Informe Final
Proyecto de grado Visión de Robots

Gonzalo Gismero
gegismero@gmail.com

Tutores
Facundo Benavides, Gonzalo Tejera, Serrana Casella

http://www.fing.edu.uy/~pgvisrob

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

6 de septiembre de 2012

Agradecimientos

Muchas experiencias nuevas fueron surgiendo a lo largo del proyecto, muchos días de investigación y otras tantas tardes de prueba y error con el hardware en el taller de robótica. Luego de casi 2 años y medio de proyecto, es conmovedor ver el fruto del esfuerzo realizado, y me llevo el mejor recuerdo del camino transitado.

Por todo esto, es imperativo agradecer a las personas que me acompañaron a lo largo de este proceso.

A Gonzalo Tejera, Facundo Benavides y Serrana Casella, los tutores de este proyecto, por su guía y aporte de ideas durante el transcurso del mismo.

A Marcelo Baliero y Gerardo Pias integrantes del proyecto de grado Salimoo, quienes desarrollan actualmente el módulo de estrategia para un equipo de fútbol de robots y usuarios del módulo realizado por este proyecto, por sus charlas en el taller, su inmenso grano de arena a la hora de definir datos retornados por el módulo y por brindar las librerías generadas para la locomoción del robot, permitiendo probar escenarios más reales.

A Hamid Mobalegh, creador del módulo HaViMo, por la pronta respuesta a dudas sobre el módulo de hardware utilizado.

A Guillermo Reisch y Federico Andrade por la ayuda brindada frente a problemas con el adaptador Zig2serial y motores AX-12; y a Martín Llofriu por la introducción a *Probabilistic Robotics*.

Por último, agradecer y dedicar este trabajo a mi familia por brindar el apoyo incondicional durante la formación académica y especialmente en el transcurso de este proyecto, sin el cual hubiera sido imposible realizarlo.

Resumen

La visión por computador refiere a la extracción de información de una imagen para resolver una tarea específica. Actualmente juega un papel importante en áreas como la industria, la medicina y la construcción de vehículos autónomos.

El presente informe resume el proyecto realizado para la construcción y puesta en funcionamiento de un módulo de Visión por computador aplicado a fútbol de robots, según las reglas de la liga *RoboCup Soccer* 2010; en el que se utilizó la plataforma BIOLOID de ROBOTIS en conjunto con el hardware de visión HaViMo en su versión 1.5.

El módulo de Visión desarrollado, alojado en los agentes robóticos, permite identificar los objetos pertenecientes al ambiente de fútbol de robots, obteniendo la posición relativa de los mismos al agente. Cuenta además con la posibilidad de interactuar con otras instancias del módulo de Visión ejecutadas en otros robots, a partir de la tecnología Zigbee, obteniendo así información inaccesible recabada localmente por otros agentes.

En el desarrollo del módulo fueron utilizadas técnicas de clasificación bayesianas, permitiendo reconocer distintos tipos de humanoides además de otros objetos; modelos vectoriales para el cálculo de profundidad a partir de una única imagen y un enfoque *result sharing* para la comunicación necesaria inherente al problema, buscando siempre minimizar los recursos utilizados. Asimismo, se plantearon las bases para lograr el desarrollo de un módulo de localización basado en localización por Monte Carlo.

Los resultados obtenidos sobre la capacidad de identificación del módulo y extracción de la posición de objetos son alentadores, obteniendo un porcentaje de identificación mayor a 70 % en la mayoría de los casos. Sin embargo, el sistema es dependiente del error cometido en la estimación de la orientación de la cámara, dato brindado como entrada para el funcionamiento del módulo.

Palabras clave Fútbol de robots, Visión por Computador, Visión Local, *RoboCup Soccer*.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

A lo largo de los siglos, el hombre a intentado construir una máquina a su imagen y semejanza. Sin embargo, las tareas simples y cotidianas de un ser humano requieren de intrincados procesos mentales fruto del aprendizaje realizado durante toda su vida. El mundo que lo rodea es percibido mediante cinco sentidos, siendo preponderante, en la amplísima mayoría, el sentido de la vista.

La visión, permite al ser humano relacionarse con su entorno, reconocer objetos, su forma, color y tamaño, así como obtener la sensación de profundidad y conocer la distancia que lo separa de los objetos. Con este propósito surge la motivación de reproducir la funcionalidad de la vista en los agentes robóticos.

La visión por computador es la ciencia y tecnología que busca dotar a las máquinas del sentido de la vista, en lo que respecta a extraer información de una imagen. Consiste en la identificación de objetos y extracción de propiedades de interés de los mismos.

El clásico problema de visión por computador y procesamiento de imágenes es averiguar si cierta imagen o video contiene un objeto específico o una característica especial. Tareas relevantes que requieren de la visión por computador son el reconocimiento de objetos, el análisis de movimiento, la reconstrucción de escenas y la restauración de imágenes.

Gracias al desarrollo de la visión por computador, actualmente se encuentra a nivel comercial, cámaras con reconocimiento de rostros, con disparadores automáticos que detectan sonrisas y dispositivos de entretenimiento que capturan los movimientos del jugador, entre otros.

RoboCup Soccer[1] es un proyecto internacional que busca promover el desarrollo de la inteligencia artificial, la robótica y campos de investigación afines a partir de encuentros de fútbol de robots. Tiene como objetivo principal que un equipo de robots venza al equipo campeón de fútbol humano, en 2050, siguiendo las reglas del fútbol humano. Es de especial interés para el área de visión por computador ya que presenta un ambiente dinámico con un acotado número de objetos conocidos, siendo posible y en algunos casos necesario, desempeñar tareas de reconocimiento de objetos y reconstrucción de escenas.

RoboCup Soccer cuenta con varias ligas, destacándose entre ellas la liga humanoide. La misma está compuesta por tres subligas de acuerdo al tamaño de los robots: *Teen Size*, *Kid Size* y *Adult Size*. En esta liga, cada equipo está compuesto por tres agentes robóticos humanoides, de los cuales uno de ellos desempeña el rol de golero. Un partido se compone entonces de dos equipos rivales, dos arcos y una pelota, donde cada equipo intenta anotar la mayor cantidad de goles en un tiempo establecido.

Existen otros formatos alternativos a partidos de fútbol, como el saque de lado, esquivar obstáculos y dribleo, tiro de penales y otros.

El presente proyecto utilizará el formato de partido de fútbol de la liga humanoide, *Kid Size*, que engloba

la mayor parte de los desafíos presentes en los demás formatos.

En tal sentido, se busca la creación de un módulo de visión para un equipo de fútbol de robots que respete las reglas mencionadas, el cuál brindará una interfaz amigable y sencilla, para ser utilizado posteriormente por la estrategia del equipo.

La principal motivación del proyecto surge a raíz de su propio contexto. El hecho de desarrollar un módulo de Visión que cumpla con los requerimientos impuestos por *RoboCup Soccer* es un paso más para lograr competir en dicha liga. Además, la realización de un módulo de Visión Local robusto, permite iniciar nuevos proyectos dentro del área de robótica que hagan uso del mismo, como son la estrategia, navegación, locomoción y localización, siempre restringiéndose al escenario de fútbol de robots.

1.1. Antecedentes

Si bien se cuenta con antecedentes de proyectos dentro del área de Visión por Computador en el instituto de Computación de la Facultad de Ingeniería, específicamente el proyecto "Visión de Robots" realizado por Gastón Fernández, Claudia Stocco y Natalia Tourn[1], el mismo no se aplica al tipo de visión que se aborda en el presente proyecto, correspondiéndose en su lugar con el paradigma de Visión Global dentro del campo. Esto generó la necesidad de utilizar un marco teórico basado puramente en el trabajo de terceros.

1.2. Objetivos

Los principales objetivos del proyecto fueron:

- Estado del Arte referente a los temas Visión por Computador, Reconstrucción Espacial y Visión Cooperativa.
- Estudio, evaluación e implementación de algoritmos de visión orientados al escenario de Fútbol de Robots.
- Estudio, evaluación e implementación de algoritmos de colaboración entre robots, orientado al escenario de Fútbol de Robots.

Sin embargo, existe como objetivo de mayor alcance la creación de un equipo competitivo para la liga humanoide *Kid Size* de *RoboCup*. Para ello, el grupo MINA[2] del Instituto de Computación cuenta con varios proyectos que engloban su construcción.

De esta manera se interactuó con demás proyectos[3], que se relacionan con el uso del módulo de visión local, para abordar el objetivo principal.

1.3. Principales referencias bibliográficas

Es de interés recalcar dentro de las referencias utilizadas el trabajo de Dieter Fox *et ál.*, en la introducción a modelos probabilísticos de clasificación en [2] y localización en [3], así como el trabajo de Michael Wooldridge en [4], el cual fue tenido en cuenta al desarrollar protocolos de comunicación, distribución de información y cooperación entre agentes.

[1] `http://www.fing.edu.uy/inco/grupos/mina/pGrado/vision2003/`

[2] Network Management - Artificial Intelligence (`http://www.fing.edu.uy/inco/grupos/mina/index.php`)

[3] Salimoo (`http://www.fing.edu.uy/~pgsalimoo/`)

En contraste con las referencias mencionadas, las cuales proveyeron el marco teórico necesario para el proyecto, fueron también de gran utilidad tanto el Manual del dispositivo HaViMo[5], la página oficial de soporte de ROBOTIS[6] y la comunidad de internet ROBOSAVVY[7] mediante la cual fue posible entrar en contacto directamente con el creador del módulo de hardware de la cámara.

1.4. Organización del documento

El presente documento se encuentra organizado de la siguiente manera.

El capítulo 2 presenta la base teórica que llevó al desarrollo de la solución, incluyendo conceptos generales del proceso de visión, clasificación de módulos de visión y tecnologías para la identificación y extracción de la posición de objetos en imágenes.

En el capítulo 3 se describen las principales características del módulo de visión, haciendo énfasis en los aspectos relevantes al proceso de visión, y comunicación entre agentes.

En el capítulo 4, se describen las pruebas realizadas y se presentan los resultados obtenidos, componiéndose de pruebas elementales y complementarias, buscando documentar la calidad evaluada y alcanzada así como brindar resultados de utilidad para quien utilice el módulo.

Por último, en el capítulo 5 se realiza un resumen de las conclusiones recabadas durante el proyecto, proponiendo a su vez los posibles pasos siguientes en lo que respecta a la solución y recomendaciones para un mejor funcionamiento del módulo.

Capítulo 2

Marco teórico

En este capítulo se presenta una introducción al concepto de módulo de Visión, así como al proceso de Visión en sí; describiendo además las restricciones impuestas para el módulo de Visión que se desea construir en particular, en el marco de la liga humanoide *kid-size*, de *RoboCup Soccer*.

Se realiza un resumen de los principales tópicos que fueron necesarios para la construcción de la solución. Los mismos se corresponden a:

- Reconocimiento de objetos: Necesario para la identificación de los objetos presentes en las imágenes capturadas (sección 2.3).
- Extracción de distancia: Requerimiento esencial del módulo de Visión para fútbol de robots (sección 2.4).
- Cooperación y comunicación entre agentes: Utilizado para acceder a información más completa del entorno de juego (sección 2.5).

Por más información sobre los temas presentados, se invita a consultar [8].

2.1. Conceptos generales

2.1.1. ¿Qué es un módulo de Visión?

Un módulo de Visión tiene la responsabilidad de, a partir de una imagen, reconocer los objetos en ella presentes.

La función de un módulo de Visión en un agente corresponde a la captura y procesamiento de imágenes, buscando el mejor tiempo de respuesta y maximizando la cantidad de cuadros por segundo procesados; reconociendo así objetos y sus características.

La figura 2.1.1 muestra los pasos básicos del proceso utilizado en el proyecto para el reconocimiento de objetos, siendo los mismos:

- *Obtención de imagen*: Realizado por hardware, a partir de uno o varios sensores de tipo cámara[1]. El

[1]
- Que varían desde cámara convencionales a dispositivos termográficos, radares y cámaras ultrasónicas.

resultado obtenido varía desde una imagen en dos o tres dimensiones hasta una secuencia de video, obteniendo una representación digital de la disposición de los objetos en la escena, para un determinado instante o secuencia de tiempo.

- *Segmentación de la imagen*: Consiste en el preprocesamiento de la imagen obtenida, de manera que sea más sencilla de procesar, identificando regiones de interés dentro de la escena, a partir de algoritmos de detección de contorno o agrupación de píxeles similares.

- *Reconocimiento de los objetos*: A partir de las regiones de interés, se intenta clasificar los objetos que componen la escena.

Figura 2.1.1: Proceso de Visión

Sin embargo, un módulo de Visión con las características requeridas para el fútbol de robots, debe brindar no sólo la identificación del objeto, sino también información esencial de los mismos, para ser utilizada en la toma de decisiones de componentes superiores. Es así que resulta necesario obtener la posición de los objetos reconocidos en la escena.

De esta manera, el proceso de Visión para el módulo de Visión a construir debe culminar con los siguientes pasos:

- *Extracción de posición de los objetos reconocidos respecto al robot*: Tomando como base las regiones de interés que componen al objeto identificado, se obtiene información relativa a la posición del objeto.

2.1.2. Clasificación de un módulo de Visión de fútbol de robots

Un módulo de Visión para fútbol de robots puede ser clasificado a partir de su posición en la escena. A continuación se describen dos tipos de módulo de Visión según su localización en la cancha y la interacción que realizan con los robots del campo.

Visión Global o aérea

Llamaremos módulo de Visión Global a los módulos de Visión construidos en base a una o más cámaras aéreas colocadas por encima de la cancha.

Los módulos de Visión Global utilizan imágenes que contemplan la cancha[2] y sus elementos desde un punto de vista estático, generalmente aéreo, y requieren de una infraestructura extra en la cancha como muestra la figura 2.1.2. Se caracterizan comúnmente por utilizar parches de colores sobre los objetos a identificar.

En general, la información de la imagen es recopilada por un computador externo y procesada, transmitiendo a los agentes en el campo el resultado del proceso de visión.

[2]Total o parcialmente

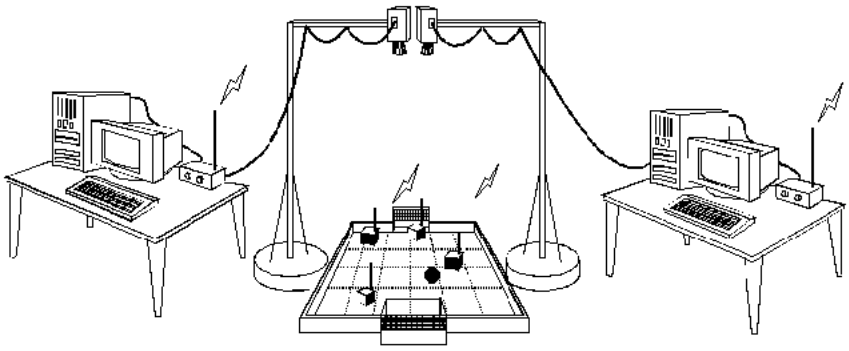

Figura 2.1.2: Esquema de Visión Global

Visión Local

Por el contrario, llamaremos módulos de Visión Local a los módulos de Visión que cuentan únicamente con una, o más cámaras alojadas en los agentes robóticos dentro del campo.

A diferencia de una Visión Global, la Visión Local cuenta con la capacidad de ver la altura de los objetos de la cancha. Sin embargo, su posición es variable dentro del campo mientras el agente robótico cuente con la capacidad de trasladarse. Por otro lado, considerando que la cámara debe estar embebida en los agentes, no es necesario montar una infraestructura extra.

A pesar de esto, la Visión Local cuenta con varias desventajas frente a la Visión Global:

Rango de visión: Una cámara aérea puede ver la mayoría de los elementos en la cancha en todo momento, mientras que una cámara local sólo puede hacerlo si está ubicada en una esquina de la cancha con la correcta orientación. En la mayoría de los escenarios de un partido de fútbol de robots, la cámara de Visión Local no estará orientada enfocando todos los objetos de la cancha.

Problema de objetos ocultos: Una cámara aérea puede ver todos los objetos de la cancha, sin embargo en las imágenes tomadas por la cámara local puede ocurrir que haya objetos detrás de otros, ocultando así objetos en la escena.

Interferencia del fondo: Los objetos de una imagen obtenida a partir de una cámara local pueden pertenecer tanto a objetos de la cancha como a objetos del fondo. Una cámara aérea enfoca únicamente la cancha, por lo que todos los objetos de la imagen pertenecerán a objetos de la cancha y por ende, conocidos. Si bien, el problema de interferencia del fondo existe para visión global, resulta más fácil asilar el fondo que en visión local.

Cámara móvil: Con Visión Local, la cámara no está en una posición fija, lo que la hace sensible a caídas y golpes; y no es posible escoger donde ubicarse. El mayor problema reside en las caídas que se pueden producir, ya que generan imágenes que no contienen información de interés, por ende haciendo inútil al hardware de visión momentáneamente. Además, las caídas pueden comprometer la integridad del hardware de la cámara.

Recursos compartidos: El módulo de Visión debe compartir espacio en disco, tiempo de procesador y ancho de banda. Esto limita la variedad de algoritmos que puede utilizar la Visión Local, frente a la Visión Global, esencialmente por el tiempo de ejecución de los mismos.

Sistema de coordenadas: Si bien no es una desventaja, una observación interesante es que es natural trabajar con coordenadas cartesianas cuando se tiene imágenes obtenidas de una cámara global donde el origen se corresponde con una de las esquinas de la cancha; sin embargo cuando se cuenta con imágenes de una cámara local, los datos referentes a la posición de los objetos se obtienen en coordenadas polares con origen la cámara.

En atención a los puntos mencionados surge como objetivo secundario mitigar las limitaciones de Visión Local frente a Visión Global.

2.2. Restricciones impuestas

RoboCup Soccer publica cada año reglas específicas que restringen el escenario que deben enfrentar los robots en las distintas ligas. Las restricciones impuestas para la liga humanoide *Kid Size* para el campeonato del mundo 2010, desarrollado en Singapur, se describen en [9]. Las reglas contienen información referente a dimensiones de la cancha, composición del hardware para los agentes robóticos, restricciones de tamaño y distribución de marcas en el campo entre otros.

Las restricciones pertinentes al módulo de Visión para la liga *Kid Size*, para el año 2010[3] se resumen en las siguientes:

- *Los robots deben ser autónomos*: Los agentes robóticos de cada equipo no deben ser guiados por humanos.
- *Se prohíbe la comunicación entre robots y dispositivos externos*: Se restringe la comunicación que los robots pudiesen tener con terminales que realicen procesamiento externo, o distribuyan información referente al ambiente; no es así entre los robots de la cancha.
- *Cada robot puede contar con un único sensor de tipo cámara*: Cada robot integrante del equipo puede contar con una única cámara, la cual debe ser colocada en la cabeza.
- *El rango de visión del sensor de tipo cámara, dada una imagen, está limitado a 180 grados:* De esta manera, en una misma imagen no se podrá obtener información de todos los elementos del campo en todo momento.
- *En caso de que se cuente con un cuello móvil, se permite inclinar el mismo* ± 135 *grados verticalmente y* ± 90 *grados horizontalmente.*

Las restricciones impuestas implican que tanto el hardware de visión como el software deberán estar localizados en el agente, siendo el procesador del robot el que deba realizar los cálculos pertinentes a la identificación y extracción de propiedades de los objetos. Puesto que dentro del robot deberán convivir tanto el sistema de Visión como otros diversos componentes, el módulo de Visión debe por lo tanto minimizar el uso de recursos como son espacio en disco, tiempo de procesador y ancho de banda.

2.3. Reconocimiento de objetos

2.3.1. Segmentación de imagen obtenida

La segmentación de la imagen refiere al preprocesamiento realizado durante el proceso de Visión previo al reconocimiento de objetos. Tiene como fin la identificación de regiones de interés, que tienen una alta

[3] El proyecto fue iniciado en 2010.

posibilidad de pertenecer a objetos reconocibles, obteniendo una representación más sencilla de la escena.

Las dos técnicas más comunes para definir regiones de interés son:

Algoritmos de detección de contornos

Los algoritmos de detección de contornos, se caracterizan por definir las regiones de interés detectadas a partir de su contorno. Es necesario recorrer la imagen, describiendo líneas paralelas como muestra la figura 2.3.1, en busca de variaciones de color que definan los contornos [10]. Una vez identificadas las regiones de interés, las mismas pueden ser representadas a partir histogramas de orientación [11], producto de la subdivisión de la región en ventanas, a las que se le aplica un operador Sobel[4], utilizando la imagen capturada en escala de grises.

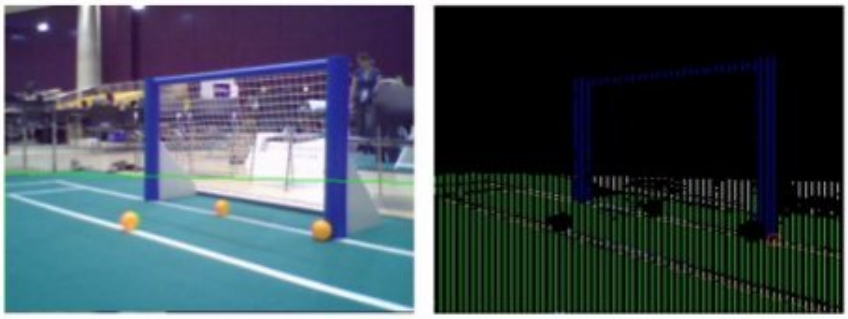

Figura 2.3.1: Detección de contornos

La figura 2.3.2 muestra un ejemplo de histograma de orientación para una imagen en particular. En ella, se muestra la división de la imagen en 9 ventas o secciones, y el resultado de la aplicación del operador Sobel para cada una de ellas.

Agrupación en *blobs*

Un *blob* es un conjunto de píxeles de la imagen con características similares. En lugar de definir exactamente el contorno de los objetos, se define el conjunto de píxeles que componen al objeto. Los *blobs* suelen estar definidos como figuras geométricas sencillas (rectángulos) que contienen los píxeles de interés. De esta manera es posible obtener una representación del objeto menos exacta, en busca de performance.

El sensor de cámara HaViMo, realiza la etapa de segmentación por hardware, aliviando la carga del procesador del robot. Sin embargo, esto condiciona los métodos de reconocimiento de objetos que se quieran utilizar, ya que no se tiene acceso a la imagen capturada.

La figura 2.3.3 muestra tres imágenes correspondientes a la captura de una escena correspondientes al dispositivo HaViMo en su versión 1.5, el cual realiza agrupación en *blobs* por hardware. La primer imagen muestra la captura obtenida por el dispositivo. La segunda imagen muestra los píxeles detectados según la calibración realizada. Por último, la tercer imagen muestra los *blobs* obtenidos por hardware, los cuales son retornados para el reconocimiento de objetos.

