Hints for RoboNewbie

Hans-Dieter Burkhard
June 2014

Resources

Required special resources, download from

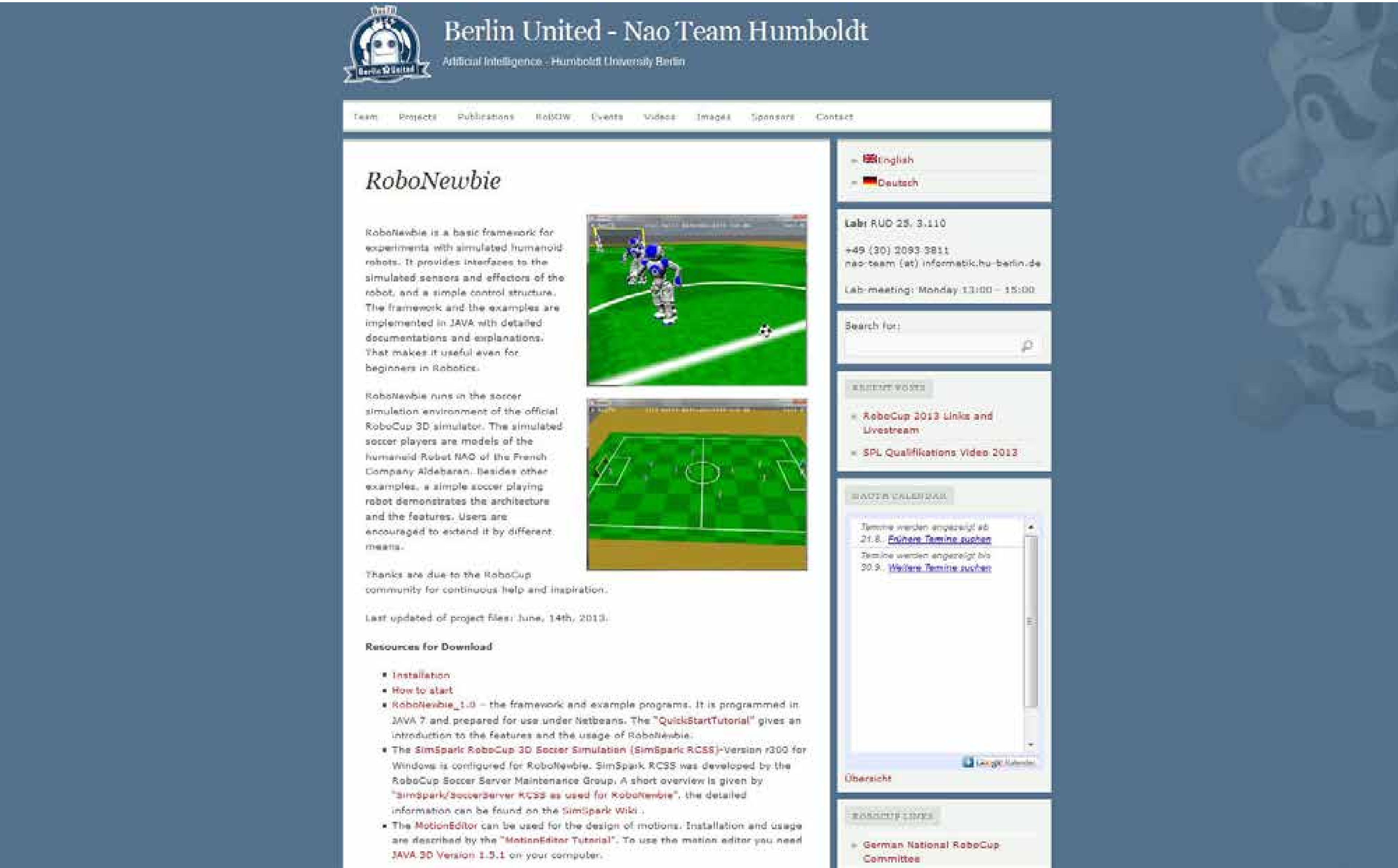
http://www.naoteamhumboldt.de/projects/robonewbie

1. RoboNewbie
2. MotionEditor
3. SimSpark RoboCup 3D Soccer Simulation (SimSpark RCSS)

Additional materials for installation on that page.

Programs and related instructions are available on

http://www.naoteamhumboldt.de/projects/robonewbie/



RoboNewbie

RoboNewbie is a basic framework for experiments with simulated humanoid robots. It provides interfaces to the simulated sensors and effectors of the robot, and a simple control structure. The framework and the examples are implemented in JAVA with detailed documentations and explanations. That makes it useful even for beginners in Robotics.

RoboNewbie runs in the soccer simulation environment of the official RoboCup 3D simulator. The simulated soccer players are models of the humanoid Robot NAO of the French Company Aldebaran. Besides other examples, a simple soccer playing robot demonstrates the architecture and the features. Users are encouraged to extend it by different means.

Thanks are due to the RoboCup community for continuous help and inspiration.

Last updated of project files: June, 14th, 2013.

Resources for Download

- Installation
- How to start
- RoboNewbie_1.0 – the framework and example programs. It is programmed in JAVA 7 and prepared for use under Netbeans. The "QuickStartTutorial" gives an introduction to the features and the usage of RoboNewbie.
- The SimSpark RoboCup 3D Soccer Simulation (SimSpark RCSS)-Version r300 for Windows is configured for RoboNewbie. SimSpark RCSS was developed by the RoboCup Soccer Server Maintenance Group. A short overview is given by "SimSpark/SoccerServer RCSS as used for RoboNewbie", the detailed information can be found on the SimSpark Wiki .
- The MotionEditor can be used for the design of motions. Installation and usage are described by the "MotionEditor Tutorial". To use the motion editor you need JAVA 3D Version 1.5.1 on your computer.

Lab: RUD 25, 3.110

+49 (30) 2093 3811
nao-team (at) informatik.hu-berlin.de

Lab-meeting: Monday 13:00 – 15:00

Search for:

RECENT POSTS

- RoboCup 2013 Links and Livestream
- SPL Qualifikations Video 2013

NAOTH CALENDAR

Termine werden angezeigt ab 21.8. Frühere Termine suchen

Termine werden angezeigt bis 30.9. Weitere Termine suchen

Übersicht

ROBOCUP LINKS

- German National RoboCup Committee

Simulation

Communication
via protocols (TCP)

