SELF-LOCALIZATION BASED ON KALMAN FILTER AND MONTE CARLO FUSION OF ODOMETRY AND VISUAL INFORMATION

David Cabecinhas, João Nascimento, João Ferreira, Paulo Rosa, Pedro Lima

Instituto de Sistemas e Robótica – Instituto Superior Técnico
Av. Rovisco Pais, 1, 1049-011 Lisboa, PORTUGAL

Abstract: *The main goal of this paper is to describe the application of two self-localization methods, based on Kalman filter and Monte Carlo fusion of odometry and visual information, to omni-directional soccer robots. The two methods are compared, and the usage of several candidates for the robot posture, provided by the vision-based observation step, is discussed. Simulated and real robot results are presented.*

Keywords: self-localization, Kalman filter, Monte Carlo localization, sensor fusion, robot vision.

1. INTRODUCTION AND MOTIVATION

One of the most important problems in mobile robotics is the self-localization of the robot in specific environments. In Robotic Soccer this is highly important because the robot needs to "know" its position on the field, to perform adequately most of its tasks.
This paper introduces Kalman Filter (KF) based localization and MCL methods [2][4][5], taking advantage of them to fuse sensor information, and applying it to an omni-directional robot. By fusing different sensor measurements, such as image features and odometry, the robot should be able to get a better estimate of its location, rather than just using one of the two measurements separately. Furthermore, the KF and the MCL also output a measure of uncertainty of the estimate. This might be useful in some tasks. For instance, the robot should not get close to the lateral lines of the field if the estimate variance is too high, since it could be closer than it would expect.
In the next section, we describe a vision-based self-localization algorithm, introduced in [6] and improved in [3], which determines the robot's posture, and that will be called the "self algorithm" throughout this paper. Finally, some results and conclusions are presented, and a few suggestions for future work on this area are presented.

2. KALMAN FILTER

The Kalman filter block diagram is shown in Fig. 1.

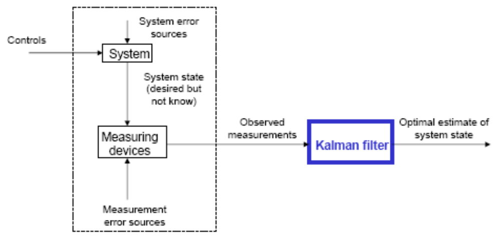
System error sources
Controls
System
System state (desired but not know)
Measuring devices
Observed measurements
Kalman filter
Optimal estimate of system state
Measurement error sources

Figure 1- KF block diagram (reprinted from [1])

The KF is an optimal recursive data processing algorithm, as explained in the sequel. In this paper, only its discrete time version will be used. The KF is the optimal estimator in the minimum mean square error sense, for linear systems driven by Gaussian noise.
The following equations describe a (non-linear) discrete-time dynamic system and its sensors' measurements:

$$x(k+1) = f(x(k), u(k), w(k))$$
$$z(k+1) = h(x(k+1), v(k+1))$$

- x - state
- f - system dynamics
- h - measurement function
- u - controls
- w - system error sources
- v - measurement error sources
- z - observed measurements

What the extended version of the KF does is to estimate the state of the dynamic system, based on its linear model, as it will be shown later on this paper, which can be written as

$$x(k+1) = A_k x(k) + B_k u(k)$$

Where

$$A_k = \frac{\partial}{\partial x} f(x,u,w)\Big|_k \qquad B_k = \frac{\partial}{\partial u} f(x,u,w)\Big|_k$$

The KF algorithm is recursive in the sense that it does not need to process all the previous data to estimate the next state. To estimate the state at k + 1 only the estimate at k, and the measurement in k + 1, are needed.
There are two matrices that define the behaviour of the Kalman filter, expressing the covariance of the measurements errors, and the covariance of the model noise. These are the matrices **R** and **Q**, respectively, and can be seen as adjustment factors of the filter. Formally, they can be defined by:

$$E\left[\begin{pmatrix} w_k \\ v_k \end{pmatrix}\begin{pmatrix} w_k^T & v_k^T \end{pmatrix}\right] = \begin{bmatrix} Q_k & 0 \\ 0 & R_k \end{bmatrix}$$

2.1 Kalman Filter Algorithm

Prediction Cycle Given the state estimate at k, $\hat{x}(k \mid k)$, the predicted state estimate at the next sampling time is:

$$\hat{x}(k+1 \mid k) = A_k \hat{x}(k \mid k) + B_k u_k$$

Furthermore, the conditional covariance is given by:

$$P(k+1 \mid k) = A_k P(k \mid k) A_k^T + G_k Q_k G_k^T$$

It should be noticed that the matrix P is important not only to get a measure of the uncertainty of the state estimate, but also because it will be used to calculate the Kalman gain, in the update (or filtering) cycle.

