MATRIX: A force field pattern matching method for mobile robots

Felix von Hundelshausen, Michael Schreiber, Fabian Wiesel,
Achim Liers, and Raúl Rojas
Technical Report B-08-03

Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany
http://www.fu-fighters.de

Abstract. In this paper we describe a method for localizing and tracking a mobile robot in the type of field used for RoboCup mid-size soccer robots. Each robot carries an omnidirectional camera which provides a 360 degree view of the robot's surroundings. The only features that we use for localizing the robot are white lines present on the green carpet. These white lines are easy to find and discriminate – they are not too dependent on the color calibration of the camera. The MATRIX method localizes a robot in a field by testing a grid of hypothetical positions for a best fit to the data. Once localized, tracking the robot from frame to frame, in real time, can be done by combining the odometric data with the MATRIX pattern matching.

1 Mobile Robots Localization

Localizing mobile robots in a known environment is easier that localizing robots in a space that has to be explored and mapped first. However, if the sensors used by the robots are noisy or if the computational capabilities of the robot are limited, localizing a robot in a known environment can also be a hard problem. This is the case in the RoboCup environment, where mobile robots compete trying to be as fast, yet as precise as possible. Usually, the robots have only a partial view of the field and color calibration tends to be difficult, due to varying lightning conditions. Also, spectators sitting or standing around the field, can be mistakenly identified with the yellow and blue poles used at the corners of the field. Therefore extensive color calibration is done by all teams before a game starts. Some groups have studied methods that could avoid work even without color information, for example, for finding the ball, but they are computationally expensive and not feasible yet [8].

It would be important to be able to depend less on the color calibration of the cameras and more on certain prominent features of the field. The white lines on the green carpet are a very good example, since they are specially prominent and easy to find. Some RoboCup teams already exploit such features in order to localize the robot. Utz et al. ([9]) work with this kind of information, but map it to Hough space which provides a kind of assessment of the number of

expected features visible on the field. It is a form of pattern matching, using the information available in Hough spaces for lines and circles.

Nevertheless, it is very difficult to leave the localization of a mobile soccer robot alone to the vision system. If the robot gets confusing images (because, for example, another robot is obstructing the camera), then the robot loses its initial position which can be difficult to recompute again – the robot gets lost on the field. Therefore, we need a method for localizing the robot globally on the field, and another method for tracking the robot from one known position to the next [7].

Odometry is a straightforward method for disambiguating vision data. Human soccer players do not look at the field constantly, they have a "feeling" for their current position extrapolated from the last known position. They know in which direction they have moved, and for how long, and so they only need to look occasionally to the field, in order to "recalibrate" their current position. This is also the best strategy for a mobile robot, specially since odometric data tends to be much more precise than data for legged robots. Our mid-size robots can drive forward for about three meters, and the localization error after stopping is just several centimeters (just how many, depends on the velocity and direction of the robot). A combination of odometric with vision data can reduce the load on the control computer enormously. Instead of having to process each frame of a video in search of field features, it would be possible to track the robot during short periods based fully on the reported odometry, while the vision can recalibrate the odometric data periodically, for example once a second. Then only one frame every second has to be fully processed for determining the robot's position, while the odometry is reaching the robot in a continuous stream. Of course, for finding the ball and going for it, it is important to process 30 fps, but only in order to find the orange blob inside the field. Since our localization method is so fast, we can even use it to correct the robot position in every frame.

One problem with odometry data is that, in the case of omnidirectional robots, three wheels produce the desired movement, sometimes by working against each other. The wheels should not slip on the floor, and if they do, the odometry data has to be corrected to take this slippage into account. In this paper we present a method to correct the odometry data.

2 Localization in a flat featured environment

A flat featured environment is one in which lines on the otherwise featureless floor provide position information. This is the case of the RoboCup environment, in which white lines on a green carpet delineate a soccer field of up to 10 meters long by 5 meters wide. We would like to localize the mobile robot using this information. Fig. 1 shows a photograph of the field and the markers used. One goal box is blue, the other yellow. The poles at the four corners are painted yellow and blue.