<u>Effector messages</u>

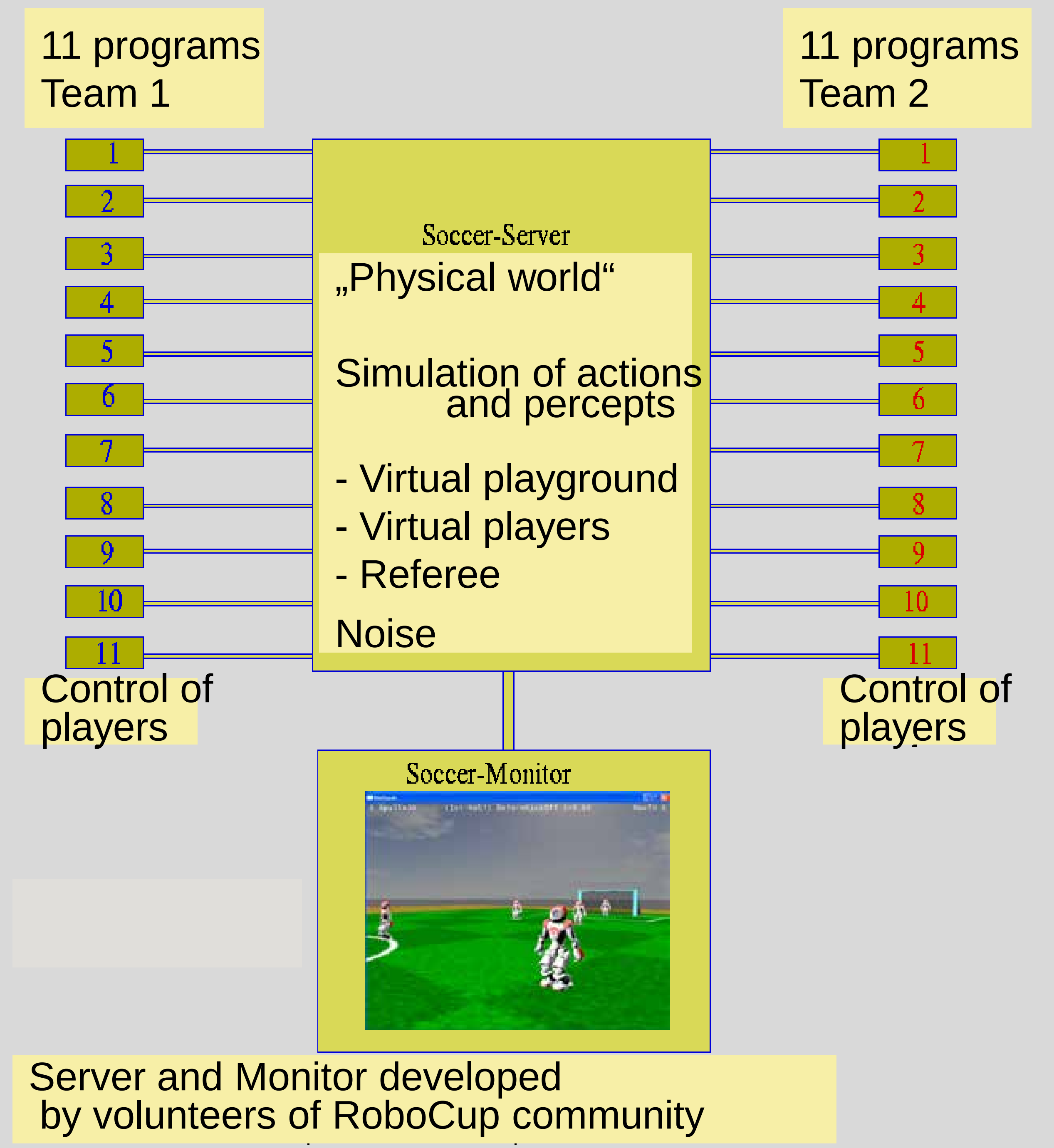
Motor commands

similar to real robot

<u>Perceptor messages</u>

Vision, acoustic, inertial,

….



11 programs
Team 1

11 programs
Team 2

Soccer-Server

„Physical world“

Simulation of actions
and percepts

- Virtual playground
- Virtual players
- Referee

Noise

Control of
players

Control of
players

Soccer-Monitor

Server and Monitor developed
by volunteers of RoboCup community

Open Software

You can make your own experiences by using open software from RoboCup community (explore the internet):

- 3D-Simulation League:
  SimSpark (Server + Monitor)

Thanks to
RoboCup Community

http://simspark.sourceforge.net/wiki

- RoboNewbie Agents of NaoTeam Humboldt

All resources are placed on
our web page (NaoTeam Humboldt)

Thanks to
NaoTeam Humboldt
Magma Offenburg

Start of programs

- Start the server with „rcssserver3d.exe“
- Start your (example) program in NetBeans
- Klick “k” in the monitor window (“kick-off”)
- Klick “b” in the monitor window (“play-on”)

According to soccer rules,
game state should be “play-on”,
because otherwise players are not
allowed to cross over middle line

Simulation Cycle

Cycles (basically 20 msec) with the following steps:

- server sends individual server message with perceptor values (“sensations”) to the agents.
- agents can process perceptor values
- agents can make decisions for next actions
- agent can send agent messages with effector commands
- server collects the effector commands of all agents and calculates resulting new situations

Note that messages are interleaved (next slide)!