Filtering Cycle The state estimate is updated with the current measurement in the following way:

$$\begin{aligned}\hat{x}(k+1 \mid k+1) &= \hat{x}(k+1 \mid k) + \\ &+ K(k+1)\left(z_{k+1} - C_{k+1}\hat{x}(k+1 \mid k)\right)\end{aligned}$$

K(k + 1) is the Kalman gain, given by:

$$\begin{aligned}&K(k+1) = \\ &= P(k+1 \mid k) C_{k+1}^T \left[C_{k+1} P(k+1 \mid k) C_{k+1}^T + R_{k+1}\right]^{-1}\end{aligned}$$

In addition, the conditional covariance of the state is:

$$\begin{aligned}&P(k+1 \mid k+1) = P(k+1 \mid k) - \\ &- P(k+1 \mid k) C_{k+1}^T \left[C_{k+1} P(k+1 \mid k) C_{k+1}^T + R_{k+1}^T\right] C_{k+1} P(k+1 \mid k)\end{aligned}$$

The discrete Kalman filter provides an unbiased estimate of the state, and it is a consistent estimator, for linear systems and Gaussian white noise.

3. MONTE CARLO LOCALIZATION

The MCL algorithm [4] is a particle filter combined with probabilistic models of robot perception and motion. The working principle of the MCL algorithm is Bayes filtering, which is the estimation of the state of a dynamical system based on information from sensor measurements. Bayes filters assume that the environment is Markov, that is, past and future data are (conditionally) independent if one knows the current state.
The key idea of Bayes Filtering is to estimate the posterior probability density over the state space conditioned on the data available from sensor measurements. This posterior is usually denoted as *belief.*

$$Bel(x_t) = p(x_t \mid d_{0\ldots t}) \qquad (1)$$

Here x denotes the state, x_t is the state at time t, and $d_{0..t}$ denotes the data starting at time 0 up to time t.
Using the Bayes rule and the Markov assumption (which states that measurements y_t are conditionally independent of past measurements and odometry readings given knowledge of the state x_t) equation (1) can be rewritten as a recursive estimator for Bel(x_t) known as *Bayes Filter*.

$$Bel(x_t) = \eta\, p(y_t \mid x_t) \int p(x_t \mid x_{t-1}, u_{t-1}) Bel(x_{t-1}) dx_{t-1} \qquad (2)$$

where η is a normalizing constant and data d was separated in *perceptual data*, y, and *control/odometry data*, u.
Bayes Filtering is usually known in mobile robotics as Markov Localization. As one can see from equation (2) we need to know three distributions: the initial belief, Bel(x_0) (which is usually an uniform distribution because in the beginning the robot does not know where it is, i. e., the global localization problem), the next state probability or motion model, p($x_t \mid x_{t-1}, u_{t-1}$), and the perceptual likelihood or perceptual model, p($y_t \mid x_t$).

3.1 Particle Filter Implementation

In continuous state space, implementing the belief update equation (2) is not trivial and can become a serious problem in terms of efficiency. Using a grid representation is not practical either because to obtain greater precision one has to use smaller grids, making the calculations computationally heavy and impractical to use for real-time purposes.
To tackle this problem, MCL represents the belief over the state space of x as a set of m weighted samples distributed according to Bel(x).

$$Bel(x) = \{x^{(i)}, p^{(i)}\}_{i=1,\ldots,m} \qquad (3)$$

Here each $x^{(i)}$ represents a sample (state) and $p^{(i)}$ represents its corresponding importance factor. The importance factors sum up to 1 and determine the weight of each sample.
With this approach one can approximate a large range of probability distributions and, once a robot's belief is focused on a subspace of the space of all postures, it is computationally efficient, since they focus their resources on regions of state space with high likelihood. This method proves to be advantageous when compared with grid-representation for the belief, because it is more computationally efficient and accurate.
The MCL algorithm implemented is based on the particle filter with the following proposed distribution to approximate Bel(x_t) via importance sampling:

$$q := p(x_t \mid x_{t-1}, u_{t-1}) Bel(x_{t-1}) \qquad (4)$$

The particle filter is implemented in 4 steps:

1. A state $x^{(i)}_{t-1}$ is sampled from $Bel(x_{t-1})$ according to the discrete distribution defined through the importance factors $p^{(i)}_{t-1}$
2. The sample $x^{(i)}_{t-1}$ and the action u_{t-1} are used to sample $x^{(j)}_t$ from the distribution $p(x_t \mid x_{t-1}, u_{t-1})$.
3. Samples $x^{(j)}_t$ are weighted by the non-normalized importance factors $p(y_t \mid x^{(j)}_t)$, the likelihood of the sample $x^{(j)}_t$ given the measurement y_t.
4. Normalize the new importance factors, so that they sum up to 1.

