Users Manual

RoboCup Soccer Server
for Soccer Server Version 7.07 and later

Mao Chen†, Klaus Dorer, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang†, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang† and Xiang Yin†

January 22, 2003

† <ustc9811@sina.com>
<klaus.dorer@living-systems.de>
<foroughi@ce.sharif.edu>
<frehe@ida.liu.se>
<spiros@cs.york.ac.uk>
<kkosti@essex.ac.uk>
<johan.kummeneje@generalwireless.se>
<murray@uni-koblenz.de>
<noda@etl.go.jp>
<fruit@uni-koblenz.de>
<pfr+@cs.cmu.edu>
<timosteffens@gmx.de>
Acknowledgements

We are very grateful for the work of the authors from the previous versions of the manual that could not help out on this version:

- *David Andre* at Berkeley, University of California, USA.
- *Pascal Gugenberger* at Humboldt University, Berlin, Germany.
- *Marius Joldos* at Technical University of Cluj-Napoca, Romania.
- *Paul Arthur Navratil* at University of Texas at Austin, USA.
- *Peter Stone* at University of Texas at Austin, USA.
- *Tomoichi Takahashi* at Chubu University, Japan.
- *Tralvex Yeap* at KRDL, Singapore.
- *Emiel Corten* at University of Amsterdam, Netherlands.
- *Helmut Myritz* at Humboldt University, Germany.
- *Jukka Riekki* at Oulu University, Finland.

Besides the authors, we would also like to thank Stefan Sablatnög from the University of Ulm, Germany, and Mike Hewett from University of Texas at Austin, USA, for a thorough proofreading of the soccermanual 4.00. We have also received a lot of good suggestions from Erik Mattsson at the University of Karlskrona/Ronneby, Sweden.

We would not have been able to do this manual without the above mentioned people.¹

¹The persons listed on the title page are the persons responsible for the different sections of the manual.
Acknowledgements
Contents

Acknowledgements i

1. Introduction 1
   1.1. Background ......................................................... 1
   1.2. The Goals of RoboCup ............................................. 2
      1.2.1. Simulated League ........................................... 2
      1.2.2. What is the Soccerserver ................................. 3
   1.3. History ........................................................... 4
      1.3.1. History of the Soccer Server ................................. 4
      1.3.2. History of the RoboCup Simulation League ................. 5
      1.3.3. History of the Soccer Manual Effort ..................... 8
   1.4. About This Manual ............................................... 8
   1.5. Reader’s Guide to the Manual .................................. 8

2. Overview 11
   2.1. Getting Started .................................................. 11
      2.1.1. The Server .................................................. 11
      2.1.2. The Monitor .................................................. 11
      2.1.3. The Logplayer ................................................. 12
      2.1.4. The Demo Client .............................................. 12
   2.2. The Rules of the Game .......................................... 13
      2.2.1. Rules Judged by the Automated Referee ................... 13
      2.2.2. Rules Judged by the Human Referee ....................... 15

3. Getting Started 17
   3.1. Getting and installing the server ............................. 17
   3.2. Download sites .................................................. 19
   3.3. How to start the server ........................................ 20
   3.4. How to stop the server ......................................... 21
   3.5. Supported platforms ............................................ 22
   3.6. The process of a match ......................................... 22

4. Soccer Server 23
   4.1. Objects .......................................................... 23
   4.2. Protocols ........................................................ 24
## Contents

4.2.1. Client Command Protocol ................................. 24  
4.2.2. Client Sensor Protocol .................................. 27  
4.3. Sensor Models .................................................. 28  
  4.3.1. Aural Sensor Model ...................................... 28  
  4.3.2. Vision Sensor Model .................................... 29  
  4.3.3. Body Sensor Model ...................................... 35  
4.4. Movement Model ............................................... 36  
  4.4.1. Movement Noise Model .................................. 36  
  4.4.2. Collision Model ......................................... 37  
4.5. Action Models .................................................. 37  
  4.5.1. Catch Model .............................................. 37  
  4.5.2. Dash Model (incl. stamina model) ....................... 38  
  4.5.3. Kick Model ............................................... 39  
  4.5.4. Move Model .............................................. 43  
  4.5.5. Say Model ................................................ 43  
  4.5.6. Turn Model .............................................. 45  
  4.5.7. TurnNeck Model ......................................... 45  
4.6. Heterogeneous Players ....................................... 46  
4.7. Referee Model .................................................. 47  
  4.7.1. Play Modes and referee messages ....................... 47  
4.8. The Soccer Simulation ....................................... 48  
  4.8.1. Description of the simulation algorithm .................. 48  
4.9. Using Soccerserver .......................................... 49  
  4.9.1. The Soccerserver Parameters ............................ 49  

5. The Soccer Monitor ............................................... 53  
  5.1. Introduction .................................................. 53  
  5.2. Getting started ............................................... 53  
  5.3. Communication from Server to Monitor ..................... 53  
    5.3.1. Version 1 ............................................... 53  
    5.3.2. Version 2 ............................................... 56  
  5.4. Communication from Monitor to Server ..................... 57  
  5.5. How to record and playback a game ........................ 58  
    5.5.1. Version 1 Protocol ..................................... 59  
    5.5.2. Version 2 Protocol ..................................... 59  
    5.5.3. Version 3 Protocol ..................................... 60  
  5.6. Settings and Parameters .................................... 61  
  5.7. What’s New .................................................... 61  

6. Soccer Client ..................................................... 65  
  6.1. Protocols ...................................................... 65  
    6.1.1. Initialization and Reconnection ......................... 65  
    6.1.2. Control Commands ....................................... 67  
    6.1.3. Sensor Information ..................................... 69
## Contents

6.2. How to Create Clients ........................................ 70  
   6.2.1. Sample Client ........................................ 71  
   6.2.2. Simple Clients ....................................... 73  
   6.2.3. Tips .................................................. 77  

7. The coach ..................................................... 79  
   7.1. Introduction ............................................. 79  
   7.2. Distinction between trainer and online coach ............... 79  
   7.3. Trainer ................................................ 80  
      7.3.1. Connecting with and without the soccerver server referee . 80  
   7.4. Commands .............................................. 80  
      7.4.1. Commands that can be used only by the trainer .......... 80  
      7.4.2. Commands that can also be used by the online coach with certain restrictions ............... 83  
      7.4.3. Commands that can be used by both trainer and online-coach ... 85  
      7.4.4. Commands that can be used only by the online-coach ....... 86  
   7.5. Messages from the server .................................. 86  
   7.6. Online coach ........................................... 87  
      7.6.1. Introduction ....................................... 87  
      7.6.2. Communication with the players ....................... 88  
      7.6.3. Changing player types ............................ 89  
   7.7. The standard coach language ................................ 89  
      7.7.1. General properties .................................. 89  
      7.7.2. Example Language Utterance .......................... 90  
      7.7.3. Overview of the five message types ................... 91  
      7.7.4. Defining rules ...................................... 92  
      7.7.5. Semantics and syntax details of the components ........ 93  
      7.7.6. Further resources ................................... 98  
      7.7.7. Syntax ............................................. 98  
   7.8. Trainer commands ......................................... 102  
   7.9. Coach commands .......................................... 102  
   7.10. Shared Trainer and Online Coach Interactions with the Server ...... 105  

8. References and Further Reading .................................. 107  
   8.1. General papers .......................................... 107  
   8.2. Doctoral Theses ......................................... 108  
   8.3. Undergraduate and Master’s Theses ......................... 108  
   8.4. Platforms to start building team upon ..................... 109  
   8.5. Education-related articles .............................. 109  
   8.6. Machine Learning ....................................... 109  
   8.7. Decision Making ........................................ 109  
   8.8. Other supporting documents ............................. 109  
   8.9. Team Descriptions ..................................... 110  
      8.9.1. 1996 .............................................. 110
Contents
1. Introduction

We are in the early days of RoboCup [7], with half a century to go before we can “...build a team of robot soccer players, which can beat a human world cup champion team” ([6], p. v). The challenge posed by the goal is enormous and inspires hundreds of researchers yearly throughout the world to engage themselves and their students in RoboCup. RoboCup has been used as a research challenge in parallel with a usage for educational purposes, and to stimulate the interest of the public for robotics and artificial intelligence (AI). Each year since 1997, researchers from different countries have gathered to play the world cup. The event has drawn an increasing amount of interest from the public, as robotics is still not commonplace.

The intention of this manual is to guide the developers of simulated league teams in the beginning steps, and also serve as a reference manual for the experienced users.

1.1. Background

Mackworth [11] introduced the idea of using soccer-playing robots in research. 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 programme, called Robot J-League. 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. [7] proposed the first Robot World Cup Soccer Games and Conferences to take place in 1997. 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. RoboCup differs from previous research in AI by focusing on a distributed solution instead of a centralised 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

---

1Parts of this chapter is taken directly from [18]
2The J-League is the professional soccer league in Japan.
1. Introduction

the body of researchers is quite large and widespread, local committees are formed to promote RoboCup-related events in their geographical area.

1.2. The Goals of RoboCup

The RoboCup Federation has set goals and a timetable for the research. Setting goals and a timetable are means of pushing the state-of-the-art further, in conjunction with formalised test-beds. In resemblance with John F. Kennedy’s national goal of “landing a man on the moon and returning him safely to earth” ([4], p. 8276), the main accomplishment was not to land a man on the moon and returning him safely, but the overall technological advancement. Therefore, the most important goal of RoboCup is to advance the overall technological level of society, and as a more pragmatic goal to achieve the following:

By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA\(^4\), against the winner of the most recent World Cup [15].

There will be several technological advancements, even if the goal of the robotic soccer team is not reached, starting with Team-Partitioned, Opaque-Transition Reinforcement Learning (TPOT-RL) [19] which has found application in the domain of packet routing in computer networks. TPOT-RL is a distributed learning method in domains where “agents have limited information about environmental state transitions” ([19], p. 22).

In most RoboCup leagues, the teams consist of either robots or programs that cooperate in order to defeat the opponent team. RoboCup Rescue and the commentator exhibition diverge from the other RoboCup leagues. The goal of defeating an opponent would raise ethical issues in RoboCup Rescue, since we cannot assign comparable utilities to human lives and buildings. Hence, the focus in RoboCup Rescue is on the co-operative efforts between heterogeneous agents. In the commentator exhibition, the goal is to observe and comment.

Besides the commentator exhibition and RoboCup Rescue, the main body of the RoboCup challenge consists of several leagues for soccer playing. However, as this manual is about the simulated league we will only focus on it.

1.2.1. Simulated League

The RoboCup simulator league is based on the RoboCup simulator called the soccer server [13], a physical soccer simulation system. All games are visualised by displaying the field of the simulator by the soccer monitor on a computer screen. 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

\(^4\)Fédération Internationale de Football Association (FIFA) defines the rules of soccer [27].
researchers from having to handle robot problems such as object recognition [9], 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 mechanically be 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. All participating teams are also obliged to play according to a gentlemen’s agreement, e.g., not to use loopholes.

Since the first version of the soccer server was completed in 1995, there have been four world cups and one pre-world cup event, not to mention all other RoboCup-related events. The 1996 pre-RoboCup event [5] was held in Osaka, with only seven entrants in the competition which ended with a Japanese victory by the team Ogalets from Tokyo University. In Nagoya the following year, the first formal competition was held in conjunction with the IJCAI’97 conference. The competition had 29 teams participating, and the winner was AT Humboldt [2]. The RoboCup world cup of 1998 was played in conjunction with the human world cup in Paris, and the winning team was CMUnited98 [29]. During the world cup, media was heavily covering the event, as it was public in a museum in the suburbs of Paris. The year after, the world cup was held in conjunction with IJCAI’99 in Stockholm, and the winners (once again) were CMUnited99 [30]. An unchanged version of the champion team must participate, as a benchmark, in the next world cup. The benchmarking teams have always been able to win their group, but only in 2000 did the benchmark team advance further than the first game after group play.

1.2.2. What is the Soccerserver

Soccer Server is a system that enables autonomous agents consisting of programs written in various languages to play a match of soccer (association football) against each other.

A match is carried out in a client/server style: A server, soccerserver, provides a virtual field and simulates all movements of a ball and players. Each client controls movements of one player. Communication between the server and each client is done via UDP/IP sockets. Therefore users can use any kind of programing systems that have UDP/IP facilities.

The soccerserver consists of 2 programs, soccerserver and soccermonitor. Soccer Server is a server program that simulates movements of a ball and players, communicates with clients, and controls a game according to rules. Soccermonitor is a program that displays the virtual field from the soccerserver on the monitor using the X window system. A number of soccermonitor programs can connect with one soccerserver, so we can display field-windows on multiple displays.

A client connects with soccerserver by an UDP socket. Using the socket, the client sends commands to control a player of the client and receives information from sensors of the player. In other words, a client program is a brain of the player: The client receives visual and auditory sensor information from the server, and sends control-commands to
1. Introduction

The server.

Each client can control only one player\textsuperscript{56}. So a team consists of the same number of clients as players. Communications between the clients must be done via soccerserver using say and hear protocols. (See section 4.2.1.) One of the purposes of soccerserver is evaluation of multi-agent systems, in which efficiency of communication between agents is one of the criteria. Users must realize control of multiple clients by such restricted communication.

1.3. History

In this section we will first describe the history of the soccerserver and thereafter the history of the RoboCup Simulation League. To end the section we will also describe the history of the manual effort.

1.3.1. History of the Soccer Server

The first, preliminary, original system of soccerserver was written in September of 1993 by Itsuki Noda, ETL. This system was built as a library module for demonstration of a programming language called MWP, a kind of Prolog system that has multi-threads and high level program manipulation. The module was a closed system and displayed a field on a character display, that is VT100.

The first version (version 0) of the client-server style server was written in July of 1994 on a LISP system. The server shows the field on an X window system, but each player was shown in an alphabet character. It used the TCP/IP protocol for connections with clients. This LISP version of soccerserver became the original style of the current soccerserver. Therefore, the current soccerserver uses S-expressions for the protocol between clients and the server.

The LISP version of soccerserver was re-written in C++ in August of 1995 (version 1). This version was announced at the IJCAI workshop on Entertainment and AI/Alife held in Montreal, Canada, August 1995.

The development of version 2 started January of 1996 in order to provide the official server of preRoboCup-96 held at Osaka, Japan, November 1996. From this version, the system is divided into two modules, soccerserver and soccerdisplay (currently, soccer-monitor). Moreover, the feature of coach mode was introduced into the system. These two features enabled researchers on machine learning to execute games automatically. Peter Stone at Carnegie Mellon University joined the decision-making process for the development of the soccerserver at this stage. For example, he created the configuration files that were used at preRoboCup-96.

After preRoboCup-96, the development of the official server for the first RoboCup, RoboCup-97 held at Nagoya, Japan, August 1997, started immediately, and the version
\textsuperscript{5}Technically, it is easy to cheat the server. Therefore this is a gentleman's agreement.
\textsuperscript{6}In order to test various kinds of systems, we may permit a client to control multiple players if the different control modules of players are separated logically from each other in the client.
1.3. History

3 was announced in February of 1997. Simon Ch’ng at RMIT joined decisions of regulations of soccerserver from this stage. The following features were added into the new version:

- logplayer
- information about movement of seen objects in visual information
- capacity of hearing messages

The development of version 4 started after RoboCup-97, and announced November 1997. From this version, the regulations are discussed on the mailing list organized by Gal Kaminka. As a result, many contributors joined the development. Version 4 had the following new features:

- more realistic stamina model
- goalie
- handling offside rule
- disabling players for evaluation
- facing direction of players in visual information
- sense_body command

Version 4 was used in Japan Open 98, RoboCup-98 and Pacific Rim Series 98.

Version 5 was used in Japan Open 99, and will also be used in RoboCup-99 in Stockholm during the summer of 1999.

In Melbourne 2000, version 6 was used, and for the world cup in 2001 version 7 will be used.

1.3.2. History of the RoboCup Simulation League

The RoboCup simulation league has had five main official events: preRoboCup-96, RoboCup-97, RoboCup-98, RoboCup-99, and RoboCup 2000. Research results have been reported extensively in the proceedings of the workshops and conferences associated with these competitions. In this section, we focus mainly on the competitions themselves.

preRoboCup-96

preRoboCup-96 was the first robotic soccer competition of any sort. It was held on November 5–7, 1996 in Osaka, Japan [5]. In conjunction with the IROS-96 conference, preRoboCup-96 was meant as an informal, small-scale competition to test the RoboCup soccerserver in preparation for RoboCup-97. 5 of the 7 entrants were from the Tokyo region. The other 2 were from Ch’ng at RMIT and Stone and Veloso from CMU.

The winning teams were entered by:
1. Introduction

1. Ogawara (Tokyo University)
2. Sekine (Tokyo Institute of Technology)
3. Inoue (Waseda University)
4. Stone and Veloso (Carnegie Mellon University)

In this tournament, team strategies were generally quite straightforward. Most of the teams kept players in fixed locations, only moving them towards the ball when it was nearby.

RoboCup-97

The RoboCup-97 simulator competition was the first formal simulated robotic soccer competition. It was held on August 23–29, 1997 in Nagoya, Japan in conjunction with the IJCAI-97 conference [6]. With 29 teams entering from all around the world, it was a very successful tournament.

The winning teams were entered by:

1. Burkhard et al. (Humboldt University)
2. Andou (Tokyo Institute of Technology)
3. Tambe et al. (ISI/University of Southern California)
4. Stone and Veloso (Carnegie Mellon University)

In this competition, the champion team exhibited clearly superior low-level skills. One of its main advantages in this regard was its ability to kick the ball harder than any other team. Its players did so by kicking the ball around themselves, continually increasing its velocity so that it ended up moving towards the goal faster than was imagined possible. Since the soccerserver did not (at that time) enforce a maximum ball speed, a property that was changed immediately after the competition, the ball could move arbitrarily fast, making it almost impossible to stop. With this advantage at the low-level behavior level, no team, regardless of how strategically sophisticated, was able to defeat the eventual champion.

At RoboCup-97, the RoboCup scientific challenge award was introduced. Its purpose is to recognize scientific research results regardless of performance in the competitions. The 1997 award went to Sean Luke [10] of the University of Maryland “for demonstrating the utility of evolutionary approach by co-evolving soccer teams in the simulator league.”

RoboCup-98

The second international RoboCup championship, RoboCup-98, was held on July 2–9, 1998 in Paris, France [1]. It was held in conjunction with the ICMAS-98 conference.

The winning teams were entered by:
1.3. History

Unlike in the previous year’s competition, there was no team that exhibited a clear superiority in terms of low-level agent skills. Games among the top three teams were all quite closely contested with the differences being most noticeable at the strategic, team levels.

One interesting result at this competition was that the previous year’s champion team competed with minimal modifications and finished roughly in the middle of the final standings. Thus, there was evidence that as a whole, the field of entries was much stronger than during the previous year: roughly half the teams could beat the previous champion.

The 1998 scientific challenge award was shared by Electro Technical Laboratory (ETL), Sony Computer Science Laboratories, Inc., and German Research Center for Artificial Intelligence GmbH (DFKI) for ”development of fully automatic commentator systems for RoboCup simulator league.”

To encourage the transfer of results from RoboCup to the scientific community at large, RoboCup-98 was the first to host the Multi-Agent Scientific Evaluation Session. 13 different teams participated in the session, in which their adaptability to loss of team members was evaluated comparatively. Each team was played against the same fixed opponent (the 1997 winner, AT Humboldt’97) four half-games under official RoboCup rules. The first half-game (phase A) served as a base-line. In the other three half-games (phases B-D), 3 players were disabled incrementally: A randomly chosen player, a player chosen by the representative of the fixed opponent to maximize ”damage” to the evaluated team, and the goalie. The idea is that a more adaptive team would be able to respond better to these.

Very early on, even during the session itself, it became clear that while in fact most participants agreed intuitively with the evaluation protocol, it wasn’t clear how to quantitatively, or even qualitatively, analyse the data. The most obvious measure of the goal-difference at the end of each half may not be sufficient: some teams seem to do better with less players, some do worse. Performance, as measured by the goal-difference, really varied not only from team to team, but also for the same team between phases. The evaluation methodology itself and analysis of the results became open research problems in themselves. To facilitate this line of research, the data from the evaluation was made public at: http://www.isi.edu/~galk/Eval/

RoboCup-99

The third international RoboCup championship, RoboCup-99, was held in late July and early August, 1999 in Stockholm, Sweden [3]. It was held in conjunction with the IJCAI-99 conference.
1. Introduction

RoboCup 2000
The fourth international RoboCup championship, RoboCup 2000, was held in early September, 2000 in Melbourne, Australia [16]. It was held in conjunction with the PRICAI-2000 conference.

1.3.3. History of the Soccer Manual Effort
The first versions of the manual were written by Itsuki Noda, while developing the soccerserver, and around version 3.00 there were several requests on an updated manual, to better correspond to the server as well as enable newcomers to more easily participate in the RoboCup World Cup Initiative. In the fall of 1998 Peter Stone initiated the Soccer Manual Effort, over which Johan Kummeneje took responsibility to organize and as a result the Soccer Server Manual version 4.0 was released on the 1st of November 1998.

In 1999, the manual for the soccerserver version 5.0 was released. Unfortunately the manual lost part of its pace, and there was no release of the manual for soccerserver version 6.0.

Since 1999, the soccerserver has changed major version to 7 and is continuously developed. Therefore the Soccer Manual Effort has developed a new version, which you are currently reading.

1.4. About This Manual
This manual is the joint effort of the authors from a diverse range of universities and organizations, which build upon the original work of Itsuki Noda.
If there are errors, inconsistencies, or oddities, please notify johank@dsv.su.se or fruit@uni-koblenz.de with the location of the error and a suggestion of how it should be corrected.

We are always looking for anyone who has an idea on how to improve the manual, as well as proofread or (re)write a section of the manual. If you have any ideas, or feel that you can contribute with anything to the SoccerServer Manual Effort please mail johank@dsv.su.se or fruit@uni-koblenz.de.

The latest manual can be downloaded at http://www.dsv.su.se/~johank/RoboCup/manual.

1.5. Reader’s Guide to the Manual
The thesis is written for a wide range of readers, and therefore the chapters are not equally important to all readers. We shortly describe the remaining chapters to give an overview of the thesis.

Chapter 2 introduces the concepts of the simulated league and will help the newcomer to get to terms with the different parts.

Chapter 3 helps the beginners to start compiling and running the software.

Chapter 4 describes the soccerserver.
1.5. Reader’s Guide to the Manual

Chapter 5 describes the soccermonitor.
Chapter 6 describes the soccercient and how to create one.
Chapter 7 describes the coachclient.
Chapter 8 suggests some further reading.
1. Introduction
2. Overview

2.1. Getting Started

This section is designed to give you a quick introduction to the main components of the RoboCup simulator. For each of these components you will find detailed information (i.e. configuration parameters, run-time options, etc.) later on in this manual.

2.1.1. The Server

The server is a system that enables various teams to compete in a game of soccer. Since the match is carried out in a client-server style, there are no restrictions as to how teams are built. The only requirement is that the tools used to develop a team support client-server communication via UDP/IP. This is due to the fact that all communication between the server and each client is done via UDP/IP sockets. Each client is a separate process and connects to the server through a specified port. After a player connects to the server, all messages are transferred through this port. A team can have up to 12 clients, i.e. 11 players (10 fielders + 1 goalie) and a coach. The players send requests to the server regarding the actions they want to perform (e.g. kick the ball, turn, run, etc.).

The server receives those messages, handles the requests, and updates the environment accordingly. In addition, the server provides all players with sensory information (e.g. visual data regarding the position of objects on the field, or data about the player’s resources like stamina or speed). It is important to mention that the server is a real-time system working with discrete time intervals (or cycles). Each cycle has a specified duration, and actions that need to be executed in a given cycle, must arrive at the server during the right interval. Therefore, slow performance of a player that results in missing action opportunities has a major impact on the performance of the team as a whole. A detailed description of the server can be found in Chapter 4.

2.1.2. The Monitor

The Soccer Monitor is a visualisation tool that allows people to see what is happening within the server during a game. Currently the monitor comes in two flavors, the rcssmonitor and the rcssmonitor_classic. The information shown on both monitors include the score, team names, and the positions of all the players and the ball. They also provide simple interfaces to the server. For example, when both teams have connected, the "Kick-Off" button on the monitor allows a human referee to start the
2. Overview

game. The **rcssmonitor**, which is based on the **frameview** by Artur Merke, extends the functionality of the classic monitor by several features.

- It is possible to zoom into areas of the field. This is especially useful for debugging purposes.

- The current positions and velocities of all players and the ball can be printed to the console at any time.

- A variety of information can be shown on the monitor, e.g. a player’s view cone, stamina or (in the case of heterogeneous players) player type.

- Players and the ball can be moved around with the mouse.

As you will discover later on, to run a game on the server, a monitor is not required. However, if needed, a number of monitors can be connected to the server at the same time (for example if you want to show the same game at different terminals). For further details on the monitor please have a look at Chapter 5.

### 2.1.3. The Logplayer

The logplayer can be thought of as a video player. It is a tool that is used to replay matches. When running the server, certain options can be used that will cause the server to store all the data for a given match on the hard drive. (Pretty much like pressing the record button on your video). Then, the program **rcsslogplayer** combined with a monitor can be used to replay that game as many times as needed. This is quite useful for doing team analysis and discovering the strong or weak points of a team. Much like a video player, the logplayer is equipped with play, stop, fast forward and rewind buttons. Also the logplayer allows you to jump to a particular cycle in a game (for example if you only want to see the goals). Finally the logplayer allows you to edit existing recordings, i.e. you can save interesting scenes from a match (or several matches) to another log file and thus create a presentation easily.

The logplayer can be controlled via a small GUI or a command line interface. In addition commands can be read from a file, which adds limited scripting capabilities to the logplayer.

### 2.1.4. The Demo Client

Bundled with the RoboCup Soccer Simulator is a program called **rcssclient**, which implements a very primitive textbased client for the simulation. The purpose of this program is to give you a first idea of how the whole affair works.

When **rcssclient** is started, it connects to the server. You are presented with a simple ncurses-based interface. You can then enter commands that are executed by the server. Any information that is received by the client will be shown in a different section of the screen according to its type (**visual**, **sense body** or **other**). By entering commands and see what happens you can get a first idea of the way things work in the simulation.
2.2. The Rules of the Game

During a game, a number of rules are enforced either by the automated referee within the server, or by a human referee. The aim of this section is to describe how these rules work, and how they affect the game.

2.2.1. Rules Judged by the Automated Referee

Kick-Off

Just before a kick off (either before a half time starts, or after a goal), all players must be in their own half. To allow for this to happen, after a goal is scored, the referee suspends the match for an interval of 5 seconds. During this interval, players can use the `move` command to teleport to a position within its own side, rather than run to this position, which is much slower and consumes stamina. If a player remains in the opponent half after the 5-second interval has expired or tries to teleport there during the interval, the referee moves the player to a random position within their own half.

Goal

When a team scores, the referee performs a number of tasks. Initially, it announces the goal by broadcasting a message to all players. It also updates the score, moves the ball to the centre mark, and changes the play-mode to kick_off_x (where x is either left or right). Finally, it suspends the match for 5 seconds allowing players to move back to their own half (as described above in the "Kick-Off" section).

Out of Field

When the ball goes out of the field, the referee moves the ball to a proper position (a touchline, corner or goal-area) and changes the play-mode to kick_in, corner_kick, or goal_kick. In the case of a corner kick, the referee places the ball at (1m, 1m) inside the appropriate corner of the field.

