Architecture and basic skills of the FC Portugal 3D simulation team

Hugo Marques, Nuno Lau, Luis Paulo Reis

Resumo -**A liga de simulação do RoboCup, iniciou em 2004 uma nova competição que utiliza um simulador de futebol em que o ambiente virtual tem 3 dimensões. Este artigo descreve a arquitectura dos agentes da equipa FC Portugal 3D que concorreram ao campeonato mundial do RoboCup 2004 em Lisboa. Serão descritas as características do novo simulador 3D e os aspectos principais da arquitectura da equipa FC Portugal e do desenvolvimento dos comportamentos básicos dos agentes. A equipa FC Portugal classificou-se em 8º lugar da competição 3D do campeonato mundial de 2004 do RoboCup.**

Abstract – **The RoboCup Simulation League introduced in 2004 a new competition based on a soccer simulator which implements a 3D virtual environment. This paper presents the architecture of the agents made by the FC Portugal 3D team in order to participate in the RoboCup 2004 World Championship competition which happened in Lisbon. We will describe the new 3D simulator and the most important characteristics of the architecture, basic behaviour and skills of the agents developed agents. The FC Portugal 3D team achieved 8th place in the RoboCup 2004 World Championship.**

I. INTRODUCTION

The first version of the 3D simulation league simulator was made available to the RoboCup community during January 2004. The proposal of a new simulator had the following objectives:

- Replace the 2D environment of previous simulator with a 3D environment
- New, more realistic, physics model
- Simulation results should not be dependent on available computational power or on the quality of network resources

The first version of the simulator was very immature. Still it allowed us to contact with some of the new models of robots, of their sensors and actuators, and some of the new features related with the physics model and synchronization of agents with simulator.

The differences between the new 3D simulator [1] and the 2D simulator [2] used in previous RoboCup competitions, and in our previous research, are very significant. These differences led us to the decision of starting to develop from scratch a new agent for the new 3D simulator. Of course, we intended to apply, with proper adaptations, most of the methodologies we had previously developed for the 2D simulator [3-5]. However the code of the new agent is almost completely new and did not result from the adaptation of our 2D agent code.

The following sections describe the architecture and some of the algorithms that are used by the new FC Portugal 3D agent.

II. SIMULATION ENVIRONMENT FOR 3D SOCCER

The simulation environment of the RoboCup 3D Simulation League is based on a client-server model. The simulator is the server and agents and visualization tools are the clients. The simulator creates the virtual environment (soccer field, markers, goals, etc...) where agents live, sends sensory information to the agents, receives their actions and applies the virtual physics model in order to resolve positions, collisions and interactions with the ball. Each team plays with 11 agents that must cooperate to score as much goals as possible while not allowing the other team to score.

The development of the 3D simulator used available open-source tools extensively. It uses the SPADES [6,7] framework for the management of agent-world communication and synchronization, ODE [8] for the physical model, expat [9] for XML processing, Ruby [10] for scripting language support and boost [11] for several utilities.

A. SPADES

The 3D simulation server is implemented above a platform called SPADES (System for Parallel Agent Discrete Agent Simulation) [6]. SPADES is a middleware system for agent-based distributed simulation. It aims to provide a generic platform to run in multi-computer systems. It implements the basic structure to allow the interaction between agents and a simulated world so that the users do not have to worry about communication and synchronization mechanisms such as sockets, addresses, etc.

SPADES' main features are:

- Agent based execution - support to implement sensations, thinking and actions.
- Distributed processing – support to run the agents applications on many computers.
- Results unaffected by network delays or load variations among the machines – SPADES ensure

that the events are processed in the appropriate order.

- Agents can be programmed independently from the programming language – the agents can be programmed in any language once it provides methods to write/read to/from Pipes.
- Actions do not need to be synchronized in the domain – the actions of the agents can take effect at varying times during the simulation.

A.1 Components organization

SPADES components are organized in a client-server architecture (Fig. 1). The Simulation Engine and the Communication Server are provided by SPADES; while the Agents and the World Model are built by the user and run upon the formers.

The Simulation Engine is a generic piece of software that provides abstractions to create specific world models upon it. Agents may run in the same computer of the Simulation Engine on in remote computers linked to the network, in this case a Communication Server must be running in the remote computer. The World Model module must be running in the same computer of the Simulation Engine. This module specifies the characteristics of the environment where the agent will live.

