Particle Filter-based State Estimation in a Competitive and Uncertain Environment

Tim Laue
Universität Bremen
Fachbereich 3 – Mathematik / Informatik
Enrique-Schmidt-Straße 5
Bremen, Germany
Tel.: +49 / (421) – 218 64 209.
Fax: +49 / (421) – 218 9864 209.
E-Mail: timlaue@informatik.uni-bremen.de

Thomas Röfer
DFKI-Labor Bremen
Sichere Kognitive Systeme
Enrique-Schmidt-Straße 5
Bremen, Germany
Tel.: +49 / (421) – 218 64 200.
Fax: +49 / (421) – 218 9864 200.
E-Mail: Thomas.Roefer@dfki.de

Acknowledgements

The authors would like to thank the Deutsche Forschungsgemeinschaft (German Research Agency) for supporting this work through the priority program "*Cooperative Mobile Robots in Dynamic Environments*" as well as through the Transregional Collaborative Research Center "*Spatial Cognition*".

Keywords

Robotics, Noise, Modelling.

Abstract

In this paper we present the application of particle filters for state estimation on a humanoid robot. These filters are used for self-localization and ball tracking in a competitive soccer scenario using robots with limited perceptual and processing capabilities. Some extensions have been applied to the basic algorithms to adapt them to the special needs of this domain which is a subject of high noise and dynamic changes. Different experiments have been carried out to confirm the applicability and the precision of the approach.

Introduction

The RoboCup initiative is an international joint project to promote artificial intelligence and robotics. It is an attempt to foster AI and intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined [1]. The RoboCup domain is divided into several different categories and leagues. The approaches described in this paper have been applied the RoboCup Soccer Humanoid Kid-Size League in which teams of humanoid robots play soccer against each other. Two important sub-tasks in this domain (among stable walking, action selection, and several others) are the estimation of the own pose as well as the estimation of the position and velocity of the ball. The main source of information about the environment is – for our application – a camera inside the robot's pan-tilt head. Several properties of this set-up let accurate state estimation become a challenging task:

1. The robot's field of view is limited and only a section of the environment can be perceived at once.
2. All perceptions are noisy, perceiving false positives is also possible.
3. Imperceptible changes of the environment may occur, e.g. a replacement of the ball or the robots by a referee.
4. The computational resources are limited, but data needs to be processed in real-time to react successfully on changes of the environment caused by the opponent team.

One widespread approach for robot state estimation is the *Monte Carlo Localization* [2]. This probabilistic algorithm approximates a state by maintaining a set of hypotheses, denoted as particles as well as samples. The approach is able to deal with noise and non-linear changes of the environment

(e.g. kicked balls or “robot-kidnapping”). Different implementations of this algorithm have been used for localization by several RoboCup teams before, e.g. by [3].

In this paper, we describe its application on the two above-mentioned estimation tasks in the humanoid soccer scenario, realized on the robots of the B-Human team [4,5]. A closely related approach for humanoid self-localization has recently been published by [6], however using a robot with superior perceptual capabilities. A sophisticated – but for out application demanding too high computational efforts – approach for tracking a ball in a soccer environment was presented by [7].

By using extensions [8] of the basic Monte Carlo Localization approach, we show that it is possible to compute sufficiently accurate estimates even on platform with comparably low perceptual capabilities. The results of our robot experiments document the precision of the approach and its performance in dynamic situations.

This paper is organized as follows: First, we describe the scenario to which this work applies to, i.e. the environment as well as the robots and their perceptual capabilities. Thereafter, a short introduction on the Monte-Carlo-Localization algorithm and its extensions is given. This is followed by the description of their adaption and application to this domain. The paper is concluded by a presentation of experimental results.

a)  b) 

Fig. 1: a) A robot of the B-Human team in detail. b) A scene from a robot soccer match.

The Domain of this Research

As afore-mentioned, this work is settled in a competitive robot soccer scenario. This section gives a short introduction into the RoboCup Soccer Humanoid Kid-Size League and describes our robots and their perceptual capabilities and limitations.

The RoboCup Soccer Humanoid Kid-Size League

In this RoboCup league, two teams of two robots each play against each other on a 5.7m x 4.4m large soccer field[1]. To reduce complexity for computer vision, all elements of the environment are color-coded: the ball is orange, one goal is yellow, and the other goal is blue. In the corners of the field, additional beacons for localization are placed. The floor is an even green carpet with wide white field lines (see Fig. 1b) at specified positions. To avoid disturbances with this color scheme, all robots have to be mostly black.

[1] For 2008, it is intended to enlarge the field to 7.4m x 5.4m and to increase the number of players to three per team.

One match consists of two halves, 10 minutes each. During this time, the robots operate completely autonomously, no human intervention is allowed. The players can to communicate with each other, but additional external control computers are forbidden.

The Humanoid Robots

The robots have a maximum height of 60cm and must have human proportions. This means that they have to walk on two legs and fulfill a set of constraints regarding, e.g., their center of mass and the size of their feet. Special devices for holding or kicking the ball are not permitted. External sensing is only allowed via cameras; active sensors such as laser range finders are not allowed.

For our research, we used the robots of our B-Human team, which participated in RoboCup competitions in 2007; one of these robots is shown in Fig. 1a. Each robot is equipped with 20 servo motors (6 per leg to allow omni-directional walking gaits, 3 per arm, and 2 in a pan-tilt unit which carries the head). The total height of the robot is 48cm; its weight is 2.5kg. It is based upon a Bioloid construction kit which already includes a microcontroller for controlling the servo motors. The main software runs an on a PDA (Fujitsu-Siemens Pocket LOOX 720) that has 128 MB RAM and an XScale processor with 520 MHz. These are quite limited processing capabilities, but still allow running the cognitive parts of the software (including vision, the localization algorithms presented in this paper, and action selection) with 15 Hz and the motion control with 50 Hz.

