Advanced Robotics, 26(14):1603-1621, 2012

Utilizing the Structure of Field Lines for Efficient Soccer Robot Localization

Hannes Schulz Sven Behnke

University of Bonn, Institute for Computer Science VI,
Autonomous Intelligent Systems, Friedrich-Ebert-Allee 144,
53113 Bonn, Germany
{*schulz, behnke*}*@ais.uni-bonn.de*

Abstract

Self-localization on the field is one of the key perceptual tasks that a soccer robot, e. g. in the RoboCup competitions, must solve. This problem becomes harder, as the rules in RoboCup more and more discourage a solely color-based orientation on the field. While the field size increases, field boundary markers and goals become smaller and less colorful. For robust game play, robots therefore need to maintain a probabilistic pose estimate and rely on more subtle environmental clues. Field lines are particularly interesting features, because they are hardly ever completely occluded and observing them significantly reduces the number of possible poses on the field.

In this work, we present a method for line-based self-localization on a soccer field. Unlike previous work, our method first recovers a line structure graph from the image. From the graph, we can then easily derive features such as lines and corners. Finally, we describe optimizations for efficient use of the derived features in a particle filter. The method described in this paper is used regularly on our humanoid soccer robots, which won the RoboCup TeenSize competitions in the years 2009–2011.

keywords: robot, soccer, humanoid, vision, self-localization

1 Introduction

RoboCupSoccer is a popular benchmark for AI and robotics research [1]. On its way to realistic soccer environments, RoboCup started out with small-sized, color-coded scenarios. Gradually, artificial markers, colors and special lighting are removed and the soccer field size is increased to encourage teams to build reliable vision systems which can compete under real-world conditions. While other leagues, like the MidSize league, already went a long way, the Humanoid League is still at the beginning of this transition. Especially the small available computational power and noisy observations due to mostly unmodelled camera motion prevented large steps so far. From the experience in MidSize-league, however, we can learn that the removal of colored landmarks emphasizes the importance of field lines. For precise

positioning, for example during the setup phase of the game, the use of lines and crossings is already mandatory since the uncertainties of other landmarks are prohibitively large. Finally, the restricted field of view in humanoid robots—as opposed to omnidirectional vision systems commonly used in other leagues—can be partially compensated for using field lines.

In this work, we present a system for detecting field lines and employ them for localization on the field. This is by far not the first method presented for this purpose; however, our approach has several advantages. The structure of the field lines is determined efficiently using algorithms inspired from work on analysis of handwritten digits [2]. This method employs local and global cues to determine a graph which captures the essential structure of lines in an image taken by the robot's camera. The graph has a particularly nice structure: nodes are candidates for corners where the field lines meet—the branching factor determines the corner type. Edges in the graph correspond to field lines. Most importantly, the estimates of line parameters are not influenced by spurious segments or noisy local orientation estimates.

Notably, the favorable properties of the algorithm come at little cost. We incorporated line and corner features into a particle filter which runs online on our humanoid soccer robots. This is possible, because the costly per-particle association decision of observed features to landmarks can be simplified.

The remainder of the paper is organized as follows. The next section reviews previous work on line-based localization. Section 3 and 4 describe preprocessing and feature extraction, respectively. Section 5 presents how line and corner features can be used in a particle filter. In Section 6 we evaluate the proposed methods. We provide qualitative and quantitative experimental results using motion capture as ground truth. For pose tracking, we show that our system can rely solely on line-based clues, greatly reducing the dependence on a color-coded soccer environment. Finally, we report on experiences made during RoboCup competitions.

2 Related Work

Work on field-lines for localization in RoboCup environments can be described on three axes. First, how candidate line segments are found; second, how the line segments are merged to lines; and third, how detected lines are used for localization. Naturally, the literature describes only combinations of all three approaches and this work is no exception. Nevertheless, this work contributes to all three areas.

Finding candidate line segments is usually performed using green-white-green transition filters [3] or scan-lines [4, 5] on the image. Convolving the image is an expensive operation, however, and scan-lines ignore context. Our method simply makes use of all white pixels and rejects those which are not part of the field lines.

Candidate line segments are then processed to find actual lines. Hough-space techniques are commonly used for this purpose (e. g. [6], [7]). These methods need to calculate a large accumulator array and require double book-keeping of line-pieces for post-processing the found lines. Also, estimated line orientations of small segments tend to be quite noisy. The same is true for the approach followed by the Nao team NUManoid [3], where lines are determined by adding and removing points to candidate

lines, which requires tuning of numerous parameters. In the approach presented here, candidates for lines emerge naturally from the determined line structure graph.

Finally, candidate line segments or lines can be used for localization. Lauer et al. [8] use all candidate line segments for performing gradient descent from the currently estimated pose. This technique relies on the favorable properties of omnidirectional cameras, which are not allowed in the humanoid league, and on stable distance estimates, which are hard to determine for humanoid robots with unknown camera pitch and roll angles. Röfer et al. [5] also make use of all candidate line segments using a pre-calculated lookup table for the observation model. With this method, segments which are observed on a line can be associated with different lines in the world, supporting improbable poses. In our experience, this approach also requires a large floor and wide standard deviations in the observation model such that spurious line segments do not completely destroy the belief. For precise localization, this approach is not helpful. Furthermore, the computational load is high if it is used in combination with particle filters, where each particle has to integrate information from all candidate line segments (see however Whelan et al. [9] for an efficient integration of this approach with a Kalman filter). Consequently, in humanoid and Aibo teams, field line extraction from candidate line segments prevails for localization purposes. The resulting long lines can then either be used as pose constraints [10] or directly used in a particle filter. Pose constraints rule out many possible poses and need to be relaxed iteratively in the case where no allowed poses are left. Particle filters can represent more complex beliefs. To achieve real-time performance, however, speed optimization is critical. In this work, we describe optimizations used to acquire a high frame-rate even when many features are detected in an image.

