The Robolog Soccer Server Interface

Prolog Programmers Manual
Edition 2, for Robolog Version 0.5.0
1 November 1998

Oliver Obst

This is the second edition of the Robolog documentation.

Published by the AI Group at University Koblenz
Rheinau 1
56075 Koblenz
Germany
Printed copies are available for $15 each.

Documentation is like sex: when it is good, it is very, very good; and when it is bad, it is better than nothing.

Dick Brandon

Robolog Copying Conditions

The programs currently being distributed that relate to Robolog are *free*; this means that everyone is free to use them and free to redistribute them on a free basis. The Robolog-related programs are not in the public domain; they are copyrighted and there are restrictions on their distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of these programs that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the programs that relate to Robolog, that you receive source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the Robolog related programs, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the programs that relate to Robolog. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the licenses for the programs currently being distributed that relate to Robolog are found in the General Public Licenses that accompany them.

1 Overview of Robolog

Robolog is an interface to RoboCup soccer server for C and ECLiPSe-Prolog. There are C-functions and Prolog-predicates to initialize a new client and to use the commands provided by the soccer server, such as dash or turn. There are other more advanced predicates that perform tasks like evaluating absolute positions or kicking the ball to a desired position.

For writing C programs with Robolog, you have to include the Robolog header file into your program, `#include "robolog.h"`. The C functions are not documented within this manual, this will be subject of another manual (to be written).

A straightforward way to use Robolog with ECLiPSe-Prolog is to use the Robolog boot file when starting ECLiPSe. You can do so by typing `eclipse -b <PATH>/robolog-b.pl`, or use a shell script that does this for you. Once a new client is created, Robolog connects to the server and a background process continuously receives messages which are parsed and stored into memory.

Some of the functions/predicates for controlling the client have to send commands to the soccer server. Robolog translates these predicates into the corresponding soccer server commands and queues them up for delivery. As soon as the soccer server is ready to receive, the commands are dequeued and transmitted.

Since the client perpetually receives new facts, it would be hard (if not impossible) to work on a changing set of facts. To facilitate processing, the client stores the facts for several simulation steps. Unless you tell the client to use data from an other (probably newer) simulation step, the facts you are working with will not change (for at least 100 steps). It is a good idea to refresh the simulation step pointer at the beginning of every new predicate you define.

The following chapters will describe several new predicates, which can be grouped into sections according to their functionality:

2 The A and O of Robolog

There is a predicate to initialize and connect a client to the soccer server and a predicate to disconnect after the game. You can choose if your client should be a normal player or a goal keeper. There can be at most one goalie in a team.

init (+*Player*, +*Name*, [+*Server*, [+*Port*, [+*Protocol*]]]) Predicate

The `init/2` - `init/5` predicates are used to connect a player to the server.A interrupt procedure is set up, which gets new information from the soccer server every 10ms (or so). You should connect only once to the soccer server; that means, for every player on the field you have to have a new prolog process.

+*Player*:

One of `player` or `goalie`, determining the type of the client you want to connect. A goalie client can catch the ball in the goal area. There can be only one goalie client in a team.

+*Name*:

A name for your team. All clients of a team have to use the same name, of course. Don't use blanks within the name — use an underscore (‘`_`’) instead.

+*Server*

The name of the machine the soccer server is running on, something like ‘`foo.bar.org`’. Use ‘`localhost`’ if running the server on the same machine.

+*Port*

The port number the server is using, usually 6000.

+*Protocol*

The protocol version you want to use. Use ‘`4.0`’ for soccer server 4.xx.

stop Predicate

The `stop/0` predicate stops the client from continously receiving data and releases allocated memory. Do not use any Robolog predicates after calling `stop/0`.

3 Go for it!

3.1 Predicates to move the client

RoboCup clients are able to perform two basic actions to move on the soccer field: *dash* and *turn*. Dashing accelerates the client into the direction it is facing; valid accelerations are from -30 up to 100. Negative values accelerate the client into backward direction. The `dash/1` predicate sends the corresponding command to the soccer server.

You can change the direction the client is facing using the `turn/1` predicate; valid angles range from -180 to +180 degrees. However, if the client is dashing fast, the change of direction is reduced.

Additionally, there is one more predicate to move the client: `moveto/2` moves the player to an absolute postion on the field. This predicate is working only in *before_kick_off* mode and 5 seconds after a team scored a goal. (For play modes, see Section 3.3 [referee/1], page 8.)

dash (+*Power*) Predicate

Accelerates the client and decreases its stamina. Each client has a certain stamina and effort. Effort is a real number between 0.1 and 1. It is decreased, if stamina is below `Stamina_Max * 0.4` and it is increased, if stamina is above `Stamina_Max * 0.9` (see Section 3.5.2 [The Stamina Model], page 12).

