Estado del arte
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 el estado del arte referente a la visión por computador y colaboración, en el contexto de la liga humanoid kid-size de RoboCup, para el proyecto de grado de Visión de Robots, del Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República. En el mismo se realiza una introducción básica al proceso de visión por computador y se describen técnicas de reconstrucción y reconocimiento de objetos. Posteriormente, se mencionan los algoritmos de localización más utilizados en el ámbito de fútbol de robots como son la localización por Monte Carlo, o la localización a partir de landmarks que se abstrae del contexto de fútbol de robots; realizando luego una introducción a cooperación en sistemas multiagentes. Finalmente se describe la plataforma robótica con que se cuenta para el desarrollo del proyecto de grado.

Índice

1. Introducción **1**

2. Visión por Computador **2**
2.1. Tareas de la Visión por Computador 2
2.1.1. Reconocimiento 2
2.1.2. Análisis de movimiento 2
2.1.3. Reconstrucción de escenas 3
2.1.4. Restauración de imágenes 3
2.2. Aplicaciones 3

3. Reconocimiento de Objetos **5**
3.1. Introducción 5
3.2. Segmentación 5
3.2.1. Algoritmos de detección de contorno 5
3.2.2. Agrupación por colores similares (*blobs*) 6
3.3. Clasificación de objetos 7
3.3.1. Clasificadores bayesianos 7
3.3.2. Template matching 8
3.3.3. Redes neuronales probabilísticas 8
3.4. Reconocimiento de objetos en Fútbol de Robots 8
3.4.1. Confidence system 9
3.4.2. Reconocimiento de la Pelota 9
3.4.3. Reconocimiento de la Portería 10

4. Técnicas de Reconstrucción **11**
4.1. Introducción 11
4.2. Shape from silohuette techniques 11
4.3. Voxel Coloring 12
4.4. Direct Linear Transformation (DLT) Method 12
4.4.1. Pinhole camera model 12
4.4.2. Stereo Image 12
4.5. Vector based calculation 13

5. Localización **16**
5.1. Localización mediante Landmarks 16
5.1.1. Triangulación 16
5.1.2. Estimación de posición lineal 17
5.2. Monte Carlo Localization 19
5.2.1. Markov Localization 20
5.2.2. Implementación de Monte Carlo Localization 20

6. Agentes Inteligentes o Racionales **22**

7. Definiendo el ambiente – Fútbol de Robots **23**
7.1. Propiedades de los ambientes 23
7.2. Clasificación de un partido de Fútbol de Robots de Robocup . . 24

8. Robótica Cooperativa **25**
8.1. Taxonomía de los sistemas robóticos cooperativos 25
8.2. Interacciones Multiagente 26
8.2.1. Relaciones de dependencia en sistemas multiagente 27
8.2.2. Logrando acuerdos 27
8.2.3. Cooperative Distributed Problem Solving (CDPS) 31
8.2.4. Manejo de inconsistencias 32

9. BIOLOID - Premium kit **33**

10. Hamid's Vision Module (HaViMo) **36**
10.1. Funcionamiento del dispositivo 37
10.2. Calibración en CM-510 37

Índice de figuras

1. Reconstrucción 3D a partir del dispositivo Kinect 3
2. Procesamiento en líneas paralelas de una imagen de un arco . . . 5
3. Ejemplo de histograma de orientación 6
4. Segmentación por hardware en *blobs* de HaViMo 1.5 6
5. Sistemas de coordenadas en DLT 13
6. Puntos evanescentes o puntos de fuga 14
7. Modelo para Vector based calculation 15
8. Triangulación 16
9. Estimación de posición lineal 18
10. Modelos de BIOLOID 34

1. Introducción

El presente documento corresponde al marco referencial del proyecto de grado, donde se expone los tópicos principales y secundarios referentes al objetivo del mismo. Dichos objetivos abarcan la creación de un módulo de software a ser utilizado por un equipo de Fútbol de Robots, respetando las normas establecidas para ROBOCUP[1] sobre la plataforma BIOLOID de ROBOTIS[2]. El módulo de software resultado del proyecto podrá, a partir de los datos recibidos de un módulo de hardware externo encargado de la obtención de imágenes ubicado en el robot[3], identificar los distintos objetos involucrados en un partido de Fútbol de Robots humanoide de la liga *kid-size*, así como estimar sus posiciones, retornando además un grado de certeza independiente para cada uno. Para ello cooperará con los distintos robots de su mismo equipo obteniendo una mejor estimación de las propiedades de los objetos reconocidos.

El documento se distribuye de la siguiente manera: En la sección 2 se realiza una introducción al concepto de Visión por Computadora, donde se describen las tareas y aplicaciones pertinentes a dicha área de la computación. En la sección 3 se describen técnicas utilizadas en el reconocimiento de objetos, orientado a un partido de Fútbol de Robots, haciendo énfasis en el reconocimiento de la pelota y los arcos. La sección 4 describe técnicas de reconstrucción de objetos como son *Shape from Silohuette*, *Voxel Coloring*, DLT y *Vector based calculation*. La sección 5 contiene información sobre técnicas de localización utilizando landmarks, presentando dos enfoques distintos, Triangulación y Estimación de la posición lineal; así como la localización por Monte Carlo, utilizada ampliamente por varios equipos de Fútbol de Robots a nivel competitivo. La sección 6 es una introducción del concepto de agente utilizado en las secciones siguientes. En la sección 7 se define el entorno de un partido de Fútbol de Robots. En la sección 8 se hace una cobertura de los temas relacionados a la cooperación, mostrando los posibles escenarios en una interacción multiagente, y formas de lograr acuerdos entre dichos agentes. Finalmente, en las secciones 9 y 10 se muestra el hardware a utilizar en el desarrollo del proyecto como son las plataformas BIOLOID de ROBOTIS y el módulo de visión HaViMo.

2. Visión por Computador

Visión por Computador (*computer vision*) 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, para poder resolver una tarea específica. Tiene como principales objetivos:

- Detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes.
- Seguimiento de ciertas porciones de una imagen o un objeto a lo largo de una secuencia, conocido como *tracking*.
- Reconstrucción de una escena en un modelo interno (por ejemplo 3 dimensiones), comúnmente utilizado por robots para la navegación.

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. Estas tareas son resueltas diariamente por el ser humano de forma satisfactoria, no siendo así en los sistemas computacionales en los casos más generales, es decir, objetos arbitrarios, en posiciones arbitrarias, con condiciones ambientales arbitrarias. Los métodos existentes han sido desarrollados para objetos específicos, como formas geométricos simples, rostros humanos, escritura, y para condiciones específicas como: iluminación definida, en una posición predeterminada, o para un fondo específico[4][5].

2.1. Tareas de la Visión por Computador

2.1.1. Reconocimiento

Es la tarea de determinar si una imagen contiene o no algún objeto específico, característica o actividad. Variaciones del problema de reconocimiento incluyen:

Reconocimiento de objetos donde se busca reconocer un objeto o clase de objeto preestablecido o aprendido.

Identificación identificación de una huella digital, un rostro, etc.

Detección se busca una condición especial en la imagen.

2.1.2. Análisis de movimiento

Varias tareas están relacionadas con la estimación de velocidades y posiciones a partir de una secuencia de imágenes, como ejemplos de esta tarea se encuentran:

Ergomotion identificación del movimiento de la cámara (traslación y rotación) a partir de una secuencia de imágenes.

Tracking realizar el seguimiento de un conjunto de puntos u objetos en una secuencia de imágenes.

Optical Flow determinar para cada punto de la imagen, el movimiento relativo que tienen con el plano de la imagen.

2.1.3. Reconstrucción de escenas

Dada una o más imágenes de una misma escena, la reconstrucción de escenas apunta a realizar un modelo en tres dimensiones de la escena, que varía desde un simple conjunto de puntos en 3D a superficies en 3D. En la figura 1 se encuentra la imagen obtenida por el dispositivo Kinect[6] y la reconstrucción 3D realizada por el mismo.

Figura 1: Reconstrucción 3D a partir del dispositivo Kinect

2.1.4. Restauración de imágenes

Busca quitar impurezas en las imágenes, como ruido del sensor, o desenfoques causados por movimiento.

2.2. Aplicaciones

Típicas aplicaciones de Visión por Computador son sistemas de vigilancia que incluyen procesamiento de imagen, así como efectos visuales en la industria cinematográfica, por ejemplo realizando *video tracking*[1] .

Uno de los campos que hace uso de la Visión por Computador es la medicina, donde se utiliza el procesamiento de imágenes para realizar diagnósticos en pacientes. Estas imágenes resultado de la Visión por Computador, corresponden a imágenes microscópicas, rayos X, ultrasonidos y tomografías, a las cuales es posible extraer características de interés a partir de procesamiento automático como detección de tumores, arteriosclerosis, tamaño de órganos, etc.

Como segunda área de aplicación se encuentra la industria, donde se extrae información para ayudar en el proceso de manufacturación. Por ejemplo, existen controles de calidad donde se automatiza las inspecciones para encontrar defectos de fabricación. También es utilizada para detectar la posición y orientación de objetos, que ayuda a que un brazo robótico pueda cumplir tareas como asir objetos.

El rubro militar es una de las áreas donde más uso encuentra la Visión por Computador. La detección de enemigos o vehículos y el direccionamiento de

[1] Proceso de localizar objetos en movimiento utilizando una cámara de video (`http://en.wikipedia.org/wiki/Video_tracking`)

misiles por imagen son algunas de las aplicaciones que tiene. Los sistemas más avanzados de direccionamiento de misiles envían el misil a un área en lugar de un objetivo específico, luego basándose en imágenes capturadas, se utiliza el procesamiento de imágenes para elegir el mejor objetivo. Existe además el concepto de "*battle awareness*", que implica obtener la mayor cantidad de información sobre el campo de batalla para así tomar la mejor decisión estratégica, lo que se logra a partir de la obtención de características específicas de imágenes, pudiendo procesar previamente la información para reducir la complejidad.

Sin embargo, el campo más conocido en el que se utiliza la Visión por Computador es el sistema de navegación de agentes autónomos, incluyendo vehículos submarinos, terrestres y aéreos. El nivel de autonomía de estos agentes varía entre totalmente autónomos (vehículos no tripulados) a vehículos donde el sistema de visión ayuda al piloto en distintas situaciones, como por ejemplo, autos que avisan de obstáculos y sistemas para aterrizajes autónomos.