3 Vectorization

The first step of our algorithm is the vectorization of the field lines. Our robots are equipped with three color cameras (IDS uEye 1226 LE) which capture images with a WXGA (752×480) resolution in YUV 4:2:2 color space. The individual pixels are classified to color classes as follows. First, the Y-component is compared to luminance thresholds for classification of black and white. For pixels with intermediate luminance, the color class is defined by a lookup-table for the U and V values. The six color classes (orange, green, blue, yellow, magenta, and cyan) are described by ellipses in the UV plane. In addition, each color class is restricted to an interval in the Y dimension.

The pixels assigned to a color class are counted in a separate 94×60 grid (one eighth of the our camera resolution in each dimension). Each pixel in this color class image represents the number of occurrences of its color in a 8×8 window of the original image. While this subsampling reduces spatial resolution, it allows for quick access to the density of the corresponding color in image regions. All algorithms below are performed on these subsampled color class images, albeit we use subpixel precision when possible.

In the next processing step, simple pre-processing operations are performed on the color images. We reduce unwanted effects such as Bayer-pattern induced orange and cyan colors next to sharp contrasts. In unreliable corner regions of our wide-angle lens, we delete all classified colors. In Figure 1(a), we

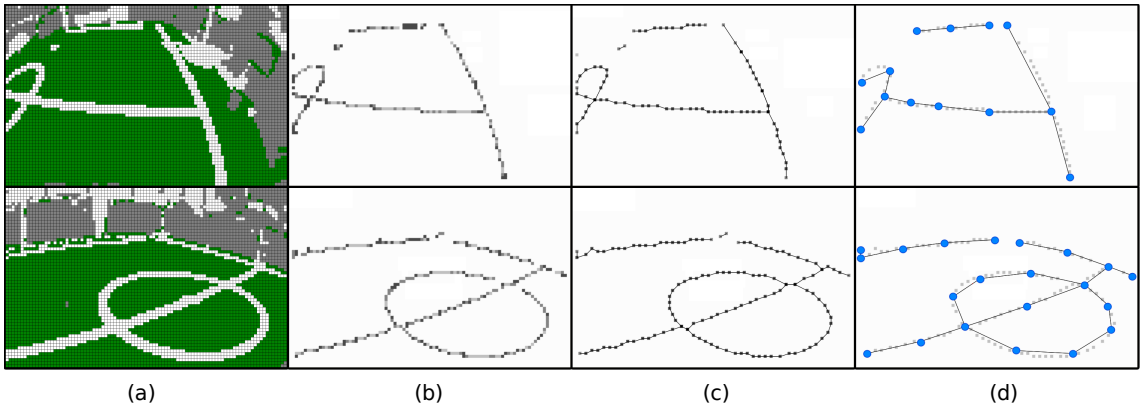

Figure 1: Visualization of field line vectorization process. (a) subsampled image of pixels classified as "white" and "green", with cluttered background; (b) skeleton inside field boundary; (c) all nodes with connections; (d) keynode structure graph.

exemplarily show the result of classification for green and white.

The vectorization of lines is now implemented in two major steps: We first extract the field boundary and, second, determine the structure of lines within the field boundary. The following sections describe the two steps in detail.

3.1 Field Boundary Extraction

In robot soccer games, everything of interest is located on a green carpet. While the area outside the field is undefined and cluttered, objects on the field can be clearly distinguished by color and structure. Consequently, as a first step we segment the field region, thereby further reducing the area of interest.

We use a four-step boundary scanning algorithm. This algorithm first merges subsampled color images into a field-color image and binarizes them. Then, it retrieves a boundary estimate for all columns independently. In a final step, we smooth the estimates and fill gaps in the boundary.

3.1.1 Merging Color Images

We want to determine the boundary of the green field in situations where objects on the field might occlude some parts of the it. To this end, we create a new 94×60 field-color image which is composed as a weighted sum of previously classified colors. The weights are chosen to roughly approximate the probability of a color being part of the field or occluding it. For green and orange, we choose the weight 1.0, since these colors rarely appear outside the field, but for white and black 0.5, since besides field lines and robots, bright lights in the background and spectators also appear in these colors. A pixel value at position (i, j) in this summed image is denoted as $g_{i,j}$.

3.1.2 Binarization of Field Color

For a pixel to be part of the field, it must exceed a minimum threshold, be part of a largely field-colored line or have enough field pixels below. We use three thresholds t_{pix}, t_{row}, and t_{win} to determine whether some pixel $g_{i,j}$ is likely to be inside the field. The threshold tests are arranged in order of complexity. First, we check the value of the pixel itself. If $g_{i,j} \geq t_{pix}$, the pixel undergoes further examination. If the pre-calculated row-sum $r_{i,j} = \sum_{i'} \sum_{j' \in \{-1,0\}} g_{i',(j+j')}$ is larger than some threshold t_{row}, the pixel is binarized to 1. The row-sum acts here as a prior which biases weak pixels in rows with many high-valued pixels to be part of the field. Pixels which do not pass t_{row} are examined by the most expensive test, which compares the sum $s_{i,j} = \sum_{i'=i-4}^{i+4} \sum_{j'=j-8}^{j-4} g_{i',j'}$ of their neighbors below to t_{win}.

3.1.3 Retrieving Field Boundary

We assume that the robot is located somewhere on the field. The field boundary can therefore be represented as a height value b_i for every column i of the subsampled image, which we retrieve from the binarized field color image of the previous section. We process columns independently, starting at the bottom. When we encounter four consecutive zeros, we assume that the field boundary is reached.

3.1.4 Smoothing and Gap Filling

The field boundary so far is a rough field boundary estimate with gaps and peaks caused by uneven illuminations and imprecise color classification. We can improve by smoothing the boundary using a 1D Gaussian filter for slightly uneven bins and a median filter for small gaps and peaks.

Finally, we employ a local convex corner algorithm to fill-in the remaining gaps without including larger unwanted regions like the opponent robot's body or goal posts. If the local convexity

$$v_i = 2 \cdot (b_i - b_{i-1}) - (b_{i+1} - b_{i-1}), \tag{1}$$

is zero, b_i is on the line between its left and right neighbors, if v_i is positive, it is above this line and it is below that line if v_i is negative. The locally convex hull of the field is then determined by connecting all b_i with $v_i \geq 0$. Iterating this scheme will eventually lead to a globally convex hull, but for our purposes one iteration suffices. The result of the field boundary extraction is exemplarily shown in Figure 2.

