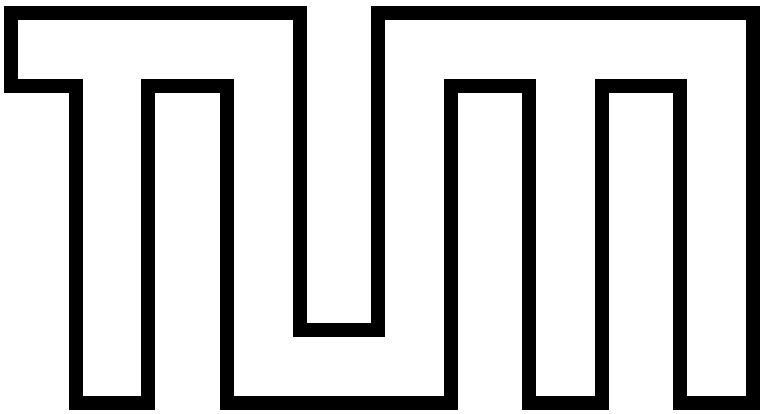


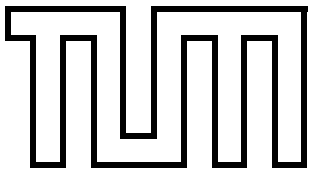
TECHNISCHE UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK

Bayes Pose Estimation by Modeling Color Distributions

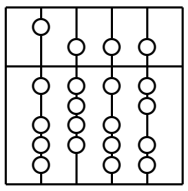
Diplomarbeit

Dirk Neumann



FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN
UNIVERSITÄT MÜNCHEN



Forschungs- und Lehreinheit IX
Bildverstehen und Wissensbasierte Systeme

Bayes Pose Estimation by Modeling Color Distributions

Diplomarbeit

Dirk Neumann[1]

Lehrstuhl	:	Prof. Bernd Radig
Betreuer	:	Dipl.-Inf. Thorsten Schmitt
Abgabedatum	:	November 2003

[1]Email: dirk@caltech.edu

Hiermit versichere ich, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmitteln verwendet habe.

Pasadena, 10. November 2003

_____

(*Dirk Neumann*)

Abstract

Bayes filtering techniques have been successfully used for a variety of tasks, for instance to navigate a robot through a museum [27] or to track diverse objects [10]. Many of these probabilistic search techniques are based on Markov and Monte-Carlo methods and allow the robust tracking of multiple hypotheses.

The basis of every Bayes-based filtering technique is its likelihood model. For vision based based tracking, it computes for an image and for each vector in the state space the probability that the image could have occurred in that state. We here explore the possibility of using the distribution of pixel color as the only criteria to compute the image likelihood. Thus, we circumvent the need for classical image processing methods such as color classification, segmentation, or landmark detection. Instead, a sketch of the expected image is generated for each pose using a simplified version of a voxel-based rendering technique. Based on Gaussian models of the different colors in RoboCup, the robot's pose could be evaluated by looking at the current camera image. Experiments show that the metric is robust to translation, but very sensitive to the orientation dimension.

A Monte-Carlo-Markov-Chain (MCMC) approach is used to integrate and track the robot pose distribution over time. The implications of the properties of the likelihood model for the particle filter will be discussed and in addition a more robust, conventional landmark-based metric will be suggested.

Contents

1 Introduction

Camera-guided orientation and localization for mobile robots is currently a very active field of research. Different classes of algorithms were proposed to solve the navigation problem by using landmarks, by tracking contours, or with statistical filters for spatial transforms such as Hough transformation. The common precondition of these algorithms is a model of the surrounding that is used to compute the current position based on the known location of the landmarks or other properties of surrounding. The necessity of a *global* model can, to some extend, be circumvented if the localization algorithm is used to propagate (and thereby possibly correct) an initial start position. Iterative methods usually search for local changes of image features and uses this information to constantly update the estimate.

The objective of the algorithm presented here was to investigate the possibility of removing two major constrains from the problem of self-localization: the sensitivity to changes in illumination, and the dependency on application-specific features for different surroundings. Instead of using topological or spatial operators to compute the position information directly, the unprocessed images are used to localize the robot. For this purpose, the robot possesses a 3-dimensional model of the RoboCup environment and a fast voxel-based technique is used to compute expected images for various likely robot poses. These images are then compared with the current image obtained from the on-board camera. The differences between the sketch of the expected image and the actual image is modeled with the help of precise color statistics that are obtained from a sequence of images at different positions in the environment. The statistical color model describes for the different objects how likely different color tones occur. The likelihood that an

image corresponds to a sketch can then be computed from the likelihoods of the single pixels in a straightforward manner.

The center of the pose estimation algorithm is a probabilistic Monte-Carlo-Markov-Chain (MCMC) algorithm that starts with a random subset of all possible poses, computes for each pose the sketch of expected environment, and estimates for each selected pose the likelihood that the sketch is "congruent" with the image. In a second step the current pose subset is reevaluated and the most likely poses are selected for the next step. The probability that a pose is selected for the next iteration is thereby proportional to the likelihood of the observed differences between the sketch and the image.

The odometry information from the robot is used to propagate the pose estimates in accordance with the movements of the robot. In agreement with most MCMC implementations, additional spatial noise is added to the pose estimates with each iteration. Thus, the single pose estimates can be seen as representations for these (randomly generated) paths that best correspond to the observed image sequence. This MCMC approach and the Kalman filter are optimal for Gaussian probability distributions (measurements with an observed Gaussian error), but the Bayesian approach can be seamlessly applied to non-linear problems too (for a comparison in RoboCup see e.g. [19]).

However, the major drawback of the MCMC algorithms is their dependency on the smoothness and broadness of the likelihood function used to evaluate the single estimates. If the likelihood distribution is to narrow or can drastically change between different iterations, the estimates will not converge, or may even converge to temporal stable, but non-optimal (and thus incorrect) paths. Unfortunately, the camera images dramatically change with rather small changes in the orientation of the robot. Hence, the resulting likelihood function is too steep for

the MCMC algorithm and it may only be reliable used to either predict small local pose changes, or to globally localize the robot if the orientation is known from other sources (e.g. a landmark).

Therefore, the voxel-based MCMC localization could not be reliably used in the RoboCup environment. For the application in the RoboCup competition a simple landmark-based MCMC variant was used instead. It employs classical color classification and topological image operators to extract the positions of predefined color-based landmarks (corner flag posts and the goals). The differences between the expected and observed positions of the landmarks are then used to form the likelihood function in pose space, and best estimates are selected with a MCMC algorithm. Since this algorithm does not extend the established MCMC image-processing approaches, it will be discussed here only shortly.

The main emphasis of this thesis is on the one hand to describe the probabilistic color model and the voxel-based world model, and secondly, to give an overview of the results of the evaluation of the likelihood function. Although the localization procedure can not be reliable employed in RoboCup, the straightforward comparison of images with rough sketches based on the color discrepancy may be adequate for other applications. For most cases it would be sufficient to estimate the translational coordinates, or to consider small local movements only.

The next chapter will give an overview of different MCMC methods and their current applications in robotics and self-localization. Chapter 3 explains the statistical model of the color metric. The voxel-based world model and the fast sketch rendering technique is explained in chapter 4. The image probability metric is described in chapter 4.3. Chapter 5 gives an overview of Monte-Carlo and Markov-Chain methods. An overview of the implementation in C++ is given in chapter 6. The main properties of the algorithm are evaluated in chapter 7.

2 Related Work

MCMC methods have been previously applied to problems in computational vision, robotics and to self-localization with great success. The most prominent examples are presumably the contour tracking algorithm *Condensation* [10] and the *Minverva* museum robot project [28], [29]. Both algorithms use a Monte-Carlo-Markov-Chain (MCMC) model to track the position, either of a contour in a video sequence, or of the current position of the robot in the museum. Based on the most likely current positions, the algorithms use Bayes models to generate the most likely hypotheses for the next iteration. In the *Condensation* algorithm a local search procedure is used to estimate the agreement of the hypothesis and the image. For the *Minerva* robot the likelihood of a possible position is estimated by comparing the light distribution of the ceiling with the expected values for each position.

Another often chosen approach is to use local color histograms to track objects (e.g. [21]). These techniques are commonly utilized to roughly localize saturated color regions (such as faces) in video sequences, but do not provide a precise and timely estimate of the position since the implicit histogram-based color classification does not extract reliable region estimates. For mobile robotics self-localization, Hough transforms has been often used (e.g. [9]). Another prominent example is the maximum likelihood method developed for the *Sojourner Mars rover* [20]. It determines the best predictions based on the previous pose, feature maps (from vision, sonar or laser range-finder) and a branch-and-bound search that gradually subdivides the three-dimensional state space into a smaller cubes.

In contrast to the previous approaches, the localization algorithm presented in this paper does not use high level image features such as contours or Hough

transforms. Instead a probabilistic color metric is used that is generated from sample image sequences for a particular environment. As it has been shown in Klinker [12], the primary directions of color distributions of illuminated surfaces can be interpreted as the combined effects of shading and highlighting and both can be derived theoretically. In the algorithm presented here, a color-based pixel likelihood is used to compare the observed color values with the predictions of a voxel-based world model. In computer vision, voxel-based techniques have been previously applied for the reconstruction of scenes and objects. Body shapes were constructed from multiple synchronized video stream in [17], and [26] used it to reconstruct a photorealistic scene and circumvented the correspondence problem by the occlusion order of the voxel map.

As Thrun et al. [29] pointed out, MCMC methods are usually simple and astonishingly easy to implement. Good reviews of particle filters are available from Doucet [6] and Neal [18]. Liu [15] and Pitt [22] discuss problematic issues of MCMC methods and suggest extensions. Introductory texts to particle filter can be found in [16] and [1].

3 Color

The algorithm presented here consists of two major components: an image likelihood metric and a Bayes particle filter. The image comparison engine uses an a-priori model of the RoboCup environment that can generate, for each possible pose, a rough sketch of what the camera image should look like. Then the actual image is compared with the generated image and a likelihood value is computed that indicates how similar they are. The second component, the Bayes-based sampling algorithm chooses at each time step, based on the results of previous observations, a random subset of likely poses from the infinite space of possible poses. The selected poses are then used by the image likelihood module to generate sketches at these positions and to compare the sketches with the current image from the frontal camera.

The basic idea of this first image comparison step is thereby to overlay the actual image from the camera with different sketches from various positions, and to precisely estimate for each of these positions the likelihood that the image may be taken at that position. This is done on a per-pixel basis. By quantifying for a particular pixel, how distinct the observed color value is from the expected color in the generated sketch, the likelihood of each pixel can be calculated. The likelihood of the whole image is estimated by averaging the likelihood of the single pixels. The challenge thereby is that it is not sufficient to take an arbitrary and simple metric, for instance to classify the colors into 'white', 'green', 'red' etc. and then compute the number of mismatching pixels, but to get this comparison step, all the other software layers rely on, as precise and reliable as possible.

In this chapter we will therefore present a method that extracts the correct color distributions for the different classes (lawn, own/enemy goal, robot, etc.) from a

set of preprocessed images taken on the RoboCup field and characterizes them in a linear model. For the first step of mapping the image colors to different object classes, a comfortable manual color labeling program is used that was developed as part of the RoboCup project at the Munich University of Technology.

3.1 Bayes Color Model

In the following the spatial relationship of pixels is unimportant, and images are just seen as a stream of (spatially) independent color values. The problem of finding for such a color stream an appropriate color metric can be best visualized by plotting all the pixels of an image according to their color coordinates in a chart (see figure 3.1). In the plot the color values of the different objects (e.g. green lawn, yellowish background, reddish ball) constitute distinct stripes in different regions of the color space. The stripes are primarily oriented along the red-green-blue diagonal (intensity axis) and overlap, at least in the 2D representation, to some extend.

The color model of the algorithm uses for each of the color classes a 9-dimensional linear transformation (translation, rotation, scaling) to model their distributions. Each color class is thereby represented by its mean, its unitary rotation matrix and a diagonal scaling matrix. One may think of the model as 3-dimensional ellipsoids encircling the various color clusters in the RGB color space. Yet, the notion of fixed-sized ellipsoids is a bit misleading. Contrary to classical computer vision approaches, with Bayes-based image processing algorithms there is usually no need to do classification. Rather than having fixed class boundaries, the model transforms the complete color space into a probability space. Therefore, it is not interesting whether a particular, e.g. gray, pixel is decided to be black or

(a) Sample image taken on a AGILO robot.

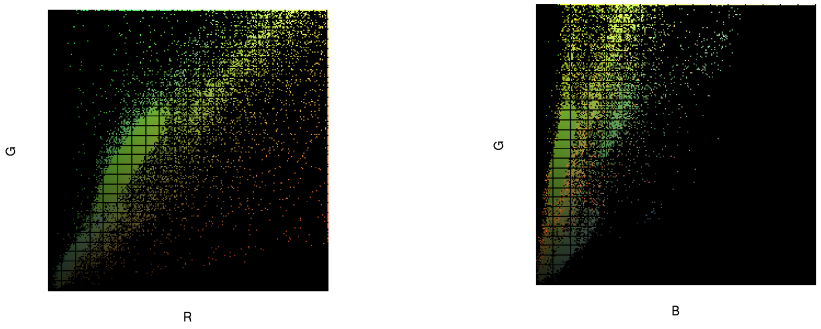


Figure 1: Pixels of the image 1(a) are plotted with respect to their red and green 1(b), and blue and green 1(c) coordinates in RGB color space. If multiple pixels had the same coordinates in the particular color plane their color values were averaged.

white. What is important in a Bayes framework, is to estimate the correct likelihood that an observed (gray) pixel originally is black (or white). Thus from a statistical point of view, the information about the uncertainty of the color is not discarded, but instead is used in the higher levels of processing.

The likelihood model can be directly derived from the empirical color distribution in the RGB space. The model assumes that the different objects have different mean colors (e.g. green, yellow or blue) and that the observed color values devi-

ate from the mean color due to random and normally distributed processes. The major cause of color variation in the RoboCup environment is the illumination. Different degrees of illumination, e.g. due to shading and inhomogeneities of the lighting cause variation in the color values. For illuminated matt surfaces, the reflected color can be computed as the convolution of the reflectance spectrum of the object and the spectrum of the illumination. This reflectance component can be modeled as a random distribution having different orientations for differently colored objects [12]. Besides illumination, color variation is caused by inhomogeneities in the painting, local colored shades, or dirt. These influences are covered by the model's remaining six degrees of freedom.

In order to extract the parameters from the color distribution it is necessary to decide first, what color values belong to which class. We tried a variety of cluster analysis algorithms to automate the process, but then decided to stay with the current manual color selection procedure. Non of the studied algorithms solved the problem reliable. The main problem for all algorithms seem to be the different set sizes of the clusters. For instance the lawn and the background account for the majority of the pixels, whereas the pixels belonging to the ball or the magenta and turquoise markers are very rare. With a hierarchical cluster analysis we could not obtain any reliable separation of the color space, and the best results have been achieved with a k-means cluster analysis when the number of clusters was fixed (though the extracted clusters are often inadequate). The outcome could be enhanced if the (normalized) x- and y-positions of each pixel are included as a fourth and fifth dimension, in addition to the three color dimensions. This favors clusters that are coherent in both, the color space and the images. Yet, the k-means algorithm still tends to separate the large classes (background and lawn) into a light and a dark component and occasionally drops the smaller classes. For

the application in a RoboCup we consider the results of the algorithms as too unreliable.

In the current, manual color classification program the user can take pictures at different positions on the field. In each image the user can draw regions and identifies to which class they belong to. For each RGB value to program counts how often it belongs to each class, and a growing operator is used to fill the whole color space based on the sampled pixels. As a result the program computes a lookup table that indicates for each value in the RGB16 color space to which class the color was most frequently assigned to.