+*Power*

Accelerate the client according to *Power*. *Power* should be a real or an integer number between -30 and 100. The client is accelerated by `Power * Effort`.

turn (+*Moment*) Predicate

Change the direction of the client.

+*Moment*

Moment should be a real or an integer number between -180 and 180.

moveto (+*X*, +*Y*) Predicate

Move the client to position (X, Y). (0, 0) is the center mark, the X-axis and the Y-axis are facing towards the opponents goal and the right side respectively. Putting the client on the opponents side (i.e. X >= 0) will cause the soccer server to put the client onto a random position back on its side.

X, *Y*

X and *Y* should be real or integer numbers.

3.2 Predicates to act with the ball

Kicking against it is the only thing a normal player can do to the ball. This is what the `kick/2` predicate does. Since *kicking against the ball* is not the same as *shooting the ball*, say, for a certain distance, there is another predicate, `kick_to/2`, that tries to achieve this.Additionally, a goalie client can use the `catch/1` predicate. The goalie client can catch the ball within the penalty area, if the ball is close enough.

kick (+*Power*, +*Dir*) Predicate

Kick the ball by Power *Power* into direction *Dir*.

`kick/2` accelerates the ball into the desired direction, if it is near enough (within about 1.8m, no restrictions regarding the position of the ball, i.e. the ball can be in front or next to or behind the client). **Please note:** the ball cannot be kicked "through" the client.

Power

An integer or a real number, should be between -30 and 100.

Direction

An integer or a real number, should be between -180 and 180.

kick_to (+*Distance*, +*Dir*) Predicate

`kick_to/2` kicks the ball for *Distance* meters into direction *Dir*.

If the distance you want the ball to kick is very far, the client will try to kick several times against the ball to achieve this. However, if the distance is too far, the ball will move only a part of the desired distance.

Distance

A positive integer or a real number. The maximum distance depends on several soccer server parameters (such as *kickable_area*, *ball_decay*, ...) and on the distance of ball and client.
Currently, the maximum distance is about 37m, in contrast to

approximately 27m distance of a single `kick(100,Dir)` (assuming the ball does not move before the kick).

Direction

An integer or a real number, should be between -180 and 180.

`kick_to/2` cannot only be used for performing long distance kicks, but also for putting the ball from one side of the client to the other side. Assume the ball is close to the right of the client, we put the ball behind the client first (Dir == 180) and kick it to the left-hand side of the client.

```
% for ball_dist/3 see the chapter on objects on the field.
[eclipse 6]: ball_dist(-1, Dist, Dir).
Dist = 1.0
Dir = 90.0
yes.
[eclipse 7]: kick_to(1,180).

yes.
[eclipse 8]: kick_to(1,-90).

yes.
[eclipse 9]: kick_to(1,-70).

yes.
[eclipse 10]: kick_to(1,-60).

yes.
[eclipse 11]: ball_dist(-1, Dist, Dir).

Dist = 1.6000000000000001
Dir = -90.0
yes.
```

catch (+*Dir*) Predicate

Goalie catch command.

This predicate is for the goalie only. The goalie can catch within the penalty area, if the ball is closer than two meters in front of the goalie and 0.5m to the left or right of the goalie.After the goalie got the ball, the goalies team gets a free kick and any close opponent client is moved away from the ball for a distance of 9.15m.

Please note: The propability for actually catching the ball with `catch/1` may be below 1.0, depended on the soccer server configuration.

Dir

An integer or a real number for the Direction (-180..180)

3.3 Predicates for communication

The only way soccer clients are allowed to exchange information is via the soccer server. The `say/1` command is for sending messages, which will be broadcasted by the soccer server to clients within a radius of 50m around the sending client. Soccer clients hear messages of other clients automatically, and Robolog stores these messages so that the client can read them with the `message/5` predicate.

Referee messages change the play mode, so there is a predicate `referee/1`, that returns the current play mode.

> *The Robolog message interface ist still experimental and subject to change!*

say (+*String*) Predicate

Send string to the soccer server.

If you want to tell longish stories, please note that the maximum length of a say-string is limited to 256 characters by the soccer server.

String

The string you want to send.

msgidx (*?Time*) Predicate

Returns simulation time of latest read messages.