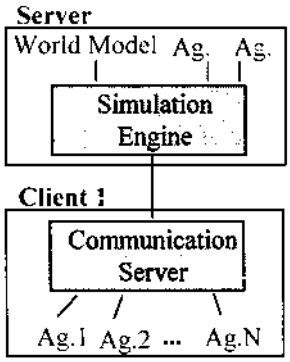

Fig. 1 – SPADES Components diagram.

A.2 Sense-Think-Act Cycle

SPADES implements what it calls the sense-think-act cycle in which each agent receives sensations and replies with actions. That means that an agent is only able to react after receiving a sensation message. The agent is also capable of requesting its own sensations, but the principle remains - a sensation must always precede an action. In order to allow actions between "normal" sensations, SPADES provides an action called *request time notify* that returns an empty sensation and after receiving it the agent is able to respond with actions. For example, if an agent received a sensation at cycle 100 and wants to produce an action at cycle 110, and if the next sensation will only arrive at cycle 120, the agent can ask to receive a *time notify message* at cycle 110 and just reply with the desired action after receiving it.

Fig. 2 depicts the sense-think-act cycle and the time where each of its components run. From A to B a sensation is sent to the agent. After receiving the sensation (from B to C) the agent decides which actions will be executed; then from (C to D) the actions are sent do the server.

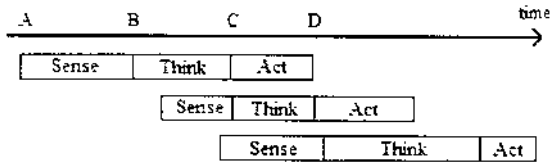

Fig. 2 – SPADES Sense-Think-Act Cycle.

In many agents, the sense, think and act components may be overlapped in time (like in Fig. 2). There is just one restriction: The thinking cycles for one agent cannot be overlapped. This constraint makes sense, since just a single processing unit is used per agent, and thus, just one sensation at time can be processed.

B. SIMULATION SERVER

As stated before the simulator runs upon the SPADES, and uses ODE to calculate the physical interactions between the objects of the world. The graphical interface is implemented using OpenGL.

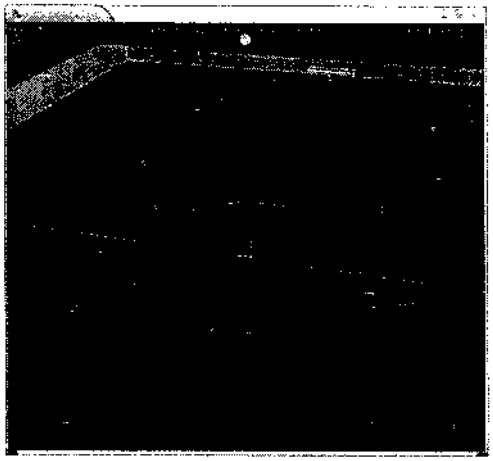

Fig. 3 – Snapshot of the 3D Simulator.

The 3D simulation server (Fig. 3) [1] allows twenty two agents (eleven from each team) to interact with the server in order to play a simulated robotic soccer game. Each agent receives sensations about the relative position of the other players and field goals and other information concerned with the game state and conditions. At this time the information about the positioning of the objects in the world is given by an awkward omnivision that allows the agent to receive visual information in 360 degrees. The agents have the shape of a sphere. Replying each sensation an agent sends actions like *drive* or *kick*. Driving implies applying a force on the body with a given direction and kicking implies applying a force on the ball radially to the agent. Each sensation is received on every 20 cycles of the server and each cycle takes 10 ms.

B.1 Sensations

There are already several sensations that an agent is able to receive. Every sensation starts with the character 'S' followed by two integers (*time*). The first is the cycle in which the message was sent and the second is the cycle in which the message arrived to the agent. A sensation message has the format:

```
Stime time data
```

where *data* is the string with the information related with the sensation itself[1].