We use these lookup tables to estimate the parameters of the color distributions. First, for each color class the mean RGB value is computed. Then, the single value decomposition (SVD) of the covariance matrix is used to calculate the eigen vectors and eigen values of each distribution. We used the fast implementation of the Lapack library [8]. The SVD decomposes the (covariance) matrix C into two unitary transformation matrices V and U, and a diagonal matrix S containing the eigen values in descending order.

$$cov\,(X) = U\,S\,V^t \tag{3.1}$$

The matrix V transforms the distribution into the corresponding eigen space. If the transformed color values $\left(X - \bar{X}\right)\,V$ are further divided by the square root of the diagonal matrix S, the resulting distribution has then a standard deviation of 1 along each dimension (thus the covariance matrix of the transformed colors is the identity matrix). Let $\tilde{X} = X - \bar{X}$ be the color variation and assume that S

has full rank:

$$cov\,(X) \;=\; \tilde{X}^t\,\tilde{X} \;=\; U\,S\,V^t \quad (\Rightarrow U = V)$$

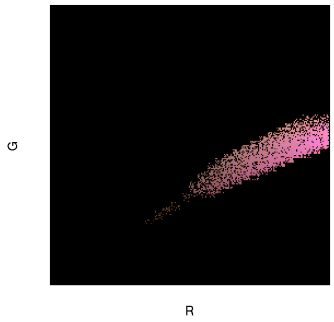
$$\begin{aligned} & U\,S\,V^t &=\;& \tilde{X}^t\,\tilde{X} \\ \Leftrightarrow\quad & S &=\;& U^t\tilde{X}^t\,\tilde{X}V \\ \Leftrightarrow\quad & S^{\frac{1}{2}}\,S^{\frac{1}{2}} &=\;& V^t\tilde{X}^t\,\tilde{X}V \\ \Leftrightarrow\quad & I &=\;& S^{-\frac{1}{2}}V^t\tilde{X}^t\,\tilde{X}VS^{-\frac{1}{2}} \\ \Leftrightarrow\quad & I &=\;& cov\left(\tilde{X}VS^{-\frac{1}{2}}\right) \qquad\qquad \square \end{aligned} \tag{3.2}$$

Two steps in the proof need a further comment. First, since the covariance matrix is symmetric and S is diagonal, then unitary matrices U and V must be equal. Second, if the S (and therefore $cov\,(X)$) has no full rank, then the above would not hold. Instead the first d diagonal elements of the resulting covariance matrix would be one, and the rest zero.

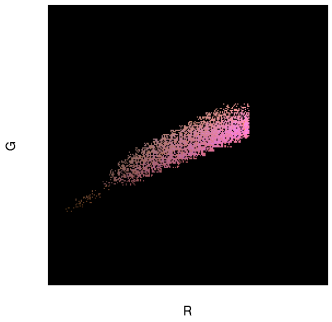
The transformation from the original RGB space to the eigen space is shown in figure 2. Subtracting the mean shifts the distribution to the center. The unitary matrix V rotates the cluster such that the main axes are aligned orthogonal. In the final step the distribution is scaled by the diagonal matrix $S^{-\frac{1}{2}}$ resulting in a distribution that is almost homogenous along all three dimensions.

In the transformed space, it is now possible to compute the probability of each color value. In the eigen spaces the color clusters are roughly normally distributed. There may be slight derivations from this assumption since only 9 of the possible 12 linear transformations were used (shear was not included). Although, when we manually inspected the color distributions we usually found that the algorithm aligns the clusters orthogonal along all three dimensions.

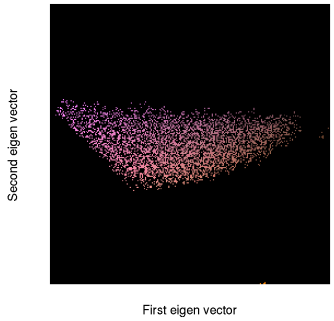
The color distributions are assumed to be normally distributed. After trans-



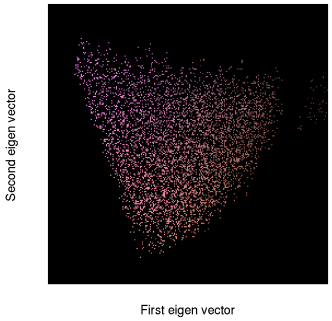
(a) Distribution of pixels manually classified as magenta in the red-green plane of the RGB color space.



(b) Distribution of the magenta pixels if normalized with respect to their mean.



(c) Distribution of the magenta pixels if rotated by V. The axes are the first and the second eigen vector. The main axis of variance is now aligned horizontally.



(d) Distribution of the magenta pixels if scaled according to $S^{\frac{1}{2}}$. The pixels are now homogenously distributed along the color space axes. The plot shows the range from $z = -3$ to 3.

Figure 2: Illustration of the effect of eigen space transformation using single value decomposition (SVD). See text for explanation.

formation into the eigen spaces, the mean and the standard deviation along each dimension are then 0 and 1 respectively. The likelihood of each color value can now be easily computed by multiplying the probabilities along each eigen vector. The basic assumptions is hereby that in the transformed space the distribution is independent along the coordinate axes e_i in eigen space.

$$x = (r, g, b) \tag{3.3}$$

$$z_i = \langle x, e_i \rangle \tag{3.4}$$

$$P(x) = \prod_{i=1}^{3} N(z_i,\, 0,\, 1) \tag{3.5}$$

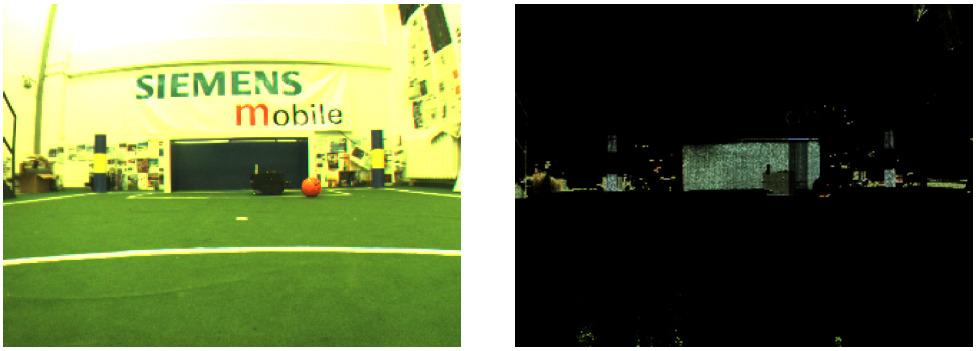
Thus, the likelihood of an RGB value can be calculated by subtracting the mean and applying the rotation and scaling matrices obtained by the SVD. In the actual implementation the linear transformations is summarized in a single affine Lapack operation ($X \leftarrow BX + A,\ B = V\, S^{-\frac{1}{2}}, A = -\mu$).

$$P(x) = P\left((x - \mu)\, V\, S^{-\frac{1}{2}}\right) \tag{3.6}$$

The above formula is the centerpiece of the image comparison module of the rather simple algorithm. Based on an a-priori color classification, that is currently done manually, it uses the color space transformation described to decompose the color values into three independent dimensions. From these (independent) channels is then easy to compute the likelihood of the original color value (Eq. 3.3).

Another way of thinking of the transformation in equation (3.6) is the often used Mahalanobis metric. The square of the Mahalanobis distance r is defined as a bilinear function of the normalized $\tilde{X}$ and the inverse of the covariance matrix.

$$r^2 = \tilde{x}\, C^{-1}\, \tilde{x}^t = \left(x - \bar{X}\right) C^{-1}\ \left(x - \bar{X}\right)^t$$



(a) Original image (b) Likelihood chart

Figure 3: Liklihood plot of the image for the blue color class. The hue and the saturation of each point was taken from the original image. The luminance of the pixels corresponds to the likelihood of the pixel belonging the the blue class.

The equivalence of the Mahalanobis distance and the and the SVD-based transformation can be easily shown:

$$
\begin{aligned}
r^2 &\stackrel{!}{=} ||\tilde{x}\, V\, S^{-\frac{1}{2}}\,|| \\
&= \left(\tilde{x}\, V\, S^{-\frac{1}{2}}\right)\, \left(\tilde{x}\, V\, S^{-\frac{1}{2}}\right)^t \\
&= \left(\tilde{x}\, V\, S^{-\frac{1}{2}}\right)\, \left(S^{-\frac{1}{2}}\, V^t\, \tilde{x}^t\right) \\
&= \tilde{x}\, V\, S^{-\frac{1}{2}}\, S^{-\frac{1}{2}}\, V^t\, \tilde{x}^t \\
&= \tilde{x}\, V\, S^{-1} V^t\, \tilde{x}^t = \tilde{x}\, \left(V\, S\, V^t\right)^{-1}\, \tilde{x}^t \\
&= \tilde{x}\, C^{-1}\, \tilde{x}^t \quad \square
\end{aligned}
$$

Equipped with the normalizing transformation $\tilde{X}\, V\, S^{-\frac{1}{2}}$ (or the Mahalanobis metric), it is therefore possible to state, for each color class and every possible color value, the likelihood that it belongs to the class. Figure 3(b) illustrates this. It shows for each pixel the likelihood that it belongs to the blue goal/blue corner flag post class. The more likely this is, the lighter is the luminance in the chart.

The likelihood transformation can be computed for each class and the color class parameter can be included into the formula using conditional probabilities. The Bayes theorem gives then the likelihood of the color classes.

$$P\left(class \,|\, rgb\right) = \frac{P\left(rgb \,|\, class\right) \cdot P\left(class\right)}{P\left(rgb\right)} \tag{3.7}$$

$$P\left(rgb \,|\, class\right) = P\left(\left(rgb - \mu_{class}\right) V_{class}\, S_{class}^{-\frac{1}{2}}\right) \tag{3.8}$$

3.2 Maximum Likelihood Classification

Before we go on with the description of the second part of the algorithm in the next chapter, we will present a useful application of the Bayes model called maximum likelihood classification. It uses the Bayes equation (3.7) to choose for every pixel the class it most likely belongs to. Although classification is here only used to detect occlusions by opponents or the referee, the maximum likelihood classification may be useful in other modules of the RoboCup software too. In addition it is helpful to examine how well the new color model segmentation compares to the manually crafted lookup tables currently used.

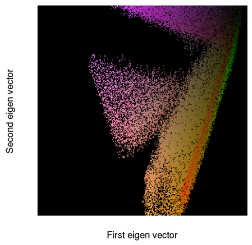
The idea of the Bayes classification, or maximum likelihood estimation in general, is to choose from a parameter space that parameter that maximizes the probability that the observed value originated from that parameter configuration. In the context of this RoboCup color model the state space is discreet and very simple: green, red, yellow, blue, magenta, turquoise, and black. With each class, a set of transformation matrices is associated that transforms the RGB color space into probability spaces.

Figure 4 illustrates the classification problem for the magenta cluster. The first chart 4(a) shows some clusters that are located in the neighborhood of the magenta colors: the dark pink part of the blue cluster and the yellowish colors. The large clusters (lawn, white background, ball) are not shown because they fill a large part of the color space and overlap to a large extend (especially in two dimensions) with the other clusters. In figure 4(b) the classification for the same section of the color space is shown. The main size and form (circular in its eigen space) of the distribution is maintained in the classification (the light pink blob). Nearby colors, that are more reddish are classified as ball (darker red) and colors in the

left of the images belong to the large, green lawn cluster. Although the models for the classes are regular 3-dimensional ellipsoids, a 2-dimensional cut through the classification space can be quite complex, as in figure 4(b). This is the result of the tight packing of the color clusters in the RGB space.

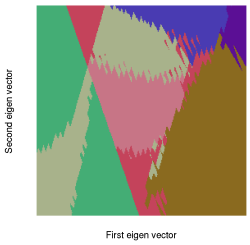
Figure 4(c) shows the classification done with the original lookup table. As far as one can tell it from the little charts, the major organization of the classification could be reproduced by the 9-dimensional model. Albeit the Bayes model was obtained from the same lookup table that is used in figure 4(c), the resulting classification differ in some regions of the color space. The Bayes model assigns more of the violet pixel to the ball class, and the green cluster is larger in figure 4(b).

Looking into the color space is a rather weak method of evaluating the correctness of the model and the classification results for the actual images are of greater interest. Yet, if the color model does not seem to be correctly working then it might



(a) Some of the color classes nearby magenta. The heterogenous classes (lawn, white matter, ball) are *not* shown.

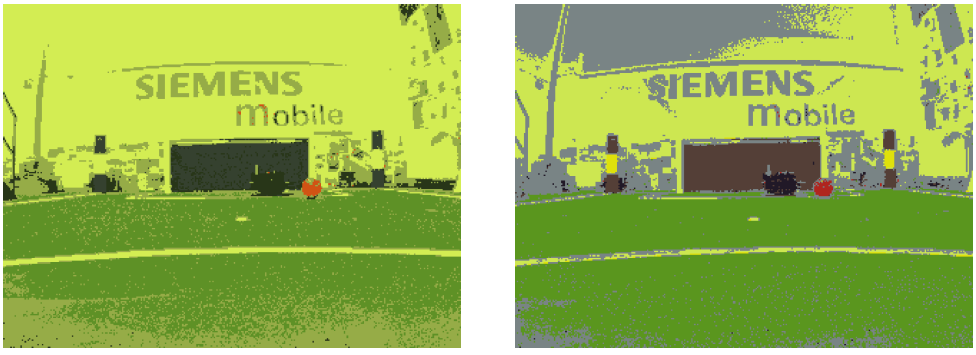
(b) Classification based on the maximum likelihood decision using the Bayes classes.



(c) Classification based on the original lookup table.

Figure 4: Illustration of the maximum likelihood classification on a color plane through the magenta color cluster (z values are in the range of $-5 \ldots 5$ of the magenta eigen space). The jags in the classification charts are due to the conversion from the 16bit to the 24bit color space. The Bayes model itself is smooth and in the algorithm no (16bit) lookup table is used.

be often helpful to generate some classification charts. A sample image that was classified with either the Bayes model (5(a)) or the original lookup table (5(b)) in shown below. The Bayes model usually produces a much smoother classification.



(a) Classified with Bayes classificator.

(b) Classified with manually obtained lookup table.

Figure 5: Classified images.

4 Voxel-based Image Model

In the previous chapter we described how the colors and their distributions are transformed from the original RGB space to the corresponding probability spaces. In the next chapter this probability metric will be used to compare the actual images with the model's prediction. In this chapter we will specify first the world model, providing the missing link from the robot's position and orientation to the camera images.

The world model constist of two parts. The first part is the model of the external world containing the information about the structure of the environment: the form of the objects, their size and location, and their colors. It will be described in detail below. The second component of the model handles the transformation from the external world coordinates to the pixel coordinates of the camera. It converts the three-dimensional global coordinates to a robot-centric system, and then maps these to CCD plane of the camera. The transformation does not only take the position and rotation of the camera into account, but models the distortion of the cameras too. The necessary camera parameters for the linear, geometric projection (f, position, rotation) and the distortion (κ) are estimated once for each robot in a manual calibration procedure. The transformation function is part of the AGILO RoboCup software.

To model the external world, an voxel-based visualization algorithm is used. Voxel-based algorithms are very simple techniques that are popular in games with a limited degree of freedom, and they are employed for simple rendering algorithms too. The basic elements of the algorithm are small cubes used to build up the environment. These cubes are called voxels, in analogy to the two-dimensional pixels. The implementations of voxel techniques can be very simple

if they have limited degrees of freedom, and if the voxels are allowed to be stacked only atop of each other. These constrains make the processing and the voxel space itself very easy. If the degrees of freedom are limited e.g. to translations and one rotational degree of freedom orthogonal to the ground plane, then the scanpaths through the voxel space will be very simple too. If furthermore, the voxels are restricted to lie upon each other, then the three-dimensional voxel configuration can be fully described by a two-dimensional height map.

Fortunately, RoboCup is perfectly suited for these simplifications. The robot platform has only 3 degrees of freedom: translation in x and y, and the orienation ϕ of the vehicle. In addition, the field consists, with one exception, only of solid and simple objects having the same shape at every height. The exception is the crossbeam of the goal. As an approximation the crossbeam is modeled as bars on top of the three goal walls. This is sufficient since the bar would not visible if the robot is close to the goal, and for the larger distances the error is negligible.

4.1 Voxel Map

To generate the height map the algorithm needs the parameters of the surrounding. The AGILO RoboCup software keeps a single file that specifies the parameters of the current field: its length and width, the length and widths of the lines, the position and size of the goals, the corner posts, and many more. This file can be easily edited to adjust the software to variations of the field. Based on these parameters it is then straightforward to generate to appropriate height map.

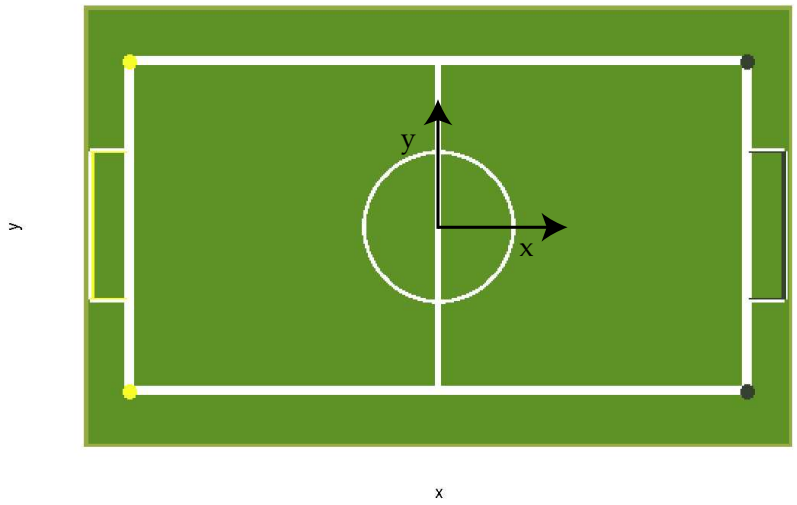


Figure 6: Field model generated by the algorithm showing the topmost color class for each voxel column.

