Actas do Encontro Científico
3º Festival Nacional de Robótica - ROBOTICA2003
Lisboa, 9 de Maio de 2003.

MARKOV LOCALIZATION IN THE ROBOCUP SIMULATION LEAGUE [1]

Carla Penedo, João Pavão, Pedro Lima, M. I. Ribeiro

Instituto de Sistemas e Robótica
Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa, Portugal
{ccfp,jpp}@rnl.ist.utl.pt, {pal,mir}@isr.ist.utl.pt

Abstract: For mobile robots, localization is the process of updating the pose of a robot, given information about its environment and the history of its sensor readings. This paper describes an implementation of the Markov localization method using a probability distribution across a fine-grained grid of robot poses to globally localize a robot even in the presence of noise. In particular, we applied this technique to self-localize a soccer player in the RoboCup Simulation League. This simulation features a highly dynamic environment and inaccurate sensor readings similar to real-world situations. We provide an experimental analysis of this implementation and show results indicating that the robot is able to remain relatively well localized in terms of position.

Keywords: Markov Localization, Belief, Grid-based Representation, RoboCup Simulation League.

1. INTRODUCTION

A mobile robot must know where it is in order to autonomously operate in its environment. The ability to maintain a reasonable estimate of its location with respect to its environment is a key problem in mobile robotics and many distinct localization techniques have emerged in recent years.

In the context of mobile robots, the general problem of localization can be stated as follows: given a model of the environment such as a grid-based geometric description of landmarks or a topological map of the environment, the aim is to estimate the location of the robot within the environment based on observations. These observations typically consist of a mixture of odometric information about the robot movements and information obtained from the robot proximity sensors or cameras (Fox 1998).

In this paper we followed a Markov localization approach that belongs to the family of the global localization methods. Our implementation assumes that initially, the robot (*soccer player*) is given a map of its environment, but it does not know where it is. Hence, it has to solve a difficult localization problem, which is to estimate its position from scratch. The system described herein follows a position probability grid approach which provides accurate position estimates in a much broader range of environments. In fact, it allows to integrate raw (proximity) sensor readings instead of only depending on abstract features.

The remainder of this paper is organized as follows. In the next section we introduce the basic idea behind Markov localization and an imple-

[1] Work supported by the FCT "Programa Operacional Sociedade de Informação (POSI)" in the frame of QCA III.

mentation of its general scheme. In Section 3 we present the environment in which the localization technique is developed. To cope with the huge state space, several optimizations were taken into account. They are explained in Section 4. This section also describes how our implementation of Markov Localization was integrated into the global architecture of the RoboCup simulation system. The experimental results are shown in Section 5. Finally, conclusions are drawn in Section 6.

2. MARKOV LOCALIZATION

At any point in time, Markov localization maintains a position probability density (*belief*) over the entire configuration space of the robot based on an incoming stream of sensor data (observations) and an outcome of actions. This probability framework employs *multi-modal* distributions for the robot belief enabling the representation of ambiguous situations by considering multiple hypotheses in parallel. Such feature overcomes the main disadvantage of *local* techniques as *Kalman filtering* which rests on the restrictive assumption that the position of the robot can be modelled by a uni-modal Gaussian distribution.

2.1 Basics

Most techniques to position estimation follow a probabilistic approach to deal with the uncertainty inherent to the sensor data. In the context of Markov localization, the robot position is modelled by a random variable L_t representing the robot true location (position and orientation) at time t. The state space of this variable contains all the locations considered for location estimation. The robot belief at time t, more precisely, the probability (density) that it assigns to the possibility that its location at time t is l, is denoted by $Bel(L_t = l)$. This belief represents a probability distribution over the entire space of L. Here, l is a location in $\langle x, y, \theta \rangle$ where x and y are Cartesian coordinates and θ is the robot orientation.