[4]Operador diferencial discreto que calcula una aproximación al gradiente de la función de intensidad de una imagen

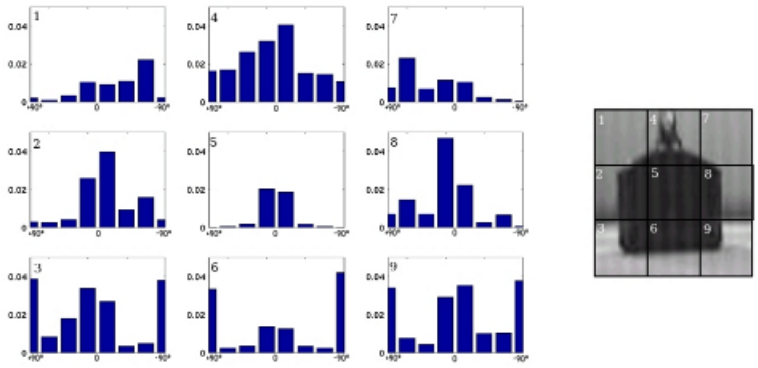

Figura 2.3.2: Ejemplo de histograma de orientación

Figura 2.3.3: Agrupación en *blobs* por HaViMo 1.5

2.3.2. Reconocimiento de objetos

La identificación de objetos consiste en la identificación del tipo de objeto al que pertenecen las regiones de interés obtenidas bajo la etapa de segmentación. De esta manera, se aplican distintas técnicas que comparan las propiedades de las regiones buscando identificar un objeto previamente conocido.

En [8] se comentan las principales técnicas de reconocimiento de objetos, donde destacan:

- *Template matching:* Donde se compara uno o varios patrones de los objetos que se intentan clasificar contra las regiones de interés [12]. Como desventaja, requiere de un gran tiempo de ejecución, inviable para la plataforma robótica utilizada.

- *Redes neuronales probabilísticas:* Consiste en la construcción de una red neuronal capaz de realizar la clasificación del objeto [13].

- *Clasificadores bayesianos:* Utiliza la probabilidad de que la clasificación de un objeto sea o no exitosa para distinguir frente a qué elemento se encuentra [2].

Clasificadores bayesianos

Los clasificadores bayesianos basan su técnica en el principio de Bayes, utilizando la probabilidad de encontrarse en un estado x del ambiente, dado que los sensores retornan la información z:

$$p(x|z)$$

Al aplicar bayes sobre la probabilidad anterior, se permite inferir el valor de x, utilizando la probabilidad de haber sensado z, asumiendo que x era el resultado esperado:

$$p(x|z) = \frac{p(z|x) * p(x)}{p(z)} = \frac{p(z|x) * p(x)}{\sum_{x'} p(z|x') * p(x')}$$

Donde x' se corresponde con los estados del ambiente distintos a x.

De esta manera, si se recorre todas los tipos de objetos w_i, calculando $p(w_i|z)$, es posible clasificar un objeto como de tipo w_i si

$$p(x|w_i) > p(x|w_j) \nabla j,\ j \neq i$$

2.4. Extracción de distancia

Una vez reconocidos los objetos en la imagen, es de interés obtener propiedades sobre los mismos. Para el caso específico de un módulo de Visión para *RoboCup Soccer*, interesa obtener principalmente la posición de los objetos.

Dado que no se cuenta con un módulo de localización, el cual brinda información sobre la posición del agente en el campo, no es posible retornar información de la posición de los objetos en coordenadas cartesianas de la cancha. Sin embargo, sí es posible obtener la posición de los objetos en coordenadas polares con centro el robot.

La posición en coordenadas polares puede ser obtenida mediante dos modelos:

- *Basado en proporciones:* Bajo la premisa que se mantiene la proporción entre el tamaño y la distancia de los objetos a medida que se alejan de la cámara, se obtiene la distancia y ángulo desde la cámara al centro del objeto.
- *Vectorial:* Realiza una representación vectorial de la configuración de la cámara en la escena, obteniendo la distancia y ángulo de un punto en el plano del piso a la base de la cámara.

Modelo de proporciones

El modelo de proporciones trabaja bajo la siguiente premisa:

$$\frac{ancho(cm)}{distancia(cm)} = \frac{ancho(píxeles)}{distancia(píxeles)}$$

Donde $distancia(píxeles)$ es un valor constante, propio de cada cámara, conocido como distancia focal que se corresponde con la distancia de la cámara al plano de la imagen. La distancia focal es obtenible en la práctica, conociendo los otros tres valores.

A partir de experimentos, donde se conoce previamente el *ancho* de un objeto tanto en *centímetros* como en *píxeles*, y la *distancia* desde la cámara al objeto en *centímetros*, se puede obtener la *distancia* en *píxeles*.

Es así que, obteniendo el $ancho(píxeles)$ de un objeto en cuestión, del que se conoce su verdadero $ancho(cm)$, es posible obtener su $distancia(cm)$. Esto es posible aplicarlo en un ambiente donde se conozca previamente las dimensiones de los objetos a identificar.

La figura 2.4.1 muestra el esquema del modelo de proporciones para el objeto pelota. Para este caso particular, la distancia a la pelota en el campo es derivada de la siguiente manera:

$$|\overleftrightarrow{OP_2}| = \frac{DF * |\overleftrightarrow{CD}|}{|\overleftrightarrow{AB}|}$$

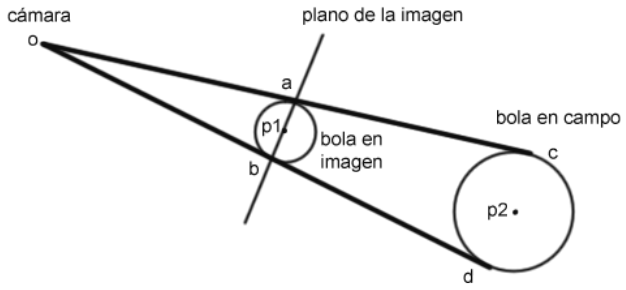

Figura 2.4.1: Esquema de modelo de proporciones

Modelo vectorial

En oposición al modelo de proporciones, el modelo vectorial utiliza un enfoque más costoso computacionalmente, pero obtiene mejores resultados según pruebas realizadas durante el proyecto.

Bajo el supuesto que todos los elementos del campo permanecen en contacto con el suelo, busca obtener la distancia desde la base del robot, al objeto en cuestión.

Se trabaja en base a dos sistemas de coordenadas cartesianas. Por un lado, las coordenadas de mundo, centradas en la proyección de la posición de la cámara sobre el piso (el punto O), cuentan con tres abscisas:

- x: Orientada al costado derecho del robot.
- y: Orientada al frente del robot.
- z: Orientada hacia la cámara.

Por otro lado, las coordenadas de imagen, utilizadas para localizar puntos en el plano de la imagen. Cuenta con su origen H en el centro de la imagen, y dos abscisas:

- i: Orientada hacia el margen izquierdo de la imagen.
- j: Orientada hacia el margen superior de la imagen.

La figura 2.4.2 presenta el modelo vectorial, donde se observa desde la posición de la cámara E, el punto de interés Q que tiene como proyección en el plano de la imagen el punto T.

El modelo vectorial permite obtener el valor de $|\overleftrightarrow{OQ}|$, que define la distancia de la base del robo al punto Q, y el ángulo Θ que define la distancia angular del frente del robot al punto Q.

Como principales desventajas, se destacan:

- *Es necesario conocer la altura de la cámara al suelo*

En robots rodado o que no cuentan con locomoción, la altura de la cámara podría considerarse constante; sin embargo, para el contexto de fútbol de robots donde se cuenta con humanoides bípedos, la altura de la cámara varía durante el ciclo de caminata.

- *Es necesario conocer los ángulos de pan y tilt de la cámara*

Los ángulos de *pan* y *tilt* definen hacia donde se encuentra orientada la cámara, respecto al plano del piso (*pan*) y al plano del frente del robot (*tilt*), que en la figura 2.4.2 se corresponden con el plano *xy* y el plano *yz* respectivamente. La figura 2.4.3 muestra la disposición de los ángulos de *pan* y *tilt*. Estos ángulos pueden ser variables debido a movimientos durante la caminata de un robot, o incluso por el uso de un cuello móvil.

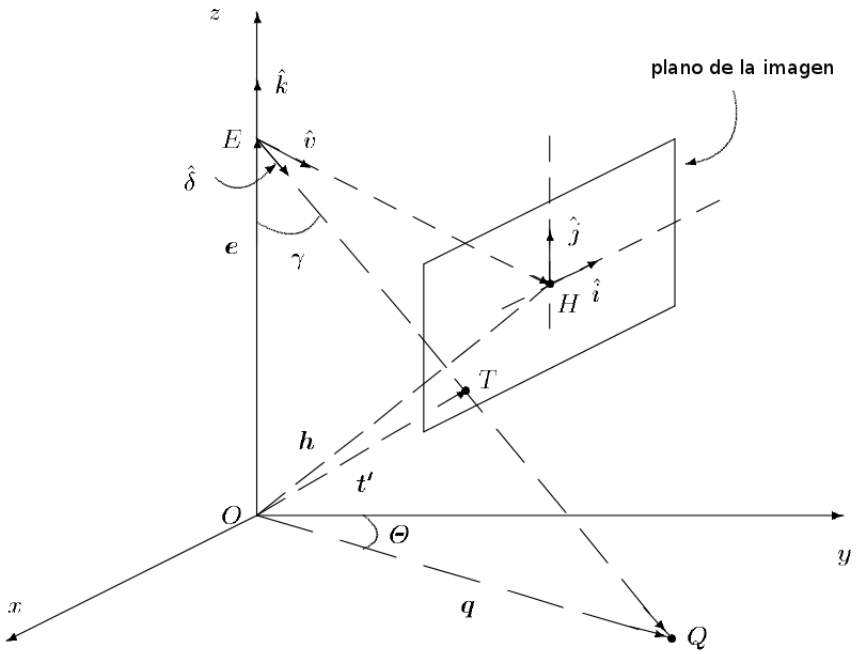

Figura 2.4.2: Esquema de modelo vectorial

En la práctica, cámaras de baja resolución presentan mejores resultados utilizando el modelo vectorial que con el modelo de proporciones, siempre que los datos de altura y orientación de la cámara sean precisos.

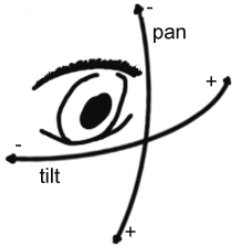

Figura 2.4.3: Ángulos de *pan* y *tilt*

2.5. Cooperación entre agentes

Según la interacción que los robots puedan tener con otros, un sistema puede ser clasificado como individual o multiagente. La caracterización de un sistema como multiagente es una decisión de modelado, pudiendo considerar un sistema multiagente cuando es de utilidad. Específicamente en el fútbol de robots, se cuenta con más de un robot interactuando con el ambiente de los cuales es posible obtener información que mejore las capacidades de los integrantes del equipo, clasificándose como multiagente [8]. Los sistemas multiagente presentan la posibilidad de que los robots compitan o cooperen por objetivos comunes.

[14] define la cooperación como el aumento de la utilidad total del sistema a raíz de un mecanismo cooperativo.

Es de interés realizar un módulo de visión local cooperativo, en pos de mitigar las restricciones que presenta frente a módulos de Visión Global.

Por mayor información, en [8] se realiza una introducción a cooperación multiagente.

2.5.1. Cooperación para solución distribuida de problemas

CDPS [15] estudia como un grupo de agentes puede trabajar para resolver problemas que exceden las capacidades individuales de cada uno. La cooperación en estos casos resulta necesaria dado que los agentes no cuentan con la experiencia y/o información necesaria para resolver la totalidad del problema.

Se distinguen dos enfoques para la resolución del problema:

- *Task Sharing*: Propone la distribución de porciones del problema a los distintos agentes.
- *Result Sharing*: Propone la distribución de información entre los distintos agentes.

Task Sharing

Un problema es descompuesto en pequeños subproblemas y asignado a diferentes agentes. Surge la interrogante de qué tarea se debe asignar a cuál agente.

Si todos los agentes son homogéneos en términos de sus capacidades, entonces cualquier tarea puede ser asignada a cualquier agente. Sin embargo, en los casos no triviales, donde se tengan agentes autónomos que puedan no aceptar determinadas tareas, se podría negociar la asignación de tareas mediante una subasta o una negociación [15].

Result Sharing

Involucra agentes compartiendo información relevante de sus subproblemas. Esta información puede ser compartida proactivamente[5], o reactivamente[6].

En *result sharing*, un problema es solucionado mediante la cooperación de los agentes, intercambiando información a medida que la solución es desarrollada. Típicamente, estos resultados progresarán desde una solución a pequeños problemas, que son luego refinadas en soluciones más abstractas. [15] menciona que, a partir del uso de *result sharing*, la *performance* del conjunto de agentes puede mejorar, según las siguientes características:

Confianza en la solución Soluciones independientes pueden ser utilizadas para comparar, encontrando posibles errores, e incrementado la confianza en la solución global.

Completitud de la solución Los agentes pueden compartir su enfoque del problema y su solución, para obtener una solución más completa.

Precisión de la solución Los agentes pueden compartir resultados para asegurar que la precisión de la solución global aumente.

Ahorro de tiempo Incluso si bastase con un solo agente para solucionar un problema, al compartir los resultados intermedios de la solución, el resultado puede ser derivado con mayor rapidez.

2.5.2. Visión multiagente en fútbol de robots

La visión multiagente surge en pos de mitigar las restricciones que presenta la visión local frente a la visión global. El escenario de fútbol de robots, presenta la oportunidad de enfocar el problema de visión como un sistema multiagente, en el que se cuenta con tres cámaras, en lugar de una. Es así que se busca obtener mayor información del ambiente, contando con mayor cantidad de puntos de vista[7].

De los enfoques presentados por CDPS, *Result Sharing* se aplica de manera natural al concepto de visión multiagente, donde se busca obtener más información de la que se cuenta localmente para obtener la posición de todos los objetos del campo.

La distribución de información puede darse a niveles de:

- Informar imagen obtenida
- Informar resultado del proceso de visión

Distribuir las imágenes obtenidas no es de utilidad para otros robots debido a que no se conoce la posición de los agentes que se están comunicando[8]. Además, el procesamiento local de más de una imagen reduciría la *performance* del sistema considerablemente.

Sin embargo, la distribución del resultado local de visión, permite a otros agentes conocer la posición relativa de los objetos del campo a sus aliados, siendo de gran utilidad para decisiones del módulo de estrategia y en un segundo lugar, para el módulo de localización.

[5] Donde un agente envía a otro agente información porque cree que al otro agente le interesa
[6] Donde un agente envía información a otro agente en respuesta a un pedido previamente realizado
[7] Donde cada punto de vista del ambiente es aportado por un robot del equipo.
[8] Debido a que no se cuenta con un módulo de localización

Capítulo 3

Descripción de la Solución

En el presente capítulo se describen las características del módulo de visión local desarrollado. En la sección 3.1 se plantea el problema que debe resolver la solución desarrollada, enumerando las características del hardware utilizado (sección 3.2) y las limitaciones operativas que se pudieran presentar.

En la sección 3.3 se repasan las restricciones de un módulo de visión local frente a uno global indicadas en la sección 2.1.2, describiendo cómo son mitigadas por el módulo construido.

Se realiza además una introducción a los componentes de la solución en 3.5, los cuales pueden ser consultados con más detalle en [16]; mencionando además, las interacciones del módulo con otros sistemas, como VisRobUtils, en la sección 3.6.

Por último se comentan los principales desafíos en la construcción del módulo:

- *Objetos reconocidos*: Se describen los objetos identificables por el módulo de visión, sección 3.7.
- *Propiedades de los objetos*: Se mencionan las propiedades de interés para componentes de estrategia y localización, así como las propiedades retornadas, sección 3.8.
- *Comunicación entre agentes*: Introducción a las características de la comunicación y protocolo utilizado, sección 3.9.

3.1. Requerimientos

La aplicación práctica del módulo de visión desarrollado, corresponde a la liga humanoide *kid-size* de *RoboCup Soccer*. Dadas las restricciones de la liga, mencionadas en la sección 2.2, se desprende que el módulo debe estar alojado en el agente, correspondiéndose con un módulo de visión local.

Dada la naturaleza multiagente del ambiente de fútbol de robots, es de interés que el módulo pueda comunicarse con los demás agentes, mitigando así algunas de las desventajas frente a la visión global.

De esta manera, surge la división del problema en dos subproblemas:

- Implementar el proceso de visión.
- Implementar comunicación entre agentes.

A su vez, se busca que el resultado cuente con características deseables en software de propósito general. La solución debe ser:

- *Configurable*: El módulo de visión debe poder ser calibrado para distintas configuraciones, respondiendo a cambios en las reglas de *RoboCup Soccer*.

- *Extensible*: Es necesario que la solución pueda ser extendida, en caso de que componentes superiores[1], deban tener acceso a otras propiedades de los objetos identificados.

3.2. Plataforma utilizada

3.2.1. BIOLOID - kit PREMIUM

Para el desarrollo del módulo de visión se contó como plataforma robótica con el kit BIOLOID PREMIUM de ROBOTIS.

Entre las características principales del kit se encuentran:

- Permite la construcción de un agente robótico bípedo que cumple con las dimensiones establecidas por la liga *RoboCup Soccer* 2010.

- Cuenta con un paquete de aplicaciones básicas para programar movimientos, comunicarse mediante una terminal o comprobar el estado de motores y otros periféricos integrados al robot.

- ROBOTIS pone a disposición el código fuente de librerías implementadas en el lenguaje de programación C para desarrollar código embebido en el agente. El código desarrollado puede ser cargado al robot mediante la misma terminal que provee ROBOTIS.

- Utiliza actuadores de la línea dynamixel, brindando librerías para su comunicación.

- Presenta varios métodos de comunicación entre el agente y una computadora de escritorio, a través del uso de un cable serial o con la tecnología Zigbee[2]. Además brinda librerías nativas para el uso de dichos métodos.

La figura 3.2.1 muestra uno de los modelos estándar de humanoides que pueden ser construidos con el kit.

A pesar de estas virtudes, el kit PREMIUM no cuenta con un hardware de visión, por lo que para la captura de imágenes fue necesario el uso de un dispositivo de hardware de un proveedor distinto, en este caso HaViMo en su versión 1.5[3].

Cuello móvil En el documento de Estado del Arte[8] se presentan los tres modelos de robots bípedos descritos en el manual de BIOLOID[17], los cuales pueden ser construidos a partir de las piezas que contiene el kit PREMIUM. De los tres modelos, resultó de particular interés construir el modelo B, que utiliza 16 de los 18 motores del kit, lo que permitió la posibilidad de ensamblar un cuello móvil para la cámara. Si bien el cuello móvil no fue desarrollado, el módulo de Visión brinda funcionalidades para establecer el ángulo de la cámara relativo al frente del robot, permitiendo el uso del mismo. Se alienta a quien utilice el módulo de Visión con este kit robótico a desarrollar un cuello móvil que potencie las características del mismo.

[1] Componentes que utilicen el módulo de visión, como componentes dedicados a la estrategia de juego y toma de decisiones.

[2] Tecnología de comunicación inalámbrica similar a *bluetooth*.

[3] Si bien ROBOTIS cuenta con hardware de visión para la línea BIOLOID, el mismo transmite imágenes a un computador externo, y no forma parte del paquete PREMIUM.

Figura 3.2.1: BIOLOID kit PREMIUM

3.2.2. Cámara HaViMo 1.5

El dispositivo de hardware HaViMo, se compone principalmente de una cámara de resolución 160 x 120 píxeles, un microcontrolador y una memoria flash utilizada para almacenar la calibración del dispositivo.

HaViMo se destaca principalmente por dos características:

- Para transmitir los datos procesados de la imagen capturada se utiliza un protocolo similar del que hace uso BIOLOID para comunicarse con los dispositivos dynamixel. Por consiguiente, es posible reutilizar las librerías de ROBOTIS encargadas de la comunicación con dichos motores, para comunicarse también con HaViMo.

- El dispositivo no permite acceder a la imagen que captura la cámara. En cambio, realiza por hardware la etapa de segmentación del proceso de visión[4], es decir, a partir de la imagen capturada obtiene zonas que agrupan píxeles contiguos con características similares[5]. De esta manera, en lugar de transmitir la totalidad de la imagen, el módulo comunica únicamente los *blobs* detectados.

También es de interés comentar que el ángulo de visión de la cámara es de 45 grados, por lo que no transgrede las reglas de *RoboCup Soccer* descritas en la sección 2.2. Por último, con respecto a su montaje en el robot, su pequeño tamaño hace sencilla dicha tarea utilizando una de las extensiones extra con las que cuenta el kit, como se muestra en la figura 3.2.2. Con el objetivo de preservar la cámara durante las pruebas físicas, los integrantes del proyecto de grado Salimoo [18], responsables de estrategia y locomoción del robot, diseñaron un contenedor para la cámara buscando protegerla de posibles caídas, el cual no obstaculiza la captura de imágenes del dispositivo. La misma se puede apreciar en 3.2.3.

[4] Ver sección 2.3

[5] Para el caso particular de HaViMo, la única característica que se toma en cuenta es el color de los píxeles

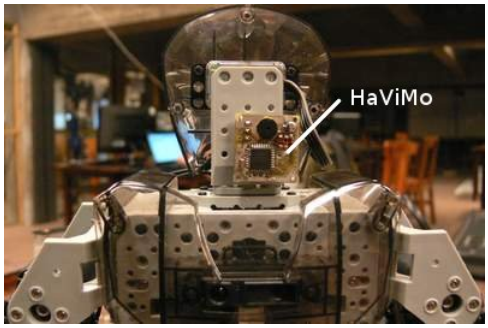

Figura 3.2.2: HaViMo 1.5 montado en PREMIUM BIOLOID

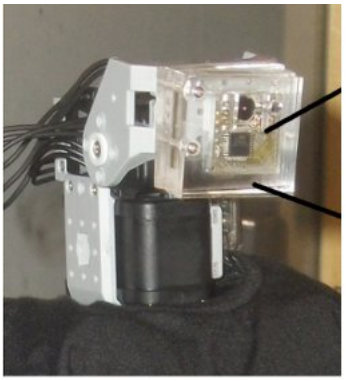

Figura 3.2.3: HaViMo 1.5 en contenedor

3.2.3. Zig110 y Zig100

El dispositivo de hardware Zig110 implementa la comunicación de la tecnología Zigbee, y es utilizado por los controladores CM-510[6]. ROBOTIS pone a disposición la librería libzigbee que implementa la comunicación y recepción de mensajes de 16 bits a partir de Zig110, con detección de error[7].