Figure 6 shows such a map, that was generated from the parameter file for the field in Munich. The model includes the four corner posts, the outer lines, the mid-line and the inner circle. The goals are modeled by a double wall: the inner walls have the colors of the own and opponent goal respectively, the outer walls

and the goal posts (not observable in this image) are colored white.

The background is modeled in the map too. The whole field is encircled by a small "wall" that represents the (unknown) color distribution of the off-field environment. The reason for this "wall" is that the algorithm only compares the regions of the image where it expects objects. If there would be other items beyond the expected field (e.g. the goal) they would be simply ignored. For that reason, a pose where the robot is looking towards either side of the field would almost always match any image. The position can be chosen such that the green matches perfectly, and a corner posts or a goal in the image would just be outside of the model, and would be ignored. The background class does not contain an assumption about the actual color of the surrounding (in the figures, the background objects are drawn only for illustration purposes in their the average color tone). Instead, it is solely assumed that the background is neither green, blue, nor yellow. Thus in Bayes terms, object off the field that are colored green, blue or yellow speak against the hypothesis that the supposed pose is correct.

In addition to the "background walls", background objects were inserted on top of the goals. This allows to model a mismatch in the height of the goal and increases the accuracy along the x-axis (distance towards the goal). The sample height map (figure 6) is visualized in figure 7. The background objects around the field and at the goals are clearly visible.

The field map must encode for each voxel whether it is transparent or solid, and in the later case which color it has. Normally both kinds of information are stored in the height map. But unlike the usual mountain environments used in voxel game engines, the RoboCup environment contains perpendicular objects (corner posts, goal walls) with different textures at different heights. To include this information columns are introduced. A column consists of (currently) up to

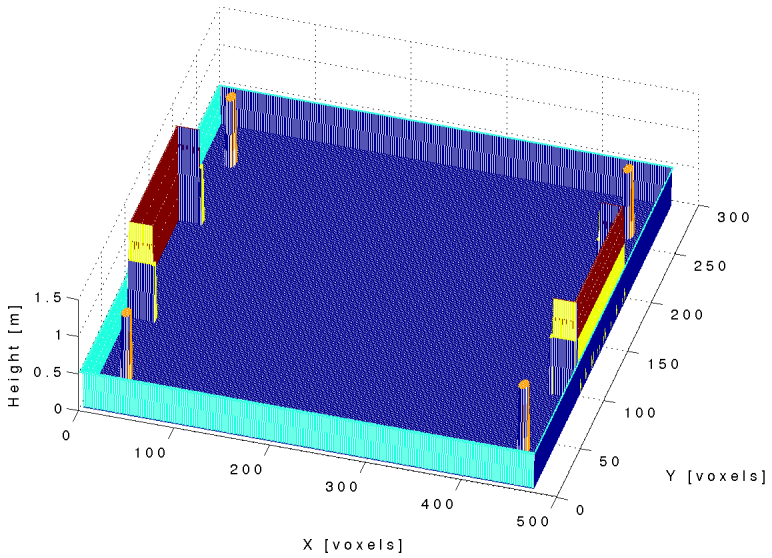


Figure 7: Field model generated by the algorithm showing the height of each voxel column.

three slices that are stacked upon each other. Every slice has a color and stores its height. The field map itself contains (8-bit) references to the columns. The classical height and class maps can be generated by taking for each position the height or color value of the top slice. Figure 6 and 7 show this information.

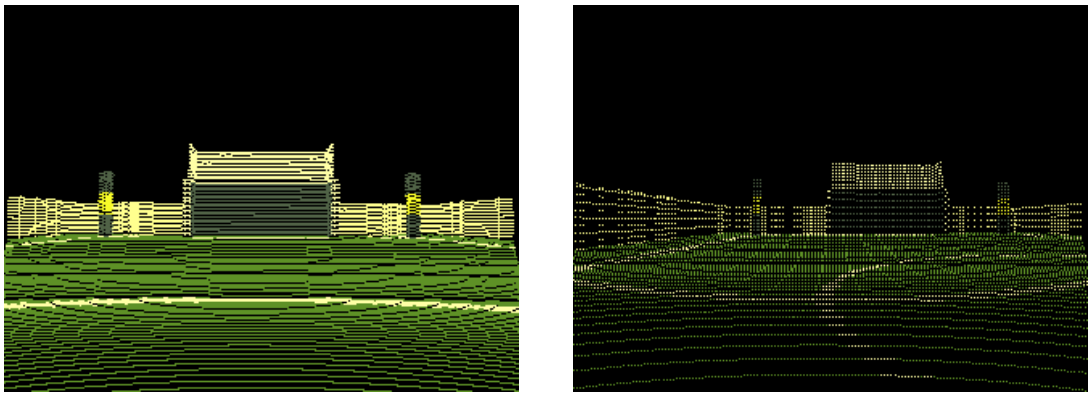
4.2 Scan Path and Ray Casting

One method to create the image from the height map is ray casting. Starting at the position of the robot, it computes the paths of light rays in one direction. Whenever the light ray hits a new voxel, its three-dimensional position is projected onto the image plane of the camera. If camera distortion is ignored, all voxels on a ray will lie on the a straight line in the resulting image. The position on the line is a function of the height and distance of each voxel. Voxels of greater heights will appear above lower voxels, and the vertical position steadily increases with distances. One can therefore think of this rendering as "surfing" along the height profile of the ray, and voxel ray casting is for this reason often refered to as wave surfing. With wave surfing computing occluded voxels is very simple. The latest maximum vertical pixel position is stored, and a new pixel is only drawn if its position is higher than the previous maximum.

In normal voxel algorithms the points obtained by ray casting are then connected by polygons, and usually multiple filters with different spatial scales are used to remove the jitter in the positions of polygons between consecutive frames. In the context of the pixel-based Bayes metric, jitter filtering is not necessary. In fact, additional spatial noise among the pixels used to compare the images is very desirable as it removes possible correlation between the color value of proximate pixels. The interconnection of the voxel positions is not required either, since the Bayes metric assumes uncorrelated color variations and works best with independent pixels.

Figure 8(a) shows a sample image computed from the voxel model. Starting from the origin of the map $(0, 0)$ for each voxel the corresponding pixel was plotted into the image. This results in a very dense pixel distribution that is fur-

thermore condensed in the upper (farther) region of the image. In the context of the Bayes image comparison metric it would be computational too expensive to use all these pixels, and, as already mentioned, using adjacent pixels increases the correlation between them which would invalidate the independence assumption of the Bayes metric. Therefore, the standard voxel technique must be further modified in order to achieve a less dense and more homogenous distribution.



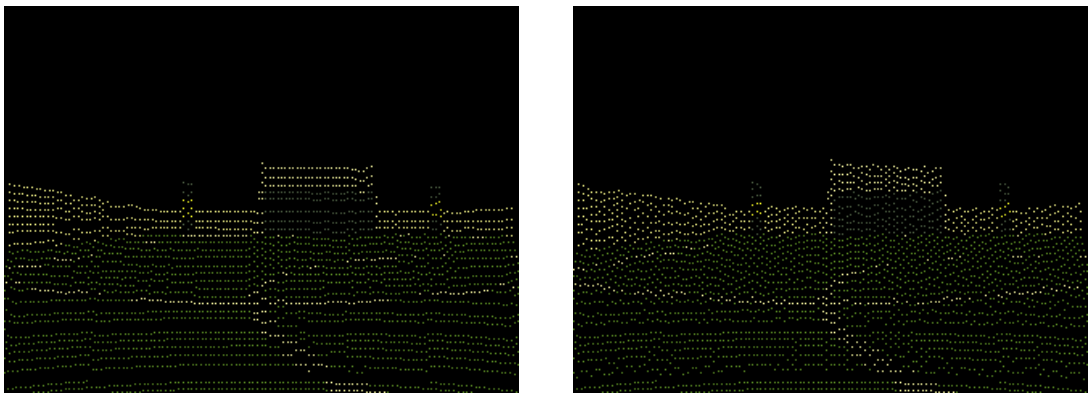
(a) Full raycasting.

(b) Raycasting with larger step size.

Figure 8: Field model at a central position.

A first step would be to use a larger step size by taking only every nth pixel in each ray and including only every nth rays in the analysis (see figure 8(b)). This results in a sparser image but does not correct for the inhomogeneity between the nearer and the distant parts of the model. On the other hand, skipping voxels creates another problem: objects having a very small footprint may be completely omitted. This is a serious problem for objects that are elevated with respect to their surrounding. For instance, it may happen that the corner posts are missed out in some rays. As a consequence, the lawn and background objects that are normally occluded would then be visible. To prevent such errors we adopted an adaptive

strategy. In the beginning, each voxel of a ray is processed, but only when its position on the screen is at least d pixels apart from the previous plotted pixel it will occure in the image. Further on, the step size is increased by 1. This results in a step size that increases adaptively with distance. To prevent the omission of small, elevated objects, for each skipped voxel in the height map it is checked whether it has a different color that the previous voxel in the ray path. If this is the case the step size is reset. This results in an image with a homogenous pixel distribution as shown in figure 9(a). With this algorithm small objects that are at the same level with their surrounding, such as the thin lines in the rear of the field, are still very likely to be skipped. But occulusion errors can not occur.



(a) Ray casting with a minimal pixel distance of 4.

(b) Randomized raycasting with a minimal pixel distance of 4.

Figure 9: Field model at a central position.

In the voxel algorithm the pixel density can be scaled by the minimal distance d between two adjacent pixels within a ray. The angular step size between two neighboring rays is adjusted with respect to d. We found a factor of $d \cdot \pi/1000$ to produces a quite even pixel density. Thus, for the (high) density of $d = 4$ that is used in the illustrations the angular step size is $\pi/250$. Yet, the resulting pixel

distribution if not homogenous in the small scale. Pixel positions of adjacent rays are still correlated since the deterministic step size increment is likely to be the same between the rays going through similar parts of the map. Increasing the step size randomly prevents this correlation. We chose a probability of 0.5 that the step size increases by 1. The results of the randomized version of the algorithm is shown in figure 9(b) where the constitutive rays are scarcely distinguishable.

As already mentioned, the voxel positions are corrected for the individual camera distortions using the model of AGILO RoboCup software. In most instances, this correction is only of minor interests, but it is essential for small distances toward objects. A sample transformation is shown in figure 10. The distortions are clearly visible. But as one can see from figure 10 clipping is still computed with sufficient precision. The errors that result from the violation of the ray casting "straight line" assumption, are minimal.

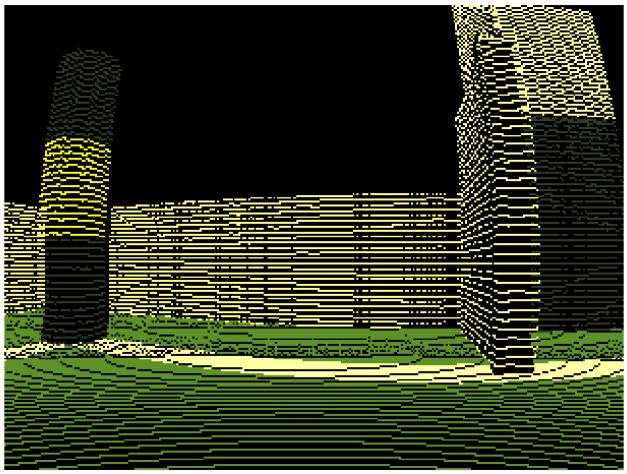


Figure 10: Field model near the line.

4.3 Image Likelihood

To estimate its positions, the robot can take a picture with the frontal camera and in addition it has access to the odometry information about the recent movements. With the voxel-based world model described in the previous chapter, a precise sketch of the expected image can be derived for each imaginable pose, and additionally, the Bayes model captures the color variation for each surface class. In this section these two information sources will be combined into a probabilistic model. The basic idea of the metric is to superimpose the current camera image and the voxel-based sketch, and to compute their disagreement with the Bayes color likelihood model.

Mathematically, one can think of the sketch as a set of 3-dimensional tuples that map the nine color classes to pixel positions. Such a labeled pixel set is a strong simplification of the original images. It discards most of the spatial information (e.g. correlation of color values among neighboring pixels, information about homogenous surfaces, etc.) that is usually used in computer vision algorithms (particularly by morphological operators). The reason behind this simplification is the necessity to assume independence of the color *variations* in order to easily compute the compound probabilities from the single pixel probabilities.

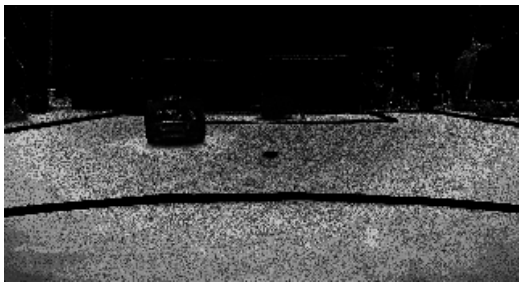
$$image = \{(rgb_{1,1}, rgb_{1,2}, \ldots, rgb_{1,h}, rgb_{2,1} \ldots rgb_{w,h})\} \tag{4.1}$$

$$pixels = \{(x, y, rgb_{x,y})\} \tag{4.2}$$

$$sketch = \{(x, y, class)\} \tag{4.3}$$

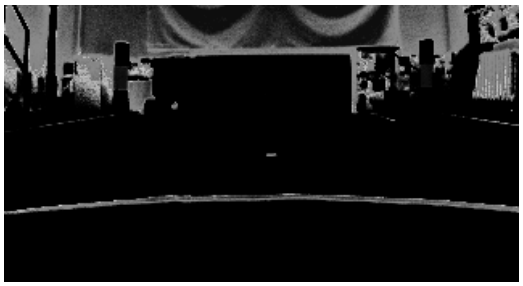
But before we continue to derive the model for the sketch likelihood, it is worth to review the Bayes color model that was presented in chapter 3. There it was mainly discussed in color space, and less at the level of the actual images. Applied

(a) Original image



(b) Green likelihood

(c) Blue likelihood



(d) White likelihood

(e) Magenta likelihood

(f) Black likelihood

Figure 11: Logarithmic likelihood values for different color classes for the image on the top left

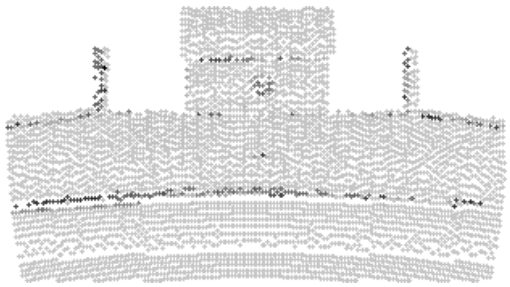


Figure 12: Logarithmic likelihood estimates for the image with the two robots. White circles have a high likelihood in the expected color class, gray and black indicate mismatches. See figure 11 for likelihood images of the single color classes.

to an image, the problem of probabilistically comparing the camera images with a generated sketch can be visualized by substituting the pixels of the captured color image with the corresponding likelihoods. Figure 11 shows such a substitution for the likelihood of the lawn, blue goal, lines, teammate and for the robots color class. For better visualization, the probabilities are plotted in a logarithmic scale.

The image sketch, on the other hand, can be viewed as an operator that selects the probability values from the likelihood charts. If the pixel corresponds with the expected color class, then the likelihood value will be high (white in the chart). Otherwise, when the pixel has an unexpected color, the likelihood would be low (dark in the chart). The result of this operation (again in logarithmic units) is shown in figure 12. Pixels that agree with the expected colors are white, mismatches have a low likelihood and are shown in dark gray. In addition to the operation shown in figure 12, an obstacle and enemy detection will be performed to exclude pixels from the computation that likely belong to other robots, the referee or other dark obstacles. A RGB value is classified as an obstacle if the likelihood of one of the obstacle classes exceeds the likelihood any the other non-obstacle class (maximum likelihood classification; figure 11(e) and 11(f)).