The task of Markov localization is to update the belief $Bel(L_t = l)$ for any location l in the environment as the robot moves and senses by estimating the posterior distribution over L_t conditioned on all available data. The computation of such a conditional probability grows exponentially with the number of conditioning variables. Therefore, two crucial independence assumptions were made in order to efficiently update the state variable L:

i) *Independence of actions* – knowledge about the position and the motion command executed at time $(t-1)$ is sufficient to predict the localization of the robot at time t, meaning that all actions and positions prior to $(t-1)$ provide no additional information about its current position. This is known as the *Markov assumption*.

ii) *Independence of perceptions* – perceptions received at time t, revealing what the robot sees, depend only on the state of the world at that instant, L_t (Fox 1998).

2.2 Algorithm

The state represented by L_t is updated upon robot motion (odometry readings from wheel revolution count) and upon the arrival of sensor measurements. Thus, localization consists of determining the robot belief of being at a location l conditioned on all previous actions a_n and percepts s_n $(n = 1 \ldots N)$ by assuming independence between them (see (1)). Here, d is a set that comprises all available sensor data (percepts and actions) (Fox 1998).

$$Bel(L_t = l) = P(L_t = l \mid d_{1,\ldots,t}) \qquad (1)$$

The general belief of being at location l at time t by applying Bayes' rule can be described by (2).

$$Bel(L_t = l) = \frac{P(s_t \mid L_t = l, d_{1,\ldots,t-1}) P(L_t = l \mid d_{1,\ldots,t-1})}{P(s_t \mid d_{1,\ldots,t-1})} \qquad (2)$$

In practice, it is too difficult to determine the joint effect of all sensor and position integration readings; instead, a recursive approximation is assumed. The different terms of this equation are computed in a loop that unfolds in two different steps: *update phase*, when the most recent data is a sensor measurement, and *prediction phase* if it is an odometry reading.

Prediction Phase In (2), $P(L_t = l \mid d_{1,\ldots,t-1})$ can be written as $P(L_t = l \mid a_{1,\ldots,t-1}, s_{1,\ldots,t-1})$, representing the probability of the robot being at pose l *after* action a_{t-1} has been executed and *before* any sensor measurement at time t has been perceived. Conditioning this term on the previous state, L_{t-1}, and considering Markov assumption of independence of actions yields (3).

$$P(L_t = l \mid a_{1,\ldots,t-1}, s_{1,\ldots,t-1}) = \sum_{l'} \big(P(L_t = l \mid L_{t-1} = l', a_{t-1}) \cdot Bel(Lt - 1 = l') \big) \qquad (3)$$

The term $P(L_t = l \mid L_{t-1} = l', a_{t-1})$ is also called *motion model* as it reflects the influence of the performed actions in updating the state. It describes the probability that action a_{t-1}, when executed at l', drives the robot to a different position l. It is

characterized by a normal distribution (4) where $\sigma(a_{t-1})$ is the standard deviation considered given (a_{t-1}).

$$P(L_t = l \mid L_{t-1} = l', a_{t-1}) = \frac{1}{\sigma(a_{t-1})\sqrt{2\pi}}\, e^{-\frac{(\mid l - l' \mid - a_{t-1})^2}{2\sigma^2(a_{t-1})}} \quad (4)$$

In most applications of Markov localization such actions are measured by the robot wheel encoders. However, the integration of incremental motion over time (odometry) leads inevitably to the accumulation of errors increasing the uncertainty in the robot pose. Typically, the inherent errors are modelled as a normal distribution (Gaussian) centered at the predicted position of the robot.

Update Phase The term $P(s_t \mid L_t = l, d_{1,\ldots,t-1})$ in (2) can be simplified to $P(s_t \mid L_t = l)$ by the Markov assumption of independence of perceptions. It denotes the probability of measuring s_t if the robot is at location l and is often referred to as the *sensor model*. $P(s_t \mid L_t = l)$ is also equivalent to $P(s_t \mid l)$ since the model of the sensor is assumed to be static. The perception s_t can be a distance measure or an abstract feature observed in the environment. In our case, s_t will be a distance to a known obstacle, more precisely, the flags on the soccer field (see Section 3.1). Every time a sensor detects an obstacle, the resulting distribution is modelled by a Gaussian with mean equal to the measured distance to this obstacle. One single variance is not sufficient to describe the precision of this mean, since the variance depends on the distance of the closest flag and its angle. So, the outcome of the visual sensor is modelled by a two-dimensional normal distribution

