UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS FISICAS Y MATEMATICAS

DEPARTAMENTO DE INGENIERIA ELECTRICA

CONTROL ACTIVO Y DIFERENCIAL EN EL TIEMPO DE ROBOTS HUMANOIDES

TESIS PARA OPTAR AL GRADO DE MAGISTER EN CIENCIAS DE LA INGENIERIA MENCION INGENIERIA ELECTRICA

MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL ELECTRICISTA

JAVIER ENRIQUE TESTART PACHECO

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
PAUL VALLEJOS SÁNCHEZ

ALVARO SOTO ARRIAZA

SANTIAGO DE CHILE

MAYO 2011

RESUMEN DE LA MEMORIA
PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERIA
MENCION INGENIERIA ELECTRICA
Y AL TITULO DE
INGENIERO CIVIL ELECTRICISTA
POR: JAVIER TESTART PACHECO
FECHA: MAYO DE 2011
PROF. GUIA: SR. JAVIER RUIZ DEL SOLAR

"CONTROL ACTIVO Y DIFERENCIAL EN EL TIEMPO DE ROBOTS HUMANOIDES"

El objetivo de esta tesis es el desarrollo e implementación de una arquitectura híbrida de control para robots bípedos humanoides. Una arquitectura híbrida es aquella que permite tener diferentes niveles de control que funcionan en paralelo, cada uno de los cuales se evalúa a diferente frecuencia con diferente información sensorial. La arquitectura implementada se estructura en tres niveles. La meta es lograr que los niveles superiores tengan información de más alto nivel que se puede obtener a una frecuencia más baja, a diferencia de lo que ocurre en los niveles inferiores, en donde la información sensorial es de menor calidad pero obtenida a una frecuencia mayor. Esto permite jerarquizar el desarrollo de los comportamientos según la calidad de información que requieren y su frecuencia de actualización.

Los principales aportes de esta tesis son la implementación de una estructura híbrida en tres niveles: reactivo, deliberativo y planificación; que permiten manejar comportamientos individuales o grupales de robots, que logra operar con múltiples objetivos en los diferentes niveles de la toma de decisiones. Esta arquitectura se diseña de tal forma que los comportamientos grupales, definidos en el nivel de planificación, configuran misiones para los comportamientos del nivel deliberativo, los cuales a su vez definen objetivos para los del nivel reactivo, los que finalmente seleccionan las acciones a ejecutar. El manejo de objetivos múltiples se basa en una evaluación de los diferentes objetivos a lograr en el nivel deliberativo, generando un puntaje por cada uno de ellos, los que son actualizados en el nivel reactivo para que reflejen los cambios en el entorno del robot.

Para evaluar el funcionamiento de la arquitectura diseñada se realizaron diferentes pruebas, tanto en simulador -para así disminuir los daños- como en los robots reales. El objetivo de éstas fue verificar que el diseño cumplía con los requerimientos de las diferentes misiones, sin mermar la reactividad del robot en ningún instante. En efecto, en los experimentos efectuados con los robots reales, la efectividad del robot en marcar goles fue sobre 80% para las misiones de atacante demostrando gran reactividad y precisión. Además en cuanto al manejo de múltiples objetivos, se destaca que en la misión de arquero el robot pudo realizar visión activa sin perder la reactividad lo que permite al robot detectar las caídas en 57 [ms]. Adicionalmente, se lograron generar comportamientos grupales y desarrollar estrategias conjuntas, las que no afectaron las misiones individuales de los robots cuando hubo problemas en la comunicación de datos. Estos resultados demuestran las ventajas de la estructura híbrida en cuanto al manejo de los recursos computacionales del robot gracias a la ejecución de los comportamientos en diferentes niveles permitiendo a éste cumplir diferentes misiones sin afectar su capacidad de reacción.

Finalmente, cabe destacar que esta arquitectura fue exitosamente usada para competir en competencias internacionales de robótica RoboCup 2010.

AGRADECIMIENTOS

El presente trabajo fue financiado parcialmente por el proyecto FONDECYT 1061158.

TABLA DE CONTENIDOS

I. INTRODUCIÓN

1.1 ANTECEDENTES

El modo en que un robot toma una determinada decisión con la que luego ejecuta una acción, dada la información que posee (llámese estrategia, toma de decisiones, etc.) se ha solucionado creando modelos basados en la etimología (ciencia que estudia el comportamiento animal [2], así como también desde la teoría de control (se fija un *set point* para las variables controladas del proceso y a través las variables manipuladas del proceso, los actuadores actúan sobre éste para lograr que llegue al *set point*) [9][22][26]. Ambas visiones del tema son muy diferentes y tienen diferentes implicancias: la primera es más fácil de implementar, sin embargo, no siempre se logra una ejecución eficiente por parte del robot; mientras que la segunda es difícil de implementar para situaciones complejas (por ejemplo, que un robot haga más que avanzar hacia un punto) pero se sabe que si se logra realizar, se puede crear un controlador óptimo.

El desafío que se presenta es el cómo lograr que el robot realice una misión teniendo un modelo impreciso del mundo que lo rodea, dado únicamente por la información obtenida desde sus sensores. Este modelo impreciso se explica porque los sensores que permiten descubrir el medio contienen ruido y además éste es dinámico. El ruido se define como los errores asociados a las mediciones realizadas en el ambiente [9]. Además, el robot se mueve gracias a actuadores, los que según el tipo específico de robot tienen tiempos de reacción y latencia, por lo que la toma de decisiones del robot debe estar coordinar al momento de ejecutar los comandos de éstos. En definitiva, las decisiones del robot deben tener en consideración esta coordinación al momento de seleccionar entre qué acciones son factibles de ejecutar y escoger cuál es la acción óptima entre las factibles a realizar en ese instante. Dado lo anterior, con el objeto de efectuar una misión de manera exitosa, el robot debe procesar conjuntamente el modelo parcial entregado por los sensores con la coordinación de los actuadores que posee. Por último, a esto se le debe agregar la dinámica del mundo en que se mueve el robot, ya

que los objetos pueden ser no estáticos, lo cual agrega otra arista al desafío planteado para ejecutar una misión en forma exitosa.

Otro aspecto a considerar en un robot, es que posee una cantidad limitada de sensores para realizar su exploración del mundo; de los cuales puede tan solo obtener una reducida cantidad de información de los objetos. Por lo tanto, si se quiere acrecentar la frecuencia de las detecciones de los objetos y, por consiguiente, aumentar la información disponible del mundo, es necesario estar constantemente indagando en los alrededores.

Existen dos enfoques frente a esta disyuntiva: uno pasivo y otro activo. El pasivo es el enfoque en el cual no se toma ninguna política especial respecto de dónde dirigir un sensor y los descubrimientos que efectúa el robot son fruto del objeto que se cruzó en el área de detección del sensor. El enfoque activo, en cambio, trata de ser lo contrario, llevando el área de detección del sensor a una posición en la cual tenga cierta probabilidad de percibir alguno de los objetos detectables de interés [20][21].

1.1.1 Fútbol Robótico

Dentro del campo de la robótica móvil existe un área denominada fútbol robótico, la cual consiste en competencias de este conocido deporte entre robots. Este ambiente es muy propicio para el desarrollo de esta investigación, ya que es un mundo de gran dinamismo que se ejecuta en un espacio limitado. El objetivo de esta competencia es desarrollar equipos de robots que hagan más goles al rival. El desafío consiste en poder combinar una buena capacidad de reacción, ya que la pelota está en constante movimiento, con una estrategia de juego que lleve al equipo a ganar. Existen diferentes ligas en las cuales los robots deben jugar autónomamente fútbol, según reglas especificadas por cada liga [46][47][48][49]. El gran objetivo que se ha propuesto el fútbol robótico es el siguiente: “hacia mediados del siglo 21, un equipo de robots humanoides autónomos ganará el partido, siguiendo las reglas oficiales establecidas por la FIFA, contra la selección que haya ganado el reciente mundial” [45].

1.1.2 Implementación Existente

Hoy en día, los robots ya tienen un software en funcionamiento que les permite desempeñarse y realizar algunas tareas. La implementación actual se encuentra explicada en detalle en la sección 3.1.

1.2 HIPÓTESIS

La hipótesis bajo la cual se plantea esta tesis es que las estructuras híbridas son las adecuadas para resolver de manera eficiente y escalable la toma de decisiones de un robot. En efecto se propone crear una arquitectura híbrida para robots humanoides futbolista.

Una segunda hipótesis planteada para esta tesis es que el uso de la arquitectura híbrida permite que los comportamientos logren sus objetivos de mejor forma al ejecutarlos en diferentes niveles.

1.3 OBJETIVOS

El objetivo de esta tesis es -a partir de los códigos existentes creados por el equipo de fútbol robótico de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile- mejorar las estrategias utilizada por los robots a través del desarrollo de un nuevo modelo de toma de decisiones, el cual se implementará introduciendo cambios en la arquitectura actual. La idea es que este nuevo modelo entregue mayor confiabilidad en la toma de decisiones tomando en cuenta las estimaciones en las detecciones de los sensores, con el fin que éstas sean simultáneamente coordinadas con las acciones que se irán ejecutando, para así no perder reactividad. Para esto, se propone una arquitectura híbrida de estrategia en tres niveles: reactivo, deliberativo y de planificación, en la que cada uno ejecute acciones de distintas fases de abstracción que lleven al robot a cumplir la misión que se requiere. Además, se plantea como objetivo central de la tesis el funcionamiento del sistema propuesto en los robots reales, con el fin de poder ser usada durante las competencias de *RoboCup* por el equipo de fútbol robótico de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

Otro objetivo planteado es la obtención de un control activo de sensores. Con esto, se refiere a tener un manejo del sensor de manera tal, que se posea una mayor frecuencia

de detecciones de objetos, y por lo tanto, el autómata obtendrá un incremento en la cantidad de información del mundo, tomando -en consecuencia- mejores decisiones.

Por último, esta tesis se desarrollará en la librería UChile-Lib, con la finalidad de quedar disponible para uso por parte de los equipos de fútbol robótico, además de crear una nueva línea de investigación en los modelos decisionales de los robots. En concordancia con lo anterior, la arquitectura diseñada para la toma de decisiones de robots fue aceptada para su publicación en una revista acreditada (ISI) durante 2011 [42].

1.4 METODOLOGÍA

Para realizar esta tesis, es necesario desarrollar -en conjunto con el equipo de fútbol robótico- un simulador de alto nivel denominado HLSIM (High-Level Simulator) con la finalidad de poder realizar experimentos simulados y disminuir el daño en los robots. HLSIM permite la simulación de las detecciones del robot y de la ejecución de sus acciones. Se plantea usar la siguiente metodología para el desarrollo de esta tesis:

- Realizar toda la programación de la estructura hibrida usando el simulador. Esto con el fin de disminuir el daño en los robots usados y disminuir el tiempo de desarrollo ya que el simulador permite detectar errores de software más fácilmente que el robot.
- Pruebas en el simulador. Una vez testeada la nueva arquitectura en el simulador se pasará a realizar la programación de los comportamientos específicos en el simulador. Con esto se pretende realizar pruebas extensivas de los comportamientos, además de poder verificar su funcionamiento en situaciones especiales.
- Pruebas en el robot verdadero. Las pruebas en el robot verdadero son necesarias ya que si bien el simulador ayuda a programar los comportamientos, se requieren realizar ajustes a éstos para que logren funcionar correctamente en el robot.

En relación al funcionamiento del manejo activo del sensor, la metodología de verificación que se plantea es a través de dos mediciones: por una parte, medir la

reactividad del robot con el enfoque activo, y observando si éste mejora el trabajo previo realizado en esta materia por el equipo y, por otra parte, comprobar que el robot logre ejecutar la misión deseada satisfactoriamente. Para esto se usara la misma metodología propuesta pero probar específicamente este aspecto.

1.5 Aportes de la Tesis

Las estructuras híbridas en tres capas han sido implementadas por varios grupos de investigación [4][5][6][8][10][11][13][15][27] y el primer aporte de esta tesis es generar una arquitectura hibrida para el manejo de la estrategia de control de los robots humanoides de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. Esta arquitectura está diseñada para manejar robots individualmente como también grupos de robots. El segundo aporte de esta tesis es el manejo de múltiples objetivos en diferentes niveles de tomas de decisiones del modelo hibrido, permitiendo hacer uso de las características de este modelo. Esto se logra con la continuación en el uso de trabajos previos [18][19][20][21], pero ahora usando la estructura hibrida a crear.

1.6 Estructura de la Tesis

Esta tesis comienza con el capítulo II “Revisión Bibliográfica” donde se analizarán trabajos que se enfocan en las metodologías actualmente utilizadas para resolver los problemas que plantea la toma de decisiones de robots: teoría de comportamientos, el manejo de estructuras de programación eficientes, el manejo de información de distintos niveles de confiabilidad y diversas políticas de manejo de sensores.

A continuación, en el capítulo III “Arquitectura Híbrida Propuesta” se expondrá el sistema propuesto para la toma de decisiones y de control activo, para el caso de robots humanoides futbolistas.

Luego, en el capítulo IV “Experimentos y Resultados”, se describirán las pruebas realizadas en el robot, los experimentos efectuados en el simulador y los resultados obtenidos al medir el desempeño del sistema propuesto.

Finalmente, en el capítulo V “Conclusiones”, se presentarán las conclusiones obtenidas en el desarrollo de este trabajo de tesis.

II. REVISIÓN BIBLIOGRÁFICA

A continuación se exponen los principales trabajos relevantes analizados. A partir de los conocimientos adquiridos y la experiencia señalada por los distintos autores se construye el modelo desarrollado en esta tesis.

2.1 Estructuras de Construcción de Arquitecturas de Toma de Decisiones en Robots

Algunos modelos de inteligencia artificial se fundan en la observación de la naturaleza y el comportamiento de los seres vivos. En efecto, la conducta de animales y humanos puede aportar bastante, ya que en éstos se resuelve el problema de elegir qué acción realizar para lograr un objetivo. Por otro lado, esta materia tiene una larga data de estudio, tanto en el campo de la biología como en el de la sicología, por lo que son un buen punto de partida al momento de realizar un modelo. Esto último, dado que los robots pueden -cada vez menos- tener rutinas de ejecución pre-programadas, buscándose que éstos desarrollen más autonomía en su funcionamiento. Lo anterior lleva a plantearse la interrogante respecto a qué se considera inteligencia para un robot móvil, concepto que puede ser considerado, en este campo, como la habilidad para actuar apropiadamente y producir un cambio en un ambiente incierto [1]. Las habilidades necesarias para lograr ese objetivo son numerosas, pero se ha llegado a un consenso en robótica, en el cual se consideran tres bases para resumir las habilidades imprescindibles para esto: *sensar*, *planear* y *actuar*. Las definiciones de estas bases llevan a crear dos corrientes teóricas en robótica: el *Paradigma Deliberativo* y el *Paradigma Reactivo*, los cuales son explicados en los siguientes párrafos [2].

2.1.1 Paradigma Deliberativo

Los primeros desarrollos en robótica usaron este tipo de arquitectura para funcionar. Este paradigma consiste en sensar el mundo, lo que implica la obtención de toda la información disponible para luego planear una trayectoria o un set de acciones, y

enseguida ejecutarla, esperando lograr el objetivo planteado. Después de un intervalo de tiempo, este ciclo se vuelve a ejecutar, repitiéndose la secuencia *sensar*, *planear* y *actuar*, esto se observa en la figura 1. Es importante recalcar que -en estas estructuras- toda la información del mundo es fusionada en una estructura global de datos o modelo del mundo. En cada iteración del ciclo, el modelo del mundo es actualizado y un nuevo plan de acción es resuelto para volver a ejecutarse. Esta arquitectura se desarrolló principalmente pensando en que el robot pudiera navegar en ambientes cerrados, sin importar su velocidad de movimiento [2].

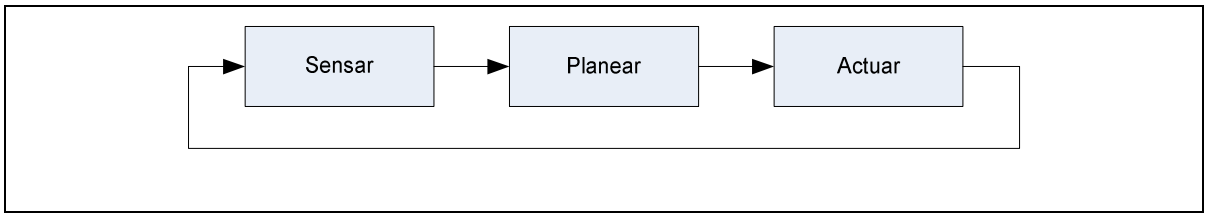


Figura 1: Organización de las estructuras deliberativas

Por definición, esta estructura usa la planificación, o un algoritmo para analizar toda la información disponible del medio, para luego buscar deducir el mejor set de acciones durante el futuro periodo. En este enfoque, la velocidad de respuesta del robot es dependiente de la planificación que se realiza; mientras más exhaustivo y complejo es el algoritmo de planificación el robot reacciona más lentamente.

Este paradigma tiene la ventaja de permitir que el robot pueda navegar o realizar tareas de modo eficaz. El problema que enfrenta es qué hacer si el mundo cambia mientras el robot está planificando las próximas acciones en base a las mediciones que tiene. Con esta arquitectura no queda claro cómo resolver esa dificultad que puede ser tan simple como evitar un obstáculo que se movió en la trayectoria planeada. Por lo tanto, este enfoque funciona muy bien si antes de cada ciclo se puede obtener toda la información disponible y que nada cambie mientras se ejecuta el plan. Esto resulta en una complicación en la implementación de estas estructuras en la medida que se le pide a los robots llevar a cabo misiones en ambientes dinámicos y no solo estáticos, como los laboratorios.

La robótica móvil, vista solo desde este paradigma, no permite lo que se busca hoy en día en los robots, que es, salir de los laboratorios y desenvolverse en ambientes dinámicos.

2.1.2 Paradigma Reactivo

Aspirar a tener un modelo acabado del mundo no siempre es posible en este campo, ya que depende de sensores limitados. A raíz de este punto, y de la observación de la naturaleza, se inició un razonamiento sobre un nuevo paradigma, el paradigma reactivo, en el cual se toma como primicia que el robot es parte del medio, por lo que él también afecta a éste; y si está bien diseñado, el robot debe poder manejar las oportunidades y problemas que puedan aparecer en el entorno [2]. Con este enfoque se puede comenzar a pensar en comportamientos ligados a percepciones que llevan a acciones distintas como se observa en la figura 2.

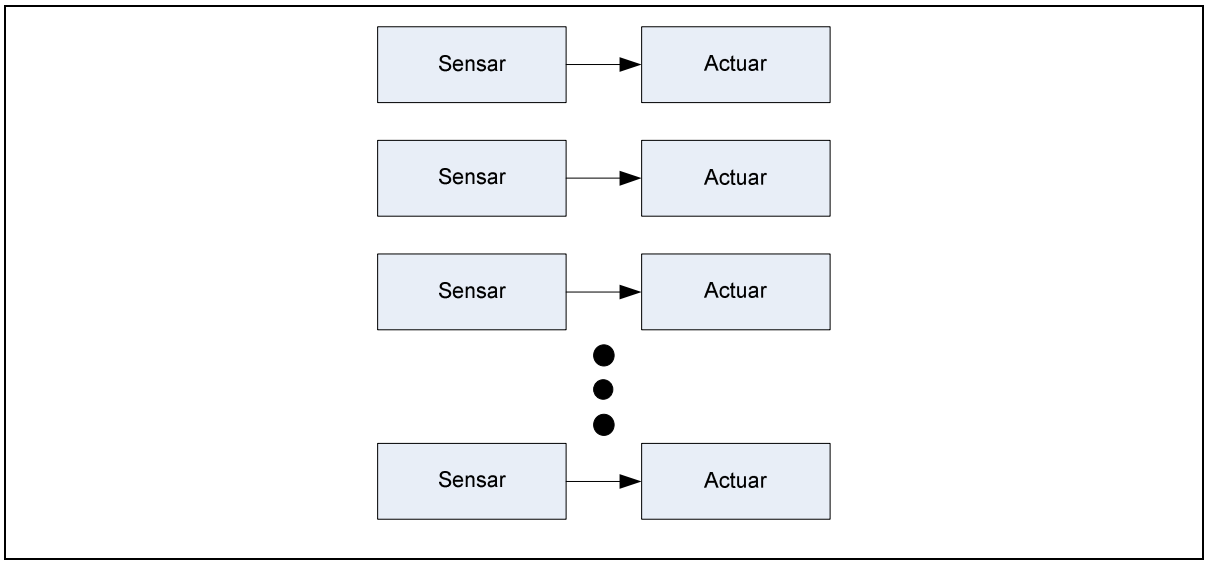


Figura 2: Organización de estructuras reactivas

Desde este punto de vista, la misión de los comportamientos es ser una función de transferencia, de manera de mapear las percepciones del robot en los actuadores. El control de esta arquitectura queda en manos del funcionamiento concurrente de los comportamientos del robot. En este esquema la información sensorial es específica de cada comportamiento; por ejemplo, si se usa en el robot una cámara como sensor, a cada comportamiento le llega una información procesada de manera diferente. Esto lleva a una gran diferencia con el paradigma anterior, dado que en

éste no se requiere una información global del mundo y, por lo tanto, permite que los robots puedan funcionar no solo en ambientes controlados. Cabe recalcar que en este enfoque los un solo comportamiento selecciona la acción a efectuar.

Este enfoque tiene la ventaja de permitir la construcción de los comportamientos de manera similar a la biológica. Cuando un comportamiento falla, otro toma el control y -de este modo- el robot no falla en lograr su objetivo, a diferencia de lo que ocurre con el otro paradigma. La gran ventaja de estos sistemas es su velocidad de reacción debido al funcionamiento de algunos comportamientos en paralelo, eliminando el "cuello de botella" que se tiene con la aproximación deliberativa. Por ejemplo, fácilmente es posible hacer navegar un robot tipo *khepera* en una pieza sin que este choque con obstáculos usando algoritmos de muy baja complejidad. Estos algoritmos con planteamiento reactivo suelen no tener una trayectoria clara y terminan navegando aleatoriamente evadiendo obstáculos.

Si bien esta manera de organizar el control del robot resulta más fácil, eficaz y robusta -comparada con la aproximación anterior-, ella es mucho menos precisa. Es complejo determinar en esta arquitectura qué cantidad de comportamientos son necesarios para poder obtener uno de mayor complejidad y, por otro lado, de no ser efectuadas cuidadosamente tales conductas, el robot puede llegar a tornarse incontrolable. Tampoco se puede demostrar matemáticamente que los comportamientos son los correctos para una determinada tarea. Por otro lado, cabe recalcar que esta arquitectura, al plantear la inexistencia de un modelo del mundo, implicaría la ausencia de memoria y en consecuencia la capacidad de razonar acerca de lo que ocurre alrededor, no permitiéndose componente alguno de planificación en la misión. En definitiva, el modelo resulta también incompleto, en la medida que el sistema se va complejizando.

2.1.3 Estructuras Híbridas

Dado que, tanto el planteamiento deliberativo y reactivo no satisfacen del todo las necesidades para diseñar estructuras de control de robots móviles, y cada uno tiene sus ventajas y desventajas, se pensó en crear estructuras hibridas que unieran los dos tipos de arquitecturas [2][3][4][5][6][7][8][10][11][13][15][17][25][40]. Con esto,

se buscaba tener la velocidad de reacción del enfoque reactivo con la capacidad de planear misiones, realizar mapas, y organizar los comportamientos que tiene el enfoque deliberativo. En general, para tratar de unir estos enfoques, la parte deliberativa opera de manera asincrónica con la reactiva, determinando objetivos intermedios que guían los comportamientos de la parte reactiva. Con las arquitecturas híbridas se logra mejorar lo que se tenía en las estructuras reactivas con los comportamientos que debían ser sólo acciones de reflejos; llegando a tener estados internos y memoria, que pueden ser modificados por la parte deliberativa. De este modo, este paradigma se puede visualizar como una estructura paralela en la cual la planificación ocurre en una escala de tiempo más lenta, pero mira un horizonte más lejano de tiempo que ayuda a influenciar qué comportamiento reactivo debe ejecutarse en cualquier momento lo cual se puede interpretar gráficamente en la figura 3.

La ventaja de estas estructuras es la flexibilidad que exhiben para el manejo de la información, ya que permite la toma de decisiones a largo plazo pero que también esté disponible para los comportamientos de bajo nivel o reactivos de manera que éstos puedan reaccionar rápidamente a lo que está ocurriendo en el ambiente.

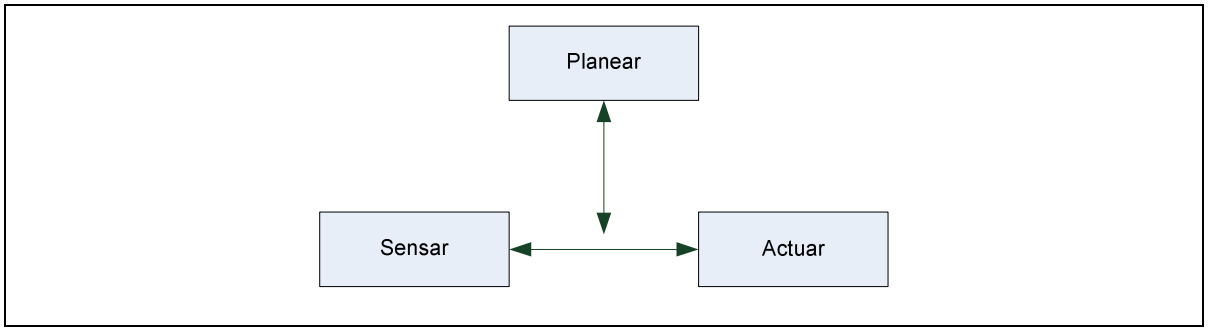


Figura 3: Organización de las estructuras hibridas

Por otro lado, la posibilidad de organizar el funcionamiento de estas estructuras con multi-threads[1], colabora en su construcción de manera modular, lo que ayuda a tener un mayor orden en el diseño e implementación. Con esto, se consigue que cada módulo funcione a su propia frecuencia de ejecución y éstos tan solo compartan

[1] Multi-thread se le llama a la posibilidad de simular la existencias de múltiples procesadores, posibilitando la programación en paralelo.

datos relevantes, según corresponda. El problema con esta nueva estructura radica en definir qué parte corresponde a la deliberativa y cuál a la reactiva, ya que no está claro el límite entre ambas. Esto aumenta, además, con el hecho de que no hay una teoría clara al respecto y existen múltiples arquitecturas que han sido desarrolladas bajo este enfoque.

En lo que sí concuerdan las arquitecturas basadas en este concepto, es que se desarrollan en niveles de menor a mayor deliberación, donde el nivel más bajo es el que corresponde al reactivo y, a medida que se va subiendo, va haciéndose más abstracto como se observa en la figura 4A. Este orden viene dado de manera natural, ya que corresponde a una escala de tiempo porque es sabido que las planificaciones de misiones o cálculos de trayectorias suelen ser más lentas que las órdenes que deben enviarse a los actuadores del robot [2][4][5][6][26]. Si bien no hay claridad respecto a cuántos niveles son los óptimos a usar, existe concordancia respecto a que debe separarse la planificación de la parte de ejecución como se observa en la figura 4B. El tema que queda por responder en torno a estas estrategias de control es que dependen -en gran medida- del diseño que se realice de éstas y no así de una demostración matemática de su funcionamiento. Este punto es importante recalcarlo, ya que las evaluaciones que se realizan de estas estrategias de control son más bien subjetivas, puesto que dependen de los sensores, de la capacidad de procesamiento y actuadores disponibles en cada uno de los robots que fueron utilizados. Es muy difícil, o bien no existe manera de evaluarlas en igualdad de condiciones, lo que deja un poco en manos del diseño de éstas el resultado final.

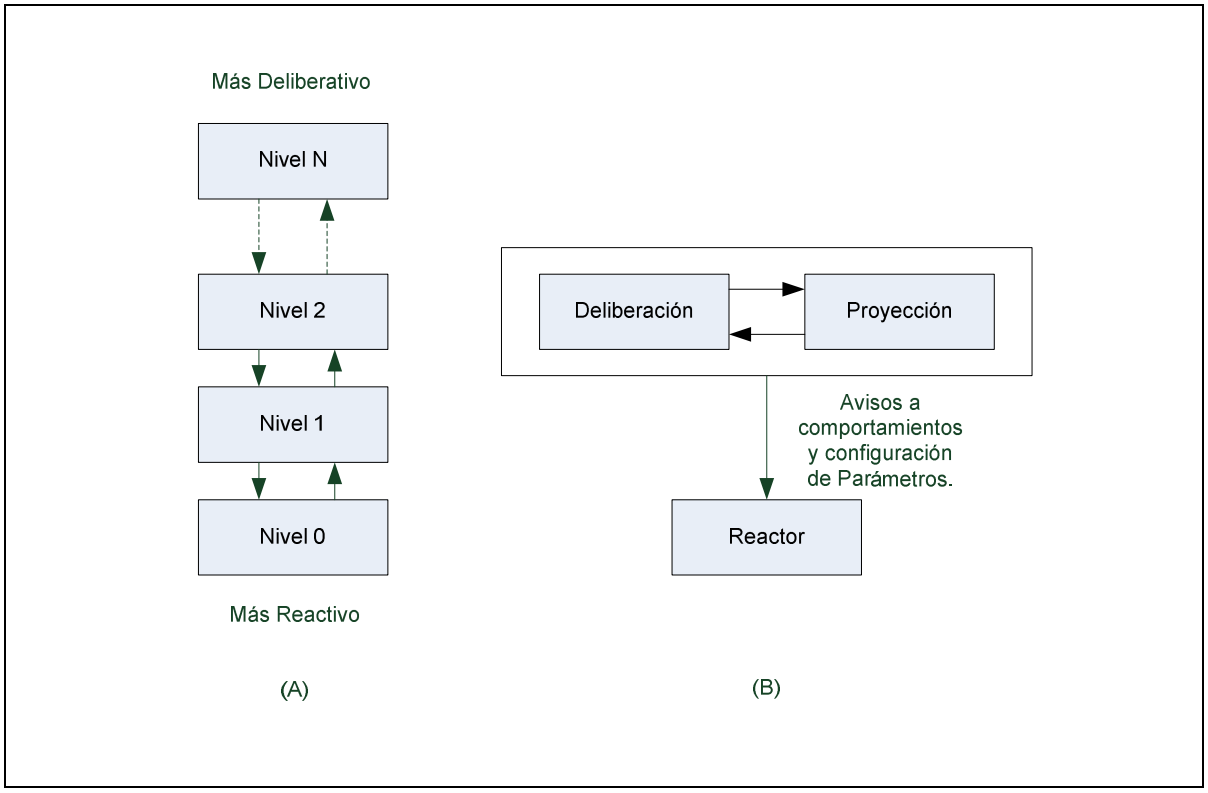


Figura 4: Diagrama de distintos tipos de ordenamientos de capas en arquitecturas híbridas.

Por otro lado, en las estructuras híbridas pueden ocurrir -si no hay un buen manejo respecto a los asincronismos y los mensajes- más problemas que beneficios [7]. Esto ocurre cuando éstas no tienen bien implementado los mensajes entre las distintas capas y cómo comparten la memoria entre ellas. Por esto es clave al momento de diseñar la estructura tener un completo conocimiento de toda la información disponible y su frecuencia de actualización. Además puede ocurrir también que haya comunicaciones innecesarias entre las distintas capas, lo que debe ser regulado por el programador, ya que tampoco hay una teoría respecto a cuáles son el número óptimo.

2.1.3.1 Dual Dynamics

Este es el nombre que recibe una arquitectura híbrida [22][26], en la cual se propone que los comportamientos sean formulados como sistemas dinámicos, usando ecuaciones diferenciales. Esto último se apoya en que las capas superiores funcionan a una frecuencia más lenta pero ejerciendo influencia en las capas inferiores. Por otro lado, se postula que el robot puede operar bajo

distintos “modos”, lo que desde un punto de vista teórico corresponde a bifurcaciones, al cambiar los parámetros que contienen los comportamientos. Con esto, se refiere a que un comportamiento puede actuar de manera distinta según el modo que éste operando. Por ejemplo, en un robot de dos ruedas el comportamiento “doblar a la izquierda”, puede ser activado bajo diferentes circunstancias, dependiendo si está en el modo “esquivar objetos” o “ir a objeto”. De esta forma, los comportamientos elementales se ejecutan de manera reactiva y dependen del modo en el que están para reaccionar de manera diferente. El procedimiento en el que se cambian estos modos viene dado por los comportamientos de la capa superior. Para realizar esto, se definen la “activation dynamics” de la “target dynamics” [22][26]. La primera se refiere a cómo se activan los comportamientos elementales con valores entre 0 y 1, dependiendo de los estímulos que los activan. La segunda, en cambio, corresponde a las configuraciones que entregan los comportamientos de nivel superior, que cambian a una tasa más lenta. Los conceptos que aporta esta teoría respecto a otras, es que propone que los comportamientos de la capa inferior funcionen aún cuando las capas superiores no estén activadas; y otra contribución es lograr modelar en base a ecuaciones diferenciales la dinámica bajo la cual se rige el sistema. Aunque esto último, desde el punto de vista práctico, no es muy útil, ya que los comportamientos a nivel de programación no siempre pueden ser ecuaciones diferenciales dependiendo de la complejidad que tengan éstos.

