Localizing a robot by the marking lines on a soccer field

Felix v. Hundelshausen, Raúl Rojas

Department of Computer Science, Free University of Berlin
hundelsh@inf.fu-berlin.de

Abstract *In this paper we show how precise localization of a RoboCup[1] soccer robot can be achieved by perceiving and recognizing the soccer field's marking lines. We apply the method within our omnidirectional vision system, but it can be easily transferred to other setups.*

The method is based on detecting and matching features like the circle in the mid of the field, corners, parallel lines and other features. In contrast to other methods which directly match features to a model, we introduce a new layer which mediates between the observed features and the global model. Features are first registered to this mediating layer and the layer itself is matched to the model. The method even works with a bad distance calibration, which is important since we want to use the method as a basis for automatic calibration. The method runs in real time.

1 Introduction

The aim of RoboCup is to foster the development of robots that can act in our real world. One primary issue is a flexible perception. But at the moment we are far away from flexibility. Most teams use color information for localization of their robots by recognizing the goals or posts (see figure 1). In this paper we show how the robot can be localized

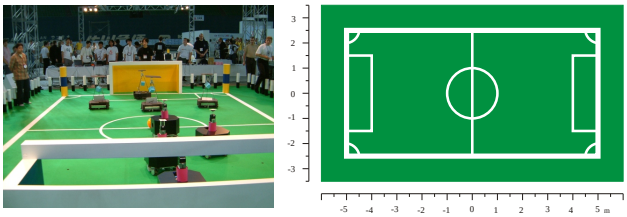


Figure 1: On the left, a photo of the world championships in Fukuoka (Japan). On the right, a model of the marking lines on the playing field by which we want to localize our robots.

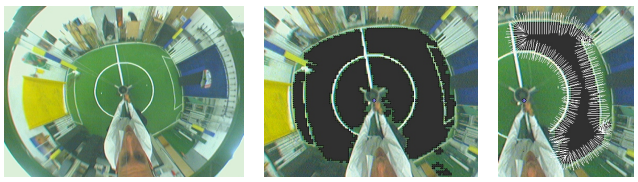
by recognizing the marking lines on the playing field. They consist of circles, corners, parallel lines and other features which will be defined later. Our localization procedure is based on detecting and matching these features.
Localization is a much studied problem, for a review see [9][15][8]. One can distinguish between absolute localization, which determines the absolute position in a coordinate system within a map or a model and relative localization in which we recover the relative movement of the robot between two time points without knowing where the robot actually is in the model. The work by [11] is an example of relative localization using laser range scans. The key idea is to register a scan to a previous scan and obtaining thereby the relative movement. To carry out the registration the method calculates tangent lines at the scan points which help to define correspondences between them. However, no higher features like corners or arcs are detected and used for the registration. On the other hand there are global localization methods, the most robust being based on particle filters like [8]. Here the idea is to represent a possibility distribution of the robot's position by particles. When some knowledge about the relative movement of the robot is available (either by odometry or by some relative localization method), the particles are moved accordingly. When some feature is observed then the particles are weighted with the probability that the feature could have been observed from its position. Finally, they are resampled which can be seen as removing each particle and generating new ones in its closed neighborhood according to the weight of the original particle.
We want to combine both methods. The relative method [11] has been developed for laser range scans. We modify it to work with the contours of the marking lines extracted from the images. We also modify the method in that we recognize higher features such as corners, circles, parallel lines and other features. We introduce a new layer which represents some kind of temporary map in which features are registered, accumulated and even higher ones recognized. The detection of these features is valuable for two reasons. First, it makes the registration process of the relative localization more robust as we will explain later and second it yields high-level observations for the global localization. With high-level observation we mean features that give a strong hint to the robot's position. They give a low probability to most particles of the monte carlo localization method[8], and a high probability to few particles. The circle in the mid of the playing field together with its mid-line is an example. Detecting this feature leaves over only two possible positions for the robot.
Although we provide a solution for a specific problem in the RoboCup domain, there is one important insight which is general: There is a clear correspondence between the robot localization problem and object recognition. Consider

