J Intell Robot Syst
DOI 10.1007/s10846-010-9515-7

A Real-Time Hybrid Architecture for Biped Humanoids with Active Vision Mechanisms

Javier Testart · Javier Ruiz del Solar · Rodrigo Schulz · Pablo Guerrero · Rodrigo Palma-Amestoy

Received: 14 June 2010 / Accepted: 1 December 2010

Abstract A real-time hybrid control architecture for biped humanoid robots is proposed. The architecture is modular and hierarchical. The main robot's functionalities are organized in four parallel modules: perception, actuation, world-modeling, and hybrid control. Hybrid control is divided in three behavior-based hierarchical layers: the planning layer, the deliberative layer, and the reactive layer, which work in parallel and have very different response speeds and planning capabilities. The architecture allows: (1) the coordination of multiple robots and the execution of group behaviors without disturbing the robot's reactivity and responsivity, which is very relevant for biped humanoid robots whose gait control requires real-time processing. (2) The straightforward management of the robot's resources using resource multiplexers. (3) The integration of active vision mechanisms in the reactive layer under control of behavior-dependant value functions from the deliberative layer. This adds flexibility in the implementation of complex functionalities, such as the ones required for playing soccer in robot teams. The architecture is validated using simulated and real Nao humanoid robots. Passive and active behaviors are tested in simulated and real robot soccer setups. In addition, the ability to execute group behaviors in real-time is tested in international robot soccer competitions.

Keywords Hybrid control architecture · Active vision · Nao humanoid robots · Robot soccer

1 Introduction

Mobile autonomous robots are becoming complex systems that are able to interact with humans and other robots in complex, challenging, and often dynamically changing environments. Examples of such challenging conditions are the ones defined in

J. Testart · J. Ruiz del Solar (✉) · R. Schulz · P. Guerrero · R. Palma-Amestoy
Department of Electrical Engineering & Advanced Mining Technology Center, Universidad de Chile, Santiago, Chile
e-mail: jruizd@cec.uchile.cl

Published online: 30 December 2010

robot soccer competitions (e.g. RoboCup [36]), the Darpa Grand and Urban Challenge competitions [39], and field robotics applications (e.g. autonomous vehicles in underground mines). Robots operating in such challenging conditions need to be controlled appropriately. Robot control is *the process of taking information about the environment through the robot's sensors, processing it as necessary in order to make decisions about how to act, and executing actions into the environment* [25].

Deliberative, reactive and pure behavior-based control paradigms have been known for many years (e.g. sense-plan-act, *subsumption*, and motor schemas, just to name a few), and combinations of them are often used in today's applications. Robot control architectures must incorporate reactivity, planning capabilities, and a modular and hierarchical design [24]. Hybrid, or tiered control architectures incorporate dynamic, concurrent, and time-responsive control (reactive control), together with efficient decision making over long time scales (deliberative control) [25]. In addition, their clear interface definition has the advantage that the different layers can be developed and modified in parallel [24].

Taking all these elements into consideration, we have developed a hybrid control architecture for biped humanoid robots. The architecture is modular and hierarchical. It organizes the main robot-control functionalities in four parallel modules: perception, actuation, world-modeling, and hybrid control. The hybrid control module is distributed among three behavior-based hierarchical layers: the planning layer, the deliberative layer and the reactive layer, which work in parallel and have very different response speeds and planning capabilities. The architecture allows (1) the coordination of multiple robots and the execution of group behaviors without disturbing robot reactivity and responsivity, which is very relevant for biped humanoid robots whose gait control requires real-time processing, (2) the straightforward management of the robot's resources using resource multiplexers, and (3) the integration of active vision mechanisms in the reactive layer, under control of behavior-dependant value functions from the deliberative layer. This last feature adds flexibility in the implementation of complex functionalities, such as the ones required for playing soccer on robot teams.

Although the architecture is designed to be general-purpose, the short-term goal is to use it in robot soccer applications. For this reason, the descriptions of some module's implementations are exemplified for the robot soccer case.

This paper is organized as follows. In Section 2, some related work is presented. The proposed hybrid control architecture is described in Section 3. In Section 4, a validation of the architecture using simulated and real Nao humanoid robots is presented. This validation includes the testing of group behaviors in international robot soccer competitions. Finally, in Section 5 some conclusions of this work are given.

2 Related Work

The development of robot control architectures for mobile robots started in the late 60's with the development of the Shakey robot, whose control architecture was known later as the *sense-plan-act* paradigm. Some of the limitations of this "full" deliberative architecture were tackled in the seminal works of Brooks (*subsumption* architecture) [3] and Arkin (motor schemas, AuRA architecture) [1], who developed the *reactive control* and *behavior-based control* paradigms, respectively. In both

views the real-time coupling between sensing and action is maintained. However, behavior-based systems can store representations, while reactive systems cannot [25]. The hybrid control paradigm incorporates different layers of reactive, deliberative and behavior-based control [24], i.e. the 3T architecture defines the following layers: planning, executive, and behavioral control [26]. Currently, hybrid control architectures incorporate dynamic, concurrent, and time-responsive control, together with efficient decision-making over long time scales [25]. In addition, their clear interface definition has the advantage that the different layers can be developed and modified in parallel [24].

In the last years, several works have reported interesting hybrid architectures and applications of hybrid architectures [5, 8, 11–13, 16–18, 20, 22, 27], as well as related approaches, such as the dual-dynamics one [4]. It is beyond the scope of this paper to make an extensive review of hybrid control architectures, however, an overview of the state of the art in behavior-based, hybrid, and behavior-based hybrid control architectures can be found in [24, 25].

Of special interest for this work are approaches devoted to describing control systems for legged robots [2, 10, 14, 15], legged soccer robots [6, 7, 19, 21], and simulated soccer robots [9, 23]. In [2, 10, 14, 15], the EGO (Emotionally GrOunded) control architecture for QRIO humanoids is described, as well as some of their main variants and extensions to AIBO robots. The architecture is related to the one being proposed in this work, but it includes an emotional model, and short-term and long-term memory. In our case we do not include emotional functionality, but we do include short-term memory of the objects of interest, and active vision mechanisms, which are not included in the EGO architecture. In [6, 7] a robot control architecture for wheel-based soccer robots, which was later extended to humanoid robots, is proposed. As in our case, this architecture defines three hierarchical organization levels that are called *team*, *player*, and *body*. However, that architecture is not able to manage active vision behaviors, and it does not define specific robot's resources managers. In [9], a control architecture for simulated robot soccer players is proposed. Main modules include a *deliberator*, an *executor*, a *hierarchical behavior model*, and a *context pool*. In [23], the DAInamite framework combines reactive and deliberative control with learning in the case of simulated soccer robots. The main difference of our architecture from the ones proposed in [9, 23] is that ours takes into account the real-time control requirements of real humanoid robots, and it includes active vision mechanisms at the reactive level. The hybrid control architecture proposed in [19] is intended for AIBO robots, and therefore it does not incorporate some of the functionalities required in biped humanoids, or active behaviors. Finally, in [21], a behavior-based control architecture for humanoid robots is proposed. This architecture is simpler than the one proposed here, and it doesn't have the flexibility of layered architectures, or active vision mechanisms.

3 A Real-Time Hybrid Architecture for Humanoid Robots

3.1 Architecture Overview

The architecture is modular and hierarchical. The robot's main functionalities are organized in four parallel modules: perception, actuation, world-modeling, and hybrid control. Figure 1 shows a block diagram of the architecture.