$$P(x) = \frac{1}{2\pi \mid \Sigma \mid^{1/2}}\, e^{-\frac{1}{2}(x-\mu)' \Sigma^{-1} (x-\mu)} \quad (5)$$

where x represents the vector $[d \;\; \theta]'$ of the random variables *distance* and *angle*, which are characterized by the mean μ and the covariance matrix Σ. This matrix can be easily determined by

$$\Sigma = \begin{bmatrix} {\sigma_d}^2 & \rho\sigma_d\sigma_\theta \\ \rho\sigma_d\sigma_\theta & {\sigma_\theta}^2 \end{bmatrix} = \begin{bmatrix} {\sigma_d}^2 & 0 \\ 0 & {\sigma_\theta}^2 \end{bmatrix} \quad (6)$$

The variances σ_d and σ_θ model the uncertainty of the measured distance and angle, reflecting the granularity of the discretization of L, the accuracy of the world model and the precision of the sensor. The correlation coefficient ρ is equal to 0 because the two variables are assumed independent.

Normalization The denominator in (2) is constant, since it does not depend on L_t. It acts as a normalizer parameter α_t ensuring that $Bel(L_t)$ sums up to 1 over all locations:

$$\alpha_t = \frac{1}{P(s_t \mid d_{1,\ldots,t-1})} \quad (7)$$

Initialization $Bel(L_0)$ reflects the initial knowledge: it is centered on the current location when the robot is aware of its starting position; otherwise $Bel(L_0)$ is uniformly distributed to reflect the global uncertainty of the robot. The latter is the case in our work.

2.3 World Model

Our implementation of Markov localization uses a fine-grained geometric discretization to represent the position of the robot (Burgard *et al.* 1997). A position probability grid is three-dimensional, as each possible location l is defined by a tuple $\langle x, y, \theta \rangle$ representing the robot position and orientation.

The principle of the *position probability grid approach* ascribes to each cell of the grid the probability of the robot being located in that cell.

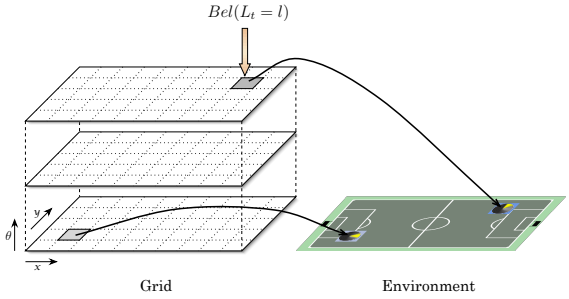

Fig. 1. Transformation of grid coordinates into field coordinates.

Figure 1 illustrates the structure of a position probability grid and the correspondence between grid and field positions in the RoboCup simulation system. Each layer of the grid assigns all possible poses of the robot with the same orientation.

Such a representation has the advantage that it provides an accurate estimation of the location of a robot. However, an evident disadvantage of this method lies in the space required to store the grid. To overcome this drawback, instead of updating all the possible positions of the robot every time it senses or acts we implemented an optimization technique named *Selective Update* (see Section 4.3.1).

3. ROBOCUP SIMULATION LEAGUE

The RoboCup Simulator League is based on the RoboCup Simulator called the *soccer server*, a

physical soccer simulation system. All games are visualized through the *soccer monitor* by displaying the field of the simulator on a computer screen. Each player must be controlled by a completely independent process.

The robotic soccer simulator is an instance of a client/server application in which each client communicates with the server via a UDP socket. Each client is a separate process and connects to the server using a given port through which the messages are transferred. The server receives the requests from the clients (actions they want to perform) and updates the environment accordingly. Furthermore, it supplies the sensory information (aural, visual and body) to the players (see Section 4.1).