Los vehículos no tripulados utilizan el sistema de visión para poder orientarse, producir mapas del ambiente y detectar obstáculos. También puede ser utilizado para detectar ciertas características del ambiente, por ejemplo un bosque en llamas. Los ejemplos más conocidos son los exploradores Spirit y Oportunity de la misión de exploración a Marte de la NASA, y su contraparte MAX-C enviado por la European Space Agency[7].

3. Reconocimiento de Objetos

3.1. Introducción

La etapa de reconocimiento de objetos en el proceso de visión consiste en, a partir de una imagen obtenida previamente, reconocer los distintos objetos que aparecen en la escena capturada, clasificándolos en diferentes categorías predefinidas (auto, persona, etc).

Antes de poder iniciar el proceso de reconocimiento es necesario preprocesar la imagen, de manera de obtener una abstracción de la misma que sea más sencilla de procesar. Este preprocesamiento consiste en la identificación de regiones de interés en la imagen, llamado segmentación.

Una vez identificadas las regiones de interés, se procede a reconocer las mismas de entre un conjunto acotado de objetos, mediante algoritmos de clasificación como Clasificadores bayesianos, Template matching o Redes neuronales probabilísticas que se presentan a continuación.

3.2. Segmentación

Existen diferentes formas de realizar la segmentación de una imagen dependiendo de cuál sea la entrada que se utilice para reconocer los objetos. El preprocesamiento puede ser realizado por software como en el caso de los algoritmos de detección de contorno, o por hardware como en el caso de la agrupación en *blobs*.

3.2.1. Algoritmos de detección de contorno

Consisten en el procesamiento ordenado de la imagen, recorriendo la misma en líneas paralelas, identificando variaciones en el color.[8]

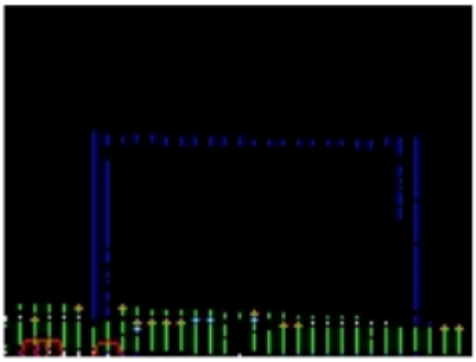

Figura 2: Procesamiento en líneas paralelas de una imagen de un arco

Una vez identificados las regiones de interés, se utiliza un histograma de orientación para describir la forma de la imagen.

La región de interés es subdividida en ventanas, donde se calcula el gradiente en las direcciones X e Y de la imagen en escala de grises, utilizando un operador Sobel[2]. De esta forma se logra representar el contorno de un objeto.[9]

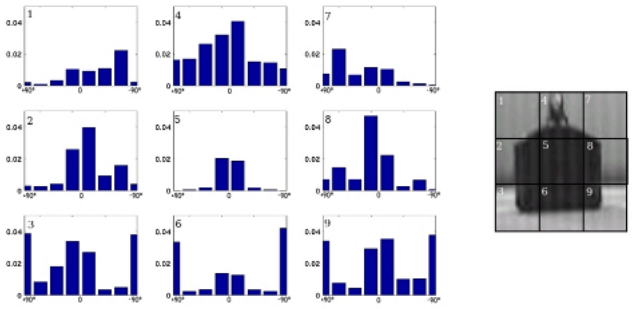

Figura 3: Ejemplo de histograma de orientación

3.2.2. Agrupación por colores similares (*blobs*)

Una alternativa a los algoritmos de detección de contorno se presenta en la forma de *blobs*, donde las objetos de interés de la imagen son descritos a partir de formas mas sencillas, como rectángulos.

Los píxeles de la imagen suelen ser agrupados en *blobs* según su similitud en color, pero también pueden utilizarse otras propiedades complementarias, como la distancia entre los mismos.

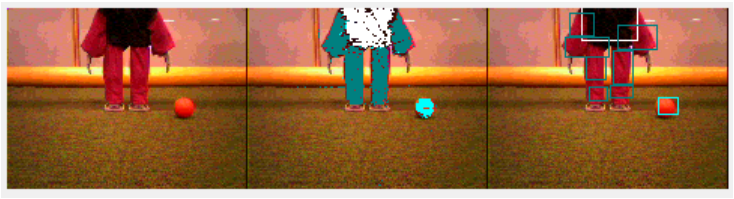

Figura 4: Segmentación por hardware en *blobs* de HaViMo 1.5

En la figura 4 se muestra la segmentación realizada por hardware de la imagen de un robot y una pelota, utilizando la cámara HaViMo. La primer imagen muestra el *frame* capturado por HaViMo, la segunda imagen muestra la calibración realizada, y la tercera el resultado de la segmentación. La definición de las propiedades que caracterizan cada tipo de *blob* se realiza mediante una

[2]Operador diferencial discreto que calcula una apoximación al gradiente de la función de intensidad de una imagen (`http://es.wikipedia.org/wiki/Operador_Sobel`)

tabla de colores conocida como *Lookup table* en la que se le asigna a un tipo de *blob* un conjunto de colores. En la imagen se pueden apreciar tres tipos distintos de *blobs*:

- el tipo de *blob* CUERPO DEL ROBOT calibrado con el color negro y resaltado con blanco.
- el tipo de *blob* EXTREMIDADES DEL ROBOT calibrado con el color magenta y resaltado con el color celeste oscuro.
- el tipo de *blob* PELOTA calibrado con el color anaranjado y resaltado con celeste claro.

3.3. Clasificación de objetos

3.3.1. Clasificadores bayesianos

Los clasificadores bayesianos utilizan la probabilidad de que la clasificación de un elemento sea o no exitosa para distinguir frente a qué elemento se encuentra. Son los clasificadores más utilizados debido a sus buenos resultados, pese a un alto costo computacional.

En su forma más básica, los clasificadores bayesianos se basan en el uso del principio de Bayes en la siguiente probabilidad:

$$p(x|z) \tag{1}$$

Donde z son las características sensadas y x es el estado del ambiente, es decir, cuál es la probabilidad de encontrarse en un estado x dado que los sensores retornan la salida z. Al aplicar bayes se obtiene que:

$$p(x|z) = \frac{p(z|x) * p(x)}{p(z)} = \frac{p(z|x) * p(x)}{\sum_{x'} p(z|x') * p(x')} \tag{2}$$

Donde x' son estados distintos de x, $p(x)$ es conocida como la *"prior probability distribution"* y resume los datos observados de X (estado del ambiente) previo a la incorporación de z. La probabilidad $p(x|z)$ es llamada *"posterior probability distribution"* sobre X. Por lo tanto, si estamos interesados en inferir el valor de x, Bayes nos permite realizarlo utilizando la probabilidad inversa, que especifica la probabilidad de haber sensado z, asumiendo que x era el resultado esperado.[10]

Es de particular interés notar que $p(z)$ no depende de x, por lo que suele utilizarse el factor $p(z)^{-1}$ como una constante normalizadora η.

De esta forma, si recorremos todas las clases de objetos w_i calculando $p(w_i|z)$, un objeto es clasificado del tipo w_i si $p(x|w_i) > p(x|w_j)$ para todo j diferente de i.

3.3.2. Template matching

El método de *Template matching* se basa en la comparación de uno o varios patrones de un objeto en cuestión, con el objeto que se intenta clasificar. Cada patrón puede presentar una variante del mismo objeto lo que lleva a que, para intentar corresponder un patrón con un objeto, deban realizarse múltiples pruebas. El mayor desafío presente en *template matching* se corresponde con el tiempo de ejecución del algoritmo y su crecimiento lineal respecto al tamaño del conjunto de patrones para mapear.[11]

3.3.3. Redes neuronales probabilísticas

Las redes neuronales probabilísticas son un tipo de clasificador que consiste en la construcción de una red neuronal capaz de realizar la clasificación de objetos. La dificultad de este método se encuentra en la obtención de la red neuronal deseada, ya que una vez obtenida, el procesamiento *online* es bastante rápido. En [12] se presenta un algoritmo para encontrar de forma eficiente una red neuronal para reconocer patrones. Estas redes neuronales tienen una estructura de cuatro capas: entrada, *pattern*, sumador y decisión.

El algoritmo presentado se basa en el uso de algoritmos genéticos para construir una red probabilística con una estructura apropiada, basada en 4 pasos.

1. Generación de números aleatorios como valores iniciales para los parámetros de la red.
2. Selección de neuronas de mayor importancia para el parámetro candidato generado, utilizando un algoritmo ortogonal (por aplicar una transformación ortogonal a la matriz de salida de las neuronas).
3. Construcción de las redes probabilísticas utilizando las neuronas correspondientes a cada parámetro y clasificación de ejemplos de entrenamiento.
4. Generación de nuevos parámetros utilizando un algoritmo genético.
5. Se continúa con la siguiente generación en el punto 2.

3.4. Reconocimiento de objetos en Fútbol de Robots

En el fútbol de robot se busca, a partir de un sistema de detección de objetos (conformado a partir de la segmentación y clasificación de objetos de una imagen), identificar los objetos presentes en el campo, en este caso porterías, jugadores, pelota, cancha y postes laterales. Es de interés mencionar que no se consiguió información suficiente acerca de las tecnologías que utilizan los distintos equipos para resolver el reconocimiento de objetos, sin embargo, a continuación se describe un sistema de visión implementado por NUbots[3] para

[3]NUbots compite en la liga ROBOCUP desde 2002 a 2007 en la categoría de perros robóticos y desde 2008 en adelante, en las ligas humanoides, obteniendo el primer puesto de la competencia en 2006, Bremen, Alemania (`http://robots.newcastle.edu.au/robocup.html`)

los agentes robóticos SonyAIBO para fútbol de perros robóticos[13, 14]. Se hace referencia al sistema de confianza utilizado y a dos de los objetos clasificados, que son de interés también para el fútbol de robots humanoide:

3.4.1. Confidence system

Un *confidence system* permite dotar a los objetos identificados de un valor que define el grado de certeza con el que el sistema de visión clasificó el objeto y procesó su posición. Este tipo de sistema busca no rechazar las regiones de interés que no son mapeadas con objetos, sino darle más oportunidades antes de descartarlos. El grado de confianza de un objeto candidato comienza en cero, y a partir del procesamiento realizado sobre dicho objeto, se le asigna nuevos grados de confianza según los resultados del mismo. Una vez acabado el procesamiento, el sistema se queda con aquellos objetos que tengan mayor grado de confianza. Ejemplos de test que pueden ser aplicados en un *confidence system* para la pelota son *Sorrounding colour test* y *Circle fitting*, presentados a continuación.