This procedure implements equation (2) using an approximate sample-based representation. This algorithm is just one possible implementation of the particle filter method. Other sampling schemes exist.
A MCL algorithm was developed for simulation according to these steps. Good results where obtained on the position tracking and global localization problems, when using a monomodal distribution for the sensor model. Not so satisfactory results where obtained for the kidnapped robot problem [5] or when using a multimodal distribution for the sensor model.
This was mainly because the proposal distribution, which is used to generate the samples, places too little samples in regions where the desired posterior $Bel(x_t)$ is large, i.e., after kidnapping the robot the particles will be concentrated mainly around the old robot position and there are not enough particles near the new robot position, where the desired posterior probability is high.
This problem has also been noticed by other authors [4] who noted that the MCL performs poorly when the noise level of the perceptual sensors is too small, because the distribution which is used for sampling is an approximation of the product distribution

$$\frac{p(y_t \mid x_t)\, p(x_t \mid u_{t-1}, x_{t-1}) Bel(x_{t-1})}{p(y_t \mid d_{0\ldots t-1}, u_{t-1})} \quad (5)$$

and not the real one, since sampling from the original distribution is too difficult. When sensors are completely uninformative, i.e., $p(y_t \mid x_t)$ is uniform and (4) equals (5), for low noise sensors, the distribution is usually quite narrow hence MCL has a slow convergence.

3.2 Improved Sampling Method

The solution for this problem is to use an alternative distribution for sampling in order to accommodate highly accurate sensors. This can be done by sampling accordingly to the most recent sensor measurements y_t, which is proposed in [4]. That is, instead of getting a reading from odometry and propagate it to the particles and then sample according to the *belief*, what is done is to back propagate the odometry reading from the pose given by the latest *perceptual* sensor reading, and then sample according to the *belief*. This procedure is a dual of the regular MCL algorithm.
In order to do this x_t is sampled directly from a distribution that is proportional to the perceptual likelihood $p(y_t \mid x_t)$.

$$\bar{q} := \frac{p(y_t \mid x_t)}{\pi(y_t)} \quad \text{with} \quad \pi(y_t) = \int p(y_t \mid x_t)\, dx_t \quad (6)$$

These new generated samples are highly consistent with the most recent sensor measurement but ignorant of the belief $Bel(x_{t-1})$ and the control u_{t-1}.
The alternative proposal distribution can be obtained following these steps:

1. Build a *k*d-tree [7] based on $Bel(x_{t-1})$
2. Generate a set $x^{(i)}_t$ of samples according to the distribution $p(y_t \mid x_t) / \pi(y_t)$
3. For each sample in $x^{(i)}_t$, generate a sample $x^{(i)}_{t-1}$ according to

$$\frac{p(x_t^{(i)} \mid u_{t-1}, x_{t-1})}{\pi(x_t^{(i)} \mid u_{t-1})} \quad (7)$$

where

$$\pi(x_t^{(i)} \mid u_{t-1}) = \int p(x_t^{(i)} \mid u_{t-1}, x_{t-1})\, dx_{t-1} \quad (8)$$

thus back propagating the odometry results.

4. Set the importance factors of each sample to a value proportional to the posterior probability of $x^{(i)}_{t-1}$ under the *k*d-tree that represents $Bel(x_{t-1})$, built in step 1.

It is necessary to build a *k*d-tree because $Bel(x_{t-1})$ is only defined for poses occupied by one or more particles, and the back propagation of the motion model causes the particles to be in poses for which $Bel(x_{t-1})$ does not exist and thus must be approximated.

3.3 Mixture Sampling

Neither distribution alone (the original one and the alternative one) is satisfactory because the first fails if the perceptual likelihood is too peaked and the other because only considers the most recent sensor measurements. The solution is to use a mixture of both proposed distributions:

$$(1-\phi)q + \phi\bar{q} \quad (9)$$

where ϕ (with $0 \geq \phi \geq 1$) denotes the *mixture ratio* between regular and dual MCL.
This mixture proposal distribution gives good results and was the distribution used in the course of this work.

