Master Thesis
Software Engineering
Thesis no: MSE-2003:19
June 2003

Machine learning in simulated RoboCup
Optimizing the decisions of an Electric Field agent

Markus Bergkvist Tobias Olandersson

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden

This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 2×20 weeks of full time studies.

Contact Information:

Author: Markus Bergkvist
Address: Blasius Königsgatan 30B, 372 35 Ronneby
E-mail: markus.bergkvist@telia.com

Author: Tobias Olandersson
Address: Blåbärsvägen 27, 372 38 Ronneby
E-mail: pt99tol@student.bth.se

University advisor:
Stefan Johansson
Department of Software Engineering and Computer Science

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden

Internet : www.bth.se/ipd
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

Abstract

An implementation of the Electric Field Approach applied to the simulated RoboCup is presented, together with a demonstration of a learning system. Results are presented from the optimization of the Electric Field parameters in a limited situation, using the learning system. Learning techniques used in contemporary RoboCup research are also described including a brief presentation of their results.

Keywords: Self-learning, Parameter optimization, Simulated RoboCup, Multi-Agent system, Electric Field Approach

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> "By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team."
> -RoboCup Federation [2]

The goal and timetable is aimed to advance the overall level of technology in society. There will be several technological achievements even if the goal is not completely fulfilled, ranging from improved sensor reading computations to advanced multi-agent coordination policies.

1.1 RoboCup

The idea of using soccer-playing robots in research was introduced by Mackworth [1]. Unfortunately, the idea did not get the proper response until the idea was further developed and adapted by Kitano, Asada, and Kuniyoshi, when proposing a Japanese research program, called Robot J-League, a professional soccer league in Japan [1].

During the autumn of 1993, several American researchers took interest in the Robot J-League, and it thereafter changed name to the Robot World Cup Initiative or RoboCup for short. RoboCup is sometimes referred to as the RoboCup challenge or the RoboCup domain. In 1995, Kitano et al. proposed the first Robot World Cup Soccer Games and Conferences to take place in 1997 [1].

The aim of RoboCup was to present a new standard problem for AI and robotics, somewhat jokingly described as the life of AI after Deep Blue [1]. RoboCup differs from previous research in AI by focusing on a distributed solution instead of a centralized solution, and by challenging researchers from not only traditionally AI-related fields, but also researchers in the areas of robotics, sociology, real-time mission critical systems, etc.

To co-ordinate the efforts of all researchers, the RoboCup Federation was formed. The goal of RoboCup Federation is to promote RoboCup, for example by annually arranging the world cup tournament. Members of the RoboCup

Federation are all active researchers in the field, and represent a number of universities and major companies. As the body of researchers is quite large and widespread, local committees are formed to promote RoboCup-related events in their geographical area.

In order for a robot team to actually perform a soccer game, various technologies must be incorporated including: design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion. RoboCup is a task for a team of multiple fast-moving robots under a dynamic environment.

1.1.1 Simulated league

The RoboCup simulator league is based on the RoboCup simulator, the soccer server [1]. The soccer server is written to support competition among multiple virtual soccer players in an uncertain multi-agent environment, with real-time demands as well as semi-structured conditions.

One of the advantages of the soccer server is the abstraction made, which relieves the researchers from having to handle robot problems such as object recognition, communications, and hardware issues, e.g., how to make a robot move. The abstraction enables researchers to focus on higher level concepts such as co-operation and learning. Since the soccer server provides a challenging environment, i.e., the intentions of the players cannot be mechanically deduced, there is a need for a referee when playing a match. The included artificial referee is only partially implemented and can detect trivial situations, e.g., when a team scores. However, there are several hard to detect situations in the soccer server, e.g., deadlocks, which brings the need for a human referee. There have been five world cups and one pre-world cup event [1].

1.1.2 Environmental issues

The type of environment in RoboCup is one of the hardest to deal with according to the definition of environments in "Artificial Intelligence, A modern approach" [9].

- It is inaccessible since the field of view is limited to the view angle and it is only possible to see a part of the soccer field, i.e. the agent has to maintain an internal state of the soccer field.
- It is nondeterministic from the agents viewpoint[1] because the next state of the environment cannot be determined by its current state and the actions selected by the agent. For instance, there are 21 other players the agent can only guess what actions they will select, and it is not possible to calculate the exact trajectory of the ball.
- It is nonepisodic because the agent has to plan its actions several cycles ahead, and every action the agent does has impact on the subsequent cycle.

[1] It is deterministic from the viewpoint of the system, since the noise can be predicted given that the randomization key and the movement model is known.

- It is semidynamic because the environment will not change while the agent is deliberating if the agent comes to a decision and acts within passage of the cycle time (currently 100ms). If the agent does not send a command to the server before the ending of the cycle, the server calculates the next state without any action from the agent.
- It is discrete in the sense that the agent know what flags and number of players he can see, and what actions the agent can do. But it is continuous in the sense that the possible perceptions of the flags and the players are neither limited nor clearly defined.

1.2 Electric Field Approach

In autonomous robotics, artificial potential fields are often used to plan and control the motion of physical robots [6]. The *Electric Field Approach* [4] is proposed as a generalization of traditional potential field approaches, which allows for both motion control and object manipulation. The main concepts of the electric field approach are artificial charges and probes, where strategically important positions of the probe are positively charged and "dangerous" positions negatively charged. The electric field is then used as a heuristic[2] for action selection by simulating the potential at the probed position for each action. The action resulting in the highest potential in the probed positions are regarded as the most suited for execution in the current situation. This can be achieved either by moving the probes or moving the charges.

1.3 Learning techniques

The classical approach to building intelligent agents, by manually configuring the agent to fit the environment suffers from a number of disadvantages [9]. In order to create such an agent the designer must have a complete model of the environment, as well as a model of which actions to take in each situation.

