Grid-based Obstacle Mapping and Gaze Control for Soccer Playing Humanoid Robots

Gitterbasierte Hinderniskartierung und Blickrichtungssteuerung für fußballspielende humanoide Roboter
Bachelor Thesis

Alexander Stumpf

30. November 2010

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Grid-based Obstacle Mapping and Gaze Control for Soccer Playing Humanoid Robots
Bachelor Thesis
Eingereicht von Alexander Stumpf
Tag der Einreichung: 30. November 2010

Gutachter: Prof. Dr. Oskar von Stryk
Betreuer: Dipl.-Inform. Stefan Kohlbrecher

Technische Universität Darmstadt
Department of Computer Science

Simulation, Systems Optimization and Robotics (SIM)
Prof. Dr. Oskar von Stryk

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelor Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 30. November 2010 Alexander Stumpf

Abstract

An important research topic at the Simulation, Systems Optimization and Robotics group at TU Darmstadt is research and development of autonomous robot systems. One of the research projects is the Darmstadt Dribblers, developing autonomous soccer playing humanoid robots and participating in the annual RoboCup competition.

In this work a new approach for modeling the environment of the robot is introduced. For this purpose an occupancy grid mapping system is developed which provides a more detailed model of obstacles in the dynamic environment than previously used approaches.

Furthermore, an entropy-based active gaze control system generating gaze commands to maximize perceptual input is introduced. This system accounts for the reliability estimates in existing modules like ball modeling and is used to automatically control the camera gaze of the robot. Evaluation of the system shows that the system performs well on real hardware.

Kurzzusammenfassung

Ein wichtiges Forschungsthema des Fachgebiets Simulation, Systemoptimierung und Robotik der TU Darmstadt ist die Erforschung und Entwicklung autonomer Robotersysteme. Das Darmstadt Dribblers Team ist eines der Forschungsprojekte, welches autonome fußballspielende humanoide Roboter entwickelt und damit jährlich am RoboCup Wettkampf teilnimmt.

Im Rahmen dieser Arbeit wird ein neuer Ansatz zur Modellierung der Roboterumgebung vorgestellt. Hierfür wird eine gitterbasierte Hinderniskartierung entwickelt, welches im Gegensatz zu dem bisherigen Ansatz detailgetreuere Informationen über Hindernisse in einer dynamischen Umgebung zur Verfügung stellt.

Zusätzlich wird eine auf Entropie basierende Blickrichtungssteuerung eingeführt, welche neue Kamerakopfbewegungen für maximale visuelle Eingabedaten generiert. Das System berücksichtigt hierbei die Schätzung der Zuverlässigkeit der bereits existierenden Module wie z.B. des Ballmodells und wird dazu verwendet die Kamerablickrichtung des Roboters automatisch zu steuern. Die Evaluierung des Systems zeigt, dass das System gute Ergebnisse mit der echten Hardware erzielt.

Contents

1 Introduction

1.1 Motivation

Initiated in 1997, the RoboCup initiative pursues the vision that by the year 2050, humanoid robots should be able to win a soccer match against the human Soccer World Champion team. Inspired by this idea, the Darmstadt Dribblers of the SIM group at TU Darmstadt participates in the RoboCup championship since 2004.

The current soccer scenario in the RoboCup Humanoid Kid Size League involves two teams, each with three players playing on a 4m x 6m field. The soccer match is split into two half times of ten minutes each while the rules are evolved successively towards the FIFA-rules. External influence and help is not allowed, meaning the robots must be able to play soccer autonomously. For this purpose they have to carry all necessary sensors, computational resources and energy supply with them. When a robot is placed on the field no remote control is allowed except start and stop signals. However, the robots are allowed to use wireless communication to exchange knowledge such as estimated ball positions. They are also allowed to communicate for role negotiation, so dynamic team behavior is possible.

As the robots have to act autonomously on the soccer field, all programming and configuration has to be done before. To be feasible for operation on the real robot, the computational capabilities of the hardware have to be considered for every implemented part of the control software. Playing soccer demands a capable world model which is generated from available sensor data. For this purpose, the robot is equipped with monocular camera whose images are used to obtain measurements of the state of the surrounding environment. This environment representation is needed by the robot for planning purposes and obstacle avoidance. In recent years the Darmstadt Dribblers used a sector-based obstacle modeling approach. This approach does not provide accurate obstacle modeling and information about free space, thus an approach providing more accurate and consistent obstacle data is desirable. The new approach should enable robots to perform more reliable goal-oriented path planning and more complex team behaviors (i.e. ball passing). Case studies about successful robotic systems by Kortenkamp et al. [22] showed that occupancy grid mapping is one of the most successful environment modeling approaches in mobile robotics. For this reason, an approach based on occupancy grid mapping is adopted for obstacle modeling in this work.

The RoboCup Humanoid Kid Size League rules define the type of allowed sensors and their operational limits (i.e. maximum field of view). Due to limited sensing capabilities, sensor systems (here the camera) are often mounted on articulated joints. The field of view of the camera and motion capability of actuators is limited, thus it is necessary to increase perceptual input with an active gaze control system. The current system of the Darmstadt Dribblers is based on predefined head movement trajectories. This approach does not consider the reliability of world state estimation, so it is possible that important features needed by cognitive processes are not available, because the gaze control does not point the camera in the right direction. For this reason an active gaze control is implemented, improving the performance of the occupancy grid map, ball modeling and self-localization modules. Instead of using camera images, a new approach for active gaze control based on already modeled informations and the occupancy grid map is introduced.

1.2 Overview of Sections

This section gives a short overview over the work's structure. In **chapter 2** some important basics are described, establishing a better understanding of the work's topic and motivation.

Chapter 3 illustrates the DD2010 robot used as experimental platform in this work. Furthermore a quick overview of the software architecture is given here.

The **chapter 4** gives a overview of other publicized approaches in occupancy grid mapping and gaze control. Furthermore, the state of the art in the RoboCup legged leagues is presented as well as the current approach used by the Darmstadt Dribblers, which is analyzed for weak points.

In **chapter 5** the basic concepts are described. Finally, the implementation and developed techniques are specified in **chapter 6**. Here are all details and further design consideration explained.

All experiments and their results are described in **chapter 7** and the conclusion and outlook for further work is given in **chapter 8**.

1.3 Source Code

All source code used and written for this work is located in the Subversion repository of the Darmstadt Dribblers. As the source code depends on existing code resources of the Darmstadt Dribblers, there is no stand-alone code of this work available.

2 Foundations

2.1 Notation

In this work the following notation will be used:

x_t		State vector of robot at time t
z_t		Measurement vector at time t
$x_{0:t}$	$:=$	$x_0, x_1, ..., x_t$
$z_{0:t}$	$:=$	$z_0, z_1, ..., z_t$
m		Grid map
$\mathbf{m}_i$		Cell with index i of the grid map m
$p(\mathbf{m}_i)$	$:=$	$p(\mathbf{m}_i = 1)$, cell's probability of being occupied
c_t		Count value at time t
$p_0(\mathbf{m}_i)$		Cell's prior state
$bel_t(\mathbf{m}_i)$		Belief about cell's state at time t
${}^{C_1}T_{C_2}$		Homogeneous transformation matrix from coordinate system C_1 to C_2
$H(X)$	$:=$	$\sum_{i=1}^{n} p(x_i) log_b p(x_i)$ Shannon entropy of a set X
$H(\mathbf{m}_i)$	$:=$	$p(\mathbf{m}_i) log_2 p(\mathbf{m}_i)$ Shannon entropy of grid cell with index i
$H(\mathbf{m}_{x,y})$	$:=$	$p(\mathbf{m}_{x,y}) log_2 p(\mathbf{m}_{x,y})$ Shannon entropy of grid cell at position $(x, y)^T$

2.2 Soccer field

The soccer field consists of two goals, two poles and a conventional green carpet. The goal frames are colored blue or yellow which are also the used team colors. The poles consist of three colored segments of 15 cm height and are placed in the intersection area between touch line and center line which can be used for self-localization purposes.

The lines are normal white tape with a width of 5cm. Additional three crosses are applied to the soccer field giving additional localization possibilities. The crosses are placed in the center of each field half and at the kick-off point.

The environment is defined by humanoid league rules [1]. In particular, all dimensions of the field are specified accurately in these rules and can be utilized among other in self-localization and map building. Figure 2.1 visualizes all dimensions which are outlined in table 2.1.

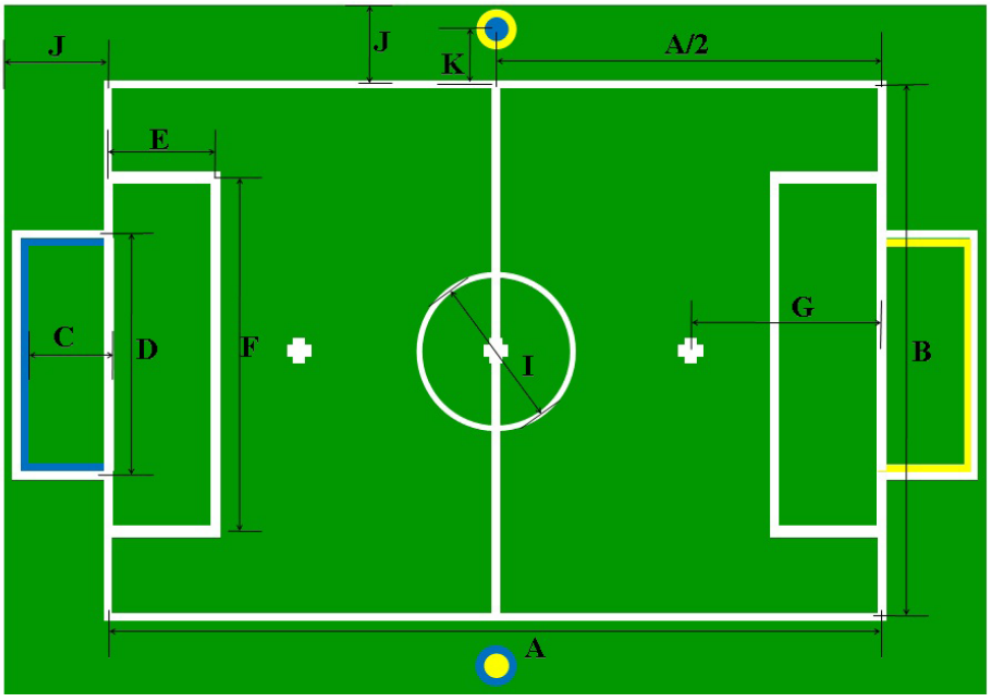


Figure 2.1: RoboCup soccer field dimensions in KidSize league [1].

	description	dimension [cm]
A	Field length	600
B	Field width	400
C	Goal length	50
D	Goal width	150
E	Goal area length	60
F	Goal area width	300
G	Penalty mark distance	180
I	Center circle diameter	120
J	Border strip width (min.)	70
K	Distance of pole to field	40

Table 2.1: Dimensions of soccer field [1].

2.3 Cartesian Coordinate System and Transformations

All collected and stored data is given in different coordinate systems whose relationship has to be defined. All used coordinate systems respect the right-hand rule.

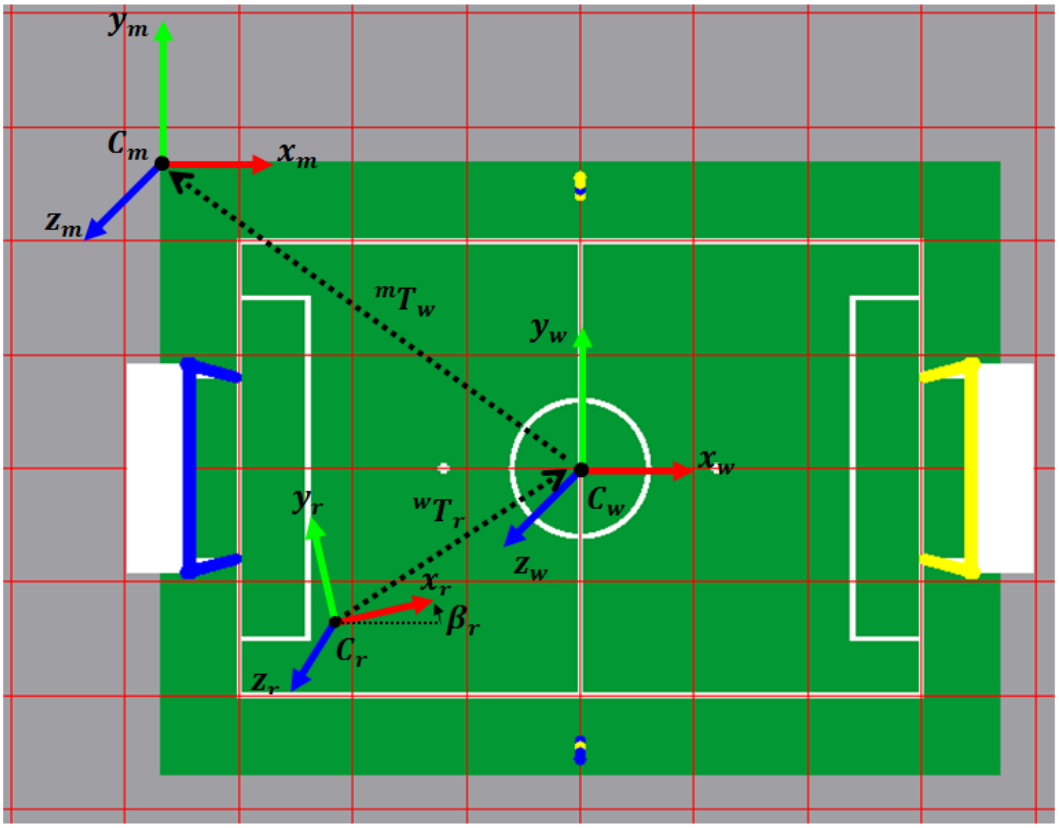


Figure 2.2: World (C_w) and map (C_m) Cartesian coordinate systems.

In figure 2.2 an overview of world and map coordinate systems is given:

- $\mathbf{C_w}$ is the word coordinate system and is used as global reference frame. The origin is placed at the center of the soccer field which enables to utilize the field's symmetric properties.
- $\mathbf{C_m}$ defines the map coordinate system which is fixed relative to the world coordinate system. But the origin is placed at the soccer field corner, so the map can be stored and addressed easily with an array.
- $\mathbf{C_r}$ is the robot base frame which is defined relative to the world frame. This frame can move relative to the world frame.
- ${}^{\mathbf{m}}\mathbf{T_w}$ is the homogeneous transformation matrix between the world and map coordinate systems. In this special case the transformation consist only of a translation. No rotation is needed here.
- ${}^{\mathbf{w}}\mathbf{T_r}$ is the homogeneous transformation matrix between the robot and world coordinate systems. Due to the robot is able to move around the soccer field, the homogeneous transformation matrix ${}^wT_r$ has to be determined by robot's self localization process.
- $\beta_\mathbf{r}$ is the rotation angle of the robot coordinate system's z-axis relative to the world frame.

Additional some definitions for sensing are needed. In this case only the camera's coordinate system as well as tilt and pan has to be defined. An overview is given in figure 2.3:

- $\mathbf{C_c}$ describes the camera coordinate system and is used for projection of camera pixels to soccer field ground in robot relative coordinates. This matrix includes only the extrinsic camera parameters.
- $\alpha_\mathbf{c}$ is the tilt angle of the camera head.
- $\beta_\mathbf{c}$ is the pan angle of the camera head.

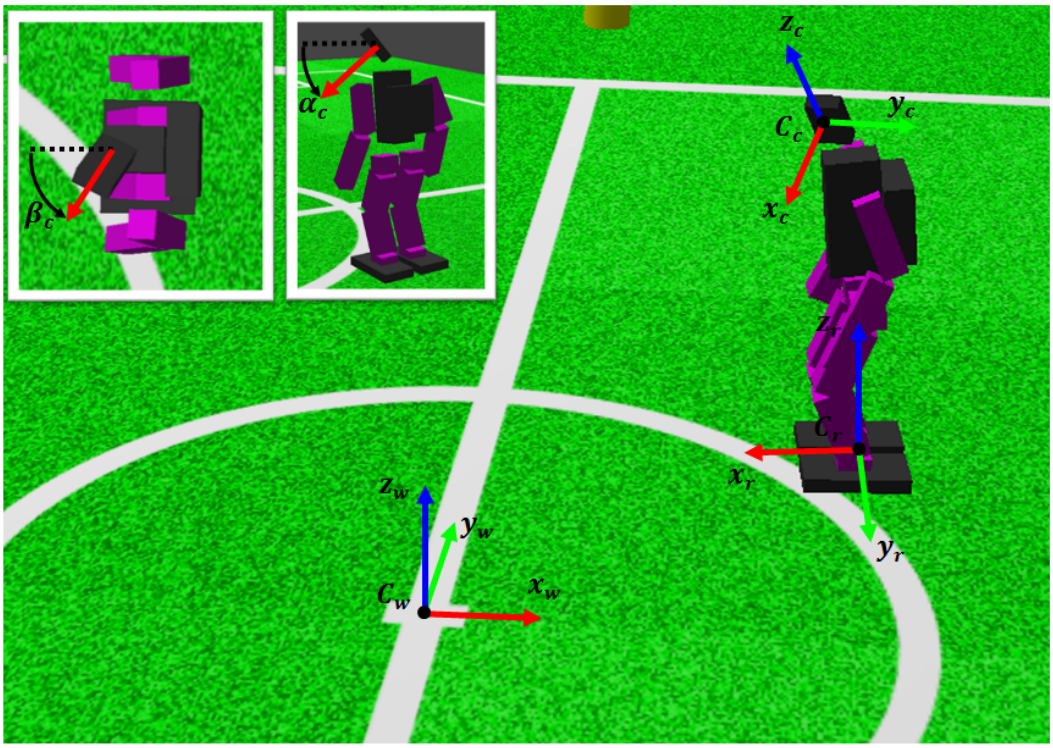


Figure 2.3: Robot (C_r) and camera (C_c) Cartesian coordinate systems.

A transformation between different coordinate systems is realized by homogeneous transformation matrices. The transformation matrix ${}^{C_1}T_{C_2}$ transforms a point ${}^{C_2}p$ of the coordinate system C_2 into the corresponding point ${}^{C_1}p$ in coordinate system C_1. For instance, every world coordinate ${}^{w}p$ can be transformed into map coordinate ${}^{m}p$ with a given homogeneous transformation matrix ${}^{m}T_{w}$ using the equation:

$$ {}^{m}p = {}^{m}T_{w}\ {}^{w}p \tag{2.1} $$

It is also possible to use chains of homogeneous transformation matrices to realize more complex transformations. For instance, a robot relative coordinate ${}^{r}p$ can be transformed into map coordinate ${}^{m}p$ with multiple use of transformations. In the first step the homogeneous transformation matrix ${}^{w}T_{r}$ can be used to transform ${}^{r}p$ into world coordinate ${}^{w}p$. Afterwards the resulting coordinate can be transformed into map coordinate using equation 2.1. Of course the whole transformation can be processed in one single step by previous multiplication of transformation matrices:

$$ {}^{m}T_{r} = {}^{m}T_{w}\ {}^{w}T_{r} \tag{2.2} $$

$$ {}^{m}p = {}^{m}T_{r}\ {}^{r}p \tag{2.3} $$

2.4 Occupancy Grid Maps

Building maps is an important challenge for mobile autonomous robots. For this purpose occupancy grid maps are often used. They were first introduced by Moravec and Elfes in [26] called evidence maps. These maps represent the environment in a metric grid. In theoretical considerations the posterior over all possible maps m can be estimated usually with a set of sensor measurements $z_{1:t}$ up to time t and the robot's path $x_{1:t}$:

$$ p(m|z_{0:t}, x_{0:t}). \tag{2.4} $$

As already mentioned an occupancy grid map divides the environment into a finite set of cells. For this reason let $\mathbf{m}_i$ denote a single cell with index i:

$$m = \{\mathbf{m}_i\} \tag{2.5}$$

Every cell has a binary state which specifies whether a cell is occupied ("1") or free ("0"). Furthermore $p(\mathbf{m}_i = 1)$ or $p(\mathbf{m}_i)$ describes the cell's probability of being occupied. The main problem with equation 2.4 is the dimension space. Just a small grid map e.g. with a dimension of 100x100 cells can represent $2^{10.000}$ different maps. Calculating the posterior of each possible map is too expensive and for this reason unattractive. But with the assumption that all cells are independent from each other the problem can be broken down into a collection of subproblems estimating the probability of each single cell:

$$p(\mathbf{m}_i | z_{0:t}, x_{0:t}). \tag{2.6}$$

The most common update method used for a single cell is the *Binary Bayes Filter* which is described in the following.

2.4.1 Binary Bayes Filter

The binary Bayes filter addresses binary state estimation. In robot application the binary Bayes filter enables estimation of a binary quantity from a sequence of sensor readings. In the context of occupancy grid mapping it can be used to estimate the cells' probability of being occupied. When the cell's state is static, the belief is reduced to a function of measurements:

$$bel_t(\mathbf{m}_i) = p(\mathbf{m}_i | z_{0:t}, x_{0:t}). \tag{2.7}$$

Because the cells have a binary state, the complement belief can be computed with:

$$bel_t(\neg\mathbf{m}_i) = 1 - bel_t(\mathbf{m}_i). \tag{2.8}$$

Finally under the assumption that the map is static, the Bayes' update equation is

$$bel_t(\mathbf{m}_i) = 1 - \left(1 + \frac{p(\mathbf{m}_i | z_t, x_t)}{1 - p(\mathbf{m}_i | z_t, x_t)} \frac{bel_{t-1}(\mathbf{m}_i)}{1 - bel_{t-1}(\mathbf{m}_i)} \frac{1 - p_0(\mathbf{m}_i)}{p_0(\mathbf{m}_i)}\right)^{-1}, \tag{2.9}$$

where $p(\mathbf{m}_i | z_t, x_t)$ is usually the inverse sensor model and $p_0(\mathbf{m}_i)$ the cell's prior state. The prior defines the initial cell value without any influence of measurements. Including the cell's prior state in the equation let the belief converge against the prior in absence of any measurement. When the prior $p_0(\mathbf{m}_i) = 0.5$ is given, then the last term disappears and the Bayes' update equation simplifies to:

$$bel_t(\mathbf{m}_i) = 1 - \left(1 + \frac{p(\mathbf{m}_i | z_t, x_t)}{1 - p(\mathbf{m}_i | z_t, x_t)} \frac{bel_{t-1}(\mathbf{m}_i)}{1 - bel_{t-1}(\mathbf{m}_i)}\right)^{-1}. \tag{2.10}$$

Commonly it is convenient to use log odds representation of cell's occupancy probability, because it prevents numerical instabilities and updates can be computed efficiently by a sum. The probability can be transformed into log odds representation with:

$$l(x) := log\left(\frac{p(x)}{1 - p(x)}\right). \tag{2.11}$$

And of course the occupancy probability can be recovered with:

$$p(x) = 1 - \frac{1}{1 + exp(l(x))}. \tag{2.12}$$

The resulting Bayes update equation in log odds representation is:

$$l_t(\mathbf{m}_i) = l_{t-1}(\mathbf{m}_i) + log\left(\frac{p(\mathbf{m}_i|z_t, x_t)}{1 - p(\mathbf{m}_i|z_t, x_t)}\right) + log\left(\frac{1 - p_0(\mathbf{m}_i)}{p_0(\mathbf{m}_i)}\right). \tag{2.13}$$

In this case the log odds equation will also simplify, if the cell's prior is defined with $p_0(\mathbf{m}_i) = 0.5$:

$$l_t(\mathbf{m}_i) = l_{t-1}(\mathbf{m}_i) + log\left(\frac{p(\mathbf{m}_i|z_t, x_t)}{1 - p(\mathbf{m}_i|z_t, x_t)}\right). \tag{2.14}$$

The Binary Bayes Filter uses inverse sensor models which are popular because they can often be used in situations, where measurements are more complex than the binary state (e.g. in dynamic environments).

Special case of Bayes update rule

Using the binary Bayes filter with a measurement $p(\mathbf{m}_i|z_t, x_t) = 0.5$ has no effect. Inserting it into the equation 2.10 has the result:

$$\begin{aligned} bel_t(\mathbf{m}_i) &= 1 - \left(1 + \frac{0.5}{1 - 0.5}\frac{bel_{t-1}(\mathbf{m}_i)}{1 - bel_{t-1}(\mathbf{m}_i)}\right)^{-1} && (2.15)\\ &= 1 - \left(1 + \frac{bel_{t-1}(\mathbf{m}_i)}{1 - bel_{t-1}(\mathbf{m}_i)}\right)^{-1} && (2.16)\\ &= 1 - \frac{1}{1 + \frac{bel_{t-1}(\mathbf{m}_i)}{1 - bel_{t-1}(\mathbf{m}_i)}} = 1 - (1 - bel_{t-1}(\mathbf{m}_i)) && (2.17)\\ &= bel_{t-1}(\mathbf{m}_i). && (2.18) \end{aligned}$$

Hence, it is proven that updating with a measurement of maximal uncertainty has no effect in the binary Bayes filter and can be ignored. Analogously, this can be proven with the log odds variant of binary Bayes filter. This property is used to improve the performance of the implemented approach in this work (see chapter 6).

