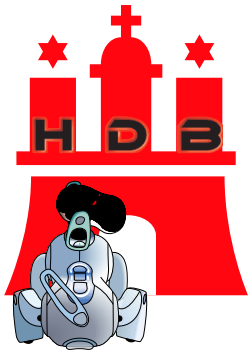

Hamburg Dog Bots Team Report 2005

Universität Hamburg
MIN-Fakultät/Informatik
Arbeitsbereich Technische Informatiksysteme
Vogt-Kölln-Strasse 30
22527 Hamburg, Germany
http://www.hamburgdogbots.de
Team Leader: Birgit Koch
(koch@informatik.uni-hamburg.de)

Nils Bertling, Joachim Dubber, Sonia Haiduc, Birgit Koch,
Valerij Krichevskyi, Matthias Niess, Jonas Reese, Peter Roßmeyer,
Janis Schönefeld, Gunnar Selke, Benjamin Seppke, Malte-Nils Sörensen

Abstract

In this report we give an overview over the research and the team of the Hamburg Dog Bots. The team is composed of undergraduate and graduate students of the Universität Hamburg and Technische Universität Hamburg-Harburg. We participated for the second year the SONY Fourlegged league, but were using ERS-7 SONY Aibo robots for the first time. The report covers the history of the team, our approaches to the technical challenges, as well as a brief description of the relevant topics of research and the student research papers in the year 2005.

Contents

1 Team Development **3**
- 1.1 Team Members . . . 3
- 1.2 Team History . . . 3

2 Challenges **4**
- 2.1 Variable Lighting Challenge . . . 4
- 2.2 Free Challenge . . . 6
- 2.3 Almost SLAM Challenge . . . 6

3 Topics of Research **7**
- 3.1 Low-level Vision . . . 7
- 3.2 Behavior . . . 8
- 3.3 Motion . . . 8
- 3.4 Reasoning with Uncertainties . . . 8
- 3.5 Self Localization . . . 8
  - 3.5.1 Motivation . . . 8
  - 3.5.2 The Changes to the Particle Filter . . . 11
  - 3.5.3 Lines and Corners instead of Points . . . 15

4 Student Research Papers **18**

5 Conclusion and Future Work **19**

6 Acknowledgements **20**

A Student Research Papers (some only available in German) **21**

Chapter 1

Team Development

1.1 Team Members

PhD students: Birgit Koch (Team Leader)
Undergraduate and graduate students: Nils Bertling, Joachim Dubber, Sonia Haiduc, Valerij Krichevskyi, Matthias Niess, Jonas Reese, Peter Roßmeyer, Janis Schönefeld, Gunnar Selke, Benjamin Seppke, Malte-Nils Sörensen

1.2 Team History

In 2003 the Universität Hamburg, Germany, received some Sony Aibo Robots. A project with the intention to participate in the RoboCup Sony Four-Legged League 2004 was started. Essential part of the project is the integration of engineering research solutions from application areas that are part of the Hamburg Institute of Information technology (MIN-Fakultät/Informatik, Universität Hamburg) and develop these solutions for further research. The cooperative work of students with different orientations in the information technology is specific for this project and unlike to typical projects with tight focus on a specific topic. In September 2004 some students of the Technische Universität Hamburg-Harburg joined the Hamburg Dog Bots team. The cooperation of both groups implicated new requirements for the team. The members of the Hamburg Dog Bots decided to switch from the Hamburg Dog Bots Code 2003 to the German Team Code 2004 and are now working with the new Sony Aibo ERS-7 robots. The German Open 2005 in Paderborn was the first competition under these new conditions and a test for the world championship in Osaka in 2005.

Chapter 2

Challenges

2.1 Variable Lighting Challenge

As the visual cognition of the robot is based on color tables, which tell the robot about the interpretation of a percepted pixel, an environment with changing lighting conditions is quite hard to handle. Color tables are static and unique to an environmental setting. Color tables are not independent of any changes of lighting and white balances. If you have a color table which fits for the given environmental settings, there are a lot of benefits. For example, the color classification is very fast. In the robotic soccer domain, speed is a substantial argument. We decided not to use a single, but multiple color tables for the variable lighting challenge. Experiments have shown that color classification with a "standard color table" becomes unsuitable, if the lighting conditions are going to be darker than under normal conditions. If lightning becomes brighter, we found out that the influence of the colors classification using the "standard color table" is much lower. We decided to use two color tables, one for normal lighting of the field and one for darker lighting conditions. Additionally we created a software module, which allows the robot to recognize the ambient lighting conditions - and finally to choose which color table to use for getting optimal results in classifying the colors percepted. As an option, not only the color tables, but also the camera parameters (e.g. white balance, shutter speed and gain factor) can be changed by the robot if necessary. The algorithm requires to be robust against outlier images, which are darker even if the environment itself is still bright, and needs to react with an acceptable latency. The main idea behind measuring the brightness of the environment is simple: Use the Y-channel of the image, which describes the brightness. To be protected against outlier pixel (within a single image) we performed an averaging over a grid of pixels. Sometimes, the robot looks at the ground, which looks dark to the robot, so there is a need for averaging over some images (or over time). After these two steps of smoothing the incoming data, the robot can reliably switch between the two color tables. We will now present the pseudo code and a state machine of the implemented algorithm, and later on, discuss several main difficulties it comes along with.

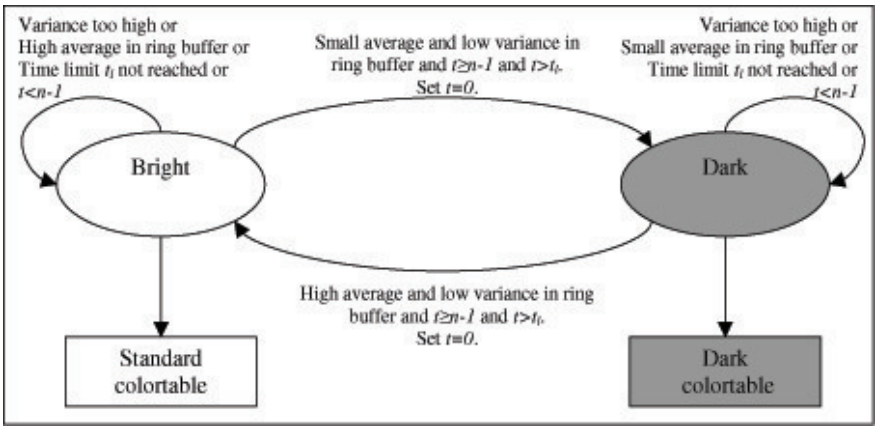

Figure 2.1: Two Color Tables for Variable Lighting Challenge

Algorithm for the variable lighting challenge using color table switching:

1. Load "standard color table".
2. Set current state to "bright".
3. Create a ring buffer of given length n.
4. Declare $t = 0$ as an iterator over the ring buffer.
5. For the first n frames: Write the averages of the relative brightness of each image (calculated with the help of the image grid) to the buffer.
6. For all other frames: Get the average and the variance of all elements of the ring buffer
   (a) If variance is too high or if time limit tl is not reached or if $t < n - 1$ then: Wait for switching the color tables.
   (b) If average is "small" and current state is "bright" then: Switch to dark color table (and adjust camera parameters). Set state to "dark". Set $t = 0$.
   (c) If average is "high" and current state is "dark" then: Switch to standard color table (and adjust camera parameters). Set state to "bright" Set $t = 0$.
   (d) For all other configurations wait for switching the color tables.

The presented algorithm ensures that judgements for switching between the color tables are only made if the ring buffer contains completely new data. Although the algorithm is quite simple, we expected some difficulties. As the presented procedure is threshold-based, the main problems occur in finding proper values for the thresholds:

- Average brightness threshold within the ring buffer
- Threshold for the variance within the ring buffer

- Size n of the ring buffer
- Time limit tl for staying in one state (should be greater than $n - 1$)

What we did, is to let the robot measure and log the percepted brightness for every frame over a long period of time while the robot was playing soccer. This was separately done for normal and dark environmental lighting settings. We did some statistical analysis on the recorded data to extract the minimal size n of the ring buffer that leads to a stable threshold decision about the environmental lighting. If n is very low, we set an additional time limit $tl = 30$, which means that the robot is only allowed to switch the color table at most once a second. This leads to more stable results, as our prior knowledge tells us that lightning conditions do not change every second. If the difference between the lightning conditions is small, it is hard to find a threshold to divide them. In that case, we can additionally use the variance inside the ring buffer to achieve more stable results. The presented algorithm shows a very effective way for the robot to detect changes in the environmental lighting. During tests in our laboratory it was possible for the robots to play soccer even if the very bright spotlights were switched off during the game. We also plan to integrate this low-level computer vision algorithm as one of the parts of our vision pipeline for feature extraction.

2.2 Free Challenge

In the free challenge, the Hamburg Dog Bots presented a commentator-system. The original intention was to build a totally autonomous system that could recognize certain game situations (e.g.: goals, fouls, ball out, etc.) and comment these situations. The purpose of the commentator-system could be to replace the human referees with Aibo robots, since players and referees are usually of the same kind. The commentator-system would also add a special feature to public presentations. Because of problems with opponent recognition (which in the case of the commentator-system would be the team recognition) and difficulties with accuracy of the line and ball recognition we had to show a prototype version at the world championship 2005 in Osaka. This prototype could read the messages from the game controller for both teams and give comments according to the messages. The outer appearance was largely hindered by the poor quality of the Aibo robot speakers in combination with the poor microphone at the competition site. This caused the comments to be mostly not understandable.

2.3 Almost SLAM Challenge

For the Almost SLAM Challenge the Hamburg Dog Bots used the self localization of the Hamburg Dog Bots soccer code augmented with a configuration solver. The self locator relied on lines, corners and combinations of lines and corners to determine the position of the Robot.

Chapter 3

Topics of Research

3.1 Low-level Vision

The key aspect of the low-level vision group activity is the real-time feature extraction based on the camera images of the (ERS-7 Aibo) robot. As this feature extraction is needed for the robot to act properly in the soccer environment it has to be performed within one perception action cycle, which leads to hard time constraints. One of the main goals of the low-level vision group is to achieve a robust enemy robot localization based on the camera image. To achieve this, a fast region-growing algorithm is executed. The seed points for this region growing are determined by a fast scan line based algorithm. After the regions are calculated, some low level vision features are extracted, e.g. the circumscribing contour, bounding box and moments. These features are stored and then used by the high-level vision group to generate appropriate enemy robot percepts. Another goal is the automatic robust color segmentation and the real-time adaptation to dynamic lighting conditions. By now, the low level vision is almost completely based on hardware color segmented images. The quality of the segmentation is critical for all following computational tasks. Segmentation tables made by humans require a lot of time, accuracy and skill. Our goal is to automatically obtain the color tables with less or even without human interaction. Another fact is that changes in the lighting conditions have great impact on the correctness of the color table. So, we research for methods to automatically adjust the color table to lighting changes.

Looking back at the hard real time constraints the low level vision has to fulfill, we are working on methods to extract details from detected features that cannot be computed in real-time using a pipeline for detail extraction. Some objects like landmarks are not very often seen by the robots but contain valuable information. The real-time feature detection cannot analyze every detail of the features. By analyzing the regions of interest over a series of perception action cycles, maximum feature extraction is possible without slowing down the perception action cycles frequency.

3.2 Behavior

Another group of students is working on high-level behavior. At the German Open 2005 the behavior module of the German Team Code 2004 was used. This module was programmed using the XABSL language developed by the Humboldt University of Berlin, which is based on XML. In XABSL, agents are described by a hierarchy of agents ("options"), in which the next decision is made by traversing a state machine. It turned out that the static nature of the state machines made it difficult to add new behavior strategies on the fly.

3.3 Motion

The removal of the surrounding walls made a re-engineering of some kicks necessary. Additionally a group of students developed new successful kicks.

3.4 Reasoning with Uncertainties

Many variables in the RoboCup domains world state are, in different aspects, uncertain. We are researching ways to exploit uncertain information, to minimize uncertainties and maximize information gathering in the areas of self-localization and sensor-control. We have achieved useful progress by augmenting the world state with uncertainties. The use of uncertain - previously ignored - variables have resulted in an improved self-localization as demonstrated in the Variable Lighting Challenge during the German Open 2005. We are currently researching means to use the uncertainties for sensor-control and maximizing information-gathering depending on the current task of the robot. Our research is focused on mathematical models, rather than machine learning.

3.5 Self Localization

The self localization of the Hamburg Dog Bots is a modified version of the original German Team code 2004 self localization. Major changes are the introduction of uncertainty values for all landmarks, and the removal of the points-table in favor of lines and corners.

3.5.1 Motivation

After the changes to the field and the positions of the landmarks, we discovered that we had a serious problem with the self localization of the robots.

Analysis of the Self localization from the viewpoint of general robotics

The self localization of the Hamburg Dog Bots is an absolute localization on a known map using fixed landmarks on known positions. The advantages of this kind of localizations are:

- The positions of all landmarks are known.

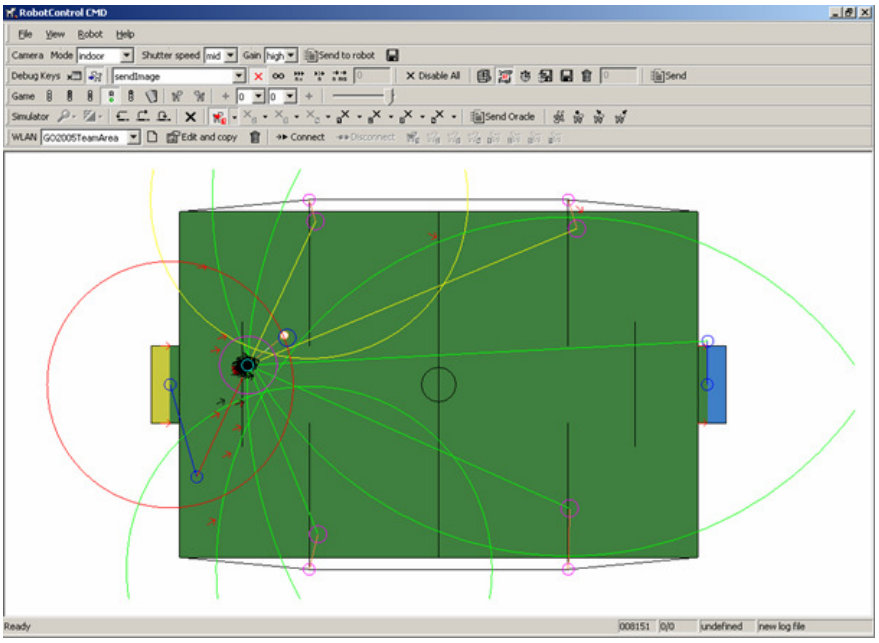

Figure 3.1: Self localizator of the Hamburg Dog Bots, the color of the lines indicate the uncertainty of the corresponding landmark percepts

- If it is possible to get the relative angles to several landmarks the position of the robot is computable.
- If a landmark is recognized the quality of the perception can be used to estimate the quality of previous perceptions of other landmarks.
- For every assumed position of the robot the relative positions of all landmarks are known.
- For every position of the robot the expected perceptions are known.
- The robot is able to search directly for the landmarks with the highest information value.

Special Features of the Landmarks in the Self Localization

The Landmarks in the Sony Four-legged League can be distinguished in two general classes. The first class (explicit landmarks) is made up of those landmarks that only serve the purpose of a landmark: the goals and the pylons. In the other class are implicit landmarks: lines, corners and robots. While explicit landmarks have a higher information value and are unique, their position makes them hard to percept in important game situations like handling the ball near the opponent goal. Perceptions of lines are almost always available in such situations but cannot be assigned without difficulties to the line they belong. Almost all configuration of lines visible in a short series of pictures taken during game play could belong to more than one position on the field.

The Pylons There are four pylons on the map. They have a unique color pattern and are cylindric which makes them easy to identify. However they don't have an orientation other than up down which can be used for the localization.

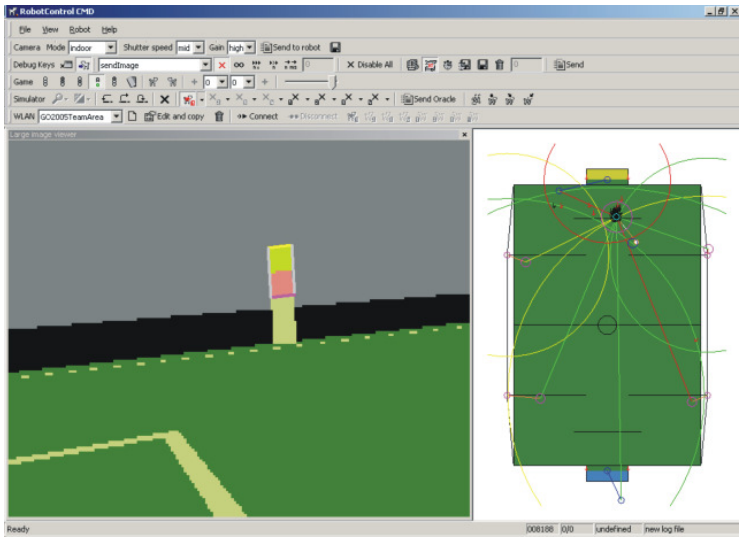

Figure 3.2: A Pylon

The pylons are tall and the distance between the robot and the pylons can be calculated thanks to the color patterns and the height of the pylons which are higher than the robots camera. The uncertainty of a pylon perception can be estimated by the number of recognized pixel belonging to the landmark. The bottoms of the pylons are also usable as landmarks.

The Goals The Goals are 20cm tall and 60cm wide. There is one yellow and one blue goal. Their size and color makes them excellent landmarks. The distance to a goal can be estimated easily, as long as the goal is not too close to the robot.

The Lines and Corners of the Field Border The lines and corners of the field border can be used to estimate the position of the robot, but only from a short distance. They are very useful for estimating the rotation of the robot.

The Circle in the Middle Field The circle on the middle field is also a unique landmark and as such very useful for the localization of the robot.

The Lines and Corners in the Penalty Area The lines of the penalty area are well suited as a landmark particularly for the goalie. The goalie is almost always able to recognize the corners of the penalty area.

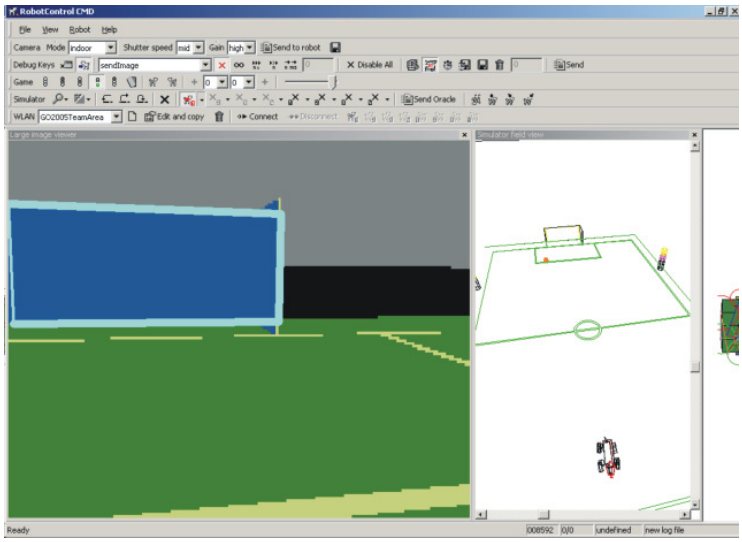

Figure 3.3: A Goal

3.5.2 The Changes to the Particle Filter

The Evaluation Function of the Particle Filter

The evaluation function of the particle filter provides a heuristic to determine which samples are worthless and need to be respawned, and which samples are good and should be kept for the computation of the position of the robot. Every solution for a subproblem of the self localization problem is a useful heuristic and every combination of these solutions is a better heuristic. To determine an optimal solution, the self localization is decomposized and solutions for all subproblems are discussed.

The Decomposition of the Self Localization Problem The complete solution of the self localization problem is to compute the position of the robot as such that for every landmark the direction and distance from the robot to the landmark is the same as from the computed position. The measured direction and the measured distance have different behaviors regarding the movement of the robot and its perception, and they have to be treated according to those. In our particle filter, samples have one quality value for the direction and one for the angle which are also used during resampling and computation of the position from all samples.

So each perception of a landmark can be used to compute the position of the robot using the direction and the distance from the robot to the landmark as well as the bearing of the landmark. The best results are obtained by using all landmarks.

The problem is to get all landmarks into the computation since the robot is not able to see all the landmarks at once with its camera and is moving most of the time. The particle filter used by the German Team code 2004 provides a solution for this problem but uses only immediate sensor inputs. The information provided by these percepts is not fully used afterwards. Our

approach is to keep this information by tracking the positions of the perceptions relative to the robot. The positions are moved according to the odometry just like the samples of the particle filter. Unfortunately our odometric sensor inputs are not of good quality. To deal with this problem we introduced two values representing the uncertainty of the distance and the direction to a landmark perception. Each landmark is augmented with these values that are also useful for sensor control. As long as the robot is just rotating (x) or translating one uncertainty value, it is almost unaffected, a fact that is very useful in some game situations.

The Evaluation Function of the Released German Team Code from 2004 The evaluation function used in the German Team Code released 2004 relies heavily on single landmark perception. After the changes to the field, there where lots of perception action cycles without landmark perceptions. The not too reliable odometry had to provide the necessary information in such situations.

The evaluation function of the German Team Code is:

$$P_{hl}(t) = \begin{cases} e^{\delta(2-(\omega_{seen}-\omega_{exp}))^2} \\ e^{\delta(\omega_{seen}-\omega_{exp})^2} \end{cases} \tag{3.1}$$

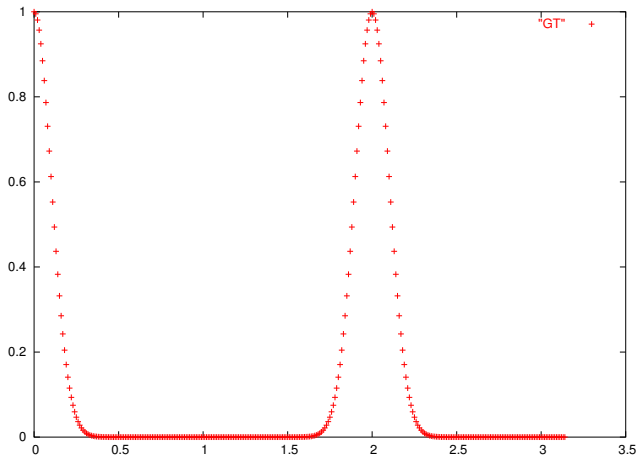

Figure 3.4: Evaluation Function of GermanTeam code 2004

Quality of a Landmark Perception Not all perceptions have the same quality. The error of the estimated distance grows with the distance. The error of the joints affects the estimated angle to a landmark. The quality of cached landmark perceptions degrades also accordingly to the error of the odometry. The perceptions still contain useful information to determine the position of the robot. The quality of a perception can be integrated into the evaluation function. The quality of the direction to a landmark given a perception is:

$$bva_l = \frac{2\pi - 2ce_{or_l}}{2\pi} \tag{3.2}$$

where ce_{or_l} is the cumulative estimated odometric error since the last sensor reading from the landmark l.

The quality of the estimated distance to a landmark is:

$$bvd_l = \frac{d_l \cdot (\frac{bvf_{dl}+bvf_{ol}}{2}) - 2ce_{ot_l}}{d_l} \tag{3.3}$$

where bvd_{dl} is the estimated quality of the distance to the landmark based on the distance from the robot to the landmark l. bvf_{ol} is the estimated quality based on computation of the image processor. ce_{ot_l} is the cumulative estimated error of the odometry regarding the translation since the last sensor reading of the landmark l. d_l is a factor to norm the bva to $[1, 0]$.

Adding the Quality of Percepts to the Evaluation Model The integration of the quality or uncertainty of the percepts into the evaluation model is after these considerations an easy task. Each Landmark has an "belief value" for the direction (bva) and the distance (bvd) computed from the sensor readings. The bvd and bva degrade when the robot moves. They are set every time the robot recognizes a landmark. The change of the quality of a sample in the particle filter given a landmark percept converges to the original quality of the sample as the "belief values" degrade.

The New Evaluation Function using Uncertainty The evaluation function based on the distance between robot and landmark takes the quality of the measured distance into account resulting in a more precise computation of the robot position. The landmark percepts in the German Team code 2004 provided basic algorithms that where appropriate to set the initial quality value of the distance measurement.

The maximum probability or quality of a distance measurement given a landmark percept is 1, in this case the measured distance is equal to the real distance. A simple measurement for the probability of a sample is $1 - na_{li}$, where na_{li} is the normalized difference of the distance between sample i and landmark l and the measured distance to the landmark l.

The new function for the evaluation of samples based on the distance between sample and landmark is:

$$P_{hl}(t) = (1 - bvd) \cdot P_h(t-1) + bvd \cdot e^{\delta(1-\frac{d_{seen}-d_{exp}}{1000})^2} \tag{3.4}$$

This function converges as the bvd degrades to the last probability of the sample. The figure shows the function given a last probability of $Ph(t-1) = 0.3$. The bvd is located on the z axis the difference between measured distance and distance from sample to landmark is located on the x axis. The computed probability $P_h(t)$ is located on the y axis.

The new evaluation function based on the direction to a landmark The new evaluation function based on the direction from a sample to a landmark is:

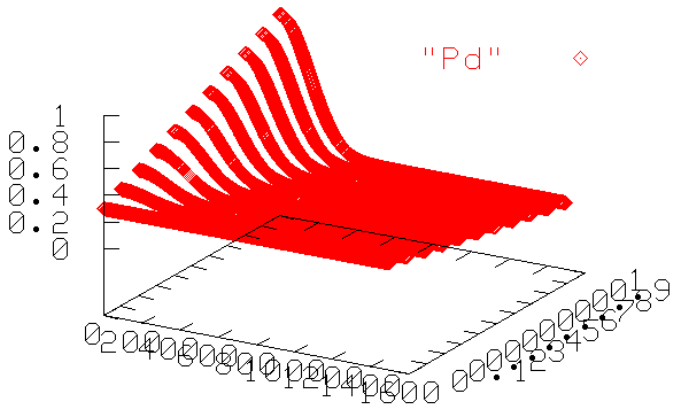

Figure 3.5: New Evaluation Function based on Distance Measurements

$$P_{hl}(t) = \begin{cases} (1 - bva) \cdot P_h(t-1) + bva \cdot e^{\delta(2-(\omega_{seen} - \omega_{exp}))^2} \\ (1 - bva) \cdot P_h(t-1) + bva \cdot e^{\delta(\omega_{seen} - \omega_{exp})^2} \end{cases} \tag{3.5}$$

This function is an extension of the original evaluation function of the German Team.

The function converges to $P_h(t-1)$ as the the bva degrades. The figure shows the function given $P_h(t-1) = 0.5$. The bva is located on the z axis. The difference between the expected angle to a landmark l from a sample h and the measured angle is located on the x-axis. On the y axis is the probability $P_{hl}(t)$ of the sample h given the landmark l.

Comparison of the New Evaluation Function and the Original Evaluation Function

Assuming that sensor readings may contain errors, the evaluation functions do not set the probability of samples but adjust them in the direction of the computed probability using the function:

$$P_h(z) = \begin{cases} P_h(z-1) + max_{up}, b_h > max_{up} \\ P_h(z-1) - max_{down}, b_h < max_{down} \\ b_h \end{cases} \tag{3.6}$$

Misreadings don't cause too much trouble using this approach so that valuable samples are not lost because of a single sensor misreading. The figure shows the evolution of the probability of a sample over 30 perception action cycles, with a landmark percept in every third cycle.

The red line represents the difference between the expected value and the measured value of the direction plus the normalized difference between the expected distance and the measured distance.

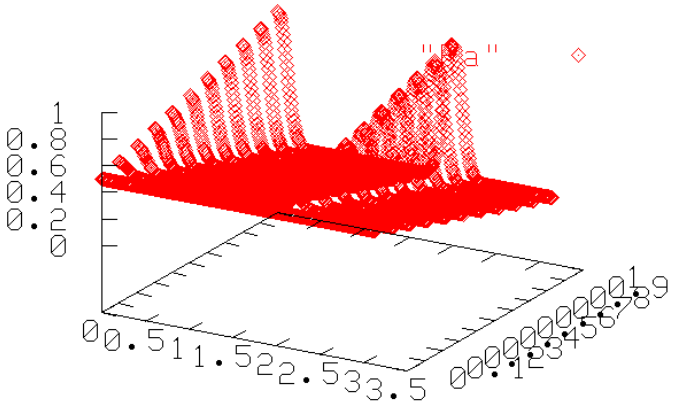

Figure 3.6: New Evaluation Function based on Direction

The green line shows the evolution of the probability of the sample using the Hamburg Dog Bots evaluation function.

The blue line shows the evaluation of the probability of the sample using the original function from the German Team code 2004.

3.5.3 Lines and Corners instead of Points

The point based evaluation of the German Team was replaced by a line and corner based localization. The main reason is that a line represents a whole range of points with two points, and can be used to compute corners. Both corners and lines have a known orientation on the field which makes them very well suited for the localization task.

The Hamburg Dog Bots use a different algorithm to find lines and corners. The algorithm uses the field line points provided by the Image Processor of the German Team. The points are fitted into a raster and are grouped to clusters. The center of mass of each cluster is used to build cluster pairs that represent line segments. The cluster pairs are not treated as landmarks but have a bva and a bvd value because they are cached like landmarks. Lines are build from cluster pairs and corners are computed from lines.

The Evaluation Function using the Lines as Single Landmarks

Lines can be used as single landmarks. Since the orientation of the lines on the field is known, it is appropriate to exploit this knowledge. The normalized orientation[1] of a lines percept based on a sample is used to evaluate the quality of the orientation of the sample.

[1] not the direction from the robot to the line

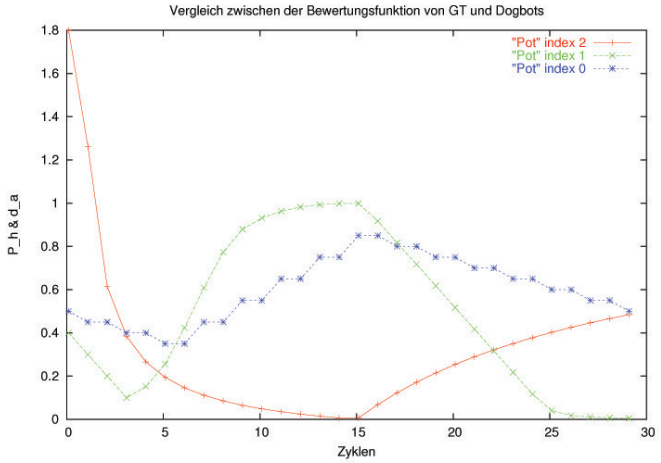

Figure 3.7: Comparison of the Original Evaluation Function and the Hamburg Dog Bots Function

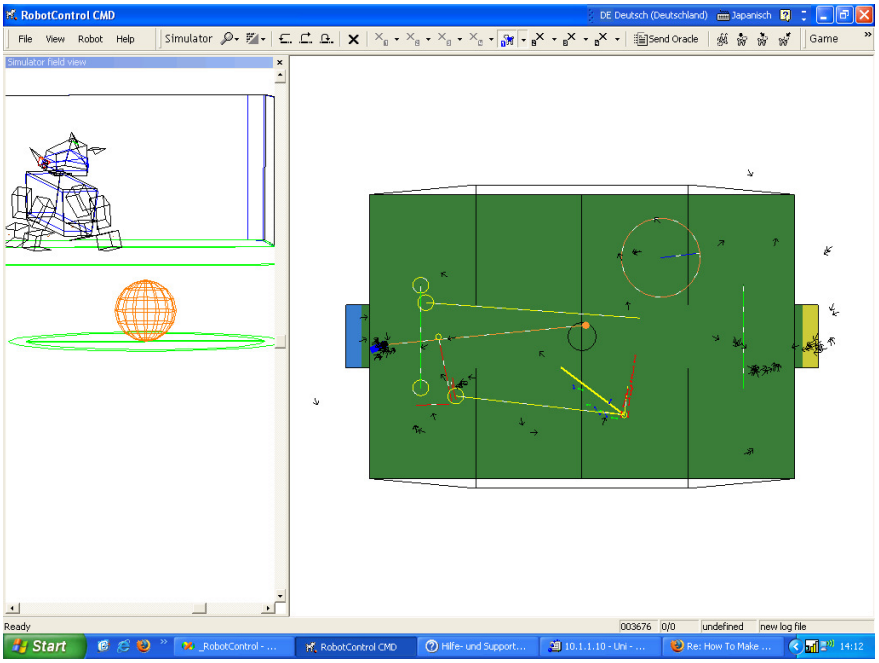

Figure 3.8: Self Localizator of the Hamburg Dog Bots, using only lines and corners

The Evaluation Function using the Corners as Single Landmarks

Corners are identified using their orientation and distance from a given sample. The identified corners are then used like pylons to evaluate the samples.

Lines and Corners as a Configuration Problem

We use the cached cluster pairs to generate lines. If the robot is not moving too much, we get many lines that are integrated into a configuration which assigns each lines percept to a field line and each corner to a corner on the field. Although, almost always, at least two configurations are possible. These configurations provide more information compared to the single landmark approach.

Chapter 4

Student Research Papers

During the project several project-, studies- and diploma-thesis evolved from different approaches:

- Jana Passow: Geometrieidentifikation aus den Kamerabildern mobiler autonomer Systeme in der Sony Fourlegged Robot League. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, June 2004.
- Peter Roßmeyer: Teamwork exemplified by the Four-Legged League in RoboCup. Studienarbeit, MIN-Fakultät/Informatik, Universität Hamburg, to appear in spring 2006.
- Valerij Krichevskyi: Analyse und Implementierung der 'Variable Lighting Challenge' der Four-Legged-Liga im RoboCup. Studienarbeit, Fachbereich Informatik, Universität Hamburg, September 2005
- Malte-Nils Sörensen: Autonomes zweibeiniges Stehen von Sony Aibo Robotern durch Einbettung von Beschleunigungssensoren. Baccalaureatsarbeit, Fachbereich Informatik, Universität Hamburg, January 2005.
- Jonas Reese: Lernsystem für die Verhaltenssteuerung in der RoboCup Four Legged League. Projektarbeit, Fachbereich Informatik, Universität Hamburg, March 2005
- Sonia Haiduc: Opponent Recognition and Localization, Part 1 and Part 2. Projektarbeit, Fachbereich Informatik, Universität Hamburg, 2005

The student research papers of 2005 are attached in the appendix of the team report.

Chapter 5

Conclusion and Future Work

In this report, we introduced our current approaches for the Sony Four legged league.

Because of the issues with the behavior as mentioned in section 3.2, we decided to substitute the XABSL based behavior module by a new module based on another agent description language, an easy to understand programming language for defining behavior developed by the Hamburg Dog Bots team.

Our overall goal is to include more collaborative behavior in future work. By means of integrating socionic and game theory the robots should play as a team instead of playing antagonism. Another aspect of future work relates to the little interest of women in the Sony Four-legged League. Besides including socionic aspects like communication and cooperation in the behavior of the soccer-playing robots we will develop new additional scenarios to foster interest of women in robotics.

Chapter 6

Acknowledgements

Our participation in the RoboCup 2005 event could not have been possible without the help from many persons all around the world. The team of the Hamburg Dog Bots is grateful to all friends, previous members, and other supporters of the team including Klaus-Dieter Florstedt and other technical support and administrative staff of the MIN-Fakultät/Informatik of the Universität Hamburg. We thank the organizers of RoboCup 2005 for travel support and the great organization of the World Cup. We greatfully acknowledge the support and cooperation of the "Behörde für Wirtschaft und Arbeit" in Hamburg, the Department "Forschung und Wissenschaftsförderung" of Universität Hamburg, the Goethe-Institut of Osaka, the International Relations Department of the City of Osaka and the employees of the Truck Terminal of Osaka, who made the journey to Osaka possible. Further, we would like to thank the German Team for providing their source code in 2004.

Our publications as well as videos and pictures are available on the Hamburg Dog Bots' webpage: http://www.hamburgdogbots.de.

Appendix A

Student Research Papers (some only available in German)

Universität der Freien und Hansestadt Hamburg
Fachbereich Informatik
Arbeitsbereich Technische Informatiksysteme

Studienarbeit

Analyse und Implementierung der „Variable Lighting Challenge" der Fourlegged-Liga im RoboCup

Valerij Krichevskyi
Matr. Nr. 4865042

9. September 2005

betreut durch
Prof. Dr.-Ing. Dietmar P. F. Möller,
Dipl.-Inform. Birgit Koch

Inhaltsverzeichnis

1. Abstract 4
2. Einleitung 6
3. Grundbegriffe 8
4. Aufgabenstellung 10
5. Farbmodelle 14
   5.1. Das RGB-Modell 14
   5.2. Das YUV-Modell 14
   5.3. Das HSV-Modell 17
   5.4. Das TSL-Modell 18
6. Analyse 21
   6.1. Testaufnahmen 21
   6.2. Bildanalyse 21
7. Farbklassifikationsmodule 30
   7.1. ColorTable32 und ColorTable64 30
   7.2. ColorTableTSL 31
   7.3. HSIColorTable 32
8. Dynamische versus statische Farbklassifikation. 34
9. Analyse, Implementation und Einbindung in den Hamburg Dog Bots-Code 35
10. Ausblick: Automatisierung der Farberkennung 38
11. Fazit 39
12. Literaturliste 40
13. Anhang 42
    13.1. What is RoboCup? 42
    13.2. The Variable Lighting Challenge 44
    13.3. Sony Four Legged Robot Football League Rule Book 46
14. Eidesstattliche Erklärung 67

1. Abstract

Bei RoboCup-Spielen der Fourlegged-Liga werden Sony Aibo Roboter eingesetzt, deren Kamerabildqualität sehr stark von den Lichtverhältnissen abhängt. Je nach Beleuchtung können die von der Kamera erfassten Farben sich so von der Realität unterscheiden, dass sie als andere Farben erkannt werden. Die erkannten Farben stellen einen Grundbaustein der weiteren Bildbearbeitung dar. Damit die Bildbearbeitung zuverlässig funktioniert, haben alle RoboCup-Spiele bis zur heutigen Zeit bei fast konstanten Lichtverhältnissen stattgefunden. Da nicht immer konstante Lichtverhältnisse gegeben sind, und die Roboter auch bei realen Bedingungen spielen können sollen, müssen die Roboter in der Lage sein, trotz sich ändernder Lichtverhältnisse Farben bestimmten Farbklassen zuzuordnen.

In der vorliegenden Arbeit wird untersucht, wie die Farben fehlerfrei im Echtzeit-Betrieb identifiziert werden können. Im Weiteren wird analysiert, welche Verfahren bei anderen RoboCup-Teams eingesetzt werden. Zum Schluss wird ein Ausblick auf eine Automatisierung der Farberkennung gegeben.

2. Einleitung

Robotervision

Roboter, die sich autonom in einer dynamischen Umgebung bewegen sollen, müssen in Echtzeit eine Vielzahl von Aufgaben erfüllen. Sie sind mit Kameras und diversen Sensoren ausgerüstet. Damit sie eine Entscheidung treffen und entsprechende Aktionen ausführen können, müssen sie die Informationen verarbeiten. Aus den Informationen wird zum Beispiel eine Karte des Spielzustandes erstellt, die eigene Position, die der Mitspieler, Gegner und des Balls bestimmt. Die Informationen müssen dabei rechtzeitig zur Verfügung stehen, damit der Roboter rechtzeitig eine Entscheidung treffen kann, und entsprechend der Situation reagieren kann.

Eine zentrale Rolle, Informationen aktuell und rechtzeitig zu bekommen, spielt dabei die Bilderkennung. Aus Bildern werden Objekte identifiziert sowie ihre Lage bestimmt und für die weitere Bearbeitung bereitgestellt.

Die verwendeten Sony Aibo Roboter ERS-210A und ERS-7 liefern Kamerabilder, deren Qualität je nach Beleuchtung stark schwankt. Deswegen werden alle Fußballspiele bei fast konstanten Lichtverhältnissen durchgeführt.

Die Lichtverhältnisse hängen von dem wirklichen Wettbewerbsort ab. Im Jahr 2005 können sie sich bedeutsam von den vorherigen Jahren unterscheiden, weil nur Deckenlichter verwendet werden. Die Beleuchtung wird etwa 1000 Lux sein.[1]

Es kommt trotzdem zu Änderungen der Lichtverhältnisse, wie z.B. durch Schatten von Robotern oder Zuschauern, oder sogar durch das Außenlicht. Aber das Ziel des RoboCup-Projektes ist es, bis zum Jahr 2050 eine Gruppe *Humanoider Roboter* zu entwickeln, die ein menschliches Weltmeisterteam im Fußball schlagen können.[2]

Damit die Bilderkennung zuverlässig funktioniert, muss sie in der Lage sein, trotzt sich ändernder Lichtverhältnisse „gesehene“ Farben bestimmten Farbklassen (Objekten) zuzuordnen.

Die einzelnen zu erkennenden Objekte unterscheiden sich durch eindeutig zugeordnete Farben.

Diese Arbeit befasst sich mit der “The Variable Lighting Challenge“[3] in der “Sony **Four-legged robot league**”**.**

Im Kapitel 3 werden die Grundbegriffe erklärt.

Im Kapitel 4 wird die Aufgabenstellung beschrieben und allgemeine Spielregeln erklärt.

Die wichtigsten Farbmodelle werden im Kapitel 5 vorgestellt.

[1] Siehe den Anhang: Abschnitt 13.3, Spielregeln RoboCup 2005, Unterabschnitt 1.4
[2] http://www.robocup.org, 19.04.2005
[3] Siehe den Anhang: Abschnitt 13.2 „The Variable Lighting Challenge“

Im Kapitel 6 wird eine Analyse des Farbverhalten verschiedener Objekte bei diversen Belichtungsstärken durchgeführt und anschließend verglichen.

Danach werden im Kapitel 7 vorhandene Module des GermanTeam Projektes untersucht und Vor- und Nachteile sowie Gemeinsamkeiten festgestellt.

Die Möglichkeiten dynamischer Verfahren der Farbklassifikation werden im Kapitel 8 angedeutet.

Weiter werden im Kapitel 9 eine Zusammenfassung des Kapitels 6 bis 8 vorgestellt, sowie die Analyse, die Implementation und die Einbindung in den GermanTeam-Code beschrieben.

Im Kapitel 10 wird ein Ausblick auf die Automatisierung der Farberkennung gegeben.

Eine Zusammenfassung wird im Kapitel 11 vorgestellt.

3. Grundbegriffe

Was ist unter dem Begriff *RoboCup* zu verstehen?[4]

Der Begriff *RoboCup* bezeichnet eine internationale Forschungs- und Ausbildungsinitiative, deren Ziel es ist, die Künstliche Intelligenz und die Robotertechnik-Forschung zu fördern und eine ganze Reihe von Standardproblemen zu untersuchen.