4. APPLICATION TO AN OMNI-DIRECCTIONAL SOCCER ROBOT

Having described the KF and the MCL method, it is now necessary to describe the robot kinematics.

In Figure 2 is shown the kinematic structure of the omni-directional robot with 3 Swedish wheels.

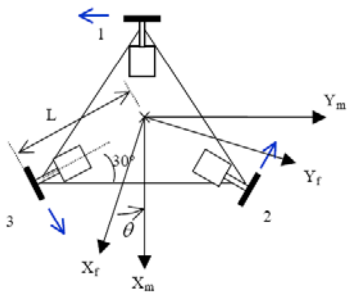

Figure 2 – Omni-directional kinematics structure

The kinematics model can be stated in the following equations, relating the angular velocity of the wheels with the linear and angular velocities of the robot itself.

$$\begin{bmatrix} V_x \\ V_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{\sqrt{3}}r & \frac{1}{\sqrt{3}}r \\ -\frac{2}{3}r & \frac{1}{3}r & \frac{1}{3}r \\ \frac{r}{3L} & \frac{r}{3L} & \frac{r}{3L} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

L is plotted in Figure 2 and *r* is the radius of the wheels. w_i is the angular speed of the i^{th} wheel.
To relate the velocities in the robot frame with the corresponding ones in the world frame, we use the following matrix **J**:

$$\mathbf{J} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This way, the posture of the robot can be determined by integrating the velocity of the robot, if the initial state is known.
At this point, there are several ways to use the odometry in the prediction cycle. For instance, one can use the angular velocity of each wheel to predict the posture in the next sampling time. Another way of doing this, which is the one used throughout this paper, is to measure the position directly from the odometry, and then run the prediction and the update cycles at the same sampling time. This has several advantages, e.g., the possibility to use a variable sampling time, since one only needs to know the time interval after the sampling.
However, there are several ways of implementing this latter method. If one measures the angular velocities of the wheels, making them the **u** vector in the KF, then the model of the system is clearly non-linear, because of the matrix **J**. In this case, the Extended KF is required.
Nonetheless, the EKF is not guaranteed to converge and to be the optimal filter. So, the result may not be the expected, and is typically dependent of the initial state. In this case, several simulations and experimental tests should be made, to ensure the correct operation of the self-localization algorithm.
Another approach to this problem is to linearize the readings from the odometry, by first calculating the posture of the robot and then using the posture dynamic equations in the prediction cycle of the KF. This is similar to feedback linearization methods used in control theory, sharing with them the problem of might having unexpected results if the parameters of the inverse function used deviate even a little from the real inverse function. Moreover, if the noise is white and Gaussian in the measured wheels speed, then it will not be in the calculated posture, due to the non-linearities.
One way to reduce the problems of biased odometry errors is to add the increments of odometry to the Kalman estimate at each sampling time, instead of having a different accumulator. However, one should be aware that this does not guarantee better results, so simulation and experimental tests have to be done.
For the MCL implementation a motion model is needed. The odometry update for each particle was obtained sampling each wheel angular displacement from a normal distribution with mean given by the odometry readings from the wheels and a standard deviation of 5% of the wheel angular displacement, modelling the noise in the readings. The kinematics model was then used to convert the wheel angular displacement to the robot displacement in its body frame. This displacement was then converted to the field (global) frame, using the procedure described previously for the KF, and the position of each particle is then updated. This provides a much more accurate motion model then to just sample the displacements in *x*, *y* and θ.

5. SELF ALGORITHM

The robot used is shown in Figure 3. A catadioptic vision system can be seen on the top of the robot. This way, at least half of the field can be seen from every point, and thus the posture of the robot can be estimated according to some known characteristics of the soccer field.
In this work the self algorithm, originally described in [6] and improved in [3], has been used. It detects features, lines and goal colours, in the field image taken from the catadioptic system to locate the robot. Due to the field symmetry, the self algorithm cannot always disambiguate the robot posture, therefore returning several possible postures.

The self algorithm can sometimes fail to determine the robot posture. In such situations, no output is given. In those cases, the update cycle of the KF is omitted and in the MCL algorithm the importance factors are not updated. In the KF, a matching step is inserted before the update cycle, to check if the result is in a neighbourhood of the expected posture. This ensures no outliers are taken into account. The self algorithm has also been modified to output a probability of the given result to be correct. This can be used in the KF to change the matrix **R** at each sampling time. Nevertheless, this does not guarantee better results, though it is expected that the filter converges faster. With the MCL algorithm this could be used to change the parameters of the distribution p(y | x).

