UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

VISIÓN ACTIVA EN UN ROBOT HUMANOIDE ANTROPOMORFO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIRO CIVIL ELECTRICISTA

RODRIGO ANDRÉS SCHULZ SERRANO

PROFESOR GUÍA:
SR. JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
SR. HÉCTOR AGUSTO ALEGRÍA
SR. PABLO GUERRERO PÉREZ

SANTIAGO DE CHILE
JUNIO 2010

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: RODRIGO SCHULZ SERRANO
FECHA: 03 DE JUNIO DE 2010
PROF. GUIA: DR. JAVIER RUIZ DEL SOLAR

"VISIÓN ACTIVA EN UN ROBOT HUMANOIDE ANTROPOMORFO"

La estimación de la pose es un elemento de gran importancia para sistemas robóticos móviles que se desenvuelven en ambientes dinámicos. Existen diversas metodologías utilizadas para estimar esta pose, siendo esencial para el desempeño de todas ellas la calidad y abundancia de las observaciones obtenidas desde el ambiente.

El objetivo del presente trabajo es aumentar la cantidad y mejorar la calidad de las observaciones de un robot humanoide antropomorfo. Para esto se implementó un sistema de visión activa, el cual permite discernir qué objeto o grupo de objetos resulta más conveniente observar para reducir la incerteza en la estimación de la pose del robot, desarrollando con tal propósito la capacidad para observar simultáneamente más de un objeto.

Durante el período de trabajo se realizaron cuatro tareas principales para posibilitar el funcionamiento del sistema desarrollado. Estas consistieron en: (i) implementación de un detector de faros basado en el análisis de los puntos característicos de las regiones de color,(ii) habilitación en el simulador HL-Sim de la posibilidad de observar gráficamente las poses de los objetos presentes en el mapa local del robot, (iii) implementación de la funcionalidad de realizar seguimiento por posición, la que además contó con la capacidad de planificar trayectorias para la cabeza que consideren objetos extras durante el desplazamiento y de realizar seguimiento basado en la función de distribución de probabilidad de la ubicación del objeto, (iv) implementación de un algoritmo para discernir qué elementos observar, junto con la capacidad para observarlos.

Los resultados obtenidos muestran en el perceptor de faros una alta tasa detecciones correctas (92,76%) y a la vez una baja tasa de falsos positivos (1,1%), lo cual corresponde a un resultado satisfactorio. En la rutina de seguimiento, se observó coherencia entre el seguimiento por posición y el seguimiento visual (diferencia angular del orden de 1 a 3 grados), además de evidenciar el correcto funcionamiento de las capacidades implementadas. Por último, en la rutina de visión activa los resultados y el comportamiento observado, revelan un correcto funcionamiento bajo un espacio de acciones en el que se consideran posibles la observación de objetos individualmente o en forma grupal.

En conclusión, los resultados obtenidos revelan el buen funcionamiento de los métodos y algoritmos propuestos. Particularmente, el sistema implementado para la selección de los objetos a observar, a pesar de no mostrar una tendencia clara respecto a la reducción de la incerteza, permitió generar observaciones de diferentes objetos o grupos de estos, lo que resulta positivo para el sistema pues genera un mayor flujo de información que ingresa a éste.

Dedicatoria.

Con cariño a mi familia y mis amigos, mi segunda familia.

Agradecimientos.

Me gustaría agradecer a toda la gente que me ha apoyado durante este largo camino. A mi familia, por apoyarme en esta odisea. Gracias a mis papás por su preocupación desde que soy chico, a mi papá, a mi hermana y en especial a mi mamá, que siempre me apoyó y se preocupó de que los trasnoches fueran más agradables con un pancito y un vaso de bebida, así como de que los almuerzos fueran los mejores con un potecito de comida fresca.

También me gustaría agradecer a la gente del Laboratorio de Robótica de la Universidad de Chile por su gran ayuda en el desarrollo de este trabajo. A mi profesor guía Dr. Javier Ruiz del Solar por su confianza en el desarrollo de esta memoria.

Por último quisiera a agradecer a mis amigos, mi segunda familia, todos aquellos quienes hicieron más placentero el pasar por la universidad, gracias a quienes era posible tomarse con humor las tareas interminables y los controles imposibles, quienes compartimos noches enteras trabajando y otras tantas festejando. Gracias por estar ahí, por ser ustedes y por compartir esta odisea.

Gracias por el apoyo con el consejo de las pasas :)

Rodrigo Schulz Serrano

Santiago, Junio de 2010.

El presente trabajo fue financiado parcialmente por el proyecto FONDECYT 10900250.

Índice de Contenidos.

Contenidos

Índice de Figuras.

Índice de Tablas.

1. Introducción.

El presente trabajo se desarrolla en el marco de las actividades realizadas por el equipo del Laboratorio de Robótica del Departamento de Ingeniería Eléctrica de la Universidad de Chile.

1.1. Motivación.

Para el hombre siempre ha sido de interés, y en parte un sueño, el crear un ser semejante en forma y fondo. La robótica es una de las disciplinas que aborda esta idea de manera directa, intentando emular diversas características de un ser humano y siendo un área de particular interés el desarrollo de robots autónomos, es decir robots capaces de actuar de forma independiente, tomando sus propias decisiones de acuerdo al ambiente.

Esto es un desafío de gran dificultad por diversas razones, entre las que se puede destacar que el ambiente en que se desarrollan las actividades humanas tiende fuertemente al cambio, además de ser altamente complejo. Por esto es necesario el contar con sistemas sensoriales capaces de adquirir la información necesaria sobre el ambiente, siendo la visión uno de los sentidos que permite obtener la mayor cantidad de información, pero que a la vez es el más complejo de reproducir de manera artificial.

Por otro lado, aunque puedan parecer elementales tareas como conocer la propia ubicación de forma relativa a los objetos y de forma absoluta dentro del ambiente, representa un problema difícil para la robótica y para el cual no se conoce una solución única. Este problema es conocido como la localización del robot y constituye un elemento fundamental para el desarrollo de cualquier actividad, ya que dependiendo de la calidad con

que esta se lleve a cabo, tareas posteriores podrán ser mejor abordadas. En particular, uno de los factores que influye de manera importante en la localización es la identificación de ciertos elementos en el ambiente denominados *landmarks* o puntos de referencia, los cuales tienen posiciones absolutas conocidas y por tanto permiten estimar la propia ubicación dentro del ambiente, de la misma forma que lo haría un humano al reconocer un objeto y estimar la profundidad de una sala, por ejemplo.

Por esta razón es importante la información proveniente de las observaciones hechas por los sensores, y dado que estos sólo pueden obtener a la vez datos de partes aisladas del ambiente y no simultáneamente de todo este (puesto que en presentan limitaciones como la existencia de un campo sensorial limitado), también resulta importante que tales observaciones sean realizadas en diferentes puntos. Entonces ¿no sería de gran ayuda para la localización del robot contar con un algoritmo que indique hacia qué lugares resulta más conveniente observar para lograr una mejor estimación de la propia posición? ¿No sería útil poder evaluar esto último de forma concurrente a la tarea que se esté realizando, tal como los humanos observan a su alrededor para orientarse mientras llevan a cabo alguna actividad? Un algoritmo orientado a evaluar este último punto es lo que presenta P. Guerrero, J. Ruiz-del-Solar y M. Romero en [1], el cual apunta a reducir la incerteza en la localización del robot durante la ejecución de otra tarea.

1.1.1. La Robótica Móvil en General.

Como ya se mencionó, la robótica es una disciplina que busca crear máquinas capaces de emular diferentes capacidades del ser humano. En particular la robótica móvil se enfoca en el desarrollo de robots que a pesar de presentar diferentes funciones y capacidades, comparten como característica el poder desplazarse de manera autónoma.

Para poder contar con esta última característica es necesario que el robot presente un sistema de locomoción adecuado, junto con un mapa de su entorno o en su defecto que al menos sea capaz reconocer elementos dentro de este, de modo de poder desplazarse libremente y sin chocar. En este punto es que adquiere importancia la localización del robot, puesto que sin una adecuada localización, la existencia de un mapa del entorno no representa ninguna ventaja para el robot.

1.1.2. El Fútbol Robótico y La Robocup.

El fútbol es una actividad compleja en la que intervienen diversas habilidades del jugador, tales como ser capaz de estimar las posiciones de los objetos dentro del campo de juego, ser capaz de estimar la trayectoria de la pelota, tomar decisiones acertadas según el estado del juego, etc. De forma análoga, en el fútbol robótico es necesario contar con tales habilidades, lo que implica un constante desafío en todo sentido, puesto que el juego se desarrolla en un ambiente dinámico.

Bajo este marco, el fútbol robótico ha sido un ambiente fructífero para el desarrollo de nuevos algoritmos y sistemas, los que a pesar de ser relativos al contexto de robótica y estar desarrollados para una tarea específica, generalmente extensibles a nuevas situaciones.

La RoboCup[1] es una organización internacional dedicada a fomentar el desarrollo de la robótica y la inteligencia artificial. Para esto la Robocup organiza anualmente una competencia internacional de robótica en la cual se lleva a cabo un mundial de fútbol robótico, así como competencias en otras categorías. Estas tienen por objetivo final el llegar a contar con robots capaces de abordar exitosamente un juego de fútbol, siendo incluso capaces de vencer a un equipo de fútbol humano.

1.1.2.1. El Juego y sus Reglas.

Las reglas del fútbol robótico [4] (en sus distintas categorías[2]) son, en términos generales, las mismas que las existen en el fútbol humano, es decir, se juega con los pies, en un terreno limitado y el objetivo es anotar en el arco del equipo adversario. No obstante existen ciertas restricciones adicionales para hacer el juego más apropiado al desarrollo de los equipos y las capacidades de los robots, como por ejemplo restricciones en el tiempo de juego, tiempo máximo que puede transcurrir sin que un robot toque la pelota, número de jugadores en el campo, tiempo que puede permanecer inactivo un robot, etc.[3]

---

[1] Más información puede ser encontrada en www.robocup.org

[2] Existen varias categorías en la competencia, entre las que se pueden mencionar *"Simulation"*, *"Small Size"*, *"Middle Size"*, *"Standard Platform"* (o SPL) y *"Humanoid"*.

[3] Más información sobre las reglas de cada liga puede ser encontrada en la página de la organización, www.robocup.org

El campo de juego en el que se desarrolla la acción es como el que se presenta en la Figura 1.1, mostrando algunas variaciones en el tamaño dependiendo de la liga. En particular, las dimensiones del campo de juego de la Figura 1.1 son las correspondientes a las utilizadas en la liga SPL o Liga de Plataforma Estándar. Adicionalmente las dimensiones de los arcos y los eventuales puntos de referencia (*landmarks*) existentes dentro del campo de juego se encuentran normados para cada liga.

Por último, es pertinente señalar que todos los elementos del campo de juego (cancha, líneas, arcos, pelota, puntos de referencia, etc.) son coloreados con tonalidades particulares y que se suponen conocidas de antemano, permitiendo la aplicación de sistemas de visión basados en color.

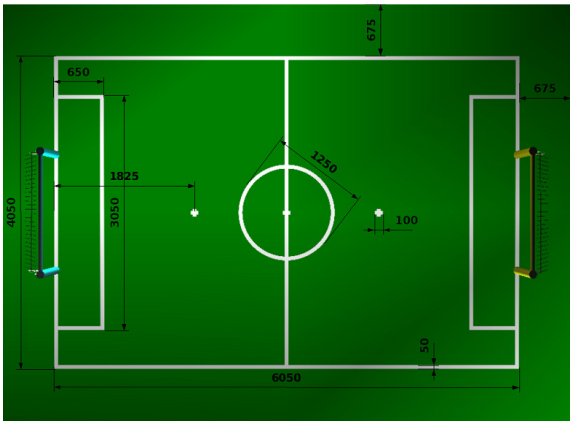


Figura 1.1: *Campo de juego de la liga SPL.*

1.1.3. El Problema.

Un robot jugando en la posición de arquero debe estar simultáneamente mirando a la pelota y moviéndose para cubrir el arco, por lo tanto si desea mantener una estimación de su posición actual debe confiar en la información proveniente de los sensores de sus motores. Dicha información le permite evaluar cuánto se ha movido respecto a su situación anterior, lo que es conocido como la *odometría* del movimiento.

Sin embargo debido a imprecisiones existentes en la odometría (puesto que los sensores de los motores no son perfectos), si el robot únicamente observara la pelota y no a los objetos usados como referencias espaciales, el error existente en la localización crecería hasta niveles inaceptables. Para evitar esto es necesario alternar la atención del robot entre

el mirar a la pelota y a los objetos de referencia dentro del campo de juego, siendo una estrategia para llevar a cabo la selección del objeto a observar lo que se propone en [1].

Para lograr lo antes descrito es necesario definir un espacio de acciones sensoriales posibles con las cuales trabajar, espacio que de manera general puede ser definido como *los comandos de acción dados directamente a los actuadores presentes en la cabeza del robot y con los cuales se modifica la dirección hacia la cual se enfoca la cámara*, quedando el espacio de acciones resultante de esta definición dado por *cada una de las posiciones posibles de la cabeza del robot*. A pesar de esto, en [1] se utiliza un espacio definido como *mirar hacia un objeto*, existiendo un número limitado de objetos en los que el módulo de visión puede fijar su atención. Por lo tanto, el espacio de acciones se encuentra discretizado de acuerdo a las posiciones de los objetos de interés (las cuales pueden variar con el tiempo).

Esto corresponde a una simplificación del problema e implica que en cada observación lo más probable será obtener la información de un único objeto, corrigiendo la localización solamente con tal información y despreciando la posibilidad de ver dos o más objetos al mismo tiempo, lo que aportaría más información al sistema y permitiría realizar una mejor corrección a la localización. De la misma forma, al llevar la cámara a la posición deseada se podrían observar algunos de los objetos de interés durante el trayecto si se considerara un espacio de acción menos restrictivo que solamente enfocar en uno de los objetos de interés.

Es en estos dos últimos puntos es donde se enmarca el trabajo de memoria que se presentará a lo largo de este informe, el que consistió en realizar una mejora a la implementación del algoritmo presentado en [1], particularmente modificando el espacio de acciones sensoriales posibles para llegar a uno menos restrictivo y que permita considerar la incertidumbre en las posiciones de los objetos y la posibilidad de ver más de un objeto simultáneamente.

1.2. Equipo UChile.

El equipo de robótica UChile corresponde a un proyecto del Departamento de Ingeniería Eléctrica de la Universidad de Chile, el cual está enfocado en el estudio e

investigación de temas relativos a la robótica y la visión computacional, contando con numerosas publicaciones sobre estos tópicos.

Este equipo ha participado en reiteradas ocasiones en las competencias del mundial de robótica organizado por la Robocup, participando la liga de robots de servicio "@Home", así como en las ligas de fútbol robótico "SPL" (Standard Platform League o Liga de Plataforma Estándar) y "Humanoid", obteniendo importantes reconocimientos.

1.2.1. Arquitectura de la Librería UChile.

La arquitectura de software de la librería [2] del equipo UChile está dividida en dos secciones principales: una independiente de la plataforma y otra dependiente de esta. Esta división está enfocada al desarrollo de métodos y funciones que sean aplicables a diferentes plataformas mediando la menor cantidad posible de cambios.

La etapa dependiente de la plataforma contiene las comunicaciones entre las unidades de hardware de cada robot y la etapa independiente de la plataforma. Por su parte esta última está relacionada con la planificación y los objetivos de alto nivel, los procesos perceptuales y el modelamiento del mundo.

La etapa independiente de la plataforma está subdividida en 4 módulos funcionales: visión, actuación, localización y estrategia como se muestra en la Figura 1.2.

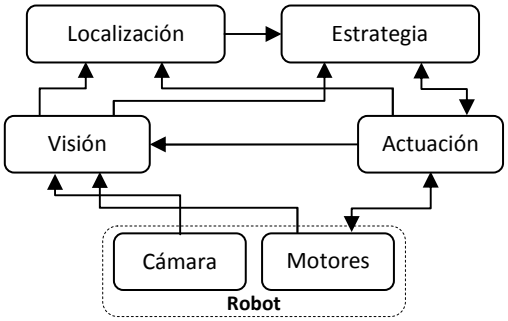


Figura 1.2: *Arquitectura de software de la librería utilizada por el equipo UChile.*

Los módulos de visión y actuación se comunican directamente con la etapa dependiente de la plataforma para interactuar con el hardware del robot para enviar y recibir información de los actuadores, así como obtener datos de los sensores visuales. El módulo de localización estima las posiciones de los objetos y la posición absoluta del robot

dentro el ambiente. Finalmente el módulo de estrategia será detallado brevemente en la sección 1.2.2.

1.2.2. Módulo de Estrategia.

El módulo de estrategia coordina las acciones de los diferentes robots presentes en la cancha, realizando la asignación de roles (arquero, atacante, etc.) de manera dinámica, de acuerdo al estado del juego y a las posiciones de los jugadores, así como también es quien toma las decisiones sobre las acciones a realizar por cada robot.

Este módulo está compuesto por 3 capas funcionales: la capa de planificación, la capa deliberativa y la capa reactiva. Cada una de ellas cumple distintas funciones dentro de la arquitectura, mostrando diferentes velocidades de respuesta y capacidades de planeamiento. Además estas basan su funcionamiento en la ejecución de *behaviors* o rutinas de comportamiento, las cuales están enfocadas en cumplir con una determinada tarea, por ejemplo ir a la pelota.

La capa de planificación está encargada de planear las acciones a largo plazo y para ello utiliza la información proveniente de todos los robots en el campo de juego, asignando roles y coordinando las acciones. La capa deliberativa ejecuta la misión definida por la capa de planificación, utilizando para ello rutinas de comportamiento organizadas como una máquina de estado donde cada estado corresponde a una rutina de comportamiento distinta. De tal manera, la capa deliberativa configura y entrega los parámetros necesarios a las rutinas de comportamiento para poder llevar a cabo la misión impuesta. Por último, la capa reactiva es la encargada de discernir qué acciones deben y pueden ser ejecutadas de acuerdo a los recursos disponibles en cada instante. En la Figura 1.3 se presenta un diagrama de la arquitectura utilizada en la librería UChile

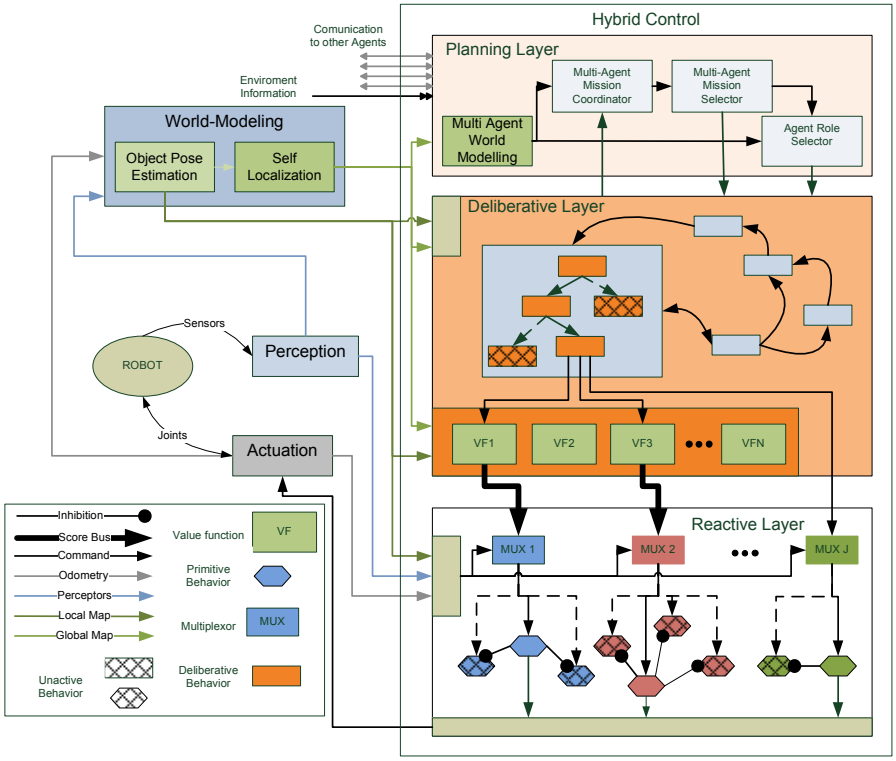


Figura 1.3: *Arquitectura de la librería UChile.*

Se puede hallar información más detallada sobre el funcionamiento de esta arquitectura en [2] y [3].

1.2.3. Simulador de Alto Nivel HL-Sim

El simulador HL-Sim, es una herramienta para el desarrollo y evaluación de estrategias de alto nivel, permitiendo evaluar de manera controlada el funcionamiento del módulo de estrategia.

Dado que el objetivo de HL-Sim es la simulación del comportamiento de alto nivel del robot, éste no se ocupa de emular variables de más bajo nivel como la posición de cada motor del robot o el proceso de captura y procesamiento de datos visuales. El simulador HL-Sim únicamente reproduce el funcionamiento de los módulos de localización y estrategia tal como funcionaría en los robots, generando de forma artificial datos provenientes de los módulos de actuación y visión, para el funcionamiento de los módulos de más alto nivel.

1.3. Objetivos.

Dada la problemática a resolver, presentada en el punto 1.1.3, los objetivos planteados para este trabajo de título son los siguientes.

- Desarrollo e implementación de una mejora al algoritmo de visión activa propuesto en [1], ampliando para ello el espacio de acciones sensoriales posibles de realizar. Tal ampliación consiste en pasar de sólo poder enfocar la atención visual en un objeto a poder también fijar la atención en un punto determinado, en el cual se puede obtener información visual de varios objetos a la vez (cuando esto sea posible).
- Realizar una evaluación del desempeño del algoritmo luego de las modificaciones.

1.3.1. Objetivos Específicos.

- Cambiar la forma de caracterizar la ubicación de los objetos para pasar de un único punto a una función de distribución de probabilidad (discretizada mediante un muestreo determinístico), en los casos que exista una alta incertidumbre asociada a la ubicación del objeto de interés.
- Implementar un algoritmo para la generación de trayectorias con las cuales sea posible recorrer los puntos de la función de densidad de probabilidad relativa al punto anterior. De este modo, ser capaz de realizar seguimiento visual basado en la distribución de probabilidad de la ubicación del objeto de interés.
- Considerar en la generación de las trayectorias la posibilidad de observar otros objetos durante el recorrido de esta y también al momento de enfocarse en un punto particular.

1.4. Estructura de la Memoria.

El trabajo de memoria que se presenta a continuación está organizado en 5 capítulos. En estos se revisan los antecedentes, el trabajo realizado y los resultados obtenidos y sus conclusiones.

En el capítulo 2 se hace una revisión de los antecedentes generales que es necesario tener presente para comprender a cabalidad el trabajo realizado

En el capítulo 3 se presenta el trabajo llevado a cabo. En la sección 3.1 se presentan las modificaciones efectuadas al simulador de alto nivel HL-Sim, las cuales fueron necesarias para efectuar un mejor desarrollo y evaluación de los métodos implementados.

En la sección 3.2 se detalla la implementación de un perceptor de faros, utilizado por la liga Humanoide de la Robocup y también utilizado intensamente durante este trabajo.

En la sección 3.3 se describen las modificaciones en implementación de nuevos métodos, realizados en la rutina de seguimiento visual de objetos.

En la sección 3.4 se describe en detalle las modificaciones efectuadas en la rutina de visión activa para llevar a cabo la ampliación del espacio de acciones sensoriales.

En el capítulo 4 se describen las pruebas realizadas al trabajo desarrollado, se presentan los resultados obtenidos de dichas pruebas y se lleva a cabo el análisis de tales resultados.

Finalmente en el capítulo 5 se lleva a cabo la discusión de los comportamientos observados durante las pruebas y los resultados arrojados por estas, estableciendo las conclusiones obtenidas del trabajo desarrollado.

2. Antecedentes Teóricos.

2.1. Trabajos Anteriores.

En la literatura es posible encontrar diversos métodos para la reducción de la incerteza existente en la localización de un robot mediante el uso de datos visuales. De la misma forma es posible encontrar trabajos que presentan aplicaciones de visión activa enfocadas a distintos objetivos, encontrando trabajos enfocados en la estimación de mapas 3D del ambiente, planeado por Flandin [7]. Algunas aproximaciones al problema de localización de un robot mediante el uso de visión activa han sido llevadas a cabo por Krose [5], ideas que también ha sido extendidas a problemas más complejos como la localización de vehículos en exteriores mediante visión activa, como lo presenta Tessier [6].

De forma particular, el algoritmo expuesto por P. Guerrero, J. Ruiz Del Solar y M. Romero en [1] presenta un método para reducir la incerteza en la localización de un robot móvil durante la ejecución de alguna tarea en específico, estimando la conveniencia de observar los diferentes objetos dentro del ambiente y basándose en el principio de visión activa para llevar a cabo la acción sensorial más conveniente desde el punto de vista de la reducción de la incerteza en la localización.

2.1.1. Rutina de Visión Activa: Algoritmo Básico.

El objetivo principal de este algoritmo es reducir la incerteza existente en la localización del robot, para lo cual el método se busca determinar cuál de las acciones de sensoriales disponibles es la genera la mayor reducción de la incertidumbre.

Así, en términos generales el funcionamiento del algoritmo se basa en simular una iteración del filtro de Kalman que realiza la estimación de la pose del robot, para ello simulando también los movimientos y las eventuales observaciones que podrían darse como fruto de tales movimientos.

En consecuencia se llevan a cabo las etapas predictiva y correctiva del filtro de Kalman, para la primera utilizando la simulación de la política de juego del robot como fuente de odometría (simulada) y para la segunda simulando las eventuales observaciones que tendría el robot de acuerdo a su posición.

En este punto es conveniente señalar que si bien el espacio de acciones sensoriales posibles del robot en un principio corresponde a "todas las posiciones posibles de la cabeza en el intervalo de ángulos permitidos por las limitaciones físicas del robot", para este algoritmo en particular se utilizó un espacio de acciones sensoriales simplificado, definido como *mirar hacia un determinado objeto*. En consecuencia las observaciones simuladas deben ser afines con el espacio de acciones sensoriales utilizado, y por lo tanto estas se asume que sólo es posible ver un único objeto (lo que correspondería a una nueva simplificación).

Posteriormente al simular una visión de un determinado objeto (por ejemplo un arco), se lleva a cabo la etapa correctiva del filtro de Kalman donde se estima la que sería la pose corregida del robot si este realmente viera dicho objeto. Con dicha pose se evalúa una función de valor definida de forma particular para la tarea que se esté llevando a cabo, obteniendo así un puntaje que indica la conveniencia que reviste realizar dicha observación de manera real y que además permite discernir de entre todas las acciones sensoriales posibles, cuál resulta más beneficiosa.

Para el cálculo de dicho punta y la selección del objeto más conveniente es posible utilizar diferentes criterios, minimización de la varianza, maximización del espacio de acciones, etc. De manera particular, el criterio utilizado en este algoritmo corresponde a la minimización de la varianza de la función de valor, lo que será explicado con mayor profundidad en la Sección 2.1.3.

Un esquema del funcionamiento básico del algoritmo se presenta en la Figura 2.1.

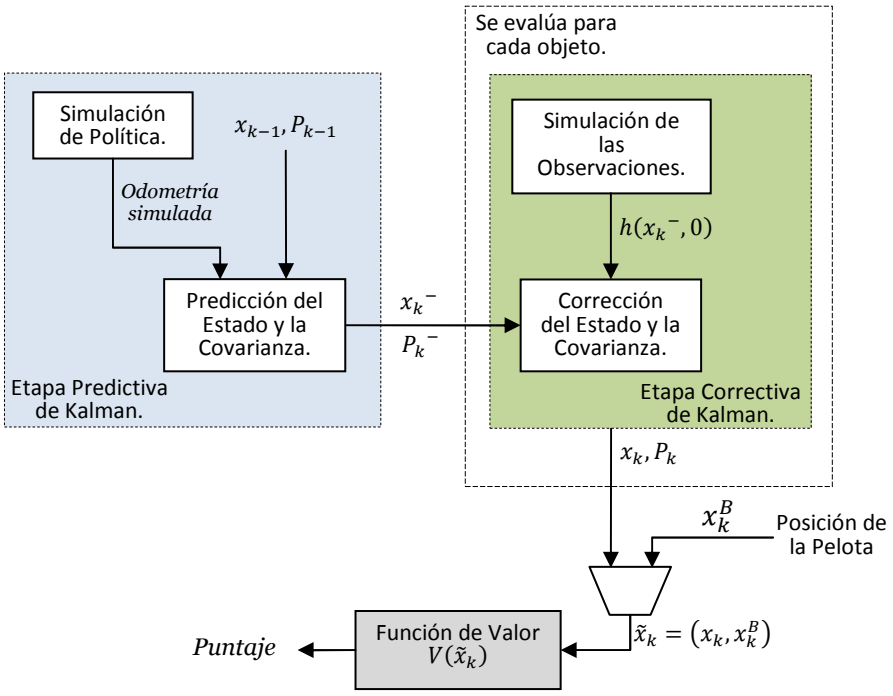


Figura 2.1: *Esquema básico de funcionamiento del algoritmo presentado.*

Adicionalmente, es necesario hacer la aclaración que para efectos del algoritmo las acciones del robot pueden ser divididas en 2 grupos, acciones motoras (las que modifican la pose del robot y no afectan las observaciones) y acciones sensoriales (las que no afectan la pose sino que sólo pueden modificar la posición de la cámara, afectando las observaciones). Esta suposición en la práctica es completamente realizable, siendo únicamente necesaria para ello la existencia de una cámara actuada e independencia en los movimientos de dicho actuador.

2.1.2. Probabilidades de Observación y Muestreo.

Un punto importante a considerar en el algoritmo descrito son las probabilidades de observar un objeto, dado que de esa probabilidad dependerá también el puntaje asignado a la eventual visión de cada uno de los elementos incluidos en el algoritmo.

Haciendo referencia a esto, la probabilidad de observar un objeto corresponderá a la probabilidad de que este sea realmente enfocado por la cámara y que además sea detectado por el módulo de visión del robot. Luego si se definen los eventos $\kappa_{i,k}$ como “en el momento k el robot enfoca el i-ésimo objeto” y $\varphi_{i,k}$ como “en el momento k el módulo de visión del

robot detecta el i-ésimo objeto", se tiene que la probabilidad de observar un determinado objeto, dado que este es enfocado, será:

$$\alpha_{i,k} = p(\varphi_{i,k}|\kappa_{i,k}) = \int_{x_k \epsilon X} p(\varphi_{i,k}|x_k, \kappa_{i,k}) p(x_k|\kappa_{i,k}) dx_k = \int_{x_k \epsilon X} p(\varphi_{i,k}|x_k, \kappa_{i,k}) p(x_k) dx_k \quad (2.1)$$

Donde se utilizó que $p(x_k|\kappa_{i,k}) = p(x_k)$ debido a la descomposición en acciones sensoriales y motoras, señalada en el punto anterior. No obstante, dado que a priori no es posible realizar dicho cálculo de manera continua, es necesario tener una representación discreta de la función de distribución de probabilidad, para lo cual se pueden utilizar diferentes formas de muestreo, en particular la descomposición en valores singulares (*Singular Value Decomposition* o SVD) con lo cual se obtienen pares $\{\chi_i, \omega_i\}$ correspondientes a los valores de la variable a la probabilidad asociada a este.

Utilizando esta representación la probabilidad de observar un objeto queda como sigue:

$$\alpha_{i,k} \approx \sum_{j \epsilon [0,2N]} p(\varphi_{i,k}|x_k = \chi_k^j, \kappa_{i,k}) \omega_{j,k} \quad (2.2)$$

Conjuntamente el evento $\varphi_{i,k}$ puede ser descrito como la ocurrencia simultánea de los 3 eventos siguientes:

- $\upsilon_{i,k}$: el objeto está en el campo de visión de la cámara.
- $\vartheta_{i,k}$: el objeto i-ésimo no es ocluido por otros objetos.
- $\rho_{i,k}$: cámara y módulo de visión son capaces de percibir el objeto.

Dado que $\upsilon_{i,k}$, $\vartheta_{i,k}$ y $\rho_{i,k}$ son eventos independientes entre si, se llega a la expresión $p(\varphi_{i,k}|x_k, \kappa_{i,k}) = p(\upsilon_{i,k}|x_k, \kappa_{i,k}) p(\vartheta_{i,k}|x_k, \kappa_{i,k}) p(\rho_{i,k}|x_k, \kappa_{i,k})$, donde el primer término puede ser calculado usando las posiciones del robot y los objetos, el segundo término puede ser estimado usando la pose del robot y la posición del objeto y el tercer término puede ser estimado como una función de la distancia entre el robot y el objeto, algunos parámetros de la cámara y del módulo de visión de la librería.

Finalmente utilizando ambas expresiones es posible obtener la probabilidad de observación de un objeto $\alpha_{i,k}$.

2.1.3. Función de Valor y Conveniencia de las Observaciones.

La determinación de la conveniencia de observar cada uno de los diferentes objetos, y por tanto la evaluación del criterio para la selección de la acción sensorial a realizar, se lleva a cabo mediante una función de valor definida específicamente para la tarea que se esté ejecutando. Concretamente el criterio utilizado consiste en minimizar la varianza de la función de valor de la tarea, esto basándose en la suposición que las componentes de incertidumbre más importantes son las que hacen variar con mayor fuerza la función de valor, y por lo tanto al minimizar la varianza de dicha función se estará minimizando dichas componentes. La varianza de la función de valor de la tarea al enfocar el i-ésimo objeto dentro del campo de juego está definida como:

$$\sigma_{i,k}^{y} = var\left(V_k(\tilde{x}_k)|k_{i,k}\right) \tag{2.3}$$

Donde $\tilde{x}_k$ es el vector de estado extendido, el cual tiene la información del estado (posición absoluta del robot) y la posición de la pelota de forma relativa al robot. Dicha expresión puede ser evaluada siguiendo 3 etapas consecutivas:

I. Utilizando un filtro de Kalman y los modelos del proceso y observacional, $x_{k+1} = f(x_k, u_k, w_k)$ y $z_k = h(x_k, u_k, v_k)$ respectivamente, se obtiene el estado y la covarianza corregidos debido a la observación del i-ésimo objeto x_{k+1}^{i} y P_{k+1}^{i}. La observación de dicho objeto es simulada mediante el modelo observacional. Adicionalmente se determina la probabilidad de observación asociada a tal objeto $p\left(x_{k+1}|\varphi_{i,k+1}\right)$.

II. Utilizando la definición de la varianza de una variable aleatoria, esto es $var(X) = S - E(X^2)$, se determina la varianza de la función de valor al ser evaluada con el vector de estado extendido:

$$\bar{V}_{\phi,k} = \int_{x_k \epsilon X} V_k(x_k) p(x_k|\phi_k) dx_k \tag{2.4}$$

$$S_{\phi,k} = \int_{x_k \epsilon X} V_k(x_k)^2 p(x_k|\phi_k) dx_k \tag{2.5}$$

Estas expresiones pueden ser evaluadas utilizando una representación discreta de la creencia a través de Puntos Sigma $\{\chi_{i,k+1}^{j}, \omega_{i,k+1}^{j}\}$, los cuales son posibles de obtener como se señaló en el punto anterior.

III. Con los datos señalados es posible calcular la varianza de la función de valor al momento de observar el i-ésimo objeto, proceso que se repite para cada uno de los objetos presentes en el campo de juego.

Finalmente, el criterio para la selección de la acción sensorial a realizar es la minimización de la varianza de la función de valor. Luego el objeto que resulta más conveniente observar en términos de la corrección de la localización del robot será aquel cuya observación resulte en una menor varianza en el cálculo de la función de valor.

2.1.4. Política de Juego.

La política de juego corresponde a las acciones que lleva a cabo el robot de acuerdo al estado del juego y la información que recibe desde el ambiente, luego la política en términos generales obedece al rol que el robot cumple dentro de la cancha. De forma particular a esta implementación y dado que el algoritmo presentado es de tipo orientado a tareas (lo cual se refleja en el uso de una función de valor específica a la tarea que se esté ejecutando), la tarea definida como objetivo es la "*cobertura del arco*", debido a las razones expuestas en la presentación del problema en la sección 1.1.3.

2.1.4.1. Cobertura del Arco.

La rutina de cobertura del arco apunta a posicionar al robot arquero de manera tal de minimizar la probabilidad de recibir un gol en contra. Para ello, el robot se posiciona de forma que un eventual robot atacante tenga dos ángulos libres iguales a ambos lados del arquero, los cuales sean mínimos, logrando cubrir lo mejor posible el arco y a la vez estando sujeto a la restricción de no salir del área penal. En la Figura 2.2 se pueden observar los ángulos β_{izq} y β_{der}, correspondientes a los ángulos libres entre atacante y el arco.

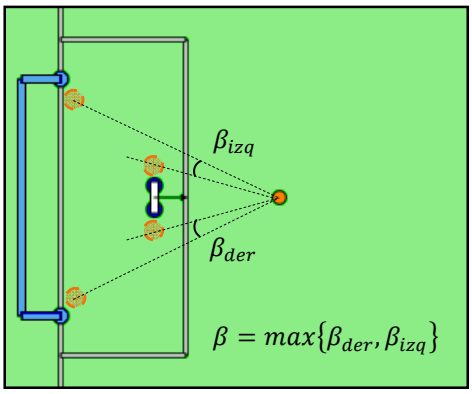


Figura 2.2: *Política de juego del robot arquero.*

Así, la tarea de cobertura del arco busca encontrar la distancia d^* desde la pelota tal que β sea mínimo y estando sujeto a mantenerse dentro del área penal, logrando eventualmente que $\beta = 0$, lo que se traduce en un arco completamente cubierto.

2.2. Visión Computacional.

La visión computacional busca desarrollar sistemas de visión artificiales basados en el procesamiento de las imágenes obtenidas con una cámara, generado de tal manera un sistema sensorial avanzado y complejo, el cual aportan una mayor cantidad de información acerca del entorno que otros sistemas sensoriales (como lasers, emisores IR, etc.).

2.2.1. Visión Activa.

La visión activa en términos simples consiste en el control o manipulación de una cámara utilizada como sensor visual, con el fin de mejorar la percepción del ambiente y obtener una mayor cantidad de información desde éste.

En general las estrategias utilizadas radican en el control del ángulo al que es dirigida la cámara. No obstante, también es posible desarrollar estrategias que consideren elementos tales como las regiones de interés dentro de la imagen, parámetros de la cámara, etc.

2.2.2. Segmentación de Colores.

Dada la complejidad que representa en desarrollar un sistema de visión artificial que intente emular el sistema visual humano y en particular dado las limitaciones de procesamiento existentes en los diferentes sistemas, a menudo se realizan simplificaciones

en los sistemas de visión tanto en la forma en que estos trabajan como en la aplicabilidad de cada uno. Estas simplificaciones están enfocadas principalmente a mejorar o facilitar la percepción de los objetos de interés y a la vez reducir la intensidad de los cálculos necesarios para ello, lo que en general se traduce en un menor tiempo de procesamiento.

En este contexto se enmarca la visión basada en la segmentación de colores, la que consiste en observar los objetos del ambiente reconocibles por su forma y por su color característico. En particular, para esta simplificación se reconocen 2 etapas principales: la segmentación y la generación de regiones de color o *blobs*.

- La segmentación corresponde al proceso de clasificación de los píxeles de la imagen de acuerdo a su color. Para esto es posible utilizar clasificadores estadísticos o de algún otro tipo. De tal manera, los píxeles pertenecientes a una determinada clase son reemplazados o representados por el color asociado a dicha clase, información que luego pasa a la etapa de generación de regiones de un mismo color.

