J Intell Robot Syst (2012) 66:75–109
DOI 10.1007/s10846-011-9627-8

Odometry-Based Viterbi Localization with Artificial Neural Networks and Laser Range Finders for Mobile Robots

Adalberto Llarena · Jesus Savage · Angel Kuri · Boris Escalante-Ramírez

Received: 9 December 2010 / Accepted: 25 July 2011 / Published online: 26 August 2011

Abstract This paper proposes an approach that solves the Robot Localization problem by using a conditional state-transition Hidden Markov Model (HMM). Through the use of Self Organized Maps (SOMs) a Tolerant Observation Model (TOM) is built, while odometer-dependent transition probabilities are used for building an Odometer-Dependent Motion Model (ODMM). By using the Viterbi Algorithm and establishing a trigger value when evaluating the state-transition updates, the presented approach can easily take care of Position Tracking (PT), Global Localization (GL) and Robot Kidnapping (RK) with an ease of implementation difficult to achieve in most of the state-of-the-art localization algorithms. Also, an optimization is presented to allow the algorithm to run in standard microprocessors in real time, without the need of huge probability gridmaps.

A. Llarena (✉) · J. Savage · B. Escalante-Ramírez
Universidad Nacional Autónoma de México UNAM, Ciudad Universitaria no. 3000, Col. Copilco Universidad, Del. Coyoacán, México, D.F., C.P. 04360, México
e-mail: adallarena@aol.com

J. Savage
e-mail: savage@servidor.unam.mx

B. Escalante-Ramírez
e-mail: boris@servidor.unam.mx

A. Kuri
Instituto Tecnológico Autónomo de México ITAM, Río Hondo No. 1, Col. Progreso Tizapán, México, D.F., C.P. 01080, México
e-mail: akuri@itam.mx

Keywords Artificial neural networks · Simultaneous localization and mapping · Mobile robots · Hidden Markov models

1 Introduction

In mobile robotics, one of the most basic problems to be solved is the Simultaneous Localization and Mapping or SLAM problem [1], where an autonomous robot must be capable of generating a representation often called a ‘map’ of an unknown environment, at the same time that traverses it and gets localized into that environment representation.

Although robotic localization is considered a solved problem, it is far from being a closed investigation topic. It involves three aspects: (a) data collecting and pre-processing to build an observation model, (b) *a priori* motion kinematics knowledge for building a movement model (or odometer estimations, not always incorporated on cheaper or smaller robots) and (c) an observation-movement relationship, used for updating the position estimation through time.

While computers are approaching the processing power of the human brain [2], the complexity of operating systems, algorithms and the sensorial robot information is also increasing. For this reason, algorithms and architectures suitable of being adapted or re-trained will constitute a good alternative in the future. The best example is the animal brain, capable of performing very complex tasks and adapting to the environment, all just done with neurons. Based on these premises, a kind of Artificial Neural Network (ANN) inspired in the way the brain organizes information, the Self Organized Map (SOM) has been widely used in robotics because its properties for organizing high-dimensional observation vectors. In this way, the nature has helped the researchers in finding novel ways to manage the increase in the data to be processed.

This paper is organized as follows: Sections 1 to 3 present the general robotic localization problem and the most common solutions up to date. Sections 4 and 5 introduce the Viterbi Localization Method (VL) and propose a solution based on ANNs and odometry-based motion models named Odometry-dependent Viterbi Localization (OVL). Section 6 presents an error-tolerant observation model by using SOMs. Section 7 explains the full path reconstruction from discrete sample nodes. Section 8 shows the method optimizations. Section 9 shows the OVL Algorithm. Sections 10 and 11 present the experiments and show the results. Finally, Sections 12 to 14 present the conclusions and the future work.

2 Previous Work

2.1 Robot Localization with Artificial Neural Networks

Since the first efforts for effectively locating a robot using ANNs the main purpose has been relating the robot pose as a function exclusively of the observations. Although this has been made possible using several kinds of ANNs such as Feed Forward Networks (FFNs) [3], Hopfield [4], Kohonen Self Organized Maps (SOMs) [5], Fuzzy-Adaptive Resonance Theory (Fuzzy-ARTs) [6], General Regression Neural Networks (GRNNs) [7] or Multi-Layered Perceptrons (MLPs) [8], such direct relationship between pose and observations makes it difficult to carry on a mixed hypothesis based on both odometry estimations and perceptual data because the displacement information is ignored.

2.2 Robot Localization and Hidden Markov Models

Probabilistic localization with Hidden Markov Models (HMMs) has been explored in the past by [9, 10] with good results. Unfortunately, with the discretization of the configuration space, large amounts of memory were required for storing a location probability distribution. For this reason they and other researchers started a new successful approach known as Monte Carlo Localization (MCL) [11]. The main feature of this method is the use of probabilistic Motion and Observation Models (MM and OM respectively). With the aid of these models, the new robot pose Probability Density Function (PDF) can be estimated based on odometry data and by comparing the actual observation against the stored model. This important modeling approach is also incorporated into the Kalman Filter (KF) Localization [12] technique and makes probabilistic methods tolerant to occlusions and sensor noise.

2.3 Viterbi Localization

In 2005, Savage et al. [13] proposed a method that used the Viterbi Algorithm to solve the robot localization seen as a HMM with a fixed-probability state-transition model. That method was unable to give precise location estimations, due to the lack of an odometry-dependent Motion Model (ODMM). The present work extends that approach with an ODMM, incorporating rescaling techniques in order to avoid the tendency of probabilities to zero out after several iterations.

3 Robot Localization and HMMs

3.1 The Robot Localization Problem

The robot localization problem consists of determining the robot's pose with respect to some known references (a common origin in a Cartesian plane or environment landmarks), involving three basic aspects:

1. *Global Localization (GL)*. The mobile robot must determine its initial pose, based on the universe of known locations and sensor readings.
2. *Position Tracking (PT)*. The robot has prior knowledge about its previous location and keeps track of the position changes, through the integration of the successive displacements calculated by the PT algorithm.
3. *Robot Kidnapping (RK)*. The robot is abruptly taken out of its current location and it is repositioned into an arbitrary place of the working environment, but the robot is not informed about this change. As a consequence, the robot must be able to detect the position change and find its global location again.

Robot localization intends to determine the actual robot pose $\boldsymbol{x}_t$:(x_t,y_t,θ_t), where (x_t,y_t) are the robot coordinates in the Cartesian plane and θ_t is the robot's heading, based on a set of control actions $a_{1:t-1}$, a set of odometry displacements estimations $u_{1:t}$, a set of previous visited locations $\boldsymbol{x}_{0:t-1}$ and a sensor readings set $z_{0:t}$. It is assumed that the robot starts at location $\boldsymbol{x}_0$ observing z_0 (Fig. 1).

A new control action a_1 is issued and the robot performs some displacement $\Delta\boldsymbol{x}_1$ according to its kinematics, arriving at location $\boldsymbol{x}_1$ with odometer estimation u_1 and observing z_1. A new action a_2 is issued, displacing the robot $\Delta\boldsymbol{x}_2$, arriving at $\boldsymbol{x}_2$, observing z_2 and estimating u_2. This process continues up to some time instant t, with a_t, $\Delta\boldsymbol{x}_t$, $\boldsymbol{x}_t$, u_t and z_t, where a re-localization method is launched to correct the robot position.

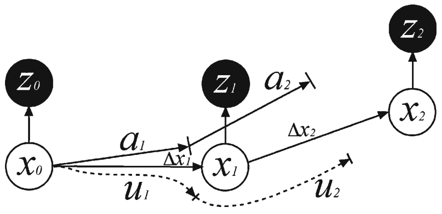

Fig. 1 Robot Localization Problem. The robot starts at x_0 perceiving z_0. As the robot moves, it looses certainty in the pose estimation. [a_1, a_2]: planned trajectory. [u_1, u_2]: odometry estimation. [Δx_1, Δx_2]: actual displacement

3.2 Classical Approaches in Robot Localization

Historically, two kinds of methods have been developed to solve the localization problem, *data–data* and *data–model* associations.

3.2.1 Data–data Associations

The *data–data* association calculates x_t as

$$x_t = f\left(x_{0:t-1}, z_{0:t}, u_{1:t}, a_{1:t-1}\right) \tag{1}$$

The simplest way to do this association is by collecting and storing a set of observations at known locations L: (x_i, z_i), gathered in a previous sampling phase. It is possible to find the closest observation z_k in L to a given observation z_t and assuming corresponding x_k as the true robot location. Although this association directly relates x_i with z_i and vice versa (apparently solving GL and RK but not PT), this procedure is not completely adequate because it finds the location most similar to a given observation, regardless of robot motions and sensor errors or occlusions.

Another kind of methods like [14] calculate successive robot displacements with x_t as $x_{t-1} + \Delta x_t$ where

$$\Delta x_t = f\left(z_t, z_{t-1}\right) \tag{2}$$

By exploiting a relation between z_{t-1} and z_t they are able to find the exact amount of displacement Δx_t. They are adequate for calculating PT by integrating successive Δx_t but cannot therefore deal neither with GL nor RK.

Because *data–data* approximations directly relate position with observations they result very sensitive to sensor noise and occlusions. Moreover, they are incapable of deciding between two possible robot locations x_1 and x_2 with identical observation vectors, due to the lack of movement estimators.

3.2.2 Data–model Associations

The data–model associations calculate x_t by incorporating some robot and world models. While traversing the environment, some conditions must be met according with those models, thus:

$$x_t = f\left(x_{0:t-1}, z_{0:t}, u_{1:t}, a_{1:t-1}, \Psi\right) \quad (3)$$

where

$$\Psi : \{S, W, O, M\} \quad (4)$$

is a set of models regarding the robot sensors S, the world model W, a location–observation relationship O and a motion model M.

3.3 The Sensor Model

The Sensor Model stores a representation about the way that specific sensors (like range finders, temperature sensors or digital cameras) behave under real world inputs. They model the sensor output z_k for a given measuring condition ϑ_r, based on specific sensor parameters ς_k according to

$$z_k = S\left(\vartheta_r, \varsigma_k\right) \quad (5)$$

where ϑ_r is a measuring condition vector comprising important parameters such as the object's distance, scanning angle, surface incidence angle (for range scanners), surface color and roughness and etcetera. ς_k is the specific sensor parameter vector containing information such as sensibility, operative range, distortion and etcetera; used for predicting the noise added to z_k by the S function. For digital cameras ϑ_r is equivalent to extrinsic parameters while ς_k corresponds with the intrinsic parameters.

3.4 The World Model

The World Model W, often implemented trough a *map*, has a representation of the robot environment (such as the physical location of the objects and their attributes). It constitutes the reference frame where the robot is located into. It is important to have all the World Model's information as sensor independent as possible (the World Model must be *complete*). Regardless of the impossibility for any sensor to detect some features from a certain location inside the world (like black polished objects, invisible for the laser range finder) the World Model must incorporate as much characteristics as possible. In this sense, Sensor Fusion Methods [15, 16] are relevant to build more complete World Models than those conformed only with information coming from a single sensor.

Several kinds of maps are commonly used as world models such as occupancy grids [17], metric maps [18], topological maps [18, 19], feature maps [20] and many others. In any case the main goal in SLAM will be making the robot able to build by itself those maps, while exploring an unknown environment.

3.5 The Observation Model

The Observation Model (OM) O stores a direct association between locations x_r and observations z_r for a given sensor as

$$z_r = O\left(x_r\right) \quad (6)$$

Most of existing localization algorithms consider the environment as static. This issue simplifies the location procedure by applying the Markov Assumption [21] where the observation made at certain state is only dependent of the state itself (the current robot's pose). Under normal operation conditions only a few section of the environment keeps static, basically the walls and some big furniture, the rest of objects (including human and robots) are constantly moving. For this reason, it is necessary to differentiate between *scene*, *view* and *observation*.

3.6 Scene, View and Observation

A *scene* corresponds with the instantaneous state W_t of all the actors inside the environment (i.e. the location of all the objects in the World Model). Inside a static environment as several methods propose, the world model does not change over time, so it warranties the applicability of the Markov Assumption.

A *view* is a particular section of the environment, only visible from a particular location and depending on some sensor parameters such as absolute location and heading, pan, tilt and field of view. In other words, a view is a portion of the environment suitable of being processed by the sensor at a given location.

An *observation* is the measure that a specific sensor performs from certain view. It is obvious that depending on the sensor model, and observation conditions, some of the world characteristics will not be perceivable by the sensor so the measure condition will lead to bad measuring errors produced for instance by lens distortion or moisture in case of having digital cameras.

3.7 Simulating the Observations

As an alternative to implementation of one Observation Model for every robot sensor by gathering a huge set of sample readings, the association x_r–z_r is often achieved by using a software simulator that predicts observation conditions ϑ_r from a given location x_r by using a world model W, having on mind specific sensor parameters ς_k as field of view, resolution (dimensionality), maximum and minimum measure ranges and etcetera:

$$\vartheta_r = O\left(W_t, x_r, \varsigma_k\right) \tag{7}$$

The use of a simulation method is especially adequate when working with high dimensional observation vectors like cameras or range finders. By using ray-tracing techniques (for range based algorithms) or 3D renders and virtual cameras (for simulating the actual observation for vision systems) expected observations can be generated with the expense of processor and memory of course. Many authors build a readings [22] or features [23] databases with optimized algorithms like Kd-trees [11] to search for the direct or inverse location–observation value.

3.8 The Motion Model

Often referred as *Plant Model* in Probabilistic Localization [1], the Motion Model (MM) establishes the estimated movement the robot will perform under certain control action a_t based on odometry estimations u_t. Examples of these models are differential drive, omni-directional, synchro-drive or legged robot control. This model predicts the final robot location after performing the a_t command.

Usually robots incorporate odometers to help the control modules in estimating the robot displacements based on u_t.

In general, a Motion Model is a transfer function that computes the new robot location x_{t+1}, in terms of current location x_t, robot constrains ρ_r and odometry data u_t.

$$x_{t+1} = M\left(x_t, u_t, \rho_r\right) \tag{8}$$

The Motion Model imposes the restrictions a mobile robot must meet between consecutive movements. While a set of encoders is supposed to be mounted on the robot wheels and the robot dimensions and mechanics are known, the Motion Model establishes which locations transfers are valid (or more 'probable' in terms of probability theory) between consecutive movements, based either on odometry estimations u_t (for wheeled robots), the commanded movement a_t or a movement statistical analysis (for legged robots).

In this way, with the help of the abovementioned models, *data–model* associations match actual observations against those predicted by OM and MM.

3.9 Probabilistic Localization

The most widely used *data–model* localization approaches are of probabilistic nature, where the goal is estimating the posteriori Probability Density Function (PDF) $p(x_t)$ according to

$$Bel\,(t) = p\left(x_t | z_{0:t}, x_{0:t-1}, a_{1:t-1}, u_{1:t-1}\right) \tag{9}$$

where x_t is called the *robot state* representing the robot pose.

The most popular probabilistic localization methods are Kalman Filters (KFs) and Particle Filters (PFs). Both are basically *data–model* associators.

Probabilistic methods are also known as Bayesian Filters because they support their operation upon certain *a-priori* assumptions:

3.9.1 Markov Assumption

The Markov Assumption [21] considers observations z_t dependent only on the current state x_t (and the immediate previous and next states), this is

$$p\left(z_t|z_{0:t-1}, x_{0:t}, a_{1:t}, u_{1:t}\right) = p\left(z_t|x_t\right) \tag{10}$$

and therefore

$$p\left(x_t|x_{0:t-1}, a_{1:t-1}, u_{1:t-1}\right) = p\left(x_t|x_{t-1}, u_{t-1}\right) \tag{11}$$

in consequence

$$p\left(z_t|a_{1:t-1}, u_{1:t-1}\right) = p\left(z_t\right) \tag{12}$$

The Markov Assumption is not completely valid in crowded environments or scenarios with moving objects where observations are difficult to predict only in function of the current robot location. More robust methods incorporate also the robot observation history in order to give more accurate predictions.

3.9.2 Bayes Rule

In order to calculate $Bel(t)$, KFs and PFs apply Bayes Rule [24], having on mind Eqs. 10 to 12 as

$$p\left(x_t|z_{0:t}, x_{0:t-1}, a_{1:t-1}, u_{1:t-1}\right) = \frac{p\left(z_t|x_t\right)\ p\left(x_t|u_{t-1}\right)}{p\left(z_t\right)} \tag{13}$$

While KFs keep track of the position using parametric continuous Gaussian functions and a linear motion and observation models (something very difficult to achieve in real scenarios), PFs carry on a multi-hypothesis posterior PDF through a set of individual samples called 'particles', each one of them corresponding with a possible robot location.

3.10 Kalman Filters

Kalman Filters [12] have a linear Motion Model (so called the *Plant Model*) of the form

$$x\left(k+1\right) = \Phi x\left(k\right) + \Gamma u\left(k\right) + Cv \tag{14}$$

where $x(k)$ is the previous state, Φ indicates the state variation in the absence of inputs, Γ is a state-transition matrix, $u(k)$ is the control command and Cv is the state covariance matrix.

