Vision-Based Fast and Reactive Monte-Carlo Localization

Thomas Röfer†, Matthias Jüngel‡

† Bremer Institut für Sichere Systeme, Technologie-Zentrum Informatik, FB 3, Universität Bremen. E-Mail: roefer@tzi.de
‡ Institut für Informatik, LFG Künstliche Intelligenz, Humboldt-Universität zu Berlin. E-Mail: juengel@informatik.hu-berlin.de.

Abstract— **This paper presents a fast approach for vision-based self-localization in RoboCup. The vision system extracts the features required for localization without processing the whole image and is a first step towards independence of lighting conditions. In the field of self-localization, some new ideas are added to the well-known Monte-Carlo localization approach that increase both stability and reactivity, while keeping the processing time low.**

I. Introduction

THE Sony Four-Legged Robot League is one of the official leagues in RoboCup. In a match, two teams of four robots each compete against each other on a field of approximately 5×3 m^2 in size. All teams consist of robots of the same type, i. e. the Sony Aibo ERS 210 (cf. Fig. 1b). The Aibo is a four-legged robot with 20 degrees of freedom, a color camera, and more than 30 further sensors. The specialty of the league is that, on the one hand, the movements of the robots are the most complex in RoboCup so far, and on the other hand, an on-board camera is the most central sensor. As the robots are only equipped with a 200 MHz processor, all algorithms used, e. g. for image-processing or self-localization, have to be highly efficient to run in real-time.

II. Grid-Based Vision

As the main sensor of the robot is a camera, all objects on the RoboCup field are color coded. There are two-colored flags for localization (pink and either yellow, green, or sky-blue), the two goals are of different color (yellow and sky-blue), the ball is orange (as in all RoboCup leagues), and the robots of the two teams wear tricots in different colors (red and blue).

A very common preprocessing step for vision-based object recognition in such scenarios is color segmentation using color tables, e. g. [1], [2]. Such methods directly map colors to color classes on a pixel by pixel basis, which has some crucial drawbacks. On the one hand, the color mapping has to be adapted when the lighting conditions change, on the other hand, the mapping results in a loss of information, because the membership of a pixel in a certain class is a yes/no decision, ignoring the influences of the surrounding pixels. Some researchers try to overcome these limitations [3], but the solutions are too slow to work under real-time conditions on a robot such as the Aibo.

The Deutsche Forschungsgemeinschaft supports this work through the priority program “Cooperating teams of mobile robots in dynamic environments”.

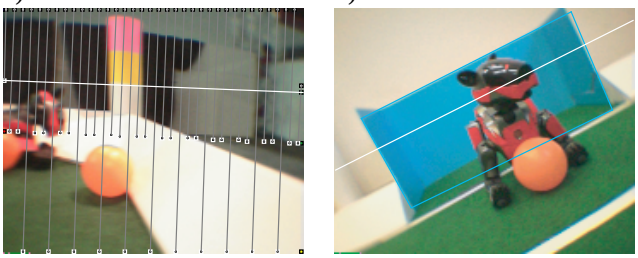

Fig. 1. Images taken by the robot’s camera. a) Pink-yellow flag and the horizon-aligned grid. The U-channel of the part of the image near the flag is shown as height map in figure 2d. b) Skyblue goal and Sony Aibo robot.

In this paper a method is presented to detect features in images very quickly without color calibration.

A. Approach

The key ideas of the image-processing method presented in this paper are that speed can be achieved by avoiding to process all pixels of an image, and a certain independence of the lighting conditions can be reached by focusing on contrast patterns in the three different color channels. In case of the Aibo, the camera takes YUV-images. From these images, objects such as flags, goals, and field lines have to be extracted. For such a feature extraction, a high resolution is only needed for far away and thus small objects, but such far away objects cannot appear anywhere in an image, instead they will be close to the *horizon*. The horizon is the intersection of the plane parallel to the ground on the height of the robot’s camera and the projection plane. Following this approach, only regions near the horizon need to be scanned at high resolution, while the rest of the image can be searched using a relatively wide spaced grid (cf. Fig. 1a).

The position of the horizon in the image can be calculated from the rotation of the head and the rotation of the body. The roll and tilt of the body are estimated from the readings of the robot’s acceleration sensors indicating the direction of the gravity, while the rotation of the head is determined from the angles of the three head joints (tilt, pan, and roll).