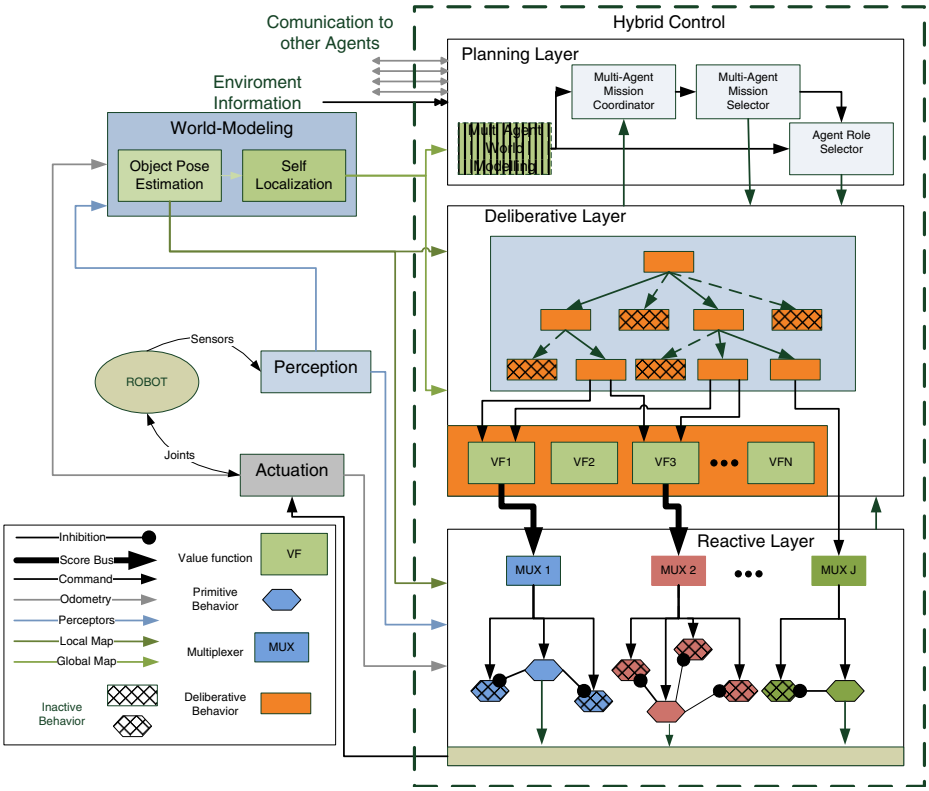

Fig. 1 Block diagram of the proposed hybrid architecture

Perception and actuation are the modules that interact directly with the robot hardware (sensors and motors). Perception is in charge of acquiring, processing and analyzing all data coming from the robot's external and internal sensors. In a humanoid robot, perceptual data typically includes inertial (e.g. accelerometer and gyroscope sensors) and visual data. Any given vision algorithm can be used in this module, for instance color-based or SIFT-based vision. The perceptual data is transmitted to the lowest level of the hybrid control module (reactive layer), and to the world-modeling module where it is further processed. Actuation is in charge of translating orders generated in the reactive layer (e.g. looking at a given position) into orders to be sent directly to the robot's motors. In addition, actuation is in charge of reading encoders' data, which is transmitted back to the reactive layer and to the world-modeling module.

World-modeling maintains a representation (or map) of the robot's environment. We improve classical self-localization approaches by estimating, independently and in addition to the robot's pose, the pose of the static and mobile objects of interest. This allows using, in addition to fixed landmarks, dynamic landmarks such as temporally local objects (mobile objects) and spatially local objects (view-dependent objects or textures). Moreover, the estimation of the pose of objects of interest allows the robot to carry out certain tasks, even when having high uncertainty in its own

pose estimation. This is especially valuable when performing attention-demanding tasks, like tracking a ball. Another nice feature of the proposed system is that the robot is able to correct its odometry even when it is totally lost [28]. In this sense, this approach moves in the direction of performing tasks with much less use of global localization, as humans certainly do. Furthermore, the estimation of the pose of the fixed-landmarks allows having global measures of the robot's localization accuracy, by comparing the real map, given by the real (known *a priori*) position of the fixed-landmarks, with the estimated map, given by the estimated pose of these landmarks. Finally, having an estimation of the pose of the objects allows the straightforward implementation of active vision behaviors (see an example in Section 4). Object pose estimation and robot self-localization are implemented using Bayesian filters. In our current implementation [28], several Extended Kalman Filters (EKFs), one for estimating the pose of each object of interest and one for estimation the robot's own pose, are used. The observations of each object are used to correct the estimation of the corresponding EKF. For instance, in the robot soccer case the observation of a goal-landmark is used to correct the EKF that estimate the pose of this landmark. The same procedure is used to correct the object's pose estimation of the other filters. Then, the estimated poses of all objects of interest (goal-landmarks and lines in case of robot soccer) are used as observations in the EKF in charge of estimating the robot's pose.

The hybrid control module is organized in three behavior-based hierarchical layers: planning layer, deliberative layer and reactive layer, which work in parallel and have very different response speeds and planning capabilities. The planning layer is in charge of long-term planning, using information coming from other robots and eventually external devices (e.g., a game controller in the case of robot soccer) in order to determine the robot's mission and role, and its coordination with other robots. The deliberative layer executes the mission defined in the planning layer using deliberative behaviors, each of them organized as state machines, in which each state is implemented as a single behavior or as a behavior-tree. The deliberative layer configures goals for the primitive behaviors available in the reactive layer, and sets up the parameters needed by these behaviors. It uses behavior-dependent value functions (VFs) to score the goals needed by primitive behaviors, which allows implementing active vision mechanisms. Finally, the reactive layer directly decides the actions that must be executed when resources from the robots are available. The robot resources are managed by resource-multiplexers (MUXs), which update goals in order to make decisions in real time.

It is worth to mention that all described modules run in the robot's internal computers.

3.2 Basic Definitions

A *mission* is a high-level task to be carried out by robot agents. The mission can be accomplished by a single robot or by several robots, and it is coordinated in the planning layer. For instance, in soccer a mission could be to *defend the goal*, which is carried out by the goalkeeper, or to *attack* the opponent team, which can be carried out by several strikers, depending on the game situation. If more than one robot carries out the mission, different roles can be assigned to the different robots participating in the mission. For instance, in a soccer attacking situation with two attacking robots,

Table 1 Missions and roles defined in the control architecture for a robot soccer application

Mission name	Description	Roles names	Description
Defend the goal	The goalkeeper should prevent the opposite team from scoring by defending the goal.	Goalie	The goalie is a special role, because only one player can be a goalkeeper during the match.
Attack	All field players trying to score a goal for their team are in attack mission.	Striker	The attacking robot that is nearest to the ball. It can dribble and then kick the ball to the opponent goal or to a partner.
		Striker cover	The robot that supports the striker, keeping a nearby position.
		Pass receiver	A robot waiting for a pass in a free position.
Defend	All field players preventing the opposite team from scoring a goal are in a defend mission.	Active defender	The robot that is nearest to the ball position.
		Defender going to position	A robot that is far from the position of the ball.

the robot that is closer to the ball can take the *striker* role, while the second robot can take the *striker cover* role. The mission and role information of the robot are transmitted from the planning layer to the deliberative layer. Table 1 shows some examples of missions and roles for a robot soccer application.

Deliberative Behaviors are in charge of executing the mission targeted by the planning layer, by dividing it into different *goals*. The goals are selected by the deliberative behaviors depending on the world-state, and the internal state of the robot. The deliberative behaviors must set up at least one possible goal for each of the resources offered by the robot. A *goal* is the mechanism used in the reactive layer to configure its *primitive behaviors* in order to produce an action. For example, in a soccer application the *observe-ball* goal can trigger the primitive behavior *search-object-ellipse* in the reactive layer if the robot has no information about the ball position, or the primitive behavior *object-tracking* if it has such information. In Tables 2, 3 and 4, examples of deliberative behaviors, goals, and primitive behaviors for a robot soccer application are shown.

3.3 Planning Layer

The planning layer is designed to control multiple robots' coordination and group behaviors. Team decisions are taken distributely (no leader robot is required), and fluid communications between robots are a key issue in order to implement this. However, unavoidable interruptions in the communication channels should be taken into account when designing the system. For instance, the experience of the robot soccer community (e.g. RoboCup [36]) is that wireless communication failures can largely affect the performance of world-class soccer teams.[1] Thus, the first principle

[1]The authors have the experience of several years participation in RoboCup soccer competitions.

