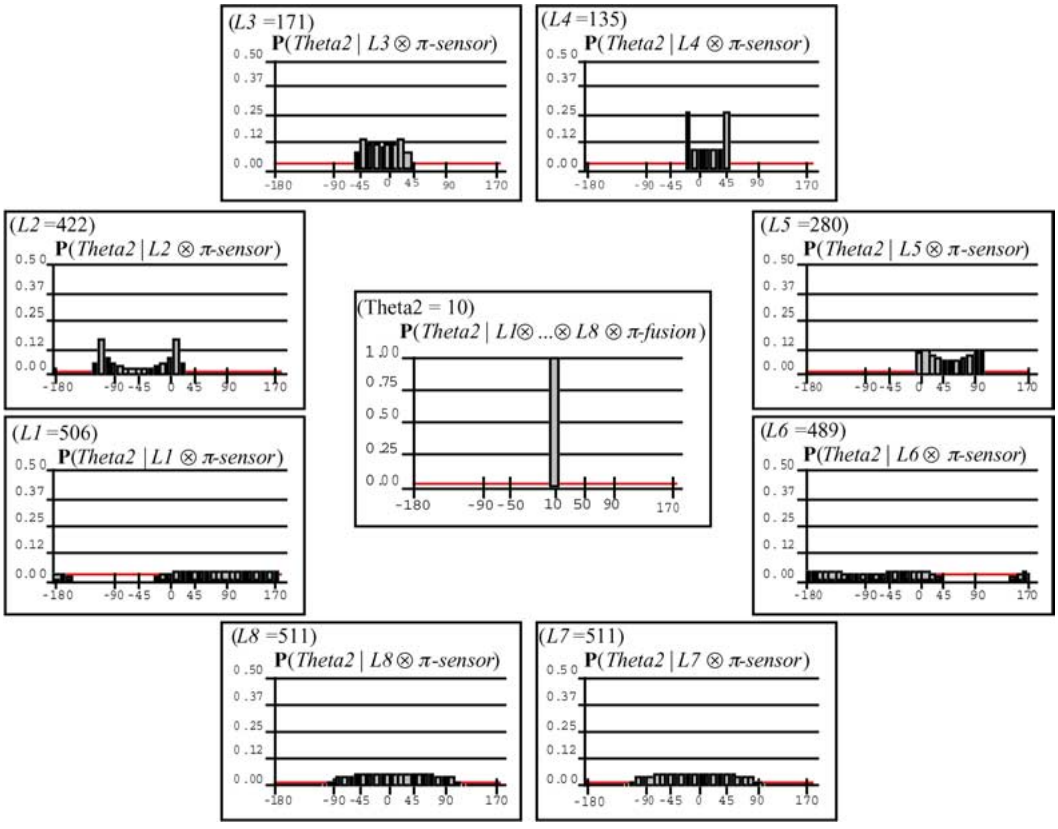


Figure 6. The result of a sensor fusion for a light source with a bearing of 10°.

Lesson 3: Breaking the complexity using conditional independencies

The conditional independencies hypothesis which permits to transform:

$$\mathbf{P}(Theta2 \otimes L1 \otimes L2 \otimes L3 \otimes L4 \otimes L5 \otimes L6 \otimes L7 \otimes L8 \mid \Delta \otimes \pi\text{-}fusion) \quad \text{[E6.6]}$$

into:

$$\mathbf{P}(Theta2 \mid \pi\text{-}fusion) \times \prod_{t=1}^{8} \mathbf{P}(Li \mid Theta2 \otimes \Delta \otimes \pi\text{-}fusion) \quad \text{[E6.7]}$$

is the main tool at hand to simplify the treated problem. More than any clever inference algorithm they are the essential way to keep computation tractable. For instance, here the size of the search space for the joint distribution [E6.7] is $36 \times 512^8 \propto 2^{79}$, when the size of the search space for the decomposition [E6.7] is $36 \times (512 \times 36) \times 8 \propto 2^{22}$.

Lesson 4: Calling Bayesian subroutines

Specification [S6.8]: $\mathbf{P}(Li \mid Theta2 \otimes \Delta \otimes \pi\text{-}fusion) \equiv \mathbf{P}(Li \mid Theta2 \otimes \pi\text{-}sensor)$, where a distribution appearing in a decomposition is defined by a question to another Bayesian program, may be seen as the probabilistic anologous to subroutine calls in regular programming.

This Bayesian subroutine call mechanism will play the same role than the usual one: allowing to build complex Bayesian programs as hierarchies of embedded calls to simpler and simpler Bayesian programming building blocks. Section 10 will present a more complex instance of this.

Lesson 5: Sensor fusion method

In the experiment just presented, we have seen a simple instance of a general method to carry out data fusion.

The key point of this method is in the decomposition of the joint distribution, which has been considerably simplified under the hypothesis that "knowing the cause, the consequences are independent". This is a very strong hypothesis, although it may be assumed in numerous cases.

This way of doing sensor fusion is very efficient. Its advantages are manifold.

- The signal is heightened.
- It is robust to a malfunction of one of the sensors.
- It provides precise information even with poor sensors.
- It leads to simple and very efficient computations.

We presented this method on a very simple case for didactic purposes. However, it is a very popular technique to do sensor fusion which can be used for much more complicated cases in a large variety of applications. For instance, we used refinement of this technique for ADAS (Advanced Driver Assistance System) to merge information coming from two radars and one lidar in order to partially automate car driving (Coué et al., 2002, 2003).

Lesson 6: No inverse and no ill-posed problems in the probabilistic framework

In this experiment, another fundamental advantage of Bayesian programming is clearly evident. The description is neither a direct nor an inverse model. Mathematically, all variables appearing in a joint distribution play exactly the same role. This is why any question may be asked of a description. Consequently one may define the description in one way ($\mathbf{P}(Li \mid Theta2)$) and question it in the opposite way ($\mathbf{P}(Theta2 \mid l1_t \otimes \cdots \otimes l8_t)$). In theory, any inverse problem may be solved when expressed in a probabilistic framework. In practice some of these inverse problems may require a lot of computational resources. However, this is a major difference with non probabilistic modeling where inverse problem may only be solved in rare cases.

Furthermore, there is none ill-posed problem in a probabilistic framework. If a question may have several solutions, the probabilistic answer will simply have several peaks.

7. Behavior Combination

7.1. Goal and Experimental Protocol

In this experiment we want the robot to go back to its base where it can recharge.

This will be obtained with no further teaching. As the robot's base is lit, the light gradient usually gives

good hints on its direction. Consequently, we will obtain the homing behavior by combining together the obstacle avoidance behavior and the phototaxy behavior. By programming this behavior we will illustrate one possible way to combine Bayesian programs that make use of a "command variable."

7.2. Specification

7.2.1. Variables. We need *Dir*, *Prox*, *Theta1* and *Vrot*, the four variables already used in the two composed behaviors. We also need a new variable H which acts as a command to switch from avoidance to phototaxy.

$$\begin{aligned} Dir &\in \{-10, \ldots, 10\}, \lfloor Dir \rfloor = 21 \\ Prox &\in \{0, \ldots, 15\}, \lfloor Prox \rfloor = 16 \\ Theta1 &\in \{-170, \ldots, 180\}, \lfloor Theta1 \rfloor = 36 \\ Vrot &\in \{-10, \ldots, 10\}, \lfloor Vrot \rfloor = 21 \\ H &\in \{avoidance, phototaxy\}, \lfloor H \rfloor = 2 \end{aligned} \quad [S7.1]$$

7.2.2. Decomposition. We believe that the sensory variables *Dir*, *Prox* and *Theta1* are independent from one another. Far from any objects, we want the robot to go toward the light. Very close to obstacles, we want the robot to avoid them. Hence, we consider that H should only depend on *Prox*. Finally, we believe that *Vrot* must depend on the other four variables. These programmer choices lead to the following decomposition:

$$\begin{aligned} &\mathbf{P}(Dir \otimes Prox \otimes Theta1 \otimes H \otimes Vrot \mid \Delta \otimes \pi\text{-}home) \\ &\quad = \mathbf{P}(Dir \mid \pi\text{-}home) \times \mathbf{P}(Prox \mid \pi\text{-}home) \\ &\qquad \times \mathbf{P}(Theta1 \mid \pi\text{-}home) \times \mathbf{P}(H \mid Prox \otimes \pi\text{-}home) \\ &\qquad \times \mathbf{P}(Vrot \mid Dir \otimes Prox \otimes Theta1 \otimes H \otimes \pi\text{-}home) \end{aligned} \quad [S7.2]$$

7.2.3. Parametric Forms. We have no *a priori* information about either the direction and distance of objects or the direction of the light source. Consequently, we state:

$$\begin{aligned} \mathbf{P}(Dir \mid \pi\text{-}home) &\equiv Uniform \\ \mathbf{P}(Prox \mid \pi\text{-}home) &\equiv Uniform \\ \mathbf{P}(Theta1 \mid \pi\text{-}home) &\equiv Uniform \end{aligned} \quad [S7.3]$$

H is a command variable to switch from avoidance to phototaxy. This means that when $H = avoidance$ the robot should behave as it learned to do in the description $\mathbf{P}(Dir \otimes Prox \otimes Vrot \mid \delta\text{-}avoid \otimes \pi\text{-}obstacle)$ and when $H = phototaxy$ the robot should behave according to the description $\mathbf{P}(Theta1 \otimes Vrot \mid \delta\text{-}phototaxy \otimes \pi\text{-}phototaxy1)$. Therefore, we state:

$$\begin{aligned} &\mathbf{P}(Vrot \mid Dir \otimes Prox \otimes Theta1 \otimes avoidance \otimes \pi\text{-}home) \\ &\quad \equiv \mathbf{P}(Vrot \mid Dir \otimes Prox \otimes \delta\text{-}avoid \otimes \pi\text{-}obstacle) \\ &\mathbf{P}(Vrot \mid Dir \otimes Prox \otimes Theta1 \otimes phototaxy \otimes \pi\text{-}home) \\ &\quad \equiv \mathbf{P}(Vrot \mid Theta1 \otimes \delta\text{-}phototaxy \otimes \pi\text{-}phototaxy1) \end{aligned} \quad [S7.4]$$

We want a smooth transition from phototaxy to avoidance as we move closer and closer to objects. Hence, we finally state:

$$\begin{aligned} &\mathbf{P}(avoidance \mid Prox \otimes \pi\text{-}home) \\ &\quad \equiv \mathbf{Sigmoid}_{\alpha,\beta}(Prox) \quad (\alpha = 9), (\beta = 0, 25) \\ &\mathbf{P}(phototaxy \mid Prox \otimes \pi\text{-}home) \\ &\quad = 1 - \mathbf{P}(avoidance \mid Prox \otimes \pi\text{-}home) \end{aligned} \quad [S7.5]$$

The discrete approximation of the Sigmoid function we use above, which will not be defined in the current paper, is shown in Fig. 7.

The preliminary knowledge π-*home* is defined by specifications [S7.1], [S7.2], [S7.3], [S7.4] and [S7.5].

7.3. Identification

There are no free parameters in preliminary knowledge π-*home*. No learning is required.

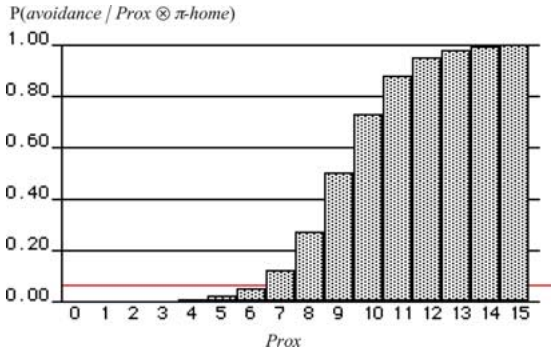


Figure 7. $\mathbf{P}(avoidance \mid Prox \otimes \pi\text{-}home)$.

7.4. Utilization

While Khepera returns to its base, we do not know in advance when it should avoid obstacles or when it should go toward the light. Consequently, to render the homing behavior we will use the following question where H is unknown:

$$\begin{aligned}&\mathbf{P}(Vrot \mid Dir \otimes Prox \otimes Theta1 \otimes \pi\text{-}home)\\ &= \sum_H \mathbf{P}(Vrot \otimes H \mid Dir \otimes Prox \otimes Theta1 \otimes \pi\text{-}home)\\ &= \frac{1}{\Sigma} \times \left[\begin{array}{l}[\mathbf{P}(avoidance \mid Prox \otimes \pi\text{-}home)\\ \quad \times \mathbf{P}(Vrot \mid Dir \otimes Prox \otimes \delta\text{-}avoid\\ \quad \otimes \pi\text{-}obstacle)]\\ +[\mathbf{P}(phototaxy \mid Prox \otimes \pi\text{-}home)\\ \quad \times \mathbf{P}(Vrot \mid Theta1 \otimes \delta\text{-}phototaxy\\ \quad \otimes \pi\text{-}phototaxy1)]\end{array}\right]\end{aligned} \quad \text{[E7.1]}$$

Equation [E7.1] shows that the robot does a weighted combination between avoidance and phototaxy. Far from any objects ($prox = 0, \mathbf{P}(phototaxy \mid prox \otimes \pi\text{-}home) = 1$) it does pure phototaxy. Very close to objects ($prox = 15, \mathbf{P}(avoidance \mid prox \otimes \pi\text{-}home) = 1$) it does pure avoidance. In between, it mixes the two.

7.5. Results, Lessons and Comments

7.5.1. Results. Figure 8 and Movie 4[13] show efficient homing behavior obtained this way.

Figures 9 and 10 present the probability distributions obtained when the robot must avoid an obstacle on the left with a light source also on the left. As the object is on the left, the robot needs to turn right to avoid it. This is what happens when the robot is close to the objects (see Fig. 9). However, when the robot is further from the object, the presence of the light source on the left influences the way the robot avoids obstacles. In that case, the robot may turn left despite the presence of the obstacle (see Fig. 10).

Figure 8. Homing behavior (The arrow points out the light source) (superposed images).

Lesson 7: A probabilistic if-then-else

In this experiment we present a simple instance of a general method to combine descriptions to obtain a new mixed behavior. This method uses a command variable H to switch from one of the composing behaviors to another. A probability distribution on H knowing some sensory variables should then be specified or learned.[14] The new description is finally used by asking questions where H is unknown. The resulting sum on the different cases of H does the mixing.

This shows that Bayesian robot programming allows easy, clear and rigorous specifications of such combinations. This seems to be an important benefit compared to some other methods that have great difficulties in mixing behaviors with one another, such as Brooks' subsumption architecture (Brooks, 1986; Maes, 1989) or neural networks.

Description combination appears to naturally implement a mechanism similar to HME[15] (Jordan and Jacobs, 1994) and is also closely related to mixture models (see McLachlan and Deep, 2000 for a reference document about mixture models and see Bessière et al., 2003 for details about the relation between description combination and mixture models).

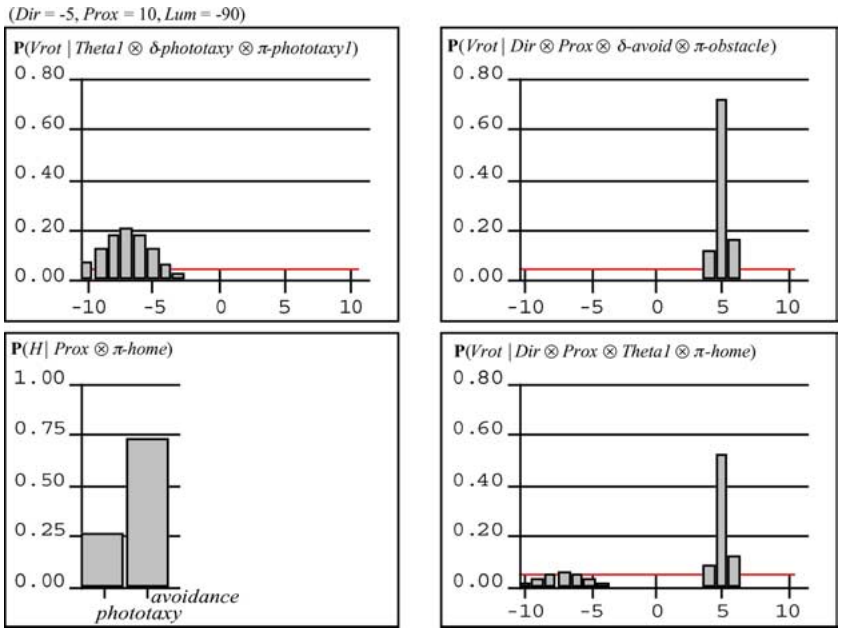


Figure 9. Homing behavior (Khepera close to an object on its left and has also the light source on its left). The top left distribution shows the knowledge on *Vrot* given by the phototaxy description; the top right is *Vrot* given by the avoidance description; the bottom left shows the knowledge of the "command variable" H; finally the bottom right shows the resulting combination on *Vrot*.

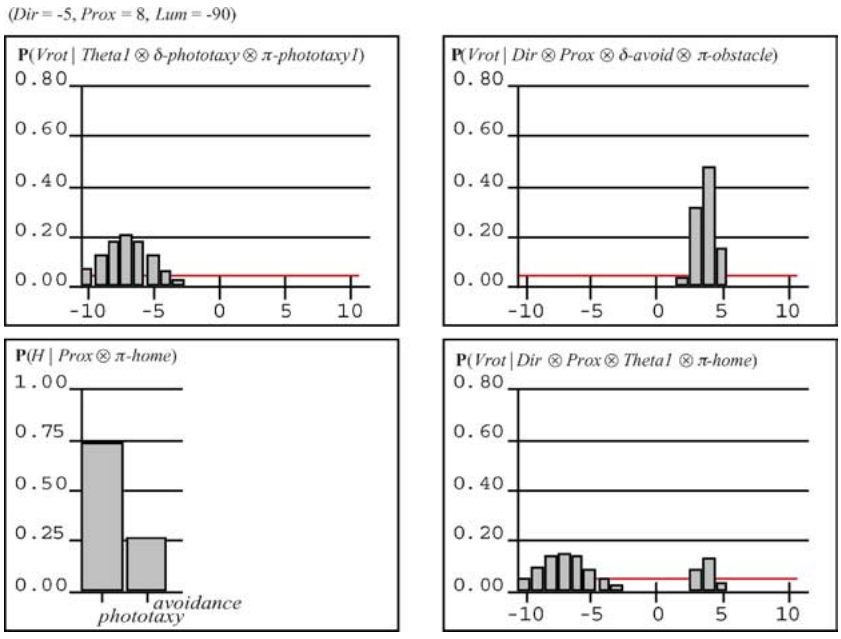


Figure 10. Homing behavior (Khepera further from the object on its left). This figure is structured as Fig. 9.

Finally, from a programming point of view, description combination can be seen as a probabilistic if-then-else. H is the condition. If H is known with certainty then we have a usual branching structure. If H is known with some uncertainty through a probability distribution then the two possible consequences are automatically mixed using weights proportional to this distribution.

8. Situation Recognition

8.1. Goal and Experimental Protocol

The goal of this experiment is to distinguish different objects from one another.

At the beginning of the experiment the robot does not know any object (see below for a precise definition). It must incrementally build categories for the objects it encounters. When it knows n of them, the robot must decide if a presented object enters in one of the n categories or if it is something new. If it is a new object, the robot must create a new category and should start to learn it.

8.2. Specification

8.2.1. Variables. Khepera does not use its camera for this task. It must "grope" for the object. It uses the "contour following" behavior to do so (see Fig. 4). It does a tour of the presented object and computes at the end of this tour four new variables: *Nlt* the number of left turns, *Nrt* the number of right turns, *Per* the perimeter and *Lrl* the longest straight line. The values of these variables are not completely determined by the shape of the object, given that the contour following behavior is quite choppy.

We also require a variable O to identify the different classes of object. The value $O = 0$ is reserved for the class of unknown (i.e., not yet presented) objects.

Finally, we obtain:

$$\begin{aligned} Nlt &\in \{0, \ldots, 24\}, \lfloor Nlt \rfloor = 25 \\ Nrt &\in \{0, \ldots, 24\}, \lfloor Nrt \rfloor = 25 \\ Per &\in \{0, \ldots, 9999\}, \lfloor Per \rfloor = 10000 \\ Lrl &\in \{0, \ldots, 999\}, \lfloor Lrl \rfloor = 1000 \\ O &\in \{0, \ldots, 15\}, \lfloor O \rfloor = 16 \end{aligned} \quad [S8.1]$$

8.2.2. Decomposition. Obviously, the four variables *Nlt*, *Nrt*, *Per* and *Lrl* are not independent of one another. However, by a reasoning similar to the sensor fusion case (see Section 6), we consider that knowing the object O, they are independent. Indeed, if the object is known, its perimeter or the number of turns necessary to complete a tour are also known. This leads to the following decomposition:

$$\begin{aligned} &\mathbf{P}(O \otimes Nlt \otimes Nrt \otimes Per \otimes Lrl \mid \Delta \otimes \pi\text{-}object) \\ &= \mathbf{P}(O \mid \pi\text{-}object) \times \mathbf{P}(Nlt \mid O \otimes \Delta \otimes \pi\text{-}object) \\ &\times \mathbf{P}(Nrt \mid O \otimes \Delta \otimes \pi\text{-}object) \\ &\times \mathbf{P}(Per \mid O \otimes \Delta \otimes \pi\text{-}object) \\ &\times \mathbf{P}(Lrl \mid O \otimes \Delta \otimes \pi\text{-}object) \end{aligned} \quad [S8.2]$$

8.2.3. Parametric Forms. We have no *a priori* information on the presented object:

$$\mathbf{P}(O \mid \pi\text{-}object) \equiv Uniform \quad [S8.3]$$

For an already observed object ($O \neq 0$), we state that the distributions on *Nlt* and *Nrl* are Laplace succession laws[16] and that the distributions on *Per* and *Lrl* are Gaussian laws:

$$\begin{aligned} &\forall o_i \in O, o_i \neq o_0 \\ &\mathbf{P}(Nlt \mid o_i \otimes \Delta \otimes \pi\text{-}object) \equiv \mathbf{L}_1(n_{Nlt}(o_i)) \\ &\mathbf{P}(Nrt \mid o_i \otimes \Delta \otimes \pi\text{-}object) \equiv \mathbf{L}_2(n_{Nrt}(o_i)) \\ &\mathbf{P}(Per \mid o_i \otimes \Delta \otimes \pi\text{-}object) \equiv \mathbf{G}_1(\mu(o_i), \sigma(o_i)) \\ &\mathbf{P}(Lrl \mid o_i \otimes \Delta \otimes \pi\text{-}object) \equiv \mathbf{G}_2(\mu(o_i), \sigma(o_i)) \end{aligned} \quad [S8.4]$$

Finally, we state that for a new object ($O = 0$) we have no *a priori* information about *Nlt*, *Nrt*, *Per* and *Lrl*:

$$\begin{aligned} \mathbf{P}(Nlt \mid o_0 \otimes \pi\text{-}object) &\equiv Uniform \\ \mathbf{P}(Nrt \mid o_0 \otimes \pi\text{-}object) &\equiv Uniform \\ \mathbf{P}(Per \mid o_0 \otimes \pi\text{-}object) &\equiv Uniform \\ \mathbf{P}(Lrl \mid o_0 \otimes \pi\text{-}object) &\equiv Uniform \end{aligned} \quad [S8.5]$$

The preliminary knowledge composed of specifications [S8.1], [S8.2], [S8.3], [S8.4] and [S8.5] is named π*-object*.

8.3. Identification

When an object is presented to the robot, if it is recognized as a member of a class o_i, the parameters of the two Laplace succession laws and the two Gaussian laws corresponding to this class are updated.

If the object is considered by Khepera to be a new one, then a new class is created and the parameters of the distributions are initialized with the values of *Nlt*, *Nrt*, *Per* and *Lrl* just read.

The learning process is incremental. Contrary to what we have seen up to this point, the identification and utilization phases are not separated. Each new experience changes the set of data Δ, and leads to a new description $\mathbf{P}(O \otimes Nlt \otimes Nrt \otimes Per \otimes Lrl \mid \delta_n \otimes \pi\text{-}object)$.

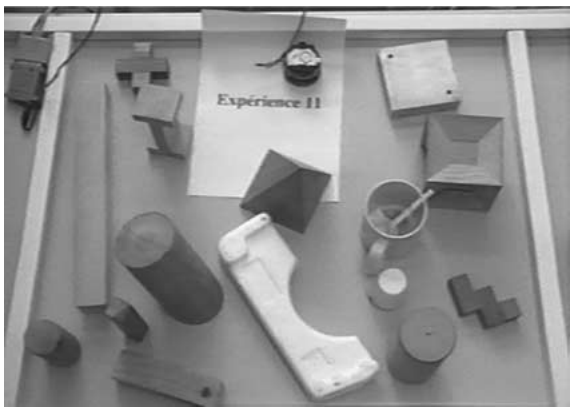


Figure 11. The different objects presented to Khepera.

8.4. *Utilization*

After $n-1$ experiences, to recognize a presented object, the question to answer is:

$$\mathbf{P}(O \mid nlt_n \otimes nrt_n \otimes per_n \otimes lrl_n \otimes \delta_{n-1} \otimes \pi\text{-}object) \quad \text{[E8.1]}$$

This may be simply computed by:

$$\begin{aligned}\mathbf{P}(O \mid Nlt \otimes Nrt \otimes Per \otimes Lrl \otimes \delta_{n-1} \otimes \pi\text{-}object) \\ = \frac{1}{\Sigma} \times \mathbf{P}(Nlt \mid O \otimes \delta_{n-1} \otimes \pi\text{-}object) \\ \times \mathbf{P}(Nrt \mid O \otimes \delta_{n-1} \otimes \pi\text{-}object) \\ \times \mathbf{P}(Per \mid O \otimes \delta_{n-1} \otimes \pi\text{-}object) \\ \times \mathbf{P}(Lrl \mid O \otimes \delta_{n-1} \otimes \pi\text{-}object) \quad \text{[E8.2]}\end{aligned}$$

If the most probable value for O is zero, then Khepera assumes that it is facing a new object. Otherwise, this most probable value is considered to correspond to the recognized object.

8.5. *Results, Lessons and Comments*

8.5.1. Results. The 15 objects shown on Fig. 11 have been presented to the robot, five times each, in random order. Each time the question was as follows: "Do you know this object, or is it a new one?" mathematicaly stated as $\mathbf{P}(O \mid nlt_n \otimes nrt_n \otimes per_n \otimes lrl_n \otimes \delta_{n-1} \otimes \pi\text{-}object)$. The obtained results are presented on Table 1.

It should be first noticed that two objects (top right of Fig. 11, 3 and 7 in the order of first presentation) have the exact same square basis and thus may not be distinguished from one another given the four chosen variables. In these 2 cases, Khepera was in the position of someone asked to identify the color of an object by groping it.

The robot did not ever fail to recognize novelty (column 1) but for the first presentation of object 7 which was recognized as object 3.

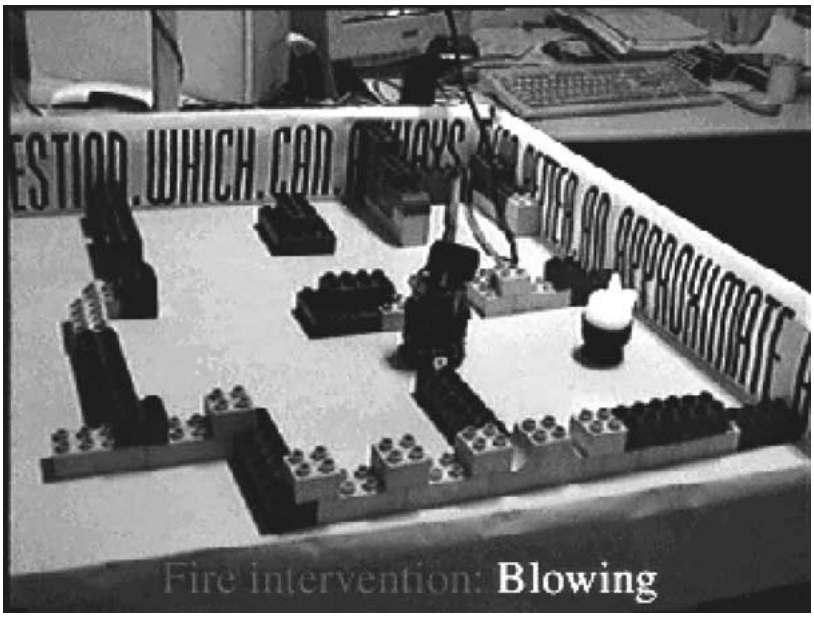


Figure 12. The night watchman task.

Table 1. Situation Recognition. The columns correspond to the "really" presented objects. The first column (F) corresponds to the first presentation of an object and the lines to the recognize objects.

	F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14															
1		4														
2			4													
3	1			4				4					1			
4					4											
5						4										
6							4									
7								0								
8									4							
9										4						
10											4					
11												4				
12													3			
13														4		
14															4	
15																4

At the second presentation of each object the robot recognized the object except for object 7 which was always identified as 3 and but once for object 12 (the wood pyramid) which was also recognized as 3.

Lesson 8: Categorization

The main lesson to retain from this experiment is that categorization of objects or situations may be considered as developing some specific sensor. Indeed, the method used in this section for object recognition is very similar to what was achieved for sensor fusion in Section 6. The hypotheses are similar and the advantages are the same.

9. Temporal Sequences

9.1. Goal and Experimental Protocol

In this section, to exemplify the Bayesian programming method, we choose a "night watchman task". This may be obtained as temporal sequences of six simpler behaviors:

1. *Idle*: The robot is at its base, recharging its batteries. It waits for both an order and enough energy to leave.
2. *Patrol*: It wanders around its environment and sounds an alarm if it detects any movement.
3. *Recognition*: The robot tours object to identify them.
4. *Fire-intervention*: Khepera tries to extinguish fires by blowing on them using its micro-turbine.
5. *Homing*: It goes to its base when ordered to do so.
6. *Recharge*: When low on energy, it goes to its base to recharge.

The purpose of this section is to show how such temporal sequences may be specified in the Bayesian framework.

9.2. Specification

9.2.1. Variables. The first variable to consider is *Behavior*, which may take the six preceding values *idle*, *patrol*, *recognition*, *fire-intervention*, *homing* and *recharge*. This variable will be used to select a given behavior.

This selection will be made according to the values of the six following variables:

- *Vigil*: a binary variable, used to order the Khepera to work.

- *Energy*: a variable that measures the level of available energy. *Energy* may take four different values: *very-high*, *high*, *low* and *very-low.*
- *Base*: a binary variable, true if the robot is at its base.
- *Fire*: a binary variable, true if the robot detects any fire.
- *Identify*: a binary variable, used to order Khepera to recognize an object.
- Finally, *Behavior_t-1* a variable taking the same six values as *Behavior*, used to memorize which behavior was selected at time $t-1$.

This may be summed up as usual:

$$\begin{aligned} Behavior &\in \{idle, patrol, recognition, \\ &\quad fire\text{-}intervention, homing, \\ &\quad recharge\}, \lfloor Behavior \rfloor = 6 \\ Vigil &\in \{true, false\}, \lfloor Vigil \rfloor = 2 \\ Energy &\in \{very\text{-}high, high, low, \\ &\quad very\text{-}low\}, \lfloor Energy \rfloor = 4 \\ Base &\in \{true, false\}, \lfloor Base \rfloor = 2 \\ Fire &\in \{true, false\}, \lfloor Fire \rfloor = 2 \\ Identify &\in \{true, false\}, \lfloor Identify \rfloor = 2 \\ Behavior\_t\text{-}1 &\in \{idle, patrol, \ldots, recharge\}, \\ &\quad \lfloor Behavior\_t\text{-}1 \rfloor = 6 \end{aligned} \quad [S9.1]$$

9.2.2. Decomposition. At each time step the robot will select a behavior knowing the values of these six variables by answering the question:

$$\mathbf{P}(Behavior \mid Vigil \otimes Energy \otimes Base \otimes Fire \otimes Identify \otimes Behavior\_t\text{-}1 \otimes \pi\text{-}behavior) \quad [E9.1]$$

It is tempting to specify this distribution directly. It would correspond to the usual programming method where the conditions at time $t-1$ establish what should be done at time t.

We propose to do the exact opposite. Indeed, it is quite easy, knowing the behavior, to have some notion of the possible values of the variables *Vigil*, *Energy*, *Base*, *Fire*, and *Identify*. For instance, if the Khepera is patrolling, it means that it has been necessarily ordered to do so and that *Vigil* is *true*. Furthermore, we consider that knowing the behavior, these five variables are independent. These assumptions lead to the following decomposition:

$$\begin{aligned} &\mathbf{P}(Behavior \otimes Vigil \otimes Energy \otimes Base \otimes Fire \\ &\quad \otimes Identify \otimes Behavior\_t\text{-}1 \mid \pi\text{-}behavior) \\ &= \mathbf{P}(Behavior\_t\text{-}1 \mid \pi\text{-}behavior) \\ &\quad \times \mathbf{P}(Behavior \mid Behavior\_t\text{-}1 \otimes \pi\text{-}behavior) \\ &\quad \times \mathbf{P}(Vigil \mid Behavior \otimes \pi\text{-}behavior) \\ &\quad \times \mathbf{P}(Energy \mid Behavior \otimes \pi\text{-}behavior) \\ &\quad \times \mathbf{P}(Base \mid Behavior \otimes \pi\text{-}behavior) \\ &\quad \times \mathbf{P}(Fire \mid Behavior \otimes \pi\text{-}behavior) \\ &\quad \times \mathbf{P}(Identify \mid Behavior \otimes \pi\text{-}behavior) \end{aligned} \quad [S9.2]$$

9.2.3. Parametric Forms. First we choose a uniform *a priori* value for $\mathbf{P}(Behavior\_t\text{-}1 \mid \pi\text{-}behavior)$:

$$\mathbf{P}(Behavior\_t\text{-}1 \mid \pi\text{-}behavior) \equiv Uniform \quad [S9.3]$$

We chose to specify all the other terms of this decomposition as discrete distributions. Their different values will be given *a priori*, one by one, using tables.

For instance, $\mathbf{P}(Behavior \mid Behavior\_t\text{-}1 \otimes \pi\text{-}behavior)$ is specified by Table 2.

This table should be read by column. Each column corresponds to the probability of *Behavior* knowing a certain behavior of the robot at time $t-1$. Consequently, each column should sum to 1 to respect the normalization constraint.

Table 2. $\mathbf{P}(Behavior \mid Behavior\_t\text{-}1 \otimes \pi\text{-}behavior)$.

Behavior/ Behavior_t-1	*idle*	*patrol*	*recognition*	*fire-interv.*	*homing*	*recharge*
idle	0.9	×	0	0	×	×
patrol	×	0.9	×	×	×	×
recognition	0	×	0.99	0	×	×
fire-interv.	×	×	×	×	×	×
homing	0	×	×	0	0.9	×
recharge	0	×	×	×	×	0.9

Table 3. **P**(*Vigil* | *Behavior* ⊗ π-*behavior*).

Vigil/ Behavior	*idle*	*patrol*	*recognition*	*fire-interv.*	*homing*	*recharge*
false	0.9	0	0	×	1	×
true	0.1	1	1	×	0	×

For instance, the first column of Table 2 specifies the probabilities of the variable *Behavior* knowing that the behavior of the robot at time $t-1$ was *idle*. If Khepera was *idle*, then it may stay *idle* with a high probability (90%), it may not directly change its behavior to either *recognition*, *homing* or *recharge* (probability 0), it may switch to *patrol* or *fire-intervention* with a low probability (0.05 for both cases obtained by normalization as specified by the "×").

If Khepera was in mode *patrol* (second column), the most probable behavior is that it stays in this mode, although it can switch to any other one. If Khepera was in mode *recognition* (third column) we set a very high probability for it to stay in this mode because we do not want it to be easily distracted from this task and we preclude any possibility of switching to *idle*. In mode *fire-intervention* (column 4) we exclude any switch to *idle*, *recognition* or *homing*. Finally, when in mode *homing* or *recharge*, the most probable behavior is to stay in the current mode, although nothing is definitely excluded.

Table 3 mainly says that *patrol* and *recognition* suppose that *Vigil* is true and that *homing* supposes that *Vigil* is false. When *idle* the probability that *Vigil* is true is not 0, because Khepera may be *idle* to recharge its batteries even when ordered to work.

Table 4 specifies that when *idle* it is more probable that *Energy* is low than high. It also says that *patrol* and *recognition* suppose a high *Energy* and *recharge* the opposite.

Table 5 says that *idle* imposes that *Base* is true, when *patrol*, *recognition*, *homing* and *recharge* suppose with a high probability that Khepera is not at its base.

Table 4. **P**(*Energy* | *Behavior* ⊗ π-*behavior*).

Energy/ Behavior	*idle*	*patrol*	*recognition*	*fire-interv.*	*homing*	*recharge*
very-low	0.325	0	0	×	×	0.8
low	0.325	0.1	0.1	×	×	0.2
high	0.25	×	×	×	×	0
very-high	0.1	×	×	×	×	0

Table 5. **P**(*Base* | *Behavior* ⊗ π-*behavior*).

Base/ Behavior	*idle*	*patrol*	*recognition*	*fire-interv.*	*homing*	*recharge*
false	0	0.99	0.99	×	0.99	0.99
true	1	0.01	0.01	×	0.01	0.01

Table 6. **P**(*Fire* | *Behavior* ⊗ π-*behavior*).

Fire/ Behavior	*idle*	*patrol*	*recognition*	*fire-interv.*	*homing*	*recharge*
false	1	1	1	0	1	1
true	0	0	0	1	0	0

Table 7. **P**(*Identify* | *Behavior* ⊗ π-*behavior*).

Identify/ Behavior	*idle*	*patrol*	*recognition*	*fire-interv.*	*homing*	*recharge*
false	×	×	0	×	×	×
true	×	×	1	×	×	×

Table 6 means that when Khepera is facing a fire, it is necessarily in mode *fire-intervention*.

Finally, Table 7 says *recognition* imposes that Khepera has been ordered to do so (*Identify* is true).

9.3. Identification

No identification is required, as there are no free parameters in π-*behavior*.

9.4. Utilization

The robot chooses its behavior with the following query:

$$\textbf{Draw}(\textbf{P}(\textit{Behavior} \mid \textit{Vigil} \otimes \textit{Energy} \otimes \textit{Base} \otimes \textit{Fire} \otimes \textit{Identify} \otimes \textit{Behavior\_t-1} \otimes \pi\textit{-behavior})) \quad \text{[E9.1]}$$

that can be easily computed:

$$\textbf{P}(\textit{Behavior} \mid \textit{Vigil} \otimes \textit{Energy} \otimes \textit{Base} \otimes \textit{Fire} \otimes \textit{Identify} \otimes \textit{Behavior\_t-1} \otimes \pi\textit{-behavior}) = \frac{1}{\Sigma} \times \textbf{P}(\textit{Behavior\_t-1} \mid \pi\textit{-behavior})$$

$$\times \mathbf{P}(Behavior \mid Behavior\_t\text{-}1 \otimes \pi\text{-}behavior)$$
$$\times \mathbf{P}(Vigil \mid Behavior \otimes \pi\text{-}behavior)$$
$$\times \mathbf{P}(Energy \mid Behavior \otimes \pi\text{-}behavior)$$
$$\times \mathbf{P}(Base \mid Behavior \otimes \pi\text{-}behavior)$$
$$\times \mathbf{P}(Fire \mid Behavior \otimes \pi\text{-}behavior)$$
$$\times \mathbf{P}(Identify \mid Behavior \otimes \pi\text{-}behavior) \quad [E9.2]$$

9.5. Results, Lessons and Comments

9.5.1. Results. Using these techniques, Khepera obtains temporal sequences of behaviors that appear convincing to a human observer (an instance of such a sequence will be given in the next section, see Movie 5[17]).

For instance, these sequences are stable. Khepera does not behave like a weathercock that changes its mind every second.

Lesson 9: Inverse programming

This experiment demonstrates a completely new method of specifying temporal sequences of tasks that could be called "inverse temporal programming." Indeed, the programmer does not specify, as usual, the necessary conditions for an action. On the contrary, he or she specifies for each action the expected observations and assumes that knowing the action these observations are independent.

Inverse programming presents two main advantages.

- It is robust to unforeseen situations. A sequence of actions is always produced, even in cases that the programmer did not explicitly take into account.
- Due to the conditional independence assumption, the number of cases to take into account grows only linearly with the number of conditioning variables.

The *a priori* specification of the probability distributions of the observed variables knowing the behavior may be a difficulty. However, it is possible to learn these distributions (see Diard and Lebeltel, 1999).

Furthermore, the stability of the behavior according to the values in the tables is a critical question. We do not have yet any quantified results to answer this question but we are trying to set up an experimental protocol to evaluate it.

10. Integration: A Night Watchman Task

10.1. Goal and Experimental Protocol

The practical goal and experimental protocol of the night watchman task has already been presented in Section 9.1.

The scientific purpose of this last experiment is to prove that Bayesian robot programming is an efficient constructive methodology and that all the previous descriptions may be integrated into a single synthetic one.

Three descriptions and a few corresponding variables necessary for the night watchman task have not yet been presented to keep the paper short:

1. $\mathbf{P}(Base \otimes Px1 \otimes \cdots \otimes Px8 \otimes L1 \otimes \cdots \otimes L8 \mid \pi\text{-}base)$ used by Khepera to decide if it is at its base.
2. $\mathbf{P}(Move \otimes Behavior \otimes Move\_t\text{-}1 \otimes Tempo \otimes Tour \mid \pi\text{-}move)$ another temporal sequencing description required because some of the behaviors are successions of reactive movements.
3. $\mathbf{P}(Vrot \otimes Vtrans \otimes Move \otimes H \otimes Dir \otimes Prox \otimes DirL \otimes ProxL \otimes Vtrans\_c \otimes Theta2 \mid \pi\text{-}speed)$ built on the reactive behaviors to finally decide the rotation and translation speeds.

10.2. Specification

10.2.1. Variables. The nightwatchman task requires 41 variables.

- Thirty-three "sensory" variables that Khepera may read every tenth of a second. When convenient, we will summarize these 33 variables by their conjunction (a variable named *Sensory-variables*).

$$\begin{aligned} &Sensory\text{-}variables \\ &\equiv Px1 \otimes \cdots \otimes Px8 \otimes L1 \otimes \cdots \otimes L8 \\ &\quad \otimes Vigil \otimes Energy \otimes Fire \\ &\quad \otimes Identify \otimes Behavior\_t\text{-}1 \\ &\quad \otimes Move\_t\text{-}1 \otimes Tempo \otimes Tour \otimes Dir \\ &\quad \otimes Prox \otimes DirL \otimes ProxL \otimes Vtrans\_c \\ &\quad \otimes Nlt \otimes Nrt \otimes Per \otimes Lrl \end{aligned} \quad [E10.1]$$

- Five internal variables: *Base*, *Theta2*, *Behavior*, *Move*, *H*.
- Three "motor" variables that Khepera must compute. These three variables are the rotation speed *Vrot*, the translation speed *Vtrans* and the identity of the object O.

10.2.2. Decomposition and Parametric Forms. The decomposition of the joint distribution on these 41 variables is a product of a uniform distribution on the sensory variables (**P**(*Sensory-variables* | π*-watchman*)) and eight questions addressed to the previously defined descriptions:

"What order should be sent to the motors, knowing the sensory state, and ignoring the values of the internal variables?"

The answer to that question is obtained, as usual, by summing over the five ignored variables. This leads to the following result:

$$\begin{aligned}
&\mathbf{P}(Vrot \otimes Vtrans \otimes O \mid Sensory\text{-}variables \otimes \pi\text{-}watchman)\\
&= \frac{1}{\Sigma} \times \mathbf{P}(O \mid Nlt \otimes Nrt \otimes Per \otimes Lrl \otimes \pi\text{-}object)\\
&\quad \times \sum_{Move\, Theta2\, H} \left(\begin{array}{l}
\left(\sum_{Behavior} \left(\begin{array}{l} \mathbf{P}(Move \mid Behavior \otimes \cdots \otimes Tour \otimes \pi\text{-}move) \\ \times \sum_{Base} \left(\begin{array}{l} \mathbf{P}(Behavior \mid Vigil \otimes \cdots \otimes Behavior_t\text{-}1 \otimes \pi\text{-}behavior) \\ \times \mathbf{P}(Base \mid Px1 \otimes \cdots \otimes L8 \otimes \pi\text{-}base) \end{array} \right) \end{array} \right) \right) \\
\times \mathbf{P}(Theta2 \mid L1 \otimes \cdots \otimes L8 \otimes \pi\text{-}fusion) \\
\times \mathbf{P}(H \mid Prox \otimes \pi\text{-}home) \\
\times \mathbf{P}(Vrot \otimes Vtrans \mid Move \otimes \cdots \otimes Theta2 \otimes \pi\text{-}speed)
\end{array} \right)
\end{aligned} \quad \text{[E10.4]}$$

$$\begin{aligned}
&\mathbf{P}\left(\begin{array}{c} Sensory\text{-}variables \\ Base \otimes Theta2 \otimes Behavior \otimes Move \otimes H \mid \pi\text{-}watchman \\ Vrot \otimes Vtrans \otimes O \end{array} \right)\\
&= \mathbf{P}(Sensory\text{-}variables \mid \pi\text{-}watchman)\\
&\quad \times \mathbf{P}(Base \mid Px1 \otimes \cdots \otimes Px8 \otimes L1 \otimes \cdots \otimes L8 \otimes \pi\text{-}base)\\
&\quad \times \mathbf{P}(Theta2 \mid L1 \otimes L2 \otimes L3 \otimes L4 \otimes L5 \otimes L6\\
&\qquad \otimes L7 \otimes L8 \otimes \pi\text{-}fusion)\\
&\quad \times \mathbf{P}(Behavior \mid Vigil \otimes Energy \otimes Base \otimes Fire\\
&\qquad \otimes Identify \otimes Behavior_t\text{-}1 \otimes \pi\text{-}behavior)\\
&\quad \times \mathbf{P}(Move \mid Behavior \otimes Move_t\text{-}1 \otimes Tempo\\
&\qquad \otimes Tour \otimes \pi\text{-}move)\\
&\quad \times \mathbf{P}(H \mid Prox \otimes \pi\text{-}home)\\
&\quad \times \mathbf{P}(Vrot \otimes Vtrans \mid Move \otimes H \otimes Dir \otimes Prox \otimes DirL\\
&\qquad \otimes ProxL \otimes Vtrans_c \otimes Theta2 \otimes \pi\text{-}speed)\\
&\quad \times \mathbf{P}(O \mid Nlt \otimes Nrt \otimes Per \otimes Lrl \otimes \pi\text{-}object) \qquad \text{[E10.2]}
\end{aligned}$$

10.3. Identification

No identification is required.