La comunicación efectuada por el módulo de visión, será realizada utilizando el componente Zig110, que viene incluido en todos los kit PREMIUM.

La figura 3.2.4 muestra los dispositivo Zig110 y Zig100. Por defecto un dispositivo Zig110 se encuentra emparejado con un dispositivo Zig100, permitiendo únicamente comunicación entre dispositivos pareados. Sin embargo, es posible configurarlos para establecer comunicación en modo *broadcast* necesaria para que los robots del campo logren comunicación entre todos.

Es posible utilizar el adaptador Zig2Serial en conjunto con Zig100 para lograr acceso al canal de Zigbee desde una computadora externa. La figura 3.2.5 muestra a Zig100 montado en Zig2Serial.

[6]Controlador de robots de la línea BIOLOID de ROBOTIS
[7]Los mensajes corruptos son desechados.

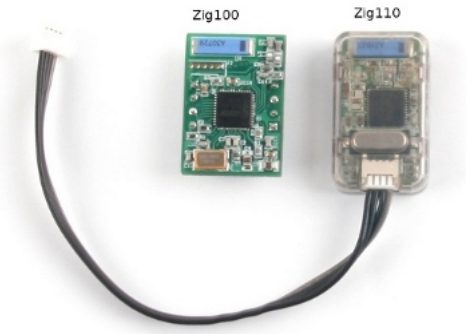

Figura 3.2.4: Zig110 y Zig100

Figura 3.2.5: Zig100 montado en Zig2Serial

3.3. Mitigando las limitaciones

Las limitaciones de un módulo de visión local presentadas en la sección 2.1.2, colocan a la visión global como una mejor alternativa frente a éste. Sin embargo, la visión local es un requerimiento necesario para cumplir la meta de *RoboCup Soccer*, vencer a un equipo de fútbol humano.

Es de esencial interés lograr mitigar las limitaciones para realizar un módulo de visión fidedigno y que sea una opción viable a la visión global.

Rango de visión Quizás la mayor limitación del sistema de visión local. El acotado rango de visión impide conocer la posición de todos los objetos de la cancha en todo momento, a menos que estén dentro de la imagen capturada. Aún así se tendría que contar con la suficiente casualidad de no estar frente a un escenario de objetos ocultos ni interferencia del fondo, para lograr identificar los objetos sin error. El enfoque del módulo de visión como sistema multiagente, permite abordar el problema no como una única visión local en la cancha, sino como tres lecturas de visión local en un mismo momento desde puntos de visita distintos, abarcando así un mayor rango de visión.

Problema de objetos ocultos El problema de objetos ocultos ocurre no solo con agentes robóticos, sino con cualquier sistema que utilice visión local, incluso humanos. No hay manera de poder identificar objetos ocultos por otro, sin embargo la visión multiagente, ayudaría a obtener mayor información del ambiente, desde distintos puntos de vista, reduciendo así el número de escenarios con casos de objetos ocultos.

Interferencia del fondo La interferencia del fondo se hace presente al existir tanto en la cancha, como fuera de ella, objetos con los mismos colores. Además de realizar una buena calibración inicial, este problema puede ser mitigado por quien controle los motores del robot, inclinando los motores del cuello del agente, buscando que la cámara enfoque únicamente los objetos de interés.

Cámara móvil Si bien no es posible obtener *blobs* de interés cuando un agente robótico cae, la comunicación realizada por la visión permite obtener la información de otros agentes, que puede o no resultar de interés en dichos casos.

Recursos compartidos Debido a que el procesamiento necesario por el sistema debe realizarse exclusivamente en el robot, el módulo de Visión estará consumiendo ciclos del procesador para analizar la imagen obtenida. Los algoritmos de visión por computador generalmente son de un alto costo computacional, a lo que sumado que el controlador CM-510 del kit PREMIUM se compone por un procesador ATMega2561 de 16Mhz de frecuencia se agrava la situación[8].

No obstante, la elección de un hardware de cámara, como HaViMo, que realiza la etapa de segmentación de la imagen por hardware, libera considerablemente la carga del procesador. Incluso, el consumo de memoria es despreciable, dado el acotado número de *blobs* reconocidos por imagen procesada[9].

Sin embargo, la inclusión de una capa de comunicación para contrarrestar el acotado rango de visión de un módulo local, hace necesaria la transmisión de datos entre agentes, específicamente a través de la tecnología Zigbee. Este mismo canal de comunicación es utilizado por otros componentes del robot, por lo que se debe diferenciar a qué componente de software corresponde cada mensaje, enviado o recibido. Es así que surge la necesidad de implementar un filtro sobre el acceso al recurso, que agregue a los mensajes enviados información sobre qué componente fue el autor y provea algún mecanismo para obtener mensajes de un autor específico. Debido a esta necesidad de autenticación, existen restricciones sobre los mensajes que pueden enviar otros componentes, acotando el espacio de mensajes posibles.

Dentro de los roles que desempeñan los agentes en un equipo de fútbol de robots, sobresale el de golero. Su posición casi estática sumado al hecho de que su cámara está orientada, la mayoría de las veces hacia el centro de la cancha lo propone como un objeto interesante a considerar en el módulo de visión, a fin de aliviar la carga algorítmica.

No obstante, no es posible basarse en el supuesto antedicho, dado que la dinámica del ambiente no permite establecer una posición fija al golero y es una elección correspondiente a quién desarrolle la estrategia de juego. En consecuencia, la solución desarrollada es homogénea para todos los agentes sin importar el rol que interpretan dentro del equipo.

3.4. Limitaciones operativas

Es de interés mencionar algunas limitaciones sufridas a lo largo del proyecto que pudieron tener un impacto en el resultado obtenido.

Como principal, destaca la limitada capacidad de depuración en la plataforma robótica[10], lo cual generó la necesidad de reproducir el ambiente utilizado a fin de poder lograr una mejor productividad. Por esta

[8] A nivel comparativo, en un procesador core i3 de 2.13GHz × 4, algoritmos de detección de relieve en imágenes de resolución de 500 x 500 píxeles pueden demorar 10 minutos.

[9] HaViMo 1.5 reconoce hasta 15 *blobs* en cada imagen

[10] No es posible usar un *debugger* en el robot, pero sí acceder a una consola para imprimir mensajes de control.

razón se dedicó el tiempo que se contó con acceso a la plataforma robótica[11] a obtener datos de prueba, para depurar la capa de visión externamente. Fue necesario además la construcción de un simulador de hardware de comunicación para depurar la codificación de mensajes.

Más importante aún fue la incapacidad de depurar el módulo de visión una vez incorporada la capa de comunicación[12], lo que presentó una seria restricción en cuanto al acceso a resultados de pruebas. Lo antedicho impuso la construcción de una aplicación de escritorio capaz de comunicarse con el robot, recibiendo esporádicamente los resultados de la ejecución del módulo de visión, utilizando la tecnología Zigbee.

Por último, comentar brevemente los problemas surgidos para lograr comunicación en modo *broadcast* entre Zig110 y Zig100 utilizando el adaptador Zig2Serial, para lo cual fue necesario modificar el hardware Zig2Serial; y el poder realizar la correcta calibración del hardware de visión alimentando de corriente a la cámara en la etapa de calibración. Una explicación más detallada de la solución a estos problemas puede encontrarse en [19].

3.5. Componentes de la solución

La arquitectura escogida corresponde a un diseño en capas, separando las 3 principales funciones del módulo:

- Acceso al hardware de cámara HaViMo.
- Proceso de Visión.
- Comunicación con los demás agentes.

La figura 3.5.1 muestra el diagrama de componentes del módulo de visión desarrollado, donde los componentes internos del módulo se encuentran comprendidos por la línea punteada. Es posible que el Módulo de Estrategia utilice al Módulo de Visión sin comunicación, accediendo directamente a las funcionalidades de la Capa de visión. A continuación se describen las responsabilidades de cada componente de la solución; una descripción más detallada de los componentes internos y externos al módulo de visión puede ser consultada en [16].

3.5.1. Capa de acceso a HaViMo

La Capa de acceso a HaViMo abstrae el uso de la cámara para la Capa de visión, siendo responsable del acceso al hardware de HaViMo[5]. Está compuesta por un único módulo:

- *Interfaz HaViMo*: Abstrae el uso de HaViMo, realizando el pasaje del resultado de segmentación a memoria interna del controlador, e invocando las funciones necesarias para procesar la siguiente imagen disponible.

3.5.2. Capa de visión

La Capa de visión, encargada de realizar el proceso de visión por computador, presenta los componentes:

[11] El acceso al laboratorio donde se encuentran los robots y el hardware de visión es limitado.

[12] La librería de acceso a Zigbee provista por BIOLOID no es compatible con la librería de acceso a la salida serial del robot, utilizada para imprimir en consola y también provista por BIOLOID.

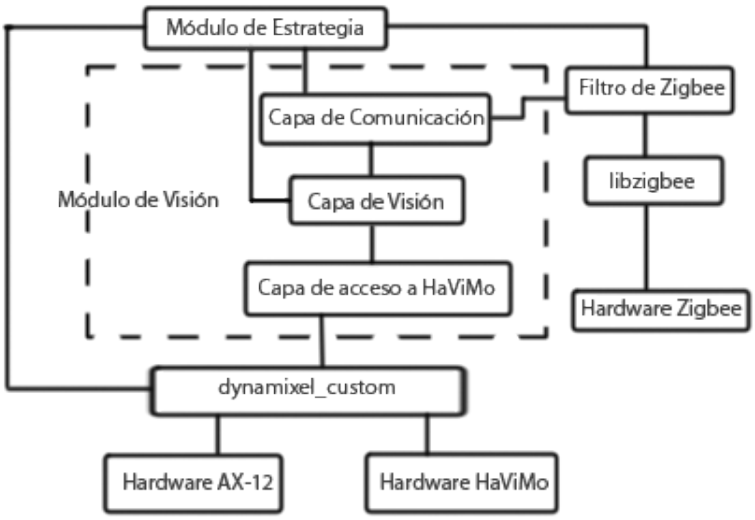

Figura 3.5.1: Arquitectura del módulo de visión

- *Clasificación*: Responsable de la agrupación de *blobs*, en objetos de mayor tamaño, facilitando la tarea de identificación.

- *Reconocimiento de objetos*: Implementación de las etapas de reconocimiento de objetos y extracción de propiedades del proceso de visión, utilizando clasificadores bayesianos para el reconocimiento de robots y modelo vectorial para la extracción de distancia a los objetos.

- *Interfaz de Visión*: Presenta los datos obtenidos a otros componentes.

3.5.3. Capa de comunicación

La Capa de comunicación tiene la responsabilidad de interactuar con los demás agentes, obteniendo las propiedades de los objetos identificados por otras instancias del módulo residentes en otros agentes robóticos, y presentar la información recabada a quien invoque el módulo de visión. Se compone de:

- *Adaptador Zigbee*: Implementación del patrón *Adapter*[20], creando paquetes que puedan ser transmitidos por Zigbee, y entendidos por la Capa de comunicación.

- *Buffer de Objetos*: Almacena los mensajes recibidos de las demás instancias de visión temporalmente.

- *Interfaz de Comunicación de Visión*: Implementa el protocolo de comunicación, presentando los resultados de la visión de otros agentes a componentes superiores.

3.5.4. Componentes extra

Fue necesario el desarrollo de componentes extra externos al módulo de visión, para permitir la interacción con demás componentes.

dynamixel_custom

Basado en la librería base de comunicación con dispositivos dynamixel provista por BIOLOID. Permite la comunicación de instrucciones específicas para la cámara HaViMo.

Filtro de Zigbee

Brinda una interfaz transparente para el acceso a mensajes de Zigbee. Permite distinguir mensajes de visión del resto.

En la sección D se presentan las funciones accesibles por componentes superiores del módulo de visión y del Filtro de Zigbee.

3.6. VisRobUtils

Para verificar que los datos que brinda el módulo de visión se correspondan a la realidad, es necesario poder monitorear los resultados obtenidos. Sin embargo, los agentes no pueden almacenar información dentro del robot por lo que surge la necesidad de monitorear el módulo *online*[13].

La comunicación estándar de los robots BIOLOID con un PC se realiza mediante un cable serial utilizando herramientas del proveedor[14]; sin embargo el puerto serial presenta conflictos con la librería que provee acceso a Zigbee, utilizada por la capa de comunicación del módulo de visión[15].

Una alternativa a la comunicación serial, es el adaptador Zig2Serial, el cual brinda comunicación de manera inalámbrica, permitiendo el uso de un dispositivo Zig100 desde el computador. Esto abre la posibilidad no sólo de monitorear la visión local de los agentes, sino de poder interceptar los paquetes enviados por la capa de comunicación de la visión.

Es así que fue desarrollado VisRobUtils como una aplicación de escritorio en Java que permite, mediante una interfaz gráfica, monitorear los valores obtenidos por la visión de los agentes de manera inalámbrica.

3.6.1. Características principales

Como características principales de VisRobUtils, se destacan:

- Monitoreo de *blobs* retornados por HaViMo.
- Monitoreo de posiciones de agentes BIOLOID.
- Monitoreo de mensajes de visión enviados por la capa de comunicación.
- Simulación de agente que trasmite resultados de visión.

[13] Acceder a los resultados del módulo de Visión, mientras es ejecutado
[14] Específicamente, utilizando ROBOPLUS Terminal[19]
[15] También provista por ROBOTIS

- Resultado del proceso de visión, a partir de archivos que contienen información de los *blobs* detectados por la cámara.
- Envío de mensajes genéricos a través de Zigbee.
- Control locomotor básico de agentes bípedos BIOLOID.
- Consola que permite ver en pantalla texto enviado por el robot[16].
- Portabilidad entre sistemas operativos y arquitecturas.

3.6.2. Implementación

Para la implementación de VisRobUtils fueron utilizadas las librerías de acceso al puerto serial brindadas por BIOLOID en [6], así como la librería Gyovinet[21] para obtener los puertos seriales libres del sistema.

Quizás la mayor dificultad encontrada fue lograr la comunicación entre BIOLOID y el computador. Los modelos BIOLOID PREMIUM utilizan como periférico para comunicación Zigbee, a Zig110. Por otro lado, el adaptador Zig2Serial, que provee compatibilidad entre un puerto serial y el dispositivo Zig100, obliga a este último a trabajar en una frecuencia de *broadcast* distinta de la que trabaja Zig110. En razón de lo expuesto fue necesario realizar una modificación sobre el hardware del adaptador Zig2Serial para permitir que Zig100 trabaje en el mismo canal que su contraparte[17].

3.7. Identificación de objetos

La identificación de objetos es la función característica de un módulo de visión. En ella se busca distinguir un conjunto de *blobs* como un objeto conocido. La identificación realizada por el módulo construido se realiza en base a cada imagen obtenida por HaViMo, a partir de la calibración de colores que debe realizarse previamente[18].

3.7.1. Objetos reconocidos

Fue relevante identificar los objetos de la cancha que son de interés para los componentes de estrategia y localización. En primera instancia, los objetos retornados por el módulo de visión consistían en:

- *Pelota*: Única en la cancha, con dimensiones similares a una pelota de tenis, de color anaranjado.
- *Arco*: Tanto el propio como el contrario, de posición fija dentro de la cancha. Se diferencian entre ellos por sus colores.
- *Landmarks:* Postes colocados en los extremos de la línea central de la cancha, fuera del terreno, de colores característicos, utilizados para localización del agente. Su posición es fija en el terreno.
- *Robots aliados*: Robots del equipo del agente, que utilizan los mismos colores que el agente para identificarse.

[16] Haciendo uso de la librería libzgb_console, implementada en conjunto con VisRobUtils

[17] Las modificaciones realizadas fueron las explicitadas por el manual de Zigbee [22] para el cambio de canal de *broadcast*, que consisten en desoldar una sección de la placa.

[18] [5] muestra los pasos necesarios para realizar la calibración de colores para los objetos a reconocer.

- *Robots rivales*: Robots del equipo contrario del agente, que utilizan el color opuesto al del agente para identificarse.

Sin embargo, debido a restricciones en cuanto a la distancia mínima necesaria para identificar el arco, se decidió sumar a la lista anterior el objeto:

- *Poste de arco*: Postes laterales de los arcos, que son de interés cuando no es posible identificar el arco al que pertenecen.

La figura 3.7.1 muestra los objetos reconocibles de la cancha.

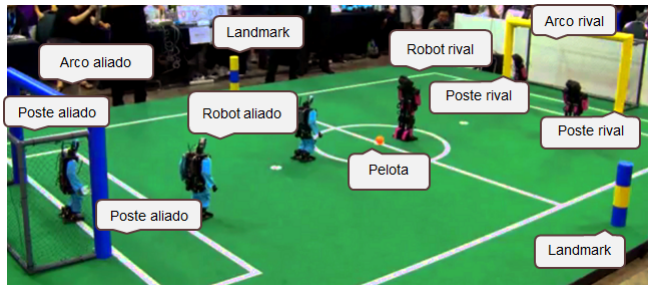

Figura 3.7.1: Objetos reconocibles

A continuación se describen las características de la identificación para cada objeto. En la sección A se presenta el pseudocódigo para cada identificación.

Identificación de Pelota

La identificación de la pelota se realiza tomando el *blob* del color de la pelota, de mayor tamaño, que posea un coeficiente de redondez mayor que el definido según un umbral ajustable.

El coeficiente de redondez se define como

$$ball\_roundness = \frac{min(ball\_width, ball\_height)}{max(ball\_width, ball\_height)}$$

Donde una pelota perfecta debe poseer un coeficiente de redondez igual a uno.

Además, considerando los casos en que la pelota se encuentra tan cerca del robot que no se ve en su totalidad, todos aquellos *blobs* de color configurado para la pelota que se encuentren cercanos al margen inferior de la imagen son completados hasta obtener una forma redonda. Esto es necesario para el modelo vectorial utilizado al obtener la distancia a los objetos, el cual requiere conocer el punto inferior del objeto. Un escenario típico de este problema es el caso cuando la pelota se encuentra a los pies del agente, pero la cámara no puede enfocarla en su totalidad.[19]

En el ejemplo de la figura 3.7.2, el *blob* delineado en rojo, retornado por el hardware de cámara, pasaría a ocupar la totalidad de la pelota.

En los casos que solo se ve parte de la región superior de la pelota, como en la figura 3.7.3, el *blob* completado seguirá sin contener la pelota en su totalidad, puesto que el ancho del *blob* no se corresponde con el ancho verdadero de la bola.

[19] Debido tanto a que el cuello no permite físicamente enfocar la pelota en los pies del robot, o que secciones del robot, como las rodillas, evitan ver la pelota.

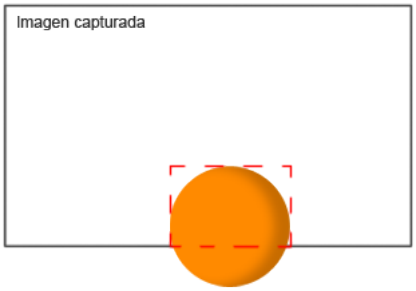

Figura 3.7.2: *Blob* detectado de pelota

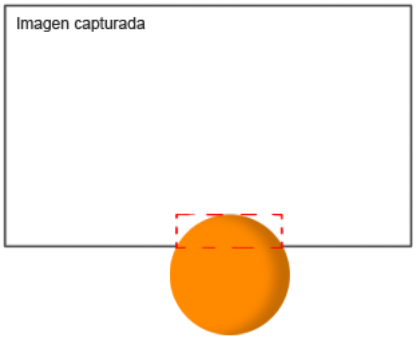

Figura 3.7.3: *Blob* detectado de la parte superior de pelota

Identificación de Landmarks

Los *landmarks* son identificados buscando las secciones que los componen. De las tres secciones, los colores de la sección superior e inferior coinciden, pudiendo variar entre dos colores configurables[20]. El color de la sección del medio es del opuesto. Los *landmarks* se diferencian entre sí, debido a que los colores de las secciones están invertidos para su contraparte.

Es así que para cada *blob* que sea de uno de los colores configurados para los *landmarks*, se intenta buscar las secciones superior e inferior, siendo el umbral de distancia aceptada entre secciones un parámetro ajustable.

No obstante, considerando nuevamente el escenario en que el objeto se encuentra demasiado cerca del robot, se realizan aproximaciones de la altura del *landmark*, a partir de la altura de las secciones que lo componen.

Por último, el módulo no requiere que sean identificadas las tres secciones sino que a partir de dos secciones continuas se intentará identificar si se trata del *landmark* posicionado a la derecha o a la izquierda de la cancha. Esto es llevado a cabo a partir de un método complementario para calcular la distancia a objetos, basado en proporciones[16].

En la figura 3.7.4 se muestra las dos secciones detectadas (A) y los posibles *landmarks* (B) y (C). En este

[20] Actualmente, los colores establecidos para los *landmarks* son amarillo y azul. Los mismos colores que los arcos.

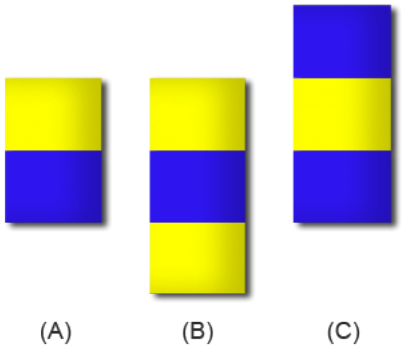

Figura 3.7.4: Posibles *landmarks* según secciones detectadas

caso, se obtiene la distancia a las secciones bajo el modelo de proporciones, y la distancia a los posibles *landmarks* bajo el modelo vectorial. Si la distancia proporcional se corresponde, bajo un umbral ajustable, a una de las distancias, se puede afirmar cuál de los dos *landmarks* se corresponde con las regiones identificadas.

Identificación de Arcos

La identificación de los arcos se realiza a partir de los dos postes que los componen y el travesaño, iterando entre los *blobs* que no han sido correspondidos con un objeto, determinando los mejores postes y travesaño teniendo en cuenta la forma y distancia entre ellos.

Es necesario identificar un poste y un travesaño o los dos postes para obtener los valores requeridos por el modelo vectorial. Sin embargo, para identificar todas las secciones del arco requeridas, es necesario encontrarse a una distancia considerable del arco, por lo que surge la necesidad de brindar datos para la orientación del robot cuando se encuentra dentro del área rival o la propia.

Identificación de Postes

Los postes son retornados únicamente cuando no se identifica el arco correspondiente, permitiendo que se mantenga referencia a los arcos aunque el agente se encuentre muy cerca de ellos. Su identificación consiste en la agrupación menos rigurosa[21] de *blobs* alineados verticalmente, para los colores del arco.

Identificación de Robots

La identificación de robots resulta difícil en la práctica dada la amplia variedad posible de robots y las posiciones en que pueden encontrarse. Existen restricciones en cuanto a la proporción de los miembros del robot respecto a su altura, así como el tamaño de su cabeza y el de sus pies. Además, dichos robots deben llevar marcas de color[22] en sus extremidades para diferenciar a qué equipo pertenecen[9].