Table 2 Examples of deliberative behaviors defined in the control architecture for a robot soccer application

Deliberative behaviors	Description
Field playing	This behavior is the top-level behavior for the robot in case the robot is not the goalkeeper, but a field player.
Goalie playing	This behavior is the top-level behavior for the goalkeeper.
Partner following	This behavior is selected when the robot has the attack mission, and its role is striker cover.
Going to field position	This behavior is used when the robot must go to a specific area on the field, or for getting a proper positioning in the case of the goalkeeper.
Go to ball	This is the main behavior for going to the ball and then kicking the ball or dribbling. It evaluates targets for the ball depending on the mission selected for the robot and the robot's role.
Goal covering	This behavior is designed to cover the maximum angle to the goal. The setup used by this behavior depends on the robot mission and on its role (field player or goalkeeper).
Localizing	This behavior allows the robot to localize itself.
Searching for object	This behavior is set up to handle a passive and active search of an object in the field. Passive means searching just around its own place, while active search involves changes in the robot position.
Avoiding obstacle	When an obstacle is detected, this behavior handles obstacle avoidance at a high level.

chosen for minimizing this issue is to have robot communication only at the highest layer of the hybrid control module, so that in case communication is lost, a robot would not lose its own abilities. A second principle is to limit the amount and frequency of data transmission between robots; only relevant information should be communicated. Thus, only the internal-state, basic communication information and the world-model are communicated. In Fig. 2 is shown the data structure used for robot communication. The internal state includes the robot's mission, role, status, and identifier (ID). In the case of humanoid robots, status information includes basic

Table 3 Examples of goals and associated robot's resources for a robot soccer application

Resource	Goal	Parameter
Robot head	Observe ball	
	Observe goal	Own or opposite goal.
	Observe field line	Line identifier.
	Observe partner	Partner identifier.
	Observe rival	Rival identifier.
	Observe target	Target position.
Robot body	Go to position	Position.
	Go to ball and kick	Kick identifier.
	Go to ball and dribble	
Robot head and body	Stand-up	Stand-up sequence identifier.
	Fall	Fall sequence identifier.
	Dive	Dive sequence identifier.

Table 4 Examples of primitive behaviors defined in the control architecture for a robot soccer application

Multiplexer	Primitive behaviors	Description
Robot head	Search object ellipse	Predefined head ellipse enabling the search of a given object. The ellipse shape and size depend on the object.
	Object tracking	Track an object with the camera and the head.
	Look at target	Target the camera to a given position.
Robot body	Go to position	Allow the robot go to a given position.
	Go to ball	Similar to go to position but taking care that the final robot position will allow it to manipulate the ball (e.g. kick).
	Dribbling	For dribbling the ball while walking.
	Avoid collision	For avoiding an imminent collision.
	Kick ball	For kicking the ball when necessary.
Robot head and body	Stand-up	For standing-up after a fall.
	Fall	To minimize the joint torque when instability is detected, and a fall sequence is triggered to avoid robot damage.
	Dive	Dive sequence used by a player (normally the goalkeeper), to prevent a goal.

status data such as the robot standing status (stable, unstable or fallen). The basic communication information includes the time elapsed since the last communication and a data received flag. The world model includes the robot's pose, the local map, and the associated confidence values. The local map includes the position of objects of special interest for the robot (e.g. the ball in case of robot soccer).

The coordination between robots assumes that robots are homogeneous: all robots have similar motor and perceptual capabilities. In the case of a soccer application, it is also assumed that all robots move in a known field, allowing the use of a single coordinate's system in the global map of the robots.

The Multi-Agent Mission Coordinator (MAMC) is in charge of evaluating the internal-state of the other team members, and it determines which partners are available for a group mission. The evaluation is made considering the other robots' missions, roles and deliberative-layer states. The MAMC must be able to detect when a robot is not accomplishing the assigned mission or when it has not transmitted

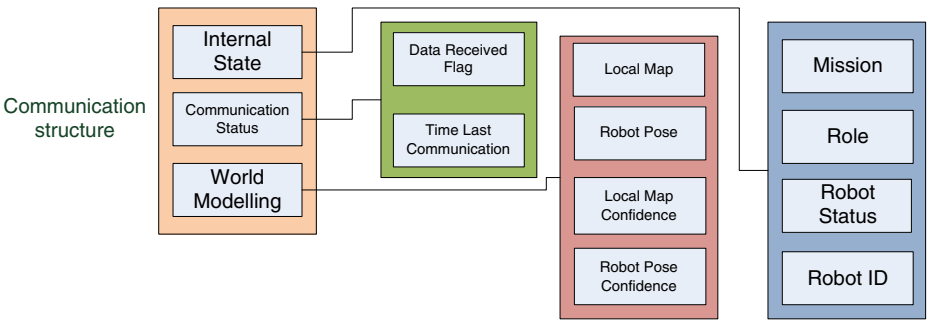

Fig. 2 Diagram of the data structure used for communication between the robots

status information for a long period, to set it as unable to accomplish the mission that was assigned to it, and to free this role in the mission. For instance, in a soccer application, if the mission is to attack and the striker falls down, the MAMC of the other robots must eliminate it from the available team members until the robot stands up and localizes itself again. The Multi-Agent Mission Selector (MAMS) is in charge of evaluating the information that all teammates make available, and determining the mission to be accomplished by the robot or the team, in case the selected mission can be executed by more than one robot. After a mission is assigned, each robot should continuously evaluate if the mission can still be accomplished (the environment may have changed), or if a new mission should be selected. The MAMS has a default mode or a default mission in case there is no communication or communication is lost. For instance, in a soccer application a field robot sets up *attack* as the default mission, and *striker* as the role. The Agent Role Selector (ARS) selects a role in the context of a specific mission. It considers the MAMC information about the teammates roles to select its own role. If different team members are executing the same role, the first robot that detects the double role assignment is in charge of informing the others.

The Multi-Agent World Modeling (MAWM) is in charge of fusing the different world-models transmitted by the available teammates. The fusion procedure can be implemented with an approach similar to that of [32], in which each robot uses a Kalman filter to fuse the data corresponding to each object of interest.

In our current implementation for Nao humanoid robots (see Section 4), the communication between the different robots is carried out using UDP broadcasts over a wireless network. The data structure transmitted contains the estimated global position of the ball in the local map, the estimated own pose computed by the self-localization module, the mission, the role, and the internal status of the transmitter robot. Each robot transmits this information every 30 s. In case the robot falls down or loses any critical perceptual information for determining its mission and role, it informs immediately the change in it internal state (status field) to the team, by sending an instant message without waiting the timer. The communication between robots is completely asynchronous, so whenever a robot receives new information from a teammate it performs an evaluation of the planning layer, and it only sends information to the deliberative layer in case its mission and role need to be changed.

3.4 Deliberative Layer

The deliberative layer manages the mission that the planning layer has defined. At this level, deliberative behaviors are organized as state machines, whose structure depends on the robot role. Each state can be structured as a single behavior or as a behavior-tree. The deliberative layer is able to make long-term decisions such as trajectory planning. It configures the multiplexers available in the reactive layer, and it sets up the parameters required by them in order to select the primitive behaviors.

The deliberative behaviors configure the resource multiplexers directly or through a *Value Function*, which are used in case that multiple objectives need to be handle. In both cases a *score vector*, which contains the score corresponding to each goal that can be achieved by the robot, is sent to one or several resource multiplexers. The different score values depend on the mission being executed, the robot's internal state, the active deliberative behavior, and the world state. The mission's time

requirements define the minimal rate at which the scores are updated. In addition, even though the resource multiplexers update the score values using perceptual data (see Section 3.5.2), they cannot change the overall system state or the goals.

Value Functions are in charge of evaluating and sometimes merging robot's goals, depending on the current behavior and mission, and they permit implementing active vision behaviors in the architecture. A value function is defined as a function of the world-state, whose value increases with the degree of benefit of the world-state for the execution of the corresponding deliberative behavior [30]. Thus, they allow selecting the benefit of observing a given target, depending on the deliberative behavior being executed. For instance, while executing the *goal-covering* behavior that requires that the goalkeeper places itself in such a way that it covers the maximal free angle of the goal when observed from the ball position (see Section 4 for a more detailed explanation of this behavior), the corresponding value function will evaluate the benefit of observing the ball for having a better estimation of its position, or of observing the landmarks for having a better estimation of the robot's pose. Value functions also evaluate the option of looking at multiple objects in case this is possible, and they are able to determine a spatial position from which they can be observed simultaneously. Section 3.6 describes the module that implements the active vision system where value functions are used.