The field of *machine learning* is a subfield to AI (Artifical intelligence) concerned with programs that learn from experience [9]. Machine learning can therefore be a reasonable alternative to classical approaches when dealing with incomplete knowledge of the environment of operation. The idea behind learning is that an agent should not only use its percepts for acting, but also to improve the way in which it acts.

The following sections describe some of the learning techniques currently used in the RoboCup domain.

1.3.1 Reinforcement learning

Reinforcement learning is a technique that has gained a lot of attention from AI researchers in the last years, due to its effectiveness of operation in complex domains [5].

[2]The word "heuristic" is derived from the Greek verb *heuriskein*, meaning to "find" or "to discover".

The basic concept of reinforcement learning is to let the agent know when it is performing well or badly. The reinforcement is a kind of reward (or punishment) given to the agent as a performance measure after a finished learning phase.

Since reinforcements are given to the agent only after a finished learning phase, and not after each decision made, the agent is only aware of its overall performance in that phase. The agent can therefore not directly know which of the individual actions that were good or bad respectively.

The task of the learning system is to utilize the reinforcements given after each learning phase to learn a mapping of environment states with a utility function. The obvious advantage with this approach is that it is not needed to have a human supervisor that evaluates each situation. The task of the supervisor is hence only to define the overall goal of the agent, e.g. to win a game of chess. A disadvantage is that a large number of learning phases is needed in order to transform the knowledge given by the reinforcement into a state → utility mapping.

Consider the task of learning to ride a bike. The reinforcement in this case is a punishment given each time the bike falls over. By mapping the states leading to the bike falling over with negative reinforcements, the learning system will soon prefer taking actions not leading to these "dangerous" states.

1.3.2 Q-learning

Q-learning is a form of reinforcement learning where not only the states, but also the actions, are associated with a utility.

$$U(s) = \max_a Q(s, a) \tag{1.1}$$

where $U(s)$ is the utility in state s and $Q(s, a)$ is the Q-value associated with taking action a in state s [9].

An advantage with q-learning over ordinary reinforcement learning is that a model of the environment is not needed in order to select the action leading to the preferred state. All it has to know is which actions that are legal in the current state, and compare the utilities for each action in that state. This leads to efficient implementations since the (state, action) → utility mapping can be stored in a lookup-table, usually referred to as a q-table [11].

1.3.3 Hill-climbing

Hill climbing is an iterative search heuristic that tries to find peaks on a surface of states where the height is defined by the evaluation function [9].

The search starts at a selected state in the multidimensional search space, where each dimension corresponds to one of the state-parameters. The algorithm then evaluates all neighboring states and proceeds to the state with the highest value. When no progress is made a local optimum is found and the algorithm stops.

This algorithm has three well-known draw-backs [9]:

Local optimum The algorithm stops when finding a local optimum. There is no way to know if the local optimum in fact is the global optimum. This is a serious problem since the local optimum found may not be nearly as high as the global optimum.

Plateaux A plateaux is an area in the search space where the evaluation function is essentially flat. In such a case the algorithm will not know which path to choose. The result in this case will be a random walk until an end of the plateaux is found.

Ridges A ridge may have steeply sloping sides leading to fast progress to the top of the ridge, but if the ridge itself is only slowly increasing towards the optimum little further progress will be made.

A technique to reduce the risk of getting stuck on local optima is to run the algorithm a number of times with randomly chosen starting states, called *random restart hill-climbing* [9]. The success of the random restart hill-climbing is very much dependant upon the topology of the search space, if there is only a small number of local optima the solution will be found quickly. Usually a reasonably good solution can be found in only a few iterations.

1.4 Contemporary research

To set a frame of reference for the further discussion in this report, we here present a few examples of contemporary RoboCup teams utilizing machine learning techniques. The teams have been chosen based on their outstanding performance in the RoboCup Championships, and on the relevance of comparison to the approach presented in this report.

1.4.1 Brainstormers

A most interesting team is the german Karlsruhe Brainstormers, which draw the use of machine learning to its peak. Their final goal is to have a team where the agents have learned all of their behaviour, from low-level skills like kicking the ball, to high level team strategies [8].

Their team has currently reached a development stage where reinforcement learning is used for all the low-level skills. These skills are; kick, intercept-ball, dribble, positioning, stop-ball and hold-ball.

Brainstormers are also conducting early experiments on reinforcement learning on the tactical level. Their initial experiment was to learn team strategies for the situations; 2 attackers vs. 1 defender, and 2 attackers vs. 2 defenders. The task for the attackers was simply to score as quickly as possible, and for the defenders to get the ball from the attackers.

Table 1.1 shows the results of the reinforcement learning approach contrasted against a greedy policy, which gets the ball and tries to score. The percentages show the success ratio of the attackers. The results are promising, but are currently not incorporated in the competition agent. Issues still remain concerning how to scale from 2 vs 2 to a full team.

	Greedy	RL trained
2 against 1	35 %	85 %
2 against 2	10 %	55 %

Table 1.1: Reinforcement Learning approach vs Greedy policy.

1.4.2 Tsinghuaeolus

The Chinese team Tsinghuaeolus of Tsinghua University, Beijing is another team that utilizes machine learning. Tsinghuaeolus is also the Simulated RoboCup Champions of 2001 and 2002.

Tsinghuaeolus use a strategy for low-level skills that is very similar to the Brainstormers approach. The main difference between the approaches of the two teams lie in the configuration of the higher-level decision-making. Tsinghuaeolus consider the use of learning on higher-levels inferior to more analytical approaches, whereas Brainstormers preliminary results shown in Table 1.1 indicates the opposite. [15]

1.4.3 CMUnited

The American team CMUnited, from Carnegie Mellon, who won the Simulated RoboCup Championship in 1998 and 1999 use a hierarchical machine learning paradigm which they refer to as layered learning [13]. Layered learning applies to tasks for which learning a direct mapping from perceptual inputs to actuator outputs is intractable due to the complexity of the domain. The concept of layered learning is to decompose the learning task into subtasks, where the learning of each subtask facilitates the learning of the next higher layer subtasks.

