3

Steps Towards Safe Navigation in Open and Dynamic Environments

Christian Laugier, Stéphane Petti, Dizan Vasquez, Manuel Yguel, Thierry Fraichard, and Olivier Aycard

INRIA Rhöne-Alpes & GRAVIR Lab. (CNRS, INPG, UJF)
http://emotion.inrialpes.fr

Summary. Autonomous navigation in open and dynamic environments is an important challenge, requiring to solve several difficult research problems located on the cutting edge of the state of the art. Bassically, these problems can be classified into three main categories: SLAM in dynamic environments; Detection, characterization, and behavior prediction of the potential moving obstacles; On-line motion planning and safe navigation decision based on world state predictions. This paper addresses some aspects of these problems and presents our latest approaches and results. The solutions we have implemented are mainly based on the followings paradigms: *Characterization and motion prediction* of the observed moving entities using bayesian programming; *On-line goal-oriented navigation decisions* using the Partial Motion Planning (PMP) paradigm.

3.1 Introduction

3.1.1 Outline of the Problem

To some extent, autonomous navigation for robotic systems placed in stationary environments is no longer a problem. The challenge now is autonomous navigation in open and dynamic environments, *i.e.* environments containing moving objects (potential obstacles) whose future behaviour is unknown. Taking into account these characteristics requires to solve several difficult research problems at the cutting edge of the state of the art. Basically, these problems can be classified into three main categories:

- Simultaneous Localisation and Mapping (SLAM) in dynamic environments;
- Detection, tracking, identification and future behaviour prediction of the moving obstacles;
- On-line motion planning and safe navigation

In such a framework, the system has to continuously characterize the fixed and moving objects that can be observed both with on-board or off-board sensors. As far as the moving objects are concerned, the system has to deal with problems such as interpreting appearances, disappearances, and temporary occlusions of rapidly manoeuvring objects. It also has to reason about their future behaviour (and consequently to make predictions). From the autonomous navigation point

C. Laugier and R. Chatila (Eds.): Autonomous Navi. in Dyn. Environ., STAR 35, pp. 45–82, 2007.
springerlink.com

of view, this means that the system has to face a double constraint: constraint on the response time available to compute a safe motion (which is clearly a function of the dynamicity of the environment), and a constraint on the temporal validity of the motion planned (which is a function of the validity duration of the predictions). In other words, one needs to be able to plan motion fast, but one does not need to plan motion very far in the future.

This paper addresses some aspects of the previous problem, and presents our latest approaches and results. The solutions we have implemented rely on the following modules:

- *Scene interpretation and short-term motion prediction* for the moving obstacles, using the new concept of Bayesian Occupancy Filters and an efficient wavelet-based representation of the related occupancy grids;
- *Medium-term motion and behaviour prediction* for the observed moving objects, using motion pattern learning and Hidden Markov Models;
- *On-line goal-oriented navigation decision* using the Partial Motion Planning (PMP) paradigm.

3.1.2 Case Study: The Automated Valet Parking

One possible (and very relevant) target application for the techniques presented in this paper is that of the Automated Valet Parking (AVP). The robotic system considered is a "smart" car operating autonomously in a "smart" city parking. Both the car and the parking are equipped with sensors providing them with information about the world. Let us imagine the following scenario: you drive your car and leave it at the entrance of a given parking. From then on, it operates autonomously and go park itself. As soon as the car enters the parking, the car on-board intelligent system connects to the parking's own intelligent system and request a free parking place. The parking then confirms the availability of a parking space and provides the car with a model of the parking and an itinerary to the empty place. From then on, the car, using information obtained from both its own sensors and the parking sensory equipment, go park itself.

From an architecture point of view, the AVP scenario involves two "intelligent" systems communicating with one another: the car on-board system and the parking off-board system. As mentioned earlier, it is assumed that both systems are equipped with sensors providing them with information about the environment considered. While the car sensors will provide it with a local view of its surroundings, it can be expected that the parking sensors will provide the car with an overall view of what is going on in the whole parking.

To address the AVP scenario, we have devised a scheme relying upon the following functionalities (split between the car and the parking).

Parking abilities

- Parking monitoring: at any time, the parking should know which places are free or not (and by whose car they are occupied).

- Route planning: the parking should be able to provide the car with a model of the parking premises along with the best itinerary to reach a given place.
- Moving objects monitoring: any moving objects (vehicles, pedestrians, etc.) should be monitored and tracked. The parking should be able to provide information such as position, speed and expected trajectory (*i.e.* future behaviour). Expected trajectories can come from different clues: typical movements of the different kinds of moving objects, learnt from previous observation, or knowledge of a planned trajectory.
- Car localisation: given its moving objects' monitoring functionality, the parking can provide the car with its current state in the parking premises.

Car abilities

- Localisation: the car should be able to maintain an estimate of its localisation in the parking. It can be the result of a data fusion between parking information and on-board localisation.
- Environment modelling: the car on-board sensor are primarily used to build a model of the surroundings of the car. This local model should be enriched using the global information provided by the parking (in particular, the information concerning the moving objects' future behaviour).
- Automated driving: given the parking model and the route to the goal, the car should be able to determine its future course of action so as to reach its goal efficiently and safely.

One can notice that some of the functionalities mentioned above are somewhat redundant (in particular when dealing with sensing data). This property has intentionally been chosen in order to increase the robustness and the efficiency of the system:

- Fusion of data from multiple source increase overall accuracy.
- Using several data source increase fault tolerance.
- By correlating different inputs, it is possible to diagnose if an input is failing or becoming unreliable.

3.1.3 Outline of the Paper

In this paper, we focus on two of the functionalities mentioned in the previous section: *Motion prediction of the observed moving objects* and *on-line goal-oriented navigation decisions*. The paper is organized in three main sections. The section 3.2 describes how we have solved the problem of interpreting and representing the dynamic environment of the robot using the "Bayesian Occupancy Filtering"(*BOF*) approach; this approach relies on a local world-state bayesian interpretation scheme, including a short-term motion prediction mechanism. The section 3.3 deals with the problem of the prediction of the most likely behaviors of some observed objects executing "intentional motions"; the proposed solution relies on the use of a motion pattern learning mechanism and of an extended Hidden Markov Model. The section 3.4 deals with the problem of planning safe

motions in a reconstructed dynamic environment; the proposed paradigm (called "Partial Motion Planning", or PMP) takes into account (at each iteration step) both the time constraints and the current model of the future state of the robot environment.

3.2 Scene Interpretation and Short-Term Motion Prediction

3.2.1 Overview of the Problem

The problem addressed in this section concerns the interpretation of the observed dynamic scene in terms of *potential moving obstacles*, i.e. obstacles which may generate a collision in the near future with the robot). The objective is to be able to correctly interpret the dynamic scene in the presence of noisy or missing data, and to be as robust as possible to temporary or partial occlusions. Our approach for solving this problem is based on the new concept of *Bayesian Occupancy Filtering (BOF)* [1], where the robot environment is represented using a *4-dimensional occupancy grid*, i.e. an occupancy grid which includes the velocity dimension.

The *occupancy grids* [2, 3] framework is a classical way to describe the environment of a mobile robot. It has extensively been used for static indoor mapping [4] using a 2-dimensional grid. More recently, occupancy grids have been adapted to track multiple moving objects [5]. However, a major drawback of these approaches, is that a moving object may be lost due to occlusion effects.

The BOF approach avoid this problem for short temporary occlusions (e.g. a few seconds), by combining two complementary phases in a recursive loop: the *estimation phase* which estimate the occupancy probability of each cell of the 4-dimensional grid, using recursively the set of sensor observations; the *prediction phase* which estimate an a priori model of the grid occupancy at time $k + 1$, using a "dynamic model" and the latest estimation of the grid state (figure 3.2 illustrates). This approach has been developed using the *Bayesian Programming* Framework [6, 7, 8]; it is described in the next sections.

However, large scale environments can hardly been processed in real-time because of the intrinsic complexity of the related inferences and numerical computations (see section 3.2.6). The section 3.2.7 presents the outline of the WOG model ("Wavelet Occupancy Grid") we are developing for trying to meet the required efficiency property.

3.2.2 Estimation of the Occupancy Probability

The *estimation phase* consists in estimating, at each time step, the occupancy probability of each cell of the 4-dimensional grid. This estimation is performed using recursively the set of "observations" (i.e. pre-processed sensors data) provided by the sensors at each time step. These observations are represented by

a list of detected objects, along with their associated positions and velocities in the reference frame of the processed sensor (several sensors may be used in parallel). In practice, this set of observations could also contain two types of false measurements : the *false alarms*, i.e. when the sensor detects a non existing object; the *missed detection*, i.e. when the sensor does not detect an existing object.

Solving the related estimation problem is done by using the available instantaneous information about the environment state (i.e. the current observations and grid state). A sketch of the algorithm is given below using our *Bayesian Programming* Framework [6, 7, 8]; a more complete description of the method (which includes a "sensor data fusion" step) can be found in [1].

(i) Choosing the relevant variables and decomposition

- O_{x,y,v_x,v_y}: The occupancy of the cell (x, y, v_x, v_y) at time t: occupied or not. This variable is indexed by a 4-dimensional index that represents a position and a speed relative to the vehicle.
- $\mathcal{Z}$: The sensor observation set; one observation is denoted Z_s; the number of observation is denoted S.

If we make the reasonable assumption that all the observations of the sensors are independent when knowing the occupancy state of a cell, we can choose to apply the following decomposition of the related joint distribution:

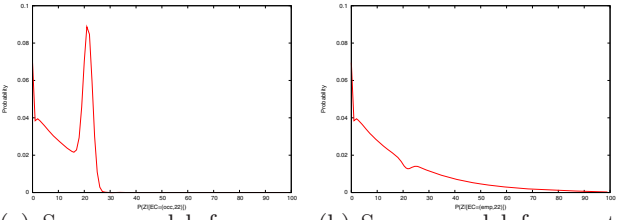
$$P(O_{x,y,v_x,v_y}, \mathcal{Z}) = P(O_{x,y,v_x,v_y}) \times \prod_{s=1}^{S} P(Z_s \,|O_{x,y,v_x,v_y}). \tag{3.1}$$

(ii) Assigning the parametric forms
According to our knowledge of the problem to be solved, we can assign the following parametric forms to each of the terms of the previous decomposition:

- $P(O_{x,y,v_x,v_y})$ represents the *a priori* information on the occupancy of each cell. If available, a *prior* distribution could be used to specify it. Otherwise, a uniform distribution has to be selected. The next section will show how the prior distribution may be obtained from previous estimations.
- The shape of $P(Z_s \,|O_{x,y,v_x,v_y})$ is given by the *sensor model*. Its goal is to model the sensor response knowing the cell state. Details about this model can be found in [3]; an example is given for a telemetric sensor in Fig. 3.1. Such models can easily been built for any kind of sensor.

(iii) Solution of the problem
It is now possible to ask the *Bayesian question* corresponding to the searched solution (i.e. the searched probability distribution). Since the problem to solve consists in finding a good estimate of the cell occupancy, the question can be stated as follows:

$$P(O_{x,y,v_x,v_y} \mid \mathcal{Z})? \tag{3.2}$$



(a) Sensor model for occupied cells (b) Sensor model for empty cells

Fig. 3.1. Example of one-dimensional sensor models. The sensor is a laser-range finder located in $x = 0$ and detecting an object at $x = 22$. The following property holds : $P(Z|[O_x = occ]) = P(Z|[O_x = emp])$ for $x > z$, which implies that $P(O_x|Z)$ is unknown when the cell is behind the obstacle.