3.5 Reactive Layer

This layer is the fastest one, and it decides directly which action must be executed when resources from the robots are available. It is built using primitive-behaviors, which are able to inhibit other primitive-behaviors that use the same resources. The robot's resources are joints that need to work simultaneously in order to reach a goal. A resource could be just a joint or a group of them. For instance, in a humanoid robot all joints of the legs are considered as a single resource required for walking. Actuation is implemented to coordinate and monitor the different joints in the system, and it asks the reactive layer for the next action in a specific resource when this resource is going to be released. The reactive layer gets data from all sensors available in the robot (e.g. accelerometers, gyroscopes, cameras), and also from the deliberative layer. The data coming from the deliberative layer is used to configure activation conditions and short-term goals for the primitive-behaviors, while sensor data is used as parameters for the primitive behaviors.

Resource multiplexers allow handling multiple goals for the same robot's resources by allowing the activation of primitive behaviors that will use the corresponding resources. Tables 3 and 4 show some examples of resources, resource multiplexers, and the associated goals and primitive behaviors for the case of a robot soccer application. The sensorial data is collected at different rates depending on the sensor measurement rates (e.g. accelerometer data is obtained at a much higher rate than images), and primitive behaviors always use the latest available data.

3.5.1 Primitive Behaviors

Primitive behaviors are the fundamental building blocks of the hybrid control system. They correspond to simple behaviors which can be executed very quickly, and that are associated with group of robot resources. For instance, in robot soccer the *object-tracking* and *search-object-ellipse* primitive behaviors are associated with the robot

head. Whenever new data is available, the selected primitive behavior determines the next action to be executed by the robot in order to accomplish the desired goal. The deliberative layer is responsible for the configuration of the primitive behaviors that will be used, but the final decision about which primitive behaviors will be activated is made using perceptual data. This allows the system to be more reactive. For example, the selection of the *kick-ball* or *go-to-ball* primitive behaviors is made using perceptual data, while the kick to be used is given as a parameter defined by the deliberative layer. The primitive behaviors are only aware of a short-term policy, and they do not take into account the consequences of the actions or the changes produced by them in the environment. An activated primitive behavior inhibits all other primitive behaviors that require the use of the same robot resources. In this sense, all activated primitive behaviors are orthogonal behaviors [25], because they do not use the same resources.

The deliberative layer can deactivate primitive behaviors. In fact, a deliberative behavior can decide that a primitive behavior must not be activated for the accomplishment of a given mission. For instance when the robot decides it needs to dribble the ball, instead of kicking it, the *kick-ball* primitive behavior is deactivated.

The primitive behaviors can decide on three different types of policies to be used for sending commands or orders to actuation:

Queue Policy	The command is placed as the last one in the queue of actions to be executed. The principle behind having a queue of commands is to allow the continuous movement of the robot (e.g. continuous walk), by avoiding the initializing and ending steps that are only performed when they are required.
Non-queue Policy	The command is not queued, and it overwrites all queued commands.
Special Policy	The command corresponds to a non-parametric sequence of movements, for instance, kick movements, stand-up sequences and fall sequences.

Actuation sends three different kinds of messages to the reactive layer:

Command-Odometry	Odometric data of the current robot command, sent from the actuation module to the reactive layer.
Command-Finishing-Notification	Actuation notifies the reactive layer that a command execution is going to finish, and that resources used by this command will be released.
Command-Cyclic-Part	In case of cyclic movements, actuation notifies that the command being executed can be changed without disturbing the robot. The underlying concept is that a cyclic command can be divided into three stages: *initialization*, *execution*, and *end*. While the initialization and end stages are common to all movements, the execution part is the one that can be modified while executing cyclic movement (e.g. robot cyclic walk).

3.5.2 Resource Multiplexers

Resource multiplexers manage robot resources by selecting the goal for primitive behaviors that will access them. In the case of our implementation for humanoid robots, the resources to be managed are the robot body and the robot head. There are multiplexers in charge of selecting goals that will activate primitive behaviors that will send commands to the head, to the body, and to the head and the body at the same time (see Table 4). Each multiplexer receives the perceptual data acquired at time step k, and a score vector from the deliberative layer, which include a *deliberative score Sd*, and some additional parameters for each possible goal. The structure of the score vector can be seen in Fig. 3.

In the multiplexer, an activation function computes new goal scores in two stages. The first stage is optional, and it is used only if the score vector includes more than one goal. In this case, whether the goals can be executed simultaneously or not is verified. An example of simultaneous goals is observing two different objects at the same time (e.g., a ball and a goal in robot soccer). If this simultaneous action is not possible given the current situation (e.g., robot pose), then the goal can be divided into two, or even more goals. The second stage consists of computing a new score for each goal using *Sd* and a reactive score *Sr*, which is refreshed every time a new perception is received. *Sr* and *Sd* can be combined in different ways. In our current implementation *Sr* and *Sd* are averaged. Computing the final scores in the reactive layer allows the system to react to fast changes in the environment. In fact, if no scores were actualized in the reactive layer, the system would only be able to evaluate goals at the deliberative layer's frequency. However, it is important to note that the deliberative score *Sd* is associated to a long-term objective so it remains unchanged until a new one is computed in the deliberative layer.

Once the goal is selected by the multiplexer, one primitive behavior is activated and executed taking the perceptual data into account. The selected primitive-behavior inhibits the others that could require the same robot's resources. For example, in the case of *observe-goal*, two primitive behaviors can be selected *object-tracking*, in order to track the goal, or *search-object-ellipse*, in order to find the goal.

In the case of primitive behaviors associated with the robot head (see Table 4), the multiplexer should also determine, in case of having multiple goals (i.e., multiple objects need to be observed), the spatial position that need to be observed or tracked (see [33] for details).

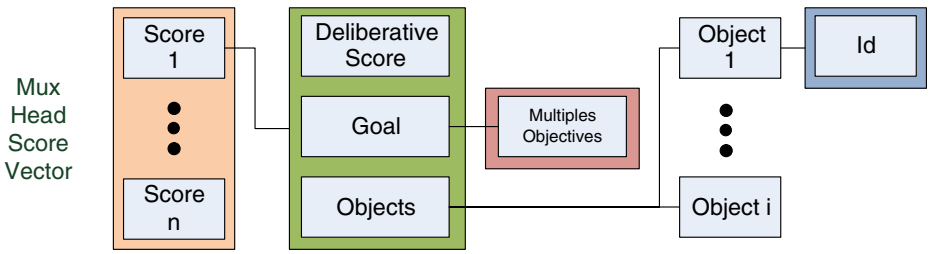

Fig. 3 Data structure for the score vector used for the robot head multiplexor. This data structure is dynamic because its size depends on runtime information added by the deliberative layer. Score information (deliberative score and objects) is defined for each possible goal (see Table 3)

3.6 Active Vision System

The active vision system is an optional module of the architecture that allows adding flexibility in the implementation of complex behaviors. This system tries to reduce the most relevant components of the uncertainty in the world-state, for the execution of the current deliberative behavior that the robot is performing. The world-state, $\mathbf{x}_k$, is the set of relevant variables of the world model (robot localization, other objects positions, etc.) needed for the robot to make decisions. A value function is defined as a function of the world-state ($V_k(\mathbf{x}_k)$), whose value increases with the degree of convenience of the world-state for the execution of the corresponding deliberative behavior.

The system requires the existence of a subset of primitive-behaviors that influences the observations but not the state. This assumption holds in many mobile robots, in which there exist acting primitives, $\mathbf{u}_k^{act}$, which allow moving the robot and therefore directly influencing state $\mathbf{x}_k$ and indirectly observation $\mathbf{z}_k$ through $\mathbf{x}_k$, as well as sensing primitives, $\mathbf{u}_k^{sen}$, that directly influences the observation $\mathbf{z}_k$ but not $\mathbf{x}_k$. For instance, in the case of a typical humanoid robot with a camera mounted on an articulated neck, the camera movements do not influence the world-state; $\mathbf{u}_k^{sen}$ corresponds to orders for the neck joints, and $\mathbf{u}_k^{act}$ corresponds to orders for the rest of the robot body. In terms of the primitive behaviors described in Section 3.5.1, a sensing order $\mathbf{u}_k^{sen}$ can be generated by any primitive behavior associated with the head resource, while an acting order $\mathbf{u}_k^{act}$ can be generated by any primitive behavior associated with the body resource.