Player Clearance

When the play-mode is kick_off, free_kick, kick_in, or corner_kick, the referee removes all defending players located within a circle centred on the ball. The radius of this circle is a parameter within the server (normally 9.15 meters). The removed players are placed on the perimeter of that circle. When the play-mode is offside, all offending players are moved back to a non-offside position. Offending players in this case are all players in the offside area and all players inside a circle with radius 9.15 meters from the ball. When the play-mode is goal_kick, all offending players are moved outside the penalty area.
offending players cannot re-enter the penalty area while the goal kick takes place. The play-mode changes to play on immediately after the ball goes outside the penalty area.

**Play-Mode Control**

When the play-mode is kick off, free kick, kick in, or corner kick, the referee changes the play-mode to play on immediately after the ball starts moving through a kick command.

**Offside**

A player is marked offside, if it is
- in the opponent half of the field,
- closer to the opponent goal than at least two defending players,
- closer to the opponent goal than the ball,
- closer to the ball than 2.5 meters (this can be changed with the server parameter offside-active-area-size).

**Backpasses**

Just like in real soccer games, the goalie is not allowed to catch a ball that was passed to him by a teammate. If this happens, the referee calls a back pass l or back pass r and assigns a free kick to the opposing team. As such a back pass can only happen within the penalty area, the ball is placed on the corner of the penalty area that is closest to the position the goalie tried to catch. Note, that it is perfectly legal to pass the ball to the goalie if the goalie does not try to catch the ball.

**Free Kick Faults**

When taking a free kick, corner kick, goalie free kick, or kick in, a player is not allowed to pass the ball to itself. If a player kicks the ball again after performing one of those free kicks, the referee calls a free kick fault l or free kick fault r and the opposing team is awarded a free kick.

As a player may have to kick the ball more than once in order to accelerate it to the desired speed, a free kick fault is only called if the player taking the free kick

1. is the first player to kick the ball again, and
2. the player has moved (= dashed) between the kicks.

So issuing command sequences like kick kick dash or kick turn kick is perfectly legal. The sequence kick dash kick, on the other hand, results in a free kick fault.
2.2. The Rules of the Game

Half-Time and Time-Up

The referee suspends the match when the first or the second half finishes. The default length for each half is 3000 simulation cycles (about 5 minutes). If the match is drawn after the second half, the match is extended. Extra time continues until a goal is scored. The team that scores the first goal in extra time wins the game. This is also known as the “golden goal” rule or “sudden death”.

2.2.2. Rules Judged by the Human Referee

Fouls like “obstruction” are difficult to judge automatically because they concern players’ intentions. To resolve such situations, the server provides an interface for human-intervention. This way, a human-referee can suspend the match and give free kicks to either of the teams. The following are the guidelines that were agreed prior to the RoboCup 2000 competition, but they have been used since then.

- Surrounding the ball
- Blocking the goal with too many players
- Not putting the ball into play after a given number of cycles.
  By now this rule is handled by the automatic referee, as well. If a team fails to put the ball back into play for drop_ball_time cycles, a drop_ball is issued by the referee. However, if a team repeatedly fails to put the ball into play, the human referee may drop the ball prematurely.
- Intentionally blocking the movement of other players
- Abusing the goalie catch command (the goalie may not repeatedly kick and catch the ball, as this provides a safe way to move the ball anywhere within the penalty area).
- Flooding the Server with Messages
  A player should not send more than 3 or 4 commands per simulation cycle to the soccer server. Abuse may be checked if the server is jammed, or upon request after a game.
- Inappropriate Behaviour
  If a player is observed to interfere with the match in an inappropriate way, the human-referee can suspend the match and give a free kick to the opposite team.
2. Overview
3. Getting Started

This section contains all the information necessary to get the Soccer Server source files and to install the software. The procedure shown was performed on a computer running GNU/Linux 2.2.17 (check your version with `uname -sr`) with egcs 2.91.66 (check your version with `which g++`) but any reasonably up-to-date installation should do it. In the commands shown below, `->` is supposed to be the command-line prompt.

3.1. Getting and installing the server

1. Get source files from one of the Soccer Server sites:

   - [http://www.robolog.org/server/](http://www.robolog.org/server/) (Germany)

   This is done by:
   
   a) clicking the Download link and
   
   b) getting the gzipped file `sserver-*.*.tar.gz`

   where * is the version number\(^1\) of the software. This file contains all the necessary sources for Soccer Server and can be found either by going directly to the Soccer Server FTP site at [http://www.uni-koblenz.de/ag-ki/ROBOCUP/sserver/pub/soccer/server/](http://www.uni-koblenz.de/ag-ki/ROBOCUP/sserver/pub/soccer/server/) or by clicking the Newest version link from the download page.

2. Extract the source files by running:

   -> tar zxvf sserver-*.*.tar.gz

   A directory `sserver-*` is created. Now, change the working directory to `sserver-*`. This directory contains the following files:

   -> ls -a

   . Makefile logplay sserver
   .. README logplayer sserver-csh.tmpl
   .cvsignore coach_lang_grammar monitor sserver-tcsh.tmpl

\(^1\)Version 7.04 is the current version at the time of writing (March 2001). The example list is output for version 7.04.
3. Getting Started

Acknowledgement configure recfile_change sserver.org
Changes configure2 sampleclient sserver.tmpl
Licence drawcheck server tools

Always read the README file first:

-> cd sserver-7.04
-> more README

[Directories]
server/
    Source files of soccerserver
monitor/
    Source files of soccermonitor
sampleclient/
    Source files of a very simple client

[How to Make]
(1) Do configure
(2) Do make

[How to Start]
(1) start soccerserver.
(2) start a couple of soccermonitors you want.

"sserver" is a sample script to invoke soccerserver and soccermonitor.

[Required Softwares]
GCC 2.7.0 or later
X11R5 or R6
    Some old version of R5 may cause problems of display.

[Suppoted OSs]
SunOS 4.1.x
Solaris 2.x
DEC OSF1
Linux RH 4.xx, 5.xx, 6.xx

The rest of the README file contains the license under which you may use and modify the software. Please, make sure you read it in your own time.

③ Do “configure” following the instructions in the README file for your platform:

-> ./configure
3.2. Download sites

Do you use X11R6.x? [y or n]
[default=y]: y
Enter X11R6 includes directory.
[default=/usr/X11R6/include]:
Enter X11R6 libraries directory.
[default=/usr/X11R6/lib]:
Do you use dynamic linking? [y or n]
[default=y]: y
Enter compiler flag(s).
[default=-02 -pipe]:

Configuration Summary:
OS type = Linux_22
X11 revision = 6
X11 include PATH = /usr/X11R6/include
X11 libraries PATH = /usr/X11R6/lib
Link style = Dynamic
Compiler flag(s) = -02 -pipe

Creating Makefile...
[server][monitor][sampleclient][recfile_change][logplayer][drawcheck]
Creating sserver script.
Done.

Do “make”

-> make
g++ -c -pipe -DLinux -DLinux_2_2 main.C
g++ -c -pipe -DLinux -DLinux_2_2 field.C
g++ -c -pipe -DLinux -DLinux_2_2 object.C
g++ -c -pipe -DLinux -DLinux_2_2 netif.C
.
.
.
g++ -c -02 -pipe -DLinux -DLinux_2_2 -I../server drawcheck.C
g++ -c -02 -pipe -DLinux -DLinux_2_2 -I../server netif.C
g++ -o drawcheck drawcheck.o netif.o -lm

If there are errors in the make, please check the g++ environment in your system.

3.2. Download sites

There are two download sites for the Soccer Server:

3. Getting Started

2. http://www.robolog.org/server/ (Germany)

The original home of the Soccer Server is the top site but it is soon to go out of commission. At the time of writing (March 2001) both sites are still up-and-running but the German site is to take over as the official home of the Soccer Server at some point in the summer of 2001. Users are prompted to use the German site to avoid future problems.

3.3. How to start the server

The `sserver-*` directory contains a `sserver` file. This is a shell script that is made by `configure`. Running it with `sserver` does three things:

1. it starts a Soccer Server
2. it starts a Soccer Monitor
3. when no longer needed, it stops the Soccer Server process

The Soccer Server runs in the background and produces some output to the terminal where the script was started from. Also, a window appears on the screen. This window is the Soccer Monitor so the user can actually watch the game. The output of running the `sserver` script should look like this:

```
-> sserver
Soccer Server Ver. 7.04
Itsuki Noda, Yasuo Kuniyoshi and Hitoshi Matsubara.
2001 RoboCup Soccer Server Maintainance Group.
Patrick Riley, Tom Howard, Jan Wendler, Itsuki Noda
Wind factor: rand: 0.000000, vector: (0.000000, 0.000000)
```

In order to actually start a match on the Soccer Server, the user must connect some clients to the server (maximum of 11 per side). When these clients are ready, the user can click the Kick Off button on the Soccer Monitor to start the game. It is likely that you have not yet programmed your own clients, in which case, you can read section 3.6 for instructions how to set up a whole match with two of the available teams that other RoboCuppers have contributed.

Also, there is a sample client included with every distribution of the Soccer Server which can be found in the `sampleclient` directory. To run one of these clients:

```
-> cd sampleclient
-> ls
Makefile       Makefile.tmpl2  client.c  client.o
Makefile.tmpl  client        client.h

-> client
```
3.4. How to stop the server

Running client attempts to connect to the server using default parameters (host=localhost, port=6000). Of course, these server parameters can be changed using the arguments that Soccer Server accepts when it is started. When the client is started, you need to initialise its connection to the server. The sampleclient is made so that the user types Soccer Server commands on the command line. So, to initialise the connection:

(init MyTeam)

You will notice that one of the two teams is now named “MyTeam” and one of the players that are standing by the side-line is active. This player corresponds to the client you’ve just initialised. Also, notice the information that the client writes on the terminal. This is what the client receives from the server. In the following text, the first two lines correspond to the initialisation and the other data is the information that the server sends to this client:

send 6000 : (init MyTeam)
recv 1067 : (init l 1 before_kick_off)

recv 1067 : (see 0 ((goal r) 66.7 33) ((flag r t) 55.7 3) ((flag p r t) 42.5 23) ((flag p r c) 53.5 43))
recv 1067 : (see 0 ((goal r) 66.7 33) ((flag r t) 55.7 3) ((flag p r t) 42.5 23) ((flag p r c) 53.5 43))

You can still type commands (such as (move 0 0) or (turn 45)) that the player will then send to the server. You should be able to see the result of these commands on the Soccer Monitor window.

3.4. How to stop the server

The correct procedure for stopping the server is:

1. Stop all clients (players)
2. Stop all Soccer Monitors by clicking on the Quit button
3. Hit CTRL-C at the terminal window where you started the Soccer Server in order to terminate it

If you follow this procedure, you will not only stop all visible running processes but also make sure that all those processes that may be running in the background (such as the Soccer Server) are also stopped. The problem that arises when you don’t properly stop all visible running processes and processes that may be running in the background is that they may continue to run in the background and affect the performance of other applications.

---

2The response from the server means that the client plays for the left side, has the number one and the play mode is before kick off.
3. Getting Started

shut down the Soccer Server is that you may not be able to start another process unless you start it with different parameters.

**NOTE**: It is sometimes useful and convenient to terminate processes using their name. Using the `kill` operating system command involves finding the process number of the process you want to stop using the `ps` command. A simpler way to eradicate all processes that have a specific name is by means of the `killall` command, for example: “killall soccermonitor” is sufficient to kill all processes with the name Soccer Monitor.

3.5. Supported platforms

The Soccer Server supports the following platforms:

- SunOS 4.1.x
- Solaris 2.x
- DEC OSF1
- Linux RedHat 4.xx, 5.xx, 6.xx

3.6. The process of a match

---

3\footnote{Supported platforms may change. Please, check the README file that came with your version of the software.}
4. Soccer Server

4.1. Objects

Figure 4.1.: UML diagram of the objects in the simulation
4. **Soccer Server**

## 4.2. Protocols

### 4.2.1. Client Command Protocol

**Connecting, reconnecting, and disconnecting**

<table>
<thead>
<tr>
<th>From client to server</th>
<th>From server to client</th>
</tr>
</thead>
<tbody>
<tr>
<td>(init TeamName [(version VerNum)] [(goalie)]) TeamName ::= (-</td>
<td>(a - z</td>
</tr>
<tr>
<td>(reconnect TeamName Unum) TeamName ::= (-</td>
<td>(a - z</td>
</tr>
<tr>
<td>(bye)</td>
<td></td>
</tr>
</tbody>
</table>

If your client connects or reconnects successfully with a protocol version \(\geq 7.0\), the server will additionally send following messages: a message containing the server parameters, a message containing the player parameters and a message containing the player types. The format is given below. Finally, the player will receive a message on changed players (see Sec. 4.6).

- **(server_param**
  - gwidth inertia\_moment psize pdecay prand pweight
  - pspeed\_max paccel\_max stamina\_max stamina\_inc recover\_init recover\_dthr recover\_min recover\_dec effort\_init effort\_dthr effort\_min effort\_dec effort\_dthr effort\_inc kick\_rand team\_actuator noise prand\_factor\_l prand\_factor\_r kick\_rand\_factor\_l kick\_rand\_factor\_r bsize bdecay brand bweight bspeed\_max baccel\_max dprate krate kmargin ctd\_radius ctd\_radius\_width maxp minp maxmn minmn maxnp minnp max\_area\_w catch\_area\_w catch\_prob goalie\_max\_moves ckmargin\_offside\_area\_w no win\_random say\_cnt\_max Say\_Coach\_Msg\_Size clang\_win\_size clang\_define\_win clang\_meta\_win clang\_advice\_win clang\_info\_win clang\_mess\_delay clang\_mess\_per\_cycle half\_time sim\_st send\_st recv\_st sb\_step lcm\_st Say\_Msg\_Size hear\_max hear\_inc hear\_decay cb\_cycle slow\_down\_factor use\_offside kick\_offside\_offside\_offside\_kick\_margin audio\_dist dist\_qstep land\_qstep\_l dir\_qstep\_r land\_qstep\_r dir\_qstep\_l dir\_qstep\_r Coach\_Mode Cw\_R\_Mode old\_hear sv\_st start\_goal\_l start\_goal\_r full\_state\_l full\_state\_r drop\_time)

- **(player_param**
  - player\_types
  - subs\_max pt\_max
  - player\_speed\_max\_delta\_min
  - player\_speed\_max\_delta\_max
  - stamina\_inc\_max\_delta\_factor
  - player\_decay\_delta\_min
  - player\_decay\_delta\_max
  - inertia\_moment\_delta\_factor
4.2. Protocols

- for each available player type a message of the form
  \[(\text{player_type id player_speed_max stamina_inc_max player_decay - inertia_moment dash_power_rate player_size kickable_margin kick_rand extra_stamina effort_max effort_min delta_factor_max delta_factor_min})\]

**Client Control**

<table>
<thead>
<tr>
<th>Operation</th>
<th>From client to server</th>
<th>Only once per cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>(catch <strong>Direction</strong>)</td>
<td>Direction ::= minmoment ~ maxmoment degrees</td>
<td>Yes</td>
</tr>
<tr>
<td>(change_view <strong>Width Quality</strong>)</td>
<td>Width ::= narrow</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>normal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wide</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quality ::= high</td>
<td></td>
</tr>
<tr>
<td></td>
<td>low</td>
<td></td>
</tr>
<tr>
<td>(dash <strong>Power</strong>)</td>
<td>Power ::= minpower ~ maxpower</td>
<td>Yes</td>
</tr>
<tr>
<td>Note: backward dash consumes double stamina.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(kick <strong>Power Direction</strong>)</td>
<td>Power ::= minpower ~ maxpower</td>
<td>Yes</td>
</tr>
<tr>
<td>Direction ::= minmoment ~ maxmoment degrees</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(move <strong>X Y</strong>)</td>
<td>X ::= -52.5 ~ 52.5</td>
<td>Yes</td>
</tr>
<tr>
<td>Y ::= -34 ~ 34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(say <strong>Message</strong>)</td>
<td>Message ::= a message</td>
<td>No</td>
</tr>
<tr>
<td>(sense <strong>body</strong>)</td>
<td>The server returns</td>
<td>No</td>
</tr>
<tr>
<td>(sense <strong>body Time</strong>)</td>
<td>(view_mode high</td>
<td>low) {narrow</td>
</tr>
<tr>
<td></td>
<td>(stamina Stamina Effort)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(speed AmountOfSpeed DirectionOfSpeed)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(head_angle HeadAngle)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(kick KickCount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(dash DashCount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(turn TurnCount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(say SayCount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(turn_neck TurnNeckCount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(catch CatchCount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(move MoveCount)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(change_view ChangeViewCount)</td>
<td></td>
</tr>
<tr>
<td>(score) The server returns</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>(score Time OurScore TheirScore)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(turn <strong>Moment</strong>)</td>
<td>Moment ::= minmoment ~ maxmoment degrees</td>
<td>Yes</td>
</tr>
<tr>
<td>(turn_neck <strong>Angle</strong>)</td>
<td>Angle ::= minneckmoment ~ maxneckmoment degrees</td>
<td>Yes</td>
</tr>
<tr>
<td>turn_neck is relative to the direction of the body.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can be invoked in the same cycle as a turn, dash or kick.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Soccer Server

The server may respond to the above commands with the errors:
(error unknown command)
(error illegal command form)
4.2. Client Sensor Protocol

From server to client

(hear Time Sender "Message") (hear Time Online_Coach Coach Language Message)

Time ::= simulation cycle of the soccerserver
Sender ::= online_coach_left | online_coach_right | coach | referee | self | Direction
Direction ::= -180 ~ 180 degrees
Message ::= string
Online_Coach ::= online_coach_left | online_coach_right
Coach_Language_Message ::= see the standard coach language section

(see Time ObjInfo)

Time ::= simulation cycle of the soccerserver
ObjInfo ::= (ObjName Distance Direction DistChange DirChange BodyFacingDir HeadFacingDir)
| (ObjName Distance Direction DistChange DirChange)
| (ObjName Distance Direction)
| (ObjName Direction)
ObjName ::= (p ["Teamname" [UniformNumber [goalie]]])
| (b)
| (g l|r)
| (f c)
| (f l|r] t|b])
| (f p [l|r] t|c|b])
| (f g [l|r] t|b])
| (f [l|r|t|b] 0)
| (f t|b] [l|r] [10:20:30:40:50])
| (f [l|r] t|b] [10:20:30])
| (l [l|r|t|b])
| (B)
| (F)
| (G)
| (P)

Distance ::= positive real number
Direction ::= -180 ~ 180 degrees
DistChange ::= real number
DirChange ::= real number
HeadFaceDir ::= -180 ~ 180 degrees
BodyFaceDir ::= -180 ~ 180 degrees
Teamname ::= string
UniformNumber ::= 1 ~ 11

(sense_body)

Time
(view_mode {high | low} {narrow | normal | wide})
(stamina Stamina Effort)
(speed AmountOfSpeed DirectionOfSpeed)
(head_angle HeadAngle)
(kick KickCount)
(dash DashCount)
(turn TurnCount)
(say SayCount)
(turn_neck TurnNeckCount)
(catch CatchCount)
(move MoveCount)
(change_view ChangeViewCount))

Time ::= simulation cycle of the soccerserver
Stamina ::= positive real number
Effort ::= positive real number
AmountOfSpeed ::= positive real number
DirectionOfSpeed ::= -180 ~ 180 degrees
HeadAngle ::= -180 ~ 180 degrees
*Count ::= positive integer
4.3. Sensor Models

A RoboCup agent has three different sensors. The aural sensor detects messages sent by the referee, the coaches and the other players. The visual sensor detects visual information about the field, like the distance and direction to objects in the player’s current field of view. The visual sensor also works as a proximity sensor by “seeing” objects that are close, but behind the player. The body sensor detects the current “physical” status of the player, like its stamina, speed and neck angle. Together the sensors give the agent quite a good picture of the environment.

4.3.1. Aural Sensor Model

Aural sensor messages are sent when a client or a coach sends a `say` command. The calls from the referee is also received as aural messages. All messages are received immediately.

The format of the aural sensor message from the soccer server is:

\[(\text{hear Time Sender ”Message”})\]

`Time` indicates the current time.

`Sender` is the relative direction to the sender if it is another player, otherwise it is one of the following:

- `self`: when the sender is yourself.
- `referee`: when the sender is the referee.
- `online_coach_left` or `online_coach_right`: when the sender is one of the online coaches.

`Message` is the message. The maximum length is `say_msg_size` bytes. The possible messages from the referee are described in Section 4.7.1.

The server parameters that affects the aural sensor are described in Tab. 4.1.

<table>
<thead>
<tr>
<th>Parameter in server.conf</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>audio_cut_dist</code></td>
<td>50.0</td>
</tr>
<tr>
<td><code>hear_max</code></td>
<td>2</td>
</tr>
<tr>
<td><code>hear_inc</code></td>
<td>1</td>
</tr>
<tr>
<td><code>hear_decay</code></td>
<td>2</td>
</tr>
<tr>
<td><code>say_msg_size</code></td>
<td>512</td>
</tr>
</tbody>
</table>

Table 4.1.: Parameters for the aural sensor
Capacity of the Aural Sensor

A player can only hear a message if the player’s hear capacity is at least $hear_{\text{decay}}$, since the hear capacity of the player is decreased by that number when a message is heard. Every cycle the hear capacity is increased with $hear_{\text{inc}}$. The maximum hear capacity is $hear_{\text{max}}$. To avoid a team from making the other team’s communication useless by overloading the channel the players have separate hear capacities for each team. With the current server.conf file this means that a player can hear at most one message from each team every second simulation cycle.

If more messages arrive at the same time than the player can hear the messages actually heard are undefined. (The current implementation choose the messages according to the order of arrival.) This rule does not include messages from the referee, or messages from oneself. In other words, a player can say a message and hear a message from another player in the same timestep.

Range of Communication

A message said by a player is transmitted only to players within $audio_{\text{cut}}_{\text{dist}}$ meters from that player. For example, a defender, who may be near his own goal, can hear a message from his goal-keeper but a striker who is near the opponent goal can not hear the message. Messages from the referee can be heard by all players.

Aural Sensor Example

This example should show which messages get through and how to calculated the hear capacity.

Example: Each coach sends a message every cycle. The referee send a message every cycle. The four players in the example all send a message every cycle. Show which messages gets through during 10 cycles (6 might be enough).

4.3.2. Vision Sensor Model

The visual sensor reports the objects currently seen by the player. The information is automatically sent to the player every $sense_{\text{step}}$, currently 150, milli-seconds.

Visual information arrives from the server in the following basic format:

$(\text{see ObjName Distance Direction DistChng DirChng BodyDir HeadDir})$

where
4. Soccer Server

ObjName ::= (p "Teamname" UniformNumber goalie) \\
   | (g [l|r]) \\
   | (b) \\
   | (f c) \\
   | (f [l|c|r] [t|b]) \\
   | (f p [l|r] [t|c|b]) \\
   | (f g [l|r] [t|b]) \\
   | (f [l|r|t|b] 0) \\
   | (f [t|b] [l|r] [10|20|30|40|50]) \\
   | (f [l|r] [t|b] [10|20|30]) \\
   | (l [l|r|t|b])

Distance, Direction, DistChng and DirChng are calculated in the following way:

\[
\begin{align*}
pr_x &= p_{xt} - pxo \\ 
pr_y &= p_{yt} - p_{yo} \\
v_{rx} &= v_{xt} - v_{xo} \\
v_{ry} &= v_{yt} - v_{yo} \\
\text{Distance} &= \sqrt{p_{rx}^2 + p_{ry}^2} \\
\text{Direction} &= \arctan(p_{ry}/p_{rx}) - \alpha_o \\
e_{rx} &= p_{rx}/\text{Distance} \\
e_{ry} &= p_{ry}/\text{Distance} \\
\text{DistChng} &= (v_{rx} * e_{rx}) + (v_{ry} * e_{ry}) \\
\text{DirChng} &= \left| (-v_{rx} * e_{rx}) + (v_{ry} * e_{rx}) \right|/\text{Distance} \times (180/\pi) \\
\text{BodyDir} &= \text{PlayerBodyDir} - \text{AgentBodyDir} - \text{AgentHeadDir} \\
\text{HeadDir} &= \text{PlayerHeadDir} - \text{AgentBodyDir} - \text{AgentHeadDir}
\end{align*}
\]

where \((p_{xt}, p_{yt})\) is the absolute position of the target object, \((pxo, p_{yo})\) is the absolute position of the sensing player, \((v_{xt}, v_{yt})\) is the absolute velocity of the target object, \((v_{xo}, v_{yo})\) is the absolute velocity of the sensing player, and \(\alpha_o\) is the absolute direction the sensing player is facing. The absolute facing direction of a player is the sum of the BodyDir and the HeadDir of that player. In addition to it, \((p_{rx}, p_{ry})\) and \((v_{rx}, v_{ry})\) are respectively the relative position and the relative velocity of the target, and \((e_{rx}, e_{ry})\) is the unit vector that is parallel to the vector of the relative position. BodyDir and HeadDir are only included if the observed object is a player, and is the body and head directions of the observed player relative to the body and head directions of the observing player. Thus, if both players have their bodies turned in the same direction, then BodyDir would be 0. The same goes for HeadDir.

The \((\text{goal r})\) object is interpreted as the center of the right hand side goalline. \((\text{f c})\) is a virtual flag at the center of the field. \((\text{f l b})\) is the flag at the lower left of the field. \((\text{f p l b})\) is a virtual flag at the lower right corner of the penalty box on the left side.
4.3. Sensor Models

of the field. \((\text{flag g l b})\) is a virtual flag marking the right goalpost on the left goal. The remaining types of flags are all located 5 meters outside the playing field. For example, \((\text{flag t l 20})\) is 5 meters from the top sideline and 20 meters left from the center line. In the same way, \((\text{flag r b 10})\) is 5 meters right of the right sideline and 10 meters below the center of the right goal. Also, \((\text{flag b 0})\) is 5 meters below the midpoint of the bottom sideline.

In the case of \((1 \ldots)\), Distance is the distance to the point where the center line of the player’s view crosses the line, and Direction is the direction of the line.

Range of View

The visible sector of a player is dependent on several factors. First of all we have the server parameters sense step and visible angle which determines the basic time step between visual information and how many degrees the player’s normal view cone is. The current default values are 150 milli-seconds and 90 degrees.

The player can also influence the frequency and quality of the information by changing ViewWidth and ViewQuality.
4. Soccer Server

To calculate the current view frequency and view angle of the agent use equations 4.13 and 4.14.

\[
\text{view frequency} = \text{sense step} \times \text{view quality factor} \times \text{view width factor} \quad (4.13)
\]

where view quality factor is 1 iff ViewQuality is high and 0.5 iff ViewQuality is low; view width factor is 2 iff ViewWidth is narrow, 1 iff ViewWidth is normal, and 0.5 iff ViewWidth is wide.

\[
\text{view angle} = \text{visible angle} \times \text{view width factor} \quad (4.14)
\]

where view width factor is 0.5 iff ViewWidth is narrow, 1 iff ViewWidth is normal, and 2 iff ViewWidth is wide.

The player can also “see” an object if it’s within visible distance meters of the player. If the objects is within this distance but not in the view cone then the player can know only the type of the object (ball, player, goal or flag), but not the exact name of the object. Moreover, in this case, the capitalized name, that is “B”, “P”, “G” and “F”, is used as the name of the object rather than “b”, “p”, “g” and “f”.

