Intel Serv Robotics (2008) 1:51–72
DOI 10.1007/s11370-007-0004-1

ORIGINAL RESEARCH

Geometric and Bayesian models for safe navigation in dynamic environments

C. Laugier · Dizan Vasquez · M. Yguel · Th. Fraichard · O. Aycard

Received: 2 March 2007 / Accepted: 25 March 2007 / Published online: 27 July 2007
© Springer-Verlag 2007

Abstract 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. Basically, these problems may be classified into three main categories: (a) SLAM in dynamic environments; (b) detection, characterization, and behavior prediction of the potential moving obstacles; and (c) online 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: *multiscale world representation* of static obstacles based on the wavelet occupancy grid; *adaptative clustering* for moving obstacle detection inspired on Kohonen networks and the growing neural gas algorithm; and *characterization and motion prediction* of the observed moving entities using Hidden Markov Models coupled with a novel algorithm for structure and parameter learning.

Keywords Multiscale occupancy grids · World state estimation · Online reconstruction · Motion prediction · Intelligent vehicles

1 Introduction

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 behavior is unknown. Taking into account these characteristics requires solving of 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 localization and mapping (SLAM) in dynamic environments;
- detection, tracking, identification and future behavior prediction of the moving obstacles;
- online 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 behavior (and consequently to make predictions).

From the autonomous navigation point 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 environment's dynamics), 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. Figure. 1 shows an overview of the whole system, along with the sys-

C. Laugier · D. Vasquez · M. Yguel · Th. Fraichard · O. Aycard
INRIA Rhône-Alpes & GRAVIR Lab. (CNRS, INPG, UJF),
Saint Ismier, France

D. Vasquez (✉)
Swiss Federal Institute of Technology, Zurich, Switzerland
e-mail: vasquez@mavt.ethz.ch

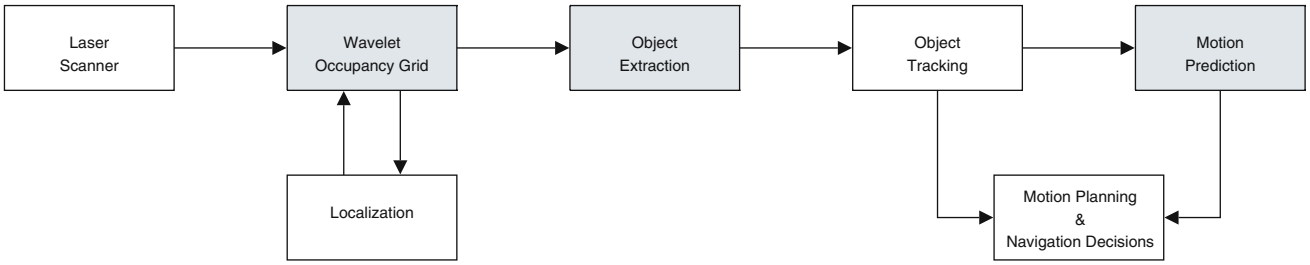


Fig. 1 System overview. *Shadowed blocks* indicate the components presented in this paper

tem components which are detailed in this paper. The solutions we have implemented rely on the following modules:

- *multiscale world representation* of static obstacles based on wavelet occupancy grid;
- *adaptive clustering* for moving obstacle detection inspired on Kohonen networks and the growing neural gas algorithm;
- *characterization and motion prediction* of the observed moving entities using incremental motion pattern learning in order to adapt hidden Markov models.

This paper is organized in four main sections. Section 2 describes how we use wavelets to build an occupancy grid based representation of the world. Section 3 deals with the process which goes from distinguishing between grid cells that belong to static and moving objects, up to tracking individual objects in space. Section 4 describes how tracking output is used to learn and predict typical motion patterns of the objects populating the environment. In Sect. 5, we describe our experimental platform—the automated valet parking—and discuss our experimental results. We finish the paper by presenting our conclusions.

2 Wavelet-Based world reconstruction

Occupancy grids (OG), [1], partition the workspace into a cartesian grid. Each cell of the grid stores the probability that an obstacle lies at the cell location. It provides robots with the ability to build accurate dense map of the static environment, which keeps track of all possible landmarks and represents open spaces and static obstacles at the same time. Only simple update mechanism, which filters moving obstacles naturally and performs sensor fusion, is required. However, this simplicity comes with a major drawback: to efficiently represent the world a huge amount of cells are needed. To counter this problem we propose, here, a new algorithm for a new representation that allows to keep advantages of a grid with a compact representation. This model is based upon a mathematical tool for sparse function coding called wavelets, that will be introduced now, along with our notations.

2.1 Wavelets

In this paper, the occupancy state is represented as a spatial function. Our main contribution is an occupancy function updating technique that can be performed in a compact manner. The mechanism behind several compression schemes is to project a data function onto a set of elementary functions which is a basis for the *vector space* of approximation functions. For example, the Fourier transform projects a data function onto a set of sine and cosine functions. The approximation process consists of selecting a finite set of components from the lower frequencies and rejects the high-frequency components, which are frequently considered as noise. However, this leads to poor compression results, especially for non-linear functions such as OGs (due to structures such as walls or corners, for example). Indeed, a similarity exists between OGs and images, [1]. An approximation space that is useful for these type of signals are wavelet spaces, [2]. Wavelets work by averaging neighboring samples to get a new lowerresolution image of the signal (Table. 1). Clearly, some information has been lost in this averaging and downsampling process. In order to recover the original signal, *detail coefficients* are required to capture the missing information. The popular wavelet transform known as the Mallat algorithm successively averages each scale, starting from the finest scale. The averaging produces the next coarser scale and differences with neighboring samples at the finer scale gives the associated detail coefficients.

Table 1 Elementary step of direct and inverse 1D Haar transform for two neighboring samples $2i$ and $2i+1$ (s_i is the new coarser scale coefficient whereas d_i is the detail coefficient necessary to perform exact reconstruction in the inverse transform)

Haar wavelet transform	Haar inverse wavelet transform
$d_i = p(x_{2i}) - p(x_{2i+1})$	$p(x_{2i+1}) = s_i - \frac{1}{2}d_i$
$s_i = p(x_{2i+1}) + \frac{1}{2}d_i = \frac{p(x_{2i})+p(x_{2i+1})}{2}$	$p(x_{2i}) = d_i + p(x_{2i+1})$

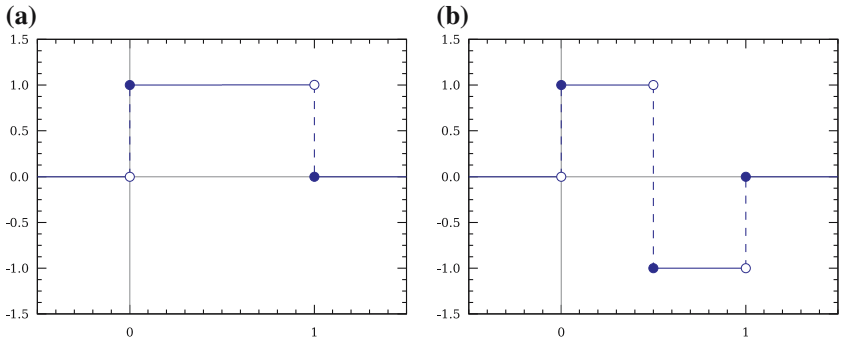


Fig. 2 1D Haar mother functions, *left* the scale mother function Φ and *right* the scale wavelet function Ψ

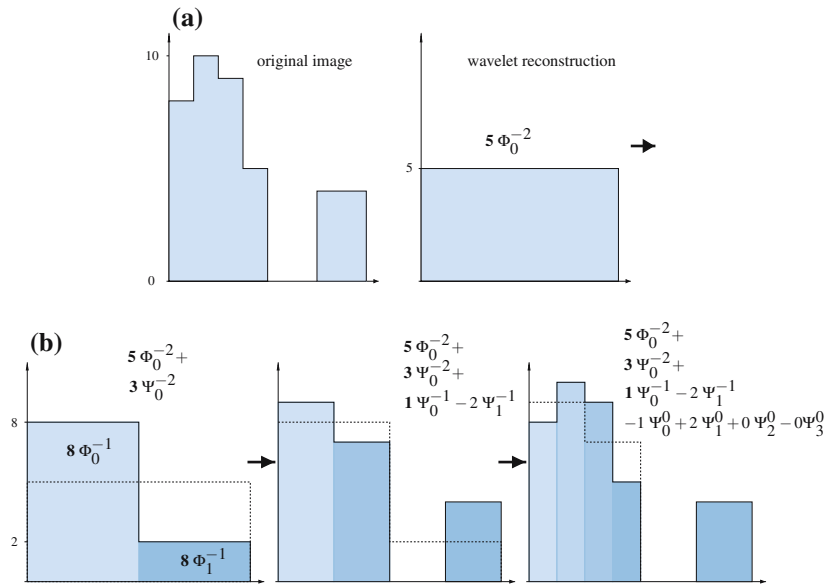


Fig. 3 The 1D image (*upper*, *left*) is: [8, 10, 9, 5, 0, 0, 4, 4], and its unormalized (used here because it is simpler to display) Haar representation is: [5, 3, 1, −2, 0, 0]. The image is then reconstructed one level at a time as follows: $[5] \rightarrow [5+3, 5-3] = [8, 2] \rightarrow [8+1, 8-1, 2-2, 2+2] = [9, 7, 0, 4]$ and so on. Here 0 is the finest scale index or the scale where data is gathered and −2 is the coarsest scale

There is no loss of information in that process since the information contained in the finer scale can be recovered from its average and detail coefficients difference. Since two neighboring samples are often similar, a large number of the detail coefficients turn out to be very small in magnitude, truncating or removing these small coefficients from the representation introduces only small errors in the reconstructed signal, giving a form of "lossy"signal compression. Loss less compression is obtained by removing only zero coefficients.

In this paper wavelets are just used as a special kind of vector space basis that allows good compression. Details about wavelet theory is beyond the scope of this paper and references can be found in [2–4].

2.1.1 Notations

Wavelets are built from two set of functions: scaling and detail functions (also known as wavelet functions). Scaling functions, $\Phi(x)$, capture the average or lower frequency information. Detail functions, $\Psi(x)$, capture the higher frequency information.

The set of wavelet basis functions can be constructed by the translation and dilation of the scaling and detail functions (Figs. 2 and 4). Thus each of the basis function is indexed by a scale l and a translation index t: $\Phi_t^l(x)$ and $\Psi_t^l(x)$. In this paper, the non-standard Haar wavelet basis is used. For non-standard Haar wavelet basis, there is only one mother scaling function and $2^d - 1$ mother wavelet functions, where d is the dimension of the signal. Expanding a function O in the Haar wavelet basis is described as:

$$O(x) = s_0^{-N}\Phi_0^{-N} + \sum_{l=-N}^{l=0}\sum_t\sum_f d_{t,f}^l\,\Psi_{t,f}^l, \tag{1}$$

where the second subscript f is an index to one of the $2^d - 1$ detail function, and N the level such that the whole grid appears as a point. As can be seen in Eq. 1, only one scaling coefficient and one scaling function are required in the expansion of any function $O(x)$. As shown in Fig. 3, the scaling coefficients at other levels are computed as part of the decompression or compression process.

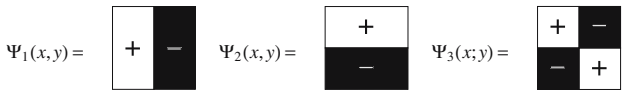


Fig. 4 These graphically defined 2D mother wavelet functions are +1 where *white* and −1 where *black* in the unit square shown and implicitly zero outside that domain. They are tensorial product of 1D wavelet and scaling functions. The 2D scaling function is +1 over the whole unit square and zero outside

The scaling coefficient for a certain level l and translation t holds the average of values contained in the support of the scaling function. The support of any Haar basis function in dimension d is a d-cube, e.g.a square in 2D. If the finest level is 0 and coarser levels are indexed by decreasing negative integers, the side of such a d-cube is 2^{-l} where the unit is in number of samples at level 0.

2.1.2 Tree structure

The key step in a wavelet decomposition is the passage from one scale to another. The support of a Haar wavelet function at level l is exactly partitioned by the support of the 2^d wavelet functions at level $l+1$ (see Fig. 3 for dimension 1).

