Documento de Arquitectura
Proyecto de grado Visión de Robots

Gonzalo Gismero
http://www.fing.edu.uy/~pgvisrob - gegismero@gmail.com

Tutores: Facundo Benavides, Gonzalo Tejera, Serrana Casella
InCo - FIng- UDELAR
Montevideo - Uruguay

6 de septiembre de 2012

Resumen

El presente documento describe la arquitectura y diseño del módulo de Visión desarrollado bajo el marco del proyecto de grado Visión de Robots, bajo las reglas de liga humanoide, *kid Size* de *RoboCup Soccer* 2010. El módulo desarrollado tiene como fin servir a proyectos que requieran el uso de un módulo de Visión para el hardware de cámara HaViMo en el ambiente de fútbol de robots.

Índice

1. Introducción

1.1. Objetivos del módulo de software

El objetivo principal del módulo consiste en brindar a otros componentes de software, los elementos identificables en una cancha de fútbol de robots y sus propiedades; esto es: un módulo de visión.

A raíz de las restricciones de la liga humanoide, *kid Size* de *RoboCup Soccer* 2010 [1], el módulo deberá ejecutarse en la plataforma robótica, sin comunicación con dispositivos externos, a excepción de otros robots. Es por esta razón que el módulo de visión deberá ser desarrollado como un módulo de visión local, embebido en el robot, el cual se comunicará con los demás robots de su equipo para obtener datos recabados tanto localmente como por sus aliados, resultado de un proceso de visión.

1.2. Proceso de Visión

El proceso de visión que define un módulo de visión local, consiste en:

1. Obtención de imagen.
2. Interpolación de color.
3. Segmentación de la imagen.
4. Identificación de objetos.
5. Extracción de propiedades de objetos reconocidos.

Debido al hardware de cámara escogido[2], tanto la interpolación de color, como la segmentación de la imagen son etapas realizadas por hardware, reduciendo la implementación del módulo de visión.

Sin embargo, el objetivo del módulo a construir abarca además la comunicación entre los distintos agentes del campo para mejorar la visión percibida, por lo que las responsabilidades del módulo de visión quedan definidas por:

1. Acceso a resultados de segmentación de hardware de cámara
2. Identificación de objetos.
3. Extracción de propiedades de objetos reconocidos.
4. Enviar información recabada localmente a los demás agentes.
5. Recibir información de la visión de los demás agentes.

Como resultado de este proceso específico de visión se obtiene el conjunto de objetos identificados en la imagen y sus propiedades, así como el conjunto de objetos identificados por los demás integrantes del equipo y sus propiedades.

1.3. Propiedades de interés

Las propiedades de los objetos que son de interés a componentes de software superiores, consisten en:

- Tipo de objeto identificado
- Distancia del agente al objeto identificado.
- Ángulo del frente del agente, al objeto identificado.

El tipo de objeto identificado, es el resultado de la identificación realizada; mientras que la distancia y el ángulo son obtenidos mediante un modelo vectorial[3].

1.4. Comunicación con otros agentes

La comunicación entre agentes se realizará mediante la tecnología Zigbee, la cual permite la transmisión de 16 bits por mensaje. El proveedor de la plataforma robótica brinda una librería para el acceso al hardware, la cual implementa detección de errores de transmisión.

Zigbee cuenta con 4 canales distintos para la transmisión de mensajes en modo *broadcast*, por lo que se asume que no hay colisiones entre los mensajes de equipos distintos.

2. Arquitectura 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.

Implementado bajo la Capa de acceso a HaViMo, abarca:

- Acceso a resultados de segmentación

Proceso de Visión.

Implementado bajo la Capa de Visión, consisten en:

- Identificación de objetos
- Extracción de propiedades de objetos reconocidos

Comunicación con los demás agentes.

Implementado bajo la Capa de Comunicación, abarca:

- Envío de información de visión local a los demás agentes
- Recepción de visión de otros agentes

La figura 1 muestra la composición del módulo de visión desarrollado. A continuación se describen las responsabilidades de cada componente.