The computation of the Bayes color model was derived in chapter 3. The like-

lihood of the RGB color values is assumed to be a 3-dimensional normal distributions and the parameters of the ellipsoids (mean, size and orientation) are determined for the color lookup table. To generate the lookup table, images from different positions at the field are labeled with an interactive program prior to a game. Single value decomposition (SVD) is used to compute the optimal parameters of the color covariance matrices, and the likelihood of a RGB value can then be computed by Eigen transformation.

$$P\left(rgb \,|\, class\right) = P\left(\left(rgb - \mu_{class}\right) V \, S^{-\frac{1}{2}}\right)$$

The probability of a RGB value is then the sum of the likelihoods in the single color classes (4.4). Since in praxis, the color distributions often do not significantly overlap, the sum could be substituted with the maximum. Since the algorithm uses lookup tables for all probability distributions, the likelihoods have to be computed only once and therefore in the current implementation the mathematically correct summation is used.

$$P\left(rgb\right) = \sum_{class} P\left(rgb \,|\, class\right) \cdot P\left(class\right) \tag{4.4}$$

$$\approx \max_{class} P\left(rgb \,|\, class\right) \cdot P\left(class\right) \tag{4.5}$$

Based on the RGB likelihoods, the likelihood of an image can then be derived. Assuming independence of the color *variations*, the probability of a subset of an image's pixels can be stated in terms of the pixels' color probabilities. The likelihood of a pixel set to have certain RGB values is thus the product of the probabil-

ities of the color values.

$$P\left(image \,|\, sketch\right) = \prod_{(x,y,class)\in sketch} P\left(rgb_{x,y} \,|\, class\right), \quad rgb_{x,y} \in image \tag{4.6}$$

For any given image it is hence possible to compute the probability that it corresponds to the sketch. Using the Bayes theorem, it is easy to compute the likelihood that the sketch corresponds to a given image too. The required image probability can be computed with the color model. The assumption is here the same as for the sketch: that the color variability of the pixels are independent. Since it is sufficient to consider only the subset of the pixels that are used to compare it to sketch, the independence assumption will be met.

$$P\left(sketch \,|\, image\right) = \frac{P\left(image \,|\, sketch\right)}{P\left(image\right)} P\left(sketch\right) \tag{4.7}$$

$$P\left(sketch \,|\, image\right) = \frac{\prod_{sketch} P\left(rgb_{x,y} \,|\, class_{x,y}\right)}{\prod_{sketch} P\left(rgb_{x,y}\right)} P\left(sketch\right) \tag{4.8}$$

$$P\left(sketch \,|\, image\right) = \prod_{sketch} \frac{P\left(rgb_{x,y} \,|\, class_{x,y}\right)}{P\left(rgb_{x,y}\right)} P\left(sketch\right) \tag{4.9}$$

To localize the robot, it is not sufficient to calculate the sketch likelihood; one wants to know the likelihood that a given pose corresponds to the image. To compute this via a Bayes approach it would be necessary to model the sketch generation process too. But with the independence assumption it is simpler. A sketch can be seen as repetitions of the same probability experiment: obtaining a random color variation for a given object. Thus, the likelihood of the sketch is the likelihood of the pose likelihood to the power of number of pixels in the sketch, and the pose likelihood is then the nth root of the probability product.

$$P\left(pose \,|\, image\right) = \sqrt[n]{\prod_{sketch} \frac{P\left(rgb_{x,y} \,|\, class_{x,y}\right)}{P\left(rgb_{x,y}\right)}} P\left(pose\right) \tag{4.10}$$

Optimization Estimating the likelihood of an image – as described so far – will be very expensive in terms of computational costs. Furthermore, the above formula can be, due to the small probability values involved, numerical instable. The numerical stability, as well as the computation time, can be improved by computing the probabilities in a logarithmic scale.

$$\begin{aligned} \log P\left(pose \,|\, image\right) = \frac{1}{n} \sum_{sketch} \Big[\log P\left(rgb \,|\, class\right) \; - \log P\left(rgb\right) \Big] \\ + \; \log P\left(pose\right) \end{aligned} \tag{4.11}$$

These computations are done with floating point precision. To further speed up the processing, the logarithmic rgb likelihoods $P(rgb|class)$ and probabilities $P(rgb)$ are computed for the complete $RGB16$ color space (G: 6 bits, R, B: 5 bits) once during the program startup, and are then accessed via 256 kBytes lookup tables.

5 Probability and Monte Carlo Methods

To find out its position, just by looking at the camera images, the robot must possess a profound model of its environment. Such a model was suggested in the previous chapter, predicting either a quick, rough sketch or, if one wishes, can even generate a fine-grain pointillistic image of the expected surroundings. The voxel-based rendering technique, though a simplistic version, is yet very complex when it is used inversely, as a probability metric. For most localization problems, even for the simplest environments, one could not expect to find closed form solutions solving them.

In addition, modeling the progress of image genesis will get even more complex if the robot starts moving or if environment properties change. Currently, the RoboCup regulations still define the illumination as fixed, but it is planned to release these restrictions in the next years. Furthermore, changes in the robot's position imply changes in the image, and in addition robots and static objects may occlude each other. At first glance, these problems seems to be easily solvable. Occlusion can be easily modeled and the pose of the robot is a simple 3-dimensional variable. But in real-world situation it is sometimes, and in RoboCup quite often, hard to tell where the robot is. The odometry information is imprecise due to factors such as unequal wheel sizes, slipping or collisions and in a short time scale such errors makes purely cumulative odometry useless, and the likelihood metric is neither smooth nor unambiguous.

Therefore, methods that employ only a single estimate can not be robust. If there is no linear relationship between the error of the estimation and the observed measurements (e.g. for the robot's orientation), then iterative, local search algorithms will be of limited use too. In such cases one solution would be to use

multiple estimates, to update them separately, and to choose the best fitting or the average estimate as the result. Exactly this is the basic idea of particle filters. Mathematically, choosing multiple estimates is equivalent to drawing samples from the theoretic distribution of the estimation error. The theoretic framework for the sampling technique, as well as the underlying probability and Markov chain theory, will be discussed in the following sections.

Trivially, to draw from a distribution, it must first be known. As already mentioned, for complex random sequences (such as images), it is impossible to know it a-priori. However, it is almost always possible to give a recursive solution for the problem, where the distribution is a (probabilistic) function of the distribution at the previous time step. Markov chains are a very powerful class of iterative probabilistic models that can be applied to finite as well as to countable sets.

Drawing from a Markov chain, thus combining the sampling and recursive methods, is called a Monte Carlo Markov Chain (MCMC) approach. It will be used to evaluate the images using a likelihood estimate of the previous chapter and to combine the resulting pose distribution with prior information together with the odometry measurements within the same probabilistic model.

5.1 Probability Space

To solve the estimation problem it is necessary to have a probability theory that is capable to handle infinite sets, since the space of possible poses is infinite $X \in \mathbb{R}^3$.

Let Ω be a space or set of *sample points* ω. Ω consists of all possible outcomes of an experiment. Thus, for each possible result their exists an $\omega \in \Omega$. A probability theory must assign probability values to the possible outcomes Ω. But rather than to assign the values to each single ω it is more useful to assign them to subsets

of Ω. Firstly, assigning a probability value to each single sample point ω is not always possible for infinite sets. Secondly, if we have theories about measurement variables we can work with them (and the partition they create) and need not to care about the underlying set Ω.

Let F be a class of subsets of Ω. F is called a *field* or algebra if Ω is nonempty, Ω is contained F and if F is closed for formation of complements and finite unions:

$$\Omega \in F \tag{5.1a}$$

$$A \in F \Rightarrow A^c \in F \tag{5.1b}$$

$$A, B \in F \Rightarrow A \cup B \in F \tag{5.1c}$$

A field is called a *σ-field* if it is closed under the formation of countable unions:

$$A_1, A_2, \ldots \in F \Rightarrow \bigcup_i A_i \in F \tag{5.2}$$

Now it is possible to assign probability values to the elements of F. The set function assigns to each event $A \in F$ a probability value. P is called a *probability measure* if:

$$0 \leq P(A) \leq 1 \tag{5.3a}$$

$$P(\varnothing) = 0,\ P(\Omega) = 1 \tag{5.3b}$$

$$A_1, A_2, \ldots \in F \ \wedge \bigcap_i A_i = \varnothing \ \Rightarrow \ P\left(\bigcup_i A_i\right) = \sum_i P(A_i) \tag{5.3c}$$

(Ω, F, P) is called a *probability space*. For the application to Markov chains it will be handy to define the sample points as countable or finite sequences in a

multidimensional space:

$$\omega = (\omega_1, \omega_2, \ldots) \quad \omega_i \in S,\, \omega \in S^\infty \tag{5.4a}$$

$$\omega_{1..t} = (\omega_1, \omega_2, \ldots, \omega_t) \quad \omega_i \in S,\, \omega_{1..t} \in S^t \tag{5.4b}$$

Up to now, we have only arbitrary events $A \in F$ and a measurement function P assigning probabilities to them. To observe or measure something in the probability space it is necessary to define (random) variables upon Ω. X is called a *random variable* if X is a real-valued function on Ω and X is measurable F. It assigns to each possible outcome ω a value in the measurement space.

$$[\, \omega \colon X(\omega) = x \,] \in F \tag{5.5}$$

For multivariate statistics a n-dimensional random variable or a *random vector* can be defined as a mapping from Ω to $\mathbb{R}^n$:

$$X \colon \omega \to X(\omega) = (X_1(\omega), \ldots, X_n(\omega)) \tag{5.6}$$

A random vector X is equivalent to a n-tuple $X = (X_1, \ldots, X_n)$ of random variables, since X is measurable if and only if all X_i are measurable. [2]

Now it is possible to formulate the process of measurement in probability theory. Odometry measurements are random vectors that assign to each realisation of an experiment and for each time step the distance currently traveled and the change in heading.

$$Y_t \colon \omega \to (d_t, \Delta\phi_t) \tag{5.7a}$$

$$Y \colon \omega \to (Y_1(\omega), Y_2(\omega), \ldots) \tag{5.7b}$$

To actually control a robot with a probability theory, two more concepts are necessary. First, the *distribution function* F of a random variable gives the probability of the internal $(-\infty, x]$:

$$F(x) = P[X \leq x] = P[X(\omega) \in (-\infty, x]\ ] \tag{5.8}$$

F is often called the cumulative density function (CDF). The probability density function (PDF) is defined as a limit in x. Although the probability measure P is well designed on F, the PDF may not be defined for each value x.

$$P(x) = F(x) - P[X \in (-\infty, x)] \tag{5.9}$$

The *expectation value* of a random variable is defined as the convolution:

$$E[X] = \int XP = \int_{\Omega} X(\omega) P(\omega)\ \mathrm{d}\omega \tag{5.10}$$

It is the goal of many statistics and thus robotics approaches to estimate the expectation value $E[X]$ given a single realization of an experiment ω. The Kalman filter, for example, is the optimal solution if X is a vector of n uncorrelated, multidimensional measurements X_i and if all X_i are Gaussian distributions with the same mean and covariance.

5.2 Markov Chains

Markov chains are a very powerful methods to model probabilistic processes. They can be easily applied to a wide variety of problems, and when used in connection with Monte Carlo methods they can be applied to complex models and large state spaces too. In the MCMC localization algorithm, Markov processes are

used to combine the image likelihood distribution with the a-priori probabilities, and to propagate the pose estimates when the robot is moving. In the following, the Markov model will be explained for the odometry propagation.

Let S be a countable or finite set. For the odometry example, S is the set of possible robot poses. We assume, that its initial pose distribution is known. (It may be a normal distribution with a mean at the center of the field and a heading of 0.)

$$\alpha_i = P\left[\, X_0 = i \,\right], \quad i \in S \tag{5.11}$$

The probability of the next state can then be related to the odometry model. If the measurement $(d, \Delta\phi)$ and the constants r_d and $r_{\Delta\phi}$ are known, then the PDF may be calculated.

$$P\left[\, X_1 = j \,|\, X_0 = i \,\right] = f\left(d_1, \Delta\phi_1, j - i\right) = f\left(y_1, j - i\right) \tag{5.12}$$

The transition probabilities for the following state are calculated in the same way. If the transition probability is constant then is it said that the Markov chain has *stationary transition probabilities*. Otherwise, as for the odometry problem, the Markov chain has *dynamic transition probabilities*.

$$P\left[\, X_n = j \,|\, X_{n-1} = i \,\right] = p_{ij} \quad stationary \tag{5.13}$$

$$P\left[\, X_n = j \,|\, X_{n-1} = i \,\right] = p_{ij}^{(n)} \quad dynamic \qquad i, j \in S \tag{5.14}$$

p_{ij} needs to be nonnegative and the probability to reach any state in S must be 1.

$$\sum_{j \in S} p_{ij} = 1, \quad i \in S \tag{5.15}$$

The sequence $X = (X_0, X_1, \ldots)$ is called a *Markov chain* if the X_n depends only on the distribution of the previous state and the transition probability.

$$\begin{aligned} P\,[\,X_n = i_n \,|\, X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-i}\,] \\ = P\,[\,X_n = i_n \,|\, X_{n-1} = i_{n-1}\,] = p_{ij}^{(n)} \end{aligned} \tag{5.16}$$

The probability of single sequences is determined by the transition probabilities and the initial probabilities.

$$\begin{aligned} & P\,[\,X_0 = i_0, X_1 = i_1, \ldots, X_n = i_n\,] \\ & = P\,[\,X_0 = i_0\,]\,P\,[\,X_1 = i_1\,] \cdots P\,[\,X_b = i_n\,] \\ & = \alpha_{i_0}\, p_{i_0 i_1}^{(1)} \cdots p_{i_{n-1} i_n}^{(n)} \end{aligned} \tag{5.17}$$

Higher order transitions and (unconditional) probability of states can be calculated by averaging over all possible sequences that lead from i to j.

$$\hat{p}_{ij} = P\,[\,X_{m+n} = j \,|\, X_m = i\,] \tag{5.18a}$$

$$= \sum_{k_{m+1} \ldots k_{m+n-1}} p_{i k_{m+1}}^{(m+1)}\, p_{k_{m+1} k_{m+2}}^{(m+2)} \cdots p_{k_{m+n-1} j}^{(m+n)} \tag{5.18b}$$

$$P\,[\,X_n = j\,] = \sum_{i_0 \ldots i_{n-1}} \alpha_{i_0}\, p_{i_0 i_1}^{(1)} \ldots p_{i_{n-1} j}^{(n)} \tag{5.19}$$

Theoretically, we can now derive the 'banana shape' of the odometry error distribution if we use $d \propto N(d_t, r_d d_t)$ and $\Delta\phi \propto N(\Delta\phi_t, r_{\Delta\phi}\Delta\phi_t)$ to compute $p_{ij}^{(t)}$ and then compute the sum (5.19) or better (5.18b). Fortunately, this will not be necessary. For the Monte Carlo method used, it will be sufficient if we can draw samples from the distribution $p_{ij}^{(t)}$ to iteratively compute the probability of the current pose distribution $P\,[\,X_t\,]$.

5.3 Monte Carlo Markov Chain Methods (MCMC)

Monte Carlo Methods were first developed to solve pure statistical problems and have been used for centuries. The name 'Monte Carlo' is of newer origin, it was coined by Metropolis during the Manhattan project in World War II. The name refers to the similarity to statistical models of games of chance and the famousness of Monte Carlo for gambling [23].

Suppose, you have a known, non-elemental distribution function $P[X]$ and want to compute the expectation value $E[X]$. But it is not possible to solve the convolution $\int_\Omega XP \,\mathrm{d}\,\omega$. The solution is, since it is possible to evaluate P, to draw some random samples Λ from the distribution such that the likelihood of a drawn sample to have the value x is equivalent to $P[X = x]$. $\frac{1}{n}\sum_{\lambda\in\Lambda} X(a)$ will converge to $E[X]$ for $n = |\Lambda| \to \infty$. It depends on $P[X]$ how many samples are needed. [16] report that about a dozen samples are sufficient in most cases.

Drawing from more complex distributions is not trivial, neither it is computational inexpensive and there exist a great variety of Monte Carlo methods, that are either very good approximations of perfect random samplers or very fast. For the application on Markov chains, it is often better to use a fast, but potentially inaccurate sampler.

The Monte Carlo Markov Chain (MCMC) methods were first developed to simulate molecular motion. In such systems each molecule has an inner state, notably the impulse and the changes of the next state is a stationary probabilistic function of that state. The task is to find an *invariant distribution* π with respect to the transition of the Markov chain.

$$\pi[X' = j] = \sum_{i\in S} P[X' = j \,|\, X = i]\ \pi[X = i], \quad \forall\, j \in S \tag{5.20}$$

For Monte Carlo methods it will be required that the Markov chain will converge to its invariant distribution π with $n \to \infty$ regardless of the initial distribution α. Such Markov chains are call *ergodic*. Per this definition of ergodicity, ergodic chains have exactly one invariant distribution.

$$P\,[\,X_n\,] \to \pi\,[\,X\,], \quad n \to \infty\,, \forall\,\alpha \tag{5.21}$$

Thus, samplers need not to produce exactly random, neither perfectly correct distribution. But they must not impair the ergodic behavior the Markov chains. For criteria for good samplers the reader is refered to an introduction of MacKay [16] and the review of Monte Carlo methods by Radford Neal [18].[2]

5.4 Sampling Methods

As already mentioned, sampling from an arbitrary distribution can be hard. However, for simple distributions there is no need for advanced samplers. For uniform distributions most programming languages have good implementations of pseudo-random number generators and if the inverse of the cumulative density function (ICDF) or a good approximation is known, there is no need for more advanced sampling algorithms. Fortunately, this is the case for the Gaussian distribution. Good approximations can be found in the literature, an overview give [4] and [3].

$$P\,[\,U\,] \;=\; \begin{cases} 1, & 0 \leq U < 1 \\ 0, & otherwise \end{cases} \tag{5.22a}$$

[2] Both papers are available via CiteSeer.

$$P[N] = P[ICDF_N(U)] \tag{5.22b}$$

For multivariate normal distributions or to generate multiple, correlated random variables, first n independent normal derivates are generated with (5.22b). Then, each variable is multiplied with the square root of the eigenvalues (the standard deviation of the distribution along each eigen vector). Finally, these independent variables are rotated such that they are correlated as specified and the mean is then added. The necessary parameters can be calculated with a single value decomposition (SVD) from the covariance matrix.

$$\Sigma = U\,D\,U^t, \quad D \text{ is diagonal, } \Sigma \text{ is symmetric} \tag{5.23}$$

$$N(\mu, \Sigma) \propto \mu + U D^{\frac{1}{2}} (N_1, \ldots, N_m)^t \tag{5.24}$$

Importance Sampling

If it not feasible to find a good and easy to compute approximation for the derivate, then it may be possible to find a fast, bad one. The idea of *importance sampling* is then to introduce a weight to each sample. Let X be the wanted distribution and B the approximating function. Then the weight for a sample $w(\lambda)$ is the proportion:

$$w(\lambda) = \frac{P[X = \lambda]}{B(\lambda)}, \quad \lambda \in S \tag{5.25}$$

Correctly, importance sampling is not a general sampling method. Instead, it can be used to estimate parameters of the distribution or it is used in conjunction with other sampling techniques. To compute for example the expectation value each particle would be multiplied with its weight. If B is a good approximation to $P[X]$, then the weighted sum would be a good estimator for the expectation

value $E[X]$.

$$\frac{1}{\sum w(\lambda)} \sum_{\lambda \in \Lambda} \lambda \cdot w(\lambda) \rightarrow E[X], \quad n = |\Lambda| \tag{5.26}$$

Rejection Sampling

For rejection sampling an envelope B for the distribution X is needed. A sample λ from B will be accepted with the probability $w(\lambda)$, otherwise it will be discarded and a new sample is drawn that is tested again. Thus, the probability that an accepted sample λ has the value x is equivalent to $P[X = x]$ and if drawing from B is random, then the samples Λ will be independent, too.

