Comparing Bayesian and Montecarlo localization for a robot with local vision

María A. Crespo, José M. Cañas, Vicente Matellán

Universidad Rey Juan Carlos,
Móstoles, Spain
{mangeles,jmplaza,vmo}@gsyc.escet.urjc.es
http://gsyc.escet.urjc.es/robotica

Abstract. Position estimation is one of the classical problems in mobile robotics. For instance, robots have to know where in the map they are in order to use maps in any task involving navigation. Even in highly dynamical environments such as the RoboCup competition the robot behaviour or attitude depends on its position in the playground. The goal of this paper is to compare two probabilistic localization methods based on local vision for a mobile robot. The experimental set up is based on the Aibo league of the RoboCup, where the robotic dogs major sensor is the on-board camera. Two localization algorithms, Bayesian and Montecarlo ones, have been implemented and compared, and their behaviour studied in several situations. A simulator has been developed which adds actuation noise to the commands ordered to the motors and sensor errors to the images perceived. This way both algorithms use exactly the same data collection to estimate the robot position.

1 Introduction: Localization sensors and techniques

In order to use a map the robot has to locate itself in the frame of reference of the map, that way, it can use information of obstacles out of the current sensor scope, plan paths through the non-visible environment, or just take decisions according to its position.

Perception is a key element for localization. In fact, we can divide the solutions to the localization problem between the ones which are just confident in specialized localization sensors, and the ones which use indirect information. Localization sensors provide explicit position information, local or global. For instance, the encoders provide pulses proportional to the wheels displacement, which allow us to estimate the robot position knowing the initial one. The disadvantage of these sensors is the odometric errors, which can be systematic and non-systematic. Systematic errors are due to the robot and its sensors; they are important because they accumulate as the robot moves, but can be easily corrected by calibrating the system. Non-systematic errors are unpredictable as they are due to the environmental conditions and therefore more difficult to fix.

Another type of localization sensor is the GPS (Global Positioning System) sensor, widely used in outdoors applications. It uses a receiver of the radio signals from at least four satellites. Triangulation is used to get latitude, longitude and height. Its main advantage is that it can provide good location metrics for big areas. There are mainly two types of GPS errors: those derived form the transmission of the radio signals and those due to the system configuration.

There are other localization strategies based on different methods, some of them require environmental engineering such as the placement of active beacons, passive marks, etc.. For instance, the use of marks consists in positioning *beacons* in known places in a given map. Then, using either triangulation or trilateration we obtain the robot pose (x, y, θ). Another localization technique is the *scan matching*. It allows locating the robot by comparing sensor readings obtained as the robot moves. It requires knowing the initial pose and although this method can provide a very accurate estimate, it fails to recover when major localization errors occur.

Another family of localization techniques include those which integrate the information obtained from the robot sensors, non directly related to position. For instance, some of them are based on the *Kalman* filtering [Welch02]. Given some initial estimates, this filter allows the parameters of a model to be predicted, and adjusted with each new measurement obtained, providing a dynamic estimate of error at each update. It has been successfully used in various systems such as golf lawn mowers [Kyriy02]. Its main drawback is that it cannot store multimodal evidences of localization.

Fuzzy logic techniques have also been widely used to integrate position information, for instance in [Buschka00] a fuzzy localization method for a legged robot is described. It relies on the observation of known landmarks using a camera sensor and on the integration of the position information derived from these observations into a fuzzy position grid. Main advantages of fuzzy calculus are the limited computational cost and the ability to recover from large localization errors.

Probabilistic algorithms have also been widely used to gather localization information, and have proved very successful [Thrun00a] in many robotic environments. This paper describes the implementation and the experiments made using two of these probabilistic methods. The scenario is inspired in the Aibo league where a robot with local vision moves in the RoboCup field.

The rest of the paper will be organized as follows: next section will be dedicated to detail the underlying theory of the two probabilistic methods used in our experiments; in the third section the simulator used will be described; fourth section presents the experiments carried out and fifth section analyzes the results obtained summarizing the conclusions we have reached.