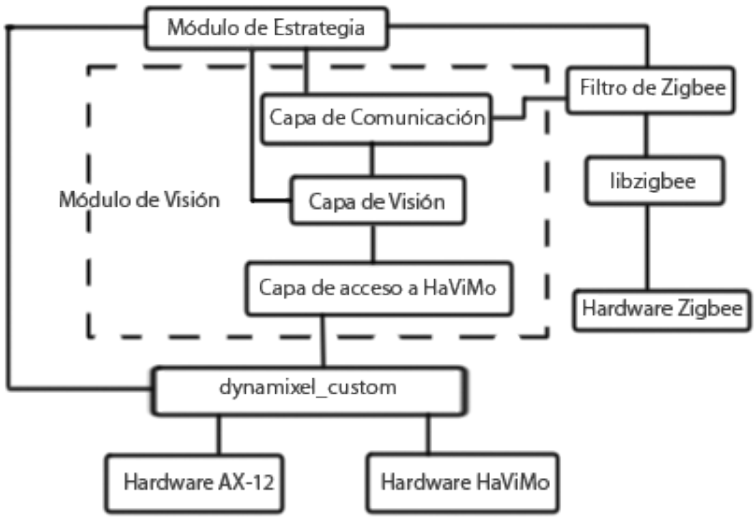

Figura 1: Arquitectura del módulo de visión

2.1. 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[2]. La figura 2 muestra el único componente de la capa.

Figura 2: Componente de la Capa de acceso a HaViMo

Interfaz HaViMo responsable de controlar el acceso a la cámara, solicitar al hardware el procesamiento del siguiente *frame* y realizar el traspaso de los resultados de la segmentación realizada por HaViMo desde la memoria de la cámara, a memoria interna del controlador.

El resultado retornado por el dispositivo de *hardware* HaViMo, no es la imagen en bruto, sino un conjunto de hasta 15 *blobs* [1] que identifican sectores con propiedades similares en dicha imagen, que son el resultado de la segmentación realizada por *hardware*. La figura 3 muestra el resultado de la segmentación para una imagen en particular realizada por HaViMo.

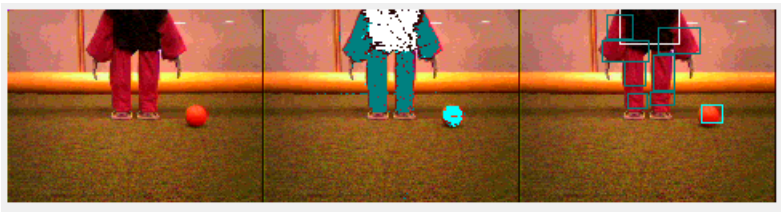

Figura 3: Resultado de HaViMo
A la izquierda, imagen original. Al centro, calibración realizada. A la derecha, conjunto de *blobs*, resultado de segmentación de HaViMo.

Cada *blob* resultante contiene las siguientes características:

region_index índice del *blob* en el conjunto. El *blob* es válido si el índice es mayor a 0.

region_color color de la región. Manteniendo la correspondencia que realiza el proveedor de HaViMo, region_color se encuentra entre los siguientes valores:

- 0 UNKOWN
- 1 BALL

[1] Agrupaciones de píxeles, representadas por un rectángulo.

- 2 FIELD
- 3 MY GOAL
- 4 OPPONENT GOAL
- 5 ROBOT
- 6 CYAN
- 7 MAGENTA

number_of_pixels cantidad de píxeles en el *blob.*

sum_of_x_coord suma de las coordenadas en la abscisa X de la imagen, de todos los píxeles pertenecientes al *blob*.

sum_of_y_coord suma de las coordenadas en la abscisa Y de la imagen, de todos los píxeles pertenecientes al *blob*.

max_x máximo valor de las coordenadas en la abscisa X de la imagen, del *blob* identificado.

min_x mínimo valor de las coordenadas en la abscisa X de la imagen, del *blob* identificado.

max_y máximo valor de las coordenadas en la abscisa Y de la imagen, del *blob* identificado.

min_y mínimo valor de las coordenadas en la abscisa Y de la imagen, del *blob* identificado.

Las coordenadas retornadas se corresponden con coordenadas de la imagen, donde el eje X crece desde el margen izquierdo al derecho, y el eje Y crece del margen superior al inferior.

2.2. Capa de visión

La Capa de visión, encargada de realizar el proceso de visión, contiene los componentes presentes en la figura 4.

Figura 4: Componentes de la Capa de visión

Clasificación encargado de la clasificación de *blobs*, realizada sobre la segmentación de HaViMo, buscando conjuntos particulares de *blobs* para ser agrupados. Invoca directamente a la Interfaz HaViMo

Reconocimiento de objetos implementa la identificación de los objetos sobre el resultado de la clasificación. Extrae además la distancia y ángulo a los objetos identificados a partir de modelos vectoriales[2].