The Robot's Perceptual Capabilities

Our robot's only external sensor is a camera inside the head. We use the PDA's original camera that has been lead outside the case. This camera provides 15 images per second with a resolution of 320 by 240 pixels; its opening angles are 45.1° by 34.8°. This is a quite limited field of view, as apparent in Fig. 2. To keep track of the own pose as well as of the ball's position, the camera needs to be moved constantly by the pan-tilt unit. Currently, many teams avoid this problem by using cameras pointed at omni-directional mirrors (that provide a field of view of 360°) or a set of cameras pointing in different directions. Having these perceptual capabilities, localization becomes trivial. But these non-human-like solutions will be disallowed in the future. The solutions presented in this paper anticipate these changes.

The vision software running on the PDA is able to extract some basic features that provide the base for the localization algorithms. These features (i.e. the ball and significant points on the field) are depicted in Fig. 2. Their recognition is based on the detection of significant color changes in the picture. Since the camera's perspective is known, it is possible to differentiate between features which actually belong to the field and those that don't (e.g. spectators). Nevertheless, the computation of the feature positions relative to the robot is subject to heavy noise, since the camera is on top of a walking (and shaking) robot.

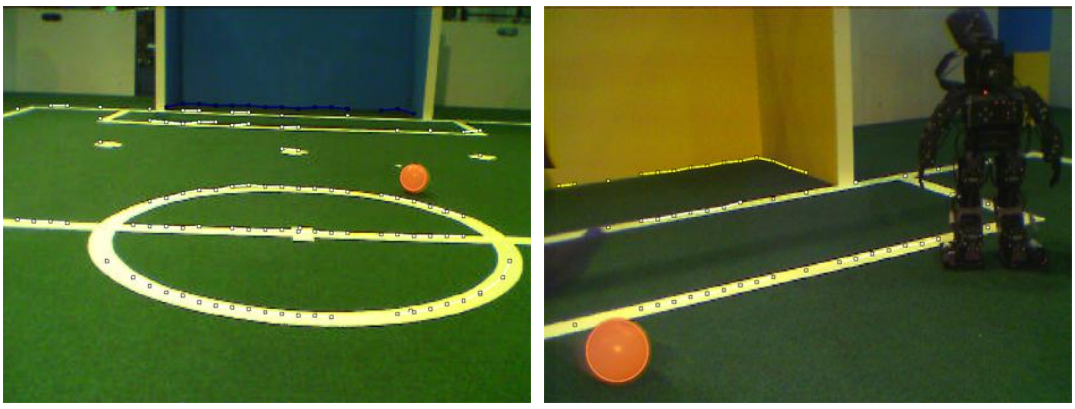


Fig. 2: Two images from the robot's camera. Some drawings in the images depict perceived elements of the environment: orange circles for balls, white dots for field line elements, and yellow and blue dots for the lower edges of goals.

Monte-Carlo Localization

This section gives an overview about the Monte-Carlo localization algorithm, which is the base for both, the self-localization and the ball tracking. Some extensions, which are crucial for the successful application in a RoboCup scenario, are also presented.

The General Algorithm for Localization

The Monte-Carlo algorithm for robot localization has been introduced by [2]. It is a probabilistic algorithm which approximates a state and its variance by a set of samples (also referred to as particles), with each sample consisting of one possible state and a weight which represents the likelihood of this state. This is illustrated by Fig. 3.

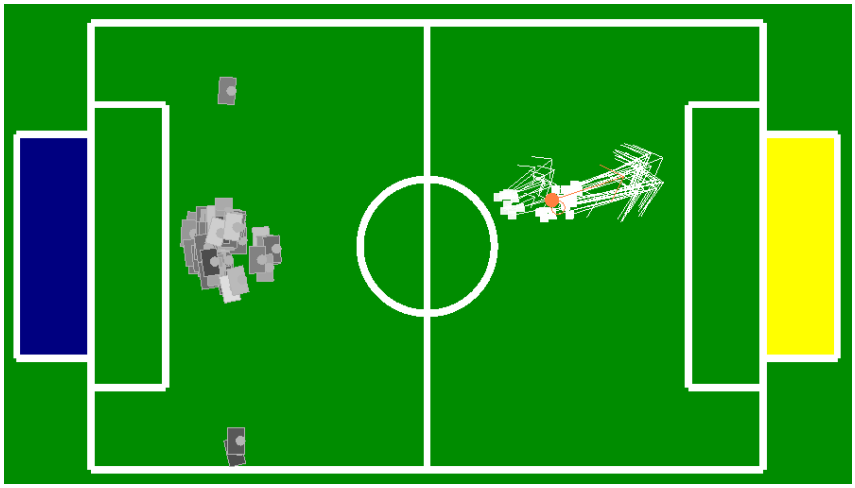


Fig. 3: Illustration of particle-based representation of the robot pose and the ball position and velocity. Each gray box denotes one possible pose of the robot; light boxes are more likely than dark boxes. The black box shows the resulting robot pose from the filter. The dots with arrows describe the ball samples (describing position and velocity of a ball). The orange one is the resulting ball estimate.

This algorithm is – in contrast to standard Kalman-filter-based [9] approaches – able to deal with non-linear state changes as well as – by using an extension of the basic approach – to efficiently cope with the kidnapped-robot problem, i.e. replacing a robot which is not able to recognize this state change and which has to re-estimate his position.