Time

Unifies with an an integer value.

nextmsgidx (*?Time*) Predicate

Returns simulation time of next unread messages. Fails, if no new messages arrived.

Time

Unifies with an an integer value.

messages (+*Time*, +*Sender*, ?*Number*) Predicate

Returns number of messages received at simulation step *Time*.

Time

An integer value for the simulation step to look at.

Sender

An atom, indicating the sources of the messages. *Sender* can be `self`, `referee`, `other` or `all`.

Number

The number of messages at simulation step *Time*. Unifies with an integer value.

messages (+*Time*, +*Sender*, +*Index*, ?*Message*, ?*Direction*) Predicate

Returns the *Index*'th message of simulation step *Time* from *Sender* clients.

Time

An integer value for the simulation step to look at.

Sender

An atom, indicating the sources of the messages. *Sender* can be `self`, `referee`, `other` or `all`.

Index

The number of the message at simulation step *Time*. The first message of a simulation step has index 0.

Message

A string that unifies with the message.

Direction

If the message if from "other" clients, *Direction* is a real number that unifies with the direction of the sender of the message. Otherwise, *Direction* is 9998.0 in case of own messages and 9999.0 in case of referee messages.

referee (*?PlayMode*) Predicate

Check current playmode

`referee/1` succeeds, if *?PlayMode* unifies with the current play mode.

?PlayMode

A free variable or an atom, that unifies with the current play mode. If *PlayMode* is a free variable, `referee/1` returns one of the following atoms:

- `before_kick_off`
- `kick_off_l`
- `kick_off_r`
- `kick_in_l`
- `kick_in_r`
- `corner_kick_l`
- `corner_kick_r`
- `goal_kick_l`
- `goal_kick_r`
- `free_kick_l`
- `free_kick_r`
- `play_on`
- `half_time`
- `time_over`
- `extend`
- `foul_l`
- `foul_r`
- `goal_l`
- `goal_r`
- `goalie_catch_ball_l`
- `goalie_catch_ball_r`
- `undefined`

3.4 Tuning the Sensors

view (*?Width*) Predicate

Gets / Sets clients width of view.

?Width

An atom or a free variable. If *Width* is a free variable, `view/1` returns the clients current width of view. If *Width* is instantiated, the clients view will be set, if *Width* unifies with one of `normal` (for a 90 degree view cone), `narrow` (for a 45 degree view cone) or `wide` (for a 180 degree view cone).

view (*?Width, ?Quality*) Predicate

Gets / Sets clients width and quality of view.

?Width

An atom or a free variable. If *Width* is a free variable, `view/1` returns the clients current width of view. If *Width* is instantiated, the clients view width will be set, if *Width* unifies with one of `normal` (for a 90 degree view cone), `narrow` (for a 45 degree view cone) or `wide` (for a 180 degree view cone).

?Quality

An atom or a free variable. If *Quality* is a free variable, `view/2` returns the clients current quality of view. If *Quality* is instantiated, the clients view quality will be set. Use `high` (default) or `low` as view qualities.

3.5 Effects of clients' actions

3.5.1 The Command Queue

There are some additional things to know about the predicates you can use to perform actions with your client. At first, it is important to know that it is not guaranteed that a command reaches the soccer server. The protocol used for exchanging data between client and server is UDP/IP, so some commands may get lost. Second, clients are not allowed to send commands any time, but only every 100ms. Robolog sends commands to the server only when it is allowed to do so, you don't need to worry about that. However, performing a lot of actions in a short time can be a problem, the `wait_for_robo/0` predicate deals with it.

wait_for_robo Predicate

The soccer server can receive commands like '`kick`' and '`turn`' in intervals of 100ms. Using these commands very quickly causes the queue of outgoing commands to fill up. To wait until the last command reached the soccer server use `wait_for_robo/0`. It will return when the queue of outgoing commands is empty.

Doing `wait_for_robo/0` before every sensing predicate is a safe bet, but will slow down the performance of your client.

3.5.2 The Stamina Model

Every client has a certain stamina, the `dash/1` predicate will consume. At the beginning of a half, a client has a stamina of *MaxStamina*, *effort* of 1.0 and *recovery* of 1.0. A `dash(`X`)` reduces stamina by X, while each simulation step increases stamina by *recovery* `*` *MaxRecovery* (up to the maximum).