The accuracy of importance sampling and the efficiency of rejection sampling depend on the closeness of the approximation B. If B is badly chosen then small, but highly probable regions in X may not be considered by importance samplers. Rejection samplers are extremely inefficient if $B(x)$ is often larger than $P[X]$. Both problems get worse in higher dimensional spaces when the tails of distributions are more sparse.

Metropolis Sampler

The Metropolis and the Gibbs sampler are methods that can be applied to any distribution and they do not require approximation functions as the previous methods. Both methods instead use the the previously generated samples to generate new values. Thus, such samplers are often called Monte Carlo Markov Chain methods. In contrast to the importance and rejection methods, the samples are correlated.

In a Metropolis sampler for each particle λ a random value from a noise function N is drawn. The noise function does not need to be related to $P[X]$ and

often a normal distribution is chosen. Let $N(x)$ be a noise function function with the mean x. A new particle λ_{n+1} from $N(\lambda_n)$ will be accepted with the probability

$$p = \min\left(1\,,\ \frac{P\,[\,X=\lambda_{n+1}\,]\ P\,[\,N\,(\lambda_n)=\lambda_{n+1}\,]}{P\,[\,X=\lambda_n\,]\ P\,[\,N\,(\lambda_{n+1})=\lambda_n\,]}\right) \tag{5.27}$$

If the new state λ_{n+1} is rejected then the old state λ_n is taken. For symmetric noise functions equation (5.27) will reduce to the comparison of the two particle probabilities $P\,[\,X=\lambda_{n+1}\,]$ and $P\,[\,X=\lambda_n\,]$.

Gibbs Sampler

The noise function for the Metropolis function, although it is not critical for most applications, must be chosen for each problem $P\,[\,X\,]$ and will have an influence on the convergence rate. The Gibbs sampler treats each dimension of a n-dimensional state $\lambda = (\lambda_1, \ldots, \lambda_n)$ separately and uses $P\,[\,X=(X_1,\ldots,X_n)\,]$ directly as a 'noise function' to generate the new samples Λ'

$$P\,[\,\Lambda'_1=\lambda'_1\,] \;=\; P\,[\,X_1=\lambda'_1\,|\,\lambda_1,\lambda_2,\ldots,\lambda_n\,] \tag{5.28a}$$

$$P\,[\,\Lambda'_2=\lambda'_2\,] \;=\; P\,[\,X_2=\lambda'_2\,|\,\lambda_1,\lambda_3,\ldots,\lambda_n\,] \tag{5.28b}$$

$$P\,[\,\Lambda'_n=\lambda'_n\,] \;=\; P\,[\,X_n=\lambda'_n\,|\,\lambda_1,\lambda_2,\ldots,\lambda_{n-1}\,] \tag{5.28c}$$

If the problem allows a decomposition then the Gibbs sampler (or one of its many variants) is the preferred method.

5.5 Sequential Monte Carlo Methods (SMC)

The purpose of MCMC methods is to find the invariant distribution $\pi\,[\,X\,]$ in the state space of an ergodic Markov chains. Sequential Monte Carlo (SMC) methods mainly use the same sampling techiques as MCMC models, but to solve a completly different problem.

Let S be a state space and $X = (X_1, X_2, \ldots)$ a (dynamic) Markov chain that is characterized by the inital distribution α_i and transition probabilities $p_{ij}^{(n)}$.

$$P\,[\,X_0 = i\,] \;=\; \alpha_i \tag{5.29a}$$

$$P\,[\,X_n = j \,|\, X_{n-1} = i\,] \;=\; p_{ij}^{(n)} \tag{5.29b}$$

At each time step t, the state of the state of the Markov chain can be measured by a random variable Y_t and there exists a probabilistic relationship between the measurement and the state space $P\,[\,X_t \,|\, Y_t\,]$. SMC methods estimate the conditional distribution of the chains $X_{1..t}$, given the observed measurement vector $y_{1..t}$.

$$P\,[\,X_{1..t} \,|\, Y_{1..t} = (y_1, y_2, \ldots, y_t)\,] \tag{5.30}$$

The term hidden Markov model (HMM) is used for discrete state spaces when there is no need for Monte Carlo sampling. SMC methods are often called particle filters, a term that was introduced by Kitagawa [11] ([6]) which referes to the analysis (filtering) of nonstationary time series. Two well known SMC applications are contour tracking algorithm Condensation [10] and the Monte Carlo localization algorithm (MCL) for a mobile museum robot [29].

Monte Carlo Sampling

In the previous section the sampling techniques were introduced as methods that can be used to estimate the expectation value of an intractable distribution $P[\,X\,]$ (equation 5.26, p. 44). To estimate the conditional distribution $P[\,X_{1..n} \,|\, y_{1..n}\,]$ this can now be expanded to mappings f_n from $X_{1..n}$ to $\mathbb{R}$ that are integrable $P[\,X_{1..n} \,|\, y_{1..n}\,]$.

$$f_n \colon S^n \mapsto \mathbb{R} \tag{5.31a}$$

$$I(f_n) = E_{y_{1..n}}(f_n(x_{1..t})) = \int f_n(x_{1..n})\; P[\,x_{1..n} \,|\, y_{1..n}\,]\;\; \mathrm{d}\, x_{1..n} \tag{5.31b}$$

If $I(f_n)$ is approximated by m samples λ_i that are drawn from $P[\,X_{1..t} \,|\, y_{1..t}\,]$ then the estimator $\hat{I}_m(f_n)$ will, almost sure, converge to $I(f_n)$. [5]

$$\widehat{I_m}(f_n) \;=\; \frac{1}{m}\sum_{i=1}^{m} f_n(\lambda_i)\,, \quad \lambda_i \sim P[\,X_{1..t} \,|\, y_{1..t}\,] \tag{5.32a}$$

$$\widehat{I_m}(f_n) \;\overset{a.s.}{\to}\; I(f_n)\,, \quad m \to \infty \tag{5.32b}$$

Since in most cases it is impossible to draw random samples from $P[\,X_{1..t} \,|\, y_{1..t}\,]$ almost all particle filter use an importance function B to draw samples from and weights to correct the sampling error. If B is well chosen (such that $var\;[\,w_i\,]$ is small), the estimators $\widehat{I'_m}(f_n)$ will still converge against the real value $I(f_n)$ when $m \to \infty$. (see again [5])

$$\widehat{I'_m}(f_n) \;=\; \frac{1}{\sum w(\lambda_i)}\sum_{i} w(\lambda_i)\, f_n(\lambda_i)\,, \quad \lambda_i \sim B[\,X_{1..t} \,|\, y_{1..t}\,] \tag{5.33a}$$

$$\widehat{I'_m}(f_n) \;\overset{a.s.}{\to}\; I(f_n)\,, \quad m \to \infty \tag{5.33b}$$

$$w(\lambda_i) \;=\; \frac{P[\,X_{1..t} = \lambda_i \,|\, y_{1..t}\,]}{B[\,X_{1..t} = \lambda_i \,|\, y_{1..t}\,]} \tag{5.33c}$$

Importance sampling can be used to estimate some parameter $I(f_n)$ of the theoretic distribution $P[\, X_{1..n} \mid y_{1..n} \,]$. But how can $P[\, X_{1..n} \mid y_{1..n} \,]$ be computed? Normally, it is not possible to find a formula that calculates the likelihood of any sequence $x_{1..n}$ for a given sequence of measurements $y_{1..n}$.

Bayesian Filtering

With the Bayes theorem and the Markov assumption $P[\, x_{1..n} \mid y_{1..n} \,]$ can easily transformed to the recursive formula

$$P[\, x_{1..n} \mid y_{1..n} \,] = \frac{P[\, y_{1..n} \mid x_{1..n} \,]\; P[\, x_{1..n} \,]}{P[\, y_{1..n} \,]} \tag{5.34a}$$

$$= \frac{P[\, y_n \mid x_n \,]\; P[\, x_n \mid x_{n-1} \,]}{P[\, y_n \mid y_{1..n-1} \,]}\, P[\, x_{1..n-1} \mid y_{1..n-1} \,] \tag{5.34b}$$

$P[\, y_n \mid x_n \,]$ is the sensor model, $P[\, x_n \mid x_{n-1} \,]$ the known transition probability of the hidden Markov chain $p^{(n)}_{x_{n-1}x_n}$. $P[\, x_{1..n-1} \mid y_{1..n-1} \,]$ is recursively defined. Unfortunately, it is almost always impossible to calculate $P[\, y_n \mid y_{1..n-1} \,]$. (Otherwise we would not need a particle filter.)

$P[\, y_n \mid y_{1..n-1} \,]$ does not depend on X. Thus, for a given experiment ω where $Y_{1..n}(\omega) = y_{1..n}$ was observed, $P[\, y_n \mid y_{1..n-1} \,]$ is a constant. Therefore, we can compute the weight $w(\lambda_i)$ (5.33c) up to a constant and since the weights are normalized by $\frac{1}{\sum w_i}$ (5.33a) the convergence behavior of an estimator $\widehat{I'_m}(f_n)$ is not altered.

The distribution of the hidden Markov sequences can therefore be modeled by

$$P[\, x_{1..n} \mid y_{1..n} \,] \;\propto\; P[\, x_1 \mid y_1 \,] \prod_{t=2}^{n} P[\, y_t \mid x_t \,]\, P[\, x_t \mid x_{t-1} \,] \tag{5.35}$$

Degeneracy

If the likelihood $P\,[\,x_{1..n}\,|\,y_{1..n}\,]$ is calculated by (5.35) then it seems to be a good idea to choose a similar recursive formula for the importance function B.

$$B\,[\,x_{1..n}\,|\,y_{1..n}\,] \;=\; B\,[\,x_1\,|\,y_1\,] \prod_{t=2}^{n} B\,[\,x_t\,|\,x_{1..t-1}, y_{1..t}\,] \tag{5.36}$$

It had been proven that for such recursive importance functions the variance of the importance weights will increase stochastically over time ([13]; see [6]).

Sampling/Importance Resampling (SIR)

Ideally, B should be identical to P. Thus, w_i would be 1 and $var\,[\,w_i\,]$ would be zero. The SIR algorithmus suggested by Rubin [24] samples from the transition function $P\,[\,X_t\,|\,X_{t-1}\,]$ and then associates the new samples with the weight $P\,[\,y_t\,|\,x_t\,]$ (see 5.35)

$$\Lambda^{(t)} \;\propto\; P\left[\,X_t\,|\,X_{t-1} = \lambda_i^{(t-1)}\,\right] \tag{5.37a}$$

$$w\left(\lambda_i^{(t)}\right) \;=\; P\left[\,y_t\,|\,X_t = \lambda_i^{(t)}\,\right] \tag{5.37b}$$

As the name SIR suggests, after sampling (5.37a) and importance weighting (5.37b) the samples $\lambda_i^{(t)}$ will be resampled. The probability that a sample $\lambda_i^{(t)}$ is selected is proportional to its weights w_i. A sample would be inserted multiple times in the new set if w_i is high. After the resampling step the probability of a

sample $P\left[\, \lambda_i^{(t)} \in \Lambda^{(t)} \,\right]$ will equivalent to $P\left[\, x_{1..t} \,|\, y_{1..t} \,\right]$ again.

$$\begin{aligned} \Lambda^{(t)} &\sim w_i^{(t)}\ P\left[\, X_t \,|\, \lambda_i^{(t-1)} \,\right] && \text{(5.38a)} \\ &\sim P\left[\, y_t \,|\, X_t \,\right]\ P\left[\, X_t \,|\, \Lambda^{(t-1)} \,\right] && \text{(5.38b)} \\ &\sim P\left[\, X_{1..t} \,|\, y_{1..t} \,\right] && \text{(5.38c)} \end{aligned}$$

$P\left[\, X_t \,|\, \lambda_i^{(t-1)} \,\right]$ is the odometry model. d_t and $\Delta\phi_t$ are the polar coordinates of vector $\lambda_i^{(t)} - \lambda_i^{(t-1)}$ of the first sampling step. $P\left[\, y_t \,|\, X_t \,\right]$ is the image likelihood model.

The big disadvantage of the SIR method is, that it will produce correlated results. Highly probable samples will occure multiple times in the particle set. The correlation is normally removed by the sampling (Markov transition) step (5.37a). If the noise generated by $P\left[\, X_t \,|\, X_{t-1} \,\right]$ is too small (for example if the robot does not move) then an extra random noise can be added. Alternatively, the importance function may always contain a random noise component

$$B\left[\, X_t \,|\, X_{t-1} \,\right] \;=\; P\left[\, X_t \,|\, X_{t-1} \,\right] + N\left(0, \Sigma\right) \qquad \text{(5.39)}$$

If B is well choosen and if the measurement model $P\left[\, y_t \,|\, X_t \,\right]$ is smooth and not too steep, then $\Lambda^{(t)}$ will sample high probable regions and thus a parameter estimator $\widehat{I_m}(f_n)$ will converge to the real value $I(f_n)$ with $m \to \infty$.

6 Implementation

The software architecture follows the general outline of the probabilistic models. It consists of various objects that live in the C++ namespace *ParticleLoc*. The objects were design to be either usable within the AGILO RoboCup framework, and to be executed (and tested) by small stand-alone command line programs.[3] Figure 13 gives an overview of the relationship between the most important classes.

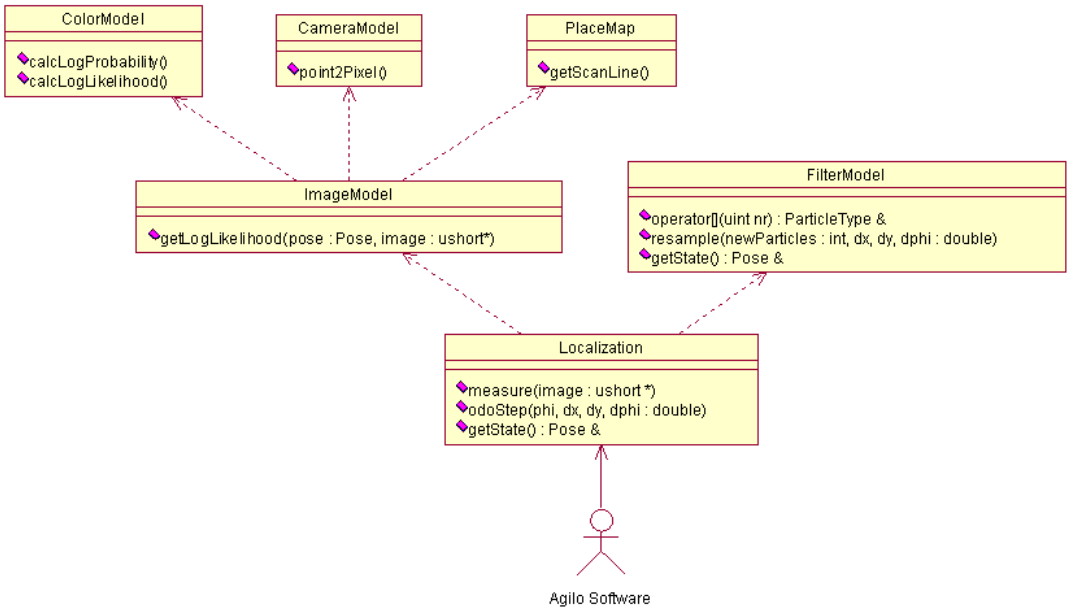


Figure 13: UML class diagram for the voxel-based localization

6.1 Voxel-based Localization

At the first level *ColorModel*, *CameraModel*, *PlaceMap* model the color distributions, the camera distortions and the RoboCup field. If employed within the RoboCup

[3] By setting the compiler symbol *ROBOCUPFREE*.

source, the objects use the standard configuration directory to read the RGB color class lookup table (*rgb16-samples.tiff*) or the internal and external camera parameters (*Camera1.par*, *Camera1-intern.par*). If compiled without the RoboCup software, then the program does not use the Halcon library functions. Instead it uses a simple file format for the configuration files (*rgb16-samples.pgm* and *Camera1.conf*).