Therefore it leads to a quadtree for the case of 2D space that hierarchically maps the whole space. A node of the 2D-tree stores three detail coefficients and potentially four children that encode finer details if they are necessary to reconstruct the expanded function (Fig. 4, 5). The key step of a node creation is described in Figure 6.

This data structure is exactly a quadtree, but it not only stores spatially organized data, but also summarizes the data at different resolutions. The root of the tree stores the scaling coefficient at the coarsest level and the support of the corresponding scaling function includes all the spatial locations of the signal data.

2.2 Occupancy grids and telemetric sensor models

OG is a very general framework for environment modeling associated with telemetric sensors such as laser range-finders, sonar, radar or stereoscopic video camera. Each measurement of the range sensor consists of the range to the nearest obstacle for a certain heading direction. Thus a range measurement divides the space into three area: an *empty* space before the obstacle, an *occupied* space at the obstacle location and the *unknown* space everywhere else. In this context, an OG is a stochastic tessellated representation of spatial information that maintains probabilistic estimates of the occupancy state of each cell in a lattice [1]. In this framework, every cell is independently updated for each sensor measurement, and the only difference between the cells is their positions in the grid. The distance which we are interested in, so as to define

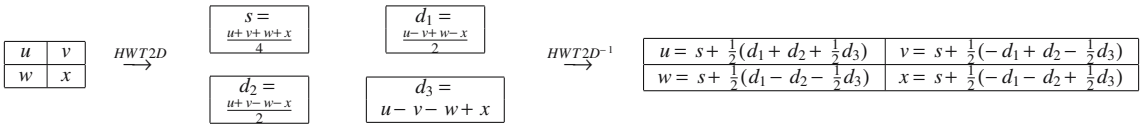


Fig. 5 2D Haar wavelet transform and inverse transform with lifting scheme: algorithm elementary step. It is straightforward that the computing of this step is simple and fast

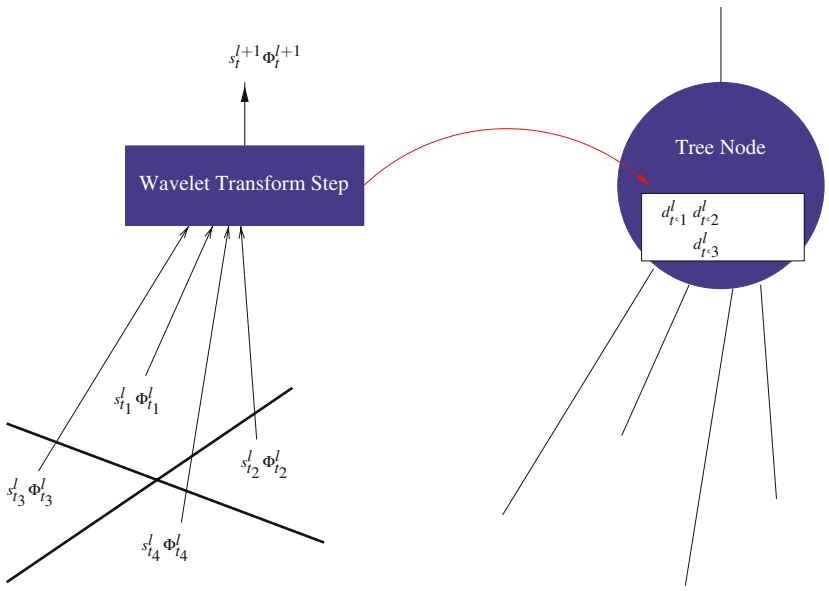


Fig. 6 A key step of a Haar wavelet transform in 2D. 4 scaling samples at scale l generates 1 coarser scaling coefficient at scale $l+1$ and 3 details coefficients at scale l that are stored in a wavelet tree node. In general the tree node has 4 children that described finer resolutions for each space subdivision. But if each child is a leaf and has only zero details coefficients then all the child branches can be pruned without information loss. And the tree node becomes a leaf

cell occupancy, is the relative position of the cell with respect to the sensor location. In the next subsection, the bayesian equations for cell occupancy update are specified with cell positions relative to the sensor.

2.2.1 Bayesian cell occupancy update

Probabilistic variable definitions:

- Z a random variable[1] for the sensor range measurements in the set $\mathcal{Z}$.
- $O_{x,y} \in \mathcal{O} \equiv \{\text{occ}, \text{emp}\}$. $O_{x,y}$ is the state of the cell (x, y), where $(x, y) \in \mathbb{Z}^2$. $\mathbb{Z}^2$ is the set of indexes of all the cells in the monitored area.

Joint probabilistic distribution (JPD): the lattice of cells is a type of Markov field and in this article sensor model assumes cell independence. It leads to the following expression of a joint distribution for each cell.

$$P(O_{x,y}, Z) = P(O_{x,y})P(Z|O_{x,y}) \tag{2}$$

Given a sensor measurement z we apply the Bayes rule to derive the probability for cell (x, y) to be occupied (Fig. 7).

$$p(o_{x,y}|z) = \frac{p(o_{x,y})p(z|o_{x,y})}{p(\text{occ})p(z|\text{occ}) + p(\text{emp})p(z|\text{emp})} \tag{3}$$

Thus the two conditional distributions $P(Z|\text{occ})$ and $P(Z|\text{emp})$ must be specified in order to process cell occupancy update. Defining these functions is an important part of many works [1,5] and, in the following, the results in [6] which proves that for certain choice of parameters[2] these functions are piecewise constants:

$$p(z|[O_{x,y} = \text{occ}]) = \begin{cases} c_1 & \text{if } z < \rho \\ c_2 & \text{if } z = \rho \\ c_3 & \text{otherwise.} \end{cases} \tag{4}$$

$$p(z|[O_{x,y} = \text{emp}]) = \begin{cases} c_1 & \text{if } z < \rho \\ c_4 & \text{if } z = \rho \\ c_5 & \text{otherwise.} \end{cases} \tag{5}$$

when ρ is the range of the cell (x, y).

As explained in [7], the cell update requires operations that are not base inner operators of a vector space (product and quotient). Thus a better form is necessary to operate update on wavelet form of occupancy functions.

[1] For a certain variable V we will note in upper case the variable, in lower case v its realization, and we will note $p(v)$ for $P([V = v])$ the probability of a realization of the variable.

[2] The parameters are the prior occupancy probability which is chosen very low, the world is assumed to be very empty, the sensor model failure rate and the sensor range discretization. Only the first parameter is relevant for establishing the piece-wise constantness of the functions [6].

2.2.2 Log-ratio form of occupancy update

As the occupancy is a binary variable, a quotient between the likelihoods of the two states of the variable is sufficient to describe the binary distribution. The new representation used is:

$$\text{odd}(O_{x,y}) = \log \frac{p([O_{x,y} = \text{occ}])}{p([O_{x,y} = \text{emp}])} \tag{6}$$

In the bayesian update of the occupancy, the quotient makes the marginalization term disappear and thanks to a logarithm transformation, sums are sufficient for the inference:

$$\log \frac{p(\text{occ}|z)}{p(\text{emp}|z)} = \log \frac{p(\text{occ})}{p(\text{emp})} + \log \frac{p(z|\text{occ})}{p(z|\text{emp})} \tag{7}$$

$$= \text{odd}_0 + \text{odd}(z) \tag{8}$$

Therefore the vector space generated by the wavelet basis with its sum inner operator is sufficient to represent and update OGs.

2.2.3 Log-ratio form of sensor model functions

It is straightforward to derive from Eqs. 4 and 5, the sensor model equations in log-ratio form that we note as the following:

$$\text{odd}(z) = \begin{cases} 0 & \text{if } z < \rho \\ \log(c_2/c_4) = \text{odd}_{\text{occ}} & \text{if } z = \rho \\ \log(c_3/c_5) = \text{odd}_{\text{emp}} & \text{otherwise.} \end{cases} \tag{9}$$

when ρ is the range of the cell (x, y). One can notice that the update term is zero if the cell is beyond the sensor readings, thus no update is required in this case.

2.3 Hierarchical rasterization of polygons

This section describes the main contribution of this article which consists of a fast algorithm for updating an occupancy grid expanded as a non-standard Haar wavelet series from a set of range measurements.

2.3.1 Problem statement

The standard approach for updating occupancy grids, in the context of laser sensors, will be to traverse the cells along each laser sensor ray and update the cells. This method of traversal induces difficulties in calculating the area of coverage for each laser sensor ray in order to avoid inaccuracies such as aliasing. An easier alternative will be to traverse every cell of the grid and for each cell, perform a simple test to determine the state of the cell. In this case, with a grid size of 1,024 cells per dimension, a 2D square grid contains more than 1 million cells and for relative small cells (5 cm) the covered area

is smaller (51.2 m) than the common lidar maximal range (≃100 m). Even if real-time performance can be obtained in 2D for small field of view, it is not the case as soon as the field of view is large. Therefore the problem is to find a method that efficiently updates the grid without traversing every cell of the grid. As shown in Fig. 7 and Eq. 9, a range measurement defines three sets of cells (Fig. 8). The first set, E, contains cells that are observed as empty. The second set, U, contains cells that are considered as unknown. The third set, B, contains cells that are partially empty, unknown or occupied. The elements of the third set are mainly found at the boundaries formed by the sensor beams at its two extreme angles and at the location and in the neighborhood of an obstacle. The last remark of the previous section states that the U set can be avoided in the update process. Therefore an update step must iterate through the cells that intersect the polygon that describes the sensor beam boundaries, i.e.,the B set (Fig. 8). The following describes an algorithm that performs the correct iteration through the grid in an efficient manner through the utilization of wavelets.

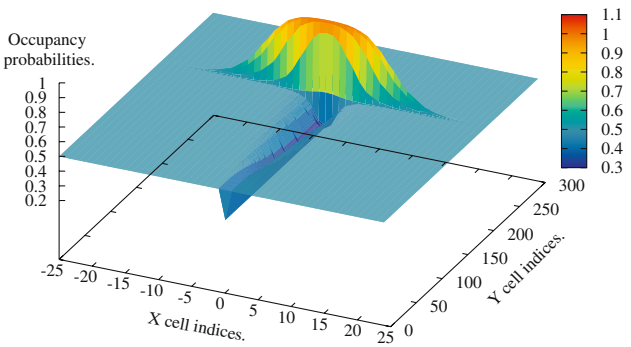


Fig. 7 Update of a 2D OG after a sensor reading, initially each cell occupancy was unknown, i.e.,0.5 probability. The sensor beam has an aperture of 7°. The sensor is positioned in (0,0)

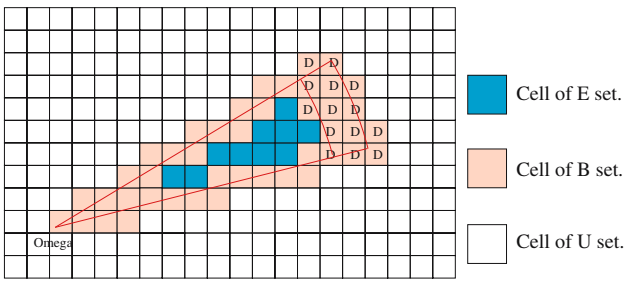


Fig. 8 A range-finder beam. The range finder is located at Ω and its field of view is surrounded by red boundaries. It defines the three kind of cell types. The band, within the obstacle, lies is at the *top right* end of the field of view. Thus the cells marked with a "D" stand for cells where a detection event occurs

2.3.2 Hierarchical space exploration

The key idea in the exploration of the grid space is to define a predicate: *existIntersection* that is true if a given set of grid cells intersect the surface defined by the field of view of the sensor beams (the most outer boundaries in red, Fig. 8). The absence of intersection indicates that the given set of cells are outside the sensor field of view and can be discarded for occupancy update. For the case of *existIntersection* evaluating to true, a special sub case would be when the set of cells are totally included in the sensor field of view, then all the cells of the set belong to E and their occupancy are decreased by the same amount of odd_{emp}. So it is equivalent to decrease the occupancy of the coarse area that contains all these cells by odd_{emp}.

As the algorithm is able to obtain uniform regions recursively, the grid representation should allow the update of regions, and wavelets provide a natural mechanism for doing so. In this algorithm, the grid is traversed hierarchically following the Haar wavelet support partition. For each grid area, the *existIntersection* predicate guides the search. If there is intersection the traversal reaches deeper into the grid hierarchy, i.e.,explores finer scales. Otherwise it stops the wavelet transform for the current branch of the wavelet tree as described in Fig. 6 for the 2D case.