2.5 Active Gaze Control

Equipping a robot with sensors is always a trade-off in regarding perception capabilities, weight, size, power consumption, etc.. For this reason it is desirable to make most effective use of the available sensor systems.

In the research community, active gaze control is differentiated between overt and covert attention. Overt attention describes the act of directing the sensor to the focused point. Covert attention is the act of mentaly focusing a high stimuli region in sensor input without any physical movements. For overt attention purposes the sensor systems are often mounted on controllable actuators giving the sensor additional degrees of freedom. In case of Darmstadt Dribblers the humanoid robot is equipped with a camera head having two degrees of freedom. For optimal environment exploration the important question has to be answered: "Where to look next?".

Approaches answering to this question are called *Active/Intelligent Gaze Control*, *Active Perception* or similar. There are three commonly used techniques that are used with camera systems which are described in the following sections.

2.5.1 Gaze Control based on Salient Features

One set of approaches is inspired from human physiology, especially human cognition and perception. For this reason the developed methods usually work directly on grabbed camera images. Here, overt as well as covert approaches are possible. The main goal of these methods is to extract salient features from a sequence of camera images. A salient feature could be for instance a pixel whose color or brightness changes fast over a short time. In [38], [39] such salient features are collected in a map to identify temporary spatial high stimuli which should be focused on. However, salient feature based approaches are reactive approaches and have the common drawback that they do not account for the surrounding area which is not grabbed by the camera.

2.5.2 Gaze Control based on Object Tracking

A simple but effective method is based on tracking several known objects. In general, not all known objects can be seen in a camera scene at once. Hence these approaches have the ability to alternate the focused object. For this purpose the approach must be able to recover objects with possibly outdated information. In dynamic environments new object information thus has to be predicted for a successful recovery. A priority scheduling can be used to optimize the control. An approach taking into consideration of object priorities is introduced in [32]. This kind of technique is designed to reduce the uncertainty of known objects. For this reason it is not able to handle previously unknown objects, so new objects or areas are detected or explored randomly.

2.5.3 Entropy-based Gaze Control

Entropy-based gaze control systems address the prediction of an entropy of possible future measurements. For this purpose every possible control is simulated and the entropy of the possible measurement is predicted based on this simulation. Usually these scenarios are high dimensional, so they cannot be solved in realtime on resource constrained systems. This makes it difficult to implement approaches of entropy-based gaze control on small soccer playing humanoid robots. In [3], [33] and [35] are examples given how to migrate these issues. Entropy-based gaze control provides potential for robust exploration and map building, because it considers uncertainty.

2.5.4 Shannon Information Theory

The presented techniques are based on Shannon Information Theory introduced in "A Mathematical Theory of Communication" written in 1948 by Claude Shannon [34]. This theory presents a new measure for average information content called Shannon entropy. Shannon entropy deals with the probability of occurrence of a single value and the information contained in a message whose units are typically given in bits. For computing the Shannon entropy the equation

$$H(X) = \sum_{i=1}^{n} p(x_i) log_b p(x_i) \tag{2.19}$$

is used, in which n is the length of the message and b is the number of different possible message units (usually Bits).

Shannon Information Theory is commonly used for entropy prediction in entropy-based gaze control approaches. For this purpose the following assumptions are used:

1. The closer an object is seen the more details can be obtained by a camera system. For this reason the entropy is assessed in such cases higher than for distant objects.

2. Partial-vision is one of the worst case situations for vision cognition for which reason it is preferred to center objects in the image. Therefor focusing objects provides usually more information content.

The information content of a cell in an occupancy grid map with binary state can be obtained with:

$$H(\mathbf{m}_i) = -(p(\mathbf{m}_i)log_2 p(\mathbf{m}_i) + (1 - p(\mathbf{m}_i))log_2(1 - p(\mathbf{m}_i))). \tag{2.20}$$

The resulting graph is a curve (see figure 2.4), which takes the maximum exactly with $p(\mathbf{m}_i) = 0.5$, representing the maximum amount of uncertainty. In contrary at the left ($p(\mathbf{m}_i) = 0$) and right ($p(\mathbf{m}_i) = 1$) side of the graph the entropy is minimal.

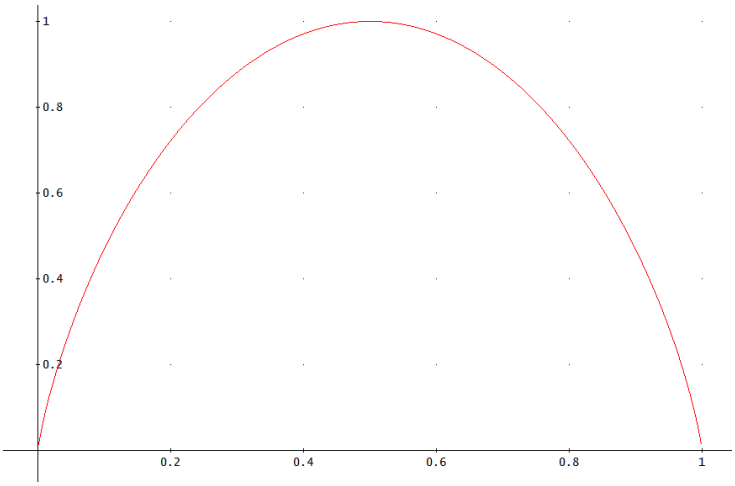


Figure 2.4: Entropy curve for occupancy probability.

2.6 Gaussian Distribution

The Gaussian (or normal) distribution is a continuous probability distribution. This distribution is often used to describe random values. The graph is also known as Gaussian bell curve where random variables cluster around a single mean value denoted by μ. Variance is labeled with σ^2 and defines the width of the distribution. The Gaussian distribution with the parameters $\mu = 0$ and $\sigma^2 = 1$ is also called standard normal. In the simple 1-dimensional case the Gaussian distribution is calculated by:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{2.21}$$

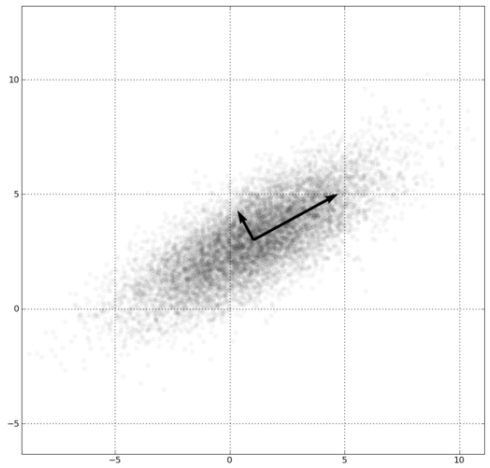
In scientific applications this distribution is commonly used as simple model for complex phenomena. Many deviations in measurements and observations in scientific processes can be described with a Gaussian distribution. For this reason, in this work the Gaussian distribution is used for modeling the uncertainty in measurements and obstacle states.

Furthermore 2-dimensional grid maps are used throughout this work, for which a 2-dimensional Gaussian distribution is needed. The multidimensional Gaussian distribution in the bivariate case is computed by:

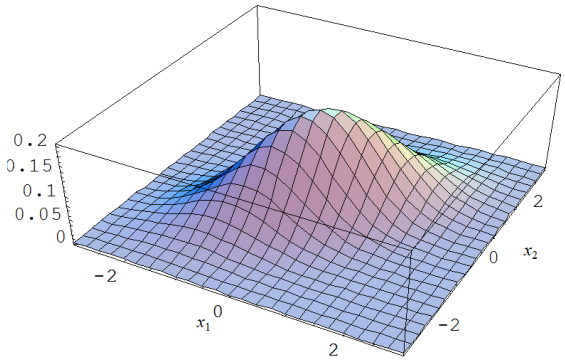
$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left[\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}\right]\right), \tag{2.22}$$

with:

$$\mu = \begin{pmatrix}\mu_x \\ \mu_y\end{pmatrix}, \quad \Sigma = \begin{pmatrix}\sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2\end{pmatrix}, \tag{2.23}$$



(a) Scattering of a Gaussian distribution[1].



(b) Example for 2-dimensional Gaussian distribution[2].

Figure 2.5: Illustrations of Gaussian distribution.

where ρ is the correlation between X and Y which can be determined by covariance matrix Σ:

$$\rho_{X,Y} = \text{corr}(X,Y) = \frac{\Sigma(X,Y)}{\sigma_x \sigma_y} = \frac{E[(X-\mu_x)(Y-\mu_y)]}{\sigma_x \sigma_y}. \tag{2.24}$$

In figure 2.5, two illustrations of probability density of Gaussian distribution are given. The correlation factor ρ is decisive for the distribution's shape.

Given the covariance, the probability for every point can be obtained. For this, the statistical distance to the distribution's mean has to be calculated. This distance in multidimensional space is also called Mahalanobis distance and is defined with:

$$\Delta(x,y) = \sqrt{(x-y)^T \Sigma^{-1} (x-y)}. \tag{2.25}$$

All points (x,y) with the same Mahalanobis distance $\Delta(x,y)$ lie on an ellipse. The probability of a random point drawn from this distribution to lie within the ellipse can be obtained from the χ_2^2 quantiles. The squared Mahalanobis distance matches the χ_2^2 quantile, so for instance, a point drawn from the distribution has the probability of 25% to lie within the ellipse defined with $\Delta(x,y)^2 = \chi_2^2(0.75)$.

2.7 Bresenham Line Algorithm

The Bresenham line algorithm is the most common method for rasterizing lines on n-dimensional grid. The algorithm provides a good performance because it uses only integer addition, subtraction and shifting operators. No floating point operations are needed, so the algorithm can also efficiently be used on architectures without floating point unit. Grid maps are just rasterizated environment representations, so the Bresenham line algorithm can easily be applied there.

As the basic Bresenham algorithm can handle only lines which descend to the right, it can must be generalized in order to support all possible directions. Algorithm 4 in Section 9.2 gives an overview of the generalized Bresenham line algorithm. An example of this algorithm is illustrated in figure 2.6.

[1] http://en.wikipedia.org/wiki/Multivariate_normal_distribution

[2] http://psymet03.sowi.uni-mainz.de/meinharg/Lehre/WS2008_2009/MultivariateI/Stundenfolien/multivariate-distanz-normalverteilung.ppt

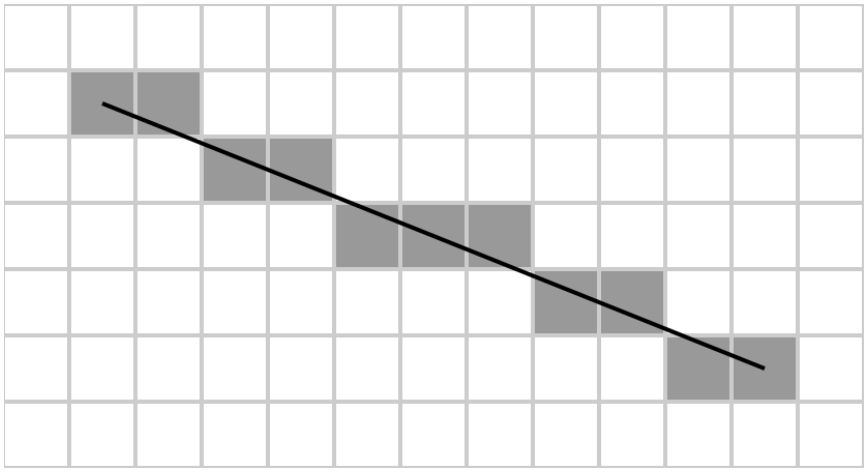


Figure 2.6: Illustration of the Bresenham's line algorithm[3].

The generalized Bresenham algorithm was already implemented for a grid-based SLAM approach[20] which is adapted in this work.

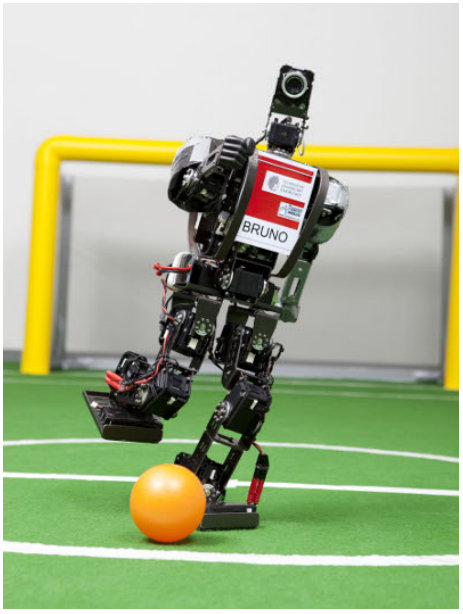
[3] http://en.wikipedia.org/wiki/Bresenham

3 Experimental Platform

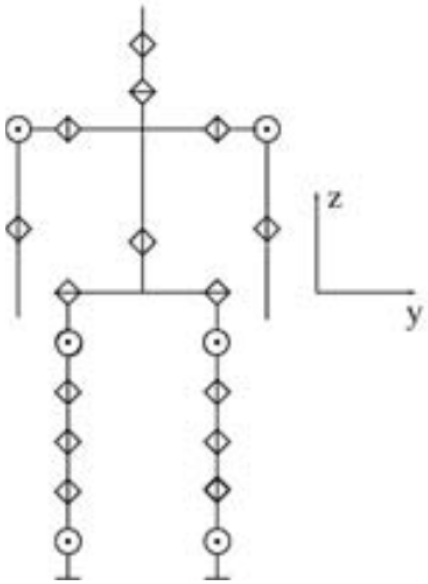
3.1 DD2010 humanoid robot

The implemented approach is designed to be used on the humanoid robot DD2010 of the Darmstadt Dribblers [25]. The DD2010 is based on the Hajime Robot 30 robotic platform (HR30) of Hajime Research Ltd. with minor modifications to the hardware for improving the capability to play autonomous robot soccer (see figure 3.1a). For this purpose the robot is equipped with several sensors in combination with a microcontroller for motion control and an embedded PC board for all other functions like cognitive processing and communication.

3.1.1 Technical Specifications



(a) Humanoid robot Bruno kicking a ball (©Katrin Binner).



(b) Kinematical robot structure [25].

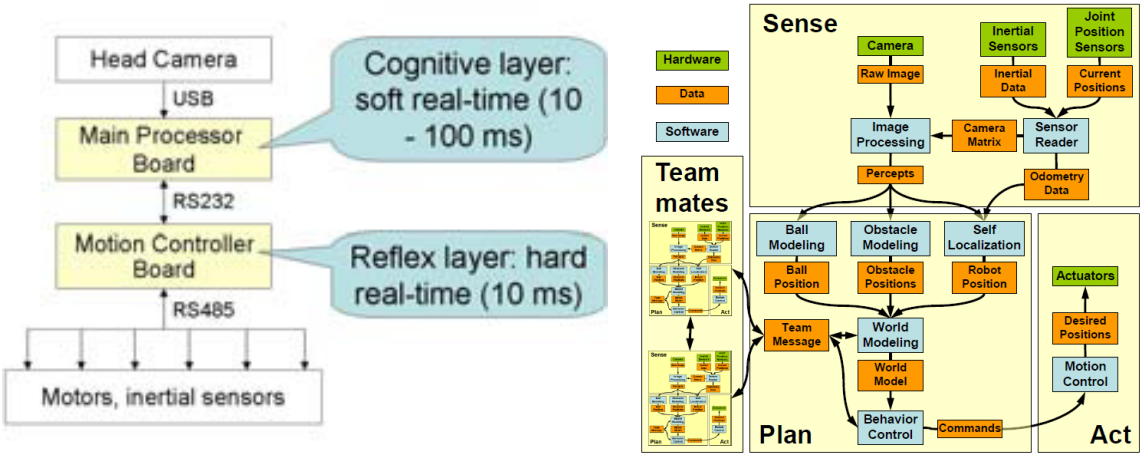
Figure 3.1: DD2010 robot.

The DD2010 robot structure is made of anodized aluminum. The 21 degrees of freedom are partitioned as follows: six in each leg, three in each arm, one in the hip and two in the neck (see figure 3.1b). All joints actuated by Robotis Dynamixel servos (3xRX-64 and 18xRX-28). The servos are connected by a bus system to a microcontroller controlling the motions. A 3-axes-accelerometer board and 3 single axis gyroscopes are used for motion sensing and stabilization.

For cognitive processing and high level behavior control Compulab FitPC2 is used based on Intel Atom technology. This small PC is equipped among others with an Z530 Atom CPU (1,6 GHz), USB ports and 802.11b/g wireless LAN. As the microcontroller communicates with the FitPC2 over RS232, a USB-to-serial converter is used. For environment sensing, a commercial webcam is connected to the FitPC2 via USB. Camera images are grabbed at a 30Hz rate and directly processed.

The DD2010 is powered with a Lithium-polymer battery that provides a maximum operation time of 30min.

3.1.2 Software Architecture



(a) Main layers of processing. **(b)** Architecture of software components.

Figure 3.2: Overview of software structure [25].

The control software is distributed into three layers each computed on a separate hardware (see figure 3.2a). The lowest control layer is performed directly in each of the 21 servos. For this purpose every servo is equipped with a microcontroller controlling position and velocity. Furthermore the operation conditions are monitored, so for instance, an emergency shutdown is possible when the servo overheats.

The second layer is also called reflex layer and controls motion sequences including stability control, running on the main microcontroller. For this purpose a microcontroller is chosen which is able to meet the 10ms control cycle in hard realtime. This enables robust and fast walking with smooth transitions between different walking modes using the 6 DOF robot legs inverse kinematics. The gyroscopes and accelerometer are used to stabilize the walk based on ZMP theory [15]. The layer is also able to detect if the robot has fallen down and allows the robot to stand up autonomously.

At the top level is the cognition layer, where the cognitives functions are processed. The software is based on the middleware RoboFrame described in Section 5.1.2, running on a Linux operating system. Functionality of the cognition layer is distributed over several modules interacting with each other which is illustrated in figure 3.2b. The main parts are the image processing, world modeling and behavior control based on XABSL (see Section 5.1.1). Every component of the world modeling is divided into a modeling and model part. The modeling provides the processing parts and stores all results in the corresponding model providing this data to other modelings and behavior control layer. Furthermore RoboFrame allows to share these models between robots easily.

3.2 Simulator (*MuRoSimF*)

The Multi-Robot-Simulation-Framework (*MuRoSimF*) is a realtime simulator replacing a real robot for software-in-the-loop (SIL) tests [14]. It provides the simulation of the humanoid robot kinematics and dynamics, interaction with the environment and simulation of the sensors. Due to the available interfaces between the simulator and the robot control software, the real robot is fully replaced by a virtual robot which is illustrated in figure 3.3. *MuRoSimF* supports the unique feature of different levels of abstraction. For each robot the complexity and level of detail of motion and sensor simulation can be chosen individual. For this reason the computational resources for the simulator can be tailored individually for a specific SIL test.

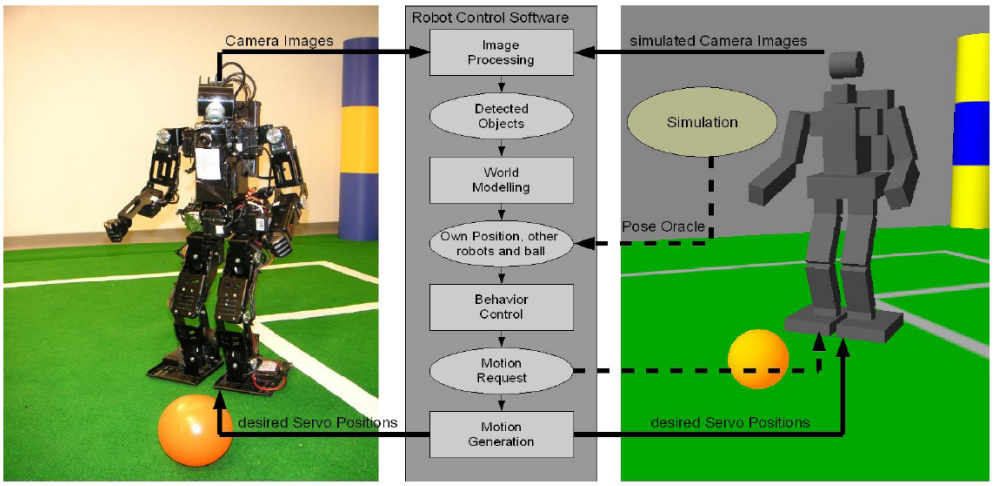


Figure 3.3: Data flow and interfaces of simulator [14].

During development the *MuRoSimF* simulator is used for testing and debugging. Only when the software has been sufficiently tested, test on the real robot are performed. This approach speeds up the development process and prevents ware on the real robot hardware.

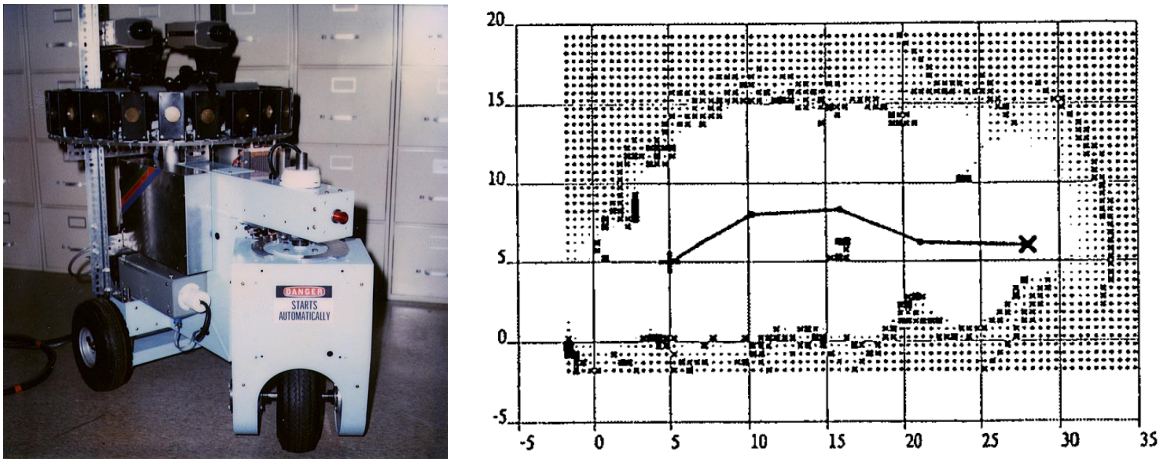
4 Related Work

In this chapter a short outline of historical development of research of occupancy grid maps and active gaze control is given. The presented references give an overview of already implemented approaches whose results are considered in this work. Furthermore, the current state of the Darmstadt Dribblers is presented and analyzed for issues which are considered in this work.

4.1 Occupancy Grid Maps

In [26], Elfes and Moravec presented the first use of occupancy grid maps, named evidence map in that context. In the following years they extended the approach in [8], [9] using sonar and experiments with the robot Neptune (figure 4.1a). The approach generates two intermediate maps which are integrated to build a final map. While one map models the free space, the other contains the surface hypotheses. The sensor model based on a two-dimensional Gaussian uses a binary interpretation to update the cell states, being either in free space or in a surface (occupied). Finally, the data of both maps is integrated using a heuristic technique. The approach assumes that all sonar sensor returns are caused by simple reflection. All geometric uncertainty like multiple reflections on surfaces are ignored. Considering free and occupied probability in separate maps is adapted for occupancy grid mapping. But in this work these probability are modeled as counters, counting how often a cell was observed as free and occupied.

Elfes reformulated the occupancy grid approach to a Bayesian updating problem in [11], [12]. The sensor readings are modeled with stochastic sensor models using Gaussian distribution as noise for sensor readings. The map is also updated via Bayesian methods. But there are still some issues in using Bayesian update in this way. Firstly, a single sensor reading can change the occupancy value of a cell drastically, so fluctuation in cell's value are possible. Secondly when a cell convergences once to a value of 0 or 1, it never can be changed anymore. Despite these shortcomings, this work provided the probabilistic framework for occupancy grid maps which is currently widely used for navigation, localization, path planning or collision avoiding purposes. An example of such a grid map is shown in figure 4.1b. In this work a sensor model is introduced using Gaussian distribution as uncertainty for observation. But cell state is not updated with Bayes update, resolving the cell convergence. Here a cell state update rule for camera observation is introduced. Additionally, modeled data is updated into the map using Bayesian updates.



(a) Neptune (1984) **(b)** Grid map generated by Neptune moving in the laboratory.

Figure 4.1: Alberto Elfes' robot using occupancy grid mapping [26], [8], [9].

