CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS

AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco

Departamento de Computación

Coordinación de pases de pelota entre robots móviles sobre terrenos exteriores

TESIS

Que presenta

Juan José Mondragón Sandoval

Para obtener el Grado de

Maestro en ciencias

de la

Computación

Director de Tesis:

Dr. José Matías Alvarado Mentado

México, D.F. Febrero 2013

RESUMEN

En este proyecto de tesis se pretende lograr una coordinación entre robots móviles autónomos en terrenos exteriores. La coordinación de robots autónomos actualmente se ha desarrollado de manera experimental en interiores pero no se ha explorado de la misma manera en exteriores por las problemáticas que estos terrenos presentan como son: la rugosidad y la pendiente del terreno que en ocasiones son obstáculos para robots de tamaño reducido ya que producen derrapes o atascos, el alto brillo que se genera por la luz del sol y que provoca una reduccion en la deteccion de los objetos en las imágenes usadas para el procesamiento visual. Al finalizar el desarrollo de esta tesis se espera tener una coordinación de robots que se manden pases con una pelota desde un punto inicial a un punto final en terrenos exteriores.

ABSTRACT

In this thesis project is intended to achieve coordination between autonomous mobile robots in outdoor terrain. Coordination of autonomous movile robots currently developed experimentaly on indoors, has not been explored in the same way in outdoors for the problems that occur in these terrains such as: the roughness and slope are sometimes obstacles for reduced size robots as drift or jams occur, the brightness that is generated by sunlight and causes a reduction in the detection of objects in the images used for visual processing. Upon completion of this thesis is expected to have a coordination of robots that sent ball passes from a start point to an end point in outdoors.

AGRADECIMIENTOS

Agradezco a los profesores que me dieron clases ya que sin ellos no hubiera podido adquirir los conocimientos que hoy me permiten presentar esta tesis.

Agradezco a todos mis compañeros de generación, ya que en menor o mayor grado compartí con ellos los retos y las satisfacciones que representa el estudiar aquí y de todos y cada uno aprendí algo, gracias.

Agradezco al CONACYT por el apoyo economico otorgado en la duración de la maestría de la beca CVU 243131.

Agradezco al Cinvestav, este centro de calidad internacional por haberme brindado la oportunidad y los recursos para poder realizar mis estudios de maestría.

Agradezco a mi asesor, el Dr. Matías Alvarado Mentado, por el apoyo y la confianza que me brindó para la culminación de este trabajo.

Agradezco a los doctores miembros del jurado, Dr. Amilcar Meneses Viveros y al Dr. Farid García Lamont, por su tiempo, observaciones, consejos y aportaciones que permitieron mejorar este documento.

Agradezco a Sofía Reza, Felipa Rosas y Erika por todo su apoyo en cada momento, y su excelente actitud de ayuda hacia nosotros los estudiantes.

Agradezco a mis amigos y familiares que siempre estuvieron ahí para brindarme su apoyo incondicional y me motivaron para culminar este trabajo.

DEDICATORIA

Quiero dedicar este trabajo de Tesis a mis hermanos Diana, Norma y Lalo, ya que siempre me han enseñado a ser una mejor persona con su ejemplo y me han alentado a seguir adelante, los quiero mucho.

A mis padres, Ma. Del Consuelo Sandoval G. y Juan José Mondragón O., por que siempre me han enseñado que todo en esta vida es posible. Han mostrado una fortaleza enorme en los momentos mas dificiles poniendose como ejemplo de valor y coraje. Me han brindado su amor y enseñanzas las cuales me han permitido llegar ser la persona que soy y tambien les agradezco infinitamente su enorme apoyo y paciencia, gracias padres, los amo.

Contenido

1. Introducción

Resumen. En las diferentes ligas de la robocup, sobre superficies lisas, mandar un pase dado ha sido resuelto eficientemente; sin embargo, cuando la textura sobre la cual viaja la pelota no es lisa el problema sigue abierto y es el problema a considerar en esta tesis. Junto con la descripción del problema, en este capítulo se describen los objetivos a lograr, así como la justificación de su relevancia en el área de coordinación de robots móviles autónomos.

1.1 Antecedentes y motivación para el proyecto

La principal motivación para desarrollar robots autónomos es que estos se encarguen de tareas peligrosas que para los seres humanos representan un riesgo físico e incluso pueden ser mortales como por ejemplo: la limpieza de un derrame nuclear, el rescate de personas en terrenos peligrosos, la exploración en otros planetas, etc.

Por otra parte, es importante su desarrollo para que realicen las tareas monótonas que para los seres humanos son muy aburridas, como por ejemplo las líneas de producción de diversas fábricas.

Si bien un robot experto es bueno para realizar una tarea específica, este no es capaz de realizar una tarea de mayor complejidad por sí solo, es por esto que el desarrollo de sistemas multirobot ha tenido un importante crecimiento. Dentro de la coordinación de robots uno de los principales problemas que nos podemos encontrar es la asignación de recursos para que la tarea se lleve a cabo de manera eficiente y confiable. Desarrollar la coordinación entre robots es importante ya que uno de los principales objetivos a futuro es que esta coordinación sea tan natural que los robots sean capaces de trabajar de manera coordinada con los humanos.

Una iniciativa muy importante es la RoboCup ("RoboCup," 2012), se trata de un proyecto internacional que trata de motivar el desarrollo robots autónomos y la investigación de la inteligencia artificial con el objetivo principal de que en el 2050 se tenga un equipo de robots humanoides autónomos que sean capaces de vencer a la selección ganadora de la copa mundial de futbol de la FIFA.

*[i]

Figura 1.- Actualidad de la RoboCup

En la actualidad este tipo de coordinación solo se realiza en interiores, de ahí la importancia de comenzar con la coordinación en exteriores para atender las problemáticas que se presentan en este tipo de terrenos.


*[ii]

Figura 2.- Un desafío en exteriores

1.2 Definición del problema

Como se mencionó anteriormente, uno de los principales problemas en los sistemas multirobot es el de la coordinación, actualmente existen numerosos trabajos que hablan sobre la coordinación de robots pero en ambientes interiores, el objetivo de este trabajo es el de lograr una coordinación en terrenos exteriores atacando algunas problemáticas propias de estos ambientes. Para comprobar que se realiza una coordinación de manera correcta se tienen que realizar una serie de experimentos. La evaluación de los métodos de

coordinación está dada por el número de veces que se realiza una tarea de forma satisfactoria. Existen algunas tareas "clásicas" para la evaluación de los métodos de coordinación aunque están pensadas para interiores como son: recolección de objetos, seguimiento de un objeto en movimiento, traslado de una caja de un punto inicial a un punto final, etc. Para probar que la coordinación propuesta funcione se aplicara para resolver un problema en particular, a continuación se plantea este problema tomando en cuenta que se llevará a cabo en exteriores. Cabe destacar algunos puntos clave para lograr que la tarea se realice de manera correcta:

- Que el robot sea capaz de identificar una pelota en exteriores
- Que el robot sea capaz de seguir a esta pelota
- Que el robot pueda predecir a donde llegara la pelota que el golpee
- Que se pueda coordinar con otro u otros robots

Se pretende coordinar un par de robots R_1 y R_2 para que desde sus posiciones iniciales $P_1(x_1, y_1)$ y $P_2(x_2, y_2)$ respectivamente lleven a una pelota que se encuentra en un punto inicial $P_o(x_o, y_o)$ hasta un punto final $P_f(x_f, y_f)$ como se muestra en la Figura 3.

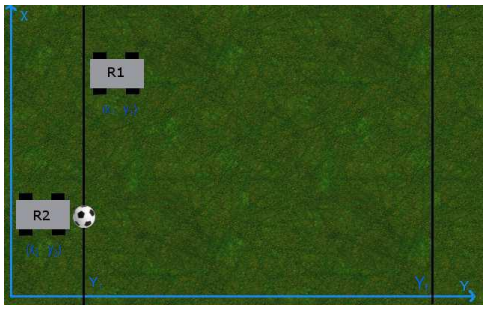

Figura 3.- Posiciones iniciales de los robots

Este recorrido lo realizarán mandándose pases con una pelota en ambientes exteriores. La textura de la superficie será tomada en cuenta para calcular la fuerza con la que se golpeará la pelota y así poder realizar los pases de manera precisa. La coordinación de este sistema multirobot se llevara a cabo mediante una modificación de la arquitectura MURDOCH descrita en (Gerkey & Mataric, 2002) por ser escalable, tolerante a fallos y que al estar basada en el método de subastas, la cual es una técnica de negociación, se conoce que es funcional para este tipo de problema (Dias, Zlot, Kalra, & Stentz, 2006). Esta arquitectura coordina los robots de manera distribuida mediante una asignación de tareas

descentralizada por medio de subastas y aunque solo se trabajará con 2 robots, se espera una solución escalable por las características antes mencionadas. La modificación de esta arquitectura contempla añadir una etapa de retroalimentación de la posición de la pelota,0la cuál servirá para que los robots aprendan a golpear la pelota de manera precisa dependiendo el terreno donde se encuentren.

Se toman en cuenta dos estados del robot:

1. Robot con posesión de la pelota.- En este estado el robot tendrá que mandar un pase de manera precisa a otro robot que se encuentre disponible y en condiciones de recibir el pase, de tal forma que se reduzca la distancia a la posición objetivo de la pelota. Como el objetivo principal es avanzar en el eje de las y debemos de enviar el pase en una diagonal cuya componente en y sea mayor o igual que la componente en x, esto es, el pase se realizará entre $45°$ y $135°$ desde la perspectiva del robot que realizara el pase orientado de manera perpendicular hacia la línea de meta.

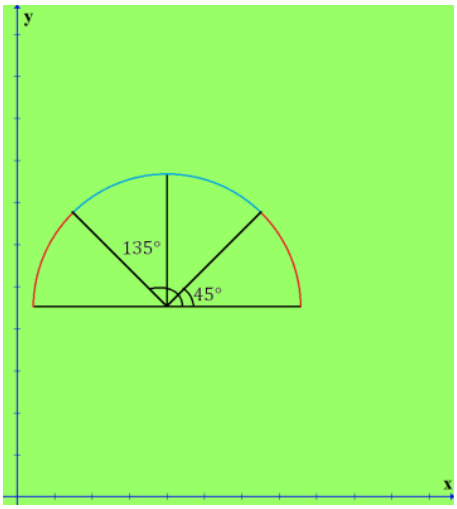

Figura 4 .- Angulo preferido para mandar el pase

Para saber que robot puede recibir el pase se requiere de un procedimiento que evalué a todos los robots disponibles y capaces, para así, conocer al que mejor pueda llevar a cabo esta tarea. Después de decidir cuál robot es el más apto para realizar esta tarea se le notificará y calculará el ángulo que coincida con la trayectoria de este robot como se muestra en la Figura 5. La textura del terreno se tomará en cuenta para realizar el cálculo de a que distancia se puede hacer llegar la pelota para así realizar un pase preciso al mandar el pase.

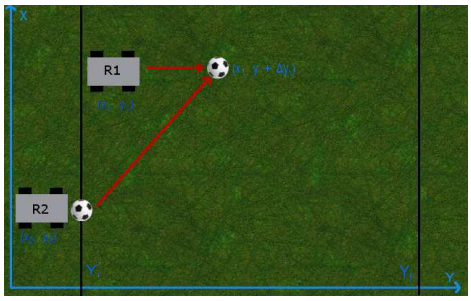

Figura 5.- Cálculo de la posición final

2. Robot sin posesión de la pelota.- Este robot esperará por una nueva tarea y en ese momento intentara ganarla evaluándose a sí mismo. Una métrica posible que tendrá para evaluarse será recibir la posición inicial de la pelota $P_i(x_{pi}, y_{pi})$ y a partir de su posición $P_r(x_r, y_r)$ calcular la distancia euclidiana D_e entre esta coordenada y su posición (Figura 6)

$$D_e = \sqrt{(x_r - x_p)^2 + (y_r - y_p)^2}$$

Después de evaluarse contestará con la distancia calculada D_e y esperará por la notificación de la asignación de la tarea. Si se le notifica que gano la tarea se pondrá a realizarla. En caso de no ser seleccionado para la tarea se regresara a la espera de una notificación de una tarea nueva. Al momento de realizar la tarea se dirigirá a la coordenada destino calculada como su posición final y avanzará realizando un ajuste de su trayectoria para intersectar a la pelota.

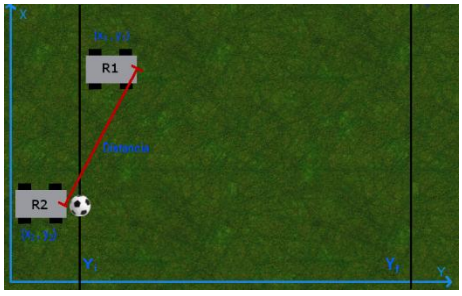

Figura 6.- Cálculo de la distancia Euclidiana

Cualquiera que sea el estado del robot el sistema terminará en caso de que el objetivo sea alcanzado tal como se muestra en la Figura 7.

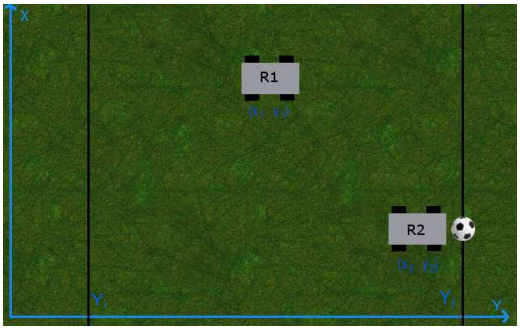

Figura 7.- Objetivo alcanzado

La principal dificultad para realizar este trabajo se encuentra en el hecho de trabajar en terrenos exteriores ya que, como se había mencionado anteriormente, se presentan los problemas relacionados a la rugosidad y pendiente del terreno que en ocasiones son obstáculos para robots de tamaño reducido ya que estos producen derrapes o atascos, además de presentar un problema para el desplazamiento de la pelota, también los problemas relacionados con la visión en exteriores como el alto brillo que se genera por la luz del sol y las sombras en las imágenes usadas para la visión por computadora.

Es importante la solución de este problema pues daría inicio a la coordinación de robots en ambientes más reales que los utilizados actualmente (ambientes interiores), en particular a la interacción de robots que se pasan una pelota en un ambiente más natural como el césped o la tierra. Este trabajo es importante porque será el comienzo de un aprendizaje de la fuerza con la cual se debe golpear una pelota en distintos terrenos en el contexto del futbol para acercarnos cada vez más a la solución del objetivo planteado por la RoboCup. El trabajo se realizará con un par de robots LEGO.

1.3 Objetivos de la tesis

1.3.1 General

Lograr la coordinación de robots en terrenos exteriores para que realicen la tarea de llevar mediante pases una pelota de un punto inicial a un punto final en terrenos exteriores y con aprendizaje para conseguir una mayor precisión en los pases dependiendo del terreno.

1.3.2 Particulares

- Implementar un algoritmo de coordinación de robots cuyo objetivo es llegar a una posición final con una pelota en posesión de uno de ellos por medio de pases controlados distribuyendo la tarea por medio de subastas.

- Implementar un algoritmo de seguimiento de una pelota por medio de un algoritmo de detección de color con una cámara para poder estimar su posición y así ajustar la trayectoria del robot que recibe el pase y poder retroalimentar con esta información al robot que mandó el pase para el aprendizaje.

- Implementar en los algoritmos el cálculo de la odometría, basada en la velocidad y dirección del robot, para conocer en todo momento la posición de los robots.

- Realizar pruebas para el cálculo de la constante de fricción para cada tipo de terreno exterior haciendo uso del aprendizaje autónomo.

1.4 Organización de la tesis

En el Capítulo 2 se da una breve descripción del estado del arte en cuanto a la coordinación de robots, en el Capítulo 3 se describe como se implementó la arquitectura seleccionada y su adaptación para ambientes exteriores, en el Capítulo 4 se describen las pruebas realizadas con los robots y en el Capítulo 5 se discuten los resultados obtenidos. Por último en la sección de apéndices se muestran los códigos fuente de los programas realizados.

2. Estado del arte

Introducción. En este capítulo se da el estado del arte sobre la coordinación de robots y se explica por qué se adopta un método basado en subastas para resolverlo. Asimismo, se introduce la visión en exteriores y algunos métodos utilizados para la predicción de color en exteriores, y finalmente se habla de la polarización de la luz y como esta ha sido usada para diversas aplicaciones dentro de la visión por computadora.

2.1 Arquitecturas de coordinación

La clave para lograr la coordinación de grupos de robots es la cooperación. Los robots deben de trabajar de manera conjunta para maximizar su rendimiento global. La cooperación emergente (Fukuda, Nakagawa, Kawauchi, & Buss, 1989) es un modelo muy popular en el cual los robots no trabajan juntos explícitamente, sino que su comportamiento de cooperación a nivel grupal emerge de entre las interacciones entre ellos y el medio ambiente.

Estos sistemas emergentes son en extremo efectivos y seguramente son también la solución más elegante y simple a un problema. Sin embargo esta solución es específica a un problema y si queremos robots de uso general, estos deben ser capaces de resolver una gran variedad de problemas. A continuación se presentan algunas arquitecturas para la cooperación en sistemas multirobot.

2.1.1 Open Agent Architecture

Esta arquitectura se enfoca en proporcionar un medio ambiente de lo más general en el que una población diversa de agentes, como las interfaces de usuario y sistemas de administración de base de datos heredadas, pueden interactuar y coordinarse. Puesto que no se enfocan en la asignación de recursos en robots físicos se genera una sobrecarga por las especificaciones de la ontología para permitir que el sistema sea tan general. Hay un

componente para la asignación de tareas entre sus agentes, a pesar de que las tareas son meramente informativas, en lugar de físicas. Esta Arquitectura logra la asignación de tareas a través de un intermediario (denominado facilitador), que sirve para asignar las tareas nuevas con los agentes que previamente han mostrado tener las capacidades relevantes a la tarea(Martin, Cheyer, & Moran, 1999).

2.1.2 ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation

Una arquitectura para la coordinación de robots, basada en comportamiento, que deben cooperar para lograr una tarea. Cada robot conoce la tarea entera que se debe completar, los robots usan comportamientos de motivación para competir por los diversos componentes de tarea en el tiempo. Cada robot registra tanto su condición física y progreso como la de sus compañeros de equipo, incorporando esta información de rendimiento en las medidas locales de impaciencia y complacencia. Si un robot se vuelve lo suficientemente impaciente, podrá ocupar una tarea de un robot lo suficientemente complaciente, proporcionando tolerancia a fallos(Parker, 1998).

2.1.3 BLE: Broadcast of Local Eligibility for Multitarget Observation

Presenta un enfoque similar al de ALLIANCE pero más minimalista sobre la asignación de tareas de los robots, un derivado moderno de la arquitectura de subsunción que ha sido demostrado en la tarea de seguimiento de objetivo. En el marco de BLE, los robots no tienen ningún compromiso con su tarea; se comunican continuamente para decidir quién es el más apto para cada trabajo. Puesto que los robots pueden cambiar tareas en cualquier momento, el sistema tiene una buena respuesta dinámica a los cambios ambientales, a expensas de la comunicación añadida necesaria para las transmisiones continuas de aptitud. Esta arquitectura asume un control de los robots en el equipo basado en comportamiento. Implementa la asignación de las tareas a través de la inhibición de la conducta, cada robot tiene un deseo para ejecutar cada tarea y los robots suprimen sus actividades en el nivel de comportamiento directamente (Werge & Mataric, 2000).

2.2 Arquitectura MURDOCH

En esta arquitectura se plantea una estrategia basada en el modelo de cooperación intencional. En este modelo los robots cooperan de manera explícita y con un propósito, a

menudo a través de una comunicación relacionada a una tarea, por este tipo de cooperación esta arquitectura se encuentra dentro de la clasificación hecha en (Farinelli, Locchi, & Nardi, 2004) como una cooperación donde los robots son conscientes de que están cooperando y la coordinación es muy estrecha además de que la organización es distribuida. Comparándola con la cooperación emergente, la cooperación intencional se ajusta mejor al objetivo a largo plazo en el área de cooperación entre robots, el cuál es que la cooperación entre robots y seres humanos se realice de manera natural.

Para decidir que robot realiza una tarea se emplea una subasta, la cual basa su métrica de evaluación en la aptitud y con esto se muestra que la asignación de tareas basada en negociación explicita, como lo es la subasta, es un método efectivo y tolerante a fallos para un sistema multirobot.

Esta arquitectura tiene los siguientes componentes:

- Comunicación anónima.- Se utiliza una comunicación de tipo broadcast por su mayor economía ya que resulta más costoso mandar mensajes dirigidos a una gran cantidad de robots y como los robots entran y salen del rango de comunicación, la comunicación anónima aporta la tolerancia a fallos.

- Estructura de tareas jerárquicas.- Se establece un modelo de tareas jerárquico donde una tarea compleja puede ser compuesta por tareas más sencillas. Esta descomposición jerárquica debe ser desarrollada por el programador dependiendo de la tarea que se quiera realizar.

- Subastas.- Se seleccionó el método de subasta para asignación de tareas por ser escalable y eficiente en tiempo de cómputo y comunicación además de que resulta apropiado para este problema específico.

Las principales características de esta arquitectura son las siguientes:

Publicación/Suscripción de mensajes.- Los mensajes se en rutan por contenido y no por destino. La manera de designar el nombre del contenido es creando una semántica que identifica las necesidades de la tarea a realizar.

Protocolo de Subasta.- Básicamente se resume en los siguientes pasos:

1. Anuncio de una nueva tarea.

El usuario, un evento, una alarma, un calendarizado o una tarea en curso activan el ingreso de una nueva tarea en el sistema e indican a un robot que la debe subastar, este robot se convierte en el agente subastador y su tarea es la de reconocer la tarea y la métrica de evaluación que se utilizara en la subasta. Después envía un mensaje de tipo broadcast con el nombre de la tarea y la métrica a ser evaluada y espera por las respuestas de los robots un tiempo determinado.

2. Evaluación de la métrica.

Cada robot se evalúa para esa tarea. En un ambiente de cooperación suena raro utilizar un mecanismo competitivo para la asignación de tareas pero como nosotros programamos a los robots sabemos que estos son honestos por lo que se evaluarán de manera honesta y por lo tanto los robots cooperarán entre ellos para lograr el mejor resultado.

3. Envió de ofertas.

Después de que cada robot haya evaluado su capacidad para realizar la tarea enviara este resultado como su oferta por la tarea subastada.

4. Cierre de la subasta.

Después de un tiempo específico el agente subastador procesa las ofertas para determinar al ganador y notifica el resultado con un mensaje de fin de subasta. Al robot ganador se le asigna un contrato por un tiempo limitado para llevar a cabo la tarea mientras que los demás robots regresan a la espera de una nueva subasta.

5. Monitoreo y renovación de contrato.

El agente subastador se encarga de monitorear el progreso de la tarea y si le parece que el progreso es aceptable renueva el contrato con el robot mandándole un mensaje de renovación al cual el robot que se encuentra realizando la tarea contesta con un mensaje de recibido, en caso de que el progreso no sea el esperado o el mensaje de recibido no llegue al agente subastador, este se encargara de subastar la tarea para asignarla a otro robot que la pueda realizar correctamente.

Tolerancia a Fallos.- Una parte importante de la tolerancia a fallos de MURDOCH, es la asignación temporal de las tareas, esto permite detectar fallos en una unidad que no responda luego de que se le reasigna la tarea o en caso de que su progreso sea pobre.

Esta arquitectura se comporta como un calendarizado de tareas glotón e instantáneo y por lo tanto sufre de las desventajas de los algoritmos glotones. No siempre se realiza la asignación de tareas más óptima porque la llegada de tareas es estocástica.

La métrica de evaluación de la subasta debe regresar un valor escalar que represente la aptitud de un robot para realizar una tarea.

A continuación se muestra una tabla que resume las características de las arquitecturas anteriores.

Tabla 1.- Comparativa de las arquitecturas

Arquitecturas	Ventajas	Desventajas
Open Agent Architecture	Solución General	No para robots físicos Tareas informativas Asignación de tareas centralizada
BLE: Broadcast of Local Eligibility for Multitarget Observation.	Para robots físicos No hay compromiso con la tarea Buena respuesta ambiente dinámico	Necesita gran ancho de banda Comportamiento Motivacional
ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation	Robots Heterogéneos Tolerante a fallos	Comportamiento Motivacional
MURDOCH	Para robots físicos Asignación de tareas descentralizada Tolerante a fallos Escalable	No divide las tareas

Después de analizar las distintas arquitecturas podemos concluir que la más adecuada para resolver nuestro problema es la arquitectura MURDOCH (Gerkey & Mataric, 2002) y que además es fácil de modificar para implementar el aprendizaje necesario para ajustar la velocidad de golpeo de la pelota. De manera general después de describir el estado del arte respecto a la coordinación de robots podemos concluir que muy pocas de estas arquitecturas han sido pensadas para implementación física y las que se han implementado físicamente, como la seleccionada, han sido puestas a prueba en terrenos interiores. Es por

esto que la propuesta aquí descrita (Coordinación en terrenos exteriores) se considera un trabajo con el potencial de desarrollar algo novedoso y útil para, en un futuro, poder cumplir con el objetivo final de la RoboCup.

2.3 Visión en exteriores

En los modelos de predicción de color, existe una manera de encontrar superficies parecidas bajo una iluminación variante y desconocida sin tener información sobre la función de reflexión de la superficie llamada constancia de color (N. Sahragard, A. R. B. Ramli, 2009). Los algoritmos que proporcionan esta constancia de color pueden separarse en seis grupos (N. Sahragard & Ramli, 2009):

El primer grupo es de aquellos que requieren de un número de fuentes de luz, pero la necesidad de identificar la misma superficie en dos partes distintas de la imagen que está siendo sujeta a diferentes fuentes de luz es un problema, pero no para imágenes restringidas. D'Zmura e Iverson (Dzmura & Iverson, 1993) y Finlayson et al (Finlayson, Funt, & Barnard, 1995) hicieron algunos trabajos con este enfoque y una variación de la misma fue realizada por Finlayson (Finlayson, Hordley, & HubeL, 2001) la cual mostró resultados favorables al correlacionar los colores de la imagen con los colores que se pueden producir en cada conjunto de fuentes de luz posible.

