Tigers Mannheim

(Team Interacting and Game Evolving Robots)

Team Description for RoboCup 2012

Malte Mauelshagen, Daniel Waigand, Christian Koenig, Steinbrecher Oliver, Georg Leuschel, Nico Scherer, Manuel Schroeer,Frieder Berthold, Dion Hornig, Marian Franke, Marius Barthel, Britta Weber

Department of Information Technology, Department of Mechanical Engineering
Baden-Wuerttemberg Cooperative State University,
Coblitzallee 1-9, 68163 Mannheim, Germany
management@tigers-mannheim.de
http://www.tigers-mannheim.de

Abstract. This paper presents a brief technical overview of the main systems of Tigers Mannheim, a Small Size League (SSL) Team intending to participate in RoboCup 2012 in Mexico City. First there is a description of our hardware system followed by our software modules. Furthermore an outlook displays the upcoming goals of our team.

1 Introduction

Tigers (Team Interacting and Game Evolving Robots) Mannheim is a team of students of the Cooperative State University Baden-Wuerttemberg Mannheim. Last year we had a good start at the Robocup 2011 in Istanbul and were able to proof the robustness of our system.
Due to the fact that we decided to publish all our available source code and documentation of our system after RoboCup 2012, this paper should give an overview of our system only. Everyone who is interested in special parts of our soft- or hardware may freely download it from our website after the tournament. We believe that the most benefit is given to the community when all parts of our system can be accessed by everyone who is interested in.
This paper is divided into three sections. New changes of the hardware are explained first. Next, an overview of our main control software is given while a few parts will be described more detailed. At last there is an outlook on what we will be able to finish until RoboCup 2012 and also on some long term goals for the next years.

2 Hardware

2.1 Mechanical System

Drive The omnidirectional drive of the robot consists of four wheels with twenty transverse rollers (assembly of aluminum rim and a nitrile rubber tyre) that are driven by one Maxon 30W EC-drive each. Torque is converted by steel gearwheels with a gear ratio of about 3.33. Thus the drive is optimized for high acceleration (top speed is reached in less than one second even if acceleration starts while the robot stands still) and little residual heat build-up inside the motors. Nevertheless, the robot achieves a top speed of approximately 3 m/s when driving forward.

Kicking Device Due to the ambition of continuous improvement we develop a new kicking device which works with a crank gear. An electro motor starts the crank. A crankshaft relays the kick to the ram. This ram is guided by tracks whereby a certain movement is assured.

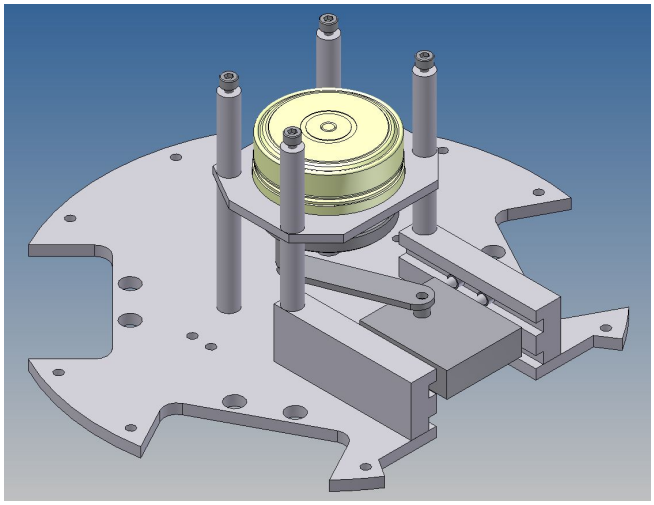

Fig. 1. New kicker approach

The new construction is supposed to be completed in 2012 and might be assembled in the bots for the Robocup in Mexico City (2012) when tests have shown the robustness of the new approach.