[1] RoboCup has been founded in 1997 with the aim to foster artificial intelligence by robotic soccer competitions. In the midsize league four against four robots play on a playing field of size 10x5m. For more information on RoboCup see http://www.robocup.org

a fixed camera that observes a moving object. The task is twice: Reveal the movement of the object and recognize the object. In primate vision, these two tasks are believed to be processed in two different pathways, called the "Where" and the "What" pathway[13]. In the localization problem the corresponding questions are: What is the actual movement of the robot (relative localization) and where is the robot within a general map (global localization). We shall show that both relative localization and global localization can elegantly be combined when detecting high-level features.
The remainder of this paper is organized as follows: Section 2 describes how the marking lines are efficiently extracted from the images. Section 3 describes the features, their detection and registration. Section 4 introduces the idea of a mediating layer between the observed contours and a global model. Section 5 shows experimental results. Finally, section 6 concludes the paper.

2 Extracting the Line Contours

We use a catadioptric setup where the camera is directed upwards looking into a convex mirror yielding omnidirectional images as depicted in figure 2. The robot has an embedded processor to perform the visual and behavioral processing. The computation must run in real time.
To efficiently extract the contours of the marking lines we use the region tracking algorithm we proposed in [14]. We discovered that the lines can be easily detected when tracking the green regions of the playing field, and searching along short lines orthogonal to the boundary of the tracked regions for green-white-green transitions. In principle, one does not need color information and can search for dark-light-dark transitions instead. Figure 2 illustrates this idea. We finally obtain a list of line sequences. Each line sequence



(a) (b) (c)

Figure 2: (a) The omnidirectional image of a robot in the mid of the soccer playing field. (b) The regions between the lines as obtained by the tracking algorithm. (c) One region as an example: A detector is applied along lines perpendicular to its boundary.

itself is a list of points. The number of points within a line sequence is typically between 1 and 120. In contrast to laser range scans which are used in[11] the information of our line contours is richer. The reason is that a range scan is a list of points, ordered by angles, and at each angle there is at most one point. Describing a range scan in a polar coordinate system, it can never occur that a contour lays between the origin and another contour, since the laser beam measures only one distance at each angle. This is different with the contours we extract from the images. Here, it frequently occurs that lines are arranged one after another. Informally speeking, if the lines would be walls, then a laser range scan would only detect the closest walls, but our method would even detect the walls behind them. Figure 3 shows a typical example of the contours we extract from the images. As can

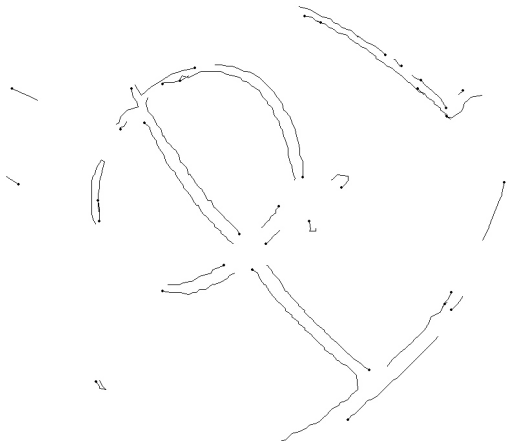


Figure 3: Extracted contours of the marking lines, with the robot being located near the mid of the playing field. The first point of each line sequence is emphasized with a small bullet.

be seen, some lines are detected twice, each of them having been detected from one of the two opposite regions which neighbor the lines. We will recognize later that this is an advantage for the rapid detection of certain corners.

Of course, there are other ways to detect the line contours. For instance one could apply some edge detector [2] to the images and then link the edges to form line sequences [4]. Another approach which is adopted in [5] uses the hough transform[10] to determine lines. However our way of proceeding is much more efficient, because our region tracking and line extraction algorithm only accesses about 10 percent of image information within each frame. Moreover, the lines obtained by our algorithm are much more accurate (curves are not approximated by relatively long lines but by a sequence of closely spaced points) and complete as can be seen by comparing figure 1 d) of [5] and figure 3.