3.2 Preprocessing and Skeletonization

We only process the classified white pixels inside the estimated field boundary. Here, a low-pass filter is applied to make sure that each line cross-section has only a single maximum-valued pixel. Skeletonization [2] is used to reduce the line width to approximately one pixel. Unlike morphological methods which start from the border of the line and iteratively erode it, we use a ranking operator, which directly finds the skeleton in the middle of a line. The operator observes 3×3 pixel regions to decide if the central pixel belongs to the skeleton. Pixels having a white-count of zero do not belong to the skeleton, but to the background. For all other pixels, the number $c_{i,j}$ of neighboring pixels (8-neighborhood)

having an equal or higher white-count is computed and if this number is less than three, the central pixel is added to the skeleton. Figure 1(b) visualizes two resulting skeletons.

3.3 Placement and Connection of Nodes

Nodes are placed starting at peaks of the skeleton ($c_{i,j} = 0$). This emphasizes crossings, which tend to appear as peaks in the skeleton. Note, however, that crossing detection does not depend on correct node placement at this stage. Next, nodes are placed at least two pixels apart at pixels belonging to ridges ($c_{i,j} = 1$ and $c_{i,j} = 2$).

The nodes now need to be connected to represent field lines. First, the connection structure of the skeleton is reconstructed by inserting connections where 3×3 regions of nodes overlap or touch on the skeleton. In order to insert the few remaining connections necessary to recover the original field lines, more global information of adjacent connections is used. Lines are stretched by moving end-nodes to the last pixel of the skeleton and degree-0 nodes are split to fill-in the gaps. Finally, candidate connections are created and evaluated according to continuity, closure, and simplicity. Specifically, the distance between nodes should be small and the whiteness on the line between nodes should be similar to whiteness of nodes. Furthermore, we place restrictions based on the degree of nodes, ruling out crossings of degree greater than four and crossings which do not result in continuous lines. Examples are shown in Figure 1(c); we refer the interested reader to [2] for details.

4 Feature Extraction

We can now easily extract features such as crossings or lines from the node structure and verify them in the image. A typical image and its detected features is displayed at the top of Figure 2.

4.1 Line Crossing Detection

The node connections produced so far represent the original structure of the field lines in the image with many details and discretization noise. To extract the key structure, the lines are smoothed and nodes are removed at locations of low curvature. Short lines ending in junctions are eliminated and junctions that are close together and connected are merged to form a crossing. When merging and moving nodes, we make sure that the new nodes are placed on the skeleton to guarantee an undistorted world view. The result is shown in Figure 1(d).

Crossing detection is now dramatically simplified due to the keynode-structure representation. A degree-2 node with a sharp angle between edges is an L-crossing candidate, a degree-3 node is a T-crossing candidate, and a degree-4 node is an X-crossing candidate. To verify whether one candidate represents a real crossing, we first use a state machine to confirm the green-white and white-green color transitions along a circle centered at the crossing. Then, we check whether there is a white path from the crossing to the next neighbors in each direction. Both checks are performed in the sub-sampled color images.

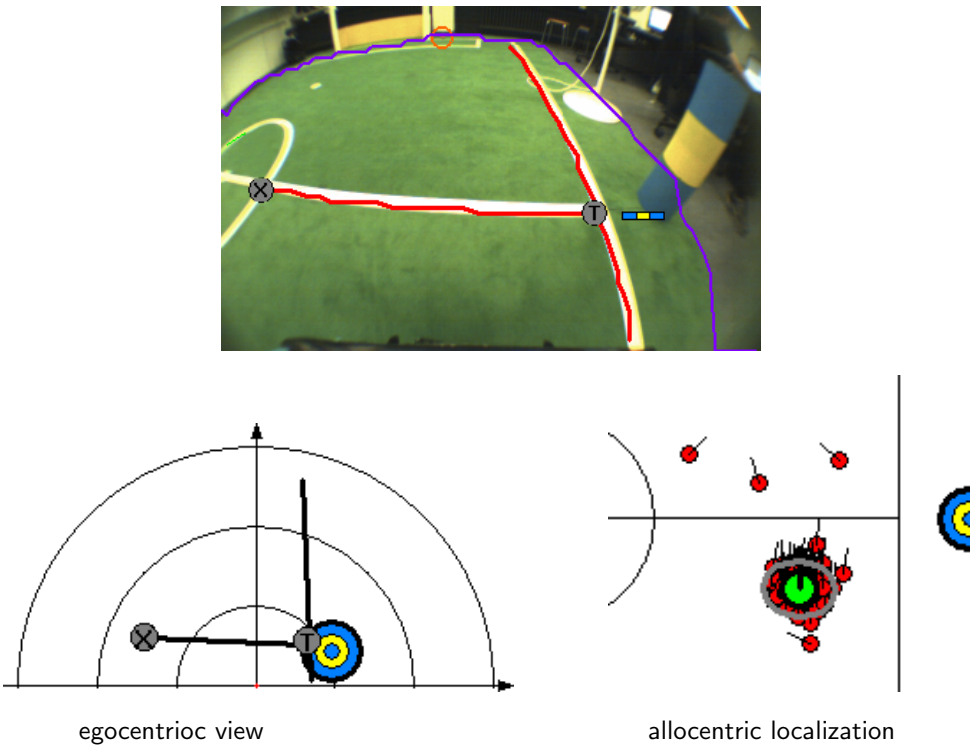

Figure 2: Localization using line and corner features. The top-figure shows an image taken from the robot's front camera. The purple line denotes the detected field boundary. Red lines show field lines used for localization. Detected corners are marked as "X" or "T". Bottom left: egocentric view with landmarks used for localization. Bottom right: resulting localization using the particle filter.