RoboCup - Fußball[5]

Der Hauptfokus der RoboCup Tätigkeiten ist der Wettbewerbsfußball. Die Spiele sind die wichtigste Gelegenheit für die Forscher, um technische Information auszutauschen. Der RoboCup-Fußball wird in die folgenden Ligen geteilt[6]:

	**Simulation league** Mobile autonome Softwareagenten spielen Fußball auf einem virtuellen Feld innerhalb eines Computers. Die Spiele haben 5-minütigen Halbzeiten. Das ist eine der ältesten Liegen im RoboCup-Fußball.
	**Small-size robot league** Kleine Roboter mit bis zu 18 cm im Durchmesser spielen Fußball mit einem orangefarbenen Golfball in Mannschaften von bis zu 5 Robotern auf einem Feld mit der Größe größer als einen Pingpongtisch. Die Spiele haben 10-minutigen Halbzeiten.
	**Middle-size robot league** Roboter mittlerer Größe, die bis zu 50 cm Durchmesser haben, spielen Fußball in Mannschaften von bis zu 4 Robotern mit einem orangefarbenen Fußball auf einem Feld der Größe 12x8 Meter. Die Spiele werden in 10-minütigen Halbzeiten geteilt.
	**Four-legged robot league** Mannschaften von 4 vierbeinigen Roboterhunden (AIBO von SONY) spielen Fußball auf einem 6 x 4 Meter-Feld. Die Spiele haben 10-minutige Halbzeiten.

[4] Siehe den Anhang: Abschnitt 13.1 „What is RoboCup?“
[5] Siehe den Anhang: Abschnitt 13.1 „RoboCupSoccer“
[6] http://www.robocup.org/Intro.htm, 21.04.2005

	**Humanoid league** Diese Liga wurde 2002 eingeführt. Zweifüßige autonome humanoide Roboter spielen im "Elfmeterschießen", "1 gegen 1" und "2 gegen 2". "Freistil" Wettbewerbe werden ebenso erwartet.

RoboCup-Challenges

Zusätzlich zu dem RoboCup-Fußball werden auch Challenges organisiert. Eine Challenge ist eine Herausforderung, um Mannschaften zu ermuntern, ein bestehendes Hauptproblem im RoboCup zu untersuchen und zu lösen.

GermanTeam Projekt

Das GermanTeam ist eine deutsche Roboter-Fußballmannschaft, die in der Sony **Four-legged robot league** an internationalen RoboCup Wettkämpfen teilnimmt. Das Projekt ist eine Zusammenarbeit von vier deutschen Universitäten: der Humboldt Universität zu Berlin, der Universität Bremen, der Technischen Universität Darmstadt, und der Universität Dortmund.[7]

Der von dem GermanTeam in der RoboCup-Weltmeisterschaft 2004 verwendete Quellcode ist für das Download verfügbar. Das Archiv enthält die notwendige Software, um Roboter-Fußballspiele durchzuführen. Die Komponenten sind sowohl als C++ Quellcode als auch für Windows (XP/2000) kompilierte Module (Programme) im Archiv enthalten.[8]

Hamburg Dog Bots[9]

Hamburg Dog Bots ist das Team der Universität Hamburg, das in der Sony **Four-legged robot league** an internationalen RoboCup Wettkämpfen teilnimmt. Das Team hat 2004 den Quellcode des GermanTeams übernommen und entwickelt seitdem darauf aufbauend eigenständigen Quellcode.

[7] http://www.germanteam.org, 19.04.2005
[8] http://www.germanteam.org, 19.04.2005
[9] http://www.informatik.uni-hamburg.de/TIS/index.php?content=robocup/index.htm, 15.04.2005

4. Aufgabenstellung

Die "Variable Lighting Challenge"[10]

Diese Challenge beabsichtigt Mannschaften zu ermuntern, die Robustheit ihrer Vision zu Beleuchtungsänderungen zu verbessern. Sie beruht auf dem Penalty Shootout. Die Mannschaft, die die Herausforderung versucht, stellt einen einzelnen Roboter (mit einer blauen Uniform) auf das Feld. Dieser Roboter muss so viele Treffer erzielen, wie er in das gelbe Tor in drei Minuten kann. Die Mannschaft, die die meisten Treffer hat, gewinnt die Challenge. Wenn zwei Mannschaften dieselbe Zahl von Treffer haben, dann gewinnt die Mannschaft mit der niedrigsten durchschnittlichen Trefferzeit (Hinweis: das ist dasselbe wie die Auswahl einer Mannschaft, die ihr letztes Tor am frühsten schoss). Wenn keine Mannschaft Treffer hat, dann gewinnt die Mannschaft, die mit dem Ball am Ende ihrer Zeit am nächsten zum Tor war.

Zusätzlich zum einzelnen blauen Roboter werden zwei rote Gegner-Roboter auf das Feld gestellt. Diese beiden Roboter werden gestoppt und in dem Pause (*UNSW)* Zustand eingefroren. Einer wird in eine Tormann-Position auf einer Seite des gelben Tors gestellt. Der andere wird ins Drittel vom Feld am nächsten zum gelben Tor, mindestens 30 cm weg vom Rand gestellt. Die genauen Positionen aller Roboter werden vom Schiedsrichter entschieden, und sind für alle Mannschaften gleich. Abbildung 1 zeigt eine mögliche Situation.

Es gibt einen einzelnen Ball auf dem Feld. Am Anfang wird er an die Anstoßposition gelegt. Nach jedem Tortreffer wird der Ball an die Anstoßposition zurückgebracht. Der Roboter wird vom Schiedsrichter nicht bewegt und muss seinen eigenen Weg zurück zum Zentrum des Feldes finden, um den Ball wieder zu erreichen. Der Roboter wird eine Nachricht vom Game Controller mit seinem neuen Spielstand erhalten. Da das Feld keine Banden hat, ist es sehr wahrscheinlich, dass der Ball aus dem Feld hinausrollt. In diesem Fall wird er zurück an den Einwurfpunkt gelegt (siehe das Dokument mit den Spielregeln für weitere Details) nahe dem Punkt, wo der Ball hinaus gekickt wurde.

Die Hauptherausforderung dieser Aufgabe ist, dass die Beleuchtung von der RoboCup Standardbeleuchtung verschieden sein wird. Einige zusätzliche Scheinwerfer werden aufgestellt, um variable Lichtverhältnisse zu liefern (wir stellen uns die Theaterbeleuchtungsausrüstung vor, die variable Beleuchtungskraft verwendet). Diese zusätzlichen Scheinwerfer sollen weißes Licht der unbekannten Farbtemperatur haben. Die Lampen können bedeckt werden, um variable Lichtverhältnisse zu erreichen.

Vor der Challenge wird der Schiedsrichter eine Liste von Beleuchtungsänderungen vorbereiten. Diese muss Perioden der unveränderlichen Beleuchtung, Perioden der langsamen Änderung der Beleuchtung und Perioden von schnellen Änderungen der Beleuchtung einschließen. Man stellt sich vor, dass wenn die zusätzliche Beleuchtung über dem Feld ungleichmäßig ist, auch die Beleuchtungsänderungen ungleichmäßig sind. Diese Beleuchtungsliste, die bis direkt vor der Challenge unbekannt ist, ist gleich für alle Mannschaften.

[10] Sehe den Anhang: Abschnitt 13.2 „The Variable Lighting Challenge"

Wenn sonst nichts angegeben ist, gelten normale Penalty Shootout-Regeln. Es gibt keine Strafe für den Angriff auf einen Gegner-Roboter. Es ist nicht erlaubt, einem Roboter zu helfen, den Angriff zu beenden.

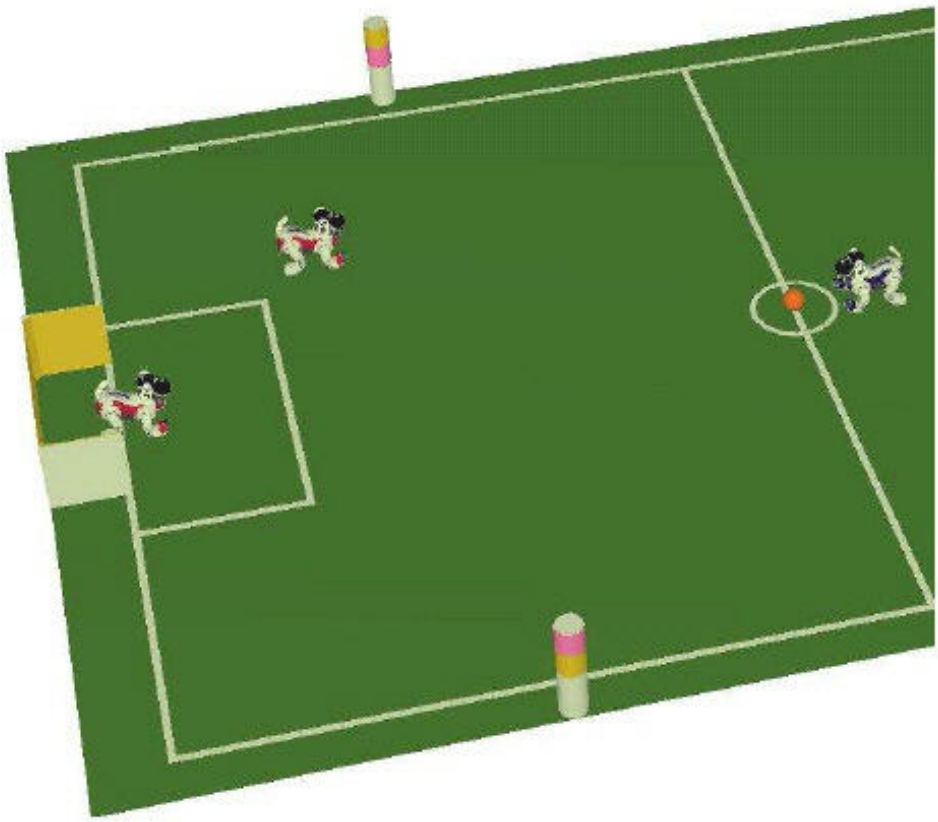

Abbildung 1 Ein Beispiel einer Aufstellung von Robotergegnern für die „Variable Lighting Challenge“

Allgemeine Bestimmungen und Regeln[11]

Die Sony Fourlegged Robot League ist ein Teilprojekt des RoboCup, in dem Aibo-Roboterhunde vom Typ ERS-210, ERS-210A oder ERS-7 von SONY verwendet werden. Diese sind entweder schwarz oder weiß. An der Hardware der Roboter sind keine Veränderungen erlaubt. Es dürfen weder zusätzliche Sensoren noch Speichererweiterungen oder leistungsfähigere Prozessoren eingebaut werden. Es ist ebenfalls nicht erlaubt, Berechnungen auf einen externen Rechner auszulagern. Ein WLAN wird über einen Rechner der Organisatoren bereitgestellt. Über diesen werden lediglich die Nachrichten zwischen den Robotern weitergeleitet. Berechnungen auf diesen Rechner auszulagern, ist nicht gestattet.

In der Tabelle 1 sind festgelegte Farben und entsprechende Objekte dargestellt[12].

[11] Siehe Anhang, Abschnitt 13.3, Spielregeln RoboCup 2005
[12] Siehe Anhang, Abschnitt 13.3, Spielregeln RoboCup 2005

Farben	Objekte
Orange	Ball
Dunkelblau	Teammarker
Himmelblau	Tor und Orientierungsturm
Grün	Spielfeld
Rot	Teammarker
Gelb	Tor
Pink	Orientierungsturm
Schwarz	Roboterkörper
Weiß	Orientierungstürme, Feldlinien, Roboterkörper

Tabelle 1: Farben der Objekte des Spielfeldes und der Roboter.

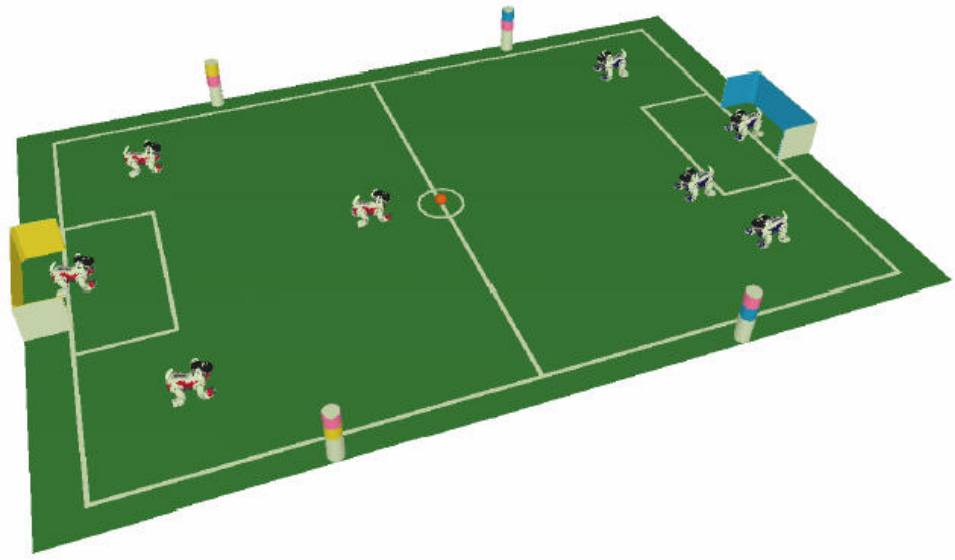

Abbildung 2: Das Spielfeld 2005.[13]

Die Ausgangslage sieht so aus:

1. Die Farbklassen sind konstant und vor dem Spiel (Challenge) bekannt.
2. Die Objekte sind zu den bestimmten Farbklassen zugeordnet. Die Roboterkörper sind mit Teammarkern der bestimmten Farbklassen markiert. Orientierungstürme bestehen aus mehreren Farbklassen.
3. Es gibt bestimmte Objektformen:

Objektform	Objekte
Einfache (flache) Formen	Spielfeld, Tore, Feldlinien, Spielwände
Zylinderformen	Orientierungstürme
Kugelformen	Ball

[13] Siehe Anhang, Abschnitt 13.3, Spielregeln RoboCup 2005

Komplizierte Formen	Roboterkörper
Andere Formen	Andere Objekte, die außerhalb des Spielfeldes sind, z.B. Zuschauer.

4. Die Lichtverhältnisse können sich während des Spiels (Challenge) ändern.
5. Die Rechenleistung ist knapp und begrenzt.
6. Die Speicherkapazität ist knapp und begrenzt.
7. Die Rechenzeit ist stark beschränkt.
8. Es werden Aibo-Roboterhunde vom Typ ERS-210, ERS-210A oder ERS-7 verwendet.
9. Das Kamerafarbsystem ist YCbCr.
10. Die Kameras der typgleichen Roboter sind fast identisch eingestellt und liefern Bilder verhältnismäßig gleicher Qualität.

Aufgabenstellung

Es muss ein Verfahren entwickelt und implementiert werden, das in der Lage ist, trotz sich ändernder Lichtverhältnisse Farben bestimmten Farbenklassen zuzuordnen.

Die Lösung muss auf dem Kode des Hamburg Dog Bots Projektes aufgebaut werden und mit deren Schnittstellen kompatibel sein.

5. Farbmodelle

Farbmodelle dienen der formalen Beschreibung von Farben. Eine Farbe lässt sich durch Angabe von Farbton, Sättigung und Helligkeit eindeutig beschreiben. Diese Parameter werden zum Beispiel im HSV-Modell (Kapitel 5.3) direkt verwendet. Für verschiedene technische Anwendungen wurden verschiedene Farbmodelle entwickelt, die für die jeweils zugeordneten Aufgabengebiete vorteilhafte Eigenschaften besitzen.

5.1. Das RGB-Modell[14]

Das **RGB-Modell** ist eines der wichtigsten Farbmodelle, bei dem die Grundfarben **R**ot **G**rün und **B**lau (englisch Red Green Blue) sich zu Weiß addieren (Lichtmischung). Das ist ein additives Farbmodell. Jeder Farbanteil liegt zwischen 0% und 100%. Der Farbraum wird in diesem System häufig als Würfel dargestellt (sehe Abbildung 3, die Werte sind zwischen 0 und 255 dargestellt), so dass bei (R=0%; G=0%, B=0%) ein Schwarzpunkt ist, und bei (R=100%; G=100%, B=100%) ein Weißpunkt ist. Dabei liegen die Grautöne auf der Diagonalen zwischen dem Schwarzpunkt und dem Weißpunkt. Die Helligkeit einer Farbe wächst mit der Entfernung zum Schwarzpunkt. Viele andere Farbmodelle beziehen sich bei der Definition ihrer Parameter auf das RGB-Modell.

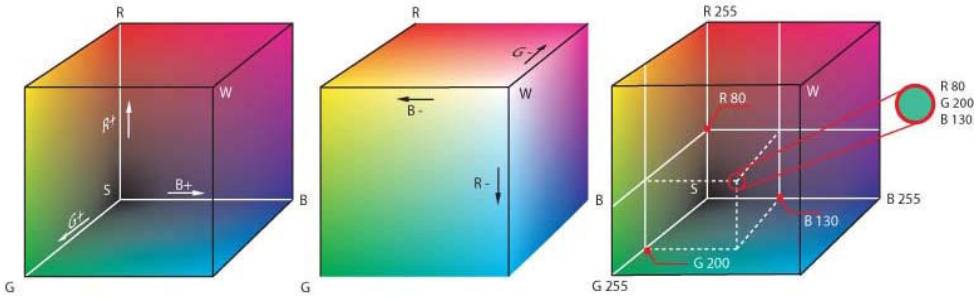

Abbildung 3: Der RGB-Würfel.[15]

5.2. Das YUV-Modell[16]

Das YUV-Farbmodell verwendet zur Darstellung der Farbinformation zwei Komponenten, die Luma (Lichtstärke pro Fläche) und die Chrominanz oder Farbanteil (chroma), wobei die Chrominanz wiederum aus zwei Komponenten (siehe weiter unten) besteht. Die Entwicklung des YUV-Farbmodells geht auf die Entwicklung des Farbfernsehens (PAL) zurück, wo nach Wegen gesucht wurde, zusätzlich zum Schwarz/Weiß-Signal die Farbinformation zu übertragen, um eine Abwärtskompatibilität mit alten Schwarz/Weiß-Fernsehgeräten zu erreichen, ohne die zur Verfügung stehende Übertragungsbandbreite erhöhen zu müssen. Aus dem YUV-Farbmodell der analogen Fernsehtechnik wurde das YCbCr-Farbmodell entwickelt, das bei den meisten Arten der digitalen Bild- und Videokompression eingesetzt wird. Fälschlicherweise wird in jenem Bereich auch oft vom YUV-Farbmodell gesprochen, obwohl eigentlich das YCbCr-Modell

[14] http://de.wikipedia.org/wiki/RGB-Farbraum, 21.04.2005
[15] http://de.wikipedia.org/wiki/RGB-Farbraum, 21.04.2005
[16] http://de.wikipedia.org/wiki/YUV-Farbmodell, 21.04.2005

benutzt wird. Dies sorgt oft für Verwirrung (siehe Quellkode unter „Image::convertFromYUVToRGB“).

Beim YUV-Modell wird die Intensität im Y-Parameter gespeichert.

$$Y = R + G + B$$

Die Crominanzsignale, auch Farbdifferenzsignale, enthalten die Farbinformation. Sie entstehen aus der Differenz Blau minus Luma bzw. Rot minus Luma.

$$U = B - Y$$

$$V = R - Y$$

Aus den drei erzeugten Komponenten Y, U und V können später wieder die einzelnen Farbanteile der Grundfarben berechnet werden:

$$Y + U = Y + (B - Y) = Y - Y + B = B$$

$$Y + V = Y + (R - Y) = Y - Y + R = R$$

$$Y - B - R = (R + G + B) - B - R = G$$

Das YCbCr-Farbmodell wird von der im AIBO ERS-210, AIBO ERS-210A und ERS-7 eingebauten Kamera verwendet.

Die Umrechnung der Farbraumdarstellung von YCbCr nach RGB und zurück wird durch Matrizenmultiplikation durchgeführt.

So ist die Umrechnung im GT2004-Quellkode[17]:

```
static void convertFromYCbCrToRGB(unsigned char Y,
                                  unsigned char Cb,
                                  unsigned char Cr,
                                  unsigned char& R,
                                  unsigned char& G,
                                  unsigned char& B)
{
         int r = (int)(Y + 1.4021 * (Cb - 128)),
             g = (int)(Y - 0.3456 * (Cr - 128) - 0.71448 * (Cb - 128)),
             b = (int)(Y + 1.7710 * (Cr - 128));
    if(r < 0) r = 0; else if(r > 255) r = 255;
    if(g < 0) g = 0; else if(g > 255) g = 255;
    if(b < 0) b = 0; else if(b > 255) b = 255;
    R = (unsigned char) r;
    G = (unsigned char) g;
    B = (unsigned char) b;
}

void Image::convertFromYUVToRGB(const Image& yuvImage)
{
  Image::convertFromYCbCrToRGB(yuvImage);
  /*
  int Y,U,V;
```

[17] \Src\Representations\Perception\Image.h und \Src\Representations\Perception\Image.cpp

```
  int R,G,B;
  for(int y=0; y < yuvImage.cameraInfo.resolutionHeight; y++)
  {
    for(int x=0; x < yuvImage.cameraInfo.resolutionWidth; x++)
    {
      Y = yuvImage.image[y][0][x];
      U = yuvImage.image[y][1][x];
      V = yuvImage.image[y][2][x];
      R = (int)(Y + 1.140 * (U - 128));
      G = (int)(Y - 0.394 * (V - 128) - 0.581 * (U - 128));
      B = (int)(Y + 2.028 * (V - 128));
      if(R < 0) R = 0; if(R > 255) R = 255;
      if(G < 0) G = 0; if(G > 255) G = 255;
      if(B < 0) B = 0; if(B > 255) B = 255;
      image[y][0][x] = (unsigned char)R;
      image[y][1][x] = (unsigned char)G;
      image[y][2][x] = (unsigned char)B;
    }
  }
  this->cameraInfo = yuvImage.cameraInfo;
  */
}

void Image::convertFromRGBToYUV(const Image& rgbImage)
{
  int Y,U,V;
  int R,G,B;

  for(int y=0; y < rgbImage.cameraInfo.resolutionHeight; y++)
  {
    for(int x=0; x < rgbImage.cameraInfo.resolutionWidth; x++)
    {
      R = rgbImage.image[y][0][x];
      G = rgbImage.image[y][1][x];
      B = rgbImage.image[y][2][x];

      // RGB2YUV Method 1
      //Y =       (int) ( 0.299 * R + 0.587 * G + 0.114 * B);
      //V = 127 + (int) (-0.147 * R - 0.289 * G + 0.437 * B);
      //U = 127 + (int) ( 0.615 * R - 0.515 * G - 0.100 * B);

      // RGB2YUV Method 2
      Y =       (int)( 0.2990 * R + 0.5870 * G + 0.1140 * B);
      V = 127 + (int)(-0.1687 * R - 0.3313 * G + 0.5000 * B);
      U = 127 + (int)( 0.5000 * R - 0.4187 * G - 0.0813 * B);

//     if (R+G+B != 3*204)
//         int i=0;

      if(Y < 0) Y = 0; if(Y > 255) Y = 255;
      if(U < 0) U = 0; if(U > 255) U = 255;
      if(V < 0) V = 0; if(V > 255) V = 255;

      image[y][0][x] = (unsigned char)Y;
      image[y][1][x] = (unsigned char)U;
      image[y][2][x] = (unsigned char)V;
    }
  }
  this->cameraInfo = rgbImage.cameraInfo;
}
```

Im weiteren Verlauf wird von YUV-Modell gesprochen, obwohl eigentlich das YCrCb-Modell gemeint ist.

5.3. Das HSV-Modell[18]

Die Abkürzungen **HSV**, **HSI**, **HSL** oder **HSB** stehen für das Farbmodell, bei dem man die Farbe mit Hilfe des Farbtons (***H**ue*), der Sättigung (***S**aturation*) und der Helligkeit (***V**alue*, ***B**rightness*, ***L**ightness*, ***I**ntensity*) definiert.

Dabei wird

- der Farbton als Winkel auf dem Farbkreis angegeben (z. B. 0° = Rot, 60° = Gelb, 120° = Grün, 180° = Cyan, 240° = Blau, 300° = Magenta),
- die Sättigung als Prozentwert angegeben (z. B. 0% = keine Farbe, 50% = ungesättigte Farbe, 100% = gesättigte Farbe),
- die Helligkeit als Prozentwert angegeben. (z. B. 0% = keine Helligkeit, 100% = volle Helligkeit).

Diese Parameter werden zur Veranschaulichung häufig auf einem Kegel dargestellt (Abbildung 4).

Null Prozent Helligkeit bedeutet in jedem Fall Schwarz, unabhängig von den Werten der anderen Parameter.

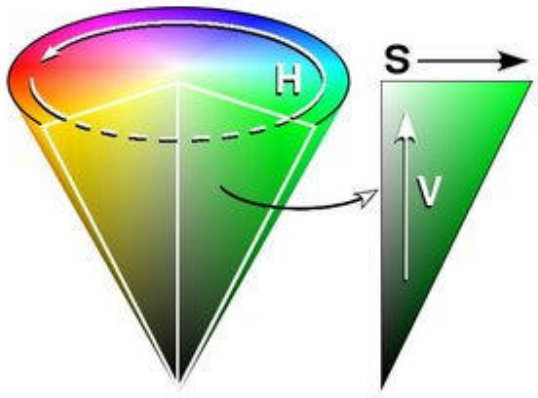

Abbildung 4: HSV-Farbraum als Kegel.[19]

So ist die Umrechnung im GT2004-Quellkode[20]:

```
static void convertFromYCbCrToHSI(unsigned char Y,
                                  unsigned char Cb,
                                  unsigned char Cr,
                                  unsigned char& H,
                                  unsigned char& S,
                                  unsigned char& I)
{
  const double sqrt3 = 1.732050808;
  unsigned char R,
```

[18] http://de.wikipedia.org/wiki/HSV-Farbraum, 21.04.2005
[19] http://de.wikipedia.org/wiki/HSV-Farbraum, 21.04.2005
[20] \Src\Representations\Perception\Image.h und \Src\Representations\Perception\Image.cpp

```
                G,
                B;
  convertFromYCbCrToRGB(Y, Cb, Cr, R, G, B);
  I = R;
  if(G > I) I = G;
  if(B > I) I = B;
  if(I)
  {
    S = R;
    if(G < S) S = G;
    if(B < S) S = B;
    S = (unsigned char) (255 - 255 * S / I);
    if(S)
    {
      int h = int(atan2(sqrt3 * (G - B), 2 * R - G - B) / pi2 * 256);
      if(h > 256) h -= 256;
      else if(h < 0) h += 256;
      H = (unsigned char) h;
    }
    else
      H = 0;
  }
  else
    S = H = 0;
}
```

5.4. Das TSL-Modell[21]

Dieses Farbmodell wird häufig in der Gesichtserkennung eingesetzt, da es sehr gut zum Erkennen der Hautfarbe geeignet ist. Es wird beispielsweise vom German Team (Deutschland) und dem Team ASURA (Japan) für die Farbklassifikation benutzt.

Die Parameter tint (T), saturation (S) und luminance (L) werden durch die Gleichungen (Abbildung 5) definiert. Diese beschreiben die Konvertierung vom RGB-Farbraum in den TSL-Raum.

[21] Passow, J. *Diplomarbeit: Geometrieidentifikation aus den Kamerabildern mobiler autonomer Systeme in der Sony Four-legged Robot League*. Universität der Freien und Hansestadt Hamburg, Fachbereich Informatik, 2004. Seite 22f.

$$T = \begin{cases} \frac{\arctan(\frac{r'}{g'})}{2\Pi} + \frac{1}{4}, & g' > 0 \\ \frac{\arctan(\frac{r'}{g'})}{2\Pi} + \frac{3}{4}, & g' < 0 \\ 0 & g' = 0 \end{cases}$$

$$S = \sqrt{\frac{9}{5}(r'^2 + g'^2)}$$

$$L = 0.299R + 0.587G + 0.144B$$

$$r' = \frac{R}{R+G+B} - \frac{1}{3}$$
$$g' = \frac{G}{R+G+B} - \frac{1}{3}$$

Abbildung 5: Umrechnung vom RGB-Farbraum in den TSL-Raum.

So ist die Umrechnung im GT2004-Quellkode[22]:

```
void Image::convertFromYUVToTSL(const Image& yuvImage)
{
  int Y,U,V;
  int T,S,L;
  double  y_in, u_in, v_in, tmp, tmp_r, tmp_g, tmp_b, t_out, s_out;

  for(int y = 0; y < yuvImage.cameraInfo.resolutionHeight; y++)
  {
    for(int x = 0; x < yuvImage.cameraInfo.resolutionWidth; x++)
    {
      Y = yuvImage.image[y][0][x];
      U = yuvImage.image[y][1][x];
      V = yuvImage.image[y][2][x];


      const double pi2_div = 0.15915494309189533576888376337251;  /* 1.0 / (PI * 2.0) */

      y_in = (double) Y;
      u_in = (double) U;
      v_in = (double) V;
      u_in -= 128.0;
      v_in -= 128.0;
      tmp = 1.0 / (4.3403 * y_in +  2.0 * u_in + v_in);
      tmp_r = (-0.6697 * u_in + 1.6959 * v_in) * tmp;
      tmp_g = (-1.168  * u_in - 1.3626 * v_in) * tmp;
      tmp_b = ( 1.8613 * u_in - 0.331  * v_in) * tmp;
      if (tmp_g > 0.0)
      {
        t_out = (atan2(tmp_r, tmp_g) * pi2_div + 0.25) * 255.0;
      }
      else if (tmp_g < 0.0)
      {
        t_out = (atan2(-tmp_r, -tmp_g) * pi2_div + 0.75) * 255.0;
      }
      else
      {
        t_out = 0.0;
      }
      s_out = sqrt(1.8 * (tmp_r * tmp_r + tmp_g * tmp_g + tmp_b * tmp_b)) * 255.0;
```

[22] \Src\Representations\Perception\Image.h und \Src\Representations\Perception\Image.cpp

```
      /* Crop T and S values */
      if (t_out < 0.0)
      {
        t_out = 0.0;
      }
      else if (t_out > 255.0)
      {
        t_out = 255.0;
      }
      if (s_out < 0.0)
      {
        s_out = 0.0;
      }
      else if (s_out > 255.0)
      {
        s_out = 255.0;
      }

      T = (unsigned char) t_out;
      S = (unsigned char) s_out;
      L = Y;

      image[y][0][x] = (unsigned char)T;
      image[y][1][x] = (unsigned char)S;
      image[y][2][x] = (unsigned char)L;
    }
  }
  this->cameraInfo = yuvImage.cameraInfo;
}
```

6. Analyse

6.1. Testaufnahmen

Für die Analyse wurden so genannte **log-Dateien** aufgenommen. Bei den Aufnahmen wurden von einem stehenden Roboter unter verschiedenen Lichtverhältnissen Kamerabilder aufgenommen. Dabei wurden zuerst alle Lampen eingeschaltet und dann nacheinander stufenweise abgeschaltet, bis es ganz dunkel im Labor war. Ferner wurde bei jeder Änderung der Lichtverhältnissen zusätzlich jedes Objekt nacheinander direkt von allen Seiten bestrahlt.

Dieses Verfahren wurde mit verschiedenen Robotertypen in unterschiedlichen Roboterpositionen wiederholt.

6.2. Bildanalyse

Als erstes werden die Bilder in Abbildung 6 betrachtet. Diese sind statische Aufnahmen einer Roboterkamera des Roboters ERS-210A bei verschiedenen Lichtverhältnissen.

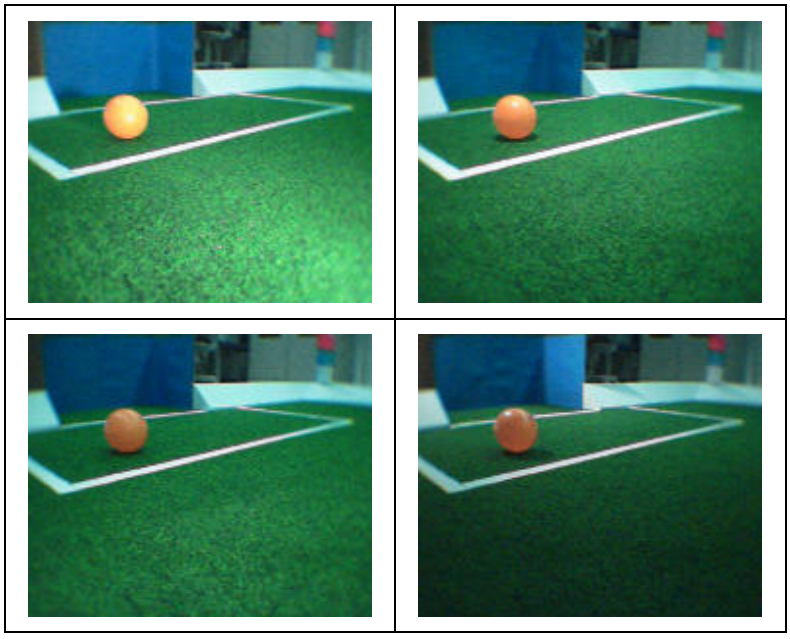

Abbildung 6: Aufnahmen der Aibo-Roboter Typ ERS-210A.

Zunächst wird nur ein Ball betrachtet und dann werden die Balleigenschaften mit den Eigenschaften anderer Objekte verglichen.

Farbverlauf eines Balls

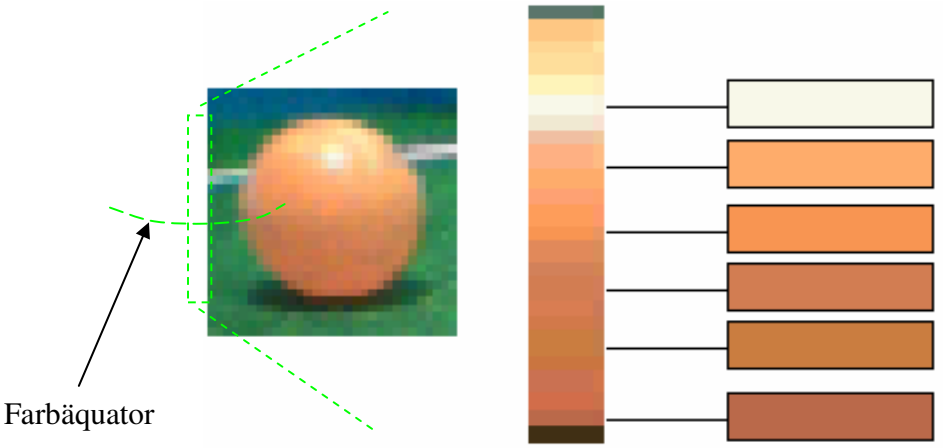

Abbildung 7: Farbverlauf am Ball.

So wie in Abbildung 7 dargestellt, ist gut zu erkennen, dass der Ball einen Farbverlauf von braun bis orange im Bereich unterhalb des Farbäquators und von orange bis fast weiß oberhalb des Farbäquators aufweist. Der Farbäquator ist die Linie, die zwischen dem maximalen und minimalen Helligkeitswert liegt und ohne auf den Ball fallende Schatten fast die gleiche Farbe (in diesem Fall orange) hat. Da die Farben unterhalb des Äquators keine direkte Lichtstrahlung bekommen, liegen sie im Schatten. Deswegen weisen sie die Farbpalette von orange bis braun in manchen Fällen bis schwarz auf. Die Farben oberhalb des Äquators weisen umgekehrt die Farbpalette von orange bis weiß auf.

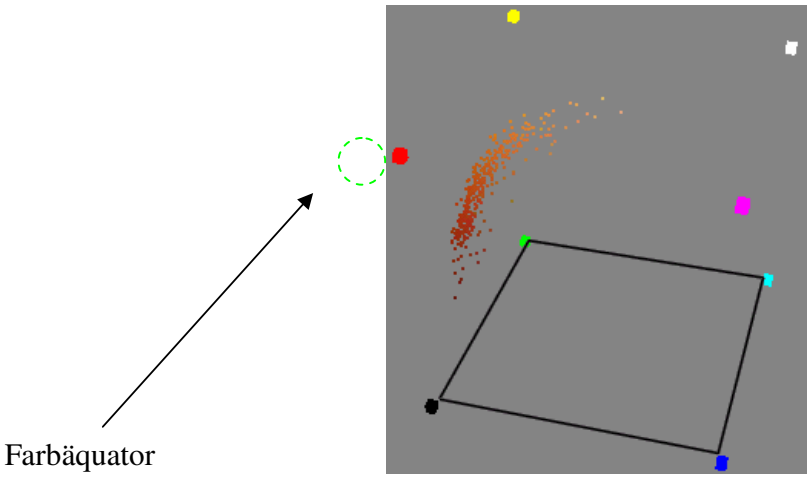

Abbildung 8: Farbverlauf eines Balls im RGB-Raum (Aufnahme Aibo ERS-7).

Aus der Abbildung 8 ist erkennbar, wie sich die Farben im RGB-Raum verteilt haben. Die Farben, die oberhalb des Äquators liegen, verteilen sich von den Äquatorfarben bis hin zum weißen Würfel. Im anderen Fall (Schattenbereich) werden die Farben bis zum schwarzen Würfel hin verteilt.

Es stellt sich die Frage, wie die Farbverteilung bei einem extrem hellen und extrem dunklen Zustand aussieht.

In Abbildung 10 ist erkennbar, dass bei sehr starker Beleuchtung, orange zu gelb und weiter zu weiß wird. Die Ursache liegt in dem Aufbau des CCD-Chip der Kamera.

Exkurs: 1 Chip CCD

CCD ist die Abkürzung für Charge Coupled Device. Auf Deutsch heißt das soviel wie: ladungsgekoppeltes Bauelement. CCD-Sensoren bestehen aus einem ein- oder zweidimensionalen Array von Speicherelementen. Verwendung finden sie hauptsächlich als Bildsensor bei Videokameras, Scannern und digitalen Fotoapparaten.[23]

Die 1 – Chip CCD Technik basiert auf einem Farbfiltermosaik, was vor der lichtempfindlichen Sensorfläche angebracht ist. Dabei erfasst jedes lichtempfindliche Element des CCD das Licht von nur einer der drei Grundfarben. Da das menschliche Auge im Grünbereich weitaus empfindlicher ist, als im übrigen Farbspektrum, benutzt man i.d.R. doppelt so viele grüne wie blaue und rote Filter. Da jeder einzelne Sensor nur jeweils eine Farbe aufzeichnet, müssen die anderen Farbanteile softwaremäßig aus den umliegenden Pixel rekonstruiert werden, der Fachbegriff hierfür heißt "Farbinterpolation". Die Interpolationsalgorithmen sind von Hersteller zu Hersteller verschieden, so ergeben sich auch zum Teil Qualitätsunterschiede im Bildergebnis verschiedener Digitalkameras, selbst wenn diese den gleichen CCD-Sensor eingebaut haben.[24]

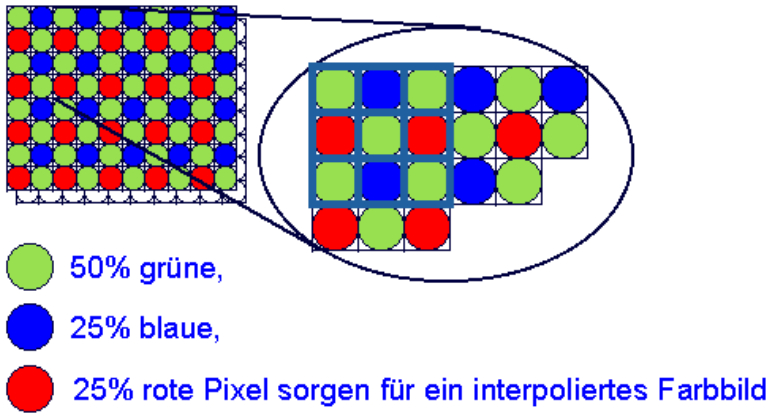

Abbildung 9: Aufbau eines farbigen CCD-Chips[25].

Exkurs Ende.

Bei steigender Beleuchtungskraft wird die Ballfarbe von orange zu gelb und weiter zu weiß. Ab einem bestimmten Wert ist der Rotanteil (siehe Abbildung 9) übersättigt und kann nicht mehr wachsen. Die Grün- und Blauanteile können weiter steigen. Zunächst wachsen beide Farben,

[23] http://www.ccd-sensor.de/html/grundprinzip.html, 10.04.2005
[24] http://www.ccd-sensor.de/html/1_-_chip_ccd.html, 10.04.2005
[25] http://www.ccd-sensor.de/html/1_-_chip_ccd.html, 10.04.2005

dann wird auch der Grünanteil übersättigt. In diesem Zustand ist die Farbe gelb. Der Blauanteil wächst weiter, bis alle Farbanteile übersättigt sind und die Farbe zu weiß wird.

Der komplette Farbverlauf wird auf den Abbildung 11 und Abbildung 12 dargestellt.

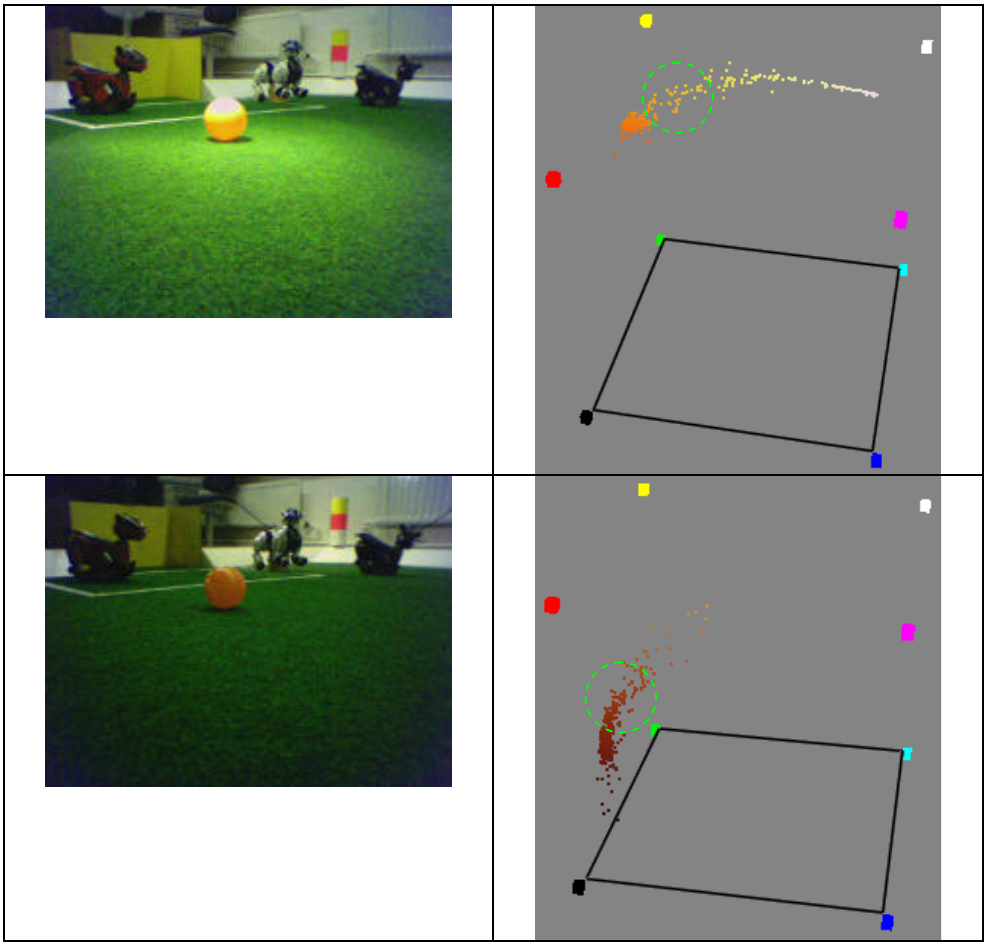

Abbildung 10: Ball bei starker (oben) und schwacher (unten) Beleuchtung und dazugehöriger Farbverläufe (Aufnahme Aibo ERS-7).