Each simulation step stamina is below `0.4 *` *MaxStamina*, *effort* is reduced by `0.05` up to a minimum of `0.1`. A `dash(`X`)` will be executed as `dash(`*effort* `*` X`)`, but consume X stamina.

effort will increase again by `0.05` each simulation step, if *stamina* is higher than `0.9 *` *MaxStamina*.

Each simulation step *stamina* is below `0.3 *` *MaxStamina*, *recovery* is reduced by `0.05` up to a minimum of `0.1`.

4 Inventory.

This chapter deals with the predicates related to the clients' status.

4.1 Teamnames and Uniformnumber

A client receives its uniform number after connecting to the soccer server. Get the number with the `uniformnumber/1` predicate.If your clients have amnesia, there is a predicate that returns the teamname. You can also use `teamname/2` to get the teamname of the opponent team.

uniformnumber (*?N*) Predicate

N

Unifies with an an integer value between 1 and 11.

teamname (+*Team*, *?Name*) Predicate

Team

An atom indicating what teamname you want to know. `self` stands for your own teamname and `other` is for the opponents team.

Name

A string that unifies with the name you want to know.

4.2 Where am I?

Some predicates that reveal information about the clients position on the field and and which side the goals have to be scored.

self (+*Time*, *?X*, *?Y*, *?Dir*) Predicate

Gets the clients absolute position.

Use this predicate with free variables X, Y and *Dir* to retrieve clients absolute position on the field. Robolog estimates the clients position by the relative distances of the landmarks on the soccer field.

Please note: Coordinates for `self/4` are *not* the same as with `moveto/2`. The X-Axis and the Y-Axis are towards the right goal and the bottom[1] respectively. Direction 0 is towards the right goal, direction increases if turning *clockwise*. All predicates (except `moveto/2`) are using this grid.

Sometimes it is not possible for a client to estimate its own position. `self/4` will not fail, but return a value less then -9000 to indicate that a coordinate is unknown.

```
[eclipse 100]: self(42, X, Y, Dir).
X = -9999.0
Y = -9999.0
Dir = -9999.0
yes.
```

X

Unifies with a number. If the client is inside the field, if -52.5 <= *X* <= 52.5.

Y

Unifies with a number. The client is inside the field, if -34 <= *Y* <= 34.

Dir

Also unifies with a number. -180 <= *Dir* <= 180.

myside (*?Side*) Predicate

What side of the field do we play on?

?Side

An atom, either `l` or `r`, for the left or the right side, respectively.

goal_side (*?Side*) Predicate

Where do we have to score? – the opposite predicate to `myside/1`.

?Side

[1] i.e. the right side for the left team or the left side for the right team

An atom, either `l` or `r`, for the left or the right side, respectively.

speed (*?Speed*) Predicate

The `sense_body/0` predicate queries the soccer server for the clients speed (among other things). The information the clients gets is returned by the `speed/1` predicate.

See Section 4.5 [Queries], page 18.

?Speed

Unifies with a real number.

4.3 Who is Who?

There are three predicates that support identification of the goalie client. The clients cannot distinguish the goalie from other clients by visual data, but there are other things a client can do to get this information: the goalie client could tell the other clients its uniform number at the beginning of the game, for example, or the client estimates the goalie by looking who is next to the goal.

goalie/0 Predicate

Succeeds if the client itself is the goalie.

goalie (+*Team*, *?Nr*) Predicate

Returns goalie uniform number of own/other team, fails if unknown.

+*Team*

An atom, `self` for the own team and `other` for the opponents team.

?Nr

An integer that unifies with the uniform number of the goalie client, a value between 1 and 11.

set_goalie (+*Team*, -*Nr*) Predicate

Sets the goalie uniform number of own/other team.

+*Team*

An atom, `self` for the own team and `other` for the opponents team.

-*Nr*

An integer for the uniform number of the goalie client, a value between 0 and 11. Use 0 to set the value to undefined.

4.4 Counting Events

Predicates for the current simulation time and other various counting things.

simstep (*?Time*) Predicate

Returns the simulation time (i.e. the number of simulation steps). A whole game lasts 6000 simulation steps or 10 minutes.

?Time

An integer value that unifies with the current simulation step.

roboclock (*?Clock*) Predicate

Returns the number of timer interrupts since the client connected to the soccer server. The counter is increased every 10ms and can be used as a high resolution timer.