2 Probabilistic localization

Probabilistic localization has been widely used to solve the global localization problem, i.e., where the initial position of the robot is unknown. It calculates the probability of each possible position given some sensor readings and movement data provided by the robot. It copes with uncertainty and sensor errors and can recover from major localization errors.

Its main drawback is the computational cost of keeping the history of the probabilities (using Bayes's rule) for all the possible locations in the map. Because of this limitation several sampling techniques have been probed, keeping the power of the Bayes reasoning. In particular Montecarlo techniques, MCL, have gained popularity recently [Thrun01,Montemerlo02]. In MCL, instead of storing the probabilities for all the possible positions, a small number of representative samples are randomly selected. Following three steps for *predicting* the pose using actuation data, *updating* the samples probabilities using observation data and *resampling*, the robot pose can be estimated in a much more efficient way.

2.1 Bayes localization

Probabilities are held in *cubes of probabilities* that represent each possible position in the playground. The ratio (3) used to keep the probability for each (x, y, θ) is calculated applying Bayes's rule and following the formulation developed at [Margaritis98]:

$$P_{position}(C_{(x,y,\theta)}, t) = P(position/obs(t), data(t-1)) \tag{1}$$

$$r_{map} = \frac{P_{position}}{1 - P_{position}} \tag{2}$$

$$r_{map}(C_{(x,y,\theta)}, t) = \frac{r_{obs}}{r_{apriori}} * r_{map}(C_{(x,y,\theta)}, t-1) \tag{3}$$

Initially, the probability is set to 0.5 for all positions in the cube (we don't know where the robot is). Then, when new values are combined using Bayes's rule, in order to avoid saturation we set the maximum probability to 0.999999 and the minimum to 0.000001. In order to use the images taken from the camera, an observation model must be developed to translate that information into the probability framework. In our case we defined $p(position/obs) = e^{-d^2}$ where d is the difference between the ideal image at some location and the real one (4). Such difference is defined as the mean distance from the beacons i in one image to their corresponding ones j in the second image, being B_{ideal} and B_{real} the total number of beacons for each one of them.

$$d = ((\sum_{i=1}^{B_{ideal}} d_i/B_{real}) + (\sum_{j=1}^{B_{real}} d_j/B_{ideal}))/2 \tag{4}$$

2.2 MonteCarlo localization

We use the CONDENSATION algorithm as described in [Sáez02]. This algorithm works with a population of samples $M_t = \{(\varphi_t^1, \omega_t^1), (\varphi_t^2, \omega_t^2), ...(\varphi_t^n, \omega_t^n)\}$. Each sample φ_t represents a possible pose of the robot, and has a probability ω_t associated. Initially, the probability is set to 0.5 for all samples (we don't know where the robot is). This population evolves as new motor commands and new sensor observations are integrated, iterating a three steps loop:

1. **Prediction phase:** Given the action a_{t-1} performed by the robot in t-1, the predicted pose in t for each sample $\varphi^i_{t-1} \in M_{t-1}$ is obtained by adding some actuation noise. This way, we build a new set of N samples $\widetilde{M_t}$ in t.

$$\widetilde{\varphi}^i_t = \varphi^i_{t-1} + a_{t-1} + (N(0,\sigma_x), N(0,\sigma_y), N(0,\sigma_\theta)), i = 1,2,...N$$

2. **Observations update phase:** Given the observation v_t performed by the robot in t, the new probability $\widetilde{\omega}^i_t$ for each sample $\varphi^i_t \in M_t$ is updated according to the belief that the observation v_t corresponds to the state $\widetilde{\varphi}^i_t$. The observation model used was defined as $p(position/obs) = e^{-d^2/256}$ where d is the distance in pixels between the ideal image and the real one (4). To help the next resampling step to work properly, we set the maximum probability to 0.95 and the minimum to 0.30.

$$\widetilde{\omega}^i_t = p(\widetilde{\varphi}^i_t | v_t), i = 1,2,...N$$

3. **Resampling phase:** In this step a new set of N samples M_t is built resampling with substitution the set $\widetilde{M_t}$, in a way that each sample $\widetilde{\varphi}^i_t$ is chosen with a probability proportional to $\widetilde{\omega}^i_t$. Finally the probabilities ω^i_t of the samples in M_t are normalized to satisfy $\sum_{i-1}^{N} \omega^i_t = 1$.

$$(\varphi^i_t, \omega^i_t) \longleftarrow \text{ Choose a sample from } \widetilde{M_t}, i = 1,2,...N$$

As opposed to the Bayesian approach, there is no explicit history for the set of samples; history is implicit to the new set of samples concentrated around the most possible positions thanks to the resampling phase.