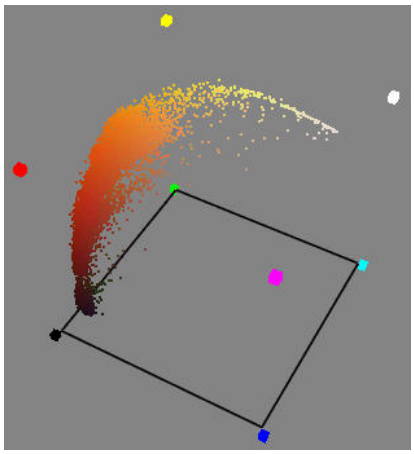

Abbildung 11: Kompletter Farbverlauf eines Balls von sehr hell bis sehr dunkel im RGB-Raum (Aufnahme Aibo ERS-7).

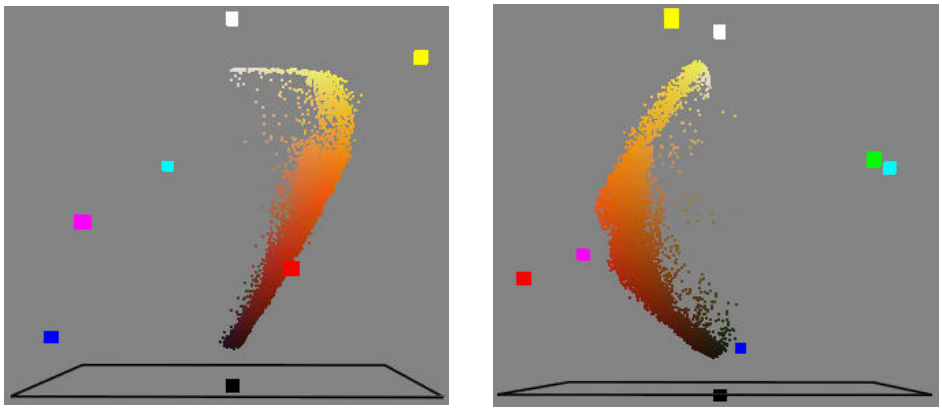

Abbildung 12: Kompletter Farbverlauf eines Balls von sehr hell bis sehr dunkel im YUV-Raum. Links liegt die Perspektive auf dem roter Würfel, rechts auf dem gelber Würfel. (Aufnahme Aibo ERS-7).

Farbverlauf eines Orientierungsturms

Die Zylinderform ist im Prinzip der Kugelform ähnlich. Der einzige Unterschied besteht darin, dass der Äquator bei den Zylinderformen parallel der Zylinderachse verläuft.

In Abbildung 13 ist ein Orientierungsturm dargestellt. Der entsprechende Farbverlauf ist in der Abbildung 14 zu sehen. Die Abbildung 15 zeigt vergrößerte Ausschnitte des Farbraumes des Originalbildes.

Abbildung 13: Orientierungsturm (Aufnahme Aibo ERS-210A).

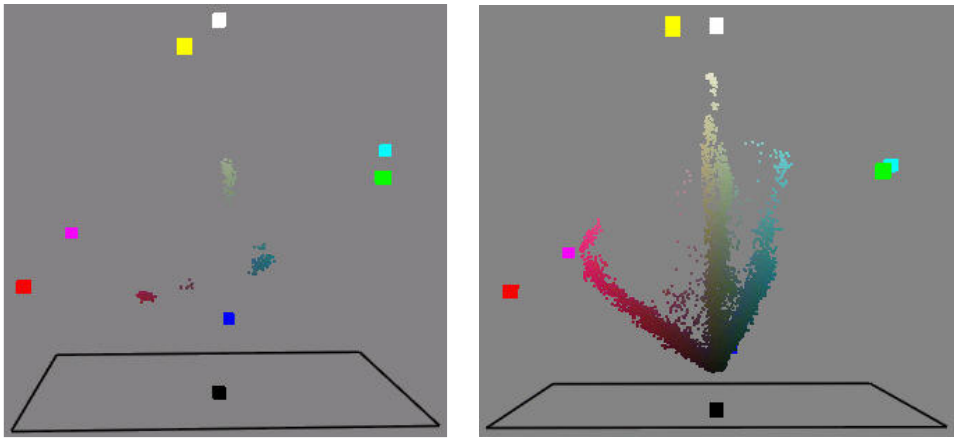

Abbildung 14: Links der Farbverlauf eines Orientierungsturm im YUV-Raum, rechts der gesamte Farbverlauf eines Orientierungsturms im YUV-Raum.

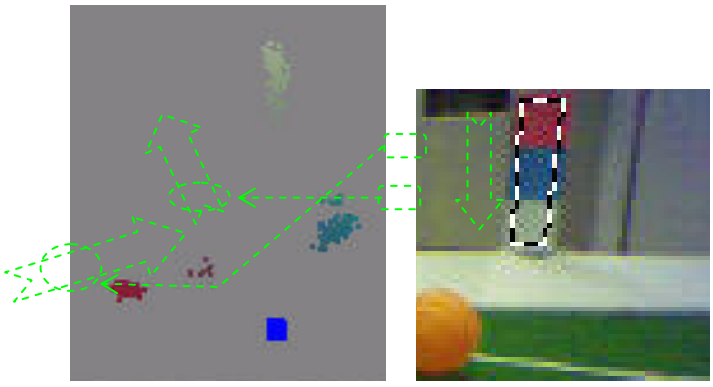

Abbildung 15: Farbverlauf eines Orientierungsturms und deren Zwischenfarben.

Wie in Abbildung 15 zu erkennen ist, existieren noch Zwischenfarben, die im ganzen Übergang auftauchen können, zum Beispiel zwischen Pink und Himmelblau. Dieses Thema wird im weiteren Verlauf des Abschnittes noch näher betrachtet.

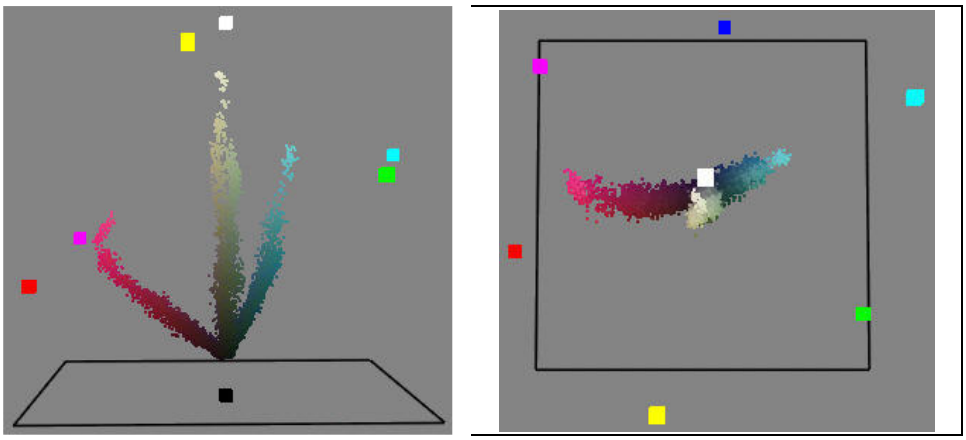

Abbildung 16: Gesamter Farbverlauf eines Orientierungsturms ohne Zwischenfarben im YUV-Raum. Links die Perspektive von der Seite, rechts die Perspektive von oben.

In Abbildung 16 ist der gesamte Farbverlauf ohne Zwischenfarben dargestellt. Wie eindeutig zu erkennen ist, sind die Farben sehr gut unterscheidbar, sofern das Bild nicht zu dunkel ist.

Farbverlauf einfachen Formen z.B. Spielfeldes

Bei einfachen Formen ist das Verhalten ähnlich. Der komplette Farbverlauf bleibt relativ einfach ohne bedeutende Verzerrungen (siehe Abbildung 17).

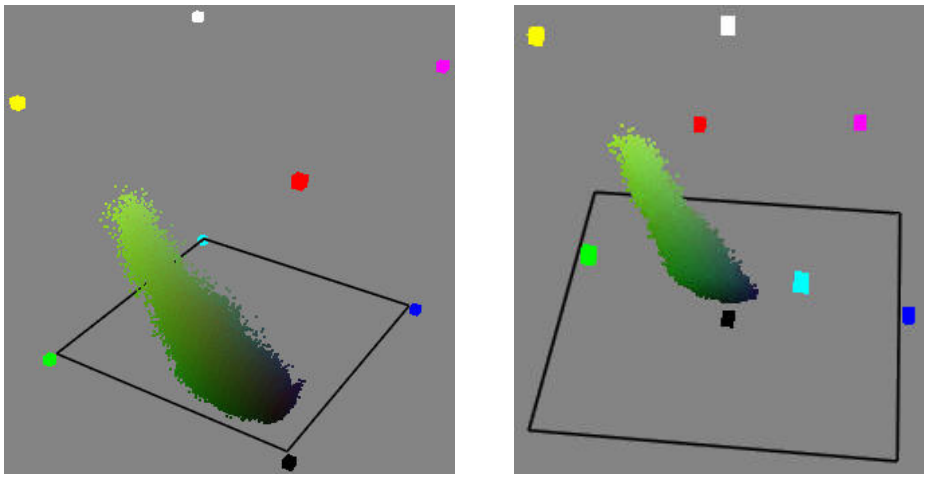

Abbildung 17:Der Farbverlauf des Spielfeldes im RGB-Raum (links) und im YUV-Raum (rechts), Aufnahme Aibo ERS-7.

Farben am Rande des Objektes

Zuerst wurde nur der Rasen selektiert, dann nur die weiße Linie ohne Rand. Das Ergebnis ist in der Abbildung 18 unten links zu sehen. Hier sind die Farben relativ zusammen gruppiert.

Als nächstes wurde die Linie mit dem Rasen zusammen selektiert (siehe Abbildung 18 oben rechts und links und unten rechts). Es sind deutlich mehr Punkten zu erkennen, die zwischen den Gruppen liegen.

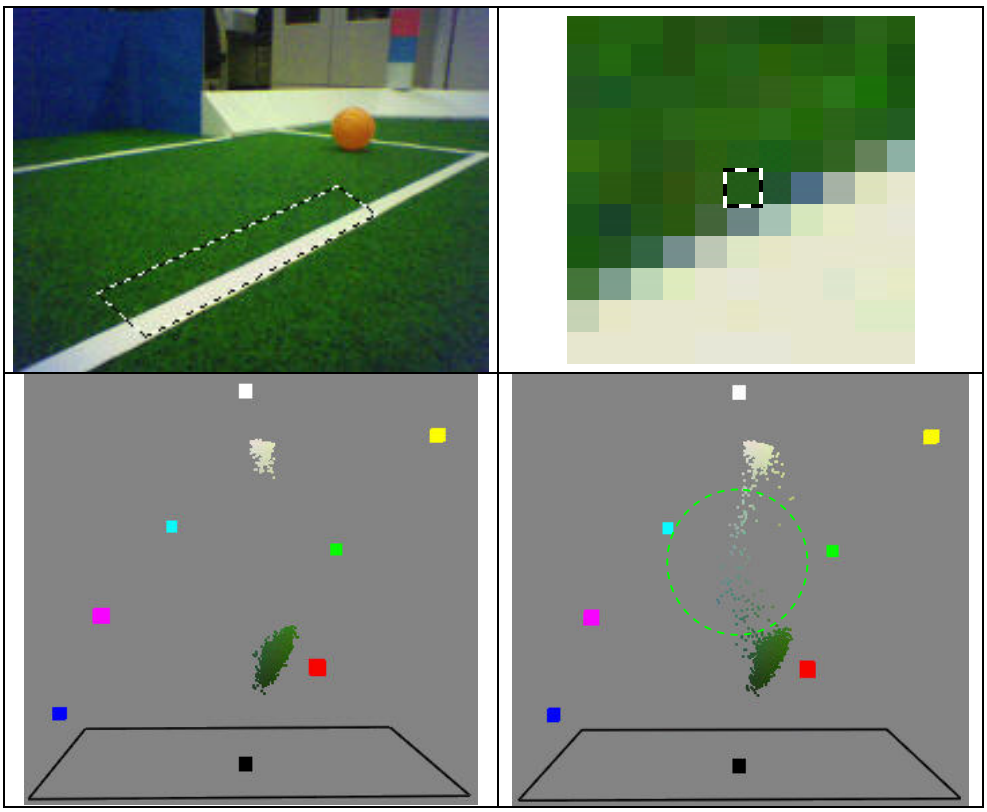

Abbildung 18: Farben am Rande des Objektes.

In der Abbildung 19 sind die Farbverläufe des gesamten Feldes im YUV-Farbraum zu erkennen. Links ohne Zwischenfarben, rechts mit Zwischenfarben. In der rechten Abbildung ist es sehr schwer, z.B. die weiße Farbe von der grünen zu unterscheiden, da der Übergang nicht eindeutig erkennbar ist. Links hingegen ist es einfacher, weil weiß und grün sehr weit auseinander liegen.

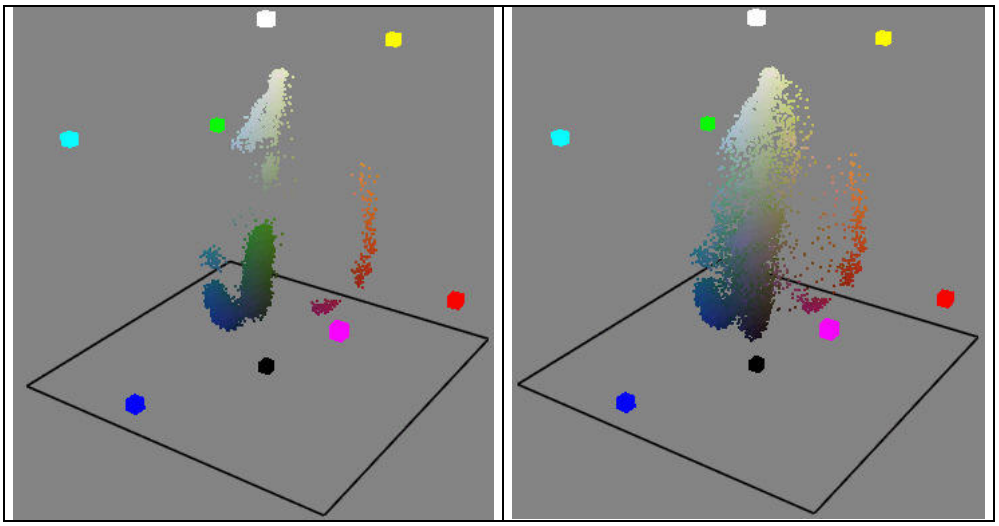

Abbildung 19: Links der Farbverlauf des gesamten Feldes ohne Zwischenfarben und rechts mit Zwischenfarben (Abbildung 18 oben links ohne Zuschauerbereich).

Ein Überlappen der Farben

Ein Überlappen der Farben ist aus der Perspektive in Abbildung 19 nicht erkennbar. Nur wenn rangezoomt wird, ist erkennbar, dass auch die Farben Grün und Blau getrennt liegen.

Auch in Abbildung 16 sind Farbunterschiede erst ab einem bestimmten Helligkeitswert erkennbar.

In machen Fällen kann es vorkommen, dass der Rasen oder sogar andere Objekte in den Ecken des Bildes als blau erkannt werden. Das liegt daran, dass die Kameraqualität nicht so gut ist und die Bilder an den Ecken Farbverfälschungen haben. Das ist korrigierbar, wenn vorher aus dem Bild die Farbverfälschung rausgerechnet wird.[26]

[26] Siehe [GTRe04] Abschnitt 3.2.3 „Camera Calibration“, Seite 27ff, und dazugehörige C++ Klasse \Src\Modules\ImageProcessor\ImageProcessorTools\ColorCorrector.h

7. Farbklassifikationsmodule

Farbklassifikationsmodule (ColorTableModule) dienen der Zuordnung eines Farbwertes zu einer Farbklasse. Vor der Benutzung des Farbklassifikationsmoduls müssen noch die Kameraeinstellungen vorgenommen werden.

Die Kameras haben dabei folgende Einstellungen[27]:

1. Weißausgleich – innen, fluoreszent, draußen (White balance – indoor, fluorescent, outdoor);
2. Kameraverstärkung – hoch, mittel, schwach (Camera gain – high, mid, slow);
3. Kameraverschlussgeschwindigkeit – schnell, mittel, langsam (Shutter speed – fast, mid, slow).

Sie sind in der Datei „\Src\Representations\Perception\CameraParameters.h“ implementiert und können mittels der Datei „\Src\Platform\Aperios1.3.2\Sensors.h“ an den Roboter gesendet werden.

Eine Farbverfälschung der Kamera kann mit der Klasse ColorCorrector in der Datei “ColorCorrector.h“ aus dem Verzeichnis „\Src\Modules\ImageProcessor\ImageProcessorTools\“ behoben werden.

7.1. ColorTable32 und ColorTable64

ColorTable64 ist in den Dateien „ColorTable64.h“ und „ColorTable64.cpp“ im Verzeichnis „\Src\Representations\Perception\“ implementiert. Im gleichen Verzeichnis befindet sich „ColorTable32.h“ und „ColorTable32.cpp“. ColorTable64 ist eine Weiterentwicklung des ColorTable32.

ColorTable64 ist ein dreidimensionales Array [64x64x64] für Y-, U- und V-Werte. Für jede Dimension ist ein Wertebereich von 0 bis 63 definiert. Da ein Pixel für jeden Wert einen Wertebereich von 0 bis 255 hat, wird jede Dimension bei der Farberkennung restlos durch vier geteilt.

In dem Array ist die entsprechende Zuordnung zu einer Farbklasse gespeichert.

ColorTable64 ist nur eine Erweiterung des ColorTable32, in dem von der Dimension [8x64x64] zu [64x64x64] gewechselt worden ist.

Vorteile:

- Die Kamera liefert Bilder im YCrCb-Format, daher ist keine Konvertierung nötig, und es gibt **keinen Performanceverlust**.
- Für ein Pixel sind nur drei restlose Divisionen und ein Speicherzugriff nötig.

[27] In OPENR von Sony sind mehr Einstellungen vorhanden. Hier werden nur mit GT2004 einstellbare Einstellungen angeschaut.

- **Speichersparend**, da ein Eintrag im Array für 64 (4x4x4) YUV-Werte steht. Speicherbedarf ist 64^3 = 262144 Werte. Wenn für jeden Wert ein Byte verwendet wird, dann sind das 256 KBytes. Sonst müssten 256^3 = 16777216 Werte verwendet werden, oder 16384 KBytes.

Nachteile:

- Statisch.
- Relativ mühsame Erstellung.
- Es bleiben immer „Löcher“ in der Matrix (Array), die bei der Kalibrierung entstehen.
- **Keine** Korrektur der Farbverfälschung der Kamera wird verwendet.
- Objektränder werden mitberücksichtigt.
- Schon bei mittleren Lichtänderungen werden Farben falsch oder nicht erkannt.
- „NoColor“ wird in Außenbereich eingesetzt und damit die normalen Farben überschrieben. Oder bei leichten Lichtänderungen wird „NoColor“ als normales Feldobjekt erkannt.
- Die Grenze zwischen zwei Farben ist unvorhersehbar. Sie muss oft frei positionierbar verlaufen und einstellbar sein und nicht parallel zur Achse verlaufen.
- Genauigkeit ist relativ, da ein Eintrag im Array für 64 (4x4x4) YUV-Werte steht.

Vorschläge:

- Es kann an der Stelle einer “Durch-Vier-Division“ eine Right Shift[28]-Operation eingesetzt werden, um Performance zu gewinnen.

7.2. ColorTableTSL

ColorTableTSL ist in den Dateien „ColorTableTSL.h“ und „ColorTableTSL.cpp“ im Verzeichnis „\Src\Representations\Perception\“ implementiert.

ColorTableTSL hat auch fast wie ColorTable64 ein dreidimensionales Array [32*64*64] für Y-, U- und V-Werte. Außerdem sind für jede Farbe T_{min}-, T_{max}-, S_{min}-, S_{max}-, L_{min}- und L_{max}-Werte definiert. Damit kein Performanceverlust auftritt, werden vor der Benutzung der ColorTableTSL die TSL-Werte in das Array transferiert. D.h. bei der Kalibrierung werden einzelnen Pixel aus dem YUV-Farbsystem in das TSL-Farbsystem konvertiert. Danach werden Bereiche (die Werte T_{min}, T_{max}, S_{min}, S_{max}, L_{min} und L_{max}) bestimmt und abgespeichert. Ein Roboter lädt die gespeicherte Datei und konvertiert die Bereiche in den YUV-Farbraum. Somit werden beim Zugriff auf ein Pixel nur drei Dimensionsanpassungen (Right Shift 2 oder Right Shift 3) und ein Speicherzugriff durchgeführt.

Vorteile:

- Die Kamera liefert Bilder im YCrCb-Format, daher ist keine Konvertierung nötig, und es gibt **keinen Performanceverlust im Echtzeitbetrieb**.
- Für ein Pixel sind nur drei Right Shifts und ein Speicherzugriff nötig.

[28] Bitwiese Left Shift und Right Shift Operatoren

- **Speichersparend,** da ein Eintrag im Array für 128 (8x4x4) YUV-Werte steht. Speicherbedarf ist 32*64*64 = 131072 Werte. Wenn für jedes Wert ein Byte verwendet wird, dann sind das 128 KBytes. Sonst müssten 256^3 = 16777216 Werte verwendet werden, oder 16384 KBytes.
- Eine Korrektur der Farbverfälschung der Kamera wird verwendet.
- Relativ schnelle Erstellung und leichte Anpassung.
- Bei mittleren Lichtveränderungen werden Farben gut erkannt.
- Reihenfolge der Farben ist einstellbar.

Nachteile:

- Statisch.
- Es bleiben immer „Löcher“ in der Matrix (Array), die bei der Kalibrierung entstehen.
- Objektränder werden mitberücksichtigt.
- Die Grenze zwischen zwei Farben im YUV-Farbraum ist unvorhersehbar.
- Korrektur der Farbverfälschung der Kamera funktioniert nicht gut.
- Genauigkeit ist relativ, da ein Eintrag im Array für 128 (8x4x4) YUV-Werte steht.
- Die Reihenfolge der Farben kann einige untergeordnete Farben benachteiligen. Zum Beispiel kann Pink so definiert werden, dass Rot nicht mehr erkannt wird.
- T – Histogramm funktioniert fehlerhaft.
- „Auto“ – Taste funktioniert fehlerhaft.

7.3. HSIColorTable

HSIColorTable ist in den Dateien „HSIDataTypes.h“ und „HSIDataTypes.cpp“ im Verzeichnis „\Src\RobotControl\HSITools\“ implementiert.

HSIColorTable verwendet direkt die Klasse ColorTable64. Außerdem sind für jede Farbe H_{min}-, H_{max}-, S_{min}-, S_{max}-, I_{min}- und I_{max}-Werte definiert. Damit kein Performanceverlust auftritt, werden vor der Benutzung der HSIColorTable die HSI-Werte in die Instanz ColorTable64 konvertiert, genau wie im TSLColorTable.

Somit werden beim Zugriff auf ein Pixel nur drei Dimensionsanpassungen (restlose Division durch 4) und ein Speicherzugriff durchgeführt.

Vorteile:

- Die Kamera liefert Bilder im YCrCb-Format, daher keine Konvertierung nötig, und es gibt **kein Performanceverlust im Echtzeitbetrieb**.
- Für einen Pixel sind nur drei restlose Divisionen und ein Speicherzugriff nötig.
- **Speichersparend,** da ein Eintrag im Array für 64 (4x4x4) YUV-Werte steht. Speicherbedarf ist 64^3 = 262144 Werte. Wenn für jeden Wert ein Byte verwendet wird, dann sind das 256 KBytes. Sonst müssten 256^3 = 16777216 Werte verwendet werden, oder 16384 KBytes.
- Relativ schnelle Erstellung und leichte Anpassung.

- Bei mittleren Lichtänderungen werden Farben schwächer als ColorTableTSL erkannt, dennoch relativ gut.

Nachteile:

- Statisch.
- Es bleiben immer „Löcher“ in der Matrix (Array), die bei der Kalibrierung entstehen.
- **Keine** Korrektur der Farbverfälschung der Kamera wird verwendet.
- Objektränder werden mitberücksichtigt.
- Die Grenze zwischen zwei Farben im YUV-Farbraum ist unvorhersehbar.
- Genauigkeit ist relativ, da ein Eintrag im Array für 64 (4x4x4) YUV-Werte steht.
- Die Reihenfolge der Farben ist nicht einstellbar.
- Die Reihenfolge der Farben kann einige untergeordnete Farben benachteiligen. Zum Beispiel kann Pink so definiert werden, dass Rot nicht mehr erkannt wird.

8. Dynamische versus statische Farbklassifikation.

Leider ist es so, dass die Performance an erster Stelle steht. Schon die einfachen Filter z. B. Sobel-Filter schaffen Performanceeinbußen. Es ist möglich, einen halbautomatischen Algorithmus zu implementieren. Die Roboter senden Bilder an einen Desktop-PC, der notwendige Berechnungen ausführt. Das ist leider nur außerhalb des Spieles erlaubt, stellt aber eine Basis zur automatischen Farberkennung dar. Außerdem bietet dies eine große Erleichterung bei der Einstellung. So könnten alle Roboter individuell eingestellt werden.

Bei statischen Tabellen muss zusätzlich Speicherkapazität berücksichtigt werden.

Der Vorteil statischer Tabellen liegt in ihrer Schnelligkeit. Dynamische Verfahren verlangsamen die Prozesse.

Der Nachteil statischer Tabellen ist allerdings, dass sie vor dem Spiel oder dem Challenge bestimmt werden müssen und **alle** zu erwartenden Lichtänderungen beinhalten müssen. Bei dynamischen Verfahren ist das nicht notwendig.

9. Analyse, Implementation und Einbindung in den Hamburg Dog Bots-Code

Dies ist eine kurze zusammenfassende Bewertung der Kapitel 6 bis 8:

Alle Objekte haben bestimmte Farbverläufe, die ab einer bestimmten Beleuchtung auseinanderliegen. Dabei müssen keine Kanten berücksichtigt werden. Der Zuschauerbereich muss bei der Kalibrierung nicht berücksichtigt werden. Aber das kann durchaus normale Farbklassen beinhalten. Bevor mit einem Bild gearbeitet wird, muss die Kameraverzerrung errechnet werden.

Alle oben beschriebene Module verwenden ColorTable64 (YUV-Farbraum) oder generieren selbst eine ähnliche Tabelle. Bei allen Verfahren wird nur die statische Farberkennung verwendet.

Alle oben beschriebene Bewertungen führen zu der Lösung: **Aufteilung in getrennte Farbräume und Generierung der ColorTable64 oder einer vergleichbaren Farbklassifikationsmodule.**

So kann der gesamte YUV-Farbraum in Unterräume aufgeteilt werden. Jeder Unterraum bestimmt eine Farbe. Das ist möglich mit Hilfe von Ebenen oder Trennwänden. Eine Trennwand kann mit vier Parametern beschrieben werden. Die Gleichung sieht so aus:

$$A*Y + B*U + C*V + D = 0$$

wobei A, B, C, D Konstanten sind, Vektor (A,B,C) ein Normalvektor der Ebene, und D ein Verschiebungsfaktor der Ebene vom Punkt (0,0,0) in Richtung des Normalvektors.

Es kann auch leicht festgestellt werden, ob ein Punkt (y,u,v) sich auf der einen oder der anderen Seite von der Ebene oder sogar auf der Ebene befindet. Dafür wird der Wert R bestimmt:

$$R = A*y + B*u + C*v + D.$$

- Wenn $R = 0$, dann ist der Punkt (y,u,v) auf der Ebene;
- Wenn $R > 0$, dann ist der Punkt (y,u,v) auf der Seite der Ebene, die in der Richtung vom Normalvektor aus der Ebene zeigt;
- Wenn $R < 0$, dann ist der Punkt (y,u,v) auf der anderen Seite der Ebene.

Weiter sind es nur zwei Fälle von Bedeutung: $R \geq 0$ und $R < 0$.

Somit ist schnell feststellbar (drei Multiplikationen, drei Additionen und ein Vergleich), ob ein Punkt sich auf der einen oder der anderen Seite befindet.

Trennwände müssen so eingesetzt werden, dass sie genau zwischen den Farbmengen liegen. Die Reihenfolge der Farben muss einstellbar sein.

Die Aufnahmen müssen bei allen Lichtverhältnissen gemacht werden.

So sieht die optimale Farbraumverteilung aus (Abbildung 20):

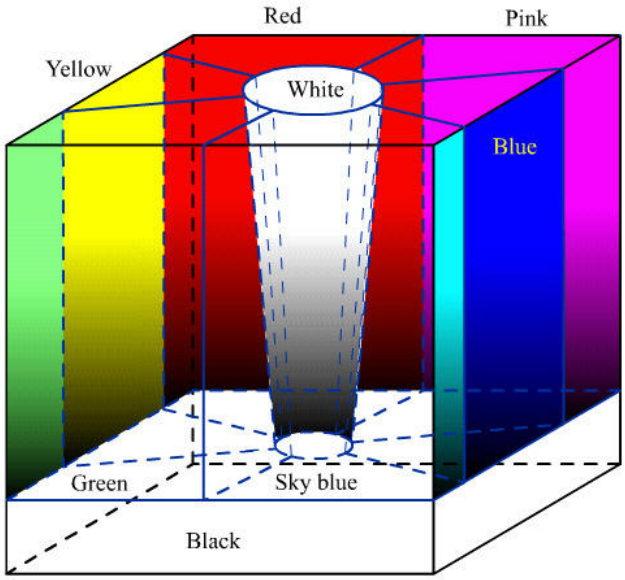

Abbildung 20: Optimale YUV–Farbraumverteilung.

Im unteren Bereich ist schwarz unabhängig von den U- und V-Werten durch eine Wand abgetrennt. Die Farben teilen den Raum in sechs gleiche Unterräume. Die weiße Farbe befindet sich in der Mitte und hat die Form eines abgeschnittenen - auf dem Kopf stehenden - Kegels, damit die Farbtemperatur und das Rauschen abgefangen werden können.

Ein Kegel kann mit drei Parameter beschrieben werden. In diesem Fall liegen die Y – Achse und die Kegelachse auf einer Geraden, was die Berechnung erleichtert. Punkt (0,0,0) befindet sich in der Mitte der Bodenfläche des YUV-Würfels. Parameter h –die Höhe des abgeschnittenen Kegels, r_1 – der untere Radius und r_2 – der obere Radius des Kegels. Dann ist der Wert R des Punktes (y,u,v) so zu bestimmen:

Wenn $y < (255 - h)$, dann ist die Farbe Schwarz (sehen Abbildung 20),

sonst

$$R = \mathrm{sqrt}(u^2 + v^2) - (r_1 + (r_2 - r_1)*(y - (255 - h))/h)$$

oder einfacher

$$R = \mathrm{sqrt}(u^2 + v^2) - r_1 - (r_2 - r_1)*(y - 255 + h)/h$$

wo $\mathrm{sqrt}(u^2 + v^2)$ die Entfernung des Punktes von der Y – Achse, $r_1 + (r_2 - r_1)*(y - (255 - h))/h$ der Radius des Kegels bei dem Wert Y = y ist.

- Wenn R = 0, dann ist der Punkt (y,u,v) auf dem Kegel;
- Wenn R > 0, dann ist der Punkt (y,u,v) außerhalb des Kegels;

- Wenn $R < 0$, dann ist der Punkt (y,u,v) innerhalb des Kegels.

Wie bei den Ebenen sind nur zwei Fälle von Bedeutung: $R \geq 0$ und $R < 0$.

Da diese Berechnung relativ kompliziert ist und nur im Idealfall auftritt, können es hier bei Ebenen, die mit UND-Operator verknüpft sind, angewendet werden. Somit entsteht eine konvexe Hülle. Die Berechnung wird vereinfacht, und es existiert nur eine Trennobjekt – Ebene. Außerdem kann die Farbverteilung im Normalfall so eingesetzt werden, dass zwischen den Farben Hohlräume entstehen, und nur die abgegrenzten Farben erkannt werden. Das ist praktisch, weil die Module höherer Schichten, die auf dem Farbklassifikationsmodul aufgebaut sind, Zwischenfarben (siehe Kapitel 6.2 Teil „Farben am Rande des Objektes“ Seite 27) erkennen und richtig interpretieren müssen, was noch nicht der Fall ist.

Die Implementierung der im Rahmen der vorliegenden Studienarbeit entwickelten Lösung basiert auf dem Programmcode des GermanTeam, wie er 2004 bei der Weltmeisterschaft in Portugal verwendet wurde. Um die Vergleichbarkeit mit dem vorliegenden Ansatz zu gewährleisten, wurde die vorgegebene Struktur des Codes beibehalten.

Es wurde ein neues Farbklassifikationsmodul implementiert, das mit Ebenen arbeitet. Das Modul heißt ColorTableVLCMod und verwendet ColorTable64. Das Modul läuft aufgrund der unbegrenzten Anzahl von Ebenen nur auf Windows-Systemen. Für Roboter wird ColorTable64 erzeugt und an Roboter gesendet, oder als eine Datei gespeichert.

Neu implementierte Dateien:

```
\Src\Modules\ColorTableMod\ColorTableVLCMod.h
\Src\Modules\ColorTableMod\ColorTableVLCMod.cpp
\Src\RobotControl\Bars\ColorTableVLCDlgBar.h
\Src\RobotControl\Bars\ColorTableVLCDlgBar.cpp
\Src\Representations\Perception\ColorTableVLC.h
\Src\Representations\Perception\ColorTableVLC.cpp
\Src\RobotControl\RES\undo.ico
\Src\RobotControl\RES\redo.ico
\Src\RobotControl\Visualization\SelectionPolygon.h
\Src\RobotControl\Visualization\SelectionPolygon.cpp
```

Geänderte Dateien:

```
\Make\RobotControl.dsp
\Src\Tools\Module\SolutionRequest.h
\Src\Modules\ColorTableMod\ColorTableModSelector.h
\Src\RobotControl\RobotControlMainFrame.cpp
\Src\RobotControl\RobotControl.rc
\Src\RobotControl\resource.h
\Src\RobotControl\RobotControlMenu.cpp
\Src\RobotControl\RobotControlMessageHandler.cpp
\Src\Tools\MessageQueue\MessageIDs.h
\Src\Processes\CMD\Debug.cpp
\Src\Modules\ImageProcessor\GT2004ImageProcessor\GT2004ImageProcessor.cpp
\Src\Tools\Debugging\DebugKeyTable.h
\Src\RobotControl\Visualization\OpenGLMethods.h
\Src\RobotControl\Visualization\OpenGLMethods.cpp
\Src\Processes\CMD\Cognition.cpp
\Src\Processes\CMD\Cognition.h
```

10. Ausblick: Automatisierung der Farberkennung

Eine Automatisierung der Farberkennung kann die mühsame manuelle Einstellung erleichtern oder überflüssig machen. Dann können alle Roboter individuell und schnell eingestellt werden. Dabei müssen folgende Schritte gemacht werden:

1. Kantenerkennung
2. Feststellung der mittleren Farben und deren Bereiches
3. Bestimmung Spielfeldbereichs
4. Zuordnung festgestellten Farben zu definierten Farbklassen
5. Abgrenzung festgestellten Farbklassen durch Trennwände
6. Generierung neue ColorTable64

Es kann ein halbautomatischer Algorithmus eingesetzt werden. Dabei sendet ein Roboter ein Bild (oder mehrere Bilder) an einen Desktop-PC. Ein Benutzer selektiert mit bestimmten Farben alle Objekte im Bild (in den Bildern). Somit fallen die ersten vier Schritte der automatischen Farberkennung aus. Der Desktop-PC führt Schritte fünf und sechs aus, und sendet eine generierte Tabelle an den Roboter.

Eine ausreichende Betrachtung dieses Themas würde den Rahmen dieser Arbeit sprengen.

11. Fazit

Zunächst wurde eine Analyse des Farbverhalten verschiedener Objekte bei diversen Belichtungsstärken durchgeführt und verglichen.

Danach wurden bestehende Module des GermanTeam Projektes untersucht und Vor- und Nachteile sowie Gemeinsamkeiten festgestellt.

Weiter wurden die Möglichkeiten eines dynamischen Verfahrens der Farbklassifikation angedeutet.

Dann wurde ein Verfahren entwickelt, das in der Lage ist, das gestellte Ziel optimal auf Basis des Hamburg Dog Bots Projektes zu erreichen.

Es wurde dabei die Möglichkeit der automatischen Einstellungsmöglichkeit untersucht.

Auf Basis der Hamburg Dog Bots Quellcodes wurde das Verfahren implementiert, in das Projekt integriert und erprobt.

Obwohl die Einstellung manuell erfolgt und noch schwierig ist, können Roboter sehr gut die Farben klassifizieren. Erste wichtige Tests gab es bei den German Open 2005 in Paderborn.

Und so sah das Ergebnis aus:

„**1.Platz!!!**
In der Variable Lighting Challenge - wo die Lichteinstellungen alle 20 Sek. verändert werden - konnte unser Spieler zwei mal ins Tor treffen. Als zusätzliches Hindernis zu den sich wechselnden Lichtverhältnissen gab es noch zwei passive Gegner. Diese wurden aber bei den gezielten Schüssen vom Mittelkreis beachtet und der Ball schlug zwei mal innerhalb 1:57 Minuten in den Kasten ein. Danach nahm er sich dann eine geistige Auszeit und versuchte den passiven Torwart richtig zu positionieren ,)

Der Stürmer der MS Hellhounds (Dortmund) konnte zwar auch zwei Treffer erzielen, brauchte aber länger (ca. 2:30 min)... Es gab insgesamt drei Minuten Zeit.

Damit sind wir deutsche Meister in dem inoffiziellen Wettbewerb der Variable Lighting Challenge!!!„[29]

Die Weiterentwicklung des automatischen Farberkennung bietet die Möglichkeit der schnellen, einfachen und individuellen Einstellung.

[29] http://www.20six.de/HDB-GermanOpen2005, 20.04.2005.

12. Literaturliste

[BäKr04] Bässmann, H., Kreyss, J. *Bildverarbeitung Ad Oculos*. Springer, Berlin, 4. aktualisierte Aufl., 2004.

[Beck92] Becker, B. *Künstliche Intelligenz : Konzepte, Systeme, Verheißungen*. Campus Verlag, Frankfurt am Main, 1992.

[BeMa02] Bennamoun, M., Mamic, G. J. *Object recognition : fundamentals and case studies*. Springer, London, 2002.

[Brei03] Breitner, M. H. *Nichtlineare, multivariate Approximation mit Perzeptrons und anderen Funktionen auf verschiedenen Hochleistungsrechnern.* Akademische Verlagsgesellschaft Aka, Berlin, 2003.

[Deng04] Dengel, A. *Reading and learning : adaptive content recognition*. Springer, Berlin, 2004.

[Fink03] Fink, G. A. *Mustererkennung mit Markov-Modellen : Theorie, Praxis, Anwendungsgebiete*. Teubner, Stuttgart, 1. Aufl., 2003.

[JüHL03] Jüngel, M., Hoffmann, J., Lötzsch, M. *A Real-Time Auto-Adjusting Vision System for Robotic Soccer*. Humboldt-Universität zu Berlin, Institut für Informatik, LFG Künstliche Intelligenz, 2003.

[Jüng04] Jüngel, M. *Using Layered Color Precision for a Self-Calibrating Vision System*. RoboCup2004 Symposium, Instituto Superior Técnico, Lisboa, Portugal, July 4-5, 2004.

[Müll03] Müller, R. *Vorlesungsscript: Graphische Datenverarbeitung*. Version vom 11.07.2003. Fachhochschule Aargau, http://www.cs.fh-aargau.ch/~gdv/script/script.pdf, 15.04.2004.

[Pass04] Passow, J. *Diplomarbeit: Geometrieidentifikation aus den Kamerabildern mobiler autonomer Systeme in der Sony Four-legged Robot League*. Universität der Freien und Hansestadt Hamburg, Fachbereich Informatik, 2004.

[Paul01] Pauli, J. *Learning-based robot vision : principles and applications.* Springer, Berlin, 2001.

[PeWe03] Petkov, N., Westenberg, M. A. *Computer analysis of images and patterns : 10th international conference ; proceedings / CAIP 2003, Groningen, The Netherlands, August 25 - 27, 2003*. Springer, Berlin, 2003.

[SrSt04] Sridharan, M., Stone, P. *Towards Illumination Invariance in the Legged League*. RoboCup2004 Symposium, Instituto Superior Técnico, Lisboa, Portugal, July 4-5, 2004.

[Robo05] *RoboCup Official Site*. http://www.robocup.org, 7.04.2005.

[Wiki05] *Wikipedia, die freie Enzyklopädie*. www.wikipedia.de, 03.03.2005.

[Four05] *Website - Sony Four-Legged Robot League*. www.tzi.de/4legged/bin/view/Website/WebHome, 3.04.2005.

[Rule05] *Sony Four Legged Robot Football League Rule Book*. www.tzi.de/4legged/pub/Website/History/Rules2005.pdf, 11.04.2005.

[CcdS05] *CCD-Sensoren*. http://www.ccd-sensor.de, 14.04.2005.

[Chal05] *Challenges der Sony Four-Legged Robot League.* www.tzi.de/4legged/pub/Website/Downloads/Challenges2005.pdf, 1.04.2005.

[GTRe04] *The team report describes the robot code and all tools developed by the GermanTeam for the RoboCup 2004*. http://www.germanteam.org/GT2004.pdf, 11.04.2005.

13. Anhang

13.1. What is RoboCup?[30]

RoboCup is an international research and education initiative. Its goal is to foster artificial intelligence and robotics research by providing a standard problem where a wide range of technologies can be examined and integrated.

The concept of soccer-playing robots was first introduced in 1993. Following a two-year feasibility study, in August 1995, an announcement was made on the introduction of the first international conferences and football games. In July 1997, the first official conference and games were held in Nagoya , Japan . Followed by Paris , Stockholm , Melbourne and Seattle where the annual events attracted many participants and spectators. The 6 th RoboCup2002 was held in Fukuoka , Japan in cooperation with Busan , Korea , while the 7 th edition in 2003 took place in Padua , Italy, then in 2004 in Lisbon, Portugal. The events were covered by national and international media all over the world.

RoboCupSoccerThe main focus of the RoboCup activities is competitive football. The games are important opportunities for researchers to exchange technical information. They also serve as a great opportunity to educate and entertain the public. RoboCupSoccer is divided into the following leagues:

	**Simulation league** Independently moving software players (agents) play soccer on a virtual field inside a computer. Matches have 5-minute halves. This is one of the oldest fleet in RoboCupSoccer.
	**Small-size robot league (f-180)** Small robots of no more than 18 cm in diameter play soccer with an orange golf ball in teams of up to 5 robots on a field with the size of bigger than a ping-pong table. Matches have 10-minute halves.
	**Middle-size robot league (f-2000)** Middle-sized robots of no more than 50 cm diameter play soccer in teams of up to 4 robots with an orange soccer ball on a field the size of 12x8 metres. Matches are divided in 10-minute halves.

[30] http://www.robocup.org/Intro.htm, 21.04.2005

	**Four-legged robot league** Teams of 4 four-legged entertainment robots (SONY's AIBO) play soccer on a 3 x 5[31] metre field. Matches have 10-minute halves.
	**Humanoid league** This league was introduced in 2002 and the robots will have their third appearance ever in this year's RoboCup. Biped autonomous humanoid robots play in "penalty kick," and "1 vs. 1", " 2 vs. 2" matches. "Free style" competitions are to be expected as well.

[31] Auf der Quellseite stehen noch die alte Angaben. Richtig ist „6 x 4 Meter“. Siehe dazu Spielregeln 2005 und Figure 2.