3 Features

Different types of features can be detected in the marking lines. They form a hierarchy which can be implemented in an object orientated way. Figure 4 shows the class diagram.

3.1 Feature Detection

In this section we describe the feature recognition process. The process involves several stages. The contours stores *nLines* line sequences whose points are stored in the array *p*. The index of the first and last point of line sequence i is *startIndex[i]* and *endIndex[i]*. The array $groupId$ assigns a number to each line. Lines with the same number belong to a group. As mentioned aforehead we obtain the contours by the region tracking algorithm [14] and they are grouped initially. Lines that neighbor the same region belong to the

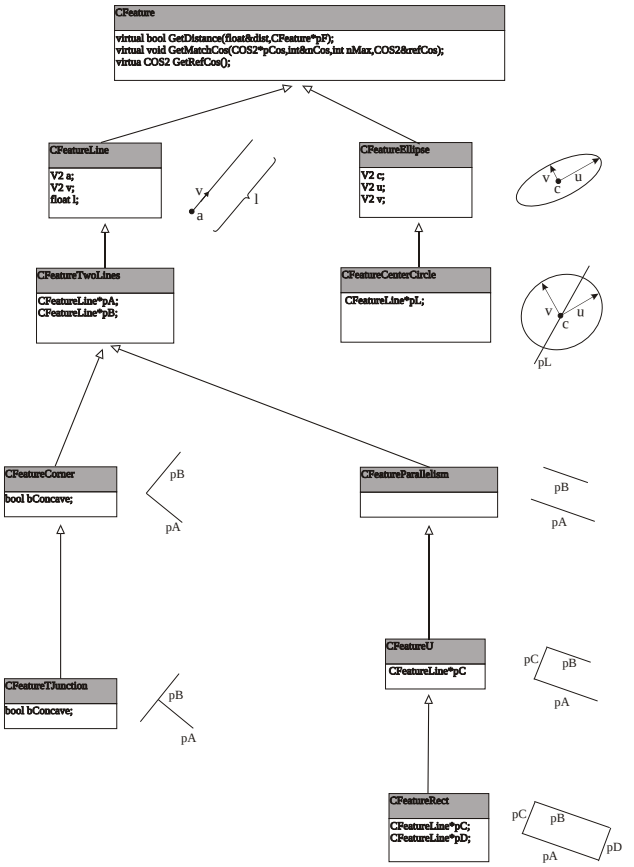
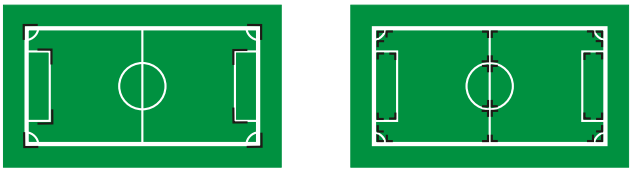


Figure 4: Class Diagram of the feature hierarchy. The geometric meaning is depicted to the right of each class. Vectors are denoted by *V2* and coordinate systems by *COS2*.