En general la segmentación es un proceso estático con respecto al ambiente. Es decir los píxeles de la imagen son clasificados usando las siempre las mismas reglas. Esto hace posible el uso tablas de datos en la clasificación, las que pueden ser generadas mediante una etapa de entrenamiento del clasificador. Así el proceso de segmentación puede ser reducido únicamente a la consulta de dicha tabla para cada uno de los píxeles de la imagen, disminuyendo el tiempo de procesamiento requerido.

Sin embargo, cabe señalar que para que la segmentación mantenga su característica de estática respecto al ambiente, este debe presentar condiciones de iluminación controladas, dado que la tabla generada pierde validez al variar la iluminación.

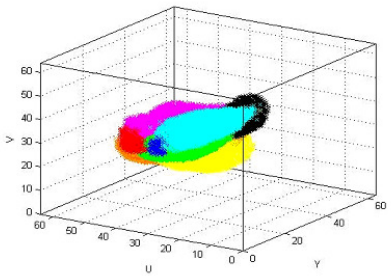


Figura 2.3: *Ejemplo de una tabla de valores (Look Up Table) utilizada en la segmentación.*

- La formación de regiones es la agrupación de píxeles contiguos que pertenezcan a la misma clase. En esta etapa es factible utilizar filtros de distinta naturaleza sobre la imagen segmentada con el objetivo de alcanzar una generación de regiones más representativa de la imagen original.

El uso de este tipo de simplificaciones agiliza el funcionamiento de los sistemas de visión, sin embargo condiciona el ambiente en el cual estos pueden ser aplicados, como en este caso condicionando el color de los elementos presentes.

2.2.2.1. Puntos Característicos.

Entre los diferentes tipos de información que es posible determinar para cada región, tales como centro de masa, tamaño, etc., es posible determinar un set de ocho puntos que corresponden a la intersección entre el borde de la región y las curvas de nivel en las direcciones que se muestran en la Figura 2.4.

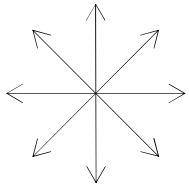


Figura 2.4: *Direcciones de los puntos característicos.*

Se hace particular énfasis en estos descriptores ya que son parte de la información determinada por el módulo de visión de la librería UChile en la etapa de generación de regiones, y fueron utilizados en el trabajo que se presentará más adelante en este documento.

2.3. Filtro de Kalman.

El filtro de Kalman es un algoritmo para la estimación de las variables de estado de un sistema dinámico que está bajo la acción de ruido de origen estocástico. Este método tiene la ventaja de estimar de forma recursiva los parámetros del sistema y además permite estimar de manera óptima las variables de estado del sistema. Inicialmente este algoritmo supone la existencia de un sistema lineal y los efectos de ruido observacional de tipo Gaussiano, sin embargo para los casos en que el supuesto de linealidad no se cumple existen

extensiones de este filtro, conocidas como *Extended Kalman Filter* (EKF) y *Unscented Kalman Filter* (UKF).

2.3.1. Algoritmo de Kalman.

El filtro de Kalman basa su funcionamiento en el uso de dos modelos f y h, ambos lineales y que representan diferentes aspectos del sistema. La primera corresponde al modelo del proceso y relaciona el estado futuro x_k con el estado actual x_{k-1}, la entrada del sistema en el instante actual u_{k-1} y una variable aleatoria w_{k-1} que representa el ruido del proceso y que debe ser de origen Gaussiano.

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}) = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{2.6}$$

La segunda función corresponde al modelo del proceso observacional y relaciona la observación o medición actual z_k, con el estado actual x_k y una variable aleatoria para representar el ruido observacional v_k.

$$z_k = h(x_k, v_k) = Hx_k + v_k \tag{2.7}$$

Luego el algoritmo del filtro de Kalman se compone de una etapa predictiva y una etapa correctiva posterior a la predictiva, configurando así una estrategia de estimación de tipo recursivo.

I. Etapa Predictiva.

En esta etapa el algoritmo realiza una predicción del estado futuro x_k^-, basado en el modelo del sistema y los datos actuales (estado actual y entrada actual) y además se estima la matriz de covarianza del error de la estimación del estado P_k^-.

Las ecuaciones que describen esta etapa en términos generales son:

$$x_k^- = f(x_{k-1}, u_{k-1}, 0) \tag{2.8}$$

$$P_k^- = A_k P_{k-1} {A_k}^T + W_k Q_{k-1} {W_k}^T \tag{2.9}$$

Donde A_k y W_k corresponden a los jacobianos (en el caso de ser sistemas no lineales) de f con respecto al estado, x_k, y al ruido del proceso, w_k, respectivamente, y además Q_k representa la matriz de covarianza del ruido del proceso (dada por $Q_k = E[w_k {w_k}^T]$). Para el caso de sistemas lineales, A_k y W_k salen directamente de la definición de f.

En particular dada la naturaleza lineal del modelo del proceso, las ecuaciones descriptivas de esta etapa pueden ser expresadas como sigue:

$$x_k^- = Ax_{k-1} + Bu_{k-1} \tag{2.10}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2.11}$$

II. Etapa Correctiva.

En esta etapa las estimaciones anteriormente determinadas son corregidas mediante la información proveniente de las observaciones o mediciones realizadas al sistema. En este algoritmo el estado es corregido de forma proporcional a la diferencia entre la predicción y la observación, siendo a su vez ponderada esta diferencia por la "ganancia de Kalman" K_k, la cual corresponde a la ganancia óptima desde el punto de vista de la corrección del error en la estimación.

La ganancia de Kalman está dada por la siguiente expresión. Esta ganancia es escogida de forma óptima por el algoritmo de modo de obtener la mejor estimación de las variables de estado del sistema.

$$K_k = P_k^- {H_k}^T \left(H_k P_k^- {H_k}^T + V_k R_k {V_k}^T\right)^{-1} \tag{2.12}$$

Luego de forma análoga a la etapa predictiva, el estado y la covarianza del error de la estimación son calculados para esta etapa, pero esta vez como una corrección a los valores antes determinados, para ello utilizando las siguientes expresiones:

$$x_k = x_k^- + K_k\left(z_k - h(x_k^-, 0)\right) \tag{2.13}$$

$$P_k = (I - K_k H_k)P_k^- \tag{2.14}$$

Dada la linealidad del modelo observacional, tales ecuaciones quedan como:

$$x_k = x_k^- + K_k(z_k - Hx_k^-) \tag{2.15}$$

$$P_k = (I - K_k H)P_k^- \tag{2.16}$$

Y de la misma forma, la expresión de la ganancia de Kalman resulta simplificada gracias al supuesto de linealidad de los modelos, obteniendo como expresión resultante lo siguiente:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{2.17}$$

La relación existente entre la planta y el filtro de Kalman puede ser esquematizada en el diagrama de bloques que se presenta a continuación. En este se puede distinguir las etapas antes descritas del algoritmo para la estimación del estado.

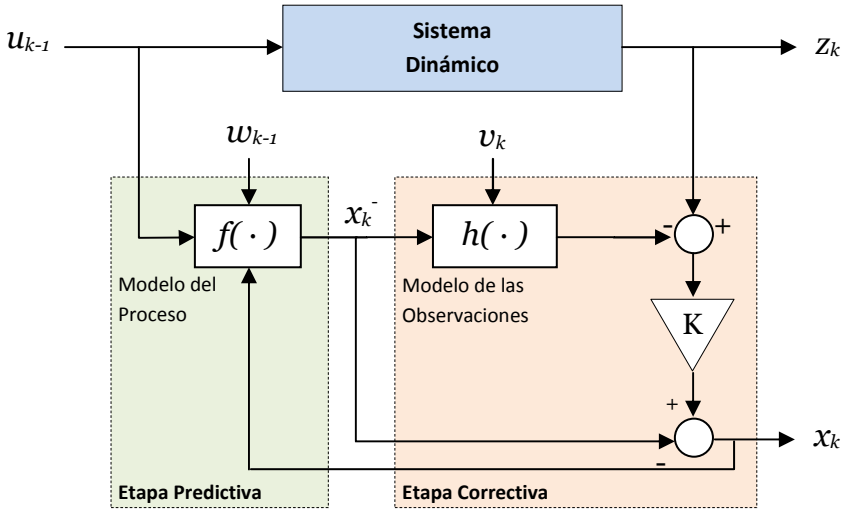


Figura 2.5: *Diagrama de bloques del filtro de Kalman.*

2.3.2. Filtro de Kalman Extendido (EKF).

Una extensión del filtro de Kalman, para poder ser utilizado en el caso de no cumplir con la condición de linealidad, corresponde al Filtro de Kalman Extendido (EKF). Esta implementación permite aplicar el algoritmo del filtro de Kalman en los casos en que ya sea el modelo del proceso f y/o el modelo observacional h son no lineales.

El filtro de Kalman extendido fundamenta su funcionamiento a través del uso de una linealización tanto del modelo del proceso y como del modelo de las observaciones, ambos en torno a la media y covarianzas actuales. De tal modo, las ecuaciones que describen el filtro permanecen inalteradas, sin embargo se debe tener presente que corresponden a funciones no lineales.

Etapa Predictiva
$x_k^- = f(x_{k-1}, u_{k-1}, 0)$ $P_k^- = A_k P_{k-1} {A_k}^T + W_k Q_{k-1} {W_k}^T$

Etapa Correctiva
$K_k = P_k^- {H_k}^T \left(H_k P_k^- {H_k}^T + V_k R_k {V_k}^T\right)^{-1}$ $x_k = x_k^- + \left(z_k - h(x_k^-, 0)\right)$ $P_k = (I - K_k H_k) P_k^-$

Figura 2.6: *Ecuaciones de las etapas predictiva y correctiva del filtro de Kalman.*

2.4. Muestreo Determinístico.

Un muestreo de tipo determinístico corresponde a una metodología para la toma de muestras de un conjunto continuo, siendo un caso particular de esto el de una función de densidad de probabilidad (*PDF*). De interés son las técnicas que permiten muestrear una *PDF* conservando algunos de los momentos de la función original, lo que posibilita llevar a cabo una mejor propagación de dichos momentos a través de los procesos a los cuales está asociada la función de densidad de probabilidad.

Una forma de realizar la extracción de un set de puntos $\{(\chi_j^a, \omega_j^a)\}$, llamados *Sigma Points* (Puntos Sigma), que conserven los 2 primeros momentos de la *PDF,* es el obtener una matriz S_a tal que corresponda a la raíz cuadrada de la matriz de covarianza de la PDF, esto es una matriz que cumpla con la relación $\Sigma_a = S_a {S_a}^T$, donde Σ_a es la matriz de covarianzas. Luego seleccionando el set de puntos $\{(\chi_j^a, \omega_j^a)\}$ como sigue, se asegura la conservación de los 2 primeros momentos de la PDF.

$$\chi_i^a = \begin{cases} \mu_a & , i = 0 \\ \mu_a + \sqrt{(D_a + \eta)} S_a^i & , i \epsilon [1, D_a] \\ \mu_a - \sqrt{(D_a + \eta)} S_a^{i - D_a} & , i \epsilon [D_a + 1, 2D_a] \end{cases} \tag{2.18}$$

$$\omega_i^a = \begin{cases} \omega_0 = \dfrac{\eta}{D_a + \eta} & i = 0 \\ \dfrac{1}{2(D_a + \eta)} & \sim \end{cases} \tag{2.19}$$

Donde S_a^i corresponde a la i-ésima columna de la matriz S_a, D_a es la dimensión del espacio en que se encuentra la PDF y η es un factor de escala.

Existen diferentes formas de obtener la matriz S_a, necesaria para llevar a cabo el muestreo, entre las cuales se encuentra la descomposición en valores singulares.

2.4.1. Descomposición en Valores Singulares (SVD).

Este procedimiento se aprovecha del hecho que para una matriz de covarianzas la descomposición en valores propios es igual a la descomposición en vectores propios (dado que esta es simétrica y semi-definida positiva). Luego la descomposición en valores propios de $\Sigma_a$está dada por $\Sigma_a = V_a \Lambda_a {V_a}^T$, donde Λ_a es una matriz diagonal con los valores propios

de Σ_a. Por lo tanto, siendo L_a una matriz diagonal con las raíces cuadradas de Λ_a, la matriz S_a estará dada por $S_a = V_a L_a$.

2.5. El Robot Nao.

El robot Nao corresponde a un dispositivo de entretenimiento desarrollado por la empresa francesa Alderaban Robotics y que también presenta ediciones y soporte enfocado al desarrollo académico. Este robot es la plataforma utilizada en la liga SPL de la Robocup desde el año 2008.

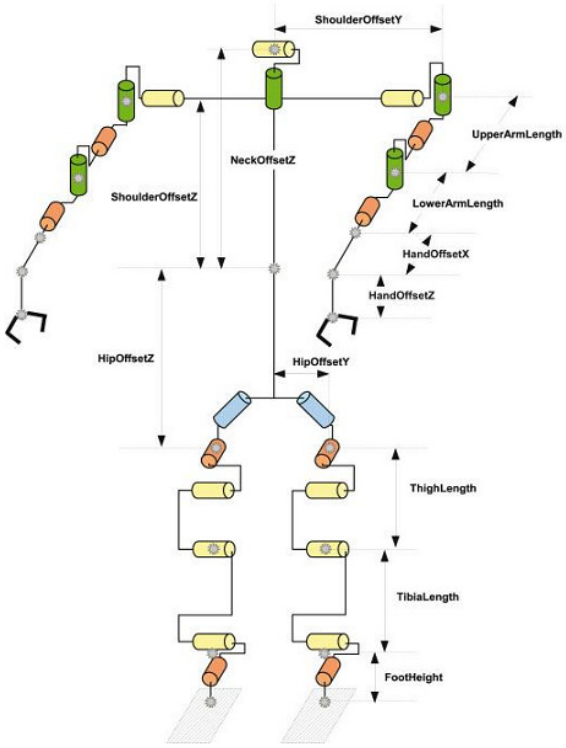
2.5.1. Características.

El robot Nao de Aldebaran Robotics tiene una altura de 58 cm aproximadamente y pesa cerca de unos 4.3 Kgs, entre las características de interés presentes en el Nao se puede destacar que este corresponde a un robot con 25 grados de libertad los cuales hay 2 presentes en la cabeza, 5 en cada brazo, 1 en la pelvis, 5 en cada pierna y 1 en cada mano. Este robot está equipado con un procesador AMD Geode con una frecuencia de 500MHz y 256MB de memoria tipo SDRAM, además de 2GB memoria tipo flash.

Por último, este robot cuenta con un set de sensores entre los que se pueden mencionar sensores de tacto, sonares, giróscopos, acelerómetros y sensores visuales, siendo estos últimos de particular interés para el trabajo desarrollado. Estos corresponden a 2 sensores visuales tipo CMOS, con resolución de 320x240 y situados en la cabeza del robot de acuerdo a como son presentados en la imagen de la Figura 2.8.

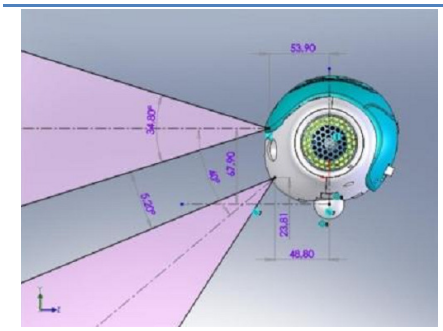
2.5.2. Dimensiones.

En la Figura 2.7 se exhiben las dimensiones del cuerpo del Robot Nao, mientras que en la Figura 2.8 se exhiben las dimensiones de la cabeza de este y la ubicación de los sensores visuales presentes en el robot.



Dimensión	Largo [mm]
Neck Offset Z	126,5
Shoulder Offset Y	98
Upper Arm Length	90
Lower Arm Length	50,65
Shoulder Offset Z	100
Hand Offset X	58
Hip Offset Z	85
Hip Offset Y	50
Thigh Length	100
Tibia Length	100
Foot Height	46
Hand Offset Z	15,9

Figura 2.7: *Dimensiones de cuerpo del Robot Nao.*



Dimensión	Largo [mm]
Camera Top Offset X	53,90
Camera Top Offset Z	67,90
Camera Bottom Offset X	48,80
Camera Bottom Offset Z	23,81

Figura 2.8: *Dimensiones de la cabeza del Robot Nao.*

3. Trabajo Desarrollado.

3.1. Modificaciones al Simulador de Alto Nivel.

La evaluación del funcionamiento de las rutinas implementadas en el módulo de estrategia tiene como condición inherente el que los datos con los que se trabaja en éste sean correctos. Luego, es necesario que la localización tanto del robot como de los objetos, dada por el módulo localización, sea correcta para descartar comportamientos inesperados debido a una mala localización, presentándose por lo tanto la necesidad de poder comprobar rápidamente si dichos datos son correctos o no.

Por esta razón se implementó dentro del simulador HL-Sim, una herramienta para la depuración de la localización de los objetos (la que es estimada mediante filtros denominados *trackers*) que permitiera observar en todo instante los datos provenientes del módulo de localización.

3.1.1. Herramienta de Depuración de Trackers.

Esta herramienta se implementó adicionando el menú "*Trackers Debug*" como parte de las opciones de depuración que ofrece el simulador de alto nivel para el módulo de modelamiento del mundo (o módulo de localización), como se puede ver en la Figura 3.1.

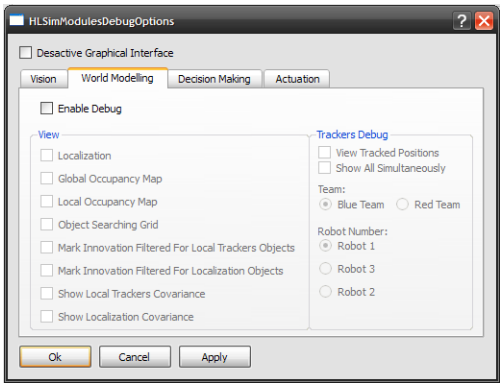
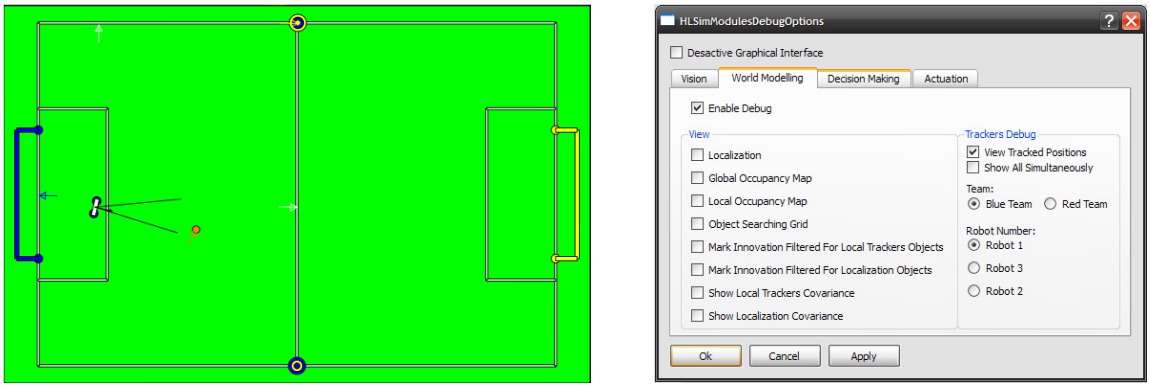


Figura 3.1: *Opciones de depuración del módulo de modelamiento del mundo.*

Estas opciones permiten representar gráficamente las posiciones de los objetos según el mapa local[4] asociado a uno de los robots presentes en la cancha. De tal manera, al seleccionar la opción *View Tracked Positions*, como se puede apreciar en la Figura 3.2, es posible ver la posición estimada de un objeto al hacer *click* sobre este y luego ocultar dicha información al hacer *click* nuevamente sobre el objeto.

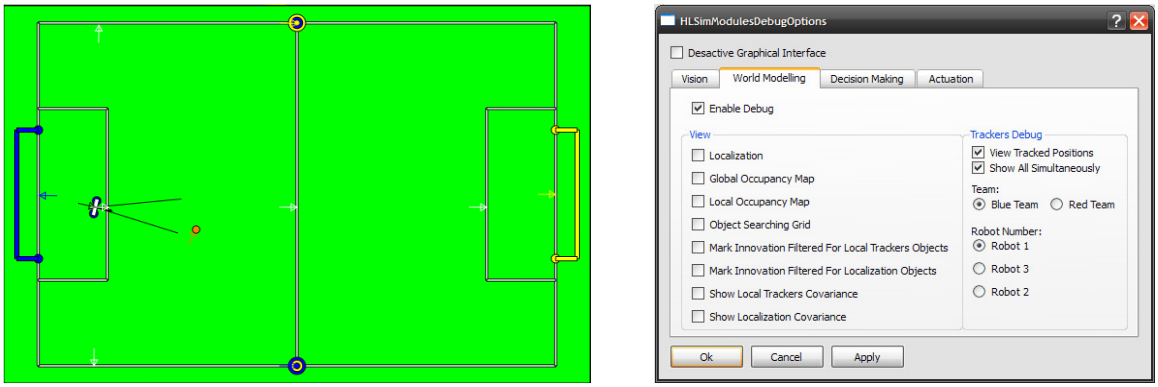


a) Posiciones estimadas de diversos objetos. *b) Configuración de las opciones de depuración.*

Figura 3.2: *Depuración de la posición algunos objetos presentes en la cancha.*

Conjuntamente existe la opción *Show All Simultaneously*, que permite observar de forma simultánea las posiciones de todos los objetos presentes en el mapa local del robot. Esta opción opera de manera similar a la anterior, mostrando las posiciones de todos los objetos al seleccionar el robot de interés. En la Figura 3.3 se puede observar un ejemplo del funcionamiento de esta opción.

[4] El mapa local de un robot corresponde a un mapa de las posiciones de los diversos objetos de interés y donde dichas posiciones se encuentran expresadas de manera relativa a la posición del robot.



a) Posiciones estimadas de todos los objetos. *b) Configuración de las opciones de depuración.*

Figura 3.3: *Depuración de la posición de todos los objetos presentes en la cancha.*

Con estas opciones para la depuración es posible evaluar de forma rápida y sencilla la calidad de la información que es generada por el módulo de localización a partir de los datos proveídos por el simulador y que es entregada al módulo de estrategia, siendo información relevante y en base a la cual se lleva a cabo la toma de decisiones.

3.2. Implementación de un Perceptor de Faros.

Como parte del trabajo realizado se llevó a cabo la implementación de un perceptor de faros (*beacons*), el que corresponde a un clasificador diseñado para de detectar la presencia de un objeto en particular, a partir de las regiones de color generados en las etapas previas del módulo de visión.

Un faro corresponde a un cilindro de 60cm altura y 20cm de diámetro, que presenta 3 secciones de color de 20cm de altura cada una, apiladas una sobre otra. Existen dos tipos de faro dentro del campo de juego, el faro azul-amarillo-azul y el amarillo-azul-amarillo, cada uno de los cuales tiene una posición definida dentro del campo de juego, como se puede apreciar en la Figura 3.4.

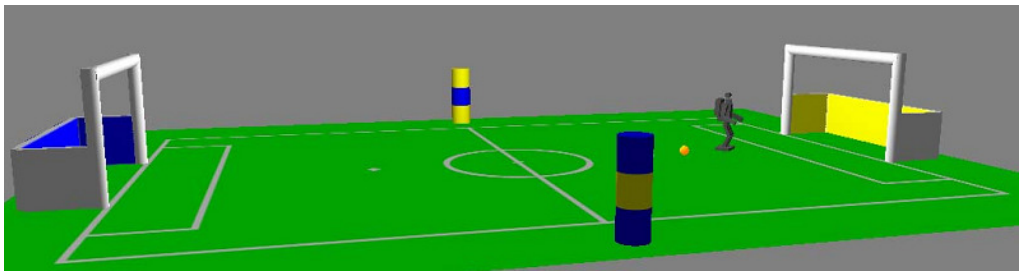


Figura 3.4: *Ubicación de los faros dentro del campo de juego.*

3.2.1. Algoritmo de Funcionamiento.

El perceptor está organizado en 3 etapas principales: primero es procesada una imagen ya segmentada, proveniente del módulo de visión (correspondiente a los datos de entrada al perceptor), mediante un set de reglas binarias se descartan las regiones que no presenten las características mínimas esperables para poder componer el objeto de interés, formando con estas los candidatos a faro. En la segunda etapa se evalúa y caracteriza cada candidato, obteniendo un puntaje que busca reflejar el ajuste de cada uno a lo que se espera de un faro. En la tercera etapa se escoge el mejor candidato de acuerdo al su puntaje, entregando la información de la detección a las siguientes etapas de procesamiento.

3.2.1.1. Reglas Binarias.

El set reglas binarias utilizadas para obtener los candidatos está enfocado en determinar las regiones que formarían parte de un faro y agruparlas como tal, para poder ser evaluados y caracterizados en la siguiente fase del perceptor.

El criterio principal para la formación de candidatos consiste en la búsqueda de regiones que presenten puntos característicos que se encuentren alineados y estén próximos dentro de la imagen tal como se muestra en la Figura 3.5.

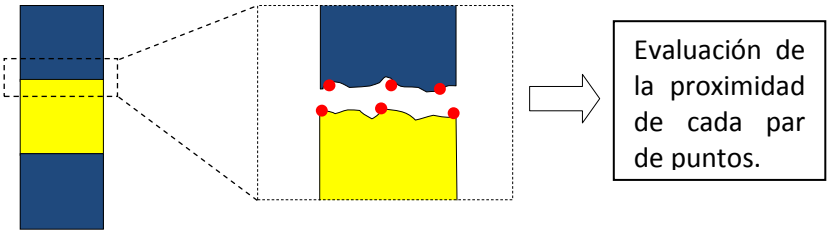


Figura 3.5: *Criterio de búsqueda de las junturas de los faros.*

Este criterio resulta más amplio que el uso de razones de aspecto o áreas de las regiones, los cuales se ven afectados por la distancia a la que se encuentra el objeto de interés y eventualmente por el hecho que el objeto podría estar en el borde de la imagen, alterando su relación de aspecto y tamaño de las áreas.

Utilizando este principio, el funcionamiento de la primera etapa del perceptor se enfoca en la búsqueda de las junturas presentes en un faro (junturas de una región azul y una amarilla).

El criterio sugerido por la Figura 3.5 se aplica intentando identificar las 2 junturas existentes en un faro, así una vez reconocida una de las junturas se intenta encontrar una región que satisfaga el criterio para encontrar una segunda juntura y en consecuencia un candidato para la siguiente fase.

El algoritmo completo de esta etapa se presenta en el diagrama de la Figura 3.6. Cabe mencionar que con este esquema el perceptor evalúa la presencia de cualquiera de los 2 posibles faros en una imagen, siendo capaz de encontrar ambos faros en caso de estar presentes.

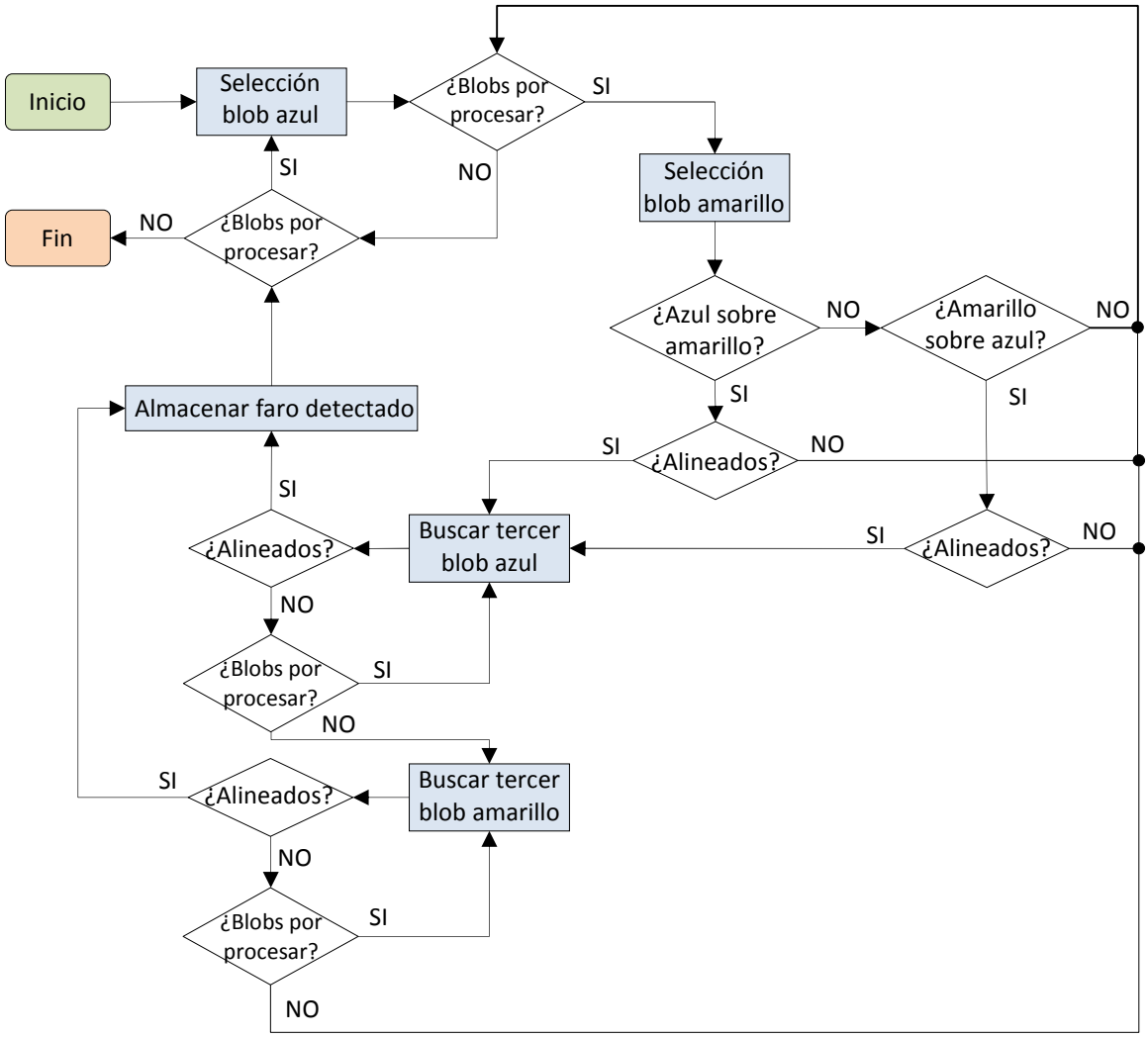


Figura 3.6: *Algoritmo de funcionamiento de la etapa de reglas binarias del perceptor de faros.*

Al observar el diagrama de la Figura 3.6 es posible apreciar la naturaleza iterativa del algoritmo implementado, lo que se debe a la necesidad de comprobar las diferentes combinaciones de regiones azules y amarillas en búsqueda de un candidato a faro.

El ajuste del criterio para determinar cuando un par de puntos característicos se encuentran alineados se realizó de manera empírica durante la etapa de entrenamiento del perceptor.

3.2.1.2. Evaluación de los Candidatos a Faro.

Dado que cada candidato a faro está constituido por regiones dispuestas de acuerdo a una lógica particular, disposición que no es muy probable encontrar de manera natural en el ambiente, en esta etapa se asume que todos los candidatos generados previamente son

válidos, es decir todos corresponden a un faro presente en la imagen, y en el caso de existir falsos positivos, estos serán descartados en la siguiente fase correspondiente a la selección de las mejores detecciones de acuerdo al puntaje determinado en esta etapa.

En esta etapa se lleva a cabo la extracción de características y datos relevantes de la detección tales como la pose del faro, distancia a la que se encuentra, varianza asociada a la detección, posición del objeto dentro de la imagen y puntaje de la detección.

3.2.1.3. Distancia y Pose Detectadas.

En la determinación de la distancia a la cual se encuentra el faro detectado se utilizó una proyección desde la imagen usando la distancia focal de la cámara, esto es:

$$distancia = \frac{tamaño\ real \cdot distancia\ focal}{tamaño\ en\ la\ imagen} \tag{3.1}$$

Para obtener la posición detectada del faro se utilizó la función `getXYProjection`, implementada en la librería UChile. Esta función realiza la proyección en el suelo de la posición del objeto, de acuerdo a la ubicación de los blobs dentro de la imagen y de acuerdo a la altura respecto al suelo que estos presentan en la realidad. Dicha función efectúa la proyección considerando las rotaciones y traslaciones de los sistemas de coordenadas correspondientes, de acuerdo al estado de los motores del robot.

3.2.1.4. Varianza de la Detección.

La estimación de la varianza asociada a la detección de un faro se realizó mediante una aproximación basada en resultados empíricos. Esto fue llevado a cabo mediante la toma de una serie de muestras distribuidas en diferentes puntos del campo de juego y con las cuales posteriormente se efectuó un ajuste polinomial, obteniendo de esta manera una función continua que permite estimar la varianza de cada una de las variables de la posición del faro.

- Metodología:

La recolección de datos consistió en la toma de 200 muestras (200 detecciones del perceptor) en 10 posiciones diferentes dentro del campo de juego, como se muestra en la Figura 3.7. Cada set de muestras fue generado con el robot de pie en la posición respectiva y

moviendo la cabeza de modo de posicionar el objeto de interés en diferentes partes dentro de la imagen.

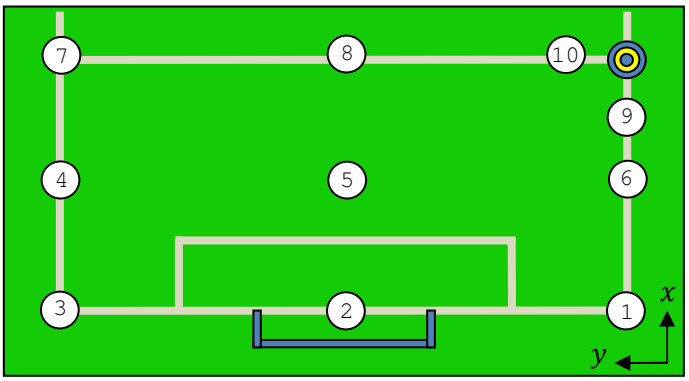


Figura 3.7: *Disposición de las ubicaciones para la recolección de datos.*

De tal modo se evaluó el perceptor posicionando al robot en 10 diferentes posiciones relativas al faro, las cuales son representativas de los posibles escenarios en los que este es utilizado. En la Tabla 3.1 se presentan las 10 posiciones donde se realizó el muestreo, expresado como la posición del faro de forma relativa al robot.

N° de Posición	**Ubicación del Faro** (x, y) $[cm]$	**N° de Posición**	**Ubicación de Faro** (x, y) $[cm]$
1	$(300,0)$	6	$(150,0)$
2	$(300,-200)$	7	$(0,-400)$
3	$(300,-400)$	8	$(0,-200)$
4	$(150,-400)$	9	$(65,0)$
5	$(150,-200)$	10	$(0,-60)$

Tabla 3.1: *Posiciones de muestreo relativas al robot.*

- <u>Aproximación de la Varianza de Detección:</u>

A partir de los datos obtenidos del muestreo se determinó la varianza asociada a cada set de datos, lo cual puede ser apreciado en la Tabla 3.2. Utilizando esta información se llevó a cabo un ajuste numérico para estimar los polinomios descriptivos de la varianza tanto en el eje de coordenadas X como en el eje Y.

Adicionalmente, y sólo por simplicidad, se supuso que la covarianza entre estas variables es nula.

Varianza Eje X $[cm^2]$	Varianza Eje Y $[cm^2]$
459,084	598,0258
382,037	467,124
739,815	703,739
626,640	491,055
244,662	396,962
387,923	229,190
272,244	367,571
147,433	319,756
55,666	107,644
136,908	61,345

Tabla 3.2: *Varianza asociada a la posición detectada de los faros.*

De tal manera, mediante el ajuste polinomial llevado a cabo se obtuvieron las siguientes expresiones para la varianza de las variables involucradas.

$$p(x,y) = -1{,}368 \cdot 10^{-3}x^2 - 6{,}944 \cdot 10^{-4}xy + 1.458x + 3{,}630 \cdot 10^{-3}y^2 + 9{,}049 \cdot 10^{-1}y + 116.120 \quad (3.2)$$

$$q(x,y) = 1{,}882 \cdot 10^{-3}x^2 + 2{,}074 \cdot 10^{-3}xy + 1.050x + 3{,}065 \cdot 10^{-4}y^2 - 7{,}972 \cdot 10^{-1}y + 50{,}557 \quad (3.3)$$

Es pertinente señalar que el dominio válido de los polinomios presentados en las ecuaciones 3.2 y 3.3 es $D = \{(x,y) \mid (x,y) \in (0{,}300) \times (-400{,}0)\}$, correspondiente al dominio en que se llevó a cabo la recolección de datos.

Este dominio no representa todas las posibles de ubicaciones relativas al robot de un faro, sin embargo dada la simetría del problema se utilizó este argumento para extender este dominio a todos los casos posibles. Adicionalmente y debido a que es imposible detectar un faro a una distancia menor a 50cm aproximadamente (ya que no es posible ver todas las regiones de color a tal distancia), se impuso el caso particular en que cualquier detección que presente una distancia relativa al robot menor de 50cm tendrá una alta varianza, buscando de este modo reflejar incerteza asociada a este tipo de detección.

Por lo tanto, las funciones para la estimación de la varianza en X e Y quedaron definidas como se muestra en las ecuaciones 3.4 y 3.5.

$$varX(x,y) = \begin{cases} 4000000 & si \quad \sqrt{x^2+y^2} < 50 \\ p(x,y) & si \quad (x \geq 0) \ \wedge \ (y \leq 0) \\ p(-x,y) & si \quad (x \leq 0) \ \wedge \ (y \leq 0) \\ p(x,-y) & si \quad (x \geq 0) \ \wedge \ (y \geq 0) \\ p(-x,-y) & si \quad (x \leq 0) \ \wedge \ (y \geq 0) \end{cases} \tag{3.4}$$

$$varY(x,y) = \begin{cases} 4000000 & si \quad \sqrt{x^2+y^2} < 50 \\ q(x,y) & si \quad (x \geq 0) \ \wedge \ (y \leq 0) \\ q(-x,y) & si \quad (x \leq 0) \ \wedge \ (y \leq 0) \\ q(x,-y) & si \quad (x \geq 0) \ \wedge \ (y \geq 0) \\ q(-x,-y) & si \quad (x \leq 0) \ \wedge \ (y \geq 0) \end{cases} \tag{3.5}$$

En dichas expresiones $p(x,y)$ y $q(x,y)$ corresponden a los polinomios presentados en las ecuaciones 3.2 y 3.3.

A continuación se presentan las superficies correspondientes a las funciones con las cuales se aproxima la varianza de las variables. Ese pertinente hacer notar que con motivo de visualizar correctamente las superficies asociadas a cada función se truncó a un menor valor el caso asociado a detecciones a una distancia inferior a 50cm.