A RoboCup agent has three distinct sensors: aural, visual and body. The aural sensor is used for communication among the several agents (referee, coaches and other players). The visual sensor provides information about the field, like the distance and direction to objects in the player's current field of view. The farther the sensed object is from the agent the less precise the information received by these two sensors will be. Finally, the body sensor detects the "physical" status of the player (stamina, speed and neck angle) (Noda *et al.* 2001). In the sequel we will explore the body and visual sensors as they are fundamental to implement the Markov localization algorithm.

3.1 Visual Sensor Model

Each player automatically receives a message from the server every 150ms describing what it sees. The player's normal view cone, i.e., its visible sector, is 90 degrees wide.

Visual information arrives from the server in the following basic format:

(**see** *ObjName Distance Direction DistChng DirChng BodyDir HeadDir*)

where *ObjName* stands for the identifier of the seen object (ball, players, flags, goal poles or boundary lines) and the other parameters give more detailed information about it.

Our implementation of the Markov localization method takes advantage of the several existent flags (represented by dots in Figure 2) placed around the field. These marks can be easily recognized from the robot sensory input since they have a fixed and known position relative to which a robot can localize itself. The algorithm allows the computation of $P(s_t \mid l)$ based only on the distance to the closest flag sensed as it is the one that provides the most accurate information (precision decreases with the distance).

Visual sensing provides a tremendous amount of information about a robot environment and even when used as the sole source of sensor data it can result in a reasonable approximation to determine the robot location.

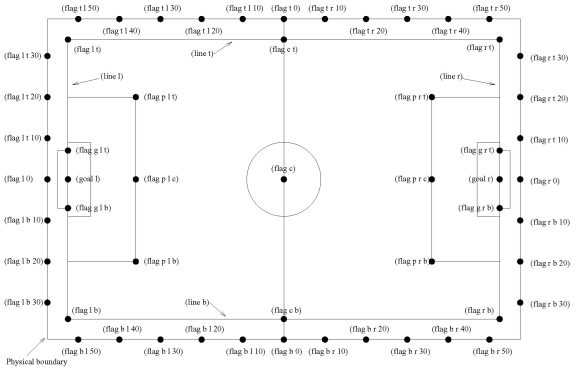

Fig. 2. Flags and lines in the RoboCup Simulation League (from (Noda et al. 2001, p.29)).

3.2 Motion Model

The body sensor reports automatically the "physical" status of the player (such as stamina and speed) every 100ms. The robot speed is then integrated to determine the distance it has travelled from its last known position.

Despite odometry readings being only capable of providing short-term accuracy, they are very useful in complementing the visual data. In fact, as the visual information arrives every 150ms, odometry is crucial in order to update the robots position in the time between visual sensing.

3.3 Noise Models

The server adds some white noise to all the players' sensory inputs and actions to simulate the type of restrictions one would expect if the agents were real robots.

3.3.1. Visual Sensor Noise Model The data values sent from the server to the visual sensor are quantized and, consequently, the players cannot know the exact position of the objects. In the case of flags and field lines, the distance value is quantized in the following manner:

$$d' = \text{Quant}(\exp(\text{Quant}(\log(d), 0.01)), 0.1) \quad (8)$$

where d and d' are the exact and quantized distances, respectively, and

$$\text{Quant}(V, Q) = \lceil V/Q \rceil \cdot Q \quad (9)$$

For example, when a flag or line is about 10m away, the maximum noise is about 10cm since the quantize step of distance for landmarks is 0.01.

The maximum error associated to the measured angles is 0.5° as this value is always rounded to the nearest integer.

3.3.2. Motion Noise Model To better mimic the movement of objects in the real world, a small amount of noise is added to the movement of objects and parameters of commands. This noise is completely random and uniformly distributed over the range $[-rmax, rmax]$, where $rmax$ is a parameter that depends on the velocity of the object as follows:

$$rmax = \text{rand} \cdot v_{obj}^{t} \tag{10}$$

where rand assumes different values depending on the type of object (player or ball).