The frame captured by an omnidirectional camera is processed by our mobile robot, which tracks regions in the field using an algorithm described elsewhere

Fig. 1. A view of the mid-size field used at RoboCup 2002

(cite). The robot then looks for color transitions between green and white, and dark and white. This color transitions represent possible lines on the field, that we want to identify and track. Fig. 2 shows a cloud of points, from the perspective of the robot. The dark spots represent edges between white and green colors, or viceversa. It is easy for a human to match this cloud to a model of the field, but it has to be done automatically by the robot and in only a few milliseconds.

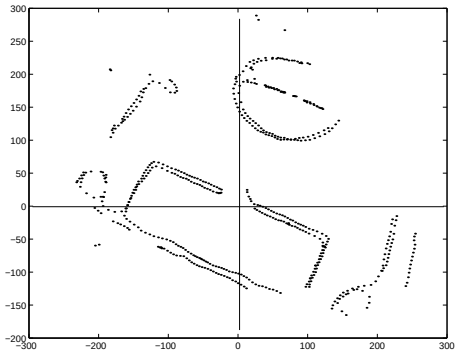

Fig. 2. A cloud of vision data points representing white lines on the field, from the perspective of the robot

2.1 Global localization

If the robot does not have its initial position, we use a pattern matching approach to find the best possible hypothesis. The idea is the following. If we have the cloud of vision points, we can rotate it several times (for example 16 times) and then we can translate each rotated cloud to each of several hypothetical regions of space (see Fig. 3, left). These position hypotheses form a grid on one half of the field (Fig. 3, right). We only use one half of the field, because the other half is symmetrical. The robot can be localized on one side or the other. Additional information has to be used to disambiguate the choice (like the color of the box goals seen).

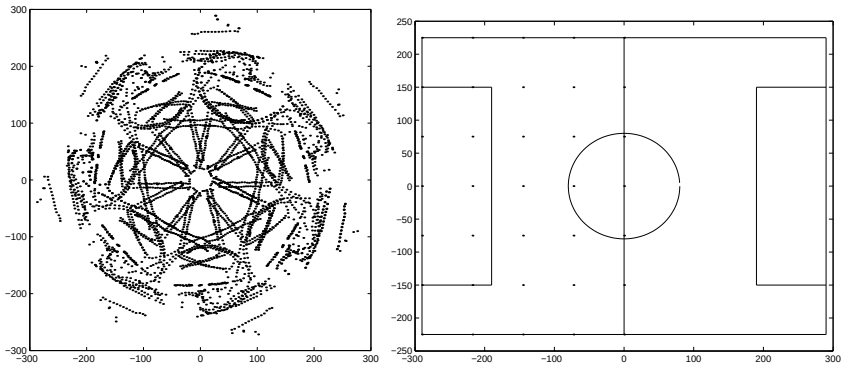

Fig. 3. The points on the field represent the hypothesis tested for global localization. 16 orientations are tested for each position.

Given the rotation and centering of the cloud, the quality of the match to the field is measured by computing, for each point, the distance to the nearest segment or circle in the model of the field. These distances are added for all points, are averaged, and the final number represents the quality of the match. To speed up the computation, we compute beforehand a matrix of distances of every element in a 100 by 70 grid to the segments and circles in the model. Given the cloud of points it then is easy to do a table look-up to associate each point with the distance to the nearest segment. A cloud of points has typically 500 to 700 points, so that the table lookup and averaging takes only a few microseconds. Fig. 4 shows the matrix of "qualities, that is, the distance vector from each point in a grid that covers the field (and also points outside the field) to the nearest model element.

Ideally, we would like to compute the quadratic error associated with a least squares match of the cloud point to the model. However, the minimal distance function is not smooth: midway between two parallel line segments the distance vector changes direction suddenly. It is then difficult to provide an analytic model of the field, that does not contain "if" decisions.