Una arquitectura basada en esta teoría es la propuesta en [4][5][6][35], la cual se usó para dos categorías de fútbol robótico. Primero, en la liga “small size” y luego, en la liga “humanoid”. Esta estructura divide jerárquicamente el agente (articulación, extremidad del cuerpo, agente, equipo) y el tiempo. A diferencia de lo que ocurría en la proposición explicada anteriormente, se aleja un poco de lo referente a la formulación en base a ecuaciones diferenciales del sistema, en cambio, se basa en la división del agente para modelar la dinámica. En esta estructura, a medida que las capas van siendo superiores, reciben sensores con mayor información y su salida son actuadores más abstractos. Por ejemplo, en el nivel inferior los sensores son los valores de las articulaciones, giróscopos y

acelerómetro, y en un nivel superior, es la altura del robot y la odometría. Por otro lado, en esta arquitectura las conductas se activan entre 0 y 1, pero además, los que tienen conflictos entre ellos se inhiben. Al inhibir los comportamientos que generan conflicto entre ellos, se puede usar el concepto de la teoría basada en comportamiento para manejar la unión de comportamientos.

2.1.4 Teoría “Basada en Comportamientos”

El estudio de la etología, vale decir, del comportamiento de animales en su ambiente natural, tiene bastante que aportar a la robótica [2], ya que propone una teoría para describir y explicar el comportamiento animal. En esta disciplina se acepta que la conducta de los animales sólo tiene sentido en su medio natural, entendiéndose de esta manera que el animal es parte del ambiente. Este concepto se extiende a la robótica bajo la siguiente forma: los comportamientos de los animales son fruto del nicho ecológico del cual son parte, y esto -aplicado a la robótica- lleva a pensar que un robot está diseñado para desempeñarse en un determinado nicho con el fin de realizar un cierto número de objetivos. Como consecuencia se deriva que, para el diseño de un robot, se debe tener un conocimiento acabado del nicho en el cual se va a desenvolver, para poder elaborar un modelo acabado del mundo y, de esta forma, lograr los objetivos deseados. O bien, visto de otro modo, el robot debe poder sobrevivir y convivir con el ambiente, de lo contrario, nunca va a llegar a ser exitoso [2].

Un punto interesante que define esta teoría, es la descripción de las conductas de los animales en tres grandes tipos [2]:

- Reflejos: Son rápidos, automáticos e involuntarios, motivados por algún estímulo externo. El reflejo se prolonga tanto como perdura el estímulo y la intensidad de la respuesta es proporcional a la del mismo.
- Taxis: Son comportamientos que atraen o repelen al animal y pueden ocurrir como respuestas visuales, sonoras, o cualquier sensor que tenga el animal.

- Patrones de acción fija: Son respuestas de más largo plazo que se desarrollan en los animales y son gatilladas por estímulos. Además, pueden ser generados por estados internos del animal, como el hambre, por ejemplo.

La clasificación basada en los animales sirve para pensar en una cierta manera, en la cual ordenar e implementar un sistema de inteligencia artificial, haciendo uso de una codificación cimentada en conductas. De aquí se toma el desarrollo de la teoría basada en comportamiento, la cual -considerando lo anteriormente explicado- mejora el paradigma reactivo. Esta teoría consiste en que los comportamientos de los robots son activados según los estímulos percibidos en el exterior, o bien, internos, y el robot finalmente efectúa una unión de los comportamientos que se desean ejecutar.

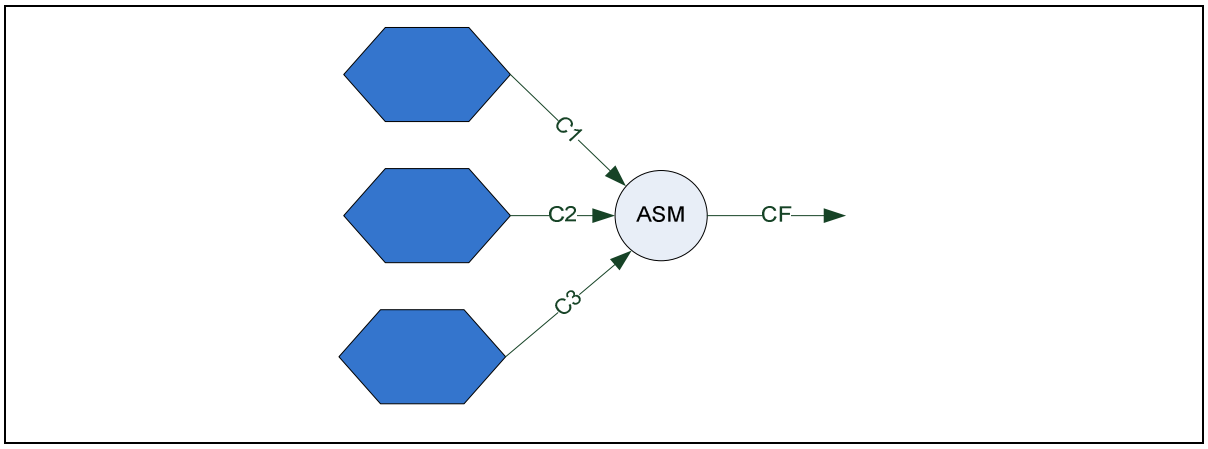


Figura 5: Esquema de funcionamiento de la teoría basada en comportamiento. C1, C2 y C3 son comandos de comportamientos distintos que serán unidos por el ASM para convertirlo en CF, el comando final.

Para unir los comportamientos del robot, esta teoría tiene lo que se llama "Mecanismo de Selección de Acciones" (ASM en inglés). El ASM es el encargado de unir los comportamientos como se observa en la figura 5, ejecutándolos de distintas formas:

- Competición
- Arbitraje
- Cooperativos

- Fusiones

Estos diferentes tipos de uniones de comportamientos serán explicados más adelante. Lo importante a retener de esta teoría es que, en definitiva, el ASM elige la acción a realizar. De esta manera, en la teoría de comportamientos el funcionamiento es bastante similar a la teoría reactiva; con la diferencia que el sistema permite que los comportamientos sean activados con estímulos tanto externos como internos y que la respuesta final sea manejada a través del ASM. Estructuras creadas bajo este concepto son las que se encuentran en las siguientes referencias [2][3][12][28][30][31][32][38].

2.2 Estructuras de Programación de Comportamientos

En la sección anterior se describió, desde un nivel teórico, distintas arquitecturas para el control de robots y se señaló el uso de comportamientos para codificar su propio funcionamiento. En esta sección se abordará el tema específico de qué es el comportamiento de un robot, qué puede esperarse de éste y cuáles son las distintas maneras de construirlos y organizarlos.

2.2.1 Descripción del Problema

En robótica, cuando se desea mover un robot desde un punto a otro, es necesario -inicialmente- generar un comando a los actuadores, hasta que el robot llegue a la posición deseada. Está claro que el comando que se da a los actuadores depende de la naturaleza del controlador de éstos, pero viendo el sistema globalmente esto se puede abstraer a una posición r_t en un instante t, independiente que se requieran varias órdenes a varios actuadores para realizar ese movimiento. De igual forma, el robot tiene percepciones S_t correspondientes a los datos entregados por los sensores en ese instante. Por lo tanto, un comportamiento de un robot corresponde a una función $\beta(s_t) \rightarrow r_t \; \forall \; s_t = \{s_0, \cdots, s_i\} \; i = cantidad\ de\ sensores$ y $r_t = [x, y, z, \theta, \varphi, \rho]$ aunque r puede ser de menor tamaño dependiendo de los movimientos posibles del robot, y r se descompone luego en $k(r_t) = \alpha_t \; \forall \propto = \{\propto_0, \cdots, \propto_n\} \; n = cantidad\ de\ actuadores$ con la función k, la cinemática directa del robot. Cada comportamiento que reacciona a uno o varios estímulos implementa

una función β particular a cada uno de los comportamientos deseados. Por lo tanto, cada una de estas funciones debe activarse cuando se recibe un estímulo. Estas funciones pueden ser combinadas en casos en que se codifiquen para distintos actuadores independientes, con lo cual no hay problemas para combinarlos, como por ejemplo, ocurre en los robots humanoides con las órdenes de la cabeza versus las del resto del cuerpo. Ello permite que los comportamientos del robot puedan ser divididos en tareas genéricas y no realizar una programación tan compleja que requiera de muchos casos independientes que abarque todos los casos. Este concepto se conoce también como comportamientos primitivos u ortogonales [31], los cuales son comportamientos básicos que necesita el robot para funcionar y que pueden desempeñarse en paralelo sin necesidad de alterar el funcionamiento de éste.

Complementando lo explicado anteriormente, se usa también como salida de los comportamientos el concepto de "actuador virtual" (*VA*) [39], el cual se refiere a que, independiente del grado de abstracción del comportamiento, la salida de éste es una acción, aunque no necesariamente llegue a ser un comando. Por ejemplo, en un comportamiento de bajo nivel, el *VA* puede ser perfectamente una orden r_t; en cambio, en un comportamiento de más alto nivel puede ser el cambio de misión del robot. Esto porque en el nivel superior se evalúa hasta ese nivel de abstracción, y luego el cambio de misión lleva a que de los comportamientos de menor nivel de abstracción se transforme, finalmente, en un comando a los motores. Al agregarse el concepto de *VA*, la definición de los comportamientos puede generalizarse y así dividir aún más las tareas en pequeños bloques más manejables.

2.2.2 Taxonomía de Mecanismos de Selección de Acciones

La clasificación de los distintos tipos de ASM es útil para poder compararlas y evaluarlas. Este trabajo fue realizado en [33], donde se clasifican los diversos tipos. Se diferencian dos subclases principales: de fusión de comandos y de arbitraje. Según esta clasificación, los ASM de arbitraje permiten que uno, o un grupo de comportamientos, en un instante tomen el control por un periodo de tiempo hasta que se activen nuevamente otros comportamientos. Por lo tanto, los mecanismos de selección de acciones por arbitraje permiten garantizar que no habrá conflicto en las

salidas de los comportamientos en ejecución y, por consiguiente, no es necesario que haya algún método de fusión y combinación entre ellos. Los ASM de arbitraje se subdividen en las siguientes categorías: basado en prioridad, máquinas de estados o el ganador toma todo.

Por otra parte, los ASM de fusión permiten que múltiples comportamientos, que se activan a distintos tiempos, contribuyan juntos a la acción final que realizará el robot. En vez de estar preocupados en seleccionar el comportamiento a ejecutar, estas estructuras posibilitan que todos los comportamientos se activen concurrentemente y luego se preocupan de ir uniendo y filtrando las salidas de los comportamientos. Estos métodos deben lidiar con el hecho de que múltiples conductas pueden requerir el mismo control sobre el robot y, por lo tanto, se han desarrollado varios métodos de colaboración entre comportamientos. Gracias a la cooperación que se desarrolla entre los comportamientos, es que generalmente los ASM de fusión permiten que se resuelvan problemas de múltiples objetivos. Por ejemplo, en los problemas de navegación, estas estructuras facilitan la solución del problema de ir a un objetivo con el de evadir obstáculos, de manera conjunta. En un ASM de arbitraje se haría lo uno o lo otro, pero no ambos combinados. Esta categoría se subdivide en cuatro subclases: votaciones, superposición, lógica difusa y múltiples objetivos. La clasificación de los ASM fue hecha en [33] y se observa en la figura 6.

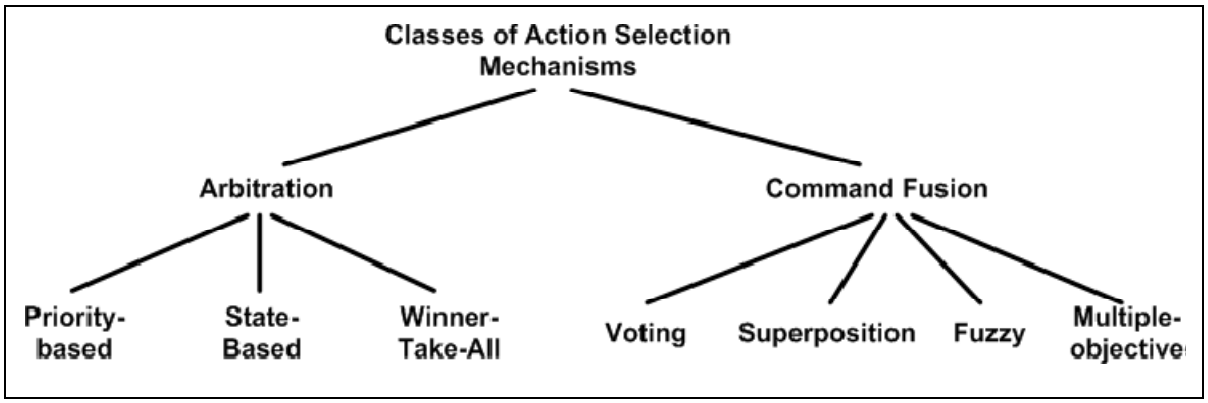


Figura 6: Clasificación realizada por Pirjanian [33] de los ASM.

Ambos tipo de ASM tienen sus fortalezas y debilidades; por ejemplo, los ASM de arbitraje admiten un uso más eficiente de los recursos del sistema, ya que permiten

usar toda la capacidad de procesamiento disponible en el momento, para ejecutar lo que el comportamiento requiere. En cambio, en un ASM de fusión de comando se requiere que todos los comportamientos se ejecuten en todo momento, para obtener la acción final a ejecutar. Esto último – y en la medida que el sistema se va volviendo más complejo -lleva a que la cantidad de comportamientos vaya creciendo y, con esto, los recursos computacionales necesarios para procesarlos también aumentan. En los ASM de arbitraje, aunque la cantidad de comportamientos aumente, no implica problema, ya que en estos casos sólo uno, o un grupo de comportamientos, se ejecuta durante un lapso de tiempo.

2.2.3 Arbitraje y Fusión de Comandos

En esta sección se revisarán las estructuras y reglas que se analizaron en mayor profundidad para resolver el objetivo planteado en la tesis. La implementación realizada toma en consideración estos aspectos teóricos.

2.2.3.1 Estructuras Jerárquicas

Las estructuras jerárquicas son las más usadas para organizar los comportamientos de los robots. Se estructuran en base a árboles y se van recorriendo desde arriba hacia abajo como se observa en la figura 7. En la parte superior del árbol se encuentran los comportamientos más abstractos, o de nivel más alto y, a medida que se va recorriendo el árbol, éstos comienzan a ser más concretos. Dado lo descrito anteriormente, el comportamiento ubicado en la parte superior del árbol es aquel de mayor jerarquía en el sistema y a medida se va bajando están los de menor jerarquía. Esto gráficamente se representa con estructura como la figura 7. Cada comportamiento tiene como entrada las percepciones y como salida un actuador *VA* que, a su vez, sirve de configuración para el comportamiento que se encuentra abajo en la jerarquía del árbol. Los comportamientos que están en el camino recorrido son los que se hallan activos y los otros están inactivos. Al camino recorrido se le puede denominar camino de activación. Cada comportamiento, o nodo de este árbol, es responsable de determinar qué sub-comportamiento debe ser activado. Esto se hace mediante el estado interno que tenga cada comportamiento.

Una vez recorrido el árbol, esta estructura permite que se ejecuten solamente los comportamientos que se encuentran activos, hasta que se recorra nuevamente el árbol. En estas estructuras los comportamientos tienen relaciones, dependiendo de la posición en la que se ubican en el árbol. Estas relaciones pueden ser del tipo padre-hijo o entre pares. Es importante recalcar que aquí no importa el orden o la prioridad con la que se ejecute un comportamiento, sino que, más bien, importa el camino escogido para recorrer el árbol. Por otro lado, puede haber también coordinaciones entre comportamientos de un mismo nivel, si así fuese necesario. Sin embargo, en ningún caso hay competencia entre comportamientos por el control de la acción a ejecutar. Este tipo de estructura es de por sí un ASM de arbitraje, ya que sólo los comportamientos activos son usados por el robot.

Por último, cabe recalcar que este ordenamiento lleva fácilmente a descomponer una tarea compleja en varias sub-tareas más específicas que pueden ser reutilizables para solucionar otras tareas.

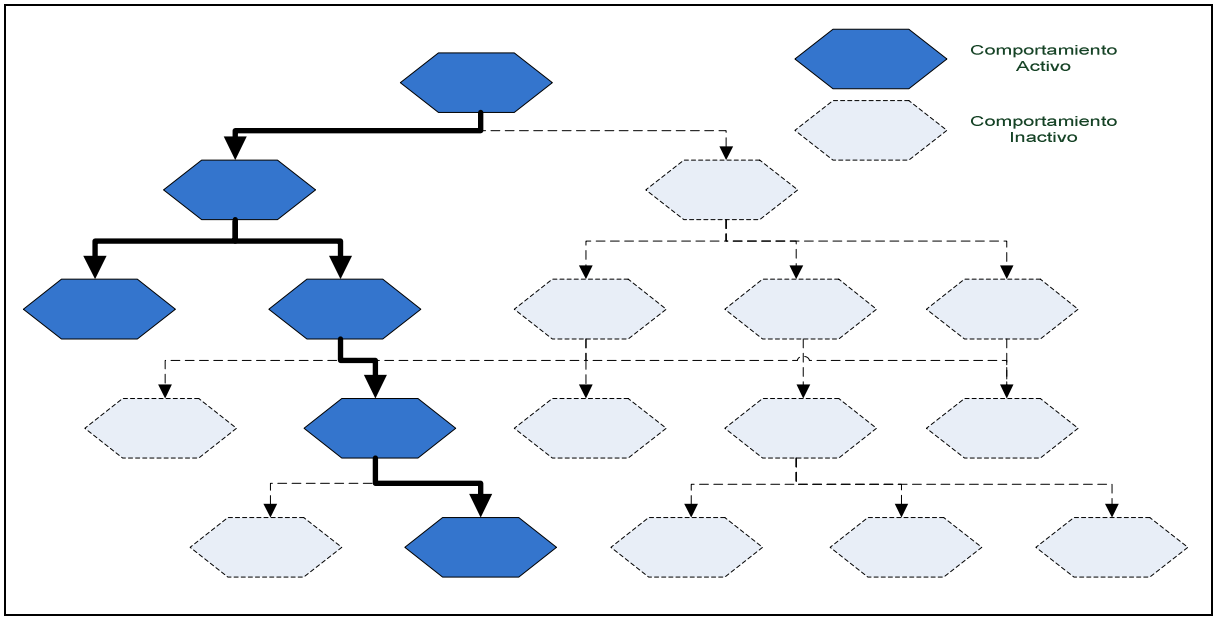


Figura 7: Estructuras Jerárquicas

2.2.3.2 Comportamientos basados en máquinas de estado

Cada comportamiento está constituido por una máquina de estado finita como por ejemplo la máquina de estado de la figura 8 correspondiente a la del comportamiento Go to Ball and Kick. De este modo, un comportamiento puede ser descrito formalmente como un set de variables de control $cs_i \in CS$ el espacio de todas las variables de control, en que cada variable es parte de una política de control π_{va}. Esta política π_{va} es la función que permite transformar las percepciones S_t en una acción o *VA* cuando se selecciona cs_i. Además, cada cs_i tiene programado qué sub-comportamientos deben ser activados cuando ese estado es activado [2][33].

La función de transición de estados depende de la información perceptual del robot S_t y de variables internas del comportamiento. Aún cuando cada tarea tenga un inicio y un fin, las máquinas de estado no necesariamente lo tienen, ya que no hay un estado inicial ni final y dependen de las condiciones dadas por las percepciones S_t que se tengan en el momento en que se evalúen. La ventaja de las máquinas de estados es que facilitan el manejo del ruido, ya que la condición para pasar del estado A al estado B no necesariamente es la misma para pasar del estado B hacia A. Esto también permite controlar que el sistema no se quede oscilando entre estados y, por lo tanto, no cumpla la tarea que se le asigna.

En la Figura 8 se describe la máquina de estado del comportamiento de “ir a la pelota y pegarle”. Cada estado de este comportamiento define cuáles son los objetos de interés a observar y cuál es la posición objetivo a la que se quiere llegar. Por ejemplo, en el estado Go To Ball Without Info es el estado en el cual el robot no tiene buena información del arco rival pero sí de la pelota y, por lo tanto, la trayectoria final a la que desea llegar es sólo acercar a la pelota sin tener clara la orientación. En este estado se genera que el objeto de interés para la visión del robot sea alguno de los arcos y no la pelota, lo que provoca que el *VA* sea alguno de los arcos como objeto de interés. Esto lleva a que el robot comience a buscar alguno de los arcos mientras va en dirección a la pelota. Este comportamiento se ve modificado cuando el robot logra ver un arco y, en

consecuencia, ya puede definir una trayectoria definitiva para acercarse a la pelota.

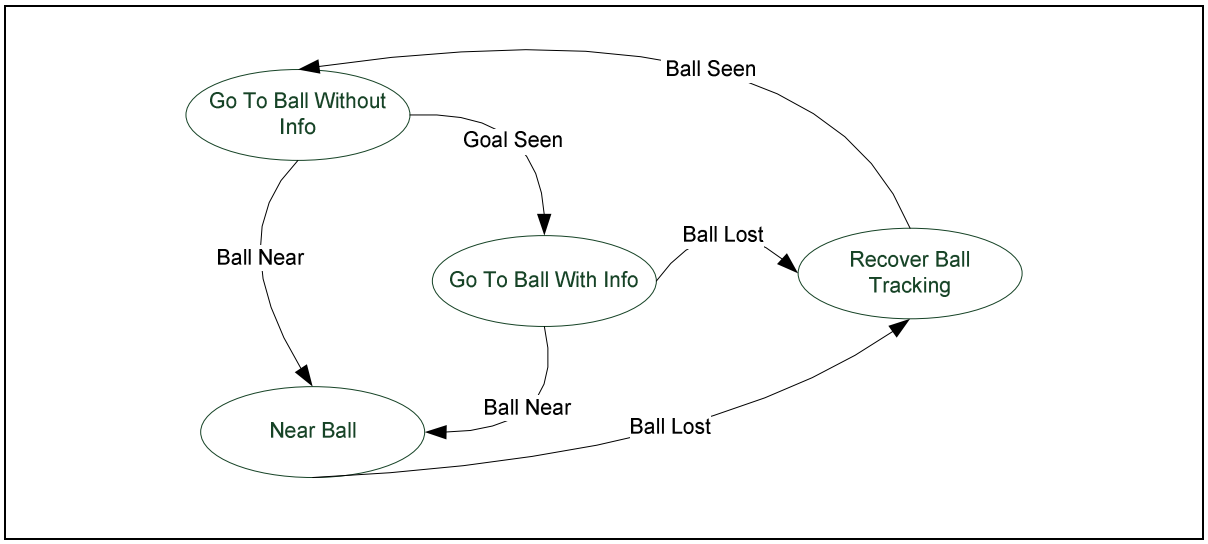


Figura 8: Máquina de estado del comportamiento Go To Ball and Kick.

2.2.3.3 Fusión de Comandos y comportamientos

Existen numerosos métodos para la fusión de comando y comportamientos; cada uno, buscando resolver algún tipo de problema. En la investigación de este campo existe consenso en que todos tienen sus fortalezas y debilidades, dependiendo de cómo se quieran aplicar. Otro punto en que también hay concordancia es que para facilitar esto se debe estandarizar la escritura de los comportamientos y comandos que se deben fusionar [2]. Al respecto, un método muy usado para unificar trayectorias en un robot es el de los campos de potenciales. Este método posibilita la generación de puntos de atracción (objetivo donde se desea llegar) y puntos de repulsión (objeto o lugar donde no se quiere dirigir al robot) [2]. La teoría de campos de potenciales está inspirada en los campos eléctricos y generalmente se usa el funcional $F \propto \frac{1}{distancia\ a\ objetivo^2}$. Para combinar los diferentes *VA* generados por los comportamientos se fija un grado de activación del comportamiento. El grado de activación se obtiene generalmente de la información de las percepciones y luego, con umbrales específicos de cada comportamiento se procede a calcular la fuerza y se determina si es atractiva o repulsiva. Dependiendo de la cantidad

de percepciones que se tengan, será la cantidad de objetos que se deberá sumar al mapa para obtener la trayectoria final. La ventaja de este método es que permite de manera fácil obtener trayectorias para el robot. Su desventaja es que requiere de una calibración grande, ya que se podrían obtener resultados no deseados, sobre todo cuando se está cerca de los umbrales. Por otro lado, este método presenta problemas para fusionar comportamientos más abstractos, en los cuales la distancia a los objetivos no es tan directa de obtener.

Es importante recalcar que, para poder generar fusiones de comportamientos se requiere tener salidas compatibles entre ellos. Es por esto que el concepto de *VA* se torna importante si se desea realizar esto último. En efecto, otro tipo de fusión de conducta es a través de votaciones, la cual consiste en que se enumere la cantidad de comportamientos que votaron por un determinado *VA* y luego seleccionar el que posea mayor votación. Otra variación de este método se puede realizar obteniendo un nivel de activación de cada *VA* y luego, tomando el máximo. La dificultad de estos métodos radica en definir *VA* que sean lo suficientemente genéricos como para lograr tener el mayor número de comportamientos que los usen, ya que, de no ocurrir aquello, termina siendo un sistema jerárquico que además evalúa a todos los comportamientos en paralelo.

2.3 Control Activo de Sensores

El control activo de sensores se plantea como una solución al problema de obtener poca información desde algún sensor. Control activo de un sensor puede entenderse bajo diferentes definiciones, una es optimizar el procesamiento de un sensor realizándolo sólo en áreas donde se estima que haya información extraíble y, de este modo, poder obtener más rápidamente la información. Por ejemplo, se puede dar el caso de llevar varias imágenes similares y, por lo tanto, se puede pensar en buscar dentro del espacio de la imagen, únicamente en los sectores donde hubo información en las imágenes anteriores, para lograr disminuir el tiempo de procesamiento del robot. Otra forma puede ser entendida como la forma de mover el sensor y, por último, puede ser el concepto de qué buscar con el sensor. En el caso de este trabajo, el primer concepto no se abordará, ya que se quiere solucionar el hecho de tener una política de uso del sensor

que en este contexto es una cámara. Efectivamente, en el caso de una cámara con un ángulo de apertura limitado, no se puede observar completamente el medio y, por lo tanto, no se tiene toda la información disponible. La información que se puede extraer es sólo la que está en el rango del lente. La solución para cubrir un área mayor es moviendo el sensor a través de actuadores, pero en el caso de un robot móvil, se debe agregar además la dificultad del movimiento del robot, dado que la cámara puede tener movimientos independientes (dentro de un rango) del cuerpo del robot. El objetivo de obtener mayor información con los sensores es necesario para lograr una mejor auto-localización del robot. En el caso del fútbol robótico esto es fundamental, por ejemplo, con el arquero, para poder quedarse en el arco, y en el caso de los jugadores, si se desea poder compartir mejor información y con esto, establecer algún tipo de estrategia colectiva. Por lo tanto, el problema que se quiere resolver es el de disminuir la incertidumbre en la localización del robot, obteniendo mayor información del ambiente.

El equipo venía realizando investigación en este campo desde antes de la realización de esta tesis [19][20][21]. La solución planteada en ese trabajo consistía en realizar una iteración del filtro de Kalman [36][37] como se observa en la figura 9, usado para la localización del robot, simulando la política "qué hará en un futuro cercano"; es decir, dónde se moverá, y también las observaciones que pueden llegar a tener lugar durante esa política. Con esto, se puede determinar cuánto afectarían las posibles observaciones en la localización. Para efectuar estas simulaciones, se realiza el cálculo tanto de la parte predictiva (simulando la política de juego) estimando un estado futuro del robot (x_k^-, P_k^-), como también la correctiva (con la simulación de los posibles objetos vistos) generando el estado futuro del robot (x_k, P_k). La simulación de las visiones de los objetos $h(x_k^-, 0)$ permite representar cómo se corregiría la pose del robot si efectivamente se viese generando el estado $\tilde{x}_k = (x_k, x_k^B)$, y luego, con esa pose, se determina una función de valor $V(\tilde{x}_k)$, la cual depende de la tarea que está realizando el robot como se aprecia en la figura 9.

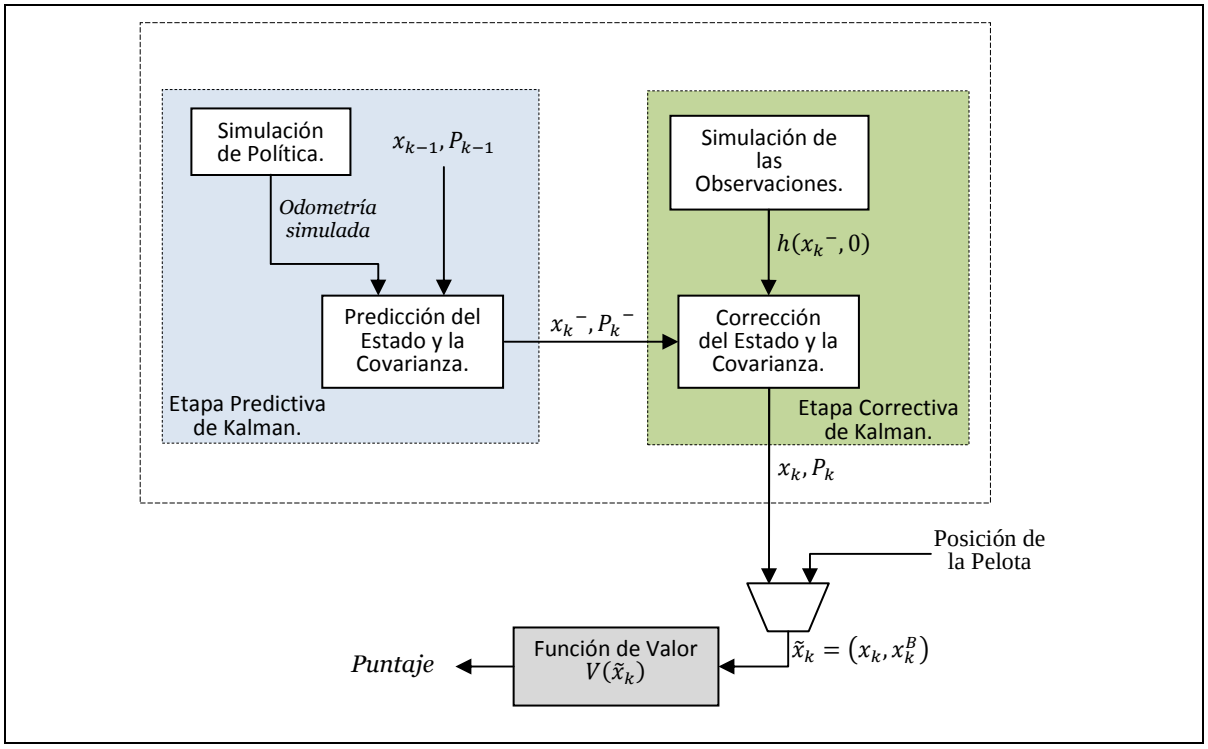


Figura 9: Esquema de funcionamiento del algoritmo usado en visión activa.

Este algoritmo toma en cuenta las varianzas en los objetos al momento de simularlos, lo que permite que la corrección de la pose al mirar el objeto también considere este ruido. Como se observa en la figura 9, la pelota se agrega sólo al final a la función de valor, ya que al ser ésta un objeto móvil, no se usa para la corrección de la pose pero sí debe ser tomada en cuenta, ya que influye en el cálculo de la función de valor (pues ésta depende de la política de juego que tiene en el instante el robot). La salida de este algoritmo es un puntaje para cada uno de los objetos evaluados además de la pelota, y con este puntaje el robot escoge la dirección de la mirada hacia el objeto con mayor puntaje asociado.

Esta solución tiene el reparo de que el proceso de simulación puede ser bastante costoso computacionalmente, dependiendo de la cantidad de objetos de interés que hayan. Además, se debe sopesar que este proceso se realiza cada vez que el robot puede ejecutar un movimiento de la cabeza, por lo que si el cálculo toma tiempo, el resto de los movimientos se ejecuta más lentamente. Por otro lado, este esquema se ha usado únicamente para el comportamiento del arquero, ya que en éste la localización es fundamental para su funcionamiento, y no se ha extendido aún para otros

comportamientos, justamente porque en los otros se requiere de mayor reactividad. Además, en la implementación descrita en el trabajo anteriormente publicado [18][19][20][21], no se toma en cuenta el hecho que dos objetos podrían observarse en conjunto. Otro punto a recalcar es que este algoritmo sólo se ejecuta cuando el robot ya ha detectado al menos una vez los distintos objetos en la cancha y, por consiguiente, ya tiene una estimación de su pose, y mientras no haya detección de algo, el algoritmo no se usa. Para poder seguir estimando dónde están los objetos a pesar de los desplazamientos del robot se usa el módulo de *World-Modelling* [36][37].

Se revisaron también otros trabajos realizados en este campo, como [14], que usa filtro de partículas, ya que se evaluó al comienzo de este trabajo usar un nuevo algoritmo de visión activa. Finalmente la decisión de ingeniería por la que se optó fue la de dar continuidad a los trabajos previos realizados por el equipo [18][19][20][21], con la ventaja de que no requería comenzar desde los inicios las investigaciones y la implementación dentro de la arquitectura híbrida que se propone a continuación.

III. ARQUITECTURA HÍBRIDA PROPUESTA

3.1 DESCRIPCIÓN DE ARQUITECTURA EXISTENTE

A continuación, se describe la arquitectura y librería existentes antes del trabajo realizado, gran parte de esta descripción será utilizada en el desarrollo de la tesis.

