Bahia3D - A Team of 3D Simulation for Robocup[⋆]

Alan Santos, Ayran Cruz, Diego Frias, Emmanuel Argollo, Josemar R. Souza, Lorena Pereira, Marco A. C. Simões, and Murilo Alves

Bahia State University (ACSO/UNEB), Salvador, BA, Brazil
{alandossantossoaress, ayranccruz, diegofriass, enmanueru3553, lorena.santpe}@gmail.com, {msimoes,josemar}@uneb.br

1 Introduction

Team Bahia3D has been developed since 2008 with participations in RoboCup 2009, 2010 and 2011. After RoboCup 2011, the entire agent's architecture has been redesigned. This paper describes the second version of team Bahia3D with its new architecture and its current development state. We also describe the future work.

Using this new version on its earliest stage the team won the runner-up in Latin American RoboCup Open 2010 and the Third Place in Latin American RoboCup Open 2012. In this last competition the team has reached a more mature stage. It was the second best defense in the competition and has showed the best goalkeeper and kick skills.

Based on this recent performance, we believe that it is time to retest the team in the worldwide competition. In the following sections we describe the agent architecture and the modules we have already developed such as world model, movement manager, and artificial intelligence. We also describe the work in progress and future work.

2 Architecture

The team Bahia3D is based on a modularized agent architecture which has its complexity concentrated on two aspects: implementation of basic movements and implementing reasoning rules[1]. Fig. 1 illustrates the agent architecture as described below.

The Sensor and Actuator enable communication between server and agent.The PKS (Perception Kinematic State) stores information about the kinematic state of the agent, the ball and the other agents present in the game. This state is divided on visual information and sensory intelligence (SI). The visual information provides the exact location of all objects including the agent and possible evolution of current states. The SI generate greater information storage of the agent about itself.

⋆ This project is partially funded by UNEB, FAPESB and CNPQ

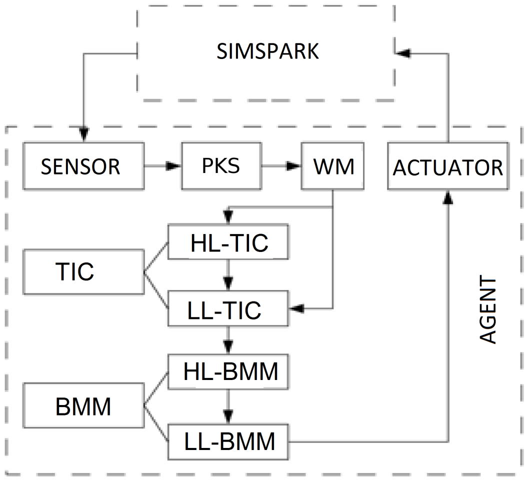


Fig. 1. Bahia3D agent architecture.

The WM (World Model) was built using the singleton standard. It works based on two data types: i) the raw data, representing information in the exact form it was received from the server; ii) calculated data resulting from calculations performed on raw data providing greater semantic information to the agent.

TIC (Tactical Intelligence Center)decides which action the agent should perform according to its role in the game (defender, forward, goalkeeper) and environment information from the WM. HL-TIC (High Level TIC) analyses the information and decides which action to take. LL-TIC (Low Level TIC) defines the best way to perform the action decided by HL-TIC. It defines the movement sequence that best describes the action using time and precision as parameters.

The BMM (Body Movements Manager) is responsible for execution of motion scripts. Each movement in the sequence defined by LL-TIC has a corresponding motion script. Scripts are divided into blocks and blocks divided into poses. The HL-BMM (High Level BMM) receives a LL-TIC movement and turns it into a sequence of blocks. The LL-BMM (Low Level BMM) receives and executes this sequence of blocks turning it into a sequence of poses that matches the chosen movement. It may take one or more server cycles to execute one movement.

This modular approach can be repeated internally enhancing the agent control level. For example, a strategy module can be created above the TIC to define high-level AI strategies. This architecture is very flexible and extensible. It can be very useful as a base for newbie teams to start their agent projects.

3 The World Model (WM)

The WM is an abstraction of the real environment which contains the agents and objects of a football match and which is used in decision taking to perform an action according to the state represented by WM. The world model stores all relevant information about the field, ball and agents.

The WM is a class that implements the singleton pattern. This standard ensures that there is only one instance of a class, maintaining a global point of access to its object. It guarantees that all parts of agent code use the same information avoiding possible inconsistencies. The WM instance can be accessed anywhere in the code using the *getInstance()* method that returns a reference to the WM.