3 Simulator

The *Simulator* generates the required information for an off-line processing of data by the localization algorithm. This allows us to control the robot movements and to compare the Bayesian and MCL approaches. It simulates the movement orders given to the robot platform, adding a Gaussian noise both in translation and rotation. In addition, it simulates the images that the robot obtains through its local camera. In particular, for a certain environment, it computes the ideal image the robot would perceive given the map and the current robot position and orientation.

Simulated images are arrays of 80 pixels, reflecting the presence of a beacon, or not, and specifying its colour. The images are computed using a simple sensor model for the camera: a cone with certain scope and depth (right picture on Fig. 1). Two random sensor noises are added to ideal images before delivering them to the localization algorithm: offset (P_{offset}) and mutation ($P_{mutation}$). P_{offset} adds offset noise that may displace the image a random number of pixels. $P_{mutation}$ noise adds false positives and false negatives, flipping the pixel value with certain probability. Noise probabilities, their amplitude and the sensor model can be easily changed at will.

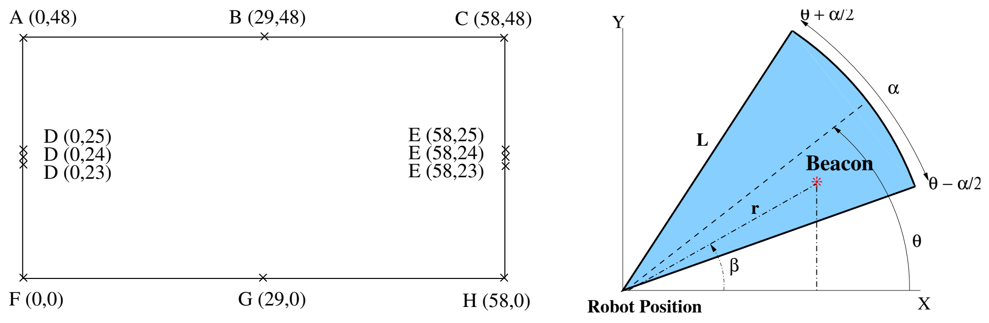

Fig. 1. Sample map provided to the simulator (left) and camera model used (right)

Right picture of Fig. 1 shows the calculations to check if a beacon is in the visual field of the camera. Given a visual field α and a scope L, the beacon B is in the visual field of the camera if $r <$ L and $\beta \in [\theta - \frac{\alpha}{2}, \theta + \frac{\alpha}{2}]$. The corresponding pixel is computed following:

$$pixel = 80 * \frac{((\theta + \alpha/2) - \beta)}{\alpha} \text{ for } \beta \leq (\theta + (\alpha/2)) \tag{5}$$

$$pixel = 80 - \frac{80 * (\beta - (\theta - \alpha/2))}{\alpha} \text{ for } \beta > (\theta + (\alpha/2)) \tag{6}$$

The environment is provided to the simulator through a map file which describes the white lines of the field and the position of the beacons and their colours. The size, shape of the field and the number of beacons and their locations, can be easily changed by modifying the map file.

4 Experiments

Bayesian localization was implemented as shown in section 2, and the behaviour of the algorithm was tested over the simulator. Typical localization error and execution time were measured. Figure 2 represents a typical run using the RoboCup map shown in the left part of figure 1 (real dimension $290cmx240cm$). The map was tessellated in 58x48 cells, and the angle discretized in 360 values, having $5cmx5cmx1°$ cells. Therefore, we had 58x48x360=1,002,240 possible poses where the robot could be located, held in what we called *probability cubes*. For each possible 3D cell the algorithm stores and updates its likelihood given the set of observations and motor actions.