They have presented results from their experiment on the use of reinforcement learning to learn an optimal policy for the high level subtask of keep-away. The goal in the keep-away situation is for the keepers to keep the ball within the team for as long as possible, while the takers try to take control of the ball.

The results shows that their reinforcement approach to keep-away soccer performed 2 to 3 times better than any of the hand-coded policies in the benchmark. These policies where; *Random* which chooses actions randomly from the set of applicable actions, *Hold* in which the keeper with the ball remains stationary while trying to keep the ball without of the range of the takers, and *Hand-coded* which holds the ball if no taker is within 10m, otherwise passes the ball if another keeper is in a better location [12].

CMU have also investigated the importance of taking temporal aspects into consideration when applying reinforcement learning methods on high-level behaviours [7].

1.5 Problem description

Can the action selection of autonomous RoboCup agents be improved by applying self-learning methods to the Electric Field Approach?

1.6 Delimitations

The scope of this thesis is to investigate the applicability of machine learning to simulated RoboCup agents utilizing the Electric Field Approach. The applicability of machine learning for other RoboCup agents will not be discussed.

1.7 Method

Two research methods was utilized in the project; a literature study and an evaluation through experimentation.

1.7.1 Literature survey

The first task was to conduct a literature survey of architectures for RoboCup multi-agent systems and self-learning methods in the RoboCup domain. This was performed to clarify what has already been done, and consequently which questions remain open.

1.7.2 Experiments

The second task was to adjust the Electric Field Approach described by Saffiotti and Johansson [4], and designed for the Sony Aibo legged league, to fit the particular characteristics of the simulated RoboCup domain. The main work has been to design and implement a team for the simulated RoboCup domain. A large effort has been put into constructing a self-learning model that utilizes the EFA for action selection. The performance of the self-learning method was then evaluated by comparing a learned team against a team that has been manually set up.

1.8 Thesis outline

This thesis begins in chapter 2 with a description of the implementation needed to fulfill the project. Chapter 3 continues by describing the experiments conducted and presentating the results thereof. In chapter 4 a discussion is held on the results of the experiments. The thesis is then concluded in chapter 5, together with a suggestion on further work in the area.

Chapter 2

Implementation

2.1 CRaPI, a RoboCup API

CRaPI - Correct Robust and Perfect API for simulated RoboCup, is our attempt to provide an API that is well documented, simple to use and agent architecture independent.

The responsibilities of CRaPI is to provide an up-to-date model of the current state of the environment, a means of communication with the soccer server and a set of utility functions to relieve the user of tedious calculations not related to agent design. In order to keep CRaPI independent of the agent architecture chosen by the user, it is driven by events. The current size of CRaPI is 7228 LOC (lines of code). See the CRaPI manual concerning the details of how to set up a team using CRaPI [14].

An architectural overview of CRaPI is shown in Figure 2.1. The main parts are;

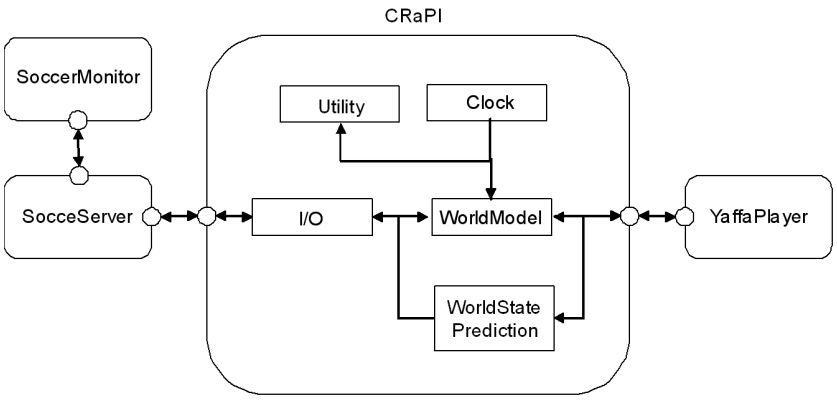

Figure 2.1: Architectural overview of CRaPI

WorldModel The WorldModel is the agent's internal representation of the current state of the environment. The responsibilities of the WorldModel

is to calculate the positions, velocities and directions of all objects in the environment, based on perceptual inputs.

WorldState Prediction Since perceptual inputs are not synchronized with the server cycles, a prediction of the world state is needed to maintain high coherence with the actual state of the world.

Clock The internal clock is needed to keep track of when a cycle is about to end, and hence when a command has to be sent to be executed by the server in the current cycle.

Utility The utility module provides a set of useful tools, mostly concerned with geometrical calculations.

I/O The I/O module is responsible for maintaining the communication with the soccer server.

2.2 Yaffa, a RoboCup player

Yaffa - Yet Another Force Field Approch, is our RoboCup player. The development of Yaffa has been the single largest task of the project. Yaffa is built on top CRaPI, and uses the electric field model for action selection. The current size of Yaffa is 8630 LOC.

An architectural overview of Yaffa is shown in Figure 2.2. The main parts are;

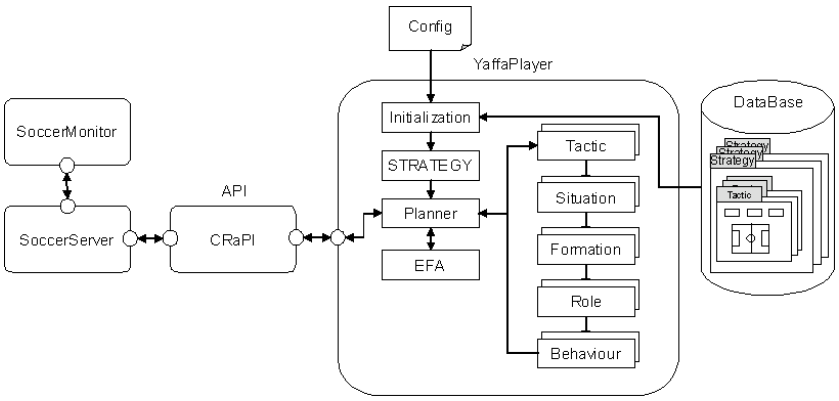

Figure 2.2: Architectural overview of Yaffa

Planner The main part of the decision maker. Based on the current Strategy, Tactic and Situation a Formation is chosen. A set of Behaviours is then evaluated depending on which Role the individual player has in the current Formation. The Behaviour found to be most appropriate according to the evaluation is chosen for execution in the current cycle.