Case The outer case of the robot is made of a combination of metal panels and leather. This elastic material assures even with high impacts no damage in the housing and is still able to save the electronic parts inside the bot. Leather is a completely new idea and a good opportunity as a substitute for all kinds of plastic materials.

- **External diameter:**180,00 mm
- **Height:** 148,00 mm
- **Maximum ball coverage:** 15,29

3 Software

3.1 Overview

An important decision we made at the beginning of our project was to write all software, not running on the robot, using the JAVA programming language. Due to the simple structure and its compatibility, it is easier to work with within a big development team. At the moment there are no major performance losses or other disadvantages compared to C++, referred to the SSL environment.

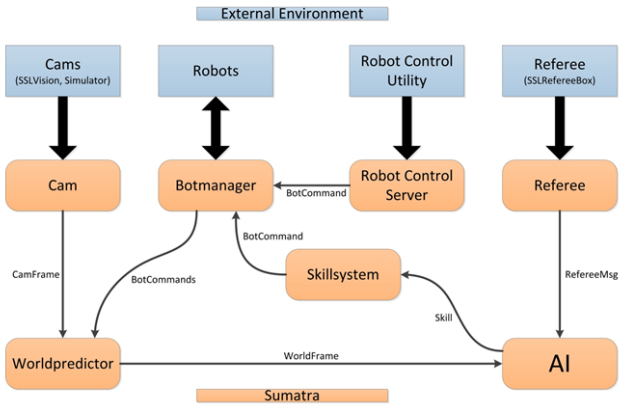

Fig. 2. Sumatra Architecture and its connection to the periphery

Our software system consists mainly of two programs: the simulator Tigers Cage Simulator and the main control software Sumatra. The simulator has been programmed for testing our Artificial Intelligence (AI) in a virtual environment.

This part is described in section 3.2 in more detail.
Sumatra interacts directly with the SSL environment. The software is responsible for getting the current image-data from SSL-vision, reacting to the newest referee instructions, calculating the best strategy for the next interactions for the robots and sending the resulting commands to the robots. Of course, Sumatra can also interact with our Simulator, so no real environment is needed. In Figure 2 there is an overview concerning the SSL environment and the internal structure of Sumatra.

MoveInCircle Skill A *Skill* is a set of basic robot commands to fullfil a special purpose, for example move to a destination or turn around the ball. They are necessary to improve the robot handling within the AI.
The *MoveInCircle Skill* enables the robot to move on a circular trajectory. Generally, commanding a move is again as simple as sending a *TigerMotorMoveCommand*. The *Command* takes a vector as parameter. The direction of the vector determines the direction of the move, the length corresponds to the velocity.
The difficulty here is, that with the basic *TigerMotorMove* only straight moves are possible. Curved trajectories need to be approximated with a reasonable amount of straight parts. This is done on *Skill* level. Since the aiming capability requires only circular movements, the development of a generic "move-on-curve" ability which might be based upon splines, was delayed.
The *Skill* uses the following algorithm to approximate a circular trajectory. The goal is in each cycle to calculate a point on the circle (B') towards the robot (B) shall move. The circle is defined by the center (C) and the radius ($|\boldsymbol{cb}|$), which are both input parameters. See Figure 3 for an overview.

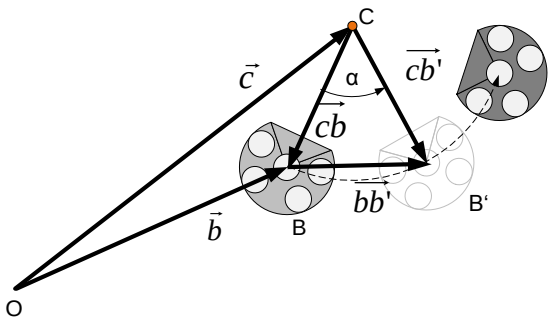

Fig. 3. How to move on a circular trajectory

Another important input parameter is the target angle from which the arc the bot shall travel is calculated. α defines the current robot position (in reference to the center and the x-axis) whereas the target position is defined by β. (See figure 4)