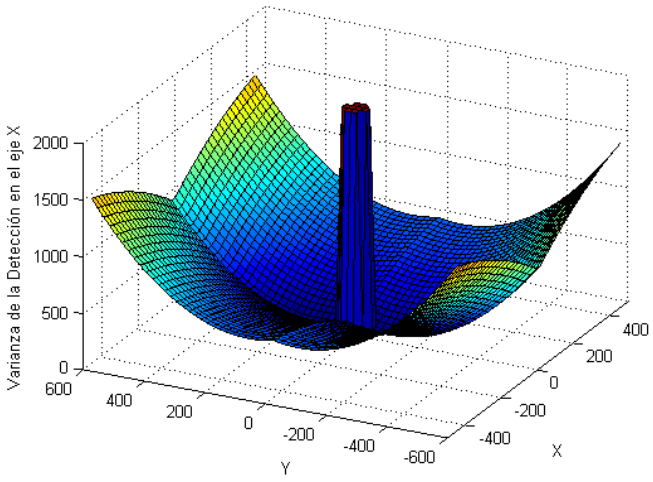


Figura 3.8: *Superficie de la aproximación de la varianza de x.*

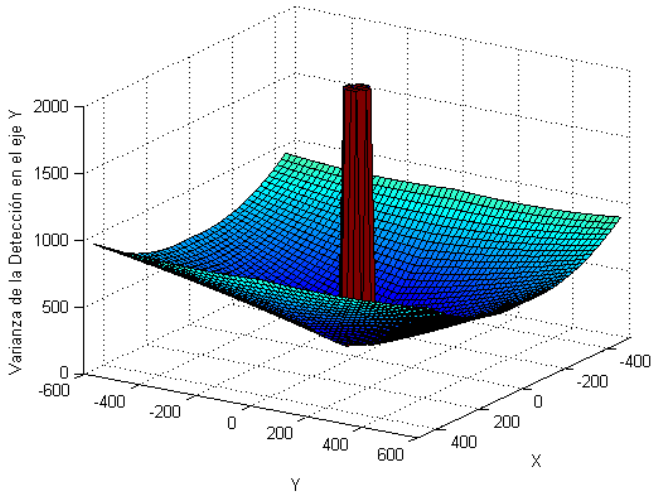


Figura 3.9: *Superficie de la aproximación de la varianza de y.*

3.2.1.5. Puntaje de la Detección.

El puntaje asociado a cada detección se compone de 3 términos provenientes de diferentes ámbitos del procesamiento de las regiones para la generación de los candidatos a faro. El cálculo del puntaje asociado a cada detección se implementó de acuerdo con la expresión:

$$puntaje = 0{,}4 \cdot PAJ + 0{,}4 \cdot PAL + 0{,}2 \cdot PRA \tag{3.6}$$

Donde *PAJ* es el Puntaje por la Alineación de las Junturas, PAL es el Puntaje por la Alineación de los Lados y *PRA* es el Puntaje por la Relación de Aspecto.

El primer término de la ecuación 3.6 refleja cuán apropiado es el candidato de acuerdo a la alienación de las junturas del faro. Este término es calculado con la expresión:

$$PAJ = 0{,}5 \cdot \left(\frac{1}{1 + a \cdot (\bar{x}_1 + \bar{y}_1)} + \frac{1}{1 + a \cdot (\bar{x}_2 + \bar{y}_2)} \right) \tag{3.7}$$

Donde $\bar{x}_i$ e $\bar{y}_i$ son el promedio de las diferencias en x e y entre los puntos característicos de la juntura i-ésima y a es un factor determinado empíricamente y utilizado para reducir la tasa de decrecimiento de este término.

El segundo término de la ecuación 3.6 refleja la desviación que presentan los lados del faro, para esto utilizando las posiciones de los puntos característicos laterales de las 3 regiones que forman cada faro. Este término es calculado con la siguiente expresión:

$$PAL = 0{,}5 \cdot \left(\frac{1}{1 + b \cdot \Delta l} + \frac{1}{1 + b \cdot \Delta r} \right) \tag{3.8}$$

Donde Δl y Δr corresponden a la máxima diferencia en x entre los puntos característicos del lado izquierdo y derecho del faro respectivamente. Además b, al igual que a, es un factor determinado de forma empírica para reducir la tasa de decrecimiento de este término.

Por último, el tercer término de la ecuación 3.6 intenta reflejar el ajuste del faro a la relación de aspecto que debería presentar cada una de las regiones que lo componen, buscando penalizar de esta manera los casos en que las regiones no son completamente observables o se ven afectadas por las condiciones de luz locales. Dicho término se calcula mediante la ecuación:

$$PRA = \frac{1}{5} \cdot (ra_1 + ra_2 + ra_3) \tag{3.9}$$

, donde ra_i es la relación de aspecto de la región i-ésimo del candidato y donde esta relación de aspecto corresponde al valor mínimo resultado de las 2 posibles formas de cálculo de la relación de aspecto, es decir $min\{alto/ancho\,, ancho/alto\}$.

3.3. Modificaciones a la Rutina de Seguimiento Visual.

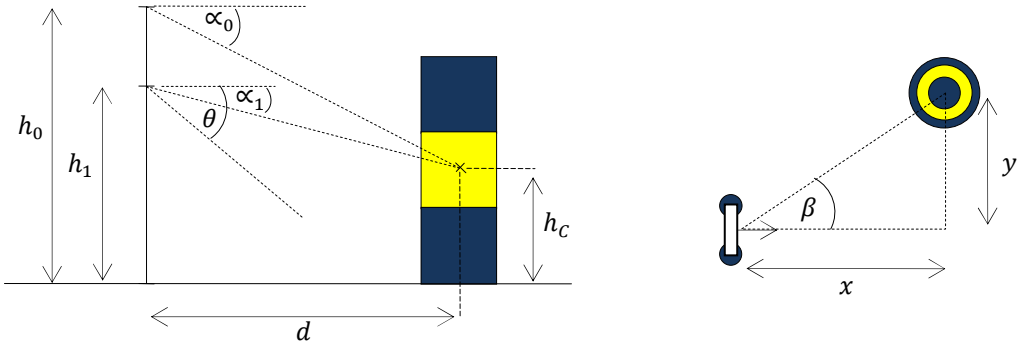
Dada la naturaleza del algoritmo de visión activa con el cual se trabajó, a menudo se desea enfocar objetos que no están siendo vistos en el momento pero de los cuales es conocida su posición relativa. Tal opción es denominada en este trabajo como “seguimiento por posición” y consiste en términos simples en intentar enfocar el objeto de interés basado en su posición estimada. Luego debido a la necesidad de contar con esta opción y dado que la actual rutina de seguimiento visual no presentaba dicha funcionalidad es que se debieron implementar los métodos que hicieran esto posible.

Junto con lo anterior, también se implementaron los métodos que hicieran posible el efectuar seguimiento a más de un objeto a la vez, lo cual implicó no sólo el desarrollo del algoritmo de seguimiento necesario si no que también fue necesaria la modificación de la rutina de control (denominada *MuxHead*) encargada de manejar las transiciones de estado de la cabeza, controlando el paso del estado de seguimiento a un determinado objeto al de búsqueda de dicho objeto en caso de no verlo cuando se espera que debería estar siendo visto.

3.3.1. Seguimiento de Posición.

El seguimiento por posición de un objeto corresponde al proceso tras el cual un objeto que no está siendo visto en el momento termina estando en el campo visual del robot, siendo la efectividad de este proceso susceptible a la incertidumbre existente en la localización del objeto que se intenta enfocar.

Para llevar a cabo este proceso, como primer punto es necesario determinar los ángulos en los que debe ser puesta la cabeza del robot para poder ver el objeto de interés. Esto es determinar el ángulo de inclinación vertical o ángulo de *tilt* y el ángulo con que hay que rotar la cabeza en el plano horizontal o ángulo de *pan*.



a) Determinación del ángulo de tilt. *b) Determinación del ángulo de pan.*

Figura 3.10: *Determinación de los ángulos para el seguimiento por posición.*

En la Figura 3.10 a) se muestra un diagrama en el cual se puede apreciar las distancias involucradas en la determinación del ángulo de *tilt* de la cabeza, correspondientes a los ángulos α_0 y α_1. Es oportuno señalar que la existencia de 2 ángulos obedece únicamente a la existencia de 2 cámaras en el robot, las que se encuentran en diferentes posiciones para facilitar la observación de distintos puntos en el ambiente.

En estricto rigor existe una diferencia en posición de las cámaras con respecto al eje central del robot, lo que lleva a que la distancia que se debe considerar para el cálculo del *tilt* tenga presente la influencia de dicha diferencia. Sin embargo, esta puede ser despreciada debido a que es mucho menor que el rango en que varía la distancia d. Adicionalmente, existe una diferencia en el ángulo de inclinación que presenta cada cámara con respecto a la línea del horizonte, encontrándose la cámara superior sin inclinación vertical (ángulo de tilt igual a cero) y teniendo la cámara inferior una inclinación $\theta = 40°$. Vale la pena hacer notar que a pesar que este valor para la inclinación de la cámara inferior es nominal, en la práctica se comprueba que una inclinación $\theta \sim 31{,}3°$ presenta un mejor desempeño en la rutina de seguimiento.

De tal manera, el cálculo del *tilt* de la cabeza para cada una de las cámaras está dado por las expresiones:

$$\alpha_0 = \tan^{-1}\left(\frac{h_0 - h_C}{d}\right) \tag{3.10}$$

$$\alpha_1 = \tan^{-1}\left(\frac{h_1 - h_C}{d}\right) - \theta \tag{3.11}$$

En las ecuaciones anteriores h_i es a la altura a la cual se encuentra la cámara que se está considerando para el cálculo, h_C es a la altura a la que se encuentra el centro del objeto, d es la distancia en el plano horizontal que separa al objeto del robot y θ corresponde a la inclinación de la cámara inferior respecto al horizonte (como se señaló anteriormente).

De forma similar, en la Figura 3.10 b) se presenta un diagrama para el cálculo del ángulo de *pan*, el que resulta dado por la expresión:

$$\beta = \tan^{-1}\left(\frac{y}{x}\right) \tag{3.12}$$

Donde x e y corresponden a las coordenadas del objeto de interés, en el mapa local del robot.

Por lo tanto el seguimiento de la posición del objeto se reduce a mover la cabeza del robot (con el objetivo de mover los sensores visuales presentes en ésta) a la posición angular determinada por las expresiones 3.10, 3.11 y 3.12, teniendo la precaución de utilizar el ángulo de *tilt* asociado a la cámara con la cual se espera ver el objeto.

3.3.1.1. Seguimiento con Objeto Extra.

A pesar que el objetivo del seguimiento por posición es llegar a observar un determinado objeto, es claro que en el trayecto de la cabeza hasta su posición objetivo es probable que sean vistos otros elementos del ambiente, los cuales podrían resultar de interés. Para aprovechar este hecho se implementó adicionalmente la capacidad de hacer seguimiento por posición a un objeto, considerando la posibilidad de ver objetos extra.

Para llevar a cabo esto, una vez determinada la posición objetivo de la cabeza se evalúa que objetos es factible observar en el trayecto, basándose en las posiciones de los objetos en el mapa local del robot.

El criterio utilizado en la selección de los eventuales objetos extra, es que la posición de estos (en el espacio de ángulos de la cabeza) no debe estar fuera del área rectangular formada entre la posición actual de la cabeza y la posición objetivo. Adicionando además una zona de holgura extra, orientada a considerar los objetos que se encuentra levemente fuera del área rectangular antes mencionada. En la Figura 3.11 se presenta un diagrama de la disposición de áreas consideradas para el criterio de selección de objetos.

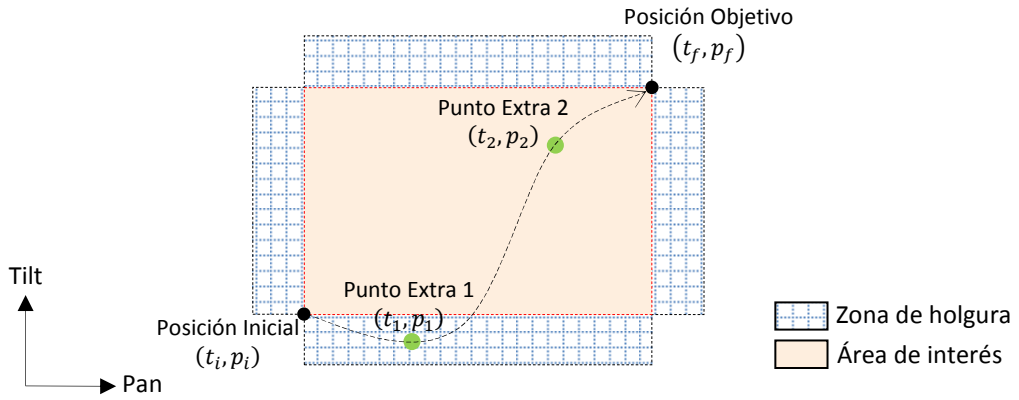


Figura 3.11: *Criterio de selección de objetos adicionales.*

En consecuencia, al existir un objeto que se satisfaga el criterio de selección se genera una trayectoria para la cabeza, para lo cual se utiliza como único criterio la menor distancia entre puntos, lo que asegura que la posición objetivo final del seguimiento sea la última en ser visitada y por consiguiente permite visitar las restantes posiciones determinadas.

3.3.2. Seguimiento de Distribución de Probabilidad.

Como se vio en la sección anterior, el seguimiento por posición es realizado utilizando las posiciones relativas de los objetos presentes en el mapa local del robot. Esto implica que todo el proceso está sujeto a la incertidumbre proveniente de las posiciones de los objetos, las que son estimadas utilizando filtros de Kalman.

Por esta razón en los casos en que dichas posiciones presentan alta incerteza existe una gran probabilidad de que el seguimiento por posición resulte infructuoso, esto es mover la cabeza hacia la posición determinada y que sin embargo no sea posible ver el objeto en cuestión debido a que su posición real no coincidía con la estimada.

Para abordar estos casos se implementó el seguimiento de la función de distribución de probabilidad de la ubicación del objeto, proceso que consiste en hacer seguimiento visual a una representación discreta de dicha función y que se compone de 3 etapas consecutivas: obtención de la representación discreta, transformación al espacio de los ángulos de la cabeza y planificación del recorrido del set de puntos.

3.3.2.1. Representación Discreta y Cálculo de Ángulos de la Cabeza.

Para la obtención de la representación discreta de la función de distribución de probabilidad se utiliza muestreo determinístico mediante descomposición en valores singulares o SVD (*Singular Value Decomposition*).

Ya que para el cálculo de los ángulos de cabeza es necesaria únicamente la posición del objeto y no su orientación, la descomposición es llevada a cabo en la función de distribución de probabilidad que representa la posición del objeto sin su orientación, obteniendo de esta manera un set de Puntos Sigma correspondientes a las posibles posiciones en el plano horizontal.

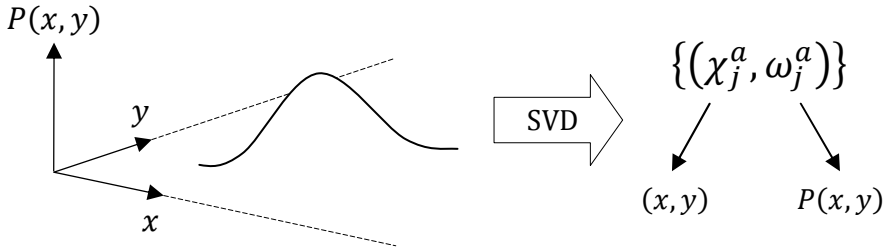


Figura 3.12: *Muestreo determinístico de la ubicación del objeto de interés.*

Posteriormente estos puntos en el espacio (x, y) son transformados en un set de coordenadas en el espacio de los ángulos de la cabeza $(tilt, pan)$ utilizando las ecuaciones 3.10, 3.11 y 3.12. Cada uno de estos puntos representa una posible posición para la cabeza donde existe una probabilidad asociada de encontrar el objeto de interés.

3.3.2.2. Planificación de la Trayectoria de la Cabeza.

A diferencia del seguimiento por posición, el seguimiento de la distribución de probabilidad de la posición de un objeto presenta múltiples posiciones objetivo para la cabeza. Por consiguiente es necesario llevar a cabo una planificación de la forma de recorrer dichos puntos, utilizando algún criterio para la selección de la mejor ruta a realizar.

En este caso particular la planificación de la trayectoria, con la cual son recorridos los puntos de la función de distribución de probabilidad del objeto, es realizada utilizando el criterio de la distancia mínima entre puntos. Por lo tanto, el primer punto de la trayectoria es el punto de la función de distribución de probabilidad que se encuentra más cercano a la posición actual de la cabeza. Luego los restantes puntos son escogidos

buscando siempre el punto más cercano al último considerado para la trayectoria de la cabeza.

Dado que el objetivo final de la rutina de seguimiento es llegar a observar el objeto de interés, no es estrictamente necesario recorrer todos los puntos existentes en la trayectoria si no que sólo hasta que el objeto de interés sea visto, caso en el cual se pasa a la función de seguimiento visual del objeto y dando por terminado el recorrido por la función de distribución de probabilidad.

3.3.3. Seguimiento de Coordenadas.

El seguimiento de coordenadas corresponde simplemente a desplazar la cabeza a una posición tal que sea factible observar a un par de coordenadas (x, y), dadas como parámetro al momento de la ejecución de la rutina. Esta funcionalidad hace posible el observar un grupo de objetos, para lo cual sólo basta con determinar un punto que permita visualizar todos los elementos del conjunto de interés.

Esta capacidad fue implementada de manera particular para ser utilizada junto con la rutina de visión activa existente, de modo de poder realizar seguimiento a los esquemas de observación determinados por esta y que serán detallados con posterioridad.

Para llevar a cabo esto basta con utilizar las expresiones presentadas con anterioridad en la Sección 3.3.1, ecuaciones 3.10, 3.11 y 3.12, con las cuales es posible efectuar la transformación del espacio de coordenadas (x, y) al espacio de ángulos de la cabeza $(tilt, pan)$.

Dado que el par de coordenadas (x, y) es entregado a la rutina de seguimiento como parámetro, para llevar a cabo el seguimiento basta aplicar las ecuaciones antes mencionadas y utilizando siempre la altura del centro del objeto como $h_C = 0$, valor que será justificado con posterioridad en la Sección 3.4.1.5.

3.3.4. Modificaciones a la Rutina de Control de Cabeza.

Las modificaciones que fue necesario efectuar a la rutina de control de cabeza estuvieron enfocadas principalmente a controlar los cambios de estado cuando se lleva a cabo el seguimiento de un conjunto de objetos, debido fundamentalmente a que cuando se

está observando un grupo de objetos, puede suceder que a pesar de estar viendo todos los objetos luego de un instante se deje de ver uno o varios de estos, pero sin llegar a dejar de verlos todos. Esto implica que las transiciones de estado no sean únicamente entre seguimiento y búsqueda, sino que se deba considerar la existencia de estados intermedios en los cuales se pase, por ejemplo, de observar a un conjunto completo a observar sólo una parte del conjunto.

En la Figura 3.13 se presenta un diagrama con la máquina de estados que realiza el control de la cabeza[5]. En este diagrama se puede apreciar que las transiciones de estado pueden obedecer a diferentes motivos, tales como cambios en los parámetros que están siendo entregados a la rutina de seguimiento y que indican los objetos de interés, o también cambios en las condiciones en que se está realizando el seguimiento de un determinado objeto o grupo de objetos, por ejemplo cuando uno o todos dejan de ser vistos, gatillando la transición al estado de búsqueda o bien al estado de seguimiento de los objetos restantes.

[5] Este corresponde a un diagrama de la máquina de estados que presenta las condiciones para las transiciones de estado de manera simplificada para facilitar su comprensión dado que el detallar todas las condiciones que influyen en cada una de estas transiciones complejizaría notablemente el diagrama, dificultando su lectura.

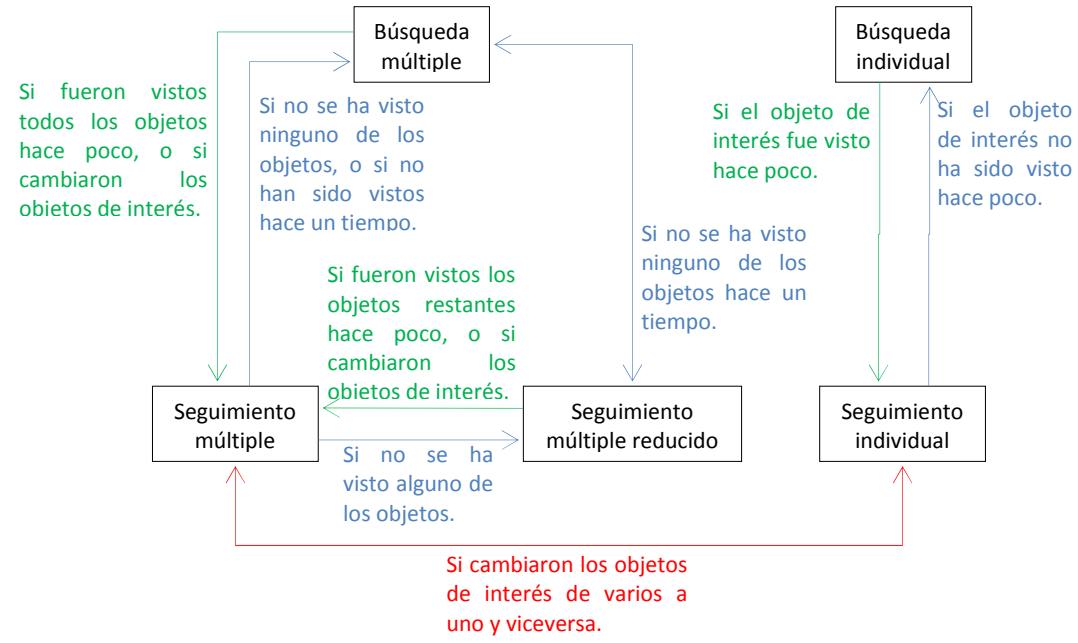


Figura 3.13: *Diagrama de la máquina de estados de la cabeza.*

Cabe mencionar que para las transiciones entre el estado de "*seguimiento múltiple*"[6] y "*seguimiento múltiple reducido*"[7] se implementó la opción que permite escoger (al momento de configurar los parámetros del seguimiento) que en el caso de estar haciendo seguimiento múltiple se deja de ver un objeto, se pueda pasar directamente al estado de búsqueda múltiple o bien se pueda seguir haciendo seguimiento de los restantes objetos que si están siendo vistos.

[6] El seguimiento múltiple es la funcionalidad que permite realizar seguimiento de un conjunto de objetos.
[7] El seguimiento múltiple reducido es el estado al cual se puede acceder cuando al realizar seguimiento a un conjunto de objetos, uno o más de estos dejan de ser vistos, luego el seguimiento múltiple reducido es el estado en que se mantiene el seguimiento a los objetos que aún están siendo vistos, en vez de pasar al estado de búsqueda de los objetos que dejaron de ser vistos.

3.4. Modificaciones a la Rutina de Visión Activa.

Debido a que las modificaciones realizadas en la rutina de visión activa no alteraron mayormente el algoritmo básico de funcionamiento, presentado en las secciones 2.1.1, 2.1.2 y 2.1.3, a continuación se expondrán únicamente los cambios realizados asumiendo un conocimiento previo del algoritmo.

3.4.1. Ampliación del Espacio de Acciones Sensoriales.

La ampliación del espacio de acciones sensoriales posibles, estuvo enfocada en la implementación de los criterios de análisis, que permitieran seleccionar como objetivo para la cabeza una determinada posición angular en la que fuera posible observar 2 o más objetos de manera simultánea.

Para afrontar el problema se efectuó un planteamiento basado en la generación de una matriz de adyacencia y el establecimiento algunas restricciones en el funcionamiento.

3.4.1.1. Planteamiento del Problema.

El planteamiento del problema se realizó en 3 etapas consecutivas, luego de las cuales es posible obtener como resultado diferentes grupos de objetos, que pueden ser vistos simultáneamente y con los cuales posteriormente se efectúa la etapa correctiva del filtro de Kalman presente en el algoritmo de visión activa (como se señala en la Sección 2.1.3). En la Figura 3.14 se presenta un diagrama del proceso de generación de grupos de posibles objetos a observar.

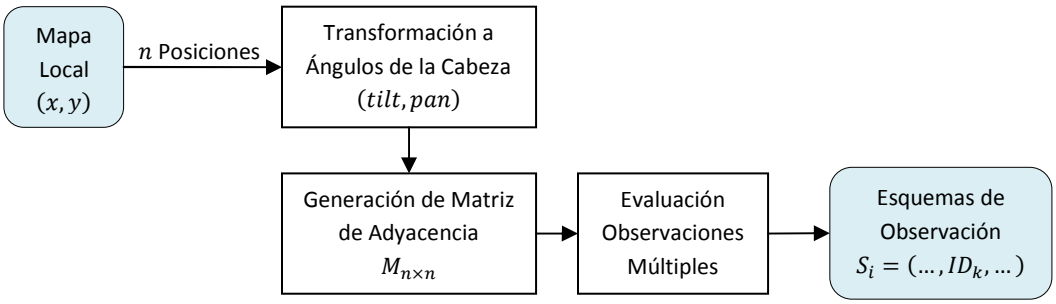


***Figura 3.14**: Generación de esquemas de observación.*

De acuerdo a esto, el funcionamiento básico del algoritmo de visión activa, presentado en las secciones anteriores, es modificado principalmente en la etapa de

simulación de las observaciones para la etapa correctiva del filtro de Kalman (y por supuesto en la implementación de dicho filtro de Kalman para operar con múltiples observaciones simultáneas). Por lo tanto al ser integradas las modificaciones planteadas, el esquema presentado en la Figura 2.1 queda como se puede apreciar en la Figura 3.15.

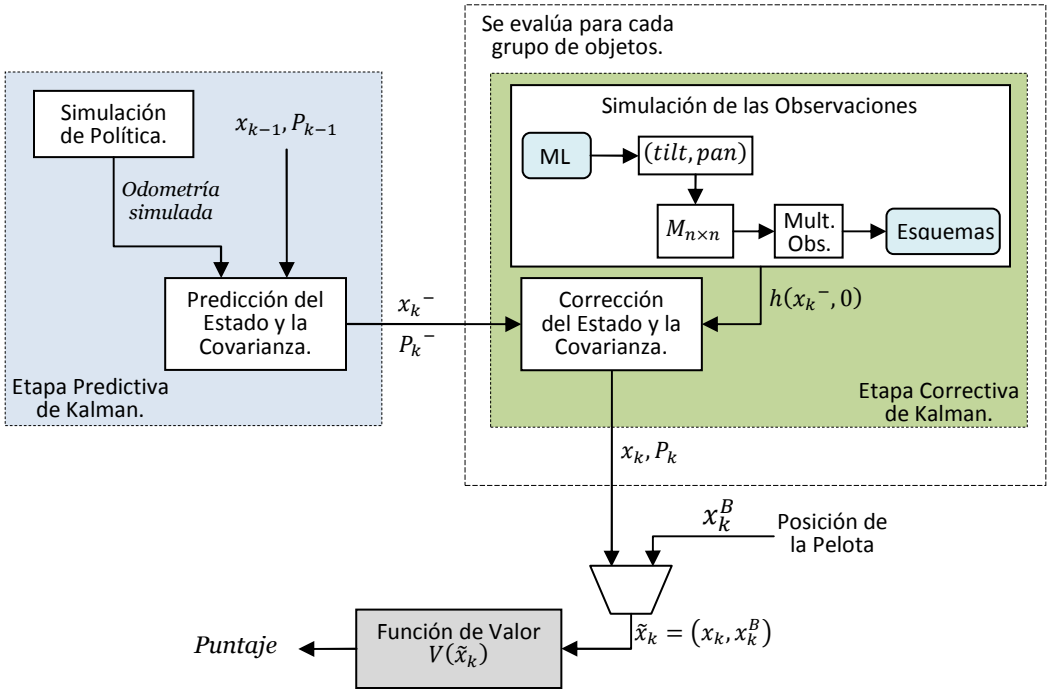


Figura 3.15: *Esquema de funcionamiento del algoritmo post-modificaciones*

De acuerdo a este esquema, las observaciones simuladas pasan de ser un vector, conteniendo la información del i-ésimo objeto, a ser un vector con la información de de los m objetos presentes en el j-ésimo esquema de observación, determinados en la etapa de evaluación de múltiples observaciones, como se presenta en la Figura 3.16.

$$h(x_k^-, 0)_i = \begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix} \quad \Longrightarrow \quad h(x_k^-, 0)_j = \begin{pmatrix} x_1^j \\ y_1^j \\ \theta_1^j \\ \vdots \\ \vdots \\ x_m^j \\ y_m^j \\ \theta_m^j \end{pmatrix}$$

Figura 3.16: *Vector de observaciones esperadas antes y después de modificar el algoritmo.*

3.4.1.2. Transformación de Posiciones a Ángulos.

La transformación de posiciones en ángulos hace referencia a la determinación de la posición angular de la cabeza del robot para ser capaz de ver un determinado objeto del mapa local de éste.

En este punto se llevan a cabo los mismos cálculos presentado en las ecuaciones 3.10, 3.11 y 3.12, generando así un set de n vectores (un vector por cada uno de los n objetos presentes en el mapa local y que son considerados en el algoritmo), cada uno de los cuales contiene el *tilt* y *pan* asociados a la posición del objeto, así como el índice de la cámara a la cual está asociado dicho ángulo de *tilt*.

3.4.1.3. Generación de Matriz de Adyacencia.

Utilizando la información obtenida en la etapa anterior, es generada una matriz de adyacencia. Para realizar esto se utilizan las posiciones de los objetos en el espacio $(tilt, pan)$ y el tamaño del campo visual del robot. Luego utilizando una ventana de las dimensiones del campo de visión del robot, se itera sobre todos los objetos de interés, observando cuales es posible situarlos al mismo tiempo dentro de dicha ventana.

Este constituye una primera estimación de la factibilidad de observaciones múltiples simultáneas, información que es utilizada en la siguiente etapa para realizar una reevaluación en la que es considerada la posibilidad de ver simultáneamente más que un par de objetos.

3.4.1.4. Evaluación de Observaciones Múltiples.

Dado que al generar la matriz de adyacencia sólo se considera la posibilidad de observar pares de objetos, es necesario efectuar una segunda evaluación en la que se considere eventuales grupos de más de 2 objetos. Para esto el algoritmo desarrollado basa su funcionamiento en el uso del cuadrado de la matriz de adyacencia, el cual en su diagonal presenta el número de objetos que pueden ser vistos con el objeto en cuestión. Además el algoritmo desarrollado para la generación de los esquemas de observación utiliza el principio de transitividad para el análisis de observaciones múltiples.

El funcionamiento del algoritmo puede ser descrito en 2 pasos, los cuales son llevados a cabo secuencialmente para cada uno de los objetos de interés.

I. Evaluación del número de objetos adyacentes.

El primer paso en el algoritmo es evaluar la cantidad de objetos visualmente adyacentes al objeto que está siendo analizado. Esto es el número de elementos que pueden ser vistos junto con dicho objeto, para lo cual se utiliza el cuadrado de la matriz de adyacencia como ya se mencionó.

En esta etapa se comprueba si el número de elementos adyacentes al objeto es 0 y en tal caso se genera un esquema de observación que contiene sólo a éste, el que es marcado para no volver a ser procesado. En caso que el número de elementos adyacentes al objeto sea mayor o igual que 1 se pasa a la etapa de evaluación de la adyacencia única entre objetos.

II. Evaluación de la adyacencia única entre objetos.

En esta fase se evalúa si los elementos adyacentes al objeto bajo análisis son únicamente adyacentes a este y entre ellos, o si por el contrario existen adyacencias con terceros objetos, es decir elementos que son adyacentes a uno de los objetos pero no a los demás.

La Figura 3.17 representa gráficamente este tipo de relación entre objetos. En dicha figura los elementos 1, 2 y 3 son adyacentes, así como también 2 y 4. Sin embargo en esta representación sólo los elementos 1 y 3 son únicamente adyacentes entre ellos dado que el elemento 2 es adicionalmente adyacente al elemento 4, el que no es adyacente a 1 ni 3.

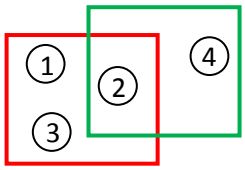


Figura 3.17: *Relación de adyacencia entre objetos.*

En el primer caso (todos los objetos únicamente adyacentes) se genera un esquema de observación que contiene a todos los elementos adyacentes al objeto bajo análisis y a

este mismo, los cuales son marcados como ya procesados. En el segundo caso (sólo algunos objetos únicamente adyacentes) se genera el mismo esquema de observación, pero sólo son marcados como utilizados los objetos que presentan adyacencia única al objeto de interés, de esta forma los elementos que presenten más objetos adyacentes siguen siendo válidos para una siguiente evaluación en la cual sean considerados a sus restantes pares, siguiendo la misma lógica descrita en los puntos *I* y *II*.

De tal manera, el algoritmo con el cual se realiza la segunda evaluación de las posibles múltiples, orientada a considerar los casos en que observaciones de más de 2 objetos son posibles, puede ser esquematizado como se presenta en la Figura 3.18.

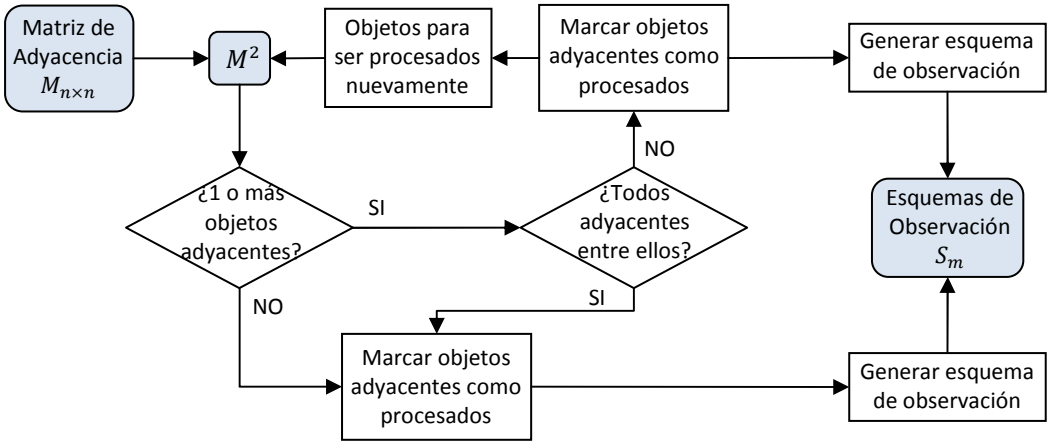


Figura 3.18: *Algoritmo de evaluación de múltiples observaciones.*

3.4.1.5. Cálculo de la Posición para el Seguimiento Visual.

La última etapa en la ejecución del algoritmo de visión activa[8], luego de la generación de los esquemas de observación y de la asignación del puntaje asociado a este, corresponde al cálculo de la posición que será utilizada para llevar a cabo el seguimiento visual. Cabe mencionar que esta etapa solamente se lleva a cabo en los esquemas que consideran múltiples objetos, puesto que para el caso de un único objeto dicha posición está dada por el mapa local del robot.

[8] En la Figura 3.21, es posible observar un diagrama en el cual se presenta completo el algoritmo de visión activa modificado, donde se aprecia cada una de las etapas constitutivas de este, entre ellas el cálculo de las posición para el seguimiento visual.

Para calcular dicha posición, como primer paso se determina el punto medio entre los objetos presentes en el esquema que está siendo procesado, para esto simplemente seleccionando el punto medio entre los ángulos de pan mínimo y máximo y el punto medio entre los ángulos de tilt mínimo y máximo, como se representa en la Figura 3.19.

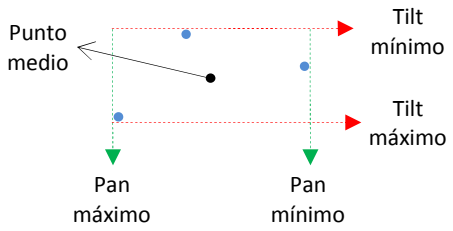


Figura 3.19: *Determinación del punto medio de un esquema de observación.*

Una vez determinado dicho punto en el espacio de los ángulos de la cabeza del robot $(tilt, pan)$ se hace la transformación al espacio de las coordenadas relativas al robot (x, y), para lo cual se utilizan las ecuaciones 3.10, 3.11 y 3.12. Con estas expresiones y dado que d corresponde a la distancia a la cual se encuentra el objeto de interés, es decir $d = \sqrt{x^2 + y^2}$, se establece un sistema de ecuaciones, desde el cual es posible obtener expresiones para x e y como se presenta a continuación.

$$x = \frac{h_0 - h_C}{|\tan(\alpha_0)|} \cdot \frac{1}{\sqrt{1 + \tan(\beta)^2}} \tag{3.13}$$

$$y = \frac{h_0 - h_C}{|\tan(\alpha_0)|} \cdot \frac{\tan(\beta)}{\sqrt{1 + \tan(\beta)^2}} \tag{3.14}$$

Sin embargo, en estas expresiones se utiliza la altura del centro del objeto al cual se le está haciendo seguimiento visual, h_C. Dicha altura no puede ser definida fácilmente en el caso de un grupo de objetos. Por esta razón, para efectuar tal transformación se asumió $h_C = 0$, con lo cual se obtiene un par de coordenadas ficticias (x', y'), correspondientes a la posición en la que una recta trazada desde el centro de la cámara y con la orientación dada por los ángulos *tilt* y *pan*, intersecta el piso, como se muestra en la Figura 3.20.

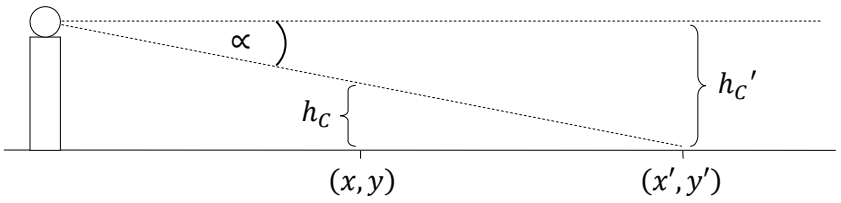


Figura 3.20: *Correspondencia entre coordenadas (x,y) y el par ficticio de coordenadas (x',y').*

De esta manera se obtienen las expresiones utilizadas en la transformación de espacio y con las cuales se determina el punto utilizado para el seguimiento visual del esquema de observación.

$$x' = \frac{h_0}{|\tan(\alpha_0)|} \cdot \frac{1}{\sqrt{1 + \tan(\beta)^2}} \tag{3.15}$$

$$y' = \frac{h_0}{|\tan(\alpha_0)|} \cdot \frac{\tan(\beta)}{\sqrt{1 + \tan(\beta)^2}} \tag{3.16}$$

Siendo las ecuaciones 3.15 y 3.16 válidas para los casos en que se utiliza la cámara superior del robot. En los casos en que se utiliza la cámara inferior dichas expresiones resultan como sigue.

$$x' = \frac{h_0}{|\tan(\alpha_0) + \theta|} \cdot \frac{1}{\sqrt{1 + \tan(\beta)^2}} \tag{3.17}$$

$$y' = \frac{h_0}{|\tan(\alpha_0) + \theta|} \cdot \frac{\tan(\beta)}{\sqrt{1 + \tan(\beta)^2}} \tag{3.18}$$

Donde θ corresponde a un *offset* dado por la inclinación de la cámara inferior, como se mencionó anteriormente en la Sección 3.3.1.

3.4.1.6. Cálculo del Puntaje del Grupo de Posibles Observaciones.

Puesto que en la práctica se pudo ver que el uso únicamente de la función de valor, como criterio para la selección de los objetos/grupos de objetos a observar, no siempre privilegiaba a los elementos con una importancia relativa mayor (como la pelota por ejemplo) es que se implementó una manera para reevaluar los puntajes asociados a cada grupo, en cual se establecieron prioridades para los objetos de mayor importancia relativa.

El objetivo principal tras esta implementación es basarse en las confidencias de cada uno de los objetos de interés, que han sido vistos y de acuerdo a esto reevaluar el puntaje,

de modo de privilegiar la visión de algunos objetos y así mantener una mejor estimación de su posición. De tal manera, la función para el cálculo del nuevo puntaje grupal grupo quedó implementada como sigue.

$$Puntaje = \left(\frac{Puntaje\ inicial + \sum_{i=0}^{N} a_i \cdot (1 - C_i)}{2} \right) \cdot 0{,}95 + 0{,}05 \tag{3.19}$$

En la expresión 3.19 "*Puntaje inicial*" representa el puntaje asociado al grupo antes de efectuar el cálculo del nuevo puntaje, el factor C_i corresponde a la confidencia del i-ésimo objeto perteneciente al esquema de observación (grupo de objetos), a_i corresponde a un ponderador particular a cada objeto, en el cual se expresa la prioridad existente para cada objeto. El valor de dichos ponderadores se presenta en la Tabla 3.3.