En el segundo grupo se toma la gama de colores de la imagen. El algoritmo de Forsyth (Forsyth, 1990) mapea los colores de manera que el número de mapeos restringe el conjunto de fuentes de iluminación posibles. En este sentido, Finlayson et al (Finlayson et al., 1995) aplica una transferencia espectral de nitidez a los datos sensoriales con el fin de relajar las restricciones de la gama. Aunque el algoritmo de Forsyth fue un avance importante en la constancia de color, está limitado a la superficie mate de Mondrian o a la iluminación controlada. Ohta y Hayashi (Ohta & Hayashi, 1994) también asumieron una gama conocida de fuentes de luz para interiores, siguiendo el modelo CIE. Su método sólo se aplica a una limitada cantidad de imágenes de interiores mediante la restricción de la iluminación (N. Sahragard & Ramli, 2009).

El tercer grupo supone que la distribución estadística de los colores de la escena no sirve de nada sin el conocimiento de la reflectancia. Ellos sólo son buenos para escenas restringidas.

El cuarto grupo obtiene la medida de iluminación de manera indirecta. Shafer (Shafer, 1985) y Klinker (Klinker, Shafer, & Kanade, 1988) utilizan la especularidad de la superficie, Dana et al. (Dana, Van Ginneken, Nayar, & Koenderink, 1999) utiliza interreflexiones para medir la iluminación. Todos estos métodos utilizan una sola fuente de iluminación lo cual limita su aplicación para imágenes al aire libre ya que la luz del día es una fuente de luz compuesta y extendida.

Al quinto grupo pertenecen los métodos que necesitan en la escena superficies de reflexión conocida para encontrar la fuente de iluminación. El algoritmo Retinex de Edwin H. Land (Land & McCann, 1971) y sus múltiples variantes utilizan la máxima reflexión de superficie blanca para una mejor estimación. También el algoritmo supervisado de constancia de color de Novak y Shafer (Novak & Shafer, 1994), requiere superficies de reflexión conocida.

El sexto y el último grupo lo componen los algoritmos que suponen la dimensión de las funciones base del espectro, como en (Simonds, 1963) requieren que el modelo de reflexión de superficie sea correcto. (Maloney & Wandell, 1986) y (Yuille, 1987) suponen que una combinación lineal de dos funciones base es suficiente. Pero no está claro cómo esa suposición en el espacio de longitud de onda se aplica a un espacio de color de dimensión reducido, tales como el triestímulo RGB que trata (Finlayson et al., 1995) ampliamente. Este método sólo se aplica a una cantidad limitada de escenas debido a las restricciones hechas sobre la iluminación.

De entre estos grupos de constancia de color podemos observar que en su mayoría son pensados para interiores, ya que necesitan condiciones de luz conocidas como la posición de la fuente de iluminación, y las que se han intentado usar para exteriores se basan en las ya mencionadas de interiores, lo cual representa una gran limitación ya que no se trata de adaptar sino de buscar una mejor alternativa de visión pensada desde el principio para exteriores. Por otra parte los modelos de predicción de color basados en la iluminación de la luz de día están basados en experimentos en los cuales influyen mucho tanto las condiciones atmosféricas, como la geolocalización de estas pruebas.

2.4 Modelos de color

El uso de color en el procesamiento de imágenes es muy utilizado, esto es debido a que un color puede simplificar la identificación, así como la extracción de un objeto en una escena. El procesamiento de imágenes a color se divide en dos áreas: procesamiento con color real y procesamiento con pseudocolor.

El propósito de un modelo de color es facilitar la especificación de los colores en algún estándar. Un modelo de color es una especificación de un sistema coordenado y un subespacio dentro de este sistema donde cada color es representado por un punto. El color, como cualquier otro recurso, también tiene su técnica y está sometido a ciertas leyes, y según la aplicación que se desea, se trabaja con distintos modelos de color. Los modelos de color describen los colores que se ven en las imágenes digitales e impresas y el trabajo con ellos.

Los modelos de color permiten no sólo establecer un espacio único común a todos los equipos que forman parte de la cadena de adquisición y reproducción de color, sino que también permiten simular cómo lucirá la imagen y su color en otro dispositivo de la cadena.

Cada modelo de color representa un método diferente de descripción de los colores.

En términos del procesamiento digital de imágenes, los modelos de color más comúnmente utilizados son: RGB (Red, Green, Blue), que es utilizado en monitores de color y en video cámaras; CMY (Cyan, Magenta, Yellow), que es utilizado en impresoras, y HSI (Hue, Saturation, Intensity), el cual corresponde con la forma como los humanos describen e interpretan el color

2.4.1 Modelo RGB

El modelo RGB es de síntesis aditiva del color, o color luz. La forma más directa de operar consiste en utilizar los vectores de los colores rojo, verde y azul en el rango [0; 1]. A esta convención se le llama modelo RGB. Este es el modelo de definición de colores en pantalla usado para trabajos digitales.

La pantalla está compuesta por píxeles, donde en realidad cada píxel está formado por un conjunto de tres subpixeles: uno rojo, uno verde y uno azul, cada uno de los cuales brilla con una determinada intensidad. El monitor produce entonces los puntos de luz, partiendo de tres tubos de rayos catódicos, uno para cada color.

Para indicar con qué proporción mezclamos cada color en pantalla, se asigna un valor a cada uno de los colores primarios, como se puede observar en la Figura 8, de manera que el valor 0, por ejemplo, significa que no interviene en la mezcla, y a medida que ese valor aumenta, se entiende que aporta más intensidad a la mezcla.

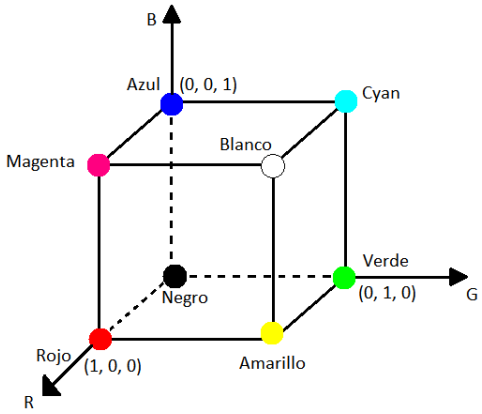

Figura 8 Espacio de color RGB

2.4.2 Modelo HSI

Los modelos RGB y CMY son ideales para implementaciones en hardware, sin embargo no pueden utilizarse para la descripción e interpretación de los colores comparados con la forma en la que los humanos lo hacen, es decir, cuando una persona observa un color, no lo describe indicando sus componentes, el porcentaje de cada uno de los colores primarios, ni siquiera le es posible saber qué colores lo conforman.

Para solucionar el problema anterior, se utiliza el modelo HSI, que define un modelo de color en términos de sus componentes constituyentes. Funciona de manera similar a como los humanos observan los colores.

Cuando un humano observa un color, en lugar de hacer su descripción en base a los porcentajes de los colores primarios, lo hace en base al color puro, la saturación y el brillo. El modelo HSI se encuentra formado por los componentes descritos a continuación.

- Matiz (H). Este componente describe el color puro del objeto, como por ejemplo: rojo, azul, verde, amarillo, etc.
- Saturación (S): Proporciona la medida en la que el color puro ha sido diluido por el color blanco.
- Intensidad (I): Proporciona el nivel del brillo del objeto.

En la Figura 9 se puede observar cómo es que funciona cada uno de los valores de HSI.

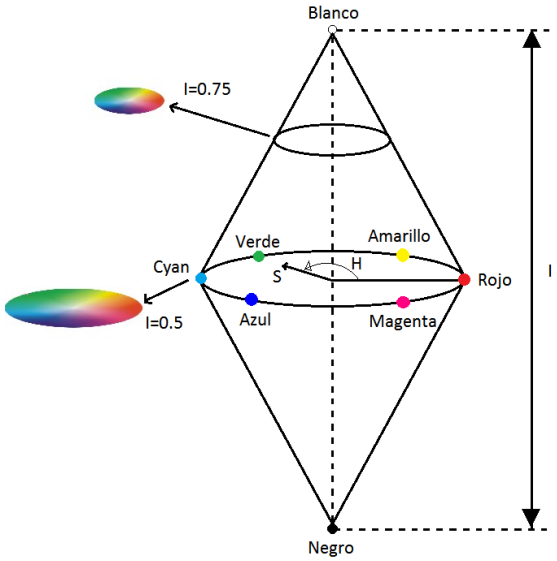

Figura 9 Espacio de color HSI

Para realizar la transformación de RGB a HSI se utiliza un programa, del cual se presenta a continuación su pseudocódigo, que primero obtiene los valores R, G, B y posteriormente se calculan los valores de H, S e I.

```
Funcion: Conversion RGB a HSI
Comienza
   para( y=0; y<altura; y++ )
      para( x=0; x<ancho; x++ )
         indice = ancho*y + x;
         B = pRGB[3*indice]/255.0;
         G = pRGB[3*indice+1]/255.0;
         R = pRGB[3*indice+2]/255.0;
         Min = min(min(R,B),G);

         I = (R + G + B)/3.0;

         Si( (R == G) && (G == B) )
            S = 0.0;
            H = 0.0;
         Fin si
         en otro caso
            S = 1.0 - (3.0 / (R + G + B)) * Min;
            Angulo=(R-G*0.5-B*0.5)/sqrt((R-G)*(R-G)+(R-B)*(G-B));
            H = arcocoseno( Angulo ) * 57.29577951;
            si( B > G )
               H = 360.0f - H;
            Fin si
         Fin otro caso
         iH = (int)(H * 255.0 / 360.0);
         pHSI[3*indice] = iH;
```

```
         pHSI[3*indice+1] = S * 255.0;
         pHSI[3*indice+2] = I * 255.0;
      fin para
   fin para
fin
```

2.5 Filtro de la luz en exteriores

2.5.1 Luz Polarizada

Como vimos anteriormente, un problema común a la visión en exteriores es el brillo causado por el reflejo de la luz del sol. Un filtro polarizador elimina eficazmente el brillo en las superficies producido por la luz solar, mejorando así la visión del ser humano al utilizarlo en lentes de sol y en cámaras fotográficas (Wolff, 1995); como la visón por computadora intenta imitar la visión del ojo humano es natural pensar que si un filtro polarizador mejora la visión del ojo humano entonces pueda aplicarse este mismo en la optimización de la visión en robots. Ejemplos de que la polarización ha sido aplicada en el área de visión los podemos encontrar en Scheshner (Schechner, Narasimhan, & Nayar, 2003) donde se plantea un algoritmo que elimina la neblina en una imagen utilizando imágenes polarizadas y en Chen (Chen & Wolff, 1998) donde se logra una diferenciación entre materiales conductivos y dieléctricos, por medio de visión por computadora con incorporación de luz polarizada y filtros polarizadores.

Después de tener una visión global de las técnicas existentes usadas en la visión por computadora en ambientes exteriores, a continuación se presenta un resumen de la polarización de las ondas electromagnéticas para tener una mejor comprensión de cómo funciona el filtrado.

2.5.2 Polarización de la luz

Una onda electromagnética es una onda transversal compuesta por un campo eléctrico y un campo magnético simultáneamente Figura 10. Ambos campos oscilan perpendicularmente entre sí; las ecuaciones de Maxwell modelan este comportamiento (Born & Wolf, 1999).

Habitualmente se decide por convenio que para el estudio de la polarización electromagnética se tome en cuenta exclusivamente el campo eléctrico, ignorando el campo

magnético, ya que el vector de campo magnético puede obtenerse a partir del vector de campo eléctrico, pues es perpendicular y proporcional a él.

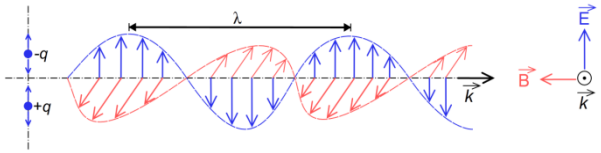

Figura 10.- Onda Electromagnética

Un ejemplo sencillo para visualizar la polarización es el de una onda plana, que es una buena aproximación de la mayoría de las ondas luminosas.

En un punto determinado la onda del campo eléctrico puede tener dos componentes vectoriales perpendiculares (transversales) a la dirección de propagación. Las dos componentes vectoriales transversales varían su amplitud con el tiempo, y la suma de ambas va trazando una figura geométrica. Si dicha figura es una recta, la polarización se denomina lineal; si es un círculo, la polarización es circular; y si es una elipse, la polarización es elíptica.

Si la onda electromagnética es una onda armónica simple, como en el caso de una luz monocromática, en que la amplitud del vector de campo eléctrico varía de manera sinusoidal, las dos componentes tienen exactamente la misma frecuencia. Sin embargo, estas componentes tienen otras dos características de definición que pueden ser diferentes. Primero, las dos componentes pueden no tener la misma amplitud. Segundo, los dos componentes pueden no tener la misma fase, es decir, pueden no alcanzar sus máximos y mínimos al mismo tiempo.

En la polarización lineal la oscilación del plano perpendicular a la dirección de propagación se produce a lo largo de una línea recta. Se puede representar cada oscilación descomponiéndola en dos ejes X e Y. La polarización lineal se produce cuando ambas componentes están en fase (con un ángulo de desfase nulo, cuando ambas componentes alcanzan sus máximos y mínimos simultáneamente) o en contrafase (con un ángulo de desfase de 180º, cuando cada una de las componentes alcanza sus máximos a la vez que la

otra alcanza sus mínimos). La relación entre las amplitudes de ambas componentes determina la dirección de la oscilación, que es la dirección de la polarización lineal.

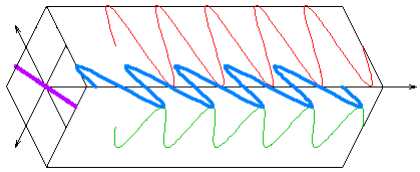

Figura 11.- Polarización lineal

En la polarización circular, las dos componentes ortogonales tienen exactamente la misma amplitud y están desfasadas exactamente 90º. En este caso, una componente se anula cuando la otra componente alcanza su amplitud máxima o mínima. Existen dos relaciones posibles que satisfacen esta exigencia, de forma que la componente x puede estar 90º adelantada o retrasada respecto a la componente Y. El sentido (horario o anti horario) en el que gira el campo eléctrico depende de cuál de estas dos relaciones se dé. En este caso especial, la trayectoria trazada en el plano por la punta del vector de campo eléctrico tiene la forma de una circunferencia, por lo que en este caso se habla de polarización circular.

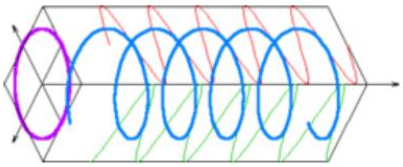

Figura 12.- Polarización circular

La polarización elíptica corresponde a cualquier otro caso diferente a los anteriores, es decir, las dos componentes tienen distintas amplitudes y el ángulo de desfase entre ellas es diferente a 0º y a 180º (no están en fase ni en contrafase).

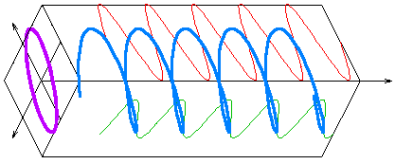

Figura 13.- Polarización elíptica

Algunos materiales absorben selectivamente una de las componentes transversales del campo eléctrico de una onda. Esta propiedad se denomina dicroísmo. La luz experimenta una absorción en ciertos estados de polarización. El término dicroísmo proviene de las observaciones realizadas en épocas muy tempranas de la teoría óptica sobre ciertos cristales, tales como la turmalina. En estos cristales, el efecto del dicroísmo varía en gran medida con la longitud de onda de la luz, haciendo que aparezcan diferentes colores asociados a la visión de diferentes colores con diferentes planos de polarización. Este efecto es también denominado pleocroísmo, y la técnica se emplea en mineralogía para identificar los diferentes minerales. En algunos materiales, tales como la herapatita (sulfato de iodoquinina) o las capas Polaroid, el efecto no es tan fuertemente dependiente de la longitud de onda, y ésta es la razón por la que el término dicroico se emplea muy poco.

2.5.3 Filtros Polarizadores

Un efecto de los filtros polarizadores comúnmente observado es la reducción drástica del resplandor de la luz del sol reflejado sobre superficies horizontales, que es la razón principal por la que a menudo se usan filtros polarizadores en lentes de sol.

Para polarizar la luz a partir de luz natural no polarizada se usan unos elementos llamados filtros que son dispositivos análogos a la ranura para las ondas mecánicas. Los filtros polarizadores para las ondas electro-magnéticas (OEM) presentan diferentes detalles en su construcción dependiendo de la longitud de onda de la OEM. Para microondas con una longitud de onda de unos cuantos centímetros, un buen polarizador es una disposición de cables conductores paralelos colocados muy juntos y aislados unos de otros, en estos los electrones se pueden mover libremente a lo largo de la longitud de los cables y lo harán como respuesta a una onda cuyo campo $\vec{E}$ es paralelo a los cables. Las corrientes resultantes en los conductores disipan energía mediante calentamiento Ri^2; esta energía disipada proviene de la onda, de manera que la onda que pase por la rejilla verá muy reducida su amplitud. Las ondas con el campo $\vec{E}$ orientado de manera perpendicular a los cables pasa a través de la rejilla casi sin verse afectadas, puesto que los electrones no se pueden desplazar a través del aire que hay entre los cables conductores. Por tanto una onda que atraviese dicho filtro estará predominantemente polarizada en una dirección perpendicular a los cables. Para la luz visible el filtro polarizador más común es un material conocido con el nombre comercial de Polaroid, utilizado en gafas de sol y en objetivos de cámaras fotográficas. Dicho material incorpora sustancias que presentan la propiedad de

dicroísmo, que significa que presenta una absorción selectiva en la que una de las componentes del campo se absorbe con mucha más intensidad que la otra, de modo que el material Polaroid trasmite el 80 % o más de la intensidad de una onda polarizada según una dirección paralela a cierto eje del material, llamado eje de polarización P_e, pero sólo el 1% o menos de la intensidad de las ondas polarizadas perpendicularmente a este eje. Las moléculas que componen estos materiales presentan una forma alargada, indicada por la orientación u_M, de modo que en el proceso de construcción del material todas las moléculas del filtro se sitúan de tal modo que las orientaciones u_M sean todas ellas paralelas. Esta dirección alargada u_M es perpendicular al eje de polarización P_e; y estas moléculas absorben preferentemente la luz que está polarizada a lo largo de u_M.

2.5.4 Angulo de Brewster

La luz relejada por una superficie puede estar polarizada total o parcialmente (Born & Wolf, 1999). Si una luz natural incide sobre una superficie reflectora que separa dos materiales ópticos transparentes, para la mayoría de los ángulos de incidencia la componente del campo eléctrico perpendicular al plano de incidencia (o paralela a la superficie reflectora) se refleja con más intensidad que la componente paralela al plano de incidencia. En este caso la luz reflejada se dice que está parcialmente polarizada en la dirección normal al plano de incidencia. En la Figura 14 podemos observar este efecto.

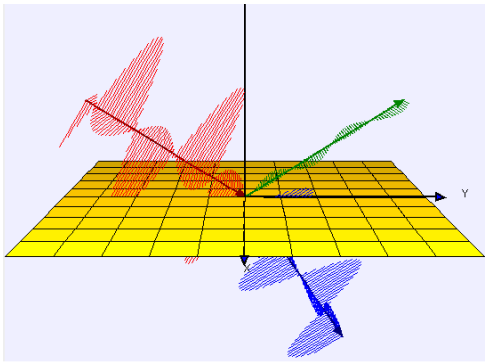

Figura 14.- Onda polarizada horizontalmente

2.6 RoboCup

Los trabajos más relacionados en el área de coordinación de robots y que además resuelven el problema específico de que un par de robots se den pases con una pelota en ambientes interiores y sobre superficies lisas, que es el problema específico de esta tesis con la

diferencia de ser en exteriores y sobre superficies rugosas, los encontramos en la RoboCup. En la RoboCup existen 2 tipos de competiciones, la simulada, y la física. En la competición simulada existen la liga de futbol simulada en 2D y la liga de futbol simulada en 3D, ambas competiciones, por ser simuladas y no tener que preocuparse por los problemas físicos, se enfocan principalmente en el desarrollo de estrategias de equipo e inteligencia artificial. La diferencia entre ambas ligas es la complejidad del ambiente en la simulación de la liga en 3D por las leyes físicas más reales. La mayor ventaja del ambiente de simulación en 3D es que sirve como plataforma de pruebas del robot NAO de Aldebaran antes de programar el robot físico para observar su funcionamiento, esto es posible gracias a que el robot usado en la simulación en 3D es un modelo del robot NAO de Aldebaran Robotics que como se verá más adelante es el robot utilizado en la liga estándar. La competición física se compone de las siguientes ligas.

2.6.1 Liga de tamaño pequeño

En esta liga se permiten 5 robots que juegan con una pelota de golf en un campo del tamaño de una mesa de pingpong. El tamaño de los robots es limitado ya que estos deben de caber en un cilindro de 15 cm de alto por 18 cm de radio. Los componentes externos comúnmente consisten de una cámara aérea para la retroalimentación visual, una computadora para procesar esta información y determinar el comportamiento del equipo y también un sistema de comunicación inalámbrico entre la computadora y los robots (RoboCup, 2011b).

2.6.2 Liga de tamaño mediano

En esta liga los equipos se componen por un portero y tres jugadores de mediocampo que se distinguen gracias a unos marcadores de colores cian y magenta. El tamaño de los robots está delimitado por el volumen de un cilindro de 50 cm de diámetro y 80 cm de altura. El tamaño del campo de juego es de 5 por 9 metros y la pelota con la que se juega es un balón del número 4, que es el tamaño oficial para futbol de salón, y es de color anaranjado. Todos los sensores deben de estar en el robot, por lo tanto no se permite el uso de una visión global ni de sensores externos (Candea, Hu, Iocchi, Nardi, & Piaggio, 2001).

2.6.3 Liga estándar

Existen otras dos ligas, una es la liga estándar (RoboCup, 2011a) que utiliza 4 robots NAO de Aldebaran (Robotics), estos robots son humanoides y cuentan con 25 grados de libertad, un par de cámaras para la visión, sensores ultrasónicos en el pecho y conexión WiFi para la comunicación entre ellos, la cancha mide 4 m de ancho por 6 m de largo y la pelota que se usa es una pelota de hockey callejero marca Mylec. En esta liga no se permite ningún sensor ni equipo de procesamiento exterior así como tampoco se permite modificar en ningún aspecto al robot.

2.6.4 Liga humanoide

Otra liga es la liga de robots humanoides (RoboCup), que a su vez se divide en tres tamaños, tamaño infantil de 30 a 60 cm de altura, tamaño adolecente de 100 a 120 cm de altura y tamaño adulto de tamaño superior a los 130 cm. Estas ligas al ser de robots humanoides se salen del contexto de este tema de investigación y es por eso que se omitirán en la siguiente parte donde se explicara como algunas arquitecturas de coordinación se han implementado en los robots que juegan futbol.

2.7 Más allá de RoboCup y sus limitaciones

Dentro de la liga de tamaño pequeño es donde mejor se ubica nuestro problema porque es en esta donde el procesamiento visual se realiza en una computadora externa al robot y solo se le transmiten los resultados de este procesamiento de manera inalámbrica y el robot por su parte cuenta con un programa que tiene descrito el comportamiento cooperativo que debe llevar a cabo. La diferencia más importante es de donde se obtiene la información visual, ya que mientras en la liga de tamaño pequeño se obtiene una panorámica del terreno por medio de una vista aérea con la cual se pueden observar a todos los robots y la pelota al mismo tiempo, en nuestro caso la información visual se obtendrá desde la cámara a bordo de cada robot como en la liga de tamaño medio.

Un ejemplo de arquitectura de coordinación implementado en la liga de tamaño pequeño lo encontramos en MAPS (Tews & Wyeth, 2000), el cual es un algoritmo de planeación de alto nivel que envía coordenadas a los robots y se resume en tres pasos. Primero obtiene información nueva de los sensores de los robots del equipo y de la predicción del

comportamiento de los otros robots para poder actualizar el modelado del mundo que se representa por medio de campos potenciales. En segundo lugar y basándose en los campos potenciales de: objetos, posiciones de los robots, distancia a un punto y trayecto libre a un punto se toma la decisión de que es lo que se va a hacer evitando que los robots se estorben y respetando el objetivo principal del equipo. Por último se debe encontrar la ubicación de la acción a realizarse en base a esta y desde la perspectiva individual del robot y la acción a realizarse.

Dentro de la liga de tamaño mediano encontramos algunos ejemplos como ART (Nardi et al., 2000) que es una arquitectura cuya coordinación se realiza por medio de un sistema compuesto por formación y roles, como fueron introducidos por (Stone & Veloso, 1999) en donde por cada formación hay roles específicos. Cada robot posee el conocimiento necesario para llevar a cabo cada rol y por lo tanto estos se pueden intercambiar sobre la marcha en caso de ser necesario. Existen dos motivos principales para la creación de los roles, la primera es que si dos robots trataran de ir a la pelota al mismo tiempo se estorbaría y esto perjudicaría al equipo y la segunda que para ganar es necesario ocupar las zonas de la cancha de manera apropiada, por ejemplo un robot que ataque junto con uno que acompañe al ataque mientras que otro se quede a defender (Candea et al., 2001).