3.1.1 Diseño de Módulos del Robot

Dentro de los diferentes robots[2], el software está estructurado en 4 módulos que interactúan entre ellos [36][37]:

- *Vision*: es el encargado de capturar y procesar la imagen del robot. La detección del objeto se hace mediante la segmentación de colores en la imagen, y luego, formando regiones, a las cuales se le aplican filtros dependiendo de la forma, para determinar si los objetos están presentes en la imagen. En todos los objetos, con excepción de la pelota, se determina su pose (x, y, θ) relativa al robot. En el caso de la pelota, es sólo la posición (x, y) relativa al robot. En el robot este módulo funciona aproximadamente a 30 cuadros por segundos y es utilizado para coordinar el resto de los módulos, ya que de él provienen las percepciones del robot. En efecto cada módulo tiene su reloj interno pero se usa visión para coordinar los módulos. Este módulo consta, en realidad, de 2 threads, ya que uno se encarga de la captura de la imagen y otro del procesamiento. El thread de captura de la imagen debe ser adaptado dependiendo del robot en el cual funcione el código.
- *World-Modelling*: es el encargado de realizar el modelo del mundo. Éste guarda en memoria las posiciones de los objetos relativas al robot y realiza seguimientos de éstos, corrigiendo la pose que tienen con los movimientos realizados por el robot. Este seguimiento se efectúa mediante el uso de

[2] Esta librería de software está diseñada para funcionar en diferentes plataformas y no solo en un modelo de robot específico. Esto será explicado en detalle en la sección 3.1.3.

filtros de Kalman. Además, mediante los trackers[3] de los objetos, este módulo determina la localización global del robot dentro de la cancha. Dado que hay información de objetos que puede ser muy antigua (basta con que el robot haya visto una sola vez un objeto para que le realice seguimiento) se calcula la confidencia de los objetos y varianzas de las variables asociadas a los objetos. La confidencia se define como un valor asociado a cuán buena es la memoria que se tiene del objeto y ésta decrece con el tiempo cuando el objeto no es visto nuevamente. Por otro lado, la varianza es un indicador de la calidad del seguimiento del objeto y viene dado por el uso del filtro de Kalman para predecir la pose del objeto, dado los movimientos del robot.

- *Actuation*: este módulo depende completamente del robot en que se usa, salvo la interfaz con el resto de los módulos del código. En efecto este módulo presenta diferencias en la implementación de las funciones en las plataformas robot Nao y Hajime, en las que se usa. En el robot Nao es el encargado de generar las órdenes para sus distintos motores, de manera que éste pueda caminar, no siendo así en otros. Este módulo permite que el robot pueda dirigirse a la posición que la estrategia determinó, generando las secuencias de órdenes para las distintas etapas de la caminata. Para llegar al objetivo, el acotamiento de las órdenes se realiza transformando a tramos de pasos la distancia. Además, el módulo permite que éstas sean modificadas durante la marcha, en caso de un cambio de objetivo. Por último, recibe comandos para levantarse, caer, y golpear a la pelota. En cambio, en el robot Hajime, este módulo tiene muchas funciones delegadas en el micro controlador, pero debe coordinar la comunicación a través del puerto serial, además de escucharlo.

[3] Trackers es el nombre que reciben filtros de Kalman implementados en *World-Modelling* que tienen como fin realizar un seguimiento de la posición del objeto en memoria. Esto permite estimar la posición de un objeto sin que se reciba información sensorial reciente. Los trackers son los encargados de evaluar las confidencias del objeto que permiten clasificar la calidad de la información disponible en éstos. En el software usado por los robots, existe un tracker por cada objeto de interés.

- *Decision Making*: es el módulo que maneja la estrategia del robot. Este módulo existía con una estructura de comportamiento jeráraquica y sin una priorización en el uso de las diferentes fuentes de información. El trabajo de esta tesis consistió precisamente en cambiarlo por un módulo híbrido. Todo el trabajo realizado se encuentra integrado con la librería UChile-lib.

El uso de estos cuatro módulos se definió al crear la librería y en la figura 10 se muestran las conexiones entre éstos. Esta separación es usada frecuentemente en el campo de la robótica móvil [3][12][15][16][23][25][27][29][37][38], ya que permite analizar cada uno de éstos como una caja negra y, por tanto, ser implementada bajo distintos algoritmos.

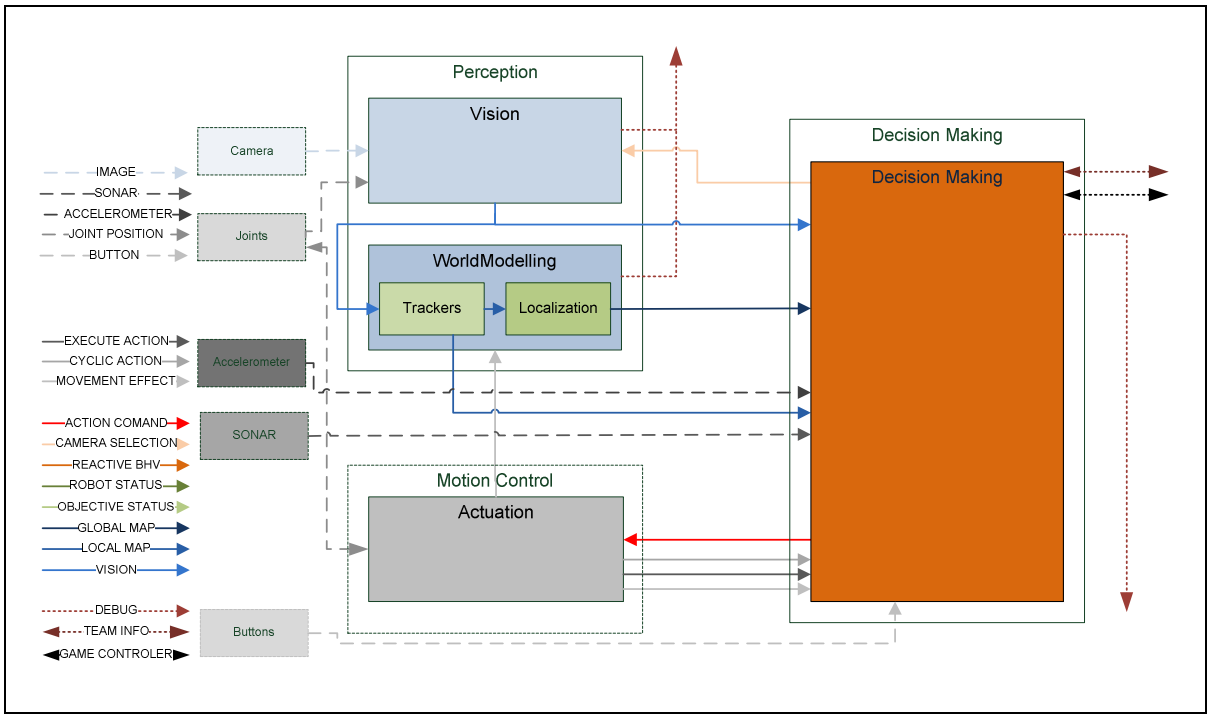


Figura 10: Diagrama de bloques de los módulos y las conexiones existentes entre ellos en el robot

3.1.2 Librería de Software UChile-lib

Este es el nombre que recibe la librería desarrollada por el equipo de fútbol robótico de la Facultad de ciencias físicas y matemáticas de la Universidad de Chile. Se implementó usando el código de programación C++, y gracias al uso de la librería

Boost [44] es independiente del sistema operativo que se use y, por lo tanto, puede ser usada en Windows, Linux y MacOs. Esta librería permite el manejo de threads y las comunicaciones entre ellos; las comunicaciones vía socket, y el uso de archivos xml, entre otras características. Otra particularidad de la librería es que el código está organizado en módulos, los cuales se adaptan a la plataforma en la que funcionan mediante el uso de la herencia. Con esto, se puede lograr que distintas plataformas robóticas compartan la mayor cantidad de códigos posible. Cada módulo tiene una interfaz configurable de comunicación para generar conexiones hacia otros módulos. Las comunicaciones permiten recibir y enviar -de manera transparente- información entre los distintos módulos del robot, o entre robots.

El manejo de comunicaciones entre threads se da de la siguiente manera:

- Se define el término paquete, como un grupo de datos que se quiere enviar. El transmisor se encarga de crear el paquete y copiar la memoria necesaria.
- El paquete es puesto en memoria y se notifica al receptor que va a recibir un paquete. Si el receptor está disponible el paquete es enviado directamente, si no, hay tres tipos de políticas que se pueden realizar: encolar el mensaje, sobrescribir el anterior, o unir los paquetes definiendo para esa comunicación un manejo específico de la información. En caso de que no haya comunicación, el receptor usa la última información que recibió.
- Cuando el receptor recibe la comunicación, se ejecuta una función que se preocupa de leer la información disponible. Estas funciones se definen en cada módulo.

Las ventajas de estas comunicaciones es que permiten abstraerse de todas las coordinaciones necesarias entre los distintos threads para compartir memoria y se encapsulan en módulos.

Estas comunicaciones se extendieron con la finalidad de que pudiesen ser vía socket. La idea consistía en poder cambiarla de socket a comunicación interna con el fin de que fuera transparente desde el punto de vista de los módulos. Para esto se definió que la comunicación levantara dos threads para permitir el

manejo asíncrono de ésta. Un thread se levanta para escuchar en el socket, el otro para escribir en él. Esto último permite que una comunicación que se envía fuera del robot y el canal esté saturado, no bloquee el módulo que envía el mensaje. Además, permite al canal recibir información por el hecho de estar en paralelo. La etapa anterior al envío por el socket, consiste en codificar la comunicación de manera que ésta pueda ser recuperada íntegramente por el receptor.

3.1.2.1 Simulador HLSIM

En la librería se diseñó un simulador de alto nivel para realizar pruebas y ajustes antes de ejecutar en el robot, llamado HLSIM. Este simulador, si bien no replica todas las interacciones reales (el objetivo no es simular un robot perfectamente), permite efectuar cambios en la estrategia, de manera de asegurarse que ésta funcione bien antes de realizar las pruebas en el robot y así, lograr reducir el tiempo de desarrollo de nuevos comportamientos y además proteger el robot. HLSIM está diseñado como un servidor al cual se conectan los robots como clientes como se muestra en la figura 11. El servidor -donde se sitúa la interfaz gráfica mostrada en la figura 12- simula los módulos de *Vision* y de *Actuation* del robot como se muestra en la figura 11. En ambos casos las simulaciones pueden ser perfectas o con un ruido agregado, de manera de hacerlas más realistas. El cliente, en cambio, contiene los módulos *World-Modelling* y *Decision Making.*

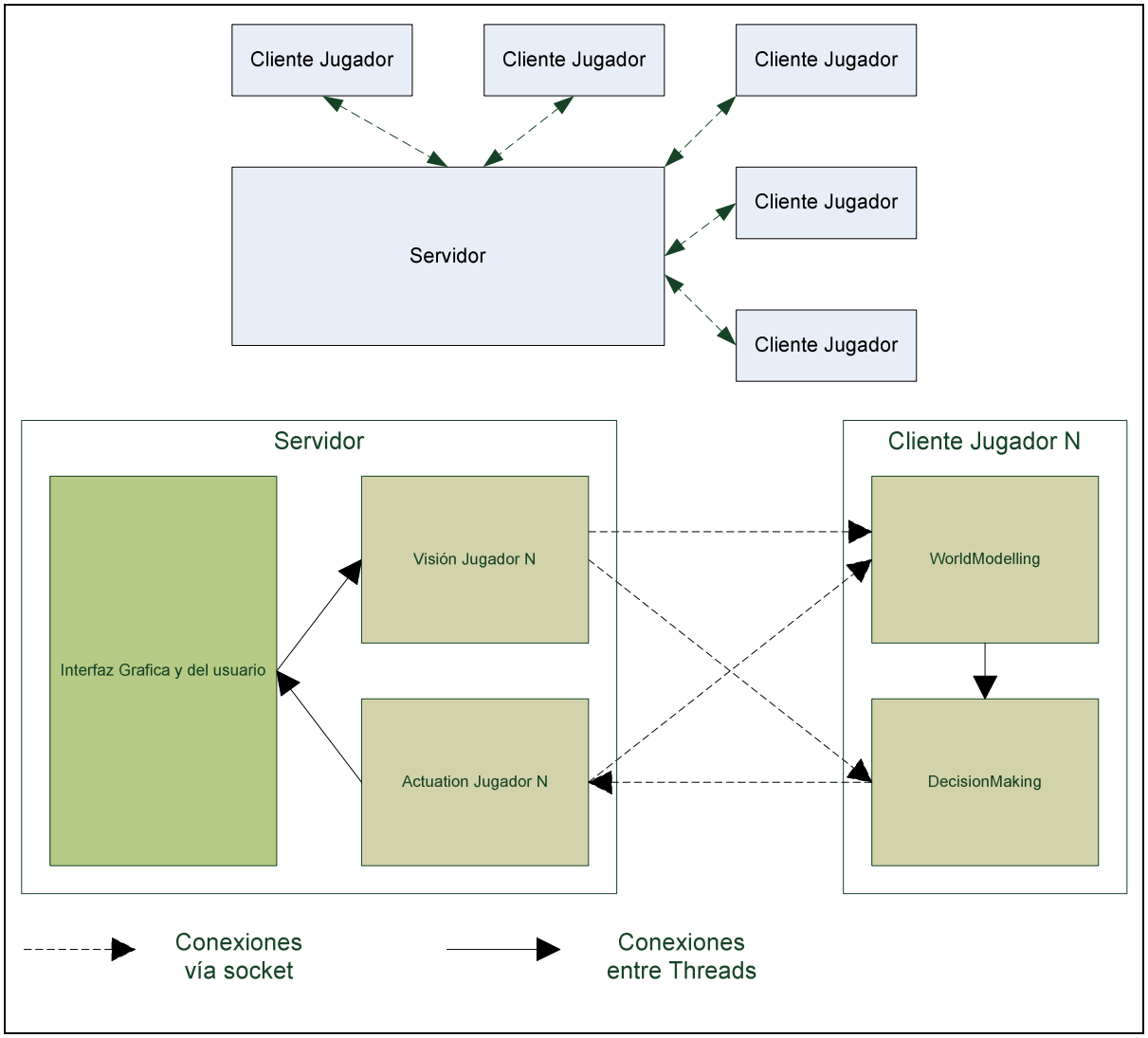


Figura 11: Diagrama de funcionamiento del simulador de alto nivel

HLSIM configura al cliente, según las opciones seleccionadas por el usuario, y permite simular los distintos tipos de robots. El simulador ayuda a crear comportamientos para el robot que luego deben ser ajustados al ser utilizados en él, ya que aún cuando HLSIM simula la física de cómo interactúan los distintos elementos en la cancha (pelota, robots, arcos), no lo hace con las caídas que puede tener el robot mientras se va acercando a su objetivo. El simulador permite agregar ruido a las simulaciones para generar comportamientos más robustos pero, aún así, se requieren ajustes al ser implementados después en el robot. En la figura 12 se muestra la interfaz gráfica del sistema con la cual interactúa y configura el usuario.

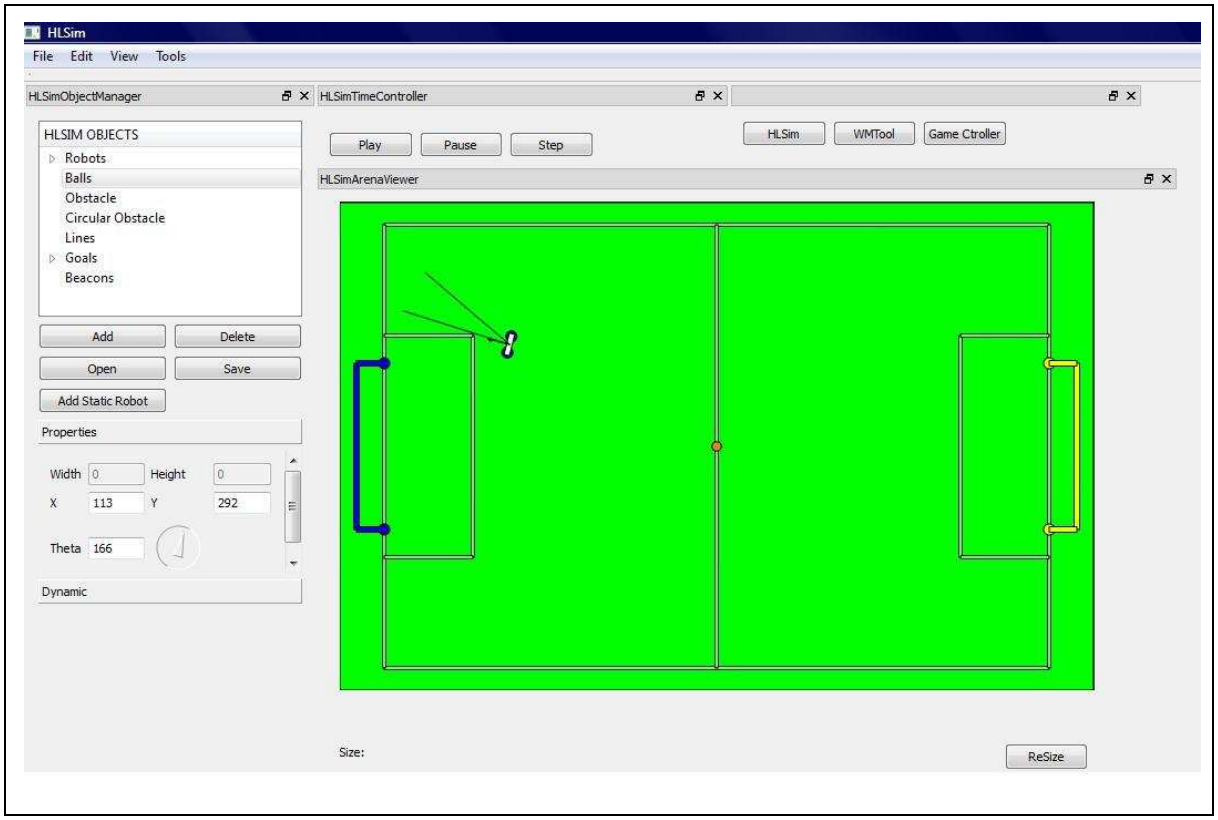


Figura 12: Interfaz gráfica del simulador

3.1.3 Descripción de los Robots

En este trabajo de tesis se harán pruebas con el robot Nao de Aldebaran [43], el cual compite en la Standart Platform League. Entre las características de este robot se encuentran: ser un humanoide de 25 grados de libertad, tener un procesador AMD Geode, usar el sistema operativo Linux, tener 2 cámaras en la cabeza, entre otros sensores. En la figura 13, se puede observar el modelo del robot. En este, el fabricante facilita una librería llamada NAOQI mediante la cual se puede acceder a los distintos componentes del hardware.

Toda la interacción con el hardware en el robot se hace mediante la librería entregada por el fabricante. Este aspecto facilita la coordinación del hardware con el software.

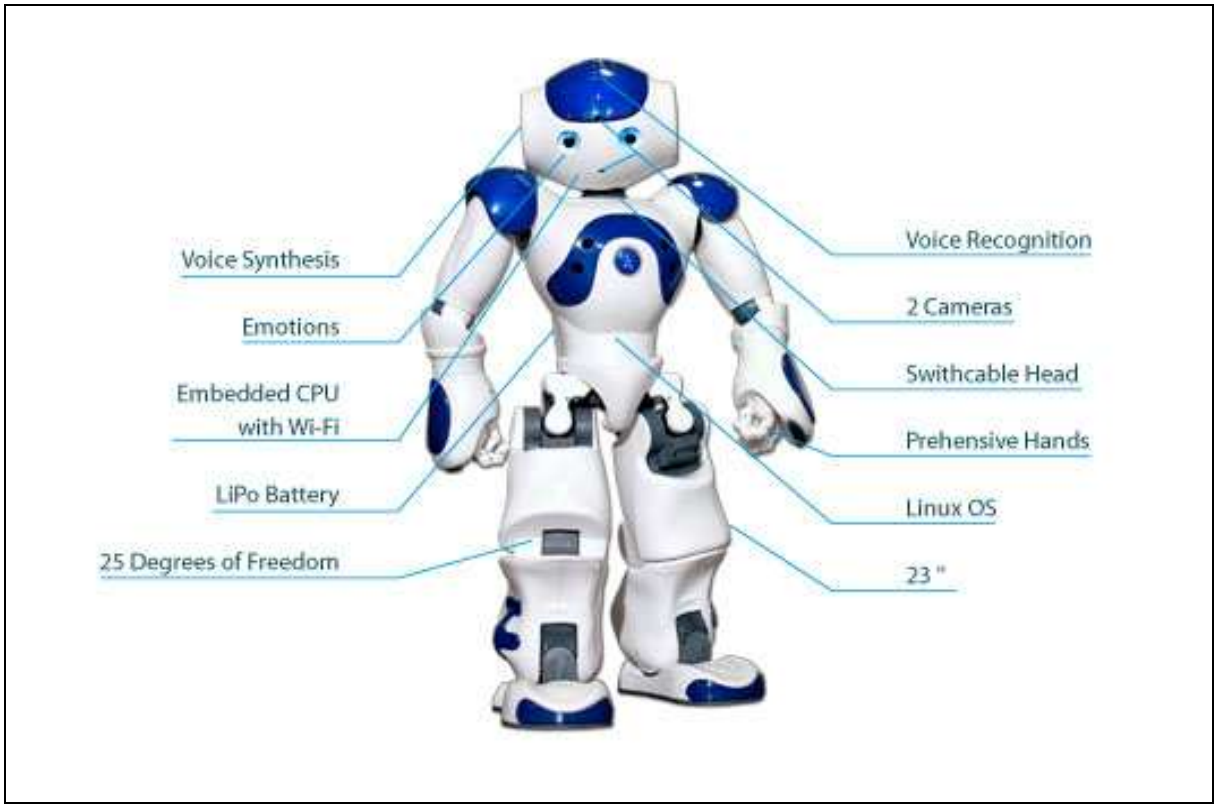


Figura 13: Robot Nao [43].

Por otro lado, se comprobó el funcionamiento la arquitectura diseñada –con algunas modificaciones- en el robot Hajime HR-18 el cual se usa en la liga humanoide y tiene las siguientes características: 21 grados de libertad, procesador Intel Atom Z530, usa el sistema operativo Linux, tiene una cámara Philips SPC900 y una placa controladora con el DSP TMS320F28335. En la figura 14 se puede observar el robot y sus dimensiones.

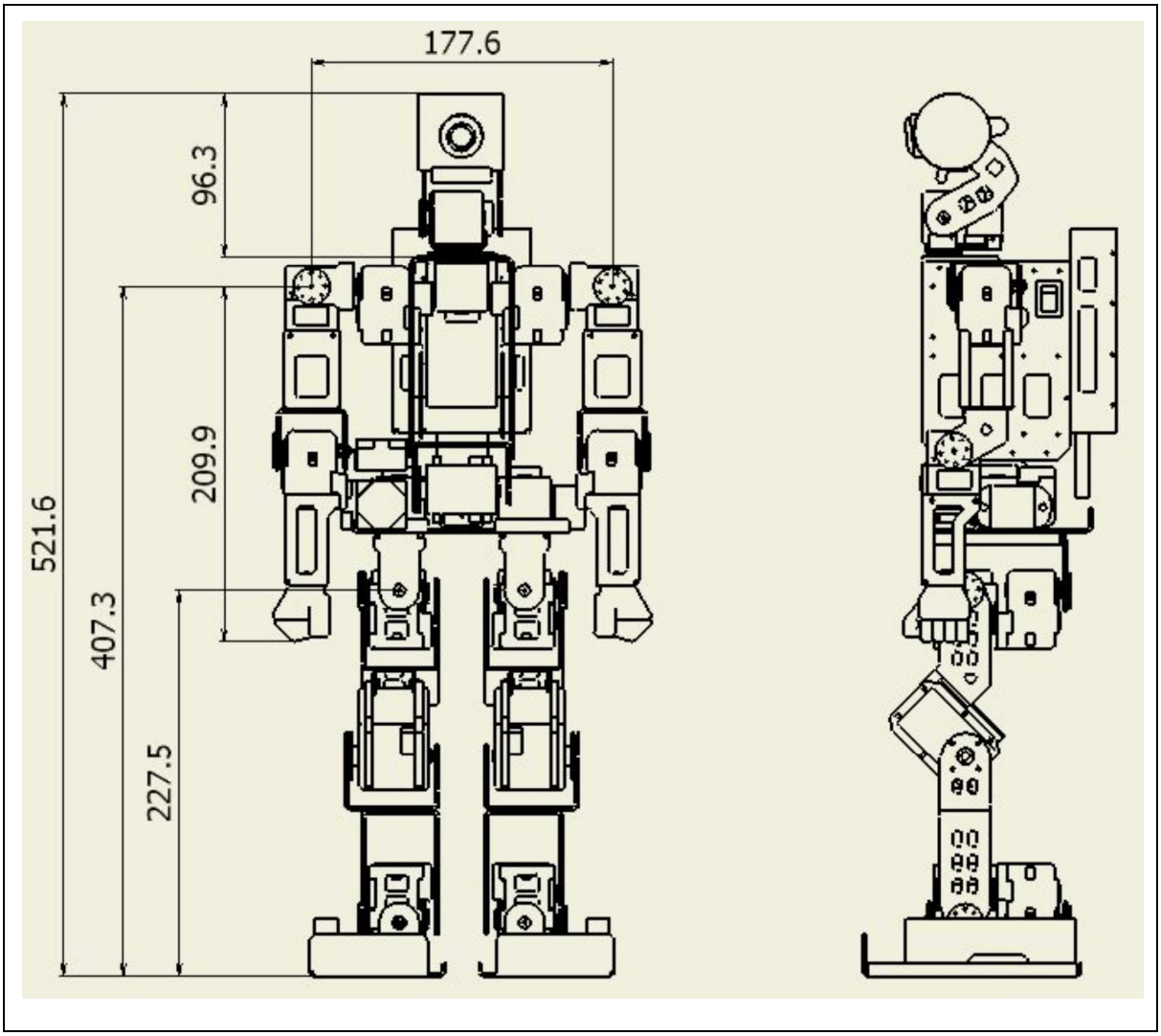


Figura 14: Robot HR-18

A diferencia de lo que ocurre en el Nao, en este robot se deben coordinar todas las conexiones con el hardware. Estas son la cámara USB que captura la imagen, y además el micro controlador que genera las caminatas, con el cual se debe comunicar mediante un puerto serial. La coordinación con el hardware es fundamental, en especial con la controladora, ya que ella no puede recibir comandos en todo momento por el puerto serial. Esto, porque se satura y el robot comienza a tener problemas de locomoción. Además, ese mismo problema lleva a que se encolen o pierdan comandos en la controladora y, a consecuencia de esto, el robot pierde reactividad. Se comprobó empíricamente que la controladora no puede recibir una orden nueva para el cuerpo en menos de 330 milisegundos y para la

cabeza tal lapso no debe ser inferior a 40 milisegundos, lo que conlleva al cuerpo a presentar una limitación en la frecuencia de las órdenes de aproximadamente 3 [Hz] y la cabeza de 25 [Hz]. Adicionalmente, cabe recalcar que la controladora no puede recibir comandos con un desfase menor a 30 milisegundos. Esto último se debe a los ciclos que debe ejecutar la controladora para interpretar un comando desde el puerto serial.

3.2 DEFINICIONES BÁSICAS

A continuación, se explicarán las modificaciones que se harán al módulo de estrategia (*Decision Making*) de la librería UChile-Lib y por lo tanto de los dos modelos de robots. Para ello, se usarán los siguientes conceptos:

- Capa o nivel: Se refiere por estos términos a un thread en el cual correrá una parte de la nueva estrategia. Estos threads heredan todas las características que tenían los módulos que se describieron anteriormente. La arquitectura se diseñó usando tres capas: reactiva (la de nivel más bajo), deliberativa y, por último, la de planificación, siendo ésta la de mayor nivel.
- Comportamientos deliberativos: se definen como una función $\beta(Sd_t) \rightarrow VA$ con Sd_t las percepciones disponibles en la capa deliberativa, en donde se ejecutan en el instante t, y siendo la salida VA un actuador virtual que será usado como configuración de los comportamientos reactivos.
- Comportamientos reactivos: se definen como una función $\beta(Sr_t, VA, M_t) \rightarrow r_t$ con Sr_t las percepciones disponibles en la capa reactiva en el instante t, VA los actuadores virtuales obtenidos de la capa deliberativa, M_t los motores libres para realizar algún movimiento, y siendo la salida r_t una de las siguientes opciones: una posición (r, θ) a la cual debe llegar el robot, una coordenada $(tilt, pan)$ en la cual mover la cabeza o un movimiento especial, como por ejemplo, pararse. Los motores libres M_t se agrupan en dos tipos: los del cuerpo y los de la cabeza. Estos pueden funcionar en paralelo sin perjuicio alguno para el robot y, por lo

tanto, cumplen con el principio de ortogonalidad [24][31], no siendo necesaria ninguna coordinación para el correcto funcionamiento.

- Actuadores virtuales (*VA*): se definen como la salida de los comportamientos deliberativos y tienen como finalidad configurar los comportamientos reactivos. Un ejemplo de actuador virtual es el objeto de interés que se desea enfocar en la cámara, permitiendo a los comportamientos de la capa reactiva ver si se le puede hacer tracking o se le debe buscar.

3.3 Esquema Propuesto de Arquitectura Híbrida

Antes de realizar el cambio al módulo de *Decision Making* se elaboró un diagnóstico de las limitantes y problemas que representaba la antigua arquitectura.

3.3.1 Análisis Módulo Original de Estrategia

Las principales razones para realizar el control híbrido en el robot es el de poder tener un robot lo más reactivo posible al tiempo que realiza tareas complejas. Esto no se estaba logrando con el enfoque anterior, por las siguientes razones:

- Evaluación constante de todos los comportamientos, no importando si son de alto o bajo nivel se evalúan constantemente, teniendo como resultado una ejecución más lenta del sistema. Como todos los comportamientos son del mismo tipo, se perjudica a los que requieren una frecuencia de ejecución más rápida, como por ejemplo, el seguimiento de objetos.
- Múltiples conexiones a un solo módulo: Se tenía variadas conexiones a solo un módulo, lo que implicaba que tuviera colas de mensajes y, por ende, tardaba en procesarlas. De estas conexiones había algunas que andaban a velocidades mayores que las otras y, por lo tanto, no siempre estaba disponible para procesar un nuevo mensaje.
- Desorden en programación de comportamientos: Desde este punto de vista, no había un orden respecto a cuándo usar información de alto nivel versus información más reciente pero ruidosa. Esto llevaba a una confusión

importante al momento de programar los comportamientos, ya que todos disponían de las mismas percepciones sensoriales y no había un consenso respecto de dónde ni qué información usar.

- En esta arquitectura se incursionó en el uso de visión activa, pero se pudo aplicar únicamente en pruebas específicas, ya que dejaba al robot muy poco reactivo.
- Todos los threads tienen la misma prioridad en el sistema operativo, por lo que se les otorga a todos los mismos tiempos de ejecución. Por ello, el thread con mayor carga demorará aún más tiempo debido a que se ejecutan por ciclo todos los threads del sistema.
- Las comunicaciones externas entre robots estaban planificadas para que fueran en este módulo, lo que aumentaría aún más los problemas mencionados con anterioridad.

La Figura 15 muestra todas las conexiones que tiene el módulo de estrategia antes del trabajo a realizar en esta tesis y sobre el cual se realizo el diagnóstico de funcionamiento señalado en los párrafos anteriores.