4. IMPLEMENTATION

4.1 Distributed Architecture

The RoboCup simulation system is comprised by several independent processes that are not necessarily running on the same host machine. This is basically a distributed system controlled by a central process — the soccer server — that runs the whole simulation, sending and receiving all the relevant information to and from all the client programs connected to it. There are several kinds of client programs that can connect to the central soccer server. However, in this paper we will only deal with soccer player and soccer monitor clients (see Figure 3).

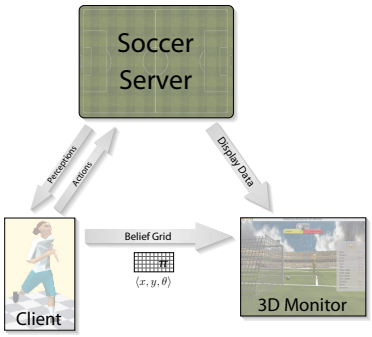

Fig. 3. Global architecture of the distributed system.

The soccer player is an agent program that communicates with the soccer server receiving perceptions about the environment and sending the actions it wants to perform. The soccer monitor connects to the soccer server and receives all the information necessary to correctly display the state of the game and of all the players on field. A third connection was added specifically for this project that enables us to visualize the state of the Markov localization algorithm running at the heart of the soccer playing agent. The soccer monitor connects to the soccer player in order to receive a grid containing the beliefs of the Markov localization algorithm.

4.2 3D Monitor

The soccer monitor used in the current project was built upon the RoboCup Advanced 3D Monitor (RA3DM) (Penedo et al. 2002), a real-time 3D monitor for the RoboCup Simulation League which we had previously developed. This monitor can be easily adapted to fit the visualization needs at hand. For this project, we changed the communications routines and the soccer field drawing routines in the following manner:

Communications The third connection added to the system enables us to display the state of the probability grid directly on top of the soccer field. The RA3DM connects to a special port on the soccer player program that is to be monitored. This soccer player — that is running a Markov localization algorithm — will then start to dump the contents of its beliefs grid to RA3DM via a dedicated UDP socket.

Drawing The drawing routines were extended so that the contents of the grid that is received could be displayed on the soccer field. Each grid cell is drawn in a shade of blue (instead of the usual green used for the field) whose brightness is directly proportional to the belief that the robot is in that cell (see Figure 1). Moreover, the cell with the highest belief at each time step is distinguished with a red color. However, only cells with a belief value above a certain threshold are drawn so that the field doesn't become too cluttered (see Section 4.3.2). Each one of these cells also features a bright circular sector that indicates the range of orientations with the highest belief.

Figure 4 illustrates the cells with the highest probability of the Markov grid of a single player, as seen through RA3DM.

4.3 Optimizations

Due to the large volume of information that a grid-based Markov localization algorithm has to process and send across a network, some optimizations had to be integrated for the feasibility and efficiency of our implementation. So, we considered two different ways to increase the algorithm performance that will be discussed in the upcoming sections. One of them is internal to the localization algorithm and drastically reduces the number of cells to process (Section 4.3.1). The second one optimizes the amount of data to be

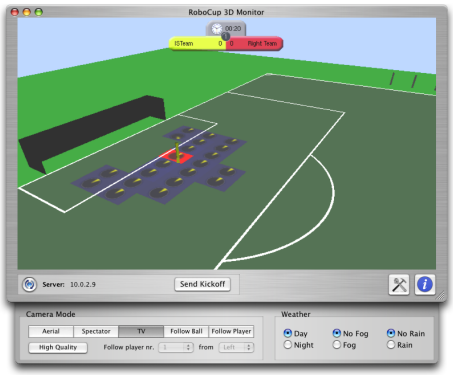

Fig. 4. Screen capture of a RA3DM window displaying the state of the Markov probability grid of the goalie.

transmitted over the network link used to send the grid to the soccer monitor (Section 4.3.2).

4.3.1. Selective Update To obtain the most accurate possible estimates of the robot position, our Markov localization uses a fine-grained discretization of the state space. The three-dimensional grid depicted in Figure 5 represents L, where each layer contains the probability of all possible poses with the same orientation. This kind of grid obviously results in a huge state space that has to be maintained and updated throughout the execution of the localization algorithm. The basic Markov localization algorithm updates each one of the elements of the grid at each iteration, i.e., at each atomic movement of the robot corresponding to a sensor or odometric reading. This makes the algorithm unusable in practical applications because it is not adequate for real-time environments, as current computer speeds cannot deal with such an amount of information manipulation at this speed.