10.4. Utilization

The ultimate question that Khepera must answer is:

$$\begin{aligned}
&\mathbf{P}(Vrot \otimes Vtrans\\
&\quad \otimes O \mid Sensory\text{-}variables \otimes \pi\text{-}watchman) \qquad \text{[E10.3]}
\end{aligned}$$

This expression may seem complex. In fact, it exactly reflects the structure of the reasoning required to solve the problem.

- Recognizing the object is independent of Khepera control.
- The innermost sum searches the *Behavior* ignoring *Base*

$$\begin{aligned}
&\sum_{Base} \left(\begin{array}{l} \mathbf{P}(Behavior \mid Vigil \otimes \cdots \otimes Behavior_t\text{-}1 \\ \quad \otimes \pi\text{-}behavior) \times \mathbf{P}(Base \mid Px1 \\ \quad \otimes \cdots \otimes L8 \otimes \pi\text{-}base) \end{array} \right)\\
&\quad = \mathbf{P}(Behavior \mid Vigil \otimes \cdots \otimes Behavior_t\text{-}1\\
&\qquad \otimes Px1 \otimes \cdots \otimes L8 \otimes \pi\text{-}watchman) \qquad \text{[E10.5]}
\end{aligned}$$

- The intermediary sum searches the movement ignoring the *Behavior* and *Base*.
- The position of the light source (*Theta2*) is estimated by the fusion of the light sensors information.
- The command variable *H* is estimated according to the value of *Prox*.
- The outermost sum searches for *Vrot* and *Vtrans* ignoring the precise values of the five internal variables.

No decision is made except the ultimate one about *Vrot* and *Vtrans*. Uncertainty is propagated from the innermost level to the outermost. All the available information is taken into account. The resulting observed

robot behavior is, indeed, a probabilistic mixture of the different component descriptions.

Discarding no information has an obvious computational cost. The evaluation of the three levels of cascading sums may be very time consuming. Thus, the programmer may choose to make decisions on any intermediary variables. This choice will always trade a gain of efficiency for a loss of information. For instance, the most efficient possible program would make a decision for all the internal variables:

1. **Draw**(**P**(*Base* | $Px1 \otimes \cdots \otimes L8 \otimes \pi$-*base*)) `to decide if the robot is at its base`,
2. **Draw**(**P**(*Behavior* | $\cdots \otimes$ *Base* $\otimes \cdots \otimes \pi$-*behavior*)) `to decide the` *Behavior* `knowing` *Base*,
3. **Draw**(**P**(*Move* | *Behavior* $\otimes \cdots \otimes \pi$-*move*)) `to choose a movement knowing the` *Behavior*,
4. **Draw**(**P**(*Theta*2 | $L1 \otimes \cdots \otimes L8 \otimes \pi$-*fusion*)) `to decide the position of the light source`,
5. **Draw**(**P**(*H* | *Prox* $\otimes \pi$-*home*)) `to decide between avoidance and phototaxy`,
6. and finally, **Draw**(**P**(*Vrot* $\otimes$ *Vtrans* | $\ldots \otimes \pi$-*speed*)) `to control the robot.`

10.5. Results, Lessons and Comments

The results obtained are satisfactory to a human observer. Khepera performed this task hundreds of time in various environments and conditions. The behavior was very robust; for instance, this experiment ran without interruption, 10 hours a day for three days as a demonstration during a public science fair.

The Movie 5[17] shows Khepera during one of these experiments. It successively shows:

- Khepera identifying an object,
- Khepera aborting its object recognition due to a possible fire detection,
- Khepera verifying that it is really facing a fire by trying to blow it,
- Khepera extinguishing the fire,
- Khepera patrolling the environment (it stops occasionally to detect movement and sounds an alarm if it succeeds),
- Khepera returning to its base.

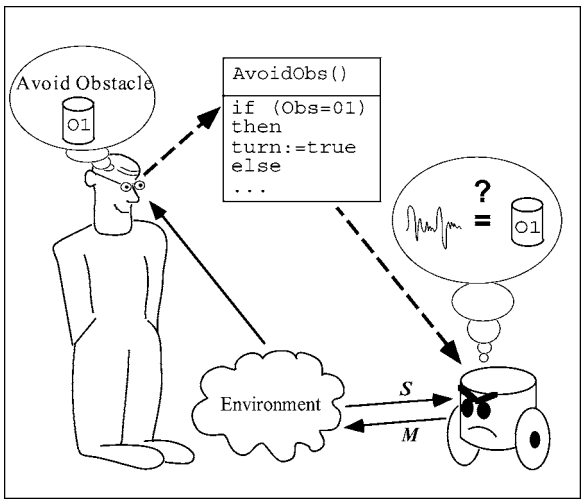


Figure 13. The symbolic approach in robotics.

11. Synthesis

11.1. Principles, Theoretical Foundation and Methodology

11.1.1. Principles. The dominant paradigm in robotics may be caricatured by Fig. 13.

The programmer of the robot has an abstract conception of its environment. He or she may describe the environment in geometrical terms because the shape of objects and the map of the world can be specified. He or she may describe the environment in analytical terms because laws of physics that govern this world are known. The environment may also be described in symbolic terms because both the objects and their characteristics can be named.

The programmer uses this abstract representation to program the robot. The programs use these geometric, analytic and symbolic notions. In a way, the programmer imposes on the robot his or her own conception of the environment.

The difficulties of this approach appear when the robot needs to link these abstract concepts with the raw signals it obtains from its sensors and sends to its actuators.

The central origin of these difficulties is the irreducible incompleteness of the models. Indeed, there are always some hidden variables, not taken into account in the model, that influence the phenomenon. The effect of these hidden variables is that the model and the phenomenon never behave exactly the same. The hidden variables prevent the robot from relating the abstract

concepts and the raw sensory-motor data reliably. The sensorimotor data are then said to be "noisy" or even "aberrant". A queer reversal of causality occurs that seem to consider that the mathematical model is exact and that the physical world has some unknown flaws.

Compelling the environment is the usual answer to these difficulties. The programmer of the robot looks for the causes of "noises" and modifies either the robot or the environment to suppress these "flaws". The environment is modified until it corresponds to its mathematical model. This approach is both legitimate and efficient from an engineering point of view. A precise control of both the environment and the tasks ensures that industrial robots work properly.

However, compelling the environment may not be possible when the robot must act in an environment not specifically designed for it. In that case, completely different solutions must be devised.

The purpose of this paper is to propose Bayesian Robot Programming (BRP) as a possible solution.

Figure 14 presents the principles of this approach.

The fundamental notion is to place side by side the programmer's conception of the task (the preliminary knowledge) and the experimental data to obtain the programming resources called "descriptions". As seen in the different examples described in this paper, both the preliminary knowledge and the descriptions may be expressed easily and efficiently in probabilistic terms.

The preliminary knowledge gives some hints to the robot about what it may expect to observe. The preliminary knowledge is not a fixed and rigid model purporting completeness. Rather, it is a gauge, with open

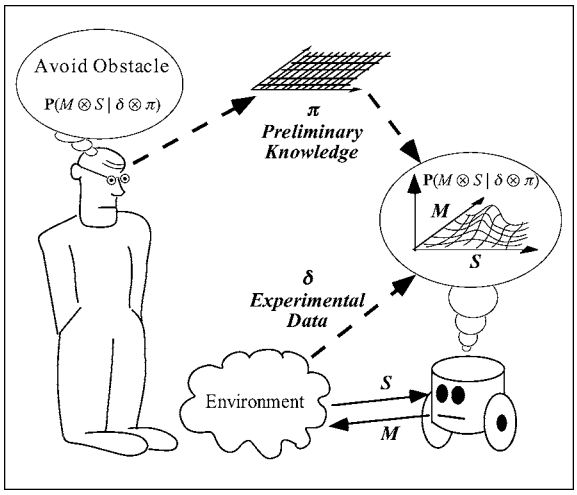


Figure 14. The BRP approach in robotics.

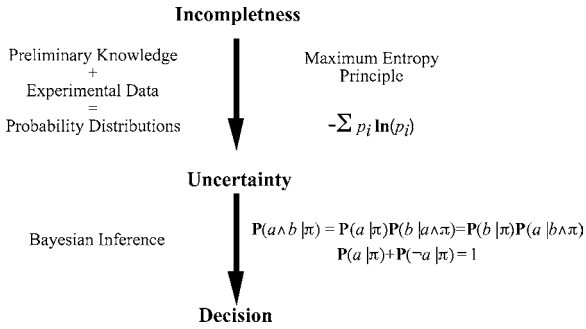


Figure 15. Theoretical foundation.

parameters, waiting to be molded by the experimental data. Learning is the means of setting these parameters. The resulting descriptions result from both the views of the programmer and the physical specificities of each robot and environment. Even the influence of the hidden variables is taken into account and quantified; the more important their effects, the more noisy the data, the more uncertain the resulting descriptions.

However, Bayesian Robot Programming preserves two very important merits of the symbolic approach. Thanks to the preliminary knowledge, the descriptions are comprehensible to the programmer. Thanks to Bayesian inference, complex reasoning is possible.

11.1.2. Theoretical Foundations. The theoretical foundations of Bayesian robot programming may be summed up by Fig. 15.

The first step transforms the irreducible incompleteness to uncertainty. Starting from the preliminary knowledge and the experimental data, learning builds probability distributions.

The maximum entropy principle is the theoretical foundation of this first step. Given some preliminary knowledge and some data, the probability distribution that maximizes the entropy is *the* distribution that *best* represents this couple. Entropy gives a precise, mathematical and quantifiable meaning to the "quality" of a distribution (for justifications of the maximum entropy principle see, for instance (Jaynes, 1982; Robert, 1990; Bessière et al., 1998b)).

Two extreme examples may help to understand what occurs:

- Suppose that we are studying a formal phenomenon. There are no hidden variables. A complete model may be proposed. If we select this model as the preliminary knowledge, any data set will lead to a description made of Dirac probability distributions.

There is no uncertainty, any question may be answered either by true or false. Logic appears as a special case of the Bayesian approach in that particular context (see Cox, 1979).

- On the opposite extreme, suppose that a preliminary knowledge consists of very poor hypotheses about the modeled phenomenon. Learning will lead to "flat" distributions, containing no information. No relevant decisions can be made, only completely random ones.

Hopefully, most common cases are somewhere in between these two extremes. Preliminary knowledge, even imperfect and incomplete, is relevant and provides interesting hints about the observed phenomenon. The resulting descriptions are neither Diracs nor uniform distributions. They give no certitudes, although they provide a means of taking the best possible decision according to the available information.

The second step consists of reasoning with the probability distributions obtained by the first step.

To do so, we only require the two basic rules of Bayesian inference (see Section 3). These two rules are to Bayesian inference what the resolution principle is to logical reasoning (see Robinson, 1965; Robinson, 1979; Robinson and Sibert, 1983a; Robinson and Sibert, 1983b). These inferences may be as complex and subtle as those usually achieved with logical inference tools, as illustrated in the different examples in this paper.

11.2. Advantages

In this section we survey, comment and briefly discuss the advantages of the BRP method proposed in this paper.

- *Ability to treat incomplete and uncertain information*: The basis of this work is related to the fundamental difficulty of robot programming in real environment. For us this difficulty is the direct consequence of the irreducible incompleteness of models. Consequently, the first advantage of BRP is its ability to take into account this incompleteness and the resulting uncertainty. This is obtained in three steps, thanks to the following three abilities of the method:
  - *Ability to convert incompleteness to uncertainty by learning*, as demonstrated in the numerous instances where the free parameters of the preliminary knowledge are identified from experimental data (see, for instance, Section 5 concerning reactive behaviors or Section 8 concerning object recognition). Object recognition, for instance, shows that with simple preliminary knowledge, we are able to learn descriptions sufficient for classification. However, in this task there are numerous ignored variables such as, for instance, the color and material of the objects, the global lighting of the room, the approximate quality of the contour following behavior or the position from where the robot has started.
  - *Ability to reason despite uncertainty*, as demonstrated by all the experiments requiring inference (see, for instance, Section 6 about sensor fusion or Section 8 about object recognition). The "nightwatchman" task (see Section 10) shows the complexity of the possible reasoning (41 variables, 12 descriptions, four hierarchical levels).
  - *Ability to decide, taking uncertainty into account*: The decision strategy selected in this work has been to draw the values of the searched variables from the distributions obtained by the preceding inference step. This strategy "renders" uncertainty, the decision are nearly deterministic when the distributions are sharp, and conversely, nearly random when they are flat.

- *Simple and sound theoretical bases*: The proposed approach is founded on simple theoretical bases. Essential questions may be asked clearly and formally and eventually answered by mathematical reasoning. For instance, one may consider to fairly compare Bayesian inference and logic as two possible models of reasoning. Thanks to these theoretical bases, the experimental results (successes or even more enlightening failures) may be analyzed and understood in detail.
- *Generic, systematic and simple programming method*: BRP is simple, systematic and generic. Simple, as this method may be learned and mastered easily. Systematic, as it may be applied with rigor and efficiency. Generic, as this method may be also used in numerous other domains than robot programming.
- *Homogeneity of representations and resolution processes*: BRP is based on a unique data structure, called a description, associated with two inference rules. This homogeneity leads to simple and generic program development.

- *Obligation to state all hypothesis*: Choosing a description as the only data structure to specify robotics programs and following a systematic method to do so compel the programmer to exhaustively express his knowledge about the task. Everything that should be known about a given robotic problem is in its description: the synthesis between the preliminary knowledge and the experimental data. There is no hidden knowledge in either the inference program or the decision algorithm. As the description encapsulates all the relevant information, exchanging, sharing or discussing models is easy and rigorous.
- *Large capacity of expression*: Descriptions offer a large capacity of expression to specify models and to question them as well.
  - *Specification capacity*: The different experiments described in this paper prove that descriptions may be used to specify numerous different models. Let us recall that we used descriptions to learn simple reactive behaviors (Section 5), to combine them (Section 6), to hierarchically compose them (Section 8), to merge sensor information (Section 6), to recognize situations (Section 8), to carry out temporal sequencing (Section 9) and finally, to specify a task integrating all the previously defined descriptions (Section 10).
  - *Question capacity*: Let us also recall that any question may be asked to a joint distribution. Mathematically, all variables appearing in a joint distribution play the exact same role. They may all, indifferently, be known, unknown or searched. The description is neither a direct nor an inverse model. Sensor fusion (Section 6), situation recognition (Section 8) or inverse programming (Section 9) offer instances where the questions asked do not correspond to the specification order. Furthermore, there is no ill-posed problem. If a question may have several solutions, the probabilistic answer will simply have several peaks. Some instances of sensor fusion exemplified this point (see Section 6.3).
- *Ability for real incremental development of robots*: Bayesian Robot Programming, thanks to its clear theoretical foundations and to its rigorous programming methodology, appears to be an incremental method of developing robot programs that can really be used in practice. The final experiment (Section 10) demonstrates that point.
  - *Ability to combine descriptions*: The first incremental development tool is description combination (Section 6). With this tool it is possible to define new behaviors as weighted mixtures of different simpler ones.
  - *Ability to compose descriptions*: The second incremental development tool is hierarchical description composition (Section 8). It is in some senses similar to calling sub-procedures in classical programming, as some of the parametric forms appearing in a decomposition may be questions addressed to more basic descriptions.
  - *Description = Resource*: More generally, a description, as an internal representation of a physical phenomenon, may be considered as a programming resource. For instance, a description may offer new variables to be used in other descriptions. This is the case with the variable O that identifies the object, provided by the object recognition description (Section 8). The object recognition also proposes another example of the use of a description as a programming resource. Indeed, the contour following behavior is a necessary tool for computing the four variables *Nlt*, *Nrt*, *Per* and *Lrl* used by the object recognition description. Numerous other possibilities for enhancing the capacity of a robot using descriptions as resources may be found in Dedieu's Ph.D. thesis (Dedieu, 1995).

11.3. BRP in Practice

BRP is a very simple, practical and efficient way to program robots for the 3 main following reasons:

- BRP proposes a generic methodology for robot programming:
  1. *Specification*: define the preliminary knowledge.
     - 1.1 Choose the pertinent variables.
     - 1.2 Decompose the joint distribution.
     - 1.3 Define the parametric forms.
  2. *Identification*: identify the free parameters of the preliminary knowledge.
  3. *Utilization*: ask a question of the joint distribution.

- The conditional independencies are used to break the complexity and keep the computation tractable.
- BRP may really be considered as a programming language because it proposes three main tools to build complex programs from simpler ones:
  - Calling Bayesian subroutines.
  - Probabilistic if-then-else.
  - Inverse programming to sequence behaviors.

11.4. Scalability

This paper presented only very simple instances of robot programming for didactic purposes. The goal was to illustrate the fundamental principles rather than to present applications.

Consequently two fundamental questions are still open: is BRP scalable? May it be used for practical robotics applications?

Elements of answer to these two questions may be found in other works that used the BRP approach:

- Kamel Mekhnacha built a Bayesian robotic CAD system which offers the usual capabilities of such a system but also handles the uncertainties. This system works with hundreds of variables and tens of probability distributions (Mekhnacha, 1999; Mekhnacha, Mazer and Bessière, 2000; Mekhnacha, Mazer and Bessière, 2001)
- Julien Diard proposed in his Ph.D. thesis the concept of Bayesian Maps, which is a specialization of the BRP formalism, for representing the environment in navigation tasks. In particular, the descriptions and the combination operators considered in this work are special cases of the one presented here (Diard, 2003).
- Christophe Coué is using BRP for an Advanced Driving Assistance System (ADAS) (Coué et al., 2002; Coué et al., 2003).

Ongoing works include manipulation with 6 DOF arms, pick and place problems, videogame Bot programming, outdoor SLAM with automotive vehicles and indoor service robotics.

Notes

1. Partially Observable Markov Decision Process.
2. Hidden Markov Models.
3. By contrast, the disjunction of two variables, defined as the set of propositions $x_i \vee y_j$, is not a variable. These propositions are not mutually exclusive.
4. See some references on justifications of these two rules in Section 2.
5. Order of magnitude on a standard desktop computer for the inferences required by the experiments described in the sequel.
6. Application Programming Interface.
7. Ecole Polytechnique Fédérale de Lausanne (Switzerland).
8. http://www.K-team.com/.
9. http://www-laplace.imag.fr/ENGLISH/PRESENTATION/Semaine-Science/Trans7/T7.mov.
10. http://www-laplace.imag.fr/ENGLISH/PRESENTATION/Semaine-Science/Trans8/T8.mov.
11. http://www-laplace.imag.fr/ENGLISH/PRESENTATION/Semaine-Science/Trans9/T9.mov.
12. $\mathbf{P}(Dir \otimes Prox \otimes Vrot \mid \delta\text{-}avoid \otimes \pi\text{-}obstacle)$ and $\mathbf{P}(Dir \otimes Prox \otimes Vrot \mid \delta\text{-}follow \otimes \pi\text{-}obstacle)$.
13. http://www-laplace.imag.fr/ENGLISH/PRESENTATION/Semaine-Science/Trans10/T10.mov.
14. see Diard and Lebeltel (1999).
15. Hierachical Mixture of Expert.
16. A Laplace succession law on a variable V is defined by: $\frac{1+n_v}{N+\lfloor V \rfloor}$ with N the total number of observations, $\lfloor V \rfloor$ the number of possible values for V and n_v the number of observations of the specific value v.
17. http://www-laplace.imag.fr/ENGLISH/PRESENTATION/Semaine-Science/Trans12/T12.small.mov.

References

Aji, S.M. and McEliece, R.J. 2000. The generalized distributive law. *IEEE Trans. Information Theory*, 46(2).

Alami, R., Chatila, R., Fleury, S., Ghallab, M., and Ingrand, F. 1998. An architecture for autonomy. *International Journal for Robotics Research* (*IJRR*), 17(4):315–337.

Arulampalam, S., Maskell, S., Gordon, N., and Clapp, T. 2001. A tutorial on particle filters for on-line nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing.* Available at citeseer.nj.nec.com/maskell01tutorial.html.

Aycard, O. 1998. Architecture de contrôle pour robot mobile en environnement intérieur structuré. Ph.D. thesis, Université Henri Poincaré, Nancy, France.

Bernhardt, R. and Albright, S.L. (Ed.). 1993. *Robot Calibration*. Chapman & Hall.

Beetz, M. and Belker, T. (2001). Learning strucured reactive navigation plans from executing MDP navigation policies. In *Proceedings of Agents 2001*.

Bessière, P., Dedieu, E., Lebeltel, O., Mazer, E., and Mekhnacha, K. 1998a. Interprétation ou Description (I): Proposition pour une théorie probabiliste des systèmes cognitifs sensori-moteurs. *Intellectica*, 26/27:257–311, Paris, France.

Bessière, P., Dedieu, E., Lebeltel, O., Mazer, E., and Mekhnacha, K. 1998b. Interprétation ou Description (II): Fondements mathématiques de l'approche F+D. *Intellectica*, 26/27:313–336, Paris, France.

Bessière, P. 2002. Procédé de détermination de la valeur à donner à différents paramètres d'un système: Demande de brevet d'invention n°0235541154.4, Institut Européen des Brevets.

Bessière, P. and the BIBA-INRIA Research Group. 2003. Survey: Probabilistic methodology and techniques for artefact conception and development. INRIA Technical Report RR-4730. Available at http://www.inria.fr/rrrt/rr-4730.html.

Borrelly, J.-J., Coste, E., Espiau, B., Kapellos, K., Pissard-Gibollet, R., Simon, D., and Turro, N. 1998. The ORCCAD architecture. *International Journal for Robotics Research* (*IJRR*), 17(4):338–359.

Boutilier, C., Reiter, R., Soutchanski, M., and Thrun, S. 2000. Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of AAAI 2000*.

Brafman, R.I., Latombe, J.-C., Moses, Y., and Shoham, Y. 1997. Applications of a logic of knowledge to motion planning under uncertainty. *Journal of the ACM*, 44(5):633–668.

Bretthorst, G.L. 1988. *Bayesian Spectrum Analysis and Parameter Estimation*. Springer Verlag.

Brooks, R.A. 1986. A robust layered control systems for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1): 14–23.

Cooper, G. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405.

Coué, C. and Bessière, P. 2001. Chasing an elusive target with a mobile robot. In *IEEE/IROS 2001*, Maui, Hawaii, USA.

Coué, C., Fraichard, Th., Bessière, P., and Mazer, E. 2002. Multisensor data fusion using bayesian programming: An automotive application. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Coué, C., Fraichard, Th., Bessière, P., and Mazer, E. 2003. Using Bayesian programming for multi-sensor multi-target tracking in automotive applications. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)* (in Press).

Cox, R.T. 1961. *The Algebra of Probable Inference.* The John Hopkins Press: Baltimore, USA.

Cox, R.T. 1979. Of inference and inquiry, an essay in inductive logic. In Raphael D. Levine and Myron Tribus (Eds.), *The Maximum Entropy Formalism*, M.I.T. Press: USA.

Dagum, P. and Luby, M. 1993. Approximate probabilistic reasoning in Bayesian belief network is NP-Hard. *Artificial Intelligence*, 60:141–153.

Darwiche, A. and Provan, G. 1997. Query DAGs: A practical paradigm for implementing belief-network inference. *Journal of Artificial Intelligence Research* (*JAIR*), 6:147–176.

Dedieu, E. 1995. La représentation contingente: Vers une reconciliation des approches fonctionnelles et structurelles de la robotique autonome. Thèse de troisième cycle INPG (Institut National Polytechnique de Grenoble) Grenoble, France.

Dekhil, M. and Henderson, T.C. 1998. Instrumented sensor system architecture. *International Journal for Robotics Research* (*IJRR*), 17(4):402–417.

Delcher, A.L., Grove, A.J., Kasif, S., and Pearl, J. 1996. Logarithmic-time updates and queries in probabilistic networks. *Journal of Artificial Intelligence Research* (*JAIR*), 4:37–59.

Diard, J. and Lebeltel, O. 1999. Bayesian learning experiments with a Khepera Robot, In Experiments with the Mini-Robot Khepera: Löffler Mondada Rückert (Ed.), *Proceedings of the 1st International Khepera Workshop*, December 1999, Paderborn, HNI-Verlagsschriftenreihe, Band 64, Germany, pp. 129–138.

Diard, J. and Lebeltel, O. 2000. Bayesian programming and hierarchical learning in robotics. In Meyer, Berthoz, Floreano, Roitblat and Wilson (Eds.), *SAB2000 Proceedings Supplement Book*, Publication of the International Society for Adaptive Behavior: Honolulu.

Diard, J. 2003. La carte bayésienne: Un modèle probabiliste hiérarchique pour la navigation en robotique mobile. PhD thesis, Institut National Polytechnique de Grenoble (INPG), 27 janvier 2003.

Donald, B.R. 1988. A geometric approach to error detection and recovery for robot motion planning with uncertainty. *Artificial Intelligence*, 37:223–271.

Erickson, G.J. and Smith, C.R. 1988a. *Maximum-Entropy and Bayesian Methods in Science and Engineering*, Vol. 1: *Foundations*. Kluwer Academic Publishers.

Erickson, G.J. and Smith, C.R. 1988b. *Maximum-Entropy and Bayesian Methods in Science and Engineering*, Vol. 2: *Applications*. Kluwer Academic Publishers.

Frey, B.J. 1998. *Graphical Models for Machine Learning and Digital Communication.* MIT Press.

Fox, D., Burgard, W., Kruppa, H., and Thrun, S. 2000. A Probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8:325–344.

Fox, D., Thrun, S., Dellaert, F., and Burgard, W. 2001. Particle filters for mobile robot localization. In Doucet, A., de Freitas, N., and Gordon, N. (Eds.), *Sequential Monte Carlo Methods in Practice*, Springer-Verlag: New York, USA.

Gutmann, J.-S., Burgard, W., Fox, D., and Konolige, K. 1998. Experimental comparison of localization methods. In *Interenational Conference on Intelligent Robots and Systems.*

Halpern, J.Y. 1999a. A counterexample to theorems of Cox and Fine. *Journal of Artificial Intelligence Research* (*JAIR*), 10:67–85.

Halpern, J.Y. 1999b. Cox's Theorem revisited. *Journal of Artificial Intelligence Research* (*JAIR*), 11:429–435.

Jaakkola, T.S. and Jordan, M.I. 1999. Variational probabilistic inference and the QMR-DT network. *Journal of Artificial Intelligence Research* (*JAIR*), 10:291–322.

Jaynes, E.T. 1979. Where do we stand on maximum entropy? In Raphael D. Levine and Myron Tribus (Eds.), *The Maximum Entropy Formalism,* M.I.T. Press: USA.

Jaynes, E.T. 1982. On the rationale of maximum-entropy methods. In *Proceedings of the IEEE*.

Jaynes, E.T. 2003. *Probability Theory—The logic of Science*. Cambridge University Press (in press).

Jensen, F., Lauritzen, S., and Olesen, K. 1990. Bayesian updating in recursive graphical models by local computations. *Computational Statistical Quaterly*, 4:269–282.

Jordan, M.I. and Jacobs, R.A. 1994. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214.

Jordan, M. 1998. *Learning in Graphical Models.* MIT Press.

Jordan, M., Ghahramani, Z., Jaakkola, T.S., and Saul, L.K. 1999. An introduction to variational methods for graphical models. *Machine Learning* (in press).

Kaelbling, L.P., Littman, M.L., and Cassandra, A.R. 1996. Partially observable Markov decision processes for artificial intelligence, reasoning with uncertainty in robotics. In *International Workshop, RUR'95, Proceedings*, Springer-Verlag, pp. 146–162.

Kaelbling, L.P., Cassandra, A.R., and Kurien, J.A. 1996. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems.*

Kaelbling, L.P., Littman, M.L., and Cassandra, A.R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101.

Kapur, J.N. and Kesavan, H.K. 1992. *Entropy Optimization Principles with Applications*. Academic Press.

Koening, S. and Simmons, R. 1998. A robot navigation architecture based on partially observable Markov decision process models. In D. Kortenkamp, R.P. Bonasso, and R. Murphy (Eds.), *Mobile Robots and Artificial Intelligence*, AAAI Press.

Koller, D. and Pfeffer, A. 1997. Object-oriented Bayesian networks. In *Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI)*, Providence, Rhode Island, USA.

Konolidge, K. 1997. Improved occupancy grids for map building. *Autonomous Robots*, 4:351–367.

Konolidge, K. and Chou, K. 1999. Markov localization using correlation. In *International Joint Conference on Artificial Intelligence*, Stockolm, Sweden.

Lane, T. and Kaelbling, L.P. 2001. Toward hierachical decomposition for planning in uncertain environments. *Workshop on planning under Uncertainty and Incomplete Information at the 2001 International Joint Conference on Artificial Intelligence* (IJCAI-2001).

Laplace, Pierre Simon de. 1774. *Mémoire sur les probabilités des causes par les évènements*; Mémoire de l'académie royale des sciences. Reprinted in *Oeuvres complètes de Laplace*, vol. 8, Gauthier Villars, Paris, France.

Laplace, Pierre Simon de. 1814. *Essai philosophique sur les probabilités*; Courcier Imprimeur, Paris. Reprinted in *Oeuvres complètes de Laplace*, vol. 7, Gauthier Villars, Paris, France.

Lauritzen, S. and Spiegelhalter, D. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 50:157–224.

Lauritzen, S. L. 1996. *Graphical Models*. Oxford University Press.

Lebeltel, O. 1999. Programmation Bayésienne des Robots. Ph.D. Thesis, Institut National Polytechnique de Grenoble (INPG); Grenoble, France.

Lebeltel, O., Diard, J., Bessière, P., and Mazer, E. 2000. A bayesian framework for robotic programming. In *Twentieth International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering (MaxEnt 2000)*, Paris, France.

Lozano-Perez, T., Mason, M.T., and Taylor, R.H. 1984. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24.

MacKay, D.G. 1996. Introduction to Monte Carlo methods. In M. Jordan (Ed.), *Proc. of an Erice Summer School.*

Maes, P. 1989. How to do the right thing. *Connection Science Journal*, 1(3): 291–323.

Matalon, B. 1967. Epistémologie des probabilités. In Jean Piaget (Ed.), *Logique et connaissance scientifique*, Encyclopédie de la Pléiade, Editions Gallimard, Paris, France.

Mazer, E., Boismain, G., Bonnet des Tuves, J., Douillard, Y., Geoffroy, S., Dubourdieu, J., Tounsi, M., and Verdot, F. 1998. START: An industrial system for teleoperation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, vol. 2, Leuven (BE), pp. 1154–1159.

McLachlan, G.J. and Deep, D. 2000. *Finite Mixture Models*. Wiley: New York, USA.

Mekhnacha, K. 1999. Méthodes probabilistes bayésiennes pour la prise en compte des incertitudes géométriques: Application à la CAO-robotique. PhD. thesis INPG (Institut National Polytechnique de Grenoble), Grenoble, France.

Mekhnacha, K., Mazer, E., and Bessière, P. 2000. A Robotic CAD system using a Bayesian framework. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2000*, *Best Paper Award*), vol. 3, Takamatsu, Japan, pp. 1597–1604.

Mekhnacha, K., Mazer, E., and Bessière, P. 2001. The design and implementation of a Bayesian CAD modeler for robotic applications. *Advanced Robotics*, 15(1).

Mohammad-Djafari, A. and Demoment, G. 1992. *Maximum Entropy and Bayesian Methods*. Kluwer Academic Publishers.

Murphy, K. 1999. Bayesian map learning in dynamic environments. In *Proceedings of NIPS 99*.

Neal Radford, M. 1993. Probabilistic inference using Markov chain Monte-Carlo methods. Technical Report, CRG-TR-93-1, University of Toronto.

Parr, R. and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In *Proceedings of NIPS*, 1998.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers: San Mateo, California, USA.

Robert, C. 1990. An entropy concentration theorem: Applications, in artificial intelligence and descriptive statistics. *Journal of Applied Probabilities.*

Robinson, J.A. 1965. A machine oriented logic based on the resolution principle. *Jour. Assoc. Comput. Mach.*, 12.

Robinson, J.A. 1979. *Logic: Form and Function.* North-Holland, New York, USA.

Robinson, J.A. and Sibert, E.E. 1983a. LOGLISP: An alternative to PROLOG. *Machine Intelligence*, 10.

Robinson, J.A. and Sibert, E.E. 1983b. LOGLISP: Motivation, design and implementation. *Machine Intelligence*, 10.

Rosenblatt, J.K. 2000. Optimal selection of uncertain actions by maximizing expected utility. *Autonomous Robots*, 9:17–25.

Roumeliotis, S.I. and Bekey, G. 2000. Collective localization: A distributed Kalman filter approach to localization of groups of mobile robots. In *IEEE International Conference on Robotics and Automation*.

Roumeliotis, S.I. and Bekey, G.A. 2000. Bayesian estimation and Kalman filtering: A unified framework for Mobile Robot localization. In *Proc. IEEE Int. Conf. on Robotics and Automation*, San Fransisco, CA, pp. 2985–2992.

Ruiz, A., Lopez-de-Teruel, P.E., and Garrido, M.C. 1998. Probabilistic Inference from arbitrary uncertainty using mixtures of factorized generalized Gaussians. *Journal of Artificial Intelligence Research* (*JAIR*), 9:167–217.

Saul, L.K., Jaakkola, T., and Jordan, M.I. 1996. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research* (*JAIR*), 4:61–76.

Schneider, S.A., Chen, V.W., Pardo-Castellote, G., and Wang, H.H. 1998. ControlShell: A software architecture for complex electromechanical systems. *International Journal for Robotics Research* (*IJRR*), 17(4):360–380.

Shatkay, H. 1998. Learning models for robot navigation. PhD. dissertation and Technical Report cs-98-11, Brown University, Department of Computer Science, Providence, RI.

Smith, C.R. and Grandy, W.T. Jr. 1985. *Maximum-Entropy and Bayesian Methods in Inverse Problems.* D. Reidel Publishing Company.

Tarentola, A. 1987. *Inverse Problem Theory: Methods for Data Fitting and Model Parameters Estimation*. Elsevier: New York, USA.

Thrun, S. 1998. Bayesian landmark learning for mobile robot localization. *Machine Learning*, 33(1):41–76.

Thrun, S., Burgard, W., and Fox, D. 1998. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots*, 5:253–271.

Thrun, S. 2000. Towards programming tools for robots that integrate probabilistic computation and learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Zhang, N.L. and Poole, D. 1996. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research* (*JAIR*), 5:301–328.

Olivier Lebeltel received his Ph.D. in cognitive sciences from the Institut National Polytechnique de Grenoble, France, in 1999. Currently, he is a research associate at the Institut National de Recherche en Informatique et Automatique de Grenoble. Dr. Lebeltel works on modeling, inference and learning with Bayesian approaches applied to bio-inspired robotics and virtual reality.

Pierre Bessière is a senior researcher at CNRS (Centre National de la Recherche Scientifique) since 1992. He took his Ph.D. in Artificial Intelligence in 1983 from the Institut National Polytechnique of Grenoble, France. He did a post-doctorate at the Stanford Research Institute and then worked for several years in the computer science industry. He has been working for the last 15 years on evolutionary algorithms and Bayesian inference. He leads, with Emmanuel Mazer, the LAPLACE research group on stochastic models for perception, inference and action (http://www-laplace. imag.fr).

Julien Diard received his Ph.D. in computer science from the Institut National Polytechnique de Grenoble in 2003. He is currently a postdoctoral fellow with both the Laboratoire de Physiologie de la Perception et de 1'Action, of the Collège de France, Paris, and the Department of Mechanical Engineering of the National University of Singapore. Dr. Diard works on hierarchical Bayesian modelling and learning for mobile robot navigation, and in particular, testing the biological relevance of his model, the Bayesian map.

Emmanuel Mazer (born in 1953) is a senior researcher at CNRS since 1982. He took a Master in mathematics and his Ph.D. (1989) in computer science from Grenoble University. He did a post-doctorate at the department of Artificial Intelligence of the University of Edinburgh. He is a co-founder of two industrial company in Europe (ITMI and Aleph-technologies) and one in the USA. He worked as a research fellow at the MIT AI lab for three years and he is a co-author of a book on automatic robot programming (Handey MIT press). Currently he leads with Pierre Bessière the Laplace research group and works on Bayesian programming.

The design and implementation of a Bayesian CAD modeler for robotic applications

K. MEKHNACHA * E. MAZER † P. BESSIÈRE ‡

Abstract

We present a Bayesian CAD modeler for robotic applications. We address the problem of taking into account the propagation of geometric uncertainties when solving inverse geometric problems. The proposed method may be seen as a generalization of constraint-based approaches in which we explicitly model geometric uncertainties. Using our methodology, a geometric constraint is expressed as a probability distribution on the system parameters and the sensor measurements, instead of a simple equality or inequality. To solve geometric problems in this framework, we propose an original resolution method able to adapt to problem complexity. Using two examples, we show how to apply our approach by providing simulation results using our modeler.

Keywords: Robotics, CAD, Bayesian reasoning, Monte Carlo methods, geometric constraints.

1 Introduction

The use of geometric models in robotics and CAD systems necessarily requires a more or less realistic modeling of the environment. However, the validity of calculations with these models depends on their degree of fidelity to the real environment and the capacity of these systems to represent and take into account possible differences between the models and reality when solving a given problem.

This paper presents a new methodology based on Bayesian formalism to represent and handle geometric uncertainties in robotics and CAD systems. The approach presented in this paper may be seen as a generalization of constraint-based approaches. This generalization consists of explicitly taking into account the uncertainties in models. A constraint on a relative pose between two frames is represented by a probability distribution on the parameters of this pose instead of simple equality or inequality. In this framework, modeling information given by the programmer and the measurements obtained using sensors are represented and used in a homogeneous way. For a given problem, all the information we include on the geometric model and on the responses of the sensors used, is used optimally applying Bayesian reasoning.

Since the work of Laplace [Laplace, 1774], numerous results have been obtained using Bayesian inference techniques to take account of uncertainty. Bayesian formalism has been applied in diverse research fields. Numerous applications have been developed in physics [Jaynes, 1996, Neal, 1993], in artificial intelligence [Jaakkola and Jordan, 1999], as well as in mobile robotics [Thrun, 1998, Bessière et al., 1998] and computer vision [Weiss and Adelson, 1998], and especially in parameter identification problems [Presse and Gautier, 1992].

The principle of the proposed method is to infer, for a given problem, the marginal distribution of the unknown parameters using the probability calculus. The original geometric problem is reduced to an optimization problem over the marginal distribution to find a solution with maximum probability. In the general case, this marginal probability may contain an integral on a large dimension space.

The resolution method used to solve this integration/optimization problem is based on an adaptive genetic algorithm. The problem of integral numerical estimation is approached using a stochastic Monte Carlo method. The accuracy of this estimation is controlled by the optimization process to reduce computation time.

*Laboratoire LEIBNIZ, 46, avenue Félix Viallet, 38031 Grenoble, France
†Laboratoire GRAVIR, INRIA Rhône-Alpes, ZIRST 38030 Montbonnot, France
‡Laboratoire LEIBNIZ, 46, avenue Félix Viallet, 38031 Grenoble, France

A large category of robotic applications are instances of inverse geometric problems in the presence of uncertainties, for which our method is well suited. The simplicity of our specification method and the robustness of our resolution method make our approach applicable to numerous robotic applications [Mekhnacha, 1999, Mekhnacha et al., 2000], such as:

- kinematics inversion under geometric uncertainties using possibly redundant mechanisms,
- robot and sensor calibration,
- parts' pose and shape calibration using sensor measurements,
- robotic workcell design to obtain a configuration that can accomplish a given task with maximum accuracy.

Extensive experimentation on the approach was made possible thanks to the design and the implementation of a Bayesian CAD modeler. Experimental results obtained with this modeler have demonstrated the effectiveness and the robustness of our approach. Two examples of this experimentation are presented in this paper.

This paper is organized as follows. We first report related work in Sect. 2. In Sect. 3 we present our specification methodology and show how to formulate an optimization problem. In Sect. 4 we describe our numerical resolution method. Section 5 is an overview of the implementation of our modeler. We present two examples to illustrate our approach in Sect. 6 and Sect. 7 and give some conclusions and perspectives in Sect. 8.

This paper summarizes the work presented in the Ph.D. thesis of Kamel Mekhnacha [Mekhnacha, 1999].

2 Related work

The representation and handling of geometric uncertainties is a central issue in the fields of robotics and mechanical assembly. Since the precursor work of Taylor [Taylor, 1976], in which geometric uncertainties were taken into account in the robot manipulators planning process, numerous approaches have been proposed to model these uncertainties explicitly.

Methods modeling the environment using "certainty grids" [Moravec, 1988] and those using uncertain models of motion [Lozano-Pèrez, 1987, Alami and Simeon, 1994] have been used extensively, especially in mobile robotics.

Gaussian models to represent geometric uncertainties and to approximate their propagation have been proposed in manipulator programming [Puget, 1989] as well as in assembly [Sanderson, 1997]. Kalman filtering is a Bayesian recurrent implementation of these models. This technique has been used widely in robotics and vision [Zhang and Augeras, 1992] and particularly in data fusion [Bar-Shalom and Fortmann, 1988]. Gaussian model-based methods have the advantage of economy in the computation they require. However, they are only applicable when a linearization of the model is possible. Another limitation of these methods is their inability to take account of inequality constraints.

Geometric constraint-based approaches [Taylor, 1976, Owen, 1996] using constraint solvers have been used in robotic task-level programming systems. Most of these methods do not represent uncertainties explicitly. They handle uncertainties using a least-squares criterion when the solved constraints systems are over-determined. In the cases where uncertainties are explicitly taken into account (as is the case in Taylor's system), they are described solely as inequality constraints on possible variations.

3 Specification of probabilistic geometric constraints

In this section, we describe our methodology by giving some concepts and definitions necessary for probabilistic geometric constraint specification. We further show how to derive an objective function to maximize from the original geometric problem.

3.1 Probabilistic kinematic graph

A geometric problem is described as a "probabilistic kinematic graph," which we define as the directed graph having a set of n frames $S = \{S_1, \cdots, S_n\}$ as vertices and a set of m edges $A = \{A_{i_1 j_1}, \cdots, A_{i_m j_m}\}$, where $A_{i_k j_k}$ denotes an edge between the parent vertex S_{i_k} and its child S_{j_k} and represents a probabilistic constraint on the corresponding relative pose. We call these edges "probabilistic kinematic links". A given edge may describe:

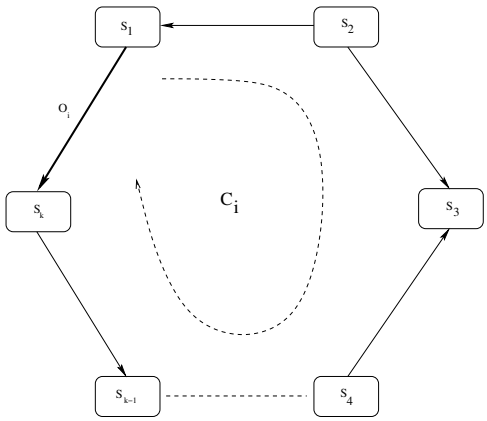


Figure 1: Example of a cycle in the kinematic graph.

- a modeling constraint (a piece of knowledge) on the relative pose of the parent frame and its child,
- a sensor measurement of the pose of a given entity,
- or a constraint we wish to satisfy to solve the problem (an objective value with a given precision, for example).