same group. Grouping, seeing it from the perspective of divide and conquer, greatly speeds up our feature detection, since most features can be detected within each group.
According to figure 1 lines can be classified into two groups: straight and non-straight lines. We do not want to speak of curved lines here, because there are non-straight lines, which aren't curves (i.e. a line sequence which forms a corner). Although even the straight lines in the perceived contours are slightly curved due to the distortion of the catadrioptic image system, the difference in amount of curvature still allows the distinction between originally curved and straight lines. If the start and end point of line i have indices s_i and e_i, then we calculate the mid-index $m_i = \frac{s_i+e_i}{2}$ and obtain the mid-point m of the line. Here we assume that the points of the line sequence are equally spaced which is true for the extraction algorithm we use. Then we consider the vector v_a from the start to the point and the vector v_b from the mid point to the end point and we calculate the angle between these vectors. Using an empirical threshold we classify each line to be straight or non-straight. This simple method does not always classify the lines correctly, but the further processing also does not rely on the correct classification of all lines. Next, we consider the non-straight lines. We want to know whether they are smoothly curved as the ones due to the center circle of the playing field, or if they have some high curvature points. There are numerous methods to calculate the curvature of line sequences, for instance see [3]. However, for the sake of efficiency we use a very simple method. For each point on line sequence i we calculate a local curvature measure by calculating the angle of two adjacent vectors, meeting at the actual point, the start of the first vector is w points before and the end of the second vector is w points after the actual point. We determine local maxima of the curvature values. The values can be negative or positive depending on the direction of curvature. If they exceed some threshold, we have a hypotheses for a corner. We distinguish two different types of corners according to the region from which they stem: Convex and concave corners. Figure 5 shows where each type can be found in the model of the lines. Interestingly, there are only few concave corners, they can be detected at a very early stage and they yield a strong hint to the robot's position. After having detected the corners, we split the line contours at their local maxima of high curvature. The contours now consist of lines, which are either straight or curved. They do not contain any high-curvature points any more. The parts of each line having been split are again classified by the aforemention method to be straight or curved.
Next, we want to detect the center circle if present in the contours. In a first approach we fitted an ellipse [6] to each curved line and tried to cluster the ellipses. However, we achieved better results by first grouping the curved lines and then fitting an ellipse to them. For the grouping we applied the domain-specific knowledge that there is only one big ellipse and we just grouped the five longest curved lines being longer than a certain threshold. Of course, for a more general solution this processing step must be replaced. After



(a) (b)

Figure 5: (a) concave corners (b) convex corners

the lines have been split we consider straight lines that exceed a certain length as line features. Typically, there are no more than about 6 line features within a contour. From these lines together with the yet detected corners we recognize higher features such as U-features, T-junctions, parallelisms or rectangles. Their recognition is based on any-to-any comparisons but since their number is low we can afford these quadratic running times.
In the following we want to describe the registration mechanism. Therefore we have make some definitions.

3.2 Reference coordinate system

When representing points in homogeneous coordinates [7] a two-dimensional coordinate system C can be represented

by a 3×3 matrix:

$$C = \begin{pmatrix} u_x & v_x & p_x \\ u_y & v_y & p_y \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

A point p_{global} can be transformed into the local system by:

$$p_{local} = C p_{global} \quad (2)$$

Each feature defines a unique two-dimensional *reference coordinate system*. This reference coordinate system can be obtained by calling the function *GetRefCos()* of the respective feature.

3.3 Matching distance function

Each feature class defines a matching distance function *bool GetDistance(float&dist, CFeature*pF)*. The function stores the matching distance in *dist* and returns a boolean value indicating whether the match is valid. This is typically the case when the distance is below a certain threshold. Otherwise the match is considered as an outlier.

3.4 Feature map

Suppose that we have detected some features within the last contours. We want to register the features to another set of features. In particular, we want to register a certain type of feature only to the same type. Therefore, when representing a set of features, we store them in separate lists, one for each type. We refer to this construct as feature map. More formally we define.

Definition 1 *A **feature map** M_F over a set $F = \{T_0, T_1, ...T_n\}$ of n feature types is a list of n lists, $M = \{L_0, L_1, ..., L_n\}$, with list L_i containing n_i features of type T_i,*
$L_i = \{f_0, f_1, ..., f_{n_i} \mid type(f_i) = T_i\}$

3.5 Registration

Assume that we want to register the feature map $A = \{L_0^A, L_1^A, ...L_{n_A}^A\}$ against a feature map $B = \{L_0^B, L_1^B, ...L_{n_B}^B\}$. We register each pair of corresponding list separately. To register list L_i^A against list L_i^B we successively consider the features in list L_i^A. For each of them we pass through the features in list L_i^B and we call the matching distance function. If the function returns false, we do not consider the actual correspondence. We choose the feature f_r^B in L_i^B as correspondence for a feature f_k^A in L_i^A which has the lowest value.