The result of the related Bayesian inference[1] can be written as follows:

$$P(O_{x,y,v_x,v_y} \mid \mathcal{Z}\ ) \propto \prod_{s=1}^{S} P(Z_s \mid O_{x,y,v_x,v_y}\ ). \tag{3.3}$$

3.2.3 The Bayesian Occupancy Filter

We are now interested in taking into account the sensor observation history, in order to be able to make more robust estimations in changing environments (i.e. in order to be able to process temporary objects occlusions and detection problems). A classical way to solve this problem is to make use of Bayes filters [9]. Basically, the goal of such a filtering process is to recursively estimate the probability distribution $P(O^k_{x,y,v_x,v_y} \mid Z^k)$, known as the *posterior* distribution. In general, this estimation is done in two stages: a *prediction* stage whose purpose is to compute an *a priori* estimate of the target's state known as the *prior* distribution; an *estimation* stage whose purpose 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 (such as the Kalman filter [10][11] when linear functions and gaussian distributions are used).

Our approach for solving this problem, is based on the new concept of *Bayesian Occupancy Filter (BOF)*; Fig. 3.2 shows the related estimation loop. This approach is derived from the general bayesian filtering paradigm; it provides a powerfull formalism for solving difficult estimation problems expressed in our 4-dimensional occupancy grid representation.

The basic idea underlying the conception of the *BOF* is to make use of the velocities measured in the past, for predicting the near future and propagating this information through time. Indeed, knowing the current velocity of a mobile,

[1] This result is computed using our *ProBT* inference engine, currently commercialized by our spin-off company *Probayes*.

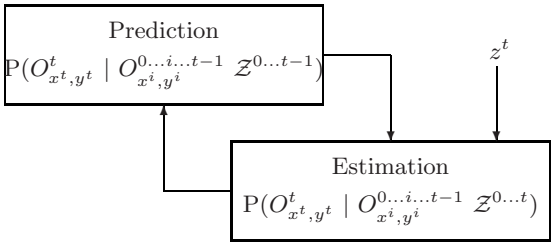


Fig. 3.2. Bayesian Occupancy Filter as a recursive loop

it is possible to predict its future motion under the hypothesis of a constant velocity for the next time step (in practice, possible velocity changes will generate several possible future positions).

A complete presentation of the *BOF* can be found in [1]. In the sequel, we will just give an overview of the approach under the following simplifying assumptions (for clarity reasons) : use of a single sensor and constant velocity for the observed moving objects. A consequence of this last assumption, is that we can deal with a single "antecedent cell" when evaluating the occupancy state of a given cell.

(i) Choosing the relevant variables and decomposition

- $O^t_{x,y}$: Occupancy of the cell $c = (x, y, v_x, v_y)$ at time t, occupied or not.
- $O^{t-\delta t}_{x_p,y_p}$: Occupancy of the cell which is the antecedent of $C^t_{x,y}$, occupied or not. In this model, $x_p = x - v_x\delta t$ and $y_p = y - vy\delta t$, since the velocity is constant.
- Z_s : A sensor observation.

Under the previous simplifying assumptions, the following decomposition of the joint distribution determined by these variables can be expressed as follow:

$$P(O^{t-\delta t}_{x_p,y_p}, O^t_{x,y}, Z_s) = \begin{pmatrix} P(O^{t-\delta t}_{x_p,y_p}) \\ \times P(O^t_{x,y}|O^{t-\delta t}_{x_p,y_p}) P(Z_s|O^t_{x,y}) \end{pmatrix}. \tag{3.4}$$

(ii) Assigning the parametric forms

- $P(O^{t-\delta t}_{x_p,y_p})$ is the *prior* for the future occupancy state of the cell $c = (x, y, v_x, v_y)$. For each cell c such as the antecedent (x_p, y_p, v_x, v_y) is out of the current grid, this prior is the probability that a new object enters in the monitored space; since we usually have no real information about such an event, this probability is represented by a uniform distribution.
- $P(O^t_{x,y}|O^{t-\delta t}_{x_p,y_p})$ is related to the very simple "dynamic model" we are using in this case. It is defined as a transition matrix $\begin{bmatrix} 1-\epsilon & \epsilon \\ \epsilon & 1-\epsilon \end{bmatrix}$, which allows the

system to take in account the fact that the null acceleration hypothesis is an approximation; in this matrix, ϵ is a parameter representing the probability that the object in $c = (x_p, y_p, v_x, v_y)$ does not follow the null acceleration model.

- $P(Z_s|O^t_{x,y})$ is the sensor model (see section 3.2.2).

(iii) Solution of the problem.
Similarly to the estimation process described in the section 3.2.2, the solution of the problem to be solved by the BOF can be defined by the following Bayesian question : $P(O^t_{x,y}|Z_s)$?. Answering this question (i.e. computing the related probability distribution) is achieved using our inference engine; this inference involves a marginalization sum over $O^{t-\delta t}_{x_p,y_p}$.

3.2.4 Experimental Results

This approach has been implemented and tested on our experimental platform : the Cycab vehicle equiped with a laser sensor. Fig. 3.3 shows some resulting grid estimations, for an environment containing two stationary objects and an object moving from the left to the right at a velocity of 0.8 m/s; in this example, the robot is not moving.

Fig. 3.3b depicts the occupancy probability of each cell corresponding to a null relative velocity (*i.e.* $c = [x, y, 0, 0]$). As expected, two areas with high occupancy probabilities are visible; these probability values depends on several factors attached to the sensor model : the probability of true detections, the probability of false alarms, and the sensor accuracy. Since the measured speed for the third obstacle is not null, any area of high occupancy probability corresponding to this observation is only represented in the related slices of the grid (i.e. the slice corresponding to $c = [x, y, 0, 0.8]$ in this case, see Fig. 3.3c).

It should be noticed that the cells located outside of the sensor field of view, or the cells "hidden" by one of the three sensor observations (i.e. the cells located behind the three detected obstacles) cannot be observed; consequently, nothing really consistent can be said about these cells, and the system has given an occupancy probability value of 0.5 for these cells.

Fig. 3.4 shows a sequence of successive prediction and estimation results given by the BOF. The experimental scenario involves a stationary obstacle and the Cycab moving forward at a velocity of 2.0 m/s. The obstacle is detected using the laser sensor; it finally goes out of the sensor field of view (see Fig. 3.4-d1), since the Cycab is moving forward. It should be noticed that the prediction step allows to infer knowledge about the current occupancy state of the cycab environment, even if the object is no longer observed by the sensor; this is the situation depicted by fig 3.4-d3, where an area of high occupancy probability still exists when the object is going out of the sensor field of view. In some sense, our prediction step can be seen as a "short-term memory", which allows to combine in an evolutive way past and current observations.

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

b)
$P([O_C = occ] \mid \mathcal{Z})$
$[c = (x, y, 0, 0)]$

c)
$P([O_C = occ] \mid \mathcal{Z})$
$[c = (x, y, 0, 0.8)]$

Fig. 3.3. Example of a grid estimation using a single laser sensor located at $(0, 0)$, in a scene including two stationary objects and an object moving from the left to the right at a velocity of 0.8 m/s. In this example, the robot is not moving. The occupancy probability value of each cell is represented by a gray level (see the colors correspondences on the right side of the figure). (a) The sensed environment and the three instantaneous sensor observations expressed in the $(x, y, \dot{x}, \dot{y})$ space. (b) Occupancy probability in a 2-dimensional slice of the grid corresponding to a null relative velocity (*i.e.* $c = [x, y, 0, 0]$). (c) Occupancy probability in a 2-dimensional slice of the grid corresponding to a relative horizontal velocity 0.8 (*i.e.* $c = [x, y, 0, 0.8]$).

3.2.5 *BOF* Based Collision Avoidance

In [1], we have shown how to avoid a mobile obstacle by combining the occupancy grid result given by the *BOF*, with a *danger probability* term computed using a "time to collision" criteria. In this approach, each cell O^k_{x,y,v_x,v_y} of the grid is characterized by two probability distributions : the "occupancy" term $P(O^k_{x,y,v_x,v_y} \mid \mathcal{Z}^k)$ and the "danger" term $P(D^k_{x,y,v_x,v_y} \mid O^k_{x,y,v_x,v_y})$. Using this model, it becomes possible to tune the velocity controls of the Cycab, according to the results of a combination of the two previous criteria.

This approach has experimentally been validated using the following scenario : the Cycab is moving forward; a pedestrian is moving from right to left, and during a small period of time the pedestrian is temporarily hidden by a parked car. Fig 3.5 shows some snapshots of the experiment : the Cycab brakes to avoid the pedestrian which is temporarily hidden by the parked car, then it accelerates as soon as the pedestrian has crossed the road.

3.2.6 Discussion and Performances

Thanks to the previous assumptions, both the prediction step and the estimation step complexities *increases linearly with the number of cells* of the grid. This make the approach tractable in real situations involving reasonable grid sizes. This is the case for the experimental scenarios described in this section (e.g. size of the sensed space of 10 m m with a resolution of 0.5 m, longitudinal

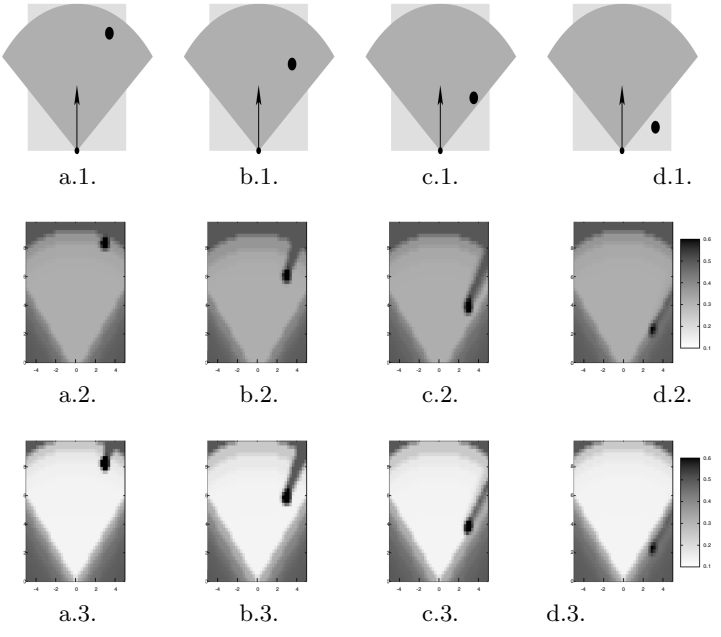


Fig. 3.4. A short sequence of a dynamic scene involving a stationary obstacle and the Cycab moving forward at a velocity of 2.0 m/s. The second and the third row respectively show the results of the prediction and of the estimation steps of the BOF, in a 2-dimensional slice of the grid corresponding to a relative speed of $\dot{x} = -2.0$, i.e. $P([O^t_{x\ y\ [\dot{x}=-2.0]\ [\dot{y}=0.0]} = occ]\ )$.