Each edge $A_{i_k j_k}$ is labeled by:

1. a probability distribution $p(Q_{i_k j_k})$ where $Q_{i_k j_k}$ is the relative pose vector (six-vector) $Q_{i_k j_k} = (t_x t_y t_z r_x r_y r_z)^T$. The first three parameters of this six-vector represent the translation, while the remaining three represent the rotation.
2. possible equality/inequality constraints ($E_k(Q_{i_k j_k}) = 0$,$C_k(Q_{i_k j_k}) \leq 0$). These constraints represent possible geometric relationships between the two geometric entities attached to these two frames. Their shapes depend on the type of the geometric relationship. We implement several relationships between geometric entities in this work, such as points, polygonal faces, edges, spheres and cylinders. The details on equality/inequality constraints induced by these relationships can be found in [Mekhnacha, 1999].
3. a "status" six-vector describing for each parameter of $Q_{i_k j_k}$, its role (nature) in the problem. A status can take one of the three following values:
   - *Unknown* (denoted by X) for parameters representing the unknown variables of the problem and whose values must be found to solve the problem.
   - *Free* (denoted by L) for parameters whose values are only known with a probability distribution. This allows to express uncertainties on the model.
   - *Fixed* (denoted by F) for parameters having known fixed scalar values that cannot be changed.

In the general case, the kinematic graph may contain a set of cycles. The presence of a cycle represents the existence of more than one path between two vertices (frames) of the graph. To ensure the geometric coherence of the model, the computation of the relative pose between these two frames using all paths must give the same value. For each cycle containing k edges (see Fig. 1), we have:

$$T_{S_i S_i} = T_{S_i S_{i+1}}^{d_{S_i S_{i+1}}} * T_{S_{i+1} S_{i+2}}^{d_{S_{i+1} S_{i+2}}} * \cdots * T_{S_{k-1} S_k}^{d_{S_{k-1} S_k}} * T_{S_k S_1}^{d_{S_k S_1}} * T_{S_1 S_2}^{d_{S_1 S_2}} * \cdots * T_{S_{i-1} S_i}^{d_{S_{i-1} S_i}} = I_4, \quad (1)$$

where T_{ij} is the 4×4 homogeneous matrix corresponding to the pose vector Q_{ij}, I_4 is the 4×4 identity matrix and $d_{ij} \in \{-1, 1\}$ is the direction in which the edge A_{ij} has been used.

We call these additional equality constraints the "cycle-closing constraints". They are global constraints involving, for each cycle, all the parameters it contains. The minimal number of cycles allowing coverage of a connected graph having n vertices and m edges is $p = m - n + 1$ [Gondran and Minoux, 1990]. Consequently, we obtain p cycle-closing constraints for a given problem.

3.2 Objective function

Given a probabilistic kinematic graph, we are interested in constructing a marginal distribution over the unknown parameters (parameters having the *unknown* status) of the problem. Maximizing this distribution will provide a solution to the problem.

We define the following sets of propositions:

- A set of p propositions $\{\mathcal{K}_i\}_{i=1}^{p}$ such as:
  $\mathcal{K}_i \equiv$ "cycle c_i is closed".
- A set of m propositions $\{\mathcal{H}_k\}_{k=1}^{m}$ such as:
  $\mathcal{H}_k \equiv$ "$C_k(Q_{i_k j_k}) \leq 0$ and $E_k(Q_{i_k j_k}) = 0$".

If we denote the unknown parameters of the problem by X, a solution to a problem is a value of X that maximizes the marginal distribution

$$p(X|\mathcal{H}_1 \cdots \mathcal{H}_m \mathcal{K}_1 \cdots \mathcal{K}_p).$$

If we denote by L' the concatenation of the parameters having status L and by X the concatenation of the parameters having status X, we can write using the probability calculus:

$$\begin{aligned} p(X|\mathcal{H}_1 \cdots \mathcal{H}_m \mathcal{K}_1 \cdots \mathcal{K}_p) &\propto \int dL' \;\; p(XL'\mathcal{H}_1 \cdots \mathcal{H}_m \mathcal{K}_1 \cdots \mathcal{K}_p) \\ &= p(X) \int dL' \;\; p(L')p(\mathcal{H}_1 \cdots \mathcal{H}_m \mathcal{K}_1 \cdots \mathcal{K}_p|XL'). \end{aligned}$$

To use the global equality constraints (Eq. 1), we take for each cycle $c_i,\ i = 1 \cdots p$ a pose vector we rename O_i (Fig. 1). This pose vector is chosen so that it contains no parameters having the X status. Equation 1 allows us to compute the value of O_i using the values of all the other pose vectors pertaining to c_i:

$$\begin{aligned} O_i &= Q_{S_1 S_k} \\ &= F_i\left(Q_{S_1 S_2}, Q_{S_2 S_3}, \cdots, Q_{S_{k-1} S_k}\right) \\ &= vect\left((mat(Q_{S_1 S_2}))^{d_{S_1 S_2}} * (mat(Q_{S_2 S_3}))^{d_{S_2 S_3}} * \cdots * \left(mat(Q_{S_{k-1} S_k})\right)^{d_{S_{k-1} S_k}}\right), \end{aligned}$$

where

- *vect* is the function allowing to get a pose vector from the corresponding 4×4 homogeneous matrix,
- *mat* is the function allowing to get a 4×4 homogeneous matrix from the corresponding pose vector,
- $d_{ij} \in \{1, -1\}$ denotes the direction in which the edge A_{ij} has been used.

Using this equality constraint cancels the integrals over the parameters of L' that pertain to O_i, because the integrand takes a non-null value only for the point that respect Eq. 1.

For each edge A_{ij}, if we denote by L_{ij} the set of parameters having status L and by X_{ij} the parameters having status X, we can write, using appropriate independence assumptions, the following general form:

$$p(X|\mathcal{H}_1 \cdots \mathcal{H}_m \mathcal{K}_1 \cdots \mathcal{K}_p) \propto p(X)I(X),$$

where

$$\begin{aligned} I(X) = \int dL \;\; & p(L_{i_1 j_1})p(\mathcal{H}_1|X_{i_1 j_1} L_{i_1 j_1}) \\ & \vdots \\ & p(L_{i_{m-p} j_{m-p}})p(\mathcal{H}_{m-p}|X_{i_{m-p} j_{m-p}} L_{i_{m-p} j_{m-p}}) \\ & p_{O_1}(F_1(X, L))p(\mathcal{H}_{m-p+1}|F_1(X, L)) \\ & \vdots \\ & p_{O_p}(F_p(X, L))p(\mathcal{H}_m|F_p(X, L)). \end{aligned} \tag{2}$$

For each cycle c_i, $i = 1 \cdots p$, p_{O_i} denotes the distribution over O_i, while $L \subset L'$ is the concatenation of $L_{i_1 j_1}, \cdots, L_{i_{m-p} j_{m-p}}$.

The distribution $p(X)$ is called the *a priori* distribution over the unknown parameters X (before incorporating the constraints), while the distribution $p(X|\mathcal{H}_1 \cdots \mathcal{H}_m \mathcal{K}_1 \cdots \mathcal{K}_p)$ is called the *a posteriori* distribution over X (after incorporating the constraints).

For each $A_{i_k j_k}, k = 1, \cdots, m-p$, marginalizing (by integration) over the free parameters $L_{i_k j_k}$ allows to take into account the propagation of the uncertainties expressed using the distribution $p(L_{i_k j_k})$ *corrected* using the local constraints $\mathcal{H}_k$.

Maximizing the a posteriori distribution $p(X|\mathcal{H}_1 \cdots \mathcal{H}_m \mathcal{K}_1 \cdots \mathcal{K}_p)$ provides the "Maximum A Posteriori" (MAP) solution of the problem.

4 Resolution method

We described in the previous section how to express a geometric problem as an integration/optimization problem:

$$X^* = \max_X \left[p(X|\mathcal{H}_1 \cdots \mathcal{H}_m \mathcal{K}_1 \cdots \mathcal{K}_p) \right].$$

In this section, we will present the practical numerical methods we used to solve these two problems.

4.1 Numerical integration method

Integral calculus is the basis of Bayesian inference. Unfortunately, analytic methods for integral evaluation seem very limited in real-world applications, where integrands may have complex shapes and integration spaces may have very high dimensionality.

Domain subdivision-based methods (such as trapezoidal or Simpson's methods) are often used for numerical integration in low-dimensional spaces. However, these techniques are poorly adapted for high-dimensional cases.

4.1.1 Monte Carlo methods for numerical estimation

Monte Carlo (MC) methods are powerful stochastic simulation techniques that may be applied to solve optimization and numerical integration problems in large dimensional spaces. Since their introduction in the physics literature in the 1950s, Monte Carlo methods have been at the center of the recent Bayesian revolution in applied statistics and related fields, including econometrics [Geweke, 1996] and biometrics. Their application in other fields such as image synthesis [Keller, 1996] and mobile robotics [Dellaert et al., 1999] is more recent.

Principles

The principle of using Monte Carlo methods for numerical integration is to approximate the integral

$$I = \int f(l) g(l) \, d^d l,$$

by estimating the expectation of the function $g(l)$ under the distribution $f(l)$

$$I = \int f(l) g(l) \, d^d l = \langle g(l) \rangle.$$

Suppose we are able to obtain a set of samples $\{l^{(i)}\}_{i=1}^N$ (d-vectors) from the distribution $f(l)$. We can use these samples to derive the estimator

$$\hat{I} = \frac{1}{N} \sum_i g(l^{(i)}).$$

Clearly, if the vectors $\{l^{(i)}\}_{i=1}^N$ are generated from $f(l)$, the variance of the estimator $\hat{I} = \frac{1}{N} \sum_i g(l^{(i)})$ will decrease as $\frac{\sigma^2}{N}$, where σ^2 is the variance of g:

$$\sigma^2 = \int f(l)(g(l) - \hat{g})^2 \, d^d l,$$

and $\hat{g}$ is the expectation of g.

This result is one of the important properties of Monte Carlo methods:
"The accuracy of Monte Carlo estimates is independent of the dimensionality of the integration space".

4.1.2 Using MC methods for our application

Using an MC method to estimate the integral (2) requires the following steps.

1. Sample a set of N points $\{L^{(i)}\}_{i=1}^{N}$ from the prior distribution $p(L)$ such that the sampled points respect local equality/inequality constraints (i.e., $\{\mathcal{H}_i\}_{i=1}^{m-p}$ have the value *true*).

2. Estimate the integral $I(X)$ using the set $\{L^{(i)}\}_{i=1}^{N}$ of points as follows.

$$\hat{I}(X) = \frac{1}{N}\sum_{i=1}^{N} \quad p_{O_1}(F_1(X, L^{(i)}))p(\mathcal{H}_{m-p+1}|F_1(X, L^{(i)}))$$
$$\vdots$$
$$p_{O_p}(F_p(X, L^{(i)}))p(\mathcal{H}_m|F_p(X, L^{(i)})).$$

Points sampling
The set of N points used to estimate the integral may be sampled in various ways. Since parameters pertaining to different pose vectors are independent, we can decompose the "state vector" L to $m-p$ components $\{L_{i_k j_k}\}_{k=1}^{m-p}$ and apply a local sampling algorithm [Geweke, 1996, Neal, 1993]. Using a local sampling algorithm, updating the state vector L

$$L^{(t)} = (L^{(t)}_{i_1 j_1}, L^{(t)}_{i_2 j_2}, \cdots, L^{(t)}_{i_k j_k}, \cdots, L^{(t)}_{i_{m-p} j_{m-p}})$$

only requires updating one component $L_{i_k j_k}$

$$L^{(t+1)} = (L^{(t)}_{i_1 j_1}, L^{(t)}_{i_2 j_2}, \cdots, L^{(t+1)}_{i_k j_k}, \cdots, L^{(t)}_{i_{m-p} j_{m-p}}).$$

N iterations of this procedure give us the set $\{L^{(i)}\}_{i=1}^{N}$, which will be used to estimate the integral.

To update a component $L_{i_k j_k}$ (a set of parameters pertaining to the same pose vector $Q_{i_k j_k}$), we must take into account possible dependencies between these parameters. Consequently, we face two problems.

- **Candidate point sampling**
  A candidate $L^c_{i_k j_k}$ is drawn from the distribution $p(L_{i_k j_k})$. Direct sampling methods from simple distributions such as uniform distributions and Gaussians are available. If we do not have a direct sampling method from $p(L_{i_k j_k})$ at our disposal, an indirect sampling method must be used. In this work, we chose a Metropolis sampling algorithm [Geweke, 1996, Neal, 1993].

- **Candidate validity checking**
  Suppose we have a geometric relationship between two geometric entities E_i and E_j. A geometrical calculus depending on the type of this relationship allows checking of the constraint $C_k(Q_{i_k j_k}) \leq 0$. If this constraint is respected (i.e., $p(\mathcal{H}_k|X_{i_k j_k} L_{i_k j_k}) = 1$), the candidate $L^c_{i_k j_k}$ is accepted, otherwise it is rejected. Figure 2 shows a *Face-On-Face* relationship example.

Optimization of computation time
Using a local sampling method to update the state vector L allows a reduction in the computation time of the estimates of integrals. If, for a given point $L^{(t)}$, we denote the values of functions $F_i(X)$ by $F_i^{(t)}(X)$, $\quad i = 1 \cdots p$, then the values of $F_i(X)$ in the next step $F_i^{(t+1)}(X)$ are obtained by partly updating $F_i^{(t)}(X)$.

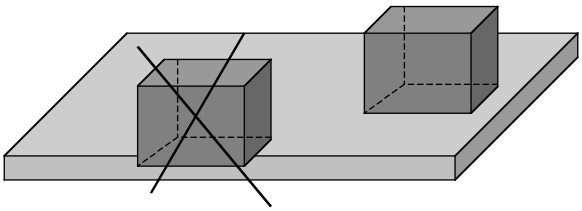


Figure 2: The candidate point is rejected because it does not respect the *Face-On-Face* constraint.

4.2 Optimization method

The optimization method to be chosen for our application must satisfy a set of criteria in relation to the shape and nature of the function to optimize. The method must:

1. be global, because the function to optimize is often multimodal,
2. allow multiprecision computation of the objective function. Its estimation with high accuracy may require long computation times,
3. allow parallel implementation to improve efficiency.

For our application, we chose a genetic algorithm that satisfies these criteria. First, we present the general principles of these algorithms. Then, we discuss the practical problems we faced when using standard genetic algorithms in our application and give the required improvements.

4.2.1 Principles of Genetic Algorithms

Genetic algorithms (abbreviated GA) are stochastic optimization techniques inspired by the biological evolution of species. Since their introduction by Holland [Holland, 1975] in the seventies, these techniques have been used for numerous global optimization problems, thanks to their ease of implementation and their independence of application fields. They are widely used in a large variety of domains including artificial intelligence [Grefenstette, 1988] and robotics [Mazer et al., 1998].

The goal of a GA is to find a global optimum of a given function F over a search space S.

During an initialization phase, a set of points (*individuals*) are drawn at random from the search space S that is discretized with a given resolution. This set of points is called a *population*.

Each individual I is coded by a string of bits. It represents a solution of the problem and its *adequacy* is measured by a value $F(I)$.

The fundamental principle of genetic algorithms is: "the better the adequacy of an individual, the larger is the probability of selecting it for reproduction". "Genetic operators" are applied to the selected individuals to generate new ones. For a given size of population, better individuals obtained by reproduction replace initial ones. This process is iterated until a convergence criterion is reached.

The standard sequential genetic algorithm can be described as follows. First, an initial population is drawn at random from the search space and the following cycle is then performed (see Fig. 3).

1. **Selection:** Using the function F, pairs of individuals are selected. The probability of selecting an individual I grows with the value of $F(I)$ for this individual.
2. **Reproduction:** Genetic operators are applied to the selected individuals to produce new ones.
3. **Evaluation:** The values of F are computed for the new individuals.
4. **Replacement:** Individuals in the current population are replaced by better new individuals.

Many genetic operators are available. However, the more commonly used are "mutation" and "cross-over". For a given pair of individuals, the cross-over operator consists of first cutting the two strings of bits in a randomly chosen place and then building two new individuals by interchanging the cut parts of the starting strings. The mutation operator consists of flipping some randomly chosen bits of an individual.

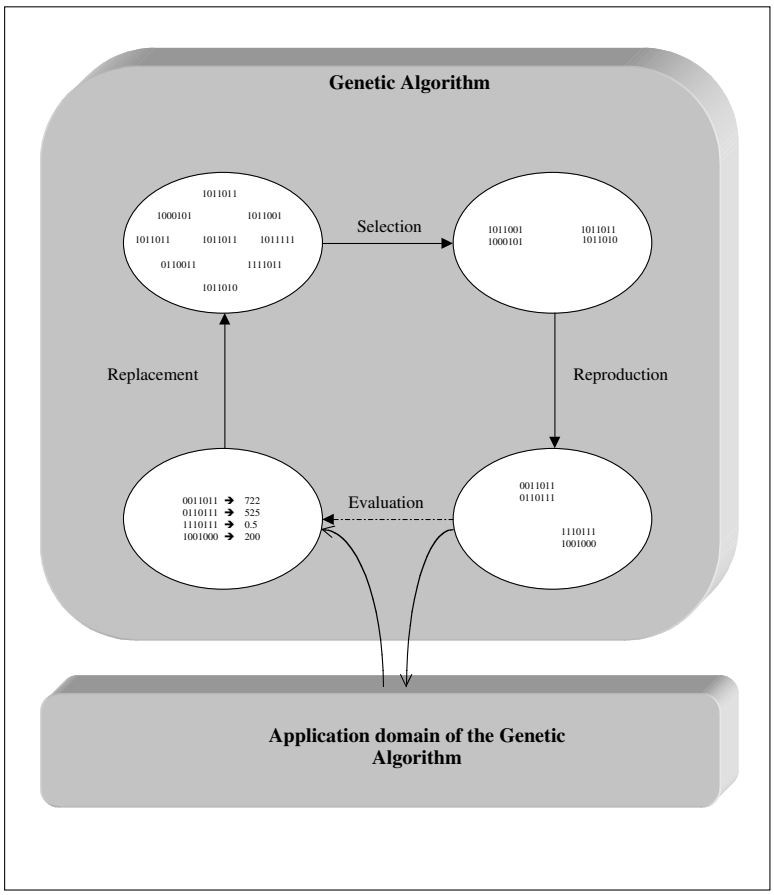


Figure 3: Genetic Algorithm iterations.

In this work, we use a population with a constant size of 100 individuals. We discretize the search space with a resolution of 10^{-4} rad for orientation parameters and of 10^{-3} mm for translation ones. For example, an orientation parameter that takes a value between 0.0 and 2π rad (discretized with a 10^{-4} rad resolution) is coded on a 16 bits string ($0.0 \equiv \underbrace{00\cdots0}_{16}$, while $2\pi \equiv \underbrace{11\cdots1}_{16}$). A string coding a given configuration is simply the concatenation of the strings codings each parameter. For reproduction, we use both the cross-over and the mutation operators. First, we use the cross-over operator to get an intermediate individual (string). Then, the mutation operator is applied with a probability of 0.2 on this intermediate individuals to get the final individuals.

In the following, we will use $G(X)$ to denote the objective function $p(X|\mathcal{H}_1 \cdots \mathcal{H}_m \mathcal{K}_1 \cdots \mathcal{K}_p)$.

4.2.2 Narrowness of the objective function - constraint relaxation

In our applications, the objective function $G(X)$ may have a narrow support (the region where the value is not null) for very constrained problems. The initialization of the population with random individuals from the search space may give null values of the function $G(X)$ for most individuals. This will make the evolution of the algorithm very slow and its behavior will be similar to random exploration.

To deal with this problem, a concept inspired from classical simulated annealing algorithms consists of introducing a notion of "temperature". The principle is to first widen the support of the function by changing the original function to obtain non-null values even for configurations that are not permitted. To do so, we introduce an additional parameter we call T (for temperature) for the objective function $G(X)$. Our goal is to obtain another function $G^T(X)$ that is smoother and has wider support, with

$$\lim_{T \to 0} G^T(X) = G(X).$$

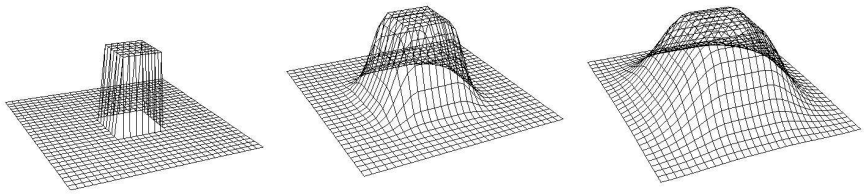


Figure 4: The distribution corresponding to inequality constraints induced by a *Point-On-Face* relationship for a square face at different values of temperature. The left figure shows the original constraints ($T = 0$), while the middle and the right ones show these constraints relaxed at ($T = 50$) and ($T = 100$) respectively.

To widen the support of $G(X)$, all elementary terms (distributions) of $G(X)$ are widened, namely:

- distributions $p_{O_i}(F_i(X, L))$, where $i = 1 \cdots p$.
- inequality constraints $p(\mathcal{H}_{m-p+j}|F_j(X, L))$, where $j = 1 \cdots p$.

For example:

- for a Gaussian distribution:

$$\begin{aligned} f(x) &= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \\ f^T(x) &= \frac{1}{\sqrt{2\pi}\sigma(1+T)} e^{-\frac{1}{2}\frac{(x-\mu)^2}{[\sigma(1+T)]^2}} \end{aligned}$$

- for an inequality constraint over the interval $[a, b]$:

$$\begin{aligned} f(x) &= \begin{cases} 1 & \text{if } a \le x \le b \\ 0 & \text{else} \end{cases} \\ f^T(x) &= \begin{cases} 1 & \text{if } a \le x \le b \\ e^{-\frac{(x-a)^2}{(b-a)T}} & \text{if } x < a \\ e^{-\frac{(x-b)^2}{(b-a)T}} & \text{otherwise} \end{cases} \end{aligned}$$

In the general case, inequality constraints may be more complex. Figure 4 shows the case of a *Point-On-Face* inequality constraint for a square face[1].

4.2.3 Accuracy of the estimates - multiprecision computing

The second problem we must face is that only an approximation $\hat{G}(X)$ of $G(X)$ is available, of unknown accuracy. Using a large number of points to obtain sufficient accuracy may be very expensive in computation time, so that use of a large number of points in the whole optimization process is inappropriate.

Since the accuracy of the estimate $\hat{G}(X)$ of the objective function depends on the number of points N used for the estimation, we introduce N as an additional parameter to define a new function $\hat{G}_N(X)$.

Suppose we initialize and run for some cycles a genetic algorithm with $\hat{G}_{N_1}(X)$ as evaluation function. The population of this GA is a good initialization for another GA having $\hat{G}_{N_2}(X)$ as evaluation function with $N_2 > N_1$.

[1]The formulas we obtain by relaxation are not effective probability distributions, but only "kernels," because they do not satisfy the normalization condition $\int_{-\infty}^{\infty} f(x)dx = 1$. Since we are interested in optimizing the marginal distribution, computing the normalization factor is not necessary.

4.2.4 General optimization algorithm

In the following, we label the evaluation function (the objective function) by the temperature T and the number N of points used for estimation. It will be denoted by $G_N^T(X)$.

Our optimization algorithm may be described by the following three phases.

1. Initialization and initial temperature determination.

2. Reduction of temperature to recreate the original objective function.

3. Augmentation of the number of points to increase the accuracy of the estimates.

Initialization: The population of the GA is initialized at random from the search space. To minimize computing time in this initialization phase, we use a small number N_0 of points to estimate integrals. We propose the following algorithm as an automatic initialization procedure for the initial temperature T_0, able to adapt to the complexity of the problem.

INITIALIZATION(GA)
BEGIN
FOR each population[i] $\in$ GA's population DO
REPEAT
population[i] = random(S)
value[i] = $G_{N_0}^T$(population[i])
if (value[i] == 0.0)
T = T + ΔT
UNTIL (value[i]> 0.0)
FEND
Re-evaluate(population)
END
where ΔT is a small increment value.

Temperature reduction: To obtain the original objective function ($T = 0.0$), a possible scheduling procedure consists of multiplying the temperature, after running the GA for a given number of cycles nc_1, by a factor α ($0 < \alpha < 1$). A small value for α may cause the divergence of the algorithm, while a value too close to 1.0 may considerably increase the computation time. In this work, the value of α has been experimentally fixed to 0.8. We can summarize the proposed algorithm as follows.

TEMP_REDUCTION(GA)
BEGIN
WHILE (T > T_ϵ) DO
FOR i=1 TO nc_1 DO
Run(GA)
FEND
T = T * α
WEND
T = 0.0
Re-evaluate(population)
END
where T_ϵ is a small threshold value.

Augmenting the number of points: At the end of the temperature reduction phase, the population may contain several possible solutions for the problem. To decide between these solutions, we must increase the accuracy of the estimates. One approach is to multiply N, after running the GA for a given number of cycles nc_2, by a factor β ($\beta > 1$) so that the variance of the estimate is divided by β:

$$Var(G_{\beta * N}^0(X)) = \frac{1}{\beta} Var(G_N^0(X)).$$

We can describe this phase by the following algorithm.

N_POINTS_AUGMENTATION(GA)
BEGIN
WHILE (N < N_{max}) DO
FOR i=1 TO nc_2 DO
Run(GA)
FEND
N = N * β
WEND
END
where N_{max} is the number of points that allows convergence of the estimates $\hat{G}_N^0(X)$ for all individuals of the population.

In this work, the value of β has been fixed to 2.

5 Overview of the implementation

In this section, we present an overview of the implementation of the CAD modeler that follows the principles presented above.

5.1 Specification language

A workcell is constructed by evaluating a script file. This script contains a set of Lisp-like instructions used to:

- create geometric entities,
- create parts,
- describe probabilistic constraints between parts.

After evaluation of the script, a graphic model of the cell is constructed and passed to a 3D viewer.

5.1.1 Geometric entities creation

Geometric entities creation uses a specialized method for each entity. When creating an entity, a frame attached to it is automatically created. The following methods are used:

- *New_Vertex*(x, y)
- *New_Edge*(vertex1, vertex2)
- *New_Face*(list_of_vertices)
- *New_Sphere*(center, radius)
- *New_Cylinder*(center, radius, direction, length)

5.1.2 Parts creation

A part is a set (possibly empty when only the attached frame is modeled) of geometric entities. This set of entities is given as a parameter when creating the part. An additional graphic object can be added to give a realistic graphic representation. We use the following method.

- *New_Part*(list_of_geom_entities, add_graph_obj)

5.1.3 Probabilistic kinematic links description

Creating a probabilistic kinematic link between two frames or two geometric entities uses the following instructions to create the probabilistic kinematic link and use it to attach entities.

- *New_Link*(status_vector, distribution)

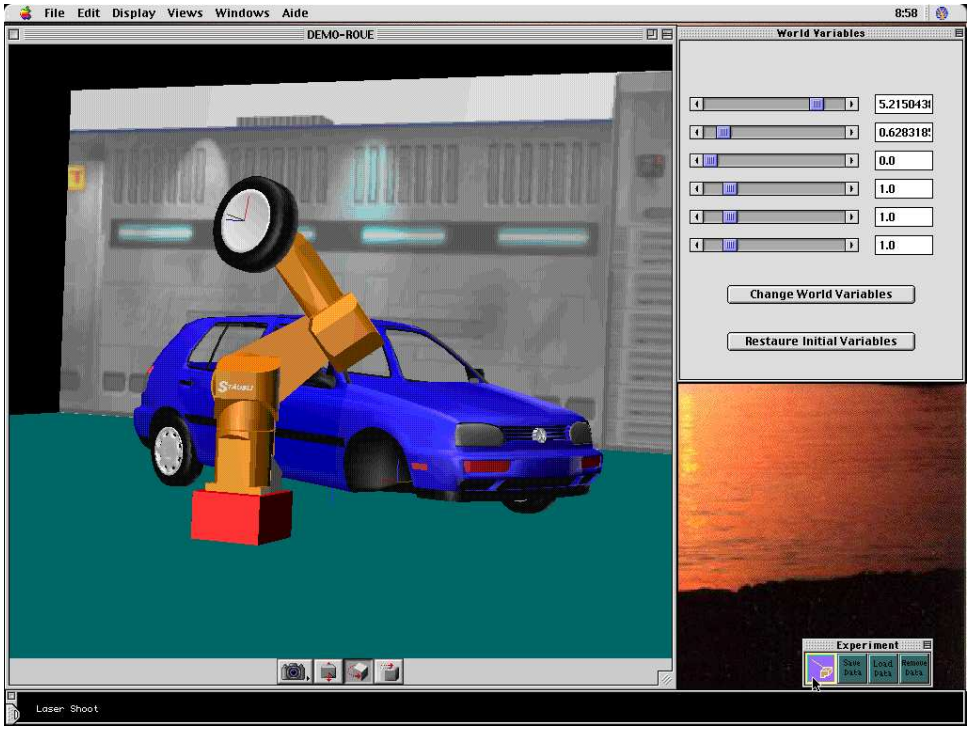


Figure 5: A screen copy of our CAD modeler. It shows an application of our method: the problem of positioning a robot arm to allow maximum accuracy when mounting a car wheel.

- *Attach*(parent_item, child_item, link)

If *parent_item* and *child_item* are geometric entities (instead of simple frames), the corresponding equality and inequality constraints are automatically added by the system.

5.2 Graphics system and geometric uncertainties visualization

The use of graphic support has an indisputable interest for 3D geometric workcells modeling and for appreciation of the calculated solutions for a given problem. Moreover, it may allow in our case, a visualization of geometric uncertainties and make their perception easier.

5.2.1 Graphics system

A workcell is constructed by evaluating a script containing a set of instructions, as described above. Besides the construction of the internal representation of the workcell, the evaluation of the script constructs a graphic model corresponding of this workcell and passes it as a parameter to the invoked 3D viewer (see Fig. 5).

The implemented 3D-visualization system is based on the *Quickdraw3D* graphic library developed and proposed by *Apple* for MacOS and Windows 95/98/NT platforms. This library proposes primitives for creating, positioning and displacement of geometric features. It also proposes an integrated 3D viewer that can be easily invoked from any application. The application must construct a *graphic group* to be viewed and pass it as a parameter when invoking the viewer.

5.2.2 Geometric uncertainties visualization

Since the relative poses of parts are described as probability distributions instead of single scalar values, we are interested in developing a graphic representation that takes account of this probabilistic aspect of poses on a display screen.

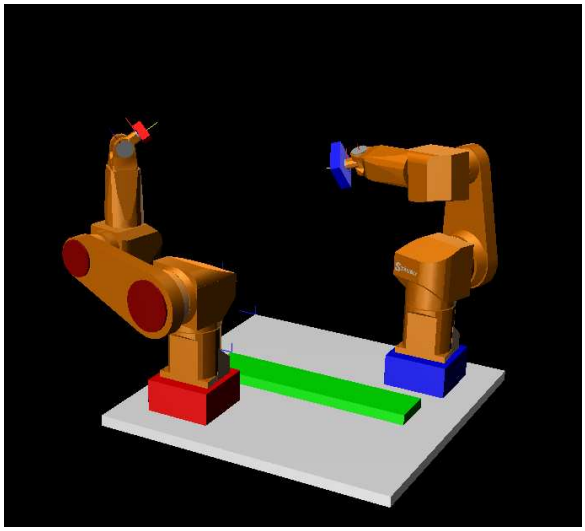


Figure 6: Kinematics inversion example using two Stäubli Rx90 arms.

The proposed method is to simulate uncertainties in the poses of parts. The principle is to use a Monte Carlo simulation by sampling the values of the parameters of the poses in the workcell from probability distributions over these parameters. Instead of displaying a part in a fixed pose in the graphic scene, the part is displayed, with a given frequency, in the poses obtained by this sampling. If the frequency of sampling is high enough, this will give a good visual perception of the geometric uncertainties in the model of the workcell.

This *visualization* of uncertainties allows a more concrete perception of their propagation in a given configuration. In particular, it allows *graphic* comparison of two different solutions for a given geometric problem.

6 A kinematics inversion example

In this section we describe how to use our CAD modeler for concrete problems. We present in detail a kinematics inversion problem under geometric uncertainties.

6.1 Problem description

Using two Stäubli Rx90 robot arms with six revolute joints, we are interested in placing two prismatic parts one against the other. The only constraint is that a face of the first part will be in a *Face-On-Face* relationship with a face of the second.

The two arms are modeled as a set of parts attached to each other using probabilistic kinematic links. We assume that the more significant uncertainties are on zero positions of the joints. The two parts are also attached to arms' end effectors using probabilistic kinematic links. The added constraint we wish to satisfy to solve the problem is represented by a link between the two faces to place in *Face-On-Face* relationship. We use for this link three Gaussians on the three constrained parameters t_z, r_x and r_y with zeros as mean values and 0.5 mm, 0.01 rad and 0.01 rad respectively as standard deviations. Figure 6 shows the two arms, while Fig. 7 gives the corresponding kinematic graph.

We suppose in this example that the zero position uncertainties of the arm on the right of Fig. 6 ($Right\_Arm$) are five times more important than the ones of the arm on the left ($Left\_Arm$) (for each joint, we suppose a Gaussian distribution on the zero position with 0.01 rad as the standard deviation for $Left\_Arm$ and with 0.05 rad for $Right\_Arm$). Our aim is to comment qualitatively on the solution obtained and to show the importance of taking uncertainties propagation into account when choosing a solution.

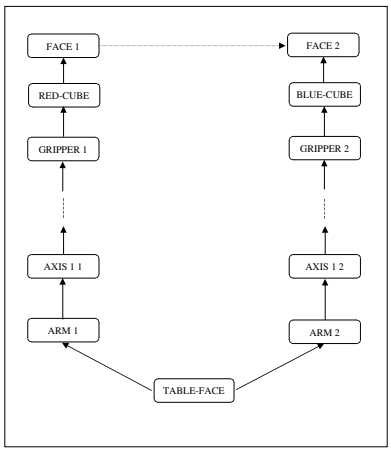


Figure 7: The corresponding kinematic graph.

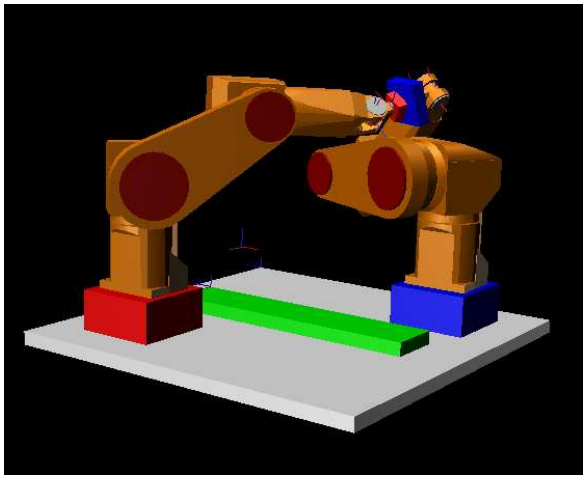


Figure 8: The solution obtained by the system.

6.2 Results

Figure 8 shows the solution obtained by the system. This solution gives maximal precision for the required *Face-On-Face* relationship because:

1. *Right_Arm* (the less accurate) is coiled to minimize the propagation of the uncertainties on its zero positions.

2. Rotation axes are perpendicular to the common normal of the two faces.

Table 1 summarizes the problem complexity and the system performances for this problem using a PowerPC G3/400 machine.

6.3 Discussion

This example shows how the proposed method takes geometric uncertainties into account in a general and homogeneous way. No assumptions have been made, either on the uncertainties models (shapes of the used distributions), or on the linearity of the model or the possibility of it being linearized. It also shows how possible redundancy of the system relating to the required task is used to find the most accurate solution.

Integration space dimension	50
Optimization space dimension	12
Number of cycles	1
Number of frames	28
Number of inequality constraints	16
Computation time (seconds)	13

Table 1: Some parameters summarizing the problem complexity and the system performances for this kinematics inversion problem.

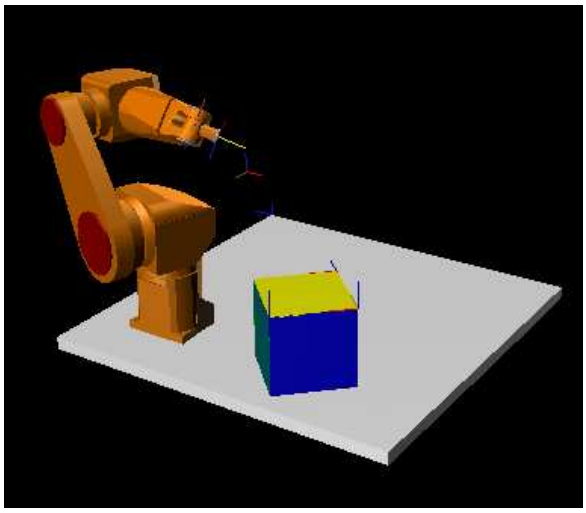


Figure 9: A parallelepiped pose and dimensions calibration problem using contact relationships.

7 A calibration example

In this section, we present a calibration problem.

7.1 Problem description

The purpose of this example is to calibrate the pose and the size of a 3-D part. More precisely, we are interested in identifying the parameters of the pose of a parallelepiped on a table and the three dimensions of this parallelepiped (see Fig. 9).

The experimental protocol is as follows. For each measurement, a six DOF arm is moved to a configuration that allows obtaining a contact between a touch sensor mounted on the end effector of the arm and a face of the parallelepiped. A set of N contacts between the touch sensor and the faces will give the set of N measurements (configurations that allow contact) we will use for calibration (see Fig. 10). The geometric model of the arm is the same used for $Left\_Arm$ in the previous example.

We suppose that the parallelepiped lies on the table. Consequently, we have to identify only the x and y position parameters and the α orientation parameter. For the size of the parallelepiped, we have to identify the parameters sx, sy and sz representing distances between each pair of parallel faces. We used for this example a set of ten contacts. For each face (except for the inferior face which lies in the table), two measurements have been taken. Figure 11 shows the contact points and the corresponding faces to put back in contact to solve this calibration problem, while Fig. 12 gives the kinematic graph corresponding to this problem.

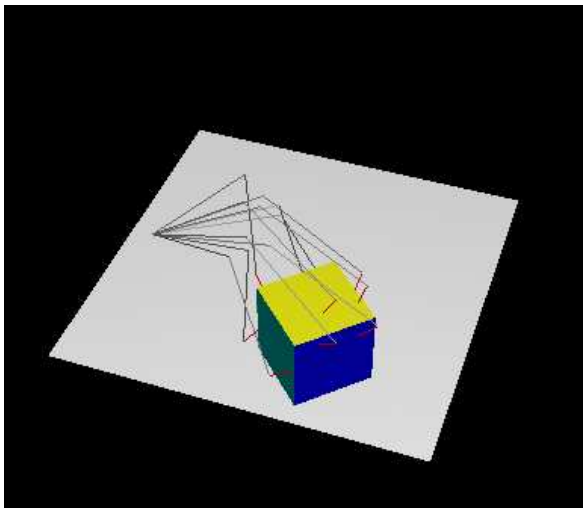


Figure 10: The set of contacts to use for calibration.

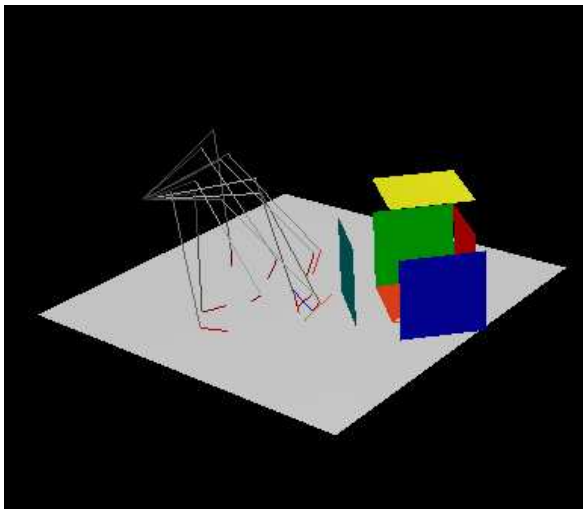


Figure 11: Contact points and the parallelepiped faces to put back in contact to solve the calibration problem.

7.2 Results

The a priori distribution $p(X)$ on the search space $X = (x, y, \alpha, sx, sy, sz)^T$ expresses our prior knowledge on the parameters to be identified. For this example, we have assumed an uniform distribution $p(X)$ to express the fact that no initial estimation of these parameters is available.

We summarize the problem complexity and the system performances for this problem using a PowerPC G3/400 machine in Tab. 2.

The simulated contacts have been taken at non-null distances between the touch sensor and the parallelepiped faces. Table 3 gives error values for the ten measurements. We have to underline that all these contact errors have positive values because the touch sensor cannot overlap the parallelepiped.

Table 4 gives simulation values of the parameters to calibrate and the values obtained after calibration.

7.3 Discussion

This example presents an application of our method for parameter identification problems. We show especially that using this method allows:

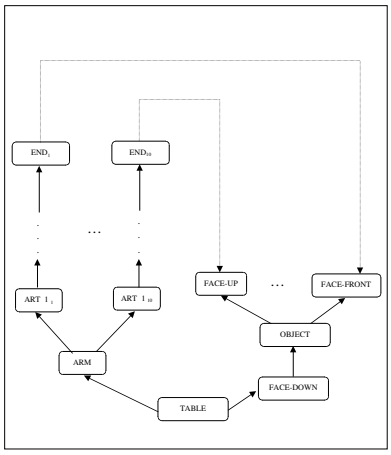


Figure 12: The kinematic graph for the calibration problem.

Integration space dimension	30
Optimization space dimension	6
Number of cycles	10
Number of frames	77
Inequality constraints number	40
Computation time (seconds)	23

Table 2: Some parameters summarizing the problem complexity and the system performances for this calibration problem.

- To take into account prior information on the parameters to estimate.
- To take into account, for each measurement (contact), its accuracy by propagating the uncertainties of the arm model. This allows an implicit weighting of these measurements (the more accurate the measurement, the more importance it has in the calibration process).
- To take into account prior information on the used measurement tool. In this particular example where measurements are contact relationships, we have expressed the non-overlap phenomenon using a non-symmetrical distribution

$$p(t_z) = \begin{cases} \frac{2}{\sqrt{2\pi}\sigma_c} e^{-\frac{1}{2}\frac{t_z^2}{\sigma_c^2}} & \text{if } t_z \geq 0 \\ 0 & \text{else} \end{cases}$$

  where σ_c was 0.5 mm.

8 Conclusion and Future Research

We have presented a generic approach for geometric problem specification and resolution using a Bayesian framework. We have shown how a given problem is first represented as a probabilistic kinematic graph and then expressed as an integration/optimization problem. Appropriate numerical algorithms used to apply this methodology are also described. For generality, no assumptions have been made on the shapes of distributions or on the amplitudes of uncertainties.

Numerous geometric problems have been specified and resolved using our system. We have presented in this paper a kinematics inversion under uncertainties problem and a part pose and shape calibration.

Experimental results made on our system have demonstrated the effectiveness, the robustness, and the homogeneity of representation of our approach. However, additional studies are required to improve both the integration and the optimization algorithms.

	Contact 1	Contact 2	Contact 3	Contact 4	Contact 5
Simulated errors (mm)	0.677	0.567	0.303	0.792	0.724
	Contact 6	Contact 7	Contact 8	Contact 9	Contact 10
Simulated errors (mm)	0.791	0.883	0.858	0.383	0.111

Table 3: Error values used when simulating contacts.

	x (mm)	y (mm)	α (rad)	sx (mm)	sy (mm)	sz (mm)
Simulation values	900.000	-900.000	0.7854	300.000	300.000	300.000
Calibration results	900.195	-900.000	0.7853	299.238	299.238	299.238

Table 4: Initial values (simulation values) of the parameters to calibrate and calibration results.

For the integration problem, numerical integration can be avoided when the integrand is a product of generalized normals (Dirac delta functions and Gaussians) and when the model is linear or can be linearized (errors are small enough). The optimization algorithm may also be improved by using a local derivative-based method after the convergence of our genetic algorithm.

Future work will aim at allowing the use of high-level sensors such as vision-based ones. We are also considering extending our system so that it can include non-geometrical parameters (inertial parameters for example) in the problem specification.

References

[Alami and Simeon, 1994] Alami, R. and Simeon, T. (1994). Planning robust motion strategies for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 1312–1318, San Diego, California.

[Bar-Shalom and Fortmann, 1988] Bar-Shalom, Y. and Fortmann, T. E. (1988). *Tracking and Data Association*. Academic Press.

[Bessière et al., 1998] Bessière, P., Dedieu, E., Lebeltel, O., Mazer, E., and Mekhnacha, K. (1998). Interprétation ou description (I): Proposition pour une théorie probabiliste des systèmes cognitifs sensori-moteurs. *Intellectica*, 26-27:257–311.

[Dellaert et al., 1999] Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte Carlo localization for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Detroit, MI.

[Geweke, 1996] Geweke, J. (1996). Monte Carlo simulation and numerical integration. In Amman, H., Kendrick, D., and Rust, J., editors, *Handbook of Computational Economics*, volume 13, pages 731–800. Elsevier North-Holland, Amsterdam.

[Gondran and Minoux, 1990] Gondran, M. and Minoux, M. (1990). *Graphes et Algorithmes*. Eyrolle, Paris.