In order to select an "optimal" objective to sense with a primitive-behavior $\mathbf{u}_k^{sen^*}$, it is necessary to define an optimality criterion, which will depend on the corresponding deliberative-behavior. If we consider a discrete sensing primitive-behavior space, U^{sen}, then $\mathbf{u}_k^{sen^*}$ can be found by iterating over U^{sen}, and for each $\mathbf{u}_k^{sen}$ calculating the value function by computing a given optimality criterion. As behavior dependent optimality criterion the *minimum expected value variance*, which minimizes the expected variance of the value function after the next sensing primitive-behavior $\mathbf{u}_k^{sen}$, or the *maximum expected action task value*, which maximizes the expected value of the value function after the next acting primitive-behavior $\mathbf{u}_{k+1}^{act}$, can be used (see explanation in [30]).

Thus, the active vision system consists of the following steps:

(a) In time step k, the reactive layer decides the best acting primitive-behavior $\mathbf{u}_k^{act}$, which should maximize the expectation of the corresponding deliberative-behavior value function $E\left(V_k(\mathbf{x}_k)\,\middle|\,\mathbf{u}_k^{act}\right)$. $\mathbf{u}_k^{act}$ is executed, and according to the process dynamics, the world-state moves from $\mathbf{x}_{k-1}$ to $\mathbf{x}_k$.

(b) The world-modeling module makes a prediction (preliminary estimation) about $\mathbf{x}_k$, $\bar{b}_k(\mathbf{x}_k) = p\left(\mathbf{x}_k\,\middle|\,U_{k-1}, Z_{k-1}, \mathbf{u}_k^{act}\right)$.

(c) In the deliberative layer the robot selects the optimal sensing primitive-behavior by evaluating the corresponding value function, computed using the optimality criterion regarding the resulting belief $b_k(\cdot)$. This requires considering all possible objects to be observed, and in each case simulating the execution of the corresponding $\mathbf{u}_k^{sen}$, which also requires simulating the operation of the correction stage of the world-modeling module (simulation of steps (d) and (e)). The information about the selected sensing primitive-behavior is sent to the reactive layer as a deliberative score.

(d) The reactive layer uses the deliberative score, in addition to the reactive score, in order to compute the final score and to activate the final sensing primitive-behavior $\mathbf{u}_k^{see^*}$. Then a sensing action is executed and a new observation $\mathbf{z}_k$ is obtained.
(e) The world-modeling module uses $\mathbf{z}_k$ to update the belief $b_k(\cdot)$ (correction stage of the filter).

In [30] this active vision system is explained in detail.

3.7 Modules Communication and Synchronization

The architecture described above is implemented using the robotics library, UChileLib [31]. This library provides several functionalities regarding computer processes and communications, and it allows the development of modular software. UChileLib makes use of the *Boost* Libraries [35], is OS independent, and can be compiled and executed in Windows or Linux.

In the library, a module is a part of the software with an encapsulated functionality that has a clear and explicit interface with the rest of the software. In order to allow modules to work concurrently, to share memory, and to communicate easily with each other, we implement each of them as a thread. We define a *package* as a group of data that will be transmitted as a whole from one module to another. Every time a module wants to send some information to another module, it generates a package. If the receiving module is ready to receive, the package is sent immediately. If not, the package is stored for a later delivery. When a new package needs to be sent, and there are other packages waiting to be sent, three possible courses of action can be taken, depending on the specific connection: (1) overwriting the existing package and keeping just the newest one, (2) queuing all the packages in order to make sure that all of them will arrive at the listening module, or (3) re-packaging the packages' data so that only one contains the information of all of them. This can be done when the data contained in the packages can be somehow "summed up", for example, the odometry in a self-localization filter.

A module that has incoming and outgoing communication connections has to handle two types of messages coming from other modules: (1) the messages that notify the module that a new package has arrived from an incoming connection, and (2) the messages that notify the module that another module is ready to receive a new package through an outgoing connection. Since the module has only one thread to process these messages, they can be queued. Consequently, each module has a queue of messages to be processed, and it processes them in the order in which they arrive.

In order to make the library multi-platform, and to reuse as much code as possible, the modules that need to communicate with the robot hardware have all their functionality in a platform independent layer. This platform independent layer of modules is implemented as an abstract class, in which the functions that communicate with the hardware are declared but not implemented. Then, using class inheritance, these platform independent modules are instantiated for every required robot platform. Consequently, the platform-specific instances of these modules need to implement only the functions that communicate with the hardware.

4 Validation in Nao Humanoid Robots

4.1 Experimental Setup

One of the main features of robot soccer is that playing soccer is a complex behavior that requires solving several simultaneous problems in real-time (sensor fusion, vision, self-localization, decision making, team coordination and gait control, just to name the most important ones). Thus, robot soccer seems to be an adequate benchmark for a real-time robot architecture as the one being proposed here. In the experimental setup, Nao humanoid robots (produced by Aldebaran Robotics [34]) and the RoboCup SPL (Standard Platform League) rule definition [38] are used. In the performed experiments real and simulated Nao robots, both running the UChileLib control architecture (see Section 3.7), are used.

The Nao robot has 21 degrees of freedom and its main external sensors are two cameras placed on its head. In addition, it has several other sensors such as sonar, pressure sensors, and accelerometers. As its main processor unit, the Nao has an AMD GEODE 500 MHz CPU, with a 256 MB SDRAM. The robot can communicate with other robots or with an external computer using a wireless IP network. The native Nao's OS is an Embedded Linux (32 bit $\times$ 86 ELF) using a custom Open Embedded based distribution.

The *UChile HL-SIM* simulator [29], a 2D simulator developed by our research group, was used as the simulation environment. In this simulator, the sensing data is replaced by simulated data, and the actuation module commands are executed virtually by a virtual robot running in the graphic interface of the simulator. These modules—vision and actuation—have controlled noise, and they are connected with the other modules in the same way as they are connected in the real robots. This allows us to run the hybrid control and world-modeling modules in the simulated environment as if they were running in the real robot. The graphical interface of the simulator and its engine run in a single process that communicates with the robot processes via sockets (allowing the robot clients to run in different computers). By having this parallelism, the simulator is able to take into account the time that the robot expends in processing, thus achieving more realistic experiments. In the simulator, communications are implemented in a slightly different way from the robots to avoid Internet port conflicts when several simulated robots are running in the same computer. The simulator's server centralizes all intra-robot communications, and every simulated robot has a full duplex TCP socket communication with the server. The broadcasts are emulated using the following procedure: the transmitter robot sends the message to the server, and then the server distributes it to the rest of the robots.

The reported experiments validate the ability of the architecture to react to fast changes in the environment while executing expensive behaviors (active vision while playing soccer), and the ability to execute group behaviors (team soccer playing) without disturbing the robot's reactivity.

4.2 Active Vision Experiments in a Simulated Environment

The main goal of these experiments is to validate the ability of the architecture to react to fast changes in the environment, while executing an expensive task in terms

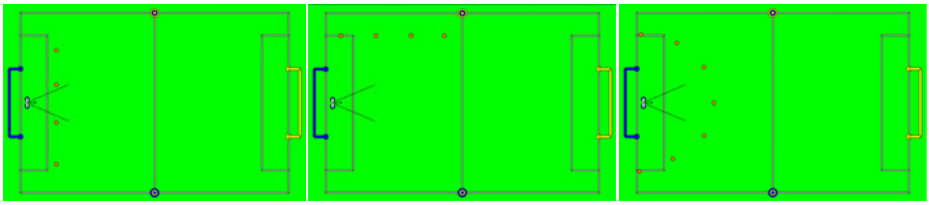

Fig. 4 Ball trajectories used in the goal-covering experiments (*UChile HL-SIM simulator* visualization)

of processing time. The selected task is the execution of an active vision behavior, the "goal-covering behavior".