Fig. 3.5. Snapshots of the experimental pedestrian avoidance scenario

velocity of $-3\ m.s^{-1}$ to $1\ m.s^{-1}$ with a resolution of $0.4\ m.s^{-1}$, lateral velocity of $-3.2\ m.s^{-1}$ to $3.2\ m.s^{-1}$ with a resolution of $0.4\ m.s^{-1}$). Using such a grid of 64.000 cells, the computation time for both prediction and estimation steps is about 100 ms on a 1 GHz computer. This is fast enough to control the CyCab at a maximum speed of $2\ m.s^{-1}$.

However, this grid size is not fine enough for some other large scale applications involving higher speeds. In this case, the number of cells increases quickly,

and the required computational time becomes too high for satisfying the real-time constraint.

In order to try to solve this problem, we are working in two directions. The first one consists in developing a dedicated hardware exploiting the *highly parallel* structure of the BOF algorithm[2]. The second approach consists in using a multi-resolution framework for representing the 4-dimensional grid and the related processing models. The outline of this approach is described in the next section.

3.2.7 Wavelet-Based Model for the BOF (the WOG Model)

(i) Overview of the problem

The goal of this new model is to provide a "coarse-to-fine" representation of the underlying BOF model, in order to be able to optimize the processing of large homogeneous regions, and to refine the model only when this is necessary. We have chosen to make use of the wavelet framework [12] [13], which allows the processing of large sets of data including non-linearities (as it is the case for our dynamic maps). Pai and Reissell [14] have shown that wavelets could be used for representing 3D static maps for path planning. Sinopoli [15] has extended this approach for solving a global path-planning problem, while using traditional 3D Occupancy Grids (OG) for local navigation.

Our approach, called *"Wavelet Occupancy Grid"* (WOG), can be seen as a tight combination of wavelet and OG representations, allowing us to perform *OG updates in the wavelet space*, and later on to make "prediction inferences" within the wavelet space (i.e. to fully implement the BOF paradigm in this multi-resolution formalism).

(ii) The WOG model

The first objective is to develop a wavelet-based model for 2-dimensional OG (i.e. without representing velocities). At this step of the development, we will only consider the random variables $O^t_{x,y}$ and Z^t (defined above). Since each occupancy variable ($O_{x,y}$) is binary, its probability distribution is completely defined by $P([O^t_{x,y} = occ])$. Then, we consider now the occupancy function $p^t(x,y) = P([O^t_{x,y} = occ])$ which allows to capture the space homogeneity.

The used wavelet model

Basically, linking OG and wavelet representations, leads to project a huge function representing the OG into a wavelet vector space. In our approach, we have used the 2D Haar model [13], built upon two types of basis functions: the "scale" and "detail" functions, where the scaling mother function is defined as follows:

$$\Phi(x,y) = 1 \text{ for } (x,y) \in [0,1]^2, \text{ zero elsewhere}$$

[2] Thanks to the hypothesis that each cell is independent, the state of each cell can be computed independently.

Then, the Haar basis at scale s is defined as the union of the set of "scaled" functions $\{\Phi^{sij} | (i,j) \in \mathbb{Z}^2\}$ and the set of "details" functions $\{\Psi_M^{lij} | l = 0, \ldots, s; (i,j) \in \mathbb{Z}^2\}$, where:

$$\Phi^{lij} = 2^{-l}\Phi(2^{-l}x - i, 2^{-l}y - j) \tag{3.5}$$

and the type M can take three values (01, 10 or 11) corresponding to one of the three mother wavelets for horizontal, vertical and diagonal differencing.

Each t-uplet (s,i,j) defines a *wavelet square* at scale s and an offset (i,j). Thanks to the Haar model property, the projection of the occupancy function over a basis vector function Φ^{sij} is given by the following scalar product of Φ^{sij} with the occupancy function:

$$< p(x,y) \mid \Phi^{sij} >= \int_{x,y \in \mathbb{R}^2} p(x,y)\Phi^{sij}(x,y)d_{x,y} \tag{3.6}$$

Logarithmic form for OG updates

In the sequel, we will omit the index (x,y) associated to the OG cells (for simplifying the notations). A standard update in OG is defined by the following equation:

$$p([O^t = occ]) = \frac{p([O^{t-1} = occ])p(z^t|[O^{t-1} = occ])}{\sum_{o^t} p(o^t)p(z^t|o^t)} \tag{3.7}$$

Let $p^t(occ) = p([O^t = occ])$ and $p^t(emp) = p([O^t = emp])$; then we can write $p^t(occ) = 1 - p^t(emp)$, and we can summarize this property using a single coefficient q^t:

$$q^t = p^t(occ)/p^t(emp) \tag{3.8}$$

Substituting eq. 3.7 into eq. 3.8 leads to the elimination of the marginalisation term; then, we can write the following recursive formulation:

$$q^t = \frac{p(z^{t-1}|occ)}{p(z^{t-1}|emp)}q^{t-1} = q^0 \prod_{i=0}^{t-1} \frac{p(z^i|occ)}{p(z^i|emp)} \tag{3.9}$$

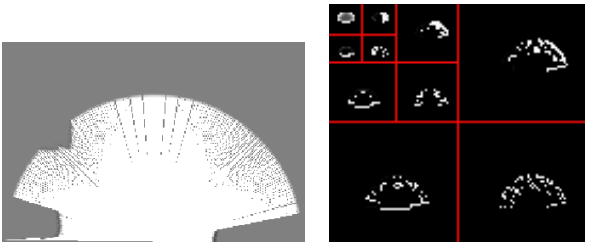
where q_{t-1} is recursively evaluated.

Such updating operations are clearly non-linear. In order to be able to perform the update operations using only sums, we have decided to make use of a logarithmic form of the model:

$$odds^t = odds^0 + \sum_{i=0}^{t-1} \log(\frac{p(z^i|occ)}{p(z^i|emp)}) \tag{3.10}$$

where $odds^t = log(q^t)$.

Let $Obs(z^{t-1})$ be the term $\log(\frac{p(z^{t-1}|occ)}{p(z^{t-1}|emp)})$ in the eq. (3.10). $Obs(z^{t-1})$ represents the "observation" term of the update, and $odds^(t-1)$ is the a priori



(a) (b)

Fig. 3.6. Mapping obtained by a laser range-finder : (a) the obtained *OG* model; (b) the related three first "detail spaces" of the WOG model and the complementary scaled space. The density of non-zero coefficients decreases drastically in wavelet space.

term. Then, updating a WOG can be done by adding the wavelet transform of observation terms to the wavelet representation of the map.

Multi-scale semantic of logarithmic OG

In the case of the Haar wavelet basis (eq. 3.5), $\Phi_s^{(i,j)}$ has a constant value (k) over the domain[3] (s,i,j), and is equal to zero elsewhere. Then the integral of the scalar product is a sum over the cells enclosed in the domain (s,i,j); this sum can be re-written as follows [16]:

$$< \log(\frac{p^t(x,y)}{1-p^t(x,y)}) \mid \Phi^{sij} >$$

$$= \int_{x,y\in\mathbb{R}^2} \log(\frac{p^t(x,y)}{1-p^t(x,y)})\Phi^{sij}(x,y)d_{x,y} \tag{3.11}$$

$$= k\log(\frac{\Pi_{c\in(s,i,j)}p([O_c=occ])}{\Pi_{c\in(s,i,j)}p([O_c=emp])}) \tag{3.12}$$

$$= k\log(q_s) \tag{3.13}$$

where c is the index of a cell in the square domain. $k\log(q_s)$ is the log of the ratio of two geometric means (cells are "all occupied" and cells are "all empty") which leads us to a continuous semantics. Let define *full* as the event "every subcell is occupied" then:

$$\Pi_{c\in(s,i,j)}p([O_c=occ]) = p(\bigwedge_{c\in(s,i,j)}[O_c=occ]) = p(full)$$

Let define *open* as the event "every subcell is empty".

q_s leads immediately to the conditional probabilities:

$$P(open|open\vee full) = q_s/(1+q_s) \tag{3.14}$$

$$P(full|open\vee full) = 1/(1+q_s) \tag{3.15}$$

which express a continuous ratio between the two events *full* and *open* (Fig. 3.7).

[3] With value: $k=2^{-s}$.

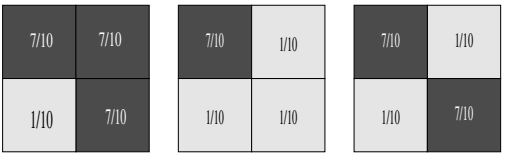


Fig. 3.7. Three configurations of a subsquare domain at scale 1/2: the probability that the square domain 7(a) is fully "occupied" is 0.0343; its probability of being fully "empty" is 0.0243. The occupancy ratio related to the "occupied" and "empty" properties is $P(open|open \vee full) = 0.58$; this means that the related square domain is considered as "occupied". The occupancy ratio in 7(b) is 0.003; it is of 0.06 in 7(c).

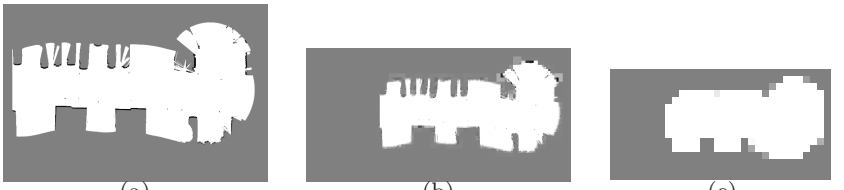
The multi-scale information which can be computed by this approach, is directly derived from these two basic cases. Consequently, only 2 relevant events can be considered for a square domain (s, i, j) containing n subcells (i.e. where 2^n possibilities can be defined by the binary vector representing the possible occupancy of these n subcells), Fig. 8(c) illustrates. Fig. 3.7 shows the information which can be derived from the previous multi-scale occupancy model. The next step will be to exploit this concise information for elaborating multi-scale algorithms.

(iii) WOG implementation and experimental results

This approach has been implemented and tested on the Cycab equipped with a laser sensor an with our SLAM method [17]. In the experiments, the size of the finer scale is equal to $6.25cm$, and we have used 5 different scales (where the size is multiplied by 2 for each new scale); thus, the size of the coarsest cells is equal to 1 m. The size of the square window which is used for constructing the WOG model, is chosen in order to contain the whole sensor field of view. The content of this window is updated at each laser scan (see Fig. 3.6; then, the Haar wavelet transform is applied to this sensory data, and incrementally added to the current WOG model. The chosen compression threshold is a compromise between data fitting and sparsity (wavelet terms which are lower than the threshold are removed).

Fig. 3.8 shows the results obtained on the car park of INRIA. These experimental results clearly shows that we have obtained a significant reduction of the size of the model (about 80% relatively to the OG model), and that the interesting details are still represented (such as the beacons represented by dark dots in Fig. 8(b)). It should be noticed that the coarser model give a quite good representation of the empty space (see Fig. 8(c)); this model could be used for path planning, and refined when necessary using the wavelet model. In the previous experiments, the map building has been done in real-time.