Strategy A high-level configuration of the team, allowing for different set ups when meeting different teams.

Tactic A configuration of Formations depending on the current situation in a game. Different tactics can be used to customize the play when the team is at a disadvantage, compared to when it is leading.

Situation A description of the state of the game.

Formation A collection of Roles, that form the set up of the team in a situation. E.g. 4-3-3, 3-3-5, etc.

Role A definition of the responsibilities of a player. i.e. the Behaviours to run and the position in the Formation to take.

Behaviour A high-level action, e.g. getting passable or block an opponent.

EFA A heuristic for action selection, used to evaluate the Behaviours. See section 1.2 for more information.

Database Used to store the configuration of Strategies, Tactics, Situations, Roles and Behaviours.

2.3 Our approach

This section describes the implementation of the Electric Field Approach used in Yaffa, adjusted to work in the simulated RoboCup domain. It also describes the implementation of the learning algorithm used in the experiment, and how we selected which parameters to optimize.

2.3.1 Conceptualization of EFA

The behaviours in our model are divided into *Navigation* and *Manipulation*. The behaviours in Navigation are; BlockOpponent, GetPassable, IntersectBall and RunHome. The behaviours in Manipulation are; DribbleWithBall and PassTheBall. For more detailed information regarding these behaviours see Appendix A.

Navigation

The behaviours in Navigation are concerned with the positioning of the agent. The task of these behaviours is to localize the most beneficial location for the agent, and then to navigate the agent to this location.

Different charge configurations are used for the Navigation behaviours, since the preconditions and the objectives varies. For certain behaviours it is positive to stand close to an opponent, whereas in others it is clearly negative. Due to the preconditions these behaviours will never compete, e.g. in BlockOpponent it is the opponent team that controls the ball, whereas in GetPassable the ball is within the control of the own team.

All Navigation behaviours have in common that the goals have no influence on the navigation. Instead the special purpose fields, Defensive and Offensive,

are used to help to navigate when the agent are in defense or attack mode respectively.

The object to probe in Navigation is the agent, hence the probe positions for each behaviour are the predicted positions of the agent after it is done executing the behaviour.

Manipulation

The behaviours in Manipulation are concerned with manipulating the location of the ball. The task of these behaviours is to localize the most beneficial location of the ball, and then to transfer the ball to this location.

The charges in Manipulation are; PosMate, NegOpponent, Self, Offensive, TeamGoal, OpponentGoal and Boundary. The configuration of charges is identical for the Manipulation behaviours, since the common goal is to transfer the ball to the most positive location. The only difference is how the transfer is performed.

The object to probe in Manipulation is the ball, hence the probe positions for each behaviour are the predicted positions of the ball after it is done executing the behaviour.

2.3.2 Implementation of EFA

Our implementation is based on two main components, the Engine and the FieldGenerator. The responsibility for generating the fields is assigned to the FieldGenerator, whereas all electric field calculations are handled by the Engine.

The main aspect of our design is to let the FieldGenerator pre-calculate as much as possible before an actual game begins. Since the interaction between electric fields is only a matter of superposition[1], electric fields for all specified types of charges can be pre-calculated. To improve the performance of the superposition, the fields are implemented as layers of matrices in a *composite electrical field*, hence no complete superposition of the fields are made, only the probed position is superimposed.

The generation of a field surrounding a charge is made in three steps. The first step is to calculate the size of the matrix that will represent the field. This is done by calculating the radius around the charge where the potential reaches a predefined threshold value, all values below this threshold value is ignored in further calculations. The radius is calculated by applying a distribution formula, which loosely mimics the potential distribution around real electric charges[2]:

$$p = \frac{p_{center}}{d^k} \tag{2.1}$$

where p is the potential at distance d from p_{center} and k is the decrease constant. Extracting d from equation (2.1) we get the formula for the radius:

[1]Superposition of electric fields means that the potential in a point p is the sum of the potentials for all fields in point p. A field of charge x and a field of charge $-x$ placed on the same position results in a field of strength 0.

[2]The potential distribution around real electric charges is described by Coulumbs law. Equation (2.1) is equivalent to Coulumbs law when $k = 2$

$$r = \sqrt[k]{\left(\frac{p_{center}}{p_{threshold}}\right)} \tag{2.2}$$

where r is the radius of the charge and $p_{threshold}$ is the threshold potential. The second step of the field calculation is to calculate the potential for each cell in the field matrix. These calculations uses equation (2.1) with distance d set to the distance in squares from the center multiplied with the size of each square. The third and final step is to use the concept of superposition to add together all the charges in the current field.

Apart from field generation based on potential distribution around artificial charges, we also utilize a special purpose field generation of the boundary lines. The purpose of this field is to defer the players from running outside the field, but without setting an absolute condition. This field uses the same distribution formula as equation (2.1). The difference between this calculation and the standard charge-based calculation is the way in which the potentials are distributed. Since the boundary line field is designed to simulate a set of charges placed with infinite density around the field, the potentials are decreasing only inwards on the field, without using superpositioning.

Visualization of EFA

During the development of *CRaPI* [14], we learned that a tool to visualize the view of the agent can be very useful. With such a tool we can confirm that the agent has the view of the environment that it is supposed to have. The tool we developed to visualize the Electric Field Engine offers the following functionality:

- Configure the charges of the predefined fields that are generated for the boundary line, a mate, an opponent, the ball and the agent.
- Position and configure static charges, e.g. the goal-charges.
- Compare the strength of the charges. You should be able to see both the relative strength and the propagation of the charge.