They also consider a linear Observation Model

$$z_i\left(k+1\right) = \Lambda_{\mathrm{E}} x\left(k\right) + W_i \tag{15}$$

where $z_i(k+1)$ indicates the sensor reading at time $k+1$, Λ_E is a matrix relating observations with robot states $x(k)$. $W_i(k)$ is a zero-mean Gaussian noise function.

In its main stage, KFs compute a prediction of the sensor readings for the next state $k+1$ by using Bayes Rule with

$$K_t = \bar{\Sigma}_t C_t^T \left(C_t \bar{\Sigma}_t C_t^T + Q_t\right)^{-1} \tag{16}$$

where $\bar{\Sigma}_t$,C_t and Q_t are matrixes. K_t is known as Kalman Gain and requires a matrix inversion with O($k^{2.4}$) [25] being k the dimension of the observation vector.

Kalman Filters, carry on a robot location estimate based on the parameters of Gaussian functions, they assume a Gaussian, zero mean movement and observation errors. This, together with the linearity model assumption and matrix inversion makes difficult to implement KFs under real world conditions in limited devices. For this reason some improvements have been developed as the Extended Kalman Filter EKF [26], the Unscented Kalman Filter UKF [27] and the Information Filter IF [1]. KFs can carry on only one localization hypothesis [28], limiting their use in changing environments.

3.11 Monte Carlo Localization

Particle Filters surpass some of the KFs limitations by approximating the posterior PDF $p(x_t)$

by a set of samples called 'particles'. The particle density over the configuration space reflects the probability $p(x_t)$ calculated according to Bayes Rule as

$$Bel\left(x_t\right)=\eta p\left(z_t|x_t\right)\int p\left(x_t|x_{t-1},u_t\right)Bel\left(x_{t-1}\right)dx_{t-1} \tag{17}$$

where $Bel(x_t)$ is the pose belief at time t, $p(z_t|x_t)$ is the Observation Model, $p\left(x_t|x_{t-1},u_t\right)$ is the Motion Model, $Bel(x_{t-1})$ the pose belief at time $t-1$ and η is a normalizer.

The most popular PF is the Monte Carlo Localization method (MCL) [11]. MCL approximates Eq. 17 by randomly sampling locations according to the Motion Model $p\left(x_t|x_{t-1},u_t\right)$, based on robot motions and odometer estimations u_t. After a sample set is generated, an importance weight w_i is calculated for every particle location x_i based on the Observation Model $p(z_t|x_t)$. After computing all pairs (x_i, w_i), every w_i is rescaled, multiplying its value by their individual scaling factor η_i (equivalent to $p(z_t)$ in Bayes Rule) calculated with

$$\eta_i=\frac{w_i}{\sum\limits_{j=1}^{n}w_j} \tag{18}$$

where n is the size of the sample set in the PF.

Then, a method known as *Importance Sampling* renews the entire particle set by generating new random particles, based on importance factors w_i, generating more new samples in those locations with higher matching between real observations z_t and supposed observations z_x for that location. This process avoids the effects of progressive degradation of $Bel(x_t)$ due to the implicit computation of probabilities that tend to zero.

In many MCL implementations for laser range finders, the OM is computed by using a ray-tracing simulator in real-time, based on world model W or a set of stored observations L at known locations and an optimized-search method. Observation probability is calculated by evaluating a Probabilistic Sensor Model (as the one proposed in [11] for range scanners) for every simulated individual reading z_{x_i} and the actual individual sensor reading z_{r_i} according with

$$p\left(z_t|x_t\right)=\prod_{i=1}^{n}p\left(z_{t_i}|z_{x_i}\right) \tag{19}$$

By incorporating measuring errors and occlusion probabilities in the Probabilistic Sensor Model $p\left(z_{k_i}|z_{x_i}\right)$ the MCL filter is capable to deal with both sensor errors and occlusions. By generating a small extra set of randomly samples all over the configuration space eventually MCL can recover from robot kidnapping.

Finally, as every particle has an importance weight associated to it, the current robot location can be computed as

$$E\left(x_t\right)=\sum_{i=1}^{n}\left(x_i\cdot w_i\right) \tag{20}$$

In conclusion, MCL has some important advantages

1. *Multi Hypotheses*. The sample set distribution allows managing many possible robot locations at once. Once the particles have converged to a narrow area, it can be assumed as the true robot pose.
2. *Completeness*. By having an evenly distributed initial particle set, MCL can handle GL. By adding some random particles across all the configuration space in the resampling process MCL can eventually recover from a RK. As robot motions u_t are used for sampling the new particle set, MCL can handle PT.
3. *Memory Reduction*. It reduces drastically the amount of memory required by discrete localization techniques that use three-dimensional spatial grids as Markov Localization [9]. It is capable of integrating observations at high rates.
4. *Easiness of Implementation*. MCL It is easy to implement in systems with limited resources of memory and speed, because it does not require matrix inversion like KFs.

Unfortunately, MCL also has some disadvantages

1. *Sensor Error*. The MCL performance decays when having sensors with low error rates (like laser range finders).

2. *Simulation Processor Cost*. As each particle represent a possible robot location, the Observation Model must be able to predict a precise observation for each particle, this is often performed with the help of a software simulator. When managing laser range finders with hundreds of readings per scan or mounted vision cameras, the simulation process results very expensive in terms of processor consumption.
3. *Number of Particles*. The MCL performance depends mostly on the re-sampling process. At less one particle must be located very close to the actual robot position to ensure the method's convergence. In some scenarios bigger than a dozen of meters long and width, thousands of particles can be needed for performing GL or RK.
4. *Independent Sensor Readings*. As proposed in Eq. 19 individual sensor readings are considered as conditionally independent. The true is exactly the opposite: individual sensor readings z_{r_i} are highly correlated.

Some additions to MCL have been proposed by the authors to overcome some of these disadvantages like an inverse sampling model Mixture-MCL [11] for predicting locations with a Kd-tree forest storing observations, or the Rao-Blackwellized Particle Filters [29] that carries on a whole map with every particle. They both increase the performance of PFs but also their complexity and thus the required computer resources for running those algorithms.

In this form, spatial grids and particle filters have proved their applicability to the SLAM problem [11], but they require large amounts of processing time and memory resources. Topological localization [13] has proved its simplicity in terms of computational resources but they fail in the presence of occlusions and partial readings due to range sensors sensing limits. By the other hand, one fundamental issue is the inherent complexity of the localization problem. It requires many resources for every square meter added to the environment.

3.12 Hidden Markov Models and Robot Localization

Hidden Markov Models (HMMs) [21] are probabilistic directed graphs, where the current state $x_i \in X{:}[x_1 .. \ x_n]$ is not directly observable (i.e. measurable). There is an observation $z_i \in Z{:}[s_1 .. \ s_m]$ associated with every state in the model and through the observation sequence and model parameters the hidden variable (the state) can be estimated. X is a set of states and Z is a set of observations (symbols).

The parameters of a Hidden Markov Model λ are

$$\lambda = (\mathbf{A}, \mathbf{B}, \mathbf{\Pi}) \tag{21}$$

where $\mathbf{A} = \{a_{ij}\}$ is a state-transition probability distribution over $X \times X$ such as

$$a_{ij} = p\left(x_t = x_j | x_{t-1} = x_i\right), \quad 1 \le i \le n \tag{22}$$

$\mathbf{B} = \{b_j(k)\}$ is an observation probability distribution over $X \times Z$ (relating states with observed symbols) such that

$$b_j(k) = p\left(z_t = s_k | x_t = x_j\right), \quad 1 \le k \le m \tag{23}$$

and $\mathbf{\Pi} = \{\pi_i\}$ is an initial state distribution

$$\pi_i = p\left(x_t = x_i\right), \quad 1 \le i \le n \tag{24}$$

where n is the number of states and m is the number of symbols in the HMM.

$\mathbf{A}$, $\mathbf{B}$ and $\mathbf{\Pi}$ can be expressed in matrix form: $\mathbf{A}(n \times n)$ is the state transition matrix, $\mathbf{B}(n \times m)$ is the observation matrix and $\mathbf{\Pi}(1 \times n)$ is the initial state probability matrix.

4 Viterbi Localization

As proposed in [13], robot localization can be seen as the process of estimating the hidden variable in a HMM (the current state x corresponding to a discrete robot pose, see Fig. 2), based on the sensor observation sequence $\{z_0, z_1, ..., z_t\}$.

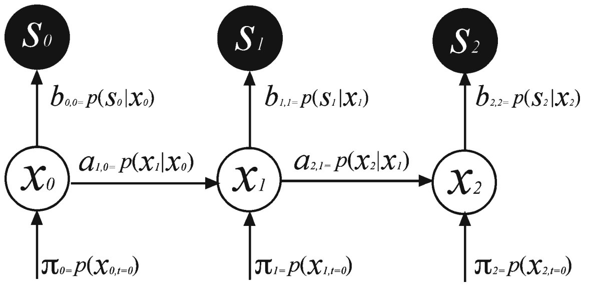

Fig. 2 Robot Localization Problem as HMM. Every state has transition and observation probabilities associated to it (not all of them are drawn)

4.1 The Viterbi Algorithm

The Viterbi Algorithm (VA) [30] proposes a solution for estimating $p(O|\lambda)$, i.e. the probability of the observations O given the model λ, where $O=(z_0, z_1,..., z_t)$ is the observation sequence and λ is the HMM. The Viterbi Algorithm has two ways of calculating $p(O|\lambda)$, the forward and backward procedures.

The Viterbi's forward procedure (VFP) has three steps:

A. Initialization

$$\alpha_1(i) = \pi_i b_i(z_1), \quad 1 \le i \le n \tag{25}$$

$\alpha_1(i)$ stores the joint probability $p(x_i, z_1)$ of being at state i at $t = 0$ and observing z_1, equals to the probability π_i of starting at state i by the probability of observing the symbol z_1 at that state, while n is the number of states in the model.

B. Induction

$$\alpha_t(j) = b_j(z_t)\left[\sum_{i=1}^{n} a_{ij}\alpha_{t-1}(i)\right] \quad \begin{array}{l} 1 \le t \le t_k \\ 1 \le j \le n \end{array} \tag{26}$$

where t_k *is* the time when the last observation z_k was taken.

$\alpha_t(j)$ computes the joint probability $p(x_j, z_{0:t})$ of being at state i at time t while observing z_t. By summing the probability $p(x_j|x_i, z_{0:t-1})$ of performing a transition to the state j from every state i (total probability) $p(x_j, z_{0:t-1})$ is obtained. If this result is multiplied by the probability of observing the symbol z_t at the state j then results $p(x_j, z_{0:t})$, because $p(x_j, z_{0:t}) = p(z_t|x_j)p(x_j, z_{0:t-1})$.

C. Termination

$$p(O|\lambda) = \sum_{i=1}^{n} \alpha_t(i) \tag{27}$$

where

$$\alpha_t(i) = p(x_t = x_i, z_0, z_1, \cdots, z_t|\lambda). \tag{28}$$

By rewriting Eq. 26, according with 23 and 24

$$\begin{aligned}\alpha_t(j) &= p(z_t|x_j)\sum_{i=1}^{n} p(x_j|x_i)\,\alpha_{t-1}(i)\\ &= p(z_t|x_j)\,p(x_j, z_{0:t-1}) = p(z_{0:t}, x_j)\end{aligned} \tag{29}$$

can be noticed that Eq. 26 computes the joint probability of the observation sequence up to z_t finishing at state x_j. By summing all $\alpha_t(j)$ with $1\le j \le n$, $p(z_{0:t})$ is obtained in Eq. 27.

4.2 Viterbi Algorithm vs. Probabilistic Localization Approaches

Main probabilistic localization methods estimate the robot location given the observation sequence, this is $p(x_j|z_{0:t})$. They apply the Bayes Rule (BR) to simplify this calculation as $p(x_j\,|\,z_{0:t}) = p(z_{0:t}|\,x_j)\;p(x_j)\,/\,p(z_{0:t})$.

By analyzing the Particle Filters (PFs) update rule [11]

$$Bel(x_t) = \eta p(z_t|x_t)\int p(x_t|x_{t-1}, u_t)\,Bel(x_{t-1})\,dx_{t-1} \tag{30}$$

it can be seen that the BR is applied by having $\eta = p(z_t)$. If we compare Eq. 29 with Eq. 30 it can be seen that Eq. 29 is the discrete case of Eq. 30, except by two important absences: the term u_t (the motion estimation) and the normalizer η.

In a similar way, Eq. 22 corresponds to $p(x_t|x_{t-1}, u_t)$, the Particle Filter Motion Model M (the state-transition rule in Mobile Robotics, regardless of u_t) while Eq. 23 would precisely correspond to an Observation Model O (the relationship between locations and observations). In this sense, Particle Filters implement continuous versions of the Viterbi Algorithm with an u_t-dependent Motion Model. As shown above, Observation and Motion Models have their origin in the Viterbi Algorithm.

4.3 Viterbi Forward Procedure

The Viterbi Forward Procedure (VFP) in Eq. 29 computes the observation probability in a very similar way that PFs do, except by the normalizer η and the odometer estimation u_t. By modifying the VFP incorporating these parameters, robot pose estimation can be computed in a discrete form, without the need of solving a continuous model as Eq. 30 with the corresponding saving of time and resources. If the physical location of the states is known in advance, then the location–observation relationship can be built off-line, avoiding the need of precise observation estimations in real-time as PFs and KFs need [11, 12] often computed with the aid of a software simulator (through a metric map and ray-tracing techniques for simulating range scans at given locations).

By substituting Eq. 26 in Eq. 27 it can be shown that

$$\sum_{i=1}^{n} p\left(x_i, z_{0:t}\right) = p\left(z_{0:t}\right) \tag{31}$$

and therefore

$$p\left(x_t | z_{0:t}\right) = \frac{\alpha_t\left(i\right)}{\sum\limits_{j=1}^{n} \alpha_t\left(j\right)} \tag{32}$$

that exactly corresponds to the rescaling factor η in PFs.

In order to solve Eq. 29, $p(x_j\,|\,x_i)$ can be computed as

$$p\left(x_j | x_i\right) = \int p\left(x_j | x_i, u_t\right) p\left(u_t\right) du_t \tag{33}$$

Unfortunately, due to the unpredictable nature of control actions $a_{1:t-1}$ that produce robot displacements estimations $u_{1:t}$, the former integral computation could result impractical, because it would be necessary to calculate *a priori* the overall transition probability between two consecutive robot locations, regardless of the control actions issued to the robot.

4.4 Disadvantages of Viterbi Localization against Particle Filters

The Viterbi Localization (VL) presented in [13] which approximated $p(x_j\,|\,x_i)$ through a PDF by randomly simulating robot displacements discards one of the most important parameters involved in the pose estimation (especially when solving Position Tracking), the odometry displacement estimation u_t. Also, this lack of odometry data makes the localization method incapable of deciding between two consecutive locations with similar observations z_t located at the same distance to the previous location $\boldsymbol{x}_{t-1}$, something very common in long corridors. An alternative to the computation of $p(x_j\,|\,x_i)$ is including u_t both in the HMM and the VA (Fig. 3).

4.5 Viterbi Localization Mathematical Foundations

Trellis diagrams are interesting tools to understand the VA (Fig. 4), since they show in a graphical form how the forward VA procedure works. As the Fig. 4 depicts, in the induction phase of the forward procedure there is a summation that implies the state-transition probabilities a_{ij} that are part of the parameters of the HMM.