Synchronization Server/Agent

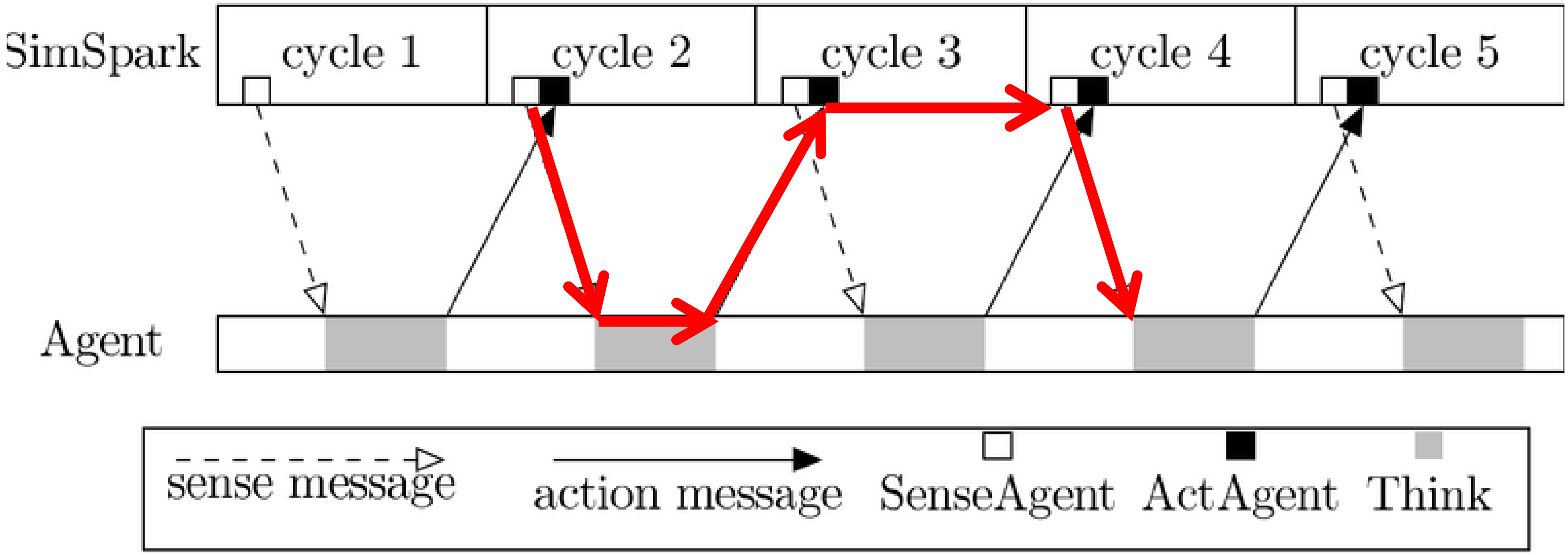
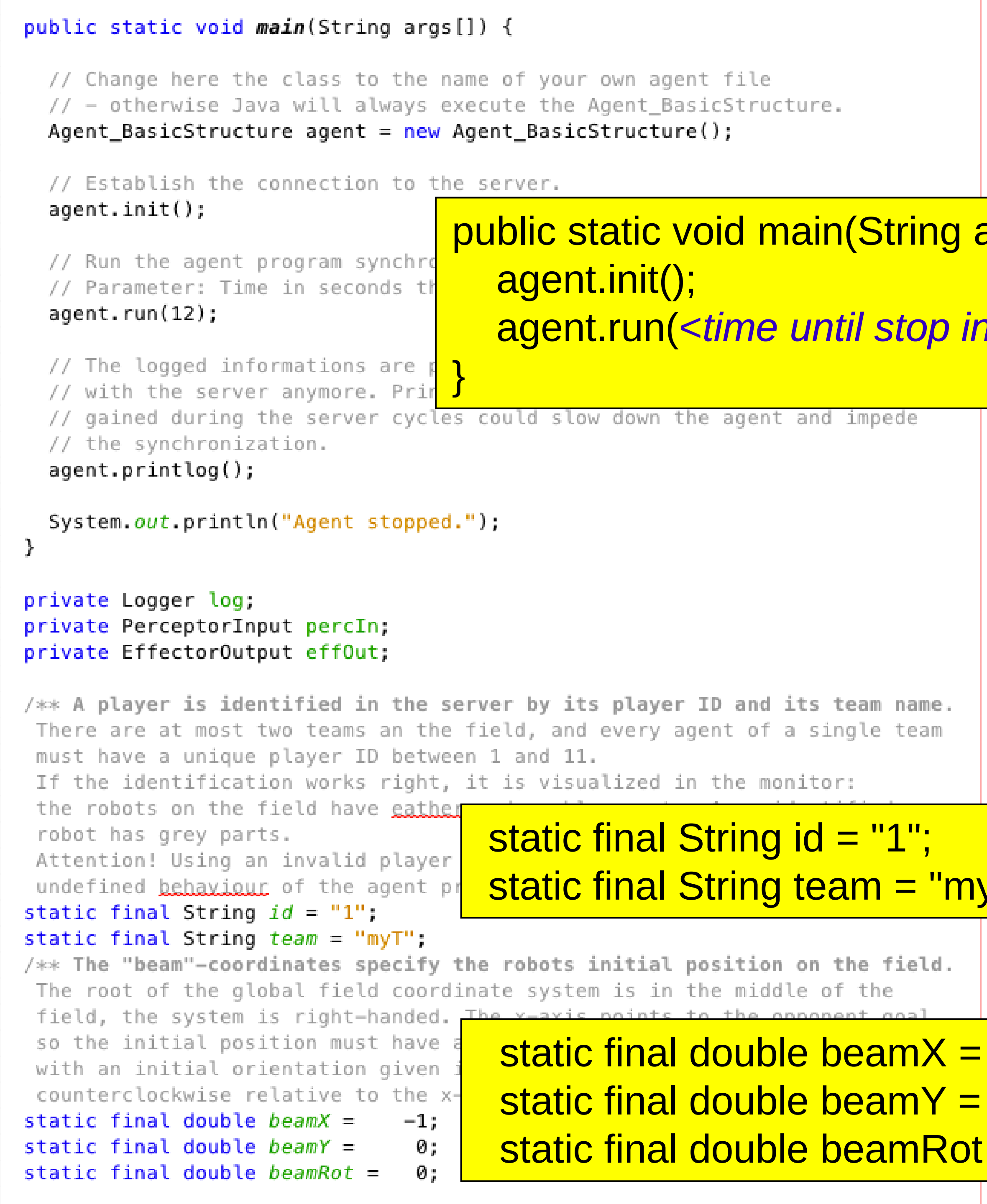


Figure from the SimSpark-Wiki :
http://simspark.sourceforge.net/wiki/i



```
public static void main(String args[]) {

  // Change here the class to the name of your own agent file
  // - otherwise Java will always execute the Agent_BasicStructure.
  Agent_BasicStructure agent = new Agent_BasicStructure();

  // Establish the connection to the server.
  agent.init();

  // Run the agent program synchro
  // Parameter: Time in seconds th
  agent.run(12);

  // The logged informations are p
  // with the server anymore. Prin
  // gained during the server cycles could slow down the agent and impede
  // the synchronization.
  agent.printlog();

  System.out.println("Agent stopped.");
}

private Logger log;
private PerceptorInput percIn;
private EffectorOutput effOut;

/** A player is identified in the server by its player ID and its team name.
 There are at most two teams an the field, and every agent of a single team
 must have a unique player ID between 1 and 11.
 If the identification works right, it is visualized in the monitor:
 the robots on the field have eather
 robot has grey parts.
 Attention! Using an invalid player
 undefined behaviour of the agent pr
static final String id = "1";
static final String team = "myT";
/** The "beam"-coordinates specify the robots initial position on the field.
 The root of the global field coordinate system is in the middle of the
 field, the system is right-handed. The x-axis points to the opponent goal
 so the initial position must have a
 with an initial orientation given i
 counterclockwise relative to the x-
static final double beamX =    -1;
static final double beamY =     0;
static final double beamRot =   0;
```

public static void main(String args[ ]) {
    agent.init();
    agent.run(*<time until stop in seconds>*);
}

Player Identification (example)