?Clock

An integer value that unifies with the number of timer interrupts.

stamina (*?Stamina*) Predicate

Unifies with the current stamina of the client. Robolog tries to keep track of the stamina itself (may be not exactly); `sense_body` requests the exact information. (See Section 4.5 [sense_body], page 18.)

effort (*?Effort*) Predicate

Unifies with the current effort of the client. Robolog tries to keep track of the effort itself (may be not exactly); `sense_body` requests the exact information. (See Section 4.5 [sense_body], page 18.)

recovery (*?Recovery*) Predicate

Unifies with the current recovery of the client. Robolog tries to keep track of the recovery itself (may be not exactly); however, there is no way to get the exact information.

dashs (*?N*) Predicate

Returns the number of dashs the client performed until now.

N

Unifies with an an integer value.

kicks (*?N*) Predicate

Returns the number of (simple) kicks the client performed until now.

N

Unifies with an an integer value.

says (*?N*) Predicate

Returns the number of say commands the client performed until now.

N

Unifies with an an integer value.

turns (*?N*) Predicate

Returns the number of turn commands the client performed until now.

N

Unifies with an an integer value.

goals (+*Time*, +*Team*, ?*N*) Predicate

Returns the number of goals for own/other team.

+*Time*

The simulation step you want to look at. Use -1 for the latest information.

+*Team*

An atom, `self` for the own team and `other` for the opponents team.

-*N*

The number of goals a team scored.

4.5 Queries to the Soccer Server

The `sense_body` predicates query the soccer server for information on the client.

sense_body Predicate

Query the soccer server about clients status. See Section 4.5 [sense_body/9], page 18.

sense_body (?*View_width*, ?*View_quality*, ?*Stamina*, ?*Effort*, ?*Speed*, ?*Kicks*, ?*Dashs*, ?*Turns*, ?*Says*) Predicate

Query the soccer server about clients status and returns all the information. This predicate waits until the server sends the information and therefore may be a little slower than `sense_body/0`.

View_width

A free variable or an atom that unifies with the clients width of view. Valid atoms are `normal` for a 90 degree view cone, `narrow` for a 45 degree view cone and `wide` for a 180 degree view cone.

View_quality

A free variable or an atom that unifies with the clients view quality. Valid atoms are `high` for detailed information about position of objects and `low` for reduced information, where only directions of objects will be transmitted.

Stamina

A free variable or an integer that unifies with the clients stamina. Stamina decreases when the client dashes, and increases by `recovery` until it reaches the maximum.
(See Section 3.5.2 [The Stamina Model], page 12.)

Effort

A free variable or a real number that unifies with the clients effort. Effort decreases until it reaches the minimum (0.1), if the clients stamina is below `0.4 * MaxStamina`; it increases constantly until it reaches the maximum (1.0), if the clients stamina is above `0.9 * MaxStamina`.
(See Section 3.5.2 [The Stamina Model], page 12.)

?Speed

Speed of the client. Unifies with a real number.

Kicks

A free variable or an integer that unifies with the number of kicks a client performed.

Dashs

A free variable or an integer that unifies with the number of dashs a client performed.

Turns

A free variable or an integer that unifies with the number of turns a client performed.

Says

A free variable or an integer that unifies with the number of says a client performed.

5 About other players, flags and goals

5.1 The `object/10` predicate

There is only one important predicate you need to know to get information about all objects on the field: `object/10`. All other predicates described within this chapter can be derived from this predicate.

object (+*Time*, *?Name*, *?Id*, *?Dist*, *?Dir*, *?DDist*, *?DDir*, *?X*, *?Y*, *?FaceDir*) Predicate

A non-deterministic predicate to get information about objects on the field.

The first parameter +*Time* indicates the simulation step, you want information about. See Section 4.4 [simstep/1], page 16.

The object you want the information about is fixed with *?Name* and *?Id*.

The succeeding parameters return the distance *Dist*, direction *Dir*, distance change *DDist*, direction change *DDir* and absolute coordinates *X* and *Y* of an object. A client knows about the distance and direction of an other object, if either it is in its view cone or if the other client is closer than 3 meters (any direction). *DDist* and *DDir* are known, if the client sees the object. Absolute coordinates of an other object are known, if it is a fixed object (such as a goal or a flag) or if the client knows about the objects relative position and about its own absolute position. The *?FaceDir* is applicable for other clients only and returns the direction the other client is facing.

Any return value less than -9000 can be considered unknown. `object/10` fails, if no (more) object is known.

?Time

An integer value that unifies with a simulation step. Use -1 for the latest known simulation step.

?Name

An atom that unifies with the name of an object. Valid values are `ball`, `goal`, `partner`, `opponent`, `player`, `flag` and `line`.

The clients sees other clients as a `player` clients, if they are too far away to be distinguished.