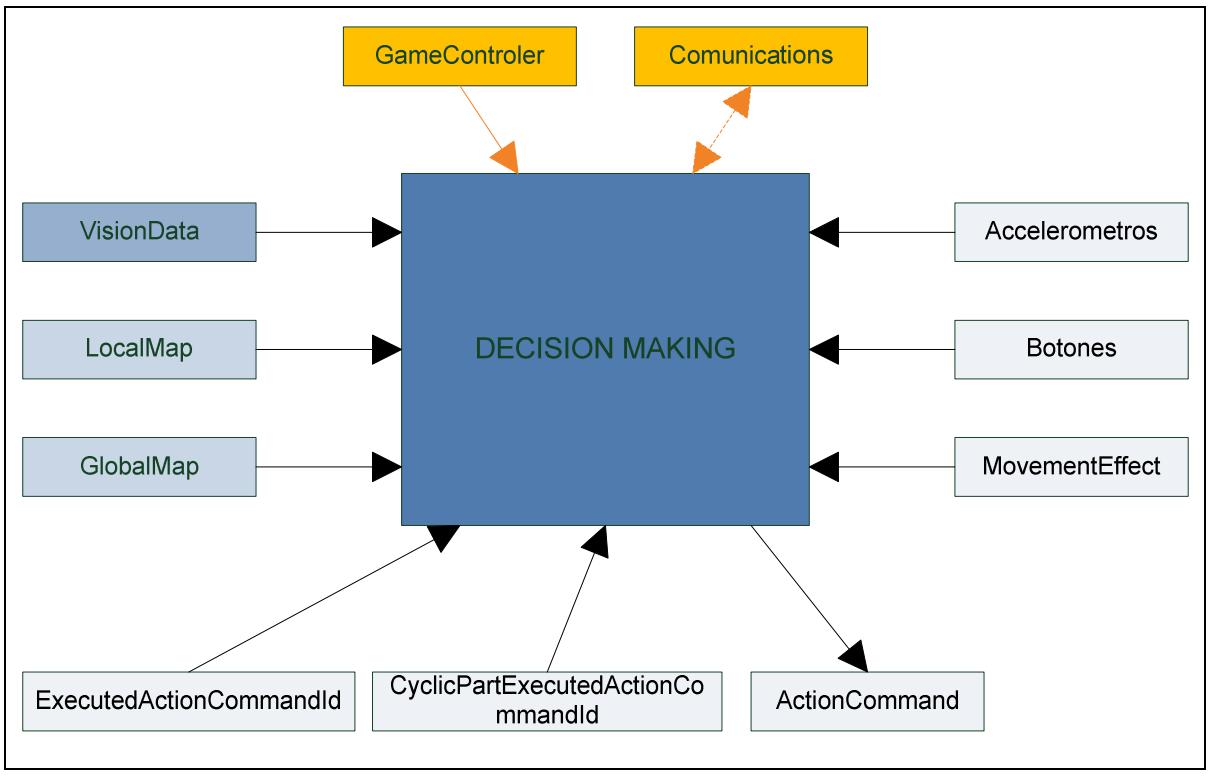


Figura 15: Distintas conexiones al módulo de estrategia

3.3.2 Estrategia Propuesta

La mejora propuesta es dividir el actual módulo de estrategia en 3 threads independientes, de manera que se pueda dividir la estrategia y usar al máximo las posibilidades de paralelismo que ofrece el sistema operativo. La división del módulo de *Decision Making* queda establecida de la siguiente manera:

- Capa Reactiva: Tiene la frecuencia más rápida, recibe información directamente desde los sensores y tiene los comportamientos reactivos. Estos últimos reciben un objetivo contenido en *VA* y seleccionan acciones a ejecutar por el robot, y por lo tanto, corresponden a movimientos que efectivamente éste puede cumplir, por ejemplo, en un robot no omnidireccional, estas conductas debiesen ser “avanzar hacia adelante”, “girar derecha” y “girar izquierda”, como mínimo. Estos comportamientos se ejecutan cuando llega una nueva percepción o se liberan motores desde *actuation*.

- Capa Deliberativa: Tiene menor frecuencia, pero recibe la información más procesada de los sensores del robot y, por ende, más confiable. Los comportamientos deliberativos se ejecutan en este nivel y llevan a que se pueda cumplir una misión y rol dados. La frecuencia de esta capa se controla mediante un timer interno con el cual regula el cambio a los *VA* que recibe la capa reactiva.

- Capa Planificación: La frecuencia en la que entrega información es aún menor que las anteriores pero más confiable. Se encarga de generar las misiones que debe ejecutar el robot y, en caso de ser un equipo de robots, se ocupa, además, de generar el rol que debe cumplir. En este caso, es responsable de las comunicaciones del equipo. Esta capa funciona bajo operación normal con un timer que controla cuándo transmitir información al resto de los robots.

Para implementar esta separación, se reconfiguraron las comunicaciones que antes correspondían al módulo *Decision Making*, distribuyéndolas entre las distintas capas. El esquema propuesto (que será justificado de manera más detallada en los próximos párrafos) busca solucionar los problemas detectados, dividiendo el módulo de estrategia en los tres niveles ya mencionados. Las razones en las cuales se fundamenta la división son las siguientes:

- Se divide el uso de información disponible en cada thread, lo que permite que la antigua estrategia sea evaluada dependiendo del horizonte de tiempo en el cual debe reaccionar. Esto es muy importante, ya que hay comportamientos que no requieren la misma frecuencia de ejecución que otros. Dividir la estrategia permite que los comportamientos críticos se vean menos afectados por comportamientos que requieren de mayor tiempo de procesamiento.

- Posibilita la existencia de un orden en cuanto a la información sensorial disponible en las distintas capas. Esto, debido a que se segmenta la información disponible por las capas dependiendo del nivel en el que están. De esta manera, la de más bajo nivel recibe la información más ruidosa pero instantánea, y la de nivel más alto recibe una menos ruidosa pero con algunos retardos.

- Comportamientos que necesitan información actualizada, como los que tienen relación con los acelerómetros, no son retrasados al tener el thread menos mensajes que atender y pueden ejecutarse lo más rápido posible. Esto ocurre de igual manera con el comportamiento de golpear a la pelota, el cual necesita ser ejecutado cuando la pelota se halla en la posición exacta para ser golpeada.
- Se crea una capa aparte para tratar las comunicaciones, con lo que se permite crear comportamientos de grupos de manera más robusta, ya que se separa al robot del equipo. Esto es importante para no complejizar la creación de comportamientos al tener que incluir en éstos información proveniente desde los otros robots.

Sin embargo, la división de la estrategia -en mayor cantidad de módulos- puede generar nuevos problemas y, por tanto, deben ser resueltos:

- Aumentar el número de threads en el proceso genera que las comunicaciones sean más lentas, puesto que deben buscar en una mayor cantidad de threads al que se enviará la comunicación. Se produce, además, un incremento de la cantidad de comunicaciones. Esto, porque hay más threads comunicándose entre sí.
- Dado que ahora se crean tres niveles de estrategia que comparten información, se requiere de coordinación entre ellos. Esta coordinación debe efectuarse de manera que el sistema no se sobrecargue con comunicaciones ineficientes, ni que haya oscilación por efecto de las comunicaciones entre las mismas. Este punto es importante, ya que puede eliminar toda la eficiencia que se pretende ganar, volviendo el sistema híbrido por mensajes innecesarios no controlados.

En el diseño de la estructura se consideraron estos problemas y se tomaron las siguientes estrategias para minimizarlos y evitarlos:

- Se centraliza todo el envío de información entre capas a través de una sola conexión. Con esto, se minimiza el aumento de las comunicaciones entre los threads, pero para realizarlo, se deben generar estructuras de datos que puedan contener toda la información que se desee enviar y, en consecuencia, tienden a

ser grandes, y no siempre se van a usar en su totalidad. Para enfrentar esto último, se hace uso de la herencia desde el punto de vista de la programación, de manera que la estructura que se va a enviar pueda tomar distintos tamaños dependiendo de la información.

- Para coordinar las capas se usa un timer en la capa deliberativa y en la de planificación. Esta idea no es nueva y es recurrentemente usada en la literatura en otras estructuras híbridas como [4][5][6][35].

Las conexiones propuestas entre módulos para la arquitectura diseñadas se observan en la figura 16.

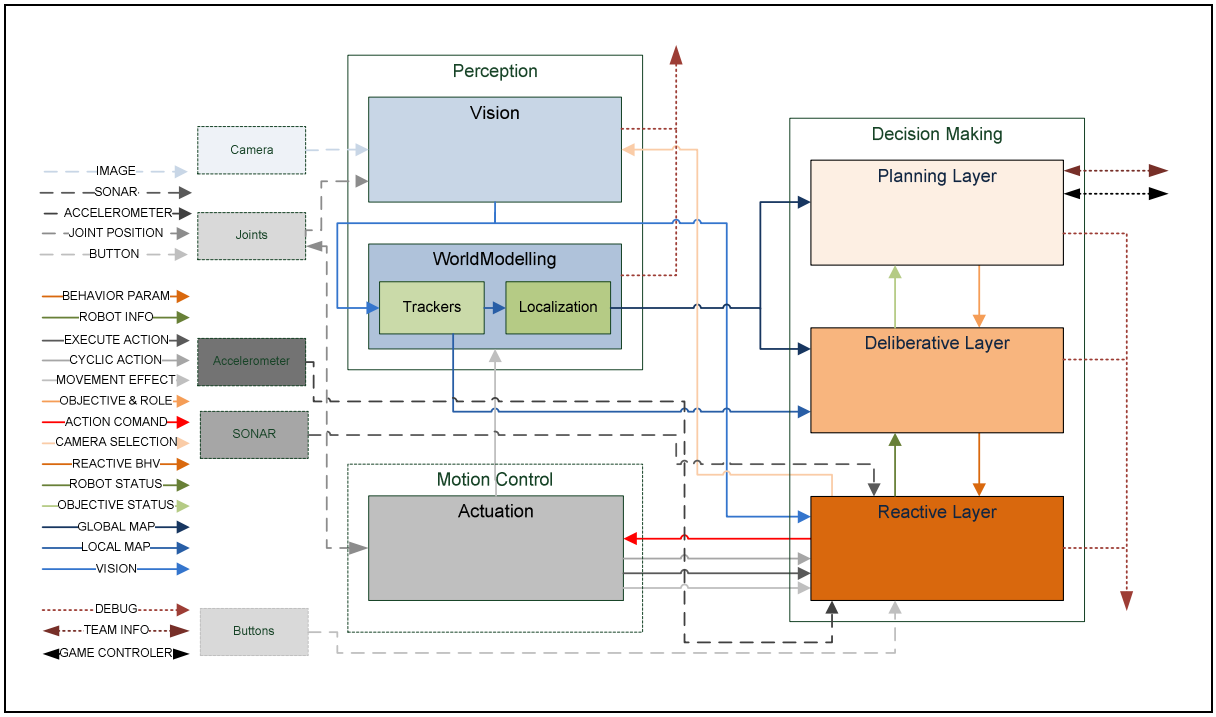


Figura 16: Conexiones entre módulos con la nueva arquitectura

Otra de las metas propuestas es que la estrategia maneje múltiples objetivos, para lo cual la solución planteada contempla contar con un ASM para fusionar comportamientos y, de este modo, lograr que el robot ejecute más de un objetivo a la vez. Los ASM, como se explicó en capítulos anteriores, suelen ser grandes cálculos. La manera de incluirlos en el diseño y evitar que afectaran la reactividad fue creando los conceptos de multiplexor y las funciones de valor. Estos dos

elementos en conjunto pasan a realizar la tarea de un ASM clásico, pero su trabajo se divide entre la capa deliberativa y reactiva. La ventaja del esquema que se presenta es que la capa deliberativa es la que absorbe la mayor cantidad de cálculos con la función de valor, dejando al nivel reactivo con el multiplexor, que maneja objetivos determinados y priorizados, logrando uno de los fines que se pretendía con la división de la estrategia en varias capas. En efecto, la función de valor en base a una creencia del estado futuro evalúa los posibles objetivos otorgando un puntaje. El multiplexor en el nivel reactivo debe tan solo actualizar cada uno de los objetivos, según el grupo de motores al cual vaya dirigido. Este esquema permite que el robot no pierda reactividad, ya que los objetivos se van reevaluando en la capa reactiva con las actualizaciones reiterativas que se van recibiendo. Un objetivo no puede desaparecer, a menos que la capa deliberativa lo cambie. Lo relevante es tener en cuenta que el sistema para funcionar requiere de una frecuencia mínima de cambio de los objetivos por parte de la función de valor, que luego se actualizan en el nivel reactivo. La manera de controlar la frecuencia mínima de la capa deliberativa pasa por la configuración que realizan los comportamientos deliberativos de las funciones de valor. En la medida que haya más objetivos a tener en cuenta, mayor será la carga computacional. En el nivel reactivo, además de actualizar los puntajes, se determina si sigue siendo factible realizar un objetivo conjunto, eliminarlo o dividirlo. Se puede eliminar en caso que el objetivo conjunto esté compitiendo con los objetivos que lo componen por separado y se divide, si los objetivos separados no están siendo usados individualmente. Lo importante que debe ocurrir es que no haya objetivos redundantes. A diferencia de otras estructuras de comportamiento híbrido, se dejó la fusión de los comportamientos fuera del nivel de las acciones a los motores y se introdujo en una de las capas de la arquitectura híbrida.

3.3.3 Mecanismo de Visión Activa

Uno de los objetivos que debe cumplir la nueva arquitectura es facilitar la implementación de visión activa en el robot, sin que esto interfiera con la reactividad del robot y, para esto, se usa la función de valor y el multiplexor. Como se explicó anteriormente, la librería UChile-Lib tiene una metodología para determinar cuánto mejora la localización del robot dependiendo del objeto que se

detecta. Uno de los problemas que tiene este método es que resulta costoso desde el punto de vista computacional y, por lo tanto, realizarlo tiene un costo en la reactividad del robot. Para resolver ese inconveniente, se propone dividir el algoritmo usado entre la capa deliberativa y la reactiva. Las predicciones que se elaboran usando Kalman se harán en la capa deliberativa en la función de valor. Luego, en la capa reactiva se ejecutará una actualización de los puntajes calculados en la capa deliberativa, con el fin de que éstos puedan reflejar los cambios en el ambiente al multiplexor. En definitiva, la idea es usar las ventajas que entrega esta arquitectura híbrida para manejar frecuencias y tiempos de ejecución distintos.

Para realizar esto último, se extendió el concepto de funciones de valor que se usaba en la implementación anterior del equipo, el cual se había efectuado en el trabajo previo con el arquero [18][19][20][21], hacia otros jugadores. Las funciones de valor están asociadas a la tarea que debe realizar el robot y mediante la cual se le otorga un puntaje a dirigir la mirada a un determinado objeto, en base al objetivo que se quiere realizar. La idea es usar el mismo algoritmo de simulación que ésta implementado en la librería UChile-Lib. Por otro lado, y con el fin de regular la carga computacional que se le exige al algoritmo, se debe flexibilizar el uso, permitiendo que, dependiendo de la etapa en la cual se encuentra la misión, se limite el número de objetos al cual buscar observar. Esto logra que el algoritmo se ejecute más rápido, ya que el tiempo de procesamiento está directamente relacionado con la cantidad de objetos a los cuales calcularles puntaje. Lo que se realizará es que, dependiendo de la misión y los comportamientos que se ejecuten en la capa deliberativa, se seleccionará una función de valor y la cantidad de objetivos a considerar. Por último, una mejora que se realiza en el algoritmo es que actualmente se permite agrupar objetos, de manera que la meta ya no sea un solo objeto, sino buscar, si es posible observar al mismo tiempo, numerosos objetos. Esto será explicado en mayor detalle en las próximas secciones.

En el planteamiento realizado para producir una visión activa en la nueva arquitectura, se considera enviar la información de los puntajes a la capa reactiva y que ésta sea capaz de actualizarlos. Para proceder con lo primero, se crea el *Score Vector* (*SV*), el cual es una estructura dinámica para enviar información entre la

capa deliberativa y reactiva. Esta estructura tiene la característica que permite enviar puntajes, ya sea de un grupo o de un solo objetivo al cual mirar en el caso de la visión activa. El objetivo de los *SV* es poder enviar información de distinto tipo entre las dos capas. Estos últimos, independiente de su tamaño, van incluidos a su vez en la estructura de transmisión de información entre la capa deliberativa y reactiva. Por otro lado, los *SV* son usados en la capa reactiva por el multiplexor (*mux*), el cual se encarga de actualizar los puntajes y determinar qué política presentar a los comportamientos deliberativos con respecto a los objetivos puntuados por la capa deliberativa. El *mux* cumple dos roles fundamentales; el primero, actualizar los puntajes calculados en la capa deliberativa y el segundo, seleccionar cuál es el que será usado por los comportamientos deliberativos.

3.3.4 Funcionamiento de la Arquitectura Propuesta

El funcionamiento de la arquitectura es asincrónico, ya que cada capa funciona a distintas frecuencias. Todas las capas generan -al momento de la creación- un mensaje que es enviado a las otras para coordinar la información. Mientras no se cambie el estado de juego, el cual se hace vía una conexión con un programa o presionando un botón en el robot, los comportamientos no se activan. Esto asegura que la información sea la misma en todas las capas. La capa de planificación define un rol y una misión al robot, a la vez que chequea un cambio de estado en el juego, lo cual se envía a la capa deliberativa. El comportamiento de más alto nivel en la capa deliberativa, revisa el estado de juego, la misión y el rol, y selecciona los comportamientos deliberativos que se activarán. Éstos ejecutan el comportamiento deliberativo de visión activa, configurando a éste la función de valor a usar y cuántos objetos evaluar en el ciclo. La función de valor determina el *SV* que se transmitirá al *mux* de la cabeza. En caso que se configure un valor para el cuerpo o que no desee ejecutar visión activa, la salida de los comportamientos deliberativos será un *VA*. Esta estructura se transmite y el *mux* usa los *SV* o *VA*, dependiendo del caso, para elegir el objetivo simple o múltiple a intentar observar y luego seleccionar qué comportamiento es el que va a ejecutar. En el caso de los *VA*, estos se encaminan directamente a los comportamientos reactivos, los cuales determinan

sí deben ejecutarse dependiendo de las observaciones disponibles y llevar a un comando para *actuation* que, finalmente, se convierte en un movimiento del robot.

En la Figura 17 se observa el esquema completo de la arquitectura propuesto.

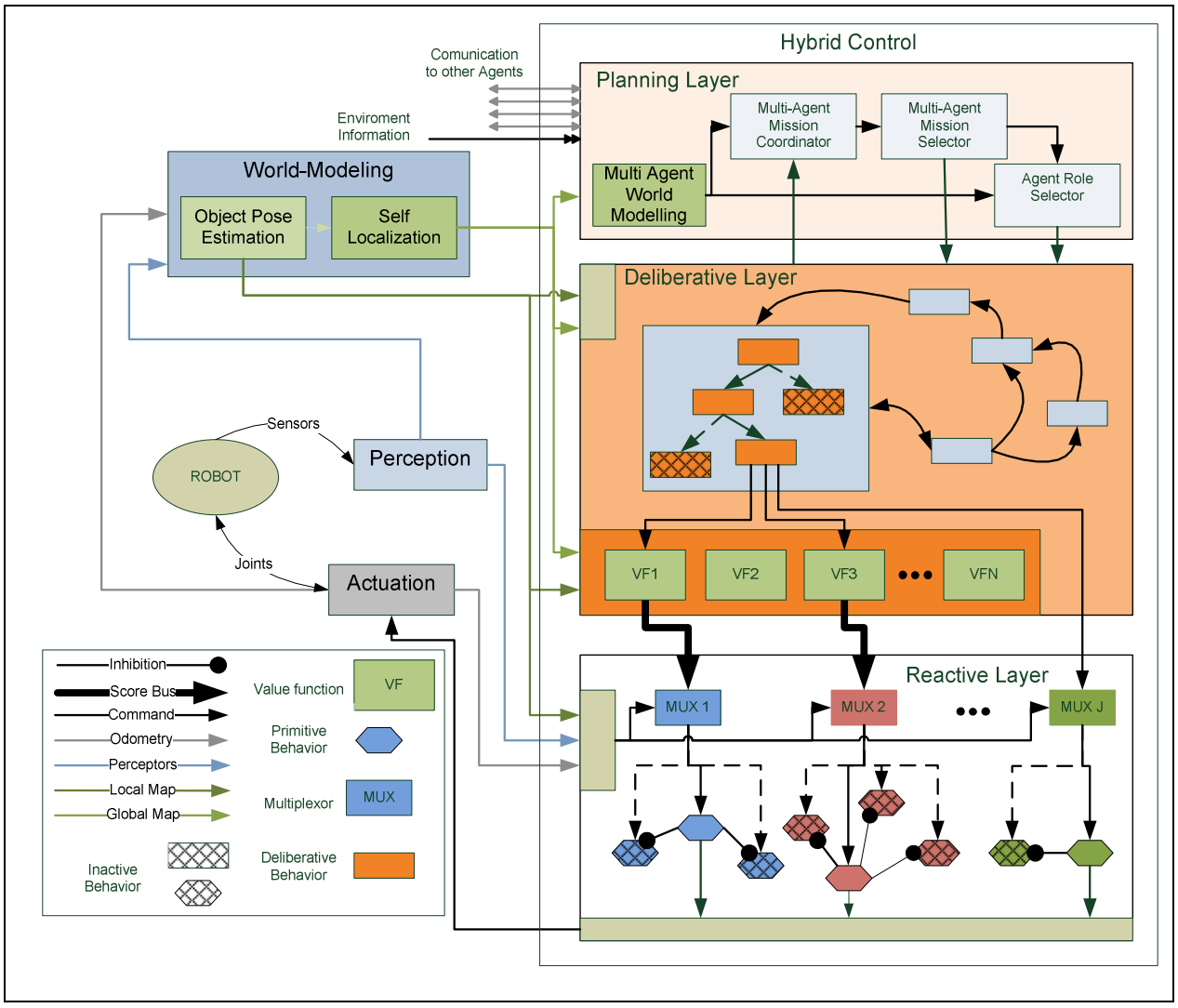


Figura 17: Diagrama completo de la arquitectura con los mecanismos de visión activa.

3.4 Implementación de Arquitectura Propuesta

A continuación, se entrega una explicación detallada del implementación de cada capa y sus características, además de las estructuras usadas para comunicarse entre ellas.

3.4.1 Capa Reactiva

Esta capa es la base del sistema. Procede conjuntamente, con *Actuation*, *Vision*, *World-Modelling* y la capa deliberativa. También contiene los comportamientos reactivos, además de multiplexores y es la que funciona a mayor frecuencia en el sistema. En la figura 18 se observa un diagrama detallado de las conexiones de los diferentes bloques internos que componen este nivel. En esta figura se observa el flujo de información existente en el nivel.

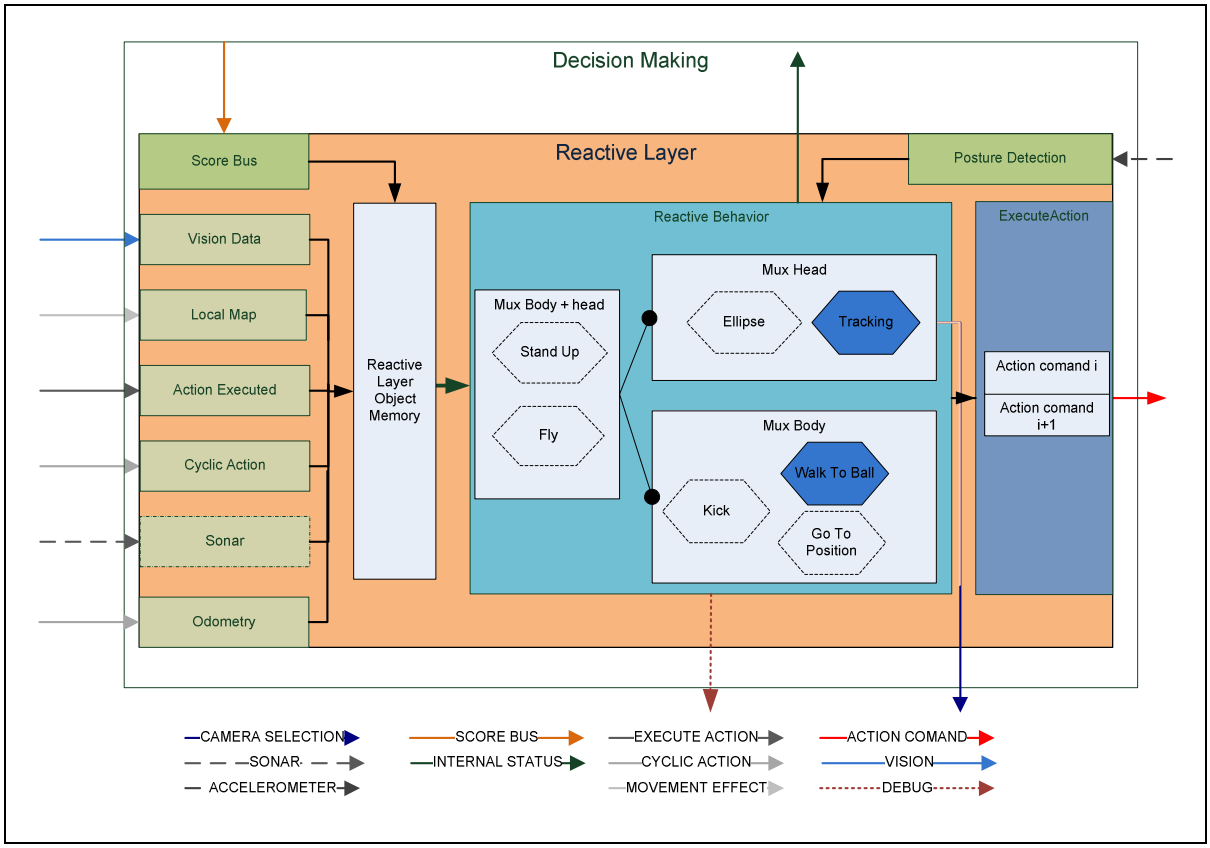


Figura 18: Diagrama de la capa reactiva.

A continuación, se explican sus componentes y distintas características.

3.4.1.1 Comunicaciones y Manejo de Información

Este módulo tiene tres tipos de manejo de mensajes.

- Tipo 1: consiste en actualizar la información que recibe desde los otros módulos y capas. En este caso lo que realiza la capa es actualizar la

información que dispone y dejarla vacante para que los comportamientos la usen.

- Tipo 2: consiste en recibir y actualizar la información, y además activar los comportamientos. Este tipo de conexión tiene por finalidad que el robot ejecute una acción con la información que reciba.
- Tipo 3: tiene una ejecución diferente a las anteriores, realizándose a través de punteros desde el sistema operativo. Esta comunicación se realiza de este modo, ya que es efectuada por el hardware del robot. Con esta comunicación se activan también los comportamientos del robot. Tiene la particularidad de no respetar lo que se ejecuta, interviniéndolo. Estas señales se pueden activar mediante un temporizador o por una señal emitida desde el sistema operativo.

La implementación de los tres tipos de comunicaciones tiene como finalidad jerarquizar las comunicaciones y permitir un mejor control en la ejecución de las rutinas. Como se mencionó anteriormente, las arquitecturas híbridas pueden tener complicaciones en cuanto a oscilaciones de información entre las capas, si no se ejerce algún mecanismo de control. Aquí se maneja la activación de la rutina de evaluación de comportamientos mediante los distintos tipos de comunicaciones.

A continuación, se describen las distintas comunicaciones que recibe esta capa:

- VisionData: Información que proviene desde la módulo *Vision* luego de ser procesada. Es del tipo 2.
- LocalMapData: Información con el mapa local que proviene desde *World-Modelling* apenas acaba de ser procesada. Es del tipo 1.
- Sonar: Está diseñada para cuando se use el sonar. Es del tipo 3.
- PostureDetection: Es la lectura de los acelerómetros y es del tipo 3. Se activa mediante un temporizador.
- ButtonPushed: Detecta cuando se presiona un botón, es del tipo 3.

- DeliberativeLevelInfo: Es la información proveniente desde la capa superior y contiene los *VA* y los *SV*. Es del tipo 1.

Las siguientes comunicaciones vienen desde *actuation*:

- MovementEffect: Es la odometría del robot y corresponde a la del tipo 1.
- ActionExecuted: Aviso de cuando una acción se ha ejecutado y libera los motores; viene de *actuation* y es del tipo 2.
- CyclicAction: Avisa cuando un movimiento que es cíclico puede ser modificado. Ésta se usa en el caso de la caminata y es del tipo 2.

Cabe recalcar que las conexiones del tipo 3 no se usan en el robot Hajime. Las salidas de la capa reactiva son conexiones a otros módulos y capas. Corresponden a las siguientes:

- ExecuteAction: Se envía nuevo comando a *actuation*.
- ReactiveLevelInfo: Envía notificaciones a la capa deliberativa en caso de caída y cuando se usan los botones del robot.
- CamaraSelect: Usada solo en el robot Nao para cambiar de cámara cuando se requiere.

Las estructuras de las comunicaciones son estructuras definidas con excepción de DeliberativeLayerInfo, la cual contiene la siguiente estructura dinámica en su interior como se observa en la figura 19. En esta figura se muestra gráficamente toda la información que se envía desde la capa deliberativa a la capa reactiva correspondiente a los *SV* y *VA*.

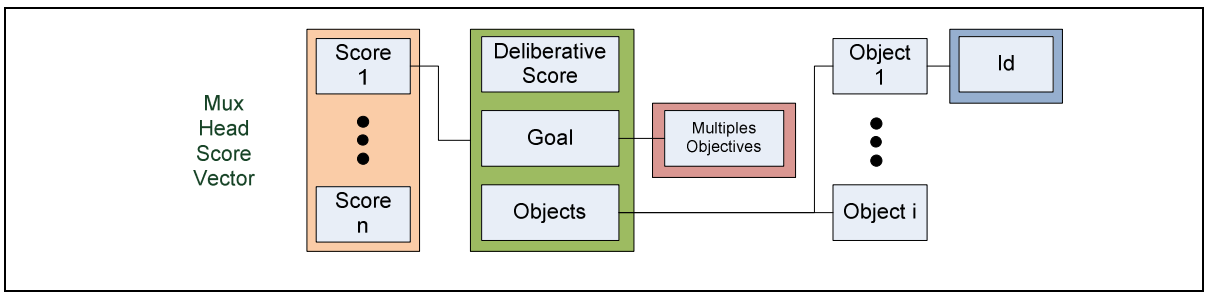


Figura 19: Estructura dinámica usada para enviar *VA* y *SV* entre la capa deliberativa y la reactiva.

Esta estructura está compuesta de elementos fijos y de otra parte dinámica que corresponde a los *SV* y *VA*. La parte dinámica se creó porque no se puede saber a priori qué cantidad de *SV* y *VA* serán enviados desde la capa deliberativa como se aprecia en la figura 19. La manera de manejar esta comunicación es a través de punteros a una clase padre, el cual se envía al otro thread y éste apunta al arreglo dinámico que contiene el resto de los elementos, los cuales heredan de la clase padre e implementan el objeto. El uso de la herencia, en este caso, permite que los objetos puedan ser distintos pero se manejen como si fueran idénticos. En un comienzo, se había optado por realizar la comunicación con una estructura fija, pero en la medida que los comportamientos fueran creciendo esto iba a ser una limitante para el sistema, por lo que se optó por elaborarlo de forma dinámica.

3.4.1.2 Interfaz con módulo Actuation

La interacción entre el nivel reactivo y *actuation* es considerada crítica, ya que influye en la capacidad de reacción del robot y, por lo tanto, deben controlarse los flujos y desfases que pueden producirse. Para poder controlar aquellos problemas, esta conexión consta de tres instancias: una conexión en que el nivel reactivo envía comandos y, dos conexiones con las que *actuation* notifica que puede ejecutar distintos tipos de comandos. El nivel reactivo -para coordinar de manera más eficiente los movimientos del robot- envía comando a los motores del cuerpo o a los de la cabeza. Aunque podría enviar comandos a un solo motor, esto puede producir descoordinaciones, por lo cual los comandos son agrupados como tales para el cuerpo o para la cabeza. Las otras dos conexiones son notificaciones que envía *actuation* para coordinar este flujo de comando al cuerpo o a la cabeza. Estos comandos para la cabeza o el cuerpo se envían de manera independiente, ya que no se interfieren en su ejecución. Por otro lado, los motores de la cabeza pueden recibir comandos a una frecuencia más rápida que los del cuerpo. Esto se debe básicamente porque la caminata requiere de una

secuencia de partida y término para todo desplazamiento. Cuando el robot envía un comando al cuerpo o a la cabeza, estos motores pasan a bloquearse. Cuando esto ocurre los comportamientos que envían nuevas órdenes a esos motores no se ejecutan, ya que éstas no pueden ser tomadas en cuenta por el robot aunque tenga nueva información. Para que el robot pueda moverse fluidamente, *actuation* puede recibir tres tipos de comandos:

- Comandos encolables: Son los que requieren continuidad y son necesarios para que el robot se mueva fluidamente. Los motores del robot no necesariamente llegan a la referencia que se les pide, pero como hay una cola de movimientos se asegura que puedan seguir moviéndose. Esta estrategia se usa para las caminatas, de manera que se pueda modificar la dirección de ellas una vez que ya se encuentra caminando.
- Comandos únicos: Son los que requieren precisión y se necesita que los motores lleguen a esa referencia exacta. En este caso se espera a que el motor haya llegado al punto para ejecutar el próximo movimiento. Éstos se usan en los movimientos de la cabeza, ya que se requiere que el robot mire a una posición determinada.
- Comandos fijos: Son una secuencia de movimientos fijos que se envían al robot. Una vez enviada la secuencia no puede ser modificada ni detenida. Esto se usa para los golpes y movimientos especiales, como las paradas.

La secuencia de comunicaciones entre los dos módulos comienza con un comando desde los comportamientos en el nivel reactivo, y dependiendo del tipo de comando, *actuation* responde con una de las dos notificaciones que ya puede volver a recibir un nuevo movimiento. Si es un comando único o fijo, envía la notificación cuando los motores están por llegar a la referencia, de manera que éstos se liberen en el nivel reactivo y los comportamientos que los modifican se ejecuten. En el caso de un comando encolable, lo que se realiza es diferente, ya que en este caso, una vez que el robot ya ejecutó la secuencia de partida de movimiento, notifica que el desplazamiento puede ser modificado, con lo cual

se liberan los motores del cuerpo en el nivel reactivo. Cuando esto ocurre, los comportamientos reactivos pueden volver a comenzar la secuencia descrita con otro comando. Estos nuevos movimientos se van encolando en los que había anteriormente, con lo cual el robot no pierde continuidad en ellos. Para evitar redundancia en las órdenes, éstas se van sumando a la distancia relativa a la cual se quiere enviar el robot, por lo que la caminata tiene siempre una posición a la cual converger. Por otro lado, el nivel reactivo recibe las odometrías de las caminatas del robot, con lo que en caso de enviar la misma posición, éstas descuentan la odometría de los movimientos ya ejecutados. Estas secuencias son acumulables hasta un cierto umbral que corresponde al movimiento mínimo que puede ejecutar el robot. Cuando este umbral es alcanzado, el robot ejecuta la secuencia de detención de la caminata, con lo cual se detiene por completo. Cuando se está ejecutando la secuencia de detención de la caminata, el nivel reactivo deja de recibir la notificación que el movimiento es modificable y debe esperar la notificación de que el movimiento terminó, quedando libres nuevamente los motores del cuerpo. Finalmente, cuando se envía un comando fijo, los motores que lo ejecutan quedan bloqueados, hasta que la secuencia está terminada completamente y sólo ahí se notifica al nivel reactivo que puede volver a recibir un movimiento.