Vision. The vision sensation gives the agent the spatial arrangement of the world objects in the field. World objects are the players, the ball, the goals and flags in the corners. The vision is, on the 0.3d version, omni-directional and the objects are considered transparent (at least for the vision sense). The position of the objects is given in polar coordinates relative to the respective agent. The coordinates are given by the distance, the horizontal angle – theta – and the elevation angle – phi.

```
Stime time (Vision
  (Flag  (id id) (pol d theta phi)) …
  (Goal (id id) (pol d theta phi)) …
  (Ball (pol d theta phi))
  (teamname (id id) (pol d theta phi)) …
          )
```

GameState. The game state sensation gives the agent all the information concerned with the game properties. It gives information about the dimensions of the field, the goals, the ball and the agents. It also gives information about the mass of the objects in the world and other aspects of the game like: time, play mode agent number, and if the agent's team is the right or the left one. Here is the format of the given information:

```
Stime time (GameState
             (team side)
             (unum number)
             (FieldLength length)
             (BallMass mass)
             (playmode playmode)
              …)
```

AgentState. The agent state gives the agent information about its internal state. For now, just information regarding the battery condition and the temperature are given.

```
Stime time (AgentState
             (battery battery)
             (temp temp)
            )
```

[1] Note that one sensation message can have more than one sensation.

B.2 Actions

As well as sensations, several actions are implemented that allow an agent to interact with the world. Every action message starts with the 'A' character and it is followed by a string,

```
Adata
```

where *data* contains the information about the action itself.

Create. The first action that an agent must send is the create action. This action allows the server to register an agent and thus establish the communication with it. The action create as the form:

```
A(create)
```

Init. The init message allows the server to receive essential information about the agent, namely its number and its team. If the number is passed as 0, the server automatically attributes a number to the agent. The init action has the format:

```
A(init (unum number) (teamname name))
```

Beam. The beam action allows an agent to move to a given point. It may only be executed before game kickoff and it does not obey to any physical law. Its structure is the following:

```
A(beam x y z)
```

Drive. The drive action allows an agent to move. It applies a force vector (x, y, z) to the center of the agent's body with the maximum length of 100.0 units. It has the format:

```
A(drive x y z)
```

Kick. The kick action follows the laws of gravity and movement of physics and allows an agent to kick the ball with a given force intensity and a given vertical direction (see *3.4 Physics – Kick*). The vertical direction is passed as an argument - the elevation angle. The horizontal direction is radial to the agent's body. The action must be sent like:

```
A(kick angle force)
```

B.3 Other important communication procedures

The server establishes the communication by sending a *done* ('D') message (Riley *et al.* 2003). When the agent receives this message it should execute its initialization procedures and when it finishes them it must send an *initdone* ('I') message. After that the server starts to send

sensations and the agent replying with actions. Every set of actions must finish with a *done* ('D') message (Fig. 4).

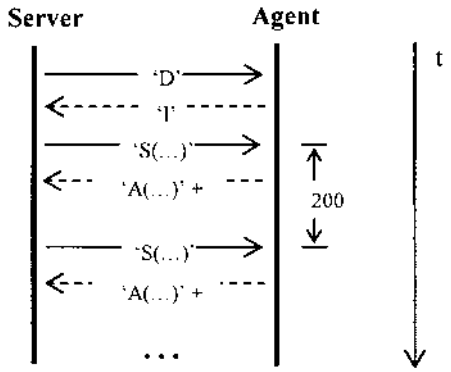

Fig. 4 – Temporal diagram of the communication between the agent and the server.

A sensation is received in every 20 server cycles, which means that an agent should only be able to execute actions within 20 cycles intervals. However, as stated before, an agent can ask the server to receive a sensation in a given cycle by sending a *request time notify* (R) message. The format of the message is the following:

```
Rtime
```

where *time* is the time at which the server must reply. This procedure makes the server reply with an empty sensation ('T') at the cycle given. As stated before the only reason to receive an empty sensation is to be able to act in a given time between two sensations.

An example where the *request time notify* makes sense is when an agent wants to kick the ball in given position. In the 3D server simulator, to kick the ball in a given direction, an agent must place itself quite accurately. Thus, because the interval between sensations is too long (200ms), it can happen that an agent that is running to the kicking point at cycle t is before that point and at cycle $t + 1$ has already passed the point. To surpass this, one can predict the time that the agent arrives to the desired position and ask to receive a time notify message at that time in order to be able to kick at the right moment.