In this experiment, the goalkeeper intends to cover the goal at all times while minimizing the available shot angles for an attacking player. Hence, the robot needs to localize itself in order to be sure that it covers the goal, while at the same time it looks at the ball for detecting whenever it is coming toward the goal. To accomplish this task, active vision (see description in Section 3.6) is used intensively in order to reduce uncertainty in localization, while estimating the score associated to each object of interest (beacons and the ball), given from the value function. This score, which is computed using the deliberative and the reactive scores (see Section 3.5.2), shows the best object in the field for correcting the pose estimation, while keeping attention on the ball. The experiments are made with a robot executing the goal-covering behavior, and with the ball moving through three different trajectories, as shown in Fig. 4. In the simulator, the ball is moving at a constant speed of 0.2 m/s.

Experiments with the deliberative layer running at three different speeds (0.5, 1, and 2 Hz) were carried out. In these experiments the frequency of the reactive layer varied between 14 and 100 Hz, with a mean value of 57 Hz.[2] Results are shown graphically in Fig. 5. In each case, the deliberative-, reactive- and final-score of each object of interest (ball, goal-beacon 1, and ball-beacon 2), as well as the ball confidence, are shown. It should be remembered that the final objects scores are used to determine the objects to be observed; the object with the highest score is always observed. On the other hand, the confidence values are indications of how good the state of a tracked object is. The higher the confidence of an object, the lower its associated score. Table 5 shows the average percentage of time that the robot decides to observe the ball, and the average percentage of time that the ball is in an unknown position (lost), as a function of the speed of the deliberative layer.

Two main conclusions can be drawn from the experiments. The first one is that the computation of the final score, using the deliberative and reactive scores in the reactive layer, is the feature that enables the system to react to fast changes in the environment. The reactive score allows the system to react quickly, and to handle quick changes in the environment, while the deliberative score handles a long-term goal

[2]The reactive layer always runs at the maximal possible execution rate. Frequency variations are due to variations in process execution conditions, such as image acquisition execution, variations in behavior-tree evaluation conditions, etc.

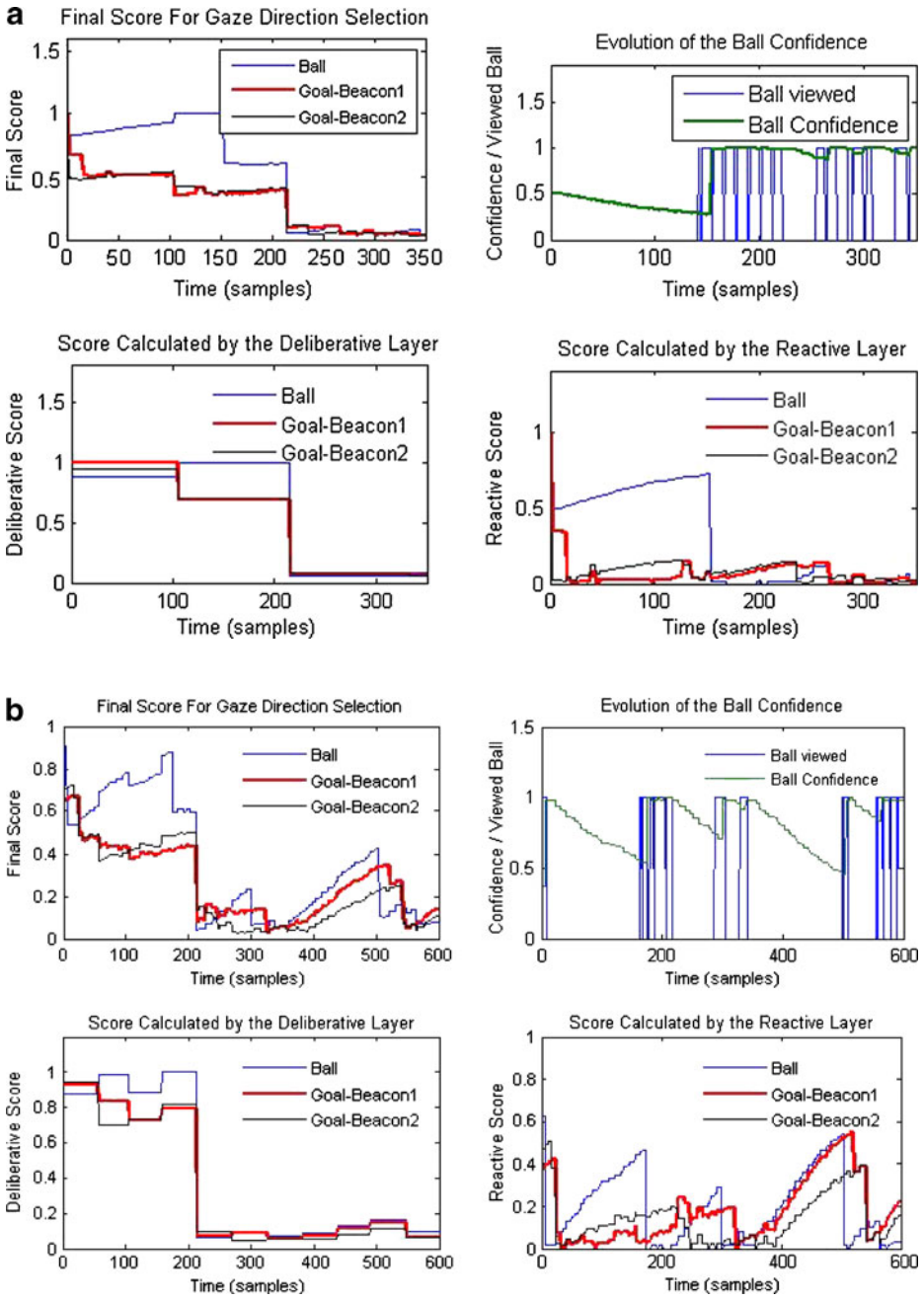

Fig. 5 Goal-covering results for three different situations: Deliberative layer running at frequencies of 0.5 Hz (**a**), 1 Hz (**b**), and 2 Hz (**c**). In all cases the reactive layer is running at a mean frequency of 57 Hz (see main text for details)

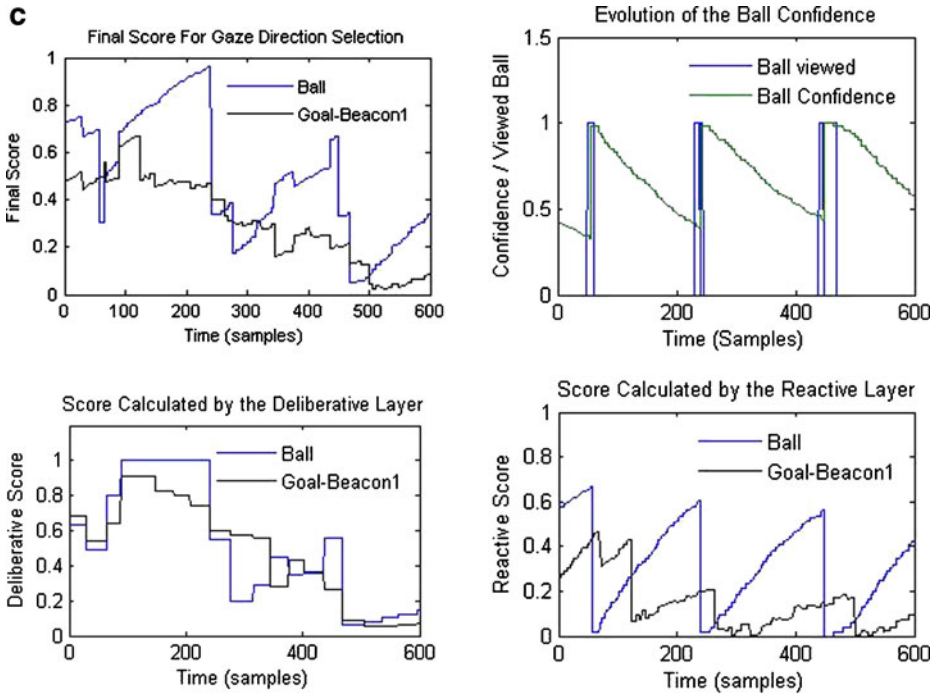

Fig. 5 (continued)

that is needed to accomplish a mission. For instance, in the case of the deliberative layer running at 1 Hz, the ball moves 20 cm between each iteration of the deliberative layer. Making decisions at this speed (1 Hz) would not be enough to track the ball properly. However, as can be seen in Fig. 5b, the system is able to keep track of the ball, and its associated confidence value remains high most of the time.