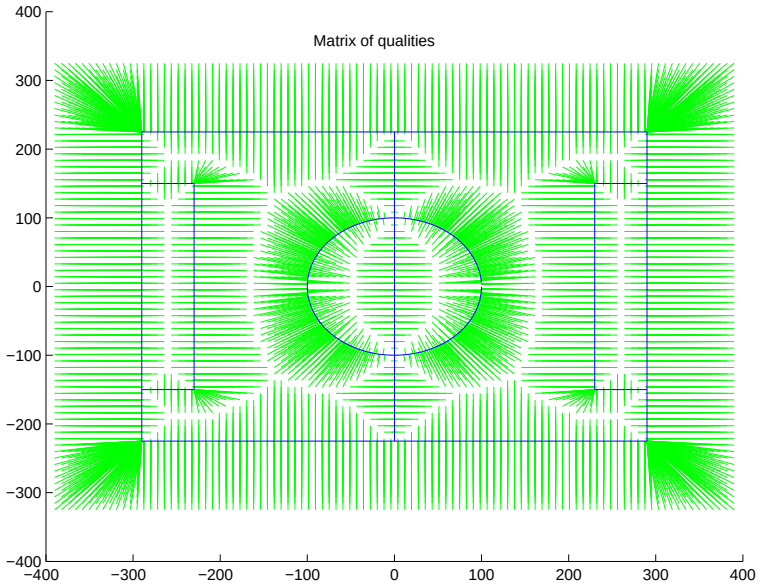

Fig. 4. Matrix of distance vectors to the nearest segment or circle on the field, for every point on a grid laid over the field and its surroundings

2.2 Tracking the robot

Tracking the robot can be done also by using pattern matching. The idea is to start from a known position and orientation of the robot. We rotate the cloud of points by the known orientation and translate the cloud to the known position. The cloud should be a good match of the model. If the robot moves, the next cloud from the camera (repositioned and reoriented as before) will almost match the model, but will have moved. The translation and rotation of the cloud with respect to the model is the translation and rotation of the robot. We need to compute the inverse transformation, in order to recenter the new cloud and make it fit the model as well as possible.

The inverse transformation is computed iteratively. We start from the new cloud of points and compute the total average "force" on each point. The force on a point is the attraction that a segment or circle of the model exert on each point. The force is proportional to the distance to each line element. We add all forces acting on a point, but weighting them in a way that ensures that the nearest line elements have a greater contribution to the estimated force.

If (x, y) are the coordinates of a point and the array $\ell_i,\ i = 1, \ldots, 11.$ represents its distances to the 11 model elements, the weight w_i for each single

distance is computed as:

$$w_i = \mathrm{e}^{(-\ell_\mathrm{i}/10)} / (\sum_{\mathrm{k}=1}^{\mathrm{k}=11} \mathrm{e}^{(-\ell_\mathrm{k}/10)}$$

The constant 10 in the exponential's argument provides good results for distances measured in centimeters. It must be adjusted by hand to the units of the model.

The total force f_i on the i-th point is then

$$f_i = \sum_{k=1}^{k=11} w_k d_k$$

where d_k is the vector that joins the point (x, y) with the nearest point of the k-th model's element.

The idea behind smoothing the force function is that when two line segments compete agains each other trying to "pull" on a point, the forces will equilibrate if the distance to two segments is very similar. It is also easy to transform this weighting function in an approximation to the minimum selection rule used for measuring the quality of a model, by just adjusting the proportionality constant in the exponential's argument.

As in the case of the matrix of qualities, we precompute the matrix of forces, using a grid laid over the field model. Fig. 5 shows the result. The small line segments represent the total force on each point on the grid.