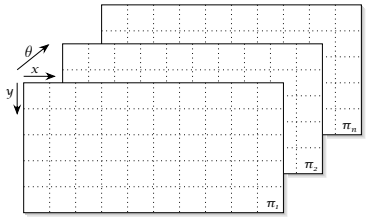

Fig. 5. Grid-based representation of the partitioned state space.

To resolve this issue, we have implemented a technique (presented in (Fox et al. 1999, pp.409–411)) that largely improves the efficiency of the computation of L. It enables the algorithm to be applied on-line in a real time environment by only updating the cells which seem to be relevant — the ones that have their Belief value above a certain threshold. The key idea of this method is to take advantage of the fact that briefly after the localization begins, the probability of the position estimation rapidly concentrates around the true position of the robot and decreases on the other cells. By knowing this in advance, we can then chose to not update cells that are unlikely to be close to the true position and, consequently, have a very low probability value of $Bel(L_t = l)$.

We define a threshold ε and only the cells l for which $Bel(L_t = l) > \varepsilon$ are updated. In order to maintain a consistent density over the whole state space, the cells with a value below the threshold ε are updated using an approximation of $P(s_t \mid l)$ by the a priori probability of measuring s_t, which is determined by averaging $P(s_t \mid l)$ over all possible locations of the robot:

$$\widetilde{P}(s_t) = \sum_l P(s_t \mid l)\, P(l) \qquad (11)$$

As $\widetilde{P}(s_t)$ is independent of the current belief state, it is built beforehand upon program initialization. We build a table of values of $\widetilde{P}(s_t)$ for a discretized range of values of s_t. This is a very heavy and time consuming calculation since the table has dozens of entries, each of them being the result of the average over all cells of the grid calculated through (11). Each time a sensor measurement s_t is received, the cells update is done according to

$$Bel(L_t{=}l) \leftarrow \begin{cases} \alpha_t \cdot P(s_t|l) \cdot Bel(L_{t-1}{=}l) & \text{if } Bel(L_{t-1}{=}l) > \varepsilon \\ \alpha_t \cdot \widetilde{P}(s_t) \cdot Bel(L_{t-1}{=}l) & \text{otherwise} \end{cases} \qquad (12)$$

To focus the computation on the relevant regions of the state space, we partition it into several parts $\pi_1, \pi_2, \ldots, \pi_n$ which can be active or passive. A part π_i is active at time t when it contains at least one location l with probability above the threshold ε. Otherwise, it is passive.

When a partition π_i becomes passive, a variable $\beta_i(t)$ is used to accumulate the factors in (12) for this partition until it becomes active again. The variable $\beta_i(t)$ is initialized to 1 and subsequently updated as:

$$\beta_i(t+1) = \alpha_t \cdot \widetilde{P}(s_t) \cdot \beta_i(t) \qquad (13)$$

When the partition π_i becomes active again, we multiply the probabilities of all its cells by the value of $\beta_i(t)$ that has been accumulated so far. We also keep track of the robot motion since the time the partition became passive. The accumulated motion is incorporated upon its activation.

$P_i^{\max}$ is the maximum probability (or belief) value of partition π_i and is calculated by

$$P_i^{\max} = \max_j Bel(L_t = l_j), \quad l_j \in \pi_i \qquad (14)$$

This value is stored when the partition changes its state from active to passive. To determine if the

partition becomes active again it suffices to verify whether condition $P_i^{\max} \cdot \beta_i(t) > \varepsilon$ is met.

In our implementation, each partition consists of all the possible robot locations that have equal orientations. This enables us to mirror the changes in the robot orientation in a highly efficient manner. In fact, when the robot turns an angle γ, we only need to change the orientation to which every partition is associated (by γ) in order to reflect such a rotation.