The following example and Fig. 4.3 are taken from [19].

The meaning of the view angle parameter is illustrated in Fig. 4.3. In this figure, the viewing agent is the one shown as two semi-circles. The light semi-circle is its front. The black circles represent objects in the world. Only objects within view angle/2, and those within visible distance of the viewing agent can be seen. Thus, objects b and g are not visible; all of the rest are.

As object f is directly in front of the viewing agent, its angle would be reported as 0 degrees. Object e would be reported as being roughly -40°, while object d is at roughly 20°.

Also illustrated in Fig. 4.3, the amount of information describing a player varies with how far away the player is. For nearby players, both the team and the uniform number of the player are reported. However, as distance increases, first the likelihood that the uniform number is visible decreases, and then even the team name may not be visible. It is assumed in the server that unum_far_length ≤ unum_too_far_length ≤ team_far_length ≤ team_too_far_length. Let the player’s distance be dist. Then

- If \( \text{dist} \leq \text{unum_far_length} \), then both uniform number and team name are visible.
- If \( \text{unum_far_length} < \text{dist} < \text{unum_too_far_length} \), then the team name is always visible, but the probability that the uniform number is visible decreases linearly from 1 to 0 as \( \text{dist} \) increases.
- If \( \text{dist} \geq \text{unum_too_far_length} \), then the uniform number is not visible.
- If \( \text{dist} \leq \text{team_far_length} \), then the team name is visible.
- If \( \text{team_far_length} < \text{dist} < \text{team_too_far_length} \), then the probability that the team name is visible decreases linearly from 1 to 0 as \( \text{dist} \) increases.
Figure 4.3.: The visible range of an individual agent in the soccer server. The viewing agent is the one shown as two semi-circles. The light semi-circle is its front. The black circles represent objects in the world. Only objects within \( \text{view\_angle}/2 \), and those within \( \text{visible\_distance} \) of the viewing agent can be seen. \( \text{unum\_far\_length} \), \( \text{unum\_too\_far\_length} \), \( \text{team\_far\_length} \), and \( \text{team\_too\_far\_length} \) affect the amount of precision with which a players’ identity is given. Taken from [19].
4. Soccer Server

- If \( \text{dist} \geq \text{team\_too\_far\_length} \), then the team name is not visible.

For example, in Fig. 4.3, assume that all of the labeled circles are players. Then player \( c \) would be identified by both team name and uniform number; player \( d \) by team name, and with about a 50% chance, uniform number; player \( e \) with about a 25% chance, just by team name, otherwise with neither; and player \( f \) would be identified simply as an anonymous player.

<table>
<thead>
<tr>
<th>Parameter in server.conf</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sense_step</td>
<td>150</td>
</tr>
<tr>
<td>visible_angle</td>
<td>90.0</td>
</tr>
<tr>
<td>visible_distance</td>
<td>3.0</td>
</tr>
<tr>
<td>unum_far_length(^a)</td>
<td>20.0</td>
</tr>
<tr>
<td>unum_too_far_length(^a)</td>
<td>40.0</td>
</tr>
<tr>
<td>team_far_length(^a)</td>
<td>40.0</td>
</tr>
<tr>
<td>team_too_far_length(^a)</td>
<td>60.0</td>
</tr>
<tr>
<td>quantize_step</td>
<td>0.1</td>
</tr>
<tr>
<td>quantize_step_l</td>
<td>0.01</td>
</tr>
</tbody>
</table>

\(^a\)Not in server.conf, but compiled into the server

Table 4.2.: Parameters for the visual sensors

Visual Sensor Noise Model

In order to introduce noise in the visual sensor data the values sent from the server is quantized. For example, the distance value of the object, in the case where the object in sight is a ball or a player, is quantized in the following manner:

\[
d' = \text{Quantize}(\exp(\text{Quantize}((\log(d), \text{quantize\_step}))), 0.1) \quad (4.15)
\]

where \( d \) and \( d' \) are the exact distance and quantized distance respectively, and

\[
\text{Quantize}(V, Q) = \text{ceiling}(V/Q) \cdot Q \quad (4.16)
\]

This means that players can not know the exact positions of very far objects. For example, when distance is about 100.0 the maximum noise is about 10.0, while when distance is less than 10.0 the noise is less than 1.0.

In the case of flags and lines, the distance value is quantized in the following manner.

\[
d' = \text{Quantize}(\exp(\text{Quantize}((\log(d), \text{quantize\_step\_l}))), 0.1) \quad (4.17)
\]
4.3. Sensor Models

4.3.3. Body Sensor Model

The body sensor reports the current “physical” status of the player. The information is automatically sent to the player every \texttt{sense_body\_step}, currently 100, milli-seconds.

The format of the body sensor message is:
\begin{verbatim}
(sense_body Time
  (view_mode ViewQuality ViewWidth)
  (stamina Stamina Effort)
  (speed AmountOfSpeed DirectionOfSpeed)
  (kick KickCount)
  (dash DashCount)
  (turn TurnCount)
  (say SayCount)
  (turn\_neck TurnNeckCount)
  (catch CatchCount)
  (move MoveCount)
  (change\_view ChangeViewCount))
\end{verbatim}

\texttt{ViewQuality} is one of \texttt{high} and \texttt{low}.

\texttt{ViewWidth} is one of \texttt{narrow}, \texttt{normal}, and \texttt{wide}.

\texttt{AmountOfSpeed} is an approximation of the amount of the player’s speed.

\texttt{DirectionOfSpeed} is an approximation of the direction of the player’s speed.

\texttt{HeadDirection} is the relative direction of the player’s head.

The \texttt{Count} variables are the total number of commands of that type executed by the server. For example \texttt{DashCount} = 134 means that the player has executed 134 \texttt{dash} commands so far.

The semantics of the parameters are described where they are actually used. The \texttt{ViewQuality} and \texttt{ViewWidth} parameters are for example described in the Section 4.3.2. The server parameters that affects the body sensor are described in Tab. 4.3.

\begin{center}
\begin{tabular}{|l|c|}
\hline
\texttt{Parameter in server.conf} & \texttt{Value} \\
\hline
\texttt{sense_body\_step} & 100 \\
\hline
\end{tabular}
\end{center}

Table 4.3.: Parameters for the body sensor
4. Soccer Server

4.4. Movement Model

In each simulation step, movement of each object is calculated as following manner:

\[
(u^{t+1}_x, u^{t+1}_y) = (v^t_x, v^t_y) + (a^t_x, a^t_y): \text{accelerate (4.18)}
\]

\[
(p^{t+1}_x, p^{t+1}_y) = (p^t_x, p^t_y) + (u^{t+1}_x, u^{t+1}_y): \text{move}
\]

\[
(v^{t+1}_x, v^{t+1}_y) = \text{decay} \times (u^{t+1}_x, u^{t+1}_y): \text{decay speed}
\]

\[
(a^{t+1}_x, a^{t+1}_y) = (0, 0): \text{reset acceleration}
\]

where, \((p^t_x, p^t_y)\), and \((v^t_x, v^t_y)\) are respectively position and velocity of the object in timestep \(t\). decay is a decay parameter specified by ball_decay or player_decay. \((a^t_x, a^t_y)\) is acceleration of object, which is derived from Power parameter in dash (in the case the object is a player) or kick (in the case of a ball) commands in the following manner:

\[
(a^t_x, a^t_y) = \text{Power} \times \text{power_rate} \times (\cos(\theta^t), \sin(\theta^t))
\]

where \(\theta^t\) is the direction of the object in timestep \(t\) and \(\text{power_rate}\) is dash_power_rate or is calculated from kick_power_rate as described in Sec. 4.5.3. In the case of a player, this is just the direction the player is facing. In the case of a ball, its direction is given as the following manner:

\[
\theta^t_{\text{ball}} = \theta^t_{\text{kicker}} + \text{Direction}
\]

where \(\theta^t_{\text{ball}}\) and \(\theta^t_{\text{kicker}}\) are directions of ball and kicking player respectively, and Direction is the second parameter of a kick command.

4.4.1. Movement Noise Model

In order to reflect unexpected movements of objects in real world, \(\beta\) adds noise to the movement of objects and parameters of commands. Concerned with movements, noise is added into Eqn. 4.18 as follows:

\[
(u^{t+1}_x, u^{t+1}_y) = (v^t_x, v^t_y) + (a^t_x, a^t_y) + (\tilde{r}_{\text{max}}, \tilde{r}_{\text{max}})
\]

where \(\tilde{r}_{\text{max}}\) is a random number whose distribution is uniform over the range \([-\text{rmax}, \text{rmax}]\). \(\text{rmax}\) is a parameter that depends on amount of velocity of the object as follows:

\[
\text{rmax} = \text{rand} \cdot |(v^t_x, v^t_y)|
\]

where \(\text{rand}\) is a parameter specified by player_rand or ball_rand.

Noise is added also into the Power and Moment arguments of a command as follows:

\[
\text{argument} = (1 + \tilde{r}_{\text{rand}}) \cdot \text{argument}
\]
4.4.2. Collision Model

If at the end of the simulation cycle, two objects overlap, then the objects are moved back until they do not overlap. Then the velocities are multiplied by -0.1. Note that it is possible for the ball to go through a player as long as the ball and the player never overlap at the end of the cycle.

4.5. Action Models

4.5.1. Catch Model

The goalie is the only player with the ability to catch a ball. The goalie can catch the ball in play mode `play_on` in any direction, if the ball is within the catchable area and the goalie is inside the penalty area. If the goalie catches into direction \( \varphi \), the catchable area is a rectangular area of length \( \text{catchable_area}_l \) and width \( \text{catchable_area}_w \) in direction \( \varphi \) (see Fig. 4.4). The ball will be caught with probability \( \text{catch_probability} \), if it is inside this area (and it will not be caught if it is outside this area). For the current values of catch command parameters see Tab. 4.4.

![Figure 4.4.: Catchable area of the goalie when doing a catch 45](image)

Figure 4.4.: Catchable area of the goalie when doing a catch 45

If a catch command was unsuccessful, it takes \textit{catch_ban_cycle} cycles until another catch command can be used (catch commands during this time have simply no effect). If the goalie succeeded in catching the ball, the play mode will change to `goalie_catch_ball_[l|r]` first and `free_kick_[l|r]`, after that during the same cycle. Once the goalie caught the ball, it can use the \textit{move} command to move with the ball inside the penalty area. The goalie can use the \textit{move} command \textit{goalie_max_moves} times before it kicks the ball. Additional \textit{move} commands do not have any effect and
the server will respond with (error too many moves). Please note that catching the ball, moving around, kicking the ball a short distance and immediately catching it again to move more than goalie_max_moves times is considered as ungentlemanly play.

<table>
<thead>
<tr>
<th>Parameter in server.conf</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>catchable_area_l</td>
<td>2.0</td>
</tr>
<tr>
<td>catchable_area_w</td>
<td>1.0</td>
</tr>
<tr>
<td>catch_probability</td>
<td>1.0</td>
</tr>
<tr>
<td>catch_ban_cycle</td>
<td>5</td>
</tr>
<tr>
<td>goalie_max_moves</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.4.: Parameters for the goalie catch command

4.5.2. Dash Model (incl. stamina model)

Dash Model

The dash command is used to accelerate the player in direction of its body. dash takes the acceleration power as a parameter. The valid range for the acceleration power can be configured in server.conf, the respective parameters are minpower and maxpower. For the current values of parameters for the dash model, see Tab. 4.5.

Each player has a certain amount of stamina that will be consumed by dash commands. At the beginning of each half, the stamina of a player is set to stamina_max. If a player accelerates forward (power > 0), stamina is reduced by power. Accelerating backwards (power < 0) is more expensive for the player: stamina is reduced by −2 · power. If the player’s stamina is lower than the power needed for the dash, power is reduced so that the dash command does not need more stamina than available. Heterogeneous players will use some extra stamina every time the available power is lower than the needed stamina. The amount of extra stamina depends on the player type and the parameters extra_stamina_delta_min and extra_stamina_delta_max.

After reducing the stamina, the server calculates the effective dash power for the dash command. The effective dash power edp depends on the dash_power_rate and the current effort of the player. The effort of a player is a value between effort_min and effort_max; it is dependent on the stamina management of the player (see below).

\[ edp = \text{effort} \cdot \text{dash}_{-}\text{power}_{-}\text{rate} \cdot \text{power} \quad (4.19) \]

edp and the players current body direction are transformed into vector and added to the players current acceleration vector \( \mathbf{\tilde{a}}_n \) (usually, that should be 0 before, since a player cannot dash more than once a cycle and a player does not get accelerated by other means than dashing1).

At the transition from simulation step \( n \) to simulation step \( n + 1 \), acceleration \( \mathbf{\tilde{a}}_n \) is applied:

\[ \text{is that so?} \]
4.5. Action Models

1. $\vec{a}_n$ is normalized to a maximum length of $\text{player\_accel\_max}$. 

2. $\vec{a}_n$ is added to current players speed $\vec{v}_n$. $\vec{v}_n$ will be normalized to a maximum length of $\text{player\_speed\_max}$. For heterogeneous players, the maximum speed is a value between $\text{player\_speed\_max} + \text{player\_speed\_max\_delta\_min}$ and $\text{player\_speed\_max} + \text{player\_speed\_max\_delta\_max}$ in player.conf.

3. Noise $\vec{n}$ and wind $\vec{w}$ will be added to $\vec{v}_n$. Both noise and wind are configurable in server.conf. Parameters responsible for the wind are $\text{wind\_force}$, $\text{wind\_dir}$ and $\text{wind\_rand}$. With the current settings, there is no wind on the simulated soccer field. The responsible parameter for the noise is $\text{player\_rand}$. Both direction and length of the noise vector are within the interval $[-|\vec{v}_n| \cdot \text{player\_rand}, \ldots |\vec{v}_n| \cdot \text{player\_rand}]$.

4. The new position of the player $\vec{p}_{n+1}$ is the old position $\vec{p}_n$ plus the velocity vector $\vec{v}_n$ (i.e. the maximum distance difference for the player between two simulation steps is $\text{player\_speed\_max}$).

5. $\text{player\_decay}$ is applied for the velocity of the player: $\vec{v}_{n+1} = \vec{v}_n \cdot \text{player\_decay}$. Acceleration $\vec{a}_{n+1}$ is set to zero.

Stamina Model

For the stamina of a player, there are three important variables: the stamina value, recovery and effort. stamina is decreased when dashing and gets replenished slightly each cycle. recovery is responsible for how much the stamina recovers each cycle, and the effort says how effective dashing is (see section above). Important parameters for the stamina model are changeable in the files server.conf and player.conf, see also Tab. 4.5. Basically, the algorithm shown in Fig. 4.5 says that every simulation step the stamina is below some threshold, effort or recovery are reduced until a minimum is reached. Every step the stamina of the player is above some threshold, effort is increased up to a maximum. The recovery value is only reset to 1.0 each half, but it will not be increased during a game.

4.5.3. Kick Model

There are no principal changes to the kick model from soccer server version 6 to soccer server version 7, so your old implementation should still work. However, due to changes in the server parameter file, in some cases multiple kicks are not necessary anymore.

The kick command takes two parameters, the kick power the player client wants to use (between $\text{minpower}$ and $\text{maxpower}$) and the angle the player kicks the ball to. The angle is given in degrees and has to be between $\text{minmoment}$ and $\text{maxmoment}$ (see Tab. 4.6 for current parameter values).

Once the kick command arrived at the server, the kick will be executed if the ball is kick-able for the player and the player is not marked offside. The ball is kick-able
### 4. Soccer Server

#### Basic Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minpower</td>
<td>-100</td>
<td>maxpower</td>
<td>100</td>
</tr>
<tr>
<td>stamina_max</td>
<td>4000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stamina_inc_max</td>
<td>45</td>
<td>stamina_inc_max_delta_factor</td>
<td>-100.0</td>
</tr>
<tr>
<td>player_speed_max_delta_min</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>player_speed_max_delta_max</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>extra_stamina</td>
<td>0.0</td>
<td>extra_stamina_delta_min</td>
<td>0.0</td>
</tr>
<tr>
<td>extra_stamina_max</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dash_power_rate</td>
<td>0.006</td>
<td>dash_power_rate_delta_min</td>
<td>0.0</td>
</tr>
<tr>
<td>dash_power_rate_max</td>
<td>0.002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>effort_min</td>
<td>0.6</td>
<td>effort_min_delta_factor</td>
<td>-0.002</td>
</tr>
<tr>
<td>extra_stamina_delta_min</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>extra_stamina_delta_max</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>effort_max</td>
<td>1.0</td>
<td>effort_max_delta_factor</td>
<td>-0.002</td>
</tr>
<tr>
<td>extra_stamina_delta_min</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>extra_stamina_delta_max</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>effort_dec_thr</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>effort_dec</td>
<td>0.005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>effort_inc_thr</td>
<td>0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>effort_inc</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>recover_dec_thr</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>recover_dec</td>
<td>0.002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>recover_min</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>player_accel_max</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>player_speed_max</td>
<td>1.0</td>
<td>player_speed_max_delta_min</td>
<td>0.0</td>
</tr>
<tr>
<td>player_speed_max_delta_max</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>player_rand</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wind_force</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wind_dir</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wind_rand</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>player_decay</td>
<td>0.4</td>
<td>player_decay_delta_min</td>
<td>0.0</td>
</tr>
<tr>
<td>player_decay_delta_max</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Not in server.conf, but compiled into the server*

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>minpower</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpower</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stamina_max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stamina_inc_max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>player_speed_max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>player_speed_max_delta_min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>player_speed_max_delta_max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>effort_min</td>
<td></td>
<td>0.4</td>
</tr>
<tr>
<td>effort_max</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>player_accel_max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>player_speed_max</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>player_speed_max_delta_min</td>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>player_speed_max_delta_max</td>
<td></td>
<td>1.2</td>
</tr>
</tbody>
</table>

Table 4.5.: Dash and Stamina Model Parameters for Soccer Server 7
{if stamina is below recovery decrement threshold, recovery is reduced}
if stamina \leq \text{recover\_dec\_thr} \cdot \text{stamina\_max} \text{ then}
   \quad \text{if recovery > recover\_min then}
   \quad \quad \text{recovery} \leftarrow \text{recovery} - \text{recover\_dec}
   \quad \text{end if}
\text{end if}

{if stamina is below effort decrement threshold, effort is reduced}
if stamina \leq \text{effort\_dec\_thr} \cdot \text{stamina\_max} \text{ then}
   \quad \text{if effort > effort\_min then}
   \quad \quad \text{effort} \leftarrow \text{effort} - \text{effort\_dec}
   \quad \text{end if}
   \quad \text{effort} \leftarrow \max(\text{effort}, \text{effort\_min})
\text{end if}

{if stamina is above effort increment threshold, effort is increased}
if stamina \geq \text{effort\_inc\_thr} \cdot \text{stamina\_max} \text{ then}
   \quad \text{if effort < effort\_max then}
   \quad \quad \text{effort} \leftarrow \text{effort + effort\_inc}
   \quad \quad \text{effort} \leftarrow \min(\text{effort}, \text{effort\_max})
   \quad \text{end if}
\text{end if}

{recover the stamina a bit}
stamina \leftarrow \text{stamina + recovery \cdot stamina\_inc\_max}
stamina \leftarrow \min(\text{stamina}, \text{stamina\_max})

Figure 4.5.: The stamina model algorithm
for the player, if the distance between the player and the ball is between 0 and \textit{kickable\_margin}. Heterogeneous players can have different kickable margins. For the calculation of the distance during this section, it is important to know that if we talk of distance between player and ball, we talk about the minimal distance between the outer shape of both player and ball. So the distance in this section is the distance between the center of both objects minus the radius of the ball minus the radius of the player.

The first thing to be calculated for the kick is the effective kick power \( ep \): 

\[
ep = \text{kick power} \cdot \text{kick\_power\_rate}
\] (4.20)

If the ball is not directly in front of the player, the effective kick power will be reduced by a certain amount dependent on the position of the ball with respect to the player. Both angle and distance are important.

If the relative angle of the ball is 0° wrt. the body direction of the player client — i.e. the ball is in front of the player — the effective power will stay as it is. The larger the angle gets, the more the effective power will be reduced. The worst case is if the ball is lying behind the player (angle 180°): the effective power is reduced by 25%.

The second important variable for the effective kick power is the distance from the ball to the player: it is quite obvious that — should the kick be executed — the distance between ball and player is between 0 and \textit{kickable\_margin}. If the distance is 0, the effective kick power will not be reduced again. The further the ball is away from the player client, the more the effective kick power will be reduced. If the ball distance is \textit{kickable\_margin}, the effective kick power will be reduced by 25% of the original kick power.

The overall worst case for kicking the ball is if a player kicks a distant ball behind itself: 50% of kick power will be used. For the effective kick power, we get the formula 4.21. (\( \text{dir\_diff} \) means the absolute direction difference between ball and the player’s body direction, \( \text{dist\_diff} \) means the absolute distance between ball and player.)

\[
0 \leq \text{dir\_diff} \leq 180° \quad \land \quad 0 \leq \text{dist\_diff} \leq \text{kickable\_margin}:
ep = ep \cdot \left(1 - 0.25 \cdot \frac{\text{dir\_diff}}{180°} - 0.25 \cdot \frac{\text{dist\_ball}}{\text{kickable\_margin}}\right)
\] (4.21)

The effective kick power is used to calculate \( \vec{a}_n \), an acceleration vector that will be added to the global ball acceleration \( \vec{a}_n \) during cycle \( n \) (remember that we have a multi agent system and each player close to the ball can kick it during the same cycle).

There is a server parameter, \textit{kick\_rand}, that can be used to generate some noise to the ball acceleration. For the default players, \textit{kick\_rand} is 0 and no noise will be generated. For heterogeneous players, \textit{kick\_rand} depends on \textit{kick\_rand\_delta\_factor} in \texttt{player.conf} and on the actual kickable margin. In RoboCup 2000, \textit{kick\_rand} was used to generate some noise during evaluation round for the normal players.

During the transition from simulation step \( n \) to simulation step \( n + 1 \) acceleration \( \vec{a}_n \) is applied:

1. \( \vec{a}_n \) is normalized to a maximum length of \textit{baccel\_max}. Currently (Server 7), the maximum acceleration is equal to the maximum effective kick power.
2. \( \vec{a}_n \) is added to the current ball speed \( \vec{v}_n \). \( \vec{v}_n \) will be normalized to a maximum length of \( \text{ball speed max} \).

3. Noise \( \vec{n} \) and wind \( \vec{w} \) will be added to \( \vec{v}_n \). Both noise and wind are configurable in \texttt{server.conf}. Parameters responsible for the wind are \texttt{wind force}, \texttt{wind dir} and \texttt{wind rand}. The responsible parameter for the noise is \texttt{ball rand}. Both direction and length of the noise vector are within the interval \([-|\vec{v}_n| \cdot \text{ball rand} \ldots |\vec{v}_n| \cdot \text{ball rand}]\).

4. The new position of the ball \( \vec{p}_{n+1} \) is the old position \( \vec{p}_n \) plus the velocity vector \( \vec{v}_n \) (i.e. the maximum distance difference for the ball between two simulation steps is \( \text{ball speed max} \)).

5. \texttt{ball decay} is applied for the velocity of the ball: \( \vec{v}_{n+1} = \vec{v}_n \cdot \text{ball decay} \). Acceleration \( \vec{a}_{n+1} \) is set to zero.

With the current settings the ball covers a distance up to 45, assuming an optimal kick. 53 cycles after an optimal kick, the distance from the kick off position to the ball is about 43, the remaining velocity is smaller than 0.1. 15 cycles after an optimal kick, the ball covers a distance of 27 – 28 and the remaining velocity is slightly larger than 1.

Implications from the kick model and the current parameter settings are that it still might be helpful to use several small kicks for a compound kick – for example stopping the ball, kick it to a more advantageous position within the kickable area and kick it to the desired direction. It would be another possibility to accelerate the ball to maximum speed without putting it to relative position \((0,0)\) using a compound kick.

### 4.5.4. Move Model

The \texttt{move} command can be used to place a player directly onto a desired position on the field. \texttt{move} exists to set up the team and does not work during normal play. It is available at the beginning of each half (play mode ‘\texttt{before kick off}’) and after a goal has been scored (play modes ‘\texttt{goal r.n}’ or ‘\texttt{goal l.n}’). In these situations, players can be placed on any position in their own half (i.e. \( X < 0 \)) and can be moved any number of times, as long as the play mode does not change. Players moved to a position on the opponent half will be set to a random position on their own side by the server.

A second purpose of the \texttt{move} command is to move the goalie within the penalty area after the goalie caught the ball (see also Sec. 4.5.1). If the goalie caught the ball, it can move together with the ball within the penalty area. The goalie is allowed to move \texttt{goalie max moves} times before it kicks the ball. Additional \texttt{move} commands do not have any effect and the server will respond with \texttt{(error too many moves)}.

### 4.5.5. Say Model

Using the \texttt{say} command, players can broadcast messages to other players. Messages can be \texttt{say msg size} characters long, where valid characters for \texttt{say} messages are from
### Basic Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>minpower</td>
<td>-100</td>
<td>maxpower</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>minmoment</td>
<td>-180</td>
<td>maxmoment</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>kickable_margin</td>
<td>0.7</td>
<td>kickable_margin_delta_min</td>
<td>0.0</td>
<td>0.7</td>
</tr>
<tr>
<td>kickable_margin</td>
<td></td>
<td>kickable_margin_delta_max</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>kick_power_rate</td>
<td>0.027</td>
<td>kick_rand</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>kick_rand</td>
<td></td>
<td>kick_rand_delta_factor</td>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>ball_size</td>
<td>0.085</td>
<td>ball_decay</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>ball_rand</td>
<td>0.05</td>
<td>ball_speed_max</td>
<td>2.7</td>
<td></td>
</tr>
<tr>
<td>ball_accel_max</td>
<td>2.7</td>
<td>wind_force</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>wind_dir</td>
<td>0.0</td>
<td>wind_rand</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6.: Ball and Kick Model Parameters

### Parameter for heterogeneous Players

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>goalie_max_moves</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.7.: Parameter for the move command
4.5. Action Models

the set [-0-9a-zA-Z () .+*/?<>] (without the square brackets). Messages players say can be heard within a distance of `audio_cut_dist` by members of both teams (see also Sec. 4.3.1). Say messages sent to the server will be sent back to players within that distance immediately. The use of the `say` command is only restricted by the limited capacity of the players of hearing messages.

<table>
<thead>
<tr>
<th>Parameter in server.conf</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>say_msg_size</code></td>
<td>512</td>
</tr>
<tr>
<td><code>audio_cut_dist</code></td>
<td>50</td>
</tr>
<tr>
<td><code>hear_max</code></td>
<td>2</td>
</tr>
<tr>
<td><code>hear_inc</code></td>
<td>1</td>
</tr>
<tr>
<td><code>hear_decay</code></td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.8.: Parameters for the `say` command

4.5.6. Turn Model

While `dash` is used to accelerate the player in direction of its body, the `turn` command is used to change the players body direction. The argument for the `turn` command is the moment; valid values for the moment are between `minmoment` and `maxmoment`. If the player does not move, the moment is equal to the angle the player will turn. However, there is a concept of inertia that makes it more difficult to turn when you are moving. Specifically, the actual angle the player is turned is as follows:

\[
\text{actual\_angle} = \text{moment}/(1.0 + \text{inertia\_moment} \cdot \text{player\_speed})
\]

`inertia\_moment` is a `server.conf` parameter with default value 5.0. Therefore (with default values), when the player is at speed 1.0, the maximum effective turn he can do is ±30. However, notice that because you can not dash and turn during the same cycle, the fastest that a player can be going when executing a `turn` is `player\_speed\_max` - `player\_decay`, which means the effective turn for a default player (with default values) is ±60.