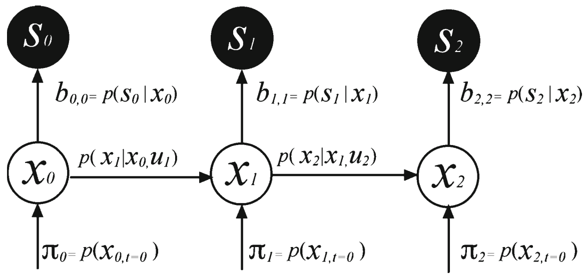

Fig. 3 Modified robot localization HMM. State transitions are control dependent, the rest of the model is a classical HMM

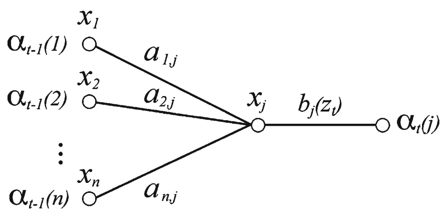

Fig. 4 Trellis diagram of Viterbi's forward procedure in HMM. State-transition probabilities a_{ij} are considered as fixed values

By using matrix notation and substituting Eq. 26 in Eq. 27, the forward phase of the VA can be rewritten as

$$p\left(z_{0:t}\right) = \Phi_{t-1}\mathbf{A}\mathbf{B}\nu_t^{\mathrm{T}} \tag{34}$$

A and **B** are the state-transition and observation matrices of the HMM respectively, while

$$\Phi_{t-1} = \left[\alpha_{t-1}(1) \quad \alpha_{t-1}(2) \quad \ldots \quad \alpha_{t-1}(n)\right] \tag{35}$$

is the previous belief vector and

$$\nu_t = \left[z_t \equiv s_1 \quad z_t \equiv s_2 \quad \ldots \quad z_t \equiv s_m\right]^{\mathrm{T}} \tag{36}$$

is a binary unitary vector with the exact correspondence of z_t with every symbol $[s_1 .. s_m] \in Z$. For instance, if $z_t = s_1$:

$$\nu_t = [1 \quad 0 \quad \ldots \quad 0]^{\mathrm{T}}$$

By analyzing the product $\Phi_{t-1}\mathbf{A}$ it can be noticed that

$$\begin{bmatrix} \alpha_{t-1}(1) \\ \alpha_{t-1}(2) \\ \cdots \\ \alpha_{t-1}(n) \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} = \begin{bmatrix} p\left(x_1, z_{0:t-1}\right) \\ p\left(x_2, z_{0:t-1}\right) \\ \cdots \\ p\left(x_n, z_{0:t-1}\right) \end{bmatrix}^{\mathrm{T}} \tag{37}$$

and similarly, the product $\mathbf{B}\nu^T$

$$\begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1m} \\ b_{21} & b_{22} & \ldots & b_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \ldots & b_{nm} \end{bmatrix} \begin{bmatrix} (z_k = s_0) \\ (z_k = s_1) \\ \cdots \\ (z_k = s_{m-1}) \end{bmatrix} = \begin{bmatrix} b_1(z_t) \\ b_2(z_t) \\ \cdots \\ b_n(z_t) \end{bmatrix} \tag{38}$$

Now, let us define a Dimensional Vector Product (DVP) as

$$\bar{u} \otimes \bar{v} = \left(u_1 v_1, u_2 v_2, \ldots, u_n v_n\right) \tag{39}$$

and a Dimensional Vector Contraction (DVC) as

$$\bar{v}^{\oplus} = \left(v_1 + v_2 + \ldots + v_n\right) \tag{40}$$

then, the dot product could be expressed as

$$\bar{u} \cdot \bar{v} = \left(\bar{u} \otimes \bar{v}\right)^{\oplus} \tag{41}$$

By applying $\Phi_{t-1}\mathbf{A}\otimes\mathbf{B}\nu^T$ (giving more precedence to matrix multiplications than the dimensional product), a new Φ_t is computed as

$$\Phi_t(i) = b_i\left(z_t\right) p\left(x_i, z_{0:t-1}\right) \tag{42}$$

As $b_i(z_t) = p(z_t \mid x_i)$, when multiplying $[\Phi_{t-1}\mathbf{A}]$ by $[\mathbf{B}\nu^T]$ in Eq. 34 results (dot product formula):

$$\sum_{i=1}^{n} p\left(z_t | x_i\right) p\left(x_i, z_{0:t-1}\right) = p\left(z_{0:t}\right). \tag{43}$$

It can be seen that Eq. 42 is included in Eq. 43, and

$$p\left(x_i | z_{0:t}\right) = \frac{p\left(x_i, z_{0:t}\right)}{p\left(z_{0:t}\right)} = \frac{\Phi_t(i)}{p\left(z_{0:t}\right)}. \tag{44}$$

The forward procedure of VA calculates the total probability $p(O|\lambda)$ by having $\{z_0, z_1,..., z_t\}$ as the observation sequence. After the initialization step when $\Phi_0(i) = \pi_i b_i(z_0)$, in each iteration $\Phi_t(i)$ computes

$$\Phi_0(i) = p\left(x_i, z_0\right), \Phi_1(i) = p\left(x_i, z_0, z_1\right), \ldots, \Phi_t(i) = p\left(x_i, z_{0:t}\right) \tag{45}$$

that is, the joint probability $p(x_i, z_{0:t})$. If summating all $\Phi_t(i)$, with $1 \le i \le n$, on every iteration then $p(z_{0:t})$ is obtained, corresponding to $p(O|\lambda)$.

If every $\Phi_t(i)$ in Φ_t obtained with Eq. 42 is divided by $p(z_{0:t})$ (by using Eq. 44) before the next iteration, then

$$\Phi'_0(i) = p\left(x_i | z_0\right), \Phi'_1(i) = p\left(x_i | z_0, z_1\right), \ldots, \Phi'_t(i) = p\left(x_i | z_{0:t}\right) \tag{46}$$

$$\frac{\Phi_t(i)}{\sum_{i=1}^{n}\Phi_t(i)} = \Phi'_t(i) = Bel_t(i) \tag{47}$$

As shown above, Viterbi's forward procedure can be easily adapted for carrying on a pose belief.

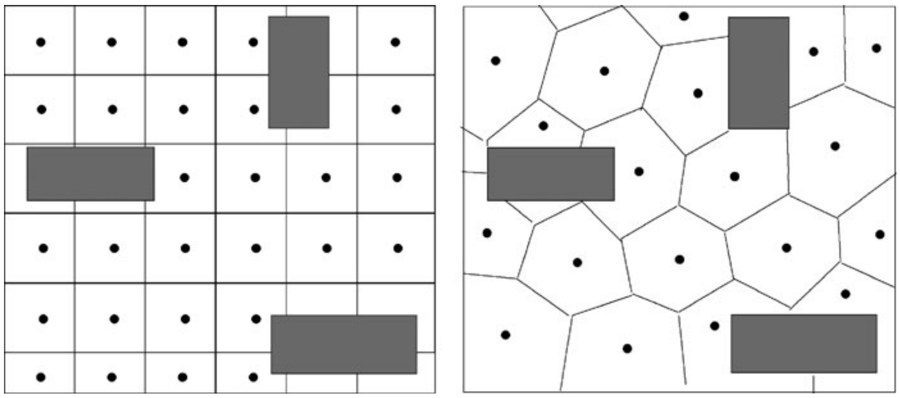
Fig. 5 Space–state partitions from [33]: Occupancy Grid OG (*left*) and Voronoi Diagram VD (*right*). VQNs can be used for transforming OGs into VDs

5 Motion Model Implementation

Based on a scan-matching algorithm as [14] or in the robot self-odometry, it is possible to build a Odometry-Dependent Motion Model (ODMM) with differential movement estimations u_t, plus the previous pose beliefs, in order to compute a complete motion estimation as Particle Filters do.

5.1 Partitioning the State–Space

The Viterbi Algorithm operates in a discrete state–space. For this reason it is necessary to partition the working environment —the Configuration Space (CS)— into discrete regions. The CS is usually represented with occupancy grids (Fig. 5, left) or metric maps, then the partitioning can be easily done [31, 32]. In this work a Vector Quantization Network (VQN) was used because it provides locations uniformly distributed, similar to Voronoi Diagrams (Fig. 5, right).

From the CS representation, a random sampling $L{:}(x_i, y_i, \theta_i)$ of free locations and headings can be obtained (Fig. 6, left). By using a VQN fed with these vectors, after a few number of training epochs, the found vectors will correspond to the states of the HMM (Fig. 6, right).

The advantage of this method is that the final number of states (nodes) can be decided in advance and the final locations will be uniformly-distributed.

5.2 Vector Quantization Considerations

Although Vector Quantization (VQ) can be performed in the three dimensions of the state–space (x, y and θ), more convenient (and faster) results are obtained by performing only a 2D state partitioning (x and y dimensions) and the final set of 3D states can be built by sharing every xy location of the 2D nodes with a group of absolute headings like [0, 0.78, 1.57,..., 5.49], where every value stands for an absolute heading angle respect to the global coordinates frame expressed in radians (Fig. 7). By dividing 2Π by a fixed number of

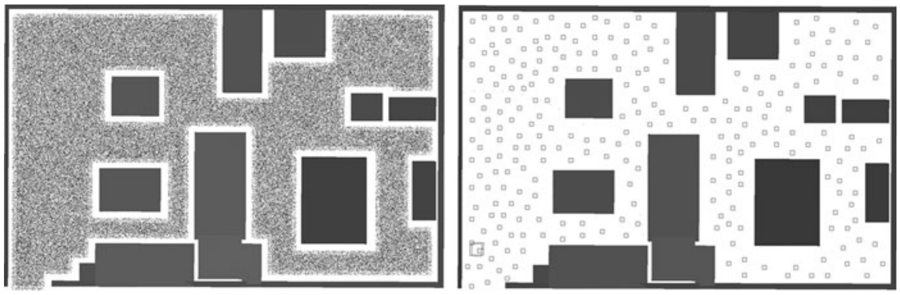
Fig. 6 *Left*. 10,000 random pose samples in the free space for quantizing locations. *Right*. VQN with 256 neurons (states) trained with the random sample set, by using 100 training epochs

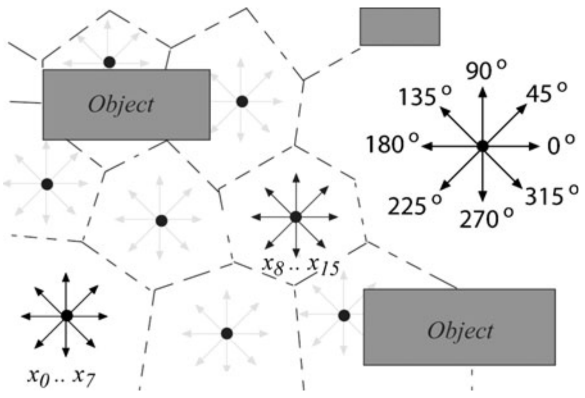

Fig. 7 State absolute heading composition. *Arrows* indicate the node's heading

steps (eight in the former example) consecutive absolute headings can be defined from [0..2Π).

In this way, if a 2D node location is (3.2, 5.4) it is possible to set a group of eight nodes (states) with absolute locations (3.2, 5.4, 0), (3.2, 5.4, 0.78),..., (3.2, 5.4, 5.49), and so on. This discretization reduces the memory needed for storing the node's location as it is assumed that for a given *xy* obstacle-free location the robot is able to be oriented in any direction. Otherwise, if VQ is performed over the 3 dimensions of the state–space, as θ is expressed in radians going either from [$-\pi..\pi$] or [$0..2\pi$], the VQN will select during training the same neuron *i* as the closest for a small neighborhood of (x_i, y_i) locations, regardless of θ_i (due to its smaller value compared with *x* and *y*). As a result, all found discretized nodes will have the same heading: 0 radians for θ_i in [$\pi..\pi$] or π radians for θ_i in [$0..2\pi$], because the VQN will compute the average θ_i. In this case all the HMM states would have the same absolute heading and it would be impossible to estimate the true robot's heading with that set of states if the robot is oriented in a different direction than the one all the states have.

5.3 Transition Vectors

If the node (state) locations in absolute world coordinates $x_i : (x_{w_i}, y_{w_i}, \theta_{w_i})$ are considered as fixed, the Motion Model has to estimate the overall transition probability, given the odometry estimation u_t (or the a_t command in systems without odometers), this is $p(x_j \mid x_i, u_t)$, where u_t is:

$$u_t : (d_t, \phi_t, \Delta\theta_t) \tag{48}$$

where d_t is the distance between consecutive robot locations $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$, ϕ_t is the direction of the displacement (relative to the robot's front), and $\Delta\theta_t$ is the absolute heading change (Fig. 8, upper-left).

For each state x_i it is possible to transform the odometry estimation u_t (given in a robot centered coordinate system) into an absolute displacement $u_t^i = \left(u_{t_x}^i, u_{t_x}^i, u_{t_\theta}^i\right)$ relative to each node absolute location $\boldsymbol{x}_i$ with

$$u_{t_x}^i = d_t \cos\left(\theta_{w_i} + \phi_t\right) \tag{49}$$

$$u_{t_y}^i = d_t \sin\left(\theta_{w_i} + \phi_t\right) \tag{50}$$

$$u_{t_\theta}^i = \Delta\theta_t \tag{51}$$

Once the absolute displacement has been calculated, it can be added to $\boldsymbol{x}_i$, computing a new absolute location

$$x'_i = x_i + u_t^i \tag{52}$$

The closer $\boldsymbol{x}'_i$ gets to every pose $\boldsymbol{x}_j$, the larger the probability of a true transition between x_i and x_j will exist.

One important issue is that, based on the connectivity network between nodes, not all transitions $p(x_j \mid x_i, u_t)$ would be valid, due to the existence of obstacles in the environment. In our case, all transitions are considered valid because under continuous movement conditions it is perfectly possible for a mobile robot to avoid an object by circumnavigating its contours. If location $\boldsymbol{x}_i$ is

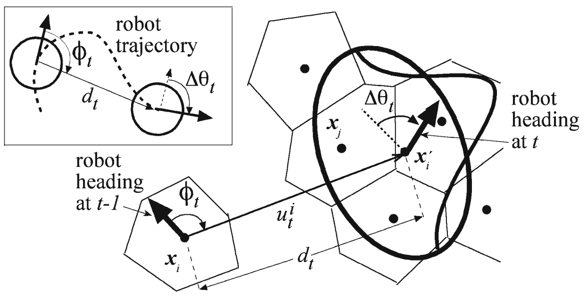

Fig. 8 Actual robot movement (*above left*) and Motion Model parameters

computed at time $t-1$ (before avoiding the obstacle) and the robot pose reaches $\boldsymbol{x}_j$ at time t (when the robot has surpassed the obstacle), u_t will reflect this change of pose, corresponding to a perfectly valid displacement.

5.4 Estimating Transition Probabilities

As $\boldsymbol{x}_i$ is an absolute state location and u_t is the robot-centered displacement estimation, the transition probability of reaching $\boldsymbol{x}_j$ starting from $\boldsymbol{x}_i$ will depend on how $\boldsymbol{x}\prime_i$ approximates to $\boldsymbol{x}_j$, in a global coordinates frame, this is:

$$p\left(x_j | x_i, u_t\right) = \frac{D\left(x_i', x_j\right)}{\sum_{j=1}^{n} D\left(x_i', x_j\right)} \tag{53}$$

where D is a distance-based PDF that can be a Gaussian function centered at $\boldsymbol{x}\prime_i$ (Fig. 6, right):

$$D\left(x_i, x_j\right) = e^{-\left(\frac{\left(x_{w_j} - x_{w_i}\right)^2}{2\sigma_d^2} + \frac{\left(y_{w_j} - y_{w_i}\right)^2}{2\sigma_d^2} + \frac{\left(\theta_{w_j} - \theta_{w_i}\right)^2}{2\sigma_\theta^2}\right)} \tag{54}$$

where σ_d and σ_θ can be set depending on the xy node's spatial separation to become more or less inclusive (the w sub-index stands for 'in world coordinates').

In this way, a new transition matrix $\mathbf{A}$ can be computed in each iteration, based on the u_t estimation by assigning

$$a_{ij} = p\left(x_j | x_i, u_t\right) \tag{55}$$

calculated with Eq. 53. As Fig. 8 shows, $p(x_j \,|\, x_i, u_t)$ depends on how probable is that u_t takes $\boldsymbol{x}_i$ exactly into $\boldsymbol{x}_j$.

5.5 Complexity

Derived from Eq. 53 it can be seen that the computation of $\mathbf{A}$ matrix is $O(n^2)$(where n is the number of nodes (states) in the configuration space) because it implies calculating a Euclidean distance between $\boldsymbol{x}_i'$ and every $\boldsymbol{x}_j$ given u_t, $\forall x_i$, $x_j \in X$. Based on this premise, the total number of required operations for the computation of $\mathbf{A}$ in each iteration could make this method incapable to run in real-time, considering that there will be only a small number of significant transitions for every node (those inside the influence ratio of D function) and the rest of transition probabilities would be almost zero. These extra calculations can be avoided if Eq. 53 is computed only for those states into the neighborhood of $\boldsymbol{x}_i'$, having numerical significance for D, the remaining transitions can be assumed with probability near to zero.

6 Observation Model Construction

6.1 Random Sampling

Once a space partitioning has been computed, it is necessary to relate observations with locations for building the Observation Model (OM). This implies obtaining a Probability Density Function (PDF) of observations per discrete state. For this purpose, a set of observations (raw sensor vectors) must be obtained from random locations. The goal is to have a uniformly distributed observation sampling to relate enough observations with every discrete robot pose (state) of the model without introducing observation tendencies.

The main objective of this sampling is to have enough observations per state in order to build a complete observation PDF. If some observation probable to occur at a given state is not contemplated, the algorithm would fail because the OM does not relate that state-observation. In the next sections a Tolerant Observation Model (TOM) will be presented for dealing with such cases.

It is convenient using the same set of random locations generated in Section 5.1 for obtaining the 'supposed' observation at each random location (Fig. 6), thus using a single set of random locations with their corresponding simulated observation for building the OM. While more similar the artificially generated observation to the actual sensor data, more precise the observation model will become. For this reason it will be important to use a method which provides as much realistic observations as possible.

6.2 Simulating Observations vs. Real Gathering

The best way to do this is by artificially generating those observations with a previously built World Model of the environment (metric map) and a software simulator with ray tracing (Fig. 9).