By receiving this sensation the agent is able to respond with action messages.

Each sensation takes 10 cycles to reach the agent (send delay) and actions sent by the agent take 10 cycles to reach the simulator. Hence a sent action starts to take effect at the time the next sensation is sent by the server as it is shown in Fig. 5.

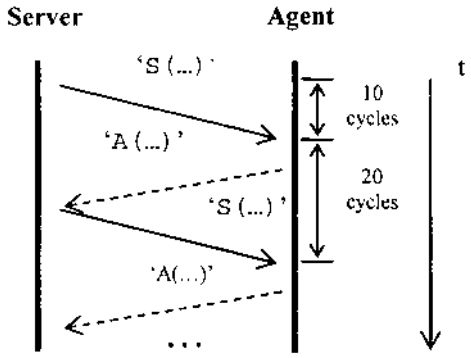

Fig. 5 – Communication delays.

B.4 Physics

The physical interactions of the game are made in a discrete way that is, in every cycle the new forces to be applied to the bodies, their current positions, velocities, etc. are calculated. Every cycle is simulated to take approximately 10 ms

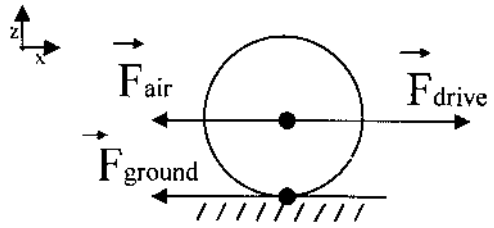

Fig. 6 – Forces applied to the body of an agent when the drive effector is used

During a regular drive, the body of an agent is under the influence of three forces: the drive force, the drag force caused by the contact of its body with the ground and the force caused by the friction of its body with the air. Additionally a drag torque is also applied. The ground force is such that the robot rotates without sliding over the field. The drag force and drag torque are proportional to the robot's speed. The drive force is controlled by the agent through the *drive* action.

III. FCPORTUGAL 3D AGENT

The FC Portugal 3D was developed from scratch. Any attempt to use the code of the FC Portugal 2D agents would run into serious problems, not because of the addition of an extra coordinate, but because of the huge difference in the 2D and 3D servers functioning.

A. Agent's Architecture

The agent structure includes six main modules/packages that cover different parts of its functioning (Fig. 7).

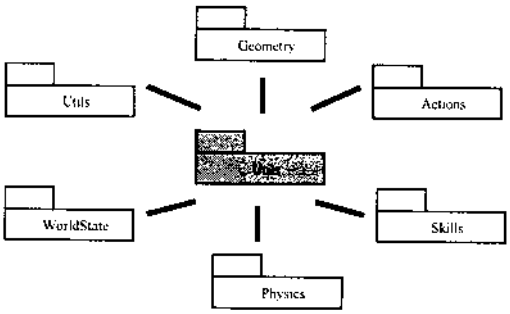

Fig. 7 – Modular division of FC Portugal 3D Basic Agent.

World State. The "World State" package (Fig. 8) is probably the most complex one. It has all the information that the FCP Agent needs to decide which action it should take. There are three kinds of information that the WorldState needs: information about the objects (like players, landmarks and the ball), information about the conditions of the game (like field length, goal width, etc.) and the state of the game (like the current play mode, the result, time, etc).

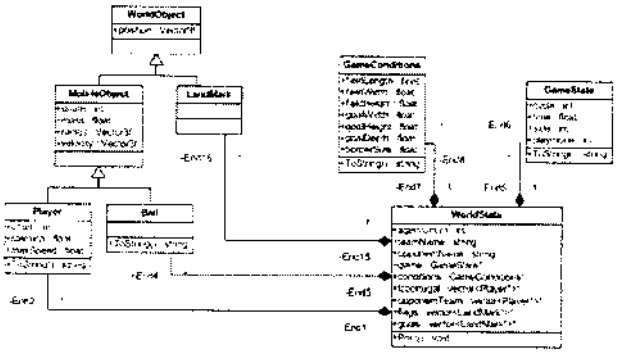

Fig. 8 – FC Portugal World State package architecture.

Physics The physics package aims to reproduce the physical interactions between the bodies in the world using the same model as the server does. At the present one can estimate the velocity and the acceleration of an object, the current forces applied in a given body, the breaking distance and the time that an agent takes to move from one position to another applying a given force.