For heterogeneous players, the inertia moment is the default `inertia\_value` plus a value between `min\_player\_decay\_delta\_min` - `inertia\_moment\_delta\_factor` and `max\_player\_decay\_delta\_max` - `inertia\_moment\_delta\_factor`.

4.5.7. TurnNeck Model

With `turn\_neck`, a player can turn its neck somewhat independently of its body. The angle of the head of the player is the viewing angle of the player. The `turn` command changes the angle of the body of the player while `turn\_neck` changes the neck angle of the player relative to its body. The minimum and maximum relative angle for the player’s neck are given by `minmoment` and `maxmoment` in `server.conf`. Remember
4. Soccer Server

<table>
<thead>
<tr>
<th>Basic Parameters</th>
<th>Parameters for heterogeneous Players</th>
</tr>
</thead>
<tbody>
<tr>
<td>server.conf</td>
<td>player.conf</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Value</td>
<td>Value</td>
</tr>
<tr>
<td>Range</td>
<td>Range</td>
</tr>
<tr>
<td><strong>minmoment</strong></td>
<td><strong>player_decay_delta_min</strong></td>
</tr>
<tr>
<td><strong>maxmoment</strong></td>
<td><strong>player_decay_delta_max</strong></td>
</tr>
<tr>
<td><strong>inertia_moment</strong></td>
<td><strong>inertia_moment_delta_factor</strong></td>
</tr>
<tr>
<td>minmoment -180</td>
<td>player_decay_delta_min 0.0 0.2</td>
</tr>
<tr>
<td>maxmoment 180</td>
<td>player_decay_delta_max 5.0 10.0</td>
</tr>
<tr>
<td>inertia_moment 5.0</td>
<td>inertia_moment_delta_factor 25.0 10.0</td>
</tr>
</tbody>
</table>

Table 4.9.: Turn Model Parameters

that the neck angle is relative to the body of the player so if the client issues a `turn` command, the viewing angle changes even if no `turn neck` command was issued.

Also, `turn neck` commands can be executed during the same cycle as `turn`, `dash`, and `kick` commands. `turn neck` is not affected by momentum like `turn` is. The argument for a `turn neck` command must be in the range between `minneckmoment` and `maxneckmoment`.

<table>
<thead>
<tr>
<th>Parameter in <code>server.conf</code></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>minneckang</code></td>
<td>-90</td>
</tr>
<tr>
<td><code>maxneckang</code></td>
<td>90</td>
</tr>
<tr>
<td><code>minneckmoment</code></td>
<td>-180</td>
</tr>
<tr>
<td><code>maxneckmoment</code></td>
<td>180</td>
</tr>
</tbody>
</table>

Table 4.10.: Parameter for the `turn neck` command

4.6. Heterogeneous Players

With Soccer Server 7, heterogeneous players were introduced. For heterogeneous players, Soccer Server generates `player_types` random player types at startup. The player types have different abilities based on the tradeoffs defined in the `player.conf` file. Both teams of a match use the same player types. Type 0 is the default type and always the same.

When the players connect to the server, the receive information on the available player types (see Sec. 4.2.1). The online coach can change player types unlimited times in ‘before kick off’ mode and change player types `subs max` times during other non-`play on’ play modes using the `change_player_type ...` command (see Sec. 7.4).

Each time a player is substituted by some other player type, its stamina, recovery and effort is reset to the initial (maximum) value of the respective player type.
4.7. Referee Model

The Automated Referee sends messages to the players, so that players know the actual play mode of the game. The rules and the behavior for the automated referee are described in Sec. 2.2.1. Players receive the referee messages as hear messages. A player can hear referee messages in every situation independent of the number of messages the player heard from other players.

4.7.1. Play Modes and referee messages

The change of the play mode is announced by the referee. Additionally, there are some referee messages announcing events like a goal or a foul. If you have a look into the server source code, you will notice some additional play modes that are currently not used. Both play modes and referee messages are announced using (referee String), where String is the respective play mode or message string. The play modes are listed in Tab. 4.12, for the messages see Tab. 4.13.

<table>
<thead>
<tr>
<th>Play Mode</th>
<th>tc</th>
<th>subsequent play mode</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘before_kick_off’</td>
<td>0</td>
<td>‘kick_off_side’</td>
<td>at the beginning of a half</td>
</tr>
<tr>
<td>‘play_on’</td>
<td></td>
<td></td>
<td>during normal play</td>
</tr>
<tr>
<td>‘time_over’</td>
<td></td>
<td></td>
<td>announce start of play</td>
</tr>
<tr>
<td>‘kick_off_side’</td>
<td></td>
<td></td>
<td>(after pressing the Kick Off button)</td>
</tr>
<tr>
<td>‘kick_in_side’</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘free_kick_side’</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘corner_kick_side’</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘goal_kick_side’</td>
<td></td>
<td>‘play_on’</td>
<td>play mode changes once</td>
</tr>
<tr>
<td>‘goal_side’</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘drop_ball’</td>
<td>0</td>
<td>‘play_on’</td>
<td></td>
</tr>
<tr>
<td>‘offside_side’</td>
<td>30</td>
<td>‘free_kick_side’</td>
<td></td>
</tr>
</tbody>
</table>

where Side is either the character ‘1’ or ‘r’, OSide means opponent’s side.

tc is the time (in number of cycles) until the subsequent play mode will be announced

Table 4.12.: Play Modes

<table>
<thead>
<tr>
<th>Parameter in player.conf</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>player_types</td>
<td>7</td>
</tr>
<tr>
<td>subs_max</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.11.: Parameter for substitutions and heterogeneous player types
4. Soccer Server

<table>
<thead>
<tr>
<th>Message</th>
<th>$t_c$</th>
<th>subsequent play mode</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal_Side_n</td>
<td>50</td>
<td>'kick_off_OSide'</td>
<td>announce the $n$th goal for a team</td>
</tr>
<tr>
<td>foul_Side</td>
<td>0</td>
<td>'free_kick_OSide'</td>
<td>announce a foul</td>
</tr>
<tr>
<td>goalie_catch_ball_Side</td>
<td>0</td>
<td>'free_kick_OSide'</td>
<td></td>
</tr>
<tr>
<td>time_up_without_team</td>
<td>0</td>
<td>'time_over'</td>
<td>sent if there was no opponent until the end of the second half</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>sent once the game is over (if the time is $\geq$ second half and the scores for each team are different)</td>
</tr>
<tr>
<td>half_time</td>
<td>0</td>
<td>'before_kick_off'</td>
<td></td>
</tr>
<tr>
<td>time_extended</td>
<td>0</td>
<td>'before_kick_off'</td>
<td></td>
</tr>
</tbody>
</table>

where $\text{Side}$ is either the character ‘l’ or ‘r’, $\text{OSide}$ means opponent’s side.
$t_c$ is the time (in number of cycles) until the subsequent play mode will be announced

Table 4.13.: Referee Messages

4.8. The Soccer Simulation

In Sec. 4.4, we gave a description on how the objects are moved with respect to their accelerations and velocities. In this section, we describe at what point in time acceleration and velocities are applied to the objects during the simulation.

4.8.1. Description of the simulation algorithm

In Soccer Server, time is updated in discrete steps. A simulation step is 100ms. During each simulation step, objects (i.e. players and the ball) stay on their positions. If players decide to act within a step, actions are applied to the players and the ball at the transition from one simulation cycle to the next. Depending on the play mode, not all actions are allowed for the players (for instance in ‘before\_kick\_off’ mode, players can turn and move, but they cannot dash), so only allowed actions will be applied and take effect.

If during a step, several players kick the ball, all the kicks are applied to the ball and a resulting acceleration is calculated. If the resulting acceleration is larger than the maximum acceleration for the ball, acceleration is normalized to its maximum value. After moving the objects, the server checks for collisions and updates velocities if a collision occurred (see also Sec. 4.4.2).

When applying accelerations and velocities to the objects, the order of application is randomized. After changing objects positions, and updating velocities and accelerations, the automated referee checks the situation and changes the play mode or the object positions, if necessary. Changes to the play mode are announced immediately. Finally, stamina for each player is updated.
4.9. Using Soccerserver

4.9.1. The Soccerserver Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Current Value in server.conf</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal_width</td>
<td>7.32</td>
<td>14.02</td>
<td>goal width</td>
</tr>
<tr>
<td>player_size</td>
<td>0.3</td>
<td>player size</td>
<td></td>
</tr>
<tr>
<td>player_decay</td>
<td>0.4</td>
<td>player decay</td>
<td></td>
</tr>
<tr>
<td>player_rand</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>player_weight</td>
<td>60.0</td>
<td>player weight</td>
<td></td>
</tr>
<tr>
<td>player_speed_max</td>
<td>1.0</td>
<td>max. player velocity</td>
<td></td>
</tr>
<tr>
<td>player_accel_max</td>
<td>1.0</td>
<td>max. player acceleration</td>
<td></td>
</tr>
<tr>
<td>stamina_max</td>
<td>4000.0</td>
<td>max. player stamina</td>
<td></td>
</tr>
<tr>
<td>stamina_inc_max</td>
<td>45.0</td>
<td>max. player stamina increment</td>
<td></td>
</tr>
<tr>
<td>recover_dec_thr</td>
<td>0.3</td>
<td>player recovery decrement threshold</td>
<td></td>
</tr>
<tr>
<td>recover_min</td>
<td>0.5</td>
<td>min. player recovery</td>
<td></td>
</tr>
<tr>
<td>recover_dec</td>
<td>0.002</td>
<td>player recovery decrement</td>
<td></td>
</tr>
<tr>
<td>effort_dec_thr</td>
<td>0.3</td>
<td>player dash effort</td>
<td>decrement threshold</td>
</tr>
<tr>
<td>effort_min</td>
<td>0.6</td>
<td>min. player dash effort</td>
<td></td>
</tr>
<tr>
<td>effort_dec</td>
<td>0.005</td>
<td>dash effort decrement</td>
<td></td>
</tr>
<tr>
<td>effort_inc_thr</td>
<td>0.6</td>
<td>dash effort increment treshold</td>
<td></td>
</tr>
<tr>
<td>effort_inc</td>
<td>0.01</td>
<td>dash effort increment</td>
<td></td>
</tr>
<tr>
<td>kick_rand</td>
<td>0.0</td>
<td>noise added directly to kicks</td>
<td></td>
</tr>
<tr>
<td>team_actuator_noise</td>
<td></td>
<td></td>
<td>flag whether to use team specific actuator noise</td>
</tr>
<tr>
<td>prand_factor_l</td>
<td></td>
<td>factor to multiply prand</td>
<td>for left team</td>
</tr>
<tr>
<td>prand_factor_r</td>
<td></td>
<td>factor to multiply prand</td>
<td>for right team</td>
</tr>
<tr>
<td>kick_rand_factor_l</td>
<td></td>
<td>factor to multiply kick_rand</td>
<td>for left team</td>
</tr>
<tr>
<td>kick_rand_factor_r</td>
<td></td>
<td>factor to multiply kick_rand</td>
<td>for right team</td>
</tr>
<tr>
<td>ball_size</td>
<td>0.085</td>
<td>ball size</td>
<td></td>
</tr>
<tr>
<td>ball_decay</td>
<td>0.94</td>
<td>ball decay</td>
<td></td>
</tr>
<tr>
<td>ball_rand</td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ball_weight</td>
<td>0.2</td>
<td>weight of the ball</td>
<td></td>
</tr>
<tr>
<td>ball_speed_max</td>
<td>2.7</td>
<td>max. ball velocity</td>
<td></td>
</tr>
<tr>
<td>ball_accel_max</td>
<td>2.7</td>
<td>max. ball acceleration</td>
<td></td>
</tr>
<tr>
<td>dash_power_rate</td>
<td>0.006</td>
<td>dash power rate</td>
<td></td>
</tr>
<tr>
<td>kick_power_rate</td>
<td>0.027</td>
<td>kick power rate</td>
<td></td>
</tr>
<tr>
<td>kickable_margin</td>
<td>0.7</td>
<td>kickable margin</td>
<td></td>
</tr>
<tr>
<td>control_radius</td>
<td></td>
<td>control radius</td>
<td></td>
</tr>
<tr>
<td>catch_probability</td>
<td>1.0</td>
<td>goalie catch probability</td>
<td></td>
</tr>
<tr>
<td>catchable_area_l</td>
<td>2.0</td>
<td>goalie catchable area length</td>
<td></td>
</tr>
<tr>
<td>catchable_area_w</td>
<td>1.0</td>
<td>goalie catchable area width</td>
<td></td>
</tr>
<tr>
<td>goalie_max_moves</td>
<td>2</td>
<td>goalie max. moves after a catch</td>
<td></td>
</tr>
<tr>
<td>maxpower</td>
<td>100</td>
<td>max power</td>
<td></td>
</tr>
</tbody>
</table>
### Table 4.14.: (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Current Value in <code>server.conf</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>minpower</code></td>
<td>-100</td>
<td></td>
<td>min power</td>
</tr>
<tr>
<td><code>maxmoment</code></td>
<td>180</td>
<td></td>
<td>max. moment</td>
</tr>
<tr>
<td><code>minmoment</code></td>
<td>-180</td>
<td></td>
<td>min. moment</td>
</tr>
<tr>
<td><code>maxneckmoment</code></td>
<td>180</td>
<td></td>
<td>max. neck moment</td>
</tr>
<tr>
<td><code>minneckmoment</code></td>
<td>-180</td>
<td></td>
<td>min. neck moment</td>
</tr>
<tr>
<td><code>maxneckang</code></td>
<td>90</td>
<td></td>
<td>max. neck angle</td>
</tr>
<tr>
<td><code>minneckang</code></td>
<td>-90</td>
<td></td>
<td>min. neck angle</td>
</tr>
<tr>
<td><code>visible_angle</code></td>
<td>90.0</td>
<td></td>
<td>visible angle</td>
</tr>
<tr>
<td><code>visible_distance</code></td>
<td>50.0</td>
<td></td>
<td>visible distance</td>
</tr>
<tr>
<td><code>audio_cut_dist</code></td>
<td>50.0</td>
<td></td>
<td>audio cut off distance</td>
</tr>
<tr>
<td><code>quantize_step</code></td>
<td>0.1</td>
<td></td>
<td>quantize step of distance for movable objects</td>
</tr>
<tr>
<td><code>quantize_step_l</code></td>
<td>0.01</td>
<td></td>
<td>quantize step of distance for landmarks</td>
</tr>
<tr>
<td><code>quantize_step_dir</code></td>
<td>1.0</td>
<td></td>
<td>corner kick margin</td>
</tr>
<tr>
<td><code>wind_dir</code></td>
<td>0.0</td>
<td>0.0</td>
<td>wind direction</td>
</tr>
<tr>
<td><code>wind_force</code></td>
<td>10.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td><code>wind_rand</code></td>
<td>0.3</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td><code>wind_random</code></td>
<td>false</td>
<td></td>
<td>wind factor is none</td>
</tr>
<tr>
<td><code>inertia_moment</code></td>
<td>5.0</td>
<td></td>
<td>inertia moment for turn</td>
</tr>
<tr>
<td><code>half_time</code></td>
<td>300</td>
<td></td>
<td>length of a half time in seconds</td>
</tr>
<tr>
<td><code>drop_ball_time</code></td>
<td>200</td>
<td></td>
<td>number of cycles to wait until dropping the ball automatically</td>
</tr>
<tr>
<td><code>port</code></td>
<td>6000</td>
<td></td>
<td>player port number</td>
</tr>
<tr>
<td><code>coach_port</code></td>
<td>6001</td>
<td>(offline) coach port</td>
<td></td>
</tr>
<tr>
<td><code>olcoach_port</code></td>
<td></td>
<td>online coach port</td>
<td></td>
</tr>
<tr>
<td><code>say_coach_cnt_max</code></td>
<td>128</td>
<td></td>
<td>upper limit of the number of online coach's message</td>
</tr>
<tr>
<td><code>say_coach_msg_size</code></td>
<td>128</td>
<td></td>
<td>upper limit of length of online coach's message</td>
</tr>
<tr>
<td><code>simulator_step</code></td>
<td>100</td>
<td>time step of simulation [unit:msec]</td>
<td></td>
</tr>
<tr>
<td><code>send_step</code></td>
<td>150</td>
<td>time step of visual information [unit:msec]</td>
<td></td>
</tr>
<tr>
<td><code>recv_step</code></td>
<td>10</td>
<td>time step of acceptance of commands [unit: msec]</td>
<td></td>
</tr>
<tr>
<td><code>sense_body_step</code></td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>say_msg_size</code></td>
<td>512</td>
<td></td>
<td>string size of say message [unit:byte]</td>
</tr>
<tr>
<td><code>clang_win_size</code></td>
<td>300</td>
<td></td>
<td>time window which controls how many messages can be sent (coach language)</td>
</tr>
<tr>
<td><code>clang_define_win</code></td>
<td>1</td>
<td></td>
<td>number of messages per window</td>
</tr>
<tr>
<td><code>clang_meta_win</code></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 4.9. Using Soccerserver

Table 4.14.: (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Current Value in <code>server.conf</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clang_advice_win</code></td>
<td>1</td>
<td></td>
<td>delay between receipt of message and send to players</td>
</tr>
<tr>
<td><code>clang_info_win</code></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>clang_message_delay</code></td>
<td>50</td>
<td></td>
<td>maximum number of coach messages sent per cycle</td>
</tr>
<tr>
<td><code>clang_message_per_cycle</code></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>hear_max</code></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>hear_inc</code></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>hear_decay</code></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>catch_ban_cycle</code></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>coach</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>coach_w_referee</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>old_coach_hear</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>send_v1_step</code></td>
<td>100</td>
<td>on</td>
<td>interval of online coach’s look</td>
</tr>
<tr>
<td><code>use_offside</code></td>
<td></td>
<td>on</td>
<td>flag for using off side rule</td>
</tr>
<tr>
<td><code>offside_active_area_size</code></td>
<td>5</td>
<td>on</td>
<td>offside active area size</td>
</tr>
<tr>
<td><code>forbid_kick_offside</code></td>
<td></td>
<td>on</td>
<td>forbid kick off side</td>
</tr>
<tr>
<td><code>log_file</code></td>
<td></td>
<td>on</td>
<td></td>
</tr>
<tr>
<td><code>record_version</code></td>
<td>3</td>
<td>on</td>
<td>flag for record log</td>
</tr>
<tr>
<td><code>record_log</code></td>
<td></td>
<td>on</td>
<td>flag for record client command log</td>
</tr>
<tr>
<td><code>record_messages</code></td>
<td></td>
<td>on</td>
<td></td>
</tr>
<tr>
<td><code>send_log</code></td>
<td></td>
<td>on</td>
<td>flag for send client command log</td>
</tr>
<tr>
<td><code>log_times</code></td>
<td></td>
<td>off</td>
<td>flag for writing cycle length to log file</td>
</tr>
<tr>
<td><code>verbose</code></td>
<td></td>
<td>off</td>
<td>flag for verbose mode</td>
</tr>
<tr>
<td><code>replay</code></td>
<td></td>
<td>off</td>
<td></td>
</tr>
<tr>
<td><code>offside_kick_margin</code></td>
<td>9.15</td>
<td>on</td>
<td>offside kick margin</td>
</tr>
<tr>
<td><code>slow_down_factor</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>start_goal_l</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>start_goal_r</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>fullstate_l</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>fullstate_r</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

51
4. Soccer Server
5. The Soccer Monitor

5.1. Introduction

Soccernonitor provides a visual interface. Using the monitor we can watch a game vividly and control the proceeding of the game. By cooperating with logplayer, soccernonitor can replay games, so that it becomes very convenient to analyze and debug clients.

5.2. Getting started

To connect the soccernonitor with the soccerserver, you can use the command following:

-> cd monitor
-> soccernonitor -f ConfFileName [-ParameterName Value] *

By specifying the arguments, you can modify the parameters of soccernonitor (See 5.6 Settings and Parameters) instead of modifying monitor configuration file.

If you use script “sserver” to start soccerserver, a monitor will be automatically started and connected with the server:

-> sserver

5.3. Communication from Server to Monitor

Soccernonitor and soccerserver are connected via UDP/IP on port 6000 (default). When the server is connected with the monitor, it will send information to the monitor every cycle. Soccerserver 7.xx provides two different formats (version 1 and version 2). The server will decide which format is used according to the initial command sent by the monitor (see 5.4). The detailed data structure information can be found in appendix C.

5.3.1. Version 1

Soccerserver and logplayer send dispinfo struct to the soccernonitor. Dispinfo struct contains a union with three different types of information:

- showinfo: information needed to draw the scene
5. The Soccer Monitor

- msginfo_t: contains the messages from the players and the referee shown in the bottom windows

- drawinfo_t: information for monitor to draw circles, lines or points (not used by the server)

The size of dispinfo_t is determined by its largest subpart (msg) and is 2052 bytes (the union causes some extra network load and may be changed in future versions). In order to keep compatibility between different platforms, values in dispinfo_t are represented by network byte order. Which information is included is determined by the mode information. NO_INFO indicates no valid info contained (never sent by the server), BLANK_MODE tells the monitor to show a blank screen (used by logplayer) (server/param.h).

```
NO_INFO  0
SHOW_MODE  1
MSG_MODE  2
DRAW_MODE  3
BLANK_MODE  4
```

Following is a description of these structs and the ones contained:

**Showinfo**

A showinfo_t struct is passed every cycle (100 ms) to the monitor and contains the state and positions of players and the ball.

```c
typedef struct {
  char   pmode ;
  team_t team[2] ;
  pos_t  pos[MAX_PLAYER * 2 + 1] ;
  short  time ;
} showinfo_t ;
```

- pmode: currently active playmode of the game (server/types.h)

```c
PM_Null,
PM_BeforeKickOff,
PM_TimeOver,
PM_PlayOn,
PM_KickOff_Left,
PM_KickOff_Right,
PM_KickIn_Left,
PM_KickIn_Right,
PM_FreeKick_Left,
PM_FreeKick_Right,
```
5.3. Communication from Server to Monitor

- **team**: information about the teams. Index 0 is for team playing from left to right.
- **pos**: position information of players. Index 0 represents the ball, indices 1 to 11 is for team[0] (left to right) and 12 to 22 for team[1].
- **time**: current time ranging from 1 to 12000 (in extra time)

```c
typedef struct {
  char name[16]; /* name of the team */
  short score; /* current score of the team */
} team_t;

typedef struct {
  short enable;
  short side;
  short unum;
  short angle;
  short x;
  short y;
} pos_t;
```

Values of the elements can be

- **enable**: state of the object. Players not on the field (and the ball) have state DISABLE. The other bits of enable allow monitors to draw the state and action of a player more detailed (server/types.h).

```c
DISABLE  (0x00)
STAND    (0x01)
KICK     (0x02)
KICK_FAULT (0x04)
GOALIE   (0x08)
CATCH    (0x10)
CATCH_FAULT (0x20)
```
5. The Soccer Monitor

- side: side the player is playing on. LEFT means from left to right, NEUTRAL is the ball (server/types.h).

  LEFT 1
  NEUTRAL 0
  RIGHT -1

- unum: uniform number of a player ranging from 1 to 11

- angle: angle the agent is facing ranging from -180 to 180 degrees, where -180 is view to the left side of the screen, -90 to the top, 0 to the right and 90 to the bottom.

- x, y: position of the player on the screen. (0, 0) is the midpoint of the field, x increases to the right, y to the bottom of the screen. Values are multiplied by SHOWINFO_SCALE (16) to reduce aliasing, so field size is PITCH_LENGTH * SHOWINFO_SCALE in x direction and PITCH_WIDTH * SHOWINFO_SCALE in y direction.

Messageinfo
Information containing the messages of players and the referee.

typedef struct {
  short board ;
  char message[2048] ;
} msginfo_t ;

- board: indicates the type of message. A message with type MSG_BOARD is a message of the referee for the left text window, LOG_BOARD are messages from and to the players. (server/param.h)

  MSG_BOARD 1
  LOG_BOARD 2

- message: zero terminated string containing the message.

Drawinfo
Allows to specify information for the monitor to draw circles, lines or points.

5.3.2. Version 2
Soccerserver and logplayer send dispinfo_t2 structs to the soccermonitor instead of dispinfo_t structs which is used in version 1. Dispinfo_t2 contains a union with five different types of information (the data structures are printed in appendix C):
5.4. Communication from Monitor to Server

- showinfo_t2: information needed to draw the scene. It includes all information on coordinates and speed of players and the ball, teamnames, scores, etc.

- msginfo_t: contains the messages from the players and the referee shown in the bottom windows. It also contains information on team’s images and information on player exchanges.
  - team graphic: The team graphic format requires a 256x64 image to be broken up into 8x8 tiles and has the form
    
    \[
    \text{team\_graphic\_{\{l|r\}} \ (\langle X \rangle \ \langle Y \rangle \ \"\langle XPM \ line\rangle\" \ \ldots \ \"\langle XPM \ line\rangle\"})
    \]
    
    Where X and Y are the co-ordinates of the 8x8 tile in the complete 256x64 image, starting at 0 and ranging up to 31 and 7 respectively. Each XPM line is a line from the 8x8 xpm tile.
  - substitutions: substitutions are now explicitly recorded in the message board in the form
    
    \[
    \text{change\_player\_type \{l|r\} \ <\unum> \ <\player\_type>}
    \]

- player_type_t: information describing different player’s abilities and tradeoffs

- server_params_t: parameters and configurations of soccerserver

- player_params_t: parameters of players

Which information is contained in the union is determined by the mode field. NO_INFO indicates no valid info contained (never sent by the server). BLANK_MODE tells the monitor to show a blank screen (used by logplayer) (server/param.h).

- NO_INFO 0
- SHOW_MODE 1
- MSG_MODE 2
- BLANK_MODE 4
- PT_MODE 7
- PARAM_MODE 8
- PPARAM_MODE 9

5.4. Communication from Monitor to Server

The monitor can send to the server the following commands (in all commands, <variable> has to be replaced with proper values):

- (dispinit) \mid (dispinit version <version>)

sent to the server as first message to register as monitor (opposed to a player, that connects on port 6000 as well). ”(dispinit)” is for information version 1, while ”(dispinit version 2)” is for version 2. You can change the version by setting the according monitor parameter. (See 5.6 Parameters and Configurations)
5. The Soccer Monitor

(sent to start (kick off) a game, start the second half or extended time. Ignored, when
the game is already running.

(x) (y) (side)
sent to indicate a foul situation. x and y are the coordinates of the foul, side is LEFT
(1) for a free kick for the left team, NEUTRAL (0) for a drop ball and RIGHT (-1) for
a free kick for the right team.

(sent to show a player the red card (kick him out). side can be LEFT or RIGHT, unum
is the number of the player (1 - 11).

(side) (unum)
sent to place player at certain position with certain body angle, side can be LEFT (1) or
RIGHT (-1), unum is the number of the player (1 - 11). Posx and posy indicate the new
position of the player, which will be divided by SHOWINFO_SCALE. And ang indicate
the new angle of a player in degrees. This command is added in soccerserver 7.02.