Computing the total force on a cloud of points takes a few microseconds. We also compute the angular momentum around the center of mass of the cloud. We then pull the cloud of points in a direction proportional to the total force, with a maximum pull of 10 centimeters. We rotate the cloud with an angle proportional to the angular momentum with a maximum rotation of 2 degrees. These maximum values were selected because the robot will not move farther away that 10 cm or rotate more than 2 degrees in one frame. These constants can be adjusted to fit the robot speed. Also, several iterations of the force algorithm can be run, to ensure that the computation converges to the appropriate position and orientation estimation. Fig. 6 shows a cloud of points and the forces acting on them, as an illustration of the method.

3 Monte Carlo MATRIX localization

The method that we finally implemented for tracking our robots is the following.

At the start of a game we estimate the robot position by using a global match to the quality grid. We can also determine the position of the goal boxes, yellow and blue, and this disambiguates the side of the field, the robot is in. A global match of the robot's position takes a few milliseconds. Fig. 7 shows the result of matching a cloud of points to a grid with only 35 hypotheses and using 16 orientations. A few force field iteration displace the cloud of points to a very good match, that provides the initial position and orientation of the robot.

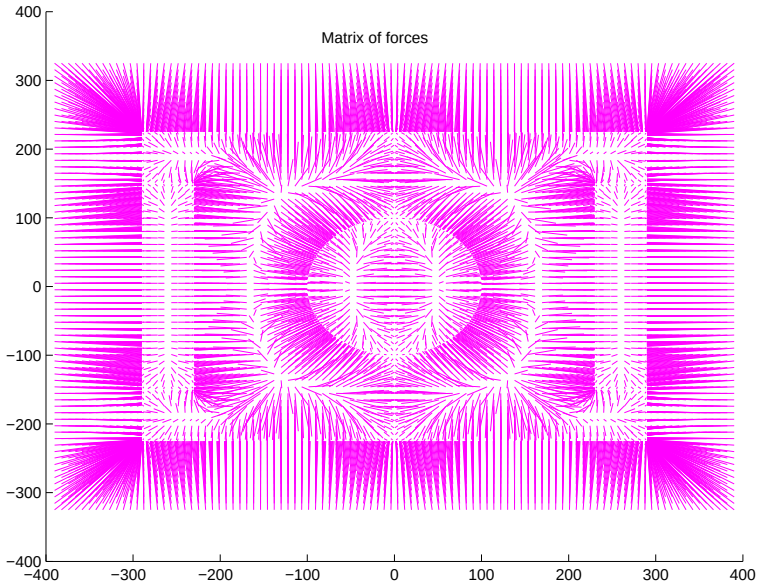

Fig. 5. Matrix of weighted forces to the segments and circle on the field, for every point on a grid laid over the field and its surroundings

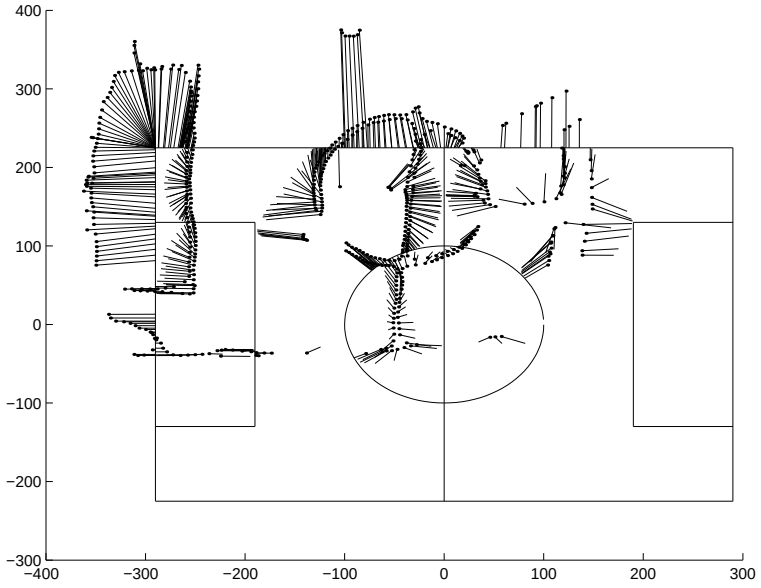

Fig. 6. A cloud of points and the forces exerted on them by the model elements

Even when a small part of the while lines are visible, the matching is good. Notice also that in this experiment the calibration of the distance function for the omnidirectional mirror is far from being perfect, yet the matching found is excellent.