The general algorithm – for self-localization as well as for ball tracking tasks – looks as follows (adapted from [10], who also provide a comprehensive description):

1: *Algorithm_MCL*(X_{t-1}, u_t, z_t, e): X_t'(∅), X_t(∅)
2: for m = 1 to M do
3: $x_t^{[m]}$ = *motion_update*(u_t, $x_{t-1}^{[m]}$)
4: $w_t^{[m]}$ = *sensor_update*(z_t, $x_t^{[m]}$, e)
5: X_t' = X_t' + ($x_t^{[m]}$, $w_t^{[m]}$)
6: for m = 1 to M do
7: draw i with probability $\propto w_t^{[i]}$
8: add $x_t^{[i]}$ to X_t
9: return X_t

The algorithm processes the current sample set X_{t-1} in two passes to generate an up-to-date sample set X_t. During the first pass, the state of each sample x_t is modified according to the previously performed action u_t and thus resulting state change (line 3: *motion_update*). Afterwards, the weighting w_t of the samples is computed based on the current sensor data z_t (i.e. the percepts from the vision system in our case) and their fit to the given sample state and model e of the environment (line 4: *sensor_update*).

Both actions are – of course – dependent on the current domain and purpose of the filter. They are different for self-localization and ball tracking and therefore described in detail in the according later sections. In the second pass, the new set X_t is made up by randomly drawing elements from X_{t-1} proportional to their weight. This implies that samples that are more likely according to the current sensor data are more likely to be added (some of them even multiple times) to the new sample set whilst those with a low weighting might drop out. This pass is called *resampling* (lines 6-8). If no sensor update was performed (because of no new input from the vision system) and therefore no weightings have been computed, the resampling step is skipped.

By the time, the samples are distributed around the state which should be estimated. It is possible to get a result at any time, e.g. by weighting-based averaging of all samples.

Enhancements for Resampling

In theory, this algorithm is already sufficient to e.g. estimate a soccer robot's pose (consisting of a position and rotation on a two-dimensional plane, both in continuous coordinates). In practice, problems arise when the initial pose is not known or the robot is replaced during operation. When using a low number of samples (<100), it might take a long time until a sample comes close to the real position (given the fact that the motion model is noisy and thus the variance of the distribution increases by the time) and gets duplicated during resampling, since the state space is too large to be covered appropriately. Increasing the number of samples obviously also increases the runtime of the algorithm and therefore should be avoided. One common solution for this problem is to add new samples which have not been drawn from a previous distribution. Two crucial factors are the source and the number of new samples.

New samples could be added by just computing random samples inside the given state space. This is an appropriate solution, but might also not be very efficient since it could take too long, until a sample comes across an adequate position. In this domain, it is possible to compute new samples directly from the given sensor data and thus move the distribution to the real state quite quickly. This procedure will be described in the later sections for both self-localization and ball tracking.

The number of new samples might be set to a constant value. But this implicates two drawbacks: If the distribution already approximates the state quite well, these new samples might cause a higher variance, if their number is too high. On the other hand, if the number is too low, the particle filter might not be able to adapt to the state change fast enough. One solution to overcome this issue is the *Augmented_MCL* algorithm proposed by [6]. The following description is adapted from [10]:

1: *Algorithm_Augmented_MCL*(X_{t-1}, u_t, z_t, e): $X_t'(\emptyset)$, $X_t(\emptyset)$
2: static w_{slow}, w_{fast}
3: for $m = 1$ to M do
4: $x_t^{[m]} = motion\_update(u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = sensor\_update(z_t, x_t^{[m]}, e)$
6: $X_t' = X_t' + (x_t^{[m]}, w_t^{[m]})$
7: $w_{avg} = w_{avg} + M^{-1} w_t^{[m]}$
8: $w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
9: $w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
10: for $m = 1$ to M do
11: with probability $\max\{0, 1 - w_{fast} / w_{slow}\}$ do
12: add new pose to X_t
13: else
14: draw $i \in \{1,\ldots,N\}$ with probability $\propto w_t^{[i]}$
15: add $x_t^{[i]}$ to X_t
16: return X_t

This extension of the base algorithm keeps track of the overall weighting of the distribution over time. This weighting decreases, if the current sensor data does not match with the distribution anymore, e.g.

after replacing the robot or the ball. The weighting w_{slow} adapts slower to such changes than w_{fast}. Through their quotient, a requirement of new samples is computed (line 11). The adaptiveness of this process is steered via the two factors α_{slow} and a α_{fast} (with $0 \leq \alpha_{slow} < \alpha_{fast}$ cf. lines 8 and 9).

Both, the self-localization and the ball tracking make use of this approach, which is later referred to as *sensor resetting*.

Self-Localization

The particle filter for self-localization estimates the robot's pose. It uses the points on field lines and on the edges between the field and the beacons and goals as observations, and the motion estimated by the walking engine as input. The robot's pose $x_t^{[m]}$ consists of the (x, y)-position and its orientation θ on the field.