13.2. The Variable Lighting Challenge[32]

This second challenge is intended to encourage teams to increase the robustness of their vision to illumination changes. It is based on a penalty shoot out. The team attempting the challenge places a single *blue robot* (robot with a blue uniform) on the field. That robot must score as many goals as it can into the yellow goal in three minutes. The team that scores the most goals wins. If two teams score the same number of goals, then the team with the lowest average time to score a goal wins (Note: this is the same as choosing the team who scored their last goal earliest). If no team scores, then the team with the ball closest to the goal at the end of their time wins.

In addition to the single blue robot, two red *opponent* robots are also placed on the field. Both of these robots are paused, frozen in the *UNSW stance*. One is placed in a goalie position on one side of the yellow goal. The other is placed in the third of the field nearest the yellow goal, at least 30cm away from the edge. The exact locations of all the robots shall be determined by the referee, and will be the same for all teams. Figure 1 shows one possible situation.

There is a single ball upon the field. Initially it is placed in the center kickoff position. Upon each score, the ball is moved back to the center kickoff position. The robot is not moved by the referee and must make its own way back to the center of the field to reach the ball again. The robot will receive a message from the game controller with its new score. Since the field does not have borders anymore, it is very likely that the ball may be kicked out of the field. In this case, it is placed back at the *drop in* point (see the document specifying the rules for more details) closest to the point where the ball was kicked out of bounds.

The main challenge in this task is that the illumination shall be different from standard RoboCup lighting. Some additional lights and suitable equipment shall be brought in to supply variable lighting conditions (we envisage variable strength lighting using theatrical lighting equipment). These additional lights shall be *white* light of deliberately unspecified color temperature. Lights may also be covered to achieve variable lighting conditions.

Before the challenge the referee shall prepare a schedule of illumination changes. This shall include periods of constant illumination, periods of slow change in lighting and periods of rapid changes in lighting. It is envisaged that the additional lighting will be non-uniform across the field and hence the lighting changes will be non-uniform. This lighting schedule, though unknown until right before the challenge, shall be the same for all teams.

Unless otherwise specified, normal penalty shootout rules apply. There will be no penalty for charging the *opponent* robots. But, it is *not* allowed to help a robot stop charging and move away from or around another robot.

[32] http://www.tzi.de/4legged/pub/Website/Downloads/Challenges2005.pdf, 21.04.2005

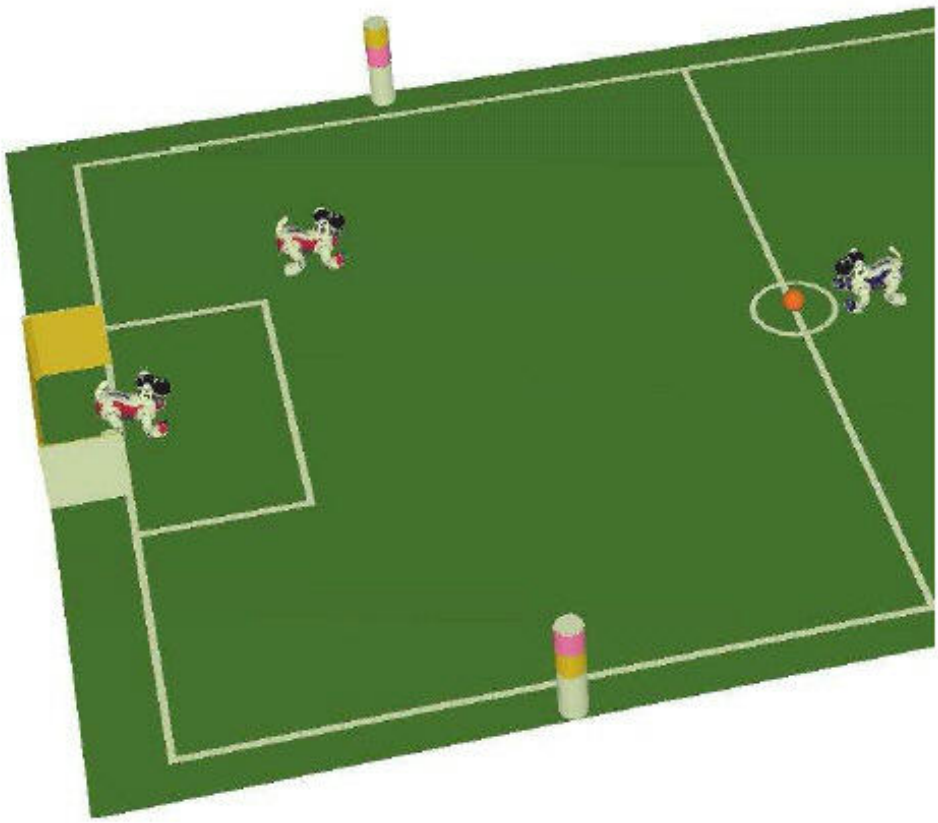

Figure 1: An example placement of opponent robots for the variable lighting challenge.

13.3. Sony Four Legged Robot Football League Rule Book[33]

RoboCup Technical Committee

(As of December 2, 2004)

Contents

1. Setup of the Environment 46
2. Robot Players 48
3. Game Process 51
4. Forbidden Actions and Penalties 58
5. Judgement 64
6. Questions/Comments 66

1. Setup of the Environment

1.1. Field Construction

The dimensions of the soccer field are shown in Figure 2. The construction of the goals and the beacons is depicted in Figure 4.

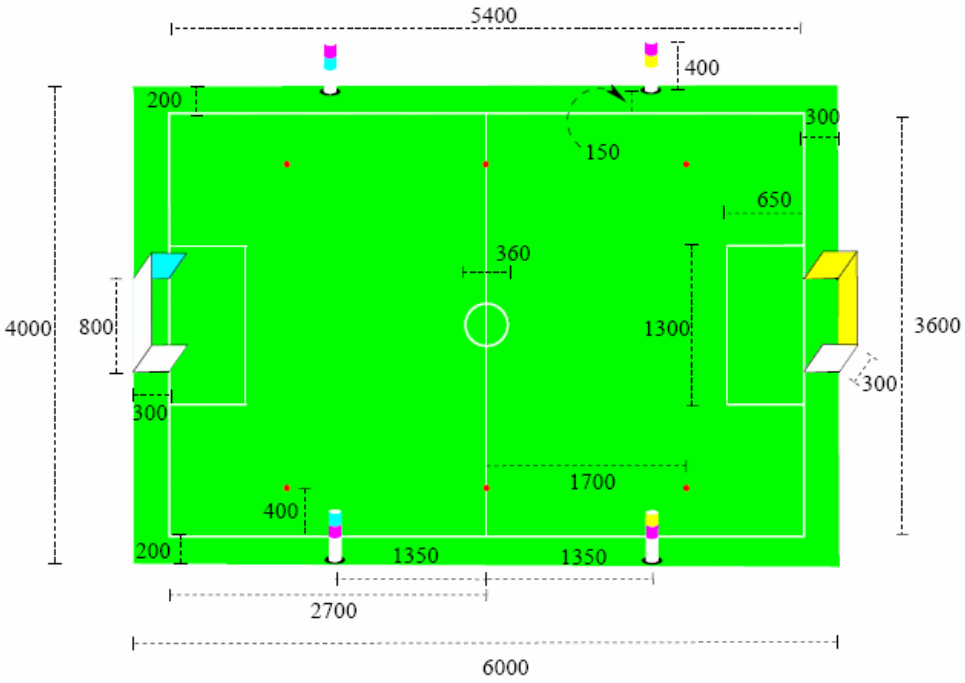

Figure 2: Field dimensions in mm.

[33] http://www.tzi.de/4legged/pub/Website/History/Rules2005.pdf, 21.04.2005

1.2. Lines

All the lines on the soccer field (the halfway line, the lines surrounding the penalty areas, the goal lines, and the center line) are drawn with white stripes of 25 mm in width. The circle on the midfield line has a diameter of 360 mm from the middle of the white stripe on one side to the middle of the white stripe on the other side.

1.3. Field Colors

The colors of the soccer field are shown in Figure 5. All items on the RoboCup field are colorcoded:

- The field (carpet) itself is green.
- The lines on the field are white.
- The red team defends the yellow goal.
- The blue team defends the sky-blue goal.
- Cylindrical beacons (cf. Figure 4b) are placed on the edge of the field and 1350mm from halfway. Part C is always white. When looking from the yellow goal toward the sky-blue goal, part B of the two beacons on the right of this axis shares the color with the neighboring goal, and part A is pink. On the left side of the field, part B of the beacons is pink and part A shares the goal color.

1.4. Lighting Conditions

The lighting conditions depend on the actual competition site. In 2005, they may differ significantly from previous years, because only ceiling lights are used. The lighting will be approximately 1000 lux, i. e. it will be similar to that used in previous years.

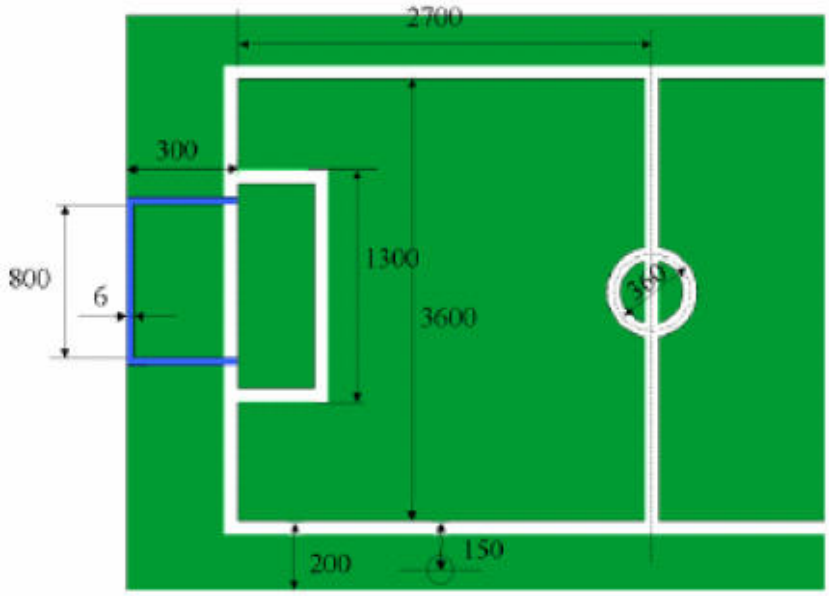

Figure 3: 2D picture indicating field dimensions in mm.

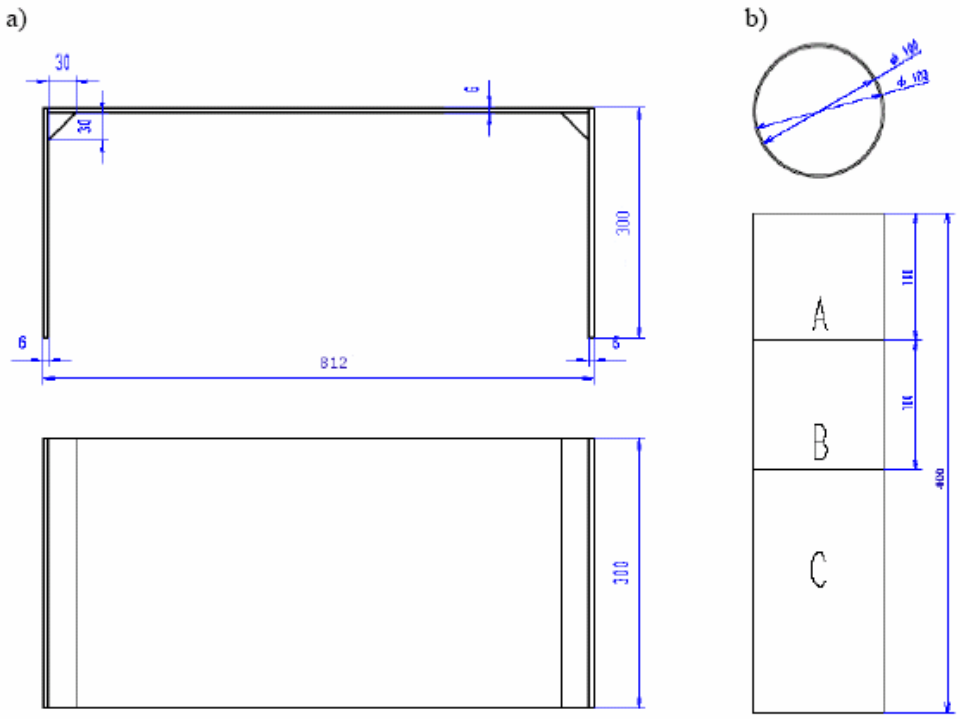

Figure 4: Top and side views of the goals and the beacons. a) Goal. b) Beacon.

2. Robot Players

2.1. Hardware

All teams must use robots of the following types:

- Sony AIBO ERS-210 (black (i. e. dark gray) or white)
- Sony AIBO ERS-210A[34] (black (i. e. dark gray) or white)
- Sony AIBO ERS-7 (white)
- Sony AIBO ERS-7M2[35] (white)

The ERS-7M2 or ERS-7 are recommended as they have a faster processor, a higher camera resolution, and they are the only models currently being sold that are permitted in RoboCup. Absolutely no modifications or additions to the robot hardware are allowed. No additional hardware is permitted including off-board sensing or processing systems. Additional sensors besides those originally installed on the robots are likewise not allowed. The only exceptions are:

- Attaching the red or blue team markers provided by Sony to the robots.
- Attaching the jersey numbers provided by the league to the robots.
- Adding some foam or sticky tape to the inner side of the cover of the battery compartment to prevent the battery from falling out.

[34] The identifier *ERS-210* is used for any version of the ERS-210(A/B) in this document.
[35] The identifier *ERS-7* is used for any version of the ERS-7(M2) in this document.

A computer will be provided by the event organizers for the purpose of sending GameController messages to the robots.

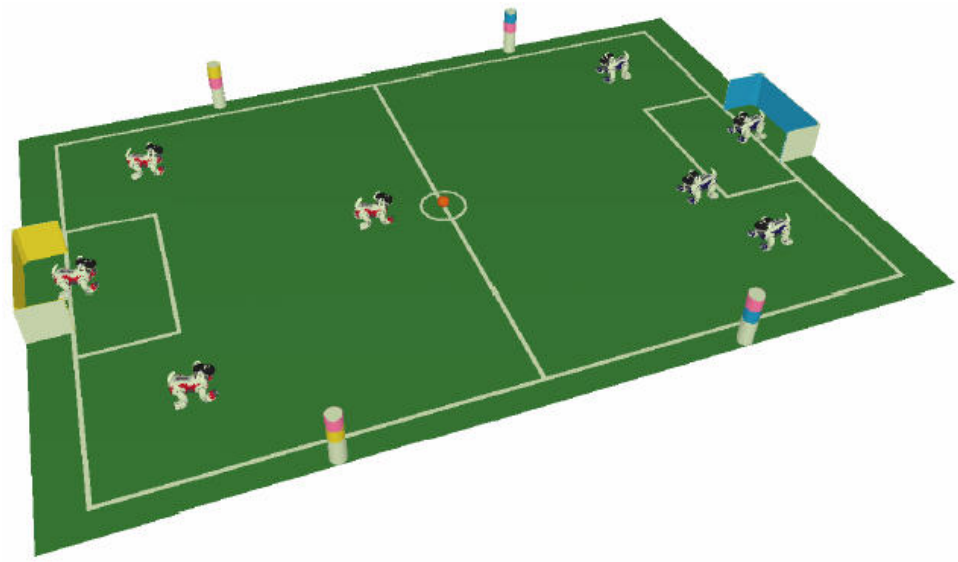

Figure 5: Field colors and manual setup for kick-off.

2.2. Teams

Each team consists of no more than 4 robots including the goal keeper.

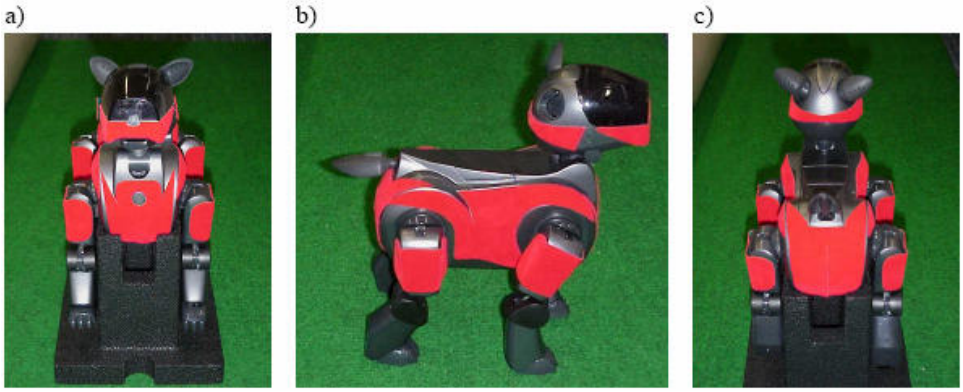

Figure 6: ERS-210 team markers. a) Front view. b) Side view. b) Back view.

2.3. Goal Keeper

The goal keeper is the only player that is allowed to stay within the penalty area of its own team. It always has the jersey number “1” attached to its back.

2.4. Field Players

The field players are not allowed to enter their own penalty area. The three field players robots have the jersey numbers “2”, “3”, and “4” attached to their backs.

2.5. Team Markers

Red and blue team markers will be provided to each team. All markers must be attached to each robot playing in a game (cf. Figure 6 and Figure 7). As opponent players may try to detect these markers, playing with any markers missing is not allowed.

2.6. Communications

2.6.1. Acoustic Communications

There are no restrictions on communication between the robots using a microphone or a speaker.

2.6.2. Wireless Communications

The only wireless hardware allowed to be used by the teams are the wireless network cards built into the AIBOs, and the access points provided by the event organizers. Or other wireless hardware must be deactivated. A team may be disqualified if one of the team members violates this rule. The MAC-addresses of all AIBOs participating in the competition will be registered. Only these MACaddresses can be reached through the access points provided by the event organizers. In addition, the access points will be secured by different SSIDs and WEP-keys. Two of the access points will be connected to PCs running GameController. A third access point is used for practise. It is connected to a hub with one port for each team. Teams must bring their own ethernet cables.

Each team will get a range of IP-addresses that can be used both for their robots and their computers. The IP-addresses, channels, SSIDs, and WEP-keys of the fields will be announced at the competition site.

Teams can use a bandwidth of up to 500 Kbps of the wireless. This includes any data transferred, namely the actual payload and any protocol overhead created, e. g., by TCP, UDP, or the Game-Controller.

tcpGateway is **no longer** supported. Teams may still choose to use tcpGateway but it will no longer be a requirement for GameController.

Any form of wireless robot-to-robot communication is allowed, as long as it uses the access points provided by the event organizers (using the so-called ad-hoc mode is prohibited), it does not conflict with TCP/IP or UDP, and the maximum bandwidth allowed for each team is not exceeded. Each team will be assigned a range of 256 IP-addresses that can be used for direct robot-to-robot communication. Each team will also be allocated a limited range of UDP ports on which broadcast will be permitted.

The GameController will use UDP to connect to the robots. The source distribution of the GameController provides the header file *RoboCupGameControlData.h* that defines all messages sent by the GameController to the robots. They correspond to the *robot states* described in Section 3.2.

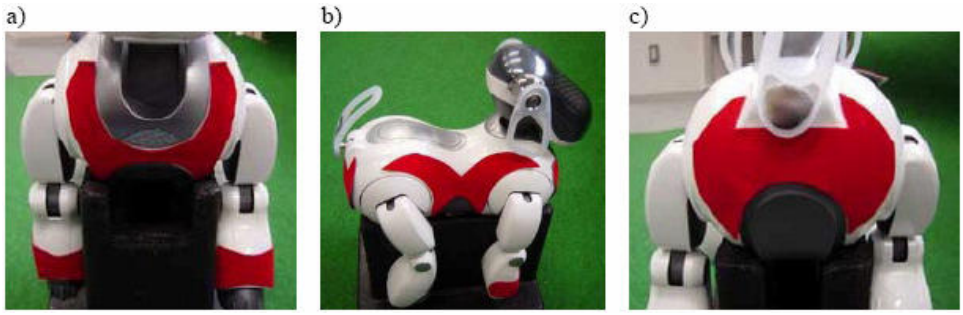

Figure 7: ERS-7 team markers. a) Front view. b) Side view. c) Back view.

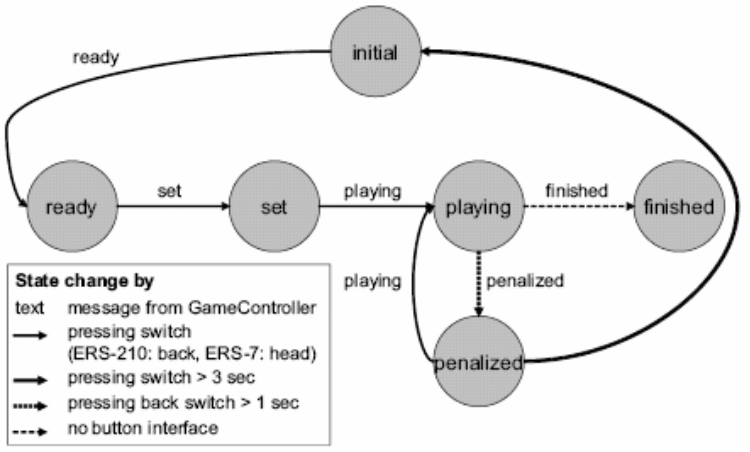

Figure 8: Robot states.

The use of remote processing/sensing is prohibited.

3. Game Process

3.1. Structure of the Game

A game consists of three parts, i. e. the first half, a half-time break, and the second half. Each half is 10 minutes. The clock stops during stoppages of play[36] (such as kick-offs after goals). The extra time over ten minutes total is referred to as “lost time”. The half-time break is also ten minutes, during this time both teams may change robots, change programs, or anything else that can be done within the time allotted. In the preliminaries a game can finish in a draw as no extra time or penalty shoot-out will follow. In the finals a game that ends in a draw will be followed by

[36] This may not be the case during the preliminary games.

five minutes of extra time in which the team that scores first wins (golden goal). There is no break between the second half and the extra time, i. e. the game will simply continue without a kickoff or restart. If no team scored a golden goal, a penalty shoot-out employing sudden death (see Section 3.8) is started after a 5 minute break.

The teams will change the goal defended and color of the team markers during the half-time break.

3.2. Robot States

Robots can be in six different states (cf. Figure 8). If the wireless is available, these states will be set by the GameController. However, as the availability of the wireless cannot be guaranteed, it must also be possible to switch between the states manually. As the assistant referees handle the robots, the button interface must be the same for all teams. However, it cannot be the same for all robot types, because the ERS-210 and the ERS-7 differ in the number of the buttons they have and the sensitivity of these buttons. For most switching between states, the back switch is used on the ERS-210 while the head switch is used on the ERS-7. The only exception is switching from the playing state into the penalized state, which is done with the back switch(es) for both robots, because it would be too complicated to continuously press the switch on a moving head for a longer period of time.

Initial. After booting, the robots are in their *initial* state. In this state, the button interface for manually setting the team color and whether the team has kick-off is active. The robots are not allowed moving in any fashion besides initially standing up.

ERS-210. The front part of the head switch toggles between the own kick-off and the opponent kick-off, while the top LED displays the current selection (on: own kick-off). The back head switch toggles between the red and blue team color. The team color is displayed by the LED in the robot's tail during the whole game (orange for the red team, blue for the blue team).

ERS-7. The middle back switch toggles between the own kick-off and the opponent kickoff, while the orange LED around this switch displays the current selection (on: own kick-off). To select the blue team color, press the front back switch, to select the red team color, press the rear back switch. The team color is displayed by the LEDs surrounding these switches, i. e. the red LED is on when the red team color is selected and the blue LED is switched on when the robot is part of the blue team.

Pressing the back switch of an ERS-210 or the head switch of an ERS-7 will switch it to the *ready* state.

Ready. In this state, the robots walk to their legal kick-off positions (cf. Section 3.5). They remain in this state, until all the robots have reached legal positions and have stopped, or until the head referee decides that there is no significant progress anymore. By

pressing the back switch of an ERS-210 or the head switch of an ERS-7, it will be switched into the *set* state.

Set. In this state, the robots stop and wait for kick-off (cf. Section 3.5). If they are not at legal positions, they will be placed manually by the assistant referees to the positions shown in Figure 5. They are allowed to move their heads and tails before the game (re)starts but are not allowed moving their legs or locomote in any fashion.

Pressing the back switch of an ERS-210 or the head switch of an ERS-7 will bring them into the *playing* state, e. g. for kick-off.

Playing. In the *playing* state, the robots are playing soccer. When at least one of the back switches is pressed for more than one second, the robot is switched into the *penalized* state. Please note that this will happen automatically in most cases when an ERS-7 is picked up.

Penalized. A robot is in this state when it has been penalized. It is not allowed moving in any fashion, i. e. also the head has to stop turning. From this state, the robot can be reset to the *initial* state by pressing the back switch (ERS-210) or the head switch (ERS-7) for more than three seconds. If these switches are pressed shorter than that, the robot is returned into the *playing* state.

Finished. This state is reached when a half is finished. There is no possibility to reach this state when wireless is not working.

Without the wireless, the procedure after a goal has been scored is a little bit complex. The robots have first to be switched into the *penalized* state, from which it is possible to return to the *initial* state. From there, everything is the same as before the first kick-off.

The team color should be displayed during the whole game. The selection whether the robot's team has kick-off should be visible in the states *initial*, *ready* and *set*. Both selections should also be visualized if the wireless is working.

3.3. Goal

A goal is achieved when the entire ball (not only the center of the ball) goes over the field-side edge of the goal line, i. e. the ball is completely inside the goal area[37]. The restart after the goal shall adopt the same rules as the kick-off.

3.4. Applying Penalties

See Section 4.1.

[37] The goal line is part of the goal area.

3.5. Kick-off

For kick-off, the robots run through three states: *ready*, *set*, and *playing*. In the ready state, they should walk to their legal kick-off positions. These positions are always located inside their own side of the field. Only two field players of the attacking team can walk to positions between the center line and the middle of their side. They may put their leg(s) on the center circle line, but no leg may be inside the circle line. The other field players (one of attacking team, three of defending team) have to be located behind the middle of their side (none of their legs are allowed to go beyond the middle), but have to stay outside their own penalty area with at least two feet. In contrast, the goal keepers have to stay *inside* the penalty areas with at least two feet.

If robots collide during the autonomous placement, the "Goalie Pushing" and the "Field Player Pushing" rules are applied, but the penalty is manual placement by the assistant referees.

If all the robots have reached legal positions and have stopped, or if the head referee decides that there is no significant progress anymore, they will be switched into the *set* state (by the GameController or manually by the assistant referees), in which they must stop walking. Each robot that is not at a legal position at this point in time will be placed manually by the assistant referees to the positions as shown in Figure 5. Robots that are legally positioned will not be moved by the assistant referees unless a manual position is requested by the team leader.

There are extra restrictions on the legal positions of manually positioned robots. The kicking-off robot shall be one robot length away from the center circle, while one robot of the other team shall be just in front of one corner of the penalty area. The other robots shall be on the left and on the right of their own penalty area. As autonomously placed robots are allowed to be much closer to the ball, successful autonomous placement results in a significant advantage over manual placement.

After all robots are at legal positions, the ball is placed on the center point of the center circle by one of the referees.

After the head referee has signaled the kick-off, the robot's state is switched to *playing* (again either by the GameController or manually), in which they can actually play soccer.

Note that a goal can never be scored directly from a shot from the kick-off. See Section 4.4 for details.

If the assistant referees have misconfigured the robots (e. g. they set the wrong team color), the kick-off is repeated. In general, goals scored with at least one misconfigured robot on the field are not counted. The time that was played with a wrong configuration is counted as "lost time", i. e. the half should be lengthened by it.

3.6. Free Kick

None.

3.7. Penalty Kick

A penalty kick is carried out with one attacking robot and one goalie. Other robots should be powered off and stay outside of the field. Teams are allowed to switch to specially designed software for a penalty kick. The attacker has 1 minute in which to score a goal. If the ball leaves the playing area it is not replaced and this penalty kick attempt is deemed unsuccessful.

Standard penalty kicks are taken against the opponent goal (i. e. a blue robot will attack the yellow goal).

The ball is placed 1m from the centre of the field in the direction of the defender's goal. The attacking robot is positioned 20cm behind the ball. The goalie is placed with its front legs on the goal line and in the centre of the goal. Neither robots shall not move their legs before the penalty kick starts. Movements of the robot's head and tail are allowed as long as the robot does not locomote. Technically, the robots are in the *set* state when waiting for the penalty kick to start. The robots are started by setting it into the *playing* state, which can be done either via the wireless or manually by an assistant referee. The penalty kick ends when the kicker scores the goal, the time expires or the ball leaves the field. The time limit for the kicker is 1 minute after the penalty kick starts. The ball must be in the goal within this time limit in order to count as a goal.

The goalie is not allowed to leave the penalty area (i. e. 3 legs outside) and the attacking robot is not allowed to enter the penalty area (i. e. 3 legs inside the area). If the attacker enters the penalty area then the penalty shot is deemed unsuccessful. If the goalie leaves the penalty area then a goal will be awarded to the attacking team.

All the rules such as "Ball Holding", "Goalie Pushing" and others are also applied during the penalty kick. The only exception is the "Illegal Defender" rule, i. e. the penalty shooter is allowed to enter its own penalty area.

3.8. Penalty Kick Shoot-out

A penalty kick shoot-out is used to determine the outcome of a tied game after the extra time. The penalty kick shoot-out is a sudden death contest. Each team selects an attacking robot and a memory stick to be used in the penalty kick shoot-out. They also select a goalie robot and a memory stick for this robot. The memory sticks cannot be changed in between kicks.

In a penalty kick shoot-out, all penalty kicks are taken against the *blue* goal. This will require each team to do a jersey colour swap on one robot before the penalty shoot-out can start.

The actual procedure is the same as for the normal penalty kick described in Section 3.7. If one team scores within the allotted time and the other does not, the scoring team wins. For the first four attempts, the time limit for the kicker is 1 minute after the penalty kick starts, the same as the normal penalty kick. However, for the fifth attempt it is two minutes. The fifth attempt is timed, i. e. if both teams score, the faster team wins, but only if the faster team is at least two seconds faster than the other team. Otherwise, the timed attempt is repeated.

3.9. Throw-in

If the ball leaves the field it will be replaced using the "boy" model. The game does not stop and the assistant referee will immediately place the ball onto one of six known points on the field.

These six points are shown in red on Figure 2, 'b' points are the two points closest to the blue goal, 'o' points are those on the halfway line and the 'y' points at those closest to the yellow goal.

A ball that last touched a blue robot before leaving the field is replaced at either a 'b' or an 'o' position on field, as follows:

- A ball kicked out in the blue half of the field is placed on a 'b'.
- A ball kicked out in the yellow half of the field is placed on an 'o'.

A ball that last touched a red robot before leaving the field is replaced at either a 'y' or an 'o' position on field, as follows:

- A ball kicked out in the yellow half of the field is placed on a 'y'.
- A ball kicked out in the blue half of the field is placed on an 'o'.

In all cases, the ball is placed on the appropriately named point closest to where it rolled out, i. e., on the same side that it rolled out on. Balls will be placed 'near' the specified point in such a way that no robots are within 50 cm of the ball, i. e. the ball does not have to be placed directly on the "boy" point but as close to the point as possible.

Note that goal lines and side lines are not treated differently. The above rules apply regardless of which line the ball rolled over.

Example 1. The red goalie kicks the ball over its goal line to the right of the yellow goal. The ball is placed on the right 'y' point.

Example 2. A blue attacking robot kicks the ball over the goal line to the right of the yellow goal. The ball is placed on the right 'o' point.

Example 3. A blue robot kicks the ball over the left sideline just into the yellow half of the field. The ball is replaced on the left 'o' point.

Example 4. A blue robot kicks the ball over the left sideline just inside the blue half of the field. The ball is replaced on the left 'b' point.

In a case where the referee is unable to determine who last touched to ball, the ball will be replaced at the nearest "boy" position. Balls are deemed to be out based on the team that last touched the ball, irrespective of who actually kicked the ball.

Example. A blue robot kicks the ball but the ball touches a red robot before leaving the field over the right yellow goal line. The ball is regarded as being out off red and therefore is replaced at the right 'y' point.

3.10. Game Stuck

3.10.1. Local Game Stuck

In the event of no substantial change in the game state for 30 seconds, the referee shall pick up the robots which are jamming around the ball and move them to the half way line. The referee does *not* replace the ball. If the ball is accidentally bumped when removing the robots, the ball should be replaced where it was when the game stuck was called. As a special exception, if the goalie is involved in a game stuck situation while having 3 feet on its own half (on half line or closer to goal), the goalie will not be removed from the game stuck situation. In this case, only the other robots involved will be removed.

3.10.2. Global Game Stuck

If no robot touches the ball for 30 seconds, the referee shall stop the game and restart the game from the kick-off formation. The kick-off will be awarded to the team defending the side of the field the ball is on when the game stuck is called.

3.11. Request for Pick-up

Either team may request that one of their players be picked up only for hardware dysfunction and software crashes at any point in the game (called “Request for Pick-up”). It is permitted to change batteries, fix mechanical problems or, if necessary, reboot the robots, but not to change or adjust their program. Any strategic “Request for Pick-up” is not allowed. The head referee will indicate when the robot is no longer affecting play and can be removed from the field by an assistant referee. The robot will be replaced on the half way line after 30 seconds following the normal replacement procedure used after the standard removal penalty (see Section 4.2).

If a robot has been rebooted and the wireless is not working, it is the responsibility of the team members (not the assistant referees) to configure its team color correctly.

3.12. Request for Timeout

At any stoppage of play (after a goal, stuck game, before half, etc.) either team may call a timeout. Each team can call a maximum of 1 timeout per game with a total time totaling no more than 5 minutes. During this time, both teams may change robots, change programs, or anything else that can be done within the time allotted. The timeout ends when the team that called the timeout says they are finished, at which time they must be ready to play. At this time the other team must either be ready to play or call a timeout of its own. The clock stops during timeouts, even during the preliminaries.

After the completion of the timeout, the game resumes with a kick off for the team which did not call the timeout.

If a team is not ready to play at the assigned time for a game, the referee will call the timeout for that team.

3.13. Winner and Rankings

The team which scored more goals than the other is the winner of the match. If the two teams scored the same number of goals, the game will be a draw. The draw will follow the same system defined in Section 3.1. Total (and final) standings will be decided on points as follows (the points will be given based on the result of each game):

Win = 3 pts Draw = 1 Lose = 0 pts

If a team's obtained points is the same as another team's after the preliminary round is complete, the following evaluations will be applied in order to qualify the finalists.

1. The points obtained
2. The average difference between goals for and goals against per game
3. The average goals for per game
4. Game result between the teams directly

3.14. Rules for Forfeiting

If a team chooses to forfeit a match then the result will be 10-0 against the team that forfeited. Teams may choose to forfeit games at any stage. Any game with a final score differential greater than 10 points will be considered a forfeit.

4. Forbidden Actions and Penalties

The following actions are forbidden. In general, when a penalty applies, the robot shall be replaced, not the ball. For penalties that are timed, the penalty time is considered to be over whenever the game time stops (for goals, half-time, and game stuck).

4.1. Penalty procedure

When a robot commits a foul, the head referee shall call out the infraction committed, the jersey color of the robot, and the jersey number of the robot. Each robot will be labeled with a jersey number before the game. The penalty for the infraction will be applied immediately by an assistant referee. The assistant referees should perform the actual movement of the robots for the penalty so that the head referee can continue focusing on the game. The operator of the GameController will send the appropriate signal to the robots indicating the infraction committed.

4.2. Standard Removal Penalty

Most infractions in this league result in the removal of the infringing robot from the field of play for a period of time. This process is called the standard removal penalty. When the head referee indicates a foul has been committed that results in the standard removal penalty, the assistant referee closest to the robot will remove the robot immediately from the field of play. The robot should be removed in such a way as to minimize the movement of the other robots and the ball. If

the ball is inadvertently moved when removing the robot, the ball should be replaced to the position it was in when the robot was removed.

The operator of the GameController will send the appropriate signal to the robot indicating the infraction committed. If the wireless is not working and the penalty is timed, the assistant referee handling the robot will reset the robot into the *penalized* state for the duration of the penalty. This is not done if the penalty is not timed, i. e. it is a 0 seconds penalty. After a penalty was signaled to the robot, it is not allowed moving in any fashion, such as being in the *initial* state. The removed robot will be placed outside of the field facing away from the field of play. The best option is to put them on robot stands that are located behind the goals.

The GameController will keep track of the time of the penalty. The operator of the GameController will signal the assistant referees when the penalty is over, so that one of them can put the robot back on the field. The assistant referee will then place the robot on the field on the halfway line as close to the sideline as possible. The robot should be pointed towards the opposite sideline. The robot should be placed on the side of the field furthest from the ball. If there is another robot already in this position, the robot should be replaced at a nearby location along the sideline facing towards the opposite sideline. If there are no practical locations nearby, a location along one of the sidelines should be found that is away from the ball (the robot should be set down facing the opposite sideline). When finding a nearby location, locations away from the ball should be preferred.

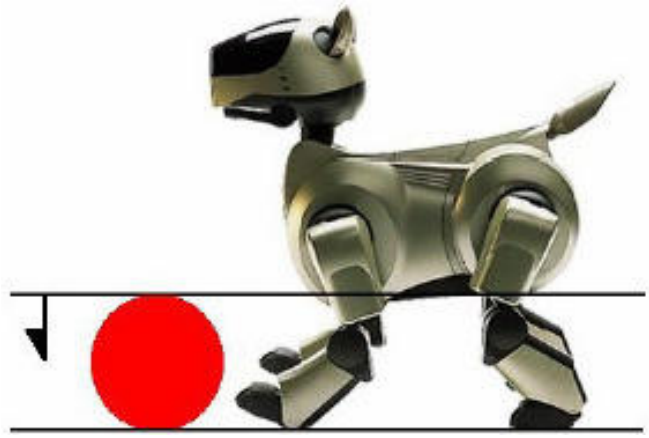

Figure 9: Robot parts which are taken into account for "Ball Holding".

When the robot is on the field again, the operator of the GameController will send the *playing* signal to it. If the wireless is not working, the assistant referee who placed the robot back on the field has to bring it into the *playing* state again by pressing the back switch three times. Please note that this will also work if the robot actually is in the *penalized* state (previously signaled through the wireless) and is only waiting for a single click on its back switch, because any further short touches on that switch will be ignored.

4.3. Manual Interaction by Team Members

Manual interaction with the robots, either directly or via some communications mechanism, is not permitted. Team members can only touch one of their robots when an assistant referee hands it over to them after a "Request for Pick-up".

4.4. Kick-off Shot

A "kick-off shot" can never score a goal. A "kick-off shot" is a shot taken after a kick-off before the entire ball having left the center circle, including the boundary line. The ball must touch a player from the kick-off team after leaving the center circle before a goal can be scored by the team taking the kick-off. If a kick-off shot enters the goal (either directly or via contact with an opposing robot), no goal will be scored and a kick-off will be awarded to the defending team (as per Section 3.5).

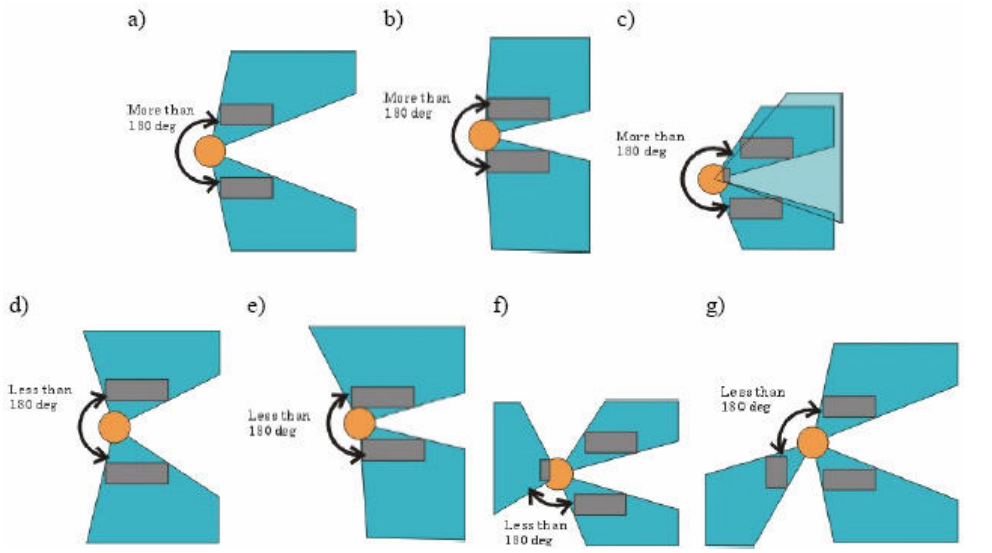

Figure 10: Examples for "Ball Holding". The orange circle is the ball, the gray boxes are parts of the robot, and the blue area visualizes the ranges occupied by the robot. Situations a), b) and c) are legal, whereas d), e), f) and g) violate the rule.

4.5. Ball Holding

The goalie is allowed to hold the ball for up to 5 seconds as long as it has 2 feet inside in its own penalty area. In all other cases, robots are allowed to hold the ball for up to 3 seconds. Holding the ball for longer than this is "Ball Holding" and is not allowed.

A robot which does not leave enough open space around the ball will be penalized as "Ball Holding" if that situation continues more than 3 seconds. The robot parts taken into account are only those parts that are below the horizontal plane that is at the same height as the top of the ball (cf. Figure 9). The occupation of the ball is judged using the projection of the related robot parts onto the ground. "Enough open space" means one successive open area that covers more than 180° around the ball. It is not important whether the robot actually touches the ball (cf. Figure

10a, b vs. Figure 10d, e). If the chin of the robot touches the ball, that part is also lower than the top the ball, and therefore must also be taken into account (cf. Figure 10c, f, g).

Intentional continual holding is prohibited even if each individual holding time does not continue for up to the time limit. In this case, the continual holding is regarded as a continuous hold from the very beginning and the holding rule is strictly applied. The violation of this rule will result in having the penalized robot removed from the field for 30 seconds as per the standard removal penalty (see Section 4.2 for details). In case of a violation by the goal keeper, the robot will be removed for 0 seconds as per the standard removal penalty, i. e. he will be placed on the halfway line immediately (no need to be kept outside of the field). The ball should be removed from the possession of the robot and placed where the foul occurred. If the robot that held the ball has moved the ball before the robot can be removed, the ball shall be replaced where the foul occurred.

Example. A robot holds the ball and before the referees can remove the robot, it shoots the ball into the goal. The goal will not be counted and the ball will be replaced where the robot held the ball.

4.6. Goalie Pushing

When the goalie is in its own penalty area (2 feet on or inside line), no attacker may be in contact with the goalie for more than 3 seconds **or** be pushing the goalie indirectly through the ball for more than 3 seconds. If two attacking robots are in simultaneously in contact with the goalie then the second attacking robot will be removed **immediately** and penalised for 30 seconds.

Successive short pushes shall be counted as a single push even if each individual contact does not continue for three seconds. A robot should obviously back off for at least as long as it pushed for the three second clock to reset. When violating this rule the attacker will be removed for 30 seconds as per the standard removal penalty (see Section 4.2).

If a goal is scored by an action performed after the 3 seconds of contact are over (or by a second robot that is touching the goalie), but before the robot can be removed, the ball and the robots involved in that action (the goalie, other robots, but not the penalized robot) shall be replaced where they were when the foul occurred.

Example 1. An attacker that has pushed the goalie shoots a goal after the violation but before it was removed. In that case, the goalie and the ball are replaced to the locations they were when the violation happened, and the goal is not counted.