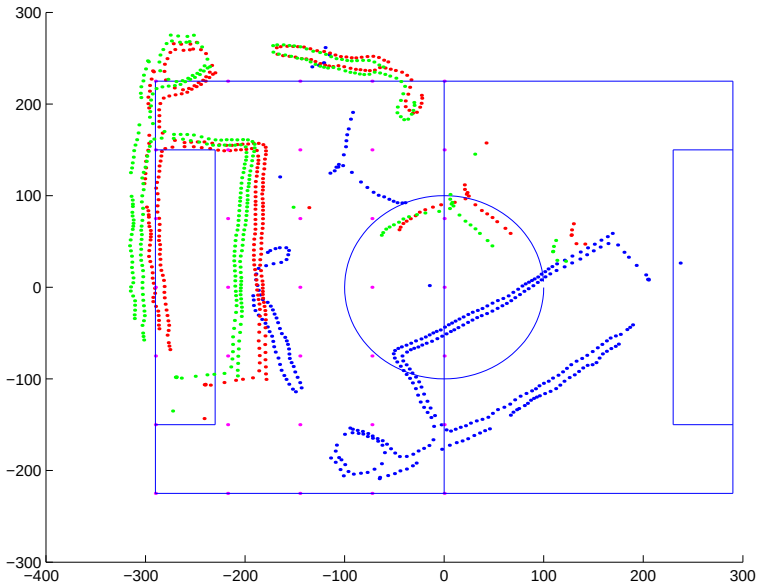

Fig. 7. Result of a global match. The initial cloud is centered around the origin. Two clouds show that the left rectangle box has been matched to the data. One of the matched clouds is the initial guess, the second is a corrected cloud, computed by the force field method.

Once the position of a robot on the field is known, we track the robots using odometric and vision data. Our approach resembles a Kalman filter implemented with Monte Carlo sampling of the vision data.

Fig.8 shows the odometric data over a period of several seconds. As can be seen, the data is of good quality, since the robot position does not leave the field (this would happen if the error was large, the robot position would then drift away from the field).

For every vision frame we obtain an odometric reading from the robot. The odometry provides a displacement and rotation. From the old position and orientation (x, y, θ) we compute the new position and orientation (x', y', θ') using the odometry data. We then produce 20 random positions and orientations, normally distributed and centered around the new estimate. We compute the quality of each position and select the best. We then recompute the position and orientation of the robot by adding a fraction α of the change in displace-

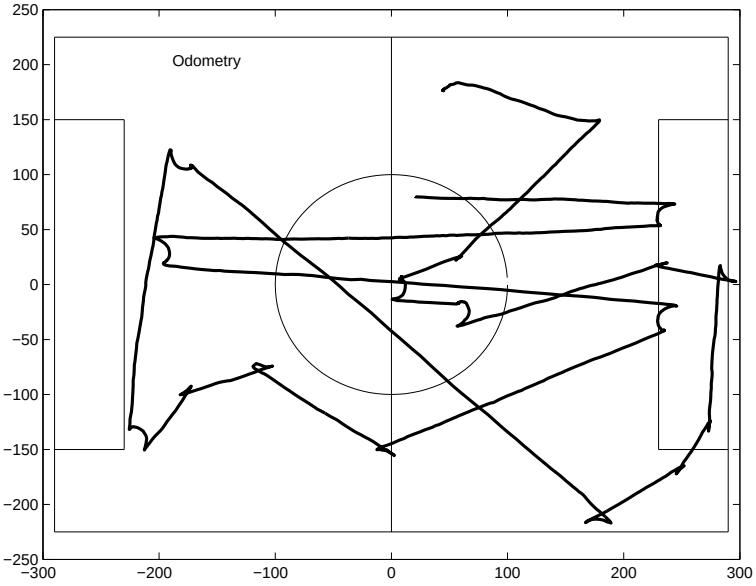

Fig. 8. Movement of a robot as reported by the odometry

ment and orientation of the robot (from the computed odometric state to the hypotheses selected) to (x', y', θ'). This is the final new estimate and we process a new frame.