[Grefenstette, 1988] Grefenstette, J. J. (1988). Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning*, 3:225–245.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

[Jaakkola and Jordan, 1999] Jaakkola, T. and Jordan, M. (1999). Variational probabilistic inference and the QMR-DT network. *J. Artif. Intellig. Res. (JAIR)*, 10:291–322.

[Jaynes, 1996] Jaynes, E. T. (1996). Probability Theory - The Logic of Science. Unfinished book publicly available at http://bayes.wustl.edu.

[Keller, 1996] Keller, A. (1996). The fast calculation of form factors using low discrepancy point sequence. In *Proc. of the 12th Spring Conf. on Computer Graphics*, pages 195–204, Bratislava.

[Laplace, 1774] Laplace, P. S. (1774). Mémoire sur la probabilité de causes par les évenements. *Mémoire de l'Académie Royale des Sciences*, 6:621–656.

[Lozano-Pèrez, 1987] Lozano-Pèrez, T. (1987). A simple motion-planning algorithm for general robot manipulators. *IEEE J. of Robotics and Automation*, 3(3):224–238.

[Mazer et al., 1998] Mazer, E., Ahuactzin, J., and Bessière, P. (1998). The Ariadne's Clew algorithm. *J. Artif. Intellig. Res. (JAIR)*, 9:295–316.

[Mekhnacha, 1999] Mekhnacha, K. (1999). *Méthodes probabilistes Bayesiennes pour la prise en compte des incertitudes géométriques: application à la CAO-robotique*. Thèse de doctorat, Inst. Nat. Polytechnique de Grenoble, Grenoble, France.

[Mekhnacha et al., 2000] Mekhnacha, K., Mazer, E., and Bessière, P. (2000). A Bayesian CAD system for robotic applications. In *Proc. of the IASTED Int. Conf. on Modelling and Simulation*, pages 527–534, Pittsburgh, PA, USA.

[Moravec, 1988] Moravec, H. P. (1988). Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74.

[Neal, 1993] Neal, R. M. (1993). Probabilistic inference using Markov Chain Monte Carlo methods. Research Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto.

[Owen, 1996] Owen, J. C. (1996). Constraints on simple geometry in two and three dimensions. *Int. J. of Computational Geometry and Applications*, 6(4):421–434.

[Presse and Gautier, 1992] Presse, C. and Gautier, M. (1992). Bayesian estimation of inertial parameters of robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 364–369, Nice, France.

[Puget, 1989] Puget, P. (1989). *Vérification-Correction de programme pour la prise en compte des incertitudes en programmation automatique des robots*. Thèse de doctorat, Inst. Nat. Polytechnique de Grenoble, Grenoble, France.

[Sanderson, 1997] Sanderson, A. C. (1997). Assemblability based on maximum likelihood configuration of tolerances. In *Proc. of the IEEE Symposium on Assembly and Task Planning*, Marina del Rey, CA.

[Taylor, 1976] Taylor, R. (1976). *A synthesis of manipulator control programs from task-level specifications*. Ph.d thesis, Stanford University, Computer Science Department.

[Thrun, 1998] Thrun, S. (1998). Bayesian landmark learning for mobile robot localization. *Machine Learning*, 33(1):41–76.

[Weiss and Adelson, 1998] Weiss, Y. and Adelson, E. H. (1998). Slow and smooth: a Bayesian theory for the combination of local motion signals in human vision. Research Report AI Memo 1624, MIT.

[Zhang and Augeras, 1992] Zhang, Z. and Augeras, O. (1992). *3D Dynamic Scene Analysis: A Stereo Based Approach*. Springer, Berlin, Heidelberg.

Proceedings of the 2004 IEEE
International Conference on Robotics & Automation
New Orleans, LA • April 2004

Hierarchies of probabilistic models of navigation: the Bayesian Map and the Abstraction operator

Julien Diard, Pierre Bessière and Emmanuel Mazer
Laboratoire GRAVIR / IMAG – CNRS
INRIA Rhône-Alpes, 655 avenue de l'Europe
38330 Montbonnot Saint Martin FRANCE
Julien.Diard@free.fr

Abstract— **This paper presents a new method for probabilistic modeling of space, called the *Bayesian Map* formalism. It offers a generalization of some common approaches found in the literature, as it does not constrain the dependency structure of the probabilistic model. The formalism allows incremental building of hierarchies of models, by the use of the *Abstraction* operator. In the resulting hierarchy, localization in the high level model is based on probabilistic competition of the lower level models. Experimental results validate the concept, and hint at its usefulness for large scale scenarios.**

I. Introduction

In robotics, modeling the environment that a robot has to face in a navigation task is a crucial problem, that has received a lot of attention in the community. The most promising approaches rely on the probability calculus, especially for its capacity to handle incomplete models and uncertain information. These approaches include – but are far from limited to – Kalman Filters [1], Markov Localization models [2], (Partially or Fully) Observable Markov Decision Processes [3], and Hidden Markov Models [4]. We will here assume that the reader has some familiarity with these approaches.

In this domain of probabilistic modeling for robotics, hierarchical solutions are currently flourishing. The more active domain in this regard is decision theoretic planning: one can find variants of MDPs that accommodate hierarchies or that select automatically the partition of the state-space (see for instance [5], [6], or browse through the references in [7]). More exceptionally, one can find hierarchical POMDPs, as in [7], which is arguably the work that bears the most resemblance to the one presented here, although we do not use reward functions in this work. The current work can also be related to Thrun's object mapping paradigm [8], in particular concerning the aim of transferring some of the knowledge the programmer has about the task, to the robot. Some hierarchical approaches outside of the MDP community include Hierarchical HMMs and their variants (see [9] and references therein), which, unfortunately, rely on the notion of final state of the automata. Another class of approaches relies on the extraction of a graph from a probabilistic model, like for example a Markov Localization model [10], or a MDP [11]. Using such deterministic notions is inconvenient in a purely probabilistic approach, as we are pursuing here. Indeed, the current work uses probabilities in all layers of the hierarchy of representations, allowing us to propagate and handle uncertainties in a uniform and formally coherent manner.

Moreover, the main philosophy used by all the previous approaches is to try to extract, from a very complex but intractable model, a hierarchy of smaller models. Of course, *automatically* selecting the relevant decomposition of a problem into sub-problems is quite a challenge – this challenge being far from restricted to the domain of navigation for robots facing uncertainties.

We pursue here an alternate route, investigating how, starting from a set of simple models, one can combine them for building more complex models. The goal of this paper is therefore to present a new formalism for building models of the space in which a robot has to navigate (the *Bayesian Map* model), and a method for combining such maps together in a hierarchical manner (the *Abstraction operator*).

This formalism allows for a new representation of space, in which the final program is built upon many imbricate models, each of them deeply rooted into lower level sensorimotor relationships. Such hierarchies of sensorimotor models seem relevant to biologically inspired models, as it appears that no single metric model can account alone for large scale navigation capacities of animals (see [12], [13]).

We will also argue that our approach draws away from the usual characteristics of the common models of space (Section II-C), and that it is also more general than these models (Sections III and VI). For brevity, this paper will discuss neither of the learning methods that can be included into Bayesian Maps (mapping process), nor of another operator for merging Bayesian Maps (the Superposition operator). Preliminary work about these issues and all the details missing in the current paper can be found in Diard's Ph.D. thesis [14].

The rest of this paper is organized as follows. Section II presents the Bayesian Robot Programming methodology, and discusses some of its characteristics. Sections III and IV will quickly define our notion of Bayesian Map, and the Abstraction operator, respectively. The paper concludes on the presentation of experimental results, Section V.

II. Bayesian Robot Programming

The work we present here is based on BRP, a Bayesian Robot Programming methodology. We summarize it here, but

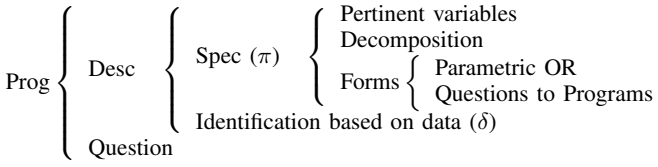


Fig. 1. Structure of a Bayesian Robotic Program.

still invite the reader to refer to [15] for all the details.

A. Definition

In the BRP formalism, a bayesian robotic program is a structure (see Fig. 1) made of two components.

The first is a *declarative* component, where the user defines a **description**. The purpose of a description is to specify a method to compute a joint distribution over a set of relevant variables $\{X_1, X_2, \ldots, X_n\}$, given a set of experimental data δ and preliminary knowledge π. This joint distribution is denoted $P(X_1\ X_2\ \ldots\ X_n \mid \delta\ \pi)$. To specify this distribution, the programmer first lists the pertinent variables (and defines their domains), then decomposes the joint distribution as a product of simpler terms (possibly stating conditional independence hypotheses so as to simplify the model and/or the computations), and finally, assigns forms to each term of the selected product (these forms can be parametric forms, or recursive questions to other bayesian programs). If there are free parameters in the parametric forms, they have to be assessed. They can be given by the programmer (*a priori* programming) or computed on the basis of a learning mechanism defined by the programmer and some experimental data δ.

The second component is of a *procedural* nature, and consists of using the previously defined description with a **question**, *i.e.* computing a probability distribution of the form $P(Searched \mid Known)$. Answering a "question" consists in deciding a value for the variable $Searched$ according to $P(Searched \mid Known)$. Different decision policies are possible, in our robotic experiments we usually choose to draw a value at random according to that distribution. It is well known that general Bayesian inference is a very difficult problem, which may be practically intractable. But, as this paper is mainly concerned with modeling issues, we will assume that the inference problems are solved and implemented in an efficient manner by the programmer.

B. Example

Since the BRP formalism is only based on the inference rules needed for probability calculus, it is very general. Indeed, a very wide class of probabilistic models found in the literature can be rewritten as BRP programs, as is shown in [16]. For example, we can rewrite the Markov Localization model into the BRP formalism. The ML model is basically a Hidden Markov Model with an additional action variable. Recall that a HMM is basically the decomposition $P(O_t\ S_t\ S_{t-1}) = P(S_{t-1})P(S_t \mid S_{t-1})P(O_t \mid S_t)$, where O_t is a perception variable, S_t and S_{t-1} are location variables at time t and $t-1$. Starting from this structure, the action variable A_t is used to refine the transition model $P(S_t \mid S_{t-1})$ into $P(S_t \mid A_t\ S_{t-1})$. The resulting BRP model for Markov Localization is shown Fig. 2.

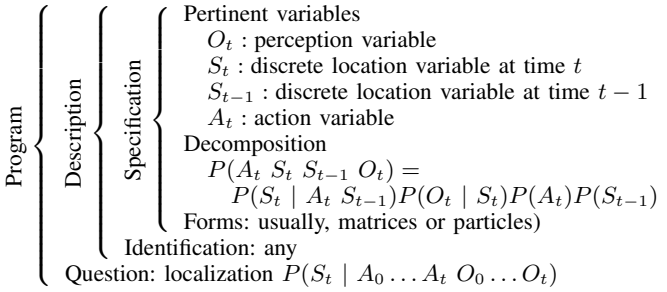


Fig. 2. The Markov Localization definition expressed in the BRP formalism.

C. BRP vs. other models

Let us now develop some remarks that arise from the comparison between the use of the BRP formalism and some aspects of the more common models of the representation of space (see Section I). In particular, we now focus on solving navigation tasks using BRP programs.

The first remark relies on the fact that, in BRP, a form appearing in a description c^1 can be a question to another description c^2. This allows the programmer to decompose a robotic program into sub-programs, as in structured computer programming. Therefore, the first step for solving a navigation task is to imagine, or to copy from living beings (see [12], [13]), intermediary levels of descriptions or skills, that are relevant. This is somewhat different from most probabilistic models of space, that only rely on one level of description, *i.e.* that try to represent the environment using only one type of features. Forms being questions to other descriptions is a key feature of our Abstraction operator (see Section IV).

The second remark is that the first step when designing a BRP description is the choice of variables. When dealing with the representation of space, one usually selects a perception variable, an action variable, and a location (or state) variable. Therefore, the programmer has to choose a set of locations that are *relevant for solving the task at hand, in the class of environments the robot will likely face*. The choice of the nature of these locations (metric or topologic, or dense or sparse, for instance) should come *as a consequence of these considerations*. This, again, somewhat differs from existing approaches, where the choice of model (Markov Localization or Kalman Filter, for instance), is rather a choice of a *dependency structure* or *form definition*, that implies properties on the choice of variables (Kalman Filters are well suited to continuous variables, for instance). In contrast, in the Bayesian Map formalism, we will not put constraints on the choice of decomposition or forms: the programmer will have all latitude left for choosing the semantic of the location variable that

solves his navigation task (the constraints on the choice of variables will merely be syntactic).

The third and final remark is that, in BRP, the description phase is considered independent of the utilization phase. This contrasts with most probabilistic models, where the terms appearing in the decomposition are usually chosen for a particular inference. For example, action or transition models, which can be difficult to assess when the variables are not chosen well, are still very common because they are easily integrated into the location estimation. In our Bayesian Map formalism, we will constraint what maps are used for (the questions), but not how the knowledge necessary for using the map is structured (the decomposition).

III. Bayesian Maps

A Bayesian Map c is a description that defines a joint distribution $P(P\ L_t\ L_{t'}\ A \mid c)$, where:

- P is a perception variable (the robot reads its values from physical sensors or lower level variables),
- L_t is a *location* variable at time t,
- $L_{t'}$ is a variable having the same domain than L_t, but at time t' (without loss of generality, let us assume $t' > t$),
- and A is an action variable (the robot writes commands on this variable).

For simplicity, we will assume here that all these variables have finite domains.

The choice of decomposition is not constrained: any probabilistic dependency structure can therefore be chosen here. Finally, the definition of forms and the learning mechanism (if any) are not constrained, either.

For a Bayesian Map to be useable in practice, we need the description to be rich enough to generate *behaviors*. We call *elementary behavior* any question of the form $P(A^i \mid X)$, where A^i is a subset of A, and X a subset of the other variables of the map (*i.e.*, not in A^i). A behavior can be not elementary, for example if it is a sequence of elementary behaviors, or, in more general terms, if it is based on elementary behaviors and some other knowledge (which need not be expressed in terms of maps).

For a Bayesian Map to be interesting, we will also require that it generates *several* behaviors – otherwise, defining just a single behavior instead of a map is enough. Such a map is therefore a resource, based on a location variable relevant enough to solve a class of tasks: this internal model of the world can be reified.

A "guide" one can use to "make sure" that a given map will generate useful behaviors, is to check if the map answers in a relevant manner the three questions $P(L_t \mid P)$ (localization), $P(L_{t'} \mid A\ L_t)$ (prediction) and $P(A \mid L_t\ L_{t'})$ (control).

By "relevant manner", we mean that these distributions have to be informative, in the sense that their entropy is "far enough" of its maximum (*i. e.* the distribution is different from a uniform distribution). This constraint is not formally well defined, but it seems intuitive to focus on these three questions. Indeed, the skills of localization, prediction and control are well identified in the literature as means to generate behaviors.

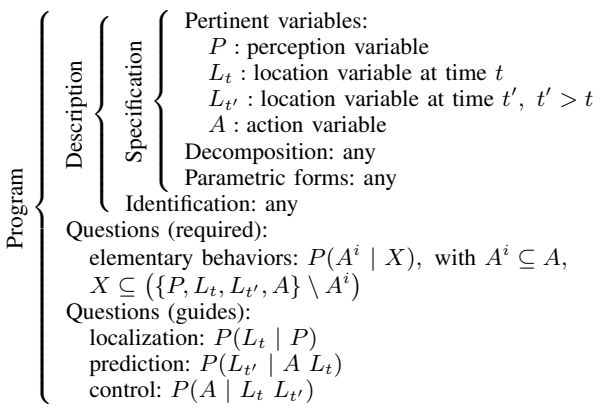


Fig. 3. The Bayesian Map model definition expressed in the BRP formalism.

Checking that the answers to these questions are informative is a first step to evaluate the quality of a Bayesian Map with respect to solving a given task.

Fig. 3 is a summary of the definition of the Bayesian Map formalism.

A. Generality of the Bayesian Map formalism

We now invite the reader to verify that the Markov Localization model is indeed a special case of the Bayesian Map model by comparing Fig. 2 and Fig. 3. Recall that Kalman Filters and Particle Filters are special cases of Markov Localization, as they add hypotheses over the choice of dependency structure made by the Markov Localization model. This implies that Kalman Filters and Particle Filters also are special cases of Bayesian Maps.

Bayesian Maps can therefore accommodate many different forms, depending on the needs or information at hand: for example, one Bayesian Map can be structured like a real valued Kalman Filter for tracking the angle and distance to some feature when it is available. If that feature is not present, or in cases where the linearity hypotheses fail, we can use another Bayesian Map, which need not be a Kalman Filter (for example, based on a symbolic variable).

IV. Abstraction of Bayesian Maps

Having defined the Bayesian Map concept, we now turn to defining operators for putting Bayesian Maps together. The one we present here is called the Abstraction of maps, it is defined Fig. 4, and commented in the rest of this section.

As stressed above, in a Bayesian Map, the semantics of the location variable can be very diverse. The main idea behind the abstraction operator is to build *a Bayesian Map c whose different locations are other Bayesian Maps* $c^1, c^2, \ldots, c^n$. The location variable of the abstract map will therefore take n possible symbolic values, one for each underlying map c^i. Each of these maps will be "nested" in the higher level abstract map, which justifies the use of the term "hierarchy" in our work. Recall that Bayesian Maps are designed for generating behaviors. Let us note $a^1, a^2, \ldots, a^k$ the k behaviors defined

Program

Description

Specification

Pertinent variables:

$P = \bigwedge_{i=1}^{n} \left(P^i \wedge L_t^i \wedge L_{t'}^i \wedge A^i\right)$

$L_t : \mathcal{D}_{L_t} = \{c^1, c^2, \ldots, c^n\}, k_{L_t} = n$

$L_{t'} : \mathcal{D}_{L_{t'}} = \{c^1, c^2, \ldots, c^n\}, k_{L_{t'}} = n$

$A : \mathcal{D}_A = \{a_1, a_2, \ldots, a_k\}, k_A = k$

Decomposition:

$P(P\ L_t\ L_{t'}\ A) =$

$P(L_t) \prod_{i=1}^{n} P(P^i\ L_t^i\ L_{t'}^i\ A^i \mid L_t)$

$P(L_{t'}) P(A \mid L_t\ L_{t'})$

Parametric forms:

$P(L_t) =$ Uniform

$P(P^i\ L_t^i\ L_{t'}^i\ A^i \mid [L_t = c]) = \begin{cases} \text{if } c = c^i \text{ then } P(P^i\ L_t^i\ L_{t'}^i\ A^i \mid c^i) \\ \text{else Uniform} \end{cases}$

$P(L_{t'}) =$ Uniform

$P(A \mid L_t\ L_{t'}) =$ Table

Identification:

a priori programming or learning of $P(A \mid L_t\ L_{t'})$

Questions:

$$\begin{aligned} P(L_t \mid P) &= \tfrac{1}{Z_1} \prod_{i=1}^{n} P(P^i\ L_t^i\ L_{t'}^i\ A^i \mid L_t) \\ P(L_{t'} \mid A\ L_t) &= \tfrac{1}{Z_2} P(A \mid L_t\ L_{t'}) \\ P(A \mid L_t\ L_{t'}) &= P(A \mid L_t\ L_{t'}). \end{aligned}$$

Fig. 4. The abstraction operator definition expressed as a Bayesian Map.

in the n underlying maps. In the abstract map, these behaviors can be used for linking the locations c^i. The action variable of the abstract map will therefore take k possible symbolic values, one for each behavior of the underlying maps. In order to build an abstract map having n locations, the programmer will have to have previously defined n lower level maps, which generate k behaviors. The numbers n and k are therefore small, and so the abstract map deals with a small internal space, having retained of each underlying map only a symbol, and having "forgotten" all their details. This justifies the use of the name "abstraction" for this operator. But this "summary mechanism" has yet to be described: that is what the perception variable P of the abstract map will be used for, as it will be the list of all the variables appearing in the underlying maps: $P = P^1, L_t^1, L_{t'}^1, A^1, \ldots, P^n, L_t^n, L_{t'}^n, A^n$.

Given the four variables of the abstract map, we define its joint distribution with the following decomposition:

$$\begin{aligned} &P(P\ L_t\ L_{t'}\ A) \\ &= P(L_t) \prod_{i=1}^{n} P(P^i\ L_t^i\ L_{t'}^i\ A^i \mid L_t) P(L_{t'}) P(A \mid L_t\ L_{t'}). \end{aligned}$$

In this decomposition, $P(L_t)$ and $P(L_{t'})$ are defined as uniform distributions. All the terms of the form $P(P^i\ L_t^i\ L_{t'}^i\ A^i \mid [L_t = c])$ are defined as follows: when $c \neq c^i$, the probabilistic dependency between the variables P^i, L_t^i, $L_{t'}^i$, A^i of the map c^i is supposed unknown, therefore defined by a uniform distribution. Whereas when $c = c^i$, this dependency is exactly what the map c^i defines. Therefore this term is a question to the description c^i, but a question that includes the whole sub-description by asking for the joint distribution it defines. Since the last term, $P(A \mid L_t\ L_{t'})$, only includes symbolic variables that have a small number of values, it makes sense to define it as a table, which can be easily *a priori* programmed or learned experimentally.

The abstract Bayesian Map is now fully defined, and, given n underlying maps, can be automatically built. The last step is to verify that it generates useful behaviors. We will examine the guide questions of localization, prediction and control.

The localization question leads to the following inference (derivation omitted): $P(L_t \mid P) \propto \prod_{i=1}^{n} P(P^i\ L_t^i\ L_{t'}^i\ A^i \mid L_t)$. The interpretation of this result will be explained with an example, Section V. The derivations for solving the prediction $P(L_{t'} \mid A\ L_t)$ and control $P(A \mid L_t\ L_{t'})$ questions are also straightforward, and given Fig. 4.

Recall that the final goal of any Bayesian Map is to provide behaviors. In the abstract map, this is done by answering a question like $P(A \mid [L_{t'} = c^i]\ [P = p])$: what is the probability distribution over lower level behaviors, knowing all values p of the variables of the lower level, and knowing that we want to "go to map c^i?" Answering this question thus allows selecting the most relevant underlying behavior to reach a given high level goal. The computation is as follows:

$$\begin{aligned} &P(A \mid L_{t'}\ P) \\ &= \frac{1}{Z} \sum_{L_t} \left(\prod_{i=1}^{n} P(P^i\ L_t^i\ L_{t'}^i\ A^i \mid L_t) \right) P(A \mid L_t\ L_{t'}). \end{aligned}$$

This computation includes the localization question, to weigh the probabilities given by the control model $P(A \mid L_t\ L_{t'})$. In other words, the distribution over the action variable A includes all localization uncertainties. Each underlying model is used, even when the robot is located at a physical location that this model is not made for. As a direct consequence, there is no need to *decide* what map the robot is in, or to *switch* from map to map: the computation considers all possibilities and weighs them according to their (localization) probabilities. Therefore the underlying maps need not be "mutually exclusive" in a geographical sense.

V. Experimental validation

We report here an experiment made on the well-known Koala mobile robot platform (K-team company). In order to keep as much control as possible over our experiments and the different effects we observe, we simplify the sensorimotor system and its environment. We only use the 16 proximeters $Px = Px_0 \wedge \ldots \wedge Px_{15}$ of our robot, and keep two degrees of freedom of motor control, via the rotation and translation speed $Vrot$ and $Vtrans$. The environment we use is a 5 m $\times$ 5 m area made of movable planks (see a typical configuration we use Fig. 5). The goal of this experiment is to solve a navigation task: we want the robot to be able to go hide in any corner, as if the empty space in the middle of the area were dangerous.

The first programming step is to analyze this task into sub-tasks. We particularize three situations that are relevant for solving the task: the robot can either be near a wall, and it should follow it in order to reach the nearest corner, or the robot can be in a corner, and it should stop, or finally it could

be in empty space, and should therefore go straight, so as to leave the exposed area as quickly as possible.

A. Low level Bayesian Maps

Given this analysis, the second programming step is to define one Bayesian Map for each of the three situations. They all use the same perception variable $P = Px$ and the same action variable $A = Vrot \wedge Vtrans$.

The first map, c^{wall}, describes how to navigate in presence of a single wall, using a location variable $L_t = \theta \wedge Dist$: the phenomenon "wall" is summed up by an angle θ and a distance $Dist$. Therefore, c^{wall} defines $P(Px\ \theta_t\ Dist_t\ \theta_{t'}\ Dist_{t'}\ Vrot\ Vtrans \mid c^{wall})$. We have implemented this map using 12 possible angle values, and 3 different distances. This lead to a compact model, yet accurate enough to solve the sub-tasks we wanted to solve. The dependency structure we choose is (c^{wall} on right hand sides omitted):

$$\begin{aligned} & P(Px\ \theta_t\ Dist_t\ \theta_{t'}\ Dist_{t'}\ Vrot\ Vtrans) \\ = \quad & P(\theta_t\ Dist_t) \prod_i P(Px_i \mid \theta_t\ Dist_t) P(\theta_{t'}\ Dist_{t'}) \\ & P(Vrot \mid \theta_t\ Dist_t\ \theta_{t'}\ Dist_{t'}) \\ & P(Vtrans \mid \theta_t\ Dist_t\ \theta_{t'}\ Dist_{t'}). \end{aligned}$$

$P(\theta_t\ Dist_t)$ and $P(\theta_{t'}\ Dist_{t'})$ are uniform probability distributions. Each term of the form $P(Px_i \mid \theta_t\ Dist_t)$ is a set of Gaussians, that were identified experimentally, by a supervised learning phase: we physically put the robot in all 36 possible situations, and recorded proximeter values so as to compute experimental means and standard deviations. Finally, the two control terms $P(Vrot \mid \theta_t\ Dist_t\ \theta_{t'}\ Dist_{t'})$ and $P(Vtrans \mid \theta_t\ Dist_t\ \theta_{t'}\ Dist_{t'})$ were programmed "by hand": given the current angle and distance, and the angle and distance to be reached, what should be the motor commands?

This map successfully solves navigation tasks like "follow-wall-right", "follow-wall-left", "go-away-from-wall", "stop", using behaviors of the same name. For example, "follow-wall-right" is defined by the probabilistic question $P(Vrot\ Vtrans \mid Px\ [L_{t'} = \langle 90, 1 \rangle])$: compute the distribution on motor variables knowing the sensory input and knowing that the location to reach is $\theta = 90\ ^\circ$, $Dist = 1$ (wall on the right at medium distance).

This map is an instance where a Kalman Filter based Bayesian Map could have been used instead: for example, if we had required more accuracy on the angle and distance to the wall, using continuous variables. The coarse grained set of values we used were actually sufficient for our experiments.

The two other Bayesian Maps we define are the following. 1) c^{corner} describes how to navigate in a corner, using a symbolic location variable that can take 4 values: $FrontLeft$, $FrontRight$, $RearLeft$ and $RearRight$. This is enough for solving tasks like "quit-corner-and-follow-right", "away-from-both-walls", "stop". 2) $c^{empty-space}$describes how to navigate in empty space, *i.e.* when the sensors do not see anything. The behaviors defined here are "straight-ahead" and "stop".

B. Abstract Bayesian Map

Given these three maps, the third and final programming step is to apply the abstraction operator on them. We obtain a map c, whose location variable is $L_t = \{c^{wall}, c^{corner}, c^{empty-space}\}$. The action variable lists the behaviors defined by the low level maps: $A = \{\text{follow-wall-right}, \text{go-away-from-wall}, \ldots\}$. The rest of the abstract map is according to the schema of Fig. 4.

We want here to discuss the localization question. Let us assume that the robot is in empty space: all its sensors read 0. Let us also assume that the robot is currently applying the "straight-ahead" behavior, that sets $Vrot$ and $Vtrans$ near 0 (no rotation) and 40 (fast forward movement), respectively, using sharp Gaussian distributions.

Let us consider the probability to be in location $c^{empty-space}$ (with w standing for $wall$, c for $corner$ and e for $empty - space$):

$$\begin{aligned} & P([L_t = c^{empty-space}] \mid P) \\ & \propto \begin{pmatrix} P(P^w\ L_t^w\ L_{t'}^w\ A^w \mid [L_t = c^{empty-space}]) \\ P(P^c\ L_t^c\ L_{t'}^c\ A^c \mid [L_t = c^{empty-space}]) \\ P(P^e\ L_t^e\ L_{t'}^e\ A^e \mid [L_t = c^{empty-space}]) \end{pmatrix}. \end{aligned}$$

Of the three terms of the product, two are uniforms, and one is the joint distribution given by $c^{empty-space}$. That joint distribution gives a very high probability for the current situation, as describing the phenomenon "going straight ahead in empty space" basically amounts to favoring sensory readings of 0 and motor commands near 0 and 40 for $Vrot$ and $Vtrans$, respectively. The situation is quite the opposite for $P([L_t = c^{wall}] \mid P)$: for example, c^{wall} does not favor at all this sensory situation. Indeed, the phenomenon "I am near a wall" is closely related to the fact that the sensors actually sense something. The probability of seeing nothing on the sensors knowing that the robot is near a wall is very low: $P([L_t = c^{wall}] \mid P)$ will be very low. The reasoning is similar for $P([L_t = c^{corner}] \mid P)$.

This computation can thus be interpreted as the *recognition of the most pertinent underlying map* for a given sensorimotor situation. Alternatively, it can be seen as a *measure of the coherence of the values of the variables* of each underlying map, or even as a *Bayesian comparison of the relevance of models*, as assessed by the numerical value of the joint distributions of each lower level model. Since these distributions include (lower level) location and action variables, the maps are not only recognized by sensory patterns, but also by what the robot is currently doing.

The localization question can therefore be used to assess the "validity zones" of the underlying maps, *i.e.* the places of the environment where the hypotheses of each model hold. Experimentally, we have the robot navigate in the environment, and ask at each time step the localization question. We can summarize visually the answer, for example by drawing values for L_t, and report the drawn value on a Cartesian map of the environment. A (simplified but readable) result is shown Fig. 5. As can be seen, the robot correctly recognizes each

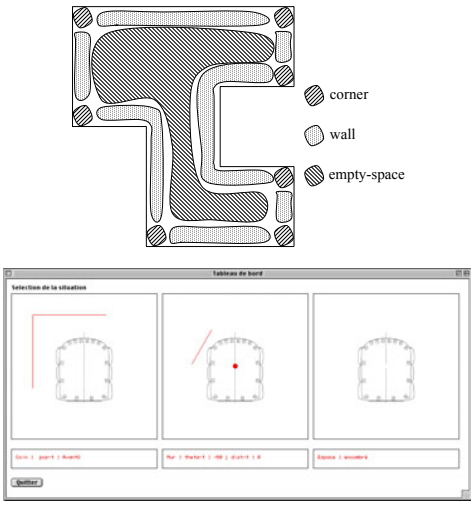


Fig. 5. 2D projection of the estimated "validity zones" of the maps c^{wall}, c^{corner} et $c^{empty-space}$. The bottom part of the figure is a screenshot of the localization module of the abstract map: it shows the "comparison" and competition between the underlying models. The winner is marked by the central dot: in this case, the robot was near a wall.

situation that it has a model for. Let us note that the resulting zones are not contiguous in the environment: for example, all the corners of the environment are associated with the same symbol, namely, c^{corner}. This effect is known as *perceptual aliasing*. But this very simple representation is sufficient for solving the task that was given to the robot: we report here that the behavior "go-hide-in-any-corner" is indeed generated by the abstract map.

A typical trajectory for the robot, starting from the middle of the arena, is to start by going straight ahead. As soon as a couple of forward sensors sense something, the "empty-space" situation is not relevant anymore, and the robot applies the best model it has, depending on the correlation between what the sensors see: if it looks like a wall and moves like a wall, then the probability for the "wall" model is high; on the other hand, if it rather feels like a corner, then the corner model wins the probabilistic competition. Suppose it was near a wall, then it starts to follow it, until a corner is reached. In our first version, the corner model was designed "too independently" of the wall model: the validity zone of the c^{corner} map was too small, and seldom visited by the robot as it passed the corner using the "follow-wall-right" behavior, defined by c^{wall}. The robot would then miss the first corner, and stop at another one. This shows that the decomposition of the task gives independent sub-tasks only as a first approximation. We solved the problem by modifying the "corner" model, so that it would recognize a corner on a typical "follow-wall-right" trajectory.

VI. Conclusion

We have presented the Bayesian Map formalism: it is a generalization of most probabilistic models of space found in the literature. Indeed, it drops the usual constraints on the choice of decomposition, forms, or implementation of the probability distributions. We have also presented the Abstraction operator, for building hierarchies of Bayesian Maps.

The experiments we presented are of course to be regarded only as "proofs of concept". Their simplicity also served didactic purposes. However, these experiments, in our view, are a successful preliminary step toward applying our formalism. Part of the current work is of course aimed at enriching these experiments, in particular with respect to the *scaling up* capacity of the formalism.

Moreover, since each map of the hierarchy is a full probabilistic model it is potentially very rich. Possible computations based on these maps include questions like the prediction question $P(L_{t'} \mid A\ L_t)$, which can form the basis of *planning* processes. Hierarchies of Bayesian Maps are therefore to be placed alongside model based approaches, instead of pure reactive approaches. Exploiting such knowledge by integrating a planning process in our Bayesian Map formalism is also part of the ongoing work.

References

[1] J. Leonard, H. Durrant-Whyte, and I. Cox, "Dynamic map-building for an autonomous mobile robot," *The International Journal of Robotics Research*, vol. 11, no. 4, pp. 286–298, 1992.

[2] S. Thrun, "Probabilistic algorithms in robotics," *AI Magazine*, vol. 21, no. 4, pp. 93–109, 2000.

[3] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[4] L. R. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*. Englewood Cliffs, New Jersey: Prentice Hall, 1993, ch. Theory and implementation of Hidden Markov Models, pp. 321–389.

[5] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier, "Hierarchical solution of Markov decision processes using macro-actions," in *Proceedings of the 14th Conf. on Uncertainty in Artificial Intelligence (UAI-98)*, G. F. Cooper and S. Moral, Eds. San Francisco: Morgan Kaufmann, July, 24–26 1998, pp. 220–229.

[6] T. Lane and L. P. Kaelbling, "Toward hierarchical decomposition for planning in uncertain environments," in *Proceedings of the 2001 IJCAI Workshop on Planning under Uncertainty and Incomplete Information*. Seattle, WA: AAAI Press, August 2001.

[7] J. Pineau and S. Thrun, "An integrated approach to hierarchy and abstraction for POMDPs," Carnegie Mellon University, Technical Report CMU-RI-TR-02-21, August 2002.

[8] S. Thrun, "Robotic mapping: A survey," Carnegie Mellon University, Technical Report CMU-CS-02-111, February 2002.

[9] K. Murphy, "Dynamic bayesian networks: Representation, inference and learning," Ph.D. thesis, University of California, Berkeley, Berkeley, CA, July 2002.

[10] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[11] T. Lane and L. P. Kaelbling, "Nearly deterministic abstractions of markov decision processes," in *18th Nat. Conf. on Artificial Intelligence*, 2002.

[12] B. J. Kuipers, "The spatial semantic hierarchy," *Artificial Intelligence*, vol. 119, no. 1–2, pp. 191–233, 2000.

[13] O. Trullier, S. Wiener, A. Berthoz, and J.-A. Meyer, "Biologically-based artificial navigation systems: Review and prospects," *Progress in Neurobiology*, vol. 51, pp. 483–544, 1997.

[14] J. Diard, "La carte bayésienne – un modèle probabiliste hiérarchique pour la navigation en robotique mobile," Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, Janvier 2003.

[15] O. Lebeltel, P. Bessière, J. Diard, and E. Mazer, "Bayesian robot programming," *Autonomous Robots (in press)*, vol. 16, no. 1, 2004.

[16] J. Diard, P. Bessière, and E. Mazer, "A survey of probabilistic models, using the bayesian programming methodology as a unifying framework," in *The Second Int. Conf. on Computational Intelligence, Robotics and Autonomous Systems (CIRAS)*, Singapore, December 2003.

International Journal of Robotic Research

Bayesian Occupancy Filtering for Multi-Target Tracking : an Automotive Application

C. Coué, C. Pradalier, C. Laugier, Th. Fraichard and P. Bessière

Inria[a] Rhone-Alpes & Gravir[b]-CNRS[c]
655 av. de l'Europe, Montbonnot, 38334 St Ismier Cedex, France
Tel. +33 476 61 54 36 Fax. +33 476 61 52 10
christophe.coue@inria.fr
http://www.inrialpes.fr/sharp

January 7, 2004

Submitted Version

Keywords — Bayesian reasonning, multitarget tracking, occupancy grids.

Acknowledgements — This work was partially supported by the French programme *"La Route Automatisée"* (http://www.lara.prd.fr/) and the European project IST-1999-12224 *"Sensing of Car Environment at Low Speed Driving"* (http://www.carsense.org).

[a]Institut National de Recherche en Informatique et en Automatique.
[b]Lab. Graphisme, Vision et Robotique.
[c]Centre National de la Recherche Scientifique.

Bayesian Occupancy Filtering for Multi-Target Tracking: an Automotive Application

C. Coué, C. Pradalier, C. Laugier, T. Fraichard and P. Bessière
Inria Rhône-Alpes & Gravir -CNRS
http://www.inrialpes.fr/sharp

Abstract— **Reliable and efficient perception and reasoning in dynamic and densely cluttered environments is still a major challenge for driver support systems. Most of today system use target tracking algorithms based on rigid model. They work quite reliably in simple environments with few potential obstacles like on freeways. However, these approaches usually fail in more complex environments including a large number of dynamical potential obstacles, as it is usually the case in city driving conditions. In this paper, we propose a new approach for robust perception and danger assesment of highly dynamic environments. This approach is called *Bayesian Occupancy Filtering*. It basically combines a 4-dimensionnal occupancy grid representation of the obstacle state-space with Bayesian filtering techniques.**

I. Introduction and Problem Statement

A. The ADAS context

Unlike regular cruise control systems, Adaptive Cruise Control (ACC) systems use a range sensor to regulate the speed of the car while ensuring collision avoidance with the vehicle in front. ACC systems were introduced on the automotive market in 1999. Since then, surveys and experimental assessments have demonstrated the interest for this kind of systems. They are the first step towards the design of future Advanced Driver Assistance Systems (ADAS) that should help the driver in increasingly complex driving tasks. The use of today commercially available ACC systems is pretty much limited to motorways or urban expressways without crossings. The traffic situations encountered are rather simple and attention can be focused on a few, well defined detected objects (cars and trucks). Nonetheless, even in these relatively simple situations, these systems show a number of limitations: they are not very good at handling fixed obstacles and may generate false alarms; moreover, in some 'cut-in' situations, *i.e.* when the intrusion of an other vehicle or a pedestrian in the detection beam is too close to the vehicle, they may be unable to react appropriately.

A wider use of such systems requires to extend their range of operation to some more complex situations in dense traffic environments, around or inside urban areas. In such areas, traffic is characterized by lower speeds, tight curves, traffic signs, crossings and fragile traffic participants such as motorbikes, bicycles or pedestrians.

B. The related multi-target tracking problem

A prerequisite to a reliable ADAS in such complex traffic situations is an estimation of dynamic characteristics of the traffic participants, such as position and velocity. Most of today system use various *target tracking* algorithms [1]. The objective is to collect *observations*, *i.e.* data from the sensor, on one or more *potential obstacles* in the environment of the vehicle, and then to estimate at each time step and as robustly as possible the obstacles position and velocity. Classical approach is to track the different objects independently, by maintaining a list of *tracks*, *i.e.* a list of currently known objects. The main difficulty of multi-target tracking is known as the *Data Association* problem. It includes observation-to-track association and track management problems. The goal of observation-to-track association is to decide whether a new sensor observation corresponds to an existing track or not. Then the goal of track maintenance is to decide the confirmation or the deletion of each existing track, and, if required, the creation of new tracks. Numerous methods exist to perform this data association problem [2], [3], [4]. A complete review of the tracking methods with one or more sensors can be found in [5].

Urban traffic scenarios are still a challenge in multi-target tracking area: the traditional data association problem is intractable in situations involving numerous appearances, disappearances and occlusions of a large number of rapidly maneuvering targets.

C. Outline of our approach

The objective of this paper is to propose a new approach for a robust perception and analysis of highly dynamic environments. This approach has been designed in order to avoid the data association problem previously mentioned. It is based on a probabilistic *grid representation of the obstacles state space*. As we consider the position and the velocity of the potential obstacles with respect

to our vehicle, this grid is 4-dimensional and is called the *obstacle state space (OSS)* grid . Then for each cell of the grid, different properties (occupancy, danger ...) are estimated using sensor observations and some prior knowledge.

Bayesian Programming: uncertainty is inherently present in any model of a real phenomenon. A robust estimation of these properties requires to take into account this uncertainty. This is done using *probabilistic reasoning*, which has recently become a key paradigm in robotics. Probabilistic approaches have already been used to address various robotic problems, such as CAD modeling, map building and localization [6], [7], [8]. The probabilistic framework we are using is based on an implementation of the Bayesian theory [9]. This framework, referred as *Bayesian Programming*, initially developed to design robust robot control programs [10], [11], [12]. However, we have shown that its scope of application is much broader, and that it can be used whenever one has to deal with problems involving uncertain or incomplete knowledge [13], [14]. The Bayesian Programming paradigm is introduced in section II.

Obstacle State Space grid: two main properties have to be estimated for the grid cells: occupancy and danger. Concerning the occupancy probability, the objective is to compute from the sensor observations the probability that each cell is full or empty. To avoid a combinatorial explosion of grid con guration, the cell states are estimated as *independent* random variables. This estimation detailed in section III is in fact an original implementation of the *occupancy grids* framework [15], [16], which has already been extensively used for static indoor mapping [17] using a 2-dimensional grid. More recently, occupancy grids have been adapted to track multiple moving objects [18]. In this approach, spatio-temporal clustering applied to *temporal maps* is used to perform motion detection and tracking. A major drawback, relatively to the ADAS context, of this work is that a moving object may be lost due to occlusion effects.

Fig. 1. The CyCab, equipped with a Sick laser range finder

Bayesian Occupancy Filter: to take into account the dynamic environment, and to be as robust as possible relatively to objects occlusions, it is necessary to take into account the *sensor observations history and the temporal consistency of the scene*. This is done by introducing a two-step mechanism in the occupancy grid estimation. This mechanism includes a prediction (history) and an estimation (new measurements) steps. This mechanism is derived from the *Bayes filters* approach [19] and it is called the *Bayesian Occupancy Filter* (BOF). It is detailed in section IV.

Danger assessment: The most relevant property for an ADAS system is the danger associated to a cell and to the related controls of the vehicle. Our approach for solving this problem is to combine the previous occupancy property with a danger criteria depending on the relative localization of the considered cells with the controlled vehicle, and on the control characteristics of the vehicle. This approach has been implemented as a collision avoidance system on a CyCab (see g 1), an electric car[1] equipped with a Sick laser range nder, allowing the system to estimate targets position and velocity. The collision avoidance system is described in section V.

II. Bayesian Programming

Any model of a real phenomenon is inherently incomplete. There are always some hidden variables, not taken into account in the model that in uence the phenomenon. The effect of these hidden variables is that the model and the phenomenon never behave exactly the same way. Furthermore, perception and control are inherently uncertain. Uncertainty arises from sensor limitation or noise. Rational reasoning with incomplete and uncertain information is quite a challenge. Bayesian Programming addresses this challenge relying upon a well established formal theory: the probability theory [9].

The usual notion of *logical proposition* (either true or false) is the rst key concept of probabilistic reasoning. Logical operators can be used to derive new propositions (conjunction, disjunction, negation). *Discrete variable* is the second concept that is needed: it is a set of logical proposition that are exhaustive and mutually exclusive (at least one is true, only one is true). Discrete variables can be combined too (conjunction). To deal with uncertainty, *probabilities* are attached to propositions, and to manipulate probabilities, usual inference rules are used:

- Conjunction rule:

$$P(X\ Y) = P(X)P(Y \mid X)$$

[1] `http://www.robosoft.fr`

- Normalization rule:

$$\sum_X P(X) = 1$$

with X and Y discrete variables and P a probability.

$$Program \begin{cases} Description & \begin{cases} Relevant\ Variables \\ Decomposition \\ Parametric\ Forms \end{cases} \\ Question \end{cases}$$

Fig. 2. structure of a Bayesian Program.

In this framework, a Bayesian Program is made up of two parts: a *description* and a *question*.

The description can be viewed as a knowledge base containing the a priori information available on the problem at hand. It is essentially a joint probability distribution. The description is made up of three components:

1) A set of *relevant variables* on which the joint distribution is de ned. Typically, variables are motor, sensory or internal.
2) A *decomposition* of the joint distribution as a product of simpler terms. It is obtained by applying Bayesian rules and taking advantages of the conditional independencies that may exists between variables.
3) The *parametric forms* assigned to each of the terms appearing in the decomposition (they are required to compute the joint distribution).

Given a distribution, it is possible to ask *questions*. Questions are obtained rst by partitioning the set of variables into three sets:

1) *S*: the searched variables.
2) *K*: the known variables.
3) *F*: the free variables.