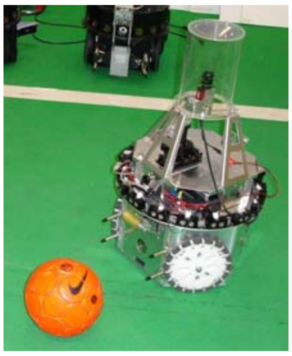
Figure 3 – Omni-directional Soccer Robot

The self algorithm measurement probabilistic model was estimated according to several factors, such as the number of field lines detected, the number of points defining each of those lines, the accuracy of the distances between them, and the accuracy of the goals estimated positions.

6. SIMULATION RESULTS

6.1 Simulation results for KF

Some prior simulations of the KF have been made in MATLAB. Four state vectors have been kept at each iteration: one for the real posture of the robot, one for the self algorithm measured posture, one for the posture calculated with the odometry, and finally one for the KF estimate.

The real posture of the robot is calculated using a pre-defined vector of angular velocities for each wheel, which generate the linear and angular velocities of the robot, using the kinematics model and the Jacobian, **J**.

The odometry noise has been modelled in two different ways. The radiuses of the wheels and the distance L differ from the real ones about 5 per cent. This is a reasonable assumption, since it is what happens in practice, where one cannot have infinite precision about any parameters. To turn the noise model more realistic, zero mean white and Gaussian noise with a standard deviation of 0.05 rad/s has been added to the measured angular velocities of the wheels. This is intended to model mainly slippage and terrain irregularities.

The self algorithm has been simulated as an unbiased white and Gaussian estimate of the real posture with a standard deviation of 0.1 m for the position, and of 0.1 rad for the orientation. This is a fairly good approximation to the real model.

In the simulations described next, the KF estimate has been initialized with zeros, and the matrix **P** with ones. This means the actual estimate of the position has a standard deviation of 1 m, and the estimate of the orientation has a standard deviation of 1 rad. The matrices **Q** and **R** are diagonal. At each iteration, the prediction and the update cycle are executed. Each result presented corresponds to 10 000 iterations.

Simulation 1:
Initial posture: x = 0, y = 0, θ = 0
Diag(**Q**) = Diag(**R**) = 0,01; 0,01; 0,1

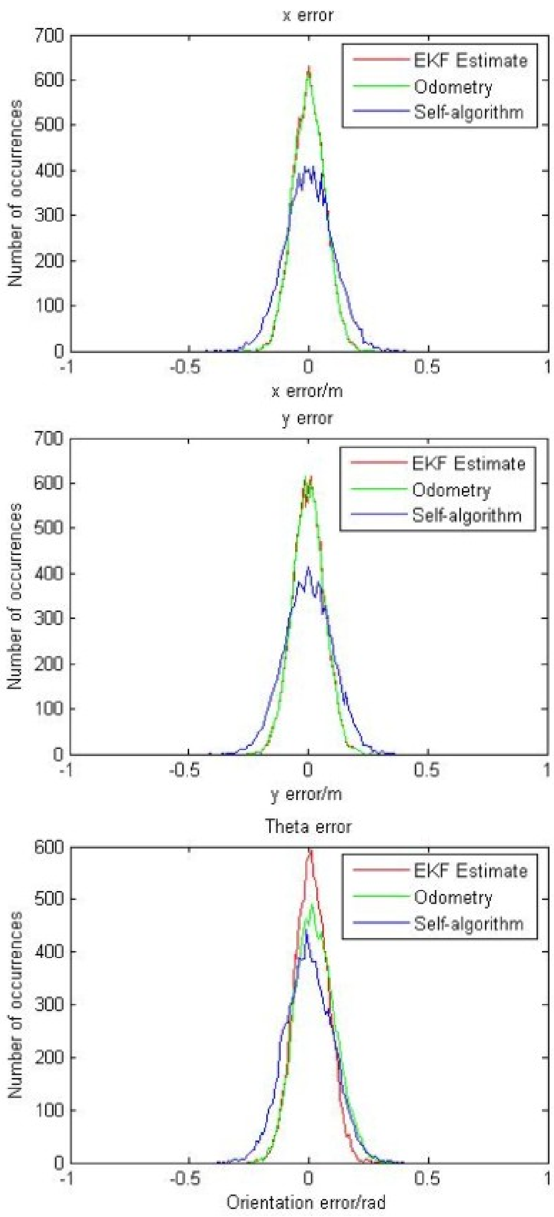

Figure 4 – Simulation 1 results

First of all, it is important to notice that the odometry and the KF estimates are very close. This is due to the fact that, as stated before, the increments of the posture read from the odometry are added to the last KF estimate, instead of having a different accumulator. As can be seen in Figure 4, the results are clearly better than using only the self algorithm to estimate the posture of the robot. If only odometry were used, then the error of the estimation would increase with the complexity of the trajectory taken by the robot.