Otro ejemplo de coordinación en la liga de tamaño mediano es el equipo CS Freiburg (Weigel, Gutmann, Dietl, Kleiner, & Nebel, 2002) el cual aporta un método novedoso de seguimiento de la pelota y de los adversarios por medio de múltiples observaciones utilizando un filtro de Kalman el cual descarta las mediciones que están fuera de la concentración de la distribución de probabilidad obtenida por las otras mediciones. Para la coordinación nuevamente se utilizan roles. Estos roles son los siguientes:

- Activo.- Este rol indica que es el encargado de manejar la pelota y tiene varias opciones para aproximarse a la pelota y también alejarla o acercarla a la portería contraria.
- Estrategia.- Este rol es como el defensivo descrito anteriormente, el robot asegura una posición defensiva dentro de su medio campo.
- Soporte.- Este rol tiene dos comportamientos dependiendo la situación del juego, si se está defendiendo complementa la formación defensiva y si se está atacando se coloca en una posición para recibir pase cerca de la portería rival.
- Portero.- Este rol mantiene al portero en la línea de gol y solamente se mueve dependiendo de la velocidad, posición y dirección de la pelota.

3. Coordinación de robots en exteriores

Resumen. Se describe la implementación de la arquitectura MURDOCH para la coordinación de robots al mandarse pases de pelota sobre terrenos exteriores: se definen los componentes de la arquitectura, se describe el algoritmo para detectar la pelota en exteriores y seguir su trayectoria, junto con la detección mejorada del color de la pelota aplicando un filtro polarizado.

Después de presentar algunas implementaciones de arquitecturas de coordinación de robots podemos concluir que ninguna aborda específicamente nuestro problema ya que, por ejemplo, en MAPS (Tews & Wyeth, 2000) no existe ninguna acción implícita de mandar un pase; esta acción se obtiene de juntar los campos potenciales de "patear a" en el caso del robot con la pelota y el de "ir a" del robot con el rol de soporte. En nuestro problema "mandar pase" se define como una tarea así como "recibir pase", es por esto que ese enfoque no funciona además de estar altamente motivado por estrategias de ataque y por las posiciones de un equipo rival.

En el caso de la liga mediana, que también es interesante para nuestro trabajo por llevar la visión dentro del robot, el enfoque de asignar roles parece una buena opción. Sin embargo, debido a que solo tendremos un par de robots, este enfoque no serviría ya que está basado en poses predefinidas que son conocidas y evaluadas por ambos robots para seleccionar la mejor. En consecuencia, podría no haber mayoría de votos y por ende el sistema podría presentas fallas. En el caso del seguimiento de la bola se puede ver que tampoco sirve dicho enfoque porque de igual manera utiliza un consenso entre las mediciones de todos los robots y en caso de que una medición fallara, el sistema también fallaría.

Este capítulo presenta el desarrollo de esta propuesta como se enuncia a continuación:

- Diseño de robot
- Arquitectura MURDOCH
- Cálculo de la odometría
- Procesamiento visual
- Comunicación inalámbrica
- Diseño del aprendizaje de golpeo

3.1 Robot NXT MINDSTORM de LEGO

Este robot cuenta con una serie de piezas que facilitan el armado de estructuras de entre las cuales se pueden destacar: figuras humanoides, de animales, de vehículos de 2, 3 y 4 ruedas, palancas mecánicas, etc. El diseño se debe de concentrar alrededor de un ladrillo llamado Brick NXT, el cual es el dispositivo programable que funciona como el cerebro del robot que se desee construir. A este ladrillo se le pueden conectar sensores para brindarle a nuestro robot una percepción del mundo que lo rodea y servomotores para proveer un mecanismo de interacción o reacción a este mundo. A continuación se mencionan brevemente algunas características tanto del ladrillo como de los sensores y servomotores.

3.1.1 Ladrillo NXT

Como se mencionó anteriormente, el ladrillo es el cerebro programable de nuestro robot y a continuación se describen sus características de procesador, puertos y comunicación que tiene integradas.

El microcontrolador que posee es un ARM7 de 32 bits, que incluye 256 Kb de memoria Flash y 64 Kb de RAM externa. Además, posee mayores capacidades de ejecución de programas que las de su predecesor, evitando que los procesos inherentes de varios paquetes de datos colisionen y produzcan errores en la ejecución del software. Su presentación es similar a la del microcontrolador Hitachi H8 ya que se encuentra en el circuito impreso del bloque, junto a la memoria FLASH.

En el bloque de NXT existen cuatro entradas para los sensores y las salidas de energía son tres y están localizadas en la parte posterior del bloque, haciendo que la conexión para los motores y partes móviles sea de más fácil acceso.

El bloque de NXT puede comunicarse con la computadora mediante la interfaz de USB que posee, la cual viene en la versión 2.0. Además, puede comunicarse con otros robots en las cercanías, ya que posee una interfaz Bluetooth que es compatible con la Clase II versión 2.0. Esta conectividad con Bluetooth no sólo le permite conectarse con otros bloques, sino también con computadoras, palms, teléfonos móviles y otros aparatos con esta interfaz de comunicación.

El *firmware* del robot Lego Mindstorms consta de las instrucciones básicas que posee el bloque NXT para hacer las distintas tareas que se le pueden programar. El *firmware* viene en el CD-ROM que se adjunta en el empaque original y debe ser cargado todas las veces que el robot se inicialice o se cambien las baterías, ya que la memoria se borra.

Si no se carga el firmware, el robot queda en modo de arranque, lo cual hace que se pueda usar un programa que viene en forma nativa dentro del robot. Para cargar completamente el firmware básico debe ejecutarse el programa adjunto y luego esperar cerca de 3 minutos.

Las versiones más actuales de Lego Mindstorms NXT, como la versión 2.0, son compatibles con las versiones anteriores del bloque, haciendo que los programas escritos en versiones más nuevas también puedan ser ejecutadas en las generaciones previas.

3.1.2 Motores

Los motores de la serie Lego Robotics han sido de tres tipos, los cuales son independientes al bloque, lo que entrega movilidad al sistema dinámico según las necesidades de construcción.

En la tabla de medición, el motor estándar es más veloz que el de 9 volts, pero este último posee más fuerza para mover el robot, ya que puede levantar cerca de 240 piezas de 8x8, aunque es más lento pero a la vez más preciso. El motor Micro es sólo para funciones menores debido a su escaso torque y la mínima velocidad de rotación.

Los motores desmontables son alimentados mediante cables que poseen conductores eléctricos, los cuales transmiten la energía a los inductores. Como son motores paso a paso, el sentido de conexión no entrega la misma dirección de movimiento.

Los motores integrados al bloque son menos versátiles, pero no dependen de conexiones externas, lo cual ayuda visualmente al robot en su presentación.

El modelo NXT usa servo motores, los cuales permiten la detección de giros de la rueda, indicando los giros completos o medios giros, que son controlados por el software.

3.1.3 Sensores

Los sensores del robot le permiten captar el entorno donde se encuentra e interactuar con él adecuadamente, en nuestro caso, visualizando correctamente la pelota. Los sensores son:

- de luz
- de contacto
- de ultrasonido

El sensor de luz permite tomar una muestra de luz mediante un bloque modificado que en un extremo trae un conductor eléctrico y en el otro una cámara oscura la cual es capaz de captar luces entre los rangos de 0,6 a 760 lux. Este valor lo considera como un porcentaje,

el cual es procesado por el bloque lógico, obteniendo un porcentaje aproximado de luminosidad.

El bloque NXT calcula con la fórmula $Luz = 146 - \frac{RAW}{7}$ el porcentaje obtenido por la lectura de la luz, tomando una muestra cada 2,9 ms, siendo leído en 100 μs el valor que se lee a partir del sensor. Debido a que este sensor capta grados de luminosidad, no es capaz de distinguir colores, sólo captando la existencia del blanco (claridad), negro (oscuridad) y los tonos de grises que corresponden a los distintos porcentajes de luz existentes en el medio.

El sensor de contacto permite detectar si el bloque que lo posee ha colisionado o no con algún objeto que se encuentre en su trayectoria inmediata. Al tocar una superficie, una pequeña cabeza externa se contrae, permitiendo que una pieza dentro del bloque cierre un circuito eléctrico y comience a circular energía, provocando una variación de energía de 0 a 5 V. En este caso, si la presión supera una medida estándar de 450, mostrada en la pantalla de LCD, se considera que el sensor está presionado; de otro modo, se considera que está sin presión.

El sensor ultrasónico sólo se incluye en el empaque de Lego Mindstorms NXT y su principal función es detectar las distancias y el movimiento de un objeto que se interponga en el camino del robot, mediante el principio de la detección ultrasónica. Este sensor es capaz de detectar objetos que se encuentren desde 0 a 255 cm, con una precisión relativa de +/- 3 cm. Mediante el principio del eco, el sensor es capaz de recibir la información de los distintos objetos que se encuentren en el campo de detección. El sensor funciona mejor cuando las señales ultrasónicas que recibe provienen de objetos que sean grandes, planos o de superficies duras. Los objetos pequeños, curvos o suaves, como pelotas, pueden ser muy difíciles de detectar. Si en el cuarto se encuentra más de un sensor ultrasónico, los dispositivos pueden interferir entre ellos, generando detecciones pobres.

3.1.4 Diseño y ensamble del robot

Primero se ensambló un robot de ruedas de tipo diferencial, esto para lograr un control más fácil sobre la posición del robot en todo momento. Después se le adaptaron llantas con alta tracción que son especiales para terrenos con irregularidades en su textura, pero como son de plástico resultan muy poco útiles en terrenos planos. Para resolver este inconveniente se les adaptó una cubierta de hule la cual les brindó la tracción necesaria para trabajar en terrenos lisos. A continuación se añadió al robot móvil una extremidad en forma de garra, la cual se fijó a un motor en la parte delantera del carro, para que permitiera golpear a la

pelota de manera precisa y en dirección frontal a una velocidad determinada por la potencia con la que se activará el motor al cual se fijó la garra. Para finalizar con el diseño del robot móvil se le implementó la cámara inalámbrica que sirvió para capturar la imagen de la pelota y permitir su seguimiento.

3.2 Arquitectura MURDOCH

Para resolver el problema de la coordinación en exteriores propuesto se pretende realizar una separación del procesamiento de las imágenes y el comportamiento autónomo del robot dado que la arquitectura del robot no soportaría una operación eficiente si tuviera que hacer el procesamiento visual. Para poder unir estas dos partes se cuenta con la comunicación de tipo inalámbrica Bluetooth. A continuación se muestra un diagrama con dicha separación del software:

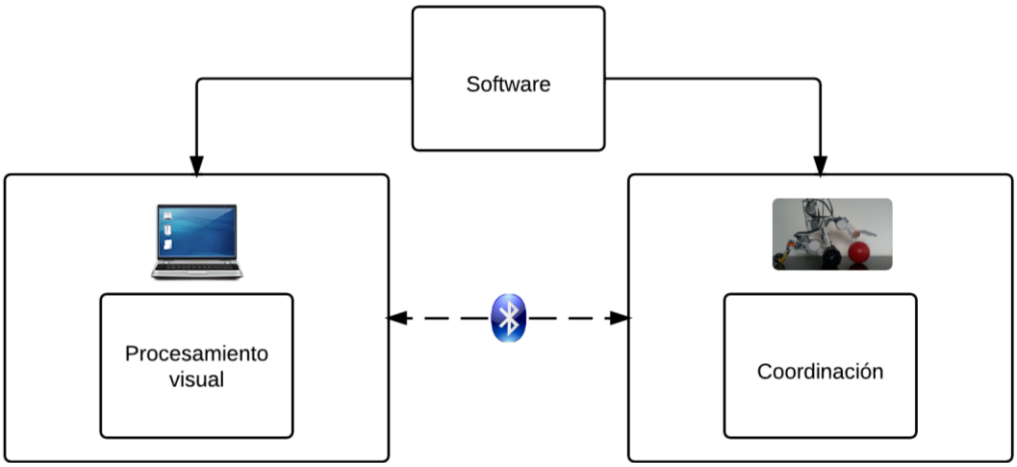

Figura 15.- Arquitectura general

El objetivo principal de esta etapa es elaborar el algoritmo de coordinación, el cual se compondrá de un ciclo que se repetirá hasta que la tarea principal sea realizada. Con la arquitectura que se empleará para la coordinación de los robots es necesario descomponer la tarea principal en subtareas que puedan ser realizadas de manera individual por los robots. La descomposición de la tarea principal en subtareas se describe a continuación.

Como ya se explicó en el capítulo 1, la tarea principal es la de llevar una pelota desde un punto inicial hasta un punto final por medio de pases, entonces las subtareas en las que la tarea principal puede descomponerse son:

- buscar pelota
- mandar pase y
- recibir pase

A continuación se plantean las acciones a ser realizadas en cada una de estas tareas enfocándonos en la arquitectura MURDOC la cual está basada en subastas.

1.- Buscar pelota.- Esta es una acción que en principio solo se realizara al comenzar cada tarea ya que el robot desconoce la posición inicial de la pelota y es por eso que debe buscarla. Esta acción será realizada por los dos robots al mismo tiempo los cuales estarán buscando y preguntando a la vez si ya se encontró la pelota. Si un robot encuentra la pelota, mandará un mensaje de tipo difusión para que los demás robots

Aquí se plantean 2 estados del robot:

1. **Robot con pelota**.- Este será el agente encargado de subastar la tarea de recibir pase. Cuando esté en posesión de la pelota se encargará de anunciar la tarea de recibir pase, enviando el nombre de la tarea así como la métrica a ser evaluada para decidir a qué robot se le asigna la tarea. Después esperará un tiempo determinado para dejar de recibir ofertas y decidirá a quien se le asigna la tarea en base a las ofertas recibidas. Como la comunicación es anónima realizará la notificación con un mensaje de tipo *broadcast* con el valor de la métrica de la oferta ganadora y la posición a la que calculó que haría llegar el pase dependiendo de la rugosidad del terreno, después realizará el pase a la posición calculada y a partir de ese momento comenzará a monitorear el progreso de la tarea, la cual debe ser completada en un cierto tiempo antes de que decida reasignar la tarea a otro robot, esto último se hace para brindar un soporte a fallos. Por último y como notificación de que el robot que recibe el pase lo haya hecho correctamente, esperará por las coordenadas reales a donde llego la pelota y ajustará la fuerza con la cual deberá de golpear a la pelota la siguiente vez en ese terreno.

2. **Robot sin pelota**.- Este robot esperará por la notificación de una tarea nueva y evaluará su aptitud para llevarla a cabo en base a la métrica recibida. Después enviará su oferta con el valor de la evaluación de la métrica y esperara por la notificación del agente subastador. En caso de resultar ganador se moverá hacia las coordenadas de la posición final de la pelota y en el camino buscara a la pelota para ajustar su trayectoria de tal manera que la pueda intersectar. Por ultimo informara de la posición final de la pelota y se la enviara al agente subastador para mejorar el golpeo dependiendo del terreno en el que se encuentre.

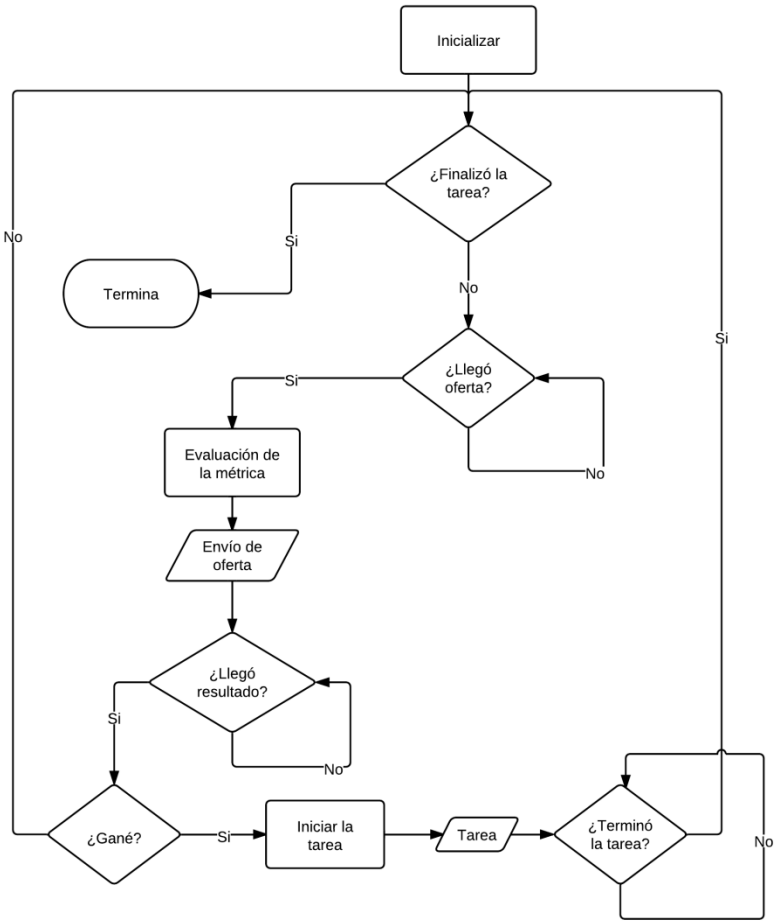

Figura 16.- Estado de robot sin pelota

Como se mencionó anteriormente la arquitectura MURDOCH utiliza una comunicación por mensajes de tipo suscripción/publicación. La principal característica por la cual se utiliza este tipo de comunicación es porque los mensajes son direccionados por contenido en vez de destino. Esta idea se utiliza para dividir a la red en una asociación débilmente unida de productores y consumidores de información anónimos. Un productor de datos simplemente etiqueta su mensaje con un tema y lo publica en la red, los consumidores que hayan registrado su interés en ese tema al suscribirse a él lo reciben de manera automática. La arquitectura MURDOC modifica un poco este modelo de tal manera que un mensaje no sea etiquetado con un solo tema, sino con un conjunto de temas. Un consumidor recibirá el mensaje si el conjunto de temas del mensaje comprende algún subconjunto de los temas que se encuentren en la lista de suscripciones de ese consumidor.

El primer paso para crear un sistema de suscripción/publicación es el de designar las semánticas de los nombres de dominio. Análogamente a cuando se define una estructura de datos, la selección de las características tendrá una gran influencia en el resto del sistema.

En MURDOCH se usan palabras que representen sus recursos, los cuales pueden ser dispositivos físicos, capacidades de alto nivel y nociones abstractas de su estado actual.

Definición del espacio de características:

Dispositivos físicos

Cámara.- El robot cuenta con una cámara montada en su estructura física como se muestra en la imagen

Garra.- El robot cuenta con una garra montada en su estructura física como se muestra en la imagen.

Figura 17.- Detalle de la garra incorporada al robot

Capacidades de alto nivel

Móvil.- Que el robot sea capaz de desplazarse.

Golpear.- Que el robot sea capaz de golpear a la pelota.

Seguir.- Que el robot sea capaz de seguir a la pelota, debe contar con cámara y ser móvil.

Ajustar.- Para que el robot ajuste la distancia a la que puede llegar la pelota al ser golpeada.

Estado actual

Buscando.- El robot no conoce la ubicación de la pelota y la está buscando.

Siguiendo.- El robot se encuentra en seguimiento de una pelota.

Golpeando.- El robot está golpeando a la pelota con su garra.

Esperando.- El robot espera por una nueva tarea.

La primera tarea será lanzada por el usuario y esta consistirá en buscar la pelota, quien la encuentre primero se considerara en posesión de esta y será la encargada de realizar el pase y a su vez de subastar la tarea de recibir pase.

3.3 Comunicación inalámbrica tipo *broadcast*

La comunicación entre los robots debía de cumplir 2 características, ser inalámbrica y ser de tipo *broadcast*.

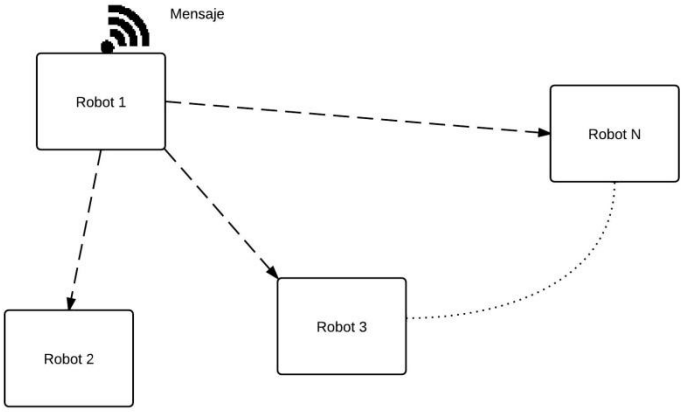

Figura 18 La comunicación de tipo *broadcast*

Como no se contaba con ningún dispositivo de radiofrecuencia que se le pudiera adaptar al robot y nos permitiera hacer uso de la comunicación broadcast, se decidió utilizar la comunicación de tipo *bluetooth,* que ya viene incorporada en los robots, y por medio de la comunicación maestro esclavo del bluetooth, se pudo simular la comunicación de tipo *broadcast* de tal manera que cuando un robot necesitara realizar una transmisión, la computadora leyera el mensaje del robot y lo mandara a todos los demás robots uno a uno por medio también del *bluetooth*. De esta manera se logró la comunicación de tipo *broadcast* simulándola con la comunicación maestro esclavo entre la computadora y los robots como se puede observar en la figura 19.

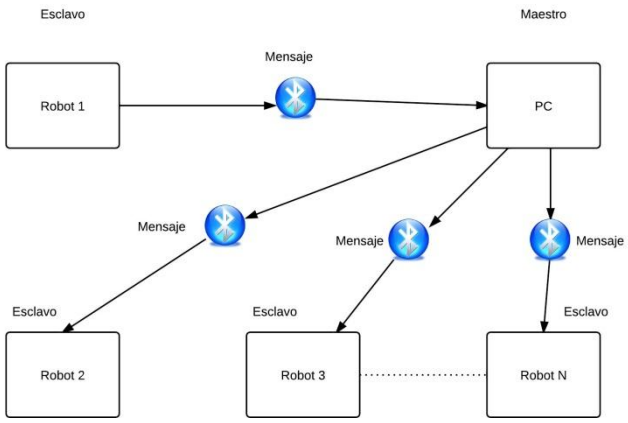

Figura 19 Comunicación de tipo *Broadcast* simulada por medio de Bluetooth

Cabe destacar que los robots no son capaces de iniciar una transmisión por medio del bluetooth, ya que la arquitectura de comunicación de los robots lego es por medio de buzones. Estos buzones se dividen en buzones de entrada y buzones de salida, en los buzones de entrada se escriben los mensajes enviados al robot y este, escribe en los buzones de salida los mensajes que quiere enviar por *broadcast*.

3.4 Cálculo de la odometría

Empecemos definiendo, ¿Qué es la odometría?, la odometría es el proceso por el cual se puede estimar la posición actual de un robot en un sistema de coordenadas con un origen conocido, a partir de la medición de cambios en los giros de sus ruedas. Seleccionar la manera más adecuada que exista para obtener la odometría del robot y así lograr una mayor precisión en la estimación de la posición del robot. Se tiene contemplado que la posición del robot pueda ser calculada a partir de la velocidad y dirección que este tenga porque serán datos conocidos para cada robot. La odometría calculada por encoder es la opción más fácil de implementar en los robots móviles ya que únicamente se lee de las llantas el número de vueltas o grados que ha girado y como el diámetro de la rueda es conocido se puede estimar la posición del robot.

Para obtener la odometría en exteriores, es importante tomar en cuenta algunos aspectos importantes: primero tenemos el tipo de llantas que se utilizarán con el robot, estas deben de tener el mayor agarre posible para minimizar los errores causados por los derrapes. Otro aspecto importante es la distribución del peso del robot, como estamos trabajando con un robot con orugas, es importante que el peso recaiga sobre las ruedas motorizadas para

mejorar el agarre del robot al piso. Una vez minimizados los derrapes se puede comenzar a calcular la odometría por medio de la lectura de los grados que gira cada rueda. Para realizar este cálculo es necesario contar con 2 medidas: La distancia que hay entre las ruedas (D) y la distancia que avanza por cada grado (CMxG) que gire la llanta en cuál es obtenido a partir del diámetro de la rueda, aunque en este caso, por tratarse de las orugas para tener una mejor medición de este valor se realizó una corrida en línea recta del robot por un tiempo pequeño y se contaron las vueltas que dio la rueda motorizada y estas vueltas se dividieron entre los cm medidos con una cinta métrica. Se repitió este procedimiento hasta que se encontró un valor estable de CMxG.

El cálculo de la odometría se lleva a cabo de la siguiente manera:

Primero se toman las rotaciones actuales de los dos motores

$$Rd = CuentaDeRotacionesMotorDerecho$$
$$Ri = CuentaDeRotacionesMotorIzquierdo$$

Después se cuenta la diferencia entre esta cuenta con la anterior

$$\delta Rd = Rda - Rdp$$
$$\delta Ri = Ria - Rip$$

A continuación se calcula la distancia absoluta recorrida por cada llanta

$$\delta d = \delta Rd * CMXG$$
$$\delta i = \delta Ri * CMXG$$

Se calculan δs y δt

$$\delta s = \frac{\delta i + \delta d}{2}$$
$$\delta t = \frac{\delta d - \delta i}{D}$$

Se obtiene la posición en x y y

$$x = x + \delta s * \cos\left(theta + \left(\frac{\delta t}{2}\right)\right)$$
$$y = y + \delta s * \sin\left(theta + \left(\frac{\delta t}{2}\right)\right)$$

Se actualiza el ángulo

$$theta = theta + \delta t$$

Se guarda la cuenta de las rotaciones actual en la anterior

$$Rdp = Rda$$
$$Rip = Ria$$

La odometría se utilizó para guardar la posición del robot en las coordenadas x, y, el ángulo de orientación, y el tiempo de cada muestra. Este muestreo no pudo ser continuo ya que la memoria del NXT es limitada por lo cual se optó por tomar un registro cada cierto intervalo de tiempo únicamente si había un cambio significativo en el movimiento realizado, por ejemplo si el robot estaba girando solo se ingresaría un nuevo registro si el ángulo cambiaba en más de 0.1°.