A long time, it was a challenging to provide occupancy grid maps in dynamic environments. Even Thrun wrote in 1998: "It is an open question as to how to incorporate models of moving objects into a grid-based representation." [37]. But in the last years it could be shown that occupancy grid maps can also be used in dynamic environments with moving obstacles [6], [7], [18]. The investigations in dynamic grid mapping can be split up in two main groups. One group discusses how to extract a static map of the robot's environment despite moving obstacles disturbing sensor measurements of the surrounding world. In other approaches occupancy grid maps are used to detect dynamic moving obstacles for a collision-free path planing. For this purpose occupancy grid map techniques are used to estimate which cells contain dynamic obstacles whose movement has to be predicted. In the context of the RoboCup soccer scenario the environment is already known, so the dynamic elements can be easily separated from static environment. Thus such technique must not be considered in this work.

In [28] a trinocular stereo camera system is used to detect obstacle which are mapped into an occupancy grid map. The grid cells are updated by incrementing or decrementing the cell value in dependence of the located obstacles in the camera's field of view. The resulting map is used for navigation, path-planning and exploration purposes.

In [18] a probabilistic map based on Bayesian model for objects is introduced. In addition the system saves date of recognized objects to update their state if they are detected later. If an object just has been observed by a sensor the position information is directly added to the map. They assume in their approach that there are no static objects. For this reason all data of objects are predicted if they cannot be detected. Their movements are predicted in dependence of the probabilistic model which increases the occupancy probability of the surrounding area. This approach introduced a good method to model the uncertainty of objects whose tracking is lost.

A new approach called Bayesian occupancy filtering based on a 4-dimensional probabilistic grid representation of the obstacle state space is presented in [6]. The goal of this approach is to build an alternative to complex multi-target tracking algorithms, because traditional approaches fail on high dynamic environments with temporary occlusions. To avoid this problem, explicit modeling of uncertainty is used in the probabilistic grid map representation. Furthermore instead of tracking of objects this approach reasons directly on the probabilistic grid map using a Bayes filter. For this purpose a novel two-step estimation mechanism is introduced using sensor observations history and the temporal consistency of the scene. The probabilistic grid map is updated at each time step by combining a prediction step obtained from history and an estimation step using the new measurements. This method is derived with Bayes filter introduced in [17]. The presented approach is validated successfully on an experimental vehicle for avoiding partially observed moving objects.

Interestingly, there are only a few approaches such as [6], [18] which are able to estimate the occupancy of cells of unseen obstacles and simultaneous restore stepwise the probability value of a cell to an initial value (e.g. 0.5), if there is no observation of this region available for a long time. But just this seems to be a good approach to model moving obstacles on humanoid robots with a limited field of view. Restoring the occupancy probability to an initial value is equivalent to increase the uncertainty of a region of cells and can be compared with aging an occupancy grid map or just forgetting. Exactly this feature can be exploited for an active gaze control which goal is to reduce the uncertainty of the occupancy grid map.

4.1.1 Fusion of Occupancy Grid Maps

Sensor fusion is an important feature for multi-sensor systems. Occupancy grid maps lend themselves well for this task and this property is often used in the scientific community. In this context many results were obtained in researching multi-exploration and mapping.

The first approach for multi-sensor systems using several maps is suggested by Moravec [27]. He uses a certainty grid representation which is incrementally updated by sonar, stereo vision, proximity

and contact sensors fused by probabilistic methods. The approach of integration of several maps from multiple robots presented in [5] is also based on Moravec's approach.

In [36], data from a monocular camera and LIDAR is merged into an occupancy grid map. For self-localization a landmark-based Markov localization approach is used. In the first step the camera image is transformed into a rangefinder measurement based on predefined color representing free space in the image. So the resulting measurement is similar to a LIDAR measurement. From each sensor types a occupancy grid map is learned which are merged via Bayesian methods. This approach respects even the different accuracy of the sensor types which increase additionally the quality of the final occupancy grid map. Finally a harmonic potential field method based on this map is used as path-planner. Here, an new evaluation method of occupancy grid maps is introduced. It uses a measure of safety which is defined as the shortest distance from the path to all obstacles. This approach allows evaluation the practical utility of the fusion methods and the resulting map for path-planning. In this work a similar fusion method is presented to learn occupancy grid maps.

In [19] an efficient hierarchical sensor fusion method is introduced. Here, a complex sensor system is used which consists of a ring of ultrasonic sensors, a ring of infrared sensors and a camera system using a halogen lamp for shape detection of obstacles in the environment. The hierarchical sensor fusion method consists of three levels: level 1 merges ultrasonic and infrared sensor measurement, level 2 combines a halogen lamp based active vision sensor and a subset of ultrasonic and infrared sensors. Finally level 3 fuses the both other levels to an occupancy map. Level 1 and level 2 merging is based on sensor models but level 3 uses Bayesian methods for sensor fusion. Evaluation result shows this approach increases accuracy of the occupancy grid map.

A framework for three merging approaches is formulated in [16]: Fuzzy, Dempster and Bayesian approach. An A* path-planning algorithm is used to compare the resulting occupancy maps of all approaches with each other. The result of the evaluation shows that the map created by the Bayesian method is more accurate than the other two approaches and yields the best planned paths.

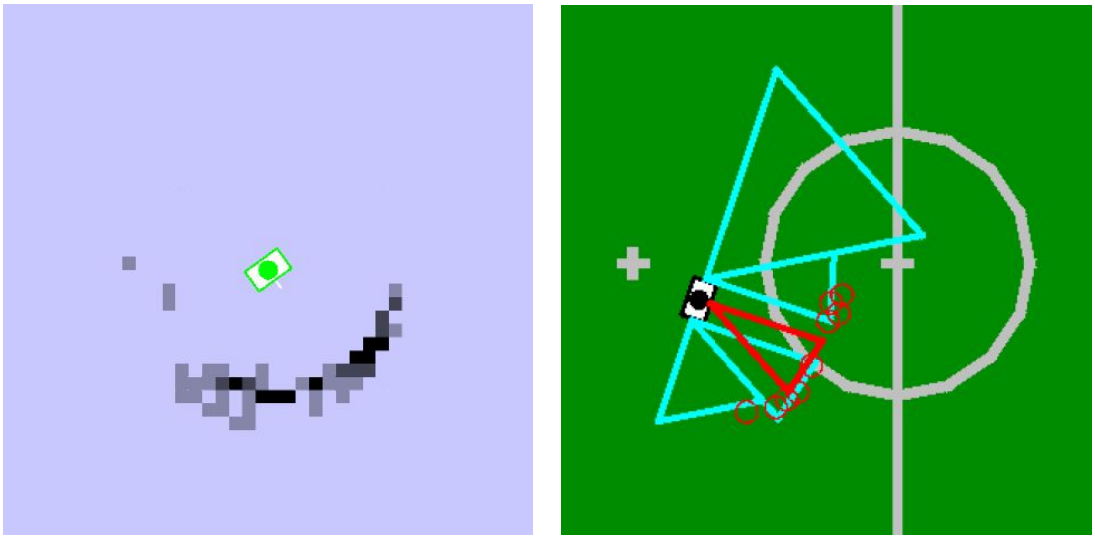
4.1.2 Occupancy Grid map used by RoboCup Teams

Based on the team description papers and reports of the participants the main research topic in the legged RoboCup leagues is located in kinematics, locomotion, localization, navigation and behavior control. For this reason there are only a few approaches for explicit environment modeling. The usual simple obstacle avoiding approaches use object tracking methods, estimating their position. In addition explicit ball, goal and robot modeling is often denoted as world model and used for path-planning. It can hence be concluded that occupancy grid mapping play a minor role in the legged RoboCup leagues. The robots in the humanoid Standard Platform League are based on the Nao[1] robot, where some approaches are already using occupancy grid map which are presented in the following.

In [31] a grid based modeling of the sonar measurements storing the number of positive obstacle measurements is used. For every measurement the counter of every cell with the corresponding distance in the sonar cone is increased while all cell values within the measured distance are decreased (figure 4.2a). The space in front of the robot is split up into four sectors. So the final obstacle representation contains the minimal distance to an obstacle of each sector which is extracted from grid map. The resulting obstacle model corresponds to a triangle for each sector (figure 4.2b).

A dynamic grid mapping approach is presented in the team description paper [7]. They use a local polar grid map representing the environment relative to the robot's position. The cells are initialized with the value 0.5 which indicates the absence of knowledge. All cells within the sonar cone are updated by a given measurement. The cells' value is increased at the distance of the obstacle and is decreased within the obstacle to model free space. To model the uncertainty in the dynamic environment the cell's value is aged by updating stepwise to the initial value of 0.5, if there is no observation available for the cell. But anyway there's no explicit obstacle modeling used.

[1] Nao is a commercial robot produced by Aldebaran Robotics, URL: `http://www.aldebaran-robotics.com/en/Nao.php`



(a) The grid map built on sonar measurements. The counter is represented in the cells' color depth.

(b) The obstacle model based on grid map in (a).

Figure 4.2: Grid based obstacle modeling of the "B-Human" team [31].

4.2 Current Obstacle Modeling of Darmstadt Dribblers

The obstacle approach used by the Darmstadt Dribblers was developed in context of a diploma thesis [24]. In this work it is demonstrates how to extract percept information in a camera scene for obstacles modeling.

In the first step a color segmentation algorithm is performed on the grabbed camera image. Then bottom-up scanlines are used to detect obstacles. For percept detection the colors of the soccer field (white and green) are defined as free space and black is defined as occupied. The position of a detected obstacle can be transformed with a camera transformation matrix into robot relative coordinates.

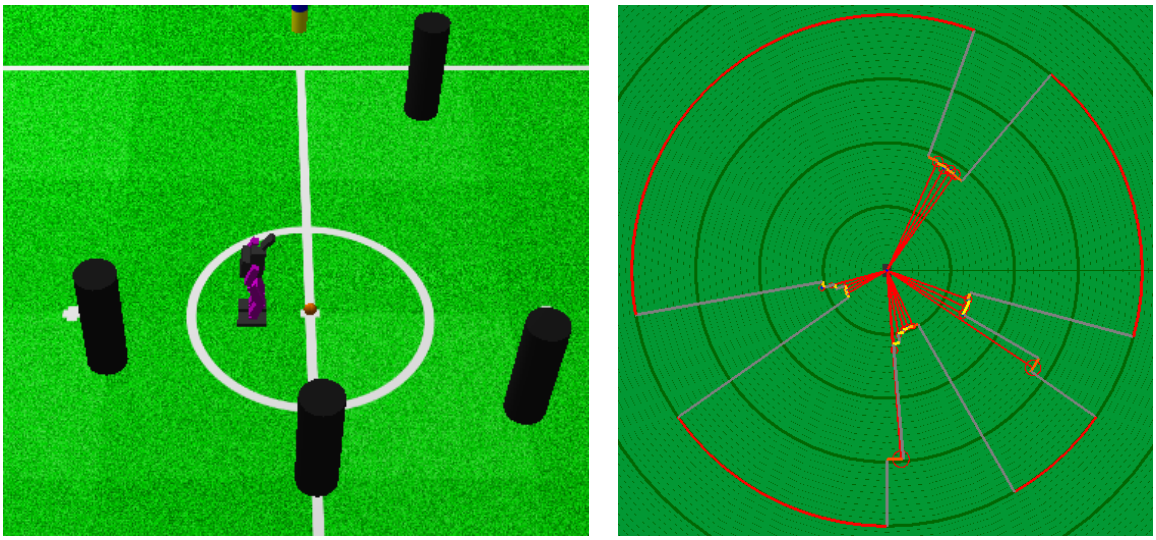
A sector model based on this data which represents the robot's surrounding environment in robot coordinates is built. The surrounding robot world is split into several sectors which is similar to a LIDAR measurement (see figure 4.3). Every sector holds three values: distance, confidence and a point. The distance describes the space to the next obstacle being located within the sector while the confidence represents the trust in the informations about this sector. The confidence depends directly on the age of the represented precept. Additionally a point representing the percept coordinate is needed for sector model transformation purposes, because the robot's position usually changes and with it the relative positions of obstacles.

Updating the sector model consists of three phases:

1. aging of the model
2. model transformation according to odometric data
3. model updating by new obstacle percepts

The aging of the model describes the decreasing of sector's reliability and is applied in every iteration. Sectors within the field of view of the camera are aged five times faster before they are updated by observation. This accelerated aging is needed, because there's no explicit modeling of free space in the sector model. If the sector's reliability reached zero, the percept is deleted and the sector is defined as unoccupied. The other update phase is only executed when new information is available. Further details of obstacle percept generation and the sector model can be obtained from [24].

The sector model is currently used for the path-planner based on a potential field method. For this purpose, a list of obstacle coordinates is generated from sector model which are added as repulsive



(a) The given scenario. **(b)** The resulting sector model.

Figure 4.3: Sector model for obstacle modeling used by the Darmstadt Dribbler team: On the left side the scenario is shown which is modeled by the sector model on the right picture.

potential to the field. This approach is independent of self-localization estimation, because the sector model uses robot centric representations of obstacles.

While simple and computationally efficient, there are some drawbacks to this approach. Due to the missing reasoning about free space every movement based on the sector model is acting on the assumption that a sector classified as unoccupied indeed contains no obstacles. In fact it is also possible that a sector is just not observed for which reason collision with unseen obstacles are possible.

Robot state changes have to be considered in the sector model. For this purpose, the sector model has to be updated with a transformation given by robot movement, usually odometry. Maintaining the consistency of the sector model is a difficult problem. Because of the discretization of the environment into sectors, the shape of obstacles cannot be determined. Accurate transformation of obstacle data is thus not possible and has to be approximated. Exchange of sector models between robots is additionally a difficult problem, as transformation of sector models into the coordinate systems of different robots is not possible in a lossless and consistent fashion.

Another issue is given by the fact that uncertainty in observations currently is not modeled in the sector model. It provides no probabilistic modeling of neighboring sectors and no measurement filtering. For this reason it is susceptible to measurement errors of obstacle percepts. There are two main causes of failures:

1. Projection errors in object's base estimation
2. Position deviation caused by fast camera head movements

The first error produces outliers in the sector model which can be seen clearly in figure 4.3. When the camera head pans fast, due to timesstamps of camera and read out joint values not being perfectly synchronized, the deviation in pan angle increases noticeable, so the estimation of real object's position is imprecise. In worst case scenario, single small objects can produce estimated walls in the sector model. While the approach is used successfully in practice, there is room for improvement with regards to the stated problems.

In contrast occupancy grid maps have a global fixed representation size of the environment in form of grid cells. Using the soccer field as coordinate system enables even to get rid of map transformation calculations for every movement of the robot. Therefore, a good estimation of robot pose is needed to integrate correctly new measurements. If the self-localization problem is solved reliably, a consistent

environment model can be created which can be transferred to other robots without any transformation. Because of the model's fixed resolution, it is possible to integrate consistently small obstacles into the grid map which are still far away. Here, it is assumed that these properties enables better and more reliable path-planning realized using the potential field approach.

For these reasons the sector model is replaced with an occupancy grid map representation introduced in this work. More details are given in chapter 6.

4.3 Active Gaze Control

Active gaze control also called intelligent perception was researched by Elfes in [10]. Here stochastic sensor models are used to determine useful sensory actions and compute local extrema of relevant observations. These models allows to develop perception control strategies on mobile multi-sensor robots.

In [38] an biological inspired approach for active vision is introduced. Here the "perceptual flow" which is also referred as "stimulus dynamics" is used as input for a saliency map. Using this map, a "winner takes all" concept is realized by an artificial neural network deciding which "stimulus dynamic" wins. Afterwards the activation dynamics determines which targets should be given attention. In this work an technique for Inhibition of Return (IOR) is presented. IOR is a problem in gaze control system because they have the tendency not returning the attention to already seen objects. For this purpose a large negative activation value is given to the position in the saliency map, after giving attention to a target.

Another saliency based approach is presented in [39]. Here saliency features are extracted by classifying changes of low level properties such as color, illumination or orientation in pixel. All found salient features are modeled by probability distributions. Afterwards the Kullback-Leibler divergence is used to compute the differences of probability distributions to obtain a "surprise map". The sensor action is driven by the most surprising feature in the map. Evaluation of the approach show promising results, but realtime operation can currently only be achieved by using GPU implementations. This is a typically problem of salient-feature-based gaze control.

Salient-feature-based approaches are reactive methods whose advantages is combined with proactive methods in [32]. For this purpose object information is rated with confidence value within $[0;1]$. When an object is updated by a percept its confidence value is set to 1. Afterwards this value is stepwise decreased when the state of the object has to be predicted, because of missing perceptual information. Additional the need of each object for the actual task is considered which value is defined from behavior layer. Finally the importance of each object α is calculated with:

$$importance(\alpha) = need(\alpha) \cdot (1 - confidence(\alpha)). \tag{4.1}$$

At the end, the object with the highest importance is given attention. This approach will be taken up later again in this work.

A novel entropy-based approach is introduced in [33]. This approach does not use probabilistic models such as Kalman-Filters or Markov Models. Instead the incertitude of each object is used as measure. Furthermore the Shannon Information Theory is respected by considering the distance and focusing of each object for best gaze. Given these considerations, the information content of each object is calculated. The existing knowledge about the environment is used to approximate the best gaze. For this purpose the sum over all information content of the objects has to be maximized for the next gaze. The evaluation results are satisfactory and shows that a non-probabilistic approach is able to provide effective gaze control. In this work a similar approach is used.

4.3.1 Active Gaze Control used by RoboCup Teams

Due to the high computational demands, active gaze control is not used by many RoboCup teams. As already mentioned in chapter 4.1.2 the most investigation of the teams is focused on kinematics, loco-

motion, localization, navigation and behavior control. Many teams use still passive gaze control, but there are also a few teams using active approaches.

In traditional passive gaze control systems the head movement behavior cannot be influenced by incoming sensor measurements. The head control control system does not consider the information gain of incoming sensor data which can lead to poor results. A frequent solution is to predefine the trajectory of head movements by experimentation. Using this approach systems can be realized easily and need less computational resources than an active gaze control.

A common simple solution for active gaze control is based on object tracking. For every tracked object a position can be estimated and even predicted. Hence many RoboCup teams use the estimated or predicted position of an object which shall be updated by gaze control. The gaze control uses this position to calculate a trajectory for the camera head, so the next camera images will hopefully contain the desired object in the environment [32].

Some teams are even able to reason directly from camera images. They extract salient features of a grabbed image e.g. in form of regions or even pixels. Due to this extracted salient features an active vision system is driven to focus the most salient region under the assumption of getting new informations or just being able to update old information [2].

In [3] entropy-based active vision approach is presented. One of the main constraint of the active vision system is the possibility of real-time computation. This system uses a particle filter for self-localization and ball state estimation. The belief of all states is represented by a set of samples which approximates a density function over all particles. Because the entropy cannot be calculated directly over particles, a simple histogram-based entropy estimation is used, approximating the belief in a grid-based representation. This builds the base for estimation of all possible entropies produced by every head moving. Of course the head moving is chosen which promises to gaze the region with minimal entropy and costs. For this purpose a Monte Carlo Exploration is used to estimate the best observation. In this case an observation is defined as raw measurements called features, extracted from camera image. Because the prediction of every possible observation is too expensive, for each feature a sensor model is used which is learned by a feed-forward neural network with backpropagation. The entropy is predicted only for one step, because planning the camera control for more than one single step would increase computational costs exponentially. Additionally single step planning is more reactive towards dynamic environments. Furthermore they used following computational optimization:

- All hypothetic particle weightings are approximated by a preprocessed table.
- The logarithm function used by entropy calculation is sped up via offline computed table.
- Only the most significant samples are used as robot hypotheses for entropy calculation.

With the help of these optimizations the approach is enabled for real-time active vision controlling. The evaluation shows that the active vision system decreases the error of localization by 20% and the error of ball model even by 44%. Thus this paper gives an idea how active vision system can increase system performance in comparison with passive vision systems.

The RoboCup team "Nao Devils Dortmund" presents a similar approach for real-time active vision computation [35] based on Thrun's work [13]. The localization task is also solved by particle filters, so a grid discretization of environment is used. Active vision is used to minimize the estimated entropy for an action reducing the error of self-localization. For this reason the calculation of the entropy depends only on the localization task. Because of limited computational resources, particles are clustered to approximate the density distribution representing the current belief. Furthermore, only the possible observations given by particle set or the chosen position discretization are predicted, so the computation can be accelerated. In this work is also suggested to use pre-computed lookup tables for all possible observation in dependence of the robot's state. Analogously the measurement probabilities can be added to this lookup table. As the table grows exponentially for useful resolutions, it is not a practicable solution for limited hardware. The evaluation of the approach was done in a 3D Simulator, in order to obtain the ground truth and repeatable tests. It can be shown that the approach improves self-localization of the

robot. For future development is planed to integrate dynamic features for instance the ball in the active vision system.

There are no further known projects working on active perception systems and even less nobody uses occupancy grid maps for entropy-based gaze control. Consequently this approach of active gaze control seems to be unique in the RoboCup community.

4.4 Current Gaze Control of Darmstadt Dribblers

In the current state of work the humanoid robot's gaze control is based on behavior control using preprogrammed camera head motions which are created by experimentation. The behavior control is realized with XABSL, a behavior description language using finite state machines. Further information about XABSL can be found in [23].

In dependence of the robot's state and task the behavior control chooses one of the following camera head motions:

- **look_straight**: Let the robot look straight forward.
- **look_lateral**: Let the robot look to the side (see figure 4.5).
- **look_around**: Exploration of the environment (see figure 4.4).
- **look_in_front_of**: Looks in front of an object, so it will appear in the lower part of the image.
- **look_at**: Look to a specific point in robot coordinates.
- **look_for_ball**: Search for the ball.

To prevent staring e.g. at the ball a timer is used in behavior control which direct temporary the view to another point. This step should prevent an aging out of important information needed by localization and path-planning. But the current approach does not use any knowledge such as confidence of an object modeling and just assumes that new observations can be obtained with the short camera moving sequence. There are only a few cases, where the camera head movement pattern is changed in dependence of existing modeling. If for instance the validity of ball modeling decreases, due to losing the tracking of the ball, the behavior control will change its state, so another camera head moving is used.

The usual semi-active head control behavior is tracking the ball. In this state two different ball tracking modes are available: one for tracking a close ball and another for tracking a distant ball. For both, the camera head trajectories are created by experimentation. Both modes are created to prevent staring only at the ball. Thus this approach is a semi-active gaze control system.

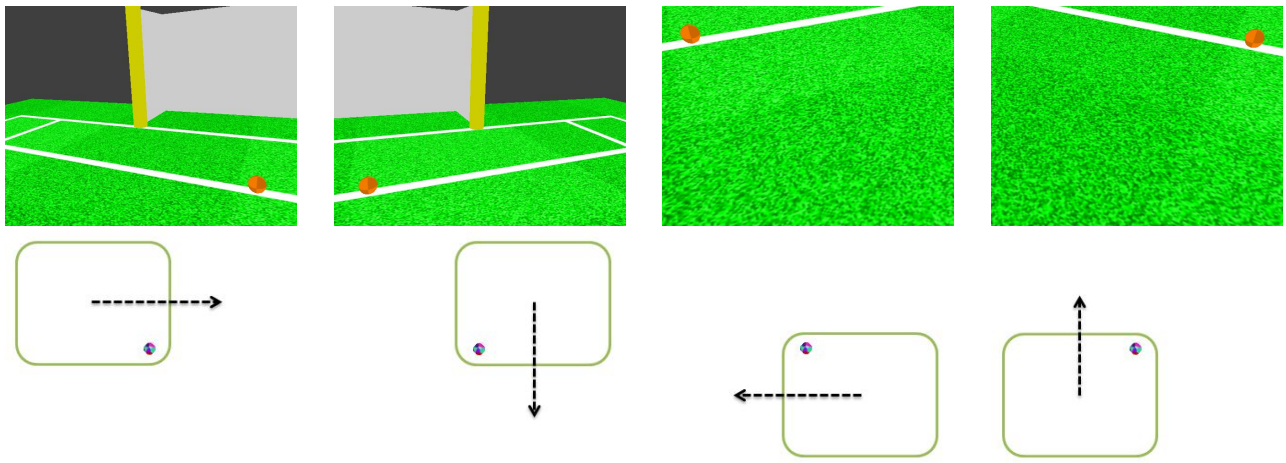


Figure 4.4: Look around head movement cycle when ball is distant. The first line shows the images grabbed from camera while the second line illustration shows the view relative to the ball. The arrow indicates the next movement step.