[21] Específicamente, aumentando los umbrales para los cuales se acepta que dos *blobs* se encuentra alineados
[22] Cyan o magenta.

En busca de obtener una identificación aceptable para todos los modelos que se puedan diseñar bajo estas restricciones, se utilizó un clasificador bayesiano bajo el concepto de qué probabilidad se tiene de reconocer un conjunto de *blobs* como un robot, dado que el objeto en la imagen es un robot.[2]

Es necesario identificar las secciones que componen un robot, como el cuerpo, los dos brazos y las piernas[23]. Las restricciones que modelan la función de probabilidad son las siguientes:

- Si se ve el centro del robot

    1. El punto medio de ambas piernas debe estar alineado verticalmente con el punto medio del cuerpo. En la figura 3.7.5, $a_x = b_x$.
    2. El ancho de la unión de las piernas debe coincidir con el ancho del cuerpo. En la figura 3.7.5, $f_x - e_x = d_x - c_x$.
    3. El ancho de los brazos debe ser mayor que un tercio del ancho de la unión de las piernas. En la figura 3.7.5, $h_x - g_x > \frac{f_x - e_x}{3}$ y $j_x - i_x > \frac{f_x - e_x}{3}$.
    4. La altura de los brazos debe ser mayor que un tercio de la altura del cuerpo. En la figura 3.7.5, $g_y - k_y > \frac{m_y - c_y}{3}$ y $j_y - l_y > \frac{m_y - c_y}{3}$.

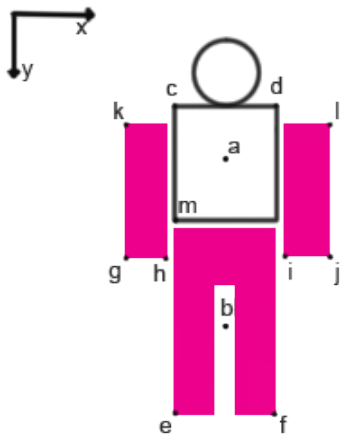

Figura 3.7.5: Identificación de Robots, vista frontal

- Si no se ve el centro del robot

    1. El centro de los brazos y el centro de ambas piernas deben distar en la abscisa X menos de un medio del ancho de la unión de las piernas. En la figura 3.7.6, $a_x - b_x < \frac{d_x - c_x}{2}$.
    2. La separación entre los brazos y las piernas debe ser menor a un medio de la altura de las piernas. En la figura 3.7.6, $f_y - e_y < \frac{c_y - f_y}{2}$.
    3. El ancho de los brazos debe coincidir con el ancho de la unión de las piernas. En la figura 3.7.6, $g_x - e_x = d_x - c_x$.

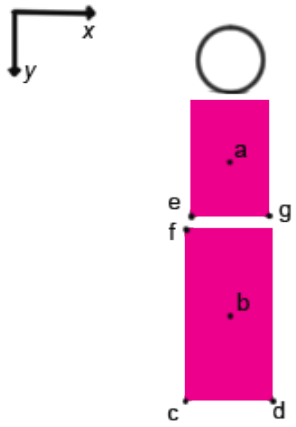

Figura 3.7.6: Identificación de Robots, vista lateral

Como entrada previa se realizan controles sobre las secciones pertenecientes al robot. Como resultado, se retornan todos los robots encontrados junto con su probabilidad[24], no descartando de esta manera posibles robots por muy baja que sea su probabilidad.

3.8. Propiedades de los objetos

La decisión más sensible del módulo de visión fue definir qué propiedades de los objetos de la cancha identificados interesa retornar a quien utilice el módulo de visión. Siendo ésta una decisión que depende del contexto donde se utilice el módulo, se buscó satisfacer los requerimientos genéricos para un componente de estrategia[25].

3.8.1. Propiedades de interés

Las propiedades de interés para componentes de estrategia y localización, se corresponden con:

- Tipo de objeto identificado.
- Valor de la posición del centro del objeto en la abscisa X de la cancha.
- Valor de la posición del centro del objeto en la abscisa Y de la cancha.
- Valor de la posición del centro del objeto en la abscisa Z de la cancha.
- Rotación del objeto en la abscisa X de la cancha.
- Rotación del objeto en la abscisa Y de la cancha.

[23] Las piernas son detectadas como una única región por HaViMo

[24] La cual se corresponde con el nivel de semejanza.

[25] Para ello se consultó a los integrantes del proyecto Salimoo[18], actuales usuarios del módulo de visión, encargados de la implementación de estrategia y locomoción.

- Rotación del objeto en la abscisa Z de la cancha.
- Radio del volumen acotante del objeto.

La figura 3.8.1 muestra los parámetros relacionados a las abscisas de la cancha.

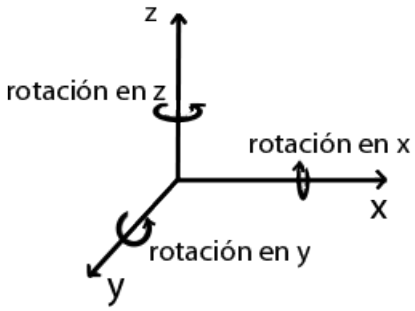

Figura 3.8.1: Parámetros de utilidad para componentes de estrategia y localización.

3.8.2. Propiedades retornadas para objetos identificados

Con el fin de minimizar el tiempo de procesamiento y el flujo de información entre los agentes que requiere la solución, se decidió utilizar los siguientes cuatro parámetros:

Tipo de objeto: Necesario para identificar la naturaleza del objeto y poder conocer otras características implícitas, como sus dimensiones, permitiendo así obtener el radio del volumen acotante, ya que el mismo depende únicamente del tipo de objeto. Esto es posible dado que se trabaja sólo con objetos conocidos.

Coordenada en abscisa X: Requerimiento principal para el módulo de visión, para el escenario del fútbol de robots.

Coordenada en abscisa Y: Requerimiento principal para el módulo de visión, para el escenario del fútbol de robots.

Dadas las características de la visión local y que no se cuenta con un módulo de localización implementado, no es posible obtener las coordenadas para las abscisas X e Y; sin embargo, sí es posible obtener la posición polar de los objetos identificados respecto al robot. Los parámetros retornados entonces por el módulo de visión son:

- *Tipo de objeto*: Perteneciente a alguno de los siguientes:
  - ROBOT ALIADO
  - ROBOT RIVAL
  - PELOTA
  - ARCO ALIADO
  - ARCO RIVAL

- POSTE DE ARCO ALIADO
- POSTE DE ARCO RIVAL
- *LANDMARK* IZQUIERDO
- *LANDMARK* DERECHO.

- *Distancia del agente al objeto*: En centímetros, desde la base del robot a la base del objeto identificado.
- *Ángulo del frente del agente al objeto identificado*: En grados, con 0 en el frente del robot, en sentido antihorario.

Las rotaciones en las abscisas X, Y y Z no pueden ser derivadas a partir de los parámetros anteriores, sin embargo se consideró demasiado costoso obtenerlas a partir del procesamiento de la imagen, por lo que no fueron incluidos como datos retornados. Además, serían de interés únicamente para los objetos detectados como robots. Asimismo, el cálculo de las rotaciones necesita el reconocimiento del frente del objeto, lo que resulta inviable, debido a la simetría entre el frente y la parte de atrás de los robots y a la baja resolución de la cámara HaViMo[26]. Por otra parte, el error del cálculo de las rotaciones dependería en gran medida de la calibración realizada, lo que conlleva a grandes errores en estas medidas, a partir del ruido del ambiente y la calibración, aumentando más aún a medida que crece la distancia al objeto. Por último, mencionar que actualmente la lógica utilizada para las estrategias de fútbol de robots a nivel competitivo consideran a los robots rivales únicamente como obstáculos.

No se consideró necesario retornar el valor de la coordenada en la abscisa Z puesto que, según registros de partidos en competiciones oficiales, los objetos se mantienen constantemente en contacto con el suelo.

3.8.3. Semejanza del objeto

El nivel de exactitud con que un conjunto de *blobs* representan fidedignamente un objeto es una propiedad de interés para componentes superiores. La certeza que el módulo tiene de la identificación realizada se reduce a medida que los *blobs* se alejan de la representación que se utiliza internamente para su objeto.

Es así que como propiedad complementaria para los objetos se retorna:

- *Nivel de semejanza del objeto*: Semejanza entre el conjunto de *blobs* y el modelo interno que se usa para identificar el objeto.

Sistemas con nivel de semejanza permiten no desechar identificaciones con baja similitud al objeto, permitiendo a la estrategia tomar la decisión de utilizarlos o no. Esto permite que módulos superiores tengan noción del error cometido por la identificación, frente al modelo interno del objeto.

La funciones de cálculo de semejanza varían según el objeto a reconocer, dependiendo de características como forma, tamaño, posición, movilidad, etc, reduciendo la semejanza a medida que la configuración de *blobs* que componen un objeto se aleja de dichas características. En la sección B se presenta el pseudocódigo de las funciones de estimación de nivel de semejanza para cada uno de los tipos de objetos retornados.

El valor de la semejanza es propio de cada familia de objetos. Si bien objetos como la pelota tienen un único modelo establecido por *RoboCup Soccer*[9] y deberían tener un alto nivel de semejanza respecto a su modelo interno, objetos más complejos como los robots, para los cuales solo se mencionan restricciones de diseño, no necesariamente deben tener un alto nivel de semejanza, pudiendo variar además entre los distintos equipos.

[26]La resolución de HaViMo 1.5 es 160 x 120 píxeles

3.9. Cooperación entre agentes

La cooperación entre agentes se puede definir con el aumento en la utilidad total de sistema a raíz de un mecanismo cooperativo. El nivel de cooperación alcanzado por el módulo de visión es mínimo en lo que respecta a unificar la información conseguida mediante la comunicación de visión local.

Esto es debido, en gran parte, a la ausencia de un componente de localización que permita acceder a información extra sobre la ubicación de los agentes en el campo. No obstante, los resultados del módulo pueden ser tomados para generar un nivel de cooperación más fuerte por componentes superiores.

En la sección C se presenta el pseudocódigo de referente a la comunicación implementada.

3.9.1. Comunicación de visión como interrupción

Se estudió la posibilidad de que el procesamiento referente a la comunicación de visión fuera ejecutado bajo una interrupción cada vez que fuera recibido un mensaje. Sin embargo, esto no permite controlar el tiempo de ejecución que se utiliza para el procesamiento de mensajes de modo que fue implementado como una función invocada por la estrategia, para que sea este componente el que decida cuándo ejecutarla.

3.9.2. Autoría de mensajes

El uso de un canal común de comunicación, obliga a distinguir qué componente de software es autor de un mensaje. Para ello fue necesario desarrollar un componente que abstraiga el acceso a Zigbee: Filtro de Zigbee.

De esta manera, el componente agrega a los mensajes enviados por el módulo de visión, bits de autenticación, pudiendo así diferenciar los mensajes recibidos[27].

Una vez recibidos los mensajes, los mismos se colocan en dos colas distintas, pudiendo acceder de esta manera a los mensajes de visión, o a mensajes de otros componentes de manera transparente.

Sin embargo, esto lleva a una reducción del espacio de posibles mensajes que puede enviar un componente externo. En [16] se describen las restricciones que deben tener en cuenta componentes superiores para no superponer mensajes de visión con mensajes propios del componente.

3.9.3. Protocolo de comunicación

El protocolo de comunicación utilizado responde a dos etapas de comunicación.

- Distribuir a todos los agentes el resultado del procesamiento de la visión local.
- Recibir los resultados de otros agentes.

Distribución de visión local

La distribución de visión realiza un *broadcast* de la información de los objetos identificados, para una imagen en particular. No todas las imágenes procesadas desembocan en la transmisión de información; la cantidad de imágenes que deben ser procesadas entre *broadcast* de información puede ser configurada.

La distribución de la visión local se realiza entonces siguiendo un enfoque *result sharing*[4], a partir de un primer mensaje (de tipo RESET) que informa del inicio de la transmisión, seguido de varios mensajes (de tipo BROADCAST) con la información de los objetos identificados en la imagen procesada.

[27] Los bits de autenticación no son calculados en función del contenido, sino que se encuentran definidos, siendo configurable su largo y valor.

Recepción de visión externa

Debido a la acotada cantidad de información que puede ser transmitida por Zigbee en un único mensaje, la información de un procesamiento de visión es recibida en varios mensajes. Es así que el componente *Buffer de Objetos*, de la Capa de Comunicación, almacena las secciones de información recibidas, reconstruyendo parcialmente la información transmitida. De esta manera, no es necesario que se complete toda la transmisión para obtener datos útiles de la visión de otros agentes. La información recibida es presentada como la visión de otros agentes, a los componentes superiores.

Cabe destacar que la velocidad de transferencia de Zigbee soporta la velocidad de procesamiento de imágenes.

La velocidad de procesamiento de HaViMo 1.5 es de 8fps máxima, mientras que la velocidad de transferencia en Zig110 es de media 57600 bps[22]. En el peor caso, se transmitirían 29 mensajes[28] de 16 bits. Según la implementación de libzigbee provista por ROBOTIS[29] para la transmisión de datos, los mensajes se transmiten en 2 paquetes para agregar redundancia. De esta manera se transmite $29 * 16\,bits * 2 = 928\,bits$ por iteración. La configuración de la capa de comunicación especifica por defecto la transmisión de información cada 2 segundos:

- Se transmiten $928\,bits$ cada 2 segundos. Compatible con la tasa de transferencia de Zig110.

Sin embargo, es posible especificar que la transmisión de información se realice en cada iteración de visión[30]:

- Se transmiten $7424\,bits$ por segundo. Compatible con la tasa de transferencia de Zig110.

3.9.4. Estructura de mensajes

Para la transmisión de los mensajes de visión fue necesaria su codificación en cadenas de 16 bits, restricción inherente a la tecnología Zigbee.

Los últimos cinco bits de los mensajes de tipo RESET y BROADCAST son utilizados para:

- *Identificador de agente emisor*: Identificador del agente que envía el mensaje, variando entre 0 a CANT_ALIADOS - 1.
- *Bits de autenticación de visión*: Bits definidos para los mensajes de visión, para diferenciarlos de mensajes de otros componentes.

Por defecto, se utilizan dos bits para el identificador y tres para la autenticación, siendo configurable el largo utilizado para cada uno, mientras que su largo conjunto siga siendo de 5 bits. De esta manera se puede obtener mayor cantidad de identificadores de agentes aliados a costo de reducir el espacio de mensajes posibles para otros componentes.

La figura 3.9.1 muestra la estructura de un mensaje de tipo RESET. Los mensajes RESET son enviados al inicio de la difusión de información.

La figura 3.9.2 muestra la estructura de un mensaje de tipo BROADCAST, el cual transmite información parcial de cada propiedad, para cada objeto reconocido, donde:

obj es el tipo de objeto, perteneciendo a:

[28] Cuando se reconoce la máxima cantidad de objetos: 2 aliados, 3 rivales, 1 pelota y 1 arco (o poste). Dada la configuración del campo no es posible reconocer los 2 arcos en una misma imagen tomada dentro del campo.

[29] libzigbee es utilizada como librería base para la transmisión de paquetes a través de Zigbee.

[30] Si bien es posible especificar esta configuración, no es la recomendada.

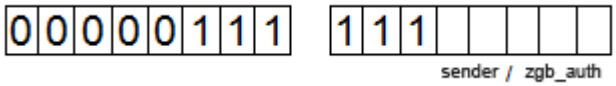

Figura 3.9.1: Mensaje de tipo RESET

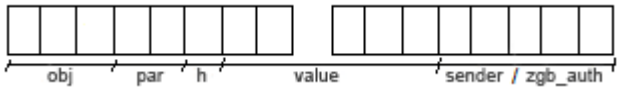

Figura 3.9.2: Mensaje de tipo BROADCAST

0	aliado
1	rival
2	pelota
3	arco aliado
4	arco rival
5	poste aliado
6	poste rival

par es el tipo de parámetro

0	ángulo
1	distancia
2	sin uso - mantenido para extensibilidad
3	sin uso - mantenido para extensibilidad

h especifica si se trata de la parte alta (1) o baja del valor (0)

value es la parte alta o baja del valor del parámetro transmitido.

sender es el identificador del agente que envía el mensaje. De 0 a la cantidad de aliados definida - 1.

zgb_auth es el valor de autenticación de visión.

Debido al tamaño de los mensajes, cada valor de cada parámetro necesita de dos mensajes para ser transmitido, por lo que asumiendo que se ve la máxima cantidad de objetos posibles: una pelota, dos aliados, tres rivales y un arco (o un poste), se necesita de un mensaje de tipo RESET y 28 mensajes de tipo BROADCAST.

3.10. Proceso de desarrollo

Para la realización de la solución se optó por un proceso de desarrollo iterativo incremental compuesto por: Análisis, Diseño, Implementación, Integración y Pruebas:

3.10.1. Análisis del problema

En esta primera etapa se realizó, en conjunto con los tutores, una estimación inicial del tiempo necesario para realizar las tareas planteadas, así como la división de trabajo que involucró:

- Investigación sobre visión por computador y reconstrucción de escenas.
- Investigación y clasificación de algoritmos utilizados para identificación , clasificación y localización de objetos a partir de imágenes, orientado a la liga *RoboCup Soccer.*
- Investigación de sistemas multiagente y cooperación entre agentes.

Fue definido además el alcance del proyecto y la plataforma robótica a utilizar, así como el proveedor de la cámara externa.

3.10.2. Diseño de solución

En la etapa de Diseño, se prosiguió a diseñar la arquitectura de la solución. Paralelamente, fue realizado el primer contacto con la plataforma robótica, iniciando la construcción del robot y el montaje de la cámara en el mismo; efectuando las primeras pruebas sobre la cámara externa.

Dada la naturaleza del problema, la investigación previa realizada, la documentación del hardware a utilizar y las pruebas efectuadas sobre la cámara se llegó a la conclusión de realizar una implementación dividida en tres capas:

- Capa de acceso a HaViMo
- Capa de Visión
- Capa de Comunicación

La división en capas realizada añade varias fortalezas a la solución, como son la modularización de la aplicación, la división del problema en subproblemas más sencillos y la posibilidad de paralelizar el trabajo, así como la reutilización de las mismas.

3.10.3. Implementación

En la etapa de Implementación, se desarrollaron las capas mencionadas, en forma paralela. Se realizó por un lado la Capa de acceso a HaViMo y la Capa de Comunicación, y por otro lado la Capa de Visión. Tanto la Capa de Visión como la Capa de Comunicación fueron realizadas por fuera del contexto del laboratorio, es decir, sin utilizar la plataforma robótica, buscando aumentar la productividad:

- Para la implementación de la Capa de Visión se utilizó la aplicación VisRobUtils (ver sección 3.6), utilizando como ejemplos de entrada, lecturas de la capa de acceso a HaViMo obtenidas en el laboratorio, para escenas genéricas, así como particulares.
- Para la implementación de la Capa de Comunicación se utilizó como *stub* de la Capa de Visión, archivos con valores preestablecidos, y para suplir el acceso a Zigbee se desarrolló una librería que implementa las mismas operaciones que presenta la librería brindada por ROBOTIS, basada en *sockets*.

Consecuentemente, se tuvo que dedicar tiempo para portar la implementación realizada, a la plataforma robótica final.

3.10.4. Integración

La integración de las distintas capas fue realizada siguiendo un modelo *bottom up*[31], integrando en una primera etapa la Capa de acceso a HaViMo con la Capa de Visión. Una vez realizadas las pruebas de integración correspondientes, se procedió por último a integrar la Capa de Comunicación, con el resultado de la integración anterior, realizando luego las pruebas pertinentes.

La etapa de Integración fue además la instancia donde se migró del ambiente de simuladores a la plataforma robótica final.

3.10.5. Pruebas

En la etapa de Pruebas se definió un conjunto de pruebas del sistema necesarias para comprobar el correcto funcionamiento del módulo.

Dichas pruebas fueron efectuadas para todos los objetos de la cancha (Pelota - *Landmarks* - Arco - Poste - Robots), consistiendo en un conjunto de pruebas elementales del sistema, y un conjunto de escenarios de interés, donde las pruebas elementales del sistema fueron realizadas para cada uno de los escenarios identificados. Basándose en la experiencia de cómo fue construido el módulo y en videos de finales de *RoboCup Soccer* de años anteriores, se buscó abarcar los casos más interesantes en el contexto de fútbol de robots.

Ante lo expuesto, fue de especial interés obtener información del comportamiento del módulo en dichos escenarios de manera que, quien utilice el módulo de visión, conozca el estado en el que quedaría el mismo y pueda así reaccionar de la mejor manera frente a dichos casos.

Como complemento a la aplicación VisRobUtils, para la verificación del protocolo fue construido un *stub*[32] correspondiente a Zigbee que permite independizarse del hardware de comunicación. El *stub* provee las funcionalidades básicas, como son el envío y recepción de mensajes entre distintas instancias del módulo. Utilizando comunicación mediante *sockets*, se buscó representar fielmente la implementación de la librería de ROBOTIS que administra el acceso al dispositivo; sin embargo no fueron reproducidos los problemas clásicos de comunicación como son ruido en el canal o retraso de paquetes entre otros. De esta manera fue posible depurar el protocolo de intercambio de mensajes y los resultados obtenidos por la Capa de Comunicación, a partir de datos de visión local obtenidos de un archivo de configuración.

[31] `http://en.wikipedia.org/wiki/Top-down_and_bottom-up_design`
[32] Código que simula la actividad de componentes faltantes.

Capítulo 4

Pruebas sobre el sistema

En este capítulo se describen las pruebas realizadas sobre el sistema para verificar y validar los componentes y el proceso de Visión implementado. En la sección 4.1 se describen los tipos de pruebas realizadas. En las secciones siguientes se muestra el resultado obtenido para:

- Pruebas de visión con un agente: Pruebas esenciales del sistema. Sección 4.2.
- Pruebas de visión multiagente: Pruebas necesarias para asegurar la comunicación entre agentes. Sección 4.3.
- Pruebas complementarias: Pruebas que definen las características del módulo construido. Sección 4.4.

4.1. Tipos de pruebas definidas

4.1.1. Pruebas de visión con un agente

Con el objetivo de probar el módulo desarrollado se definió un conjunto de pruebas elementales a realizarse para todos los objetos del ambiente.

- *Reconocimiento de objetos estáticos, desde una posición estática*: Representando la configuración ideal para el uso del módulo.
- *Reconocimiento de robot y pelota en movimiento, agente estático*: Representando la configuración estándar en un campo de fútbol de robots.
- *Reconocimiento de objetos estáticos, mientras el agente se mueve*: Abarcando las configuraciones de escenarios más difíciles para el uso del módulo.