The world model is composed of several attributes as playing time, agent state that indicates whether the agent is fallen, lifting, standing or walking. The attribute *mRunningAction* stores the current agent action. Fig. 2 illustrates the class diagram of the WM.

The class *Field* stores the dimensions of the field and lines positions. The class *Ball* contains information about position, status, speed and direction of the ball.

There is a class called *Me* that represents the agent and all relevant information about itself. The main properties are the names of the joints, movement limits of the joints and forces acting on the agent.

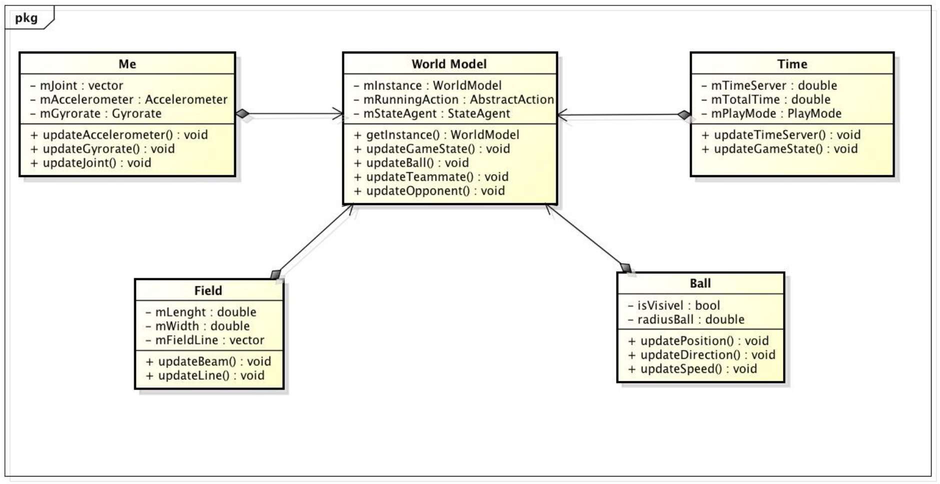


Fig. 2. Class Diagram of the World Model.

The WM is updated by the *update* methods that are called every cycle. The main methods are *updateGameState()*, *updateBall()*, *updateTeammate()* and *updateOpponent()*.

4 Body Movements Manager(BMM)

The BMM controls robot's movements using scripts. It solved problems in execution of movements like freezing and shakes in previous versions of Bahia3D agent.BMM is divided in two levels: high level, which uses the block concept, and low level, which uses the pose concept.

4.1 HL-BMM: High Level

The high level is an abstraction of the set point in classic control theory. A block is a sequence of poses, that always finish in a stable position. Using blocks, allows the intelligence to stop movements when needed, and create composite movements. For example: walking is a movement that can be factored into steps: initial step, regular step and final step, each type of step is a block. A reader loads from XML scripts the poses of each block, and creates templates to be used by HL-BMM to make a movement in execution time. For example, using the 3 types of step for each leg(6 blocks) and providing the number of steps the needed blocks is calculated using equations (1) and (2).

$$if N_{step} >= 3\, is\, odd$$

$$M_{walk}(N_{step}) = [B_{pi-pe} + (\frac{N_{step}-3}{2})B_{pr-pd} + B_{pr-pe} + B_{pr-pd} + B_{pf-pe}] \tag{1}$$

$$if N_{step} >= 2\, is\, even$$

$$M_{walk}(N_{step}) = [B_{pi-pe} + (\frac{N_{step}-2}{2})B_{pr-pd} + B_{pr-pe} + B_{pf-pd}] \tag{2}$$

The structure can be reflected by the simplified diagram in Fig. 3.

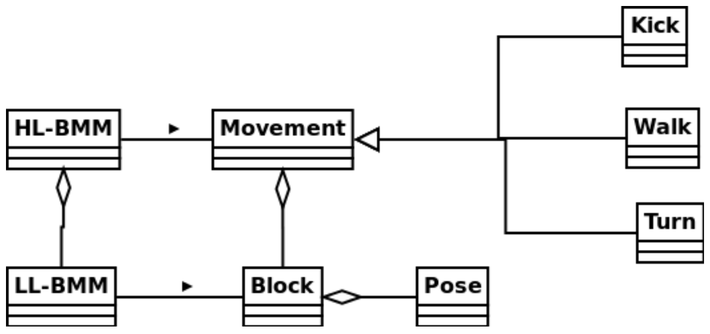


Fig. 3. Simplified Diagram of the BMM.