Motion Model

When applying the motion update, each $x_{t-1}^{[m]} \in X_{t-1}$ needs to be moved according to the robot's motion. An estimate of the robot's motion u_t is usually provided as odometry. In the B-Human robot control software, the walking engine provides odometry information, i.e. the offsets since the last motion update Δx_t, Δy_t, and $\Delta\theta_t$. However, since this information is only based on the motion of the legs, and not on the real motion of the robot, it is prone to large errors. Therefore, large amounts of noise are added to the pose of each sample during the motion update:

$$\begin{pmatrix} x_t^{[m]} \\ y_t^{[m]} \end{pmatrix} = \begin{pmatrix} x_{t-1}^{[m]} \\ y_{t-1}^{[m]} \end{pmatrix} + R_{t-1}^{[m]} \begin{pmatrix} \Delta x_t + sample(\max(\lambda_+ \Delta x_t, \lambda_- \Delta y_t, \lambda_n \overline{w}_{t-1}^{[m]})) \\ \Delta y_t + sample(\max(\lambda_+ \Delta y_t, \lambda_- \Delta x_t, \lambda_n \overline{w}_{t-1}^{[m]})) \end{pmatrix} \tag{1}$$

$$\theta_t^{[m]} = \theta_{t-1}^{[m]} + \Delta\theta_t + sample(\max(\lambda_\theta \Delta\theta_t, \lambda_d |(\Delta x_t, \Delta y_t)|, \lambda_r \overline{w}_{t-1}^{[m]})) \tag{2}$$

$R_{t-1}^{[m]}$ is the rotation matrix corresponding to $\theta_{t-1}^{[m]}$. *sample(x)* is a function that returns a random value in the interval $[-x,x]$. All λ are factors that scale the noise ratio depending on the robot's motion and the current weighting of the sample. λ_+ models noise along same translational direction. λ_- describes the influence of motion in the orthogonal direction. λ_θ scales rotational noise based on the robot's rotation. λ_d models rotational noise based on the robots translation. And finally λ_n and λ_r scale translational and rotational noise based on the relative weighing of each sample. $\overline{w}_t^{[m]}$ defines how the weighting of an individual sample relates to the average of all samples. It is defined as:

$$\overline{w}_t^{[m]} = \max\left(\frac{\sum_i w_t^{[i]}}{M w_t^{[m]}} - 1, 0\right)^2 \tag{3}$$

As a result, samples with a less than average weighting are moved even if the robot is not in motion at all, allowing the samples to move toward the real position of the robot.

Sensor Model

The image processing system recognizes points on field lines and points on edges between the field and the goals and the beacons. All these points are on the height of the field. Therefore, it is possible to determine their position relative to the camera by intersecting the ray starting in the camera center through their position in the image with the ground plane. Since the position of the camera relative to the ground plane is only roughly known, the resulting relative positions of the points are prone to high levels of noise. Since the vision system connects neighboring points, the orientation and a minimal length of the field line or edge are known for most of the points. For each image frame, a fixed number of such points is selected by random (the set of observations z_t). These points are used to determine for each sample how well the pose of the sample matches the current observations, resulting in the weighting w of the particle:

1: *Algorithm_sensor_update*(z_t, $x_t^{[m]}$, e) : $w_t^{[m]}$(1)

2: for each $p_{rel} \in z_t$ do

3: $$p_{abs} = \begin{pmatrix} x_t^{[m]} \\ y_t^{[m]} \end{pmatrix} + R_t^{[m]} \left(\begin{pmatrix} x_t^{cam} \\ y_t^{cam} \end{pmatrix} + R_t^{cam} p_{rel} \right)$$

4: $p_{model} = closest\_point(p_{abs}, e)$

5: $$\begin{pmatrix} x_{error} \\ y_{error} \end{pmatrix} = R_t^{cam\,-1} R_t^{[m]\,-1} \left(p_{abs} - p_{model} \right)$$

6: $$w_t^{[m]} = w_t^{[m]} \cdot gaussian\left(\left| \frac{x_{error}}{h + x_{rel}} \right|, \sigma \right) \cdot gaussian\left(\left| \frac{y_{error}}{h + x_{rel}} \right|, \sigma \right)$$

7: return $w_t^{[m]}$

Each point is projected to the field, given the current pose of the sample and the current pose of the camera relative to the robot (line 3). Then the closest corresponding point in the field model is determined (line 4). For this purpose, a number of tables were precomputed (called the environment e). They map (x, y)-positions on the field to the closest model point. Tables exist for field lines, green/blue edges, and green/yellow edges each of them in variants for edges of unknown orientation, edges along the field, and edges across the field, and also whether or not an edge is longer than a minimum length. The latter distincion helps to separate goal points from beacon points and points on the throw-in and penalty marks from regular field lines. Figure 4a depicts a table for mapping green/yellow edge points with unknown orientation and of any length.

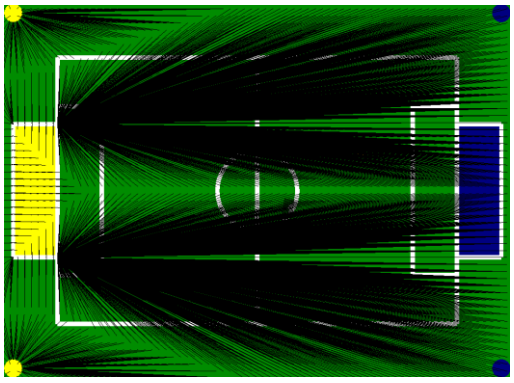
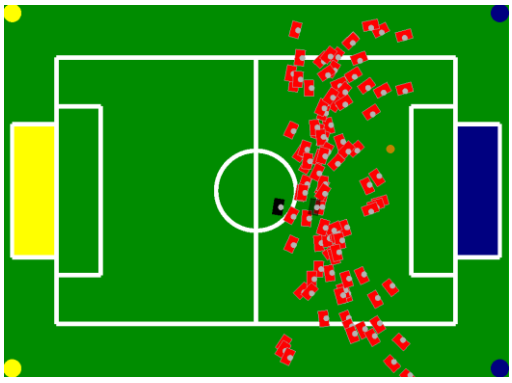
a)  b) 

Fig. 4: a) Depiction of a table mapping points to the closest green/yellow edge. b) Samples generated from green/blue edge points.