Simulation 2:
Initial posture: x = 0, y = 0, θ = 0
Diag(**Q**) = 0,01; 0,01; 0,1
Diag(**R**) = 1; 1; 1

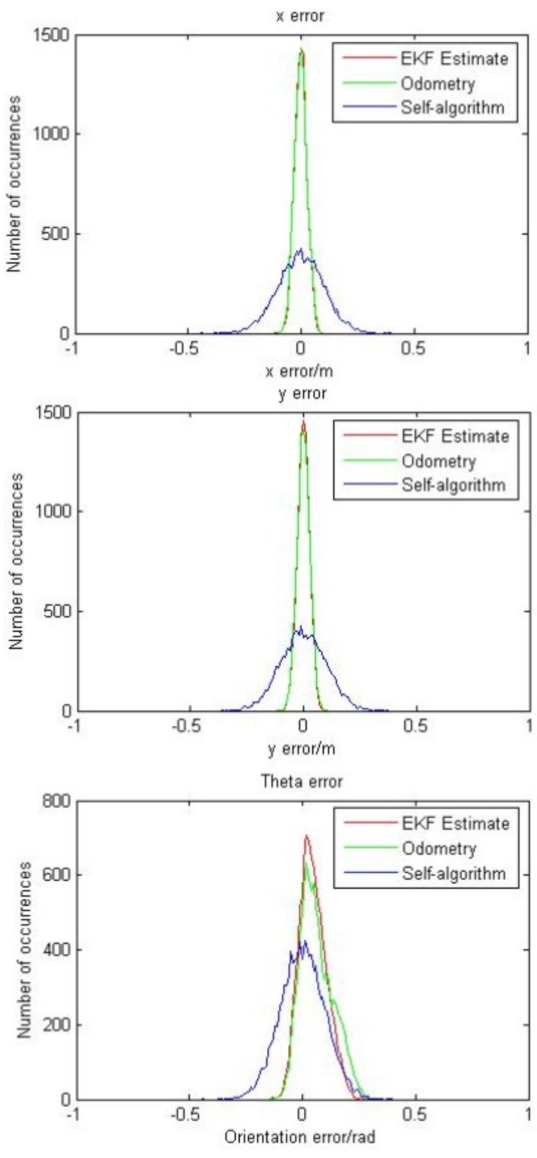

Figure 5 – Simulation 2 results

In the second simulation is quite clear that the posture of the robot is better estimated with the sensor fusion than using the sensorial information separately. However, since the matrix **R** has greater values in the diagonal than **Q**, the KF relies more on the odometry. As a consequence, the error of the estimation may not have zero mean, since odometry is usually biased because of hardware asymmetries. This can be seen in Figure 5, for the orientation errors. Hence, the parameters in matrices **Q** and **R** should take into account a tradeoff between having a small variance of the estimate error, and ensuring that it is unbiased.

6.2 Simulation results for MCL

Simulations of the MCL algorithm were made to test the implementation of the algorithm and the various parameters that influence its behavior. Simulation results will be shown for normal and mixture sampling with 1 and 4 results returned from the self algorithm. Some simulations were also ran to test the MCL algorithm robustness to robot kidnapping.
In all the simulations the initial pose was unknown. Because the initial distribution is unknown, the *belief* samples were sampled randomly according to a uniform distribution in *x*, *y* and *θ*. The simulations were ran with 1000 particles. The standard deviation used in the perceptual model was of 10 cm for *x* and *y* position values, and 1.3 degrees for the orientation. The axes are labeled in centimeters.
In this first test a monomodal perceptual model was used, i.e., only one result from the self algorithm was used. Readings from odometry had a 2% standard deviation and the motion model had a 10% standard deviation. The robot starts its sinusoidal path from left to right and is processed in 70 steps. As can be seen in Figure 7, the MCL algorithm solves the global localization problem. The path the robot follows is shown in red and the posture estimations are shown in blue.