Objeto	Ponderador
Pelota	0,35
Arco Propio	0,05
Arco Enemigo	0,25
Faro	0,175

Tabla 3.3: *Ponderadores utilizados en la función de reevaluación de puntaje.*

La adición de un término constante al final de la expresión 3.19 obedece únicamente a la intención de evitar la generación de puntajes nulos debido a que esto genera problemas en la rutina de control de la cabeza.

Por último, se debe señalar que existe un caso particular bajo el cual la función para el cálculo puntaje de grupo es ignorada y dicho valor es fijado automáticamente en 1. Este caso corresponde al escenario en que la pelota presenta una confidencia menor a 0.25 y con este se busca privilegiar el mantener una correcta estimación de la posición de dicho objeto. De tal manera, al existir un esquema de observación que presente a la pelota entre sus elementos, este será favorecido inmediatamente, llevando el puntaje de grupo a 1 y asegurando de esta manera la visión de la pelota.

3.4.1.7. Limitaciones de la Formulación Utilizada.

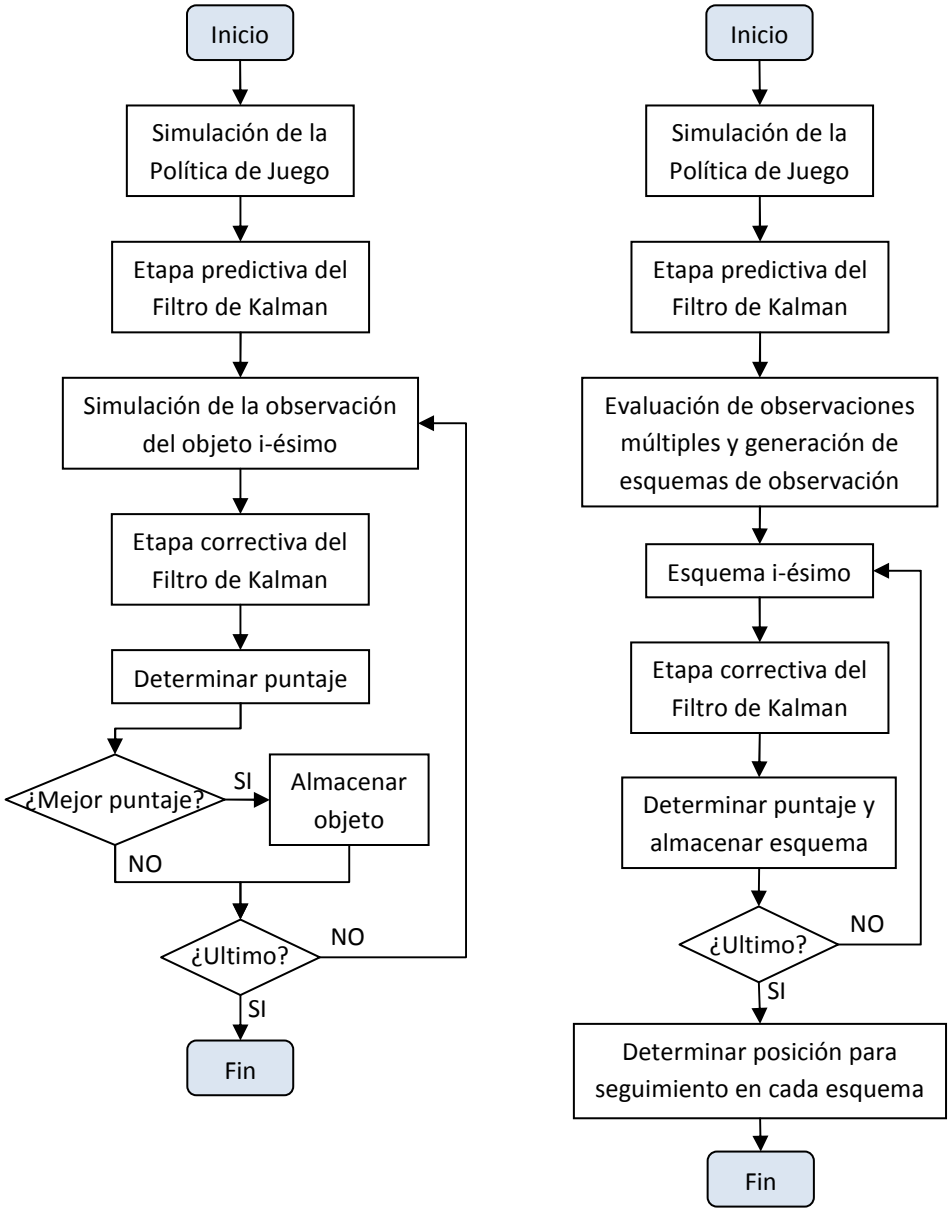
A pesar de que la formulación presentada buscó ser lo más general posible, fue necesario el realizar alguno supuestos con los que se trabajó y que imponen limitaciones en el funcionamiento del algoritmo implementado.

Tales supuestos y sus limitaciones se describen a continuación.

- El número de objetos de interés que se consideró en el funcionamiento del algoritmo es de 5 objetos, los cuales corresponden a la pelota, arcos azul y amarillo, faros azul-amarillo-azul y amarillo-azul-amarillo. A pesar de que este supuesto no representa una limitación directa al algoritmo desarrollado, sino más bien una simplificación del ambiente para acotar los elementos de interés existentes, impone un escenario en el cual se desenvuelve el algoritmo.
- El número máximo de objetos que pueden ser vistos simultáneamente se limitó en 4. Esta restricción fue impuesta para hacer abordable el problema correspondiente a las posibles combinaciones de objetos que es posible ver dentro del ambiente e introduce una limitación directa al funcionamiento de la etapa de evaluación de múltiples observaciones. No obstante, la condición impuesta puede ser considerada lo suficientemente general como para cubrir prácticamente todos los casos posibles, dado que en el ambiente del fútbol robótico es poco factible encontrar 4 objetos que puedan ser ubicados simultáneamente en un cuadro del sensor visual. Adicionalmente y debido a que el supuesto anterior limita el número de objetos de interés, se hace poco probable que 4 de estos objetos puedan ser vistos al mismo tiempo, brindando validez a este supuesto a pesar de las limitaciones introducidas.

3.4.2. Algoritmo de Visión Activa Modificado.

A continuación, en la Figura 3.21, se presenta un diagrama comparativo entre el flujo de ejecución para cada llamado al algoritmo de visión activa, antes y después de las modificaciones realizadas para la ampliación del espacio de acciones.



a) Algoritmo original *b) Algoritmo modificado*

Figura 3.21: *Comparación entre versiones del algoritmo de visión activa con el cual se trabajó.*

Como es posible observar en la Figura 3.21, los cambios efectuados sobre el algoritmo de visión activa afectaron principalmente a la evaluación de la etapa correctiva del filtro de Kalman, cambiando las observaciones que son utilizadas en este y adicionando el cálculo de la posición utilizada para el seguimiento de coordenadas.

4. Pruebas y Análisis de Resultados.

4.1. Pruebas Realizadas.

A continuación se describen en detalle las pruebas efectuadas en cada uno de los puntos desarrollados. Estas pruebas fueron llevadas a cabo en el robot humanoide Nao de Aldebaran Robotics y en algunos casos en el simulador de alto nivel HL-Sim.

4.1.1. Pruebas a la Herramienta de Depuración de Trackers.

Las pruebas a la herramienta de depuración, consisten en evaluar el funcionamiento de las opciones de despliegue de información implementadas, esto es:

- Evaluar que al marcar la opción *View Tracked Positions* se muestren las posiciones de los objetos seleccionados.
- Evaluar que al marcar la opción *Show All Simultaneously* las posiciones de todos los objetos presentes en el mapa local del robot sean desplegadas.

Para llevar a cabo esto en el simulador HL-Sim deben ser seleccionadas las opciones señaladas y luego comprobar su correcto funcionamiento, es decir el despliegue la información seleccionada de acuerdo a las opciones y que además esta información sea la correcta.

Resultados Obtenidos:

Al efectuar las pruebas descritas se apreció que las posiciones de los objetos presentes en el mapa local del robot eran desplegadas correctamente, como se puede

apreciar en la Figura 4.1, resultado correspondiente al uso únicamente de la opción *View Tracked Positions*.

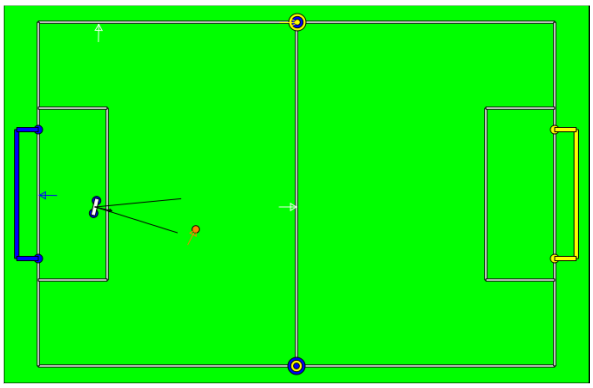


***Figura 4.1**: Datos desplegados con la opción de depuración View Tracked Positions.*

De manera similar, en la Figura 4.2 se puede apreciar el comportamiento de la información desplegada al seleccionar la opción *Show All Simultaneously*.

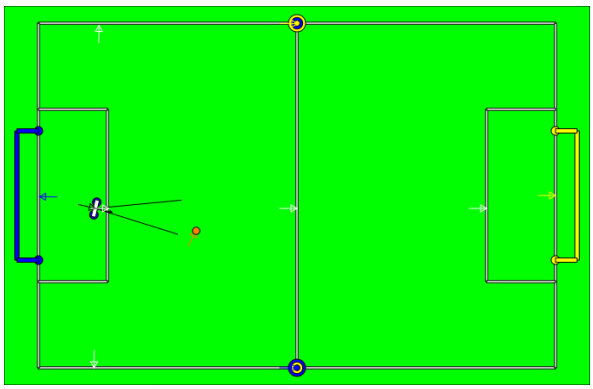


***Figura 4.2:** Datos desplegados con la opción de depuración Show All Simultaneously.*

Cabe mencionar que al desplegar simultáneamente las posiciones de todos los objetos presentes (ya sea con la opción *Show All Simultaneously* o sin esta y simplemente seleccionando todos los objetos), se pudo apreciar una disminución en la frecuencia de refresco de la imagen del campo de juego, hecho que no se produjo de forma perceptible al mostrar las posiciones sólo de algunos objetos. Esto último presumiblemente debido al uso más intensivo de la comunicación con el controlador de la pantalla, lo cual será abordado con mayor profundidad en la sección 0

4.1.2. Pruebas al Perceptor de Faros.

A continuación se detallan las pruebas llevadas a cabo en el perceptor de faros implementado, las cuales fueron realizadas utilizando el robot Nao de Aldebaran Robotics y

se enfocaron principalmente en determinar la tasa de detecciones correctas y de falsos positivos. Este enfoque obedece a que los datos de visión son los que entregan información sobre el ambiente y en base a los cuales se estima la localización del robot, por lo cual resulta esencial que estos sean correctos y en particular que no presenten información falsa (proveniente de percepciones erróneas) dado que esto afecta directamente a todo el sistema.

Para llevar a cabo la prueba se utilizan 6 videos del campo de juego, capturados con el robot Nao y donde estén presentes los faros. Tales videos generan un conjunto con un total de 1449 imágenes. Finalmente la prueba consiste simplemente en evaluar dentro del set de imágenes la cantidad de detecciones correctas e incorrectas, haciendo la distinción entre "verdaderos positivos", "falsos positivos", "verdaderos negativos" y "falsos negativos".

Resultados Obtenidos:

Durante la prueba se registraron datos acerca de las clasificaciones efectuadas por el perceptor, las cuales fueron comparadas con los datos para la validación (*ground truth*) generado para el conjunto de imágenes, obteniendo la cantidad de aciertos y detecciones erróneas. Los resultados de esta prueba se aprecian en la Figura 4.3.

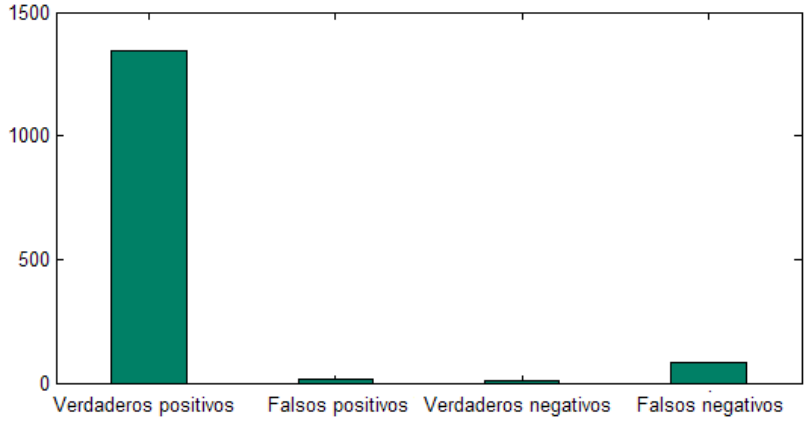


Figura 4.3: *Desempeño del perceptor de faros.*

Para entregar una mejor idea sobre el desempeño del perceptor esta información puede ser expresada como porcentaje de detecciones correctas en incorrectas, obteniendo los resultados que se presentan en la Tabla 4.1.

	Cantidad de Clasificaciones [n]	Porcentaje sobre el total de clasificaciones [%]
Verdaderos Positivos	1344	92,76
Falsos Positivos	16	1,10
Verdaderos Negativos	9	0,62
Falsos Negativos	80	5,52
Total	1449	100

***Tabla 4.1**: Desempeño del perceptor de faros.*

Esta prueba fue realizada utilizando para la segmentación una tabla de valores construida previamente para el trabajo con el robot, la cual en virtud de esto resulta robusta ante los cambios de iluminación existentes en el lugar donde se efectuó la captura de los videos[9].

Por esta razón, al individualizar las pruebas a cada video y generar una tabla de segmentación particular para cada uno, los resultados no se vieron afectados considerablemente e incluso en algunos casos estos empeoraron, ya que la tabla de valores para la segmentación resultó de menor calidad que la existente, como se muestra en la Tabla 4.2, donde se presentan los resultados para 4 de los 6 videos utilizados.

	Resultados con Segmentación Inicial		Resultados con Segmentación Individual	
	Verdaderos Positivos	**Falsos Positivos**	**Verdaderos Positivos**	**Falsos Positivos**
Video N°1	199	14	194	12
Video N°2	344	1	356	0
Video N°3	264	1	264	1
Video N°4	236	0	238	0

***Tabla 4.2:** Resultados obtenidos con segmentación para cada video.*

4.1.3. Pruebas a la Rutina de Seguimiento Visual.

A continuación se detallan las pruebas efectuadas a la rutina de seguimiento visual. Estas pruebas fueron realizadas en su totalidad en el robot Nao de Aldebaran Robotics, a excepción de la prueba N°4 que fue realizada utilizando el simulador de alto nivel HL-Sim.

[9] Laboratorio de Robótica de Departamento de Ingeniería Eléctrica de la Universidad de Chile (DIE).

- Prueba N°1:

Esta prueba está enfocada a obtener información respecto al funcionamiento del seguimiento por posición, en particular el comportamiento de este considerando objetos extras al realizar el seguimiento por posición de un objeto (funcionalidad descrita en la Sección 3.3.1.1). Para tales propósitos, la prueba consiste en establecer el montaje presentado en la Figura 4.4, en el cual el robot se encuentra en la posición (300,400) y orientado como se observa, los arcos azul y amarillo en las posiciones (0,200) y (600,200) respectivamente, el faro azul-amarillo-azul en la posición (300,0) y la pelota en (400,100).

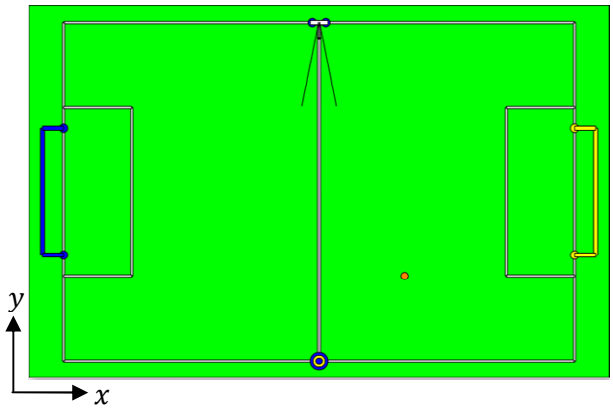


Figura 4.4: *Montaje utilizado en la prueba N°1.*

Teniendo este montaje se ejecuta una rutina en la cual el robot se mantiene de pie, sin moverse y una vez detectadas las posiciones de los arcos, faro y pelota, se observa alternadamente los arcos azul y amarillo, cada uno por 5 segundos.

Resultados Obtenidos:

Para esta prueba se realizaron 2 experimentos de aproximadamente 2 minutos de duración durante los cuales el robot se comportó de acuerdo a lo esperado, siendo posible observar la inclusión de los objetos adicionales en la trayectoria de la cabeza. En cada experimento se alternó la atención visual entre los arcos sucesivas veces, siendo posible apreciar la generación de trayectorias.

Durante los experimentos se obtuvo información acerca de los ángulos a los que debía moverse la cabeza de acuerdo al seguimiento por posición, los ángulos determinados por el seguimiento visual (luego de que el objeto fuera visto), las trayectorias planificadas y

las realizadas. En la Tabla 4.3 se presentan los datos acerca de las posiciones de la cabeza al llegar a los arcos.

		Seguimiento Visual				Seguimiento por Posición			
	Arco	**Promedio** $(tilt, pan)$ [°]		**Varianza** $(\sigma^2_{tilt}, \sigma^2_{pan})$ [°²]		**Promedio** $(tilt, pan)$ [°]		**Varianza** $(\sigma^2_{tilt}, \sigma^2_{pan})$ [°²]	
Experimento N°1	Azul	2,11	-56,99	0,78	42,36	2,74	-44,12	0,78	321,34
	Amarillo	2,99	56,49	0,29	9,70	2,02	52,02	0,00	0,03
Experimento N°2	Azul	2,55	-52,87	1,04	30,11	1,98	-55,43	0,00	0,38
	Amarillo	3,60	59,81	0,20	8,46	2,02	55,4	0,01	0,87

Tabla 4.3: *Datos correspondientes a la prueba N°1 de la rutina de seguimiento.*

La diferencia entre las posiciones dadas por el seguimiento por posición y el seguimiento visual se presentan en la Tabla 4.4.

		Error			
	Arco	**Promedio** $(tilt, pan)$ [°]		**Varianza** $(\sigma^2_{tilt}, \sigma^2_{pan})$ [°²]	
Experimento N°1	Azul	0,58	11,77	1,19	348,17
	Amarillo	-0,97	-4,37	0,29	9,67
Experimento N°2	Azul	-0,57	-2,45	1,02	30,38
	Amarillo	-1,58	-4,47	0,21	9,75

Tabla 4.4: *Diferencia en ángulos entre seguimiento visual y por posición, Prueba N°2.*

Cabe mencionar que dadas las posiciones de los objetos, es posible determinar los ángulos asociados a cada uno, pudiendo establecer de tal manera un parámetro de comparación. En la Tabla 4.5 se presenta dicha información.

	Tilt [°]	Pan [°]
Arco Azul	1,99	-56,31
Arco Amarillo	1,99	56,31
Pelota	8,68	18,44
Faro	3,23	0,00

Tabla 4.5: *Ángulos teóricos asociados a las posiciones de cada objeto en la Prueba N°1.*

En la Figura 4.5 se presentan algunas de las trayectorias efectuadas durante los experimentos. En estas se puede apreciar que los ángulos recorridos alcanzan las

coordenadas correspondientes tanto al faro como a la pelota (objetos extra dentro de la trayectoria de la cabeza).

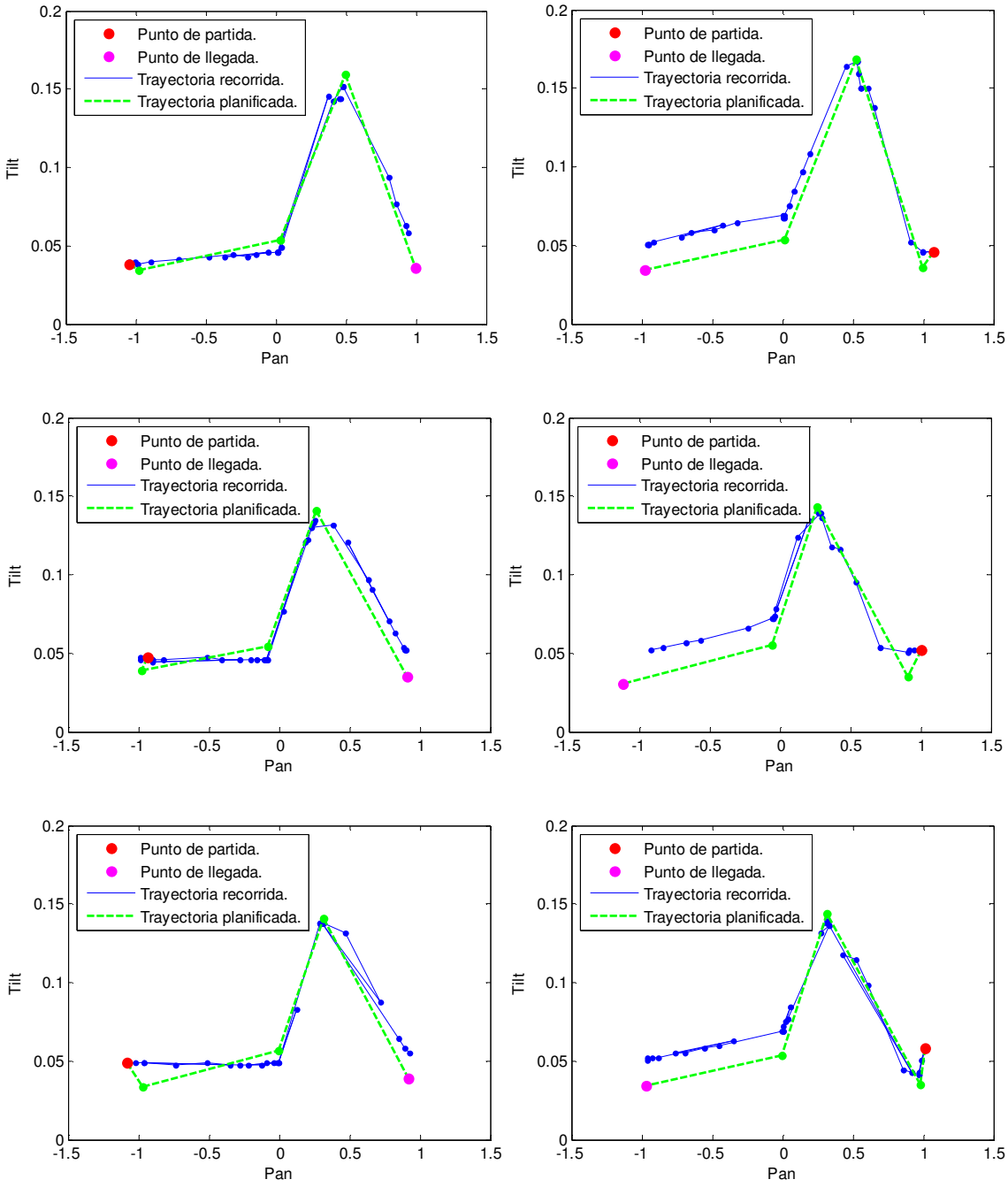


Figura 4.5: *Trayectorias de la cabeza efectuadas por el robot durante la prueba N°1.*

- Prueba N°2:

Esta prueba se orienta a la evaluación del seguimiento por posición cuando se efectúa un cambio en la cámara que está usando el robot. Para esto se utiliza un montaje como el que se muestra en la Figura 4.6, donde la pelota está ubicada a 40 cm del robot.

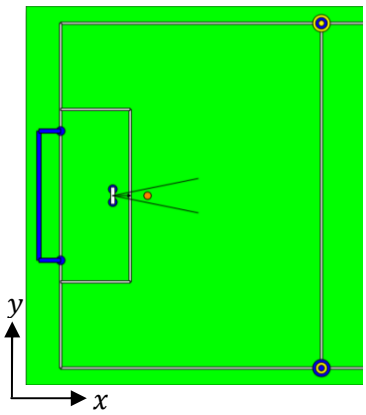


Figura 4.6: *Montaje utilizado en la prueba N°2.*

Con este montaje se ejecuta una rutina en la que el robot se mantiene de pie la posición (60,200) y alterna su atención visual para observar el faro azul-amarillo-azul en la posición (300,0), la pelota en la posición (100,200) y el faro amarillo-azul-amarillo en la posición (300,400), repitiendo este ciclo y siempre observando la pelota después de haber observado uno de los faros. Dado que la pelota se encuentra cercana al robot, esta será observada con la cámara inferior del robot mientras que los faros serán observados con la cámara superior, permitiendo evaluar el comportamiento al producirse un cambio de cámara.

Resultados Obtenidos:

Para esta prueba se realizaron 2 experimentos de 2 minutos de duración donde, al igual que en la prueba anterior se obtuvo información acerca de los ángulos de la cabeza, los ángulos a la que esta debía moverse de acuerdo al seguimiento por posición y los ángulos determinados por el seguimiento visual luego de ver el objeto de interés.

Durante los 2 experimentos el robot tuvo el comportamiento esperado, fijando su atención visual en cada uno de los objetos, realizando correctamente el cambio de cámara necesario para poder observar la pelota. En la Tabla 4.6 se presentan los datos obtenidos de los experimentos.

		Seguimiento Visual				Seguimiento por Posición			
		Promedio $(tilt, pan)$ [°]		**Varianza** $(\sigma^2_{tilt}, \sigma^2_{pan})$ [°²]		**Promedio** $(tilt, pan)$ [°]		**Varianza** $(\sigma^2_{tilt}, \sigma^2_{pan})$ [°²]	
Experimento N°1	Pelota	8,03	-6,16	0,03	0,02	9,65	-6,55	4,99	0,16
	Faro 0	5,01	-41,73	0,00	0,00	4,08	-39,98	0,00	0,00
	Faro 1	5,28	36,51	0,02	0,07	3,64	36,37	0,01	0,02
Experimento N°2	Pelota	10,25	-3,73	0,07	0,08	12,99	-5,58	3,17	14,99
	Faro 0	5,13	-38,05	0,14	0,04	4,00	-36,44	0,02	0,18
	Faro 1	5,21	40,65	0,03	0,06	3,83	39,83	0,01	0,28

Tabla 4.6: *Datos correspondientes a la prueba N°2 de la rutina de seguimiento.*

La diferencia entre las posiciones usadas para el seguimiento por posición y el seguimiento visual se presenta en la Tabla 4.7.

		Error			
		Promedio $(tilt, pan)$ [°]		**Varianza** $(\sigma^2_{tilt}, \sigma^2_{pan})$ [°²]	
Experimento N°1	Pelota	-3,03	0,63	0,56	0,07
	Faro 0	0,93	-1,75	0,00	0,00
	Faro 1	1,57	0,04	0,03	0,11
Experimento N°2	Pelota	-3,21	0,79	1,05	1,84
	Faro 0	1,14	-1,64	0,25	0,14
	Faro 1	1,41	0,66	0,03	0,22

Tabla 4.7: *Diferencia en ángulos entre seguimiento visual y por posición, Prueba N°2.*

Es pertinente señalar que dadas las posiciones de los objetos, los ángulos asociados a cada uno son los presentados en la Tabla 4.8, los cuales pueden ser determinados utilizando las ecuaciones 3.10, 3.11 y 3.12.

	Tilt [°]	Pan [°]
Pelota	16,34	0,00
Faro 0	4,13	-39,80
Faro 1	4,13	39,80

Tabla 4.8: *Ángulos teóricos asociados a las posiciones de cada objeto en la Prueba N°2.*

- Prueba N°3:

Esta prueba se enfoca en emular el acercamiento a la pelota, mientras es necesario mantener actualizada la posición de un objeto determinado, posibilitando la evaluación del comportamiento del seguimiento por posición con el robot en movimiento. Adicionalmente, esta prueba permite apreciar nuevamente el comportamiento del seguimiento por posición cuando es necesario realizar un cambio de cámara para observar los diferentes objetos de interés.

La prueba consiste en situar al robot a 1 metro de la pelota y a una distancia arbitraria de algún objeto de interés. El robot comienza a acercarse a la pelota observándola y si la confidencia de la estimación de la posición del objeto de interés baja de un umbral establecido, el robot intenta realizar seguimiento de la posición de dicho objeto, con el fin de observarlo y de tal forma corregir la estimación de la posición de este. Una vez mejorada la confidencia, se vuelve a enfocar la atención visual en la pelota. Este comportamiento continúa hasta que el robot se encuentre a menos de 20cm de la pelota, momento en el cual se da por terminada la prueba.

Resultados Obtenidos:

Esta prueba se llevó a cabo con 2 configuraciones de objetos. La primera consistió en acercase a la pelota al tiempo que se buscaba mantener actualizada la posición de un faro ubicado al lado derecho del robot (correspondiente al faro azul-amarillo-azul), mientras que en la segunda configuración se realizó el acercamiento a la pelota manteniendo actualizada la posición de un arco ubicado frente a la pelota y el robot. Para cada una de estas configuraciones se efectuaron 3 experimentos.

Durante cada experimento el robot se comportó de acuerdo a lo esperado, alternando la visión entre los objetos de interés al tiempo que se acercaba a la pelota. En la Figura 4.7 es posible apreciar la variación temporal de la confidencia del objeto de interés junto con la cámara que está siendo utilizada para llevar a cabo las observaciones[10], el objeto que desea observar y las variaciones en los ángulos de *tilt* y *pan* determinados por la

[10] Representada en este caso con los valores 0,5 para la cámara superior y -0,5 para la cámara inferior del robot.

rutina de seguimiento. Adicionalmente, las diferentes zonas de color dan cuenta del tipo de seguimiento realizado por la cabeza en cada instante, correspondiendo la zona verde a los momentos en que se ejecutaba seguimiento visual y la zona celeste a los momentos en que se ejecutaba seguimiento por posición. Dicha figura corresponde a los resultados obtenidos en el experimento N°1, utilizando la configuración N°1. Las figuras con los resultados de los restantes experimentos pueden ser observadas en el Anexo N°1.

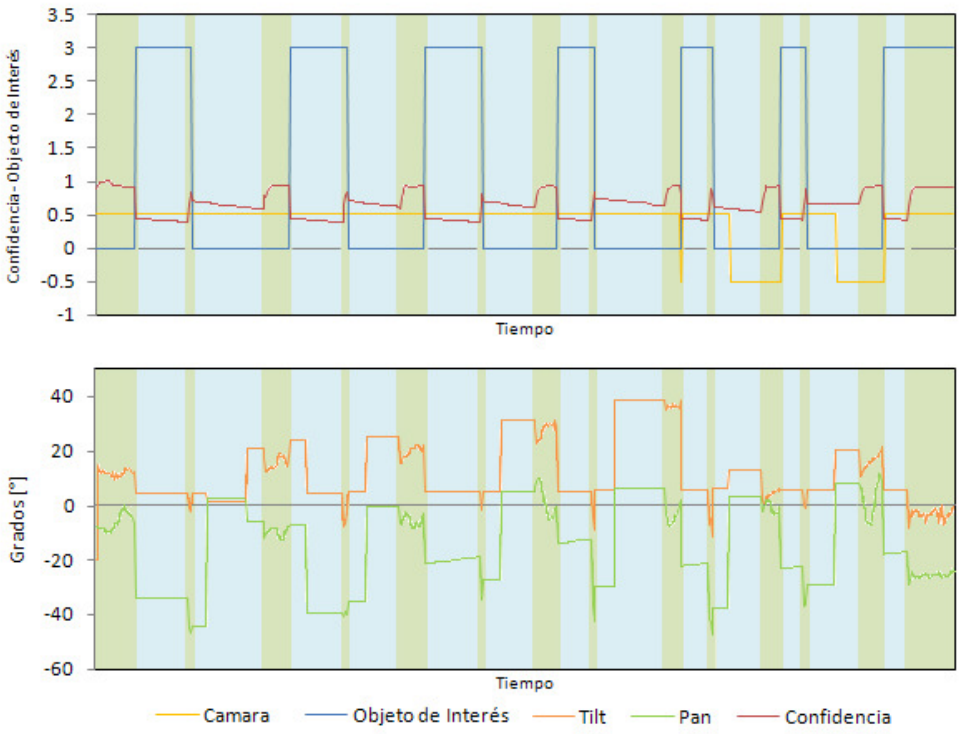


Figura 4.7: *Experimento N°1, configuración N°1.*

Es necesario señalar que en el gráfico de la Figura 4.7, la curva "Objeto de Interés" corresponde al objeto que tiene la atención visual del robot en dicho momento y los valores que esta presenta corresponden a las ubicaciones de los objetos dentro del mapa local del robot, ubicaciones que son detallas en la Tabla 4.9.

Objeto	Ubicación dentro del mapa local
Pelota	0
Arco Azul	1
Arco Amarillo	2
Faro Azul-Amarillo-Azul	3
Faro Amarillo-Azul- Amarillo	4

Tabla 4.9: *Ubicación de los objetos dentro del mapa local del robot.*

- Prueba N°4:

Esta prueba se orienta en comprobar el correcto funcionamiento del seguimiento en base a la función de distribución de probabilidad de la ubicación de un objeto (funcionalidad descrita en la Sección 3.3.2). Para esto se obliga al seguimiento por posición a utilizar dicha funcionalidad puesto que, en una situación de juego normal, los objetos de interés no siempre presentan alta incertidumbre en la estimación de sus posiciones. La manera de forzar al algoritmo a utilizar tal funcionalidad es ajustando el umbral de trabajo de este, para admitir como elementos con alta incertidumbre a objetos que tienen buenas estimaciones de sus posiciones.

Dado que en la prueba N°2 se pudo evaluar la capacidad de la cabeza para seguir una trayectoria planificada, esta prueba consiste sólo en comprobar la correcta generación de trayectorias a partir de la función de distribución de probabilidad de la ubicación de un objeto.

Resultados Obtenidos:

En esta prueba se evaluó el comportamiento utilizando 2 objetos diferentes: la pelota y el arco amarillo. Durante la prueba tanto el robot como la pelota estuvieron situados en posiciones arbitrarias dentro del campo de juego.

En ambos casos que la generación de trayectorias fue correcta, de acuerdo a los criterios utilizados para ello. En la Figura 4.8 se presenta la trayectoria generada en el caso de la pelota y en la Figura 4.9 la trayectoria generada en el caso del arco amarillo.

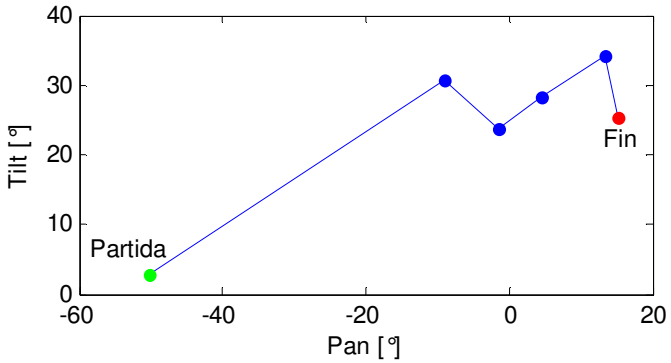


Figura 4.8: *Trayectoria generada en el caso de la pelota.*

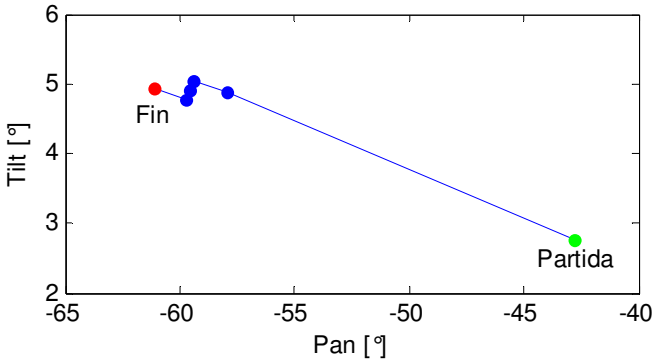


Figura 4.9: *Trayectoria generada en el caso del arco amarillo.*

Ambas trayectorias fueron determinadas como se describe en la Sección 3.3.2, a partir de las funciones de distribución de probabilidad de la ubicación de los objetos, las cuales estuvieron caracterizadas por su media y matriz de covarianza como sigue.

$$\mu_{PELOTA} = \begin{pmatrix} 89{,}12 \\ 6{,}88 \end{pmatrix} \quad \wedge \quad {\sigma^2}_{PELOTA} = \begin{bmatrix} 166{,}68 & -4{,}51 \\ -4{,}51 & 159{,}22 \end{bmatrix}$$

$$\mu_{ARCO} = \begin{pmatrix} 284{,}53 \\ -484.22 \\ -0{,}9645 \end{pmatrix} \quad \wedge \quad {\sigma^2}_{ARCO} = \begin{bmatrix} 60{,}26 & 0{,}4808 & 0{,}0002 \\ 0{,}4808 & 59{,}41 & 0{,}0001 \\ 0{,}0002 & 0{,}0001 & 0{,}1678 \end{bmatrix}$$

- <u>Prueba N°5:</u>

Esta prueba se enfoca en evaluar el desempeño del seguimiento de coordenadas. Para esto la rutina de visión activa es forzada a evaluar un set de objetos dispuestos espacialmente de modo de poder ser observados en conjunto. A dicho set de objetos se les asigna artificialmente un puntaje alto, de forma de asegurar la selección de estos elementos como objetivos del seguimiento. La prueba se inicia fijando la atención visual en cada uno de los objetos por separado, para así poder determinar su posición en el mapa local y subir la confidencia de la estimación. Una vez hecho esto se pasa a la observación del conjunto.

De tal manera, esta prueba permite apreciar el funcionamiento del seguimiento de coordenadas e indirectamente, evaluar el funcionamiento de la rutina de visión activa, de la misma forma que los criterios utilizado en las transiciones de estado existentes en la rutina de control de la cabeza.

Resultados Obtenidos:

Durante la prueba el robot intentó hacer el seguimiento de los múltiples objetos, para lo cual se favorecieron 2 configuraciones espaciales, las que correspondieron los grupos (pelota, arco amarillo, faro azul-amarillo-azul) y (pelota, faro azul-amarillo-azul). Estas configuraciones fueron consecuentemente dispuestas en el campo de juego, de modo de hacer posible la observación de dichos conjuntos. De tal manera, esta prueba cubre los 2 casos más significativos de la implementación realizada de la funcionalidad antes descrita.

Para cada una de las configuraciones se llevó a cabo 2 experimentos de aproximadamente 1 minuto de duración cada uno.

En la Figura 4.10 se presenta un diagrama temporal, en el cual se puede apreciar la variación de la confidencia asociada a la estimación de la posición de cada uno de los objetos del grupo de interés (en la figura, grupo conformado por pelota y faro), la posición de la cabeza (en grados) y los ángulos objetivo determinados por el seguimiento de coordenadas. De forma análoga a la prueba N°3, las zonas de color dan cuenta de los estados del seguimiento, correspondiendo las zonas verdes a la ejecución del seguimiento visual y las zonas celestes a la ejecución del seguimiento de coordenadas. Las figuras correspondientes a los resultados de las restantes pruebas pueden ser apreciadas en el Anexo N°2.

Es necesario señalar que en este caso por tratarse de grupos, la curva "Objeto de Interés" debe reflejar las posibles combinaciones de estos, utilizando para ello la representación antes detallada en la Tabla 4.9, a la cual se le adiciona la información que se presenta en la Tabla 4.10.