The difference between the closest model point and the projected measured point is rotated back to the coordinate system relative to the camera (line 5). As a result and given that the opening angles of the camera are rather small, x_{error} roughly corresponds to the distance error, and y_{error} corresponds to the bearing error. Both error distances are transferred back into pixel distance errors by dividing them by the forward distance to the measured point plus the height of the camera above the ground. Please note that these are rough approximations of the calculations that would be necessary, but since these operations are performed for each sample and for each selected point, the simplifications were necessary to achieve real-time performance on a computer without a floating point unit. The weight $w_t^{[m]}$ is determined by modeling the errors as Gaussian normal distributions based on the computed error values and a standard deviation σ (line 6).

Sensor Resetting

Since the camera only perceives a small part of the field at once, sensor resetting has to be done with caution. It is possible that the observations made by the robot match with several positions on the field. Therefore, adding new samples always increases the risk that they achieve the highest weighting for several frames and therefore attract the whole distribution. While a large σ prevents the

distribution from being to reactive, conservative values for α_{slow} and α_{fast} (0.05 and 0.0505) are used to making sensor resetting a rare event.

New samples are only generated from green/yellow and green/blue edge points (i.e. goals and beacons), because field line points are too ambiguous. Since the robot does not always perceive such points, a buffer of the previous points recognized is maintained. To generate a new sample position, a buffer entry is selected by random. Then a random point on a corresponding edge in the field model is selected. Afterwards, a random pose matching the measurement's distance and rotation from that point is constructed. Figure 4b shows such random poses for green/blue edge points, i.e. they result from the blue goal and beacons.

Ball Tracking

A particle filter for tracking the ball aims to estimates the current velocity of the ball to enable the robot to anticipate future game states, e.g. to make a goalie "jump" to the right corner of its goal in case of a shot. An additional effect of the application of a filter is the smoothing of the ball position's estimate. The rough perceptions of the ball are – especially when walking or moving the head fast – quite noisy and would cause the robot's pan-tilt head to jitter and also make the action selection more unstable. For our application, the ball state b is a four-dimensional vector $(x, y, v_x, v_y)^T$ with x and y representing its position and v_x and v_y representing its velocity. The ball position and velocity are tracked in the robot's coordinate system.

Motion Model

When applying the motion update, each $b \in X_{t-1}$ needs to be moved according to the robot's motion as well as to its own velocity, both in relation to the last execution of the filter. Let $(\Delta x_t, \Delta y_t)^T$ be a vector describing the translational part of the robot motion and ΔR_t^{-1} be a matrix describing the rotation of the coordinate system according to the robot's rotation. The time between two executions of the filter is called Δt. The scalar values λ_v and λ_t serve as user-defined parameters for controlling the amount of uncertainty added in each motion update.

The update according to the velocity provides a preliminary result $(x_p, y_p)^T$:

$$\begin{pmatrix} x_p \\ y_p \end{pmatrix} = \begin{pmatrix} x_{t-1}^{[m]} \\ y_{t-1}^{[m]} \end{pmatrix} + \Delta t \begin{pmatrix} v_{x_{t-1}}^{[m]} \\ v_{y_{t-1}}^{[m]} \end{pmatrix} + \Delta t \begin{pmatrix} sample(\lambda_v \left| \left(v_{x_{t-1}}^{[m]}, v_{y_{t-1}}^{[m]} \right| \right)) \\ sample(\lambda_v \left| \left(v_{x_{t-1}}^{[m]}, v_{y_{t-1}}^{[m]} \right| \right)) \end{pmatrix} \tag{4}$$

Through the addition of a random term, the uncertainty of the position increases dependent on the current velocity of the ball. Afterwards, the position is adapted to the robots motion:

$$\begin{pmatrix} x_t^{[m]} \\ y_t^{[m]} \end{pmatrix} = \Delta R_t^{-1} \begin{pmatrix} x_p \\ y_p \end{pmatrix} - \begin{pmatrix} \Delta x_t \\ \Delta y_t \end{pmatrix} + \begin{pmatrix} sample(\lambda_t \left| (\Delta x_t, \Delta y_t) \right|) \\ sample(\lambda_t \left| (\Delta x_t, \Delta y_t) \right|) \end{pmatrix} \tag{5}$$

Again, a term of uncertainty is added; this time depending on the speed of the robot.

Finally, the robot's motion has to be incorporated in the ball velocity together with a term of uncertainty dependant on the absolute velocity:

$$\begin{pmatrix} v_{x_t}^{[m]} \\ v_{y_t}^{[m]} \end{pmatrix} = \Delta R_t^{-1} \begin{pmatrix} v_{x_{t-1}}^{[m]} \\ v_{y_{t-1}}^{[m]} \end{pmatrix} + \Delta t \begin{pmatrix} sample(\lambda_v \left| \left(v_{x_{t-1}}^{[m]}, v_{y_{t-1}}^{[m]} \right| \right)) \\ sample(\lambda_v \left| \left(v_{x_{t-1}}^{[m]}, v_{y_{t-1}}^{[m]} \right| \right)) \end{pmatrix} \tag{6}$$

In addition to this straight-forward model, another feature for decreasing the ball position's variance has been implemented. Through often occurring – hardly perceivable – robot instabilities, the sensing of the ball position is subject of strong variations which cannot be distinguished from real ball motions. This might produce and advance samples with high speeds and thus let the distribution become too fluctuating. In fact, the ball is not moving most of the time during a humanoid robot soccer match. This situation can be considered by marking a subset of all samples as non-movable. When generating new samples (cf. subsection *Sensor Resetting*), a subset of these will receive a flag which prevents all updates considering ball velocities. Through the constant resampling process, these samples stabilize the distribution in case of lying balls and don't interfere in case of real ball motion.