4.2 Field Line Extraction

Starting with the fine-grained, connected graph of observed line segments (Figure 1(c)), we can extract field lines. Here, a field line is a connected set of nodes with degree two and the nodes connected directly to the set that have different degree.

The nodes can be approximately connected by a straight line. Consequently, we first traverse the graph to extract connected components of degree two nodes. Because we rely on wide-angle lenses, straight lines in the world do not result in straight lines in the image. Before proceeding, we therefore calculate undistorted coordinates of the nodes. Each component κ_i is then processed using the split-and-merge algorithm: we fit a line l_i to κ_i using least squares regression [11] on the coordinates of the nodes. The node $n^* = \arg\max_{n \in \kappa_i} \text{dist}(n, l_i)$ with the largest distance to the fitted line defines a splitting point. We split κ_i into two components $\kappa_i^{1/2}$ if the node-line distance is sufficiently large and recursively process the resulting components. Components containing less than three nodes are discarded. During the merging phase, we merge components κ_i and κ_j, if the parameters of l_i and l_j are sufficiently similar.

The final field line in image coordinates is determined by projecting the end points of the component onto the fitted line. We do not use the end points directly, since these can represent part of another line which barely did not pass the splitting threshold. If κ_i contains many nodes, its (possibly wrongly matched) end points will have limited influence on l_i, resulting in a better approximation. Lastly, by mapping both end points through the camera matrix, we can infer the line parameters (distance and angle) in an egocentric coordinate frame.

4.3 Other Landmarks

The other main landmarks—goal posts and center-line poles—extend vertically into the third dimension. Since our camera points downward, these objects do not appear upright in the image. The angle at which they appear largely depends on the angle in egocentric coordinates and thus on the column in the image. Anticipating this, we determine the sums of the blue and yellow subsampled color images along the expected orientations. Every pixel in the subsampled image contributes to exactly one sum. We then search the resulting vector for robust maxima. Only image regions corresponding to a maximum are then analyzed for pixels of the respective color. We further check that the found pixels are really upright by verifying the main orientation (goal posts) and that colors appear in the correct order (center-line poles).

4.4 Determining Egocentric Coordinates

Since the precision of available kinematic models for our robot is insufficient, we instead estimate the inverted camera matrix mapping from undistorted pixel coordinates to world-coordinates on the soccer field plane. Given four correspondences of points in the world plane (here, field corners relative to a known position) and the respective manually selected image positions, we determine the (inverted) plane-to-plane homography [12]. The transformation can be pre-calculated for every image pixel in the subsampled image. This homography, however, is only correct as long as the head is level, which is

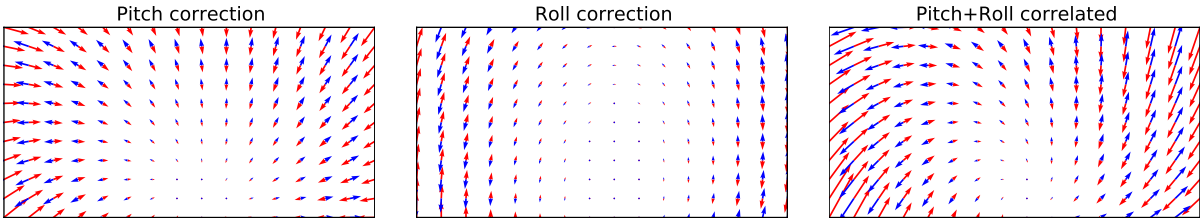

Figure 3: Learned mapping from IMU readings to image offsets. The arrows are learned correction magnitudes and directions of image coordinates for a given pitch and roll. Red/blue represents correction for positive/negative pitch and roll.

violated during walking, where the robot shifts its weight from left to right and slowly tilts back and forth.

Our robots are equipped with an inertial measurement unit (IMU) that allows for estimating pitch and roll of the robot with approximately the same lag as camera frames need for processing. This correspondence enables us to learn a function which corrects the pixel positions w.r.t. pitch and roll. For this purpose, we observed undistorted image coordinates $\mathbf{c}$ of four objects in front of the robot. The initially observed image coordinates $\mathbf{c}^*$ can be regarded as the true ones. We then let the robot walk in place, additionally recording the IMU measurements $\boldsymbol{\phi} = (\phi_p, \phi_r)$. Finally, we find a matrix C that solves the least squares optimization problem

$$\min_C \left\| \mathbf{x}^* - \mathbf{c} - Cf(\mathbf{c}, \boldsymbol{\phi}) \right\|^2 \qquad \text{based on monomials } f(\mathbf{x}, \boldsymbol{\phi}) = \begin{pmatrix} (\phi_p - \bar{\phi}_p)c_0 \\ (\phi_r - \bar{\phi}_r)c_1 \\ (\phi_p - \bar{\phi}_p)c_1 \\ (\phi_r - \bar{\phi}_r)c_0 \\ 1 \end{pmatrix}, \tag{2}$$

where $\bar{\boldsymbol{\phi}}$ is the (sometimes drifting) IMU reading currently estimated for the upright pose. We plot the learned pixel offsets for three pitch/roll combinations in Figure 3. The clear advantage of this approach is that a robot model is not required and all sensor errors are taken into account during optimization. An example of image features projected to egocentric coordinates is displayed on the bottom left of Figure 2.