The second conclusion is that the system needs a critical frequency at which the deliberative score can be refreshed. This can be observed in Fig. 5a, which shows the robot's performance when the deliberative layer runs at 0.5 Hz. In this case the ball moves 40 cm between each iteration of the deliberative layer, and updating the final scores of the objects in the reactive layer at 57 Hz is not enough to keep track of the objects properly. As shown in Table 5, in 34.2% of the cases the ball position is unknown. In fact the ball confidence stays low, even though the robot is trying to look at the ball most of the time (45.2% of the time). Moreover, the robot has localization

Table 5 Statistics about ball perception, while the robot is executing goal-covering behavior

Frequency of the deliberative layer (in Hz)	Average percentage of the time that the robot decides to observe the ball	Average percentage of the time with the ball in an unknown position
0.5	45.2%	34.2%
1	28.5%	19.0%
2	31.5%	10.9%

In all cases the reactive layer is running at a mean frequency of 57 Hz

Table 6 Reactive and deliberative layers frequency for different missions and roles, in real-world experiments with a Nao Humanoid robot

Mission/role	Layer	Max frequency Hz	Average frequency Hz	Min frequency Hz
Defend the goal/ goalkeeper	Reactive	69	54	42
	Deliberative	1.91	1.6	1.5
Attack/striker	Reactive	70	58	47
	Deliberative	1.9	1.71	1.6

problems because most of the time it is trying to observe the ball, but not the goal-beacons. With the deliberative layer running at a high speed, the robot switches the target of the gaze to different objectives more precisely than when it runs at a low rate, due to the fact that in cases when the environments present large changes, deliberative goal selection becomes important. As can be observed in Table 5, when the speed of the deliberative layer increases, both the percentage of time that the robot decides to observe the ball and the percentage of time that the ball is in an unknown position, decrease.[3]

4.3 Active Vision Experiments in a Real Environment

The main goal of these experiments is to validate the usefulness of the proposed architecture when controlling a humanoid robot in the real-world, and its ability to react to fast changes in the environment, while executing an expensive task in terms of processing time.

To accomplish this, the architecture was tested in extensive experiments in which a robot was playing soccer for several minutes. In these experiments the robot executed several different missions in different roles (see Table 1), and it used different deliberative behaviors (see Table 2), some of them including active vision mechanisms. In all cases the robot behaved properly, and the use of active vision behaviors did not affect the robot's responsivity and reactivity.

In Table 6 the average frequency of the reactive and deliberative layers when executing two different missions and roles are shown. These mean values were measured in several game situations, during more than 60 min of operation. As can be observed the average frequency of the deliberative layer is ∼1.65 Hz, while the average frequency of the reactive layer is ∼56 Hz.

The robot reactivity was measured by letting the robot fall while executing the goal-covering behavior. After falling, the robot detected the fall in just 51 ms, and after some seconds it was already back in a proper playing position (see Fig. 6). The standup sequence takes several seconds because of mechanical and electronic limitations of the robots. Even though the deliberative layer is executed every 600 ms, the robot detects the fall in just 51 ms, because the reactive layer executes every 18 ms. The fall is detected as soon as the reactive layer receives and processes the accelerometers and gyroscope readings.

[3]Note for the reviewers. A second set of experiments was carried out for the case of a striker player executing an active vision behavior. In these experiments similar results as the ones of the goal-covering behavior were obtained Due to space limitations, they are not included in the paper. However, they can be incorporated upon request.

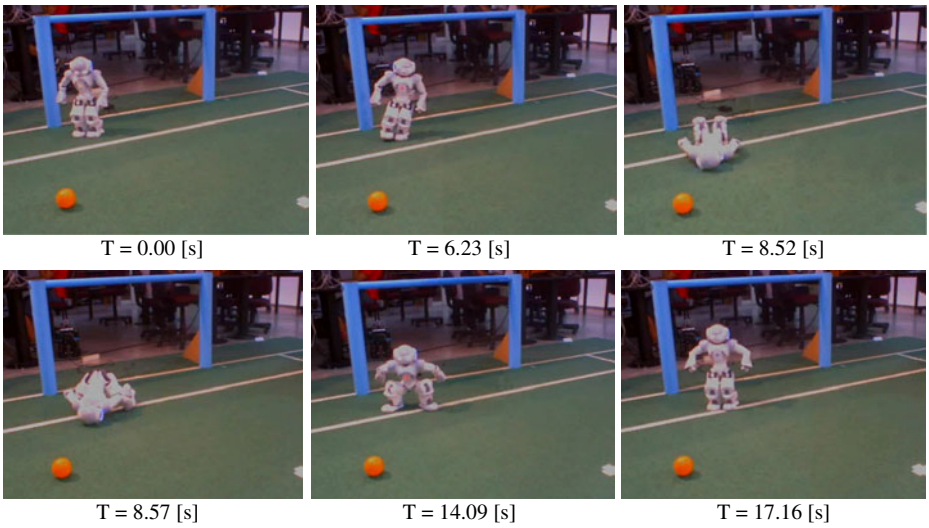

Fig. 6 Exemplar situation of a robot fall and recovery while executing the goal-covering behavior. In T = 8.52 the robot falls. In T = 8.57 the robot detects the fall and starts the recovering process. In T = 14.09 the robot is again in a proper playing position. The standup sequence takes several seconds because of mechanical and electronic limitations of the robot

4.4 Group Behaviors Execution While Playing Soccer in Real Competitions

The ability of a team of Nao humanoid robots to play soccer properly and to execute group behaviors while using the proposed architecture was validated in three testing environments. In the first environment, the architecture was tuned and tested using the UChile HL-SIM simulator. Several complete games between teams of four Nao robots were successfully simulated. In these games robots behaved properly, and they were able to execute complex group's behaviors.

In the second set of experiments, the architecture was tested in the SPL soccer robotics field of our laboratory (see Fig. 6). In this case several games between teams of two robots were carried out. In the experiments, the humanoid robot players were able to carry out group's behaviors and act very reactively at the same time.

In the third set of experiments, the architecture was validated in soccer games against other teams during the RoboCup 2010 world-championship (Singapur, July 2010) [37]. In this competition our UChile team built a joint team, the CHITA team, together with the SPQR team from the Sapienza Università di Roma. During games robots of each sub-team used their own control code. Data communication between robots of both sub-teams was implemented with the common communication protocol described in Section 3.7. In the four games that the joint CHITA team played, a good performance of the UChile robots was observed, due to the use of the proposed control architecture. In particular:

The robots of the UChile sub-team kicked three goals during the competition—one against Nao Devils (Germany), one against Kouretes (Grece), one against Les 3 Mousquetaires (France) - [37]. In addition, robust behaviors were observed; the robots kicked to the opposite goal every time they could, and when they fell down

they stood up very fast in a very reactive way. These observations demonstrated that the architecture allowed good coordination between the internal robot' control-processes, allowing robots taking the right decisions at the right time. The architecture worked in a reactive way when it was required.

The data communication with the robots of the SPQR sub-team worked correctly and allowed the implementation of team behaviors. Using the communication protocol, each robot transmitted a flag indicating when the ball was observed, and the distance to the ball. The planning layer of each robot received this information, and used it in order to determine the mission and role of each robot. For example, if two robots observed the ball at approximately the same distance, then both robots set up the *attack* mission and the *striker* role, and they went to the ball, at the same time. In case one robot observed the ball at a shorter distance than the other robot, then this robot set up the *attack mission* and the *striker* role and it went to the ball. The other robot set up the *attack* mission and the *striker cover* role and kept a fixed distance form the *striker*, avoiding disturbing it. The observed correct execution of team behaviors implies that all layers of the proposed architecture, as well as robot's communication, were working coordinately. In addition, thanks to our modular architecture, implementing communication and coordination of team behaviors among robots running different control codes was straightforward.

One of the main reasons for not having a better performance during the RoboCup 2010 competition was the fact that our robots have a slower gait than robots from other teams. This is an aspect that is being improved.

5 Conclusions

In this paper, a real-time hybrid control architecture for biped humanoid robots has been proposed. In this architecture the main robot's functionalities are organized in four parallel modules: perception, actuation, world-modeling, and hybrid control. The hybrid control module is divided into three different layers with different response speed and planning capabilities: the planning layer, the deliberative layer and the reactive layer.