Interfaz de Visión realiza el traspaso de los datos obtenidos mediante el procesamiento de la imagen, a memoria global para ser accedida por cualquier componente externo.

Como resultado de la capa de visión, se obtiene un conjunto de objetos de visión, que representan los objetos identificados en la imagen. Los objetos de visión se conforman por:

type tipo de objeto identificado, entre los siguientes elementos:

- 0 MYSELF
- 1 BALL
- 2 PARTNER
- 3 ENEMY
- 4 RIGHT_LANDMARK
- 5 LEFT_LANDMARK
- 6 MY_GOAL
- 7 OPP_GOAL
- 8 MY_LATERAL_POST
- 9 OPP_LATERAL_POST

[2]Ver sección 3.4

Si bien no se retornan objetos MYSELF, se mantiene por extensibilidad para la implementación de localización.

x ángulo al objeto, desde el frente del robot, en grados, en sentido antihorario. Abarcando de -90 a 90 grados.

y distancia del objeto a la base del robot, en centímetros.

z sin uso, mantenido por extensibilidad.

likeness_level nivel de semejanza del objeto detectado frente al modelo interno del objeto. Describe que tan similar es la configuración de *blobs* con el objeto en cuestión. Valor positivo, siendo configurable su máximo.

2.3. Capa de comunicación

La Capa de comunicación tiene la responsabilidad de interactuar con los demás agentes, a través de Zigbee, para obtener las propiedades de los objetos identificados por otros agentes, y presentarlos a quién invoque al módulo de visión. La figura 5 muestra los componentes de la capa de comunicación.

Figura 5: Componentes de la Capa de comunicación

Adaptador Zigbee implementa el patrón *Adapter*[4], creando mensajes entendibles por componentes de la Capa de comunicación, de más alto nivel, a partir de los paquetes recibidos por Zigbee; y generando y transmitiendo paquetes según los mensajes a enviar.

Buffer de Objetos responsable de almacenar los mensajes de visión local de otros agentes, a medida que son recibidos.

Interfaz de Comunicación de Visión encargado de implementar el protocolo de comunicación definido en la sección 3.7, copiando además a memoria global la visión obtenida de los demás agentes.

Como resultado, se obtiene la visión local de todos los agentes aliados de la cancha, compuesta por objetos del mismo tipo que retorna la Capa de visión.

A excepción de los *landmarks* identificados, todos los objetos de la visión local son comunicados.

2.4. Componentes extra

Fueron desarrollados dos componentes extra externos al módulo de visión. Los mismos son necesarios para acceder al hardware de Zigbee y HaViMo y deben ser utilizados también por componentes externos si se requiere su uso.

libdynamixel_custom basado en la librería de ROBOTIS libdynamixel[5] la cual es utilizada para desarrollar controles de software sobre dispositivos Dynamixel (como motores AX-12). El hardware HaViMo utiliza un protocolo similar al de los motores AX-12 de Dynamixel, pero con más instrucciones; de esta forma, libdynamixel fue modificada para ser utilizada para la comunicación con la cámara, permitiendo el uso de funciones no mapeadas.

Este componente debe ser utilizado, en lugar de libdynamixel, por quien desee acceder a dispositivos Dynamixel y utilice el módulo de visión.

ZigbeeVisionFilter abstrae el uso de Zigbee y proporciona 2 *buffers* distintos, el primero para los mensajes enviados por los componentes de la visión y el segundo para ser utilizado por los demás componentes. A su vez, a los mensajes transmitidos por la visión se les agrega un identificador para poder clasificarlos cuando sean recibidos como de “visión” o de “no visión”; los mensajes ajenos a dicho módulo permanecen inalterados permitiendo de esta manera diferenciar la fuente de los mensajes recibidos.

Este componente debe ser utilizado por todos aquellos componentes que deban utilizar a Zigbee para transmitir mensajes.

2.5. Componentes externos

libzigbee librería provista por ROBOTIS[5] encargada de la comunicación con el dispositivo de *hardware* Zigbee. La misma no fue modificada y permite el traspaso de un entero de 2 bytes, agregando redundancia en los mensajes para controlar errores de transmisión. En caso de que un mensaje llegue modificado, es decir, no verifique el control de error de transmisión, el mismo es desechado.