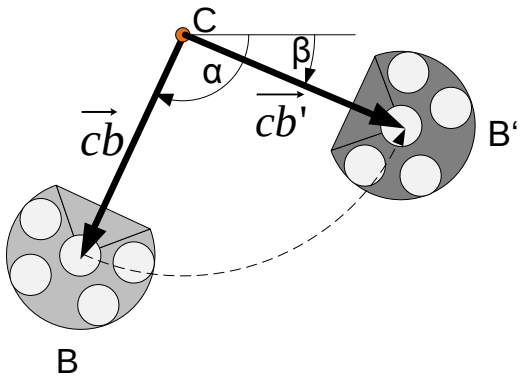

Fig. 4. robot and target position

The arc can now be calculated with

$$\text{arc} = \beta - \alpha \tag{1}$$

A positive value indicates a counter-clockwise movement and vice versa. The robot will always automatically take the shorter distance.

The next point on the circle the bot shall move to is now calculated with a scaling and rotation of the vector $\boldsymbol{cb}$. The vector is scaled to the constant passed in aiming distance to ensure that the robot keeps that distance. The new vector is calculated as follows (Figure 3):

$$\boldsymbol{cb'} = R_\alpha \cdot \boldsymbol{cb} \tag{2}$$

where

$$R_z, \alpha = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{3}$$

The rotation angle α determines the smoothness of the approximation. Of course it is also dependent on the radius, but since the skill is currently used just for

aiming purpose, the angle is adapted to that scenario. The default value which accomplished good results is $\pi/18 = 10\,\text{deg}$. If the arc is smaller than that default step size, the rotation is performed with $\alpha = \text{arc}$.

Now everything is known to calculate the new move vector:

$$\boldsymbol{bb'} = \boldsymbol{cb'} - \boldsymbol{cb} \tag{4}$$

Now, that vector is still in global coordinates and needs to be transformed. The final move vector ($\boldsymbol{m}$) that gets commanded to the robot is obtained from a simple rotation:

$$\boldsymbol{m'} = R_\beta * \boldsymbol{m}\beta = \frac{\pi}{2} - \alpha \tag{5}$$

The final operation that needs to take place before the move vector can be transferred to the robot is the application of the desired velocity. Velocity is represented by the length of the move vector. Thus a simple scaling of the vector applies the velocity.
The velocity is a linear function of the length of the arc the bot shall travel, where the slope is determined by the maximum allowed velocity and the distance it takes to decelerate. Both parameters are variable and dependent on the environment (eg. friction).

3.2 Module: Artificial Intelligence

Our AI uses still the same approach then in the year 2011. For a better understanding of the following play description important terms are explained. The internally used data structures are the *Play*, *Role* and *Condition*. A *Play* defines what the overall-plan for the next seconds is, e.g. an indirect shot. It contains a set of *Roles*, not necessarily a *Role* for every robot, since there can be more than one *Play* active at a time. A *Role* is a specific task within a *Play* and is mapped to our robots 1:1. For example there are two *Roles*, one that passes and one that shoots. A *Role* is defined by its *Conditions*. Such a *Condition* may be a *LookAt condition*, meaning that the owner of the *Role* has to aim at a specific point, or *Destination condition*, defining a destination for the owner. Each Role tries to fulfill its set of *Conditions* as good as possible.

Defense play with 1 robot Due to the official conditions of a Small Size League participation, a qualification video must be shown. This video shall prove

that the team's robots are able to take part in an active game. The first requirement is as follows:

a) One or more robots competing against an active goalkeeper.

To accomplish this requirement, a *KeeperSoloPlay* with an associate *KeeperSoloRole* was programmed. According to the official rules, no field player is allowed to move in the goal area. To prevent interference with other robots, the goalkeeper's movement shall be limited to this area.
A first idea of positioning the goalkeeper was a circle around the goal's central point. The radius was set to the distance between the goal line and the forward border of the goal area. Possible target points are thereby defined by the resulting semicircle.