3.5 Procesamiento visual de la pelota

Es el que se llevó a cabo en una computadora que regresó los resultados al robot en forma de mensajes por medio de la comunicación *bluetooth*:

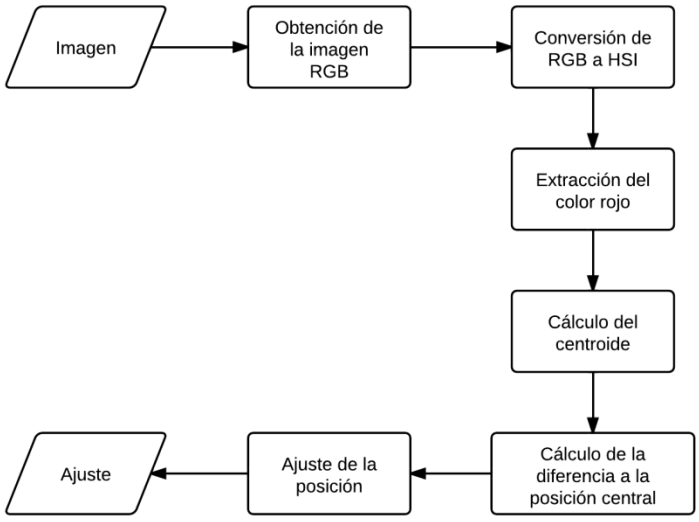

Figura 20.- Procesamiento visual

Para lograr esto necesitamos realizar un programa que permita reconocer una pelota en movimiento o estática basándonos en su color y también permita saber su posición relativa al centro del campo de visión del robot. Aquí es donde será importante la investigación del área de visión por computadora en ambientes exteriores ya que es diferente reconocer un color en interiores que en exteriores. Como mencionamos anteriormente el brillo de la luz del sol reflejada en las superficies mete mucho ruido a una imagen para poder identificar un color en específico.

La imagen que obtenemos de la cámara está en formato RGB-24. En este formato se utilizan 24 bits para describir el color de cada pixel, compuesto de rojo, verde y azul, con un byte cada uno. Este modelo de color no es muy apropiado para identificar un color ya que existen combinaciones muy distantes de los componentes que describen al mismo color. Sin embargo existe un modelo de color llamado HSI, el cual describe al color con los componentes de tono, saturación e intensidad, en otras palabras tiene 2 campos para describir el color y uno para describir el brillo de este color. Es por esto que se pretende utilizar el formato de imagen HSI para poder identificar con mayor facilidad un color en específico sin importar como se vea afectado por la luz.

Después de tener la imagen con el color rojo de la pelota se calcula el centroide de la imagen para ubicar el centro de masa de la pelota, después se pasa este valor a un módulo de ajuste el cual, como su nombre lo indica, ajustará la posición del robot para alinearlo con la pelota en una posición determinada. Este módulo se describe a continuación:

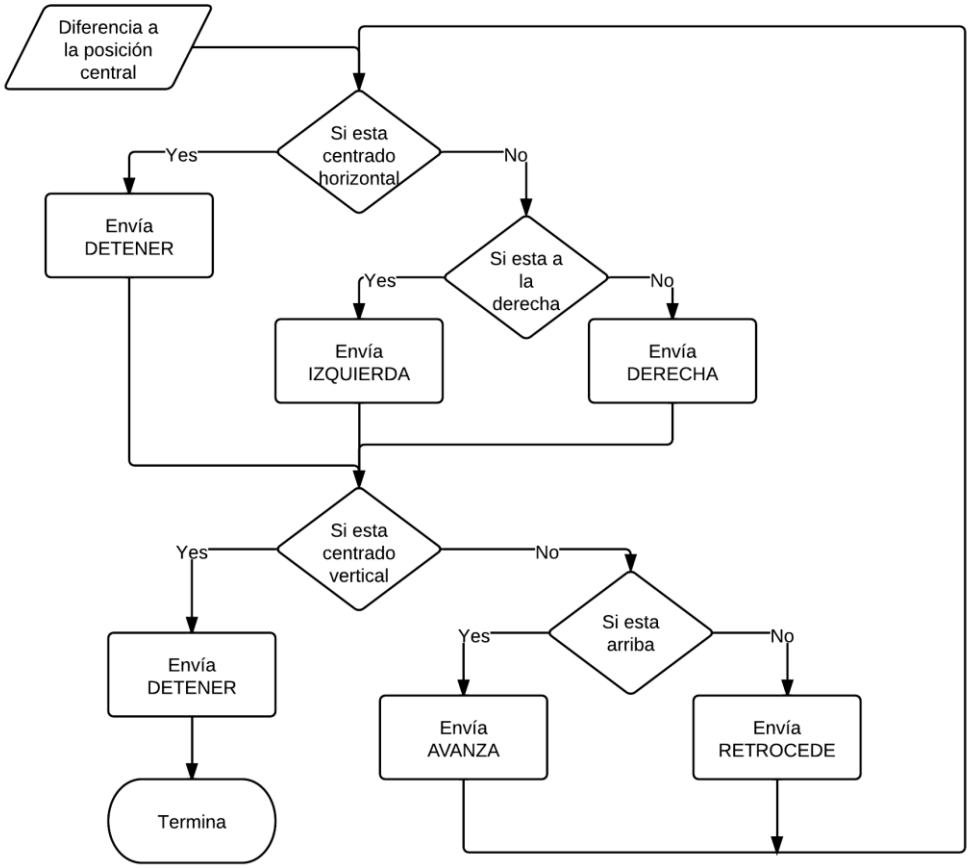

Figura 21.- Ajuste de posición

Otra etapa para reducir el brillo de las imágenes es la de filtrado por medio de un lente de polarización horizontal, ya que este elimina la mayor parte del brillo proveniente de las superficies planas como el asfalto el cual polariza la luz de manera horizontal. En la siguiente sección se explica cómo funciona este filtrado con mayor detalle.

3.5.1 Búsqueda y seguimiento

Es importante definir cómo es que la pelota va a ser detectada y validada, anteriormente se mostró como se identifica la pelota en una imagen, pero aún no se definen los parámetros para validar su detección. Como estamos trabajando con imágenes de 320x240 píxeles es claro que debemos definir un área mínima para saber si se ha detectado una pelota o no, esta área mínima que debería estar detectada es de 100 píxeles.

Al comenzar las pruebas se requiere que los robots localicen la pelota dentro de un terreno delimitado que equivale a un tercio del tamaño total de la cancha donde se llevarán a cabo

las pruebas, para esta detección se cuenta con la problemática de decidir una ruta por donde se deba buscar la pelota. Es claro que no toda el área debe ser recorrida por el robot ya que la detección de la pelota se hace por medio de la cámara lo cual nos brinda una detección a distancia. En la siguiente imagen podemos observar cómo se reduce en un principio el área de búsqueda.

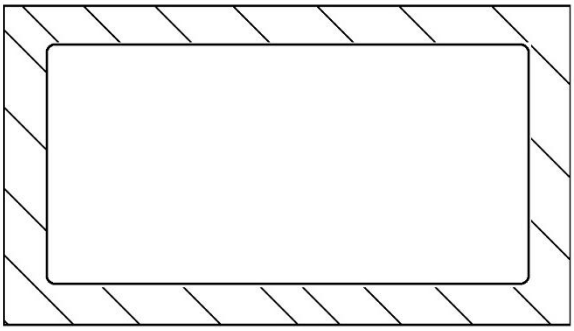

Figura 22 Área donde no puede comenzar inicialmente el robot

3.5.2 Visión mejorada con filtro polarizador

Se plantea realizar algunas pruebas para demostrar que existe una mejora con la incorporación del filtro polarizador a la cámara del robot para la detección de la pelota en exteriores.

El escenario de pruebas tiene la siguiente configuración, se tiene una cámara apuntando hacia la pelota roja y en medio se coloca el filtro polarizador con la polarización orientada de forma horizontal como se muestra en la siguiente imagen:

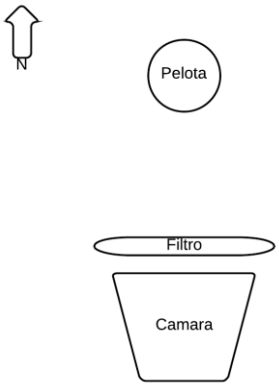

Figura 23 Escenario del experimento

Se puede observar que esta configuración se diseñó de tal manera que se pueda orientar en distintas direcciones para así poder observar cómo funciona el filtro en la dirección en el que es mayor el brillo, junto con distintos horarios para conocer en qué porcentaje se mejora la visión a cualquier hora del día.

3.6 Aprendizaje de golpeo de pelota

Como uno de los objetivos principales de esta tesis, aparte de lograr una coordinación entre robots, es que se adapte a terrenos exteriores, se propuso implementar una etapa de retroalimentación para informar al robot la distancia a la que llego el pase y así poder deducir en qué tipo de terreno exterior nos encontramos, rugoso o liso. De esta manera el siguiente pase se realizara con mayor precisión. Se tomó la arquitectura MURDOCH y se modificó en la parte final del protocolo de subasta donde el agente subastador realiza el monitoreo del progreso del robot que realiza la tarea, es ahí donde se lleva a cabo la retroalimentación de la posición en la que se encontró la pelota por parte del robot que realiza la tarea de recibir el pase, enviando esta información al robot que subasto la tarea.

Tomando en cuenta la textura rugosa de los exteriores es importante señalar que el valor que se irá actualizando es la distancia máxima a la que la pelota puede ser enviada cuando se le golpea con la mayor fuerza permitida por el motor del robot.

4. Pruebas y resultados

Resumen. Se describen las pruebas realizadas con los robots mandándose pases sobre terrenos exteriores, con y sin el filtro polarizado, mostrando las gráficas con el porcentaje de tareas realizadas correctamente. El diseño de pruebas es tal que el éxito de la tarea de los robots depende del éxito de las sub-tareas. Se hace una discusión con los trabajos relacionados para evidenciar el aporte de nuestro enfoque.

4.1 Diseño de pruebas a realizar

Dentro de las pruebas que se realizaron para este trabajo las podemos dividir en 2:

- Pruebas Preliminares
- Pruebas de coordinación

Las pruebas preliminares son las que se realizaron para implementar el reconocimiento de la pelota y su seguimiento en terrenos exteriores, son necesarias porque primero debemos lograr que los robots sean capaces de realizar las subtareas de manera correcta para que después puedan realizarlas de manera coordinada.

Las pruebas de coordinación son aquellas donde se puso a prueba la arquitectura MURDOCH adaptada para realizar la tarea principal de mandar pases con una pelota de este trabajo.

Primero se describirán las pruebas preliminares.

4.1.1 Pruebas preliminares

En la primer prueba se pretende que el robot se capaz de encontrar una pelota en exteriores y dentro de un terreno delimitado de 2.5 m de largo por 1.5 metros de ancho. La prueba inicia con la pelota y el robot en posiciones aleatorias dentro de un tercio del área del terreno. El robot sabe en qué posición se encuentra porque se le puso en la posición (0, 0), este calcula un punto inicial al azar dentro del primer tercio del terreno y se dirige hasta ahí manteniendo el cálculo de su posición en todo momento por medio de la odometría.

En esta etapa de inicialización se pudo observar que cuando el robot tardaba un poco en posicionarse en el ángulo calculado para desplazarse a su posición inicial, el archivo que

almacenaba todos los datos de la odometría crecía muy rápido aún sin tener muchos cambios en la posición estimada por la odometría. Es importante señalar que este robot cuenta con una memoria flash de tan solo 256 KB, y como nuestro archivo donde se almacenan los datos de la odometría tiene reservado 50 KB, hay que tener cuidado de solo almacenar valores significativos. El espacio reservado para el archivo nos permite almacenar 50KB, sabemos que se utilizarán entre 26 y de 45 caracteres por línea, lo cual es equivalente a un uso de entre 26 y 45 Bytes para guardar los cuatro valores deseados y que son:

- posición en x
- posición en y
- ángulo de dirección y
- tiempo

Lo cual nos deja con solo 1000-1700 líneas para almacenar los datos y suponiendo que un experimento dure 4 minutos, tenemos que guardar un dato cada 200 ms, con lo cual generamos:

$$5 * 240 = 1200\ líneas$$

Entonces se genera una optimización colocando un retardo de espera y una restricción para que solo se almacene el valor si hubo un cambio significativo en la orientación del robot, esto es, que se mueva. Si no se observa ningún cambio no se escribe en el archivo pero aun así cada 5 veces que no se detecte movimiento, se escribirá una actualización en el archivo con lo cual se consigue ahorrar espacio en esta arquitectura que tiene memoria limitada.

En la siguiente tabla se muestran algunos datos correspondientes a los archivos de la odometría con y sin la optimización del ahorro de espacio en el archivo.

Tabla 2 Comparación de las líneas escritas con y sin la consideración del ahorro de espacio

	Tiempo transcurrido (s)	Líneas escritas	Líneas por segundo
Sin optimización	12.52	1483	118.45
Con optimización	80.835	385	4.76

Como se puede observar es claro el beneficio ya que se podrán almacenar los datos por el tiempo necesario.

Una vez que el robot llega a su posición inicial se coloca la pelota y comienza la búsqueda. En la siguiente figura se muestra el diagrama de esta búsqueda.

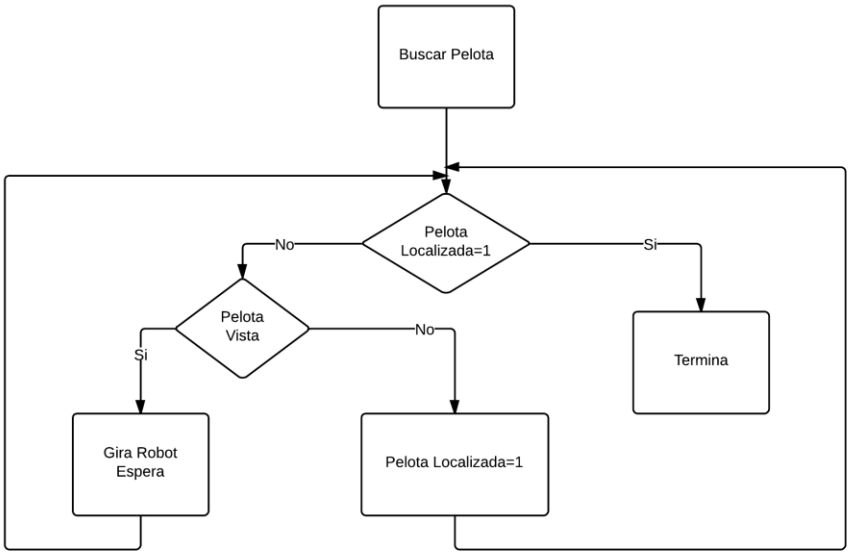

Figura 24 Función Buscar pelota

Para buscar la pelota el robot tiene que girar sobre sí mismo para así poder observar con la cámara el terreno que lo rodea y de esta forma poder encontrarla. Mientras no esté localizada se gira un poco y se espera, cuando la pelota entra en el campo de visión de la cámara del robot, este la marca como localizada y se finaliza su búsqueda.

En la segunda prueba se pretende lograr que el robot sea capaz de golpear la pelota una vez que la tiene localizada hacia un punto específico de tal manera que se avance en el eje y. Esta prueba comienza con la pelota enfrente del robot y este se tiene que acercar a ella de tal manera que llegue perfilado en la dirección de golpeo necesaria.

Primero se aproxima la posición de la pelota con respecto al robot. Como se tiene de frente solo hace falta sumarle una distancia aproximada en la dirección en la que se encuentra el robot. Contamos con los siguientes datos:

$$theta :: \text{ángulo de posición actual del robot}$$

$$pos_x :: \text{posición del robot en } x$$

$$pos_y :: \text{posición del robot en } y$$

$$distPel :: \text{distancia de la pelota}$$

La distancia aproximada a la pelota se calcula a partir de una regla de tres dependiendo de donde está ubicada dentro de las coordenadas en pixeles de la imagen de la cámara y las distancias mínima y máxima a la que puede ser vista como se puede observar en la siguiente figura.

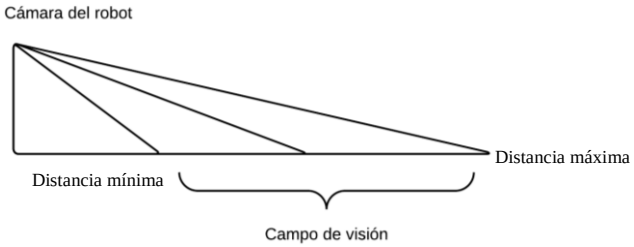

Figura 25 Campo de visión de la cámara de robot.

Una vez aproximada la distancia a la pelota podemos calcular su posición auxiliándonos de la siguiente figura:

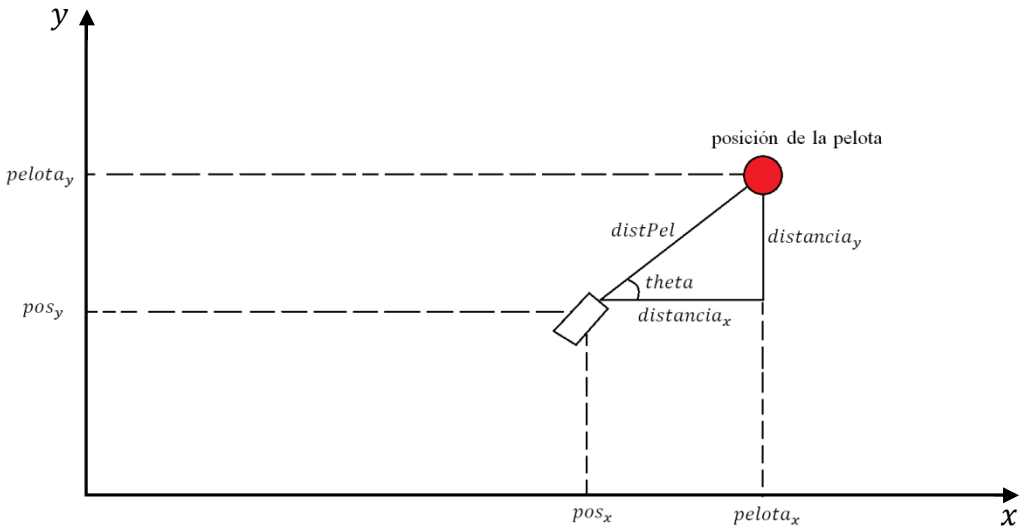

Figura 26 Cálculo de la posición de la pelota

Como se puede observar en la imagen para obtener la posición aproximada de la pelota es necesario calcular la $distancia_x$ y la $distancia_y$ y sumar estas distancias a las respectivas coordenadas del robot para así poder obtener $pelota_x$ y $pelota_y$.

$$pelota_x = pos_x + distPel * \cos(theta)$$

$$pelota_y = pos_y + distPel * \text{sen}(theta)$$

Conociendo la posición de la pelota calculamos la dirección del pase, esta dirección dependerá de la mitad del terreno en la que se encuentre la pelota. Para mantener a la pelota dentro de la cancha se mandarán pases hacia la mitad del terreno ($\frac{tam_x}{2}$) más una distancia fija, hasta un máximo, hacia el lado contrario a la ubicación de la pelota, ver figura 28.

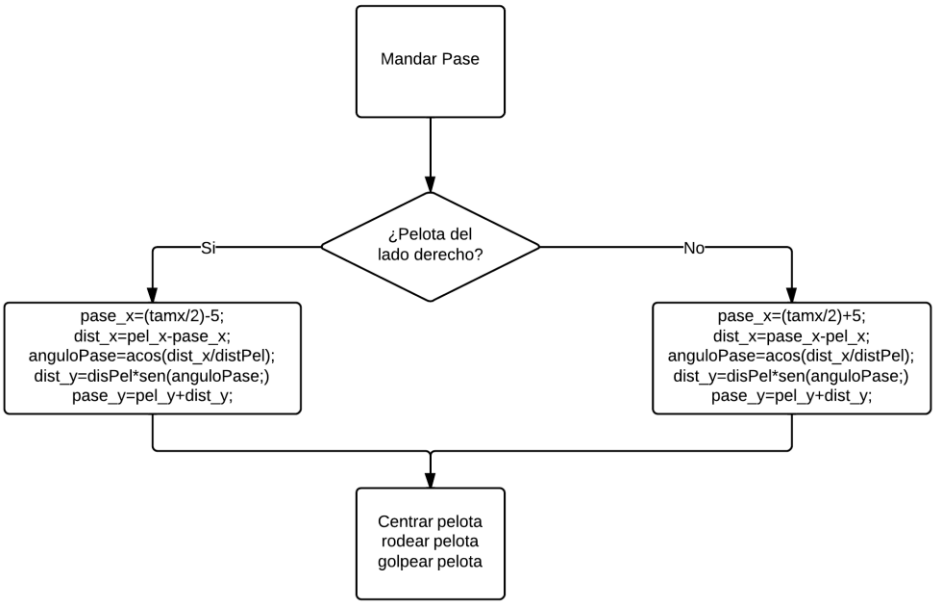

Figura 27 Función para mandar pase

Las ecuaciones descritas se obtuvieron partiendo del siguiente análisis con los datos conocidos:

$$pelota_x :: posición\ de\ la\ pelota\ en\ x$$

$$pelota_y :: posición\ de\ la\ pelota\ en\ y$$

$$disPel :: distancia\ que\ puede\ avanzar\ la\ pelota\ en\ el\ terreno\ actual$$

Para ayudarnos con una imagen y así poder ver de manera mucho más clara lo que necesitamos hacer tenemos la siguiente figura.

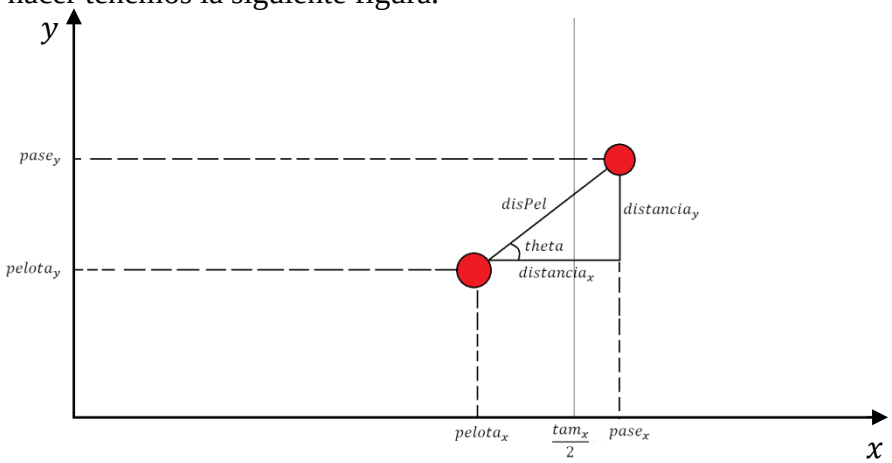

Figura 28 Cálculo del pase

Lo primero a calcular es el ángulo $theta$, que en este caso es el ángulo de la dirección del golpeo con el que se realiza el pase, de la siguiente manera: Si la pelota se encuentra del lado derecho del terreno, la posición del pase deseada en el eje coordenado x, será la de la mitad del terreno menos 5 cm, y si se encuentra del lado izquierdo será de la mitad del terreno más 5 cm. Con este dato podemos calcular el valor de la $distancia_x$ con:

$$pase_x = \frac{tam_x}{2} + 5$$

$$distancia_x = pase_x - pelota_x$$

Ahora que tenemos el valor del cateto adyacente ($distancia_x$) y el valor de la hipotenusa, el cual corresponde a la distancia de desplazamiento de la pelota sobre el terreno en que se encuentre. Podemos calcular el ángulo $theta$

$$distancia_x = distPel * \cos(theta)$$

$$\frac{distancia_x}{distPel} = \cos(theta)$$

$$theta = \text{acos}\left(\frac{distancia_x}{disPel}\right)$$

Una vez obtenido el ángulo de dirección del pase y la posición del pase en el eje x, podemos calcular la posición del pase en el eje y mediante las siguientes formula:

$$distancia_y = disPel * sen(theta)$$

$$pase_y = pelota_y + distancia_y$$

En la tercera prueba el robot R_1 recibirá un pase para lo cual debe ser capaz de seguir la pelota en movimiento sin perderla de vista: esta prueba comienza igual que la primera, con el robot R_1 en una posición aleatoria inicial pero ahora sin la restricción del primer tercio del terreno. El robot R_1 recibe la coordenada del pase estimada del robot R_2 y la señal de inicio, es en este momento cuando el robot R_2 golpea la pelota hacia la coordenada estimada del pase y el robot R_1 que la tiene que recibir comienza a avanzar. Si el robot R_1 llega al punto y la pelota no está ahí tendrá que buscarla.

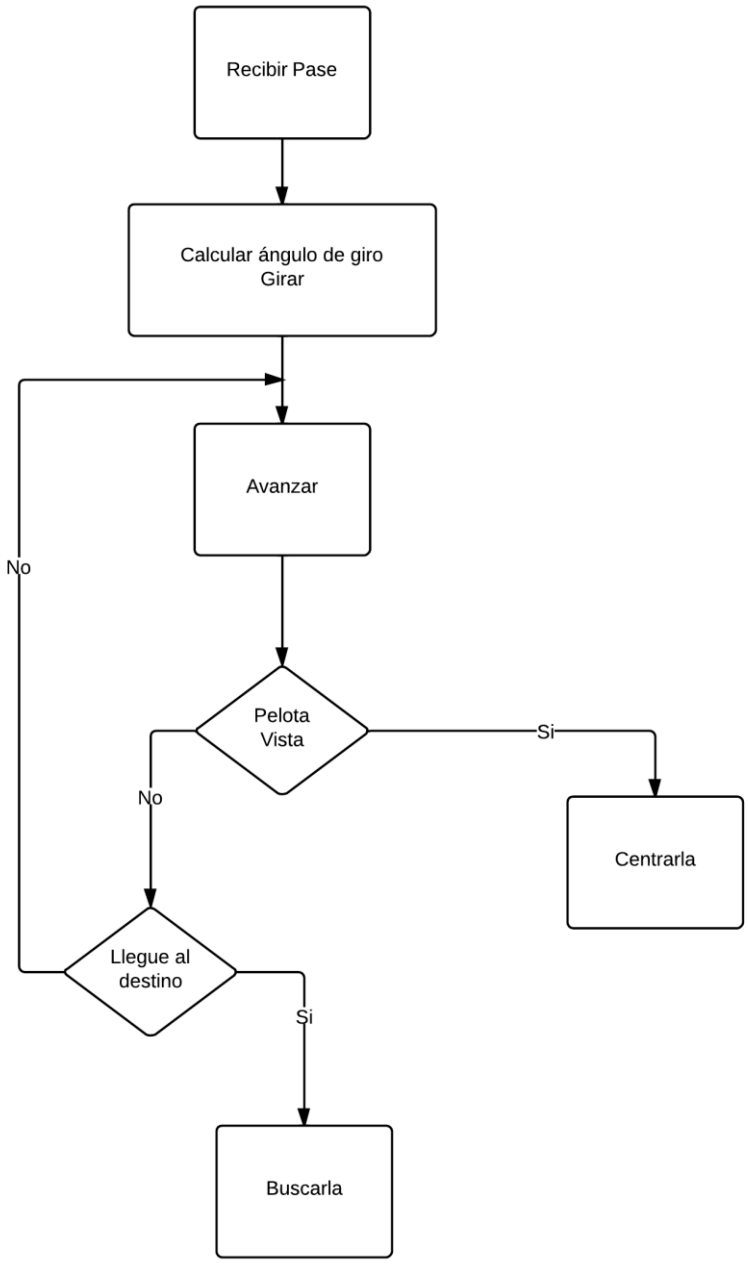

Figura 29 Diagrama de Recibir Pase.