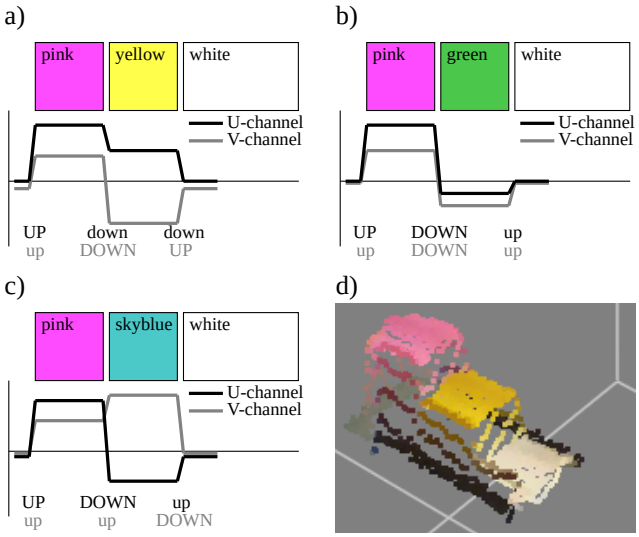

Fig. 2. Detection of flags. a) Pink-yellow flag. b) Pink-green flag. c) Pink-skyblue flag. d) U-channel of a pink-yellow flag as height map

B. Flags and Goals

To detect the flags in the image the vertical grid lines are scanned from top to bottom. During this scan the increase and decrease of the values of the two color channels containing information about the hue (U- and V-channel) is observed. Each position on the grid line with a large or a very large increase or decrease in one of the channels is marked with a *change marker* (up, UP, down, DOWN). This leads to a characteristic sequence of change markers for each grid line. Each flag is characterized by a 3×2 *pattern* containing the expected change markers at the upper edge of the flag at the color change inside the flag and at the lower edge of the flag for both color channels (cf. Fig. 2). Thus a simple pattern matching algorithm is sufficient to detect all flags.

Flags and goals found in an image are represented by the four angles describing their bounding rectangle (top, bottom, left, and right edge) in a system of coordinates that is parallel to the field. The angles to the top and the bottom of the flag are determined from the positions of the first and the last change markers of that flag. To determine the angles to the left and the right edge of the flag, a new scan parallel to the horizon is started from the center of the section with higher contrast to the background, i. e. the pink one, in both directions. The first large decrease in the U-channel on this scan line marks the "end" of the flag and provides a point on a vertical edge.

As goals are coded with only one color they are characterized by a 2×2 *pattern* containing the expected change markers at the upper and lower edges for both color channels. As such small patterns sometimes might match at noise in the image, all matches at the vertical scan lines are compared horizontally to filter the errors. This also provides the leftmost and the rightmost scan line intersecting

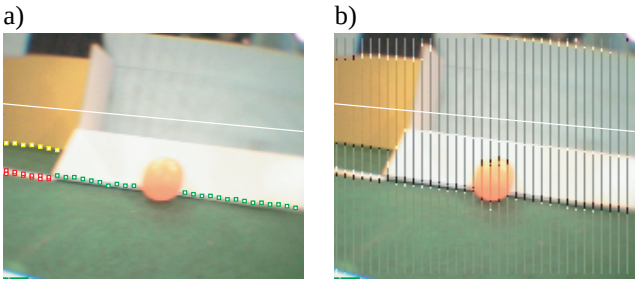

Fig. 3. Detection of lines. a) Three types of lines: field/goal, field/border, and field/line. b) The vertical scan lines are scanned from top to bottom. White pixels: increase in Y-channel, black pixels: decrease in Y-channel.

the goal. The exact horizontal extension of the goal is determined with two more horizontal scans near these scan lines. The angles to the top and the bottom of the goal are determined from the positions of the first and the last change markers of that goal; the angles to the left and the right edges result from the pixels found on the vertical edges.

C. Lines

Four different types of lines can be detected on the RoboCup field: edges between the skyblue goal and the field, edges between the yellow goal and the field, edges between the border and the field, and edges between the field lines and the field (cf. Fig. 3a). The key idea of the method presented here is not to actually extract *lines* from the image, but *pixels on lines* instead. This approach is faster and more robust against misinterpretations, because lines are often partially hidden either by other robots or due to the limited opening angle of the camera.