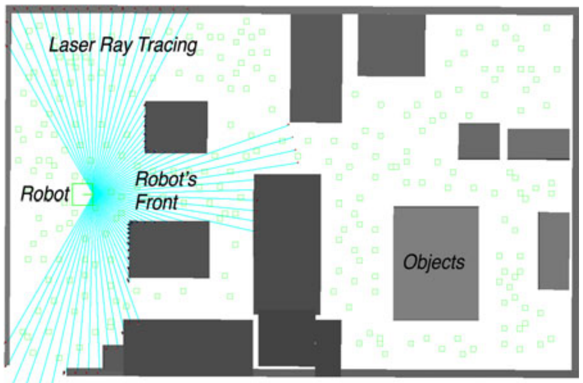

Fig. 9 Metric map and simulated range scan (*cyan*)

The use of a metrical representation avoids the need of collecting as many samples as needed with the real robot for almost every navigable pose (the whole Configuration Space), something clearly impractical in environments as those where service robots must operate (Fig. 10).

By the other hand, if a real robot is used for collecting observations inside the real environment, it is almost impossible to ensure that the OM will have enough observations for all navigable locations and headings (states), otherwise the Observation Model will be incomplete. The only case where using real samples for building the OM is convenient is in such cases where the mobile robot is always displacing along the same trajectories over a track or railroad.

6.3 Building a Metrical Representation

Although it is possible to use a rough polygonal representation by manually measuring the objects in the environment (Fig. 11, left), more accurate environment representations and simulations, as those obtained by generating a map of the environment using the actual robot sensors (Fig. 11, right), give better results because the accuracy of the simulated observations is greater.

Both types of Metrical Maps were used in the present work for evaluating the proposed approach. A metrical representation of a working environment scanned with the actual robot sensors (a laser range scanner) was built by using a Clustering Artificial Neural Network (CANN) [34] with 10000 neurons (Fig. 12a). The network was fed with the (x,y) coordinates corresponding to projecting laser range scans as points into the xy plane. The CANN generates a metrical representation by placing clusters of a fixed size (10 cm $\times$ 10 cm) grouping the xy points (Fig. 12b). After 300 training epochs those neurons that were not updated (i.e. there are far away from any xy point) are removed (Fig. 12c) and a metrical representation is obtained (Fig. 12d). Each cluster represents an object.

Once the detailed metric map is built with the CANN, the difference between actual observations (Fig. 13, center) and those simulated is notorious for the case of the rough map (Fig. 13, left) and the detailed map (Fig. 13, right).

As it can be seen in Fig. 13, a detailed map will bring observations more similar to 'real' ones. In the above example, Gaussian noise has been added to the ray-tracing so the resulting scan contour is not as sharp as Fig. 13 left.

One important fact is that the small details (i.e. the high frequencies) corresponding with variations in the scans due to the presence of small objects, given the nature of the Observation Model

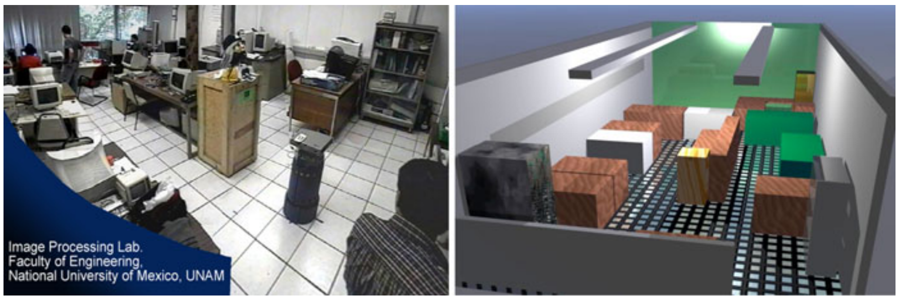

Fig. 10 Typical environment for a service robot. (*Left*) Real Env. (*Right*) Rough 3D Model

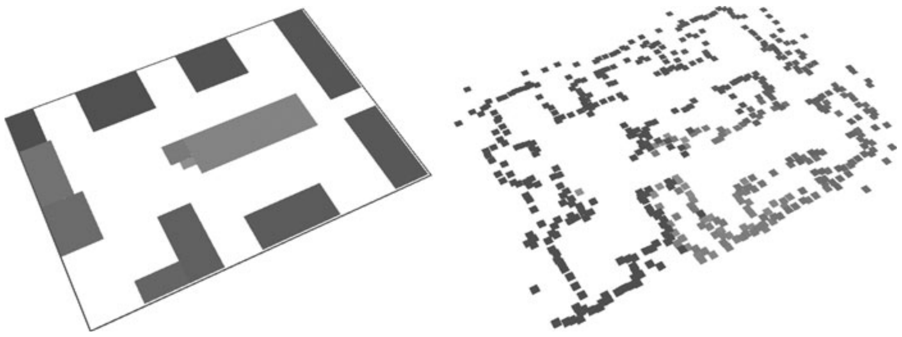

Fig. 11 Rough polygonal map (*left*) and real-scanned (*right*) maps of the environment of Fig. 10

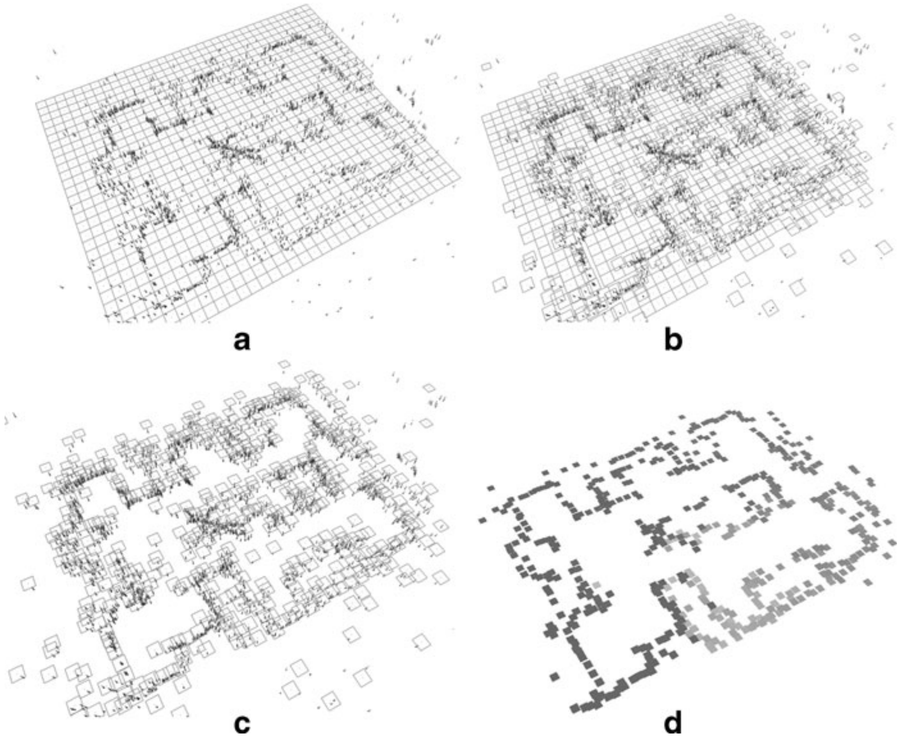

Fig. 12 Clustering ANN. **a** Initialization. **b** Training. **c** Pruning. **d** Final Map

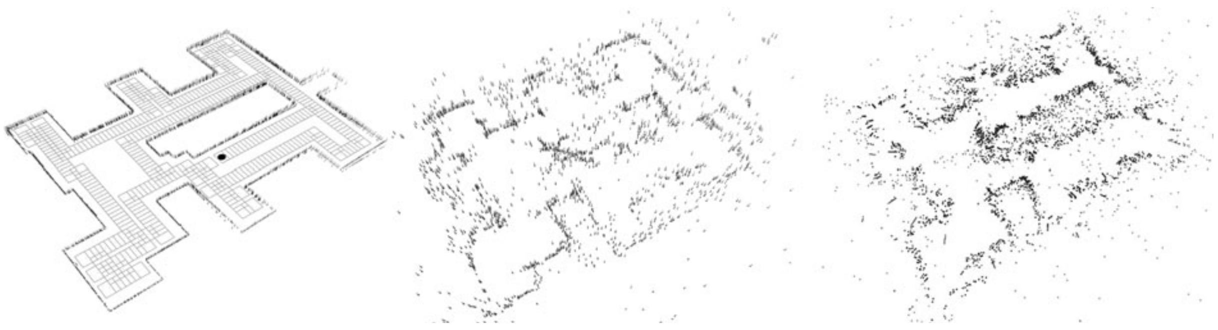

Fig. 13 Ray-tracing. (*Left*) Rough map. (*Right*) Detailed map. (*Center*) Real observations

(OM) proposed in the next sections will have less importance in the OM than big changes. For this reason both type of metrical maps (rough and detailed) will be perfectly suitable for generating the OM. This issue will be discussed in the experiments section.

6.4 Simulating the Observations

The advantage of artificially generating the observations is that obviously is possible to generate a raw sensor vector (so called the 'supposed' observation) for any robot pose. Unfortunately, the 'real' sensor data (the observation made with the real robot at the given pose) could vary respect to that 'supposed' one, due to sensor noise and reflections (Fig. 14). For this reason is very important adding noise to the generated observations, as proposed in [11].

Also, when simulating observations it is very important considering more random locations than states in the model (10 times or more), because each discrete state will comprise a region of poses and headings surrounding it. The Observation Model (OM) must consider as many different observations as possible in every region in order to relate states with real observations (environment dependency).

6.5 Lowering the Dimensionality

The next step consists in lowering the dimensionality of the artificially generated observation vectors, in order to simplify the OM construction. The idea is to build the observation–location relationship from quantized observations and quantized locations (states) instead of using the raw vectors.

One important issue is that by quantizing the observations, a type vector index will substitute the whole multidimensional sensor observation, so every range scan will be labeled with a scalar (or a couple of scalars if a 2D VQ map is used). In the case of Robot Localization this could imply gathering information from multiple scenarios before performing the VQ. Nevertheless, in this work we propose performing VQ exclusively using simulated observations for every evaluated scenario (this implies repeating the VQ for a new scenario), in order to avoid having type vectors whose probability of occur will be always practically zero thus saving memory and processing time. In further works we will perform VQ with simulated scans coming from many scenarios, comparing the results.

6.5.1 Vector Quantization ANNs

If a standard VQN [34] is used to obtain a set of observation vectors (to compute the Observation Model), the result will be similar to the one shown in Fig. 15.

It can be seen that there is no similarity between consecutive vectors in the 2D VQN array, both in columns and rows (r and c indexes). Although this quantization of observations will solve the task, in case of having occlusions or large reading noise (like the one presented by black polished objects for laser scanners or soft materials in the case of sonar range finders), two found type vectors ν_i:(r_i, c_i) and ν_j:(r_j, c_j) (with and without noise respectively, see Fig. 15) would

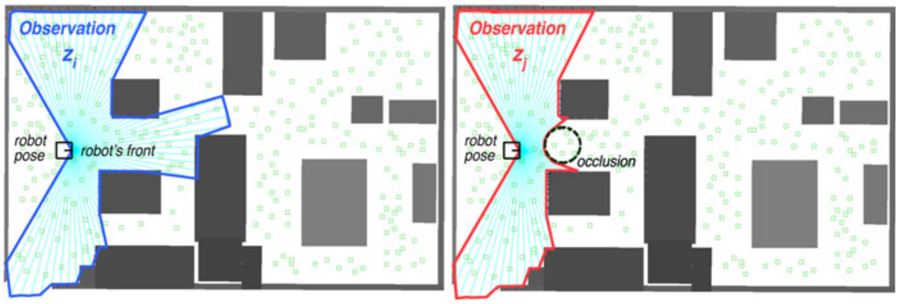

Fig. 14 Location of nodes (*green squares*), polygonal objects (*gray*) and 68-readings range scanning simulation (*cyan lines*), from -135 to 135°. Robot is facing 0°. The convex hulls of supposed (*left*) and occluded (*right*) observations are shown

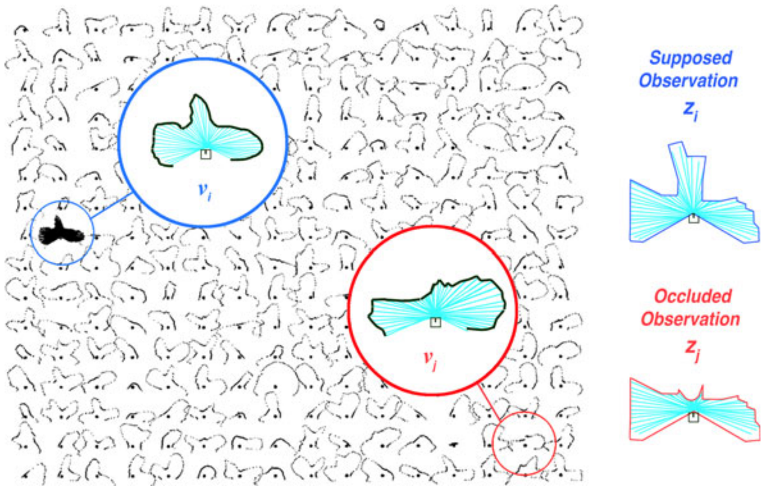

Fig. 15 256-units standard VQN without neighborhood relationships. The network was trained with the simulated observations at the random locations depicted in Fig. 6 left. Also the corresponding vector quantization of observations in Fig. 14 are shown: *blue*: supposed, *red*: occluded. Each of the small figures represents a laser reading. The robot is considered facing 90°

become very distant in the 2D VQ array (Fig. 8). As $p(z_t = s_k | x_i)$ must be calculated for building the OM, the distance $|r_j - r_i, c_{j-}c_i|$ between real and noisy observations in the VQ array would complicate the process of establishing an *a priori* observation probability for a given location x_i, because the actual observed symbol indexes (r_s,c_s), measured with real sensors, could vary substantially in the presence of sensor noise or occlusions respect to the supposed observations stored in the OM.

This distance between noisy and noise-free observations in the VQ array can be lowered with the help of a *neighborhood* relationship, facilitating the process of including noise and occlusions tolerance in the OM.

6.5.2 Vector Quantization with Self Organized Maps

Self Organized Maps (SOMs) [34] are a special kind of VQNs where a neighborhood relationship is defined. Like in a standard VQN, in a SOM the neuron whose weights are the closest to an input pattern is selected, but not only its weight is approached to the input vector during training, also are the weights of the neighbors in a lower proportion, according to

$$\bar{w}_{k(t+1)} = \bar{w}_{k(t)} + \gamma \cdot \left(\bar{x}_{(t)} - \bar{w}_{k(t)}\right) \cdot h\left(i - g\right) \tag{56}$$

where $h(i-g)$ is a neighborhood update function in the SOM array, calculated based on the index separation $d = i - g$, as

$$h\left(d\right) = e^{\frac{-d^2}{\sigma^2_{SOM}}} \tag{57}$$

In this way, SOMs can organize the observations space in a more correlated form. When training a 16 × 16 units SOM with the samples described in Section 6.1, similar type vectors will be closer in the map (Fig. 16).

6.6 Building a Tolerant Observation Model

Once the SOM has been built, the next step will be identifying the Observation Model, by simulating the corresponding observations for locations x_i.

Every simulated sensor reading $z_i^s = S\left(x_i\right)$ is fed to the VQN V and a corresponding symbol $s_i^s = V\left(z_i^s\right)$ is obtained as the closest SOM vector, then a probability of observation can be established, based on a neighborhood relationship as

$$G\left(s_k, s_i\right) = e^{\frac{-\left|\gamma\left(s_i^s\right)-\gamma\left(s_k\right)\right|^2}{2\sigma^2_{tol}}} \tag{58}$$

where $\gamma\left(s_i^s\right)$ and $\gamma\left(s_k\right)$ calculate the 2D indexes $\left(i_i^s, j_i^s\right)$ and $\left(i_k, j_k\right)$ respectively into the SOM, corresponding to symbols s_i^s and s_k (Fig. 17), while σ_{tol} (or the 'tolerance') can be set in experimental form, depending on the probability of having

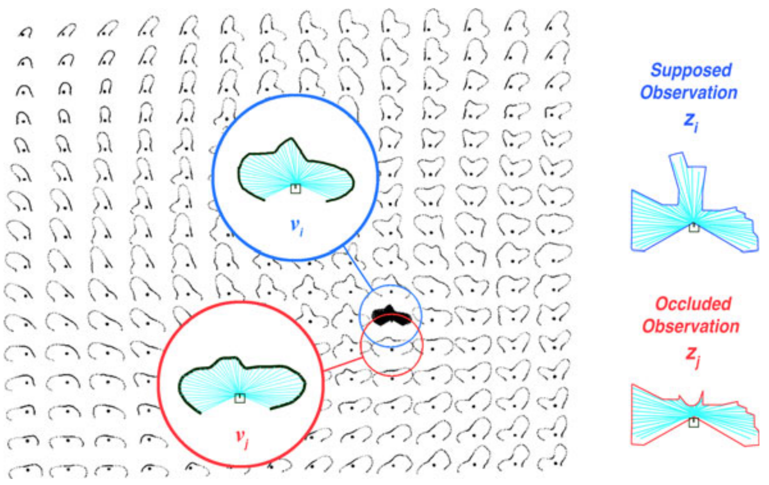

Fig. 16 16-by-16 Kohonen SOM trained with the simulated observations at the random locations depicted in Fig. 6 left. The corresponding vector quantization of observations in Fig. 14 are shown: *blue*: supposed, *red*: occluded. Occluded observation is closer to supposed in the SOM

occlusions and sensor errors (as the case of populated environments). This parameter gives the OM the property of 'tolerating' some observation symbol variations, mainly produced by occlusions and sensor reflections. The formula

$$p\left(s_k|x_i\right) = \frac{G\left(s_k, V\left(S\left(x_i\right)\right)\right)}{\sum\limits_{n=1}^{m} G\left(s_n, V\left(S\left(x_i\right)\right)\right)} \tag{59}$$

calculates the Tolerant Observation Model probabilities, with S as a relationship between location and observations (a 'simulator' or 'sampler') and V gives the closest SOM vector s_i^s to the supposed observation z_i^s, given by the sampler.