?Id

An atom or an integer value, depending on the name of the object. Other clients have numeric Id's, indicating the uniform number. A value less than -9000 means that the object Id is unknown to

the client.
The balls' Id is 0, all other objects have symbolic Id's.

?Dist

A real number that unifies with the distance of the object. A value less than -9000 means that the distance is unknown to the client.

?Dir

A real number that unifies with the direction of the object. A value less than -9000 means that the direction is unknown to the client.

?DDist

A real number that unifies with the distance change of the object. A value less than -9000 means that the distance change is unknown to the client.

?DDir

A real number that unifies with the direction change of the object. A value less than -9000 means that the direction change is unknown to the client.

?X

A real number that unifies with the X coordinate of the object. A value less than -9000 means that this coordinate is unknown to the client.

?Y

A real number that unifies with the Y coordinate of the object. A value less than -9000 means that this coordinate is unknown to the client.

?FaceDir

A real number that unifies with the direction an other client is facing. A value less than -9000 means that the facing direction of the object is unknown to the client or not applicable.

5.2 Where is the ball?

ball (+*Time*, *?Id*, *?Dist*, *?Dir*, *?DDist*, *?DDir*, *?X*, *?Y*, *?FaceDir*) Predicate

Get ball coordinates and ball speed, like calling `object/10` with *?Name* `ball`. Since there is only one ball, the *?Id* is always 0. *?FaceDir* is always -9999.0.

5.3 Other clients

opponent (+*Time*, *?Id*, *?Dist*, *?Dir*, *?DDist*, *?DDir*, *?X*, *?Y*, *?FaceDir*) Predicate
partner (+*Time*, *?Id*, *?Dist*, *?Dir*, *?DDist*, *?DDir*, *?X*, *?Y*, *?FaceDir*) Predicate
player (+*Time*, *?Id*, *?Dist*, *?Dir*, *?DDist*, *?DDir*, *?X*, *?Y*, *?FaceDir*) Predicate

Get information about other clients. These predicates are like calling the object predicate with +*Name* unifying with `opponent`, `partner` or `player`, respectively. *?Id* unifies with the uniform number of the clients. A client recognizes uniform numbers of other clients within a distance of 20, max. 40 meters.

5.4 The Goals

goal (+*Time*, *?Id*, *?Dist*, *?Dir*, *?DDist*, *?DDir*, *?X*, *?Y*, *?FaceDir*) Predicate

Get goal coordinates. Like calling the object predicate with `goal` for *Name*

Relative coordinates of a goal are known if either the client sees the goal or if the client knows about its own absolute position. If a client fails to estimate its own position and does not see the goal, neither, it is not possible to calculate the goals distance and direction. A distance resp. direction of less than -9000 indicates an unknown value.

?Id is an atom, where 'l' stands for the left goal, while 'r' stands for the right goal. Values other than 'l' or 'r' give an 'out of range' error.

Since *?FaceDir* is not applicable for goals, it always unifies with -9999.0.

5.5 Flags and Lines

The flags and lines on the field are for the clients' orientation. Calling the `flag/9` or the `line/9` predicate is like calling the `object/10` predicate with `flag` resp. `line` for *?Name*.

flag (+*Time*, *?Id*, *?Dist*, *?Dir*, *?DDist*, *?DDir*, *?X*, *?Y*, *?FaceDir*) Predicate
line (+*Time*, *?Id*, *?Dist*, *?Dir*, *?DDist*, *?DDir*, *?X*, *?Y*, *?FaceDir*) Predicate

?Id is a symbolic value for the position of the flag/line. There are 53 flags and 4 lines on the field.

?FaceDir is always -9999.0.

5.6 Number of Objects seen

These predicates return the number of objects seen at a time.

flag (-*Time*, ?*Flags*) Predicate

goal (-*Time*, ?*Goals*) Predicate

line (-*Time*, ?*Lines*) Predicate

Returns the number of lines in front (i.e. 0 degrees) of the client. (=> if the client "sees" two lines, it is outside of the soccer field).

opponent (-*Time*, ?*Opponents*) Predicate

player (-*Time*, ?*Players*) Predicate

partner (-*Time*, ?*Partners*) Predicate

5.7 Shortcuts

There are shortcuts for all predicates in the previous sections, so you don't always have to use predicates with arity 9 or 10.

The `_xy` predicates return the objects absolute position, the `_dist` predicates the distance and direction, the `_ddist` predicates the distance change and the direction change and finally, the `_face` predicates return the face direction (only for clients).