To find pixels on edges, in the image vertical lines having a distance of five pixels to one another are scanned from top to bottom following the method described in [4] (cf. Fig. 3b). In contrast to this method no color classification is applied. As the green of the field is very dark, all edges are characterized by a big difference of the Y-channel of adjacent pixels. An increase in the Y-channel followed by a decrease is an indication for an edge.

If the color above the decrease in the Y-channel is skyblue or yellow, the pixel lies on an edge between a goal and the field. The differentiation between a field line and the border is a bit more complicated. In most of the cases the border has a bigger size in the image than a field line. But a far distant border might be smaller than a very close field line. For that reason the pixel where the decrease in the Y-channel was found is assumed to lie on the ground. With the known height and rotation of the camera the distance to that point is calculated by projecting it to the ground plane. The distance leads to expected sizes of the border and the field line in the image. For the classification these sizes are compared to the distance between the increase and the decrease of the Y-channel in the image. The projection of the pixels on the field plane is also used to determine their rel-

ative position to the robot.

III. Self-Localization in RoboCup

An approach to self-localization is the so-called Monte-Carlo localization (MCL) by Fox *et al.* [5]. It is a probabilistic method, in which the current location of the robot is modeled as the density of a set of particles (cf. Fig. 5a). Each particle can be seen as the hypothesis of the robot being located at that position. Therefore, such particles mainly consist of a robot pose (x, y, θ), i. e. a vector representing the robot's x/y-coordinates and its rotation θ.

In many implementations, MCL was used on robots equipped with distance sensors such as laser scanners or sonar sensors, e. g. in the original one [5]. Only in a few approaches, vision is used for self-localization [6], [7]. Self-localization in RoboCup is different, because the area the robots can be located at is relatively small, i. e. the field, but in that area the position of the robots has to be determined quite precisely to allow different robots of the same team to communicate about objects on the field, and to follow some location-based rules of the game. Odometry is very unreliable, because the robots walk, and they tend to push each other around. As the Aibo is equipped with a sensor with a narrow opening angle of 58°, only a few objects usable for self-localization can be seen at once, and sometimes misreadings are in the majority. The method presented here takes these circumstances into account.

A. Monte-Carlo Localization

A Markov-localization method requires both a motion model and an observation model. The motion model expresses the probability for certain actions to move the robot to certain relative positions. The observation model describes the probability for taking certain measurements at certain locations.

The localization approach works as follows: first, all particles are moved according to the motion model of the previous action of the robot. Then, the probabilities p_i are determined for all particles on the basis of the observation model for the current sensor readings. Based on these probabilities, the so-called *resampling* is performed, i. e. moving more particles to the locations of samples with a high probability. Afterwards, the average of the probability distribution is determined, representing the best estimation of the current robot pose. Finally, the process repeats from the beginning.

The rest of this section will describe all these steps except from the observation model. This is described in two different versions in the sections IV and V.

B. Motion Model

The motion model represents the effects of actions on the robot's pose. First of all, an odometry position is maintained that is derived from the motions performed (gaits, kicks, etc.). As this value is only a rough estimate, in addition a random error Δ_{error} is assumed that depends on the distance travelled and the rotation performed since the last self-localization. For each sample, the new pose is determined as $pose_{new} = pose_{old} + \Delta_{odometry} + \Delta_{error}$. Note that the operation $+$ involves coordinate transformations based on the rotational components of the poses.

C. Resampling

In the resampling step, the samples are moved according to their probabilities. There is a trade-off between quickly reacting to unmodeled movements, e. g., when the referee displaces the robot, and stability against misreadings, resulting either from image processing problems or from the bad synchronization between receiving an image and the corresponding joint angles of the head. Therefore, resampling must be performed carefully. One possibility would be to move only a few samples, but this would require a large number of particles to always have a sufficiently large population of samples at the current position of the robot. The better solution is to limit the change of the probability of each sample to a certain maximum. Thus misreadings will not immediately affect the probability distribution. Instead, several readings are required to lower the probability, resulting in a higher stability of the distribution. However, if the position of the robot was changed externally, the measurements will constantly be inconsistent with the current distribution of the samples, and therefore the probabilities will fall rapidly, and resampling will take place.