Algorithm 1: HierarchicalWavRaster(subspace S, sensor beam B)

1: **for** each subspace i of S: $i = 0, \ldots, 3$ **do**
2: **if** sizeof(i) = minResolution **then**
3: v_i = evalOccupancy(i)
4: **else if** existIntersection(i, B) **then**
5: **if** $i \in E$ **then**
6: $v_i = \text{odd}_{\text{emp}}$ /*eq. 9*/
7: **else**
8: v_i = HierarchicalWavRaster(i, B)
9: **end if**
10: **else**
11: $v_i = 0$ /*$i \in U$*/
12: **end if**
13: **end for**
14: $\{s_S^{l+1,obs}, d_{f_1,S}^{l,obs}, \cdots, d_{f_3,S}^{l,obs}\}$ =waveletTransform($\{v_0, \cdots, v_3\}$)
15: **for** each $d_{f,S}^{l}$: **do**
16: $d_{f,S}^{l} \leftarrow d_{f,S}^{l} + d_{f,S}^{l,obs}$ /*update inference*/
17: **end for**
18: returns the scaling coefficient $s_S^{l+1,obs}$

Algorithm 1 gives the pseudo-code of the hierarchical grid traversal. The algorithm is recursive and begins with the root of the wavelet tree as the initial subspace. The result of this first function call is used to update the mean of the wavelet tree which is also the coefficient of the scaling function. The root represents the whole grid and is subdivided in to four subspaces following quad-tree space organization. Then each

considered subspace S is either an area that have four subspaces or a leaf of the tree, that is a cell in the grid. The *sizeof* function get the resolution of the subspace i and *minResolution* represents the resolution of a cell in the grid.

The *evalOccupancy* function evaluates the occupancy of a cell; it can proceed by sampling the cell, or by calculating if each sample point lies in an occupied, empty or unknown area. Then, the odd values for each sample point in a cell are averaged. Alternatively, the max occupancy in the cell can also be chosen.

Such an algorithm is far more efficient than a complete traversal of the grid especially with range-finder that measure several beams at a time. With this kind of range-finder the field-of-view polygon is very large and the U and the E set are very large causes early cut in the space traversal. Therefore the leaf of the space representation are almost an obstacle. Then the representation is almost as simple as a sensor impact record in a quad-tree but stores all the occupancy volumetric information in addition.

2.3.3 Bounded map

To ensure good compression and map dynamics, the possible occupancies are bounded. On the one hand, these bounds allow that large portions of the map that are empty converge towards the lower bound and are considered at large scale as constant values. Thus the wavelet representation allows to store only large scale coefficient, early in the tree, to describe these areas. On the other hand, these bounds make it possible that the map is updated at correct speed: if an occupied part of the map represents a car parked in a parking lot, the map must be rapidly updated when the car drives away. In that example, the map must evolve from the upper occupancy bound to the lower occupancy bound. If these bounds are controlled, it is possible to fix it according to odd_{emp} and odd_{occ} such that the update speed needed is reached.

The building map process begins with a tree that is composed of a single root. While sensor information is processed, nodes are added in the tree when new regions are discovered. But as the world is not static, there are moving obstacles that produce measurement (as with a parked car that moves). Therefore there are occupied or empty area appearance and disappearance. But as the grid is bounded it is possible for each subspace at each scale to deduce if the maximum depth is reached. Indeed if the occupancy equals one of the two bounds, no finer information is necessary to describe the area.

Here follows the demonstration for the lower bound:
if $\forall c_i, \text{odd}(c_i) \geq o_{\min}$ and $s = \frac{1}{n}\sum_i^n \text{odd}(c_i) = o_{\min}$
then $\forall c_i, \text{odd}(c_i) = o_{\min}$.
Otherwise:

$$\exists j, \text{odd}(c_j) > o_{\min} \tag{10}$$

$$\frac{1}{n-1}\sum_{i\neq j;1\leq i\leq n}\text{odd}(c_i) + \frac{1}{n-1}\text{odd}(c_j) = \frac{n}{n-1}o_{\min} \tag{11}$$

$$\frac{1}{n-1}\sum_{i\neq j;1\leq i\leq n}\text{odd}(c_i) = \frac{1}{n-1}(\text{no}_{\min} - \text{odd}(c_j))$$

But $\text{no}_{\min} - \text{odd}(c_j) < (n-1)o_{\min}$ following Eq. (10), then

$$\frac{1}{n-1}\sum_{i\neq j;1\leq i\leq n}\text{odd}(c_i) < o_{\min} \tag{12}$$

which is not acceptable under the first hypothesis.

In such case when only coarse resolution are updated it is possible that some nodes remain at finer resolution that have no more sense in a bounded map context. Therefore a function is needed to prune the tree in this particular case.

The algorithm described above is generic. It works for every kind of range-finder but it requires that the *existIntersection* function is also defined.
The efficient defining of these two functions is the subject of the next section.

2.3.4 2D laser scans and map correction

One of the most important part of the previous algorithms are the intersection queries. So they must be really optimized in order to retrieve fast algorithms. For 2D laser scans, a bundle of range measurements in contiguous beams, we propose to use an approximation of the query which only ensures that if there is an intersection the result will be correct.

The principle is to compute the polar bounding box[3]: $[r_{\min}; r_{\max}] \times [\theta_{\min}; \theta_{\max}]$ of the cartesian cell traversed and then verifying if there exists a laser beam with a range measurement inside the bounding box. As each range beam is usually indexed by its angle, the comparison is very fast. If the bounding box is inside the angle field of view of the sensor, the ranges are checked to classify the bounding box position inside one of the two cell sets E or B.

As this intersection function is an approximation, sometimes finer traversals are done to evaluate the occupancy of cells whereas those cells belong to a coarse area which is part of set U or set E. In this case, the leafs or nodes are useless and the associate detail coefficients are zero. To take this case into account and the useless nodes added due to moving obstacles also, the compression function is launched but at a low frequency (every 100 scans for instance). It removes all the extra nodes: when a coarse area with bound value has child, all childs are removed and when all the detail coefficients are zero in a branch of the tree, the whole branch is

[3] $r_{\min}$ and $r_{\max}$ stand for the maximum and minimum range of the cell and $\theta_{\min}$ and $\theta_{\max}$ for the maximum angle and the minimum angle of the cell respectively.

removed. In addition, the compression function can change all the coefficients with a norm lower than a certain threshold to zero to provide lossy compression feature.

3 Object extraction

3.1 Foreground selection

In the previous sections, an efficient algorithm was presented to build a dense and precise map of the static environment. In the following, this map is named the background in reference to the computer vision terminology when the set of all moving obstacles in the field of view are thus named as foreground.

The aim of this subsection is to present how to extract cells of the map that are occupied by a moving obstacle in front of the sensor field of view i.e.,the foreground. Then this set of foreground cells is communicated to a second algorithm that computes a clustering step in order to decrease the number of observations for the data association step of the target tracking algorithm.

This extraction is based on the difference map $\mathcal{M}_\mathrm{d}$ which computes an estimation of the similarity between the map constructed with the current measurement (the observed map $\mathcal{M}_\mathrm{o}$) and the static map ($\mathcal{M}_\mathrm{s}$) of the environment for each cell (Fig. 9).

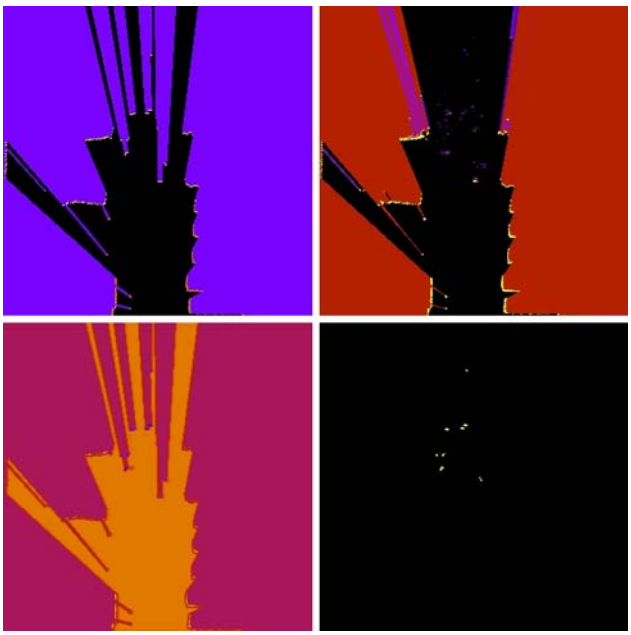


Fig. 9 *Upper left* the observed map $\mathcal{M}_0$. *Upper right* the static map $\mathcal{M}_\mathrm{s}$. *Bottom left* the difference map $\mathcal{M}_\mathrm{d}$. *Bottom right* the extracted foreground. For all the maps, the more hot is the color, the more positive is the value

3.1.1 Difference map

The matching measure $\mathcal{S}$ used in this algorithm was first proposed in Moravec and Elfes [8]. For each cell c in $\mathcal{M}_\mathrm{o}$, the similarity is:

$$s_c = \mathrm{odd}_c^{\mathcal{M}_\mathrm{o}} * \mathrm{odd}_c^{\mathcal{M}_\mathrm{s}}. \tag{13}$$

It consists in the product of the log-ratio occupancy of the observed map and the static map. Thanks to the log-ratio, an occupied cell has a positive log-ratio whereas an empty cell has a negative one. Therefore the product of the log-ratios of two cells that agree about their occupancy is positive (negative by negative or positive by positive) while the product for two cells that disagree about their occupancy is negative. This similarity value is low whenever a change in the environment occurs. So there are no difference between a place left by a parked object and a place temporarily occupied by a moving object. Therefore we focus only on the cells detected as occupied by the current sensors: all cells c where $\mathrm{odd}_c^{\mathcal{M}_\mathrm{o}}$ is strictly positive.

3.2 Clustering

Assuming that the number of objects k in a difference map is known, applying clustering to object extraction using the k-means [9,10] algorithm is straightforward:

(a) Initialize k cluster centers μ_i with arbitrary values.
(b) Assign each foreground cell to its closest cluster center.
(c) Reestimate every cluster center μ_i as the mean of the centers of the cells allocated to that cluster.
(d) Repeat steps 2–4 until some convergence criterion is met (e.g., minimal cluster reassignment).

However, in most cases, the value of k is unknown. Furthermore, even knowing k, the quality of the obtained clustering depends heavily on initialization, since the algorithm trends to get stuck in local minima. Finally every iteration has a cost of $O(N_\mathrm{f}k)$ (where N_f is the number of foreground cells) and, sometimes, many iterations are needed before converging.

In order to deal with those problems, we have proposed an object extraction approach which combines a self-organizing network inspired by Kohonen self-organizing maps [11] and the growing neural gas [12] as well as a graph theoretic algorithm used to cut edges in the network's graph.

3.2.1 SON clustering

The network is built from $M = W \times H$ nodes connected with undirected edges, arranged in a grid with H rows and W columns (Fig. 10a). This means that, with the exception of nodes located in the borders, every node i will be connected to four

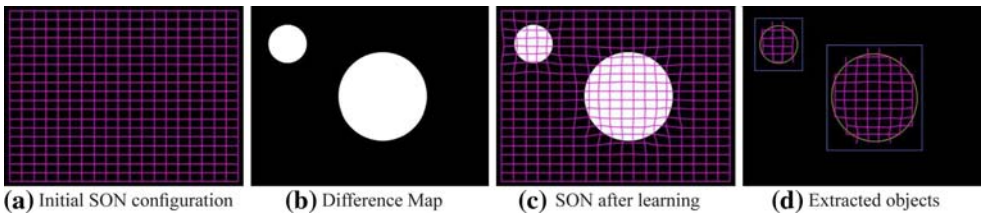


Fig. 10 Approach overview. The images show the different steps of our algorithm using a synthetic difference map

other nodes or neighbors $neigh(i)$, individually denoted by $u(i)$, $d(i)$, $r(i)$ and $l(i)$ for up, down, right and left, respectively. Every node i has two associated variables: its mean value $\mu_i = (x_i, y_i)$ and a counter $c_i \in [0, N_f]$. In a similar manner, for every edge connecting nodes i and j there will be a counter $e_{i,j} \in [0, N_f]$. Besides W and H, the algorithm has other two parameters: $0 < \epsilon_n < \epsilon_w \leq 1$, which are learning rates whose exact meaning will become clear later.