Un caso especial en estas secuencias de comunicaciones es lo que ocurre cuando el robot detecta una caída, en ese caso, se ejecuta el comportamiento de caída y se activa la secuencia de parada del robot. Una vez efectuada esta secuencia, los comportamientos no son realizados hasta que *actuation* avisa que la secuencia está terminada.

En el caso del robot Hajime este esquema de comunicación no funciona de la misma manera, ya que en este robot el módulo *actuation* es sólo una interfaz para comunicarse a través del puerto serial a la controladora de los motores. En este robot no se corren los movimientos encolables ni tampoco se sabe desde la controladora cuándo un movimiento ha finalizado. Lo que sí se sabe es que ésta puede recibir una orden de caminata cada 400 [mS] y una orden de cabeza cada 40 [mS]. Con estos tiempos se implementaron *timers* en *actuation* que verifican

que se cumplan los ciclos de los comportamientos y estén sincronizados a ésta. En este robot las caídas son detectadas desde la controladora donde comienza la ejecución de la secuencia de parada. En el hardware del robot HR18 el esquema de la arquitectura parece no ser el óptimo, ya que el módulo de *actuation* no es más que un intérprete para la comunicación serial con la controladora, a diferencia de lo que ocurre en los robots Nao, pero se decidió no realizar cambios muy significativos entre ambas, con el fin de que sean similares y tengan compatibilidades.

3.4.1.3 Comportamientos Reactivos

En esta capa los comportamientos se manejan de tres modos: cuerpo, cabeza y ambos sincronizados (comportamiento del cuerpo completo). Salvo que se usen combinados, no hay problemas para que cuerpo y cabeza reciban comandos simultáneamente, ya que -como se explicó anteriormente- pueden funcionar de manera independiente. La secuencia de evaluación comienza revisando con los comportamientos los *VA* recibidos. Éstos, según la información perceptual y de los motores, ven su activación. En caso de venir un *VA* para un comportamiento de cuerpo completo y éste termina en una acción, se inhiben los comportamientos del cuerpo y la cabeza que pudiesen resultar activados. Con este procedimiento se le da prioridad a los comportamientos de cuerpo completo.

Se escogió usar la inhibición, ya que es una propuesta bastante usada en el desarrollo de comportamientos de robots y permite una rápida coordinación con el resto de los comportamientos del cuerpo y cabeza, bloqueando ambos grupos de motores, hasta que haya terminado el movimiento y, de este modo, no ocurran problemas de sincronismos. La inhibición consiste en que, el comportamiento que se activa, bloquea los otros. Antes de activarse un comportamiento de cuerpo completo, se debe esperar que se liberen los motores que están siendo ocupados. En el caso de la cabeza simplemente detiene el movimiento de ella, pero en el caso del cuerpo, el esquema tiene mayor complejidad, porque tiene que detener la secuencia de comandos que está

ejecutando el robot y luego, comenzar el movimiento. Es necesario realizar este procedimiento, dado que, en el caso que el robot esté caminando y se requiera “golpear”, se debe primeramente, “detener” al robot y luego “golpear”. En el caso de la caída, es más simple, ya que, al detectarla con los acelerómetros, se detiene automáticamente la caminata mediante un comando de cancelación. Cabe recordar que la lectura de los acelerómetros se realiza mediante un temporizador y es la única señal que puede detener la ejecución de los comportamientos. En el caso del robot Hajime, el esquema usado no es el mismo para la caída, porque esto es detectado al nivel del microcontrolador, pero sí para los golpes, con la salvedad que aquí se envía la orden del golpe, siendo el microcontrolador el que se preocupa de detener la caminata.

Cada grupo de motores tiene un multiplexor que maneja los parámetros recibidos desde la capa deliberativa y verifica que los puntajes de los objetivos contenidos en los *VA* recibidos sigan siendo válidos. Los *VA* vienen priorizados por la evaluación que realizan las funciones de valor de la capa deliberativa, y en la capa reactiva sólo se actualizan los puntajes con la información disponible. En caso de no haber ningún *VA* y, por lo tanto, tampoco objetivos enviados desde la capa deliberativa, se ejecuta la secuencia de comportamientos básicos. Esto corresponde a la secuencia de los comportamientos necesarios para “buscar la pelota”, “ir hacia ella” y “patearla hacia el frente del robot”, esto fue lo que se decidió que debía ejecutar el robot en caso de no estar la capa deliberativa funcionando. En caso de que llegara a haber un empate entre objetivos, éstos se evalúan priorizando los que usan la pelota. Ello se decidió para que el jugador siempre busque la pelota en sus objetivos. Se optó por elaborar este esquema para no realizar cálculos exhaustivos en la capa reactiva.

Todos los comportamientos programados en el nivel reactivo en esta tesis fueron:

- UCh_Bhv_HeadTracking: Es el comportamiento de seguimiento de objetos o posiciones con la cabeza. Requiere un objetivo que obtiene en

el *VA* para activarse, que en este caso puede ser una posición o la identificación de un objeto.

- UCh_Bhv_HeadEllipse: Es el comportamiento de realizar una secuencia de elipse con la cabeza del robot. El robot puede realizar 2 tipos de elipse, las cuales dependen del objetivo incluido en el *VA* que se les envíe.
- UCh_Bhv_GoToPosition: Comportamiento que sirve para que el robot ejecute una caminata y llegue a una posición relativa, la cual viene incluida en el *VA* que recibe.
- UCh_Bhv_Kick: Comportamiento que usa todo el cuerpo y envía el comando de golpe. Requiere recibir la identificación del golpe en el *VA* de activación para evaluarse.
- UCh_Bhv_KickPreparation: Comportamiento del cuerpo que va aproximando el robot a la pelota, dependiendo de un arco. Requiere recibir el arco rival en el *VA* para activarse.
- UCh_Bhv_StandUp: Comportamiento de todo el cuerpo para levantarse tras caerse. Este es especial, ya que se activa con una lectura de los acelerómetros y giróscopos.

Estos fueron los comportamientos reactivos que se diseñaron para que funcionaran en la capa reactiva. Cabe recalcar que éstos son los mínimos necesarios para el funcionamiento del robot. Además fue creado el concepto del multiplexor de la cabeza, el cual se explica a continuación.

3.4.1.4 Multiplexor Cabeza

El multiplexor de la cabeza es clave en el funcionamiento de la visión activa del robot, ya que se encarga de actualizar los puntajes del *ScoreVector* de manera que los *VA* queden disponibles para los comportamientos de la cabeza. El *mux*,

como se explicó anteriormente, fue diseñado para poder reflejar lo que ocurría en el ambiente, en las decisiones que toma la capa deliberativa cuando tiene más de un objetivo. En el caso de la cabeza, esto se aplica al concepto de la visión activa en la que el robot puede escoger entre observar a múltiples objetivos; o dirigir la mirada a un solo objetivo entre los que puede elegir. El *mux* se activa sólo en caso de que haya más de un objetivo predefinido por la capa deliberativa, ya que, en el caso contrario, no se hace necesaria su activación. Es lo que ocurre con el cuerpo, puesto que aún no se ha hecho necesario tener múltiples objetivos.

Para manejar la reactividad del robot se diferencia el puntaje obtenido en la capa deliberativa del de la reactiva. Esto se debe a que el puntaje de la capa reactiva se calcula con detecciones más ruidosas y, por lo tanto, tiene sólo un valor instantáneo que no es almacenado. En cambio, el puntaje deliberativo es más preciso y toma en consideración factores de largo plazo, como la trayectoria que va a seguir el robot. El puntaje deliberativo es guardado en memoria hasta que llegue uno nuevo, a diferencia del puntaje reactivo, que se calcula durante una iteración pero en la siguiente es vuelto a calcular sin tener en memoria el anterior. Con ambos puntajes se calcula el instantáneo, el cual es usado para tomar la decisión de qué objetivo realizar. Se puede definir, entonces, que un intervalo entre dos puntajes deliberativos consecutivos de tiempo es independiente a nivel de la capa reactiva de lo que ocurre en el intervalo siguiente y, por lo tanto, a nivel reactivo, sólo debe sortear cambios en el ambiente dentro de este intervalo. En el próximo puntaje deliberativo todos los cambios que ocurrieron desde el puntaje anterior ya están considerados en el nuevo.

En la Figura 20 se puede observar las diferencias entre el puntaje reactivo y el deliberativo. Este último tiene una frecuencia menor y esto se observa en que sus cambios se producen sólo en el punto 100 y 210, a diferencia de lo que ocurre en el reactivo, que cambian punto a punto.

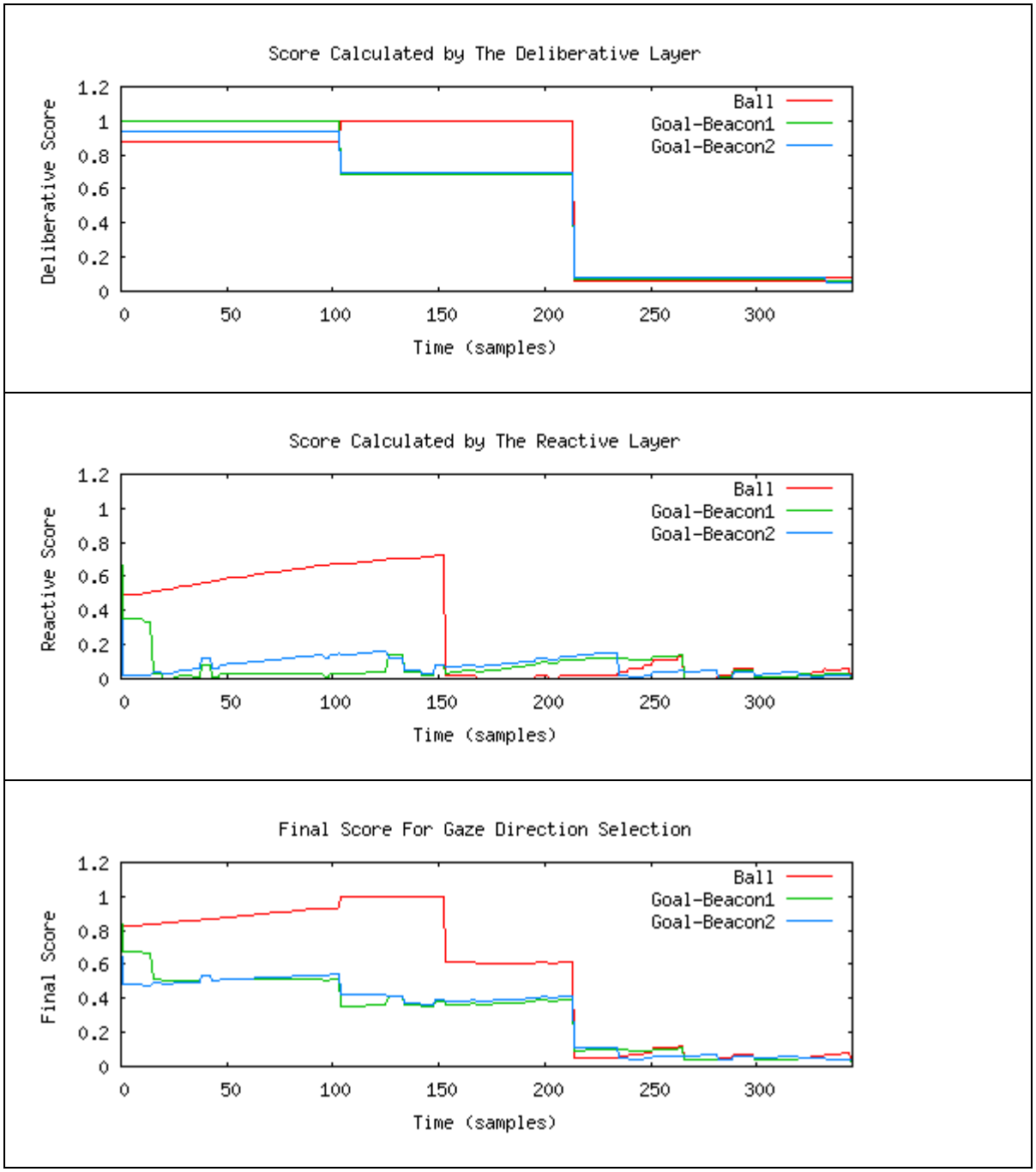


Figura 20: Concepto de puntaje deliberativo, reactivo y final

La actualización de los puntajes se realiza en cada uno de los *SV* y cabe recordar que éstos pueden ser también un grupo de objetos. En el caso de ser un solo objeto, se le aplica directamente la actualización del puntaje, pero en el caso contrario, se revisa primero que el grupo aún siga unido y, por lo tanto, que los

elementos que constituyen el *SV* sigan siendo visibles juntos. Para esto, se evalúa si el tiempo desde que fue visto el objeto se encuentra bajo un umbral específico de él y si la posición en la cual debiese estar es posible observarla en conjunto con los otros objetos del grupo. El tiempo desde que fue observado el objeto se obtiene desde la memoria visual. En el caso del tiempo, el umbral debe ser propio a cada objeto, ya que los objetos móviles deben tener umbrales más estrictos para reflejar la dinámica que detentan. Por ejemplo en el caso de los arcos, *World-Modelling* permite corregir la posición de uno respecto al otro y, por lo tanto, el no ver un arco específico en un tiempo grande no es crítico, ya que basta con ver uno para mejorar la posición del otro, a diferencia de lo que ocurre con la pelota. Por esta razón el umbral de tiempo para la pelota es menor que el de los arcos. La ubicación que se usa del objeto para determinar su posición, es la que se obtiene del mapa local dado por *World-Modelling*, ya que toma en consideración la odometría desde que fue observado el objeto, actualizándola. Se usan solamente el umbral de tiempo y la verificación de la posición del objeto, puesto que se determinó que ellos eran los más influyentes a la hora de verificar la consistencia de un grupo, y que por lo tanto el cálculo de la posición a la que debe apuntar la cámara, estaba influenciado mayoritariamente por estos factores. Básicamente, no se necesitan más, pues hay que recordar que estos filtros se usan por la capa reactiva sólo durante una iteración de la capa deliberativa, y en la siguiente, la información de cambios en el ambiente ya es considerada en la capa superior. Los elementos que no cumplen con el umbral de tiempo y el rango de visión son eliminados del grupo, pero no del *SV*, ya que antes se verifica que estén incluidos en otro grupo, y en caso de no estarlo, se agrega como objetivo individual en el *SV*. Una vez actualizado el grupo se pasa a calcular su puntaje reactivo. El puntaje reactivo se calcula usando los valores de las confidencias de los objetos almacenados en el mapa local de la capa reactiva. Las fórmulas con las cuales se actualizan los puntajes se muestran a continuación:

$$PR_i = \sum_{j \in G} \alpha_{obj} \times \left(1 - conf_{obj}\right) \qquad (1)$$

$$Pinst_i = \beta \times PR_i + (1 - \beta) \times PD_i \qquad (2)$$

$Donde,$

$PR_i: puntaje\ reactivo\ del\ i - ésimo\ grupo$

$G: total\ de\ objetos\ en\ el\ grupo$

$\alpha_{obj}: peso\ de\ objeto\ en\ el\ grupo$

$conf_{obj}: confidencia\ del\ objeto$

$Pinst_i: puntaje\ instantaneo\ del\ i - esimo\ grupo$

$PD_i: puntaje\ deliberativo\ del\ grupo$

$\beta: factor\ de\ influencia\ puntaje\ deliberativo$

La ecuación (1) representa el puntaje reactivo que es calculado en cada iteración de la capa reactiva. Esta ecuación fue escogida para actualizar los puntajes de la capa reactiva en función de la confidencia del objeto de manera que los objetos menos vistos sean priorizados. Los α_{obj} son dependientes de cada objeto y deben representar la dinámica de los objetos y de la tarea que se le está pidiendo al robot; esto quiere decir que, si es un objeto con movimiento su valor debe ser mayor que 1 y si es un objeto estático, entre 0 y 1. La ecuación (2) es para definir el puntaje final que permite definir el objetivo final a observar. En esta ecuación, el factor β influye en el peso que posee el puntaje deliberativo versus el reactivo. Un β muy pequeño significa que el robot se vuelve muy reactivo y, por lo tanto, cambia su objetivo frecuentemente. En el caso contrario, el robot deja de ser reactivo y no cambia los objetivos a mirar. Las confidencias de los objetos aumentan cuando éstos son vistos, dependiendo de cuán buena fue la calidad de la detección del objeto (la calidad está relacionada con la ubicación del objeto en la imagen y el tamaño del perceptor, y esto se hace en el módulo *vision*). El valor de la confidencia se determina en *World-Modelling* y es usado en el nivel reactivo para tener un índice de la confianza en la posición que se guarda del objeto. Este índice se había trabajado por el equipo y está integrado a las funcionalidades de la librería de software Uchile-Lib. Las confidencias están

diseñadas para que su valor vaya aumentando cuando se van obteniendo detecciones de un objeto. Dado que la confidencia aumenta su valor cuando un objeto es visto, el puntaje asociado a un objeto en un grupo o individualmente en el *mux* va a descender si éste es observado, permitiendo que el puntaje de los grupos de objetos no vistos vaya aumentando. Esto último es lo que posibilita que se produzcan cambios en los puntajes de los grupos calculados en el *mux*. Una vez actualizados los puntajes en el *mux,* el grupo de mayor puntaje instantáneo es escogido.

Cuando se realizaron las primeras pruebas de este sistema en el robot, se detectó que el sistema requería de una memoria de los objetos a los que había observado la cabeza. Esto ya que cuando se cambiaba de objetivo a mirar el sistema pasaba a realizar la búsqueda del objeto perdiendo dinamismo y continuidad en las acciones. Para resolver esto se decidió diseñar una máquina de selección de estados en el *mux* para forzar algunas selecciones de comportamientos cuando el objetivo a observar cambia. En caso de ser un objetivo nuevo se pasa a observar la última posición que se tenía disponible de éste y, en caso de ser un grupo, se pasa a calcular cuál es la posición a la que se debiese observar para ver los objetos. La máquina de selección de estado fuerza a que se realice seguimiento del nuevo objetivo, aún cuando no se cumplan las condiciones normales. Esto se realiza por algunos instantes y si el objetivo no es visto se pasa a realizar la búsqueda del objeto. Al tener memoria acerca de los estados de la cabeza, permite diferenciar la búsqueda a realizar de un objeto permitiendo que se optimice ya que fuerza ir a la posición previa, en memoria, antes de comenzar una búsqueda completa. En el caso de los grupos, no hay memoria, ya que es poco probable que un mismo grupo vuelva a generarse, y lo que se controla es la posición de la cabeza cuando se observa a una ubicación, no siendo todos los objetos observados. Lo que se busca disminuir son las oscilaciones entre las posiciones, privilegiando la estabilidad. Además esta máquina de estado permite controlar la interferencia de la cabeza y el cuerpo ya que maneja información directa del estado del cuerpo. En este caso la máquina de estado puede suavizar

un movimiento de la cabeza calculando puntos intermedios en vez de ir directamente al punto deseado.

3.4.2 Capa Deliberativa

La capa deliberativa, como se explicó anteriormente, es el eje central de la arquitectura, ya que entrega inteligencia al robot programando los objetivos a largo plazo. Esta capa interactúa principalmente con *World-Modelling* de donde recibe toda la información sensorial que usa. Además, recepciona información del estado del robot de la capa reactiva y el rol y la misión que debe ejecutar desde la capa de *planning*. A continuación se muestra en la figura 21 un diagrama de la capa deliberativa y sus estructuras más relevantes. En esta figura se muestran las conexiones que recepcionan información el almacenamiento que se realiza en la estructura de memoria deliberativa, y luego los bloques de comportamientos y el de las funciones de valor.

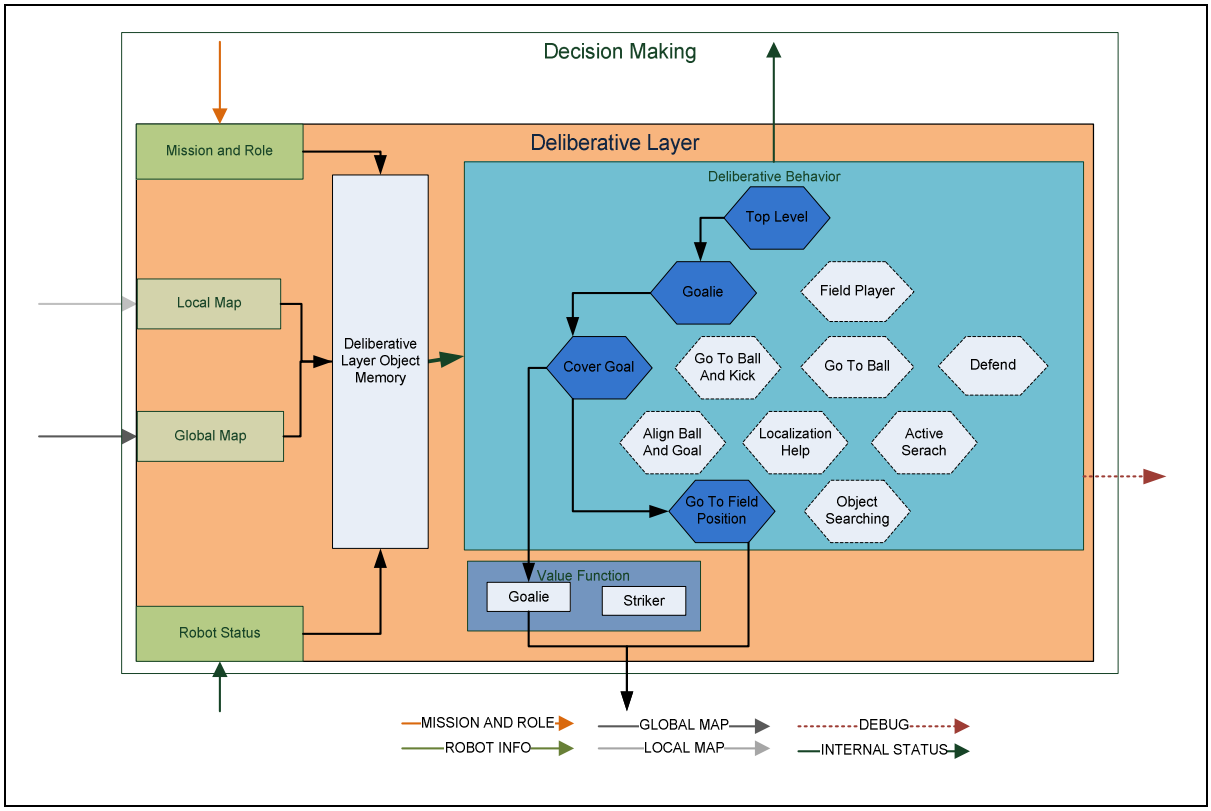


Figura 21: Diagrama de la capa deliberativa.

A continuación, se explican las distintas características de este nivel.

3.4.2.1 Comunicaciones y Manejo de Información

En esta capa la ejecución de los comportamientos está limitada en frecuencia. Esto debido fundamentalmente a que si esta capa funciona sin limitar su frecuencia, comienza a producirse un fenómeno de oscilación entre los comportamientos [34], básicamente por las histéresis entre las selecciones de los estados y las que se pueden generar entre capas, provocando que el sistema se colapse debido al intercambio de información repetida entre ellas [7]. El control de la oscilación entre capas se realizó, permitiendo que la ejecución de los comportamientos en la capa reactiva fuera independiente de la información enviada desde esta capa, lo cual posibilita que la ejecución de los comportamientos reactivos esté únicamente ligada a la información sensorial. Esto logra que la capa reactiva funcione desacoplada y a su propia frecuencia, no siendo alterada por la ejecución de la capa deliberativa. El control sobre las oscilaciones en las máquinas de estado se resuelve de dos formas; por un lado, usando información más procesada desde *World-Modelling,* y regulando la frecuencia de ejecución de los comportamientos. Esto último es bastante difícil de estimar, ya que determinar el tiempo que toma la ejecución de los comportamientos en el thread, no es directo, relacionándose estrechamente con la carga del procesador y, a su vez, dependiendo del resto del sistema (es sabido que el procesamiento de mayor información sensorial conlleva una mayor carga del procesador). Lo que sí se puede controlar es la frecuencia a la cual el thread ejecuta la secuencia de comportamientos. Esto requiere estimar qué frecuencia máxima se requiere que funcionen los comportamientos de la capa reactiva. Estos experimentos se mostrarán en la sección 4.1. Al reducir la frecuencia -si el tiempo de procesamiento es mayor al que se tiene fijado- produce que el thread no se detenga nunca, siendo su ejecución constante y dejando la administración de los recursos al sistema operativo. A nivel de sistema esto se puede ver reflejado en una mayor carga en el procesador, afectando al resto de los módulos. Sin embargo, reducir la frecuencia tiene el beneficio a nivel deliberativo, que las oscilaciones de los comportamientos disminuyen, ya que

estas son producidas por reprocesar los comportamientos con la misma información sensorial. Con la división del código en módulos, se tiene que, mientras haya threads con menos carga, el resto puede tener mayores tiempos de ejecución. Para aprovechar esto último, se definió que los threads asociados a actividades sensoriales (*vision*, *actuation*, nivel reactivo) tuvieran prioridad frente al resto. Este manejo de prioridades es realizado por el sistema operativo. En este nivel todas las comunicaciones son idénticas, salvo la que viene desde la capa reactiva, la cual pasa a ejecutar directamente los comportamientos, ya que trae información de caídas o que se presionaron los botones externos del robot y, en cuanto se recibe, provoca la ejecución de los comportamientos para cambiar los estados internos del robot. Las otras, en cambio, dejan la información disponible para la ejecución por tiempo de la capa deliberativa. Las comunicaciones de esta capa son las siguientes:

- LocalMapData: es la información del mapa local de objetos obtenida desde *World-Modelling*. La posición de los objetos viene en coordenadas relativas al robot.
- GlobalMapData: es la información del mapa global que tiene el robot. Incluye además, la información de su auto-localización y de los demás objetos detectados. La información viene en coordenadas globales de la cancha.
- PlanningLevelInfo: es la información proveniente de la capa de *planning* y consiste en la misión, rol y además del estado del juego del *GameController*[4].
- ReactiveLevelInfo: es la información proveniente desde el nivel reactivo y contiene el estatus interno del robot.

La salida que comunica con los otros niveles de estrategia del robot es:

[4] *GameController* es un software para manejar los partidos de robots en las competencias *RoboCup*. Éste permite que se envíe información a los robots del estado del partido (tiempo, resultado, robots en cancha) y además sirve para comunicar los robots que fueron penalizados. Este software transmite un broadcasting a todos los robots con la información constantemente.

- DeliberativeLevelInfo: es la información para activar los comportamientos de la capa reactiva, los cuales van como *VA* o *SV* En el caso que la comunicación vaya a la capa de *planning*, se envía la estructura ReactiveLevelInfo.

Las comunicaciones funcionan de la misma forma que los de la capa reactiva, puesto que eso es una funcionalidad incluida en la librería UChile-Lib.

3.4.2.2 Comportamientos Deliberativos

Los comportamientos deliberativos se estructuran como árbol de decisión con máquinas de estados. Se definió realizarlos de este modo, ya que la experiencia previa del equipo con este tipo de estructuras había mostrado buenos resultados en cuanto a su robustez, y los mayores problemas detectados se resolvían con la nueva arquitectura híbrida. Estos problemas eran la reevaluación constante del árbol completo en cada ciclo, y el no poder discriminar qué tipo de información usar en cada uno de los comportamientos. La mayor diferencia con respecto a los comportamientos reactivos radica en la estructura de los deliberativos, ya que son seleccionados por el comportamiento superior en el árbol y los reactivos están diseñados para competir por cuál es activado. Otra diferencia radica en que los comportamientos de esta capa dirigen el robot completo y no hacen diferencia de la cabeza o el cuerpo. Por esta razón, el no control de la frecuencia puede provocar que este nivel funcione más rápido, ya que no tiene ningún tipo de restricción desde el hardware, a diferencia de lo que ocurre en la capa reactiva con los motores.

El proceso de evaluación de los comportamientos deliberativos comienza con el de más alto nivel, el cual evalúa en base a su estado interno, misión y rol, un nuevo estado interno, con el cual se configuran los *VA* de los comportamientos que serán seleccionados. Una vez terminada la selección del comportamiento, se evalua el próximo comportamiento que se encuentra en el mismo nivel del anterior, repitiéndose el proceso. En caso de que no haya ningún comportamiento en el mismo nivel, se ejecuta el primero de los comportamientos seleccionados por los del nivel superior y éste repite el

proceso antes descrito y así sucesivamente. Esto se repite hasta que las ramas determinan objetivos mediante *VA* que deben ser ejecutados en el nivel reactivo. En caso de haber más de un *VA* para los mismos recursos, se les asocia una función de valor que los prioriza antes de enviarlos al *mux* de los motores como un solo *SV*.

Desde el punto de vista de la construcción de estos comportamientos, no hay problemas en cuanto a la carga computacional que tienen, a diferencia de lo que ocurría en la capa reactiva. Además, un mismo comportamiento puede ser reevaluado si su *VA* es diferente.

Los comportamientos disponibles en esta capa son:

- UChPLBhv_TopLevel: es el comportamiento de más alto nivel, y comienza en la parte superior del árbol de ejecución.
- UChPLBhv_Goalie: Comportamiento de alto nivel para el arquero.
- UChFieldPlayer: Comportamiento de alto nivel para los jugadores.
- UChPLBhv_GoToBall: Comportamiento de alto nivel para los atacantes.
- UChPLBhv_Defend: Comportamiento de alto nivel para los defensores.
- UChPLBhv_GoToBallAndKick: Selecciona la trayectoria y cuál de los golpes pueden activarse en la capa reactiva.
- UChPLBhv_ObjectSearching: Maneja los *VA* para coordinar estrategias de búsquedas de objetos.
- UChPLBhv_GoToFieldPosition: Sirve para llevar al robot a una posición determinada en la cancha.
- UChPLBhv_LocalizationHelp: Ayuda a mejorar la localización del robot.
- UChPLBhv_CoverGoal: Rutina del arquero para tapar el arco.
- UChPLBhv_AlignBallAndGoal: Alineación de la pelota con el arco para el arquero.

- UChPLBhv_ActiveSearch: Búsqueda activa de objetos; con esto se refiere a ir a buscar objetos en un determinado sector de la cancha.

Estos comportamientos son los correspondientes a la capa deliberativa. Cada vez que se pasa por una de las ramas de selección se debe llegar hasta enviar un *VA* a la capa reactiva. Éstos se pueden repetir al momento de ejecutarse si el comportamiento superior así lo determina. En caso de repetirse, sólo se examina que la configuración con la cual es ejecutado sea distinta a la que ya realizó para que no haya repetición en los cálculos hechos. El hecho de que se activen *VA* para un mismo recurso no es tomado en cuenta por los comportamientos deliberativos, ya que se deja su evaluación en manos de las funciones de valor.

En estos comportamientos la capa de planificación les define la misión y el rol, según lo cual determinan la manera de actuar. Un ejemplo de esto ocurre cuando un jugador tiene el rol de atacante o de defensa. En el rol de defensa, en caso de tener que ir a la pelota, el robot va derecho y lo único que toma en cuenta es no pegarle a la pelota en dirección al arco propio, a diferencia de lo que ocurre cuando es atacante, en donde intenta llegar a la pelota alineado.