Here is the complete list:

ball_xy (+*Time*, ?*X*, ?*Y*) Predicate
ball_dist (+*Time*, ?*Dist*, ?*Dir*) Predicate
ball_ddist (+*Time*, ?*DDist*, ?*DDir*) Predicate

flag_xy (+*Time*, ?*Id*, ?*X*, ?*Y*) Predicate
flag_dist (+*Time*, ?*Id*, ?*Dist*, ?*Dir*) Predicate
flag_ddist (+*Time*, ?*Id*, ?*DDist*, ?*DDir*) Predicate

goal_xy (+*Time*, ?*Id*, ?*X*, ?*Y*) Predicate
goal_dist (+*Time*, ?*Id*, ?*Dist*, ?*Dir*) Predicate
goal_ddist (+*Time*, ?*Id*, ?*DDist*, ?*DDir*) Predicate

line_xy (+*Time*, ?*Id*, ?*X*, ?*Y*) Predicate
line_dist (+*Time*, ?*Id*, ?*Dist*, ?*Dir*) Predicate
line_ddist (+*Time*, ?*Id*, ?*DDist*, ?*DDir*) Predicate

object_xy (+*Time*, ?*Name*, ?*Id*, ?*X*, ?*Y*) Predicate
object_dist (+*Time*, ?*Name*, ?*Id*, ?*Dist*, ?*Dir*) Predicate
object_ddist (+*Time*, ?*Name*, ?*Id*, ?*DDist*, ?*DDir*) Predicate
object_face (+*Time*, ?*Name*, ?*Id*, ?*FaceDir*) Predicate

opponent_xy (+*Time*, *?Id*, *?X*, *?Y*) Predicate
opponent_dist (+*Time*, *?Id*, *?Dist*, *?Dir*) Predicate
opponent_ddist (+*Time*, *?Id*, *?DDist*, *?DDir*) Predicate
opponent_face (+*Time*, *?Id*, *?FaceDir*) Predicate

partner_xy (+*Time*, *?Id*, *?X*, *?Y*) Predicate
partner_dist (+*Time*, *?Id*, *?Dist*, *?Dir*) Predicate
partner_ddist (+*Time*, *?Id*, *?DDist*, *?DDir*) Predicate
partner_face (+*Time*, *?Id*, *?FaceDir*) Predicate

player_xy (+*Time*, *?Id*, *?X*, *?Y*) Predicate
player_dist (+*Time*, *?Id*, *?Dist*, *?Dir*) Predicate
player_ddist (+*Time*, *?Id*, *?DDist*, *?DDir*) Predicate
player_face (+*Time*, *?Id*, *?FaceDir*) Predicate

6 Miscellaneous Predicates

6.1 Supporting qualitative reasoning

There are two predicates to support qualitative spatial reasoning. For this version of Robolog, the quality predicates are rather simple and only a first step of what could (and hopefully will) be done in future versions.

dir_quality (+*Degrees*, ?*Direction*) Predicate

Translates an angle +*Degrees* into an atom ?*Direction*.

`n`	from -22.5 to 22.5
`ne`	from 22.5 to 67.5
`e`	from 67.5 to 112.5
`se`	from 112.5 to 157.5
`s`	from 157.5 to 180 and from -180 to -157.5
`sw`	from -157.5 to -112.5
`w`	from -112.5 to -67.5
`nw`	from -67.5 to -22.5

dist_quality (+*Meters*, ?*Distance*) Predicate

Translates a number +*Meters* into an atom ?*Distance*.

`close`	if +*Meters* is smaller than `kickable_area` (currently 1.885 meters).
`near`	if +*Meters* is smaller than 20 (and bigger than `kickable_area`).
`short`	if +*Meters* is smaller than 40 (and bigger than 20).
`far`	if +*Meters* is smaller than 60 (and bigger than 40).
`remote`	else

6.2 Some trigonometry

atan2 (-*X*, -*Y*, +*Result*) Predicate

Unifies +*Result* with the arc tangent of the two numbers X and Y. It is similar to calculating the arc tangent of Y / X, except that the signs of both arguments are used to determine the quadrant of the result.

dist (-*X*, -*Y*, ?*Dist*) Predicate

Returns distance (in meters) from clients current position to (X, Y). `dist/3` fails, if the clients position is unknown.

absangle (-*X*, -*Y*, ?*Dir*) Predicate

Returns direction of (X, Y) from the clients point of view. `absangle/3` fails, if the clients position is unknown.

deg2rad (-*Deg*, +*Rad*) Predicate

Transform degrees into radiants.

rad2deg (-*Rad*, +*Deg*) Predicate

Transform radiants into degrees.