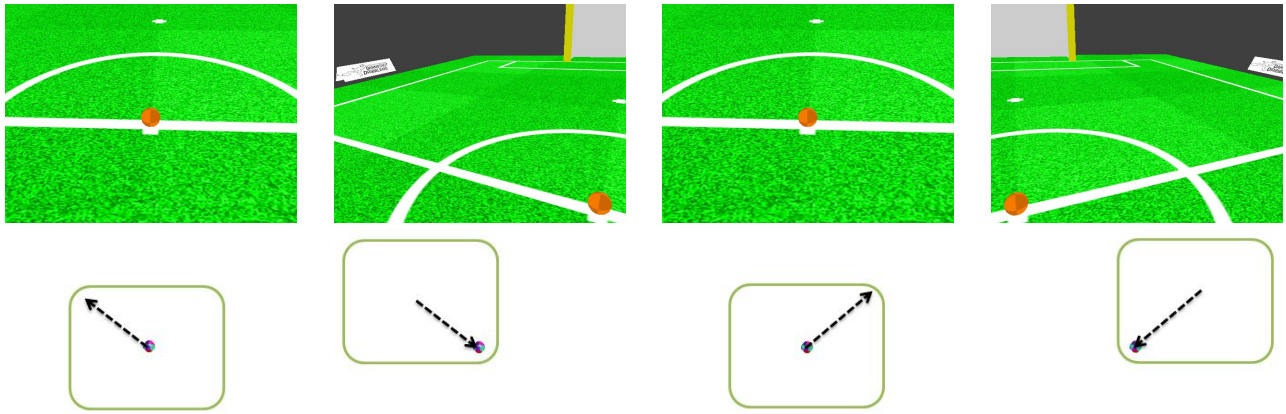


Figure 4.5: Lateral look-up head movement cycle when ball is close. The first line shows the images grabbed from camera while the second line illustration shows the view relative to the ball. The arrow indicates the next movement step.

In figure 4.4 the full sequence of head moving is shown, for the case the ball is distant. The head is turning around a rectangular path in a way that the ball is always seen at the corner of the image. So after every movement step the ball can be seen in another corner of the image. Furthermore an optimization within the tilt axis is already included for head movings. Assuming nothing is flying and larger than a goal the tilt axis control is limited by the goal's crossbar. The maximal tilt can be computed by robot's pose and a pre-known model of the goal, so the head is only moved upwards until at most the latch can be seen. This optimization reduces camera movements and decreases blur effects produced by them.

If the ball is close to the robot, another head moving sequence is used which is illustrated in figure 4.5. This head control behavior executes periodically lateral upward movements in alternating directions. Having looked to the side for a short time, the head control turns the head back to focus the ball. The trajectory of lateral movements is fixed, so moved distance in tilt and pan are constant. For this reason it is also possible that the ball won't be seen while the head control behavior executes a lateral look.

This solution works in practice but also has some drawbacks. As mentioned in chapter 2.5 there are worst case scenario, where this approach fails. A typical scenario occurs when the robot approaches the ball. The robot approaches the ball in a curved trajectory, so the ball can be kicked or dribbled directly towards to the goal. While moving in a curve trajectory the robot keeps tracking the ball, so possible

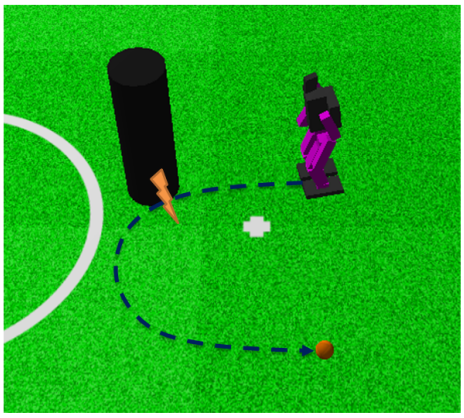


Figure 4.6: Curve trajectory with possible collision.

obstacles in front of the robot are potentially seen. This scenario is illustrated in figure 4.6. All mentioned issues can be solved by using information about the uncertainty of the environment knowledge. Hence the goal of the new gaze control is improving the camera head control in a way that the modeling and with it the behavior control receive enough information to solve the current task (often scoring).

For this reason a new active, goal-oriented gaze control, improving perceptual input, is presented in this work. This approach is used by head behavior to generate commands, where to look next. These commands are based on the learned grid map and models e.g. the ball model. Further details can be found in chapter 6.

5 Concepts

5.1 Design Considerations

The main goal is to improve the world modeling system of the humanoid soccer playing robots, especially with regards to obstacle avoiding and active intelligent gaze control. In contrast to the sector-based approach used so far, the grid-based map is able to additionally model free space and uncertainty. For this reason grid maps will enable more complex path planning and behavior.

In recent years many technical advancements allow to use powerful processors on robots with payload restrictions. These extended computational resources allow online computation of approaches in realtime that were not feasible before. This approach should use less resources than possible in order to leave room for other developments.

In contrast to the SLAM problem, the soccer scenario has previously known static features. Here, the self-localization is solved by mapping the observation of known poses. In this work just these additional informations are used to create more accurate grid maps. Later they are also used to obtain a basic entropy for active gaze control.

The second goal of this work is to realize a low-cost entropy-based active gaze control which can be computed online on robot's hardware. All entropy-based approaches have in common that their computational costs are too high without using approximations. In this work it is thus required to reduce the computational cost to a minimum for entropy prediction. A lot of information can already be gathered while grid map generation, so computational costs can be saved already at this point. The result of this entropy-based gaze control is used by head behavior generating new head motion commands.

All implemented source code of this work is integrated into the framework of the Darmstadt Dribblers and is runable on the current humanoid robot model DD2010.

5.1.1 XABSL

XABSL is short for *Extensible Agent Behavior Specification Language*, introduced in [23]. It is a simple programming language for behavior control of autonomous robots. Behavior modeling in XABSL is based on hierarchical finite state machines. XABSL has been adopted by multiple teams in RoboCup among them multiple winners of different leagues like the German Team in the Four Legged League and the Darmstadt Dribblers in the Kid Size League. For seamless integration into the existing system, the active gaze control approach developed in this work can be used by XABSL as head movement behavior.

5.1.2 RoboFrame and RoboGUI

The Darmstadt Dribblers successful use RoboFrame developed at TU Darmstadt [29] since several years [25]. RoboFrame is a platform independent software framework for teams of heterogeneous robots. Application using RoboFrame can be run using the common operating systems like Linux, Windows or Mac in 32 and 64 Bit varieties. Several platforms are used by the Darmstadt Dribblers members for development, proving true platform independence.

The framework uses C++ and modern object oriented design techniques. For the Darmstadt Dribblers team, image processing, motion and especially behavior control in form of XABSL [30] are already implemented. Because of transparent communication mechanism for data exchange, modules can run in different threads or even on different CPUs and data can be exchanged efficiently via message buffers.

RoboGUI is a graphical user interface (GUI) for RoboFrame applications. It provides a simple framework allowing developer the creation of new dialogs and visualizations without any knowledge about

the underlying communication mechanism. Furthermore, logging and connectivity functionality as well as several debug dialogs are available which help to check the validity of models, debug the code and assist in monitoring.

5.2 Obstacle Percepts

In this work the most important data source are the perceptions of obstacles. These are generated by the obstacle perceptor which uses the grabbed camera images to generate obstacle percepts [24]. For this purpose, the image is sampled with vertical and horizontal scanlines. Along these scanlinesl pixels are classified to belong to different color classes using a colortable.

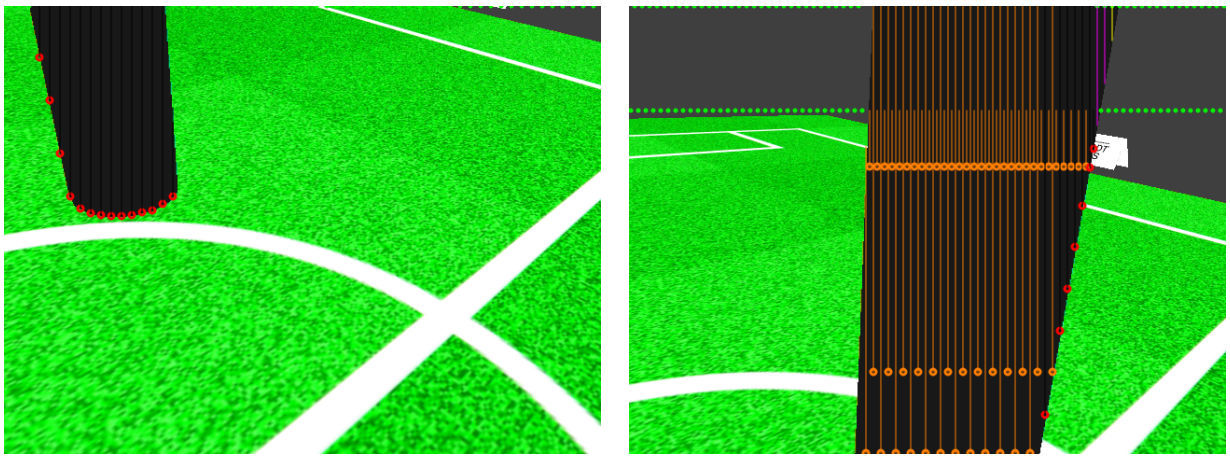
For obstacle segment detection only the vertical scanlines are used. Along such a vertical scanline the obstacle segments are searched starting at the bottom of the image. For every scanline an percept is generated which can be classified in three categories:

- **red:** obstacle
- **orange:** obstacle, but is partially visible because base point can not be determined in the image
- **green:** free

When an obstacle segment is detected, the correspondent point in the percept is set as occupied. Otherwise the search ends at the upper border of the image no obstacle is detected and the percept is set as free. Afterwards the classified obstacle percepts are projected to the ground into robot relative coordinates. The obstacle percepts have the property of a LIDAR scan and thus can be treated as LIDAR measurement.

The obstacle percept model is a simple and efficient technique but can generate faulty data in some situations. When an object appears tilted in an image the scanline technique detects the object's side border as base points and thus generates there phantom obstacle which is illustrated in figure 5.1a. Additional these so called outliers are projected on the ground far away from real object's base point. To achieve the best possible performance, these artifacts are considered in this work.

A worst case for obstacle perceptions occurs when the base points of objects are not in the camera image. This often happens if obstacles are close to the robot. Then, the base point cannot be projected and no distance information can be provided for the obstacle percept. An example of this situation can be seen in figure 5.1b.



(a) Wrong obstacle percept generation on the left border caused by aslope object.

(b) Worst case situation when object is only partially visible.

Figure 5.1: Demonstration of obstacle percept model's main issues and worst case situation.

5.3 Occupancy Grid Mapping and Active Gaze Control

In this work two problems are tackled. The first problem is to create an occupancy grid map while the second considers how to control camera head movements to obtain as much as possible informations about the environment which are in turn used for occupancy grid map generation. As the environment is a predefined soccer field (see Section 2.2), self-localization, while still challenging due to large uncertainties, is becoming more reliable in recent years due to leveraging robust probabilistic techniques. For this reason the generation of grid map is independent form self-localization task and can be focused in this work.

Furthermore, the RoboCup scenario is different to many other scenarios where occupancy grid maps are used. As self-localization is performed without relying on the grid map in contrast to SLAM scenarios, here the occupancy grid map is supposed to be only used for obstacle avoiding and global path planning. The soccer scenario gives the robot a perfect ground truth for static obstacles and does not have to be explored anymore. For this reason, the occupancy grid map update does not have to be able to differentiate all obtained measurements if they are caused by static or dynamic objects.

As data about static obstacles and data from processed models is available, so it is reasonable to use this for map building, because they contain many useful informations about the robot's environment. For this reason a container class is introduced whose instances are collected in lists. In this way an interface is built allowing general access to this data for map building purposes. Due to this data is already modeled no further modeling is used such as filtering or data approximation. Furthermore the soccer field is predefined, so it is used as ground truth for a static model here. In figure 5.2 the conceptual data flow is illustrated, where the preprocessed and predefined data is represented as static and dynamic elements which are stepwise merged into the final occupancy grid map. In [19], [36] is shown that stepwise merging of different data resources is more robust.

The obstacle percepts can also be used for occupancy grid map updating. Therefore a camera model is needed which enables to aggregated the obstacle percepts into an obstacle map. In [36] learning of occupancy grid maps based on monocular camera is successfully demonstrated. Most approaches deal only with robot systems which are especially developed for solving the SLAM problem or just generating occupancy grid maps. So using preprocessed information in combination with a sensor measurement is a novel approach because this type of information is rarely available.

Finally, all data has to be merged into a single map. In figure 5.2 it is shown that map merging functionality is needed in seven cases. Obviously for better code reusing an map merging interface is needed to provide generic and flexible data aggregation. In [16] it is shown that the Bayesian update rule is the most reliable approach in occupancy map generation. Furthermore in [36] and [18] Bayesian update rule is already applied with very promising results. For this reason, the Bayesian update equation in Section 2.4.1 is used as basic aggregation method here (see Section 6.1.2).

After an occupancy grid map was successful obtained, it can be used to find the gaze direction for which the highest information gain can be expected, corresponding to reduction of uncertainty. For this purpose, the learned obstacle map and the available models are used to approximate the expected entropy for a given gaze direction which is described in Section 6.3. Entropy-based active gaze control is an optimization problem which tries to find the best gaze direction with given constraints.

In this work, these constraints are selected considering of Shannon's Information Theory [34]. In practice, the following issues have to be solved by an active gaze control system [4]:

1. **Partial vision:** Image processing modules perform unreliable, when objects are only partially visible. The best example for such an issue is given by the obstacles perceptor in Section 5.2. For this reason it is desirable to focus objects preventing partial vision.
2. **Inhibition of Return (IOR):** Active gaze approaches have all to enforce the problem of returning the visual attention to already seen objects, because another object provides a high stimuli. The implementation of an active vision systems supporting IOR is just a challenging task.

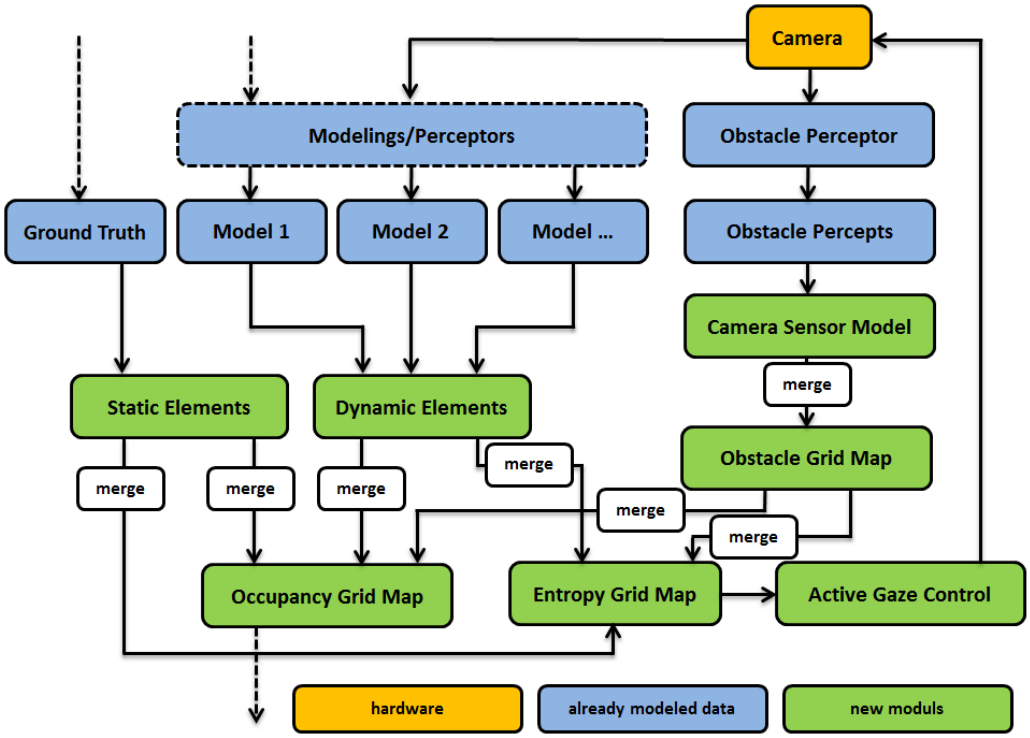


Figure 5.2: Data flow of concept.

3. **Decision uncertainty:** There are a various number of real-world implementation issues such as camera blurring and distortion, non-constant lighting conditions and disturbance in camera head joint position estimation. These factors affect the perceptual uncertainty of the system and also have to be considered for gaze control. Furthermore, close objects provide more information, because sensor measurement reliability often decreases with distance to the object.

Due to the nature of the problem the most promising approach of these mentioned in Section 2.5 is obviously an entropy-based approach, because such approaches consider uncertainty in state estimation. The generated occupancy grid map can easily be used in such an approach. Due to modeled data of other cognition modules is available, their information confidence can be used to approximate an entropy for each object. In [33] such an object-based gaze control is already presented. In this approach the "total information content" is computed which is used to approximate the best gaze direction. Furthermore, the distance and centering of focused objects are considered. All these ideas are taken up in this work.

The hardware on mobile humanoid robots is limited in size and weight and with it the computational resources. The resources consumed by the gaze control system have to be strictly watched and limited. Other teams ([3], [35]) participating in the RoboCup competition are also considering this issue. In [3] it is also considered that the generation of a sequence of camera movements produce a higher information gain but also is more expensive to compute. For this reason in this work only the immediate next gaze direction is calculated which is assumed to be more reactive. A reactive gaze control is preferred, because the environment changes dynamically. Due to the availability of preprocessed data and the hardware limitations, the usage of a probabilistic approach is not feasible. Besides the work [33] no further approaches are known using non-probabilistic methods. In this work information gain is sampled by so-called particles, presenting a novel feasible approach for resource-constrained systems.

In [32] an effective method is presented which uses priority and simultaneously the confidence of object estimation to decide where to look next. This approach is also adapted here, enabling goal-oriented prioritization of objects. This feature enables a simple method to support IOR on modeled objects through decreasing object's priority.

6 Implementation

In this chapter the developed approach and implemented techniques are described.

6.1 Obstacle Grid Map

This section describes the used and developed basics of this work which are needed to learn and maintain an obstacle grid map. All calculations are executed by *ObstacleMapModeling* while all data is held in *ObstacleMapModel*.

6.1.1 Map Modeling

The grid map implementation is adapted from earlier work [20]. Here, the basic grid map is provided by the class *GridMapBase* allocating a 2-dimensional array of cells. The cell type is specified by a C++ templates which can be changed during compile time. Using templates has the advantage that the code is inlined by the compiler, improving execution time. In this work different cell types are developed, described in Section 6.1.2. The information itself is stored and managed by the cells. Usually, the cell value is only calculated by an update and is afterwards cached. For this reason accessing the map is much cheaper than updating.

Each cell is assigned a specific region of the world. Every point given in world coordinates can be transformed by the transformation matrix ${}^{m}T_{w}$ into map coordinates. The map coordinates are given as integer, so the inverse transformation is ambiguous. For this reason all transformations from map into world uses the cell's center point as coordinate which is transformed back into world coordinates.

Furthermore the class *EntropyGridMap* inherits *GridMapBase* and provides extended features for active gaze control. This includes maximum searching approach based on particles representing local maxima.

6.1.2 Cell Modeling

As already mentioned the map's cell type can be chosen using template. The cell type itself must provide certain methods (see appendix 9.1). Among these methods there is the cell's value handling which includes the cell's update rules. For this reason every cell type can implement its own update and aging rules. Swapping out the value handling into cell's code has the advantage that the same map algorithms especially *GridMapDrawings* (see Section 6.1.4) can be used with different cell types. Additionally every cell must manage an update index labeled with *UpdateIndex* . This update index indicates, if a cell was already updated within an iteration.

In this work following cell types are implemented. It is also possible to develop new cell types in as long as the required methods are provided.

GridCellBase

The class *GridCellBase* provides a basic implementation for grid cells. This cell can handle probability values represented as *floats* in the intervall of $[0;1]$. The cell value is managed as an occupancy value without any transformation. A sequential cell update is realized by the binary Bayes filter described in Section 2.4.1. Additionally this cell type assumes that the prior $p_0(\mathbf{m}_i) = 0.5$ is given, so the simplified equation 2.10 is used here.

As the environment is dynamic for the given scenario, the grid map should have the ability to age cell values. This feature is implemented by the cells too, so every cell type can have own aging rules. Cell aging is computed by the *doAging* method which depends from elapsed time since last update and a free selectable aging parameter which defines usually the maximum non-updated data age of the cell.

LogOddsGridCell

The class *LogOddsGridCell* extends *GridCellBase*, so all base functionalities of *GridCellBase* are available. The only difference to *GridCellBase* is the used internal value representation. Here the log odds representation instead of the plain occupancy probability is used. Analogous to *GridCellBase*, the assumption of the prior $p_0(\mathbf{m}_i) = 0.5$ is given here, so for update purposes the equation 2.14 is used in this case. This cell type is currently not used because the computational cost of logarithm and exponential operations are too expensive.

CountGridCell

Especially for the camera sensor model (see Section 6.1.6) the *CountGridCell* is developed which inherits *GridCellBase*. In addition to the occupancy probability two counters are introduced. These counter are used to count how often the cell is visited as free (c_{free}) as well as occupied (c_{occ}). and are modeled as floating points numbers in order to handle uncertainty in measurements which are updated by separate methods.

An update for a counter c is executed by *updateFree* or *updateOccupied* using:

$$c_t = c_{t-1} + c_{update}, \tag{6.1}$$

with c_{update} denoting the update value.

Furthermore the cell holds a reliability value which depends on the cell information age. For this reason, the *doAging* method is overloaded to age only the cell reliability instead the occupancy value. Let rel denote the cell's reliability and Δt the time since the last cell update, then following aging is used at time t:

$$rel_t = max\{0; rel_{t-1} - \Delta t / t_{max}\} \tag{6.2}$$

where t_{max} is a free selectable parameter declaring the maximum age of non-updated data. Finally the occupancy value of a cell can be calculated by:

$$p(\mathbf{m}_i) = \begin{cases} 0.5, & \text{for } c_{free} + c_{occ} = 0 \quad (6.3) \\ \dfrac{c_{occ}}{c_{free} + c_{occ}} * rel + 0.5 - \dfrac{rel}{2}, & \text{otherwise} \quad (6.4) \end{cases}$$

If reliability reaches zero, the cell is reset to its initial values. Aging a separate value instead the counting values has the advantage that accumulated knowledge will not be lost until the reliability reached zero or in other words the cell will not be reset while updates are performed within the time t_{max}. For this reason, an update will be applied considering the cell data, even if the reliability is next to zero. This approach is introduced because the camera field of view is much smaller in contrast to laser scanner, so the camera head must be moved very often. In this situation it is not desirable to immediately start aging the cell state.

Du to the dynamic nature of the environment during a soccer game it is required that the cell is able to update quickly adapt to measurements. For this reason two modifications are introduced. First of all

the counters are limited to an upper bound, so the cell occupancy probability can change faster in the presence of moving obstacles. For this reason equation 6.1 is changed to:

$$c_t = min\{c_{max}; c_{t-1} + c_{update}\}, \tag{6.5}$$

where c_{max} defines the maximum count and can be selected free. Furthermore the contrary count is always reduced by an update. When the free count is updated, the occupied count is decreased by following equation:

$$c_{occ} = max\{0; c_{occ} - c_{update} * \frac{c_{occ}}{2}\}, \tag{6.6}$$

and analogous in the inversed situation the free count is decreased by an update of occupied count:

$$c_{free} = max\{0; c_{free} - c_{update} * \frac{c_{free}}{2}\}, \tag{6.7}$$

This last modification is intended to enable more reactive cell updates towards changes in the measurement input caused by moving obstacles and even so be robust against occlusions.

EMACountGridCell

The *EMACountGridCell* is an extension of *CountGridCell* and uses exponential moving average (EMA) for count updates. EMA is a type of infinite impulse response filtering method and also known as an exponentially weighted moving average. All collected data is added with weighting factors to the final value. For each older data the weighting is decreased exponentially. EMA can be calculated using:

$$c_t = \alpha * c_{update} + (1 - \alpha) * c_{t-1}, \tag{6.8}$$

where the coefficient α is the degree of weighting decrease. A higher value discounts old data faster.

EMACountGridCell is implemented for just experimental purposes and is not used for the final approach, because updating as well as discounting depends on the single parameter α which cannot be handle complex update rules such as *CountGridCell*. Either it updates quickly but ages data slowly or vice versa, so in dynamic environments it cannot be used without any modifications.

WrapperCell

Sometimes standard data types like *float* should be used, but this data types do usually not satisfy the requirements for usage as cell type (see appendix 9.1). For this reason the *WrapperCell* is introduced which implements all required methods as templates and thence allows the usage of standard data types.

6.1.3 Merging of Maps

The *MapMerger* is a generic cell as well as map fusion module. It provides merging of heterogen cell types and whole maps with the aid of externally implemented merging functions, denoted *MergeFunction* throughout this work. These are given as parameter to the *MapMerger* and implement the merge functionality. Swapping out the merging code enables to use individual merging strategies from a simple maximum operation up to a complex update rule like the binary Bayes filter.