Notice that we can also improve the estimate, by doing a force field matching after a hypotheses has been selected. We can also periodically check the estimated position against a global matching of the vision data. In case that the robot starts to lose its actual position, a global matching will recalibrate the robot to a best new estimate of its position.

4 Correcting the odometry

Omnidirectional robots move activating three wheels. In the case of our robots, the two front wheels are placed at an angle of 150 degrees relative to each other (instead of 180 degrees, as in two-wheeled robots). The third wheel, in the back of the robot, is parallel to the line joining the centers of the two front wheels.

When the robot is moving forwards, only the two front wheels rotate, and the third wheel slides passively (small wheels on the omnidirectional wheels let them slide almost without friction in the direction perpendicular to the wheel). When the robot moves forwards, but with a certain rotation of the third wheel, the result is a forward and lateral displacement of the robot.

In the case of our robots we have detected that fast forward motion, combined with lateral motion, sometimes leads to slippage of the third wheel. The

odometry data then overestimates the amount of lateral displacement. The slippage is due to the fact that the two forward wheels bring the small wheels on periphery of the third wheel into fast motion, which reduces their grip on the carpet surface, due to the different dynamic friction coefficient. A reduced grip translates into reduced traction and wheel slippage.

However, the overestimation of the odometric error must be proportional to the forward velocity of the robot. Therefore, we collected odometric data, allowing the robot to move from a known position to another, at different velocities. The table of real versus estimated displacement provides a way to train a predictor that can correct the odometric data when the robot is moving. The predictor provides a correction factor f between 0 and 1. The real contribution to the global displacement forf a wheel is given by its rotation, times the correction factor.

Notice that the correction has to be done for every wheel. Every time the robot moves in such way that one of the wheels experiences a large perpendicular effective movement, its effective grip is reduced and its traction too. The ideal case, for minimal slippage, is when all wheels cooperate moving the robot, that is when the robot rotates in place. In this case, there is no perpendicular displacement of the wheels and the correction factor for the odometry can be set to 1.0 for each wheel.

The important factor is therefore the amount of displacement of the robot normal to each wheel. When the robot is moving forward at a velocity v, and with wheels placed at an angle of 150 degrees, the perpendicular displacement in relation to the wheels is given by $sin(\Pi/12)$, which is 0.2588. The front wheels experience a velocity component which is around $v/4$ of the velocity component experience by the third wheel. It is to be expected that the odometric error is much larger for the third wheel.

The odometric error does not need to change linearly with the perpendicular motion of each wheel. Friction is a nonlinear phenomenon, and therefore the best and simplest approach is to work with tables that correct the pulses transmitted by each wheel, according to the expected displacement of the robot. This in turn, affects the expected displacement, which in turn affects the correction. The process converges after two or three iterations to an appropriate value.

Fig. 9 shows a robot whose predicted displacement is the vector (x,y). The displacement is obtained by multiplying a matrix M with the vector of motor pulses $(t_1, t2, t3)$. It is necessary to know the time elapsed in order to have an estimate of the velocity. If the odometry data is obtained regularly, the time difference is fixed and the motor pulses and (x, y) are proportional to the velocity. It is easy to compute the orthogonal projection of (x, y) to each wheel.

For the first wheel at an angle $\Pi/12$, it is $x\cos(\Pi/12) + y\sin(\Pi/12)$. For the second wheel at an angle $\Pi - \Pi/12$, it is $-x\cos(\Pi/12) + y\sin(\Pi/12)$. And for the third wheel, it is simply $-y$. The absolute values of these quantities are proportional to the velocity and provide a first correction to the pulse values t'_1, t'_2, t'_3. This correction can be used to compute the next correction and so on.