3.4.2. Reconocimiento de la Pelota

Las propiedades examinadas en los objetos retornados por la clasificación antes de que sean identificados como una pelota incluyen la altura sobre el piso, color debajo del objeto y *roundness*.

Altura del objeto En el contexto del fútbol de robots caninos, difícilmente la pelota abandone el suelo por lo que la altura respecto al piso debería mantenerse constante. Se puede utilizar esta información para identificar pelotas, comparando la altura calculada contra el esperado.

Sorrounding colour test Durante el juego, la pelota siempre está dentro de la cancha, por lo que resulta sensato estudiar el color debajo del *blob* para compararla contra el color de la cancha (verde, blanco).

Circle fitting Es sensato verificar que los objetos reconocidos como pelota posean un contorno redondo. Por ellos, se encuentran los puntos superior, inferior, izquierdo, derecho y las cuatro diagonales del *blob*, tal que tengan el primer píxel del color de la bola. A partir de esos ocho puntos se obtiene entonces un círculo de la forma $<< x, y >, radio >$ y una desviación estándar a partir de dichos puntos. Si el diámetro del círculo obtenido es menor que el ancho del *blob* (según cierto margen), entonces se asume que el *circle fit* no es correcto. En caso de que sí, los cálculos que necesiten de la pelota (distancia, ángulos, elevación, etc) serán llevados a cabo a partir del círculo retornado $<< x, y >, radio >$.

Distancia a la pelota Finalmente, para obtener la distancia del objeto identificado como pelota a la cámara, se compara el diámetro en píxeles de la imagen y el verdadero en cm, obteniendo la distancia.

$$distancia(cm) = \frac{Distancia\,a\,la\,camara(pixel) * Diametro\,de\,pelota(cm)}{Diametro\,visual\,de\,pelota(pixel)} \quad (3)$$

Donde la distancia a la cámara en píxeles es calculada a partir de la posición del objeto en la imagen y el parámetro constante de distancia entre la cámara y el plano de la imagen. Esto es válido debido a que se mantiene la proporción del diámetro de la pelota con la distancia al robot.

3.4.3. Reconocimiento de la Portería

Identificación de la portería Si la clasificación retornó un objeto como candidato a una portería de tamaño bastante grande, se lo define directamente como la misma, ya que no hay objetos que se asemejen y tengan esas características. Así como con la identificación de la pelota, la altura de la portería debe ser constante.

Cut-off detection Por lo general, cuando se detecta una portería, la misma está segmentada por culpa de un robot (el golero). Se deben juntar los conjuntos de objetos y relacionarlos para identificar la portería. Los defectos de orientación de la cámara influyen en la detección de la portería también por lo que se realizan transformaciones para eliminarlos, esto permite elegir 8 puntos (los mismos utilizados para la pelota), que definen la portería.

Distancia a la portería Para obtener la altura de la portería se utiliza la altura en píxeles, que es lo único que se mantiene constante desde los distintos ángulos de la cancha, basándose en la misma idea que en el cálculo de la distancia de la pelota. Sin embargo para poder obtener la distancia es necesario ver la portería en su totalidad, lo que no sucede si se está demasiado cerca a dicho objeto. Para estos casos, la distancia es utilizada como cota máxima en lugar de valor exacto.

$$distancia(cm) = \frac{multiplicador}{altura\,del\,blob\,de\,la\,portería} + desplazamiento \quad (4)$$

Tanto *multiplicador* como *desplazamiento* fueron obtenidos en la práctica para obtener una relación lineal entre la altura en píxeles del blob de la portería y la distancia a la misma.

4. Técnicas de Reconstrucción

4.1. Introducción

Todos los algoritmos para reconstruir una escena 3D a partir de un conjunto de imágenes 2D se basan en que dichas imágenes fueron obtenidas como proyecciones (transformaciones) de los objetos del mundo 3D. Basados en este principio y dados algunos parámetros de la proyección, más datos sobre los objetos identificados en la escena, se intenta obtener un conjunto de transformaciones que realicen el proceso inverso.

El proceso de reconstrucción de la escena se realiza luego de haber preprocesado la imagen con técnicas de segmentación y clasificación. Para cada uno de los objetos identificados en las etapas anteriores, es necesario obtener puntos que permitan definir el plano en el que se encuentran. Para el caso del Fútbol de Robots, estos objetos serían los agentes robóticos, la pelota, los arcos y las marcas de la cancha. Es usual que se realicen simplificaciones en los pasos previos, optimizándolos para la técnica de reconstrucción espacial que se vaya a utilizar.

La cantidad de imágenes que se posee de una misma escena en el tiempo es crucial para la reconstrucción. Al tener una única imagen, solamente es posible realizar una reconstrucción de un objeto en forma plana. Para realizar una reconstrucción más precisa de los objetos es necesario utilizar múltiples imágenes, encontrando puntos comunes entre ellas para luego aplicar diferentes proyecciones y obtener una reconstrucción en tres dimensiones.

Por otro lado, hay situaciones donde no es importante obtener una buena reconstrucción del mundo sino que el factor crucial es el tiempo de procesamiento. En estos casos, se simplifica la estructura espacial obtenida como resultado, en pos de obtener de manera eficiente las posiciones relativas de los objetos. El contexto del Fútbol de Robots se corresponde con el segundo caso.

4.2. Shape from silohuette techniques

Técnica de reconstrucción de objetos que utiliza la intersección de volúmenes donde se usan varias imágenes de un mismo objeto desde distintas posiciones. Como precondición es necesario contar con varias cámaras con distintos puntos de vista del objeto, conociendo a su vez la posición relativa entre ellas.

A partir de la imagen obtenida de cada cámara, se crea un volumen "cónico" con centro en el foco de la cámara y base en la silueta percibida del objeto en la imagen generada por dicha cámara. La intersección de todos los volúmenes cónicos de las distintas imágenes da como resultado un objeto llamado *visual hull*[15], definido como la máxima reconstrucción del objeto que devuelve la misma silueta que el objeto original, desde todos los puntos de vista (cámaras).

Si se utiliza proyección perspectiva (cámaras), se intersecan conos; por el contrario, si se utiliza la proyección ortográfica, se intersecan cilindros.

Esta técnica de reconstrucción supone que el objeto puede ser separado del fondo[4].

El defecto principal de las técnicas *shape from silohuette* es que no pueden detectar concavidades en los objetos a reconstruir. A su vez, presentan fallas cuando, en la escena, aparecen múltiples obstáculos que no permiten ver la totalidad de los objetos.

4.3. Voxel Coloring

Un vóxel es un píxel llevado a tercera dimensión. Además de tener información del color, un vóxel contiene la información de su ubicación en tres dimensiones. A partir de imágenes se busca crear vóxeles para reconstruir así los objetos, con su respectivo color, modelando así el objeto como un conjunto de partículas que contienen información no solo de su posición sino también de su color; obteniendo así una definición del objeto mejor que un *visual hull*.

Como defecto cabe destacar que este tipo de técnica no es buena para escenas grandes debido a la cantidad de información que se debe representar, ni objetos con grandes diferencias en escalas debido a la granularidad con la que se deben describir los objetos[5].

4.4. Direct Linear Transformation (DLT) Method

El método DLT permite, a partir de dos imágenes, obtener una función que retorna la distancia de un punto de la escena a una de las cámaras. Para ello es necesario identificar puntos de control que se correspondan en las dos imágenes. Si bien se obtienen buenos resultados en la práctica, cuenta con desventajas como el tiempo de procesamiento y la obtención de los puntos correspondientes entre las imágenes.[16]

4.4.1. Pinhole camera model

La proyección de un punto en el plano de la imagen, coincide con la proyección de todos los puntos que pertenecen a la recta formada por C_x y el centro óptico de la cámara, de ahora en más llamado r_x. Por cada cámara utilizada se obtendrá un r_x distinto.

4.4.2. Stereo Image

Mediante el uso de dos imágenes podemos hallar la distancia a la que se encuentra un punto. La técnica DLT propone, mediante 8 puntos correlacionados entre las dos imágenes, encontrar la profundidad de los puntos que aparecen en ambas imágenes. Si se llegase a poder relacionar un punto de una imagen con su correspondiente en la segunda, se podría obtener la intersección de las rectas r_x en las distintas cámaras y encontrar así el punto x sin proyectar. Sin embargo, es muy costoso comparar uno por uno todos los puntos de la proyección obtenida por la primera cámara con los puntos pertenecientes a la segunda proyección (de la segunda cámara).

Gracias a los 8 puntos de control obtenemos una función que devuelve para un punto, la proyección de la recta formada por r_x de la primera cámara, sobre

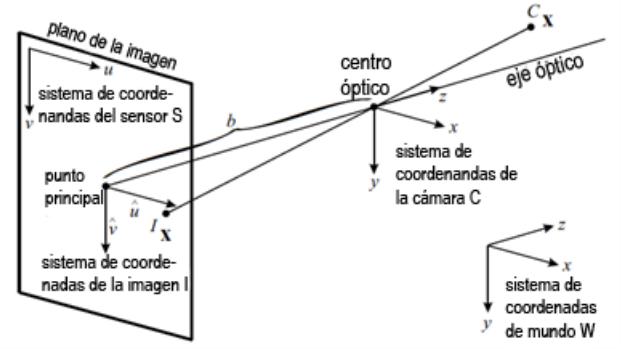

Figura 5: Sistemas de coordenadas en DLT

C_x**:** x en el sistema de coordenadas de la cámara.

W_x**:** x en el sistema de coordenadas de mundo.

I_x**:** x en el sistema de coordenadas de la imagen.

la segunda imagen. El siguiente paso luego de obtener esta recta es comparar el punto deseado solamente contra aquellos puntos pertenecientes a la proyección de la recta r_x, reduciendo así el espacio de búsqueda. Esta técnica es muy costosa, dependiendo de la resolución de las imágenes con que se trabaje.

4.5. Vector based calculation

Algoritmo presentado en [17] y de aplicación específica para el fútbol de robots.

Para calcular la distancia y el ángulo del robot a los objetos de la cancha se definen dos sistemas de coordenadas. El primero dinámico, cuyo origen coincide con la proyección en el plano del suelo del robot (x, y, z), y el eje y coincide con el frente del robot. El segundo, localizado en el centro del plano de la imagen (i, j, v).