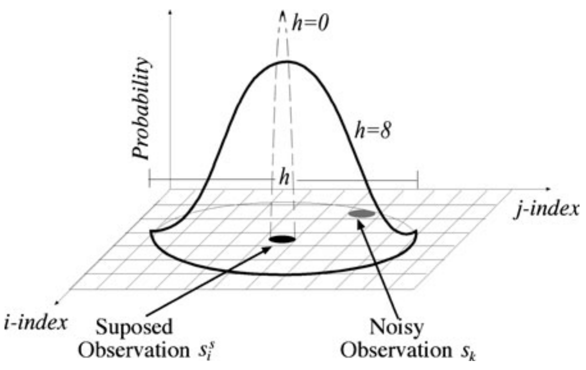

Fig. 17 Observation PDF based on SOM neighborhood. Noisy observation indexes closer to supposed observation will be more probable to occur, due to occlusions and reflections

The normalization in Eq. 59 avoids the need of an extra k parameter to ensure its integral equals to 1. Finally,

$$b_{jk} = p\left(s_k|x_j\right) \tag{60}$$

can be calculated with Eq. 60 for building the observation probability matrix **B** in order to

```
------------------------------- initialization -----------------------------
1- Get the set of randomly sampled observations $\Omega$: $(x_i, z_i)$
2- Set all $\mathbf{B}(i, j)$ to zero
3- For i=1 to number of nodes
    3.1 Set accumulator $g(i)$ to zero
------------------------------- quantization ------------------------------
4- For every sample $\omega_k \in \Omega$ do
    4.1- Get the corresponding state index $n_k$ by quantizing $x_k$
    4.2- Get the corresponding symbol $s_k$ by quantizing $z_k$
--------------------------------- valuation --------------------------------
    4.3- For all $s_i$ in the neighborhood of $s_k$ of size $h$
        4.3.1- Calculate $p_{ki} = G(s_k, s_i)$ with Eq. (59)
        4.3.2- Increment index $\mathbf{B}(n_k, s_i)$ with $p_{ki}$
        4.3.3- Increment accumulated sum $g(n_k)$ with $p_{ki}$
------------------------------- normalization ----------------------------
5- For i = 1 to the number of states
    5.1 For j = 1 to the number of symbols
        5.1.1 Set $\mathbf{B}(i, j) = \mathbf{B}(i, j) / g(i)$
```

Fig. 18 Tolerant Observation Model construction pseudocode

initialize the Tolerant Observation Model (TOM). By having a neighborhood of size $h = 0$ (only the probability for the current symbol s_i^s is updated) a non-tolerant observation model is obtained (Fig. 17).

Figure 18 shows the final Tolerant Observation Model construction pseudocode.

7 Path Reconstruction

As we are using a probabilistic method for pose and each node has a probability $Bel_t\,(i) = p(x_i \,|\, z_{0:t}, u_{1:t})$ assigned to it after the update phase, then, by using the expected value formula, a continuous robot location $\boldsymbol{x}_t$ can be computed from discrete samples (nodes) with

$$x_t = \sum_{i=1}^{n} \left[x_i\, p\,(x_i | z_{0:t}, u_{0:t}) \right] = \sum_{i=1}^{n} \left[x_i\; \Phi'\,(i) \right] \qquad (61)$$

after each iteration, being possible to build a path with every successive location calculated with Eq. 61.

8 Optimizations

The motion estimation proposed in this work has the time complexity of $O(n^2)$, with n as the number of discrete states in the model, because in each iteration the transition probabilities between every state x_i and the remaining states x_j must be calculated (total probability). Some assumptions can be done in order to reduce drastically the number of required operations without loosing generality and precision (this is especially useful when running the algorithm in standard non-parallel processors):

8.1 Node Location Selection

Because transitions between nodes $x_i : (x_{w_i}, y_{w_i}, \theta_{w_i})$ and $x_j : (x_{w_j}, y_{w_j}, \theta_{w_j})$ must be calculated in each iteration, it results much simpler partitioning the free space with a 2D vector quantization network in order to get the node's xy location. Once calculated, a group of nodes can share the same (x_{w_i}, y_{w_i}), only differing on their θ_{w_i} coordinates, starting from zero and with a heading at predefined regular intervals (see Section 6.1). This can help to use one single distance estimation $(x_{w_j} - x_{w_i})$ and $(y_{w_j} - y_{w_i})$ in several transition updates.

8.2 Update Triggering

The belief update is calculated with $\Phi_{t-1}\mathbf{A}\otimes\mathbf{B}\nu^T$, but in general only a reduced number of pose beliefs in the Φ_{t-1} vector will have a significant value. In this form, all the transition and observation calculations can be performed only for those $\Phi_{t-1}(i) \geq \varepsilon$. If nested loops are used for calculating the a_{ij} transitions, such transitions can be considered as almost zero if $\Phi_{t-1}(i) < \varepsilon$ (or a minimum probability value near to zero, in order to not discard any state transition in future computations). This optimization has the important role of selecting which states (and state-transitions) must be evaluated during the pose belief update, acting as a trigger of GL after a RK as follows: if the robot is performing PT (the current robot pose is known) some $\Phi_0\,(i)$ will tend to one while the rest will tend to zero, due to the normalization after the observation update in Eq. 47. After a RK, $\Phi_t(i)$ will rapidly tend to a uniformly distributed probability value (due to the difference between real and supposed observations (as the robot is not informed about the displacement) equals to 1.0/*number_of_states*. If ε is set just below this uniformly distributed probability value then the method will trigger all the state-transitions evaluation after the RK (when $\Phi_t(i)$ reaches the uniform probability).

As soon as some $\Phi_t(i)$ increase their probability, the rest will continue lowering their values below ε, discarding them from future computations. This will be equivalent to performing GL with an unknown initial pose.

In a similar way, if the initial robot pose is unknown or a GL command is issued to the robot, it will be enough to set every $\Phi_0(i)$ equals to 1.0/*number_of_states*, forcing the method to reevaluate all state transitions (the operation of the update triggering will be demonstrated in the results section).

8.3 Observation Update Nesting

The main Viterbi update is calculated through Eq. 26 that is equivalent to

$$Bel\,(j)_t = \sum_{i=1}^{n} \left[p\left(x_j | x_i, u_t\right) Bel\,(i)_t\, b_j\,(z_t) \right] \qquad (62)$$

By nesting the observation update into the transition update, only a couple of nested loops can be used to compute the belief update. Then, they can be triggered together by ε (Fig. 19).

This observation update nesting allows using a single vector array $\mathbf{A}(j)$ in each iteration, with $1 \le j \le$ *number of states*, instead of a whole transition square matrix $\mathbf{A}(i, j)$. The same loop (Fig. 19, step 1) is being used both for the calculation of the transition probabilities for the current state i to every state j and for the calculation of Eq. 62 so there is no need to store the transition probability values for previous states i, thus saving much memory space. According to this, it is possible to substitute the $\mathbf{A}(i, j)$ term in steps 1.3.1, 1.3.2 and 1.4 by $\mathbf{A}(j)$.

8.4 Memory Usage

With all the abovementioned optimizations, if *nodes* is the number of (x_i, y_i) quantized locations (see 8.1), then *nodes*×2 is the required memory for storing them. If *headings* is the number of absolute state headings (see 5.2), then *states* = *nodes* × *headings* is the number of states in the OVL and *states*×2 is the memory required for storing Φ_t and Φ_{t-1}. If *symbols* is the number of neurons in the SOM (see 6.5.2) and *dimobs* is the dimensionality of the raw observation vector z_t, then *symbols*×*dimobs* is the memory necessary to store the SOM weights, *states* is the memory required to store the $\mathbf{A}$ vector and *states* × *symbols* is the memory required for storing the $\mathbf{B}$ matrix. Then the total memory usage will be:

$$\begin{aligned} \text{Memory} = {} & \text{nodes} \times \left(2 + \text{headings} \times \left(\text{symbols} + 3\right)\right) \\ & + \text{symbols} \times \text{dimobs} \end{aligned} \qquad (63)$$

```
1. For i = 1 to number of states
    If $\Phi_{t-1}(i) \ge \varepsilon$ then
        1.1 Set $\alpha_{acc} = 0$
        1.2 Calculate $\mathbf{x}'_i$ with Eq. (49)-(52)
        1.3 For j = 1 to number of states
            1.3.1 Calculate $\mathbf{A}(i, j) = D(\mathbf{x}'_i, \mathbf{x}'_j)$ with Eq. (54)
            1.3.2 Increment $\alpha_{acc}$ with $\mathbf{A}(i, j)$
        1.4 For j = 1 to number of states
            1.4.1 Increment $\Phi_t(j)$ with $\mathbf{B}(j, s_t)\ \Phi_{t-1}(i)\ \mathbf{A}(i, j) / \alpha_{acc}$
```

Fig. 19 Observation update nested in the transition evaluation cycle (pseudocode)

9 Odometry-based Viterbi Localization Algorithm

In this section we present the final Odometry-based Viterbi Localization (OVL) Algorithm, shown in Fig. 20.

```
----------------------------------------- initialization -------------------------------------
1.- Set $\xi = 1 \times 10^{-10}$, $\varepsilon = 1/(number\_of\_states) - \xi$, $\alpha_{acc} = 0$,
    $\sigma_d = 2/3 \times avg\_node\_separation$, $\sigma_\theta = \Pi/(2 \times num\_of\_absolute\_headings)$
3- For i = 1 to number_of_states
    3.1- If Initial Position $\mathbf{x}_0$ is known (Position Tracking)
      3.1.1- Set $\Phi_0(i) = \mathbf{D}(\mathbf{x}_0, \mathbf{x}_i)\ \mathbf{B}(i, s_0) + \xi$ with Eq. (54)
    3.2- else $\Phi_0(i) = \varepsilon\,\mathbf{B}(i, s_0) + \xi$ (Global Localization)
    3.3- Increment $\alpha_{acc}$ with $\Phi_0(i)$
4- For i = 1 to number_of_states Set $\Phi_0(i) = \Phi_0(i) / \alpha_{acc}$
------------------------------------------ recursion -----------------------------------------
5- For t =1 to the number of observations
    5.1- Get robot odometer estimation $u_t : (d_t, \phi_t, \Delta\theta_t)$
    5.2- Get current symbol $s_t$ by quantizing $z_t$
    5.3- For i = 1 to number of states, Set $\Phi_t(i) = \xi$
-------------------------------------- A matrix calculation ---------------------------------
    5.4- For i = 1 to number of states
      If $\Phi_{t-1}(i) \ge \varepsilon$ then
        5.4.1- Set $\alpha_{acc} = 0$
        5.4.2- Calculate $\mathbf{x}'_i$ with Eq. (49)-(52)
        5.4.3- For j = 1 to number of states (or those in the neighborhood of $\mathbf{x}'_i$)
            5.4.3.1- Calculate $\mathbf{A}(j) = D(\mathbf{x}'_i, \mathbf{x}'_j)$ with Eq. (54)
            5.4.3.2- Increment $\alpha_{acc}$ with $\mathbf{A}(j)$
        5.4.4- For j = 1 to number of states
            Increment $\Phi_t(j)$ with $\mathbf{B}(j, s_t)\ \Phi_{t-1}(i)\ \mathbf{A}(j) / \alpha_{acc}$
------------------------------------------ normalization ---------------------------------------
    5.5- Set $\alpha_{acc} = 0$
    5.6- For i = 1 to number of states, Increment $\alpha_{acc}$ with $\Phi_t(i)$
    5.7- For i = 1 to number of states, Set $\Phi_t(i) = \Phi_t(i) / \alpha_{acc}$
----------------------------------------- pose estimation -----------------------------------
    5.8- Set $\mathbf{x}_t = (0,0,0)$
    5.9- For i = 1 to number of states, Increment $\mathbf{x}_t$ with $\mathbf{x}_t\ \Phi_t(i)$
    5.10- Consider $\mathbf{x}_t$ as the current robot pose
```

Fig. 20 Optimized OVL pseudocode for non-parallel microprocessors

9.1 Initialization

If the initial pose (x_0, y_0, θ_0) is known, Eq. 54 can be used to evaluate the initial pose probabilities Π_i (as it evaluates the probability for any x' to correspond with every state x_i):

$$\Pi_i = D\left(x_0, x_i\right) \tag{64}$$

By the other hand, if the initial pose is unknown, Π_i can be set to a uniformly distributed PDF, this is

$$\Pi_i = 1/number\_of\_states. \tag{65}$$

forcing the method to evaluate all possible state transitions depending on u_1.

9.2 Recursion

9.2.1 Transition Probability Estimation

This section is the core of the Odometry-Based Viterbi Algorithm. First, steps 5.1 and 5.2 obtain the odometer estimation u_t and compute the current symbol s_t by quantizing the current observation z_t. Step 5.4.1 cleans the accumulator α_{acc} that serves to store the sum of state-transition probabilities. Step 5.4.2 calculates the new location x_i' given the state location x_i and u_t. Step 5.4.3 computes the corresponding state-transition probability $p(x_j|x_i, u_t)$ and stores that value in the temporal accumulator $\mathbf{A}(j)$.

Step 5.4.4 increments the state probability $\Phi_t(j)$ (already initialized with a value close to zero in step 5.3) with the probability of observation $p(z_t|x_j)$ multiplied by the previous state probability $\Phi_{t-1}(j)$, the state-transition probability $p(x_j|x_i, u_t)$ already stored as $\mathbf{A}(j)$ and divided by the probability sum α_{acc}. In this way, in a single loop, every state probability $\Phi_t(j)$ is incremented with the individual contribution of a transition to x_i from every x_j and normalized by α_{acc}.

Below step 5.4, by using a trigger value ε, only the overall contribution to $\Phi_t(j)$ of those states whose probability in the previous iteration $\Phi_{t-1}(j)$ was above the trigger value will be evaluated (update triggering). This important feature allows optimized-OVL to save time, memory and managing GL, PT and RK (this will be discussed in the results section).

9.2.2 Normalization

Steps 5.5 to 5.7 compute Eq. 27 to obtain p$(x_i\,|z_{0:t})$. This normalization ensures

$$\sum\nolimits_{i=1}^{n} p\left(x_i|z_{0:t}\right) = 1$$

9.2.3 Pose Estimation

Steps 5.8 to 5.10 compute the current robot pose with Eq. 61. Although this equation calculates only one localization hypothesis based on the expected value formula, OVL is always carrying-on a multi-hypothesis PDF through the state probability vector Φ_t. Other methods that can be used for estimating multiple hypotheses are Clustering Artificial Neural Networks (CANNs) [20] and Radial Basis Functions (RBFs) [21]. By isolating a group of states with high-probability as a cluster, they can represent multiple localization hypotheses. When having only one cluster it would represent the true robot pose.

The advantage of using the expected value formula in Eq. 62 is that it can be also used for computing the average pose value for a group of states (those in the influence ratio of every found cluster).

9.3 Termination

As a difference with classical Viterbi Localization [13], after every update the robot pose is calculated with Eq. 62 so there is no need of backtracking (i.e. searching for the most probable individual state after every observation update). The proposed approach in fact calculates a continuous pose after every observation update, this is an important improvement against [13].

10 Experiments

10.1 Physical Robot

The robot ARTUR-ito (Autonomous Ready-To-Use Robot, Fig. 21) is a differential drive robot equipped with two computers: an Apple Mac

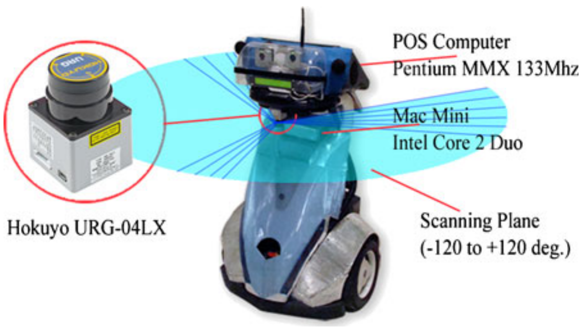

Fig. 21 Robot ARTUR-ito

Mini® with an Intel Core 2 Duo processor at 2.4 Ghz running Mac OS X Leopard, and a Javelin Wedge® Point-Of-Sales (POS) touchscreen computer with a Pentium MMX processor at 233 Mhz, running Windows 98. The robot has on-board stereo vision and Wi-Fi communications.