4.1.2 Pruebas de mejora de visión con filtro polarizado

Se realizaron algunas pruebas para observar la mejora de visibilidad de la pelota haciendo uso del filtro polarizado para reducir el brillo provocado por el reflejo de la luz del sol sobre la superficie de la pelota.

Tomando en cuenta el escenario de pruebas de la figura 25, se procedió a tomar capturas de como la cámara observa a la pelota con y sin el filtro polarizado, en la siguiente figura se muestra una de estas capturas:

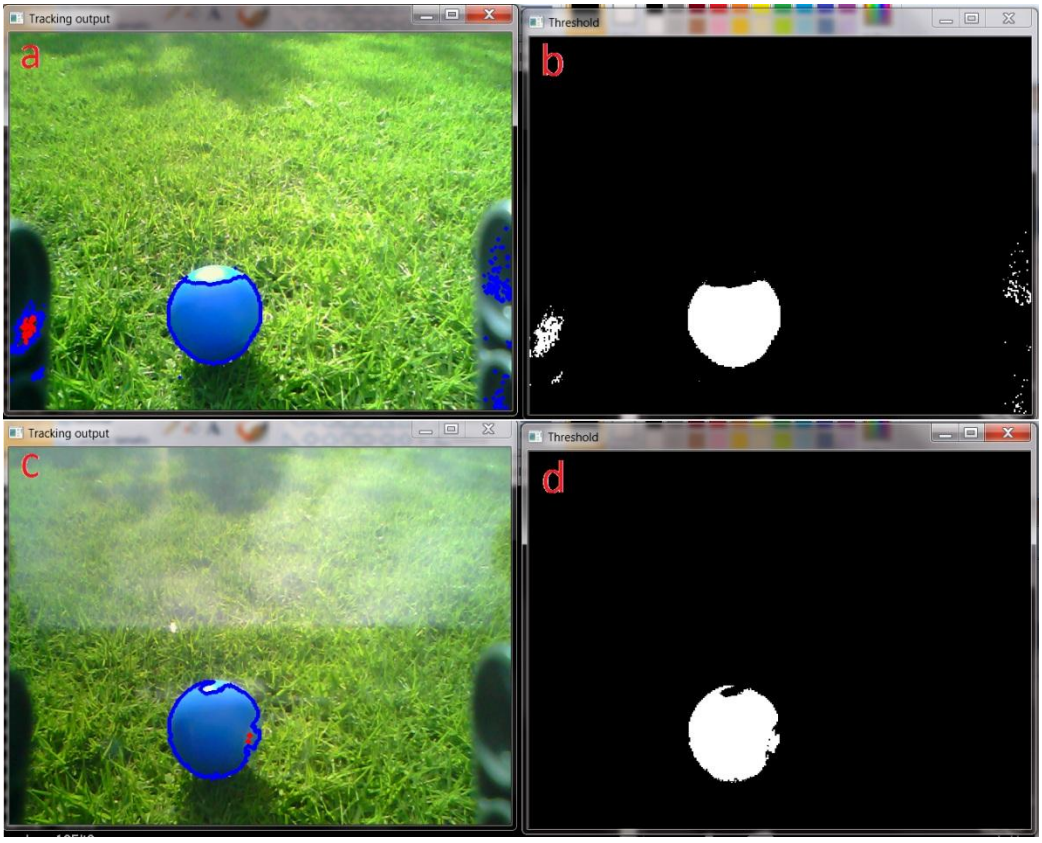

Figura 30 Captura de la mejora de visión con filtro polarizado. a. Imagen tomada sin filtro b. La parte detectada como pelota sin filtro c. Imagen con filtro d. La parte detectada como pelota con filtro

Como se puede observar en la figura 30.c con el uso del filtro se obtiene una mejora de la zona de la pelota donde sin el filtro se ve el brillo de la luz del sol.

En la siguiente tabla se muestran las mediciones de los pixeles de este experimento.

Tabla 3 Número de pixeles detectados con y sin filtro en los 4 puntos cardinales

Orientación	sin filtro	con filtro	% de Mejora
12:50 PM			
oeste	6902	7116	3.10055057
norte	5379	5422	0.79940509
este	5490	7374	34.3169399
sur	4545	5919	30.2310231
3:30 PM			
oeste	641	1816	183.307332
norte	6397	6286	-1.73518837
este	10255	10571	3.0814237
sur	6889	7447	8.09986936

Como podemos observar la mejora que el filtro logra para la percepción de la pelota es aceptable y en el caso en donde incluso se observa una disminución de la visibilidad de la pelota no es considerable ya que apenas es del 1.7%, también cabe señalar que no solo se logró una mejora en la detección de la pelota, sino que también se observó que con el uso del filtro se redujo el ruido detectado fuera de la pelota mejorando aún más la correcta localización de esta.

4.1.3 Pruebas de coordinación

Para la coordinación de los robots el programa se diseñó conforme el diagrama del módulo principal, ver figura 30, donde se conjuntan las subtareas que se tenían hasta el momento:

- La comunicación simulada por *broadcast*,
- la comunicación de los resultados del procesamiento visual y
- la generación de las subastas.

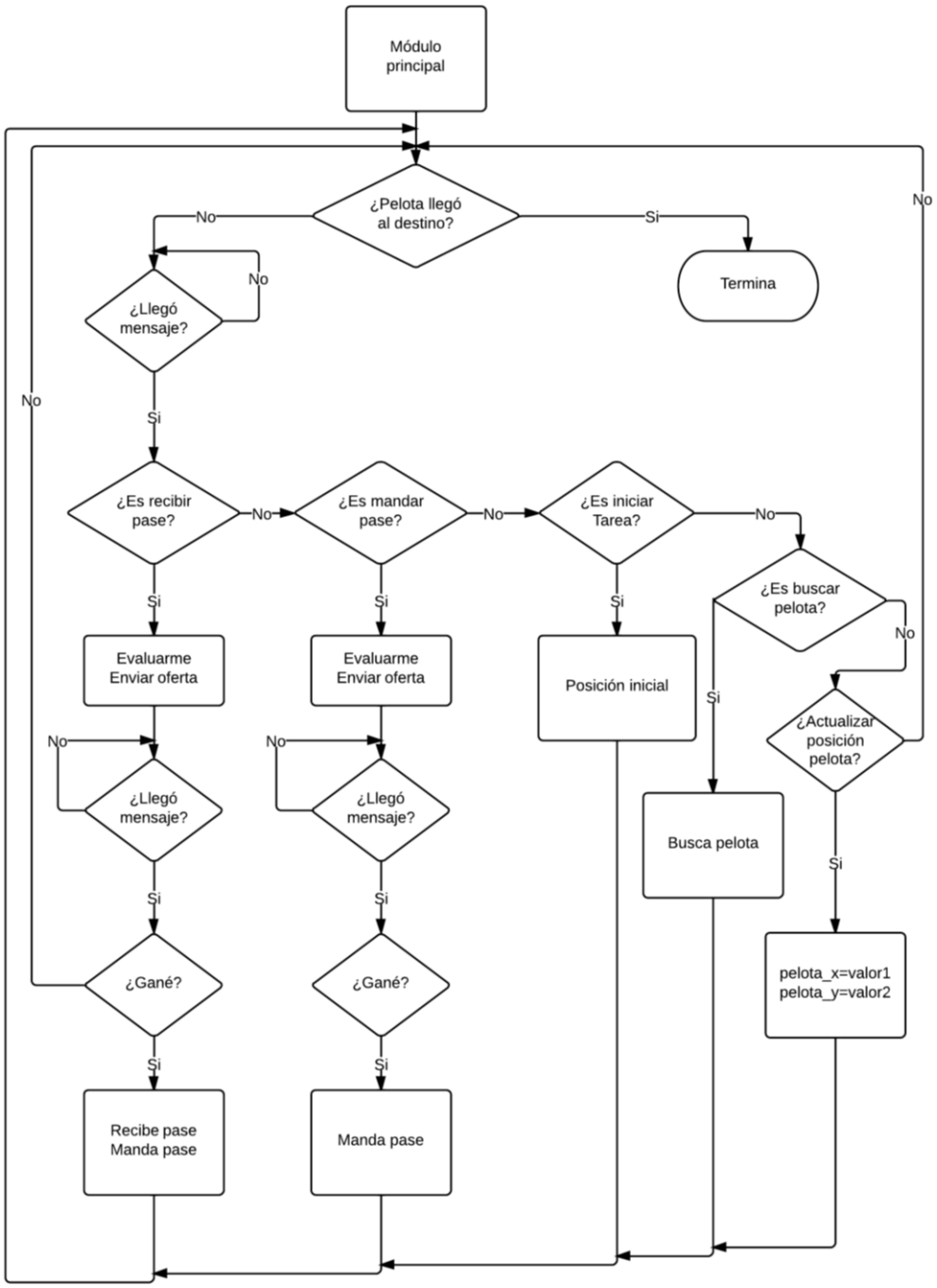

Figura 31 Diagrama del módulo principal del programa del robot

Se conoce por la literatura que el comportamiento coordinado entre robots se considera exitoso, cuando el porcentaje de veces en las que una tarea es llevada a cabo con éxito es mayor al 80% independientemente del número de veces que se realice la tarea. Para calcular el porcentaje de éxitos de la coordinación en exteriores así como también para comprobar si efectivamente funciona mejor la visión con la incorporación de un filtro polarizador o no; se propone realizar pruebas con y sin el filtro.

Prueba 1

Realizar repetidamente la tarea de llevar la pelota de un punto inicial a un punto final con el algoritmo de coordinación sin la retroalimentación para ver como mejora en el segundo y tercer experimentos.

Prueba 2

Realizar nuevamente un número de repeticiones de la tarea de coordinación pero esta vez con la retroalimentación para la mejora de la coordinación y medir en cuantos experimentos se llega a un valor casi constante que ya no cambia más allá del 10%.

Prueba 3

Realizar la tarea el mismo número de veces que en la prueba uno para poder comparar el número de éxitos de la realización correcta de la tarea con y sin el aprendizaje adquirido en la prueba 2.

Pruebas 4, 5 y 6

Serán igual a las descritas arriba pero ahora con la incorporación del filtro polarizador para ver como mejora la detección y el seguimiento de la pelota a la hora de que el robot reciba el pase.

4.2 Resultados obtenidos

Se consiguió realizar el seguimiento de la pelota con el robot Bioloid. Sin embargo, como este robot no es capaz de proporcionarnos una medida precisa para hacer el cálculo de la odometría, se tuvo que cambiar por el robot NXT.

Se logró publicar un artículo basado en la novedosa aplicación de un filtro polarizador para la mejora de la visión en exteriores.

Se logró implementar el cálculo de la odometría en el robot NXT con una optimización en el archivo que almacena estos datos para robots con memoria limitada.

Se propuso una modificación a una arquitectura de coordinación, para poder usarla en exteriores.

4.3 Discusión de resultados

Pese a no haber podido llevar a cabo las pruebas para observar la coordinación de los robots a causa de una falla técnica explicada en el apéndice D. Se cuentan con resultados muy favorables y queda como trabajo a futuro la implementación de este programa a un dispositivo móvil, con lo cual ya no se tendría que depender de la computadora para el procesamiento visual y así se eliminarían las fallas técnicas que causaron las cámaras, además se podría realizar la comunicación de tipo *broadcast* por medio de la comunicación inalámbrica por WiFi con la que cuentan estos dispositivos.

A continuación se muestra una tabla donde se examinan las características implementadas en cada uno de los robots utilizados:

Tabla 4 Comparativa de los resultado obtenidos en las arquitecturas de robot utilizadas.

	Robot Bioloid	Robot NXT
Seguimiento de un objeto	Si	Si(sin probar)
Comunicación *broadcast*	Si	Si (simulada por bluetooth)
Cálculo de odometría	No	Si
Coordinación	No	Si(no probada)

5. Conclusiones

Resumen. En este capítulo se discuten los resultados obtenidos analizando el porqué de los mismos así como proponiendo ideas que quedan como trabajo a futuro para mejorar el comportamiento de la coordinación entre robots en exteriores.

La coordinación entre robot es un problema muy importante ya que el objetivo de su desarrollo es el de evitarnos a los seres humanos los riesgos que se corren al realizar tareas peligrosas. En la actualidad diferentes arquitecturas de coordinación se han probado en interiores y es por esto que nuestra modificación aquí propuesta a una de estas arquitecturas

5.1 Aportaciones

- Se logró adaptar una arquitectura de coordinación, diseñada para interiores, para que pudiera ser usada en exteriores con la inclusión de la etapa de retroalimentación, la cual se realiza en el paso previo de la evaluación. Gracias a esta etapa añadida, el robot puede "percibir" el tipo de terreno donde se encuentra.
- Se logró publicar un artículo, en el cuál se propone el uso de los filtros polarizadores para la mejora de la visión de robots en exteriores durante un seguimiento de pelota.
- Se diseñó la solución individual a cada subtarea y también se diseñó y programó el código del robot que conjunta las subtareas, la comunicación de tipo *broadcast* simulada con bluetooth y la comunicación del procesamiento visual.

5.2 Líneas de investigación abiertas

- Migración del código a un dispositivo móvil con cámara.
- Mejora del algoritmo de búsqueda para poderse aplicar a terrenos de mayor tamaño y aprovechando el número de robots con los que se cuente.
- Probar el impacto que tiene la mejora visual con el filtro polarizador dentro de la realización de la tarea coordinada.
- Optimización del código de detección de la pelota por medio de la detección de circunferencia para mejorar la posición central de la pelota.

Apéndices

En esta sección se presentan: el código utilizado para la detección del color y su seguimiento, el código para la coordinación de los robots y algunas fotos de los experimentos realizados en el exterior.

A. Código para vision y seguimiento de la pelota en c

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <conio.h>
#include "nxt.h"
#define btcomm 4
#include "main.h"

using namespace std;
#pragma warning(disable : 4995)
#pragma warning(disable : 4996)
#pragma region Constantes
// User message for zigbee communication
#define         WM_ZIGBEE_RECEIVE               WM_USER+1
#define     START               ( 'S' )
#define         END_PROC( 'E' )

#define detecbola       100                     //Cuenta minima de pixeles rojos para detectar una
pelota
#define ruidbola40                      //Separacion maxima entre pixeles pertenecientes a la pelota

// Default values to display for received and sent data
WORD SendData = '_';            // PC -> CM-5 data
WORD RcvData = '_';                     // CM-5 -> PC data
WORD GIRA = 'V';

clock_t last_data, actual_time; // Estructuras para calcular el tiempo
clock_t beg_exp,fin_exp;
#pragma endregion
//Variables para conexion por bluethooth con NXT
Connection *connection = new Bluetooth();
Brick *nxt = new Brick(connection);

HWND g_hWnd;
double v=0,sl=0;
double roughness=0;

// Variables globales para aplicación del algoritmo
bool working;

int LoadHyperVector(void);
int LoadingThreadID=1;
int VectorsLoaded;
int countcolor;
int colorsino=0;
int stablished=0;
int detectada=0;
int max_x;
HANDLE Handle_Loading_Thread =0;
```

```
DWORD WINAPI LoadHyperVector( LPVOID lpParam );
string program_name = "buzon2.rxe";


#pragma region Definiciones_Metodos

// Funciones prototipo para la parte de vision
void ImageProcess(unsigned char* pImageData,DWORD dwDataSize);
void RGB24toGray( unsigned char* pGray, unsigned char* pRGB24, int width, int height );
void Sobel_Edge( unsigned char* pResult, unsigned char* pOrigin, int width, int height );
void Soft3( unsigned char* pResult, unsigned char* pOrigin, int width, int height );
void Soft5( unsigned char* pResult, unsigned char* pOrigin, int width, int height );
void Circle( unsigned char* pResult, unsigned char* pOrigin, int width, int height );
bool countRoughness(unsigned char pixel, int x, int y);


void RotateRGB90( unsigned char* pHSI, unsigned char* pRGB, int width, int height );
void RotRGBHSI( unsigned char* pHSI, unsigned char* pRGB, int width, int height );
void RGB24toHSI( unsigned char* pHSI, unsigned char* pRGB, int width, int height );
void FindRedFromHSI( unsigned char* pBinary, int range, unsigned char* pHSI, int width, int height
);
void FindGreenFromHSI( unsigned char* pBinary, int range, unsigned char* pHSI, int width, int height
);
void FindBlueFromHSI( unsigned char* pBinary, int range, unsigned char* pHSI, int width, int height
);
void GetCenterPos( int* pX, int* pY, unsigned char* pBinary, int width, int height );
void Tracking_PanTilt( int diff_pan, int diff_tilt, HDC hdc );
void CountColor(int* max_x, int* countcolor, unsigned char* pBinary, int width, int height);
//void Gira();

unsigned char* pGrayImage = NULL;
unsigned char* pSoftImage = NULL;
unsigned char* pEdgeImage = NULL;

unsigned char* pRGB = NULL;
unsigned char* pHSI = NULL;
unsigned char* pBinary = NULL;
unsigned char* pCBinary = NULL;


bool Vision_init(HWND hWnd);
bool ZigBee_Init(HWND hWnd);
bool Comm_init(HWND hWnd);

void StatusDisplay(HDC hdc);
#pragma endregion

LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE g_hInst;
LPSTR lpszClass="Sigue Pelota";

// Win32 main
int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
                  ,LPSTR lpszCmdParam,int nCmdShow)
{
        MSG Message;
        WNDCLASS WndClass;
        g_hInst=hInstance;

        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.hInstance=hInstance;
        WndClass.lpfnWndProc=(WNDPROC)WndProc;
        WndClass.lpszClassName=lpszClass;
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;
```

```
	RegisterClass(&WndClass);

	g_hWnd=CreateWindow(lpszClass,lpszClass,WS_OVERLAPPEDWINDOW,
		  CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,
		  NULL,(HMENU)NULL,hInstance,NULL);
	ShowWindow(g_hWnd,nCmdShow);

	while(GetMessage(&Message,0,0,0))
	{
		TranslateMessage(&Message);
		DispatchMessage(&Message);
	}
	return Message.wParam;
}

// Win32 message procedure
LRESULT CALLBACK WndProc(HWND hWnd,UINT iMessage,WPARAM wParam,LPARAM lParam)
{
	HDC hdc;

	switch(iMessage)
	{
	case WM_PAINT:
		hdc = GetDC( g_hWnd );
		WmPaintCallBack(hdc);
		StatusDisplay(hdc);
		InvalidateRect( g_hWnd, NULL, FALSE ); // Draw result
		ReleaseDC( g_hWnd, hdc );
		return 0;
	case WM_CHAR:

		SendData = (WORD)wParam;
		/*if (SendData=='D')
			if(working== false)
				working=true;
		WmCharCallBack(hWnd, SendData);
		*/

		/*if(SendData==GIRA)
			Gira();
		*/
		return 0;
	case WM_CREATE:
		//initFuzzy();
		Vision_init(hWnd);
		//ZigBee_Init(hWnd);
		Comm_init(hWnd);
		WmCreateCallback();
		working= false;
		return 0;
	case WM_ZIGBEE_RECEIVE: // Zigbee data arrived message
		RcvData = (WORD)wParam; // WPARAM is received data
		if(RcvData==START && working == false){
			ProcessStart(hWnd);
			working = true;
		}else if(RcvData == END_PROC && working == true){
			ProcessEnd(hWnd);
			working = false;
		}else
			WmZigBeeReceiveCallBack(RcvData);

		InvalidateRect( hWnd, NULL, TRUE ); // Draw result
		return 0;
	case WM_DESTROY:
		vision_close();
		//zigbee_close();
		delete pGrayImage;
		delete pEdgeImage;
		delete pSoftImage;
```

```
                PostQuitMessage(0);
                return 0;
        }

        return(DefWindowProc(hWnd,iMessage,wParam,lParam));
}

int times=0;


void ImageProcess( unsigned char* pImageData, DWORD dwDataSize )
{
        HDC hdc;
        HBITMAP hBitmap;

        VISION_STATE VisionState; // Monitoring data of Vision library
        char strText[128];
        int velocity,
                radius;
        int center_x, center_y,
                circle_x, circle_y,
                object_x, object_y;

        // Display state
        VisionState = vision_get_state();

        // Initialize

        center_x = VisionState.width/2;
        center_y = VisionState.height/2;



        ////////////////////// Image Process //////////////////////////
        RGB24toGray( pGrayImage, pImageData, VisionState.width, VisionState.height );
        Soft5(pSoftImage, pGrayImage, VisionState.width, VisionState.height);
        Sobel_Edge( pEdgeImage, pSoftImage, VisionState.width, VisionState.height );

        //RotateRGB90( pHSI, pImageData, VisionState.width, VisionState.height );
        RGB24toHSI( pHSI, pImageData, VisionState.width, VisionState.height );
        //RotRGBHSI( pHSI, pImageData, VisionState.width, VisionState.height );
        FindRedFromHSI( pBinary, 60, pHSI, VisionState.width, VisionState.height );

        //FindGreenFromHSI( pBinary, 60, pHSI, VisionState.width, VisionState.height );
        //FindBlueFromHSI( pBinary, 60, pHSI, VisionState.width, VisionState.height );
        Soft5(pSoftImage, pBinary, VisionState.width, VisionState.height);


        //if(colorsino==10)
        //{
                //CountColor(&countcolor, pBinary, VisionState.width, VisionState.height);
                CountColor(&max_x, &countcolor, pSoftImage, VisionState.width, VisionState.height);
        //      colorsino=0;
        //}
        //else
        //      colorsino++;

                if (countcolor>detecbola)
                        detectada=1;
                else
                        detectada=0;

        // Ready
        hdc = GetDC( g_hWnd );

        //GetCenterPos( &object_x, &object_y, pBinary, VisionState.width, VisionState.height );
        GetCenterPos( &object_x, &object_y, pSoftImage, VisionState.width, VisionState.height );
```

```
        //#############################################################################################
#################################################
    //GetCircle( &circle_x, &circle_y, &radius, pSoftImage, VisionState.width, VisionState.height );

        if (stablished)
        Tracking_PanTilt( center_x - object_x, center_y - object_y, hdc );

        // Draw DDB of converted image
//      hBitmap = vision_GraytoDDB( hdc, pEdgeImage );
        //hBitmap = vision_GraytoDDB( hdc, pSoftImage );
        hBitmap = vision_RGB24toDDB( hdc, pImageData );
        vision_DrawDDB( hdc, 10, 10, 450, 320, hBitmap );

//      hBitmap = vision_GraytoDDB( hdc, pBinary );
        hBitmap = vision_GraytoDDB( hdc, pSoftImage );
        vision_DrawDDB( hdc, 460, 10, 450, 320, hBitmap );

        sprintf( strText,"Roughness: %5.2f   ",roughness);
        TextOut( hdc, 200, 380, strText, strlen(strText) );
        sprintf( strText,"Pelota en x: %d   ",object_x);
        TextOut( hdc, 200, 425, strText, strlen(strText) );
        sprintf( strText,"Pelota en y: %d   ",object_y);
        TextOut( hdc, 200, 470, strText, strlen(strText) );
        sprintf( strText,"Pixeles rojos: %d   ",countcolor);
        TextOut( hdc, 200, 515, strText, strlen(strText) );
        if(detectada)
        {
                sprintf( strText,"Pelota detectada con ancho    %d", max_x);
                TextOut( hdc, 200, 555, strText, strlen(strText) );
        }
        else
        {
                sprintf( strText,"Pelota no detectada con ancho %d", max_x);
                TextOut( hdc, 200, 555, strText, strlen(strText) );
        }


        velocity = WmImageProcessingCallBack(roughness);
        switch(velocity){
                case 1: strcpy( strText, "Speed: Fast      ");  break;
                case 2: strcpy( strText, "Speed: High      ");  break;
                case 3: strcpy( strText, "Speed: Medium    ");  break;
                case 4: strcpy( strText, "Speed: Slow      ");  break;
        }
        TextOut( hdc, 200, 395, strText, strlen(strText) );

        sprintf( strText, "Slope: %2.2f",sl);
        TextOut( hdc, 200, 415, strText, strlen(strText) );

        sprintf( strText, "Width: %d", VisionState.width );
        TextOut( hdc, 10, 380, strText, strlen(strText) );                      // Print image
width
        sprintf( strText, "Heigth: %d", VisionState.height );
        TextOut( hdc, 10, 395, strText, strlen(strText) );                      // Print image
height
        sprintf( strText, "Fps: %d  ", VisionState.fps );
        TextOut( hdc, 10, 410, strText, strlen(strText) );                      // Print fps
        sprintf( strText, "Frame time: %.2fms  ", VisionState.capTime_ms );
        TextOut( hdc, 10, 425, strText, strlen(strText) );                      // Print time of
frame interval
        sprintf( strText, "Process time: %.2fms  ", VisionState.procTime_ms );
        TextOut( hdc, 10, 440, strText, strlen(strText) );                      // Print time of
user's process