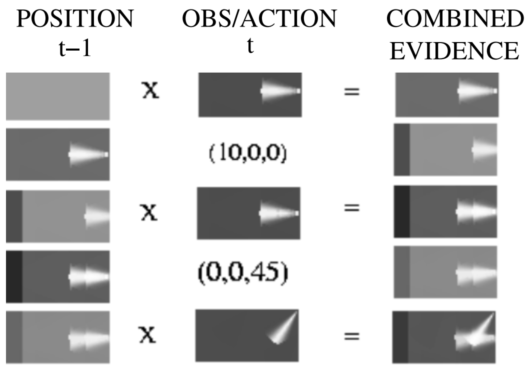

Fig. 2. Probability accumulation using Bayes's rule

Such likelihood is shown on figure 2 in a greyscale, the darker the cell, the lower its probability. For the sake of clarity we haven't shown the probability cube, but only the slot of the cube for the actual orientation of the robot in each iteration. First, the left column shows the evidence stored at $t - 1$. The central column indicates the sensor information or the actuation that takes place in t. The right column results as the combination of the prior ones, and represents the state of the robot's belief after incorporating sensor data or making a move (translation and/or rotation); it becomes the left column in the following iteration.

The top left slot represents the initial estimation. All the cells have the same likelihood. After the first image, positions compatible with such observation rise in probability. As can be seen in top right slot, one cone includes all the compatible cells. After that, a robot forward movement of 10 cells causes the corresponding displacement of the robot's evidence, as can be noted in the second row of figure 2. After the 45° turn shown in the fourth row, the robot sees the C-beacon, which let it discriminate the most likely cells fusing with the prior evidence, as displayed in the bottom right picture.

Typically, the Bayesian algorithm locates the robot in 2-4 iterations, which takes 24-48 seconds long in a Pentium-III at 1.1 GHz. In addition, Bayesian localization has proved to be very robust to actuation and sensor errors. Combining all noises, which resembles the real scenario, it copes with 30% error motor commands, $P_{offset} < 0.2$ and $P_{mutation} < 0.001$. In such realistic setting, it delivers localization errors smaller than 2.5cm in x, y and 5° in orientation.

In addition, multiple experiments were carried out in order to test the algorithm in many scenarios, and to study the effect of different parameters in its performance. It has proved robustness to different observation model and diverse maps. For instance a map similar to the Robocup one was provided, but changing the colour of the beacons to be bottom-up symmetrical. In such scenario the algorithm managed to locate the robot in the same number of iterations, but the differences in likelihood among the cells were smaller.

The number of beacons was incremented and reduced to test its effect on the performance. The intuition was confirmed: the more beacons, the shorter it takes to locate the robot. The Bayesian algorithm degrades gracefully when the number of beacons is decreased. It was also confirmed that the localization improvement after a new observation depends more on the discriminative power of the last image than on the previous history of the cells.

Montecarlo localization has been implemented using local vision only. Instead of computing the probability of all plausible poses, a population of samples evolves in time as new images are collected. The samples are relocated using the information from the last image and they converge to the real location after some iterations, as can be seen in the typical run shown in figure 3.

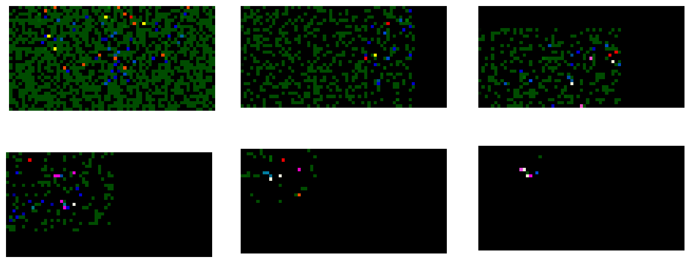

Fig. 3. Samples evolution using Montecarlo localization

For the Robocup map, this convergence can be noted in the dotted line of figure 4. Left and central pictures show the standard deviation of the set of samples around its mean position in X and Y axes, respectively. The right picture shows the standard deviation of the samples in the orientation θ, in particular in the $\cos\theta$ (to avoid considering 1° and 359° as very different angles). As new observations are collected the population reduces its dispersion, converging around certain pose (x, y, θ). In order to check whether that pose is a good one or not, the number of samples which are likely according to the last image v_t is displayed in figure 5. A sample $\widetilde{\varphi}$ is considered likely if its likelihood $p(\widetilde{\varphi}|v_t) > 0.8$. It can be seen that the percentage of likely samples in the population grows as more observations are taken into account.