The following paragraphs describe the steps that our algorithm performs *for every time step*, using the difference map as input.

Initialization The network is initialized by assigning values to all the μ_i node centers in order to form a regular grid (Fig. 10a). Also, the values of all the weights are set to zero.

$$\{c_i \leftarrow 0, e_{i,j} \leftarrow 0 \,\forall\, i, j \mid i \in [1, M], j \in neigh(i)\} \tag{14}$$

Learning The learning stage takes every foreground cell p of the difference map (Fig. 10b) and process it in three steps:

(a) Determine the two nodes whose means are closest to p:

$$w_1 = \arg \min_{i \in [1,M]} \|p - \mu_i\| \tag{15}$$

and

$$w_2 = \arg \min_{i \in [1,M] \setminus w_1} \|p - \mu_i\| \tag{16}$$

(b) Increment the values of e_{w_1,w_2} and c_{w_1}:

$$e_{w_1,w_2} \leftarrow e_{w_1,w_2} + 1 \tag{17}$$

and

$$c_{w_1} \leftarrow c_{w_1} + 1 \tag{18}$$

(c) Adapt the mean of w_1 and all its neighbors:

$$\mu_{w_1} \leftarrow \mu_{w_1} + \frac{\epsilon_w}{c_{w_1}}(p - \mu_{w_1}) \tag{19}$$

$$\mu_i \leftarrow \mu_i + \frac{\epsilon_n}{c_i}(p - \mu_i) \quad \forall i \in neigh(w_1) \tag{20}$$

Relabeling nodes As a result of the learning step, the network adapts its form to cover the objects in the difference map (Fig. 10c). The last step of our algorithm finds groups of nodes by merging nodes according to the weight of their common edges $e_{i,j}$. The idea is that a higher value of $e_{i,j}$ corresponds to a higher likelihood that nodes i and j belong to the same object. Under this assumption, it is possible to compute a maximum likelihood estimation of the probability, denoted by $P_{i,j}$, that two nodes "belong together" using the Laplace law of succession[4]:

$$P_{i,j} = \frac{e_{i,j} + 1}{N_f + (W-1)H + (H-1)W} \tag{21}$$

Also by using the Laplace law of succession, we calculate the value of the uniform link probability distribution, which may be seen as the maximum entropy estimate of $P_{i,j}$ *prior to learning*.

$$\mathbf{U}_{\text{links}} = \frac{1}{(W-1)H + (H-1)W} \tag{22}$$

In a similar fashion, the weight c_i is an indicator of the likelihood that node i belongs to an object (i.e., instead of the background), which may be formulated as a probability P_i.

$$P_i = \frac{c_i + 1}{N_f + WH} \tag{23}$$

With the corresponding uniform being:

$$\mathbf{U}_{\text{nodes}} = \frac{1}{WH} \tag{24}$$

We use a conventional scanning algorithm to relabel the nodes. The only particularity of our approach is that $P_{i,j}$ is used as the region-merging criterion instead of using colors or other features. Here, we will outline the labeling algorithm, however, the presentation of the complete implementation details is beyond the scope of this paper. The reader is referred to [13,14] for efficient linear-time ways to implement the algorithm.

[4] $P_{i,j}$ is a notational shortcut introduced for the sake of readability, being rigorous it should be written as $P([O_i = m] \mid [O_j = m])$, where $O_i = m$ indicates that node i has been assigned to cluster m. A similar shortcut has been used with P_i for the same reasons.

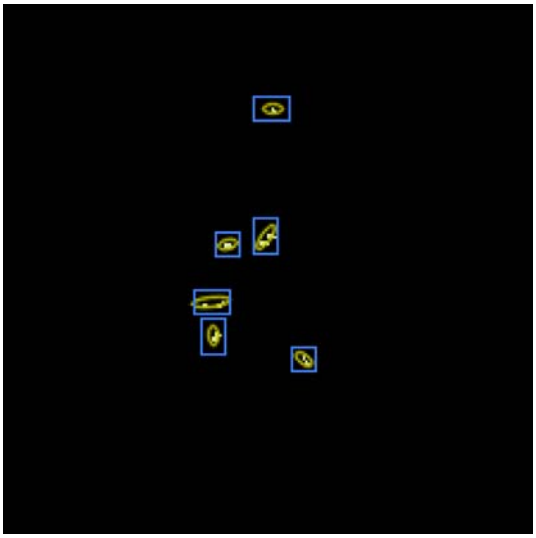


Fig. 11 An example of clustering on the occupancy grid shown in Fig. 6. It may be seen that pedestrian legs detected by the laser range-finder are successfully paired by the clustering algorithm

The algorithm starts from the upper left node and proceeds by scanning from left to right and from top to bottom, for every node i the following steps are applied (Figs. 11, 12):

a. Assign the label ∞ to i.
b. If $P_{i,l(i)} > \mathbf{U}_{\text{links}}$, assign to i the label of $l(i)$ (merge with left region).
c. If $P_{i,u(i)} > \mathbf{U}_{\text{links}}$, assign to i the minimum between its current label and the label of $u(i)$. Let a be that minimal label and let b be the label of $u(i)$. Relabel all nodes on the previous rows having label b to a (merge with upper region).
d. If i's label is ∞ assign the next unused label to i (create a new region).

Computing cluster representations Having labeled the nodes, a cluster m may be represented using the gaussian distribution of a point p[5]:

$$P^*(p \mid m) = \mathbf{G}(p;\ \mu_m^*,\ S_m^*) \tag{25}$$

The cluster's prior may be used to filter out clusters whose prior is below a given threshold, it is computed as:

$$P_m^* = \sum_{i \in m} P_i \tag{26}$$

Its mean value,

$$\mu_m^* = \frac{1}{P_m^*} \sum_{i \in m} P_i \mu_i \tag{27}$$

[5] Hereafter, cluster parameters will be denoted by a superscript asterisk, in order to distinguish them from node parameters.

And its covariance,

$$S_m^* = \sum_{i \in m} \frac{P_i}{P_m^*} \begin{bmatrix} (x_i - x_m^*)^2 & (x_i - x_m^*)(y_i - y_m^*) \\ (x_i - x_m^*)(y_i - y_m^*) & (y_i - y_m^*)^2 \end{bmatrix} \tag{28}$$

3.2.2 Discussion

Our self-organizing network may be seen as a cross between Kohonen self-organizing maps (SOM) and the growing neural gas algorithm. With the first, our approach shares the fixed topology and number of nodes M, which may be easily fixed due to the fact that the number of samples per frame is bounded by the size of the difference map. This should work well on the condition that M is much greater than the maximum expected number of objects in the image.

On the other hand, our SON borrows GNG idea of assigning weights to links between nodes. However, the way these weights get updated in GNGs is highly discontinuous and, from our point of view, not well suited for modeling it as a probability. Hence, we have profited from the fact that no link deletion/addition takes place in our setting and replaced the GNG age parameter with a counter, which we consider as more appropriate for representing probabilities.

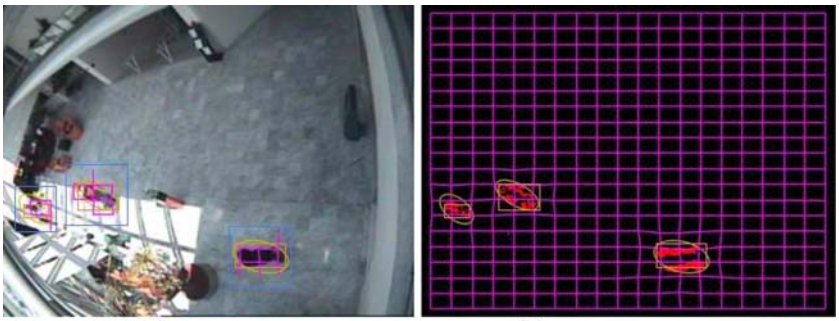
Finally, weight updating bears similarities to both approaches. It applies the GNG use of two constant learning rates ϵ_w and ϵ_n, however, since in our case, foreground cells are processed from top to bottom and from left to right, this results in a skewing phenomenon in which the same nodes get updated many consecutive times and tends to "follow" the direction of sampling. In order to alleviate this situation we have chosen to use a decreased learning rate's influence with the number of cells c_i that have been assigned to the given node, in a way that resembles SOM decaying learning rates.

3.2.3 Complexity analysis

Thanks to the existence of efficient algorithms, the cost of labeling is linear with respect to the number of nodes in the SON, moreover, the computation of the cluster representation (i.e., gaussian parameters, mixture of gaussian parameters and bounding boxes) may be performed at the same time than labeling. Thus, the algorithm's complexity is bounded by the learning algorithm complexity which is $O(N_f M)$.

3.3 Tracking

The tracking procedure is an object-based tracking system like in Schulz et al. [15]. For each object a particle filter is used to robustly estimate its position. For the data association a joint probabilistic data association is performed. The observed map is again used to evaluate which objects are



(a) Hall image **(b)** Classification bitmap

Fig. 12 A typical frame of our system running with a CAVIAR video. The *small boxes* correspond to ground truth, the bigger ones and the gaussians (*ellipses*) are estimated by our system

occluded and where static obstacles forbid the presence of a moving target. That ensures that no unobserved target is removed from the target pool and that the prediction step of the particle filters is coherent with the static map.

4 Motion pattern-based motion prediction

Motion tracking provides an estimation about the current state of the dynamic obstacles that populate the environment, however, this information is not enough for motion planning: in order to be able to deal with a dynamic environment, it is also necessary to know how those obstacles will evolve in the future. Since, in most cases, this knowledge is not available, it is necessary to resort to prediction.

The conventional approach of predicting motion on the basis of the object's kinematic and dynamic properties is not well suited for objects such as humans, vehicles, and the like, because their motion is motivated by other factors which are, in general, difficult to model (e.g.perception, internal state, intentions etc.). Over the last decade, an alternative approach has emerged; it is based on the idea that objects follow typical motion patterns within a given environment [16–23].

A motion prediction system based on this idea should fulfill two tasks:

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

However, existing approaches focus primarily on off-line learning algorithms, implying that learning should be performed before prediction. This is problematic because, for this to work, it is necessary that learning data includes all the possible learning patterns, which is a very difficult condition to meet. It would be preferable to have a system which operates in a "learn and predict" fashion, where learning takes place continuously, taking as input the same data that is used for prediction.

Since uncertainty is inherent to both prediction and sensing, we have decided to base our solution in a probabilistic framework—Hidden Markov Models (HMM) [24]—which is widely used in the literature of motion prediction [22,25, 26] using off-line learning. In contrast, our approach is based on a novel incremental parameter and structure learning algorithm which enables the system to work with a "learn and predict" framework.

4.1 Problem overview

We assume that tracking data is available as a collection of observation sequences (i.e.,trajectories). Every individual sequence[6] $O_{1:T} = \{O_1, \ldots, O_T\}$ corresponds to the tracker's output for a single object and its observations are evenly spaced on time. Taking this information as input, our approach consists of the following:

1. *Learning:* complete observation sequences are used to learn the HMM structure and parameters. This process is incremental, and takes place immediately after the final observation for a trajectory has been received. Our learning algorithm is described in Sect. 4.4.
2. *Prediction:* for every new observation that is output by the tracking system, bayesian inference is applied to maintain the probabilistic belief of the object's state, and to project it into the future. Thus, a prediction consists of a probability distribution of the object's state for a given time horizon H. Prediction is described in Sect. 4.5.

4.2 Hidden Markov models

For lack of space, this discussion of Hidden Markov models (HMM) is summary and strongly biased towards their application as motion models, the interested reader is referred to Rabiner [24] for a more comprehensive introduction.

[6] For the sake of readability, notation $O_{1:t}$ will be used as a shortcut for the variable conjunction $O_1\ O_2\ \cdots\ O_{t-1}\ O_t$.

HMM are a popular probabilistic framework used to describe the evolution of dynamic systems. In the context of this paper, an HMM may be regarded as a quantization of the object's state space into a small number of discrete states, together with probabilities for transitions between states. The system state at time t is indexed by a single finite discrete variable S_t, however, it is assumed that its value is unknown and hidden (i.e.,not observable), thus, a probability distribution (i.e.,the belief state) is used to represent it. The system state may be updated on the basis of observations O_t gathered through sensors, which are related to the state through an observation probability.