For this purpose, the *MapMerger* calls the given *MergeFunction* which is available as a function pointer. The *MergeFunction* takes two references to cells which should be merged. An example of merging function is given in algorithm 1.

The fusion process itself uses the methods specified by cell requirements (see appendix 9.1) to perform the merging strategy. The result is returned as a new cell. Analogously the *MapMerger* can handle full maps by iterating over all cells in both maps and cell-wise calling of merging functions. The results are written back into the target map. Here, it has to be guaranteed that both maps have the same dimensions. Algorithm 2 illustrates the functionality of *MapMerger* merging $MapFrom$ into $MapTo$ in pseudo code. The algorithm of *MapMerger* for single cell is analogous, but without the inner loops.

In several situations the merging process can sped up by exploiting certain cell characteristics. In Section 2.4.1 is already proven that updating a cell with the measurement $p(\mathbf{m}_i|z_t, x_t) = 0.5$ via Bayes rule has no effect. For this reason the merging process can be instructed to skip applying updates with certain values. This instruction is denoted by the parameter *IgnoreValue* throughout this work. When *IgnoreValue=-1* is given to a *MapMerger* or rather to the *MergeFunction*, no special values will be handled and every update will be processed.

Analogously the cell's generation of a new *UpdateIndex* can be suppressed. When *IgnoreValue=-1* is given to a *MergeFunction*, the cell's last update index is kept.

Algorithm 1 MergeCells(cell1,cell2,IgnoreValue,UpdateIndex)

```
1: create new cell
2: if IgnoreValue = -1 or cell1.value = IgnoreValue then
3:    return cell1
4: end if
5: cell = result of merging strategy with cell1 and cell2
6: if UpdateIndex ≠ -1 then
7:    cell.setUpdateIndex(UpdateIndex)
8: end if
9: return cell
```

Algorithm 2 MapMerger(MapTo,MapFrom,MergeFunction,IgnoreValue,UpdateIndex)

```
1: if MapTo.dimension ≠ MapFrom.dimension then
2:    return
3: end if
4: for x := 0 to MapTo.sizeX do
5:    for y := 0 to MapTo.sizeY do
6:       cell := MergeFunction(MapTo.getCell(x, y), MapFrom.getCell(x, y),UpdateIndex,IgnoreValue)
7:       MapTo.setCell(x, y, cell)
8:    end for
9: end for
```

6.1.4 Grid Map Drawings

In the RoboCup soccer scenario arbitrary obstacle shapes can occur in several situations. For this reason the abstraction layer *GridMapDrawing* for updating a grid map is introduced. Every *GridMapDrawing* module realizes with given drawing parameters its own drawing rules for updating grid maps. In the scope of this work several *GridMapDrawing* modules are implemented which are described below. It is also possible to extend the available set of modules for future applications.

All *GridMapDrawing* modules accept world coordinates as drawing parameters which are transformed internally into map coordinates with the map's transformation matrix ${}^{m}T_{w}$. They allow to scale all resulting update values with a factor called *ValueScaling*. This factor enables the possibility to respect for instance the reliability of the current self-localization state estimate. This feature is important because the consistence of a learned grid map can only be guaranteed, when the robot's state especially the actual reliability of robot pose is considered.

The features of *MapMerger* (see Section 6.1.3) are supported. For this reason a map merging function can be given as argument to every *GridMapDrawing* module. This approach provides more flexibility in update strategies using a *GridMapDrawing*.

GridMapDrawingBeam

In Section 5.2 the obstacle percepts are introduced. There it was explained that a set of obstacle percepts has similar properties to a LIDAR scan. For this reason a drawing is required which is able to draw single rays into a grid map. This drawing is called *GridMapDrawingBeam* throughout this work.

A ray is drawn by giving to *GridMapDrawingBeam* a start and end point in the world. All cells within start and end point are updated as free. For this purpose the generalized Bresenham line algorithm described in Section 2.7 is used as drawing algorithm for rays. While the start cell is always updated as free, the end cell can either be updated as free as well as occupied. This depends on the given boolean parameter *IsObstacle*.

Furthermore the measurement error has to be considered. The higher cell's distance to robot, the lesser the cell update is counted. In this work a linear measurement error is assumed which can be computed more efficient than non-linear error estimations. This error is modeled as measurement reliability and describes the trust in measurement for the actual visited cell. This approximation can be computed efficiently using:

$$rel_d(x,y) = 1 - min\{1, d_r(x,y)/d_{max}\}, \tag{6.9}$$

where $d_r(x,y)$ is the cell's distance of cell at position (x,y) to the robot. The parameter d_{max} defines the maximal trusted measurement distance.

GridMapDrawingEllipse

Unknown shapes can be approximated by ellipses. Ellipses have the advantages that they can approximate many shapes appearing during a soccer game well and fortunately they can be derived efficiently.

Due to a normal filled circle can be computed easily, it is considered first. In this work all cells within the Manhattan distance of the circle's radius c_r are treated as potential element of the circle. All these cells are checked if they are within the distance c_r to circle's center. When a cell is in fact within the cell's radius, it will be updated by the given *MergeFunction*. All updates are done with the same value given as parameter to the drawing.

An ellipse can be seen as a stretched circle, so this approach can be modified to handle any ellipse. For this purpose the scaling matrix T_s is introduced. This transformation matrix defines the stretch factor between a vector $\vec{x} = (x,y)^T$ on the non-rotated ellipse and the vector $\vec{u} = (u,v)^T$ on the coordinate system:

$$\vec{u} = T_s \cdot \vec{x} = \begin{pmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{b} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \tag{6.10}$$

$$\Leftrightarrow \quad \vec{x} = T_s^{-1} \cdot \vec{u} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix} \tag{6.11}$$

where $\vec{x}$ is a vector on the non-rotated ellipse coordinate system and $\vec{u}$ is the same vector on the unstretched circle coordinate system. For this holds:

$$d_u = \sqrt{u^2 + v^2} \tag{6.12}$$

$$= \sqrt{\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2} \tag{6.13}$$

$$= \vec{x}^T T_s \vec{x} \tag{6.14}$$

which is illustrated in figure 6.1 by an example, where the unstretched distance d_u is computed by equation 6.14 from vector $\vec{r}$ in the non-rotated ellipse.

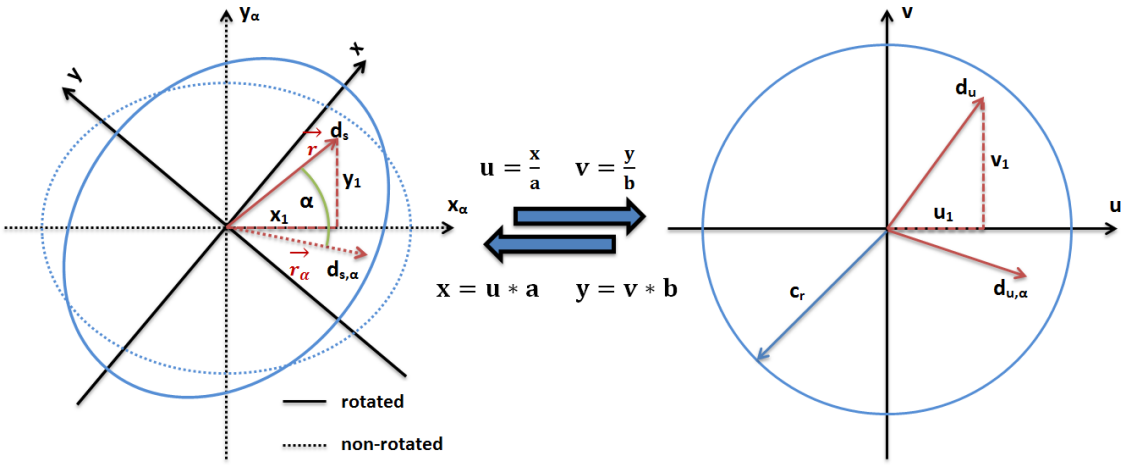


Figure 6.1: Illustration of standard scale.

Thus with equation 6.14 it is possible to calculate unstretched distance d_u of an ellipse but the rotation is still missing. For this purpose the rotation angle is needed which is used to compute the transformation matrix T_α:

$$T_\alpha = \begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix}. \tag{6.15}$$

The distance vector between ellipse's center and cell is given by $\vec{r} = (r_x, r_y)^T$. A rotation can be easily executed by multiplying the transformation matrix T_α to the distance vector $\vec{r}$ with:

$$\vec{r}_\alpha = \begin{pmatrix} r_{\alpha,x} \\ r_{\alpha,y} \end{pmatrix} = T_\alpha \cdot \begin{pmatrix} r_x \\ r_y \end{pmatrix} = T_\alpha \vec{r}. \tag{6.16}$$

Finally $\vec{r}_\alpha$ is applied as $\vec{x}$ into equation 6.14 to obtain the rotated unstretched distance $d_{u,\alpha}$. Both steps can be summarized to:

$$d_{u,\alpha} = \vec{x}^T (T_s \cdot T_\alpha) \vec{x}. \tag{6.17}$$

This result allows to decide by comparing $d_{u,\alpha}$ with unstretched ellipse's radius c_r, if a cell lies within the ellipse. Algorithm 3 demonstrates a pseudo code for the *GridMapDrawingEllipse* module.

GridMapDrawingCircle

The *GridMapDrawingCircle* module is a simple specialization of *GridMapDrawingEllipse* which draws filled circles with a given value. Here the *GridMapDrawingEllipse* is simply called by *GridMapDrawingCircle* without any rotation ($\alpha = 0$) and with fixated scaling matrix

$$T_s = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{6.18}$$

Algorithm 3 DrawEllipse(MapTo,x,y,MergeFunction,diameter,value,angle,UpdateIndex,IgnoreValue)

1: $(x_m, y_m)^T$:= MapTo.getMapCoords(x, y)
2: $c_r := \frac{diameter}{2}$
3: $T_e := (T_s \cdot T_\alpha)$
4: **for** $i_x := x_m$ - c_r to $x_m + c_r$ **do**
5: **for** $i_y := y_m$ - c_r to $y_m + c_r$ **do**
6: $\vec{r}$:= obtain distance vector from i_x and i_y
7: $d_{u,\alpha} := \|T_e \vec{r}\|_2 \cdot cellSize$
8: **if** $c_r \geq d_{u,\alpha}$ **then**
9: cell := new cell
10: cell.setValue(value)
11: MapMerger(MapTo,cell,MergeFunction,i_x,i_y,IgnoreValue,UpdateIndex)
12: **end if**
13: **end for**
14: **end for**

GridMapDrawingGaussian

2-dimensional Gaussian distributions all have in common that their shape is an ellipse. For this reason algorithm 3 of *GridMapDrawingEllipse* can be used as groundwork here. Mean μ and covariance matrix Σ of a Gaussian distribution defines parameters of an ellipse thus diameter, angle and the scaling matrix has to be determined from the covariance matrix. In contrast the ellipse's center point can be obtained directly from distribution's mean.

First of all the scaling matrix and angle α can be calculated by determining the Eigenvalues and Eigenvectors of the given covariance matrix with:

$$\Sigma \cdot \vec{x}_i = \lambda_i \cdot \vec{x}_i, \quad \vec{x}_i \neq \vec{0}, \tag{6.19}$$

where λ_i denotes the Eigenvalue and $\vec{x}_i$ the corresponding Eigenvector. Both Eigenvalues are the diagonal elements of the scaling matrix T_s of the Gaussian distribution and can be used directly in equation 6.14. The angle α can be obtained from Eigenvector $\vec{x}_1 = (x_1, x_2)^T$ with:

$$\alpha = atan2(x_2, x_1). \tag{6.20}$$

Now the ellipse's main attributes are defined but the unstretched diameter d_u is still missing. Obviously the distribution's diameter is theoretical unbounded, so it is necessary to set a limit. This limit can be taken from χ_2^2 quantile which defines the probability p that a distribution's point is within the ellipse, so the final diameter is composed by:

$$d_u = 2\sqrt{\chi_2^2(p)} \tag{6.21}$$

Finally all parameters for drawing the Gaussian distribution's shape are collected. But in contrast to the *GridMapDrawingEllipse* all cells within the distribution are not updated by the same value. Here, the Mahalanobis distance (see equation 2.25) of each cell is computed and used as updating value. For performance purposes the inverse covariance matrix Σ^{-1} is cached. The result is similar to real Gaussian distributions which is illustrated in figure 6.2. The darker a cell the higher is the occupancy probability.

Usually, the pose information of an object provides no covariance matrix. Nevertheless an uncertainty of the information should be modeled into the map. For this purpose, the goal is to model a Gaussian distribution that the object is localized within the ellipse given by a diameter d, a scaling matrix T_s and

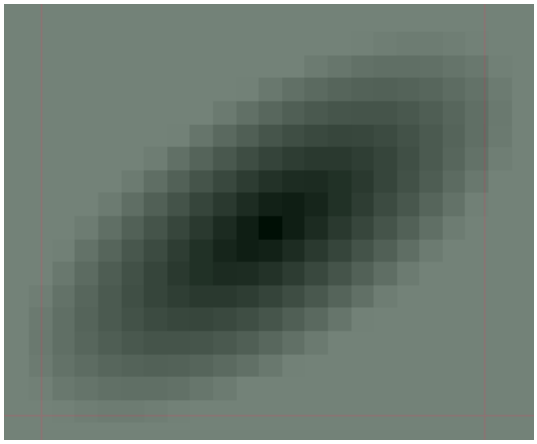


Figure 6.2: Example draw of *GridMapDrawingGaussian*.

a rotation angle α. The given diameter and scaling matrix is used to derive a distribution's standard deviation σ:

$$\sigma = \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix} = \begin{pmatrix} d \cdot a \\ d \cdot b \end{pmatrix}. \tag{6.22}$$

Afterwards a simple basic covariance is calculated with:

$$\Sigma_0 = \begin{pmatrix} {\sigma_1}^2 & 0 \\ 0 & {\sigma_2}^2 \end{pmatrix}. \tag{6.23}$$

In equation 6.23 the assumption is made that there is no correlation between both dimensions. Finally the covariance matrix has only to be rotated with transformation matrix T_α (see equation 6.15):

$$\Sigma = T_\alpha \cdot \Sigma_0. \tag{6.24}$$

The resulting covariance matrix can be used to draw a Gaussian distribution which shape is defined by given parameters. If additional a $\chi_2^2(p)$ quantile should be respected, it must be added previously to the diameter with:

$$d_{\chi_2^2(p)} = d \cdot \sqrt{\chi_2^2(p)}. \tag{6.25}$$

GridMapDrawingRectangle

Sometimes the shape of an object is known and can be approximated by a rectangle. In the soccer field scenario the goals can be approximated by drawing a filled rectangle which is implemented by *GridMapDrawingRectangle*. Therefor only the rectangle's center as well as length and width has to be defined.

6.1.5 Map Objects

Every modeling holds different data and even provides other methods to request this data. Due to they cannot be used in a generic way, the *MapObject* class is introduced, abstracting these modelings and their

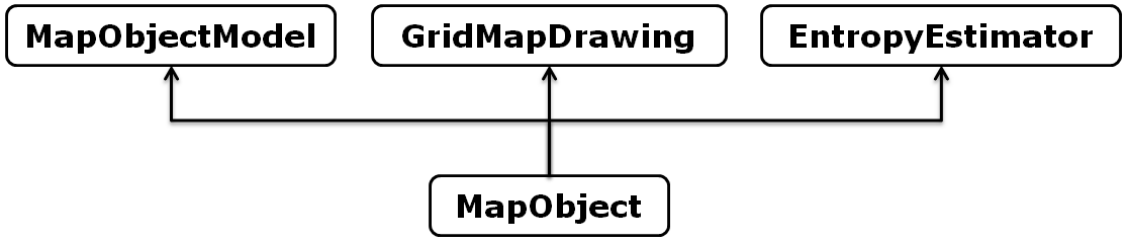


Figure 6.3: Design and modules of map objects.

data. This abstraction offers to handle every modeling's data in a generic way and enables the use for mapping purposes. The basic structure is illustrated in figure 6.3 and is explained in the following.

At first a container class called *MapObjectModel* is introduced which saves all model data like position or model confidence including a covariance matrix. The usage of an extra container class has the design consideration of preventing dependencies cycles of all *MapObject* classes. The data must be updated explicit by calling the corresponding method in the *MapObject* class. Some drawing modules need a covariance matrix, why it is automatically generated, if no one given with the last update. This generation is already described in Section 6.1.4.

Furthermore, every *MapObject* holds an instance of the *GridMapDrawing* and *EntropyEstimator* module. Each *MapObject* is able to draw itself in an occupancy grid map and estimates its information content in an entropy map. For this purpose every call of such instance gets the *MapObject*'s *MapObjectModel* as parameter to obtain all necessary data.

For both instance types a dummy class exists providing the *MapObject* to be used exclusive either for occupancy grid map or entropy map.

Map Object List

MapObjectLists are a container for a set of *MapObjects*. Elements can be added, updated and removed, so this lists can be updated dynamically. Each list's element has to be assigned a global unique identification number (short: GUID) which is used to identify the element for every operation. Additionally these lists are able to draw their content in a given map by iterating over all elements. Using lists instead saving all data in separate maps has considerable performance advantages. Adding the list's content to a map is computationally more efficient, because only concerned cells are updated instead of fusing several complete maps together.

In practice, two kind of list are used in this work. The first list is called *StaticMapObjectList* containing all static object modeling, thus the environment elements which will not move during a soccer game. Usually, these elements are not updated except for entropy updates. In contrast the *DynamicMapObjectList* manages all movable objects of the game.

Map Objects Drawings

Before the *GridMapDrawing* module can be used in combination with *MapObject*, they have to be interfaced with this class. For this reason *MapObjectDrawing* modules are introduced which are simple wrapper classes for *GridMapDrawing* modules. They are able to extracts all parameters from *MapObjectModel* needed by the corresponding drawing. Every *MapObject* holds an instance of a *MapObjectDrawing*, so each *MapObject* is able to draw itself into a grid map. In this work following *MapObjectDrawing* modules are available:

- **MapObjectDrawingCircle:** Draws a *MapObject* as filled circle.

- **MapObjectDrawingGaussian:** Draws the *MapObject*'s position represented by a Gaussian.
- **MapObjectDrawingRectangle:** Draws a *MapObject* as filled rectangle.
- **MapObjectDrawingNone:** Dummy drawing which draws nothing in grid map.

Entropy Estimator

Analogous to *MapObjectDrawing* the *EntropyEstimator* modules are wrapper classes for drawing functionality. But in this case they estimate the information content of an *MapObject* and are able to draw their result in a entropy map. All used data is directly taken from other modelings saved in *MapObjectModel*. The drawings are usually Gaussian distributions which well suited for modeling for uncertainty. In future work it is intended to extend the *EntropyEstimator* for instance with a history. In this way it is possible to predict the future movement of objects giving a better entropy estimation of the map. In the context of this work the following estimators are implemented:

- **MapObjectEntropyEstimatorGaussian:** Approximates entropy as a Gaussian distribution.
- **MapObjectEntropyEstimatorNone:** Dummy estimator which estimates no entropy.

One of these instances is hold by every *MapObject*, so they are able draw their own entropy in a grid map.

6.1.6 Camera Sensor Model

An approach using a monocular camera for learning occupancy grid maps is presented in [36]. The Darmstadt Dribblers use a similar robot system which can analogously be extended for use with occupancy grid maps.

The first challenge of a camera sensor model is to determine the projection of the camera field of view onto the soccer field. The obstacle perceptor has the ability to project every camera pixel into robot coordinates, using the camera head matrix which includes the robot's whole kinematic chain down to ground touching foot. As it is necessary to obtain the world coordinates of the camera image's four corners, in this work the obstacle perceptor is extended to provide even these information which is only possible if the distortion of the camera lense is small. In figure 6.5c the final projection of camera's field of view can be seen as a red box.

The set of cells within the camera projection can thus be identified. This area can potentially be updated by obstacle percepts. As mentioned in Section 5.2 the obstacle percepts can be modeled as

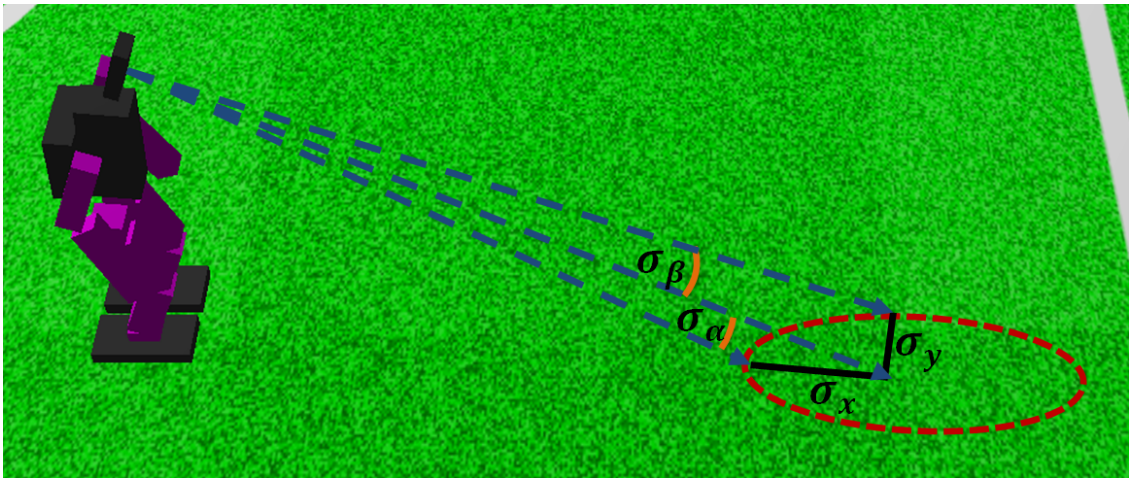


Figure 6.4: Standard deviation of obstacle percept.

rays casted into the environment. Every obstacle percept is drawn as a ray from the projected start coordinates of its original scanline to the percept's own coordinate. Usually an obstacle percept which is classified as free has its coordinates on the upper edge in the projection field, so all cells within the scanline projection edges are updated as free.

The rays are drawn by using the *GridMapDrawingBeam* described in Section 6.1.4. The ray endpoints are non-linearly distributed caused by field projection, so it is possible that single cells are not visited by the Bresenham line algorithm. In figure 6.5c such cells can be seen within the projection field. This lack is not considered further, because the permanent head movement and accumulation of updates will cover all cells without side effects.

When an obstacle percept is classified as occupied, its projected position corresponds to the lower edge of the detected obstacle. For this position the measurement error has to be considered. For this purpose the head's angular standard deviation $\sigma_\alpha, \sigma_\beta$ is used which are approximated dynamically, depending on current robot state (i.e. head angular rates and robot motion state) [21]. The standard deviation with distance σ_x, σ_y is approximated by projecting points in angular difference $\sigma_\alpha, \sigma_\beta$ for each head axis and resolve the distance to original obstacle percept coordinate (see figure 6.4). It must be noticed that the standard deviation in y axis behaves non-linear.

For simplicity it is assumed that the angular standard deviations of the camera head are independent from each other, so when σ_x, σ_y are obtained, they can analogously be used with equation 6.23 to compute Σ_0 which is transformed with equation 6.24 to the covariance matrix. In this case the relative angle between robot orientation β_r and object is used as rotation angle α.

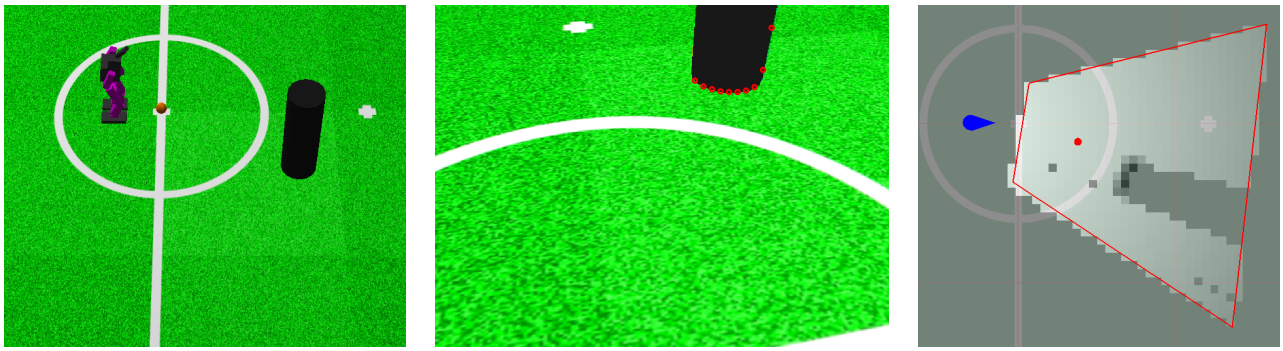
As the measurement error is determined with a covariance matrix and the mean can be taken from obstacle percept's coordinate, the obstacle percept can be drawn into the map using a Gaussian distribution. This parallels the approach used by Elfes [11], [12]. A ray is drawn in direction of to the obstacle percept's coordinate, where finally the percept's Gaussian distribution is drawn.