(compression <level>)
The server supports compression of communication with its clients and monitors (since
version 8.03). A monitor can send the above compression request to the server to start
compressed communication. If the server is compiled without ZLib, the server will re-
spond with (warning compression unsupported) else if <level> is not a number be-
tween 0 and 9 inclusive, the server will respond with (error illegal command form)
else the server will respond with (ok compression <level>) and all subsequent mes-
sages to that client will be compressed at that level, until a new compression command
is received. If a compression level above zero is selected, then the monitor is expected
to compress its commands to the server. Specifying a level of zero turns off compression
completely (default).

5.5. How to record and playback a game
To record games, you can call server with the argument:

-record LOGFILE
(LOGFILE is the logfile name) or set the parameter in server.conf file:

record.log : on.
To specify the logfile version, you can call server with the argument:

-record_version [1/2/3]
or set the parameter in server.conf file:

```plaintext
record_version : 2
```

The logplayer allows you to replay recorded games. Logfiles can be read in by the logplayer and sent to the connected soccermonitors. To replay logfiles just call logplayer with the logfile name as argument, start a soccermonitor and then use the buttons on the logplayer window to start, stop, play backward, play stepwise.

### 5.5.1. Version 1 Protocol

Logfiles of version 1 (server versions up to 4.16) are a stream of consecutive `dispinfo_t` chunks. Due to the structure of `dispinfo_t` as a union, a lot of bytes have been wasted leading to impractical logfile sizes. This lead to the introduction of a new logfile format 2.

### 5.5.2. Version 2 Protocol

Version 2 logfile protocol tries to avoid redundant or unused data for the price of not having uniform data structs. The format is as follows:

- **head of the file:**
  - the head of the file is used to autodetect the version of the logfile. If there is no head, Unix-version 1 is assumed. 3 chars 'ULG': indicating that this is a Unix logfile (to distinguish from Windows format)
  - `char version`:
    - version of the logfile format
  - `body`:
    - the rest of the file contains the data in chunks of the following format:
  - `short mode`:
    - this is the mode part of the `dispinfo_t` struct (see 5.5.1 Version 1) SHOW_MODE for `showinfo_t` information MSG_MODE for `msginfo_t` information
      - If mode is SHOW_MODE, a `showinfo_t` struct is following.
      - If mode is MSG_MODE, next bytes are
        * short board: containing the board info
        * short length: containing the length of the message (including zero terminator)
        * string msg: length chars containing the message

Other info such as DRAW_MODE and BLANK_MODE are not saved to logfiles. There is still room for optimization of space. The team names could be part of the head of the file and only stored once. The unum part of a player could be implicitly taken from array indices.
5. *The Soccer Monitor*

Be aware of, that information chunks in version 2 do not have the same size, so you can not just seek SIZE bytes back in the stream when playing logfiles backward. You have to read in the whole file at once or (as is done) have at least to save stream positions of the showinfo_t chunks to be able to play logfiles backward.

In order to keep compatibility between different platforms, values are represented by network byte order.

5.5.3. *Version 3 Protocol*

The version 3 protocol contains player parameter information for heterogenous players and optimizes space. The format is as follows:

- head of the file:
  Just like version 2, the file starts with the magig characters 'ULG'.

- char version:
  version of the logfile format, i.e. 3

- body:
  The rest of the file contains shorts that specify which data structures will follow.

  - If the short is PM_MODE,
    * a char specifying the play mode follows.
    This is only written when the playmode changes.
  
  - If the short is TEAM_MODE,
    * a team_t struct for the left side and
    * a team_t struct for the right side follow.
    Team data is only written if a new team connects or the score changes.

  - If the short is SHOW_MODE,
    * a short_showinfo_t2 struct specifying ball and player positions and states follows.

  - If the short is MSG_MODE,
    * a short specifying the message board,
    * a short specifying the length of the message,
    * a string containing the message will follow.

  - If the short is PARAM_MODE,
    * a server_params_t struct specifying the current server parameters follows.
    This is only written once at the beginning of the logfile.

  - If the short is PPARAM_MODE,
    * a player_params_t struct specifying the current hetro player parameters.
    This is only written once at the beginning of the logfile.
If the short is PT_MODE,
* a player_type_t struct specifying the parameters of a specific player type follows.

This is only written once for each player type at the beginning of the logfile.

Data Conversion:
- Values such as x, y positions are meters multiplied by SHOWINFO SCALE2.
- Values such as deltax, deltay are meters/cycle multiplied by SHOWINFO SCALE2.
- Values such as body_angle, head_angle and view_width are in radians multiplied by SHOWINFO SCALE2.
- Other values such as stamina, effort and recovery have also been multiplied by SHOWINFO SCALE2.

5.6. **Settings and Parameters**

Soccermonitor has the following modifiable parameters:

- "Used Value" is the current value of the parameter which is encoded in the monitor.conf file. "Default Value" is the value encoded in the source files and will be used if the user doesn’t give one.

You can specify parameters described in the table above in command line as following: You can also modify the parameters by specifying them in configuration file monitor.conf. In the configuration file, each line consists a pair of name and value of a parameter as follows: ParameterName : Value Lines that start with ‘’#’’ are comment lines.

5.7. **What’s New**

8.03 :
- The server supports compressed communication to monitors as described in section 5.4
- Player substitution information is added to the message log
- Team graphics information is added to the message log

7.07 :
- The logplayer did not send server param, player param, and player type messages. This has been fixed.
- The monitor would crash on some logfiles because stamina_max seemed to be set to 0. The monitor will no longer crash this way.
5. *The Soccer Monitor*

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Used Value</th>
<th>Default</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>localhost</td>
<td>Localhost</td>
<td>hostname of soccerserver</td>
</tr>
<tr>
<td>port</td>
<td>6000</td>
<td>6000</td>
<td>port number of soccerserver</td>
</tr>
<tr>
<td>version</td>
<td>2</td>
<td>1</td>
<td>monitor protocol version</td>
</tr>
<tr>
<td>length,magnify</td>
<td>6.0</td>
<td>6.0</td>
<td>magnification of size of field</td>
</tr>
<tr>
<td>goal_width</td>
<td>14.02</td>
<td>7.32</td>
<td>goal width</td>
</tr>
<tr>
<td>print_log</td>
<td>off</td>
<td>On</td>
<td>flag for display log of communication [on/off]</td>
</tr>
<tr>
<td>Log_line</td>
<td>6</td>
<td>6</td>
<td>size of log window</td>
</tr>
<tr>
<td>Print_mark</td>
<td>on</td>
<td>On</td>
<td>flag for display mark on field [on/off]</td>
</tr>
<tr>
<td>mark_file_name</td>
<td>mark.RoboCup.grey.xbm</td>
<td>Mark.xbm</td>
<td>mark on field use file name</td>
</tr>
<tr>
<td>ball_file_name</td>
<td>ball-s.xbm</td>
<td>Ball.xbm</td>
<td>ball use file name</td>
</tr>
<tr>
<td>player_widget_size</td>
<td>9.0</td>
<td>1.0</td>
<td>size of player widget</td>
</tr>
<tr>
<td>player_widget_font</td>
<td>5x8</td>
<td>Fixed</td>
<td>font(uniform number) of player widget</td>
</tr>
<tr>
<td>Uniform_num_pos_x</td>
<td>2</td>
<td>2</td>
<td>position (X) of player uniform number</td>
</tr>
<tr>
<td>Uniform_num_pos_y</td>
<td>8</td>
<td>8</td>
<td>position (Y) of player uniform number</td>
</tr>
<tr>
<td>Team_L_color</td>
<td>Gold</td>
<td>Gold</td>
<td>Team_L color</td>
</tr>
<tr>
<td>team_R_color</td>
<td>Red</td>
<td>Red</td>
<td>Team_R color</td>
</tr>
<tr>
<td>goalie_L_color</td>
<td>Green</td>
<td>Green</td>
<td>Team_L Goalie color</td>
</tr>
<tr>
<td>goalie_R_color</td>
<td>Purple</td>
<td>Purple</td>
<td>Team_R Goalie color</td>
</tr>
<tr>
<td>neck_L_color</td>
<td>Black</td>
<td>Black</td>
<td>Team_L Neck color</td>
</tr>
<tr>
<td>neck_R_color</td>
<td>Black</td>
<td>Black</td>
<td>Team_R Neck color</td>
</tr>
<tr>
<td>Goalie_neck_L_color</td>
<td>Black</td>
<td>Black</td>
<td>Team_L Goalie Neck color</td>
</tr>
<tr>
<td>Goalie_neck_R_color</td>
<td>Black</td>
<td>Black</td>
<td>Team_R Goalie Neck color</td>
</tr>
<tr>
<td>status_font</td>
<td>7x14bold</td>
<td>Fixed</td>
<td>status line font [team name and score,time,play_mode]</td>
</tr>
<tr>
<td>popup_msg</td>
<td>off</td>
<td>Off</td>
<td>flag for pop up and down “GOAL!!” and “Offside!” [on/off]</td>
</tr>
<tr>
<td>Goal_label_width</td>
<td>120</td>
<td>120</td>
<td>pop up and down “GOAL!!” label width</td>
</tr>
<tr>
<td>Goal_label_font</td>
<td>-adobe-times-bold-r-.<em>-34-.</em>-.<em>-.</em>-.<em>-</em>-<em>-</em>-<em>-</em>-*</td>
<td>Fixed</td>
<td>pop up and down “GOAL!!” label font</td>
</tr>
<tr>
<td>Goal_score_width</td>
<td>40</td>
<td>40</td>
<td>pop up and down “GOAL!!” score width</td>
</tr>
<tr>
<td>Goal_score_font</td>
<td>-adobe-times-bold-r-.<em>-25-.</em>-.<em>-.</em>-.<em>-.</em>-<em>-</em>-<em>-</em>-<em>-</em></td>
<td>Fixed</td>
<td>pop up and down “GOAL!!” score font</td>
</tr>
<tr>
<td>Offside_label_width</td>
<td>120</td>
<td>120</td>
<td>pop up and down “Offside!” label width</td>
</tr>
<tr>
<td>Offside_label_font</td>
<td>-adobe-times-bold-r-.<em>-34-.</em>-.<em>-.</em>-.<em>-.</em>-<em>-</em>-<em>-</em>-<em>-</em></td>
<td>Fixed</td>
<td>pop up and down “Offside!” label font</td>
</tr>
<tr>
<td>eval</td>
<td>off</td>
<td>Off</td>
<td>flag for evaluation mode</td>
</tr>
<tr>
<td>redraw_player</td>
<td>on</td>
<td>Off</td>
<td>always redraw player (needed for RH 5.2)</td>
</tr>
</tbody>
</table>
5.7. What’s New

7.05:

- For quite some time, the logplayer has occasionally “skipped” so that certain cycles were never displayed by the logplayer. This seems to be caused by the logplayer sending too many UDP packets for the monitor to receive. Therefore, a new parameter has been added to the logplayer ‘message\_delay\_interval’. After sending that many messages, the logplayer sleeps for 1 microsecond, giving the monitor a chance to catch up. This is not a guaranteed to work, but it seems to help significantly. If you still have a problem with the logplayer/monitor “skipping”, try reducing message\_delay\_interval from it’s default value of 10. Setting message\_delay\_interval to a negative number causes there to be no delay.

- The server used to truncate messages received from the players and coach to 128 characters before recording them in the logfile. This has been fixed.

7.04:

- If a client connects with version > 7.0, all angles sent out by the server are rounded instead of truncated (as they were previously) This makes the error from quantization of angles (i.e. conversion of floats to ints) both uniform throughout the domain and two sided. This change was also made to all values put into the dispinfo structure for the monitors and logfiles.

7.02:

- A new command has been added to the monitor protocol:
  (dispplayer side unum posx posy ang)
  (contributed by Artur Merke)
  See 5.4 Commands From Monitor to Server

7.00:

- Included the head angle into the display of the soccermonitor. (source contributed by Ken Nguyen)
- Included visualization effect when the player collided with the ball or the player collided with another player. The monitor displays both cases with a black circle around the player.
- Introduced new monitor protocol version 2. (See 5.5.2 Version 2 and 5.4 Commands From Monitor to Server)
- Introduced new logging protocol version 3. (See 5.5.3 Version 3 Protocol)
- Fixed logging so that the last cycle of a game is logged.
5. The Soccer Monitor
6. Soccer Client

6.1. Protocols

This section provides a brief overview of the protocol between the Soccer Client and the Soccer Server. More details on these protocols can be found in the Soccer Server section. Note that the init and reconnect commands should be send to the player’s UDP port (default: 6000) of the Soccer Server machine, and after the response they should be sent to the port assigned to your player by the server, in a valid format. The server sends the init response from this port (refer to section 1.2.1). All the commands sent to or received from the server are strings of common character and are in a pair of parenthesis.

6.1.1. Initialization and Reconnection

Every player wanting to connect to the server should introduce himself. This is like a handshake and is done only at the beginning and optionally in the half time when you want to reconnect.

 Initialization

Your client should send an init command to the server in the following format:

\[(\text{init TeamName} \ [(\text{version VerNum)]) \ [(\text{goalie})]\
\]

The goalie should include the ”(goalie)” in the init command to be allowed by the server to catch the ball or do another special goalie action. Note there can only be one or no goalie in each team. (You are not obliged to use a goalie)

The Server welcomes you with a response to your init message in the following format:

\[(\text{init Side UniformNumber PlayMode})\]

Or by an error message (if there is an error, i.e. you have initiated more than two team, more than 11 players in a team or more than one goalie in a team):

\[(\text{error no_more_team_or_player_or_goalie})\]

Side is your team’s side of play, a character, l(left) or r(right). UniformNumber is the player’s uniform number (the players of each team are known by their uniform number). PlayMode is a string representing one of the valid play modes.
6. Soccer Client

If you connect to server with versions 7.00 or higher you will receive additional server
parameters, player parameters and player types information (the last two are related
to the hetero players feature). For the exact format refer to the appendix.

(server_param Parameters . . . )

(player_param Parameters . . . )

(player_type id Parameters . . . )

Here the hand shaking is finished and your client is known as a valid player.

Reconnection

Reconnection is useful for changing the client program of a player without restarting the
game. It can only be done in a non-PlayOn playing mode (e.g. in the half time).

For reconnection you should send a reconnect command in the following format:

(reconnect TeamName UniformNumber)

And you will receive a response in the following format:

(reconnect Side PlayMode)

Or one of the following errors:

(can’t reconnect)

if the game is in the PlayOn mode.

(error reconnect)

when no client reconnected due to an error. You may also receive the following error
if the team name is invalid (error no more team or player or goalie)

Here again if you are connecting to the server with version 7.00 or higher you will
receive additional server parameters, player parameters and player types information.

Disconnection

Before you disconnect, you can send a bye command to the server. This command will
remove the player from the field.

(bye)

There will be no answers from the server.
Version Control

Due to the progressive development of the Soccer Server, new features have been added every year and this resulted in changes and improvements in the protocols to support these features. In order to keep compatibility with the older clients and making it easier to work with (specially for researchers), a system has been implemented for the Protocols Version Control. Every client should tell the server the version of its communication protocol in the `init` command so that the server would be able to send the messages in the proper format.

But note that although the communication protocol remains unchanged, the judgment and the simulation rules may change and this will affect the whole game.

6.1.2. Control Commands

During the game each player can send action commands. The server executes the commands at the end of the cycle and simulates the next cycle regarding the received commands and the previous cycles data.

Body Commands

All the playing and movement behaviors of the player are consisted from a few commands known as body commands that are presented briefly below.

The results of these commands are a little complicated and depend on many simulation factors. For the details of the execution of each command refer to the Soccer Server Section.

(turn Moment)

The Moment is in degrees from $-180$ to $180$. This command will turn the player’s body direction Moment degrees relative to the current direction.

(dash Power)

This command accelerates the player in the direction of its body (not direction of the current speed). The Power is between minpower (used value: $-100$) and maxpower (used value: $100$).

(kick Power Direction)

Accelerates the ball with the given Power in the given Direction. The direction is relative to the the Direction of the body of the player and the power is again between minpower and maxparam.

(catch Direction)
6. Soccer Client

Goalie special command: Tries to catch the ball in the given Direction relative to its body direction. If the catch is successful the ball will be in the goalie’s hand until kicked away.

(move X Y)

This command can be executed only before kick off and after a goal. It moves the player to the exact position of X (between $-54$ and $54$) and Y (between $-32$ and $32$) in one simulation cycle. This is useful for before kick off arrangements.

Note that in each simulation cycle, only one of the above five commands can be executed (i.e. if the client sends more than one command in a single cycle, one of them will be executed randomly, usually the one received first)

(turn_neck Angle)

This command can be sent (and will be executed) each cycle independently, along with other action commands. The neck will rotate with the given Angle relative to previous Angle. Note that the resulting neck angle will be between \texttt{minneckang} (default: $-90$) and \texttt{maxneckang} (default: 90) relative to the player’s body direction.

Communication Commands

The only way of communication between two players is broadcasting of messages through the \texttt{say} command and hearing through the \texttt{hear} sensor.

(say Message)

This command broadcasts the Message through the field, and any player near enough (specified with \texttt{audio\_cut\_dist}, default: 50.0 meters), with enough hearing capacity will hear the Message. The message is a string of valid characters.

(ok say)

Command succeeded.

In case of error there will be the following response from the Server:

(error illegal\_command\_form)

Misc. Commands

Other commands are usually of two forms:

- Data Request Commands

  (sense\_body)
6.1. Protocols

Requests the server to send sense body information. Note the server sends sense body information every cycle if you connect with version 6.00 or higher.

(score)

Request the server to send score information. The server’s reply will be in this format:

(score Time OurScore OpponentScore)

• Mode Change Commands

(change_view Width Quality)

Changes the view parameters of the player. Width is one of narrow, normal or wide and Quality is one of high or low. The amount and detail of the information returned by the visual sensor depends on the width of the view and the quality. But note that the frequency of sending information also depends on these parameters (e.g. if you change the quality from high to low, the frequency doubles, and the time between two see sensors will be cut to half).

6.1.3. Sensor Information

Sensor information are the messages that are sent to all players regularly (e.g. each cycle or each one and a half a cycle). There is no need to send any message to the server to get these information.

All the returned information of the sensors have a time label, indication the cycle number of the game when the data have been sent (indicated by Time). This time is very useful.

Visual Sensor

Visual Sensor is the most important sensor, and is a little bit complicated. This sensor returns information about the objects that can be seen from the player’s view (i.e. objects that are in the view angle and not very far).

The main format of the information is:

(see Time ObjInfo ObjInfo . . .)

The ObjInfos are of the format below:

(ObjName Distance Direction [DistChange DirChange [BodyFacingDir HeadFacingDir]])

or
6. Soccer Client

(ObjName Direction)

Note that the amount of information returned for each object depends on its distance. The more distant the object is the less information you get. For more detailed information regarding ObjInfo refer to Appendix.

ObjName is in one of the following formats:

(p [TeamName [Unum]])
(b)
(f FlagInfo)
(g Side)

p stands for player, b stands for ball, f stands for flag and g stands for goal.

Side is one of l for left or r for right. For more information on FlagInfo refer to Appendix.

Audio Sensor

Audio sensor returns the messages that can be heard through the field. They may come from the online coach, referee, or other players.

The format is as follows:

(hear Time Sender Message)

Sender is one of the followings:

self: when the sender is yourself.
referee: when the sender is the referee of the game.
online_coach_l or online_coach_r

Direction: when the sender is a player other than yourself the relative direction of the sender is returned instead.

Body Sensor

Body sensor returns all the states of the player such as remaining stamina, view mode and the speed of the player at the beginning of each cycle:

(sense_body Time (view_mode { high | low } { narrow | normal | wide }) (stamina Stamina Effort) (speed Speed Angle) (head_angle Angle) (kick Count) (dash Count) (turn Count) (say Count) (turn_neck Count) (catch Count) (move Count) (change_view Count))

The last eight parameters are counters of the received commands. Use the counters to keep track of lost or delayed messages.

6.2. How to Create Clients

This section provides a brief description to write a first-step program of soccer client.
6.2. How to Create Clients

6.2.1. Sample Client

The Soccer Server distribution includes a very simple program for soccer clients, called `sampleclient`. It is under the "sampleclient" directory of the distribution, and is automatically compiled when you make the Soccer Server.

The `sampleclient` is not a stand-alone client: It is a simple 'pipe' that redirects commands from its standard input to the server, and information from the server to its standard output. Therefore, nothing happens when users invoke the `sampleclient`. The users must type-in commands from keyboards, and read the sensor information displayed on the terminal. (Actually it is impossible to read sensor information, because the server sends about 17 sensor informations (see information and sense_body information) per second.)

The `sampleclient` is useful to understand what clients should do, and what the clients will receive from the server.

How to Use `sampleclient`

Here is a typical usage of the `sampleclient`.

1. Invoke `client` under `sampleclient` directory of the Soccer Server.

   ```
   % ./client SERVERHOST
   ```

   Here, `SERVERHOST` is a hostname on which Soccer Server is running.

   Then the program awaits user input.

   If the Soccer Server uses an unusual port, for example 6005, instead of the standard port (6000), the users should use the following form.

   ```
   % ./client SERVERHOST 6005
   ```

2. Type in `init` command from the keyboard.

   ```
   (init MYTEAMNAME (version 7))
   ```

   Here `MYTEAMNAME` is a team name the users want to use.

   Then a player appears on the field. In the same time, the program starts to output the sensor information sent from the server to the terminal. Here is a typical output:

   ```
   send 6000 : (init foo (version 7))
   recv 1567 : (init r 1 before_kick_off)
   recv 1567 : (server_param 14.02 5 0.3 0.4 0.1 60 1 1 4000 45 0 0.3 0.5 ... 
   recv 1567 : (player_param 7 3 3 0 0.2 -100 0 0.2 25 0 0.002 -100 0 0.2 ... 
   recv 1567 : (player_type 0 1.16432 28.5679 0.533438 8.33595 0.00733326 ... 
   recv 1567 : (player_type 1 1.19861 25.1387 0.437196 5.92991 0.00717675 ... 
   ```
6. Soccer Client

recv 1567 : (player_type 3 1.04904 40.0956 0.436023 5.90057 0.00631769 ...
recv 1567 : (player_type 4 1.1723 27.7704 0.568306 9.20764 0.00746072 ...
recv 1567 : (player_type 5 1.12561 32.4392 0.402203 5.05509 0.00621539 ...
recv 1567 : (player_type 6 1.02919 42.0812 0.581564 9.53909 0.00688457 ...
recv 1567 : (sense_body 0 (view_mode high normal) (stamina 4000 1) ...
recv 1567 : (see 0 ((g r) 61.6 37) ((f r t) 49.4 3) ((f p r t) 37 27) ...
recv 1567 : (sense_body 0 (view_mode high normal) (stamina 4000 1) ...
...

The first line, “send 6000 : (init foo (version 7))”, is a report what the client sends to the server. The second line, recv 1567 : (init r 1 before kick off) is a report of the first response from the server. Here, the server tells the client that the assigned player is the right side team (r), its uniform number is 1, and the current playmode is before kick off. The next 9 lines are server param and player param, which tells various parameters used in the simulation. Finally, the server starts to send the normal sensor informations, sense body and see. Because the server sends these sensor information every 100ms or 150ms, the client continues to output the information endlessly.

3. Type in move command to place the player to the initial position. The player appears on a bench outside of the field. Users need to move it to its initial position by move command like:

(move -10 10)

Then the player moves to the point (-10,10).

Because, as mentioned before, the client program outputs sensor information endlessly, users can not see strings they type in. So, they must type-in commands blindly. ¹

4. Click ‘Kick-Off’ button on the Soccer Server. Then the game starts. The users can see that the time data in each sensor information (the first number of see and sense body information) are increasing.

5. After then, users can use any normal command, turn, dash, kick and so on. For example, users can turn the player to the right by typing:

(turn 90)

The player can dash forward with full power by typing:

(dash 100)

¹Users can redirect the output to any file or program. For example, you can redirect it to /dev/null to discard the information by invoking “% client SERVERHOST > /dev/null”. Then, the users can see the string they type-in.
When the player is near enough to the ball, it can kick the ball to the left with power 50 by:

\[(\text{kick 50 -90})\]

Note again that because of endless sensor output, users must type-in these commands blindly.

**Overall Structure of Sample Client**

The structure of the sampleclient is simple. The brief process the client does is as follows:

1. Open a UDP socket and connect to the server port. (init\_connection())

2. Enter the read-write loop (message\_loop), in which the following two processes are executed in parallel.
   - read user's input from the standard input (usually a keyboard) and send it to the server (send\_message()).
   - receive the sensor information from the server (receive\_message()) and output it to the standard output (usually a console).

In order to realize the parallel execution, sampleclient uses the select() function. The function enables to wait for multiple input from sockets and streams in a single process. When select() is called, it waits until one of the sockets and streams gets input data, and tells which sockets or streams got the data. For more details of the usage of select(), please refer to the man page or manual documents.

An important tip in the sampleclient is that the client must change the server’s port number when it receives sensor informations from the server. This is because the server assign a new port to a client when it receives an init command. This is done by the following statement in "client.c" (around line 147):

```c
printf( "recv %d : ", ntohs(serv_addr.sin_port));
+ sock->serv_addr.sin_port = serv_addr.sin_port ;
buf[n] = '\0';
```

**6.2.2. Simple Clients**

In order to develop complete soccer clients, what users must do is to write code of a ‘brain’ part, which performs the same thing as users do with the sampleclient described in the previous section. In other words, users must write a code to generate command strings to send to the server based on received sensor information.

Of course it is not a simple task (so that many researchers tackle RoboCup as a research issue), and there are various ways to implement it. Simply saying, in order to develop player clients, users need to realize the following functions:
6. Soccer Client

[Sensing] To analyze sensor information: As shown in the previous section, the server sends various sensor information in S-expressions. Therefore, a client needs to parse the S-expressions. Then, the client must analyze the information to get a certain internal representation. For example, the client needs to analyze a visual information to estimate player’s location and field status, because the visual information only include relative locations of landmarks and moving objects on the field.

[Action Interval] To control interval of sending commands: Because the server accepts a body command (turn, dash and kick) per 100ms, the client needs to wait appropriate interval before sending a command.

[Parallelism] To execute sensor and action processes in parallel: Because the Soccer Server processes sensor information and command asynchronously, clients need to execute a sensor process, which deals with sensor information, and an action process, which controls to send commands, in parallel.

[Planning] To make a plan of play: Using sensor information, the client needs to generate appropriate command sequences of play. Of course, this is the final goal of developing soccer clients!!

Here are two simple examples of stand-alone players, sclient1 and sclient2, which just chase the ball and kick it to the opponent goal. The sources are available from:


In the examples, the functions listed above are realized as follows:

- For Sensing function, both examples use common facilities of class BasePlayer, class FieldState, and estimatePos functions. By these facilities, the example programs do:
  - receive data from a socket connected with the server,
  - parse the data as S-expression,
  - interpret the expression into internal data format (class SensorInfo),
  - and in the case the received data is visual sensor information, estimate player’s and other object’s positions.

For more detail, please read the source code.

- For Action Interval and Parallelism functions, the two examples use different methods. The first example, sclient1 uses timeout of select() function. The second one, sclient2 uses the multi-thread (pthread) facility. These are described below.

- For Planning function, both examples have very simple planners as follows:
  - If the player does not see the ball in recent 10 steps, or if the player can not estimate its position in recent 10 steps, it looks around.
6.2. How to Create Clients

- If the ball is in kickable area, it kicks the ball to the opponent goal.
- Otherwise, the player rushes to the ball (turns to the ball and dashes).

sclient1

The sclient1 uses the timeout facility of select() function to realize Action Interval and Parallelism.