        InvalidateRect( g_hWnd, NULL, FALSE );                                          // Draw
result

        // Release
        DeleteObject( hBitmap );
```

```
        ReleaseDC( g_hWnd, hdc );
}

#pragma region MetodoWnProc

//actual_time = clock();
//last_data = clock();        // Actualiza reloj desde la ultima transmision con el robot

///////////////// Initialize Vision library ////////////////////////
bool Vision_init(HWND hWnd){
        VISION_PARAM VisionParam; // Parameters of Vision library

        VisionParam.width = 320; // Desired image width
        VisionParam.height = 240; // Desired image height
        VisionParam.pProcessFunc = ImageProcess; // User's function

        if( vision_open( &VisionParam ) == false )
        {
                MessageBox( hWnd, "Fail to open Vision!", "Error" , MB_OK );
                PostQuitMessage(0);
        }
        // Create image data
        pGrayImage = new unsigned char[VisionParam.width * VisionParam.height];
        pEdgeImage = new unsigned char[VisionParam.width * VisionParam.height];
        pSoftImage = new unsigned char[VisionParam.width * VisionParam.height];
        pRGB       = new unsigned char[(VisionParam.width * VisionParam.height)*3];
        pHSI       = new unsigned char[(VisionParam.width * VisionParam.height)*3];
        pBinary    = new unsigned char[VisionParam.width * VisionParam.height];


        if( vision_start() == false )
        {
                MessageBox( hWnd, "Fail to start Vision!", "Error" , MB_OK );
        }
        return true;
}
///////////////// Initialize Zigbee library ////////////////////////
bool ZigBee_Init(HWND hWnd){
        if( zigbee_open( PORTNAME, hWnd, WM_ZIGBEE_RECEIVE ) == false )
        {
                MessageBox( hWnd, "Falló al abrir el Zigbee!", "Error" , MB_OK );
                PostQuitMessage(0);
        }
        if( zigbee_clear() == false )
        {
                MessageBox( hWnd, "Falló al limpiar el buffer del Zigbee!", "Error" , MB_OK );
        }
        return true;
}

bool Comm_init(HWND hWnd){

  try{
    //cout << "Try to connect to the NXT" << endl;
    connection->connect(btcomm);
    nxt->start_program(program_name,true);
        //cout << "Connected" << endl;
    //cout << "Read and write a message to the mailbox system on the NXT - hit any key to end" <<
endl;
    }
    //connection->disconnect();
   catch (Nxt_exception& e){
    //some error occurred - print it out
    cout << e.what() << endl;
    cout << "error code: " << e.error_code() << endl;
        MessageBox( hWnd, "Error al abrir el bluetooth", "Error" , MB_OK );
    cout << "error type: " << e.error_type() << endl;
    cout << e.who() << endl;
    connection->disconnect();
```

```
        PostQuitMessage(0);
  }
   stablished=1;
  return true;
}

void StatusDisplay(HDC hdc){

        char strText[128];

        sprintf( strText, "Vision: OK");
        TextOut( hdc,10,340 , strText, strlen(strText) );

        sprintf( strText, "NXT: OK");
        TextOut( hdc,10,355 , strText, strlen(strText) );

        sprintf( strText, "Data Sent:");
        TextOut( hdc,200,340 , strText, strlen(strText) );
        sprintf( strText, "%c (%d)", SendData ,SendData);
        TextOut( hdc, 310,340, strText, strlen(strText) );

        sprintf( strText, "Data Received:");
        TextOut( hdc,200,355 , strText, strlen(strText) );
        sprintf( strText, "%c (%d)", RcvData ,RcvData);
        TextOut( hdc, 310,355, strText, strlen(strText) );

        sprintf( strText, "Start Time:");
        TextOut( hdc,10,465 , strText, strlen(strText) );

        sprintf( strText, "End Time:");
        TextOut( hdc,10,480 , strText, strlen(strText) );

        sprintf( strText, "Elapsed Time:");
        TextOut( hdc,10,495 , strText, strlen(strText) );

        sprintf( strText, "Current X:");
        TextOut( hdc,500,340 , strText, strlen(strText) );

        sprintf( strText, "Current Y:");
        TextOut( hdc,500,355 , strText, strlen(strText) );

        sprintf( strText, "Status:");
        TextOut( hdc,500,370 , strText, strlen(strText) );

        if(working== false)
                sprintf( strText, "Press D button to Start!");
        else
                sprintf( strText, "Mobile robot is following ball: Stand by.");
        TextOut( hdc,500,385 , strText, strlen(strText) );                      // Print image
width

}


#pragma region ConvertirRGB-Gris
void RGB24toGray( unsigned char* pGray, unsigned char* pRGB24, int width, int height )
{
        int x, y, index;
        unsigned char r, g, b, gray;

        // Image Process( RGB -> Gray )
        for( y=0; y<height; y++ )
        {
                for( x=0; x<width; x++ )
                {
                        index = y*width + x;

                        b = pRGB24[3*index];
                        g = pRGB24[3*index+1];
```

```
                        r = pRGB24[3*index+2];

                        gray = (unsigned char)((r + g + b) / 3);
                        pGray[index] = gray;
                }
        }
}
#pragma endregion



void Sobel_Edge( unsigned char* pResult, unsigned char* pOrigin, int width, int height )
{
        int x, y, i, j, index;
        int sum_x, sum_y, sum, origin;
        double tempRoughness;
        int gx_mask[3][3] = {-1, 0, 1, -2, 0, 2, -1, 0, 1}; // Sobel gx mask
        int gy_mask[3][3] = {1, 2, 1, 0, 0, 0, -1, -2, -1}; // Sobel gy mask
        int* pTempImage;
        int min, max, newValue;
        float constVal1, constVal2;

        pTempImage = new int[width*height];

        // Edge detection by Sobel mask
        for( y=0; y<height; y++ ){
                for( x=0; x<width; x++ ){
                        sum = 0;

                        if( (x != 0 && x != width-1) && (y != 0 && y != height-1) ){
                                sum_x = 0;
                                sum_y = 0;
                                for( j=-1; j<=1; j++ ){
                                        for( i=-1; i<=1; i++ ){
                                                origin = (int)pOrigin[(y+j) * width + (x+i)];
                                                sum_x += origin * gx_mask[j+1][i+1];
                                                sum_y += origin * gy_mask[j+1][i+1];
                                        }
                                }
                                sum = abs(sum_x) + abs(sum_y);
                                sum = max(0, min(sum, 255));
                        }

                        index = y*width + x;
                        pTempImage[index] = sum;
                }
        }

        // Find min, max value
        min = (int)10e10;
        max = (int)-10e10;
        for( y=0; y<height; y++ ){
                for( x=0; x<width; x++ ){
                        index = y*width + x;
                        if( pTempImage[index] < min )
                                min = pTempImage[index];
                        if( pTempImage[index] > max )
                                max = pTempImage[index];
                }
        }

        // Normalize level
        constVal1 = (float)(255.0 / (max-min));
        constVal2 = (float)(255.0*min / (max-min));

        tempRoughness = 0;
        for( y=0; y<height; y++ ){
                for( x=0; x<width; x++ ){
                        index = y*width + x;
```

```
                        newValue = (int)(constVal1 * (float)pTempImage[index] + constVal2);
                        pResult[index] = (unsigned char)newValue;
                        //UMBRAL  para hacer una funcion binaria
                        if(pResult[index]>60){
                                pResult[index] = 0xFF;
                                tempRoughness++;
                        }else{
                                pResult[index] = 0x00;
                        }
                }
        }
        roughness = (tempRoughness-3000)/1000;
        delete pTempImage;
}

bool countRoughness(unsigned char pixel, int x, int y){
        if(pixel>240)
                return true;
        return false;
}

void Soft3( unsigned char* pResult, unsigned char* pOrigin, int width, int height )
{
        int x, y, i, j, index;
        int operand, origin;
        int mask[3][3] = {1, 1, 1, 1, 1, 1, 1, 1, 1};

        for( y=0; y<height; y++ ){
                for( x=0; x<width; x++ ){
                        operand = 0;
                        if( (x != 0 && x != width-1) && (y != 0 && y != height-1) ){
                                for( j=-1; j<=1; j++ ){
                                        for( i=-1; i<=1; i++ ){
                                                origin = (int)pOrigin[(y+j) * width + (x+i)];
                                                operand += origin * mask[j+1][i+1];
                                        }
                                }
                        }
                        index = y*width + x;
                        pResult[index] = operand/9;
                }
        }
}

void Soft5( unsigned char* pResult, unsigned char* pOrigin, int width, int height )
{
        int x, y, i, j, index;
        int operand, origin;
        int mask[5][5];

        for (i=0;i<5;i++)
                for(j=0;j<5;j++)
                        mask[i][j]=1;

        for( y=0; y<height; y++ ){
                for( x=0; x<width; x++ ){
                        operand = 0;
                        if( (x > 1 && x < width-2) && (y > 1 && y < height-2) ){
                                for( j=-2; j<=2; j++ ){
                                        for( i=-2; i<=2; i++ ){
                                                origin = (int) pOrigin[(y+j) * width + (x+i)];
                                                operand += origin * mask[j+2][i+2];
                                        }
                                }
                        }
                        index = y*width + x;
                        pResult[index] = operand/25;
                }
        }
```

```
}

void RotateRGB90( unsigned char* pHSI, unsigned char* pRGB, int width, int height )
{
        float R,G,B, H,S,I;
        float Min, Angle;
        int iH;
        int x, y, index;


        for( y=0; y<height; y++ )
        {
                for( x=0; x<width; x++ )
                {
                        index = width*y + x;

                        B = (float)pRGB[3*index]/255.0f;
                        G = (float)pRGB[3*index+1]/255.0f;
                        R = (float)pRGB[3*index+2]/255.0f;

                        Min = min(min(R,B),G);

                        I = (R + G + B)/3.0f;

                        if( (R == G) && (G == B) )
            {
                                S = 0.0f;
                                H = 0.0f;
                        }
                        else
                        {
                                S = 1.0f - (3.0f / (R + G + B)) * Min;

                                Angle = (float)((R - G*0.5f - B*0.5f) / sqrt((R-G) * (R-G) + (R-B) *
(G-B)));

                                H = (float)acos( Angle ) * 57.29577951f;
                                if( B > G )
                                        H = 360.0f - H;
                        }

                        iH = (int)(H * 255.0f / 360.0f);
                        pHSI[3*height*width-(3*index)-3] = (unsigned char)iH;
            pHSI[3*height*width-(3*index)-2] = (unsigned char)(S * 255.0f);
            pHSI[3*height*width-(3*index)-1] = (unsigned char)(I * 255.0f);
                }
        }
}

void RotRGBHSI( unsigned char* pHSI, unsigned char* pRGB, int width, int height )
{
        float R,G,B, r,g,b, H,S,I, h,s,i;
        float Min, Angle;
        int iH;
        int x, y, index;


        for( y=0; y<height; y++ )
        {
                for( x=0; x<width; x++ )
                {
                        index = width*y + x;

                        B = (float)pRGB[3*index];
                        G = (float)pRGB[3*index+1];
                        R = (float)pRGB[3*index+2];

                        r=R/(R+G+B);
                        g=G/(R+G+B);
                        b=B/(R+G+B);
```

```
                        Min = min(min(r,b),g);

                        if (b<=g)
                        {
                          //Angle = (float)((R - G*0.5f - B*0.5f) / sqrt((R-G) * (R-G) + (R-B) * (G-
B)));
                                Angle = (float)(((r-g)+(r-b))*0.5f/sqrt(((r-g)*(r-g))+(r-b)*(g-b)));
                                h=(float)acos(Angle);
                        }
                        else
                        {
                                Angle = (float)(((r-g)+(r-b))*0.5f/sqrt(((r-g)*(r-g))+(r-b)*(g-b)));
                                h=(float) (PI*2.0f - acos(Angle));
                        }

                        s=1-3*Min;
                        i=(R+G+B)/(3.0f*255);

                        H=h*(180*PI);
                        S=s*100;
                        I=i*255;

                        pHSI[3*height*width-(3*index)-3] = (unsigned char)H;
            pHSI[3*height*width-(3*index)-2] = (unsigned char)S;
            pHSI[3*height*width-(3*index)-1] = (unsigned char)I;
                }
        }
}


void RGB24toHSI( unsigned char* pHSI, unsigned char* pRGB, int width, int height )
{
        float R,G,B, H,S,I;
        float Min, Angle;
        int iH;
        int x, y, index;


        for( y=0; y<height; y++ )
        {
                for( x=0; x<width; x++ )
                {
                        index = width*y + x;

                        B = (float)pRGB[3*index]/255.0f;
                        G = (float)pRGB[3*index+1]/255.0f;
                        R = (float)pRGB[3*index+2]/255.0f;

                        Min = min(min(R,B),G);

                        I = (R + G + B)/3.0f;

                        if( (R == G) && (G == B) )
            {
                                S = 0.0f;
                                H = 0.0f;
                        }
                        else
                        {
                                S = 1.0f - (3.0f / (R + G + B)) * Min;

                                Angle = (float)((R - G*0.5f - B*0.5f) / sqrt((R-G) * (R-G) + (R-B) *
(G-B)));
                                H = (float)acos( Angle ) * 57.29577951f;
                                if( B > G )
                                        H = 360.0f - H;
                        }
```

```
                        iH = (int)(H * 255.0f / 360.0f);
                        pHSI[3*index] = (unsigned char)iH;
                pHSI[3*index+1] = (unsigned char)(S * 255.0f);
                pHSI[3*index+2] = (unsigned char)(I * 255.0f);
                    }
            }
}

void FindRedFromHSI( unsigned char* pBinary, int range, unsigned char* pHSI, int width, int height )
{
        int x, y, index;
        int H, S;
        int radius;

        radius = range / 2;
        for( y=0; y<height; y++ )
        {
                for( x=0; x<width; x++ )
                {
                        index = y*width + x;

                        H = (int)pHSI[3*index];
                        S = (int)pHSI[3*index+1];

                        // Red: (0 < H < 30, H > 225), (80 < S < 175), (I = don't care)
                        // Green: (60 < H < 120), (80 < S < 175), (I = don't care)
                        // Blue: (140 < H < 200), (80 < S < 175), (I = don't care)

                        if( S > 80 && S < 175 )
                        {
                                if( H <= radius || H >= (360-radius) )
                                        pBinary[index] = 255;
                                else
                                        pBinary[index] = 0;
                        }
                        else
                                pBinary[index] = 0;
                }
        }
}


void FindGreenFromHSI( unsigned char* pBinary, int range, unsigned char* pHSI, int width, int height
)
{
        int x, y, index;
        int H, S;
        int radius;

        radius = range / 2;
        for( y=0; y<height; y++ )
        {
                for( x=0; x<width; x++ )
                {
                        index = y*width + x;

                        H = (int)pHSI[3*index];
                        S = (int)pHSI[3*index+1];

                        // Red: (0 < H < 30, H > 225), (80 < S < 175), (I = don't care)
                        // Green: (60 < H < 120), (80 < S < 175), (I = don't care)
                        // Blue: (140 < H < 200), (80 < S < 175), (I = don't care)
                        if( S > 80 && S < 175 )
                        {
                                //if( H <= radius || H >= (255-radius) )
                                if( H > 60- radius && H < 61+radius )
                                        pBinary[index] = 255;
                                else
                                        pBinary[index] = 0;
```

```
                        }
                        else
                                pBinary[index] = 0;
                }
        }
}

void FindBlueFromHSI( unsigned char* pBinary, int range, unsigned char* pHSI, int width, int height
)
{
        int x, y, index;
        int H, S;
        int radius;

        radius = range / 2;
        for( y=0; y<height; y++ )
        {
                for( x=0; x<width; x++ )
                {
                        index = y*width + x;

                        H = (int)pHSI[3*index];
                        S = (int)pHSI[3*index+1];

                        // Red: (0 < H < 30, H > 225), (80 < S < 175), (I = don't care)
                        // Green: (60 < H < 120), (80 < S < 175), (I = don't care)
                        // Blue: (140 < H < 200), (80 < S < 175), (I = don't care)
                        if( S > 80 && S < 175 )
                        {
                                //if( H <= radius || H >= (255-radius) )
                                if( H > 140- radius && H < 141+radius )
                                        pBinary[index] = 255;
                                else
                                        pBinary[index] = 0;
                        }
                        else
                                pBinary[index] = 0;
                }
        }
}

void GetCenterPos( int* pX, int* pY, unsigned char* pBinary, int width, int height )
{
        int x, y, index;
        int count=0;
        long sum_x=0, sum_y=0;

        // Using the centroid method
        for( y=0; y<height; y++ )
        {
                for( x=0; x<width; x++ )
                {
                        index = y*width + x;
                        if( pBinary[index] == 255 )
                        {
                                sum_x += x;
                                sum_y += y;
                                count++;
                        }
                }
        }
        if( count > 0 )
        {
                *pX = (int)(sum_x / count);
                *pY = (int)(sum_y / count);
        }
        else
        {
                *pX = (int)(width / 2);
```

```
                *pY = (int)(height / 2);
        }
}

void CountColor(int* max_x, int* countcolor, unsigned char* pBinary, int width, int height)

//void GetCenterPos( int* pX, int* pY, unsigned char* pBinary, int width, int height )
{
        int x, y, index;
        int count=0;
        int sum_x=0, sum_y=0, ruido=0, max_y=0, may_x=0, may_y=0;

        *max_x=0;

        // Using the centroid method
        for( y=0; y<height; y++ )
        {
                sum_x=0;
                ruido=0;
                for( x=0; x<width; x++ )
                {
                        index = y*width + x;

                        if( pBinary[index] == 255 )
                        {
                                count++;
                                sum_x++;
                                ruido=0;
                        }
                        else if(ruido<ruidbola)
                        {
                                ruido++;
                                sum_x++;
                        }
                        else
                        {
                                ruido=0;
                                sum_x=0;
                        }

                }
                if (sum_x > *max_x)
                        *max_x=sum_x;
        }

        if( count > 0 )
        {
                *countcolor = count;//(int)(sum_x / count);
        }
        else
        {
                *countcolor = 0; //(int)(height / 2);
        }
}

void Tracking_PanTilt( int diff_pan, int diff_tilt, HDC hWnd )
{
        string temp;

        // Tracking pan
        if(     diff_pan > -(CENTER_WIDTH/2) && diff_pan < (CENTER_WIDTH/2) )
        {
                temp="T";                               //a la variable string temp se le asigna el
contenido de char input[10]
                nxt->write_msg(temp, 0, false);         //El mensaje se coloca en el buzon 0 de
entrada del NXT
                Sleep(10);                              //Espera por el NXT 100 milisegundos
                //if (!(nxt->read_msg(1,true)))          //Recupera la cadena en el buzón 1 (el NXT
escribira la misma cadena que reciba)
```

```
                //if ((nxt->read_msg(1,true))!=temp)
                //{}
        }
        else
        {
                if( diff_pan < 0 )
                {
                        temp="L";                               //a la variable string temp se le asigna
el contenido de char input[10]
                        nxt->write_msg(temp, 0, false);     //El mensaje se coloca en el buzon 0 de
entrada del NXT
                        Sleep(200);                             //Espera por el NXT 100 milisegundos
                        //if (!(nxt->read_msg(1,true)))      //Recupera la cadena en el buzón 1 (el
NXT escribira la misma cadena que reciba)
                //      if ((nxt->read_msg(1,true))!=temp)
                //      {}
                }

                else
                {
                        temp="R";                               //a la variable string temp se le asigna
el contenido de char input[10]
                        nxt->write_msg(temp, 0, false);     //El mensaje se coloca en el buzon 0 de
entrada del NXT
                        Sleep(200);                             //Espera por el NXT 100 milisegundos

//#########################################################

                        //if ((nxt->read_msg(1,true))!=temp) //Recupera la cadena en el buzón 1 (el
NXT escribira la misma cadena que reciba)
                //      if ((nxt->read_msg(1,true))!=temp)
                //      {}
                }
        }

        // Tracking tilt
        if(     diff_tilt > -(CENTER_HEIGHT/2) && diff_tilt < (CENTER_HEIGHT/2) )
        {
                temp="T";                                   //a la variable string temp se le asigna el
contenido de char input[10]
                nxt->write_msg(temp, 0, false);         //El mensaje se coloca en el buzon 0 de
entrada del NXT
                Sleep(10);                                  //Espera por el NXT 100 milisegundos
                //if (!(nxt->read_msg(1,true)))          //Recupera la cadena en el buzón 1 (el NXT
escribira la misma cadena que reciba)
                //if ((nxt->read_msg(1,true))!=temp)
                //{}
        }
        else
        {
                if( diff_tilt > 0 )
                {
                        temp="B";                               //a la variable string temp se le asigna
el contenido de char input[10]
                        nxt->write_msg(temp, 0, false);     //El mensaje se coloca en el buzon 0 de
entrada del NXT
                        Sleep(200);                             //Espera por el NXT 100 milisegundos
                        //if (!(nxt->read_msg(1,true)))      //Recupera la cadena en el buzón 1 (el
NXT escribira la misma cadena que reciba)
                //      if ((nxt->read_msg(1,true))!=temp)
                //      {}
                }
                else
                {
                        temp="F";                               //a la variable string temp se le asigna
el contenido de char input[10]
                        nxt->write_msg(temp, 0, false);     //El mensaje se coloca en el buzon 0 de
entrada del NXT
                        Sleep(200);                             //Espera por el NXT 100 milisegundos
```

```
                        //if (!(nxt->read_msg(1,true)))      //Recupera la cadena en el buzón 1 (el
NXT escribira la misma cadena que reciba)
                //      if ((nxt->read_msg(1,true))!=temp)
                //      {}
                }
        }
}
```

B. Código para visión en c++ con incorporación de las librerías de OpenCV y las librerías de comunicación del NXT

```
#include "stdafx.h"
#include <opencv\cv.h>
#include <OPENCV\highgui.h>
#include <opencv\cxcore.h>
#include <cstdlib>
#include <iostream>
#include <string>
#include <sstream>
#include <conio.h>
#include "nxt.h"

using namespace std;
using namespace cv;

#define NumRobots       2

// Definimos el puerto bt donde se encuentra el NXT
#define btcomm 4
#define btcomm2 5

//Inboxes
#define INBOX  1        //Buzon de entrada
#define OUTBOX 0        //Buzon de salida
#define RESP   2        //Buzon de respuesta leida y borrada
#define TAREA  3        //Tarea que se va a realizar
#define OJOS_OUT 2      //Buzon que notifica que se encontró la pelota
#define OJOS_IN  3      //Buzon que notifica que se encontró la pelota

//Mensajes
//Programa
#define START           "S"
#define END_PROC        "E"

//Vision
#define BALL_FOUND      "B"
#define GOLPEA          "G"
#define DERECHA         "D"
#define IZQUIERDA       "I"
#define RETROCEDE       "R"
#define ADELANTE        "A"
#define CENTRADA        "C"

//Defino las tareas de los ojos (mensajes de visión)
#define  pelotaVista       6    //Actualizar estado de pelota_visible
#define  posicionPantalla  7    //Actualiza posición en pantalla de la pelota

#define umbralPelota    100

//Subastas


int colorVal=0;
int alpha_slider=90,
        alpha_slider_max=255;

int pelotaEncontrada=0;

Connection *connection = new Bluetooth();
Brick *nxt = new Brick(connection);

Connection *connection2 = new Bluetooth();
Brick *nxt2 = new Brick(connection2);

string program_name = "bp7.rxe";
```

```
string read_msgg(int inbox, bool remove){
  unsigned int i;
  unsigned char answer[NXT_BUFFER_SIZE+2];
  unsigned char command[7];
  command[0]=0x05;  //command length
  command[1]=0x00;

  command[2]=0x00;
  command[3]=0x13;
  command[4]=10+inbox;
  command[5]=0x00;
  command[6]=remove;
  connection->send(&command[0],7);
  connection->receive(&answer[0],65);
  if(answer[4]){
    throw Nxt_exception::Nxt_exception("read_msg","Brick", answer[4]);
  }
  i=7;
  string result;
  result.resize(59);
  while((answer[i]!='\0')){
    result[i-7]=answer[i];
    i++;
  }
  result[i-7]='\0';
  //connection->flush();
  return result;
}

string convertInt(int number)
{
   stringstream ss;//create a stringstream
   ss << number;//add number to the stream
   return ss.str();//return a string with the contents of the stream
}


string ensamblaCad(int number, int number2, int number3)
{
   stringstream ss; //creamos un stringstream
   ss << number;//añadimos el primer numero, tipo de tarea
   ss << "  ";          //añadimos 2 espacios vacios
   ss << number2;
   ss << "  ";
   ss << number3;
   ss << "  ";
   return ss.str();//regresamos la cadena con el contenido del strinstream
}



//////////////////Deteccion del contorno////////////////////////////////
IplImage* Threshold(IplImage* src)
//void Threshold(IplImage* src, IplImage* dest)
{

IplImage* srcHSV = cvCreateImage(cvGetSize(src), 8, 3);//get size of the "src" and asign same
parameters to srcHSV

//Mat srcHSV = cvCreateImage(cvGetSize(src), 8, 3);//get size of the "src" and asign same parameters
to srcHSV

cvCvtColor(src, srcHSV, CV_BGR2HSV);// Convert the src into an HSV image

IplImage* Thresholdimg = cvCreateImage(cvGetSize(src), 8, 1); //get size of the "src" and asign same
parameters to Thresholdimg
```

```
// Change cvScalar values in the order of desired H(hue),S(saturation),V(value) . now its blue color
to threshold
// Verde: (52,100,50)-(76,255,255)  Azul(94,145,50)-(130,255,255) Rojo()-()
cvInRangeS(srcHSV, cvScalar(colorVal,145,50), cvScalar(colorVal+35,255,255), Thresholdimg);
//cvInRangeS(srcHSV, cvScalar(colorVal,145,50), cvScalar(colorVal+35,255,255), dest);

cvReleaseImage(&srcHSV);

return Thresholdimg;
}


void countdetect(const Mat& myImage,int& Result, int& pX, int& pY)
{
    int sum_x=0, sum_y=0;