A question is then de ned as the distribution:

$$P(S \mid K) \tag{1}$$

Given the description, it is always possible to answer a question, *i.e.* to compute the probability distribution $P(S \mid K)$. To do so, the following general inference is used:

$$\begin{aligned} P(S \mid K) &= \frac{\sum_F P(S\ F\ K)}{P(K)} \\ &= \frac{1}{\alpha} \times \sum\nolimits_F P(S\ F\ K), \end{aligned} \tag{2}$$

where α is a normalization term.

As such, the inference is computationally expensive (Bayesian inference in general has been shown to be NP-Hard [20]). A symbolic simpli cation phase can reduce drastically the number of sums necessary to compute a given distribution. However the decomposition of the preliminary knowledge, which express the conditional independencies of variables, still plays a crucial role in keeping the computation tractable.

Our group developped an API[2], called ProBT®, in order to express Bayesian programs, and the corresponding inference engine to automate computation. This engine operates in two stages:

- a symbolic simpli cation stage which allows to reduce the complexity of the probability distribution to be computed.
- a numeric stage that actually computes the distribution.

This engine is now commercialized by the Probayes[3] company.

III. Static estimation of the occupancy probability

A. Bayesian Program

Our goal is to estimate the occupancy probability of each cell of the grid, using the last sensor observations. The variables that are relevant here are:

- C : the cell itself. This variable is 4-dimensional and represents a position and a speed relative to the vehicle;
- E_C : the state of the cell C, occupied or not;
- $\mathcal{Z}$: the sensor observation set. One observation is denoted Z_s. The number of observation is denoted S;
- M : the matching variable. Its goal is to specify which observation of the sensor is currently used to estimate the state of a cell.

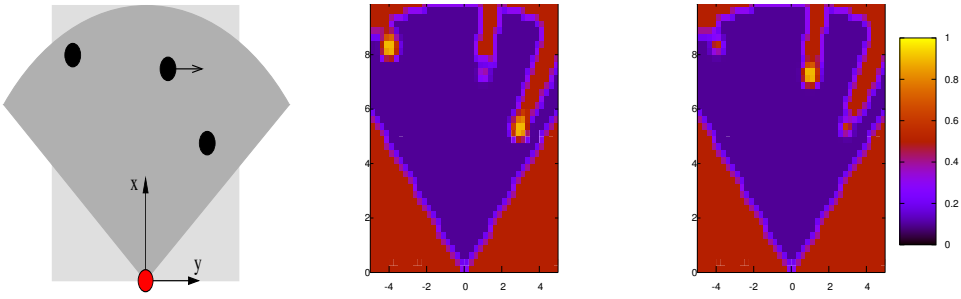
The following decomposition of the joint distribution determined by these variables is chosen:

$$P(C\ E_C\ \mathcal{Z}\ M) = \left(\begin{array}{c} P(C)P(E_C \mid C)P(M) \\ \times \prod_{s=1}^{S} P(Z_s \mid C\ E_C\ M) \end{array} \right). \tag{3}$$

Parametric forms have to be assigned to each of the terms of this decomposition:

- $P(C)$ represents the information on the cell itself. As we always know the cell for which we are currently estimating the state, this distribution does not need to be speci ed;
- $P(E_C \mid C)$ represents the *a priori* information on the occupancy of the cell. If available, a *prior* distribution could be used to specify it. Otherwise, a uniform

[2]Application Programming Interface
[3]`http://www.probayes.com`



a)
$z_1 = (8.3, -4, 0, 0)$
$z_2 = (5, 3, 0, 0)$
$z_3 = (7.3, 1.9, 0, 0.8)$

b)
$P([E_C = 1] \,|\, \mathcal{Z}\, c)$
$[c = (x, y, 0, 0)]$

c)
$P([E_C = 1] \,|\, \mathcal{Z}\, c)$
$[c = (x, y, 0, 0.8)]$

Fig. 3. Example of static grid estimation. a) the situation and the sensor observations, given in $(x, y, \dot{x}, \dot{y})$; b) and c) values of occupancy probability for two "slices" of the 4-dimensionnal grid, *i.e.* for all possible positions at a given speed.

distribution has to be selected. Next section will explain how the prior distribution may be obtained from passed estimation;
- $P(M)$ is chosen uniform;
- the form of $P(Z_s \,|\, C\, E_C\, M)$ depends on the value the matching variable:
  - if $M \neq s$, the observation is not due to the cell C. Consequently, we cannot say anything on this observation. $P(Z_s \,|\, C\, E_C\, M)$ is de ned by a uniform distribution;
  - if $M = s$, the form of $P(Z_s \,|\, C\, E_C\, M)$ is given by the *sensor model*. Its goal is to model the sensor response knowing the cell state. Details on this model can be found in [16].

This ends the description. We can now ask questions. Since we want an estimate of a cell occupancy, the question to ask to the inference engine is the following:

$$P(E_C \mid \mathcal{Z}\ C). \tag{4}$$

Following the general inference mechanism given by (2), the result of the inference can be written as follows:

$$P(E_C \mid \mathcal{Z}\ C) \propto \sum_{M=1}^{S} \left(\prod_{s=1}^{S} P(Z_s \mid E_C\ C\ M) \right). \tag{5}$$

During the inference, the sum on this variable allows to consider every sensor observation to update the state of one cell. Please note that the estimation step is performed without explicit association between cells and observations, replaced by the integration on all the possible values of M.

B. Experimental result

To test the estimation of occupancy grids presented in the previous section, both a simulator and the real CyCab vehicle were used.

Fig 3 shows rst results of grid estimation. The left picture depicts the situation: the CyCab is modeled as the red circle; the part of the environment covered by the grid is represented by the light gray rectangle; the sensor eld of view is modeled by the dark gray area. In this situation, three obstacles (black circles) are present in front of the cycab. Two obstacles are static, the third one is moving from the left to the right, at a speed of $0.8\ m/s$ represented by a black arrow. The CyCab is not moving in this rst experiment.

As mentioned earlier, we use a 4-dimensional grid. Thus only 2-dimensional slices of the grid are depicted by Fig. 3 b) and c). Fig. 3 b) depicts the occupancy probability of each cell corresponding to a null relative velocity (*i.e.* $c = [x, y, 0, 0]$), which is the velocity of two sensor observation. As expected, two areas with high occupancy probabilities are visible. These probability values depends on probability of detection, probability of false alarm, and on sensor precision. All these characteristics of the sensor are taken into account in the sensor model. As the measured speed of the third obstacles is far from null speed, any area of high occupancy probability corresponds to this observation in this slice of the grid. The cells hidden by one of the three sensor observations or located outside the sensor eld of view have not been observed. Thus we can not conclude about their occupancy. That explains the areas of probability values equal to 0.5 (red areas). Finally, for cells located far from any sensor observation, the occupancy probability is low (purple areas).

Fig. 3 c) depicts the occupancy probability of each cell corresponding to the velocity of the third observation, *i.e.* $c = [x, y, 0, 0.8]$. Consequently we nd an area of high occupancy probabilities corresponding to this observation.

Areas of probability values equal to 0.5 corresponding to unobserved cells (*i.e.* hidden or located outside the sensor eld of view) are still present on this slice of the grid.

IV. Bayesian Occupancy Filter

Bayes lters [19] address the general problem of estimating the state sequence x^k, $k \in \mathbb{N}$ of a system given by:

$$x^k = f^k(x^{k-1}, u^{k-1}, w^k), \tag{6}$$

where f^k is a possibly nonlinear transition function, u^{k-1} is a control variable (*e.g.* speed or acceleration) for the sensor which allows to estimate its ego-movement between time $k-1$ and time k, and w^k is the process noise. This equation describes a Markov process of order one.

Let z^k be the sensor observation of the system at imte k. The objective of the ltering is to recursively estimate x^k from the sensor measurements:

$$z^k = h^k(x^k, v^k). \tag{7}$$

where h^k is a possibly nonlinear function and v^k is the measurement noise. This function models the uncertainty of the measurement z^k of the system's state x^k.

In other words, the goal of the ltering is to recursively estimate the probability distribution $P(X^k \mid Z^k)$, known as the *posterior* distribution. In general, this estimation is done in two stages: *prediction* and *estimation*. The goal of the prediction stage is to compute an *a priori* estimate of the target's state known as the *prior* distribution. The goal of the estimation stage is to compute the *posterior* distribution, using this *a priori* estimate and the current measurement of the sensor.

Exact solutions to this recursive propagation of the posterior density do exist in a restrictive set of cases. In particular, the Kalman lter [21][22] is an optimal solution when the functions f^k and h^k are linear and the noises w^k and v^k are Gaussian. But in general, solutions cannot be determined analytically, and an approximate solution has to be computed.

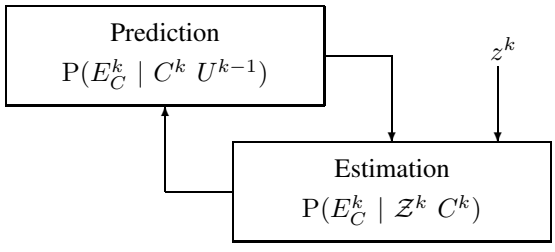


Fig. 4. Bayesian Occupancy Filter as a recursive loop.

In our case, the state of the system is given by the occupancy state of a cell of the grid. For that purpose, we hade developped the new concept of *Bayesian Occupancy Filter*. This consists in a two-steps estimation of the occupancy state, as depicted in g 4.

A. Estimation step

In this loop, the estimation step is similar to the static estimation of the grid depicted in the previous section, except that the *a priori* on the cell occupancy $P(E_C^k \mid C^k)$ is not given by an uniform distribution, but by the result of the previous prediction step.

B. Prediction step

The goal of this processing step is to estimate an *a priori* model of the occupancy probability at time k of a cell using the latest estimation of the occupancy grid, *i.e.* the estimation at time $k-1$. The variables that are relevant here are:

- C^k : the cell C considered at time k;
- E_C^k : the state of this cell C, at time k;
- C^{k-1} : the cell C at time $k-1$;
- E_C^{k-1} : the state of this cell at time $k-1$;
- U^{k-1} : the control input of the cycab at time $k-1$. For example, it could be a measurement of its instantaneous velocity at time $k-1$.

The following decomposition of the joint distribution determined by these variables can be obtained:

$$P(C_k\, E_C^k\, C^{k-1}\, E_C^{k-1}\, U^{k-1}) = \begin{pmatrix} P(U^{k-1})P(C^{k-1}) \\ \times P(E_C^{k-1} \mid C^{k-1}) \\ \times P(C^k \mid C^{k-1}\, U^{k-1}) \\ \times P(E_C^k \mid E_C^{k-1}\, C^{k-1}\, C^k) \end{pmatrix} \tag{8}$$

Parametric forms must be assigned to each of the terms appearing in the decomposition:

- $P(C^{k-1})$ and $P(U^{k-1})$ are chosen as uniform distributions;
- $P(E_C^{k-1} \mid C^{k-1})$ is given by the result of the estimation step at time $k-1$;
- $P(C^k \mid C^{k-1}\, U^{k-1})$ is given by the dynamic model. It represents the probability that an object has moved from the cell C^{k-1} to the cell C^k. This movement is due to the object himself and to the cycab movement between $k-1$ and k;
- $P(E_C^k \mid E_C^{k-1}\, C^{k-1}\, C^k)$ represents the probability that an existing object at time $k-1$ (*i.e.* $[E_C^{k-1} = 1]$ still exists at time k (*i.e.* $[E_C^k = 1]$). As we consider that objects can nott disappear, Dirac are chosen for these distributions.

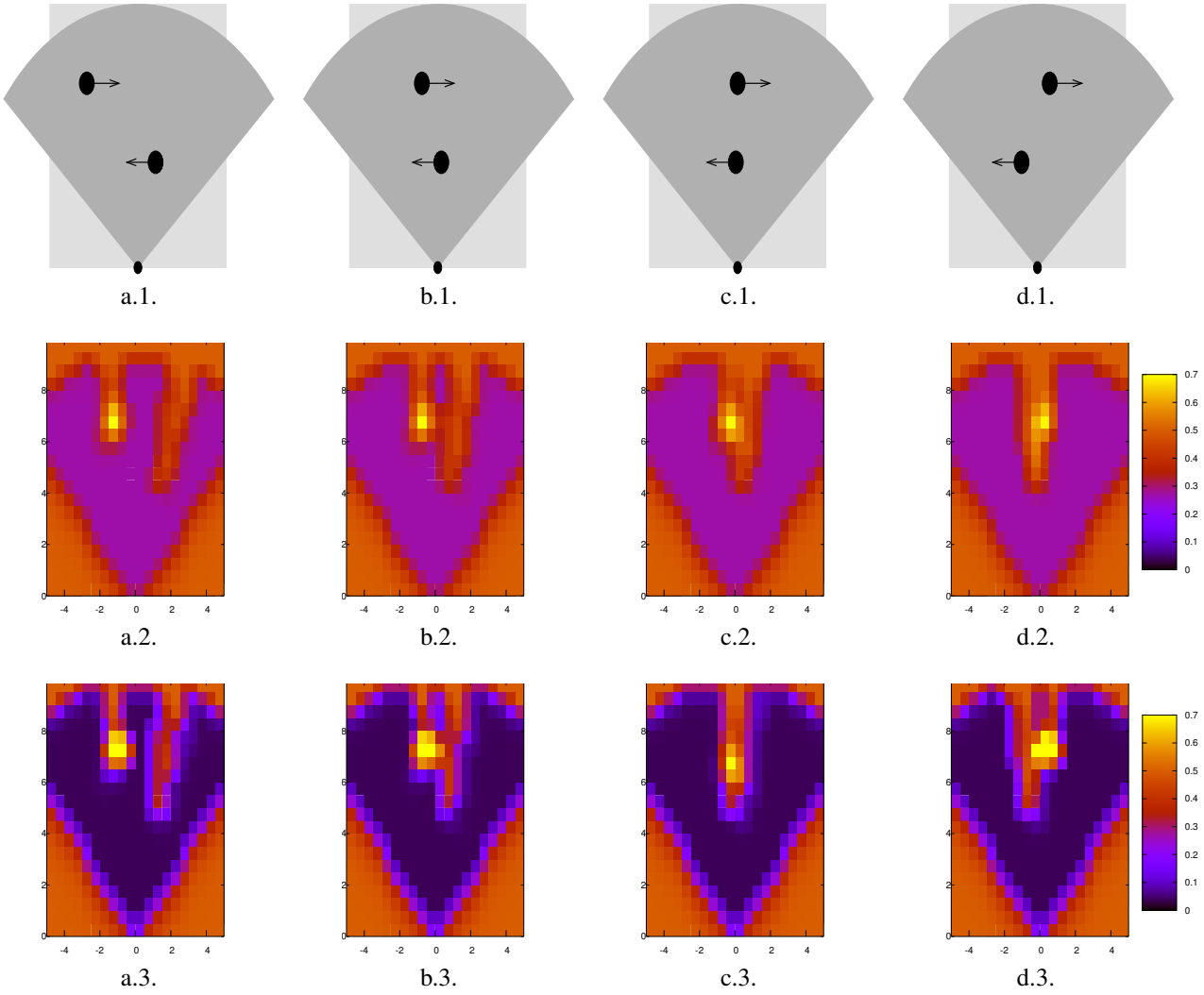


Fig. 5. A short sequence of a dynamic scene. The first row describes the situation: a moving object is temporary hidden by a second object. The second row shows the predicted occupancy grids, and the third row the result of the estimation step. The grids show $P([E_C^k = 1] \mid x\ y\ [\dot{x} = 0.0]\ [\dot{y} = 1.0])$

Now the description is complete and question can be asked. Since we want an estimate of the occupancy probability for each cell, we ask the inference engine to answer the question:

$$P(E_C^k \mid C^k\ U^{k-1}). \tag{9}$$

The result of the inference is:

$$P(E_C^k \mid C^k\ U^{k-1}) \propto \sum_{\substack{C^{k-1} \\ E_C^{k-1}}} \begin{pmatrix} P(C^k \mid C^{k-1}\ U^{k-1}) \\ \times P(E_C^{k-1} \mid C^{k-1}) \end{pmatrix}. \tag{10}$$

In general, this expression cannot be determined analytically, and cannot even be computed in real time. Thus an approximate solution of the integral has to be computed.

Our approximation algorithm is based on the basic idea that only few points are needed to approximate the integral. Thus, for each cell of the grid at time $k-1$, we compute the probability distribution $P(C^k \mid C^{k-1})$. A cell c^k is drawn according to this probability distribution. Then the cell C^{k-1} is used to update only the predicted state of the cell c^k. The complexity of this algorithm increases linearly with the number of cells in our grid, and ensures that the most informative points are used to compute the sum appearing in (10).

So the estimation of the occupancy grid at time k is done in two steps. The prediction step uses the estimation step at time $k-1$ and a dynamical model to compute an *a priori* estimate of the grid. Then the estimation step

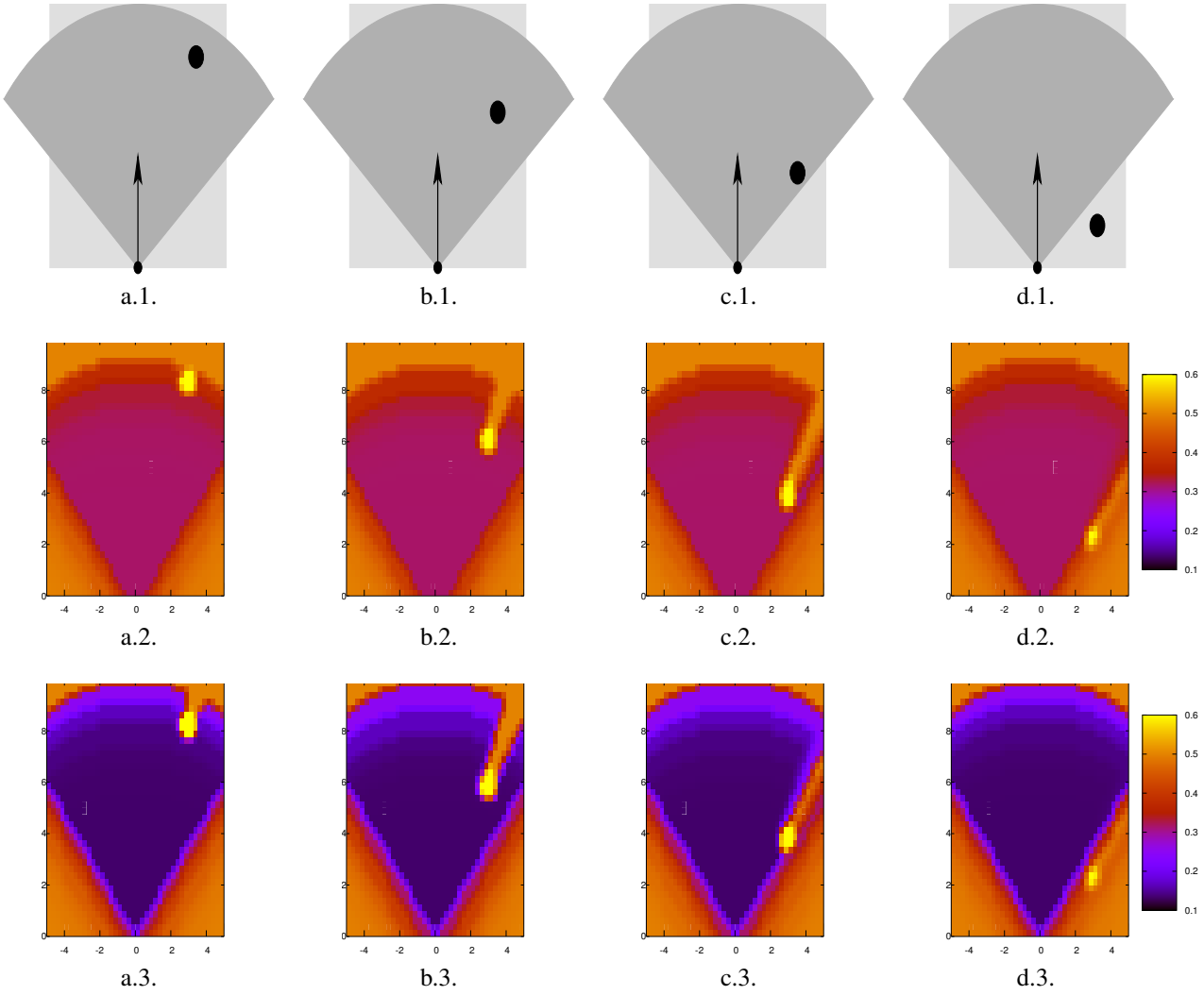


Fig. 6. A short sequence of a dynamic scene. The cycab is moving forward at a constant speed. The grids show $P([E_C^k = 1] \mid x\ y\ [\dot{x} = -2.0]\ [\dot{y} = 0.0])$

uses this prediction and the sensor observations at time k to compute the grid.

C. Experimental results

Fig 5 shows an extract of a short sequence of successive prediction and estimation results. Its goal is to demonstrate the robustness of our approach to objects occlusions, without any special logic. The rst row describes the situation : the cycab is immobile, two moving objects are in the sensor eld of view. One object is moving from left to right, the second one from right to left. In the situation depicted by the g 5 (c1), the rst object is hidden by the second one, and thus is not detected by the Sick laser range nder.

Second and third rows present respectively results of the prediction step and of the estimation step. We choose to represent only the cells of the grid corresponding to relative speed equals to $\dot{x} = 0.0,\ \dot{y} = 1.0m/s$, which is close to the speed of the temporary hidden object. The color represents the occupancy probability of the cell.

An area of high occupancy probability is well de ned in gs 5 (a2) and 5 (a3). This area corresponds to the moving object. We remark an area of occupancy probability values equals to 0.5, which corresponds to the cells hidden by the second object. Same areas are still de ned in gs 5 (b2) and 5 (b3) . The g 5 (c2) presents the result of the prediction step, based on the grid presented in g 5 (b3), and on a dynamic model. This prediction shows that an

object should be located in the area hidden by the static object. Consequently, even if this object is not detected by the laser, an area of high occupancy probability is found in the g 5 (c3). Of course, the con dence in object presence, *i.e.* the values of the occupancy probability in the grid, decreases when the object is not observed by the sensor.

In g 5 (d3) the moving object is no longer hidden by the static object. Thus it is detected by the laser, and the occupancy probability values increase.

Fig 6 shows another sequence of successive prediction and estimation results. The rst row describes the situation: in this case the cycab is moving forward. Its longitudinal speed is constant, and equals to $2.0m/s$. A static object is present in front of the cycab, in the sensor eld of view. Since the cycab is moving forward, this object nally goes out the sensor eld of view, as depicted in g 6 (d1).

As in g 5, second and third rows present respectively results of the prediction and of the estimation step. In this sequence we choose to represent only the cells of the grid corresponding to relative speed equals to $\dot{x} = -2.0,\ \dot{y} = 0.0m/s$, which is close to the relative speed of the object.

What should be noticed from this gure is that the prediction step allows to express knowledge outside the sensor eld of view. In particular in g 6 (d3), an area of high occupancy probability still exists, even if the object has not been detected by the sensor. In gs 6 (a3), (b3) and (c3), areas of occupancy probability values smaller than 0.5 could be seen outside the sensor eld of view. Thus the prediction step could be seen as a memory , since the eld of view of the sensor is increased by remembering its last observations.

Humans interpretation of an environment representation is not enough to ensure that this representation is usable for a robot. The next section shows how the Bayesian occupancy lter could be used to perform a basic and vital behavior.

V. Application to collision avoidance

The goal of this application is to prove that the Bayesian occupancy lter described in the previous section is usable and relevant in the context of ADAS. In this section, the Bayesian occupancy lter is applied to the collision avoidance of the CyCab. Our goal is to select the CyCab's forward speed in order to avoid the dynamic objects of the environment (cars and pedestrians).

As mentioned in earlier, the cell state can be used to encode a number of properties of the robot environment. Properties of interest for robot programming could include occupancy, observability, reachability, etc. In the previous section, it was used to encode the occupancy of the cell. In this section, we show how it could be used to encode the *danger* of the cell. By this way, the CyCab is longitudinally controlled by combining the occupancy and the danger of all cells.

A. Estimation of danger

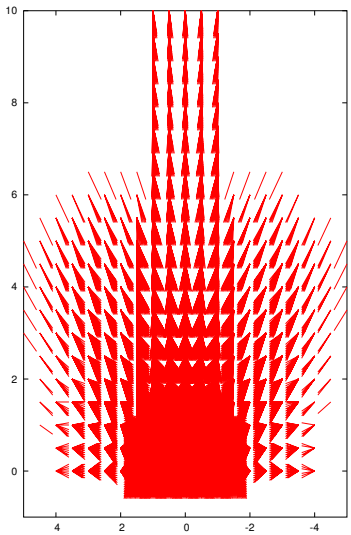


Fig. 7. Cells of high danger probabilities. For each position, arrows model the speed.

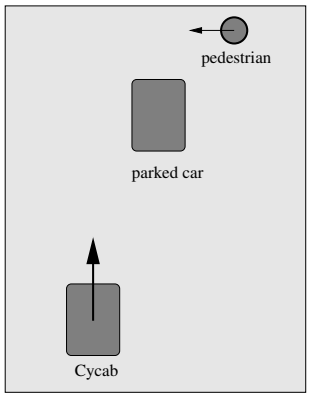


Fig. 8. Scenario description : the pedestrian is temporary hidden by a parked car.

For each cell of the grid, the probability that this cell is hazardous is estimated. This estimation is done without considering the occupancy probability of the cell. Thus we estimate the probability distribution $P(D_C^k \mid C^k)$, for each cell C of the CyCab environment. D_X^k is a boolean variable that indicates whether the cell C^k is hazardous or not.

As a cell C of our grid represents a position and a velocity, the TCPA (Time to the Closest Point of

Fig. 9. Example of the CyCab control (see Extension 1 for the video).

Approach) and the DCPA (Distance to the Closest Point of Approach) can be estimated for each cell. Thanks to TCPA and DCPA, the estimation of the danger is more intuitive than if we had considered directly the relative speed encoded in the grid : the lower the DCPA and the shorter the TCPA, the more hazardous the cell.

Fig 7 shows the cells for which danger probability is greater than 0.7. Each cell is modeled with an arrow: the beginning of the arrow indicates the position, the length and the direction indicates the speed. First, we can see that any cell located close to the CyCab is considered as hazardous, whatever the speed is. For other locations, the more hazardous cells are those which speed is in the direction of the CyCab. As we consider relative speed in the danger grid, this grid does not depend of the actual CyCab velocity.

B. Control of the CyCab

Our goal here is to control the longitudinal speed of the CyCab, in order to avoid moving objects. The behavior we want the CyCab to adopt is very simplistic : brake or accelerate whether it feels itself in danger or not.

To program this behavior, we consider simultaneously for each cell X of the environment its danger probability (given by the distribution $P(D_C^k \,|\, C^k)$ explained in the § V-A) and its occupancy probability (given by the posterior distribution $P(E_C^k \,|\, \mathcal{Z}^k\, C^k)$). We look for the most hazardous cell that is considered as occupied, that is:

$$\max_{C^k} \; \{P(D_C^k \mid C^k),\; with\; P(E_C^k \mid C^k) > 0.5\}.$$

Then the longitudinal acceleration of the CyCab is decided according to this level of danger and to its actual velocity.

Fig 8 depicts a scenario used for the test of the CyCab control. This situation involves the CyCab, a parked car and a pedestrian. The CyCab is moving forward, the pedestrian is moving from right to left. During the experimentation, the pedestrian is temporarily hidden by the parked car.

Fig 9 shows few snapshots (see also Extension 1, which shows the entire video) of the experiment. The CyCab brakes to avoid the pedestrian, and accelerate when the pedestrian has passed.

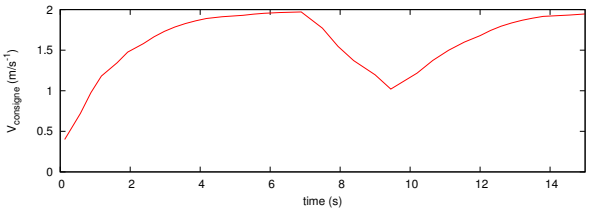


Fig. 10. Velocity of the cycab during the experiment involving a pedestrian occlusion.

Fig 10 shows the velocity of the CyCab during this experiment. From $t = 0\ s$ to $t = 7\ s$, the CyCab accelerates, up to $2\ m/s$. At $t = 7s$, the pedestrian is detected. As a collision could possibly occur, the CyCab brakes. From $t = 8.2\ s$ to $t = 9.4\ s$, the pedestrian is hidden by the car. Due to the Bayesian occupancy lter, hazardous cells of the grid are still considered as occupied. Thus the CyCab still brakes. When the pedestrian reappears, *i.e.* $t = 9.4s$, there is no more a risk of collision, so the CyCab could accelerate.

What it is to be noted here is that no decision is taken before the choice of the command applied to the CyCab. In particular, we do not know the exact number of object located in the CyCab environment. Furthermore, exact positions and velocity of these objects are not estimated.

VI. Conclusion

This paper addressed the problem of 4-D occupancy grid estimation in an automotive context. According to us, this grid can be an alternative to complex multi-target tracking algorithms for applications which does not require information such as the number of objects. To improve the estimation, a prediction step has been added. Thanks to this prediction, the estimation of the grid is robust to temporary occlusions between moving objects. To validate the approach, an application involving the CyCab vehicle has been shown. The CyCab is longitudinally controlled in order to avoid obstacles. This basic behavior

is obtained by combining the occupancy probability and the danger probability of each cell of the grid.

Future developments will include: a) improvements of the approximation algorithm for the prediction step. These improvements should allow to estimate a bigger grid, which is required to control a car in urban areas. b) fusion of the occupancy grid with higher-level information, such as GPS maps, to better estimate the danger of the situation.

Acknowledgements. This work was partially supported by the European project IST-1999-12224 *"Sensing of Car Environment at Low Speed Driving"* (`http://www.carsense.org`).

VII. REFERENCES

[1] X. Clady, F. Collange, F. Jurie, and P. Martinet. Object tracking with a pan-tilt-zoom camera : application to car driving assistance. In *Proceedings of IEEE International Conference on Robotics and Automation*, Seoul (Korea), May 2001.

[2] Y. Bar-Shalom and X. Li. *Multitarget Multisensor Tracking : Principles and Techniques*. YBS Publishing, 1995.

[3] H. Gauvrit, J.P. Le Cadre, and C Jauffret. A formulation of multitarget tracking as an incomplete data problem. *IEEE Trans. on Aerospace and Electronic Systems*, 33(4), 1997.

[4] R.L. Streit and T.E. Luginbuhl. Probabilistic multihypothesis tracking. Technical Report 10,428, Naval Undersea Warfare Center Division Newport, 1995.

[5] S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 2000.

[6] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1), 1998.

[7] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

[8] K.O. Arras, N. Tomatis, and R. Siegwart. Multisensor on-the- y localization : precision and reliability for applications. *Robotics and Autonomous Systems*, 44:131 143, 2001.

[9] E. T. Jaynes. *Probability Theory: the Logic of Science*. Cambridge University Press, 2003.

[10] O. Lebeltel. *Programmation Bayésienne des Robots*. These de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, Septembre 1999.

[11] O. Lebeltel, P. Bessiere, J. Diard, and E. Mazer. Bayesian robot programming. *Autonomous Robots*, 16:49 79, 2004.

[12] C. Coue and P. Bessiere. Chasing an elusive target with a mobile robot. In *Proceedings of the IEEE-RSJ International Conference on Intelligent Robots and Systems*, Hawai (HI), 2001.

[13] K. Mekhnacha, E. Mazer, and P. Bessiere. The design and implementation of a bayesian CAD modeler for robotic applications. *Advanced Robotics*, 15(1):45 70, 2001.

[14] R. Le Hy, A. Arrigoni, P. Bessiere, and O. Lebeltel. Teaching bayesian behaviours to videogame characters. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas (NV), October 2003.

[15] H.P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2), 1988.

[16] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer, Special Issue on Autonomous Intelligent Machines*, Juin 1989.

[17] S. Thrun. Robotic mapping: A survey. In *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[18] E. Prassler, J. Scholz, and A. Elfes. Tracking multiple moving objects for real-time robot navigation. *Autonomous Robots*, 8(2), 2000.

[19] A. H. Jazwinsky. *Stochastic Processes and Filtering Theory*. New York : Academic Press, 1970.

[20] G. Cooper. The computational complexity of probabilistic inference using bayesian belief network. *Artificial Intelligence*, 42(2-3), 1990.

[21] R.E. Kalman. A new approach to linear ltering and prediction problems. *Journal of basic Engineering*, 35, Mars 1960.

[22] G. Welch and G. Bishop. An introduction to the Kalman lter. available at `http://www.cs.unc.edu/~welch/kalman/index.html`.

The *CyCab*: a Car-Like Robot Navigating Autonomously and Safely Among Pedestrians

Cédric Pradalier, Jorge Hermosillo, Carla Koike,
Christophe Braillon, Pierre Bessière, Christian Laugier

firstname.lastname@inrialpes.fr
GRAVIR – INRIA – INPG Grenoble
INRIA Rhône-Alpes, 38334 Saint Ismier cedex France

Abstract

The recent development of a new kind of public transportation system relies on a particular double-steering kinematic structure enhancing manoeuvrability in cluttered environments such as downtown areas. We call *bi-steerable car* a vehicle showing this kind of kinematics. Endowed with autonomy capacities, the bi-steerable car ought to combine suitably and safely a set of abilities: simultaneous localisation and environment modelling, motion planning and motion execution amidst moderately dynamic obstacles. In this paper we address the integration of these four essential autonomy abilities into a single application. Specifically, we aim at reactive execution of planned motion. We address the fusion of controls issued from the control law and the obstacle avoidance module using probabilistic techniques.

Key words: Car-like robot, navigation, path planning, obstacle avoidance, autonomous navigation.

1 Introduction

The development of new Intelligent Transportation Systems (ITS), more practical, safe and accounting for environmental concerns, is a technological issue of highly urbanised societies today [18]. One of the long run objectives is to reduce the use of the private automobile in downtown areas, by offering new modern and convenient public transportation systems. Examples of these, are the CyCab robot – designed at INRIA and currently traded by the Robosoft company (see www.robosoft.fr) – and the pi-Car prototype of IEF (Institut d'Electronique Fondamentale, Université Paris-Sud).

The kinematic structure of these robots differs from that of a car-like vehicle in that it allows the steering of both the front axle and the rear one. We call a vehicle showing this feature a bi-steerable car (or BiS-car for short).

Endowed with autonomy capacities, the bi-steerable car ought to combine suitably and safely a set of abilities that eventually could come to the relief of the end-user in complex tasks (e.g. parking the vehicle). Part of these abilities have been tackled separately in previous work: simultaneous localisation and environment modelling, motion planning execution amidst static obstacles and obstacle avoidance in a moderately dynamic environment without accounting for a planned motion.

In this paper we address the integration of these four essential autonomy abilities into a single application. Specifically, we aim at reactive execution of planned motion. We address the fusion of controls issued from the control law and the obstacle avoidance module using probabilistic techniques. We are convinced that these results represent a step further towards the motion autonomy of this kind of transportation system. The structure of the paper follows.

In section 2, we sketch the environment reconstruction and localisation methods we used and we recall how the central issue regarding the motion planning and execution problem for the general BiS-car was solved. Section 3 explains how our obstacle avoidance system was designed and section 4 how it was adapted to the trajectory tracking system. In section 5 we present experimental settings showing the fusion of these essential autonomy capacities in our bi-steerable platform the CyCab robot. We close the paper with some concluding remarks and guidelines on future work in section 6.

2 Localisation, Environment modelling, Motion planning and execution

In the design of an autonomous car-like robot, we are convinced that localisation, modelling of the environment, path planning and trajectory tracking are of fundamental importance.

2.1 Map-building and Localisation

The CyCab robot is the size of a golf-cab capable of attaining up to 30Km/h. Its “natural” environment is the car-park area of the INRIA Rhône-Alpes (about $10000m^2$). For localisation purposes, we did not want to focus on the detection of natural features in the environment, since such detection is often subject to failure and not very accurate. So, in order to ensure reliability, we decided to install artificial landmarks in the environment. These landmarks had to be detected easily and

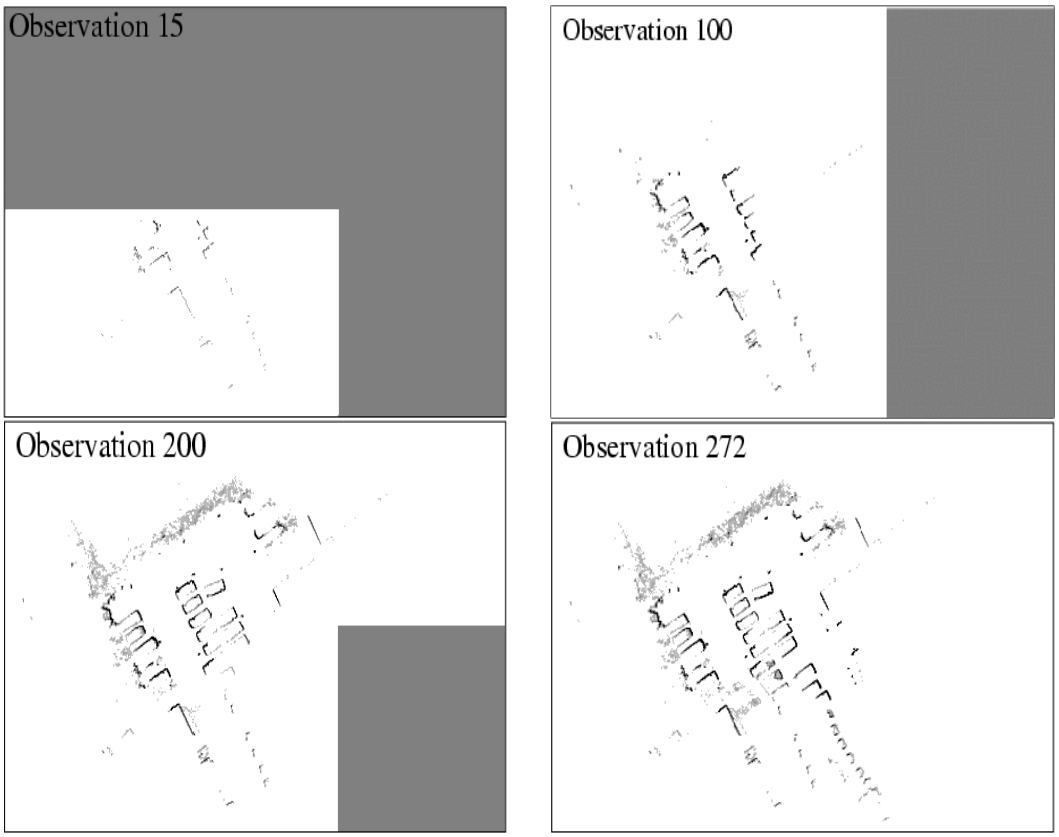


Fig. 1. Obstacle map evolution: Experimental images during the obstacle map-building phase. The vehicle is driven within the car-park area as long as needed. Simultaneously, the laser range sensor is used to detect the landmarks to build-up the localisation map.

accurately, and they should be identified with a reasonable computation effort. Fig. 2 shows our robot, its sensor and the landmarks : cylinder covered with reflector sheets, specially designed for our Sick laser range finder.

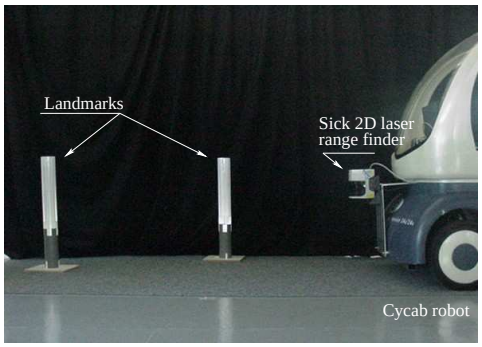


Fig. 2. Cycab robot and its landmarks for localization

Moreover, in order to keep flexibility, we wanted to be able to equip the environment with non permanent beacons. For this reason, we could not rely on a definitive landmark map, and we had to build a system able to learn the current state of the

car-park area. This led us to use SLAM [1] methods. The method which was best suited to our needs was the Geometric Projection Filter (see [21] for reference, and [24] for implementation details). It consists in building a map of features uncorrelated with the robot state. Such features are, for instance, the distance between landmarks or angles between three of them.

Owing to the accuracy of the laser range finder, to the good choice of our landmarks, and to the strength of the SLAM methods we use, we evaluate the worst case accuracy of our localisation system to the following value: about 10 centimetres in position and 2 degrees in orientation. We refer the reader to [24] for more details about the way we evaluate these values.

2.2 The Obstacle Map

The previous method localises the robot and builds a landmark map. But, we still miss a map of observed obstacles in order to plan safe paths. To achieve this goal, we build a kind of simplified occupancy grid[8] on the environment. This structure gives us informations correlated with the probability that a given place is the boundary of an obstacle.

Both maps are built online, in real-time, by the robot during the construction phase. Fig. 1 shows how the obstacle map evolves while we are exploring the environment. This map is made of small patches which are added according to the need of the application. In this way, the map can be extended in any direction, as long as memory is available. Once the map-building phase has finished, the obstacle map is converted into a pixmap and passed to the Motion Planning stage.

2.3 Motion Planning Amidst Static Obstacles

The Motion Planner adopted for the CyCab was presented in [26]. Essentially, it is a two step approach, dealing separately with the physical constraints (the obstacles) and with the kinematic constraints (the non-holonomy). The planner first builds a collision-free path without taking into account the non-holonomic constraints of the system. Then, this path is approximated by a sequence of collision-free feasible sub-paths computed by a *suitable* [2] steering method. Finally, the resulting path is smoothed.

A key issue in non-holonomic motion planning is to find a steering method accounting for the kinematics of the robot. One way of designing steering methods for a

[1] Simultaneous Localisation And Mapping

[2] i.e. Verifying the topological property as explained in [26].

non-holonomic system is to use its ***flatness*** property [10] allowing also for feedback linearisation of the nonlinear system (this is discussed in section 2.6). This is what we did for the general BiS-car for which a flat output—or linearising output—was given in [26].

2.4 Steering a BiS-car

The kinematics model of a general bi-steerable vehicle and its flat output are shown in Fig. 3.

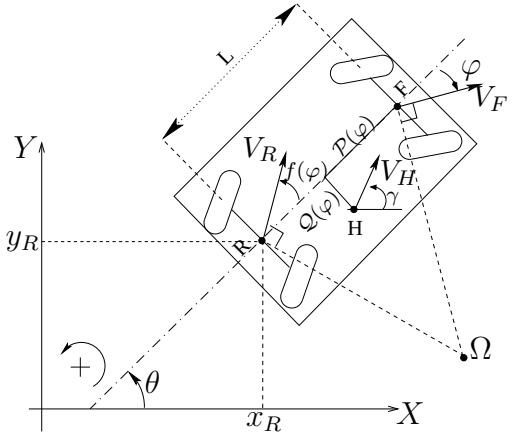


Fig. 3. Cycab robot, its landmarks and its kinematics model showing the coordinates of the flat output (point H) with respect to the reference frame of the robot placed at point F. In our case we have that $(x_{\scriptscriptstyle F}, y_{\scriptscriptstyle F}, \theta, \varphi)$ is the state of the robot.

The striking advantage of planning a path in the flat space is that we only need to parameterise a 2-dimensional curve whose points and derivatives define everywhere the current n-dimensional state[3] of the robot (in the case of the BiS-car $n = 4$). The main characteristic of such a curve is its curvature κ from which the steering angle can be computed.

Fig. 4 shows the outcome of the motion planner using an obstacle map generated as described in the previous section.

2.5 User-Planner Interface

The User-Planner interface in the CyCab is achieved through a ***touch-screen*** superposed to a 640×480 pixels LCD display. Additionally, we use the keyboard to allow for the entrance of data.

[3] The configuration space in robotics is called the ***state space*** in control theory, so we will use indistinctly both terms.

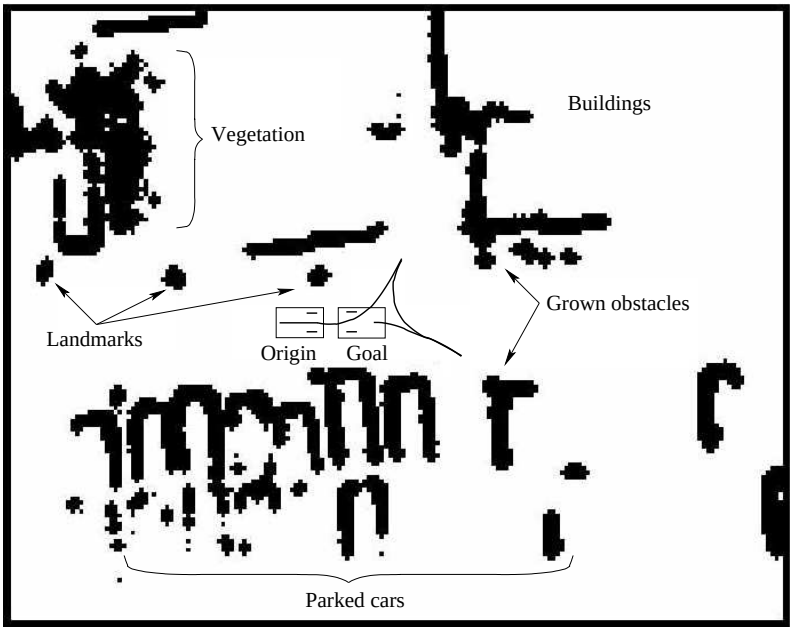


Fig. 4. Path computed by the motion planner using a real obstacle map. The obstacles are grown as well as the robot before computing the path.