El método consiste en proyectar el punto de apoyo T de los objetos del plano de la imagen (i, j), en el plano del suelo (x, y), obteniendo el punto Q, con lo que podemos reducir el problema a encontrar el módulo del vector $\overrightarrow{OQ}$, y el ángulo Θ.

Para realizar los cálculos pertinentes es necesario conocer tanto el ángulo en el que enfoca la cámara con el plano x, y conocido de aquí en más como *pan*, así como con en el plano y, z conocido como *tilt*.

A partir del punto E donde se encuentra localizada la cámara, se obtiene el vector $\hat{v}$ unitario y perpendicular al plano de la imagen como:

$$\hat{v} = (cos(pan) * sin(tilt),\ cos(pan) * cos(tilt),\ E_z - sin(pan)) \qquad (5)$$

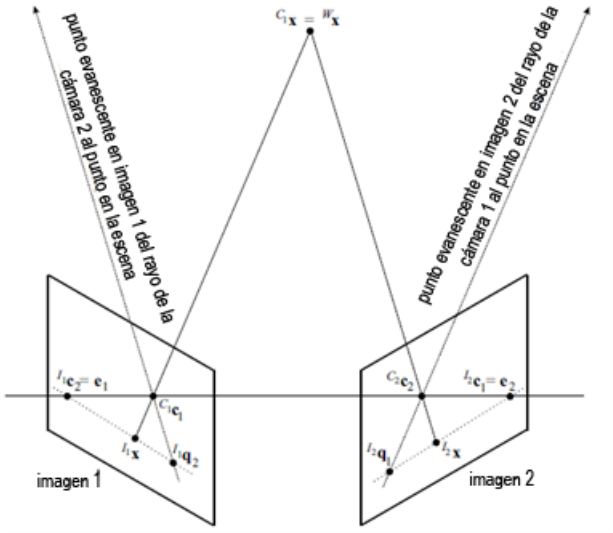

Figura 6: Puntos evanescentes o puntos de fuga

Sea H el punto medio en el plano de la imagen, $f = |\overrightarrow{EH}|$ es la distancia focal del plano de la imagen, constante propia de cada cámara y obtenible experimentalmente. Definimos entonces:

$$\overrightarrow{h} = \overrightarrow{e} + f * \overrightarrow{v} \tag{6}$$

$$\hat{i} = \hat{k} \times \hat{v} \tag{7}$$

$$\hat{j} = \hat{v} \times \hat{i} \tag{8}$$

Sin embargo, el punto T en el plano de la imagen queda definido por (T_x, T_y) y $\overrightarrow{t'}$ puede ser calculado como:

$$\overrightarrow{t'} = \overrightarrow{h} + T_x * \hat{i} + T_y * \hat{j} \tag{9}$$

y

$$\overrightarrow{\delta} = \overrightarrow{t'} - \overrightarrow{e} \tag{10}$$

pudiendo obtener entonces el vector unitario $\hat{\delta}$ a partir de $\overrightarrow{\delta}$.
Sabemos además que:

$$\overrightarrow{q} = \overrightarrow{e} + \overrightarrow{EQ} \Rightarrow \overrightarrow{q} = \overrightarrow{e} + (\frac{E_z}{cos(\gamma)})\hat{\delta} \tag{11}$$

donde E_z es la altura del robot y $cos(\gamma)$ es el producto punto de los vectores unitarios $\hat{k}$ y $\hat{\delta}$. Por lo tanto, podemos obtener:

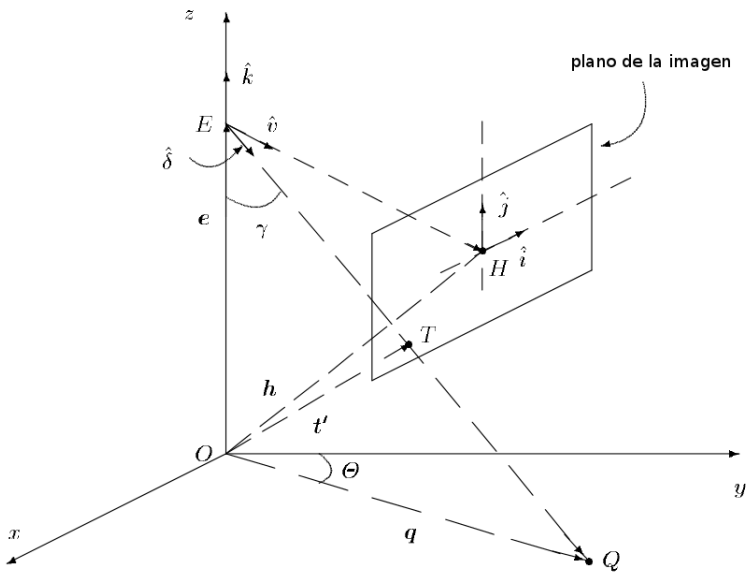

Figura 7: Modelo para Vector based calculation

$$\Theta = arctan(\frac{Q_y}{Q_x}) \tag{12}$$

y

$$|OQ| = \sqrt{Q_x^2 + Q_y^2} \tag{13}$$

Donde las ecuaciones 12 y 13 son la solución del problema inicial.

A partir del modelo de *Vector based calculation* se obtienen buenos resultados, sin embargo presenta como desventaja el tener que conocer tanto la orientación de la cámara como la altura a la que está posicionada.

5. Localización

La tarea de localización consiste en situar espacialmente un objeto en un modelo del mundo, y tiene particular importancia en los sistemas de navegación que hacen uso de la visión por computador.

Orientado al Fútbol de Robots, interesa conocer la posición de todos los objetos que pueden variar sus posiciones, es decir, los agentes robóticos y la pelota.

5.1. Localización mediante Landmarks

Un *landmark* es una marca conocida en el entorno, una característica de interés perceptualmente distintiva sobre un objeto o lugar. Si se encuentra una *landmark* en el mundo y en el modelo que tiene el agente de él, entonces, el robot puede localizarse en el mundo[18][19].

5.1.1. Triangulación

Una de las técnicas presentadas en [18] busca medir ángulos entre *landmarks* para obtener así la posición del robot y su orientación. Para ello se debe contar de antemano con un mapa donde se localicen los *landmarks* con los que se va a trabajar. La idea detrás de medir los ángulos entre los *landmaks* es obtener el arco capaz que tiene como base 2 *landmarks*; de tal manera de conocer el arco capaz al que pertenece el robot.

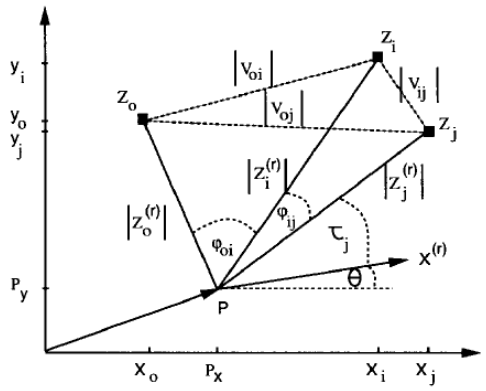

Figura 8: Triangulación

Para lograr la localización es necesario:

1. Identificar los *landmarks* del ambiente mediante los sensores.
2. Encontrar los correspondientes *landmarks* en el mapa, y por ende la posición de cada uno de ellos.

3. Medir el ángulo de separación entre los *landmarks*.

4. Computar la posición eficientemente.

Si 1, 2 y 3 se ejecutan sin error, con tres *landmarks* el agente puede obtener su posición. Esto se cumple a menos que todos los *landmarks* y el agente pertenezcan a un círculo o estén alineados.

Usando el teorema del coseno, podemos obtener las siguientes ecuaciones:

$$\begin{aligned} |v_{0i}|^2 &= |z_0^{(r)}|^2 + |z_i^{(r)}|^2 - 2|z_0^{(r)}|^2|z_i^{(r)}|^2 \cos\varphi_{i0} \\ |v_{0j}|^2 &= |z_0^{(r)}|^2 + |z_j^{(r)}|^2 - 2|z_0^{(r)}|^2|z_j^{(r)}|^2 \cos\varphi_{j0} \\ |v_{ij}|^2 &= |z_i^{(r)}|^2 + |z_j^{(r)}|^2 - 2|z_i^{(r)}|^2|z_j^{(r)}|^2 \cos\varphi_{ji} \end{aligned} \tag{14}$$

Donde z_0, z_i y z_j son los tres *landmarks* identificados en la imagen (conocidos); $|z_0^{(r)}|$, $|z_i^{(r)}|$ y $|z_j^{(r)}|$ se corresponden con las distancias entre los respectivas *landmarks* y el robot (desconocidas); φ_{i0} , φ_{j0} y φ_{ji} son los ángulos entre los distintas *landmarks* con eje en el robot r (conocidos); y $|v_{0i}|$, $|v_{0j}|$ y $|v_{ij}|$ son las distancias (conocidas) entre los respectivos *landmarks*. De esta manera se obtiene un sistema no lineal de ecuaciones que puede ser resuelto utilizando mínimos cuadrados. Una vez obtenida la distancia de los *landmarks* al robot r, se puede expresar dichas distancias, ahora conocidas, a partir de las coordenadas (x, y) (no conocidas) del robot y de los respectivos *landmarks*, obteniendo que:

$$\begin{aligned} |z_0^{(r)}|^2 &= (x_0 - p_x)^2 + (y_0 - p_y)^2 \\ |z_i^{(r)}|^2 &= (x_i - p_x)^2 + (y_i - p_y)^2 \\ |z_j^{(r)}|^2 &= (x_j - p_x)^2 + (y_j - p_y)^2 \end{aligned} \tag{15}$$

Donde p_x y p_y son las coordenadas del robot r en los ejes x e y respectivamente en el sistema de coordenadas de mundo y x_n y y_n son las coordenadas del *landmark* z_n en los ejes x e y respectivamente, en el sistema de coordenadas de mundo.

5.1.2. Estimación de posición lineal

La segunda técnica que descrita en [18] consiste en pensar la posición de los *landmarks* como números complejos en las coordenadas del robot, donde el origen coincide con la posición del robot.

Para cada *landmark* $z_i^{(r)}$ con $i = 1..n$ se define la expresión:

$$z_i^{(r)} = l_i e^{j\tau_i} \tag{16}$$

Donde l_i es la distancia, desconocida, del robot r al *landmark* $z_i^{(r)}$, el ángulo τ_i es el ángulo medio entre $z_i^{(r)}$ y el eje de coordenada $x^{(r)}$, y $j = \sqrt{-1}$.