For each corresponding pair of features we want to obtain a transformation which tells us how feature f_k^A must be transformed in order to fit to feature f_r^B. We represent this transformation by specifying a pair of coordinate systems. For some features, for instance corners or ellipses, the transformation is unique and we represent the transformation by their reference coordinates systems. If we translate and rotate the reference coordinate system of f_k^A to be equal to the reference coordinate system of f_r^B, then the two features also fit. However, for some features the transformation is not unique. Consider a line segment from point R to point S, for instance. As illustrated in figure 4 we represent the line by a point p, a direction vector v normalized to length one and its length l. We can define the reference coordinate system of a line to be located at the mid of the line and its x-axis to be v. But the same line segment can be defined in two different ways: Either $v = norm(S - R)$, where *norm* expresses the normalization to length one, or $v = norm(S - R)$. This means that having two line segments, which should be registered, there are two possible matchings, which are depicted in 6. Therefore, we allow a matching pair of features

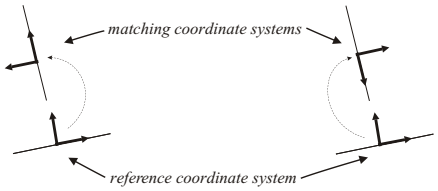


Figure 6: There are two possibilities to match two lines. The coordinate system of the bottom line is its *reference coordinate system*. It remains the same. To the left and right the two possible *matching coordinate systems* of the upper line are depicted.

two yield several transformations. Each feature implements the function *void GetMatchCos(COS2*pCos,int& nCos,int nMax,COS2& refCos)*. When the possible transformations between feature f_k^A and f_r^B should be calculated, we call that function, with *refCos* being the *reference coordinate system* of f_k^A, *pCos* being an array where to store the resulting *matching coordinate systems*, *nCos* being a variable to be increased by the number of returned systems and *nMax* indicating the maximum number of systems the array *pCos* can store. From the n returned coordinate systems we obtain n transformations Q_i from *refCos* to the respective system $pCos_i$. They are calculated by:

$$Q_i = pCos_i refCos^{-1} \quad (3)$$

Finally, we store all obtained transformations in a list, each having an additional weight which depends on the type of feature. The weight specifies the influence of the respective transformation to the desired overall transformation, which consist of a relative translation T and rotation w of the robot. To obtain these quantities we first have to perform a change of coordinate systems for the individual transformations. The question is, if feature a has to be transformed by Q to fit to feature b, what is the corresponding relative translation T and rotation ω of the robot that yields the same transformation? (see figure 7)

If we represent Q by

$$Q = \begin{pmatrix} r_x & s_x & t_x \\ r_y & s_y & t_y \\ 0 & 0 & 1 \end{pmatrix}, \quad (4)$$

then $T = \binom{t_x}{t_y}$ and $\omega = arctan_2(\binom{r_x}{r_y})$. We finally obtain a list of relative translations T_i and rotations ω_i. The overall translation and rotation can be calculated by voting or clustering scheme[12].

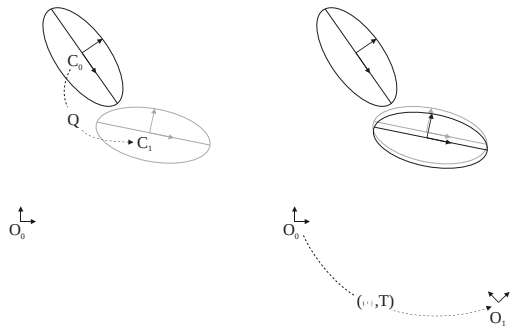


Figure 7: (a) The matching transformation of the ellipse features is given by Q. (b) We are interested in the relative rotation ω and translation T of the robot's local coordinate system O_0 which yields the same transformation.