ARTUR-ito is equipped with a Hokuyo URG-04LX laser scanner, with a sensing range from 20 mm to 4 m and 240° of field-of-view. Although this laser is able to provide almost 700 readings per scan. It was decided to use only 68 readings while conserving the maximum field-of-view. This is, one ray scan at every 3.53 °(from −120° to 120°) conformed one 68-scans laser observation. The laser is situated at 80 cm high to the floor and the laser-scanning plane remains parallel to the floor (Fig. 21).

10.2 The Experiments

Several experiments were conducted to evaluate the OVL performance in both real and simulated environments. The diagram in Fig. 22 shows the experiment sequence. The initial task was defining the best set of parameters for OVL. Then an exhaustive evaluation of OVL solving PT, GL

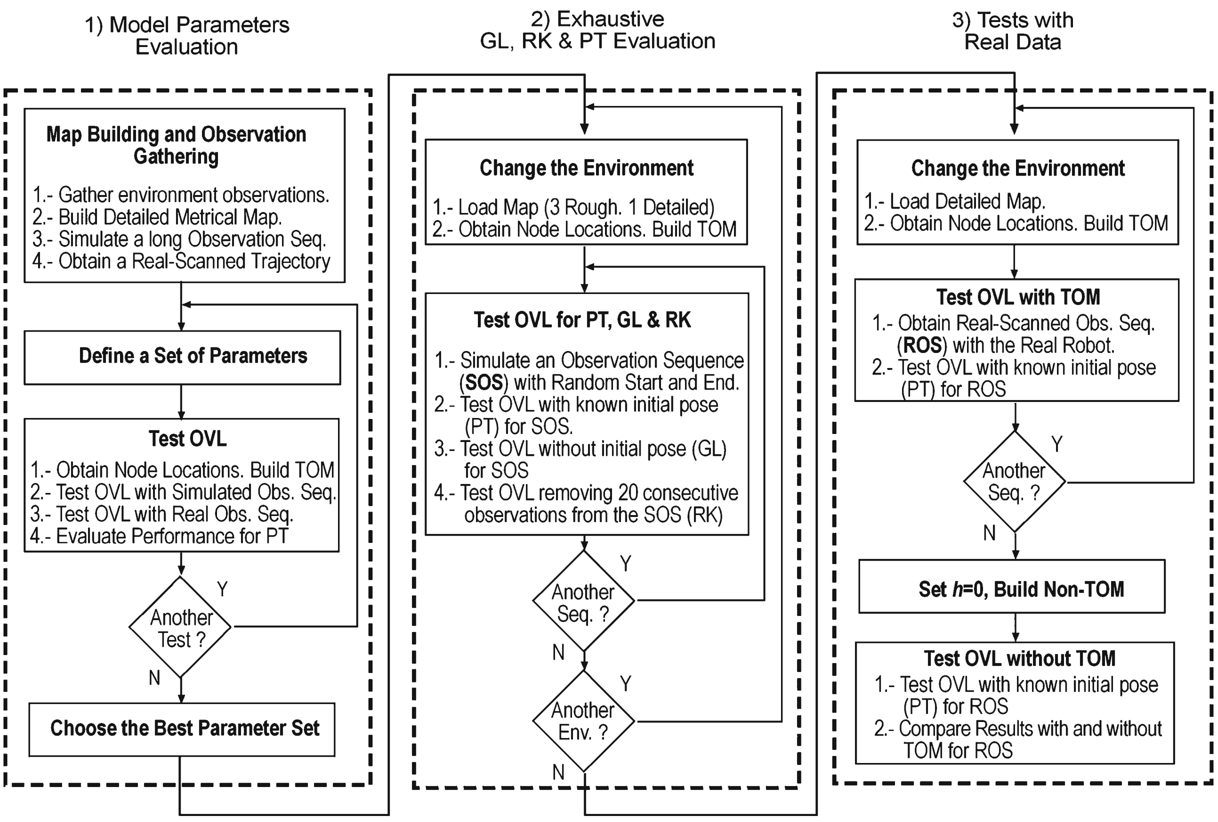
1) Model Parameters Evaluation
Map Building and Observation Gathering
1.- Gather environment observations.
2.- Build Detailed Metrical Map.
3.- Simulate a long Observation Seq.
4.- Obtain a Real-Scanned Trajectory
Define a Set of Parameters
Test OVL
1.- Obtain Node Locations. Build TOM
2.- Test OVL with Simulated Obs. Seq.
3.- Test OVL with Real Obs. Seq.
4.- Evaluate Performance for PT
Another Test ?
Y
N
Choose the Best Parameter Set
2) Exhaustive GL, RK & PT Evaluation
Change the Environment
1.- Load Map (3 Rough. 1 Detailed)
2.- Obtain Node Locations. Build TOM
Test OVL for PT, GL & RK
1.- Simulate an Observation Sequence (SOS) with Random Start and End.
2.- Test OVL with known initial pose (PT) for SOS.
3.- Test OVL without initial pose (GL) for SOS
4.- Test OVL removing 20 consecutive observations from the SOS (RK)
Another Seq. ?
Y
N
Another Env. ?
Y
N
3) Tests with Real Data
Change the Environment
1.- Load Detailed Map.
2.- Obtain Node Locations. Build TOM
Test OVL with TOM
1.- Obtain Real-Scanned Obs. Seq. (ROS) with the Real Robot.
2.- Test OVL with known initial pose (PT) for ROS
Another Seq. ?
Y
N
Set h=0, Build Non-TOM
Test OVL without TOM
1.- Test OVL with known initial pose (PT) for ROS
2.- Compare Results with and without TOM for ROS

Fig. 22 Experiment sequence for evaluating optimized-OVL

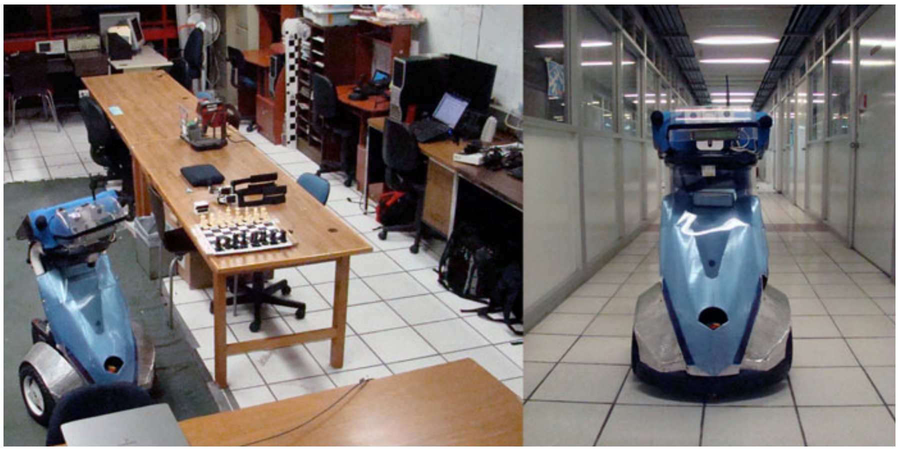

Fig. 23 ARTUR-ito in a real scenario. *Left*. Bio-robotics laboratory. *Right*. Long corridor

and RK both in detailed and rough maps was performed. Finally, tests with real data with and without a TOM were conducted to evaluate the aid of such kind of pose-observation relationships.

10.3 Model Parameters Evaluation

10.3.1 Map Construction

First, a detailed metrical map representation was built by manually displacing the real robot in the environment and gathering laser range readings. The real environment is the Biorobotics Laboratory of the National Autonomous University of Mexico UNAM and a long corridor (Fig. 23).

In order to minimize the initial pose error when building the map, the floor tiling (at every 30 cm) was used to set the robot at every 2.10 m ± 3 cm, performing four scans at the absolute headings of zero, 90°, 180° and 270° at every robot stop (Fig. 24, left). This process took about 2 hours for scanning the 50 m long corridor and the 5 × 7 m Biorobotics Laboratory.

With the set of gathered observations a metric map representation was built (Fig. 24, right) following the procedure described in Section 6.3.

10.3.2 Best OVL Parameter Set

Five OVL parameters were evaluated to test their contribution to the overall OVL performance: (a) the number of *xy* discrete nodes (see Section 5.1), (b) the number of absolute headings (see Section 5.2), (c) SOM size in neurons (see

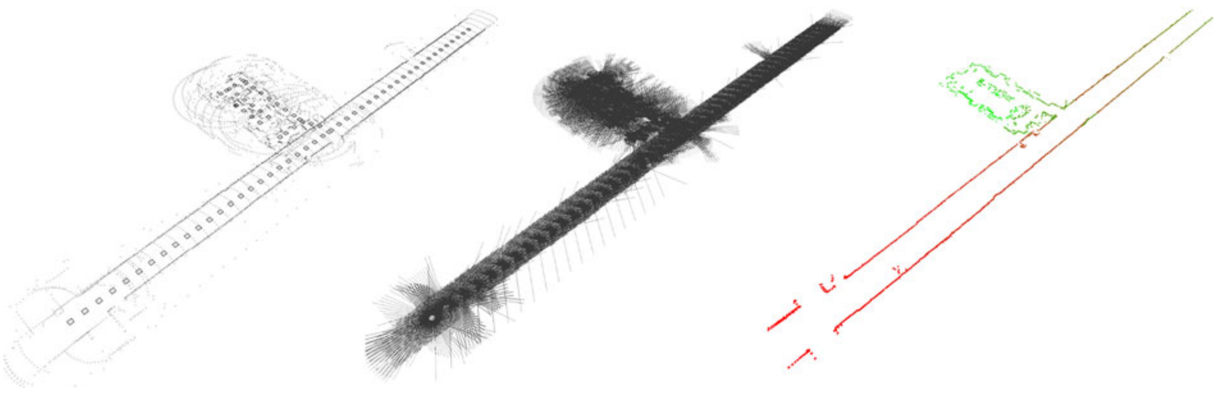

Fig. 24 Detailed environment. Robot stops (*left*). Actual scans (*center*). Final map (*right*)

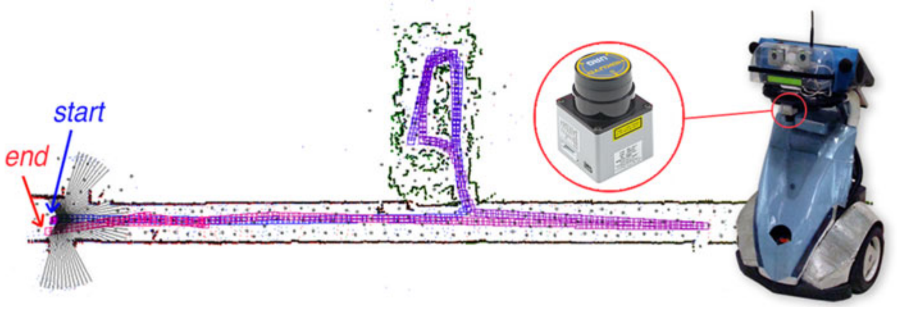

Fig. 25 Real scenario (50 × 12 m.) built with Hokuyo URG laser, ARTUR-ito and 258 m path used for the simulated tests

Section 6.5.2), (d) TOM's neighborhood *h* (see Section 6.6) and (e) samples per state for building the TOM state-observation relationship (see Section 6.4).

The procedure described from Section 5.1 to Section 6.2 was followed for building the set of states, training the SOM and building the TOM by artificially generating supposed observations at the navigable locations. Then, eight optimized-OVL PT trials with the same trajectory (a 258 m path with 314 observations, see Fig. 25), one observation simulated roughly at every 80 cm in the detailed environment (Fig. 24) were run to determine the best set of parameters.

After each simulated test, another OVL run with the same set of parameters but with a real observation sequence (Fig. 26) was run to compare the simulated OVL performance under real observation conditions, as the true robot pose in the real environment cannot be precisely evaluated under continuous robot motions. In order to introduce noise and occlusions some objects like chairs, lockers, boxes and small furnitures were moved (Fig. 26 left, red squares) from their original positions (Fig. 26 left, green squares).

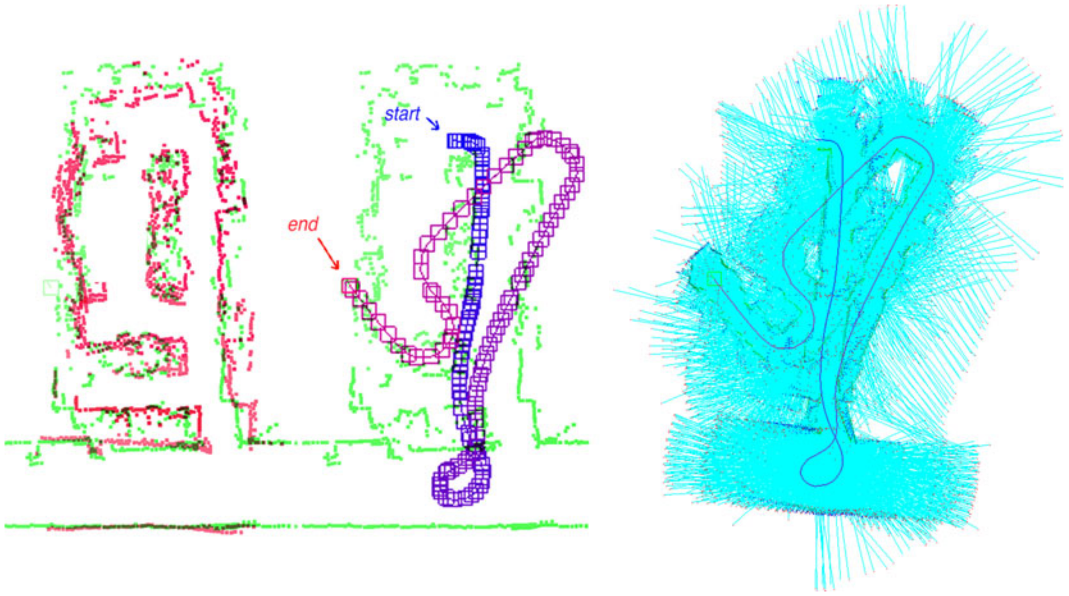

Fig. 26 Real-data Testing Sequence. Original map (*left*, *green squares*). Testing map (*left*, *red squares*). Pure odometry motion estimation (*center*). Actual sensor readings gathered (*right*)

10.3.3 OVL Performance Evaluation

An average location error was calculated between the true robot pose and the one calculated with OVL for the set of 314 simulated observations as

$$Err_\theta = \frac{1}{n}\sum\nolimits_{i=1}^{n} |\theta r_i - \theta c_i| \tag{66}$$

$$Err_{xy} = \frac{1}{n}\sum\nolimits_{i=1}^{n} \sqrt{(xr_i - xc_i)^2 + (yr_i - yc_i)^2} \tag{67}$$

where n is the number of states in Φ_t, $(xr_i, yr_i, \theta r_i)$ is the actual robot pose at every step of the 314 observation's trajectory, while $(xc_i, yc_i, \theta c_i)$ is the pose computed with the OVL algorithm with Eq. 61.

10.4 Exhaustive OVL Evaluation for PT, GL and RK

With the set of parameters that provided the best trade-off between location error, processing time and noise tolerance, a test of 10 trials at random starting and ending locations in four different simulated scenarios was run to get the overall performance under PT, GL and RK sub-problems (120 trials in total). The first, second and third simulated environments correspond to robocup@home 2006, 2007 and 2008 competition scenarios. The fourth environment is the detailed map of the Biorobotics Laboratory (Fig. 27).

10.5 Tests with Real Data with and Without a TOM

Ten trials were conducted to test the OVL performance with real data variations by randomly choosing a start and ending poses into the real environment (Fig. 23, Left) with the environment differences depicted in Fig. 26 left (red squares). Then, the experiment was repeated removing the Tolerant Observation Model, in order to prove the aid of the tolerance to sensor noise and occlusions.

10.6 Comparison Against a State-of-the-Art Algorithm

A comparison of optimized-OVL vs. classical Monte Carlo Localization (MCL) [11] was performed. By using the ray-tracing simulator with the detailed environment, it was possible to simulate a 'precise' supposed observation for MCL in real-time.