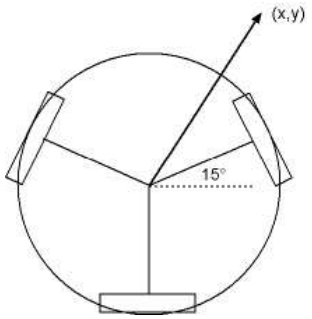

Fig. 9. Displacement of a robot and placement of the three wheels

If the robot rotates when it moves the distance (x, y), then the above projections can be calculated twice: once for the initial orientation of the robot, a second time for the predicted final orientation of the robot. The average of both projections gives a better estimate of the necessary correction factor for each wheel.

(and here we need a table of correction factors, to be done by Felix+Fabian?)

5 Results

The MATRIX method has been implemented in our mobile mid-size robots. The main advantage of this method, compared to others, is the low computational load that it imposes on the host computer. Also, discriminating the white points on the field from other colors is easier that segmenting colors. The method is less dependent in a correct camera calibration than in the case of other localization techniques.

Our method is computationally cheaper than the approach followed by Hanek et al. ([9]), since we do not map the data points to a Hough space. Our matching reduces to a table look-up on a matrix of qualities, to assess the total quality of a match, or to table look-up in a table of forces, when adjusting the known position iteratively.

Goord odometric data is very important, so that tracking the robots does not require to check too many hypotheses for evey frame. Therefore, we have shown how to correct the odometry data in the case of omnidirectional robots.

In the future we envision a combined system in which the vision adjustment helps to calibrate both the distance function of the camera and the odometric data. A slightly decalibrated robot could then enter the field and calibrate its distance function and odometry by just moving around in the field, to known positions, and checking its vision and odometric data. The calibration time for mobile robots would then be dramatically reduced.

References

1. Rojas, Raúl: Neural Networks – A Systematic Introduction. Springer Verlag, Heidelberg, 1996.
2. Simon, Mark; Behnke, Sven; Rojas, Raúl: Robust Real Time Color Tracking. *Lecture Notes in Artificial Intelligence* **2019** (2001) 239–248.
3. Rojas, Raúl; Behnke, Sven; Liers, Achim; Knipping, Lars: FU-Fighters 2001 (Global Vision). *Lecture Notes in Artificial Intelligence* **2377** (2002) 571–574.
4. Fox, Dieter, Burgard, Wolfram, Dellaert, Frank, and Thrun, Sebastian, Monte Carlo Localization: Efficient Position Estimation for Mobile Robots, *Proc. 16th National Conference on Artificial Intelligence*, AAAI99, July 1999.
5. Fox, Dieter, Burgard, Wolfram, and Thrun, Sebastian, Markov Localization for Mobile Robots in Dynamic Environments, *J. of Artificial Intelligence Research* 11 (1999) 391-427.
6. Thrun, Sebastian, Probabilistic Algorithms in Robotics, Technical Report CMU-CS-00-126, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA, 2000.
7. Ribeiro, Isabel, and Lima, Pedro, "Markov Lokalization", (tutorial), Instituto Superior Tecnico, Lisbon, Portugal, 2002.
8. Hanek, Robert, Schmitt, Thorsten, Buck, Sebastian, Beetz, Michael, "Towards RoboCup Without Color-Labeling", in Kaminka, Lima, Rojas (eds.), *RoboCup 2002: Robot Soccer World Cup VI*, Springer-Verlag, 2003.
9. Utz, Hans, Neubeck, Alexander, Mayer, Gerd, and Kraetzschmar, Gerhard, "Improving Vision Based Self-Localization", in Kaminka, Lima, Rojas (eds.), *RoboCup 2002: Robot Soccer World Cup VI*, Springer-Verlag, 2003.