The HL-BMM is divided into next, continue, stop and finish modules:

- Next: Initialization of a new movement
- Continue: Continuing a previous movement, verify if the block execution is finished and send the next one, otherwise activate the continue module of LL-BMM.
- Stop: Emergency break, this is used if the robot has fallen.
- Finish: Systematic break, this respects the robot stability, calculating the blocks necessary to stop the movement and leave the robot stable.

4.2 LL-BMM: Low Level

The LL-BMM is the controller in classic control theory. A pose is a posture that the robot must achieve, which is composed of 22 angle values corresponding to the robot joints. The LL-BMM calculates the angular velocities needed to bring the joints' angles to values equals to the pose. The resulting values are the information sent to server to execute the movements. The LL-BMM has the next, continue, stop and init modules:

- Next: Receive a new block from HL-BMM
- Continue: Continues the execution of a block, verify if current pose has ended and start executing the next one, otherwise continues to execute the current pose.
- Stop: Send velocity 0 to the server to stop all joints and ends the block execution, waiting for the HL-BMM to give the next block.
- Init: This is a module that is part of Next, and it calculates the velocities from a given pose to the initial pose of the current block and it relies in a diferent algorithm for transition checking.

4.3 Nominal velocities and Transition checking

The LL-BMM velocities are calculated in a way to synchronize the movement, so they are scaled. These velocities are called nominal velocities. The transition checking of LL-BMM is based on signal of angle variation and the sum of speeds. The init module relies in a ϵ that is used to verify if every joint achieved the given value within a error margin.

5 TIC: Tactical Intelligence Center

In order for the agent to decide what action to take during the game, it was created the TIC. It is responsible for taking decisions during the simulation by analysis of the current state of the robot and the game.

5.1 Decisions based on the player's role

Once the simulation starts and the agent processing begins, it receives a static role according to its number. The possible roles are: Forward, Goalkeeper and Defender. Each role leads to a different line of reasoning when taking a decision. There is a generic reasoning shared by all players' roles and specific reasoning for each role.

Generic reasoning It's the first part in the robot decision taking and it is shared between all agents in the team. It defines the most basic priorities for any player. During this stage, the agent updates it's Kinematic State and verifies if it's fallen. In this case, it sets the action of standing up as its first priority.

If the robot is already standing, it checks the field to know its position and finally, sets up the movement of its head, which is managed differently from the other parts of its body.

Head movement The head management is responsible for moving the head to find the ball, when it's not seen, and to focus an object in the game.

To find the ball the agent scans the field by moving its head all the way to the left and right until it's found. Then the agent will focus on it by rotating its head in a way that the ball is always within its vision range.

Forward reasoning. Once the Generic reasoning is over, the forward players starts their own specific reasoning. The Forward basic decision is to kick the ball, aiming for the goal, whenever possible.

The first step is to verify if it's possible to kick in that exact moment. In this case, this action is taken. Otherwise, the agent will check if it is one of the two players closest to the ball among the agents of its own team. If so, the agent decides to walk towards the ball.

If it isn't one of the two players closest to the ball, it will move towards the end line, while the ball is in its own team possession, so when one of the two closest to the ball kicks it, the agent will be in a position with high probability to reach the ball.

Defender reasoning. The defender has a predefined sector for its own positioning in the field. When any opponent enters into this sector with the ball, the defender moves towards the ball trying to take its possession. If it is successful, the agent kicks the ball to a forward teammate.

Whenever no opponent goes into the defender's sector, the agent stays on its strategic position.

Goalkeeper reasoning. The goalkeeper is responsible for defending the goal. Once the opponent is close enough to kick the ball into the goal, it prepares itself to defend, by positioning in the middle of the goal. Once the ball is at a certain distance, the goalkeeper will fall down in the direction the ball is coming in order to block it.

5.2 Positioning

The positioning of the robot is divided into three different types, which are used differently depending on the request. They are: *PositionBallGoal*, *RelativePositioning* and *DefendingPositioning*.

PositionBallGoal This positioning is responsible for making the agent position itself behind the ball, and close to it, aiming the opponent goal with its front vector pointing towards the goal. This method is capable of position the robot this way from whichever position it is, dodging the ball so it won't move it unnecessarily as it can be seen in the scenarios in Fig. 4.