Symbolic object identifier

These constants are used to distinguish the different objects of an object class (the third parameter of `object/10`). The numbers in the brackets represent the coordinate of the object on the soccer field.

Goal identifier

`l`	Left goal (-52.5, 0)
`r`	Right goal (52.5, 0)

Line identifier

`line_t`	Top line (0, -34)
`line_b`	Bottom line (0, 34)
`line_l`	Left line (-52.5, 0)
`line_r`	Right line (52.5, 0)

Flag identifier

`flag_l_t`	Left top corner flag (-52.5, -34)
`flag_l_b`	Left bottom corner flag (-52.5, 34)
`flag_c_t`	Center top line flag (0, -34)
`flag_c_b`	Center bottom line flag (0, 34)
`flag_r_t`	Right top corner flag (52.5, -34)
`flag_r_b`	Right bottom corner flag (52.5, 34)
`pflag_l_t`	Left top penalty flag (-36, -20.16)
`pflag_l_c`	Left center penalty flag (-36, 0)
`pflag_l_b`	Left bottom penalty flag (-36, 20.16)
`pflag_r_t`	Right top penalty flag (36, -20.16)
`pflag_l_c`	Right center penalty flag (36, 0)
`pflag_l_b`	Right bottom penalty flag (36, 20.16)
`flag_c`	Center flag (0, 0)

`gflag_l_t`
: Left top goal flag (-52.5, -7.32)

`gflag_l_b`
: Left bottom goal flag (-52.5, 7.32)

`gflag_r_t`
: Right top goal flag (52.5, -7.32)

`gflag_r_b`
: Right bottom goal flag (52.5, 7.32)

`oflag_t_l_10`
: Outside, top left (-10, -39)

`oflag_t_l_20`
: Outside, top left (-20, -39)

`oflag_t_l_30`
: Outside, top left (-30, -39)

`oflag_t_l_40`
: Outside, top left (-40, -39)

`oflag_t_l_50`
: Outside, top left (-50, -39)

`oflag_t_r_10`
: Outside, top right (10, -39)

`oflag_t_r_20`
: Outside, top right (20, -39)

`oflag_t_r_30`
: Outside, top right (30, -39)

`oflag_t_r_40`
: Outside, top right (40, -39)

`oflag_t_r_50`
: Outside, top right (50, -39)

`oflag_b_l_10`
: Outside, bottom left (-10, 39)

`oflag_b_l_20`
: Outside, bottom left (-20, 39)

`oflag_b_l_30`
: Outside, bottom left (-30, 39)

`oflag_b_l_40`
: Outside, bottom left (-40, 39)

`oflag_b_l_50`
: Outside, bottom left (-50, 39)

`oflag_b_r_10`
: Outside, bottom right (10, 39)

`oflag_b_r_20`
: Outside, bottom right (20, 39)

`oflag_b_r_30`
: Outside, bottom right (30, 39)

`oflag_b_r_40`
: Outside, bottom right (40, 39)

`oflag_b_r_50`
: Outside, bottom right (50, 39)

`oflag_l_t_10`
: Outside, left top (-57.5, -10)

`oflag_l_t_20`
: Outside, left top (-57.5, -20)

`oflag_l_t_30`
: Outside, left top (-57.5, -30)

`oflag_l_b_10`
: Outside, left bottom (-57.5, 10)

`oflag_l_b_20`
: Outside, left bottom (-57.5, 20)

`oflag_l_b_30`
: Outside, left bottom (-57.5, 30)

`oflag_r_t_10`
: Outside, right top (57.5, -10)

`oflag_r_t_20`
: Outside, right top (57.5, -20)

`oflag_r_t_30`
: Outside, right top (57.5, -30)

`oflag_r_b_10`
: Outside, right bottom (57.5, 10)

`oflag_r_b_20`
: Outside, right bottom (57.5, 20)

`oflag_r_b_30`
: Outside, right bottom (57.5, 30)

`oflag_t_0`
: Outside, top center (0, -39)

`oflag_b_0`
: Outside, bottom center (0, 39)

`oflag_l_0`
: Outside, left center (-57.5 0)

`oflag_r_0`
: Outside, right center (57.5 0)

Predicate Index

This is an alphabetical list of all Robolog predicates.

Concept Index

Short Contents

Table of Contents