Como $\tau_i - \tau_0$ es φ_i, para $i = 1..n$, al dividir $z_i^{(r)}$ entre $z_0^{(r)}$ se obtiene:

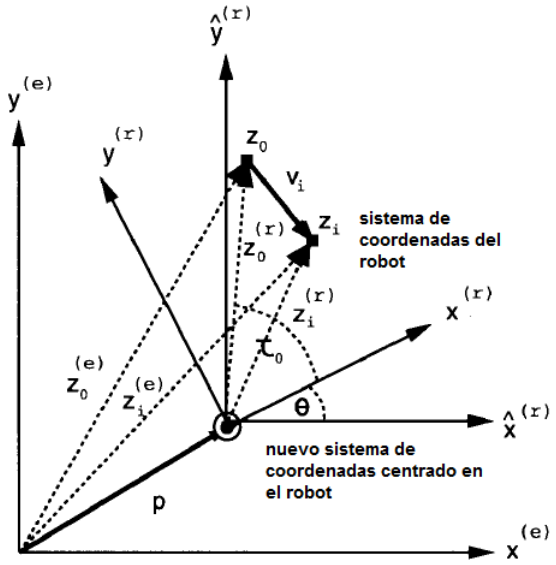

Figura 9: Estimación de posición lineal

$$\frac{z_i^{(r)}}{z_0^{(r)}} = \frac{l_i}{l_0} e^{j(\tau_i - \tau_0)} = \frac{l_i e^{j\varphi_i}}{l_0} \tag{17}$$

Si $z_i^{(r)} = z_0^{(r)} + v_i^{(r)}$, la igualdad anterior se convierte en:

$$\begin{aligned} \frac{z_0^{(r)} + v_i^{(r)}}{z_0^{(r)}} &= \frac{l_i e^{j\varphi_i}}{l_0} \ para\, i = 1..n \\ \frac{1}{z_0^{(r)}} &= \frac{l_i}{l_0} \frac{1}{v_i^{(r)}} e^{j\varphi_i} - \frac{1}{v_i^{(r)}} \ para\, i = 1..n \end{aligned} \tag{18}$$

Si quitamos la dependencia con $z_0^{(r)}$ obtenemos:

$$\frac{l_k}{l_0} \frac{1}{v_k^{(r)}} e^{j\varphi_k} - \frac{1}{v_k^{(r)}} = \frac{l_i}{l_0} \frac{1}{v_i^{(r)}} e^{j\varphi_i} - \frac{1}{v_i^{(r)}} \ con\, k, i = 1..n,\ k \neq i \tag{19}$$

Sustituyendo $\frac{l_k}{l_0}$ por r_k obtenemos finalmente que:

$$r_k \frac{1}{v_k^{(r)}} e^{j\varphi_k} - \frac{1}{v_k^{(r)}} = r_i \frac{1}{v_i^{(r)}} e^{j\varphi_i} - \frac{1}{v_i^{(r)}} \tag{20}$$

Donde el *landmark* z_i es descrito por el vector $z_i^{(r)}$, $r_i = {}^{|z_i^{(r)}|}/_{|z_0^{(r)}|}$ y $v_i^{(r)} = z_i^{(r)} - z_0^{(r)}$

Eligiendo el sistema de coordenadas del robot, de tal manera que los ejes queden paralelos a las coordenadas de mundo podemos reescribir la última ecuación como:

$$r_k \frac{1}{v_k^{(e)}} e^{j\varphi_k} - \frac{1}{v_k^{(e)}} = r_i \frac{1}{v_i^{(e)}} e^{j\varphi_i} - \frac{1}{v_i^{(e)}} \tag{21}$$

Donde las únicas incógnitas son $r_1, .., r_n$.

Este juego de ecuaciones puede ser transformado en una ecuación matricial $Ar = c$ donde el vector $r = (r_1, .., r_n)$ es el vector de las incógnitas r, largos de los vectores $z_i^{(r)}$, c es un vector de dimensión $n(n-1)$ creado con la diferencia de los números complejos $c_i = (c_{x_i}, c_{y_i}) = {}^1/_{v_i^{(r)}}$ y A es una matriz $n(n-1) \times n$ generada con los números complejos $b_i = (b_{xi}, b_{yi}) = c_i e^{j\varphi_i}$ de manera que:

$$A = \begin{matrix} b_1 & -b_2 & 0 & 0 & ... & 0 & 0 \\ b_1 & 0 & -b_3 & 0 & ... & 0 & 0 \\ b_1 & 0 & 0 & -b_4 & ... & 0 & 0 \\ ... & & & & & & \\ b_1 & 0 & 0 & 0 & ... & 0 & -b_n \\ -b_1 & b_2 & 0 & 0 & ... & 0 & 0 \\ 0 & b_2 & -b_3 & 0 & ... & 0 & 0 \\ 0 & b_2 & 0 & -b_4 & ... & 0 & 0 \\ ... & & & & & & \\ 0 & b_2 & 0 & 0 & ... & 0 & -b_n \\ ... & & & & & & \\ -b_1 & 0 & 0 & 0 & ... & 0 & b_n \\ 0 & -b_2 & 0 & 0 & ... & 0 & b_n \\ 0 & 0 & -b_3 & 0 & ... & 0 & b_n \\ ... & & & & & & \\ 0 & 0 & 0 & 0 & ... & -b_{n-1} & b_n \end{matrix} \quad y \;\; c = \begin{matrix} c_1 - c_2 \\ c_1 - c_3 \\ c_1 - c_4 \\ ... \\ c_1 - c_n \\ c_1 - c_1 \\ c_1 - c_3 \\ c_1 - c_4 \\ ... \\ c_2 - c_n \\ ... \\ c_n - c_1 \\ c_n - c_2 \\ c_n - c_3 \\ ... \\ c_n - c_{n-1} \end{matrix} \tag{22}$$

Una solución de la ecuación $Ar = c$ conduce a un vector r cuyos componentes r_i pueden utilizarse para resolver la ecuación:

$$\frac{1}{z_0^{(r)}} = r_i \frac{1}{v_i^{(e)}} e^{j\varphi_i} - \frac{1}{v_i^{(e)}} \tag{23}$$

Una vez obtenido el valor de $z_0^{(r)}$ utilizando algún i, la posición del robot p, es:

$$p = z_0^{(e)} - z_0^{(r)} \tag{24}$$

5.2. Monte Carlo Localization

Es de gran interés mencionar la técnica de localización por Monte Carlo, utilizada ampliamente por los distintos equipos de la liga de Fútbol de Robots y descrita en [20].

La localización por Monte Carlo tiene sus bases en el algoritmo de localización de Markov el cual utiliza la idea de computar la distribución de probabilidad de todas las posibles posiciones en el ambiente. Es conocida alternativamente como *bootstrap filter*, *Monte-Carlo filter*, algoritmo de condensación y *the survival of the fittest algorithm;* todos estos métodos son llamados genéricamente como filtros de partículas.

5.2.1. Markov Localization

Sea $l =< x, y, \theta >$la definición de una posición donde x e y son las coordenadas cartesianas del robot en coordenadas cartesianas de mundo y θ la orientación del robot; la distribución $Bel(l)$ muestra las creencias del robot de encontrarse en el estado l. Inicialmente, si la posición inicial del robot es conocida, $Bel(l)$ se encuentra centrada en dicha posición; de lo contrario, se encuentra uniformemente distribuida, reflejando la incertidumbre de la posición. A medida que el robot opera, $Bel(l)$ es recalculada, a partir de dos modelos probabilísticos que incorporan los movimientos del robot, y el resultado del sensado.

Movimiento del Robot Los movimientos del robot se modelan a partir de una probabilidad condicional, $P(l|l', a)$, que define la probabilidad de encontrarse en la posición l dado que anteriormente el agente se encontraba en la posición l' y se ejecutó el movimiento a. $Bel(l)$ es actualizada de la siguiente manera:

$$Bel(l) = \int P(l|l', a)Bel(l')dl' \tag{25}$$

Lectura de Sensores La lectura de los sensores se utilizan a partir de la regla de Bayes:

$$Bel(l) = \alpha P(s|l)Bel(l) \tag{26}$$

Donde s es la información obtenida de los sensores y $P(s|l)$ es la probabilidad de sensar s dado que el agente se encuentra en la posición l. α es un normalizador que asegura que $Bel(l)$ integra a 1.

5.2.2. Implementación de Monte Carlo Localization

La idea clave es representar a $Bel(l)$ como un conjunto de N partículas $S = s_i|i = 1..N$también conocidas como *muestras al azar con pesos*. Un conjunto S, constituye una aproximación discreta de una distribución de probabilidad.

Estas partículas o muestras se definen de la siguiente manera:

$$<< x, y, \theta >, p >$$

donde $< x, y, \theta >$ denotan la posición l de un robot y $p \geq 0$ es un factor de peso, de manera que $\sum_{n=1}^{N} p_n = 1$.

Análogo a la localización de Markov, MCL actualiza a $Bel(l)$ en dos fases:

Movimiento del Robot Se generan N nuevas muestras que aproximan la posición del robot luego de la acción realizada. Cada una de estas muestras se genera escogiendo al azar una muestra del conjunto previamente computado, con probabilidad determinada por sus p_i. Sea l' la posición de la muestra escogida, la nueva muestra con posición l se genera escogiendo un solo valor de $P(l|l', a)$ donde a es la acción realizada. El peso de la nueva muestra será N^{-1}.

Lectura de Sensores La lectura de los sensores se incorporan recalculando los pesos de las muestras escogidas. Dada una muestra $< l, p >$:

$$p = \alpha P(s|l) \tag{27}$$

donde s es el la información obtenida de los sensores y α es un normalizador de manera que $\sum_{n=1}^{N} p_n = 1$.

En la práctica, es de utilidad colocar algunas posiciones al azar del robot (uniformes en la cancha) en el conjunto de muestras S de manera de incluir ruido en el modelo y de esta manera poder evitar mínimos locales, siempre que el factor p de estas muestras sea pequeño para evitar impactar en gran medida en la distribución que se intenta discretizar a partir de S.