4.3.2. Selective Sending A regular match in the RoboCup Simulation League usually involves more than twenty five programs running at the same time distributed among several networked computers. The soccer server delivers perceptions and receives actions from these programs ten times per second. This poses a big burden on the network which can get clogged up pretty easily. By sending the Markov grid several times per second we are loading the network even more. In our implementation this problem was seriously taken into account as we tried to minimize the network bandwidth needed to deliver the probability grid to the soccer monitor. To achieve this, the grid is compressed and coded into a message that only contains information about the cells with belief values above a certain threshold. This message will then be decoded and decompressed upon reception by the monitor, that stores it in a form suitable for a fast display.

5. EXPERIMENTAL RESULTS

Our implementation of the Markov localization algorithm was written in the C language running on the Linux operating system.

We used an angular resolution of 40 degrees in the three experiments described in this section, meaning that we divided the state space into 9 partitions. Moreover, we attempted to collect results using different angular resolutions of 30, 20 and 10 degrees with 12, 18 and 36 partitions, respectively. However, by increasing the number of partitions and, consequently, defining a narrower angle, the resulting errors were too high. This is mainly due to the increasingly higher delay in response times that the Markov algorithm faces when the total number of cells is augmented. In fact, the average time interval between every two consecutive iterations of the Markov localization algorithm for a 40×40 resolution is 0.226s for 9 partitions, 0.512s for 12 partitions and 0.860s for 18 partitions. These delays lead to a heavy loss of odometric and perceptual (visual) information (that are delivered to the player at fixed time intervals), resulting in an increase in the error of the location estimation as time goes by.

Table 1. Error measurements in experiments with different grid resolutions.

Resolution	40×40	60×40	50×50
Cell Area (m)	2.63×1.70	1.75×1.70	2.10×1.36
Total Cells	14 400	21 600	22 500
Error pos_{mean} (m)	2.445	3.767	17.003
Error pos_{max} (m)	62.242	62.511	62.537
Error θ_{mean} (°)	21.812	20.185	100.765
Error θ_{max} (°)	108.000	159.000	176.000
$Bel^{\max}{}_{mean}$	0.157	0.041	0.007

The total number of grid cells and the dimensions of each cell are shown in Table 1 for the three resolutions that were tested using 9 partitions. The error measurements in position estimation were obtained by calculating the distance between real and estimated positions (in meters), while the orientation error was calculated as the difference between the real and the estimated orientations (in degrees). We present the mean and maximum errors (assumed Gaussian) for each type of measurement. Additionally, $Bel^{\max}{}_{mean}$ represents the mean value of the belief on the cell of the estimated location over time, i.e., the cell with the highest confidence.

Figure 6 plots the evolution of the position error for some of the best localization results achieved at each waypoint of the path followed by the player. The average sample rates are 4.425 Hz, 2.387 Hz and 0.372 Hz for the 40×40, 60×40 and 50×50 resolutions, respectively.

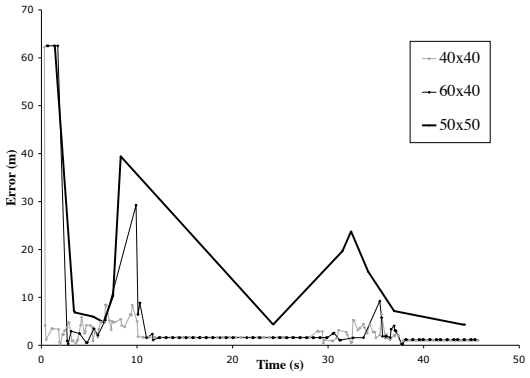

Fig. 6. Position error measured with different grid resolutions.

By the analysis of the results we verify that both types of errors (position and orientation) usually increase significantly when the grid resolution is higher. This was somewhat expected, as a growing number of grid cells requires more calculations and the system isn't able to keep pace with the RoboCup Simulator in this situation. Peaks in localization errors occur mainly in two different situations: at the beginning of the game, when the robot is self-localizing for the first time — bootstrap problem — which is particularly obvious in Figure 6, and when the robot scores a goal and is moved instantly to its home position —

kidnapped robot problem. However, such errors aren't problematic since the robot easily recovers from them (for example, after 1s it was able to localize itself with a position error of 3.46m using a grid of 40×40).