In Figure 2.3 some of the fields that can be configured with the Electric Field Generator is visualized.

2.3.3 Implementation of a learning system

We have chosen to adopt the learning agent partitioning suggested by Norvig et. al. [9] which states that the agent can be partitioned into four main components;

Performance element: the Yaffa action model, which is the core of the decision making in our agent. The decisions are based on probe-values in the electric field, where the positions to probe are chosen by the individual behaviours. The behaviour resulting in the highest probe-value is chosen for execution.

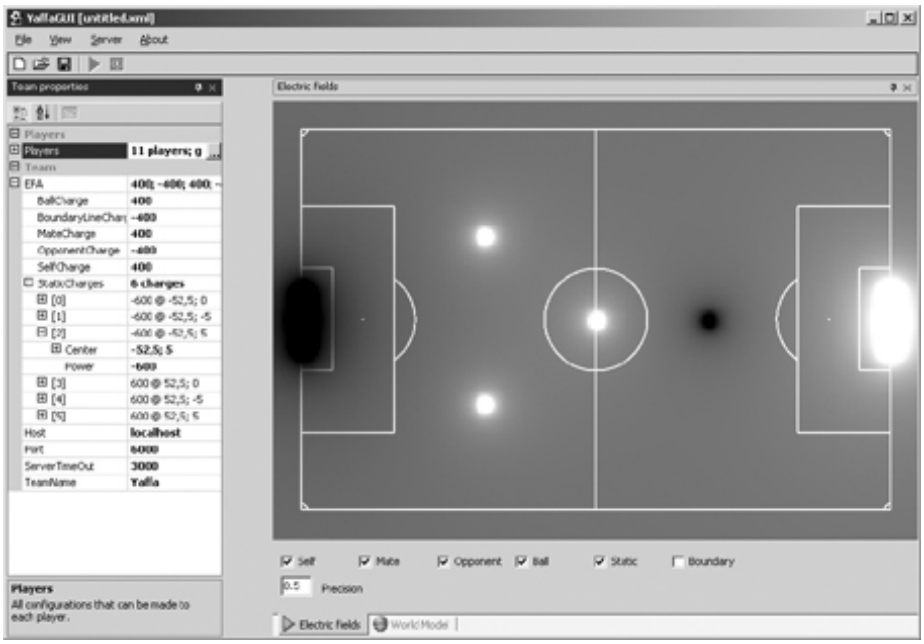

Figure 2.3: Visualization of the Electric Field Generator

Critic: utility calculation implemented by a trainer.

Learning element: hill-climbing heuristic to find the optimal electric field charge configuration in the space of all possible charge combinations.

Problem generator: for each training round, a starting state in the search space is selected at random by the trainer, to reduce the risk of get stuck on local optima [9].

Using the EFA for action selection implies the opportunity to optimize the field parameters; *charge* and *spread function*[3]. In order to delimit the size and complexity of the project our effort has been focused on optimizing the charges only. The parameters selected for optimization can be described with:

$$a_1, ..., a_n, a_1 \in A_1, ..., a_n \in A_n \tag{2.3}$$

where A_n is the set of possible charges for object n. For more details about which charges used in each behaviour, see Appendix A.

The fitness function $f_s(a_1, ..., a_n)$ given the situation $s \in \mathbb{S}$ describes the utility of the parameter configuration $a_1, ..., a_n$ in situation s.

The situation we have chosen to test our approach on, is a 3 vs 2 keep-away soccer as described by Stone and Sutton in *Scaling Reinforcement Learning toward RoboCup Soccer* [12].

The purpose of the situation is for the keepers to keep possession of the ball, within a limited boundary, for as long as possible. The situation is considered

[3]The spread function describes the potential distribution around a charge, see equation (2.1)

to be over when either one of the takers retrieves the ball, or when the ball crosses the boundary of the training field.

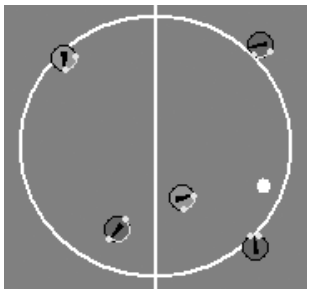

Figure 2.4: Snapshot of a keep-away situation

Despite the delimitations made, the state space of possible charge configurations is too vast to enable a complete search. A solution to this problem is to use a heuristic that can approximate the best configuration without conducting a full search. Such a heuristic that fits our learning task is the hill-climbing algorithm, described in section 1.3.3.

Chapter 3

The experiment

In order to automate the learning of our agents, we implemented a *trainer*, which connects to the RoboCup server [2] as a coach. The trainer then gets an exact and complete view of the play-field each simulation cycle, and it also has the means to set the state of the server and the positions of the player and the ball. The objectives of the trainer are to:

1. initialize the situation
2. start an episode
3. determine when the episode is over
4. calculate the utility

The initialization of the situation is to position the players and the ball at their starting positions to match the training situation. The episode is then started by setting the state of the RoboCup server to `play_on`[1].

The episode is over when the rules for the situation is broken. E.g. if the situation is dribble, the episode is over when the player looses control of the ball i.e. when the ball is no longer within kickable distance from the player. Since the trainer gets the exact view of the play-field, i.e. the location of both the players and the ball, it is an easy task to determine when the episode is over.

When the episode is over, the trainer calculates the utility, so the hill-climbing algorithm can decide which path to take, or if a local optimum is found.

3.1 Set up

Situation As explained in section 2.3.3 , the situation we set up for the training is a 3 vs 2 keep-away soccer. The goal of the training is to learn the configuration of the electric field charges for the keepers, optimal for maximizing the possession of the ball.

[1]The RoboCup server can take several states, but it is only in `play_on` the players can act unimpededly

Charges The initial value for the keepers electric field charges are randomized within the range according to Figure 3.1(a). The takers electric field charges are established according to Figure 3.1(b).