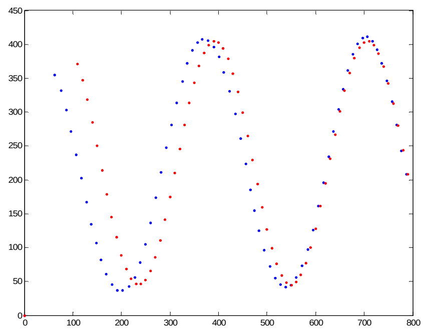

Figure 6 – Global localization using MCL

The average localization error of the MCL estimation throughout a series of tests was 2.14 cm, while the odometry error was 1.78 cm.
Using a larger standard deviation for the odometry results, 10%, one can see how the MCL improves on odometry alone. Using that, MCL's average localization error was 3.54 cm, while using just odometry was 13.9 cm.

As one can see, given the initial sample disposition, the first estimation still has a large error. This error is then minimized according to the MCL algorithm. The noise of the motion model controls the rate of convergence of the estimation but a larger noise will worsen its accuracy.

However, the kidnapped robot problem has presented some problems, due to the slow convergence rate. Using 4 results from the self-algorithm was not very helpful because, should the MCL algorithm localize the robot near one of the false possible localizations, it would be almost impossible for it to escape and direct towards the robot true position.

Using the mixture sampling, however, proved to be the solution to both of these problems. Figure 7 depicts a run of the MCL algorithm, with the robot being "kidnapped" along its path and the MCL algorithm quickly estimating the robot new pose. Again, the path the robot follows is shown in red and the posture estimations are shown in blue.

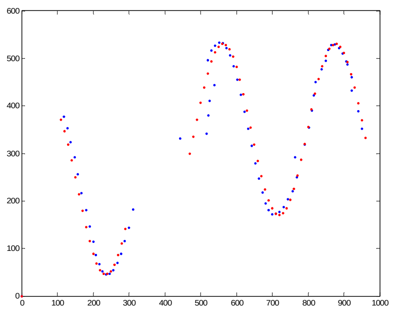

Figure 7 – Kidnapped robot using MCL

Here the robots begins his walk on the upper left side (100,380), being "teleported" to a new location (450, 300) when it arrives at (300,150). The error obtained is identical to the one obtained without the kidnap, because once the robot position is well estimated, both runs become a position tracking problem.

Runs using a perceptual model considering 4 postures returned by the self algorithm results were also made, having identical results to the one using just 1 self algorithm returned posture.

7. EXPERIMENTAL RESULTS

In this section we report on the results of applying the KF and MCL to the robots, shown in Fig. 3.

7.1 Experimental results for KF

First of all, as it is usually done in Kalman filter applications, several tests had to be done to find out matrices **R** and **Q** that give reasonable results. For the sake of simplicity, these matrices were considered diagonal. The variance of the position error was considered larger for the self algorithm than for the odometry, since the last one is more accurate for short distances. However, the self algorithm is more precise for the orientation. Moreover, the matrix **R** depends on the probability of the estimate given by the self algorithm. In this case, each element was considered to be proportional to the inverse of the referred probability. This way, larger variances are used when the estimate is less probable to be correct.

The elements of the matrix **P** were initialized with high values, meaning that initially the robot does not "know" where it is located. This matrix is supposed to be diagonal, like **R** and **Q**. However, it does not need to be initialized like that since the algorithm will make it tend to a diagonal matrix.

In a first test the odometry was correctly initialized. The errors for 5 random experiments are given in Table 1. The experiments were done making the robot follow a path with a few turns around the field (9 m x 4.5 m), in different trajectories, allowing time for the KF to converge. The errors were found calculating the difference between the approximately (hand measured) real posture and the last estimate.

Test Nr.	Error in **x**	Error in **y**	Error in **θ**
1	1.4 m	0.1 m	0 º
2	1.7 m	0.0 m	0 º
3	0.5 m	-0.4 m	0 º
4	0.8 m	-0.3 m	0 º
5	0.4 m	-0.6 m	5 º

Table 1

In a second test the posture of the robot differed from the initial odometry estimate (and from the KF). The results obtained are summarized in Table 2, noting that the final posture is also different from the one of the previous tests.

Test Nr.	Error in **x**	Error in **y**	Error in **θ**
1	0.3 m	-0.7 m	0 º
2	0.2 m	-0.4 m	0 º
3	1.0 m	0.3 m	0 º
4	0.6 m	0.3 m	0 º
5	-1.0 m	0.3 m	0 º

Table 2

If only odometry were used, then the errors would be quite larger. In the previous trials, the average error for the odometry was about 3 m for **x** and **y**, and 100º for the orientation. On the other hand, if only the self algorithm were used, then the posture of the robot would only be estimated a few times per minute, since the algorithm often fails to return a posture.

The results show that there is bias in the error in **x**, which can be due to miscalibration of the catadioptric vision system. However, the KF algorithm is quite accurate concerning the orientation.