5 Integration of Features

As described in [7], we use Monte-Carlo localization (MCL, [13]) to estimate the current 3D pose of our soccer robots. The pose is a tuple (x, y, θ), where (x, y) denotes the position on the field and θ is the orientation of the robot. The belief is updated recursively with:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}, \mathbf{u}_{1:t-1}) = \eta \cdot p(\mathbf{z}_t|\mathbf{x}_t) \cdot \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \cdot p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1}, \mathbf{u}_{0:t-2}) d\mathbf{x}_{t-1}, \tag{3}$$

where η is a normalization constant resulting from Bayes' rule, $\mathbf{u}_{0:t-1}$ is the sequence of all motion commands executed by the robot up to time $t-1$, and $\mathbf{z}_{1:t}$ is the sequence of all observations. The

term $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ is called motion model and denotes the probability that the robot ends up in state $\mathbf{x}_t$, given it executes the motion command $\mathbf{u}_{t-1}$ in state $\mathbf{x}_{t-1}$. The observation model $p(\mathbf{z}_t|\mathbf{x}_t)$ is the likelihood of making the observation $\mathbf{z}_t$, given the robot's current pose is $\mathbf{x}_t$. MCL uses a set of random samples to represent the belief of the robot about its state at time t. Each sample consists of the state vector $\mathbf{x}_t^{(i)}$ and a weight factor $\omega_t^{(i)}$. The update of the belief is carried out according to the sampling-importance resampling particle filter. First, the particle states are predicted according to the motion model. For each particle, a new pose is drawn, given the motion command executed since the previous update. In the second step, new importance weights are assigned according to the observation likelihood $p(\mathbf{z}_t|\mathbf{x}_t^{(i)})$. Finally, a new particle set is created by sampling from the old set according to the particle weights. Each particle survives with a probability proportional to its importance weight. This step is called resampling.

In order to not only track a pose, but also allow for global localization, e. g. in case of the "kidnapped robot problem", a small amount of the particles is replaced by particles with uniformly drawn poses. Additional particles are used if pose certainty suddenly drops (Augmented MCL, [14]).

5.1 Crossing Observations

We treat crossing observations similar to other point features on the field (e. g., center of goal, goal posts, marker poles). In contrast to the other point features, however, crossings are not unique. To calculate the likelihood of an observation in the particle filter, we have to make an association decision: for a line crossing observation o, each particle i and all crossings C of the same type (X/T/L), we must calculate the most likely association

$$o' := \arg\max_{c \in C} p\left(c \left| \mathbf{x}_t^{(i)} \right.\right). \tag{4}$$

While the result of the calculation can be re-used in the second step of sampling-importance resampling, evaluating the observation model repeatedly is expensive and limits the number of particles. We can considerably reduce the size of C, however, by considering the egocentric orientation of T and L-crossings. Consider, for example, a particle at the centerline looking towards the yellow goal and observing an L-crossing oriented towards the particle. This particle is quite unlikely to observe the L-crossings next to the blue goal, which are oriented exactly opposite allocentrically. It is also unlikely to see the L-crossings next to the yellow goal which point towards the yellow goal. Consequently, we split the set $\mathcal{L}$ of L-crossings into four sets $\mathcal{L}_{45^\circ}, \ldots, \mathcal{L}_{315^\circ}$ containing two L-crossings of equal global orientation each. A pre-calculated, coarse lookup table $\mathbb{R}^2 \times \{45^\circ, 135^\circ, 225^\circ, 315^\circ\} \mapsto \mathcal{L}$ then associates an observation position on the field and an allocentric, oriented L-crossing observation with the closest L-crossing. We proceed similarly with the T-crossings, but since the distance between different T crossings is large, a lookup table mapping positions to closest T-crossings is sufficient. For X-crossings, we calculate the most likely crossing on a per-particle basis according to Equation (4) using the observation model. For an egocentric observation (d_o, β_o) with range d_o and bearing β_o and an expected egocentric position

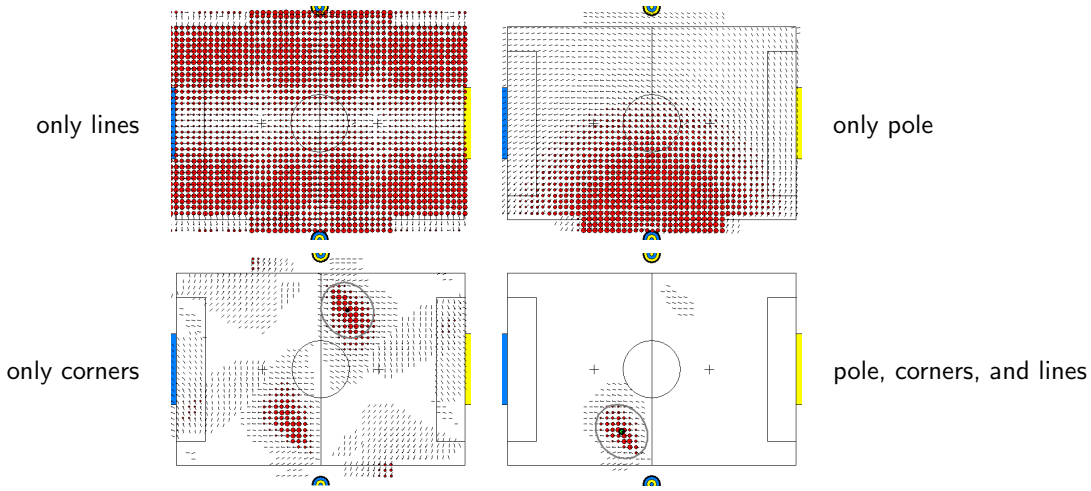

Figure 4: Particle filter belief visualization based on observed landmarks. We used particles on a 3D-grid and show sizes and orientations based on the likelihood. For the given camera frame, we integrate all observations and select the most likely orientation θ^* for each planar position (x, y). We then show a particle sized in proportion to the likelihood $\mathbf{x} = (x, y, \theta^*)$ at position (x, y). The features used are the same as in Figure 2.

$(d_e^{(i)}, \beta_e^{(i)})$ relative to the current particle i, we define the observation model to be

$$p_{\text{point}}\left(o \,\middle|\, \mathbf{x}_t^{(i)}\right) \propto \exp\left(-\frac{\left\|d_e^{(i)} - d_o\right\|^2}{2(\sigma_d + \lambda d_o)^2} - \frac{\left\|\beta_e^{(i)} - \beta_o\right\|^2}{2\sigma_\beta^2}\right), \tag{5}$$

where σ_d and σ_β represent the uncertainty of a range and bearing measurement, respectively. Note that the uncertainty of distance measures increases for far away objects to compensate for the unknown pitch and roll angle of our camera. Figure 4 (top right) shows the belief resulting from the observation of a point landmark.