Using a Gaussian distribution based on angular standard deviation in head angles, respects the measurement errors due to fast head movements. Unlike when using the old sector model, no walls are built up on fast pan movements, because in this case the angular standard deviation is high which is considered in the map update. Cells are updated with higher uncertainty which shows that the Gaussian distribution deals satisfyingly with high inaccurate measurement scenarios. Furthermore, outliers generated by obstacle perceptor are filtered. An obstacle percept is only drawn using a Gaussian drawing if the preceding obstacle percept is closer than a threshold distance away. This filtering assumes that every obstacle is large enough to induce at least two obstacle percepts within the threshold distance.

Figure 6.5a shows a typically situation where an obstacle is in front of the robot, a pylon is used for testing purposes here. In figure 6.5b the grabbed camera image with detected obstacle percepts is shown which is transformed into a map update, illustrated in figure 6.5c. Here it can clearly be seen that outliers are not updated into the map.

The numbers of scanlines are normally higher than the map's resolution can gather, so the closer the more often a cell will be visited by different rays. Due to these overlays would update a cell several time within a single update cycle, here the cell's *UpdateIndex* is used enabling unique cell updates by skipping already visited cells. Furthermore a beam drawing will be interrupted, when a visited cell classifies itself as occupied and was already updated in the same iteration. For this reason a shading appears behind an obstacle which models the assumption that the robot never can see the space behind an obstacle. Obviously there is a difference between high and low resolutions maps. High resolutions have high memory and processing time requirements and cannot be transmitted to other team mates easily. On the other hand coarse resolutions are not able to map all updates properly, so information representation becomes fuzzy. For these reasons the map resolution should be chosen carefully.

Analogous to the sector model the problem with partially visible obstacles stays unresolved, because their base point in the image cannot be projected into the soccer field. For this reason such obstacles are not updated into the grid map preventing inconsistency map state and thus are not visible in grid map. In usual cases it was already seen in past including its base point. Knowledge about this obstacle thus



(a) Original scenario in simulator.

(b) Grabbed camera image with detected obstacle percepts (red dots).

(c) Sensor Model obtained from obstacle perceptor.

Figure 6.5: Illustration of camera sensor model.

is available in the map which can also used by active gaze control to generate a new action, where the obstacle's base point in image can be seen again.

6.2 Occupancy Grid Map Generation

In previous Section 6.1 the whole framework for maintaining obstacle grid maps is presented. Using this as groundwork, learning of dynamic grid maps is described in this section.

For obstacle percepts the map *ObstacleGridMap* is introduced, using the *CountGridCell* described in Section 6.1.2 which is designed for this map type. All update and aging operations are performed in this map, so the map's content is based only on obstacle percepts. This assures that all updates are proceeded consistently. Every time when a image is grabbed from camera, all image processes are run. Among these is the obstacle perceptor, so new obstacle percepts become available. In the first step these obstacle percepts are used to update the obstacle grid map with the support of the camera sensor model (see Section 6.1.6). Afterwards all non updated cells are aged by the given aging rule in *CountGridCell* (see Section 6.1.2).

In figure 6.6 the given scenario in the simulator is shown. The learned occupancy grid map is illustrated in figure 6.7a. Here the higher the cell's occupancy probability the darker it will be displayed

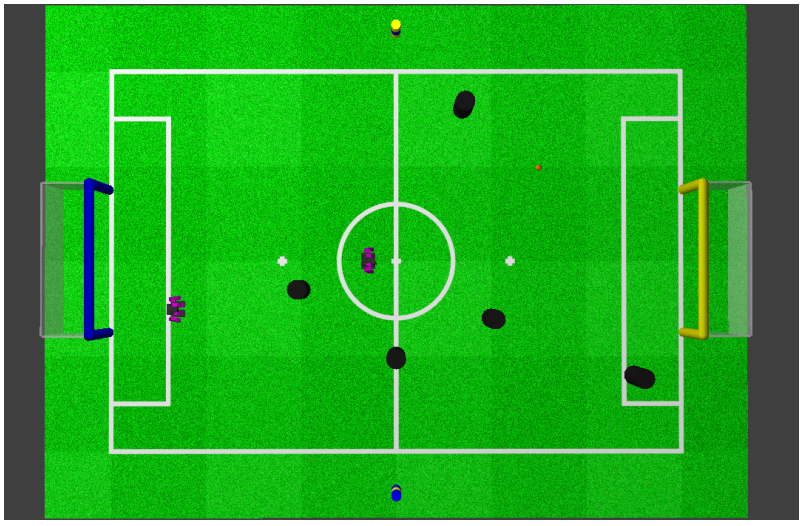
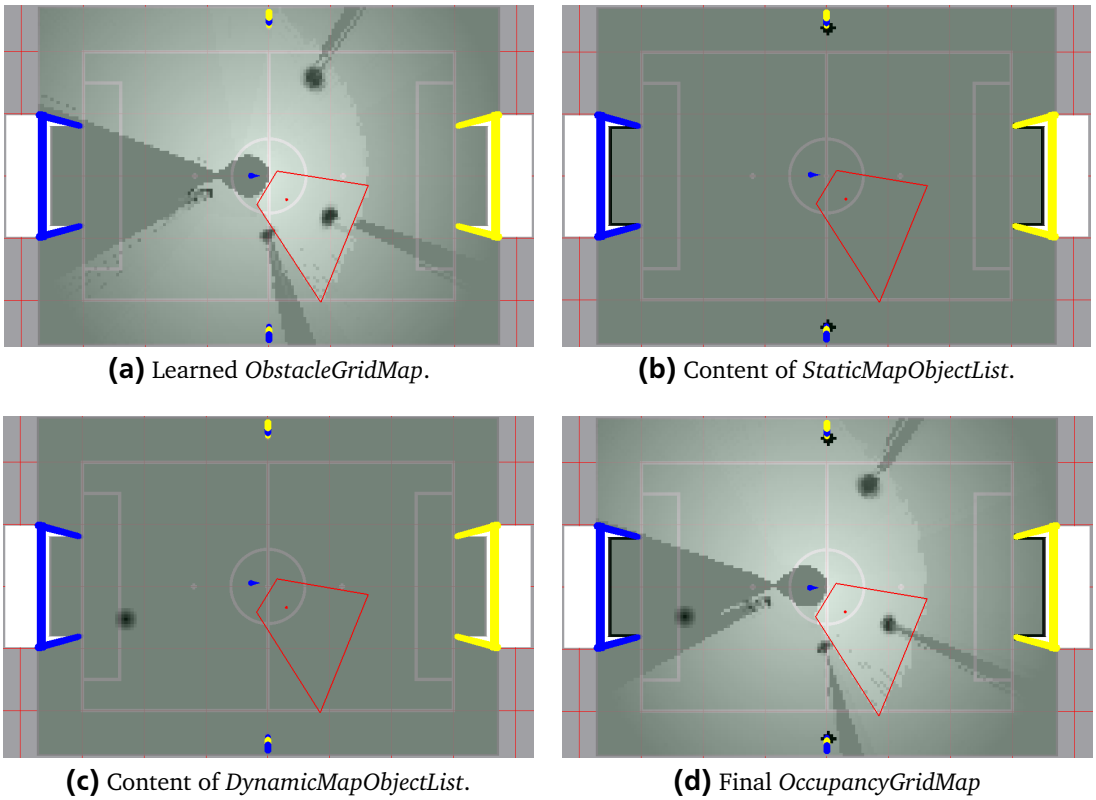


Figure 6.6: Scenario given in simulator.



(a) Learned *ObstacleGridMap*.

(b) Content of *StaticMapObjectList*.

(c) Content of *DynamicMapObjectList*.

(d) Final *OccupancyGridMap*

Figure 6.7: The learned *ObstacleGridMap* is shown in (a). Finally the *StaticMapObjectList* (b) and *DynamicMapObjectList* (c) are added into the resulting *OccupancyGridMap* shown in (d).

in the map which is clamped between "0" (free) and "1" (occupied). For this reason cells without any knowledge and with it the maximum uncertainty are shown in gray. Interestingly, the pylon in the yellow penalty area is invisible in this scenario. This phenomenon results from occlusion by another pylon in line of sight.

Furthermore in every iteration of *MapObjectModeling* all *MapObject* data collected in a *MapObjectList* is updated. For this purpose, the *ObstacleMapModeling* requests all other models and pushes their information back into the corresponding *MapObject* identified by its GUID. In figure 6.7b only the *StaticMapObjectList* is shown. The *DynamicMapObjectList* contains just position information of team mates which are communicated via wireless (see figure 6.7c). This allows to cover regions of the map which cannot be seen from robot's current pose.

In the final step the *OccupancyGridMap* is generated. To assure consistency of the *ObstacleGridMap*, all values are copied into the *OccupancyGridMap*. Afterwards the content of both *MapObjectLists* (*StaticMapObjectList* and *DynamicMapObjectList*) are added to the *OccupancyGridMap*. All data from the *DynamicMapObjectList* is regularly aggregated into the target map using a Bayesian update. In contrast, the *StaticMapObjectList* is added by overwriting all cell values in the target map, because this data is static. The resulting *OccupancyGridMap* is shown in figure 6.7d.

6.2.1 Failure Handling

During the robot operation several failures can occur. One of these failures is falling to the ground. When the robot falls to ground the kinematic chain cannot be obtained anymore, so no projections of camera images possible. The system can detect such situations and among other measures stops computing the camera head matrix. All image processes suspends automatical, if no new camera head matrix is

given. Therefrom also no obstacle percepts are generated until the robot stands up, so automatically the obstacle *ObstacleGridMap* is not updated.

Another difficult failure is wrong-location of the robot. While jumps in self-localization as a symptom can easily be detected, the same is not true for wrong pose estimates themselves. Currently, jumps in self-localization suspend building of the grid map, until the robot is located at one pose again.

The map's aging process is still executed in any failure case, considering the growing uncertainty of old data.

6.2.2 GUI Parameters

In figure 6.8 the current available parameters for setting up the *OccupancyGridMap* generation during runtime is shown.

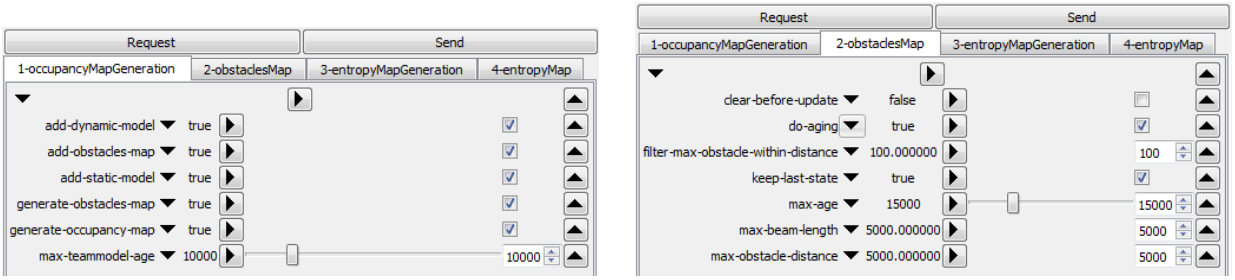


Figure 6.8: GUI parameters of occupancy grid map generation.

In the following these parameters are explained in table 6.1:

parameter	**description**	**units**
add-dynamic-model	enables adding *DynamicMapObjectList* into *OccupancyGridMap*	-
add-obstacles-map	enables adding *ObstacleGridMap* into *OccupancyGridMap*	-
add-static-model	enables adding *StaticMapObjectList* into *OccupancyGridMap*	-
generate-obstacles-map	enables updating *ObstacleGridMap*	-
generate-occupancy-map	enables generating *OccupancyGridMap*	-
max-teammodel-age	defines max age of pose data from team mates	ms
clear-before-update	enables clearing *ObstacleGridMap* before updating	-
do-aging	enables aging	-
filter-max-obstacle-within-distance	max distance within two following obstacle percepts	mm
keep-last-state	enables saving last state of *ObstacleGridMap* when updating is disabled	-
max-age	max age of non-updated cells	ms
max-beam-length	maximal distance where a cell will updated as free	mm
max-obstacle-distance	maximal distance where a cell will updated as occupied	mm

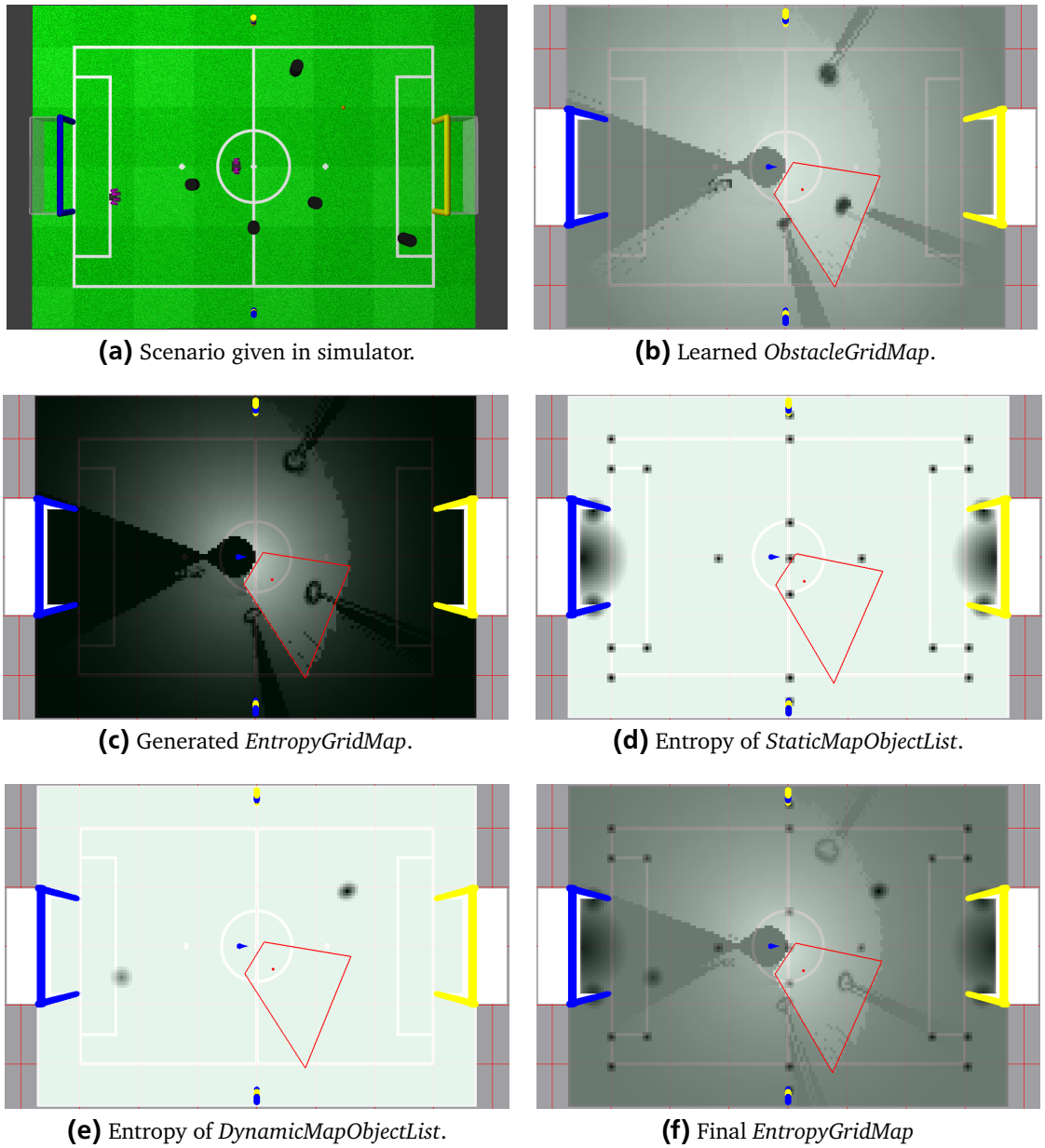
Table 6.1: Explanation of GUI parameters.

6.3 Active Gaze Control

In this section the implemented techniques in Section 6.1 are reused and extended for use in entropy-based gaze control. In the following section a novel light weight approach for entropy-based gaze control is introduced.

6.3.1 Entropy Map Generation

The basic *EntropyGridMap* is generated directly from the learned *ObstacleGridMap*. The entropy of all cells is estimated by using the Shannon entropy equation 2.20 described in Section 2.5.4. A resulting map is illustrated in figure 6.9c generated from the occupancy grid map shown in figure 6.9b. A white cell



(a) Scenario given in simulator.

(b) Learned *ObstacleGridMap*.

(c) Generated *EntropyGridMap*.

(d) Entropy of *StaticMapObjectList*.

(e) Entropy of *DynamicMapObjectList*.

(f) Final *EntropyGridMap*

Figure 6.9: The learned *ObstacleGridMap* in (b) is transformed into the *EntropyGridMap* in (c). Finally the *StaticMapObjectList* (d) and *DynamicMapObjectList* (e) are added into the map. The resulting *EntropyGridMap* is shown in (f).

indicates that no useful information is available, while dark cells potentially provide large information gains. For visualization purposes, entropy is displayed in gray scale and mapped from white (minimum entropy) to black (maximum entropy).

It should be noted that obstacle edges provide a high entropy. This feature is desirable, so that obstacles are also considered as potential gaze points. Focusing on obstacle edges prevents partially visible objects that let the obstacle perceptor fail (see Section 5.2).

After the basic *EntropyGridMap* is available, both *MapObjectLists* have to be added. The list elements' entropy is modeled by Gaussian distribution (see Section 6.1.5). The information content of *StaticMapObjectList* is seen in figure 6.9d and of *DynamicMapObjectList* in figure 6.9e. In both cases, only around places where objects are located an entropy value is available. The entropy informations is added by addition resulting in the final *EntropyGridMap* which is illustrated in figure 6.9f. It should be noted that in figure 6.9f the *EntropyGridMap* gray scaling is already clamped.

In addition line intersection such as crosses are included in the *StaticMapObjectList*, so important visible landmarks are considered. The landmark entropy is coupled with the current reliability estimate of self-localization, so landmark entropy is increased, when self-localization becomes unreliable. This increases the priority of landmarks for active gaze control, so they will be focused on. The *DynamicMapObjectList* only contains the ball model and team pose model for entropy estimation. Analogously to landmarks the ball entropy increases if the estimated ball model reliability decreases. Adding the world state of opponent team members is planned for the future.

It is possible to give every object a priority within the interval $[0; 1]$ which can dynamically be adjusted at run time. This allows prioritize an object to be seen more often than others which enables role-based active gaze control. For instance, a goalie should focus more often to the ball than a striker who has to dribble towards the goal and simultaneously avoid obstacles. In the current state of work these priority are constant, but can be changed online through the GUI. The capability to dynamically change these priorities (e.g. depending on current robot behavior) is provided but not used yet.

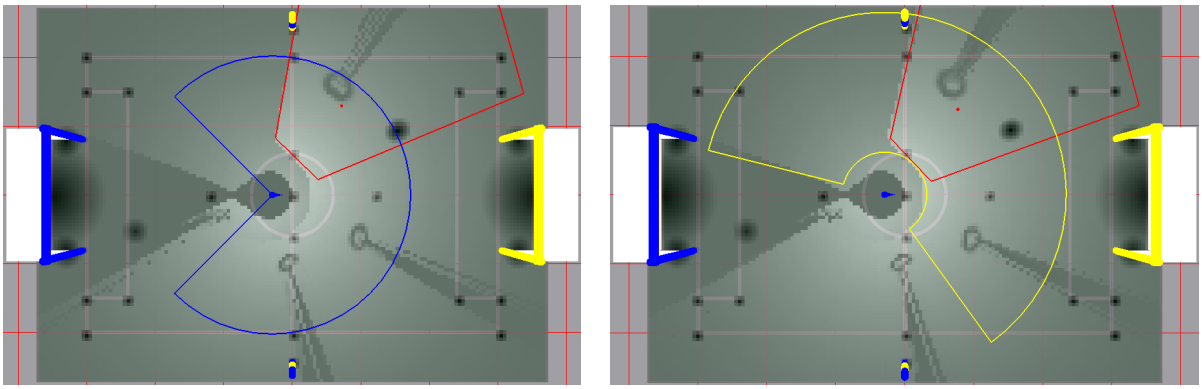
A potential weakness of the approach is the reliance on the self-localization of the robot in the global reference frame. In cases of self-localization failure, a backup system for generating head motion commands has to be used until self-localization is reliable again. A similar problem occurs, when the tracking of the ball is lost. The approach works on outdated information then the ball will probably not be rediscovered, when it decides to gaze on the ball. The active gaze control system should thus be used with fallback head behaviors to handle both mentioned failure cases.

6.3.2 Search Space and Movement Costs

To determine space and movement costs for the head control system, a target function to be maximized has to be found. The current focus point is already illustrated as red dot in figure 6.9, so the goal of this work is steering this red dot over the map in a way that maximal information is obtained and all necessary goal-oriented data is up-to-date. The problem is to find the focus point on the map that provides the highest potential information gain.

The constraints for search in the entropy map have also to be defined. Here two types of constraints are introduced: search space limitation and movement costs. In this work these constraints are modeled as costs respecting the current head state and head movement costs.

The search space is first to be limited to improve the time spend for searching the maximum. In this approach the search space is defined by maximum distance d_{space} and opening angle β_{space} relative to the robot. These parameters are used to reflect the physical as well as technical limitations of the robot. The opening angle represents the neck joint restrictions and the maximum distance respects the maximal useful tilt angle before looking too far over the horizon. Here, a metric parameter is chosen because it is more intuitive and the restrictions can be easier checked for a given point. Introducing the search space has the additional positive side effect that the maximal number of visible cells is limited, reducing the



(a) Illustration of search space.

(b) Movement costs in dependence of current head joint values.

Figure 6.10: Search space and movement costs.

computational resources for an *ObstacleGridMap* update by camera sensor model. In figure 6.10a the search space limitation is shown in blue.

Additionally, the cell position within the search space is weighted. For every point ${}^{r}p$ given in robot coordinates a linear weighting is calculated with:

$$w_{d_{space}} := \omega(\|{}^{r}p\|_2, d_{space}) \tag{6.26}$$

$$w_{\beta_{space}} := \omega(atan2({}^{r}p_y, {}^{r}p_x), \beta_{space}) \tag{6.27}$$

$$w_{space} := w_{space,d} \cdot w_{space,\beta}, \tag{6.28}$$

where ω is a clamp function defined with:

$$\omega(x, x_{max}) := 1 - min\left(1, \frac{x}{x_{max}}\right), \; x_{max} \neq 0. \tag{6.29}$$

Analogous to the search space, the movement cost can be defined by a maximal distance d_{cost} and opening angle β_{cost}. In this case these parameters are represented relative to current camera head joints. The opening angle restricts the maximal angular distance between the current and the next gaze point. Analogously, the metric maximal distance limits the pan movement distance across the map. The resulting set of allowed points is shown in yellow in figure 6.10b. The weighting for movement cost is:

$$w_{d_{cost}} := \omega(abs(\|{}^{r}p\|_2 - \|{}^{r}c\|_2), d_{cost}) \tag{6.30}$$

$$w_{\beta_{cost}} := \omega(atan2({}^{r}p_y, {}^{r}p_x) - \beta_c, \beta_{cost}) \tag{6.31}$$

$$w_{cost} := w_{cost,d} \cdot w_{cost,\beta}, \tag{6.32}$$

where $\|{}^{r}c\|_2$ is the relative distance from robot position and the next cell which should be focused on. Considering movement cost, reduces measurement inaccuracy caused by fast pan movement and asynchronous reading of camera images and head joint angles.

Finally, all possible solutions can be found within the intersection of the search space limitation and movement costs. It has to be remarked that also objects outside of this intersection can possibly be seen. This formulated restrictions are only valid for camera's focus point and not for the viewed area. The final cell weight w is given by:

$$w := w_{space} \cdot w_{cost} \tag{6.33}$$

and represents the total amount of costs, if this cell would be focused on.

6.3.3 Search for Best Gaze using Particles

Due to the nature of given entropy map, the searching problem has to be formulated as maximum search. The map's structure and building process is known, hence it is possible to take advantage of this knowledge, reducing computational cost for maximum search.

Many approaches use complex and expensive search algorithms which are often based on artificial neural networks. In [3] a Monte Carlo Exploration and feed-forward neural network is used to obtain a good hypothesis for gaze control. It requires so much resources that many simplifications and approximations have to be incorporated to make realtime operation feasible.

Hence the general idea is to track every single update of each cell of the *EntropyGridMap*. For this purpose the best n cells are marked by particles at any time. It has to be noticed that the concept of particles is used in the context of gaze control here and should not be confused up with particles from particle filtering approaches. Each particle holds the position p_{pos}, entropy $p_{H(X)}$ and weight p_w of a cell. All these particles are collected in a sorted list, where the best particle is on top and the worst at the end.