One of the major issue of our approach is now, to combine bayesian occupancy filter and object tracking such as the informations about the object motions



(a) (b) (c)

Fig. 3.8. Maps of the car park of INRIA. (a) The OG model contains $393,126$ cells, while the WOG model contains $78,742$ cells. (b) The OG model reconstructed from the previous WOG model; it can be seen that significant details such as beacons have been captured by the WOG representation (the shapes of maps (a) and (b) are very close). (c) The empty space is pretty well approximated at the coarser scale.

could be transmitted to the high level object motion prediction module. One of the possibilities, we explore now is to extract coherent zones of the BOF in term of occupancy and velocity and that we call them objects. Our hope is that these zones would appear hierarchically through the scales in the wavelet representation, and so woubld be easy to find.

3.3 Medium-Term Motion Prediction

Motion planning for dynamic environments is a very active research domain. Because the problem is NP-Hard [18], most of the research effort has been directed towards finding algorithms that are able to cope with this complexity. There is, however, another aspect of motion planning that is often overlooked despite its importance: motion planning algorithms need to know in advance the motion of the objects which populate the environment.

The problem is that, in most real applications, the future motion of the moving objects is a priori unknown, making it necessary to predict it on the basis of observations of the objects' past and present states. These observations are gathered using various sensors (*eg* radars, vision systems, etc.) which have limited precision and accuracy.

Until recently, most motion prediction techniques have been based on kinematic or dynamic models that describe how the state (*eg* position and velocity) of an object evolves over time when it is subject to a given control (*eg* acceleration) (cf. [19]). These approaches proceed by estimating the state, using techniques such as the Kalman Filter [20], and then applying the estimate to its motion equations in order to get state predictions.

Although these techniques are able to produce very good short-term predictions, their performance degrades quickly as they try to see further away in the future. This is especially true for humans, vehicles, robots, animals and the like, which are able to modify their trajectory according to factors (*eg* perception, internal state, intentions, etc.) which are not described by their kinematic or dynamic properties.

To address this issue, a different family of approaches has emerged recently. It is based on the idea that, for a given area, moving objects tend to follow typical motion patterns that depend on the objects' nature and the structure of the environment. Such approaches operate in two stages:

1. *Learning stage:* observe the moving objects in the workspace in order to determine the typical motion patterns.
2. *Prediction stage:* use the learnt typical motion patterns to predict the future motion of a given object.

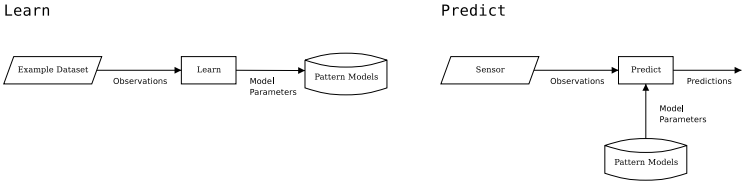


Fig. 3.9. Learn then predict

Thus, learning consists in observing a given environment in order to construct a representation of every possible motion pattern. But, how long should we observe the environment in order to construct such a "pattern library"? Given the enormous number of possible patterns for all but the simplest environments, there is not a simple answer. This raises an important problem of existing learning techniques [21, 22, 23]: they use a "learn then predict" approach, meaning that the system goes through a learning stage where it is presented with a set of observations (an example dataset) from which it builds its pattern models. Then, the models are "frozen" and the system goes into the prediction stage.

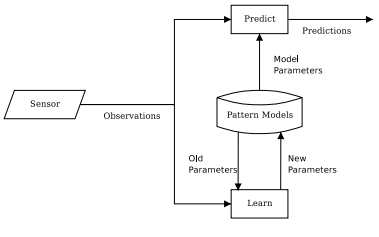


Fig. 3.10. Learn and Predict

The problem with this approach is that it makes the implicit assumption that all possible motion patterns are included in the example dataset, which, as we have shown, is a difficult condition to meet. In this paper we present an approach which, in contrast to the approaches mentioned above , works in a "learn and predict" fashion (fig. 3.10). That is, learning is an incremental process, which continuously refines knowledge on the basis of new observations which are also

used for prediction. To the extent of our knowledge, this is the first learning based motion prediction technique in the literature to have this property.

Our work is based on Hidden Markov Models [24], a probabilistic framework used to describe dynamic systems which has been successfully applied to "learn then predict" approaches [22, 23]. A Hidden Markov Model (HMM) may be viewed as a graph, where nodes represent states (*eg* places in the environment) and edges represent the transition probability of moving from one node to another in a single time step, the number of nodes and the existence of nodes determines the model's structure. The model also assumes that states are not directly observable but produce observations with a given probability.

Our approach is founded on the hypothesis that objects move in order to reach specific places of the environment called goals. Hence, motion patterns are represented using a different Hidden Markov Model (HMM) for every goal. Each such HMM describes how objects move to reach the corresponding goal. It is this set of HMMs that are used for prediction purposes.

Our main contribution is a novel approximate HMM learning technique based on the Growing Neural Gas algorithm [25]. It is able to process observations incrementally (*ie* one by one), in order to find the HMM's structure as well as to estimate the model's transition probabilities. Moreover, the algorithm is adaptive, meaning that it is able to insert or delete states or transitions in order to respond to changes in the underlying phenomenon. The same incrementallity and adaptivity properties apply to goal identification, hence, it is able to create models for motion patterns that have just been discovered or to delete old ones when the corresponding patterns are no longer observed. The cost of learning for each iteration is linear with respect to the number of states, as is the inference (*ie* prediction). This enables real-time processing, which is indispensable for a "learn and predict" approach.

3.3.1 Theoretical Framework

This section provides a gentle introduction to the two main tools which are used by our approach. Readers which are already familiar with the Hidden Markov Models or the Growin Neural Gas algorithm may safely skip the corresponding sections and proceed directly to §3.3.2.

Hidden Markov Models

This is a concise discussion on Hidden Markov Models (HMM), the interested reader is referred to [24] for an excellent tutorial on the subject.

An HMM may be viewed as a stochastic automaton which describes the temporal evolution of a process through a finite number of discrete states. The process progresses in discrete time steps, going from one state into another according to a given *transition probability*. It is assumed that the current state of the system is not directly observable, instead, it produces an output (i.e. observation) with a certain probability known as the *observation probability*.

Representation

An HMM describes the system using three stochastic variables: a) the present state s_t, b) the previous state s_{t-1}, and the current observation o_t, the rest of the model is defined by the following elements:

- The number of discrete states in the model N. A discrete state is denoted by its number $i : 1 \leq i \leq N$.
- The transition probability function, expressed by $P(s_t \mid s_{t-1})$. This probability is represented with a $N \times N$ *transition matrix* A where each element $a_{i,j}$ represents the probability of reaching the state j in the next time step given that the system was in state i.

$$a_{i,j} = P([s_t = j] \mid [s_{t-1} = i]) \tag{3.16}$$

- The observation probability function, expressed by $P(o_t \mid s_t)$. In general, for each state, the observation probability is represented by a gaussian distribution[4].

$$P(o_t \mid [s_t = i]) = \mathbb{G}(o_t; \mu_i, \sigma_i) \tag{3.17}$$

  The set of all the gaussians' parameters is denoted by $B = \{(\mu_1, \sigma_1), \cdots, (\mu_N, \sigma_N)\}$.
- The initial state distribution, expressed by $P(s_0)$. This represents our knowledge about the state of the system before receiving any observation and is denoted by Π. It is often represented by a uniform distribution $P([s_0 = i]) = \frac{1}{N}$ or by a $N \times 1$ matrix where $P([s_0 = i]) = \Pi_i$.

The three probability functions defined above form a Joint Probability Function (JPD) which encodes two conditional independence assumptions: a) knowing the state, observations do not depend on each other, and b) knowing the present state, the past and future states are mutually independent (eq. 3.18).

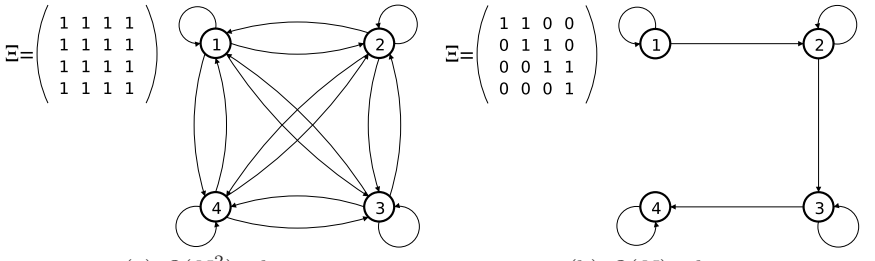
$$P(s_{t-1}, s_t, o_t) = P(s_{t-1})P(s_t \mid s_{t-1})P(o_t \mid s_t) \tag{3.18}$$

The parameters of these three probabilities are often denoted using the compact notation $\lambda = \{A, B, \Pi\}$, they are known together as the *model's parameters*, as opposed to the *model's structure*, which we will now discuss.

The structure of a model, Ξ, is the specification of its number of states N and the valid transitions between states. It may be visualized using the model's connectivity graph, where each vertex represents one state, vertices are joined by directed edges such that every strictly positive element of A ($a_{i,j} > 0$) will be represented by a directed edge from vertex i to vertex j. Hence, a matrix having only strictly positive elements corresponds to a fully connected, or *ergodic*, structure graph, while a sparse matrix corresponds to a graph having fewer edges. This is illustrated in fig. 3.11.

[4] It is also common to represent it by a mixture of gaussians, or, for discrete observations, by a table.

Structure is important because it affects the complexity of inference in the model, for example, filtering (state estimation) has complexity $O(N^2)$ for a fully connected structure graph, however, if the graph is sparse and each state has at most P predecessors, the complexity is $O(NP)$ [26]. Moreover, structure also influences the quality of inference [27, 28].



(a) $O(N^2)$ edges (b) $O(N)$ edges

Fig. 3.11. Two examples of an order 4 structure graph

Inference

HMM's are used to perform bayesian inference: compute the probability distributions of unknown variables given the known ones. Like in other Dynamic Bayesian Networks, inference is often used in HMM's to perform filtering $P(s_t \mid o_{1:t})$, smoothing $P(s_t \mid o_{1:T})$ and prediction $P(s_{t+K} \mid o_{1:t})$. Other two probabilistic questions which are more specific to HMM's are: a) evaluating the probability of a sequence of observations, and b) finding the sequence of states that is more likely to correspond to a sequence of observations. This last question is answered using a dynamic programming technique known as the Viterbi Algorithm [29].

Learning

In order to perform inference with an HMM, it is necessary to define its structure Ξ as well as its parameters λ. But, how to choose these values for a particular application? The solution is to estimate (*ie* learn) these values from data.

Most approaches in the literature assume that the structure is known and only try to learn the model's parameters λ. The most widely used of these *parameter learning* approaches is the Baum-Welch algorithm which is an application of Expectation-Maximization [30].

Despite being an active research subject, *structure learning*, no structure learning algorithm has become standard, probably the most popular are [31] and [32].

Growing Neural Gas

We will briefly introduce the Growing Neural Gas (GNG) algorithm, which is explained in detail in [25]. The Growing Neural Gas is an unsupervised competitive learning algorithm which may be applied to a variety of problems

(e.g. vector quantization, topology learning, pattern classification and clustering). It processes input vectors and constructs a network of N elements called *units*, each of these units has an associated D-dimensional vector called its *weight* (w_i) and is linked to other units called its *neighbors* $\mathfrak{N}_i$. The algorithm starts with two units linked together, then, as new input vectors are processed, units and links may be added or deleted to the network.