The filtered probability p' is calculated as

$$p'_{new} = \begin{cases} p'_{old} + 0.1 & \text{if } p > p'_{old} + 0.1 \\ p'_{old} - 0.05 & \text{if } p < p'_{old} - 0.05 \\ p & \text{otherwise.} \end{cases} \quad (1)$$

Resampling is done in two steps: First, the samples are copied from the old distribution to a new distribution. Their frequency in the new distribution depends on the probability p'_i of each sample, so more probable samples are copied more often than less probable ones, and improbable samples are removed. In a second step that is in fact part of the next motion update, the particles are moved locally according to their probability. The more probable a sample is, the less it is moved. This can be seen as a probabilistic random search for the best position, because the samples that are randomly moved closer to the real position of the robot will be rewarded by better probabilities during the next observation update steps, and they will therefore be more frequent in future distributions. The samples are moved according to the following equation:

$$pose_{new} = pose_{old} + \begin{pmatrix} \Delta_{trans}(1 - p') \times rnd \\ \Delta_{trans}(1 - p') \times rnd \\ \Delta_{rot}(1 - p') \times rnd \end{pmatrix} \quad (2)$$

rnd returns random numbers in the range $[-1 \ldots 1]$. Typical values used for Δ_{trans} and Δ_{rot} are 10 cm and 30°.

D. Estimating the Pose of the Robot

The pose of the robot is calculated from the sample distribution in two steps: first, the largest cluster is determined, and then the current pose is calculated as the average of all samples belonging to that cluster. To calculate the largest cluster, all samples are assigned to a grid that discretizes the x-, y-, and θ-space into $10 \times 10 \times 10$ cells. Then, it is searched for the $2 \times 2 \times 2$ sub-cube that contains the maximum number of samples. All samples belonging to that sub-cube are used to estimate the current pose of the robot. Whereas the mean x- and y-components can directly be determined, averaging the angles is not straightforward, because of their circularity. Instead, the mean angle θ_{robot} is calculated as:

$$\theta_{robot} = \text{atan2}\left(\sum_i \sin\theta_i, \sum_i \cos\theta_i\right) \tag{3}$$

IV. Landmark-Based Self-Localization

Instead of using the distances and directions to the landmarks in the environment, i. e. the flags and the goals, this localization approach only uses the directions to the vertical edges of the landmarks. The advantage of using the edges for orientation is that one can still use the visible edge of a landmark that is partially hidden by the image border. Therefore, more points of reference can be used per image, which can potentially improve self-localization.

A. Flags and Goals

The image-processing described in section II determines bearings on the edges of flags and goals. These have to be related to the assumed bearings from hypothetical positions. To determine the expected bearings, the camera position has to be determined for each particle first, because the real measurements are not taken from the robot's body position, but from the location of the camera. From these hypothetical camera locations, the bearings on the edges are calculated. As the flags are cylinders, the edges of the flags seen are the tangents to these cylinders starting in the camera center. In case of the goals, their front posts are used as points of reference. As the goals are colored on the inside, but white on the outside, the left and right edges of the colored area even correlate to the posts if the goal is seen from the outside.

B. Probabilities

The observation model only takes the bearings on the edges into account that are actually seen, i. e., it is ignored whether the robot has *not* seen a certain edge that it should have seen according to its hypothetical position and the camera pose. Therefore, the probabilities of the particles are only calculated from the similarities of the measured angles to the expected angles. Each similarity s is determined from the measured angle ω_{seen} and the expected angle ω_{exp} for a certain pose by applying a sigmoid function to the difference of both angles:

$$s(\omega_{seen}, \omega_{exp}) = \begin{cases} \mathrm{e}^{-\sigma d^2} & \text{if } d < 1 \\ \mathrm{e}^{-\sigma(2-d)^2} & \text{otherwise} \end{cases} \quad \text{where } d = \frac{|\omega_{seen} - \omega_{exp}|}{\pi} \tag{4}$$