Es utilizada exclusivamente por el componente ZigbeeVisionFilter para el acceso al *hardware* de Zigbee.

3. Diseño de la solución

Los factores que primaron en el diseño de la solución fueron:

- Reusabilidad
- Bajo consumo de recursos
- Extensibilidad
- Mantenibilidad

Se intentó desarrollar un módulo de visión confiable, pero que su uso no afecte otros componentes esenciales del agente. A continuación se presentan los principales desafíos del módulo de visión y el enfoque con el que fueron resueltos.

3.1. Agrupación de *blobs*

Los casos de mala calibración de colores, trae consigo el error en la identificación de los objetos en la cancha. Aún si la calibración es buena, las condiciones de luz en la cancha pueden no ser uniformes en todas las regiones, por lo que a menos que la calibración haya sido realizada para todos los puntos de vista de los objetos en todos los lugares de la cancha, los *blobs* que representan los objetos en la imagen no estarán completos.

La agrupación de *blobs* es necesaria para contemplar casos de mala calibración y variación de condiciones de luz. De esta manera se consigue eliminar *blobs* de ruido, reforzando aquellos de mayor tamaño detectados, según un umbral ajustable.

Los *blobs* son agrupados según su cercanía, durante el proceso previo a la detección de objetos en la capa de visión, obteniendo como resultado de la agrupación el menor rectángulo que contiene *blobs* cercanos.

3.2. 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.

Los objetos identificables por el módulo de visión son:

Pelota Única en la cancha, de color anaranjado uniforme. Ver figura 6.

Figura 6: Pelota

Landmarks Postes erguidos compuestos de tres secciones. Existen dos *landmarks*, uno a cada lado de la mitad de la cancha. Ver figura 7.

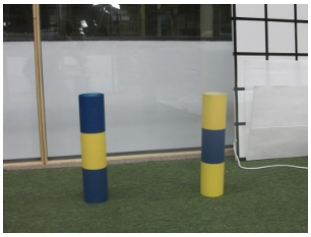

Figura 7: Landmarks

Arcos Uno por equipo, de color azul o amarillo, dependiendo del equipo. Ver figura 8.

Postes Postes de los arcos. Ver figura 8.

Figura 8: Arco y poste

Robots Tanto aliados como rivales. Distinguibles por llevar marcas de color magenta o cyan en piernas y brazos. Ver figura 9.

Figura 9: Robots

El pseudocódigo de la identificación de los distintos objetos puede encontrarse en la sección A.1. La identificación de cada objeto se realiza de la siguiente manera:

3.2.1. 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* del color de 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.

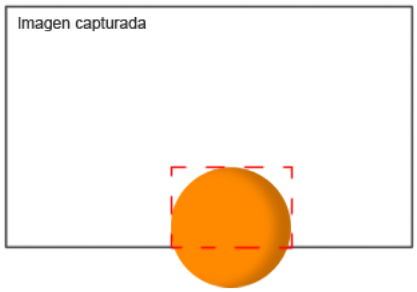

Figura 10: *Blob* detectado de pelota

En el ejemplo de la figura 10, el *blob* delineado de rojo pasaría a ocupar la totalidad de la pelota. Sin embargo, en los casos que solo se vea una pequeña parte de la pelota, el *blob* completado seguirá sin contenerla en su totalidad.

3.2.2. Identificación de Landmarks

Los *landmarks* son identificados buscando las secciones que los componen. De las tres secciones, los colores de la sección superior y la inferior coinciden, pudiendo ser de color azul o amarillo; mientras que el color de la sección del medio es el opuesto. Los *landmarks* se diferencian entre sí, dado que los colores de las secciones están invertidos para su contraparte.

Por esta razón, para cada *blob* que sea de uno de estos colores, se intenta buscar las secciones que lo rodean, siendo el umbral de distancia 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 *landmarks*, 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 *landmarks* 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[3].

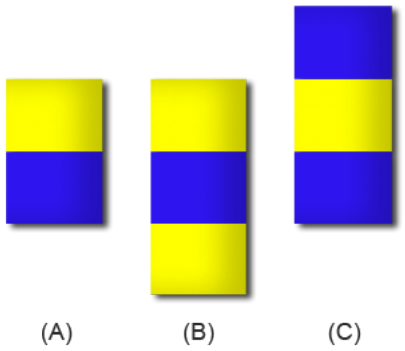

Figura 11: Posibles *landmarks* según secciones detectadas

En la figura 11 se muestra las dos secciones detectadas (A) y los posibles *landmarks* (B) y (C). En este 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* fue identificado.