3.4.2.3 Funciones de Valor

La creación del concepto de funciones de valor nace de la necesidad de contar con la opción de ejecutar múltiples objetivos y se inspira en el estudio de los diferentes ASM y los roles que cumplen. En efecto, como se explicó en las secciones anteriores, los ASM constituyen métodos para la selección de acciones. En las arquitecturas de comportamiento, los ASM permiten manejar múltiples objetivos y a continuación se presenta el diseño implementado para esta arquitectura. En este contexto, la selección de acciones está determinada por los comportamientos reactivos. Por lo tanto, se analizaron dos opciones para manejar objetivos múltiples; realizar un ASM clásico y arbitrar los comandos antes de la salida de la capa reactiva, o bien efectuarlo en algún nivel superior y manejarlo a nivel de la estrategia a calcular. Tener el ASM para unir comportamientos en el nivel inferior -que es el de las acciones a ejecutar por el robot- posee el riesgo de que el robot deje de ser reactivo, ya que podría llegar a

ser costoso operacionalmente manejar varios objetivos en este nivel. Por otro lado manejar múltiples tareas en un nivel superior presenta la ventaja de que se pueden obtener ahorros computacionales, al no activarse los comportamientos inferiores para calcular las acciones finales del robot. Habiendo realizado este análisis, lo que se diseñó fue la conducción de los múltiples objetivos y posibles fusiones a nivel de la capa deliberativa, usando sólo los objetivos y no las acciones del robot. En términos de esta arquitectura, lo que se plantea es priorizar y fusionar los objetivos contenidos en los *VA* que dan pie a la salida de la capa deliberativa. Los comportamientos reactivos están diseñados sólo para manipular un único *VA*, pero a la vez deben lograr manejar múltiples objetivos. Finalmente, la solución creada consiste en el bloque de función de valor y su tarea es solucionar estos problemas de dos formas: por un lado, puntuar los diferentes *VA*, en caso que haya más de uno para un determinado grupo de objetivos y además, ver la opción de unir objetivos que puedan llevarse a cabo de manera conjunta. El puntuar los distintos objetivos debe sopesar dos cosas: la posibilidad que se logre ejecutar la acción que se desea y, por otra parte, el valor esperado que aporta esa acción al estado del juego. Para esto, la función de valor se divide en dos etapas. Primeramente, realizar una simulación de lo que ocurrirá y, en segundo lugar, la evaluación del beneficio obtenido en el estado interno, o de juego, de la simulación efectuada. Una opción de lograr simular el ambiente, es usando un filtro de Kalman, que simula el estado futuro al que el robot llegaría si ejecutase una determinada acción y luego, utilizando la simulación , se evalúa si esto resulta favorable con la misión a cumplir. En efecto, al proceder con la simulación se obtiene un estado posible al cual puede llegar el robot y éste es finalmente evaluado por la función de valor, asignándose un puntaje al objetivo que llevó a ese estado. Esta función de valor depende de la misión y rol que tiene el robot asignado, ya que, en base a esto, se determina qué coeficiente se usa para valorar el estado futuro. Otra manera de realizar simulaciones es usando probabilidades y filtros de partículas. Este enfoque consiste en evaluar la probabilidad de que dado el estado actual se llegue a un estado futuro, logrando concretarse una determinada acción. Dado

que hay probabilidades asociadas al resultado de una acción, se usa el filtro de partículas para simular una distribución de probabilidad y así obtener un estado futuro. En él es necesario tener una estimación de la distribución de probabilidad del resultado que genera una determinada acción que se quiere ejecutar. Para obtener esto último, es importante proceder con numerosos experimentos con el robot y aún así puede no ser suficiente, ya que no es posible asegurar que se recorran todos los eventuales escenarios.

En caso de tener dos objetivos evaluados con el mismo puntaje, se procede a recalcular éste con un factor de aumento a los objetivos que tengan la pelota. Esto se hizo para que dirigir la mirada hacia la pelota siempre fuese lo que más se favoreciera en la ejecución de los comportamientos.

3.4.3 Capa Planificación

Esta es la capa superior de la arquitectura y está diseñada para ser la interfaz de comunicaciones del robot. Aquí, se genera, recibe y administra la comunicación entre robots. Además, se recepciona la comunicación de estado del juego, enviada desde un servidor, en donde se lleva el estatus del partido. En este nivel no hay comportamientos, a diferencia de lo que ocurría en los niveles inferiores y está constituido por bloques diseñados para interactuar entre ellos como se observa en la figura 22. En esta figura se muestran las conexiones internas entre los bloques del módulo.

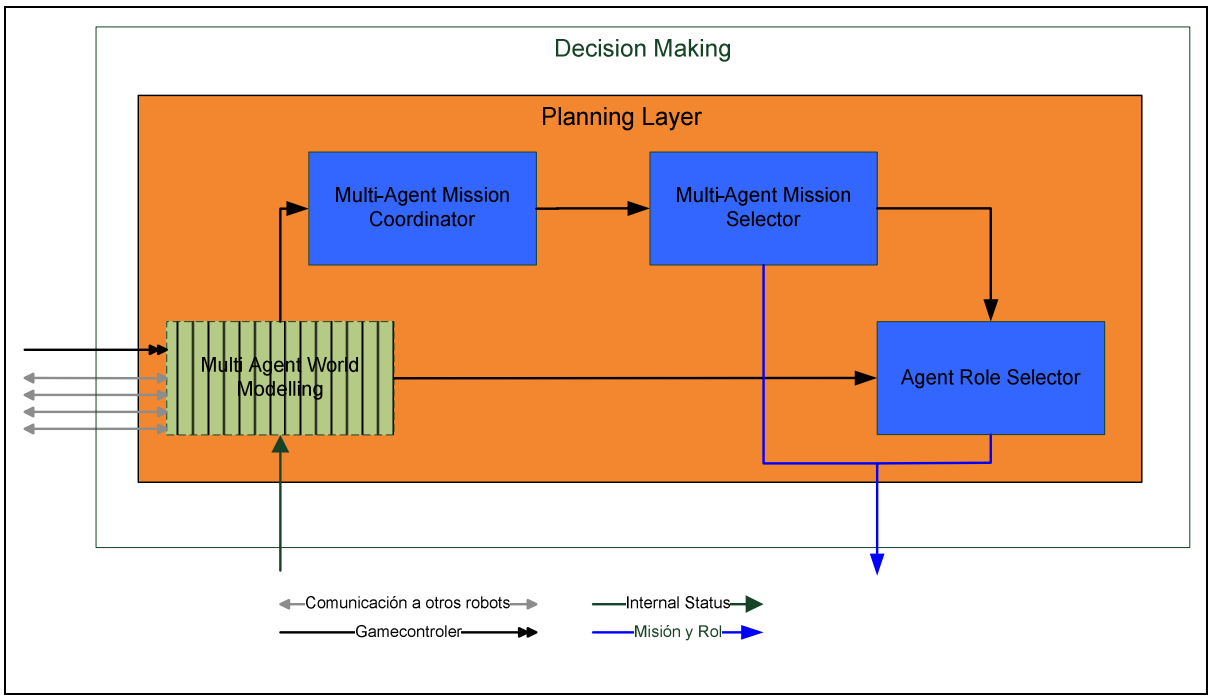


Figura 22: Diagrama de la capa de planificación. *Multi Agent World-Modelling* aparece tachado, ya que aún no se ha construido este bloque.

La comunicación entre distintos agentes puede ser muy beneficiosa, pero también perjudicial, si ocurren interrupciones inevitables en el canal de transmisión, y éstas no son tomadas en cuenta. Por ejemplo, durante las competencias de *RoboCup*, las comunicaciones suelen ser un problema, puesto que, generalmente están saturadas y afectan notoriamente el desempeño de los robots. En consecuencia, el primer principio escogido para minimizar estos últimos problemas es tener interacción entre agentes sólo en el nivel superior del módulo de control híbrido, para que, en caso que se pierda la comunicación, el robot no desaproveche sus propias habilidades. Un segundo principio es limitar la cantidad y la frecuencia de los datos transmitidos entre los robots; únicamente información relevante debiese transmitirse. Por lo tanto, sólo el estado interno, la misión y rol son comunicados. Por último, el tercer principio corresponde a transmitir el modelo del mundo propio del robot, acompañado de una medida de su confidencia. Para realizar esto, el modelo del mundo de cada robot es transmitido con una marca temporal y, además, con información acerca de la

media y covarianza en la posición de cada objeto, estimada en el mapa global. La coordinación entre agentes asume que los robots son homogéneos: todos tienen capacidades sensoriales y motores similares. Además, en el caso del fútbol robótico, los robots se mueven en un ambiente conocido, permitiendo que usen un mismo sistema de coordenadas globales.

3.4.3.1 Comunicaciones y Manejo de la Información

En esta capa las comunicaciones diseñadas son en su mayoría para conectar el módulo con el exterior vía socket. Las comunicaciones internas están dadas para manejar las distintas interacciones de este módulo con el exterior. En este módulo existe un control de frecuencia diseñado para manejar el envío de información a los otros robots. Las comunicaciones de esta capa son las siguientes:

- TeamLevelInfo: Información que se transmite al nivel deliberativo que contiene el estado del juego, misión y roles que debe cumplir el robot. Esta estructura es fija en tamaño y siempre envía la misma información.
- GameStateAndPlayerIdentity: Información del estado del juego e identidad del robot enviada a *World-Modelling.*
- DeliberativeLevelInfo: Información de estatus del robot recibida desde la capa deliberativa.
- GlobalMap: Mapa de la localización del robot recibido desde *World-Modelling.*

Las siguientes comunicaciones se realizan vía sockets:

- *GameController*: Información obtenida desde el servidor de juego de la liga. Esta información se recibe constantemente, aunque no contenga ningún cambio, por lo que se filtra y propaga únicamente cuando contiene cambios.

- PlayerData: Información que se comparte entre los distintos miembros del equipo. Incluye datos de la localización, el estado interno, la misión y el rol del robot.

En este módulo se controla la frecuencia con la que se envían datos, la cual es cada 20 [s]. La limitación en la frecuencia de transmisión del robot estuvo siempre entre los requerimientos del equipo, ya que previamente se había detectado que no era necesario que los robots se comunicaran en cada momento. Además, en el ambiente de las competencias, la mayor parte del tiempo hay saturación de las redes y, por lo tanto, esto conlleva a que se generen mayores problemas de comunicación, lo que puede llegar a producir consumos no deseados del procesador por parte de la comunicación. En consideración a ello, se usan comunicaciones entre robots de tipo UDP y usando broadcasting, de manera que la información de cada robot sea enviada con una sola escritura en el socket. Las comunicaciones recibidas del equipo son únicamente aquellas que están correctas (dentro del protocolo de comunicación se hace manejo de errores) y cuando son recibidas, se ejecutan los bloques correspondientes a esta capa. En este caso, la frecuencia de ejecución no es constante y es bastante irregular, debido a que las comunicaciones son completamente asincrónicas. En consecuencia, puede ocurrir que se ejecuten dos iteraciones muy seguidas y luego se pase a un periodo de inactividad mayor.

Acerca de la comunicación entre robots, desde siempre se pensó que ésta fuera una estructura única y no múltiples comunicaciones, porque ayuda a lograr un mejor control sobre el tráfico en la red. Por último, en referencia a las comunicaciones internas, se establece que la comunicación proveniente de *World-Modelling* sólo actualiza con nueva información; es decir, no lleva a la ejecución de ningún bloque de la capa, a diferencia de la del nivel deliberativo que sí lo hace, ya que trae información del estatus interno del robot. En resumen, en este nivel sólo las comunicaciones externas y la proveniente del nivel inferior producen una reevaluación de la estrategia de planificación.

3.4.3.2 Descripción de Bloques Internos

La mayor diferencia entre la estructura de esta capa y las otras, es que ésta no posee comportamientos como se observa en la figura 22. La razón por la cual no se implementaron comportamientos a este nivel fue básicamente porque aún el grado de tareas que pueden realizar en conjunto es limitado. Como antecedente, se tiene que en los robots cuadrúpedos *Aibo*, aún más dinámicos que los humanoides, no se logró obtener en toda la liga comportamientos complejos de equipo. Con este antecedente, y conociendo las limitantes de los robots humanoides, se determinó que era mejor no implementar un complejo sistema de comportamientos, sino más bien estructurar un protocolo para manejar las comunicaciones, de manera que éstas afecten lo menos posible al robot. Esto, porque se tenía por experiencia que una información errónea en la comunicación afecta más que no tenerla. Es que es preferible no tener la comunicación de los demás robots a tenerla de mala calidad, lo cual se puede explicar, por ejemplo, con un robot que, con mala percepción, le dice a otro que está más cerca de la pelota, cuando ello no es efectivo, impidiendo que el más cercano vaya a la pelota, o bien, que se produzca una oscilación entre la idea de un robot y luego la del otro, y así sucesivamente. Esto último se vio varias veces en las competencias con robots *Aibo* en donde, por problema de comunicaciones, ningún robot iba a la pelota o tomaba demasiado tiempo. Por otro lado, el tener comunicaciones resulta atractivo por el potencial que se puede lograr usándolas. Por esto, lo que se diseñó para esta capa consiste en tres módulos que interactúan entre ellos.

El primero de estos bloques es el *Multi-Agent Mission Coordinator* (MMC) y su función es detectar posibles problemas en las comunicaciones. En efecto, este bloque lleva estadísticas de todas las comunicaciones que recibe el robot y, además, monitorea su información interna. Las estadísticas que maneja son el tiempo desde la última comunicación, cantidad de mensajes transmitidos desde cada robot y el tiempo promedio entre comunicaciones. De este modo, el bloque MMC recibe comunicaciones y la salida del bloque

es una lista con los robots del equipo que están disponibles para realizar misiones. Cabe recalcar que en caso que este bloque notifique que sólo el mismo robot está habilitado, se considera que no hay comunicaciones y, por tanto, la misión y rol que cumple el robot es una predeterminada por el archivo de configuración de la estrategia. Esto se hace para que el robot logre hacer frente a problemas en las comunicaciones. Otro punto importante a recalcar es que, en ningún caso este bloque usa la información contenida en la comunicación, sino que su función es ser un filtro para el posterior uso de ésta. Dentro de este bloque se realizan las siguientes inhabilitaciones a los robots:

- Inhabilitación por time-out: Si la comunicación de un robot demora más que un umbral determinado, se considera no disponible para ninguna misión. Lo que se sopesó para crear esta regla es que algún robot tenga alguna falla en las comunicaciones y, con esto, se desconecta de los otros. Para estos efectos, el sistema usa su propio reloj interno, tomando como tiempo para medir esto, el que recibió la comunicación del otro robot.
- Inhabilitación interna: Es propagada a través de todos los niveles de la estrategia, y ocurre en caso de que el robot se caiga o sea presionado el botón de detención. Cuando llega esta información se envía inmediatamente a los otros robots.
- Inhabilitación externa: Ocurre cuando el robot es penalizado mediante el *GameController* de la liga. Esta comunicación llega en todo momento a los distintos robots y, por tanto, no requiere ser reenviada.

Para ser habilitados en el receptor, se necesita que el robot se comunique una vez. Estos bloques funcionan de manera independiente en cada uno de los robots.

El siguiente bloque es el *Multi-Agent Mission Selector* (MMS) que es el que se encarga de generar las posibles misiones del robot. Está contemplado en el diseño como la etapa siguiente al MMC y consiste en generar las misiones

para los robots. Para realizar esto, verifica la lista de los robots disponibles, obtenida desde el MMC y, en base a la información de las distintas localizaciones y de las misiones de los otros robots, se generan misiones. Para que no haya problemas por oscilación en los cambios de misión, el robot siempre toma la que es enviada por otro robot, salvo que haya cambios en las disponibilidades de los robots, con lo cual evalúa una nueva misión. Las misiones proyectadas para el robot son las que se pueden ver en la tabla 1.

Por último, el bloque llamado *Agent Rol Selection* (ARS), está encargado de manejar el rol que le corresponde al robot dentro de la misión. Este bloque, al igual que el anterior, usa la información que envían los otros robots, pero evalúa el rol en base a sus condiciones. En efecto, puede darse que dos robots vayan a la pelota al mismo tiempo, pero eso se da hasta un punto en el cual haya uno más cercano y el otro active el rol de Defensor.

Cabe recalcar que el bloque *Multi Agent World Modelling* (MAWM) no se encuentra implementado aún, lo cual no excluye que fue diseñado para su futura implementación. La labor de este modulo es fusionar de mejor forma las comunicaciones [41]. Éste se dejo de lado por motivos de tiempo en su realización.

Tabla 1: Descripción de misiones y roles

Nombre misiones	**Descripción**	**Nombre Roles**	**Descripción**
Defender el arco	El arquero trata de impedir que el equipo contrario marque un gol.	Arquero	El arquero es un rol especial, ya que sólo un robot puede serlo durante todo el partido.
Ataque	Todos los jugadores del equipo intentan hacer un gol.	Ataque	El robot atacante es el más cercano a la pelota y éste puede llegar a pegarle.
		Defensor	El robot más lejano a la pelota y que queda en una posición defensiva.
		Receptor de Pase	Robot espera un pase en una posición libre en la cancha.
Defensa	Todos los jugadores del equipo intentan defender el arco.	Defensa	El robot más cercano a la posición de la pelota.
		Defensa a posición	El robot más lejano que se posiciona en un sector de la cancha para defender.

IV. EXPERIMENTOS Y RESULTADOS

A continuación, se describen las diferentes pruebas de la arquitectura, realizadas en el ambiente simulado y en el robot. El trabajo comenzó por la adecuación de los comportamientos existentes en el robot, con el fin de que funcionaran con la nueva arquitectura. Una vez que se logró esto en el simulador, se procedió a ajustarlo para su funcionamiento en el robot real. Los experimentos consistieron en elaborar una misión, como atacar, que se realizaba con la arquitectura anterior, usando la arquitectura nueva. Cuando esto se cumplió, se adecuaron los comportamientos para usar visión activa en las misiones. En visión activa se efectuaron numerosos experimentos en el simulador, los que consistieron en dejar al robot ejecutando una misión y mover la pelota en una trayectoria predefinida, observando si el robot lograba “no perder la pelota”. Estos experimentos se realizaron solamente en el simulador. En el robot, para este caso, se evaluó la manera en que ejecutó la misión y cómo estaba la reactividad de éste mientras realizaba estas tareas.

4.1 AJUSTE DE COMPORTAMIENTOS A LA ARQUITECTURA PROPUESTA

El ajuste de los comportamientos se realizó bajo un enfoque bottom-up, debido a que los comportamientos básicos son los bloques sobre los que se construyen comportamientos más complejos. El criterio usado con éstos es que pasaran directamente a la capa reactiva. Con este concepto las modificaciones que sufrieron fueron desde el punto de vista de cómo se seleccionaban, ya que anteriormente eran escogidos mediante el comportamiento superior. El cambio que se realizó fue permitir que la lista de comportamientos reactivos disponibles, tanto para la cabeza como para el cuerpo, se revisara según los *VA* recibidos desde la capa deliberativa. En un principio, el uso de los múltiples objetivos no se incorporó, por lo que siempre se tendría un solo *VA* por grupo de motores a lo más, y el *VA* permite su identificación ya que incluye el grupo de motores en su interior. La rutina que tiene la capa reactiva, para revisar los comportamientos reactivos es la siguiente:

(1)Para todos los VA disponibles

(2) Seleccionar todos los comportamientos correspondientes

(3) Revisar comportamiento seleccionado

(4) Enviar comando de comportamientos activados a actuation

Este pseudocódigo explica la rutina que realiza la capa reactiva para revisar los comportamientos. Las reglas para decidir la activación de los comportamientos son específicas a cada comportamiento y grupo de motores.

El criterio de activación para los nuevos comportamientos de la cabeza son los siguientes y se pueden interpretar gráficamente en la figura 23:

- Puntaje de la confidencia del objetivo a mirar C: se define que éste es un buen criterio de activación para los objetos, en la medida que la confidencia sea confiable. Además, es un buen umbral en cuanto asegura que es 0 si el objeto no se ha visto nunca.
- Tiempo desde la última vez que se observó el objeto T: este parámetro permite controlar la visión específica al objeto del que se desea observar.

Con estos dos criterios se crea un área en que corresponde la activación del *seguimiento del objeto* y donde concierne la activación de la *búsqueda*. Los umbrales de estas dos variables dependen de cada uno de ellas y varían según cada objetivo. El comportamiento que se activa inhibe al otro, lo que, en este caso, corresponde a que uno de los dos se active únicamente.

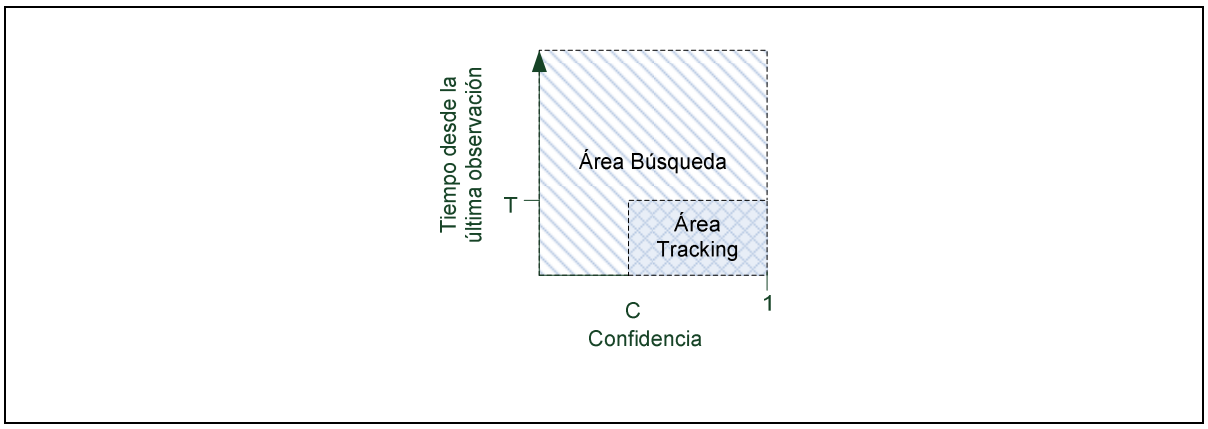


Figura 23: Diagrama de área de activación del tracking. Está determinada por el tiempo desde la última observación y la confidencia del objeto.

Para el cuerpo, las reglas de activación cambian un poco, ya que se tienen tres comportamientos que pueden ser activados, por lo que las reglas comienzan a complicarse para la activación de uno respecto a otro. En este caso, la regla para la activación viene dada por la prioridad en que se recorren los comportamientos del cuerpo. Este último poseía 3 comportamientos; el de "ir a una posición", "ir a la pelota y golpear". El de "levantarse" es especial, ya que se activa bajo un estado diferente. Dado lo que se generó, en este caso, es que se evalúa desde el más reactivo al menos. Se definió este esquema, ya que así el más reactivo bloquea instantáneamente al resto. Cabe recalcar que esto tiene la desventaja que, para un número grande de comportamientos, esto no es simple de realizar, sin embargo, dado que esto únicamente está ocurriendo en este caso para los comportamientos básicos, es posible priorizarlos. Además, se le agregó la flexibilidad que el nivel deliberativo inhibe un comportamiento al no agregarlo en el *VA* y, de esta manera, modifica las prioridades.

En el caso del cuerpo, los comportamientos requieren de las siguientes configuraciones:

- Posición relativa: esto no es un criterio en sí, pero es una posición en la cancha y se usa para definir la ubicación a llegar por el comportamiento de ir a un lugar.
- Objetivo: si está incluido en la comunicación, se usa para tomar la posición del objeto como referencia de la posición a llegar.
- Golpe seleccionado: sólo si hay algún golpe seleccionado.

Usando estas configuraciones se tienen los siguientes criterios de configuración:

- Distancia a pelota: esta distancia relativa se usa para determinar si el acercamiento comienza a requerir de mayor precisión y se va realizando paso a paso, a una velocidad de caminata inferior.
- Posición de la pelota: con la posición relativa de la pelota, se verifica si ésta se encuentra en el área en que el golpe seleccionado tiene efectividad y logra golpearla.

En el caso del cuerpo, la mayor complicación resultó en la coordinación con los comportamientos de la cabeza. Esto se da, ya que cuando el robot se encuentra buscando la pelota, ejecuta elipses de búsquedas con la cabeza y, al mismo tiempo, efectúa giros con el cuerpo para lograr cubrir un área mayor de búsqueda. Lo que se realizó para coordinar esto último, fue crear notificaciones entre los comportamientos de la cabeza y los del cuerpo, con lo cual el cuerpo podía saber si la cabeza estaba terminando o no una búsqueda en una posición, con el fin de ejecutar el movimiento de posición.

En este escenario, en caso de ser detectada la pelota, la cabeza pasa directamente a realizar el seguimiento de ésta y el robot a caminar hacia ella.

En el contexto de la capa deliberativa, la estructura de toma de decisiones es mediante la selección piramidal de los comportamientos; esto quiere decir que comienza con uno de nivel superior y va bajando, dependiendo de los estados activados en éstos y así, sucesivamente. Como se dijo anteriormente en la figura 7, la repetición de un mismo comportamiento es factible si tiene los parámetros cambiados y, por último, la cadena de selección de comportamientos termina con un *VA.* Para adaptar estos comportamientos, que corresponden a conductas de alto nivel, de atacante, de buscar la pelota, de ir a la pelota y el de ir a la pelota y pegarle fue necesario modificar las máquinas de estados que éstos tenían. Los cambios realizados consistieron en eliminar el uso de umbrales de tiempo, ya que ahora, en la capa deliberativa, no están en sintonía con los tiempos que tiene el robot, además de cambiar los usos de información visual por la del *local map* de *World-Modelling*. Los umbrales de tiempo pasaron a ser reemplazados por el uso de las confidencias. Estas nuevas condiciones llevaron a

modificar las máquinas de estados de los comportamientos antiguos quedando algunos completamente diferentes.

El cambio de los comportamientos se realizó, en primera instancia, usando HLSIM. Su uso permitió ver la activación de los distintos comportamientos y que se cumplieran los objetivos planteados en esta tesis, respecto a la reactividad del robot. Se pudo simular que el robot cumpliera con lo que se le pedía: que fuera “buscar la pelota”, “ir a ella” y “golpearla”. En estos experimentos lo que se privilegió fue la robustez de la arquitectura, por lo que las pruebas consistieron en un simple ajuste de umbrales y luego, dejar largo tiempo de simulación para asegurar que no hubiera problemas en el código.

4.1.1 Pruebas en el Robot

La siguiente etapa consistió en probar la nueva arquitectura funcionando en el robot y que lograra ejecutar lo que se obtenía en el simulador. Este paso fue especialmente largo, ya que los robots venían de un extenso tiempo de inactividad y los motores estaban dañados. En el proceso de lograr el objetivo se constató que el funcionamiento de las confidencias y de los tiempos es completamente distinto en el robot que en el simulador. El objetivo del robot de “buscar la pelota”, “ir hacia ella” y “pegarle” se logró de manera satisfactoria. Esta prueba fue considerada clave para el funcionamiento del robot y marcó un hito en el desarrollo de la tesis, y sus resultados se encuentran en la tabla 2. Lo importante del experimento es porque en éste se encuentran en uso todas las habilidades básicas que requiere el robot para poder jugar fútbol. En estas pruebas lo que se comprobó fue el funcionamiento del *tracking* a la pelota, evaluando la velocidad de respuesta del robot frente a cambios en la posición de la pelota y el acercamiento final a ella. Por último, se comprobó, además, que la detección de la caída y las interfaces externas funcionaran correctamente.

En estas pruebas la capa de planificación funcionó tan solo como interfaz para poder manejar el robot mediante el *GameController* y poder realizar mejor las pruebas.

En estos tests se limitó además la frecuencia de la capa deliberativa; la finalidad era comprobar hasta qué frecuencia se podía evaluar esta capa, sin que la reactividad se viera afectada. Para esto, se limitó la frecuencia en distintos valores para ver cómo esto afectaba al robot y cuáles eran los comportamientos observados. Estos experimentos se realizan repitiendo el mismo procedimiento una cantidad determinada de veces y luego, tomando estadísticas de si logró o no el objetivo. Estos experimentos son efectuados colocando la pelota en diferentes partes de la cancha y se repiten al menos 10 veces, tratando que sean idénticos los resultados se encuentran en la tabla 2. Estas pruebas son estándares dentro del equipo y se realizan cada vez que hay cambios en el robot para asegurar que los cambios son mejoras en el funcionamiento de éste. Además, para estos experimentos se usó la posición de la pelota alimentada desde la capa deliberativa, para ver el efecto que esto producía en el robot.

Tabla 2: Evaluación de las distintas habilidades con diferentes frecuencias de funcionamiento de la capa deliberativa. El color verde significa sobre 80% de efectividad, el amarillo entre 80% y 50% y el rojo bajo 50%.

Frecuencia capa deliberativa [Hz]	Reactividad Tracking	Reacción cambio de posición pelota	Acercamiento a la pelota	Efectividad golpe
0.001				
0.1				
0.5				
1				
2				
5				

De la tabla 2 con los resultados de estas pruebas, las conclusiones que se pueden obtener son:

El cuerpo del robot se puede observar con movimiento fluido si percibe cambios con un periodo mayor a 500 [ms], siendo este último valor algo que ya se había detectado previamente.

En el caso de los movimientos que requieren mayor precisión, mientras más rápida sea la entrega de información, mejor es el desempeño.

El tracking no se vio afectado, ya que la capa deliberativa para este caso sólo fija el objetivo a mirar y no la posición de los ángulos a los cuales debe hacerlo.

El objetivo de estas simples pruebas es estimar la tasa de refresco que requieren los comportamientos reactivos, de cara a volverlos más complejos y, además fijar una frecuencia de funcionamiento de la capa deliberativa.

Estas mismas pruebas mejoran notablemente al dejar que la posición de la pelota se obtenga desde las fuentes de información de la capa reactiva.

Estas pruebas se realizaron, tanto con el robot Nao, como el robot Hajime. Para adaptar estas estructuras al robot Hajime, se requirió realizar cambios tanto en los umbrales como modificaciones a nivel de manejo de información, ya que, por ejemplo, los valores de odometría no son compatibles y se obtienen con tasas y valores diferentes. Otro punto a recalcar es el cambio en el manejo de comando y, por lo tanto, del funcionamiento de la capa reactiva, puesto que en este robot no se puede tener comandos enviados de la misma manera. Con las diferencias mencionadas, el robot logró ejecutar sin inconvenientes la misión encomendada.

4.2 Mejora en Comportamientos del Robot

Lo que se muestra a continuación es el desarrollo de una mejora en los comportamientos del robot para la misión “atacar” con el rol de atacante. Lo que se requería era que el robot le pegara a la pelota alineado con el arco. Para esto, se

modificaron los comportamientos preparados anteriormente en la capa deliberativa (ver figura 24).

Se determinó que se requerían visiones de buena calidad respecto al arco y a la pelota, además de definir puntos intermedios de aproximación, para lograr la misión. Por otro lado, se detectó que la manera de asegurar la aproximación del robot de manera adecuada a la pelota era realizando la última parte de la tarea de la forma más recta posible. Esto último se decidió de este modo, ya que la caminata del robot resultaba más precisa cuando el robot caminaba en una trayectoria recta y, además, presentaba mayor estabilidad.

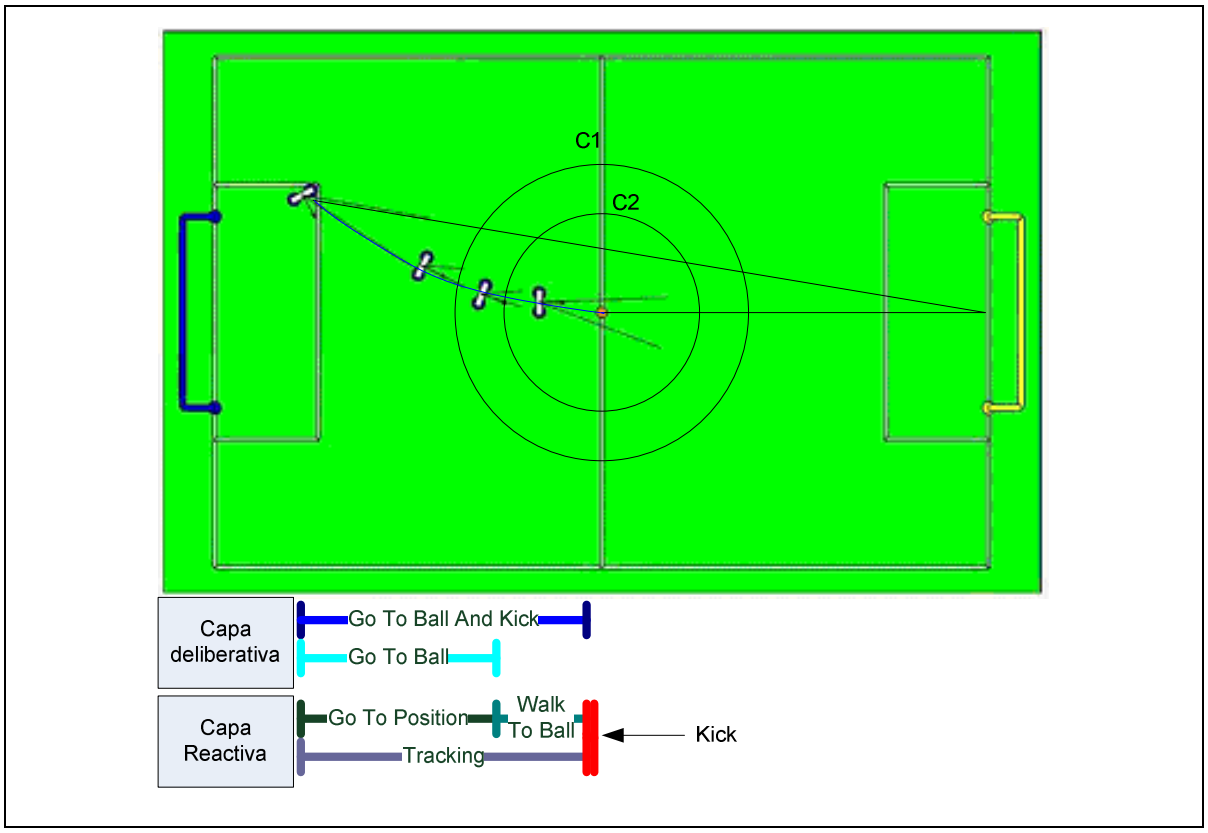


Figura 24: Diagrama con activación de comportamientos a medida que el robot se aproxima a la pelota en la capa deliberativa y en la reactiva. El círculo C1 es un umbral de Go To Ball en el que corrige solo un poco la alineación con el arco. C2, en cambio, es un umbral de Go To Ball And Kick, en donde le entrega el control de la llegada a la capa reactiva y la alineación con respecto al arco ya no es importante, sino que la llegada a la pelota.