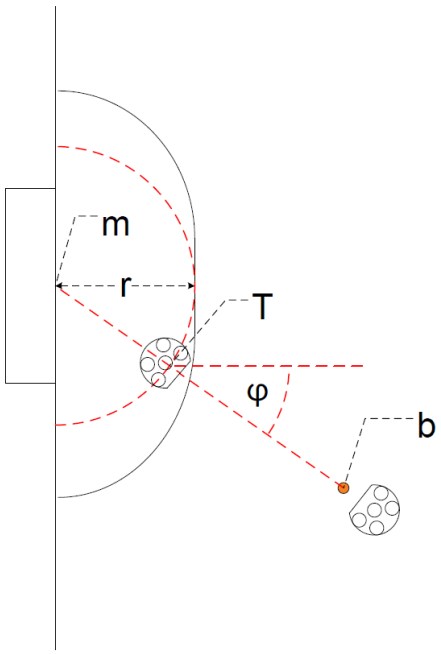

Fig. 5. positioning of the keeper

As shown in picture 5, the point of intersection of the line goal's central point (m) - ball position (b) and the semicircle sets the defense position T of the goalkeeper, calculated as follows:

$$\boldsymbol{T} = \boldsymbol{m} + \frac{r}{|\boldsymbol{b} - \boldsymbol{m}|} * (\boldsymbol{b} - \boldsymbol{m}) \tag{6}$$

In addition to this, the robot's front is supposed to always be directed to the ball's position. In doing so, catched balls are easier to control. All angles beneath the x-axis are negativ, while above positive. Therefor, all angles lie between [$-\pi$; π]. Regarding picture 5, the angle of orientation is $-\phi$ here.
Two skills are belonging to the *KeeperSoloRole*: A *MoveToXY-skill* with corresponding target position parameter and a *Rotate-skill* with orientation angle. The pathfinding and the parallel processing of the skills are taken care of by the skill system, the high-level programmer need not to mind that.

Defense play with 3 robots One defensive play to keep our goal clean is the *KeeperPlus2DefenderPlay*. It controls the keeper and two field players to block our goal. Thereby the keeper role is used as a master to correctly synchronize the three robots. The defender roles only keep track of the keeper and position the defenders accordingly. The keeper role of *KeeperSoloPlay* could be used but for organisational reasons, a new keeper role is used. Additionally, new approaches for a central defending goalkeeper can be implemented. Instead of using the line between central goal point and the ball position, the angle bisector between the left goalpost-to-ball-line and right goalpost-to-ball-line is considered. This implementation allows a more effective positioning for the keeper, especially from side attacks. For easy computations of the angle a simple geometric attribute of a triangle is used:

When the bisector of an angle divides the opposite side, the proportions of these parts are accordingly to the lengths of their adjoining sides. Thus equation 7 is true for 8.

$$\frac{a1}{a2} = \frac{s1}{s2} \tag{7}$$

$$s1 = \frac{a1}{a1 + a2} * s_{total} \tag{8}$$

The intersection S of the bisector angle w with the goal line is the new origin for the positioning of the goal keeper. The directional vector of the bisector angle with an adjusted length is used to get the desired point T. To get the positions for the two defenders, the keeper's position is used as the starting point. For both defenders the vectors moved towards the ball and then orthogonal to the bisector angle, once to the left and once to the right. The length should be implemented dynamically to adapt to the current game.