An order of particles has to be defined:

$$p_a < p_b := \begin{cases} p_{a,H(X)} \cdot p_{a,w} < p_{b,H(X)} \cdot p_{b,w}, & \text{or} \quad (6.34) \\ p_{a,H(X)} \leq p_{b,H(X)} \; and \; p_{a,w} \leq p_{b,w}, & (6.35) \end{cases}$$

where equation 6.34 has a higher priority than 6.35.

Every time when the *EntropyGridMap* is recomputed, all particles are initialized by randomizing their position around the known ball position, why it is possible that particles are not within the constraints (see Section 6.3.2). This has the effect that the particle weight p_w becomes zero, for which reason the equation 6.35 is introduced. This step gives a certain possibility that the ball is chosen for next gaze, although the constraints are not met. Currently there is no information about the path-planning available, so the ball is usually the best indicator in which direction the robot is moving. Whenever during an cell update a better position is found than the worst particle, this particle is moved to this position. As the particle approach has the trend of clustering particles around a global maximum, a

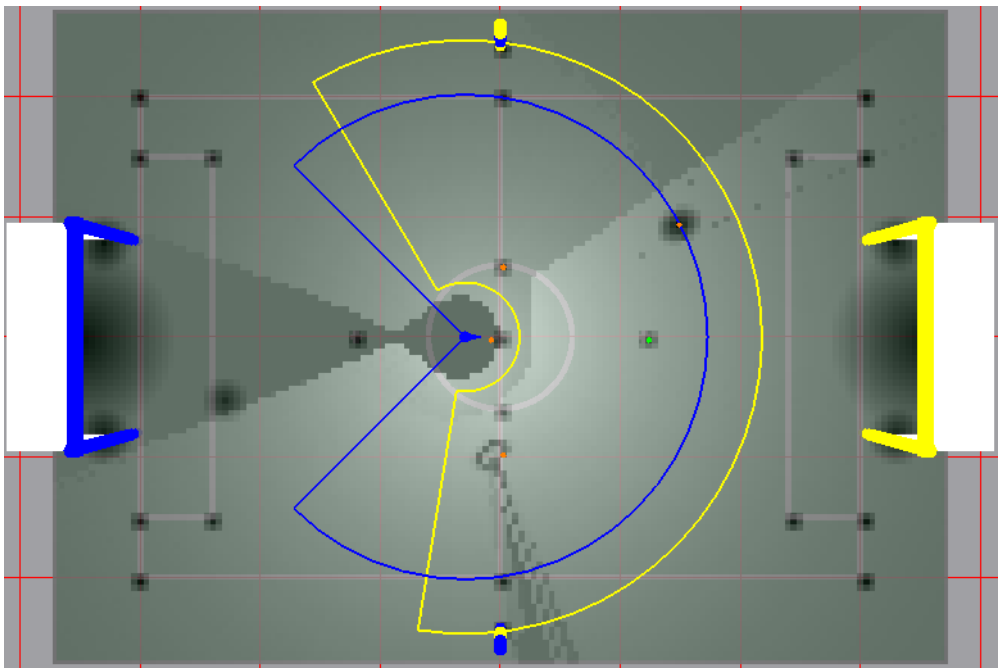


Figure 6.11: Result of searching for maximum via particles with $n = 5$.

minimal distance between particles must be maintained. If they are several particles within the minimal distance, the particle having the worst weight p_w is chosen. In this case the minimal distance requirement might be violated, but no critical clustering of particles is watched so far.

The most important and also dynamical game element is the soccer ball. The ball is able to move with high speed around the soccer field, so the ball has to observed more frequently than other objects. If the ball state is estimated with high reliability, one particle is added to this cell. This way it is assured that the ball is always a candidate for the next gaze. Analogously it has to be assured that the immediate region in front of the robot is well explored, avoiding collision and issues with partially visible objects (see Section 5.2). For this reason one particle is always added in front of the robot, similar to the method used for the ball.

When the *EntropyGridMap* generation is completed, the n cells with the highest entropy and weight are simultaneously available, representing local maxima. These are the candidates where to focus on next. In figure 6.11 an example for such candidates are given, where the green dot mark the best candidate and the orange dots all other candidates.

6.3.4 Field of View Entropy

The estimated entropy of a single cell does not provide information about the potential entropy of the whole field of view if the cell would be focused by camera. Considering only a single point for ranking of all particles unavoidably leads to stagnation of attention, because the candidates' surrounding entropy is ignored. Therefore the sum over all cells' entropy within the new field of view has to be approximated which is called field of view entropy (FOVE) throughout this work. Now the motivation of using particles for maximum search is given. Typically for each cell a full FOVE estimation has to be calculated, which has high computational overhead. For this reason the particle approach is used to consider a small set of candidates whose FOVE is computed saving a lot of resources. The main advantage is that computational cost scales linear with the number of used particles.

An ellipse is used to approximate the FOVE. Using an ellipse is much cheaper than exact computation, so it costs less computational resources. The camera matrix for the new gaze point has to be determined to simulate the camera head movement. Analogous to Section 6.1.6 the camera's opening angles are used to obtain the distance from the focused cell to the edges of the new FOVE. These distance are used for the elliptical approximation. In this case the lower edge of FOVE has to be used, because the resulting ellipse obtained by projection of the upper edge gives an insufficient approximation. It is much larger than the original FOVE.

The size of the field of view depends on the relative position to the robot. A field of view next to the robot is much smaller than one which is distant (see figure 6.12). Hence the more distant a particle the higher is the sum of entropy, for which reason only distant particles would always be ranked high. This problem is solved by determining the weighted mean of each FOVE. Every cell in a FOVE is weighted by its Mahalanobis distance to the center (see equation 2.25), using a covariance matrix with the standard deviation $\sigma = 1$ which is fitted to the elliptical approximation of field of view. Here, a Gaussian weighting is chosen to consider partial vision described in Section 5.3. Finally the weighted mean entropy is computed by:

$$\overline{H_{FOVE}(x,y)} = \frac{\sum_{x_i}\sum_{y_i}(1-\Delta(x_i,y_i))\cdot H(m_{x_i,y_i})}{n}, \tag{6.36}$$

where $(x_i, y_i)^T$ takes the position of every cell within the approximated FOVE and n denotes the amount of cells considered in the weighted entropy sum.

During the head movement to a new cell, several cells are explored on the way which are not necessary in the final FOVE. These seen cells should be considered in the ranking of all candidates. Here, this property is not considered, because is requires too much processing power.

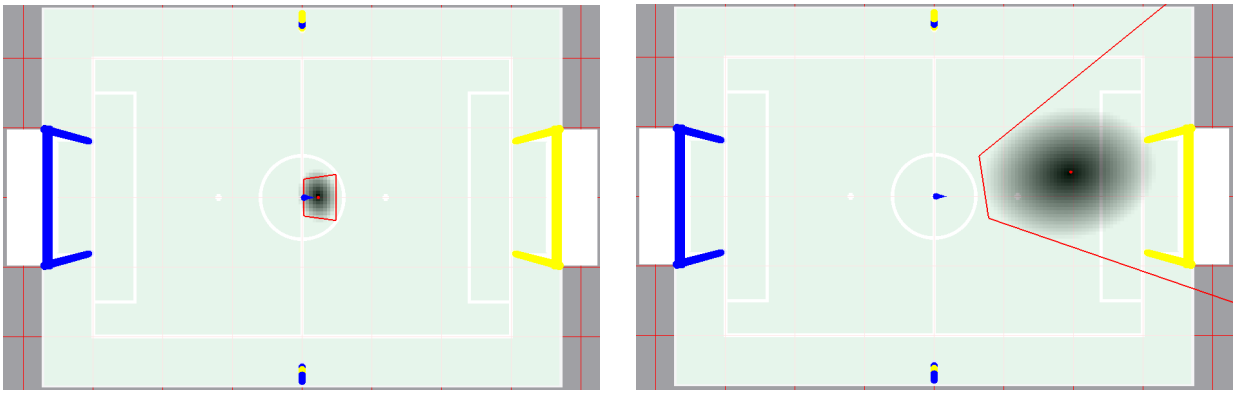


Figure 6.12: FOVE with Gaussian weighting.

The best gaze point is obtained by choosing the candidate with the highest mean of FOVE. Interestingly, it is not necessary that the best candidate has the highest mean of FOVE. After finishing the last movement the active gaze behavior requests the gaze point with the highest mean for next head movement.

It is still possible that the active gaze control attention remains static for a certain time. This behavior can be observed when a large obstacle is close to the robot, so the cells behind this object are not seen and thus are not updated. For this reason the entropy behind the object is still high and the active gaze control system will give it further attention. As countermeasure the concept of "activation dynamics" in [38] could analogously be adapted in this approach. For this purpose negative entropy particles have to be added to every used gaze target point which are aging out over the time.

6.3.5 GUI Parameters

The parameters of *EntropyGridMap* generation can be changed during the runtime. The available parameters are shown in figure 6.13.

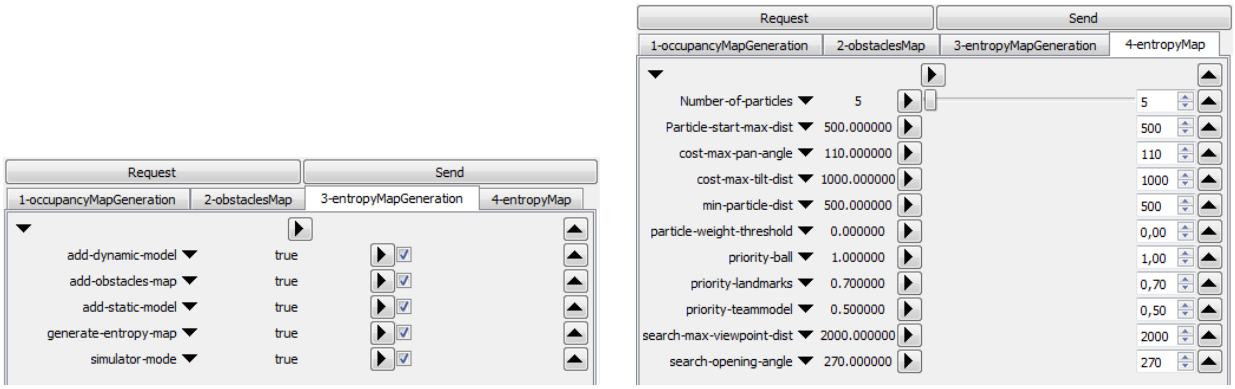


Figure 6.13: GUI parameters of active gaze control.

In the following these parameters are explained in table 6.2:

parameter	description	units
add-dynamic-model	enables adding *DynamicMapObjectList* into *EntropyGridMap*	-
add-obstacles-map	enables adding *ObstacleGridMap* into *EntropyGridMap*	-
add-static-model	enables adding *StaticMapObjectList* into *EntropyGridMap*	-
generate-entropy-map	enables *EntropyGridMap* generation	-
simulator-mode	self-localization and ball model reliability is inverted, hence landmark and ball entropy becomes visible in simulator	-
Number-of-particles	number of particles	-
Particle-start-max-dist	scattering of randomized particles	mm
cost-max-pan-angle	maximal allowed movement in pan	rad
cost-max-tilt-dist	maximal allowed movement in tilt	mm
min-particle-dist	minimal distance of particle to be respected	mm
particle-weight-threshold	minimal weight a cell must have to be considered	[0;1]
priority-ball	observation priority for the ball	[0;1]
priority-landmarks	observation priority for all landmarks	[0;1]
priority-teammodel	observation priority for all team mates	[0;1]
search-max-viewpoint-dist	maximal search distance to robot	mm
search-opening-angle	maximal search angle relative to robot	rad

Table 6.2: Explanation of GUI parameters.

7 Results

All experiments are performed with a real robot on the experiment soccer field in the lab of Darmstadt Dribblers. The experiments on the real robot prove that the use of the implemented approach is not only possible in simulation environment but also on the real robot, which provides additional constraints like limited processing capabilities.

7.1 Runtime Efficiency

An important property of the implemented approach is its computational cost and with it the execution time. To be feasible for operation on the real robot, the runtime has to be low enough as to not affect the existing control software. The high level cognition layer currently runs at a 30Hz update rate. This should be kept when the implemented approach is used to prevent any possible overall performance. For this reason the consumed processing resources are analyzed in this section.

In a first experiment performance is profiled using *Valgrind*[1] on Linux system. All results in table 7.1 are given relative to the total CPU usage of the whole robot application. The first notable result is the immense cost of logarithm calculation used for entropy estimation. For this reason a look-up table for entropy estimation is introduced which uses linear interpolation between two sampled points. This table reduces relative CPU payload by 5%.

	without look-up table [%]	**with look-up table [%]**
occupancy grid map	11.5	14.22
entropy grid map	32.95	25.12
total	47.52	42.3

Table 7.1: Comparison of relative CPU payload using a look-up table for entropy approximation which is determined by *Valgrind*.

An important performance measure is runtime of the software on a real robot placed on the soccer field. In addition, two poles are placed on the field, with observation of these triggers the use of the camera sensor model. On the real robot hardware the absolute CPU payload is obtained by the Linux tool *top*. The measurement is not as accurate as profiling with *Valgrind*, but gives a good tendency of the real CPU usage. The result in table 7.2 shows that absolute CPU usage is reduced by approximately 9% by using a look-up table for entropy estimation. It can additionally be seen that the final absolute CPU usage of this approach is around 35%.

Further the analysis with *Valgrind* and *top* shows that the use of active gaze behavior does not have any measurable effects on the CPU usage because the behavior generates a new head movement command approximately twice times a second

	minimum [%]	**maximum [%]**	**mean [%]**
without grid mapping	48	56	52
without look-up table	84	90	87
with look-up table	76	82	78

Table 7.2: Comparison of absolute CPU payload on real robot hardware which is determined with *top*.

[1] Framework for individual analysis tools: `http://valgrind.org/`

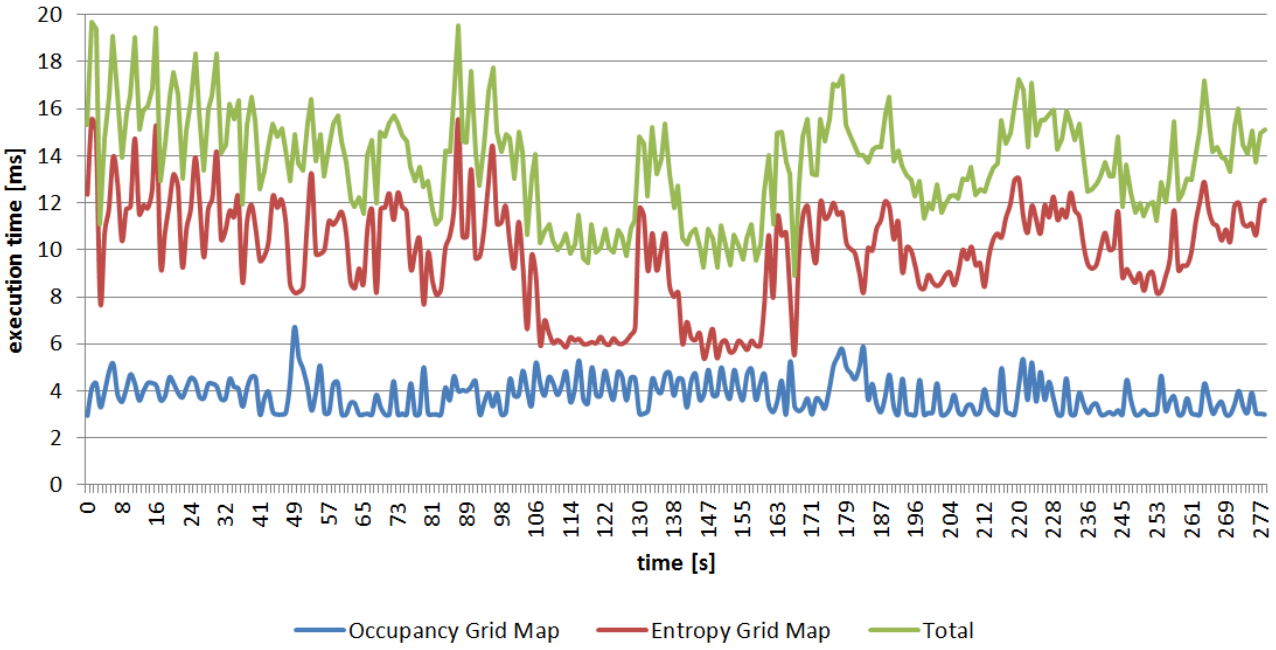


Figure 7.1: The execution time of occupancy and entropy grid map on a real robot while playing soccer.

In another experiment the execution times of the approach is analyzed with a robot playing soccer. This experiment provides information about the real CPU usage in different situation during a soccer game in the RoboCup scenario. The resulting graph is shown in figure 7.1.

Around the first half *(time < 160s)* in this experiment the robot is instructed to take the goalie role. Starting at the middle of the soccer field the goalie walks into its own half and successfully clears the own penalty area for 100 seconds. Afterwards the goalie positions itself in the goal. While the goalie does not move in the goal, the computational costs of entropy grid map are reduced noticeable. This effect is resulting from the executed head movement which stares on the ball. For this reason only a minimal region is explored, so the entropy can be computed faster. The short peak after 130s is caused by repositioning of the robot in the goal. In the second half in this experiment *(time > 160s)* the robot behavior is changed to kicking the ball into the opponent goal.

A summary of this experiment is provided in table 7.3. The approximate CPU usage of the occupancy and entropy grid map generation components is determined by sampling execution times. Here, the approximative total CPU utilization by the implemented approach is 37% which is very close to the result determined with *top*.

	minimum [ms]	**maximum [ms]**	**mean [ms]**	**approx. CPU usage [%]**
occupancy grid map	2.95	6.7	3.8	10.35
entropy grid map	5.4	15.6	9.9	26.92
total	8.9	19.68	13.7	37.3

Table 7.3: Execution times and approximated absolute CPU usages of every component.

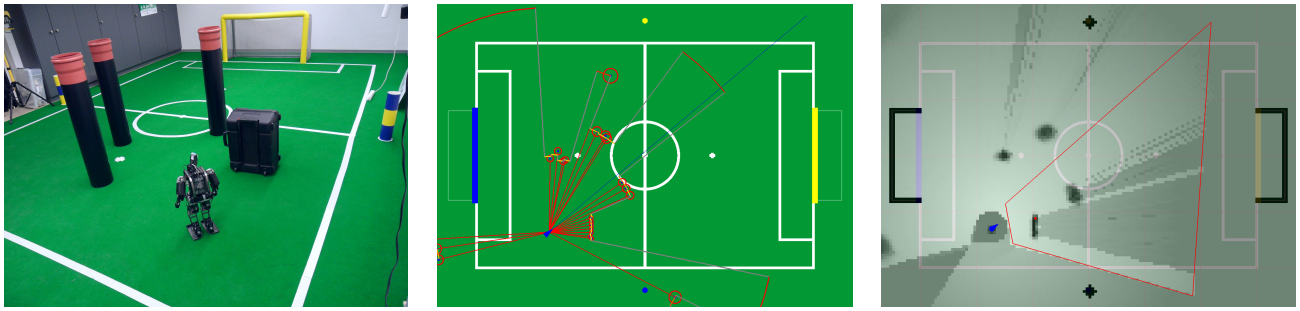
Concluding it can be observed that the implemented approach runs in realtime without overloading the CPU. Thus the occupancy grid map and active gaze control is applicable using the real robot hardware.

7.2 Validity of Occupancy Grid map

In this section a simple validation should show that the map is consistent and robust. For this validation the figure 7.2 is regarded in the following.

Three poles and a box which can be seen in figure 7.2a are given as ground truth. The first challenge is to map the observations of these obstacles into the occupancy grid map. Obviously these objects are mapped correctly in figure 7.2c. As the sector model does not use any filtering, all learned data in the occupancy grid map must be also found in the sector model which is obviously fulfilled.

On the other side no phantom obstacle should be provided in the model. The sector model (see 7.2b) does not fulfill this property, because at least two sectors are defined as occupied, although there is no obstacle in the ground truth. In contrast, the occupancy grid does not classify these regions as occupied, so it provides a consistent model of the surrounding objects. It shows that the occupancy grid map is robust against single failure detections of obstacles.



(a) Ground truth. **(b)** Resulting sector model. **(c)** Learned occupancy grid map.

Figure 7.2: Comparison of sector model and occupancy grid map.

7.3 Accuracy of Occupancy Grid map

In this experiment the discretization and obstacle modeling error of sector model and occupancy grid mapping should be analyzed. There are to many possible failures such as falling or occlusion of other obstacles around the soccer field which disturb the measurements, this experiment is only performed with a robot which is not moving. Additional these failures are non-deterministic, so the results of this experiment would become not reproducible.

In this experiment three measurements are recorded. The pole is placed relative to the robot which is outlined in table 7.4.

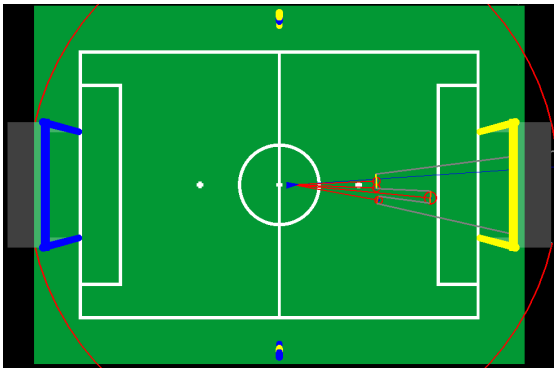
	experiment 1	**experiment 2**	**experiment 3**
Δx [mm]	1500	2500	1500
Δy [mm]	0	0	1000

Table 7.4: The pole positions relative to the robot in the experiments.

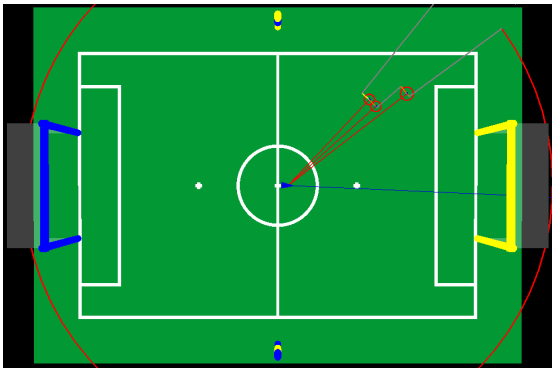
In figures 7.3c and 7.3d the resulting sector models of experiment 1 (see figure 7.3a) and 3 (see figure 7.3b) are shown which can be compared to the corresponding learned occupancy grid map in figures 7.3e and 7.3f. A sample containing the analyzed state of the occupancy map and sector model is taken every 500ms. For this purpose the binary state of every cell in the occupancy grid map is determined by thresholding its occupancy value. For this experiment a cell is treated as occupied when its occupancy value exceeds 0.55. The cell is treated as free when this value falls below 0.45. For every cell state

(a) Setup of experiment 1 in lab.

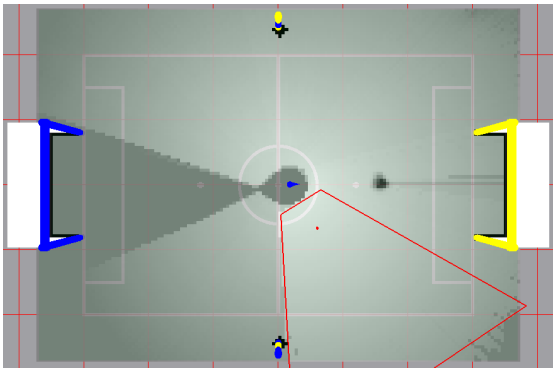
(b) Setup of experiment 3 in lab.



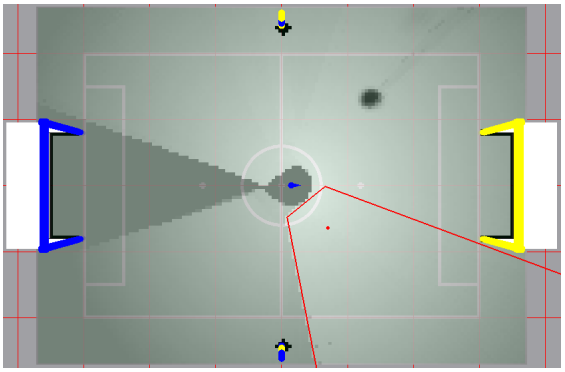
(c) Resulting sector model of experiment 1.



(d) Resulting sector model of experiment 3.



(e) Resulting occupancy grid map of experiment 1.



(f) Resulting occupancy grid map of experiment 3.