Typically, 13-16 iterations are required to locate the robot. Nevertheless, those iterations are much faster than the Bayesian ones, as they take only 4 seconds long in the same machine. Regarding noise, Montecarlo localization has proved to be more sensitive to actuation and sensor errors than the Bayesian algorithm. Combining all noises, which resembles the real scenario, the algorithm tolerates 10% error in motor commands, $P_{offset} < 0.2$ and $P_{mutation} < 0.001$. For those settings in the RoboCup map shown in figure 1 it provides real location of the robot with $10cm$ maximum deviation, and 0.12 in $\cos\theta$. The bigger the noise, the further from real position is the mean value in the final sample set, and fewer samples fall close to it.

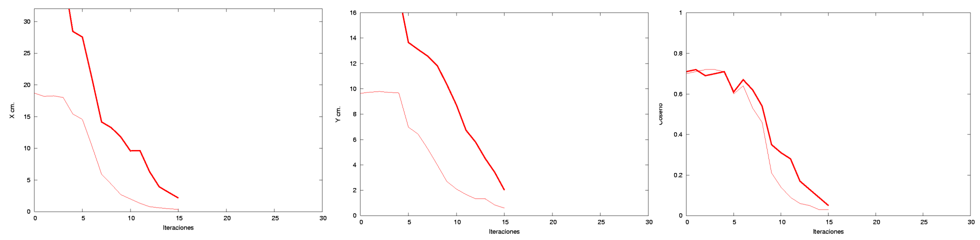

Fig. 4. Typical deviation of samples in x, y and $\cos\theta$

A meaningful parameter for the Montecarlo algorithm is the movement error introduced to shift a little bit the samples after every ideal motor command. It helps to avoid stall and to cope with sensor and motor errors. The bigger this random movement, the more stable the algorithm is, although worse resolution is achieved. The best results were obtained with a random movement of 15%.

To study how well the Montecarlo algorithm scales up, we tested over a bigger map, a $580cmx480cm$. The evolution of the samples is displayed in the continuous line of figures 4 and 5. Initial deviations are naturally bigger as the samples are spread over a larger area. Afterwards, the population gradually converges to some position. After the same number of iterations the localization error was bigger than the one obtained in the regular map in absolute terms, but the ratio error/map size kept constant.

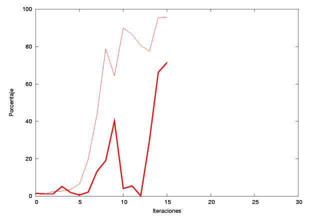

Fig. 5. Number of likely samples

Several tests were carried out to study the effect of other different parameters in the performance of the algorithm. For instance, the probabilistic sensor model was tuned to $e^{-d^2/256}$. The algorithm is very sensitive to this model, and other sharper or softer functions didn't work. In addition more beacons or more samples don't always help convergence or better resolution. A minimum number of them has been identified in both cases, so under 500 samples or 12 beacons, this algorithm doesn't get good position guesses.

5 Conclusions

Two probabilistic localization methods, Bayesian and MonteCarlo, have been implemented and tested over a developed simulator for the Robocup environment. They only use the information provided by robot encoders and the images obtained through its local vision. The aim of this work was to test both methods before their implementation on real robots endowed with camera like the EyeBot and the Aibo platforms.

First, a simulator has been developed. It simulates the motor commands and image observations, adding noise to emulate real uncertainty. A simple cone model was used to generate simulated one-dimensional images. Noise probabilities, their amplitude and the sensor model can be easily changed at will. The environment, including colour beacons, is provided to the simulator through a map file.

Secondly, a Bayesian localization algorithm has been implemented discretizing the official 290x240cm playground into 58x48x360 cells of $5cmx5cmx1^{\circ}$. This cell size is similar to other approaches to the same scenario. This algorithm has proved to be very robust to model parameters, motor and sensor errors, which makes it suitable to cope with uncertainty in real sensors and actuators. Despite its good resolution it doesn't scale up to larger environments. Because it computes the probability of all plausible locations, the processing time grows exponentially with the environment size. Such processing greediness prohibits its implementation on board a robot, typically endowed with relatively slow processors.