Geometry. The "Geometry" module is used to make easier the execution of geometrical calculations. It is used to compute data concerning distances, vectors, etc.

Skills. The skills are the low-level actions that an agent is able to perform. Kicking the ball, moving its body, intercepting the ball, or dribbling are examples of agent's skills. These are also the ones implemented at the moment by FC Portugal team. The scheme of the skills architecture is in Fig. 9.

Every skill implements the *GenericSkill* interface. When a skill is initialized it immediately computes the necessary calculations to execute itself. However, the initialization does not execute the skill. Every skill has a method named *Execute()* that allows its execution.

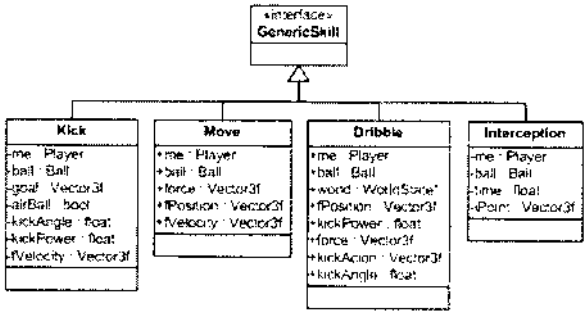

Fig. 9 – Skills' architecture

Actions. An action is a group of skills that, together, produce higher lever behaviours. Sample of actions may be: passing, shooting, making forwards, etc. Not all of the implemented actions are used in the current version of the FC Portugal team. However, the architecture of the FCP Agent supports the implementation of passes, shoots and forwards.

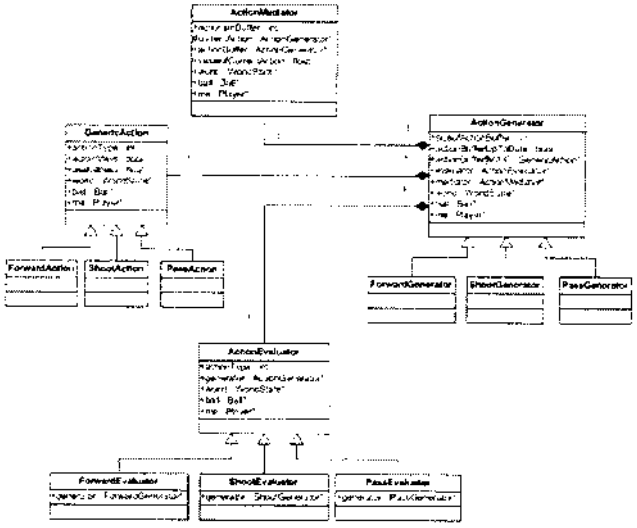

Fig. 10 – Actions Architecture.

In order to determine which action should be executed, four classes are involved: a mediator, an evaluator, a generator and the action itself. The generator (*ActionGenerator*) is the class that allows the creation of potential actions that are able be performed. There are 3 classes that extend the *ActionGenerator*, one per type of action – pass, shoot and forward. Each class is able to return a set of actions of its type, adjusted to the current situation, and that should be considered for future evaluation. The actions returned by each generator have their own properties according with its type and all of them extend the *GenericAction* class. The evaluation of the actions is done by the evaluator (*ActionEvaluator*). This is a class that enables the agent to estimate the usefulness of every generated action. The evaluator has also 3 classes (one per type of action) which extend the *ActionEvaluator* class; each of them has its one evaluation components that allow them to estimate the usefulness of every action of its type.

To join everything together the FCP Agent has a mediator (*ActionMediator*) which is a class that is able to call the necessary functions to generate and evaluate every type of action and to decide which will be the next action to be performed.

Utils. The package *Utils* was made to contain classes that do not have a direct relevance on the agent behaviour but help to make some tasks easier. Examples of the operations of these classes are the creation of log files, communication with the simulator, a message parser and a message composer to send the actions to the simulator.

B. Localisation system

The agent gets the objects position by the vision perceptor, which gives the relative position of all objects in the world in polar coordinates. The absolute position of the landmarks is set at the time the agent receives the field dimensions and the goals position. This information is usually sent by the server in the second or third sensation.