Estas pruebas elementales fueron ejecutadas en un conjunto predefinido de escenarios.

- *Escenario común*: El robot se encuentra erguido.
- *Agente caído*: El robot tiene la cámara orientada contra el campo o lo ve de costado, o hacia arriba. Este es un escenario típico en un partido de la liga humanoide de *RoboCup Soccer*.

- *Ingreso espontáneo de un agente al campo*: Dentro de las reglas de *RoboCup Soccer*, es posible que un agente ingrese de manera espontánea[1].
- *Apagado espontáneo de un agente aliado en el campo*: Para probar casos de mal funcionamiento de agentes aliados.

La combinación objetos en movimiento con el agente en movimiento no fue realizada debido a que no varía de la combinación objetos estáticos, mientras el agente se mueve, donde se realizan varias pruebas variando la posición de los objetos.

4.1.2. Pruebas de visión multiagente

Las pruebas de visión con varios módulos de visión fueron realizadas con el fin de rectificar la Capa de Comunicación, verificando la correcta comunicación de mensajes entre agentes aliados. Los escenarios definidos se corresponden con:

- *Un agente en el campo, simulando aliados*: Al agente se le envía la simulación de una iteración de la Capa de comunicación en la que un aliado ve solamente el objeto pelota.
- *Dos agentes en el campo*: Ambos agentes son calibrados para ver el mismo objeto.
- *Tres agentes en el campo*: Utilizando un *stub* de la Capa de Visión, dos agentes envían información de visión a un tercero.

4.1.3. Pruebas complementarias

A su vez, se realizó un conjunto complementario de pruebas para identificar las características del módulo en el campo de juego que incluyen:

- *Verificación del modelo vectorial*: Realizado a partir de un objeto simple. El modelo vectorial obtiene la distancia a los objetos sin importar su forma, por lo que es posible abstraerse de los objetos a identificar, verificando así la correctitud del modelo.
- *Identificación del error del modelo vectorial, con el agente en movimiento*: Utilizando un objeto simple. El error del modelo vectorial no depende de la forma del objeto, sino de los valores de ángulo de *pan* y *tilt* con que es utilizado. Cabe destacar que el error será característico de la implementación de la caminata del robot.
- *Umbral de reconocimiento para cada objeto*: Se busca identificar la cota máxima de distancia a la que se puede detectar un objeto. De interés tanto para componentes de estrategia como de localización.
- *Identificación de un robot desde todos los puntos de vista*: Buscando puntos de vista donde no se detecta un robot.
- *Identificación de menor ángulo de reconocimiento de arco*: Buscando el ángulo en el que el módulo no identifica el arco.
- *Identificación de dos robots superpuestos*: Buscando la mínima distancia entre robots, para que sean detectados como uno único.

[1] Durante un partido de la liga humanoide, los robots pueden abandonar el campo debido a mal funcionamiento y regresar.

4.2. Resultado de las pruebas de visión con un agente

En las secciones siguientes se describen los resultados de las pruebas para cada uno de los escenarios definidos:

4.2.1. Escenario común

Las pruebas fueron realizadas para los objetos pelota, *landmark* , arco, agente y postes, distribuidos en distintas iteraciones. Para las pruebas donde se utilizan objetos estáticos, los datos devueltos por el módulo de visión fueron comparados con las medidas exactas en la que se encontraban los objetos. Para las pruebas con objetos en movimiento, se tomó la distancia del agente, a ciertos puntos de su trayectoria.

Reconocimiento de objetos estáticos, desde una posición estática

A continuación se describen las distintas configuraciones de pruebas para el reconocimiento de objetos estáticos con el agente quieto. En cada caso se plantea un cuadro con los resultados obtenidos, el cual contiene:

- *Objeto identificado*.
- *Distancia real al objeto*. Desde la base del robot, en centímetros.
- *Ángulo real al objeto en grados*. Desde el frente del robot, en sentido antihorario.
- *Porcentaje de iteraciones de la prueba en las cuales fue reconocido el objeto*.
- *Promedio de distancia detectada al objeto*.
- *Promedio de ángulo detectado al objeto*.
- *Promedio de nivel de semejanza*. Donde 0 marca la menor semejanza, y 15 la mayor.[2]

La elección de la posición de los objetos en los escenarios fue arbitraria en vista que la detección de los objetos no depende de su posición. La verificación del cálculo de distancia y ángulo a los objetos, donde sí es de interés la posición del objeto a identificar, es abordado en la sección 4.4.1.

Un rival y una pelota.

- 1 rival a 126 cm, 7.4 grados izquierda
- 1 pelota a 89 cm, 18.3 grados derecha

El cuadro 4.1 muestra el resultado promedio de 75 pruebas. Los resultados respecto al porcentaje de detección tanto para el robot rival como la pelota son altos (mayor a 70 %).

El bajo nivel de semejanza para el rival muestra la diferencia entre el modelo interno perfecto de un robot humanoide, y la agrupación de *blobs* detectada por la cámara.

[2] El valor del nivel máximo de semejanza es configurable.

Objeto	Distancia	Ángulo	% de detección	Distancia detectada	Ángulo detectado	Semejanza
Rival	126 cm	+ 7.4	73 %	122 cm	+ 6.9	2.2
Pelota	89 cm	- 18.3	82.5 %	86.2 cm	- 19.3	14.9

Cuadro 4.1: Escenario común, primer iteración.

Un *landmark*

- 1 *landmark* a 134.5 cm, 11 grados izquierda

El cuadro 4.2 muestra el resultado promedio de 75 pruebas. El porcentaje de identificación del objeto *landmark* decrece en cuanto al escenario anterior. El *landmark* es un objeto compuesto y necesita de la identificación de más de una zona para poder ser reconocido.

Objeto	Distancia	Ángulo	% de detección	Distancia detectada	Ángulo detectado	Semejanza
Landmark	134.5 cm	+ 11	53 %	130.9 cm	+ 11	15

Cuadro 4.2: Escenario común, segunda iteración.

Un arco

- Arco a 224 cm, 5 grados izquierda

El cuadro 4.3 muestra el resultado promedio de 100 pruebas. El arco presenta el peor valor de porcentaje de detección de los objetos del campo. Su gran tamaño y su forma rectangular son retornados intermitentemente por HaViMo como dos postes y un travesaño, o como un rectángulo que engloba el arco.

Objeto	Distancia	Ángulo	% de detección	Distancia detectada	Ángulo detectado	Semejanza
Arco	224 cm	+ 5	39 %	250 cm	+ 1	13

Cuadro 4.3: Escenario común, tercer iteración.

Un poste de arco

- Poste a 92 centímetros, 0 grados

El cuadro 4.4 muestra el resultados promedio de 75 pruebas. El porcentaje de detección de los postes de arco resulta el mejor de los demás objetos. Es posible referirse a este objeto cuando la identificación del arco falla.

Objeto	Distancia	Ángulo	% de detección	Distancia detectada	Ángulo detectado	Semejanza
Poste	92 cm	0	88 %	94 cm	- 0.5	4

Cuadro 4.4: Escenario común, cuarta iteración.

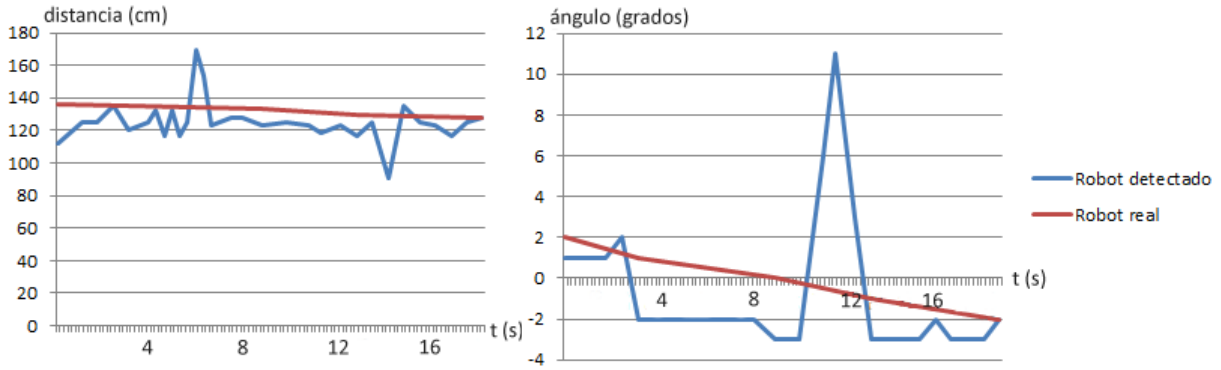

Figura 4.2.1: Robot externo en movimiento.
A la izquierda distancia, a la derecha ángulo.

Reconocimiento de robot y pelota en movimiento, agente estático

Robot externo en movimiento La figura 4.2.1 muestra la distancia y el ángulo detectados para un robot externo al agente en movimiento, en función del tiempo, contra la distancia y ángulos reales.

A una distancia de 120 centímetros, el error de la distancia sobrepasa en dos lecturas los 30 centímetros de error, mientras que en los demás puntos de la trayectoria el error oscila en valores menores a 20 centímetros.

En la detección del ángulo de un robot en movimiento se detecta un pico que sobrepasa los 10 grados de error.

Pelota en movimiento La figura 4.2.2 describe la distancia y el ángulo detectados por el módulo de visión para una pelota durante su trayectoria en función del tiempo.

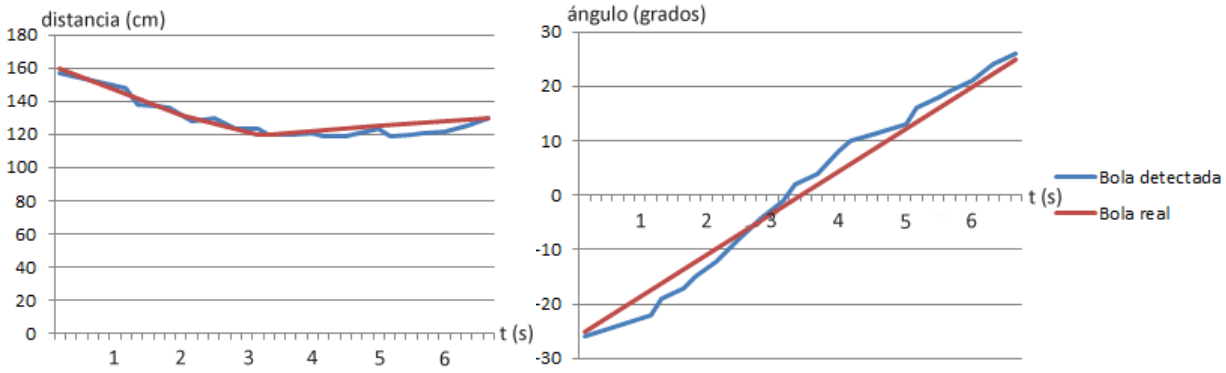

Figura 4.2.2: Pelota en movimiento.
A la izquierda distancia, a la derecha ángulo.

La distancia detectada para la pelota no sobrepasa en ningún momento 10 centímetros de error, mientras

que el ángulo detectado contiene errores menores a 5 grados, considerando que la pelota se encuentra a distancias mayores a 1 metro.

La pelota es el mejor objeto identificado, obteniendo resultados alentadores.

Reconocimiento de objetos estáticos, mientras el agente se mueve

Pelota La figura 4.2.3 muestra la distancia y ángulo percibido para una pelota en reposo, mientras el agente se aleja de ella.

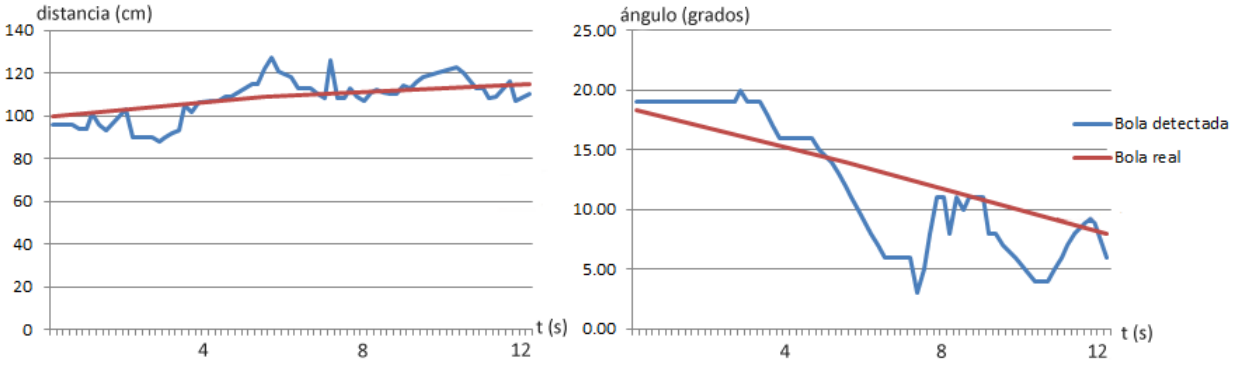

Figura 4.2.3: Distancia y ángulo a pelota estática, con agente en movimiento.

El error de la detección de distancia no sobrepasa los 20 centímetros, mientras que en la detección del ángulo se perciben errores menores a 10 grados, a más de 1 metro de separación entre el agente y el objeto.

Robot La figura 4.2.4 describe la distancia y ángulos detectados para un robot rival, mientras el agente se acera al mismo en línea recta.

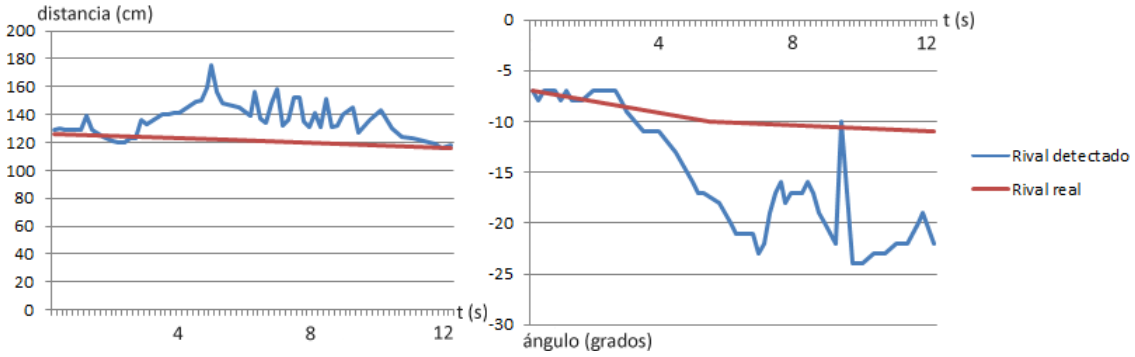

Figura 4.2.4: Distancia y ángulo a robot estático, con agente en movimiento.

El error en la distancia durante la trayectoria no llega a superar los 50 centímetros de error a una distancia de más de 1 metro; y casi la totalidad de los valores obtenidos son mayores que la distancia real al robot.

Por otro lado, el ángulo obtenido para el robot rival percibe errores menores a 15 grados.

La implementación de la caminata del agente juega un papel importante en el error cometido tanto para el cálculo de la distancia como para el ángulo, debido a que el movimiento inherente de la caminata conlleva el movimiento de la cámara, de la cual se asume que su altura y ángulos de orientación son estáticos.

En la gráfica del ángulo de 4.2.4, se observan grandes variaciones en la medida del ángulo, variando 5 grados por momentos. Esto puede deberse a que el movimiento inherente de la caminata mueve el frente del robot junto con la cámara.

Landmark La figura 4.2.5 muestra la distancia y ángulos percibidos por el módulo de visión, para un *landmark*, mientras el agente se dirige a él en línea recta.

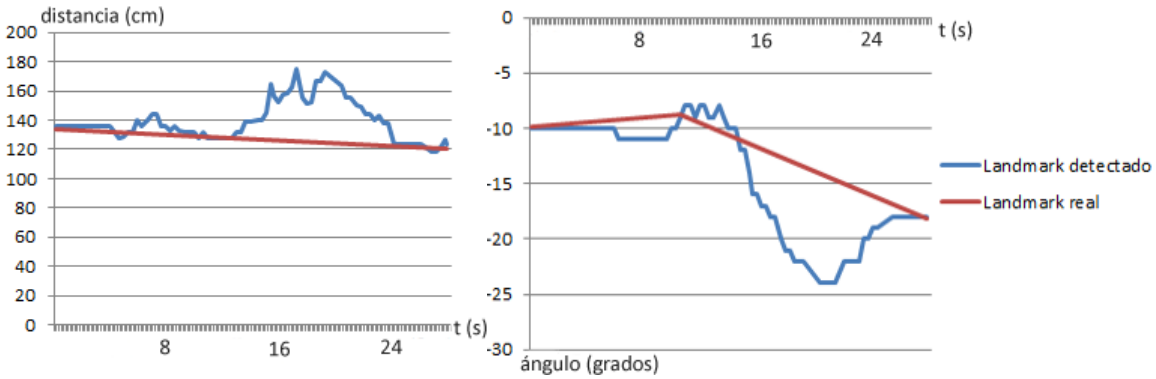

Figura 4.2.5: Distancia y ángulo a *landmark*, con agente en movimiento.

Al igual que para el robot rival en la figura 4.2.4, se observa un error menor a 50 centímetros mientras el agente avanza; de una manera similar, se observa también que casi la totalidad de las lecturas muestran una distancia superior a la real.

En lo que respecta al ángulo obtenido, se observa un error menor a 10 grados.

Es interesante la relación de los picos de error detectados tanto para la distancia y ángulo del *landmark*. Ambos picos comienzan en $t = 14$ y culminan en $t = 25$. Una explicación factible es que hubo una variación en la orientación de la cámara durante la trayectoria del robot debido a la implementación de la caminata[3].

Arco La figura 4.2.6 describe los datos obtenidos de distancia y ángulo para un arco, mientras el agente avanza hacia él en línea recta.

El error en la distancia obtenida para el arco es la peor de todos los objetos estudiados. A una distancia mayor de 2 metros, se detectan tres picos donde el máximo error cometido se corresponde con 3,5 metros. Para las demás lecturas, el error no sobrepasa 50 centímetros.

Por otro lado, el error cometido para el ángulo no sobrepasa 5 grados, siendo el segundo objeto con el menor error de ángulo, mientras el agente se encuentra en movimiento. Esto se debe al gran tamaño del arco que presenta sencilla la labor de detección de ángulo. Sin embargo, como se ve para la distancia calculada el error cometido aumenta considerablemente, siendo proporcional a la distancia a la que se encuentra el agente del arco.

[3]La orientación de la cámara se consideró constante en todas las pruebas.

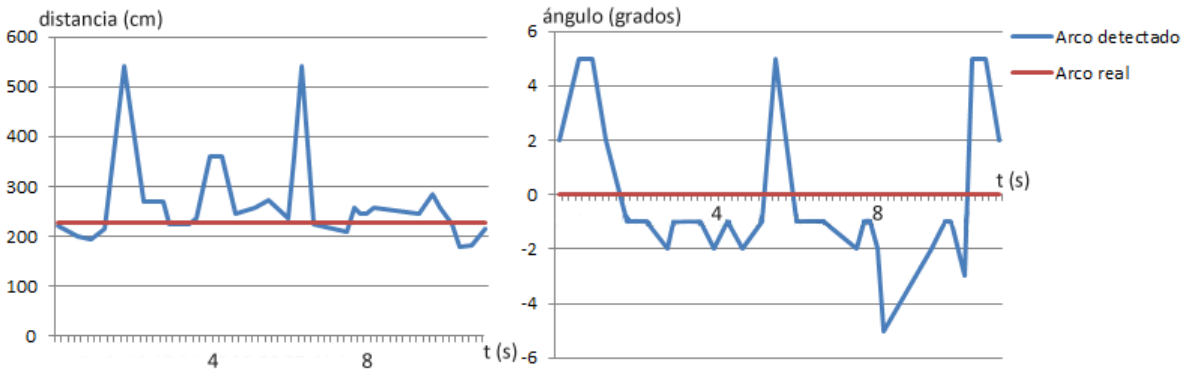

Figura 4.2.6: Distancia y ángulo a arco, con agente en movimiento

Poste La figura 4.2.7 muestra la distancia y ángulo retornados por el módulo para el objeto poste, a medida que el agente avanza hacia él.

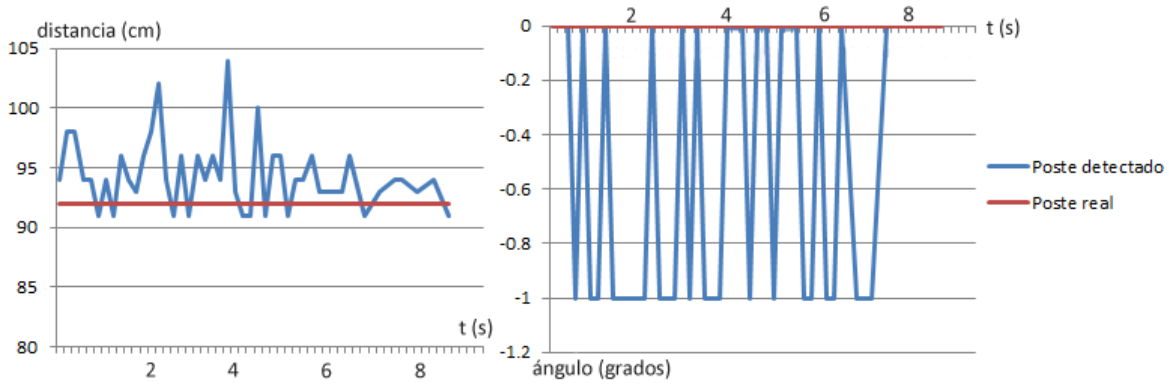

Figura 4.2.7: Distancia y ángulo a poste, con agente en movimiento

La distancia percibida para el poste oscila en 10 centímetros de error, a una distancia aproximada de 90 centímetros. Para el ángulo detectado, se obtienen buenos resultados, con error máximo de 1 grado. Esto es debido al gran tamaño del objeto a identificar, al igual que para el arco.

A partir del conjunto de pruebas realizadas para el escenario común, es posible afirmar que el módulo de visión presenta resultados mayores a 70 % en cuanto a identificación de objetos para la mayoría de los casos, siendo el peor caso la detección del arco, con un porcentaje de identificación menor a 40 %. En los casos donde el agente se encontraba en posición estática, el modelo vectorial obtuvo de forma correcta el ángulo y distancia a los objetos, con errores mínimos. Sin embargo, para las pruebas del agente en movimiento, la implementación de la caminata que conlleva el error en los datos de entrada del modelo vectorial junto con el ruido de fondo mostraron un aumento del error del cálculo de distancia y ángulo.