4.2.1 Pruebas en Simulador

El desarrollo de los comportamientos se hizo mediante el uso del simulador de alto nivel. Se definieron 2 umbrales de acercamiento a la pelota como se observa en la figura 24, de manera que se pudiese manejar la curva que tomaría la trayectoria del robot. A este comportamiento se le agregó, además, una máquina de estados, descrita en la figura 25. Ésta fue creada para lograr que el robot detectase cuándo debía buscar el arco o la pelota, para así, lograr alinearse adecuadamente. Los cambios de estados están asociados a la confidencia de los objetos de interés de cada estado.

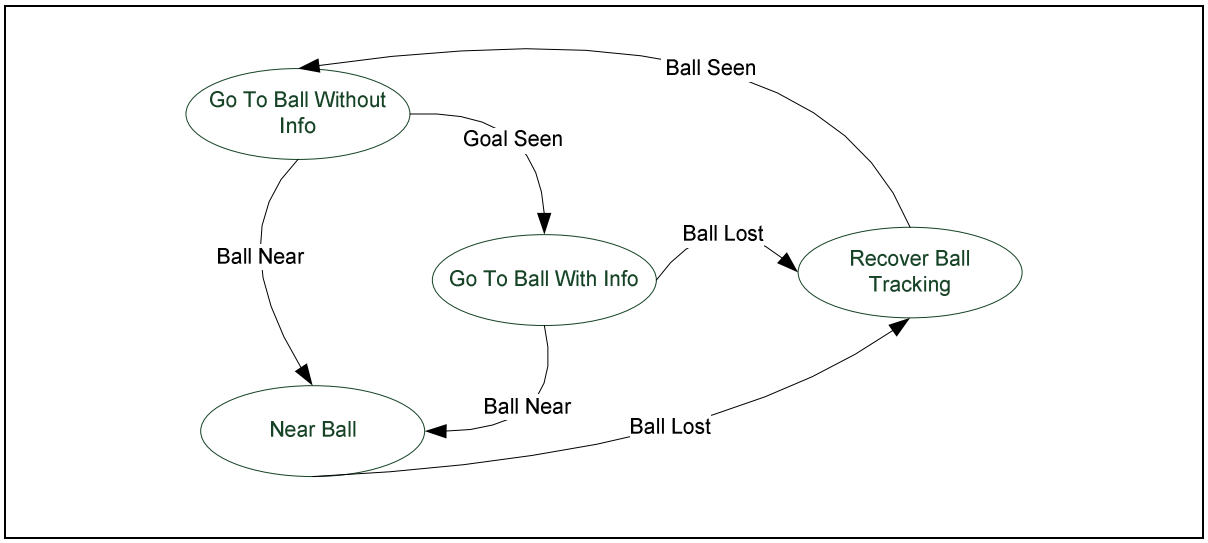


Figura 25: Máquina de estado generada para el comportamiento de acercamiento a la pelota. Salvo en el estado Near Ball en el cual deja el acercamiento fino de la pelota en manos de la capa reactiva, en los restos de los estados la trayectoria es manejada por la capa deliberativa a través de Go To Ball. El manejo de visión se realiza dependiendo del estado y escoge entre la pelota y los arcos, dependiendo de la orientación que tiene el robot.

La secuencia de comportamientos activados, cuando el jugador va a la pelota, es la siguiente:

- High Level selecciona Field Player

- Field Player tiene una máquina de estado que para la misión de atacante puede seleccionar entre Go To Ball and Kick si ha detectado la pelota y Look for object con *VA* la pelota.
- Go To Ball and Kick, dependiendo del estado en que se encuentra, selecciona los comportamientos que van a seguir funcionando. Go To Ball define la trayectoria del robot y envía un *VA* a la capa reactiva con la posición a la cual moverse.

Este comportamiento tiene una variante que toma en cuenta la localización del robot en la cancha, cuando se encuentra bajo los umbrales de incertidumbre. Consiste en que el robot use los límites de la cancha en las trayectorias de aproximación a la pelota, para no salirse de los bordes.

Otro punto que estuvo en discusión para la generación de este comportamiento fue revisar si era más efectivo realizar la aproximación a ésta de manera omnidireccional (usando trayectorias curvas), o bien, realizando la secuencia giro sobre sí mismo, caminata en línea recta y luego, ajuste girando. Esto, ya que generalmente los robots logran mayor velocidad de caminata cuando van en línea recta que cuando caminan con una trayectoria curva. Finalmente, se realizó de manera omnidireccional en el robot Nao, porque la estabilidad del robot cuando giraba en torno a sí mismo, era inferior a la que tenía cuando caminaba omnidireccionalmente. Para el robot Hajime, en cambio, se usó el otro enfoque, ya que éste presentaba problemas manejando la omnidireccionalidad.

4.2.2 Pruebas en el Robot

Al igual que en el comportamiento anterior, en el robot hubo que modificar los umbrales respecto a los generados en el simulador. Otro resultado que se obtuvo fue que el movimiento cabeza afectaba a la caminata con movimientos continuos. En este caso, el *switcheo* constante entre objetos a observar provocó, en reiteradas ocasiones, desestabilizaciones del robot. Éstas ocurrieron con la capa deliberativa funcionando a 10 [Hz], lo cual es una frecuencia que -como se había visto en los experimentos anteriores- es sobre las que necesita el sistema para funcionar. Limitando la capa deliberativa a 5 [Hz] las desestabilizaciones, producto del

movimiento de la cabeza se reducen, pero aún siguen estando presentes. Por último, con la capa deliberativa funcionando a 2 [Hz], éstas desaparecen casi por completo, únicamente complicándose cuando los cambios de posición de la cabeza requieren movimientos largos, como, por ejemplo, cuando se está cercano a la pelota y se quiere observar el arco, lo que provoca que el robot levante bruscamente la cabeza. Esto no se había detectado antes, puesto que nunca se había experimentado. Cabe recordar que, a esta altura, aún no se agrega visión activa a la arquitectura. La solución a esto fue una modificación a las transiciones de estado, no permitiendo al robot observar al arco cuando éste se encuentra pasado un umbral de distancia. El cambio de frecuencia, en este caso, no afectó en nada al cuerpo, ya que la posición de la pelota para el acercamiento final siempre la obtuvo de la información sensorial disponible en la capa reactiva.

Respecto a los resultados obtenidos por este comportamiento en el robot, resultan ser satisfactorios para el equipo, porque el robot logra hacer más goles. Además, y como resultado adicional, se obtiene un mejor acercamiento a la pelota, con lo cual aumenta la cantidad de golpes correctos que realiza el robot. La regularidad con que se logró que el robot hiciera goles no se había logrado antes y fue fruto de mejoras en la percepción de los objetos, así como del funcionamiento de la estrategia. Cabe recalcar que la mejora en la percepción de los objeto, es decir mejoramientos en los algoritmos de visión, no fue parte del trabajo de esta tesis pero si lo fueron las mejoras en el manejo de esta información en estrategia. En este caso, el robot logró en promedio una efectividad de 70% en el área marcada en la figura 26. Se eligió esta área, ya que fuera de ella el robot no tenía ángulo suficiente para funcionar. En el caso de las áreas más lejanas, los goles se hicieron usando varios golpes consecutivos.

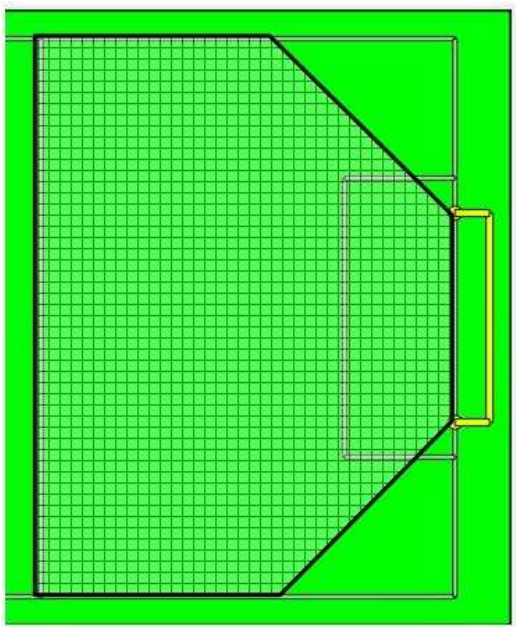


Figura 26: Área en que se realizaron las pruebas para esta misión.

4.3 Adaptación de Visión Activa

Una vez realizadas las adaptaciones de los comportamientos de la misión atacante, se procedió a efectuar la conversión de los comportamientos de la misión arquero. Al cumplir esto, fue necesario adaptar visión activa y, por lo tanto, agregar el manejo de múltiples objetivos a la arquitectura. Como se dijo con anterioridad, para esto se creó el concepto del multiplexor.

En la rutina de “arquero intenta tapar el arco”, se minimiza el ángulo de tiro que puede tener el atacante. En la Figura 27 se observan los parámetros de la rutina, en la cual el arquero debe tratar de minimizar β para lograr cubrir correctamente el arco. Por lo tanto, para realizar esta tarea el arquero necesita estar localizado para asegurarse de que está bloqueando el arco, al mismo tiempo que tiene que mirar si la pelota viene hacia el arco. Para realizar esta tarea fue que se implementó visión activa. Mientras este comportamiento se ejecuta, visión activa es usada intensivamente. Sólo se deja de usar cuando la pelota se aproxima al arco o la localización está muy mala, por lo tanto, el

algoritmo es usado la mayor parte del tiempo que el arquero se encuentra en funcionamiento.

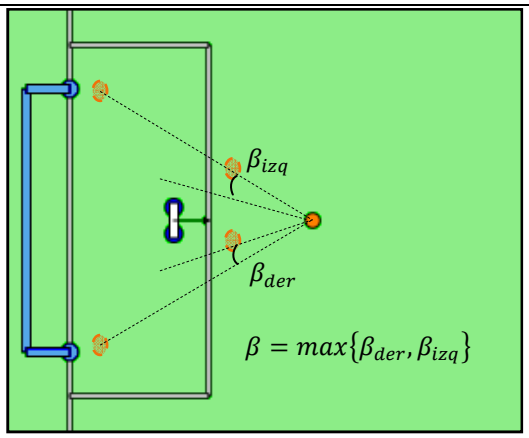
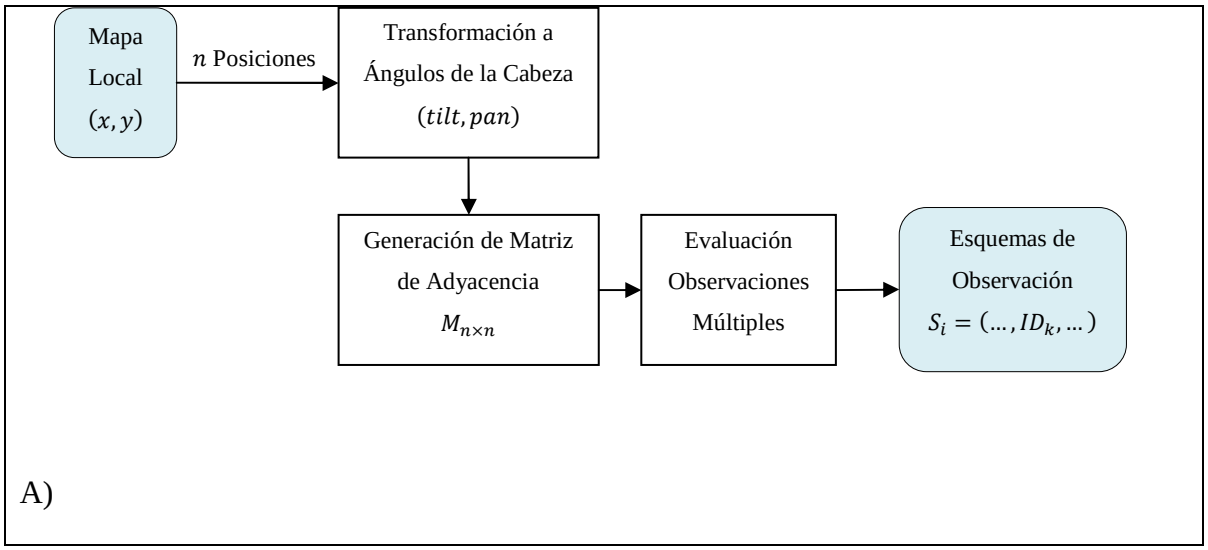


Figura 27: Rutina de cubrir el arco desarrollada por el equipo.

El trabajo de adaptación de la misión del arquero consistió -principalmente- en la creación y luego, ajuste del multiplexor. En la mayoría de los comportamientos que componen la misión arquero, los cambios fueron similares a los que se hicieron en la misión anterior; es decir, condiciones de los estados. El cambio mayor fue modificar el comportamiento de visión activa para que pudiera ser usado genéricamente por distintas funciones de valor y, además, que pudiese ser evaluado bajo distintos conjuntos de parámetros. Un diagrama del algoritmo usado por el comportamiento de visión activa está en la figura 28, la cual consta de dos partes, la primera explica la generación de los esquemas posibles observar desde los objetos de interés, y, una segunda en que explica el algoritmo usado para la evaluación de los esquemas a observar. El primer cambio en el comportamiento significó dejar éste habilitado para que pudiese usar diferentes clases que hicieran la evaluación de las simulaciones con Kalman que ahí se realizan. Estas clases heredan de una genérica y son las encargadas de evaluar numéricamente los estados simulados que se suscitan. Estás últimas deben ser programadas en función de la misión y rol que se quiere realizar. El segundo cambio es agregar parámetros al funcionamiento del algoritmo. Los argumentos que se requieren son cantidad de grupos y número máximo de elementos por grupo. Estos dos factores afectan directamente el

rendimiento computacional de las simulaciones de estados, porque el algoritmo usa muestreo determinístico. Por tanto, para efectos de la estrategia, manejar estos dos aspectos resulta clave al momento de no realizar simulaciones muy prolongadas.

Todo el algoritmo de simulación del equipo [11], incluyendo la función de valor, se determinó que estuviese en la capa deliberativa. Al realizar esto, el esquema de creación de grupos de objetos a observar (Figura 28 A) es generado transformando el mapa local de objetos en el mapa de *encoders* de *tilt* y *pan* del robot. Esta transformación permite verificar mejor en ese espacio qué objetos pueden ser vistos juntos, tomando en cuenta la apertura focal y horizontal de la cámara del robot. Los grupos para escogidos para ser observados conjuntamente son a los que se les simula las observaciones que generarían en caso de ser vistos $h(x_k^-, 0)$, las cuales son usadas para corregir la etapa predictiva del filtro de kalman (x_k^-, P_k^-) que se observa en la figura 28 B. El algoritmo completo, al funcionar en la capa deliberativa, está sujeto a limitaciones de velocidad de funcionamiento y, en este caso, la capa deliberativa está desempeñándose a un máximo de 2 [Hz], pudiendo ser esta frecuencia menor, en caso que el algoritmo demore más en iterar. En el caso de las trayectorias en este comportamiento, se fijaron que se generarán en el nivel deliberativo mientras esté ejecutándose el algoritmo de visión activa.



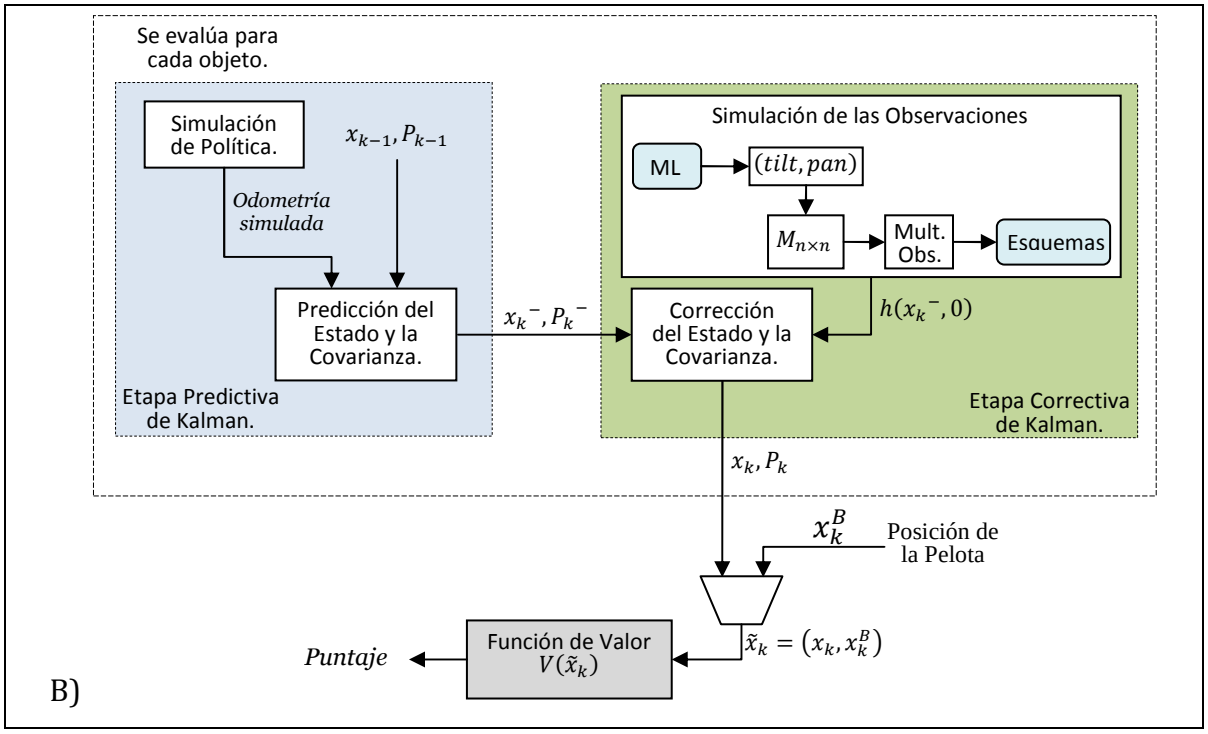


Figura 28: Esquema de funcionamiento del algoritmo de visión activa que pasa a ser ejecutado en la capa deliberativa. A) muestra la generación de esquemas de observación por parte del algoritmo, esta parte ejecuta un muestreo determinístico, mediante el cual determina qué objetivos pueden agruparse. B) esquema completo del algoritmo de visión activa que incluye la función de valor.

El multiplexor mejora la rapidez de reacción de la capa deliberativa, al permitir que el objetivo a observar se pueda escoger dinámicamente y no depender de la capa deliberativa para esto. Como se detectó en los experimentos anteriores, los movimientos de la cabeza del robot afectan el funcionamiento del robot, pero esto ocurre cuando el robot ejecuta largos desplazamientos, a diferencia de lo que sucede en este caso. De igual forma, a nivel de la capa reactiva, está disponible toda la información del robot, por lo que para evitar estos problemas, se decidió generar una máquina de estados, como la que se tiene en los comportamientos deliberativos, en el multiplexor. Las funciones de la máquina de estado del multiplexor de la cabeza son:

- Evitar inestabilidades en las caminatas del robot por movimientos de la cabeza.
- Forzar la ejecución de seguimiento del objeto, en al menos unos milisegundos antes de comenzar la búsqueda.

- Manejar las oscilaciones de la cabeza para el caso de las observaciones de múltiples objetos.

Esta máquina de estado se encuentra en la figura 29. Para evitar las inestabilidades durante la caminata, se tiene control respecto de la posición de la cabeza en todo instante y la posición a la cual se desea enviar. Luego de las pruebas realizadas con el comportamiento anterior, se detectó que los mayores problemas ocurrían después de cambios reiterados de la posición de la cabeza y cuando el robot se encontraba ejecutando pasos largos. Por lo que, para eliminar esas interferencias, se pasa a usar el estado de la caminata que maneja el nivel reactivo además del comando y tiempo en que se envió, con el fin de limitar los movimientos máximos permitido a la cabeza. Con la información de la posición de la cabeza, se puede tomar la determinación de realizar *tracking* con una de las dos cámaras, con el fin de disminuir los movimientos de la cabeza, siempre y cuando el robot esté ejecutando una caminata con pasos largos, información que está disponible en este nivel. La cabeza puede enviar comandos rápidos, por lo que los ciclos son de aproximadamente 30 [Hz] provocando que la determinación de un ciclo no se afecte en el resultado global.

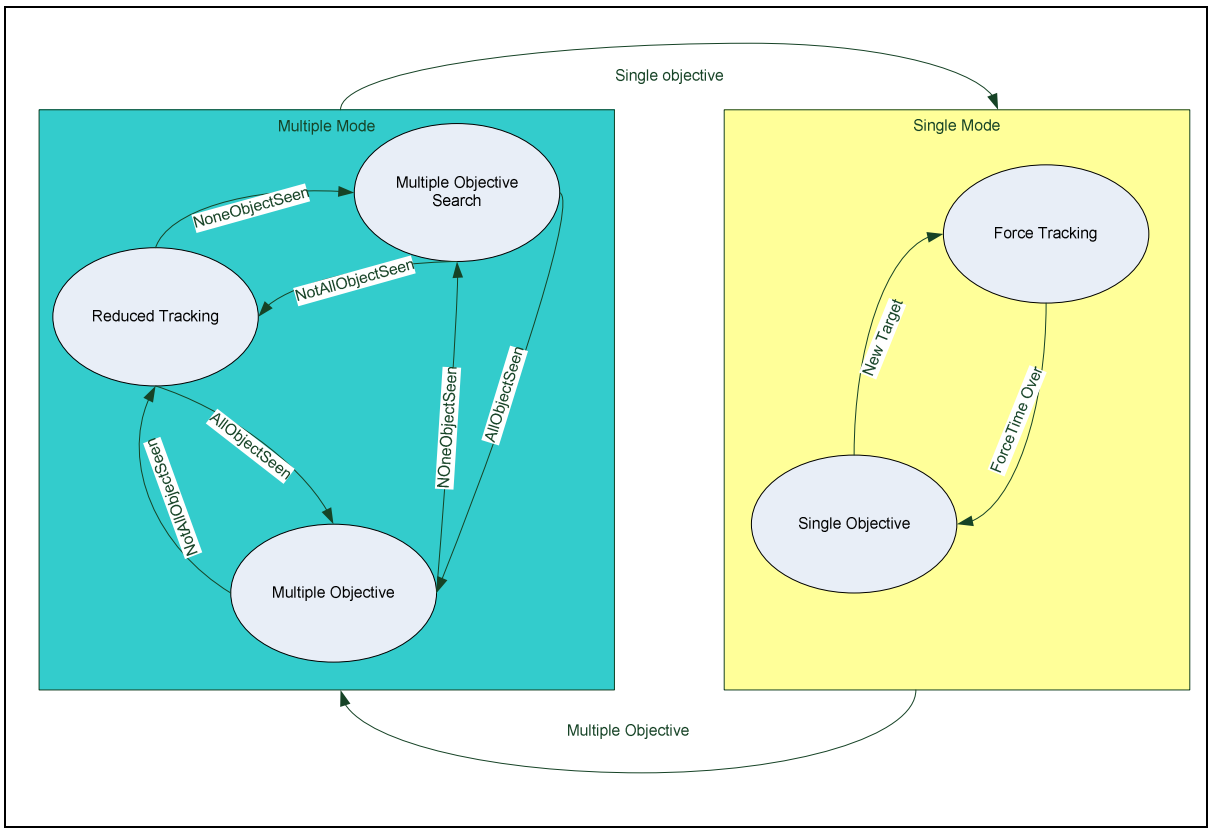


Figura 29: Máquina de estados del multiplexor de la cabeza. Esta máquina de estado cuenta con la particularidad que tiene dos modos de funcionamiento: uno para objetivos simples y el otro para objetivos compuestos.

La máquina de estados del *mux*, cuando se tiene múltiples objetivos, permite controlar los movimientos de la cabeza y, además, lograr observar los objetos deseados. El estado Reduced Tracking permite al robot realizar seguimiento sólo a los objetos del robot que se encuentran en el rango y, con esto se reducen las oscilaciones de la cabeza, ya que le permite seguir ejecutando seguimiento, aún cuando no detectase todos los objetos. Multiple Objective Search permite priorizar la búsqueda de los objetos dentro del grupo, seleccionando primero la pelota y luego, el resto de los objetivos del grupo. El cambio entre la máquina de estado que maneja múltiples objetivos y la simple, depende de la selección de objetivo a mirar por puntaje. Cuando cambia de una a otra, evalúa el estado interno en que debiese encontrarse. En el caso de la simple, la salida del estado forzado depende de condiciones de tiempo y confidencia del objetivo.

4.3.1 Pruebas en Simulador

Al igual que en los comportamientos anteriores, las pruebas del simulador sólo son para corroborar que la arquitectura diseñada realiza las tareas que se desean y que se logre observar un comportamiento aceptable, dejando las sintonizaciones para cuando se hacen los experimentos en el robot.

En este caso, además, se usó el simulador, para efectuar pruebas que en la realidad resultan difíciles de realizar, tales como movimientos continuos con la pelota. Estas pruebas se hacen con el objetivo de verificar el correcto desempeño del algoritmo, realizando seguimiento a la pelota en movimiento y no en una posición fija. Esto último se puede realizar de manera real, pero resulta costoso y sólo se pueden hacer a pequeña escala, resultando complejo de evaluar cualitativamente. En este caso, se proponen tres trayectorias diferentes de la pelota a 0,2 metros por segundo como se observa en la figura 30:

- Paralela a la línea del arco.
- Perpendicular a la línea del arco.
- Elíptica en torno al área del arco.

Se escogió 0,2 metros por segundo, ya que se estimó como velocidad similar a la de un golpe por parte de un jugador.

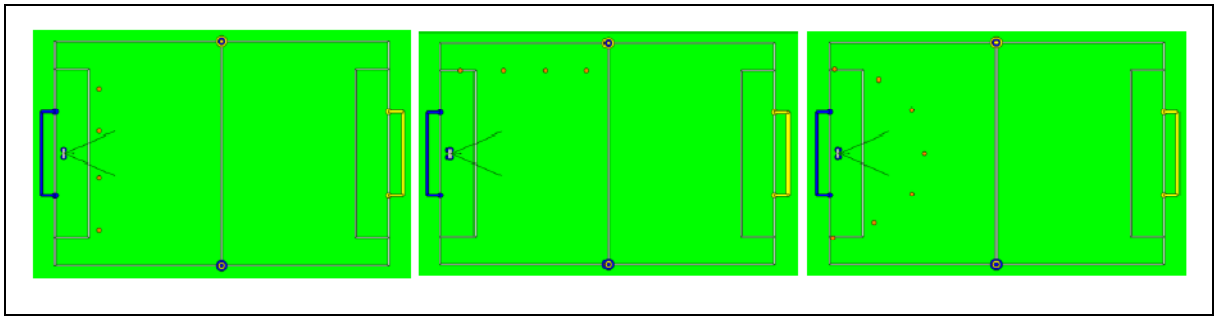


Figura 30: Trayectorias usadas para evaluar visión activa en el robot. En cada uno de estos experimentos la pelota tuvo un movimiento de 0.2 metros por segundo.

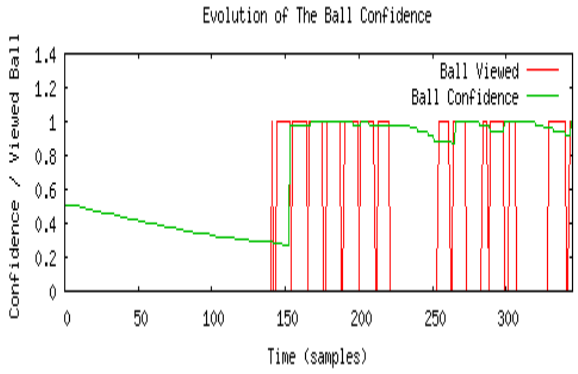
Además, para cada una de estas trayectorias se observaron los comportamientos de la capa deliberativa funcionando a 0,5, 1 y 2 [Hz]. En estos experimentos la

frecuencia de la capa reactiva varió entre 14 y 100 [Hz], con una media de 57 [Hz]. La frecuencia de la capa reactiva funciona al máximo posible, por lo que las variaciones son consecuencia de cambios en las condiciones de ejecución, tales como procesamiento de imágenes, variaciones en los comportamientos activados o evaluados, o cualquier otro proceso que ocurra dentro del robot. Los resultados de estos experimentos se muestran a continuación en la figura 31. La figura está compuesta por tres casos diferentes, cada uno con la capa deliberativa a diferente frecuencia: A) 0,5 [Hz], B) 1 [Hz], C) 2 [Hz]. Por cada una de las frecuencias monitoreadas, se generan cuatro figuras (1, 2, 3, 4), las cuales son:

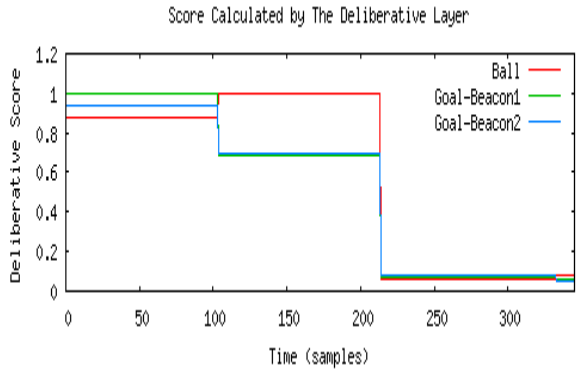
1. Muestra la evolución de las confidencias de la pelota y las observaciones puntuales a ésta. Las intermitencias en las observaciones de la pelota están dadas por la dinámica del sistema.
2. Muestra el puntaje deliberativo calculado por la capa deliberativa para cada grupo de objetos a observar. Los cambios corresponden a cada evaluación nueva de la capa deliberativa.
3. Muestra el puntaje reactivo calculado en el multiplexor en la capa reactiva de los grupos seleccionados en la capa deliberativa.
4. Muestra el puntaje final de cada grupo con el cual el multiplexor escoge finalmente que objetivo utilizar

En cada caso los puntajes finales mostrados corresponden al de los objetos de interés que son la pelota, el arco poste1 y poste2, agrupados según el sistema estime conveniente.

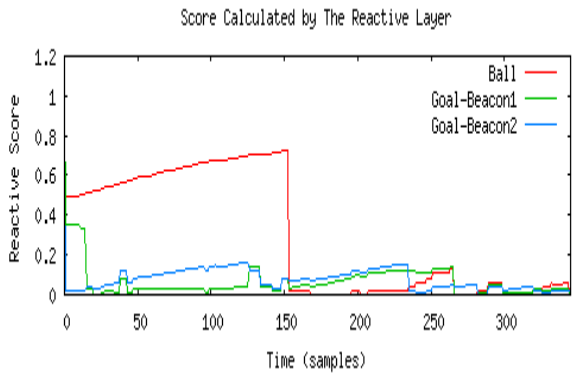
Durante todos estos experimentos el promedio de la capa deliberativa fue de 57 [Hz]. Luego en la tabla 3, se presenta el porcentaje promedio de tiempo que el robot decide observar la pelota y el promedio de tiempo en los cuales la posición de la pelota es desconocida en función de la frecuencia de la capa deliberativa.



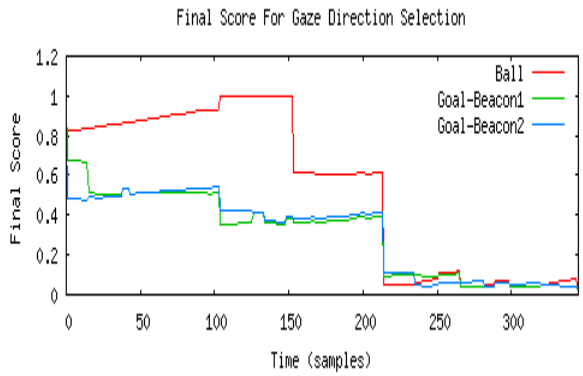
1



2

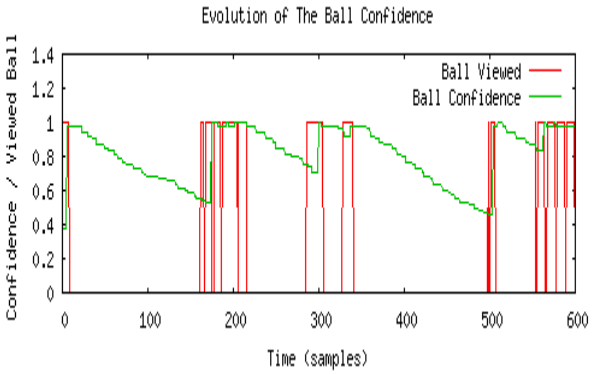


3

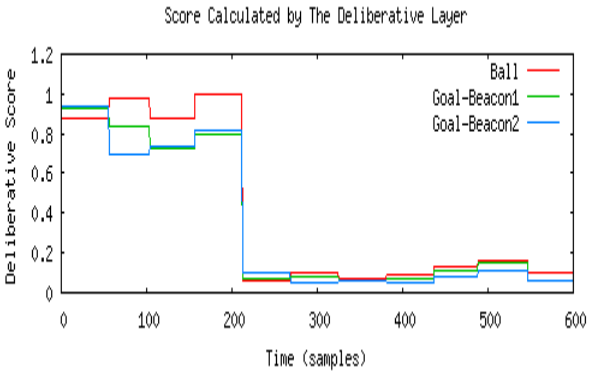


4

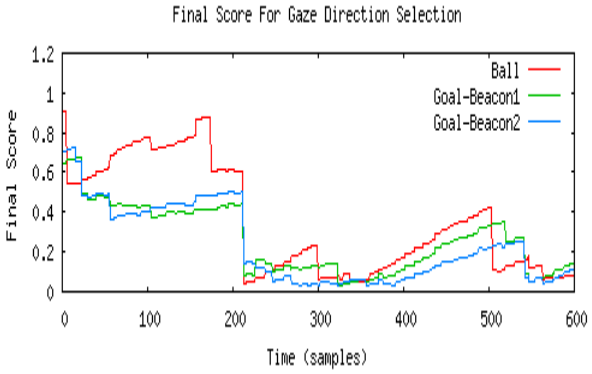
Figura 31 A) Funcionamiento de la capa deliberativa a 0,5 [Hz] con la capa reactiva funcionando a 57 [Hz] en promedio.