The localization algorithm is quite straightforward. The agent starts to seek the closer landmark. If that landmark is closer than 20.0m the agent determines its position by the absolute position of that landmark and its relative position to the agent. If the landmark is farther than 20.0m the agent combines, using a simple average, the position of the closer landmark with the position of the second closer one to determine its position.

Several experiments were conducted by moving the agent inside the pitch and determining its position using the algorithm described above. During these experiments the maximum error of the algorithm was around 20.0cm and the frequency of errors of this magnitude was very low.

C. Physics

The agent should be able to predict the future state of the world if he decides to act in a certain way. This knowledge is essential for making the right decisions. In order to accomplish this functionality, and as there is no documentation on these matters for the 3D simulator, several experiments were conducted to infer which is the physics model of the simulator. The results of these experiments are presented in this section.

To be able to capture the effect of a given driving force the agent must know the magnitude of the friction force and its own velocity. Two distinct methods were used to obtain these informations.

To calculate the force in the agent caused by the friction with the air quadratic regression was used. It is given by:

$$\vec{F}air = A * \vec{v}^2 + B * \vec{v} + C$$

where A = -0.84, B = 179.9 and C = 112.3.

The graphic in Fig. 11 shows the performance of the formula used during a two step movement – acceleration using maximum force in the positive direction of the x axis followed by an acceleration also using maximum force in the opposite direction. One can see that in the first part of the movement (accelerating on the direction of the x axis) the approximation used is very good since the prediction of the agent is very near the server data. However, the second part, immediately after the agent starts to brake, one can see that there is a big difference between the server data and the agent's approximation, meaning that the agent's calculation is clearly not good enough.

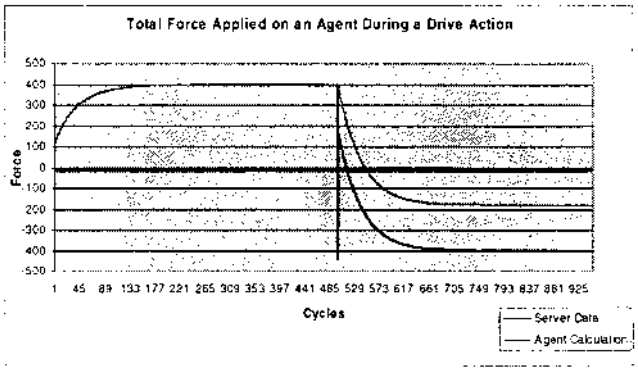

Fig. 11 – Graphic with the forces applied in the agent's body during a drive action

The player velocity was estimated based on the previous three positions of the agent assuming that acceleration was constant during that period. Analytically this provided the following equation for the velocity and acceleration:

$$\begin{cases} v_0 = \dfrac{4p_1 - 3p_0 - p_2}{2t} \\ a = \dfrac{p_2 - p_0 - 2v_0 t}{2t^2} \end{cases}$$

This methodology has been tested and Fig. 12 shows an example of the incurred errors.

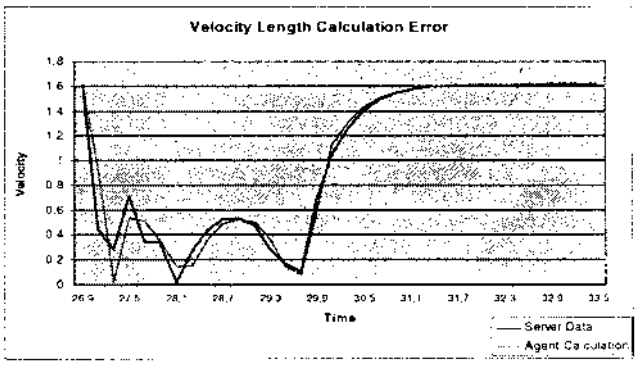

Fig. 12 – Error on the calculation of the length of the velocity vector between time 26.9s and 33.5s

To simulate the ball movement in the ground the laws of physics were applied to the situation of a free running ball with a drag force. This has led to the following velocity and position formulas:

$$v = e^{-\frac{K}{m^2}t} \times v_0$$

$$x = -2e^{-\frac{K}{m^2}t} \times v0 + 2 \times v0 + x0$$

Where x0 and v0 are the initial position and velocity of the ball. Fig. 13 shows that the results obtained are very close to the simulator's calculations. One cannot distinguish between the real values and the approximation made.