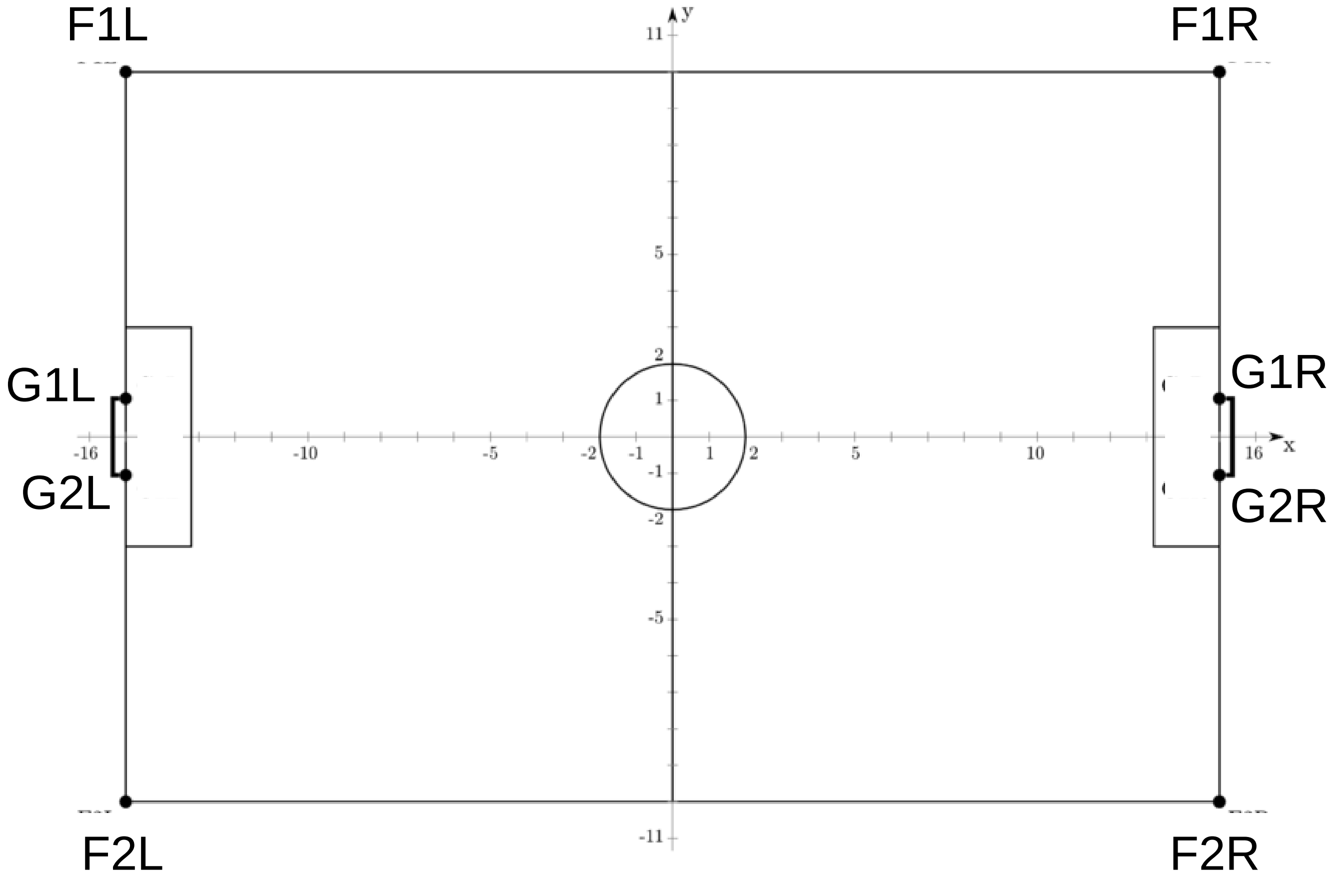
static final String id = "1";
static final String team = "myT";

initial position (“beam”) (example)

static final double beamX = -1;
static final double beamY = 0;
static final double beamRot = 0;

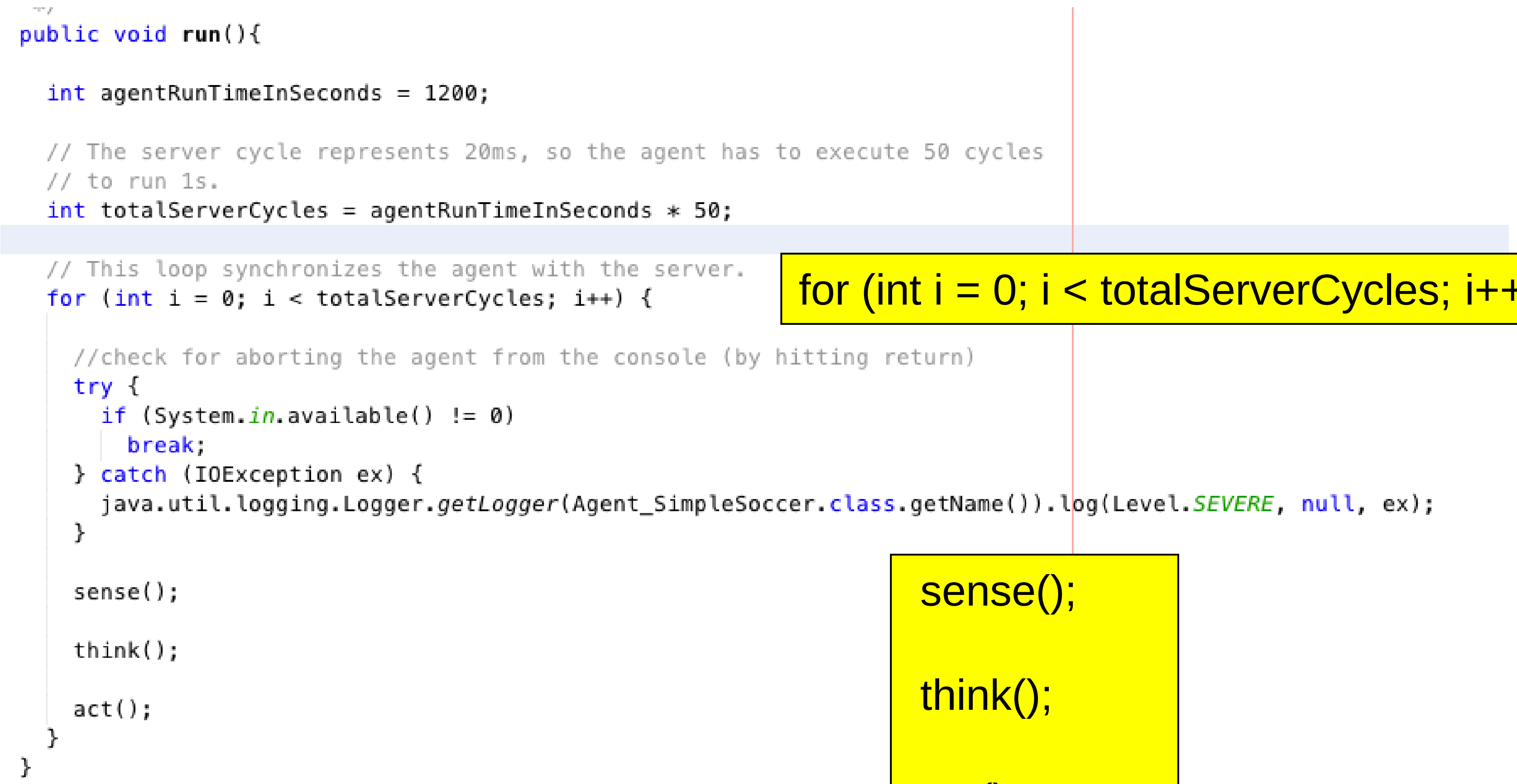
```
static final double beamX =    -1;
static final double beamY =     0;
static final double beamRot =   0;
```

initial position must be
in the own (left) half,
i.e. beamX *must be negative*



Actual sizes in our distribution are 10x7 m

Basic cycle in Agent_SimpleSoccer



```
public void run(){

  int agentRunTimeInSeconds = 1200;

  // The server cycle represents 20ms, so the agent has to execute 50 cycles
  // to run 1s.
  int totalServerCycles = agentRunTimeInSeconds * 50;

  // This loop synchronizes the agent with the server.
  for (int i = 0; i < totalServerCycles; i++) {

    //check for aborting the agent from the console (by hitting return)
    try {
      if (System.in.available() != 0)
        break;
    } catch (IOException ex) {
      java.util.logging.Logger.getLogger(Agent_SimpleSoccer.class.getName()).log(Level.SEVERE, null, ex);
    }

    sense();

    think();

    act();
  }
}
```