7.2 Experimental results for MCL

Similar tests were made using the MCL algorithm as well. Tests were made for the global location problem, i.e., the robot did not know its initial localization, and using a different number of particles. The results after the robot moved for some time around the field are shown in Tables 4 and 5. More results were obtained for different number of particles, but are not reported here.

Test Nr.	Error in **x**	Error in **y**	Error in **θ**
1	0.1 m	0.7 m	-1 º
2	0.15 m	-0.35 m	1 º
3	0.2 m	0.0 m	5 º
4	-0.5 m	1.2 m	5 º
5	0.2 m	-0.3 m	0 º

Table 4 – 10 particles

Test Nr.	Error in **x**	Error in **y**	Error in **θ**
1	0.0 m	0.3 m	0 º
2	0.1 m	-0.85 m	3 º
3	-0.1 m	0.95 m	0 º
4	0.0 m	0.0 m	5 º
5	-0.3 m	0.3 m	0 º

Table 5 – 1000 particles

The results obtained with the MCL algorithm were similar to those previously obtained with the Kalman filter. The results for *x* and *y* are accurate to the meter and the orientation is correct to 5 degrees. Moreover, the bias in the *x* error does not occur. Also worth noticing is that the error is similar in all the tests, regardless of the number of particles used.

8. CONCLUSIONS AND FUTURE WORK

This paper presented the application of self-localization methods based on odometry and visual measurements applied to omnidirectional soccer robots. The Kalman filter showed to be a solution for integrating different sensorial information, which can be expanded in future, by adding accelerometers, gyroscopes, etc.

The MCL algorithm also proved to be a good approach to the robot localization problem. The results obtained were a big improvement on using the sensor information separately. Fusion with more sensors is also possible, requiring however an adjustment of the motion model and perception model, to reflect information from the additional sensors.

Several improvements can be done to the presented KF algorithm. The Extended KF was not tested when the prediction cycle is made with the readings of the angular displacement of the wheels. The results could be better since the non-linearity of the kinematics model of the robot is explicitly in the EKF. However, there are no guarantees of convergence of the filter.

The matching step implemented checks if the estimate of the self algorithm is in a predefined neighbourhood of the latest KF estimate. This neighbourhood could also change over time, since in the beginning it is expected that the KF estimate is further away of the real posture of the robot, so this neighbourhood could be bigger. Another solution is to decrease it when the values of the diagonal of **P** also decrease.

Finally, the initialization could be different than the one used. In this case, the initial estimate of the posture is (0, 0, 0). This may lead to slow convergence of the filter. Another way of doing this step is to tryout the self algorithm initially, and then use this estimate as the initial state.

As for the MCL, future developments in this area are concerned with the problem of dynamical environments and cooperative multiple robot localization (with a team of robots). Although the robotic soccer environment is dynamical in order that other robots and the ball are also moving, the objects used by the self-algorithm for localization (football field, its lines, goals and posts) are static, so it would not influence the MCL results.

Cooperative multiple robot localization is much more interesting for soccer robots. Its aim is to speed up the convergence of the process of self-localization (for the robots to “know” their positions as early as possible). Although relatively good precision is already obtained with a simple robot, the added precision obtained by cooperative work would be beneficial. Based on these facts, the cooperative multiple robot localization problem is a natural extension of the problems handled in this paper and is likely to be worked upon in the future.

REFERENCES

[1] Maria Isabel Ribeiro, (2000). Introduction to Kalman Filtering: a set of two lectures, IST.

[2] Moshe Kam, Xiaoxun Zhu, and Paul Kalata (1997). Sensor Fusion for Mobile Robot Navigation, *Proceedings of the IEEE*, **85**, No. 1, pages 108-119

[3] Hugo Costelha (2002). A Modified Localization Method for a Soccer Robot Using a Vision-Based Omni-directional Sensor, ISR/IST report

[4] D. Fox, S. Thrun, W. Burgard and F. Dellaert, (2000). Particle Filters for Mobile Robot Localization, *Sequential Monte Carlo Methos in Practice*, pages 470-498

[5] S. Thrun, D. Fox, W. Burgard and F. Dellaert (2000). Robust Monte Carlo Localization for Mobile Robots, *Artificial Intelligence Journal*, **128**, No. 1-2, pages 99-141

[6] C. Marques and P.Lima (2000). A localization method for a soccer robot using a vision-based omnidireccional sensor, In: *RoboCup 2000 Book*, Springer Verlag, 2001

[7] Bentley, J. (1980). Multidimensional divide and Conquer, *Comunications of the ACM*, **23**(4): 214-229