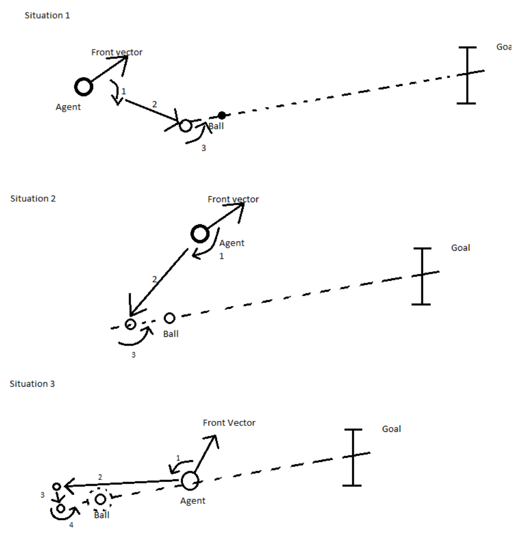


Fig. 4. PositionBallGoal scenarios.

In the first scenario, the ball is ahead of the agent, so it'll just have to go to the point behind the ball and in the direction of Ball-Goal vector. In the second case, the agent has passed the spot where the ball is, therefore it must turn back, and position itself in the same vector as in the first case. The third case is the most complex, the agent is ahead of the ball and if it tries to go to the expected position simply like the other ways, it'll collide with the ball. So, to avoid that, the robot will first go to an auxiliary position, behind the ball, passing by its side, on a determined security distance, then, it will go to the target.

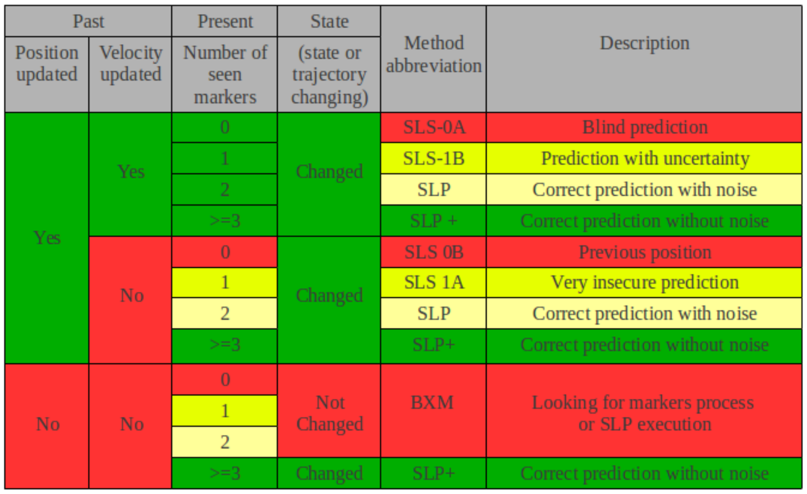
RelativePositioning is the simplest method for moving the robot. It will move the agent towards any defined point in the field, using the coordinates received, with no concerns on opponents or teammates in the way.

DefendingPositioning. Very similar to the PositionBallGoal, the Defending-Positioning will calculate the path the robot shall take to position itself in front of the opponent agent, between it and the agent's own goal.

6 Localization

The approach to calculate the robot's position depends on how many flags (markers on the field like corners or goal posts) have been seen in the last vision cycle. It also depends on changes in the current plan of action (trajectory shift or involuntary falls there demands immediate action of standing up) and also depends on the position and velocity update status (if they were successful calculated during last vision cycle).

The choice of the approach to be used is made according to the conditions established in the figure 5.



Past		Present	State	Method abbreviation	Description
Position updated	Velocity updated	Number of seen markers	(state or trajectory changing)		
Yes	Yes	0	Changed	SLS-0A	Blind prediction
		1		SLS-1B	Prediction with uncertainty
		2		SLP	Correct prediction with noise
		>=3		SLP +	Correct prediction without noise
	No	0	Changed	SLS 0B	Previous position
		1		SLS 1A	Very insecure prediction
		2		SLP	Correct prediction with noise
		>=3		SLP+	Correct prediction without noise
No	No	0	Not Changed	BXM	Looking for markers process or SLP execution
		1			
		2			
		>=3	Changed	SLP+	Correct prediction without noise

Fig. 5. SLS: Secondary Self-Localization, SLS-0X: Secondary Self-Localization without visible markers, version X, SLS-1X: Secondary Self-Localization with 1 visible marker, SLP: Simultaneous Primary Self-Localization and BXM: Searching markers process

There are four different colours to classify how hard is the situation for calculating an accurate position. The red indicates the worst case, the green the best one. The darker yellow represents a less favourable situation than the lighter one.