Training boundary The size of the training ground is set to 30*30 meters.

Utility function The utility function used for the training gives the number of cycles the keepers managed to keep the possession of the ball.

Episode length The episode ends when the takers get possession of the ball, or when the ball exits the boundary for the training ground. To have a more statistical reliable utility value, each episode is run 10 times and then the mean value is used for the next step in the hill-climbing.

Object type	Value range
PositiveMate	50, 60,..., 300
NegativeMate	-300, -290,...-50
PositiveOpponent	50, 60,..., 300
NegativeOpponent	-300, -290,...-50
Ball	50, 60,..., 300
Boundary	-200, -190,...-50

(a) Charges for the keepers

Object type	Value
PositiveMate	100
NegativeMate	-100
PositiveOpponent	100
NegativeOpponent	-100
Ball	100
Boundary	-100

(b) Charges for the takers

Figure 3.1: Charges for object types during training.

3.2 Results

This section presents the results from our experiments. Results are presented for the training phase, followed by benchmarks of the trained team compared to an identical, but untrained team, as well as an external team.

3.2.1 Training phase

The hill-climbing through each training round are shown in Figure 3.2, where each drop in the curve marks the beginning of a new training round. Each training round does not have the same amount of episodes, hence the skew scale and the utility value for each episode is not shown for clarity reasons. Table 3.1 shows the minimum, maximum and average utility values for the training. Table 3.2 shows the charges for the configuration found to be the best.

	Training
Min	28
Max	235
Avg	84

Table 3.1: Utilities for training

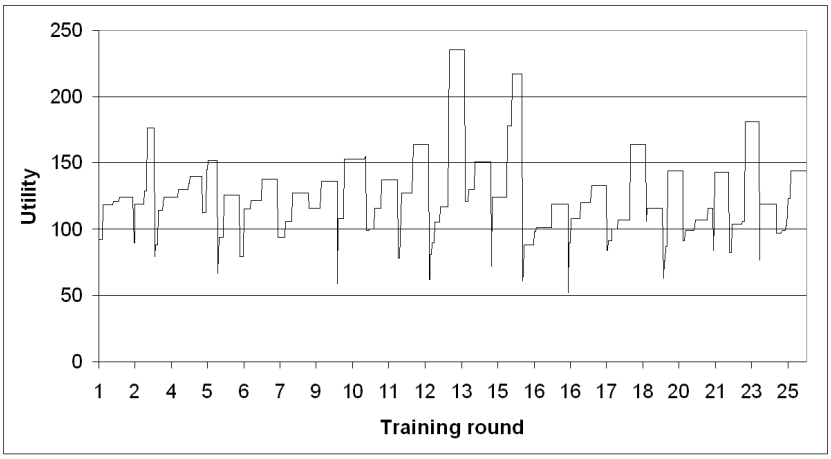

Figure 3.2: The hill-climbing through each training round.

PosMate	NegMate	PosOpponent	NegOpponent
233	-107	243	-107
Boundary	**HomePoint**	**Ball**	**Self**
-75	159	243	243

Table 3.2: Best charge configuration found during training

3.2.2 Benchmarks

The utility values for the trained team is compared to the utility values for an untrained, but otherwise identical team in Figure 3.3. In Figure 3.4 the data is filtered and shows only the values exceeding 80. This is performed in order to later be able to draw any conclusions whether the trained team has actually increased its performance in the keep-away situation.

Table 3.3 shows the minimum, maximum, average and median utility values for the trained and the untrained team, whereas Table 3.4 presents the filtered values. Table 3.5 shows the performance of the trained team compared to the untrained team and an external team, in full time matches. *Left*[2] and *Right* in the table indicates on which side of the field a team played on.

	Trained	Untrained
Min	47	43
Max	147	136
Avg	77	76
Med	75	76

Table 3.3: Utilities for trained and untrained team

The external team we have chosen to benchmark our team against is called Delicatessen [16]. It was selected because it is built upon our API, CRaPI,

[2]The first team to connect to the server plays on the left side

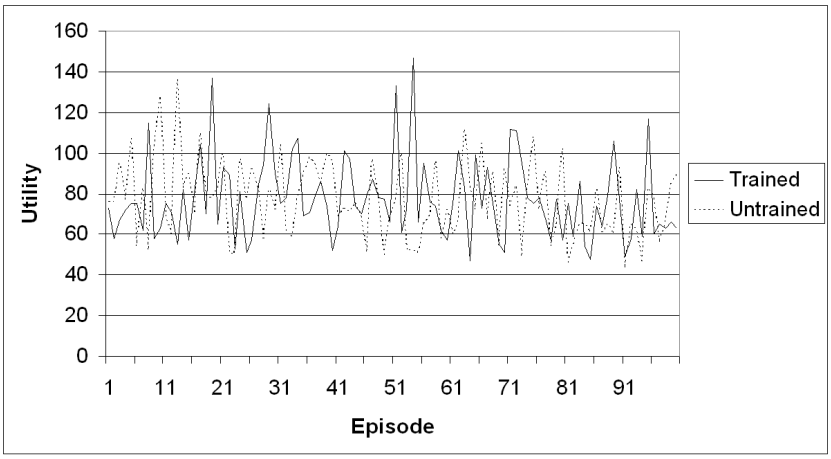

Figure 3.3: Benchmark of utilities for the keep-away situation

	Trained	**Untrained**
Min	80	81
Max	147	136
Avg	100	95
Med	95	94

Table 3.4: Filtered utilities for trained and untrained team where only values exceeding 80 are included in the calculations.

hence allowing for fair comparisons. It was also chosen because it outperformed the teams competing in 2002 years edition of the course "Agent Systems" [3].

Table 3.6 shows the ball possession in number of cycles of a full time match. A team is considered to possess the ball until the other team has it under control, i.e. a team possesses the ball during passes, even though no player is near the ball.