Sensor Model

To compute a weighting for each sample, its current position is set in relation to the currently perceived ball position. If no ball is perceived, this step is – as well as the following resampling step – skipped. Let d_o and α_o be the distance and the angle to the observed ball and d_b and α_b the distance and angle to of the currently processed ball sample, then the weighting w_b can be computed as follows:

$$w_b = gaussian(|d_o - d_b|, \sigma_d) \cdot gaussian(|\alpha_o - \alpha_b|, \sigma_\alpha) \quad (7)$$

The implementation uses different standard deviations σ_d and σ_α since most noise occurs in the perception of the distance to the ball; in general, the angle is perceived quite precisely.

Sensor Resetting

Since the ball is a unique feature, it is – in contrast to self-localization – quite easy to generate new samples by just using the current perception that is often close to the real state of the ball. In domains of low noise, the perception often even serves directly as information for the later action selection; a filtering of the position is not needed. In our implementation, the robot always stores all ball perceptions made during the last time (e.g. two seconds). Whenever a new sample needs to be generated, two perceptions p_1 and p_2 (with p_2 having been stored at a later point of time than p_1) are randomly drawn from the set. From these perceptions, a velocity for the new sample might be computed directly. An actual position is determined by using p_1 and the velocity as in the ball velocity update equation in subsection *Motion Model*.

The resampling of the ball tracking filter strongly benefits from the possibility to add a dynamic number of new samples as described in the subsection *Enhancements for Resampling*. If the ball is not moving on the field for a longer time, only very few or no new samples need to be replaced in the distribution. When it starts moving (after a kick), the distribution needs to be adapted quite quickly. This mechanism is depicted in Fig. 5. In addition, it needs to be mentioned that during a robot soccer match situations occur, in which the robot has lost track of the ball completely (e.g. after having been fallen down). When perceiving the ball again, this resampling feature also helps to reestablish a proper distribution quite fast.

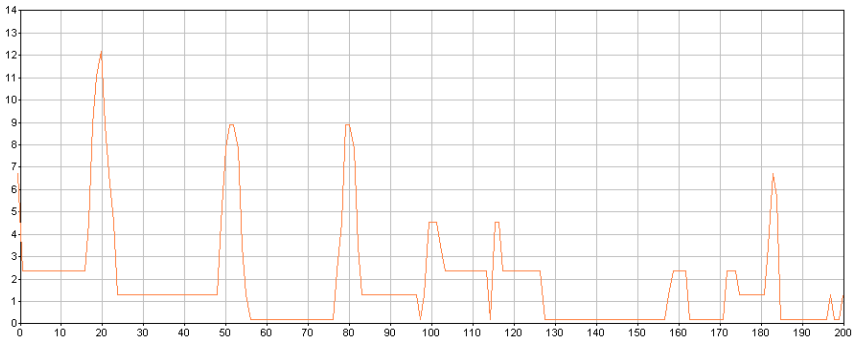


Fig. 5: Number of new samples in a sequence of consecutive executions of the ball tracking algorithm. In a simulated experiment, the ball has been immediately replaced three times (frames 20, 50, and 80). The number of new samples always immediately increases and soon after drops when the distribution fits the new sensor readings. A total of 40 samples has been used; this means that about 25% of the whole set get replaced. Afterwards, the ball was moved around the field (frame 100 and following). Again, some – comparably lower – peaks indicate changes of the motion direction.

Experimental Results

To evaluate the approaches described in this paper, several experiments have been carried out. One of the humanoid robots and a ball were placed and moved on a standard field (see Fig. 6a). To measure the correctness and precision of the robot's estimates, an environment for providing ground truth information has been set up. A camera, installed four meters above the field, provides a complete overview of the scenario. Its images are processed by the vision system of B-Smart [11], a RoboCup

team from the Small Size League, where a global vision is the current standard. To identify the robot and its orientation, an additional marker had to be placed on top of its head (see Fig. 6b).

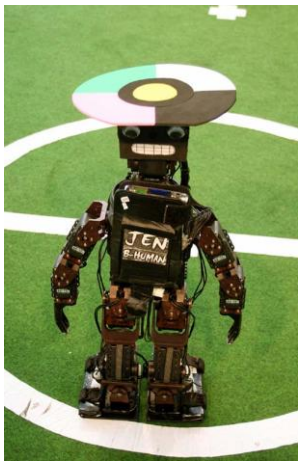
a)  b) 

Fig. 6: The experimental setup. a) The robot on a standard soccer field. b) A close view on the robot carrying the marker for tracking.

Self-Localization

To evaluate the self-localization, the robot was controlled around on the field using a joystick for slightly more than two minutes, while the robot's head was continuously turning from left to right and back. During this time, 2000 images were processed and the robot's estimate of its position was updated. Figure 7 shows the trajectory traveled as well as the trajectory according the robot's odometry calculation (cf. Fig. 7a) and when integrating external sensor readings (cf. Fig. 7b).