Figure 7.3: Accuracy experiment 1 and 3 of occupancy grid map.

the error is estimated by computing the distance to the real obstacle position. The error for the sector model can be obtained by considering the representatives of each occupied sector (see Section 4.2) and compute their distance to the obstacle. The radius of the object is subtracted from the computed error. In table 7.5 the standard deviation σ of all recorded errors is shown.

	experiment 1	**experiment 2**	**experiment 3**
σ in sector model [mm]	227.92	227.2	383.0
σ in occupied state [mm]	33.0	50.69	192.55
σ in free state [mm]	40.22	44.69	43.9

Table 7.5: Standard deviation in obstacle modeling accuracy.

The mean σ in the occupancy grid map is much smaller. This is also the case when errors in estimation of occupied and free state are considered. The large errors in the sector model are caused by representing the environment by sectors, which results in obstacle size getting overestimated. The occupancy grid map takes advantages of outlier filtering and using a probabilistic sensor model. The missing filter capability and too large obstacle approximation of the sector model is observed during experiments which can also be seen in figure 7.3. In experiment 3 the standard deviation σ increases significantly. It is assumed that these high standard deviations are produced by wrong synchronization of timestamps of camera and read out joint values timestamps. This result shows that the occupancy grid map provides a more accurate representation of the environment than the sector model. It additionally provides information about space that is estimated to unoccupied by obstacles.

7.4 Observation Coverage with Active Gaze Control

In this experiment the observation capability of the implemented approach is evaluated. In all experiments the ball validity and occupancy grid map state are analyzed. A cell is considered when its occupancy value differs from the default value (0.5). The percentage of observed cells of the soccer field and of observed cells in a cone of 120 degree and 1m in front of the robot is recorded.

Two experiments are performed. In the first experiment an obstacle and a ball are placed randomly on the soccer field while the robot is placed on the center circle observing the scenario (see figure 7.4).

In this scenario three head behaviors are compared: active gaze control, localization and ball tracking. Ball tracking head movement is already described in Section 4.4 which is usually used for tracking the ball in dribbling scenarios and is the most used head behavior. The localization head behavior provides camera head movement with a predefined trajectory for maximum environment exploration. All three head behaviors are activated for 60s. In this time period a measurement is sampled every 500ms. The mean of sampled data is shown in table 7.6.

The results shows that ball tracking has the best ball model validity and localization provides the best overall observation. As both head behaviors are specialized on a single task this result is not surprisingly. But both approaches does not consider the observation state in front of the robot, for which reason the active gaze control is the best approach here. Obviously, there is a trade-off between tracking the ball and observation of the soccer field which can be obtained in the results of localization and ball tracking head behaviors. Considering this property, active gaze control provides a compromise between map exploration and ball tracking.

Figure 7.4: Setup of the first observation coverage test.

experiment 1	**active gaze control**	**localization**	**ball tracking**
ball model validity [%]	79.99	71.98	99.26
soccer field observed [%]	75.02	87.69	41.45
robot's front observed [%]	99.64	92.33	75.38

Table 7.6: Observation coverage when the robot stands at the center circle.

The second experimental setup consists of a ball placed in the middle of the soccer field, two randomly placed obstacles and a robot walking a rectangular trajectory around the ball. As the ball tracking behavior is the most used head behavior during a soccer game, here the look at ball head behavior which is similar to ball tracking is compared to active gaze control. The summarized results are given in table 7.7.

experiment 2	**active gaze control**	**look at ball**
ball model validity [%]	79.96	97.01
soccer field observed [%]	75.31	53.33
robot's front observed [%]	100.00	45.02

Table 7.7: Observation coverage when the robot moves around the soccer field.

In this experiment the advantages of using active gaze control for map exploration and obstacle avoidance can be seen. The environment uncertainties are decreased through the use of the implemented active gaze control system. The space in front of the robot is always explored, meeting the reported issue of staring at the ball in is resolved.

In conclusion the implemented approach handles the trade-off in exploration and ball tracking well.

7.5 Inhibition of Return

This experiment is used to verify that active gaze control does not start staring at a single object of interest, a problem known to happen in gaze control approaches. Here, the same environmental setup is used as in experiment 1 in Section 7.4 (see figure 7.4).

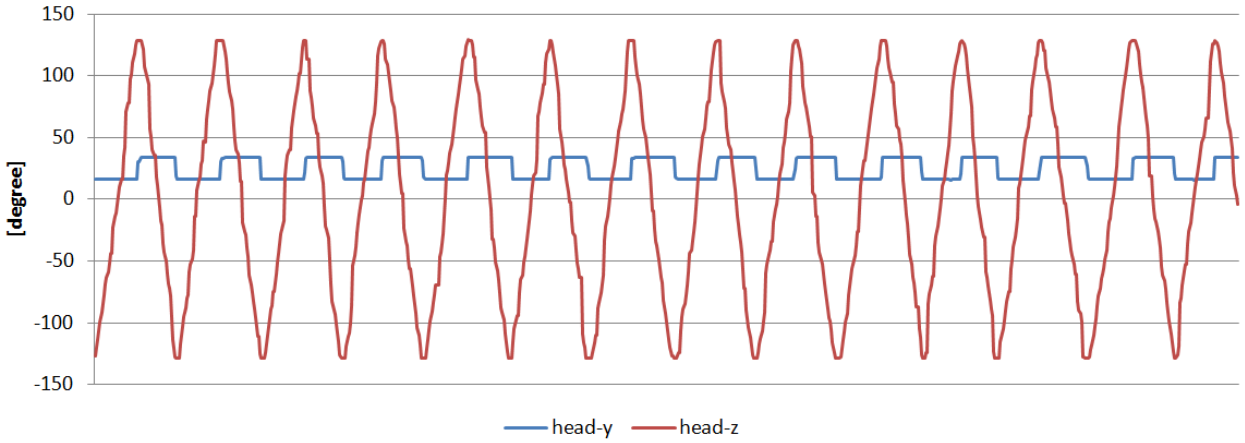
Overall three experiments are performed:

- Head movement with fixed trajectory system used so far
- Active gaze control without FOVE
- Active gaze control with FOVE

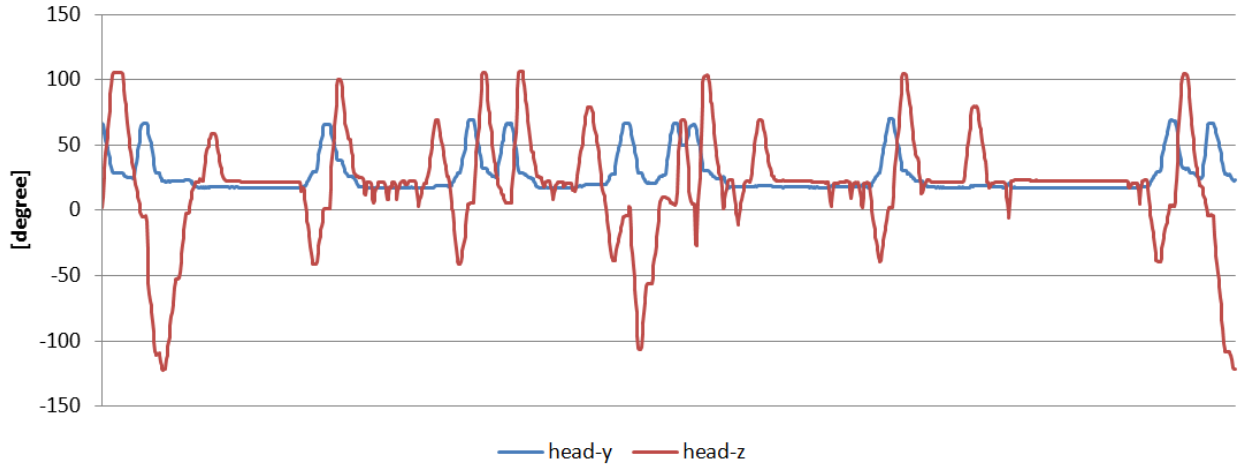
The resulting trajectories are illustrated in figure 7.5. Figure 7.5b shows the that active gaze control without FOVE has the tendency to stare in a specific direction. In this case it is attracted by the ball. The generated head movements are very unsteady with short movement paths. When FOVE is activated the active gaze control changes its attention continuous without staring at a specific location (see figure 7.5c). The resulting trajectory is similar to the fixed trajectory of localization head movement in figure 7.5a.

A worst case scenario happens when the ball position estimate is wrong and no further observation of the ball are available, even when the estimated ball position is in the image. In this case the ball model will not reject its estimate because of there is a possibility that the ball is hidden by an obstacle. The active gaze control will further attend the estimated ball position while the validity of ball model is still decreasing which increases additional the entropy of the estimated ball position. For this reason the behavior control system has to switch off the active gaze control and choose a recent head movement behavior which is able to relocate the ball. It can be concluded that the implemented approach

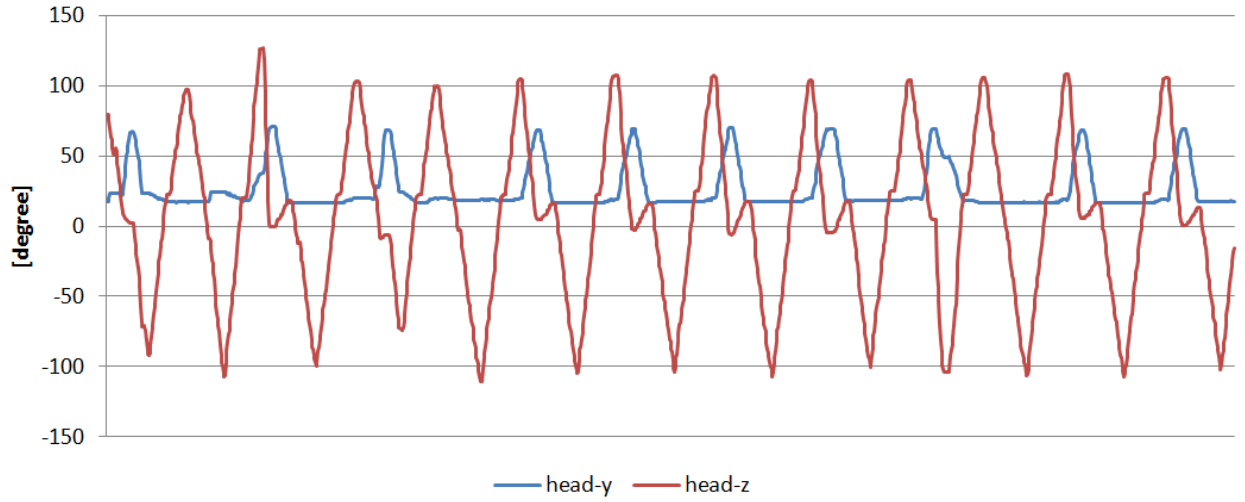
is not providing IOR perfectly, but in absence of wrong ball localization this approach works already satisfactorily.



(a) Head movement with fixed trajectory (localization mode).



(b) Active gaze control without FOVE.



(c) Active gaze control with FOVE.

Figure 7.5: Comparison of fixed head trajectory with active gaze control.

8 Conclusions and Outlook

In this work a modular framework for obstacle grid mapping and gaze control designed for autonomous humanoid soccer playing robots is introduced. It provides implementations for generic data aggregation in grid maps. These maps can also be used for other purposes than occupancy grid mapping which is demonstrated in this work by additionally using an entropy grid map for active gaze control. Due to the modular implementation it is possible to extend the framework for other applications or for using other data sources.

The implemented grid-based approach is evaluated in Chapter 7 and is found to be more accurate and consistent than the previously used sector-based obstacle model. It further provides additional information about free space which can be used for more robust path-planning and playing strategies.

In addition, a light-weight entropy-based active gaze control system used for improving the perceptual input of the robot is realized. It uses a particle-based entropy maximizer, identifying the region with potential maximum information gain and considering several sources of information as well as head movement cost. Finally, a new behavior is implemented which controls the head movements according to the computed best gaze direction.

The evaluation of the active gaze control system shows the capability of the system to deal well with the trade-off between world exploration and ball tracking. It resolves the reported issue of approaching the ball without considering the region in front of the robot (see Section 4.4). Furthermore, the experimental results shows that the generated head movement commands are supporting the occupancy grid map building. Additionally, Inhibition of Return (IOR) is considered, to prevent failures of the active gaze control system.

All mentioned functionality is implemented and fully integrated into the software based on RoboFrame for the Darmstadt Dribbler's humanoid robot HR30. The experiments show that both approaches can be computed in realtime and are applicable on the real robot hardware.

8.1 Further Improvements

As shown in Section 7, the implemented approach provides better results than the previously used obstacle estimation approach. Still, there are some possibilities to optimize runtime performance. Specifically, this could happen by analyzing if frequent used computational sequences can be cached. The *GridMap-Drawing* modules (see Section 6.1.4) could be calculated up to 50% faster by exploiting the symmetric properties of the drawn shapes. Frequently used mathematical functions such as $atan2$ can be provided by tables, improving computational efficiency.

The generation of both maps and the head movements depend on a large number of parameters. Currently all these parameters are determined by manual experimentation, but it is possible to improve these parameters using ground truth recording systems and optimization, potentially improving overall system performance.

In future work it is possible to determine a more accurate error estimation of camera sensor model (see Section 6.1.6) which should increase the accuracy of the occupancy grid map. For this purpose a large scaled series of measurement can be performed whose result allows to approximate a measurement error function. Furthermore the linear aging methods could be replaced by non-linear models. For these potential improvements, runtime efficiency has to be considered.

A system for opponent modeling currently is in development and can be added to the presented approach. In addition, the robots are able to exchange their maps among each other via wireless communication. Here, it must be considered that increasing the map's resolution also raises the traffic, for which reason a trade-off between the map's accuracy and total traffic has to be found. For this purpose

a down- and up-sampling of maps could be implemented which enables to exchange much smaller maps between robots while keeping local map accuracy.

Furthermore the introduced *MapObject* (see Section 6.1.5) can be extended to predict future states of each object based on a system model. This feature could allow the active gaze control system to compute more adaptive head movements and gain more information with them.

Consideration of the planned path for active gaze control could improve obstacle avoidance. Furthermore, the active gaze control system does not consider the robot's body, generating gaze points where the resulting image contains primarily parts of the body such as the shoulder. As countermeasure a cost can be introduced, excepting these regions for potential gaze direction.

When further computational resources are available, it is possible to extend the best gaze direction searching problem to trajectory search space. In the first step the entropy of all discovered cells during the movement to the new gaze point can be considered. Afterwards it is possible to plan a single trajectory, visiting several gaze points at once.

All landmarks currently have the same priority for active gaze control. To facilitate better performance the self-locator could be extended to provide a priority for each landmark. If the prioritization of the landmarks is chosen in way that the most important landmark assures the best reduction of localization uncertainty, the active gaze control should be able to support the self-localization process more efficiently.

9 Appendix

9.1 Cell Method Requirements

The cell type itself must provide following methods while *T* is the internal used data type which is usually *float*. All values describe normally a probability value.

- ***void setValue(T val)*:** Sets cell to specified value.
- ***void update(T val)*:** Implements update rule in which val can be especially in another number domain e.g. log odds.
- ***void updateValue(T val)*:** Implements update rule.
- ***void updateFree(T val)*:** Implements update rule, if cell is classified as free.
- ***void updateOccupied(T val)*:** Implements update rule, if cell is classified as occupied.
- ***T getValue() const*:** Returns the current cell value.
- ***bool isOccupied() const*:** Returns the cell's binary occupancy state.
- ***bool isFree() const*:** Returns the cell's binary free state.
- ***void doAging(unsigned long diff, float ageParam)*:** Implements the handle of cell's value aging in dependence of time difference and an aging parameter.
- ***void resetGridCell()*:** Resets cell to initial state.
- ***T getInitialValue()*:** Gets cell's initial value.
- ***void setUpdateIndex(int updateIndex)*:** Sets the cell's update index which indicates if cell is updated in current iteration.
- ***int getUpdateIndex()*:** Gets cell's update index.

9.2 Generalized Bresenham Line Algorithm

This algorithm can be found among under `http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm`.

Algorithm 4 Generalized Bresenham($x0, x1, y0, y1$)

1: $boolean\ steep := abs(y1 - y0) > abs(x1 - x0)$
2: **if** $steep$ **then**
3: $swap(x0, y0)$
4: $swap(x1, y1)$
5: **end if**
6: **if** $x0 > x1$ **then**
7: $swap(x0, x1)$
8: $swap(y0, y1)$
9: **end if**
10: int Δx := x1 - x0
11: int Δy := abs(y1 - y0)
12: float $error$:=0
13: float Δerr := Δy / Δx
14: int $ystep$
15: int y:=$y0$
16: **if** $y0 < y1$ **then**
17: $ystep$:=1
18: **else**
19: $ystep := -1$
20: **end if**
21: **for** $x = x0$ to x1 **do**
22: **if** steep **then**
23: $plot(y, x)$
24: **else**
25: $plot(x, y)$
26: **end if**
27: error:=error+Δerr
28: **if** error$\geq$0.5 **then**
29: y:=$y + ystep$
30: $error$:=$error - 1.0$
31: **end if**
32: **end for**

List of Figures

List of Tables

List of Algorithms

Glossary

CPU short form for central processing unit. 59, 60

FOVE short form for field of view entropy. 55, 56, 64

GUI short form for graphical user interface. 52
GUID short form for global unique identification number. 45, 49

IOR short form for Inhibition of Return. 28, 35, 36, 65, 67

LIDAR short form for light detection and ranging. 25, 26, 34, 41

RoboFrame Application framework for heterogeneous robotic platforms. 33, 67
RoboGUI GUI framework for RoboFrame. 33

SLAM short form of Simultaneous Localization And Mapping. 33, 35

XABSL Extensible Agent Behavior Specification Language. 20, 30, 33

Bibliography

[1] RoboCup Soccer Humanoid League Rules and Setup for the 2010 competition in Singapore. 2010. Available online: http://www.tzi.de/humanoid/pub/Website/Downloads/HumanoidLeagueRules2010.pdf.

[2] Brad Hall Brock White Benjamin Vance Claude Sammut David Claridge Hung Nguyen Jayen Ashar Maurice Pagnucco Stuart Robinson Yanjin Zhu Adrian Ratter, Bernhard Hengst. rUNSWift team report 2010, 2010. Only available online: http://www.cse.unsw.edu.au/~robocup/.

[3] Tim Laue Andreas Seekircher and Thomas Röfer. Entropy-based Active Vision for a Humanoid Soccer Robot. RoboCup International Symposium 2010, 2010.

[4] M. Begum, G.K.I. Mann, R. Gosine, and F. Karray. Object-and Space-based Visual Attention: An Integrated Framework for Autonomous Robots.

[5] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 476–481. Citeseer, 2000.

[6] C. Coué, C. Pradalier, C. Laugier, T. Fraichard, and P. Bessière. Bayesian occupancy filtering for multitarget tracking: an automotive application. *The International Journal of Robotics Research*, 25 (1):19, 2006.

[7] E. Orfanoudakis A. Paraschos E. Vazaios N. Spanoudakis N. Vlassis M. G. Lagoudakis E. Chatzilaris, I. Kyranou. Kouretes 2010 spl team description paper, 2010. Only available online: http://www.tzi.de/spl/pub/Website/Teams2010/Kouretes.pdf.

[8] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of robotics and automation*, 3(3):249–265, 1987.

[9] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6): 46–57, 1989.

[10] A. Elfes. Dynamic control of robot perception using stochastic spatial models. In *International Workshop on Information Processing in Mobile Robots*, 1991.

[11] A. Elfes. Occupancy grids: A stochastic spatial representation for active robot perception. Autonomous Mobile Robot, 1991.

[12] A. Elfes. Dynamic control of robot perception using multi-property inference grids. *IEEE Journal of robotics and automation*, pages 2561 –2567 vol.3, 12-14 1992.

[13] D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 25(3-4):195–207, 1998.

[14] M. Friedmann, K. Petersen, and O. v. Stryk. Adequate motion simulation and collision detection for soccer playing humanoid robots. In *Proc. 2nd Workshop on Humanoid Soccer Robots at the 2007 IEEE-RAS Int. Conf. on Humanoid Robots*, Pittsburgh, PA, USA, Nov. 29 - Dec. 1 2007. URL http://www.humanoidsoccer.org/ws07/program.html.

[15] M. Friedmann, J. Kiener, S. Petters, H. Sakamoto, D. Thomas, and O. von Stryk. Versatile, high-quality motions and behavior control of a humanoid soccer robot. *International Journal of Humanoid Robotics*, 5(3):417–436, September 2008.

[16] R. HoseinNezhad, B. Moshiri, and M.R. Asharif. Sensor fusion for ultrasonic and laser arrays in mobile robotics: a comparative study of fuzzy, Dempster and Bayesian approaches. In *Proceedings of IEEE Sensors 2002*, volume 2, pages 1682–1689, 2002.

[17] A.H. Jazwinski. *Stochastic processes and filtering theory*. Academic Pr, 1970.

[18] M. Jun and R. D'Andrea. Probability map building of uncertain dynamic environments with indistinguishable obstacles. In *American Control Conference, 2003. Proceedings of the 2003*, pages 3417–3422, 2003.

[19] G.W. Kim and B.H. Lee. Hierarchical Sensor Fusion for Building an Occupancy Grid Map using Active Sensor Modules. In *International Joint Conference SICE-ICASE, 2006*, pages 2600–2605, 2006.

[20] Stefan Kohlbrecher. A scalable, platform-independent slam system for urban search and rescue. Master's thesis, Technische Universität Darmstadt, Department of Computer Science, 2009.

[21] Stefan Kohlbrecher and Oskar von Stryk. Modeling Observation Uncertainty for Soccer Playing Humanoid Robots.

[22] D. Kortenkamp, R.P. Bonasso, and R. Murphy. *Artificial intelligence and mobile robots: case studies of successful robot systems*. MIT Press Cambridge, MA, USA, 1998.

[23] Martin Lötzsch. XABSL - a behavior engineering system for autonomous agents. Diploma thesis. Humboldt-Universität zu Berlin, 2004. Available online: `http://www.martin-loetzsch.de/papers/diploma-thesis.pdf`.

[24] Tobias Ludwig. Hindernis- und spielererkennung für humanoidroboter beim robocup. Master's thesis, TU Darmstadt, FB Informatik, 2006.

[25] S. Kohlbrecher K. Petersen S. Petters K. Radkhah M. Risler D. Scholz D. Thomas M. Friedmann, T. Hemker and O. von Stryk. Team Description for Humanoid KidSize League of RoboCup 2010.

[26] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *1985 IEEE International Conference on Robotics and Automation. Proceedings*, pages 116–121, 1985.

[27] H.P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI magazine*, 9(2):61, 1988.

[28] D. Murray and C. Jennings. Stereo vision based mapping and navigation for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 1694–1699. Citeseer, 1997.

[29] S. Petters, D. Thomas, and O. Von Stryk. Roboframe-a modular software framework for lightweight autonomous robots. In *Proc. Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware of the 2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. Citeseer, 2007.

[30] M. Risler and O. von Stryk. Formal behavior specification of multi-robot systems using hierarchical state machines in XABSL. In *AAMAS08-Workshop on Formal Models and Methods for Multi-Robot Systems,(Estoril, Portugal)*. Citeseer, 2008.

[31] Thomas Röfer, Tim Laue, Judith Müller, Oliver Bösche, Armin Burchardt, Erik Damrose, Katharina Gillmann, Colin Graf, Thijs Jeffry de Haas, Alexander Härtl, Andrik Rieskamp, André Schreck, Ingo Sieverdingbeck, and Jan-Hendrik Worch. B-human team report and code release 2009, 2009. Only available online: `http://www.b-human.de/download.php?file=coderelease09_doc`.

[32] A. Saotti and K. LeBlanc. Active perceptual anchoring of robot behavior in a dynamic environment. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3796–3802. Citeseer, 2000.

[33] JF Seara, O. Lorch, and G. Schmidt. Gaze control for goal-oriented humanoid walking. In *Proceedings of the IEEE/RAS International Conference on Humanoid Robots (Humanoids), Tokio, Japan*, pages 187–195. Citeseer, 2001.

[34] C.E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[35] Michael Kruse Stefan Czarnetzki, Sören Kerner. Real-time Active Vision by Entropy Minimization Applied to Localization. RoboCup International Symposium 2010, 2010.

[36] P. Stepan, M. Kulich, and L. Preucil. Robust data fusion with occupancy grid. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 35(1):106–115, 2005.

[37] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[38] S. Vijayakumar, J. Conradt, T. Shibata, and S. Schaal. Overt visual attention for a humanoid robot. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, volume 4, pages 2332–2337. Citeseer, 2001.

[39] T. Xu, Q. Muhlbauer, S. Sosnowski, K. Kuhnlenz, and M. Buss. Looking at the surprise: Bottom-up attentional control of an active camera system. In *10th International Conference on Control, Automation, Robotics and Vision, 2008. ICARCV 2008*, pages 637–642, 2008.