The interface is used to display the current position of the robot within its environment and to capture the goal position entered by the user. These positions together with the obstacle map is passed to the motion planner. The output path is then displayed allowing the user to validate the path or start a new search.

Finally, the reference trajectory is generated using a regular parameterisation of the path [16] and the user is requested to accept to start the execution of the trajectory.

2.6 Trajectory tracking using flatness

It is well known that a non-holonomic system cannot be stabilised using only smooth state static feedbacks [6]. Ever since then, time-varying feedbacks [25] and dynamic feedbacks have been successfully used in particular for the canonical tractor-trailer and car-like robots [9].

Flat systems are feedback linearisable by means of a restricted class of dynamic feedback called *endogenous* [10]. The interest is that we are able to use state-of-the-art linear control techniques to stabilise the system. We present here results coming from recent work on feedback linearisation of the general BiS-car.

For a reference frame of the robot placed at point F in Fig. 3, the flat output $\mathbf{y} = (y_1, y_2)^T$ of a BiS-car are the coordinates of a point $H = (x_H, y_H)^T = (y_1, y_2)^T$,

computed as a function of the state as follows:

$$H = F + \mathcal{P}(\varphi)\vec{u}_\theta + \mathcal{Q}(\varphi)\vec{u}_{\theta^\perp}$$

where $\mathcal{P}(\varphi)$ and $\mathcal{Q}(\varphi)$ are coordinate functions relative to the robot's reference frame (see [26] for details) and where $\vec{u}_\theta$ (resp. $\vec{u}_{\theta^\perp}$) is the unitary vector in the direction θ (resp. the direction $\theta + \frac{\pi}{2}$).

Looking for a tractable relation between the controls of the robot and the linearising output, we found an expression giving the flat output dynamics with respect to a more convenient reference frame placed at the middle of the front axle of the robot (point F) and having orientation $\gamma = [\theta + \beta(\varphi)] \pm \pi$ where the function $\beta(\varphi)$ is the characteristic angle of the velocity vector of the flat output.

The convenience of this new reference frame relies on the fact that the velocity of the flat output has a single component in it. More precisely—assuming that $\gamma = \theta + \beta(\varphi) + \pi$—one can show that, in this reference frame, the flat output dynamics is given by the following expression [14]:

$$\begin{aligned} \frac{\partial H}{\partial t} &= v_H \, \vec{u}_\gamma \\ v_H &= v_F[\cos(\varphi - \beta - \pi) - \mathcal{Q}\mathcal{F}] + \omega_\varphi[\frac{\partial \mathcal{P}}{\partial \varphi} - \frac{\partial \beta}{\partial \varphi}\mathcal{Q}] \\ \mathcal{F}(\varphi) &= \frac{\sin(\varphi - f(\varphi))}{L\cos(f(\varphi))} \end{aligned} \tag{1}$$

where (v_F, ω_φ) are the controls of the robot (i.e. the heading and the front-steering speeds), $(\varphi - \beta - \pi)$ is the angle subtended between the velocity vector of the robot $\vec{V}_F$ and the velocity vector of the flat output $\vec{V}_H$ (see Fig. 3).

From expression (1) the open-loop controls of the robot can be found as soon as the trajectory of point H is known. As we are interested in stabilising the BiS-car around a reference trajectory, we explored the fact that, owing to the flatness property, the system is diffeomorphic to a linear controllable one [10]. The endogenous dynamic feedback that linearises the general bi-steerable system is presented in [14]. Then, from linear control theory, it can be shown that the closed-loop control stabilising the reference trajectory $\mathbf{y}^*$ has the following form :

$$y_i^{(3)} = (y_i^*)^{(3)} - \sum_{j=0}^{2} k_{i,j} \left(y_i^{(j)} - (y_i^*)^{(j)} \right) \quad i = 1, 2 \tag{2}$$

Where $(.)^{(p)}$ stands for the total derivative of order p. See [7] for details.

3 Obstacle avoidance using probabilistic reasoning

The previous approach considers trajectories in a static environment. In order to make the execution of these trajectories more robust, an obstacle avoidance system should be prepared to react to unpredicted changes in the environment. This section presents the principles of our obstacle avoidance module.

3.1 State of the art on reactive trajectory tracking

Most of the approaches for obstacle avoidance are local([11,5]), that is they do not try to model the whole environment. They goal is rather to use sensor measures to deduce secure commands. Being simpler and less computationnaly intensive, they seem more appropriate to fast reactions in a non-static environment. On the other hand, we can not expect optimal solutions from a local method. It is possible that some peculiar obstacle configuration create a dead-end from which the robot cannot escape with obstacle avoidance only.

3.1.1 Potential fields

The general idea of potential fields methods, proposed initially by O. Khatib in 1986, is to build a function representing both the navigation goals and the need for obstacle avoidance. This function is built so has to decrease when going closer to the goal and to increase near obstacles. Then, the navigation problems is reduced to an optimisation problem, that is, to find the commands that brings the robot to the global minimum of the function. This later can be defined with respect to the goal and the obstacles but other constraints can also be added therein.

Numerous extensions to the potential fields have been proposed since 1986. Among others, we can cite the Virtual Force Fields [3], the Vector Field Histograms [4] and their extensions VFH+[28] and VFH*[29]. Basically, these methods try to find the best path to the goal among the secure ones.

3.1.2 Steering Angle Field (SAF)

The SAF method, proposed by *Feiten et al.* in 1994, use obstacles to constrain steering angle in a continuous domain. Simultaneously, speed control is an iterative negociation process between the high-level driving module and the local obstacle-avoidance module.

One of the first extension to this method was published in [27]. It express the collision avoidance problem as an optimisation problem in the robot controls space

(linear and rotational speeds).

3.1.3 Dynamic Window

The Dynamic Window approach[11] propose to avoid obstacles by exploring command space in order to maximise an objective function. This later accounts for the progression toward the goal, the position of closer obstacles and current robot controls. Being directly derived from the robot dynamic, this methode is particularly well adapted to high speed movements.

The computational cost of the optimization process is reduced using the dynamic caracteristics of the robot (bounded linear and angular acceleration) so as to reduce the searched space. This kind of constraints are called *Hard Constraints* since the must be respected. Conversely, when the objective function includes preferences on the robot movement, we call the resulting constraints *Soft Constraints*.

3.1.4 Dynamic environments and Velocity Obstacles

In the specific case of moving obstacles, special methods have been proposed[17,2] using the *Velocity Obstacle* notion. Basically, this notion consist in projecting perceived obstacles and their expected movement in the space of secure commands. So, each mobile object generates a set of obstacles in the command space. These obstacles represent the commands that will bring to a collision in the future.

In the general case, obstacle movement parameters are not known *a priori*, so they have to be deduced from sensor data. Obstacle avoidance controls are then computed in reaction to theses previsions. Currently, it is still quite difficult to get reliable previsions of the obstacles future trajectory. Consequently, these obstacle avoidance methods are not appliable in real situations yet.

3.1.5 Obstacle avoidance and trajectory following

When we want to perform obstacle avoidance manoeuvres while following a trajectory, a specific problem appear. On our non-holonomous robot, the path planning stage took into account the kinematic of the robot and planned a feasible path. When the reactive obstacle avoidance generates commands, the vehicle leaves its planned trajectory. Then, we cannot be sure anymore that the initial objective of the trajectory is still reachable.

A solution to this problem was proposed in [20]. This method tries to deform the global trajectory in order to avoid the obstacle, respect the kinematic constraints and ensure that the final goal is still reachable. Even if theoretically very interesting, this obstacle avoidance scheme is still difficult to apply in real situations

due to it computational complexities, especially on an autonomous car. In our experiments[20], the vehicle had to stop for several minutes in order to perform the trajectory deformation

3.2 Objectives

After all these results on obstacle avoidance, it seems obvious that our goal is not to propose a new solution to this problem. It has been shown[19,1] that probalities and bayesian inference are appropriate tools to deal with real world uncertainty and to model reactive behaviors. We this in mind, we wanted to think about the expression of the obstacle avoidance problem as a bayesian inference problem. Consequently, the originality of our approach is mainly its expression and the semantic we can express with it.

3.3 Specification

The CyCab can be commanded through a speed V and a steering angle Φ. It is equipped with π radians sweeping laser range finder. In order to limit the volume of the data we manipulate, we summarised the sensor output as 8 values : the distances to the nearest obstacle in a $\pi/8$ angular sector(see Fig. 5). We will call $D_k, k = 1 \dots 8$ the probabilistic variables corresponding to these measures.

Besides, we will assume that this robot is commanded by some high-level system (trajectory following for instance) which provides it with a pair of desired commands (V_d, Φ_d).

Our goal is to find commands to apply to the robot, guarantying the vehicle security while following the desired command as much as possible.

3.4 Sub-models definition

Given the distance D_i measured in an angular sector, we want to express a command to apply that is safe while tracking desired command. Nevertheless, since this sector only has limited information about robot surrounding, we choose to express the following conservative semantic: tracking the desired command should be a soft constraint whereas an obstacle avoidance command should be a hard constraint, the closer the obstacle, the harder the constraint.

We express this semantic using a probability distribution over the commands to apply (V, Φ) knowing the desired commands and the distance D_i measured in this

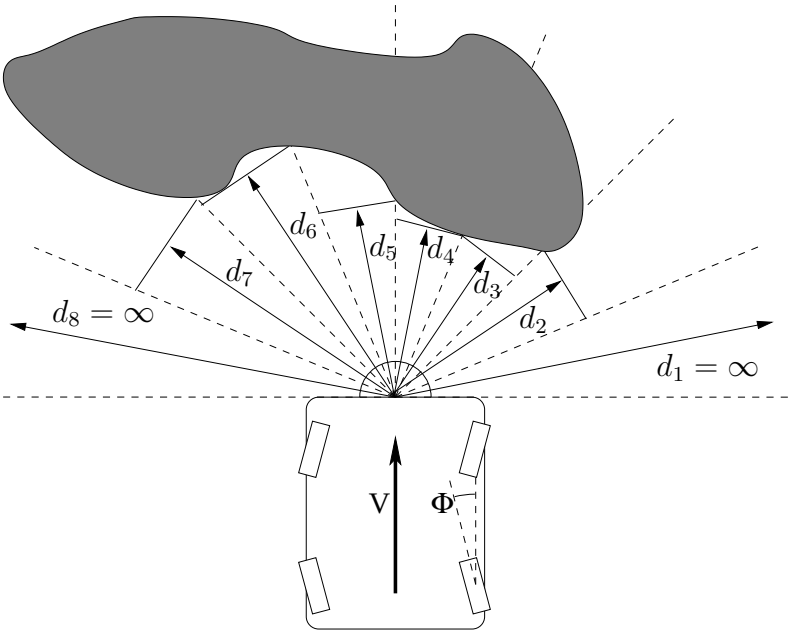


Fig. 5. Obstacle avoidance: situation

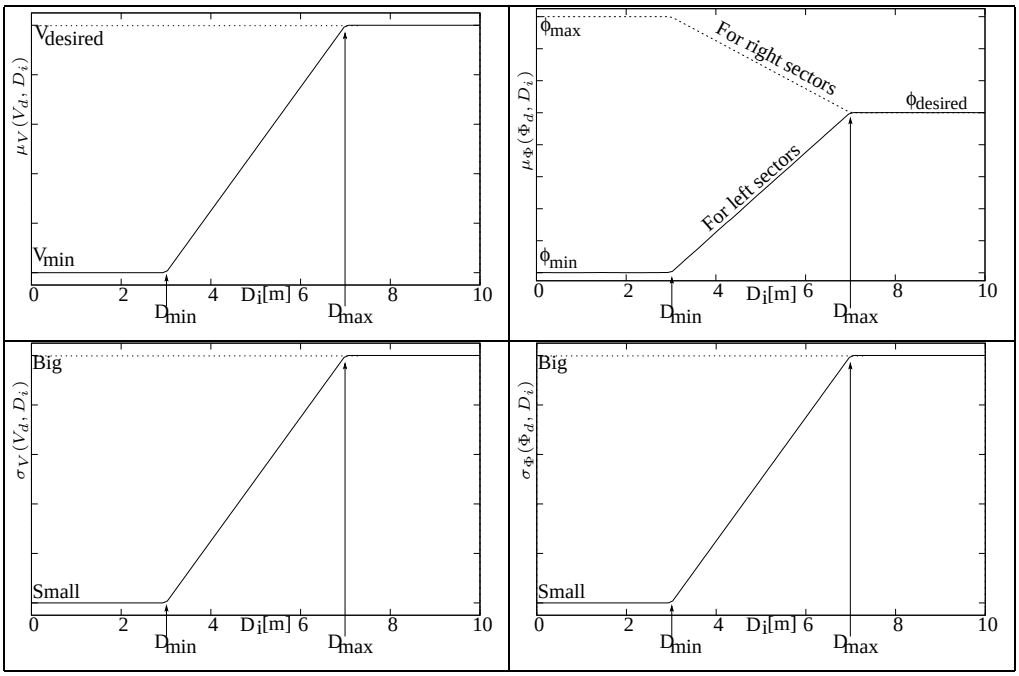


Fig. 6. Evolution of mean and standard deviation of $P_i(V \mid V_d D_i)$ and $P_i(\Phi \mid \Phi_d D_i)$ according to distance measured

sector:

$$P_i(V\Phi \mid V_d\Phi_d D_i) = P_i(V \mid V_d D_i) P_i(\Phi \mid \Phi_d D_i) \tag{3}$$

where $P_i(V \mid V_d D_i)$ and $P_i(\Phi \mid \Phi_d D_i)$ are Gaussian distributions respectively centred on $\mu_V(V_d, D_i)$ and $\mu_\Phi(\Phi_d, D_i)$ with standard deviation $\sigma_V(V_d, D_i)$ and $\sigma_\Phi(\Phi_d, D_i)$. Functions $\mu_V, \mu_\Phi, \sigma_V, \sigma_\Phi$ are defined with sigmoid shape as illustrated in Fig. 6. Example of resulting distributions are shown in Fig. 7.

There is two specific aspects to notice in Fig. 6 and 7. First, concerning the means μ_V and μ_Φ, we can see that, the farther the obstacle, the closer to the desired command μ will be, and conversely, the nearer the obstacle, the more secure μ: minimal speed, strong steering angle.

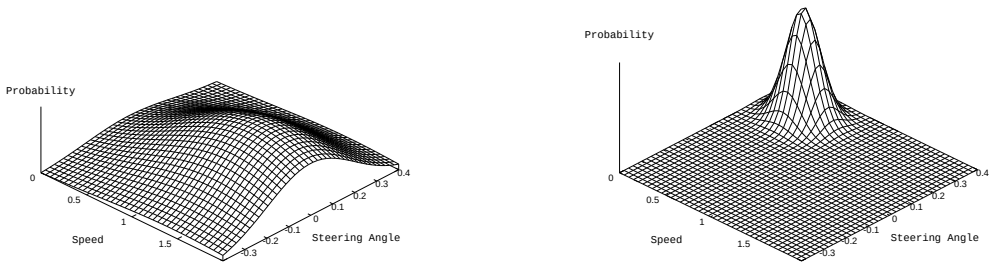


Fig. 7. Shape of $P_i(V\Phi \mid V_d\Phi_d D_i)$ for far and close obstacles

Second, the standard deviation can be seen as a constraint level. For instance, when an obstacle is very close to the robot (small D_i), its speed *must* be strongly constrained to zero, this is expressed by a small standard deviation. Conversely, when obstacle is far, robot speed *can* follow the desired command, but there is no damage risk in not applying exactly this command. This low level constraint is the result of a big standard deviation.

3.5 Command fusion

Knowing desired controls and distance to the nearest obstacle in its sector, each sub-model, defined by $P_i(V\Phi \mid V_d\Phi_d D_i)$, provides us with a probability distribution over the robot controls. As we have eight sectors, we will have to fuse the controls from eight sub-models. Then we will find the best control in term of security and desired control following.

To this end, we define the following joint distribution:

$$\begin{aligned} P(V\ \Phi\ V_d\ \Phi_d\ D_1 \ldots D_8\ S) = {} & P(D_1 \ldots D_8)\ P(V_d\ \Phi_d) \\ & P(S)\ P(V\ \Phi\ \mid\ V_d\ \Phi_d\ D_1 \ldots D_8\ S) \end{aligned} \tag{4}$$

where variable $S \in [1 \ldots 8]$ express which sector is considered. $P(D_1 \ldots D_8)$ and $P(V_d\Phi_d)$ are unknown distribution [4]. As there is no need to favour a specific sub-model, we define $P(S)$ as a uniform distribution. The semantic of S will be em-

[4] Actually, as we know we will not need them in future computation, we don't have to specify them.

phasised by the definition of $P(V\Phi \mid V_d\Phi_d D_1 \ldots D_8 S)$:

$$P(V\Phi \mid V_d\Phi_d D_1 \ldots D_8[S=i]) = P_i(V\Phi \mid V_d\Phi_d D_i)$$

In this equation, we can see that the variable S acts as model selector: given its value i, the distribution over the commands will be computed by the sub-model i, taking into account only distance D_i.

Using equation 4, we can now express the distribution we are really interested in, that is the distribution over the commands accounting for all the distances but not variable S:

$$P(V\ \Phi\ \mid\ V_d\ \Phi_d\ D_1 \ldots D_8) = \sum_S (P(S)P(V\ \Phi\ \mid\ V_d\ \Phi_d\ D_1\ \ldots\ D_8\ S)) \quad (5)$$

This equation is actually the place where the different constraint level expressed by functions σ_V and σ_Φ will be useful. The more security constraints there will be, the more peaked will be the sub-model control distribution. So sub-models who see no obstacles in their sector will contribute to the sum with quasi-flat distribution, and those who see perilous obstacles will add a peaky distribution, hence having more influence (see Fig. 8). Finally the command really executed by the robot is the one which maximise $P(V\ \Phi\ \mid\ V_d\ \Phi_d\ D_1 \ldots D_8)$ (eq. 5).

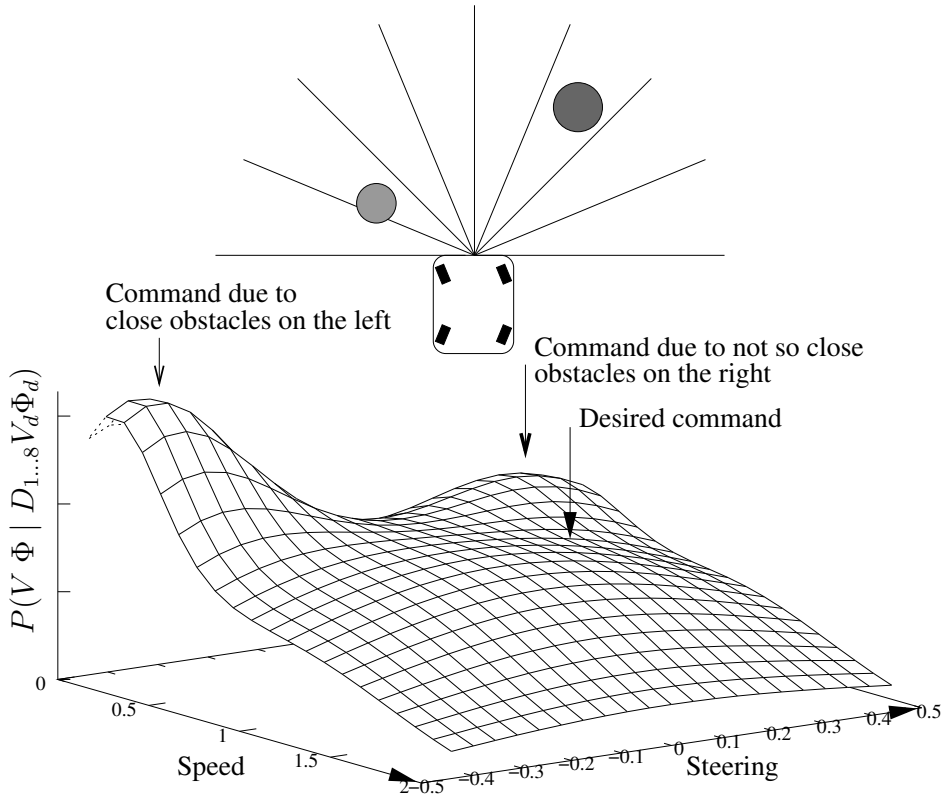


Fig. 8. Probability distribution over speed and steering, resulting from the obstacle avoidance system.

Fig. 9 illustrates the result of the obstacle avoidance system applied on a simulated example. The simulated CyCab is driven manually with a joystick in a square environment. In this specific situation, the driver is continuously asking for maximum speed, straight forward (null steering angle). We can observe on the dotted trajectory that, first obstacle avoidance module bends the trajectory in order to avoid the walls, and second, when there is no danger of collisions, desired commands are applied exactly as requested.

From the density of dots, we can figure out the robot speed: it breaks when it comes close to the walls and while its turning and try to follow desired speed when obstacles are not so threatening.

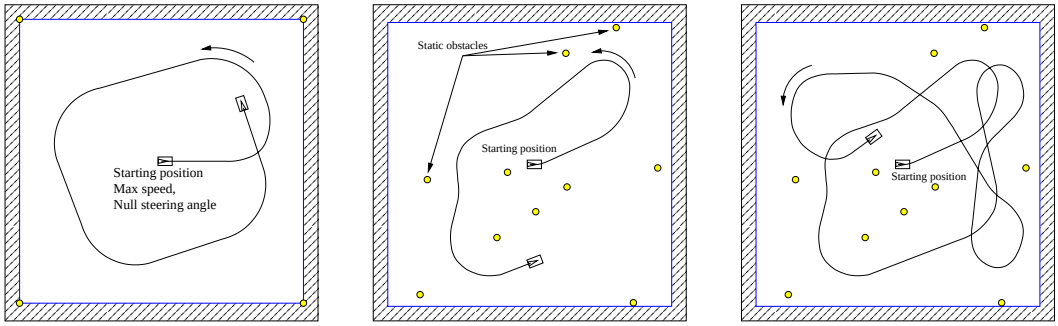


Fig. 9. Robot trajectory while driven manually with constant desired steering angle

3.7 Relation to fuzzy logic approaches

The design of our obstacle avoidance modules may remind some readers of a fuzzy logic controller[15,22,12]. It is rather difficult to say that one approach is better than the other. Both fuzzy logic and bayesian inference view themselves as extension of classical logic. Furthermore, both methods will deal with the same kind of problems, providing the same kind of solutions. Some will prefer the great freedom of fuzzy logic modelling and others will prefer to rely on the strong mathematical background behind bayesian inference.

As far as we can see, the choice between fuzzy logic and bayesian inference is rather an personal choice, similar to the choice of a programming language: it has more consequences on the way we express our solution than on the solution itself. To extend the analogy, one might relate fuzzy logic to the C language whereas Bayesian inference would be closer to Ada.

4 Trajectory tracking with obstacle avoidance

The method presented in the previous section provides us with an efficient way to fuse a security system and orders from a high level system. Nevertheless the perturbations introduced in the trajectory following system by obstacle avoidance are such that they can make it become unstable. In this section will show how we integrate trajectory tracking and obstacle avoidance.

While following the trajectory, obstacle avoidance will modify certain commands in order to follow as much as possible desired orders while granting security. These modifications may introduce delay or diversions in the control loop. If no appropriate action is taken to manage these delays the control law may generate extremely strong accelerations or even become unstable when obstacles are gone. This is typically the case when our system evolves among moving pedestrians. Thus we designed a specific behaviour to adapt smoothly our control system to the perturbations induced by obstacle avoidance.

4.1 Multiplexed trajectory tracking

4.1.1 Validity domain of flat control law

Experimentally, we found that the control law based on flatness can manage errors in a range of about 1 meter and 15 degrees around nominal trajectory. Furthermore, as this control law controls the third derivative of the flat output (eq. 2), it is a massively integrating system. For this reason, a constant perturbation such as immobilisation due to a pedestrian standing in front of the vehicle will result in a quadratic increase of the control law output. This phenomena is mainly due to the fact that when obstacle avoidance slows the robot down, it strongly breaks the dynamic rules around which the flat control law was built. So, there is no surprise in its failure.

4.1.2 Probabilistic control law

In order to deal with the situations that flat control law cannot manage, we designed a trajectory tracking behaviour (*TTB*) based again on probabilistic reasoning (section 4.2). As this behaviour has many similarities with a weighted sum of proportional control laws, we do not expect it to be sufficient to stabilise the robot on its trajectory. Nevertheless, it is sufficient to bring it back in the convergence domain of the flat control law when obstacle avoidance perturbations have occurred. Basically, the resulting behaviour is as follows: while the robot is close to its nominal position, it is commanded by flat control law. When, due to obstacle avoidance, it is too far from its nominal position, TTB takes control, and try to bring it back to

flat control law's convergence domain. When it enters this domain, flat control law is reinitialised and starts accurate trajectory tracking(this is illustrated in fig. 10).

4.1.3 Time control

Path resulting from path planning (section 2.3) is a list of robot configuration indexed by time. So when the robot is slowed down by a traversing pedestrian, it compensates its delay by accelerating. Nevertheless, when the robot is stopped during a longer time, let's say fifteen seconds, it should not consider to be delayed of fifteen seconds, otherwise it will try to reach a position fifteen second ahead, without tracking the intermediary trajectory. To tackle this difficulty, we introduced a third mode to the trajectory tracking: when the robot comes too far from its nominal position, we freeze the nominal position, and we use the TTB to reenter the domain where nominal position can be unfrozen.

The global system is illustrated by Fig. 10: we implemented some kind of multiplexer/demultiplexer which manage transitions between control laws. In order to avoid oscillating between control laws when at the interface between two domains of validity, we had to introduce some hysteresis mechanism in the switching. This is illustrated in Fig. 10.

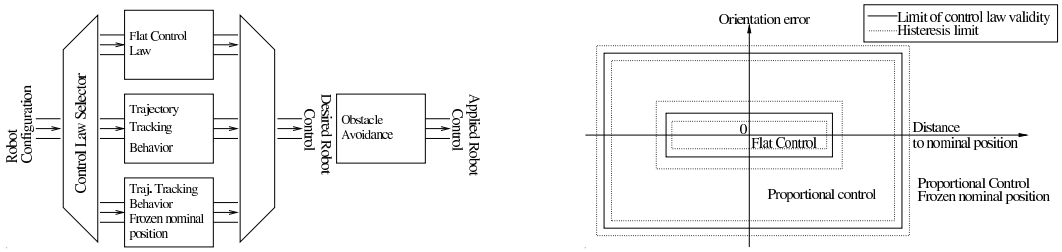


Fig. 10. Basic diagram of the control law selector mechanism and validity domains of the control laws

4.2 Trajectory tracking behaviour

Our trajectory tracking behaviour was built as a probabilistic reasoning, in a way similar to the obstacle avoidance presented above (section 3). Functionnaly, it is very similar to a fuzzy control scheme as presented in [15] and illustrated in [12].

To specify our module, we use a mechanism of fusion with diagnosis[23]. If A and B are two variables, we will define a diagnosis boolean variable I_A^B which express a consistency between A and B. Then, A and B will be called the ***diagnosed variables*** of I_A^B.

Our goal is to express the distribution over the desired controls (V_d, Φ_d) knowing reference controls (V_r, Φ_r) planned by the path planning stage, and error in position

$(\delta X, \delta Y)$ and orientation $\delta\theta$ with respect to the nominal position. Fig. 11 illustrates theses variables.

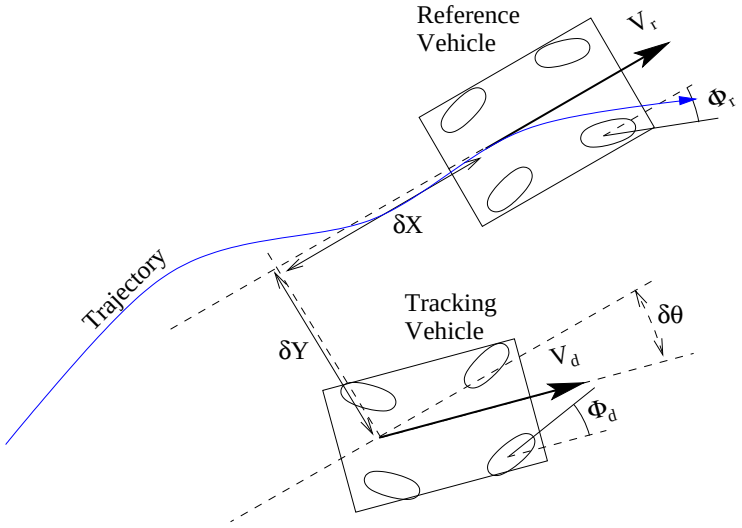


Fig. 11. Variables involved in trajectory tracking behaviour

In addition to the preceding variables, we will add five diagnosis variables $I_{V_d}^{\delta X}$, $I_{V_d}^{V_r}$, $I_{\Phi_d}^{\delta Y}$, $I_{\Phi_d}^{\delta\theta}$ and $I_{\Phi_d}^{\Phi_r}$. Variables linked to an error variable $(\delta X, \delta Y, \delta\theta)$ will diagnose if a given command helps correcting this error. Variables linked to reference commands evaluate if a command is similar to the reference one.

All these variables describe the relation between their diagnosed variables in the following joint distribution:

$$\begin{aligned} P(V_d\,\Phi_d\,V_r\,\Phi_r\,\delta X\,\delta Y\,\delta\theta\;I_{V_d}^{\delta X}\,I_{V_d}^{V_r}\,I_{\Phi_d}^{\delta Y}\,I_{\Phi_d}^{\delta\theta}\,I_{\Phi_d}^{\Phi_r}) = \\ P(V_d\;\Phi_d)\;P(V_r\;\Phi_r)\;P(\delta X\;\delta Y\;\delta\theta) \\ P(I_{V_d}^{\delta X}\mid V_d\;\delta X)\;P(I_{V_d}^{V_r}\mid V_d\;V_r) \\ P(I_{\Phi_d}^{\delta Y}\mid \Phi_d\;\delta Y)\;P(I_{\Phi_d}^{\delta\theta}\mid \Phi_d\;\delta\theta\;V_d)\;P(I_{\Phi_d}^{\Phi_r}\mid \Phi_d\;\Phi_r) \end{aligned} \tag{6}$$

Using this joint distribution and Bayes rule, we will be able to infer

$$\begin{aligned} P(V_d\,\Phi_d\,|\,(V_r\;\Phi_r)\;(\delta X\;\delta Y\;\delta\theta) \\ [I_{V_d}^{\delta X}=1]\;[I_{V_d}^{V_r}=1]\;[I_{\Phi_d}^{\delta Y}=1]\;[I_{\Phi_d}^{\delta\theta}=1]\;[I_{\Phi_d}^{\Phi_r}=1]) \end{aligned} \tag{7}$$

Basically, this equation expresses the fact that we are looking for the most likely commands in order to correct tracking error while accounting for reference commands. Having all the diagnosis variables set to one enforces this semantic.

In the preceding joint distribution (eq. 6), all the diagnosed variables are assumed to be independent, and to have uniform distributions. All the information concerning the relation between them will be encoded in the distribution over diagnosis

variables. In order to define this distributions, we first define the function $d_\sigma(x, y)$ as a Mahalanobis distance between x and y:

$$d_\sigma(x, y) = e^{-\frac{2}{2}\left(\frac{x-y}{\sigma}\right)^2}$$

Then, for two variables A and B, we define

$$P([I_A^B = 1] \mid AB) = d_{S(A,B)}(A, f(B)).$$

Let's see how preceding functions S and f are defined in specific cases.

4.2.1 Proportional compensation of errors

In the case of $I_{V_d}^{\delta X}$, we set $f(\delta X) = \alpha.\delta X$ and

$$S(V_d, \delta X) = \max((1 - \beta.\delta X)\sigma_{\max}\ ,\ \sigma_{\min}).$$

Expression of f implies that the maximum of $P(I_{V_d}^{\delta X} \mid V_d\ \delta X)$ will be for a value of V_d proportional to the error δX. Expression of S defines the constraint level associated to this speed: the bigger the error, the more confident we are that a proportional correction will work, so the smaller σ.

The basic behaviour resulting from this definition is that when the robot is behind it nominal position, it will move forward to reduce its error: the bigger its error, the faster and with more confidence that this is the good control to apply.

For $I_{\Phi_d}^{\delta Y}$, we use a similar proportional scheme. Its basic meaning is that when the robot has a lateral error, it has to steer, left or right, depending on the sign of this error. Again, the bigger the error, the more confident we are that we have to steer.

Finally, the same apply for $I_{\Phi_d}^{\delta\theta}$, except that the steering direction depends not only of the orientation error, but also of the movement direction V_d.

4.2.2 Using planned controls

In the path planning stage, the trajectory was defined as a set of nominal position, associated with planned speed and steering angle. They have to be accounted for, especially when error is small.

Let's consider first $I_{V_d}^{V_r}$. We set f and S as follows: $f(V_r) = V_r$ and $S(V_d, V_r) = \sigma_{V_r} \in [\sigma_{\min}, \sigma_{\max}]$, rather close to $\sigma_{\max}$. By this way, planned speed is used as a indication to the trajectory following system. The distribution over $I_{\Phi_d}^{\Phi_r}$ is defined using the same reasoning.

4.3 Results

Fig. 12 illustrates the basic behaviour of our trajectory tracking behaviour. In both graphs, desired command will maximise either $P(V \mid \delta X\ V_c)$ or $P(\Phi \mid \delta Y\ \delta\theta\ \Phi_c)$. Since curve $P(V \mid \delta X\ V_c)$ is closer to $P(V \mid \delta X)$ than to $P(V \mid V_c)$, we can observe that longitudinal error (δX) has much more influence than reference command on the vehicle speed. In the same manner, steering angle is a trade-off between what should be done to correct lateral error (δY) and orientation error ($\delta\theta$), lightly influenced by reference steering angle.

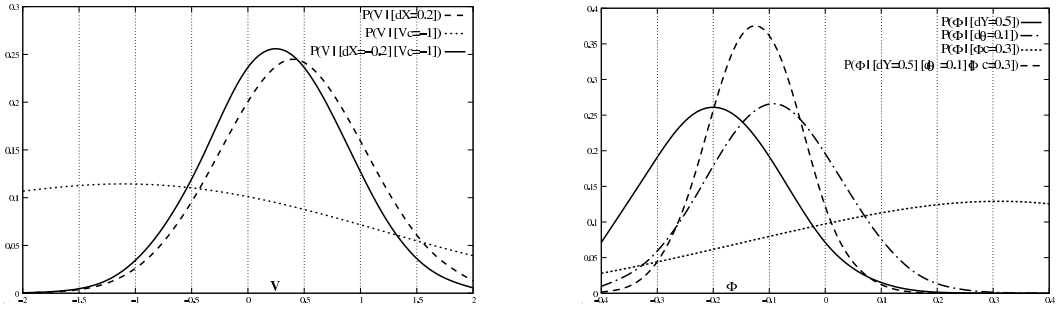


Fig. 12. Trajectory tracking : resulting command fusion

Fig. 13 shows the collaboration of obstacle avoidance and trajectory following on a simulated example. Planned trajectory passes through an obstacle which was not present at map building time. Obstacle avoidance modifies controls in order to grant security. When errors with respect to nominal trajectory is too big, our control law selector switch to the trajectory tracking behaviour. Here it is a big longitudinal error, due to obstacle avoidance slowing down the vehicle, which trigger the switching.

4.4 Discussion

Using the multiplexed control laws we managed to integrate, in the same control loop, our flat control, accurate but sensible to perturbation, with our TTB, less accurate but robust to perturbations. By this way we obtained a system capable of tracking trajectory generated by our path planner while accounting for unexpected object in the environment.

Finally, when the robot has gone too far from reference trajectory, or when reactive obstacle avoidance can not find suitable controls anymore, it may be necessary to re-plan a new trajectory to the goal. This has not been implemented on the robot yet, but this should not be considered neither a technical nor a scientific issue.

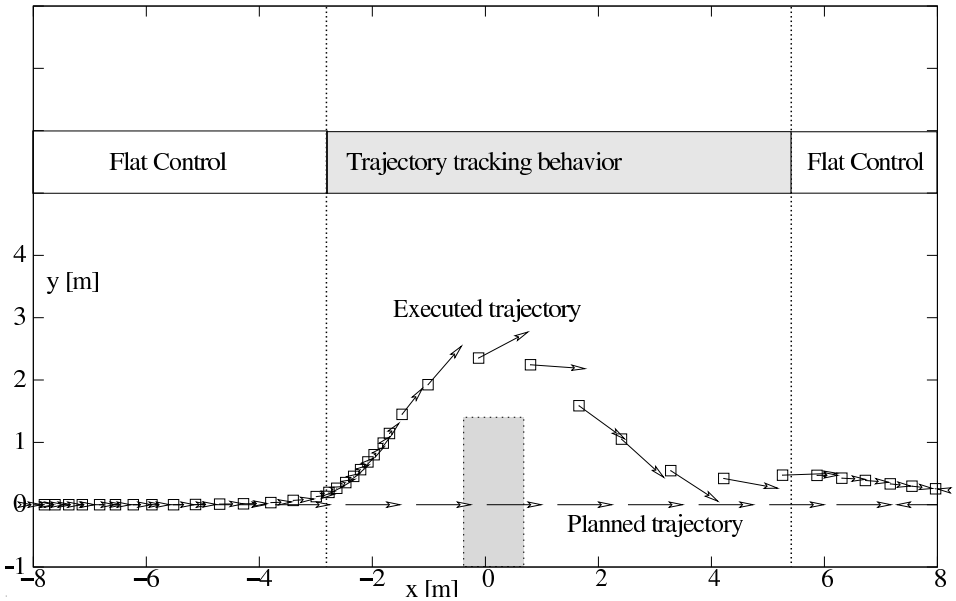


Fig. 13. Collaboration of trajectory tracking and obstacle avoidance on a simulated example

Fig. 14. An experimental setting showing from left to right: The arbitrary placing of the landmarks; the manual driving phase for landmark and obstacle map-building; the obstacle map generated together with the current position of the robot as seen on the LCD display; the capture of the goal position given by the user by means of the touch-screen; the execution of the found trajectory among aggressive pedestrians.

5 Experimental setup

We tested the integration of these essential autonomy capacities in our experimental platform the Cycab robot. The aim was to validate the theoretical considerations

made for the BiS-car and to get insight into the limitations of the whole motion scheme.

The computation power on-board the Cycab is a *Pentium II*^{TM} 233MHz running a Linux system. All programs were written in C/C++ language.

During the experiments the speed of the robot was limited to $1.5ms^{-1}$. The control rate of the robot was fixed at $50ms$. The throughput rate of the laser range-finder was limited to $140ms$ [5] ; therefore the control system has to rely momentarily in odometry[13] readings.

Fig. 14 is a set of pictures showing a complete application integrating the stages described throughout the paper.

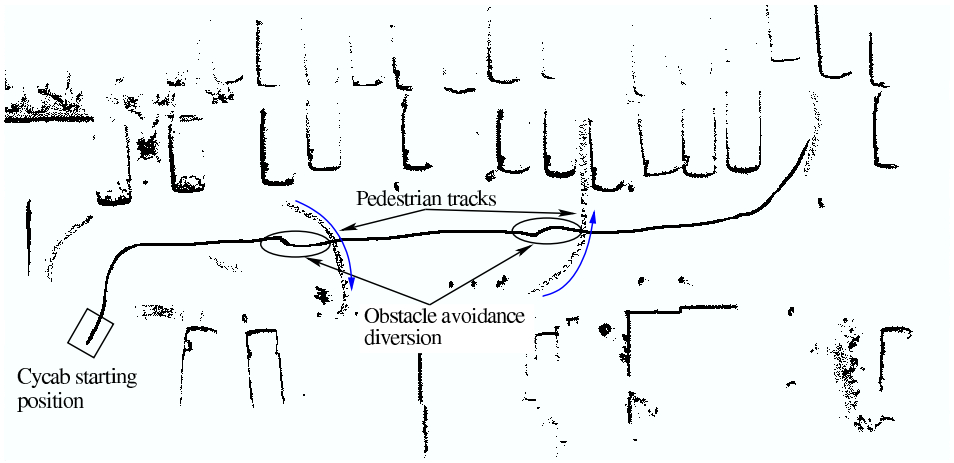


Fig. 15. Executed trajectory among static obstacles and moving pedestrians. Rear middle point (R in fig. 3) trajectory is drawn.

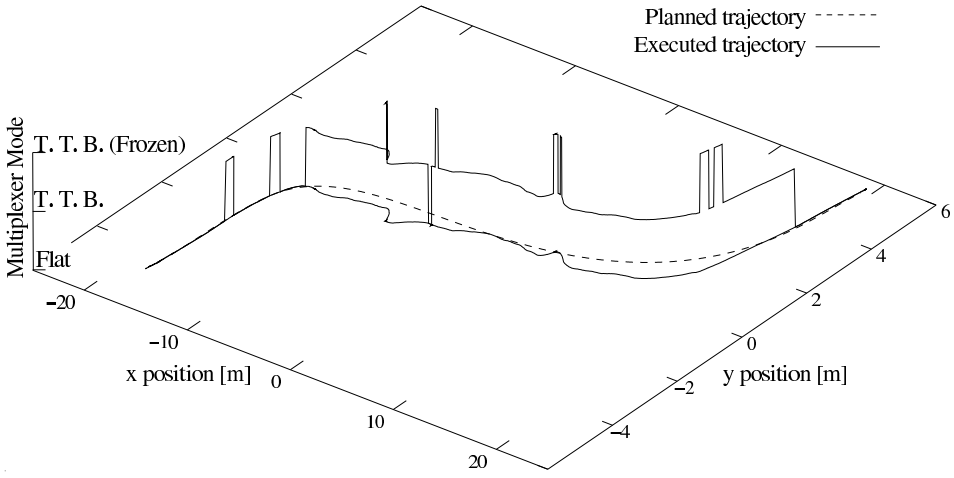


Fig. 16. Executed trajectory with respect to planned trajectory, and multiplexer mode.

[5] This rate is fair enough for our needs, even though we could use a real-time driver.

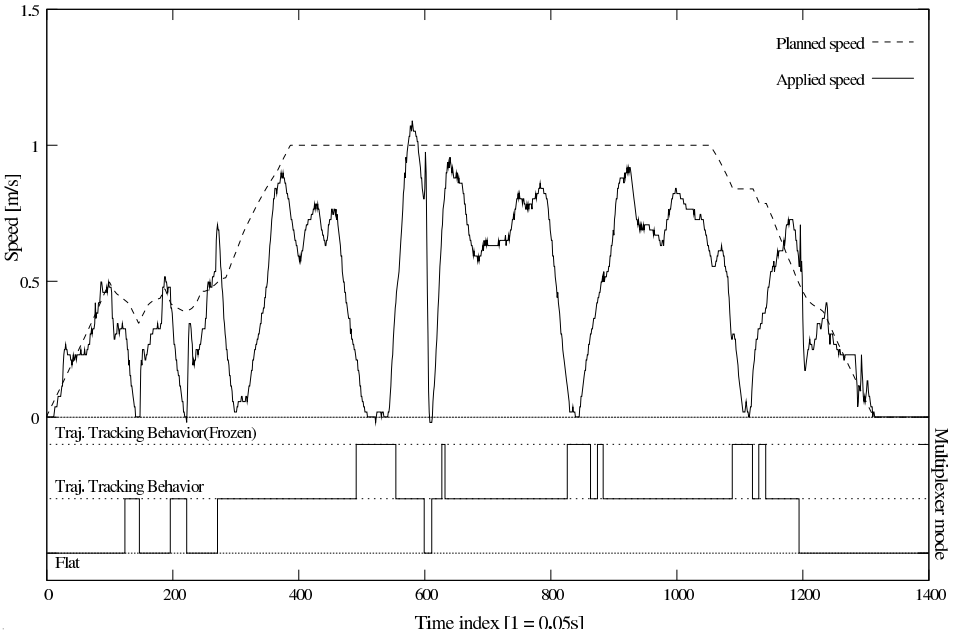


Fig. 17. Applied speeds with respect to planned speed, and multiplexer mode.