Thirdly, Montecarlo localization algorithm has also been implemented and tested in the same environments. Compared to the Bayesian approach it speeds up the localization process, making it easier to implement on board the robot and for bigger environments. In addition, this method doesn't require the tessellation of the space, and so it potentially offers higher resolution than Bayesian localization, when its parameters are tuned properly. Its main drawback is the sensitivity to sensor and motor errors and its own parameters. A fine tuning is required to get a robust behaviour for this algorithm.

We are working in implementing the Montecarlo algorithm on board an Aibo robot, which lacks of good odometry and must rely on its camera for localization. Another task we are working on, is the use of these algorithms with other "sensors" like the wireless network cards; they measure the energy from different access points at the laptop and can be used to estimate its position.

References

[Buschka00] Buschka, P., Saffiotti, A., Wasik, Z.:
Fuzzy Landmark-Based Localization for a Legged Robot.
Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS) Japan, 2000. pp 1205-1210.

[Dellaert99] Dellaert, F., Burgard, W., Fox, D., Thrun, S.:
Using the condensation algorithm for robust, vision-based mobile robot

localization.
Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR) 1999.

[Fox99a] Fox, D., Burgard, W., Dellaert, F., Thrun, S.:
MCL Localization: Efficient position estimation for mobile robots.
In Proc. of the Nat. Conf. on Artificial Intelligence (AAAI) 1999.

[Fox99b] Fox, D., Burgard, W., Thrun, S.:
Markov Localization for Mobile Robots in Dynamic Environments.
Journal of Artificial Intelligence Research 11 (1999) 391-427.

[Fox00] Fox, D., Thrun, S., Burgard, W., Dellaert, F.:
Particle Filters for Mobile Robot Localization.
In A. Doucet, N. de Freitas and N. Gordon, editors, Sequential Monte Carlo Methods in Practice.
Springer Verlag, New York 2000.

[González96] González, J., Ollero, A.:
Estimación de la posición de un Robot Móvil.
Informática y Automática Vol. 29-4/1996.

[Gutmann02] Gutmman, J-S., Fox, D.:
An Experimental Comparison of Localization Methods Continued.
Proc. IEEE/RSJ Int. Conf. on Inteligent Robots and Systems (IROS), 2002.

[Isard98] Isard, M., Blake, A.:
CONDENSATION - conditional density propagation for visual tracking.
Int. Journal of Computer Vision, 1998.

[Kyriy02] Evgeni Kyriy, Martin Buehler:
Three-state Extended Kalman Filter for Mobile Robot Localization
Technichal Report, 2002.

[Margaritis98] Margaritis, D., Thrun, S.:
Learning to Locate an Object in 3D Space from a sequence of images.
Proc. Int. Conf. on Machine Learning (1998) 332–340.

[Montemerlo02] Montemerlo, M., Thrun, S., Whittaker, W.:
Conditional Particle Filters for Simultaneous Mobile Robot Localization and People-Tracking.
IEEE Int. Conf. on Robotics and Automation (ICRA) 2002.

[Sáez02] Sáez, J.M., Escolano, F.:
Localización global en mapas 3D basada en filtros de partículas.
Proc. of III Workshop de Agentes Físicos, Murcia March 2002.

[Simmons95] Simmons, R., Koeing, S.:
Probabilistic Navigation in Partially Observable Environments.
Proc. of the Int. Joint Conference on Artificial Intelligence (IJCAI), July 1995.

[Thrun00a] Thrun, S.:
Probabilistic Algorithms in Robotics.
AI Magazine, 21(4):93-109, April 2000.

[Thrun01] Thrun, S., Fox, D., Burgard, W., Dellaert, F.:
Robust Montecarlo Localization for Mobile Robots.
Artificial Intelligence Journal, 2001.

[Welch02] Welch, G., Bishop, G.:
An Introduction to the Kalman Filter.
UNC-Chapel Hill, TR 95-041, March 11, 2002.