Left \ Right	**Untrained**	**Trained**	**Delicatessen**
Untrained	5-5	6-2	(21-0) (19-0)
Trained	5-2	(3-0) (4-3) (1-2) (3-1)	9-0
Delicatessen	0-20	(0-10) (0-11)	-

Table 3.5: Results after full time matches (6000 cycles). Multiple values are shown when matches have been played more than once.

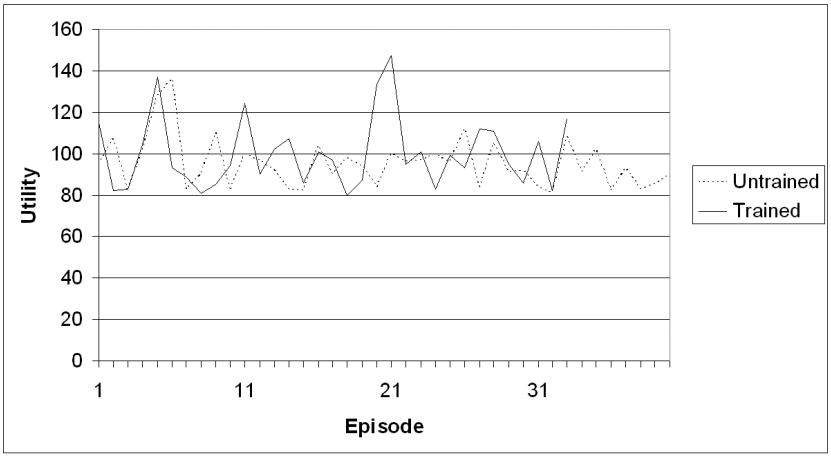

Figure 3.4: Benchmark of filtered utilities for the keep-away situation where only values exceeding 80 are included.

Left \ Right	**Untrained**	**Trained**	**Delicatessen**
Untrained	-	38%-62%	69%-31%
Trained	56%-44%	-	76%-24%
Delicatessen	32%-68%	26%-74%	-

Table 3.6: Ball possesion after full time matches (6000 cycles).

Chapter 4

Discussion

4.1 Results

4.1.1 Utilities for training

In Figure 3.2 the results for each training round are presented. They show that during each training round the utilities increases towards a local optimum. The highest local optimum is chosen as an approximation of the global optimum. The charge values for the highest local optimum, i.e. training round 13, are then returned as the result of the training.

4.1.2 Trained vs Untrained

The comparison between the untrained and the trained team, presented in Figure 3.3, shows that the performance of both teams are almost equivalent in the keep-away situation. This is supported by the results in Table 3.3, which shows that the trained team scores only slightly higher in minimum and maximum.

The strength of the charges are, according to these results, not a conclusive factor for playing good keep-away soccer. The benefit of this result is that it is easy to set up a team of electric field agents that behaves well in this situation.

The fluctuations of the values in Figure 3.3 is due to the randomized noise inherent in the world state information given from the server. Randomness is also added to the commands sent to the server. These two sources of errors have the effect that two identical actions in two perceived identical situations can produce different outcomes, and hence different utilities.

4.1.3 Game results

Even though Table 3.5 shows that both teams perform nearly equally well in the keep-away situation, Table 3.6 shows that the ball possession is clearly improved for the learned team compared to both the untrained version as well as the external team. This is due to passes being prioritized for the learned

team because passing leads to high utilities for keep-away. This is supported by the data presented in Figure 3.4 and Table 3.4, which shows higher values for maximum, average and median for the trained team.

We chose to present filtered utility values because we consider lower values to be due to the lack of well functioning behaviours and the strict rules of the training situation. Under these conditions it is the poor execution of the behaviours and not the strength of the charges that influence the outcome of the episode, resulting in misleading utility values. A drawback with this approach is that it is not obvious where to set the noise-reduction level, since it is based on subjective decisions on when the behaviours are functioning properly. The filtering could therefore lead to some of the correct values being removed. We do however not consider this to be a major issue in our case.

4.1.4 Reliability

In order to establish the reliability of the test on the training situation, we calculated the standard deviation on one third, two thirds and on the full test data, see Table 4.1. The minor difference of the deviation shows that no more test were needed on that situation.

	1/3	**2/3**	**3/3**
Trained	20.6	20.7	20.1
Untrained	21.3	19.9	19.0

Table 4.1: Standard deviation of utility values on portion of full test.

Due to lack of project time and the time needed for each test round, the testing of the ball possession was not as extensive as we would like. However, the data in Table 3.6 gives a good indication since no test results contradicted the current conclusion.

4.2 Problems

This section discusses some of the problems encountered when conducting the experiments.

4.2.1 Passing

The problems we had with the PassTheBall behaviour, described in Appendix A, was mainly due to incomplete control of the position and the motion of the the ball, making it hard to calculate exact kick power and kick direction needed in order to get the ball to the wanted position. Another problem is that it is often not enough to launch one kick command. Usually one kick is needed to get the ball to the optimal position in relation to the kicker, followed by one or more kicks to accelerate the ball to the right speed.

4.2.2 Intersecting

The behaviour IntersectBall, described in Appendix A, suffers from the same limitation as PassTheBall. Without reliable information on the current position of the ball and on the direction and the speed with which it is moving it is impossible to calculate the exact intersect point with the ball.

4.2.3 WorldModel

The problems presented with PassTheBall and IntersectBall both originate in the lack of a high precision world model. A statistical or fuzzy logic [10] based model for noise reduction would be a clear improvement over the current model.

4.2.4 Size of implementation

Even though it is not a problem in itself, a large portion of the project time had to be assigned to the making of CRaPI (section 2.1) and Yaffa (section 2.2), due to the complexity and extent of implementing a client for the simulated RoboCup domain.

Chapter 5

Conclusion

The conclusion from working with optimization of the decision level of the Yaffa player, is that it is possible to improve a RoboCup agent by applying self-learning methods to the Electric Field Approach. Above all this is shown by the substantial increase in ball-possession for the trained team.