Como ventajas, la localización por Monte Carlo cuenta con:

- Es capaz de aproximar distribuciones de probabilidad arbitrarias.
- Reduce drásticamente la cantidad de memoria que utiliza la localización de Markov en su modelo *grid-based*, necesario para resolver la localización inicial en caso de no poseer información de la ubicación inicial del robot.
- Es más preciso que la localización de Markov que utiliza un tamaño de celda fijo, ya que el estado representado por las muestras en MCL no es discretizado.
- Más sencillo de implementar que la localización de Markov.
- Por ser un algoritmo *on-line*, se puede modificar la cantidad de muestras que se sortean en cada paso, adaptándose a la situación actual del agente.
- Es capaz de resolver el problema de localización global del agente si no se conoce la posición inicial del mismo. Para ello se toma $Bel(l)$ uniformemente distribuida en la cancha.

Como desventajas, el modelo requiere una gran cantidad de puntos para la localización inicial, y una buena aproximación de la distancia que recorre un robot a partir del movimiento del robot.

6. Agentes Inteligentes o Racionales

Un agente es todo aquello que puede percibir su entrono mediante sensores y responder o actuar en el ambiente por medio de actuadores. Un agente de software es un programa de computadora capaz de realizar acciones de forma autónoma en el ambiente en que está situado; sin embargo, se conoce como agente inteligente, o agente racional, a aquellos agentes que se comporten de manera racional frente a la secuencia de percepciones del ambiente y del conocimiento incorporado. El concepto de racionalidad refiere, en este contexto, a la característica del agente de tomar la acción correcta; más específicamente, de tomar la acción que maximice su medida de rendimiento[21].

¿Qué papel juega la visión en el agente? La visión forma parte del conjunto de sensores que tiene un agente robótico. Mediante la visión, el agente busca identificar objetos, características o actividades en el ambiente. Una visión robótica ideal, debería poder identificar cualquier clase de objeto, en ambientes no conocidos por el agente, siendo lo suficientemente robusta para poder soportar distintas condiciones de luz y ruido en las imágenes capturadas, consumiendo la menor cantidad de recursos computacionales.

7. Definiendo el ambiente – Fútbol de Robots

Los ambientes son los entornos de trabajo, el contexto donde el agente está situado. Un ambiente presenta un "problema" para el cual el agente racional es la "solución".[21]

7.1. Propiedades de los ambientes

Una forma de especificar un ambiente es a partir de las siguientes propiedades:

Observable vs Parcialmente observable Si en todo momento, se obtienen los datos de todos los objetos del ambiente necesarios para tomar la mejor decisión, entonces se considera que el ambiente es totalmente observable. En caso contraria se define el ambiente como parcialmente observable.

Determinista vs Estocástico Un ambiente se dice que es determinista, si a partir de los estados anteriores es posible definir el siguiente estado del sistema.

Episódico vs Secuencial Si la experiencia del agente se divide en episodios, donde la actuación del agente es independiente de los episodios, se define el ambiente como Episódico. De lo contrario, se lo define como Secuencial.

Discreto vs Continuo La cantidad de percepciones que pueden tener los agentes y acciones distintas (y claramente discernibles) pueden ser discretas (limitadas), o continuas.

Estático vs Dinámico Si, mientras los agentes deliberan su próxima acción existe la posibilidad de que el ambiente cambie, se clasifica el mismo como Dinámico. En el caso contrario, se clasifica como Estático.

Agente individual vs Multiagente Si en el ambiente hay más de un agente se define como Multiagente, los cuales pueden tener un comportamiento cooperativo o competitivo a su vez. De lo contrario es un ambiente con Agente individual.

Ejemplos de ambientes

Ambiente	Observable	Determ.	Epis.	Est.	Discr.	Agentes
Crucigrama	Totalmente	Sí	No	Sí	Sí	Indiv.
Ajedrez con reloj	Totalmente	Sí	No	Semi	Sí	Multi
Ajedrez sin reloj	Totalmente	Sí	No	Sí	Sí	Multi
Conducir un taxi	Parcialmente	No	No	No	No	Multi
Robot industrial	Parcialmente	No	Sí	No	No	Indiv.

7.2. Clasificación de un partido de Fútbol de Robots de Robocup

Parcialmente observable: Según las reglas establecidas en [1] para la liga humanoide kid-size, en la que se estipula que los robots deben llevar cámaras locales montadas en la cabeza, a las que se les restringe el ángulo de paneo, el Fútbol de Robots se define como Parcialmente Observable.

Estocástico: En el Fútbol de Robots, no se puede definir el siguiente estado del sistema a partir del anterior ya que al estar en juego las estrategias de 2 equipos y sin conocerse entre si, no podrían preveér que acción realizarán los oponentes; ni eventuales problemas con su equipo como caídas, etc.

Secuencial: La experiencia de los agentes durante un partido no se divide en episodios.

Continuo: Existen infinidad de percepciones y acciones distintas en un partido de Fútbol de Robots, por lo que se define el ambiente como Continuo.

Dinámico: En el Fútbol de Robots mientras un agente delibera su próxima acción, los demás agentes, tanto compañeros como contrarios, son libres de interactuar con el ambiente, por lo que éste puede sufrir modificaciones.

Multiagente - competitivo, cooperativo: En un partido de la liga *humanoide – kid size* de RoboCup, participan dos equipo rivales que buscan objetivos distintos (competitivo), pero dentro de un mismo equipo, los agentes cooperan para obtener un objetivo en común (cooperativo), clasificando al ambiente como Multiagente.

8. Robótica Cooperativa

8.1. Taxonomía de los sistemas robóticos cooperativos

David A Gustafson y Eric Matson en [22], caracterizan los sistemas robóticos cooperativos según 3 dimensiones:

1. Información (información del agente y de los demás en el sistema)

   I=1 Designa que el robot solamente conoce información del estado local.

   I=2 Designa que el robot conoce información del estado de sus vecinos.

   I=3 Designa que el robot conoce información del estado global.

2. Comunicación (Tipo de comunicación con los demás agentes)

   C=0 Es la situación donde solo existe comunicación implícita. Por ejemplo, un robot sensando las acciones de otro robot.

   C=1 Es la situación donde un mensaje es transmitido (en modo *broadcast*) a otros robots, sin conocimiento de los receptores o si llega o no el mensaje. Esto incluye la comunicación *hormone-type* donde un mensaje es retransmitido por intermediarios. La idea detrás de esto es la comunicación de una sola vía y la falta de respuesta requerida.

   C=2 Es la transmisión del mensaje a un robot específico. Un mensaje *broadcast* con el nombre del receptor en él es un ejemplo de este nivel. Esta situación requiere el conocimiento local de los vecinos. El mensaje puede o no requerir que el receptor conozca al remitente del mensaje. Sin embargo, la recepción del mensaje es garantizada bajo situaciones normales.

   C=3 Trata de las interacciones del tipo *human-like* entre robots. Con poca semántica o sintaxis predefinida.

3. Goal (Conocimiento sobre el objetivo grupal e individual)

   G=1 Es la situación donde un robot solo conoce su propia meta o tarea. Si bien esto se asocia usualmente con los escenarios $I = 1$ y $C = 1$, un equipo típico de fútbol de robot podría tener robots con tareas específicas pero con conocimiento de las posiciones de los jugadores vecinos y con mensajes *broadcast* a los demás jugadores.

   G=2 Es la situación donde un robot conoce la meta de los robots vecinos pero sin conocer la de todos.

   G=3 Es la situación donde todos los robots conocen la meta de todos los demás robots.

CMUnited Robot Soccer[23] es un equipo de fútbol de robots. Cada robot tiene conocimiento global del estado, los robots utilizan mensajes del tipo human-like,

y cada robot conoce el objetivo global. CMUnited podría ser clasificado como $I = 3$, $C = 2$, $G = 3$.

CSFreiburg Robot Soccer Team[24] es otro equipo de fútbol de robots exitoso, obteniendo el primer puesto de la competencia tres veces. Cada robot utiliza sus propios sensores, transmitiendo datos a un procesador (no robótico) que realiza un *broadcast* con información global. Sin embargo, no se garantiza que esta información sea recibida correctamente. CSFreiburg podría ser clasificado como $I = 2$, $C = 1$, $G = 1$.

8.2. Interacciones Multiagente

En un encuentro multiagente, que incluye 2 agentes i y j la pregunta que deben responder estos agentes es: ¿qué hacer?

Dados 2 conjuntos de estados que puede tomar el ambiente Ω_1 y Ω_2, si cualquier estado en Ω_1 es preferido sobre cualquier estado en Ω_2 por el agente i, se dice que Ω_1 domina a Ω_2 para el agente i.

Dada una estrategia S, $S*$ es el conjunto de estados que puede tomar el ambiente a causa de que i ejecute la estrategia S. Dadas 2 estrategias S_1 y S_2, si S_1* domina a S_2*, la decisión de qué estrategia utilizar debería tener como resultado la estrategia S_1.

Dos estrategias S_1 y S_2 están en equilibrio de Nash, si:

(1) Bajo la premisa de que el agente i ejecuta la estrategia S_1, el agente j no puede obtener un mejor resultado que ejecutando S_2, y

(2) Bajo la premisa de que el agente j ejecuta la estrategia S_2, el agente i no puede obtener un mejor resultado que ejecutando S_1.

Esta forma mutua de equilibrio es importante, porque restringe a los agentes en un par de estrategias. Un ejemplo donde se puede ver el equilibrio de Nash en la vida diaria, es en el tránsito. En una calle doble vía, si el auto A_1 asume que los demás autos circularán por la derecha, no puede ejecutar una estrategia que lo beneficie más, que circular por la derecha. Este pensamiento es análogo para los demás coches. Sin embargo, cabe destacar que no todo escenario de interacción tiene un equilibrio de Nash; y algunos escenarios de interacción tienen más de un equilibrio de Nash. Esto sucede en el ejemplo brindado anteriormente; otro posible equilibrio de Nash sería que todos los autos circularan por la izquierda.

Un escenario de interacción competitiva es un escenario donde un agente sólo puede mejorar su estado, a expensas de otros agentes, es decir, dado el conjunto de estados del ambiente Ω y los agentes i, j:

$$\omega >_i \omega' \; sii \; \omega' >_j \omega \;\; \forall \omega, \omega' \epsilon \Omega \tag{28}$$

Los encuentros de *zero-sum* son aquellos en los que, para cualquier resultado particular, la utilidad de los agentes suma cero:

$$utilidad_i(\omega) + utilidad_j(\omega) = 0 \;\; \forall \omega \epsilon \Omega \tag{29}$$

Todo escenario *zero-sum* es estrictamente competitivo, es decir no hay posibilidad de comportamiento cooperativo [25].