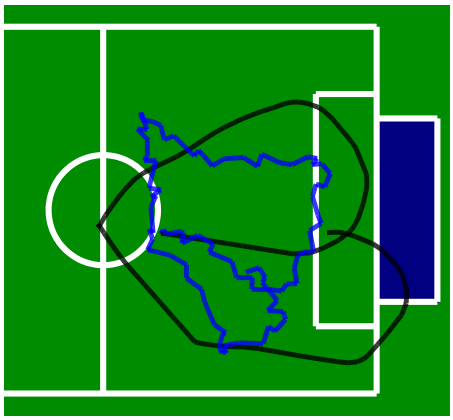
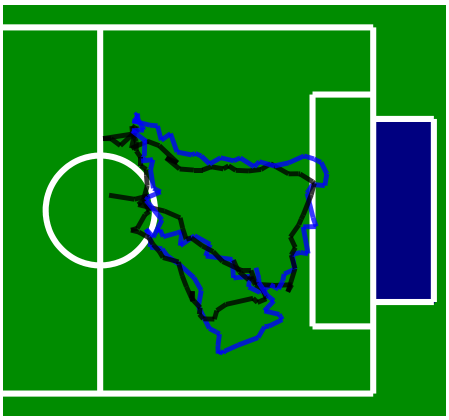
a)  b) 

Fig. 7: Experiment conducted for self-localization. The real trajectory the robot walked with up to 5cm/s is shown in blue. a) The trajectory only based on odometry calculation (in black). b) The estimated trajectory based on the self-localization module (in black).

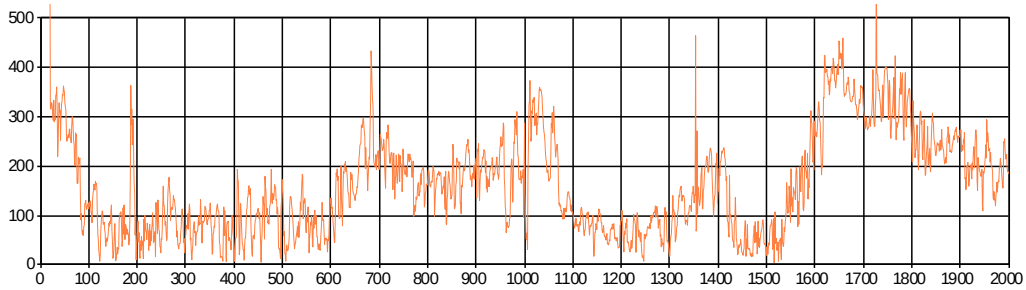


Fig. 8: Position errors in mm during the 2000 frames of the experiment depicted in Fig. 7b.

In the experiment, the robot's pose was estimated using 100 particles. Five edge points were used during the sensor update. σ was quite large (0.9) so that differences in the weights of the particles are kept small, resulting in a rather stable distribution of particles. During the 2000 frames of the experiment the mean distance error was 16.9 cm, i.e. 3% of the length of the field or 5.6% of its width. Fig. 8 shows the distribution of the error over time. In fact, the deviation is typically higher when the features used for localization are far away or out of sight. The former is for instance the case when the robot faces the goal on the other half of the field, while the latter happens when the robot is close to the sidelines and simply looks over them. In both cases, the robot has mainly to rely on odometry only, which is not very precise, as shown in Fig. 7a.

In a second experiment the same data was fed into the self-localization system, but 500 frames were skipped, simulating a kidnapped robot situation (cf. Fig. 9). It took the robot about 20 seconds to re-establish its position. Although this seems to be quite a while, please note that the robot's head could only pan back and forth about six times during this period of time. Since kidnapping of a robot happens rarely during games, the parameters of the self-localization method are tuned towards stability rather than quick adaptation.

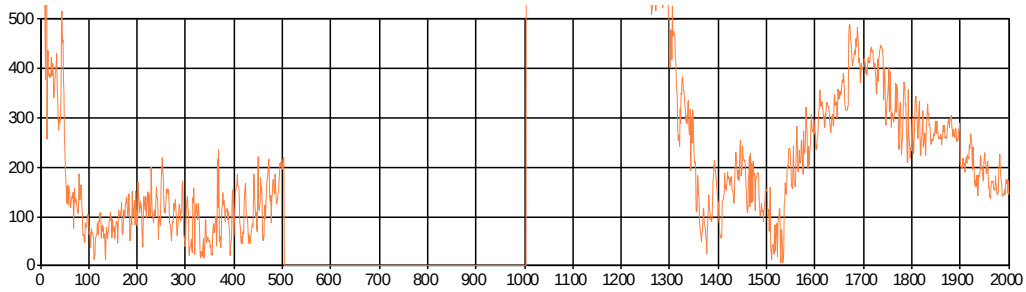


Fig. 9: Experiment on a kidnapped robot. Using the same experimental data as shown in Fig. 8, but skipping the frames 500 to 1000. Distance error again in mm.

Ball Tracking

To measure the precision of the ball tracking, the robot has been placed at the center circle of the field and the ball has been moved constantly around one half of the field. This has been done via a thin pole which is not perceivable by the robot's vision system. The whole trajectory of the ball is depicted in Fig. 10a. During the experiment, the robot's head was in its standard tracking mode for soccer matches, i.e. the robot tries to keep the ball in sight most of time, but occasionally looks around to keep localized. To avoid evaluation errors which result from changes of the ball velocity during a localization phase, we only consider so-called complete *tracks*. We define a track as a sequence of ball observations which is at least two seconds long and does not have any gaps (phases without any perception of a ball) longer than 300ms. In Fig. 10b, an example track is shown.