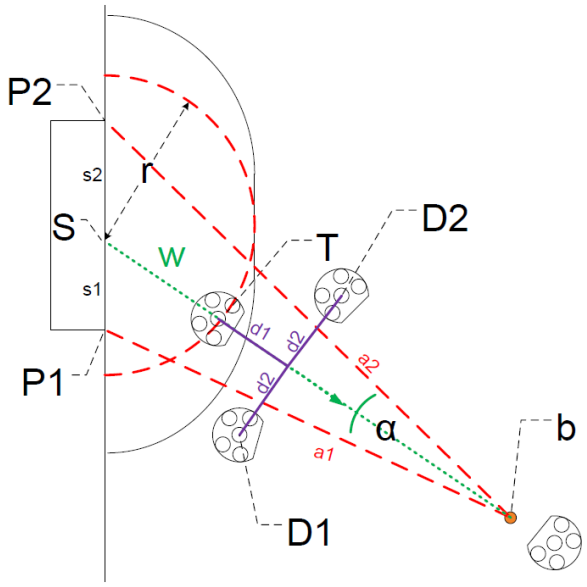

Fig. 6. Positioning of the keeper with two defenders

Defense play with 2 robots The analysis of the gameplay of other teams shows that goals are often a result of a takeover from the opposing team by long range shots. Due to the high ball-speed one keeper often has no time left to block the ballway. As a countermeasure, one defense player should stay with the keeper while the other robots are attacking.
In this case the keeper should not try to block centrally, but substitute the other defense player. Like in the play before, the player positions will be defined through an angle bisector (W). The positions of the bots are asymmetrical because keeper and defense bot must be on different sides of the goal line. To obtain high flexibility, the side at which a bot stays will be defined at runtime. Therefore the play checks through the worldframe which bot is more on the left side and chooses the side accordingly.

3.3 Tactical field assessment using grid analysis

We are doing a tactical analysis of the whole field to determine the positions of all bots and to mark good and bad positions on the field to pass or to move a bot to. Therefore the whole field is divided into several rectangles. Each rectangle is evaluated by its own, the quantity of rows and columns can be configured in a separate xml-file. Each rectangle gets a value to estimate whether it is a

good point to pass the ball or move the bot to or not. Thus a few algorithms to evaluate the rectangles have been tested and are now delineated in 3.4.
The AI module prefers moves and/or passes via positions that our own bots can easily reach. To avoid ball interceptions, enemy bots should not to be in close range to avoid interceptions of the intended moves. The tactical analysis of the field shall help to choose a play or a tactic and to accomplish this play.

Rectangle quality The rating of the rectangles is determined by the distance from their center to our own bot or to an enemy bot. Low values represent a good rectangle for our team (own bot is closer to a rectangle than an enemy bot) while a high value shows the opponent. There the rectangle is even occupied by an enemy bot or it can reach the rectangle faster than our bots could do. To rate the rectangles, a few algorithms have been implemented, tested and compared. They are described below.

3.4 Rating the rectangles

Bot position, moving direction and speed A simple approach is to iterate over all bots and check if they are currently inside the rectangle. For each bot found in the rectangle the rating is adjusted. The rating increases for each enemy bot in the rectangle and diminishes for each own bot. This approach can be modified to pay attention to the moving vector and speed of the bots. Considering fast speed and direction changes of the bots the ratings provided by this algorithm are not significant. Therefore other algorithms have been developed.