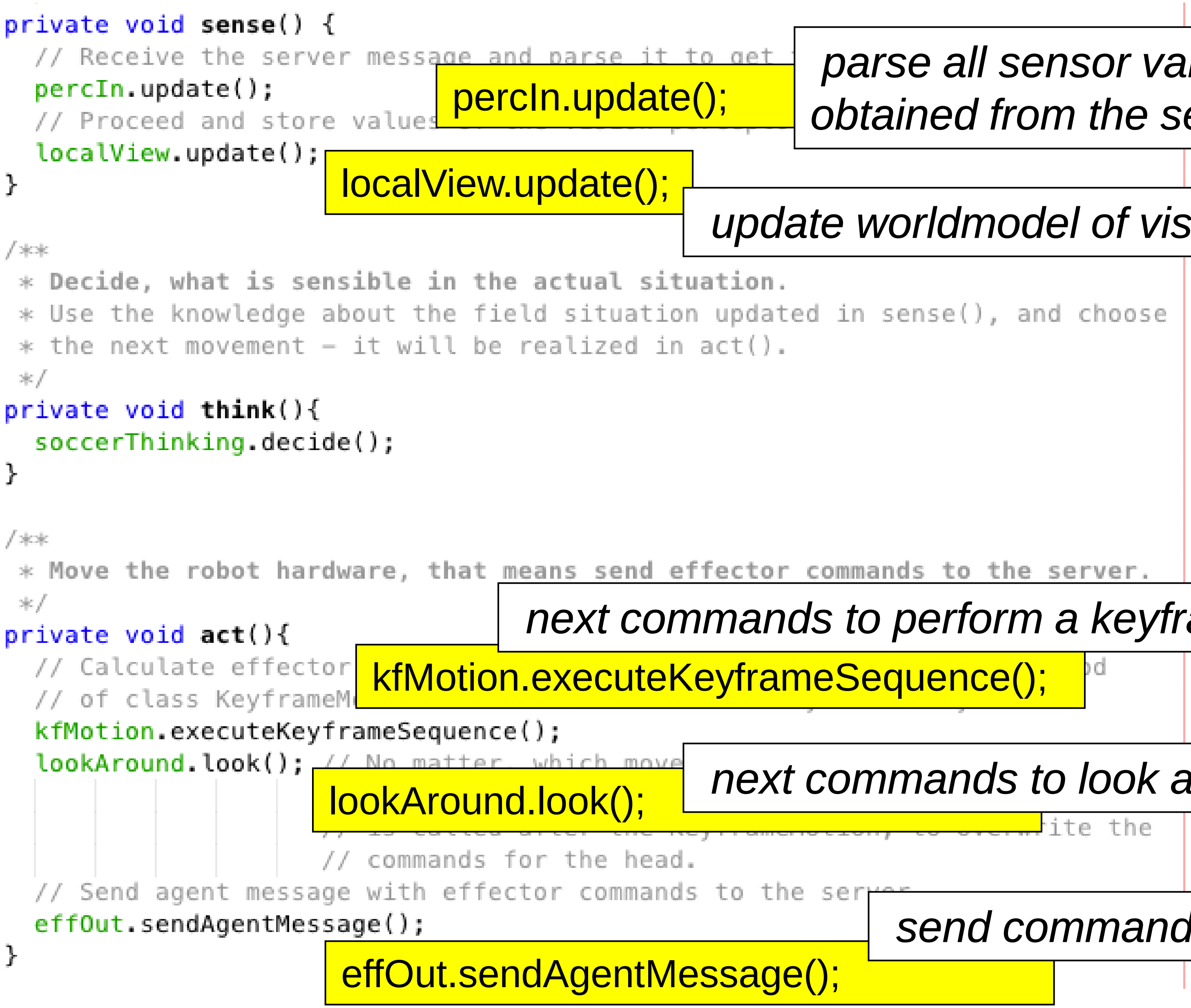
for (int i = 0; i < totalServerCycles; i++) {

sense();

think();

act();

Basic cycle in Agent_SimpleSoccer



```
private void sense() {
  // Receive the server message and parse it to get
  percIn.update();
  // Proceed and store values
  localView.update();
}

/**
 * Decide, what is sensible in the actual situation.
 * Use the knowledge about the field situation updated in sense(), and choose
 * the next movement - it will be realized in act().
 */
private void think(){
  soccerThinking.decide();
}

/**
 * Move the robot hardware, that means send effector commands to the server.
 */
private void act(){
  // Calculate effector
  // of class KeyframeM
  kfMotion.executeKeyframeSequence();
  lookAround.look(); // No matter, which move
                                                                  ite the
                      // commands for the head.
  // Send agent message with effector commands to the ser
  effOut.sendAgentMessage();
}
```

percIn.update();

parse all sensor values obtained from the server

localView.update();

update worldmodel of visual data

kfMotion.executeKeyframeSequence();