The key part of the program is in MyPlayer::run(). Here is the part of the source code:

```cpp
//----------------------------------------
// enter main loop
(SocketReadSelector selector;

TimeVal nexttic ; // indicate the timestamp for next command send
nexttic.update() ; // set nexttic to the current time.

while(True) {

    //--------------------------------------------------
    // setup selector
    selector.clear() ;
    selector.set(socket) ;

    //--------------------------------------------------
    // wait socket input or timeout (100ms) ;

    Int r = selector.selectUntil(nexttic) ;

    if(r == 0) { // in the of timeout. (no sensor input)
        doAction() ; // enter action part
        nexttic += TimeVal(0,100,0) ; // increase nexttic to 100ms
    } else { // got some input
        doSensing() ; // enter sensor part
    }
}
```

Here, class SocketReadSelector is a class to abstract facilities of select() and is defined in "itk/Socket.h". In the line “Int r = selector.selectUntil(nexttic) ;”, the program awaits the socket input or timeout indicated by nexttic, which holds the timestamp of the next tic (simulation step). The function returns 0 if timeout, or the number of receiving sockets. In the case of timeout, the program calls doAction() in
6. Soccer Client

which a command is generated and sent to the server, or otherwise, it calls doSensing() in which a sensor information is processed.

sclient2

The sclient2 uses the POSIX thread (pthread) facilities to realize Action Interval and Parallelism.

The key part of the program is also in MyPlayer::run(). Here is the part of the source code:

```
//----------------------------------------
// fork sensor thread
forkSensor();
//----------------------------------------
// main loop
while(True) {
    if (!isBallSeenRecently(10)) {
        //------------------------------
        // if ball is not seen recently
        // look around by (turn 60)
        for(UInt i = 0 ; i < 6 ; i++) {
            turn(60);
        }
    } else if (kickable()) {
        ...
    }
}
```

The statement “forkSensor();” invokes a new thread for receiving and analyzing the sensor information. (The behavior of the sensor thread are defined in "SimpleClient.*" and "ThreadedClient.*".) Then the main thread enters the main loop in which action sequences of “chasing the ball and kick to the goal” are generated. Because Sensing function is handled in the sensor thread in parallel, the main thread needs not take care of the sensor input.

In order to keep action interval to be 100ms, the sclient2 waits for the next simulation step by the function ThreadedPlayer::sendCommandPre() defined in ”ThreadedPlayer.cc” as follows:

```
Bool ThreadedPlayer::sendCommandPre(Bool bodyp) {
    cvSend.lock() ;
```
6.2. How to Create Clients

if(bodyp) {
    while(nextSendBodyTime.isFuture())
        cvSend.waitUntil(nextSendBodyTime);
}

while(nextSendTime.isFuture()) {
    cvSend.waitUntil(nextSendTime);
}

return True;
}

In this function, MutexCondVar cvSend provide a similar timeout facility of select() function used in sclient1 described above. (MutexCondVar is a combination of condition variable (pthread_cond_t) and mutex (pthread_mutex_), and is defined in "itk/MutexCondVar.h"). Because the function is called just before the player sends a command to the server, and nextSendBodyTime is controlled to indicate the timestamp of the next simulation step, the thread waits to send a command in the next tic.

6.2.3. Tips

Here we collect tips to develop soccer client programs.

- Debugging is the main problem in developing your own team. So try to find easy debugging methods.

- A nice and simple way to see your program’s variables in a condition is to use an abort() command or some asserts to force the program to core-dump; And debug the core using gdb.

- Log every message received from the server and sent to the server. It is very useful for debugging.

- Using ready to use libraries for socket and parsing problems is useful if you are a beginner.

- Remember to pass the version number to the server in the init command. Although it is optional, the default is 3.00 which usually is not desired.

- Even if the catch probability is 1.00 your catch command may be unsuccessful because of errors in returned sensors about the positions.

- The first serious problem you may encounter is the timing problem. There are many methods to synchronize your client’s time with server. One simple methods is to use received sense body information.
6. Soccer Client

- Beware of slow networks. If your timing is not very powerful your client’s will behave abnormally in a crowded or slow network or if they are out of process resources (e.g. you run many clients on one slow machine). In this case they may see older positions and will try to act in these positions and this will result in confusion (e.g. they will turn around themselves).

- The main usage of flags are for the player to find the position of himself in the field. Your very first clients may ignore flags and play with relative system of positions. But you may need a positioning module in the near future. There are many of the in the ready to use libraries.

- The program should check the end of buffer in analyzing sensor information. The sensor information uses S-expressions. But the expression may not be completed when the sensor data is longer than the buffer, so that some closing parentheses are lost. In this case, the program may core-dump if it parses the expression naively.


7. The coach

7.1. Introduction

Coaches are privileged clients used to provide assistance to the players. There are two kinds of coaches, the online coach and the trainer. The latter is often called ‘off-line coach’ as well, but for clarity sake we will use the term ‘trainer’.

7.2. Distinction between trainer and online coach

In general, the trainer can exercise more control over the game and may be used only in the development stage, whereas the online coach may connect to official games. The trainer is useful during development for such tasks as running automated learning or managing games. The on-line coach is used during games to provide additional advice and information to the players.

While developing player clients, for example when applying machine learning methods to learn skills like dribbling or kicking, it might be useful to create training sessions in an automated way. Therefore, the trainer has the following capabilities:

- It can control the play-mode.

- It can broadcast audio messages. Such a message can consist of a command or some information intended for one or more of the player-clients. Its syntax and interpretation are user-defined.

- It can move the players and the ball to any location on the field and set their directions and velocities.

- It can get noise-free information about the movable objects.

For details on these capabilities see Section 7.3.

The online coach is intended to observe the game and provide advice and information to the players. Therefore, it’s capabilities are somewhat limited:

- It can communicate with the players.

- It can get noise-free information about the movable objects.
7. The coach

To prevent the coach from controlling each client in a centralized way, communication is restricted in several ways as described in Section 7.7. The online coach is a good tool for opponent modelling, game analysis, and giving strategic tips to its teammates. Since the coach gets a noise-free, global view over the field and has less real-time demands, it is expected that it can spend more time deliberating over strategies. See Section 7.6 for more details about the online coach.

7.3. Trainer

7.3.1. Connecting with and without the soccerserver referee

By default, an internal referee module is active within the soccerserver that controls the match (see Section 4.7). If the trainer should have complete control over the match, the soccerserver must be instructed to deactivate the referee module. This means for example, that the play-mode will not change and players will not be moved back to their sides after a goal. The trainer has to react to these events by its own rules.

The soccerserver must be informed at startup-time that a trainer-client will be used. Add the option -coach\(^1\) to the command arguments of the soccerserver application when a coach-client is used and the internal referee module of the server must be deactivated. You can also add the line coach to the server.conf.

If you want to connect a trainer but let the server referee remain activated, add the option -coach\(_w\)referee to the command arguments of the server or add coach\(_w\)referee to the server configuration file.

If the server is invoked with one of the trainer modes, it prepares a UDP socket to which the trainer-client can connect. The default port number is 6001\(^2\). If a different port number is needed the new port can be set by assigning its value to the coach\(\_\)port parameter (see Section B.1).

7.4. Commands

The trainer and the online coach can use the following set of commands. The items are listed in three categories. The first category includes commands that can be used only by the trainer, the second includes commands that can be used also by the online coach with certain restrictions, and the third lists commands that can be used by both trainer and online coach.

7.4.1. Commands that can be used only by the trainer

- (change\_mode PLAY\_MODE)

---

\(^1\)Note: The name of this parameter refers to the notion of 'offline-coach', not to be mixed up with the online-coach.

\(^2\)The default port number for online coaches is 6002.
7.4. Commands

Change the play-mode to PLAY_MODE. PLAY_MODE must match one of the modes defined in Section 4.7.1. Note that for most play-mode requests the soccerserver will only change the play-mode. The position of the ball usually remains unchanged, but in some cases players will be moved. E.g. in free-kick and kick-in playmodes they will be moved away from the ball if they stand within a certain radius. When changing to `before kick off' they will be even moved to their own side.

Possible replies by the soccerserver:

- (ok change_mode)
  The command succeeded.
- (error illegal_mode)
  The specified mode was not valid.
- (error illegal_command_form)
  The PLAY_MODE argument was omitted.

• (move OBJECT X Y [VDIR [VEL_X VEL_Y]])
  This command will move OBJECT, which may be a player or the ball (see Section sec:sensormodels for format information), to absolute position (X, Y). If VDIR is specified, it will also change its absolute facing direction to VDIR (this only matters for players). Additionally, if VEL_X and VEL_Y are specified, the object’s velocity will be set accordingly.

  The trainer always uses left-hand coordinates.

  Possible replies by the soccerserver:

  - (ok move)
    The command succeeded.
  - (error illegal_object_form)
    The OBJECT specification was not valid.
  - (error illegal_command_form)
    The position, direction, and/or velocity specification was not valid.

• (check_ball)
  Ask the soccerserver to check the position of the ball. Four positions are defined:

  - in_field
    The ball is within the boundaries of the field.
  - goal_l
    The ball is within the area assigned to the goal at the left side of the field.
  - goal_r
    The ball is within the area assigned to the goal at the right side of the field.
7. The coach

- out_of_field
  The ball is somewhere else.

Note that the states ‘goal_l’ and ‘goal_r’ do not necessarily imply that the ball actually crossed the goal line.

Possible replies by the soccerserver:

- (ok check_ball TIME BALLPOSITION)
  BALLPOSITION will be one of the states specified above.

• (start)
  This command starts the server, e.g. sets the play-mode to ‘kick_off_l’. This essentially simulates pressing the kick off button on the monitor.

If the trainer does not send an init command, then the first commands of any type received from the trainer will cause the server to start, e.g. set the play-mode to ‘kick_off_l’.

Possible replies by the soccerserver:

- (ok start)
  The command succeeded.

• (recover)
  This command resets players’ stamina, recovery, effort and hear capacity to the values at the beginning of the game.

Possible replies by the soccerserver:

- (ok recover)
  The command succeeded.

• (ear MODE)
  It turns on or off the sending of auditory information to the trainer. MODE must be one of on and off. If (ear on) is sent, the server sends all auditory information to the trainer. See Table 7.3 for the format. If (ear off) is sent, the server stops sending auditory information to the trainer.

Possible replies by the soccerserver:

- (ok ear on)
  (ok ear off)
  Both replies indicate that the command succeeded.

- (error illegal_mode)
  MODE did not match on or off.

- (error illegal_command_form)
  The MODE argument was omitted.
7.4.2. Commands that can also be used by the online coach with certain restrictions

- \texttt{(init \ (version \ VERSION))} for the trainer and

- \texttt{(init \ TEAMNAME \ (version \ VERSION))} for the online coach.

These commands tell the server which protocol version should be used to communicate with the trainer or coach. In the case of the online coach \texttt{TEAMNAME} has to be specified to indicate which team the coach belongs to. Note that the coach must connect after at least one player from its team.

The trainer is \textit{not} required to issue an init command. However, it is recommended that the trainer does so. Otherwise, the server will communicate with an older protocol.

It should be mentioned that the default port is 6001 for the trainer and 6002 for the online coach.

Possible replies by the soccerserver:

- \texttt{(init \ ok)}
  The command succeeded in case of the trainer.

- \texttt{(init \ SIDE \ ok)}
  The command succeeded in case of the online coach. \texttt{SIDE} is either ‘l’ or ‘r’.

- \texttt{(say \ MESSAGE)}

  Note that the online coach can use this command with the same syntax, but there are more restrictions. See Section 7.6.2 for details.

  This command broadcasts the message MESSAGE to all clients in the case of the trainer and only to teammates in the case of the online coach. For the trainer the format of MESSAGE is the same as for a player-client. It must be a string whose length is less than \texttt{say\_coach\_msg\_size} (see Section B.1) and it must consist of alphanumeric characters and/or the symbols (.)+*/?<>.

  The format which the players hear these messages can be found in Section 4.3.1.

  Possible replies by the soccerserver:

  - \texttt{(ok \ say)}
    The command succeeded.

  - \texttt{(error \ illegal\_command\_form)}
    MESSAGE did not match the required format.

- \texttt{(change\_player\_type \ TEAM\_NAME \ UNUM \ PLAYER\_TYPE)} for the trainer and
7. The coach

- (change_player_type UNUM PLAYER_TYPE) for the online coach.

These commands can be used to change the heterogeneous player type (see Section 4.6) of the player with the number UNUM of team TEAM_NAME to the type PLAYER_TYPE. PLAYER_TYPE is a digit between 0 and 6, where 0 denotes the default player type. Note that in the case of the online coach the argument TEAM_NAME is missing, because it can only change player types in its own team.

The trainer does not have to comply to the rule that a maximum of three (specified by subs_max) players of each type can be on the field.

See Section 7.6.3 for details about the restrictions as to when and how the online coach may substitute players.

Possible replies by the soccerserver to both trainer and online coach:

- (warning no_team_found)
  The team does not exist.

- (error illegal_command_form)
  If change_player_type is not followed by a string, two integers and a close bracket.

- (warning no_such_player)
  If there is no player with that uniform number on that team.

- (ok change_player_type TEAM UNUM TYPE)
  The command succeeded.

Additionally, the soccerserver can send the following replies to the online coach:

- (warning cannot_sub_while_playon)
  If the play-mode is 'play_on'.

- (warning no_subs_left)
  If the coach has already made its three (specified by subs_max) subs for the game.

- (warning max_of_that_type_on_field)
  If the player-type is not the default and there are three (specified by subs_max) of that type already on the field.

- (warning cannot_change_goalie)
  If the coach tries to change the player type of the goalie.

The server responds to the teammates with:

- (change_player_type UNUM TYPE)

and opponents (including opponent coach) with:

- (change_player_type UNUM)
7.4. Commands

7.4.3. Commands that can be used by both trainer and online-coach

- (look)
  This command provides information about the positions of the following objects on the field:
  - The left and right goals.
  - The ball.
  - All active players.

Note that the trainer and online coach for both sides receive left hand coordinates. That is, the coaches receive information in the global coordinates that the left hand team uses. In general, the players receive no global information (the one exception being the move command), but it is common for teams to localize themselves so that the negative $x$ direction is towards the goal they defend.

Possible replies by the soccerserver:

- (ok look TIME (OBJ_1 OBJDESC_1) (OBJ_2 OBJDESC_2) ...)
  OBJ_j can be any of the objects mentioned above. See Section 4.3 for information about the way the names for those objects are composed. OBJDESC_j have the following form:
  * For goals: X Y
  * For the ball: X Y DELTA_X DELTA_Y
  * For players: X Y DELTA_X DELTA_Y BODYANGLE NECKANGLE [POINTING_DIRECTION]

The coordinates are always in left-hand orientation, no matter whether a trainer or online coach is used.

If the trainer/coach should receive visual information periodically, use the (eye on) command.

- (eye MODE)
  MODE must be one of on and off. If (eye on) is sent, the server starts sending (see global ...) information (see Section 7.5) every 100 ms (the interval is specified by the send_vi_step parameter automatically to the client. If (eye off) is sent, the server stops to send visual information automatically. In this case the trainer/coach has to ask actively with (look), if it needs visual information.

Possible replies by the soccerserver:

- (ok eye on)
  (ok eye off)
  Both replies indicate that the command succeeded.
7. The coach

- (error illegal_mode)
  MODE id not match on or off.
- (error illegal_command_form)
  The MODE argument was omitted.

- (team_names)
  This command makes the trainer/coach receive information about the names of both teams and which side they are playing on.
  Possible replies by the soccerserver:

  - (ok team_names [(team l TEAMNAME1) [(team r TEAMNAME2)]])
  Depending on whether the teams already connected no, one, or both team name(s) will be supplied. Recall that the first team that connects will be on the left side.

7.4.4. Commands that can be used only by the online-coach

- (team_graphic (X Y "XPM line" ... "XPM line"))
  The online coach can send teams-graphics as 256 x 64 XPM to the server. Each team_graphic-command sends a 8x8 tile. X and Y are the coordinates of this tile, so they range from 0 to 31 and 0 to 7 respectively. Each XPM line is a line from the 8x8 XPM tile. Monitors that are connected to the server will receive the following message on the message-board after each of the coach’s team_graphic-commands:

  (team_graphic_l—r (X Y "XPM line" ... "XPM line"))

  Possible replies by the soccerserver:

  - (ok team_graphic X Y)
    For each tile the server sends this string in order to signal its arrival.

7.5. Messages from the server

Apart from the replies to the commands mentioned above the server also sends some messages to the trainer and online coach. If the clients connect to the server with a version >= 7.0 (using the init-command), they will receive the following parameter messages just like player clients:

- (server_param ...) once
- (player_param ...) once
- (player_type ...) once for each player type
If the client chooses to receive visual information in each cycle by sending \texttt{(eye on)} it will receive messages in the following format every 100 ms \texttt{(send\_vi\_step)}:

\[
(\text{see\_global}((OBJ_1OBJDESC_1)(OBJ_2OBJDESC_2)\ldots))
\]

OBJ$_j$ denotes the name of the object. See Table 4.3 for information about the way the names for those objects are composed. OBJDESC$_j$ have the following form:

- For goals: X Y
- For the ball: X Y DELTA$_X$ DELTA$_Y$
- For players: X Y DELTA$_X$ DELTA$_Y$ BODYANGLE NECKANGLE \texttt{[POINT-\_DIRECTION]}

The syntax is the same as in the reply to the \texttt{(look)} command, so coordinates are always in left-hand orientation.

If the client wants to receive auditory information and sent \texttt{(ear on)} to the server, it will receive all auditory information, from both the referees and all of the players. There are two kinds of hear messages:

- \texttt{(hear TIME referee MESSAGE)} for all referee messages, such as \texttt{“play\_on”} and \texttt{“free\_kick\_left”}. See Section 4.7 for a list of the valid messages from the referee.

- \texttt{(hear TIME (p ”TEAMNAME” NUM) ”MESSAGE”)} for all player messages. Note the quotes around the message.

See Section 4.3.1 for more details about the players speaking and listening abilities.

### 7.6. Online coach

#### 7.6.1. Introduction

The online coach is a privileged client that can connect to the server in official games. It has the capability of receiving global and noise-free information about the objects on the field. In order to encourage research in this area there are special coach contests since 2001. This way, research groups that do not want to develop a team of player clients can participate in the RoboCup challenge by focusing on the online coach. Additionally, in order to make it possible to use a single coach with a variety of teams, a standard coach language (CLang) has been developed that can be used to communicate with the players.

See Section 7.4 and 7.5 for details about the commands that can be used by the online coach and messages that will be sent by the server.
7. The coach

7.6.2. Communication with the players

Prior to version 7.00, the online coach could say short (128 characters, `say_coach_msg_size`) alphanumeric (plus the symbols `!@#$%^&*()_)` messages when the play-mode is not `play_on`. This type of message still exists as a “freeform” message, but there are now other standard message types. Since version 8.05 there are also certain intervals in which freeform-messages can be sent even during `play_on`. Every 600 cycles (specified by `freeform_wait_period`) the coach can send freeform-messages for 20 cycles (specified by `freeform_send_period`). For example, if the playmode changes to `play_on` at cycle 420 and stays in `play_on` till the end of this example, the coach can send freeform-messages between 1020 and 1040, 1620 and 1640, etc. The coach can send `say_coach_cnt_max` freeform messages per game. The length of these messages has to be less than `say_coach_msg_size`. If the game continues into extended time, the online coaches are given an additional `say_coach_cnt_max` messages to say every additional 6000 cycles (or whatever the normal length of a game is). Allowed messages are cumulative, so if the coach does not use all its allowed messages, it can use them in the extended time. The server will send `(error said too many messages)` if the coach tries to send messages after it reached the maximum number.

It should be noted that freeform-messages are not allowed in coach-competition-games, and are only supported by CLang for compatibility reasons.

In the standard coach language there are three other types of messages: rule-, define-, and delete-messages. To prevent coaches from micro-controlling every single action of the players communication is restricted in the following ways.

Every 300 cycles (specified by `clang_win_size`) the coach can send one of each. Note that the number of allowed messages can be changed by setting the `clang_define_win`, `clang_del_win`, and `clang_rule_win` parameters (see Section B.1). The messages are heard by the players 50 cycles later. If the play-mode is not `play_on`, one message is sent to the players in each cycle, even if the delay time has not elapsed. Messages that are sent while the play mode is not `play_on` do not count towards the message number restrictions. For example, if the default values are used the coach can send one message per cycle during breaks that will be heard by the players without delay. The server guarantees that messages of each type will be sent to the players in the same order in which they were received from the coach.

The language grammar developed below does not place restrictions on the length of the messages which can be sent to the server. However, for very practical reasons, any message in the standard language can not be longer than 8154 characters (this is so the maximum message which should be sent to the player is 8K).

The first version of the coach language (CLang) was developed for server version 7.x. For server version 8.x the language has been extended. Because of this, clients that want to receive messages from their coach have to explicitly advise the server, which version of CLang they support. This is done by sending

```
(clang (ver MIN MAX))
```
where MIN and MAX are unsigned integers denoting the earliest and latest supported version of CLang, respectively. Clients that do not send such a message will not receive coach messages. The server is able to determine the version number of coach messages and will filter out any messages that are not supported by the player. If a message has been filtered out, the players will receive

\[(\text{hear TIME online\_coach\_left—right (unsupported\_clang)})\]

The coach will receive a message for each player which informs it about the supported versions:

\[(\text{clang (ver (PLAYER\_NAME) MIN MAX)})\]

This means that you have to add the sending of \((\text{clang (ver 7 7)})\), if you use version 7 source code of players with newer server versions.

The standard coach language will be described in detail in Section 7.7.

7.6.3. Changing player types

Using the change\_player\_type-command (described in in Section 7.4) the online coach can change player types unlimited times in `before\_kick\_off` play-mode. Of course these changes have to comply with the general rules about heterogeneous players (see Section 4.6). After kick-off player types can be changed three (subs\_max) times during play-modes that are not `play\_on`.

See the description of the change\_player\_type-command in Section 7.4 for details about the possible replies from the server.

Note: A player client will be informed about substitutions that occurred before the client connected by the message \((\text{change\_player\_type UNUM TYPE)}\) for substitutions in it own team and \((\text{change\_player\_type UNUM)}\) for substitutions in the opponent team.

7.7. The standard coach language

7.7.1. General properties

The standard coach language was developed to enable coaches to work together with teams from different research groups. One of the design goals was to have clear semantics that should prevent misinterpretation from both the players and the coach. The language is based on low-level concepts that can be combined to construct new high level concepts.

Additionally, coaches can communicate a certain number of freeform messages that may be arbitrary strings to the players during non-`play\_on`-modes. See Section 7.6.2 for details. Be aware though, that freeform messages probably will not be understood by other teams if you plan to use your coach with other teams.

The language description below is the improved and extended version of the language developed by the community, as it is supported by server version 8.x. While the first
version of CLang is still supported by the server, its use is not encouraged. A complete
description of this first version can be found in the manual for server version 7. It is
hoped that all interested researchers will continue to develop CLang in order to make it
a useful tool for multi-agent research.

Some concepts were derived from Unilang [14] (e.g. definitions and several actions)
and SFL [12] (e.g. variables and point arithmetic).

Note that the server itself parses all the coach messages using \texttt{lex} and \texttt{yacc} (the
GNU replacements for \texttt{lex} and \texttt{yacc}) and constructs a simple representation based on
a C++ class hierarchy. Please feel free to use and modify this code from the server to
handle the parsing of the coach messages. In particular, look at the \texttt{coach_lang*} files.

### 7.7.2. Example Language Utterance

The general idea of CLang is to describe tactics and behaviours as rules which map
directives to conditions. Each rule consists of a component which denotes a situation
(the \textit{condition}) and a list of \textit{directives} which are applicable if the situation-description
is true in the given worldstate. Rules can either be used as advise which tells the player
how to act or as information which for example describes how the opponent behaves in
certain situations. In CLang rules also have an ID, so that the coach can refer to them
later.

A simple rule which advises the player number 5 to pass to his teammate with the
number 11 if it has the ball and is in the middle of the field can be defined as follows:

```
(define
 (definerule
  MyRule1
direc
  (and
   (bowner our 5)
   (bpos (rec (pt -10 -10) (pt 10 10)))
  )
  (do our 5 (pass 11))
 )
)
```

Each of the primitives will be explained in detail later. For now it should suffice to
get the idea that the rule is assigned the ID "MyRule1" and is defined as a directive (as
compared to a model-rule which describes observed behavior). \texttt{bowner} determines that
player 5 of the coach’s team is the ballowner. \texttt{bpos} specifies the ballposition by means
of a rectangle. Finally, the directive advises player number 5 to pass to his teammate
11. In CLang lingo \texttt{(pass 11)} is an \textit{action} and \texttt{(do our 5 (pass 11))} is a \textit{directive}.

Rules are off by default. So the coach has to turn them off by sending a message like
\texttt{(rule (on MyRule1))}

90
Now the language concepts will be looked at in more detail.

### 7.7.3. Overview of the five message types

There are four types of coach messages in the standard coach language: Rule, Define, Delete, and Freeform. Their purpose and format will be described in this section, and some examples will be given.

In the following format description elements in capitals denote non-terminal symbols which are defined in section 7.7.7.

**Define-message**: Define messages are the most complex messages in CLang, because they define and combine the components which the coach wants to share with the players, like conditions, directives, regions, actions, and rules. By defining a component it is assigned an ID which the coach can use to refer to it in later messages.

- **Conditions**: Format for defining a condition: `(definec CLANG_STR CONDITION)`
  
  Example: `(definec "Defense" (bowner opp 0))` This defines the condition in which any player of the opponent team owns the ball.

- **Actions**: Format for defining an action: `(definea CLANG_STR ACTION)`
  
  Example: `(definea "Pass7" (pass 7))`

- **Directives**: Format for defining a directive: `(defined CLANG_STR DIRECTIVE)`
  
  Example: `(defined "Pass10to11" (do our 10 (pass 11)))` This directives denotes player 10 passing to player 11.

- **Regions**: Format for defining a region: `(definer CLANG_STR REGION)`
  
  Example: `(definer "OURHALF" (rec (pt -52.5 -34) (pt 0 34)))` A rectangle which covers the team’s own half is defined.

- **Rules**: Format for defining a rule: `(definerule CLANG_VAR model RULE)` or `(definerule CLANG_VAR direc RULE)`
  
  Example: `(definerule Rule1 direc ((playm bko) (do our 7 (pos (pt -20 20)))))` This rule states that player 7 should position itself at the given point before kick-off.

  See also section 7.7.4 about defining rules.

**Rule-message**: Rule messages are used to turn previously defined rules on or off. After defining a rule, it is off by default.
7. The coach

Format: (rule ACTIVATION_LIST)
Example: (rule (on rule2) (off rule1))

Delete-message: The delete message tells a player that a rule will not be used again and can be removed from the memory. This also means that after deleting a rule, its ID should not appear in other nested rule-definitions (see section 7.7.4) anymore.
Format: (delete ID_LIST)
Examples: (delete Rule1) (delete (Rule1 Rule2)) (delete all) Deletes one rule, a list of two rules, or all rules, respectively.

Freeform-message: Free form messages are arbitrary strings and can be sent according to the afore-mentioned restrictions in section 7.6.2.
Format: (freeform ”STRING”)
Note that STRING must be included in quotes.