Formally, HMMs are defined in terms of three variables:

- S_t, S_{t-1}, the current and previous states, which are discrete variables with value $S_t, S_{t-1} \in \{1, \ldots, N\}$ for a fixed N.
- O_t, the observation variable, which is a multidimensional vector in $\mathbb{R}^M$.

With the following joint probability decomposition:

$$P(S_{t-1}\ S_t\ O_t) = \underbrace{P(S_{t-1})}_{\text{stateprior}} \underbrace{P(S_t \mid S_{t-1})}_{\substack{\text{transition}\\ \text{probability}}} \underbrace{P(O_t \mid S_t)}_{\substack{\text{observation}\\ \text{probability}}} \tag{29}$$

where the state prior is computed recursively from the previous time step:

$$P(S_{t-1}) = P(S_{t-1} \mid O_{1:t-1}) \tag{30}$$

Both the observation and transition probabilities are assumed to be *stationary*, that is, independent of time:

$$P(O_i \mid S_i) = P(O_j \mid S_j) \quad \forall i,\ j \in \{1, \ldots, T\} \tag{31}$$

$$P(S_i \mid S_{i-1}) = P(S_j \mid S_{j-1}) \quad \forall i,\ j \in \{2, \ldots, T\} \tag{32}$$

This hypothesis permits to define the parametric forms of the three probabilities in the JPD without taking time into account:

- $P(S_0 = i) = \pi_i$. The state prior will be represented as a a vector $\pi = \{\pi_1, \ldots, \pi_N\}$ where each element represents the prior probability for the corresponding state.
- $P([S_t = j] \mid [S_{t-1} = i]) = a_{i,j}$. Transition probabilities are represented with a set of variables A, where each element $a_{i,j}$ represents the probability of reaching the state j in the next time step given that the system is in state i.
- $P(O_t \mid [S_t = i]) = \mathbf{G}(O_t;\ \mu_i,\ \sigma_i)$. The observation probability will be represented by a gaussian distribution for every state. The set of all the gaussians' parameters will be denoted by $b = \{(\mu_1, \sigma_1), \ldots, (\mu_N, \sigma_N)\}$.

The full set of parameters for an HMM is denoted by $\lambda = \{\pi, A, b\}$. An additional concept which is very important is *structure*, which is determined by the prior assumptions on which transitions between states are allowed (i.e., have a transition probability greater than zero). It is often useful to visualize the HMM structure as a graph, where discrete states are represented by nodes, and valid transitions are represented by directed edges between nodes (Fig. 13).

4.3 Motion patterns representation with HMMs

Practically all HMM-based motion models in the literature assume that the state space consists of the object's bidimensional position in world's coordinates, and represent motion patterns as non-connected trajectory-like subgraphs within the HMM structure (Fig. 14).

In contrast, our approach is based on an extended state definition: the state consists of the object's current and final coordinates (x_t, y_t, x_T, y_T), thus, the intended destination becomes part of the state. This extended state has two advantages: (a) by estimating the belief state, we automatically obtain an estimation of the objects final destination; and

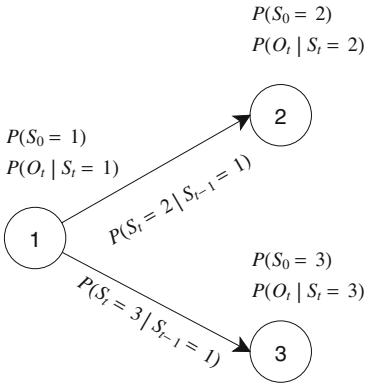


Fig. 13 A basic three-state HMM, with its associated probabilities

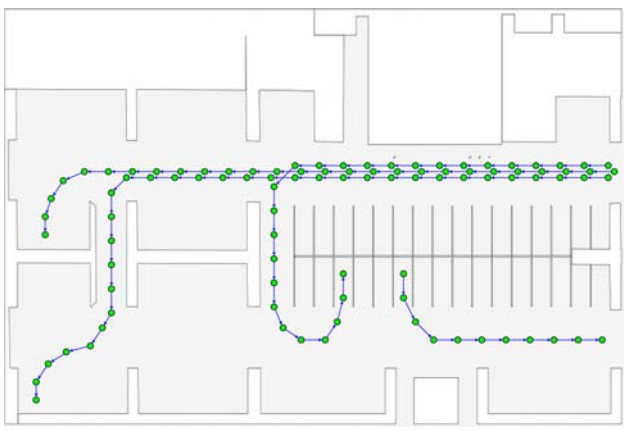
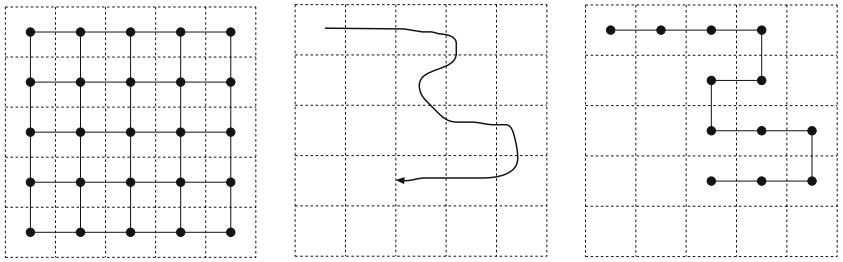


Fig. 14 An HMM structure for a parking environment, where motion patterns are represented as non connected order 2 subgraphs (only a few motion patterns are displayed)



(a) grid and map **(b)** trajectory in space **(c)** trajectory in the map

Fig. 15 **a** Example of a decomposition of space into a grid of rectangular cells (*dotted lines*), and the corresponding topological map representation (*dots* and *solid lines*); **b** a continuous trajectory; and **c** the same trajectory represented as a succession of nodes and edges in the map

(b) since learning takes place when a trajectory has ended, the final destination is available during learning, and may be used as an additional criterion to discriminate between different motion patterns.

Another particularity of our approach is that the structure does not consist of unconnected components, instead, edges are arranged in a network according to the topological properties of the states as will be explained in Sect. 4.4.

4.4 Learning algorithm

Our learning algorithm takes whole observation sequences as input and processes them in two steps: (a) update the HMM structure using a topology learning Network; and (b) update the HMM parameters according to the current structure.

The key intuition behind our learning algorithm is that the structure of the HMM should reflect the spatial structure of the state space discretization, where transitions between states are only allowed between neighboring regions. In our approach, structure learning consists basically in building a *topological map*: a discrete representation of the state-space in the form of a graph, where nodes represent discrete regions and edges exist between contiguous nodes, meaning that it is possible to move continuously between them without passing through any other region.

The idea may be illustrated on a regular grid discretization of a two-dimensional space (Fig. 15), Where a topological map may be built by connecting the center of every cell to every one of its neighbors—the cell with which it shares a common border. Notice that an object which moves continuously should inevitably pass through region borders in order to go from one region to other, therefore, all the possible continuous motion instances may be approximated exclusively in terms of the edges that exist already in the topological map.

However, grid discretizations are wasteful on computational resources, and it is preferable to discretize the space according to observed data. Nevertheless, this poses the problem of how to perform this discretization and, at the same time, identify neighbor regions in an efficient way. Moreover, the searched solution needs to be incremental.

Fortunately, there exists a family of tools which deals precisely with these problems: topology representing networks (TRNs) [27]. They incrementally build a topological map by applying two steps: (a) partition space in discrete regions using vector quantization, and (b) find pairs of regions with a common border and link their respective centers.

4.4.1 Partitioning the space with vector quantization

The idea of vector quantization is to encode a continuous d-dimensional input data manifold M by employing a finite set $C = \{c_1, \ldots, c_K\}$ of reference d-dimensional vectors. A point x of the manifold is represented using the element of C which is closest to it according to a given distance measure $d(x, c_i)$, such as the square error.

This procedure induces an implicit partition of the manifold in a number of subregions

$$\mathcal{V}_j = \{x \in M \mid d(x - c_j) \leq d(x - c_i) \forall i\} \tag{33}$$

called Voronoi regions (Fig. 16), such that every input vector that is inside a Voronoi region $\mathcal{V}_j$ is described by the corresponding reference vector c_j.

The goal of a vector quantization algorithm is to find values for the reference values in order to minimize the mean

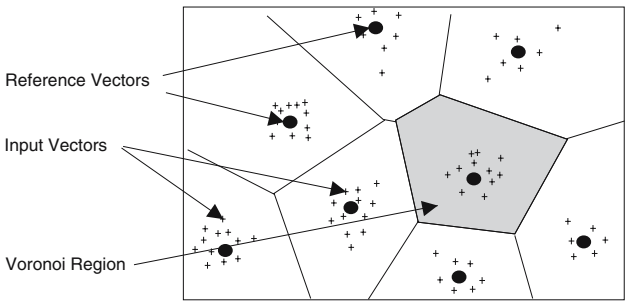


Fig. 16 Example of a partition in Voronoi regions: there are some two-dimensional input vectors (*crosses*). Reference vectors are represented by big points and Voronoi regions are indicated by boundary lines, the set of all those boundaries is called a Voronoi graph

quantization error or *distortion*:

$$E = \sum_{i=1}^{K} \int_{x \in \mathcal{V}_i} d(x, c_i) P(x) \mathrm{d}x \tag{34}$$

4.4.2 Linking neighbor regions: delaunay edges

Topology representing networks are based on the idea of connecting neighbor Voronoi regions, – e.g.regions with a common border—with edges, called Delaunay's edges. The set of all edges constitutes the dual of the Voronoi graph and is known as the Delaunay's triangulation (Fig. 17).

TRNs use a subset of the Delaunay triangulation to represent the network's topology. The edges are learned using the competitive hebbian rule, also known as hebbian learning, proposed Martinetz [28] which consists in creating a new edge between two reference vectors every time that, for a given input, they are the two closest vectors to that input and they are not linked already (Fig. 18).

4.4.3 The instantaenous topological map

We have decided to use the instantaneous topological map (ITM) proposed by Jockusch and Ritter [29], among the different TRNs exist in the literature [12,27,30], because it is specially designed for the inputs which are correlated in time, which is the case of observation sequences. Although we list it here for reference (alg. 2), here we will discuss only its application to our problem, the interested reader is referred to the original paper for details on the inner working of the algorithm.

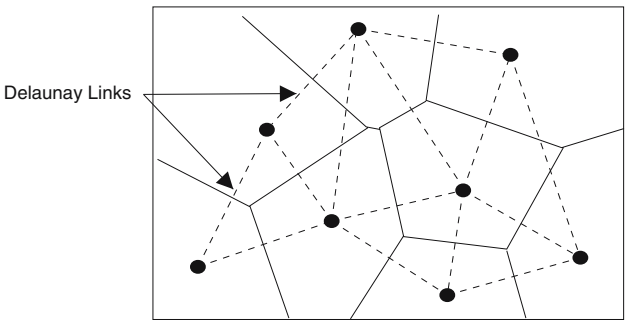


Fig. 17 Voronoi graph and delaunay triangulation

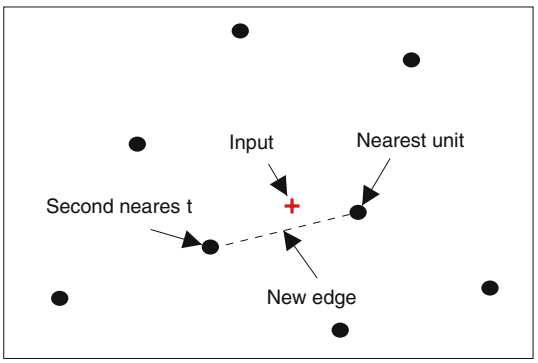


Fig. 18 Hebbian learning

Algorithm 2: *ITM-Update*$(O_t, \Sigma, \tau, \epsilon : \mathcal{U}, \mathcal{L})$

input :
Input vector O_t
Covariance matrix Σ
Insertion Threshold τ
Smoothing factor ϵ
modifies:
Topological map nodes $\mathcal{U}$
Topological map edges $\mathcal{L}$

begin
  $b = \arg\min_{i \in \mathcal{U}} \ d^2_\Sigma(w_i, O_t)$
  $s = \arg\min_{i \in \mathcal{U} \setminus b} \ d^2_\Sigma(w_i, O_t)$
  $w_b = w_b + \epsilon(O_t - w_b)$
  if $s \notin \mathcal{N}(b)$ **then** $\mathcal{L} = \mathcal{L} \cup \{(b, s)\}$
  for $i \in \mathcal{N}(b)$ **do**
    $\bar{w}_{b,i} = (w_i + w_b)/2$
    if $d^2_\Sigma(\bar{w}_{b,i}, w_s) < d^2_\Sigma(\bar{w}_{b,i}, w_i)$ **then**
      $\mathcal{L} = \mathcal{L} \setminus (b, i)$
      if $\mathcal{N}(i) = \emptyset$ **then** $\mathcal{U} = \mathcal{U} \setminus i$
    end
    $\bar{w}_{b,s,=}(w_s + w_b)/2$
    if $d^2_\Sigma(\bar{w}_{b,s}, w_s) < d^2_\Sigma(\bar{w}_{b,s}, O_t)$
    and $d^2_\Sigma(w_b, O_t) > \tau$ **then**
      $\mathcal{U} = \mathcal{U} \cup \{r\} w_r = O_t$
      $\mathcal{L} = \mathcal{L} \cup \{(b, r)\}$
      if $d^2_\Sigma(w_b, w_s) < \tau$ **then** $\mathcal{U} = \mathcal{U} \setminus s$
    end
  end
end

The ITM algorithm builds incrementally a set $\mathcal{U}$ of nodes, and a set $\mathcal{L}$ of edges connecting nodes. The input of the algorithm consists of input vectors which, here, will correspond to observations O_t of a moving object.