5.2 Line Observations

In our current system, we ignore lines which are short and far away in terms of distance of the line to the robot. For each remaining line o represented by the length of its dropped perpendicular l_o, distance to closest observed point d_o, and expected angle β_e, we evaluate the observation model

$$p_{\text{line}}\left(o \,\middle|\, \mathbf{x}_t^{(i)}\right) \propto \exp\left(-\frac{d^2\left(l_e^{(i)}, l_o\right)}{2(\sigma_l + \lambda d_o)^2} - \frac{\left\|\beta_e^{(i)} - \beta_o\right\|^2}{2\sigma_\beta^2}\right). \tag{6}$$

Here, $d(\,\cdot\,,\,\cdot\,)$ depends on the oriented distance: in contrast to point landmarks discussed above, the orientation β_o represents the angle of the observed line, not the angle of a polar coordinate. As a result, a simple observation model, which does not take into account oriented distance, would assign equal likelihood to the situation where the robot observes the line behind itself and in front of itself, although the position of the line in front of or behind the robot can be directly inferred from the observation. We

therefore set in Equation (6)

$$d(l_e, l_o) = \begin{cases} \|l_e - l_o\|_2 & \text{if } \langle l_e, l_o \rangle > 0 \\ \infty & \text{else,} \end{cases} \tag{7}$$

which eliminates high likelihood of the implausible situation. In contrast to corner observations, we cannot make a distinction between the seven long lines in our world. We use Equation (4) to determine the most likely match on a per-particle basis. Note, however, that due to monotonicity of the exponential function, for the arg max computation in Equation (4) it is not necessary to evaluate the likelihood in Equation (7) completely. Instead, we apply $\exp(\cdot)$ after the association has been made and rely on the minimum argument of $\exp(\cdot)$ for the association decision itself. In Figure 4 (top left), we visualize the belief resulting from the observation of the two lines in Figure 2.

5.3 Combined Observation Model

To limit the influence of false positive observations, we ensure that all observation likelihoods are larger than some uniform threshold. We further incorporate observation confidence values ρ_j from our vision system such that uncertain observations o_j have less influence on the final distribution than confident observations in the same frame:

$$p\left(o_j \left| \mathbf{x}_t^{(i)} \right.\right) = \alpha_{\text{uni}} p_{\text{uni}}\left(o_j | \mathbf{x}_t^{(i)}\right) + \alpha_{\text{normal}} p\left(o_j | \mathbf{x}_t^{(i)}\right),$$

where $\alpha_{\text{uni}} = \alpha_{\text{base}} + (1 - \alpha_{\text{base}}) \cdot (1 - \rho_j)$ and $\alpha_{\text{base}} \in]0, 1[$ is a uniform floor. We further set $\alpha_{\text{uni}} + \alpha_{\text{normal}} = 1$ and $p_{\text{uni}}(o_j | \mathbf{x}_t^{(i)})$ to be the Lebesgue measure of the observation range. Assuming independence of observations, as typically done in MCL, the observation likelihood of a particle then amounts to the product of all single observations

$$p_{\text{comb}}\left(\mathbf{z}_t \left| \mathbf{x}_t^{(i)} \right.\right) \propto \prod_{l \in \mathcal{L}} p_{\text{line}}\left(l \left| \mathbf{x}_t^{(i)} \right.\right) \prod_{o \in \mathcal{P}} p_{\text{point}}\left(o \left| \mathbf{x}_t^{(i)} \right.\right), \tag{8}$$

where $\mathcal{P}$ includes corners and other point landmarks such as goal centers and poles marking the end of the center line. The combined belief resulting from observations of point and line landmarks is shown in Figure 4 (bottom right).

6 Results

On the 1.3 GHz onboard computer of our robots, our vision system runs at about 24 frames per second using between 250 and 1000 particles (depending on the certainty of the current pose). Table 1 shows the relative timing results. Soccer robots profit enormously from line-based localization. Consider the pose certainty of the robot in our running example. We illustrate the beliefs in Figure 4 by equally distributing particles in a regular 3D-grid. Besides lines and corners, the robot only observes a pole at the side of the field. With color-based localization only, the position on the field is not at all clear from the image. Using only corners, the belief is reduced to two positions. The only colored landmark observed is then enough to further reduce the pose belief to an almost unimodal distribution.

Table 1: Timing of line-detection code in relation to other visual processing tasks.

Algorithm Part	Time (Standard Deviation) in μs			
Color classification	1,578.5	(33.2)		
Pre-processing	421.6	(3.7)		
Object detection	661.2	(94.5)		
Lines preprocessing			128.3	(6.9)
Connect nodes			107.0	(30.3)
Split/merge lines			101.5	(35.4)
Verify crossings			35.0	(20.5)
Particle filter	3,068.3	(1,601.4)		

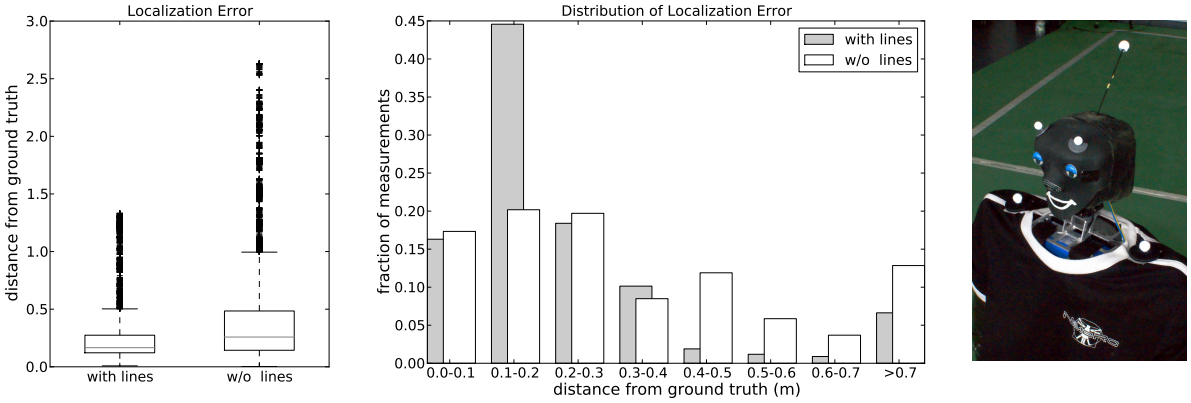

Figure 5: Localization accuracy. Left: Localization error with lines (without lines). The median accuracy is 17 cm (26 cm). Center: Distribution of distances between estimated and ground truth pose. Right: TeenSize Robot "Dynaped" with attached infrared-reflective markers.