The algorithm implicitly partitions the whole space into N *Voronoi regions* V_i which are defined by:

$$V_i = \{x \in \mathbb{R}^D : \|x - u_i\| \leq \ \|x - u_j\|, \forall j \neq i\} \tag{3.19}$$

We illustrate this in 2-dimensional space (fig. 3.12). Each unit occupies its own Voronoi region, given an input vector, the winning unit is the one having its weight vector in the same Voronoi region.

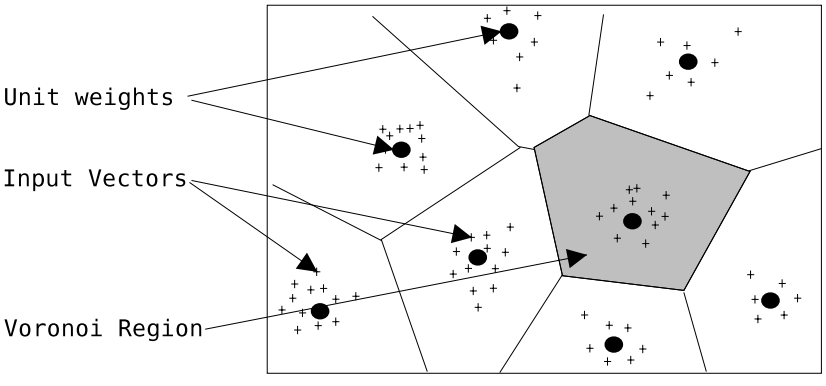


Fig. 3.12. Implicit partition: there are some 2-dimensional input vectors (crosses). The units' weights are represented by points and Voronoi regions are separated by boundary lines.

During learning, the units' weights are modified in order to minimize the *distortion* which is the mean distance between the winners and their corresponding input vectors. The algorithms also builds incrementally a topology of links which is a subset of the Delaunay triangulation of the units' weight vectors. This means that, in order to be linked together, two units must have a common border in the Voronoi region (see fig. 3.13).

Recapitulating, the GNG algorithm has the following properties:

1. No previous knowledge about the number of units N is required.
2. It is incremental: the model is updated by processing input in a one by one basis.
3. It is adaptive: units and / or links may be added or deleted to reflect changes in the underlying phenomenon.

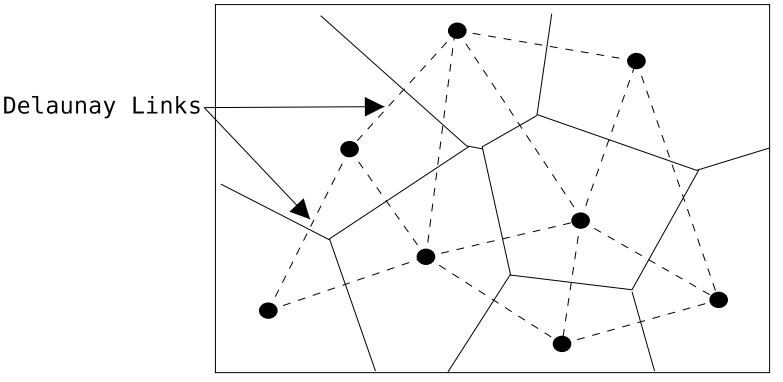


Fig. 3.13. The Delaunay triangulation for the previous example. Delaunay links are represented by dashed lines. Notice how the number of links corresponds to the number of borders for a given area.

4. It minimizes the distorion or quantization error: units' weights are placed in order to minimize their distance to input vectors.
5. Links are a subset of the Delaunay triangulation: they connect neighbor Voronoi cells.

3.3.2 Proposed Approach

The approach we propose consists of an algorithm which is able to continuously predict future motion based on a model which is constantly improved on the basis of observed motion. This *learn and predict* capability constitutes an advantage over existing techniques, making it unnecessary to have a learning dataset containing at least one example of every observable motion pattern.

The input of our algorithm consists of position observations $o_t = (x_t, y_t)$ and a trajectory termination flag η_t which is set to one when the observation corresponds to the end of a trajectory (*ie* when the object has stopped moving for a long time or exited the environment) and is set to zero otherwise. At every time step, this input is used to compute a probabilistic estimate of the state with a lookahead of K timesteps ($P(s_{t+k} \mid o_t)$) as well as to update the model (fig. 3.10).

Representation

Our approach is based on the hypothesis that objects move in order to reach specific places in the environment (*ie* goals). The idea is to identify all the possible goals. Then, for each of these goals, we construct a Hidden Markov Model (*ie* motion model) representing how an object should move in order to reach it.

It is assumed that transition probabilities depend on the goal to be reached (denoted by γ hereafter) and that structure and observation probabilities are independent of the goals, hence, they are the same for all motion models.

These assumptions lead to the following JPD:

$$P(s_{t-1}, s_t, o_t, \gamma) = P(s_{t-1})P(\gamma)P(s_t \mid s_{t-1}, \gamma)P(o_t \mid s_t) \tag{3.20}$$

So our model is defined by the following:

- The number of goals in the model G.
- The number of states in the model N.
- The transition probability function $P(s_t \mid \gamma, s_{t-1})$. This probability is represented with an unnormalized $G \times N \times N$ transition matrix[5] A where every element $a_{g,i,j}$ represents the number of times that a transition from state i to state j has been observed, given that the object is going to goal g.

$$a_{g,i,j} = P([s_t = j] \mid [\gamma = g][s_{t-1} = i]) \tag{3.21}$$

- The observation probability function $P(o_t \mid s_t)$. Is represented by gaussian distributions. The set of all the gaussians' parameters is denoted by $B = \{(\mu_1, \sigma_1), \cdots, (\mu_N, \sigma_N)\}$.
- The initial goal distribution $P(\gamma_0)$. Is considered to be a uniform distribution $P([s_0 = i]) = \frac{1}{G}, \forall i \in [1, N]$.
- The initial state distribution $P(s_0)$. Is considered to be a uniform distribution $P([s_0 = i]) = \frac{1}{N}, \forall i \in [1, N]$.

The set of *pattern models* shown in fig. 3.10 consists of structure and parameters. The structure is defined by the number of goals G, the number of states N, and the structure graph Ξ which is the same for all the A_g matrices.

The parameters consists of the transition table A, and the observation parameters B. Given that both $P(s_0)$ and $P(\gamma)$ are considered to be uniform, they do not require any extra parameter.

Learning

Learning is composed of several subprocesses, an overview is presented in fig. 3.14.

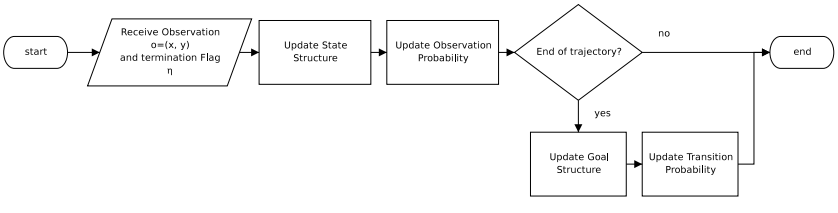


Fig. 3.14. Learning Overview

Updating State Structure

The input observation o_t is passed as an input vector to a GNG network $\mathfrak{G}_{state}$. Nodes of this network represent states, and links represent allowed transitions

[5] This is considered equivalent to having G $N \times N$ matrices $A_g : 1 \leq g \leq G$.

(*ie* each link represents two transitions, corresponding to the two possible directions). This network is used to update A as follows:

1. If a new unit has been inserted or deleted in $\mathfrak{G}_{state}$, a corresponding row and column are inserted to or deleted from all A_g matrices.
2. If a new link has been added or deleted from $\mathfrak{G}_{state}$, two corresponding transitions are added to or deleted from all A_g.

Updating Observation Probabilities

The observation probabilities are directly computed from $\mathfrak{G}_{state}$. Unit weights are used as mean values for the gaussians' centers:

$$\mu_i = w_i, \forall w_i \in \mathfrak{G}_{state} \tag{3.22}$$

And σ_i is estimated from the average distance to unit's neighbors:

$$\sigma_i = \frac{C}{|\mathfrak{N}_i|} \sum_{j \in \mathfrak{N}_i} \|w_i - w_j\| \tag{3.23}$$

Where C is a weighting constant, in our experiments, we have set it to 0.5.

Updating Goal Structure

Goals are discovered by clustering together observations that correspond to trajectories endpoints, which is indicated by $\eta = 1$. Given that the number of goals is ignored, we will use another GNG structure $\mathfrak{G}_{goal}$ to perform the clustering. The number of clusters G corresponds to the number of units in $\mathfrak{G}_{goal}$. This means that, when a new unit is inserted or deleted in $\mathfrak{G}_{goal}$ a corresponding slice (*ie* matrix) is inserted to or deleted from A.

Updating Transition Probabilities

In this step, the entire observation sequence $\{o_1, \cdots, o_T\}$ for a trajectory is used, this means that it is necessary to store all past observations in an internal data structure.

In order to choose the transition probability table to update, the attained goal γ should be identified, this is done by choosing the goal which is closest to the last observation.

$$\gamma = \arg\min_i \|o_T - w_i\|, \forall w_i \in \mathfrak{G}_{goal} \tag{3.24}$$

Transition probabilities are then updated by applying maximum likelihood: we use the Viterbi algorithm to find the most likely sequence of states $\{s_1, \cdots, s_T\}$ and then, we increment the corresponding counters in A[6].

$$a_{\gamma, s_{t-1}, s_t} = a_{\gamma, s_{t-1}, s_t} + 1, \forall t : 1 \leq t \leq T - 1 \tag{3.25}$$

[6] This contrasts from Baum-Welch which performs weighted counting according to the likelihood of every *possible* state sequence.

A problem with this approach is that the transition counters of new links in the topology will have a much smaller value than those that correspond to older links, hence, old links will dominate. This is solved by multiplying transition weights by a *fading factor* f. In general this is similar to having a bounded counter with a maximum value of $\frac{1}{1-f}$.

$$a_{\gamma,s_t,i} = a_{\gamma,s_t,i} \times f, \forall t, i : 1 \leq t \leq T-1, i \in \mathfrak{N}_{s_t} \tag{3.26}$$

Instead of normalizing A, normalization is performed when computing the transition probability.

$$P([s_t = j] \mid [\gamma = g][s_{t-1} = i]) = \frac{a_{g,i,j}}{\sum_{k \in \mathfrak{N}_i} a_{g,i,k}} \tag{3.27}$$

Finally, the data structure that was used to store the observation sequence may be cleared.

Prediction

Prediction is performed in two steps. First, the belief state (*ie* the joint probability of the present state and goal) is updated using the input observation o_t:

$$P(\gamma, s_t \mid o_t) = \frac{1}{Z} P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t \mid \gamma, s_{t-1}) P(\gamma, s_{t-1} \mid o_{t-1}) \tag{3.28}$$

Where $P(\gamma, s_{t-1} \mid o_{t-1})$ is the belief state calculated in the previous time step.