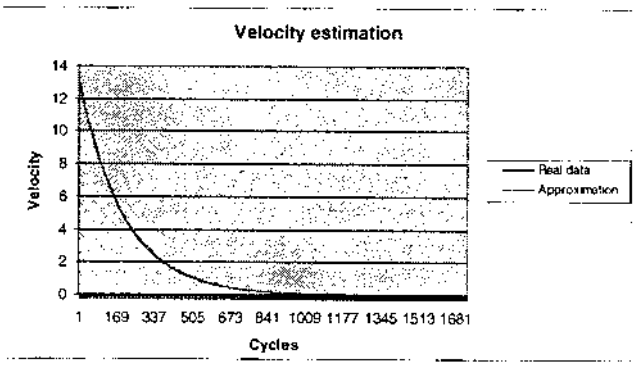

Fig. 13 – Graph showing the velocity difference between the server values and the approximation values given by the formula used

D. Skills

Move; The move skill takes three arguments: a pointer to the world object, a vector containing the point to where the agent wants to move and the velocity that it wants to arrive there. To calculate the force that should be applied at a given moment in order to move, the agent starts by getting the braking distance to reach a given velocity, given its own current position and velocity. Then it can estimate the distance that it can accelerate (*distanceToPoint – brakeDistance*). If that distance is made in more than forty cycles it accelerates, otherwise it brakes[2].

Kick; As already stated a kick action takes two arguments – the kick power and the kick angle. The FC Portugal kick skill receives four arguments: a pointer to the world object, a vector giving the point to where the ball should be kicked, a boolean informing if maximum power should be used and another boolean informing if the ball should be kicked by the air.

The power that an agent applies when kicking, is directly proportional to the distance to the final point of the kick plus an extra force. The extra force varies depending on whether the agent wants to kick the ball by air or not.

Dribble; Dribbling is the skill that allows an agent to run with the ball near its body. This is achieved by giving small kicks to the ball and running in order to catch it again. It starts by moving the agent to a position which enables it to kick the ball in the goal point (*finalPosition*) direction. Arriving there the agent tries to kick the ball as farther as possible (*maxDistance*). If there is an opponent agent that is able to steal the ball, the FCPagent starts to reduce the kick distance successively until none of the opponent players can catch the ball first.

Interception; The interception skill gives the agent the ability to catch the ball. At the moment just the quicker interception is calculated, that is, the interception that enables the agent to catch the ball in the less time possible. It receives, as arguments, a pointer to the world and the number and the team of the player to whom the interception will be calculated. The algorithm is the following: it calculates the position of the ball in each 20 cycles (interval between sensations), calculates the distance the player is able to run at maximum speed and the distance from the player to the ball. The agent is able to intercept the ball when the ball distance is smaller than the distance that is able to run in a given time.

E. Agent behaviour

```
Case (Playmode) equals
{
   BeforeKickOff:
      MoveAccordingSBSP();

   FCPortugalKickOff:
   {
      If (nyNumber == 9)
         RunToTheBallAndKickIt();
      Else
         MoveAccordingSBSP();
   }

   OpponentKickOff or OpponentKickIn or
   OpponentCornerKick or OpponentGoalKick:
   {
      MoveAccordingSBSP();
   }

   FCPortugalKickIn or FCPortugalCornerKick
   or FCPortugalGoalKick or PlayOn:
   {
      If (MyMoveToTheBallAccordingSBSP())
         RunToTheBallAndKickIt();
      Else
         MoveAccordingSBSP();
   }
}
```

The agent behaviour is very much tied with the SBPS [3] originally developed by FC Portugal 2D team. The SBSP, was successfully used by FC Portugal team on the 2D simulator and consists in assigning a strategic position to each agent on the field given the position of the ball and the current situation. The player that runs to the ball is the one that has the best interception from its strategic position to the ball. Each agent is differentiated by its number [4].

Before the game starts each agent requests the position that it must occupy on the playing field given its number.

[2] 40 cycles = 20 cycles to start the action + 20 cycles to execute the action. The position where the agent will be at the time the action is executed must be estimated by the agent.