A typical value for σ is 50. The probability p of a certain particle is the product of these similarities:

$$p = \prod_{\omega_{seen}} s(\omega_{seen}, \omega_{exp}) \tag{5}$$

C. Inserting Calculated Samples

Landmark-based self-localization allows some samples to be moved to calculated positions. This approach follows the *sensor resetting* idea of Lenser and Veloso [8], and it can be seen as the small-scale version of the Mixture MCL by Thrun *et al.* [9]: on the RoboCup field, it is often possible to directly determine the position of the robot from the bearings on landmarks. The only problem is that these positions are not always correct, because of misreadings and noise. However, if a calculated position is inserted into the distribution and it is correct, it will get high probabilities during the next observation steps and the distribution will cluster around that position. In contrast, if it is wrong, it will get low probabilities and will be removed very soon. Therefore, calculated positions are only position hypotheses, but they have the potential to speed up the localization of the robot.

Two methods were implemented to calculate possible robot positions. They are used to fill a buffer of *position templates*. The first one uses a short term memory for the bearings on the last three flags seen. From the buffer and the actual bearings on goal posts, all combinations of three bearings are used to determine robot positions by triangulation. The second method only employs flags and goals currently seen. It uses all combinations of a landmark with reliable distance information, i. e. a flag, and a bearing on a goal post or a flag to determine the current position. For each combination, one or two possible positions can be calculated.

The samples in the distribution are replaced by positions from the template buffer with a probability of $1 - p'_i$. Each template is only inserted once into the distribution. If more templates are required than have been calculated, random samples are employed.

V. Lines-Based Self-Localization

The previous observation model uses the colored flags and goals for self-localization. However, there are no flags on a real soccer field, and as it is the goal of the RoboCup initiative to compete with the human world champion in 2050, it seems to be a natural thing to develop techniques

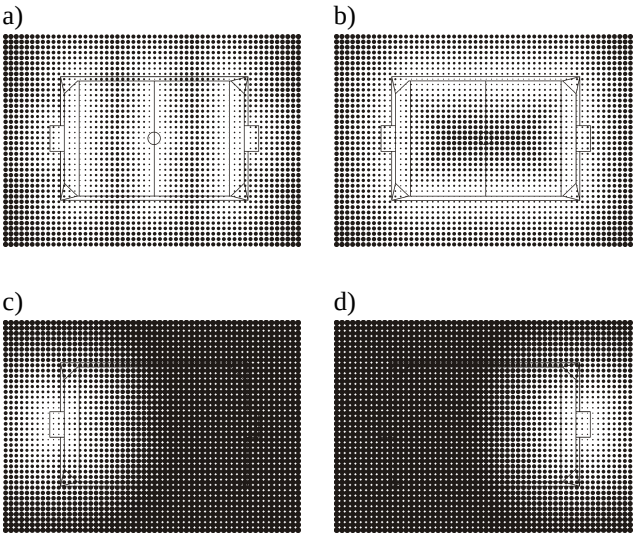

Fig. 4. Distances from lines. Distance is visualized as thickness of dots. a) Field lines. b) Border. c) One goal. d) The other goal.

for self-localization that do not depend on artificial clues. Therefore, a method to use the field lines to determine the robot's location on the field is currently under development, i. e. the approach only differs in the observation model from the one described in section IV.

A. Approach

The localization is based on the pixels on lines determined by the image-processing system (cf. Sect. II-C). Each pixel has a line type (field, border, yellow goal, or blue goal), and by projecting it on the field, a relative offset from the body center of the robot is determined. Note that the calculation of the offsets is prone to errors because the pose of the camera cannot be determined precisely. In fact, the farther away a point is, the bigger the errors will be.

The projections of the pixels are used to determine the probability of each sample in the Monte-Carlo distribution. As the positions of the samples on the field are known, it can be determined for each sample, where the measured points would be located on the field if the position of the sample would be correct. For each of these points in field coordinates, it can be calculated, how far the closest point on a real field line of the corresponding type is away. The smaller the deviation between a real line and the projection of a measured point from a hypothetic position is, the more probable the robot is really located at that position. However, the probability also depends on the distance of the measured point from the robot, because farther points will contain larger measurement errors, i. e. deviations of farther away points should have a smaller impact on the probability than deviations of closer ones.