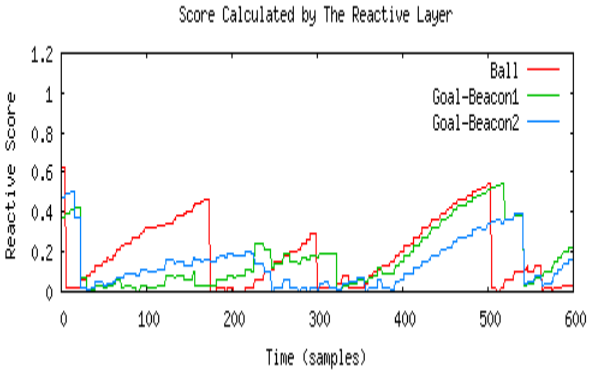
1



2

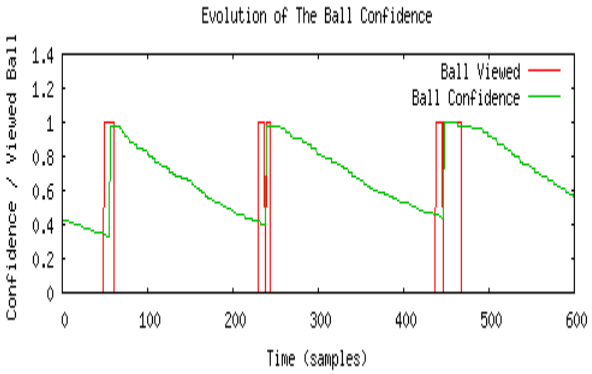


3

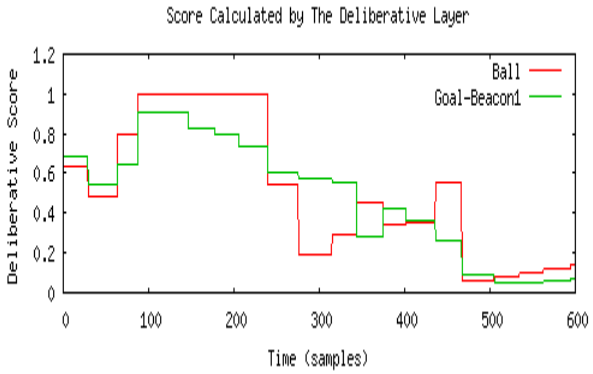


4

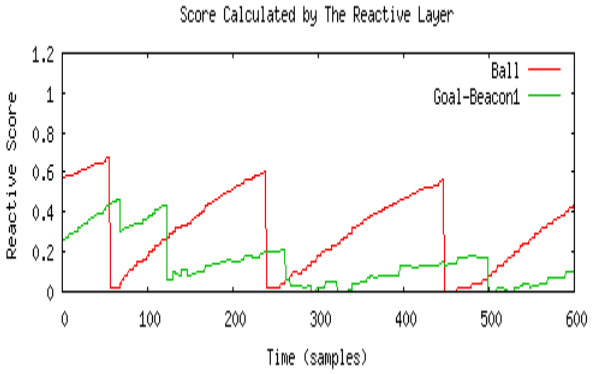
Figura 31 B) Funcionamiento de la capa deliberativa a 1 [Hz] con la capa reactiva funcionando a 57 [Hz] en promedio.



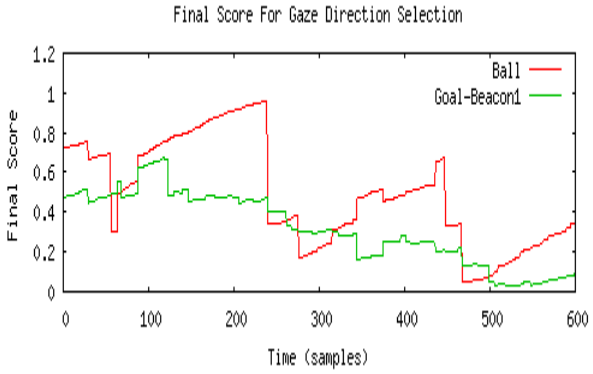
1



2



3



4

Figura 31 C) Funcionamiento de la capa deliberativa a 2 [Hz] con la capa reactiva funcionando a 57 [Hz] en promedio.

Tabla 3: Estadística del sistema funcionando con el comportamiento de arquero. En todos los casos la capa reactiva funcionó a 57 [Hz] en promedio.

Frecuencia de la capa deliberativa en [Hz]	Porcentaje del tiempo en que el robot decide observar la pelota	Porcentaje del tiempo con la pelota en una posición desconocida
0.5	45.2 %	34.2 %
1	28.5 %	19.0 %
2	31.5 %	10.9 %

En las curvas de la figura 31 A2 se observa que durante el periodo evaluado hubo 3 iteraciones de la capa deliberativa las cuales corresponden a 4 segundos de evaluación. Durante la primera iteración el grupo con el mayor puntaje corresponde al del arco y poste1, luego en la segunda es la pelota y finalmente en la tercera nuevamente el grupo inicial. En la capa reactiva, figura 31 A3, en cambio se observa un comportamiento diferente del puntaje calculado para los grupos. En efecto en esta se observa que el puntaje de la pelota es el mayor y sigue una tendencia al alza. Esta tendencia al alza es coincidente con lo que ocurre en la figura 31 A1 en donde se observa que la pelota no ha sido detectada. Finalmente en la figura 31 A4 se observa que el puntaje final usado para seleccionar el objetivo a seleccionar, el cual para este caso corresponde a la pelota. Esta elección es usada hasta el punto 220 que corresponde a cuando la capa deliberativa nuevamente cambia los puntajes calculados y se observan cambios en los objetivos a seleccionar.

El análisis efectuado corresponde al que se debe efectuar con los puntajes calculados por el sistema. Este mismo análisis en las figuras 31B y 31C se vuelve más largo debido a la mayor cantidad de cálculos efectuados por la capa deliberativa. Las diferencias en esas figuras pasan por un mejor manejo de las observaciones a la pelota lo cual se puede observar en las figuras 31 B1 y 31 C1 en

donde las visiones a la pelota se realizan de manera periódica permitiendo mantener los niveles de la confidencia en los rangos deseados.

Dos grandes conclusiones se pueden observar en los experimentos. La primera es que el cálculo del puntaje final, usando el puntaje reactivo y deliberativo en la capa reactiva, es lo que permite al sistema reaccionar a los cambios del ambiente. El puntaje reactivo hace posible que el sistema reaccione rápidamente a cambios, manejándolos ágil y eficientemente como se observa en la figura 31A; mientras, el puntaje deliberativo maneja objetivos en un horizonte de tiempo más largo, los cuales son necesarios para cumplir con la misión. Por ejemplo, para el caso en que la capa deliberativa funciona a 1 [Hz] (figura 31B), la pelota avanza 20 centímetros entre cada iteración de la capa. Tomar decisiones con la capa funcionando a esta velocidad, no permitiría realizar seguimiento a la pelota adecuadamente. Aún así, como se observa en la figura 31B, el sistema logra seguir la pelota, y su confidencia se mantiene alta durante todo el periodo. La segunda conclusión que se puede obtener, es que el sistema requiere una frecuencia mínima a la cual el puntaje deliberativo tiene que ser actualizado. Esto se puede observar en la figura 31A, en donde la capa deliberativa funciona a 0,5 [Hz]. En este caso, la pelota se mueve 40 centímetros entre cada iteración de la capa deliberativa, y realiza seguimiento sólo con la capa reactiva a 57 [Hz]. Esto se ratifica con el resultado de la tabla 3, en donde se observa que en el 34,5% de los casos, la posición de la pelota es desconocida. En efecto, las confidencias de la pelota se mantienen bajas, aun cuando el robot está intentando mirar la pelota (45,2% del tiempo). En este experimento el robot tiene problemas de localización, ya que intenta observar la pelota gran parte del tiempo, no permitiendo que observe el arco y postes para mejorar su ubicación. Los problemas de la localización se detectan porque cuando el robot está en esta situación deja de utilizar el algoritmo de visión activa ya que su prioridad pasa a ser la localización. Con la capa deliberativa funcionando a una velocidad superior, el robot realiza los cambios de mirada a los diferentes objetos, de forma más precisa que cuando ésta funciona a una tasa más lenta, debido a que en el caso que haya grandes cambios en el ambiente, la selección de los objetivos deliberativos se vuelve trascendental. Como se observa en la tabla 3, cuando la

velocidad de la capa deliberativa se incrementa, ambos porcentajes de tiempo, en que el robot decide observar la pelota y el que la pelota está en una posición desconocida, disminuyen.

4.3.2 Pruebas en el Robot

Este comportamiento se testeó funcionando en el robot. Esta vez, la pelota no tuvo una trayectoria constante como en el simulador, en cambio, se hicieron tiros al arco para comprobar la reactividad del sistema. En estas pruebas se comprobó que el sistema logró realizar el seguimiento de la pelota el 65% de las veces. Este valor se calculó tomando el promedio de diez repeticiones de la prueba, en cada repetición se lanzaban diez tiros al arco. En estas pruebas, se observó que las veces en que el sistema perdió la pelota, se debió también a que coincidió con que la localización del robot no se hallaba funcionando de manera adecuada, esto dado que se encontraba mal posicionado pensando que se encontraba en otro sector del área. Por otro lado, se observó también que el robot no tuvo los problemas que había presentado anteriormente cuando se le hizo cambiar de posición a mirar de manera constante, debido a que esta vez los movimientos de cabezas fueron suavizados por el multiplexor.

En esta prueba no se midió el rendimiento de los objetivos (goles marcados en el caso del atacante y goles atajados en el caso del arquero) de los comportamientos, ya que en el caso del arquero se requiere que la localización converja para que logre tener un funcionamiento adecuado, y esto no es necesario para el atacante. En la Tabla 4 se muestra el promedio de la frecuencia de las capas reactiva y deliberativa, ejecutando diferentes misiones y roles. Estos valores se midieron en numerosas situaciones de juego. Como se puede observar, el promedio de la capa deliberativa es de ~1.65 [Hz], mientras que el de la reactiva es ~56 [Hz]. Las oscilaciones, más notorias en la capa reactiva que en la deliberativa, son fruto de los asincronismos del sistema y que en la capa reactiva tienen mayor incidencia debido a la frecuencia de funcionamiento de los sensores. En efecto si después de evaluar los comportamientos por información visual nueva se gatillan los acelerómetros, resultan dos evaluaciones seguidas de la capa reactiva y por lo tanto la frecuencia de

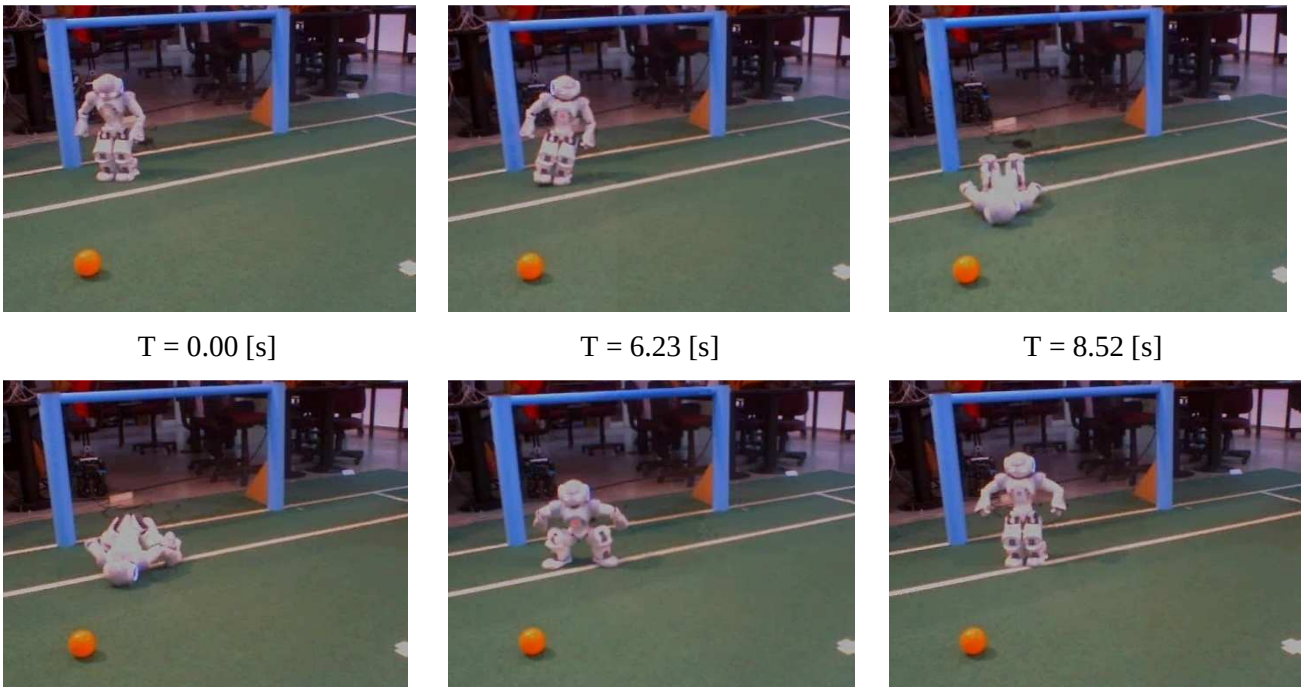
ejecución aumenta. Es por esto que en el caos de la capa reactiva el promedio es más significativo que los extremos.

Además, en estas pruebas se comprobó el funcionamiento de la capa de *planning*, realizando cambios entre las misiones de atacante y arquero, para verificar que el robot lograba estos cambios sin afectar su reactividad. Ello se refleja en que durante estos cambios la frecuencia promedio de funcionamiento de la capa reactiva no varió su frecuencia de funcionamiento entre misiones como se puede observar en la tabla 4. Es importante recalcar que la diferencia en la frecuencia promedio de trabajo de la capa reactiva entre misiones está dada por el uso de visión activa y no así por el cambio de misión en el robot. Esto último era de esperar, ya que por diseño de la arquitectura, el funcionamiento de las capas superiores no repercute en la capa reactiva, siendo una de las ventajas por las cuales se realizó la arquitectura híbrida. Los cambios de misiones fueron comunicados por red inalámbrica.

Por otro lado, se midió el tiempo en que el robot detecta la caída, dejándolo caer mientras ejecuta el comportamiento de visión activa. Esta prueba resulta clave para la arquitectura, ya que mientras se usa el comportamiento de arquero realizando visión activa, la carga del procesador es la máxima que puede tener el robot. Si éste detecta la caída de manera oportuna, implica que el sistema está manejando de forma apropiada las cargas computacionales sin mermar su reactividad. Después de caer, el robot detectó la caída en sólo 51 milisegundos, y después de algunos segundos, ya estaba en pie nuevamente (ver Figura 32). La secuencia de parada toma varios segundos, debido a restricciones de mecánicas y eléctricas del robot. Aún cuando la capa deliberativa se ejecuta cada 600 milisegundos, el robot detecta la caída en 51 milisegundos, porque la capa reactiva se ejecuta cada 18 milisegundos. La caída se detecta apenas la capa reactiva recibe y procesa las lecturas de los giróscopos y acelerómetros.

Tabla 4. Frecuencia de la capa reactiva y deliberativa para diferentes misiones y roles en experimentos reales con un robot Nao.

Misión/Rol	**Capa**	**Frecuencia Max Hz**	**Frecuencia promedio Hz**	**Frecuencia Min Hz**
Defender el arco/Arquero	Reactiva	69	54	42
	Deliberativa	1.91	1.6	1.5
Atacar/Delantero	Reactiva	70	58	47
	Deliberativa	1.9	1.71	1.6



T = 0.00 [s] T = 6.23 [s] T = 8.52 [s]

T = 8.57 [s] T = 14.09 [s] T = 17.16 [s]

Figura 32: Ejemplo de situación de una caída y parada del robot, mientras ejecutó el comportamiento de tapar el arco. En T=8.52 el robot cae. En T=8.57 el robot detecta la caída y comienza a levantarse. En T=14.09 el robot vuelve a una posición adecuada. La secuencia para pararse toma varios segundos debido a limitaciones mecánicas y eléctricas del robot.

4.4 Pruebas de Comportamientos Grupales

Las habilidades para jugar fútbol de un equipo de robots y ejecutar comportamientos grupales mientras se usa la arquitectura diseñada fueron validados, usando el simulador

y a la vez en robots reales. Lo que se realizó fue en la misión de atacante, crear el rol de defensor. En efecto, éste corresponde al robot que no va tras la pelota, sino que debe irse a una posición en la cancha en la que se evita que estorbe al compañero con rol de atacante y, además, pueda ayudar a la defensa del equipo. De esta manera para la misión de atacante hay dos roles, atacante y defensa, posibles de generar para el robot. En ningún caso se puede tener que los dos robots tengan el rol de defensa, pero si el de atacante (caso en que no hay comunicación entre los robots).

Durante los experimentos, los robots no tuvieron nunca problemas de reactividad, aún cuando se simularon fallas de la red. En los tiempos en que hubo fallas en la red, se observó que el robot que tenía el rol de defensor, al cabo de unos segundos, cambió al rol de atacante, ya que se encaminaba directo a la pelota. En ningún momento el robot con rol de atacante se detuvo, lo cual se debe a que no se vio afectado por los problemas de la red. Esto último, era uno de los objetivos de la arquitectura debido a que, por saturación de la red, generalmente los robots tienen problemas durante las competencias.

Además, en la *RoboCup* 2010 los robots jugaron en conjunto con el equipo SPQR de la Universidad de Roma, bajo el nombre de CHITA. En este caso, la arquitectura permitió una fácil adaptación de las comunicaciones con el equipo italiano, ya que solo estaban en la capa de equipo. Esto permitió que los comportamientos del robot no fueran modificados y, por lo tanto, no perdieran los ajustes previos que se realizaron para la competencia.

Durante esta competencia los resultados obtenidos por el equipo le significaron quedar entre los 12 mejores de 23 participantes en la competencia. Se considera que la participación fue exitosa.

V. CONCLUSIONES

Este trabajo de tesis creó una arquitectura híbrida de control de robots. Este modelo fue primero probado en el simulador HLSIM y luego en los robots Nao y Hajime de diferentes categoría de fútbol robótico. Esta arquitectura permitió jerarquizar la construcción de los comportamientos, posibilitando un mejor manejo de la información sensorial disponible, haciendo uso de mediciones de confiabilidad de las percepciones. Los resultados experimentales obtenidos en la aplicación de la estructura híbrida muestran que, efectivamente, el modelo logra darle orden a la estructura de comportamientos, ya que éstos son ejecutados según la necesidad que tenga el robot. Este hecho permite ahorrar procesamiento y simplifica la creación de comportamientos.

Respecto a la arquitectura propuesta en tres niveles, se puede concluir que, en estricto rigor, los beneficios de la arquitectura híbrida se podrían haber obtenido únicamente dividiendo en dos capas el control. Sin embargo, el dividir en tres capas permite agregar flexibilidad a la programación de comportamientos grupales, como fue demostrado en la fácil integración con otro equipo, tan sólo con modificar la capa de planificación. Esto es un atributo que puede tornarse relevante en un futuro cuando se creen nuevos comportamientos grupales.

La conclusión que se puede obtener sobre las arquitecturas híbridas es que, en efecto, logran ordenar el sistema de control del robot y, con esto, otorgar una mejor sincronización con su hardware.

En este trabajo se constató que la clave para el buen funcionamiento del robot es la coordinación con el hardware, y sin esto, el robot difícilmente puede ejecutar tareas más complejas; en este sentido, la arquitectura híbrida facilita la priorización de los comportamientos, disminuyendo de esta forma, las descoordinaciones que pueden ocurrir. Esto se puede ratificar con la adaptación que se realizó de la arquitectura para el robot Nao y Hajime.

Por otro lado, en este trabajo se desarrolló el control activo de la cámara del robot. Ello se hizo creando la pareja de función de valor – multiplexor, la que permite usar las ventajas de la arquitectura híbrida para mejorar el flujo de información y, con esto obtener un

mejoramiento informativo en el robot. La nueva arquitectura permitió enriquecer el trabajo realizado previamente por el equipo, ya que ahora el algoritmo no afecta la reactividad del robot, gracias a la arquitectura híbrida. En efecto, el uso de visión activa se simplificó, ya que ahora tan sólo depende de la creación de una función de valor que permita la evaluación de los distintos objetivos. En ese sentido, este trabajo de tesis es sólo el inicio del uso de esta modalidad de visión activa y se esperan futuros trabajos al respecto, mediante la introducción de algoritmos de aprendizaje para la generación de funciones de valor. En el caso del cuerpo y de todo el robot, las funciones de valor pueden ser construidas, pero hay otros inconvenientes que resolver aún, acerca de la sintonización de las caminatas, las cuales siguen siendo inestables.

Por último, cabe recalcar que todo el trabajo quedó integrado con la librería UChile-Lib, lo cual posibilita un continuo uso por el equipo y que éste no se pierda. Esto resulta clave, ya que, como se dijo anteriormente, este trabajo constituye únicamente el inicio del uso por parte del equipo de la arquitectura y, con el tiempo, se espera que los comportamientos se vuelvan más complejos y ejecuten misiones más difíciles.

VI. BIBLIOGRAFÍA

[1] James S. Albus, Outline for a Theory of Intelligence, en IEEE Trans. on Systems, Man, and Cybernetics, Vol. 21, No. 3, may/june 1991.

[2] R. Arkin, *Behavior-Based Robotics*, MIT Press 1998.

[3] R. Arkin, M. Fujita, T. Takagi, R. Hasegawa, An ethological and emotional basis for human–robot interaction, *Robotics and Autonomous Systems*, Vol. 42 Issue 3-4, 2003.

[4] S. Behnke, B. Frötschl, R. Rojas, Using hierarchical dynamical systems to control reactive behavior, *Lecture Notes in Computer Science*, pp 186-195, 2000.

[5] S. Behnke, J. Stückler, H. Strasdat, M. Schreiber, Hierarchical Reactive Control for a Team of Humanoid Soccer Robots, 2008.

[6] S. Behnke, J. Stückler, H. Strasdat and M. Schreiber, Hierarchical Reactive Control for Soccer Playing Humanoid Robots, *International journal of Humanoid Robotic*, Vol. 5 No 3, september 2008

[7] R. Berger, H. Burkhard, AT Humboldt – Team Description 2007, *Lecture Notes in Computer Science* (RoboCup 2007), 2008.

[8] M. Berger, H. Endert, S. Joecks, Combining Learning, Deliberation and Reactive Control in Simulated Soccer: DAInamite Framework, *Proc. Workshop On Hybrid Control of Autonomous Systems*, pp. 57-62, 2009.

[9] E. Bicho, G. Schöner, The dynamic approach to autonomous robotics demonstrated on a low-level vehicle platform, 1998.

[10] R.P. Bonasso, D. Kortenkamp, D.P. Miller, and M. Slack, Experiences with an architecture for intelligent, reactive agents, *J. Exp. Theor. Artif. Intell.* 237-256, 1997.

[11] R. Brooks, A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, Vol. 2 No 1, 1986.

[12] S. Chernova, R. Arkin, From Deliberative to Routine Behaviors: A Cognitively Inspired Action-Selection Mechanism for Routine Behavior Capture, *Adaptive Behavior*, Vol. 15 No. 2, pp. 199-216, 2007.

[13] H. Christensen, P. Pirjanian, Theoretical Methods for Planning and Control in Mobile Robotics, *IEEE Conf. on Knowledge-Based Intelligent Electronic System*, 1997.

[14] J. Correa, A. Soto, Active visual perception for mobile robot localization, *Journal of intelligent and robotic system*, 2009.

[15] A. Ferrein, A. Potgieter, G. Steinbahuer, Self-Aware Robots- What do we need from Learning, Deliberative and Reactive Control?, *Proc. Workshop On Hybrid Control of Autonomous Systems*, pp. 1-5, 2009.

[16] M. Friedmann, J. Kiener, S. Petters, D. Thomas, O. von Stryk, H. Sakamoto, Versatile, High-Quality Motions and Behavior Control of Humanoid Soccer Robots, *Proc. IEEE Workshop on Humanoid Soccer Robot*, pp. 9-16, 2006.

[17] E. Gat, On Three Layer Architectures, *Artificial Intelligence and Mobile Robots*, pp. 195-210, AAIA press, 1997.

[18] P. Guerrero, and J. Ruiz-del-Solar, Improving robot self-localization using landmarks' poses tracking and odometry error compensation. *Lecture Notes in Computer Science* 5001 (*RoboCup 2007*) pp. 148-158, 2008.

[19] P. Guerrero, J. Ruiz-del-Solar, G. Díaz, Probabilistic Decision Making in Robot Soccer. *Lecture Notes in Computer Science* 5001 (RoboCup 2007) pp. 29-40, 2008.

[20] P. Guerrero, J. Ruiz-del-Solar, M. Romero, Explicitly Task Oriented Probabilistic Active Vision for a Mobile Robot. *Lecture Notes in Computer Science* 5399 (RoboCup Symposium 2008) pp. 85-96.

[21] P. Guerrero, J. Ruiz-del-Solar, M. Romero, S. Angulo, Task Oriented Probabilistic Active Vision, Int. *Journal of Humanoid Robotics* (in press), 2010.

[22] J. Hertzberg; H. Jaeger, U. Zimmer, P. Morignot, A Framework for Plan Execution in Behavior-Based Robots, Proceedings, pp. 8-13, 1998.

[23] Y. Hoshino, T. Takagi, U. Di Profio, M. Fujita, Behavior Description and Control Using Behavior Module for Personal Robot, *Proc. IEEE conf. Robotic and Automation*, 2004.

[24] T. Huntsberger, P. Pirjanian, Trebi-Ollennu, H. Aghazarian, H. Das, S. Joshi, P. Schenker, CAMPOUT: A Control Architecture for Tightly Coupled Coordination of Multirobot System for Planetary Surface Exploration, *Proc. SPIE Conf. Sensor Fusion and Decentralized Control in Robotic System III*, 2000.

[25] J. Hurdus et al, Victor Tango Architecture for Autonomous Navigation in the Darpa Challenge, *Journal of the Aerospace, Computing, Information, and Communication*, Vol. 5, December 2008.

[26] H. Jaeger, T. Christaller, Dual Dynamics: Designing Behavior Systems for Autonomous Robots, *Artificial Life and Robotic*, Vol. 2 No. 3, pp. 108-112, Springer Japan 1998.

[27] D. Korterkamp, and R. Simmons, Robotic Systems Architectures and Programming, Chapter A.8, in *Springer Handbook of Robotics*, B. Siciliano & O. Khatib (Eds.), pp. 187- 206, Springer 2008.

[28] S. Lencer, J. Bruce, M. Veloso, A Modular Hierarchical Behavior-Based Architecture, *Lecture Notes in Computer Science* 2377, 2002.

[29] P. Maes, Modelling Adaptive Autonomous Agents, *Journal of Artificial Life*, Vol. 1, pp. 135-162, 1994.

[30] M. Matarić, Behavior-Based Control: Examples from Navigation, Learning and Group Behavior, *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9 No.2-3, 1997.

[31] M. Matarić, and F. Michaud, Behavior-based systems, Chapter E.38, in *Springer Handbook of Robotics*, B. Siciliano & O. Khatib (Eds.), pp. 891-909, Springer 2008.

[32] M. Müller, Hierarchical Activation Spreading: A Design pattern for Action Selection, *Proc. Workshop On Hybrid Control of Autonomous Systems*, pp. 63-70, 2009.

[33] P. Pirjanian, Behavior coordination mechanisms: State-of-the-art, *Tech Report* IRIS-99-375, Institute for Robotics and Intelligent System, University of Southern California, Los Angeles, California.

[34] M. Proetzsch, T. Luksch, and K. Berns, Development of complex robotic systems using the behavior-based control architecture iB2C, *Robotics and Autonomous Systems* 58 (2010), pp. 46-67.

[35] R. Renner, S. Behnke, Instability Detection and Fall Avoidance for a Humanoid using Attitude Sensors and Reflexes" Proceeding of the workshop on humanoid soccer robots of the 2006 IEEE-RAS international conference on humanoid robots, Italy, 2006.

[36] J. Ruiz-del-Solar, P. Guerrero, R. Palma-Amestoy, R. Marchant, J.M Yañez, UChile Kiltros 2009 Team Description Paper, *RoboCup Symposium 2009*, June 29 – July 5, Graz, Austria, 2009 (CD Proceedings).

[37] J. Ruiz-del-Solar, L. Locchil, et all, CHITA Hominids 2010 Team Description Paper, *RoboCup Symposium 2010*, June 19 – 25, Singapore, 2010 (CD Proceedings).

[38] T. Sawada, T. Takagi, M. Fujita, Behavior Selection and Motion Modulation in Emotionally Grounded Architecture for QRIO SDR-4X II, *Proc. IEEE Conf. on Intelligent Robots and Systems*, 2004.

[39] G.Schöner, M. Dose, A Dynamical System Approach to Task Level System Integration Used to Plan and Control Autonomous Vehicle Motion, *Robotics and Autonomus system*, Vol 10, 92.

[40] G. Steinbahuer, F. Wotawa, Enhacing Plan Execution in Dynamics Domains Using Model Based Reasoning, *Lecture Notes in Computer Science* 5314, pp. 510-519, 2008.

[41] A. Stroupe, M. Matrin, T. Balch, Distributed Sensor Fusion for Object Position Estimation by Multi-Robot Systems, *Proc. ICRA Conf*, Vol. 2, pp. 1092-1098, 2001.

[42] J. Testart, J. Ruiz-del-Solar, R. Schulz, P. Guerrero, R. Palma-Amestoy, A Real-Time Hybrid Architecture for Biped Humanoids with Active Vision Mechanisms, *JINT931R1*, accepted for publication *Journal of Intelligent and Robotic Systems*.

[43] Aldebaran robotics official website: http://www.aldebaran-robotics.com/

[44] Boost library official website: http://www.boost.org/

[45] RoboCup official website: http://www.robocup.org/

[46] RoboCup 2010 official website: http://www.robocup2010.org/

[47] RoboCup SPL league official website: http://www.tzi.de/spl/bin/view/Website/WebHome

[48] RoboCup Technical Committee, RoboCup Soccer Humanoid League Rules http://www.tzi.de/humanoid/pub/Website/Downloads/HumanoidLeagueRules2009.pdf

[49] RoboCup Technical Committee, RoboCup Standard Platform League (Nao) Rules http://www.tzi.de/spl/pub/Website/Downloads/Rules2009.pdf

VII. ANEXOS

7.1 PSEUDO CÓDIGO DEL MULTIPLEXOR DE LA CABEZA

```
muxHeadMultipleObjectActualization(scoreVector muxHeadScoreVector, visualInfo
perceptions)
{
for(i=0; i < scoreVector.getSize(); ++i)
{
    if (scoreVector[i].goal.multipleObjectives)
    {
            for( j=0; j< scoreVector[i].objects.getSize(); ++j)
            {
                    objectSeen = perceptions[scoreVector[i].objects[j].id].lastTimeSeen
                    < currentTime + TIME_NOT_SEEN_THREESHOLD ;
                    if(!objectSeen)
                    {
                            objectDeletedOnTracking(scoreVector[i].objects[j].id);
                            scoreVector[i].objects.delete[j];
                            --j;
                    }
                    else
                            objectUsedOnTracking(scoreVector[i].objects[j].id);

            }
    calculateMultitracking(scoreVector[i].objects);
    }
    else
            objectUsedOnTracking(scoreVector[i].objects[j].id);
    }
    if(objectDeletedFromGroup())
```

```
        {
                for(j=0; i<objectDeleted.getSize(); ++j)
                {
                        scoreVector.addScore(objectDeleted[j]);
                }
        }
}
```

La function *objectUsedOnTracking(scoreVector[i].objects[j].id)* es usada para tener la id de los objetos que hay en un grupo o en un objetivo simple. La función *objectDeletedOnTracking(scoreVector[i].objects[j].id)* se usa para saber si un objeto eliminado de un grupo ya está agregada como objetivo simple.

MuxHead Score actualization PseudoCode

```
for(i=0; i < scoreVector.getSize(); ++i)
{
        If( scoreVector[i].goal.multipleObjectives)
        {
                reactiveScore = 0;
                For( j=0; j < scoreVector.objects.getSize(); ++j)
                {
                        reactiveScore += scoreVector[i].objects[j].importance * (1-
                        perceptions[scoreVector[i].objects[j].id].confidence);
                }
        }
        else
        {
                reactiveScore = 1-perception[scoreVector[i].objects.id].confidence;
        }
        updatedScore =0.5*reactiveScore + 0.5* scoreVector[i].deliberativeScore;
}
```