Then, motion state is predicted with a look-ahead of K time steps using:

$$P(\gamma, s_{t+K} \mid o_t) = \sum_{s_{t+K-1}} P(s_{t+K} \mid \gamma, s_{t+K-1}) P(\gamma, s_{t+K-1} \mid o_t) \tag{3.29}$$

Finally, the state is obtained by marginalizing over the goal:

$$P(s_{t+K} \mid o_t) = \sum_{\gamma} P(\gamma, s_{t+K} \mid o_t) \tag{3.30}$$

An alternative to state prediction is to predict the goal by marginalizing the state from the belief state:

$$P(\gamma \mid o_t) = \sum_{s_t} P(\gamma, s_t \mid o_t). \tag{3.31}$$

3.3.3 Experimental Results

Test Environment

The environment we have chosen to validate our approach is the main lobby of our research institute. It features the main entrance to the building, a self-information directory post, the front-desk, a cafeteria area and a number of doors

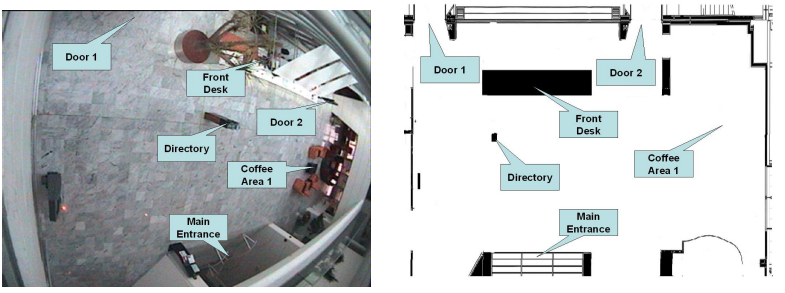


Fig. 3.15. Inria's main lobby: video view (left) and 2D map (right)

leading to various halls, rooms and auditoriums (Fig. 3.15). This environment is the heart of the institute, all the personnel passes through it at some point during the day for a reason or another (going in or out, coffee break, attending a lecture, etc). This environment is interesting because the motion patterns of the people is rich and the flow of people sufficient to ensure that it will be possible to gather a significant number of observations.

The testing of our approach has been done in two stages. First, we have used observation data coming from a software simulating the trajectories of people in the main lobby. Then we have used live observations coming from a visual tracking system.

The interest of using simulated data is that it allows to evaluate our approach in controlled conditions for which it is possible for instance to predefine the number of motion patterns.

In both cases, we have gathered a significant number of observations. We have presented 1000 trajectories to the learning algorithms in order to build an initial model, before starting to measure the results. Then, prediction performed has been measured using another 300 trajectories. The test results obtained with simulated data (resp. real data) are presented in section 3.3.3 (resp. 3.3.3).

Simulated Observations

Getting the Observations

The simulating system we have developed relies upon a number of control points representing "'places of interest" of the environment such as the doors, the front-desk, etc. Based upon this set of control points, a set of 32 typical motions patterns has been defined. Each motion pattern consists in a sequence of control points to be traversed. An observed trajectory is computed in the following way: first, a motion pattern is randomly chosen. This motion pattern provides a set of control points. Then, *goal points* corresponding to each of these control points are randomly generated using two-dimensional Gaussian distributions whose mean value are the control points. Finally, the motion between two goal points in the sequence is simulated using discrete, even-size steps in a direction drawn from a Gaussian distribution whose mean value is the direction of the next goal point.

Switching from one goal point to the next is done when the distance to the current goal point is below a predefined threshold.

We have generated the 1300 trajectories that were required for our experiments. an image of trajectories in the data set are presented in fig. 3.16.

Learning Results

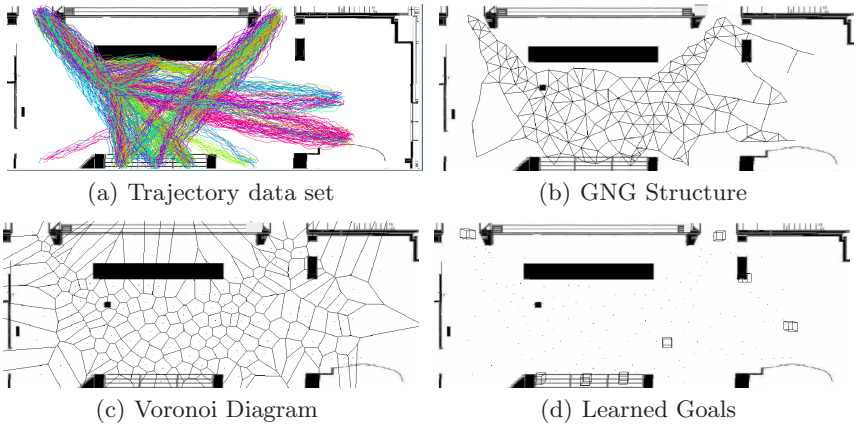
We have run our algorithm against the simulated data set 16(a). The algorithm took about 2 minutes to process the 1000 initial trajectories, which contained a total of 58,262 observations meaning an average processing frame rate of about 480 observations per second. As a result of the learning process, a structure having 196 states has been found (figs. 16(b) and 16(c)). Also a total of eight goals were identified, as it may be seen in fig. 16(d) the detected goals seem to correspond to many of the interesting points shown in fig. 3.15.

Prediction Results

Figure 3.17 illustrates the prediction process. The figures show the real future trajectory of an object, which is unknown for the system. The figures show the probability of reaching a particular goal (indicated by the size of the cubes) as well as the state probability in $t + 3$ seconds. For visual purposes this has been illustrated with particles, where a higher concentration of particles indicates a higher state probability.

In order to test prediction performance, we measure the distance between the predicted goal and the real final destination of the object (fig. 3.18). For each of the 300 trajectories we measure this distance when 10% of the total trajectory has been seen, then, we do the same for 20% of the total trajectory and so on until 90%.

We think that the obtained results are quite good. Even when only 30% of the total trajectory is used to predict, the mean error is of about 3 meters which



(a) Trajectory data set (b) GNG Structure

(c) Voronoi Diagram (d) Learned Goals

Fig. 3.16. Overview of the learning process (simulated data)

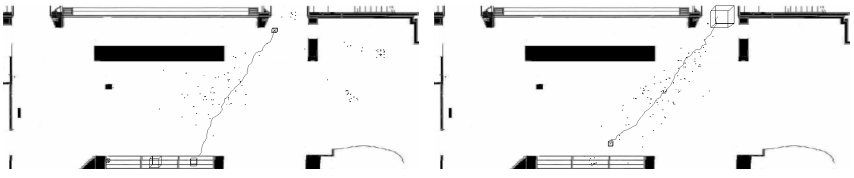


Fig. 3.17. Prediction Examples (simulated data)

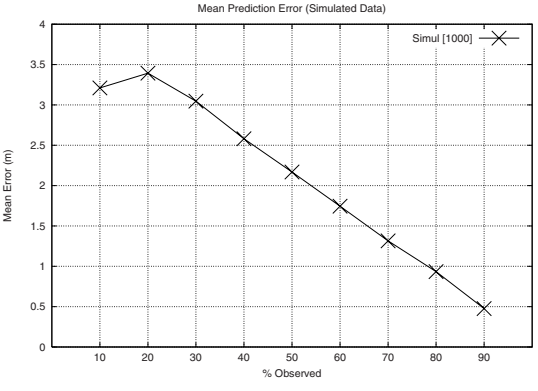


Fig. 3.18. Prediction Error (simulated data)

is relatively low with respect to the size of the environment and the distance between goals.

Real Observations

Getting the observations

To gather observations about the motions performed in the test environment, we use the visual tracking system proposed in [33]. This system detects and tracks objects moving in the images of a video stream. The information collected (position and size of the moving object, etc.) is then projected into the 2D map

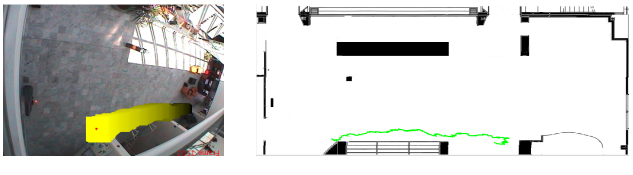
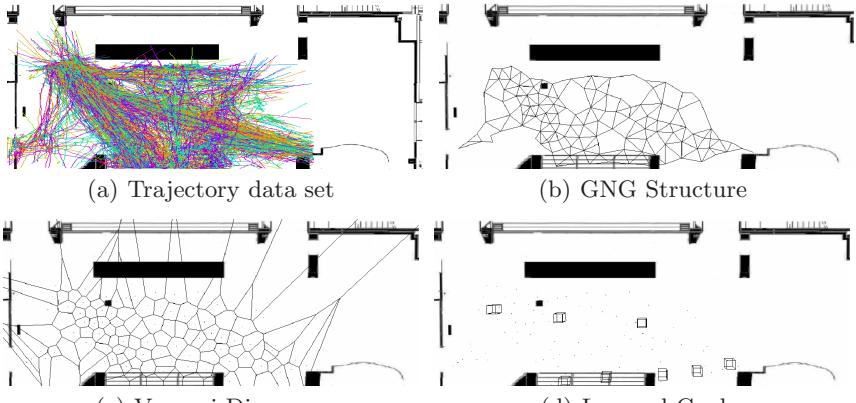


Fig. 3.19. Architecture of the visual tracking system (top). Example of a motion observed in the image (bottom-left), and its projection in the 2D map of the environment (bottom-right).

of the environment. The data flow of the overall tracking system features a detector-tracker, a module to correct the distortion of the video camera, and a final module to project the information in the 2D map of the floor.

Learning Results

Figure 20(a) shows the real data set that we have obtained. It is important to notice that trajectories in the data set do not cover the entire environment. This happens because of clipping due to the distortion correction algorithm we have used.



(a) Trajectory data set (b) GNG Structure

(c) Voronoi Diagram (d) Learned Goals

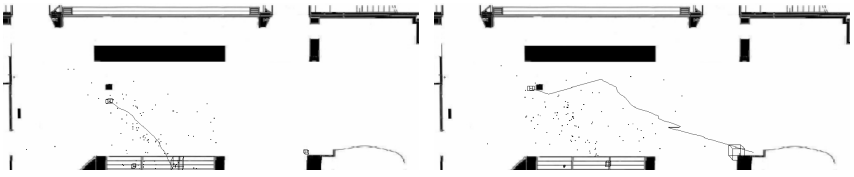
Fig. 3.20. Overview of the learning process (real data)

The algorithm took about 70 seconds to process the 36,444 observations of the first 1000 trajectories in the data set for an average of about 520 observations per second. The found structure (figs. 20(b) and 20(c)) consisted of 123 states. The algorithm has detected eight goals, shown in fig. 20(d). In this case, some of the goals do not correspond to interesting points of the environments. This happens because of two factors: a) clipping reduced the environment an creates some fake entry and exit points, and b) the tracking system sometimes losses the identifier of an object resulting in single trajectories being broken down in several smaller trajectories.

Prediction Results

Figure 3.21 illustrates prediction based on real data. The example in the right is especially interesting because, although state prediction does not seem to correspond to the real trajectory, the goal has been correctly predicted.

In fig. 3.22 we present the results of our prediction performance measure (see §3.3.3). Here, the results are even better than for simulated data, which may be surprising at first. The reason is again clipping, which reduces the effective size of the environment, thus bringing goals closer together. In the other hand, the noisy nature of real data and is reflected by the slower convergence of real data when compared with simulated one.