7.7.4. Defining rules

The definition of rules is an important part in CLang, so it will be looked at in more detail in this section. Remember that a rule consists of a condition and a list of directives, which again contain actions.

As stated above the format for defining a rule is (definerule DEFINE_RULE) using the following components:

<DEFINE_RULE> : <CLANG_VAR> model <RULE>  
| <CLANG_VAR> direc <RULE>

<RULE> : (<CONDITION> <DIRECTIVE_LIST>)  
| (<CONDITION> <RULE_LIST>)  
| <ID_LIST>

Each rule is assigned a name complying the definition of CLANG_VAR. Additionally, rules are in one of two modes, either model which states that the rule is a description of observed behavior, or direc which states that the rule is a directive to behave in a certain way.

Now, the actual content of a rule can be specified in several ways:

• (CONDITION DIRECTIVE_LIST)

This is the straight-forward way. The example in section 7.7.3 complies to this format. The CONDITION denotes a situation, and DIRECTIVE_LIST denotes the appropriate directives. Note that the list can contain directives for one, several, or all players, or even several directives for the same player. In the latter case it is up to the player to decide which directive is to be followed.
7.7. The standard coach language

- **(CONDITION RULE_LIST)**

  This is a very powerful format for combining rules to larger tactics. Since each rule in RULE_LIST already contains a condition, a definition of this form results in nested rules. It can for example be used to activate several rules simultaneously. Suppose, there are already several rules specifying the home positions of the defenders: pos2a and pos2b for player 2, and pos3a and pos3b for player 3. Now, by using

  (definerule defenseformation direc ((bowner our \{0\}) (pos2a pos3a)))

  and

  (definerule offenseformation direc ((bowner opp \{0\}) (pos2b pos3b)))

  it can be specified when the rules are supposed to be active (depending on which team owns the ball). For evaluating such definitions, the outer condition is assumed to be distributed into the inner conditions, being combined with logical **and**. E.g. assume that pos2a was specified as

  \(((time > 20) (do our \{2\} (pos (pt -40 10))))\)

  then the above definition would create

  \(((and (bowner our \{0\}) (time > 20)) (do our \{2\} (pos (pt -40 10))))\)

- **ID_LIST**

  Similar to the above format, this way several existing rules can be combined. Suppose, there have been defined two rules:

  (definerule position2 direc ((true) (home (pt -40 -10))))

  (definerule mark2 direc ((bowner opp 10) (mark 10)))

  These can be combined into a behavior for player 2:

  (definerule player2 direc (position2 mark2))

7.7.5. Semantics and syntax details of the components

In the following the syntax and semantics of the non-terminal symbols which were used in the format outlines above will be described.

Rules have a condition on the left-hand side, and a set of actions on the right hand side. Thus each rule can be thought of as essentially specifying an if-then statement:

```
if CONDITION
  then { DIRECTIVE_1 DIRECTIVE_2 ... }
```

In the player’s programs, it is easy to represent all the advice given by the coach as a small rule-base. Following the advice would be easy by matching the current world state against the condition, and trying to act on the directives. Note: If more than one condition applies to the current situation and the corresponding directives differ, it is up to the player to choose the directive. Note that the player should also exercise some
discretion in following directives. For example, if the only directive which matches is to pass to player 5, but player 5 is well-covered by opponents, the player with the ball may choose to ignore the directive for now.

- Conditions:
  A condition is made from the logical connectives over atomic state description propositions:
  
  - (true)
    Always true.
  
  - (false)
    Always false.
  
  - (ppos TEAM UNUM_SET INT INT REGION)
    The first INT is the MINIMUM and the second is the MAXIMUM. At least MINIMUM but no more than MAXIMUM players in UNUM_SET from team TEAM are in region REGION. Regions and unum sets are more precisely defined below. TEAM is either "our" or "opp". There is no ambiguity since the coach can only be heard by its own players.
  
  - (bpos REGION)
    The ball is in region REGION.
  
  - (bowner TEAM UNUM_SET)
    The ball is controlled by some player in UNUM_SET of team TEAM. The ball-owner is the last player that had ball contact (i.e. the ball was in his kickable area), even if the ball left his control after that.
  
  - (playm PLAY_MODE)
    The play-mode is PLAY_MODE. See Section 7.7.7 for the valid values of PLAY_MODE.
  
  - (COND_COMP)
    The time, goal-difference, number of own or opponent goals can be compared with constants, using the operators $<$ $>$ $<=$ $== != >=$.
    Examples: (time $>$ 20) (2 $>$= opp.goals)
  
  - (unum CLANG_VAR UNUM_SET)
    If CLANG_VAR is instantiated, it is checked whether the unum denoted by the variable CLANG_VAR is in the set UNUM_SET. If the variable is still unbound, it is bound to the specific set.

The logical connectives are:

  - (and CONDITION$_1$ CONDITION$_2$ \ldots CONDITION$_n$)
  
  - (or CONDITION$_1$ CONDITION$_2$ \ldots CONDITION$_n$)
  
  - (not CONDITION)
An example condition: "When opponent player 3 is in region X and controls the ball" would be

\[(\text{and (ppos opp \{3\} X) (bowner opp \{3\})}\)]

* Directives:

Directives are basically lists of actions for individual sets of players and come in two forms:

- \((\text{do TEAM UNUM\_SET ACTION\_LIST})\) (affirmative mode: players should take these actions)
- \((\text{dont TEAM UNUM\_SET ACTION\_LIST})\) (negative mode: players should avoid taking these actions)

If the actions in the affirmative mode are mutually exclusive, it is up to the player to decide which one is to be followed. In rules which are in the `model`-mode, directives convey knowledge about the plans/behaviors of the players or their opponents.

* Actions:

- \((\text{pos REGION})\)
  The player should position itself in REGION.
- \((\text{home REGION})\)
  The player’s default position should be in REGION. This directive is intended largely to specify formations for the team.
- \((\text{mark UNUM\_SET})\)
  The player should mark some opponent player in UNUM\_SET.
- \((\text{markl REGION})\)
  The passing lane from the current ball position to REGION should be marked.
- \((\text{markl UNUM\_SET})\)
  The passing lane from the current ball position to some opponent player in UNUM\_SET should be marked.
- \((\text{oline REGION})\)
  The offside-trap line for the player/team should be set at REGION.
- \((\text{htype TYPE})\)
  The player is of heterogeneous type TYPE. The TYPE number is as described in Section 4.6. A value of -1 should clear the player’s idea of the heterogeneous type.
- \((\text{pass REGION})\)
  The ball should be passed to some player in REGION.
- \((\text{pass UNUM\_SET})\)
  The ball should be passed to some player in UNUM\_SET.
7. **The coach**

- (dribble REGION)
  The ball should be dribbled to REGION.

- (clear REGION)
  The ball should be cleared from REGION, which means to shoot the ball to a point outside of REGION.

- (shoot)
  The ball should be shot at the goal.

- (hold)
  The player should hold the ball, i.e. stand at his position and keeping the ball away from opponents.

- (intercept)
  The player should go to the ball and try to control it.

- (tackle UNUM_SET)
  The player should tackle some player in UNUM_SET (or the ballowner?).

**Regions:**

Any REGION token can be any of the following:

- a POINT
  This is defined more precisely below

- (rec POINT₁ POINT₂)
  Defines a rectangle with its sides parallel to the pitch-lines, respectively.

- (tri POINT₁ POINT₂ POINT₃)
  Defines a triangle made up of the given points.

- (arc POINT RADIUS_SMALL_RADIUS_LARGE_RADIUS_ANGLE_BEGIN ANGLE_SPAN)
  Defines a donut-arc: the area between two circles co-centered at point POINT, having the given radii, with the arc defined starting at the beginning angle and covering the spannign angle. For example a, a circle with radius \(r\) could be defined as “(arc (pt 0 0) 0 \(r\) 0 360)”, and a U-shaped region could be defined as “(arc (pt 0 0) 5 10 0 180)”

- (null)
  The null (empty) region.

- (reg REG₁ REG₂ \ldots REGₙ)
  Defines a region made up from the union of the given regions.

A POINT is any of the following:

- (pt X Y)
  X and Y are reals and in global coordinates. This is the absolute position (X,Y):
7.7. The standard coach language

- (pt ball) The current global position of the ball.
- (pt TEAM UNUM) The current position of player number UNUM on team TEAM (either 'our' or 'opp'). Remember that UNUM can be a variable.
- (POINT_1 OP POINT_2)
  This arithmetically combines two points to a new point. POINT_1 can be made up of arithmetic operators, resulting in a recursive structure.
  The operators are defined in the natural way, for example:
  \[(pt \ X_1 \ Y_1) \ OP \ (pt \ X_2 \ Y_2) = (pt \ X_1 \ OP \ X_2 \ Y_1 \ OP \ Y_2)\]
  where OP is one of + − * /

The use of these relative points makes it easy to express ideas such as “Move to the ball”, “If there are 2 teammates within 10m of the ball”, etc.

Remember that the online coach receives visual information always in left-hand orientation, no matter which side its team plays on. Yet, when sending messages to a team that plays on the right side, the coach must use right-hand orientation in the messages. Transforming coordinates from left- to right-hand orientation is done by negating them.

• UNUM SETS:

  Unum sets are sets of player numbers. These are sets in the sense that order does not matter and may be changed by the server. If 0 is included anywhere in the set, then the set contains all players 1 - 11. The set can contain variables.

  Format: \{ NUM_1 \ NUM_2 \ldots \ NUM_n \}

• Variables:

  Technically, everywhere where UNUM occurs, a variable can be used. Yet, it is important to make sure that the variables are instantiated or ground. The scope is the innermost spanning rule, e.g. in

  1 (definerule rule1 model
  2   (bowner our \{0\})
  3   ((true) (do our \{5\} (mark 11)))
  4   ((bowner our \{X\}) (do our \{X\} (shoot)))
  5   )

  the scope of X is the complete line 4. This also shows how variables can be instantiated: Only in conditions which have UNUMs as fixed argument (i.e. UNUMs in POINTs do not count as condition UNUMS) a variable may be introduced. Its value is set by checking which unums make the condition true. In the example X is instantiated with the uniform number of the ballowner. In a condition like ppos it can be necessary to instantiate the variable as a set of unums:

97
7. The coach

\[(\text{ppos our } \{X\} \ 1 \ 1 \ \text{REGION})\]

In this example \(X\) has to be instantiated as the set of unums which are in REGION. Note that an instantiation as in
\[(\text{ppos our } \{5\} \ 1 \ 1 \ (\text{rec } (\text{pt } \text{ball}) \ (\text{pt } \text{our } \{X\}))\) is not supported.

### 7.7.6. Further resources

- The CLang Corpus contains examples of actual CLang messages:
  http://www-2.cs.cmu.edu/~pfr/soccer/clang_corpus.html
- The Multi-Agent Modeling Special Interest Group (MAMSIG) provides binaries and sources of coachable teams and online coaches:
  http://www.cl-ki.uni-osnabrueck.de/~tsteffen/mamsig
- The Coach-mailing-list discusses Clang details, competition rules, and coaching methods:
  http://robocup.biglist.com/coach-l/

### 7.7.7. Syntax

The complete grammar of the standard coach language:

\[
\text{<MESSAGE>} : \text{<FREEFORM_MESS>} | \text{<DEFINE_MESS>} | \text{<RULE_MESS>} | \text{<DEL_MESS>}
\]

\[
\text{<RULE_MESS>} : (\text{rule } \text{<ACTIVATION_LIST>})
\]

\[
\text{<DEL_MESS>} : (\text{delete } \text{<ID_LIST>})
\]

\[
\text{<DEFINE_MESS>} : (\text{define } \text{<DEFINE_TOKEN_LIST>})
\]

\[
\text{<FREEFORM_MESS>} : (\text{freeform } \text{<CLANG_STR>})
\]

\[
\text{<DEFINE_TOKEN_LIST>} : \text{<DEFINE_TOKEN_LIST>} \text{<DEFINE_TOKEN>} | \text{<DEFINE_TOKEN>}
\]

\[
\text{<DEFINE_TOKEN>} : (\text{definec } \text{<CLANG_STR>} \text{<CONDITION>})
\]

\[
\text{<DEFINE_TOKEN>} : (\text{defined } \text{<CLANG_STR>} \text{<DIRECTIVE>})
\]

\[
\text{<DEFINE_TOKEN>} : (\text{definer } \text{<CLANG_STR>} \text{<REGION>})
\]

\[
\text{<DEFINE_TOKEN>} : (\text{definea } \text{<CLANG_STR>} \text{<ACTION>})
\]

\[
\text{<DEFINE_TOKEN>} : (\text{definerule } \text{<DEFINE_RULE>})
\]

\[
\text{<DEFINE_RULE>} : \text{<CLANG_VAR>} \text{model } \text{<RULE>}
\]

\[
\text{<DEFINE_RULE>} : \text{<CLANG_VAR>} \text{direc } \text{<RULE>}
\]
7.7. The standard coach language

<RULE> : (<CONDITION> <DIRECTIVE_LIST>)
  | (<CONDITION> <RULE_LIST>)
  | <ID_LIST>

<ACTIVATION_LIST> : <ACTIVATION_LIST> <ACTIVATION_ELEMENT>
  | <ACTIVATION_ELEMENT>

<ACTIVATION_ELEMENT> : (on|off <ID_LIST>)

<ACTION> : (pos <REGION>)
  | (home <REGION>)
  | (mark <UNUM_SET>)
  | (markl <UNUM_SET>)
  | (markl <REGION>)
  | (oline <REGION>)
  | (htype <INTEGER>)
  | <CLANG_STR>
  | (pass <REGION>)
  | (pass <UNUM_SET>)
  | (dribble <REGION>)
  | (clear <REGION>)
  | (shoot)
  | (hold)
  | (intercept)
  | (tackle <UNUM_SET>)

<CONDITION> : (true)
  | (false)
  | (ppos <TEAM> <UNUM_SET> <INTEGER> <INTEGER> <REGION>)
  | (bpos <REGION>)
  | (bowner <TEAM> <UNUM_SET>)
  | (playm <PLAY_MODE>)
  | (and <CONDITION_LIST>)
  | (or <CONDITION_LIST>)
  | (not <CONDITION>)
  | <CLANG_STR>
  | (<COND_COMP>)
  | (unum <CLANG_VAR> <UNUM_SET>)
  | (unum <CLANG_STR> <UNUM_SET>)

<COND_COMP> : <TIME_COMP>
7. The coach

| <OPP_GOAL_COMP> |
| <OUR_GOAL_COMP> |
| <GOAL_DIFF_COMP> |

<TIME_COMP> : time <COMP> <INTEGER> 
  | <INTEGER> <COMP> time

<OPP_GOAL_COMP> : opp_goals <COMP> <INTEGER> 
  | <INTEGER> <COMP> opp_goals 

<OUR_GOAL_COMP> : our_goals <COMP> <INTEGER> 
  | <INTEGER> <COMP> our_goals 

<GOAL_DIFF_COMP> : goal_diff <COMP> <INTEGER> 
  | <INTEGER> <COMP> goal_diff 

<COMP> : < | <= | == | != | >= | >

<PLAY_MODE> : bko | time_over | play_on | ko_our | ko_opp 
  | ki_our | ki_opp | fk_our | fk_opp 
  | ck_our | ck_opp | gk_opp | gk_our 
  | gc_our | gc_opp | ag_opp | ag_our 

<DIRECTIVE> : (do|dont <TEAM> <UNUM_SET> <ACTION_LIST>) 
  | <CLANG_STR> 

<REGION> : (null) 
  | (arc <POINT> <REAL> <REAL> <REAL> <REAL>) 
  | (reg <REGION_LIST>) 
  | <CLANG_STR> 
  | <POINT> 
  | (tri <POINT> <POINT> <POINT>) 
  | (rec <POINT> <POINT>) 

<POINT> : (pt <REAL> <REAL>) 
  | (pt ball) 
  | (pt <TEAM> <INTEGER>) 
  | (pt <TEAM> <CLANG_VAR>) 
  | (pt <TEAM> <CLANG_STR>) 
  | (<POINT_ARITH>) 

<POINT_ARITH> : <POINT_ARITH> <OP> <POINT_ARITH> 
  | <POINT> 

100
7.7. The standard coach language

<OP> : + | - | * | /

<REGION> : <REGION_LIST> <REGION>
| <REGION>

<UNUM_SET> : { <UNUM_LIST> }

<UNUM_LIST> : <UNUM>
| <UNUM_LIST> <UNUM>

<UNUM> : <INTEGER> | <CLANG_VAR> | <CLANG_STR>

<ACTION_LIST> : <ACTION_LIST> <ACTION>
| <ACTION>

<DIRECTIVE_LIST> : <DIRECTIVE_LIST> <DIRECTIVE>
| <DIRECTIVE>

<CONDITION_LIST> : <CONDITION_LIST> <CONDITION>
| <CONDITION>

<RULE_LIST> : <RULE_LIST> <RULE>
| <RULE>

<ID-LIST> : <CLANG_VAR>
| (<ID_LIST2>)
| all

<ID_LIST2> : <ID_LIST2> <CLANG_VAR>
| <CLANG_VAR>

<CLANG_STR> : "[0-9A-Za-z\(\)\./\*\-\?\<\>\_ ]+

<CLANG_VAR> : [abe-oqrt-zA-Z_]^[a-zA-Z0-9_]*
7. *The coach*

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Used value</th>
<th>Default value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>coach_port</td>
<td>6001</td>
<td>6001</td>
<td>The port number the trainer connects to.</td>
</tr>
<tr>
<td>say_msg_size</td>
<td>512</td>
<td>256</td>
<td>Maximum length of a freeform message a player, trainer, or coach can say.</td>
</tr>
<tr>
<td>say_coach_sent_max</td>
<td>128</td>
<td>128</td>
<td>Upper limit of freeform messages an online coach can say</td>
</tr>
<tr>
<td>send_vi_step</td>
<td>100</td>
<td>100</td>
<td>Interval of online coach’s look.</td>
</tr>
<tr>
<td>clang_win_size</td>
<td>100</td>
<td>100</td>
<td>Number of cycles that lie between online coach messages</td>
</tr>
<tr>
<td>clang_define_win</td>
<td>1</td>
<td>1</td>
<td>Number of define messages that can be sent in the aforementioned interval.</td>
</tr>
<tr>
<td>clang_rule_win</td>
<td>1</td>
<td>1</td>
<td>Number of rule messages that can be sent in the aforementioned interval.</td>
</tr>
<tr>
<td>clang_del_win</td>
<td>1</td>
<td>1</td>
<td>Number of delete messages that can be sent in the aforementioned interval.</td>
</tr>
<tr>
<td>clang_mess_delay</td>
<td>50</td>
<td>50</td>
<td>Number of cycles messages from the online coach will be delayed.</td>
</tr>
<tr>
<td>clang_mess_per_cycle</td>
<td>1</td>
<td>1</td>
<td>Number of messages that will be sent to the players during non-play modes.</td>
</tr>
</tbody>
</table>
### 7.7. The standard coach language

**From trainer to server**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init (version VERSION))</code></td>
<td>( \text{VERSION ::= a real number} )</td>
</tr>
<tr>
<td><code>change_mode PLAY_MODE)</code></td>
<td>( \text{PLAY_MODE ::= one of the play-modes} )</td>
</tr>
</tbody>
</table>
| `move OBJECT X Y [VDIR [DELTA_X DELTA_Y]]` | \( \text{OBJECT ::= One of object names} \)
\( \text{X ::= -52{-52}} \)
\( \text{Y ::= -32{-32}} \)
\( \text{VDIR ::= -180{-180}} \)
\( \text{DELTA_X, DELTA_Y ::= [float]} \) |
| `check_ball)` | \( \text{(ok check\_ball TIME BPOS)} \)
\( \text{TIME ::= sim. time of server} \)
\( \text{BPOS ::= in\text{\_field} | goal\_SIDE | out\text{\_of\_field}} \)
\( \text{SIDE ::= l | r} \) |
| `start)` | \( \text{(ok start)} \) |
| `recover)` | \( \text{(ok recover)} \) |
| `change_player_type TEAM_NAME UNUM PLAYER_TYPE)` | \( \text{(warning no\_team\_found)} \)
\( \text{(error illegal\_command\_form)} \)
\( \text{(warning no\_such\_player)} \)
\( \text{(ok change\_player\_type TEAM UNUM TYPE)} \) |
| `ear MODE)` | \( \text{(ok ear on)} \)
\( \text{(ok ear off)} \)
\( \text{(error illegal\_mode)} \)
\( \text{(error illegal\_command\_form)} \) |

**From server to trainer**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>trainer: (init ok)</code></td>
<td>( \text{} )</td>
</tr>
</tbody>
</table>
| `ok change\_mode)` | \( \text{} \)
\( \text{(error illegal\_mode)} \)
\( \text{(error illegal\_command\_form)} \) |
| `ok move)` | \( \text{(error illegal\_object\_form)} \)
\( \text{(error illegal\_command\_form)} \) |
| `ok check\_ball TIME BPOS)` | \( \text{} \)
\( \text{} \)
\( \text{} \)
\( \text{} \) |
| `ok start)` | \( \text{} \) |
| `ok recover)` | \( \text{} \) |

| **Table 7.1.: Trainer Interactions with the Server** |
7. The coach

<table>
<thead>
<tr>
<th>From online coach to server</th>
<th>From server to online coach</th>
</tr>
</thead>
<tbody>
<tr>
<td>(init TEAMNAME</td>
<td>(init SIDE ok)</td>
</tr>
<tr>
<td>(version VERSION))</td>
<td>SIDE ::= l</td>
</tr>
<tr>
<td>VERSION ::= a real number</td>
<td></td>
</tr>
<tr>
<td>TEAMNAME ::= string</td>
<td></td>
</tr>
</tbody>
</table>

| (change_player_type         | (warning no_team_found)    |
|   UNUM PLAYER_TYPE)        |                             |
| UNUM ::= 1–11              | (error illegal_command_form) |
| PLAYER_TYPE ::= 0–6        | (warning no_such_player)   |
|                           | (ok change_player_type     |
|                           |   TEAM UNUM TYPE)          |
|                           | (warning cannot_sub_while_playon) |
|                           | (warning no_subs_left)     |
|                           | (warning max_of_that_type_on_field) |
|                           | (warning cannot_change_goalie) |

Table 7.2.: Online Coach Interactions with the Server
7.7. The standard coach language

<table>
<thead>
<tr>
<th>From client to server</th>
<th>From server to client</th>
</tr>
</thead>
<tbody>
<tr>
<td>(say MESSAGE)</td>
<td>(ok say)</td>
</tr>
<tr>
<td>(see Section 7.4.2)</td>
<td>(error illegal_command_form)</td>
</tr>
<tr>
<td>(look)</td>
<td>(ok look TIME</td>
</tr>
<tr>
<td></td>
<td>(OBJ₁ OBJDESC₁)</td>
</tr>
<tr>
<td></td>
<td>(OBJ₂ OBJDESC₂)</td>
</tr>
<tr>
<td></td>
<td>OBJ_j ::= object name</td>
</tr>
<tr>
<td></td>
<td>(see Section 4.3</td>
</tr>
<tr>
<td></td>
<td>OBJDESC_j ::= X Y</td>
</tr>
<tr>
<td></td>
<td>X Y DELTAₓ DELTAᵧ</td>
</tr>
<tr>
<td></td>
<td>X Y DELTAₓ DELTAᵧ</td>
</tr>
<tr>
<td></td>
<td>BODYANGLE NECKANGLE</td>
</tr>
<tr>
<td>(eye MODE)</td>
<td>(ok eye on)</td>
</tr>
<tr>
<td>MODE ::= on</td>
<td>off</td>
</tr>
<tr>
<td></td>
<td>(error illegal_mode)</td>
</tr>
<tr>
<td></td>
<td>(error illegal_command_form)</td>
</tr>
<tr>
<td>This message is sent automatically every send_vi_step milliseconds when the coach/trainer eye is on (see the “eye” commands below).</td>
<td>(see global TIME</td>
</tr>
<tr>
<td></td>
<td>(OBJ₁ OBJDESC₁)</td>
</tr>
<tr>
<td></td>
<td>(OBJ₂ OBJDESC₂)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>The trainer must use the ‘ear’ command to get these messages. The online coach always gets these messages.</td>
<td>(hear TIME referee MESSAGE)</td>
</tr>
<tr>
<td>(team_names)</td>
<td>(hear TIME</td>
</tr>
<tr>
<td></td>
<td>(p ”TEAMNAME” NUM)</td>
</tr>
<tr>
<td></td>
<td>”MESSAGE”)</td>
</tr>
<tr>
<td></td>
<td>TIME ::= time message was sent</td>
</tr>
<tr>
<td></td>
<td>TEAMNAME ::= string</td>
</tr>
<tr>
<td></td>
<td>NUM ::= 1–11</td>
</tr>
<tr>
<td></td>
<td>MESSAGE ::= string</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(ok team_names</td>
</tr>
<tr>
<td></td>
<td>[(team l TEAMNAME₁)</td>
</tr>
<tr>
<td></td>
<td>[(team r TEAMNAME₂)])</td>
</tr>
</tbody>
</table>

Table 7.3.: Server Interactions with Trainer/Coach
7. The coach
8. References and Further Reading

8.1. General papers


8. References and Further Reading


8.2. Doctoral Theses


8.3. Undergraduate and Master’s Theses

8.4. Platforms to start building team upon


8.4. Platforms to start building team upon

8.5. Education-related articles

8.6. Machine Learning


8.7. Decision Making


8.8. Other supporting documents


8. References and Further Reading

8.9. Team Descriptions

8.9.1. 1996
8.9.2. 1997
8.9.3. 1998


8.9.4. 1999


8.9.5. 2000
8.9.6. 2001
A. GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document,
but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other written document
“free” in the sense of freedom: to assure everyone the effective freedom to copy and
redistribute it, with or without modifying it, either commercially or noncommercially.
Secondarily, this License preserves for the author and publisher a way to get credit for
their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the doc-
ument must themselves be free in the same sense. It complements the GNU General
Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, be-
cause free software needs free documentation: a free program should come with manuals
providing the same freedoms that the software does. But this License is not limited to
software manuals; it can be used for any textual work, regardless of subject matter or
whether it is published as a printed book. We recommend this License principally for
works whose purpose is instruction or reference.

A.1. Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by
the copyright holder saying it can be distributed under the terms of this License. The
“Document”, below, refers to any such manual or work. Any member of the public is a
licensee, and is addressed as “you”.

A “Modified Version” of the Document means any work containing the Document
or a portion of it, either copied verbatim, or with modifications and/or translated into
another language.