4 The Feature Accumulation and Registration Map

Suppose that the robot stands near the penalty area whose marking lines expose two concave corners directed towards the inner of the playing field. Consider one of the corners and let's denote the two perpendicular lines by which it is formed A and B. We extract the marking lines of two successive images with the robot not moving, call them C_0 and C_1. It can happen that we neither can detect the corner in C_0 nor in C_1, although parts of it are represented in both contours. It might be that in C_0 line A is occluded and in C_1 line B, for instance by another moving robot. This effect can happen with other features than corners and it might not be caused only by occlusion but also by some detection failure of the low-level vision. Therefore, if we conceived a feature detection algorithm which worked only isolatedly on the extracted contours, we would never detect some features. This observation is one of the reasons to establish a temporary map, in which feature are accumulated. The map stores line A of the first frame and the line B of the second frame. Now we can detect the corner in the map. We refer to the map as "feature accumulation and registration map" (FAR-map). In the above example, the features that accumulate are the lines, and the accumulation enables the recognition of higher features. That is, we have a second feature detection stage, working with the features in the FAR-map as input, detecting higher features and integrating these to the map.

We also perform the relative localization within this layer. That is, we register the features of the actually extracted contours to the features stored in the layer, much like [11]. The difference is that our registration is based on different types of features. That is we register lines to lines, corners to corners, ellipses to ellipses, etc... and finally obtain an overall relative translation and rotation which tells us how the actually perceived contours have to be moved, that they best fit to the features in the FAR-map. Finally the actual features are themselves integrated.

The FAR-map consists of a list of queues for each feature type. Each queue has a fixed size k (k=10 in our case) and it stores the recent k detected features of a certain type at a certain location in the FAR-map. When a feature is detected in the contours we determine the queue, whose features are closest (by means of its matching distance function) to the current feature. Then we determine the relative transformation as described above and integrate the feature in the queue. At the same time we extract the last feature from the queue. We maintain the queue for two reasons. First the queue serves to store the change of geometry of a certain feature at a certain location over the last k time steps and we can use this information two calculate velocities for instance. Second we use the queue as a robustness detector. Only when a feature has been detected successively m of k times we consider the feature as robust and we use it for global localization then. This robustness constraint is very important because false positives are discarded.

Both the features and the registration process form some kind of hierarchy. On the bottom there are the lines. If not higher features can be detected, the FAR-Layer just consists of line queues and the registration is a line-to-line registration, similar to [11]. Corners and parallelisms, both consisting of lines form the next layer in the hierarchy. If they are detected, registration also takes place at a higher level. The advantage is that the registration distance can be greater for higher features, since they are not as frequently and confusion is more unlikely.

5 Results

Figure 8 shows the features having been detected from the line contours and figure 9 illustrates how the FAR-map is matched to the model. We can extract and process the

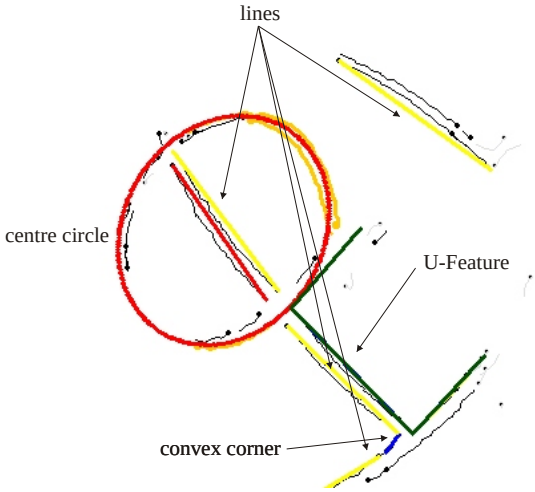


Figure 8: Detected features from the line contours.

contours at full frame rate (30 fps). The images have a resolution of 640×480 pixels. Relative localization works up to speed of approximately $2m/s$ with the playing field having a size of $10 \times 5m$. This information is important because the maximum speed is higher for scaled settings. The maximum rotational speed is approximately $500°/s$. We do not need odometric information and approximate localization is possible even with a wrong distance calibration.