(a) (b)

Fig. 3.21. Prediction Examples (real data)

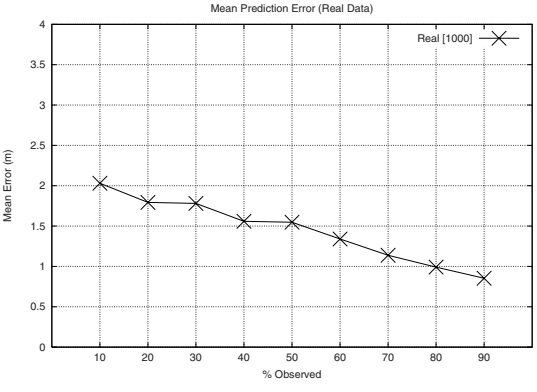


Fig. 3.22. Prediction Error (real data)

3.4 Safe Navigation in Dynamic Environments

When placed in a dynamic environment, an autonomous system must consider the real time constraint that such an environment imposes. Indeed, the system has a limited time only to make a decision about its future course of action otherwise it might be in danger by the sole fact of being passive. The time available depends upon a factor that we will call the *dynamicity* of the environment and which is a function of the system and the moving objects'dynamics.

In this context, basing the decision making process on a motion planning technique[7] leaves little hope to fulfil this real-time constraint given the intrinsic time complexity of the motion planning problem [34] (even if using randomised approaches). This certainly explain why so many reactive methods[8] have been developed in the past (*cf* [35], [36], [37], [38], [39], [40], [41], [42], [43], [44] or [45]. However, reactive approaches are confronted with two key issues: the *convergence* and the *safety* issues. As for convergence, their lack of lookahead may prevent the system to ever reach its goal. As for safety, what guarantee is there that the system will never find itself in a dangerous situation eventually yielding to a collision?

[7] Wherein a complete motion to the goal is computed a priori.
[8] Wherein only the next action is determined at each time step.

Partial Motion Planning (PMP) is the answer we propose to the problem of navigation in dynamic environments. It is especially designed in order to take into account the real-time constraint mentioned above. PMP is a motion planning scheme with an anytime flavor: when the time available is over, PMP returns the best partial motion to the goal computed so far. Like reactive scheme, PMP is also confronted to the convergence and safety issues. At this point, we have decided to focus on the safety issue and to propose a solution relying upon the the concept of *Inevitable Collision States* (ICS) originally introduced in [46]. An ICS is a state such that no matter what the future motion of the system is, it eventually collides with an obstacle. ICS takes into account the dynamics of both the system and the moving obstacles. By computing ICS-free partial motion, the system safety can be guaranteed.

PMP is detailed in section 3.4.1 while section 3.4.2 presents the ICS concept. Finally, an application of PMP to the case of a car-like system in a dynamic environment is presented in section 3.4.3.

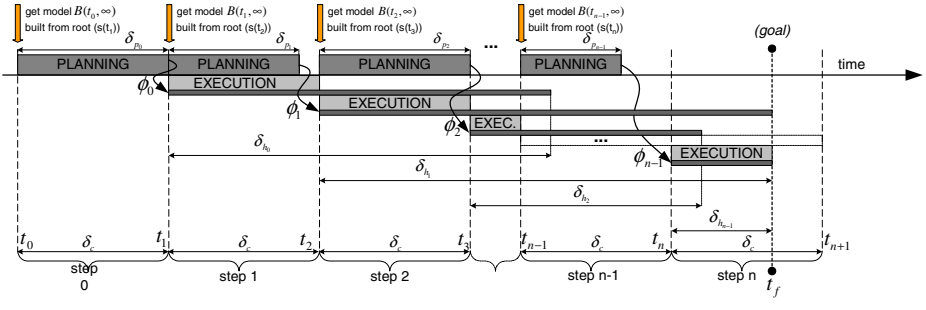


Fig. 3.23. Partial Motion Planning iterative cycle

3.4.1 Partial Motion Planning

As mentioned earlier, a robotic system cannot in general safely stand still in a dynamic environment (it might be collided by a moving obstacle). It has to plan a motion within a bounded time and then execute it in order to remain safe. The time δ_c available to calculate a new motion is function of the nature and dynamicity of the environment. To take into account the real-time constraint that stems from a dynamic environment, we propose a scheme that calculates partial motions only according to the the following cycle (also depicted in Fig. 3.23):

PMP Algorithm	
Step1:	*Get model of the future*
Step2:	*Build tree of partial motions towards the goal*
Step3:	*When δ_c is over, return best partial motion*
Step4:	*Repeat until goal is reached*

Like motion planning, partial motion planning requires a model of the environment, the first step is aimed at getting this model. The required model is provided

by the environment modelling and motion prediction functions presented earlier. The periodic iterative PMP scheme proposed in this paper accounts for both the planning time constraints and the validity duration of the predictions made.

3.4.2 Inevitable Collision States

Like every method that computes partial motion only, PMP has to face a safety issue: since PMP has no control over the duration of the partial trajectory that is computed, what guarantee do we have that the robot $\mathcal{A}$ will never end up in a critical situation yielding an inevitable collision? As per [46], an Inevitable Collision State (ICS) is defined as a state s for which no matter the control applied to the system is, there is no trajectory for which the system can avoid a collision in the future. The answer we propose to the safety problem lies then in the very fact that the partial trajectory that is computed is ICS-free. Meaning that, even in the worst case scenario where the duration δ_{h_i} of the partial trajectory is shorter than the cycle time δ_c, $\mathcal{A}$ can always execute one of the existing safe trajectory. The overall safety is guaranteed as long as the initial state is ICS-free (which is something that can be reasonably assumed). Now, determining whether a given state of $\mathcal{A}$ is an ICS or not is a complex task since it requires to consider all possible future trajectories for $\mathcal{A}$. However, it is possible to take advantage of the approximation property demonstrated in [46] in order to compute a conservative approximation of the set of ICS. This is done by considering only a subset $\mathcal{I}$ of the full set of possible future trajectories.

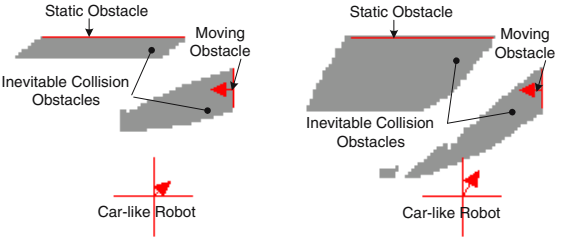
3.4.3 Case Study

In this section we present the application of PMP to the case of a car-like vehicle $\mathcal{A}$ moving on a *planar* surface $\mathcal{W}$ and within a fully observable environment cluttered with stationary and dynamic obstacles. A control of $\mathcal{A}$ is defined by the couple (α, γ) where α is the rear wheel linear acceleration. and γ the steering velocity. The motion of $\mathcal{A}$ is governed by the following differential equations:

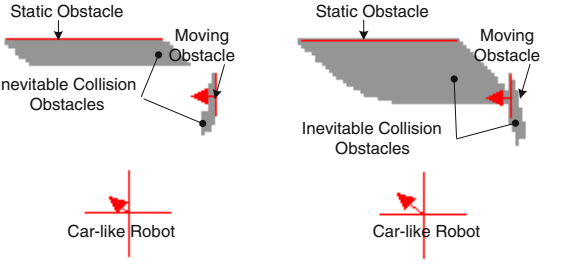
$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v}_r \\ \dot{\xi} \end{bmatrix} = \begin{bmatrix} v_r \cos\theta \\ v_r \sin\theta \\ \frac{\tan\xi v_r}{L} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \alpha + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \gamma \tag{3.32}$$

with $\alpha \in [\alpha_{min}, \alpha_{max}]$ (acceleration bounds), $\gamma \in [\gamma_{min}, \gamma_{max}]$ (steering velocity bounds), and $|\xi| \leq \xi_{max}$ (steering angle bounds). L is the wheelbase of $\mathcal{A}$.

For practical reasons, the duration of the trajectories of $\mathcal{I}$ has to be limited to a given *time horizon* that determines the overall level of safety of $\mathcal{A}$. In our case, the subset $\mathcal{I}$ considered in order to compute a conservative approximation of the set of ICS includes the braking trajectories with a constant control selected from $[(\alpha_{min}, \dot{\xi}_{max}), (\alpha_{min}, 0), (\alpha_{min}, \dot{\xi}_{min})]$, and applied over the time necessary for $\mathcal{A}$ to stop.



(a) $(\theta, v) = (0.9, 1.0)$ (b) $(\theta, v) = (0.9, 2.0)$



(c) $(\theta, v) = (2.2, 1.0)$ (d) $(\theta, v) = (2.2, 2.0)$

Fig. 3.24. (θ, v)-slices of the state space of $\mathcal{A}$. Shaded regions are ICS respectively defined for the braking trajectory of control $(\alpha_{min}, \dot{\xi}_{min})$ (top), and all braking trajectories with controls selected from $[(\alpha_{min}, \dot{\xi}_{max}), (\alpha_{min}, 0), (\alpha_{min}, \dot{\xi}_{min})]$ (bottom).

Fig. 3.24 depicts the ICS obtained when different set of braking trajectories are considered. Each subfigure represents a (θ, v)-slice of the full 5D state space of $\mathcal{A}$. In the top subfigures, only the braking trajectory of control $(\alpha_{min}, \dot{\xi}_{min})$ is considered. In the bottom subfigures, the three braking trajectories are considered.

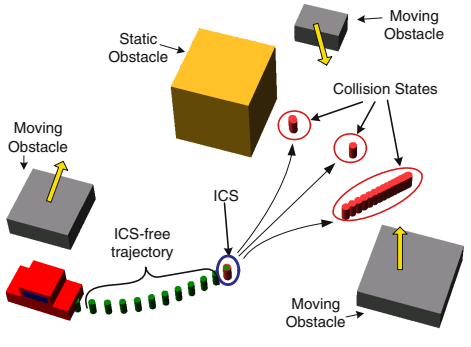


Fig. 3.25. The state labelled ICS is an ICS since the three braking trajectories issued from it yield collisions

In PMP, checking whether a state is an ICS or not is carried out by testing if all the braking trajectories yield a collision with one of the moving obstacles. If so, the state is an ICS. In fig. 3.25), the collision states in red represent the collision that will occur in the future from this state for all trajectories of $\mathcal{I}$. In this case, since all trajectories collide in the future, this state is an ICS. In PMP, every new state is similarly checked to be an ICS or not over $\mathcal{I}$. In case all the trajectories appear to be in collision in the future, this state is an ICS and is not selected.