Generally, the Markov localization method keeps the robot position error bound within very reasonable limits. In experiments using a grid of 40×40 cells (2.63m×1.70m), the average error of 2.4450m that was obtained is really good. Such results were possible due to the high volume and diversity of the received visual information and the fact that, throughout all experiments, the player was always alone in the field. This isn't what really happens in a normal soccer game, when the flags used in the localization algorithm may be obstructed by the other players in the field, resulting in a decrease in the available perceptual information that can be used.

As for the orientation estimation, the results obtained weren't so good, but they were somewhat expected given the low resolution used in our grids — each partition represents an interval of 40 degrees. The behavior of the player used in the experiments (the decision making module) also contributed to these results, as it executed lots of turns in a short period of time. This issue doesn't allow the estimation of orientation to stabilize in a more desirable manner. We also tried to change the behavior of the tested player in order to make it move and turn more gently, but then the received perceptual information was less diverse and led to even worse results.

The real path followed by the player and the corresponding Markov estimation for the first 8.7s of a game using a grid of 40×40 (the same experiment described earlier) is shown in Figure 7. Its basic behavior consisted in moving forward with the ball to the opponent's goal and scoring a goal. Although it isn't possible to discern which estimation corresponds to a certain position (real) at a given moment, it provides an interesting overview of the localization process.

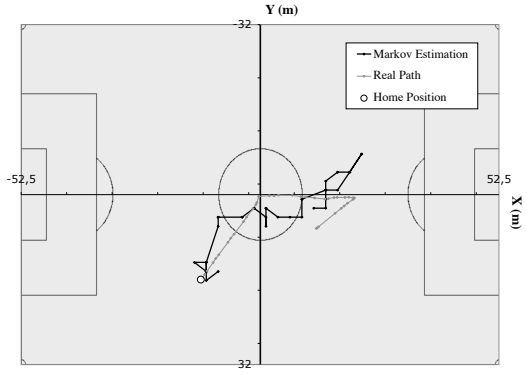

Fig. 7. Real path and Markov estimation.

6. CONCLUSIONS

In this paper, we applied the Markov localization method to self-localize a soccer player in the RoboCup Simulation League. The results obtained concerning the global position estimation were very reasonable. However, we must stress that in some tests, even considering the same resolution, the error was slightly higher. Such discrepancy takes place due to the completely random nature of the noise added by the soccer server.

The main problem when implementing this algorithm is the huge state space that is required. We have also implemented some optimizations techniques in order to cope with the dynamic environment of RoboCup. Nevertheless, this solves the problem just to a certain extent, since we still have limitations when choosing adequate grid dimensions.

Odometry readings are usually quite helpful in implementations of localization methods since they are always available, as opposed to the sparser availability of visual information. However, in the RoboCup simulator odometry readings can be computed every 100ms and visual information is received every 150ms. This way, odometry is only capable of slightly improving the estimates between two perceptual data samples.

ACKNOWLEDGMENTS

This work was produced with the support of Prof. Luis Custódio whose help and guidance was a valuable contribution.

REFERENCES

Burgard, W., D. Fox and D. Henning (1997). Fast grid-based position tracking for mobile robots. In: KI - Kunstliche Intelligenz. pp. 289–300.

Fox, D. (1998). Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation. PhD thesis. University of Bonn.

Fox, D., W. Burgard and S. Thrun (1999). Markov localization for mobile robots in dynamic environments. Journal of Artificial Intelligence Research **11**, 391–427.

Noda, I., M. Chen, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, O. Obst, P. Riley, T. Steffens, Y. Wang and X. Yin (2001). RoboCup Soccer Server: Users Manual for Soccer Server Version 7.07 and later.

Penedo, C., J. Pavão and P. Nunes (2002). Robocup advanced 3D monitor. Technical report. Instituto Superior Técnico.