Example 2. A second attacker scored a goal after the violation but before the pushing robot was removed. In that case, the goalie, the second attacker, and the ball are replaced to their original locations when the violation happened, and the goal is not counted. This is necessary, because the pushing robot prevented the goal keeper from defending its goal.

Example 3. An attacker that is currently pushing a goalie shoots a goal. If there has not yet been 3 seconds of contact between the attacker and the goalie, then the goal is allowed.

4.7. Field Player Pushing

Any robot pushing another robot for more than 3 seconds will be removed from play for 30 seconds as per the standard removal penalty (see Section 4.2). The closest robot (including the goal keeper) to the ball on each team, if it is within 2 dog-lengths of the ball, cannot be called for pushing. If a robot is standing still it cannot be called for pushing. If two moving robots are charging into each other then both robots are removed. The goalie cannot be called for pushing as long as it is within its penalty area at least with two legs.

Example 1. If two robots of different teams are fighting for the ball and a third robot touches one of them, it will be removed.

Example 2. If two robots of the same team are—for some reason—fighting for the ball, the robot that is further away from the ball will be removed.

Example 3. If two robots are pushing each other without a ball in the vicinity, they are both removed. If only one of them is moving, only the moving robot is removed.

4.8. Damage to the Field/Robots/Ball

A robot that knocks over one of the beacons will be removed for 30 seconds as per the standard removal penalty (see Section 4.2). A robot that damages the field and/or other robots will be removed from the field for the remainder of the game. Similarly a robot that poses a threat to the spectator's safety will also be removed. In such a case, a normal penalty kick will be awarded to the opposing team (cf. Section 3.7).

4.9. Leaving the Field

A robot that leaves the 4x6m carpeted area will be removed for 30 seconds as per the standard removal penalty (see Section 4.2). i. e. a robot may leave the 3.6x5.4m playing field but must stay within the 4x6m area.

4.10. Illegal Defender

Only the goalie can be within its team's penalty area. Having three legs inside the penalty area is the definition of being in the penalty area and that situation is not allowed for defending field players. When other defending robots enter the area, they will be removed for 30 seconds as per the standard removal penalty (see Section 4.2). This is called the "Illegal Defender Rule". This rule will be applied even if the goalie is outside of the penalty area, but not if an operational defender is pushed into the penalty area by an opponent.

If an illegal defender kicks an own goal, the goal is scored for the opponent.

4.11. Illegal Defense

The vertical projection of the goalie to the goal line should not occupy more than 50 percent of the length of the goal mouth. Those robots that commit this action (intentionally) will be replaced immediately on the halfway line as per the standard removal penalty for 0 seconds (see Section 4.2). Robots may violate this rule for brief periods of time to, for example, block a shot. These brief periods of time should be no longer than 3 seconds and must not be done continually as per the holding rule (see Section 4.5).

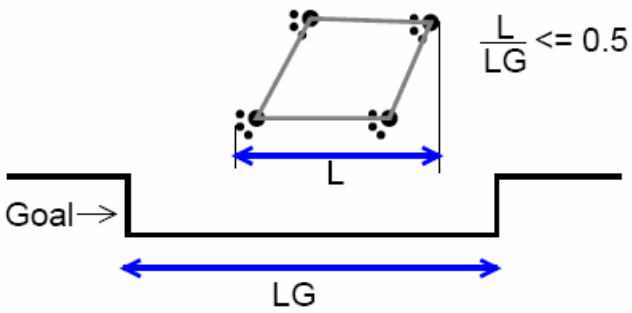

Figure 11: Illegal Defense

4.12. Obstruction

When a robot that is not heading for the ball is actively, and intentionally, blocking the way to the ball for a robot of the other team, this is an obstruction (cf. Figure 12). The obstructing robot will be removed for 30 seconds as per the standard removal penalty (see Section 4.2). Note: The obstruction rule this year is significantly different from the obstruction rule in previous years. In general, collisions should now be handled using the 'pushing' rules (see Sections 4.6 and 4.7).

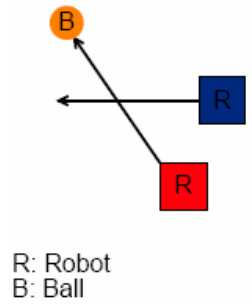

Figure 12: Obstruction

Example. If a robot of one team, robot A, is heading for the ball, but a robot of the other team, robot B, is in its way without heading for the ball, this, by itself, is not an obstruction. But, if robot A starts to move around robot B, and then robot B intentionally moves to block robot A again, this is an obstruction, even if the robots do not actually touch.

Note: The intent of the obstruction rule is to stop people implementing code that deliberately attempts to obstruct opponent robots. We hope that this penalty never needs to be called.

4.13. Jamming

During the match any robot shall never jam the communication and the sensor systems of the opponents:

Wireless communication. Teams can use a bandwidth of up to 2 Mbps of the wireless. This includes any data transferred, namely the actual payload and any protocol overhead created, e. g., by TCP, UDP, the GameController. If a team uses more bandwidth over a couple of seconds in a game, it will be disqualified for that game. Except from the wireless cards and the access points provided by the organizers of the competition, nobody close to the field is allowed using 2.4 GHz radio equipments including cellular phones and Bluetooth devices.

Acoustic communication. If acoustic communication is used by both teams, they shall negotiate before the match how they can reduce interference. If only one team uses acoustic communication, the robots of the other team shall avoid producing any sound. In addition, both the teams and the audience shall avoid intentionally confusing the robots by producing similar sounds to those used for communication.

Visual perception. To avoid confusing other robots, the robots are only allowed to use the white, green, and red LEDs. In particular, orange LEDs are not allowed. In general, the use of flashlights is not allowed during the games.

5. Judgement

The referees are the only persons that are allowed inside the playing area.

5.1. Head Referee

The head referee signals game starts, restarts, when a goal was scored, the case of *game stuck*, and penalties by a single whistle. In general, the head referee first whistles and then announces the reason for the whistle. The only exception is the case of the kick-off, in which the reason for the whistle is obvious. The whistle defines the point in time at which the decision is made. If the head referee has to announce many decisions in short sequence, he may skip whistling. For penalties, he announces the infraction committed, the team color, and the jersey number of the robot, e. g. “illegal defender, blue number 4”. In case of a goal scored, local or global game stuck, this is also announced verbally. By two whistles, the head referee terminates the first half; by three whistles he terminates the second half, i. e. the whole game. The head referee is also keeping the time of each half, i. e., he or she stops the clock after a goal or game stuck, and continues it at the kick-off[38].

In the penalty kick shoot-out, the head referee keeps the time.

Any decision of the head referee is valid. There is no discussion about decisions during the game, neither between the assistant referees and the head referee, nor between the audience or the teams

[38] The clock may not be stopped during the preliminaries.

and the head referee. Neither the teams, nor the audience can have any influence on the positions the head referee selects for the penalty kick shoot-out.

5.2. Assistant Referees

The two assistant referees handle the robots. They start them if the wireless is not working, they move them manually to legal kick-off positions, they take them out when the robots are penalized, and they put them in again. If a team requests picking up a robot, an assistant referee will pick it up and give it to one of the team members. An assistant referee will also put the robot back on the field. In addition, the assistant referees can indicate violations against the rules committed by robots to the head referee, so that the head referee can decide whether to penalize a certain robot or not.

5.3. Operator of the GameController

The operator of the GameController sits at a PC outside the playing area. He or she will signal any change in the game state to the robots via the wireless as they are announced by the head referee.

He or she will also inform the assistant referees when a timed penalty is over and a robot has to be placed back on the field.

5.4. Selection of the Referees

At least in the preliminaries, the games will be refereed by members of teams from a different Round-Robin group. Each team has to referee a number of these games. For each of the games, it can either provide the head referee and the operator of the GameController, or the two assistant referees. These persons must have good knowledge of the rules as applied in the tournament, and the operator of the GameController must be experienced in using that software. The persons should be selected among the more senior members of a team, and preferably have prior experience with games in the RoboCup Four-Legged Robot league.

Referees for playoff games will need to be certified (i. e. deemed fit to referee) by at least half the teams in the playoffs. Unless they have no eligible referees, each team in the playoffs shall supply at least one referee for the playoffs. For a particular game, each of the teams playing shall be able to veto one and only one eligible referee with no reason required.

5.5. Referees During the Match

The head referee and the assistant referees should wear black or white clothing/shoes and avoid reserved colors for the ball, the goals, and player markings in their clothing. They may enter the field in particular situations, e. g., to reposition the robots when applying a or stuck. They should avoid interfering with the robots as much as possible.

6. Questions/Comments

Questions or comments on these rules should be mailed to legged tech@tzi.de

14. Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Hamburg, 9. September 2005

Valerij Krichevskyi

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, 9. September 2005

Valerij Krichevskyi

Lernsystem für die Verhaltenssteuerung in der RoboCup Four Legged League

Implementierung eines lernfähigen Systems für die Verwendung innerhalb der XABSL2-Beschreibung des German Team-Codes

Jonas Reese
(9reese@informatik.uni-hamburg.de)

Eine Projektarbeit im Rahmen des Projekts
"RoboCups - Robotersysteme in der SONY-LeggedLiga"
SOSE 2004/WS 2005 am Arbeitsbereich TIS
des Fachbereichs Informatik, Universität Hamburg.

Projektleitung: Dipl.-Inform. Birgit Koch, Prof. Dr.-Ing. D.P.F. Möller

19. März 2005

Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Übersicht über die Architektur des German Team Codes 1
- 1.2 `BehaviorControl` - die Verhaltenskontrolle 2
- 1.3 Die Extended Agent Behavior Specification Language (XABSL) . 2
- 1.4 Wo dieses Projekt ansetzt 3
- 1.5 Begriffsdefinitionen 3

2 Das Lernsystem **4**
- 2.1 Repräsentation von lernbezogenen Erfahrungswerten 4
  - 2.1.1 Die Klasse `Experience` 7
  - 2.1.2 Die Klasse `ExperienceBase` 7
- 2.2 Das Lernmodul 8
  - 2.2.1 Aufgaben 8
  - 2.2.2 Eine Standardlösung für das Lernmodul 9
    - 2.2.2.1 Sammeln von bewertungsrelevanten Informationen 9
    - 2.2.2.2 Bewertung von lernbezogenen Erfahrungen . . . 10
- 2.3 Lernsymbole 11
  - 2.3.1 Aufgaben 11
  - 2.3.2 Aufbau der logischen Bezeichner 11
- 2.4 Der Experience Editor für RobotControl 12
  - 2.4.1 Funktion 12
  - 2.4.2 Erstellen einer Konfiguration 12
  - 2.4.3 Editieren der Erfahrungsdatenbasis 13

3 Vorgehensweise zum Hinzufügen eines neuen Erfahrungstyps **15**
- 3.1 Registrieren des Erfahrungstyps 15
- 3.2 Festlegen des Wertebereiches und des Standardwertes 16
- 3.3 Registrieren des XABSL-Eingabesymbols 16
- 3.4 Anpassung der Verhaltensbeschreibung 16

4 Durchführung des Lernprozesses **18**

5 Fazit und Bewertung **19**

Zusammenfassung

Fußballspielende Roboter bieten ein breites Spektrum von informatischen Disziplinen wie Bilderkennung, Künstliche Intelligenz *(vgl. [2, 3])*, Robotik und Kinematik oder auch Softwaretechnik. In dem German Team Code, der aus einer Allianz von vier deutschen Universitäten hervorgegangen ist und der seit 2001 bei RoboCup-Turnieren eingesetzt wird, sind bereits zu all diesen Bereichen der Informatik sehr gute Lösungen enthalten. Diese werden ständig weiterentwickelt und verbessert, oder es kommen neue Lösungsansätze hinzu.

Vor diesem Hintergrund einer mehr als vierjährigen Entwicklungsgeschichte fällt es nicht leicht, im Code des German Team Möglichkeiten zur Optimierung zu finden, die sich im Rahmen einer Projektarbeit realisieren lassen. Allerdings hat der Erfolg der Hamburg DogBots bei den German Open 2004 gezeigt, dass auch kleinere Optimierungen zu großen Vorteilen verhelfen können. Dort war es im Wesentlichen die dynamische Anpassung der Taktik an den Spielverlauf.

Grundidee dieser Projektarbeit ist es, eine Anpassung des allgemeinen Spielverhaltens über die Dauer eines Spiels hinaus zu erreichen. Hierbei sollen Trainingsspiele dazu dienen, die besten Verhaltensparameter herauszufinden, um sie dann im Wettkampf anzuwenden.

Im Folgenden werden zunächst die hierfür relevanten Teile des German Team Codes eingeführt. Anschließend wird vorgestellt, wie diese Grundidee praktisch umgesetzt wurde.

Kapitel 1

Einführung

1.1 Übersicht über die Architektur des German Team Codes

Ein wichtiges Merkmal des German Team Codes ist die Abstraktion von der Zielplattform. Enthalten sind in der aktuellen Version Implementationen für Aperios - also das Aibo-Betriebssystem - und die Simulatorsoftware für Windows-Betriebssysteme. Erreicht wird diese Plattformunabhängigkeit durch *Prozesse*, die das Bindeglied zwischen System und Anwendung bilden *(vgl. [3])*. Diese Prozesse werden von dem plattformabhängigen Prozessrahmenwerk zyklisch aufgerufen und führen die ihnen zugeordneten Aufgaben plattformunabhängig aus, da sie nur auf im Code enthaltene Datenstrukturen zugreifen, die dann vom Prozessrahmenwerk interpretiert werden. Im Wesentlichen werden zwei Aufgabengebiete unterteilt: `Motion` und `Cognition`. Der `Motion`-Prozess ist für den Bewegungsapparat zuständig, während `Cognition` alle übrigen Aufgaben wie Bildverarbeitung, Selbstlokalisation, Sensordatenerfassung und -verarbeitung und Verhaltenskontrolle übernimmt. Es existieren weitere Prozesse für Logging- und Debuggingfunktionalitäten.

Die einzelnen Aufgaben eines Prozesses werden in *Modulen* gekapselt. Hierdurch wird erreicht, dass der Quellcode klar nach Aufgaben strukturiert und modularisiert werden kann. Außerdem erleichtert diese Herangehensweise den Austausch von verschiedenen Lösungen für einzelne Aufgaben.

Prozesse kommunizieren über Nachrichtenaustausch, während die Kommunikation zwischen Modulen eines Prozesses über einen gemeinsamen Datenspeicher im Prozess oder über Nachrichtenaustausch stattfinden kann. Zwar können Prozesse nebenläufig ausgeführt werden, Module hingegen werden innerhalb eines Prozesses nacheinander ausgeführt, so dass keine Konflikte durch nebenläufigen Zugriff auf den gemeinsamen Datenspeicher auftreten.

1.2 `BehaviorControl` - die Verhaltenskontrolle

Im `Cognition`-Prozess werden zunächst alle Module ausgeführt, die Informationen über den aktuellen Zustand der Umwelt (*World State*) beschaffen oder verarbeiten. Zuletzt wird dann das Modul zur Verhaltenskontrolle (`BehaviorControl`) ausgeführt, an das die zuvor ermittelten Informationen übergeben werden. Im `BehaviorControl`-Modul wird anhand dieser Informationen nun entschieden, wie der Roboter sich verhält. Das German Team verwendet zur Realisierung dieses Entscheidungsprozesses eine spezielle Beschreibungssprache, die sich hierfür besser eignet als die ansonsten verwendete Programmiersprache C++. Sie nennt sich *Extended Agent Behavior Specification Language* (XABSL).

1.3 Die Extended Agent Behavior Specification Language (XABSL)

XABSL ist eine Sprache zur Modellierung von Verhalten in Multiagentensystemen. Sie basiert auf der *Extensible Markup Language* (XML) und ist sowohl für die Beschreibung kurzfristiger, reaktiver als auch längerfristiger, zielorientierter Verhaltensweisen geeignet.

XABSL-Beschreibungen bestehen auf oberster Ebene aus *Agents*. Diese realisieren unterschiedliche, nicht zueinander in Bezug stehende Aufgaben. Im German Team Code gibt es beispielsweise verschiedene Agents für das Fußballspielen und den Strafstoß, da diese Aufgaben komplett unterschiedliche Verhaltensweisen erfordern. Der verwendete Agent gibt die Wurzel eines azyklischen, gerichteten Graphen von *Options* und *Basic Behaviors* an, wobei ein Basic Behavior immer nur ein Blatt sein darf. Jede Option definiert einen endlichen Automaten, dessen Kanten über Entscheidungsbäume - also eine Menge von `if`-Statements mit den jeweils zugehörigen Folgezuständen (Transitionen) - abgebildet werden. Jedem Zustand innerhalb einer Option ist zusätzlich ein Nachfolgeknoten im Graphen zugeordnet, also entweder wieder eine Option oder ein Basic Behavior. Basic Behaviors sind in C++ programmiert und realisieren die grundlegenden Verhaltensweisen des Agenten.

In XABSL werden also, ähnlich wie bei State Charts, endliche Automaten in hierarchischer Anordnung abgebildet. Dadurch bleibt die Anzahl der zu modellierenden Zustände im Gegensatz zu einer flachen Struktur überschaubar.

Die Schnittstelle zwischen XABSL und der in C++ geschriebenen Applikation bilden so genannte *Ein-* und *Ausgabesymbole*. Diese haben in XABSL einen eindeutigen Bezeichner, der auf eine Funktion oder Variable in der Applikation abgebildet wird. Ausgabesymbole werden verwendet, um aus der Verhaltenssteuerung heraus Zustandsänderungen in der Applikation herbeizuführen. Eingabesymbole sind ein unverzichtbarer Bestandteil bei der Entscheidungsfindung. Nur mit Eingabesymbolen können Informationen von anderen Modulen innerhalb des `Cognition`-Prozesses überhaupt im `BehaviorControl` verarbeitet werden.

Eine ausführliche Beschreibung zu XABSL findet sich in [1].

1.4 Wo dieses Projekt ansetzt

Die Verhaltensbeschreibung des German Team lässt sich vereinfacht als hierarchischer Entscheidungsbaum bezeichnen, an dessen Blättern sich die Basic Behaviors befinden, also die programmatisch realisierten grundlegenden Verhaltensweisen und Fähigkeiten des Roboters *(siehe [1])*. Innerhalb dieses Entscheidungsbaumes werden immer wieder Grenzwerte verwendet, also Werte, die ausschlaggebend dafür sind, ob nun die eine oder die andere Entscheidung getroffen wird. Diese Grenzwerte entstehen in der Regel durch Abschätzungen seitens der Entwickler, oder es sind Erfahrungswerte, die durch Testen gewonnen wurden.

Warum aber nicht genau diese Aufgabe dem Roboter überlassen? Wenn man voraussetzt, dass zumindest in einigen dieser Grenzwerte Optimierungspotenzial steckt, so macht es Sinn, den Roboter selbst Erfahrungswerte ausprobieren und bewerten zu lassen, um dann, wenn es darauf ankommt (im Turnier), die besten Werte zu verwenden.

1.5 Begriffsdefinitionen

Wie bereits erwähnt, wird also ein Teil der konstanten Grenzwerte, die bei der Entscheidungsfindung relevant sind, in Variablen umgewandelt. Anschließend soll der Roboter selbst durch Ausprobieren und Bewerten den besten Grenzwert herausfinden. Jeder dieser zu testenden Grenzwerte wird im Folgenden *Erfahrungswert* genannt. *Erfahrungstyp* hingegen bezeichnet eine Klasse gleichartiger Grenzwerte. Ein Tupel, bestehend aus einem Erfahrungswert und einem Erfahrungstyp wird einfach *Erfahrung* genannt. Einer Erfahrung wird dann im Laufe der Zeit eine *Bewertung* zugeordnet, die die Qualität der Erfahrung widerspiegelt. Eine Menge von Erfahrungen ist eine *Erfahrungsdatenbasis* oder *Erfahrungsbasis*.

Kapitel 2

Das Lernsystem

In diesem Kapitel werden Konzept, Aufbau und Implementation des Lernsystems ausführlich behandelt. Den Kern dieses Projekts bilden drei Komponenten *(vgl. [5])*, deren funktionale Beziehungen zueinander und zu den beiden wichtigsten Prozessen `Cognition` und `Motion` (siehe Abschnitt 1.1) - in Abbildung 2.1 dargestellt sind:

- Die *Erfahrungsbasis* (in Abbildung 2.1 grün hinterlegt),
- das *Lernmodul* (in Abbildung 2.1 gelb hinterlegt) und
- die *Lernsymbole* (rot hinterlegt).

Das UML-Diagramm in Abbildung 2.2 zeigt eine Übersicht über die Klassen des Lernsystems, deren statische Assoziationen und Vererbungshierarchien, sowie die funktionale Zugehörigkeit (anhand der Farbgebung, vergleiche Abbildung 2.1). Anmerkung: Weder Attribute noch Operationen der Klassen sind vollständig; es wurden jeweils nur die wichtigsten Attribute und Operationen in das Diagramm übernommen.

Nicht in den Abbildungen enthalten ist der *Experience Editor*, der in die *RobotControl*-Software integriert wurde und dem Benutzer ermöglicht, über eine graphische Benutzerschnittstelle das Lernsystem zu administrieren. Von dem Experience Editor werden nur die „für den Benutzer sichtbaren“ Teile am Ende dieses Kapitels vorgestellt.

2.1 Repräsentation von lernbezogenen Erfahrungswerten

Erfahrungen müssen nicht nur zur Laufzeit in einer geeigneten Datenstruktur organisiert werden, sondern auch persistent gespeichert werden können, da

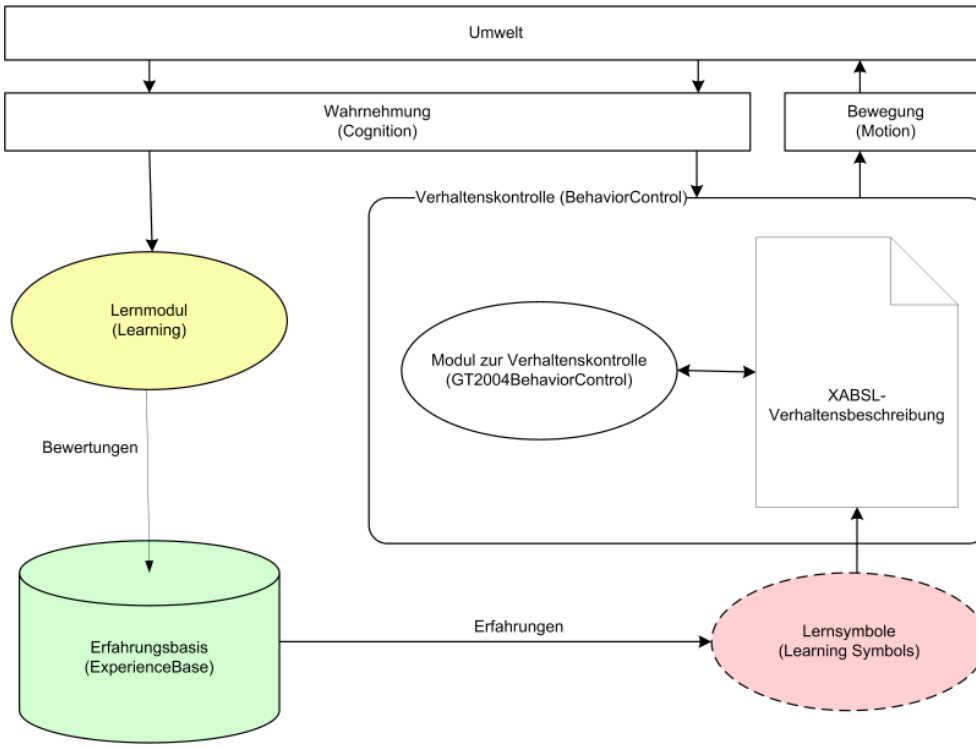

Abbildung 2.1: Funktionale Beziehungen zwischen den Komponenten des Lernsystems (farbig dargestellt), der Verhaltenskontrolle und den beiden Prozessen `Motion` und `Cognition`.

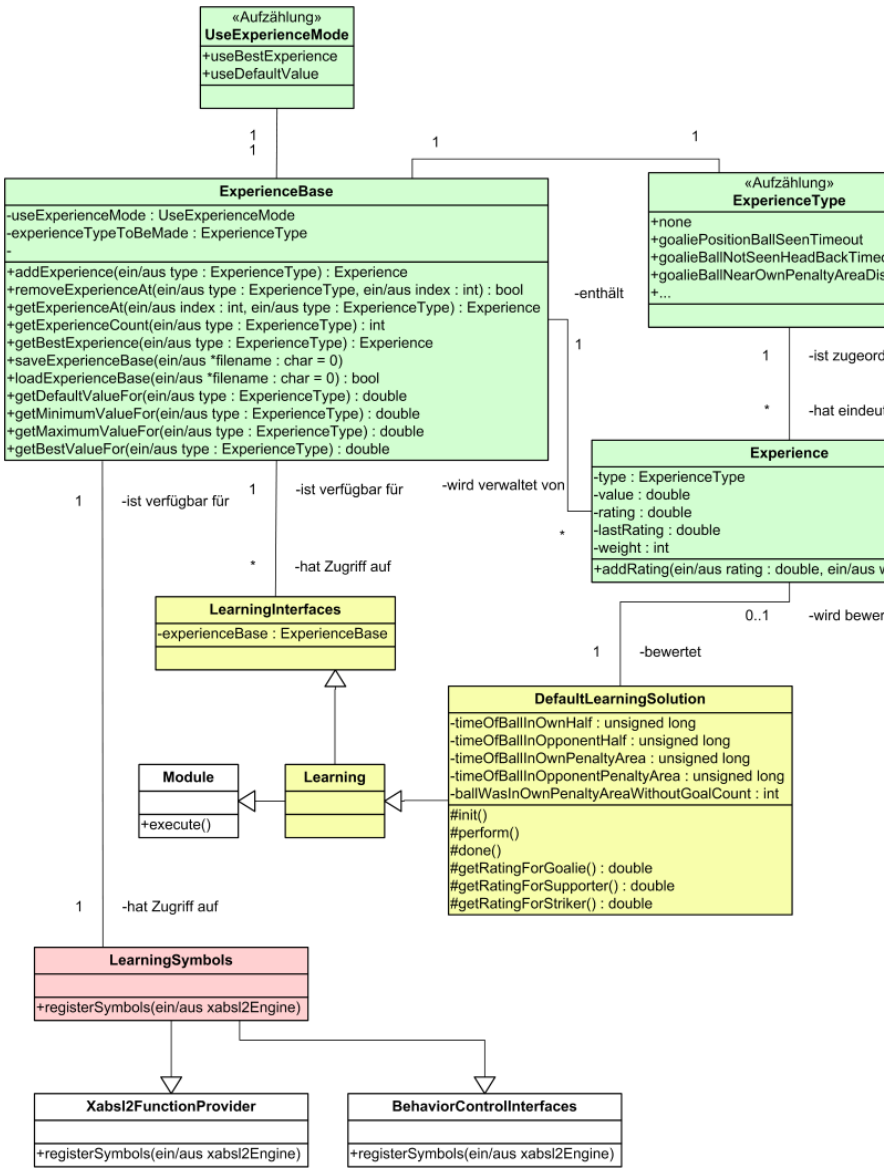

Abbildung 2.2: Vereinfachtes statisches Klassenmodell. Die Klassen sind nach ihren Funktionen farbig hinterlegt (siehe Abbildung 2.1).

sie über einzelne Spiele hinaus Bestand haben sollen. Diese Aufgaben übernehmen die Klassen `Experience` und `ExperienceBase`, die in der Headerdatei `Representations/Cognition/ExperienceBase.h` deklariert sind.

2.1.1 Die Klasse `Experience`

Diese Klasse kapselt einen einzelnen Erfahrungswert zusammen mit dessen Bewertungen und einem Erfahrungstyp. Sie ist relativ einfach aufgebaut, da sich alle Attribute als numerische Werte darstellen lassen.

Attribute der Klasse `Experience` sind der Erfahrungswert (`value`), die zuletzt erfolgte Bewertung (`lastRating`), die durchschnittliche Bewertung (`rating`) und eine Zähler, der bei Hinzufügen einer neuen Bewertung inkrementiert wird (`weight`). Solange dieser Zähler Null ist, gilt die Erfahrung als „noch nicht gemacht“.

Zusätzlich ist jedem Exemplar von `Experience` ein Erfahrungstyp zugeordnet (`ExperienceType`), der angibt, welche Art von Erfahrung hier repräsentiert wird. `ExperienceType` ist ein Aufzählungstyp, der ebenfalls in der Klasse `Experience` definiert wird. Dieser Typ bildet das Gegenstück zu dem in der XABSL-Beschreibung verwendeten Eingabesymbol (siehe Abschnitt 2.3).

Das Interface der Klasse bietet die typischen Funktionalitäten zum Abfragen und Setzen der Attribute, wobei zu erwähnen ist, dass der Erfahrungswert selbst nicht mehr geändert werden kann, wenn einmal eine Bewertung abgegeben wurde. Bewertungen für ein `Experience`-Exemplar können mit `addRating` abgegeben werden. Die Methode ermittelt automatisch den neuen Wert für die Gesamtbewertung.

Für das Lesen von einem Eingabedatenstrom und das Schreiben auf einen Ausgabedatenstrom sind für den Typ `Experience` zusätzlich die Operatoren `>>` bzw. `<<` überladen worden.

2.1.2 Die Klasse `ExperienceBase`

Diese Klasse ist das Kernstück der Erfahrungsbasis (in Abbildung 2.1 grün dargestellt). Sie übernimmt die folgenden zentralen Aufgaben:

1. Zuweisung von Minimal-, Maximal- und Standardwerten zu den Erfahrungstypen

   Jeder Erfahrungstyp hat zunächst einen Standardwert. Dieser wird als Konstante aus der vorhandenen Verhaltensbeschreibung übernommen und in `ExperienceBase` dem entsprechenden Erfahrungstyp zugewiesen. Minimal- und Maximalwert bilden eine untere und obere Schranke für den Wertebereich von Erfahrungswerten, entweder aufgrund technischer Begrenzungen (z.B. Spielfeldabmessungen) oder nach menschlichem Ermessen.

2. Konfiguration des Lernsystems

   Über das Feld `experienceTypeToBeMade` wird festgelegt, welche Art von Erfahrung gemacht wird, also was trainiert werden soll. Dies zieht die

Einschränkung mit sich, dass nicht mehrere Erfahrungen gleichzeitig gemacht werden können. Hierdurch wird ausgeschlossen, dass sich die durch eine Erfahrung bewirkte Verhaltensänderung nicht auf die Bewertung einer anderen Erfahrung auswirken kann. Trotzdem wäre es in der Zukunft eventuell wünschenswert, voneinander unabhängige Erfahrungen gleichzeitig machen zu können. Diese müssten dann allerdings auch unabhängig voneinander bewertet werden.

Das Feld `useExperienceMode` gibt an, welche Erfahrungswerte im Spiel angewendet werden sollen. Hier gibt es zwei Möglichkeiten: Die besten vorhandenen Erfahrungswerte (`useBestExperience`) oder die Standardwerte (`useDefaultValue`). Durch Letzteres kann also die Anwendung von Gelerntem deaktiviert werden.

3. Speicherung/Wiederherstellung und Verwaltung von Erfahrungen

   `ExperienceBase` bietet Funktionalität zum Hinzufügen, Entfernen und zum Zurückliefern von Erfahrungen, sowie einige Methoden zum bequemen Herausfiltern von Erfahrungen oder Erfahrungswerten für gegebene Erfahrungstypen.

   Darüber hinaus gibt es Funktionen, die das Laden aller lernrelevanten Informationen aus einer Datei oder das Schreiben dieser Informationen in eine Datei erledigen.

Zur Laufzeit existiert genau ein Exemplar dieser Klasse, das von dem Prozess `Cognition` referenziert und an das Lernmodul und die Learning Symbols übergeben wird (siehe 2.2, 2.3).

2.2 Das Lernmodul

2.2.1 Aufgaben

Das Lernmodul befindet sich in `Modules/Learning` unterhalb des Quellcodeverzeichnisses. Neben den Interfacedefinitionen befindet sich dort auch eine Standardlösung, genannt `DefaultLearningSolution`. Das `Learning`-Modul bekommt an seinen Konstruktor alle im `Cognition`-Prozess referenzierten Objekte übergeben, also alle Informationen, die auch dem `BehaviorControl` zur Verfügung stehen (siehe Abbildung 2.1). Die vom `LearningSelector` erzeugte Lösung des `Learning`-Moduls wird im `Cognition`-Prozess direkt vor der `BehaviorControl`-Lösung ausgeführt.

Dem Lernmodul fällt die schwierigste Aufgabe innerhalb des Lernsystems zu: Es muss den Erfolg (oder Misserfolg) von Erfahrungen bewerten. Die Bewertung sollte möglichst unabhängig sein von:

- Erfolg/Misserfolg der anderen Mitspieler
- Gegnerischem Können

- Eigenen bisher gemachten Erfahrungen

Diese Unabhängigkeiten sind schwer zu realisieren, daher sollten in der Praxis bei Trainingsspielen bestimmte Faktoren immer konstant bleiben (z.B. der gegnerische Code).

Das `Learning`-Modul wird während des Spiels zyklisch aufgerufen und kann somit zusätzlich zu den von anderen Modulen bereitgestellten Informationen eigene Daten über den Spielverlauf aufzeichnen. Diese können dann später in die Berechnung der Bewertung mit einfließen.

Eine Lösung für das Lernmodul muss außerdem dafür Sorge tragen, dass die Erfahrungs-Datenbasis (`ExperienceBase`) zu Beginn eines Spiels geladen und nach einer erfolgten Bewertung wieder gespeichert wird.

2.2.2 Eine Standardlösung für das Lernmodul

Die Klasse `DefaultLearningSolution` ist eine Implementation für das `Learning`-Modul. Sie bietet nicht nur eine Standardlösung für die Erfolgsbewertung, sondern ist auch als Superklasse für weiterentwickelte oder verbesserte Erfolgsbewertungsalgorithmen gedacht. Diese brauchen dann nur drei Methoden zu überschreiben:

- `getRatingForGoalie()`
- `getRatingForSupporter()`
- `getRatingForStriker()`

Diese Methoden sollen dann entsprechend der Rolle des Spielers und entsprechend ihres Bewertungsalgorithmus einen `double`-Wert zurückgeben. Sie werden von `DefaultLearningSolution` automatisch aufgerufen, wenn der Spielzustand `ROBOCUP_STATE_FINISHED` erreicht wird. Daher ist es bei Trainingsspielen erforderlich, über den Game Manager zum Ende des Spiels die Finish-Nachricht zu versenden (oder den Roboter manuell in diesen Zustand zu versetzen). Das Abspeichern der Erfahrungsdatenbasis in eine Datei, wie übrigens auch das Laden aus einer Datei, wird von `DefaultLearningSolution` erledigt. Der Dateiname ist `playerX.exp`, wobei `X` die Spielernummer ist (1 für Goalie, 4 für Striker). Die Datei muss im Arbeitsverzeichnis der German Team Applikation liegen (typischerweise das `Config`-Verzeichnis, welches auf dem MemoryStick unter `open-r/app` liegt).

2.2.2.1 Sammeln von bewertungsrelevanten Informationen

Zusätzlich zu den bereits erwähnten Funktionalitäten sammelt die Standardlösung `DefaultLearningSolution` allerlei nützliche Informationen über den Spielverlauf, die später in die Bewertung einfließen können. Hierfür wird die Methode `perform()` ausgeführt. Um den Rechenaufwand in Grenzen zu halten, wird diese Routine höchstens zweimal in der Sekunde aufgerufen. Liegt der letzte Aufruf weniger als 500 Millisekunden zurück, wird die Kontrolle vom Modul

sofort zurückgegeben. `DefaultLearningSolution` ermittelt standardmäßig die folgenden Informationen:

- Wie lange wird schon gespielt?
- Wie viel Zeit hat der Ball in der eigenen/gegnerischen Hälfte verbracht?
- Wie viel Zeit hat der Ball im eigenen/gegnerischen Strafraum verbracht?
- Wie häufig war der Ball im eigenen Strafraum, ohne dass ein Tor gefallen ist?

Auch diese Funktionalität können Subklassen erweitern, indem sie `perform()` überschreiben, darin zunächst `DefaultLearningSolution::perform()` aufrufen und anschließend Code für eigene zusätzliche Informationsgewinnung hinzufügen.

2.2.2.2 Bewertung von lernbezogenen Erfahrungen

Die Bewertung einer Erfahrung wird durch eine simple Fließkommazahl dargestellt. Hierbei ist es unerheblich, in welchem Wertebereich diese Zahl liegt. Wichtig ist, dass sich die Bewertungen für Erfahrungen desselben Erfahrungstyps sinnvoll miteinander vergleichen lassen (siehe 2.1.1). Dies bedeutet, dass die Bewertung einer Erfahrung mit einem größeren Wert (`rating`) im Vergleich zu einer geringer ausfallenden Bewertung einer anderen Erfahrung desselben Typs die Aussage zulässt, dass die erste Erfahrung qualitativ besser ist als die zweite.

In der `DefaultLearningSolution` wird die Bewertung nur in Abhängigkeit der Rolle des Spielers getroffen, also für Goalie, Supporter oder Striker. Die nächste Abstufung nach Erfahrungstyp ist zurzeit nicht implementiert.

Hauptaufgabe des Goalies ist es, „den Kasten sauber zu halten“. Daher basiert hier die Bewertung auf der Anzahl der kassierten Tore. Diese Information allein reicht allerdings für eine sinnvolle Bewertung nicht aus. So ist es beispielsweise möglich, dass die Supporter sehr schlecht spielen und dadurch der Ball häufiger in der eigenen Spielfeldhälfte unterwegs ist. Oder es verhält sich umgekehrt, und der Ball rollt kaum in die eigene Hälfte. Daher wird in das Bewertungsresultat mit eingerechnet, wie viel Zeit der Ball in der eigenen oder gegnerischen Spielfeldhälfte verbracht hat. Es dürfen für eine gute Bewertung also mehr Tore kassiert werden, wenn die eigene Mannschaft insgesamt mehr in der Defensive war.

Zusätzlich gibt es einen Bonus für jedes Mal, wenn der Ball den eigenen Strafraum passiert hat, ohne dass ein Tor gefallen ist. In diesem Falle wird von einer erfolgreichen Torverteidigung ausgegangen.

Der Striker soll die Mannschaft zum Sieg führen, also Tore schießen. Daher ist die eigene Torzahl Grundlage für die Bewertung des Spielers mit der Nummer Vier. Diese wird auf eine Stunde Spielzeit hochgerechnet. Das Ergebnis wird dann mit einer Zahl zwischen 1 und 2 multipliziert, die umso größer ist, je mehr Zeit der Ball im gegnerischen Strafraum verbringt. Hierdurch soll offensives Spiel honoriert werden, auch wenn nicht immer tatsächlich ein Tor fällt.

Die Supporter mit den Spielernummern Zwei und Drei können die Rollen zwischen *Offensive Supporter* und *Defensive Supporter* tauschen. Daher wird zwischen ihnen nicht unterschieden. Wenn sie gute Arbeit leisten, haben auch Goalie und Striker die Chance auf ein erfolgreiches Spiel. Die Bewertung der Supporter ist daher die Summe der Bewertungen für Goalie und Striker. Zusätzlich wird positiv bewertet, wenn der Ball wenig Zeit im eigenen Strafraum verbringt, da die Supporter den Ball in die gegnerische Hälfte treiben sollen.

2.3 Lernsymbole

2.3.1 Aufgaben

Lernsymbole (`LearningSymbols`) sind das Bindeglied zwischen der Erfahrungs-Datenbasis (`ExperienceBase`) und der XABSL-Verhaltensbeschreibung (siehe Abbildung 2.1). Mit ihnen werden Erfahrungswerte dem `BehaviorControl` zur Verfügung gestellt.

Im Pfad `Modules/BehaviorControl/CommonXabsl2Symbols` unterhalb des Quellcode-Verzeichnisses befinden sich die Dateien
`learning-symbols.xml`,
`LearningSymbols.h` und
`LearningSymbols.cpp`.

Die XML-Datei deklariert die Lernsymbole für die Benutzung in der XABSL-Beschreibung und deren Verifikation während des Compiliervorgangs. Die anderen beiden Dateien enthalten den C++-Code, um die Symbolbezeichner zur Laufzeit an Funktionsaufrufe zu binden, sowie die Funktionsaufrufe selbst. Das Binden von Symbolen an Funktionsaufrufe geschieht durch einen Aufruf aus dem `GT2004BehaviorControl`-Modul heraus.

2.3.2 Aufbau der logischen Bezeichner

Lernsymbole beginnen immer mit dem Präfix `learning`, gefolgt von einem Punkt und einem innerhalb des Lernsystems eindeutigen Bezeichner. Dieser Bezeichner entspricht dem Bezeichner des jeweiligen Aufzählungselements vom Typ `ExperienceType`, der in der Klasse `Experience` definiert ist (siehe 2.1.1). Allerdings wird in der XABSL-Beschreibung per Konvention keine Camel-Case-Notation[1] benutzt, stattdessen werden einzelne Wortteile mit Bindestrich getrennt. Der erste Wortteil ist hierbei immer die Spielerrolle, für die dieses Symbol gültig ist.

Ein Beispiel:

Um aus der XABSL-Beschreibung heraus für den Goalie auf den aktuellen Wert für den Erfahrungstyp `goalieBallNotSeenHeadBackTimeout` zuzugreifen, muss das Eingabesymbol mit dem Bezeichner
`learning.goalie-ball-not-seen-head-back-timeout` verwendet werden.

[1] Notation, bei der die Wortbestandteile eines Bezeichners durch großgeschriebene Anfangsbuchstaben getrennt werden.

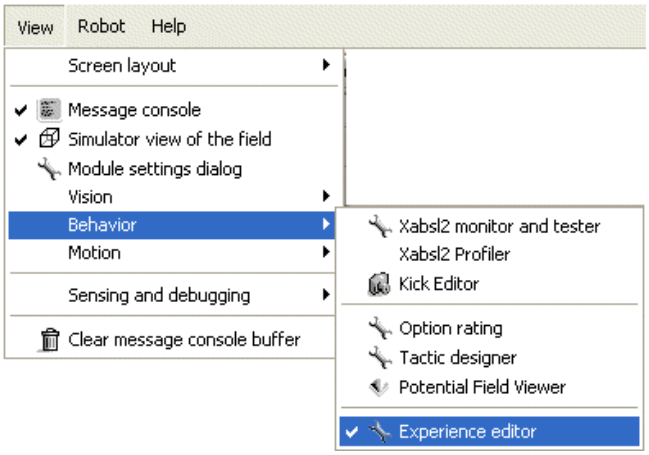

Abbildung 2.3: Öffnen des Experience Editors in RobotControl.

2.4 Der Experience Editor für RobotControl

2.4.1 Funktion

Wie bereits in Abschnitt 2.4.1 erläutert, erfolgt die Konfiguration des Lernsystems durch die Klasse `ExperienceBase`. Von dieser wird beim Starten der German Team Applikation ein Exemplar erzeugt, das dann mit Werten initialisiert wird. Diese Werte werden aus einer Binärdatei eingelesen, sofern diese Datei existiert. Existiert sie nicht, kann die Erfahrungsbasis nur die Standardwerte zur Verfügung stellen (vergleiche Abschnitt). Soll also das Lernsystem aktiviert werden, so muss für jeden Spieler eine solche Datei mit einer entsprechenden Konfiguration erstellt werden. Hierfür gibt es den *Experience Editor*, der in die RobotControl Applikation integriert ist. Mit ihm kann man Konfigurationen erstellen, vorhandene Konfigurationen ändern und die Erfahrungsdatenbasis einsehen und editieren. Geöffnet wird der Experience Editor über die Menüs `View` und `Behavior` (siehe Abbildung 2.3).