3.2.3. Identificación de Arcos

Identificados a partir de los dos postes que los componen y el travesaño. Se itera 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 en el área rival o la propia. De esta manera se resuelve identificar también los postes del arco.

3.2.4. Identificación de Postes

Los postes son retornados únicamente cuando no se identifica el arco correspondiente. Su identificación consiste en la agrupación menos rigurosa de *blobs* alineados verticalmente, para los colores del arco.

[3] Ver sección A.3

3.2.5. 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 cyan o magenta en su extremidades para diferenciar a que equipo pertenecen.[1]

En busca de obtener una identificación aceptable para todos los modelos que se puedan diseñar bajo estas restricciones, se utilizó un modelo probabilístico 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.[6]

Es necesario identificar las secciones que componen un robot, como son el cuerpo, los dos brazos y las piernas; éstas últimas son detectadas por HaViMo como un único *blob*. 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 12, $a_x = b_x$.
  2. El ancho de la unión de las piernas debe coincidir con el ancho del cuerpo. En la figura 12, $f_x - e_x = d_x - c_x$.
  3. El ancho de los brazos debe ser mayor que un tercio del ancho de la unión de las piernas. En la figura 12, $h_x - g_x > \frac{f_x - e_x}{3}$ 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 12, $g_y - k_y > \frac{m_y - c_y}{3}$ y $j_y - l_y > \frac{m_y - c_y}{3}$.

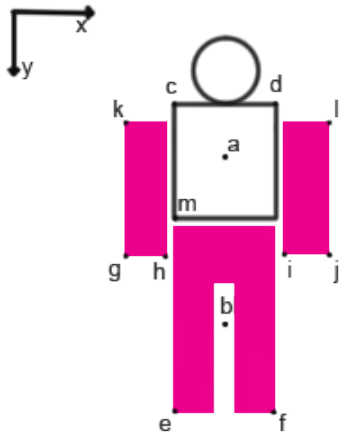

Figura 12: Identificación de Robots, vista frontal

- 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 13, $a_x - b_x < \frac{d_x - c_x}{2}$.
2. La separación entre los brazos y las piernas debe ser menor a un medio de la altura de las piernas. En la figura 13, $f_y - e_y < \frac{c_y - f_y}{2}$.
3. El ancho de los brazos debe coincidir con el ancho de la unión de las piernas. En la figura 13, $g_x - e_x = d_x - c_x$.

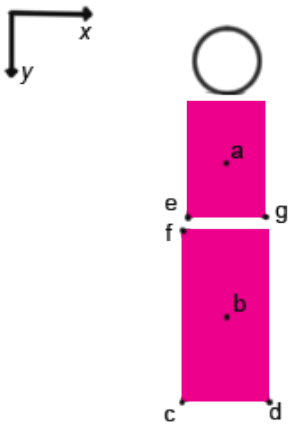

Figura 13: Identificación de Robots, vista lateral

Como entrada previa se realizan controles sobre las secciones pertenecientes al robot. Como resultado, se retornan todos los robots encontrados junto con su probabilidad[4], no descartando de esta manera posibles robots por muy baja que sea su probabilidad.

3.3. 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

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

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.

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*[1] 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.

Para cada objeto, el nivel de semejanza es calculado durante la identificación. A continuación se presenta el pseudocódigo de las funciones de estimación de nivel de semejanza para cada uno de los tipos de objetos retornados.

3.3.1. 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;
```

3.3.2. 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;
```

3.3.3. 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;
}
```

3.3.4. 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
/ LATERAL_POST_MAX_WIDTH_PX;
}
si poste no llega al margen superior de la imagen
 semejanza = semejanza / 2;
```