The architecture was validated using Nao humanoids in simulated and real environments. The experimental results have shown the feasibility of having active vision mechanisms that do not disturb the robot reactivity, and the straightforward management of the robot's resources using resource multiplexers. The control architecture was able to maintain real-time responsiveness in the reactive layer of a real humanoid robot, while the deliberative layer developed more complex behaviors at lower frequencies. In a real-game situation, the robot was able to detect a fall in just 51 ms, even though it was executing an active vision behavior (goal-covering). In addition, the ability to execute group behaviors in real-time was tested in international robot soccer competitions. In this application, the use of the architecture allowed implementing communication and coordination of team behaviors among robots running different control codes.

As future work we would like to extend the use of this control architecture to other applications (autonomous vehicles and service robot platforms), and to include learning capabilities in it. For instance, we would like to use reinforced learning in order to learn the value functions.

Acknowledgement This research was partially funded by FONDECYT (CHILE) under Project Number 1090250.

References

1. Arkin, R.: Behavior-Based Robotics. MIT Press (1998)
2. Arkin, R., Fujita, M., Takagi, T., Hasegawa, R.: An ethological and emotional basis for human–robot interaction. Robot. Auton. Syst. **42**(3–4), (2003)
3. Brooks, R.: A robust layered control system for a mobile robot. IEEE J. Robot. Autom. **2**(1) (1986)
4. Jaeger, H., Christaller, T.: Dual dynamics: designing behavior systems for autonomous robots. In: Artificial Life and Robotic, vol. 2(3), pp. 108–112. Springer, Japan (1998)
5. Hertzberg, J., Jaeger, H., Zimmer, U., Morignot, P.: A framework for plan execution in behavior-based robots. Proceedings, pp. 8–13 (1998)
6. Behnke, S., Frötschl, B., Rojas, R.: Using hierarchical dynamical systems to control reactive behavior. In: Lecture Notes in Computer Science, pp. 186–195 (2000)
7. Behnke, S., Stückler, J., Strasdat, H., Schreiber, M.: Hierarchical reactive control for soccer playing humanoid robots. International Journal of Humanoid Robotic **5**(3), 375–396 (2008)
8. Hurdus J., et al.: Victor tango architecture for autonomous navigation in the Darpa Challenge. J. Aero. Comput. Inform. Commun. **5**(12), 1542–9423 (2008)
9. Berger, R., Burkhard, H.: At Humboldt—team description 2007. In: Lecture Notes in Computer Science (RoboCup 2007) (2008)
10. Chernova, S., Arkin, R.: From deliberative to routine behaviors: a cognitively inspired action-selection mechanism for routine behavior capture. Adapt. Behav. **15**(2), 199–216 (2007)
11. Huntsberger, T., Pirjanian, P., Trebi-Ollennu, A., Aghazarian, H., Das, H., Joshi, S., Schenker, P.: CAMPOUT: a control architecture for tightly coupled coordination of multirobot system for planetary surface exploration. In: Proc. SPIE Conf. Sensor Fusion and Decentralized Control in Robotic System III (2000)
12. Christensen, H., Pirjanian, P.: Theoretical methods for planning and control in mobile robotics. In: IEEE Conf. on Knowledge-Based Intelligent Electronic System (1997)
13. Maes, P.: Modelling adaptive autonomous agents. J. Artif. Life **1**, 135–162 (1994)
14. Hoshino, Y., Takagi, T., Di Profio, U., Fujita, M.: Behavior description and control using behavior module for personal robot. In: Proc. IEEE Conf. Robotic and Automation (2004)
15. Sawada, T., Takagi, T., Fujita, M.: Behavior selection and motion modulation in emotionally grounded architecture for QRIO SDR-4X II. In: Proc. IEEE Conf. on Intelligent Robots and Systems (2004)
16. Steinbahuer, G., Wotawa, F.: Enhacing plan execution in dynamics domains using model based reasoning. In: Lecture Notes in Computer Science 5314, pp. 510–519 (2008)
17. Gat, E.: On Three Layer Architectures. Artificial Intelligence and Mobile Robots, pp. 195–210. AAIA Press (1997)
18. Matarić, M.: Behavior-based control: examples from navigation, learning and group behavior. J. Exp. Theor. Artif. Intell. **9**(2–3) (1997)
19. Lencer, S., Bruce, J., Veloso, M.: A Modular Hierarchical Behavior-based Architecture. Lecture Notes in Computer Science 2377 (2002)
20. Proetzsch, M., Luksch, T., Berns, K.: Development of complex robotic systems using the behavior-based control architecture iB2C. Robot. Auton. Syst. **58**, 46–67 (2010)
21. Friedmann, M., Kiener, J., Petters, S., Thomas, D., von Stryk, O., Sakamoto, H.: Versatile, high-quality motions and behavior control of humanoid soccer robots. In: Proc. IEEE Workshop on Humanoid Soccer Robot, pp. 9–16 (2006)
22. Ferrein, A., Potgieter, A., Steinbahuer, G.: Self-aware robots- what do we need from learning, deliberative and reactive control? In: Proc. Workshop on Hybrid Control of Autonomous Systems, pp. 1–5 (2009)
23. Berger, M., Endert, H., Joecks, S.: Combining learning, deliberation and reactive control in simulated soccer: DAInamite framework. In: Proc. Workshop on Hybrid Control of Autonomous Systems, pp. 57–62 (2009)

24. Korterkamp, D., Simmons, R.: Robotic systems architectures and programming, chapter A.8. In: Siciliano, B., Khatib, O. (eds.) Springer Handbook of Robotics, pp. 187–206. Springer (2008)
25. Mataric, M., Michaud, F.: Behavior-based systems, chapter E.38. In: Siciliano, B., Khatib, O. (eds.) Springer Handbook of Robotics, pp. 891–909. Springer (2008)
26. Bonasso, R.P., Kortenkamp, D., Miller, D.P., Slack, M.: Experiences with an architecture for intelligent, reactive agents. J. Exp. Theor. Artif. Intell. **9**, 237–256 (1995)
27. Müller, M.: Hierarchical activation spreading: a design pattern for action selection. In: Proc. Workshop on Hybrid Control of Autonomous Systems, pp. 63–70 (2009)
28. Guerrero, P., Ruiz-del-Solar, J.: Improving robot self-localization using landmarks' poses tracking and odometry error compensation. In: Lecture Notes in Computer Science 5001 (RoboCup 2007), pp. 148–158 (2008)
29. Guerrero, P., Ruiz-del-Solar, J., Díaz, G.: Probabilistic decision making in robot soccer. In: Lecture Notes in Computer Science 5001 (RoboCup 2007), pp. 29–40 (2008)
30. Guerrero, P., Ruiz-del-Solar, J., Romero, M., Angulo, S.: Task oriented probabilistic active vision. Int. Journal of Humanoid Robotics **7**(3), 451–476 (2010)
31. Ruiz-del-Solar, J., Guerrero, P., Palma-Amestoy, R., Marchant, R., Yañez, J.M.: UChile Kiltros 2009 Team description paper. In: RoboCup Symposium 2009, 29 June–5 July, Graz, Austria. CD Proceedings (2009)
32. Stroupe, A., Matrin, M., Balch, T.: Distributed sensor fusion for object position estimation by multi-robot systems. In: Proc. ICRA Conf. 2, pp. 1092–1098 (2001)
33. Schulz, R.: Active Vision in Anthropomorphic Humanoid Robots. Electrical Engineering Thesis, Universidad de Chile (2010) (in Spanish)
34. Aldebaran Robotics Official Website: http://www.aldebaran-robotics.com/. Accessed 19 December 2010
35. Boost Library Official Website: http://www.boost.org/. Accessed 19 December 2010
36. RoboCup Official Website: http://www.robocup.org/. Accessed 19 December 2010
37. RoboCup 2010 Official Website: http://www.robocup2010.org/. Accessed 19 December 2010
38. RoboCu SPL League Official Website: http://www.tzi.de/spl/bin/view/Website/WebHome. Accessed 19 December 2010
39. Darpa Challenge Official Website: http://www.darpa.mil/grandchallenge/index.asp. Accessed 19 December 2010