8.2.1. Relaciones de dependencia en sistemas multiagente

Los sistemas multiagente plantean relaciones entre los agentes que componen el sistema, en particular [26] categoriza las relaciones de dependencias del sistema a partir de si un agente necesita de los otros para lograr su meta:

Independencia: Sin dependencias entre los agentes, cada uno es autosuficiente para poder lograr sus metas.

Dependencia Unilateral: Un agente depende de otro para lograr su objetivo, pero no viceversa.

Mutua Dependencia: Ambos agentes dependen uno del otro, con respecto a alguna meta en común.

Dependencia Recíproca: Un agente depende del otro agente y viceversa, pero para metas distintas.

8.2.2. Logrando acuerdos

Según si un agente depende de los demás para lograr sus metas o no, le resultará de interés ponerse de acuerdo con los agentes que depende, de manera de llegar a un común acuerdo de la mejor forma de conseguir su objetivo. La habilidad de lograr acuerdos (sin una tercera parte dictando los términos) es una capacidad fundamental de los agentes autónomos inteligentes.

Cuando se diseñan protocolos de comunicación, típicamente se busca que sean libres de *deadlocks* y *livelocks*. Sin embargo, en los sistemas multiagente, al momento de diseñar protocolos de negociación, las propiedades que se buscan son un poco diferentes. [27] lista alguna de las propiedades mas relevantes:

- Garantizar el acuerdo: Un protocolo garantiza que se llegará a un acuerdo si en un número finito de pasos, el acuerdo se logra.
- Maximizar el bienestar social: Intuitivamente, un protocolo maximiza el bienestar social, si asegura que cualquier resultado maximiza la suma de las utilidades de los participantes de la negociación.
- Pareto eficiente: El resultado de una negociación se dice Pareto eficiente, si no hay otro resultado posible que logre aumentar la utilidad de un agente, sin quitarle utilidad a los demás. Si bien esta propiedad es similar a Maximizar el bienestar social, difiere en que este último puede tener como resultado sacrificar utilidad por parte de un agente, para que otro/s agentes aumenten su utilidad llevando a un aumento de la suma de las utilidades.

- Racionalidad individual: Un protocolo se dice que es individualmente racional si seguir el protocolo está dentro de los mejores intereses de los participantes de la negociación.
- Estabilidad: Un protocolo es estable si provee a todos los agentes con un incentivo para que se comporten de una determinada manera. (Como el equilibrio de Nash definido en la sección 8.2, donde el incentivo es obtener el mejor resultado al ejecutar la estrategia en equilibrio).
- Simplicidad: Un protocolo "simple" es uno que haga obvia la estrategia apropiada para un participante de la negociación.
- Distribución: Un protocolo debería ser diseñado para distribuir los posibles puntos de falla, y minimizar la comunicación entre los agentes.

Subastas Una subasta es un procedimiento para lograr un acuerdo. Es realizado entre un agente conocido como el subastador, y una colección de agentes conocidos como los apostadores. La meta de la subasta es, para el subastador, conseguir que un apostador se lleve el objeto, le sea asignada una tarea, etc. (donde el subastador tiene como objetivo maximizar el valor por el cual el objeto es entregado, mientras que los apostadores deben minimizarla); y para el apostador, la meta es llevarse el objeto, le sea asignada una tarea, etc, por un valor menor o igual al que está dispuesto a pagar, llamado valor privado.

Existen distintos tipos de subastas, clasificadas según las siguientes características[25]:

Determinación del ganador *First-price* vs *Second-price*: En las subastas *First-price*, el apostador que haya ganado la subasta deberá pagar el mayor valor que se ofreció por el objeto. En cambio, en las subastas *Second-price*, el ganador debe pagar la segunda mayor oferta.

Conocimiento global de las apuestas *Sealed-bid* vs *Open-cry*: En las subastas *Sealed-bid* los apostadores no conocen cuanto ofrecen los demás apostadores por el objeto, sin embargo, en las subastas *Open-cry* todos los apostadores conocen las ofertas de los demás.

Forma de apostar *Single round bidding* vs *Ascending* vs *Descending*: En las subastas *Single round bidding*, los apostadores solo tienen derecho a apostar una única vez. En las subastas *Ascending*, los apostadores empiezan con su menor oferta, aumentándola en los sucesivos turnos. Por último, en las subastas *Descending* los apostadores comienzan con su mayor oferta, y van disminuyéndola en los sucesivos turnos.

Ejemplos de Subastas

Subastas Inglesas Son el tipo de subastas más conocidas, correspondiéndose según la clasificación mencionada a las categorías: *first-price*, *open-cry*, *ascending*. La estrategia dominante[4] en este tipo de subastas es que el apostador incremente en pequeñas cantidades la apuesta actual hasta que gane la apuesta, o llegue al máximo valor que está dispuesto a pagar. [25] comenta que los ganadores de este tipo de subastas sufren la "maldición del ganador", ya que el ganador de la subasta debería estar feliz por obtener el objeto, pero a su vez preocupado, porque ninguna otra persona valoró el objeto tan alto como él.

Subastas Holandesas Son un ejemplo de subastas *open-cry*, *descending*, *first-price*, donde el subastador comienza ofreciendo el objeto desde un precio elevado y luego va disminuyendo el precio hasta que un apostador acepta el objeto por el precio ofrecido. Este tipo de subastas también sufren la "maldición del ganador", y no tienen una estrategia dominante.

Subasta de Vickrey Las subastas de Vickrey son subastas *second-price*, *sealed bid*, *single round bidding*, es decir, se componen de una sola ronda de negociación donde todos los apostadores ofertan en secreto por el objeto, y si bien la subasta la gana el apostador que ofreció más, solo debe pagar tanto como la segunda mayor oferta. Las subastas de vickrey tienen como estrategia dominante el "decir la verdad.

Negociaciones Como contraparte de una subasta, una negociación es un procedimiento para lograr un acuerdo que se diferencia de la primera principalmente en no contar con un agente mediador. Se compone de:

1. Un conjunto de negociación, el cual representa el espacio de posibles respuestas que los agentes pueden dar.
2. Un protocolo que define la propuestas legales que los agentes pueden realizar, como por ejemplo una función de *prior negotiation history*[5].
3. Una colección de estrategias, una para cada agente, que determina que propuestas el agente realizará. Usualmente la estrategia que una agente utiliza es privada.
4. Una regla que determina cuándo un trato está cerrado, y cuál es dicho trato.

[25] describe 2 enfoques con los que se puede describir una negociación.

Task Oriented Domain tupla $< T, Ag, c >$ donde:
T es el conjunto (finito) de las posibles tareas
$Ag = \{1..n\}$ es el conjunto (finito) de los agentes participantes de la negociación.

[4] Estrategia que permite obtener el mejor resultado
[5] Comunicación previa a la negociación

$c = \gamma(T) \rightarrow \mathbb{R}^+$ es una función que define el costo de ejecutar cada subconjunto de tareas. c es una función monotónica, es decir, añadir tareas no disminuye $c(T)$, y el costo de no realizar ninguna tarea es 0.

Un encuentro TOD ocurre cuando a los agentes en el conjunto Ag se les asigna una tarea para realizar del conjunto de tareas T. Cuando sucede un encuentro, los agentes pueden reasignar las tareas entre ellos, de esta manera, los agentes pueden, en principio, realizar mejor su tarea que si simplemente realizaran las tareas por ellos mismos. Formalmente, un encuentro en un TOD $< T, Ag, c >$ se define como una colección de tareas $< T_1, .., T_n >$ donde para cada i se tiene que $i \epsilon Ag$, T_i está incluido en T y T_k es el conjunto de tares definidas para el agente k, donde se debe reasignar las tareas $T_i \cup T_j$ entre los agentes i y j; $k, i, j \epsilon \{1..n\}$.

Para el caso básico de 2 agentes, un trato es una tupla $< D_1, D_2 >$ donde $D_1 \cup D_2 = T_1 \cup T_2$ y el agente i está comprometido a realizar las tareas D_i. El costo para el agente i del trato $\delta =< D_1, D_2 >$ es $c(D_i)$ denotado como $cost_i(\delta)$. La utilidad del trato δ para el agente i es la diferencia entre el costo de que i realice T_i y que i realice D_i.

$$utilidad_i(\delta) = c(T_i) - cost_i(\delta) \tag{30}$$

Si $utilidad_i(\delta) < 0$ entonces, el nuevo trato es peor que el trato por defecto.

Un trato δ_1 se dice que domina a un trato δ_2 sii:

1) δ_1 es al menos tan bueno para todo agente en Ag como δ_2

2) El trato δ_1 es mejor para algún agente en Ag que δ_2

Un trato que no es dominado por ningún otro trato es llamado Pareto óptimo. δ_1 domina levemente (*weakly*) a δ_2 si ocurre 1. δ es individualmente racional si domina levemente el trato por defecto. El conjunto de negociaciones consiste en el conjunto de tratos que son individualmente racionales y pareto óptimos.

Worth Oriented Domains En contraste a un TOD, el objetivo de un agente en un WOD es llevar el ambiente a un estado más favorable. Los planes ejecutados por los agentes transforman el ambiente de un estado a otro. Llegar a un acuerdo consiste en que los agentes negocien no sobre la distribución de las tareas, sino sobre la colección de planes que deben ejecutar. El objetivo de los agentes es entonces, llegar a un acuerdo en la realización de un plan que lleve al ambiente al estado con el mayor valor para todos.

Un WOD es una tupla $< E, Ag, J, c >$ donde:

E es el conjunto de posibles estados del ambiente.

$Ag = \{1..n\}$ es el conjunto de posibles agentes.

J es el conjunto de posibles planes.

$c: J \times Ag \rightarrow \mathbb{R}$ es una función de costo que asigna a cada plan $j \epsilon J$ y a cada $i \epsilon Ag$ un número real que representa el costo $c(j, i)$ de que ejecute i el plan j.

Un encuentro en un WOD $< E, Ag, J, c >$ es una tupla $< e, W >$ donde:

$e \epsilon E$ es el estado inicial del ambiente.

$W : E \times Ag \rightarrow \mathbb{R}$ es una función de costo que asigna a cada estado $e \epsilon E$, y a cada agente $i \epsilon Ag$ un valor real $W(e, i)$ que representa la ganancia para i de que el ambiente se encuentre en el estado e.

Los agentes que interactúan en un WOD están negociando sobre el estado en el que querrían que estuviese el ambiente, y el cómo alcanzar dicho objetivo.