B. Optimizations

Calculating the probability for all points on lines found and for all samples in the Monte-Carlo distribution would be a costly operation. Therefore the number of points is fixed (e. g. to ten), and these points are selected by random, but according to the following criteria: on the one hand, it is tried to select the same number of points from each line type, because points belonging to border lines and field lines are more frequent, but the points belonging to the goals determine the orientation on the field, because the field is mirror symmetric without the goals. On the other hand, closer points are chosen with a higher probability than farther away points because their measurements are more reliable.

Calculating the smallest distances of a small number of points to the field lines is still an expensive operation if it has to be performed for, e. g., 200 samples. Therefore, the distances are pre-calculated for each line type and stored in two-dimensional lookup tables with a resolution of 2.5 cm (cf. Fig. 4). That way, the distance of a point to the closest line of the corresponding type can be determined by a simple table lookup. Therefore, the method is computationally not slower than landmark-based Monte-Carlo localization.

However, there is a drawback in lines-based self-localization. It is not possible to directly calculate positions from the points on lines, i. e. a sensor resetting localization cannot be performed (cf. Sect. IV-C). However, this shortcoming is partially compensated by the fact that field lines are seen more often than flags.

VI. Results

The image-processing system presented in this paper was used in many RoboCup games of the GermanTeam in the Sony Four-Legged Robot League. Figure 1b shows a typical example of a detected goal, while figure 3a illustrates the results of line recognition.

Figure 5 depicts some examples for the performance of the landmarks-based self-localization using 100 samples. The experiments shown were conducted with SimRobot [10], [11]. The results demonstrate how fast the approach is able to localize and re-localize the robot. At the RoboCup 2002, the method also proved to work on real robots. For instance, before the beginning of a game the robots of the GermanTeam were just started somewhere on the field, and then—while still many people were working on the field—they autonomously walked to their initial positions. In addition, the self-localization worked very well on fields without an outer barrier, i. e. without a white background behind the flags and the goals.

The self-localization using lines is still in an experimental state. At the moment, experiments have only been performed in the simulator, but their results are quite promising. Figure 6 shows an experiment conducted with 200 samples. The method is always able to find the position of

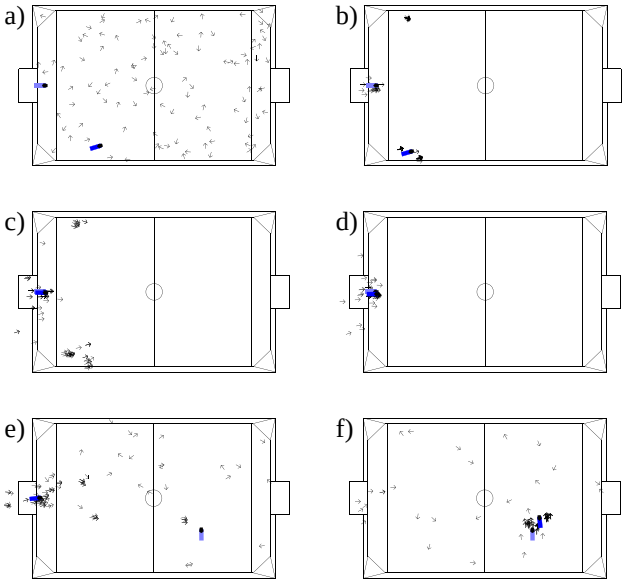

Fig. 5. Distribution of the samples during landmark-based localization while turning the head. The bright robot body marks the real position of the robot, the darker body marks the estimated location. a) After the first image processed (40 ms). b) After eight images processed (320 ms). c) After 14 images (560 ms). d) After 40 images (1600 ms). e) Robot manually moved to another position. f) 13 images (520 ms) later.