2.4.2 Erstellen einer Konfiguration

Da nach Inbetriebnahme des Lernsystems zunächst Trainingsspiele absolviert werden müssen (siehe Kapitel 4), sollte ein Erfahrungstyp in der Dropdown-Liste ausgewählt werden. Über den Button *Set Current* wird nun festgelegt, dass der hier ausgewählte Erfahrungstyp derjenige sein soll, der trainiert wird. Um zu vermeiden, dass eventuell bereits vorhandene Erfahrungen anderer Erfahrungstypen sich auf die Bewertung auswirken, sollte für Trainingsspiele *Use default value* ausgewählt sein.

Für ein Turnierspiel muss die Checkbox *Training Game* unselektiert und *Use best experience* ausgewählt sein. Dies macht natürlich nur Sinn, wenn zuvor in

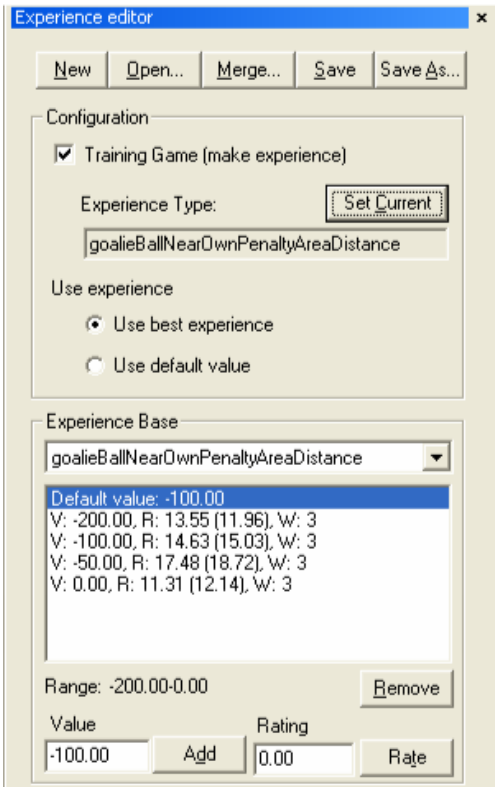

Abbildung 2.4: Die Oberfläche des Experience Editors.

Trainingsspielen bereits eine Erfahrungsdatenbasis aufgebaut wurde.

Die Konfiguration wird dann mit *Save* oder *Save As* auf dem Memory Stick des Roboters abgelegt, und zwar im Verzeichnis `/open-r/app/config`. Der Dateiname muss entsprechend der Spielernummer `player1.exp` bis `player4.exp` sein.

2.4.3 Editieren der Erfahrungsdatenbasis

Wenn eine Konfiguration erstellt wurde, kann zusätzlich die Erfahrungsdatenbasis angepasst werden. Dies geschieht im unteren Teil des Experience Editor Dialogs (siehe Abbildung 2.4). Hier kann man für den ausgewählten Erfahrungstyp Erfahrungswerte hinzufügen oder entfernen. Diese müssen natürlich im vorgegebenen Wertebereich für den Erfahrungstyp liegen. Werden keine Werte für einen Erfahrungstyp hinzugefügt, den es zu trainieren gilt, so werden beim Starten des Lernsystems auf dem Roboter (oder im Simulator) automatisch fünf Erfahrungen angelegt, deren Werte gleichmäßig über den Wertebereich verteilt sind. Zusätzlich wird, falls nicht bereits in diesen fünf Werten enthalten, eine

Erfahrung mit dem Standardwert erstellt. Für das Testen und Bewerten von Erfahrungswerten wird der Wert mit der kleinsten Anzahl von Bewertungen - also mit dem kleinsten Wert für `weight` - gewählt, und zwar in der Reihenfolge wie im Experience Editor dargestellt.

Zusätzlich gibt es die Möglichkeit, manuell Bewertungen abzugeben. Dies soll allerdings nur zu Testzwecken geschehen und ist mit einer Sicherheitsabfrage belegt.

Die Funktion *Merge* erlaubt es dem Benutzer, zu der aktuell geöffneten Erfahrungsdatenbasis eine andere Erfahrungsdatenbasis hinzuzufügen. So können unabhängig voneinander gewonnene Erfahrungen zu einer Datenbasis zusammengelegt werden. Hierbei werden ganz neue Erfahrungen neu angelegt, während bereits vorhandenen Erfahrungen die Bewertungen aus der mit der *Merge*-Funktion ausgewählten Erfahrungsdatenbasis hinzugefügt werden. Hierbei ist darauf zu achten, dass dieselben Erfahrungsbasen nicht mehrmals zusammengeführt werden, da sonst die Bewertungen identischer Erfahrungen das doppelte Gewicht bekommen.

Kapitel 3

Vorgehensweise zum Hinzufügen eines neuen Erfahrungstyps

Wenn in der Verhaltensbeschreibung konstante Werte durch veränderliche Erfahrungswerte ersetzt werden sollen, so müssen für gleichartige Klassen solcher Konstanten neue Erfahrungstypen hinzugefügt werden. Dies ist auch der Fall, wenn die Verhaltensbeschreibung unter Verwendung des Lernsystems erweitert werden soll. Im Folgenden wird chronologisch erläutert, an welchen Stellen des Codes hierfür welche Änderungen notwendig sind.

3.1 Registrieren des Erfahrungstyps

Zunächst muss der neue Erfahrungstyp beim Lernsystem registriert werden. Hierfür genügen einige Anpassungen in der Klasse Experience, Headerdatei `ExperienceBase.h`.

Die Definition für den Aufzählungstyp `ExperienceType` (siehe Klassendiagramm in Abbildung 2.2) muss um den neuen Typ erweitert werden. Hierbei ist es wichtig, dass der neue Typ in der Aufzählungsreihenfolge *nach* den bereits vorhandenen eingefügt wird, da es sonst beim Einlesen älterer Erfahrungsdatenbanken aus einer Datei zu Fehlern kommen kann. Das letzte Element der Aufzählung muss `numberOfExperienceTypes` sein.

Damit der Erfahrungstyp im Experience Editor dargestellt werden kann, sollte zusätzlich die statische Methode `getExperienceTypeName` erweitert werden, so dass für den neuen Typ der entsprechende lesbare Name zurückgegeben wird.

3.2 Festlegen des Wertebereiches und des Standardwertes

Minimal-, Maximal- und Standardwert für die Erfahrungstypen werden in der Klasse `ExperienceBase` verwaltet. Im Konstruktor dieser Klasse befinden sich die Zuweisungen hierfür. Die Arrays `defValue`, `minValue` und `maxValue` (*default value*, *minimum value* und *maximum value*) müssen hier für den neu hinzugefügten Erfahrungstyp mit Werten belegt werden, wobei gilt:

`minValue[n]` $\leq$ `defValue[n]` $\leq$ `maxValue[n]` und
`minValue[n]` $\neq$ `maxValue[n]` für einen Erfahrungstyp n

3.3 Registrieren des XABSL-Eingabesymbols

Wie in Abschnitt 2.3 beschrieben, muss der neue Erfahrungstyp nun der XABSL-Engine unter einem logischen Bezeichner bekannt gemacht werden. Wie dieser Bezeichner aufgebaut sein soll, ist ebenfalls in Abschnitt 2.3 erläutert. In der Methode
`registerSymbols(Xabsl2Engine& engine)`,
die sich in der Klasse `LearningSymbols` befindet, finden die Aufrufe zum Registrieren der Lernsymbole statt. Es gibt generell zwei Möglichkeiten, um ein Eingabesymbol einem Wert zuzuordnen:

- Über einen Zeiger

  Dieser enthält die Speicheradresse, in der sich der aktuelle Wert vom Typ `double` für das Eingabesymbol befindet.

- Über einen Funktionszeiger

  Dieser enthält die Einsprungadresse einer Funktion mit einem Rückgabewert vom Typ `double`, die den aktuellen Wert für das Eingabesymbol zurückliefert.

Für die bisher registrierten Eingabesymbole wurde die zweite Alternative gewählt, indem zu der Klasse `LeaningSymbols` Funktionen hinzugefügt wurden, die jeweils den Erfahrungswert aus dem `ExperienceBase`-Objekt liefern. Dies hat den Vorteil, dass die Auswertung eines Eingabesymbols über Debugging-Code zurückverfolgt werden kann. Hinzu kommt die Möglichkeit, zusätzliche Informationen zu gewinnen, etwa über die Häufigkeit der Auswertung bestimmter Symbole in bestimmten Spielsituationen, oder über die gesamte Länge eines Spiels. Daher wird empfohlen, bei Hinzufügen eines Erfahrungstyps dessen Eingabesymbol mit einer Funktion zu verknüpfen.

3.4 Anpassung der Verhaltensbeschreibung

Für die im Rahmen dieser Projektarbeit erstellten Erfahrungstypen wurde die XABSL-Verhaltensbeschreibung bereits für die Benutzung der entsprechenden

Eingabesymbole angepasst. Die Vorgehensweise ist denkbar einfach und am besten anhand eines Beispiels nachzuvollziehen.

Angenommen, wir finden eine solche Bedingung im Entscheidungsbaum vor:

```
<condition description="ball not known for long">
   <greater-than>
      <decimal-input-symbol-ref
         ref="ball.time-since-last-known"/>
      <decimal-value value="12000"/>
   </greater-than>
</condition>
<transition-to-state ref="ball-not-known-for-long"/>
```

Die angegebene Konstante mit dem Wert 12000, die hier für ein Timeout von zwölf Sekunden steht, soll durch den aktuellen Wert für den Erfahrungstyp mit dem Namen `supporterBallNotKnownForLongTimeout` ersetzt werden.

Wir erhalten folgendes Codefragment:

```
<condition description="ball not known for long">
   <greater-than>
      <decimal-input-symbol-ref
         ref="ball.time-since-last-known"/>
      <decimal-input-symbol-ref
         ref="learning.supporter-ball-not-known-for-long-timeout"/>
   </greater-than>
</condition>
<transition-to-state ref="ball-not-known-for-long"/>
```

Kapitel 4

Durchführung des Lernprozesses

Nachdem nun die Komponenten des Lernsystems und deren Funktionsweisen und Implementierungen vorgestellt und die Vorgehensweise für die Entwicklung und Einbindung neuer Erfahrungstypen erklärt wurden, stellt sich nun die Frage:

Wie wird das Lernsystem praktisch eingesetzt, also wie lernt der Roboter?

Die Antwort: Der Roboter lernt durch Training. Nachdem für die Spieler einer Mannschaft Konfigurationen erstellt wurden (siehe 2.4.2), müssen Trainingsspiele absolviert werden. Hierbei ist es wichtig, diese unter möglichst gleich bleibenden, wettkampfähnlichen Bedingungen durchzuführen.

Die gegnerische Mannschaft soll hierbei nicht lernen, sondern nur die Standardwerte für alle Erfahrungstypen benutzen. Für die Optimierung des Spielverhaltens im Hinblick auf einen speziellen Gegner können auch Trainingsspiele gegen Roboter ausgetragen werden, die den Code genau dieses Gegners benutzen, sofern dieser verfügbar ist. In diesem Falle müssen allerdings speziell für diesen Gegner neue Erfahrungsdatenbasen angelegt werden.

Nach Beendigung eines Trainingsspiels müssen alle Roboter in den Zustand `finished` versetzt werden, damit die Bewertung des Spiels erfolgt. Anschließend kann sofort ein neues Trainingsspiel beginnen. Der Roboter wird nun für den zu trainierenden Erfahrungstyp den nächsten Erfahrungswert auswählen. Es sollten für jeden Erfahrungstyp möglichst viele Trainingsspiele durchgeführt werden, damit eine repräsentative Gesamtbewertung entstehen kann.

Kapitel 5

Fazit und Bewertung

Das Anlegen einer umfangreichen Erfahrungsdatenbasis mit Hilfe von Trainingsspielen ist ein sehr aufwändiger Vorgang, da alle Erfahrungswerte für alle Erfahrungstypen möglichst oft bewertet werden müssen. Nur so kann das Gesamtverhalten des Roboters im Spiel mit Hilfe des Lernsystems sinnvoll optimiert werden. In der Praxis hat sich herausgestellt, dass das Austragen von Trainingsspielen in „physikalischer Form“, also mit Robotern auf dem Feld sehr viel Zeit kostet. Besondere Schwierigkeiten bereiteten hierbei variable Lichtverhältnisse.

Leider war es aus diesem Grund bisher noch nicht möglich, einen tatsächlichen Zuwachs der Siegerquote festzustellen. Die Evaluation dessen, ob und wie weit durch den Einsatz des Lernsystems ein taktischer Vorteil entsteht, wird daher noch mehr Zeit benötigen.

Tests mit dem Goalie im Simulator waren hingegen erfolgreicher. Bewertungen fielen nach erkennbar schlechter Leistung des Torwarts geringer aus als nach Spielen mit erfolgreicher Verteidigung des eigenen Tores. Setzt man nun voraus, dass die Leistung eines Spielers in einer ansonsten gleichbleibenden Testumgebung ausschließlich von den sich ändernden Erfahrungswerten abhängt, so lässt dies die Schlussfolgerung zu, dass das Prinzip der Bewertung von Erfahrungen auch in der Praxis eine Optimierung des Spielverhaltens nach sich zieht.

Diese Projektarbeit ist als die Umsetzung des Konzepts eines Lernsystems, das auf Erfahrungswerten und deren Bewertung basiert, einzustufen. Hierbei wurde besonderer Wert auf Erweiterbarkeit gelegt, das Projekt kann und soll also auch als Infrastruktur für künftige erweiterte und verbesserte Implementierungen dienen.

Um einen den praktischen Aufwand beim Einsatz des Lernsystems in Grenzen zu halten, scheint es allerdings unvermeidbar, den RobotControl-Simulator so zu erweitern, dass Trainingsspiele auch ausschließlich als Simulationen stattfinden können. Hierbei könnte man auf verschiedenen Rechnern parallel Trainingsspiele simulieren, so dass sich relativ schnell gute Lernergebnisse realisieren ließen.

Literaturverzeichnis

[1] Bach, J., Burkhard, H.-D., Jüngel, M., Lötzsch, M. Designing Agent Behavior with the Extensible Agent Behavior Specification Language XABSL. Lecture Notes in Artificial Intelligence, Padova, Italy. Springer 2004

[2] Dahm, I., Düffert, U., Hoffmann, J., Jüngel, M., Kallnik, M., Lötzsch, M., Risler, M., Röfer, T., Stelzer, M., Ziegler, J. GermanTeam2003. Lecture Notes in Artificial Intelligence, Padova, Italy. Springer 2004

[3] Röfer, T. An Architecture for a National RoboCup Team. Lecture Notes in Artificial Intelligence. Springer 417-425 2003

[4] Shoham, Y. Agent-oriented programming. Robotics Laboratory, Stanford University 1992

[5] V. Butz, M. Anticipatory behavior in adaptive learning systems. Springer 2003

Baccalaureatsarbeit

Autonomes Stehen mittels Rückkopplung über Beschleunigungssensoren bei Sony AIBO Robotern

Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Technische Informatiksysteme

17. Januar 2005

Malte-Nils Sörensen
Matrikel-Nr. 5099672
E-Mail: 8soerens@informatik.uni-hamburg.de
Chapeaurougeweg 4, 20535 Hamburg

Betreut durch
Dipl.-Inform. Birgit Koch,
Prof. Dr.-Ing. D.P.F. Möller
E-Mail: koch@informatik.uni-hamburg.de

Universität Hamburg, FB Informatik / TIS,
Vogt-Kölln-Straße 30, 22527 Hamburg

Inhaltsverzeichnis

1 Einleitung....1
1.1 Überblick / Abstract....1
1.2 Motivation....1
1.3 Notation....2
2 RoboCup....3
2.1 Übersicht....3
2.2 Zielsetzung....3
2.3 Geschichte....3
2.4 Die Meisterschaften....3
3 ERS-7....5
3.1 Beschleunigungssensoren....5
3.2 Beingelenke....5
3.3 Füße....7
3.4 weitere Systemressourcen....8
4 Theoretische Grundlagen....9
4.1 Gleichgewicht....9
4.2 Der menschliche Gleichgewichtssinn....10
4.3 Regelkreis....11
5 Implementation....13
5.1 Programmierung der AIBOs....13
5.2 Abfragen der Messwerte....14
5.3 Stellen der Gelenke....20
5.4 Auswerten der Messwerte....22
6 Abschließendes....24
7 Quellenverzeichnis....i

1 Einleitung

1.1 Überblick / Abstract

1.1.a Überblick

Der Rahmen dieser Arbeit ist die Erforschung der Möglichkeiten des autonomen zweibeinigen Stehens von Robotern des Typs *Sony AIBO ERS-7*, im Folgenden kurz *ERS-7* oder *AIBO* genannt.

Die Arbeit ist innerhalb des Projektes „*RoboCups-Robotersysteme in der SONY-Legged-Liga*“ erfolgt. Deshalb wird zu Beginn kurz erläutert, was der *RoboCup* ist (Abschnitt 2). Danach werden die für das Vorhaben relevanten technischen Eigenschaften des *ERS-7* beschrieben (Abschnitt 3) und die theoretischen Grundlagen zusammengefasst dargestellt (Abschnitt 4). Zum Abschluss wird dokumentiert, wie eine Implementation auf dem *ERS-7* mithilfe des R-CODE SDKs grundsätzlich auszusehen hat (Abschnitt 5).

1.1.b Abstract

The scope of this thesis is the exploration of the possibilities of autonomous two-legged standing of *Sony AIBO ERS-7* robots, below simply called *ERS-7* or *AIBO*.

This thesis is done within the project „*RoboCups-Robotersysteme in der SONY-Legged-Liga*“. For this reason section 2 will explain what *RoboCup* is. Afterwards the relevant technical properties for this intention will be described (section 3) and the theoretical basics displayed compendiously (section 4). The closure will document how an implementation on the *ERS-7* utilizing the R-CODE-SDK has to be structured (section 5).

1.2 Motivation

Wie sicherlich auch bei vielen anderen haben auch bei mir die Romane von Isaac Asimov das erste Interesse an der Thematik der Robotik geweckt. Später besaß ich dann auch das *Robotics Invention System* von *LEGO MINDSTORMS*™, das leider viel zu wenig Bauteile besaß.

Nach den positiven Erfahrungen, die ich damit machte, entschied ich mich später für das Projekt „RoboCups“ in dessen Rahmen ich nun diese Baccalaureatsarbeit schreibe.

Die Erfahrungen mit den *AIBOs* verliefen allerdings ganz anders als mit dem *Robotics Invention System*. Die Steuerungssoftware war bereits fertig und in vielen offiziellen *RoboCup*-Wettbewerben erprobt und verbessert worden. Sie war sehr komplex und der Einarbeitungsaufwand entsprechend hoch. Aber abgesehen von den für das Fußballspielen notwendigen Abläufen hatten die vorhergehenden Entwickler auch kleine Spielereien eingebaut, die für Demonstrationszwecke oder freie „Challenges“ (s. Abschnitt 2) verwendet wurden.

Nachdem ich die Möglichkeiten des *AIBOs* kennengelernt hatte, wurde mir klar, dass nicht alles genutzt wurde. Vor allem fiel mir auf, dass der Roboter in der Lage sein sollte, seine Lage und Bewegung zu kennen, was ihm ermöglichen sollte, unerwarteten externen Krafteinflüssen entgegenwirken zu können. Dies sollte ihn auch dazu befähigen, aufrecht zu stehen und ganz besonders stehen zu bleiben. Die Realisierung dieser Idee ist nun zum Thema dieser Arbeit geworden.

Einen direkten praktischen Nutzen für die *RoboCup*-Liga wird es in nächster Zeit wohl nicht haben, da schon jetzt zu Beginn klar ist, dass eine vierbeinige Plattform wesentlich stabiler ist, als eine zweibeinige. Man sieht dies auch sehr eindrucksvoll bei kleinen Kindern, die gerade Stehen lernen. Zudem kann man behaupten, dass Vierbeiner schneller sind, als Zweibeiner, wie man es deutlich beim Vergleich von Mensch und Gepard sehen kann. Erreicht der Gepard Geschwindigkeiten bis zu 100km/h, schafft es der Mensch gerade mal auf 36,8km/h. Der aktuelle

Weltrekord im 100-Meter-Sprint wird von Tim Montgomery mit 9,78 Sekunden gehalten, was ihn laut Guinness World Records zu dem schnellsten Menschen der Welt macht. (vgl. [GWR05])

Man kann natürlich argumentieren, dass die ebenfalls vierbeinige Schildkröte deutlich langsamer ist als ein Mensch. Es geht hier aber um das, was de facto erreichbar ist, und da hat die vierbeinige Fortbewegung offensichtlich einen deutlichen Vorsprung.

1.3 Notation

Fachbegriffe, die außerhalb der allgemeinen Informatik liegen, werden der Lesbarkeit halber immer *kursiv* gedruckt. Alle anderen Fachbegriffe werden normal gedruckt, da sie zu dem gebräuchlichen Wortschatz des Leser gehören sollten. Außerdem wird soweit möglich das deutsche Fachvokabular der Informatik verwendet.

Quelltexte und Teile davon werden, wenn sie im Fließtext enthalten sind, ebenfall *kursiv* gedruckt. Zusammenhängende Blöcke werden separat und wie folgt dargestellt:

```
10: PRINT „HELLO WORLD“
20: GOTO 10
```

Quelltext von Hello-World-Beispiel in BASIC.

Bilder und Illustrationen werden auf ähnliche Weise dekoriert:

Abbildung 1.3.1: Szene aus einem Spiel der German Open 2004

2 RoboCup

Alle folgenden Informationen über den RoboCup basieren auf [RCUP04].

2.1 Übersicht

RoboCup ist eine internationale Initiative, um die Entwicklung von künstlicher Intelligenz und Robotern zu fördern. Hierzu wurde das Fußballspielen als Standardproblem gewählt, um in diesem Rahmen den Einsatz von möglichst vielen Technologien erforschen zu können.

Dreh- und Angelpunkt von *RoboCup* sind die jährlichen Weltmeisterschaften, bei denen die teilnehmenden Mannschaften in verschiedenen Kategorien antreten können.

Eine lokale Variante des *RoboCups* ist die jährliche *German Open*. Hier treten haptsächlich verschiedene deutsche Teams gegeneinander an aber auch ein paar ausländische Teams sind hier zu finden. Eine Besonderheit stellen hier die Mitglieder des *GermanTeams* dar. Nach der *German Open* werden aus deren Programmcode die besten Komponenten zu einer gemeinsamen Programmierung zusammengefügt, mit der dann gemeinsam am *RoboCup* teilgenommen wird.

Dieser *GermanTeam-Code* steht allen Universitäten für Forschungszwecke zur freien Verfügung und diente auch unserem Fachbereich als Grundlage für die ersten Schritte im *RoboCup*-Bereich.

2.2 Zielsetzung

Der *RoboCup* wurde ins Leben gerufen um die gegenwärtige KI- und Robotik-Entwicklung international voran zu bringen, um im Jahre 2050 in der Lage zu sein, den dann amtierenden menschlichen Fußballweltmeister mit einer Mannschaft aus autonomen humanoiden Robotern schlagen zu können.

2.3 Geschichte

Die Idee zu *RoboCup* wurde 1993 entwickelt, es dauerte aber 4 Jahre, bis die erste internationale Meisterschaft in Japan stattfand. Das Vorhaben wird von verschiedensten Universitäten und Wirtschaftsunternehmen unterstützt.

Inzwischen finden die Meisterschaften im jährlichen Turnus statt. Jedes Jahr ist ein anderes Land der Gastgeber und die Teilnehmerzahl wächst auch stetig. Es haben sich auch lokale Meisterschaften entwickelt, wie z.B. die *German Open*, die 2001 zum ersten Mal veranstaltet wurde. An der *German Open* können auch ausländische Teams teilnehmen.

2004 nahm die Universität Hamburg erstmalig an der *SONY-Legged League* teil, die sich in der Kategorie *RoboCupSoccer* befindet. Sie trat mit dem Team „*Hamburg Dog Bots*“ des Fachbereichs Informatik zuerst bei den *German Open 2004* in Paderborn und danach bei dem *RoboCup 2004* in Lissabon an.

2.4 Die Meisterschaften

Der *RoboCup* ist unterteilt in einzelne Kategorien und diese wiederum in einzelne Ligen.

Als erstes wird die „4-Legged-Robot-League“ beschrieben, die auch als „SONY-Legged-League“ bekannt ist. Danach wird der Vollständigkeit halber auf die restlichen Ligen eingegangen.

2.4.a 4-Legged-League

Diese Liga erfordert Teams aus je 4 Robotern, jedoch sind hier nur die Modelle *Sony AIBO ERS-210, ERS-210A, ERS-7* und *ERS-7M2* erlaubt, daher wird diese Liga auch als „*SONY Legged League*“ oder „*SONY Four-Legged Robot League*“ bezeichnet.

Gespielt wird auf einem 4 Meter x 6 Meter großem Feld, das im Grundriss einem normalen Fußballfeld ähnelt, aber an dem zusätzlich verschiedene, eindeutige farbliche Markierungen angebracht sind, die zur Unterstützung der Selbstlokalisierung der Roboter dienen sollen. Die Tore sind ebenfalls farblich unterschiedlich markiert, genauso wie die Roboter der teilnehmenden Teams.

Die Gesamtbedingungen des Spielfeldes werden mit jeder Meisterschaft nach und nach an realere Bedingungen angepasst, was einen sukzessiven Wegfall dieser farblichen Hilfsmittel bedeutet. Ziel ist es ja nach wie vor, gegen einen menschlichen Gegner in dessen gewohnter Umgebung anzutreten.

2.4.b Kategorie „RoboCupSoccer“

In dieser Kategorie geht es um die initiale Problemstellung des Fußballspielens. Sie ist weiter unterteilt in:

- **Simulation league**

Eine Liga mit rein virtuellen Komponenten. Die Spieler sind Agenten, die auf einem virtuellem Spielfeld Fußball spielen.

- **Small-size robot league**

In dieser Liga dürfen Roboter bis 18cm Durchmesser in 5er-Teams antreten. Das Spielfeld ist nur wenig größer als eine Tischtennisplatte.

- **Middle-size robot league**

Hier sind Teams aus 4 Robotern mit bis zu 50cm Durchmesser zugelassen. Das Spielfeld ist mit maximal 16 Meter x 12 Meter entsprechend größer ausgelegt.

- **Four-legged robot league**

Diese Liga ist in 2.4.a ausführlich beschrieben.

- **Humanoid league**

Zweifüßige autonome Roboter sind Gegenstand dieser Liga, die sich in weitere Wettbewerbe gliedert, anstatt Fußballspiele auszutragem wie die vorhergehenden Ligen. Zu den Wettbewerben gehören unter anderem Gehen, (Ball-)Schießen und Strafstoß.

2.4.c Kategorie „RoboCupRescue“

RoboCupRescue wurde im Jahr 2000 eingeführt. Die grundlegende Problemstellung ist die von SAR (Search And Rescue) Einsätzen, bei denen das finden und bergen von verletzten Personen im Vordergrund steht.

Es wird weiter unterschieden in „*Rescue Real Robot League“* und „*Rescue Simulation League“*.

2.4.d Kategorie „RoboCupJunior“

RoboCupJunior dient Interessierten als Einführung in die generelle Thematik und zum Sammeln ersten Erfahrungen. Es gibt in dieser Kategorie Wettbewerbe für Rettung, Fußball und Tanz.

3 ERS-7

In diesem Abschnitt werden nun die für die Umsetzung relevanten Eigenschaften des *Sony AIBO ERS-7* beschrieben und erläutert, warum diese Eigenschaften bedeutsam für die Umsetzung sind.

Die technischen Voraussetzungen für das Stehen sind zum einen ausreichende Beweglichkeit der Hinterbeine, damit sich der Robotor aufrichten kann, und zum anderen geeignete Sensoren, um den Gleichgewichtszustand feststellen zu können. Außerdem müssen die Füße so modelliert sein, dass sie ein aufrechtes Stehen unterstützen. Je größer die Grundfläche ist, desto einfacher ist es, den Robotor aufrecht zu halten.

Natürlich werden später auch der Kopf und die Vorderbeine in andere Positionen gebracht. Dies geschieht aus technischen (Verlagerung des Schwerpunktes) und ästhetischen Gründen.

3.1 Beschleunigungssensoren

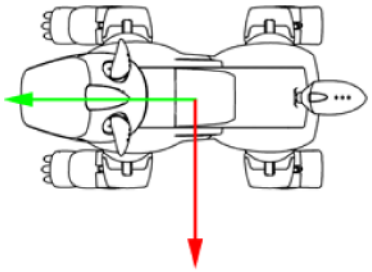

Abbildung 3.1.1: (Sicht von oben) grün: Vorne-Hinten-Achse; rot: Rechts-Links-Achse; Grafik aus [ERS04], erweitert um Darstellung der Achsen

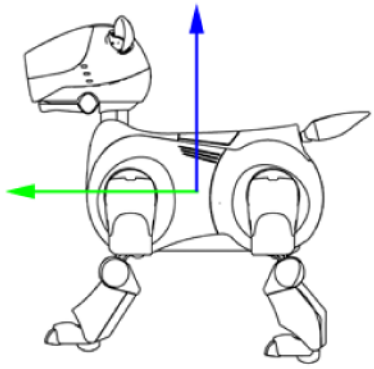

Abbildung 3.1.2: (Seitenansicht) grün: Vorne-Hinten-Achse; blau: Oben-Unten-Achse; Grafik aus [ERS04], erweitert um Darstellung der Achsen

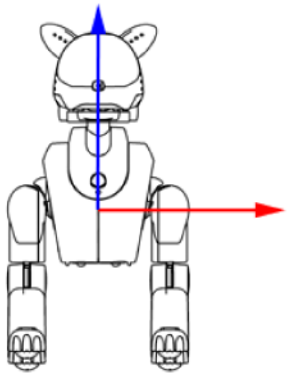

Abbildung 3.1.3: (Frontalansicht) blau: Oben-Unten-Achse; rot: Rechts-Links-Achse; Grafik aus [ERS04], erweitert um Darstellung der Achsen

Der *ERS-7* verfügt über drei Beschleunigungssensoren, die im Körper des Roboters sitzen. Die Sensoren messen jeweils die Beschleunigung entlang einer der drei Hauptachsen.

Aus der technischen Dokumentation geht jedoch nicht hervor, ob die Erdbeschleunigung in den Sensorwerten enthalten ist oder herausgerechnet wird.

Der Wertebereich jedes einzelnen Sensors umfasst das geschlossene Intervall [-19,613300 m/s^2; +19,613300 m/s^2]. Dies entspricht in jeder Richtung der 2-fachen Erdbeschleunigung ($g \approx 9{,}81\ m/s^2$).

3.2 Beingelenke

Die maximale technisch mögliche Beweglichkeit der hinteren Oberschenkel beträgt, ausgehend von der vierbeinigen Grundhaltung, jeweils 135 Grad nach vorne und 120 Grad nach hinten. Diese Werte werden jedoch softwareseitig auf 130° und 115° beschränkt.

Für die Unterschenkel sind 30° nach vorne und 127° nach hinten als maximale Drehwinkel gegeben. Die Software des *AIBOs* beschränkt auch diese und zwar auf 25° bzw. 122°.

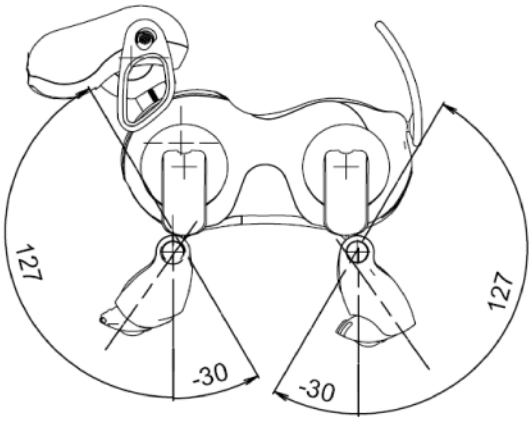

Abbildung 3.2.1: Die maximalen hardwaretechnisch machbaren Drehwinkel der Kniegelenke. Alle Angaben sind in Grad. Die Grafik wurde aus [ERS04] entnommen.

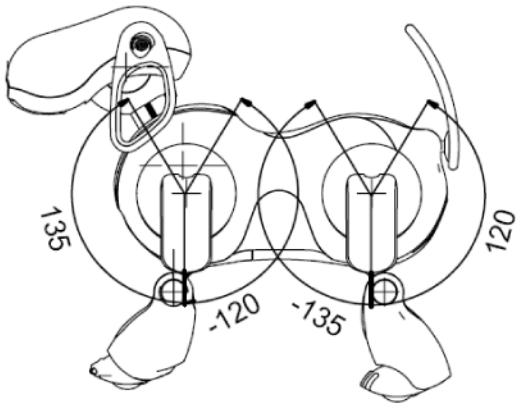

Abbildung 3.2.2: Die maximalen hardwaretechnisch machbaren Drehwinkel der Hüftgelenke. Alle Angaben sind in Grad. Die Grafik wurde aus [ERS04] entnommen.

Umgerechnet auf die angestrebte stehende Position bedeutet dies einen softwaretechnisch erlaubten Spielraum von 210° nach vorne und 25° nach hinten für die Oberschenkel. Der Bereich für die Unterschenkel wird nicht verändert, da dessen relative Ausrichtung zum Oberschenkel nicht verändert werden soll.

In der folgenden Grafik ist die angestrebte Position des *AIBOs* skizziert, wobei die veränderten Gliedmaße rot eingezeichnet sind.

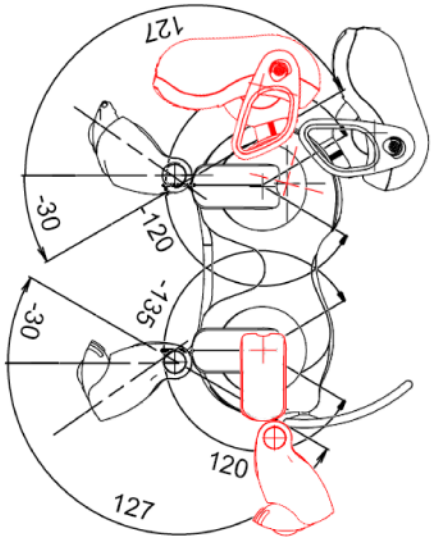

Abbildung 3.2.3: Diese Grafik aus [ERS04] wurde nachträglich durch die Stellung des AIBOs in aufrechter Position ergänzt. Rot eingezeichnet sind die ungefähren Positionen der Gliedmaße für diese Haltung. Auf die Darstellung der Vorderbeinposition wurde aus Gründen der Übersichtlichkeit verzichtet.

3.3 Füße

Wie man aus den bisherigen Grafiken entnehmen kann, sind die Fußsohlen abgerundet. Dies ist bei einem Vierbeiner nicht von großer Bedeutung, da hier die Grundfläche durch die vier Kontaktstellen der Beine mit dem Boden aufgespannt wird. Hier kann man die Berührungsstellen sogar zu Punkten idealisieren, ohne viel von der Grundfläche zu verlieren.

Bei einem Zweibeiner gibt es jedoch nur zwei Kontaktflächen, daher ist hier die Größe der Fußsohlen besonders wichtig.

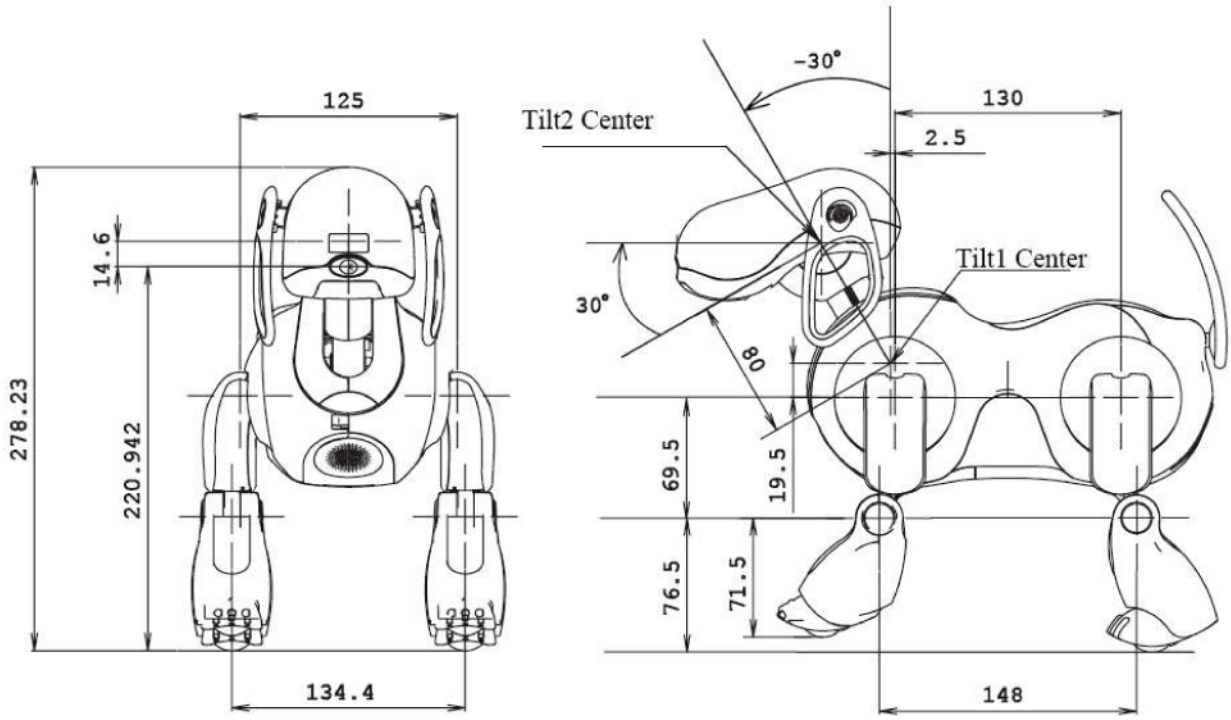

Abbildung 3.3.1: Abmessungen des ERS-7; entnommen aus [ERS04].

Aus den folgenden Grafiken kann man entnehmen, dass der Abstand der Hinterbeine 134,4 Millimeter beträgt und jeder Fuß eine Grundfläche von ca. 33x39,5 mm^2 hat. Daraus ergibt sich eine Grundfläche von 33 x 173,9 mm^2.

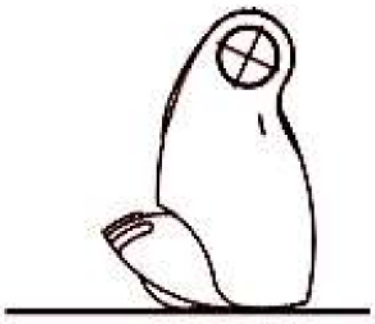

Abbildung 3.3.2: Vergrößerte Ansicht eines Hinterbeins vom Knie aus abwärts; entnommen aus [ERS04]

3.4 weitere Systemressourcen

Die Fähigkeit, alle notwendigen Berechnungen ausreichend schnell durchführen zu können, ist natürlich eine notwendige Bedingung für eine Realisierung. Dies führt direkt zu Geschwindigkeitaanforderungen an Prozessor, I/O und Servo-Motoren.

Der im *ERS-7* verwendete Prozessor ist ein *R7000* von *MIPS Technologies*. Es handelt sich dabei um einen 64-bit RISC Prozessor, der mit 576 Megahertz getaktet ist. [AEU04]

Diese Prozessorfamilie wird häufig in Embedded Systems wie Routern, Set-Top-Boxen und anderer Unterhaltungselektronik verwendet. *MIPS* steht für „*Microprocessor without interlocked pipeline stages*“.

Über das I/O-System und die Motoren lassen sich leider keine Aussagen treffen.

Ausgehend von der Komplexität des mir bekannten GermanTeam-Codes, der auf einem *ERS-7* unproblematisch läuft, und der im Vergleich dazu geringen Komplexität der vorgesehenen Implementierung, schätze ich, dass dieses System eine mehr als ausreichende Leistung besitzt. Ungeklärt bleibt alllein die Frage, ob die Motoren ausreichend schnell und stark sind.

4 Theoretische Grundlagen

4.1 Gleichgewicht

In der Physik bedeutet gemäß Definition *mechanisches Gleichgewicht*, dass sich alle am Körper angreifenden Kräfte und Drehmomente zu Null addieren.

Ferner wird zwischen verschiedenen Arten von Gleichgewicht unterschieden:

Stabiles, *labiles*, *indifferentes* und *metastabiles Gleichgewicht*.

Entscheidend für die Einordnung ist der Ort, an dem der Schwerpunkt unterstützt wird. Der Schwerpunkt eines Körpers ist jener Punkt, in dem er unterstützt werden muss, damit sich der Körper in jeder Position in einem mechanischen Gleichgewicht befindet. (vgl. [Breu88])

Ferner ist die potentielle Energie von Bedeutung, da sie in kinetische Energie umgewandelt werden kann. Das bedeutet, wenn der Körper sich in einer Lage mit minimaler potentieller Energie befindet, kann er von sich aus in keine Bewegung geraten; die dafür notwendige Energie muss ihm von außen zugeführt werden.

4.1.a Stabiles Gleichgewicht

Im stabilen Gleichgewicht ist der Körper senkrecht über dem Schwerpunkt unterstützt, so dass jede Änderung seiner Lage zu einem in die Ausgangslage rücktreibenden Drehmoment führt. Die potentielle Energie in der Ausgangslage ist minimal. (vgl. [Breu88])

Beispiel: Kugel in einer Senke.

4.1.b Labiles Gleichgewicht

Im labilen Gleichgewicht ist der Körper senkrecht unter dem Schwerpunkt unterstützt. Jede Lageänderung wird durch das entstehende Drehmoment noch weiter vergrößert. Die potentielle Energie des Körpers ist maximal. (vgl. [Breu88])

Beispiel: Ein auf der Spitze stehender Kegel oder ein auf einer seiner Kanten oder Ecken stehender Würfel.

4.1.c Indifferentes Gleichgewicht

Beim indifferenten Gleichgewicht befindet sich die Unterstützung im Schwerpunkt. Jede Lage ist eine Gleichgewichtslage und alle Lagen haben die gleiche potentielle Energie. [Breu88]

Beispiel: Kugel auf einer (unendlichen und absolut planen) Ebene.

4.1.d Metastabiles Gleichgewicht

Das metastabile Gleichgewicht ist ein stabiles Gleichgewicht, das in ein labiles Gleichgewicht übergehen kann. [Breu88]

Beispiel: Ein Würfel, der auf einer seiner Seiten steht. Je nach Maß der Auslenkung wird er auf seine ursprüngliche Standfläche zurückkippen oder auf eine andere seiner Seiten kippen.

4.1.e Standfestigkeit und Kippen

Ein Körper ist *standfest*, wenn sich die Projektion seines Schwerpunktes in Richtung der Erdbeschleunigung innerhalb seiner Auflagefläche befindet, andernfalls kippt er. [Breu88]

4.2 Der menschliche Gleichgewichtssinn

Der Gleichgewichtssinn des Menschen befindet sich bekanntlich in den beiden Innenohren. In jedem Innenohr befinden sich 5 sensorische Einheiten:

Drei so genannte Bogengänge, die näherungsweise senkrecht zueinander stehen, liefern Informationen über die Winkelbeschleunigungen des Kopfes.

Zwei weitere Organe dienen zur Wahrnehmung von horizontalen und vertikalen Beschleunigungen.

4.2.a Gehör

Das Gehör bietet außerdem weitere Unterstützung durch ein relativ genaues räumliches Hören, dass dazu beiträgt, die Position des Menschen in Beziehung zu anderen Geräuschquellen zu bestimmen.