8.2.3. Cooperative Distributed Problem Solving (CDPS)

CDPS estudia como un grupo de agentes puede trabajar en conjunto para resolver problemas que están más allá de las capacidades individuales de cada uno. Cada uno de dichos agentes es capaz de realizar tareas individuales, pero los problemas que deben enfrentar, no pueden ser solucionados sin cooperación. Dicha cooperación es necesaria, dado que ningún agente cuenta con la experiencia, recursos, y/o información suficiente para resolver el problema individualmente.

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 taras mediante una subasta o una negociación.

Result Sharing Involucra agentes compartiendo información relevante de sus subproblemas. Esta información puede ser compartida proactivamente (un agente envía a otro agente información porque cree que al otro agente le interesa), o reactivamente (un agente envía información a otro agente en respuesta a un pedido que fue previamente enviado).

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. [28] menciona que, a partir del uso de 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 vista local para obtener una mejor visión global del problema

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 más rápidamente.

8.2.4. Manejo de inconsistencias

Uno de los mayores problemas que aparecen en la actividad cooperativa es la inconsistencia entre distintos agentes. Esto sucede por ser sistemas autónomos, ruido en sensores, etc. En cuanto a cómo manejar las inconsistencias, se sugieren varios enfoques:

No permitir que ocurran. Diseñar los protocolos y sistemas para no permitir inconsistencias.

Resolver inconsistencias mediante negociación. Negociar los distintos puntos de vista, para obtener el mejor resultado.

Construir sistemas que no se degraden frente a la presencia de inconsistencias. Desarrollar sistemas lo suficientemente robustos para poder ejecutarse con inconsistencias.

9. BIOLOID - Premium kit

La empresa ROBOTIS, cuenta en su línea de robots con los modelos OLLO y BIOLOID. Dentro de la línea de BIOLOID, se encuentran los kits:

Beginner: Kit para principiantes, el manual presenta 14 tipos de robots para construir y programar.

Comprehensive: Incluye el contenido del kit *Beginner*, aumentando la cantidad de piezas para poder formar 26 tipos de robots según el manual, las cuales van desde una araña hasta una estructura humanoide.

Expert: Mejora del kit *Comprehensive*, contiene 2 controladores (CM-5) así como un cámara inalámbrica para usar como sensor.

Premium: Es la última mejora del kit *Comprehensive*. En esta versión, se utiliza un controlador más actual (CM-510).

Para el desarrollo del proyecto de grado, se cuenta con kits *Premium* de la línea BIOLOID. El mismo está compuesto por:

- CM-510 (Controlador): 1 pieza
  - CPU: ATMEGA 2561
  - Internal I/O Device: 6 botones, Mic, Sensor de temperatura, y sensor de voltaje incluido
  - External I/O Device: 6 puertos para sensores analógicos, 5 puertos TTL para conectores de la serie AX de Dynamixel
  - Mode Display & AUX LED
- Dynamixel servos (motores AX-12+): 18 piezas[29].
- 2-axis Gyro Sensor (sensor giroscópico de 2 ejes): 1 pieza.
- DMS Sensor: 1 pieza - Sensor IR: 2 piezas.
- RC-100(Control remoto): 1 pieza.
- Skin Set de humanoide.
- Batería Li-Po (11.1V, 1000mA/PCM) : 1 pieza.
- Cargador de baterías.
- USB2Dynamixel, adaptador de USB a TTL, R232 y R 213123.
- Quick Start Book en el que se especifican 3 tipos de humanoides.
- Destornillador, grapas de cables.

- RoboPlus CD

Según ROBOTIS, el kit *Premium* se destaca por:

- Excelente capacidad de caminar para los modelos sugeridos de humanoides en el manual (modelos A, B y C), ajustando automáticamente su postura mientras caminan.

  Modelo A: 18 servos, 1.7kg, 39.7cm de alto. Pasos estables y omni-direccionales.

  Modelo B: 16 servos, 1.6kg, 39.7cm de alto. Capacidad de cambiar de dirección mientras camina.

  Modelo C: 16 servos, 1.6kg, 38.6cm de alto. Capacidad de ir de lado mientras camina.

- Varios sensores incluyendo giroscopios, infrarrojos y sensores de distancia.
- Control remoto RC-100 incluido.
- Sencilla programación, utilizando el software RoboPlus, muy parecida al lenguaje C.
- *Humanoid skin* semitransparente, que permite personalizar el robot.
- Comunicación de paquetes con la topología de conexión en cadena.
- Construcción de robots a través de mecanismos fáciles de expansión.

A la izquierda, versión C. Al centro, versión A. A la derecha versión B[30].

Figura 10: Modelos de BIOLOID

El kit Premium cuenta con las piezas necesarias para armar un humanoide que cumpla con las restricciones de la liga *humanoid kid-size* de Robocup. Sin embargo, el kit no provee de un hardware dedicado a visión, como por ejemplo una cámara.

10. Hamid's Vision Module (HaViMo)

Como opción de hardware de visión para la obtención de imágenes se presenta el módulo de visión HaViMo. El mismo es compatible con los modelos de la línea BIOLOID de ROBOTIS y puede realizar el proceso de segmentación, mediante agrupación en *blobs,* por *hardware.*

Actualmente, el mismo se encuentra en su versión 2.0; sin embargo, para el proyecto de grado se cuenta con la versión 1.5 del módulo, la cual cuenta con las siguientes características[3]:

- Regiones de color: Hasta 15 regiones (*blobs*) pueden ser detectadas en cada cuadro.
- Detecta regiones (*blob*) separadas con el mismo color o diferente.
- Calcula color, centro, cantidad de píxeles y el cuadrado delimitador de cada región.
- Procesamiento de imágenes a 8 frames por segundo, con una resolución de$160px \times 120px$ en formato YCrCb.
- Pueden ser utilizados hasta 255 colores distintos en la definición de la *lookup table* incorporada.
- La *lookup table* puede ser accedida a través del bus serial.
- Filtro de ruido incorporado y ajustable, permite la detección de objetos grandes y pequeños de acuerdo al color, incluso en imágenes con gran cantidad de ruido.
- Umbral ajustable remueve automáticamente las regiones pequeñas indeseadas.
- Soporta 1Mbps de baud rate[6].
- Utiliza la misma conexión TTL de BIOLOID, facilitando la conexión del dispositivo a un *bus* de comunicación.
- Acceso directo a todos los ajustes de la cámara permitiendo el uso del dispositivo, ya sea con la exposición automática/balance de blancos o la configuración manual, lo que es ideal para situaciones de luz constante, como en un partido de RoboCup.
- Obtención de la imagen para depuración, o calibración de la *lookup table*, a través del programa de calibración provisto.

Las principales diferencias de la versión 2.0 frente a la versión 1.5 es que la primera cuenta con más características:

[6]Un Baudio corresponde al número de unidades de señal por segundo. Un baudio puede contener varios bits, por lo que la tasa de baudios siempre es menor o igual a la tasa de bits (`http://es.wikipedia.org/wiki/Baudio`).

- Conectividad con roboBuilder[31] y otras plataformas via full-duplex
- Soporte para las aplicaciones de RoboPlus
- Nuevo algoritmo de visión "Griding"
- Mayor frame rate

10.1. Funcionamiento del dispositivo

El módulo de visión opera en 2 modos llamados “Calibración” e “Implementación”. En la fase de calibración se ajusta la configuración de la cámara, definiendo los colores que se agruparan en cada tipo de región en en la *lookup table*. Todos los ajustes serán almacenados en la memoria FLASH/EEPROM del módulo, lo que significa que ante las mismas condiciones de luz, no habría necesidad de volver a calibrar el dispositivo. A partir de la versión 1.5, el módulo puede ser calibrado sin necesidad de sobrescribir el programa controlador del CM-510 con el programa de calibración, haciendo uso del adaptador USB2Dynamixel.

Luego de ser calibrado, el módulo se encuentra apto para ser usado en la etapa de implementación, donde es conectado al controlador principal (CM-5 o CM-510), pudiendo recibir comandos para adquirir y procesar imágenes y devolver el resultado.

Cuenta con la ventaja de que, al utilizar el mismo protocolo que la línea BIOLOID utiliza para comunicar el controlador con los actuadores, es posible reutilizar las librerías orientadas a la comunicación desarrolladas para BIOLOID.

10.2. Calibración en CM-510

La calibración del módulo HaViMo se realiza mediante el programa de calibración proporcionado por el proveedor.

HaViMo en su versión 1.5 fue desarrollado para el controlador CM-5, por lo que para ser calibrado utilizando el controlador CM-510 se deben realizar los siguientes pasos de manera de proveer alimentación a la cámara:

1. Conectar la cámara al controlador CM-510 a través del conector TTL.
2. Conectar el controlador CM-510 al adaptador USB2dynamixel a través del conectror TTL.
3. Cambiar el adaptador USB2dynamixel para utilizar la salida TTL.
4. Conectar el adaptador al PC.

Una vez conectada la cámara al PC, se realiza la calibración utilizando el programa de calibración USB.

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] ROBOTIS. http://www.robotis.com/xe/.

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

[4] 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.

[5] 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.

[6] Kinect for Windows. http://www.microsoft.com/en-us/kinectforwindows/.

[7] Mars Exploration Rovers. http://marsrover.nasa.gov/home/index.html.

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

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

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

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

[12] 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.

[13] Steve Nicklin. Object recogntition in robotic soccer, November 2005.

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

[15] 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.

[16] Christian Wöhler. *3D Computer Vision Efficient Methods and Applications*. Springer Science+Business Media, London, New York, 1 edition, 2009.

[17] 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.

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

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

[20] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. *Monte Carlo Localization: Efficient Position Estimation for Mobile Robots*, pages 343–349. Number Handschin 1970. JOHN WILEY & SONS LTD, 1999.

[21] Serrana Casella Santiago Margini Gonzalo Tejera, Facundo Benavides. Teórico de ia y robótica. 2009.

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

[23] CMUnited. http://www.cs.cmu.edu/ robosoccer/main/.

[24] CSFreiburg. http://www.cs-freiburg.de/.

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

[26] J. S. Sichman. A social reasoning mechanism based in dependence networks. *Proceedings of the 11th European Conference on Artificial Intelligence, Amsterdam*, pages 188–192, 1994.

[27] T. Sandholm. Distributed rational decision making. *Multiagent Systems*, pages 201–258, 1999.

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

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

[30] ROBOTIS. Bioloid premium kit quick start.

[31] RoboBuilder. http://www.robobuilder.net/.