Combinación de Objetos	Valor Representativo
0 - 2	-1
0 – 3	-2
0 - 2 - 3	-3

Tabla 4.10: *Valores representativos utilizados en la prueba N°5 de la rutina de seguimiento.*

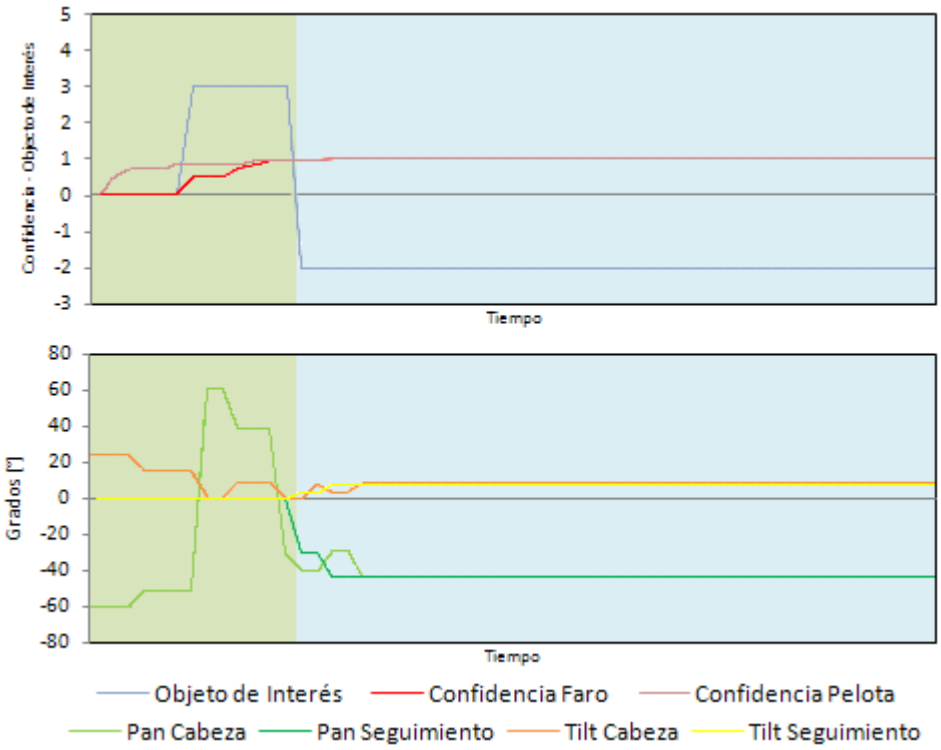
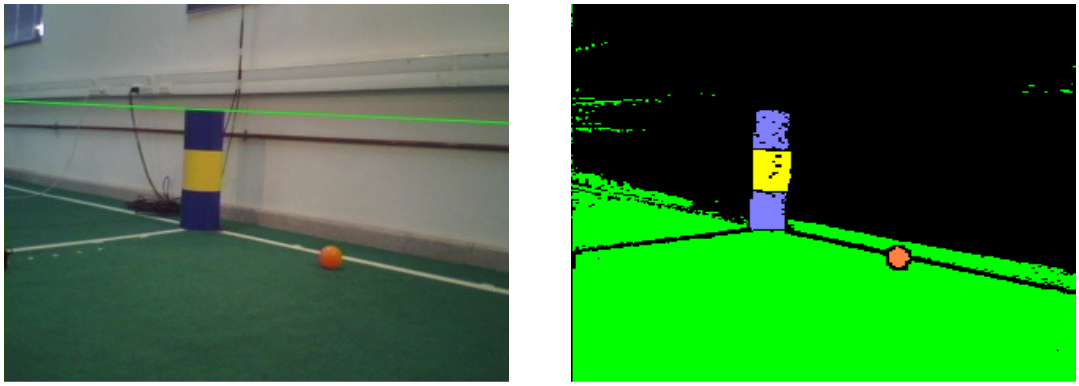


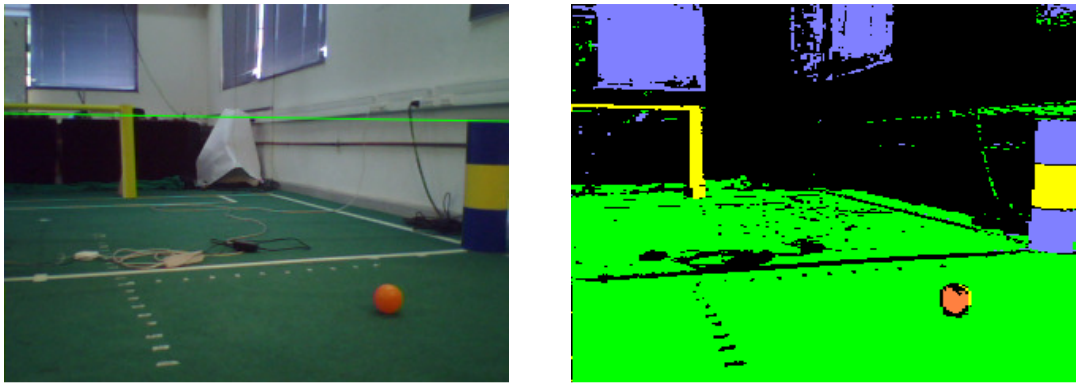
Figura 4.10: *Experimento N°1, configuración N°1, conjunto de pelota y faro.*

Durante la ejecución del seguimiento por coordenadas del grupo fue posible observar los objetos de interés con conjunto, tal como se aprecia en la Figura 4.11 y Figura 4.12, correspondientes al primer experimento de la primera y segunda configuración, respectivamente.



a) Imagen capturada. *b) Imagen segmentada.*

Figura 4.11: *Imagen de referencia, |experimento N°1, configuración N°1.*



a) Imagen capturada. *b) Imagen segmentada.*

Figura 4.12: *Imagen de referencia, experimento N°1, configuración N°2.*

4.1.4. Pruebas a la Rutina de Visión Activa.

Para evaluar el funcionamiento de la rutina de visión activa se efectuaron dos pruebas, de las cuales una fue llevada a cabo en el simulador de alto nivel HL-Sim y una fue realizada en el robot Nao de Aldebaran Robotics.

- Prueba N°1:

Esta prueba se enfoca en apreciar el funcionamiento en conjunto de la rutina de cobertura del arco (descrita en la Sección 2.1.4.1) y la rutina de visión activa, junto con lo cual se pretende evaluar el comportamiento de la función de valor y el puntaje asignado a cada grupo de posibles observaciones.

La prueba consiste en ejecutar la rutina de cobertura del arco junto con la rutina de visión activa, mientras simultáneamente la pelota se desplaza alrededor del arco a través de un set de posiciones previamente definidas como se observa en la Figura 4.13.

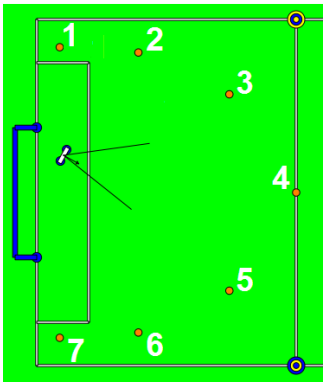


Figura 4.13: *Set de posiciones de la pelota para la prueba N°1 de la rutina de visión activa.*

Resultados Obtenidos:

Para esta prueba se realizaron 3 experimentos, utilizando el simulador de alto nivel HL-Sim, cada uno de los cuales tuvo 1 minutos de duración aproximadamente. Durante este tiempo se registró la función de valor generada para cada objeto/grupo por la rutina de visión activa. A lo largo de la prueba se pudo apreciar el comportamiento del robot, observando que este alternó su atención visual entre los diferentes objetos y grupos de objetos presentes en el campo de juego, mientras simultáneamente intentaba cubrir el arco de acuerdo a la rutina diseñada para esto, como se muestra en la Figura 4.14.

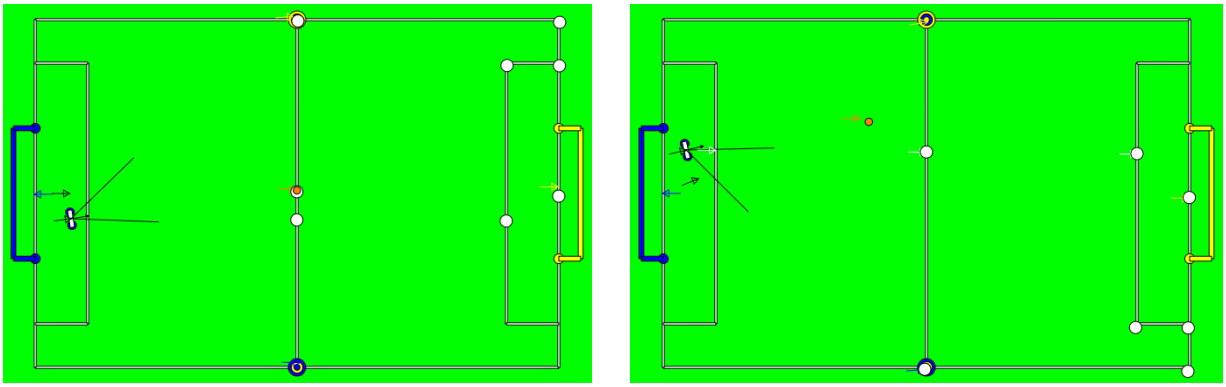


Figura 4.14: *Ejemplos del comportamiento del robot durante la prueba.*

En la Figura 4.15 se presenta la variación temporal del puntaje de grupo y la función de valor registradas durante el experimento N°1. Particularmente, en el último se presentan los cambios producidos en los objetos de interés, los cuales son detonados según al puntaje determinado por la rutina de visión activa. Ya que el interés visual puede estar en un único objeto o un grupo, la información que se muestra en dicho gráfico representa las combinaciones de objetos ocurridas durante los experimentos, combinaciones que son simbolizadas de acuerdo a lo detallado en la Tabla 4.11.

Objeto/Objetos	Valor Representativo	Objeto/Objetos	Valor Representativo
0	1	0 – 4	6
1	2	2 – 3	7
3	3	2 – 4	8
4	4	0 – 2 – 3	9
0 – 3	5	0 – 2 – 4	10

Tabla 4.11: *Correspondencia entre la Figura 4.15 y los objetos de interés.*

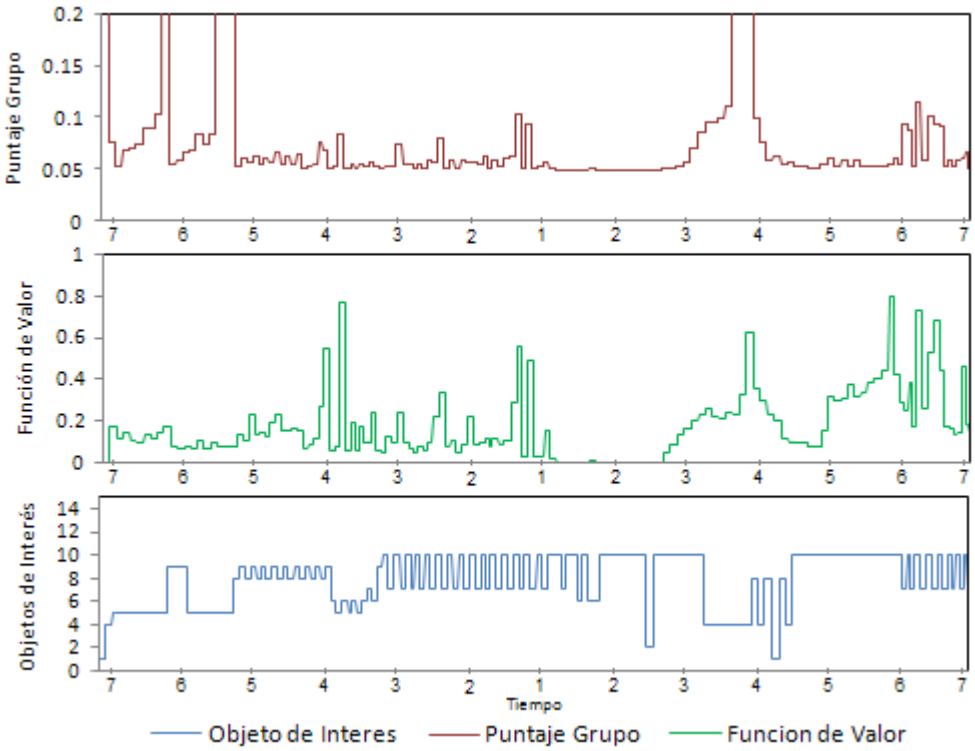


Figura 4.15: *Resultados del experimento N°1.*

Los resultados de los restantes experimentos pueden ser observados en el Anexo N°3.

- <u>Prueba N°2:</u>

Esta prueba pretende evaluar el funcionamiento de la rutina de visión activa al ser ejecutada en el robot Nao de Aldebaran Robotics, y en particular observar los cambios de la atención visual del robot hacia los diferentes los objetos o grupos de interés. Para esto se mantiene al robot caminando en su posición mientras se evalúa la rutina de visión activa, generando de tal manera datos de odometría, pero sin modificar su posición.

El robot es ubicado en la posición (15,200) frente al arco azul y orientado mirando hacia el arco amarillo, como se muestra en la Figura 4.16, mientras la pelota es desplazada arbitrariamente a través del campo de juego.

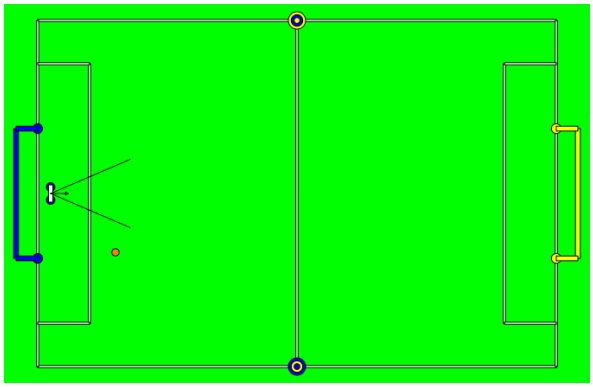


Figura 4.16: *Montaje utilizado en la prueba N°2 de la rutina de visión activa.*

Así, se puede apreciar el comportamiento de la rutina de visión activa, cambiando la atención visual del robot de acuerdo al ambiente y a la localización de este.

Resultados Obtenidos:

Durante la prueba se pudo apreciar los cambios de la atención visual, producidos por las variaciones en la localización así como por los cambios en el ambiente (dados por los movimientos de la pelota).

Se efectuaron 3 experimentos de aproximadamente 5 minutos cada uno[11], durante los cuales se registró el puntaje del objeto/grupo de interés, el estado de la rutina de control de la cabeza y el tiempo entre las visiones de cada objeto.

En la Figura 4.17 se presentan imágenes con ejemplos del comportamiento observado a lo largo de la prueba. En estas se pueden apreciar distintos grupos y objetos en los cuales se fijo la atención visual durante cada uno de los experimentos.

[11] El tiempo considerado para esta prueba resulta elevado comparativamente a las demás pruebas, sin embargo este permite evaluar completamente el comportamiento de la rutina de visión activa, probando diferentes situaciones, las que están dadas principalmente por las variaciones en la posición de la pelota.

Figura 4.17: *Ejemplos del comportamiento de la atención visual del robot durante la prueba.*

En la Figura 4.18 se presenta el puntaje de grupo y la función de valor, registrados durante el experimento N°1.

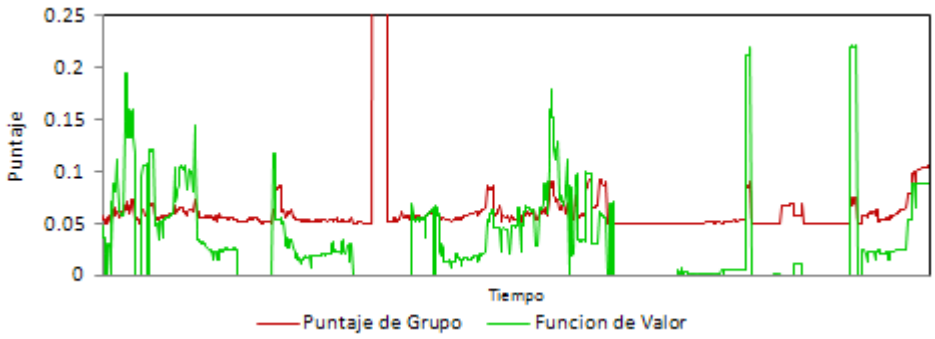


Figura 4.18: *Puntaje del grupo y función de valor en el experimento N°1.*

En la Figura 4.19 se presenta un diagrama con la evolución temporal de el/los objetos a los cuales se les realizó seguimiento (curva *"Estado Seguimiento"*), el estado final de la cabeza, en el caso de existir objetos que no fueron vistos (curva *"Estado Seguimiento Reducido"*) y finalmente en el gráfico inferior se presenta el estado de la rutina de control de la cabeza, en el cual se presentan las categorías *"SS"*, *"MS"*, *"ST"*, *"MT"* y *"RT"*, correspondientes a los estados de *búsqueda individual*, *búsqueda múltiple*, *seguimiento individual*, *seguimiento múltiple* y *seguimiento reducido*, respectivamente.

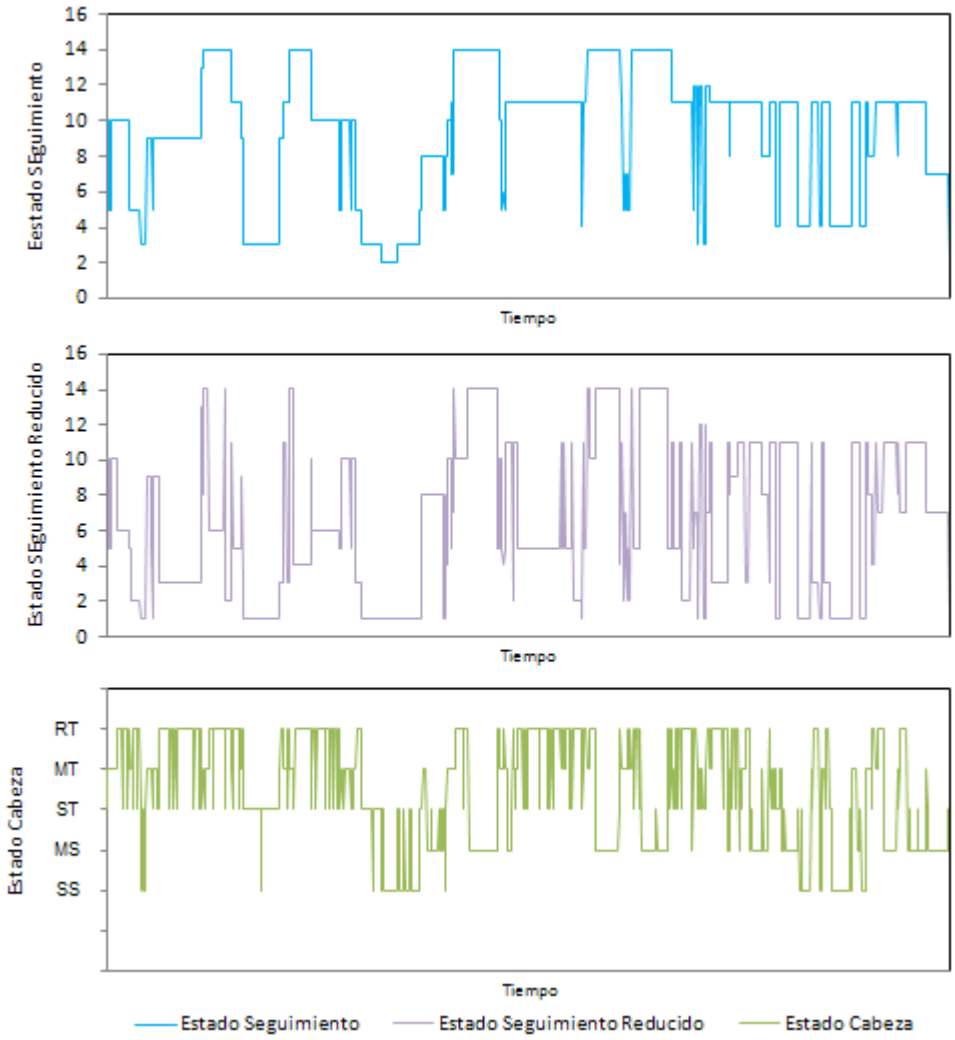


Figura 4.19: *Estado del seguimiento y de la rutina de control de la cabeza, experimento N°1.*

Es necesario aclarar que para poder ser expresados como una única curva, los datos correspondientes a las curvas "*Estado Seguimiento*" y "*Estado Seguimiento Reducido*", debieron ser representados de forma análoga lo realizado al presentar los datos de la prueba anterior. Dicha representación es detallada en la Tabla 4.12.

Objeto/Objetos	Valor Representativo	Objeto/Objetos	Valor Representativo
Ninguno	1	2 – 3	8
0	2	2 – 4	9
2	3	0 – 2 – 3	10
3	4	0 – 2 – 4	11
4	5	0 – 3 – 4	12
0 – 3	6	2 – 3 – 4	13
0 – 4	7	0 – 2 – 3 – 4	14

Tabla 4.12: *Correspondencia entre la Figura 4.19 y los objetos de interés.*

También es necesario aclarar que el estado correspondiente a "ningún objeto", que es representado por el valor 1, es válido únicamente para el seguimiento reducido y corresponde a los momentos en que se realizó seguimiento a un objeto individual, caso para el cual deja de tener sentido la existencia del seguimiento reducido.

Debido a que la cantidad de información presente en cada uno de los gráficos de la Figura 4.19 corresponde a los 5 minutos que duraron las pruebas, resulta complejo poder apreciar correctamente el comportamiento de las transiciones de estado de la rutina de control de la cabeza, así como los cambios producidos en los objetos de interés. Por esta razón, y con el fin de ilustrar de mejor manera el comportamiento observado, en la Figura 4.20 y Figura 4.21 se presenta información correspondiente a aproximadamente un minuto de prueba, en particular en la Figura 4.20 se puede observar la función de valor y el puntaje de grupo.

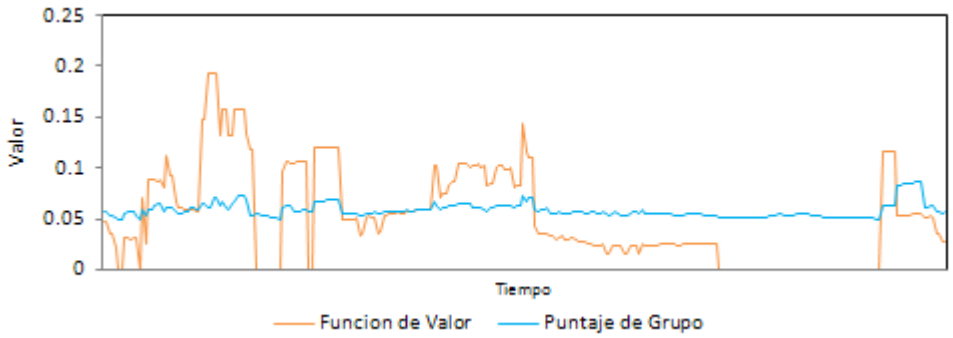


Figura 4.20: *Puntaje de grupo y función de valor en intervalo de tiempo menor.*

En la Figura 4.21 es posible apreciar la evolución de los objetos de interés (curvas "*seguimiento*" y "*seguimiento reducido*"), el estado de la rutina de control de la cabeza y adicionalmente se incluye un gráfico que presenta la evolución del "*tiempo desde la última visión*" de cada objeto.

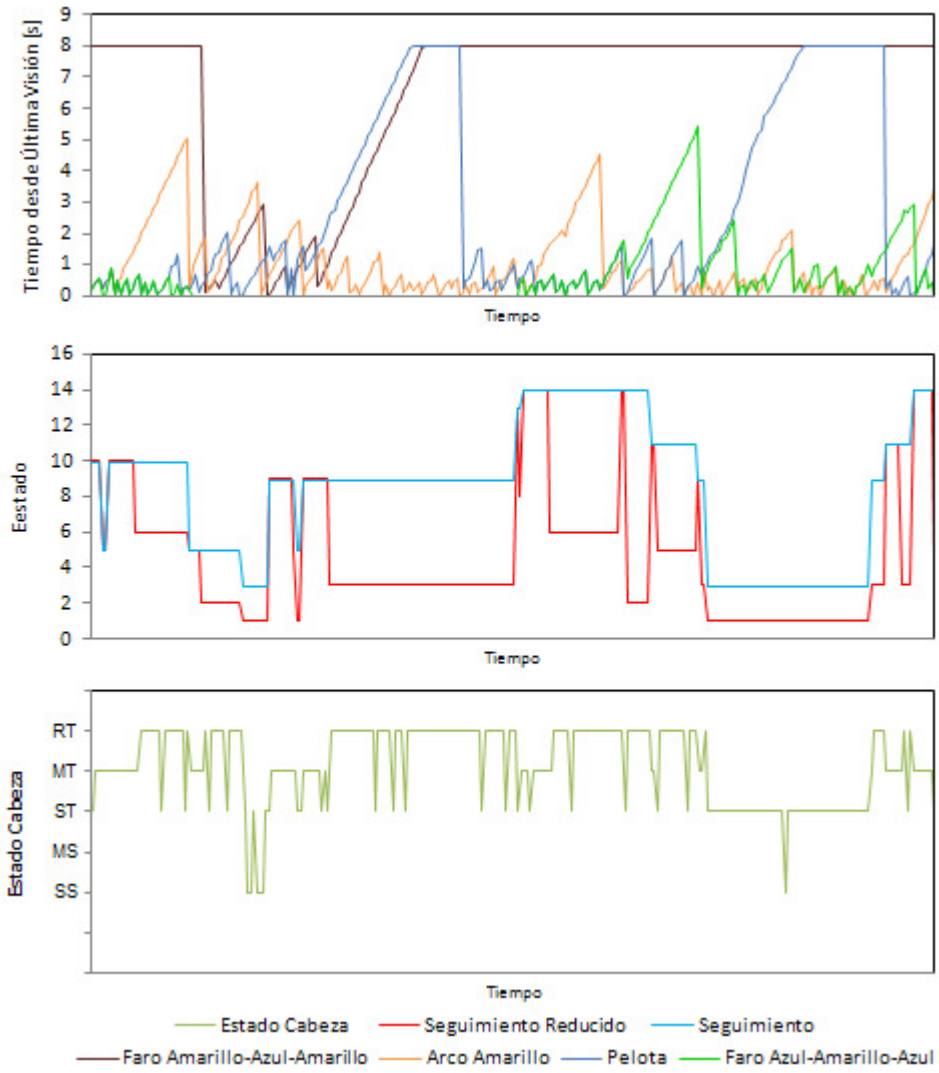


Figura 4.21: *Estado del seguimiento, rutina control de cabeza y tiempos entre observaciones.*

Los resultados correspondientes a los restantes experimentos pueden ser observados en el Anexo N°4.

En otro aspecto, en la Tabla 4.13 se presenta el porcentaje de tiempo que se mantuvo la atención visual en cada uno de los estados definidos en la Tabla 4.12. En esta tabla es posible observar el resultado conjunto de los experimentos, es decir el tiempo que se mantuvo la atención visual en cada uno de los estados a lo largo de los 3 experimentos. Para esto se consideró únicamente los estados exhibidos en la curva "*estado seguimiento*".

Estado	Porcentaje del Total del Tiempo [%]	Estado	Porcentaje del Total del Tiempo [%]
1	-	8	6,13
2	7,24	9	7,31
3	3,32	10	6,10
4	15,35	11	25,32
5	8,43	12	7,73
6	0,32	13	1,55
7	8,59	14	6,60

Tabla 4.13: *Porcentaje de tiempo que cada estado mantuvo la atención visual.*

Puesto que cada estado representa un objeto o grupo de objetos, es posible determinar para cada objeto la cantidad de tiempo que este mantuvo la atención visual del robot (ya sea individualmente o como parte de un grupo). Dichos resultados son presentados en la Tabla 4.14.

Objeto	Porcentaje del Total del Tiempo [%]
0	57,91
2	56,34
3	39,79
4	61,53

Tabla 4.14: *Porcentaje del tiempo que cada objeto mantuvo la atención visual.*

- Prueba N°3:

Esta prueba tiene por objetivo el determinar el tiempo promedio de ejecución de la rutina de visión activa, bajo diferentes combinaciones de representaciones para las funciones de distribución de probabilidad, con las cuales es evaluada la función de valor de la tarea (proceso descrito en la Sección 2.1.1 y Sección 3.4.1.1). Para esto, simplemente se ejecuta la rutina de visión activa, operando con las 4 combinaciones de representaciones posibles para la función de distribución de probabilidad de las observaciones (función de distribución de probabilidad de la estimación de la ubicación de las observaciones) y de la creencia del estado del robot (función de distribución de probabilidad de la estimación de la pose del robot). Los tiempos son registrados hasta obtener un total de 400 muestras.

Las representaciones posibles para cada función de distribución corresponden a la forma en que es representada en los cálculos dicha función. Esto puede ser realizado utilizando únicamente la media de esta o bien utilizar una representación discreta de la función distribución de probabilidad (obtenida mediante muestreo determinístico, como se señaló en la Sección 2.1.1). De tal forma se generan 4 casos posibles, los cuales son presentados en la Tabla 4.15.

Representación de la Creencia	Representación de las Observaciones
Representación discreta	Representación discreta
Sólo media	Representación discreta
Representación discreta	Sólo media
Sólo media	Sólo media

Tabla 4.15: *Combinaciones posibles de las representaciones de las funciones de distribución.*

Resultados Obtenidos:

Esta prueba fue llevada a cabo en el robot Nao de Aldebaran Robotics y en ella se registraron datos correspondientes a los tiempos de ejecución de la rutina de visión activa. En la Tabla 4.16 se presentan los datos obtenidos.

Representación de la Creencia	Representación de las Observaciones	Tiempo Promedio [s]
Representación discreta	Representación discreta	30,917
Sólo media	Representación discreta	0,841
Representación discreta	Sólo media	0,681
Sólo media	Sólo media	0,135

Tabla 4.16: *Tiempo promedio de ejecución de la rutina de visión activa.*

Cabe mencionar que los tiempos promedio fueron estimados utilizando las 400 muestras antes mencionadas, a excepción de la combinación "*Representación discreta-Representación discreta*". Esto se debió a que dado los tiempos de procesamiento observados, la obtención de 400 muestras habría significado una cantidad de tiempo inadmisible. En consecuencia, dicha estimación del tiempo promedio de ejecución fue realizada utilizando sólo 50 muestras.

4.2. Análisis de Resultados.

- **Herramienta de Depuración de Trackers.**

En los resultados obtenidos durante las pruebas efectuadas, se pudo apreciar un correcto funcionamiento de las opciones de depuración implementadas, de acuerdo a lo esperado para esta herramienta.

Al seleccionar la opción *View Tracked Positions*, ya sea individualmente o en conjunto con la opción *Show All Simultaneously*, se pudo apreciar que las posiciones eran desplegadas de forma correcta de acuerdo a la información entregada por la etapa de localización, en particular respetando a la intuición que indica que deberían ser desplegadas sólo las posiciones de los objetos previamente vistos, y desde luego desplegando la información correspondiente a el/los objeto seleccionados.

La disminución en la frecuencia de refresco del campo de juego, observada principalmente al desplegar todas las posiciones de los objetos del mapa local, obedece a que un mayor trabajo en cuanto a dibujo en la imagen, implica más tiempo de interacción con el controlador de la pantalla, dispositivo que como la mayoría de las Interfaces humanas resulta más lento que el eventual funcionamiento del resto del código, produciendo de esta manera el retardo apreciado, hecho que no obstante no perjudica el funcionamiento de los demás procesos o módulos existentes en el código.

- **Perceptor de Faros.**

En los resultados presentados en la Figura 4.3 y Tabla 4.1 se puede observar una baja tasa de detecciones correspondientes a falsos positivos (del orden del 1%) y una alto porcentaje de verdaderos positivos (cerca del 93%). Esto representa un buen resultado para el perceptor, entregando en términos generales información correcta y comparativamente poca información errónea (falsos positivos), la cual afecta fuertemente el desempeño del sistema.

También es posible observar que, al utilizar una tabla de segmentación particular para cada video, no se apreciaron diferencias sustanciales en los resultados obtenidos, como se muestra en la Tabla 4.2. Este hecho se debe a que la tabla de segmentación,

utilizada para obtener los resultados de la Figura 4.3, es robusta ante las condiciones de luz existentes en el lugar donde se efectuaron las pruebas. Por esta razón dichos resultados pueden ser considerados como representativos del comportamiento real del perceptor en el robot, plataforma donde las condiciones de luz influyen fuertemente, alterando la percepción de los colores y haciendo fácil la pérdida de información debido a una mala segmentación,

- **Rutina de Seguimiento Visual.**

En la prueba N°1 fue posible apreciar al robot variando su atención visual entre los objetos de interés (arcos amarillo y azul), a pesar de que estos se encontraban distantes y dejaban de ser vistos por un tiempo tal que el seguimiento visual dejaba de ser válido[12], dando paso a la ejecución del seguimiento por posición. Este hecho reviste un resultado positivo para la implementación efectuada.

En los resultados presentados en la Tabla 4.3 y Tabla 4.4 es posible apreciar la coherencia entre los ángulos determinados por el seguimiento visual y por posición. Esto significa que ambas funciones determinaron ángulos del mismo orden al observar un objeto particular. Este hecho permite una transición relativamente fluida entre tipos de seguimiento, es decir al pasar de seguimiento por posición a visual la cabeza no se mueve bruscamente.

Vale la pena destacar que los resultados presentados han sido expresados en términos de ángulos de la cabeza lo que implica un error creciente con la distancia a la cual se encuentra el objeto de interés. Sin embargo este punto no afecta al seguimiento visual, puesto que los ángulos determinados en tal caso son calculados con información proveniente directamente desde la imagen. Por esta razón resulta importante la existencia de coherencia, entre los ángulos determinados por el seguimiento por posición y visual, más allá de existir concordancia con los ángulos "teóricos" de cada objeto, presentes en la Tabla 4.5 y determinados en función de la posición relativa del objeto.

[12] El umbral de tiempo tras el cual se invoca el seguimiento por posición en vez del seguimiento visual es de 1,5 segundos, es decir si un objeto dejó de ser visto hace más de 1,5 segundos, la próxima vez que se intente ver dicho objeto se invocará a la función de seguimiento por posición directamente.

En las imágenes de la Figura 4.5 se pueden observar algunas de las trayectorias efectuadas por la cabeza durante la prueba, en las cuales se aprecia claramente la inclusión de 2 objetos extras (correspondientes a la pelota y uno de los faros), durante el desplazamiento de la cabeza de un punto a otro, indicando que la inclusión de objetos adicionales resulta correcta.

En los resultados obtenidos durante la prueba N°2 se puede apreciar una vez más la coherencia entre los ángulos determinados, tanto por el seguimiento visual como por el por posición, esta vez mediando un cambio de cámara al momento de efectuar el seguimiento por posición. En la Tabla 4.7 se puede observar que la diferencia entre ángulos es en promedio baja y además presenta una varianza pequeña, lo cual representa un buen resultado en términos de coherencia entre los ángulos determinados por cada tipo de seguimiento. Adicionalmente, al comparar los ángulos de la Tabla 4.6 y Tabla 4.8, es posible notar que la diferencia existente entre los ángulos "teóricos" para las posiciones de los objetos y los ángulos determinados por la rutina de seguimiento es pequeña, lo que representa también un buen resultado en términos de efectividad del seguimiento.

En los resultados de la prueba N°3 (como los que se presentan en la Figura 4.7) es posible apreciar los cambios en la confidencia durante el experimento, observando que al pasar del seguimiento por posición al seguimiento visual, la confidencia de la estimación de la posición del objeto se eleva. Este hecho pudo ser verificado en todos los instantes en que se pasó al seguimiento visual, incluyendo los casos en que se debió realizar un cambio de cámara para poder observar el objeto, lo cual corresponde a un comportamiento deseado y por tanto positivo en la evaluación del funcionamiento.

En otro aspecto, se puede ver que los cambios de cámara efectuados suceden hacia el final de la prueba y coinciden con los momentos en que el objeto de interés es la pelota (estado 0). Esto es coherente con la lógica de la prueba, la cual señala que hacia el final de esta el robot se encuentra cerca de la pelota, siendo necesario efectuar cambios de cámara para alternar la atención entre la pelota el objeto restante, ya sea un faro o un arco.

En la prueba N°4 se pudo apreciar el funcionamiento del seguimiento por posición al efectuar el seguimiento basado en la función de distribución de probabilidad de la ubicación de un objeto. En los resultados entregados por esta prueba, y presentados en la Figura 4.8 y

Figura 4.9, se puede observar que tanto el muestreo de las funciones de distribución de probabilidad como la generación de las trayectorias es correcto. Este hecho sumado a los resultados observados en la prueba N°1, los cuales evidenciaban la capacidad de la cabeza para seguir una trayectoria previamente planificada, configuran un resultado positivo para la generación y ejecución de trayectorias.

Adicionalmente, al observar la Figura 4.8 y Figura 4.9, es posible notar que el muestreo realizado a las funciones de distribución de probabilidad de la ubicación de los objetos es coherente con lo observado en la varianza de ambos objetos (pelota y arco), apreciando de tal forma una mayor dispersión de los puntos de la trayectoria de la Figura 4.8 y una menor dispersión en la trayectoria de la Figura 4.9.

Por último, en la prueba N°5, se observa el funcionamiento del seguimiento de coordenadas, característica que permite a la rutina de seguimiento (en conjunto con la rutina de visión activa) fijar la atención visual del robot no sólo en un objeto singular, sino que también en grupos de objetos espacialmente contiguos y de tal manera de obtener información simultánea de dichos objetos y no únicamente de uno sólo de ellos.

En los resultados de dicha prueba (como los presentados en la Figura 4.10) se puede apreciar cómo cambia el objeto de interés en la etapa de estimación de la posición de cada objeto, para luego pasar a la observación del conjunto. En esta última fase se puede apreciar que la confidencia de las estimaciones no baja, hecho que implica que los objetos siguen siendo vistos y por lo tanto el seguimiento del conjunto opera correctamente en tal caso.

Sin embargo, en la Figura 7.8 es posible observar insistentes cambios en los objetos de interés. Esto se explica puesto que por momentos no todos los elementos de un grupo resultan visibles, luego esto detona un cambio en el estado de la rutina de control de la cabeza, modificando el objeto de interés. Sin embargo es posible apreciar que luego este vuelve al valor anterior, dejando en evidencia que nuevamente todos los elementos del grupo de interés fueron vistos.

Vale la pena mencionar que los cambios en el grupo de interés pueden ser desencadenados por diferentes motivos, ya sea porque el objeto en cuestión no ha sido visto o porque el objeto no fue detectado correctamente por el perceptor respectivo, esto

último debido a la segmentación o a que las regiones de color generadas no configuran un elemento reconocible por el perceptor. A pesar de esto se puede apreciar que la confidencia de los objetos no se ve afectada y que los ángulos de la cabeza no varían fuertemente al momento de producirse modificaciones en el grupo de interés. Este hecho induce a pensar que tales cambios fueron rápidos (y como se puede apreciar en la Figura 7.8, sólo temporales) o bien que los objetos eran vistos intermitentemente y muy cerca del umbral de tiempo sobre el cual se producía el paso a un diferente estado, provocando el comportamiento observado.

- Rutina de Visión Activa.