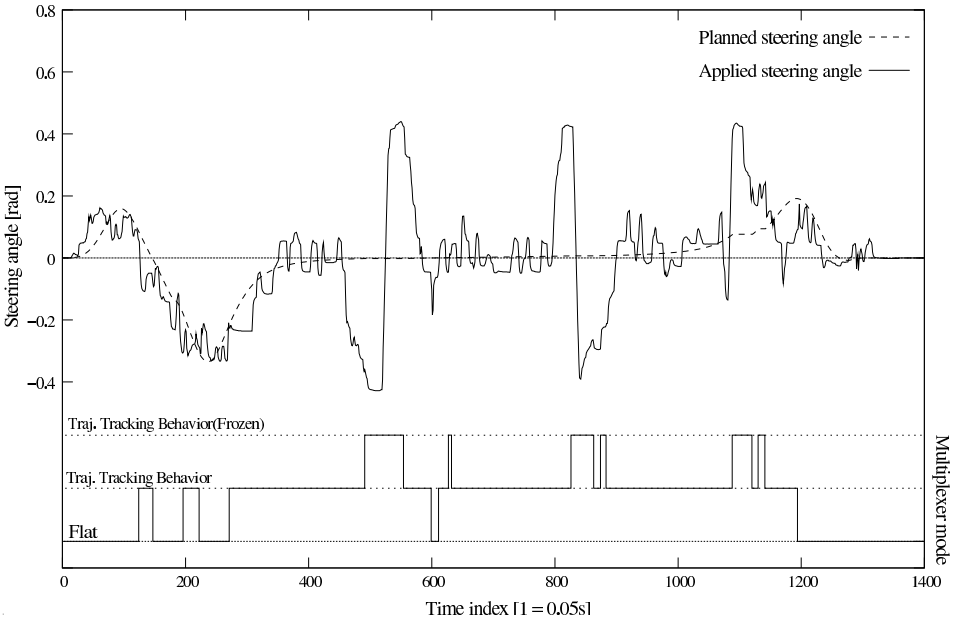


Fig. 18. Applied steering with respect to planned steering, and multiplexer mode.

Figs 15 to 18 illustrates how a planned trajectory is executed while avoiding moving pedestrians. In this environment, the control law using flatness could only be used at the beginning and at the end of the trajectory. On the remaining of the trajectory, speed and steering angle are adjusted in order to maintain security while keeping pace with the plan as much as possible.

6 Discussion & Conclusions

In this paper, we presented our new steps toward the autonomy of a bi-steerable car. The integration of localisation, map building, trajectory planning and execution in a moderately dynamic environment was discussed. Control law using the CyCab flatness property was found to be insufficient for trajectory tracking among moving pedestrians.

Even if this integration was successful and provides satisfactory results, we are convinced that a reactive behaviour cannot be sufficient for the autonomy of vehicle in a real urban environment. For this reason, we are working on the perception and identification of road users (pedestrians, cars, bikes or trucks). By this way, we will be able to predict future movement of "obstacles" and to react accordingly, in a *smarter* way than the simple scheme proposed in this paper.

References

[1] P. Bessière and BIBA-INRIA Research Group. Survey: Probabilistic methodology and techniques for artefact conception and development. Technical Report RR-4730, INRIA, Grenoble, France, February 2003. http://www.inria.fr/rrrt/rr-4730.html.

[2] S. Blondin. Planification de mouvements pour véhicule automatisé en environnement partiellement connu. Mémoire de Diplôme d'Etudes Approfondies, Inst. Nat. Polytechnique de Grenoble, Grenoble (FR), June 2002.

[3] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, Sept/Oct 1989.

[4] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, June 1991.

[5] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Detroit, MI (US), May 1999.

[6] R.W. Brockett. Asymptotic stability and feedback stabilization. In R.W. Brockett, R.S. Millman, and H.J. Sussmann, editors, *Differential Geometric Control Theory*, pages 181–191, Boston, MA: Birkhäuser, 1983.

[7] Carlos Canudas de Wit, Bruno Siciliano, and George Bastin Eds. *Theory of Robot Control*. Springer-Verlag, 1996.

[8] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer, Special Issue on Autonomous Intelligent Machines*, June 1989.

[9] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. Design of trajectory stabilizing feedback for driftless flat systems. In *Proc. of the European Control Conference*, pages 1882–1887, Rome, Italy, september 1995.

[10] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. Flatness and defects of nonlinear systems: introductory theory and examples. *Int. Journal of Control*, 61(6):1327–1361, 1995.

[11] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, March 1997.

[12] Th. Fraichard and Ph. Garnier. Fuzzy control to drive car-like vehicles. *Robotics and Autonomous Systems*, 34(1):1–22, December 2000.

[13] J. Hermosillo, C. Pradalier, and S. Sekhavat. Modelling odometry and uncertainty propagation for a bi-steerable car. In *Proc. of the IEEE Intelligent Vehicle Symp.*, Versailles (FR), June 2002. Poster session.

[14] J. Hermosillo and S. Sekhavat. Feedback control of a bi-steerable car using flatness; application to trajectory tracking. In *Proc. of the American Control Conference*, Denver, CO (US), June 2003.

[15] Lawrence A. Klein. *Sensor Data Fusion Concepts and Applications*. SPIE, 1993.

[16] F. Lamiraux, S. Sekhavat, and J.-P. Laumond. Motion planning and control for hilare pulling a trailer. *IEEE Trans. Robotics and Automation*, 15(4):640–652, August 1999.

[17] F. Large, S. Sekhavat, Z. Shiller, and C. Laugier. Towards real-time global motion planning in a dynamic environment using the NLVO concept. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Lausanne (CH), September-October 2002.

[18] Ch. Laugier, S. Sekhavat, L. Large, J. Hermosillo, and Z. Shiller. Some steps towards autonomous cars. In *Proc. of the IFAC Symp. on Intelligent Autonomous Vehicles*, pages 10–18, Sapporo (JP), September 2001.

[19] O. Lebeltel, P. Bessière, J. Diard, and E. Mazer. Bayesian robots progamming. *Autonomous Robots*, 2003. In Press.

[20] O. Lefebvre, F. Lamiraux, C. Pradalier, and Th. Fraichard. Obstacles avoidance for car-like robots. integration and experimentation on two robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, New Orleans, LA (US), April 2004.

[21] P. Newman. *On the structures and solution of simultaneous localization and mapping problem*. PhD thesis, Australian Center for Field Robotics,Sidney, 1999.

[22] G. Oriolo, G. Ulivi, and M. Vendittelli. *Applications of Fuzzy Logic: Towards High Machine Intelligent Quotient Systems*. Prentice-Hall, 1997.

[23] C. Pradalier, F. Colas, and P. Bessiere. Expressing bayesian fusion as a product of distributions: Applications in robotics. In *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, 2003.

[24] C. Pradalier and S. Sekhavat. Concurrent localization, matching and map building using invariant features. In *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, 2002.

[25] C. Samson and K. Ait-Abderrahim. Feedback stabilization of a nonholonomic wheeled mobile robot. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1242–1246, Osaka (JP), November 1991.

[26] S. Sekhavat, J. Hermosillo, and P. Rouchon. Motion planning for a bi-steerable car. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 3294–3299, Seoul (KR), May 2001.

[27] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 3375–3382, Minneapolis, MN (US), April 1996.

[28] I. Ulrich and J. Borenstein. VFH+: Reliable obstacle avoidance for fast mobile robots. In *IEEE Int. Conf. Robotics Automation*, pages 1572–1577, Leuven, Belgium, May 1998.

[29] I. Ulrich and J. Borenstein. VFH*: Local obstacle avoidance with look-ahead verification. In *IEEE Int. Conf. Robotics Automation*, pages 2505–2511, San Francisco, CA, April 2000.

Teaching Bayesian Behaviours to Video Game Characters

Ronan Le Hy[a*] Anthony Arrigoni[a]Pierre Bessière[a]Olivier Lebeltel[a]

[a]GRAVIR/IMAG
INRIA Rhône-Alpes, ZIRST
38330 Montbonnot, France

This article explores an application of Bayesian Programming to behaviours for synthetic video games characters. We address the problem of real-time reactive selection of elementary behaviours for an agent playing a first person shooter game. We show how Bayesian Programming can lead to condensed and easier formalisation of finite state machine-like behaviour selection, and lend itself to learning by imitation, in a fully transparent way for the player.

1. Introduction

Today's video games feature synthetic characters involved in complex interactions with human players. A synthetic character may have one of many different roles: tactical enemy, partner for the human, strategic opponent, simple unit amongst many, commenter... In all of these cases, the game developer's ultimate objective is for the synthetic character to act like a human player.

We are interested in a particular type of synthetic character, which we call a *bot* in the rest of this article. It is a player for a first person shooter game named *Unreal Tournament* augmented with the Gamebots control framework [1] (see figure 1). This framework provides a tridimensional environment in which players have to fight each other, taking advantage of resources such as weapons and health bonuses available in the arena. We believe that this kind of computer game provides a challenging ground for the development of human-level AI.

After listing our practical objectives, we will present our bayesian model. We will show how we use it to specify by hand a behaviour, and how we use it to learn a behaviour. We will tackle learning by example using a high-level interface, and then the natural controls of the game. We will show that it is possible to map the player's actions onto bot states, and use this reconstruction to learn our model. Finally, we will come back to our objectives as a conclusion.

Figure 1. Unreal Tournament and the Gamebots environment.

1.1. Objectives

Our core objective is to propose an efficient way to specify a behaviour for our bot. This can be broken down into several criteria that hold either for the developers or for the player.

*This work was partially funded by a grant from the french Ministry of Research, the BIBA project (funded by the Europen Commission), and the CNRS ROBEA project *Modèles Bayésiens pour la Génération de Mouvement*.

1.1.1. Development Team's Viewpoint

Programming efficiency. One crucial concern for the programmer is productivity: he needs both expressivity and simplicity of the behaviour programming system.

Limited computation requirements. The processing time alloted to AI in games is typically between 10% and 20% of the total processing time [2]; therefore it is important for the behaviour system to be light in terms of computation time.

Design / development separation. The industrial development scheme often draws a separation between game designers and software developers. The system should allow the designers to describe behaviours at a high conceptual level, without any knowledge of the engine's internals.

Behaviour tunability. The ability to program a variety of different behaviours, and to adjust each of them without having to modify the system's back end is essential to the designer.

1.1.2. Player's Viewpoint

"Humanness". As defined by Laird [3], this implies the illusion of spatial reasoning, memory, common sense reasoning, using goals, tactics, planning, communication and coordination, adaptation, unpredictability... One important criterion for the player is that the synthetic character does not cheat; its perceptions and actions should be as much as possible like a human player's.

Behaviour learning. This feature is gradually finding its place in modern games: the player can adjust its synthetic partners' behaviour. The behaviour system should therefore support learning.

The game industry mostly addresses these goals with specialised scripting systems, powerful but leading to behaviours hard to extend, maintain and learn [4]. More integrated systems are envisioned in the form of specialised inference engines or expert systems [5], but their actual use in the industry remains limited, as they seem hard to control or because of high computational costs. Flocking [6] is a popular way to yield an impression of complexity while being based on simple elementary behaviours; however it can hardly be generalised to any kind of behaviour.

Neural networks have also found their way to mainstream video games [7], and provide a promising alternative to scripted systems, well suited to learning, although the tuning of produced behaviours can be challenging. Decision trees have also been successfuly used [8] as a way to implement fully supervised and reinforced learning.

Nevertheless, finite state machines remain, in various incarnations [9,10] the most common formalisation for reactive behaviours – they are easily mastered, and combined with other techniques such as planification; however they suffer from combinatorial explosion, and remain hard to learn.

1.2. Technical Framework

As mentioned earlier, we used the Gamebots framework to conduct our experiments. This implies that our bot communicates with Unreal Tournament via a text protocol on a Unix socket. It receives messages covering its perceptions: its position and speed, health level, ammunition, visible opponents and objects, etc. In return, it sends actions: move to a given point, rotate, change weapon...

The environment is perceived by the bot as a graph, of which nodes are characteristic points of the topology and various objects. The bot perceives only what is in its field of vision.

As our objectives and framework have been exposed, we shall now proceed to explicit our model of behaviour selection, and discuss its interest for the specification and learning of behaviours.

2. Bayesian Model

Before examining our particular bot model, we review in the next section the principles of Bayesian Programming [11].

2.1. Bayesian Programming

Rational reasoning with incomplete and uncertain information is quite a challenge. Bayesian Programming addresses this challenge, and relies upon a well established formal theory: the probability theory [12]. As a modelling tool, it encompasses the framework of Bayesian Networks [13].

$$Program \begin{cases} Description & \begin{cases} Relevant\ Variables \\ Decomposition \\ Parametric\ Forms \end{cases} \\ Question \end{cases}$$

Figure 2. Structure of a Bayesian Program

In our framework, a Bayesian Program is made of two parts: a *description* and a *question*.

The description can be viewed as a knowledge base containing the a priori information available about the problem at hand. It is essentially a joint probability distribution. The description is made up of three components: 1) A set of *relevant variables* on which the joint distribution is defined. Typically, variables are motor, sensory or internal. 2) A *decomposition* of the joint distribution as a product of simpler terms. It is obtained by applying Bayes theorem and taking advantage of the conditional independencies that may exist between variables. 3) The *parametric forms* assigned to each of the terms appearing in the decomposition (they are required to compute the joint distribution). These are called *parametric* because they can include parameters that may change (*i.e.* be learned or adjusted) during the life of the model.

Given a distribution, it is possible to ask *questions*. Questions are obtained first by partitioning the set of variables into three sets: (1) *Searched*: the searched variables, (2) *Known*: the known variables, and (3) *Free*: the free variables (variables neither searched nor known for the particular question, but participating in the model). A question is then defined as the distribution:

$$P(Searched \mid Known) \tag{1}$$

Given the description, it is always possible to answer a question, *i.e.* to compute the probability distribution $P(Searched \mid Known)$. To do so, the following general inference rule is used:

$$\begin{aligned} & P(Searched \mid Known) \\ &= \frac{\sum_{Free} P(Searched\ Free\ Known)}{P(Known)} \\ &= \frac{1}{Z} \times \sum_{Free} P(Searched\ Free\ Known) \end{aligned} \tag{2}$$

where Z is a normalisation term.

As such, the inference is computationally expensive (Bayesian inference in general has been shown to be NP-Hard). A symbolic simplification phase can reduce drastically the number of sums necessary to compute a given distribution. However the decomposition of the preliminary knowledge, which expresses the conditional independencies of variables, still plays a crucial role in keeping the computation tractable.

2.2. Modelling our Bot

2.2.1. Bayesian Program

Our particular bot behaviour uses the following bayesian program.

Relevant Variables

S_t: the bot's state at time t. One of *Attack*, *SearchWeapon*, *SearchHealth*, *Explore*, *Flee*, *DetectDanger*. These states correspond to elementary behaviours, in our example programmed in a classic procedural fashion.
S_{t+1}: the bot's state at time $t+1$.
H: the bot's health level at t.
W: the bot's weapon at t.
OW: the opponent's weapon at t.
HN: indicates whether a noise has been heard recently at t.
NE: the number of close enemies at t.
PW: indicates whether a weapon is close at t.
PH: indicate whether a health pack is close at t.

The elementary motor commands of the bot are the values of variables S_{t+1} and S_t. They include an attack behaviour, in which the bot shoots at an opponent while maintaining a constant distance to him and strafing; a fleeing behaviour,

which consists in trying to escape (locally) an opponent; behaviours to fetch a weapon or a health bonus the bot noticed in its environment; a behaviour to detect possible opponents outside the current field of view of the bot; and a behaviour to navigate around the environment and discover unexplored parts of it.

Decomposition

The joint distribution is decomposed as:

$$\begin{aligned} & P(S_t\ S_{t+1}\ H\ W\ OW\ HN\ NE\ PW\ PH) \\ = \quad & P(S_t) \\ & P(S_{t+1}|S_t) \\ & P(H|S_{t+1}) \\ & P(W|S_{t+1}) \\ & P(OW|S_{t+1}) \\ & P(HN|S_{t+1}) \\ & P(NE|S_{t+1}) \\ & P(PW|S_{t+1}) \\ & P(PH|S_{t+1}) \end{aligned}$$

To write the above, we make the hypothesis that knowing S_{t+1}, any sensory variable is independent to each other sensory variable (which makes our model a Hidden Markov Model where observations are independant knowing the state). Although it may seem to reduce the expressivity of our model, it allows to specify it in a very condensed way; this point will be emphasised upon in section 2.2.2.

Parametric Forms

- $P(S_t)$: uniform
- $P(S_{t+1}|S_t)$: table (this table will be defined in section 2.2.2)
- $P(Sensor|S_{t+1})$ with *Sensor* each of the sensory variables: tables

Identification

The tables we use are probability distribution tables describing Laplace laws, whose parameters can be adjusted by hand, or using experimental data. We describe these two processes in sections 3 (*Specifying a Behaviour*) and 4 (*Learning a Behaviour*).

Question

Every time our bot has to take a decision, the question we ask to our model is[2]:

$$\begin{aligned} & P(S_{t+1}|S_t\ H\ W\ OW\ HN\ NE\ PW\ PH) \\ = \quad & \frac{P(S_{t+1}\ S_t\ H\ W\ OW\ HN\ NE\ PW\ PH)}{P(S_t\ H\ W\ OW\ HN\ NE\ PW\ PH)} \\ = \quad & \frac{P(S_t)\ P(S_{t+1}|S_t)\ \prod_i P(Sv_i|S_{t+1})}{\sum_{S_{t+1}} \left(P(S_t)\ P(S_{t+1}|S_t)\ \prod_i P(Sv_i|S_{t+1})\right)} \\ = \quad & \frac{P(S_{t+1}|S_t)\ \prod_i P(Sv_i|S_{t+1})}{\sum_{S_{t+1}} \left(P(S_{t+1}|S_t)\ \prod_i P(Sv_i|S_{t+1})\right)} \end{aligned}$$

Knowing the current state and the values of the sensors, we want to know the new state the bot should switch into. This question leads a probability distribution, on which we draw a value to decide the actual new state. This state translates directly into an elementary behaviour which is applied to the bot.

2.2.2. Inverse Programming

We shall now emphasise the peculiarities of our method to specify behaviours, compared to one using simple finite state machines (FSMs). The problem we address is, knowing the current state and the sensors' values, to determine the next state: this is actually naturally accomplished using an FSM.

Let us consider the case where each of our n sensory variables has m_i $(1 \leq i \leq n)$ possible values.

In an FSM modelling a behaviour [14,15], we would have to specify, for each state, a transition to each state, in the form of a logical condition on the sensory variables.

It means that the programmer has to discriminate amongst the $\prod_i m_i$ possible sensory combinations to describe the state transitions. Not only does this pose the difficult problem of determining the appropriate transitions, but it raises the question of convenient formalised representation. This approach could actually lead to several implementations, but will possibly [10] result in a script resembling the following:

if S_t =*A* **and** W =*None* **and** OW =*None* **then**
  if HN =*False* **and** NE!=*None*

[2] in this equation, $(Sv_i)_i$ denotes sensory variables such as H, W...

or $NE = TwoOrMore$ **then**
$S_{t+1} \leftarrow F$
else if $HN = True$ **or** $NE = One$
and $PW = True$ **then**
$S_{t+1} \leftarrow A$
else...

This kind of script is hard to write and hard to maintain.

In contrast, our approach consists in giving, for each sensory variable, for each possible state, a distribution (i.e. m_i numbers summing to 1). In practice, we write tables like table 1, which represents $P(H|S_{t+1})$. Values of H are enumerated in the first column, those of S_{t+1} in the first line; cells marked x are computed so that each column sums to 1.

Moreover, instead of specifying the conditions that make the bot switch from one state to another, we specify the (probability distribution of the) sensors' values when the bot goes into a given state. This way of specifying a sensor under the hypothesis that we know the state is what makes us call our method "inverse programming".

Although somewhat confusing at first, this is the core advantage of our way to specify a behaviour. As a matter of fact, we have to describe separately the influence of each sensor on the bot's state, thereby reducing drastically the quantity of needed information. Furthermore, it becomes very easy to incorporate a new sensory variable into our model: it just requires to write an additional table, without modifying the existing ones.

Finally, the number of values we need in order to specify a behaviour is $s^2 + snm$, where s is the number of states, n the number of sensory variables, and m the average number of possible values for the sensory variables. It is therefore linear in the number of variables (assuming m constant).

3. Specifying a Behaviour

3.1. Basic specification

A behaviour can be specified by writing the tables corresponding to $P(S_{t+1}|S_t)$ and $P(Sensor|S_{t+1})$ (for each sensory variable). Let us consider for instance table 1, which gives the probability distribution for H knowing S_{t+1}. We read the first column this way: given the fact that the bot is going to be in state *Attack*, we know that it has a very low probability (0.001) to have a low health level, a medium probability (0.1) to have a medium health level, and a strong chance ($x = 1 - 0.001 - 0.1$) to have a high health level.

$H \setminus S_{t+1}$	A	SW	SH	Ex	F	DD
Low	0.001	0.1	x_3	0.1	0.7	0.1
Medium	0.1	x_2	0.01	x_4	0.2	x_5
High	x_1	x_2	0.001	x_4	0.1	x_5

Table 1
$P(H|S_{t+1})$

This form of specification allows us to formalise conveniently the constraints we want to impose on the behaviour, in a condensed format, and separately on each sensory variable. For instance, table 1 formalises the relation of the bot's health level to its state: if it starts attacking, then its health is rather high; if it starts searching for a health pack, then its health is very probably low; if it starts fleeing, then its health is probably rather low, but with a high degree of uncertainty.

All tables on the sensory variables are built on the same pattern; the one giving $P(S_{t+1}|S_t)$ (see table 2) is special. It gives some sort of basic transition table; i.e. it answers in a probabilistic way the question: knowing nothing but the current state, what will be the next state?

$S_{t+1} \setminus S_t$	A	SW	SH	Ex	F	DD
A	x_1	x_2	x_3	x_4	x_5	x_6
SW	10^{-5}	x_2	10^{-5}	10^{-5}	10^{-5}	10^{-5}
SH	10^{-5}	10^{-5}	x_3	10^{-5}	10^{-5}	10^{-5}
Ex	10^{-5}	10^{-5}	10^{-5}	x_4	10^{-5}	10^{-5}
F	10^{-5}	10^{-5}	10^{-5}	10^{-5}	x_5	10^{-5}
DD	10^{-5}	10^{-5}	10^{-5}	10^{-5}	10^{-5}	x_6

Table 2
$P(S_{t+1}|S_t)$

The answer our sample table gives is: tend to stay in your current state (notice the xs on the diagonal) or switch to attack (notice the xs on the first line) with the same high probability; switch

to other states with a very low probability (10^{-5} – which in our example we found to be representative of "very low").

Again, this makes a parallel with an FSM with probabilistic transitions: with our transition table $P(S_{t+1}|S_t)$, we give a basic automaton upon which we build our behaviour by fusing the tendencies given separately on each sensory variable.

3.2. Tuning the behaviour

Tuning our behaviour amounts to tuning our probability distributions. For instance, to create a *berserk* character that is insensible to its health level, we put only uniform distributions (i.e. in our notation, only xs) in table $P(H|S_{t+1})$. A *berserk* is also very aggressive, so the transition table we proposed in table 2 is quite adapted. A transition table for a more prudent character would not have those xs on the first line, so that the state A would not be particular.

To create a unique behaviour, we therefore have to review all our tables, i.e. the influence of each sensory variable on the character according to the said behaviour.

3.3. Results

Several observations can be made when out bots are playing the game. The first is that their behaviour corresponds to what we want: the behaviour switching occurs reasonably, given the evolution of the sensory variables. The second is that they can't compete with humans playing the game. Noting this allows to pinpoint the fact that our method's interest mostly resides in the gain of ease and power in the design of behaviours. It does not pretend to overcome the limitations of the elementary behaviours we are switching between, nor can it do more than what the Gamebots framework allow, in terms of perception and action. Therefore, what we aimed for, and finally obtained, is a reliable, practical and efficient way to specify the real-time selection of elementary behaviours.

Our attempt to tune the behaviour shows that the differences between our 'reasonable' bot and our 'aggressive' bot are visible, and correspond to what we tried to specify in the tables. For instance, the aggressive bot is excited by the presence of several opponents, whereas this situation repels the reasonable bot; and the aggressive bot is not discouraged to attack when its health level goes low.

4. Learning a Behaviour

Our goal now is to teach the bot a behaviour, instead of specifying all the probability distributions by hand. It requires to be able to measure at each instant sensory and motor variables of the controlled bot. In particular, it is necessary to determine the state S_t at each instant. It can be done by letting the player specify it directly in real time, or by inferring it from his natural actions in the game.

4.1. Selecting behaviours

This form of learning by example presents a simple interface to the player, shown on figure 3.

Figure 3. Interface used to teach the bot: on the right is the normal Unreal Tournament window showing our bot; on the left is our interface to control the bot.

The player controls in real time the elementary behaviour that the bot executes, by using buttons that allow switching to any state with a mouse click. In addition to the ordinary Unreal Tournament window on the right, part of the internal state of the bot is summed up in the learning interface on the left.

4.2. Recognising behaviours

While the previous method of teaching a behaviour works, it deprives the player of the in-

terface he is used to; his perceptions and motor capabilities are mostly adjusted to the bot's. In order to solve this problem, it is possible to give the player the natural interface of the game, and try to recognise in real time the behaviour he is following.

To recognise the human's behaviour from his low-level actions, we use a heuristic programmed in a classical imperative fashion. It involves identifying each behaviour's critical variables (for instance, attack is characterised by distance and speed of the bot to characters in the centre of his field of view), and triggering recognition at several timescales.

Recognition is done by examining a series of criteria in sequence; the first that matches is chosen. The first criterion is a characteristic event which is back-propagated to states in the past not yet recognised (for instance picking a health bonus indicates that the character has been looking for health). The second examines critical variables over a fixed period (for instance, danger checking is characterized by a complete rotation with little translation, in a short amount of time). Finally, some short-term variations of critical variables are examined (like attacking and fleeing, which consist in moving in a particular direction in the presence of opponents). Exploration is a default state, when a state does not match any of the criteria.

We do this recognition off-line, on data representing ten to fifteen minutes of game-play; processing this data and producing the tables that represent our behaviour takes five to ten seconds.

4.3. Results

recognition learned, aggressive	4.4
recognition learned, cautious	13.9
selection learned, aggressive	45.7
manual specification, aggressive	8.0
manual specification, cautious	12.2
manual specification, uniform	43.2
native (level 3/8) UT bot	11.0

Table 3
Performance comparison on learned, hand-specified, and native Unreal bots (lower is better)

Table 3 shows a comparison between different specification methods. Numbers are the average point difference to the winning bot, over ten games won by the first bot reaching 100 kills (for example, a bot with 76 points at the end of a game has a point difference of 24 to the winning bot, since the game ends as soon as a bot reaches 100 points). Therefore, a bot winning all games would have a score of zero. Our bots compare well to the native *Unreal Tournament* bot, whose skill corresponds to an average human player. Aggressive bots (gray lines) perform significantly better, and learning by recognition does much better than learning by selection, along with hand specification.

Lessons from these results can be summed up in the following way (we will refer here to table 4, which is the same as table 1, but learnt by recognition):

1. learnt tables share common aspects with hand-written tables (as for the transition table $P(S_{t+1}|S_t)$); for instance,in the fleeing state F, health level $\ddot{r}$ is much more probably low or medium than high;

2. differences in behaviour of the teacher influence the learnt behaviour: aggressivity (or the lack of it) is found in the learnt behaviour, and translates into performance variations (in our setup, aggressive behaviours seem to be more successful);

3. nevertheless, differences between hand-specified and learnt models are noticeable; they can explained by:

   (a) player-specific behaviours: humans almost always attack and do not retreat; another example is the low probability of $P(H = High|S_{t+1} = SW)$ in the learnt table (dark gray cell on table 4): it can be explained by the fact that human players give a much higher priority to searching a good weapon over searching for health bonuses;

   (b) additional information: some parts of the hand-written tables are specified as uniform (as a result from a refusal

or impossibility to specify theoretically a link between two events, like the value of the opponent's weapon knowing that the bot is exploring), whereas their learnt counterparts include information;

(c) perceptive differences: a human player and a bot have a different perception of sound (the human perceives direction combined with the origin of sound, like an impact on a wall or the sound of the shooting itself, whereas the bot senses only direction);

(d) bias induced by data quantity: a human player has almost always a medium health level (which is due to a poor choice of discretisation for the health level variable), which explains higher values in the learnt table 4 (line of light gray cells). The discretisation is subject to the following constraints: its being too fine makes specifying behaviours by hand harder; it also slows down inference, and increase the amount of data necessary to learn the model; on the other hand, a coarse discretisation impedes the expressiveness of the model, by coagulating sensory states too different from one another. A rule to choose discretisation is therefore to take a (possibly non-linear) scale where real values from two successive steps are deemed qualitatively different by a human player. For instance, the health level could be better split in the following way: *very low* beneath 70, *low* between 71 and 95, *medium* between 96 and 110, *high* between 111 and 130, *very high* above 131.

4. our learning methods lead to functioning behaviours; learning using behaviour recognition scores best, and allows to reach the level of an average native UT bot.

$H \setminus S_{t+1}$	A	SW	SH	Ex	F	DD
Low	0.179	0.342	0.307	0.191	0.457	0.033
Medium	0.478	0.647	0.508	0.486	0.395	0.933
High	0.343	0.011	0.185	0.323	0.148	0.033

Table 4
learnt $P(H|S_{t+1})$

5. Discussion

5.1. Evaluation

We shall now come back to the objectives we listed at the beginning, to try and assess our method in practical terms.

5.1.1. Development Team's Viewpoint

Programming efficiency. Our method of behaviour design relies upon a clear theoretical ground. Programming the basic model can use a generic bayesian programming library, and needs afterwards little more than the translation into C++ (for instance) of the mathematical model. Design is really expressed in terms of practical questions to the expertise of the designer, like "if the bot is attacking, how high is his health level?"; it does not require a preliminary formalisation of the expected behaviour to program. Moreover, in our model behaviours are data (our tables). It means that they can easily be loaded and saved while the behaviour is running, or exchanged amongst a community of players or developers.

Limited computation requirements. The computation time needed for a decision under our model can be shown to be linear in both the number of sensory variables and the number of states.

Design / Development separation. Development amounts to incorporating the bayesian framework into the control architecture, and establishing the bayesian model; design consists in establishing relations between the variables in the form of probability distributions. A designer really has to know about what the bot should do, but does not need any knowledge of the implementation details; he needs but a light

background on probabilities, and no scripting or programming at all.

Behaviour tunability. We have seen that our way of specifying behaviours gives a natural way to formalise human expertise about behaviours, and that it implies that tuning a behaviour is possible, as they are expressed in natural terms and not in artificial logical or scripting terms. Moreover, the quantity of data needed to describe a behaviour is kept small compared to an FSM, and this helps keeping the analysis and control of a behaviour tractable for the designer.

5.1.2. Player's Viewpoint

"Humanness". This criterion is hard to assess, although it can be done [16] in ways comparable to the Turing test [17]. Our method of specifying a behaviour helps the designer translate his expertise easily, and therefore gives him a chance to build a believable bot.

Behaviour learning. We have seen that learning under our model is natural: it amounts to measuring frequencies. This is a chance for the player to teach its teammate bots how to play. Recognising high-level states on the basis of low-level commands is possible, and allows a player to adjust a behaviour completely transparently, with the original controls of the game.

5.2. Perspectives

We have shown a way to specify FSM-like action selection models for virtual robots, and to learn these models by example. The recognition involved in learning from the natural actions of a player in the game remains a classicaly programmed heuristic; an obvious perspective is to formalise it within the bayesian framework, in order to perform probabilistic behaviour recognition. This would grant more adaptability to variations in the behaviour model.

REFERENCES

1. G. Kaminka, et al., Gamebots: the ever-challenging multi-agent research test-bed, Communications of the ACM.
2. S. Woodcock, Game AI : The state of the industry 2000-2001, Game Developer.
3. J. Laird, Design goals for autonomous synthetic characters, draft (2000).
4. Baldur's gate scripting, Site web, http://www25.brinkster.com/iesdp/.
5. J. E. Laird, It knows what you're going to do : Adding anticipation to a quakebot, in: AAAI Spring Symposium Technical Report, 2000.
6. S. Woodcock, Flocking with teeth : Predators and prey, in: M. Deloura (Ed.), Game Programming Gems 2, Charles River Media, 2001, pp. 330–336.
7. Neural Network AI for Colin McRae Rally 2.0, Website (Generation5), `www.generation5.org/content/2001/hannan.asp`.
8. Artificial intelligence: Black and white, Website (Gamespot), `www.gamespot.com/gamespot/features/pc/hitech/p2_01.html`.
9. E. Dysband, A generic fuzzy state machine in C++, in: M. Deloura (Ed.), Game Programming Gems 2, Charles River Media, 2001, pp. 337–341.
10. Unrealscript language reference, Website, `unreal.epicgames.com/UnrealScript.htm`.
11. O. Lebeltel, P. Bessière, J. Diard, E. Mazer, Bayesian Robot Programming, Autonomous Robots.
12. E. T. Jaynes, Probability theory: the logic of science, unprinted book, available on-line at `http://bayes.wustl.edu/` (1995).
13. M. Jordan (Ed.), Learning in Graphical Models, MIT Press, 1998.
14. E. Dysband, A finite-state machine class, in: M. Deloura (Ed.), Game Programming Gems, Charles River Media, 2000, pp. 237–248.
15. M. Zarozinski, Imploding combinatorial explosion in a fuzzy system, in: M. Deloura (Ed.), Game Programming Gems 2, Charles River Media, 2001, pp. 342–350.
16. J. E. Laird, J. C. Duchi, Creating human-like synthetic characters with multiple skill-levels : A case study using the Soar quakebot, in: AAAI Fall Symposium Technical Report, 2000.
17. A. M. Turing, Computing machinery and intelligence, Mind 59 (236) (1950) 433–460.

Journal of Artificial Intelligence Research 9 (1998) 295-316 Submitted 9/97; published 11/98

The Ariadne's Clew Algorithm

Emmanuel Mazer EMMANUEL.MAZER@IMAG.FR
Laboratoire GRAVIR,
INRIA 665 Avenue de L'Europe
38330 Montbonnot, France

Juan Manuel Ahuactzin JMAL@MAIL.UDLAP.MX
Depto. de Ing. en Sistemas Computationales
Unversidad de las Americas
Puebla, Cholula, Puebla 72820, Mexico

Pierre Bessière PIERRE.BESSIERE@IMAG.FR
Laboratoire LEIBNIZ,
46 Avenue Felix Viallet
38000 Grenoble, France

Abstract

We present a new approach to path planning, called the "Ariadne's clew algorithm". It is designed to find paths in high-dimensional continuous spaces and applies to robots with many degrees of freedom in static, as well as dynamic environments — ones where obstacles may move. The Ariadne's clew algorithm comprises two sub-algorithms, called SEARCH and EXPLORE, applied in an interleaved manner. EXPLORE builds a representation of the accessible space while SEARCH looks for the target. Both are posed as optimization problems. We describe a real implementation of the algorithm to plan paths for a six degrees of freedom arm in a dynamic environment where another six degrees of freedom arm is used as a moving obstacle. Experimental results show that a path is found in about one second without any pre-processing.

1. Introduction

The path planning problem is of major interest for industrial robotics. A basic version of this problem consists of finding a sequence of motions for a robot from a start configuration to a given goal configuration while avoiding collisions with any obstacles in the environment.

A simple version of the problem, that of planning the motion of a point robot among 3-dimensional polyhedral obstacles, has been proved to be NP-complete (Canny, 1988). Generally speaking, the complexity of the problem is exponential in the number of degrees of freedom (DOF) of the robot, and polynomial in the number of obstacles in the environment. Consequently, finding a path for a robot with many DOF (more than five) in an environment with several obstacles is, indeed, a very difficult problem. Unfortunately, many realistic industrial problems deal with robots of at least six DOF and hundreds of obstacles. Even worse, often the environment is dynamic in the sense that some of the obstacles may move, thereby further requiring that new paths be found in very short computing times.

In this paper, we present a new approach to path planning, called the "Ariadne's clew algorithm [1]". The approach is completely general and applies to a broad range of path planning problems. However, it is particularly designed to find paths for robots with many DOF in dynamic environments.

The ultimate goal of a planner is to find a path from the initial position to the target. However, while searching for this path, the algorithm may consider collecting information about the free space and about the set of possible paths that lie in that free space. The Ariadne's clew algorithm tries to do both at the same time: a sub-algorithm called EXPLORE collects information about the free space with increasingly fine resolution, while, in parallel, an algorithm called SEARCH opportunistically checks whether the target can be reached.

The EXPLORE algorithm works by placing landmarks in the searched space in such a way that a path from the initial position to any landmark is known. In order to learn as much as possible about the free space, the EXPLORE algorithm tries to spread the landmarks uniformly all over the space. To do this, it places the landmarks as far as possible from one another. For each new landmark produced by the EXPLORE algorithm, the SEARCH algorithm checks (with a local method) whether the target can be reached from that landmark. Both the EXPLORE and SEARCH algorithms are posed as optimization problems.

The Ariadne's clew algorithm is *efficient* and *general*:

1. The algorithm is efficient in two senses:

   (a) Experiments show that the algorithm is able to solve path planning problems fast enough to move a six DOF arm in a realistic and dynamic environment where another six DOF robot is used as a moving obstacle.

   (b) It is well suited for parallel implementation and shows significant speed-up when the number of processors increases.

2. The algorithm is general in two senses:

   (a) It may be used for a wide range of applications in robotics with little additional effort to adapt it.

   (b) Finally, the algorithm is general in that it may be adapted for a large range of search problems in continuous spaces that arise in fields that are not related to robotics.

The paper is organized as follows. Section 2 presents the path planning problem and discusses related work. Section 3 presents the principle of the Ariadne's clew algorithm. Section 4 describes the application of the algorithm to a six DOF arm in a dynamic environment. Finally, Section 5 concludes the paper with a discussion of the contributions of our approach, the main difficulties involved, and possible improvements of our method.

1. According to Greek legend, Ariadne, the daughter of Minos, King of Crete, helped Theseus kill the Minotaur, a monster living in the Labyrinth, a huge maze built by Daedalus. The main difficulty faced by Theseus was to find his way through the Labyrinth. Ariadne brilliantly solved the problem by giving him a thread (or a clew) that he could unwind in order to find his path back.

2. The Path Planning Problem

Many versions of the path planning problem exist. An exhaustive classification of these problems and of the methods developed to solve them can be found in a survey by Hwang and Ahuja (1992). We choose to illustrate our discussion with a particular case. A robot arm is placed among a set of obstacles. Given an initial and a final position of the robot arm, the problem is to find a set of motions that will lead the robot to move between the two positions without colliding with the obstacles.

To drive the robot amidst the obstacles, early methods (Brooks, 1983) directly used the 3D CAD models of the robot and of the obstacles to find a solution, i.e., they considered the "operational 3D space". In this space, the path planning problem consists of finding the movements of a complex 3D structure (the robot) in a cluttered 3D space.

A major advance was to express the problem in another space known as the configuration space, denoted by $\mathcal{C}$ (Lozano-Pérez, 1987). In this space, the position (or configuration) of a robot is completely determined by a single point having n independent parameters as coordinates. The positions that are not physically legal (because of a collision) are represented by particular regions of $\mathcal{C}$, and are called *$\mathcal{C}$-obstacles*. In the configuration space, the path planning problem consists of finding a continuous curve (representing a path for a single geometrical point) that (i) connects the points representing the initial and the final configuration of the robot, and (ii) does not intersect any *$\mathcal{C}$-obstacles*. This method trades a simplification of the path planning problem (it searches a path for a single point) against a higher-dimensional search space (the dimension of $\mathcal{C}$ is the number DOF of the robot) and against more complex shapes of obstacles (very simple physical obstacles may result in very complex *$\mathcal{C}$-obstacles*).

For example, let us consider the planar arm of Figure 1. Its position among the obstacles is totally known once the values of the angles between its links (q_0, q_1) are known. Thus, for each pair (q_0, q_1), it is possible to decide whether the robot collides with the surrounding obstacles. This is what we did in Figure 2 to represent the mapping between the physical obstacles in the operational space and the *$\mathcal{C}$-obstacles*. Now, by moving a point along the curve joining $\hat{q}_a$ and $\hat{q}_f$ one will also define a collision-free motion for the planar arm between the corresponding positions $P(\hat{q}_a)$ and $P(\hat{q}_f)$ in the operational space. This curve is one solution to this particular path planning problem.

A recent trend in the field is to consider the "trajectory space" (Ferbach, 1996) where a whole path is represented by a single point. The coordinates of this point are the values of the parameters defining the successive movements of the robot. For instance, the list of successive commands sent to the robot controller indeed encode an entire path of the robot. In this space, the path planning problem is reduced to the search for a single point. Once again, we trade a simplification of the path planning problem (searching for a point) against a higher dimension of the search space (the dimension of the trajectory space is the number of parameters needed to specify completely a whole path). For example, in Figure 2, the path between $\hat{q}_b$ and $\hat{q}_d$ can be represented by a point in a seven-dimensional space simply by considering the length of its seven segments.

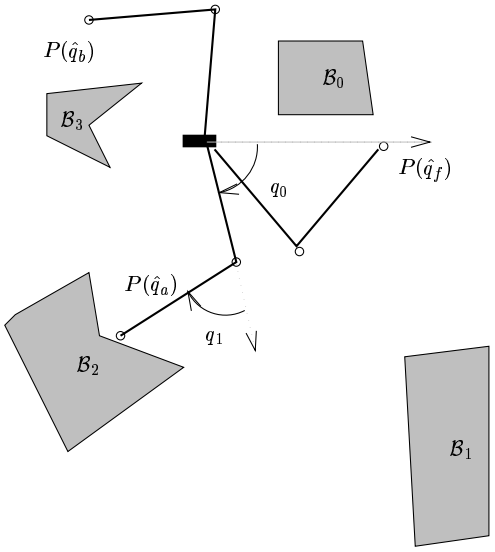


Figure 1: A two DOF arm placed among obstacles in the operational space

2.1 Global Approaches

Global approaches are classically divided into two main classes: (i) retraction methods, and (ii) decomposition methods. In the retraction methods, one tries to reduce the dimension of the initial problem by recursively considering sub-manifolds of the configuration space. In the decomposition methods, one tries to characterize the regions of the configuration space that are free of obstacles. Both methods end up with a classical graph search over a discrete space. In principle, these methods are *complete* because they will find a path if one exists and will still terminate in a finite time if a path does not exist. Unfortunately, computing the retraction or the decomposition graph is an NP-complete problem: the complexity of this task grows exponentially as the number of DOF increases (Canny, 1988). Consequently, these planners are used only for robots having a limited number (three or four) of DOF. In addition, they are slow and can only be used off-line: the planner is invoked with a model of the environment, it produces a plan that is passed to the robot controller which, in turn, executes it. In general, the time necessary to achieve this is not short enough to allow the robot to move in a dynamic environment.

2.2 Path Planning with Local Planners

One way to combat the complexity of the problem is to trade completeness against performance. To do this, the local planners are guided by the gradient of a cost function (usually the Euclidean distance to the goal) and take into account the constraints introduced by the obstacles to avoid them (Faverjon & Tournassoud, 1987). Since the path planning problem is NP-complete, knowing the cost function, it is always possible to design a deceptive environment where the method will get trapped in a local minimum. However, these methods are useful in many industrial applications because they can deal with complex robots and

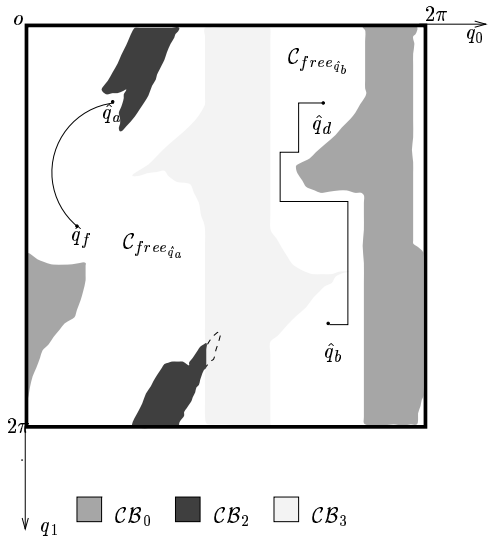


Figure 2: The configuration space corresponding to Figure 1. Note: (1) $\mathcal{C}$ is a torus, (2) it is divided into two regions $\mathcal{C}_{free_{\hat{q}_a}}$ and $\mathcal{C}_{free_{\hat{q}_b}}$ that cannot be connected by a continuous path, and (3) there is not a $\mathcal{C}$-obstacle for $\mathcal{B}_1$ because it does not interfere with the arm.

environment models having thousands of faces, that are often too time-consuming for global methods.

2.3 Path Planning with Randomized Techniques and Landmarks

The stochastic or random approach was first introduced by Barraquand and Latombe (1990), and later used by Overmars (1992), and more recently by Kavraki (1996). The main idea behind these algorithms is to build a graph in the configuration space. The graph is obtained incrementally as follows: a local planner is used to try to reach the goal. Should the motion stop at a local minimum, a new node (or landmark) is created by generating a random motion starting from that local minimum. The method iterates these two steps until the goal configuration has been reached from one of these intermediary positions by a gradient descent motion. These algorithms work with a discretized representation of the configuration space. They are known to be *probabilistically complete* because the probability of terminating with a solution (a path has been found or no path exists) converges to one as the allowed time increase towards infinity. As in the previous section, it is also possible to design simple deceptive environments that will make this kind of algorithm slower than a pure random approach. However, they have been tested for robots with a high number of DOF and they have been shown to work quickly in relatively complex and natural environments.