4.2.b Sehapparat

Weiter unterstützt wird der Gleichgewichtssinn durch die visuelle Wahrnehmung, die auch zur optischen Lokalisierung von Geräuschquellen dienen kann und damit im Zusammenspiel mit dem Gehör eine genauere Abschätzung der Position des Menschen bietet.

Außerdem kann der Sehapparat wertvolle Informationen zur Lage und Lageveränderung des Menschen liefern. Dies geschieht insbesondere über die Fähigkeit zum räumlichen Sehen und damit verbunden der Objekterkennung.

4.2.c Muskel- und Sehnen- und Gelenkrezeptoren

Diese Rezeptoren liefern ein statisches Bild der Körperhaltung.

4.2.d Einschränkungen beim *AIBO*

Der *AIBO* kann nur die lineare Beschleunigung entlang der drei Achsen messen. Informationen über Winkelbeschleunigungen wären hilfreich, sind aber nicht notwendig, wenn man davon ausgeht, dass ein sich im Gleichgewicht befindlicher *AIBO* keine Beschleunigung erfährt, außer durch die Erdanziehungskraft.

Da der *AIBO* über eine Kamera verfügt, wäre eine Unterstützung durch Auswertung des Kamerabildes denkbar, würde aber bestimmte Voraussetzungen an die Umgebung erfordern. Im bereits vorhandenen Code für die RoboCup-Liga sind geeignete Module für diese Aufgabe bereits vorhanden, jedoch würde ihre Nutzung weiteren Aufwand nach sich ziehen, der wahrscheinlich den Umfang dieser Arbeit übersteigen würde. Die Idee sollte jedoch nicht verworfen werden, auch wenn sie hier nicht umgesetzt werden soll.

Der *AIBO* verfügt ebenfalls über Mikrophone, je eines an den Stellen, an denen die Ohren am Kopf befestigt sind. Damit ist die Grundvoraussetzung für räumliches Hören gegeben. Auch diese Idee soll aus ganz ähnlichen Gründen wie das räumliche Sehen nicht umgesetzt werden.

Allerdings verfügt der *AIBO* jederzeit über genaue Angaben zu der Stellung seiner Gelenke und kann damit Rückschlüsse auf seine Haltung ziehen. Um dies tatsächlich zu nutzen, bräuchte man allerdings ein genaues Modell des *AIBO*s und eine geeignete Auswertung, um brauchbare Aussagen finden zu können. Dies ist aufwändig und daher soll diese Unterstützung nur implementiert und verwendet werden, wenn eine einfache Rückkopplung über die Beschleunigungssensoren nicht ausreichend ist.

4.3 Regelkreis

Als Grundmodell für die Implementierung soll ein Regelkreis dienen, dessen Aufgabe es sein wird, den *AIBO* im Gleichgewicht zu halten.

4.3.a Allgemeiner Regelkreis

In einem Regelkreis wird eine bestimmte messbare Größe in Richtung eines bestimmten Sollwertes reguliert. Dabei wird immer wieder der Kreislauf aus Messen und Regeln durchlaufen.

Die zu regelnde Größe wird *Regelgröße* genannt. Ihr Wert wird im *Messglied* gemessen und als *Istwert* an den *Regler* gegeben. Dem Regler ist von außerhalb des Systems ein *Sollwert* vorgegeben. Im Regler wird nun der Istwert mit dem Sollwert verglichen und in Abhängigkeit von dem Ergebnis die *Stellgröße* berechnet und diese an das *Stellglied* weitergegeben. Das Stellglied beeinflusst daraufhin direkt oder indirekt die Regelgröße.

Regelgröße und Stellglied stellen Teile der Stellstrecke dar. Diese Strecke stellt eine Abstraktion von dem konkreten Objekt dar, der die Regelgröße beherbergt. Die Strecke wird durch die *Störgröße* beeinflusst.

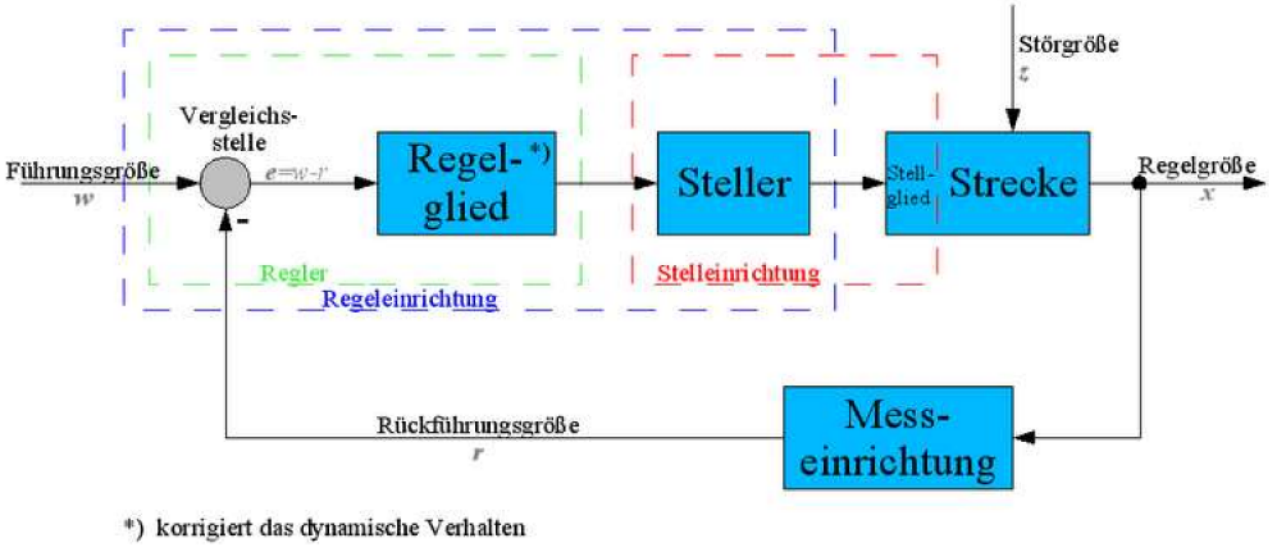

Abbildung 4.3.1: Grafik wurde [WPRT04] entnommen. Der Regler in diesem Text entspricht der Regeleinrichtung (blauer Kasten) in der Grafik. Istwert ist hier als Rückführungsgröße bezeichnet, Sollwert als Führungsgröße. Messglied und Messeinrichtung sind ebenfalls identisch.

4.3.b Konkreter Regelkreis

Bevor die zuvor genannten Begriffe genauer spezifiziert werden können, soll zuerst das ganze natürlichsprachlich beschrieben werden:

Wenn der *AIBO* steht und dann aus dem Gleichgewicht gerät, kippt er zu einer Seite weg. Anders ausgedrückt, der Kopf des *AIBO*s dreht sich um den (idealisierten) Kontaktpunkt der Füße mit dem Boden. Dieses Kippen kann durch verschiedene Ursachen hervorgerufen werden, wie z.B. Veränderung des Bodens (stellenweises Heben/Senken, Balancieren auf der Hand etc.) oder Krafteinflüsse von Außerhalb – mit Ausnahme der Schwerkraft – z.B. Stoßen oder Schieben des *AIBO*s.

Kippen bedeutet, dass der Schwerpunkt sich nicht mehr über der Grundfläche befindet. Um das auszugleichen, wird die Körperhaltung verändert, bis der Schwerpunkt wieder über der Grundfläche ruht. Solange der *AIBO* kippt, wird eine Beschleunigung messbar sein. Welche Größe sie hat, hängt von der genauen Position der Sensoren ab und kann leider nur experimentell

bestimmt werden.

Wird keine Beschleunigung gemessen, befindet sich der *AIBO* also entweder in einer stehenden Lage oder er liegt nach einem Fall auf dem Boden.

Bringt man den *AIBO* zu Beginn in eine stehende Gleichgewichtslage, so muss es das Ziel des Regelkreises sein, den gemessenen Beschleunigungen entgegenzuwirken, bis keine mehr gemessen wird.

Damit ergibt sich Folgendes:

- **Regelgröße:** Der Beschleunigungsvektor des *AIBO*s.
- **Messglied:** Die drei Beschleunigungssensoren.
- **Istwert:** Von den Sensoren gemessene Beschleunigung.
- **Sollwert:** keine Beschleunigung (=0)
- **Regler:** Eine geeignete Softwareimplementation, die noch zu finden ist.
- **Stellgröße:** Die Körperhaltung des *AIBO*s.
- **Stellglied:** Die einzelnen Motoren in den Gelenken.
- **Störgröße:** Äußere Krafteinflüsse.

Die Beschleunigungssensoren messen die Beschleunigung an einem bestimmten Punkt im *AIBO* und stellen ihre Messwerte der auf dem *AIBO* gerade laufenden Software zur Auswertung zur Verfügung. Die Software wird anhand der Richtung und Größe der Werte und evtl. unter Einbeziehung früherer Messwerte die Körperhaltung des *AIBO*s verändern, indem sie einzelne Gelenke anspricht und deren Stellung verändert.

5 Implementation

5.1 Programmierung der AIBOs

*AIBO*s können auf verschiedene Arten und Weisen programmiert werden. Sony bietet drei verschiedene Software Development Kits an, die verschiedenen Zwecken dienen.

5.1.a Verfügbare SDKs

- OPEN-R
- R-CODE
- *AIBO* Remote Framework

OPEN-R ist Framework für C++, dass dem Entwickler eine sehr systemnahe Programmierung erlaubt. OPEN-R bietet die Möglichkeit, Sensoren und Gelenke direkt anzusprechen.

R-CODE ist eine einfache Skriptsprache, die hauptsächlich auf die Steuerung von oberflächlichen Verhaltensweisen ausgelegt ist, wie z.B. Laufen, Sprachkommandos erkennen.

AIBO Remote Framework basiert auf Visual C++ und ermöglicht die Erstellung von Software, die unter Windows läuft und über WLAN einen *AIBO* fernsteuert. Dabei werden die systemnahe Ebene von OPEN-R und die oberflächennahe Ebene von R-CODE zur Verfügung gestellt.

[SDE04]

5.1.b Auswahl eines SDK

Kriterien für die Auswahl eines SDKs sind:

- Abfragemöglichkeit der Sensoren
- Möglichkeit zur Veränderung der Haltung
- vollständige Autonomie

Diese Kriterien erfüllt alleine das OPEN-R SDK, daher soll dieses verwendet werden.

5.1.c Beschreibung des SDKs

OPEN-R Software besteht aus einzelnen OPEN-R Objekten, die über eine spezielle Inter-Objekt-Kommunikation miteinander gekoppelt sind. Im Folgenden werden OPEN-R Objekte nur noch kurz als Objekte bezeichnet.

Alle Objekte laufen gleichzeitig auf dem *AIBO*, sind also jedes für sich ein eigenständiger Prozess aus Anwendungssicht.

Die Objekte kommunizieren miteinander indem sie Nachrichten austauschen. Eine Nachricht besteht aus einem so genannten *Selector* und Nutzdaten. Über den Selector wird am Empfängerobjekt die Funktion ausgewählt, welche die Nutzdaten verarbeiten soll.

Alle Objekte sind „single-threaded“, es können also nicht mehrere Funktionen gleichzeitig an einem Objekt ausgeführt werden. Das bedeut auch, dass immer nur eine Nachricht zur Zeit verarbeitet werden kann. Darum werden Nachrichten in einer Warteschlange gespeichert, bis das Objekt wieder frei ist, um die nächste Nachricht aus der Schlange zu bearbeiten.

Die Kommunikationsbeziehungen zwischen allen Objekten ist bereits vor dem Laden derselbigen bekannt. Jedes Objekt weiß, welchen anderen Objekte es Nachrichten übermitteln könnte und für wen es Nachrichtenempfänger sein kann. Dadurch ist es möglich von jedem Objekt zu verlangen, dass es alle seine potentiellen Sender benachrichtigt, wenn es empfangsbereit ist.

Empfänger werden in der OPEN-R-Terminologie als *Observer* bezeichnet, während Sender dort *Subject* heißen. Zur Kommunikation mit dem AIBO dienen die bereits vorgegebenen Objekte *OVirtualRobotComm* und *OPENR*. [PG04][L2REF04]

5.1.d OPEN-R Kommunikation im Detail

Wie bereits gesagt, stehen die Kommunikationsbeziehungen schon vor dem Laden der Objekte fest. Dazu befindet sich auf dem geladenen Dateisystem einen Datei „stub.cfg“, welche diese Infomationen beinhaltet.

Für die Identifizierung der Kommunikationsendpunkte werden sog. *Services* definiert. Ein Service beschreibt, welche konkreten Methoden für die Nutzung eines Endpunktes am Zielobjekt aufgerufen werden sollen. Jeder Service hat einen eindeutigen Namen, der sich aus dem Objektnamen, einem eindeutigen Zusatznamen, verwendetem Datentyp und dem Servicetyp ('S' für Subject, 'O' für Object) in genau dieser Reihenfolge zusammensetzt.

Die endgültigen Kommunikationsbeziehungen werden in einer weiteren Datei namens „CONNECT.CFG“ definiert. Hier werden immer zwei Services miteinander verbunden.

Beispiel für „stub.cfg“ aus [PG04]:

```
ObjectName : ObjectA
NumOfOSubject : 1
NumOfOObserver : 1
Service : "ObjectA.SendString.char.S", null, subject_a()
Service : "ObjectA.DummyObserver.DontConnect.O", null, null

ObjectName : ObjectB
NumOfOSubject : 1
NumOfOObserver : 1
Service : "ObjectB.DummySubject.DontConnect.S", null, null
Service : "ObjectB.ReceiveString.char.O", null, observer_b()
```

Beispiel für „stub.cfg“ [PG04]

Beispiel für „CONNECT.CFG“ aus [PG04]:

```
ObjectA.SendString.char.S          ObjectB.ReceiveString.char.O
```

Beispiel für „CONNECT.CFG“ [PG04]

Hier wird für Objekt A der Subject-Service „SendString“ deklariert und für Objekt B der Observer-Service „ReceiveString“. Beide werden in der „CONNECT.CFG“ miteinander verbunden.

Für OVirtualRobotComm sind drei Services vorgegeben:

- *OVirtualRobotComm.Effector.OCommandVectorData.O*
- *OVirtualRobotComm.Sensor.OSensorFrameVectorData.S*
- *OVirtualRobotComm.FbkImageSensor.OFbkImageVectorData.S*

Nur die ersten beiden sind hier von Interesse, da über sie Befehle an Gelenke gesendet bzw. Sensordaten abgefragt werden können.

[PG04][L2RG04]

5.2 Abfragen der Messwerte

Für die Abfrage der Messwerte der Beschleunigungssensoren wird der Service *OvirtualRobotComm.Sensor.OSensorFrameVectorData.S* verwendet. Von ihm erhält man ein Objekt vom Typ *OSensorFrameVectorData,* welches die gegenwärtigen Informationen aller Sensoren enthält.

Bevor man jedoch einen Sensor auslesen kann, muss man ihn vorher einschalten und zum Lesen öffnen. Beides geschieht über das Objekt *OPENR*, welches ebenfalls im OPEN-R SDK deklariert ist.

Das Einschalten der Sensoren erfolgt nur zusammen mit dem Einschalten der Motoren. Hierfür wird die Methode *OPENR::SetMotorPower(OPower power)* mit der vordefinierten Konstanten *opowerON* als Parameter aufgerufen.

5.2.a Primitives

Sensoren und Gelenke werden als *Primitives* bezeichnet, was sich hier am besten als „Basiselemente“ übersetzten lässt. Ein *Primitive* wird über die Methode *OPENR::OpenPrimitive (char* locator, &m_sensorID)* geöffnet. Der Parameter *locator* gibt den sogenannten *Locator*, den Namen des *Primitives,* an. Die *Locators* aller *Primitives* des *ERS-7* sind in [ERS04] aufgelistet. Der Rückgabeparameter *m_sensorID* enthält nach dem Methodenaufruf eine ID, welche später benötigt wird, um die gewünschten Sensordaten aus dem *OSensorFrameVectorData*-Objekt zu gewinnen.

`PRM:/a1-Sensor:a1`	Acceleration sensor(front-back)
`PRM:/a2-Sensor:a2`	Acceleration sensor(right-left)
`PRM:/a3-Sensor:a3`	Acceleration sensor(up-down)

Auszug aus [ERS04]: Locators der drei Beschleunigungssensor-Primitives.

5.2.b OSensorFrameVectorData

Nach dem Öffnen des Primitives können nun die Sensordaten abgefragt werden.

```
struct OSensorFrameVectorData {
      ODataVectorInfo         vectorInfo;
      OSensorFrameInfo        info[1];

      void SetNumData(size_t ndata){
            vectorInfo.numData = ndata;
      }
      OSensorFrameInfo* GetInfo(int index){
            return &info[index];
      }
      OSensorFrameData* GetData(int index) {
            return (OsensorFrameData*)
                  ((byte*)&vectorInfo+info[index].dataOffset);
      }
};
```

Struktur von OSensorFrameData; C++ [L2RG04]

Von Interesse sind hier der Inhalt von *vectorInfo*, einem Objekt vom Typ *ODataVectorInfo*, und dem *OSensorFrameInfo*-Array *info*, auf das mittels der Methoden *GetInfo(int index)* und *GetData (int index)* zugegriffen werden kann. Dabei enthält das von *GetInfo* zurückgelieferte *OSensorFrameInfo*-Objekt Informationen über das zum gleichen Index gehörende *OSensorFrameData*-Objekt.

In der Struktur von *vectorInfo* befindet sich die Variable *maxNumData*. Diese gibt Auskunft darüber, wieviele Elemente das bereits genannte *OSensorFrameInfo*-Array hat. Von diesem Array sind aber nicht notwendiger Weise alle Elemente belegt. Die Anzahl der tatsächlich belegten Elemente ist in der Variablen *numData* enthalten.

Der Parameter *index* der *GetInfo*- und *GetData*-Methoden korreliert mit *numData* so, dass *numData* dem um eins erhöhten maximalen Wert für *index* entspricht.

```
struct ODataVectorInfo {
      MemoryRegionID    memRegionID;
      void*             physAddr;
      size_t            offset;
      size_t            totalSize;
      OdataType         type;
      size_t            infoOffset;
      size_t            infoSize;
      size_t            maxNumData;
      size_t            numData;
      OVRSyncKey        syncKey;
      longword          wait;
      size_t            optOffset;
      size_t            optSize;
      longword          padding[3];
      byte              optional[odataOPTIONAL_MAX];
};
```

Struktur von ODataVectorInfo; C++ [L2RG04]

5.2.c OSensorFrameInfo

Um nun an die gewünschten Sensordaten zu kommen, muss GetInfo mit allen erlaubten Indizies aufgerufen und die jeweils zurückgelieferte *OSensorFrameInfo*-Struktur auf eine passende *primitiveID* überprüft werden.

```
struct OSensorFrameInfo {
      ODataType         type;
      OPrimitiveID      primitiveID;
      longword          frameNumber;
      size_t            numFrames;
      size_t            frameSize;
      size_t            dataOffset;
      size_t            dataSize;
      longword          padding[1];

      void Set(ODataType t, OPrimitiveID id, size_t nframes) {
            type = t;
            primitiveID = id;
            numFrames = nframes;
      }
};
```

Sturktur von OSensorFrameInfo; C++ [L2RG04]

Wurde eine passende *primitiveID* gefunden, kann mit dem gleichen Index die zugehörige *OSensorFrameData*-Struktur abgerufen werden, welches ein Array mit den gesuchten Sensordaten enthält. Die Größe dieses Arrays korrespondiert zu dem Wert in *numFrames* aus obiger Datenstruktur.

5.2.d Frames

Das System des AIBOs kennt nur eine diskrete Zeit, deren kleinste Einheit als *Frame* bezeichnet wird. Ein *Frame* hat eine Dauer von 8ms und eine eindeutige Nummer. Überschreitet die *Frame*-Nummer den erlaubten Maximalwert, so wird sie auf Null zurückgesetzt.

Sensordaten, die über den hier beschriebenen Weg abgefragt werden, enthalten nicht nur den Messwert eines *Frames*, sondern den von mehreren. Wieviele *Frames* tatsächlich bei einem Aufruf zurückgeliefert werden, hängt von verschiedenen Faktoren ab und ist auch nicht weiter relevant.

Die *Frame*-Nummer des ersten Frames ist in der Variablen *frameNumber* enthalten. Alle weiteren *Frames* sind fortlaufend nummeriert mit *frameNumber*+1, *frameNumber*+2 etc.

5.2.e OSensorFrameData

```
struct OSensorFrameData {
      OSensorValue      frame[osensorframeMAX_FRAMES];
};
```

Struktur von OsensorFrameData; C++ [L2RG04]

OSensorValue ist eine Abstraktion von der konkreten Datenstruktur. Je nach Sensor wird *OAcceleration, OAngularVelocity, OTemperature, OForce, OPressure, OLength, OSwitchStatus,* oder *OJointValue* verwendet. Der tatsächliche Typ ist in der Variablen *type* in *OSensorFrameInfo* enthalten.

5.2.f Beispielprogramm

Als Grundlage für das Beispielprogramm dienten Quelltextfragmente aus [PG04] und [L2RG04].

- „stub.cfg“

```
ObjectName: SensorSample
NumOfOSubject: 1
NumOfOObserver: 1
Service: ”SensorSample.ReadSensor.OSensorFrameVectorData.O”,null,ReadSensor()
Service: ”SensorSample.DummySubject.DoNotConnect.S”, null, null
```

„stub.cfg“ des Beispielprogramms

- „CONNECT.CFG“

```
OVirtualRobotComm.Sensor.OSensorFrameVectorData.S
SensorSample.ReadSensor.OSensorFrameVectorData.O
```

„CONNECT.CFG“ des Beispielprogramms. Zu beachten ist, dass es sich hier um eine einzige Zeile handelt und nicht um zwei, wie dargstellt.

- SensorSample.h

```
#ifndef SensorSample.h DEFINED
#define SensorSample.h DEFINED

#include <OPENR/OObject.h>
#include <OPENR/OSubject.h>
#include <OPENR/OObserver.h>
#include "def.h"

//Namen der Sensor-primitives
static const char* const LOCATOR_ACCEL_FRONT_BACK =
      "PRM:/a1-Sensor:a1";
static const char* const LOCATOR_ACCEL_LEFT_RIGHT =
      "PRM:/a2-Sensor:a2";
static const char* const LOCATOR_ACCEL_TOP_BOTTOM =
      "PRM:/a3-Sensor:a3";

//Jedes OPEN-R-Objekt muss von OObject erben
class SensorSample : public OObject {

public:
      SensorSample( );
      virtual ~SensorSample( ) {}

      //numOfSubject und numOfObserver werden durch Tools
      //des OPEN-R SDKs während Build-Prozesses generiert
      OSubject*   subject[ numOfSubject ];
      OObserver*  observer [ numOfObserver ];
```

```
      //Der hiesige Endpunkt der Kommunikation
      void ReadSensors ( const ONotifyEvent& event );

      //Methoden aus OOBject die überladen werden müssen
      virtual OStatus DoInit ( const OSystemEvent& event );
      virtual OStatus DoStart ( const OSystemEvent& event );
      virtual OStatus DoStop ( const OSystemEvent& event );
      virtual OStatus DoDestroy ( const OSystemEvent& event );

private:
      void printSensorData( OSensorFrameVectorData *vecData,
      int index, char* sensorName );

      OPrimitiveID      m_AccelSensorFrontBack_ID;
      OPrimitiveID      m_AccelSensorLeftRight_ID;
      OPrimitiveID      m_AccelSensorTopBottom_ID;
};
#endif // SensorSample.h DEFINED
```

C++ Header des Beispielprogramms

- SensorSample.cpp

```
#include <OPENR/OPENRAPI.h>
#include <OPENR/OUnits.h>
#include <OPENR/OSyslog.h>
#include <OPENR/core_macro.h>
#include "SensorSample.h"

SensorSample::SensorSample () : m_isInitialized ( false ) ,
      m_MatchingSensorIndex (-1) {
}

OStatus SensorSample :: DoInit( const OSystemEvent& event ) {
      //in OPEN-R vordefinierte Makros
      //zum Initialisieren der Kommunikationsbeziehungen
      NEW_ALL_SUBJECT_AND_OBSERVER;
      REGISTER_ALL_ENTRY;
      SET_ALL_READY_AND_NOTIFY_ENTRY;

      //Sensor-primitives öffnen
      OPENR::OpenPrimitiv( LOCATOR_ACCEL_FRONT_BACK,
            &m_AccelSensorFrontBack_ID);
      OPENR::OpenPrimitive( LOCATOR_ACCEL_LEFT_RIGHT,
            &m_AccelSensorLeftRight_ID);
      OPENR::OpenPrimitive( LOCATOR_ACCEL_TOP_BOTTOM,
            &m_AccelSensorTopBottom_ID);

      // Sensoren einschalten
      OPENR::SetMotorPower( opowerON );

      return oSUCCESS;
}

OStatus SensorSample::DoStart( const OSystemEvent& event ) {
      //in OPEN-R vordefinierte Makros
      //zum Starten der Kommunikationsbeziehungen
      ENABLE ALL SUBJECT;
      ASSERT READY TO ALL OBSERVER;
      return oSUCCESS;
}

OStatus SensorSample::DoStop( const OSystemEvent& event ) {
```

```
      //in OPEN-R vordefinierte Makros
      //zum Abbauen der Kommunikationsbeziehungen
      DISABLE ALL SUBJECT;
      DEASSERT READY TO ALL OBSERVER;
      return oSUCCESS;
}

OStatus SensorSample::DoDestroy( const OSystemEvent& event ) {
      //in OPEN-R vordefinierte Makros
      //zum Beenden der Kommunikationsbeziehungen
      DELETE ALL SUBJECT AND OBSERVER;
      return oSUCCESS;
}

void SensorSample::ReadSensor ( const ONotifyEvent& event ) {
      OSensorFrameVectorData *vecData =
            (OSensorFrameVectorData*)event.Data( 0 );
      //Indizes der Sensordaten
      int frontBackIndex;
      int leftrightIndex;
      int topBottomIndex;

      //alle Indizes bestimmen
      int foundCount = 0;
      int i = 0;
      while((i < vecData->vectorInfo.numData) && (foundCount<3)) {
            OSensorFrameInfo *frameInfo = vecData->GetInfo( i );
            OPrimitiveID primitive_ID = frameInfo->primitive_ID;
            if( primitive_ID == m_AccelSensorFrontBack_ID ) {
                  frontBackIndex = i;
                  foundCount++;
            }
            else if( primitive_ID == m_AccelSensorLeftRight_ID ) {
                  leftRightIndex = i;
                  foundCount++;
            }
            else if( primitive_ID == m_AccelSensorTopBottom_ID ) {
                  topBottomIndex = i;
                  foundCount++;
            }
            i++;
      }

      printSensorData(vecData, frontBackIndex, „front-back“);
      printSensorData(vecData, leftRightIndex, „left-right“);
      printSensorData(vecData, topBottomIndex, „top-bottom“);

      //Wird an dieser Stelle als Teil des
      //Inter-Objekt-Kommunikations-Protokolls erwartet
      observer[ obsReadSensor ]->AssertReady( );
}

void SensorSample::printSensorData( OSensorFrameVectorData *vecData,
                                    int index,
                                    char* sensorName ) {
      OSensorFrameInfo *frameInfo =
            vecData->GetInfo ( index );
      OSensorFrameData *frameData =
            vecData->GetData ( index );
      int frameNumber = frameInfo->frameNumber;

      //Ausgabe der Sensordaten
      for ( int i=0; i < frameInfo->numFrames; i++ ) {
```

```
            OSYSPRINT((
                  "Frame# %d, Sensor %s, Value %d\n",
                  (frameNumber + i),
                  sensorName,
                  frameData->frame[i].value ));
      }
}
```

C++ Quelltext des Beispielprogramms

5.3 Stellen der Gelenke

Um einem Gelenk einen Befehl zu erteilen, wird der Service *OVirtualRobotComm.Effector.OCommandVectorData.O* verwendet, der einen Parameter vom Typ *OCommandVectorData* benötigt. Über den selben Weg können auch die LEDs des AIBOs angeprochen werden. Dies soll aber nur der Vollständigkeit halber erwähnt werden.

Der Vorgang des Befehlesendens ist dem des Sensordatenabfragens sehr ähnlich. Daher wird hier nur eine knappe Beschreibung geliefert.

Die hier verwendeten *Primitives* werden in der Dokumentation als *Joints* bezeichnet, was direkt mit „Gelenk“ übersetzt werden kann.

Es wird auch hier wieder über *OPENR::OpenPrimitive()* zuerst ein *Primitive* geöffnet. *Locatoren* für die *Joints* sind ebenfalls in [ERS04] enthalten.

```
PRM:/r2/c1-Joint2:21          Left front legJ1
PRM:/r2/c1/c2-Joint2:22       Left front legJ2
PRM:/r2/c1/c2/c3-Joint2:23    Left front legJ3

PRM:/r3/c1-Joint2:31          Left rear legJ1
PRM:/r3/c1/c2-Joint2:32       Left rear legJ2
PRM:/r3/c1/c2/c3-Joint2:33    Left rear legJ3

PRM:/r4/c1-Joint2:41          Right front legJ1
PRM:/r4/c1/c2-Joint2:42       Right front legJ2
PRM:/r4/c1/c2/c3-Joint2:43    Right front legJ3

PRM:/r5/c1-Joint2:51          Right rear legJ1
PRM:/r5/c1/c2-Joint2:52       Right rear legJ2
PRM:/r5/c1/c2/c3-Joint2:53    Right rear legJ3
```

Locators aller Bein-Joints [ERS04]

Die mit J1 bezeichneten *Joints* sind jeweils Hüft- bzw. Schultergelenke, J3 identifiziert Kniegelenke und J2 steht für die Gelenke, welche ein seitliches Abspreizen der Beine ermöglichen.

5.3.a OCommandVectorData

```
struct OCommandVectorData {
      ODataVectorInfo   vectorInfo;
      OCommandInfo      info[1];

      void SetNumData(size_t ndata){
            vectorInfo.numData = ndata;
      }

      OCommandInfo* GetInfo(int index) {
            return &info[index];
      }

      OCommandData* GetData(int index) {
            return (OCommandData*)((byte*)&vectorInfo +
```

```
                    info[index].dataOffset);
        }
};
```

Struktur von OCommandVectorData; C++

OCommandVectorData enthält genau wie *OSensorFrameVectorData* ein Informationsobjekt mit Details über die enthaltenen Daten. Dieses ist ebenfalls vom Typ *ODataVectorInfo*. Auch hier entspricht *vectorInfo.maxNumData* der Größe des *info*-Arrays und *vectorInfo.numData* der Anzahl der tatsächlich belegten Elemente. Das *info*-Array ist hier allerdings vom Typ *OCommandInfo*.

Die Methoden *GetInfo* und *GetData* verhalten sich wie die namensgleichen Methoden in *OSensorFrameVectorData*, nur dass hier der Rückgabetyp *OCommandInfo* bzw. *OCommandData* ist.

5.3.b OCommandInfo und OCommandData

Diese Datenstruktur unterscheidet sich von dem bereits bekannten *OSensorFrameInfo* nur durch den Typnamen und die Semantik.

Hier wird mittels *primitiveID* der Empfänger des Befehls angegeben und über *type* der Befehlstyp spezifiziert.

```
struct OCommandInfo {
      ODataType         type;
      OPrimitiveID      primitiveID;
      longword          frameNumber;
      size_t            numFrames;
      size_t            frameSize;
      size_t            dataOffset;
      size_t            dataSize;
      longword          padding[1];

      void Set(ODataType t, OPrimitiveID id, size_t nframes) {
            type = t;
            primitiveID = id;
            numFrames = nframes;
      }
};
```

Struktur von OCommandInfo; C++

OCommandData besteht wie sein Äquivalent *OSensorFrameData* nur aus einem einzigen Array, hier jedoch vom Typ *OCommandValue*.

```
struct OCommandData {
      OCommandValue value[ocommandMAX_FRAMES];
};
```

Struktur von OCommandData; C++

5.3.c OCommandValue

OCommandValue ist wie *OSensorValue* nur ein abstrakter Datentyp. Der konkrete Typ wurde bereits in *OCommandInfo.type* spezifiziert und ist einer der folgenden Typen:

OJointCommandValue2, *OJointCommandValue3* oder *OLEDCommandValue2*.

OLEDCommandValue2 ist, wie man am Namen erkennen kann, für das Ansteuern von LEDs gedacht und *OJointCommandValue3* trägt Befehlsdaten für die Ohren des AIBOs.

Wir benutzten hier also *OJointCommandValue2*.

```
struct OJointCommandValue2 {
      slongword   value;
      slongword   padding;
};
```

Struktur von OJointCommandValue2; C++

Die Variable *value* nimmt hier den einzustellenden Winkel des Gelenks auf.

5.4 Auswerten der Messwerte

Dies ist der komplizierteste Teil des Ganzen: Eine angemessene Reaktion auf eine gemessene Beschleunigung zu bestimmen.

Die Gesamtbeschleunigung, die sich aus den drei messbaren Beschleunigungsvektoren zusammensetzt, wird hier jedoch nicht von Interesse sein, da zum einen eine der drei Komponenten nicht relevant ist (s. 5.4.b) und zum anderen es einfacher ist, auf eine der beiden übrigen Komponenten direkt zu reagieren (5.4.c).

5.4.a Grundhaltung des AIBOs

Im Folgenden wird von der in Abbildung 3.2.3 dargestellten Haltung ausgegangen. Dies bedeutet eine Basiseinstellung der Gelenke mit:

- Hüftgelenke rechts/links (J1): 90°
- Kniegelenke rechts/links (J3): 0°
- Spreizgelenke rechts/links (J2): unbestimmt, jedoch scheint ein Wert >= 0° aus Stabilitätsgründen sinnvoll. Das softwaretechnische Limit liegt bei 88°, allerdings kann man diese Beinposition nicht mehr als Stand bezeichnen, sondern muss dann von einem Spagat reden. Autonomes Spagat machen ist nicht Gegenstand dieser Arbeit, wäre aber sicherlich auch eine Herausforderung.

5.4.b Reaktion auf vertikale Beschleunigung

In aufrecht stehender Position entspricht dies einem Wert ungleich Null gemessen entlang der Vorne-Hinten-Achse (s. Abb. 3.1.1 u. 3.1.2). Da dies in dieser Haltung keinen Einfluss auf die Beziehung von Grundfläche und Schwerpunkt zueinander hat, kann dieser Beschleunigungsvektor ignoriert werden.

5.4.c Reaktion auf horizontale Beschleunigung

Dies entspricht einer Beschleunigung entlang der Rechts-Links-Achse und der Oben-Unten-Achse (s. Abb. 3.1.1 bis 3.1.3). Die Bezeichnungen der Achsen, auch wenn sie im folgenden Kontext manchmal verwirrend sein können, werden aus Gründen der Kompatibilität zu den vorangegangenen Abschnitten beibehalten.

Um entlang einer dieser Achsen eine Beschleunigung messen zu können, muss einer der folgenden Fälle eintreten:

1. Der *AIBO* kippt.
2. Der *AIBO* wird horizontal verschoben. Wenn die beschleunigende Kraft nicht ideal am Körper angreift, wird dies wegen der Massenträgheit und der zusätzlichen Bremswirkung der umgebenden Luft unweigerlich zu einem Kippen führen.

Man kann also in diesem speziellen Fall annehmen, dass eine gemessene horizontale Beschleunigung eine unerwünschte Kippbewegung mit sich bringt, der entgegenzuwirken ist.

Die einzige Möglichkeit, einem Kippen entgegen zu wirken, ist den Schwerpunkt zu verlagern.

Da der AIBO keine Hüfte besitzt wie der Mensch und daher nicht auf gleiche Art und Weise seine Hauptmasse verlagern kann, muss ein anderes Mittel gefunden werden.

Entlang der Rechts-Links-Achse gibt es die Möglichkeit, den Spreizwinkel der Standbeine so zu verändern, dass sich der Körper zu einer Seite neigt. Außerdem können die Vorderbeine wie Arme abgewinkelt werden, um den Schwerpunkt zu verschieben.

Entlang der Oben-Unten-Achse kann der Körper des AIBOs in den Hüftgelenken nach vorne oder hinten gebeugt werden. Zudem können auch hier die Vorderbeine/Arme zu Hilfe genommen werden, um den Schwerpunkt zu verlagern. Dies gilt außerdem auch für den Kopf.

5.4.d Vorschlag eines Regelkreises

Im Folgenden sei die Kippseite die Seite zu der der AIBO hinkippt.

Wird eine Beschleunigung entlang der Rechts-Links-Achse festgestellt, wird Folgendes durchgeführt:

1. Anhebung des der Kippseite entgegengesetzten Armes um 1°, höchstens jedoch bis zu dem maximal erlaubten Winkel von 88°.
2. Senkung des auf der Kippseite gelegenen Armes um 1°, höchstens jedoch bis zu dem Winkel von 0° (Arm liegt am Körper an)
3. Abspreizung des der Kippseite entgegengesetzten Beines um 1°, höchstens jedoch bis zu dem maximal erlaubten Winkel von 88°.
4. Ranziehen des auf der Kippseite gelegenen Beines um 1°, höchstens jedoch bis zu dem Winkel von 0°.

Wird eine Beschleunigung entlang der Oben-Unten-Achse festgestellt, wird Folgendes durchgeführt:

1. Drehung beider Hüftgelenke um je 1° entgegen der Kipprichtung, jedoch nur bis zum maximal erlaubten Winkel von 210° nach vorne und 25° nach hinten.
2. Drehung beider Kniegelenke um je 1° entgegen der Kipprichtung, jedoch nur bis zum maximal erlaubten Winkel von 25° nach vorne und 122° nach hinten.
3. Drehung beider Arme um je 1° entgegen der Kipprichtung, jedoch nur bis zum maximal erlaubten Winkel von 210° nach vorne und 25° nach hinten.

6 Abschließendes

Die mir erlaubten 6 Wochen Bearbeitungszeit reichten leider nicht aus, um die bisher erarbeiteten Ergebnisse in eine lauffähige Programmierung für den ERS-7 umsetzten zu können. Ich denke aber, dass der hier aufgezeigte Weg eine gute Erfolgswahrscheinlichkeit besitzt und es durchaus lohnend wäre, ihn versuchsweise zu vollenden.

7 Quellenverzeichnis

[AEU04] Sony Entertainment Robot Europe: *AIBO Europa – Offiziele Webseite.* http://www.aibo-europe.com/index.asp?language=de, 2004

[Breu88] Hans Breuer, Rosemarie Breuer: *Atlas zur Physik.* Deutscher Taschenbuch Verlag, 1988

[ERS04] Sony Corporation: *OPEN-R SDK, Model Information for ERS-7*, 2004

[GT05] Humboldt Universität Berlin, Universität Bremen, Technische Universität Darmstadt, Universität Dortmund: *GermanTeam.* http://www.germanteam.org/, 2005

[GWR05] Guinnes World Records: *Fastest Men's 100 m*. http://www.guinnessworldrecords.com/gwr5/content_pages/record.asp?recordid=52043, 2005

[L2REF04] Sony Corporation: *OPEN-R SDK, Level 2 Reference Guid*, 2004

[PG04] Sony Corporation: *OPEN-R SDK, Programmer's Guide*, 2004

[RCGO04] Fraunhofer-Institut für Autonome Intelligente Systeme: *RoboCup German Open.* http://www.robocup-german-open.de/, 2004

[RCUP04] The RoboCup Federation: *RoboCup Official Site.* http://www.robocup.org/, 2004

[SDE04] Sony Corporation: *[AIBO SDE] official web site*. http://openr.aibo.com/, 2004

[WPRT04] Wikipedia: *Regelungstechnik.* http://de.wikipedia.org/wiki/Regelungstechnik, 2004

University of Hamburg
Faculty of Computer Science
Vogt-Kölln Straße 30
22527 Hamburg, Germany

WS 2004/2005

Project: RoboCups – Robot Systems in the SONY-LeggedLiga (Part 1)
Project manager: *Birgit Koch, Dietmar Möller*

Opponent Recognition and Localization

High–Level Vision Team:
Gunnar Selke (Matriculation Number 5417507)
Sonia Haiduc (Matriculation Number 5730345)

- Project Report -

Author: Sonia Haiduc

Contents

1. Introduction....3
   1.1. RoboCup, Four Legged League, German Team 3
   1.2. The Aibo Project at the University of Hamburg 3

2. Opponent Recognition and Localization - General Aspects and Algorithm....4

3. Low-Level Vision, High Level Vision and the Interface between Them....5

4. High Level Vision Processing.... 6
   4.1. Poligonization....6
      4.1.1. The Algorithm....6
      4.1.2. The Editor....8
      4.1.3. The Results....10
   4.2. Attribute Generation....10
   4.3. Classification – Decision Tree Algorithm....11
      4.3.1. The ID3 Algorithm....11
      4.3.2. Pruning – Avoiding Overfitting....13
   4.4 Predefined Data Types Used – the Tools directory....13
      4.4.1. Math/Vector2....13
      4.4.2. List....13

5. Encountered Problems 14

6. Directions for the second semester....14
   6.1. Pruning....14
   6.2. GUI for the feature generation, analysis and classification 14
   6.3. Opponent Robot Localization – the Position of the Opponent on the Field.... 15

7. References....15

1. Introduction

1.1. RoboCup, Four Legged League, GermanTeam

In 1997 the international academic initiative of building robots that could play soccer as good as and even better than humans was born. The goal was "*By 2050, develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer*". Several academic communities from all over the world took this challenge and started building and programming robots for this purpose.

In order to have a real environment for testing, observing and improving the results of the research efforts, ***RoboCup***, an international robot soccer championship, was born. At the moment, RoboCup includes several soccer leagues, adapted for the different types of soccer-playing robots. One of the leagues is the *Sony Four Legged Robot League.*

In this league, teams of four Sony Aibo dog robots play against each other on a field of 6 m x 4 m. The robots operate fully autonomously, i.e. there is no external control, neither by humans nor by computers.
In this league, the *GermanTeam* participates as a national team and it currently consists of students and researchers of four german universities: the Humboldt-Universität zu Berlin, the Universität Bremen, the Technische Universität Darmstadt, and the Universität Dortmund. The universities of the GermanTeam participate as individual teams in contests like the RoboCup German Open, but they join efforts as a national team for the international RoboCup World Cup. In order to support the parallel development of the programming, the GermanTeam adopted since 2001an architecture based on dividing the information processing and control in *modules* that could accept more *solutions.* The solutions can be switched at runtime.

1.2. The Aibo Project at the University of Hamburg

The project "*RoboCups – Robot Systems in the SONY-LeggedLiga*" exists at the University of Hamburg since 2003. In that year, a team of robots was programmed with the purpose of participating at the RoboCup contests. The base code architecture used was that of the GermanTeam and the first robot model used was the Aibo ERS-210A.

The Technical University of Hamburg - Harburg joined its efforts for developing a team of robots able to compete at an international level since the beginning of the academic year 2004. The current robot model used by the team is ERS-7.

2. Opponent Recognition and Localization - General Aspects and Algorithm

A very important issue of the robot soccer playing is the recognition and localization of the opponent robots. A robot must be able to take variable decisions depending on the different situations regarding the presence or absence and the position of the enemy robots on its visual field. In the winter semester 2004/2005 a group of students participating at the project made the identification and the localization of the opponent robots their work goal.