        CV_Assert(myImage.depth() == CV_8U);  // accept only uchar images
    const int nChannels = myImage.channels();//nChanels está invertido BGR

    for(int j = 1 ; j < myImage.rows-1; ++j)
    {
        const uchar* current  = myImage.ptr<uchar>(j);

        for(int i= nChannels;i < nChannels*(myImage.cols-1); ++i)
        {
            if(current[i])
                        {
                                sum_x+=i;
                                sum_y+=j;
                                Result++;
                        }

        }
    }

        if( Result > 0 )
        {
                pX = (int)(sum_x / Result);
                pY = (int)(sum_y / Result);
        }
        else
        {
                pX = (int)( myImage.cols / 2);
                pY = (int)( myImage.rows / 2);
        }
}

void on_trackbar( int, void* )
{
 colorVal= alpha_slider;
}

int main()
{
        int run=1;
        int Result=0;
        int object_x=0, object_y=0;
        int terminaprograma=0;
        int key_presed=0;
        string mensaje;

        CvCapture* capture1 = 0;
        CvCapture* capture2 = 0;
        //IplImage* finalthreshold = 0;

        //capture = cvCaptureFromCAM(0);// capturing from the camera
        capture1 =  cvCreateCameraCapture(1);// capturing from the camera
        capture2 =  cvCreateCameraCapture(0);// capturing from the camera
```

```
        try{
            //cout << "Try to connect to the NXT" << endl;
            connection->connect(btcomm);
            nxt->start_program(program_name,true);
                cout<<"programa "<<nxt->get_current_program()<<endl;
                cout << "Connected" << endl;
            cout << "Read and write a message to the mailbox system on the NXT - hit any key to end"
<< endl;
        }

        //connection->disconnect();

        catch (Nxt_exception& e){
            //some error occurred - print it out
            cout << e.what() << endl;
            cout << "error code: " << e.error_code() << endl;
            cout << "error type: " << e.error_type() << endl;
            cout << e.who() << endl;
            connection->disconnect();
                // Release the capture
                //cvReleaseCapture(&capture1);
                //cvReleaseCapture(&capture2);
                PostQuitMessage(0);
                terminaprograma=1;
        }

        //connection->disconnect();


                // If there is no camera or some error
                if(!capture1 && !capture2)
                {
                printf("Can not capture,no device found\n");
                        getch();
                        return -1;
                }

                // create windows to show the video.
                cvNamedWindow("Tracking output");
                cvNamedWindow("Threshold");

                while(1)
            {
                        //finalthreshold = 0;
                        //frame = 0;

                        IplImage* frame = 0;
                        frame = cvQueryFrame(capture1);

                        // if there is no frame then break
                        if(!frame)
                    break;

                        // Holds the Thresholded image
                        IplImage* finalthreshold = Threshold(frame);
                        //finalthreshold=Threshold(frame);
                        //Threshold(frame, finalthreshold);
                        cvShowImage("Threshold", finalthreshold);//show the thresholded image
                        Mat mtx(finalthreshold); // convert IplImage* -> Mat
                        createTrackbar( "Color:", "Threshold", &alpha_slider, alpha_slider_max,
on_trackbar );
                        //printf("Cuenta: %d\n", mtx.total());
                        Result=0;
                        //void countdetect(const Mat& myImage,int& Result, int& pX, int& pY)
                        countdetect(finalthreshold, Result, object_x, object_y);
                        //printf("Cuenta: %d\n", Result);
                        //std::cout << "Color Count: " << Result << std::endl;
```

```
                        if (Result>umbralPelota)
                        {
                                pelotaEncontrada=1;
                                printf("\nPosicion en x: %d\n", object_x);
                                printf("\nPosicion en y: %d\n", object_y);
                                printf("\nCuenta: %d\n", Result);
                        }
                        else
                                printf("\nBaja Cuenta: %d\n", Result);

                        CvMemStorage*  storage  = cvCreateMemStorage(0);// allocate memory to store
the contour information
                        CvSeq* contours = 0;// list contains contours.
                        CvSeq* cpy = 0;// copy of list contains contours.
                        cvFindContours( finalthreshold, storage, &contours ); // built in function
to find contours

                        //cvContourArea(contours);

                        cpy=contours;

                        /*while (cpy)
                        {
                                printf("x: %d, y:%d", cpy[0], cpy[1]);
                                cpy++;
                        }
                        getch();
                        */

                        if( contours ){
                        /*
                                img – Image where the contours are to be drawn. As with any other
drawing function, the contours are clipped with the ROI.
                                contour – Pointer to the first contour
                                external_color – Color of the external contours
                                hole_color – Color of internal contours (holes)
                                max_level – Maximal level for drawn contours. If 0, only contour is
drawn. If 1, the contour and all contours following it on the same level are drawn. If 2, all
contours following and all contours one level below the contours are drawn, and so forth. If the
value is negative, the function does not draw the contours following after contour but draws the
child contours of contour up to the  level.
                                thickness – Thickness of lines the contours are drawn with. If it is
negative (For example, =CV_FILLED), the contour interiors are drawn.
                                lineType – Type of the contour segments, see Line description

                                Type of the line:

        8 - (or omitted) 8-connected line.

        4 - 4-connected line.

        CV_AA - antialiased line.*/

        cvDrawContours(frame,contours,CV_RGB(255,0,0),CV_RGB(255,0,0),2,3);// draw contours in red
color.
                                //double area = cvContourArea(contours);
                                //printf("Area del contorno: %5.3f\n", area);
                        }
                        //cvShowImage("Treshold", frame);
                        cvShowImage("Tracking output", frame);

                        // Wait for a keypress

                        if (kbhit()==1)
                        {
                                printf("\nCadena enviada: ");
                                key_presed=getche();
                                printf("\nCuenta: %d\n", Result);
                                printf("\nPosicion en x: %d\n", object_x);
```

```
                                printf("\nPosicion en y: %d\n", object_y);

                                //printf("Area del contorno: %5.3f\n", area);
                                switch(key_presed)
                                {
                                        case 83 :       nxt->write_msg(START, OUTBOX, false); //"S"
is placed in inbox 0 on the NXT
                                                                Sleep(100); //wait for nxt
                                                                cout <<"Answer from nxt: " <<
read_msgg(INBOX,TRUE) << endl; //Retrive the string in inbox 1 (the NXT will place the same string
)break;


                                                                key_presed=0;
                                                                break;
                                        case 27 :       nxt->write_msg(END_PROC, OUTBOX, false);
//"S" is placed in inbox 0 on the NXT
                                                                terminaprograma=1;
                                                                break;
                                        }
                        }

                        //Parte de la vision
                        if(pelotaEncontrada)
                        {
                                nxt->write_msg(convertInt(pelotaVista), OJOS_OUT, false); //"6" es
puesto en el buzon 2 del NXT
                                read_msgg(OJOS_IN,TRUE);
                                nxt->write_msg(ensamblaCad(posicionPantalla, object_x, object_y),
OJOS_OUT, false); //"7+objx+objy" se pone en el buzon 2 del NXT
                                read_msgg(OJOS_IN,TRUE);
                        }

                        //Simulacion de la comunicacion tipo broadcast
                        //Si fueran mas robots hacer con For y con vectores usando el contador como
indice
                        //Robot1
                        mensaje=read_msgg(INBOX,TRUE);
                        if(mensaje.size()>0)
                        {
                                nxt2->write_msg(mensaje, OUTBOX, false);
                                nxt2->read_msg(INBOX,TRUE);
                        }

                        int c = cvWaitKey(27);
                        /*
                        if(c!=-1)
                        switch(c)
                        {
                                case 83 :       nxt->write_msg(START, OUTBOX, false); //"S" is
placed in inbox 0 on the NXT
                                                        Sleep(100); //wait for nxt
                                                        //cout <<"Answer from nxt: " << nxt-
>read_msg(1,true) << endl; //Retrive the string in inbox 1 (the NXT will place the same string
)break;
                                                        break;
                                case 27 :       nxt->write_msg(END_PROC, OUTBOX, false); //"S" is
placed in inbox 0 on the NXT
                                                        Sleep(100); //wait for nxt
                                                        //cout <<"Answer from nxt: " << nxt-
>read_msg(1,true) << endl; //Retrive the string in inbox 1 (the NXT will place the same string
)break;
                                                        terminaprograma=1;
                                                        break;
                        }
                        */
                        if(terminaprograma==1)
                                break;
                        //if (run)
```

```
                // Release the thresholded image.
                cvReleaseImage(&finalthreshold);
                cvReleaseMemStorage(&storage);
            }

                // Release the thresholded image.

                //cvReleaseImage(&finalthreshold);
                // Release the capture
                cvReleaseCapture(&capture1);
                cvReleaseCapture(&capture2);
        return 0;
}
```

C. Código del robot NXT en nxc

```
#define DIST 25               //40   20    30
#define K1 2.5 //0.5555555    // 3   10     6    2.5      Giro PID
#define K2 .4  //0.0333333    // 1    2     2       .5    Giro PID
#define K3 .3                 //                         Giro PID
#define K4 10    //3   10      6
#define K5 2     //1    2      2
#define K6 0

#define K7 3     //3   10      6
#define K8 0.3   //1    2      2
#define K9 0
#define DT 10

//Defino constantes del robot
#define CMXG 0.0555555555   //cm por grado de giro
#define D    10.3           //distancia entre las ruedas
#define distanciaVision 15  //Distancia a la que el robot puede ver la pelota
#define vel_min   30        //Velocidad minima del robot
//Defino dimensiones de la cancha
#define tamx      300
#define tamy      200

#define CENTER_WIDTH 50    //Ancho de la ventana de pelota centrada
#define CENTER_HEIGHT 50   //Alto de la ventana de pelota centrada
#define tamIm_x   640      //Tamaño de la imagen en x
#define tamIm_y   480      //Tamaño de la imagen en y


//Buzones
#define INBOX     0       //Buzon de entrada
#define OUTBOX    1       //Buzon de salida
#define OJOS_IN   2       //Buzon de entrada de vision
#define OJOS_OUT  3       //Buzon de salida de vision

//Defino las tareas (mensajes de buzón de entrada)
//Tareas de subasta
#define   mandarPase       0     //Indica la subasta de mandar pase
#define   recibirPase      1     //Indica la subasta de recibir pase
//Solo ejecución
#define   iniciarTarea     2     //Iniciar en posición aleatoria
#define   buscarPelota     3     //Busca pelota, por simplicidad
//Solo actualización
#define   posicionPelota   4     //Para actualizar la posición de la pelota
#define   ofertaSubasta    5     //Para mandar mi oferta
#define   resultadoSubasta 6     //Para indicar que se eligió ganador


//Defino las tareas de los ojos (mensajes de visión)
#define  pelotaVista       6    //Actualizar estado de pelota_visible
#define  posicionPantalla  7    //Actualiza posición en pantalla de la pelota

//       0123456789012345
//Cadena XXX--NNNNN-JJJJJ
//Defino los tamaños de las cadenas
#define   tamCadTarea     3     //Tamaño de la cadena para la parte de tarea
#define   tamCadValor1    5     //Tamaño de la cadena para la parte del valor
#define   tamCadValor2    5     //Tamaño de la cadena para la parte del valor

//Mensajes
//Programa
#define   START           "S"
#define   END_PROC        "E"

//Vision
```

```
#define   BALL_FOUND  "B"
#define   BALL_LOST   "L"
#define   GOLPEA      "G"
#define   DERECHA     "D"
#define   IZQUIERDA   "I"
#define   RETROCEDE   "R"
#define   ADELANTE    "A"
#define   CENTRADA    "C"

//Definicion de prototipos
//void mandaPase();

//Subastas
//#define CMXG 0.048440748
//#define D    14

task avancepid();

mutex varMutex;

byte handle;
int p,
    pe=1,
    bd=1,
    b=1,
    bc=1,
    crec=0,
    band=1,               //Bandera de giro PID
    salir=1,              //Bandera para ciclo principal
    tiempo_impresion=0,   //Contador de veces sin guardar posicion
    fin=1,                //Para la escucha de mensaje en buzon
    ballFound=0,          //Indica si la pelota es visible
    speed=50,             //La velocidad de giro
    inicia=0,             //Bandera de inicio del main
    mensaje=0,            //Bandera de mensaje recibido
    pelotaActualizada=0,  //Bandera de actulizacion de la pelota
    tarea=0,
    pelota_vista=0,
    pelotaLocalizada=0,
    mensajeRecibido=0,
    distPel=70;           //Distancia aprox que avanza la pelota sin desviarse

float theta=0,
      thetag=0,
      c=0,
      grad=0,
      x=0,             //Posicion del robot en x
      y=0,             //Posicion del robot en y
      x1=25,           //Posicion destino en x
      y1=25,           //Posicion destino en y
      difx,
      dify,
      x2,
      y2,
      dr=0,
      dr2,
      sum=0,
      sum2=0,
      dr3,
      thetap=0,
      deltad=0,
      deltasum=0,
      gradg_ant=0,
      cambio_gradg=0,     //Cambio en la orientacion, guarda posicion
      desviacion,
      pelota_y=0,         //Posicion de la pelota en y
      pelota_x=0,         //Posicion de la pelota en x
      vista_x=0,          //Posicion de la pelota vista en x
      vista_y=0,          //Posicion de la pelota vista en y
```

```
      pase_x,             //Posicion a recibir pase en x
      pase_y,             //Posicion a recibir pase en y
      valor1=0,
      valor2=0,
      oferta=0;

long  tick_anterior;


task escucha()
{
 string mensajeEntrante;  //Cadena para guardar el mensaje entrante

 while(1)
 {
  ReceiveMessage(INBOX, true, mensajeEntrante);//Recibimos mensaje, true para borrar
  //SubStr("str", "idx", "len");
  //strncat("dest", "src", "num")
  //strcat("dest", "src")
  //FormatNum("fmt", "val")

  if(StrLen(mensajeEntrante)>0)  //Si se recibió un mensaje
  {
   mensajeRecibido=1;         //Llego tarea nueva
   tarea= StrToNum(SubStr(mensajeEntrante, 0, tamCadTarea)); //Guardo la tarea
   valor1= StrToNum(SubStr(mensajeEntrante, tamCadTarea, tamCadValor1));//Guardo valor 1
   valor2= StrToNum(SubStr(mensajeEntrante, tamCadTarea+tamCadValor1, tamCadValor2));//Guardo valor
2
   //poner la entrada en el buzon de salida como ACK
   //Esto esta almacenado en el NXT, por lo que la PC lo tiene que remover
   SendResponseString(OUTBOX,mensajeEntrante);
   mensajeEntrante="";           //Se limpia la cadena que guarda los mensajes
  }
  /*

    mensaje=1;
    switch( request )
    {
         //Movimientos del movil
         case BALL_FOUND: ballFound=1; break; //StartTask(derecha); break;
         case RETROCEDE : OnRev(OUT_BC, speed); break;
         case ADELANTE    : OnFwd(OUT_BC, speed);       break;
         //case GOLPEA    : StartTask(golpear); break;
         //case GIRA        : StartTask(girag); break;
         //Paro y fin de proceso
         //case STOP              : Off(OUT_BC); break;
         case START      : inicia=1; break;
         case END_PROC   : Off(OUT_BC); b=0; StopAllTasks(); break;
     }

        ReceiveMessage(PELOTA, true, request2);//Recieve message true will remove the message from
the nxt mailbox
        //SubStr("str", "idx", "len");
        //strncat("dest", "src", "num")
        //strcat("dest", "src")
        //FormatNum("fmt", "val")

        if(StrLen(request2)>0){ //Una actializacion de la pelota se recibió
            pelotaActualizada=1;
            switch( request2 )
                                {
                                                //Movimientos del movil
                    case BALL_FOUND     : ballFound=1; break; //Pelota encontrada
                    case BALL_LOST      : ballFound=0; break; //Pelota perdida
            }

            //place the input as a response in the OUTBOX mailbox
            //Notice that this is stored on the NXT in a mailbox untill read by the PC
            //- so make sure the PC removes the response
```

```
            SendResponseString(OUT_PELOTA,request);
            //TextOut(0,LCD_LINE1,request2,true);
            request2="";//empty request2
        }*/
    }
}

task ojos()
{
 string mensajeEntrante;  //Cadena para guardar el mensaje entrante

 while(1)
 {
  ReceiveMessage(OJOS_IN, true, mensajeEntrante);//Recibimos mensaje, true para borrar
  //SubStr("str", "idx", "len");
  //strncat("dest", "src", "num")
  //strcat("dest", "src")
  //FormatNum("fmt", "val")

  if(StrLen(mensajeEntrante)>0)  //Si se recibio un mensaje
  {
   mensajeRecibido=1;         //Llego tarea nueva
   tarea= StrToNum(SubStr(mensajeEntrante, 0, tamCadTarea)); //Guardo la tarea
   valor1= StrToNum(SubStr(mensajeEntrante, tamCadTarea, tamCadValor1));//Guardo valor 1
   valor2= StrToNum(SubStr(mensajeEntrante, tamCadTarea+tamCadValor1, tamCadValor2));//Guardo valor
2
   switch( tarea )
         {
          //Actualizacion de vista
    case pelotaVista      : pelota_vista=1; //Pelota en campo de visión
                            break;
    case posicionPantalla       : vista_x=valor1;
                            vista_y=valor2;
                            break; //Pelota perdida
   }
   //poner la entrada en el buzon de salida como ACK
   //Esto esta almacenado en el NXT, por lo que la PC lo tiene que remover
   SendResponseString(OJOS_OUT, mensajeEntrante);
   mensajeEntrante="";           //Se limpia la cadena que guarda los mensajes
  }

 }
}


float normtheta(float theta)
{
   if (theta<0)
      if (abs(theta)<360)
         return(360+theta);//rethink better
      else
         return(abs(theta)-360);
   else
      if (theta>360)
         return(theta-360);
      else
         return(theta);
}

float euclidiana(float x1, float y1, float x2, float y2)
{
 float a=0, b=0, c=0;

 a=abs(x1-x2);
 b=abs(y1-y2);
 c=sqrt((a*a)+(b*b));

 return c;
}
```

```
/*********************Odometria*************************************/
task odometria()
{
 int    tikda=0,
        tikia=0,
        tikdp=0,
        tikip=0,
        deltagd=0,
        deltagi=0;


 float  deltad=0,
        deltai=0,
        deltas=0,
        deltat=0,
        cuad=0,
        cuad2,
        distobj=0,
        distobjp=0;

 string s,sx,sy,sd,st;
 int bytesWritten;

 DeleteFile("Odo.txt");
 CreateFile("Odo.txt",50000,handle);


     while(1){
          tikda=MotorRotationCount(OUT_C);
          tikia=MotorRotationCount(OUT_B);

          deltagd=tikda-tikdp;
          deltagi=tikia-tikip;
          deltad=deltagd*CMXG;
          deltai=deltagi*CMXG;

          deltas=(deltai+deltad)/2;
          deltat=(deltad-deltai)/D;

          x=x+deltas*cos(theta+(deltat/2));
          y=y+deltas*sin(theta+(deltat/2));

          cuad=(x*x)+(y*y);
          dr=sqrt(cuad);

          x2=x1-x;
          y2=y1-y;

          cuad2=(x2*x2)+(y2*y2);

          dr2=sqrt(cuad2);
          sum=dr+dr2;

          deltasum=sum-c;

          if(crec==1)
          {
             if(deltasum>0)
               bc=0;
             else
             {
               bc=0;
               crec++;
             }
          }

          sum2=sum;
```

```
            deltad=dr2-dr3;
            dr3=dr2;
            theta=theta+deltat;
            //Acquire(varMutex);
            thetag=(theta*360)/(2*PI);

            /***************** Normalizar Theta*************************/
            //thetag=normtheta(thetag);
            //Release(varMutex);
            tikdp=tikda;
            tikip=tikia;
            //Diferencia del tiempo anterior registrado y el actual
            tick_anterior=CurrentTick()-FirstTick();
            st = NumToStr(tick_anterior);
            sx = NumToStr(x);
            sy = NumToStr(y);
            sd = NumToStr(thetag);

            s= StrCat(sx,"      ",sy,"      ",sd,"      ",st);
            //s= StrCat(sx,"      ",sy);
            //cambio en gradg
            cambio_gradg=abs(gradg_ant-thetag);
            //Si el cambio de grado es mayor a 0.01 escribo, de lo contrario no
            if (cambio_gradg>0.01)
            {
               WriteLnString(handle, s,  bytesWritten);
            }
            else
            {
               //Si han pasado 5 veces sin imprimir, imprimo
               if (tiempo_impresion>5)
               {
                  WriteLnString(handle, s,  bytesWritten);
                  tiempo_impresion=0;
               }
               else//en otro caso sigo contando las veces sin imprimir
                   tiempo_impresion=tiempo_impresion+1;
            }
            gradg_ant=thetag;
            Wait(20);
        }
        Off(OUT_BC);
        CloseFile(handle);
}
/********************** Rodear Pelota ******************************/
task rodearPelota()
{
  int i,
  pwr=-20,
  e1=0,
  e2=0,
  itg,
  der,
  pb,
  pc,
  tiksa=0,
  bw=1;

  float thetaq;

  tiksa=MotorRotationCount(OUT_A);
  OnFwd(OUT_A,20);
  while(tiksa<60)
    tiksa=MotorRotationCount(OUT_A);
  Off(OUT_A);
  Acquire(varMutex);
  thetap=thetag;
  Release(varMutex);
```

```
Off(OUT_BC);
thetaq=thetap;
thetaq=thetaq+60;
while (thetag<thetaq)
{
  pb=pwr;
  OnFwd(OUT_B, pb);
  pc=-pwr;
  OnFwd(OUT_C, pc);
}
crec=1;
bc=1;

//SetSensorLowspeed(IN_1);
StartTask(odometria);

while(deltasum>0.01 || (bc==1))
{
 TextOut(0,LCD_LINE5, "dr:");
 NumOut(25,LCD_LINE5,dr);
 TextOut(0,LCD_LINE6, "dr2:");
 NumOut(25,LCD_LINE6,dr2);
 TextOut(0,LCD_LINE3, "Ds:");
 NumOut(25,LCD_LINE3,deltasum);
 TextOut(0,LCD_LINE4, "bc:");
 NumOut(25,LCD_LINE4,bc);
 i= euclidiana(x1, y1, pelota_x, pelota_y);
 e1=i-DIST;
 der=e1-e2;
 itg=itg+e1;
 pwr=K7*e1+K8*der+K9*itg;
 if(pwr>0)
 {
   if (pwr>100)
   {
      pb=25;
      OnFwd(OUT_B, pb);
      pc=15;
      OnFwd(OUT_C, pc);
   }
   else
   {
      pb=30+(pwr);
      OnFwd(OUT_B, pb);
      pc=30-(pwr);
      OnFwd(OUT_C, pc);
   }
 }
 else
 {
    pb=30+(pwr);
    OnFwd(OUT_B, pb);
    pc=30-(pwr);
    OnFwd(OUT_C, pc);
 }
}
Off(OUT_BC);
e2=e1;
Wait(DT);
tiksa=MotorRotationCount(OUT_A);
OnFwd(OUT_A,-20);
while(tiksa>0)
  tiksa=MotorRotationCount(OUT_A);
Off(OUT_A);
pwr=20;
while (thetag<thetap)
{
  pb=-pwr;
  OnFwd(OUT_B, pb);
```

```
    pc=pwr;
    OnFwd(OUT_C, pc);
  }
  Off(OUT_BC);
  bd=0;
}

/************************** Grados de giro ******************************/
void gradg()
{
  float cuad=0,
        stet=0,
        thet=0;

  int ante=0;
  string sx;
  difx=x1-x;
  dify=y1-y;

  if (difx>=0)
     if (dify>=0)
        ante=1;
     else
        ante=4;
  else
     if (dify>=0)
        ante=2;
     else
        ante=3;


  cuad=(difx*difx)+(dify*dify);
  c=sqrt(cuad);
  stet=dify/c;
  thet=asin(stet);
  grad=(thet*360)/(2*PI);


  switch (ante)
  {
    case 1: //Queda el mismo angulo que es el obtenido
            break;

    case 2 ://En el cuadrante 2 se tiene que restar a 180
            grad=180-grad;
            break;
    case 3 ://En el cuadrante 3 se tiene que sumar a 180
            grad=180+grad;
            break;
    default ://En el cuadrante 4 se resta a 0
            grad=0-grad;
            break;
  }
   sx = NumToStr(grad);
}


/****************************** Giro PID ********************************/
task giropid()
{
   int i,
       pwr=0,
       e1=0,
       e2=0,
       itg,
       der,
       pb,
       pc;
```

```
   while(band)
   {
      TextOut(0,LCD_LINE2, "X1:");
      NumOut(25,LCD_LINE2,x1);
      TextOut(0,LCD_LINE3, "Y1:");
      NumOut(25,LCD_LINE3,y1);

      TextOut(0,LCD_LINE5, "grd:");
      NumOut(25,LCD_LINE5, grad);
      TextOut(0,LCD_LINE6, "grg:");
      NumOut(25,LCD_LINE6,thetag);
      i=thetag;
      e1=(abs(grad)-abs(i));
      der=e1-e2;
      itg=itg+e1;
      pwr=K1*e1+K2*der+K3*itg;
      if(grad>0)
      {
            pb=pwr;
            //pb=-pwr;
            OnFwd(OUT_B, pb);
            //pc=pwr;
            pc=-pwr;
            OnFwd(OUT_C, pc);
      }
      else
      {
            //pb=pwr;
            pb=-pwr;
            OnFwd(OUT_B, pb);
            //pc=-pwr;
            pc=pwr;
            OnFwd(OUT_C, pc);
      }

      e2=e1;
      Wait(DT);
      if((thetag>(grad-0.5))&& (thetag<(grad+0.5)))
      band=0;
   }
    Off(OUT_BC);
    //Wait(5000);
    //ExitTo(avancepid);
}