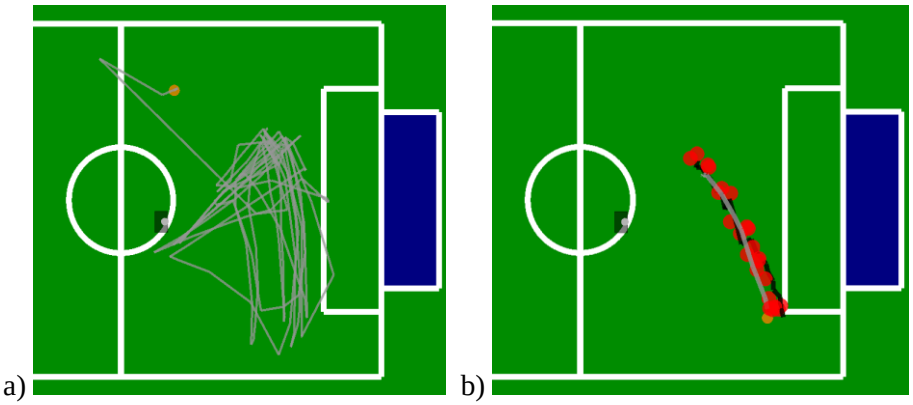


Fig. 10: Experiment conducted for ball tracking: a) The trajectory, along which the ball was moved and perceived by the ground truth vision, is depicted as a gray line. b) A single track observed by the robot. The real motion is again depicted in gray. The single perceptions are shown as red dots. The black line shows the estimated ball trajectory.

During the experiment, the ball was kept in motion all the time. The average top speed of all tracks was 1.27 m/s, the absolute top speed during the experiment was 2.66 m/s. This resulted in a mean position error of the ball of 26.6 cm. The mean velocity error was 0.28 m/s, the mean error of the ball motion's direction was 46.8°. This appears to be quite significant, but most tracks included strong changes of the ball direction, which lead to quite high errors during the adaption phase of the filter. Some tracks had been quite straight with only minimal directional changes (e.g. the one in Fig. 10b); in these cases, the mean error was 18.6° only.

This ball tracking implementation has been used by the team B-Human at RoboCup 2007. For all matches, the number of samples was set to 40 because of limited computational resources. Therefore, all experiments have been made with the same amount of samples.

Conclusion and Future Work

In this paper, we described the application of particle-filter algorithms in a humanoid soccer scenario. The experimental results show, that they enable even a perceptual limited robot to compute reasonable state estimates. As afore-mentioned, the implementation has already been used in a real robot competition. While having some of the weakest robots off all teams, B-Human was still able to reach the quarter finals with a remarkably well result in the previous round robin group: three wins and two draws with a total goal ratio of 8-1. The presented is intended to be kept for future application. Nevertheless, more computing power would contribute to achieve better results; a more complex vision module could provide more significant features for self-localization (e.g. field line crossing or goal posts) which enable a more detailed sample weighting or contribute to a better generation of new samples. The tracking of other robots is also a topic that has not been treated by us so far. The transfer of the algorithms to this issue will be an interesting subject of investigation in the near future. Additionally, cooperative state estimation in a team of humanoid robots becomes more important the larger the teams are.

References

[1] Kitano, H., Asada, M.: RoboCup Humanoid Challenge: That's One Small Step for A Robot, One Giant Leap for Mankind, International Conference on Intelligent Robots and Systems, Victoria, 1998.

[2] D. Fox, W. Burgard, F. Dellaert, and S. Thrun: Monte Carlo Localization: Efficient Position Estimation for Mobile Robots, Proc. of the National Conference on Artificial Intelligence, 1999.

[3] Röfer, T., Laue, T., Thomas, D.: Particle-filter-based self-localization using landmarks and directed lines, RoboCup 2005: Robot Soccer World Cup IX, Lecture Notes in Artificial Intelligence (LNCS 4020), Springer, 2006.

[4] Laue, T., Röfer, T.: Getting Upright: Migrating Concepts and Software from Four-Legged to Humanoid Soccer Robots, Proceedings of the Workshop on Humanoid Soccer Robots in Conjunction with the 2006 IEEE International Conference on Humanoid Robots, Genova, 2006.

[5] T. Röfer, C. Budelmann, M. Fritsche, T. Laue, J. Müller, C. Niehaus, and F. Penquitt: B-Human - Team Description for RoboCup 2007, RoboCup 2007: Robot Soccer World Cup XI Preproceedings, RoboCup Federation, 2007.

[6] J.-S. Gutmann and D. Fox: An Experimental Comparison of Localization Methods Continued, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'02), Lausanne, Switzerland, October 2002.

[7] H. Strasdat, M. Bennewitz, and S. Behnke: Multi-Cue Localization for Soccer Playing Humanoid Robots, RoboCup 2006: Robot Soccer World Cup X, Lecture Notes in Artificial Intelligence (LNCS 4434), Springer, 2007.

[8] C. Kwok and D. Fox: Map-based multiple model tracking of a moving object, RoboCup 2004: Robot Soccer World Cup VIII, Lecture Notes in Artificial Intelligence (LNCS 3276), Springer, 2005.

[9] R.E. Kalman: A new approach to linear filtering and prediction problems, Transactions of the ASME–Journal of Basic Engineering 82, 1960.

[10] S. Thrun, W. Burgard, D. Fox: Probabilistic Robotics, MIT Press, 2005.

[11] A. Burchardt, K. Cierpka, S. Fritsch, N. Göde, K. Huhn, T. Kirilov, B. Lassen, T. Laue, M. Miezal, E. Lyatif, M. Schwarting, A. Seekircher, and R. Stein: B-Smart - Team Description for RoboCup 2007, RoboCup 2007: Robot Soccer World Cup XI Preproceedings, RoboCup Federation, 2007.