4.2.2. Agente caído

Se verifica la ejecución del módulo cuando el agente se encuentra caído.

Resultado

El caso del agente caído anula la visión local. Para ningún caso probado, el agente retorna ángulo o distancia correctos. Debido al modelo utilizado, el módulo requiere identificar correctamente las coordenadas de la base del objeto sobre el suelo, lo cual no es posible cuando el robot no se encuentra erguido.

A continuación se describen los resultados con el agente caído, aún cuando puede enfocar los objetos[4].

- *Pelota*: Identificación en 80 % de los casos. Distancia y ángulo n/a.
- *Robot*: Identificación en 20 % de los casos. Distancia y ángulo n/a.
- *Landmark*: No identificado. Distancia y ángulo n/a.
- *Arco*: Identificación en 20 % de los casos, específicamente para los casos donde sólo se identifica el travesaño y un poste lateral. Distancia y ángulo n/a.
- *Poste*: Identificación en 30 % de los casos. Distancia y ángulo n/a.

4.2.3. Ingreso espontáneo de un agente aliado al campo

Durante la ejecución del módulo de visión, se procedió a iniciar un segundo módulo de visión residente en otro robot de iguales características.

Resultado

El primer módulo, recibe la información de visión del nuevo agente.

El segundo módulo recibe la información del módulo que ejecutaba previamente. El agente envía información de los objetos percibidos.

4.2.4. Apagado espontáneo de un agente aliado en el campo

Durante la ejecución de dos módulo de visión en agentes independientes, se apaga uno de ellos.

Resultado

El módulo de visión restante desecha la información recibida luego de que vence el *time to live* configurado para el último mensaje.[5]

[4] Sin embargo, la imagen capturada está rotada

[5] Todos los mensajes recibidos poseen un tiempo de vida durante el cual son válidos. Por más información referirse a [16].

4.3. Resultado de las pruebas de visión multiagente

4.3.1. Un agente en el campo

Al agente se le envía la simulación de una iteración de la capa de comunicación en la que un aliado ve solamente el objeto pelota. Se consulta el modelo del agente, comprobando que la información sea la que se le envió. Se controla que el agente realice el *broadcast* de visión local como fue configurado.

Resultado

Prueba satisfactoria.

Se recibió correctamente la información de visión del objeto pelota. Se verificó el envío de valores positivos y negativos para ángulos.

4.3.2. Dos agentes en el campo

Ambos agentes son calibrados para ver el mismo objeto, se comprueba el modelo que tiene uno de los agentes sobre los objetos que ve el otro. Se debe controlar el *time to live* de los mensajes enviados por los agentes.

Resultado

Prueba satisfactoria.

Los valores recibidos por un agente fueron los detectados por el segundo.

4.3.3. Tres agentes en el campo

Utilizando un *stub* de la Capa de Visión, dos agentes envían información de visión local. Se debe verificar el modelo del tercer agente y los mensajes recibidos.

Resultado

Prueba satisfactoria.

La información recibida por el tercer agente se corresponde con los objetos identificados en los otros dos.

4.4. Resultado de pruebas complementarias

4.4.1. Verificación del modelo vectorial

Utilizando un cubo de color, se procedió a verificar las medidas obtenidas por el modelo vectorial.

Como el modelo no depende de la forma del objeto, sino de las coordenadas de un píxel en la imagen que se corresponde con la base del objeto, es posible abstraerse de los objetos a identificar. Así mismo, una vez verificada la correctitud del método, es posible mantener el ángulo de la cámara constante, ya que el resultado sería el mismo que el estudiado en esta sección.

Se realizaron pruebas, con un único objeto, variando el ángulo de orientación de la cámara con las siguientes combinaciones[6]:

- *Prueba 1 - pan 0 grados, tilt 0 grados*: Figura 4.4.3.
- *Prueba 2 - pan 0 grados, tilt 6 grados*: Figura 4.4.4.
- *Prueba 3 - pan 0 grados, tilt -10 grados*: Figura 4.4.5.
- *Prueba 4 - pan 0 grados, tilt 3 grados*: Figura 4.4.6.
- *Prueba 5 - pan 30 grados, tilt 0 grados*: Figura 4.4.7.
- *Prueba 6 - pan 16 grados, tilt 0 grados*: Figura 4.4.8.
- *Prueba 7 - pan 25 grados, tilt 36 grados*: Figura 4.4.9.
- *Prueba 8 - pan 25 grados, tilt -36 grados*: Figura 4.4.10.

Para todas las pruebas, el cubo se encontraba a 116 centímetros de la base del agente y -4 grados al frente.

El cuadro 4.5 resume los valores obtenidos para las pruebas realizadas, donde se especifica para cada prueba los siguientes valores:

- *Distancia detectada*: Promedio de distancia detectada teniendo en cuenta todas las iteraciones.
- *Ángulo detectado*: Promedio de ángulo detectado teniendo en cuenta todas las iteraciones.
- *Máximo error de distancia*: Máximo error cometido por el módulo en una iteración para el cálculo de distancia.
- *Máximo error de ángulo*: Máximo error cometido por el módulo en una iteración para el cálculo de ángulo.
- *Desviación estándar para distancia detectada*: Desviación estándar para el promedio de distancia detectada.
- *Desviación estándar para ángulo detectado*: Desviación estándar para el promedio de ángulo detectado.

Los resultados de la mayoría de las pruebas resultan prometedores.

Para el caso de la distancia detectada, el promedio de valores detectados en cada orientación no sobrepasa 4 centímetros de error; sin embargo, el máximo error de distancia se corresponde con 8,5 centímetros.

Algo similar ocurre respecto al ángulo detectado. El error del promedio del ángulo detectado en cada orientación es menor a 1 grado, mientras que el máximo error unitario es 1 grado.

Podemos afirmar que el resultado obtenido para el ángulo por el modelo vectorial es robusto, pero a partir de éstas pruebas y anteriores que involucran el cálculo de la posición de los objetos (específicamente en la sección 4.2.1), el error cometido al calcular la distancia aumenta a medida que los objetos se separan de la cámara.

En vista del resultado obtenido sobre el error cometido para el promedio de iteraciones y el máximo error unitario, resulta sensato no utilizar los valores unitarios de la posición a los objetos, sino un promedio de varias iteraciones, mitigando así posibles situaciones de ruido.

[6]No se realizaron pruebas para ángulos de *pan* negativos, ya que eso implicaría que la cámara está orientada mirando el techo.

Prueba	Dist. detect.	Áng. detect.	Máx error dist.	Máx error áng.	Desv Estándar dist.	Desv Estándar áng.
1	115 cm	-4 grados	3 cm	1 grado	0.86 cm	0.17 grados
2	120 cm	-4 grados	8,5 cm	0 grado	3.1 cm	0 grados
3	115 cm	-3,5 grados	2 cm	1 grado	0.95 cm	0.5 grados
4	118 cm	-4 grados	5 cm	0 grado	2.0 cm	0 grados
5	113 cm	-4 grados	3 cm	0 grado	0 cm	0 grados
6	119 cm	-4 grados	3 cm	0 grado	0 cm	0 grados
7	120 cm	-4 grados	6 cm	0 grado	0.34 cm	0 grados
8	120 cm	-4 grados	6 cm	0 grado	0.28 cm	0 grados

Cuadro 4.5: Verificación de modelo vectorial.

La figura 4.4.1 muestra los errores cometidos en el calculo de la distancia promediando iteraciones, para las distintas pruebas realizadas. A medida que aumenta la cantidad de iteraciones que se agrupan se observa la reducción del error cometido en la mayoría de las pruebas.

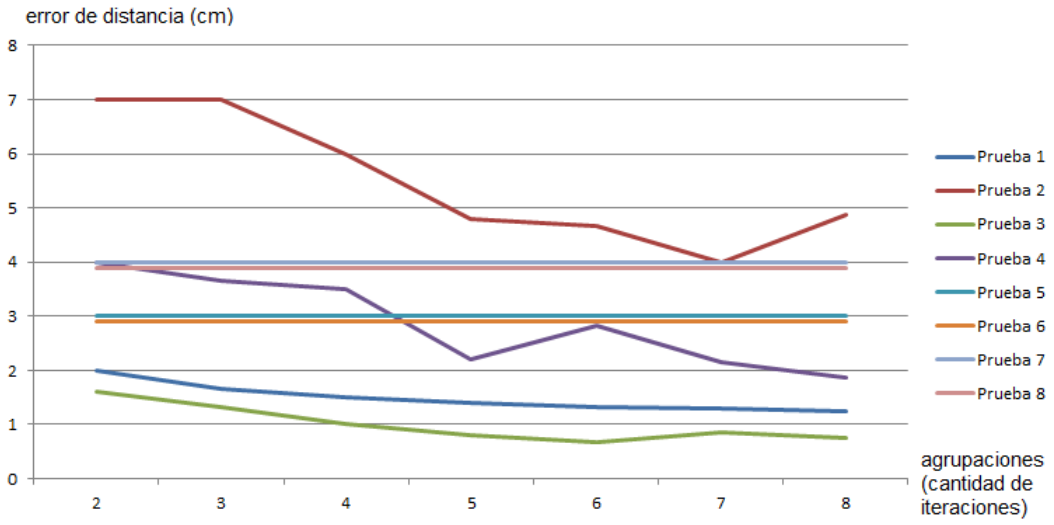

Figura 4.4.1: Errores de distancia según distintas agrupaciones de iteraciones

La figura 4.4.2 muestra el promedio de los errores cometidos a partir de agrupación de iteraciones, en las distintas pruebas. A partir de la agrupación de 5 iteraciones, la reducción del error cometido es menos notable que para agrupaciones de menor cantidad de iteraciones.

La agrupación de iteraciones permite sacrificar cantidad de *frames* útiles procesados por la visión, para reducir el error cometido. En vista de que el error del cálculo de la distancia para las pruebas es menor a 10 centímetros, la agrupación de iteraciones debería ser realizada cuando es de interés conocer la distancia exacta de objetos cercanos al robot[7].

[7]Por ejemplo, cuando interesa acercarse a una distancia establecida a la pelota para patearla.

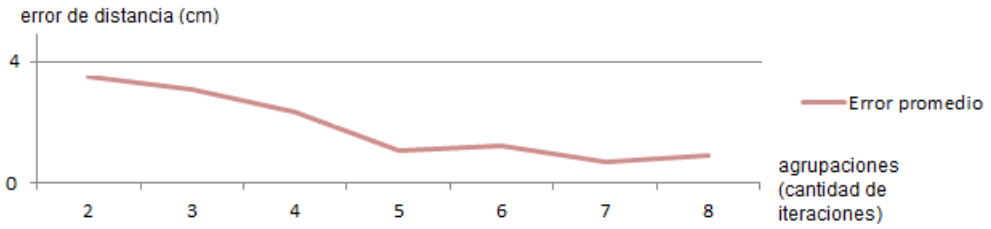

Figura 4.4.2: Promedio de error cometido en pruebas

4.4.2. Identificación del error del modelo vectorial, con el agente en movimiento

Se realizaron experimentos de detección de distancia y ángulo a un cubo, con el robot desplazándose hacia el objeto.

Las figuras 4.4.11 y 4.4.12 muestran los resultados obtenidos en las cinco pruebas realizadas, para la distancia y ángulo, respectivamente. Para la caminata, se utilizó la implementación realizada por el proyecto de grado Salimoo[18]. Los resultados aquí descritos son específicos para dicha implementación, y el error depende en gran medida de la rotación inherente que sufre la cámara durante la trayectoria.

El error cometido en el cálculo de distancia detectada en la mayoría de las pruebas resulta considerablemente alto. Para las pruebas 1, 2 y 4 se detectan picos de más de 1 metro de error. Sin embargo, a medida que el agente se acerca al objeto todas las pruebas obtienen una mejor estimación de la distancia, siendo el caso de mayor error en el final de la trayectoria solamente de 15 centímetros de error. La disminución del error a medida que se acercan los objetos al robot se destaca en todas las pruebas anteriores.

Por otro lado, en lo que respecta al cálculo del ángulo, si bien se presenta un mejor resultado que para la distancia, el error cometido durante la trayectoria resulta constante a medida que el agente se acerca al objeto, siendo el mayor error detectado 15 grados, mientras que el error promedio resulta de 5 grados. El pico de error cometido para el ángulo sucede en la misma iteración en que se encuentra el segundo pico más alto de error en la distancia.

En contraparte con los resultados obtenidos en 4.4.1, los errores cometidos, tanto para la distancia y ángulo para el caso del agente en movimiento, resultan excesivos. Esto se debe principalmente a que durante la trayectoria, se consideró que la orientación de la cámara era constante, cuando en realidad durante la caminata, la cámara se encuentra en constante variación de orientación[8].

De esta manera, se refuerza la necesidad de que el sistema debe obtener en todo momento la orientación de la cámara, a fin de brindar resultados aceptables como los presentados en 4.4.1.

4.4.3. Umbral de reconocimiento para cada objeto

Se busca especificar el umbral de distancia a partir del cual es posible reconocer los distintos objetos, más allá del nivel de semejanza sobre la identificación realizada. Los resultados mostrados son específicos del hardware de cámara utilizado[9].

A partir de pruebas realizadas se obtuvieron los siguientes resultados:

[8]La caminata implementada modifica la orientación de la cámara tanto en los ángulos de *pan* como de *tilt*.

[9]HaViMo 1.5 presenta una resolución de 160 x 120 píxeles. Cámaras con mayor resolución permitirían aumentar la distancia máxima de reconocimiento de objetos.

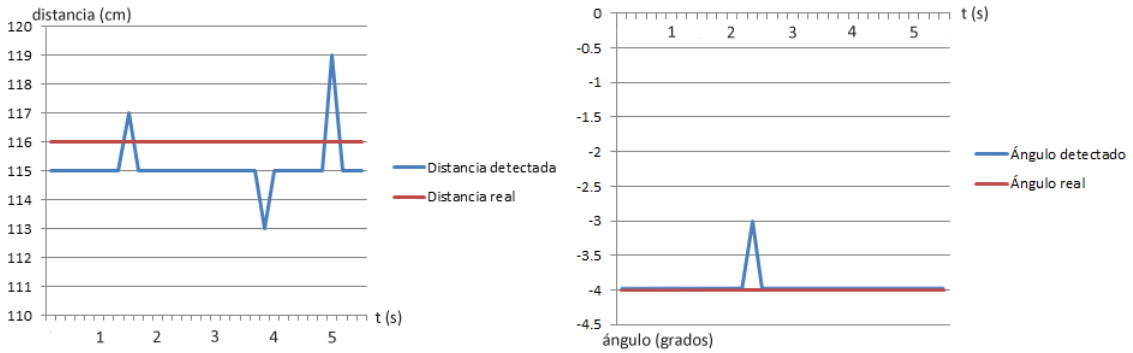

Figura 4.4.3: Modelo vectorial - Prueba 1.

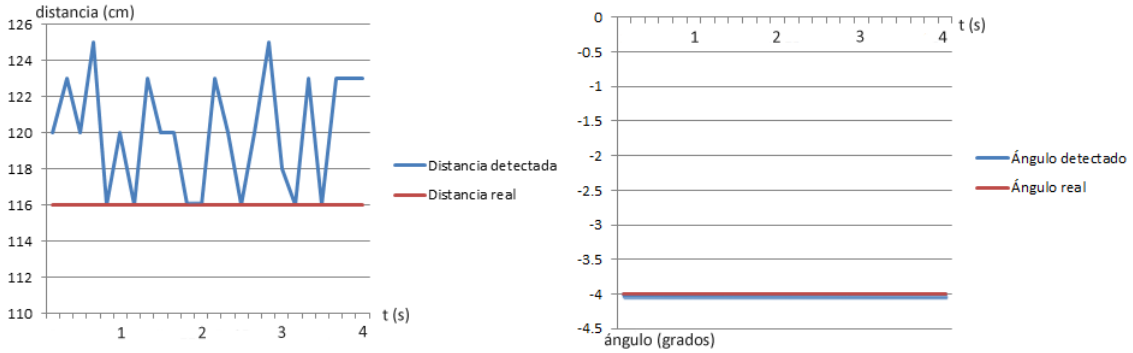

Figura 4.4.4: Modelo vectorial - Prueba 2.

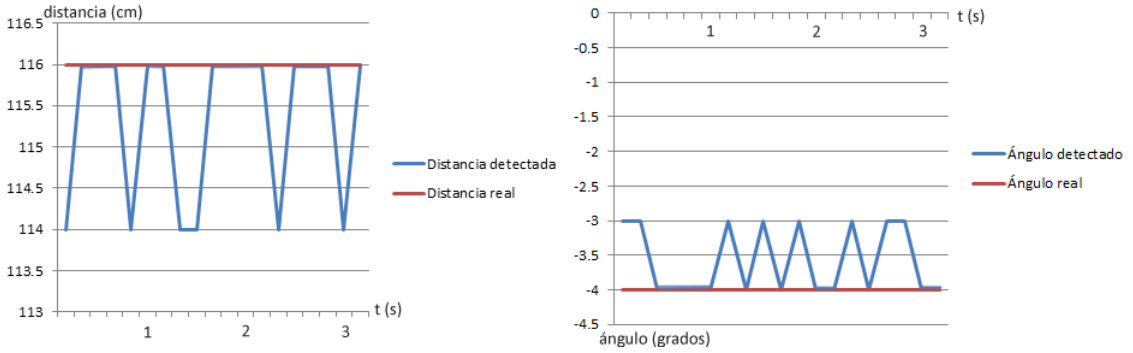

Figura 4.4.5: Modelo vectorial - Prueba 3.

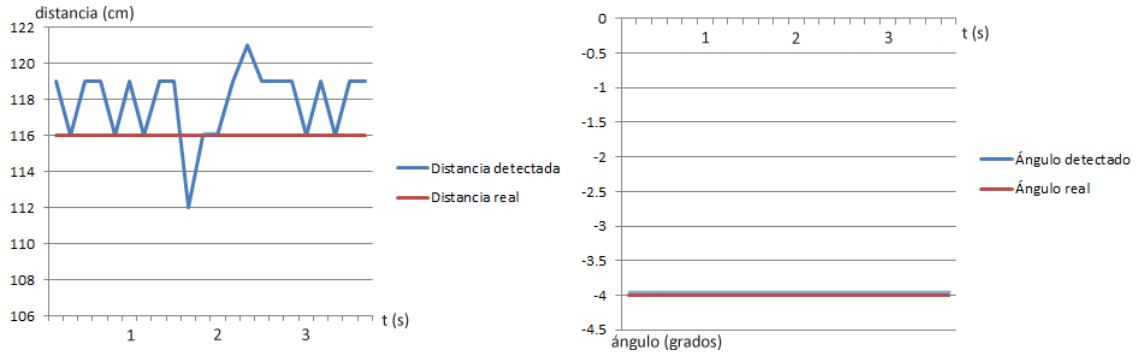

Figura 4.4.6: Modelo vectorial - Prueba 4.

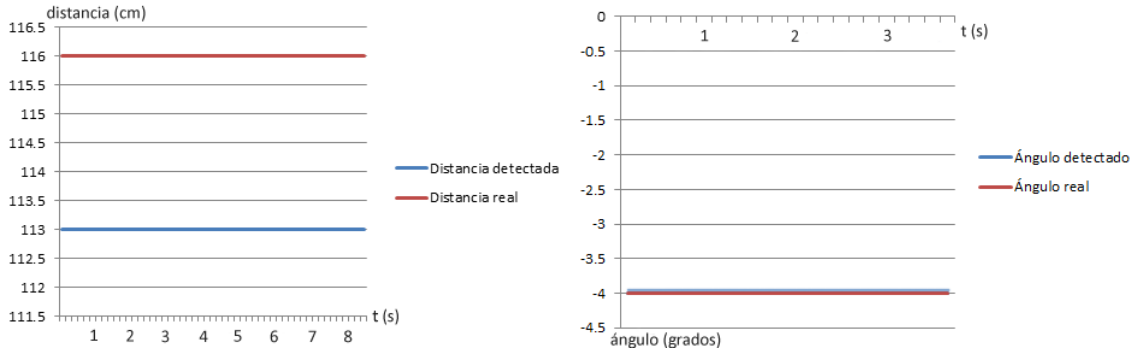

Figura 4.4.7: Modelo vectorial - Prueba 5.

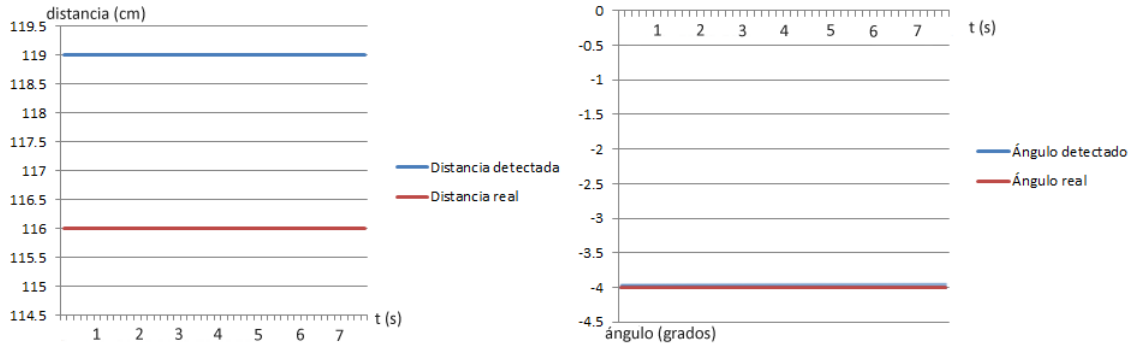

Figura 4.4.8: Modelo vectorial - Prueba 6.

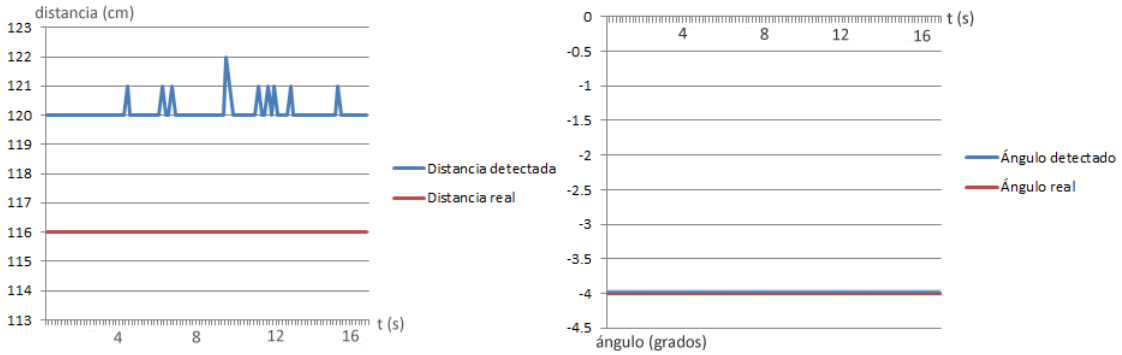

Figura 4.4.9: Modelo vectorial - Prueba 7.