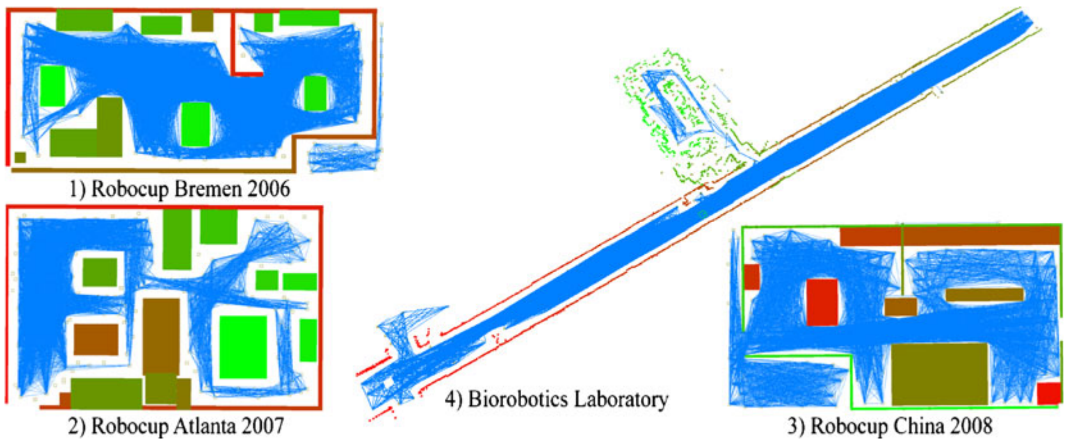

Fig. 27 Maps used in the tests. Robocup@home 2006, 2007 and 2008 rough maps (1 to 3). Detailed Biorobotics Lab. (4). The connectivity network is shown

10.7 Method Performance on Small Processors

Finally, optimized-OVL was run in real-time on a Pentium MMX 133 Mhz processor with 64 MB of RAM, incorporated in the robot ARTUR-ito (Fig. 21).

11 Results

11.1 Model Parameters Evaluation

The first column shows the number of experiment, then the number of absolute xy nodes (number of Vector Quantization neurons) is shown in column 2, followed by the number of absolute headings in the range of 2..2Π (3rd column).

The 4th column indicates the number of discrete symbols (neurons) used in the Self Organized Map while the 5th columns computes the total number of states in Φ_t as the number of xy nodes multiplied by the number of absolute headings.

The 6th column indicates the number of randomly simulated observations per state. The 7th calculates the total simulated samples used for training the SOM and building the TOM as the number of states (5th column) multiplied by the number of simulated samples per state (6th column).

In the 8th column the h parameter—the ‘tolerance’—of the TOM is shown. $h/2$ is the neighborhood influence ratio of the Gaussian function that computes $p(z_t|x_t)$ with Eq. 60. The Gaussian influence neighborhood function was estimated based on the work of Liu and Haralick [35] that helps to calculate σ^2_{tol} in Eq. 58, based on a Gaussian kernel of size h. They applied their theories in building Gaussian kernels for image processing.

Columns 9 and 10 show the average absolute location and heading errors respectively, measured with Eqs. 66 and 67.

Column 11 computes the total processing time the OVL algorithm requires for calculating the 314 observations trajectory. It can be noticed how the number of states affects the method's processing time.

Finally, the 12th column shows a qualitative analysis of the OVL performance with the real-data sequence of Fig. 26, analyzing the tolerance to environment changes, update time and sensor errors.

11.2 Best Parameter Set

Simulated (Table 1, columns 9 and 10) and real experiments (Table 1, column 12) gave the best results (a trade-off between speed and location error) with 256 xy discrete locations and 16 absolute headings, starting from 0 and up to 337.5° in 22.5° intervals (4096 states) for an average node separation of 0.68 m. 10,000 simulated observations were used to train the 16 × 16 SOM and almost 410,000 for building a TOM (a minimum of 50 samples per state was experimentally determined and 100 as the optimum). This means at least one sample per dm^2 per absolute heading.

σ_{tol} was set to $0.1092 \times h/2 + 0.4335$ based on [35], this is 0.87 for $h = 8$, while $\sigma_d = 2/3 \times avg\_node\_separation = 0.454m$ and $\sigma_\theta = \Pi/(2 \times headings) = \Pi/32$ was the best parameter set.

$\varepsilon = 1/Number\_of\_States - 1 \times 10^{-10}$ was found as the best trigger value (a uniformly distributed PDF) while consecutive observations z_t and displacements u_t should be taken when the robot displacement surpasses the average xy node separation or when the robot's heading change is above $2\Pi/headings$, otherwise the method tends to remain in the same state or in a small state neighborhood.

11.3 Exhaustive OVL Evaluation for PT, GL and RK

Table 2 (rows 1–3) and Fig. 28 show the performance of OVL solving PT in three simulated environments. The 4th column in Table 2 shows the average deviation from the true pose (red path in Fig. 28). The 4th row shows the simulation results in the real-scanned environment (Fig. 24).

Table 1 OVL PT Parameter tests for a 314 observations sequence

#	nodes *xy*	Headings	SOM Symbols	States	Samples per State	Total OM Samples	*h*	Avg. | *xy error* | Sim. (m)	Avg. | *θ error* | Sim. (m)	Total Processing Time (s)	Performance with real data.
1	256	16	256	4096	100	409600	8	0.33	0.06	6.16	Best, tolerates env. changes and errors
2	256	36	256	9216	10	92160	8	0.40	0.11	31.82	Not bad, just tolerates reflections
3	256	16	256	4096	10	40960	8	0.34	0.06	6.16	Not good, non tolerant to big changes
4	256	16	256	4096	20	81920	8	0.34	0.13	6.16	Not bad, just tolerates reflections
5	256	36	256	18432	10	184320	11	0.21	0.06	117.91	Good, tolerates most errors
6	64	36	256	4608	10	46080	6	0.11	0.06	6.7	Bad, fails on real environments
7	144	16	64	4096	20	81920	4	0.34	0.12	9.51	Good, tolerates most errors
8	256	16	144	4096	50	204800	6	0.30	0.08	8.26	Not good, non tolerant to big changes

It is notorious that the OVL accuracy is mainly related with the spatial separation between discrete nodes (not in an exact linear relation) as we are approximating the true robot pose from a discrete probability distribution. Because the method accuracy depends on several parameters (five in the Table 1) and we are testing the same parameter set in many environments, it is possible for instance that environments with more different observations than others would require bigger SOM elements. In such cases it will be desirable to vary some parameters and rebuilding the models.

In this way, the OVL accuracy becomes an optimization problem that could be treated in the future with multi-objective optimization techniques like Genetic Programming. By now the goal of this article is demonstrating that a parameter set can have an acceptable performance in many environments.

Figure 28 shows how it is possible computing a continous path from discrete samples (states). Despite the quantitative results in Table 2 show an average error above 16 cm, the comparison between the real motions path (red line) against the path calculated with optimized-OVL (blue line) showed that, despite in certain zones the calculated path is somewhat distant to the real one, in general OVL keeps track of the robot pose very well.

Based on the above results, it is possible to estimate that the OVL *xy* location error is about 50% of the average node separation and the angular error is also about 50% of 2Π/number_of_absolute_headings (32 in the tested parameter set). For instance, in order to have an average *xy* error of 5 cms, it would be necessary at least three times more nodes (states) in the environments 1 to 3 and five times more states in the 4th environment.

In the GL tests, the method was able to estimate the true robot pose after 6 updates in average. After a RK, in only two updates the probability decay is enough to force GL to relocalize the robot in 10 updates in average.

Figure 29 demonstrates OVL over PT, RK and GL with the simulated environment number 2 for 4096 states (256 *xy* locations by 16 absolute headings).

Table 2 OVL Performance tests for PT in simulated environments

Env.	Map Dimensions Width & Height (m)	Avg. Node Separation (m)	Absolute Avg. Location Error (m)	Absolute Avg. Heading Error (rad)
1	13 × 7	0.49	0.16	0.06
2	11 × 7	0.30	0.19	0.12
3	12 × 5	0.33	0.19	0.08
4	50 × 12	0.68	0.25	0.10

In Fig. 29, the robot starts at the known location x_0 performing PT, thus the maximum trigger value $\max(\phi_0)$ is near 1.0. After a RK, in only one update, the maximum trigger value $\max(\phi_t)$ decays until reaching ε thus firing the GL and evaluating all the state-transitions (4096). Ten iterations later, OVL finds the true robot pose again, continuing with PT. As time passes, the number of evaluated states (trigger firing) reaches the rates before the RK (about 80 state-transitions).

It is notorious how the method can handle GL, PT and RK based both in the update triggering and the method initialization. Nevertheless, one problem arises when GL is fired when ϕ_t decays fast (as the case of having severe occlusions): the time consumed for evaluating 4096 state-transitions is considerably larger than the time needed for performing PT (4 s for 4096 transitions in GL against 0.02 s for 80 transitions in PT). Usually the Probabilistic Localization Methods as [11] limit the amount of change in the pose beliefs ϕ_t respect to ϕ_{t-1}, in order to limit the belief changes. This would help OVL to be less sensitive to observation changes.

The total amount of memory required by optimized-OVL was 1.07 MB. Compared with 16 MB required for storing only a full $\mathbf{A}(i, j)$ transition matrix for 4096 states, the saving is notorious.

11.4 Tests with Real Data with a TOM

Table 3 shows results in the environment 4 (real scanned) with real observations (Fig. 27, 4). The largest location errors were obtained in

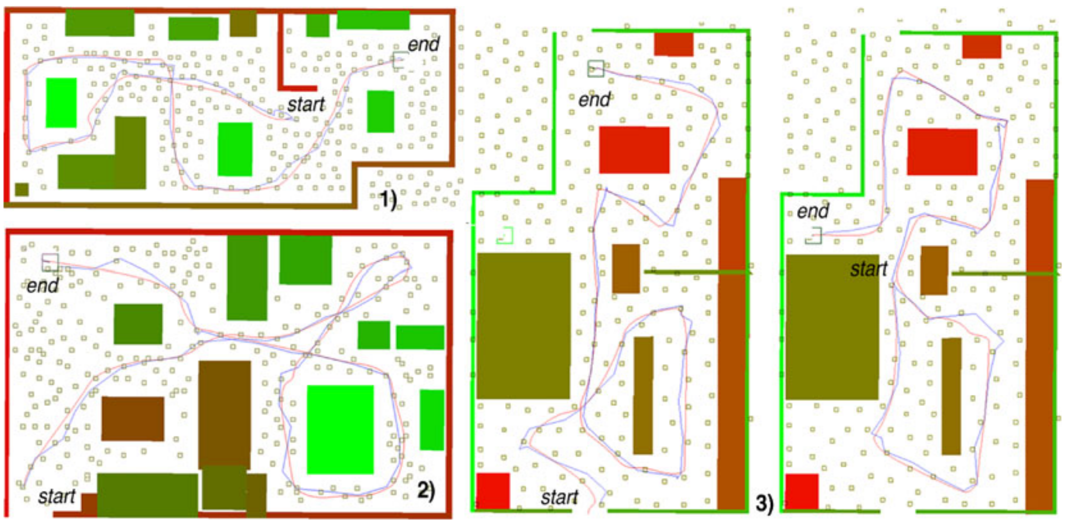

Fig. 28 OVL PT on three simulated Environments. *Red line*: real path. *Blue line*: found path. *Big square*: final robot location. *Small gray squares*: discrete states. Results show that it is possible to get a continuous position estimation from discrete node samples

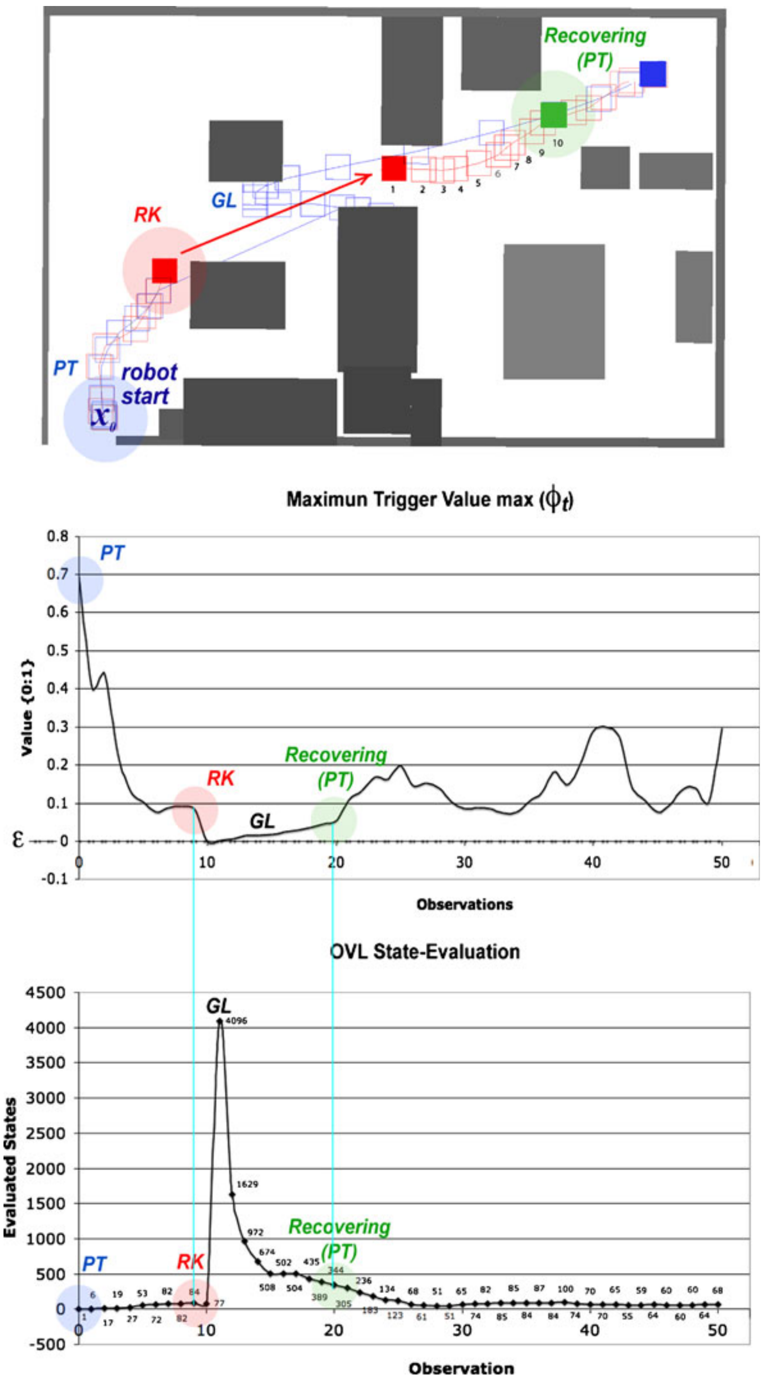

Fig. 29 OVL solving PT, RK and GL. Red path: true trajectory. *Blue path*: found trajectory. The robot starts at the known pose x_0 and starts PT (*blue circle*). After 10 observations-displacements a RK is performed by abruptly displacing the robot (*red circle* and *arrow*). OVL fires GL and in ten observations the method has found the true robot pose (*green circle*)

trajectories traversing a long corridor, but the method performance did not decay too much.

In this table, is evident that the OVL performance depends mostly on the chosen sequence path. The same issue also applies for Particle and Kalman Filters. For example, although the average node separation is the same for all the above tests (0.68 m) and all the tests use exactly the same TOM, the first sequence has a greater xy location error (0.33 m) than the fourth (0.13 m). This is because the long first sequence crosses several times the long corridor (Fig. 25) compared with

Table 3 OVL Performance tests for PT in a real environment

Run	Number of Observations	Accumulated Traveled Distance (m)	Absolute Avg. Location Error (m)	Absolute Heading Error (rad)
1	314	258.20	0.33	0.06
2	37	24.60	0.16	0.15
3	43	29.69	0.23	0.10
4	21	16.00	0.13	0.08
5	70	53.61	0.17	0.11
6	37	24.89	0.23	0.08
7	20	12.5	0.25	0.09
8	51	34.9	0.24	0.14
9	19	9.81	0.27	0.10
10	44	28.82	0.24	0.08

the small fourth sequence that reflects a robot motion inside de Biorobotics Lab (Fig. 26). In this sense, OVL presents the same problems than PFs and KFs: the long trajectories with very similar observations (i.e. the corridors).

11.5 Tests with Real Data Without a TOM

Despite the use of a TOM has almost no effects in simulated environments, under real conditions having occlusions and sensor noise, it really helped to keep track of the true robot pose (Fig. 30). Without a TOM the robot would have easily got lost.

As the Tolerant Observation Model is basically a probabilistic tool for comparing high-dimensional observations, its applicability scope is not exclusive restricted to Viterbi Localization. In this way a TOM could be incorporated to those methods that require vector comparison. Even as an optimization to the Monte Carlo Particle Filter [11]. The main contribution of the Self Organized Maps is the easiness for arranging similar observations in the 2D map.