At the second level the color, camera and place model is combined to generate the likelihood metric. The most important function of each object is listed in figure 7. The *ImageModel* method *getLogLikelihood* takes a pose and an RGB16 image as parameters, then calls *PlaceMap* to compute a random voxel subset for the pose, converts them to pixel coordinates via the *CameraModel* and then computes the Bayes likelihood of the pose through the *ColorModel*. In addition to the functions shown in figure 7, the *PlaceMap* as well as the *ImageModel* support functions that can be called for a set of poses.

The *FilterModel* organizes a set of particles. Each particle consists of a pose (x, y, ϕ) and its weight. The particles can be accessed via the *[]-operator*. In the resample step the particles are propagated by a difference vector $(dx, dy, d\phi)$, noise is added, and the particles are resampled according to their weight. During resampling, the number of particles is allowed to change. In fact, since in every step 15% uniformly distributed pixels are added, the number of particles usually decreases during the resample step. After resampling a grid-based estimation of the most likely pose ranges is computed. For the particles in the most likely pose range, the average pose is estimated and it can be accessed by the *getState* function.

The *Localization* object provides an interface to the functionality of the image and the (particle) filter model. From the outside, usually *measure* will be called to computer the weights of the particles for an image, *odoStep* to propagate and resample the particles, and the best estimated can again be obtained by *getState*.

6.2 Marker-based Localization

The voxel-based localization has a couple of restrictions. As already mentioned, due to the steep likelihood gradient along the orientation dimension, it may converge to wrong poses or may not reach convergence at all. Furthermore it is to slow and consumes too much computing resources for the application in RoboCup. Therefore a simpler model based on markers is used in the AGILO software.

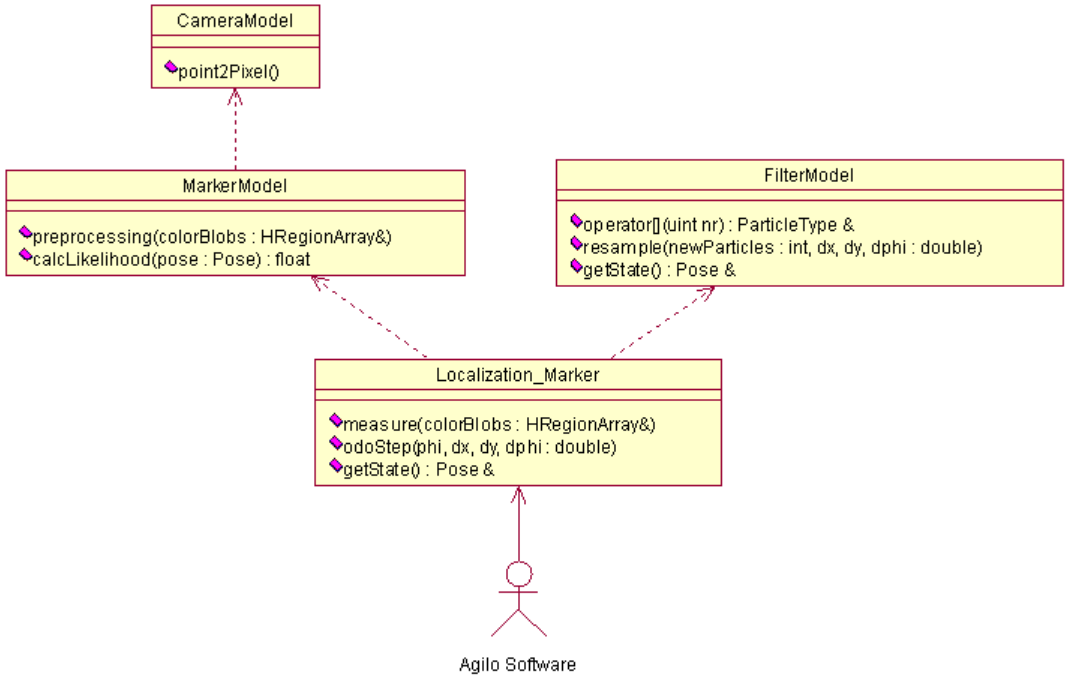


Figure 14: UML class diagram for the marker-based localization

The *MarkerModel* uses morphological operators to find the color borders at the corner flag posts and at the sides of the goal. If at least n borders are found (usually 3) then for each pose in the particle filter the marker are compared with the

expected positions. The particles are weighted according to the distance of the observed and expected positions. These computations are executed in the *preprocessing* function of the *MarkerModel*. *calcLikelihood* estimates the pose likelihood that is passed to the *FilterModel* by the *Localization_Marker* object. The functions of *Localization_Marker* are similar to *Localization*.

ErrorModel The sampling and parameter estimation for 3-dimensional normal distribution is done by the ErrorModel module. It provides two objects: *Gauss* and *Gauss3D* that can be either feeded with one- or three-dimensional values and then computes the mean and (co)variance, or it can be used to sample from such distributions.

7 Localization

In the previous chapters the main component of the localization models were specified: the color model maps the color distributions to probability values, the world model computes the expected color classes for a subset of the pixels and the Monte Carlo Markov Chain methods provides a framework to combine the likelihoods over multiple images (taken along the path the robot is travelling). Although the described models are adequate, for the RoboCup environment it not necessary clear that the combination is a correct model for the localization process too. The image model for instance assumes the independence of the of the color values, which may be violated by the color model (and the color distributions of the images), or the image likelihood function may not fulfill the smoothness or monotony requirements of the MCMC methods. Therefore an empirical evaluation and quantification of these properties is necessary.

The first component of the localization model is the color metric. In the context of the image model two values are important: the likelihood of the color $p\left(rgb|class\right)$ and the (a-priori) probabilities of the color space $p\left(rgb\right) = \sum_i p\left(rgb|class_i\right)$. The effect of the models can be visualized by plotting the logarithmic probabilities for each pixel in an image. Figure 11 on page 29 showed these logarithmic probabilities of the colors in the sample image 11(a). The likelihood distribution for each color class is characterized by its covariance matrix. The probabilities of colors belonging to large classes like the gray background colors are lower than for more narrow classes. In the Bayes equation $\frac{p(color|class)\,p(class)}{p(color)}$, the likelihoods for each color are normalized by $p\left(color\right)$. Thus, the important color values with narrower covariance matrices (yellow, blue) will be weighted higher than the larger color classes. Since the smaller areas (goal, corner flag posts) are

more homogeneous than the larger green and background class there (mis)match will be weighted much higher, effectively using these regions as *probabilistic landmarks*.

As described in chapter 4.3, the pixel likelihoods are then determined for the pixels in the sketch, and the pose likelihood is computed in accordance to equation (4.11). This likelihood, for each possible position at the RoboCup field, is shown in figure 15. The likelihood value along the z-axis shows the maximum likelihood along all possible orientations.

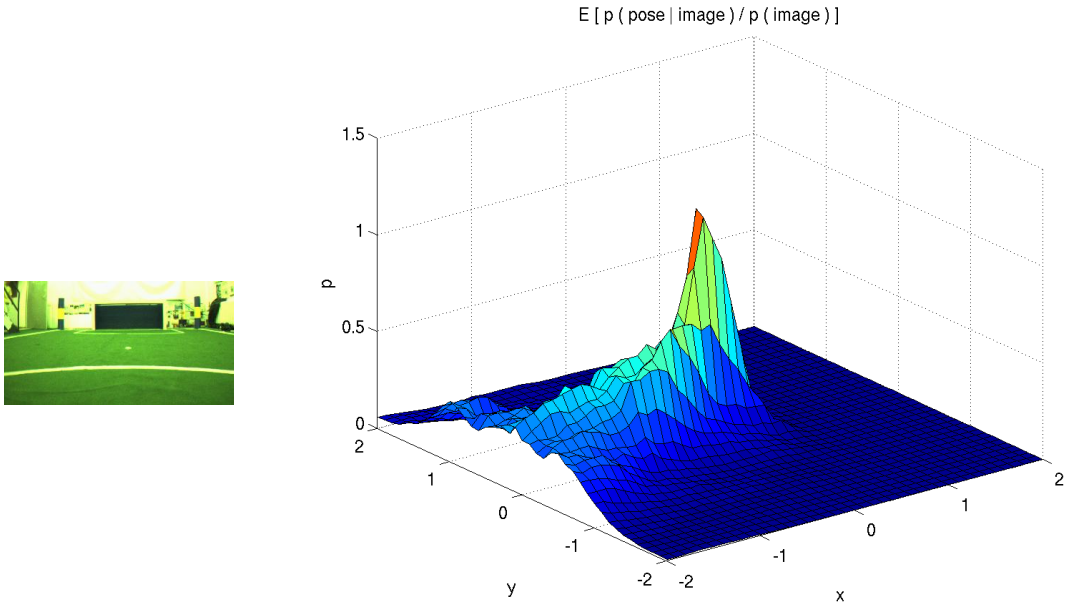
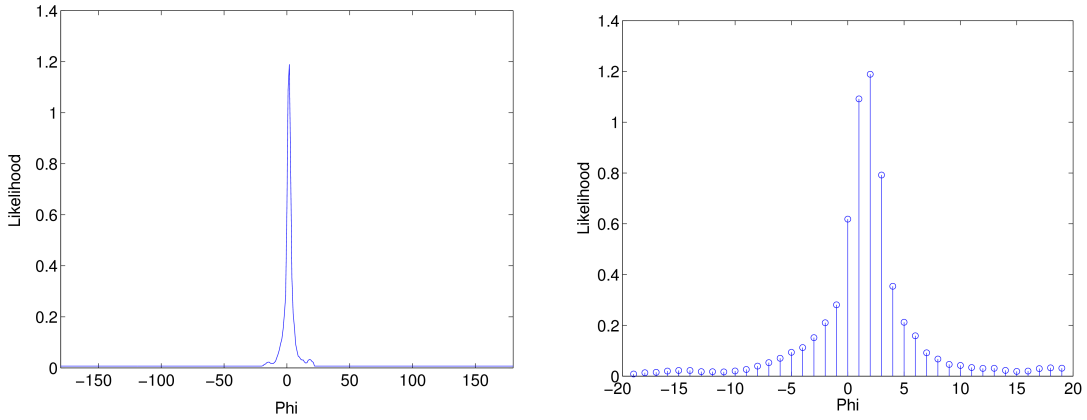


Figure 15: Likelihood distribution in the pose space for the image taken in the center of the RoboCup field

The likelihood distribution in figure 15 is smooth and seems to be well suited to be searched by any maximum likelihood methods. Unfortunately, the situation is different along the orientation dimension. Figure 16(a) shows the likelihood values at the central position $(0, 0)$ as a function of orientation. It is very narrow

and any search probabilistic algorithm must be constructed such that it will find the narrow region of $3 - 4°$ out of $360°$ (compare figure 16(b)). At other positions at the field the likelihood distribution will be even narrower.



(a) $-180 \ldots 180\,\mathrm{deg}$ (b) $-20 \ldots 20\,\mathrm{deg}$

Figure 16: Likelihood distribution in the pose space for an image 15(a) taken in the center of the RoboCup field

The tightness of the distribution has far reaching consequences for the MCMC search method. At first, it will increase the number of particles and the necessary time until the particle distribution will converge. The second aspect is the intrinsic noise of the MCMC method that is introduced at the resampling step. During resampling the particles are chosen according to their probability, and the likely particles will be repeatedly selected for the next iteration. The noise added will then move the partly identical samples randomly such that on average the space between neighboring particles will be filled.[4] Therefore, both the steepness of the likelihood function and the number of particles constitute limiting factors on the size of a minimum region in state space a MCMC method can converge to.

[4]Often, the resampling noise is part of a probabilistic forward step (e.g. an odometry model). But if the robot is robot is not moving, then it is essential to introduce such an artificial noise.

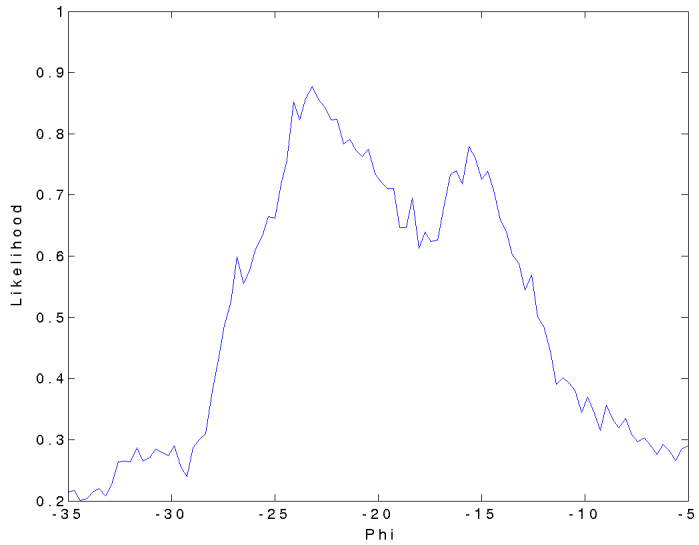


Figure 17: Likelihood distribution of the orientation at position $(0, 2)$ for an image 15(a) taken in the center of the RoboCup field

7.1 Landmark-based MCMC Localization

The major drawback of the localization procedure described so far it is steep likelihood function along the orientation dimension of the pose space. For many application such a narrow function would not be a problem, yet it prevents here the usage of the algorithm as a global relocalization procedure and limits the amount of noise the filter can effectively cope with. For the implementation in the RoboCup software a far simpler and more robust likelihood function was used. Following the fundamental idea of the famous contour tracking algorithm "Condensation" [10], region border information is extracted from the image and a probability metric was defined that compares the predicted with the observed positions.

Two different sets of landmarks were used: the yellow-blue color border of the corner flag posts, and the border between the goal side posts and the colored goal background. These markers can be reliably computed with opening-, closing-

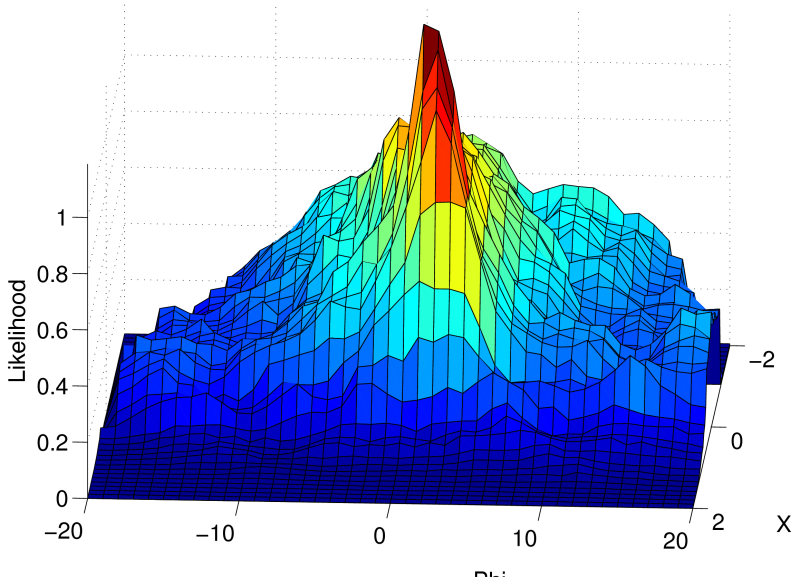


Figure 18: Likelihood distribution in the pose space for an image 15(a) taken in the center of the RoboCup field

and intersection-operators on the color-classified image. These image operators were computed with the proprietary image processing library Halcon [7] that is widely adopted throughout the AGILO RoboCup software. The yellow and blue regions (corner flag posts, goal backgrounds) are expanded vertically, whereas the white regions (goal posts) are expanded horizontally (and constricted vertically). In the best case, the operations result in six overlap regions, two at each visible corner flag post, and two at the side posts of the goal. Thereby, the centers of the intersections correspond to the center of the original color borders.

Occlusions occur in the RoboCup competition very often, mostly because the opponent players go to the ball or defend the goal. Since the robots are colored black with attached labels in turquoise or magenta only, the opponents do not lead to false-positive marker detection. To prevent wrong center detection by partly occluded posts, plausibility checks are applied to the extracted regions. For instance

for the height of the goal posts and the width of the corner flag posts minimum values are enforced.