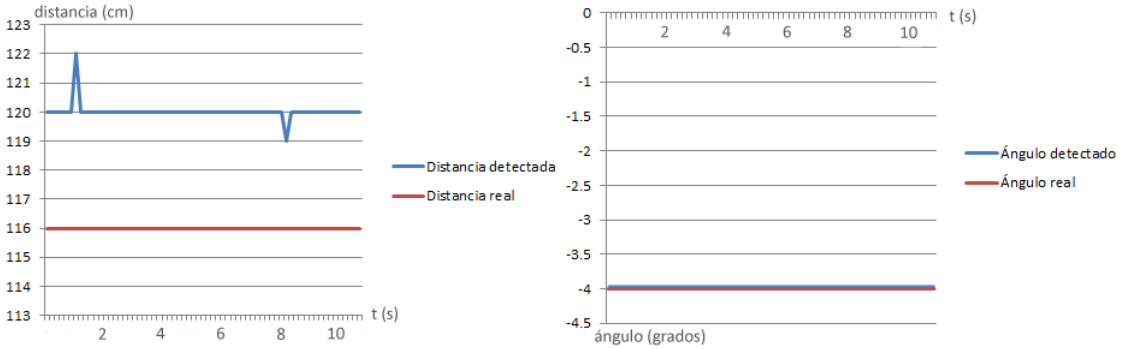

Figura 4.4.10: Modelo vectorial - Prueba 8.

Pelota

- La pelota puede ser detectada hasta 3 metros de distancia del robot.

Landmark

- Los *landmarks* pueden ser detectados hasta 2,5 metros de distancia desde el robot.

Robot

- Otros robots pueden ser detectados hasta 2,5 metros de distancia.

Arco:

- El arco puede ser detectado hasta 4.3 metros de distancia del robot.
- Enfocando el arco de frente, se requiere de una distancia mínima para ser reconocido:
  - Enfocando el arco completo, la distancia mínima al arco debe ser de 1,8 metros.

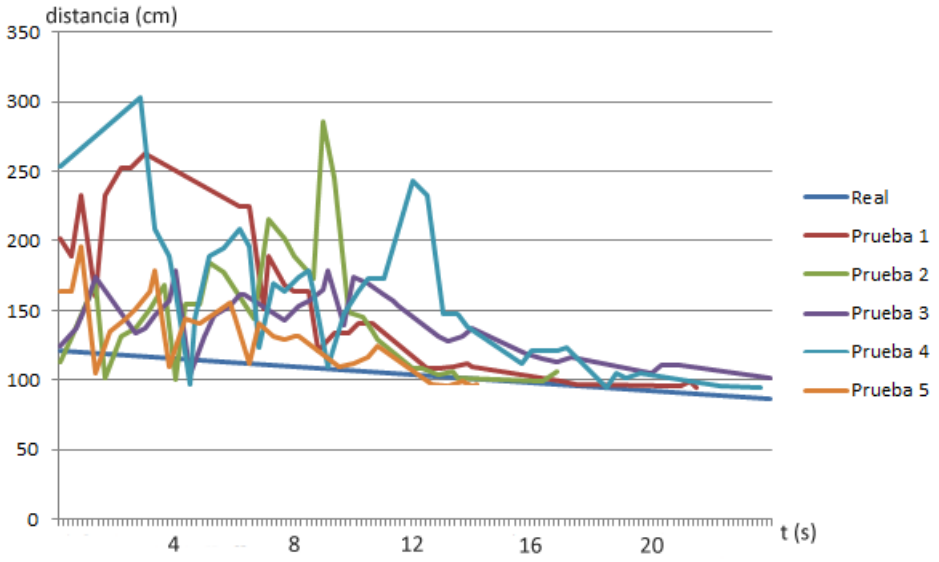

Figura 4.4.11: Distancia en movimiento.

- Enfocando el travesaño y uno de los postes, la distancia mínima debe ser 1,20 metros.

- En el caso en que se enfoque el arco de costado, el mismo puede ser reconocido por sus dos postes, reduciendo la distancia mínima para el reconocimiento.

Poste

- Los postes de los arcos pueden ser reconocidos hasta 2,5 metros.

Entre todos los objetos destaca el Arco. Debido a su gran tamaño puede ser detectado a distancias mayores de 4 metros. Sin embargo, por esta misma razón, y por ser un objeto compuesto posee una distancia mínima para ser reconocido cuando se lo tiene de frente.

Como alternativa, el objeto poste permite mantener la referencia al arco una vez que el agente se encuentra a una distancia menor que la cota mínima para el arco, el cual no presenta cota mínima.

4.4.4. Identificación de un robot desde todos los puntos de vista

Se busca identificar los puntos de vista en que no se detecta un robot, así como los de menor y mayor nivel de semejanza.

El cuadro 4.6 muestra el resultado de 50 iteraciones del módulo de visión, detectando un robot rival orientado de distintas maneras, cuya distancia y ángulo real respecto al agente fueron 116 centímetros y 4 grados, respectivamente.

La orientación con el menor error en lo que respecta a distancia detectada, se corresponde con el robot rival rotado 45 grados; así mismo, el menor error respecto al ángulo detectado surge para la misma orientación.

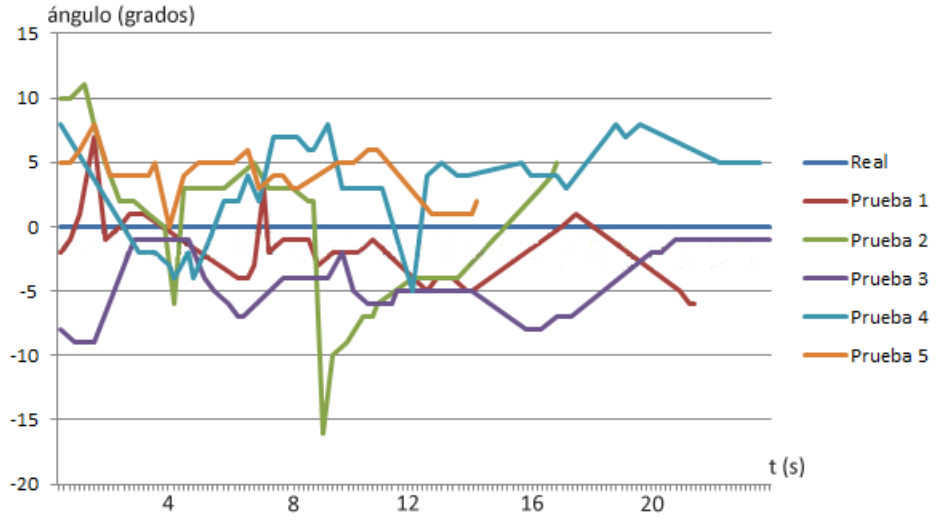

Figura 4.4.12: Ángulo en movimiento.

Orientación	Distancia detectada	Ángulo detectado	Semejanza
0	114 cm	3	2
25	114,6 cm	3,5	2,5
45	115,3 cm	4	1,5
60	117,5 cm	3	0
90	114,4 cm	2	1

Cuadro 4.6: Identificación de un robot en distintas orientaciones.

La orientación con el mayor error de distancia detectada se presenta cuando el robot rival se encuentra de frente al agente. Para el caso del ángulo, el mayor error se encuentra presente en la orientación de 90 grados.

En lo que corresponde a nivel de semejanza, el máximo alcanzado se corresponde con 2,5 para la orientación de 25 grados, que es la segunda mejor orientación en lo que respecta a distancia y ángulo detectado. Cabe destacar que ningún nivel de semejanza obtenido para el robot rival en esta etapa de prueba sobrepasó el 30 % del total. Esto se debe a que los niveles superiores se corresponden con la figura humanoide ideal en lo que respecta a relación entre extremidades, que además cumple todas las restricciones especificadas por *RoboCup Soccer*[10].

Dada la simetría del robot, los casos presentados abarcan la totalidad de la rotación del mismo.

4.4.5. Identificación de menor ángulo de reconocimiento de arco

Se busca identificar el ángulo mínimo en que se detecta un arco, realizando pruebas con el agente a distintas orientaciones del objeto.

[10]En [16] se encuentra especificado el algoritmo utilizado para detección de robots.

- Se detectó que el ángulo entre la línea de fuera y el arco debe ser mayor que 10 grados para que el módulo pueda reconocer al arco.

Esto se debe a que a ángulos menores, los postes del arco se encuentran superpuestos, obteniendo en su lugar el objeto poste.

La figura 4.4.13 muestra sombreadas las zonas de la cancha donde el módulo de visión captura los postes del arco superpuestos, imposibilitando la identificación del arco.

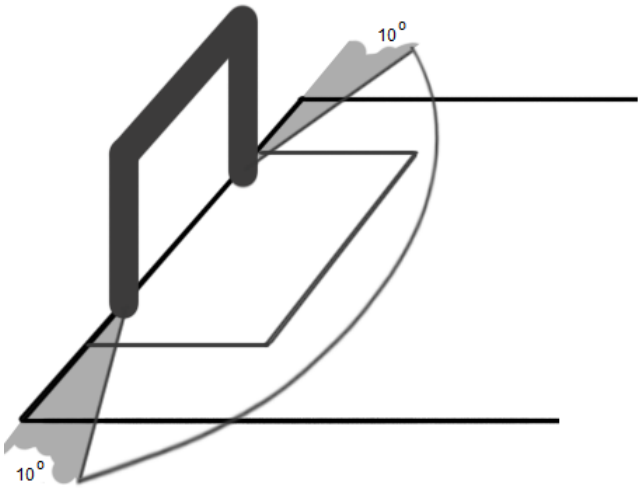

Figura 4.4.13: Zonas de no detección de arco.

4.4.6. Identificación de dos robots superpuestos

Se busca identificar la mínima distancia a la que deben encontrarse dos robots para ser identificados como uno solo.

La distancia entre robots, fue tomada desde el centro de cada uno.

Prueba 1

Reconocimiento de dos robots, el primero orientado de frente al agente, el segundo orientado de costado al agente.

El cuadro 4.7 muestra la relación entre la detección de los dos robots, a medida que se superponen. A primera instancia, es visible una reducción considerable en la identificación de los robots, cuando los mismos se encuentra a 15 centímetros[11], lo cual no sucede para distancias superiores.

Debido a la segmentación realizada por la cámara, los *blobs* que representan al robot de costado, pueden ser asimilados como parte del brazo y pierna del robot que se encuentra de frente.

Según los resultados de la prueba, es posible establecer entonces que, cuando uno de los robots se encuentra de frente y otro de costado, la identificación es constante hasta que los mismos entran en contacto.

[11] Cuando los robots distan 15 centímetros, para la configuración de la Prueba 1, se encuentran continuos.

Distancia entre robots	% de detección de dos robots	Dist. detectada entre robots	Solapamiento de robots
30 cm	50 %	42 cm	0 %
25 cm	47 %	35 cm	0 %
20 cm	50 %	29 cm	0 %
15 cm	10 %	20 cm	0 %[12]

Cuadro 4.7: Identificación de dos robots - Prueba 1

Prueba 2

Reconocimiento de dos robots, ambos de frente al agente.

Distancia entre robots	% de detección de dos robots	Distancia detectada entre robots	Solapamiento de robots
25 cm	80 %	44	0 %
20 cm	68 %	32	0 %
5 cm	20 %	30	75 %
0 cm	11 %	14	100 %

Cuadro 4.8: Identificación de dos robots - Prueba 2

Los resultados de la prueba 2 presentados en el cuadro 4.8 revelan, al igual que en la prueba 1, la reducción en la detección de los dos robots bajo un cierto umbral.

A partir de 20 centímetros, que refiere al escenario donde ambos robots se encuentran juntos, el porcentaje de detección de ambos robots, se reduce mas de 10 %.

En los escenarios donde los robots se encuentran superpuestos, la reducción de la identificación es aun mayor, llegando hasta solo 10 % cuando un robot tapa completamente al otro. Para estos casos, también es posible ver un aumento considerable del error de la distancia detectada entre ambos.

Capítulo 5

Conclusiones

Los módulos de visión tienen el importante cometido de ser el principal, o único, modo de sensado del ambiente externo que poseen los agentes robóticos en *RoboCup Soccer*. Es así que un buen rendimiento tanto en la reconstrucción del mundo, como en el uso de recursos es esencial para componentes como estrategia o localización del robot.

No obstante, la efectividad de un módulo de visión está condicionada no sólo por los algoritmos utilizados, sino también por el hardware disponible.

La solución desarrollada permite el reconocimiento de objetos y comunicación entre agentes, logrando su cometido de capacidad de configuración y facilidad de mantenimiento. Actualmente, el módulo de visión se encuentra integrado con el proyecto Salimoo[18], quien hace uso del mismo para la identificación de objetos en la cancha.

Sin embargo, hubiera sido interesante que el hardware de cámara permitiese acceder a la imagen capturada, abriendo la posibilidad a la implementación de otro tipo de algoritmos para el reconocimiento de objetos, como pueden ser la detección de contornos[8], usado ampliamente en la práctica, que permitiese obtener mejores aproximaciones de la forma de los objetos.

Hoy en día está disponible la versión 2.0 de HaViMo, que presenta un nuevo software de calibración y cuyo resultado de segmentación retorna objetos más complejos. Sería de gran interés extender la solución aquí presentada con el nuevo hardware de cámara.

El actual proyecto es el punto de partida para módulos de visión local para fútbol de robots en el Instituto de Computación y pretende, además de su fin práctico, ser un antecedente para nuevos proyectos de visión, abriendo la posibilidad a futuras investigaciones que mejoren los resultados obtenidos, y sean más eficientes.

5.1. Conclusiones

A continuación se describen las principales conclusiones de las áreas más importantes de la solución.

5.1.1. Identificación de objetos

En lo que respecta a la identificación alcanzada para los objetos de la cancha, los resultados generales son alentadores. Para la mayoría de los objetos, el reconocimiento promedio en las imágenes supera el 70 %. Sin embargo, existe gran dependencia entre la calibración del hardware de cámara y el ruido de fondo, con la identificación conseguida por el módulo.

Así mismo, casos particulares de bajo porcentaje de identificación, como la identificación del arco, plantean la necesidad de realizar su identificación de maneras alternativas, para que el mismo pueda ser utilizado a nivel competitivo, como clasificación bayesiana. Por otro lado, un aumento en la resolución de la cámara utilizada podría brindar información específica de la forma de los objetos, contribuyendo positivamente a la correcta identificación de los mismos.

5.1.2. Modelo vectorial

Los resultados obtenidos sobre el modelo vectorial lo presentan como un método exacto para la obtención de resultados, pero sensible a errores de datos de entrada.

En la mayoría de los escenarios donde se contaba con un ambiente controlado, donde los objetos tenían posición fija, y los datos de orientación y altura de la cámara eran conocidos y constantes, el modelo vectorial fue ideal para el cálculo de la posición polar de los objetos.

Por otro lado, en los ambientes dinámicos, propios del fútbol de robots, el modelo vectorial presenta errores en los resultados obtenidos.

Sin embargo, en ambos casos se detecta una reducción del error cometido cuando los objetos se encuentran a menores distancias, siendo esta una fortaleza del modelo.

Por último, el modelo vectorial cuenta con la desventaja de tener que enfocar la base de los objetos para conseguir una estimación precisa de la posición de los mismos.

5.1.3. Visión multiagente

La visión multiagente desarrollada permite reducir considerablemente las desventajas propias de un módulo de visión local, brindando acceso a visión de otros agentes, obteniendo así varios puntos de vista de la cancha.

No se cuenta con información de módulos de visión que implementen a su vez protocolos de cooperación y distribución de visión entre agentes[1], pudiendo ser ésta una de las mayores ventajas con que cuenta la solución planteada.

La solución consiguió obtener independencia entre tipos de mensaje; sin embargo, el uso del recurso Zig110 reduce el espacio de posibles mensajes que pueden utilizar otros componentes, pudiendo condicionar así el desarrollo de módulos superiores.

En contrapartida, por el hecho de no contar con un componente de localización, la cooperación realizada por el módulo es mínima, reduciéndose a la comunicación de la visión local, sin posibilidad de fusionar la información recibida con la percibida. Si efectivamente existiese dicho componente, se podría considerar retornar un único modelo del mundo, compuesto por la información de visión de todos los agentes aliados del campo.

5.2. Trabajo a futuro y posibles mejoras

5.2.1. Mejora de hardware

Como requerimiento fundamental para mejorar los resultados obtenidos, es necesario contar con un hardware de cámara que permita obtener abstracciones más complejas para definir los objetos identificados en las imágenes. Una mejora significativa en la resolución de la cámara, permitiría obtener resultados más exactos no sólo de la forma de los objetos, sino también de su posición.

[1] Dicha tarea es responsabilidad de componentes superiores como la estrategia, o localización.

5.2.2. Localización

El paso siguiente necesario dentro del lineamiento de la construcción de un equipo de fútbol de robots consiste en el desarrollo de un componente de localización. En [3] se describe el algoritmo de localización por Monte Carlo, el cual es compatible con la solución implementada, y ampliamente utilizado por equipos integrantes de *RoboCup Soccer*.

A partir del componente de localización, es posible extender la Capa de Comunicación de Visión, posibilitando la triangulación de posiciones a partir de fusión de sensores, realizando efectivamente una visión cooperativa. Como requerimiento, se debe poder identificar la correspondencia entre objetos de distintas visiones locales, pero obteniendo a cambio un modelo único del mundo, que engloba la información de visión aportada por cada uno de los agentes del campo, aumentando así el nivel de cooperación que realiza el módulo.

5.2.3. Identificación de arco

Los resultados obtenidos para la identificación del arco en la sección 4.2.1, presentan un porcentaje de identificación del arco menor a 40 %. Esto significa que en más de la mitad de las imágenes donde se captura un arco, el mismo no es identificando, siendo el arco un objeto esencial en el transcurso de un partido.

Se propone mejorar el algoritmo de identificación para el arco, pudiendo utilizar un enfoque similar con el que se identifican los robots [16], que, si bien es más costoso computacionalmente, provee buenos resultados en la práctica. La mejora del hardware propuesta ayudaría además a mejorar la identificación del arco.

Apéndice A

Pseudocódigo de Identificación de objetos

A.1. Identificación de Pelota

```
var current_ball = null;
var aux_area;
var aux_roundness;
para cada blob en blobs {
  si blob es válido y de color color_pelota {
    si blob dista menos de BOTTOM_MARGEN_PROXIMITY al borde inferior
      blob = completar(blob);
    aux_roundness = getRoundness(blob);
    si aux_roundness < MIN_BALL_ROUNDNESS
      continue;
    aux_area = blob.width*blob.height;
    si current_ball == null {
      current_ball = new ball(blob);
    } sino {
      si aux_area > current_ball.area {
        current_ball = new ball(blob);
      }
    }
  }
}
```

A.2. Identificación de Landmark

```
var current_landmark = null;
```

```
para cada region blobs[i] en blobs {
 si blobs[i] es válida y de color central {
  para cada region blobs[j] en blobs {
   si blobs[j] es válida y de color opuesto {
    var umbral = ancho de blobs[i] / 2;
    si blobs[j] está por encima de blobs[i] y a una distancia menor que umbral {
     top_mathced = true;
     top_index_aux = j;
    } sino, si blobs[j] está por debajo de blobs[i] y a una distancia menor que que umbral {
     bot_matched = true;
     bot_index_aux = j;
    }
   }
  }
  si top_matched && bot_matched {
   //Se encontró el landmark completo
   si no había un landmark detectado {
    current_landmark = new landmark(blobs[top_index_aux], blobs[i], blobs[bot_index_aux]);
    desechar blobs usados;
   } sino, si el ancho del blob actual es mayor al de current_landmark {
    current_landmark = new landmark(blobs[top_index_aux], blobs[i], blobs[bot_index_aux]);
    desechar blobs usados;
   }
  } sino, si top_matched || bot_matched {
   si no había un landmark detectado {
    si el ancho de blobs[i] sobrepasa TWO_BLOB_MIN_WIDTH_THRESHOLD {
     //Podemos usar distancia proporcional
     var distancia_proporcional = getDistanciaProporcional(blobs[i]);
     var distancia_vectorial = getDistanciaVectorial();
     si abs(distancia_proporcional - distancia_vectorial) < DISTANCE_METHOD_THRESHOLD {
      current_landmark = new landmark(blobs[top_index_aux], blobs[i], blobs[bot_index_aux]);
      desechar blobs usados;
     }
    }
   }
  }
 }
}
```

A.3. Identificación de Arcos

```
var crossbar = null;
var right_bar = null;
var left_bar = null;
```

```
para cada b1 en blobs {
  si b1 es válido y de color goal_color {
    si b1.width < b1.height {
      //Es un poste
      si right_bar == null {
        right_bar = new bar(b1);
      } sino {
        si left_bar == null {
          left_bar = new bar(b1);
          //cambiar rigth_bar por left_bar según sus .x
        } sino {
          si left_bar.area , rigth_bar,area y b1.area son mayores que MIN_GOAL_POST_AREA {
            //Postes partidos
            //Se juntan los dos postes más cercanos entre left_bar, right_bar y b1
          } sino {
            //Mantener los dos postes de mayor área
          }
        }
      }
    } sino {
      //Es travesaño
      si crossbar == null || crossbar.number_of_pixels < b1.number_of_pixels
      crossbar = new bar(b1);
    }
  }
}
si crossbar == null && (right_bar == null || left_bar == null) {
  //Datos insuficientes para obtener distancia
  return;
}
si crossbar != null {
  si left_bar != null && rigth_bar != null {
    var goal_inner_width = right_bar.min_x - left_bar.max_x;
    si goal_inner_width <= left_bar.width && goal_inner_width <= right_bar.width {
      //Los postes se encuentran muy juntos
      return;
    } sino {
      current_goal = new goal(left_bar, right_bar);
    }
  } sino, si right_bar != null {
    //Verificar que el travesaño esté cerca en x de right_bar y por encima de él
    //Verificar que la distancia en Y no sobrepase crossbar.height / 2
    si right_bar está a la izquierda de crossbar {
      left_bar = right_bar;
      right_bar = null;
      current_goal = new goal(crossbar, left_bar);
```

```
    } sino {
      current_goal = new goal(crossbar, right_bar);
    }
  }
}
```

A.4. Identificación de Postes

```
var current_bar = null;
var bar = null;
var merged = false;
repetir {
  merged = false;
  para cada b en blobs {
    si b es válido y de color goal_color y b.width > MIN_POST_BLOB_WIDTH {
      si bar == null {
        bar = new bar(b);
      } sino {
        si bar dista menos de POST_MIX_BLOBS_THRESHOLD de b {
          bar += new bar(b);
          merged = true;
        }
      }
    }
  }
} mientras (merged);
si bar != null && bar.width > LATERAL_POST_WIDTH {
  current_bar = bar;
}
```

A.5. Identificación de Robots

```
//Se buscan robots completos
para cada b en blobs {
  si b es válido y b es de color robot_color {
    var legs_found = false;
    para cada l en blobs {
      si l es válido y l es de color dress_color {
        si l está debajo de b {
          legs = l;
          si legs cumple las restricciones de diseño respecto al cuerpo b {
            legs_found = true;
            quitar l;
```

```
            break;
          }
        }
      }
    }
    si legs_found {
      qutiar b;
      para cada b2 en blobs y mientras que left_arm == null || right_arm == null {
        si b2 es válido y b2 es de color dress_color {
          //Verificar que b2 se encuentra a una distancia acorde de hombro
          si b2 se encuentra a la izquierda de b
            left_arm = b2
          sino
            right_arm = b2;
        }
      }
      robots.add(new Robot(b, legs, left_arm, right_arm));
    }
  }
}
//Se buscan robots de costado
para cada b en blobs {
  si b es válido y b es de color dress_color {
    para cada b2 en blobs, desde b {
      si b2 es válido y b2 es de color dress_color {
        si abs(b.width - b2.width) < TWO_BLOB_ROBOT_WIDTH_THRESHOLD {
          //b es el brazo
          //Verificar que se cumplan las restricciones de diseño
          robots.add(new Robot(b, b2));
        } sino {
          //b2 es el brazo
          //Verificar que se cumplan las restricciones de diseño
          robots.add(new Robot(b2, b));
        }
      }
    }
  }
}
```

Apéndice B

Pseudocódigo de nivel de Semejanza

Los niveles de semejanza refieren a la similitud de los objetos identificados, frente al modelo interno que se cuenta de dichos objetos. El cálculo del nivel de semejanza toma en cuenta no solo la configuración del conjunto de *blobs* identificado como el objeto, sino también la configuración del escenario. Es así que escenarios incongruentes, como por ejemplo, con más de una posible pelota, presentaran niveles de semejanza menores.