6.1 Effect of Lines on Localization Accuracy

To further quantify the performance of our algorithm, we let our TeenSize robot "Dynaped" walk for about 5 minutes across the field using remote control. Robot and field are configured to be rule-conformant to the rules of 2010, namely, the robot only uses a single camera, only the posts of the goal are colored and center-line pole sizes are reduced. We fix the movable head of the robot to look straight at all times to eliminate possible effects of active vision. In addition to estimated poses, we record ground truth positions using a twelve-camera Optitrack motion capture system that tracks infrared-reflective markers attached to the robot's torso and head. The experiment was performed twice, once using line and crossing detection and once without, with similar trajectories. Both trajectories contain about 10,000 poses. The average estimated speed was 20 cm/s (with lines) and 22 cm/s (without lines); we removed 10 % (7 %) of the recorded frames where the motion capturing tracking failed. We then determined the percentage of recorded frames for which the distance between the robot's estimated pose to ground truth was in some

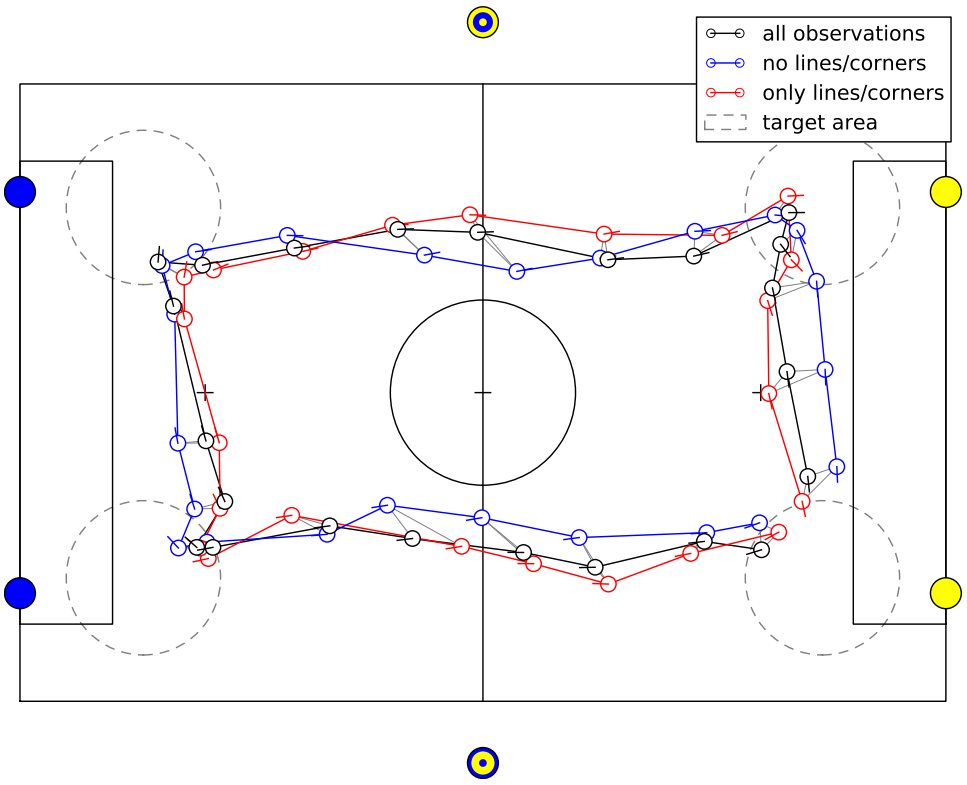

Figure 6: Pose tracking without line observations, with line observations, and using all available features—based on data recorded online while walking from target area to target area autonomously.

interval. The result is depicted in Figure 5. Note that without lines only 66 % of the recorded poses are within 40 cm of the ground truth while with line and crossing detection 89 % are in this range. With lines, the median localization error was 17 cm, which is less than the average step length. This error is also within the range of the robot's trunk movements which are due to walking-based weight shifts, which can be seen as an error of the motion capturing process.

6.2 Lines-Only Pose Tracking

In a second experiment, we set up four target areas on the playing field. Our robot autonomously walks from target area to target area, while we record observed camera frames, IMU data, and motion commands. Due to limited onboard memory, we only record one complete circle. We then process the video in three configurations: Using all available observations, using no line and corner observations, and using only line and corner observations. Since without colored markers global localization is ambiguous, we initialize the pose for all runs manually and then use the methods described in Section 5 to track the pose without global localization. Figure 6 shows the resulting tracks. All conditions yield qualitatively similar results. Unsurprisingly, the trajectory estimated from all features is in between the other two trajectories, which is supported by the quantitative measures presented in Table 2. Most notably, however, is the result that we can track our pose without the use of colored landmarks with an average deviation

Table 2: Differences between corresponding pose tracking estimates depending on the features used.

Condition	Reference	Error	
		Euclidean (m)	Angle (rad)
all observations	no lines	0.17	0.09
all observations	only lines	0.14	0.07
no lines	only lines	0.26	0.15

of 0.14 m, which is less than the error of the complete system compared to ground truth.

6.3 Integrated Evaluation in Soccer Games

Until 2009, the TeenSize robots competed in 1 vs. 1 Dribble-and-Kick—a contest with relatively controlled conditions. Since RoboCup 2010, the TeenSize robots are playing 2 vs. 2 soccer games. This poses new challenges to the visual perception of the game situation, as robots come close to each other, which calls for reliable obstacle detection and occlusion handling, and the robot poses in games include unfavorable poses on the side lines or at corners, where few landmarks are visible. Furthermore, the robots must re-localize themselves when re-entering the field after a fall.