Other methods using landmarks have been devised. For example, SANDROS, introduced by Chen and Hwang(1992), makes use of landmarks to approximate the free space. This approach is similar to the "hierarchical planning" approach used in AI: should the method

fail to reach a goal, new subgoals are generated until the problem is easy enough to be solved. In their approach, first a local planner is used to reach the final position: should the local planner fail, the configuration space is divided into two subspaces, one containing the goal and the other a new sub-goal. The problem is therefore divided into two sub-problems: (i) going from the initial position to the subgoal, and (ii) going from the subgoal to the final position. SANDROS has been shown to be particularly well adapted to find paths for manipulators. It has been implemented and tested for planning paths for Puma and Adept robots.

2.4 Path Planning in the Trajectory Space

The previous methods were essentially based on the configuration space: the retraction, the decomposition, or the optimization is made in this space. An alternative is to consider the "trajectory space". For example, in his method VDP, Ferbach (1996) starts by considering the straight line segment joining the initial and the final configuration in $\mathcal{C}$. This path is progressively modified in such a manner that the forbidden regions it crosses are reduced. At each iteration, a sub-manifold of $\mathcal{C}$ containing the current path is randomly generated. It is then discretized and explored using a dynamic programming method that uses the length across the forbidden region as the cost function in order to minimize. The search results in a new trajectory whose intersection with the forbidden regions is smaller than the original trajectory. The process is repeated until an admissible trajectory is found. As in the previous sections, it is also possible to design simple deceptive environments that will make this kind of algorithm slower than a pure random approach.

The work of Lin, Xiao, and Michalewicz (1994) is similar to our approach. As in an early version of our algorithm (Ahuactzin, Mazer, Bessière, & Talbi, 1992), genetic algorithms are used to carry out optimization in the trajectory space. Trajectories are parameterized using the coordinates of intermediary via-points. An evolutionary algorithm is used to optimize a cost function based on the length of the trajectory and the forbidden region crossed. The standard operators of the genetic algorithms have been modified and later extended to produce a large variety of paths (Xiao, Michalewicz, & Zhang, 1996). The number of intermediary via-points is fixed and chosen using an heuristic. Given this number, nothing prevents to design a deceptive problem which solution will require more intermediary points, leading the algorithm to fail while one solution exists.

3. Principle of the Ariadne's Clew Algorithm

As we have seen in the previous section, the computation of the configuration space $\mathcal{C}$ is a very time-consuming task. The main idea behind the Ariadne's clew algorithm is to avoid this computation. In order to do this, the algorithm searches directly for a feasible path in the trajectory space. The configuration space $\mathcal{C}$ is never explicitly computed.

As will be shown, in the trajectory space, path planning may be seen as an optimization problem and solved as such by an algorithm called SEARCH. It is possible to build an approximation of free space by another algorithm called EXPLORE that is also posed as an optimization problem. The Ariadne's clew algorithm is the result of the interleaved execution of SEARCH and EXPLORE.

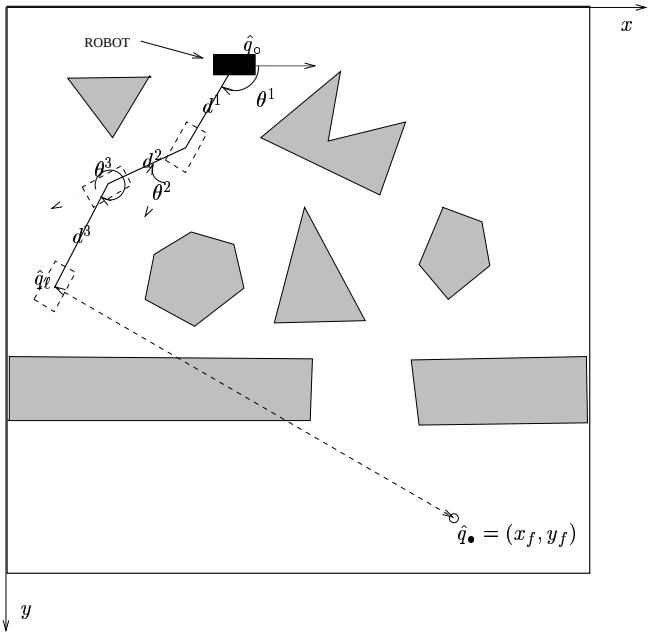


Figure 3: A parameterized trajectory $(\theta^1, d^1, \theta^2, d^2, ...\theta^l, d^l)$ and a starting point $\hat{q}_0$ implicitly define a path (in the operational space) for a holonomic mobile robot.

3.1 Path Planning as an Optimization Problem: SEARCH

Given a robot with k DOF, a trajectory of length l may be parameterized as a sequence of $n = k * l$ successive movements. A starting point $\hat{q}_\circ$ along with such a parameterized trajectory implicitly define a path and a final configuration $\hat{q}_l$ in the configuration space. For example, for a holonomic mobile robot the trajectory $(\theta^1, d^1, \theta^2, d^2, ...\theta^l, d^l)$ can be interpreted as making a θ^1 degree turn, moving straight d^1, making a θ^2 degree turn and so on. Given the starting configuration $\hat{q}_\circ$, this trajectory leads to the final configuration $\hat{q}_l$ (see Figure 3).

Given a distance function d on the configuration space, if we find a trajectory such that it does not collide with any obstacles and such that the distance between $\hat{q}_l$ and the goal $\hat{q}_\bullet$ is zero, then we have a solution to our path planning problem. Therefore, the path planning problem may be seen as a minimization problem where:

1. The search space is a space of suitably parameterized trajectories, the trajectory space.

2. The function to minimize is $d(\hat{q}_l, \hat{q}_\bullet)$ if the path is collision-free, and $d(\hat{q}_i, \hat{q}_\bullet)$ otherwise ($\hat{q}_i$ being the first collision point).[2]

2. Another possible choice would be to give the $+\infty$ value to the distance when a collision occurs. However, this is less informative than the chosen function because the first part of a colliding path could be a good start toward the goal and should not be discarded. Note that the cost function does not include any optimality criteria such as the length of the trajectory or the amount of energy used.

The algorithm SEARCH, based on this very simple technique and a randomized optimization method, is already able to solve quite complex problems of robot motion planning. For example, Figure 4 represents the two paths found for the holonomic mobile robot. Each path was computed on a standard workstation (SPARC 5) in less than 0.5 second without using any pre-computation of the configuration space. Thus, it is possible, albeit slowly, to get a planner that can be used in a dynamic environment (where the obstacles may move) by "dropping" a new world into the system every 0.5 second. SEARCH is very efficient but it is not complete, since it may fail to find a path even if one exists for two different reasons:

1. Due to the optimization-based formulation, SEARCH can get trapped by local minima of the objective function, which in turn may place the robot far away from the goal (see Figure 5).
2. The length l of the trajectories considered may be too short to reach all the accessible regions of the configuration space.

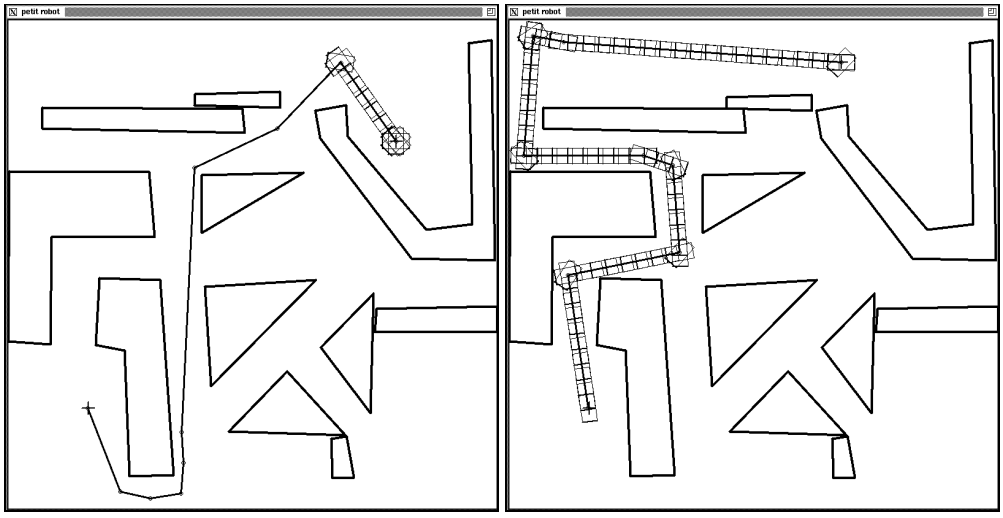


Figure 4: Reactive replanning in a changing environment

3.2 Exploring as an Optimization Problem: EXPLORE

In order to build a complete planner, we propose a second algorithm called EXPLORE. While the purpose of SEARCH was to look directly for a path from $\hat{q}_\circ$ to $\hat{q}_\bullet$, the purpose of EXPLORE is to compute an approximation of the region of the configuration space accessible from $\hat{q}_\circ$.

The EXPLORE algorithm builds an approximation of the accessible space by placing landmarks in the configuration space $\mathcal{C}$ in such a way that a path from the initial position $\hat{q}_\circ$ to any landmark is known. In order to learn as much as possible about the free space, the EXPLORE algorithm tries to spread the landmarks uniformly over the space (see Figure 6). To do this, it tries to put the landmarks as far as possible from one another by maximizing the distances between them.

Therefore, EXPLORE may be seen as a maximization problem where:

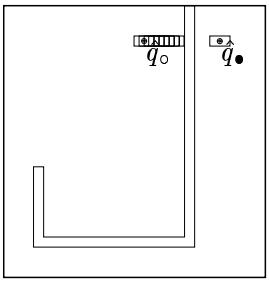


Figure 5: A problem leading to a local minimum. In such a case, a solution path has first to move away from the goal. The goal's "attraction" based on the minimization of the Euclidean distance prevents SEARCH from finding such a path.

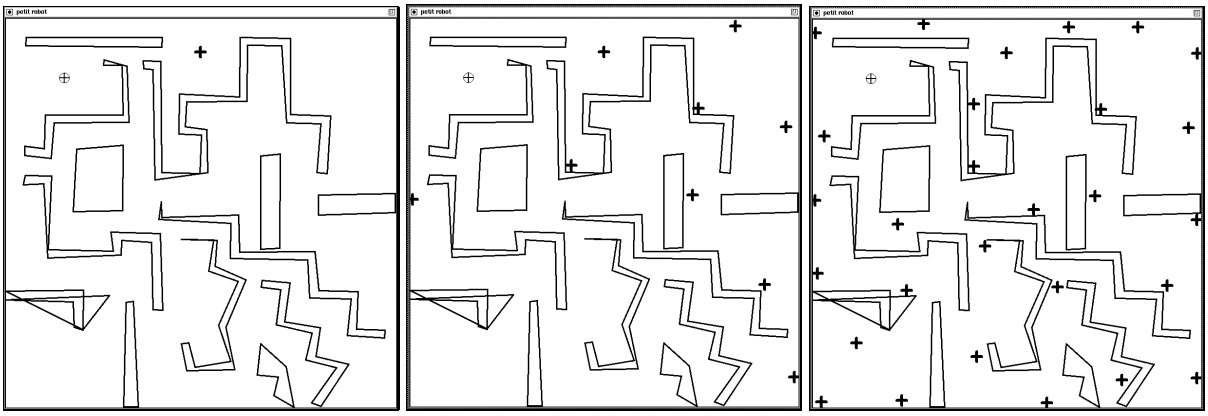


Figure 6: The first picture represents the initial position and the first landmark. The subsequent landmarks are then uniformly spread over the search space while the method keeps track of all paths joining the landmarks to the initial position. The algorithm is named after Ariadne because by placing landmarks, EXPLORE unwinds as if it were using a thread as Theseus did.

1. The search space is the set of all paths starting from one of the previously placed landmarks.

2. The function to maximize is $d(\hat{q}_l, \Lambda)$, where Λ is the set of landmarks already placed.

3.3 The Ariadne's Clew Algorithm: EXPLORE + SEARCH

In order to have a planner that is both complete and efficient, we combined the two previous algorithms SEARCH and EXPLORE to obtain the Ariadne's clew algorithm.

The principle of the Ariadne's clew algorithm is very simple:

1. Use the SEARCH algorithm to find whether a "simple" path exists between $\hat{q}_\circ$ and $\hat{q}_\bullet$.

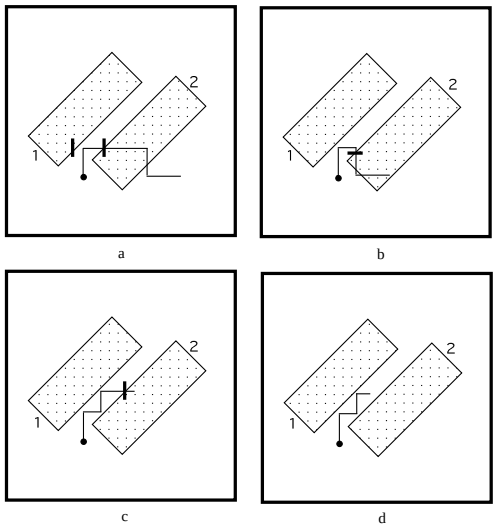


Figure 7: Bouncing against $\mathcal{C}$-obstacles. Figure (a) presents the original path in the configuration space. Figure (b) shows the same path after two bounces along the second segment on obstacle 2 and on obstacle 1. Figure (c) is the result obtained after a bounce of segment 3 against obstacle 2. Finally, Figure (d) presents a valid path obtained after a final bounce of segment 4 against obstacle 2.

2. If no "simple" path is found by step 1, then continue until a path is found.

   (a) Use EXPLORE to generate a new landmark.

   (b) Use SEARCH to look for a "simple" path from that landmark to $\hat{q}_\bullet$.

The Ariadne's clew algorithm will find a path if one exists. In an overwhelming number of cases, just a few landmarks are necessary for the Ariadne's clew algorithm to reach the target and stop.

3.4 A Major Improvement: Bouncing on $\mathcal{C}$-Obstacles

A typical difficulty for a path planning algorithm is to find a collision-free path through a small corridor in the configuration space. This is also the case for the basic version of the Ariadne's clew algorithm, presented above. The problem is that very few trajectories encode such paths and therefore they are very difficult to find. Most trajectories collide with the obstacles. We propose a very simple idea to deal with this problem: going backwards at each collision point. If, for a given trajectory, a collision is detected along the corresponding path, then we simply consider transforming that trajectory so that it encodes a new path, one that is found by bouncing off the obstacle at the collision point (see Figure 7). Note that this construction is applied recursively until the entire trajectory corresponds to a collision-free path.

Using this technique, all trajectories are so transformed that they encode valid paths. This improved version of the Ariadne's clew algorithm no longer cares about obstacles. From the point of view of a search in the trajectory space, it is as if the obstacles have simply vanished. This method is especially efficient for narrow corridors in the configuration space. Without bouncing, the mapping of a corridor in the configuration space to the trajectory space is a set of very few points. With bouncing, every single trajectory going through a part of the corridor is "folded" into the corridor (see Figure 7). The resultant mapping of the corridor in the trajectory space is consequently a much larger set of points, and therefore it is much easier to find a member of this set. This empirical improvement has a major practical impact because it makes the proposed algorithm faster (fifteen times) in the problem considered below.

3.5 The Algorithm

We can now give a final version of the Ariadne's clew algorithm. It has three inputs: $\hat{q}_\circ$ (the initial position), $\hat{q}_\bullet$ (the goal position), and ρ (the maximum allowed distance for a path to the $\mathcal{C}$-*obstacles*). It returns a legal path or terminates if no path exists at the given resolution.

```
ALGORITHM_ARIADNE(q̂_o, q̂_•, ρ)
begin
    i := 1; λ̂_1 := q̂_o;
    /* Initialize the set of landmarks with the initial position
    Λ_1 := {λ̂_1}; ε_1 = +∞;
    do while (ε_i > ρ);
        /* run SEARCH : look for the goal with a local method
        if (min_{l∈IR^ℓ} d(q̂_•, q̂(l)) == 0)
            return; /* A path has been found !
        else
        /* run EXPLORE : place a new landmark
            i := i + 1;
            λ̂_i := q̂ : sup_{l∈IR^ℓ} d(Λ_{i-1}, q̂(l));
            Λ_i := Λ_{i-1} ∪ {λ̂_i};
            ε_i := d(Λ_{i-1}, λ̂_i);
        endif
    enddo
    Λ = Λ_i;
    ε = ε_i;
    return(ε); /* No path !
end
```

Figure 8: The Ariadne's Clew Algorithm

The algorithm is based on the following optimization problems:

$$EXPLORE : \begin{cases} \sup\ d(\Lambda_{i-1}, \hat{q}(l)) \\ l \in I\!R^{\ell} \end{cases}$$

$$SEARCH : \begin{cases} \min\ d(\hat{q}(l), \hat{q}_{\bullet}) \\ l \in I\!R^{\ell} \end{cases}$$

$\hat{q}(l)$ denotes the extremity of a legal path parameterized with ℓ real parameters and starting either from each of the previously placed landmarks (EXPLORE) or from the latest placed landmark (SEARCH).

The algorithm is resolution-complete under the following assumptions:

- "Space filling completeness": The global maximum distance can be found by the optimization algorithm used in EXPLORE; the configuration space is a compact set.
- "ρ completeness": The optimization procedure used in SEARCH always find a complete path (or returns 0) when the starting and the goal positions are located within a ball of radius ρ of the free space.

In practice, the first condition cannot be met with a randomized optimization algorithm in a bounded time, and only local maxima are found. However, the landmarks placed according to the new algorithm are better distributed over the free space than landmarks placed randomly, leading to better performances. The goal of the next section is to justify this claim, experimentally.

4. Path Planning for a Six DOF Arm in a Dynamic Environment

In order to demonstrate the feasibility and qualities of the Ariadne's clew algorithm, we have developed a realistic application of the algorithm. We selected a problem where we want to have a path planner for a six DOF robot arm in a dynamic environment where another arm is used as a mobile obstacle. The robot (robot A) is under the control of the Ariadne's clew algorithm. It shares its workspace with a second robot (robot B) that is moving under the control of a random motion generator. The Ariadne's clew algorithm must be able to compute paths for A in "real time" (here, real time means fast enough to ensure that robot A will never collide with robot B).

In order to reach such a level of performance, we chose to implement the Ariadne's clew algorithm on a massively parallel machine (Meganode with 128 T800 Transputers). Furthermore, we selected a genetic algorithm as our optimization technique. The reasons for this choice are:

1. Genetic algorithms are well suited for problems where the search space is huge but where there are many acceptable solutions. This is exactly the case here. The trajectory space is huge but there are, barring exceptional cases, numerous acceptable paths going from $\hat{q}_{\circ}$ to $\hat{q}_{\bullet}$ without collision.

2. Genetic algorithms, unlike a number of the other optimization techniques (Bessière, Talbi, Ahuactzin, & Mazer, 1996), are very easy to implement on parallel architectures. We have previously developed a parallel genetic algorithm (PGA) and we have already had significant experience using it (Talbi, 1993).

3. PGA, unlike most parallel programs, shows linear speed-up (when you double the number of processors you reduce the computation time by half) and even super-linear speed-up under certain circumstances (Talbi & Bessière, 1996).

4.1 Parallel Genetic Algorithm

Genetic algorithms are stochastic optimization techniques introduced by Holland (1975) twenty years ago. They are used in a large variety of domains including robotics (Ahuactzin et al., 1992; Lawrence, 1991; Falkenauer & Bouffouix, 1991; Falkenauer & Delchambre, 1992; Meygret & Levine, 1992) because they are easy to implement and do not require algebraic expression for the function to be optimized.

4.1.1 Principle of Genetic Algorithm

The goal of the algorithm is to find a point reaching a "good" value of a given function F over a search space S. First, a quantization step is defined for S and the search is conducted over a discrete subset, S_d of S. S_d contains 2^N elements. In practice, the cardinality of S_d can be *extremely* large. For example, in our implementation of EXPLORE, $N = 116$. Thus, a continuous domain is discretized with a given resolution.

During an initialization phase a small subset of S_d is drawn at random. This subset is called a *population*. Each element of this population is coded by a string of N bits.

The genetic algorithm iterates the following four steps until a solution is found.

1. **Evaluation**: Rank the population according to the value of F for each element of S_d. Decide if the best element can serve as an acceptable solution; if yes, exit.

2. **Selection**: Use the function F to define a probability distribution over the population. Select a pair of elements randomly according to this probability distribution.

3. **Reproduction**: Produce a new element from each pair using "genetic" operators.

4. **Replacement**: Replace the elements of the starting population by better new elements produced in step 3.

Many genetic operators (Davidor, 1989) are available. However, the more commonly used are the *mutation* and the *cross-over* operators. The mutation operator consists of randomly flipping some bits of an element of the population. The cross-over operator consists of first randomly choosing a place where to cut the two strings of bits, and then building two new elements from this pair by simply gluing the right and the left parts of the initial pair of strings (see Figure 9).

We use both operators to produce new elements. First, we use the cross-over operator to get an intermediate string. Then, the mutation operator is used on this intermediate string to get the final string.

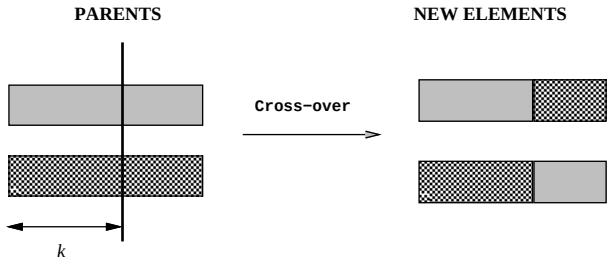


Figure 9: The cross-over operation.

4.1.2 PRINCIPLE OF THE PARALLEL GENETIC ALGORITHM (PGA)

There are many parallel versions of genetic algorithms: the standard parallel version (Robertson, 1987), the decomposition version (Tanese, 1987) and the massively parallel version (Talbi, 1993). We chose this last method. The main idea is to allocate one element of the population for each processor so that steps 1, 3, and 4 can be executed in parallel. Furthermore, the selection step (step 2) is carried out locally, in that each individual may mate only with the individuals placed on processors physically connected to it. This ensures that the communication overhead does not increase as the number of processors increases. This is the reason why PGA shows linear speed-up.

The parallel genetic algorithm iterates the following four steps until a solution is found.

1. **Evaluation**: Evaluate *in parallel* all the individuals.
2. **Selection**: Select *in parallel*, among the neighbors, the mate with the best evaluation.
3. **Reproduction**: Reproduce *in parallel* with the chosen mate.
4. **Replacement**: Replace *in parallel* the parents by the offspring.

On the Meganode, we implemented the PGA on a torus of processors where each individual has four neighbors (see Figure 10)

4.2 Parallel Evaluation of the Cost Function

The evaluation functions used in SEARCH and EXPLORE are very similar: they both compute the final position of the arm given a Manhattan path of a fixed order. In our implementation, based on experience, we chose to use Manhattan paths of order 2. Order 2 appeared to be a good compromise between the number of landmarks needed (increases as order decreases) and the computing time necessary for the optimization functions (increases as order increases). Since our robot has six DOF, the argument of the cost function in SEARCH is a vector in R^{12}: $(\Delta_1^1, \Delta_2^1, ..., \Delta_6^1, \ldots, \Delta_1^2, \ldots, \Delta_6^2)$ and the argument of the cost function used for EXPLORE is a vector in $\mathbb{N} \times \mathbb{R}^{12}$: $(i, \Delta_1^1, \Delta_2^1, ..., \Delta_6^1, \ldots, \Delta_1^2, \ldots, \Delta_6^2)$ where i codes the landmark used as a starting point for the path. In both cases the functions are defined only on a bounded subset of $\mathbb{R}^{12}$ and $\mathbb{N} \times \mathbb{R}^{12}$, whose limits are fixed by the mechanical stops of the robot and the maximum number of landmarks. A discretization step is chosen for these two subsets by defining the resolution at which each elementary motion is discretized.

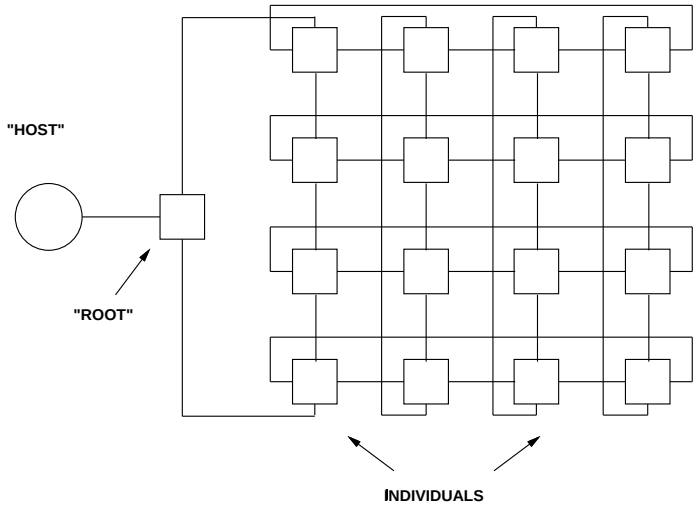


Figure 10: A torus with sixteen processors. One individual is placed on each processor. Each individual has four neighbors.

In our case, each Δ_i^j is discretized with 9 bits and the number of landmarks is limited to 256. Thus, given a binary string of $116 = 8 + 12 \times 9$ bits, we can convert it into a vector (as an argument) for the cost function of SEARCH, or EXPLORE, respectively.

Manhattan paths are evaluated in a simplified model of the environment. This model is obtained by enclosing each element of the scene into a bounding rectangular box.

The evaluation of a vector is performed as follows:

For each Δ_i^j in $(\Delta_1^1, \Delta_2^1, ..., \Delta_6^1, \ldots, \Delta_1^2, \ldots, \Delta_6^2)$

Compute the limits on the motion for joint i.

Compute $\Delta_i^{j\prime}$ by bouncing on these limits (see Section 3.4).

Update the position of the robot.

The limits on the motion of joint i are obtained by merging the legal ranges of motion of all the links that move when joint i moves, and all the obstacles. To obtain a legal range of motion between a link and an obstacle, we consider the two enclosing parallelepipeds and express their coordinates in the joint frame. Then, we use a classical method to compute the range (Lozano-Pérez, 1987).

In our parallel implementation, we distributed the geometric computations among several processors. Each processor is dedicated to the computation of a single type of interaction.

4.3 Parallel Implementation of the Ariadne's Clew Algorithm

Finally, the Ariadne's clew algorithm is implemented in parallel with three levels of parallelism.

1. Obviously, a first level of parallelization can be obtained by running SEARCH and EXPLORE at the same time on two sets of processors. While SEARCH is checking

whether a path exists between the last placed landmark and the goal, EXPLORE is generating the next landmark.

2. The second level of parallelism corresponds to a parallel implementation of both genetic algorithms employed by SEARCH and EXPLORE to treat their respective optimization problems.

3. The third level corresponds to a parallelization of the collision checking function and range computation.

We completed a full implementation of these three levels on a Meganode with 128 T800 transputers. Figure 11 represents our parallel implementation of the Ariadne's clew algorithm and Figure 12 shows how we have embedded this architecture into our experimental setup. A CAD system (ACT) is used to model the scene with the two robots. The robots are under the control of KALI (Hayward, Daneshmend, & Hayati, 1988). First, a simplified geometric model of the scene is downloaded into the memory of the transputers. Then, a Silicon Graphics workstation works as a global controller and loops over the following steps:

1. Generate and execute a legal random motion for robot B.

2. Send the new configuration of robot B to the Meganode as well as the desired final configuration for robot A.

3. Get the planned path for robot A from the Meganode and execute it.

4. Wait for a random time and stop robot A.

5. Go to 1.

This sequence allows us to test our algorithm extensively in real situations by having to deal with many different environments. Of course, the most interesting figure we can obtain from this experiment is the mean time necessary to compute one path given a new environment. For this experimental setup this mean time is 1.421 seconds. Using the same architecture with more up-to-date processors (T9000) would reduce this time by a factor of ten. The same computation on a single processor (SPARC 5) would take three times longer than the current implementation.

In summary, we have achieved our main goal by proving that it is indeed possible (with the Ariadne's clew algorithm) to plan collision-free paths for a real robot with many DOF *in a dynamic realistic environment.*

5. Conclusion: Contributions, Difficulties, and Perspectives

As mentioned in the Introduction, the Ariadne's clew algorithm has two main qualities: *efficiency*, and *generality*. Let us, in conclusion, explain and discuss these two qualities.

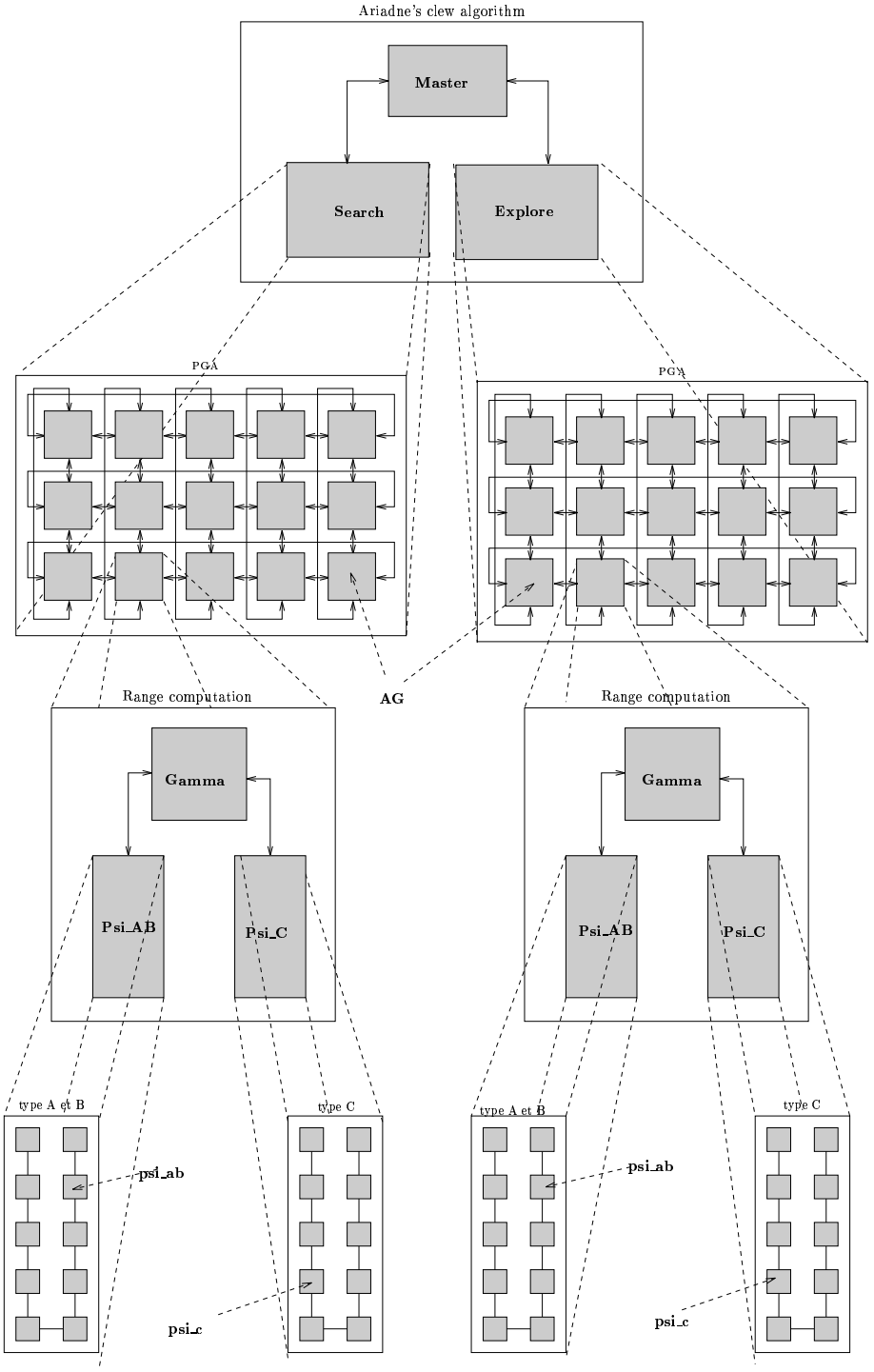


Figure 11: A parallel implementation of the Ariadne's clew algorithm

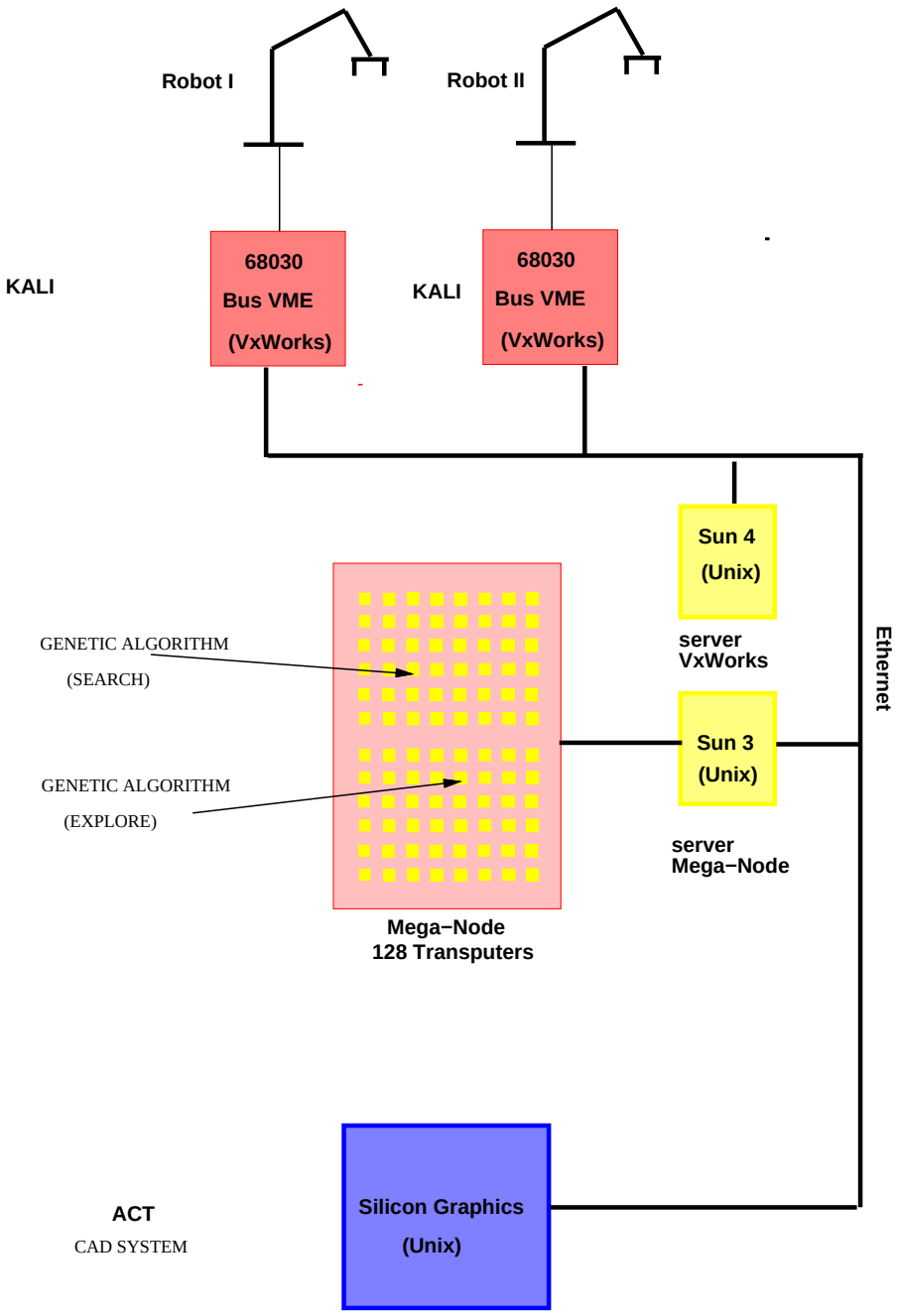


Figure 12: The experimental setup

5.1 Performance

Comparing the performance of this kind of algorithm is a very delicate subject. Performance may be a matter of computing time, efforts needed to program, or ease of application to different problems (see Section 5.2). Evaluating the performance in terms of computing time is very difficult for one fundamental and three practical reasons:

1. The fundamental reason is, once again, the NP-completeness of the path planning problem. As deceptive cases may always be designed, the only performance results one may reasonably present are always specific.

2. The three practical reasons are:

   (a) Obviously, the first requirement for such a comparison is that different algorithms run on the same machines with the same available memory. This may seem simple but it is a main difficulty in our case because our algorithm has been designed to run on rather specific kinds of machines, namely, massively parallel ones. It could also be implemented on non-parallel machines, but then it may lose part of its interest. A fair comparison would be to compare the algorithms on both types of machines. This would imply programming other algorithms in parallel, which is very difficult in practice.

   (b) Many known path planning algorithms first compute the configuration space (or an approximation of it) off-line, and then efficiently solve the path planning problem on-line. As we saw, in order to deal with a dynamic environment, the Ariadne's clew algorithm adopts a completely different approach.

   (c) For practical reasons, many test problems are toy problems (2D, few obstacles, few faces, simulated robots) and the performance results using these kinds of problems are very difficult to generalize to realistic industrial problems (3D, tens of obstacles, hundreds of faces, real robots).

Considering all these reasons, we tested our algorithm by implementing a realistic robotic application to the very end. To achieve this goal, we assembled a complex experimental setup including six different machines (1 MEGANODE, 2 68030, 2 SUN 4, and 1 SILICON GRAPHICS), two mechanical arms, and running seven different cooperative programs (2 KALI, 1 ACT, 2 VXWORKS, 1 PARX, and 1 Ariadne's clew algorithm).

Our challenge was to be able to solve the path planning problem fast enough to drive a real six DOF arm in a dynamic environment. The Ariadne's clew algorithm indeed achieved this goal in our experiments where the environment is composed of five fixed obstacles and a six DOF arm moving independently.

We are not aware of any other methods capable of such performance. To the best of our knowledge, currently implemented planners would take a number of seconds (ten) to place a set of landmarks on a 2D example for a robot with five DOF (Kavraki et al., 1996). Despite the fact that finding a general purpose planning technique for real industrial application is a very difficult problem, we believe that the Ariadne's clew algorithm provides an effective approach to such problems.

The number of range computations for a Manhattan motion of order 1 is $C\frac{k^2+k}{2} * n$ where n is the number of faces, k the number of DOF, and C a constant factor, depending on the number of parts used to model the robot. Obviously, such a number of faces may be a severe difficulty for the implementation of the Ariadne's clew algorithm described so far. To speed up the computation we use a number of geometric filters that reduce the number of pairs of entities to be analyzed.

However, it was possible to follow two research tracks in combination. First, we could use collision checking methods that allow access to the pairs in collision in a logarithmic

time (Faverjon & Tournassoud, 1987). Second, we could preserve part of the landmark graph when the environment is changing (McLean & Mazon, 1996).

5.2 Generality

The Ariadne's clew algorithm is general in the sense that it may be used for numerous and very different applications in robotics. Basically, the main thing that needs to be changed in the algorithm is the distance d used in the evaluation functions of the two optimization problems.

Several planners have been implemented in this way: a fine motion planner (De la Rosa, Laugier, & Najera, 1996), two motion planners for holonomic and non-holonomic mobile robots (Scheuer & Fraichard, 1997), a reorientation planner for an articulated hand(Gupta, 1995), a planner for grasping and regrasping (Ahuactzin, Gupta, & Mazer, 1998), and a planner for a robotic arm placed in the steam generator of a nuclear plant (McLean & Mazon, 1996). Adapting the algorithm to a new application is, therefore, clearly a very easy task. For instance, the application to path planning for the non-holonomic trailer was developed in three days.

The Ariadne's clew algorithm is also general in the sense that it may be used for any kind of path planning problem in a continuous space, in fields other than robotics. Although it may be sufficient to change the distance function d, one may also consider changing the form of the function d, or even the nature of the searched spaces. For instance, the concept of obstacles may be reconsidered. Instead of "hard" obstacles, one could replace them by zones of constraints. In that case, the path planning problem does not consist of finding a path without collisions but rather finding a path best satisfying the different constraints. Such a planner has been developed for a naval application where the problem was to find a path for a boat with various constraints on the trajectory. This opens numerous perspectives of applications for applying the Ariadne's clew algorithm in a broader field than pure robotics.

Acknowledgments

The authors are greatly indebted to Dr. Kamal Gupta from Simon Fraser University who carefully read the paper and suggested valuable corrections that greatly improve the quality of the final paper.

This work has been made possible by: Le Centre National de la Recherche Scientifique (France), Consejo Nacional de Ciencia y Tecnologia (Mexico) and esprit 2, P2528 (eec).

References

Ahuactzin, J., Gupta, K., & Mazer, E. (1998). Manipulation Planning for Redundant Robots: A Practical Approach. *The International Journal of Robotics Research*, *17*(7), 731–747.

Ahuactzin, J., Mazer, E., Bessière, P., & Talbi, E. (1992). Using Genetic Algorithms for Robot Motion Planning. In *Proceedings of the 1992 European Conference on Artificial Intelligence*, pp. 671–675.

Barraquand, J., & Latombe, J. (1990). A Monte Carlo Algorithm for Path Planning with Many Degrees of Freedom. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pp. 1712–1717.

Bessière, P., Talbi, E., Ahuactzin, J., & Mazer, E. (1996). Un Algorithme Génétique Parallèlle pour l'Optimisation. *Technique et Science Informatique*, *15*(8), 1105–1130.

Brooks, R. (1983). Solving the Find-Path Problem by Good Representation of the Free Space. *IEEE Transactions on System, Man and Cybernetics*, *13*(4), 190–197.

Canny, J. (1988). *The Complexity of Robot Motion Planning.* MIT Press.

Davidor, Y. (1989). Analogous Crossover. In *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 98–103.

De la Rosa, F., Laugier, C., & Najera, J. (1996). Robust Path Planning in the Plane. *IEEE Transactions on Robotics and Automation*, *12*(3), 347–352.

Falkenauer, E., & Bouffouix, S. (1991). A Genetic Algorithm for Job Shop. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pp. 824–829.

Falkenauer, E., & Delchambre, A. (1992). A Genetic Algorithm for Bin Packing and Line Balancing. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pp. 1186–1192.

Faverjon, B., & Tournassoud, P. (1987). A Local Based Approach for Path Planning of Manipulators with a High Number of Degrees of Freedom. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, pp. 1152–1159.

Ferbach, P. (1996). Contribution à la Planification de Trajectoires. Rapport de Recherche EDF-DER HP-28/96/026, Direction des Études et Recherches EDF.

Gupta, K. (1995). Motion Planning for Re-Orientation Using Finger Tracking: Landmarks in $SO(3) \times \omega$. In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, pp. 446–451.

Hayward, V., Daneshmend, L., & Hayati, S. (1988). An Overview of KALI: A System to Program and Control Cooperative Manipulators. In *Proceedings of the Fourth International Conference on Advanced Robotics*, pp. 236–240.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Hwang, Y., & Ahuja, N. (1992). Gross Motion Planning: A Survey. *ACM Computing Surveys*, *24*(3).

Kavraki, L., Svestka, P., Latombe, J., & Overmars, M. (1996). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, *14*(4), 566–580.

Lawrence, D. (Ed.). (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold.

Lin, H., Xiao, J., & Michalewicz, Z. (1994). Evolutionary Navigator for a Mobile Robot. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 2199–2004.

Lozano-Pérez, T. (1987). A Simple Motion-Planning Algorithm for General Robot Manipulators. *IEEE Transactions on Robotics and Automation*, *3*(3), 224–238.

McLean, A., & Mazon, T. (1996). Incremental Roadmaps and Global Path Planning in Evolving Industrial Environments. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pp. 101–106.

Meygret, A., & Levine, M. (1992). Extraction de Primitives Géométriques: Utilisation d'un Algorithme Génétique. Rapport Annuel, Center for Intelligent Machines, McGill University, Montréal.

Overmars, M. (1992). A Random Approach to Motion Planning. Technical Report RUU-CS-92-32, Department of Computer Science, Utrecht University.

Robertson, G. (1987). Parallel Implementation of Genetic Algorithms in a Classifier System. In Davis, L. (Ed.), *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers.

Scheuer, A., & Fraichard, T. (1997). Continuous-Curvature Path Planning for Car-Like Vehicles. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 997–1003.

Talbi, E. (1993). *Allocation de Processus sur les Architectures Parallèles à Mémoir Distribuée*. Ph.D. thesis, Institut National Polytechnique de Grenoble - France.

Talbi, E., & Bessière, P. (1996). A Parallel Genetic Algorithm Applied to the Mapping Problem. In Astfalk, G. (Ed.), *Applications on Advanced Architecture Computers*. SIAM.

Tanese, R. (1987). Parallel Genetic Algorithm for a Hypercube. In *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 177–183.

Xiao, J., Michalewicz, Z., & Zhang, L. (1996). Evolutionary Planner/Navigator: Operator Performance and Self-Tuning. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pp. 336–371.