The world model of the marker metric consists of the color border positions in three-dimensional space. For each evaluated pose, these 3D positions are projected into the image. As for the voxel model, the lens distortions are thereby taken into account too. The metric works on a set of $0 - 6$ labeled centers that are extracted from the image, and on a set of expected positions of up to 6 color borders. Due to the camera configuration, it is not possible to see more than two corner flag posts or more than one goal from any position.

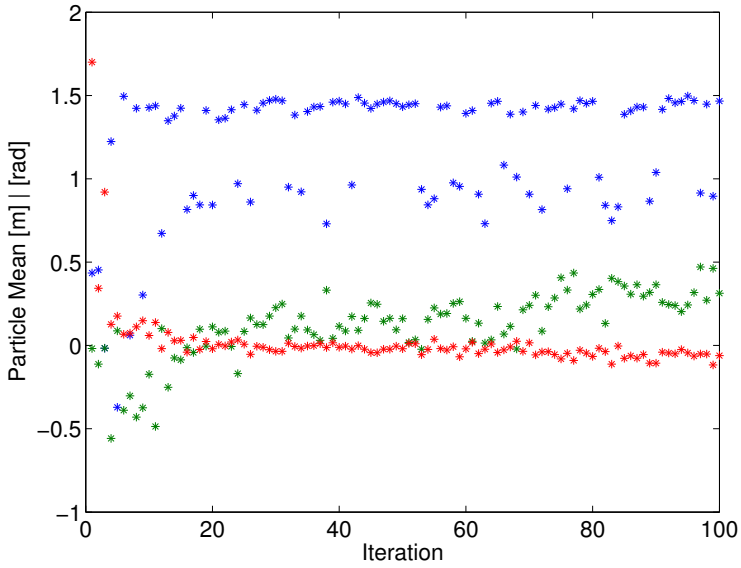
For each of the extracted region centers, the nearest of the expected color borders is determined. The difference of the positions is then normalized with a constant σ that controls the steepness of the likelihood function. Assuming independent normal distributions, the probabilities of the resulting distances are then multiplied. Since positions that are occluded are not used by the metric (they do not appear in y_{image} in the formula below), the likelihood function is quite robust. For images where only one or two markers can be extracted, no likelihood will be calculated and the particles would be propagated with the odometry information only. This constraints prevents overfitting and wrong peaks if only limited information is available.

$$p\left(y_{image}|x_{expected}\right) = \prod_{i} N\left[min_j\left(y_{image,i} - x_{expected,j}\right) \cdot \sigma^{-1}\right]$$

The above likelihood metric was then used in the described MCMC algorithm to iteratively estimate the most likely positions. A typical time course of the particle filter convergence can be seen in figure 19. The robot was placed in front of the opponent goal and a 60s image sequence was recorded. The plot show the de-

velopment of the mean for the first 100 iterations. The orientation and x-position converge after about 10-20 iterations, whereas the y-position cycles between two values after about 10 iterations. The more likely y-coordinate of 1.5 is the correct coordinate, whereas the values between .5 and 1 represent errors from falsely or undetected markers.

(a) Sample image



(b) Particle distribution for an image 19(a) taken in front of the opponent goal. The marker model was used and in each time step 50 % of the particles were resampled.

Figure 19: Convergence of the landmark-based particle filter for a position in front of the opponent goal.

7.2 Image Likelihood Evaluation

As mentioned in the previous chapter the image likelihood metric is sensitive to orientation changes of one or a few degrees. Such small orientation shifts can already be generated by the normal oscillations of the robot platform. In addition, slipping, bumps or collisions can easily generate errors of a much larger magnitude. The voxel-based image likelihood function is therefore not very suitable for the application in RoboCup. Hence, the primary aim of this chapter is not to generally evaluate the performance of the algorithm, but to pinpoint (and verify) the important properties of the function that may be of interest for other scenarios. For the more robust marker model experimental data will be provided too.

To evaluate the localization models, test sequences were recorded at the AGILO RoboCup field in Munich. It is a slightly smaller but otherwise rule-compliant replica of the environments used in the RoboCup middle size league [14]. 10 one-minute episodes were recorded for either a moving robot, and for occluded and non-occluded standstill episodes (figure 20). Each sequence consists of more than one thousand images, the extracted color regions and marker midpoints and the general AGILO log file. In addition, the ground-truth position obtained by a calibrated ceiling camera [25] was recorded. One data set is about 300 MBytes large.

As mentioned above, to evaluate the likelihood functions the robot was placed at different positions on the field (sequences 20(c)-20(i)). The first image of each sequence was used to evaluate the probability function. The general picture is that at all position the likelihood functions have their maximum at the correct poses in state space. However, the steepness of the functions vary greatly. The voxel-based metric is usually very steep. On the contrary, the marker model provides smooth, wide functions. For the kick-off position at the center the voxel-based likelihoods

(a) Center of the field.

(b) Driving through robots.

(c) Center of the field with robots.

(d) Center of the field, oriented sidewards.

(e) Center of the field oriented more sidewards.

(f) Partial occlusion.

(g) Front of the goal.

(h) Front of the goal with robots.

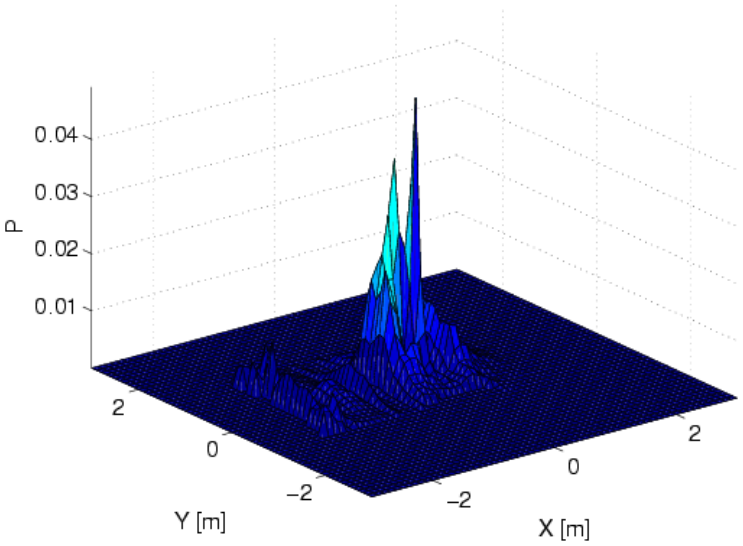
(i) Towards of a corner post.

Figure 20: First images of the sequences at various positions at the AGILO RoboCup arena. In the sequences a) and b) the robot was manually driven towards the goal. In the other experiments the data was collected to evaluate the likelihood metrics. In these cases the robot did not move.

are shown in figure 21(a). The peak of the probability function is correct, yet very narrow. For random sampling algorithms, like MCMC, this increases the average time (or the number of needed particles) until a particle, by change, falls within the elevated region. This would still be acceptable, but the peak of the likelihood function will change due to external influences over time. Thus, a product of the likelihood functions with different peaks would be a random function too and unusable for localization.

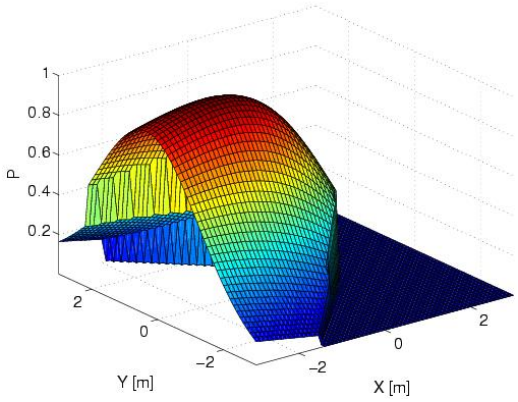
The marker model, on the other hand, is an ad-hoc function, the parameters were not obtained analytical or experimentally, but chosen such the likelihood function shows the required properties (figure 21(c)). The prominent MCMC applications ([10], [27], [29]) employ such ad-hoc models. However, the disadvantage of the marker model is that is uses only very limited proportion of image information. The two yellow-blue borders at the two corner flag posts and the midpoints of the goal posts are used as landmarks. This information is sufficient to determine the position on the field if the orientation if known (in figure 21(c) ϕ was set to 0). Yet, the information is not sufficient to uniquely estimate the pose. The maximum over all orientations in the three dimensional state space is a semicircle in the position plane as shown in figure 21(d).

Partial occlusion, as in figure 22(b), does not impair the likelihood function. As long as at least 4 markers are detected, the algorithm will use them to estimate the probabilities. The obstacles are detected by Bayes maximum classification as described in chapter 3.2. Figure 22(a) and 22(c) show the distributions for an orientation of $\phi = 0$. The functions are not significantly different from the non-occluded case. For other positions, the likelihood function are similar. An example for a position proximal to the goal is shown in figure 23.

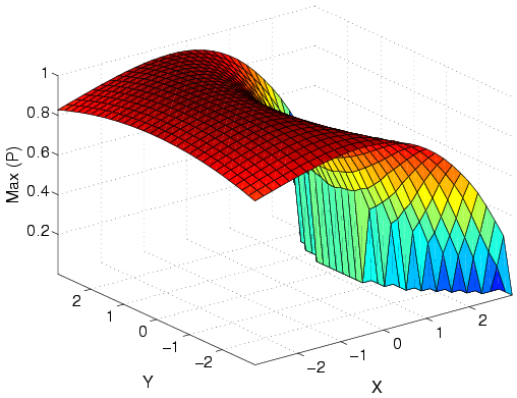


(a) Likelihood distribution in the pose space for an image 21(b) taken in the center of the RoboCup field

(b) Frontal camera image

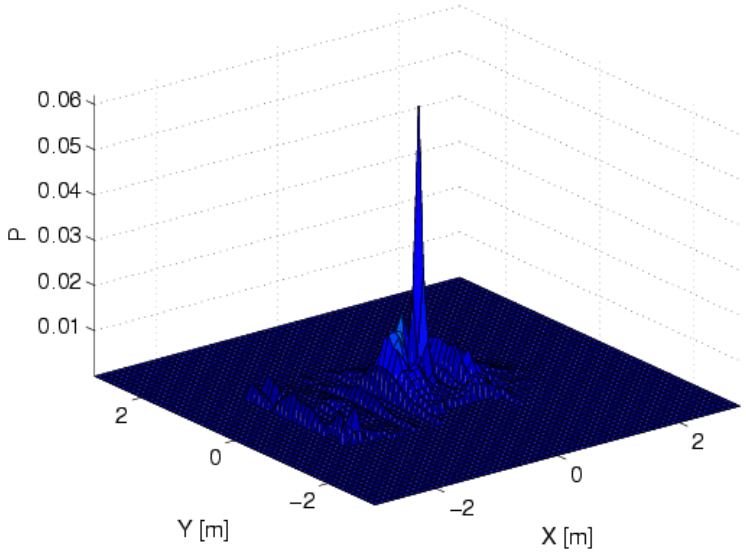


(c) Marker model



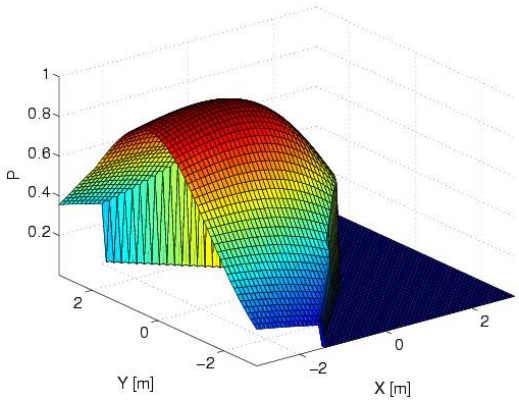
(d) Marker model max

Figure 21: Image likelihood function at the center of the field.



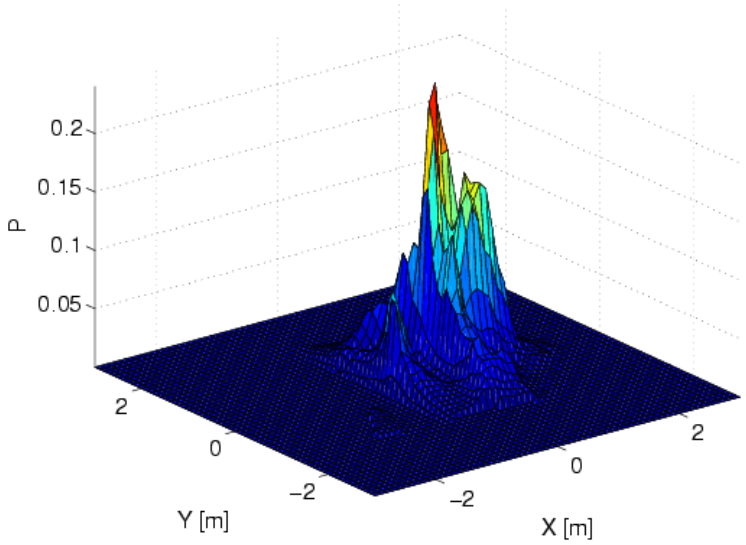
(a) Likelihood distribution in the pose space for an image 15(a) taken in the (x,0) of the RoboCup field

(b) Frontal camera image



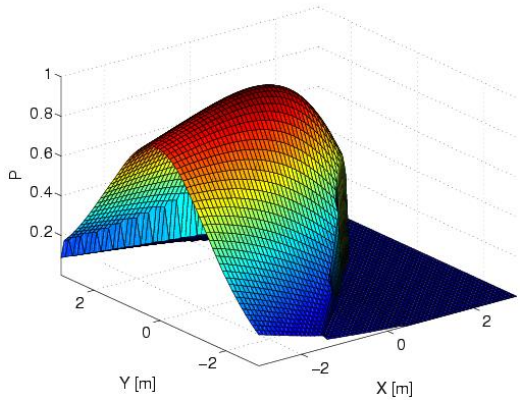
(c) Marker model

Figure 22: Image likelihood function at the center of the field with occlusion.



(a) Likelihood distribution in the pose space for an image 15(a) taken in the (x,-x) of the RoboCup field

(b) Frontal camera image

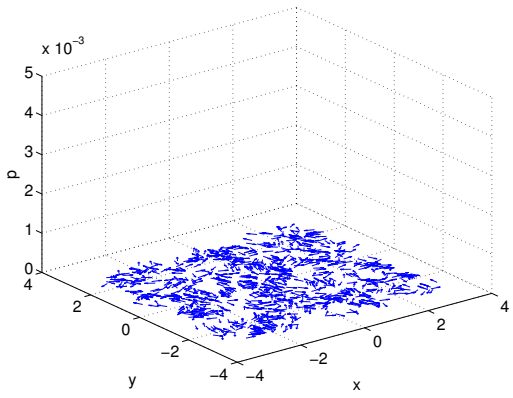


(c) Marker model

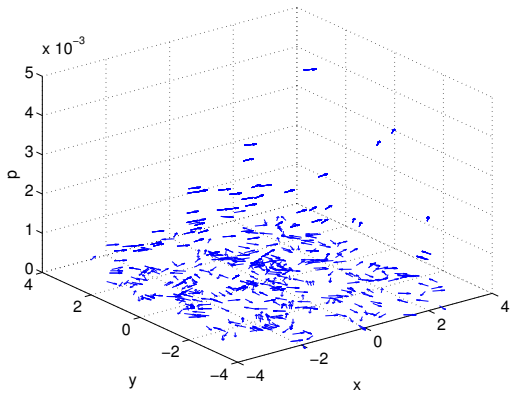
Figure 23: Image likelihood function in front of the goal.

7.3 Resampling Evaluation

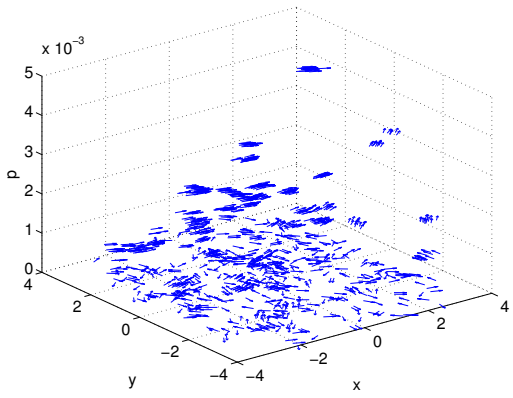
The MCMC algorithm can be described as a Markov process that conducts a random walk, gradient descent search on the likelihood functions shown in the previous charts. To visualize and evaluate the process and the underlying computation, one MCMC iteration will be split into 4 successive steps. The first step (shown in figure 24(a) for the landmark-based localization) is the generation of the a-priori distribution of the particles. In figure 24(a) the a-priori distribution after 3 iterations is shown. At this step a few of the particles have already a good orientation. The probability that a particle exists at a particular position and with a certain orientation is equal to the theoretic probability distribution in the state space. As the next step, the image will be read, the marker position are extracted and the likelihood of the existing particles is computed. The elevation of the particles in figure 24(b) corresponds to their (normalized) likelihood $p\,(pose \,|\, image)$. In the resampling step n particle will be randomly chosen from the existing set. A discreet cumulative probability function over the present particles and binary search is used to efficiently draw the particles according to their likelihood. Thereafter, a small amount of noise is added. If the robot has moved since the last measurement, that mean of the noise will be the odometry vector. The result is shown in 24(c): many particles are generated around the likely positions, whereas the particle density at unlikely positions decreases. This effect can be best seen in 24(d) when elevation (probability) of the particles is reset. The density in the state space now corresponds the a-priori probability of the Bayes equation. As a final refinement, 15 % uniformly distributed particles are added to allow for relocalization after kidnapping or wrong convergence. These particles are raised in 24(d). The new particle set is then used in the next iteration.