Associated with every node i is a reference vector or weight w_i.

An edge between nodes i and j will be denoted as (i, j), since edges are not directed, it holds that $(i, j) \equiv (j, i)$. A useful concept is the *neighborhood* of a node i, which is the set of all nodes to which i is linked:

$$\mathcal{N}(i) = \{j \in \mathcal{U} \mid (i, j) \in \mathcal{L}\} \tag{35}$$

The algorithm requires a distance metric, we have used the Mahalanobis distance:

$$d^2_\Sigma(u, v) = (u - v)^T \Sigma^{-1} (u - v) \tag{36}$$

Finally, an insertion threshold τ needs to be specified to the algorithm. It may be seen as a measure of the mean

discrete partition size to be obtained, expressed in terms of the Mahalanobis distance.

4.4.4 Integrating the parts: HMM update

Algorithm 3 presents the overall structure and parameter update for an observation sequence.

Algorithm 3: *HMM-Update*($O_{1:T}$, Σ, τ, ϵ : $\mathcal{U}$, $\mathcal{L}$, λ)

input :
Observation sequence $O_{1:T}$
Covariance matrix Σ
Insertion Threshold τ
Smoothing factor ϵ
modifies:
Topological map nodes $\mathcal{U}$
Topological map edges $\mathcal{L}$
HMM parameters $\lambda = \{\pi, b, A\}$

1 **begin**
2 **for** $t \in \{1, \cdots, T\}$ **do**
3 $EnhancedITM\_update(O_t)$
4 **end**
5 **for** *every new node* $i \in \mathcal{U}$ **do**
6 $\pi_i = \pi_0$
7 **end**
8 **for** *every node* i *that has been removed from* $\mathcal{U}$ **do**
9 $\pi_i = 0$
10 **end**
11 **for** *every new edge* $(i, j) \in \mathcal{L}$ **do**
12 $a_{i,j} = a_0$
13 $a_{j,i} = a_0$
14 **end**
15 **for** *every edge* (i, j) *that has been removed from* $\mathcal{L}$ **do**
16 $a_{i,j} = 0$
17 $a_{j,i} = 0$
18 **end**
19 **for** $i \in \mathcal{U}$ **do**
20 $\mu_i = w_i$
21 $\sigma_i = \Sigma$
22 **end**
23 Precompute forward (α_i), backward (β_i) and joint observation probabilities (p_O) for the observation sequence $O_{1:T}$
24 **for** $i \in \mathcal{U}$ **do**
25 $\pi_i = \pi_i + \frac{\alpha_t(i)\,\beta_t(i)}{P_O K}$
26 $a_{i,j} = a_{i,j} + \frac{\sum_{t=2}^{T} \alpha_{t-1}(i)\, p([S_t=j \mid [S_{t-1}=i]\ \lambda)\, p(O_t \mid [S_t=j]\ \lambda)\beta_t(j)}{\sum_{t=2}^{T} \alpha_{t-1}(i)\beta_{t-1}(i)}$
27 **end**
28 **end**

- Lines 2–4 update simply call the ITM-update for every observation in the sequence.
- Lines 5–18 insert or delete states and transitions in the HMM according to changes in the ITM. When nodes are inserted, the state prior is initialized to a default value π_0. Something similar is done for transition probabilities, which are initialized to a_0.
- *Lines* 19–27 update the HMM parameters. Since the covariance is fixed a priori, and mean values are computed as part of the ITM update, the observation probabilities may be updated directly from ITM's weights. The state prior, and transition probabilities, on the other hand, are recomputed using the well known forward and backward probabilities [24], but, in order to enable incremental learning, sums are stored in π_i and $a_{i,j}$ instead of probabilities. This permits to normalize the probabilities taking into account the whole observation history.

4.5 Predicting state

The tasks of maintaining an object's belief state and predicting its motion are carried on using exact inference. For a full connected HMM this would not work in real time, because exact inference performs marginalization over all allowed transitions ($O(N^2)$). In contrast, the number of allowed transitions in our approach depends linearly on the number of discrete states ($O(N)$) thanks to the use of the Delaunay triangulation, hence exact inference is viable even for relatively large HMMs.

The belief state of an object is updated on the basis of new observations by applying the following expression:

$$P(S_t \mid O_{1:t}) = P(O_t \mid S_t) \sum_{S_{t-1}} \left[P(S_t \mid S_{t-1}) P(S_{t-1} \mid O_{1:t-1}) \right]$$

where $P(S_{t-1} \mid O_{t-1})$ is the belief state calculated in the previous time step.

Prediction is made by propagating the belief state H time steps ahead into the future using the following expression:

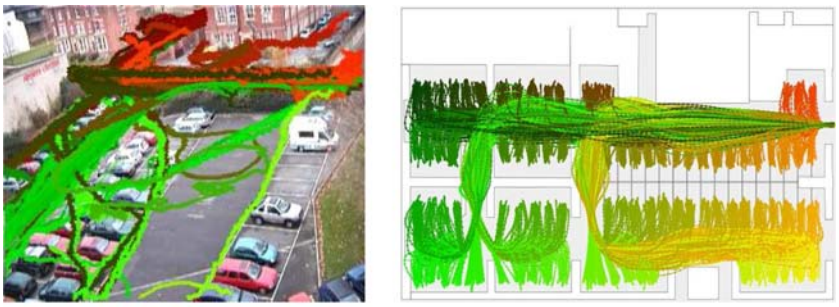
$$P(S_{t+H} \mid O_t) = \sum_{S_{t+H-1}} P(S_{t+H} \mid S_{t+H-1}) P(S_{t+H-1} \mid O_t) \tag{37}$$

5 Experiments

5.1 Experimental platform

A very relevant target application for the techniques presented in this paper is an automated valet Parking (AVP). The robotic system consists of 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 goes to park itself. As soon as the car enters the parking, the car on-board intelligent



(a) Leeds data **(b)** Simulated data

Fig. 19 Trajectory data sets used for testing motion prediction

system connects to the parking's own intelligent system and requests 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, can 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.

We have focused primarily in studying the application of the techniques described in this paper to the "smart parking" subsystem. For our experiments, we have placed a robot equipped with a LMS-291 laser scanner in the INRIA's car park, and captured 3,816 scans with it. Moving objects appearing in this data consisted mainly of members of our team which have moved freely in face of the robot.

For our motion prediction experiments, we have used two other data sets (Fig. 19): (a) a set of trajectories captured in a parking in the University of Leeds using a visual tracker; and (b) a set of synthetic data produced with a trajectory simulator (Fig. 20).

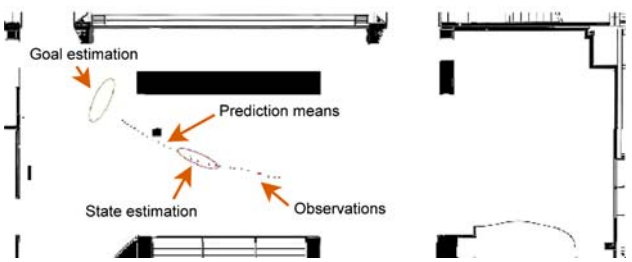


Fig. 20 Common elements in prediction example images

5.2 Wavelet occupancy grid performances

We performed experiments[7] on 2D real data in the presence of moving obstacles. The map have 900, 000 cells with a resolution of 10 cm × 10 cm. With these 2D data, the average computing time to build and fusion a grid is 29 ms (30 Hz) and the number of data gathered is 3,815 scans with a SICK LMS-291. The required memory for such a map is 3 kB (Fig. 21). It must be compared with the 3.6 MB required for such a number of cells: only 0.08% of the memory is necessary with the wavelet representation.

Comparisons with a standard grid construction algorithm were performed and show that there are no significant differences.

5.3 Clustering results

Unfortunately, at this moment we are not able to quantitatively evaluate our experiments due to the lack of ground truth information, obtained, for example, from a centimetric GPS. Nevertheless, our qualitative results seem promising (see Figs. 11, 12). Indeed, most of the time our algorithm correctly finds the number of objects in the scene and is able to detect that the non-connected regions corresponding to the legs of a person actually belong to the same object.

Concerning performance, the average processing time for our algorithm with a 3,600 node network on a 300 × 300 occupancy grid, is 6.7 ms per frame with an average number of cells above the threshold equal to 170. This seems to confirm the adequacy of our algorithm to its use in real-time, even if its performance may vary depending on the number of input cells above the threshold.

5.4 Prediction results

We have conducted extensive experiments with our motion prediction technique using the data sets mentioned above.

[7] This experiment was done with an Intel(R) Pentium(R) 4 CPU 3.00 GHz.