To make the setting more realistic, the size of side poles was reduced, and the coloring of goals was restricted to the goal posts. In addition, in 2011, the TeenSize field was enlarged to 9×6 m, which increased the average distance to landmarks.

Although all these changes made localization harder, we could adapt our probabilistic localization to the new conditions and still localize the robot reliably. Figure 7 shows the localization on the enlarged TeenSize field during RoboCup 2011.

Robust localization was one of the key features for the reliable performance of our TeenSize robots, which allowed them e. g. the precise autonomous positioning for kick-off. Consequently, our robots won the international RoboCup Championships in 2009–2011 and were elected in 2010 for the Louis Vuitton Best Humanoid Award.

7 Conclusion

In this work, we introduced a line-detection algorithm that efficiently recovers the structure of the lines on a soccer field. We first described how to find the boundary of the field. White points within this region are then skeletonized and a simplified graph structure is retrieved from the skeleton. The graph structure has advantageous properties: field-line corner candidates are represented by nodes and field-lines are represented by edges. This graph structure is then used to verify linear components and crossings and to determine their parameters. Due to the graph representation, clustering or averaging of noisy locally estimated line parameters can be avoided. Finally, we showed how lines and crossings

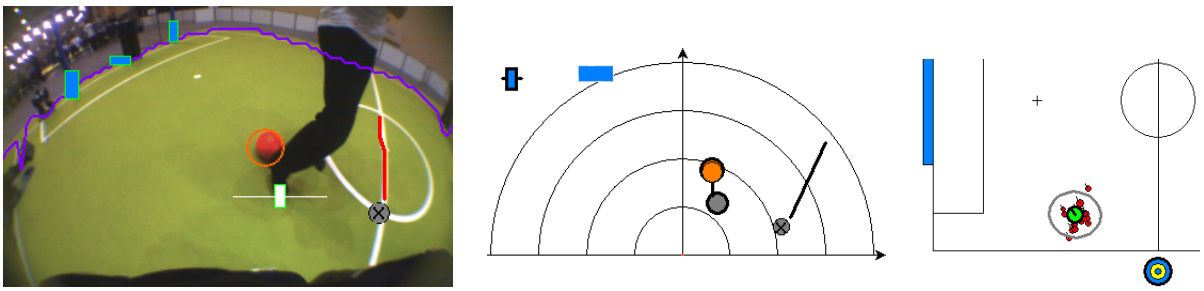

Figure 7: NimbRo 2011 perception and localization. Left: Enlarged TeenSize field with detected goal, ball, obstacle, X-crossing, and center line. Center: Egocentric world view of the robot. Right: Localization given the perceived landmarks.

can be used in Monte-Carlo localization. We proposed robust observation models and optimizations to speed-up association decisions.

The developed system runs with a frame rate of 25 Hz on a small onboard computer. A systematic evaluation of the localization accuracy on the real robot showed smaller differences to ground truth when lines and corners were used. Indeed, pose tracking was possible with a high accuracy even when only line observations were used, which greatly reduces our dependence on a color-coded soccer environment. To the best of our knowledge, this is the first line-only pose tracking system for humanoid soccer robots.

The algorithms described were used on our KidSize robots in 2009 and on the TeenSize robots that won RoboCup for the last three years in a row.

Acknowledgement

This research has been supported by German Research Foundation (DFG), grant BE2556/2-3.

REFERENCES

[1] H. Kitano and M. Asada, "The robocup humanoid challenge as the millennium challenge for advanced robotics," *Advanced Robotics*, vol. 13, no. 8, pp. 723–736, 1998.

[2] S. Behnke, M. Pfister, and R. Rojas, "Recognition of handwritten digits using structural information," in *Proceedings of the International Conference on Neural Networks (ICNN)*, Houston, USA, 1997, pp. 1391–1396.

[3] N. Henderson, P. Nicklin, A. Wong, J. Kulk, K. Chalup, R. King, H. Middleton, S. Tang, and A. Buckley, "The 2008 NUManoids Team Report," in *RoboCup SPL Team Descriptions*, Suzhou, China, 2008.

[4] M. Sridharan and P. Stone, "Real-time vision on a mobile robot platform," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Edmonton, Canada, 2005, pp. 2148–2153.

[5] T. Röfer and M. Jüngel, "Fast and robust edge-based localization in the Sony four-legged robot league," *Lecture Notes in Computer Science*, vol. 3020, pp. 262–273, 2004.

[6] A. Bais, R. Sablatnig, and G. Novak, "Line-based landmark recognition for self-localization of soccer robots," in *Proceedings of the IEEE Symposium on Emerging Technologies, 2005*, Islamabad, Pakistan, 2005, pp. 132–137.

[7] H. Strasdat, M. Bennewitz, and S. Behnke, "Multi-cue localization for soccer playing humanoid robots," *Lecture Notes in Computer Science*, vol. 4434, pp. 245–257, 2007.

[8] M. Lauer, S. Lange, and M. Riedmiller, "Calculating the perfect match: an efficient and accurate approach for robot self-localization," *Lecture Notes in Computer Science*, vol. 4020, pp. 142–153, 2006.

[9] T. Whelan, S. Stüdli, J. McDonald, and R. Middleton, "Line point registration: A technique for enhancing robot localization in a soccer environment," in *Proceedings of 15th RoboCup Symposium*, Istanbul, Turkey, 2011.

[10] *NAO-Team Humboldt 2009*, Graz, Austria, 2009.

[11] J. Kenney and E. Keeping, *Mathematics of Statistics, Linear Regression and Correlation.* Van Nostrand, 1962, ch. 15.

[12] A. Criminisi, I. Reid, and A. Zisserman, "A plane measuring device," *Image and Vision Computing*, vol. 17, no. 8, pp. 625–634, 1999.

[13] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo localization for mobile robots," in *Proceedings of International Conference on Robotics and Automation (ICRA)*, Detroit, USA, 1999, pp. 1322–1328.

[14] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics.* The MIT Press, 2005.