En la prueba N°1 se pudo observar al robot ejecutando las rutinas de visión activa y cobertura del arco de forma simultánea. En los resultados de esta prueba, presentados en la Figura 4.15, es posible apreciar que el comportamiento de la función de valor es similar al observado en la anterior implementación de la rutina de visión activa (la cual presentaba un espacio de acciones sensoriales más reducido). Las posibles diferencias de comportamiento de esta (que resultan poco perceptibles al no tener una tendencia clara), pueden explicarse debido a que la plataforma utilizada en esta prueba, es diferente a la utilizada en las pruebas presentadas en [1], a lo que debe agregarse que, para evaluación de la función de valor, se utilizó una aproximación gruesa de la función de distribución de probabilidad del punto de llegada de la pelota tras un golpe de un robot.

En contraste al comportamiento altamente variable que es posible apreciar en la función de valor, el puntaje de grupo presentó un comportamiento mucho más estable, mostrando variaciones menos bruscas (a excepción de los momentos en que era impuesto un puntaje alto para asegurar la visión de la pelota), permitiendo observar que el cálculo del puntaje de grupo tuvo el efecto de filtro pasa bajos sobre la función de valor.

En cuanto a la selección del objeto/grupo de interés, en el último gráfico de la Figura 4.15, se puede observar que existe una cierta tendencia a mantenerse en algunos estados o a oscilar entre estos estados, particularmente en los estados 7, 8, 9 y 10, correspondientes a las combinaciones (arco amarillo, faro azul-amarillo-azul), (arco amarillo, faro amarillo-azul-amarillo), (pelota, arco amarillo, faro azul-amarillo-azul) y (pelota, arco amarillo, faro amarillo-azul-amarillo), respectivamente. Esto representa un resultado destacable dado

que dichas combinaciones se dan primordialmente cuando la pelota se encuentra cercana a las posiciones 3, 4 y 5, correspondientes a las posiciones centrales y donde es posible observar más objetos de forma simultánea (al encontrarse prácticamente todos los objetos presentes en el campo de juego "de frente" al robot).

En la prueba N°2 fue posible apreciar el comportamiento del robot al ejecutar la rutina de visión activa, en conjunto con una rutina que lo mantuvo de pie caminando en su lugar.

En los resultados de la prueba N°2 (como los presentados en la Figura 4.18) se puede apreciar que la función de valor tuvo un comportamiento semejante al observado en la prueba N°1, sin presentar una tendencia definida. De la misma forma, el puntaje de grupo observado durante esta prueba presentó una menor variabilidad, pudiendo apreciarse un comportamiento más estable, y en términos generales, alterado únicamente por las variaciones más fuertes y sostenidas de la función de valor.

En la Figura 4.19 puede observarse la evolución del estado del seguimiento a lo largo de la prueba, apreciando una cierta tendencia a mantenerse en los estados 4, 10 y 14 (de acuerdo a los estados definidos en la Tabla 4.12). También es posible apreciar que el estado de la rutina de control de la cabeza osciló insistentemente entre los estados de seguimiento individual y seguimiento reducido.

Al reducir el intervalo de tiempo de los datos presentados, es posible observar con mayor cuidado el comportamiento de las variables de interés. En la Figura 4.20 se puede observar nuevamente la evolución de la función de valor determinada por la rutina de visión activa, así como el puntaje calculado para cada grupo de objetos. Análogamente en la Figura 4.21 es posible observar la evolución del estado de la rutina de control de la cabeza. En dicha figura se pueden apreciar recurrentes transiciones entre los estados de seguimiento reducido (RT) y seguimiento individual (ST), hecho que puede ser explicado debido a que algunos de los objetos de interés dejan de ser vistos, lo cual provoca los cambios de estado observados.

Con los resultados obtenidos durante la prueba, también fue posible estimar el porcentaje de tiempo que cada estado mantuvo la atención visual del robot, información

que es presentada en la Tabla 4.13. Particularmente, en la Tabla 4.14 es posible observar individualmente el porcentaje de tiempo que cada objeto mantuvo la atención visual del robot, notando que la pelota corresponde al segundo objeto que mantuvo más tiempo la atención visual, sólo superado por el faro amarillo-azul-amarillo. Este corresponde a un resultado destacable, dado que durante más de la mitad del tiempo se mantuvo la atención del robot en el objeto de mayor importancia dentro del campo de juego, lo cual se realizó sin perjuicio de los restantes objetos que también presentan altos porcentajes de tiempo bajo la atención del robot.

En la prueba N°3 se determinó el tiempo de ejecución de la rutina de visión activa bajo diferentes configuraciones. En los resultados presentados en la Tabla 4.16 se puede observar que estos siguieron la intuición, la cual señalaba que el uso de la representación discreta traería consigo un mayor tiempo de ejecución, puesto que esto implicaría la evaluación de la función de valor con mayor de puntos número de puntos. De acuerdo a esto es posible apreciar que la combinación con sólo las medias de las funciones de distribución de probabilidad presenta un tiempo de procesamiento menor a todos, mientras que las combinaciones mixtas (que utilizan la media de una de las funciones de distribución de probabilidad y la representación discreta de la otra) presentan tiempos mayores y de un orden similar.

Sin embargo es posible apreciar que la combinación que utilizó las representaciones discretas de ambas funciones de distribución de probabilidad presentó un tiempo de procesamiento varios órdenes de magnitud superior a los demás. Esta fuerte diferencia puede explicarse mediante analizando el número de combinaciones que deben ser procesadas para la evaluación de la función de valor. En el peor caso la evaluación es cuando esta se realiza con un esquema de observación compuesto de 4 objetos, luego la función de valor deberá ser evaluada con los puntos de la representación discreta de 5 funciones de distribución en total (4 correspondientes a los objetos y 1 correspondiente a la creencia). Si cada representación discreta se compone de 5 puntos, entonces la cantidad de combinaciones con las que deberá ser evaluada la función de valor es igual a $5^5 = 3125$. Por otro lado el mejor caso es cuando la evaluación se realiza utilizando un esquema de observación compuesto por sólo un objeto, luego la cantidad de combinaciones que con las que se evaluará este caso es $5^2 = 25$.

Es simple notar que en ambos casos, el número de veces que se debe evaluar la función de valor de la rutina de visión activa, es mucho mayor que las 5 veces que esto es efectuado en los casos mixtos o la única vez que se realiza cuando se utiliza sólo la media de ambas funciones de distribución.

Finalmente, los resultados obtenidos y el comportamiento observados durante ambas pruebas, no permite establecer de manera certera que la ampliación del espacio de acciones sensoriales haya tenido un efecto positivo o negativo en el comportamiento de la rutina de visión activa, ni mucho menos en la incertidumbre existente en la localización del robot.

4.3. Trabajo Futuro.

A continuación se describen algunas de las mejoras o desarrollos complementarios posibles de realizar en este trabajo.

- La herramienta de depuración de trackers implementada permite observar las poses estimadas los objetos presentes en el mapa local, sin embargo esta herramienta no es capaz de reflejar la varianza que presenta dicha estimación. Dado que esta es información importante para la evaluación de las estimaciones, una tarea posible de abordar es complementar la herramienta de depuración de trackers con el despliegue de la varianza asociada a la estimación de la pose, para lo cual una opción adecuada es el dibujo de una elipse representativa, donde los radios de esta correspondan a la desviación estándar existente en cada eje.

- Evaluar una modificación en el algoritmo de visión activa, en conjunto con el de la rutina de seguimiento, para considerar los objetos como cuerpos con un radio asociado y no únicamente como objetos puntuales (tal como se hace actualmente). De esta manera al seleccionar elementos para conformar los grupos de interés, podría privilegiarse la visibilidad completa de sólo algunos objetos por sobre la eventual visión parcial de un grupo mayor. Complementariamente, al efectuar esta modificación sería posible considerar en la generación de trayectorias los radios de los cuerpos, permitiendo eventualmente un menor movimiento de la cabeza para observar el objeto en cuestión.

– En vista de que al ser removidos los faros de la liga Humanoide de la Robocup, estos desaparecerán prontamente del campo de juego, resulta de interés reemplazar tales objetos por las "T" generadas por los cruces de las líneas laterales de la cancha y la línea de medio campo (cruces que se encuentran justo en la posición actual de los faros). Este reemplazo resulta relativamente simple en el algoritmo de visión activa, bastando para ello únicamente el ajuste de los parámetros considerados para el cálculo de los ángulos de *tilt* y *pan* asociados a cada objeto, parámetros tales como la altura del objeto y altura del centro del objeto, además del ajuste de los índices de los objetos dentro del mapa local para trabajar con las mentadas "T" en lugar de los faros.

– Por último, una modificación que podría significar un cambio importante en la filosofía de funcionamiento del algoritmo de visión activa corresponde a la ampliación del conjunto de objetos posibles de observar. Actualmente este conjunto se encuentra limitado a 5 objetos de interés, de los cuales se admiten combinaciones de hasta 4 objetos simultáneamente, limitaciones impuestas por motivos de costo computacional de cálculo y análisis combinatorial de las situaciones. Por lo tanto una optimización en la implementación realizada es propuesto para generar una versión mejorada del algoritmo de selección de esquemas de observación.

5. Conclusiones.

Durante el trabajo realizado en esta memoria se pudo desarrollar una modificación al algoritmo de visión activa presentado en [1]. Gracias a esta modificación, dicho algoritmo es capaz de operar con un espacio de acciones ampliado, en cual se existe la posibilidad de observar hasta 4 objetos simultáneamente así como objetos individualmente. Esto permite al sistema obtener una mayor cantidad de información acerca del ambiente y de los objetos de interés presentes en este.

Los resultados obtenidos en las pruebas realizadas a la rutina de visión activa, no resultan concluyentes sobre la capacidad del algoritmo post-modificaciones para reducir la incerteza y mejorar la estimación de la localización de robot. De forma particular, dichos resultados no son concluyentes acerca de que esta capacidad del algoritmo se haya visto afectada de forma positiva por los cambios efectuados. No obstante, es rescatable que gracias al nuevo espacio de acciones se puede obtener una mayor cantidad de información desde el ambiente de manera simultánea, hecho que se traduce en un mayor flujo de información entrando al sistema.

Para graficar este punto, mediante un cálculo aproximado, se puede utilizar los resultados presentes en la Tabla 4.13. Los estados del 6 al 14 corresponden a la visión de más de un objeto simultáneamente, entonces de acuerdo con los porcentajes de tiempo que recibió la atención cada uno de los estados, se llega a que un 65,66% del tiempo se está obteniendo información extra.

Otro de punto abordado durante el trabajo de esta memoria fue la implementación del seguimiento por posición, funcionalidad no existente hasta el momento y que es intensivamente utilizada por la rutina de visión activa.

En este mismo punto, se pudo implementar efectivamente la capacidad para realizar el seguimiento por posición de un objeto, funcionalidad en la que además fue incluida la posibilidad para efectuar el seguimiento basado en la función de distribución de probabilidad de la estimación de la ubicación de dicho objeto e implementando además para esto un algoritmo de generación de trayectorias que permitiera recorrer los puntos representativos de cada función de distribución de probabilidad.

En conjunto a lo anterior, también se implementó la funcionalidad de adicionar objetos extras durante el seguimiento por posición. Esta modificación permitió la generación de trayectorias entre la posición de la cabeza y la posición final deseada para esta, en las cuales se podía observar objetos situados entre ambos puntos, permitiendo obtener información visual extra durante el desplazamiento de la cabeza.

Fue posible implementar un perceptor de faros, el cual es capaz de detectar simultáneamente la presencia de los dos tipos de configuración para esto objeto (es decir faros amarillo-azul-amarillo y azul-amarillo-azul). Los resultados obtenidos de las pruebas efectuadas sobre el perceptor resultaron satisfactorios, observando un alto porcentaje de clasificaciones correctas y un bajo número de clasificaciones correspondientes a falsos positivos.

Por último, se implementó exitosamente en el simulador de alto nivel, HL-Sim, una herramienta para la depuración de la información proveniente del mapa local de cada robot, la cual permite observar selectivamente la información de las posiciones de los objetos de interés o bien desplegar toda la información, contenida en el mapa local de cada robot.

Por lo tanto y en conclusión, el trabajo realizado satisface los objetivos planteados para el desarrollo de esta memoria.

6. Bibliografía.

[1] P. Guerrero, J. Ruiz-del-Solar y M. Romero. "Explicity Task Oriented Probabilistic Active Vision for a Mobile Robot". *Robocup 2008: Robot Scoccer WorldCup XII, págs. 85-96.* 2008.

[2] J. Ruiz-del-Solar, P. Guerrero, R. Palma-Amestoy, M. Arenas, R. Dodds, R. Marchant y L.A. Herrera. "UChile Kiltros, 2008 Team Description Paper". 2008.

[3] 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". 2009. (No aceptado).

[4] Robocup Technical Committee. "Robocup Standard Platform League (Nao) Rule Book". 2009.

[5] B.J.A. Krose y R. Bunschoten. "Probabilistic Localization by Appearance Models and Active Vision". *IEEE International Conference on Robotics and Automation. Proceedings, Volume 3, págs. 2255-2260.* 1999.

[6] C. Tessier, C. Debain, R. Chapuis y F. Chausse. "Active Perception Strategy for Vehicle Localisation and Guidance". *IEEE Conference on Robotics, Automation and Mechatronics, págs. 1-6*. 2006.

[7] G. Flandin y Chaumette. "Visual Data Fusion for Objects Localization by Active Vision". *Proceedings of the 7th European Conference on Computer Vision, Volume IV, págs. 312-326*. 2002.

[8] G. Welch y G. Bishop. "An Introduction to the Kalman Filter". 2006.

[9] Sitio web de Aldebaran Robotics, http://www.aldebaran-robotics.com/.

[10] Sitio web de la RoboCup, http://www.robocup.org/.

7. Anexos.

- **Anexo N°1:** Resultados prueba N°3, rutina de seguimiento de objetos.

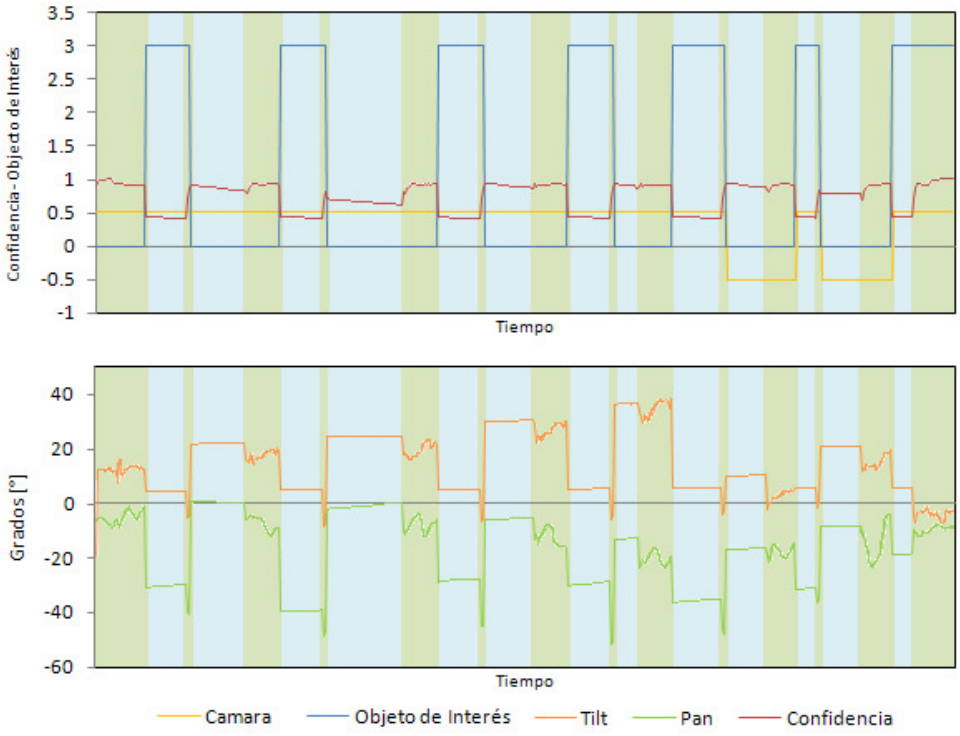


Figura 7.1: *Experimento N°2, configuración N°1.*

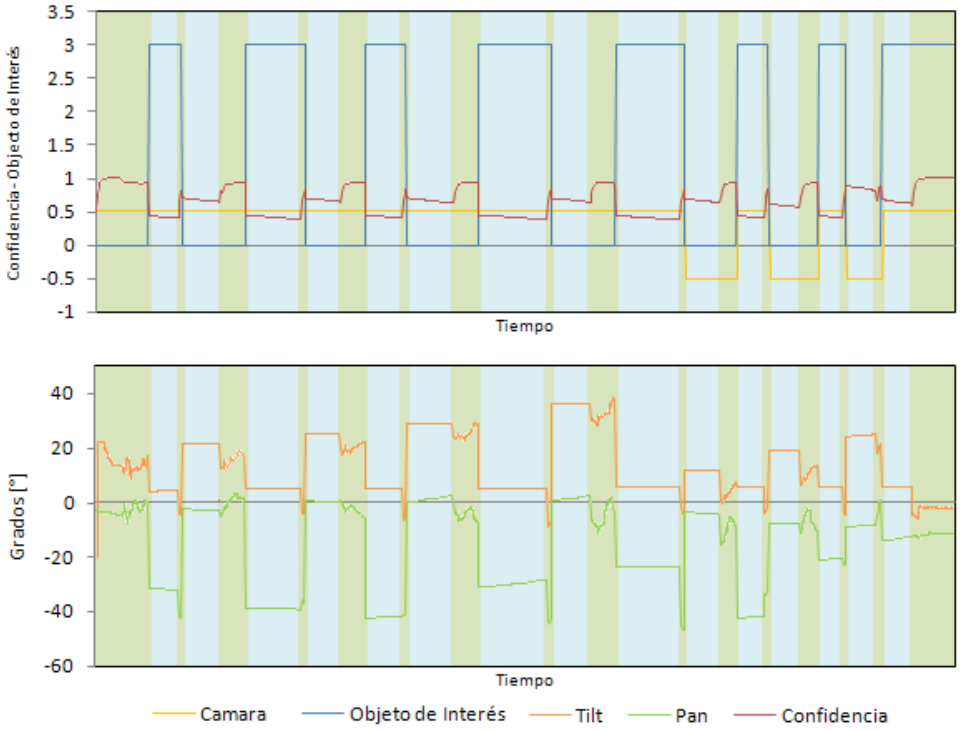


Figura 7.2: *Experimento N°3, configuración N°1.*

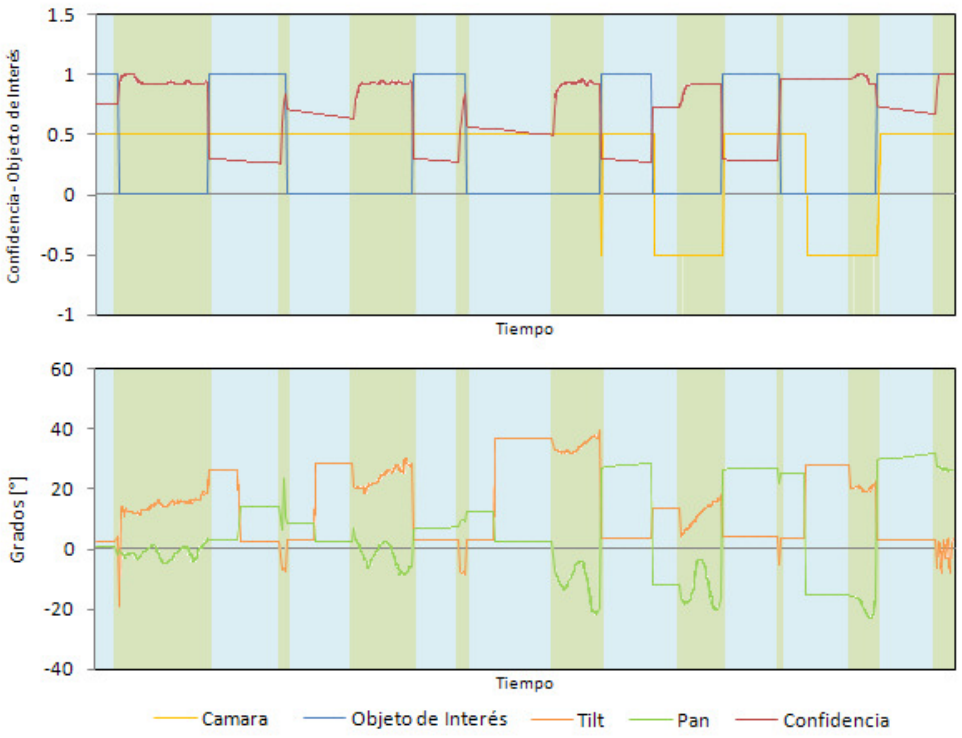


Figura 7.3: *Experimento N°1, configuración N°2.*

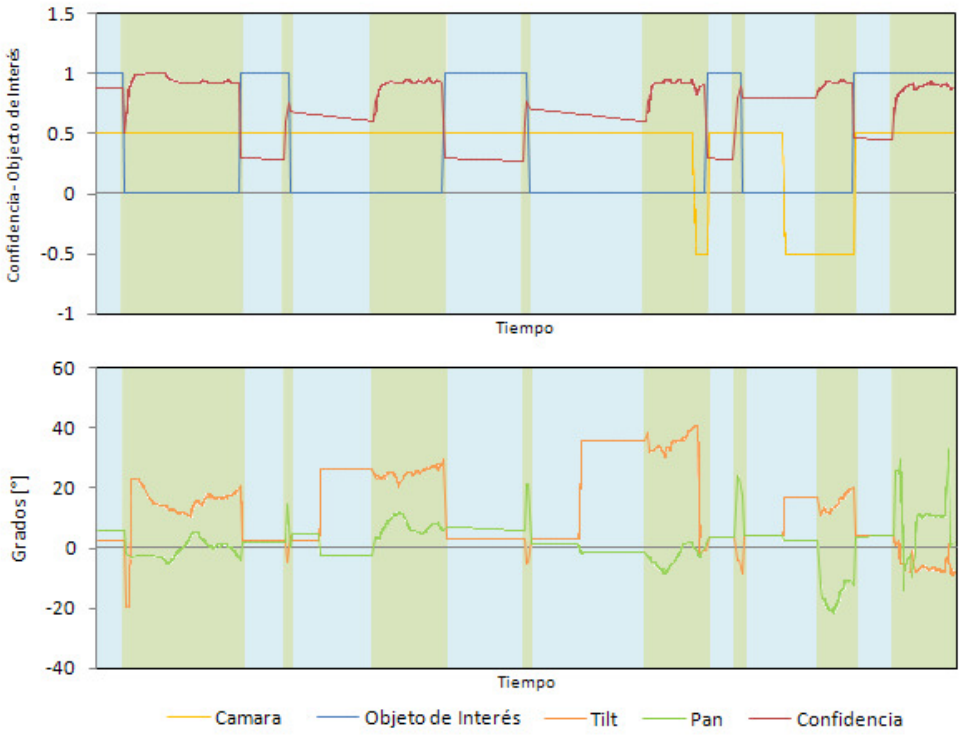


Figura 7.4: *Experimento N°2, configuración N°2.*

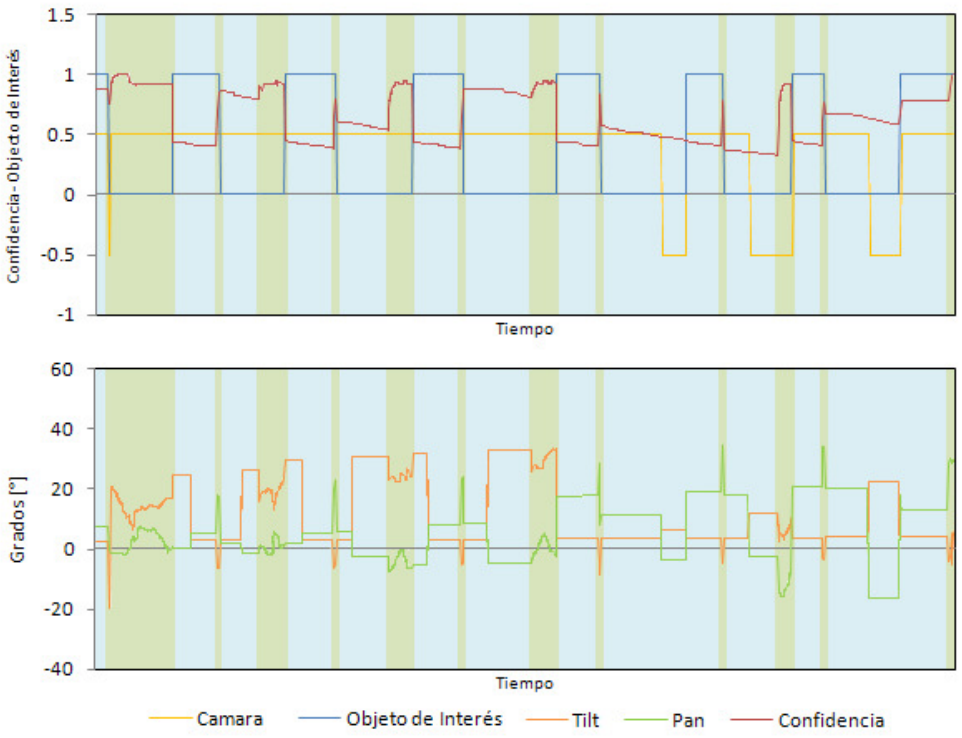


Figura 7.5: *Experimento N°3, configuración N°2.*

- **Anexo N°2:** Resultados prueba N°5, rutina de seguimiento de objetos.

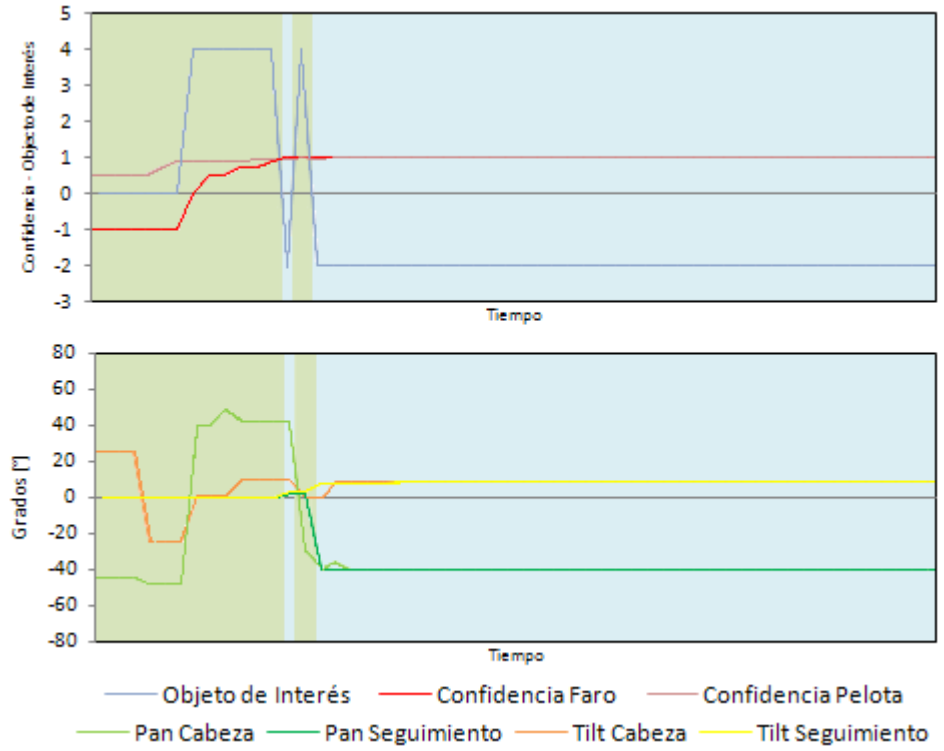


Figura 7.6: *Experimento N°2, configuración N°1, conjunto de pelota y faro.*

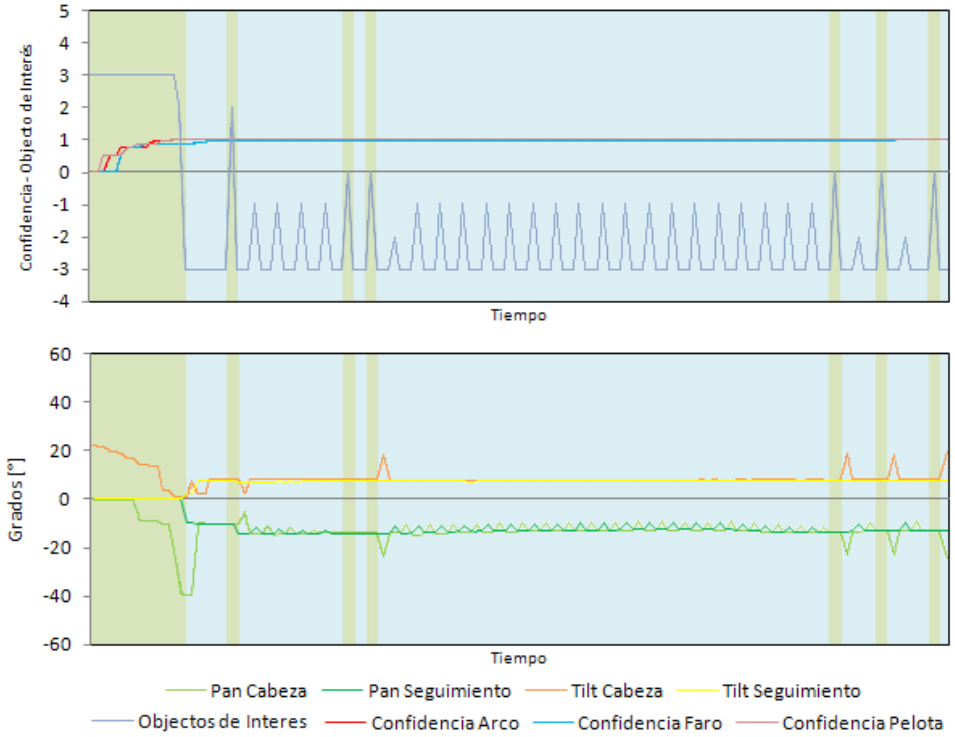


Figura 7.7: *Experimento N°1, configuración N°2, conjunto de pelota, arco y faro.*

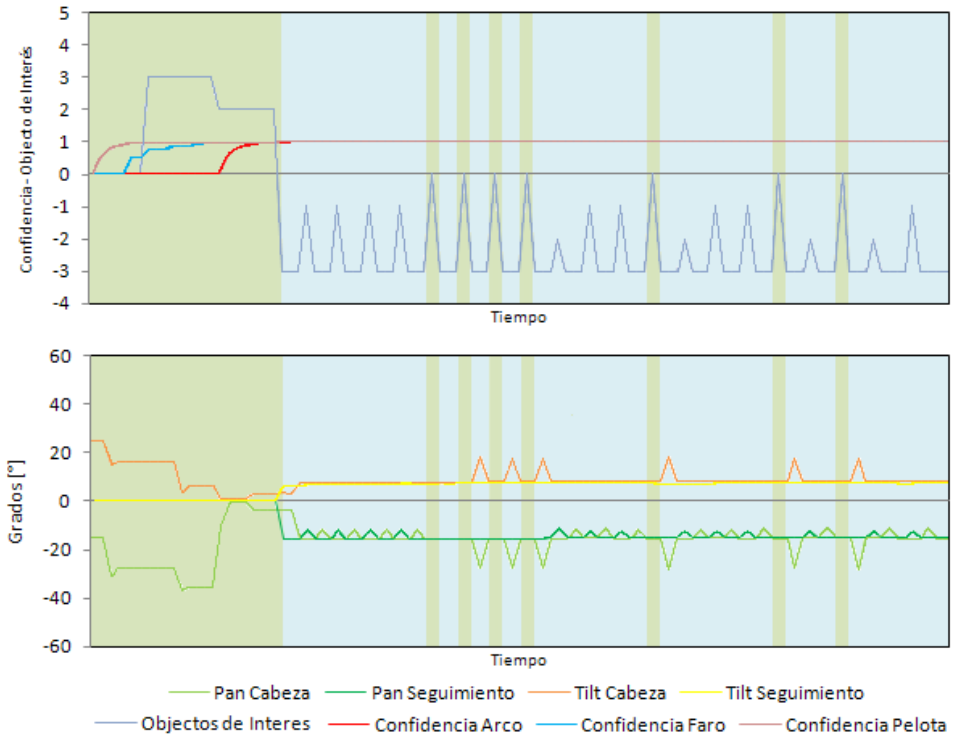


Figura 7.8: *Experimento N°2, configuración N°2, conjunto de pelota, arco y faro.*

- **Anexo N°3:** Resultados prueba N°1, rutina de visión activa.

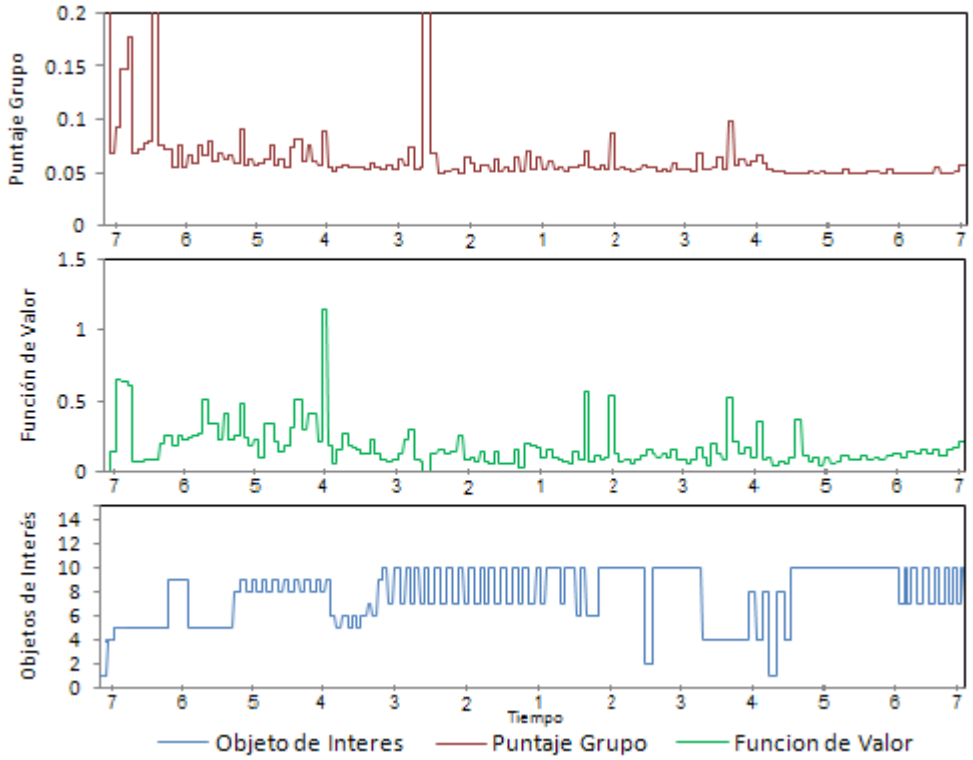


Figura 7.9: *Resultados experimento N°2.*

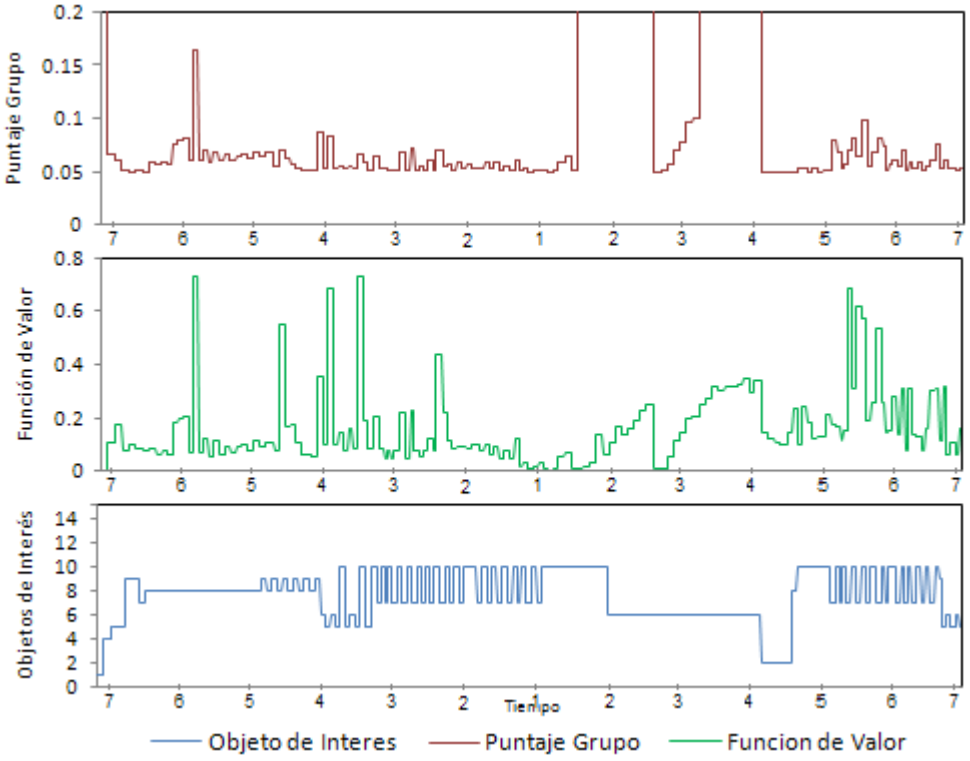


Figura 7.10: *Resultados experimento N°2.*

- **Anexo N°4:** Resultados prueba N°2, rutina de visión activa.