6.1 Self-Localization with 2 markers (SLP)

If the robot sees 2 markers one of the coordinates of an unitary vector with absolute direction from camera named observation vector $\boldsymbol{O} = (\boldsymbol{O}_x, \boldsymbol{O}_y)$ can be directly calculated, but the other one still unknown. The global coordinates of the j-nth marker seen is represented with the vector $\boldsymbol{F} = (\boldsymbol{f}_x^j, \boldsymbol{f}_y^j)$. If $\boldsymbol{f}_x^i = \boldsymbol{f}_x^j$, then the known coordinate is $\boldsymbol{O}_x$ and the unknown is $\boldsymbol{O}_y$. When $\boldsymbol{f}_y^i = \boldsymbol{f}_y^j$ the known coordinate is $\boldsymbol{O}_y$ and the unknown is $\boldsymbol{O}_x$.

The known coordinate $k = x$ or $k = y$, is given by (3), where ρ is distance and φ the angle between markers i and j.

$$O_k = \frac{\rho_i \cos(\varphi_i) - \rho_j \cos(\varphi_j)}{f_k^i - f_k^j} \tag{3}$$

By definition, the given vector $\boldsymbol{O}$ is unitary, so $\|\boldsymbol{O}\| = 1$. However, because of the noise in the system sometimes it does not occur, then this is limited for $|\boldsymbol{O}_k| \leq 1$ before calculating the two options to the other coordinate of $\boldsymbol{O}$, which is given by (4)

$$O_u = \pm\sqrt{1 - O_k^2} \tag{4}$$

Simultaneous SLP Multiple (SLP+) When more then 2 markers are seen in the same vision cycle, Self-Localization with 2 markers (SLP) must be used with multiple pairs of markers and their results used in an average value. This value is naturally less noisy then the ones used to define the average.

6.2 Secondary Self-Localization (SLS)

The SLS is used if the position of the agent in the last vision cycle $\boldsymbol{R}^{pre}$ is known, but in the current vision cycle it doesn't see 2 flags for executing the Self-Localization with two markers (SLP).

SLS without markers (SLS-0) In this case that was no markers seen, so estimating it's new position depends on the additional information stored from the past. If there is no $\boldsymbol{V}_p$(velocity in previous cycle), the best solution is assuming the same position of the last vision cycle, configuring the method SLS-0A from table 5. Knowing $\boldsymbol{V}_p$ the agent can predict it's position using the most stable tendency.

SLS with one marker (SLS-1) When there is just 1 flag seen in a distance ρ and the agent knows the marker is in the global position $\boldsymbol{F}$, it is possible to take advantage from the information about the relative position of this flag to estimate the current position of the agent. The SLS-1B algorithm is used when both $\boldsymbol{T}$ (direction tendency) and velocity $\boldsymbol{V}_p$ are known and the SLS-1A if it is not.

SLS-1A - Very insecure prediction In this case the current position is estimated by (5) and (6).

$$\boldsymbol{R}* = \boldsymbol{F} + \rho \boldsymbol{r} \tag{5}$$

$$\boldsymbol{r} = \frac{\boldsymbol{R}^{pre} - \boldsymbol{F}}{\|\boldsymbol{R}^{pre} - \boldsymbol{F}\|} \tag{6}$$

SLS-1B - Prediction with uncertainty When the needed information is available the current agent's position is estimated by (7)

$$\boldsymbol{R}* = \boldsymbol{R}^{pre} + b\boldsymbol{p} \tag{7}$$

b is the scalar vector that will be determined and $\boldsymbol{p}$ carries the information about the current movement in execution and can be chosen as $\boldsymbol{T}$ or $\frac{\boldsymbol{V}_p}{\|\boldsymbol{V}_p\|}$.

7 Work in progress

Many improvements are being developed in our team to increase their competitiveness to compete in the RoboCup 2013. The main improvements are: i) improving world model adding a memory to the agent enabling the storage of information from previous cycles; ii) detect if a robot is dropped using vision information; iii) introduce agents communication in the team; iv)enhance the walking movement. These improvements are detailed in following subsections.

7.1 Adding memory to World Model

The Bahia3D agent has a reactive behaviour turning harder to implement more complex algorithms and AI techniques. To solve this problem it was designed a memory based on decision theory described in [2] and [3], and inspired by entropy calculation of information theory described in [4], where the world model is being changed to be part of a memory which stores a sequence of last past information. The memory has the following attributes:

- Creation Cycle C_c
- Reliability R that is calculated $R = \frac{C_c}{A_c}$
- The information

Obs:A_c is the current cycle.