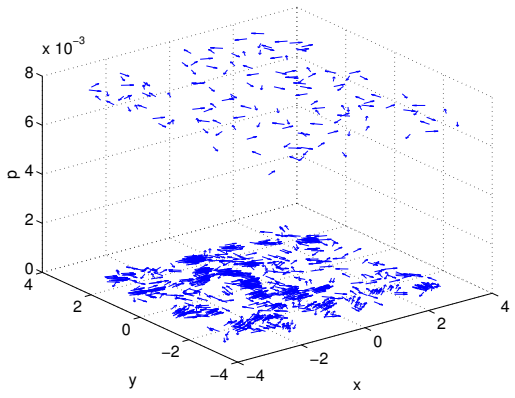
(a) Particle distribution after 3 resampling steps.



(b) Weighting of the particles based on the marker positions in the 4th image.



(c) Particles are selected during resampling according to their likelihood. Very likely particles will be more frequent and random noise is added to distribute the particles in the state space.

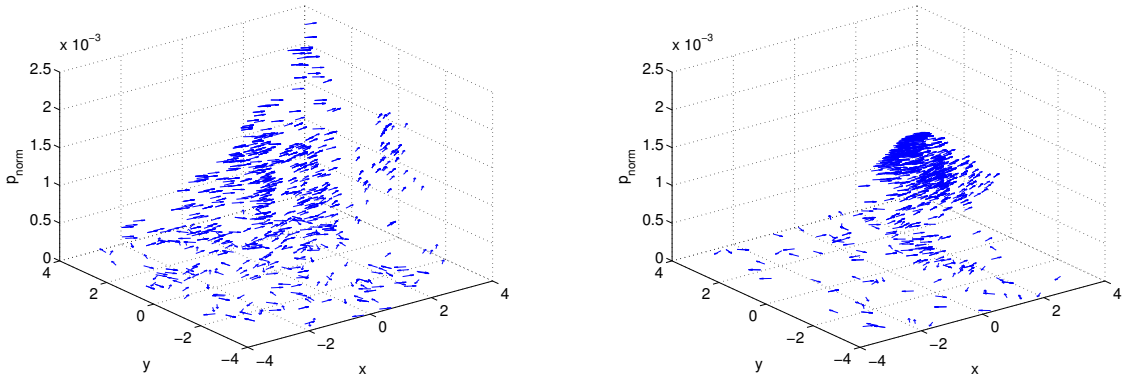


(d) As the last step, 15 % particles with a uniform distribution are added to allow relocalization after kidnapping or sensor errors.

Figure 24: Steps of the MCMC algorithm.

By repeatedly folding the likelihood functions with the existing and blurred particle distribution, the particle filter converges. The (weighted) distribution after 10 iterations is shown in figure 25(a). If the likelihood function always matches the a-priori distribution, then the density around the most likely position will in-

crease with each step and is only limited by the amount of noise added during resampling. Without lost of convergence behavior, the number of particles can be safely reduced as a function of the (co)variance of the particles [6].



(a) 10 iterations

(b) 50 iterations

Figure 25: Distribution of the particles after 10 and 50 iterations. On the z-axis the normalized p value is shown.

7.4 Discontinuities of Image Markers

The possibility of occasional large sensor errors has not yet incorporated into the model. In part, error is implicitly modeled in the additional noise of the resampling phase and the uniformly distributed particles that are introduced in each step. Unfortunately, the true error distribution is unknown and it will be beyond the scope of this paper to develop a plausible probabilistic model.

The robustness of the particle filter can be increased by decreasing the fraction of particles the is resampled in each time step. By leaving a proportion of the particles untouched, well fitting poses will be still available even after one or a few false sensor readings. The fraction of the particles that is resampled in an itera-

tion was varied in figure 26 for the landmark-based localization. If all particles are resampled, the filter will not converge within the first 50 iterations. This suggests that the actual sensor error is at least a magnitude larger than the modeled color error. Increasing the fraction of non-sampled particles drastically improves the convergence. The best results could be obtained with a fraction of $50\ \%$. The true pose of $(1.5, 0, 0°)$ is reliable found after 20 iterations. Decreasing the fraction further to $33\ \%$ impairs the convergence speed, but reduces the frequency of outliers.

7.5 Convergence of the Voxel-based Localization

The voxel-based localization produces a very steep likelihood function that, if distorted by random shifts, will result in an unpredictable combined probability distribution. However, if the propagation is precise and there are only a few outliers in the sensor readings then fast convergence can be obtained with the MCMC methods.

Exemplarily, the results for one test drive from the center of the field towards the goal should be reported. For this sequence, the robot was manually moved and the position was recorded with a ceiling camera. The particle estimates were updated with the position changes of the ceiling camera. Unfortunately, due to technical problems no calibration was obtained for the ceiling camera. Instead the groundtruth estimates were rescaled to match the mean and standard deviation of the predicted path. The results for the filter prediction with 500 particles is shown in figure 27. In the beginning the particle filter converges fast, the variance of the particles decreases in the first 2 seconds and the correct position is then tracked for the next 15 seconds. After about 18 seconds, the robot is turned to the left.

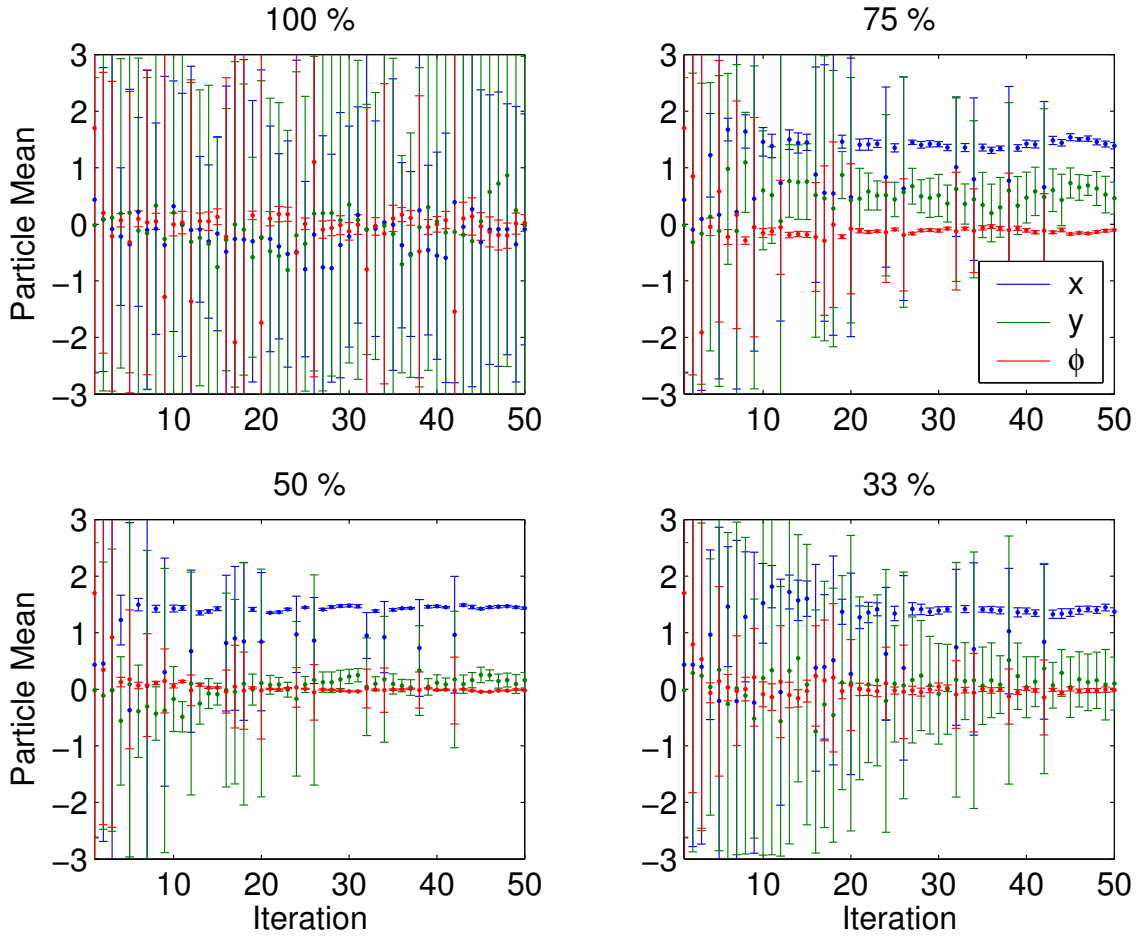


Figure 26: Convergence of the particle filter for the marker model. Shown is the mean and standard deviation of the particles for the first 50 iterations. The proportion of particles that are evaluated in each iteration was varied among the four plots.

This increases the standard deviation of the particles along the y axis. Yet the position is correctly tracked. After 22 seconds, when the robot starts turning to the right, the particle filter underestimates the y-position. After turning stops at about 28 seconds, the correct position is found again and tracked until the end of the sequence. Thus with the exception of a 5 second interval, the position could be correctly tracked.

However, it should be re-emphasized that this result is rather atypical. For other sequences the particle mean does not converge, but rather alternates between different positions due to the small changes of the pose of robot platform.

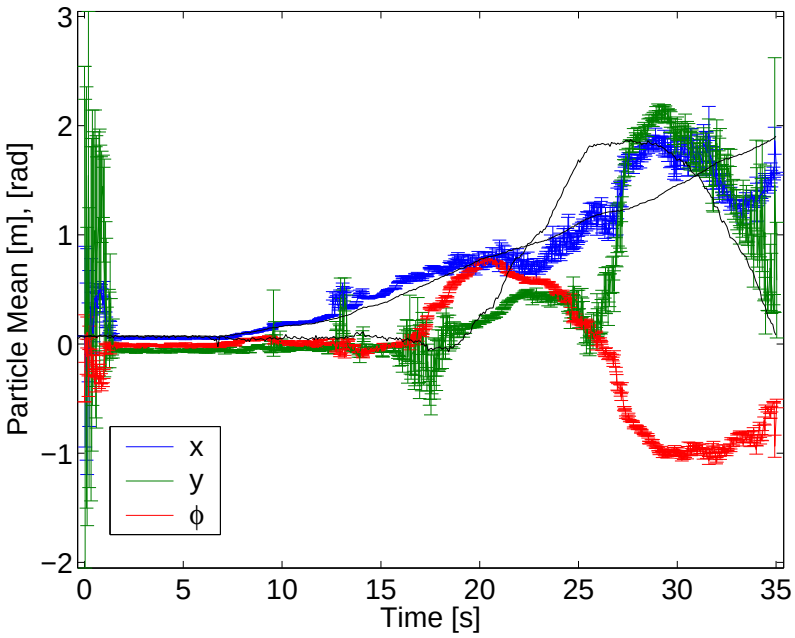


Figure 27: Convergence of the particle filter for the voxel-based localization. Shown is the mean and standard deviation of the particle for a test driving sequence from the center of the field towards to goal. 500 particles were used. The variation of the ground truth for the x and y position is shown in black.

8 Conclusion

Monte Carlo Markov Chain (MCMC) models are a powerful class of methods that can be flexibly adapted to wide variety of common problems in computer vision. To globally localize a mobile robot in the RoboCup environment, a precise color statistic and a simple voxel-based rendering technique was combined using an importance resampling MCMC method. The basic idea was thereby to compare the images from the robot's frontal camera directly with the sketch generated by the voxel rendering engine in order to estimate the likely robot poses. The Bayes color statistic is the only criteria used for the comparison step.

The algorithm starts with a uniform distribution of possible poses. For each pose, a sketch of the image is generated and the color model is used to compute the disagreement between the sketch and the actual image. The probability distributions of the color classes in RGB space is derived from a set of manually labeled sample images. The weighted pose estimates are then resampled in accordance to their likelihoods and the particles are propagated by the current odometry information.

The Bayes color model is capable to reliably identify the different object classes in RoboCup (illustration in figure 11). The computation of the RGB likelihoods is efficiently implemented as lookup tables for the 16-bit RGB color space. The voxel-based rendering technique has been modified to compute the pixel color classes as fast as possible. Randomization was introduced to generate an approximately equal density of points in the image sketch and lens distortions are modeled with the camera model of the AGILO RoboCup software.

The voxel-based image model produces very steep likelihood function. These implicit functions have been calculated for three images taken at the central po-

sition (figure 21), with partial occlusion (figure 22) and in front of the goal (figure 23). For a sample test drive, the particle convergence has been reported (figure 27). However, the very steep likelihood functions are in general very sensitive to noise in the position and especially along the orientation dimension. In addition, for the use in the RoboCup competition a more robust, landmark-based image metric was implemented.

The convergence of the particle filter could be greatly enhanced, if only a fraction of the particles was resampled at every time step. This increases the tolerance for outliers since in every time step there exist a decaying proportion of estimates from previous time steps. Yet, the overall convergence of the particle filter is not satisfying and the use of 500 particles is clearly inefficient. It would be very helpful to mathematically formalize and quantify the degree of dependency between successive iterations since the outliers present a major problem for the current implementation. Further research may investigate the possibility of combining high level features (such as known landmarks) and the brute force, voxel-based image comparison. Likely regions in state space can be identified by a feature-based comparison and could then be sampled in finer detail by the voxel algorithm.

References

[1] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions of Signal Processing*, 50:174–188, 2002.

[2] P. Billingsley. *Probability and Measure*. Wiley, 1995.

[3] P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation*. Springer, 1987.

[4] L. Devroye. *Non-Uniform Random Variate Generation*. Springer, 1986.

[5] A. Doucet. On sequential simulation-based methods for bayesian filtering. Technical Report CUED/F-INFENG/TR. 310, Cambridge University Department of Engineering, 1998.

[6] A. Doucet, S. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10(3):197–208, 1998.

[7] W. Eckstein and C. Steger. Architecture for computer vision application development within the horus system. *Electronic Imaging*, 6:244–261, 1997.

[8] E. Anderson et al. Lapack users' guide. Technical report, SIAM, 1995.

[9] Luca Iocchi and Daniele Nardi. Self-localization in the robocup environment. In *RoboCup*, pages 318–330, 1999.

[10] M. Isard and A. Blake. Condensation conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–29, 1998.

[11] Kitagawa. Non-gaussian state-space modeling of nonstationary time series. *J Amer Stat Assoc*, 82:1032–1063, 1987.

[12] Gudrun Klinker. *A Physical Approach to Color Image Understanding*. AK Peters, 1993.

[13] A. Kong, J. S. Liu, and W. H. Wong. Sequential imputations and bayesian messing data problem. *J Amer Stat Assoc*, 89:278–288, 1994.

[14] G. Kraetzschmar. Rules for robocup-2000 middle size robot league. Technical report, 2002. URL `http://robocup.elet.polimi.it/MSL-2002/Rules2002/rules02/rules2002.html`.

[15] J. S. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998.

[16] D. J. C. MacKay. *Introduction to Monte Carlo methods*. MIT press, 1999.

[17] I. Mikic, M. Trivedi, E. Hunter, and P. Cosman. Human body model acquisition and tracking using voxel data. *International Journal of Computer Vision*, 53 (3):119–223, 2003.

[18] R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[19] D. Neumann. Kalman-filter und partikel-filter zur selbstlokalisation von robotern. Technical report, Munich University of Technology, April 2003.

[20] C. F. Olson. Probabilistic self-localization for mobile robots. Technical Report DS-16543, Jet Propulsion Laboratory, California Institute of Technology, 2000.

[21] P. Perez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In *Proc. European Conf. on Computer Vision*, volume 1, pages 661–675, 2002.

[22] M. K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–, 1999.

[23] Computational Science Education Project. Introduction to monte carlo methods. Technical report, 2002. URL `http://csep1.phy.ornl.gov/mc/mc.html`.

[24] D. B. Rubin. *Using the SIR Algorithm to Simulate Posterior Distributions*. Oxford University Press, 1988.

[25] T. Schmitt and M. Beetz. Designing probabilistic state estimators for autonomous robot control. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2003.

[26] S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. CVPR*, pages 1067–1073, 1997.

[27] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot minerva, 2000.

[28] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. R. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A tour-guide robot that learns. In *KI - Kunstliche Intelligenz*, pages 14–26, 1999.

[29] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 2001.

List of Figures

Index