3.3.5. Nivel de Semejanza para 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;
  }
```

3.4. Extracción de distancia y ángulo

La extracción de propiedades de los objetos identificados es una tarea esencial del proceso de visión.

Para el módulo de visión desarrollado se requiere obtener la distancia a los objetos identificados, así como el ángulo desde el frente del robot al objeto, es decir, la posición de los objetos identificados en coordenadas polares, con centro el robot.

Para ello, se utiliza un modelo vectorial[3], bajo la premisa de que los objetos del campo se encuentran sobre el suelo. Asumiendo lo anterior, interesa obtener entonces la distancia al punto inferior de los objetos en la imagen, reduciendo el problema a encontrar la proyección de dichos puntos del plano de la imagen sobre el plano del piso, con origen la cámara.

En la figura 14 se muestra el modelo vectorial utilizado, donde:

xy campo de juego

O proyección de la cámara perpendicular al plano del suelo.

e altura de la cámara.

ij plano de la imagen capturada

q distancia al punto Q

Θ ángulo desde el frente del robot al punto Q

Q punto del objeto de interés para el que se obtendrá la distancia (q) y el ángulo al robot (Θ)

T punto Q en el plano de la imagen

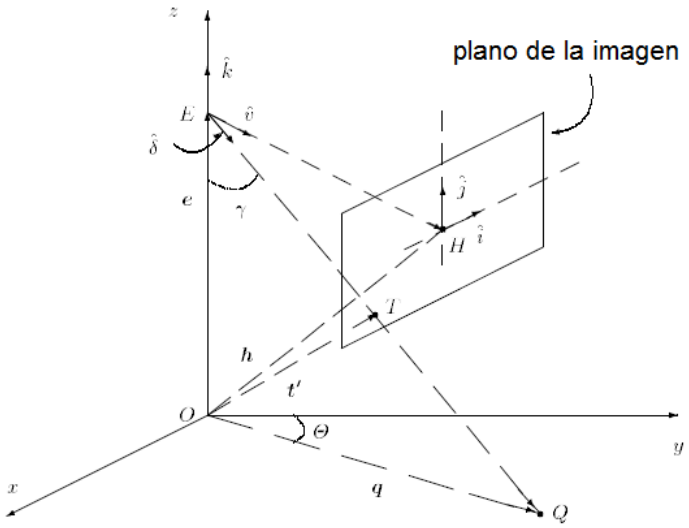

Figura 14: Modelo vectorial para el cálculo de distancia y ángulo a objetos

3.5. Acceso a resultados

Para que la información procesada por el módulo de visión pueda ser accedida desde cualquier componente externo, la misma se retornan en memoria global del controlador, en lugar de ser el resultado de una función. De esta manera, se evitan pasajes de información innecesarios entre componentes superiores. Este enfoque es utilizado con los resultados de la Capa de visión así como con los de la Capa de comunicación.

3.6. Identificación de mensajes de visión

En busca de distinguir los mensajes enviados por el módulo de visión frente a mensajes enviados por otros componentes, fue necesario agregar a los mensajes de visión bits de autenticación. De esta manera, el componente ZigbeeVisionFilter, puede diferenciar los mensajes recibidos.

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

No obstante, como desventaja, los componentes externos que deban transmitir mensajes no podrán contener el mismo valor que el configurado como bits de autenticación de visión en sus últimos bits, reduciendo el espacio posible de mensajes que pueden utilizar.

3.7. Protocolo de comunicación de Capa de comunicación

3.7.1. Ofrecer el modelo del mundo a los demás agentes

La Capa de comunicación tiene la responsabilidad de transmitir los datos obtenidos por la visión local, a los demás agentes del equipo. Dicha tarea puede ser resuelta mediante dos enfoques distintos:

- Distribuir a todos los agentes la visión local cada vez que se procesa una nueva imagen.
- Quien requiera pueda solicitar que le sean transmitidos los datos de una visión externa.

Debido a que el canal de comunicación Zigbee no es punto a punto[5], todos los agentes del equipo reciben todos los mensajes enviados por un agente. Un enfoque como el segundo, desaprovecharía mensajes que reciba un agente pero no fuesen para él. De esta manera se implementó el primer enfoque, reduciendo la tasa con que se distribuye la visión local.

La distribución de la visión local se realiza entonces siguiendo un enfoque *result sharing*[7], 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.

En la sección A.2 se encuentra el pseudocódigo ejecutado al invocar la Capa de comunicación.

3.7.2. Tipos de mensajes

Para la transmisión de mensajes 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.

[5]El dispositivo de hardware Zig110 utilizado permite comunicación punto a punto como *broadcast*. Sin embargo, debido a que un equipo se conforma con más de 2 agentes, es necesaria la comunicación en modo *broadcast*.

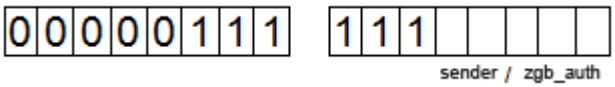

Figura 15: Mensaje de tipo RESET

La figura 15 muestra la estructura de un mensaje de tipo RESET. Los mensajes RESET son enviados al inicio de la difusión de información.

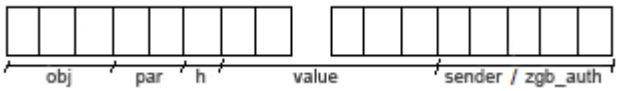

Figura 16: Mensaje de tipo BROADCAST

La figura 16 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:

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.

4. Experimentos y pruebas

4.1. Plan de pruebas preliminares

El plan de pruebas preliminares consiste en el estudio de los resultados del módulo, para distintos escenarios en la cancha.

Como escenarios, se cuenta con:

1. Reconocimiento de objetos estáticos, desde una posición estática.
2. Reconocimiento de objetos en movimiento, mientras el agente se encuentra estático.
3. Reconocimiento de objetos estáticos, mientras el agente se mueve.

Las pruebas del sistema deberán ser realizadas, en los escenarios 1 y 3, con los objetos:

- Pelota
- *Landmark*
- Portería
- Robots
- Postes de arcos

Mientras que para el escenario 2, se realizará con los objetos:

- Pelota
- Robots

4.2. Resultados esperados

Dada la naturaleza del problema, todas aquellas pruebes que contengan objetos en movimiento presentarán ruido en sus lecturas. Es más, los resultados se verán afectados directamente por la calibración previa realizada de la cámara.

Dada la definición del modelo vectorial, el algoritmo es robusto en cuanto al ángulo retornado a los objetos identificados. Como la mayor parte del error en los datos de entrada se centra en el ángulo de *pan* de la cámara[6], se espera que los ángulos obtenidos a los objetos, no difieran en gran medida de la realidad.

Para los casos donde el robot se encuentra en movimiento, el error del módulo será específico para la implementación de la caminata, donde tanto los ángulos de *pan* y *tilt* son variables así como la altura de la cámara.

Por último, cabe mencionar que debido al bajo *frame rate* de la cámara HaViMo 1.5[7] sumado al tiempo de ejecución del módulo de visión, los datos obtenidos diferirán

[6] Debido al robot caminando hacia adelante.

[7] HaViMo 1.5 puede procesar imágenes a 8 *frames* por segundo.

en tiempo de la realidad. Esto es mas notable en los escenarios donde tanto el agente, como los objetos a reconocer se encuentran en movimiento

Se espera obtener una buena aproximación de la posición de los objetos al robot, a medida que se acercan, y un alto porcentaje de identificación de objetos en los escenarios.

A. Anexos

A.1. Pseudocódigo de identificación de objetos

A.1.1. 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.1.2. 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.1.3. 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.1.4. 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.1.5. 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));
            }
          }
        }
      }
    }
```