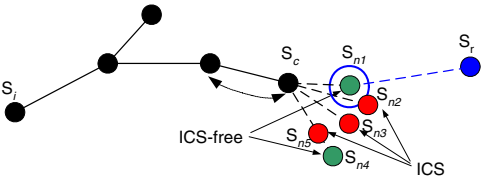
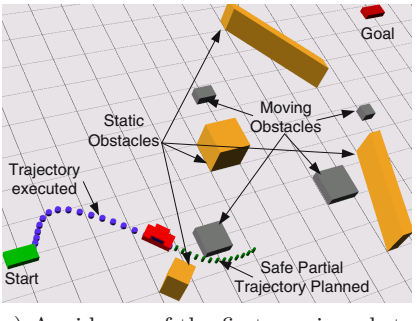
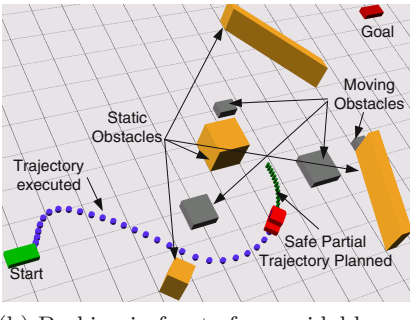


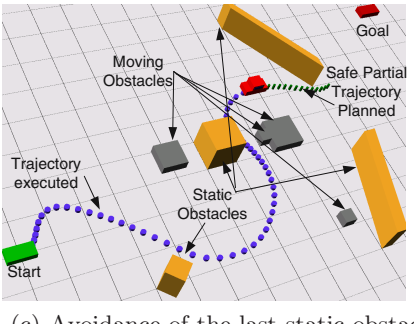
Fig. 3.26. Search tree construction principle



(a) Avoidance of the first moving obstacle



(b) Braking in front of unavoidable moving obstacles



(c) Avoidance of the last static obstacle

Fig. 3.27. Results of a 2D safe Partial Motion Planning ($\delta_c = 1s$, $v_{max} = 2.0m/s$, $\xi_{max} = \pi/3rad$, $\dot{\xi}_{max} = 0.2rad/s$, $\alpha_{max} = 0.1m/s^2$)

The exploration method used is the well known Rapidly-Exploring Random Tree method (RRT) [47]. RRT incrementally builds a tree in the state space of $\mathcal{A}$. The basic principle of RRT is depicted in Fig. 3.26. A state s_r is randomly selected first. Then, the closest node in the tree, say s_c, is determined. Constant controls selected from $\mathcal{U}_{2D}$=[$(\alpha_{max}, 0)$; $(\alpha_{max}, \dot{\xi}_{max})$; $(\alpha_{max}, \dot{\xi}_{min})$;$(0, \dot{\xi}_{max})$; $(0, 0)$; $(0, \dot{\xi}_{min})$;$(\alpha_{min}, \dot{\xi}_{max})$; $(\alpha_{min}, 0)$; $(\alpha_{min}, \dot{\xi}_{min})$] are then applied to s_c for a duration ϵ, they yield a set of candidate trajectories ending in given states s_{ni}. These candidate trajectories are pruned out: only are kept the trajectories that are collision-free and whose final state is ICS-free (as per property 2, such trajectories are ICS-free). Finally, the trajectory whose final state is closer to s_r is selected and added up to the tree. This process is repeated until the end of the time available where the best partial trajectory extracted from the tree is returned.

In Fig. 3.27 we can see an example of a navigation from a still starting state (green box) to a still goal state (red box). The environment is cluttered with moving (grey) and static (orange) obstacles. In 27(a) one can observe how the safe partial trajectory (green) is calculated and planned within the time-state space in order to avoid the obstacle moving upward. The states in blue behind the car, is the trajectory, built from partial trajectories from the previous PMP cycles and (ideally) executed by the robot. In 27(b) we can observe that the car was obliged to slow down at the intersection of several obstacles, since no other safe trajectories could be found, before to re-accelerate. In 27(c) the system has planned a partial trajectory that avoids the last static obstacle.

3.5 Conclusion

This paper addressed the problem of navigating safely in a open and dynamic environment sensed using both on-board and external sensors. After a short presentation of the context and of the related open problems, we focused on two complementary questions: how to interpret and to predict the motions and the behaviors of the sensed moving entities ? how to take appropriate goal-oriented navigation decisions in such a rapidly changing and sensed environment?

In order to answer these questions, we have proposed an approach including three main complementary functions: (1) Scene interpretation and short-term motion prediction for the sensed potential obstacles, using the "Bayesian Occupancy Filtering" approach (BOF) (2) Medium-term motion and behavior prediction for the observed entities, using motion pattern learning and hierarchical Hidden Markov Models; (3) On-line goal-oriented navigation decision in a dynamic environment, using the "Partial Motion Planning" paradigm (PMP).

The first function (BOF)has experimentally been validated on our experimental vehicle (the Cycab), for avoiding partially observed moving obstacles. A scenario involving the Cycab, a moving pedestrian, and a parked car which temporarily hide the pedestrian to the sensors of the Cycab, has successfully been executed. In this experiment, the avoidance behavior has been obtained by

combining the *occupancy probability* and the *danger probability* of each cell of the grid. The second function has experimentally been validated on some indoor data (the INRIA entry hall), using about 1000 tracked human trajectories for the initial learning phase. At the moment, the last function (PMP) has only been experimented in simulation.

Current work mainly deals with three major points: (1) Improvement of the prediction approaches for making it possible to cope with larger environments (such as complex urban traffic situations), while preserving the efficiency property; the current development on the WOG model is an example of this work. (2) Fusion of our current interpretation and prediction paradigms with higher-level information (e.g. GPS maps, moving entities properties, nominal behaviors ...) to better estimate the scene participants behaviors. (3) Integration of the three previous functions, and implementation and test this new navigation system on our experimental platform involving the INRIA car park, several Cycabs, and both inboard and infrastructure sensors.

Acknowledgements

This work has been partially supported by several national and international projects and funding bodies: European projects *Carsense, CyberCars & PreVent/ProFusion*; French Predit projects *Puvame & MobiVip*. The authors would like to thanks Pierre Bessiére, Christophe Couéé and Cedric Pradalier for their contributions and fruitful discussions.

References

1. C. Coué, C. Pradalier, and Laugier C. Bayesian programming for multi-target tracking: an automotive application. In *Proceedings of the International Conference on Field and Service Robotics*, Lake Yamanaka (JP), July 2003.
2. H.P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2), 1988.
3. A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer, Special Issue on Autonomous Intelligent Machines*, Juin 1989.
4. S. Thrun. Robotic mapping: A survey. In *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.
5. E. Prassler, J. Scholz, and A. Elfes. Tracking multiple moving objects for real-time robot navigation. *Autonomous Robots*, 8(2), 2000.
6. O. Lebeltel. *Programmation Bayésienne des Robots*. Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, Septembre 1999.
7. O. Lebeltel, P. Bessière, J. Diard, and E. Mazer. Bayesian robot programming. *Autonomous Robots*, 16:49–79, 2004.
8. C. Coué and P. Bessière. Chasing an elusive target with a mobile robot. In *Proceedings of the IEEE-RSJ International Conference on Intelligent Robots and Systems*, Hawai (HI), 2001.

9. A. H. Jazwinsky. *Stochastic Processes and Filtering Theory*. New York : Academic Press, 1970.
10. R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 35, Mars 1960.
11. G. Welch and G. Bishop. An introduction to the Kalman filter. available at http://www.cs.unc.edu/ welch/kalman/index.html.
12. I. Daubechies. *Ten Lectures on Wavelets*. Number 61 in CBMS-NSF Series in Applied Mathematics. SIAM Publications, Philadelphia, 1992.
13. Stéphane Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, San Diego, 1998.
14. D. K. Pai and L.-M. Reissell. Multiresolution rough terrain motion planning. In *IEEE Transactions on Robotics and Automation*, volume 1, pages 19–33, February 1998.
15. Bruno Sinopoli, Mario Micheli, Gianluca Donato, and T. John Koo. Vision based navigation for an unmanned aerial vehicle. In *Proceedings of the International Conference on Robotics and Automation*, May 2001.
16. Manuel Yguel, Olivier Aycard, and Christian Laugier. Internal report: Wavelet occupancy grids: a method for compact map building. Technical report, INRIA, 2005.
17. C. Pradalier and S. Sekhavat. Simultaneous localization and mapping using the geometric projection filter and correspondence graph matching. *Advanced Robotics*, 2004. To appear.
18. J.H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Symp. on the Foundations of Computer Science*, pages 144–154, Portland (OR) USA, October 1985.
19. Qiuming Zhu. A stochastic algorithm for obstacle motion prediction in visual guidance of robot motion. *The IEEE International Conference on Systems Engineering*, pages 216–219, 1990.
20. R.E. Kalman. A new approach to linear filtering and prediction problems. *Trans. Am. Soc. Mech. Eng., Series D, Journal of Basic Engineering*, 82:35–45, 1960.
21. Dizan Vasquez and Thierry Fraichard. Motion prediction for moving objects: a statistical approach. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 3931–3936, New Orleans, LA (US), apr 2004.
22. Sarah Osentoski, Victoria Manfredi, and Sridhar Mahadevan. Learning hierarchical models of activity. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.
23. Maren Bennewitz, Wolfram Burgard, Grzegorz Cielniak, and Sebastian Thrun. Learning motion patterns of people for compliant robot motion. *Internationl Journal of Robotics Research*, 24(1):31–48, January 2005.
24. Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.
25. Bernd Fritzke. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 1995.
26. Kevin Patrick Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley (USA), 2002.
27. Matthew Brand. Structure learning in conditional probability models via an entropic prior and parameter extinction. Technical report, MERL a Mitsubishi Electric Research Laboratory, 1998.

28. Henri Binsztok and Thierry Artières. Learning model structure from data: an application to on-line handwriting. *Electronic Letter on Computer Vision and Image Analyisis*, 5(2), 2005.
29. Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, April 1967.
30. N. Dempster, A.and Laird, , and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 9(1):1–38, 1977. Series B.
31. Nir Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proc. of the Fourteenth International Conference on Machine Learning*, pages 125–133, 1997.
32. Andreas Stolcke and Stephen Omohundro. Hidden markov model induction by bayesian model merging. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 11–18, Denver, USA, 1993. Morgan Kaufmann, San Mateo, CA.
33. A. Caporossi, D. Hall, P. Reignier, and J.L. Crowley. Robust visual tracking from dynamic control of processing. In *International Workshop on Performance Evaluation of Tracking and Surveillance*, pages 23–31, Prague, Czech Republic, May 2004.
34. J. H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Proc. of the IEEE Symp. on the Foundations of Computer Science*, pages 144–154, Portland, OR (US), October 1985.
35. R.C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, August 1989.
36. J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, June 1991.
37. R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995.
38. D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. Technical Report IAI-TR-95-13, 1 1995.
39. M. Khatib. *Sensor-based motion control for mobile robots.* PhD thesis, LAAS-CNRS December, 1996, 1996.
40. R. Simmons. The curvature velocity method for local obstacle avoidance. In *Proceedings of the International Conference on Robotics and Automation*, pages 3375–3382, Minneapolis (USA), april 1996.
41. P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17(7):760–772, July 1998.
42. N.Y. Ko and R. Simmons. The lane-curvature method for local obstacle avoidance. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Victoria (Canada), October 1998.
43. J. Minguez and L. Montano. Nearness diagram navigation (ND): A new real time collision avoidance approach for holonomic and no holonomic mobile robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, November 2000.
44. N. Roy and S. Thrun. Motion planning through policy search. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, 2002.

45. J. Minguez, L. Montano, and J. Santos-Victor. Reactive navigation for non-holonomic robots using the ego kinematic space. In *Proceedings IEEE International Conference on Robotics and Automation*, Washington (US), May 2002.
46. Thierry Fraichard and Hajime Asama. Inevitable collision states - a step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.
47. S. LaValle and J. Kuffner. Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 473–479, Detroit (US), May 1999.