To reduce one possible source of errors and increase the performance in the strictly controlled keep-away situation, training should preferably be conducted on the low-level skills before optimizing the decision making. The possibility to evaluate good decisions is depending on the possibility to perform them well.

5.1 Future work

The following are proposals for future work in the simulated RoboCup domain in general and CRaPI and Yaffa in particular:

- Optimize the low-level skills.
- Enhance the precision of the world model, perhaps by using a statistical or fuzzy logic based model.
- Add memory to the world model, and a model of how to adjust the confidence value for an object depending on the time since it was last perceived.
- Conduct further experiments to ascertain if left team indeed wins more games than the right team, and in that case why the phenomenon occurs.

Chapter 6

Acknowledgements

We would like to thank our advisor Stefan Johansson for his insightful comments and guidance during this project, Fredrik Heinz for providing material on world model calculations and Błażej Żak for his feed-back on CRaPI.

We would also like to thank Kenneth Henningsson for his help during the quest for an office-space, and Martin Hylerstedt and Anders Olofsson for spending time and effort to provide us with hardware.

Bibliography

[1] Mao Chen, Ehsan Foroughi, Fredrik Heinz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang, and Xiang Yin. *User Manual, RoboCup Soccer Server*, 2002.

[2] The RoboCup Federation. http://www.robocup.org/. last checked 2003-05-26.

[3] idenet: Agent Systems (DVD005). https://idenet.bth.se/, 2002. last checked 2003-05-22.

[4] Stefan Johansson and Alessandro Saffiotti. Using the electric field approach in the RoboCup domain. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, number 2377 in LNAI, pages 399–404. Springer-Verlag, Berlin, DE, 2002. Online at http://www.aass.oru.se/~asaffio/.

[5] S. Keerthi and B. Ravindran. A tutorial survey of reinforcement learning, 1995.

[6] David W. Payton, J. Kenneth Rosenblatt, and David M. Keirsey. Plan guided reaction,. *IEEE Transactions on Systems, Man and Cybernetics, No. 6, November/December 1990*, 20:1370–1382, 1990.

[7] Doina Precup, Richard S. Sutton, and Satinder P. Singh. Theoretical results on reinforcement learning with temporally abstract options. In *European Conference on Machine Learning*, pages 382–393, 1998.

[8] M. Riedmiller, A. Merke, D. Meier, A. Hoffmann, A. Sinner, O. Thate, and R. Ehrmann. Karlsruhe brainstormers — A reinforcement learning approach to robotic soccer. *Lecture Notes in Computer Science*, 2019:367–??, 2001.

[9] Stuart Russel and Peter Norvig. *Artificial Intelligence a Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey 07458, 1995.

[10] A. Saffiotti. The uses of fuzzy logic for autonomous robot navigation: a catalogue raisonn'e, 1997.

[11] Stefan Schaal. Learning from demonstration. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 1040. The MIT Press, 1997.

[12] Peter Stone and Richard S. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *Proc. 18th International Conf. on Machine Learning*, pages 537–544. Morgan Kaufmann, San Francisco, CA, 2001.

[13] Peter Stone and Manuela M. Veloso. Layered learning. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings*, volume 1810, pages 369–381. Springer, Berlin, 2000.

[14] Team Yaffa. http://crapi.sourceforge.net/. last checked 2003-05-24.

[15] Cai Yunpeng Yao Jinyi, Chen Jiang and Li Shi. Architecture of tsinghuaeolus. Technical report, State Key Lab of Intelligent Technology and System, Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, P.R.China, 2001.

[16] Błażej Żak. Delicatessen - a robocup team. http://blaise.wha.la/robocup/delicatessen.pdf, 2003.

Appendix A

Behaviours in Yaffa

This appendix describes the behaviours we have used during our experiment. Mate and Opponent charges can either be positive or negative. The charges for the Ball, the HomePoint and the OpponentGoal are always positive, whereas the TeamGoal and the Boundary charges are always negative.

Offensive and Defensive are special purpose fields, with linear spread functions. The Offensive field has a negative charge in the team goal and a positive charge in the opponent goal, and the opposite for the Defensive field.

BlockOpponent Interfere an opponent when the other team is in possession of the ball.

Charges Mate[-], Opponent[+], Ball, HomePoint, Defensive.

Probes Predicted positions of the player, next to the blockable opponents.

DribbleWithBall Advance without losing control of the ball.

Charges Mate[+], Opponent[-], Self, Offensive, TeamGoal, OpponentGoal, Boundary.

Probes A set of dribble points on the offensive side of the player, which are predicted possible positions of the ball, when the behaviour is done executing.

GetPassable Make sure to be positioned where there is a free pass-path from the ball-keeper, when the ball is in our possession.

Charges Opponent[-], Ball, HomePoint, Offensive, Boundary.

Probes Predicted possible positions of the player, when the behaviour is done executing.

IntersectBall Run to the closest possible intersection-point with the ball-trajectory.

Charges Mate[-], Opponent[+], Ball, HomePoint, Defensive.

Probes Predicted position of the player next to the ball.

LocalizeBall *This behaviour is currently not incorporated in the EFA-model.* It localizes and focuses on the ball. Chosen for execution when no other behaviour is applicable.

Charges None

Probes None

LocalizeMate *This behaviour is currently not incorporated in the EFA-model.* It localizes and focuses on the last seen team mate. Chosen for execution when no team mate is seen and no other behaviour than LocalizeBall is applicable.

Charges None

Probes None

PassTheBall Passes the ball to a team mate.

Charges Mate[+], Opponent[-], Self, Offensive, TeamGoal, OpponentGoal, Boundary.

Probes Predicted position of the ball when the behaviour is done executing.

RunHome Runs to the players home-position.

Charges Mate[-], Opponent[+], Ball, HomePoint, Offensive.

Probes The player at the home-position.