A.2. 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++;
}
```

A.3. Modelo de proporciones

La distancia a objetos puede ser obtenida considerando que la relación entre la distancia al objeto, y sus dimensiones es constante. Considerando el ancho de un objeto:

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

Donde:
$distancia(cm)$ es la distancia de la cámara al objeto en centímetros.
$ancho(cm)$ es el ancho del objeto en centímetros.
$distancia(pixeles)$ es la distancia de la cámara al plano de la imagen, en píxeles.
$ancho(pixeles)$ es el ancho del objeto en píxeles, percibido en la imagen.

Si bien este método no requiere de conocer previamente la altura del robot o la orientación de la cámara, la distancia obtenida es al centro de la cámara y no al robot. Además, es necesario un hardware de cámara con una resolución superior a la de HaViMo para obtener resultados aceptables.

Referencias

[1] 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.

[2] Hamid Mobalegh PhD. student of Freie Universität Berlin. Embedded vision module for bioloid - quick start.

[3] V. S. Mirrokni M. Kazemi H. Chitsaz A. Heydarnoori M. T. Hajiaghai E. Chiniforooshan M. Jamzad Sadjad, B. S. Sadjad. A fast vision system for middle size robots in robocup.

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

[5] ROBOTIS. Roboits e-manual. http://support.robotis.com/en/.

[6] Dieter Fox, Sebastian Thrun, and Wolfram Burgard. *Probabilistic Robotics*. 1999.

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