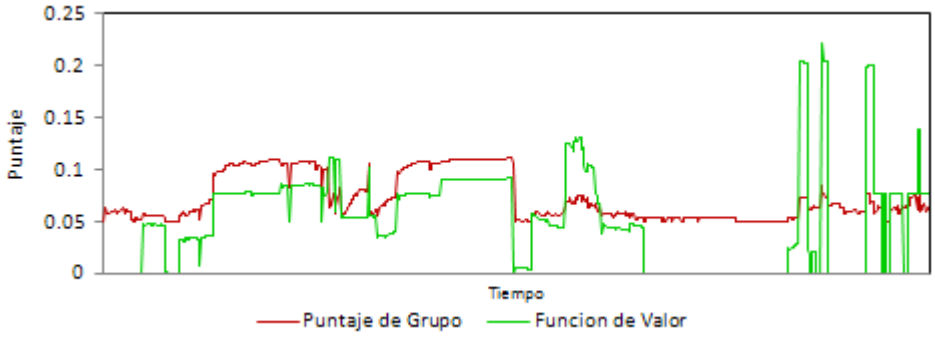


Figura 7.11: *Puntaje de grupo y función de valor, experimento N°2.*

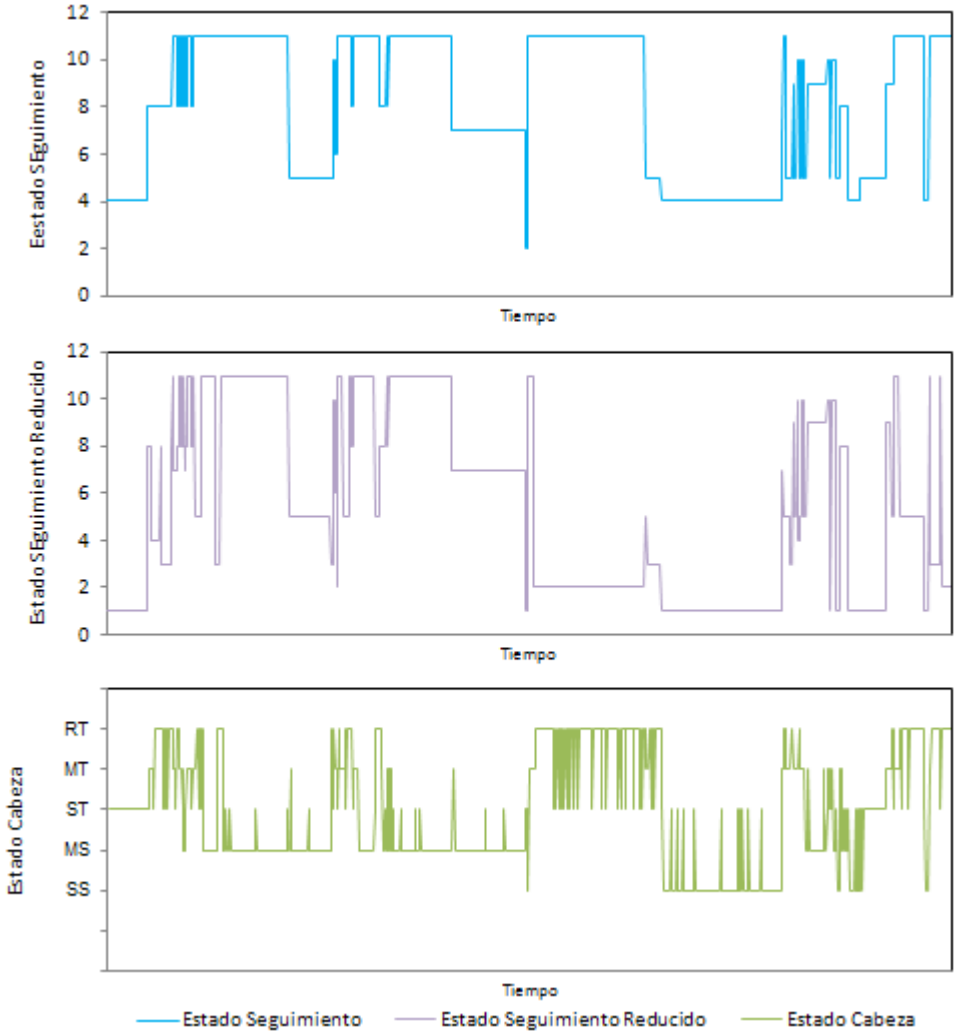


Figura 7.12: *Estado del seguimiento y de la rutina de control de la cabeza, experimento N°2.*

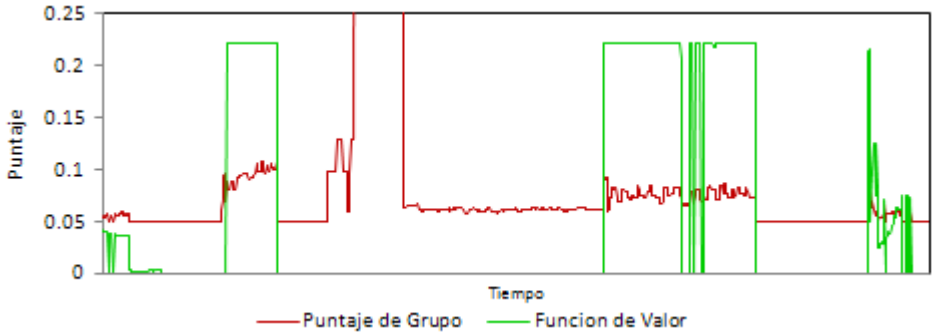


Figura 7.13: *Puntaje de grupo y función de valor, experimento N°3.*

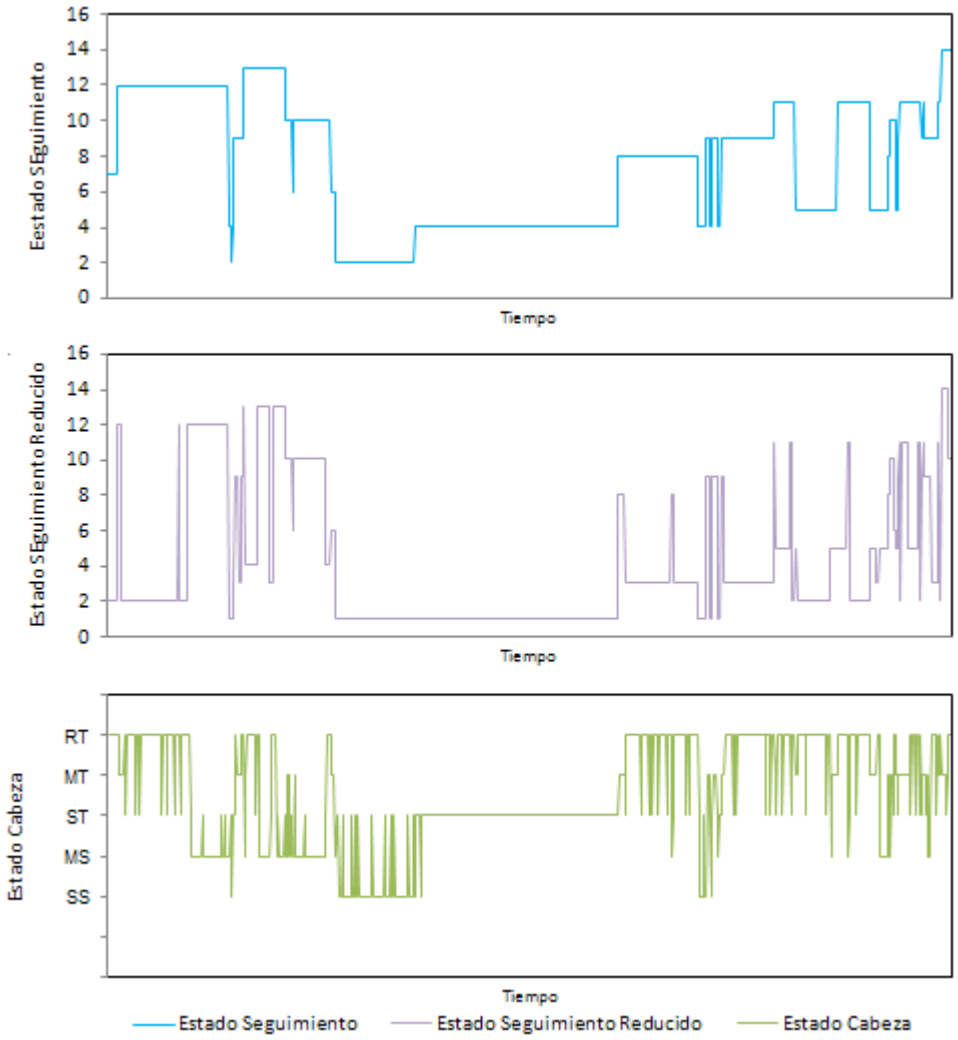


Figura 7.14: *Estado del seguimiento y de la rutina de control de la cabeza, experimento N°3.*

- **Anexo N°5:** Clases Implementadas.

UChPLBhv_ActiveVision.cpp

```
#include "UChPLBhv_ActiveVision.h"
#define WAITING_TIME 300
#define LI 537
#define HLV_Q_X_CONST 0.06
#define HLV_Q_Y_CONST 0.06
#define HLV_Q_THETA_CONST 0.00005
#define HLV_Q_X_VAR 0.3
#define HLV_Q_Y_VAR 0.3
#define HLV_Q_THETA_VAR 0.35

UChPLBhv_ActiveVision::UChPLBhv_ActiveVision(UChFrameworkDependentInformation *framework_)
: UChNewBehavior("Active Vision")
, myInfo(framework_)
, goingToHeadPosition(false)
, planTrayectory(false)
, trayectoryState(0)
{
}

UChPLBhv_ActiveVision::~UChPLBhv_ActiveVision(void)
{
}

void UChPLBhv_ActiveVision::getPDFs(UChNewExecuteBehaviorArguments arguments, UChGaussianPDF &gaussianPdf_loc,
UChGaussianPDF &gaussianPdf_ball)
{
        int ownId = arguments.info.playerIdentity.playerNumber - 1; //el id retornado comienza desde 1
    InitMask(ownId);
        gaussianPdf_loc = absSituation.SoccerSituationToGaussianPdf(mask_loc, 3);
        gaussianPdf_ball = relSituation.SoccerSituationToGaussianPdf(mask_ball, 2);
}

void UChPLBhv_ActiveVision::InitMask(int ownId)
{
        for(int i=0; i < N_RELEVANT_VALUES; ++i)
                mask[i] = mask_loc[i] = mask_ball[i] = false;

        mask[ELEMENT_SS_PARTNER1_X + 3*ownId] = mask_loc[ELEMENT_SS_PARTNER1_X + 3*ownId] = true;
        mask[ELEMENT_SS_PARTNER1_Y + 3*ownId] = mask_loc[ELEMENT_SS_PARTNER1_Y + 3*ownId] = true;
        mask[ELEMENT_SS_PARTNER1_THETA + 3*ownId] = mask_loc[ELEMENT_SS_PARTNER1_THETA + 3*ownId] = true;
        mask[ELEMENT_SS_BALL_X] = mask_ball[ELEMENT_SS_BALL_X] = true;
        mask[ELEMENT_SS_BALL_Y] = mask_ball[ELEMENT_SS_BALL_Y] = true;
}

void UChPLBhv_ActiveVision::predictionStage(UChNewExecuteBehaviorArguments &arguments, UChGaussianPDF
&gaussianPdf_loc, UChGaussianPDF &gaussianPdf_ball)
{
        /*!Lleva a cabo la etapa predictiva del filtro de Kalman.*/
        UChFrameworkDependentInformation *framework = arguments.info.worldModel.frameworkInformation;
        //localización
        UChEKF2DLocalization localization = UChEKF2DLocalization(framework-
>modulesOptions.worldModelling.localizationTrackingOptions,framework);
        framework->modulesOptions.worldModelling.localizationQParams.constQ.RepareRowReferences();
        framework->modulesOptions.worldModelling.localizationQParams.linearQDependency.RepareRowReferences();
        framework->modulesOptions.worldModelling.localizationQParams.timeVaryingQ.RepareRowReferences();
        localization.QParams = framework->modulesOptions.worldModelling.localizationQParams;
        //setear el estado de la localización de acuerdo a las pdf que llegan de WM
        UCh2DKinematicStateGaussian state_loc = UCh2DKinematicStateGaussian();
        state_loc.SetCovariance(gaussianPdf_loc.covariance,UCh2DKinematicStateRepresentation(true,true,false));
        state_loc.SetVector(gaussianPdf_loc.mean,UCh2DKinematicStateRepresentation(true,true,false));
        localization.Set2DKinematicStateGaussian(state_loc);
        //variables para inicializar el tracker de la pelota
        UChFilter *filter = (UChFilter *)&localization;
        UChObjectTypeTrackingOptions trackingOptions = framework-
>modulesOptions.worldModelling.ballTrackingOptions;
        UChLocalizationScheme scheme = LS_DIRECT_OBSERVATION;
        int objectID = (int)BALL;
        //tracker para la pelota
        UChEKF2DLocalTracker tracker =
UChEKF2DLocalTracker(objectID,filter,trackingOptions.localTracking,scheme);
        trackingOptions.QParams.constQ.RepareRowReferences();
        trackingOptions.QParams.linearQDependency.RepareRowReferences();
        trackingOptions.QParams.timeVaryingQ.RepareRowReferences();
        tracker.QParams = trackingOptions.QParams;
        //setear el estado del tracker de la pelota de acuerdo a las pdf que llegan de WM
        UCh2DKinematicStateGaussian state_ball = UCh2DKinematicStateGaussian();
        state_ball.SetCovariance(gaussianPdf_ball.covariance,UCh2DKinematicStateRepresentation(true,false,false
));
        state_ball.SetVector(gaussianPdf_ball.mean,UCh2DKinematicStateRepresentation(true,false,false));
```

```
        tracker.Set2DKinematicStateGaussian(state_ball);
        //odometría (esto es como la predicción, ie pose después de la acción a ejecutar)
        UCh2DPose policy = SimulatePolicy(arguments/*,absSituation*/);
        UChVector odom_vect = UChVector(policy);
        UChOdometry odometry = UChOdometry(odom_vect); //faltan los tiempos
        //calculo de Q y etapa predictiva de kalman
        localization.NewOdometry(odometry);
        tracker.NewOdometry(odometry);
        //asignar medias predecidas
        gaussianPdf_loc.mean = localization.GetVector(UCh2DKinematicStateRepresentation(true,true,false));
        gaussianPdf_ball.mean = tracker.GetVector(UCh2DKinematicStateRepresentation(true,false,false));
        //asignar covarianzas predecidas
        gaussianPdf_loc.covariance = localization.GetCovariance();
        gaussianPdf_ball.covariance = tracker.GetCovariance();
}

double UChPLBhv_ActiveVision::InitOptimal(void)
{
        if(TOOCType == MIN_VAR)
                return 999999;
        else
                return -999999;
}

bool UChPLBhv_ActiveVision::isBetterThan(UChTOOCType toocType, double optimal, double candidate)
{
        if(toocType == MIN_VAR && candidate < optimal)
                return true;
        if(toocType == MAX_ESP_ACTION && candidate > optimal)
                return true;
        return false;
}

double UChPLBhv_ActiveVision::calculateObjetiveFunction(UChNewExecuteBehaviorArguments arguments, UChVector
believeMean, UChSymMatrix believeCovariance, UChPointPDF pointPdfBelieve, UChMultivariateFunction
*valueFunction, UChVector &observationScheme)
{
        //variables
        double result = 0;
        int ownID = arguments.info.playerIdentity.playerNumber - 1;
        //cálculo de R y H de acuerdo al esquema de observaciones
        UChSymMatrix R_loc,R_ball;
        UChMatrix H_loc,H_ball;
        for(unsigned int i=1; i<observationScheme.getLen(); i++)
        {
                int objective = (int)observationScheme[i];
                if(objective == BALL)
                {
                R_ball.AttachBelowRight(absSituation.getSimulated_R((UChTargetObject)objective,ownID));
                        H_ball.AttachBelow(relSituation.getSimulated_H_ball());
                }
                else
                {
                        R_loc.AttachBelowRight(absSituation.getSimulated_R((UChTargetObject)objective,ownID));
                        H_loc.AttachBelow(absSituation.getSimulated_H_Loc((UChTargetObject)objective,ownID));
                }
        }
        //covarianzas
        UChSymMatrix P_pose = believeCovariance.SubSymMatrix(0,3);
        UChSymMatrix P_ball = believeCovariance.SubSymMatrix(3,2);
        //dimensión de las observaciones
        int obsDim = H_loc.Height() + H_ball.Height();
        //ganancia de Kalman
        UChMatrix K(5,obsDim);
        K.SetValues(0);
        //calculo de K
        UChSymMatrix S_ball_inv,S_loc_inv;
        if(H_ball.Height()!=0 || H_ball.Width()!=0)
        {
                S_ball_inv = (P_ball.PreAndPostTransposedMultiplied(H_ball)+R_ball).Inverse();
                //asignar valores de K
                UChMatrix aux = (P_ball*H_ball.Transposed())*S_ball_inv;
                K.SetSubMatrix(3,H_loc.Height(),aux);
                //actualización de la covarianza
                P_ball = P_ball - (aux*H_ball)*P_ball;
        }
        if(H_loc.Height()!=0 || H_loc.Width()!=0)
        {
                S_loc_inv = (P_pose.PreAndPostTransposedMultiplied(H_loc)+R_loc).Inverse();
                //asignar valores de K
                UChMatrix aux = (P_pose*H_loc.Transposed())*S_loc_inv;
                K.SetSubMatrix(0,0,aux);
                //actualización de la covarianza
                P_pose = P_pose - (aux*H_loc)*P_pose;
        }
```

```
        //unir las covarianzas de la pose es una única matriz de covarianza
        P_pose.AttachBelowRight(P_ball);
        //asignar el nuevo valor de la covarianza (correspondiente a la cov de la etapa correctiva)
        UChSymMatrix correctedCovariance = P_pose;
        //unir las covarianzas de las observaciones
        UChSymMatrix R = R_loc;
        R.AttachBelowRight(R_ball);
        //etapa observacional (cálculo de las probabilidades de observar cada objeto)
        if(onlyUseMeanOfBelief)
                //calculo utilizando sólo la media de la PDF de la creencia
                result =
observationalStage(arguments,believeMean,R,K,correctedCovariance,valueFunction,observationScheme);
        else
        {
                //calculo utilizando los puntos muestreados de la PDF de la creencia
                for(int i=0; i < pointPdfBelieve.nPoints(); ++i)
                {
                        UChVector state = pointPdfBelieve.points[i];
                        double prob = pointPdfBelieve.points[i].weight;
                        result += prob *
observationalStage(arguments,state,R,K,correctedCovariance,valueFunction,observationScheme);
                }
        }
        //salida del algoritmo
        return result;
}

double UChPLBhv_ActiveVision::correctedBelieveStage(UChNewExecuteBehaviorArguments &arguments, UChVector
estimatedState, UChVector observation, UChMatrix K, UChSymMatrix correctedCovariance, UChMultivariateFunction
*valueFunction, UChVector &observationScheme)
{
        double result = 0;
        //observación esperada (esto sería de acuerdo al modelo observacional, ie h(X_k,0) )
        UChVector ballObservation,expectedObservation;
        for(unsigned int i=1; i<observationScheme.getLen(); i++)
        {
                int objective = (int)observationScheme[i];
                if(objective == BALL)
                        ballObservation.Attach(calculate_h((UChTargetObject)objective,estimatedState));
                else
                        expectedObservation.Attach(calculate_h((UChTargetObject)objective,estimatedState));
        }
        expectedObservation.Attach(ballObservation);
        UChVector correctedBelieveMean = estimatedState + K*(observation - expectedObservation);
        UChGaussianPDF correctedBelieve;
        correctedBelieve.mean = correctedBelieveMean;
        correctedBelieve.covariance = correctedCovariance;
        //evaluación de la función de valor de la tarea del arquero
        if(TOOCType == MIN_VAR)
        {
                UChPointPDF pointPdfCorrectedBelieve =
correctedBelieve.toPointPdf((UChSamplingType)samplingType);
                //calculo de la varianza de la función de valor de la tarea
                result = pointPdfCorrectedBelieve.Covariance(valueFunction)[0][0];
        }
        else
                result = calculateExpectedActionValue(arguments,estimatedState,correctedBelieve,valueFunction);
        //salida del algoritmo
        return result;
}

double UChPLBhv_ActiveVision::observationalStage(UChNewExecuteBehaviorArguments &arguments, UChVector
estimatedState, UChSymMatrix R, UChMatrix K, UChSymMatrix correctedCovariance, UChMultivariateFunction
*valueFunction, UChVector &observationScheme)
{
        double result = 0;
        //determinar la media de la posición de las observaciones (estas son las observaciones simuladas)
        UChVector ballObservation;
        UChSymMatrix ballObservationCovariance;
        UChDynamicArray<UChVector> objectsObservation;
        UChDynamicArray<UChMatrix> observationCovariance;
        int ballObservationProcesed = 0;
        for(unsigned int i=1; i<observationScheme.getLen(); i++)
        {
                int objective = (int)observationScheme[i];
                if(objective == BALL)
                {
                        ballObservation.Attach(calculate_h((UChTargetObject)objective,estimatedState));
                        ballObservationProcesed = 1;
                }
                else
                {
                        UChVector observation = calculate_h((UChTargetObject)objective,estimatedState);
                        objectsObservation.push(observation);
                        //se extrae la matriz de cov del objeto que se está procesando
```

```
                        observationCovariance.push(R.SubMatrix(2*(i-1-ballObservationProcesed),2*(i-1-
ballObservationProcesed),2,2));
                }
        }
        //se extrae la matrix de cov de la pelota
        ballObservationCovariance = R.SubSymMatrix(2*observationCovariance.getLen(),2);
        //se realiza la etapa correctiva de Kalman con la observación simulada antes
        if(onlyUseMeanOfObservations)
        {
                //posición de las observaciones (esto sería la observación simulada)
                UChVector observationMean;
                for(unsigned int i=0; i<objectsObservation.getLen(); i++)
                        observationMean.Attach(objectsObservation[i]);
                observationMean.Attach(ballObservation);
                //calculo utilizando sólo la media de la PDF de las observaciones
                result =
correctedBelieveStage(arguments,estimatedState,observationMean,K,correctedCovariance,valueFunction,observationS
cheme);
        }
        else
        {
                UChDynamicArray<UChPointPDF> pdfArray;
                for(unsigned int i=0; i<objectsObservation.getLen(); i++)
                {
                        UChGaussianPDF observationPdf;
                        observationPdf.mean = objectsObservation[i];
                        observationPdf.covariance = (UChSymMatrix)observationCovariance[i];
                        UChPointPDF observationPointPdf =
observationPdf.toPointPdf((UChSamplingType)samplingType);
                        pdfArray.push(observationPointPdf);
                }
                //determinar la pdf de puntos de la observación de la pelota  (en caso de ver la pelota)
                UChPointPDF ballPointPdf;
                if(ballObservation.getLen() > 0)
                {
                        UChGaussianPDF observationPdf;
                        observationPdf.mean = ballObservation;
                        observationPdf.covariance = ballObservationCovariance;
                        ballPointPdf = observationPdf.toPointPdf((UChSamplingType)samplingType);
                }
                //número de las observaciones que se están considerando
                int observationNumber = observationScheme.getLen()-1;
                switch(observationNumber)
                {
                case 1:
                        //si la observación no es un objeto se agrega la pelota al arreglo
                        if(pdfArray.getLen()==0)
                                pdfArray.push(ballPointPdf);
                        //se procesa el arreglo de pdf's de las observaciones
                        for(int i=0; i < pdfArray[0].nPoints(); ++i)
                        {
                                UChVector observation = pdfArray[0].points[i];
                                double prob = pdfArray[0].points[i].weight;
                                //se lleva a cabo la última parte de la etapa correctiva de Kalman (corrección
del estado)
                                result += prob *
correctedBelieveStage(arguments,estimatedState,observation,K,correctedCovariance,valueFunction,observationSchem
e);
                        }
                        break;
                case 2:
                        //si las observaciones no son sólo objetos se agrega la pelota al arreglo
                        if(pdfArray.getLen()==1)
                                pdfArray.push(ballPointPdf);
                        //se procesa el arreglo de pdf's de las observaciones
                        for(int i=0; i<pdfArray[0].nPoints(); i++)
                        {
                                for(int j=0; j<pdfArray[1].nPoints(); j++)
                                {
                                        UChVector observation = pdfArray[0].points[i];
                                        observation.Attach(pdfArray[1].points[j]);
                                        double prob = pdfArray[0].points[i].weight *
pdfArray[1].points[j].weight;
                                        result += prob *
correctedBelieveStage(arguments,estimatedState,observation,K,correctedCovariance,valueFunction,observationSchem
e);
                                }
                        }
                        break;
                case 3:/*este caso es poco probable*/
                        //si las observaciones no son sólo objetos se agrega la pelota al arreglo
                        if(pdfArray.getLen()==2)
                                pdfArray.push(ballPointPdf);
                        //se procesa el arreglo de pdf's de las observaciones
                        for(int i=0; i<pdfArray[0].nPoints(); i++)
```

```
                        {
                                for(int j=0; j<pdfArray[1].nPoints(); j++)
                                {
                                        for(int t=0; t<pdfArray[2].nPoints(); t++)
                                        {
                                                UChVector observation = pdfArray[0].points[i];
                                                observation.Attach(pdfArray[1].points[j]);
                                                observation.Attach(pdfArray[2].points[t]);
                                                double prob = pdfArray[0].points[i].weight *
pdfArray[1].points[j].weight * pdfArray[2].points[t].weight;
                                                //se lleva a cabo la última parte de la etapa correctiva de
Kalman (corrección del estado)
                                                result += prob *
correctedBelieveStage(arguments,estimatedState,observation,K,correctedCovariance,valueFunction,observationSchem
e);
                                        }
                                }
                        }
                        break;
                case 4:/*este caso es extremadamente improbable (a no ser que se empiecen a considerar las
líneas), pero está implementado de todos modos por si las moscas.*/
                        //si las observaciones no son sólo objetos se agrega la pelota al arreglo
                        if(pdfArray.getLen()==2)
                                pdfArray.push(ballPointPdf);
                        //se procesa el arreglo de pdf's de las observaciones
                        for(int i=0; i<pdfArray[0].nPoints(); i++)
                        {
                                for(int j=0; j<pdfArray[1].nPoints(); j++)
                                {
                                        for(int t=0; t<pdfArray[2].nPoints(); t++)
                                        {
                                                for(int l=0; l<pdfArray[3].nPoints(); l++)
                                                {
                                                        UChVector observation = pdfArray[0].points[i];
                                                        observation.Attach(pdfArray[1].points[j]);
                                                        observation.Attach(pdfArray[2].points[t]);
                                                        observation.Attach(pdfArray[3].points[l]);
                                                        double prob = pdfArray[0].points[i].weight *
pdfArray[1].points[j].weight * pdfArray[2].points[t].weight * pdfArray[3].points[l].weight;
                                                        //se lleva a cabo la última parte de la etapa
correctiva de Kalman (corrección del estado)
                                                        result += prob *
correctedBelieveStage(arguments,estimatedState,observation,K,correctedCovariance,valueFunction,observationSchem
e);
                                                }
                                        }
                                }
                        }
                        break;
                default:
                        cout<<"WARNING: número de observaciones no cubierto en
UChPLBhv_ActiveVision::observationalStage."<<endl;
                        break;
                }
        }
        //salida del algoritmo
        return result;
}

double UChPLBhv_ActiveVision::calculateExpectedActionValue(UChNewExecuteBehaviorArguments &arguments, UChVector
estimatedState,UChGaussianPDF correctedBelieve, UChMultivariateFunction *valueFunction)
{
        double result = 0;
        UChSymMatrix Q(3);
        UChSoccerSituationWithUncertainty absSit;
        //media de la situacion absoluta
        UChVector absMean = correctedBelieve.mean;
        absMean[3] = absMean[3]*cos(absMean[2]) - absMean[4]*sin(absMean[2]) + absMean[0];
        absMean[4] = absMean[3]*sin(absMean[2]) + absMean[4]*cos(absMean[2]) + absMean[1];
        UChMatrix Rot;
        Rot.Rotation2x2(absMean[2]);
        //covarianza de la situación absoluta
        UChSymMatrix absCovariance = correctedBelieve.covariance;
        UChSymMatrix P_ball = correctedBelieve.covariance.SubSymMatrix(3, 2);
        absCovariance.SetSubSymMatrix(3,P_ball.PreAndPostTransposedMultiplied(Rot));
        //pdf auxiliar
        UChGaussianPDF aux;
        aux.mean = absMean;
        aux.covariance = absCovariance;
        //se pasa la pdf auxiliar a SoccerSituation
        absSit.GaussianPdfToSoccerSituation(aux,mask);
        absSit.ownID = arguments.info.playerIdentity.playerNumber;
        //se simula la siguiente acción (que sería el siguiente movimiento de cover goal)
        UCh2DPose futureAction = SimulatePolicy(arguments/*,absSit*/);
        double dx = futureAction.GetPosition().x();
```

```
        double dy = futureAction.GetPosition().y();
        double dtheta = futureAction.GetOrientation();

        for(int i=0; i < 3; ++i)
                for(int j=0; j < 3; ++j)
                        Q[i][j] = 0;
        //dejar estas variables estaticas en macros
        Q[0][0] = HLV_Q_X_CONST + fabs( dx*dx )* HLV_Q_X_VAR;
        Q[1][1] = HLV_Q_Y_CONST + fabs( dy*dy )* HLV_Q_Y_VAR;
        Q[2][2] = HLV_Q_THETA_CONST + fabs( dtheta*dtheta )* HLV_Q_THETA_VAR;

        UChGaussianPDF actions;
        UChVector actionsMean(3);
        actionsMean[0] = dx;
        actionsMean[1] = dy;
        actionsMean[2] = dtheta;
        actions.mean = actionsMean;
        actions.covariance = Q;
        UChPointPDF actionsPP = actions.toPointPdf((UChSamplingType)samplingType);
        for(int i=0; i < actionsPP.nPoints(); ++i)
        {
                UChVector aux = actionsPP.points[i];
                double prob = actionsPP.points[i].weight;
                result += prob * valueFunction->Evaluate(calculate_f(estimatedState, UCh2DPose(aux[0], aux[1],
aux[2])))[0];
        }
        return result;
}

UChVector UChPLBhv_ActiveVision::calculate_h(UChTargetObject u_sen, UChVector state)
{
        UChVector output(2);
        if(u_sen == TARGET_OBJECT_BALL)
        {
                output[0] = sqrt(state[3]*state[3] + state[4]*state[4]);
                output[1] = atan(state[4]/state[3]);
        }
        else
        {
                double Ox_sen, Oy_sen;
                switch(u_sen)
                {
                case TARGET_OBJECT_GOAL1:
                        Ox_sen = (FIELD_WIDTH-LI)/2;
                        Oy_sen = FIELD_HEIGHT/2;
                        break;
                case TARGET_OBJECT_GOAL2:
                        Ox_sen = FIELD_WIDTH - (FIELD_WIDTH-LI)/2;
                        Oy_sen = FIELD_HEIGHT/2;
                        break;
                case TARGET_OBJECT_BEACON1:
                        Ox_sen = FIELD_WIDTH/2;
                        Oy_sen = VERTICAL_DIFF;
                        break;
                case TARGET_OBJECT_BEACON2:
                        Ox_sen = FIELD_WIDTH/2;
                        Oy_sen = FIELD_HEIGHT-VERTICAL_DIFF;
                        break;
                default:
                        cout<<"WARNING: acción sensorial incorrecta en UChBhv_ActiveVision::calculate_h. El
resultado de esta función será en vector vacío."<<endl;
                        return UChVector();
                        break;
                }

                double Ox = cos(state[2])*(Ox_sen - state[0]) + sin(state[2])*(Oy_sen - state[1]);
                double Oy = -1*sin(state[2])*(Ox_sen - state[0]) + cos(state[2])*(Oy_sen - state[1]);
                output[0] = sqrt(Ox*Ox + Oy*Oy);
                output[1] = atan(Oy/Ox);
        }

        return output;
}

void UChPLBhv_ActiveVision::Execute(UChNewExecuteBehaviorArguments arguments)
{
        UChPLBhv_ActiveVision_Params *AVparams = (UChPLBhv_ActiveVision_Params *)arguments.params;
        samplingType = AVparams->samplingType;
        currentActiveVisionType = AVparams->currentActiveVisionType;
        TOOCType = AVparams->TOOCType;
        onlyUseMeanOfBelief = AVparams->onlyUseMeanOfBelief;
        onlyUseMeanOfObservations = AVparams->onlyUseMeanOfObservations;
        static int object = TARGET_OBJECT_BALL;
        static UChTime accumulatedTime = 0;
        static UChTime lastTime = GetRunningTime();
```

```
        //registro del momento actual y del tiempo acumulado entre ejecuciones
        UChTime actualTime = GetRunningTime();
        accumulatedTime += (actualTime - lastTime);
        //actualizar el último instante
        lastTime = actualTime;
        //se evalúa visión activa de acuerdo al tipo de VA que se quiere ejecutar
        if(accumulatedTime >= WAITING_TIME)
        {
                accumulatedTime = 0;
                if(currentActiveVisionType == VA_TOOC)
                        object = applyTOOC(arguments);
                else if(currentActiveVisionType == VA_MINIMUM_ENTROPY)
                        object = (int)applyMinimumEntropy(arguments);
                else if(currentActiveVisionType == VA_LOOK_BALL)
                        object = (int)applyLookBall(arguments);
                else if(currentActiveVisionType == VA_RANDOM)
                        object = (int)applyRandom(arguments);
                else
                        object = (int)applyLookBallAndLM(arguments);
        }
        //Mirar al objeto seleccionado
        if(currentActiveVisionType == VA_TOOC)
        {
                if(data.getLen()==0)
                {
                        //si no hay nada que hacer según AV se intenta trackear la pelota
                        UChLLBhv_ObjectTracking_Params *OTparams = new UChLLBhv_ObjectTracking_Params;
                        OTparams->objectId = BALL;
                        OTparams->score = 1;
                        OTparams->actionMask = canonicalMasks.wholeRobot;
                        arguments.lowLevelBehaviorsParams.push(OTparams);
                }
                else
                {
                        //encolar las opciones de trackeo
                        for(unsigned int i=0; i<data.getLen(); i++)
                        {
                                if(data[i].observationKind == OBSERVABLE_ALONE)
                                {
                                        //cout<<"observationKind: OBSERVABLE_ALONE"<<endl;
                                        UChLLBhv_ObjectTracking_Params *OTparams = new
UChLLBhv_ObjectTracking_Params;
                                        OTparams->objectId = data[i].objectId[0];
                                        OTparams->score = data[i].score;
                                        OTparams->actionMask = canonicalMasks.head;
                                        arguments.lowLevelBehaviorsParams.push(OTparams);
                                }
                                else
                                {
                                        //cout<<"observationKind: OBSERVABLE_IN_GROUP"<<endl;
                                        UChLLBhv_ObjectTracking_Params *OTparams = new
UChLLBhv_ObjectTracking_Params;
                                        getObservations(data[i],anglesScheme[i],OTparams);
                                        OTparams->multipleObjects = true;
                                        OTparams->score = data[i].score;
                                        OTparams->actionMask = canonicalMasks.head;
                                        /**/
                                        OTparams->valueFunction = data[i].valueFunction;
                                        /**/
                                        arguments.lowLevelBehaviorsParams.push(OTparams);
                                }
                        }
                }
        }
        else
        {
                //se lleva el objeto que se debe trackear a int
                int objective = GetObjectId((UChTargetObject)object,false);
                //se hace tracking del objeto
                UChLLBhv_ObjectTracking_Params *OTparams = new UChLLBhv_ObjectTracking_Params;
                OTparams->objectId = objective;
                OTparams->actionMask = canonicalMasks.head;
                arguments.lowLevelBehaviorsParams.push(OTparams);
        }
}

int UChPLBhv_ActiveVision::GetObjectId(const UChTargetObject seletedObject, bool showConsoleOutput)
{
        int output;
        switch(seletedObject)
        {
        case TARGET_OBJECT_BALL:
                output = BALL;
                if(showConsoleOutput)
                        cout<<"Objeto seleccionado: BALL"<<endl;
```

```
                break;
        case TARGET_OBJECT_GOAL1:
                output = BLUE_GOAL;
                if(showConsoleOutput)
                        cout<<"Objeto seleccionado: BLUE_GOAL"<<endl;
                break;
        case TARGET_OBJECT_GOAL2:
                output = YELLOW_GOAL;
                if(showConsoleOutput)
                        cout<<"Objeto seleccionado: YELLOW_GOAL"<<endl;
                break;
        case TARGET_OBJECT_BEACON1:
                output = trackersBeaconIndex(0);
                if(showConsoleOutput)
                        cout<<"Objeto seleccionado: BEACON 1"<<endl;
                break;
        case TARGET_OBJECT_BEACON2:
                output = trackersBeaconIndex(1);
                if(showConsoleOutput)
                        cout<<"Objeto seleccionado: BEACON 2"<<endl;
                break;
        default:
                output = BALL;
                if(showConsoleOutput)
                        cout<<"WARNING: Error en la asignación del objeto en UChBhv_ActiveVision.
Seleccionando la pelota como objetivo por defecto."<<endl;
                break;
        }
        return output;
}

int UChPLBhv_ActiveVision::applyTOOC(UChNewExecuteBehaviorArguments &arguments)
{
        //variables
        arguments.info.GetSoccerSituationWithUncertainty(absSituation);
        arguments.info.GetSoccerSituationWithUncertainty(relSituation, true);
        UChGaussianPDF predictedBelieve, gaussianPdf_loc, gaussianPdf_ball;
        UChPointPDF pointPdfBelieve;
        double actualObjetive;
        //obtener PDF's de la localización y de la pelota
        getPDFs(arguments,gaussianPdf_loc,gaussianPdf_ball);
        //función de valor
        UChGoaliePosValueFunction
goalieValueFunction(mask,arguments.actions,absSituation.ownGoal,arguments.info.worldModel.frameworkInformation,
arguments.info.playerIdentity.playerNumber,arguments.info.playerIdentity.color);
        UChMultivariateFunction *valueFunction = &goalieValueFunction;
        //etapa predictiva del algoritmo
        predictionStage(arguments,gaussianPdf_loc,gaussianPdf_ball);
        predictedBelieve = gaussianPdf_loc.join(gaussianPdf_ball);
        pointPdfBelieve = predictedBelieve.toPointPdf((UChSamplingType)samplingType);
        //se evalúa si se pueden ver varios objetos simultáneamente
        objectsPositionsToTiltAndPan(arguments,objectsAngles);
        evaluateMultipleObservations(objectsAngles,observationScheme,anglesScheme);
        //se limpia el arreglo con los datos
        data.clean();
        for(unsigned int i=0; i<observationScheme.getLen(); i++)
        {
                //se evalúa la función de valor (proceso de integración de la fn de valor)
                actualObjetive =
calculateObjetiveFunction(arguments,predictedBelieve.mean,predictedBelieve.covariance,pointPdfBelieve,valueFunc
tion,observationScheme[i]);
        actualObjetive *= 10;
                UChActiveVisionObject actualObject;
                actualObject.valueFunction = actualObjetive;
                actualObject.score = calculateScore(actualObjetive,observationScheme[i],arguments);
                actualObject.observationKind = (int)observationScheme[i][0];
                unsigned int j;
                for(j=1; j<observationScheme[i].getLen(); j++)
                        actualObject.objectId[j-1] = (int)observationScheme[i][j];
                if(j<=POSIBLE_OBSERVATIONS)
                        actualObject.objectId[j-1] = -1;
                data.push(actualObject);
        }
        //fin del algoritmo
    return 0;
}

double UChPLBhv_ActiveVision::calculateScore(double &actualScore, UChVector &actualScheme, const
UChNewExecuteBehaviorArguments &arguments)
{
    //valor actualizado del score
    double updatedScore = 0;
        bool blueIsOwnGoal = arguments.info.playerIdentity.color==BLUE_TEAM;
        UChMapData localMap = arguments.info.worldModel.localMap;
    //se evalúan todos los objetos del esquema
```

```
    for(unsigned int i=1; i<actualScheme.getLen(); i++)
    {
        //se evalúa el score proveniente de cada objeto del esquema
        switch((int)actualScheme[i])
        {
        case BALL:
            //en caso de tener la pelota completamente perdida se fuerza a verla
            if(localMap.object[BALL].confidence < 0.15)
                return 1;
            updatedScore = updatedScore + 0.35*(1-localMap.object[BALL].confidence);
            break;
        case BLUE_GOAL:
                        if(blueIsOwnGoal)
                                updatedScore = updatedScore +0.05*(1-localMap.object[BLUE_GOAL].confidence);
                        else
                                updatedScore = updatedScore +0.25*(1-localMap.object[BLUE_GOAL].confidence);
            break;
        case YELLOW_GOAL:
                        if(blueIsOwnGoal)
                                updatedScore = updatedScore + 0.25*(1-
localMap.object[YELLOW_GOAL].confidence);
                        else
                                updatedScore = updatedScore + 0.05*(1-
localMap.object[YELLOW_GOAL].confidence);
            break;
        case 3://beacon azul-amarillo-azul
            updatedScore = updatedScore + 0.175*(1-localMap.object[3].confidence);
            break;
        case 4://beacon amarillo-azul-amarillo
            updatedScore = updatedScore + 0.175*(1-localMap.object[4].confidence);
            break;
        default:
            return actualScore;
            break;
        }
    }
    //terminar de calcular el score
    updatedScore = updatedScore*actualScore*0.95 + 0.05;//el último término es para que nunca sea cero el score
    return updatedScore;
}

UCh2DPose UChPLBhv_ActiveVision::SimulatePolicy(UChNewExecuteBehaviorArguments &arguments)
{
        //parámetros de ejecución
        UChPLBhv_ActiveVision_Params *AVparams = (UChPLBhv_ActiveVision_Params *)arguments.params;
        //con esto se estima la posición del siguiente paso del robot
        UChParametricWalk paramWalk;
        paramWalk = UChParametricWalk();
        //odometría simulada
        UCh2DPose simulatedPolicy = paramWalk.BoundedDisplacement(AVparams->correctiveStagePose,false);
        return simulatedPolicy;
}

UChVector UChPLBhv_ActiveVision::calculate_f(UChVector state, UCh2DPose u_act)
{
        double dx = u_act.GetPosition().x();
        double dy = u_act.GetPosition().y();
        double dtheta = u_act.GetOrientation();
        double initialTheta = state[2];
        state[0] += dx*cos(initialTheta) - dy*sin(initialTheta);
        state[1] += dx*sin(initialTheta) + dy*cos(initialTheta);
        state[2] += dtheta;
        UCh2DPoint ballPos = UCh2DPoint(state[3], state[4], CARTESIAN2D);
        ballPos.MovedToReferenceSystem(u_act);
        state[3] = ballPos.x();
        state[4] = ballPos.y();
        return state;
}

void UChPLBhv_ActiveVision::objectsPositionsToTiltAndPan(const UChNewExecuteBehaviorArguments arguments,
UChDynamicArray<UChObjectAngles> &objectsAngles)
{
        int numberOfObjects = MAX_BALL_NUMBER + MAX_GOAL_NUMBER + MAX_BEACON_NUMBER;
        //reseteamos el arreglo con la info de los objetos
        objectsAngles.clean();
        //se pasa la posición de cada objeto a (tilt,pan) y se guardan en el arreglo dinámico
        UChFrameworkDependentInformation *framework = arguments.info.worldModel.frameworkInformation;
        for(int i=0; i<numberOfObjects; i++)
        {
            if(i==1)
            continue;
                UCh2DPoint objectPosition = arguments.info.worldModel.localMap.object[i].pose.GetPosition();
                //si vio el objeto y lo puede trackear se considera en el proceso posterior
                if(arguments.info.worldModel.localMap.object[i].confidence>0.0 &&
PointIsTrackeable(objectPosition))
```

```
                {
                        double horizontalDistance = objectPosition.r();
                        double objectCenterHeight = GetObjectCenterHeight(framework,i);
                        UChVector point = pointToTiltAndPan(arguments,objectPosition,objectCenterHeight,i);
                        //se guardan los datos de tilt y pan
                        UChObjectAngles anglesData;
                        anglesData.objectId = i;
                        anglesData.tilt = point[0];
                        anglesData.pan = point[1];
                        anglesData.cameraIndex = (int)point[2];
                        anglesData.cameraHeight = point[3];
                        anglesData.objectCenterHeight = objectCenterHeight;
                        anglesData.distance = horizontalDistance;
                        objectsAngles.push(anglesData);
                }
        }
}

UChVector UChPLBhv_ActiveVision::pointToTiltAndPan(const UChNewExecuteBehaviorArguments arguments, const
UCh2DPoint &object2DPosition, double objectCenterHeight,  int objectId)
{
        //obtenemos la altura de la cámara con la cual se puede ver el objeto y el cameraIndex asociado
        UCh2DPoint cameraData;
        if(objectId == BALL)
                cameraData = evaluateCameraChangeAndGetCameraHeight(arguments,object2DPosition.r());
        else
                cameraData = UCh2DPoint(52.54,0);
        double cameraHeight = cameraData.x();
        double cameraIndex = cameraData.y();
        //angulos de la cabeza
        double objectiveTilt = 0;
        double objectivePan = atan2(object2DPosition.y(),object2DPosition.x());
        if(cameraIndex==0)
                objectiveTilt = atan2(cameraHeight-objectCenterHeight,object2DPosition.r());
        else
        {
            double inclinationOffSet = 0.546462;//offset por la inclinación de la cámara de abajo del Nao =>
esto depende de la altura con que se esté calculando el tilt!
                objectiveTilt = (atan2(cameraHeight-objectCenterHeight,object2DPosition.r()) -
inclinationOffSet);
        }
        //se devuelve el tilt y pan calculado
        UChVector tiltAndPan(4);
        tiltAndPan[0] = objectiveTilt;
        tiltAndPan[1] = objectivePan;
        tiltAndPan[2] = cameraIndex;
        tiltAndPan[3] = cameraHeight;
        return tiltAndPan;
}
double UChPLBhv_ActiveVision::GetObjectCenterHeight(const UChFrameworkDependentInformation
*frameworkInformation, int ObjectId)
{
        //devuelve la altura del punto central del objeto
        double objectCenterHeight;
        if(ObjectId == BALL)
                objectCenterHeight = frameworkInformation->league.ballRadius;
        else if(ObjectId == BLUE_GOAL || ObjectId == YELLOW_GOAL)
                objectCenterHeight = frameworkInformation->fieldDimensions.goalHeight/2;
        else if(ObjectId == trackersBeaconIndex(0) || ObjectId == trackersBeaconIndex(1))
                objectCenterHeight = frameworkInformation->fieldDimensions.beaconBlobHigh*1.5;
        else
                objectCenterHeight = 0;//este sería el caso gral, pero además cubre las líneas
        return objectCenterHeight;
}

UCh2DPoint UChPLBhv_ActiveVision::evaluateCameraChangeAndGetCameraHeight(const UChNewExecuteBehaviorArguments
arguments, double horizontalDistance)
{
        int cameraIndex = 0;
        //evaluar como sería el cambio de cámara segun la ubicación del objeto
        int changeCamera = -1;
        if(arguments.info.worldModel.frameworkInformation->robot.camNum > 1)
        {
                if(/*arguments.*/cameraIndex == 0)
                {
                        if (horizontalDistance < 60.0)
                                changeCamera = 1;
                }
                else
                {
                        if (horizontalDistance > 80.0)
                                changeCamera = 0;
                }
                //si no fuera necesario cambiar la cámara que se quedaría con la actual
                if(changeCamera == -1)
```

```
                        changeCamera = /*arguments.*/cameraIndex;
        }
        else
                changeCamera = 0;
        //altura de la cámara respecto al suelo (primero evalúa de acuerdo al cambio de cámara y si no de
acuerdo a //la cámara que se esté usando actualmente)
        double cameraHeight = 0;
        if(changeCamera == 0)
                cameraHeight = 52.54;//camara de arriba en el NAO
        else
                cameraHeight = 48.131;//camara de abajo en el NAO
        //datos de salida
        UCh2DPoint cameraData(cameraHeight,changeCamera);
        return cameraData;
}

void UChPLBhv_ActiveVision::evaluateMultipleObservations(const UChDynamicArray<UChObjectAngles> &objectsAngles,
UChDynamicArray<UChVector> &observationScheme, UChDynamicArray<UChDynamicArray<UChObjectAngles> >
&anglesScheme)
{
        int lenght = (int)objectsAngles.getLen();
        //se genera la matriz de adyacencia
        UChSymMatrix adjacencyMatrix = generateAdjacencyMatrix(objectsAngles);
        UChSymMatrix adjacencyMatrixSquare = adjacencyMatrix*adjacencyMatrix;
        //arreglo de control
        bool *usedObject = NULL;
        usedObject = new bool[lenght];
        for(int i=0; i<lenght; i++)
                usedObject[i] = false;
        //variables para actualizar datos
        UChDynamicArray<UChVector> updatedScheme;
        UChDynamicArray<UChDynamicArray<UChObjectAngles> > updatedAnglesScheme;
        UChVector scheme;
        UChDynamicArray<UChObjectAngles> anglesData;
        UChVector adjacentObjects;
        int adjacentObject = -1;
        //se revisan los objetos que se pueden ver juntos
        for(int i=0; i<lenght; i++)
        {
                //se revisa que el objeto no haya sido marcado ya
                if(!usedObject[i])
                {
                        switch((int)adjacencyMatrixSquare[i][i])
                        {
                        /*el objeto se ve solo*/
                        case 0:
                                //esquema de objetos a observar
                                usedObject[i] = true;
                                scheme.SetSize(2);
                                scheme[0] = OBSERVABLE_ALONE;
                                scheme[1] = objectsAngles[i].objectId;
                                updatedScheme.push(scheme);
                                //angulos de los objetos
                                anglesData.clean();
                                anglesData.push(objectsAngles[i]);
                                updatedAnglesScheme.push(anglesData);
                                break;
                        /*el objeto se ve junto a otro, ie se ven 2 objetos a la vez*/
                        case 1:
                                //revisar el objeto que es adyacente con este
                                adjacentObject = (int)getAdjacentObjects(adjacencyMatrix,i)[0];
                                //si es adyacente sólo con este objeto se marcan ambos
                                if(oneObjectAdjacency(adjacencyMatrix,i,adjacentObject))
                                {
                                        usedObject[i] = true;
                                        usedObject[adjacentObject] = true;
                                }
                                //si no es único se marca sólo este
                                else
                                        usedObject[i] = true;
                                //esquema de objetos a observar
                                scheme.SetSize(3);
                                scheme[0] = OBSERVABLE_IN_GROUP;
                                scheme[1] = objectsAngles[i].objectId;
                                scheme[2] = objectsAngles[adjacentObject].objectId;
                                updatedScheme.push(scheme);
                                //angulos de los objetos
                                anglesData.clean();
                                anglesData.push(objectsAngles[i]);
                                anglesData.push(objectsAngles[adjacentObject]);
                                updatedAnglesScheme.push(anglesData);
                                break;
                        case 2:
                                //revisar los objetos que son adyacentes con este
                                adjacentObjects = getAdjacentObjects(adjacencyMatrix,i);
```

```
                                //si son todos adyacentes se marcan todos
                                if(areAdjacent(adjacencyMatrix,i,adjacentObjects))
                                {
                                        //se marcan los objetos
                                        usedObject[i] = true;
                                        usedObject[(int)adjacentObjects[0]] = true;
                                        usedObject[(int)adjacentObjects[1]] = true;
                                        //esquema de objetos a observar
                                        scheme.SetSize(4);
                                        scheme[0] = OBSERVABLE_IN_GROUP;
                                        scheme[1] = objectsAngles[i].objectId;
                                        scheme[2] = objectsAngles[(int)adjacentObjects[0]].objectId;
                                        scheme[3] = objectsAngles[(int)adjacentObjects[1]].objectId;
                                        updatedScheme.push(scheme);
                                        //angulos de los objetos
                                        anglesData.clean();
                                        anglesData.push(objectsAngles[i]);
                                        anglesData.push(objectsAngles[(int)adjacentObjects[0]]);
                                        anglesData.push(objectsAngles[(int)adjacentObjects[1]]);
                                        updatedAnglesScheme.push(anglesData);
                                }
                                //si no se marcan sólo los adyacentes
                                else
                                {
                                        for(unsigned int j=0; j<adjacentObjects.getLen(); j++)
                                        {
                                                //se revisan los objetos adyacentes al objecto actual
                                                if(!usedObject[(int)adjacentObjects[j]])
                                                {

        if(oneObjectAdjacency(adjacencyMatrix,i,(int)adjacentObjects[j]))
                                                        {
                                                                usedObject[i] = true;
                                                                usedObject[(int)adjacentObjects[j]] = true;
                                                        }
                                                        //si no es único se marca sólo este
                                                        else
                                                                usedObject[i] = true;
                                                        //esquema de objetos a observar
                                                        scheme.SetSize(3);
                                                        scheme[0] = OBSERVABLE_IN_GROUP;
                                                        scheme[1] = objectsAngles[i].objectId;
                                                        scheme[2] =
objectsAngles[(int)adjacentObjects[j]].objectId;
                                                        updatedScheme.push(scheme);
                                                        //angulos de los objetos
                                                        anglesData.clean();
                                                        anglesData.push(objectsAngles[i]);

        anglesData.push(objectsAngles[(int)adjacentObjects[j]]);
                                                        updatedAnglesScheme.push(anglesData);
                                                }
                                        }
                                }
                                break;
                        case 3:
                                adjacentObjects = getAdjacentObjects(adjacencyMatrix,i);
                                if(areAdjacent(adjacencyMatrix,i,adjacentObjects))
                                {
                                        //se marcan los objetos
                                        usedObject[i] = true;
                                        usedObject[(int)adjacentObjects[0]] = true;
                                        usedObject[(int)adjacentObjects[1]] = true;
                                        usedObject[(int)adjacentObjects[2]] = true;
                                        //esquema de objetos a observar
                                        scheme.SetSize(5);
                                        scheme[0] = OBSERVABLE_IN_GROUP;
                                        scheme[1] = objectsAngles[i].objectId;
                                        scheme[2] = objectsAngles[(int)adjacentObjects[0]].objectId;
                                        scheme[3] = objectsAngles[(int)adjacentObjects[1]].objectId;
                                        scheme[4] = objectsAngles[(int)adjacentObjects[2]].objectId;
                                        updatedScheme.push(scheme);
                                        //angulos de los objetos
                                        anglesData.clean();
                                        anglesData.push(objectsAngles[i]);
                                        anglesData.push(objectsAngles[(int)adjacentObjects[0]]);
                                        anglesData.push(objectsAngles[(int)adjacentObjects[1]]);
                                        anglesData.push(objectsAngles[(int)adjacentObjects[2]]);
                                        updatedAnglesScheme.push(anglesData);
                                }
                                //si no se simplifica y se marcan sólo los adyacentes por pares
                                else
                                {
                                        for(unsigned int j=0; j<adjacentObjects.getLen(); j++)
                                        {
```

```
                                                        //se revisan los objetos adyacentes al objecto actual
                                                        if(!usedObject[(int)adjacentObjects[j]])
                                                        {

        if(oneObjectAdjacency(adjacencyMatrix,i,(int)adjacentObjects[j]))
                                                                {
                                                                        usedObject[i] = true;
                                                                        usedObject[(int)adjacentObjects[j]] = true;
                                                                }
                                                                //si no es único si marca sólo este
                                                                else
                                                                        usedObject[i] = true;
                                                                //esquema de objetos a observar
                                                                scheme.SetSize(3);
                                                                scheme[0] = OBSERVABLE_IN_GROUP;
                                                                scheme[1] = objectsAngles[i].objectId;
                                                                scheme[2] =
objectsAngles[(int)adjacentObjects[j]].objectId;
                                                                updatedScheme.push(scheme);
                                                                //angulos de los objetos
                                                                anglesData.clean();
                                                                anglesData.push(objectsAngles[i]);

        anglesData.push(objectsAngles[(int)adjacentObjects[j]]);
                                                                updatedAnglesScheme.push(anglesData);
                                                        }
                                                }
                                        }
                                        break;
                                default:
                                        cout<<"WARNING: Caso muy poco problable no implementado en
UChBhv_ActiveVision::evaluateMultipleObservations. Terminando la ejecución de la función."<<endl;
                                        break;
                                }
                        }
        }

        //por si las moscas mato yo mismo el arreglo
        delete [] usedObject;
        usedObject = NULL;

        //se asigna el esquema de observaciones obtenido
        observationScheme = updatedScheme;
        anglesScheme = updatedAnglesScheme;
}

UChSymMatrix UChPLBhv_ActiveVision::generateAdjacencyMatrix(const UChDynamicArray<UChObjectAngles>
&objectsAngles)
{
        //umbrales del campo de vista del NAO => esto habría que leerlo desde el framework
        double tiltFOV = 0.607374;// ~ 34.8°
        double panFOV = 0.809833;// ~ 22.9148°
        double angleReduction = 0.1;// ~ 5° => esto como primer aproach para que los objetos queden
completamente dentro de la imagen

        //largo del arreglo de objetos de interés
        int lenght = (int)objectsAngles.getLen();

        //matriz de adyacencia
        UChSymMatrix adjacencyMatrix(lenght);
        adjacencyMatrix.SetValues(0);

        //comprobar que grupo de objetos puede ver a la vez
        for(int i=0; i<(lenght-1); i++)
        {
                for(int j=i+1; j<lenght; j++)
                {
                        //variables de interés
                        double tilt1 = objectsAngles[i].tilt;
                        double tilt2 = objectsAngles[j].tilt;
                        double pan1 = objectsAngles[i].pan;
                        double pan2 = objectsAngles[j].pan;
                        //evaluamos el caso en que sean observables con diferentes cámaras
                        if(objectsAngles[i].cameraIndex != objectsAngles[j].cameraIndex)
                        {
                                //altura de las cámaras
                                double cameraHeight1 = objectsAngles[i].cameraHeight;
                                double cameraHeight2 = objectsAngles[j].cameraHeight;
                                //datos de los objetos
                                double objectCenterHeight1 = objectsAngles[i].objectCenterHeight;
                                double objectCenterHeight2 = objectsAngles[j].objectCenterHeight;
                                double horizontalDistance1 = objectsAngles[i].distance;
                                double horizontalDistance2 = objectsAngles[j].distance;

                                //tilt del objeto j según la cámara con que se ve el i
```

```
                                double newTilt1 = atan2(cameraHeight2-
objectCenterHeight1,horizontalDistance1);
                                //tilt del objeto i según la cámara con que se ve el j
                                double newTilt2 = atan2(cameraHeight1-
objectCenterHeight2,horizontalDistance2);
                                //si los (tilt,pan) calculados con las mismas cámaras son observables juntos
se marcan como objetos adyacentes
                                if(abs(tilt1-newTilt2)<(tiltFOV-angleReduction) && abs(newTilt1-
tilt2)<(tiltFOV-angleReduction) && abs(pan1-pan2)<(panFOV-angleReduction))
                                        adjacencyMatrix[i][j] = 1;
                        }
                        else
                        {
                                //se actualiza la matriz de adyacencia
                                if(abs(tilt1-tilt2)<(tiltFOV-angleReduction) && abs(pan1-pan2)<(panFOV-
angleReduction))
                                        adjacencyMatrix[i][j] = 1;
                        }
                }
        }

        return adjacencyMatrix;
}

bool UChPLBhv_ActiveVision::PointIsTrackeable(UCh2DPoint relativeObjectPos)
{
        //evalúa que el punto que se quiere trackear esté hacia el frente del robot
        return fabs(BoundedAngle(relativeObjectPos.theta()))<(M_PI/2);
}

UChVector UChPLBhv_ActiveVision::getAdjacentObjects(UChSymMatrix &adjacencyMatrix, int objectId)
{
        UChVector adjacentObjects;
        //se revisa la matriz de adyacencia
        for(unsigned int i=0; i<adjacencyMatrix.Width(); i++)
        {
                if(adjacencyMatrix[objectId][i] == 1)
                        adjacentObjects.push(i);
        }
        return adjacentObjects;
}

bool UChPLBhv_ActiveVision::areAdjacent(UChSymMatrix &adjacencyMatrix, int objectId, UChVector
&adjacentObjects) const
{
        unsigned int lenght = adjacentObjects.getLen()+1;
        UChVector auxiliarArray = adjacentObjects;
        auxiliarArray.push(objectId);
        //se revisa si son adyacentes todos con todos
        bool isAdjacent = true;
        for(unsigned int i=0; i<lenght; i++)
        {
                for(unsigned int j=i+1; j<lenght; j++)
                {
                        int id1 = (int)auxiliarArray[i];
                        int id2 = (int)auxiliarArray[j];
                        isAdjacent = (isAdjacent && adjacencyMatrix[id1][id2]==1);
                        //si hay alguno que es false se detiene la comprobación y se devuelve false
                        if(!isAdjacent)
                                return false;
                }
        }

        return true;
}

bool UChPLBhv_ActiveVision::oneObjectAdjacency(UChSymMatrix &adjacencyMatrix, int actualObjectId, int
adjacentObjectId)
{
        int counter = 0;
        //se recorre la fila correspondiente de la matriz de adyacencia
        for(unsigned int i=0; i<adjacencyMatrix.Width(); i++)
        {
                if(adjacencyMatrix[adjacentObjectId][i]==1)
                        counter++;
                if(counter > 1)
                        return false;
        }
        return true;
}

UChVector UChPLBhv_ActiveVision::calculateHeadPosition(UChDynamicArray<UChObjectAngles> &objectiveAngles)
{
        double maxPan = -1000;
        double minPan = 1000;
```

```
        double maxTilt = -1000;
        double minTilt = 1000;

        //cameraIndex de la cam con la que se ven los objetos del esquema
        int cameraIndex = getCameraIndex(objectiveAngles);

        for(unsigned int i=0; i<objectiveAngles.getLen(); i++)
        {
                double tilt = objectiveAngles[i].tilt;
                double pan = objectiveAngles[i].pan;
                //evaluamos el pan
                if(pan > maxPan)
                        maxPan = pan;
                if(pan < minPan)
                        minPan = pan;
                // evaluamos el tilt
                if(tilt > maxTilt)
                        maxTilt = tilt;
                if(tilt < minTilt)
                        minTilt = tilt;
        }
        //variaciones de los ángulos
        double deltaPan = abs(maxPan-minPan);
        double deltaTilt = abs(maxTilt-minTilt);
        //posición objetivo de la cabeza
        UChVector finalAngles(3);
        finalAngles[0] = minTilt+(deltaTilt*0.5);
        finalAngles[1] = minPan+(deltaPan*0.5);
        finalAngles[2] = cameraIndex;

        return finalAngles;
}

int UChPLBhv_ActiveVision::getCameraIndex(UChDynamicArray<UChObjectAngles> &objetiveAngles)
{
        int cam0,cam1,objectId;
        double cameraHeight;
        unsigned int objects = objetiveAngles.getLen();
        switch(objects)
        {
        case 1:
                return objetiveAngles[0].cameraIndex;
                break;
        case 2:
                //se comparan los cameraIndex de los ángulos de cada vector
                if(objetiveAngles[0].cameraIndex == objetiveAngles[1].cameraIndex)
                        return objetiveAngles[0].cameraIndex;
                else
                {
                        //caso en que el 1° objeto tiene cam 0 el 2° cam 2
                        if(objetiveAngles[0].cameraIndex == 0)
                        {
                                bool cond1 = (0.5811 < objetiveAngles[0].tilt);
                                bool cond2 = (objetiveAngles[1].tilt < 0.6761);
        if(cond1 && cond2)//está en el intervalo que se ve con ambas cam
                                {
                                        //se asigna porque arbitrariamente una cámara
                                        double newTilt = atan2(objetiveAngles[0].cameraHeight -
objetiveAngles[1].objectCenterHeight,objetiveAngles[1].distance);
                                        objetiveAngles[1].tilt = newTilt;
                                        objetiveAngles[1].cameraIndex = objetiveAngles[0].cameraIndex;
                                        return objetiveAngles[0].cameraIndex;
                                }
                                else if(!cond1)//theta0<theta0** => se usa cam 0
                                {
                                        //se asigna la nueva cámara
                                        double newTilt = atan2(objetiveAngles[0].cameraHeight -
objetiveAngles[1].objectCenterHeight,objetiveAngles[1].distance);
                                        objetiveAngles[1].tilt = newTilt;
                                        objetiveAngles[1].cameraIndex = objetiveAngles[0].cameraIndex;
                                        return objetiveAngles[0].cameraIndex;
                                }
                                else if(!cond2)//theta1>theta1** => se usa cam 1
                                {
                                        //se asigna la nueva cámara
                                        double newTilt = atan2(objetiveAngles[1].cameraHeight -
objetiveAngles[0].objectCenterHeight,objetiveAngles[0].distance);
                                        objetiveAngles[0].tilt = newTilt;
                                        objetiveAngles[0].cameraIndex = objetiveAngles[1].cameraIndex;
                                        return objetiveAngles[1].cameraIndex;
                                }
                                else
                                {
                                        cout<<"Entró donde no debía!:
UChBhv_ActiveVision::getCameraIndex."<<endl;
```

```
                                        return -1;
                                }
                        }
                        //caso en que el 2° objeto tiene cam 0 el 1° cam 2
                        else
                        {
                                //estos intervalos salen de considerar los cambios de cámaras en las mismas
distancias que se usan en objectTracking
                                bool cond1 = (0.5811 < objetiveAngles[1].tilt); //=> esto es (theta0 >
theta0**)
                                bool cond2 = (objetiveAngles[0].tilt < 0.6761); //=> esto es (theta1 <
theta1**)
                                if(cond1 && cond2)//está en el intervalo que se ve con ambas cam
                                {
                                        //se asigna porque arbitrariamente una cámara
                                        double newTilt = atan2(objetiveAngles[1].cameraHeight -
objetiveAngles[0].objectCenterHeight,objetiveAngles[0].distance);
                                        objetiveAngles[0].tilt = newTilt;
                                        objetiveAngles[0].cameraIndex = objetiveAngles[1].cameraIndex;
                                        return objetiveAngles[1].cameraIndex;
                                }
                                else if(!cond1)//theta0<theta0** => se usa cam 0
                                {
                                        //se asigna la nueva cámara
                                        double newTilt = atan2(objetiveAngles[1].cameraHeight -
objetiveAngles[0].objectCenterHeight,objetiveAngles[0].distance);
                                        objetiveAngles[0].tilt = newTilt;
                                        objetiveAngles[0].cameraIndex = objetiveAngles[1].cameraIndex;
                                        return objetiveAngles[1].cameraIndex;
                                }
                                else if(!cond2)//theta1>theta1** => se usa cam 1
                                {
                                        //se asigna la nueva cámara
                                        double newTilt = atan2(objetiveAngles[0].cameraHeight -
objetiveAngles[1].objectCenterHeight,objetiveAngles[1].distance);
                                        objetiveAngles[1].tilt = newTilt;
                                        objetiveAngles[1].cameraIndex = objetiveAngles[0].cameraIndex;
                                        return objetiveAngles[0].cameraIndex;
                                }
                                else
                                {
                                        cout<<"Entró donde no debía!:
UChBhv_ActiveVision::getCameraIndex."<<endl;
                                        return -1;
                                }
                        }
                }
                break;
        case 3:
                //se busca qué cámara tiene más objetos asociados
                cam0 = 0;
                cam1 = 0;
                for(unsigned int i=0; i<objects; i++)
                {
                        if(objetiveAngles[i].cameraIndex == 0)
                        {
                                cam0++;
                                //si suma 3 esta es la cámara correcta para el grupo
                                if(cam0 == 3)
                                        return 0;
                        }
                        else
                        {
                                cam1++;
                                //si suma 3 esta es la cámara correcta para el grupo
                                if(cam1 == 3)
                                        return 1;
                        }
                }
                //se corrige el objeto con ángulos asociados a una cámara diferente a la del grupo
                objectId = -1;
                cameraHeight = 0;
                if(cam0 == 1)//ie, cameraIndex=0 no es la correcta
                {
                        //se busca el objeto
                        for(unsigned int i=0; i<objects; i++)
                        {
                                if(objetiveAngles[i].cameraIndex == 0)
                                {
                                        objectId = i;
                                        break;
                                }
                                else
                                        cameraHeight = objetiveAngles[i].cameraHeight;
                        }
```

```
                        //se corrigen los ángulos del objeto para que sean los asociados a la cámara 1
                        double newTilt = atan2(cameraHeight-
objetiveAngles[objectId].objectCenterHeight,objetiveAngles[objectId].distance);
                        objetiveAngles[objectId].tilt = newTilt;
                        objetiveAngles[objectId].cameraIndex = 1;
                        //cameraIndex del grupo
                        return 1;
                }
                else//ie, cameraIndex=1 no es la correcta
                {
                        //se busca el objeto
                        for(unsigned int i=0; i<objects; i++)
                        {
                                if(objetiveAngles[i].cameraIndex == 1)
                                {
                                        objectId = i;
                                        break;
                                }
                                else
                                        cameraHeight = objetiveAngles[i].cameraHeight;
                        }
                        //se corrigen los ángulos del objeto para que sean los asociados a la cámara 1
                        double newTilt = atan2(cameraHeight-
objetiveAngles[objectId].objectCenterHeight,objetiveAngles[objectId].distance);
                        objetiveAngles[objectId].tilt = newTilt;
                        //cameraIndex del grupo
                        return 0;
                }
                break;
        case 4:
                //se busca qué cámara tiene más objetos asociados
                cam0 = 0;
                cam1 = 0;
                for(unsigned int i=0; i<objects; i++)
                {
                        if(objetiveAngles[i].cameraIndex == 0)
                        {
                                cam0++;
                                //si suma 4 esta es la cámara correcta para el grupo
                                if(cam0 == 4)
                                        return 0;
                        }
                        else
                        {
                                cam1++;
                                //si suma 4 esta es la cámara correcta para el grupo
                                if(cam1 == 4)
                                        return 1;
                        }
                }
                //se corrige el objeto con ángulos asociados a una cámara diferente a la del grupo
                objectId = -1;
                cameraHeight = 0;
                if(cam0 == 1)//ie, cameraIndex=0 no es la correcta
                {
                        //se busca el objeto
                        for(unsigned int i=0; i<objects; i++)
                        {
                                if(objetiveAngles[i].cameraIndex == 0)
                                {
                                        objectId = i;
                                        break;
                                }
                                else
                                        cameraHeight = objetiveAngles[i].cameraHeight;
                        }
                        //se corrigen los ángulos del objeto para que sean los asociados a la cámara 1
                        double newTilt = atan2(cameraHeight-
objetiveAngles[objectId].objectCenterHeight,objetiveAngles[objectId].distance);
                        objetiveAngles[objectId].tilt = newTilt;
                        objetiveAngles[objectId].cameraIndex = 1;
                        //cameraIndex del grupo
                        return 1;
                }
                else if(cam1 == 1)//ie, cameraIndex=1 no es la correcta
                {
                        //se busca el objeto
                        for(unsigned int i=0; i<objects; i++)
                        {
                                if(objetiveAngles[i].cameraIndex == 1)
                                {
                                        objectId = i;
                                        break;
                                }
                                else
```

```
                                        cameraHeight = objetiveAngles[i].cameraHeight;
                        }
                        //se corrigen los ángulos del objeto para que sean los asociados a la cámara 1
                        double newTilt = atan2(cameraHeight-
objetiveAngles[objectId].objectCenterHeight,objetiveAngles[objectId].distance);
                        objetiveAngles[objectId].tilt = newTilt;
                        //cameraIndex del grupo
                        return 0;
                }
                //caso en que hay 2 de cada cámara
                else if(cam0 == cam1)
                {
                        int objectId1 = -1;
                        int objectId2 = -1;
                        //se buscan los objetos
                        for(unsigned int i=0; i<objects; i++)
                        {
                                if(objectId1 == -1)
                                        if(objetiveAngles[i].cameraIndex == 1)
                                                objectId1 = i;
                                        else
                                                cameraHeight = objetiveAngles[i].cameraHeight;
                                else
                                        if(objetiveAngles[i].cameraIndex == 1)
                                                objectId2 = i;
                                        else
                                                cameraHeight = objetiveAngles[i].cameraHeight;
                        }
                        //se corrigen los ángulos del objeto para que sean los asociados a la cámara 1
                        double newTilt1 = atan2(cameraHeight-
objetiveAngles[objectId1].objectCenterHeight,objetiveAngles[objectId1].distance);
                        double newTilt2 = atan2(cameraHeight-
objetiveAngles[objectId2].objectCenterHeight,objetiveAngles[objectId2].distance);
                        objetiveAngles[objectId1].tilt = newTilt1;
                        objetiveAngles[objectId1].cameraIndex = 0;
                        objetiveAngles[objectId2].tilt = newTilt2;
                        objetiveAngles[objectId2].cameraIndex = 0;
                        //cameraIndex del grupo
                        return 0;
                }
                break;
        }

        //si no es ninguno
        cout<<"WARNING: arreglo de dimensiones no consideradas en UChBhv_ActiveVision::getCameraIndex.
Asignando cámara 0."<<endl;
        return 0;
}

void UChPLBhv_ActiveVision::getObservations(UChActiveVisionObject &data, UChDynamicArray<UChObjectAngles>
&objectiveAngles, UChLLBhv_ObjectTracking_Params *OTparams)
{
        int i = 0;
        for(i=0; i<POSIBLE_OBSERVATIONS && data.objectId[i]!=-1; i++)
        {
                if(data.objectId[i]!=objectiveAngles[i].objectId)
                {
                        cout<<"WARNING: diferencia en id's en UChPLBhv_ActiveVision::getObservations. Saliendo
de la funcion."<<endl;
                        return;
                }
                OTparams->observations.objectsId[i] = data.objectId[i];
                OTparams->angles[i] = objectiveAngles[i];
        }
        if(i<POSIBLE_OBSERVATIONS)
                OTparams->observations.objectsId[i] = -1;
        UChVector headPosition = calculateHeadPosition(objectiveAngles);
        double tilt = headPosition[0];
        double pan = headPosition[1];

        double cameraHeight = 0;
        if(headPosition[2] == 0)
                cameraHeight = 52.54;//camara de arriba en el NAO
        else
                cameraHeight = 48.131;//camara de abajo en el NAO
        double aux1 = cameraHeight/abs( tan(tilt) );
        double aux2 = 1/sqrt( 1 + tan(pan)*tan(pan) );
        double fictitiousX = aux1 * aux2;
        double fictitiousY = aux1 * aux2 * tan(pan);
        OTparams->observations.trackingPosition = UCh2DPoint(fictitiousX,fictitiousY);
}
```