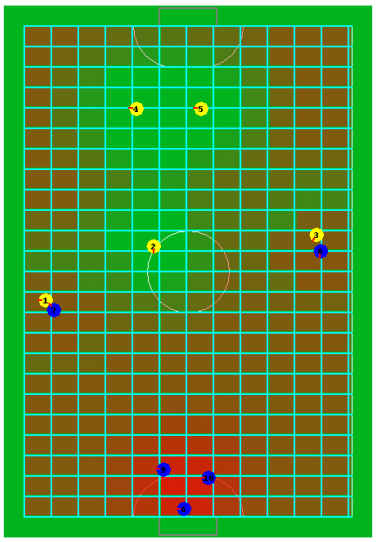

Fig. 7. Gaussian Fieldraster

Gaussian distribution This algorithm iterates over every bot and calculates the distance between the center of the rectangle and a bot, like the one mentioned in 3.4. The main difference lies in the following fact: The distance of a bot to the center of the rectangle is divided by the longer side of the rectangle to get a scaled value. If this scaled value exceeds a certain determined value, the bot is considered to be unimportant for the current reviewed rectangle, it is too far away to have an impact. Otherwise a Gaussian distribution for N(0,2) is being created. Attention should be paid to the especially therefore scaled value, which is needed because the result of this operation approaches zero for larger values. To obtain even better values, the result is multiplied with a special factor, the TigerFactor. The TigerFactor stresses out areas that are occupied by own bots with no enemy bots in range and the other way round. The TigerFactor gets the value 5 at the beginning and is reduced by every enemy bot found in a radius around the rectangle. If an enemy bot is next to a certain rectangle, the value of the TigerFactor for this rectangle is set to 1. This implies that all fields next to an enemy bot have a bad rating. A result of this calculation is shown in figure 7.

Bot distances This approach determines the own and the enemy bots with the smallest distance to a rectangle. The difference between these two distances, scaled per dividing the distance by the longer side of the rectangle (refer to chapter 3.4) and multiplied with a correction factor is being added to an initial value of a rectangle rating. Free ways for own bots can be identified with this algorithm in contrast to the algorithm described in chapter 3.4. This approach provides a more consistent view of the field and gives best results for our team. See figure 8 for an illustration of this algorithm.

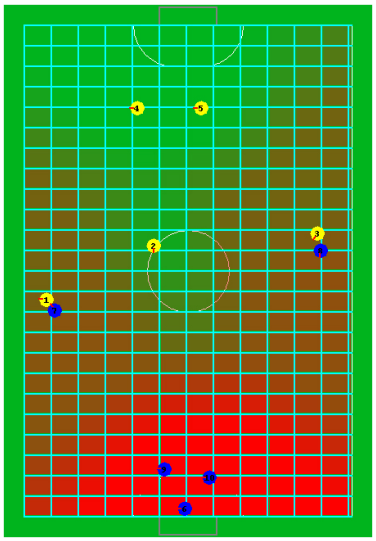

Fig. 8. Fieldraster using bot distances

Realtime considerations The tactical field assessment can create a lot of workloads for the AI system. To maintain good results from the tactical field assessment, there is no need to update all rectangles in every AI loop. The loop frequency of the AI system is usually a lot higher than notable changes on the field occur, so it does not make a huge difference if a rectangle gets updated every third or fourth AI loop. Anyway, rating calculation did not create a notable delay in our system.

4 Prospect

Our qualification video features the system at the version of January 2012. After the tournament in Mexico all available documentations and source codes will be published on our website, so that other teams can take advantage of our work. Due to the fact that a lot of members of our current team will graduate this year, a new team (which already exists) will take over this project. Besides the usual SSL work, they also will take a look on an autonomous referee robot and an automated camera calibration of the SSL Vision for the SSL.

References

1. Robocup Small Size League Homepage, http://small-size.informatik.uni-bremen.de/robocup2012:qualification
2. Waigand, D., Berthold, G.: Cooperative shoot and pass behaviour of mobile robots in the context of the TIGERS-Mannheim SSL Robocup-project (2011) http://tigers-mannheim.de/index.php/en/download/category/5-studys-en
3. Koenig, C., et al.: Articial Intelligence - Overall Documentation(2011) http://tigers-mannheim.de/index.php/en/download/category/5-studys-en
4. Mauelshagen, M.: Entwicklung und Implementierung defensiver Spielstrategien fuer das RoboCup-Projekt des Teams Tigers Mannheim(2011) http://tigers-mannheim.de/index.php/en/download/category/5-studys-en
5. Steinbrecher, O., Birkenkampf, P.: Taktische Spielfeldanalyse im Robocup mittels Rasterung des Spielfelds (2012) http://tigers-mannheim.de/index.php/en/download/category/5-studys-en