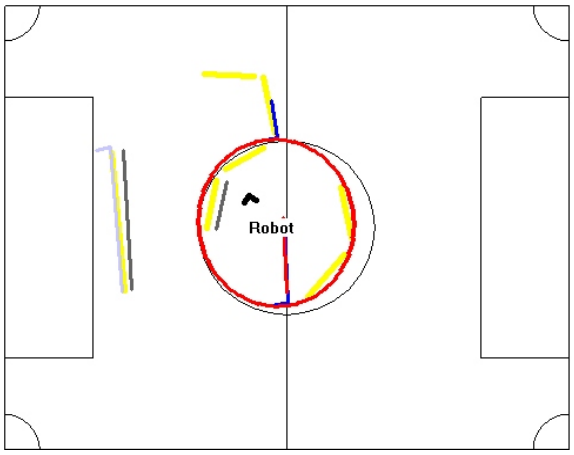


Figure 9: The features in the FAR-map are matched to the model.

6 Conclusion

We have proposed a feature based localization system. Before we developed the system we used a method like [11] for relative localization. We matched a collection of short line segments to a predefined model.
With the new system our relative localization is more robust and the robot can move with higher speed. The problem of relative localization methods based on closest-distance relationships [1][11] without recognizing higher features is that when moving with high speed confusion of the primitives they are based on can take place easily. Suppose for instance that the robot is rapidly moving over two parallel lines having a distance of approximately 1 meter. Then a registration method that does not recognize the two lines as an entity for its own and which uses the closest distance as correspondence criterion can easily confuse the two lines. When denoting the two lines with A and B, this is because line A in the first frame can be closer to line B in the second frame, when the robot is moving fast. Another advantage is that we recognize features which can be used effectively for global localization: The center circle with the mid-line or the concave corners are two examples. These features suffice to localize our robot on all locations on the soccer field. Another important issue is that the high level features allow us to work with imperfect calibration. Even without calibration we are able to detect the features and to roughly localize the robot thereby. Hence the system can be used as a base for automatic calibration and we are currently working on methods for that. We have introduced the idea of a feature accumulation and registration map. The idea is not to directly match features against a model, but instead to register features to previous detected features and to accumulate them. The accumulation allows the detection of new features and allows to verify the robustness of the detection. Although we apply the method in the RoboCup domain our contribution is more general: It is not only about localization, because when observing a moving object with a fixed camera the same problems arise. On the one hand we want to know how the object moves, on the other hand we want to recognize the object. The movement has to be revealed by registration (relative localization) and the recognition corresponds to the global localization problem.
Our conclusion is: When registering two contours, the relative displacement can be greater when high-level features are detected and the high-level features can also be used for recognition (global localization) at the same time.

References

[1] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[2] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 8:679–698, 1986.

[3] D. Chetverikov and Z. Szab. A simple and efficient algorithm for detection of high curvature points in planar curves. Technical report, 1999.

[4] G. W. Cook and E J. Delp. Multiresolution sequential edge linking. *Proceedings of the IEEE International Conference on Image Processing*, pages 41–44, October 1995.

[5] F. de Jong, J. Caarls, R. Bartelds, and P. Jonker. A two-tiered approach to self-localization, 2001.

[6] A. W. Fitzgibbon, M. Pilu, and R. B. Fisher. Direct least square fitting of ellipses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):476–480, 1999.

[7] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 2 edition, 1990.

[8] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *AAAI/IAAI*, pages 343–349, 1999.

[9] J. S. Gutmann. *Robuste Navigation autonomer mobiler Systeme*. PhD thesis, University of Freiburg, 2000. DISKI 241, ISBN 3-89838-241-9.

[10] P. V. C. Hough. Method and means for recognizing complex patterns. US Patent 3,069,654, December 1962.

[11] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *CVPR94*, pages 935–938, 1994.

[12] R. Ostrovsky and Y. Rabani. Polynomial time approximation schemes for geometric k-clustering. In *IEEE Symposium on Foundations of Computer Science*, pages 349–358, 2000.

[13] L. G. Ungerleider and J. V. Haxby. What and where in the human brain. *Current Opinion in Neurobiology*, 4:157–165, 1994.

[14] F. v. Hundelshausen and R.. Rojas. Tracking regions by shrinking and growing. *Proceedings of the Computer Vision Winter Workshop*, February 2003.

[15] J. Weber. *Globale Selbstlokalisation fr Mobile Service-Roboter*. PhD thesis, University of Kaiserslautern, 2002.