The basic idea was to start from a non-segmented image, an image so as the robot sees it. This image would then be segmented, searching for relevant parts, belonging to a robot's body. To prove that these parts really belong to a robot and to identify exactly what body part a segment represents, a decision tree would be used. After finding the robot's body parts, these would be stored in a 180 degrees memory, taking into account that the robots move their heads permanently from left to right, capturing totally 180 degrees. Using this memory, the distance to the robot, the direction and the orientation of the robot can be calculated. The distance to the robot is determined using the total area of the segments in the 180 degrees area and the distance between them. The direction of the robot can be calculated from the segment group's position in the 180 degrees memory and the orientation can be deduced using the relative position of the head in relation with the other body parts.

The initial algorithm was the following (Input: a camera image, not segmented):

I) Segmentation and Surface Detection

1) Find an "interesting" pixel in the image, based on its color, using color segmentation and color tables
2) When such a pixel is found, use a flood-fill algorithm to find all the other interesting pixels found in the same area as the initial pixel and to generate an "interesting" segment.
3) For this segment, the array of pixels that form the segment's contour will be generated, along with other features, as: color, bounding-box, area, etc
4) From the contour of the segment, obtain a polygon using the iterative-end algorithm

II) Attribute generation

1) For this polygon generate attributes significant for the classification : center of gravity, concave ratio, aspect ratio, area, perimeter, circle and rectangle similarity

2) Returning to I) when still possible; otherwise move to III)

III) Classification

1) For all the segments determined until now, use a trained decision tree to do a classification of the segments based on their attributes and assign them when possible to different parts of an enemy robot's body.

IV) Analysis

1) Put the segments belonging to an enemy robot in a 180 degrees memory
2) Group the segments in such a way that they could form a robot when put together.
3) For all the robots determined at IV) 2), compute the distance, direction and orientation relative to the observing robot.

However, the two teams have managed so far to obtain results only regarding the first part of the described process, the recognition of the opponent robots. The localization of the robots is planed for the second semester.

3. Low-Level Vision, High Level Vision and the Interface between Them

After analyzing the complexity of the problem, the process of recognition and localization of opponent robots was logically organized in two segments: the low-level vision processing and the high-level vision processing. The two tasks were assigned to two different teams, which communicated and exchanged results in order to complete the purposed goal.

The teams convened on an interface that would represent the basis for data exchange between them. The low-level vision team concentrated their work on the segmentation of the initial camera image, so on the first three steps of the I) part of the described algorithm. The high level team took the responsibility for implementing the rest of the algorithm.

At first, the program used structured data types. After a later refactoring operation, the last version of the program uses in most of the cases classes and objects.

The data that bounds the two work areas is represented by segments and some of their attributes. The description of the data interface is found in the Segmentation.h file, in the module provided by the low-level vision team.

```
class Segment {
   private:
        int color; //-> The segment color (red, blue, ...)
        int area; //-> The Area
        std::vector< Vector2<int> > contour; //-> The contour of the segment ( in
                the form of a vector list)
        Vector2<int> center; //-> The center of gravity
        Matrix2x2<int> bbox; //-> Bounding-Rectangle of the segment
        bool border; //-> Shows if the segment intersects the border of the camera
                image
        float center_angle; //-> The angle of the center
   public: ...
}
```

The segments provided by the low-level vision team are potential body parts belonging to an opponent robot. The high level vision has the task of analyzing and classifying them into different robot body parts, based on the information received from the low-level vision and other generated segment attributes.

4. High Level Vision Processing

4.1. Poligonization

The first task of the High- Level Vision Team was to generate a polygon from each set of segment contour points obtained from the Low-Level Vision Team. This was necessary for obtaining supplementary attributes of the provided segments, which will be used for the classification process.

4.1.1. The Algorithm

The algorithm used for implementing the polygonization task was Iterative End-Point Fit algorithm. This is described as follows.

Given a set of n points, we want to obtain a polygon that would reproduce at best the shape described by these points.

1) Find the two most distant points and unite them with a line. (In the example the two points are A and B)
2) Compute the distances from each point to this line.
3) If all distances are less then a predefined threshold, the polygon generation is finished.

4) If not, find the furthest point from the AB line (here C) and break the initial AB line in two new lines, AC and CB.
5) Repeat the process starting from 2) for the two new lines ("*divide et impera*") and continue until the algorithm exits at 3). The set of lines obtained will from the desired polygon.

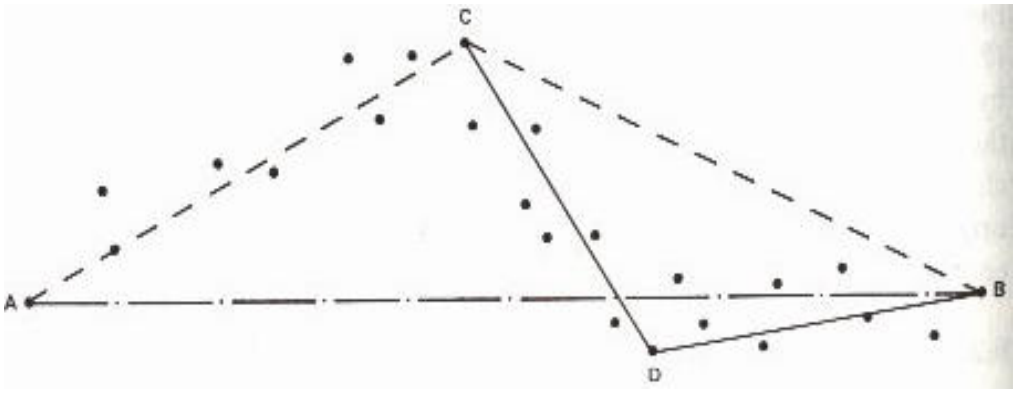

Fig. 1 - The Iterative End-Point Fit Algorithm

The algorithm has also a drawback: it can be influenced by isolated singular points. Also, a major factor that influences the algorithm's precision is the chosen threshold.

The files used for polygonization:

1) ***Polygon.h + Polygon.cpp*** - describe the Polygon data type and some useful functions in the polygonization process

The *Polygon* contains the array of the end-vertices of the lines that form it and their number. Some of the methods belonging to this class implement operations for: reversing the orientation of the polygon, inserting a vertex at a given position in the vertex array, appending a vertex, erasing a vertex, testing if the polygon is concave at a given vertex, computing the angle at a given vertex, testing if a vertex is part of a polygon, computing the area, perimeter and center of the polygon.
The V*ertex* data type defined also here is a synonym for a Vector2 (2D point) with integer coordinates. It is the data type of the vertices that form a Polygon. Some helpful functions used in the Polygon class' methods, or later for the polygonization are also defined here:

- float *SquaredPoint2LineDistance*(const Vertex &A, const Vertex &B, const, Vertex &C) --> returns the squared distance from point C to line AB

- float Point2LineDistance(const Vertex &A, const Vertex &B, const Vertex &C) --> returns the distance from point C to line AB

- bool IsConcave(const Vertex &A, const Vertex &B, const Vertex &C) --> verifies if the angle formed by the three vertices, with the center in B, is concave

- float Angle(const Vertex &A, const Vertex &B, const Vertex &C) --> returns the angle formed by the three vertices, with the center in B

2) *Polygonizer.h + Polygonizer.cpp – implement the poligonization algorithm*

The main polygonization function has the signature:

void Polygonize(Polygon *P, Vertex *ps, int nps);

, where *P* is the resulted Polygon, *ps* is the array of contour points of the segment and *ns* is the number of contour points situated in *ps*.

The *Polygonize* function uses the secondary function:

static void SplitSegment(Vertex *ps, int nps, Segment *segm),

designed to split a line in two in the polygonizer algorithm (divide et impera). The *Segment* data type is a structure defined also in *Polygonizer.cpp* and it represents a line between two vertices situated in an array (the contour). The line is also part of a list of lines that will later define a polygon.
In the definition of the *Segment* class the order number of the end-vertices in the vertices array and references to the previous and the next line in the line sequence are included.

4.1.2. The Editor

In order to be able to choose the appropriate threshold and to test our program for the poligonization process without needing to wait for the segments provided by the low-level vision team, we developed a special segment editor. With the help of the editor, we could draw segments and obtain the segment's

contour in the form of a pixel array. The set of contour pixels represented the input data for the tests.

For implementing the editor, we used the free multimedia library SDL. Because we wanted to concentrate our efforts on developing our program further and not on implementing functions using the SDL library, we used some already implemented free functions for drawing lines found on the internet. These functions are declared in the sdl-draw.h and their definition is found in the sdl-draw.cpp file. The main program for the editor is located in the file editor.cpp.

After launching the editor.exe program, the user has the possibility of manually drawing segments pixel per pixel. The drawing starts from a predefined pixel and it flows continuously until the end of the program. The user can choose the direction of the line, using the set of arrow keys situated on the right side of the keyboard (NUM-arrows).

Fig. 2 - A manually drawn segment

For saving the contour of the drawn segment, the user must press CTRL+S. The contour will be saved in a Vector array and then written in the "contour.txt" file, each Vector (pixel) on one line. If the user exits the program without saving, the contour will not be saved.

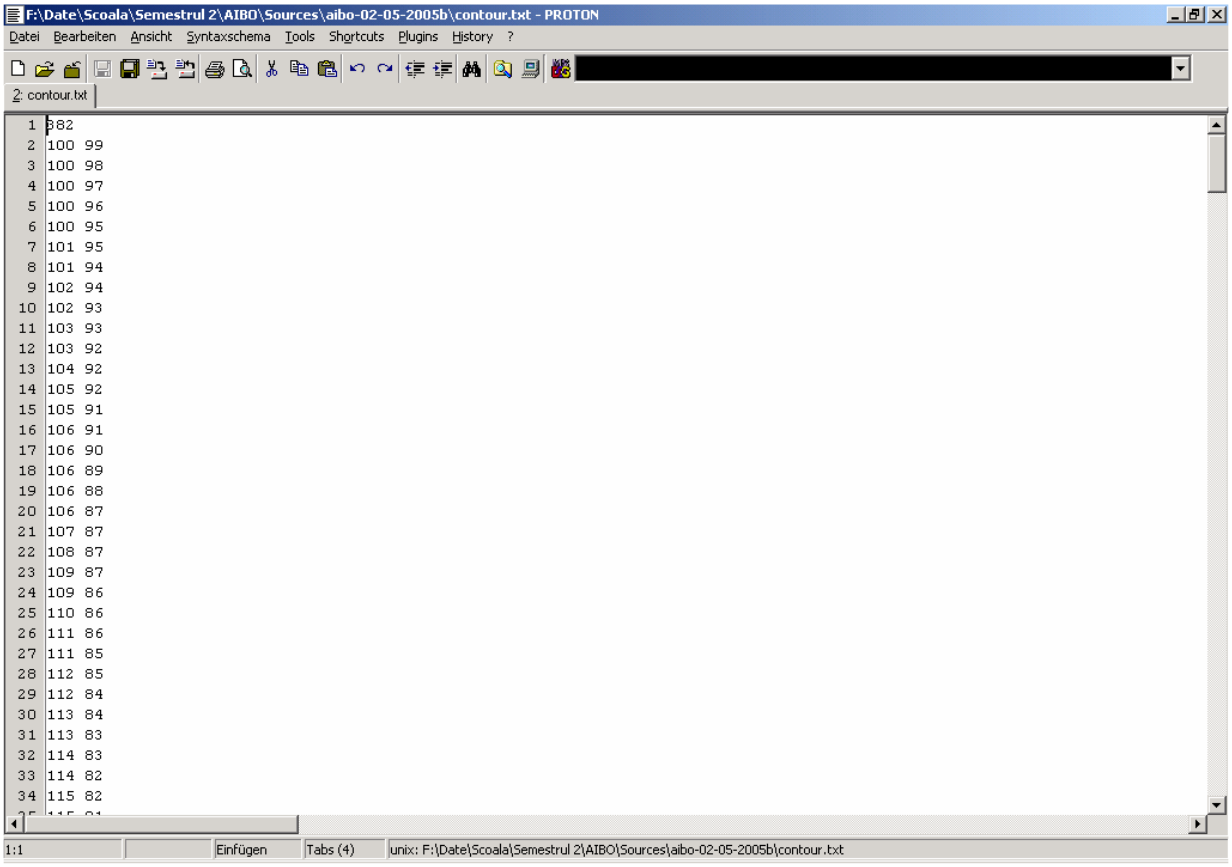

Fig. 3 – Part of the contour.txt file obtained for the drawn segment

We used the "contour.txt" file as input for tests on the polygonization algorithm and after testing and observing the results for different thresholds, we convened to setting the threshold to 3.

4.1.3. The Results

With relatively small polygons with 4-6 edges, our algorithm managed to process ca. 500.000 polygons per second on a 2.4Ghz Pentium. We approximate that on an Aibo ca. 40.000 polygons will be computed in the same amount of time.

4.2. Attribute Generation

After obtaining the polygons from the segments, the next step in the recognition algorithm is the attribute extraction. We analyzed different attributes in the purpose of finding the most relevant ones to use for the classification step. The

chosen attributes of a polygon were:

- concave ratio (= number of concave angles)
- area / perimeter^2
- circle similarity
- rectangle similarity
- aspect ratio (= width / height of the bounding rectangle)
- color

We defined a special structure that contains the attributes and will also contain the decision after the classification process. We named this structure *Example*. Our classification process will use only this type of objects and not polygons. We decided to implement it this way because, for the recognition process, it is not needed to store the polygons. We need only the polygon's attributes in order to be able to classify them as an enemy robot body part or not.

The possible decisions were stored in an enumeration data type. The decisions we considered were:

DEC_NONE - the segment is not an enemy robot body part
DEC_HEAD - the segment represents the head of an enemy robot
DEC_SIZE - the segment is a size part of an enemy robot
DEC_LEG - the segment represents a leg of an enemy robot

All these data types are defined in the file *Example.h.* In *Example.cpp* a function is defined that will extract the attributes from a given Polygon and put them into an Example: *GenerateAttributes(Polygon &P, Example &E)*.

4.3. Classification – Decision Tree Algorithm

4.3.1. The ID3 Algorithm

The classification process is made by using decision tree learning. We used the ID3 algorithm to train the tree, because it was the only good and accessible algorithm we found.
We built a database with examples that contain different values of the attributes and the decisions made based on these attributes. These examples will be used to train the decision tree.
The database is defined in the D*atabase.h* and *Database.cpp* files.
In the decision tree, the nodes will be the attributes chosen by us to do the categorization, the edges starting from a node will be the values that this attribute can take and the leaves will always be decisions.
The most important issue to build a decision tree was to decide what attribute will be tested at which level of the tree. The most relevant attribute must be placed as the root of the tree. To train a decision tree means, in fact, determining the place of the nodes in that tree (the order of the attributes to be tested and the distribution of the decisions in the leaves of the tree).

To decide the order of the attributes in the decision tree (which are more relevant), the ID3 algorithm uses two measures: *Information Gain* and *Entropy*.
The Entropy calculates the purity of a set of examples and the Information Gain measures the quantity on valuable information an attribute gives based on a set of examples.
Given a set S of examples, a categorization C in n categories (decisions) and p_i, i=1,n the proportions of the given categories in the total set of examples, the definition of S's entropy is the following:

$$Entropy(S) = \sum -p_i \log_2(p_i) , i=1,n$$

The Information Gain for an attribute A based on the set S of given examples is:

$$Gain (S,A) = Entropy(S) - \sum (|S_v| / |S|) Entropy(S_v) ,$$

where $v \in Values(A)$ and S_v is the set of examples where A has the v value.

The *Database* class contains, among other, methods implemented to calculate the Information Gain and the Entropy based on a set of examples.

The general description of the ID3 algorithm is:

Given a set of examples, S, categorized in categories c_i, then:
1. Choose the root node to be the attribute, A, which scores the highest for information gain relative to S.
2. For each value v that A can possibly take, draw a branch from the node.
3. For each branch from A corresponding to value v, calculate S_v. Then:
   - If S_v is empty, choose the category $c_{default}$ which contains the most examples from S, and put this as the leaf node category which ends that branch.
   - If S_v contains only examples from a category c, then put c as the leaf node category which ends that branch.
   - Otherwise, remove A from the set of attributes which can be put into nodes. Then put a new node in the decision tree, where the new attribute being tested in the node is the one which scores highest for information gain relative to S_v. This new node starts the cycle again (from 2), with S replaced by S_v in the calculations and the tree gets built iteratively like this.

The algorithm terminates either when all the attributes have been exhausted, or the decision tree perfectly classifies the examples.
The declaration of the DecisionTree class is found in the *DecisionTree.h* file. The ID3 Algorithm is described in the *DecisionTree.cpp* file. The *TrainID3* method implements the ID3 algorithm and the *Consult* method is to be used after the training, for categorizing an example. The *Train ID3* function will be run on a

computer, to prepare the tree for the future categorization process and the *Consult* method will be used by the Aibo for the recognition process.

4.3.2. Pruning – Avoiding Overfitting

Overfitting is a common problem in machine learning. Decision trees suffer from this, because they are trained to stop when they have perfectly classified all the training data. That means that each branch is extended as far as it takes to correctly categorize the examples relevant to that branch. This results in a problem when the branches become too large.

Several methods for avoiding overfitting are accessible at the moment. Some stop the growing of the tree before it reaches perfection and some let the tree grow and apply then a post-pruning operation, eliminating some unnecessary branches. The second approach has been found to be more successful in practice, but both methods are concerned about the same matter: finding the correct tree size.

We chose the second method: pruning. For this purpose we started implementing *Pruning.h* and *Pruning.cpp,* but the pruning algorithm is not yet completely implemented.

4.4 Predefined Data Types Used – the Tools directory

4.4.1. Math/Vector2

This class represents a generic 2-vector. The generic factor is here the type of the vector's arguments.

template <class V> class Vector2

The methods belonging to this class implement the addition, multiplication, subtraction, division, negation, comparison, transposition, normalization, angle and length calculation of vectors.

4.4.2. List

This class implements double linked lists for arbitrary data types. It uses a template for generality. It has a reference to the first and the last element as well as the number of elements in the list.

template <class T> class List;

Some of the functionalities the class' methods offer are: list concatenation, insertion and deletion of elements, reading and writing a list from a stream.

An iterator is also defined for this list: class Pos. The iterator contains a reference to the current list entry and implements a wide range of iteration functions.

5. Encountered Problems

In our activity during the first semester, we encountered some problems, but none were significant. The problems we documented are:

- C – Problems: Buffer Overflows, Memory Leaks
- The definition of rectangle and circle similarity in the attribute extraction process
- Finding documentation for pruning decision trees

6. Directions for the second semester

6.1 Pruning

For the second semester we have at first in plan finishing and testing the pruning algorithm. We already implemented some functions, but the main algorithm still needs to be coded.

6.2. GUI for the feature generation, analysis and classification

We decided to build a user interface that will integrate the functionalities implemented until now, including the manual classification of segments for the decision tree learning. A user should be able to choose from a list of available segmented images and then, in each image to choose a specific segment. For each segment will then be generated the chosen attributes. The attributes will be shown on the screen, in a section of the GUI. The user will have then the possibility of selecting the class (the decision) of the segment.

After the classification, the user will have the choice of saving the results in a database used then for the decision tree learing.

6.3. Opponent Robot Localization – the Position of the Opponent on the Field

For the second semester we have also in plan continuing the algorithm described in the first part of the paper with the part concerning the opponent localization.

7.References

1) http://www.informatik.uni-bremen.de/kogrob/papers/rc05-objectrecognition.pdf

2) http://www.doc.ic.ac.uk/~sgc/teaching/v231/lecture11.html

3) R.O.Duda, P.E.Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons Press, New York, 1973

4) http://www.informatik.uni-hamburg.de/TIS/index.php?content=robocup/index.htm/

5) http://www.robocup.org/

University of Hamburg
Faculty of Computer Science
Vogt-Kölln Straße 30
22527 Hamburg, Germany

SS 2004/2005

Project: RoboCups – Robot Systems in the SONY-LeggedLiga (Part 2)
Project coordinator: ***Birgit Koch, Dietmar Möller***

Opponent Recognition and Localization

High–Level Vision Team:
Gunnar Selke (Matriculation Number 5417507)
Sonia Haiduc (Matriculation Number 5730345)

- Project Report Part 2 -

Author: Sonia Haiduc

Contents

1. The initial plans for the second semester..... 4
1.1 Pruning..... 4
1.2 GUI for the feature generation, analysis and classification 4
1.3 Opponent Robot Localization..... 4

2. The actual directions followed this semester 5
2.1 The GUI Program 5
2.2 The Refactoring..... 8
2.3 The Attributes' Redefinition 11

3. Future Directions 12
3.1 Testing – the need for test data..... 13
3.2 Automatic extraction of attributes and building a decision tree embedded in a program. 13
3.3 Pruning..... 13
3.4 Attribute definition completion 13
3.5 Opponent Robot Localization..... 13
3.6 Possible improvements and extensions 13
3.7 Integrate our code in the Aibo-Code 14

4. RoboCup Competitions attended by the Hamburg Dog Bots in 2005 14
4.1 RoboCup German Open 2005 in Paderborn 14
4.2 RoboCup 2005 in Osaka 14

5. References 15

1. The initial plans for the second semester

At the end of the first semester we analyzed our work and proposed ourselves the following directions for the second part of the project:

1.1 Pruning

Unless pruned, the decision tree will try to categorize the data perfectly. As the tree is trained based on a set of examples, it will try to find the perfect match from this set of examples when categorizing a new item. This would practically make the decision tree useless, because it would be able to categorize only the set of examples used to train it.
We had in plan for the second semester to complete the set of pruning functions defined in the *Pruning.h* and *Pruning.cpp* files and to test their functionality.

1.2 GUI for the feature generation, analysis and classification

At the end of the first semester we decided as future direction to implement a GUI program that would integrate the existing functionalities. This would offer automatic attributes generation for a selected segment and the ability of manually classifying example - segments, used then to train the decision tree. The segmented images would be loaded in a list and the user would be able to choose an image from this list. The chosen segmented image would then be displayed in the program's interface and the user would have the possibility of selecting a specific segment. For the chosen segment, the attributes would be automatically generated and displayed on the screen. The user would then be able to choose a category from the ones available at the interface level and eventually save the categorization.

1.3 Opponent Robot Localization

In the first semester we approached only an issue of the initial proposed algorithm, the recognition of the opponent robots. The second part of the algorithm referred to also localize the recognized robots. We aimed at implementing the localization process during the second semester.

2. The actual directions followed this semester

The plans made at the end of the first semester were implemented only in part during the second semester, due to different approaches of the already implemented features, problems that interfered on the way, need for test data and refactoring. The pruning process and the opponent robot localization were not further implemented, due to the lack of time. The GUI program was implemented and a great part of our activity this semester concentrated around its implementation. The automatic generation of a segment's attributes was not included in the GUI program, as planed, but we included the segmentation process in the editor. The images that are now loaded don't have to be segmented anymore, because the segmentation is made when choosing a pixel in the image.
The other main part of our activity concentrated on the needed refactoring and redesigning procedure that applied to our code. Some main data structures were redefined and new ones were introduced. The attributes of a segment were also analyzed, reorganized and some reimplemented.
In the following sections each part of our activity in this semester will be analyzed in detail.

2.1 The GUI Program

We have developed an editor which will be used for the manual classification of given polygons as parts of a robot's body. The classified polygons will then be useful for training a decision tree into classifying on its own new, unknown polygons.
The manual classification process starts with loading images from a specified input directory. All the images in the selected directory will be listed on the left in the editor window and the user will be able then to select one image at the time for the classification process. The selected image will be displayed in the center of the editor window. The user will now be able to click on a spot in the image. When a pixel is clicked, a segmentation procedure, based on a Flood-Fill algorithm (**[4], [5]**) is started, to determine the segment in which the pixel is situated (the maximum set of pixels of the same color or with very little color difference to the clicked pixel). After finding all the pixels belonging to the segment, a Square Trace algorithm (**[6]**) is performed to find the array of pixels which define the contour of the segment. Then, starting from the contour, the Iterative End-Point Fit algorithm (**[2]**) defines a polygon which describes the segment's shape.
The determined polygon is highlighted on the screen by a white border. The user can now choose to classify the highlighted polygon by clicking on one of the buttons found on the right side of the editor window, representing the possible

recognizable parts of a robot's body: *Head*, *Side*, *Leg*, *Front and Back*. When classified, a polygon changes its border color into pink and the number found in parentheses on the chosen classifier button increases with one. Also, the user can choose another segment, without classifying the current one, in which case, the current polygon will be de-highlighted in favor of the new selected one.

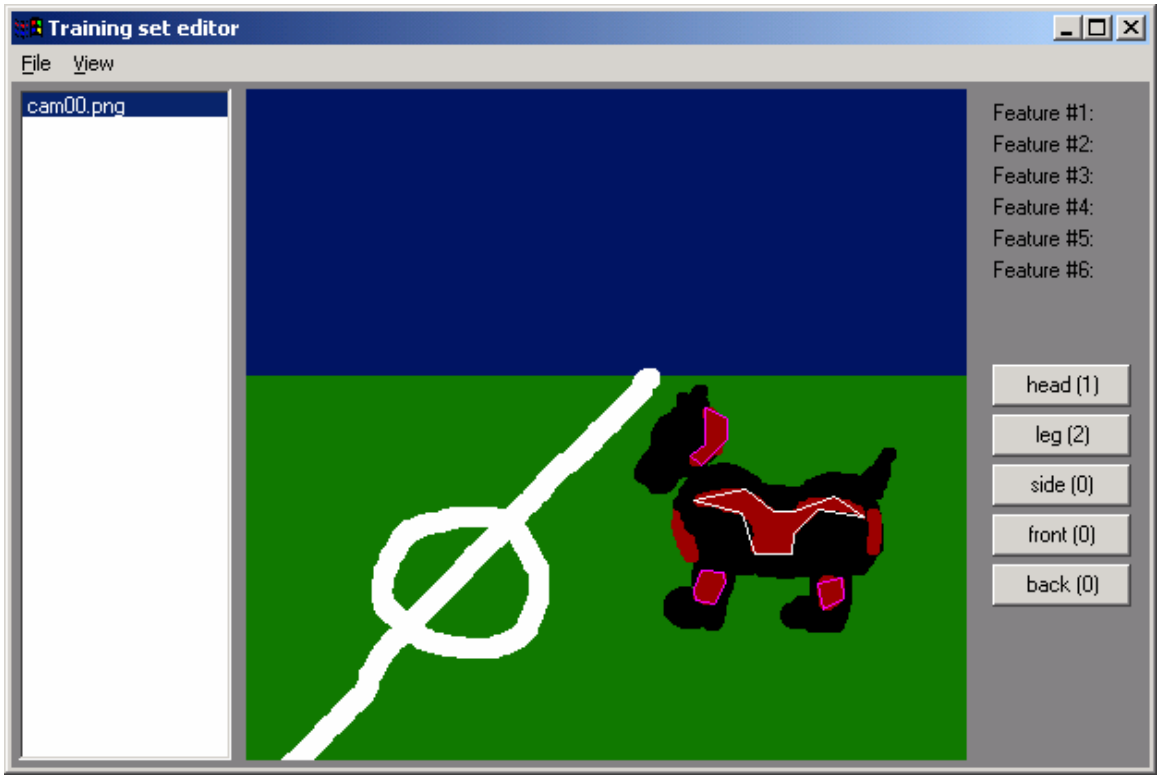

Figure 1 The Classification Editor

Once a classification is made, it can't be rolled back. The user can classify the same polygon several times, but every time the polygon will be seen as a different one.
In the previous version of our code, described in **[1]**, the classification data was stored in an intern database data structure, defined only for the use within our program. In the second semester, for the sake of reusability, we adopted another way of storing the classification data. After classifying the desired polygons, from one or more images, the user can choose to save the classification results into a database file, which can be later used by any training program. The files will store the vertices of the polygon and its determined class. The user can choose the name and the location of the file.
The database file is a text file with a special format, as seen in Figure 2. The file stores on the first row the dimension of the database, by that meaning the

number of classified polygons contained in the database. On the next rows, there are records containing the classified polygons, each one preceded by the word "classification". In each record, the name of the recognized robot part is written as it appears on the button used for the classification. Then, the polygon is represented by its size and its vertices. The vertices are written in the format (x,y), where x and y are the two coordinates of a 2D point.

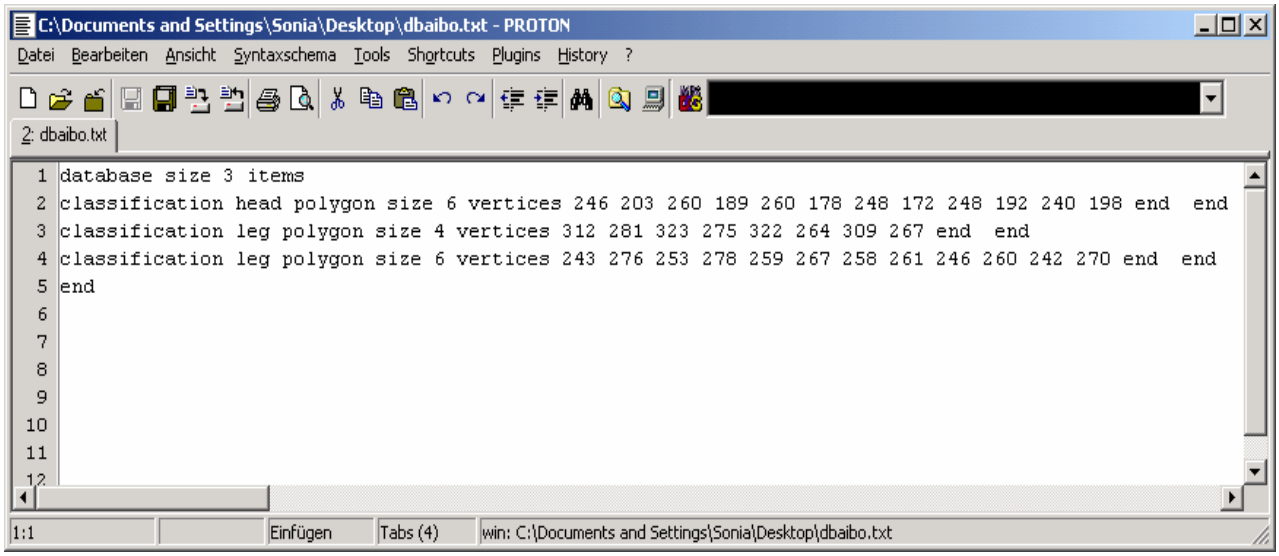

Figure 2 The database file, containing the classification data.

The Editor's menu offers the following options:

a) **File**: general options regarding loading and saving data

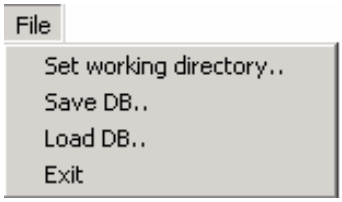

Figure 3 The File menu option in the Editor's menu

The suboptions:

a.1) Set working directory: gives the user the liberty of choosing the directory where the segmented images are to be found

a.2) SaveDB: saves the classification data generated in the editor's window into a database (a text file with a specific format – see Figure 2)

a.3) LoadDB: loads the classification data from an existing database (a text file in a specific format – see Figure 2). This way, the user can add new information to an existing database, without creating a new one.
a.4) Exit: exits the editor program

b) **View**: display options

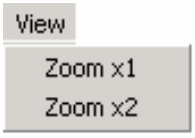

Figure 4 The View menu option in the Editor's menu

The suboptions:
b.1) Zoom x 1: reproduces the selected image at its normal size
b.2) Zoom x 2: magnifies by a factor of 2 and displays the selected image. This option is particularly useful for images taken when the robot is situated at a great distance and there is the need to see some details in the image.

2.2 The Refactoring

At the beginning of the second semester we analyzed our source code and decided to perform a refactoring procedure, due to the need for generalization, reuse and organization. There are some major changes in our code from last semester, which reflect on two levels: the organization and the content of the data. From the organizational point of view, our code was subject to significant changes. Some new directories were defined, the location of some files was changed, new files were introduced and some files removed. Major changes took place also in the implementation of the data types. Among other changes, some structured data types were redefined as classes and encapsulated. The directories currently containing our code will be approached one at a time as it follows, with the purpose of identifying the changed parts.

a) The main directory
The structure of the main directory changed significantly. The data is now organized in subdirectories, based on the functionality it implements. There are currently four directories that contain source code files: attic, Tools, learning and wx. There are also some files that are stored in the main directory. These files are platform independent and are requested for all the future programs based on our code. The files in the main directory are:

- *Classification.h* and *Classification.cpp*
  These files date from the second semester and implement the "Classification" class. The class has two members, a Class element, which reimplements the structured data type "Decision" defined last semester inside the *Example.h* file, and a Polygon element. The Class member defines, as "Decision" defined before, the class assigned to a polygon in the classification procedure. The possible classes are associated with the robot parts considered to be recognizable and are: head, side, leg, front and back (of the robot).
  The files also implement the streaming operators used to write the classification data collected from the GUI Editor into a file. The data will be written in the form shown in the example: *classification back polygon size 4 vertices 336 250 343 245 343 230 336 228 end end* , where *back* represents the class or the robot's body part and the vertices of the polygon are written in the format (x,y), where x and y are the two coordinates of a 2D point.

- *Database.h*
  In this file a declaration of a database is given, containing generic elements, defined using a template type T. Operators for adding, deleting, searching elements in the database are defined, along with streaming I/O operators. The streaming operators give the opportunity to write a database to a file in the format shown in Figure 2 or to read a database from a file written in this format.
  This generic database will be used in our code for defining a database containing classification data, but its generic definition makes it easy also for other programs to use it.

- *Polygon.h* and *Polygon.cpp*
  These files contain the representation of a polygon and remained almost the same as last semester. The only new features implemented are the streaming operators which allow a polygon to be written to and read from a file. Examples of writing a polygon to a file are seen in Figure 2.

- *Polygonizer.h* and *Polygonizer.cpp*
  The files contain an algorithm (Iterative-End-Point Fitting algorithm) that approximates polygons starting from the contour of a geometric form, represented by an array of vertices. The files date from last semester and remained unchanged.

`b) attic`

The *attic* directory dates from the last semester and has remained the same. It contains unused files that need to be reviewed and modified or deleted. In this

directory are located the segmentation files (the interface between the low level and the high level team) and the files meant to implement the interface between our program and the GermanTeam code.

c) Tools

The *Tools* directory remained unchanged. It contains supplementary data types used in our code, like vectors.

d) learning

This is a new directory that contains all the files related to the decision tree and the learning mechanism.

- *DecisionTree.h* and *DecisionTree.cpp*
  Here is implemented the Decision Tree structure and the operations related to it, including training and consulting the tree. The files have remained in essence the same from last semester. The only changes made are related to the data types and refer to modifying the names to the new defined ones.

- *Features.h* and *Features.cpp*
  In the first semester, in the *Example.h* file, the definition of the structured data type "Attribute" was given. This data type was the representation of a polygon's attribute considered to be important for the classification process. The data type contained the attribute's id (like ASPECT_RATIO) and the maximum integer value it could be assigned (range) in its discrete definition. Also in *Example.h*, an array of Attribute-s was defined, called Attrib. It would contain all the attributes considered for the classification of a polygon.
  In the second semester, we reconsidered this approach and redefined the mentioned data types in the object-oriented manner. This is done in the two above mentioned files, by the class "Features". It contains all the declarations and ranges for the considered attributes (features) but also an array containing a set of integer values attributed to the features. A Polygon has, in consequence, a correspondent Features object. The class "Features" contains even an operator for the extraction of the attributes' values for a given Polygon. It has also, like Polygon and Database, a set of streaming operators which enable reading and writing the attribute values array from and to a file.

- *Prototype.h and Prototype.cpp*
  The class defined in the two files, "Prototype", is the object-oriented successor of the structured data type Example, defined last semester in the *Example.h* file. It has two members: a Classification (Polygon + Class)

element and a Feature element. The Prototypes are the examples used for building or training the decision tree. In the resulted decision tree the attributes (features) will be stored in the non-leaf nodes, the values they take in the edges and the classes in the leaf nodes.
Like other data types mentioned, the Prototype class also implements streaming operators for I/O operations using files.

- *ProtoDb.h* and *PtotoDB.cpp*
  The "ProtoDB" class is defined in these two files. It is actually the old Database class defined in the first semester, but for reasons regarding generalization and reuse, a general database was build, in the Database.h file dated in this semester, which is used later for implementing databases with custom element types. So is the case of "ProtoDB", which represents a database with Prototype elements and it is based on the template database defined in *Database.h*.
  "ProtoDB" has also streaming operators, like many of the classes mentioned above.

e) wx

This new directory contains all the files that implement the graphical classification program. Besides the files found in this directory, we also use some functions included in a predefined graphical library, wx. The source code and the documentation for this library can be found at **[7]**.

2.3 The Attributes' Redefinition

After a careful analysis of the attributes' implementation, we decided some of the attributes were not convenient to define in a discrete form and needed to be replaced by others or redefined. For example, Circle Similarity and Rectangle Similarity are very hard to define in a discrete form and require a significant amount of test data and time to do this correctly.
The current set of chosen attributes, defined in the *Feature.h* and *Feature.cpp* files, is: *angle convexity, aspect ratio*, *number of vertices* and *compactness.*

Angle Convexity is related to the concave ratio attribute defined in the previous code version. The concave ratio was defined last semester as the number of the polygon's concave angles. We decided to redefine this attribute, so that it would offer a higher information gain. In this purpose, we took into consideration also the number of convex angles and defined *angle convexity* as the subtraction of the number of concave angles from the number of convex angles.

As this attribute must be, like all other attributes, defined in a discrete form, and the range that we chose for it was 3, we considered the following definition for *angle convexity:*

- if nr convex angles – nr concave angles = 1 or 2, ***angle convexity* = 1;**
- if nr convex angles – nr concave angles = 3 or 4, ***angle convexity* = 2;**
- forl other cases, ***angle convexity* = 3.**

Another polygon's attribute considered was *Aspect Ratio*. The attribute was chosen last semester also and it refers to the width/height ratio. The discrete definition was given this semester and is:

- if 1,5 * height <= width, *aspect ratio* = 0;
- if height >= 1,5 width, *aspect ratio* = 1;
- in all other cases (height ~ width), *aspect ratio* = 2.

The *Number of Vertices* a polygon has is also one of the attributes considered. The attribute was chosen only this semester, after a close analysis of the attribute candidates. The current discrete definition for this attribute is:

- if the number of the polygon's vertices is <= 3 or >=6, *number of vertices = 0;*
- if the number of the polygon's vertices is 5, *number of vertices = 1;*
- in all the other cases, *number of vertices = 2.*

The other attribute considered, *Compactness*, refers to the area/radius^2 ratio. It doesn't have a discrete definition yet, as attentive analysis is still required.

In the process of choosing the adequate attributes, we analyzed ourselves the possible attribute candidates, but consulted also **[2]**. We are considering adding new attributes to be evaluated in the classification process. Choosing the attributes requires, although, a systematic analysis and especially testing. In consequence, we need a significant amount of data for testing the information gain other attributes could give, but also the discrete definitions given until now.

3. Future Directions

Although most of our code for the recognition of the opponent robots is implemented, there are still some parts that need development or testing. The future directions meant to complete the code we created until now are:

3.1 Testing – the need for test data

For testing our algorithms and the information gain the chosen attributes and their discrete definitions give, we need a considerable amount of test data, by which meaning images taken from the Aibo camera.

3.2 Automatic extraction of attributes and building a decision tree embedded in a program.

We plan on implementing a program that extracts automatically the defined attributes from a polygon and using the extracted attributes, it builds (learns) a decision tree so that it would be able to recognize any other polygon as belonging to one of the defined classes.

3.3 Pruning

The pruning procedure still remains a goal for the future. Pruning the decision tree is a necessary operation for generalizing the recognition process. Some parts needed for implementing pruning for a decision tree are already implemented, but the final version is not yet ready.

3.4 Attribute definition completion

We are thinking of defining new attributes to be considered in the classification process, as the more attributes are considered, the more precise and general the recognition is. Of course, this complicates our job of defining the attributes in a discrete form and needs systematic testing. Also, the Compactness feature still needs to be defined in its discrete form.

3.5 Opponent Robot Localization

As said before, the next step in realizing the goals we set ourselves in the first semester of the academic year 2004/2005, after recognizing an opponent robot, is the robot's localization. After coding to the end the recognition, the localization will be approached.

3.6 Possible improvements and extensions

After the essential components of our code will be defined and tested, an attentive analysis is planned, with the purpose of identifying parts of code that could be improved for minimizing the execution time. This is based on the fact that the processor of 400 Mhz an Aibo has imposes visible limitations and every second in the execution time matters for the robot's reaction on the field.

3.7 Integrate our code in the Aibo-Code

Our final goal is, of course, the integration of the code we created in the Aibo code. We still need to analyze where and how it will be integrated.

4. RoboCup Competitions attended by the Hamburg Dog Bots in 2005

In the academic year 2004/2005 the University of Hamburg's team of Aibos, Hamburg Dog Bots, participated at two competitions of international rank: the fifth edition of the RoboCup German Open in Paderborn, Germany and the RoboCup in Osaka, Japan.

4.1 RoboCup German Open 2005 in Paderborn

The first competition, the RoboCup German Open, was held between the 8th and the 10th of April 2005 at the Heinz Nixdorf MuseumsForum in Paderborn, Germany. There were a total of 9 Aibo teams that participated at the competition, coming from different Universities in Germany as well as Holland, Italy, France and Spain. The Hamburg Dog Bots managed to reach the quarter finals, winning against the Team Chaos from Spain, but lost in the quarter finals in favor of the Aibo Team Humbold, from the Humbold University in Berlin. On the other hand, the Hamburg Dog Bots won one of the challenge competitions, the Variable Lighting Challenge, where the light settings are changed every 20 seconds and the robot must be able to score as many goals as possible in spite of these changes. (**[10]**)

4.2 RoboCup 2005 in Osaka

The RoboCup 2005 took place between the 13th and the 19th July 2005 in Osaka, Japan. The Four Legged League hosted 24 Teams from all over the world. The

Hamburg Dog Bots, who were present at the competition, had some problems playing due to the use of a borrowed robot, after one of their Aibos encountered technical problems. This and other problems stopped the Hamburg Dog Bots before reaching the quarter finals. As far as it concerns the challenge competitions, the Hamburg Dog Bots reached the 9th place at the Open Challenge competition, with their automatic Commentator System, that should replace the referee. They also reached the 4th place at the Almost Slam Challenge, where the robot must build itself a map of the field in 1 minute and then using this map it must find 5 specific points on the field. At the Variable Light Challenge, the team didn't get the expected result, but managed to reach the fourth place in the general challenges classification and qualified for the World Cup 2006 in Bremen, Germany.

5. References

[1] Sonia Haiduc, Project Report "*Opponent Recognition and Localization*" Part 1.

[2] Bernd Neumann, Computer Vision Course at the University of Hamburg, http://kogs-www.informatik.uni-hamburg.de/~neumann/BV-SS-2005/#Folien

[3] R.O.Duda, P.E.Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons Press, New York, 1973

[4] James D. Foley et. al., *Introduction to Computer Graphics*, Addison-Wesley Publishing Company, 1994.

[5] Michael Abrash, *Graphics Programming Black Book*, Coriolis Group Books, 1997.

[6] T. Pavlidis, ***Algorithms for Graphics and Image Processing***, Computer Science Press, Rockville, Maryland, 1982

[7] wx library, http://wxhaskell.sourceforge.net/

[8] http://www.robocup.org

[9] http://www.ais.fraunhofer.de/GO/2005/

[10] http://www.informatik.uni-hamburg.de/TIS/index.php?content=robocup /index.htm/

[11] http://hamburg-dogbots.blogspot.com/

[12] http://www.informatik.uni-hamburg.de/TIS/osaka_blog/index.php

[13] http://blog.yumdap.net/

[14] http://www.tzi.de/4legged/bin/view/Website/Results2005