The SBSP has the advantage to make easier for an agent to know where its team mates are without having to calculate their real position. This is so, because each agent is able to know the position of its companions simply by running the SBSP algorithm for all the players of its team.

If the kick-off is assigned to the opposing team the FC Portugal team places itself according with the SBSP algorithm until the ball is touched by one of the opposing players. If the kick-off is assigned to the FC Portugal team, the agent with the closest distance from the position given by the SBSP to the ball, starts running to a position that allows it to kick the ball. Arriving there it kicks the ball. The other players position themselves according to SBSP until the former touches the ball.

After the ball is touched by one of the agents of the team which has the kick-off the game state is changed to "PlayOn" and the game starts. At this state each player moves to its strategic position by running the SBPS algorithm. The only exception is the player with the best interception time, this player tries to catch the ball and to kick it. The position to where the ball is kicked depends on where the ball is situated in the field. If the ball is near FC Portugal goal then the players try to kick it to the sides, in order to avoid putting it in a frontal area near the goal (which would be very dangerous in case an opponent was present). Otherwise FC Portugal agents try to shoot in the opponent's goal direction.

IV. CONCLUSIONS

The FC Portugal 3D team participated in RoboCup 2004 achieving the 8th place. It participated in 9 games having 6 defeats, 2 draws and 1 win. The low level skills of FCP agents performed differently on the competition computers and in the tests that had been previously performed. Although the team was able to reach the ball quite fast, most of the time the agent was not capable of kicking the ball at all, the time spent in positioning itself properly was so long that the opponents could arrive and steal the ball. Unfortunately, it was not possible to perform tests in the competition computers and because the problem did not occur in the computers used in our tests the source of the problem could not be found in effective time.

V. FUTURE WORK

A new world state update and move low-level skills must be developed so that FC Portugal agents can kick the ball confidently. Without this functionality it is impossible to play soccer efficiently. From this point on the FC Portugal team wishes to implement more ideas, which have proved to be successful on the 2D Simulation League, in the 3D Simulation League, namely, the SBSP, the Low Level Skills evaluators, and others like the use of intelligent communication.

ACKNOWLEDGEMENTS

This research is supported by FCT-POSI/ROBO/43910/2002 Project – "FC Portugal New Coordination Methodologies applied to the Simulation League".

REFERENCES

[1] RoboCup Soccer Server 3D Maintenance Group, "The RoboCup 3D Soccer Simulator" http://sserver.sourceforge.net/ 2003

[2] Chen, Mao et al. "RoboCup Soccer Server", http://sserver.sourceforge.net/, 2003

[3] L.P. Reis, N. Lau, E.C. Oliveira "Situation Based Strategic Positioning for Coordinating a Team of Homogeneous Agents", In: Balancing Reactivity and Social Deliberation in Multi-Agent Systems, Markus Hannebauer, Jan Wendler, Enrico Pagello, editors, LNCS 2103, pp. 175-197, Springer Verlag, Berlin, 2001

[4] L.P. Reis and N. Lau "FC Portugal Team Description: RoboCup 2000 Simulation League Champion". In: RoboCup-2000: Robot Soccer World Cup IV, Peter Stone, Tucker Balch and Gerhard Kraetzschmar, editors, LNAI 2019, pp. 29-40, Springer Verlag, Berlim., 2001

[5] L.P Reis and N. Lau, "COACH UNILANG - A Standard Language for Coaching a (Robo)Soccer Team", In: RoboCup-2001: Robot Soccer World Cup V, Andreas Birk, Silvia Coradeshi, Satoshi Tadokoro editors, LNAI 2377, pp. 183-192, Springer Verlag, Berlim 2002

[6] Patrick Riley, "SPADES: System for Parallel Agent Discrete Event Simulation", AI Magazine, 24(2):41–42, 2003

[7] Patrick Riley, "SPADES for Parallel Agent Discrete Event Simulation", http://spades-sim.sourceforge.net/

[8] Smith, Russell. "Open Dynamics Engine v0.039 User Guide", http://opende .sourceforge.net/, 2003

[9] The Expat XML parser, http://expat.sourceforge.net, 2004

[10] Ruby Home Page, http://www.ruby-lang.org, 2004

[11] Boost C++ Libraries, http://www.boost.org, 2004