B.1. Nivel de Semejanza para Pelota

```
var semejanza = MAX_LIKENESS_LEVEL;
para cada región de tipo pelota {
  si había identificado una pelota {
    semejanza–;
  }
  si nueva pelota es mejor que la anterior {
    si coeficiente de redondez de nueva pelota es menor que la anterior
      semejanza–;
  }
}
semejanza = semejanza * coeficiente de redondez
```

B.2. Nivel de Semejanza para Landmarks

```
var semejanza = MAX_LIKENESS_LEVEL;
para cada landmark desechado {
  semejanza–;
}
si el landmark detectado no es completo
  semejanza = semejanza / 2;
```

B.3. Nivel de Semejanza para Arcos

```
var semejanza = MAX_LIKENESS_LEVEL;
para cada poste o travesaño no utilizado en el arco detectado {
  semejanza–;
}
si no se encontraron todos los postes del arco {
  semejanza = semejanza / 2;
}
```

B.4. Nivel de Semejanza para Postes

```
si poste.width >= LATERAL_POST_MAX_WIDTH_PX {
  semejanza = MAX_LIKENESS_LEVEL;
} sino {
  semejanza = (poste.width - LATERAL_POST_MIN_WIDTH_PX) * MAX_LIKENESS_LEVEL / LA-
TERAL_POST_MAX_WIDTH_PX;
}
si poste no llega al margen superior de la imagen
  semejanza = semejanza / 2;
```

B.5. Nivel de Semejanza de Robots

```
semejanza = robotProbability(body_blob, leg_blob, left_arm_blob, right_arm_blob);

function robotProbability(blob body, blob legs, blob left_arm, blob right_arm) {
  var prob1 = 0.7;
  var prob2 = 0.7;
  var prob3 = 0.7;
  var prob4 = 0.7;
  var prob5 = 0.7;
  var prob6 = 0.7;
  si body != null {
    var cuad_sigma = body_width/5;
    si legs != null {
      prob1 = normalDistrib(legs_x_center, body_x_center, cuad_sigma);
      prob2 = normalDistrib(legs_width, body_width, cuad_sigma);
    }
    si left_arm != null {
      prob3 = left_arm_width > body_width/3 ? 1;
      prob4 = left_arm_height > body_height/3 ? 1;
    }
```

```
    si right_arm != null {
      prob5 = right_arm_width > body_width/3 ? 1;
      prob6 = right_arm_height > body_height/3 ? 1;
    }
  } sino {
    si legs != null {
      var cuad_sigma = legs_width/5;
      si left_arm != null {
        prob1 = abs(legs_x_center - left_arm_x_center) <= legs_width/2 ? 1;
        prob2 = left_arm->max_y - legs->min_y <= (legs->max_y - legs->min_y) / 2 ? 1;
        prob3 = normalDistrib(left_arm_width, legs_width, cuad_sigma);
      }
      si right_arm != null {
        prob4 = abs(legs_x_center - right_arm_x_center) <= legs_width/2 ? 1;
        prob5 = right_arm->max_y - legs->min_y <= (legs->max_y - legs->min_y) / 2 ? 1;
        prob6 = normalDistrib(right_arm_width, legs_width, cuad_sigma);
      }
    }
  }
  return prob1 * prob2 * prob3 * prob4 * prob5 * prob6 * MAX_LIKENESS_LEVEL / max_prob;
}
```

Apéndice C

Pseudocódigo de Comunicación de Visión

```
var iter = 0;
si hay nueva visión {
  cant_frames_processed++
  si cant_frames_processed == FRAMES_PER_BROADCAST {
    cant_frames_processed = 0;
    enviar broadcast de visión local;
  }
  para cada ally en aliados, si había visión de ally {
    reducir time to live para ally;
    si ttl de ally == max_ttl
      borrar visión recibida de ally;
  }
}
mientras que iter < MAX_MSG_PER_ITER && hay mensajes de visión {
  var msg = getNextVisionMsg();
  resetear ttl de msg.sender;
  si msg.type == MSG_TYPE_RESET {
    borrar visión recibida de ally;
  } sino, si msg.type == MSG_TYPE_BROADCAST {
    almacenar el valor recibido del parámetro msg.param de msg.object para msg.sender;
  }
  iter++;
}
```

Apéndice D

Funciones accesibles del módulo de visión

D.1. Funciones de usuario en Capa de Visión

La capa de Visión presenta funciones que pueden ser accedidas directamente por componentes superiores.

Inicializar la Capa de visión

```
int vision_initialize(int HaViMo_dxl_ID, int baudnum, unsigned int team_color);
```

HaViMo_dxl_ID: identificador dynamixel de HaViMo.

baudnum: *baudnum* con que se inicializará la librería dynamixel o custom_dynamixel.

team_color: color del equipo aliado, debe ser CYAN_COLOR o MAGENTA_COLOR. Considera que el color no escogido es de los rivales.

Retorna: 1 en caso de que la inicialización haya sido exitosa, 0 en caso contrario.

Crea las estructuras que utiliza la capa de Visión e inferiores, inicializando el reconocimiento de objetos, la librería de comunicación con dispositivos dynamixel y la capa de acceso a HaViMo, invocando el procesamiento de *frames*.

Finalizar la Capa de Visión

```
void vision_terminate(void);
```

Finaliza la capa de Visión y el reconocimiento de objetos, liberando la memoria reservada del módulo. Finaliza también la capa de acceso a HaViMo y la librería dynamixel.

Establecer ángulo de pan y tilt en radianes

```
void vision_setPanAndTilt(double pan, double tilt);
```

pan: ángulo de *pan* de la cámara, en radianes.

tilt: ángulo de *tilt* de la cámara, en radianes.

Establece los ángulos de *pan* y *tilt* de la cámara en radianes.

El ángulo de *pan* se corresponde con el ángulo de la cámara contra el plano del suelo, positivo hacia abajo, negativo hacia arriba. El ángulo de *tilt* se corresponde con el ángulo de la cámara contra el frente del robot, antihorario.

Establecer ángulo de pan y tilt en grados

```
void vision_setPanAndTiltInDegrees(int pan, int tilt);
```

pan: ángulo de *pan* de la cámara, en grados.

tilt: ángulo de *tilt* de la cámara, en grados.

Establece los ángulos de *pan* y *tilt* de la cámara en grados.

El ángulo de *pan* se corresponde con el ángulo de la cámara contra el plano del suelo, positivo hacia abajo, negativo hacia arriba. El ángulo de *tilt* se corresponde con el ángulo de la cámara contra el frente del robot, antihorario.

Establecer altura del robot

```
void vision_setRobotHeight(double height);
```

height: altura del robot, en centímetros.

Establece la altura del robot.

En caso de que no se especifique ninguna altura, se utiliza *DEFAULT_ROBOT_HEIGHT*.

Procesar imagen

```
unsigned short int vision_checkVision(unsigned short int force_frame);
```

force_frame: con un valor mayor a 0 ejecuta la función en modo bloqueante, retornando únicamente cuando se haya procesado un nuevo *frame*.

Retorna: 1 si se procesó un nuevo *frame*, 0 en caso contrario.

Verifica si HaViMo procesó un *frame* e invoca nuevamente la captura y procesamiento sobre HaViMo.

En caso de que el procesamiento de un *frame* esté disponible, se realiza el proceso de visión identificando los objetos en la imagen. El resultado puede ser accedido en el arreglo global *objects*, que tiene tamaño *obj_size*.

D.2. Funciones de usuario en Capa de Comunicación

Las funciones accesibles desde la capa de Comunicación involucran la comunicación entre agentes, además de la ejecución del proceso de visión.

Inicializar la Capa de Comunicación

```
void vision_com_initialize(unsigned short int ag_id, unsigned int cant_frames_for_broadcast, unsigned short int min_likeness_level, unsigned int max_parsed_messages, unsigned int time_to_live);
```

ag_id: identificador del agente. Comienza en 0, hasta *CANT_ALLIES* - 1.

cant_frames_for_broadcast: cantidad de *frames* que deben ser procesados por la capa de Visión para que se distribuya la visión local a los demás agentes. Se recomienda el valor *DEFAULT_CANT_FRAMES_FOR_BROADCAST*.

min_likeness_level: nivel de semejanza mínimo para que un objeto sea transmitido. De 0 a *MAX_LIKENESS_LEVEL*. Se recomienda el valor *DEFAULT_LIKENESS_LEVEL_TO_SEND*.

max_parsed_messages: cantidad de mensajes de visión recibidos que se deben analizar como máximo en cada invocación. Se recomienda el valor *DEFAULT_PROCESSED_MSG_PER_ITERATION*.

time_to_live: cantidad de iteraciones que serán válidos los mensajes recibidos de aliados. Se recomienda el valor *DEFAULT_TIME_TO_LIVE*.

Inicializa las estructuras necesarias de la Capa de comunicación.
No se inicializa la Capa de visión.

Finalizar la Capa de Comunicación

```
void vision_com_terminate(void);
```

Finaliza la comunicación de visión y libera la memoria dinámica utilizada por dicha capa.
No se finaliza la Capa de visión.

Analizar mensajes

```
unsigned short int vision_com_parseMessages(unsigned short int force_vision);
```

force_vision: especifica si se debe forzar la obtención del *frame* en la capa de Visión.

Retorna:

NO_VISION_NO_MSG Si no se identificó ningún objeto y no se recibió ningún mensaje.

VISION_NO_MSG Si se identificaron objetos y no se recibieron mensajes.

NO_VISION_AND_MSG Si no se identificó ningún objeto y se recibieron mensajes.

VISION_AND_MSG Si se identificaron objetos y se recibieron mensajes.

Recorre los mensajes recibidos de Visión del componente ZigbeeVisionFilter, almacenando los datos recibidos. Serán analizados hasta *max_parsed_messages*.

Realiza automáticamente la invocación a la Capa de visión para procesar el *frame*, distribuyendo el resultado si se han procesado *cant_frames_for_broadcast frames*.

Actualizar los objetos recibidos de aliados

```
void vision_com_updateObjects(void);
```

Analiza los datos recibidos de los demás aliados, y los presenta a los componentes superiores.

El resultado de la Capa de Visión puede ser accedido en el arreglo global *objects*, de tamaño *obj_size*.

El resultado de la Capa de Comunicación de Visión puede ser accedido en el arreglo global *allied_objects*, de tamaño *CANT_ALLIES*.

D.3. Funciones de usuario en Filtro de Zigbee

El Filtro de Zigbee permite utilizar la tecnología Zigbee de manera transparente, evitando colisiones con mensajes del módulo de Visión. Para ello, los mensajes enviados por el usuario no pueden finalizar con *ZIGBEE_AUTHENTICATION_HEADER*. Las siguientes funciones son para acceso de componentes externos a visión.

Inicialización del Filtro de Zigbee

```
void vision_zigFilter_initalize(int dev_index, int vision_buffer_length, int msg_buffer_length);
```

dev_index: parámetro *dev_index* de libzigbee[6].

vision_buffer_length: largo del *buffer* para mensajes del módulo de Visión.

msg_buffer_length: largo del *buffer* para mensajes externos al módulo de Visión.

Inicializa el Filtro de Zigbee, y la librería libzigbee de BIOLOID.

Finalización del Filtro de Zigbee

```
void vision_zigFilter_terminate(void);
```

Finaliza el Filtro de Zigbee y la librería libzigbee, liberando la memoria reservada para los *buffers*.

Envío de mensaje

```
int vision_zigFilter_sendMsg(int msg);
```

msg: mensaje a enviar.

Retorna: 1 si el mensaje es enviado exitosamente, 0 sino.

Envía un mensaje común a través de Zigbee.

Corroborar si hay un mensaje

```
int vision_zigFilter_hasNextMsgBuffer(void);
```

Retorna: 1 si el *buffer* de mensajes no esta vacío.

Verifica si hay un mensaje común en el *buffer* de mensajes.

Obtener mensaje

```
int vision_zigFilter_getNextMsg(void);
```

Retorna: el siguiente mensaje en el *buffer* de mensajes.

Obtiene el siguiente mensaje común del *buffer* de mensajes y lo desencola.

Verificar si el *buffer* se encuentra lleno

```
int vision_zigFilter_isFullMsgBuffer(void);
```

Retorna 1 si el *buffer* de mensajes normales esta lleno.

Verifica si el *buffer* de mensajes comunes se encuentra lleno.

Nomenclatura

blob	Agrupación de píxeles adyacentes en una imagen, según una propiedad común.
CM-500	Controlador de la plataforma BIOLOID de ROBOTIS.
CM-510	Controlador de la plataforma BIOLOID de ROBOTIS, superior a CM-500. Incluido en kits BIOLOID Premium
Dynamixel	Actuador exclusivo para robots, de ROBOTIS
HaViMo	Dispositivo de hardware, utilizado como cámara para la obtención de imágenes.
pan	Ángulo de orientación de una cámara, respecto al plano del suelo.
pgVisRobUtils	Aplicación de escritorio desarrollada en el marco de este proyecto para depuración del Modulo de Visión.
sockets	Mecanismo para la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados.
stub	Código o componente usado como sustituto de alguna otra funcionalidad, simulando el comportamiento de código existente o sustituyendo temporalmente código aún no desarrollado.
tilt	Ángulo de orientación de una cámara, respecto al frente del robot.
USB2Dinamyxel	Adaptador de conexión serial (RS-232), TTL o RS-485 a USB.
Zig100	Dispositivo de hardware que implementa la tecnología Zigbee. Utilizado en CM-500 y zig2serial
Zig110	Dispositivo de hardware que implementa la tecnología Zigbee. Utilizado en CM-510.
zig2Serial	Adaptador de serial (RS-232) a Zig100.
Zigbee	Tecnología de comunicación inalámbrica, similar a bluetooth, utilizada por ROBOTIS en su linea BIOLOID.

Bibliografía

[1] RoboCup. RoboCup Soccer. `http://www.robocup.org/robocup-soccer/`. Accedida 29-Agosto-2012.

[2] D. Fox S. Thrun, W. Burgard. *Probabilistic Robotics*. MIT Press, 2005. ISBN 0-262-20162-3.

[3] F. Dellaert S. Thrun D. Fox, W. Burgard. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings Of The National Conference On Artificial Intelligence*, pages 343–349. AAAI, 1999.

[4] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, LTD, 1 edition, 2002.

[5] Hamid Mobalegh. Embedded vision module for bioloid - quick start. 2007.

[6] ROBOTIS. e-manual. `http://support.robotis.com/en/`. Accedida 29-Agosto-2012.

[7] RoboSavvy. Foro de robótica. `http://robosavvy.com/forum/`. Accedida 29-Agosto-2012.

[8] Gonzalo Gismero. Estado del arte visión por computador. `http://www.fing.edu.uy/~pgvisrob`. Accedida 29-Agosto-2012.

[9] 2010 RoboCup Soccer Humanoid League. *RoboCup Soccer Humanoid League Rules and Setup for the 2010 competition in Singapore*. RoboCup Soccer Humanoid League, 1 edition, 2010.

[10] Tomás González Domenec Puig José M. Cañas, Eduardo Perdices. Recognition of standard platform robocup goals. *Physycal Agents*, 4(1):11–18, January 2010.

[11] Gerd Mayer, Ulrich Kaufmann, Gerhard Kraetzschmar, and Günther Palm. Neural robot detection in robocup. In *Biomimetic Neural Learning for Intelligent Robots - Intelligent Systems, Cognitive Robotics, and Neuroscience, ser. LNCS 3575*, pages 349–361. Springer, 2005.

[12] Luke Cole, David Austin, and Lance Cole. Visual object recognition using template matching. In *Proceedings of Australian Conference on Robotics and Automation*, 2004.

[13] K Z Mao, K C Tan, and W Ser. Probabilistic neural-network structure determination for pattern classification. *IEEE Transactions on Neural Networks*, 11(4):1009–1016, 2000.

[14] Y. Uny Cao, Alex S. Fukunaga, and A. B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:226–234, 1997.

[15] E. H. Durfee. Distributed problem solving and planning. *Multiagent Systems*, pages 121–164, 1999.

[16] Gonzalo Gismero. Arquitectura del módulo de visión. `http://www.fing.edu.uy/~pgvisrob`. Accedida 29-Agosto-2012.

[17] ROBOTIS. Bioloid, users guide. 2006.

[18] G. Pias M. Baliero. Proyecto de grado salimoo. `http://www.fing.edu.uy/~pgsalimoo/`. Accedida 20-Julio-2012.

[19] Gonzalo Gismero. Manual de usuario del módulo de visión. `http://www.fing.edu.uy/~pgvisrob`. Accedida 29-Agosto-2012.

[20] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. *Design Patterns*. Addison-Wesley, 1995.

[21] Gyovinet.com JAVA ELECTRONICS SOLUTIONS. Gyovinet driver. `http://www.giovynet.com/`. Accedida 29-Agosto-2012.

[22] ROBOTIS. Zigbee e-manual. `http://support.robotis.com/en/product/auxdevice/communication/zigbee_manual.htm`. Accedida 29-Agosto-2012.

[23] Ted Shultz and Luis A. Rodriguez. 3d reconstruction from two 2d images. 2003.

[24] Koen Erik Adriaan van de Sande. A practical setup for voxel coloring using off-the-shelf components. Bachelor Project supervised by Rein van den Boomgaard, Junio 2004.

[25] Hanspeter Pfister y Leonard McMillan Remo Ziegler, Wojciech Matusik. 3d reconstruction using labeled image regions. In H. Hoppe L. Kobbelt, P. Schröder, editor, *Eurographics Symposium on Geometry Processing*, pages 1–12, 2003.

[26] Aldo Laurentini. The visual hull concept for silhouette-based image undestanding. *IEEE Transactions on pattern analysis and machine intelligence*, 16(2):150–162, Febrero 1994.

[27] D A Gustafson and E Matson. Taxonomy of cooperative robotic systems. *Systems Man and Cybernetics*, 2:1141–1146, 2003.

[28] Margrit Betke and Leonid Gurvits. Mobile robot localization using landmarks. *IEEE Transactions on robotics and automation*, 13(2):251–263, Abril 1997.

[29] Steven Nicklin. Object recognition in robotic soccer. Master's thesis, University of Newcastle, Australia, Noviembre 2005.

[30] Aaron Wong Jason Kulk Stephan K. Chalup Robert King Naomi Henderson, Steve P. Nicklin. The 2009 nubots team report, December 2009.

[31] Ilan Shimshoni. On mobile robot localization from landmark bearings. *IEEE Transaction on Robotics and Automation*, 18(6):971–976, December 2002.

[32] I Kallenbach, R Schweiger, G Palm, and O Lohlein. Multi-class object detection in vision systems using a hierarchy of cascaded classifiers. *2006 IEEE Intelligent Vehicles Symposium*, pages 383–387, 2006.

[33] C. E. Aguero J. M. Canas V. Matellán F. Martín. Estimación de objetos con fusión bayesiana en equpios de robots móviles. 2009.

[34] M. Jamzad, B. S. Sadjad, V. S. Mirrokni, M. Kazemi, H. Chitsaz, A. Heydarnoori, M. T. Hajiaghai, and E. Chiniforooshan. A fast vision system for middle size robots in robocup. In *5th International Workshop on RoboCup 2001 (Robot World Cup Soccer Games and Conferences), number 2377 in Lecture Notes in Computer Science*, pages 71–80. Springer, 2002.

[35] G. Fernández C. Stocco N. Tourn. Visión de robots - 2003. `http://www.fing.edu.uy/inco/grupos/mina/pGrado/vision2003/`. Accedida 29-Agosto-2012.

[36] S. Casella S. Margini G. Tejera, F. Benavides. Teórico de ia y robótica. 2009.

[37] ROBOTIS. Dynamixel ax-12, users manual.

[38] ROBOTIS. Bioloid premium kit quick start.

[39] WinAVR. Avr-gcc for windows. `http://winavr.sourceforge.net/`. Accedida 29-Agosto-2012.

[40] Nubots. Newcastle robotics laboratory. `http://robots.newcastle.edu.au/robocup.html`. Accedida 29-Agosto-2012.

[41] CMUnited. Carnegie mellon robot soccer. `http://www.cs.cmu.edu/~robosoccer/main/`. Accedida 29-Agosto-2012.

[42] CSFreiburg. The robotic soccer team of albert-ludwigs-universitat and sick ag. `http://www.cs-freiburg.de/`. Accedida 29-Agosto-2012.

[43] ROBOTIS. Roboplus terminal. `http://support.robotis.com/en/software/roboplus/roboplus_terminal_main.htm`. Accedida 29-Agosto-2012.

[44] Hamid Mobalegh. Hamid's vision module. `http://robosavvy.com/site/index.php?option=com_openwiki&Itemid=&id=havimo`. Accedida 29-Agosto-2012.