next commands to perform a keyframe motion

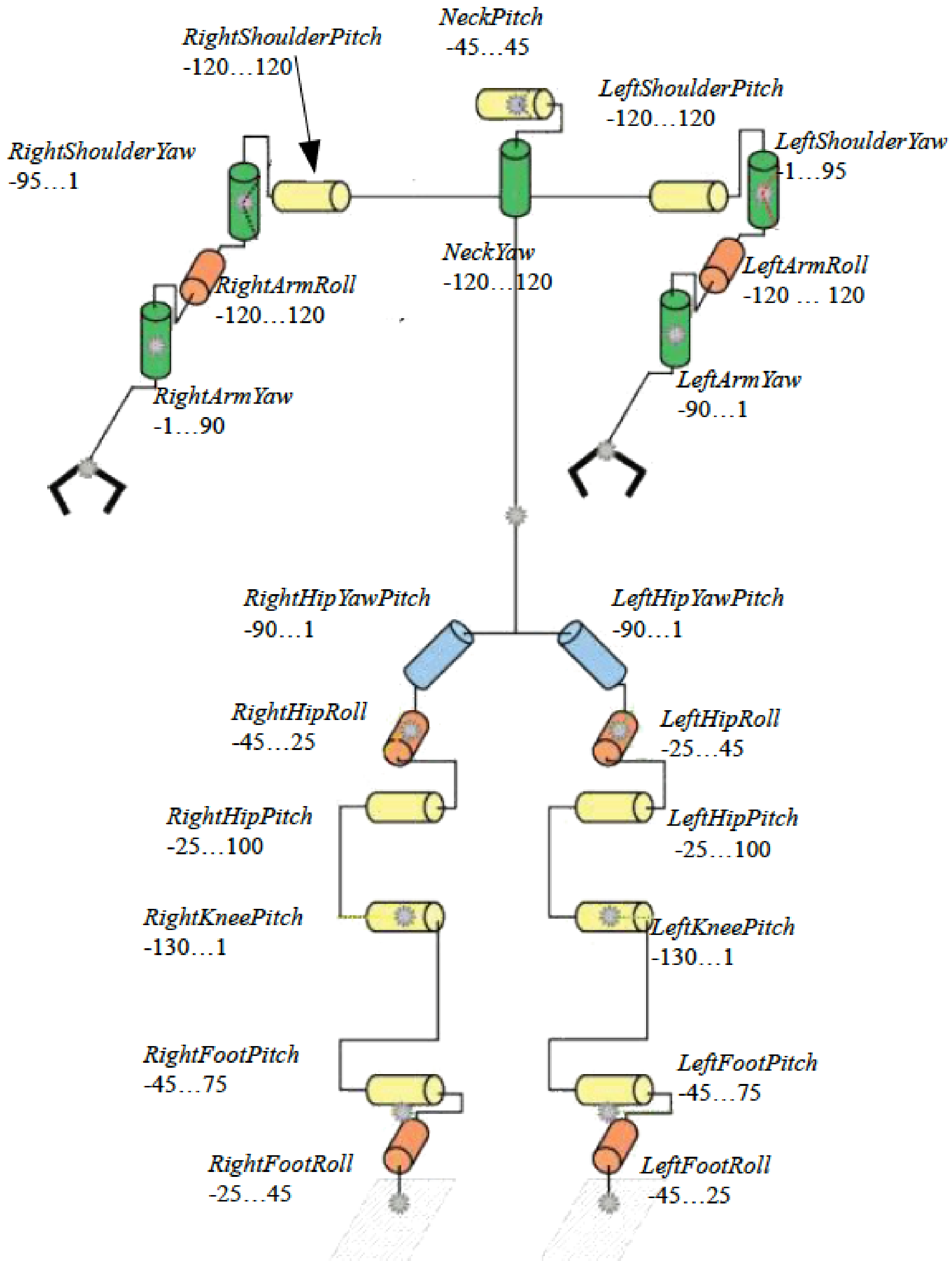
lookAround.look();

next commands to look around

effOut.sendAgentMessage();

send commands to server

This picture shows the joint names and the minimal and maximal angles they can achieve.



Motor commands

```
effOut.setJointCommand(RobotConsts.<joint-name>, <speed> );
```

<joint-name> is the name of a joint,

code completion

```
    effOut.setJointCommand(RobotConsts.     , <speed> );
```

shows all available names

<speed> sets the angular speed (radians per second)

the speed is continously maintained until a new speed is set

(hence, speed=0 must be set to stop a motion)

Motion Skill: Set of Keyframes

FILE walk_forward-flemming-nika.txt
in …/keyframes

```
300 0 -21 -62 32 -69 -59 0 -8
300 -5 -21 -62 46 -69 -59 0 0
300 0 -21 -62 60 -69 -59 0 8 -10 -0 12 -11 0 8 12 -0 -3 -11 -110 -32 69 59
300 0 -21 -75 60 -69 -59 0 8 6 -36 27 -11 0 8 12 -15 7 -11 -97 -32 69 59
300 0 -21 -86 60 -69 -59 0 8 42 -69 13 -11 0 8 12 -30 23 -11 -86 -32 69 59
300 0 -21 -110 60 -69 -59 0 8 12 -0 -9 -11 0 8 -10 -0 12 -14 -62 -32 69 59
300 -5 -21 -110 46 -69 -59 0 0 18 -0 -9 -4 0 0 -10 -0 17 -5 -62 -46 69 59
300 0 -21 -110 32 -69 -59 0 -8 12 -0 -3 11 0 -8 -10 -0 12 11 -62 -60 69 59
300 0 -21 -97 32 -69 -59 0 -8 12 -15 7 11 0 -8 6 -36 27 11 -75 -60 69 59
300 0 -21 -84 32 -69 -59 0 -8 12 -30 23 11 0 -8 42 -69 13 11 -84 -60 69 59
```

Each line starts with the transition time followed by the target angles of joints in a predefined order.

Keyframe sequences are “played” by class keyframeMotion.

Order of Joints in our Keyframes

NeckYaw = 0
NeckPitch = 1
LeftShoulderPitch =2
LeftShoulderYaw = 3
LeftArmRoll = 4
LeftArmYaw = 5
LeftHipYawPitch = 6
LeftHipRoll = 7
LeftHipPitch = 8
LeftKneePitch = 9
LeftFootPitch = 10
LeftFootRoll = 11
RightHipYawPitch = 12
RightHipRoll = 13
RightHipPitch = 14
RightKneePitch = 15
RightFootPitch = 16
RightFootRoll = 17
RightShoulderPitch = 18
RightShoulderYaw = 19
RightArmRoll = 20
RightArmYaw = 21

Development of Keyframe Motions

Develop the new motion using **MotionEditor** for creation and **agentKeyframeDeveloper** for test.

Extend the program KeyframeMotion at 3 places, e.g.:

- private static KeyframeSequence KICK_SEQUENCE;
- KICK_SEQUENCE = keyframeReader.getSequenceFromFile(„kick.txt");
- public void setKick() {… actualSequence = KICK_SEQUENCE …}

- Use the new motion by calling setKick() in your program.
  (as e.g. in Agent_SimpleSoccer)

Motion Editor

is described by

MotionEditor.pdf

Perceptors of SimSpark Soccer Simulator

- Hinge Joint Perceptors
- Vision Perceptor at the head
- Gyrometer in the torso
- Accelerometer in the torso
- Force Resistance Perceptor at the feets
- Hear Perceptor at the head
- Game State Perceptor

Positions of joints

Example:

`percIn.getJoint(RobotConsts.LeftShoulderPitch)`

returns the position of LeftShoulderPitch
in radians, can be convertd to degrees:

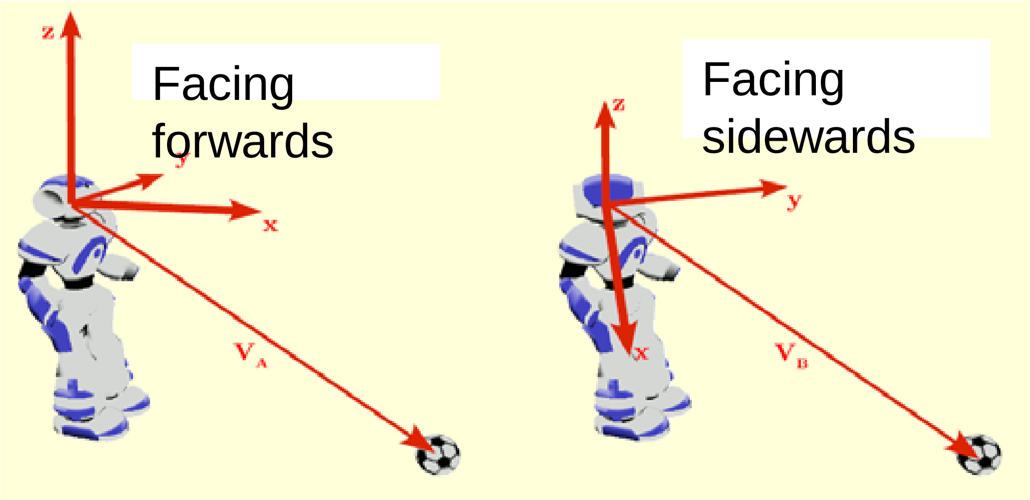
`Math.toDegrees(percIn.getJoint(RobotConsts.LeftShoulderPitch))`

Joints have same names as for motor commands.

Vision Perceptor

Information comes only each 3rd cycle, i.e. each 60 msec.
No image processing.
Simulator provides correct perceptor values:

(Polar-)Coordinates relatively to the pose of the camera
(i.e. facing direction of the robot head).

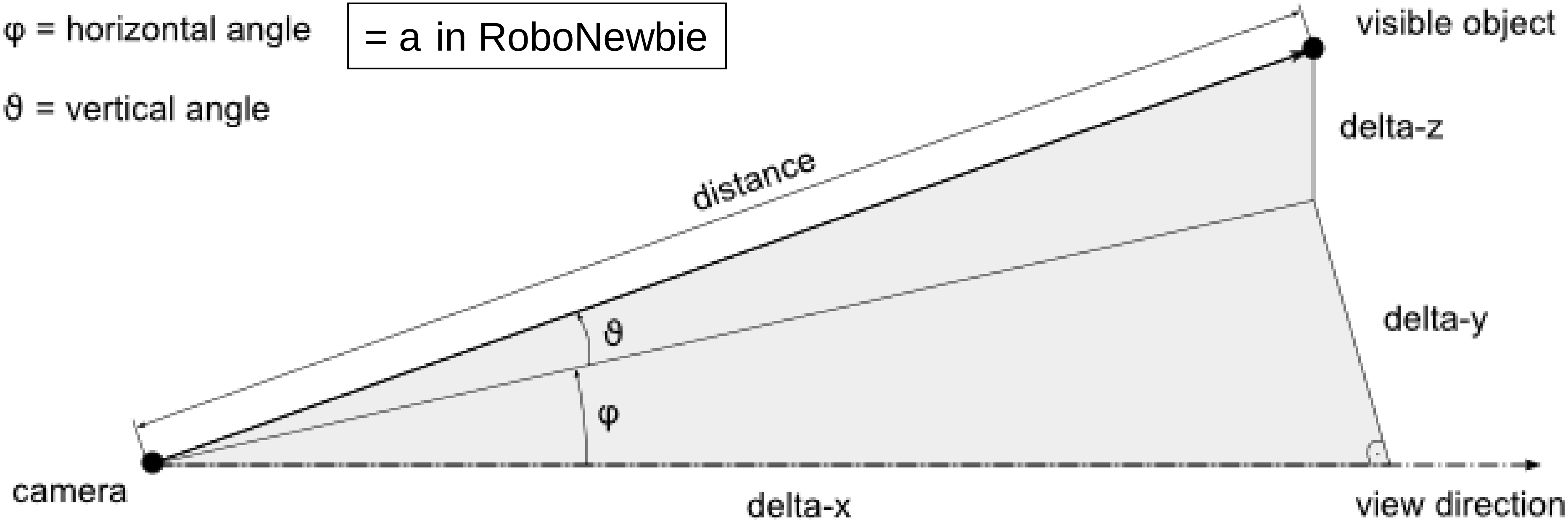


Facing forwards

Facing sidewards

View angle of camera:120 degrees horizontally and vertically

Coordinates by Vision Perceptor



= a in RoboNewbie

The server sends polar coordinates.
RoboNewbie uses Vector3D format from
org.apache.commons.math3.geometry.euclidean.threed.Vector3D
with methods for conversion and access.

Examples:

ballCoords.getAlpha()	for horizontal angle
ballCoords.getNorm()	for Distance

Visual Objects in SimSpark

Goal posts

Corner Flags

Lines

Ball

Players with

- Team name
- Player id
- Body parts
  - Head
  - Right lower arm
  - Left lower arm
  - Right foot
  - Left foot

Examples:

```
percIn.getGoalPost(FieldConsts.GoalPostID.G2L);
percIn.getBodyPart(PlayerVisionPerceptor.BodyPart.llowerarm);
```

Because Visual Perceptor comes only at each 3rd cycle, it is recommended to use LocalFieldView (to be explained later)

LookAroundMotion

LookAroundMotion moves the head (the camera) periodically:

Turns down to about 40°,
back to upright position,
then left to about 60°,
then right to about -60°
and back to initial position.

You can change this values in LookAroundMotion
(and adapt LOOK_TIME if necessary).

The period takes about 1.8 seconds, provided by

public static final double LOOK_TIME = 1.8;

Objects are perceived with coordinates relatively to camera.

LocalFieldView makes an approximative translation
to coordinates relatively for facing forwards (see below).

LocalFieldView

Maintains a ball model:

```
BallModel ball = localView.getBall();
```

It provides

- methods for coordinates:

```
Vector3D  vecBall = ball.getCoords();
vecBall.getAlpha();
vecBall.getNorm();
vecBall.getX();
```

- last time of visibility:

```
ball.getTimeStamp();
```

- actual visibility (last 3 cycles):

```
ball.isInFOVnow();
```

Calculate if ball was seen in the last lookAround period:

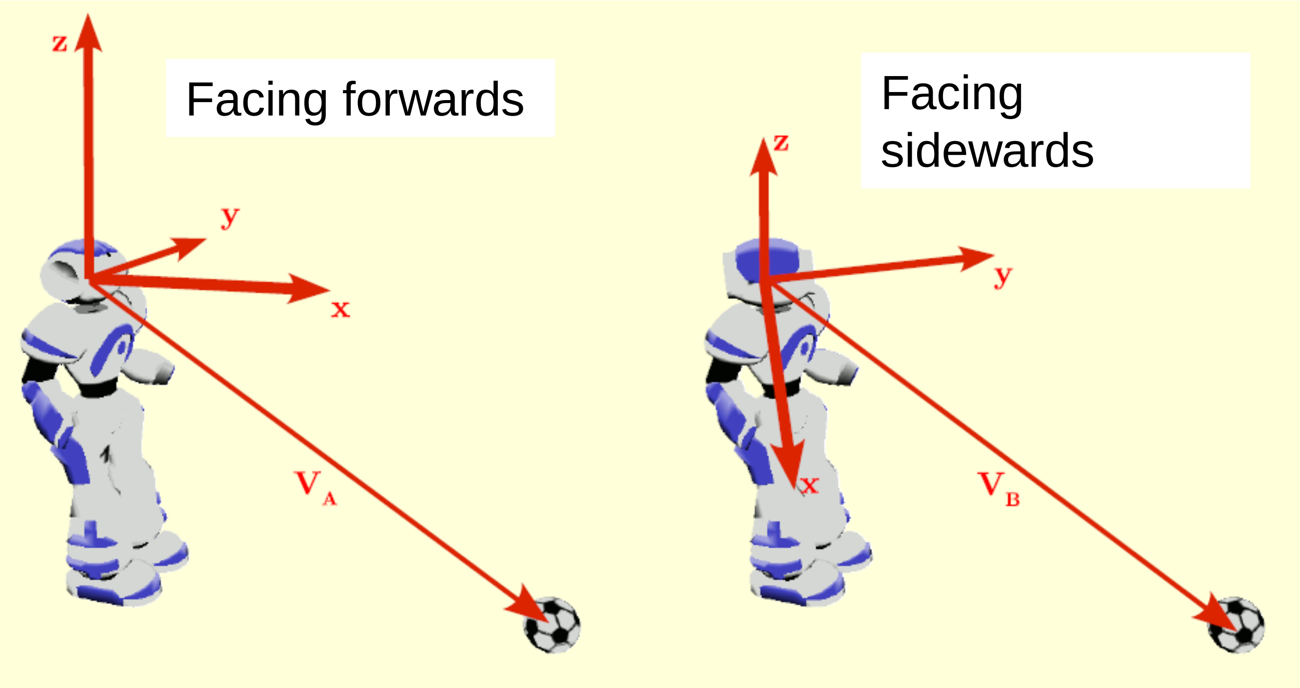
```
serverTime - ball.getTimeStamp() < lookTime;
```

Related models are maintained for other visible objects.
See agent_TestLocalFieldView for examples.

Preprocessing for Perception in LocalFieldView

LookAroundMotion moves the head (the camera) periodically as described above.

Objects perceived with different coordinates relatively to camera.



But LocalFieldView needs unique coordinates (facing forwards).

Simplification in RoboNewbie

The vision perceptor collects visual data while moving the head.

The position of an object is described by polar coordinates
(d, a, d) with distance d, horicontal angle a and vertical angle d.

Direction of the head (camera) by LookAroundMotion is:
1. in horizontal direction (yaw y) while vertical angle (pitch f) is 0.
2. in vertical direction (pitch f) while horizontal angle (yaw y) is 0.

LocalFieldView is to provide transformed data (d’, a’, d’)
according to the coordinate system when facing forward.

Simplification in RoboNewbie

The distance d remains unchanged, i.e. d’ = d,

but angles a’ and d’ need to be calculated from a, d, y, f .

Correct calculation needs related transformations.

Instead, a simple approximation is performed by RoboNewbie:

a’ and d’ are calculated using the offsets y resp. f .

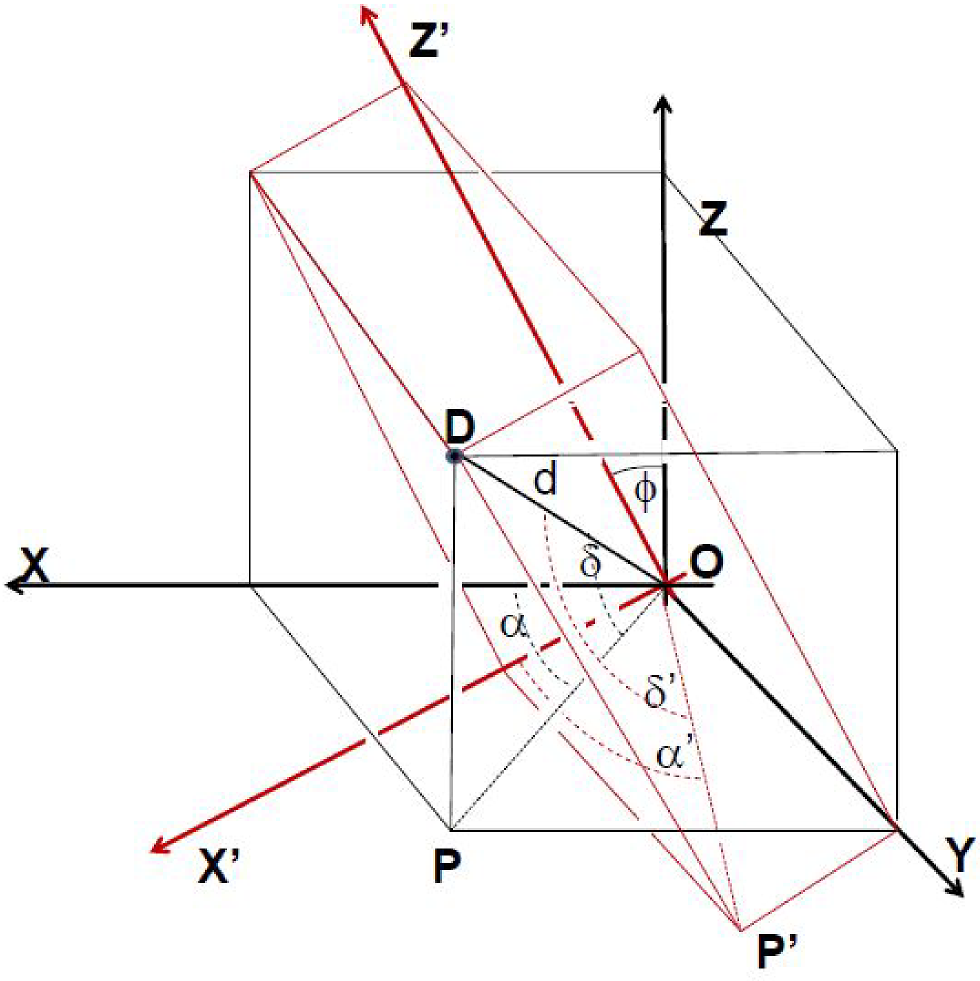
The result is correct

- for vertical angle d’ .
- for horizontal angle a’ as long as f = 0 .

It is only an approximation for angle a’ if f ≠ 0 (head tilded)

Simplification in RoboNewbie

The angles d and a of perception change according to the change from XY-plane to X’Y-plane (tilded head).



Correct transformations would need complex geometrical calculations.

Drawback of simplified calculation: Deviations of position for near objects.