UChPLBhv_ActiveVision.h

```
#ifndef _UCH_PLAYER_LEVEL_BEHAVIOR_ACTIVE_VISION_H_
#define _UCH_PLAYER_LEVEL_BEHAVIOR_ACTIVE_VISION_H_
#include "../../Common_New_Decision_Making/Behaviors/UChNewBehavior.h"
#include "../../Common_New_Decision_Making/UChPlayerLevelBehaviorExecution.h"
#include "../../../Libraries/Probability/UChGaussianPDF.h"
#include "../../../Libraries/Multivariate_Functions/UChMultivariateFunction.h"
#include "../../../Libraries/Multivariate_Functions/UChGoaliePosValueFunction.h"
#include "../../Actuation/Movements/UChParametricWalk.h"
#include "../../../Libraries/General/UChTimeLib.h"
#include "../../World_Modelling/Localization/UChEKF2DLocalization.h"
#include "../../World_Modelling/Local_Tracking/UChEKF2DLocalTracker.h"
#include "../../../Libraries/State_Estimation/UChFilter.h"
#include "../../../Libraries/Information/Communication_Data/UCh2DKinematicStateGaussian.h"
#include "../../Actuation/Movements/UChHeadMovement.h"
#include "../../Low_Level_Decision_Making/Behaviors/Parameters/UChLLBhv_ObjectTracking_Params.h"
#include "Parameters/UChPLBhv_ActiveVision_Params.h"
#include "Parameters/UChBhv_ObjectSearching_Params.h"
class UChPLBhv_ActiveVision :
        public UChNewBehavior
{
public:
        UChPLBhv_ActiveVision(UChFrameworkDependentInformation *framework_);
        virtual ~UChPLBhv_ActiveVision(void);
        virtual void Execute(UChLowLevelExecuteBehaviorArguments arguments){};
        virtual void Execute(UChNewExecuteBehaviorArguments arguments);
        void getPDFs(UChNewExecuteBehaviorArguments arguments, UChGaussianPDF &gaussianPdf_loc, UChGaussianPDF
&gaussianPdf_ball);
        void InitMask(int ownId);
        void predictionStage(UChNewExecuteBehaviorArguments &arguments, UChGaussianPDF &gaussianPdf_loc,
UChGaussianPDF &gaussianPdf_ball);
        bool isBetterThan(UChTOOCType toocType, double optimal, double candidate);
        double InitOptimal(void);
        double calculateObjetiveFunction(UChNewExecuteBehaviorArguments arguments, UChVector believeMean,
UChSymMatrix believeCovariance, UChPointPDF pointPdfBelieve, UChMultivariateFunction *valueFunction, UChVector
&observationScheme);
        double correctedBelieveStage(UChNewExecuteBehaviorArguments &arguments, UChVector estimatedState,
UChVector observation, UChMatrix K, UChSymMatrix correctedCovariance, UChMultivariateFunction *valueFunction,
UChVector &observationScheme);
        double observationalStage(UChNewExecuteBehaviorArguments &arguments, UChVector estimatedState,
UChSymMatrix R, UChMatrix K, UChSymMatrix correctedCovariance, UChMultivariateFunction *valueFunction,
UChVector &observationScheme);
        double calculateExpectedActionValue(UChNewExecuteBehaviorArguments &arguments, UChVector
estimatedState, UChGaussianPDF correctedBelieve, UChMultivariateFunction *valueFunction);
        UChVector calculate_h(UChTargetObject u_sen, UChVector state);
        UChVector calculate_f(UChVector state, UCh2DPose u_act);
        int GetObjectId(const UChTargetObject object, bool showConsoleOutput = false);
        int applyTOOC(UChNewExecuteBehaviorArguments &arguments);
        UChTargetObject applyMinimumEntropy(UChNewExecuteBehaviorArguments &arguments);
        UChTargetObject applyLookBall(UChNewExecuteBehaviorArguments &arguments);
        UChTargetObject applyRandom(UChNewExecuteBehaviorArguments &arguments);
        UChTargetObject applyLookBallAndLM(UChNewExecuteBehaviorArguments &arguments);
        UCh2DPose SimulatePolicy(UChNewExecuteBehaviorArguments &arguments);
        static int getCameraIndex(UChDynamicArray<UChObjectAngles> &objetiveAngles);
        static double GetObjectCenterHeight(const UChFrameworkDependentInformation *frameworkInformation, int
ObjectId);
        double calculateScore(double &actualScore, UChVector &actualScheme, const
UChNewExecuteBehaviorArguments &arguments);
        static void objectsPositionsToTiltAndPan(const UChNewExecuteBehaviorArguments arguments,
UChDynamicArray<UChObjectAngles> &objectsAngles);
        void evaluateMultipleObservations(const UChDynamicArray<UChObjectAngles> &objectsAngles,
UChDynamicArray<UChVector> &observationScheme, UChDynamicArray<UChDynamicArray<UChObjectAngles> >
&anglesScheme);
        void getObservations(UChActiveVisionObject &data, UChDynamicArray<UChObjectAngles> &objectiveAngles,
UChLLBhv_ObjectTracking_Params *OTparams);
        static bool PointIsTrackeable(UCh2DPoint relativeObjectPos);
        bool areAdjacent(UChSymMatrix &adjacencyMatrix, int objectId, UChVector &adjacentObjects) const;
        bool oneObjectAdjacency(UChSymMatrix &adjacencyMatrix, int actualObjectId, int adjacentObjectId);
        static UCh2DPoint evaluateCameraChangeAndGetCameraHeight(const UChNewExecuteBehaviorArguments
arguments, double horizontalDistance);
        static UChVector pointToTiltAndPan(const UChNewExecuteBehaviorArguments arguments, const UCh2DPoint
&object2DPosition, double objectCenterHeight,  int objectId);
        static UChVector calculateHeadPosition(UChDynamicArray<UChObjectAngles> &objectiveAngles);
        UChVector getAdjacentObjects(UChSymMatrix &adjacencyMatrix, int objectId);
        UChSymMatrix generateAdjacencyMatrix(const UChDynamicArray<UChObjectAngles> &objectsAngles);
        UChDynamicArray<UChVector> observationScheme;
        UChDynamicArray<UChDynamicArray<UChObjectAngles> > anglesScheme;
        UChDynamicArray<UChObjectAngles> objectsAngles;
        UChVector nextHeadPosition;
        UChDynamicArray<UChVector> headTrayectory;
        bool goingToHeadPosition,planTrayectory;
        unsigned int trayectoryState;
```

```
        UChDynamicArray<UChActiveVisionObject> data;
        bool mask[N_RELEVANT_VALUES], mask_loc[N_RELEVANT_VALUES], mask_ball[N_RELEVANT_VALUES];
        UChSoccerSituationWithUncertainty absSituation;
        UChSoccerSituationWithUncertainty relSituation;
        UChDynamicArray<UChActionCommand> myActionCommands;
        UChDecisionInformation myInfo;
        int samplingType;
        int currentActiveVisionType;
        int TOOCType;
        bool onlyUseMeanOfBelief;
        bool onlyUseMeanOfObservations;
private:
        UChPLBhv_ActiveVision(void);
};
#endif
```