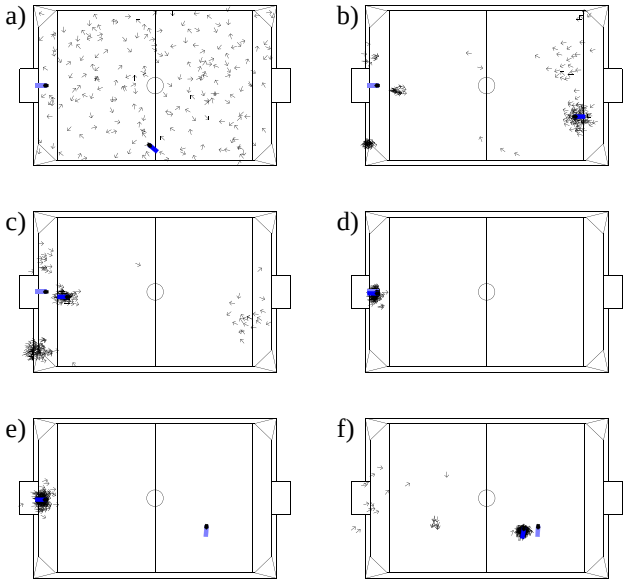

Fig. 6. Distribution of the samples during lines-based localization while turning the head. The bright robot body marks the real position of the robot, the darker body marks the estimated location. a) After the first image processed (40 ms). b) After 16 images processed (640 ms). c) After 30 images (1200 ms). d) After 100 images (4000 ms). e) Robot manually moved to another position. f) 40 images (1600 ms) later.

the robot, but it takes longer than the landmark-based localization. In addition, the mirror symmetry of the field is a problem. In figure 6b, the method first selects the wrong side of the field. However, when a goal comes into view, the position flips over to the correct side (cf. Fig. 6c), and it remains stable. Figures 6e and 6f demonstrate that the approach is also capable of re-localization after the robot was moved manually.

VII. Conclusions

This paper presents two approaches for vision-based self-localization in RoboCup. They are based on a vision system that extracts features without processing whole images, and that has reached a certain independence of lighting conditions. One method uses landmarks for localization, the other is based on field lines. Both approaches are variants of the well known Monte-Carlo localization. While using only a small number of samples, they increase the stability of the localization by a slow adaptation of the probabilities of the samples, and they speed up and increase the precision of the localization by a so-called probabilistic search that moves samples locally dependent on their probabilities. This results in a fast, reactive, and precise self-localization of the robot.

References

[1] J. Bruce, T. Balch, and M. Veloso, "Fast and inexpensive color image segmentation for interactive robots," in *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '00)*, 2000, vol. 3, pp. 2061–2066.

[2] F. K. H. Quek, "An algorithm for the rapid computation of boundaries of run length encoded regions," *Pattern Recognition Journal*, vol. 33, pp. 1637–1649, 2000.

[3] Robert Hanek, Thorsten Schmitt, Sebastian Buck, and Michael Beetz, "Fast image-based object localization in natural scenes," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2002, Lausanne*, 2002.

[4] M. Jamzad, B. Sadjad, V. Mirrokni, M. Kazemi, H. Chitsaz, A. Heydarnoori, M. Hajiaghai, and E. Chiniforooshan, "A fast vision system for middle size robots in robocup," in *5th International Workshop on RoboCup 2001 (Robot World Cup Soccer Games and Conferences)*. 2002, number 2377 in Lecture Notes in Computer Science, pp. 71–80, Springer.

[5] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo localization: Efficient position estimation for mobile robots," in *Proc. of the National Conference on Artificial Intelligence*, 1999.

[6] F. Dellaert, W. Burgard, D. Fox, , and S. Thrun, "Using the condensation algorithm for robust, vision-based mobile robot localization," in *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999.

[7] J. Wolf, W. Burgard, and H. Burkhardt, "Robust vision-based localization for mobile robots using an image retrieval system based on invariant features," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[8] S. Lenser and M. Veloso, "Sensor resetting localization for poorly modeled mobile robots," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[9] S. Thrun, D. Fox, and W. Burgard, "Monte carlo localization with mixture proposal distribution," in *Proc. of the National Conference on Artificial Intelligence*. 2000, pp. 859–865, AAAI.

[10] T. Röfer, "Strategies for using a simulation in the development of the Bremen Autonomous Wheelchair," in *Simulation-Past, Present and Future*, R. Zobel and D. Moeller, Eds. 1998, pp. 460–464, Society for Computer Simulation International.

[11] T. Röfer, "An architecture for a national robocup team," in *RoboCup 2002*. 2003, Lecture Notes in Artificial Intelligence, Springer, to appear (already published in RoboCup 2002: Robot Soccer World Cup VI Pre-Proceedings, 388-395).