After the changes, the simplified class diagram of Bahia3D agent will be as described in figure 6.

7.2 Drop detection

In order to the agent to take decisions, it needs to analyse the field, its teammates and its opponents.

One of the most important things to know about our own teammates and opponents is whether or not they are able to get the ball, or defend against a

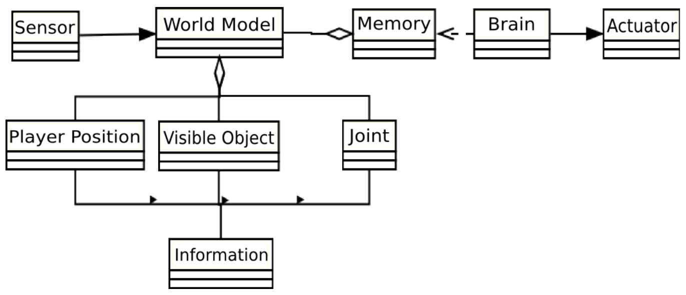


Fig. 6. Simplified Class Diagram of Bahia3D agent.

foe. To be able to do so, the agent must be standing up. So it's very important to our agent being capable of checking if another robot is fallen or not.

To verify the status of other agents, it was proposed the analysis of the z axis coordinate of that agent head, and setting a parameter value which would suggest if the robot was standing up or laid down. This method will become part of the PKS.

7.3 Introducing agents communication

Observing the execution of team Bahia 3D it was detected that all agents were sending messages but only the first connected agent is heard. It is not efficient in terms of communication because the sent messages by other agents are lost.

Thus we are developing a new communication framework based on the the idea of using the goalkeeper as the unique agent to speak. This choice was made because this agent is the unique which can see the entire field most of the time due to its strategic position.

In a first version, the goalkeeper will always send the ball's position to the other teammates. This way, every agent in our team will always know the location of the ball and can decide to walk to it when necessary.

The evolution of this framework will result in goalkeeper sending other information as the proximity of a opponent to an agent which is with the ball possession, tactical and strategic coordination and other relevant communication in the team.

This communication solution can be used while 3D Simulation rules enable speaker actuators and vision sensors to reach all the field. If the field size is enlarged or the actuators or sensors limited we would need to adapt this strategy.

7.4 Enhancing walking movement

Current walking movement of Bahia3D agent is based on predefined scripts. Studies are underway to optimize the parameters of this movement enabling a faster walking. Another work in progress is the creation of a module for real-time generated walking. This new movement tends to be more adaptable enabling the agent to deal with unpredictable situations like aperiodic walking.

Both movements (predefined and real-time generated) will be optimized using a multiobjective evolutionary framework under development in our research group. The main idea is to consider optimization goals as movement velocities, direction, stability to find optimal parameters for both movements. Then the best one will be chosen to be used by the agent. It is also possible that the performance of the movements leads us to decide for using one movement for some situations and the other one for other scenarios.

8 Conclusions

Team Bahia3D has been developed with the objective to serve as a framework for testing experiments in the areas of research of group Bahia Robotics Team. Our main research areas are optimization, multiagent systems, planning and machine learning.

The current agent architecture described in this paper supports our research interests. The new World Model is more flexible and represents an important part of the agents' knowledge base. The addition of a memory to WM turns the agent capable of work successfully on a sequential environment.

The new movements manager structure leads to a very stable movement. The unique weakness is the current movement low velocity for some important movements like *walking*. Using our research results in optimization, we plan to enhance our walking movement enabling a better team performance. We will use two approaches as describe in this paper. On a first approach offline optimization will be used to enhance the current motion scripts. In the second approach we will use real-time optimization to fine tune the movements during agent execution.

We are also introducing communication in the team to enable the use of coordination as an advanced cognitive feature for the agent structure.

Considering the current stage and the work in progress there is a high chance to present a very competitive team in RoboCup 2013.

References

1. Factum, R., Frias, D.: Proposal of architecture for a non-cooperative and non-concurrent self adaptive multi agent system on reactive level for a robot soccer team. In UNEB, ed.: XVI Journey of Scientific Initiation of the UNEB, UNEB (2012) (in Portuguese).
2. Russel, S.J., Norvig, P.: Artificial Inteligence A Modern Approach. second edn. Prentice Hall (2003)
3. Valle, S.M.L.: Planning Algorithms. Cambrige University Press (2006)
4. Jones, G.A., Jones, J.M.: Information and Coding Theory. Springer (2000)