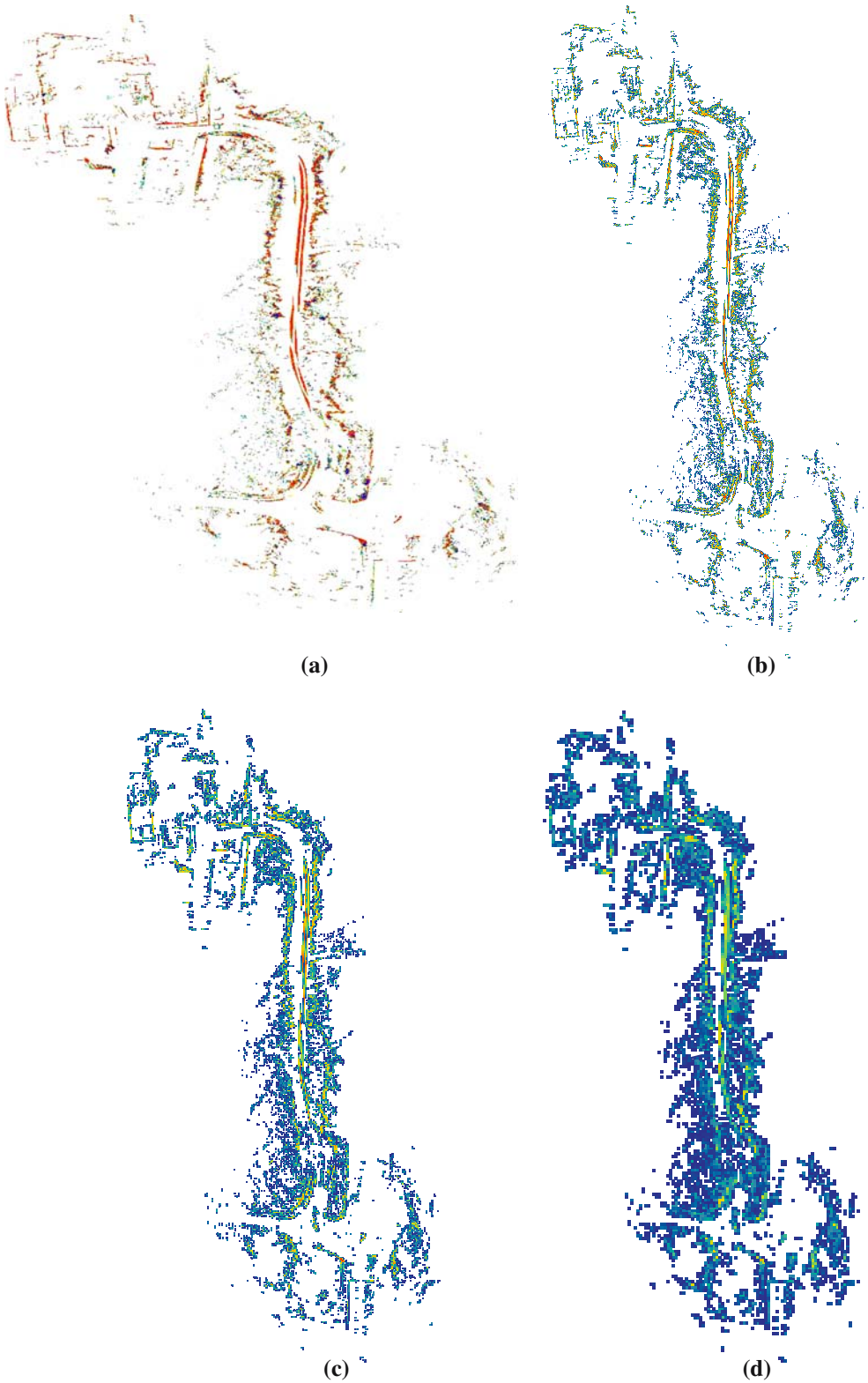


Fig. 21 Scaled view of OGs provided by different scales of WavOG representation **a** scale 0, cell size: 10 cm × 10 cm. **b** Scale 2, cell size: 40 cm × 40 cm. **c** Scale 3, cell size: 80 cm × 80 cm. (d) scale 4, cell size: 1.60 m × 1.60 m. The data were provided by CSIRO on a long-run experiment. The size of the map is approximately 450 m × 200 m. The unknown areas are not shown for visualization purposes. The number of cells vary from 9 millions to 36,000 from scale 0 to scale 4 and the ratio is 1 for 256 if the scale 4 is compared to the scale 0. It is obvious that the precision of the map decreases when the scale increases; however, the shape of the environment is sufficient enough for path planning since scale 3 in the big open area which is very interesting and promising for multiscale path planning algorithm for instance

Here, we will summarize some of the qualitative and quantitative aspects of our experimental results.

5.4.1 Qualitative results

Figure 22 shows a typical example of the prediction process using synthetic parking data. It consists of a set of images arranged in columns and rows. Rows correspond to different values of t.

The first column shows the mean values (blue to green points) of the predicted state from $t+1$ up to $t+15$. The second and third columns show the form of the predicted state and goal probability distributions, respectively, for $t+15$.

Images in the three columns share some elements (see Fig. 20): (a) the complete sequence of observations up to the current t, which is depicted as a series of red points; (b) the current state and goal estimations, or expected values, which are pictured as red and yellow ellipses, representing the expectation's covariance.

A particular difficulty of prediction in parking lots is that there are many places in the environment which could be an object's destination. This is apparent, for example, in row $t = 38$, where goals having a significant probability roughly cover half of the environment. Of course, this situation changes as the car continues to move and by $t = 49$, there are only two zones having a high probability of being the goal.

Having so many possible goals also influences the shape of the predicted state probability, which may become quite irregular, as for $t = 49$. This irregularity is also an indicator that a big number of alternative hypothesis that are being taken into account, which is unavoidable without additional information such as the free or occupied status of parking places.

5.4.2 Measuring prediction accuracy

A common performance metric for probabilistic approaches is the maximum data likelihood or approximations like the bayesian information criterion [31]. However, for our particular application, this metric has the drawback of not having any geometric interpretation. Intuitively, we would like to know *how far* was the predicted state from the real one. Hence, we have preferred to measure the performance of our algorithm in terms of the average error, computed as the expected distance between the prediction for a time horizon H and the effective observation O_{t+H}.

$$\langle E \rangle = \sum_{i \in \mathcal{S}} P([S_{t+H} = i] \mid O_{1:t}) \| O_{t+H} - \mu_i \|^{1/2} \tag{38}$$

for a single time step. This measure may be generalized for a complete data set containing K observation sequences:

$$\langle E \rangle = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{T^k - H} \times \sum_{t=1}^{T^k - H} \sum_{i \in \mathcal{S}} P([S_{t+H} = i] \mid O^k_{1:t}) \| O^k_{t+H} - \mu_i \|^{1/2} \tag{39}$$

It is worth noting that, as opposed to the standard approach in machine learning of conducting tests using a "learning" and a "testing" data sets, the experiments we have presented here will use only a single data set. The reason is that, since learning takes place after prediction, there is no need to such separation: every observation sequence is "unknown" from the perspective of prediction.

5.4.3 Leeds parking data

This is a difficult data set for a number of reasons: there are two different kinds of objects (vehicles and pedestrians) moving at very different speeds; the number of available trajectories is limited (267), hence, there are some trajectories which correspond to motion patterns that have been observed just once.

We have subsampled data by using only one out of three observations as input for our algorithm. The reason is that the position change that may be observed for pedestrians at full camera rate is far smaller than the detection noise: even assuming a relatively high mean speed of 5 Km/h, the position change between two consecutive frames at 24 frames/s is about only 6 cm. Building a model with the required resolution would be very expensive with only marginal benefits, if any, due to the high noise/signal ratio.

Figure 23 shows the evolution of both the model's size and the average error as a function of the number of processed trajectories. There seems to be an overall convergence process combined with sudden stair-like increases in both model size and average error (see for example the graph at 100, 160 and 250 processed trajectories). Actually, these increases correspond to the unique motion patterns that we have mentioned above.

Figure 23b shows a plot of the time taken by prediction and learning with respect to the number of trajectories already processed, the number of edges is also plotted as a reference. Times are given per-observation, hence, in the case of learning, they should be multiplied by the length of the observation sequence, which for this data set was 70 on the average. As it may be expected, learning and prediction processing times increase according to the growth in model size. Moreover, even in the worst case, prediction does not take more than 25 ms per observation, which is almost the double than

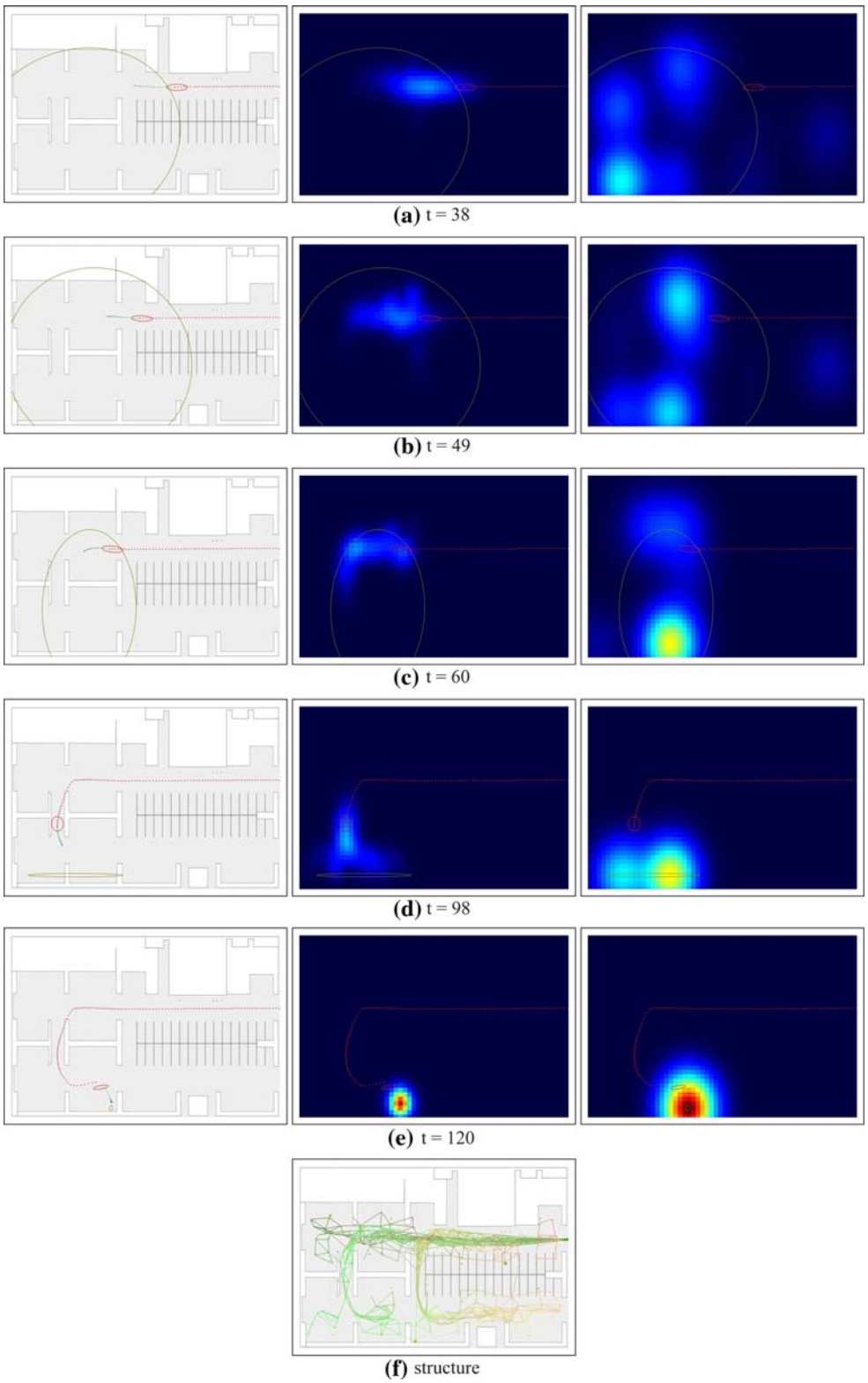


Fig. 22 Prediction example (parking environment). The *left colum* shows the object trajectory up to t in *red*, and the mean values of the predicted state, from $t + 1$ to $t + 15$ in *blue*. *Center* and *right columns* show the predicted position for $t + 15$ and the object's goal, respectively, using a grid representation

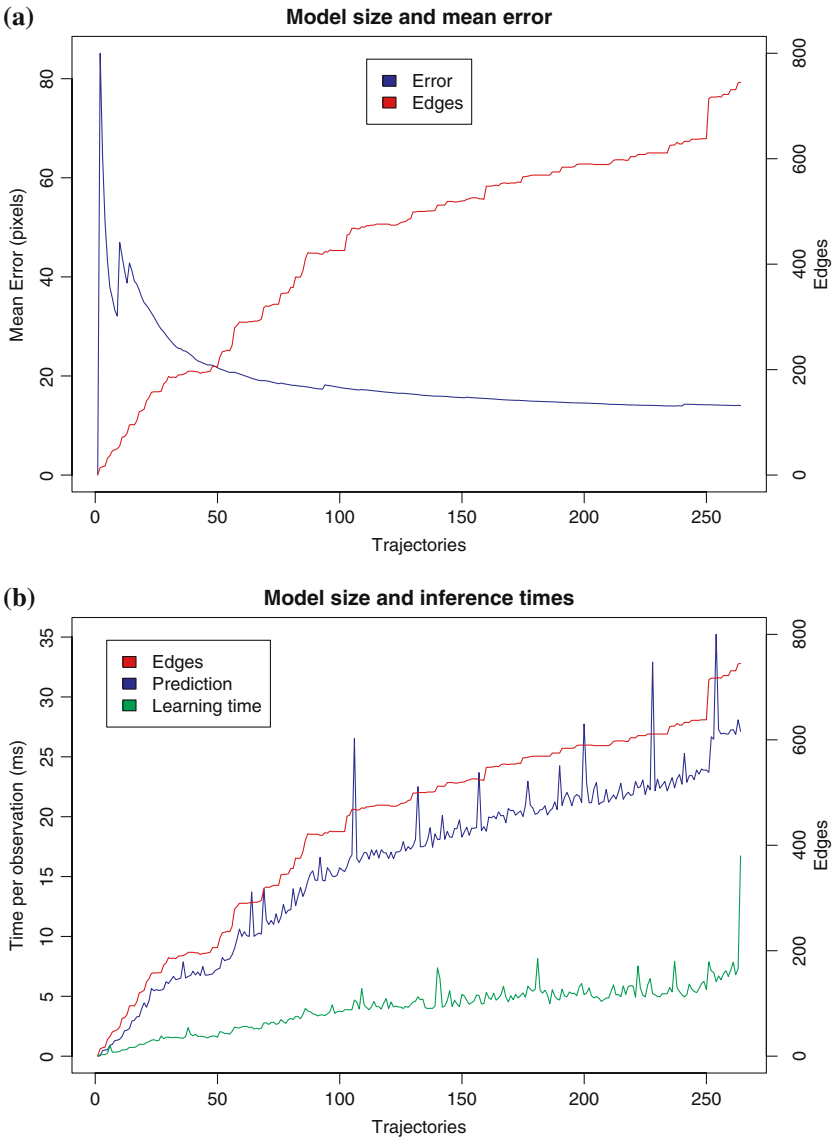


Fig. 23 Error and computation times for the Leeds Parking data set

normal camera frame rate. As for learning, it has always been less than 1 s per trajectory.