A “Secondary Section” is a named appendix or a front-matter section of the Doc-
ument that deals exclusively with the relationship of the publishers or authors of the
A. GNU Free Documentation License

Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, \LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A.2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section
3. You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.3. Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
A. GNU Free Documentation License

- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled “History”, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.
- Do not retile any existing section as “Endorsements” or to conflict in title with any Invariant Section.
A.5. Combining Documents

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”
A. GNU Free Documentation License

A.6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.7. Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this
A.10. Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ”or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no Invariant Sections, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST”; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.
A. GNU Free Documentation License
## B. Soccerserver

### B.1. Soccerserver Parameters

Table B.1.: Parameters adjustable in `server.conf`

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Current Value in <code>server.conf</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>goal_width</code></td>
<td>7.32</td>
<td>14.02</td>
<td>goal width</td>
</tr>
<tr>
<td><code>player_size</code></td>
<td>0.3</td>
<td></td>
<td>player size</td>
</tr>
<tr>
<td><code>player_decay</code></td>
<td>0.4</td>
<td></td>
<td>player decay</td>
</tr>
<tr>
<td><code>player_rand</code></td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>player_weight</code></td>
<td>60.0</td>
<td></td>
<td>player weight</td>
</tr>
<tr>
<td><code>player_speed_max</code></td>
<td>1.0</td>
<td></td>
<td>max. player velocity</td>
</tr>
<tr>
<td><code>player_accel_max</code></td>
<td>1.0</td>
<td></td>
<td>max. player acceleration</td>
</tr>
<tr>
<td><code>stamina_max</code></td>
<td>4000.0</td>
<td></td>
<td>max. player stamina</td>
</tr>
<tr>
<td><code>stamina_max</code></td>
<td>45.0</td>
<td></td>
<td>max. player stamina increment</td>
</tr>
<tr>
<td><code>recover_dec_thr</code></td>
<td>0.3</td>
<td></td>
<td>player recovery decrement threshold</td>
</tr>
<tr>
<td><code>recover_min</code></td>
<td>0.5</td>
<td></td>
<td>min. player recovery</td>
</tr>
<tr>
<td><code>recover_dec</code></td>
<td>0.002</td>
<td></td>
<td>player recovery decrement</td>
</tr>
<tr>
<td><code>effort_dec_thr</code></td>
<td>0.3</td>
<td></td>
<td>player dash effort decrement threshold</td>
</tr>
<tr>
<td><code>effort_min</code></td>
<td>0.6</td>
<td></td>
<td>min. player dash effort</td>
</tr>
<tr>
<td><code>effort_dec</code></td>
<td>0.005</td>
<td></td>
<td>dash effort decrement</td>
</tr>
<tr>
<td><code>effort_inc_thr</code></td>
<td>0.6</td>
<td></td>
<td>dash effort increment threshold</td>
</tr>
<tr>
<td><code>effort_inc</code></td>
<td>0.01</td>
<td></td>
<td>dash effort increment</td>
</tr>
<tr>
<td><code>kick_rand</code></td>
<td>0.0</td>
<td></td>
<td>noise added directly to kicks</td>
</tr>
<tr>
<td><code>team_actuator_noise</code></td>
<td></td>
<td></td>
<td>flag whether to use team specific actuator noise</td>
</tr>
<tr>
<td><code>prand_factor_l</code></td>
<td></td>
<td></td>
<td>factor to multiply prand for left team</td>
</tr>
<tr>
<td><code>prand_factor_r</code></td>
<td></td>
<td></td>
<td>factor to multiply prand for right team</td>
</tr>
<tr>
<td><code>kick_rand_factor_l</code></td>
<td></td>
<td></td>
<td>kick_rand for left team</td>
</tr>
<tr>
<td><code>kick_rand_factor_r</code></td>
<td></td>
<td></td>
<td>kick_rand for right team</td>
</tr>
<tr>
<td><code>ball_size</code></td>
<td>0.085</td>
<td></td>
<td>ball size</td>
</tr>
<tr>
<td><code>ball_decay</code></td>
<td>0.04</td>
<td></td>
<td>ball decay</td>
</tr>
<tr>
<td><code>ball_rand</code></td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ball_weight</code></td>
<td>0.2</td>
<td></td>
<td>weight of the ball</td>
</tr>
<tr>
<td><code>ball_speed_max</code></td>
<td>2.7</td>
<td></td>
<td>max. ball velocity</td>
</tr>
<tr>
<td><code>ball_accel_max</code></td>
<td>2.7</td>
<td></td>
<td>max. ball acceleration</td>
</tr>
<tr>
<td><code>dash_power_rate</code></td>
<td>0.006</td>
<td></td>
<td>dash power rate</td>
</tr>
</tbody>
</table>
## B. Soccerserver

Table B.1.: (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Current Value in <code>server.conf</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kick_power_rate</td>
<td>0.027</td>
<td></td>
<td>kick power rate</td>
</tr>
<tr>
<td>kickable_margin</td>
<td>0.7</td>
<td></td>
<td>kickable margin</td>
</tr>
<tr>
<td>control_radius</td>
<td></td>
<td></td>
<td>control radius</td>
</tr>
<tr>
<td>catch_probability</td>
<td>1.0</td>
<td></td>
<td>goalie catch probability</td>
</tr>
<tr>
<td>catchable_area</td>
<td>2.0</td>
<td></td>
<td>goalie catchable area length</td>
</tr>
<tr>
<td>catchable_area_w</td>
<td>1.0</td>
<td></td>
<td>goalie catchable area width</td>
</tr>
<tr>
<td>goalie_max_moves</td>
<td>2</td>
<td></td>
<td>goalie max. moves after a catch</td>
</tr>
<tr>
<td>maxpower</td>
<td>100</td>
<td></td>
<td>max power</td>
</tr>
<tr>
<td>minpower</td>
<td>-100</td>
<td></td>
<td>min power</td>
</tr>
<tr>
<td>maxmoment</td>
<td>180</td>
<td></td>
<td>max. moment</td>
</tr>
<tr>
<td>minmoment</td>
<td>-180</td>
<td></td>
<td>min. moment</td>
</tr>
<tr>
<td>maxneckmoment</td>
<td>180</td>
<td></td>
<td>max. neck moment</td>
</tr>
<tr>
<td>minneckmoment</td>
<td>-180</td>
<td></td>
<td>min. neck moment</td>
</tr>
<tr>
<td>maxneckang</td>
<td>90</td>
<td></td>
<td>max. neck angle</td>
</tr>
<tr>
<td>minneckang</td>
<td>-90</td>
<td></td>
<td>min. neck angle</td>
</tr>
<tr>
<td>visible_angle</td>
<td>90.0</td>
<td></td>
<td>visible angle</td>
</tr>
<tr>
<td>visible_distance</td>
<td></td>
<td></td>
<td>visible distance</td>
</tr>
<tr>
<td>audio_cut_dist</td>
<td>50.0</td>
<td></td>
<td>audio cut off distance</td>
</tr>
<tr>
<td>quantize_step</td>
<td>0.1</td>
<td></td>
<td>quantize step of distance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>for movable objects</td>
</tr>
<tr>
<td>quantize_step_l</td>
<td>0.01</td>
<td></td>
<td>quantize step of distance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>for landmarks</td>
</tr>
<tr>
<td>quantize_step_dir</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quantize_step_dist_l</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quantize_step_team_l</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quantize_step_dist_r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quantize_step_team_r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quantize_step_team_l</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quantize_step_team_r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>corner_kick_margin</td>
<td>1.0</td>
<td></td>
<td>corner kick margin</td>
</tr>
<tr>
<td>wind_dir</td>
<td>0.0</td>
<td>0.0</td>
<td>wind direction</td>
</tr>
<tr>
<td>wind_force</td>
<td>10.0</td>
<td>0.0</td>
<td>wind direction</td>
</tr>
<tr>
<td>wind_rand</td>
<td>0.3</td>
<td>0.0</td>
<td>wind direction</td>
</tr>
<tr>
<td>wind_none</td>
<td></td>
<td></td>
<td>wind factor is none</td>
</tr>
<tr>
<td>wind_random</td>
<td>false</td>
<td></td>
<td>wind factor is random</td>
</tr>
<tr>
<td>inertia_moment</td>
<td>5.0</td>
<td></td>
<td>inertia moment for turn</td>
</tr>
<tr>
<td>half_time</td>
<td>300</td>
<td></td>
<td>length of a half time in seconds</td>
</tr>
<tr>
<td>drop_ball_time</td>
<td>200</td>
<td></td>
<td>number of cycles to wait until dropping the ball automatically</td>
</tr>
<tr>
<td>port</td>
<td>6000</td>
<td></td>
<td>player port number</td>
</tr>
<tr>
<td>coach_port</td>
<td>6001</td>
<td></td>
<td>(offline) coach port</td>
</tr>
<tr>
<td>online_coach_port</td>
<td></td>
<td></td>
<td>online coach port</td>
</tr>
<tr>
<td>say_coach_cnt_max</td>
<td>128</td>
<td></td>
<td>upper limit of the number of online coach’s message</td>
</tr>
<tr>
<td>say_coach_msg_size</td>
<td>128</td>
<td></td>
<td>upper limit of length of online coach’s message</td>
</tr>
<tr>
<td>simulator_step</td>
<td>100</td>
<td></td>
<td>time step of simulation [unit:msec]</td>
</tr>
<tr>
<td>send_step</td>
<td>150</td>
<td></td>
<td>time step of visual information [unit:msec]</td>
</tr>
<tr>
<td>recv_step</td>
<td>10</td>
<td></td>
<td>time step of acception of</td>
</tr>
</tbody>
</table>
### Table B.1: (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Current Value in <code>server.conf</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sense_body_step</code></td>
<td>100</td>
<td></td>
<td>commands [unit: msec]</td>
</tr>
<tr>
<td><code>say_msg_size</code></td>
<td>512</td>
<td></td>
<td>string size of say message [unit: byte]</td>
</tr>
<tr>
<td><code>clang_win_size</code></td>
<td>300</td>
<td></td>
<td>time window which controls how many messages can be sent (coach language)</td>
</tr>
<tr>
<td><code>clang_define_win</code></td>
<td>1</td>
<td></td>
<td>number of messages per window</td>
</tr>
<tr>
<td><code>clang_meta_win</code></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>clang_advice_win</code></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>clang_info_win</code></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>clang_mess_delay</code></td>
<td>50</td>
<td></td>
<td>delay between receipt of message and send to players</td>
</tr>
<tr>
<td><code>clang_mess_per_cycle</code></td>
<td>1</td>
<td></td>
<td>maximum number of coach messages sent per cycle</td>
</tr>
<tr>
<td><code>hear_max</code></td>
<td>2</td>
<td></td>
<td>interval of online coach's look</td>
</tr>
<tr>
<td><code>hear_inc</code></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>hear_decay</code></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>catch_ban_cycle</code></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>coach</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>coach_referee</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>old_coach_hear</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>send_yi_step</code></td>
<td>100</td>
<td></td>
<td>interval of online coach's look</td>
</tr>
<tr>
<td><code>use_offside</code></td>
<td></td>
<td></td>
<td>flag for using off side rule</td>
</tr>
<tr>
<td><code>offside_active_area_size</code></td>
<td>5</td>
<td></td>
<td>offside active area size</td>
</tr>
<tr>
<td><code>forbid_kick_off_offside</code></td>
<td></td>
<td></td>
<td>forbid kick off offside</td>
</tr>
<tr>
<td><code>log_file</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>record</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>record_version</code></td>
<td>3</td>
<td></td>
<td>flag for record log</td>
</tr>
<tr>
<td><code>record_log</code></td>
<td></td>
<td></td>
<td>flag for record client command log</td>
</tr>
<tr>
<td><code>record_messages</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>send_log</code></td>
<td></td>
<td></td>
<td>flag for send client command log</td>
</tr>
<tr>
<td><code>log_times</code></td>
<td></td>
<td></td>
<td>flag for writing cycle length to log file</td>
</tr>
<tr>
<td><code>verbose</code></td>
<td></td>
<td></td>
<td>flag for verbose mode</td>
</tr>
<tr>
<td><code>replay</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>offside_kick_margin</code></td>
<td>9.15</td>
<td></td>
<td>offside kick margin</td>
</tr>
<tr>
<td><code>slow_down_factor</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>start_goal_l</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>start_goal_r</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>fullstate_l</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>fullstate_r</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### B.2. Playmodes

The playmodes (including playmodes for viewers) as defined in (server/types.h)

- `PM_Null`
- `PM_BeforeKickOff`
- `PM_TimeOver`
B. Soccerserver

PM_PlayOn,
PM_KickOff_Left,
PM_KickOff_Right,
PM_KickIn_Left,
PM_KickIn_Right,
PM_FreeKick_Left,
PM_FreeKick_Right,
PM_CornerKick_Left,
PM_CornerKick_Right,
PM_GoalKick_Left,
PM_GoalKick_Right,
PM_AfterGoal_Left,
PM_AfterGoal_Right,
PM_Drop_Ball,
PM_OffSide_Left,
PM_OffSide_Right,

// added for 3D viewer/commentator/small league
PM_PK_Left,
PM_PK_Right,
PM_FirstHalfOver,
PM_Pause,
PM_Human,
PM_Foul_Charge_Left,
PM_Foul_Charge_Right,
PM_Foul_Push_Left,
PM_Foul_Push_Right,
PM_Foul_MultipleAttacker_Left,
PM_Foul_MultipleAttacker_Right,
PM_Foul_BallOut_Left,
PM_Foul_BallOut_Right,
PM_MAX
C. Soccermonitor

Following is a description of the data structures that are used for communication to the soccermonitors.

C.1. Monitor Communication Version 1

The container for the 3 different types of information:

typedef struct {
    short mode ;
    union {
        showinfo_t show ;
        msginfo_t msg ;
        drawinfo_t draw ;
    } body ;
} dispinfo_t ;

C.1.1. Showinfo

Contains the information to display a scene.

typedef struct {
    char pmode ;
    team_t team[2] ;
    pos_t pos[MAX_PLAYER * 2 + 1] ;
    short time ;
} showinfo_t ;

typedef struct {
    char name[16]; /* name of the team */
    short score; /* current score of the team */
} team_t ;

typedef struct {
    short enable ;
    short side ;
    short unum ;
}
C. Soccermonitor

    short angle;
    short x;
    short y;
} pos_t;

C.1.2. Messageinfo

Messageinfo_t contains all messages from players to the server and from the referee.

typedef struct {
    short board;
    char message[2048];
} msginfo_t;

C.1.3. Drawinfo

Drawinfo_t allows the server to tell the monitor to draw simple graphics elements.

typedef struct {
    short mode;
    union {
        pointinfo_t pinfo;
        circleinfo_t cinfo;
        lineinfo_t linfo;
    } object;
} drawinfo_t;

typedef struct {
    short x;
    short y;
    char color[COLOR_NAME_MAX];
} pointinfo_t;

typedef struct {
    short x;
    short y;
    short r;
    char color[COLOR_NAME_MAX];
} circleinfo_t;

typedef struct {
    short x1;
    short y1;
    short x2;
}
short y2;
char color[COLOR_NAME_MAX];
} lineinfo_t;

The mode determines the kind of message the union object contains (server/param.h)

DrawClear 0
DrawPoint 1
DrawCircle 2
DrawLine 3

C.2. Monitor Communication Version 2

The container for the 5 different types of information:

typedef struct {
    short mode;
    union {
        showinfo_t2 show;
        msginfo_t msg;
        player_type_t ptinfo;
        server_params_t sparams;
        player_params_t pparams;
    } body;
} dispinfo_t2;

C.2.1. Showinfo

A showinfo_t2 struct is passed every cycle (100 ms) to the monitor and contains the state and positions of players and the ball.

typedef struct {
    char pmode; // the play mode
    team_t team[2]; // team names and scores
    ball_t ball; // ball information
    player_t pos[MAX_PLAYER * 2]; // the 22 players
    short time; // current simulation time
} showinfo_t2;

Values of the elements can be

- pmode: the playmode (see B.2).
- team: structs containing the teams with index 0 beeing the team playing from left to right.
- ball: position information of ball (see above).
C. Soccermonitor

- pos: position information of player (see above) with indices 0 to 10 for team[0] (left to right) and 11 to 21 for team[1].
- time: current time ranging from 1 to 12000 (in extra time)

```c
typedef struct {
    short mode;
    short type;
    long x;
    long y;
    long deltax;
    long deltay;
    long body_angle;
    long head_angle;
    long view_width;
    short view_quality;
    long stamina;
    long effort;
    long recovery;
    short kick_count;
    short dash_count;
    short turn_count;
    short say_count;
    short tneck_count;
    short catch_count;
    short move_count;
    short chg_view_count;
} player_t;

typedef struct {
    long x;
    long y;
    long deltax;
    long deltay;
} ball_t;

typedef struct {
    char name[16];
    short score;
} team_t;
```

C.2.2. Messageinfo

Contains information of the communication between clients and server and messages from the referee.
C.2. Monitor Communication Version 2

typedef struct {
    short board;
    char message[2048]; /* max_message_length_for_display */
} msginfo_t;

- board: indicates the type of message. A message with type MSG_BOARD (1) is a message of the referee for the left text window. LOG_BOARD (2) are messages from and to the players. (server/param.h)
- message: zero terminated string containing the message.

C.2.3. Server Parameters

typedef struct {
} server_params_t;

A complete table of the server parameters can be found in the appendix B.1.

C.2.4. Player Type

typedef struct {
    short id;
    long player_speed_max;
    long stamina_inc_max;
    long player_decay;
    long inertia_moment;
    long dash_power_rate;
    long player_size;
    long kickable_margin;
    long kick_rand;
    long extra_stamina;
    long effort_max;
    long effort_min;

    long sparelong1;
    long sparelong2;
    long sparelong3;
    long sparelong4;
    long sparelong5;
    long sparelong6;
    long sparelong7;
    long sparelong8;
    long sparelong9;
    long sparelong10;
} player_type_t;
C. Soccermonitor

C.2.5. Player Parameters

typedef struct {
    short player_types;
    short subs_max;
    short pt_max;
    long player_speed_max_delta_min;
    long player_speed_max_delta_max;
    long stamina_inc_max_delta_factor;
    long player_decay_delta_min;
    long player_decay_delta_max;
    long inertia_moment_delta_factor;
    long dash_power_rate_delta_min;
    long dash_power_rate_delta_max;
    long player_size_delta_factor;
    long kickable_margin_delta_min;
    long kickable_margin_delta_max;
    long kick_rand_delta_factor;
    long extra_stamina_delta_min;
    long extra_stamina_delta_max;
    long effort_max_delta_factor;
    long effort_min_delta_factor;
    long sparelong1;
    long sparelong2;
    long sparelong3;
    long sparelong4;
    long sparelong5;
    long sparelong6;
    long sparelong7;
    long sparelong8;
    long sparelong9;
    long sparelong10;
    short spareshort1;
    short spareshort2;
    short spareshort3;
    short spareshort4;
    short spareshort5;
}
short spareshort6;
short spareshort7;
short spareshort8;
short spareshort9;
short spareshort10;

} player_params_t;
Index

(ObjName Direction), 27, 70
(ObjName Distance Direction

  [DistChange DirChange [Body-FacingDir HeadFacingDir]]), 69

(ObjName Distance Direction

  DistChange DirChange, 27

(ObjName Distance Direction

  DistChange DirChange Body-FacingDir HeadFacingDir ),

27

(ObjName Distance Direction), 27
(b), 70
(bye), 24, 66
(can’t reconnect), 66
(catch CatchCount), 25, 27, 35
(catch Direction), 25, 67
(change_view ChangeViewCount), 25, 27
(change_view ChangeViewCount), 35
(change_view Width Quality), 25, 69
(dash DashCount), 25, 27, 35
(dash Power), 25, 67
(ear off), 82
(ear on), 82
(error illegal_command_form), 26, 68
(error no_more_team_or_player), 24
(error no_more_team_or_player_or_goalie),

24, 65, 66
(error reconnect), 24, 66
(error unknown_command), 26
(eye off), 85
(eye on), 85
(f FlagInfo), 70
(f b 0), 31
(f c), 30
(f l b), 30
(f p l b), 30
(f r b 10), 31
(f t 1 20), 31
(g Side), 70
(goal r), 30
(head_angle HeadAngle), 25, 27
(head_angle HeadDirection), 35
(hear Time Sender "Message"), 28
(hear Time Sender Message), 70
(init Side UniformNumber PlayMode), 65
(init Side Unum PlayMode), 24
(init TeamName [(version VerNum)]

  [(goalie)]), 24, 65
(kick KickCount), 25, 27, 35
(kick Power Direction), 25, 67
(l ...), 31
(look), 85
(move MoveCount), 25, 27, 35
(move X Y), 25, 68
(ok say), 68
(p [TeamName [Unum]]), 70
(player_param Parameters ...), 66
(player_type id Parameters ...), 66
(reconnect Side PlayMode), 24, 66
(reconnect TeamName UniformNumber), 66
(reconnect TeamName Unum), 24
(say Message), 25, 68
(say SayCount), 25, 27, 35
(score Time OurScore OpponentScore), 69
(score), 25, 69
(see Time ObjInfo ObjInfo ...), 69
(see_global ...), 85
(sense_body), 25, 68
(server_param Parameters ...), 66

130
(speed AmountOfSpeed DirectionOfSpeed), 25, 27, 35
(stamina Stamina Effort), 25, 27, 35
(turn Moment), 25, 67
(turn TurnCount), 25, 27, 35
(turn Neck Angle), 25, 68
(turn Neck TurnNeckCount), 25, 27, 35
(view mode TurnNeckCount), 25, 27, 35

1, 24
11, 24
34, 25
180, 27

abort(), 77
assert, 77
audio_cut_list, 28, 29, 45, 50, 68, 120

B, 32
b, 32, 70
bacce max, 42
ball_accel max, 43, 49, 119
ball_decay, 36, 43, 44, 49, 119
ball_rand, 36, 43, 44, 49, 119
ball_size, 44, 49, 119
ball_speed max, 43, 44, 49, 119
ball_weight, 49, 119

catch, 15
catch_cycle, 37, 38, 51, 121
catch_probability, 37, 38, 49, 120
catchable_area_l, 37, 38, 49, 120
catchable_area_w, 37, 38, 49, 120
change_player_type, 84, 89
change_player_type ..., 46
kick_margin, 50, 120
clang_advice_win, 51, 121
clang_define_win, 50, 121
clang_info_win, 51, 121
clang_mess_delay, 51, 121
clang_mess_per_cycle, 51, 121
clang_meta_win, 50, 121
clang_win_size, 50, 121
coach, 27, 51, 121
coach_port, 50, 120
coch_referee, 51, 121
control_radius, 49, 120
dash, 14, 35, 36, 38, 45, 46, 48
dash_power_rate, 36, 40, 49, 119
dash_power_rate_delta max, 40
dash_power_rate_delta_min, 40
drop_ball_time, 50, 120
edp, 38
effort, 39
effort_dec, 40, 49, 119
effort_dec_hr, 40, 49, 119
effort_inc, 40, 49, 119
effort_inc_hr, 40, 49, 119
effort_max, 38, 40
effort_max_delta_factor, 40
effort_min, 38, 40, 49, 119
effort_min_delta_factor, 40
extra_stamina, 40
extra_stamina_delta_max, 38, 40
extra_stamina_delta_min, 38, 40

F, 32
f, 32, 70
forbid_kick_off_offside, 51, 121
fullstate_l, 51, 121
fullstate_r, 51, 121

g, 32, 70
goal_width, 49, 119
goalie, 27, 30
goalie_max_moves, 37, 38, 43, 44, 49, 120

half_time, 50, 120
hear, 27, 68
hear_decay, 28, 29, 45, 51, 121
hear_inc, 28, 29, 45, 51, 121
hear_max, 28, 29, 45, 51, 121

131
Index

high, 32, 35, 69

inertia\_moment, 45, 46, 50, 120
inertia\_moment\_delta\_factor, 45, 46
inertia\_value, 45
init, 67, 86

kick, 14, 36, 39, 46
kick\_power\_rate, 36, 44, 49, 120
kick\_rand, 42, 44, 49, 119
kick\_rand\_delta\_factor, 42, 44
kick\_rand\_factor\_l, 49, 119
kick\_rand\_factor\_r, 49, 119
kickable\_margin, 42, 44, 49, 120
kickable\_margin\_delta\_max, 44
kickable\_margin\_delta\_min, 44

l, 65, 70
log\_file, 51, 121
log\_times, 51, 121
low, 32, 35, 69

max\_moment, 25, 39, 44-46, 50, 120
max\_neck\_angle, 46, 50, 68, 120
maxparam, 67
max\_power, 25, 38-40, 44, 49, 67, 120
min\_moment, 25, 39, 44-46, 50, 120
min\_neck\_angle, 46, 50, 68, 120
min\_neck\_moment, 25, 46, 50, 120
min\_power, 25, 38-40, 44, 50, 67, 120
move, 13, 37, 43, 48, 85

narrow, 32, 35, 69
normal, 32, 35, 69

off, 82, 85, 86
offs\_side\_active\_area\_size, 51, 121
offs\_side\_kick\_margin, 51, 121
olcoach\_port, 50, 120
old\_coach\_hear, 51, 121
on, 82, 85, 86
on\_coach\_l, 70
on\_coach\_left, 27, 28
on\_coach\_r, 70
online\_coach\_right, 27, 28
online\_coach\_l, 70
P, 32
p, 27, 30, 32, 70
player\_accel\_max, 39, 40, 49, 119
player\_accel\_delta\_max, 40, 45, 49, 120
player\_accel\_delta\_min, 40, 45, 49, 120
player\_rand, 36, 39, 40, 49, 119
player\_size, 49, 119
player\_speed\_max, 39, 40, 45, 49, 119
player\_speed\_max\_delta\_max, 40, 45, 49, 119
player\_speed\_max\_delta\_min, 39, 40
player\_speed\_min, 39, 40, 45, 49, 119
player\_speed\_min\_delta\_max, 39, 40
player\_speed\_min\_delta\_min, 39, 40
player\_types, 46, 47
player\_weight, 49, 119
port, 50, 120
prand\_factor\_l, 49, 119
prand\_factor\_r, 49, 119

quantize\_step, 34, 50, 120
quantize\_step\_dir\_team\_l, 50, 120
quantize\_step\_dir\_team\_r, 50, 120
quantize\_step\_dist\_team\_l, 50, 120
quantize\_step\_dist\_team\_r, 50, 120
quantize\_step\_l, 34, 50, 120
quantize\_step\_r, 65, 70
record, 51, 121
record\_log, 51, 121
record\_messages, 51, 121
record\_version, 51, 121
recover\_dec, 40, 49, 119
recover\_dec\_thr, 40, 49, 119
recover\_min, 40, 49, 119
recovery, 39
recv\_step, 50, 120
referee, 27, 28, 70
replay, 51, 121

say, 28, 43, 45, 68
say\_coach\_cnt\_max, 50, 120
say\_coach\_msg\_size, 50, 120

132
say_msg_size, 28, 43, 45, 50, 121
score, 25
see, 27, 29
self, 27, 28, 70
send_log, 51, 121
send_step, 50, 120
send_vi_step, 51, 121
sense_body, 25, 27, 35
sense_body_step, 35, 50, 121
sense_step, 29, 31, 32, 34
simulator_step, 50, 120
slow_down_factor, 51, 121
soccerserver, 27
stamina, 39
stamina_inc_max, 40, 49, 119
stamina_inc_max_delta_factor, 40
stamina_max, 38, 40, 49, 119
start_goal_l, 51, 121
start_goal_r, 51, 121
subs_max, 46, 47
team_actuator_noise, 49, 119
team_far_length, 32–34
team_too_far_length, 32–34
turn, 14, 45, 46, 48
turn_neck, 45, 46
unum_far_length, 32–34
unum_too_far_length, 32–34
use_offside, 51, 121
verbose, 51, 121
view_angle, 32, 33
view_frequency, 32
view_quality_factor, 32
view_width_factor, 32
visible_angle, 31, 32, 34, 50, 120
visible_distance, 32–34, 50, 120
wide, 32, 35, 69
wind_dir, 39, 40, 43, 44, 50, 120
wind_force, 39, 40, 43, 44, 50, 120
wind_none, 50, 120
wind_rand, 39, 40, 43, 44, 50, 120
wind_random, 50, 120