11.6 Comparison Against a State-of-the-Art Algorithm

Figure 31 (left) shows a comparison of standard Monte Carlo Localization (MCL) using a real-time scan simulator and 5,000 particles against optimized-OVL (Fig. 31, right), for the same

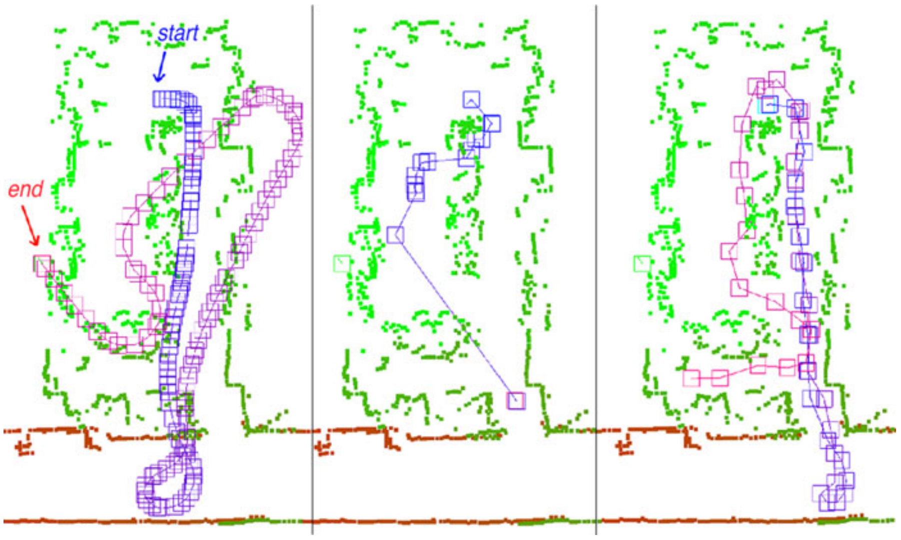

Fig. 30 OVL algorithm with real observation scans for a PT problem. Left: robot pure-odometry pose estimation. Center: OVL without a TOM. Right: OVL with TOM. The aid of a Tolerant Model in real scenarios is notorious

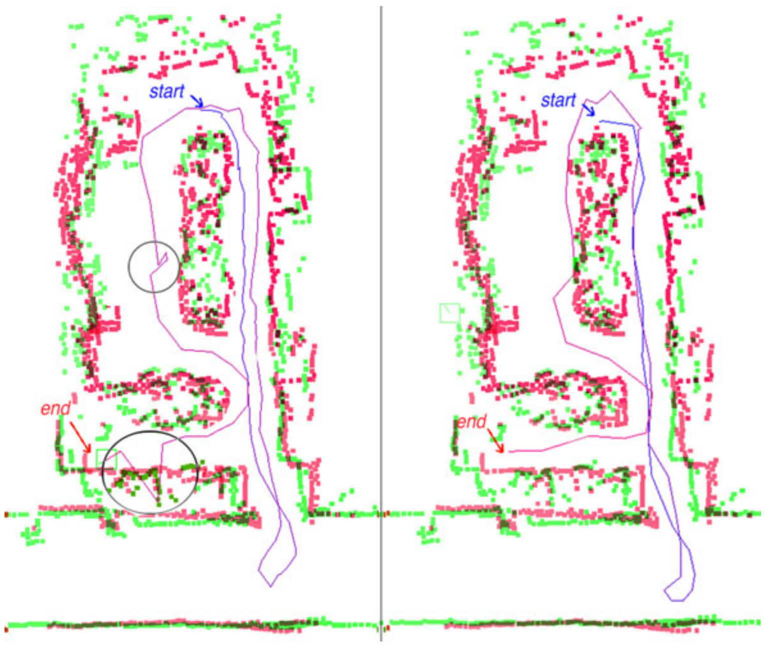

Fig. 31 Comparison between standard Monte Carlo Localization MCL (*left*) and Odometry Viterbi Localization OVL (*right*). *Green squares* depict original mapped objects. *Red squares* show object locations when running the test. It can be noticed the map variations that caused MCL to fail on circled locations

trajectory inside the Bio-robotics lab (Fig. 31, left) using the Hokuyo URG-04LX laser scanner with a 68-scans observation. MCL fails on two locations (circled) due to high changes in the environment (red squares) compared with the original mapped environment (green squares). The time spent by MCL per update was 5 s against 0.02 s of OVL. In the case of GL, about 10,000 particles are needed by MCL to correctly find the true robot pose (consuming 10 s per update). Even in the worst case, the 4 s needed by OVL for reevaluating 4096 states for GL results smaller.

After comparing the performance of OVL against classical Monte Carlo Localization one question arises:

What would happen if a PF is used with similar intermediate simulated observation and motion process, would it be improved in efficiency as well?

The answer to this question is related with the way that both methods operate: the PFs efficiency rely in the precise prediction both of the next robot location after a displacement (by means of a resampling) and the ‘supposed’ observation for that new location (by means of a simulator). In this way, PFs performance basically depends on ‘guessing’ an appropriate location based on motions and having enough noise in the sensor to approximate the simulated observation to the real laser scan. In OVL the set of samples (the *states*) remains constant during all the algorithm execution and the efficiency depends mostly on the physical separation among those states and the relationship between locations and observations stored in the OM (off-line built).

For this reason, the PFs performance would decay by having a fixed location resampling set. Despite this, the Observation Model in PFs could improve its efficiency because a SOM allows naturally correlating and arranging observations into a ‘map’, and the TOM stores that relationship. By searching for the closest state location stored in a TOM, PFs could estimate the observation for a given pose and compare it against the actual observation in a faster way than simulating the observation for a relatively high number of locations (the *samples*).

In any case, the weakness of most Probabilistic Localization approaches is the estimation of the next robot pose with the Motion Model (MM). If the MM does not represent with precision the way the mobile robot pose changes based on odometry, in a very few steps PFs and KFs can easily loose track of the robot. As OVL has a fixed set of 'samples' represented by the fixed state locations, results less sensitive to imprecise motion estimations.

According to this, OVL can be suitable of being implemented in mobile robots whose motions are difficult to predict (as the case when the robot does not have odometers). Good examples are legged robots, nano-robots, and applications that calculate the pose based on accelerometers or by matching observations without odometry.

Finally, it is appropriate to mention that the performance of all Probabilistic Localization methods is mainly driven by the similarity between observations in different locations of the map. Similarities in the observations will therefore induce a strong uncertainty in the final robot pose.

11.7 Method Performance Evaluation

The average time per iteration, running the optimized version of OVL was 0.02s in an Intel Core 2 Duo processor, enough to run in real-time. Without the optimization, a full update took about 4 s.

The optimized-OVL tests running on a Pentium 133 Mhz computer of the ARTUR-ito robot were able to locate the robot with an average update time of 0.25 s. As the maximum robot speed is limited to 1 m/s, this update time is enough for using OVL in real-time.

12 Discussion

The first noticeable fact is that OVL can effectively calculate a continuous robot pose from a set of discrete locations. As a difference with previous discrete localization methods [9], there is no need of a huge 3D location probability array (the Env. 4 would need a $200 \times 48 \times 16$ location matrix to reach the same error rates: 37.5 times more memory space than the Φ vector for 4096 states).

The best attribute of OVL is the ability of managing without any further modification all the three sub-types of localization: GL, PT and RK. With an initial pose supposition it can handle with PT, if not, in a very small number of iterations it can find the true robot pose, solving GL, and automatically begins to keep track of the robot. If a RK is suddenly performed, the belief probability degenerates fast (due to the observation probability decay) forcing a GL, finding the true pose in 10 to 15 iterations. Also, the optimized version of OVL has proven its applicability to real-time applications running in processors with limited computing resources.

The main problem that causes the method to fail consists in evaluating the new pose when the robot displacement is below the average xy node separation (e.g. below 68 cm for env. #4), because the probability of remaining in the same state for a small displacement will be considerably higher than the probability of performing a true transition to another state. In fact, after many small displacements the robot could have changed from x_i to x_j, but the method would remain the robot at state x_i as long as the observation remain somewhat similar.

Another important issue is related with the GL firing when ϕ_t decays fast. As it is stated in this work, the method fires GL almost immediately after a RK. In future works the method will be tested in crowded environments and the pose belief change will be quoted, in order to determine if OVL can be less sensitive to improbable observations. This was the main reason of integrating a TOM.

By the other hand, the main problem that compromises OVL performance in real-time is the variable processing time needed for the belief update, especially after a RK. The use of FPGA hardware could help the method to achieve the same update rates at high-speed, regardless of GL, PT or RK.

Related to this, results very important considering enough simulated observations when building the TOM, otherwise the method would be constantly performing GL due to the probability decay when in crowded environments.

13 Conclusions

Compared with the most widely used probabilistic pose estimation methods [11, 12] the Odometry-based Viterbi Localization presents serious advantages: It is not necessary to estimate with high precision the observation at specific poses during operation as Monte Carlo Localization. Neither requires matrix inversion like Kalman Filters. Despite its discrete nature, it is able to estimate a continuous pose with limited resources. It can also handle with many hypotheses at once, as most of the state-of-the-art algorithms do. Also, because of the introduction of a Self Organized Map, OVL is able to use *n-dimensional* observations (including high-resolution visual information, or data coming from many sensors) because the hard work is performed by Vector Quantization Networks (perfectly suitable of being implemented in hardware).

Compared with the previous Viterbi Localization Approach by Savage et al. [13], optimized-OVL has many important improvements:

1. The use of an odometry-dependent Motion Model.
2. Larger number of states (thousands against a dozen).
3. Continuous pose estimation. There is no need of 'backtracking' for estimating the best discrete state sequence. All the trajectories (state-transitions) are considered.
4. Ability of handling GL, PT and RK by incorporating Update Triggering.
5. Self Organized Maps for building a Tolerant Observation Model.
6. No probabilistic tendency to zero out.

In conclusion, OVL is a Non-degenerative Probabilistic Algorithm (NDPA) that can effectively solve the entire robot localization problem, based both in odometry and observation information. It saves memory and processing time due to its discrete nature, but provides continuous pose estimations. Finally, it can run in single and multi-core microprocessors in real-time.

14 Future Works

Although in this paper we have started from a previously built map and statistically generated the Observation and Motion Models, by using an ANN architecture a real-time learning procedure can be used for re-adapting these model parameters to the actual operation conditions (fine tuning). This architecture and its training will be presented in the future for adapting OVL to changing environments.

Also, some works using stereo and omni-vision cameras as the main sensor will be presented, because OVL generality easily allows using visual information for training the VQN. In this case, the narrowed field-of-view of computer cameras (45° or so) compared with the field-of-view of the laser range scanners (240°) and the two-dimensional visual information would make very interesting the process of building a TOM. In this sense, stereoscopic cameras could help to provide depth information (as laser range finders do). Another important factor would be a variable illumination, thus the Vector Quantization of observations could incorporate some previous image feature extraction (like corners of SIFT [36]), instead of the raw sensors values: in this case three values for every pixel: red, green and blue.

Acknowledgements This work was supported in part by PAPIIT-DGAPA UNAM under Grants IN107609, IN113611 and IX100610, the Mexican Council of Science and Technology CONACYT and Posgrado en Ciencia e Ingeniería de la Computación PCIC, Universidad Nacional Autónoma de México UNAM.

References

1. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT Press (2005)
2. Llarena, A.: Here comes the robotic brain. Trends in intelligent robotics. Commun. Dependability Qual. Manag. **103**(2), 114–121 (2010)
3. Choi, W.S., Oh S.Y.: Range sensor-based robot localization using neural network. In: International Conference on Control, Automation and Systems, pp. 230–234 (2007)
4. Djekoune, O., Achour, K.: Vision-guided Mobile Robot Navigation Using Neural Network. In: Proceedings of 2nd International Symposium on Image and Signal Processing and Analysis, pp. 355–361 (2001)
5. Janet, J.A., Gutierrez, R., Chase, T.A.: Autonomous mobile robot global self-localization using kohonen and region-feature neural networks. Journal of Robotic Systems 263–282 (1997)
6. Racz, J., Dubrawski, A.: Mobile Robot Localization With an Artificial Neural Network. International Workshop on Intelligent Robotic Systems IRS'94, Grenoble, France (1994)

7. Wang, K., Wang, W., Zhuang, Y.: Appearance-Based Map Learning for Mobile Robot by Using Generalized Regression Neural Network. ISNN (1): 834–842 (2007)
8. Conforth, M., Meng, Y.: An artificial neural network based learning method for mobile robot localization. Robotics automation and control, Pavla Pecherkova, Miroslav Flidr and Jindrich Dunik (Ed.), ISBN: 978-953-7619-18-3, InTech (2008)
9. Fox, D., Burgard, W., Thrun, S.: Markov localization for reliable robot navigation and people detection. In: Modeling and Planning for Sensor-Based Intelligent Robot Systems. Springer, Berlin (1999)
10. Simmons R., Koenig, S.: Probabilistic Navigation in Partially Observable Environments. IJCAI '95, Montreal Canada (1995)
11. Thrun, S., Fox, D., Burgard, W., Dellaert, F.: Robust Monte Carlo localization for mobile robots. AI **128**, 99–141 (2000)
12. Roumeliotis, S., Bekey, G.: Bayesian estimation and Kalman filtering: A unified framework for mobile robot localization. In: Proc.2000 IEEE ICRA, 22–28 April, pp. 2985–2992 San Francisco, CA (2000)
13. Savage, J., Morales, M., Márquez, E.: The Use of Hidden Markov Models and Vector Quantization for Mobile Robot Localization. Robotics and Applications (RA 2005), IASTED, Boston, U.S.A.
14. Lu, F., Milios, E.: Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. JIRS (18), No. 3, March 1997, pp. 249–275. 9705
15. Crowley, J.L., Demazeau, Y.: Principles and techniques for sensor data fusion. Signal Process. 32(1–2) S. 5–27 (1993)
16. Diosi A., Kleeman, L.: Advanced sonar and laser range finder fusion for simultaneous localization and mapping. Proc. IROS (2004)
17. Elfes, A.: Using occupancy grids for mobile robot perception and navigation. Computer. **22**(6):46–57 (1989)
18. Thrun, S., Gutmann, J.-S., Fox, D., Burgard, W., Kuipers, B.: Integrating topological and metric maps for mobile robot navigation: A statistical approach. In: Proceedings of the AAAI Fifteenth National Conference on Artificial Intelligence (1998)
19. Savage, J., Llarena, A., Carrera, G., Cuellar, S., Esparza, D., Minami, Y., Peñuelas, U.: ViRbot: a system for the operation of mobile robots. In: Proc. RoboCup, pp. 512–519 (2007)
20. Barreto, G.D.A., Araújo, A.F.R., Ritter, H.: Self-Organizing Feature Maps for Modeling and Control of Robotic Manipulators. presented at Journal of Intelligent and Robotic Systems, pp. 407–450 (2003)
21. Rabiner, L.R.: A tutorial on Hidden Markov models and selected applications in speech recognition. Proc. I.E.E.E. **77**(2), 257–285 (1989)
22. Menegatti, A., Pretto, A., Scarpa, E., Pagello: Omnidirectional vision scan matching for robot localization in dynamic environments. IEEE Trans. Robot. **22**(3), 523–535 (2003)
23. Little, J., Se, S., Lowe, D.: Vision-based mobile robot localization and mapping using scale-invariant features. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 2051–2058 (2001)
24. Jaynes, E.T.: Probability theory: the logic of science", Cambridge University Press. ISBN 9780521592710 (2003)
25. Kailath, T.: Lectures notes on Wiener and Kalman filtering. Springer (1981)
26. Julier S.J., Uhlmann. J.K.: A new extension of the Kalman Filter to Nonlinear Systems. Proc. of AeroSense, The 11th Int. Symp. On Aerospace/Defense Sensing, Simulation and Controls (1997)
27. Julier, S.J., Uhlmann, J.K., Durrant-Whyte. H.: A new approach for nonlinear systems. In: Proceedings of the American Control Conference, pp. 1628–1632 (1995)
28. Jazwinski, A.H.: Stochastic processes and filtering theory. Academic Press, New York, NY (1970)
29. Doucet, A., de Freitas, N., Murphy, K., Russell, S.: Rao-blackwellised particle filtering for dynamic Bayesian networks. In: UAI (2000)
30. Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans Inf Theory **13**(2), 260–269 (1967)
31. Fortune. S.: A sweep line algorithm for Voronoi diagrams. Proceedings of the second annual symposium on Computational geometry. Yorktown Heights, New York, United States, pp. 313–322 (1986). ISBN: 0-89791-194-6
32. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers, Boston, MA (1991)
33. Latombe, J.C.: Robot Motion Planning, 3rd edn. Kluwer, Boston (1991)
34. Haykin, S.: Neural Networks. A Comprehensive Foundation. 2nd edn. Prentice Hall (1999)
35. Liu, G. Haralick, R.M.: Two Practical Issues in Canny's Edge Detector Implementation. In: icpr. 3,15th International Conference on Pattern Recognition (ICPR'00), vol. 3, p. 3680 (2000)
36. Lowe, D.: Object recognition from local scale-invariant features. In: Proceedings of the Seventh International Conference on Computer Vision(ICCV'99), September 1999, pp. 1150–1157. Kerkyra, Greece