We may observe the presence of sharp peaks in processing times. This situation, which we will also observe in the other data set, has been caused by some unrelated process which has been executed concurrently with our tests thus reducing available CPU time for our task.

5.4.4 Synthetic parking data

This data set has two distinctive features: observations include velocities, meaning that the learnt structure occupies a six-dimensional manifold; and a big set of possible destinations, with about 90 parking places. As we will see, our algorithm performs surprisingly well, taking into account the added complexity.

Given the size of the data set and its features, it was surprising to find out that the error evolution and growth (Fig. 24a) in model size performed that well, with a final number of edges below 1,500 despite the diversity of goals. We believe that the main reason for this reduced size is that, due to the environment's structure, trajectories leading to goals in the same area tend to share a considerable number of states.

Because of the moderated model size, time performance was correct, with a prediction time of little less 60 ms, and an average learning time of about 3 s after 500 processed trajectories. Even if prediction times are slightly slower than camera frame rate, we think that there are very good results, taking into account the characteristics of this data set.

It is worth noting that the learning and prediction times for this example are longer than in the previous one, given an equivalent number of processed trajectories. For example,

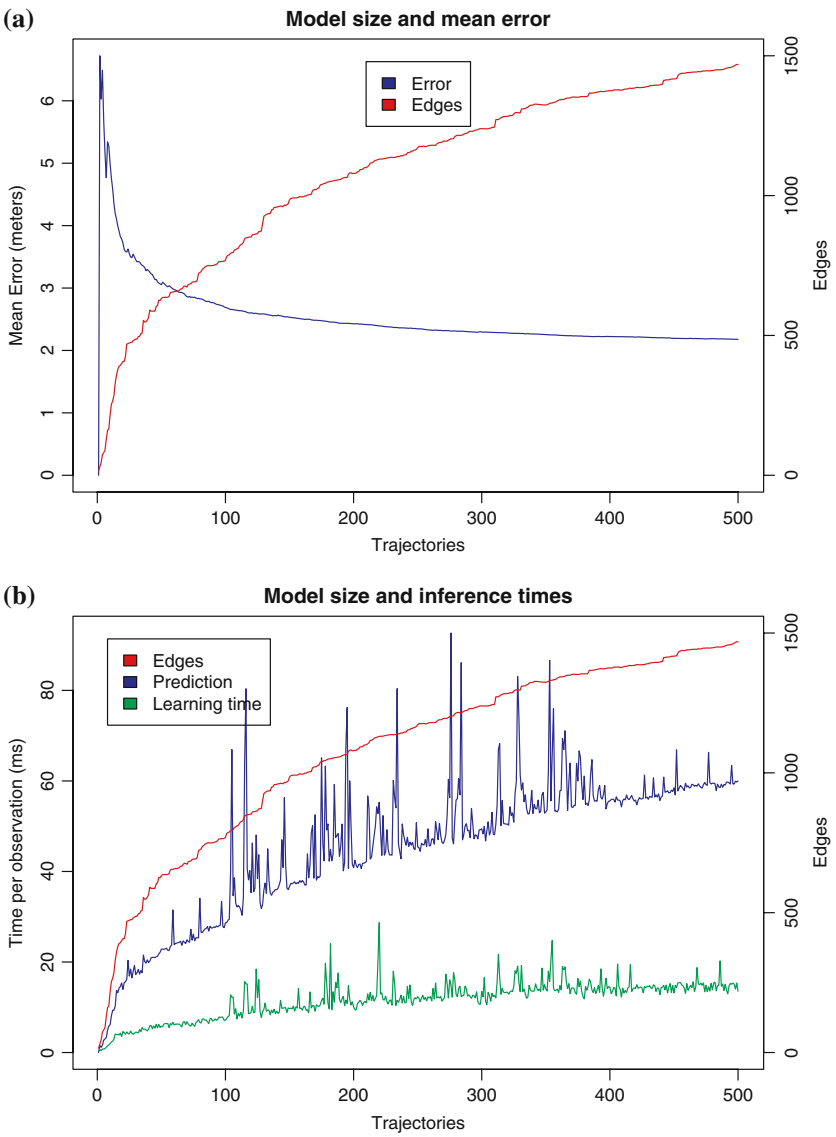


Fig. 24 Error and computation times for the synthetic parking data set

in Fig. 23, for 200 processed trajectories the learning and prediction times are around 20 and 5 ms, respectively; the equivalent values on Fig. 24 are around 40 and 10 ms, which are roughly the double. This is explained by the fact that the number of edges in the parking experiments is approximately the double of those obtained with Leeds data; this illustrates how the prediction and learning times grow in a linear fashion with respect to the size of the model.

6 Conclusions

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 three complementary questions:

1. How to build a multiscaled model of the environment allowing efficient reconstruction and updating operations?
2. How to detect moving obstacles in such an environment?
3. How to characterize and predict the motion of the observed moving entities?

Our answer to these questions relies on an appropriate combination of geometric and bayesian models, based on the following complementary approaches:

– *multiscale world representation* of static obstacles based on wavelet occupancy grid;
– *adaptive clustering* for moving obstacle detection inspired on Kohonen networks and the growing neural gas algorithm;

– *characterization and motion prediction* of the observed moving entities using incremental motion pattern learning in order to adapt a hidden Markov models.

We have validated our proposed approaches by performing experiments in the context of a specific application: the automated valet parking, at INRIA's installations. Our results show that the geometric and bayesian models we have used integrate well, and constitute a promising research direction towards achieving safe navigation in dynamic environments.

Although this paper only addresses some points, our team's research interests also cover the other tasks of the chain (Fig. 1). For an overview of our work, the reader is invited to consult our web page at http://emotion.inrialpes.fr.

Acknowledgments This work has been partially supported by several national and international projects and funding bodies: European projects *Carsense, CyberCars, PreVent/ProFusion & BACS*; French Predit projects *Puvame & MobiVip*; Mexican National Council of Science and Technology (CONACYT); ProBayes SAS and the french agency ANRT (GRANT 162/2004). We want to express our gratitude to Hannah Dee and the University of Leeds for allowing us to use their experimental data. Last, but not least, the authors would like to thank Christopher Tay Meng Keat, Christophe Braillon and Cedric Pradalier for their contributions and fruitful discussions and CSIRO for providing a consistent huge data set.

References

1. Elfes A (1989) Occupancy grids: a probabilistic framework for robot perception and navigation. PhD thesis, Carnegie Mellon University
2. Daubechies I (1992) Ten lectures on wavelets. No. 61 in CBMS-NSF series in applied mathematics. SIAM Publications, Philadelphia
3. Mallat S (1998) A wavelet tour of signal processing. Academic Press, San Diego
4. Stollnitz EJ, Derose TD, Salesin DH (1996) Wavelets for computer graphics: theory and applications. Morgan Kaufmann
5. Thrun S (2003) Learning occupancy grids with forward sensor models. Autonomous Robots, 15:111–127
6. Yguel M, Aycard O, Laugier C (to appear in 2007) Efficient gpu-based construction of occupancy grids using several laser range-finders. Int J Vehicle Autonomous System
7. Yguel M, Aycard O, Laugier C (2005) Wavelet occupancy grids: a method for compact map building. In: Proceedings of the international conference on field and service robotics, http://emotion.inrialpes.fr/bibemotion/2005/YAL05
8. Moravec HP, Elfes AE (1985) High resolution maps from wide angle sonar. In: Proceedings of the 1985 IEEE international conference on robotics and automation, pp 116–121, St Louis
9. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Cam LL, Neyman J (eds) Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, University of California Press, vol 1, pp 281–297
10. Dempster N Aand Laird, Rubin D (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc 9(1):1–38, series B
11. Kohonen T (1995) Self-organizing maps, Springer series in information sciences, vol 30. Springer, Berlin, (Second extended edition 1997)
12. Fritzke B (1995) A growing neural gas network learns topologies. Adv Neural Informat Proc Sys
13. Suzuki K, Horiba I, Sugie N (2000) Fast connected-component labeling based on sequential local operations in the course of forward-raster scan followed by backward-raster scan. In: Proceedings of the 15th International Conference on Pattern Recognition, Barcelona, vol 2, pp 434–437
14. Chang F, Chen CJ, Lu CJ (2004) A linear-time component-labeling algorithm using contour tracing technique. Computer Vision Image Understand 93(2):206–220
15. Schulz D, Burgard W, Fox D, Cremers A (2003) People tracking with a mobile robot using sample-based joint probabilistic data association filters. Int J Robotics Res
16. Johnson N, Hogg D (1995) Learning the distribution of object trajectories for event recognition. In: Proceedings of the british machine vision conference, vol 2, pp 583–592
17. Kruse E, Wahl F (1998) Camera-based observation of obstacle motions to derive statistical data for mobile robot motion planning. In: Proceedings of the IEEE international conference on robotics and automation, Leuven (BE), pp 662–667
18. Stauffer C, Grimson E (2000) Learning patterns of activity using real-time tracking. IEEE Trans Pattern Anal Machine Intell 22(8):747–757
19. Sumpter N, Bulpitt A (2000) Learning spatio-temporal patterns for predicting object behaviour. Image Vision Comput 18(9):697–704
20. Junejo I, Javed O, Shah M (2004) Multi feature path modeling for video surveillance. In: Proceedings of the 17th conference of the international conference on pattern recognition (ICPR), pp 716–719
21. Hu W, Xie D, Tan T (2004) A hierarchical self-organizing approach for learning the patterns of motion trajectories. IEEE Trans Neural Networks 15(1):135–144
22. Bennewitz M, Burgard W, Cielniak G, Thrun S (2005) Learning motion patterns of people for compliant robot motion. Int J Robotics Res 24(1):31–48
23. Hu W, Xiao X, Fu Z, Xie D, Tan T, Maybank S (2006) A system for learning statistical motion patterns. IEEE Trans Pattern Anal Machine Intelli 28(9):1450–1464
24. Rabiner LR (1990) A tutorial on hidden markov models and selected applications in speech recogn. Readings Speech Recognition. pp 267–296
25. Walter M, Psarrow A, Gong S (1999) Learning prior and observation augmented density models for behaviour recognition. In: Proceedings of the British machine vision conference, pp 23–32
26. Makris D, Ellis T (2002) Spatial and probabilistic modelling of pedestrian behavior. In: Proceedings of the British machine vision conference, Cardiff, pp 557–566
27. Martinetz T, Schulten K (1991) A "neural-gas" network learns topologies. Artif Neural Networks I:397–402
28. Martinetz T (1993) Competitive hebbian learning rule forms perfectly topology preserving maps. In: Gielen S, Kappen B (eds) In Proceedings of the international and conference on artificial neural networks (ICANN-93), Springer, Amsterdam, pp 427–434
29. Jockusch J, Ritter H (1999) An instantaneous topological map for correlated stimuli. In: In Proceedings of the international joint conference on neural Networks, Washington, vol 1, pp 529–534
30. Marsland S, Shapiro J, Nehmzow U (2002) A self-organizing network that grows when required. Neural Networks
31. Schwarz G (1978) Estimating the dimension of a model. Ann Stat 6(2):461–464