/***************************** Avance PID ******************************/
task avancepid()
{
int i,
    pwr,
    pwrd,
    pi,
    pd,
    e1=0,
    e2=0,
    itg,
    der;
   //SetSensorLowspeed(IN_1);
   while(dr2>0 && pelota_vista==0)
   {
      ClearScreen();
      TextOut(0,LCD_LINE1, "dr:           ");
      NumOut(25,LCD_LINE1,dr);
      TextOut(0,LCD_LINE2, "dr2:          ");
      NumOut(25,LCD_LINE2,dr2);
      TextOut(0,LCD_LINE3, "Desv:           ");
      NumOut(25,LCD_LINE3,desviacion);
      TextOut(0,LCD_LINE4, "bc:           ");
```

```
NumOut(25,LCD_LINE4,bc);
TextOut(0,LCD_LINE5, "grd:          ");
NumOut(25,LCD_LINE5,thetag);
TextOut(0,LCD_LINE6, "grg:          ");
NumOut(25,LCD_LINE6,grad);
p=SensorUS(IN_1);
i=dr;
e1=c-i;
der=e1-e2;
itg=e2+e1;
pwr=K4*e1+K5*der+K6*itg;


/************** Verificar desviación ****************/
desviacion= grad-thetag;
if (abs(desviacion)<0.5)
{
   if (pwr>126)
   {
      //OnFwd(OUT_BC, 50);
      OnFwd(OUT_B, 70);
      OnFwd(OUT_C, 70);
      //OnFwdSync(OUT_BC, 50,0); //(OUT_BC, 50, -2);
   }
   else
   {
      //OnFwd(OUT_BC, pwr/2);
      OnFwd(OUT_B, pwr);
      OnFwd(OUT_C, pwr);
      //OnFwdSync(OUT_BC, pwr/2,0);  //(OUT_BC, pwr/2,-2);
   }
}
else
{
//   /*
   pwrd=(desviacion*50)/90;
   if(pwrd>0)
   {
      if(pwrd>vel_min)
      {
         OnFwd(OUT_B, pwrd-abs(desviacion)*2);
         OnFwd(OUT_C, pwrd);
      }
      else
      {
         OnFwd(OUT_B, vel_min);
         OnFwd(OUT_C, vel_min+abs(desviacion)*2);
      }
   }
   else
   {
      if(abs(pwrd)>vel_min)
      {
         OnFwd(OUT_B, pwrd);
         OnFwd(OUT_C, pwrd-abs(desviacion)*2);
      }
      else
      {
         OnFwd(OUT_B, vel_min+abs(desviacion)*2);
         OnFwd(OUT_C, vel_min);
      }


   }
//   */
     /*gradg();
     StartTask(giropid);
     Wait(50);
     StopTask(giropid);
```

```
           */
      }


      e2=e1;
      Wait(500);
      //StartTask(giropid);
      //Wait(50);
      //StopTask(giropid);
      /* if(p<30){
          Off(OUT_BC);
          StartTask(pollito);
          StopTask(avancepid);
          //break;
       }*/

       if(pwr<10 && pwr>-10)
          b=0;
   }
   Off(OUT_BC);
   StopTask(avancepid);
}

/********************** Distancia Pelota ********************************/
int distpel()
{
 return (((vista_y-240)*10)/240);
}

/********************** Genera posicion inicial *************************/
void genpi() //Genera Posicion Inicial del robot
{
     x1=Random((tamx/2)-30); //A 15 cm de las orillas 200- 15(2)
     y1=Random(tamy-(2*distanciaVision));  //
     x1=x1+15;            //Rectifica el Rand que va de (0, lim-30)
     y1=y1+15;            //por el intervalo (15, lim-15)
}

/*************** Calcula la posicion de la pelota ************************/
void posPel()
{
 pelota_x=x+distpel()*cos(thetag);
 pelota_y=y+distpel()*sin(thetag);
}


/************************** Evaluar Mandar ******************************/
float evaluar_mandar()
{
 float dist=0;
 float bateria=BatteryLevel()/100; //Bateria en milivolts
 float distancia=0;                //Distancia entre pelota y robot
 float resultado=0;

 distancia=euclidiana(x1, y1, pelota_x, pelota_y);
 resultado=distancia*(bateria/10);
 return resultado;
}

/************************** Evaluar Recibir *****************************/
float evaluar_recibir()
{
 float dist=0;
 float bateria=BatteryLevel()/100; //Bateria en milivolts
 float distancia=0;                //Distancia entre pelota y robot
 float resultado=0;

 distancia=euclidiana(x1, y1, pase_x, pase_y);
 resultado=distancia*(bateria/10);
```

```
 return resultado;
}



/************************ Posicion Inicial ****************************/
void posicionInicial()
{
  genpi();
  gradg();
  StartTask(giropid);
  while(band)
  {}
  Wait(100);
  band=1;
  StartTask(avancepid);
  while(b)
  {}
}


void centrarPelota(int diff_pan, int diff_tilt)
{
 // Tracking pan
        if(     diff_pan > -(CENTER_WIDTH/2) && diff_pan < (CENTER_WIDTH/2) )
      Off(OUT_BC);
        else
        {
                if( diff_pan < 0 )
                {
        OnFwd(OUT_B, 25);
        OnFwd(OUT_C, -25);
                }

                else
                {
        OnFwd(OUT_B, -25);
        OnFwd(OUT_C, 25);
                }
        }

        // Tracking tilt
        if(     diff_tilt > -(CENTER_HEIGHT/2) && diff_tilt < (CENTER_HEIGHT/2) )
      Off(OUT_BC);
        else
        {
                if( diff_tilt > 0 )
                {
      OnFwd(OUT_B, -25);
      OnFwd(OUT_C, -25);
                }
                else
                {
      OnFwd(OUT_B, 25);
      OnFwd(OUT_C, 25);
                }
        }
}

void posPase()
{
 centrarPelota(tamIm_x/2 - vista_x, tamIm_y/2 - vista_y);
 //Rodear pelota hasta quedar en posición de pase

}


/************************ Mandar Pase *********************************/
void mandaPase()
```

```
{
 int cont=0;
 float thetaPase;
 string subastaS; //Cadena para guardar la respuesta
 string pasexS;   //Cadena para guardar la posicion del pase en x
 string paseyS;   //Cadena para guardar la posicion del pase en y
  //Calcular posicion final del pase
  if (pelota_x < tamx)
  {
   pase_x=(tamx/2)+5;
   thetaPase=acos(pase_x/distPel);
   pase_y=distPel*(sin(thetaPase));
  }
  else
  {
   pase_x=(tamx/2)-5;
   thetaPase=acos(pase_x/distPel);
   pase_y=distPel*(sin(thetaPase));
  }
   //Subastar recibir pase
   subastaS=NumToStr(recibirPase);
   pasexS=NumToStr(pase_x);
   paseyS=NumToStr(pase_y);
   strcat(pasexS, paseyS);
   strcat(subastaS, pasexS);
   SendMessage(OUTBOX, subastaS);
   //Esperar respuesta medio segundo
   while (cont < 500)
   {
    if (mensajeRecibido==1)//Si llega oferta
    {
     mensajeRecibido=0;    //reseteo la respuesta
     if (valor1>oferta)    //Si es mayor la oferta
      oferta=valor1;        //Guardo la oferta
     else                  //Si no gané
     {
      pase_x=0;            //Limpio las variables
      pase_y=0;
     }
    }
    cont++;
    Wait(1);
   }
   //anunciar ganador
   subastaS=NumToStr(resultadoSubasta);
   pasexS=NumToStr(oferta);

   strcat(subastaS, pasexS);
   SendMessage(OUTBOX, subastaS);

   //posicionarse
   posPase();

   //mandar pase

}

/************************* Buscar Pelota ******************************/
void buscaPelota()
{
 string respuestaS;  //Cadena para guardar la respuesta
 string posxS;  //Cadena para guardar la posicion de la pelota en x
 string posyS;  //Cadena para guardar la posicion de la pelota en y

 while(pelotaLocalizada==0)
 {

  if (pelota_vista==0)
  {
```

```
   OnFwd(OUT_B, 50);
   OnFwd(OUT_C,-50);
   //Area();
   Wait(500);
   //si esta encontrada bien, si no sigo buscando
  }
  else
  {
   pelotaLocalizada=1;
   Off(OUT_BC);

   respuestaS=NumToStr(pelotaVista);
   SendMessage(OJOS_OUT, respuestaS);
   posPel();
   respuestaS=NumToStr(pelotaVista);
   posxS=NumToStr(pelota_x);
   posyS=NumToStr(pelota_y);
   strcat(posxS, posyS);
   strcat(respuestaS, posxS);
   SendMessage(OJOS_OUT, respuestaS);
   mandaPase();
  }
 }
}

/************************ Recibir Pase *********************************/
void recibePase()
{
  x1=valor1;
  y1=valor2;
  gradg();
  StartTask(giropid);
  while(band)
  {}
  Wait(100);
  band=1;
  StartTask(avancepid);
  while(b)
  {}
  if (pelotaVista)
     centrarPelota(tamIm_x/2 - vista_x, tamIm_y/2 - vista_y);
  else
      buscaPelota();
}

  //SubStr("str", "idx", "len");
  //strncat("dest", "src", "num")
  //strcat("dest", "src")
  //FormatNum("fmt", "val")

/****************************** Main **********************************/
task main()
{
 string respuestaS;  //Cadena para guardar la respuesta
 string ofertaS;  //Cadena para guardar la respuesta
 string valorS;  //Cadena para guardar la respuesta

 StartTask(odometria);           //Inicializa la odometría
 StartTask(escucha);             //Inicializa la escucha de mensajes
 StartTask(ojos);                //Inicializa la escucha de la visión

 while ( pelota_y < ((3*tamy)/4))//Mientras la pelota no rebase la linea final
 {
  while(mensajeRecibido==0)      //Mientras no se reciba mensaje, espero
  {}
  mensajeRecibido=0;
  switch(tarea)
  {
   //Tareas con evaluación
```

```
   case recibirPase:  pase_x=valor1;
                      pase_y=valor2;
                      oferta=evaluar_recibir();//Se evalua para recibir pase

                      ofertaS=NumToStr(oferta);
                      respuestaS=NumToStr(ofertaSubasta);
                      strcat(respuestaS, ofertaS);
                      SendMessage(OUTBOX,respuestaS);

                      while(mensajeRecibido==0)
                      {}
                      mensajeRecibido=0;      //reseteo la respuesta
                      if (valor1==oferta)
                         recibePase();
                      mandaPase();
                      break;



   case mandarPase:   oferta=evaluar_mandar();//Se evalua para mandar pase
                      ofertaS=NumToStr(oferta);
                      respuestaS=NumToStr(ofertaSubasta);
                      strcat(respuestaS, ofertaS);
                      SendMessage(OUTBOX,respuestaS);

                      while (mensajeRecibido==0)//Mientras no llegue el ganador
                      {}
                      mensajeRecibido=0;      //reseteo la respuesta
                      if (valor1==oferta)     //Si gané
                         mandaPase();         //Mando el pase
                      else                    //Si no gané
                      {
                       pase_x=0;              //Limpio las variables
                       pase_y=0;
                      }
                      break;



   //Tareas de ejecucion
   case iniciarTarea: posicionInicial();
                      break;

   case buscarPelota: buscaPelota();
                      break;

   //Actualizar datos
   case posicionPelota: pelotaLocalizada=1;
                        pelota_x=valor1;
                        pelota_y=valor2;
                        break;

   /*case resultadoSubasta: resultado=valor1;
                          //ganado=compararResultado(tareaActual, resultado);
                          //respuest
                          break;*/

   default: break;
  }


 }

/*
while (salir)
   switch (mensaje)
   {
     case avanza:
```

```
                "body"
                break;
    case gira:
                "body"
                break;

    case inicializa:
                //genpi();
                gradg();
                StartTask(odometria);
                StartTask(giropid);
                Wait(5000);


    case salir:
                salir=0;

    default:
                "body"
  }



  genpi();
  gradg();
  StartTask(odometria);
  StartTask(escucha);
  //while(inicia==0)
  //{}
  StartTask(giropid);
  //StartTask(buscaPelota);
//   Wait(5000);
  //Wait(2000);
  //StopTask(giropid);
  //OnFwd(OUT_A,0);
  //while(b)
  //{
  //   gradg();
  StartTask(avancepid);
  StartTask(buscaPelota);
  while(b)
  {}
  //   bd=1;
  //   StopTask(pollito);
  //   StopTask(avancepid);
  //   Off(OUT_BC);
  //} */

  StopAllTasks();
}
```

D. Falla técnica de la cámara

Se contaban con 2 tipos de cámaras, un par de cámaras de los robots de Bioloid y un par de cámaras de DraganFly. Estas últimas eran las idóneas para trabajar con ellas y por su calidad de imagen, lo que paso con estas es que una dejó de funcionar y la otra tenia mecho ruido al transmitir la imagen lo cual hacia prácticamente imposible procesar la imagen. Y con las cámaras de Bioloid lo que no se tenía contemplado fueron los requerimientos de alimentación de esta cámara ya que se había usado anteriormente pero montada y conectada al módulo CM-5 del NXT. Y al querer usarla con pilas de 9v se logró observar que funcionaba intermitentemente por un tiempo muy corto y después comenzaba a cambiar el canal de transmisión lentamente, con lo cual tampoco se puede obtener una imagen con la que se pueda trabajar. El motivo de colocarle la pila de 9 volts fue que en el manual de Experto de Bioloid se menciona que la cámara funciona mientras se le alimente con una corriente continua. Y como se comportaba muy raro la cámara se procedió a tomar mediciones del voltaje y la corriente que suministran cada fuente, la de Bioloid y la de la pila de 9v. El voltaje medido de la fuente de alimentación de Bioloid fue de 12v, mientras que la de la pila fue de 9v. Viendo esto solo faltaba medir la corriente para ver si ahí se encontraba lo que explicara el porqué de la mala funcionabilidad de la cámara. En las siguientes imágenes se muestran las medidas de corriente en ambos casos.

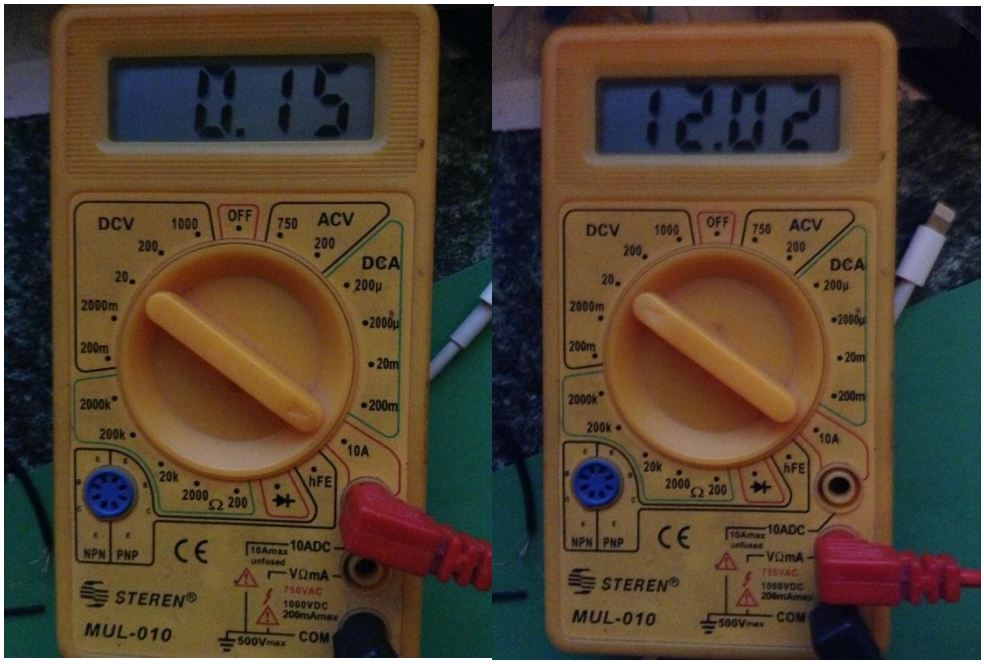

Figura 32 Corriente y voltaje de la fuente de Bioloid

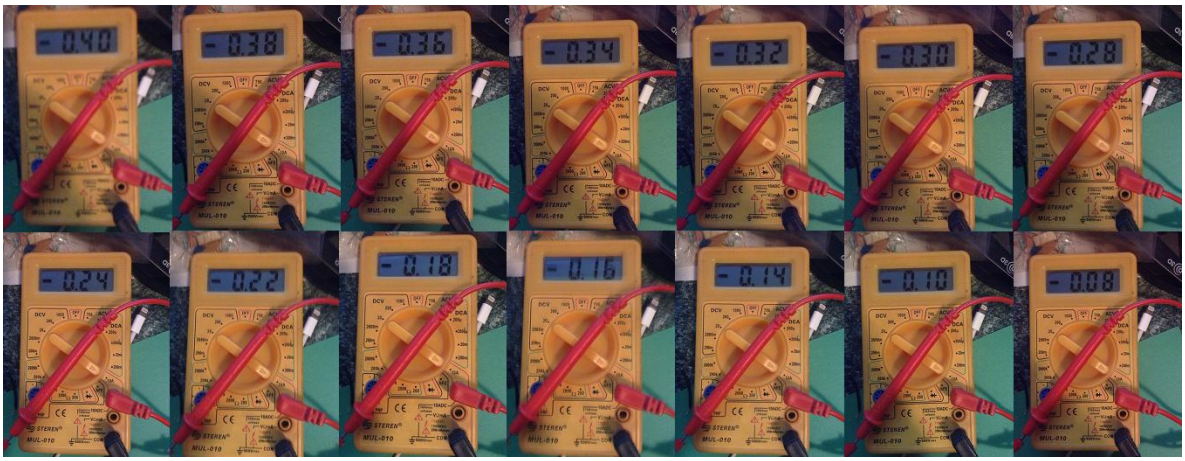

Figura 33 Fotos de la medición de la corriente con la pila de 9v

Como se puede observar claramente en la figura 32, la corriente no se mantiene fija utilizando una pila de 9v y es por esto que no funciona la cámara de Bioloid montada en el robot LEGO.

Bibliografía

Born, M., & Wolf, E. (1999). *Principles of optics : electromagnetic theory of propagation, interference and diffraction of light*. Cambridge; New York: Cambridge University Press.

Candea, C., Hu, H. S., Iocchi, L., Nardi, D., & Piaggio, M. (2001). Coordination in multi-agent RoboCup teams. [Article]. *Robotics and Autonomous Systems, 36*(2), 67-86. doi: 10.1016/s0921-8890(01)00137-3

Chen, H., & Wolff, L. B. (1998). Polarization phase-based method for material classification in computer vision. [Article]. *International Journal of Computer Vision, 28*(1), 73-83. doi: 10.1023/a:1008054731537

Dana, K. J., Van Ginneken, B., Nayar, S. K., & Koenderink, J. J. (1999). Reflectance and texture of real-world surfaces. [Article]. *Acm Transactions on Graphics, 18*(1), 1-34. doi: 10.1145/300776.300778

Dias, M. B., Zlot, R., Kalra, N., & Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. [Article]. *Proceedings of the Ieee, 94*(7), 1257-1270. doi: 10.1109/jproc.2006.876939

Dzmura, M., & Iverson, G. (1993). COLOR CONSTANCY .1. BASIC THEORY OF 2-STAGE LINEAR RECOVERY OF SPECTRAL DESCRIPTIONS FOR LIGHTS AND SURFACES. [Article]. *Journal of the Optical Society of America a-Optics Image Science and Vision, 10*(10), 2148-2163. doi: 10.1364/josaa.10.002148

Farinelli, A., Locchi, L., & Nardi, D. (2004). Multirobot systems: A classification focused on coordination. [Review]. *Ieee Transactions on Systems Man and Cybernetics Part B-Cybernetics, 34*(5), 2015-2028. doi: 10.1109/tsmcb.2004.832155

Finlayson, G. D., Funt, B. V., & Barnard, K. (1995). Color constancy under varying illumination. [Conference Paper]. *Proceedings. Fifth International Conference on Computer Vision (Cat. No.95CB35744)*, 720-725725. doi: 10.1109/iccv.1995.466867

Finlayson, G. D., Hordley, S. D., & HubeL, P. M. (2001). Color by correlation: a simple, unifying framework for color constancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 23*(11), 1209-12211221. doi: 10.1109/34.969113

Forsyth, D. A. (1990). A NOVEL ALGORITHM FOR COLOR CONSTANCY. [Article]. *International Journal of Computer Vision, 5*(1), 5-36. doi: 10.1007/bf00056770

Fukuda, T., Nakagawa, S., Kawauchi, Y., & Buss, M. (1989). Structure decision method for self organising robots based on cell structures-CEBOT. [Conference Paper]. *Proceedings. 1989 IEEE International Conference on Robotics and Automation (Cat. No.89CH2750-8)*, 695-700 vol.2700 vol.2702. doi: 10.1109/robot.1989.100066

Gerkey, B. P., & Mataric, M. J. (2002). Sold!: Auction methods for multirobot coordination. [Article]. *Ieee Transactions on Robotics and Automation, 18*(5), 758-768. doi: 10.1109/tra.2002.803462

Klinker, G. J., Shafer, S. A., & Kanade, T. (1988). Color image analysis with an intrinsic reflection model. [Conference Paper]. *Second International Conference on Computer Vision (IEEE Cat. No.88CH2664-1)*, 292-296296.

Land, E. H., & McCann, J. J. (1971). LIGHTNESS AND RETINEX THEORY. [Article]. *Journal of the Optical Society of America, 61*(1), 1-&. doi: 10.1364/josa.61.000001

Maloney, L. T., & Wandell, B. A. (1986). COLOR CONSTANCY - A METHOD FOR RECOVERING SURFACE SPECTRAL REFLECTANCE. [Article]. *Journal of the Optical Society of America a-Optics Image Science and Vision, 3*(1), 29-33. doi: 10.1364/josaa.3.000029

Martin, D. L., Cheyer, A. J., & Moran, D. B. (1999). The Open Agent Architecture: A framework for building distributed software systems. [Article]. *Applied Artificial Intelligence, 13*(1-2), 91-128. doi: 10.1080/088395199117504

Nardi, D., Adorni, G., Bonarini, A., Chella, A., Clemente, G., Pagello, E., & Piaggio, M. (2000). ART99 - Azzurra Robot Team. In M. Veloso, E. Pagello & H. Kitano (Eds.), *Robocup-99: Robot Soccer World Cup Iii* (Vol. 1856, pp. 695-698). Berlin: Springer-Verlag Berlin.

Novak, C. L., & Shafer, S. A. (1994). METHOD FOR ESTIMATING SCENE PARAMETERS FROM COLOR HISTOGRAMS. [Article]. *Journal of the Optical Society of America a-Optics Image Science and Vision, 11*(11), 3020-3036. doi: 10.1364/josaa.11.003020

Ohta, Y., & Hayashi, Y. (1994). Recovery of illuminant and surface colors from images based on the CIE daylight. [Conference Paper]. *Computer Vision - ECCV '94. Third European Conference on Computer Vision. Proceedings. Vol.II*, 235-246246.

Parker, L. E. (1998). ALLIANCE: An architecture for fault tolerant multirobot cooperation. [Article]. *Ieee Transactions on Robotics and Automation, 14*(2), 220-240. doi: 10.1109/70.681242

RoboCup. (2012). from http://www.robocup.org/

RoboCup. Liga humanoide.

RoboCup. (2011a). Liga de tamaño estandar. from http://www.tzi.de/spl/pub/Website/Downloads/Rules2011.pdf

RoboCup. (2011b). Liga de tamaño pequeño. from http://small-size.informatik.uni-bremen.de/_media/rules:ssl-rules-2011.pdf

Robotics, A. Robots Aldebaran. from http://www.aldebaran-robotics.com/

Sahragard, N., A. R. B. Ramli. (2009). Review on algorithms and techniques for outdoor machine vision. *European Journal of Scientific Research, 34*(1), 125-131.

Sahragard, N., & Ramli, A. R. B. (2009). A Review on Algorithms and Techniques for Outdoor Machine Vision. *European Journal of Scientific Research, 34*(1), 6.

Schechner, Y. Y., Narasimhan, S. G., & Nayar, S. K. (2003). Polarization-based vision through haze. [Article]. *Applied Optics, 42*(3), 511-525. doi: 10.1364/ao.42.000511

Shafer, S. A. (1985). USING COLOR TO SEPARATE REFLECTION COMPONENTS. [Article]. *Color Research and Application, 10*(4), 210-218. doi: 10.1002/col.5080100409

Simonds, J. L. (1963). APPLICATION OF CHARACTERISTIC VECTOR ANALYSIS TO PHOTOGRAPHIC AND OPTICAL RESPONSE DATA. [Article]. *Journal of the Optical Society of America, 53*(8), 968-&. doi: 10.1364/josa.53.000968

Stone, P., & Veloso, M. (1999). Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. [Article; Proceedings

Paper]. *Artificial Intelligence, 110*(2), 241-273. doi: 10.1016/s0004-3702(99)00025-9

Tews, A., & Wyeth, G. (2000). MAPS: a system for multi-agent coordination. [Article]. *Advanced Robotics, 14*(1), 37-50. doi: 10.1163/156855300741429

Weigel, T., Gutmann, J. S., Dietl, M., Kleiner, A., & Nebel, B. (2002). CS Freiburg: Coordinating robots for successful soccer playing. [Article; Proceedings Paper]. *Ieee Transactions on Robotics and Automation, 18*(5), 685-699. doi: 10.1109/tra.2002.804041

Werge, B. B., & Mataric, M. J. (2000). *Broadcast of Local Elibility for Multi-Target Observation*. New York: Springer-Verlag.

Wolff, L. B. (1995). APPLICATIONS OF POLARIZATION CAMERA TECHNOLOGY. [Article]. *Ieee Expert-Intelligent Systems & Their Applications, 10*(5), 30-38. doi: 10.1109/64.464928

Yuille, A. (1987). A method for computing spectral reflectance. *Biol. Cybern., 56*(2-3), 195-201. doi: 10.1007/bf00317994

[i] RoboCup Federation

[ii] Shawn Schaffert

Outdoor Mobile Ground Robot (2005)